User's Manual

RENESAS

CONFIDENTIAL

R-Car Gen4 AUTOSAR R19-11 MCAL

User's Manual

Modules Overview

RTM8RC79FRCMCL5DA0JCDRE, RTM8RC79FRCMCL5QA0JCDRE
RTM8RC79FRCSPL5DA0JCDRE, RTM8RC79FRCSPL5QA0JCDRE
RTM8RC79FRCCOM5DA0JCDRE, RTM8RC79FRCCOM5QA0JCDRE
RTM8RC79FGCMCL5DB0JCDRE, RTM8RC79FGCMCL5QB0JCDRE
RTM8RC79FGCSPL5DB0JCDRE, RTM8RC79FGCSPL5QB0JCDRE
RTM8RC79FGCCOM5DB0JCDRE, RTM8RC79FGCCOM5QB0JCDRE
RTM8RC779GCMCL5DA0JCDRE, RTM8RC779GCMCL5QA0JCDRE
RTM8RC779GCSPL5DA0JCDRE, RTM8RC779GCSPL5QA0JCDRE
RTM8RC779GCCOM5DA0JCDRE, RTM8RC779GCCOM5QA0JCDRE
RTM8RC779GCCDD5DA0JCDRE, RTM8RC779GCCDD5QA0JCDRE

Target Device:
R-Car S4 (Includes S4N)
R-Car V4H

Renesas Electronics
www.renesas.com

Rev.4.00 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
API	Application Programming Interface
ANSI	American National Standards Institute
AUTOSAR	AUTomotive Open System ARchitecture
CAN	Controller Area Network
DEM	Diagnostic Event Manager
DET/Det	Default Error Tracer
DIO	Digital Input Output
ETH	Ethernet
EMM	Error Management Module
FLS	FLaSh Driver
GPT	General Purpose Timer
ICU	Input Capture Unit
IIC	Inter-Integrate Circuit
LIN	Local Interconnect Network
MCAL	MicroController Abstraction Layer
MCU	MicroController Unit
PWM	Pulse Width Modulation
SPI	Serial Peripheral Interface
WDG	WatchDog driver
EcuM	Ecu state Manager
FSA	Functional Safety Assessment
CDD	Complex Device Driver
ICCOM	Inter-CPU Communication
RFSO	Failure Self-Detection Output
IPMMU	IP Memory Management Unit
THS	Thermal Sensor
CRC	Cyclic Redundancy Check
IPL	Initial Program Loader
G4MH	RH850 G4MH MCU core
CR52	ARM Cortex-R52

Definitions

Term	Represented by
Sl. No.	Serial Number

Table of Contents

List of Figures

List of Tables

AUTOSAR Partner’s Agreement

If you want to commercially exploit the AUTOSAR standard in automotive or derived applications, your company has to sign an AUTOSAR Partner's Agreement for getting a royalty-free license to use the AUTOSAR technology.

1.Introduction

This document shall be used as reference by the users for module overview, module dependencies, source code dependencies and configuration parameter dependencies.

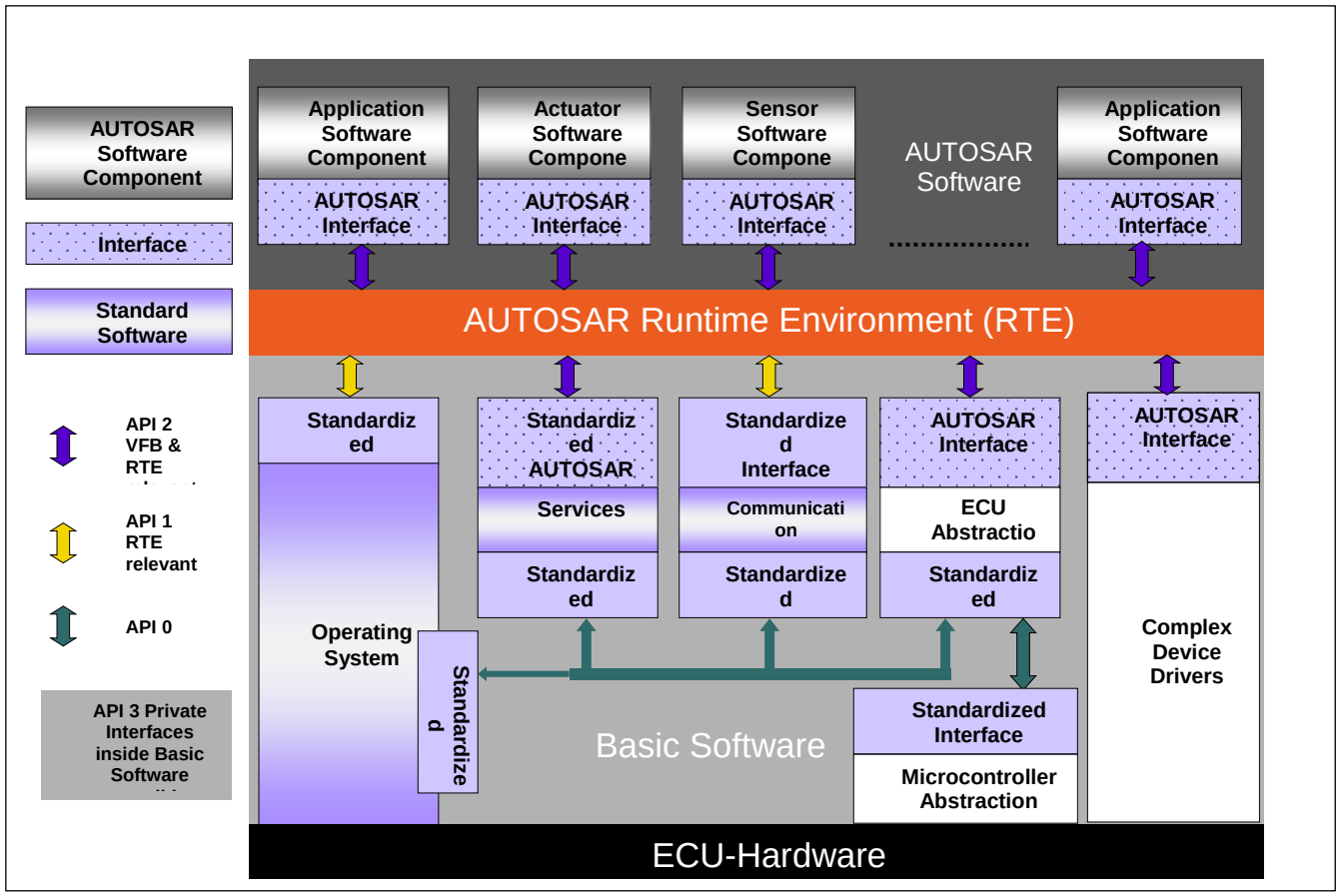

Figure 1-1 System Overview of the AUTOSAR Architecture Layer

1.1 Document Overview

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

Table 1-1 Document Overview

Chapter	Contents
Chapter 1 (Introduction)	Explains the purpose of this document.
Chapter 2 (Reference Documents)	Lists the documents referred for developing this document.
Chapter 3 (AUTOSAR Modules)	Provides the list of modules developed in the MCAL layer, and brief information about the Module overview, Modules dependency, Configuration parameter dependency, source code dependency and stubs.
Chapter 4 (Deviation List)	Lists AUTOSAR Deviation for this MCAL package.
Chapter 5 (Required Operation Conditions)	Required Operation Conditions for R-Car Gen4 product. This information includes Compiler, Configuration code generator, and RENESAS configuration.
Chapter 6 (Issue List)	Provides the issue lists.

2.Reference Documents

Table 2-1 Reference Documents (1/2)

Sl. No.	Title	Version
1.	R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual Getting Started For S4: r11uz0224ej0200-rcars4-mcal-gs.pdf For V4H: r11uz0132ej0201-rcarv4h-mcal-gs.pdf For V4M: r11uz0309ej0201-rcarv4m-mcal-gs.pdf	For S4: 2.00 For V4H: 2.01 For V4M: 2.01
2.	R-Car S4 Series User's Manual: Hardware r19uh0161ej0110-r-cars4.pdf	1.10
3.	R-Car V4H Series User's Manual r19uh0186ej0110-r-carv4h.pdf	1.10
4.	ICUMX IPL for R-Car Gen4 User's Manual r11uz0236ej0156-icumx-ipl.pdf	1.56.0
5.	R-Car Gen4 Flash Writer sample software r11uz0239ej0148-flash-writer.pdf	1.48.0
6.	R-Car V4M Series User's Manual r19uh0196ej0050-r-carv4m.pdf	0.50
7.	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 Docuents (2/2)

Sl. No.	Title	Version
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 ICU Driver AUTOSAR_SWS_ICUDriver.pdf	R19-11
9.	Specification of MCU Driver AUTOSAR_SWS_MCUDriver.pdf	R19-11
10.	Specification of PORT Driver AUTOSAR_SWS_PortDriver.pdf	R19-11
11.	Specification of PWM Driver AUTOSAR_SWS_PWMDriver.pdf	R19-11
12.	Specification of SPI Handler/Driver AUTOSAR_SWS_SPIHandlerDriver.pdf	R19-11
13.	Specification of Watchdog Driver AUTOSAR_SWS_WatchdogDriver.pdf	R19-11
14.	Specification of CAN Driver AUTOSAR_SWS_CANDriver.pdf	R19-11
15.	Specification of LIN Driver AUTOSAR_SWS_LINDriver.pdf	R19-11
16.	Specification of FlexRay Driver AUTOSAR_SWS_FlexRayDriver.pdf	R19-11
17.	Specification of Ethernet Driver AUTOSAR_SWS_EthernetDriver.pdf	R19-11
18.	Complex Driver design and integration guideline AUTOSAR_EXP_CDDDesignAndIntegrationGuideline.pdf	R19-11

3.AUTOSAR Modules

The Micro Controller Abstraction layer is the lowest software layer of the Basic Software. It contains internal drivers, which are software modules with direct access to the µC internal peripherals and memory-mapped µC external devices. Make higher software layers independent of µC.
The modules developed for MCAL layer are as follows:

DIO
FLS
GPT
ICU
MCU
PORT
PWM
SPI
WDG
CAN
LIN
FlexRay
Ethernet

CDD is a specific module located in the Complex Drivers Layer of the Basic Software which interacts with standard BSW modules or Rte. The main goal of the CDD is to implement complex sensor evaluation and actuator control with direct access to the microcontroller using specific interrupts and/or complex microcontroller peripherals, external devices to fulfill the special functional and timing requirements.
The modules developed for CDD are as follows:

ICCOM
IIC
RFSO
CRC
THS
IPMMU
EMM

3.1 Driver Component Makefile

The Makefile provided with the Driver Component consists of the GNU Make compatible script to build the Driver Component in case of any change in the configuration. It can be used in the upper-level Makefile (of the application) to link and build the final application executable.

3.2 Folder Structure

In this section, the folder structure of the Driver Component is explained.

- Top of Folder Structure

```
Install_destination
    ├─── common
    └─── rel
     ├─── common
     ├─── modules
     └─── <Device_Family>
```

Note <Device_Family>: Refer to Table 3-2.

Figure 3-1 Top of Folder Structure

Note:
In Ethernet Driver (S4_CR52), some of the files deviate from the naming conventions described in this document.

Table 3-1 Files that deviate from the naming conventions

Convention	Actual file name
App_<MSN>_<Device_Name>_Sample.c	App_ETH_S4_RSW2_Sample.c
App_<MSN>_Common_Sample.c	App_ETH_Common_RSW2_Sample.c
App_<MSN>_<Device_Name>_<Device_ID>_Sample.arxml	App_ETH_S4_RSW2_Sample.arxml
R1911_<MSN>_<Device_Name>_BSWMDT.arxml	R1911_ETH_S4_RSW2_BSWMDT.arxml
<Msn>_PBcfg.c	Eth_Rsw2_PBcfg.c
<Msn>_Cfg.h	Eth_Rsw2_Cfg.h

- rel/common

This is folder structure for devices which use GHS compiler

```
Install_destination
└── rel
    ├── common
    │   ├── generic
    │   │   ├── compiler
    │   │   │   ├── <AR>
    │   │   │   │   └── ghs
    │   │   │   │       ├── include
    │   │   │   │       │   └── <Msn>_MemMap.h
    │   │   │   │       └── make
    │   │   │   │           └── ghs_rh850_r<AR>_defs.mak
    │   │   │   └── common
    │   │   │       └── ghs
    │   │   │           ├── include
    │   │   │           │   ├── Compiler.h
    │   │   │           │   └── Compiler_Cfg.h
    │   │   │           └── make
    │   │   │               └── ghs_rh850_rules.mak
    │   │   ├── docs
    │   │   │   ├── Getting_Started
    │   │   │   └── Module_Overview
    │   │   ├── generator
    │   │   │   └── Generator Files
    │   │   ├── include
    │   │   │   ├── <AR>
    │   │   │   │   └── Platform_Types.h
    │   │   │   └── common
    │   │   │       └── ghs
    │   │   │           ├── rh850_Types.h
    │   │   │           └── log.h
    │   │   │           └── scif.h
    │   │   ├── make
    │   │   │   ├── <AR>
    │   │   │   │   └── platformtypes_r<AR>_defs.mak
    │   │   │   └── common
    │   │   │       └── platformtypes_common_defs.mak
    │   │   ├── src
    │   │   │   ├── log.c
    │   │   │   └── scif.c
    │   │   └── stubs
    │   │       ├── <AR>
    │   │       │   └──
```

Figure 3-2 rel/common Folder Structure for devices which use GHS compiler

This is folder structure for devices which use ARM compiler

```
Install_destination
└── rel
    ├── common
    │   ├── generic
    │   │   ├── compiler
    │   │   │   ├── <AR>
    │   │   │   │   └── arm
    │   │   │   │       ├── include
    │   │   │   │       │   └── <Msn>_MemMap.h
    │   │   │   │       └── make
    │   │   │   │           └── compiler_rcar_r<AR>_defs.mak
    │   │   │   └── common
    │   │   │       └── arm
    │   │   │           ├── include
    │   │   │           │   ├── cfg
    │   │   │           │   │   └── Compiler_Cfg_dep.h
    │   │   │           │   ├── Compiler.h
    │   │   │           │   └── Compiler_Cfg.h
    │   │   │           └── make
    │   │   │               └── arm_rcar_rules.mak
    │   │   ├── docs
    │   │   │   ├── Getting_Started
    │   │   │   └── Module_Overview
    │   │   ├── generator
    │   │   │   └── Generator Files
    │   │   ├── include
    │   │   │   ├── <AR>
    │   │   │   │   └── Platform_Types.h
    │   │   │   └── common
    │   │   │       └── arm
    │   │   │           ├── arm_cr.h
    │   │   │           ├── arm_cr_cp15.h
    │   │   │           ├── arm_cr_mpu.h
    │   │   │           ├── arm_cr_reg.h
    │   │   │           ├── arm_gic.h
    │   │   │           ├── Interrupt.h
    │   │   │           ├── rh850_Types.h
    │   │   │           └── utils.h
    │   │   │           ├── log.h
    │   │   │           └── scif.h
    │   │   ├── make
    │   │   │   ├── <AR>
    │   │   │   │   └── platformtypes_r<AR>_defs.mak
    │   │   │   └── common
    │   │   │       └── platformtypes_common_defs.mak
    │   │   ├── src
    │   │   │   ├── arm
    │   │   │   │   ├── Interrupt.c
    │   │   │   │   ├── rcar_cfg.c
    │   │   │   │   ├── start.s
    │   │   │   │   └── startup.c
```

```
    │   │   │   ├──── log.c
    │   │   │   └──── scif.c
    │   │   └──── stubs
    │   │       ├──── <AR>
    │   │       │   └────
```

Note <AR>, <Msn>: Refer to Table 3-2

Figure 3-3 rel/common Folder Structure for devices which use ARM compiler

- rel/modules

```
Install_destination
  └── rel
    └──── modules
      └──── <msn>
        ├──── definition
        │   ├──── <AR>
        │   │   └──── <Device>
        │   │     └──── Parameter Definition File
        ├──── generator
        │   ├──── <Device>
        │   │   └──── BSWMDT Files
        ├──── description
        │   ├──── <Device>
        │   │   └──── BSWMDT Files
        ├──── include
        │   └──── Header File
        ├──── make
        │   └──── Make Files
        ├──── Sample Application
        │   ├──── <Device>
        │   │   ├──── <AR>
        │   │   │   ├──── config
        │   │   │   ├──── include
        │   │   │   └──── src
        │   │   ├──── include
        │   │   ├──── make
        │   │   │   └──── <compiler>
        │   │   ├──── obj
        │   │   │   └──── <compiler>
        │   │   └──── src
        │   ├──── include
        │   ├──── make
        │   │   └──── <compiler>
        │   ├──── src
        ├──── src
        │   └──── Source Code File
```

Note <msn>, <AR>, <Device>: Refer to Table 3-2

Figure 3-4 rel/modules Folder Structure

Table 3-2 shows the Variant of Folder and File Name for each Device.

Table 3-2 Variant of Folder and File Name

	Device			
	R-Car S4 (Includes S4N) G4MH	**R-Car S4 (Includes S4N) CR52**	**R-Car V4H**	**R-Car V4M**
<Device_Family>	R-Car	R-Car	R-Car	R-Car
<Device_Name>	S4	S4	V4H	V4M
<Device_ID>	RTM8RC79FG	RTM8RC79FGR	V4H	V4M
<AR>	19_11	19_11	19_11	19_11
<msn>	can dio eth fls fr gpt icu lin mcu port pwm spi wdg cddiccom	can dio eth gpt mcu port cddiccom	can dio eth fls gpt mcu port spi wdg cddiccom cddrfso cddiic cddipmmu cddths cddemm cddcrc	can dio eth fls gpt mcu port spi wdg cddiccom cddrfso cddiic cddipmmu cddths cddemm cddcrc
<compiler>	ghs	arm	arm	arm

Note <MSN> and <Msn>: Same as <msn>. The only difference is uppercase and lowercase.

Table 3-3 shows a file category type which define sample or product.

Table 3-3 File Category Table

File Path	File Name	Description	Category
common	(All files under this path)	Header files defining the structures, macros (Define) etc. defined in the AUTOSAR standard	Sample
rel\common\generic\compiler rel\common\generic\include rel\common\generic\src rel\common\generic\stubs	(All files under this path)	Files defining compilation conditions Sample Application	Sample
rel\common\generic\generator\dlls\<msn>	<msn>RCar.dll	Generator File	Product
rel\common\generic\generator\dlls rel\modules\<msn>\definition\<AR>\<Device>	MCALConfGen.exe, <msn>.cfgxml <AR>_<msn>_<Device>.arxml	Generator File	Product
rel\modules\<msn>\generator\<Device>	<AR>_<msn>_<Device>_BSWMDT.arxml	BSWMDT file	Sample
rel\modules\<msn>\include	*.h	Module source code	Product
rel\modules\<msn>\src	*.c	Module header file	Product
rel\modules\<msn>\sample_application	(All files under this path)	Sample Application	Sample
rel\<Device>\common_family\config\<Device>\<AR>	(All files under this path)	Sample Application	Sample
rel\<Device>\common_family\generator\<Compiler>	<Device>_translation.h Sample_Application_<Device>.trxml	Translation Header file Sample Application	Sample
rel\<Device>\common_family\include	(All files under this path)	Sample Application Register address definition file	Sample
rel\<Device>\common_family\src	(All files under this path)	Sample Application	Sample
common\make rel\common\generic\compiler\<AR>\<Compiler>\make rel\common\generic\compiler\common\<Compiler>\make rel\common\generic\make rel\common\generic\stubs\<AR>\<MSN>\make rel\modules\<msn>\make rel\modules\<msn>\sample_application\make rel\<Device>\common_family\make	(All files under this path)	Make file	Sample

3.2.1 Source Code File

3.2.1.1 Common Source Code File

There are no common source code files.

3.2.1.2 Module Source Code File

Refer to 3.3.x.3 Folder Structure for each module.

Note x: is a number from 1 to 14.

3.2.2 Header File

3.2.2.1 Common Header File

Table 3-4 shows the list of Common Header Files.

Table 3-4 Common Header Files

Location: rel\common\generic	Description
compiler	-
<AR>\	-
<compiler>\	-
Include\	-
<Msn>_MemMap.h	This file allows mapping variables, constants, and code of modules to individual memory sections. Memory mapping can be modified as per ECU's specific needs.
common\	-
<compiler>\	-
Include\	-
Compiler.h	Provides compiler-specific (non-ANSI) keywords. All mappings of keywords, which are not standardized, and/or compiler-specific are placed and organized in this compiler-specific header.
Compiler_Cfg.h	This file contains the memory and pointer classes.
include	-
<AR>\	-
Platform_Types.h	This file provides provision for defining platform and compiler dependent types.
common	-
log.h	This file provides function to perform the console print.
scif.h	This file provides function to perform the SCIF initialization.

Note <AR> and <Msn>: Refer to Table 3-2.

3.2.2.2 Module Header File

Refer to 3.3.x.3 Folder Structure for each module.

Note x: is a number from 1 to 14.

3.2.3 Parameter Definition File

Refer to 3.3.x.3 Folder Structure for each module.

Note x: is a number from 1 to 14.

Refer to 3.4.x.3 Folder Structure for CDD module.

Note x: is a number.

3.2.4 Translation Header File

Table 3-5 shows the list of Translation Header file.

Table 3-5 Translation Header Files

Location	Files
rel\<Device_Name>\common_family\generator	<Device_Name>_translation.h

Note <Device_Family> and <Device_Name>: Refer to Table 3-2.

3.2.5 BSWMDT File

The BSWMDT file target is Sample Application.

When user use BSWMDT file for user application, refer to Msn Driver Component Embedded User's Manual and please modify it as per user application.

Table 3-6 shows the list of BSWMDT file.

Table 3-6 BSWMDT Files

Location: rel\modules\<msn>\generator\<Device_Name>	
<Device_Name>	**Files**
S4, V4H, V4M	R1911_<MSN>_<Device_Name>_BSWMDT.arxml
	R1911_CDD_<MSN>_<Device_Name>_BSWMDT.arxml

Note <msn>, <Device_Name> and <MSN>: Refer to Table 3-2.

3.2.6 Make File

Table 3-7 shows the list of Make file.

Table 3-7 Make Files

Location	**Files**
rel\modules\<msn>\make	renesas_<msn>_check.mak renesas_<msn>_defs.mak renesas_<msn>_rules.mak

Note <msn>: Refer to Table 3-2.

3.2.7 Generator File

Table 3-8 shows the list of Generator file.

Table 3-8 Generator Files

Location: rel\common\generic\generator			**Files**
			<Msn>.cfgxml
			MCALConfGen.exe
	dlls\		-
		<Msn>\	<Msn>RH850.dll <Msn>RCAR.dll

Note <Msn>: Refer to Table 3-2.

3.2.8 Sample Application File

Table 3-9 shows the list of Sample Application file.

Table 3-9 Sample Application Files

Location: rel\modules\<msn>\sample_application\			Files	Remarks
<Device_Name>\			-	-
	<AR>\		-	-
		config\	App_<MSN>_<Device_Name>_<Device_ID>_Sample.arxml	-
		include\	-	(*1)
		src\	-	(*1)
	include\			-
		<compiler>\	App_<MSN>_<Device_Name>_Sample.h	
	make\		-	-
		<compiler>\	-	-
	obj\		-	-
		<compiler>\	-	-
	src\		-	-
		<compiler>\	App_<MSN>_<Device_Name>_Sample.c	-
	stub		dummy_g4mh_case0.srec	(*3)
include\			App_<MSN>_Common_Sample.h	-
			App_<MSN>_Cbk.h	(*2)
make\			-	-
	<compiler>\		App_<MSN>_Common_Sample.mak	-
src\			App_<MSN>_Common_Sample.c	-

Note <Device_Name>, <Device_ID>, <AR>, <MSN, and <msn>: Refer to Table 3-2.

1. The details about the C Source and Header files generated by each module Driver Generation Tool are mentioned in the "<MSN> Driver Component Generation Tool User' Manual".
2. This file is a header file that contains the prototype declaration of the callback function for the sample application. This file exists only for the specific modules.
3. This file exists only for CAN, PORT in device R-Car/S4_CR52. According to *[2] R-Car S4 Series User's Manual: Hardware*, sections "158_44.5.10 PBG, 158_44.5.11 HBG", the access from the Application domain to the Control Domain is protected by default. This dummy binary file unprotects Clock Controller, RS-CAN hardware IP (for CAN) and GP4~7 (for PORT). User can replace the existing one in R-Car/S4_G4MH IPL by this file. The purpose of this file is to support executing sample application. User should self-implementation for a work product.

3.2.9 Stubs File

Table 3-10 and **Table 3-11** show the list of Stub file.

Table 3-10 Stub Files (1/2)

Location: rel/common/generic/<AR>/<stubs>																						
<AR>		<stubs>	Files	Module																		
4_3_1	19_11			CAN	DIO	ETH	FLS	FR	GPT	ICU	LIN	MCU	PORT	PWM	SPI	WDG	ICCOM	RFSO	IIC	THS	EMM	IPMMU	CRC
-	x	CanGeneral	Can_GeneralTypes.h	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	CanIf	CanIf.h	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		CanIf_Cbk.h	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		CanIf_Types.h	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	Dem	Dem.h	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
-	x		Dem_Cfg.h	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
-	x	Det	Det.h	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
-	x	EcuM	EcuM.h	x	-	-	-	-	x	x	x	-	-	x	-	x	x	-	x	-	-	-	-
-	x		EcuM_Cbk.h	x	-	-	-	-	x	x	x	-	-	x	-	x	x	-	x	-	-	-	-
-	x		EcuM_Cfg.h	x	-	-	-	-	x	x	x	-	-	x	-	x	x	-	x	-	-	-	-
-	x		EcuM_Types.h	x	-	-	-	-	x	x	x	-	-	x	-	x	x	-	x	-	-	-	-
-	x		EcuM.c	x	-	-	-	-	x	x	x	-	-	x	-	x	x	-	x	-	-	-	-
-	x	EthGeneral	Eth_GeneralTypes.h	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	EthIf	EthIf.h	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		EthIf_Types.h	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		EthIf_Cbk.c	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		EthIf.c	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	EthTrcv	EthTrcv.h	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		EthTrcv_Types.h	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		EthTrcv_Cbk.c	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 3-11 Stub Files (2/2)

Location: rel/common/generic/<AR>/<stubs>																						
AR		<stubs>	Files	Module																		
4_3_1	19_11			CAN	DIO	ETH	FLS	FR	GPT	ICU	LIN	MCU	PORT	PWM	SPI	WDG	ICCOM	RFSO	IIC	THS	EMM	IPMMU	CRC
-	x	EthSwt	EthSwt_Eth.h	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		EthSwt_Types.h	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		EthSwt_Cbk.c	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	FrGeneral	Fr_GeneralTypes.h	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	FrIf	FrIf.h	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		FrIf.c	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	LinGeneral	Lin_GeneralTypes.h	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-
-	x	LinIf	LinIf_Cbk.h	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-
-	x		LinIf_Cbk.c	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-
-	x	MemIf	MemIf.h	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x		MemIf_Types.h	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	x	Os	Os.h	x	-	x	x	x	x	x	x	x	-	x	x	x	x	-	-	x	x	x	x
-	x		Os.c	x	-	x	x	x	x	x	x	x	-	x	x	x	x	-	-	x	x	-	-
-	x	Rte	Rte.h	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	x	-	x	-	-
-	x		SchM_<Msn>.h (*1)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
-	x		SchM_<Msn>.c (*1)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
-	x		Rte_CDD_Iccom.c	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-
-	x		Rte_CDD_Iccom.h	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-
-	x		Rte_Cdd<Msn>.c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x
-	x		Rte_Cdd<Msn>.h	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x
-	x		Rte_Cdd<Msn>_Types.h	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x
-	x		Rte_Type.h	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
-	x	WdgIf	WdgIf_Types.h	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
-	x	Fee	Fee.h	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

-	x		Fee.c	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

x: applicable

-: not applicable

Note

1. The Msn of “FR” module is “Fr_59_Renesas”

3.3 MCAL Module

3.3.1 DIO Driver Component

3.3.1.1 Module Overview

The DIO Driver component accesses the hardware features directly. The upper layers call the functionalities provided by these components.

The DIO Driver component provides services for:

- Reading from / writing to DIO Channel
- Reading from / writing to DIO Ports
- Reading from / writing to DIO Channel Groups
- Reading module version.
- Flip level of single DIO Channel.
- Writing to DIO Ports with masked value.
- Reading from Port register value

3.3.1.2 Module Dependency

The dependency of DIO Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

PORT
Port pins used by the DIO Driver shall be configured using the PORT module.

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

3.3.1.3 Folder Structure

Table 3-12 and **Table 3-13** show the list of Source Code Files and Header Files for DIO module.

Table 3-12 DIO Header Files

Location: rel\modules\dio\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Dio.h	x	x	x	**x**	Product	This file contains macros, DIO type definitions, structure data types and API function prototypes of DIO Driver.
Dio_LTTypes.h	x	x	x	**x**	Product	This file contains the type definitions of Link time Parameters.
Dio_Version.h	x	x	x	**x**	Product	This file contains macros required to check the versions of modules included by DIO Driver.
Dio_LLDriver.h	x	x	x	**x**	Product	This file contains structure for DIO PFC Register Offset.
Dio_PFC_Driver.h	x	x	x	**x**	Product	This file contains macros, macros required to check the versions of modules included by DIO Driver, global structure for DIO specific device.

x: applicable

-: not applicable

Table 3-13 DIO Source Files

Location: rel\modules\dio\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Dio.c	x	x	x	x	Product	This file contains API implementations of DIO Driver Component.
Dio_Version.c	x	x	x	x	Product	This file contains the code to check the version for the modules included by DIO Driver.
Dio_PFC_Driver.c	x	x	x	x	Product	This file contains API implementations of DIO specific device.

x: applicable

-: not applicable

Table 3-14 shows the list of Parameter Definition Files for DIO module.

Table 3-14 Parameter Definition Files

Location: rel\modules\dio\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	CR52	R1911_DIO_S4_RTM8RC79FR.arxml	RTM8RC79FR
	G4MH	R1911_DIO_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R1911_DIO_V4H.arxml	V4H
	V4M	R1911_DIO_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.1.4 Configuration Parameter Dependency

None

3.3.1.5 Source Code Dependency

The followings are the dependency files commonly used by the DIO Driver module:
Det.h,
SchM_Dio.h,
Rte.h,
Std_Types.h,
Dio_MemMap.h

3.3.1.6 Stubs

Refer to 3.2.9 for the common stubs used for DIO Driver component.

3.3.1.7 Addition Error Handling

Refer to “ Development and Production Errors” section at DIO chapter in Driver Component Embedded User's Manual.

3.3.1.8 Restrictions

None

3.3.1.9 Sample Application

3.3.1.9.1 Sample Application Structure

Refer to 3.7 Sample Application.

In the Sample Application, all the DIO APIs are invoked in the following sequences:

- The API Dio_GetVersionInfo is invoked to get the version of the DIO Driver module with a variable of Std_VersionInfoType. After this API is called, the passed parameter gets updated with the DIO Driver version details.
- The API Dio_WriteChannel is invoked to set the required level of a particular channel. It sets the output level of the channel if the particular channel is output mode. Dio_WriteChannel has no effect, if the channel is input mode.
- The API Dio_ReadChannel reads the actual physical level of the required channel. The level read for the Input channel is actual physical level of that channel, if input buffer for the channel is enabled. The level read for the Output channel is actual physical level of that channel, if bidirectional control for the channel is enabled.
- The API Dio_FlipChannel reads the level of the channel and invert it, then write the inverted level to the channel if the channel is configured as output channel, and the function shall have no influence on the physical output if the channel is configured as input channel.
- The API Dio_WritePort is invoked to simultaneously set the required levels to all channels of a port. It sets the output level of the channels output mode. For the channels input mode, Dio_WritePort has no effect.
- The API Dio_ReadPort reads the actual physical level of all the channels of a required port. The level read for

the Input channel is actual physical level of that channel, if input buffer for the channel is enabled. The level read for the Output channel is actual physical level of that channel, if bidirectional control for the channel is enabled.

- The API Dio_WriteChannelGroup is invoked to simultaneously set the required levels to group of channels of a port. It sets the output level of the channels output mode. For the channels input mode, Dio_WriteChannelGroup has no effect.

- The API Dio_ReadChannelGroup reads the actual physical level of group of channels of a required port. The level read for the Input channel is actual physical level of that channel, if input buffer for the channel is enabled. The level read for the Output channel is actual physical level of that channel, if bidirectional control for the channel is enabled.

- The API Dio_MaskedWritePort provides service to set the value of the given port with the required mask. The Dio_MaskedWritePort function shall set the specified value for the channel in the specified port, if the corresponding bit in mask is ‘1’.

- The API Dio_ReadChannelOutputValue provides service to read and return the value of output data control register of the specified channel.

The API Dio_ReadChannelGroupOutputValue provides service to read and return the value of output data control register of the specified channel group.

.

3.3.1.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-15 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

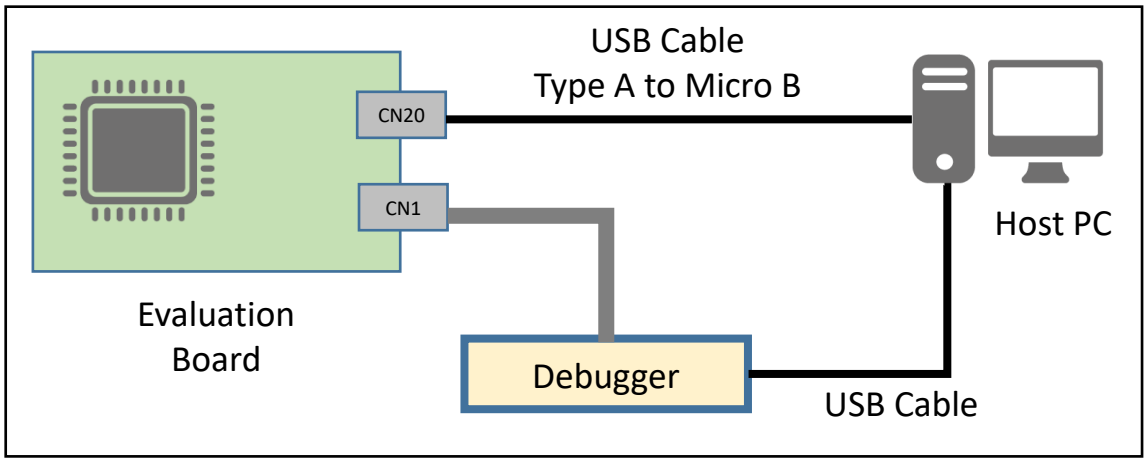

Figure 3-5 S4 Spider board connection of DIO (S4_G4MH) Sample Application

- **S4 CR52 Environment**

Table 3-16 S4 CR52 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

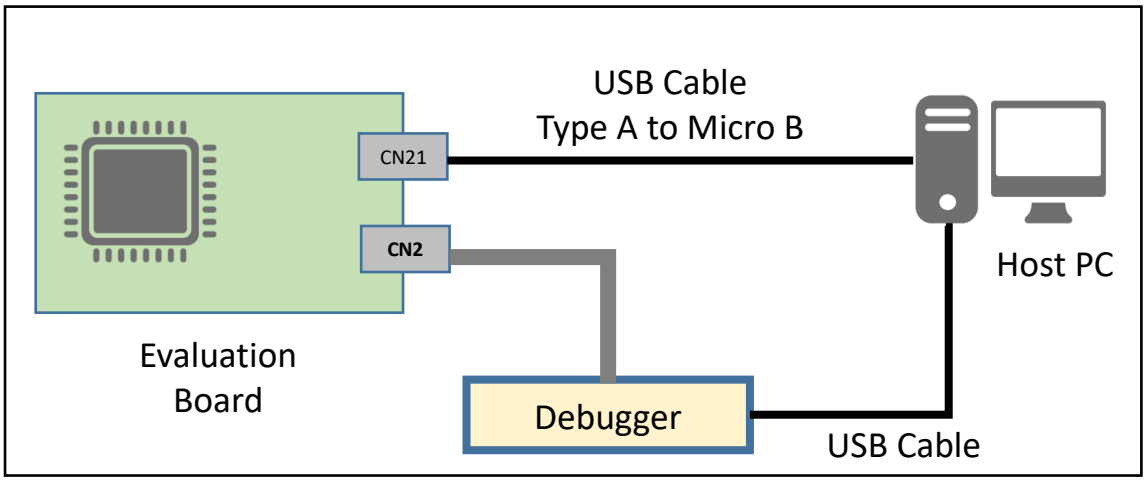

Figure 3-6 S4 Spider board connection of DIO (S4_CR52) Sample ApplicationS4 Environment

- **V4H Environment**

Table 3-17 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

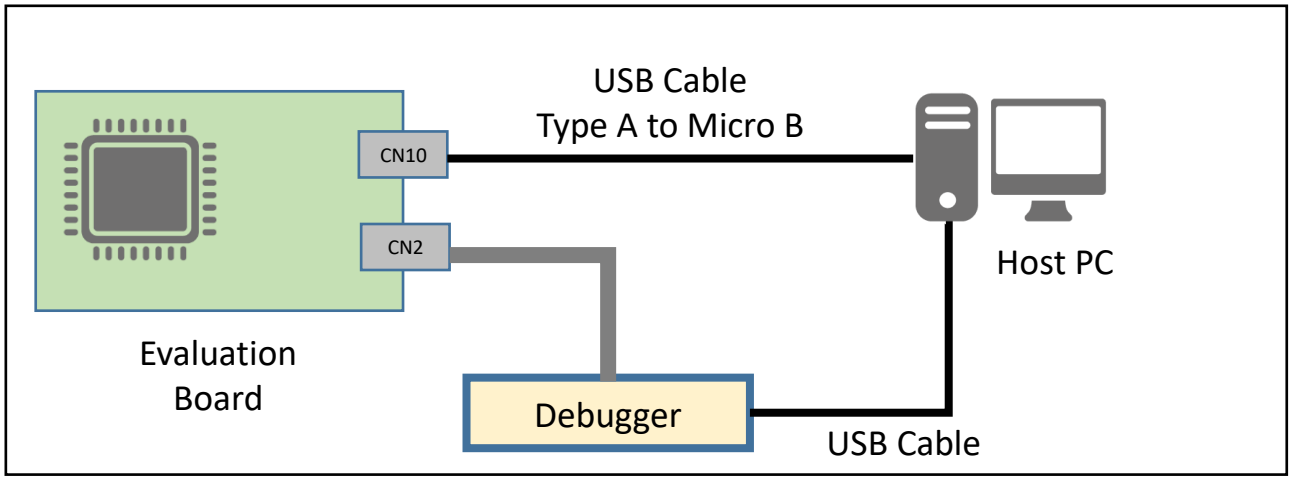

Figure 3-7 V4H White Hawk board connection of DIO (V4H) Sample Application

- **V4M Environment**

Table 3-18 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

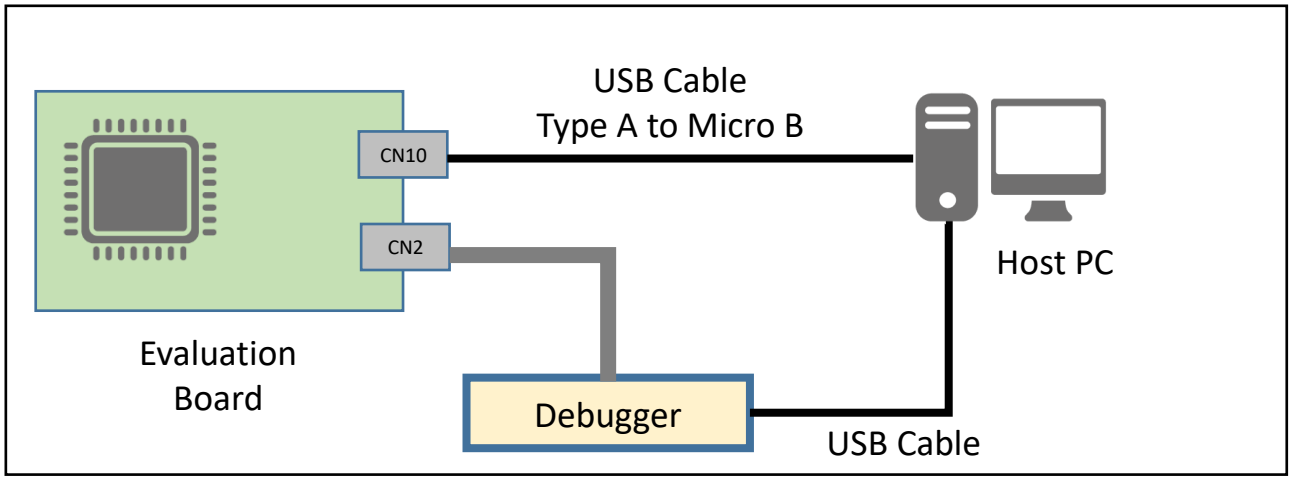

Figure 3-8 V4M Gray Hawk board connection of DIO (V4M) Sample Application

3.3.1.9.3 Preparation

Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.1.9.4 How to build sample application

Refer to 3.7.2.2 How to build the Sample Application

3.3.1.9.5 How to run sample application

Step 1: Set BreakPoint at the end of the main function.

Step 2: Run program to breakpoint, confirm all elements in the array GaaTestResult [0..n] are set to 1 or check print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.3.1.10 ROM/RAM Usage

See Appendix DIO section for information on measuring RAM/ROM consumption.

3.3.1.11 Stack Depth

See Appendix DIO section for information on measuring stack depth.

3.3.1.12 Throughput Details

See Appendix DIO section for information on measuring execution time, and functional testing.

3.3.2 FLS Driver Component

3.3.2.1 Module Overview

The FLS Driver component provides services to read, write, erase, blank check and send specific config to the flash memory.
The FLS Component conforms to the AUTOSAR standard and implements the mapping to the AUTOSAR FLS Driver Specification.
An external flash memory is connected via SPI Multi I/O Bus Controller to microcontroller; the flash driver then uses the handlers / drivers for those busses to access the external flash memory device. The driver for an external flash memory device is located in the ECU Abstraction Layer.

The FLS Driver Component provides services for:

- Initialization
- Erasing the flash memory
- Reading from the flash memory
- Writing to the flash memory
- Setting Operation Mode
- Cancellation of Request
- Reading result and status information
- Blank check for the flash memory
- Module version information
- Job Processing
- Validating the flash memory’s content.
- Suspend the on-going job
- Resume the previous suspended job
- Specific configuration for flash device (both hyper flash and serial flash).
- DDR write pattern
- DDR verify pattern
- DDR calibration

Note:

Since the FLS module also uses QSPI, when software other than MCAL uses QSPI, design it so that problems such as access conflict do not occur.

3.3.2.2 Module Dependency

The dependency of FLS Driver component on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

MEMIF
The Memory Abstraction Interface (MEMIF) module will be used for FLS Status management, FLS Job result type check and FLS Mode setting.

OS
The FLS driver need to calculate the maximum time out for each job processing and hence there is a dependency on the OS, which configured by OS’s Client-Server-Interfaces.

DEM
The Diagnostic Event manager (DEM) will be called whenever FLS module encounters a production relevant error.

FEE
The Flash EEPROM Emulation (FEE) provide declaration and definition of callback functions Fee_JobEndNotification() and Fee_JobErrorNotification() which are used by FLS module.

3.3.2.3 Folder Structure

Table 3-19 to **Table 3-20** show the list of Source Code Files and Header Files for FLS module.

Table 3-19 FLS Header Files (1/2)

Location: rel\modules\fls\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Fls.h	-	x	x	x	Product	This file provides extern declarations for all FLS Driver Component APIs. It also provides the service Ids of APIs, DET Error codes and type definitions for FLS Software initialization structure. This header file shall be included in other modules to use the features of FLS Driver Component.
Fls_Control.h	-	x	x	x	Product	This file contains the declarations of state management and progress control functions.
Fls_LLDriver.h	-	x	x	x	Product	This file contains declarations of FLS control unit specific functions which accessing hardware registers.
Fls_PBTypes.h	-	x	x	x	Product	This file contains the type definitions of post-build parameters. It also contains the macros used by the FLS Driver Component.
Fls_Ram.h	-	x	x	x	Product	This file contains the extern declarations for the global variables defined in Fls_Ram.c file
Fls_RegReadWrite.h	-	x	x	x	Product	This file is to have macro definitions for the functions which process to read or write to registers for FLS operation
Fls_RpcRegValue.h	-	x	x	x	Product	This file is to have macro definitions for the mask value of RPC's register supporting for FLS operation
Fls_Types.h	-	x	x	x	Product	This file contains the common macro definitions and the data types required internally by the FLS software component.
Fls_Version.h	-	x	x	x	Product	This file contains the macros of AUTOSAR version numbers of all modules interfaced to FLS.

x: applicable

-: not applicable

Table 3-20 FLS Source Files

Location: rel\modules\fls\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Fls.c	-	x	x	x	Product	This file contains the implementation of all APIs.
Fls_Control.c	-	x	x	x	Product	This file contains the implementation of state management and progress control functions.
Fls_LLDriver.c	-	x	x	x	Product	This file contains the implementation of FLS control unit specific functions which accessing hardware registers.
Fls_Ram.c	-	x	x	x	Product	This file contains the global variables used by FLS Driver Component.
Fls_Version.c	-	x	x	x	Product	This file contains the code to check the version of all modules interfaced to FLS.

x: applicable

-: not applicable

Table 3-21 shows the list of Parameter Definition Files for FLS module.

Table 3-21 FLS Parameter Definition Files

Location: rel\modules\fls\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19-11	S4	R1911_FLS_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R1911_FLS_V4H.arxml	V4H
	V4M	R1911_FLS_V4M.arxml	V4M
Location: rel\modules\fls\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19-11	S4	R1911_FLS_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R1911_FLS_V4H.arxml	V4H
	V4M	R1911_FLS_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.2.4 Configuration Parameter Dependency

Table 3-22 shows the list of configuration parameter dependency for FLS Driver component.

Table 3-22 FLS Driver Component Configuration Parameter Dependency

Parameter	Module	Path
FlsOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter
FLS_E_GET_CONTROL_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
FLS_E_GET_SEMAPHORE_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
FLS_E_RELEASE_SEMAPHORE_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
FLS_E_WRITE_VERIFY_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
FLS_E_CPG_GET_CONTROL_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
FLS_E_CLOCK_SET_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
FLS_E_CPG_WRITE_VERIFY_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.3.2.5 Source Code Dependency

The followings are the dependency files commonly used by the FLS Driver module:
Det.h,
Dem.h
Std_Types.h,
Fls_MemMap.h,
MemIf.h,
MemIf_Types.h,
SchM_Fls.h,
Rte.h,
Os.h,
Fee.h

3.3.2.6 Stubs

Refer to 3.2.9 for the common stubs used for FLS Driver component.

3.3.2.7 Addition Error Handling

Refer to " Development and Production Errors" section at FLS chapter in Driver Component Embedded User's Manual.

3.3.2.8 Restrictions

None

3.3.2.9 Sample Application

3.3.2.9.1 Sample Application Structure

Refer to 3.7 Sample Application.
In the Sample Application all the FLS APIs are invoked in the following sequence:

- Fls_GetVersionInfo: The API Fls_GetVersionInfo is invoked to get the version of the FLS Driver module with avariable of Std_Version InfoType after the call of this API the past parameter will get updated with the FLS Driver version details.
- Fls_Init: The API Fls_Init is invoked with a valid database address for the proper initialization of the FLS Driver, all the FLS Driver control registers and RAM variables will get initialized after this API is called.

- The API Fls_SetMode() API is invoked to change to FAST MODE.
- The API Fls_SendSpecificConfig() is invoked to perform the setting some specific features for flash device.
- The API Fls_DDRWritePattern() is invoked to write DDR pattern value to DDR pattern area.
- The API Fls_DDRVerifyPattern() is invoked to verify DDR pattern area.
- The API Fls_DDRCalibrate() is invoked to calibrate DDR.
- The API Fls_Erase() is invoked to erase one or more complete Flash Sectors.
- The API Fls_BlankCheck() is invoked to verify the requested area of flash memory is blank (not programmed).
- The API Fls_Write() is invoked to write the one or more complete flash pages to the flash device from the application data buffer
- The API Fls_Read() is invoked to read the requested length of flash memory and stores it in the application data buffer.
- The API Fls_Compare() is invoked to compare the contents of an area of flash memory with that of an application data buffer.
- The API Fls_Suspend() is invoked to suspend on-going read job.
- The API Fls_Resume() is invoked to resumed the suspended read job.
- The API Fls_Cancel() is invoked to cancel an ongoing flash operations like read, write, erase or compare job.
- The API Fls_Getstatus() returns the FLS module state synchronously.
- The API Fls_GetJobResult() returns the result of the last job synchronously.
- The API Fls_Mainfunction() is invoked performs processing of the flash Read, Erase, write, blank check or compare jobs. It’s a scheduled function. The Fls_Mainfunction() accepts only read, write, erase or compare job at a time.

3.3.2.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-23 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

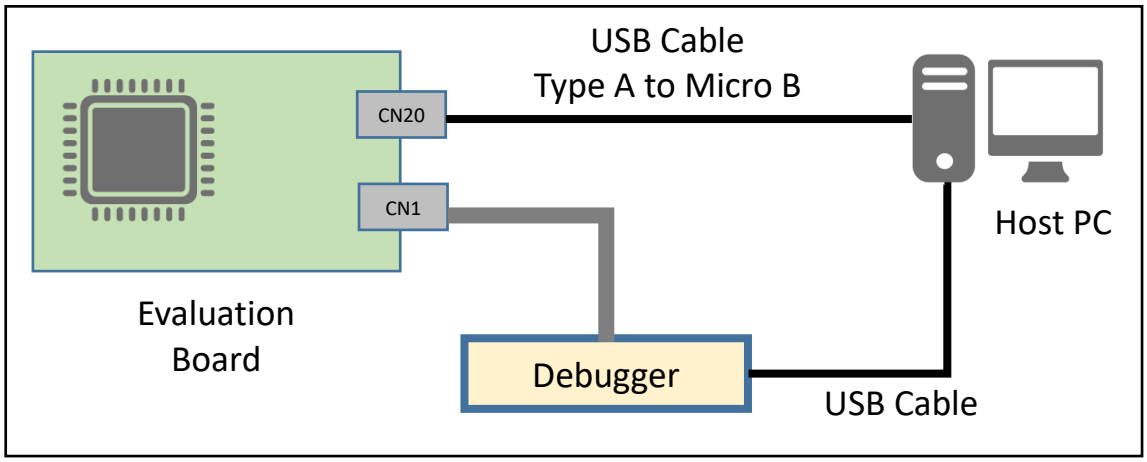

Figure 3-9 S4 Spider board connection of FLS (S4_G4MH) Sample Application

- **V4H Environment**

Table 3-24 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

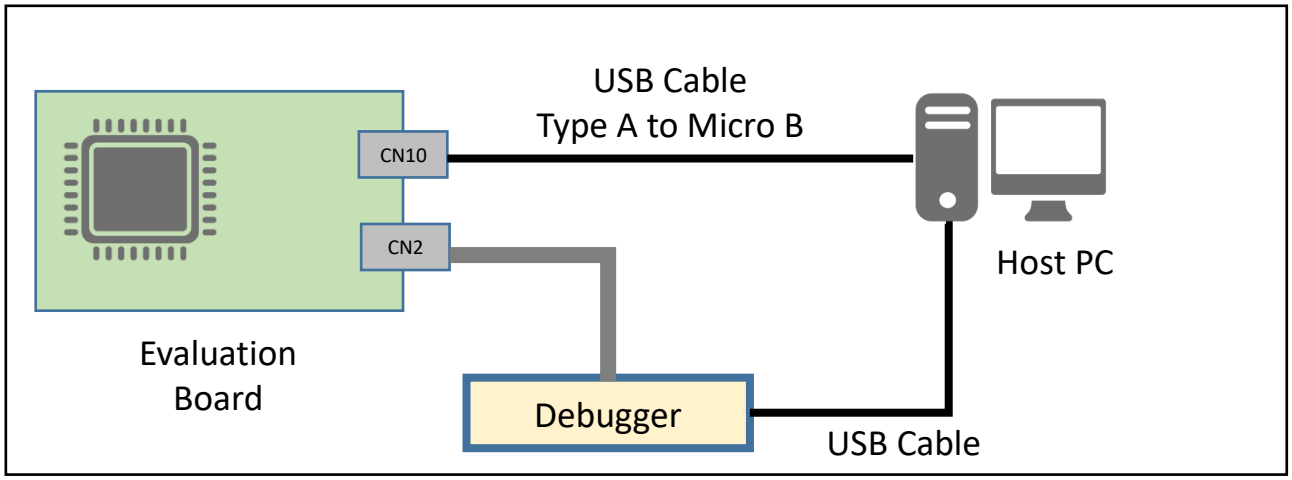

Figure 3-10 V4H White Hawk board connection of FLS (V4H) Sample Application

- **V4M Environment**

Table 3-25 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

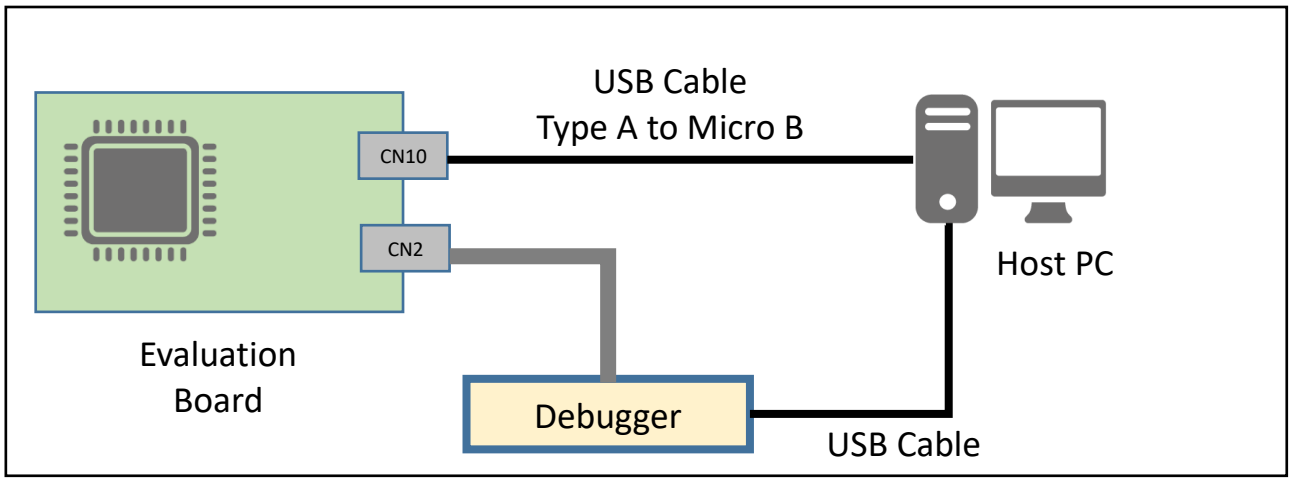

Figure 3-11 V4M Gray Hawk board connection of FLS (V4M) Sample Application

3.3.2.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.2.9.4**How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

3.3.2.9.5**How to run sample application**

Step 1: Set BreakPoint at the end of the main function.

Step 2: Run program to breakpoint, confirm all elements in the array GaaTestResult [0..n] are set to 1 or check print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.3.2.10 **ROM/RAM Usage**

See Appendix FLS section for information on measuring RAM/ROM consumption.

3.3.2.11 **Stack Depth**

See Appendix FLS section for information on measuring stack depth.

3.3.2.12 **Throughput Details**

See Appendix FLS section for information on measuring execution time, and functional testing.

3.3.3 GPT Driver Component

3.3.3.1 Module Overview

The GPT Driver Component provides services for GPT Driver Component initialization, de-initialization, setting timer start and stop, getting elapsed and remaining time, setting GPT mode (one shot, continuous) and disabling or enabling the GPT notification. The GPT Driver Component is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.

The GPT Driver Component is divided into GPT High Level Driver and GPT Low Level Driver to minimize the effort and to optimize the reuse of the software developed on different platforms.

The GPT High Level Driver exports the APIs to the upper modules. All references to the specific microcontroller features and registers are provided in the GPT Low Level Driver.

The GPT channel can be configured as 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 enabled).

Timers OSTM, TAUD, TAUJ and TMU are used in the GPT Driver Component to generate timeout periods.

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

- Initialization: Provides the service to initialize the timer control registers and interrupt registers
- De-Initialization: Provides the service to de-initialize the timer registers and to stop the channels that are running
- Reading of timer values: Provides services to read the elapsed time after the timer is started or the remaining time before the next timeout
- Reading of Predef Timer values: Provides services to read the Predef time
- Start/Stop timer: Provides the service to start/stop the requested timer channel
- Notification services: Provides services for the user to enable or disable the notification for every timeout
- Get version information: Provides the service for the user to read module version
- Wakeup: Controls time-triggered wakeup interrupt, if supported by hardware

3.3.3.2 Module Dependency

The dependency of GPT Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

IO Hardware Abstraction Layer
The GPT Driver depends on the IO Hardware Abstraction Layer, which invokes the APIs and receives the callback notifications. If IO Hardware Abstraction Layer Module is not available, then the required functionality shall be stubbed.

MCU
The GPT Driver component depends on MCU module to set system clock, prescaler(s), and PLL. Thus, any change in the system clock (For example, PLL On -> PLL Off) also affects the clock settings of the GPT hardware. If MCU module is not available, the functionality of system clock, prescaler(s), and PLL setting shall be stubbed.

EcuM
The GPT Driver shall report the wakeup interrupts to the EcuM. If the EcuM is not available, then the required functionality shall be stubbed.

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

OS
As the GPT Driver uses interrupts, it depends on the OS that configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

DEM
The Diagnostic Event manager (DEM) will be called whenever GPT module encounters a production relevant error.

3.3.3.3 Folder Structure

Table 3-26 to **Table 3-28** show the list of Source Code Files and Header Files for GPT module.

Table 3-26 GPT Header Files (1/2)

Location: rel\modules\gpt\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M	-	-
include	-	-	-	-	-	-
Gpt.h	x	x	x	x	Product	This file contains the extern declaration for the GPT Driver Component APIs, macro for service ids, and Det Errors. It also contains Type definitions and the structure declaration for initial configuration.
Gpt_MultiInstance.h	x	x	x	x	Product	This file contains MULTI macro and definitions for multi-instance.
Gpt_PBTypes.h	x	x	x	x	Product	This file contains the data structure definitions for common part of GPT configuration. It also contains the macros used by the GPT Driver Component.
Gpt_Ram.h	x	x	x	x	Product	This file contains the external declaration for the global variables used by GPT Driver Component.
Gpt_Types.h	x	x	x	x	Product	This file contains the common macro definitions and the data types required internally by the GPT software component.
Gpt_Version.h	x	x	x	x	Product	This file contains the definitions of AUTOSAR version numbers of all modules interfaced to GPT.
HWIP	-	-	-	-	-	-
Gpt_TAU_PBTypes.h	-	x	-	-	Product	This file contains the data structure definitions for TAU setting. It also contains the macros used by the GPT Driver Component.
Gpt_TAU_Ram.h	-	x	-	-	Product	This file contains the external declaration for the global variables for TAU used by GPT Driver Component.
ATU	-	-	-	-	-	-
Gpt_ATU_Irq.h	-	-	-	-	Product	This file contains the macro for the ATU channels. It also contains the external declaration for the interrupt functions used by GPT Driver component.
Gpt_ATU_LLDriver.h	-	-	-	-	Product	This file contains the external declaration for the internal functions (Low Level Driver) for ATU called by the GPT Driver APIs.
Gpt_ATU_PBTypes.h	-	-	-	-	Product	This file contains the data structure definitions for ATU setting. It also contains the macros used by the GPT Driver Component.
Gpt_ATU_Ram.h	-	-	-	-	Product	This file contains the external declaration for the global variables for ATU used by GPT Driver Component.

Table 3-27 GPT Header Files (2/2)

Location: rel\modules\gpt\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M	-	-
OSTM	-	-	-	-	-	-
Gpt_OSTM_Irq.h	-	x	-	-	Product	This file contains the macro for the OSTM channels. It also contains the external declaration for the interrupt functions used by GPT Driver component.
Gpt_OSTM_LLDriver.h	-	x	-	-	Product	This file contains the external declaration for the internal functions (Low Level Driver) for OSTM called by the GPT Driver APIs.
Gpt_OSTM_PBTypes.h	-	x	-	-	Product	This file contains the data structure definitions for OSTM setting. It also contains the macros used by the GPT Driver Component.
Gpt_OSTM_Ram.h	-	x	-	-	Product	This file contains the external declaration for the global variables for OSTM used by GPT Driver Component.
TAUD	-	-	-	-	-	-
Gpt_TAUD_Irq.h	-	x	-	-	Product	This file contains the macro for the TAUD channels. It also contains the external declaration for the interrupt functions used by GPT Driver component.
Gpt_TAUD_LLDriver.h	-	x	-	-	Product	This file contains the external declaration for the internal functions (Low Level Driver) for TAUD called by the GPT Driver APIs.
Gpt_TAUD_PBTypes.h	-	x	-	-	Product	This file contains the data structure definitions for TAUD setting. It also contains the macros used by the GPT Driver Component.
Gpt_TAUD_Ram.h	-	x	-	-	Product	This file contains the external declaration for the global variables for TAUD used by GPT Driver Component.
TAUJ	-	-	-	-	-	-
Gpt_TAUJ_Irq.h	-	x	-	-	Product	This file contains the macro for the TAUJ channels. It also contains the external declaration for the interrupt functions used by GPT Driver component.
Gpt_TAUJ_LLDriver.h	-	x	-	-	Product	This file contains the external declaration for the internal functions (Low Level Driver) for TAUJ called by the GPT Driver APIs.
Gpt_TAUJ_PBTypes.h	-	x	-	-	Product	This file contains the data structure definitions for TAUJ setting. It also contains the macros used by the GPT Driver Component.
Gpt_TAUJ_Ram.h	-	x	-	-	Product	This file contains the external declaration for the global variables for TAUJ used by GPT Driver Component.
TMU	-	-	-	-	-	-
Gpt_TMU_Irq.h	x	-	x	x	Product	This file contains the macro for the TMU channels. It also contains the external declaration for the interrupt functions used by GPT Driver component.
Gpt_TMU_LLDriver.h	x	-	x	x	Product	This file contains the external declaration for the internal functions (Low Level Driver) for TMU called by the GPT Driver APIs.
Gpt_TMU_PBTypes.h	x	-	x	x	Product	This file contains the data structure definitions for TMU setting. It also contains the macros used by the GPT Driver Component.
Gpt_TMU_Ram.h	x	-	x	x	Product	This file contains the external declaration for the global variables for TMU used by GPT Driver Component.

x: applicable

-: not applicable

Table 3-28 GPT Source Files

Location: rel\modules\gpt\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Gpt.c	x	x	x	x	Product	This file contains the implementation of GPT Driver Component APIs.
Gpt_Ram.c	x	x	x	x	Product	This file contains declarations of common global variables used by the GPT Driver Component.
Gpt_Version.c	x	x	x	x	Product	This file contains the code to check the version of all modules interfaced to GPT.
HWIP	-	-	-	-	-	-
Gpt_TAU_Ram.c	-	x	-	-	-	This file contains declarations of global variables for TAU used by the GPT Driver Component.
ATU	-	-	-	-	-	-
Gpt_ATU_Irq.c	-	-	-	-	Product	This file contains the implementation of all the ATU interrupt functions used by GPT Driver Component.
Gpt_ATU_LLDriver.c	-	-	-	-	Product	This file contains implementation of all Low-Level Driver functions for ATU invoked by GPT Driver APIs.
Gpt_ATU_Ram.c	-	-	-	-	Product	This file contains declarations of global variables for ATU used by the GPT Driver Component.
OSTM	-	-	-	-	-	-
Gpt_OSTM_Irq.c	-	x	-	-	Product	This file contains the implementation of all the OSTM interrupt functions used by GPT Driver Component.
Gpt_OSTM_LLDriver.c	-	x	-	-	Product	This file contains implementation of all Low-Level Driver functions for OSTM invoked by GPT Driver APIs.
Gpt_OSTM_Ram.c	-	x	-	-	Product	This file contains declarations of global variables for OSTM used by the GPT Driver Component.
TAUD	-	-	-	-	-	-
Gpt_TAUD_Irq.c	-	x	-	-	Product	This file contains the implementation of all the TAUD interrupt functions used by GPT Driver Component.
Gpt_TAUD_LLDriver.c	-	x	-	-	Product	This file contains implementation of all Low-Level Driver functions for TAUD invoked by GPT Driver APIs.
Gpt_TAUD_Ram.c	-	x	-	-	Product	This file contains declarations of global variables for TAUD used by the GPT Driver Component.
TAUJ	-	-	-	-	-	-
Gpt_TAUJ_Irq.c	-	x	-	-	Product	This file contains the implementation of all the TAUJ interrupt functions used by GPT Driver Component.
Gpt_TAUJ_LLDriver.c	-	x	-	-	Product	This file contains implementation of all Low-Level Driver functions for TAUJ invoked by

						GPT Driver APIs.
Gpt_TAUJ_Ram.c	-	x	-	-	Product	This file contains declarations of global variables for TAUJ used by the GPT Driver Component.
TMU	-	-	-	-	-	-
Gpt_TMU_Irq.c	x	-	x	x	Product	This file contains the implementation of all the TMU interrupt functions used by GPT Driver Component.
Gpt_TMU_LLDriver.c	x	-	x	x	Product	This file contains implementation of all Low-Level Driver functions for TMU invoked by GPT Driver APIs.
Gpt_TMU_Ram.c	x	-	x	x	Product	This file contains declarations of global variables for TMU used by the GPT Driver Component.

x: applicable

-: not applicable

Table 3-29 shows the list of Parameter Definition Files for GPT module.

Table 3-29 GPT Parameter Definition Files

Location: rel\modules\gpt\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	CR52	R19_11_GPT_S4_RTM8RC79FR.arxml	RTM8RC79FR
	G4MH	R19_11_GPT_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R19_11_GPT_V4H.arxml	V4H
	V4M	R19_11_GPT_V4M.arxml	V4M
Location: rel\modules\gpt\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	CR52	R19_11_GPT_S4_RTM8RC79FR.arxml	RTM8RC79FR
	G4MH	R19_11_GPT_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R19_11_GPT_V4H.arxml	V4H
	V4M	R19_11_GPT_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.3.4 Configuration Parameter Dependency

Table 3-30 shows the list of configuration parameter dependency for GPT Driver component.

Table 3-30 GPT Driver Component Configuration Parameter Dependency

Parameter	Module	Path
GptAtuClockReference	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig /McuLowSpeedPeriClk
GptOstmClockReference	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig /McuLowSpeedPeriClk
GptTaudClockReference	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig /McuHighSpeedPeriClk
GptTauj01ClockReference	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig /McuHighSpeedPeriClk
GptTauj23ClockReference	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig /McuTAUJClk
GptClockReference	MCU	/AUTOSAR/EcucDefs/Mcu/McuModuleConfiguration/McuClockSettingConfig/ McuClockReferencePoint
GPT_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
GptWakeupSourceRef	ECUM	/AUTOSAR/EcucDefs/EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource
GptTMUClkSrcRef	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig /McuModuleClockSetting/McuSASYNCRTClk /Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig /McuModuleClockSetting/McuSASYNCPERD2Clk
GPT_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.3.3.5 Source Code Dependency

The followings are the dependency files commonly used by the GPT Driver module:
Gpt_MemMap.h,
Std_Types.h,
Det.h,
Rte.h,
EcuM.h,
Os.h,
Dem.h,
rh850_Types.h,
SchM_Gpt.h

3.3.3.6 Stubs

Refer to 3.2.9 for the common stubs used for GPT Driver component.

3.3.3.7 Addition Error Handling

Refer to " Development and Production Errors" section at GPT chapter in Driver Component Embedded User's Manual.

3.3.3.8 Restrictions

None

3.3.3.9 Sample Application

3.3.3.9.1 Sample application structure

- **S4 G4MH:**

Refer to 3.7 Sample Application.

In the Sample Application, all the GPT APIs are invoked in the following sequences:

- The API Gpt_GetVersionInfo is invoked to get the version of the GPT Driver module with a variable of Std_VersionInfo type. After invoking this API, the parameter gets updated with the GPT Driver version details.
- The API Gpt_Init is invoked with a valid database address for the proper initialization of the GPT Driver. All the GPT Driver control registers and RAM variables get initialized after this API is invoked.
- The API Gpt_EnableNotification is invoked to get the notification after the timeout period of timer.
- The API Gpt_StartTimer is invoked to start the particular timer channel.
- The API Gpt_DisableNotification is invoked to disable the notification. After this API is invoked, the notification is not given even if the timeout period occurs for a channel.
- The API Gpt_EnableWakeup is invoked to enables the wakeup notification for particular Channel.
- The API Gpt_SetMode is invoked to change normal mode to sleep mode.
- The API Gpt_SetMode is invoked to change sleep mode to normal mode.
- The API Gpt_DisableWakeup is invoked to disable the wakeup notification for particular Channel.
- Confirm the wakeup functions. Switch the mode to SLEEP and NORMAL, and invoke Enable Wakeup/Disable Wakeup functions, if HW can support the wakeup function.
- The API Gpt_StopTimer is invoked to stop the particular timer channel.
- The API Gpt_DeInit is invoked to de-initialize the GPT.

❖ **S4 CR52/V4H/V4M:**

Refer to 3.7 Sample Application.
In the Sample Application, all the GPT APIs are invoked in the following sequences:

- The API Gpt_GetVersionInfo is invoked to get the version of the GPT Driver module with a variable of Std_VersionInfo type. After invoking this API, the parameter gets updated with the GPT Driver version details.
- The API Gpt_Init is invoked with a valid database address for the proper initialization of the GPT Driver. All the GPT Driver control registers and RAM variables get initialized after this API is invoked.
- The API Gpt_EnableNotification is invoked to get the notification after the timeout period of timer.
- The API Gpt_StartTimer is invoked to start the particular timer channel.
- The API Gpt_GetTimeElapsed is invoked to read time elapsed for a particular channel from the start of channel.
- The API Gpt_GetTimeRemaining is invoked to read the time remaining for the channel to reach timeout.
- The API Gpt_DisableNotification is invoked to disable the notification. After this API is invoked, the notification is not given even if the timeout period occurs for a channel.
- The API Gpt_StartTimer is invoked to start the particular timer channel.
- The API Gpt_StopTimer is invoked to stop the particular timer channel.
- The API Gpt_EnableWakeup is invoked to enables the wakeup notification for particular channel.
- The API Gpt_StartTimer is invoked to start the particular timer channel.
- The API Gpt_SetMode is invoked to change normal mode to sleep mode.
- The API Gpt_CheckWakeup is invoked to wakeup notification for particular channel.
- The API Gpt_DisableWakeup is invoked to disable the wakeup notification for particular channel.
- The API Gpt_StopTimer is invoked to stop the particular timer channel.
- The API Gpt_StartTimer is invoked to start the particular timer channel.
- The API Gpt_StopTimer is invoked to stop the particular timer channel.
- The API Gpt_SetMode is invoked to change sleep mode to normal mode.
- The API Gpt_GetPredefTimerValue is invoked to read the Predef Timer value.
- The API Gpt_DeInit is invoked to de-initialize the GPT.

3.3.3.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-31 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

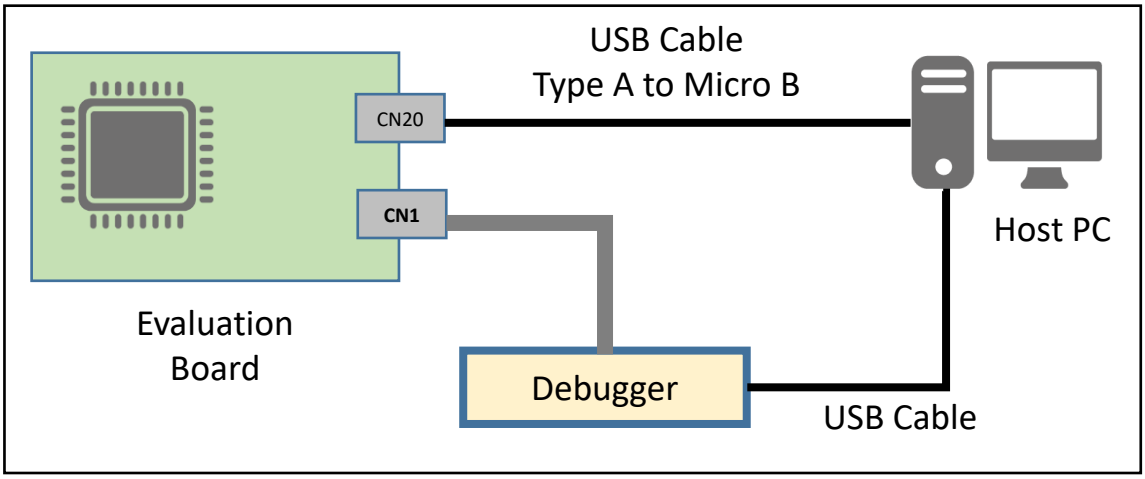

Figure 3-12 S4 Spider board connection of GPT (S4_G4MH) Sample Application

- **S4 CR52 Environment**

Table 3-32 S4 CR52 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

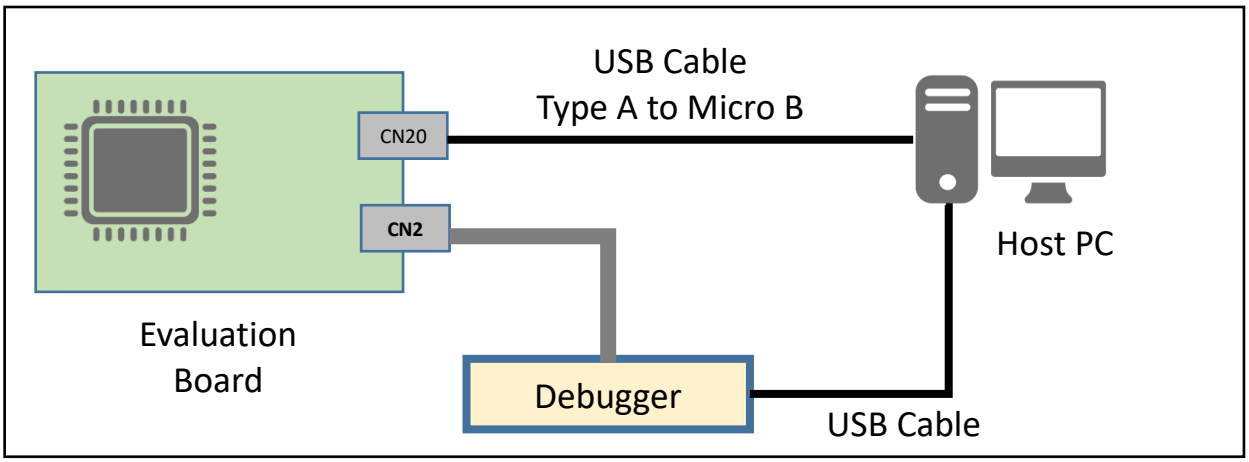

Figure 3-13 S4 Spider board connection of GPT (S4_CR52) Sample Application

- **V4H Environment**

Table 3-33 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

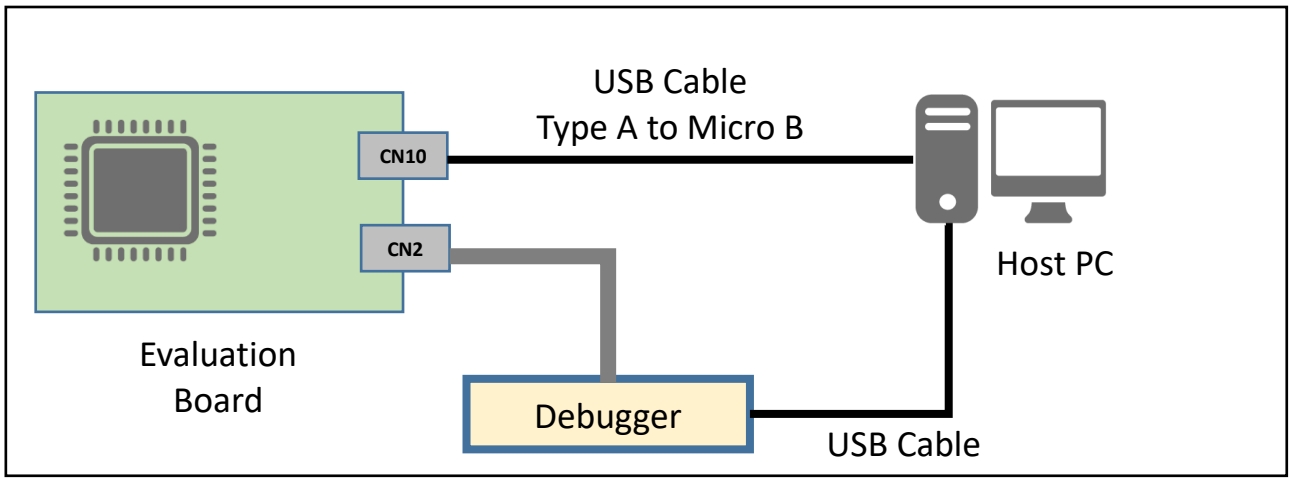

Figure 3-14 V4H White Hawk board connection of GPT (V4H) Sample Application

- **V4M Environment**

Table 3-34 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

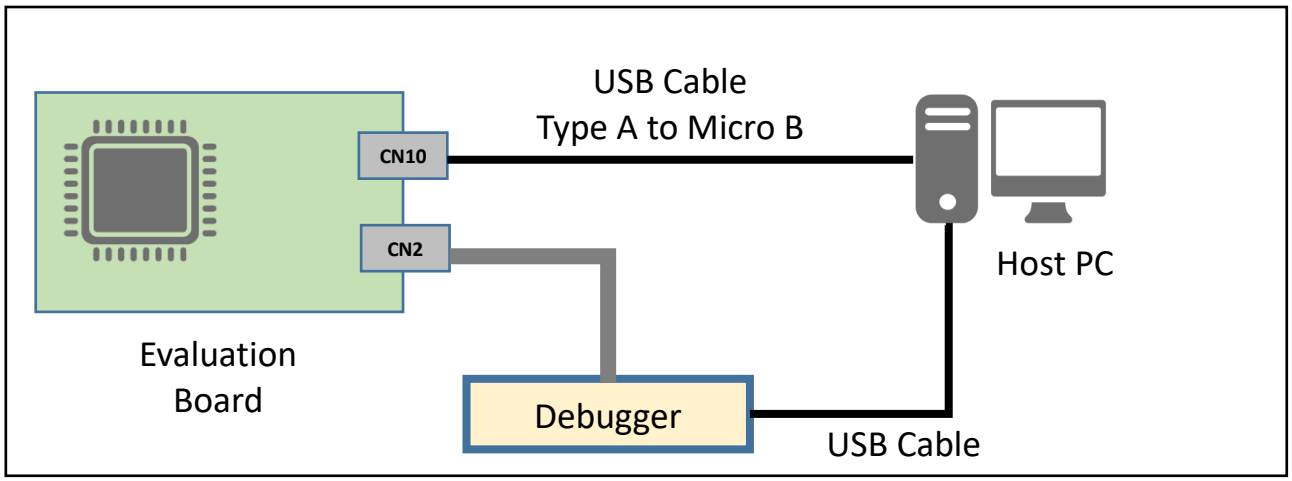

Figure 3-15 V4M Gray Hawk board connection of GPT (V4M) Sample Application

3.3.3.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.3.9.4**How to build Sample Application**

Refer to 3.7.2.2 How to build the Sample Application

3.3.3.9.5**How to run sample application**

- **S4_G4MH**

1. Set breakpoint after call Gpt_GetVersionInfo
2. Run program to breakpoint. Check value VendorID = 100 and ModuleID = 59 and version information is correct
3. Set breakpoint after call Gpt_Init
4. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_INITIALIZED
5. Set breakpoint after block “Wait until Notification occurs for the channel”
6. Run program to breakpoint. Don’t stuck in while loop
7. Set breakpoint after call Gpt_SetMode (GPT_MODE_SLEEP)
8. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_SLEEP
9. Set breakpoint after call Gpt_SetMode (GPT_MODE_NORMAL)
10. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_NORMAL
11. Set breakpoint after call Gpt_DeInit

12. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_UNINITIALIZED
13. Run continue to “Return OK”. Sample application sequence is completed.

- **S4_CR52:**

1. Set breakpoint after call Gpt_GetVersionInfo
2. Run program to breakpoint. Check value GaaTestResult = True
3. Set breakpoint after call Gpt_Init
4. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_INITIALIZED
5. Set breakpoint after call Gpt_StartTimer
6. Run program to breakpoint. Check value Gpt_GpChannelRamData[Channel].ucChannelStatus = GPT_CH_RUNNING
7. Set breakpoint after call Gpt_StopTimer
8. Run program to breakpoint. Check value Gpt_GpChannelRamData[Channel].ucChannelStatus = GPT_CH_STOPPED
9. Set breakpoint after call Gpt_SetMode (GPT_MODE_SLEEP)
10. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_SLEEP
11. Set breakpoint after call Gpt_CheckWakeup
12. Run program to breakpoint. Check Wakeup
13. Set breakpoint after call Gpt_DisableWakeup
14. Run program to breakpoint. Check Disable Wakeup
15. Set breakpoint after call Gpt_SetMode (GPT_MODE_NORMAL)
16. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_NORMAL
17. Set breakpoint after call Gpt_DeInit
18. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_UNINITIALIZED
19. Run continue to end and get into while loop. Sample application sequence is completed.

- **V4H:**

1. Set breakpoint after call Gpt_GetVersionInfo
2. Run program to breakpoint. Check value GaaTestResult = True
3. Set breakpoint after call Gpt_Init
4. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_INITIALIZED
5. Set breakpoint after call Gpt_StartTimer
6. Run program to breakpoint. Check value Gpt_GpChannelRamData[Channel].ucChannelStatus = GPT_CH_RUNNING
7. Set breakpoint after call Gpt_StopTimer
8. Run program to breakpoint. Check value Gpt_GpChannelRamData[Channel].ucChannelStatus = GPT_CH_STOPPED
9. Set breakpoint after call Gpt_SetMode (GPT_MODE_SLEEP)
10. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_SLEEP
11. Set breakpoint after call Gpt_SetMode (GPT_MODE_NORMAL)
12. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_NORMAL
13. Set breakpoint after call Gpt_DeInit
14. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_UNINITIALIZED
15. Run continue to end and get into while loop. Sample application sequence is completed.

- **V4M:**

1. Set breakpoint after call Gpt_GetVersionInfo
2. Run program to breakpoint. Check value GaaTestResult = True

3. Set breakpoint after call Gpt_Init
4. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_INITIALIZED
5. Set breakpoint after call Gpt_StartTimer
6. Run program to breakpoint. Check value Gpt_GpChannelRamData[Channel].ucChannelStatus = GPT_CH_RUNNING
7. Set breakpoint after call Gpt_StopTimer
8. Run program to breakpoint. Check value Gpt_GpChannelRamData[Channel].ucChannelStatus = GPT_CH_STOPPED
9. Set breakpoint after call Gpt_SetMode (GPT_MODE_SLEEP)
10. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_SLEEP
11. Set breakpoint after call Gpt_SetMode (GPT_MODE_NORMAL)
12. Run program to breakpoint. Check vale Gpt_GucDriverMode = GPT_MODE_NORMAL
13. Set breakpoint after call Gpt_DeInit
14. Run program to breakpoint. Check value Gpt_GblDriverStatus = GPT_UNINITIALIZED
15. Run continue to end and get into while loop. Sample application sequence is completed.

Note: The sample application execution and result can be checked by print logs throught the Teraterm if it is used:
“PROGRAM START”: Sample application execution is started.
“EXECUTED OK”: Sample application execution is successful.
“EXECUTED NOT OK”: Sample application execution is failed.
“PROGRAM STOP”: Sample application execution is completed.

3.3.3.10 ROM/RAM Usage

See Appendix GPT section for information on measuring RAM/ROM consumption.

3.3.3.11 Stack Depth

See Appendix GPT section for information on measuring stack depth.

3.3.3.12 Throughput Details

See Appendix GPT section for information on measuring execution time, and functional testing.

3.3.4 ICU Driver Component

3.3.4.1 Module Overview

The ICU Driver Component provides the following services:

- Signal Edge detection and notification
- Services for driver initialization and de-initialization
- Signal time measurement such as period and duty cycle
- Signal Edge time stamping and edge counting
- Support post-build configurations

The ICU Driver Component is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.
Different applications require different number of ICU channels in the different modes. Therefore, the timer, timer operation modes and external interrupts have to be selected depending on the ICU measurement mode. For the R-Car Gen4 microcontroller generation, the following concepts will be considered:

- Using ATU-V Timer A, ATU-V Timer C and TAUD, TAUJ for Edge Counting Measurement mode
- Using ATU-V Timer A, ATU-V Timer C and TAUD, TAUJ for Time Stamping Measurement mode
- Using ATU-V Timer A, ATU-V Timer C and TAUD, TAUJ for Signal Measurement mode
- Using ATU-V Timer A, ATU-V Timer C and TAUD, TAUJ and External Interrupts for Edge Detection mode

The ICU channel can be configured to either a timer channel or an external interrupt based on the required measurement mode. The Edge Detection measurement mode will be configured for the external interrupt channel and Timer channels. The remaining three measurement modes, namely, Edge Counting, Time Stamping and Signal Measurement should be configured only for the timer channels. The configuration of timer in different operating modes will be taken care by the software itself.

The ICU Driver component can be divided into the following sections based on the functions performed by the ICU Driver:

- Initialization
- De-Initialization
- Notification
- Signal Measurement
- Signal Activation and State Information
- Version Information

3.3.4.2 Module Dependency

The dependency of ICU Driver on other modules and the required implementation is briefed as follows:

MCU

The ICU Driver depends on MCU for setting the system clock and PLL. The length of the timer ticks depends on the clock settings made in MCU module. If MCU module is not available, the functionality of system clock and PLL settings shall be stubbed.

OS

As the ICU Driver uses interrupts, it depends on the OS that configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

PORT

The port pins used for the ICU as inputs is configured by the PORT Driver. Hence the PORT Driver has to be initialized prior to the use of ICU functions. If the PORT Driver is not available, then the configuration of port pins used for the ICU shall be stubbed.

To use the external interrupt functionality, port filter of the respective external interrupt needs to be enabled in the PORT component. ICU can override edge detection settings, and so can PORT. The registers ExtIntpcntReg are used by ICU and PORT at the same time and the order of calling APIs is important.

EcuM

The ICU Driver shall report the wakeup interrupts to the EcuM. If the EcuM is not available, then the required functionality shall be stubbed.

DET

In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

IO Hardware Abstraction Layer Module

The ICU Driver depends on the I/O Hardware Abstraction Layer which invokes the APIs and receives the call-back notifications. If I/O Hardware Abstraction Layer Module is not available, then the required functionality shall be stubbed.

RTE

The ICU Driver shall perform data protection using the SchM APIs. If the SchM is not available, then the required functionality shall be stubbed.

DEM

The Diagnostic Event manager (DEM) will be called whenever ICU module encounters a production relevant error.

3.3.4.3 Folder Structure

Table 3-35 to **Table 3-38** shows the list of Source Code Files and Header Files for ICU module.

Table 3-35 ICU Header Files (1/2)

Location: rel\modules\icu\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
include	-	-	-	-	-
Icu_Version.h	-	x	-	Product	This file contains the macros of AUTOSAR version numbers of all modules interfaced to ICU.
Icu_MultiInstance.h	-	x	-	Product	This file contains the extern declaration for the PWM Driver Component APIs and macro for multi-instance. It also contains Type definitions and the structure declaration for init configuration.
Icu_Types.h	-	x	-	Product	This file contains the common macro definitions and the data types required internally by the ICU software component.
Icu_Ram.h	-	x	-	Product	This file contains the extern declarations for the global variables defined in Icu_Ram.c file and the version information of the file.
Icu.h	-	x	-	Product	This file provides extern declarations for all ICU Driver Component APIs. It provides service Ids of APIs, DET Error codes and type definitions for ICU Driver initialization structure. This header file shall be included in other modules to use the features of ICU Driver Component.
Icu_PBTypes.h	-	x	-	Product	This file contains the macros used internally by the ICU Driver Component code and the structure declarations related to timer registers and timers interrupt control registers.
HWIP	-	-	-	-	-
Icu_LLDriver.h	-	x	-	Product	This file contains the definition of the internal functions that access the hardware registers.
ATU	-	-	-	-	-
Icu_ATU_Irq.h	-	-	-	Product	This file contains extern declarations for ISRs and the macro definition used in ISRs.
Icu_ATU_LLDriver.h	-	-	-	Product	This file contains the prototypes of the functions defined in low level file and the version information of the file.
Icu_ATU_PBTypes.h	-	-	-	Product	This file contains the macros used internally by the ICU Driver Component code and the structure declarations related to timer registers and timers interrupt control registers.
Icu_ATU_Ram.h	-	-	-	Product	This file contains the extern declarations for the global variables defined in Icu_Ram.c file and the version information of the file.

Table 3-36 ICU Header Files (2/2)

Location: rel\modules\icu\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
TAU	-	-	-	-	-
Icu_TAU_LLDriver.h	-	x	-	Product	This file contains the prototypes of the functions defined in low level file and the version information of the file in TAU.
Icu_TAU_PBTypes.h	-	x	-	Product	This file contains the macros used internally by the ICU Driver Component code and the structure declarations related to timer registers and timers interrupt control registers in TAU.
Icu_TAU_Ram.h	-	x	-	Product	This file contains the extern declarations for the global variables defined in Icu_TAU_Ram.c file and the version information of the file in TAU.
PIN	-	-	-	-	-
Icu_PIN_Irq.h	-	x	-	Product	This file contains extern declarations for ISRs and the macro definition used in ISRs in TAU PIN.
Icu_PIN_LLDriver.h	-	x	-	Product	This file contains the prototypes of the functions defined in low level file and the version information of the file in TAU PIN.
Icu_PIN_PBTypes.h	-	x	-	Product	This file contains the macros used internally by the ICU Driver Component code and the structure declarations related to timer registers and timers interrupt control registers in TAU PIN.
Icu_PIN_Ram.h	-	x	-	Product	This file contains the extern declarations for the global variables defined in Icu_PIN_Ram.c file and the version information of the file in TAU PIN.
TAUD	-	-	-	-	-
Icu_TAUD_Irq.h	-	x	-	Product	This file contains extern declarations for ISRs and the macro definition used in ISRs in TAUD.
Icu_TAUD_LLDriver.h	-	x	-	Product	This file contains the prototypes of the functions defined in low level file and the version information of the file.
Icu_TAUD_PBTypes.h	-	x	-	Product	This file contains the macros used internally by the ICU Driver Component code and the structure declarations related to timer registers and timers interrupt control registers in TAUD.
Icu_TAUD_Ram.h	-	x	-	Product	This file contains the extern declarations for the global variables defined in Icu_TAUD_Ram.c file and the version information of the file in TAUD.
TAUJ	-	-	-	-	-
Icu_TAUJ_Irq.h	-	x	-	Product	This file contains extern declarations for ISRs and the macro definition used in ISRs in TAUJ.
Icu_TAUJ_LLDriver.h	-	x	-	Product	This file contains the prototypes of the functions defined in low level file and the version information of the file in TAUJ.
Icu_TAUJ_PBTypes.h	-	x	-	Product	This file contains the macros used internally by the ICU Driver Component code and the structure declarations related to timer registers and timers interrupt control registers in TAUJ.
Icu_TAUJ_Ram.h	-	x	-	Product	This file contains the extern declarations for the global variables defined in Icu_TAUJ_Ram.c file and the version information of the file in TAUJ.

x: applicable

-: not applicable

Table 3-37 ICU Source Files (1/2)

Location: rel\modules\icu\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
src	-	-	-	-	-
Icu.c	-	x	-	Product	This file contains the implementation of all APIs.
Icu_Ram.c	-	x	-	Product	This file contains the global variables used by ICU Driver Component.
Icu_Version.c	-	x	-	Product	This file contains the code to check the version of all modules interfaced to ICU.
HWIP	-	-	-	-	-
Icu_LLDriver.c	-	x	-	Product	This file contains the definition of the internal functions that access the hardware registers.
ATU	-	-	-	-	-
Icu_ATU_Irq.c	-	-	-	Product	This file contains the definitions of the ISR functions. The ISR routines are included/excluded using the pre-compile options generated by the tool based on configuration in ATU.
Icu_ATU_LLDriver.c	-	-	-	Product	This file contains the definition of the internal functions that access the hardware registers in ATU.
Icu_ATU_Ram.c	-	-	-	Product	This file contains the global variables used by ICU Driver Component in ATU.
PIN	-	-	-	-	-
Icu_PIN_Irq.c	-	x	-	Product	This file contains the definitions of the ISR functions. The ISR routines are included/excluded using the pre-compile options generated by the tool based on configuration TAU PIN.
Icu_PIN_LLDriver.c	-	x	-	Product	This file contains the definition of the internal functions that access the hardware registers TAU PIN.
Icu_PIN_Ram.c	-	x	-	Product	This file contains the global variables used by ICU Driver Component TAU PIN.
TAU	-	-	-	-	-
Icu_TAU_LLDriver.c	-	x	-	Product	This file contains the definition of the internal functions that access the hardware registers TAU.
Icu_TAU_Ram.c	-	x	-	Product	This file contains the global variables used by ICU Driver Component TAU.
TAUD	-	-	-	-	-
Icu_TAUD_Irq.c	-	x	-	Product	This file contains the definitions of the ISR functions. The ISR routines are included/excluded using the pre-compile options generated by the tool based on configuration TAUD.
Icu_TAUD_LLDriver.c	-	x	-	Product	This file contains the definition of the internal functions that access the hardware registers TAUD.
Icu_TAUD_Ram.c	-	x	-	Product	This file contains the global variables used by ICU Driver Component TAUD.

Table 3-38 ICU Source Files (2/2)

Location: rel\modules\icu\	Supported Device	Category	Description

	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
TAUJ	**-**	**-**	**-**	**-**	**-**
Icu_TAUJ_Irq.c	-	x	-	Product	This file contains the definitions of the ISR functions. The ISR routines are included/excluded using the pre-compile options generated by the tool based on configuration TAUJ.
Icu_TAUJ_LLDriver.c	-	x	-	Product	This file contains the definition of the internal functions that access the hardware registers TAUD.
Icu_TAUJ_Ram.c	-	x	-	Product	This file contains the global variables used by ICU Driver Component TAUD.

x: applicable

-: not applicable

Table 3-39 shows the list of Parameter Definition Files for ICU module.

Table 3-39 ICU Parameter Definition Files

Location: rel\modules\icu\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4_G4MH	R1911_ICU_S4_RTM8RC79FG.arxml	RTM8RC79FG

Note Product Name**:** Product Names supported by “Files”.

3.3.4.4 Configuration Parameter Dependency

Table 3-40 shows the list of configuration parameter dependency for ICU Driver component.

Table 3-40 ICU Driver Component Configuration Parameter Dependency

Parameter	Module	Path
ICU_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
IcuChannelWakeupInfo	ECUM	/AUTOSAR/EcucDefs/EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource

3.3.4.5 Source Code Dependency

The followings are the dependency files commonly used by the ICU Driver module:

Det.h,
Dem.h,
Platform_Types.h,
rh850_Types.h,
Std_Types.h,
Icu_MemMap.h,
SchM_Icu.h,
EcuM.h,
EcuM_Cbk.h,
Compiler.h,
Compiler_Cfg.h,
Rte.h,
Os.h

3.3.4.6 Stubs

Refer to 3.2.9 for the common stubs used for ICU Driver component.

3.3.4.7 Addition Error Handling

Refer to " Development and Production Errors" section at ICU chapter in Driver Component Embedded User's Manual.

3.3.4.8 Restrictions

None

3.3.4.9 Sample Application

3.3.4.9.1 Sample Application Structure

Refer to 3.7 Sample Application.

(1) Single-core Sample Application Structure

In the Sample Application all the ICU APIs are invoked in the following sequences:

- The API Icu_GetVersionInfo is invoked to get the version of the ICU Driver module with a variable of Std_VersionInfoType.
- The API Icu_Init is invoked with a valid database address for the proper initialization of the ICU Driver, and all the ICU Driver control registers and RAM variables get initialized after this API is called.
- The API Icu_SetActivationCondition is invoked to set the passed parameter as the valid edge for timestamping functionality.
- The API Icu_EnableNotification is invoked to get the notification after the timestamp buffer gets updated with the number of timestamps equal to 'NotifyInterval'.
- The API Icu_StartTimestamp is invoked to start the timestamping functionality. The interrupt occurs when the valid edge is detected at input and timestamp buffer gets updated with the timestamps.
- The API Icu_GetTimestampIndex is invoked to returns the value of the timestamp buffer index where the next timestamp is to be written.

- The API Icu_GetTAUInCountValue is invoked to return the TAU timer channel counter value.
- The API Icu_StopTimestamp is invoked to stop the timestamping functionality.
- The API Icu_DisableNotification is invoked to disable the notification. After the API is invoked, the notification is not given even if the timestamp buffer gets updated with the timestamp values equal to 'NotifyInterval'.
- The API Icu_SetActivationCondition is invoked to set the passed parameter as the valid edge for edge counting functionality.
- The API Icu_EnableEdgeCount is invoked to start the counting of the edges at input.
- The API Icu_GetEdgeNumbers is invoked to get the count of the edges occurred at input.
- The API Icu_GetTAUInCountValue is invoked which returns the value of the counter value of TAU timer channel.
- The API Icu_ResetEdgeCount is invoked to reset the count of edges to zero.
- The API Icu_DisableEdgeCount is invoked to stop the counting of the edges at input.
- The API Icu_StartSignalMeasurement is invoked to start measuring the configured signal level at input.
- The API Icu_GetTimeElapsed is invoked to get the configured elapsed time of the signal at input.
- The API Icu_StopSignalMeasurement is invoked to stop measuring the configured signal level at input.
- The API Icu_GetDutyCycleValues is invoked to get the Active time and Period time of the signal at input. The user has to connect the signal to the configured channel as well as the channel next to it.
- The API Icu_GetTAUInCountValue is invoked which returns the value of the counter value of TAU timer channel.
- The API Icu_GetInputState is invoked to get the status at input of a channel.
- The API Icu_EnableEdgeDetection is invoked for a channel configured for edge detection mode to enable the edge detection.
- The API Icu_EnableNotification is invoked for a channel configured for edge detection mode to enable the notification.
- The API Icu_DisableEdgeDetection is invoked for a channel configured for edge detection mode to disable the edge detection.
- The API Icu_DisableNotification is invoked for a channel configured for edge detection mode to disable the notification.
- The API Icu_SetMode is invoked to set the operation mode to normal operation or reduced power operation depending on the passed parameter.
- The API Icu_DisableWakeup is invoked to disable the wakeup capability of the channel.
- The API Icu_EnableWakeup is invoked to enable the wakeup capability of the channel.
- The API Icu_CheckWakeup is invoked to check the wakeup capability of the channel.
- The API Icu_GetInputLevel is invoked to check the pin level status of the channel.
- The API Icu_DeInit is invoked to de-initialize the ICU module.

(2) Multi-core Sample Application Structure (Not supported)

The multi-core / multi-instantiation sample application provides how to use the ICU driver for multi-core device. This sample application provide instance 0 and 1. Instance 0 assumes running PE0 and Instance 1 assume running PE1.

This sample application shows that the different channel of the timer can be used in different core and different mode at the same time.
The configuration is shown as following.
[PE0] – [Instance 0]
Instance 0 uses following channels.
TAUJ sub block 0 Channel 0 (Signal Measurement mode - Duty)
TAUJ sub block 0 Channel 1 (Signal Measurement mode - Extra Channel)
TAUJ sub block 0 Channel 2 (Signal Measurement mode - High Time)
TAUJ sub block 0 Channel 3 (Signal Measurement mode - Low Time)
TAUJ sub block 1 Channel 0 (Signal Measurement mode - Period Time)
TAUD sub block 1 Channel 3 (Time Stamping Measurement mode)

[PE1] – [Instance 1]
Instance 1 uses following channels.
TAUD sub block 1 Channel 5(Edge Counting Measurement mode)
TAUD sub block 1 Channel 7(Edge detection mode)
INTP Channel 0 (Edge detection mode)
[Behavior of multi-core / multi-instantiation sample application]
This sample application assumes that the same signal is input in all configured channels.
ICU driver of instance 1 decides the start and end timing of measurement by using Edge detection mode channel and Edge counting Measurement mode channel.
ICU driver of instance 0 measures the timestamps and length using start and end timing from PE 1.

- The API Icu_GetVersionInfo is invoked to get the version of the ICU Driver module with a variable of Std_VersionInfoType.
- The API Icu_Init is invoked with a valid database address for the proper initialization of the ICU Driver in PE0. All the ICU Driver control registers and RAM variables get initialized after this API is called.
- The API Icu_StartSignalMeasurement is invoked to start measuring the configured signal level at input. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_GetTimeElapsed is invoked to get the configured elapsed time of the signal at input. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_StopSignalMeasurement is invoked to stop measuring the configured signal level at input. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_GetDutyCycleValues is invoked to get the Active time and Period time of the signal at input. The user has to connect the signal to the configured channel as well as the adjacent channel. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_SetActivationCondition is invoked to set the passed parameter as the valid edge for timestamping functionality. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_SetActivationCondition is invoked to set the passed parameter as the valid edge for edge counting functionality. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_EnableNotification is invoked to get the notification after the timestamp buffer gets updated with the number of timestamps equal to ‘NotifyInterval’.
- The API Icu_StartTimestamp is invoked to start the timestamping functionality. The interrupt occurs when the valid edge is detected at input, and timestamp buffer gets updated with the timestamps in PE0.
- The API Icu_GetTimestampIndex is invoked to return the value of the timestamp buffer index where the next timestamp will be written in PE0.
- The API Icu_GetTAUInCountValue is invoked to return the TAU timer channel counter value.
- The API Icu_StopTimestamp is invoked to stop the timestamping functionality in PE0.
- The API Icu_DisableNotification is invoked to disable the notification. After this API is invoked, the notification is not given even if the timestamp buffer gets updated with the timestamp values equal to ‘NotifyInterval’.
- The API Icu_EnableNotification is invoked to get the notification after the timestamp buffer gets updated with the number of timestamps equal to ‘NotifyInterval’in PE1.
- The API Icu_EnableEdgeCount is invoked to start the edge counting at input in PE1.
- The API Icu_GetEdgeNumbers is invoked to get the edge counting occurred at input in PE1.
- The API Icu_GetTAUInCountValue is invoked to return the TAU timer channel counter value.
- The API Icu_ResetEdgeCount is invoked to reset the count of edges to zero in PE1.
- The API Icu_DisableEdgeCount is invoked to stop the edge counting at input in PE1.
- The API Icu_GetTAUInCountValue is invoked to return the TAU timer channel counter value.
- The API Icu_GetInputState is invoked to get the status at input of a channel in PE1.
- The API Icu_EnableEdgeDetection is invoked for a channel configured for edge detection mode to enable the edge detection in PE1.
- The API Icu_EnableNotification is invoked for a channel configured for edge detection mode to enable the notification.

- The API Icu_DisableEdgeDetection is invoked for a channel configured for edge detection mode to disable the edge detection in PE1.
- The API Icu_DisableNotification is invoked for a channel configured for edge detection mode to disable the notification.
- The API Icu_SetMode is invoked to set the operation mode to normal operation or reduced power operation depending on the passed parameter.
- The API Icu_DisableWakeup is invoked to disable the wakeup capability of the channel.
- The API Icu_EnableWakeup is invoked to enable the wakeup capability of the channel.
- The API Icu_CheckWakeup is invoked to check the wakeup capability of the channel.
- The API Icu_GetInputLevel is invoked to check the pin level status of the channel.
- The API Icu_DeInit is invoked to de-initialize the ICU module.

❖ **S4_G4MH**

The multi-core / multi-instantiation sample application provides how to use the ICU driver for multi-core device. This sample application provide instance 0 and 1.Instance 0 assumes running PE0 and Instance 1 assume running PE1.

This sample application shows that the different channel of the timer can be used in different core and different mode at the same time.
The configuration is shown as following.
[PE0] – [Instance 0]
Instance 0 uses following channels.
TAUJ sub block 0 Channel 0 (Signal Measurement mode - Duty)
TAUJ sub block 0 Channel 1 (Signal Measurement mode - Extra Channel)
TAUJ sub block 0 Channel 2 (Signal Measurement mode - High Time)
TAUJ sub block 0 Channel 3 (Signal Measurement mode - Low Time)
TAUJ sub block 1 Channel 0 (Signal Measurement mode - Period Time)
TAUD sub block 1 Channel 3 (Time Stamping Measurement mode)

[PE1] – [Instance 1]
Instance 1 uses following channels.
TAUD sub block 1 Channel 5(Edge Counting Measurement mode)
TAUD sub block 1 Channel 7(Edge detection mode)
INTP Channel 0 (Edge detection mode)
[Behavior of multi-core / multi-instantiation sample application]
This sample application assumes that the same signal is input in all configured channels.
ICU driver of instance 1 decides the start and end timing of measurement by using Edge detection mode channel and Edge counting Measurement mode channel.
ICU driver of instance 0 measures the timestamps and length using start and end timing from PE 1.

- The API Icu_GetVersionInfo is invoked to get the version of the ICU Driver module with a variable of Std_VersionInfoType.
- The API Icu_Init is invoked with a valid database address for the proper initialization of the ICU Driver in PE0. All the ICU Driver control registers and RAM variables get initialized after this API is called.
- The API Icu_StartSignalMeasurement is invoked to start measuring the configured signal level at input. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_GetTimeElapsed is invoked to get the configured elapsed time of the signal at input. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_StopSignalMeasurement is invoked to stop measuring the configured signal level at input. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_GetDutyCycleValues is invoked to get the Active time and Period time of the signal at input. The user has to connect the signal to the configured channel as well as the adjacent channel. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_SetActivationCondition is invoked to set the passed parameter as the valid edge for timestamping functionality. This API is invoked sequentially by using inter core exclusive control in each PE.

- The API Icu_SetActivationCondition is invoked to set the passed parameter as the valid edge for edge counting functionality. This API is invoked sequentially by using inter core exclusive control in each PE.
- The API Icu_EnableNotification is invoked to get the notification after the timestamp buffer gets updated with the number of timestamps equal to 'NotifyInterval'.
- The API Icu_StartTimestamp is invoked to start the timestamping functionality. The interrupt occurs when the valid edge is detected at input, and timestamp buffer gets updated with the timestamps in PE0.
- The API Icu_GetTimestampIndex is invoked to return the value of the timestamp buffer index where the next timestamp will be written in PE0.
- The API Icu_GetTAUInCountValue is invoked to return the TAU timer channel counter value.
- The API Icu_StopTimestamp is invoked to stop the timestamping functionality in PE0.
- The API Icu_DisableNotification is invoked to disable the notification. After this API is invoked, the notification is not given even if the timestamp buffer gets updated with the timestamp values equal to 'NotifyInterval'.
- The API Icu_EnableNotification is invoked to get the notification after the timestamp buffer gets updated with the number of timestamps equal to 'NotifyInterval'in PE1.
- The API Icu_EnableEdgeCount is invoked to start the edge counting at input in PE1.
- The API Icu_GetEdgeNumbers is invoked to get the edge counting occurred at input in PE1.
- The API Icu_GetTAUInCountValue is invoked to return the TAU timer channel counter value.
- The API Icu_ResetEdgeCount is invoked to reset the count of edges to zero in PE1.
- The API Icu_DisableEdgeCount is invoked to stop the edge counting at input in PE1.
- The API Icu_GetTAUInCountValue is invoked to return the TAU timer channel counter value.
- The API Icu_GetInputState is invoked to get the status at input of a channel in PE1.
- The API Icu_EnableEdgeDetection is invoked for a channel configured for edge detection mode to enable the edge detection in PE1.
- The API Icu_EnableNotification is invoked for a channel configured for edge detection mode to enable the notification.
- The API Icu_DisableEdgeDetection is invoked for a channel configured for edge detection mode to disable the edge detection in PE1.
- The API Icu_DisableNotification is invoked for a channel configured for edge detection mode to disable the notification.
- The API Icu_SetMode is invoked to set the operation mode to normal operation or reduced power operation depending on the passed parameter.
- The API Icu_DisableWakeup is invoked to disable the wakeup capability of the channel.
- The API Icu_EnableWakeup is invoked to enable the wakeup capability of the channel.
- The API Icu_CheckWakeup is invoked to check the wakeup capability of the channel.
- The API Icu_GetInputLevel is invoked to check the pin level status of the channel.
- The API Icu_DeInit is invoked to de-initialize the ICU module.

Refer to 3.7 Sample Application.
In the Sample Application the ICU APIs are invoked in the following sequence:

This sample application shows that the different channel of timer can be used in difference core and different mode at the same time.
The configuration is shown as following.
[PE0] – [Instance 0]
Instance 0 uses the following channels.
- ATU5C sub block 1 Channel 0 (Signal Measurement mode)
- ATU5C sub block 1 Channel 1 (Signal Measurement mode)
- ATU5C sub block 3 Channel 0 (Signal Measurement mode)
- ATU5A sub block 0 Channel 0 (Time Stamping Measurement mode)

[PE1] – [Instance 1]
Instance 1 uses the following channels.

- ATU5A sub block 0 Channel 1 (Edge Counting Measurement mode)
- ATU5C sub block 0 Channel 0 (Edge detection mode)

[Behavior of multi-core / multi-instantiation sample application]
This sample application assumes that the same signal is input in all configured channels.
ICU driver of instance 1 decides the start and end timing of measurement by using Edge detection mode channel and Edge counting Measurement mode channel.
ICU driver of instance 0 measures timestamps and length using start and end timing from PE 1.

3.3.4.9.2**Recommended Environment**

❖ **S4_G4MH**

Table 3-41 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger + Pulse Generator
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

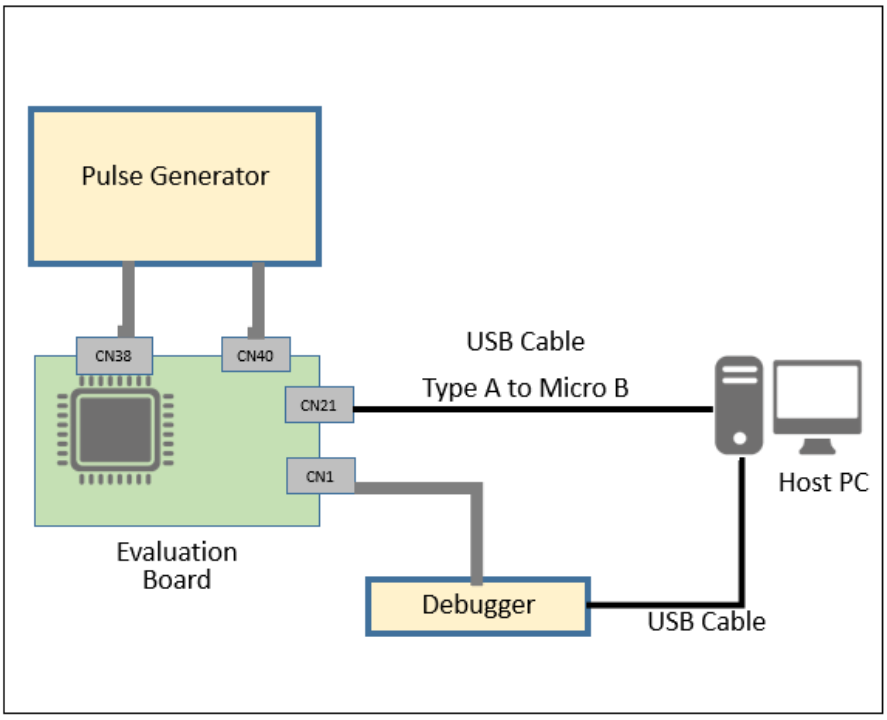

Figure 3-16 S4 Spider board connection of ICU Sample Application

3.3.4.9.3**Preparation**

- Prepare the environment for the corresponding device at the above **(**2) Recommended Environtment.
- For V4H, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.4.9.4**How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:
  external\rel\S4\common_family\make\ghs
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat Icu R19-11 S4 No

3.3.4.9.5 How to run Sample Application

Sample application shall include the following sequence for generic usage in ICU driver

1. Prepare a pulse generator to generate square pulses for the pins that have been configured according to the CDF file
2. Set the break point after the ICU version check condition.
3. Run program to breakpoint. If GddVersionInfo returns correct version information, the global variable GblTestResult will be ICU_PASS.

For Edge Counting functionality:

4. To read counted edges value after invoke Icu_EnableEdgeCount, assign the global variable GblEdgeNumberCh to Icu_GetEdgeNumbers(). Set the break point after the condition that checks the return value assigned to the local variable ret. If the number of counted edges of given channel meets the condition of the stub function Icu_Edge_Count_Test() and returns 0 for ret variable (the return value of the ret variable will depend on the value of the global variable GblEdgeNumberCh compared in the above stub function). The global variable GblBreakLoop will be ICU_TRUE(=1).
5. Check the do while loop for GblBreakLoop.

➢ If condition in do while loop is while (1), the loop continues to be executed.
➢ If condition in do while loop is while (0), exit the loop and assign global variable GblBreakLoop to ICU_FALSE(=0) to continue using this variable for the next checkpoints.

6. Similarly, repeat step 4 and step 5 after calling Icu_ResetEdgeCount and Icu_DisableEdgeCount, check the value of the global variable GblBreakLoop and confirm the checkpoint is done successfully if it is equal to ICU_TRUE.

For Edge Detect functionality:

7. To confirm detected edges after invoke Icu_EnableNotification, set the break point after the condition that checks the return value assigned to the local variable ret. If the number of detected edges of given channel meets the condition of the stub function Icu_Edge_Detect_Test() and returns 0 for ret variable. The global variable GblBreakLoop will be ICU_TRUE(=1).
8. Check the do while loop for GblBreakLoop.

➢ If condition in do while loop is while (1), the loop continues to be executed.
➢ If condition in do while loop is while (0), exit the loop and assign global variable GblBreakLoop to ICU_FALSE(=0) to continue using this variable for the next checkpoints.

9. Similarly, repeat step 7 and step 8 after calling Icu_DisableEdgeDetection, check the value of the global variable GblBreakLoop and confirm the checkpoint is done successfully if it is equal to ICU_TRUE.
10. To read the input state of given channel, set the breakpoint after calling Icu_GetInputState. The expected values are: GblInputStatus[0]==ICU_ACTIVE, GblInputStatus[1]==ICU_IDLE since Icu_GetInputState is called 2 times in a row.
11. Continue to run program until “End of main() function” sample_end() is reached. Sample application sequence is completed.

Note: The sample application execution and result can be checked by print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.3.4.10 ROM/RAM Usage

See Appendix ICU section for information on measuring RAM/ROM consumption.

3.3.4.11 Stack Depth

See Appendix ICU section for information on measuring stack depth.

3.3.4.12 Throughput Details

See Appendix ICU section for information on measuring execution time, and functional testing.

3.3.5 MCU Driver Component

3.3.5.1 Module Overview

The MCU Driver accesses the hardware features directly. The upper layers call the functionalities provided by the Driver. MCU component has functionalities related to PLL Initialization, Clock Initialization & Distribution, RAM sections, Initializations, MCU Reduced Power Modes Activation and MCU Reset Activation & Reason.

The MCU Driver component is divided into the following sub modules based on the functionality required:

- Initialization
- Clock Initialization
- PLL Clock Distribution
- RAM sections Initialization and Status Verification
- MCU Reset Reason
- MCU Reduced Power Modes Activated
- Version Information

3.3.5.2 Module Dependency

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

EcuM
The reference for the type of reset will be provided by the MCU Driver to the ECU State manager module.

OS
As the MCU Driver uses interrupts, it is dependent on the OS which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

SchM
The Basic Software Scheduler module will be called whenever a critical section protection function is called.

3.3.5.3 **Folder Structure**

Table 3-42 to **Table 3-44** show the list of Source Code Files and Header Files for MCU module.

Table 3-42 MCU Header Files (1/2)

Location: rel\modules\mcu\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Mcu.h	x	x	x	x	Product	Contains macros, MCU type definitions, structure data types and API function prototypes.
Mcu_Ram.h	x	x	x	x	Product	Contains Provision of API information
Mcu_Types.h	x	x	x	x	Product	Contains typedef used by MCU driver.
Mcu_PBTypes.h	x	x	x	x	Product	Contains the type definitions of Post Build time Parameters.
Mcu_Version.h	x	x	x	x	Product	This file contains the macros of AUTOSAR version numbers of all modules interfaced to MCU Driver Component.
CLK	-	-	-	-	-	-
Mcu_CLK_LLDriver.h	x	x	x	x	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Mcu_CLK_LLDriver.c
ECM	-	-	-	-	-	-
Mcu_ECM_Irq.h Mcu_ECM_LLDriver.h	-	x	-	-	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Mcu_ECM_Irq.h and Mcu_ECM_LLDriver.c
RAM	-	-	-	-	-	-
Mcu_RAM_LLDriver.h	x	x	x	x	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Mcu_RAM_LLDriver.c
VMN	-	-	-	-	-	-
Mcu_VMN_LLDriver.h	-	x	-	-	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Mcu_ VMN_LLDriver.c.
RST	-	-	-	-	-	-
Mcu_RST_LLDriver.h	x	x	x	x	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Mcu_ RST_LLDriver.c.
CLM	-	-	-	-	-	-
Mcu_CLM_LLDriver.h	-	x	-	-	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Mcu_ CLM_LLDriver.c.
STB	-	-	-	-	-	-

Mcu_STB_LLDriver.h	x	x	x	x	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Mcu_ STB_LLDriver.c.

Table 3-43 MCU Header Files (2/2)

Location: rel\modules\mcu\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Mcu_Ram.h	x	x	x	x	Product	Contains global variable declarations.
Mcu_Version.h	x	x	x	x	Product	Contains software version information.
Mcu_PBTypes.h	x	x	x	x	Product	Contains the type definitions of Post Build time Parameters.
Mcu_Cbk.h	-	x	-	-	Product	Contains Notification declarations.
Mcu_Cfg.h	x	x	x	x	Product	Contain the configuration parameters declared as #define.
ECC	-	-	-	-	-	-
Mcu_ECC_LLDriver.h	-	x	-	-	Product	Contains extern declaration of low level Functions called by APIs, macro definitions, and type definitions used in Mcu_ ECC_LLDriver.c
CPG	-	-	-	-	-	-
Mcu_CPG_LLDriver.h	x	x	x	x	Product	Contains extern declaration of low level Functions called by APIs, macro definitions, and type definitions used in Mcu_ CPG_LLDriver.c

x: applicable

-: not applicable

Table 3-44 MCU Source Files

Location: rel\modules\mcu\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Mcu.c	x	x	x	x	Product	Contains AUTOSAR APIs and Renesas APIs
Mcu_PBcfg.c	x	x	x	x	Product	Contains configuration parameters categorized Post build configuration.
Mcu_Cfg.c	x	x	x	x	Product	Contains configuration parameters categorized Pre-compile configuration.
Mcu_Irq.c	x	x	x	x	Product	Contain functions that handle interrupts
Mcu_Ram.c	x	x	x	x	Product	Contains global variables.
Mcu_Version.c	x	x	x	x	Product	Contains software version information.
CPG	-	-	-	-	-	-
Mcu_CPG_LLDriver.c	x	x	x	x	Product	Contains Low Level Driver function for Clock Pulse Generator.
RAM	-	-	-	-	-	-
Mcu_RAM_LLDriver.c	x	x	x	x	Product	Contains Low Level Driver function for RAM for both domains.
CLK	-	-	-	-	-	-
Mcu_CLK_LLDriver.c	x	x	x	x	Product	Contains Low Level Driver function for Clock Controller.
RST	-	-	-	-	-	-
Mcu_RST_LLDriver.c	x	x	x	x	Product	Contains Low Level Driver function for Reset controllers.
ECM	-	-	-	-	-	-
Mcu_ECM_Irq.c Mcu_ECM_LLDriver.c	-	x	-	-	Product	Contains Low Level Driver function for Error Control Module.
VMN	-	-	-	-	-	-
Mcu_VMN_LLDriver.c	-	x	-	-	Product	Contains Low Level Driver function for Voltage monitor and Delay monitor.
CLM	-	-	-	-	-	-
Mcu_CLM_LLDriver.c	-	x	-	-	Product	Contains Low Level Driver function for Clock Monitor.
STB	-	-	-	-	-	-
Mcu_STB_LLDriver.c	x	x	x	x	Product	Contains Low Level Driver function for Standby.
ECC	-	-	-	-	-	-
Mcu_ECC_LLDriver.c	-	x	-	-	Product	Contains Low Level Driver functions for MCU Driver

x: applicable

-: not applicable

Table 3-41 shows the list of Parameter Definition Files for MCU module.

Table 3-45 MCU Parameter Definition Files

Location: rel\modules\mcu\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4_CR52	R19_11_MCU_S4_RTM8RC79FR.arxml	RTM8RC79FR
	S4_G4MH	R19_11_MCU_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R19_11_MCU_V4H.arxml	V4H
	V4M	R19_11_MCU_V4M.arxml	V4M
Location: rel\modules\mcu\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4_CR52	R19_11_MCU_S4_RTM8RC79FR.arxml	RTM8RC79FR
	S4_G4MH	R19_11_MCU_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R19_11_MCU_V4H.arxml	V4H
	V4M	R19_11_MCU_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.5.4 Configuration Parameter Dependency

Table 3-46 shows the list of configuration parameter dependency for MCU Driver component.

Table 3-46 MCU Driver Component Configuration Parameter Dependency

Parameter	Module	Path
MCU_E_CLOCK_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
MCU_E_VMON_DIAG_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_001
MCU_E_DMON_DIAG_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_002
MCU_E_ECM_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_003
MCU_E_ECM_INIT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_004

3.3.5.5 Source Code Dependency

The followings are the dependency files commonly used by the MCU Driver module:

- Det.h,
- Dem.h,
- rh850_Types.h,
- Std_Types.h,
- Mcu_MemMap.h,
- SchM_Mcu.h,
- Rte.h,
- Os.h

3.3.5.6 Stubs

Refer to 3.2.9 for the common stubs used for MCU Driver component.

3.3.5.7 Addition Error Handling

Refer to " Development and Production Errors" section at MCU chapter in Driver Component Embedded User's Manual.

3.3.5.8 Restrictions

None

3.3.5.9 Sample Application

The Sample Application is provided as reference to the user to understand the method in which the MCU APIs can be invoked from the application.

3.3.5.9.1 Sample Application Structure

Refer to 3.7 Sample Application.

In the Sample Application, all the MCU APIs are invoked in the following sequences:

- The API Mcu_GetVersionInfo is invoked to get the version of the MCU Driver module.
- The API Mcu_Init is invoked with a valid database address for the proper initialization of the MCU Driver, and all the MCU Driver control registers and RAM variables get initialized after this API is called.
- The API Mcu_InitClock is invoked to initialize the clock sources. Also, the clock domains configured will also be initialized by this API to the respective clock sources.
- The API Mcu_InitRamSection is invoked to initialize the RAM section wise provided by the configuration structure. For V4H, to execute API Mcu_InitRamSection, macro RAM_INITIALIZATION should be defined (#define RAM_INITIALIZATION)
- The API Mcu_GetResetReason and Mcu_GetResetRawValue are invoked to read the reset reason from hardware.
- The API Mcu_PerformReset is invoked to perform the software reset.

3.3.5.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-47 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

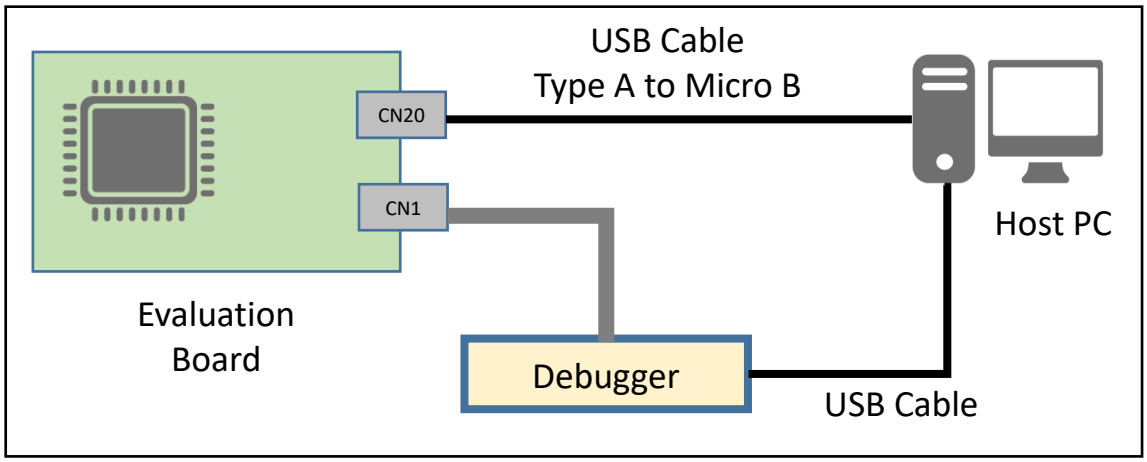

Figure 3-17 S4 Spider board connection of MCU (S4_G4MH) Sample Application

- **S4 CR52 Environment**

Table 3-48 S4 CR52 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

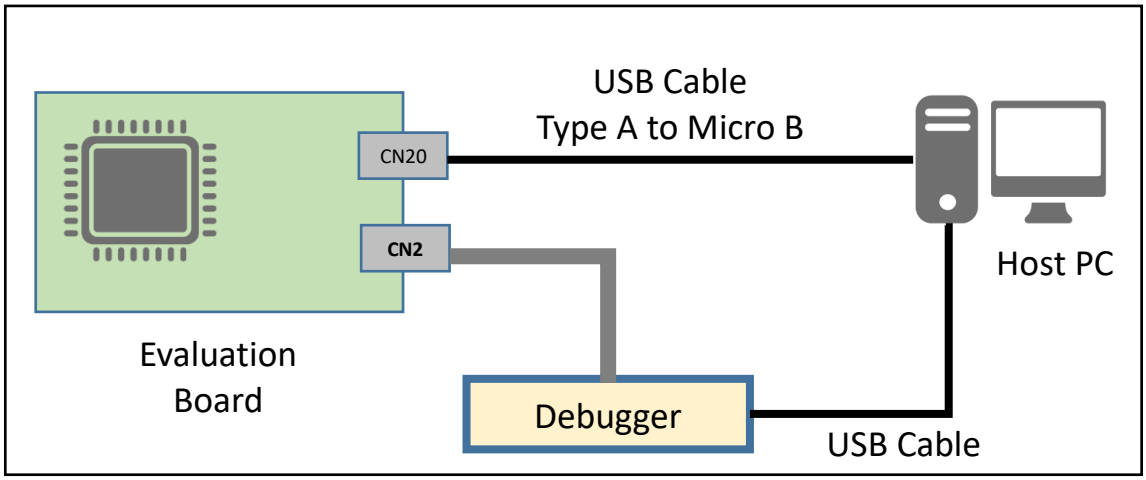

Figure 3-18 S4 Spider board connection of MCU (S4_CR52) Sample Application

- **V4H Environment**

Table 3-49 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

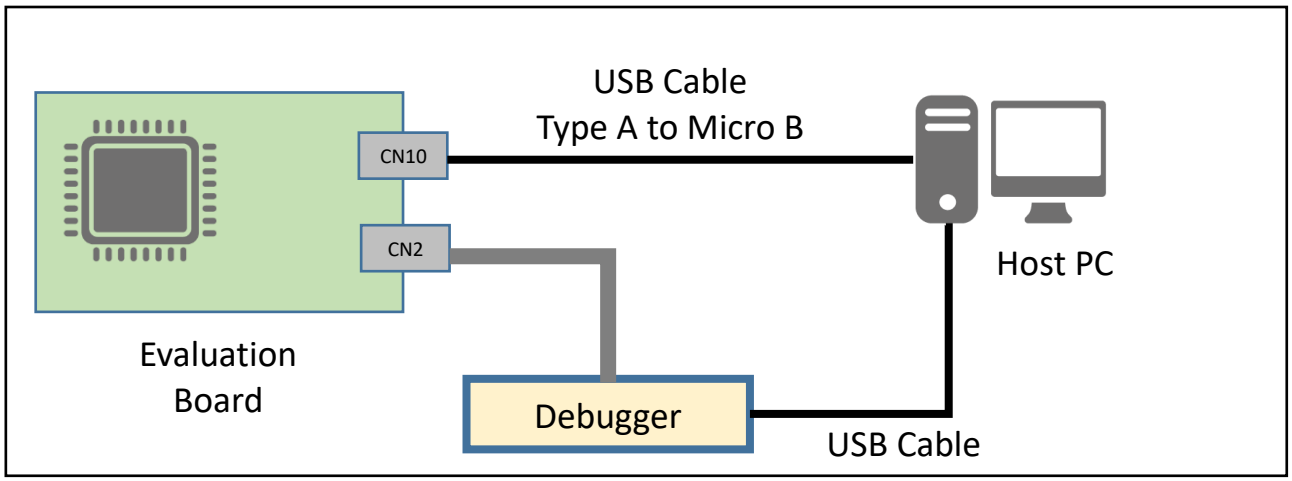

Figure 3-19 V4H White Hawk board connection of MCU (V4H) Sample Application

- **V4M Environment**

Table 3-50 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

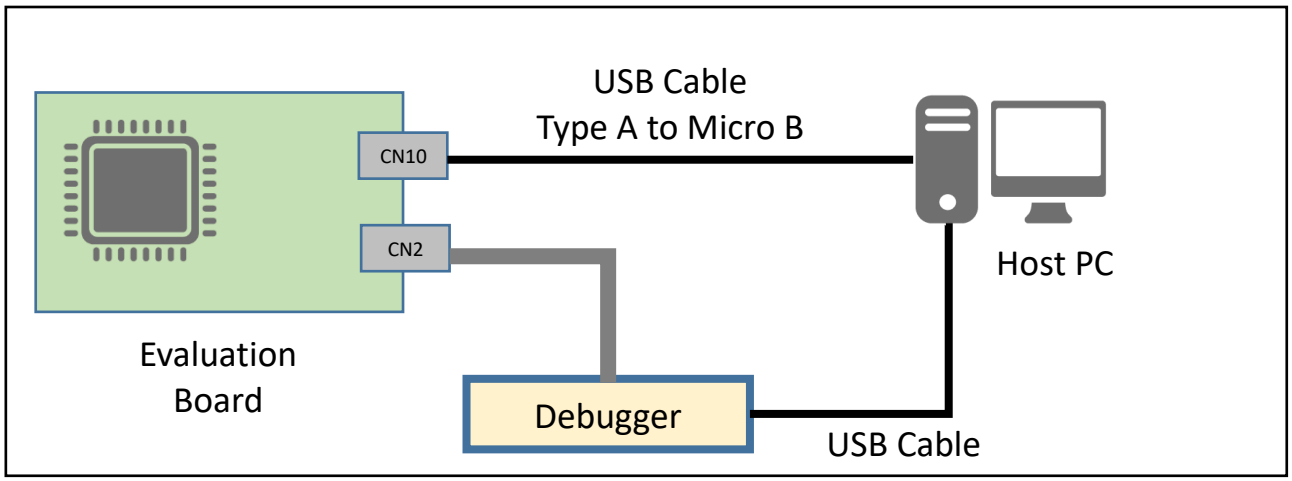

Figure 3-20 V4M Gray Hawk board connection of MCU (V4M) Sample Application

3.3.5.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.5.9.4**How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

[R-Car S4_G4MH]

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\S4\common_family\make\ghs
- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat MCU R19-11 S4 No

[R-Car S4_CR52]

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\S4\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat MCU R19-11 S4 No

[R-Car V4H]

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\V4H\common_family\make\arm

- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat MCU R19-11 V4H No

[R-Car V4M]

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\V4M\common_family\make\arm

- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat MCU R19-11 V4M No

3.3.5.9.5 How to run sample application

Step 1: Set BreakPoint at each function which needs to check and then run step over that BreakPoint or set BreakPoint at the end of the main function:

Step 2: Run program to breakpoint, confirm all elements in the array GaaTestResult [0..n] are set to 1 or check print log throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.3.5.10 ROM/RAM Usage

See Appendix MCU section for information on measuring RAM/ROM consumption.

3.3.5.11 Stack Depth

See Appendix MCU section for information on measuring stack depth.

3.3.5.12 Throughput Details

See Appendix MCU section for information on measuring execution time, and functional testing.

3.3.6 PORT Driver Component

3.3.6.1 Module Overview

The PORT Driver component accesses the hardware features directly. The upper layers call the functionalities provided by these components.

The PORT Driver component provides services for:

- Initialization of every port pins to the configured functionality.
- Changing the port pin direction during run time.
- Reading module version
- Changing the port pin direction during runtime to default setting.

3.3.6.2 Module Dependency

The dependency of PORT Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

3.3.6.3 **Folder Structure**

Table 3-51 to **Table 3-52** show the list of Source Code Files and Header Files for PORT module.

Table 3-51 PORT Header Files

Location: rel\modules\port	**Supported Device**				**Category**	**Description**
	S4 (Includes S4N) CR52	**S4 (Includes S4N) G4MH**	**V4H**	**V4M**		
include	-	-	-	-	-	-
Port.h	x	x	x	x	Product	This file contains macros, API function prototypes, and global configure data of PORT Driver. Also PORT type definitions and structure data types is included through Port_Type.h
Port_PBTypes.h	x	x	x	x	Product	This file contains the type definitions of Post-Build time Parameters.
Port_Ram.h	x	x	x	x	Product	This file contains the extern declarations of global RAM variables of PORT Driver.
Port_Types.h	x	x	x	x	Product	This file defines global data type.
Port_Version.h	x	x	x	x	Product	This file contains macros required to check the release versions of modules included by PORT Driver.
PFC	-	-	-	-	-	-
Port_PFC_LLDriver.h	x	x	x	x	Product	Contains extern declaration of low level Functions called by APIs, macro definations, and type definations used in Port_PFC_LLDriver.c

x: applicable

-: not applicable

Table 3-52 PORT Source Files

Location: rel\modules\port\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Port.c	x	x	x	x	Product	This file contains API implementations of Port Driver Component.
Port_Ram.c	x	x	x	x	Product	This file contains defination for Global RAM variables in Port Driver.
Port_Version.c	x	x	x	x	Product	This file contains code to check the version for modules included by PORT Driver.
PFC	-	-	-	-	-	-
Port_PFC_LLDriver.c	x	x	x	x	Product	Low level Driver code of the PORT Driver Component.

x: applicable

-: not applicable

Table 3-53 shows the list of Parameter Definition Files for PORT module.

Table 3-53 PORT Parameter Definition Files

Location: rel\modules\port\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4_CR52	R19_11_PORT_S4_RTM8RC79FR.arxml	RTM8RC79FR
	S4_G4MH	R19_11_PORT_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R19_11_PORT_V4H.arxml	V4H
	V4M	R19_11_PORT_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.6.4 Configuration Parameter Dependency

The below tables show the list of configuration parameter dependency for PORT Driver component.

- **S4 G4MH:**

Table 3-54 PORT Driver Component Configuration Parameter Dependency

Parameter	Module	Path
PORT_E_GET_CONTROL_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

- **S4 CR52:**

Table 3-55 PORT Driver Component Configuration Parameter Dependency

Parameter	Module	Path
PORT_E_GET_CONTROL_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
PORT_E_FUSE_MONITORING_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

- **V4H:**

Table 3-56 PORT Driver Component Configuration Parameter Dependency

Parameter	Module	Path
PORT_E_GET_CONTROL_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
PORT_E_FUSE_MONITORING_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_001
PORT_E_UNINTENDED_MODULE_STOP_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_002

- **V4M:**

Table 3-57 PORT Driver Component Configuration Parameter Dependency

Parameter	Module	Path
PORT_E_GET_CONTROL_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
PORT_E_FUSE_MONITORING_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_001
PORT_E_UNINTENDED_MODULE_STOP_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter_002

3.3.6.5 Source Code Dependency

The followings are the dependency files commonly used by the PORT Driver module:

- Det.h,
- Dem.h,
- Platform_Types.h,
- Std_Types.h,
- Port_MemMap.h,
- SchM_Port.h,
- Rte.h

3.3.6.6 Stubs

Refer to 3.2.9 for the common stubs used for PORT Driver component.

3.3.6.7 Addition Error Handling

Refer to “ Development and Production Errors” section at PORT chapter in Driver Component Embedded User's Manual.

3.3.6.8 **Restrictions**

None

3.3.6.9 **Sample Application**

3.3.6.9.1 **Sample Application Structure**

❖ **S4 G4MH:**

Refer to 3.7 Sample Application.

In the Sample Application, all the PORT APIs are invoked in the following sequences:

- Port_GetVersionInfo: The API Port_GetVersionInfo is invoked to get the version of the PORT Driver module with a variable of Std_VersionInfoType. After this API is called, the past parameter gets updated with the PORT Driver version details.
- Port_Init: The API Port_Init is invoked with a valid database address for the proper initialization of the PORT Driver. All the PORT Driver control registers and RAM variables get initialized after this API is called.
- Port_SetPinDirection: This service sets the Port Pin direction during runtime.
- Port_SetPinDefaultDirection: This service sets the Port Pin direction during runtime to the default mode set by the Port_Init().
- Port_SetToAlternateMode: This function is used to set the mode of a port pin to the configured mode during runtime.
- Port_SetToDioMode: This function is used to set the mode of a port pin to DIO mode during runtime.
- Port_Port_EcmMaskERRORIN: This function used to mask/unmask the ECM error output signal of ERRORIN.
- Port_EcmClearERRORIN: This function used to clears the ECM error status of ERRORIN.

❖ **S4 CR52:**

Refer to 3.7 Sample Application.

In the Sample Application, all the PORT APIs are invoked in the following sequences:

- Port_GetVersionInfo: The API Port_GetVersionInfo is invoked to get the version of the PORT Driver module with a variable of Std_VersionInfoType. After this API is called, the past parameter gets updated with the PORT Driver version details.
- Port_Init: The API Port_Init is invoked with a valid database address for the proper initialization of the PORT Driver. All the PORT Driver control registers and RAM variables get initialized after this API is called.
- Port_SetPinDirection: This service sets the Port Pin direction during runtime.
- Port_SetPinDefaultDirection: This service sets the Port Pin direction during runtime to the default mode set by the Port_Init().
- Port_SetToAlternateMode: This function is used to set the mode of a port pin to the configured mode during runtime.
- Port_SetToDioMode: This function is used to set the mode of a port pin to DIO mode during runtime.
- Port_Port_EcmMaskERRORIN: This function used to mask/unmask the ECM error output signal of ERRORIN.
- Port_EcmClearERRORIN: This function used to clears the ECM error status of ERRORIN.
- Port_RefreshPortDirection: This function is used to set the direction of a port pin to default direction.
- Port_SetPinMode: This function used to set the mode of a port pin to mode during runtime.
- Port_FUSEMonitoring: This function is used to check the value of configuration FUSE Monitoring register with the expected value.

❖ **V4H:**

Refer to 3.7 Sample Application.

In the Sample Application, all the PORT APIs are invoked in the following sequences:

- Port_GetVersionInfo: The API Port_GetVersionInfo is invoked to get the version of the PORT Driver module with a variable of Std_VersionInfoType. After this API is called, the past parameter gets updated with the PORT Driver version details.

- Port_Init: The API Port_Init is invoked with a valid database address for the proper initialization of the PORT Driver. All the PORT Driver control registers and RAM variables get initialized after this API is called.
- Port_SetPinDirection: This service sets the Port Pin direction during runtime.
- Port_SetPinDefaultDirection: This service sets the Port Pin direction during runtime to the default mode set by the Port_Init().
- Port_SetToAlternateMode: This function is used to set the mode of a port pin to the configured mode during runtime.
- Port_SetToDioMode: This function is used to set the mode of a port pin to DIO mode during runtime.
- Port_RefreshPortDirection: This function is used to set the direction of a port pin to default direction.
- Port_SetPinMode: This function used to set the mode of a port pin to mode during runtime.
- Port_FUSEMonitoring: This function is used to check the value of configuration FUSE Monitoring register with the expected value.
- Port_UnintendedModuleStopCheck: This function is used to check the value of configuration register with the expected value when unintended module stop occurs.

❖ **V4M:**

Refer to 3.7 Sample Application.
In the Sample Application, all the PORT APIs are invoked in the following sequences:

- Port_GetVersionInfo: The API Port_GetVersionInfo is invoked to get the version of the PORT Driver module with a variable of Std_VersionInfoType. After this API is called, the past parameter gets updated with the PORT Driver version details.
- Port_Init: The API Port_Init is invoked with a valid database address for the proper initialization of the PORT Driver. All the PORT Driver control registers and RAM variables get initialized after this API is called.
- Port_SetPinDirection: This service sets the Port Pin direction during runtime.
- Port_SetPinDefaultDirection: This service sets the Port Pin direction during runtime to the default mode set by the Port_Init().
- Port_SetToAlternateMode: This function is used to set the mode of a port pin to the configured mode during runtime.
- Port_SetToDioMode: This function is used to set the mode of a port pin to DIO mode during runtime.
- Port_RefreshPortDirection: This function is used to set the direction of a port pin to default direction.
- Port_SetPinMode: This function used to set the mode of a port pin to mode during runtime.
- Port_FUSEMonitoring: This function is used to check the value of configuration FUSE Monitoring register with the expected value.

Port_UnintendedModuleStopCheck: This function is used to check the value of configuration register with the expected value when unintended module stop occurs.

3.3.6.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-58 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

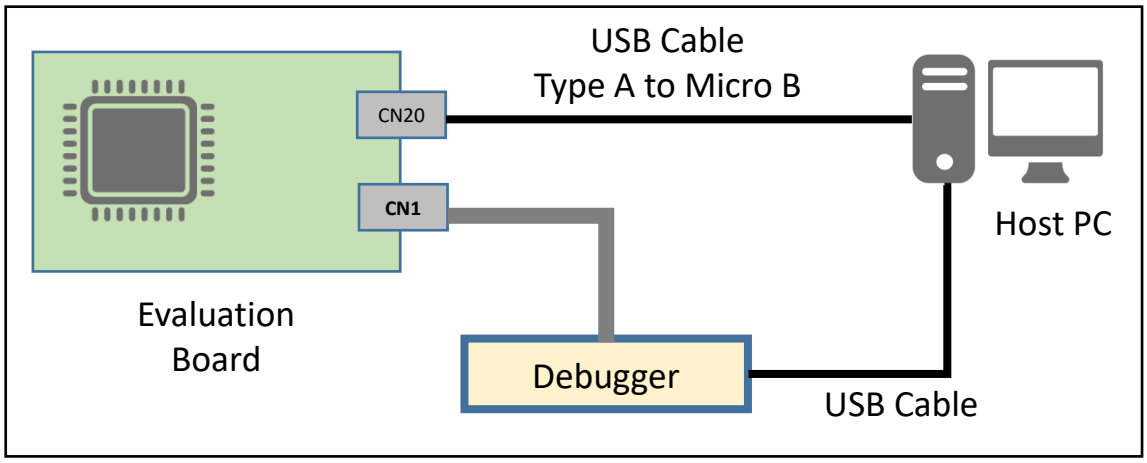

Figure 3-21 S4 Spider board connection of PORT (S4_G4MH) Sample Application

- **S4 CR52 Environment**

Table 3-59 S4 CR52 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

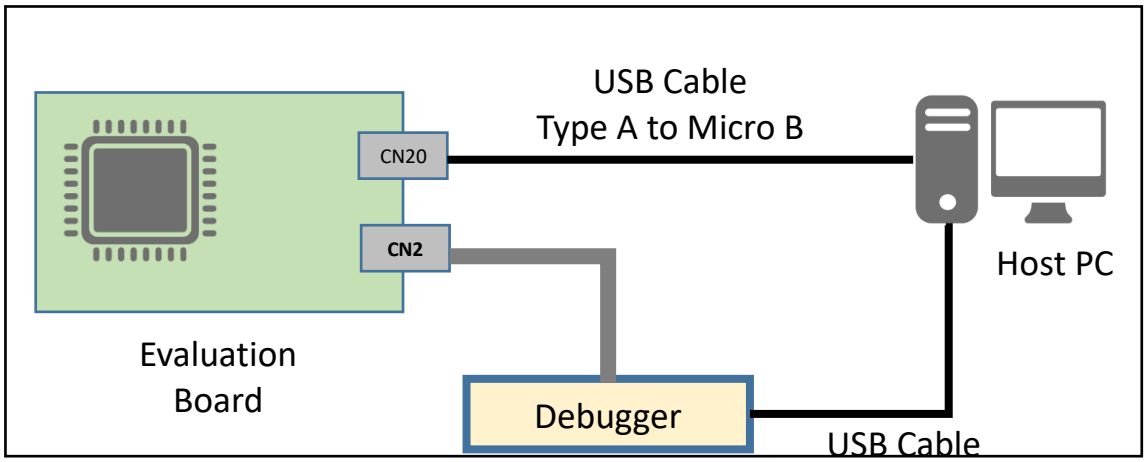

Figure 3-22 S4 Spider board connection of PORT (S4_CR52) Sample Application

- **V4H Environment**

Table 3-60 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

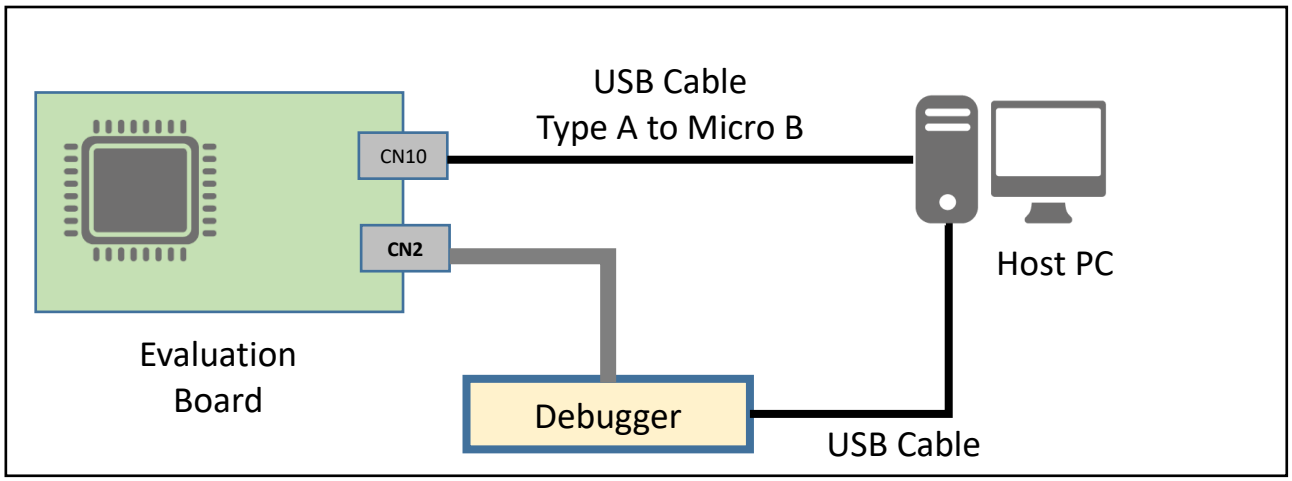

Figure 3-23 V4H White Hawk board connection of PORT (V4H) Sample Application

- **V4M Environment**

Table 3-61 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

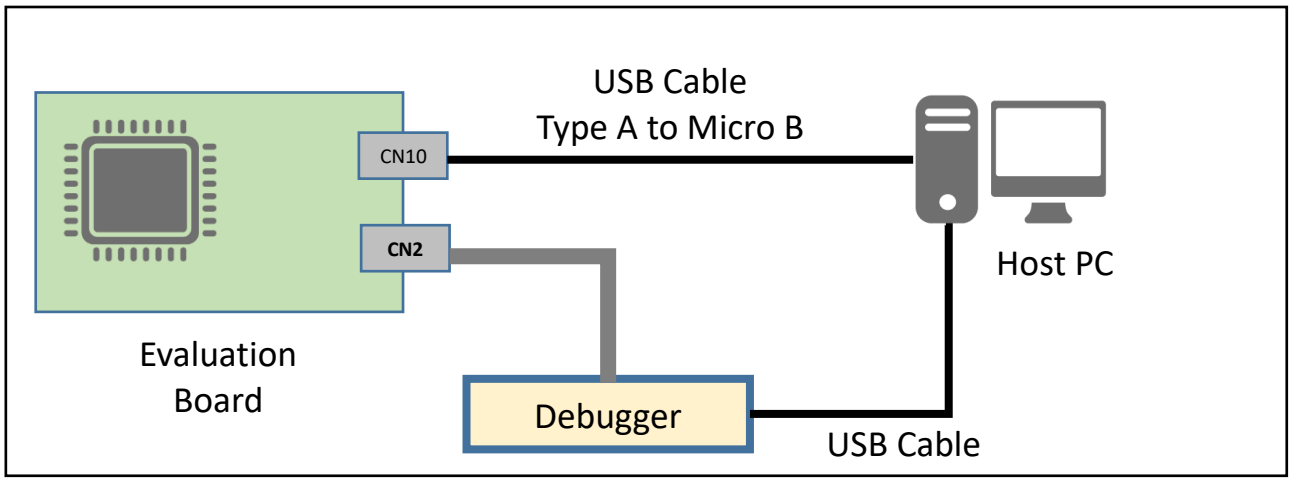

Figure 3-24 V4M Gray Hawk board connection of PORT (V4M) Sample Application

3.3.6.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.6.9.4**How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

[R-Car S4_G4MH]

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\S4\common_family\make\ghs
- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat Port R19-11 S4 no unset

[R-Car S4_CR52]

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\S4\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat Port R19-11 S4 no unset

[R-Car V4H]

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\V4H\common_family\make\arm

- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat Port R19-11 V4H no unset

[R-Car V4M]

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\V4M\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat Port R19-11 V4M no unset

3.3.6.9.5 How to run Sample application

Sample application shall include the following sequence for generic usage in PORT driver.

Step 1: Set BreakPoint at the end of the main function

Step 2: Run program to breakpoint, confirm all elements in the array GaaTestResult [0..n] are set to 1 or check by print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.
“EXECUTED OK”: Sample application execution is successful.
“EXECUTED NOT OK”: Sample application execution is failed.
“PROGRAM STOP”: Sample application execution is completed.

3.3.6.10 ROM/RAM Usage

See Appendix PORT section for information on measuring RAM/ROM consumption.

3.3.6.11 Stack Depth

See Appendix PORT section for information on measuring stack depth.

3.3.6.12 Throughput Details

See Appendix PORT section for information on measuring execution time, and functional testing.

3.3.7 PWM Driver Component

3.3.7.1 Module Overview

The PWM Driver component provides services for PWM Driver component initialization, de-initialization, setting the Period and Duty Cycle for a PWM channel, reading the internal state of PWM Output signal and setting the PWM Output to idle state and disabling or enabling the PWM signal edge notification. The PWM Driver component is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.

The PWM Driver component is divided into PWM High Level Driver and PWM Low Level Driver to minimize the effort and to optimize the reuse of the software developed on different platforms.

The PWM High Level Driver exports the APIs to the upper modules. All references to the specific microcontroller features and registers are provided in the PWM Low Level Driver.

Timer ATU-V/E, and TAUD/TAUJ timer are used in the PWM Driver Component to generate variable PWM output.

The channel level notifications are provided for the rising and falling edge, and both edges. Any of these notifications will be active only when these are configured for the corresponding channel and enabled by using PWM Driver component APIs.

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

- Initialization
- De-Initialization
- Set the channel output to Idle
- Set Duty Cycle
- Set Duty Cycle and Period
- Notification services (at the beginning, at the end and on both edged of a period)
- Get Version information
- Select channel clock source at runtime
- Synchronous start/stop of channels

3.3.7.2 Module Dependency

The dependency of PWM Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

MCU
The PWM Driver depends on MCU for setting the system clock and PLL in MCU module. If MCU module is not available, the functionality of system clock and PLL settings shall be stubbed.

PORT
Port pins used by the PWM Driver shall be configured using the PORT module.

IO Hardware Abstraction Layer
The PWM Driver depends on the IO Hardware Abstraction Layer, which invokes the APIs and receives the callback notifications. If IO Hardware Abstraction Layer Module is not available, then the required functionality shall be stubbed.

OS
As the PWM Driver uses interrupts, it depends on the OS which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

3.3.7.3 **Folder Structure**

Table 3-62 to **Table 3-64** shows the list of Source Code Files and Header Files for PWM module.

Table 3-62 PWM Header Files (1/2)

Location: rel\modules\pwm\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
Include	-	-	-	-	-
HWIP	-	-	-	-	-
ATU	-	-	-	-	-
Pwm_ATU_Irq.h	-	-	-	Product	This file contains the macro for the PWM Timer channels. It also contains the external declaration for the interrupt functions used by PWM Driver component in ATU .
Pwm_ATU_LLDriver.h	-	-	-	Product	This file contains the external declaration for the internal functions (Low Level Driver) called by the PWM Driver APIs in ATU .
Pwm_ATU_PBTypes.h	-	-	-	Product	This file contains the data structure definitions for Timer control registers, Timer interrupt control registers, and channel configuration. It also contains the macros used by the PWM Driver Component for ATU in ATU.
Pwm_ATU_Ram.h	-	-	-	Product	This file contains the external declaration for the global variables used by PWM Driver Component for ATU in ATU.
TAU	-	-	-	-	-
Pwm_TAU_PBTypes.h	-	-	-	Product	This file contains the data structure definitions for Timer control registers, Timer interrupt control registers, and channel configuration. It also contains the macros used by the PWM Driver Component for TAU.
TAUD	-	-	-	-	-
Pwm_TAUD_Irq.h	-	x	-	Product	This file contains the macro for the PWM Timer channels. It also contains the external declaration for the interrupt functions used by PWM Driver component in TAUD.
Pwm_TAUD_LLDriver.h	-	x	-	Product	This file contains the external declaration for the internal functions (Low Level Driver) called by the PWM Driver APIs in TAUD.
Pwm_TAUD_PBTypes.h	-	x	-	Product	This file contains the data structure definitions for Timer control registers, Timer interrupt control registers, and channel configuration. It also contains the macros used by the PWM Driver Component in TAUD.
Pwm_TAUD_Ram.h	-	x	-	Product	This file contains the external declaration for the global variables used by PWM Driver Component for in TAUD.

Table 3-63 PWM Header Files (2/2)

Location: rel\modules\pwm\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
TAUJ	-	-	-	-	-
Pwm_TAUJ_Irq.h	-	x	-	Product	This file contains the macro for the PWM Timer channels. It also contains the external declaration for the interrupt functions used by PWM Driver component in TAUD.
Pwm_TAUJ_LLDriver.h	-	x	-	Product	This file contains the external declaration for the internal functions (Low Level Driver) called by the PWM Driver APIs in TAUJ.
Pwm_TAUJ_PBTypes.h	-	x	-	Product	This file contains the data structure definitions for Timer control registers, Timer interrupt control registers, and channel configuration. It also contains the macros used by the PWM Driver Component in TAUJ.
Pwm_TAUJ_Ram.h	-	x	-	Product	This file contains the external declaration for the global variables used by PWM Driver Component for in TAUJ.
Pwm.h	-	x	-	Product	This file contains the extern declaration for the PWM Driver Component APIs, macro for service ids, and DET Errors. It also contains Type definitions and the structure declaration for init configuration.
Pwm_MultiInstance.h	-	x	-	Product	This file contains the extern declaration for the PWM Driver Component APIs and macro for multi-instance. It also contains Type definitions and the structure declaration for init configuration.
Pwm_PBTypes.h	-	x	-	Product	This file contains the data structure definitions for Timer control registers, Timer interrupt control registers, and channel configuration. It also contains the macros used by the PWM Driver Component.
Pwm_Ram.h	-	x	-	Product	This file contains the external declaration for the global variables used by PWM Driver Component.
Pwm_Types.h	-	x	-	Product	This file contains the common macro definitions and the data types required internally by the PWM software component.
Pwm_Version.h	-	x	-	Product	This file contains the definitions of AUTOSAR version numbers of all modules interfaced to PWM.

x: applicable

-: not applicable

Table 3-64 PWM Source Files

Location: rel\modules\pwm\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
Src	-	-	-	-	-
HWIP	-	-	-	-	-
ATU	-	-	-	-	-
Pwm_ATU_Irq.c	-	-	-	Product	This file contains the implementation of all the interrupt functions used by PWM Driver Component for ATU.
Pwm_ATU_LLDriver.c	-	-	-	Product	This file contains implementation of all Low-Level Driver functions invoked by PWM Driver APIs for ATU.
Pwm_ATU_Ram.c	-	-	-	Product	This file contains declarations of all global variables used by the PWM Driver Component for ATU.
TAU	-	-	-	-	-
TAUD	-	-	-	-	-
Pwm_TAUD_Irq.c	-	x	-	Product	This file contains the implementation of all the interrupt functions used by PWM Driver Component for TAUD.
Pwm_TAUD_LLDriver.c	-	x	-	Product	This file contains implementation of all Low-Level Driver functions invoked by PWM Driver APIs for TAUD.
Pwm_TAUD_Ram.c	-	x	-	Product	This file contains declarations of all global variables used by the PWM Driver Component for TAUD.
TAUJ	-	-	-	-	-
Pwm_TAUJ_Irq.c	-	x	-	Product	This file contains the implementation of all the interrupt functions used by PWM Driver Component for TAUJ.
Pwm_TAUJ_LLDriver.c	-	x	-	Product	This file contains implementation of all Low-Level Driver functions invoked by PWM Driver APIs for TAUJ.
Pwm_TAUJ_Ram.c	-	x	-	Product	This file contains declarations of all global variables used by the PWM Driver Component for TAUJ.
Pwm.c	-	x	-	Product	This file contains the implementation of PWM Driver Component APIs.
Pwm_Ram.c	-	x	-	Product	This file contains declarations of all global variables used by the PWM Driver Component.
Pwm_Version.c	-	x	-	Product	This file contains the code for checking version of all modules interfaced to PWM.

x: applicable

-: not applicable

Table 3-65 shows the list of Parameter Definition Files for PWM module.

Table 3-65 PWM Parameter Definition Files

Location: rel\modules\pwm\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
R19_11	S4_G4MH	R1911_PWM_S4_RTM8RC79FG.arxml	RTM8RC79FG

Note Product Name**:** Product Names supported by “Files”.

3.3.7.4 Configuration Parameter Dependency

Table 3-66 shows the list of configuration parameter dependency for PWM Driver component.

Table 3-66 PWM Driver Component Configuration Parameter Dependency

Parameter	Module	Path
PwmMcuClockReferencePoint	MCU	/AUTOSAR/EcucDefs/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockReferencePoint
PWM_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.3.7.5 Source Code Dependency

The followings are dependency files commonly used by the PWM Driver module:

- Det.h,
- Dem.h,
- rh850_Types.h,
- Std_Types.h,
- Pwm_MemMap.h,
- SchM_Pwm.h,
- Rte.h,
- Os.h

3.3.7.6 Stubs

Refer to 3.2.9 for the common stubs used for PWM Driver component.

3.3.7.7 Addition Error Handling

Refer to " Development and Production Errors" section at PWM chapter in Driver Component Embedded User's Manual.

3.3.7.8 Restrictions

None

3.3.7.9 Sample Application

3.3.7.9.1 Sample Application Structure

Refer to 3.7 Sample Application.

(1) Single-core Sample Application Structure

In the Sample Application all the PWM APIs are invoked in the following sequences:

- The API Pwm_GetVersionInfo is invoked to get the version of the PWM Driver module with a variable of Std_VersionInfoType. After this API is called, the formal parameter gets updated with the PWM Driver version details.
- The API Pwm_Init is invoked with a valid database address for the proper initialization of the PWM Driver, and all the PWM Driver control registers and RAM variables get initialized after this API is called.
- The API Pwm_SynchronousInit is invoked to initialize PWM channels configured to support synchronous start/stop of timers. (Not Supported)
- The API Pwm_SynchronousStart is invoked to start PWM channels configured to support synchronous start of timers. (Not Supported)
- The API Pwm_SetDutyCycle is invoked to set the duty cycle of a PWM channel. This API is applicable for PWM channels configured as PWM_VARIABLE_PERIOD, PWM_FIXED_PERIOD or PWM_FIXED_PERIOD_SHIFTED class type.
- The API Pwm_SetPeriodAndDuty is invoked to set the Duty cycle and Period of a PWM channel. This API is applicable for the PWM channels configured as PWM_VARIABLE_PERIOD class type only.

- The API Pwm_SetOutputToIdle is invoked to set the output of a PWM channel to the idle state configured.
- The API Pwm_GetOutputState is invoked to get the channel output state and provides the service to read the internal state of a PWM channel.
- The API Pwm_EnableNotification is invoked to enable notifications of a PWM channel. Notifications can be configured for rising/falling edge or both edges triggered. This API also enables the interrupt processing and clears the pending interrupts. (Not Supported)
- The API Pwm_DisableNotification is invoked to disable notification of PWM channel. This API disables the interrupt processing. (Not Supported)
- The API Pwm_SelectChannelClk is invoked to change the period of a PWM channel by selecting one of the four clock sources. (Not Supported)
- The API Pwm_SetChannelOutput is invoked to set the PWM output to a constant signal for the duration of the current period.
- The API Pwm_SynchronousStop is invoked to stop PWM channels configured to support synchronous stop of timers. (Not Supported)

The API Pwm_DeInit is invoked to de-initialize all the control registers and RAM variables. This service includes setting all PWM channel output signals to the idle state configured and disables the interrupts and notifications for all configured PWM channels.

(2) Multi-Core Sample Application Structure (Not supported)

The multi-core / multi-instantiation sample application provides how to use the PWM driver for multi-core device. It also provides instance 0 and 1.
The configuration is shown as follows:

U2A16:
[PE0] – [Instance 0]
Instance 0 uses the following channels.
- TAUJ sub block 0 Channel 0
- TAUJ sub block 0 channel 1
- TAUJ sub block 0 channel 2

[PE1] – [Instance 1]
Instance 1 uses the following channels.
- TAUD subblock 1 channel 10

U2A8:
[PE0] – [Instance 0]
Instance 0 uses the following channels.
- TAUJ sub block 0 Channel 0
- TAUJ sub block 0 channel 1
- TAUJ sub block 0 channel 2

[PE1] – [Instance 1]
Instance 1 uses the following channels.
- TAUD subblock 1 channel 10

[Behavior of multi-core / multi-instantiation sample application]

This sample application assumes that PWM driver use Instance 0 and Instance 1 to generate output. In PE0,there are two channels using synchronous function.
The sequence of multi-core sample application is the same as the single-core sample application.

- The API Pwm_GetVersionInfo is invoked to get the version of the PWM Driver module with a variable of Std_VersionInfoType. After this API is called, the formal parameter gets updated with the PWM Driver version details.
- The API Pwm_Init is invoked in Pe0 for all PEs with a valid database address for the proper initialization of the PWM Driver. All the PWM Driver control registers and RAM variables get initialized after this API is called.
- The API Pwm_SynchronousInit is invoked to initialize the PWM channels configured to support synchronous start/stop of timers.
- The API Pwm_SynchronousStart is invoked to start the PWM channels configured to support synchronous start of timers.
- The API Pwm_EnableNotification is invoked to enable notifications of the PWM channel. Notifications can be configured for rising/falling edge or both edges triggered. This API also enables the interrupt processing and clears the pending interrupts.
- The API Pwm_DisableNotification is invoked to disable notification of the PWM channel. This API disables the interrupt processing.
- The API Pwm_SetDutyCycle is invoked to set the duty cycle of the PWM channel. This API is applicable for the PWM channels configured as PWM_VARIABLE_PERIOD, PWM_FIXED_PERIOD or PWM_FIXED_PERIOD_SHIFTED class type. Also, it is invoked sequentially by using inter core exclusive control in each PE.
- The API Pwm_SetPeriodAndDuty is invoked to set the Duty cycle and Period of the PWM channel. This API is applicable for the PWM channels configured as PWM_VARIABLE_PERIOD class type only. Also, it is invoked sequentially by using inter core exclusive control in each PE.
- The API Pwm_SetOutputToIdle is invoked to set the output of a PWM channel to the idle state configured.

- The API Pwm_GetOutputState is invoked to get the channel output state and provides the service to read the internal state of a PWM channel.
- The API Pwm_SelectChannelClk is invoked to change the period of the PWM channel by selecting one of the four clock sources.
- The API Pwm_SetChannelOutput is invoked to set the PWM output to a constant signal for the duration of the current period.
- The API Pwm_SynchronousStop is invoked to stop PWM channels configured to support synchronous stop of timers. Also, it is invoked sequentially by using inter core exclusive control in each PE.
- The API Pwm_DeInit is invoked to de-initialize all the control registers and RAM variables. This service includes setting all PWM channel output signals to the idle state configures and disables the interrupts and notifications for all configured PWM channels.

(3) Multi-Core Sample Application Structure (Not Supported)
The multi-core / multi-instantiation sample application provides how to use the PWM driver for multi-core device. It also provides instance 0 and 1.
The configuration is shown as follows:

S4_G4MH:
[PE0] – [Instance 0]
Instance 0 uses the following channels.
- TAUJ sub block 0 Channel 0
- TAUJ sub block 0 channel 1
- TAUJ sub block 0 channel 2

[PE1] – [Instance 1]
Instance 1 uses the following channels.
- TAUD subblock 1 channel 10

[Behavior of multi-core / multi-instantiation sample application]

This sample application assumes that PWM driver use Instance 0 and Instance 1 to generate output. In PE0,there are two channels using synchronous function.
The sequence of multi-core sample application is the same as the single-core sample application.

- The API Pwm_GetVersionInfo is invoked to get the version of the PWM Driver module with a variable of Std_VersionInfoType. After this API is called, the formal parameter gets updated with the PWM Driver version details.
- The API Pwm_Init is invoked in Pe0 for all PEs with a valid database address for the proper initialization of the PWM Driver. All the PWM Driver control registers and RAM variables get initialized after this API is called.
- The API Pwm_SynchronousInit is invoked to initialize the PWM channels configured to support synchronous start/stop of timers. (Not Supported)
- The API Pwm_SynchronousStart is invoked to start the PWM channels configured to support synchronous start of timers. (Not Supported)
- The API Pwm_EnableNotification is invoked to enable notifications of the PWM channel. Notifications can be configured for rising/falling edge or both edges triggered. This API also enables the interrupt processing and clears the pending interrupts. (Not Supported)
- The API Pwm_DisableNotification is invoked to disable notification of the PWM channel. This API disables the interrupt processing. (Not Supported)
- The API Pwm_SetDutyCycle is invoked to set the duty cycle of the PWM channel. This API is applicable for the PWM channels configured as PWM_VARIABLE_PERIOD, PWM_FIXED_PERIOD or PWM_FIXED_PERIOD_SHIFTED class type. Also, it is invoked sequentially by using inter core exclusive control in each PE.

- The API Pwm_SetPeriodAndDuty is invoked to set the Duty cycle and Period of the PWM channel. This API is applicable for the PWM channels configured as PWM_VARIABLE_PERIOD class type only. Also, it is invoked sequentially by using inter core exclusive control in each PE.
- The API Pwm_SetOutputToIdle is invoked to set the output of a PWM channel to the idle state configured.
- The API Pwm_GetOutputState is invoked to get the channel output state and provides the service to read the internal state of a PWM channel.
- The API Pwm_SelectChannelClk is invoked to change the period of the PWM channel by selecting one of the four clock sources. (Not Supported)
- The API Pwm_SetChannelOutput is invoked to set the PWM output to a constant signal for the duration of the current period.
- The API Pwm_SynchronousStop is invoked to stop PWM channels configured to support synchronous stop of timers. Also, it is invoked sequentially by using inter core exclusive control in each PE. (Not Supported)
- The API Pwm_DeInit is invoked to de-initialize all the control registers and RAM variables. This service includes setting all PWM channel output signals to the idle state configures and disables the interrupts and notifications for all configured PWM channels.

(4) Multi-Core Sample Application Structure (Not supported)

The multi-core / multi-instantiation sample application provides how to use PWM driver for multi-core device. It also provides instance 0 and 1. The configuration is shown as follows.

[PE0] – [Instance 0]
Instance 0 uses the following channels.
- ATUE sub block 0 Channel 0
- ATUE sub block 0 channel 1
- ATUE sub block 0 channel 2

[PE1] – [Instance 1]
Instance 1 uses the following channels.
- ATUE sub block 1 channel 10

E2H:
[PE0] – [Instance 0]
Instance 0 uses the following channels.
- ATUE sub block 0 Channel 0
- ATUE sub block 0 channel 1
- ATUE sub block 0 channel 2

[PE1] – [Instance 1]
Instance 1 uses the following channels.
- ATUE sub block 1 channel 10

E2UH:
[PE0] – [Instance 0]
Instance 0 uses the following channels.
- ATUE sub block 0 Channel 0
- ATUE sub block 0 channel 1
- ATUE sub block 0 channel 2

[PE1] – [Instance 1]
Instance 1 uses the following channels.
- ATUE sub block 1 channel 10

[Behavior of multi-core / multi-instantiation sample application]

This sample application assumes that the PWM driver uses Instance 0 and 1 to generate output. In PE0, there are two channels using synchronous function.

The sequence of multi-core sample application is the same as single-core sample application.

- The API Pwm_GetVersionInfo is invoked to get the version of the PWM Driver module with a variable of Std_VersionInfoType. After this API is called, the formal parameter gets updated with the PWM Driver version details.
- The API Pwm_Init is invoked in Pe0 for all PEs with a valid database address for the proper initialization of the PWM Driver, and all the PWM Driver control registers and RAM variables get initialized after this API is called.
- The API Pwm_SynchronousInit is invoked to initialize the PWM channels configured to support synchronous start/stop of timers.
- The API Pwm_SynchronousStart is invoked to start the PWM channels configured to support synchronous start of timers.
- The API Pwm_EnableNotification is invoked to enable notifications of the PWM channel. Notifications can be configured for rising/falling edge or both edges triggered. This API also enables the interrupt processing and clears the pending interrupts.
- The API Pwm_DisableNotification is invoked to disable notification of the PWM channel. This API disables the interrupt processing.
- The API Pwm_SetDutyCycle is invoked to set the duty cycle of the PWM channel. This API is applicable for the PWM channels configured as PWM_VARIABLE_PERIOD, PWM_FIXED_PERIOD or PWM_FIXED_PERIOD_SHIFTED class type. Also, it is invoked sequentially by using inter core exclusive control in each PE.
- The API Pwm_SetPeriodAndDuty is invoked to set the Duty cycle and Period of the PWM channel. This API is applicable for the PWM channels configured as PWM_VARIABLE_PERIOD class type only. Also, it is invoked sequentially by using inter core exclusive control in each PE.
- The API Pwm_SetOutputToIdle is invoked to set the output of the PWM channel to the idle state configured.
- The API Pwm_GetOutputState is invoked to get the channel output state and provides the service to read the internal state of the PWM channel.
- The API Pwm_SelectChannelClk is invoked to change the period of the PWM channel by selecting one of the four clock sources.
- The API Pwm_SetChannelOutput is invoked to set the PWM output to a constant signal for the duration of the current period.
- The API Pwm_SynchronousStop is invoked to stop the PWM channels configured to support synchronous stop of timers. Also, it is invoked sequentially by using inter core exclusive control in each PE.
- The API Pwm_DeInit is invoked to de-initialize all the control registers and RAM variables. This service includes setting all PWM channel output signals to the idle state configured and disables the interrupts and notifications for all configured PWM channels.

3.3.7.9.2**Recommended Environtment**

Table 3-67 S4 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger + Logic Port Analyzer
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

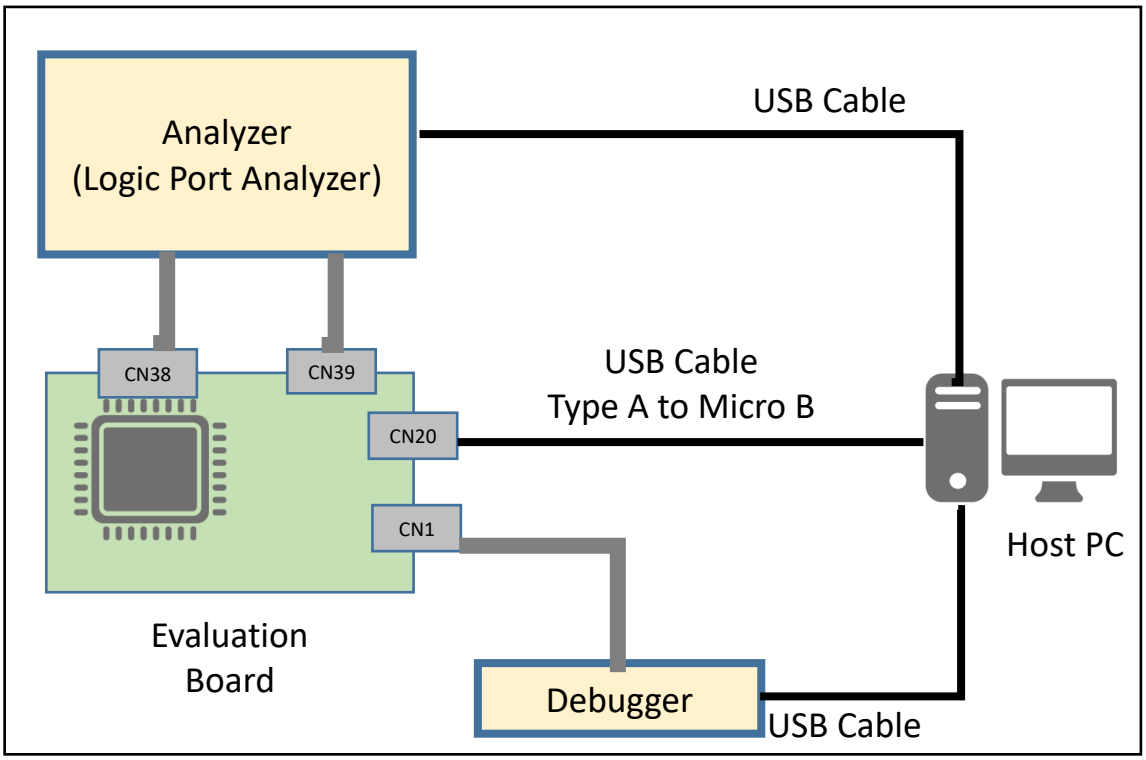

Figure 3-25 S4 Spider board connection of PWM Sample Application

3.3.7.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.

3.3.7.9.4**How to build Sample Application**

Refer to 3.7.2.2 How to build the Sample Application

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:
  external\rel\S4\common_family\make\ghs
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat PWM R19-11 S4 No

3.3.7.9.5 How to run Sample Application

In the Sample Application, all the PWM APIs are invoked in the following sequences.

- Prepare a logic port analyzer to check the output on the output pin that have been configured according to the CDF file.
- Set the break point after the PWM version check condition
- Run the program to breakpoint. If the versionInfo returns correct version information, the global variable GucVerCheckStatus will be set as TRUE.

For set duty cycle functionality:

- To read the signal of channel that set the duty cycle, the logic port should be connect to the correct output pin. Set the breakpoint after invoked the API Pwm_SetDutyCycle, the signal will be present on the logic port application with the duty cycle that set to the output pin.

For set period and duty functionality:

- To read the signal of channel that set the period and duty, the logic port should be connect to the correct output pin. Set the breakpoint after invoked the API Pwm_SetPeriodAndDuty, the signal will be present on the logic port application with the period and duty cycle that set to the output pin.

For set output to idle functionality:

- To read the signal of channel that set the output state, the logic port should be connect to the correct output pin. Set the breakpoint after invoked the API Pwm_SetOutputToIdle, the signal will be present on the logic port application with the idle state of the channel that set via the API on the output pin.

For set channel output functionality:

- To read the signal of channel that set the channel output, the logic port should be connect to the correct output pin. Set the breakpoint after invoked the API Pwm_SetChannelOutput, the signal will be present on the logic port application with the output state of channel that set via the API on the output pin.

For select channel clock functionality:

- To read the signal of channel that select the clock for the channel, the logic port should be connect to the correct output pin. Set the breakpoint after invoked the API Pwm_SelectChannelClk, the signal will be present on the logic port application with with the channel and the clock that set to the channel via the API on the output pin.
- Continue to run program until “End of main function” Sample_end() is reached. Sample application sequence is completed

Note: The sample application execution and result can be checked by print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed

3.3.7.10 ROM/RAM Usage

See Appendix PWM section for information on measuring RAM/ROM consumption.

3.3.7.11 Stack Depth

See Appendix PWM section for information on measuring stack depth.

3.3.7.12 Throughput Details

See Appendix PWM section for information on measuring execution time, and functional testing.

3.3.8 SPI Driver Component

3.3.8.1 Module Overview

The SPI Driver is split as High-Level Driver and Low-Level Driver. 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 into 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.

The SPI Driver component provides services for:

- Initialization and de-initialization
- Buffer Management
- Communication
- Status information
- Module version information
- Memory mapping
- Compiler abstraction

3.3.8.2 Module Dependency

The dependency of SPI Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

PORT
The basic SPI functionality for both CSIH and HS-SPI must be configured as an alternate functionality by the PORT module.

MCU
The configuration of SPI module for jobs contains the references to the MCU module for the input clock frequency for the SPI HW Unit. Hence, SPI baud rate depends on the frequency set in the MCU module.

IO Hardware Abstraction Layer
The IO Hardware Abstraction Layer invokes APIs of the SPI module and receives the callback notifications.

Memory Hardware Abstraction Layer
The Memory Hardware Abstraction Layer invokes APIs of the SPI module if the drivers for any external memory devices (for example, external EEPROM) are implemented through the SPI module.

Onboard Device Abstraction Layer
The Onboard Device Abstraction Layer invokes APIs of the SPI module if the drivers for any external devices (for example, external watchdog) are implemented through the SPI module.

OS
As the SPI Driver uses interrupts, it depends on the OS which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

RTE
The functions related to critical section protection area of the SPI module are invoked by the Run Time Environment (RTE) module.

DEM
The SPI module uses the DEM module to get the reference for all production errors.

3.3.8.3 **Folder Structure**

Table 3-68, **Table 3-69**, **Table 3-70** show the list of Source Code Files and Header Files for SPI module.

Table 3-68 SPI Header Files (1/2)

Location: rel\modules\spi\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Spi.h	-	x	x	x	Product	This file provides extern declarations for all SPI Driver Component APIs. It provides service Ids of APIs, DET Error codes and type definitions for SPI Driver initialization structure. This header file shall be included in other modules to use the features of SPI Driver Component.
Spi_Irq.h	-	x	x	x	Product	This file contains the definitions for the masks of EIC registers.
Spi_LTTypes.h	-	x	x	x	Product	This file contains the data structure definitions of storing the current status of SPI communication and the external buffer attributes.
Spi_MultiInstance.h	-	x	x	x	Product	This file contains MULTI macro and definitions for multi-instance.
Spi_PBTypes.h	-	x	x	x	Product	This file contains the data structure definitions of the channel configuration, job configuration, sequence configuration, hardware unit information and hardware-dependent function pointer table.
Spi_Ram.h	-	x	x	x	Product	This file contains the extern declarations for the global variables defined in Spi_Ram.c file and the version information of the file.
Spi_Scheduler.h	-	x	x	x	Product	This file contains the function prototypes defined in Spi_Scheduler.c file.
Spi_Types.h	-	x	x	x	Product	This file contains the common macro definitions and the data types required to use the SPI software component.
Spi_Version.h	-	x	x	x	Product	This file contains the version information of the file.
CSIX	-	-	-	-	-	-
Spi_CSIX_Irq.h	-	-	-	-	Product	This file contains the function prototypes defined in Spi_CSIX_Irq.c file.
Spi_CSIX_LLDriver.h	-	-	-	-	Product	This file contains the function prototypes and register definitions of SPI Low level Driver for CSIX.
HSPI	-	-	-	-	-	-
Spi_HSPI_Irq.h	-	-	-	-	Product	This file contains the function prototypes defined in Spi_HSPI_Irq.c file.
Spi_HSPI_LLDriver.h	-	-	-	-	Product	This file contains the function prototypes and register definitions of SPI Low level Driver for HS-SPI.

Table 3-69 SPI Header Files (2/2)

Location: rel\modules\spi\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
MSPI	-	-	-	-	-	-
Spi_MSPI_Irq.h	-	x	-	-	Product	This file contains the function prototypes defined in Spi_MSPI_Irq.c file.
Spi_MSPI_LLDriver.h	-	x	-	-	Product	This file contains the function prototypes and register definitions of SPI Low level Driver for MSPI.
MSIOF	-				-	-
Spi_MSIOF_Irq.h	-	-	x	x	Product	This file contains the function prototypes defined in Spi_MSIOF_Irq.c file.
Spi_MSIOF_LLDriver.h	-	-	x	x	Product	This file contains the function prototypes and register definitions of SPI Low level Driver for MSIOF.
sDMAC	-				-	-
Spi_sDMAC_Irq.h	-	x	-	-	Product	This file contains the function prototypes defined in Spi_sDMAC_Irq.c file.
Spi_sDMAC_LLDriver.h	-	x	-	-	Product	This file contains the function prototypes and register definitions of SPI Low level Driver for DMA.
SYS-DMAC	-				-	-
Spi_SYSDMAC_Irq.h	-	-	x	x	Product	This file contains the function prototypes defined in Spi_SYSDMAC_Irq.c file.
Spi_SYSDMAC_LLDriver.h	-	-	x	x	Product	This file contains the function prototypes and register definitions of SPI Low level Driver for DMA.

x: applicable

-: not applicable

Table 3-70 SPI Source Files

Location: rel\modules\spi\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Spi.c	-	x	x	x	Product	This file contains the implementation of all APIs.
Spi_Ram.c	-	x	x	x	Product	This file contains the global variables used by SPI Driver Component.
Spi_Scheduler.c	-	x	x	x	Product	This file contains the SPI Scheduler code. This contains function to schedule the sequences according to the priority of the jobs.
Spi_Version.c	-	x	x	x	Product	This file contains the code for checking version of all modules interfaced to SPI Driver.
CSIX					-	-
Spi_CSIX_Irq.c	-	-	-	-	Product	This file contains the ISR functions for SPI Low Level Driver for CSIX.
Spi_CSIX_LLDriver.c	-	-	-	-	Product	This file contains the implementation of SPI Low Level Driver code for CSIX.
HSPI					-	-
Spi_HSPI_Irq.c	-	-	-	-	Product	This file contains the ISR functions for SPI Low Level Driver for HS-SPI.
Spi_HSPI_LLDriver.c	-	-	-	-	Product	This file contains the implementation of SPI Low Level Driver code for HS-SPI.
MSPI					-	-
Spi_MSPI_Irq.c	-	x	-	-	Product	This file contains the ISR functions for SPI Low Level Driver for MSPI.
Spi_MSPI_LLDriver.c	-	x	-	-	Product	This file contains the implementation of SPI Low Level Driver code for MSPI.
MSIOF					-	-
Spi_MSIOF_Irq.c	-	-	x	x	Product	This file contains the ISR functions for SPI Low Level Driver for MSIOF.
Spi_ MSIOF_LLDriver.c	-	-	x	x	Product	This file contains the implementation of SPI Low Level Driver code for MSIOF.
sDMAC					-	-
Spi_sDMAC_Irq.c	-	x	-	-	Product	This file contains the ISR functions for SPI Low Level Driver for DMA.
Spi_sDMAC_LLDriver.c	-	x	-	-	Product	This file contains the implementation of SPI Low Level Driver code for DMA.
SYS-DMAC					-	-
Spi_SYSDMAC_Irq.c	-	-	x	x	Product	This file contains the ISR functions for SPI Low Level Driver for DMA.
Spi_SYSDMAC_LLDriver.c	-	-	x	x	Product	This file contains the implementation of SPI Low Level Driver code for DMA.

x: applicable

-: not applicable

Table 3-71 shows the list of Parameter Definition Files for SPI module.

Table 3-71 SPI Parameter Definition Files

Location: rel\modules\spi\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4	R1911_SPI_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R1911_SPI_V4H.arxml	V4H
	V4M	R1911_SPI_V4M.arxml	V4M

Note Product Name: Product Names supported by “Files”.

3.3.8.4 Configuration Parameter Dependency

Table 3-72 shows the list of configuration parameter dependency for SPI Driver component in S4 G4MH.

Table 3-72 S4 G4MH SPI Driver Component Configuration Parameter Dependency

Parameter	Module	Path
SpiClockFrequencyRef	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuMSPIClk
SPI_E_HARDWARE_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
SPI_E_DATA_TX_TIMEOUT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
SPI_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

Table 3-73 shows the list of configuration parameter dependency for SPI Driver component in V4H.

Table 3-73 V4H SPI Driver Component Configuration Parameter Dependency

Parameter	Module	Path
SpiClockFrequencyRef	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuMSOClk
SPI_E_HARDWARE_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
SPI_E_DATA_TX_TIMEOUT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
SPI_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
SPI_E_WRITE_VERIFY_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

Table 3-74 V4M SPI Driver Component Configuration Parameter Dependency

Parameter	Module	Path
SpiClockFrequencyRef	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuMSOClk
SPI_E_HARDWARE_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
SPI_E_DATA_TX_TIMEOUT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.3.8.5 Source Code Dependency

The followings are the dependency files commonly used by the SPI Driver module:

- Det.h,
- Dem.h,
- rh850_Types.h,
- Std_Types.h,
- Spi_MemMap.h,

- SchM_Spi.h,
- Rte.h,
- Os.h

3.3.8.6 **Stubs**

Refer to 3.2.9 for the common stubs used for SPI Driver component.

3.3.8.7 **Addition Error Handling**

Refer to " Development and Production Errors" section at SPI chapter in Driver Component Embedded User's Manual.

3.3.8.8 **Restrictions**

None

3.3.8.9 **Sample Application**

3.3.8.9.1 **Sample Application Structure**

Refer to 3.7 Sample Application.

(1) Single Core Sample Application Structure

❖ **S4 G4MH:**

In the S4 Sample Application, all the SPI APIs are invoked in the following sequences,

- The API Spi_Init is invoked with a valid database address for the proper initialization of the SPI Driver, All the SPI Driver control registers and RAM variables will get initialized after this API is called.
- The API Spi_GetStatus will return the status of the SPI Driver.
- The API Spi_GetVersionInfo is invoked to get the version of the SPI Driver module with a variable of Std_VersionInfoType. After invoking this API, the parameter passed will get updated with the SPI Driver version details.
- The API Spi_WriteIB is invoked to write the data to the buffer for master.
- The API Spi_SetupEB is invoked to set up the external buffer for slave.
- The API Spi_SetAsyncMode will set the asynchronous mechanism mode to interrupt mode for SPI buses handled asynchronously.
- The API Spi_AsyncTransmit will prepare data for slave sequence before master triggers transmission.
- The API Spi_AsyncTransmit will transmit data on the SPI bus asynchronously.
- This module will take the parameter passed and set the SPI Driver status to SPI_BUSY. Also, it sets the sequence result to SPI_SEQ_PENDING, and first job result to SPI_JOB_PENDING and performs the transmission.
- The API Spi_ForceCancel will cancel sequence immediately without waiting for the completion of on-going job.
- The API Spi_GetHWUnitStatus will return the status of the specified SPI Hardware microcontroller peripheral.
- The API Spi_GetJobResult will return the status of the specified job ID.
- The API Spi_GetSequenceResult will return the status of the specified sequence ID.
- The API Spi_ReadIB is invoked to read data from internal buffer for master.
- The API Spi_WriteIB is invoked to write the data to the buffer for master.
- The API Spi_SetupEB is invoked to set up the external buffer for slave.
- The API Spi_SetAsyncMode will set the asynchronous mechanism mode to polling mode for SPI module. Transmit master and slave sequence again as in the interrupt mode.
- The API Spi_AsyncTransmit will prepare data for slave sequence before master triggers transmission.
- The API Spi_AsyncTransmit will transmit data on the SPI bus asynchronously.
- The API Spi_Cancel will cancel the specified on-going sequence transmission without canceling any job transmission, and the SPI Driver will set the sequence result to SPI_SEQ_CANCELED.
- The API Spi_MainFunction_Handling is used for Asynchronous transmission of the sequences in polling mode. This service should be invoked in a scheduler loop if the asynchronous transmission mode is selected as SPI_POLLING_MODE.

- The API Spi_DeInit is invoked to de-initialize all the controls registers and RAM variables.

V4H:

In the V4H Sample Application, all the SPI APIs are invoked in the following sequences,

- The API Spi_Init is invoked with a valid database address for the proper initialization of the SPI Driver, all the SPI Driver control registers and RAM variables will get initialized after this API is called.
- The API Spi_GetStatus will return the status of the SPI Driver.
- The API Spi_GetVersionInfo is invoked to get the version of the SPI Driver module with a variable of Std_VersionInfoType. After invoking this API, the parameter passed will get updated with the SPI Driver version details.
- The API Spi_WriteIB is invoked to write the data to the buffer for master.
- The API Spi_SetupEB is invoked to set up the external buffer for slave.
- The API Spi_SetAsyncMode will set the asynchronous mechanism mode to interrupt mode for SPI buses handled asynchronously. (Not Supported)
- The API Spi_AsyncTransmit will transmit data for master sequence.
- The API Spi_SyncTransmit will transmit data for slave sequence.
- This module will take the parameter passed and set the SPI Driver status to SPI_BUSY. Also, it sets the sequence result to SPI_SEQ_PENDING, and first job result to SPI_JOB_PENDING and performs the transmission.
- The API Spi_ReadIB is invoked to read data from internal buffer for master.
- The API Spi_GetJobResult will return the status of master job.
- The API Spi_GetJobResult will return the status of slave job.
- The API Spi_GetSequenceResult will return the status of master sequence.
- The API Spi_GetSequenceResult will return the status of slave sequence.
- The API Spi_GetHWUnitStatus will return the status of the specified SPI Hardware microcontroller peripheral.
- The API Spi_AsyncTransmit will transmit data for slave sequence.
- The API Spi_AsyncTransmit will transmit data for master sequence.
- The API Spi_ForceCancel will cancel slave sequence immediately without waiting for the completion of on-going job.
- The API Spi_Cancel will cancel the master sequence transmission without canceling any job transmission, and the SPI Driver will set the sequence result to SPI_SEQ_CANCELED.
- The API Spi_MainFunction_Handling is used for Asynchronous transmission of the sequences in polling mode. This service should be invoked in a scheduler loop if the asynchronous transmission mode is selected as SPI_POLLING_MODE.
- The API Spi_GetStatus will return the status of the SPI Driver.
- The API Spi_GetJobResult will return the status of master job.
- The API Spi_GetJobResult will return the status of slave job.
- The API Spi_GetSequenceResult will return the status of master sequence.
- The API Spi_GetSequenceResult will return the status of slave sequence.
- The API Spi_DeInit is invoked to de-initialize all the controls registers and RAM variables.

V4M:
In the V4M Sample Application, all the SPI APIs are invoked in the following sequences,

- The API Spi_Init is invoked with a valid database address for the proper initialization of the SPI Driver, all the SPI Driver control registers and RAM variables will get initialized after this API is called.
- The API Spi_GetStatus will return the status of the SPI Driver.
- The API Spi_GetVersionInfo is invoked to get the version of the SPI Driver module with a variable of Std_VersionInfoType. After invoking this API, the parameter passed will get updated with the SPI Driver version details.
- The API Spi_WriteIB is invoked to write the data to the buffer for master.
- The API Spi_SetupEB is invoked to set up the external buffer for slave.
- The API Spi_SetAsyncMode will set the asynchronous mechanism mode to interrupt mode for SPI buses handled asynchronously. (Not Supported)
- The API Spi_AsyncTransmit will transmit data for master sequence.
- The API Spi_SyncTransmit will transmit data for slave sequence.
- This module will take the parameter passed and set the SPI Driver status to SPI_BUSY. Also, it sets the sequence result to SPI_SEQ_PENDING, and first job result to SPI_JOB_PENDING and performs the transmission.
- The API Spi_ReadIB is invoked to read data from internal buffer for master.
- The API Spi_GetJobResult will return the status of master job.
- The API Spi_GetJobResult will return the status of slave job.
- The API Spi_GetSequenceResult will return the status of master sequence.
- The API Spi_GetSequenceResult will return the status of slave sequence.
- The API Spi_GetHWUnitStatus will return the status of the specified SPI Hardware microcontroller peripheral.
- The API Spi_AsyncTransmit will transmit data for slave sequence.
- The API Spi_AsyncTransmit will transmit data for master sequence.
- The API Spi_ForceCancel will cancel slave sequence immediately without waiting for the completion of on-going job.
- The API Spi_Cancel will cancel the master sequence transmission without canceling any job transmission, and the SPI Driver will set the sequence result to SPI_SEQ_CANCELED.
- The API Spi_MainFunction_Handling is used for Asynchronous transmission of the sequences in polling mode. This service should be invoked in a scheduler loop if the asynchronous transmission mode is selected as SPI_POLLING_MODE.
- The API Spi_GetStatus will return the status of the SPI Driver.
- The API Spi_GetJobResult will return the status of master job.
- The API Spi_GetJobResult will return the status of slave job.
- The API Spi_GetSequenceResult will return the status of master sequence.
- The API Spi_GetSequenceResult will return the status of slave sequence.
- The API Spi_DeInit is invoked to de-initialize all the controls registers and RAM variables.

(2) Multi-core Sample Application Structure (Not Supported)
This sample application shows that different HW unit can be used in the different core at the same time.

The configuration is shown as follows.

[PE0] – [Instance 0]
Instance 0 uses the following HW units.
- CSIH0, HSPI0

[PE1] – [Instance 1]
Instance 1 uses the following HW units.
- CSIH1

The following API of both instances call in PE0 sequentially.
- Spi_Init

The inter core exclusive control implements the following APIs.
- Spi_DeInit

The application behavior is the following flow,
- Initialize all HW units by calling Spi_Init of both instances in PE0.
- Set asynchronous mode to SPI_INTERRUPT_MODE by calling Spi_SetAsyncMode in each PE.
- Write data to buffer by calling Spi_WriteIB in PE0.
- Set up the external buffers by calling Spi_SetupEB in PE1.
- Start transmission using CSIH1 in slave mode by Spi_AsyncTransmit in PE1.
- Start transmission using CSIH0 in master mode by Spi_AsyncTransmit in PE0.
- Wait for end of transmission by while loop with Spi_GetSequenceResult in each PE.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Start transmission using HSPI0 in master mode by Spi_AsyncTransmit in PE0.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Wait for the completion of processing in each PE.
- De-initialize all HW units by calling Spi_DeInit in PE0.
- De-initialize all HW units by calling Spi_DeInit in PE1.

This sample application shows that different HW unit can be used in the different core at the same time.

The configuration is shown as follows.

[PE0] – [Instance 0]
Instance 0 uses the following HW units.
- MSPI0

[PE1] – [Instance 1]
Instance 1 uses the following HW units.
- MSPI1

The following API of both instances call in PE0 sequentially.
- Spi_Init

The inter core exclusive control implements the following API.
- Spi_DeInit

The application behavior is the following flow.
- Initialize all HW units by calling Spi_Init of both instances in PE0.
- Set asynchronous mode to SPI_INTERRUPT_MODE by calling Spi_SetAsyncMode in each PE.
- Write data to buffer by calling Spi_WriteIB in PE0.
- Set up the external buffers by calling Spi_SetupEB in PE1.
- Start transmission using MSPI1 in slave mode by Spi_AsyncTransmit in PE1.
- Start transmission using MSPI0 in master mode by Spi_AsyncTransmit in PE0.
- Wait for end of transmission by while loop with Spi_GetSequenceResult in each PE.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Start transmission using MSPI0 in master mode by Spi_AsyncTransmit in PE0.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Wait for the completion of processing in each PE.
- De-initialize all HW units by calling Spi_DeInit in PE0.
- De-initialize all HW units by calling Spi_DeInit in PE1.

This sample application shows that different HW unit can be used in the different core at the same time.

The configuration is shown as follows.

[PE0] – [Instance 0]
Instance 0 uses the following HW units.
- MSPI0

[PE1] – [Instance 1]
Instance 1 uses the following HW units.
- MSPI1

The following API of both instances call in PE0 sequentially.
- Spi_Init

The inter core exclusive control implements the following API.
- Spi_DeInit

The application behavior is the following flow.
- Initialize all HW units by calling Spi_Init of both instances in PE0.
- Set asynchronous mode to SPI_INTERRUPT_MODE by calling Spi_SetAsyncMode in each PE.
- Write data to buffer by calling Spi_WriteIB in PE0.
- Set up the external buffers by calling Spi_SetupEB in PE1.
- Start transmission using MSPI1 in slave mode by Spi_AsyncTransmit in PE1.
- Start transmission using MSPI0 in master mode by Spi_AsyncTransmit in PE0.
- Wait for end of transmission by while loop with Spi_GetSequenceResult in each PE.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Start transmission using MSPI0 in master mode by Spi_AsyncTransmit in PE0.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Wait for the completion of processing in each PE.
- De-initialize all HW units by calling Spi_DeInit in PE0.
- De-initialize all HW units by calling Spi_DeInit in PE1.

This sample application shows that different HW unit can be used in the different core at the same time.

The configuration is shown as follows.

[PE0] – [Instance 0]
Instance 0 uses the following HW units.
- MSIOF1

[PE1] – [Instance 1]
Instance 1 uses the following HW units.
- MSIOF2

The following API of both instances call in PE0 sequentially.
- Spi_Init

The inter core exclusive control implements the following API.
- Spi_DeInit

The application behavior is the following flow,
- Initialize all HW units by calling Spi_Init of both instances in PE0.
- Set asynchronous mode to SPI_INTERRUPT_MODE by calling Spi_SetAsyncMode in each PE. (Not

Supported)

- Write data to buffer by calling Spi_WriteIB in PE0.
- Set up the external buffers by calling Spi_SetupEB in PE1.
- Start transmission using MSIOF2 in slave mode by Spi_AsyncTransmit in PE1. (Not Supported)
- Start transmission using MSIOF1 in master mode by Spi_AsyncTransmit in PE0.
- Wait for end of transmission by while loop with Spi_GetSequenceResult in each PE.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Start transmission using MSIOF1 in master mode by Spi_AsyncTransmit in PE0.
- Read data from buffer by calling Spi_ReadIB in PE0.
- Wait for the completion of processing in each PE.
- De-initialize all HW units by calling Spi_DeInit in PE0.
- De-initialize all HW units by calling Spi_DeInit in PE1.

3.3.8.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-75 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

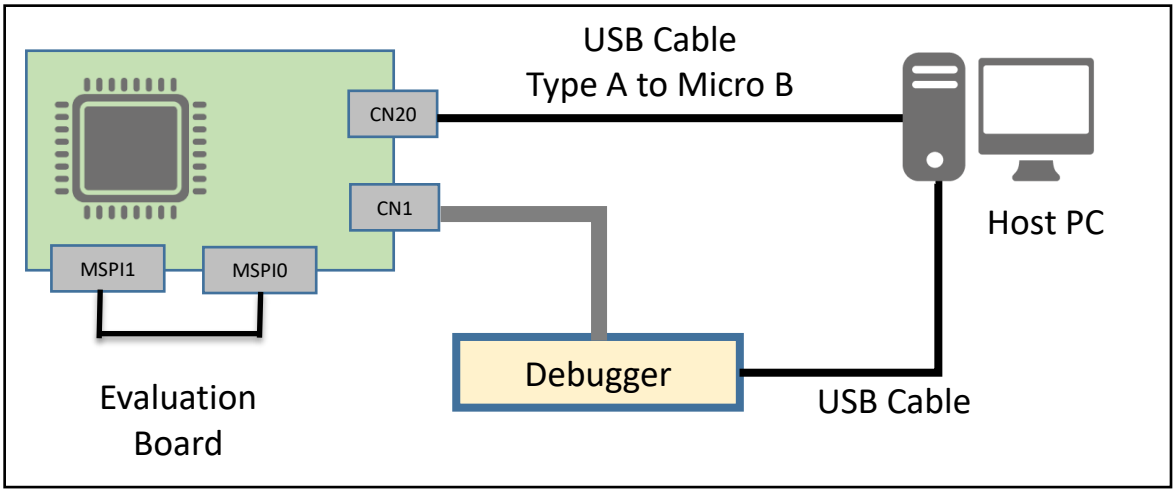

Figure 3-26 S4 Spider board connection of SPI (S4_G4MH) Sample Application

- **V4H Environment**

Table 3-76 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

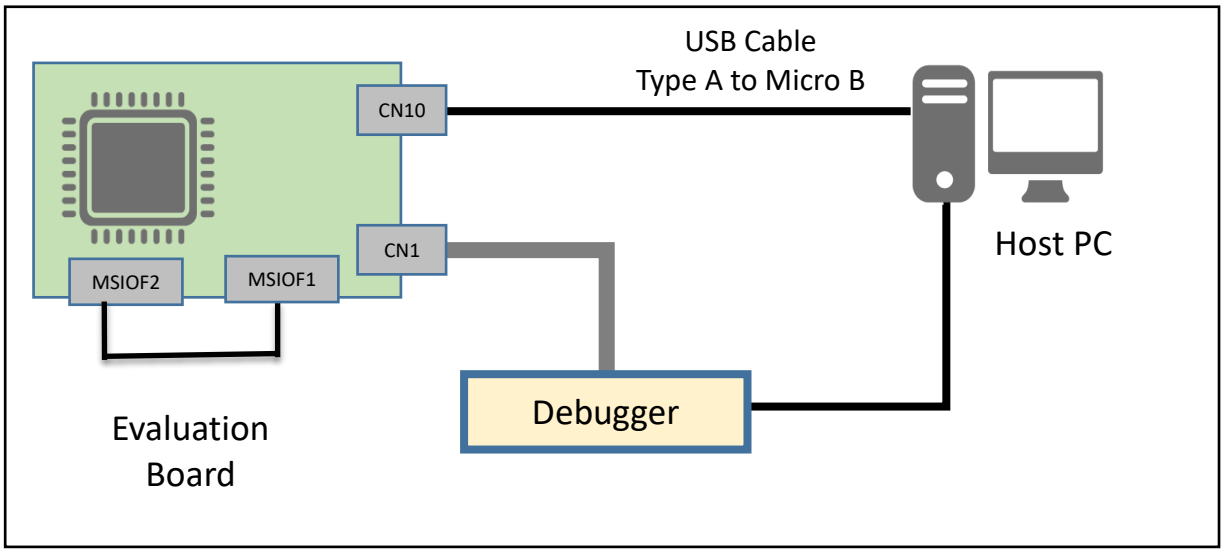

Figure 3-27 V4H White Hawk board connection of SPI Sample Application

- **V4M Environment**

Table 3-77 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

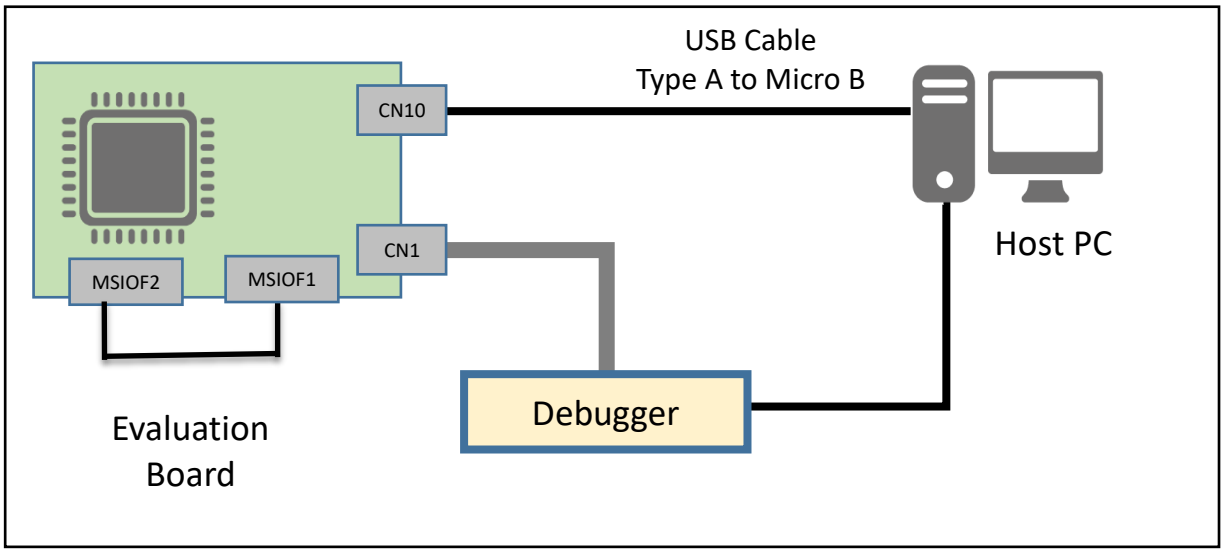

Figure 3-28 V4M Gray Hawk board connection of SPI Sample Application

3.3.8.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.
- Additional pin setting for V4H/V4M SPI Sample Application from step7 at 3.7.3.1.3 Flash bootloaders as the following,
    - Set SW02 Pin5, Pin6 as OFF
    - Set 3rd side for SW10, SW12
    - Hardware connection: MSIOF1 (Master) ⇔ MSIOF2 (Slave)

Pin connection	Description
CN5_A3 ⇔ CN6_A3	MSIOF1_SCK (P1_3) ⇔ MSIOF2_SCK (P0_17)
CN5_A4 ⇔ CN6_A4	MSIOF1_SYNC (P1_2) ⇔ MSIOF2_SYNC (P0_15)
CN5_A6 ⇔ CN6_A5	MSIOF1_TXD (P1_4) ⇔ MSIOF2_RXD (P0_18)
CN5_A5 ⇔ CN6_A6	MSIOF1_RXD (P1_5) ⇔ MSIOF2_TXD (P0_16)

3.3.8.9.4 How to build Sample Application

Refer to 3.7.2.2 How to build the Sample Application

[R-Car S4_G4MH]

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:
  external\rel\S4\common_family\make\ghs
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat Spi R19-11 S4 No

[R-Car V4H]

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:
  external\rel\V4H\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat Spi R19-11 V4H No

[R-Car V4M]

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:
  external\rel\V4M\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat Spi R19-11 V4M No

3.3.8.9.5 How to run Sample Application

Step 1: Set BreakPoint at each function which needs to check and then run step over that BreakPoint or set BreakPoint at the end of the main function.

Step 2: Run program to breakpoint, confirm all elements in the array App_GaaTestResult [0..n] are set to 1 or check by print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.3.8.10 ROM/RAM Usage

See Appendix SPI section for information on measuring RAM/ROM consumption.

3.3.8.11 Stack Depth

See Appendix SPI section for information on measuring stack depth.

3.3.8.12 Throughput Details

See Appendix SPI section for information on measuring execution time, and functional testing.

3.3.9 WDG Driver Component

3.3.9.1 Module Overview

To minimize the effort and to optimize the reuse of the software developed, the Watchdog interface will invoke the corresponding drivers when multiple drivers exist.
If more than one Watchdog device and Watchdog Driver (both internal software Watchdog and external hardware Watchdog) are used on an ECU, Watchdog Interface module allows the upper layer to select the correct Watchdog Driver and Watchdog device while retaining the API and functionality of the underlying driver.

The Watchdog Driver accesses the microcontroller hardware directly and interface communicates with the application.

The Watchdog Driver component is composed of the following modules:

- Watchdog Driver Initialization module
- Watchdog Driver SetMode module
- Watchdog Driver Trigger module
- Watchdog Driver Version info module

3.3.9.2 Module Dependency

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

MCU
The count indicating the number of times that the watchdog should be triggered for a trigger condition’s timeout value depends on McuRCLKClk. Hence MCU reference path will be provided in the parameter definition file.
The count indicating the number of times that the watchdog should be triggered for a trigger condition’s timeout value depends on McuWDTClk. Hence MCU reference path will be provided in the parameter definition file.

OS
As the WDG Driver uses interrupts, it depends on the OS which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

GPT
WDG Component uses GPT Call Back Notification functions to trigger the WDG hardware counter.

3.3.9.3 **Folder Structure**

Table 3-78 and **Table 3-79** show the list of Source Code Files and Header Files for WDG module.

Table 3-78 WDG Header Files

Location: rel\modules\wdg\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Wdg.h	-	x	x	x	Product	This file provides extern declarations for all WDG Component APIs. It provides service IDs of APIs, DET Error codes and type definitions for Watchdog Driver initialization structure. This header file shall be included in other modules to use the features of WDG Component.
Wdg_MultiInstance.h	-	x	x	x	Product	This file contains the definitions for multi-instance.
Wdg_PBTypes.h	-	x	x	x	Product	This file contains the macros used internally by the structure declarations related to watchdog control registers.
Wdg_Ram.h	-	x	x	x	Product	This file contains the extern declarations for the global variables defined in Wdg_Ram.c file and the version information of the file.
Wdg_Types.h	-	x	x	x	Product	This file contains the common macro definitions and the data types required internally by the WDG software component.
Wdg_Version.h	-	x	x	x	Product	This file contains the macros of AUTOSAR version numbers of all modules interfaced to WDG.
WDTB	-	-	-	-	-	-
Wdg_WDTB_Irq.h	-	x	-	-	Product	This file contains the implementation of all the interrupt functions used by WDG Driver Component.
Wdg_WDTB_LLDriver.h	-	x	-	-	Product	This file contains the definition of the internal functions that access the hardware registers.
Wdg_WDTB_PBTypes.h	-	x	-	-	Product	This file contains the macros used internally by the WDG Component code and the structure declarations related to watchdog control registers for WDTB.
RWDT	-	-	-	-	-	-
Wdg_RWDT_LLDriver.h	-	-	x	x	Product	This file contains the definition of the internal functions that access the hardware registers for RWDT.
Wdg_RWDT_PBTypes.h	-	-	x	x	Product	This file contains the macros used internally by the WDG Component code and the structure declarations related to watchdog control registers for RWDT.

x: applicable

-: not applicable

Table 3-79 WDG Source Files

Location: rel\modules\wdg\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Wdg.c	-	x	x	x	Product	This file contains the implementation of all APIs.
Wdg_Ram.c	-	x	x	x	Product	This file contains the global variables used by WDG Component.
Wdg_Version.c	-	x	x	x	Product	This file contains the code to check the version of all modules interfaced to WDG.
WDTB	-	-	-	-	-	-
Wdg_WDTB_Irq.c	-	x	-	-	Product	This file contains the implementation of all the interrupt functions used by WDG Driver Component.
Wdg_WDTB_LLDriver.c	-	x	-	-	Product	This file contains the definition of the internal functions that access the hardware registers.
RWDT	-	-	-	-	-	-
Wdg_RWTD_LLDriver.c	-	-	x	x	Product	This file contains the definition of the internal functions that access the hardware registers for RWDT.

x: applicable

-: not applicable

Table 3-80 shows the list of Parameter Definition Files for WDG module.

Table 3-80 WDG Parameter Definition Files

Location: external\rel\modules\wdg\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4_G4MH	R1911_WDG_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R1911_WDG_V4H.arxml	V4H
	V4M	R1911_WDG_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.9.4 Configuration Parameter Dependency

Table 3-81 shows the list of configuration parameter dependency for WDG Driver component.

Table 3-81 V4H WDG Driver Component Configuration Parameter Dependency

Parameter	Module	Path
WdgClockRef	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuRCLKClk
WdgExternalContainerRef	DIO	/AUTOSAR/EcucDefs/Dio/DioConfig/DioPort/DioChannelGroup
	SPI	/AUTOSAR/EcucDefs/Spi/SpiDriver/SpiSequence
WDG_E_DISABLE_REJECTED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WDG_E_MODE_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WDG_E_WRITE_REGISTER_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WDG_E_VALUE_COUNTER_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WdgGptContainerRef	GPT	/Renesas/EcucDefs_Gpt/Gpt/GptChannelConfigSet/GptChannelConfiguration

Table 3-82 shows the list of configuration parameter dependency for WDG Driver component.

Table 3-82 S4 G4MH WDG Driver Component Configuration Parameter Dependency

Parameter	Module	Path
WdgClockRef	MCU	S4_G4MH: /Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuWDTClk /Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuWDTClkA
WdgExternalContainerRef	DIO	/AUTOSAR/EcucDefs/Dio/DioConfig/DioPort/DioChannelGroup
	SPI	/AUTOSAR/EcucDefs/Spi/SpiDriver/SpiSequence
WDG_E_DISABLE_REJECTED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WDG_E_MODE_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WDG_E_ECM_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WDG_E_ILLEGAL_UPDATE_REGISTER	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
WDG_E_TRIGGER_TIMEOUT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

Table 3-81 shows the list of configuration parameter dependency for WDG Driver component.

Table 3-83 V4M WDG Driver Component Configuration Parameter Dependency

Parameter	Module	Path
WdgClockRef	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuRCLKClk
WdgExternalContainerRef	DIO	/AUTOSAR/EcucDefs/Dio/DioConfig/DioPort/DioChannelGroup
	SPI	/AUTOSAR/EcucDefs/Spi/SpiDriver/SpiSequence
WDG_E_DISABLE_REJECTED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.3.9.5 **Source Code Dependency**

The followings are the dependency files commonly used by the WDG Driver module:

- Det.h,
- Dem.h,
- rh850_Types.h,
- Std_Types.h,
- Wdg_MemMap.h,
- WdgIf_Types.h,
- SchM_Wdg.h,
- Rte.h,
- Os.h

3.3.9.6 **Stubs**

Refer to 3.2.9 for the common stubs used for WDG Driver component.

3.3.9.7 **Addition Error Handling**

Refer to " Development and Production Errors" section at WDG chapter in Driver Component Embedded User's Manual.

3.3.9.8 **Restrictions**

None

3.3.9.9 **Sample Application**

3.3.9.9.1 **Sample Application Structure**

Refer to 3.7 Sample Application.

(1) Single-core sample application structure

- The API Wdg_GetVersionInfo is invoked to get the version of the WDG Driver module with a variable of Std_VersionInfoType. After this API is called, the passed parameter gets updated with the WDG Driver version details.
- The API Wdg_Init is invoked with a valid database address for the proper initialization of the WDG Driver, and all the WDG Driver control registers and RAM variables get initialized after this API is called.
- The API Wdg_SetMode is invoked with the mode that needs to be set. This API changes the mode of the Watchdog.
- The API Wdg_SetTriggerCondition initializes the trigger counter global variable with timeout value divided by either ulSlowTimeValue or ulFastTimeValue based on the current mode of Watchdog'.

(2) Multi-core sample application structure (Not Supported)

CPU 0:

- The API Wdg_Init_0 and Wdg_Init_1 is invoked with a valid database address for the proper initialization of the WDG Driver, all the WDG Driver control registers and RAM variables will get initialized after this API is called.
- The API Wdg_GetVersionInfo_0 is invoked to get the version of the WDG Driver module with a variable of Std_VersionInfoType, after the call of this API the past parameter will get updated with the WDG Driver version details.
- The API Wdg_SetMode_0 is invoked with the mode which needs to be set, this API changes the mode of the Watchdog.
- The API Wdg_SetTriggerCondition_0 initializes the trigger counter global variable with timeout value divided by either usSlowTimeValue or ulFastTimeValue based on the current mode of Watchdog'.

CPU 1:

- The API Wdg_GetVersionInfo_1 is invoked to get the version of the WDG Driver module with a variable of Std_VersionInfoType, after the call of this API the past parameter will get updated with the WDG Driver version details.

- The API Wdg_SetMode_1 is invoked with the mode which needs to be set, this API changes the mode of the Watchdog.
- The API Wdg_SetTriggerCondition_1 initializes the trigger counter global variable with timeout value divided by either ulSlowTimeValue or ulFastTimeValue based on the current mode of Watchdog'.

3.3.9.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-84 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

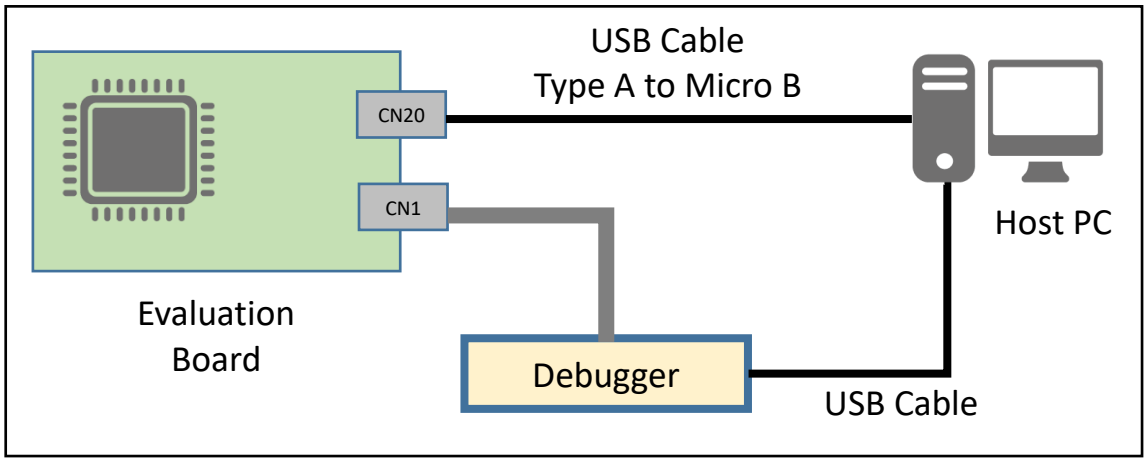

Figure 3-29 S4 Spider board connection of WDG (S4_G4MH) Sample Application

- **V4H Environment**

Table 3-85 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

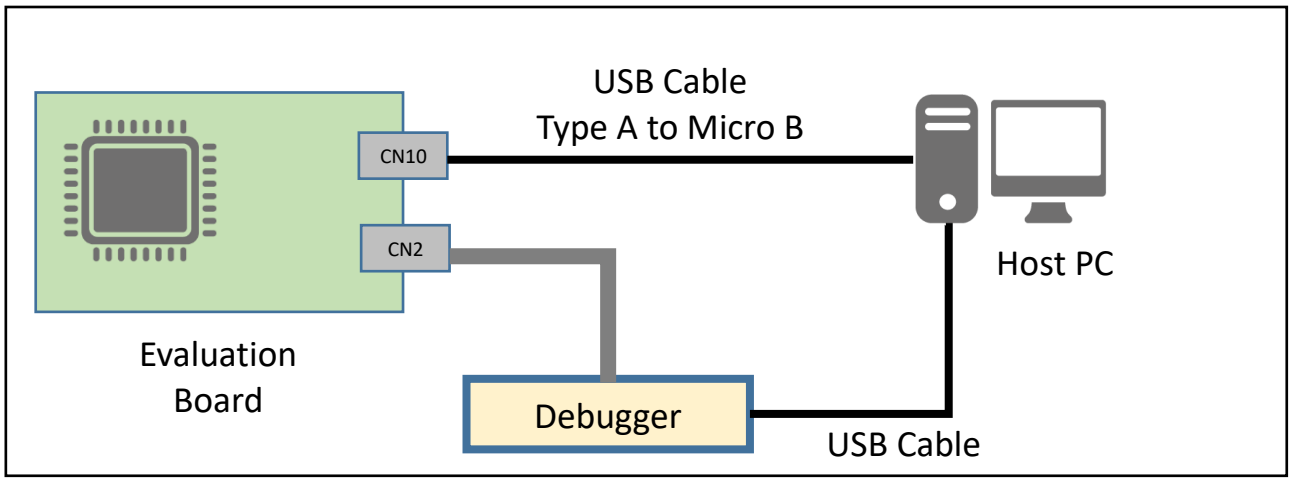

Figure 3-30 V4H White Hawk board connection of WDG (V4H) Sample Application

- **V4M Environment**

Table 3-86 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

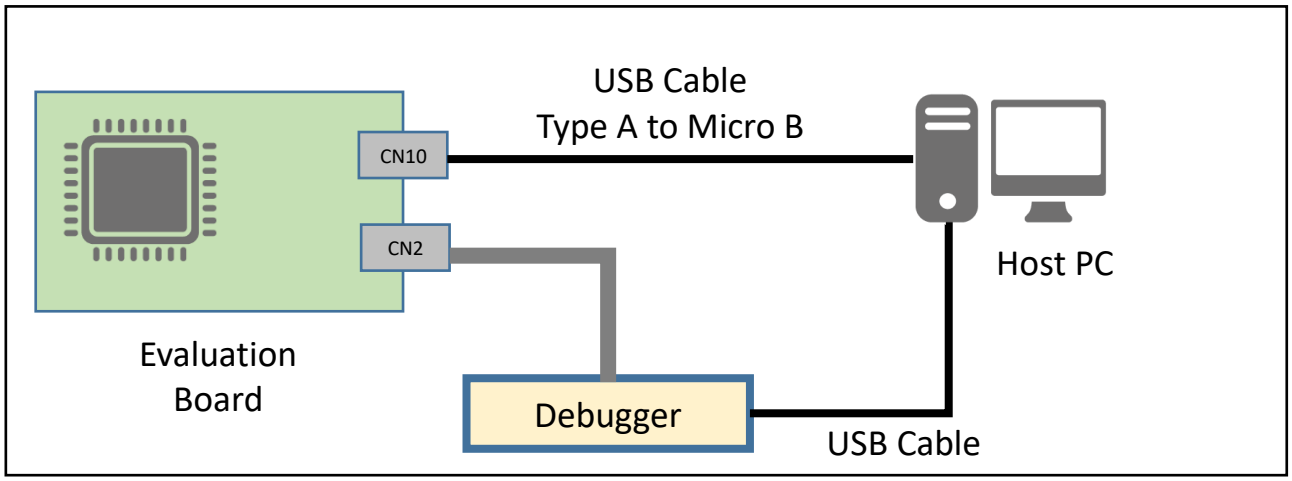

Figure 3-31 V4M Gray Hawk board connection of WDG (V4M) Sample Application

3.3.9.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.9.9.4**How to build the sample application**

Refer to 3.7.2.2 How to build the Sample Application

3.3.9.9.5**How to run sample application**

Step 1: Set BreakPoint at each function which needs to check and then run step over that BreakPoint or set BreakPoint at the end of the main function.

Step 2: Run program to breakpoint, confirm all elements in the array GaaTestResult [0..n] are set to 1 or check by print logs throught the Teraterm if it is used:

"PROGRAM START": Sample application execution is started.

"EXECUTED OK": Sample application execution is successful.

"EXECUTED NOT OK": Sample application execution is failed.

"PROGRAM STOP": Sample application execution is completed.

3.3.9.10 **ROM/RAM Usage**

See Appendix WDG section for information on measuring RAM/ROM consumption.

3.3.9.11 **Stack Depth**

See Appendix WDG section for information on measuring stack depth.

3.3.9.12 **Throughput Details**

See Appendix WDG section for information on measuring execution time, and functional testing.

3.3.10 CAN Driver Component

3.3.10.1 Module Overview

The CAN Driver is part of the microcontroller abstraction layer (MCAL), which performs the hardware access and offers hardware-independent API to the upper layer. The only upper that can access 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 shall provide the following main features:

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.

3.3.10.2 Module Dependency

The dependency of CAN Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

MCU
CAN Driver depends on the MCU Driver for setting the channel clock.

CAN Interface
The CAN Driver component provides the following functionalities to the CAN Interface layer.

- To change the operation mode of the controllers.
- To enable/disable the Controller Interrupts
- To process the L-PDU Transmission

EcuM
If controller wake-up event is detected, CAN Driver component provides the callout notification functionality to the EcuM.

OS
As the CAN Driver uses interrupts, it depends on the OS, which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

DEM
The Diagnostic Event manager (DEM) will be called whenever CAN module encounters a production relevant error.

3.3.10.3 **Folder Structure**

Table 3-87 and **Table 3-88** show the list of Source Code Files and Header Files for CAN module.

Table 3-87 CAN Header Files

Location: rel\modules\can\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Can.h	x	x	x	x	Product	This file provides extern declaration, type definition and config pointer for global as well as controller initialization of CAN Driver Component.
Can_CommonRegStruct.h	x	x	x	x	Product	This file provides the Controller register structure.
Can_Irq.h	x	x	x	x	Product	C header file for Can_Irq.c. It includes the header files required for Can_Irq.c file.
Can_LTTypes.h	x	x	x	x	Product	This file contains the data structures for AUTOSAR CAN Pre-Compile time parameters which have instances.
Can_MainServ.h	x	x	x	x	Product	C header file for Can_MainServ.c. It includes the header files required for Can_MainServ.c file and extern declaration of internal functions of CAN Driver Component.
Can_ModeCntrl.h	x	x	x	x	Product	C header file for Can_ModeCntrl.c. It includes the header files required for Can_ModeCntrl.c file and extern declaration of internal functions of CAN Driver Component.
Can_PBTypes.h	x	x	x	x	Product	This file contains the data structures for AUTOSAR CAN Post-Build time parameters.
Can_Ram.h	x	x	x	x	Product	C header file for Can_Ram.c. It includes the header files required for Can_Ram.c file.
Can_Version.h	x	x	x	x	Product	C header file for Can_Version.c. It includes the header files required for CanVersion.c file.
Can_Icom.h	x	x	x	x	Product	C header file for Can_Icom.c. It includes the header files required for Can_Icom.c file.
Can_TSCapture.h	x	x	-	-	Product	C header file for Can_TSCapture.c. It includes the header files required for Can_TSCapture.c file.
S4	-	-	-	-	-	-
Can_RegStruct.h	x	x	-	-	Product	This file contains the data structure for Controller registers and macros to access registers.
V4H	-	-	-	-	-	-
Can_RegStruct.h	-	-	x	-	Product	This file contains the data structure for Controller registers and macros to access registers.

V4M	-	-	-	-	-	-
Can_RegStruct.h	-	-	-	x	Product	This file contains the data structure for Controller registers and macros to access registers.

x: applicable

-: not applicable

Table 3-88 CAN Source Files

Location: rel\modules\can	**Supported Device**				**Category**	**Description**
	S4 (Includes S4N) CR52	**S4 (Includes S4N) G4MH**	**V4H**	**V4M**		
src	-	-	-	-	-	-
Can.c	x	x	x	x	Product	This file contains the functions that initialize the CAN Driver component and controller-specific settings. This file also contains the function to get the version information of CAN Driver Component.
Can_Irq.c	x	x	x	x	Product	This file contains the interrupt service routine functionality.
Can_MainServ.c	x	x	x	x	Product	This file contains the main service routine functionality.
Can_ModeCntrl.c	x	x	x	x	Product	This file contains the Controller mode control functionality.
Can_Ram.c	x	x	x	x	Product	This file contains the global variable used by the CAN Driver component.
Can_Version.c	x	x	x	x	Product	The file contains the version check of other modules that interacts with CAN Driver component.
Can_Write.c	x	x	x	x	Product	This file contains the function that handles the Transmission of L-PDU(s).
Can_Icom.c	x	x	x	x	Product	This file contains the initialization, re-initialization and version control functionality for Pretended Networking function.
Can_RamTest.c	x	x	x	x	Product	This file contains the function that handles the Can Ram Test Api
Can_TSCapture.c	x	x	-	-	Product	This file contains Internal Function of Can Time Sync.

x: applicable

-: not applicable

Table 3-89 shows the list of Parameter Definition Files for CAN module.

Table 3-89 CAN Parameter Definition Files

Location: rel\modules\can\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4_G4MH	R1911_CAN_S4_RTM8RC79FG.arxml	RTM8RC79FG
	S4_CR52	R1911_CAN_S4_RTM8RC79FR.arxml	RTM8RC79FR
	V4H	R1911_CAN_V4H.arxml	V4H
	V4M	R1911_CAN_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.10.4 **Configuration Parameter Dependency**

- S4_G4MH

Table 3-90 shows the list of configuration parameter dependency for CAN Driver component.

Table 3-90 CAN Driver Component Configuration Parameter Dependency

Parameter	Module	Path
CanCpuClockRef	MCU	/AUTOSAR/EcucDefs/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockReferencePoint
CanControllerPclkClock	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuHighSpeedPeriClk
CanControllerPplClock	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuLowSpeedPeriClk
CAN_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CAN_E_TIMEOUT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CanWakeupSourceRef	ECUM	/AUTOSAR/EcucDefs/EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource
CanIcomSignalRef	COM	/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal
CanOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter
CanSupportTTCANRef	CANIF	/AUTOSAR/EcucDefs/CanIf/CanIfPrivateCfg

- V4H

Table 3-91 shows the list of configuration parameter dependency for CAN Driver component.

Table 3-91 CAN Driver Component Configuration Parameter Dependency

Parameter	Module	Path
CanCpuClockRef	MCU	/AUTOSAR/EcucDefs/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockReferencePoint
CanControllerPclkClock	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuSASYNCPERD2Clk
CanControllerPplClock	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuCANFDClk
CAN_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CAN_E_TIMEOUT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CanIcomSignalRef	COM	/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal
CanOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter
CanSupportTTCANRef	CANIF	/AUTOSAR/EcucDefs/CanIf/CanIfPrivateCfg

- V4M

Table 3-91 shows the list of configuration parameter dependency for CAN Driver component.

Table 3-92 CAN Driver Component Configuration Parameter Dependency

Parameter	Module	Path
CanCpuClockRef	MCU	/AUTOSAR/EcucDefs/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockReferencePoint
CanControllerPclkClock	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuSASYNCPERD2Clk
CanControllerPplClock	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuCANFDClk
CAN_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CAN_E_TIMEOUT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CanIcomSignalRef	COM	/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal
CanOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter
CanSupportTTCANRef	CANIF	/AUTOSAR/EcucDefs/CanIf/CanIfPrivateCfg

3.3.10.5 Source Code Dependency

The followings are the dependency files commonly used by the CAN Driver module:

- Can_GeneralTypes.h,
- ComStack_Types.h,
- Can_MemMap.h,
- Os.h,
- EcuM_Cbk.h,
- Dem.h,
- SchM_Can.h,
- Det.h,
- rh850_Types.h,
- CanIf_Cbk.h,
- EcuM_Cfg.h
- EcuM_Types.h
- EcuM.h,
- Rte.h,
- CanIf.h
- Rte_Os_Type.h
- Rte_Dem_Type.h
- CanIf_Can.h

3.3.10.6 Stubs

Refer to 3.2.9 for the common stubs used for CAN Driver component.

3.3.10.7 Addition Error Handling

Refer to " Development and Production Errors" section at CAN chapter in Driver Component Embedded User's Manual.

3.3.10.8 Restrictions

None

3.3.10.9 Sample Application

3.3.10.9.1 Sample Application Structure

In the Sample Application, all the CAN APIs are invoked in the following sequences.

- All the CAN Driver control registers and RAM variables get initialized when Can_Init is invoked with a valid database address.
- The API Can_SetControllerMode is invoked to set controller mode to start mode.
- The API Can_GetVersionInfo is invoked to get the version of the CAN Driver module with a variable of Std_VersionInfoType, after this API is called, the past parameter get updated with the CAN Driver version details.
- Can_Write is invoked to write the message buffer with id 256 and data as 0x02, 0x04, 0x06, 0x08, 0x01, 0x03, 0x05, 0x07, which in turn results in transmission of the same.
- Can_SetControllerMode is invoked with CAN_CS_STOPPED as arguments to show how the API should be used.
- Invoke Can_GetControllerMode to get the CAN_CS_STOPPED status, after that, set the controller mode to start mode and invoke Can_Write to confirm the start mode.
- Can_Write is invoked to write the message buffer with id 100 and data as 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, which in turn results in transmission of the same.
- Can_DisableControllerInterrupts and Can_EnableControllerInterrupts are invoked in succession with comments in between. Critical section code should come in between.
- Invoke Can_DeInit to de-initialize CAN Driver. After that, re-initialize CAN Driver, then, invoke Can_Write to confirm the de-initialization.
- Invoking Can_RAMTest() for testing one RAM page RSCAN(FD) unit 0
- Invoke Can_SelfTestChannel to transit Controller 0 to Self-Test Internal, External Loop back mode and perform Self-Test on controller 0.

3.3.10.9.2**Recommended Environment**

- **S4 G4MH Environment**

Table 3-93 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
CAN Network Interface + Analyzer	CANoe/CANalyzer 10.0 + VN8912 CAN Cable (D-Sub 9pin)
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

- **S4 CR52 Environment**

Table 3-94 S4 CR52 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
CAN Network Interface + Analyzer	CANoe/CANalyzer 10.0 + VN8912 CAN Cable (D-Sub 9pin)
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

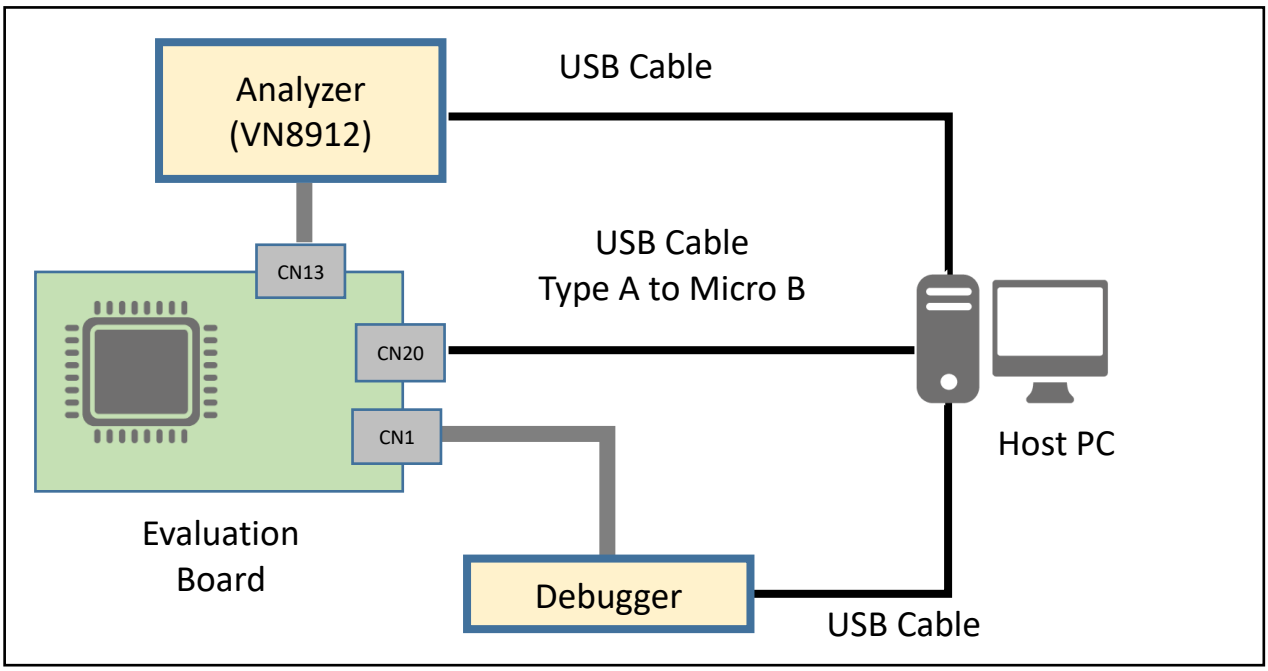

Figure 3-32 S4 Spider board connection of CAN Sample Application

Note: In case of R-Car S4 B0(2nd) evaluation board, please use “Mode Switch Board” instead of “switch on CPU Board/CPLD”

V4H Environment

Table 3-95 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

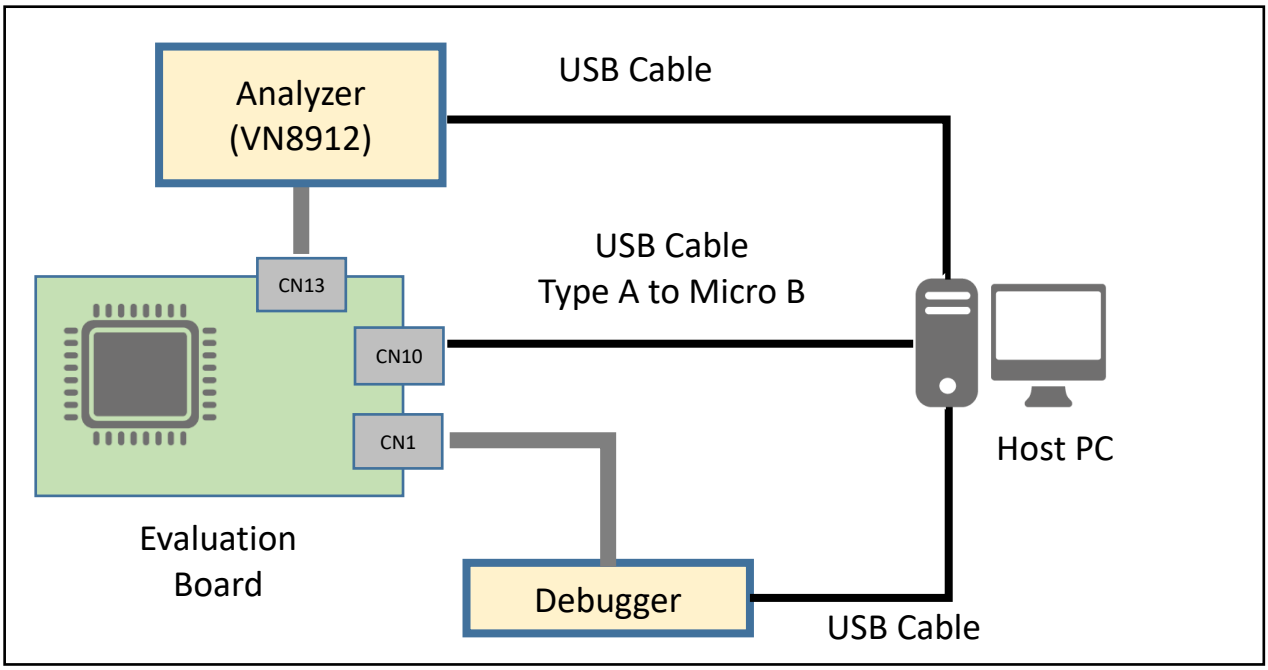

Figure 3-33 V4H White Hawk board connection of CAN (V4H) Sample Application

Note: When using CAN channel 0, 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.

- **V4M Environment**

Table 3-96 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

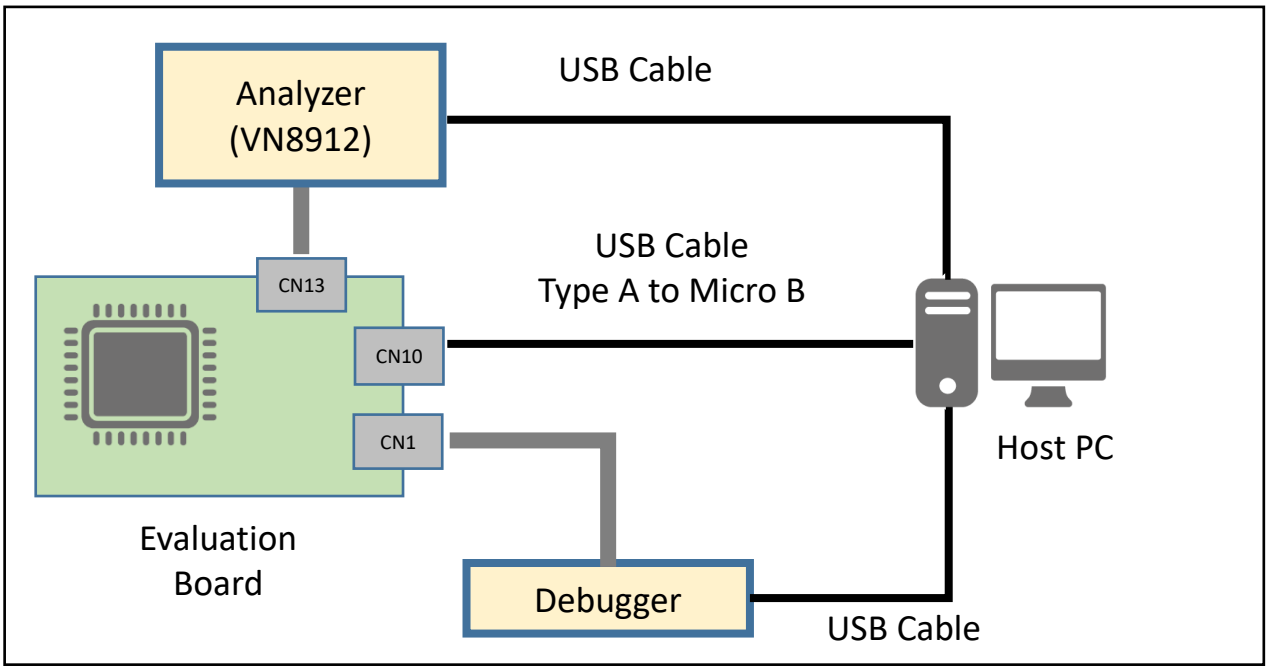

Figure 3-34 V4M Gray Hawk board connection of CAN (V4M) Sample Application

Note: When using CAN channel 0, port pin GP1_3 should be set as GPIO Output mode with HIGH Pin level value to enable the channel 0 transceiver on V4M Gray Hawk board.

3.3.10.9.3 Preparation

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.3.10.9.4 How to build Sample Application

Refer to 3.7.2.2 How to build the Sample Application

3.3.10.9.5 How to run Sample Application

1. At beginning of the program, confirm the global variable GucVerCheckStatus is TRUE for Compatible Version of software version.
2. Set breakpoint after each call off Can_SetControllerMode API.
3. After each calling the Can_SetControllerMode API, check if GucReturnCount is increased one.
4. Set breakpoint after each call off Can_Write API.

5. Each time call the Can_Write API, make sure CAN frame is transmitted to Testing Device (e.g Vector Hardware Device) and the Global variable GucReturnCount is increased one.
6. Set breakpoint after each call off Can_MainFunction_Read.
7. After each calling Can_MainFunction_Read API if Testing Device send back CAN frame, checking if the CAN frame is received on Development board with the CanIf_RxIndication callback function is called and Global variable GucReturnCountthe is increased one.
8. Continue to run program until “End trap” sample_end() is reached. Sample application sequence is completed.

Note: The sample application execution and result can be check by print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.3.10.10 **ROM/RAM Usage**

See Appendix CAN section for information on measuring RAM/ROM consumption.

3.3.10.11 **Stack Depth**

See Appendix CAN section for information on measuring stack depth.

3.3.10.12 **Throughput Details**

See Appendix CAN section for information on measuring execution time, and functional testing.

3.3.11 LIN Driver Component

3.3.11.1 Module Overview

The LIN Driver is part of the microcontroller abstraction layer (MCAL), which performs the hardware access and offers hardware-independent API to the upper layer. Several LIN Controllers are controlled by the LIN Driver as long as they belong to the same LIN Hardware Unit.

The LIN Driver component shall provide the following main features:

The LIN Driver component fulfills requirements of upper layer communication components with respect to Initialization, Transmit and Receive confirmation and Wakeup notification to ECU State Manager.

3.3.11.2 Module Dependency

The dependency of LIN Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
The Diagnostic Event manager (DEM) will be called whenever LIN module encounters a production relevant error.

MCU
LIN Driver depends on MCU Driver for setting the channel clock.

EcuM
If controller wake-up event is detected, LIN Driver component provides the callout notification functionality to the EcuM.

OS
As the LIN Driver uses interrupts, it depends on the OS, which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

LIN Interface
If controller wake-up event is detected, LIN Driver component provides the callout notification functionality to the LIN Interface (LinIf).

3.3.11.3 **Folder Structure**

Table 3-97 and **Table 3-98** show the list of Source Code Files and Header Files for LIN module.

Table 3-97 LIN Header Files

Location: rel\modules\lin\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
include	-	-	-	-	-
Lin.h	-	x	-	Product	This file provides extern declaration for all LIN Driver APIs, service IDs for the APIs and Type definitions.
Lin_PBTypes.h	-	x	-	Product	The structures related to channel information, hardware and interrupt control registers and type definitions are defined in this file.
Lin_Ram.h	-	x	-	Product	C header file for Lin_Ram c. It includes the header files required for Lin_Ram.c file.
Lin_Types.h	-	x	-	Product	This file contains provision of global variables for debugging purpose.
Lin_Version.h	-	x	-	Product	C header file for Lin_Version.c. It includes the header files required for Lin_Version.c file.
RLIN3	-		-	-	-
Lin_RLIN3_Irq.h	-	x	-	Product	C header file for Lin_RLIN3_Irq.c. This file provides extern declaration for all ISRs.
Lin_RLIN3_LLDriver.h	-	x	-	Product	C header file for Lin_RLIN3_LLDriver.c. This file provides extern declaration for all internal functions of LIN Driver.
Lin_RLIN3_PBTypes.h	-	x	-	Product	The structures related to channel information, hardware and interrupt control registers and type definitions are defined in this file.

x: applicable

-: not applicable

Table 3-98 LIN Source Files

Location: rel\modules\lin\	Supported Device			Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H		
src	-	-	-	-	-
Lin.c	-	x	-	Product	This file contains the global and channel-specific initialization function, de-initialization functionality, sleep and wake-up functionality, status information, Tx and Rx functionality. It also contains the function to get the version information of LIN Driver Component.
Lin_Ram.c	-	x	-	Product	This file contains the global variable used by the LIN Driver Component.
Lin_Version.c	-	x	-	Product	This file provides Version information of various modules used in the Lin Driver Component.
RLIN3	-		-	-	-
Lin_RLIN3_Irq.c	-	x	-	Product	This file contains the interrupt service routine functionality.
Lin_RLIN3_LLDriver.c	-	x	-	Product	This file contains the internal functions provided by LIN Driver Component.

x :applicable

-: not applicable

Table 3-99 shows the list of Parameter Definition Files for LIN module.

Table 3-99 LIN Parameter Definition Files

Location: rel\modules\lin\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4	R1911_LIN_S4_RTM8RC79FG.arxml	RTM8RC79FG

Note Product Name**:** Product Names supported by “Files”.

3.3.11.4 **Configuration Parameter Dependency**

Table 3-100 shows the list of configuration parameter dependency for LIN Driver component.

Table 3-100 LIN Driver Component Configuration Parameter Dependency

Parameter	Module	Path
LinClockRef	MCU	/AUTOSAR/EcucDefs/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuClockReferencePoint
LinChannelClockRef	MCU	/Renesas/EcucDefs_Mcu/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuModuleClockSetting/McuRLINClk
LIN_E_TIMEOUT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
LIN_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
LIN_E_TIMEOUT_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
LinChannelEcuMWakeupSource	ECUM	/AUTOSAR/EcucDefs/EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMWakeupSource

3.3.11.5 **Source Code Dependency**

The followings are the dependency files commonly used by the LIN Driver module:

- ComStack_Types.h,
- Lin_GeneralTypes.h,
- Lin_MemMap.h,
- Os.h,
- EcuM.h,
- Det.h,
- Rte.h,
- Dem.h,
- SchM_Lin.h,
- EcuM_Cbk.h,
- LinIf_Cbk.h,
- rh850_Types.h

3.3.11.6 **Stubs**

Refer to 3.2.9 for the common stubs used for LIN Driver component.

3.3.11.7 **Addition Error Handling**

Refer to " Development and Production Errors" section at LIN chapter in Driver Component Embedded User's Manual.

3.3.11.8 **Restrictions**

None

3.3.11.9 **Sample Application**

3.3.11.9.1 **Sample Application Structure**

Refer to3.7 Sample Application.

In the Sample Application all the LIN APIs are invoked in the following sequences.

- The API Lin_Init is invoked with a valid database address for the proper initialization of the LIN Driver. All the LIN Driver control registers and RAM variables get initialized.
- The API Lin_GetVersionInfo is invoked to get the version of the LIN Driver module with a variable of Std_VersionInfoType. After this API is called, the passed parameter gets updated with the LIN Driver version details.
- The API Lin_SendFrame is invoked to transmit the Header and Response as a part of the LIN Frame on the addressed LIN Channel with PID 0xDD and data as 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 which in turn results in transmission of the same.
- The API Lin_SendFrame is invoked to transmit the Header and receive the Response as a part of the LIN Frame on the addressed LIN Channel with PID 0x39 and data as 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, which in turn results in reception of the same.

3.3.11.9.2 **Recommended Environment**

- **S4 G4MH Environment**

Table 3-101 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.

Debugger Hardware	Debugger
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

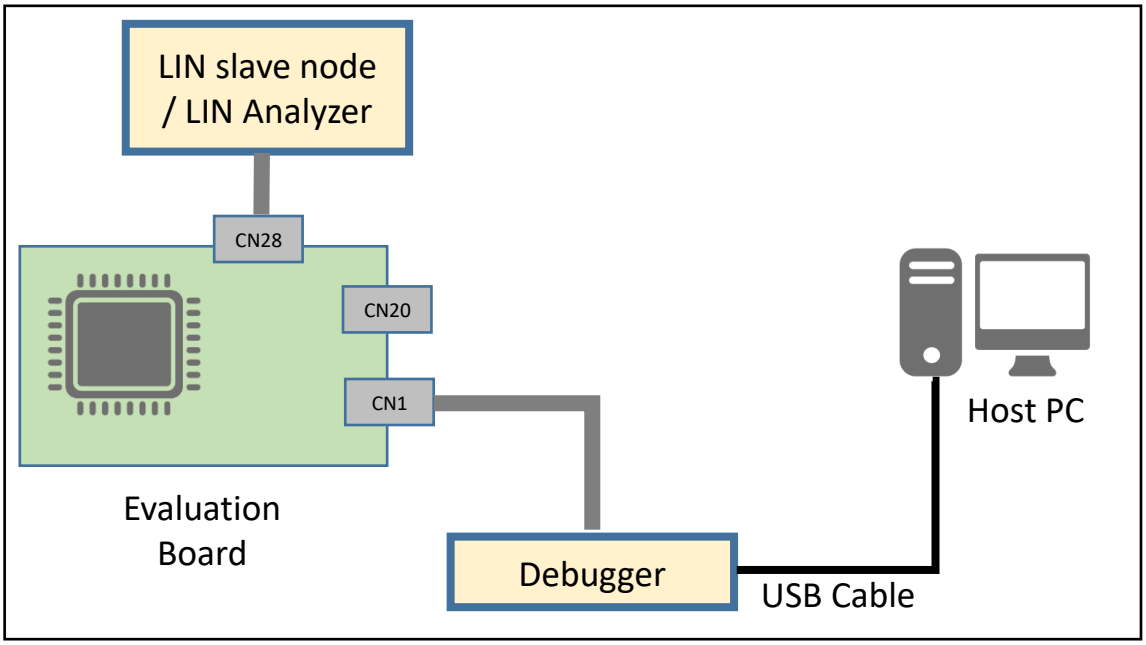

Figure 3-35 S4 Spider board connection of LIN Sample Application

3.3.11.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.

3.3.11.9.4**How to build Sample Application**

Refer to 3.7.2.2 How to build the Sample Application

- Run Window Power Shell or Command Prompt and change the current working directory to “make” directory present as mentioned in below path:
  external\rel\S4\common_family\make\ghs
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat LIN R19-11 S4 No

3.3.11.9.5**How to run Sample application**

Step 1: Set BreakPoint at each function which needs to check and then run step over that BreakPoint or set BreakPoint at the end of the main function:

Step 2: Run program to breakpoint, confirm all elements in the array App_GaaTestResult [0…n] are set to 1 or check by print logs throught the Teraterm if it is used:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed

3.3.11.10 **ROM/RAM Usage**

See Appendix LIN section for information on measuring RAM/ROM consumption.

3.3.11.11 **Stack Depth**

See Appendix LIN section for information on measuring stack depth.

3.3.11.12 **Throughput Details**

See Appendix LIN section for information on measuring execution time, and functional testing.

3.3.12 FlexRay Driver Component

3.3.12.1 Module Overview

The FlexRay Driver is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment. The FlexRay Driver component comprises embedded software and the configuration Tool to achieve scalability and configurability.

The FlexRay Driver provides services for FlexRay communication.

The FlexRay Driver component provides the following functionalities:

- To initialize the FlexRay communication controllers
- To start, halt or abort the communication
- To configure the channel to send the wakeup pattern and to transmit the wakeup pattern on the configured FlexRay channel
- To get the current POC status of CC
- To get the synchronization state of CC and to adjust the global time of a FlexRay CC to an external clock source
- To transmit the frames on the FlexRay channels
- To receive the frames transmitted on the FlexRay channels
- To get the current cycle and macrotick offset value of CC
- To set the value for absolute timer interrupt and to stop the absolute timer
- To enable/disable the absolute timer interrupt. To reset the interrupt condition of absolute timer interrupt and to get the status of absolute timer interrupt
- To get the Channel status, Clock Correction, Number of startup frames, Clock Correction, Sync frame list and wakeup Rx status of CC
- To get the Nm Vector Information received on CC
- To send CC to ALLSLOTS and ALLOW_COLDSTART modes
- To reconfigure or disable an LPdu at run-time.
- To get the information of transmitted/received frame.

3.3.12.2 **Module Dependency**

The dependency of FlexRay Driver component on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever FlexRay module encounters a development error.

DEM
The Diagnostic Event manager (DEM) will be called whenever FlexRay module encounters a production relevant error.

RTE
The Run Time Environment (RTE) module will be called whenever a critical section protection function is called.

FR Interface
The FR Interface belongs to the ECU Abstraction Layer. It will provide the upper layer with abstract interface to the FlexRay Communication System the following groups of functions:

- Initialization
- Data transmission (sending and reception)
- Start/halt/abort communication
- FlexRay specific functions (e.g., send wake-up pattern)
- Set operation mode
- Get status information
- Various timer functions

3.3.12.3 **Folder Structure**

Table 3-102 and **Table 3-103** show the list of Source Code Files and Header Files for FlexRay module.

Table 3-102 FlexRay Header Files

Location: rel\modules\fr\	Supported Device			Category	Description
	S4_CR52 (*)	S4_G4MH (*)	V4H		
include	-	-	-	-	-
Fr_59_Renesas.h	-	x	-	Product	This file provides extern declarations for all FR Driver Component APIs. It provides service Ids of APIs, DET Error codes and type definitions for FR Driver initialization structure. This header file shall be included in other modules to use the features of FR Driver Component.
Fr_59_Renesas_LLDriver.h	-	x	-	Product	This file contains the Function Prototypes defined in Fr_59_Renesas_LLDriver.c file.
Fr_59_Renesas_PBTypes.h	-	x	-	Product	This file contains the data structure definitions of the frame header type configuration and low-level configuration.
Fr_59_Renesas_Ram.h	-	x	-	Product	This file contains the extern declarations for the global variables defined in Fr_Ram.c file and the version information of the file.
Fr_59_Renesas_RegWrite.h	-	-	-	Product	This file contains the macros for register writing/verification, and HW consistency. Note: HW consistency function is always disabled in the current release version.
Fr_59_Renesas_Types.h	-	x	-	Product	This file contains the common macro definitions and the data types required internally by the FR software component.
Fr_59_Renesas_Version.h	-	x	-	Product	This file contains the definitions of AUTOSAR version numbers of all modules interfaced to FR Driver.

x: applicable
-: not applicable
(*) R-Car S4N hardware does not support FlexRay

Table 3-103 FlexRay Source Files

Location: rel\modules\fr\	Supported Device			Category	Description
	S4_CR52	S4_G4MH	V4H		
src	-	-	-	-	-
Fr_59_Renesas.c	-	x	-	Product	This file contains the implementation of all APIs.
Fr_59_Renesas_LLDriver.c	-	x	-	Product	This file contains the FR Low Level Driver code.
Fr_59_Renesas_Ram.c	-	x	-	Product	This file contains the global variables used by FR Driver Component.
Fr_59_Renesas_Version.c	-	x	-	Product	This file contains the code for checking version of all modules interfaced to FR Driver.

x: applicable

-: not applicable

Table 3-104 shows the list of Parameter Definition Files for FR module.

Table 3-104 FR Parameter Definition Files

Location: rel\modules\fr\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19-11	S4_G4MH	R1911_FR_S4_RTM8RC79FG.arxml	RTM8RC79FG

Note Product Name**:** Product Names supported by “Files”.

3.3.12.4 Configuration Parameter Dependency

Table 3-105 shows the list of configuration parameter dependency for FlexRay Driver component.

Table 3-105 FlexRay Driver Component Configuration Parameter Dependency

Parameter	Module	Path
FR_E_CTRL_TESTRESULT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
FrDemReadTimeoutFailure	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.3.12.5 Source Code Dependency

The followings are the dependency files commonly used by the FlexRay Driver module:

- Std_Types.h,
- Fr_GeneralTypes.h,
- Fr_MemMap.h,
- Platform_Types.h,
- Dem.h,
- SchM_Fr_59_Renesas.h,
- Det.h,
- Rte.h

3.3.12.6 Stubs

Refer to 3.2.9 for the common stubs used for FR Driver component.

3.3.12.7 Addition Error Handling

Refer to " Development and Production Errors" section at FR chapter in Driver Component Embedded User's Manual.

3.3.12.8 Restrictions

None

3.3.12.9 Sample Application

3.3.12.9.1 Sample Application Structure

Refer to 3.7 Sample Application.

In the FR sample application include the following sequence:

- The API Fr_59_Renesas_Init is invoked for proper initialization of FR Driver.
- The API Fr_59_Renesas_ControllerInit is invoked to initialize the communication controller.
- The API Fr_59_Renesas_GetPOCStatus is invoked to confirm the POC state is READY
- The API Fr_59_Renesas_SendWUP is invoked to send wakeup pattern.
- The API Fr_59_Renesas_AllowColdStart is invoked to perform coldstart.
- The API Fr_59_Renesas_StartCommunication is invoked to start communication.
- The API Fr_59_Renesas_GetPOCStatus is invoked to confirm the POC is NORMAL_ACTIVE.
- The API Fr_59_Renesas_EnableAbsoluteTimerIRQ is invoked to enable the interrupt timer.
- The API Fr_59_Renesas_SetAbsoluteTimer is invoked to set absolute timer.
- Waiting till the frame gets transmitted.
- The API Fr_59_Renesas_SetAbsoluteTimer is invoked to set absolute timer.
- Waiting till received a valid frame.

Note

1. Since at least two nodes are necessary to start up the cluster communication, it may require several hundred milliseconds before all nodes are completely awakened and configured.

2. The second node of the cluster shall be configured in which its configuration including FlexRay CC configuration which shall be consistent with FlexRay Sample Application Configuration and Protocol specification which shall be consistent with FlexRay Interface Configuration used in Sample Application of FlexRay.

3.3.12.9.2 Recommended Environment (Not supported)

3.3.12.9.3 Preparation (Not supported)

3.3.12.9.4 How to build Sample Application

Refer to 3.7.2.2 How to build the Sample Application

3.3.12.9.5 How to run Sample Application (Not Supported)

3.3.12.10 ROM/RAM Usage

See Appendix FR section for information on measuring RAM/ROM consumption.

3.3.12.11 Stack Depth

See Appendix FR section for information on measuring stack depth.

3.3.12.12 Throughput Details

See Appendix FR section for information on measuring execution time, and functional testing.

3.3.13 Ethernet Driver Component

3.3.13.1 Module Overview

Ethernet Driver is part of the microcontroller abstraction layer (MCAL). It performs access to the ETNC/ETNB/RSwitch2/Ethernet SERDES hardware registers and offer API/Interface to the upper layer, independent from the underlying hardware. The only upper layer that access the Ethernet Driver is the Ethernet Interface.
Ethernet Driver provides the following main features:

- Initialization, Write, Read Access to the Transceiver Registers (via the MII interface).
- Send /Receive Ethernet Frames.
- Get Statistics from /to the packet transmitted / received

3.3.13.2 Module Dependency

The dependency of Ethernet Driver on other modules and the required implementation is briefed as follows:

Ethernet Interface
Ethernet Driver use the Ethernet Interface callback to inform upper layer every time the Frame has been transmitted/received or an operation mode has been successfully changed.

Ethernet Transceiver
Ethernet Driver uses the Ethernet Transceiver callback to inform upper layer every time the MII has been read/written.

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
The Diagnostic Event manager (DEM) will be called whenever this module encounters a production relevant error.

MCU
MCU is referenced during configuration generation to read the Peripheral Input Clock that inputs to the ETNC and ETNB macro.

PORT
Ethernet Drover relies on the Port module to configure the CPU pin mode to assure the correct functionality of the Ethernet Communication.

OS
As the ETH Driver uses interrupts, it depends on the OS which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

3.3.13.3 **Folder Structure**

Table 3-106 and **Table 3-107** show the list of Source Code Files and Header Files for Ethernet module.

Table 3-106 Ethernet Header Files

Location: rel\modules\eth\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
Eth.h	x	x	x	x	Product	This file provides extern declarations for all ETH Driver Component APIs. It provides service Ids of APIs, DET Error codes and type definitions for ETH Driver initialization structure. This header file shall be included in other modules to use the features of ETH Driver Component.
Eth_Common_LLDriver.h	x	x	x	x	Product	This file contains the common prototypes and definitions for all low-level drivers.
Eth_Filter.h	x	x	x	x	Product	This file contains the prototypes and definitions for the software MAC address filter functions.
Eth_Types.h	x	x	x	x	Product	This file contains the common macro definitions and the data types required internally by the ETH Driver component.
Eth_Util.h	x	x	x	x	Product	This file contains the prototypes and definitions for the common function algorithms.
Eth_Version.h	x	x	x	x	Product	This file contains the code to check the version of all modules interfaced to ETH Driver.
RSW2	-	-	-	-	-	-
Eth_RSW2_Irq.h	x	-	-	-	Product	This file contains Interrupt Service Routine for ETH Driver Component.
Eth_RSW2_LLDriver.h	x	-	-	-	Product	This file contains the common prototypes and definitions for the RSwitch2 low-level drivers.
Eth_RSW2_Ram.h	x	-	-	-	Product	This file contains the extern declarations for the global variables and the RAM Allocation functions defined in Eth_Ram.c file and the version information of the file.
Eth_Serdes_LLDriver.h	x	-	-	-	Product	This file contains the common prototypes and definitions for the Ethernet SERDES low-level drivers.
ETNB	-	-	-	-	-	-
Eth_ETNB_Dma.h	-	x	-	-	Product	This file contains the common prototypes and definitions for the AVB-DMAC.
Eth_ETNB_Irq.h	-	x	-	-	Product	This file contains Interrupt Service Routine for ETH Driver Component.

Eth_ETNB_LLDriver.h	-	x	-	-	Product	This file contains the common prototypes and definitions for the ETNB low-level drivers.
Eth_ETNB_Ram.h	-	x	-	-	Product	This file contains the extern declarations for the global variables and the RAM Allocation functions defined in Eth_Ram.c file and the version information of the file.
AVB	-	-	-	-	-	-
Eth_AVB_Dma.h	-	-	x	x	Product	This file contains the common prototypes and definitions for the AVB-DMAC.
Eth_AVB_Irq.h	-	-	x	x	Product	This file contains Interrupt Service Routine for ETH Driver Component.
Eth_AVB_LLDriver.h	-	-	x	x	Product	This file contains the common prototypes and definitions for the AVB low-level drivers.
Eth_AVB_Ram.h	-	-	x	x	Product	This file contains the extern declarations for the global variables and the RAM Allocation functions defined in Eth_Ram.c file and the version information of the file.

x: applicable

-: not applicable

Table 3-107 Ethernet Source Files

Location: rel\modules\eth\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
Eth.c	x	x	x	x	Product	This file contains the implementation of all public APIs directly exposed to the User.
Eth_Filter.c	x	x	x	x	Product	This file contains the implementation of software MAC address filter functions.
Eth_Util.c	x	x	x	x	Product	This file contains the implementation of software common algorithm functions.
Eth_Version.c	x	x	x	x	Product	This file contains the code to check the version of all modules interfaced to ETH Driver.
RSW2	-	-	-	-	-	-
Eth_RSW2_Irq.c	x	-	-	-	Product	This file contains Interrupt Service Routine for ETH Driver Component.
Eth_RSW2_LLDriver.c	x	-	-	-	Product	This file contains the implementations of RSwitch2-specific low-level driver codes.
Eth_RSW2_Ram.c	x	-	-	-	Product	This file contains the global variables used by ETH Driver Component and the implementations of buffer manipulation functions.
Eth_Serdes_LLDriver.c	x	-	-	-	Product	This file contains the implementations of Ethernet SERDES-specific low-level driver codes.
ETNB	-	-	-	-	-	-
Eth_ETNB_Dma.c	-	x	-	-	Product	This file contains the implementations of AVB-DMAC-specific driver codes.
Eth_ETNB_Irq.c	-	x	-	-	Product	This file contains Interrupt Service Routine for ETH Driver Component.
Eth_ETNB_LLDriver.c	-	x	-	-	Product	This file contains the implementations of ETNB-specific low-level driver codes.
Eth_ETNB_Ram.c	-	x	-	-	Product	This file contains the global variables used by ETH Driver Component and the implementations of buffer manipulation functions.
AVB	-	-	-	-	-	-
Eth_AVB_Dma.c	-	-	x	x	Product	This file contains the implementations of AVB-DMAC-specific driver codes.
Eth_AVB_Irq.c	-	-	x	x	Product	This file contains Interrupt Service Routine for ETH Driver Component.
Eth_AVB_LLDriver.c	-	-	x	x	Product	This file contains the implementations of AVB-specific low-level driver codes.
Eth_AVB_Ram.c	-	-	x	x	Product	This file contains the global variables used by ETH Driver Component and the implementations of buffer manipulation functions.

x: applicable

-: not applicable

Table 3-108 shows the list of Parameter Definition Files for Ethernet module.

Table 3-108 ETH Parameter Definition Files

Location: rel\modules\eth\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19-11	S4_CR52	R1911_ETH_S4_RTM8RC79FR.arxml	RTM8RC79FR
	S4_G4MH	R1911_ETH_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R1911_ETH_V4H.arxml	V4H
	V4M	R1911_ETH_V4M.arxml	V4M

Note Product Name**:** Product Names supported by “Files”.

3.3.13.4 **Configuration Parameter Dependency**

Table 3-109 shows the list of configuration parameter dependency for Ethernet Driver component.

Table 3-109 Ethernet Driver Component Configuration Parameter Dependency

- S4_G4MH

Parameter	Module	Path
ETH_E_ACCESS	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_ALIGNMENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_CRC	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_DMA_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_LATECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_MULTIPLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_OVERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_REGISTER_CORRUPTION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_RX_FRAMES_LOST	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_SINGLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_TIMERINC_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_TIMEROFFSET_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_UNDERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
EthOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter
EthInputClockRef	MCU	/ActiveEcuC/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuHBusClk

- S4_ CR52

Parameter	Module	Path
ETH_E_ACCESS	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_ALIGNMENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_CRC	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_DMA_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_LATECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_MULTIPLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_OVERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_REGISTER_CORRUPTION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

ETH_E_RX_FRAMES_LOST	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_SINGLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_TIMERINC_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_TIMEROFFSET_FAILED	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_UNDERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
EthOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter
EthInputClockRef	MCU	/ActiveEcuC/Mcu/McuModuleConfiguration/McuClockSettingConfig/McuHBusClk

- V4H

Parameter	Module	Path
ETH_E_ACCESS	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_ALIGNMENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_CRC	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_DMA_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_LATECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_MULTIPLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_OVERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_REGISTER_CORRUPTION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_RX_FRAMES_LOST	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_SINGLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_UNDERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
EthOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter

- V4M

Parameter	Module	Path
ETH_E_ACCESS	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_ALIGNMENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_CRC	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_DMA_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_LATECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

ETH_E_MULTIPLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_OVERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_REGISTER_CORRUPTION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_RX_FRAMES_LOST	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_SINGLECOLLISION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
ETH_E_UNDERSIZEFRAME	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
EthOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter

3.3.13.5 **Source Code Dependency**

The followings are the dependency files commonly used by the Ethernet Driver module:

- Eth_MemMap.h,
- Os.h,
- Eth_GeneralTypes.h,
- EthIf.h,
- EthTrcv.h,
- Det.h,
- SchM_Eth.h,
- Dem.h,
- Std_Types.h,
- ComStack_Types.h,
- Rte.h,
- rh850_Types.h

3.3.13.6 **Stubs**

Refer to 3.2.9 for the common stubs to be used for Ethernet Driver component.

3.3.13.7 **Addition Error Handling**

Refer to " Development and Production Errors" section at ETH chapter in Driver Component Embedded User's Manual.

3.3.13.8 **Restrictions**

None

3.3.13.9 **Sample Application**

3.3.13.9.1 **Sample Application Structure**

Refer to 3.7 Sample Application

❖ **V4H**

The operation of the sample application is as follows.

1. Execute initialization routine (operates in interrupt mode or polling mode)
2. Register a filtering ID that filters IEEE1722 packets to the stream queue.
3. Register a receivable multicast address.
4. Send a unicast frame.
5. Get the current time by gPTP timer. (enable via EthGlobalTimeSupport parameter only in secure mode)
6. Wait until receiving the unicast frame transmitted in "4"
7. Send a multicast frame.
8. Send LLDP frame for PTP.
9. Transmit IEEE1722 frames not to be filtering.
10. Transmit the IEEE1722 frame to be filtered.
11. Wait until frames "7" to "10" are received

12. Get drop counter information.
13. Get statistics on transmission / reception.

In this sample application, mechanism to echo back to the partner host is recommended,
If you do not implement mechanism to echo back to the partner host, you need to use loopback mode.

- **S4_G4MH**

The operation of the sample application is as follows.

1. Execute initialization routine (operates in interrupt mode or polling mode)
2. Register a filtering ID that filters IEEE1722 packets to the stream queue.
3. Register a receivable multicast address.
4. Send a unicast frame.
5. Get the current time by gPTP timer. (enable via EthGlobalTimeSupport parameter only in secure mode)
6. Wait until receiving the unicast frame transmitted in "4"
7. Send a multicast frame.
8. Send LLDP frame for PTP.
9. Transmit IEEE1722 frames not to be filtering.
10. Transmit the IEEE1722 frame to be filtered.
11. Wait until frames "7" to "10" are received
12. Get drop counter information.
13. Get statistics on transmission / reception.

In this sample application, mechanism to echo back to the partner host is recommended,
If you do not implement mechanism to echo back to the partner host, you need to use loopback mode.

- **S4_ CR52**

The operation of the sample application is as follows.

1. Execute initialization routine (operates in interrupt mode)
2. Configure Mac Addres for Controller0, Controller1 and Controller2.
3. Register a receivable multicast address.
4. Enable Controller0, Controller1 and Controller2.
5. Get the current time by gPTP timer. (Not supported, disabled via EthGlobalTimeSupport parameter)
6. gPTP timer offset test. (Not supported, disabled via EthGlobalTimeSupport parameter)
7. gPTP timer Increment test. (Not supported, disabled via EthGlobalTimeSupport parameter)
8. Send normal unicast frame to the Contoller.
9. Wait for the frame transmitted in "8" to be looped back by the partner host and received by the Controller.
10. Verifies that the payloads of the transmitted and received frames are same.
11. Repeat Step “8”-“10” 1500 times for each Controller.
12. Get drop counter information.
13. Get statistics on transmission / reception.

- **V4M**

The operation of the sample application is as follows.

1. Execute initialization routine (operates in interrupt mode or polling mode)
2. Register a filtering ID that filters IEEE1722 packets to the stream queue.
3. Register a receivable multicast address.
4. Send a unicast frame.
5. Get the current time by gPTP timer. (enable via EthGlobalTimeSupport parameter only in secure mode)
6. Wait until receiving the unicast frame transmitted in "4"

7. Send a multicast frame.
8. Send LLDP frame for PTP.
9. Transmit IEEE1722 frames not to be filtering.
10. Transmit the IEEE1722 frame to be filtered.
11. Wait until frames "7" to "10" are received
12. Get drop counter information.
13. Get statistics on transmission / reception.

In this sample application, mechanism to echo back to the partner host is recommended,
If you do not implement mechanism to echo back to the partner host, you need to use loopback mode.

Note:
“8” and later operations are only performed on controllers that have successfully linked up.
(i.e., the RJ45 connector on the evaluation board and partner host are connected via Ethernet cable and the link is established).

3.3.13.9.2**Recommended Environment**

- **S4 CR52 Environment**

Table 3-110 S4 CR52 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for Windows host PC.
Debugger Hardware	PowerDebug USB 3.0 (License: ARMv8-A/R)
Debugger Software	TRACE32 R2021.02
RTP8A779F0ASKB0ST0S	R-CarS4 System Evaluation Board "Spider" (ETHER TSN SUB Board)
Ether Network Interface + Analyzer	CANoe/CANalyzer 8.5 + VN5610
Terminal Software	Teraterm Ver4.100

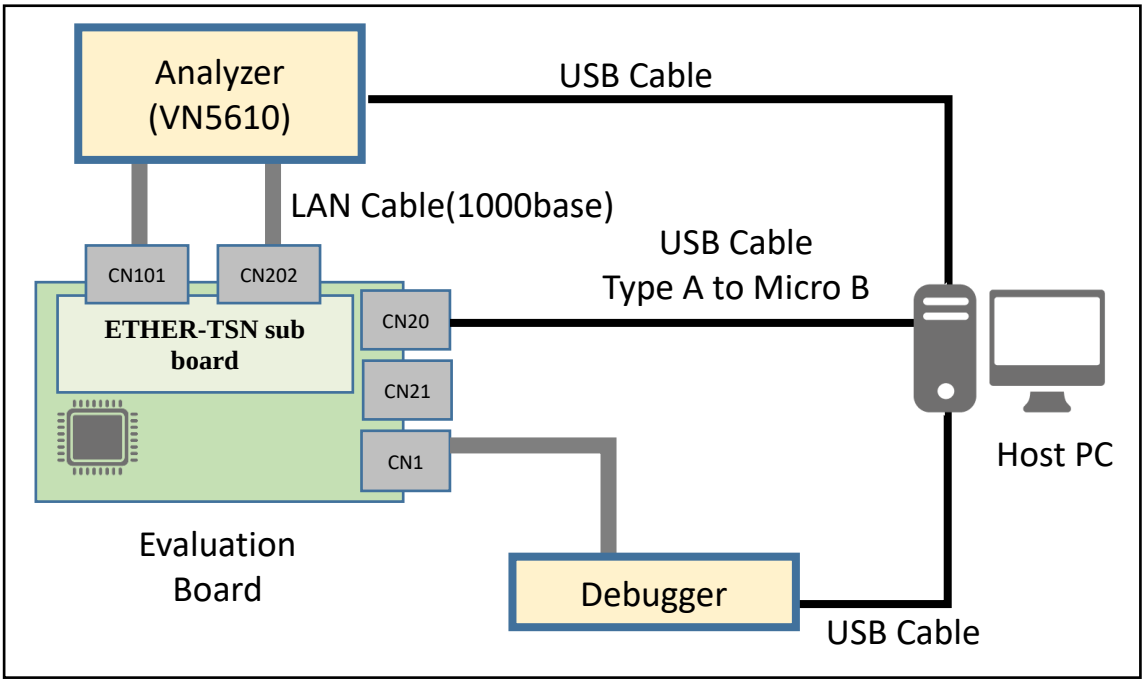

Figure 3-36 S4 Spider board connection of ETH (S4_CR52) Sample Application

- **S4 G4MH Environment**

Table 3-111 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger
Debugger Software	CS+ for CC E8.07.00c
Ether Network Interface + Analyzer	CANalyzer 10.0 (Software) + VN5610A (Hardware)
Terminal Software	Teraterm Ver4.100

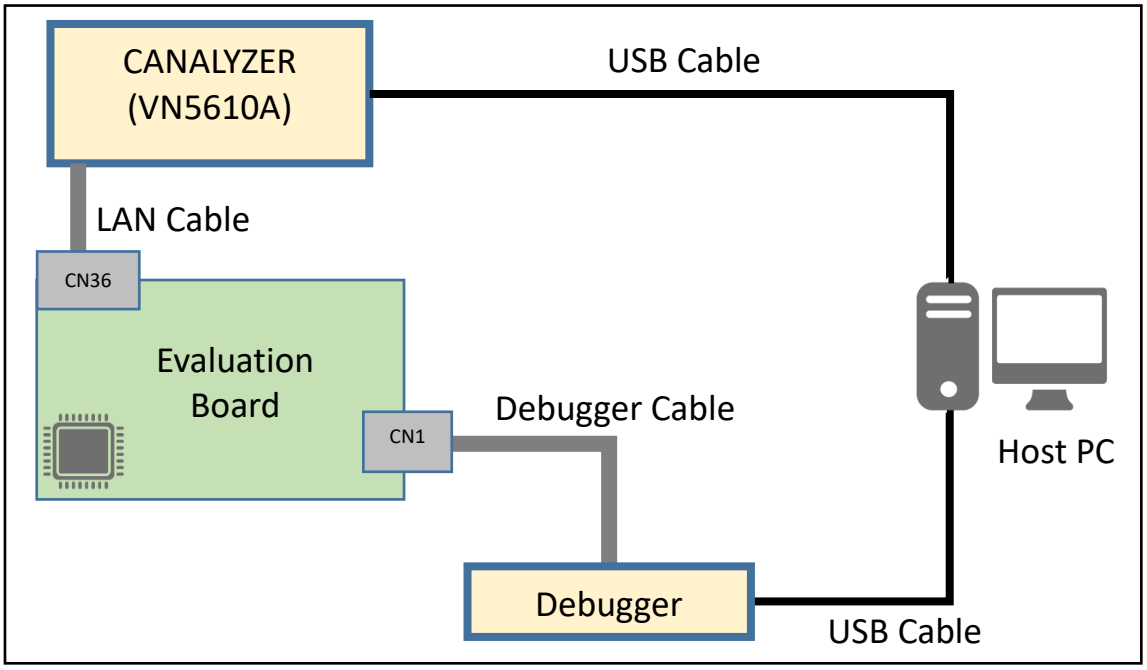

Figure 3-37 S4 Spider board connection of ETH (S4_G4MH) Sample Application

- **V4H Environment**

Table 3-112 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for Windows host PC.
Debugger Hardware	PowerDebug USB 3.0 (License: ARMv8-A/R)
Debugger Software	TRACE32 R2021.02
Ether Network Interface + Analyzer	CANoe/CANalyzer 8.5 + VN5610
Terminal Software	Teraterm Ver4.100

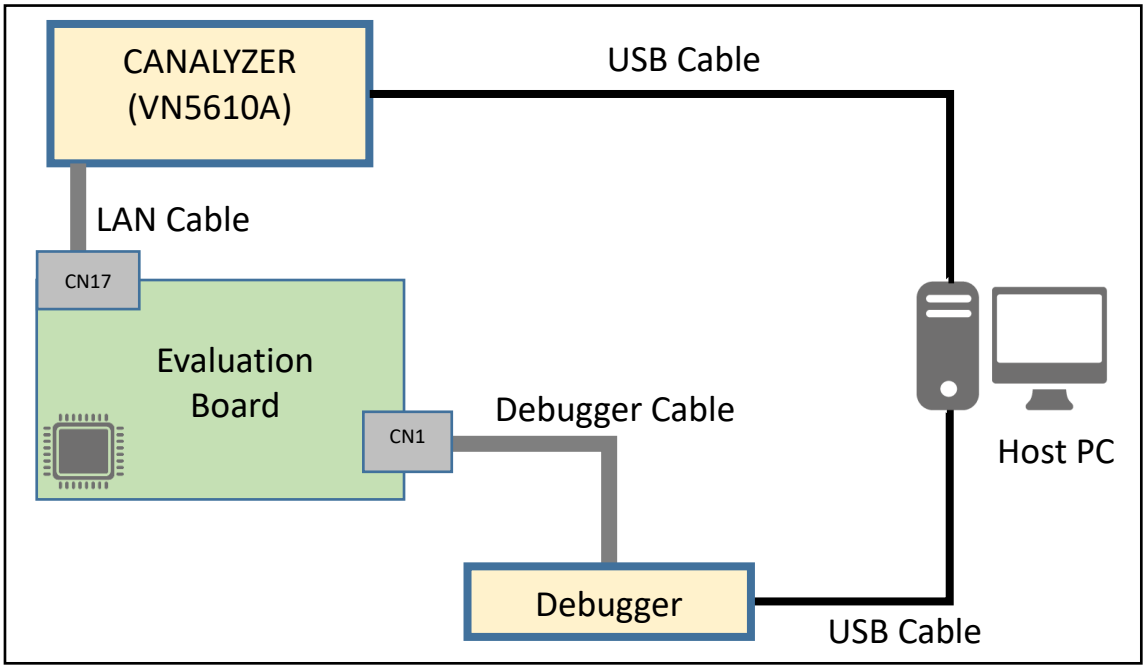

Figure 3-38 V4H White Hawk board connection of ETH (V4H) Sample Application

- **V4M Environment**

Table 3-113 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for Windows host PC.
Debugger Hardware	PowerDebug USB 3.0 (License: ARMv8-A/R)
Debugger Software	TRACE32 R2021.02
Ether Network Interface + Analyzer	CANoe/CANalyzer 8.5 + VN5610
Terminal Software	Teraterm Ver4.100

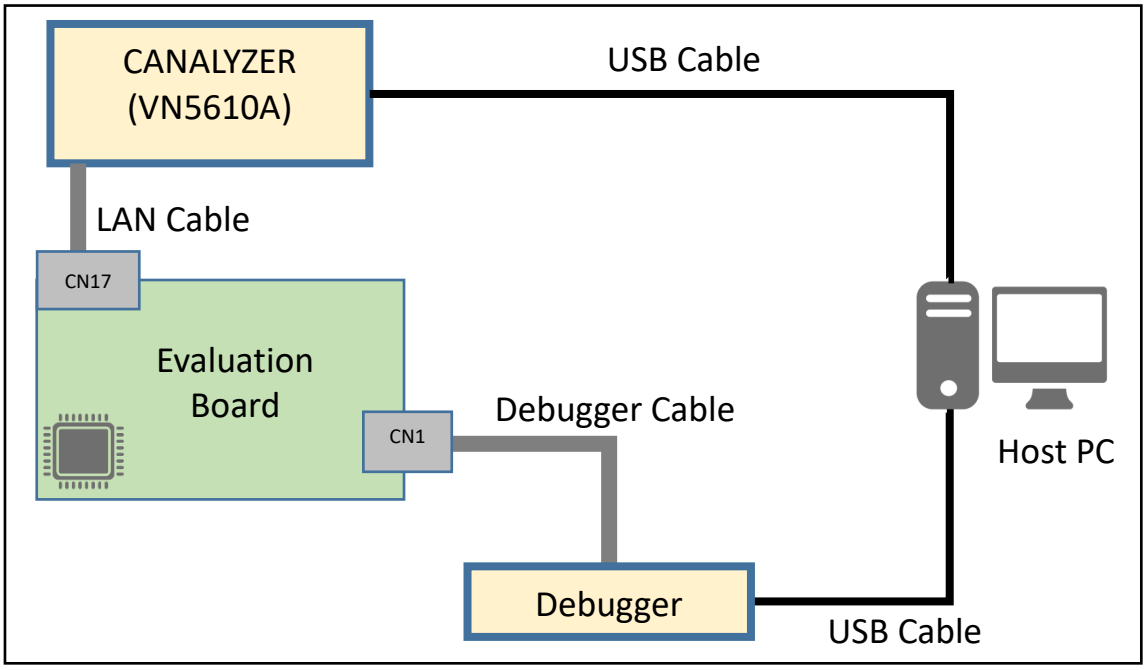

Figure 3-39 V4M Gray Hawk board connection of ETH (V4M) Sample Application

3.3.13.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H, read more environment setup and guideline at 3.7.3 How to run Sample Application.
- For S4_CR52, check the addtional steps as below,

Step1: Create linkup Script for TRACE32

Create Linkup Script File (*.cmm). The following is a example.

```
;--------------------------------------------------
;           Configure system settings (DO NOT EDIT)
;--------------------------------------------------
SYStem.RESet
SYStem.CPU CortexR52
SYStem.CONFIG COREDEBUG Base DAP:0x80c10000
SYStem.CONFIG CTI       Base DAP:0x80c20000
SYStem.CONFIG GICD Type GICv3
SYStem.CONFIG GICD Base 0xF0000000
SYStem.JtagClock 20.MHz
SYStem.Option.ResBreak off
SYStem.Option.TRST on
SYStem.Option.EnReset on
SYStem.Option WaitReset 500ms
SYStem.ATTACH

Break.Select Program Onchip
```

Step2: Write the IPL into the flash memory of the Spider board.
Please refer to “ICUMX IPL for R-Car Gen4 User’s Manual: Software” and “R-Car Gen4 Flash writer sample software User’s Manual: Software” for detailed instructions.

Step3: Create Canoe/Canalyzer Project
Create a CANoe/CANalyzer project.
Configure the CANoe/CANalyzer script so that the Analyzer loops back the Ether frames received from the evaluation board to source MAC address.

3.3.13.9.4 How to build Sample Application

Refer to 3.7.2.2 How to build the Sample Application
Note: If you want the ETH Sample Application to run in polling mode, change the following configuration parameters to false.

- EthCtrlEnableRxInterrupt
- EthCtrlEnableTxInterrupt

3.3.13.9.5 How to run Sample Application

Step1: Start Terminal Software

- Connect the terminal software to the two serial ports (CN20, CN21).

- Please set up each as follows. (**Remark**: The port number depends on your environment.)

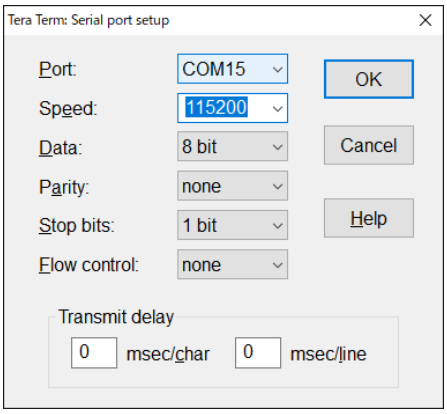

Figure 3-40 Setting for Port 1 (to CN20)

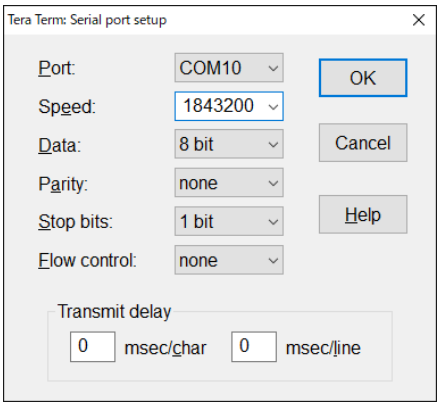

Figure 3-41 Setting for Port 2 (to CN21)

Step2: Launch the Canoe/Canalyzer

- Startup Canoe/Canalyzer application and start monitoring.

Step3: Power On

- Turn on the power switch (SW11) of the evaluation board.

Step4: Launch the TRACE32.

- Startup TRACE32 application.

Step5: Open Linkup Script

- Open Linkup Script file (*.cmm). Press [File] in Menu bar -> [Open Script…]

Step6: CPU Reset

- Press [CPU] in Menu bar -> [In Target Reset].

- Wait for the Dummy U-Boot to finish booting. You can check the status in the terminal log.

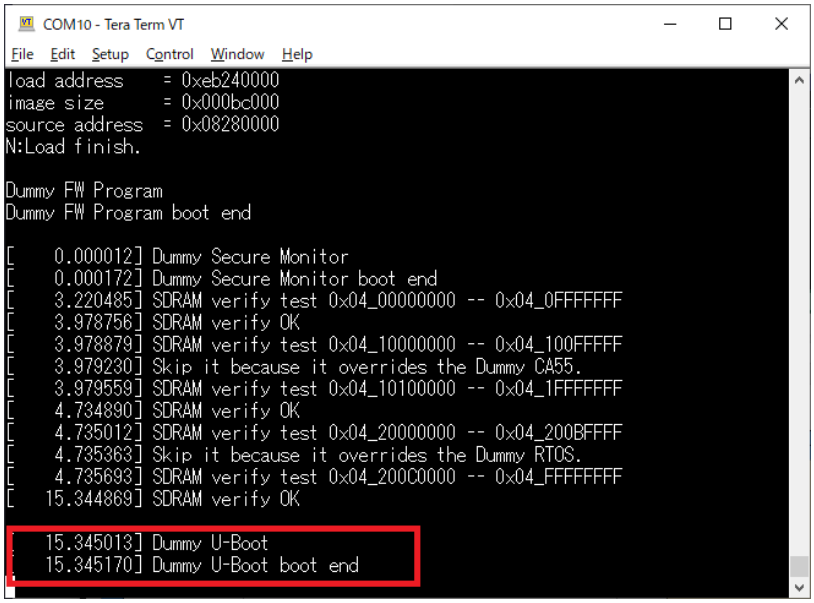

Figure 3-42 Status in the terminal log

Step7: Load Sample App binary file

- Press [Do] in Linkup Script window.
- Press [ ] in Menu bar.
- Drag the Sample App binary file(.elf) to the TRACE32 command line. And press Enter.

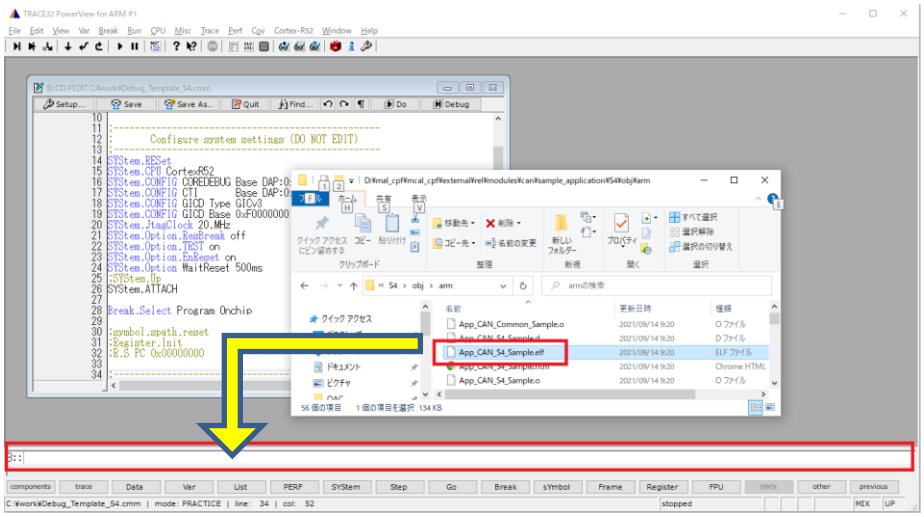

Figure 3-43 Drag the Sample App binary file

Step8: **Program Start**

- Press [ ] in Menu bar.

Step9: Confirmation Checkpoint

This sample confirms the following checkpoints:

- No DEM Error
- No DET Error
- Verification of Ether frames sent to the linked port and frames received in loopback (1500 times per port)

If all checkpoints are passed, the following log will be output to the serial port (CN20).

```
EXECUTE OK
```

If any checkpoint fails, the following log will be output to the serial port (CN20).

```
EXECUTE NOT OK
```

Step 1: After execute sample app, the program is expected to stay in a loop of a function.

Step 2: Stop the program the see the loop of which function that the program stay in:

- If final function is sample_end(), The transmitted message and the received messages are matched.(OK)
- If final function is sample_NG_end(), The transmitted message and receiving messages are different.(NG) then check elements in the array of EthPassedCount[] to see which checkpoint in sample app failed.

Step 1: After execute sample app, the program is expected to stay in a loop of a function.

Step 2: Stop the program the see the loop of which function that the program stay in:

- If final function is sample_end(), The transmitted message and the received messages are matched.(OK)
- If final function is sample_NG_end(), The transmitted message and receiving messages are different.(NG) then check elements in the array of EthPassedCount[] to see which checkpoint in sample app failed.

Note: The configuration 1000BASE of PHY 88Q2110 with the corresponding device on Ethernet sub-board are configured compliant mode in Sample Application. If the PHY corresponding device used Marvell PHY in legacy mode. In order to the communication is successful. The configuration of PHY 88Q2110 on Ethernet sub-board should be legacy mode. This can be done by setting the value of Macro PHY_MODE_SETTING to LEGACY_MODE in App_ETH_Device_Sample.h file.

3.3.13.10 **ROM/RAM Usage**

See Appendix ETH section for information on measuring RAM/ROM consumption.

3.3.13.11 **Stack Depth**

See Appendix ETH section for information on measuring stack depth.

3.3.13.12 **Throughput Details**

See Appendix ETH section for information on measuring execution time, and functional testing.

3.4 CDD module

3.4.1 ICCOM Driver Component

3.4.1.1 Module Overview

The ICCOM Driver is part of the Complex Device Driver layer (CDD), which performs the hardware access and offers hardware-independent API to the upper layer to support communication between domains.

The ICCOM Driver component shall provide the following main features:

- Initialization
- Transmission data to another domain
- Notification back to ASWC
- Reception data from another domain
- Periodic send processing to resend pending request (Send Data or ACK)
- Getting Version Information.

3.4.1.2 Module Dependency

The dependency of ICCOM Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

RTE
The Run Time Environment (RTE) module will be called to perform the communication with connected AUTOSAR Software Component.

OS
As the ICCOM Driver uses interrupts, it depends on the OS, which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed. The ICCOM Driver access to GetCounterValue and GetElapsedCounterValue services of the OS to calculate the time of transmission.

EcuM
ICCOM Driver initialization function shall be exclusively called by the EcuM.

SchM
The Basic Software Scheduler module will be called whenever a critical section protection function is called.

3.4.1.3 Folder Structure

Table 3-114 and **Table 3-115** show the list of Source Code Files and Header Files for ICCOM module.

Table 3-114 ICCOM Header Files

Location: rel\modules\cddiccom\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
CDD_Iccom.h	x	x	x	x	Product	This file provides externs declaration, type definition and config pointer for global as well as initialization of CDDICCOM.
CDD_Iccom_Version.h	x	x	x	x	Product	C headers file for CDD_Iccom_Version.c. It includes the definitions for get version info function.
CDD_Iccom_Cbk.h	x	x	x	x	Product	This file contains definition of channel notification callback function.
CDD_Iccom_MainServ.h	x	x	x	x	Product	C headers file for CDD_Iccom_MainServ.c. It includes the definitions that are required for CDD_Iccom_MainServ.c
CDD_Iccom_PBTypes.h	x	x	x	x	Product	This file provides the definition for Post-Build configuration types and setting.
CDD_Iccom_Ram.h	x	x	x	x	Product	C headers file for CDD_Iccom_Ram.c. It includes the definitions that are required for RAM data.
CDD_Iccom_Types.h	x	x	x	x	Product	This file provides the definition of CDDICCOM data types
CDD_Iccom_Cfg.h	x	x	x	x	Product	This file contains various driver Pre-compile time parameters. It also contains the macro for the total number of channels configured, acknowledgement timeout and protection area.
MFIS	-	-	-	-	-	-
CDD_Iccom_MFIS_LLDriver.h	x	x	x	x	Product	C headers file for CDD_Iccom_MFIS_LLDriver.c. It includes the definitions that are required for CDD_Iccom_MFIS_LLDriver.c
CDD_Iccom_MFIS_Irq.h	x	x	x	x	Product	C headers file for CDD_Iccom_MFIS_Irq.c. It includes the header files that are required for CDD_Iccom _MFIS_Irq.c

x: applicable

-: not applicable

Table 3-115 ICCOM Source Files

Location: rel\modules\cddiccom\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
CDD_Iccom.c	x	x	x	x	Product	This file contains CDDICCOM AUTOSAR APIs and Renesas APIs.
CDD_Iccom_Version.c	x	x	x	x	Product	This file contains the function that gets the version information.
CDD_Iccom_Ram.c	x	x	x	x	Product	This file contains the global variable used by the CDDICCOM.
CDD_Iccom_MainServ.c	x	x	x	x	Product	This file contains the message handling functionality.
CDD_Iccom_PBcfg.c	x	x	x	x	Product	This file contains the post-build configuration data.
MFIS	-	-	-	-	-	-
CDD_Iccom_MFIS_LLDriver.c	x	x	x	x	Product	This file contains the hardware access functionality.
CDD_Iccom_MFIS_Irq.c	x	x	x	x	Product	This file contains the interrupt service routines functionality.

x: applicable
-: not applicable

Table 3-116 shows the list of Parameter Definition Files for CDDICCOM module.

Table 3-116 ICCOM Parameter Definition Files

Location: rel\modules\cddiccom\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	S4	R1911_CDD_ICCOM_S4_RTM8RC79FR.arxml	RTM8RC79FR
		R1911_CDD_ICCOM_S4_RTM8RC79FG.arxml	RTM8RC79FG
	V4H	R1911_CDD_ICCOM_V4H.arxml	V4H
	V4M	R1911_CDD_ICCOM_V4M.arxml	V4M

Note Product Name**:** Product Names supported by "Files".

3.4.1.4 Configuration Parameter Dependency

Table 3-117 shows the list of configuration parameter dependency for ICCOM Driver component.

Table 3-117 ICCOM Driver Component Configuration Parameter Dependency

Parameter	Module	Path	Product Name
CDDICCOM_E_WRITE_VERIFY_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter	RTM8RC79FR RTM8RC79FG V4H
CDDICCOM_E_FATAL	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter	RTM8RC79FR RTM8RC79FG V4H
CDDICCOM_E_INIT_NEGOTIATION	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter	RTM8RC79FR RTM8RC79FG V4H
CDDICCOM_E_TIMEOUT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter	RTM8RC79FR RTM8RC79FG V4H
CDDICCOM_E_INVALID_ACK	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter	RTM8RC79FR RTM8RC79FG V4H
CDDICCOM_E_INT_INCONSISTENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter	RTM8RC79FG
CDDICCOM_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter	RTM8RC79FR V4H
CddIccomChannelTimeoutCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter	RTM8RC79FR RTM8RC79FG V4H

3.4.1.5 Source Code Dependency

The followings are the dependency files commonly used by the ICCOM Driver module:

- CddIccom_MemMap.h,
- Os.h,
- Det.h,
- Dem.h,
- Std_Types.h,
- Rte_CDD_Iccom.h,
- SchM_CddIccom.h

3.4.1.6 Stubs

Refer to 3.2.9 for the common stubs used for ICCOM Driver component.

3.4.1.7 Addition Error Handling

Refer to " Development and Production Errors" section at ICCOM chapter in Driver Component Embedded User's Manual.

3.4.1.8 Restrictions

None

3.4.1.9 Sample Application

3.4.1.9.1 Sample Application Structure

(1) S4 CR52

For more information refer to 3.7 Sample Application.

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddIccom_GetVersionInfo is invoked to get the version of the CDDICCOM Driver module with a variable of Std_VersionInfoType. After the call of this API the past parameter will get updated with the CDDICCOM Driver version details.
- The API CddIccom_Init is invoked with valid configuration address. This API performs the initialization of the ICCOM Complex Device Driver Component. This API initializes all the elements (Global Variables) of Global structure.
- The API CddIccom_Ch0SendRun is invoked 2048 times to send 2048 data with size from 1 to 2048 bytes to G4MH domain. Send data size is logged to DEBUG serial port.
- The API CddIccom_Ch1SendRun is invoked 2048 times to send 2048 data with size from 1 to 2048 bytes to AP-System Core. Send data size is logged to DEBUG serial port.
- The APIs CddIccom_Ch0ReceiveRun and CddIccom_Ch1ReceiveRun are invoked to receive data when having the notification from callback function. The receive data is checked. The result of reception task will be logged to DEBUG serial port after received 2048 bytes data package.
- The API scheduler function CddIccom_MainFunction_Send is invoked to perform periodic send processing to re-send the pending request (Send data/ACK).

(2) S4 G4MH

For more information refer to 3.7 Sample Application.

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddIccom_GetVersionInfo is invoked to get the version of the CDDICCOM Driver module with a variable of Std_VersionInfoType. After the call of this API the past parameter will get updated with the CDDICCOM Driver version details.
- The API CddIccom_Init is invoked with valid configuration address. This API performs the initialization of the ICCOM Complex Device Driver Component. This API initializes all the elements (Global Variables) of Global structure.
- The API CddIccom_Ch0SendRun is invoked 2048 times to send 2048 data with size from 1 to 2048 bytes to CR52 domain. Send data size is logged to DEBUG serial port.
- The API CddIccom_Ch1SendRun is invoked 2048 times to send 2048 data with size from 1 to 2048 bytes to AP-System Core. Send data size is logged to DEBUG serial port.

- The APIs CddIccom_Ch0ReceiveRun and CddIccom_Ch1ReceiveRun are invoked to receive data when having the notification from callback function. The receive data is checked. The result of reception task will be logged to DEBUG serial port after received 2048 bytes data package.
- The API scheduler function CddIccom_MainFunction_Send is invoked to perform periodic send processing to re-send the pending request (Send data/ACK).

(3) V4H

For more information refer to 3.7 Sample Application.

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddIccom_GetVersionInfo is invoked to get the version of the CDDICCOM Driver module with a variable of Std_VersionInfoType. After the call of this API the past parameter will get updated with the CDDICCOM Driver version details.
- The API CddIccom_Init is invoked with valid configuration address. This API performs the initialization of the ICCOM Complex Device Driver Component. This API initializes all the elements (Global Variables) of Global structure.

The API CddIccom_Ch*n*SendRun (*n* = 0) is invoked 2048 times to send 2048 data with size from 1 to 2048 bytes to AP-System Core *m* (*m* = 0). Send data size is logged to DEBUG serial port.

- The APIs CddIccom_*ChnReceiveRun* (*n* = 0) are invoked to receive data when having the notification from callback function. The receive data is checked. The result of reception task will be logged to DEBUG serial port after received 2048 bytes data package.
- The API scheduler function CddIccom_MainFunction_Send is invoked to perform periodic send processing to re-send the pending request (Send data/ACK).

(4) V4M

For more information refer to 3.7 Sample Application.

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddIccom_GetVersionInfo is invoked to get the version of the CDDICCOM Driver module with a variable of Std_VersionInfoType. After the call of this API the past parameter will get updated with the CDDICCOM Driver version details.
- The API CddIccom_Init is invoked with valid configuration address. This API performs the initialization of the ICCOM Complex Device Driver Component. This API initializes all the elements (Global Variables) of Global structure.

The API CddIccom_Ch*n*SendRun (*n* = 0) is invoked 2048 times to send 2048 data with size from 1 to 2048 bytes to AP-System Core *m* (*m* = 0). Send data size is logged to DEBUG serial port.

- The APIs CddIccom_*ChnReceiveRun* (*n* = 0) are invoked to receive data when having the notification from callback function. The receive data is checked. The result of reception task will be logged to DEBUG serial port after received 2048 bytes data package.
- The API scheduler function CddIccom_MainFunction_Send is invoked to perform periodic send processing to re-send the pending request (Send data/ACK).

3.4.1.9.2 **Recommended Environment**

(1) S4 CR52

- **S4_CR52 Environment**

Table 3-118 S4 CR52 Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

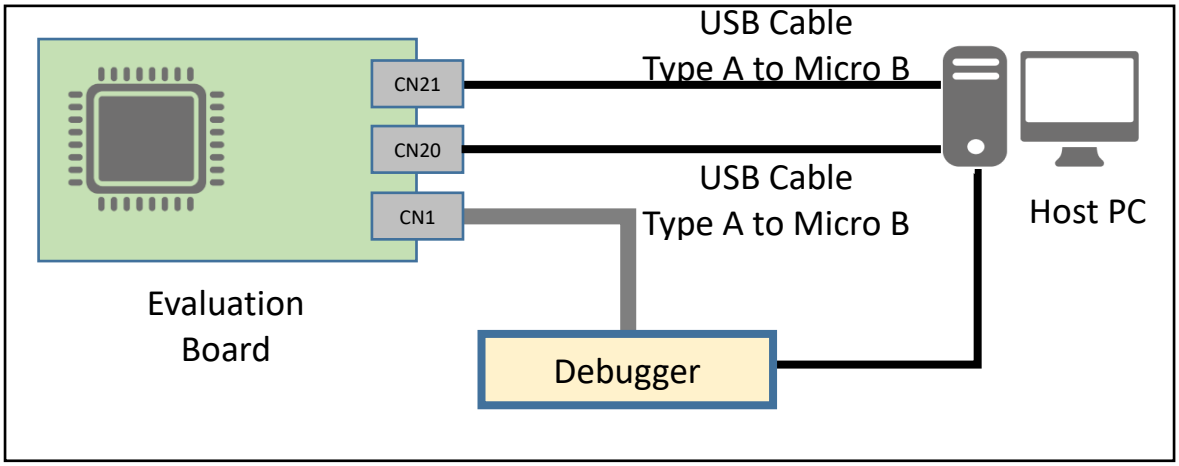

Figure 3-44 S4 Spider board connection of ICCOM Sample Application (CR52)

(2) S4 G4MH

- **S4_G4MH Environment**

Table 3-119 S4 G4MH Environment

Name	Explanation
Evaluation Board	R-Car S4 System Evaluation Board (Spider) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (E2 Renesas)
Debugger Software	CS+ for CC E8.07.00c
Terminal Software	Teraterm

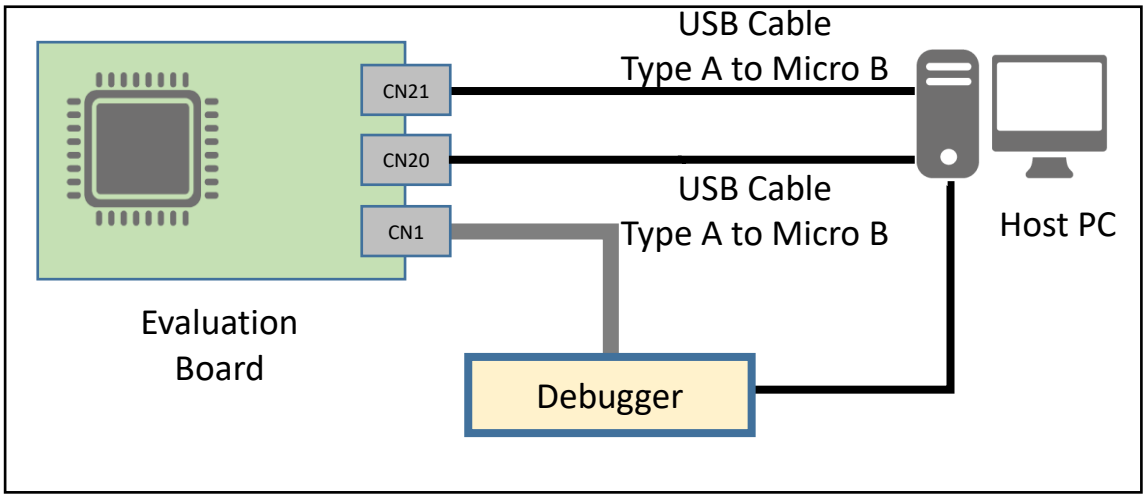

Figure 3-45 S4 Spider board connection of ICCOM Sample Application (G4MH)

(3) V4H

- **V4H Environment**

Table 3-120 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

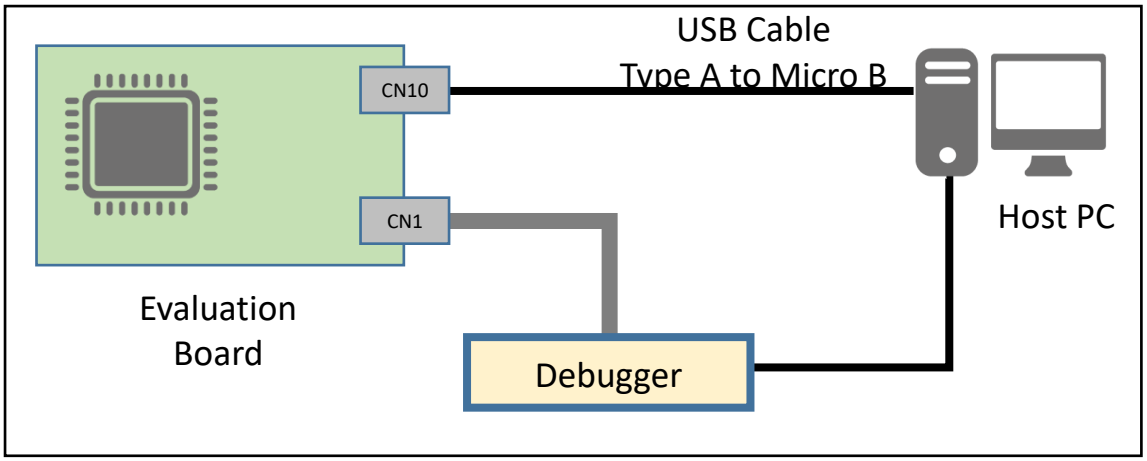

Figure 3-46 V4H White Hawk board connection of ICCOM Sample Application

(3) V4M

- **V4M Environment**

Table 3-121 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

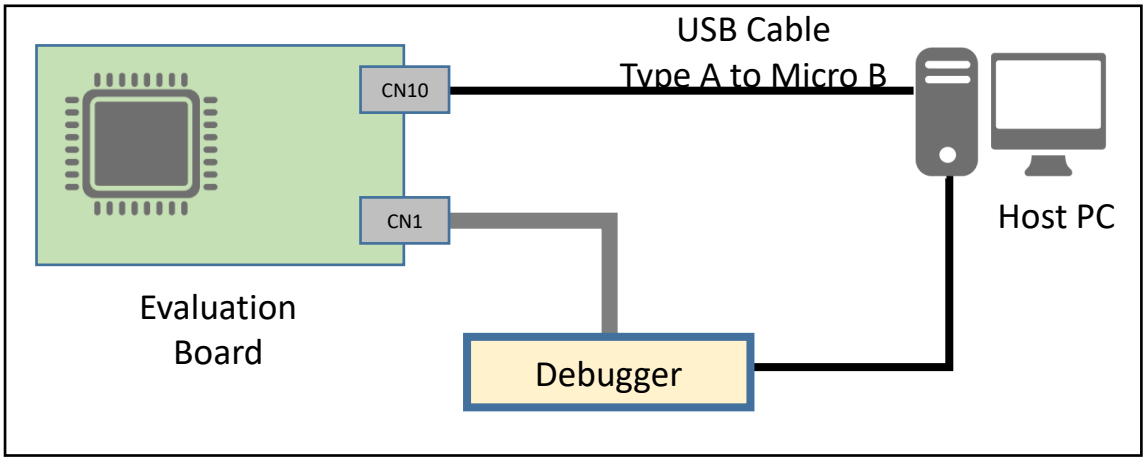

Figure 3-47 V4M Gray Hawk board connection of ICCOM Sample Application

3.4.1.9.3 **Preparation**

(1) S4 CR52

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- MCAL ICCOM Sample Application uses System Up-Time Clock (SUCMT) for scheduler function CddIccom_MainFunction_Send().

(2) S4 G4MH

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- MCAL ICCOM Sample Application uses System Up-Time Clock (SUCMT) for scheduler function CddIccom_MainFunction_Send().

(3) V4H

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H, read more environment setup and guideline at 3.7.3 How to run Sample Application.
- MCAL ICCOM Sample Application uses:
  - System Up-Time Clock (SUCMT) for scheduler function CddIccom_MainFunction_Send().
  - Timer Unit 0 (TMU0), channel 1 for Os stub (GetCounterValue() and GetElapsedValue() functions).

(4) V4M

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.
- MCAL ICCOM Sample Application uses:
  - System Up-Time Clock (SUCMT) for scheduler function CddIccom_MainFunction_Send().
  - Timer Unit 0 (TMU0), channel 1 for Os stub (GetCounterValue() and GetElapsedValue() functions).

3.4.1.9.1 **How to build sample application**

(1) S4 CR52

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\S4\common_family\make\arm

- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat CddIccom R19-11 S4 No

(2) S4 G4MH

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\S4\common_family\make\ghs

- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat CddIccom R19-11 S4 No

(3) V4H

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\V4H\common_family\make\arm

- Now execute the batch file Sample App.bat with folowing parameters:

  SampleApp.bat CddIccom R19-11 V4H No

(4) V4M

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\V4M\common_family\make\arm

- Now execute the batch file Sample App.bat with folowing parameters:

  SampleApp.bat CddIccom R19-11 V4M No

3.4.1.9.5 **How to run sample application**

(1) S4 CR52

1. Preparing the G4MH CDDICCOM Sample Application and AP-System core ICCOM Sample Application stub program in RCar S4 evaluation board.
2. The ICCOM Sample Application runs on AP-System core domain is configured to send initialization signal to CR52 core domain.
3. The CDDICCOM Sample Application runs on CR52 core domain is configured to send initialization signal to G4MH core domain.
4. Setting breakpoint at the block “Waiting for Channel 0 Ready”.
5. Run program to breakpoint, if CddIccom_GetVersionInfo API returns correct version information, variable GucVerCheckStatus will be set to CDDICCOM_TRUE. Otherwise, variable GucVerCheckStatus will be set to CDDICCOM_FALSE. At this step CDDICCOM driver is initialized.
6. Setting breakpoint at the block “Waiting for Channel 1 Ready”.
7. Continue the program, if CDDICCOM Sample application is successful send initialization signal to G4MH domain, it wills perform transmission/reception data between CR52 and G4MH core domain via CddIccom Channel 0. Otherwise, the program will be stuck at while loop command in block “Waiting for Channel 0 Ready”.
8. When the program stops at break point:

- Confirming the transmission status by CddIccom Channel 0: *GaaSndStatus[0..2047]* are set to CDDICCOM_TRUE.
- Confirming the reception status by CddIccom Channel 0: *GaaRcvStatus[0..2047]* are set to CDDICCOM_TRUE if all data reception correctness. Otherwise, *GaaRcvStatus[n]* is set to CDDICCOM_FALSE (n is error message order number).
- Confirming the reception error message position by CddIccom Channel 0: *Error_Position* variable is set to DEFAULT_ERROR_POSITION (no error). Otherwise, *Error_Position* variable’s value indicates position of the first error message.
- Confirming the Sample Application status: *GaaTestResult[0..4]*, *GaaTestResult[5]* and *GaaTestResult[6]* are set to CDDICCOM_TRUE.

9. Continue the program, if CDDICCOM Sample application received initialization from AP-System core, it wills perform transmission/reception data between CR52 and AP-System core domain via CddIccom Channel 1. Otherwise, the program will be stuck at while loop command in block “Waiting for Channel 1 Ready”.
10. At the end of program, Sample Application sequence is completed

- Confirming the transmission status by CddIccom Channel 1: *GaaSndStatus_App[0..2047]* are set to CDDICCOM_TRUE.
- Confirming the reception status by CddIccom Channel 1: *GaaRcvStatus_App[0..2047]* are set to CDDICCOM_TRUE if all data reception correctness. Otherwise, *GaaRcvStatus_App[n]* is set to CDDICCOM_FALSE (n is error message order number).
- Confirming the reception error message position by CddIccom Channel 1: Error_Position_App variable is set to DEFAULT_ERROR_POSITION (no error). Otherwise, Error_Position variable’s value indicates position of the first error message.
- Confirming the Sample Application status: *GaaTestResult[0..9]* are set to CDDICCOM_TRUE.

Note: The sample application execution and result can be check by print logs throught the Teraterm if it is used:
“PROGRAM START”: Sample application execution is started.
“EXECUTED OK”: Sample application execution is successful.
“EXECUTED NOT OK”: Sample application execution is failed.
“PROGRAM STOP”: Sample application execution is completed.

(2) S4 G4MH

1. Preparing the CR52 CDDICCOM Sample Application and AP-System core ICCOM Sample Application stub program in RCar S4 evaluation board.
2. The ICCOM Sample Application runs on AP-System core domain is configured to send initialization signal to G4MH core domain.
3. The CDDICCOM Sample Application runs on CR52 core domain is configured to send initialization signal to G4MH core domain.
4. Setting breakpoint at the block “Waiting for Channel 0 Ready”.
5. Run program to breakpoint, if CddIccom_GetVersionInfo API returns correct version information, variable GucVerCheckStatus will be set to CDDICCOM_TRUE. Otherwise, variable GucVerCheckStatus will be set to CDDICCOM_FALSE. At this step CDDICCOM driver is initialized.
6. Setting breakpoint at the block “Waiting for Channel 1 Ready”.
7. Continue the program, if CDDICCOM Sample application received initialization signal from CR52 core domain, it wills perform transmission/reception data between G4MH and CR52 core domain via CddIccom Channel 0. Otherwise, the program will be stuck at while loop command in block “Waiting for Channel 0 Ready”.
8. When the program stops at break point:
   - Confirming the transmission status by CddIccom Channel 0: GaaSndStatus[0..2047] are set to CDDICCOM_TRUE.
   - Confirming the reception status by CddIccom Channel 0: GaaRcvStatus[0..2047] are set to CDDICCOM_TRUE if all data reception correctness. Otherwise, GaaRcvStatus[n] is set to CDDICCOM_FALSE (n is error message order number).
   - Confirming the reception error message position by CddIccom Channel 0: Error_Position variable is set to DEFAULT_ERROR_POSITION (no error). Otherwise, Error_Position variable’s value indicates position of the first error message.
   - Confirming the Sample Application status: GaaTestResult[0..4], GaaTestResult[5] and GaaTestResult[6] are set to CDDICCOM_TRUE.
9. Continue the program, if CDDICCOM Sample application received initialization from AP-System core, it wills perform transmission/reception data between G4MH and AP-System core domain via CddIccom Channel 1. Otherwise, the program will be stuck at while loop command in block “Waiting for Channel 1 Ready”.
10. At the end of program, Sample Application sequence is completed
    - Confirming the transmission status by CddIccom Channel 1: GaaSndStatus_App[0..2047] are set to CDDICCOM_TRUE.

- Confirming the reception status by CddIccom Channel 1: GaaRcvStatus_App[0..2047] are set to CDDICCOM_TRUE if all data reception correctness. Otherwise, GaaRcvStatus_App[n] is set to CDDICCOM_FALSE (n is error message order number).
- Confirming the reception error message position by CddIccom Channel 1: Error_Position_App variable is set to DEFAULT_ERROR_POSITION (no error). Otherwise, Error_Position variable's value indicates position of the first error message.
- Confirming the Sample Application status: GaaTestResult[0..9] are set to CDDICCOM_TRUE.

Note: The sample application execution and result can be check by print logs throught the Teraterm if it is used:
"PROGRAM START": Sample application execution is started.
"EXECUTED OK": Sample application execution is successful.
"EXECUTED NOT OK": Sample application execution is failed.
"PROGRAM STOP": Sample application execution is completed.

(3) V4H

1. Preparing the AP-System core ICCOM Sample Application stub program in RCar V4H evaluation board.
2. The ICCOM Sample Application runs on AP-System core domain is configured to send initialization signal to CR52 domain.
3. Setting breakpoint at the block "Waiting for Channel 0 Ready".
4. Run program to breakpoint, if CddIccom_GetVersionInfo API returns correct version information, variable GucVerCheckStatus will be set to CDDICCOM_TRUE. Otherwise, variable GucVerCheckStatus will be set to CDDICCOM_FALSE. At this step CDDICCOM driver is initialized.
5. Continue the program, if CDDICCOM Sample application received initialization from AP-System core, it wills perform communication with AP-System core domain via CddIccom Channel 0. Otherwise, the program will be stuck at while loop command in block "Waiting for Channel 0 Ready"
6. At the end of program, Sample Application sequence is completed:

- Confirming the transmission status by CddIccom Channel 0: *GaaSndStatus[0..2047]* are set to CDDICCOM_TRUE.
- Confirming the reception status by CddIccom Channel 0: *GaaRcvStatus[0..2047]* are set to CDDICCOM_TRUE if all data reception correctness. Otherwise, *GaaRcvStatus[n]* is set to CDDICCOM_FALSE (n is error message order number).
- Confirming the reception error message position by CddIccom Channel 0: Error_Position variable is set to DEFAULT_ERROR_POSITION (no error).
- Confirming the Sample Application status: *GaaSampleAppStatus[0..6]* are set to CDDICCOM_TRUE.

Note: The sample application execution and result can be check by print logs throught the Teraterm if it is used:
"PROGRAM START": Sample application execution is started.
"EXECUTED OK": Sample application execution is successful.
"EXECUTED NOT OK": Sample application execution is failed.
"PROGRAM STOP": Sample application execution is completed.

(4) V4M

1. Preparing the AP-System core ICCOM Sample Application stub program in RCar V4M evaluation board.
2. The ICCOM Sample Application runs on AP-System core domain is configured to send initialization signal to CR52 domain.
3. Setting breakpoint at the block "Waiting for Channel 0 Ready".
4. Run program to breakpoint, if CddIccom_GetVersionInfo API returns correct version information, variable GucVerCheckStatus will be set to CDDICCOM_TRUE. Otherwise, variable GucVerCheckStatus will be set to CDDICCOM_FALSE. At this step CDDICCOM driver is initialized.

5. Continue the program, if CDDICCOM Sample application received initialization from AP-System core, it wills perform communication with AP-System core domain via CddIccom Channel 0. Otherwise, the program will be stuck at while loop command in block “Waiting for Channel 0 Ready”
6. At the end of program, Sample Application sequence is completed:

- Confirming the transmission status by CddIccom Channel 0: *GaaSndStatus[0..2047]* are set to CDDICCOM_TRUE.
- Confirming the reception status by CddIccom Channel 0: *GaaRcvStatus[0..2047]* are set to CDDICCOM_TRUE if all data reception correctness. Otherwise, *GaaRcvStatus[n]* is set to CDDICCOM_FALSE (n is error message order number).
- Confirming the reception error message position by CddIccom Channel 0: Error_Position variable is set to DEFAULT_ERROR_POSITION (no error).
- Confirming the Sample Application status: *GaaSampleAppStatus[0..6]* are set to CDDICCOM_TRUE.

Note: The sample application execution and result can be check by print logs throught the Teraterm if it is used:
“PROGRAM START”: Sample application execution is started.
“EXECUTED OK”: Sample application execution is successful.
“EXECUTED NOT OK”: Sample application execution is failed.
“PROGRAM STOP”: Sample application execution is completed.

3.4.1.9.6 **How to change the serial console setting of MCAL ICCOM sample application**

For V4H:

In header file ‘rel/V4H/common_family/include/arm/device_cfg.h’

When using with another baudrate and SCIF setting, change RCAR_MCAL_LOG_SELECT macro value.
From
```
#define RCAR_MCAL_LOG_SELECT RCAR_HSCIF_921600BPS
```
To
```
#define RCAR_MCAL_LOG_SELECT RCAR_SCIF_115200BPS
```

For V4M:

In header file ‘rel/V4M/common_family/include/arm/device_cfg.h’

When using with another baudrate and SCIF setting, change RCAR_MCAL_LOG_SELECT macro value.
From
```
#define RCAR_MCAL_LOG_SELECT RCAR_HSCIF_921600BPS
```
To
```
#define RCAR_MCAL_LOG_SELECT RCAR_SCIF_115200BPS
```

3.4.1.10 **ROM/RAM Usage**

See Appendix CDDICCOM section for information on measuring RAM/ROM consumption.

3.4.1.11 **Stack Depth**

See Appendix CDDICCOM section for information on measuring stack depth.

3.4.1.12 **Throughput Details**

See Appendix CDDICCOM section for information on measuring execution time, and functional testing.

3.4.2 RFSO Driver Component

3.4.2.1 Module Overview

The RFSO (Failure Self-Detection Output) is a 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.

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

- 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 ECM module and related bits control and examination
- Get version information

3.4.2.2 Module Dependency

The dependency of RFSO Driver on other modules and the required implementation is briefed as follows:

DET
In development mode the Default Error Tracer (DET) will be called whenever RFSO module encounters a development error.

DEM
DEM will be called whenever this module encounters a production error.

RTE
The Run time Environment (RTE) module will be called whenever a critical section protection function is called.

3.4.2.3 **Folder Strucure**

Table 3-123 and **Table 3-122** show the list of Source Code Files and Header Files for RFSO module.

Table 3-122 RFSO Header Files

Location: rel\modules\cddrfso	**Supported Device**				**Category**	**Description**
	S4 (Includes S4N) CR52	**S4 (Includes S4N) G4MH**	**V4H**	**V4M**		
include	-	-	-	-	-	-
CDD_Rfso.h	-	-	x	x	Product	This file provides externs declaration, type definition and config pointer for global as well as initialization of CDDRFSO.
CDD_Rfso_Ctr.h	-	-	x	x	Product	This file provides the provision of external declaration and services for APIs.
CDD_Rfso_Irq.h	-	-	x	x	Product	This file provides the provision of external declaration and services for ISR.
CDD_Rfso_Reg.h	-	-	x	x	Product	This file provides the provision of RFSO register structure and register definitions.
CDD_Rfso_Ram.h	-	-	x	x	Product	C Header file for Cdd_Rfso_Ram.c. It includes the definitions of global variables.
CDD_Rfso_Types.h	-	-	x	x	Product	This file provides definitions of CDD RFSO data types.
CDD_Rfso_PBTypes.h	-	-	x	x	Product	This file provides the definition for Post-Build configuration types and setting.
CDD_Rfos_Cfg.h	-	-	x	x	Product	This file contains various driver Pre-compile time parameters. It also contains the macro for the total number of channels configured, acknowledgement timeout and protection area.
Cdd_Rfso_Cbk.h	-	-	x	x	Product	This file contains call-back function definition.
Cdd_Rfso_Version.h	-	-	x	x	Product	C Header

x: applicable

-: not applicable

Table 3-123 RFSO Source Files

Location: rel\modules\cddrfso\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
CDD_Rfso.c	-	-	x	x	Product	This file contains the function that Provision of initialization and control functionality.
CDD_Rfso_Ctr.c	-	-	x	x	Product	This file contains the control services for CDD RFSO.
CDD_Rfso_Irq.c	-	-	x	x	Product	This file contains the interrupt services for CDD RFSO.
CDD_Rfso_Ram.c	-	-	x	x	Product	This file contains the global variables declaration.
CDD_Rfso_Version.c	-	-	x	x	Product	This file contains the function to get version information of the CDD RFSO.
CDD_Rfso_PBcfg.c	-	-	x	x	Product	This file contains the post-build configuration data.

x: applicable
-: not applicable

3.4.2.4 Configuration Parameter Dependency

Table 3-124 shows the list of configuration parameter dependency for RFSO Driver component.

Table 3-124 RFSO Driver Component Configuration Parameter Dependency

Parameter	Module	Path
CDDRFSO_E_WRITE_VERIFY	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDRFSO_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CddRfsoDemEventParameterRefs	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.4.2.5 Source Code Dependency

The followings are the dependency files commonly used by the RFSO Driver module:

- Dem.h,
- Det.h,
- CddRfso_MemMap.h,
- Platform_Types.h,
- Std_Types.h,
- SchM_CddRfso.h,
- Os.h,
- RCar_Types.h
- Rte_CddRfso.h

3.4.2.6 Stubs

Refer to 3.2.9 for the common stubs used for RFSO Driver component.

3.4.2.7 Addition Error Handling

Refer to " Development and Production Errors" section at RFSO chapter in Driver Component Embedded User's Manual.

3.4.2.8 Restrictions

Refer to "Additional Error Handling and Restriction" section at RFSO chapter in Driver Component Embedded User's Manual.

3.4.2.9 Sample Application

3.4.2.9.1 Sample Application Structure

For more information refer to 3.7 Sample Application.

- Initialize Interrupt driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddRfso_GetVersionInfo is invoked to get the version of the CDDRFSO Driver module with a variable of Std_VersionInfoType. After the call of this API the previous parameter will get updated with the CDDRFSO Driver version details.
- The API CddRfso_Init is invoked with valid configuration address. This API performs the initialization of the RFSO Complex Device Driver Component. This API initializes all of the elements (Global Variables) of Global structure.
- Use the Interval timer and Time-out timer in RFSO for the following two usages:

a) Periodical Check

- In the usage of Periodical Check, we should set up the "Interval Timer Duration" smaller than "Timeout Timer Max Duration". "Time-out Timer Min Duration" should be set as zero.
- The API CddRfso_CtrlIntervalTimerInterrupt is invoked to enable Interval interrupt.
- The API CddRfso_ChannelClockSet is invoked to set division value for clock of RFSO channel.

- The API CddRfso_IntervalCycleConfigure is invoked to configure Interval Timer of selected channel (the input timer unit is cycle).
- The API CddRfso_TimeoutCycleConfigure is invoked to configure Time-out Timer of selected channel (the input timer unit is cycle).
- Periodical Check process is implemented by invoking CddRfso_PeriodicalCheck.
- In function CddRfso_PeriodicalCheck:
    - The API CddRfso_StartIntervalTimer is invoked to start Interval Timer.
    - The API CddRfso_IntervalTimerInterruptStatus is invoked to check whether the Interval Interrupt is raised.
    - The Periodical Check procedure is implemented 6 times.
    - The API CddRfso_IntervalTimerInterruptStatus is invoked to check whether the Interval Timer interrupt flag is already cleared.
    - The API CddRfso_ClearTimeoutInterrupt is invoked to clear Time-out interrupt flag.
    - The API CddRfso_StartTimeoutTimer is invoked to start Time-out Timer.
    - The API CddRfso_GetIntervalTimerValue is invoked to read the current value of Interval Timer counter.
    - The API CddRfso_GetTimeoutTimerValue is invoked to read the current value of Time-out Timer counter.
    - The API CddRfso_GetTOESPinStatus is invoked to get the status of bits TOES, TOI, TOCUNF in order to check whether the Time-out error occurs.
    - If the time-out error occurs, the API CddRfso_ExternalPinControl is invoked to support control CFEO_0 and CFEO_1. This step aims to report Time-out error to ECM module.
    - The API CddRfso_GetCFEPinStatus is invoked to get status of bits CFES_0, CFES_1, CFEO_0, CFEO_0. This step aims to check feedback value from module ECM.
    - After getting status of CFES_0, CFES_1, CFEO_0, CFEO_0 bits, check whether the error is raised to ECM module successfully.
    - After 6 times of Interval interrupt, the API CddRfso_StopIntervalTimer is invoked to stop Time-out Timer.

b) Runtime Test

- In the usage of Runtime Test, we should set up the “Interval Timer Duration” larger than “Timeout Timer Max Duration”. “Time-out Timer Min Duration” should be set as different from zero.
- The API CddRfso_CtrlIntervalTimerInterrupt is invoked to enable Interval Interrupt.
- The API CddRfso_ChannelClockSet is invoked to divide the clock frequency.
- The API CddRfso_IntervalTimeConfigure is invoked to set up Interval Timer of selected channel (the input timer unit is microsecond).
- The API CddRfso_TimeoutTimeConfigure is invoked to set up Time-out Timer of selected channel (the input timer unit is microsecond).
- RunTime Check process is implemented by invoking CddRfso_RunTimeTest.
- In function CddRfso_RunTimeTest:
    - Invoke CddRfso_StartIntervalTimer to start Interval Timer.
    - Invoke CddRfso_IntervalTimerInterruptStatus to check whether the Interval Interrupt is raised.
    - The Runtime Test procedure is implemented for once.
    - After Interval Timer interrupt is raised, invoke CddRfso_IntervalTimerInterruptStatus to check whether the Interval Timer interrupt flag is already cleared.
    - The API CddRfso_ClearTimeoutInterrupt is invoked to clear Time-out interrupt flag.
    - The API CddRfso_StartTimeoutTimer is invoked to start Time-out Timer.
    - The API CddRfso_GetIntervalTimerValue is invoked to read the current value of Interval Timer counter.

- The API CddRfso_GetTimeoutTimerValue is invoked to read the current value of Time-out Timer counter.
- The API CddRfso_StopTimeoutTimer is invoked to stop Time-out Timer.
- The API CddRfso_GetTOESPinStatus is invoked to get the status of bits TOES, TOI, TOCUNF in order to check whether the time-out error occurs
- If the Time-out error occurs, confirm whether the Time-out stopped under the minimum time or over the maximum time by checking the value of bit TOCUNF.
- The API CddRfso_ExternalPinControl is invoked to support control CFEO_0 and CFEO_1. This step aims to report Time-out error to ECM module.
- The API CddRfso_GetCFEPinStatus is invoked to get status of bits CFES_0, CFES_1, CFEO_0, CFEO_0. This step aims to check feedback value of module ECM.
- Invoke CddRfso_GetCFEPinStatus to get status of bits CFES_0, CFES_1, CFEO_0, CFEO_0. This step aims to check feedback value of module ECM.

3.4.2.9.2 **Recommended Environment**

- **V4H Environment**

Table 3-125 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

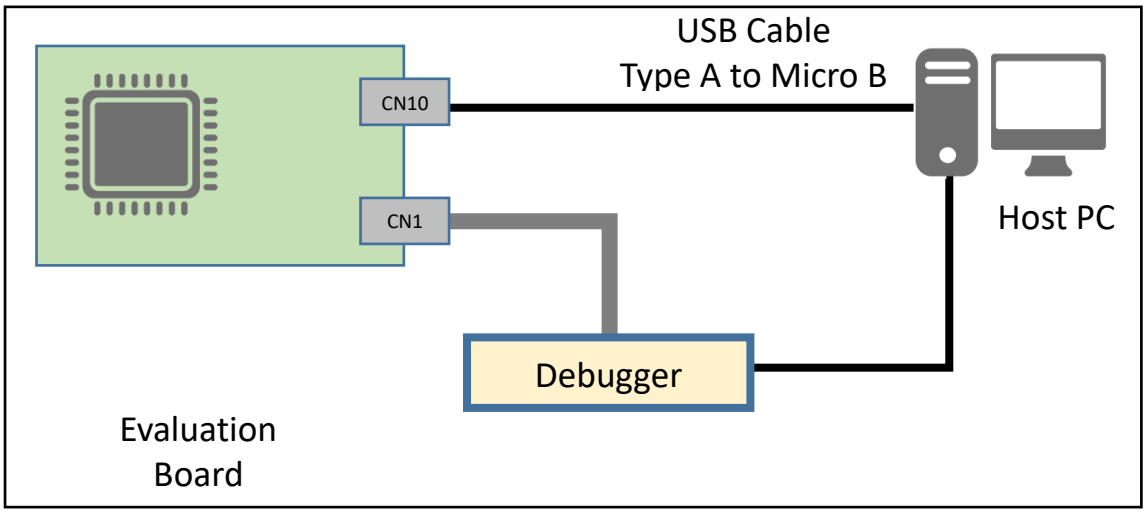

Figure 3-48 V4H White Hawk board connection of RFSO Sample Application

- **V4M Environment**

Table 3-126 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)

Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

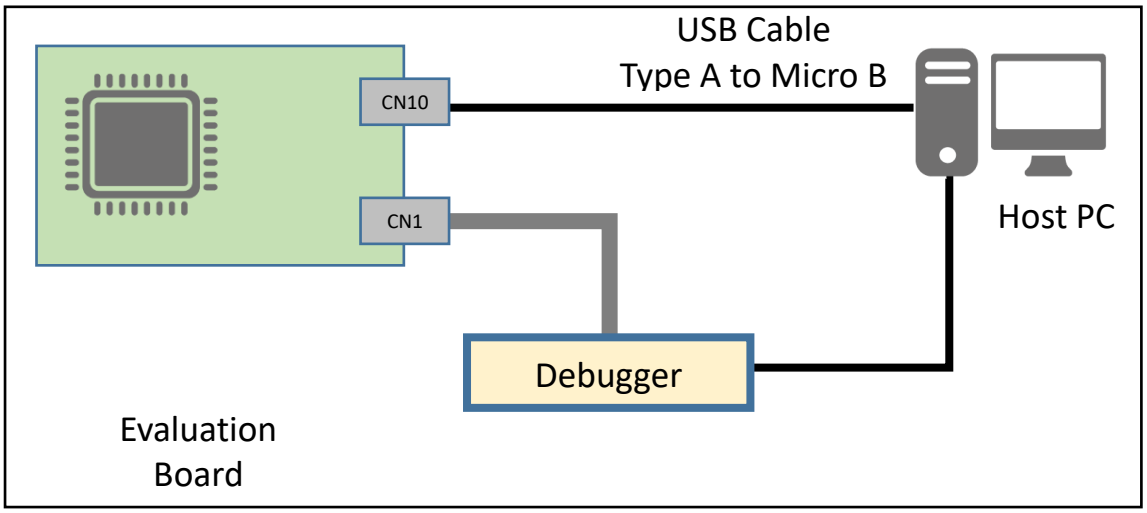

Figure 3-49 V4M Gray Hawk board connection of RFSO Sample Application

3.4.2.9.3 **Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.4.2.9.4 **How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:
  For V4H: external\rel\V4H\common_family\make\arm
  For V4M: external\rel\V4M\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:
  SampleApp.bat CddRfso R19-11 V4H No
  For V4M: SampleApp.bat CddRfso R19-11 V4M No

3.4.2.9.5 **How to run sample application**

1/ Plug in serial port to Host PC, open the Terminal Software (Teraterm)
2/ Run the program and check sample application in print logs of the Teraterm:
“PROGRAM START”: Sample application execution is started.
“EXECUTED OK”: Sample application execution is successful.
“EXECUTED NOT OK”: Sample application execution is failed.
“PROGRAM STOP”: Sample application execution is completed.

3.4.2.10 **ROM/RAM Usage**

See Appendix CDDRFSO section for information on measuring RAM/ROM consumption.

3.4.2.11 **Stack Depth**

See Appendix CDDRFSO section for information on measuring stack depth.

3.4.2.12 **Throughput Details**

See Appendix CDDRFSO section for information on measuring execution time, and functional testing.

3.4.3 IIC Driver Component

3.4.3.1 Module 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

3.4.3.2 Module Dependency

The dependency of IIC Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

RTE
The Run Time Environment (RTE) module will be called to perform the communication with connected AUTOSAR Software Component.

OS
As the IIC Driver uses interrupts, it depends on the OS, which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

EcuM
IIC Driver initialization function shall be exclusively called by the EcuM.

SchM
The Basic Software Scheduler module will be called whenever a critical section protection function is called.

3.4.3.3 **Folder Structure**

Table 3-127 and **Table 3-128** show the list of Source Code Files and Header Files for IIC module.

Table 3-127 IIC Header Files

Location: rel\modules\cddiic\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
CDD_Iic.h	-	-	x	x	Product	This file provides externs declaration, type definition and config pointer for global as well as initialization of IIC CDD.
CDD_Iic_PBTypes.h	-	-	x	x	Product	This file provides the definition for Post-Build configuration types and setting.
CDD_Iic_Ram.h	-	-	x	x	Product	C headers file for CDD_Iic_Ram.c. It includes the definitions that are required for RAM data.
CDD_Iic_Types.h	-	-	x	x	Product	This file provides the definition of CDD IIC data types
CDD_Iic_Version.h	-	-	x	x	Product	C header file for CDD_Iic_Version.c. It includes the definitions that are required for CDD_Iic_Version.h
CDD_Iic_HalDriver.h	-	-	x	x	Product	C headers file for CDD_Iic_HalDriver.c. It includes the definitions that are required for CddIic_HalDriver.c
CDD_Iic_Internal.h	-	-	x	x	Product	C headers file for CDD_Iic_Internal.c. It includes the definitions that are required for CddIic_Internal.c
CDD_Iic_Irq.h	-	-	x	x	Product	C headers file for CDD_Iic_Irq.c. It includes the definition that are required for CddIic_Irq.c

x: applicable

-: not applicable

Table 3-128 IIC Source Files

Location: rel\modules\cddiic\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
CDD_Iic.c	-	-	x	x	Product	This file contains the function that Initializes the IIC CDD component and specific settings Get the version information SWC runnable entities BSW runnable entities
CDD_Iic_Ram.c	-	-	x	x	Product	This file contains the global variable used by the IIC CDD.
CDD_Iic_Version.c	-	-	x	x	Product	This file contains the information about IIC version.
CDD_Iic_HalDriver.c	-	-	x	x	Product	This file contains the hardware access functionality.
CDD_Iic_Internal.c	-	-	x	x	Product	This file contains the message handling functionality.
CDD_Iic_Irq.c	-	-	x	x	Product	This file contains the interrupt service routines functionality.

x: applicable
-: not applicable

Table 3-129 shows the list of Parameter Definition Files for CDDIIC module.

Table 3-129 IIC Parameter Definition Files

Location: rel\modules\cddiic\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	V4H	R1911_CDD_IIC_V4H.arxml	V4H
19_11	V4M	R1911_CDD_IIC_V4M.arxml	V4M

Note Product Name**:** Product Names supported by "Files".

3.4.3.4 Configuration Parameter Dependency

Table 3-130 shows the list of configuration parameter dependency for IIC Driver component.

Table 3-130 IIC Driver Component Configuration Parameter Dependency

Parameter	Module	Path
CDDIIC_E_NON_ACKNOWLEGEMENT	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDIIC_E_WRITE_VERIFY	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDIIC_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.4.3.5 Source Code Dependency

The followings are the dependency files commonly used by the IIC Driver module:

- Dem.h,
- Det.h,
- CddIic_MemMap.h,
- Platform_Types.h,
- Std_Types.h,
- SchM_CddIic.h,
- RCar_Types.h,
- Compiler_Cfg.h.

3.4.3.6 Stubs

Refer to 3.2.9 for the common stubs used for IIC Driver component.

3.4.3.7 Addition Error Handling

Refer to " Development and Production Errors" section at IIC chapter in Driver Component Embedded User's Manual.

3.4.3.8 Restrictions

None

3.4.3.9 Sample Application

❖ **V4H:**

3.4.3.9.1 Sample Application Structure

For more information refer to 3.7 Sample Application.

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddIic_Init() is invoked with valid configuration address. This API performs the initialization of the IIC Complex Driver Component. This API initializes all the elements (Global Variables) of Global structure.

- The API CddIic_Ch0Write() is invoked to send 32 bytes to slave. Send data size is monitor at slave device.
- The API CddIic_Ch0Read() is invoked to receive 32 bytes from slave. Receive data size is monitor at master device.
- The API CddIic_Ch0WriteRead() is invoked to send/receive 32 bytes from/to slave. Send/Receive data size is monitor at master/slave device.

❖ **V4M:**

For more information refer to 3.7 Sample Application.

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddIic_Init() is invoked with valid configuration address. This API performs the initialization of the IIC Complex Driver Component. This API initializes all the elements (Global Variables) of Global structure.
- The API CddIic_Ch1SlaveInit is invoked to initialize slave at channel 1.
- The API CddIic_Ch0Write() is invoked to send 32 bytes to slave. Send data size is monitor at slave device.
- The API CddIic_Ch0Read() is invoked to receive 32 bytes from slave. Receive data size is monitor at master device.
- The API CddIic_Ch0WriteRead() is invoked to send/receive 32 bytes from/to slave. Send/Receive data size is monitor at master/slave device.

3.4.3.9.2 **Recommended Environment**

- **V4H Environment**

Table 3-131 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

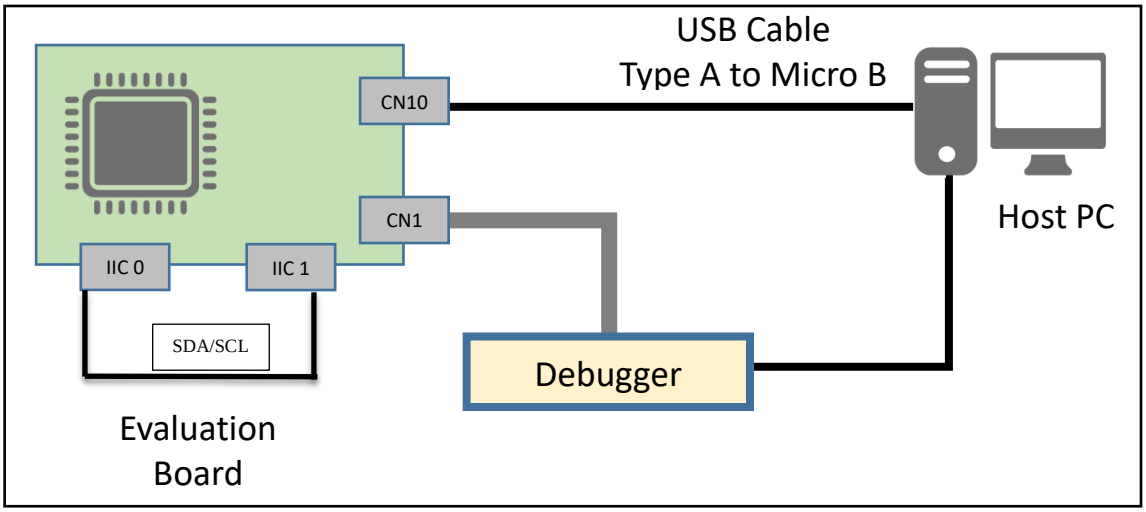

Figure 3-50 V4H White Hawk board connection of IIC Sample Application

- **V4M Environment**

Table 3-132 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

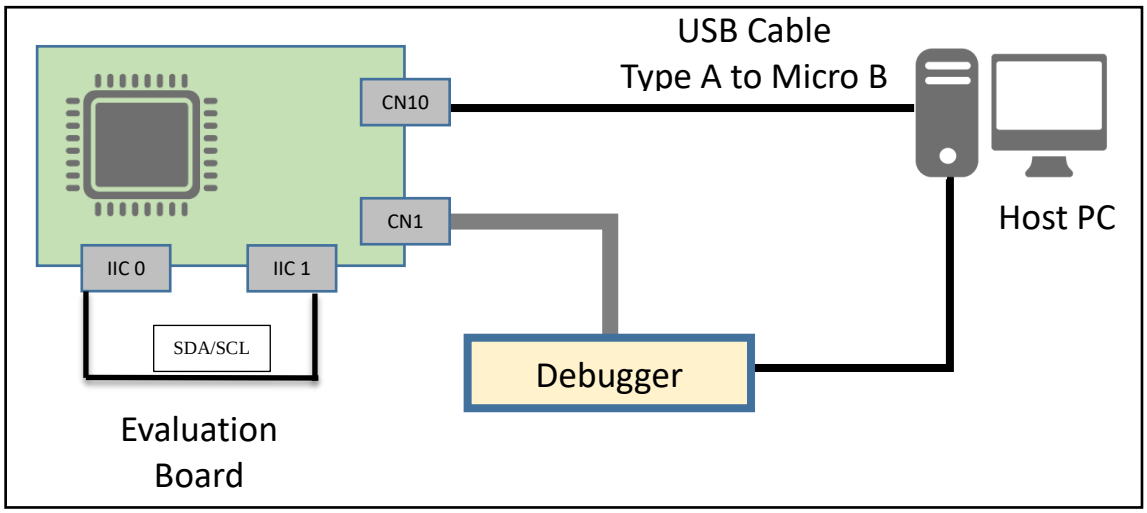

Figure 3-51 V4M Gray Hawk board connection of IIC Sample Application

3.4.3.9.3 **Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.4.3.9.4 **How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:
  For V4H: external\rel\V4H\common_family\make\arm
  For V4M: external\rel\V4M\common_family\make\arm

Now execute the batch file Sample App.bat with following parameters:

For V4H: SampleApp.bat Cddiic R19-11 V4H No

For V4M: SampleApp.bat Cddiic R19-11 V4M No

3.4.3.9.5 **How to run sample application**

1/ Connect the electric wire of IIC hardware channel 0 (master) to hardware channel 1 (slave)

2/ Plug in serial port to Host PC, open the Terminal Software (Teraterm)

3/ Run the program and check sample application in print logs of the Teraterm:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

"PROGRAM STOP": Sample application execution is completed.

3.4.3.10 ROM/RAM Usage

See Appendix CDDIIC section for information on measuring RAM/ROM consumption.

3.4.3.11 Stack Depth

See Appendix CDDIIC section for information on measuring stack depth.

3.4.3.12 Throughput Details

See Appendix CDDIIC section for information on measuring execution time, and functional testing.

3.4.4 IPMMU Driver Component

3.4.4.1 Module Overview

This document is intended for the developers of ECU software on R-Car Series, 4rd 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

3.4.4.2 Module Dependency

The dependency of IPMMU Driver on other modules and the required implementation is briefed as follows:

DET

In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM

Production errors will be reported to the Diagnostic Event Manager (DEM).

RTE

The Run Time Environment (RTE) module will be called to perform the communication with connected AUTOSAR Software Component.

OS

As the IPMMU Driver uses interrupts, it depends on the OS, which configures the interrupt sources. If OS is not available, then the configuration of interrupt sources shall be stubbed.

EcuM

IPMMU Driver initialization function shall be exclusively called by the EcuM.

SchM

The Basic Software Scheduler module will be called whenever a critical section protection function is called

3.4.4.3 Folder Structure

These below tables show the list of Source Code Files and Header Files for IPMMU module.

Table 3-133 IPMMU Header Files

Location:	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
rel\modules\cddipmmu\include	-	-	-	-	-	-
CDD_Ipmmu.h	-	-	x	x	Product	This file provides externs declaration, type definition and configure pointer for global as well as initialization of CDD IPMMU.
CDD_Ipmmu_Version.h	-	-	x	x	Product	C headers file for CDD_Ipmmu_Version.c. It includes the definitions for get version info function.
CDD_Ipmmu_Ctr.h	-	-	x	x	Product	This file provides the provision of external declaration and services for APIs.
CDD_Ipmmu_Irq.h	-	-	x	x	Product	This file provides the provision of external declaration and services for ISR.
CDD_Ipmmu_Reg.h	-	-	x	x	Product	This file provides the provision of IPMMU register structure and register definitions.
CDD_Ipmmu_Ram.h	-	-	x	x	Product	C headers file for CDD_Ipmmu_Ram.c. It includes the definitions that are required for RAM data.
CDD_Ipmmu_Types.h	-	-	x	x	Product	This file provides the definition of CDDIPMMU data types
CDD_Ipmmu_PBTypes.h	-	-	x	x	Product	This file provides the definition for Post-Build configuration types and setting.
rel\modules\cddipmmu\sample_application\V4H\19_11\include	-	-	-	-	-	-
CDD_Ipmmu_Cfg.h	-	-	x	-	Product	This file contains various driver Pre-compile time parameters. It also contains the macro for the total number of channels configured acknowledgement timeout and protection area.
CDD_Ipmmu_Cbk.h	-	-	x	-	Product	This file contains call-back function definition.
CDD_Ipmmu_Hardware.h	-	-	x	-	Product	This file contains all hardware information of IPMMU
rel\modules\cddipmmu\sample_application\V4M\19_11\include	-	-	-	-	-	-
CDD_Ipmmu_Cfg.h	-	-	-	x	Product	This file contains various driver Pre-compile time parameters. It also contains the macro for the total number of channels configured acknowledgement timeout and protection area.
CDD_Ipmmu_Cbk.h	-	-	-	x	Product	This file contains call-back function definition.
CDD_Ipmmu_Hardware.h	-	-	-	x	Product	This file contains all hardware information of IPMMU

x: applicable

-: not applicable

Table 3-134 IPMMU Source Files

Location:	Supported Devices				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
rel\modules\cddipmmu\src\	-	-	-	-	-	-
CDD_Ipmmu.c	-	-	x	x	Product	This file contains the function that Provision of initialization and control functionality.
CDD_Ipmmu_Ctr.c	-	-	x	x	Product	This file contains the global variable used by the CDD IPMMU.
CDD_Ipmmu_Irq.c	-	-	x	x	Product	This file contains the interrupt services for CDD IPMMU.
CDD_Ipmmu_Ram.c	-	-	x	x	Product	This file contains the global variables declaration.
CDD_Ipmmu_Version.c	-	-	x	x	Product	This file contains the function to get version information of the CDD IPMMU.
rel\modules\cddipmmu\sample_application\V4H\19_11\src	-	-	-	-	-	-
CDD_Ipmmu_PBcfg.c	-	-	x	-	Product	This file contains the post-build configuration data.
rel\modules\cddipmmu\sample_application\V4M\19_11\src	-	-	-	-	-	-
CDD_Ipmmu_PBcfg.c	-	-	-	x	Product	This file contains the post-build configuration data.

x: applicable
-: not applicable

The below table shows the list of Parameter Definition Files for CDDIPMMU module.

Table 3-135 IPMMU Parameter Definition Files

Location: rel\modules\cddipmmu\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	V4H	R1911_CDD_IPMMU_V4H.arxml	V4H
19_11	V4M	R1911_CDD_IPMMU_V4M.arxml	V4M

Note Product Name**:** Product Names supported by "Files".

3.4.4.4 Configuration Parameter Dependency

The below table shows the list of configuration parameter dependency for IPMMU Driver component.

Table 3-136 IPMMU Driver Component Configuration Parameter Dependency

Parameter	Module	Path
CDDIPMMU_E_WRITE_VERIFY_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDIPMMU_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.4.4.5 Source Code Dependency

The followings are the dependency files commonly used by the IPMMU Driver module:

- CddIpmmu_MemMap.h,
- Os.h,
- Det.h,
- Dem.h,
- Std_Types.h,
- Rte_CDD_Ipmmu.h,
- SchM_CddIpmmu.h,
- Compiler.h,
- Rte_CddIpmmu_Type.h

3.4.4.6 Stubs

Refer to "*3.2.9 Stubs File"* for the common stubs used for IPMMU Driver component.

3.4.4.7 Addition Error Handling

Refer to " Development and Production Errors" section at IPMMU chapter in Driver Component Embedded User's Manual.

3.4.4.8 Restrictions

Refer to " Preconditions" section and "Additional Error Handling and Restrictions" section at IPMMU chapter in Driver Component Embedded User's Manual.

3.4.4.9 Sample Application

3.4.4.9.1 Sample Application Structure

For more information refer to "*3.7 Sample Application*".

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddIpmmu_GetVersionInfo is invoked to get the version of the CDDIPMMU Driver module with a variable of Std_VersionInfoType. After the call of this API the past parameter will get updated with the CDDIPMMU Driver version details.

- The API CddIpmmu_Init is invoked with valid configuration address. This API performs the initialization of the IPMMU Complex Device Driver Component. This API initializes all the elements (Global Variables) of Global structure.
- Invoke CddIpmmu_MmuSetMemAttr to sets the memory attribute for each MMU.
- Invoke CddIpmmu_MmuSetMode to set the base address of translation table and attribute.
- Invoke CddIpmmu_ PmbSet to sets the translation for PMB's address.
- Invoke CddIpmmu_ MicroTlbSet to sets the address translation for target micro-TLB.

3.4.4.9.2**Recommended Environment**

- **V4H Environment**

Table 3-137 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

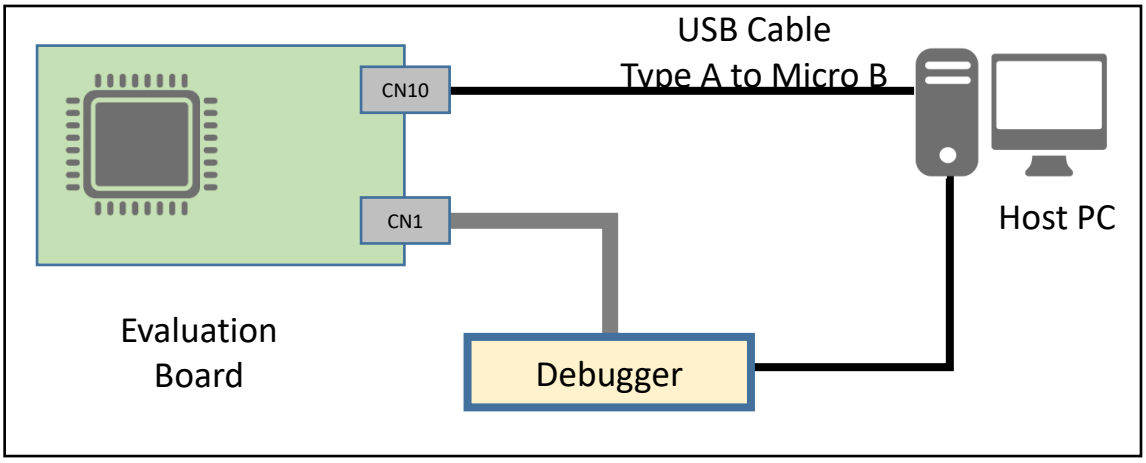

Figure 3-52 V4H White Hawk board connection of IPMMU Sample Application

- **V4M Environment**

Table 3-138 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

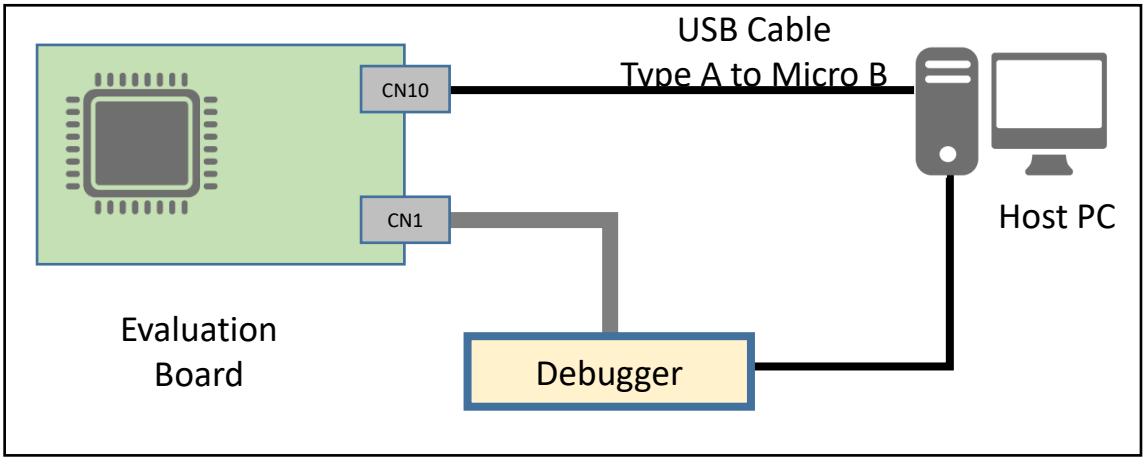

Figure 3-53 V4M Gray Hawk board connection of IPMMU Sample Application

3.4.4.9.3**Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.4.4.9.4**How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  For V4H: external\rel\V4H\common_family\make\arm

  For V4M: external\rel\V4M\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:

  For V4H: SampleApp.bat Cddipmmu R19-11 V4H No

  For V4M: SampleApp.bat Cddipmmu R19-11 V4M No

3.4.4.9.5**How to run sample application**

1/ Plug in serial port to Host PC, open the Terminal Software (Teraterm)

2/ Run the program and check sample application in print logs of the Teraterm:

“Starting IPMMU CDD Sample application”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.4.4.10 ROM/RAM Usage

See Appendix CDDIPMMU section for information on measuring RAM/ROM consumption.

3.4.4.11 Stack Depth

See Appendix CDDIPMMU section for information on measuring stack depth.

3.4.4.12 Throughput Details

See Appendix CDDIPMMU section for information on measuring execution time, and functional testing.

3.4.5 THS Driver Component

3.4.5.1 Module Overview

The THS Driver is part of the Complex Device Driver layer (CDD), which performs the hardware access and offers hardware-independent API to the upper layer to support communication between domains.

The THS Driver component shall provide the following main features:

- Initialization
- De-Initialization
- Enable/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

3.4.5.2 Module Dependency

The dependency of THS Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Default Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
Production errors will be reported to the Diagnostic Event Manager (DEM).

RTE
The Run Time Environment (RTE) module will be called to perform the communication with connected AUTOSAR Software Component.

OS
The OS provides a timer counter for THS module to detect timeout. If OS is not available, then it shall be stubbed.

3.4.5.3 Folder Structure

Table 3-139 and **Table 3-140** show the list of Source Code Files and Header Files for THS module.

Table 3-139 THS Header Files

Location: rel\modules\cddths\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
CDD_Ths.h	-	-	x	x	Product	This file provides externs declaration, type definition and config pointer for global as well as initialization of THS CDD.
CDD_Ths_PBTypes.h	-	-	x	x	Product	This file provides the definition for Post-Build configuration types and setting.
CDD_Ths_Ram.h	-	-	x	x	Product	C headers file for CDD_Ths_Ram.c. It includes the definitions that are required for RAM data.
CDD_Ths_Types.h	-	-	x	x	Product	This file provides the definition of THS CDD data types
CDD_Ths_Version.h	-	-	x	x	Product	C headers file for CDD_Ths_Version.c. It includes the definitions for get version info function.
include\THS	-	-	-	-	-	-
CDD_Ths_THS_LLDriver.h	-	-	x	x	Product	C headers file for CDD_Ths_THS_LLDriver.c. It includes the definitions that are required for CDD_Ths_THS_LLDriver.c

x: applicable

-: not applicable

Table 3-140 THS Source Files

Location: rel\modules\cddths\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
CDD_Ths.c	-	-	x	x	Product	This file contains API function implementations of THS Driver
CDD_Ths_Ram.c	-	-	x	x	Product	This file contains the global RAM variable of THS Driver
CDD_Ths_Version.c	-	-	x	x	Product	This file provides the version information of THS Driver
src\THS	-	-	-	-	-	-
CDD_Ths_THS_LLDriver.c	-	-	x	x	Product	This file contains the hardware access functionality

x: applicable
-: not applicable

Table 3-141 shows the list of Parameter Definition Files for CDDTHS module.

Table 3-141 THS Parameter Definition Files

Location: rel\modules\cddths\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	V4H	R1911_CDD_THS_V4H.arxml	V4H
19_11	V4M	R1911_CDD_THS_V4M.arxml	V4M

Note: Product Name**:** Product Names supported by "Files".

3.4.5.4 Configuration Parameter Dependency

Table 3-142 shows the list of configuration parameter dependency for THS Driver component.

Table 3-142 THS Driver Component Configuration Parameter Dependency

Parameter	Module	Path
CDDTHS_E_WRITEVERIFY_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CddThsOsCounterRef	OS	/AUTOSAR/EcucDefs/Os/OsCounter

3.4.5.5 Source Code Dependency

The followings are the dependency files commonly used by the THS Driver module:

- Det.h
- CddThs_MemMap.h
- Platform_Types.h
- Std_Types.h
- SchM_CddThs.h
- Os.h

3.4.5.6 Stubs

Refer to 3.2.9 for the common stubs used for THS Driver component.

3.4.5.7 Addition Error Handling

Refer to " Development and Production Errors" section at THS chapter in Driver Component Embedded User's Manual.

3.4.5.8 Restrictions

None

3.4.5.9 Sample Application

❖ **V4H:**

3.4.5.9.1 Sample Application Structure

Refer to 3.7.1 Sample Application Structure

3.4.5.9.2 Recommended Environment

- **V4H Environment**

Table 3-143 V4H Environment

Name	Explanation

Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

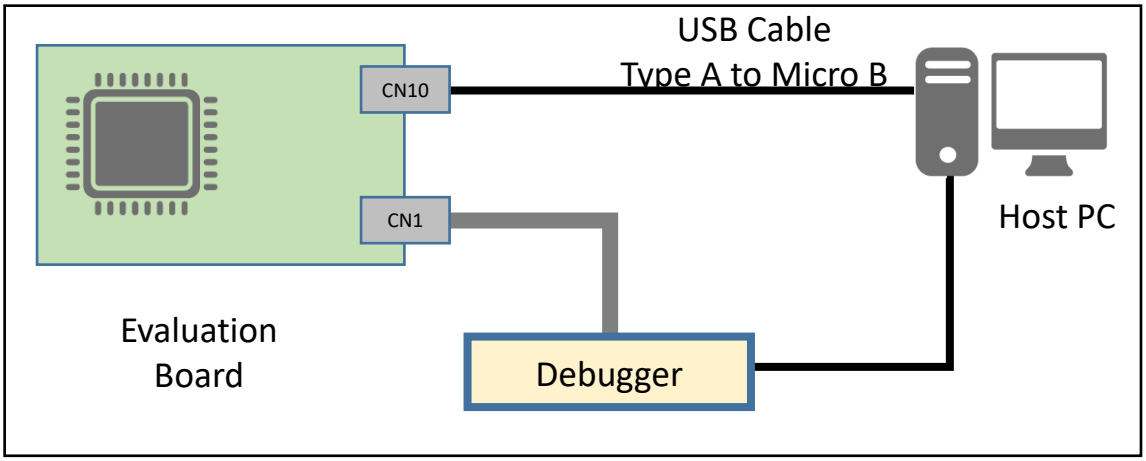

Figure 3-54 V4H White Hawk board connection of THS Sample Application

- **V4M Environment**

Table 3-144 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

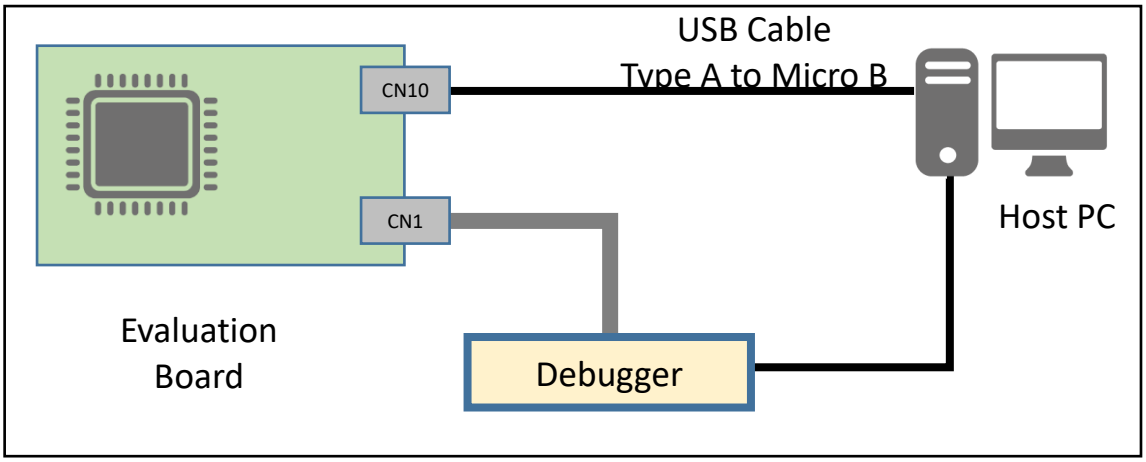

Figure 3-55 V4M Gray Hawk board connection of THS Sample Application

3.4.5.9.3 **Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.4.5.9.4 **How to build sample application**

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  external\rel\V4H\common_family\make\arm (For V4H)

  external\rel\V4M\common_family\make\arm (For V4M)
- Now execute the batch file Sample App.bat with following parameters:

  SampleApp.bat CddThs R19-11 V4H No (For V4H)

  SampleApp.bat CddThs R19-11 V4M No (For V4M)

3.4.5.9.5 **How to run sample application**

1/ Connect the electric wire between Debugger and Host PC, wire between Host PC to Evaluation Board.

2/ Plug in serial port to Host PC, open the Terminal Software (Teraterm).

3/ Run the program and check sample application in print logs of the Teraterm:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.4.5.10 **ROM/RAM Usage**

See Appendix CDDTHS section for information on measuring RAM/ROM consumption.

3.4.5.11 **Stack Depth**

See Appendix CDDTHS section for information on measuring stack depth.

3.4.5.12 **Throughput Details**

See Appendix CDDTHS section for information on measuring execution time, and functional testing.

3.4.6 EMM Driver Component

3.4.6.1 Module 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.
- Get version information.
- Support to control external error request.

3.4.6.2 Module Dependency

The dependency of EMM Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Development Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
In production mode, the Diagnostic Event Manager (DEM) will be called whenever this module encounters a development error.

RTE
The Run Time Environment (RTE) module will be called to perform the communication with connected AUTOSAR Software Component.

OS
For interrupt category 2, use ISR() macro from Os.h.

EcuM
The EcuM will initialize the CddEmm driver.

SchM
The Basic Software Scheduler module will be called whenever a critical section protection function is called.

PORT
The components which CddEmm dirver module requires its preceding initialization.

MCU
The components which CddEmm dirver module requires its preceding initialization.

3.4.6.3 Folder Structure

Table 3-146 and **Table 3-145** show the list of Source Code Files and Header Files for Emm module.

Table 3-145 EMM Header Files

Location: rel\modules\cddemm\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
include	-	-	-	-	-	-
CDD_Emm.h	-	-	x	x	Product	This file provides externs declaration, type definition and global macros.
CDD_Emm_Internal.h	-	-	x	x	Product	C headers file for CDD_Emm_Internal.c. It includes the header files that are required for CDD_Emm_Internal.c.
CDD_Emm_Irq.h	-	-	x	x	Product	C headers file for CDD_Emm_Irq.c. It includes the header files that are required for CDD_Emm_Irq.c.
CDD_Emm_Types.h	-	-	x	x	Product	This file provides the definition of CDD EMM data types
CDD_Emm_Ram.h	-	-	x	x	Product	C headers file for CDD_Emm_Ram.c. It includes the definitions that are required for RAM data.
CDD_Emm_RegReadWrite.h	-	-	x	x	Product	This file provides macros to access hardware registers.
CDD_Emm_Types.h	-	-	x	x	Product	This file provides the definition of the EMM Complex Driver Component data types.
CDD_Emm_Version.h	-	-	x	x	Product	C headers file for CDD_Emm_Version.c. It includes the definitions that are required for CDD_Emm_Version.c

x: applicable
-: not applicable

Table 3-146 EMM source file

Location: rel\modules\cddemm\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
CDD_Emm.c	-	-	x	x	Product	This file contains the core functions of the EMM Complex Driver Component.
CDD_Emm_Internal.c	-	-	x	x	Product	his file contains the internal functions used in EMM Complex Driver Component.
CDD_Emm_Irq.c	-	-	x	x	Product	This file contains the interrupt service routines functionality.
CDD_Emm_Ram.c	-	-	x	x	Product	This file contains the global variables used by the EMM Complex Driver Component.
CDD_Emm_Version.c	-	-	x	x	Product	This file contains the function to get version information of the EMM Complex Driver Component.

x: applicable
-: not applicable

Table 3-147 shows the list of Parameter Definition Files for CDDEMM module.

Table 3-147 EMM **Parameter Definition Files**

Location: rel\modules\cddemm\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	V4H	R1911_CDD_EMM_V4H.arxml	V4H
19_11	V4M	R1911_CDD_EMM_V4M.arxml	V4M

Note Product Name**:** Product Names supported by "Files".

3.4.6.4 Configuration Parameter Dependency

Table 3-148 shows the list of configuration parameter dependency for EMM Driver component.

Table 3-148 EMM Driver Component Configuration Parameter Dependency

Parameter	**Module**	**Path**
CDDEMM_E_WRITE_VERIFY_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDEMM_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.4.6.5 Source Code Dependency

The followings are the dependency files commonly used by the EMM Driver module:

- CddEmm_MemMap.h
- Det.h
- Dem.h
- SchM_CddEmm.h

3.4.6.6 Stubs

Refer to *3.2.9 "Stubs File"* for the common stubs used for EMM Driver component.

3.4.6.7 Addition Error Handling

Refer to " Development and Production Errors" section at EMM chapter in Driver Component Embedded User's Manual.

3.4.6.8 Restrictions

Refer to "Preconditions" section at EMM chapter in Driver Component Embedded User's Manual.

3.4.6.9 Sample Application

❖ **V4H:**

3.4.6.9.1 Sample Application Structure

For more information, refer *to 3.7.1 "Sample Application Structure"*.

- Initialization SCIF driver, MCU diver, PORT driver, WDG driver by stub function for user refer to develop their application.
- The API CddEmm_Init is invoked with valid configuration address. This API performs the initialization of the EMM Complex Device Driver Component. This API initializes all the elements (Global Variables) of Global structure.

- The API CddEmm_GetVersionInfo is invoked to get the version of the EMM Driver module with a variable of Std_VersionInfoType. After the call of this API the past parameter will get updated with the EMM Driver version details.
- The API CddEmm_SupportPseudoError is invoked to enable/disable the Pseudo error functionality.
- The API CddEmm_SetTarget is invoked to selects target of error to be notified when error occurred.
- The API CddEmm_SetPseudoErrorSignal is invoked to set a specified pseudo error signal.
- The API CddEmm_ReadErrorStatus is invoked to read status of safety-related errors.
- The API CddEmm_ClearPseudoErrorSignal is invoked to clear a specified pseudo error signal.
- The API CddEmm_ClearErrorStatus is invoked to clear status of all safety-related.
- The API CddEmm_GetCurrentErrorCountUpValue is invoked to get reads 1-bit error count value and multi-bit error count value of all error count registers.
- The API CddEmm_SupportControlExternalErrorRequest is invoked to enables/disables counter of the external error request control.
- The API CddEmm_SetHoldMaskCounter is invoked to set hold/mask counter of the external error request control.

3.4.6.9.2 **Recommended Environment**

- **V4H Environment**

Table 3-149 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

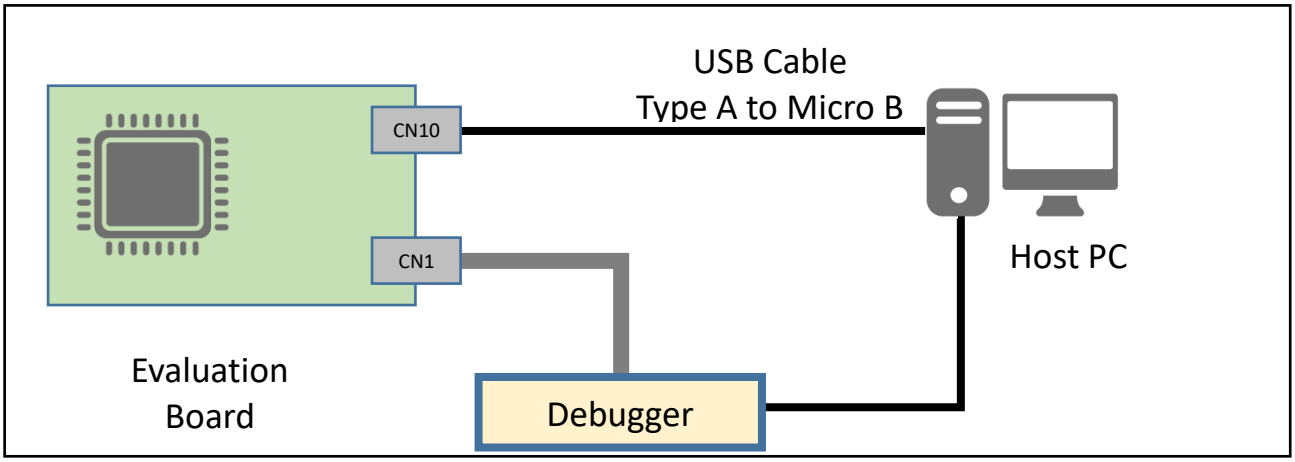

Figure 3-56 V4H White Hawk board connection of EMM Sample Application

- **V4M Environment**

Table 3-150 V4M Environment

Name	Explanation
Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02

Terminal Software	Teraterm

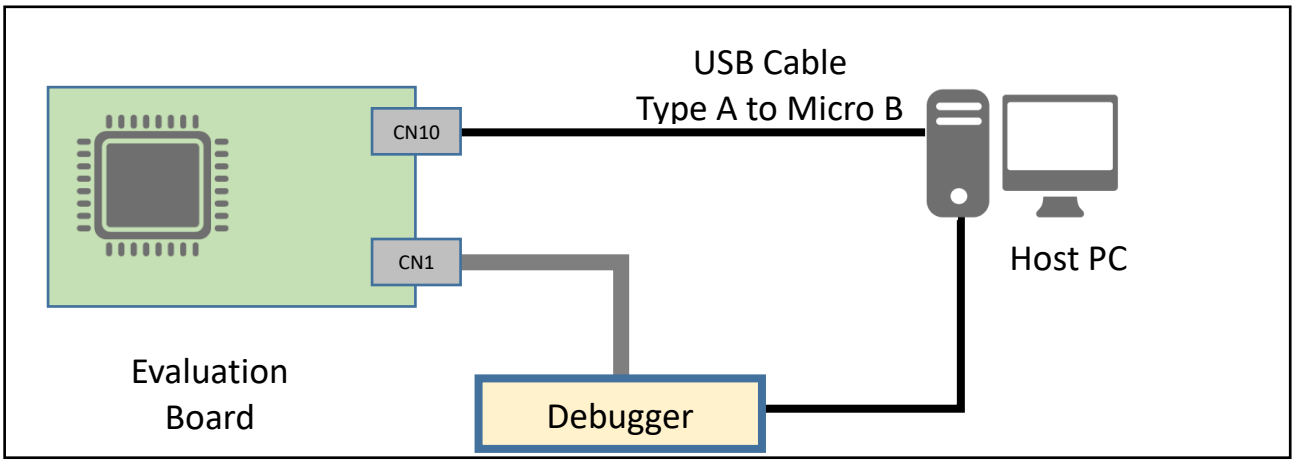

Figure 3-57 V4M Gray Hawk board connection of EMM Sample Application

3.4.6.9.3 **Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.4.6.9.4 **How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

  For V4H: external\rel\V4H\common_family\make\arm

  For V4M: external\rel\V4M\common_family\make\arm
- Now execute the batch file Sample App.bat with following parameters:

  For V4H: SampleApp.bat CddEmm R19-11 V4H No

  For V4M: SampleApp.bat CddEmm R19-11 V4M No

3.4.6.9.5 **How to run sample application**

1/ Connect the electric wire between Debugger and Host PC, wire between Host PC to Evaluation Board.

2/ Continue run the program via Debugger sofware until the to Terminal Software (Teraterm) print the log:

- "PROGRAM START"
- "[Cortex] EMM CDD Sample Application Start."
- "[Cortex] EMM CDD version information is completed."
- "CddEmm_Init COMPLETED"
- "[Cortex] Test on CddEmmDomain0 is completed."
- "[Cortex] Test on CddEmmDomain1 is completed."
- "[Cortex] Test on CddEmmDomain2 is completed."
- "[Cortex] Test on CddEmmDomain3 is completed."
- "[Cortex] Test on CddEmmDomain4 is completed."
- "[Cortex] Test on CddEmmDomain5 is completed."
- "[Cortex] Test on CddEmmDomain6 is completed."
- "[Cortex] Test on CddEmmDomain7 is completed."
- "[Cortex] Test on CddEmmDomain8 is completed."

- "[Cortex] Test on CddEmmDomain9 is completed."
- "[Cortex] Test on CddEmmDomain10 is completed."
- "[Cortex] Test on CddEmmDomain11 is completed."
- "[Cortex] Test on CddEmmDomain12 is completed."
- "[Cortex] Test on CddEmmDomain13 is completed."
- "[Cortex] Test on CddEmmDomain16 is completed."
- "[Cortex] Test on CddEmmDomain17 is completed."
- "[Cortex] Test on CddEmmDomain18 is completed."
- "[Cortex] Test on CddEmmDomain19 is completed."
- "[Cortex] Test on CddEmmDomain20 is completed."
- "[Cortex] Test on CddEmmDomain21 is completed."
- "[Cortex] Test on CddEmmDomain22 is completed."
- "[Cortex] Test on CddEmmDomain23 is completed."
- "[Cortex] Test on CddEmmDomain24 is completed."
- "[Cortex] Test on CddEmmDomain25 is completed."
- "[Cortex] Test on CddEmmDomain26 is completed."
- "[Cortex] Test on CddEmmDomain27 is completed."
- "[Cortex] Test on CddEmmDomain28 is completed."
- "[Cortex] Test on CddEmmDomain29 is completed."
- "[Cortex] Test on CddEmmDomain30 is completed."
- "[Cortex] Test on CddEmmDomain31 is completed."
- "[Cortex] Test on CddEmmDomain32 is completed."
- "[Cortex] Test on CddEmmDomain33 is completed."
- "[Cortex] Test on CddEmmDomain34 is completed."
- "[Cortex] Test on CddEmmDomain35 is completed."
- "[Cortex] Test on CddEmmDomain36 is completed."
- "[Cortex] Test on CddEmmDomain38 is completed."
- "[Cortex] Test on CddEmmDomain39 is completed."
- "[Cortex] Test on CddEmmDomain40 is completed."
- "[Cortex] Test on CddEmmDomain41 is completed."
- "[Cortex] Test on CddEmmDomain42 is completed."
- "[Cortex] Test on Get Current Error Count is completed."
- "[Cortex] Test on External Error Control Request is completed. "
- "EXECUTED OK"
- "PROGRAM STOP"

If receive the message "EXECUTED NOT OK" or not receive any log print relate to CDDEMM sample app which mean the sample application was failed.

3.4.6.10 **ROM/RAM Usage**

See Appendix CDDEMM section for information on measuring RAM/ROM consumption.

3.4.6.11 **Stack Depth**

See Appendix CDDEMM section for information on measuring stack depth.

3.4.6.12 **Throughput Details**

See Appendix CDDEMM section for information on measuring execution time, and functional testing.

3.4.7 CRC Driver Component

3.4.7.1 Module Overview

The purpose of this document is to describe the information related to CRC 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 CRC Complex Driver. The CRC Complex Driver Component provides the following services:

- CRC result from CDD CRC is created based on R-Car Series. This CRC result is XORed from AUTOSAR CRC with XOR value.
- 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

3.4.7.2 Module Dependency

The dependency of CRC Driver on other modules and the required implementation is briefed as follows:

DET
In development mode, the Development Error Tracer (DET) will be called whenever this module encounters a development error.

DEM
In production mode, the Diagnostic Event Manager (DEM) will be called whenever this module encounters a development error.

RTE
The Run Time Environment (RTE) module will be called to perform the communication with connected AUTOSAR Software Component.

OS
For interrupt category 2, use ISR () macro from Os.h.

SchM
The Basic Software Scheduler module will be called whenever a critical section protection function is called.

MCU
The components which CddCrc dirver module requires its preceding initialization.

3.4.7.3 **Folder Structure**

Table 3-152 and **Table 3-151** show the list of Source Code Files and Header Files for Crc module.

Table 3-151 CRC header file

Location: rel\modules\cddcrc	**Supported Device**				**Category**	**Description**
	S4 (Includes S4N) CR52	**S4 (Includes S4N) G4MH**	**V4H**	**V4M**		
include	-	-	-	-	-	-
CDD_Crc.h	-	-	x	x	Product	This file provides externs declaration, type definition and global macros.
CDD_Crc_Irq.h	-	-	x	x	Product	C headers file for CDD_Crc_Irq.c. It includes the header files that are required for CDD_Crc_Irq.c.
CDD_Crc_PBTypes.h	-	-	x	x	Product	This file provides the definition of CDD CRC post-build data types.
CDD_Crc_Ram.h	-	-	x	x	Product	C headers file for CDD_Crc_Ram.c. It includes the definitions that are required for RAM data.
CDD_Crc_RegisterAcess.h	-	-	x	x	Product	This file provides macros to access hardware registers.
CDD_Crc_Types.h	-	-	x	x	Product	This file provides the definition of the CRC Complex Driver Component data types.
CDD_Crc_Version.h	-	-	x	x	Product	C headers file for CDD_Crc_Version.c. It includes the definitions that are required for CDD_Crc_Version.c
CRC	-	-			-	-
CDD_Crc_LLDriver.h	-	-	x	x	Product	This file contains the external declaration for the internal functions (Low Level Driver) for CRC called by the CRC Driver APIs.
KCRC	-	-			-	-
CDD_Crc_KCRC_LLDriver.h	-	-	x	x	Product	This file contains the external declaration for the internal functions (Low Level Driver) for KCRC called by the KCRC Driver APIs.
WCRC	-	-			-	-
CDD_Crc_WCRC_LLDriver.h	-	-	x	x	Product	This file provides the declaration of low-level functions of WCRC, and RT-DMAC which is used in APIs.

x: applicable
-: not applicable

Table 3-152 CRC source file

Location: rel\modules\cddcrc\	Supported Device				Category	Description
	S4 (Includes S4N) CR52	S4 (Includes S4N) G4MH	V4H	V4M		
src	-	-	-	-	-	-
CDD_Crc.c	-	-	x	x	Product	This file contains the core functions of the CRC Complex Driver Component.
CDD_Crc_Irq.c	-	-	x	x	Product	This file contains the interrupt service routines functionality.
CDD_Crc_Ram.c	-	-	x	x	Product	This file contains the global variables used by the CRC Complex Driver Component.
CDD_Crc_Version.c	-	-	x	x	Product	This file contains the function to get version information of the CRC Complex Driver Component.
CDD_Crc_LLDriver.c	-	-	x	x	Product	This file contains implementation of all Low-Level Driver functions for CRC invoked by CRC Driver APIs.
CDD_Crc_KCRC_LLDriver.c	-	-	x	x	Product	This file contains implementation of all Low-Level Driver functions for KCRC invoked by CRC Driver APIs.
CDD_Crc_WCRC_LLDriver.c	-	-	x	x	Product	This file contains implementation of all Low-Level Driver functions for WCRC, and RT-DMAC

x: applicable
-: not applicable

Table 3-153 shows the list of Parameter Definition Files for CDDCRC module.

Table 3-153 CRC Parameter Definition Files

Location: rel\modules\cddcrc\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	V4H	R1911_CDD_CRC_V4H.arxml	V4H
19_11	V4M	R1911_CDD_CRC_V4M.arxml	V4M
Location: rel\modules\cddcrc\definition\<AR>\<Device_Name>			
<AR>	**<Device_Name>**	**Files**	**Product Name**
19_11	V4H	R1911_CDD_CRC_V4H.arxml	V4H
19_11	V4M	R1911_CDD_CRC_V4M.arxml	V4M

Note Product Name**:** Product Names supported by "Files".

3.4.7.4 Configuration Parameter Dependency

Table 3-154 shows the list of configuration parameter dependency for CRC Driver component.

Table 3-154 CRC Driver Component Configuration Parameter Dependency

Parameter	**Module**	**Path**
CDDCRC_E_WRITE_VERIFY	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDCRC_E_INTERRUPT_CONTROLLER_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDCRC_E_UNINTENDED_MODULE_STOP_FAILURE	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter
CDDCRC_E_HARDWARE_ERROR	DEM	/AUTOSAR/EcucDefs/Dem/DemConfigSet/DemEventParameter

3.4.7.5 Source Code Dependency

The followings are the dependency files commonly used by the CRC Driver module:

- CddCrc_MemMap.h
- Det.h
- Dem.h
- SchM_CddCrc.h

3.4.7.6 Stubs

Refer to 3.2.9 for the common stubs used for CRC Driver component.

3.4.7.7 Addition Error Handling

Refer to " Development and Production Errors" section at CRC chapter in Driver Component Embedded User's Manual.

3.4.7.8 Restrictions

Refer to "General" section in section at CRC chapter in Driver Component Embedded User's Manual.

3.4.7.9 Sample Application

3.4.7.9.1 Sample Application Structure

❖ **V4H:**

For more information, refer to 3.7.1 Sample Application Structure.

- The API CddCrc_GetVersionInfo is invoked to retrieve the version information of CRC Complex Driver.
- The API CddCrc_Init is invoked with valid configuration set to perform the initialization of the CRC Complex
- Driver Component. This API also initializes all the elements (Global Variables) of Global structure.
- The API CddCrc_Process is invoked to enable the process of creating/checking CRC from input data
- The API CddCrc_SetMode is invoked to switch mode from INDEPENDENT_MODE to

E2E_PLUS_DATA_THROUGH_MODE
- The API CddCrc_SetupEB is invoked twice to read data from both data port and result port.
- The API CddCrc_Write is invoked to write data to data port via DMA.
- The API CddCrc_ReadStatus is invoked until the channel status returns CDDCRC_CH_OK.
- The API CddCrc_Compare is invoked to compare an actual result with an expected one.
- The comparison result is returned by notification function.

❖

❖ **V4M**

For more information, refer to 3.7.1 Sample Application Structure.
- The API CddCrc_GetVersionInfo is invoked to retrieve the version information of CRC Complex Driver.
- The API CddCrc_Init is invoked with valid configuration set to perform the initialization of the CRC Complex
- Driver Component. This API also initializes all the elements (Global Variables) of Global structure.
- The API CddCrc_Process is invoked to enable the process of creating/checking CRC from input data
- The API CddCrc_SetMode is invoked to switch mode from INDEPENDENT_MODE to E2E_PLUS_DATA_THROUGH_MODE
- The API CddCrc_SetupEB is invoked twice to read data from both data port and result port.
- The API CddCrc_Write is invoked to write data to data port via DMA.
- The API CddCrc_ReadStatus is invoked until the channel status returns CDDCRC_CH_OK.
- The API CddCrc_Compare is invoked to compare an actual result with an expected one.
- The comparison result is returned by notification function.

3.4.7.9.2 **Recommended Environment**

- **V4H Environment**

Table 3-155 V4H Environment

Name	Explanation
Evaluation Board	R-Car V4H System Evaluation Board (White Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

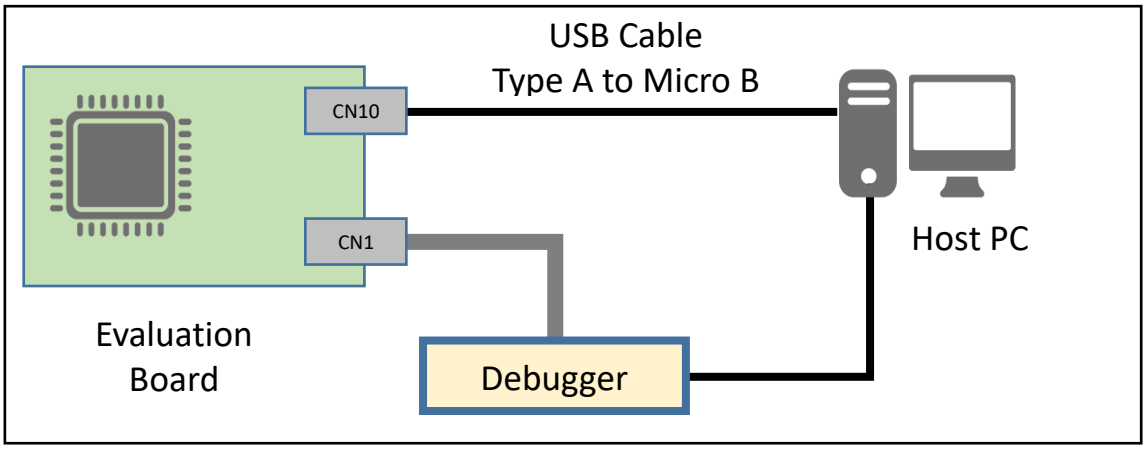

Figure 3-58 V4H White Hawk board connection of CRC Sample Application

- **V4M Environment**

Table 3-156 V4M Environment

Name	Explanation

Evaluation Board	R-Car V4M System Evaluation Board (Gray Hawk) Renesas Electronics
Switch setting on the Board	Factory settings
Host PC (windows)	Windows 10 is recommended for host PC.
Debugger Hardware	Debugger (Lauterbach ARMv8)
Debugger Software	Trace32 for R.2017.02
Terminal Software	Teraterm

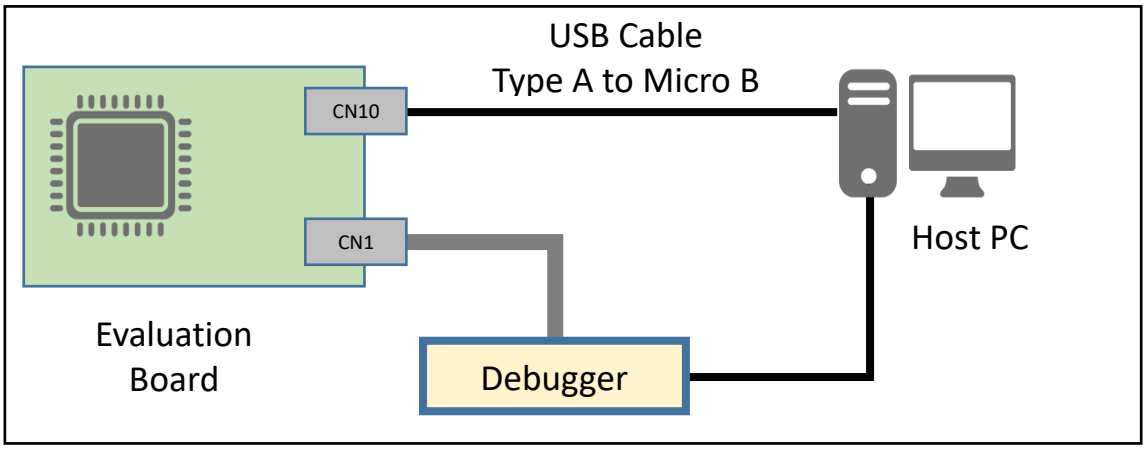

Figure 3-59 V4M Gray Hawk board connection of CRC Sample Application

3.4.7.9.3 **Preparation**

- Prepare the environment for the corresponding device at the above (2) Recommended Environtment.
- For V4H/V4M, read more environment setup and guideline at 3.7.3 How to run Sample Application.

3.4.7.9.4 **How to build sample application**

Refer to 3.7.2.2 How to build the Sample Application

- Open a Command window and change the current working directory to “make” directory present as mentioned in below path:
  For V4H: external\rel\V4H\common_family\make\arm
  For V4M: external\rel\V4M\common_family\make\arm

Now execute the batch file Sample App.bat with following parameters:

For V4H: SampleApp.bat Cddcrc R19-11 V4H No

For V4M: SampleApp.bat Cddcrc R19-11 V4M No

3.4.7.9.5 **How to run sample application**

1/ Plug in serial port to Host PC, open the Terminal Software (Teraterm).

2/ Run the program and check sample application in print logs of the Teraterm:

“PROGRAM START”: Sample application execution is started.

“EXECUTED OK”: Sample application execution is successful.

“EXECUTED NOT OK”: Sample application execution is failed.

“PROGRAM STOP”: Sample application execution is completed.

3.4.7.10 **ROM/RAM Usage**

See Appendix CDDCRC section for information on measuring RAM/ROM consumption.

3.4.7.11 Stack Depth

See Appendix CDDCRC section for information on measuring stack depth.

3.4.7.12 Throughput Details

See Appendix CDDCRC section for information on measuring execution time, and functional testing.

3.5 Preconditions

- In R-Car Gen4, it is necessary to call the API with the authority to access the register described in the Component User's Manual of MCAL each module.
- The configuration set used as argument in initialization function will be generated as “Msn_Config” as per AUTOSAR_TPS_ECUConfiguration.

3.5.1 EIC Registers

- The MCAL driver does not set the EITBn-bit of EIC Register. Therefore, it is necessary to set the EITBn-bit according to the interrupt method used by the user before initializing the MCAL driver.
- The MCAL driver does not set the EIPn[3:0]-bits of EIC Register. EIPn[3:0]-bits is specified the interrupt priority. To prioritize the interrupts, set EIPn[3:0]-bits.
- The MCAL driver for S4 G4MH device does not consider the Interrupt Overflow EIOVn bit in EICn registers. Missing interrupt request can be detected by "Periodic check of not executed interrupt requests" in Safety Application Note.

3.5.2 Exclusive Area

MCAL modules have the exclusive area to protect register read, modify, write, global variables write access and any other module-specific reason. Refer to Component User’s Manual of MCAL for each module.

3.5.2.1 Type of Critical Section

As the principle, it uses the following 2 critical sections.

❖ **<MSN>_RAM_DATA_PROTECTION**

It is used for the exclusion access of the global variable and the module-specific register.

It does not prohibit interrupt, but it prevents more than one context from being stored in the same critical section at the same time.

When OS exists, GetResource() can be used.

When OS does not exist and CAT1 ISR is used, it is necessary to disable the interrupt.

❖ **<MSN>_INTERRUPT_CONTROL_PROTECTION**

It is used to disable interrupt and task dispatch:

When reading, modifying, and writing for shared resource register.

When disabling interruption, such as changing the interrupt mask during peripheral operation.

The Timer HW resource is shared with GPT, ICU, and PWM modules. These exclusive areas relate to the timer register.

- GPT: GPT_INTERRUPT_CONTROL_PROTECTION
- ICU: ICU_INTERRUPT_CONTROL_PROTECTION
- PWM: PWM_INTERRUPT_CONTROL_PROTECTION

DIO/PORT modules use PORT register. These exclusive areas relate to the PORT setting.

- DIO: DIO_INTERRUPT_CONTROL_PROTECTION
- PORT: PORT_INTERRUPT_CONTROL_PROTECTION

The MCAL module has critical section protection parameter.

e.g. <Msn>CriticalSectionProtection

<Msn>CriticalSectionProtection parameter is configured as “true”, and critical section enters/exits via SchM_Enter/SchM_Exit APIs in the RTE module.

e.g. SchM_Enter_<Msn>_<MSN>_INTERRUPT_CONTROL_PROTECTION() and
SchM_Exit_<Msn>_<MSN>_INTERRUPT_CONTROL_PROTECTION().

Note <Msn> and <MSN>: Refer to Table 3-2.

The Component User's Manual for each module provides the detailed register information for each API.

3.5.3 User Mode and Supervisor Mode

Note: Not supported by R-Car

3.5.4 Hypervisor Mode

RCar G4MH MCAL doesn't support Hypervisor mode, it should be disabled by SW-OPBT (Software Option Byte) setting. For more information, refer to *RCar Hardware User manual*.

3.5.5 Region ID Access Protection

A Region ID (RGID) protection is a function that blocks illegal access from un-expected master to slave module. In Region ID, there are "Master Setting Register" (RGID_ <Master>), "Read/Write Enable Register" (RGIDR_<Slave>, RGIDW_<Slave>). Depending on the settings of these register values, the protection function will be supported or not. About the Region ID Register values, users can customize them to meet their environmental needs and expectations.
Recommendation: Before accessing registers in PFC, CPG, and RESET, set the Region ID Protection as follows:
1. Set Region ID Protection for each region by RGIDR_MODID, RGIDW_MODID register
2. Set Region ID for each master by RGIDM_MODID register.

3.6 Multi-Core / Multi-Instantiation

R-Car Gen4 does not support multi-core and multi-instantiation

3.7 Sample Application

3.7.1 Sample Application Structure

The Sample Application of each module is provided as reference to the user to understand the method in which the APIs can be invoked from the application.

The Sample Application for each module is available in the following path.
rel\modules\<msn>\sample_application

```
Install_destination
  └── rel
      └──── modules
      └──── <msn>
          │
          ├──── Sample Application
          │     ├──── <Device>
          │     │     ├──── <AR>
          │     │     │     ├──── config
          │     │     │     ├──── include
          │     │     │     └──── src
          │     │     ├──── include
          │     │     ├──── make
          │     │     │     └──── <compiler>
          │     │     ├──── obj
          │     │     │     └──── <compiler>
          │     │     └──── src
          │     ├──── include
          │     ├──── make
          │     │     └──── <compiler>
          │     └──── src
```

Note <msn>, <AR>, <Device>: Refer to Table 3-2.
<compiler>: arm (for CR52 core) or ghs (for G4MH core)

Figure 3-60 Sample Application Structure

For details of the folder structure under the **sample_application\,** refer to 3.2.8 Sample Application File.

3.7.2 Building Sample Application

3.7.2.1 Configuration Example

The configuration file for the sample application of each module is provided in the following path.

rel\modules\<msn>\sample_application\<Device_Name>\<AR>\config

File name: App_<MSN>_<Device_Name>_<Device_ID>_Sample.arxml
: App_<CDD>_<MSN>_<Device_Name>_<Device_ID>_Sample.arxml *1
: App_<MSN>_<Device_Name>_<Device_ID>_Multi_Sample.arxml *2

*1: CDD modules only.
*2: Multi-instantiation-supported modules only.
For the Multi-instantiation-supported modules, refer to 3.6 Multi-instantiation.

3.7.2.2 How to build the Sample Application

Note GNU Make utility version 3.81 or above must be installed and available in the path as defined by the environment user variable “GNUMAKE” to complete the build process using the delivered sample files.

1. Open a Command window and change the current working directory to the “make” directory mentioned in the following path:
   rel\<Device_Family>\common_family\make\<Compiler>
2. Execute batch file SampleApp.bat with the following parameters:
   SampleApp.bat <msn> <AR> <Device_ID> <Multi-instance> <MCAL Log Print> <Build Option>

Note:
<msn>: Module Short Name to be generated.
<AR>: AUTOSAR version to be compiled which is available.
<Device_ID>: Device Name to be compiled which is available.
<Multi-instance>: Select multi-instance sample application or single instance sample application (Optional argument).
Yes : Multi-instance
No : Single Instance
<MCAL Log Print>: Select enable or disable output MCAL driver’s log to console (Optional argument).
Yes : Output MCAL driver's log to console when using Console_Print() function in Sample Application
No : Do not output MCAL driver's log to console when using Console_Print() function in Sample Application
<Build Option>: Build option (Optional argument).
- unset(null): Build with all processes.
- clean: Only delete object files.
- generate: Only generation tool done.
- make: Only compile and link (use already generated files)

3. Then, the tool output files will be generated with the configuration available in the following path:
   rel\modules\<msn>\sample_application\<Device_Name>\<AR>\config

4. Then, all the object files, map file and the executable file App_<MSN>_<Device_ID>_Sample.out (GHS compiler) or App_<MSN>_<Device_ID>_Sample.elf (ARM compiler) will be available in the output folder
   rel\modules\<msn>\sample_application\<Device_Name>\obj\<compiler>

   The executable can be loaded into the debugger and the sample application can be executed.

Note: Executable files with ‘*.out’ (GHS) or ‘*.elf’ (ARM) extension can be downloaded to the target hardware with the help of Green Hills debugger (GHS) or Lauterbach Trace32 debugger (ARM). And the flashable Motorola S-Record file App_<MSN>_<Device_ID>_Sample.s37 (GHS) or App_<MSN>_<Device_ID>_Sample.srec (ARM) and map file App_<MSN>_<Device_ID>_Sample.map will be generated in the output folder. Those files will help debugging.

3.7.3 How to run Sample Application

3.7.3.1.1 Precondition

The switch setting is a prerequisite for the factory setting. The following chapters explain the differences from the factory settings.

3.7.3.1.2 Precaution

The sample application of MCAL driver uses *Console_Print()* function to output operation log to terminal console (e.g Tera Term software console). When using in combination with Software other than MCAL driver, it is recommended not to use Console_Print() function to output log so that serial log outputs are not mixed. It can be disabled by set <MCAL Log Print> option in Section *3.7.2.2 How to build the Sample Application* of this document to **No**.

RCar S4:

- For module runs on Cortex-R52 domain (Application domain), if it needs to access to RH850 G4MH domain (Control domain), the enable access permission need to be performed. In this case, refer to *Precaution* section of each module section (e.g. PORT, DIO, CAN, ICCOM) for more information.

RCar V4H:

- When using the V4H White Hawk board, make sure that the power switch (SW51) is turned OFF (knock down to pin3) before applying the 12V power supply to the CN20 or CN22 connector.

RCar V4M:

- When using the V4M Gray Hawk board, make sure that the power switch (SW51) is turned OFF (knock down to pin3) before applying the 12V power supply to the CN20 or CN22 connector.

3.7.3.1.3 Flash bootloaders

The followings describe the steps to flash ICUMX Loader, CX 2nd IPL and others dummy program to RCar Evaluation board.

Step 1 Connect USB cable and Power adapter

RCar S4:

- Connect USB Host connector of Windows PC that is virtual COM port to CN21 (CN20 for B0 2nd Spider board) of Spider CPU board with USB cable for displaying console.
- Connect power jack to CN45 or CN46 connector (RCar S4 Spider breakout sub-board) to supply power for Evaluation board.

RCar V4H:

- Connect USB Host connector of Windows PC that is virtual COM port to CN10 of White Hawk CPU board with USB cable for displaying console.
- Connect power jack to CN20 or CN21 connector (RCar V4H White Hawk breakout sub-board) to supply power for Evaluation board.

RCar V4M:

- Connect USB Host connector of Windows PC that is virtual COM port to CN10 of White Hawk CPU board with USB cable for displaying console.
- Connect power jack to CN20 or CN21 connector (RCar V4M Gray Hawk breakout sub-board) to supply power for Evaluation board.

Step 2 Setup the Terminal Software

(2-1) Activate the Terminal Software (e.g., Tera Term) on Windows PC, select COM Port

Note: COM6 is an example, it is defference based on each Host PC.

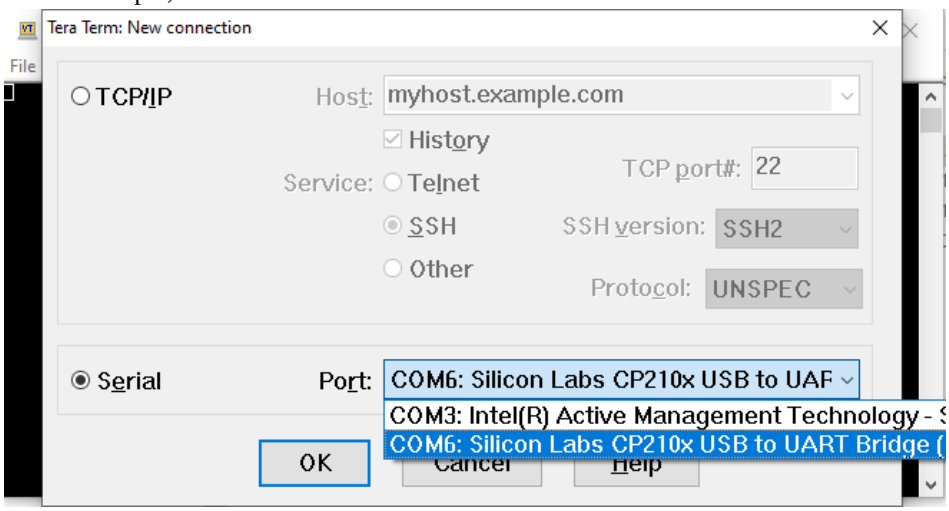

Figure 3-61 Setup Terminal Software 1

(2-2) Click to [Setup] -> [Serial port…] as the following figures

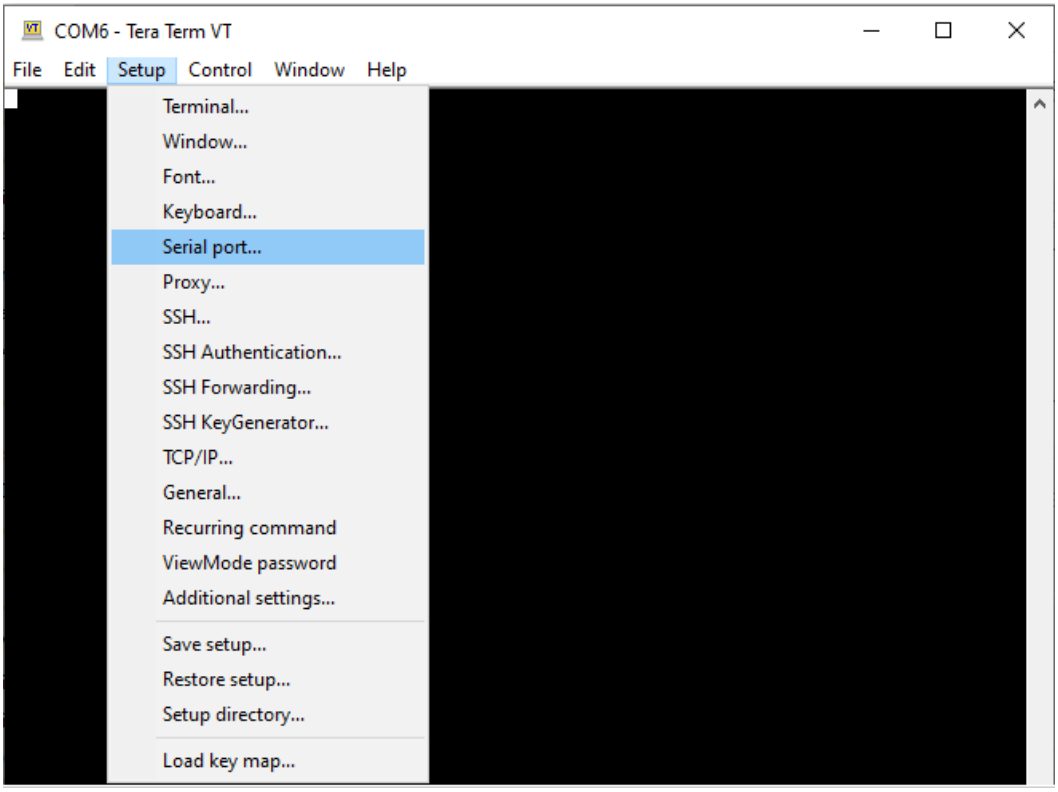

Figure 3-62 Setup Terminal Software 2

(2-3) Set serial port setting values: baud rate **1843200** (RCar S4)/**921600** (RCar V4H), 8bit data, parity none, stop 1 bit, and flow control none as the following figures. Press [OK] button to confirm the setting.

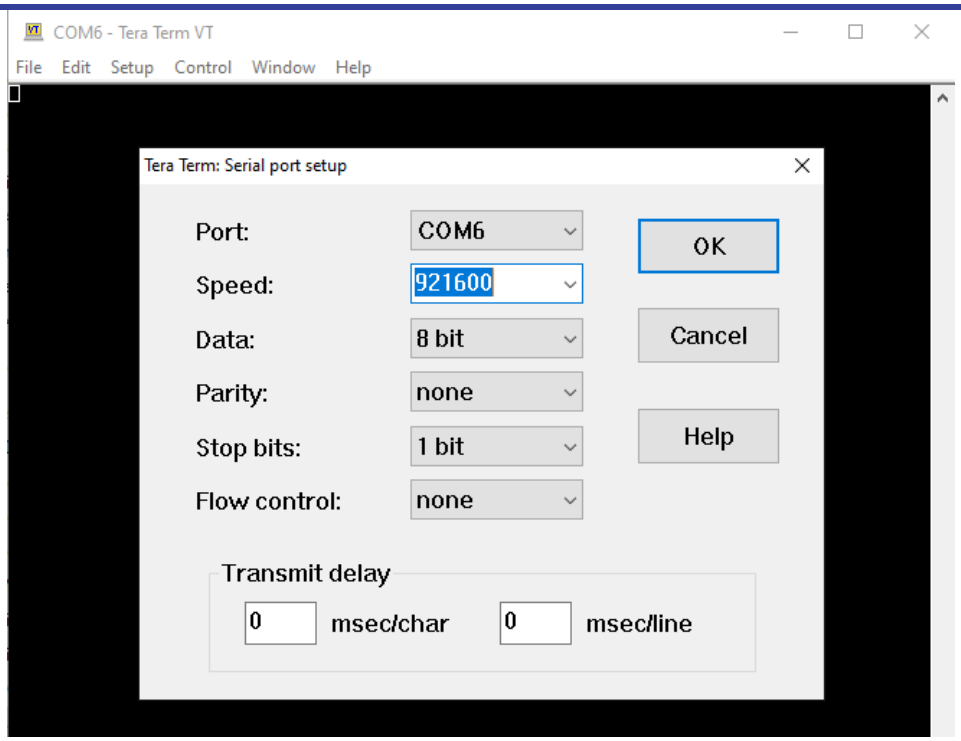

Figure 3-63 Example for Setup Terminal Software in RCar V4H

Step 3 Switch setting (SCIF download mode)

The switch setting to change MD pin setting on evaluation board. The setting to change to Flash writer mode is shown as below,

RCar S4:

❖ The board supports CPLD (e.g., S4 Spider A0 1st board)

- Open S4_Spider_Configurator.exe software
- Choose the setting following Step 1 to Step 10

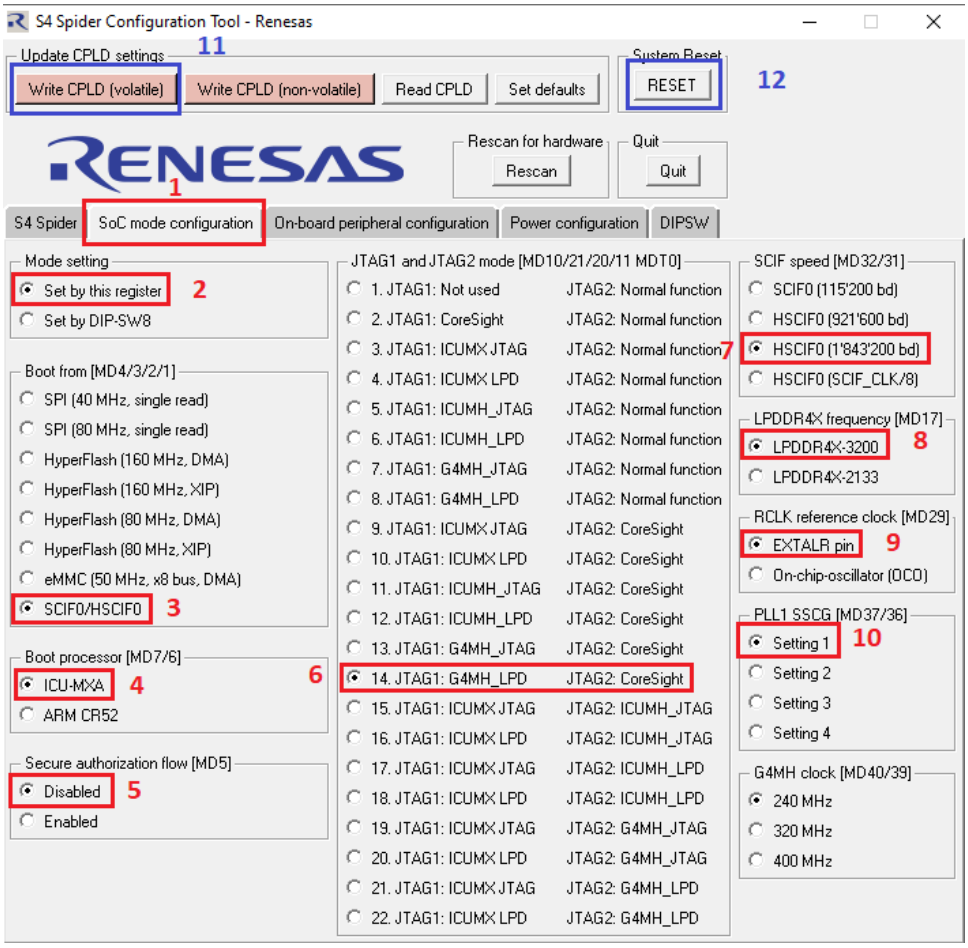

Figure 3-64 Switch setting (SCIF download mode) 1

- Step 11: Select Write CPLD (volatile) - write setting data to RAM
- Step 12: Click to "RESET" button to reset Spider board (if the board is not reset, please check whether disconnect E2 debugger or not)

❖ The board does not support CPLD (e.g., S4 Spider B0 2nd board)

Spider board CPU board

SW1	Pin1 side (dot side)
SW2	Center

Spider Mode Switch board

SW1

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	OFF	ON	OFF	OFF	OFF	OFF

SW2

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	ON	ON	ON	OFF	OFF	OFF

SW3

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	OFF	ON	ON	OFF	OFF	ON	OFF

SW4

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	ON	ON	ON	ON	OFF	OFF	ON

SW5

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Figure 3-65 Spider Mode Switch board

On the bottom of Spider Mode Switch board

- **SW6**, **SW7**, **SW8** and **SW10** are always set to **ON**.
- **SW9** does not related to MD pin setting. It should be always set to **OFF**.

Note:

- E2 debugger connects to S4 Spider CPU board via JTAG1 (CN1 - Debugger0)
- Lauterbach Trace32 debugger connects to Spider CPU board via JTAG2 (CN2 - Debugger1)
- Turn ON SW4 (nearly CN1 of Spider CPU board) to enable Lauterbach Trace32 power on reset S4 SoC

RCar V4H:

White Hawk CPU board

SW13	Dot side
SW57	Center

White Hawk Mode Switch board

SW1

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	OFF	ON	OFF	OFF	OFF	OFF

SW2

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	ON	ON	OFF	ON	OFF	OFF

SW3

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	ON	ON	ON	OFF	ON	OFF	OFF

SW4

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	ON	ON	ON	ON	ON	ON	ON

SW5

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	OFF	OFF	ON	ON	OFF	OFF	OFF

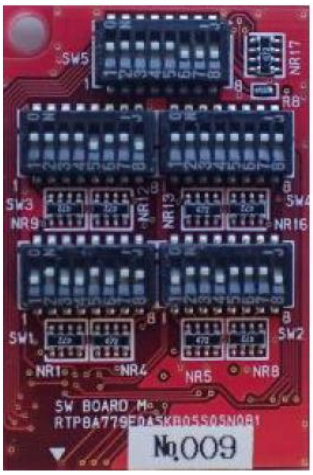

Figure 3-66 White Hawk Mode Switch board

RCar V4M:

White Hawk CPU board

SW13	Dot side
SW57	Center

White Hawk Mode Switch board

SW1

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	OFF	ON	OFF	OFF	OFF	OFF

SW2

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	ON	ON	OFF	ON	OFF	OFF

SW3

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	ON	ON	ON	OFF	ON	OFF	OFF

SW4

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	ON	ON	ON	ON	ON	ON	ON

SW5

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	OFF	OFF	ON	ON	OFF	OFF	OFF

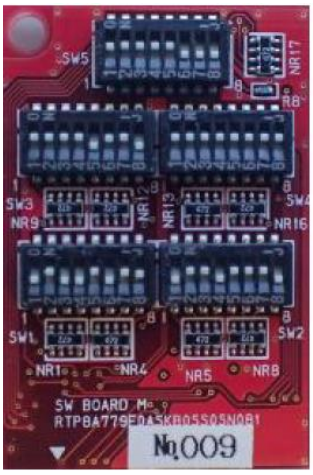

Figure 3-67 White Hawk Mode Switch board

On the bottom of White Hawk Mode Switch board

- **SW6**, **SW7**, **SW8** and **SW10** are always set to **ON**.
- **SW9** does not related to MD pin setting. It should be always set to **OFF**.

Step 4 Power ON CPU board

RCar S4: SW11, position 1 (ON side)

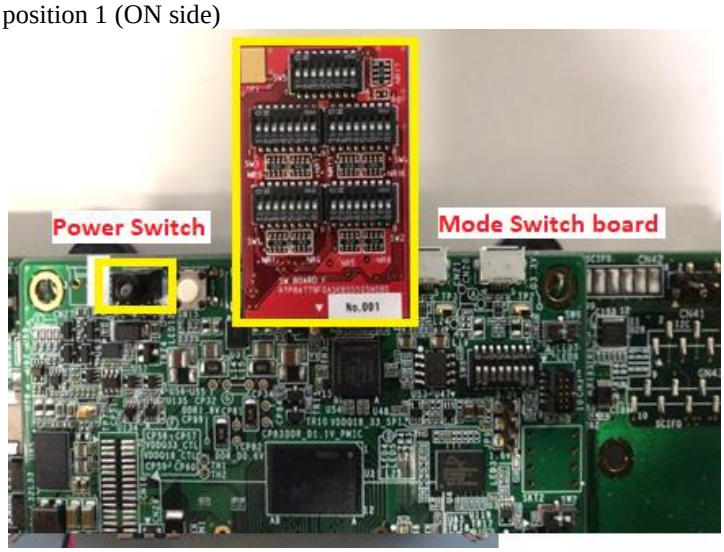

Figure 3-68 S4 Spider Power Switch

RCar V4H: SW51**,** position 1 (ON side)

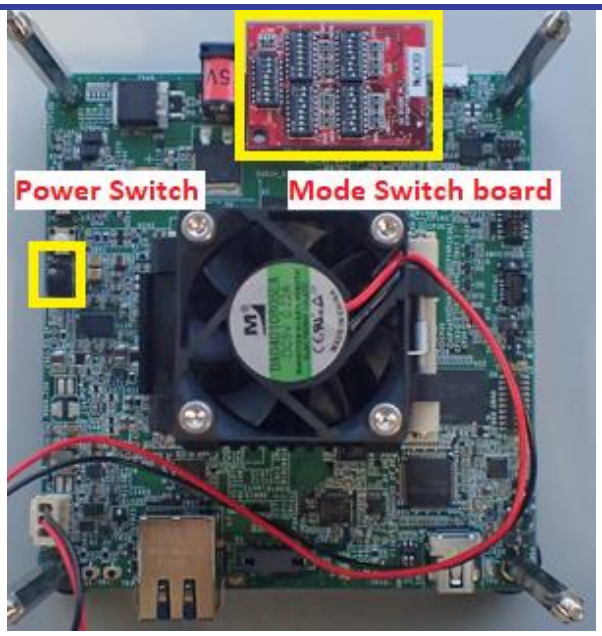

Figure 3-69 V4H White Hawk Power Switch

RCar V4M: SW51**,** position 1 (ON side)

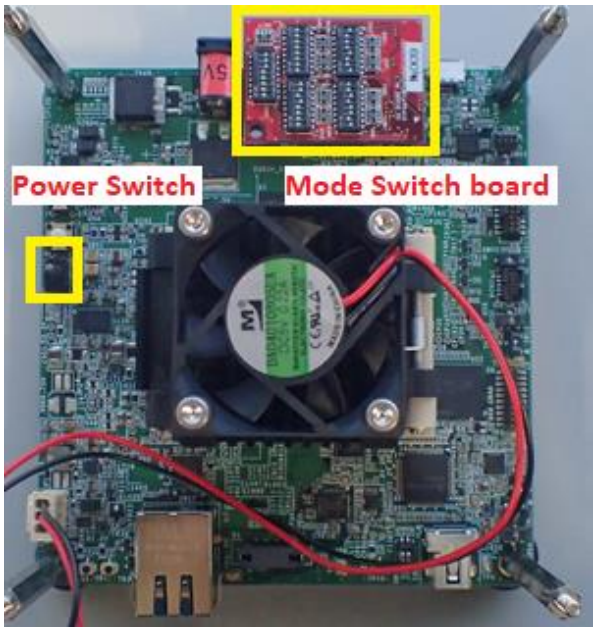

Figure 3-70 V4M Gray Hawk Power Switch

Since the SCIF download mode setting was be set in **Step 3**, the result in the Tera Term is shown as below,

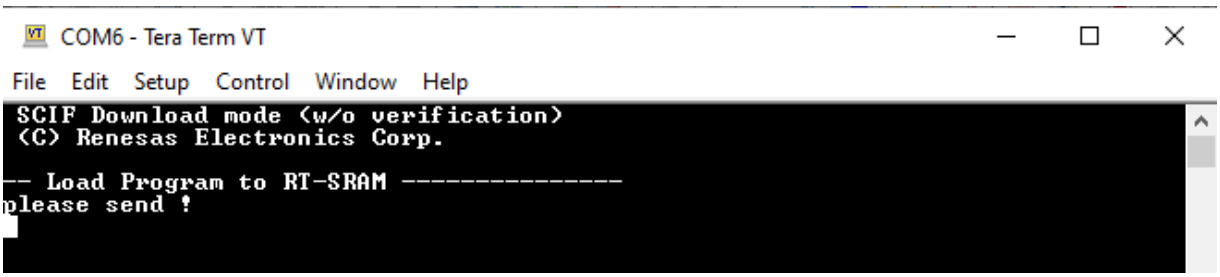

Figure 3-71 SCIF download mode log on Tera Term

Step 5 Load Flash Writer driver

RCar S4:

Drag and Drop "*ICUMX_Flash_writer_SCIF_DUMMY_CERT_EB203000_S4.mot*" into Terminal Software to transfer Flash Writer. After Flash writer loading finished, we will receive the result as below figure,

RCar V4H:

Drag and Drop "*ICUMX_Flash_writer_SCIF_DUMMY_CERT_EB203000_V4H.mot*" into Terminal Software to transfer Flash Writer. After Flash writer loading finished, we will receive the result as below figure,

RCar V4M:

Drag and Drop "*ICUMX_Flash_writer_SCIF_DUMMY_CERT_EB203000_V4M.mot*" into Terminal Software to transfer Flash Writer. After Flash writer loading finished, we will receive the result as below figure,

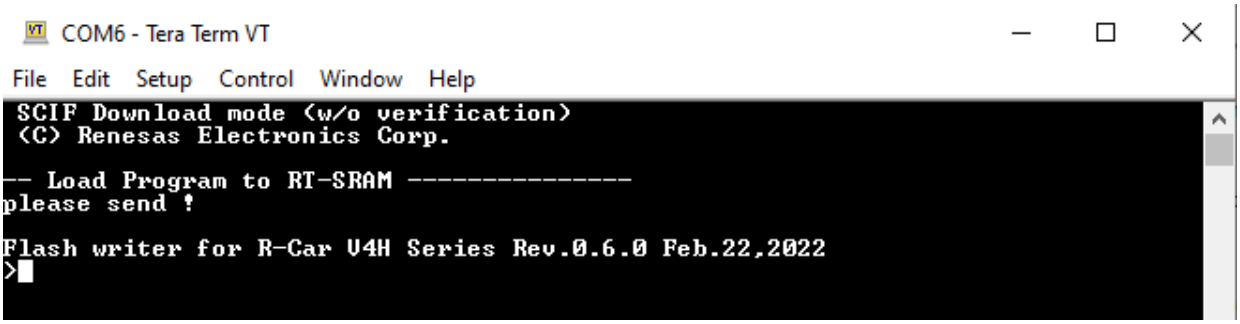

Figure 3-72 Flash Writer download successful

Note: Above figure is examble for V4H device with Flash Writer Rev.0.6.0. The Flash Writer log and it's version may be difference based on each device and Flash Writer version.

Step 6 Write data file to the Serial Flash.

- RCar Gen4 IPL supports to store or load image from Flash memory or eMMC memory or both of them. For more information, refer to section 4.2 Release image of reference document *[REF-4] ICUMX IPL for R-Car Gen4 User's Manual*.
- To write each S-record format images, please refer to IPL image writing guideline in chapter **3.6 How to use command** of reference document *[REF-5] R-Car Gen4 Flash Writer sample software* to use command **xls2** (Write S-record format images to the QSPI Flash and HyperFlash) and **em_w** (Write S-record format images to eMMC) write data to Serial Flash and eMMC memory
- Base on the IPL image writing guideline in chapter **3.6 How to use command** of reference document *[REF-5] R-Car Gen4 Flash Writer sample software*, user can self-write a 'ttl' file extension. This file is for automate processes in Tera Term by using macro. Otherwise, user can do step by step as the guideline.

The following illustrate the steps in case images are stored in both Flash memory and eMMC memory and used Tera Term macro ('ttl' file extension) to flash IPL for V4H device. Similar for S4 device.

(6-1) Prepare "Flash_Bootloader_V4H.ttl" on Windows PC and then click [Control] -> [Macro] on Tera Term Terminal.

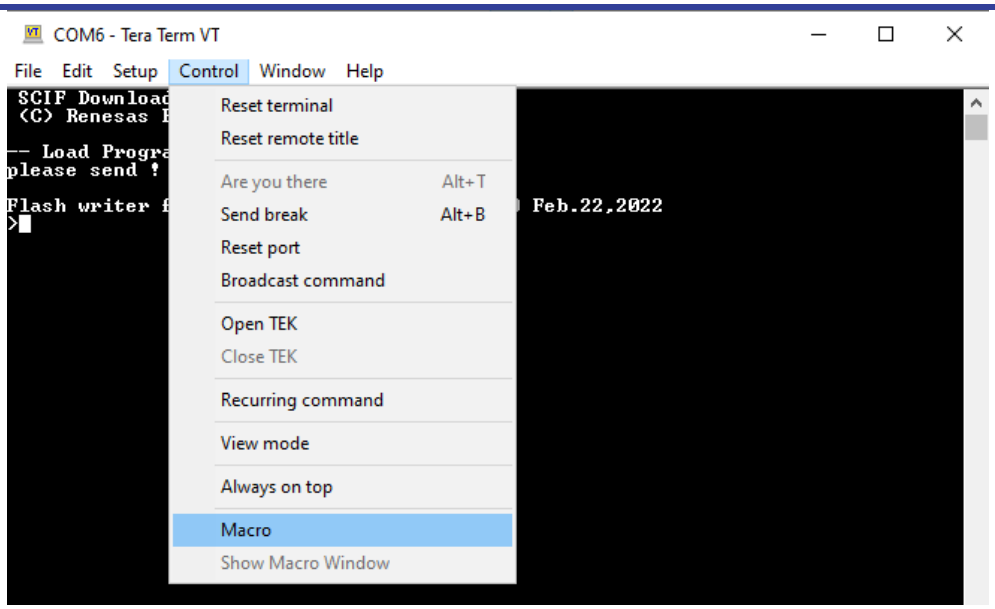

Figure 3-73 Flashing IPL 1

(6-2) Point to the path where “Flash_Bootloader_V4H.ttl” is stored and select it to execute.

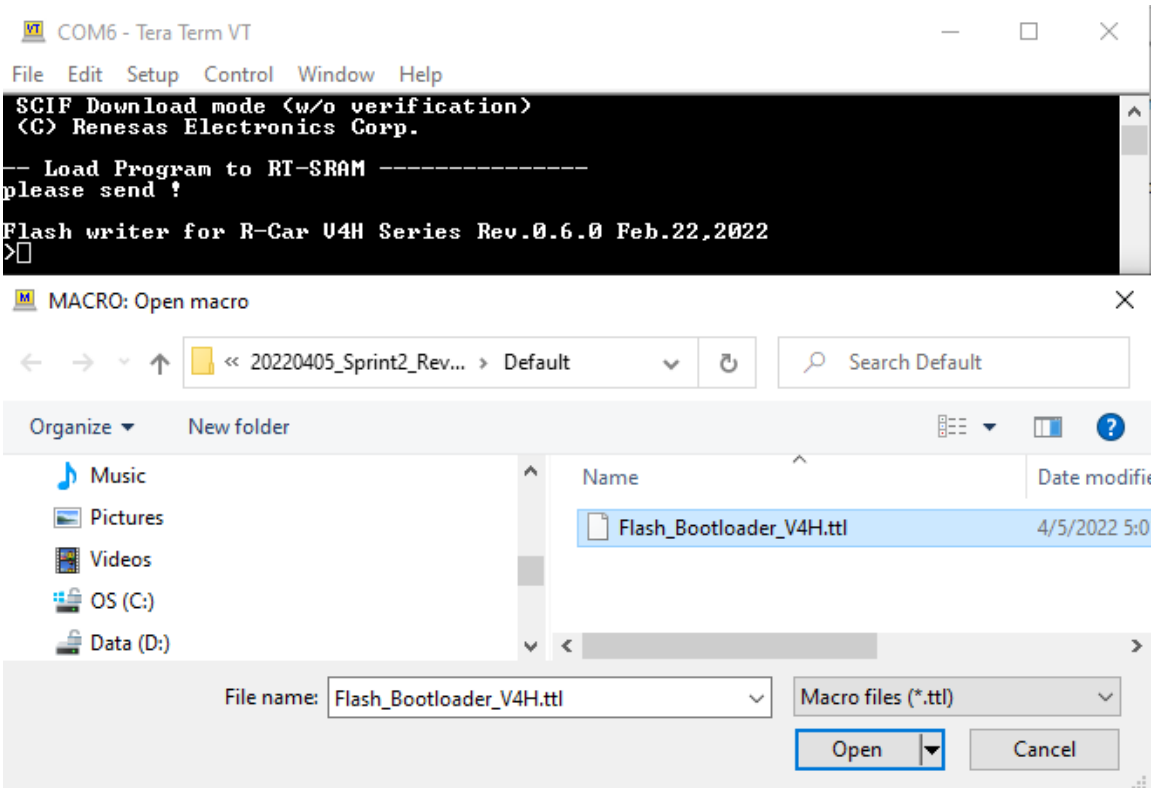

Figure 3-74 Flashing IPL 2

(6-3) When the macro execution is completed, the following log will be output.

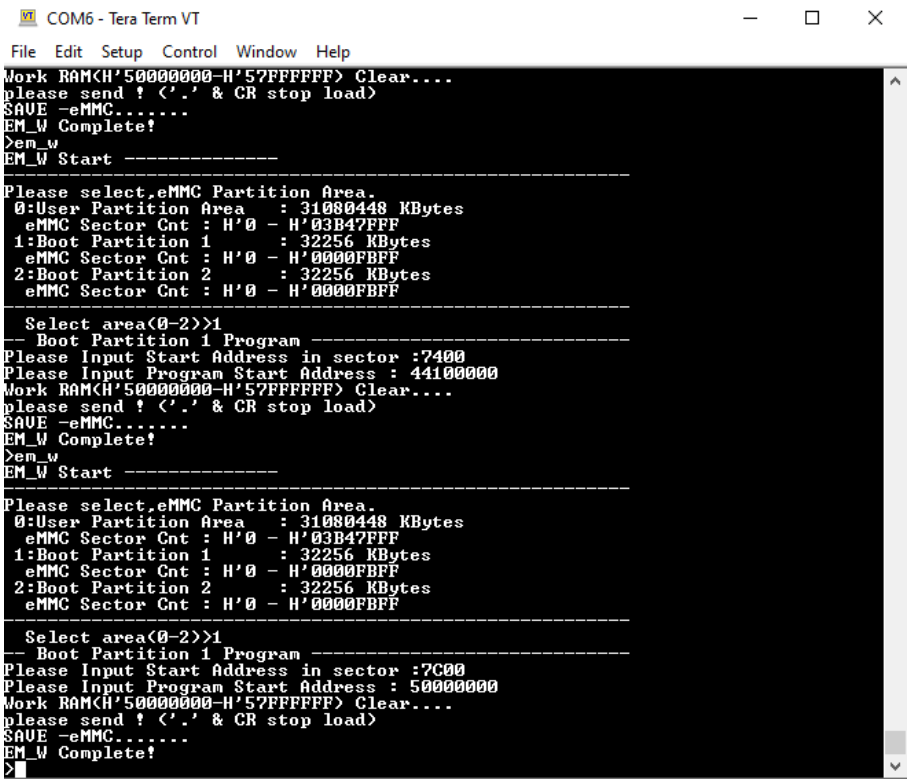

Figure 3-75 Flashing IPL 3

Step 7 Back to Normal operation mode

RCar S4:

❖ The board supports CPLD (e.g., S4 Spider A0 1st board)
- Open S4_Spider_Configurator.exe software
- Keep others setting in **Step 3**
- Following the Step1 to Step3 in the figure below to change the S4 board to normal operation mode.

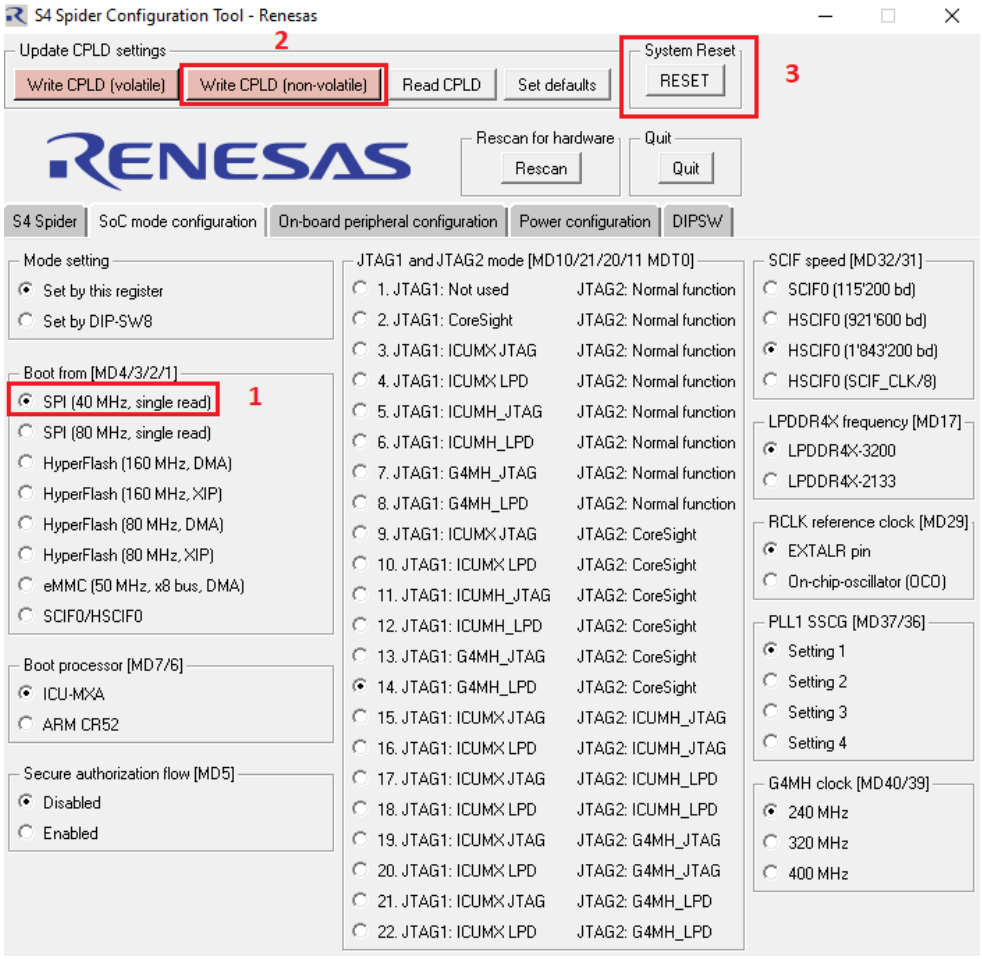

Figure 3-76 S4 CPLD normal mode setting

❖ The board does not support CPLD (e.g., S4 Spider B0 2nd board)

- Power OFF (SW11, position 3) CPU board
- Change SW1, SW2 and SW3 as the followings

SW1

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	OFF	ON	**ON**	**OFF**	**ON**	**ON**

SW2

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	**OFF**	**ON**	**OFF**	**OFF**	**ON**	OFF

SW3

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
ON	OFF	ON	ON	OFF	**OFF**	**ON**	OFF

Note: SW4 and SW5 are not neccesary to change from **Step 3**.
The Terminal Software setting should not be changed from **Step 2**.

RCar V4H:

- Power OFF (SW51, position 3) CPU board
- Change **SW1** and **SW2** as the followings

SW1

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	OFF	ON	**ON**	OFF	**ON**	**ON**

SW2

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	ON	**OFF**	**ON**	ON	**ON**	OFF

Note: SW3, SW4 and SW5 are not neccesary to change from **Step 3**.
The Terminal Software setting should not be changed from **Step 2**.

RCar V4M:

- Power OFF (SW51, position 3) CPU board
- Change **SW1** and **SW2** as the followings

SW1

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	OFF	ON	**ON**	OFF	**ON**	**ON**

SW2

pin1	pin2	pin3	pin4	pin5	pin6	pin7	pin8
OFF	ON	ON	**OFF**	**ON**	ON	**ON**	OFF

Note: SW3, SW4 and SW5 are not neccesary to change from **Step 3**.
The Terminal Software setting should not be changed from **Step 2**.

Step 8 Power ON CPU board and boot up, the startup log shown as below

- **RCar S4:** SW11, position 1 (ON side). (If the board supports CPLD, the board is already ON, SW11 does not need to change)
- **RCar V4H:** SW51, position 1 (ON side)
- **RCar V4M:** SW51, position 1 (ON side)

Note: The flowing figure is an examble for RCar V4H IPL Rev.0.4.0 after V4H SoC is bootup. The log information may be difference depended on each device and IPL driver.

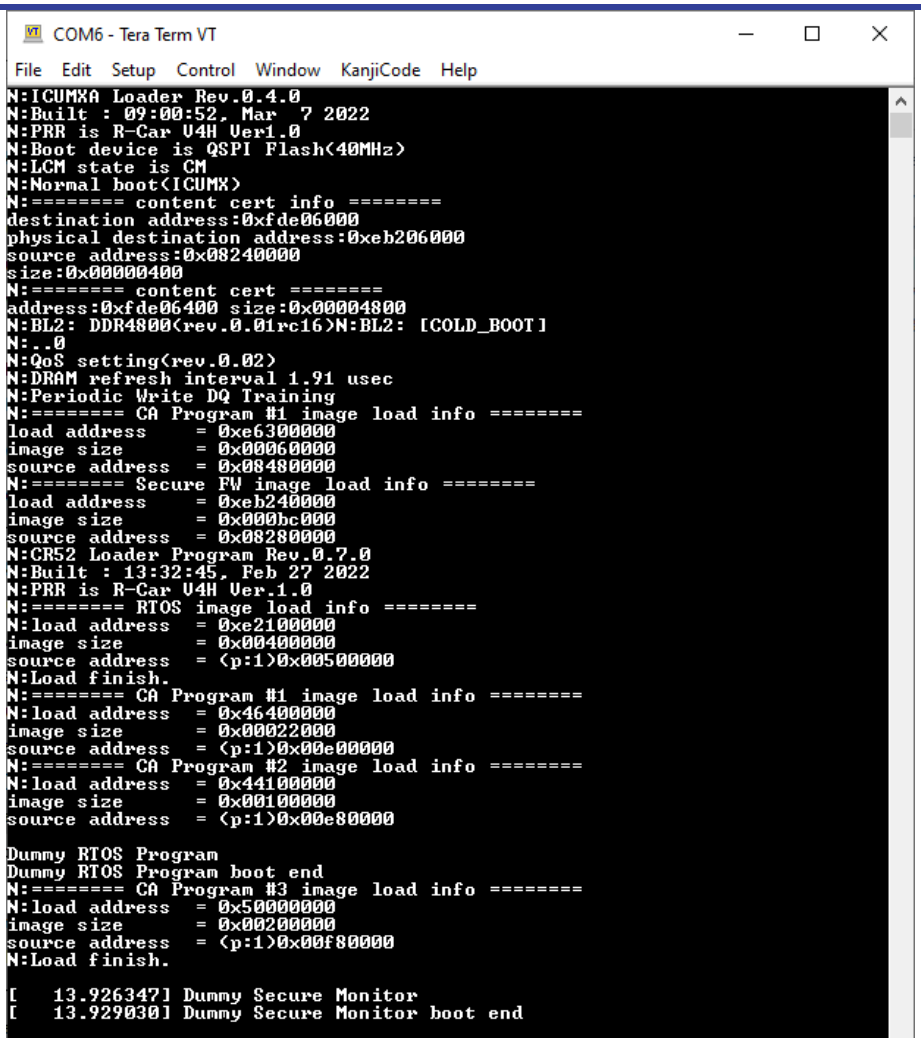

Figure 3-77 Normal operation mode's log

3.7.3.1.4 How to run a MCAL Sample Application by Lauterbach Trace32 debugger

- Please perform the steps in section *3.7.3.1.3 Flash bootloaders* and the caution in section *3.7.3.1 Precaution* of this document before running the MCAL sample application.
- Lauterbach Trace32 debugger is used to debug for ARM Cortex-R core.

Step 1: Install Lauterbach Trace32 software on Host PC.
Step 2: Go to *<install_path>\bin\windows64* and duple-click to **t32start.exe** to open **T32Start** software.
Step 3: Configuration on T32Start software to use Lauterbach Trace32 debugger same as below (in this examble, Lauterbach software is installed to C:\T32_armv8 folder)

Figure 3-78 Setting Lauterbach Trace32 debugger

After that click to **[Save]** button to save the configuration for the next time.

Step 4: Connect Lauterbach Trace32 debug cable to evaluation board

RCar S4: via CN2 in S4 Spider CPU board. Then connect Lauterbach Trace32 USB connection to Host PC via USB terminal.

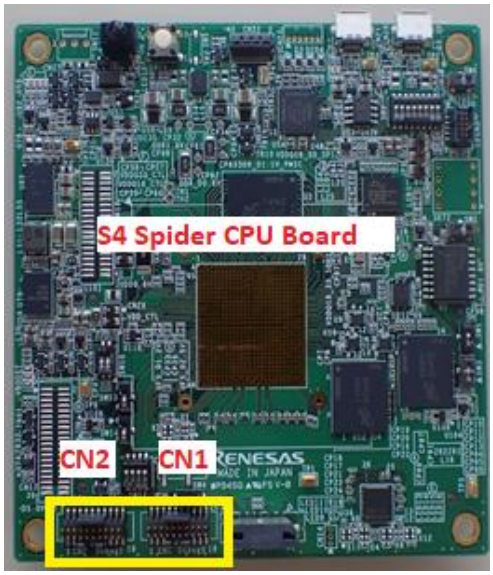

Figure 3-79 S4 Spider CPU board debugger connector CN2

RCar V4H: via CN1 connector in V4H White Hawk CPU board. Then connect Lauterbach Trace32 USB connection to Host PC via USB terminal.

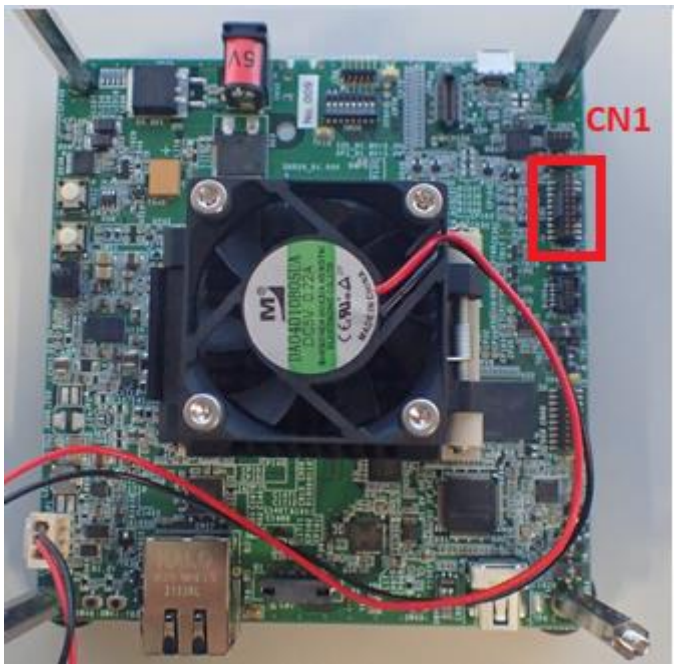

Figure 3-80 V4H White HawkCPU board debugger connector

RCar V4M: via CN1 connector in V4M Gray Hawk CPU board. Then connect Lauterbach Trace32 USB connection to Host PC via USB terminal.

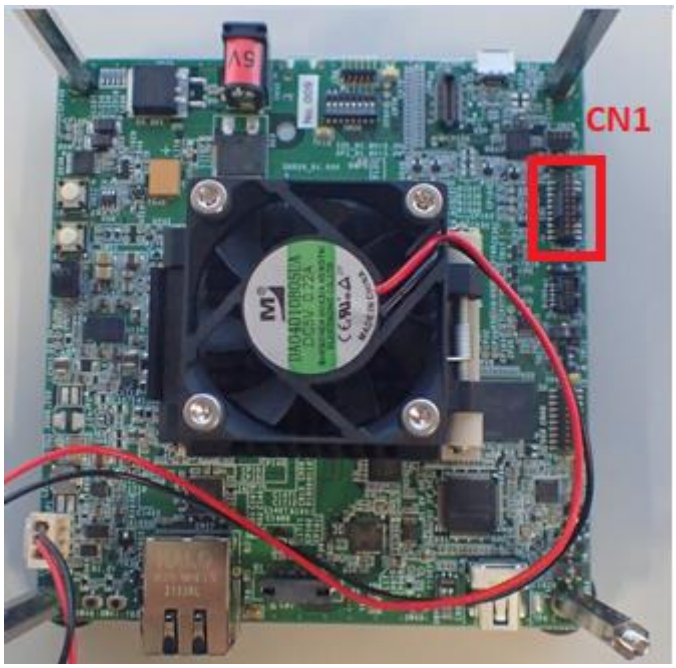

Figure 3-81 V4M Gray HawkCPU board debugger connector

Step 5: On Host PC, click to **[Start]** button of **T32Start** software, the main windows of Lauterbach Trace32 debugger will lauch same as figure below

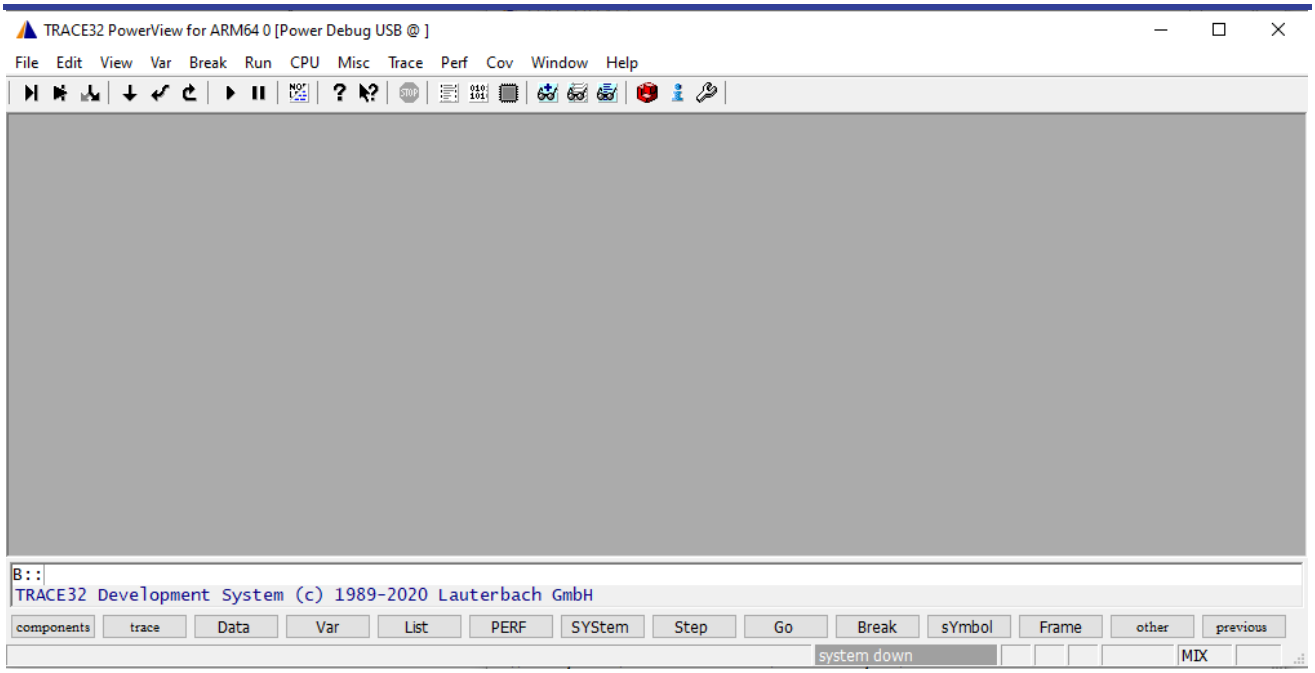

Figure 3-82 Lauterbach Trace32 debug view

Step 6: Open Lauterbach Trace32 linkup script

(6-1) Click to [File] -> [Open Script…]

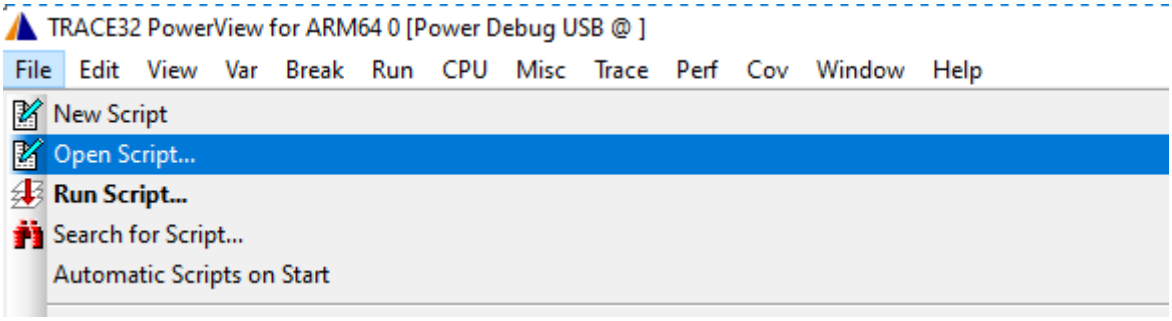

Figure 3-83 Open Lauterbach Trace32 script 1

(6-2) On the pop-up windows, browse to Lauterbach Trace32 linkup script file (*.cmm).
Select linkup script file and click to **[Open]** button

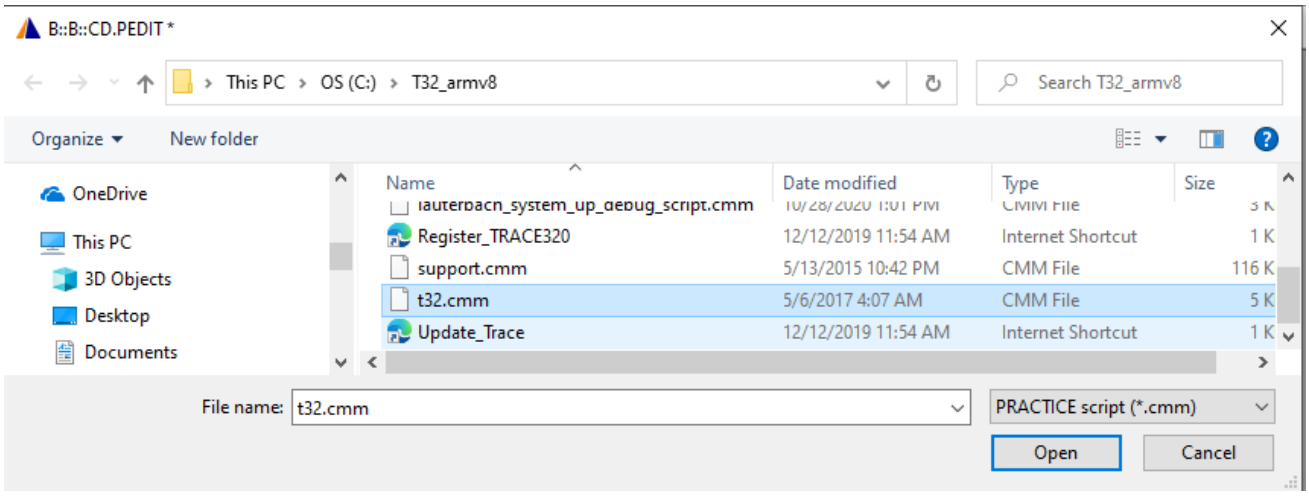

Figure 3-84 Open Lauterbach Trace32 script 2

Below is an example of Lauterbach Trace32 linkup script. Please prepare it by your self.

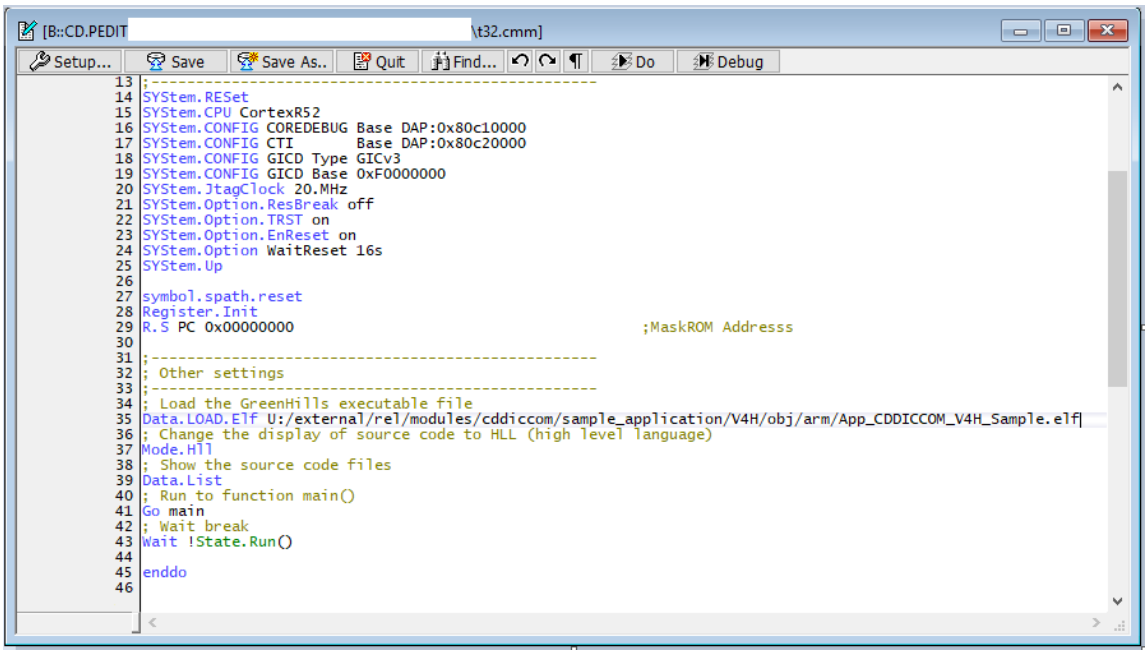

Figure 3-85 Lauterbach Trace32 script

Step 7: Prepare the MCAL binary file (*.elf) by build MCAL Sample Application following description in section *3.7.2.2 How to build the Sample Application* of this document. After that please update the path to *.elf file in Lauterbach Trace32 linkup script (line 35 **Data.LOAD.Elf** in above example)

Step 8: On Lauterbach Trace32 linkup script windows, click to [Debug] button

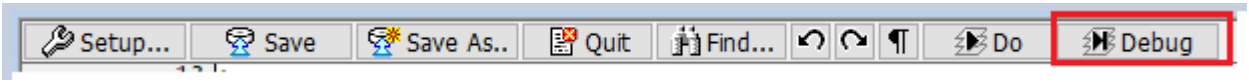

Figure 3-86 Lauterbach Trace32 debug button

On the pop-up windows, go to **Wait!State.Run()** command, then right-click and select [Go Till]

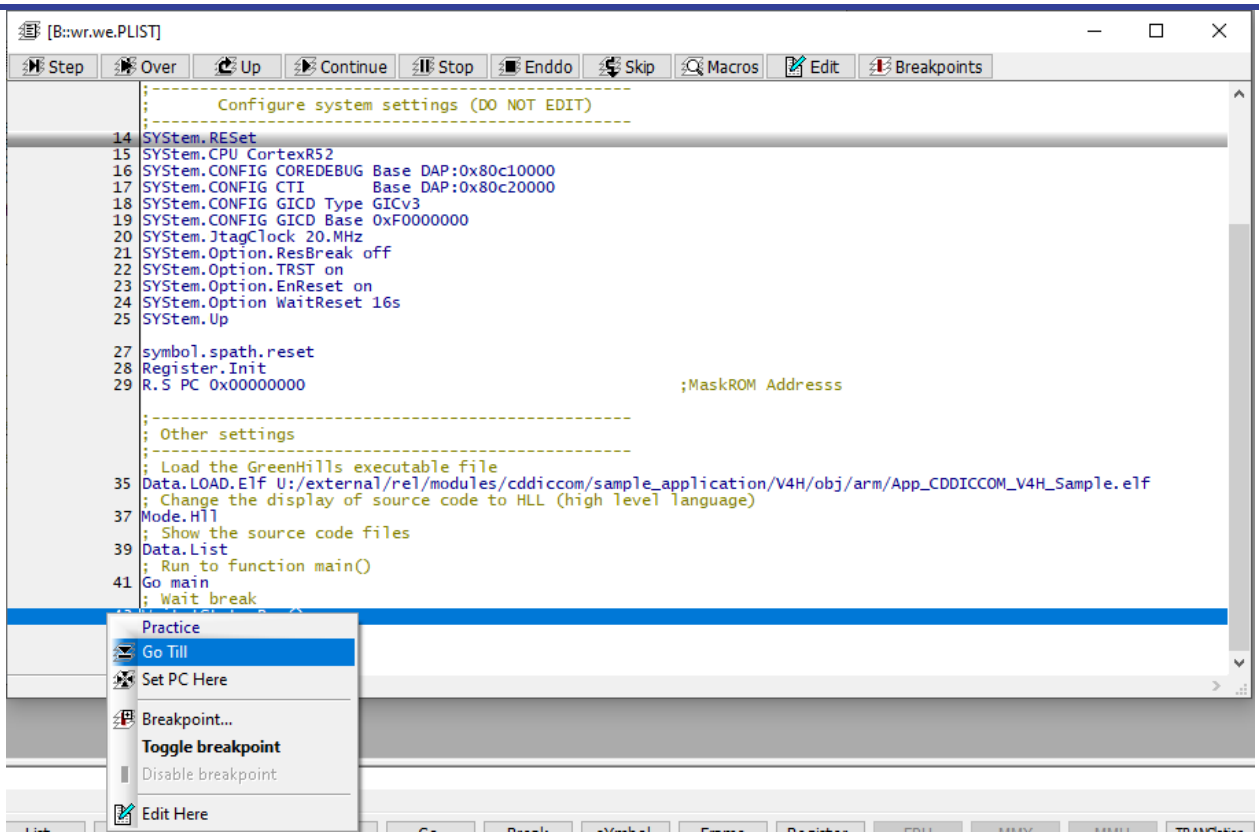

Figure 3-87 Lauterbach Trace32 debug script using

Step 9: When the program stops at **Wait!State.Run()** command, there is another pop-up window to show program stopped at beginning of MCAL driver main() function

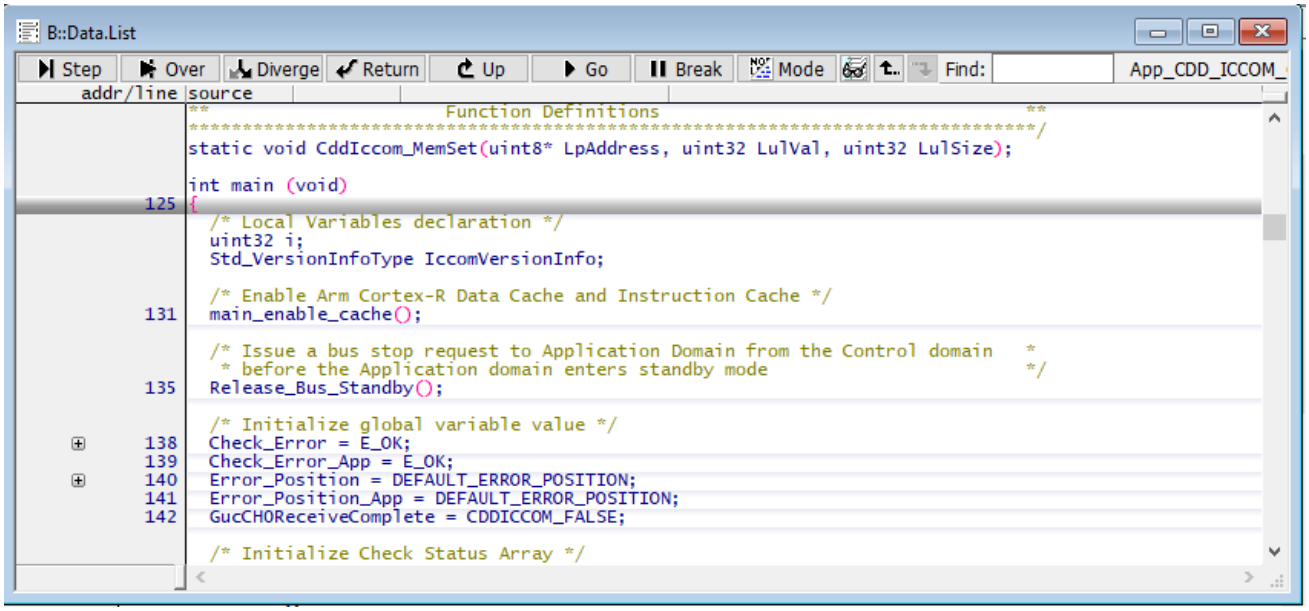

Figure 3-88 MCAL main() function in Lauterbach Trace32

Step 10: Refer to section *(5) How to run <MSN> Sample Application* of each module section to execute and check result of <MSN> Sample Application.

If module supports serial print log, the sample app execution proccess and result will be shown in the Tera Team.

E.g., At the end of MCAL ICCOM Sample Application program, the result will be same as figure below

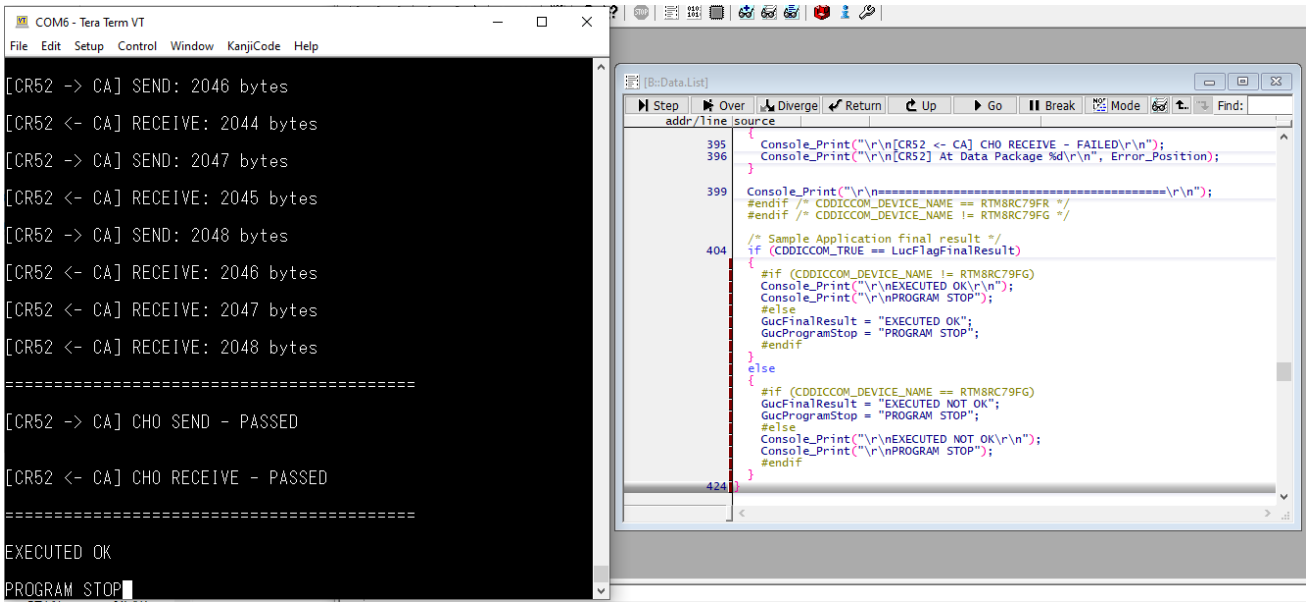

Figure 3-89 MCAL Sample Application execution result

3.7.3.1.5 **How to run a MCAL Sample Application by Renesas E2 debugger**

- Please perform the steps in section *3.7.3.1.3 Flash bootloaders* and the caution in section *3.7.3.1 Precaution* of this document before running the MCAL sample application.
- Renesas E2 debugger is used to debug for RH850 G4MH core. Currently, it is applicable for S4 device.

Step 1: Prepare board and connection

- Turn on the power switch (SW11 on S4 Spider CPU board) of the evaluation board.
- Connect Renesas E2 debugger to CN1 connector on S4 Spider CPU board.

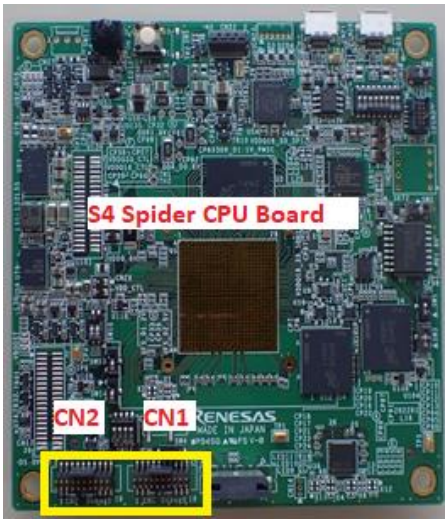

Figure 3-90 S4 Spider CPU board debugger connector CN1

Step 2: Run CS+ to create a new project by select [File] -> [New] -> [Create New Project…].

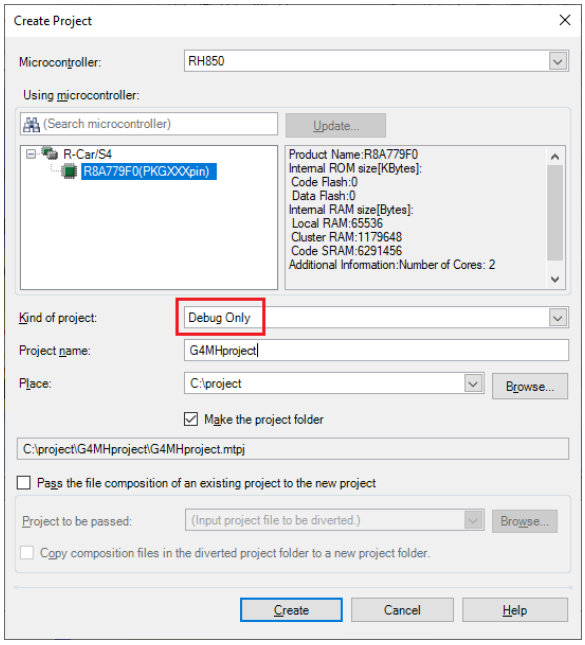

Figure 3-91 Create a new project

If you do not want to build under CS + environment, choose “Debug Only”.
Enter a project name and create one.

Step 3: Select E2 Emulator

Right click to [RH850 Simulator (Debug Tool)] -> Select [Using debug Tool] -> Select [RH850 E2]

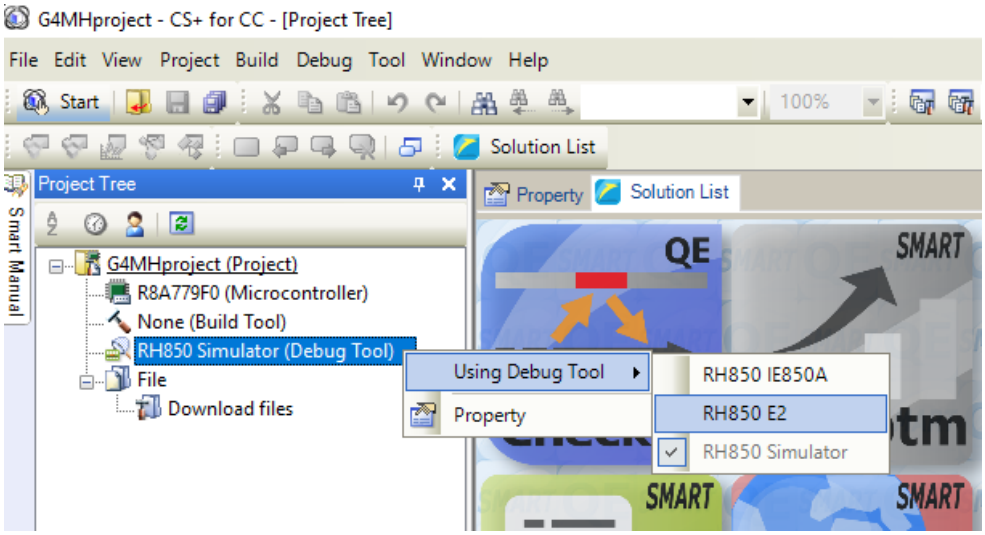

Figure 3-92 Select E2 Emulator

Step 4: Setup for E2 Emulator

(4-1) On the Property field of "**RH850 E2 (Debug Tool)**", tab "**Connect Settings**":

- Set **Flash** OCD ID to "**00000000000000000000000000000000000000000000000000000000000000000000000000000000**"
- Set **Customer ID** ID to "**00000000000000000000000000000000000000000000000000000000000000000000000000000000**"
- Set **Memory** Work RAM start addresses to "**FDC0F000**"

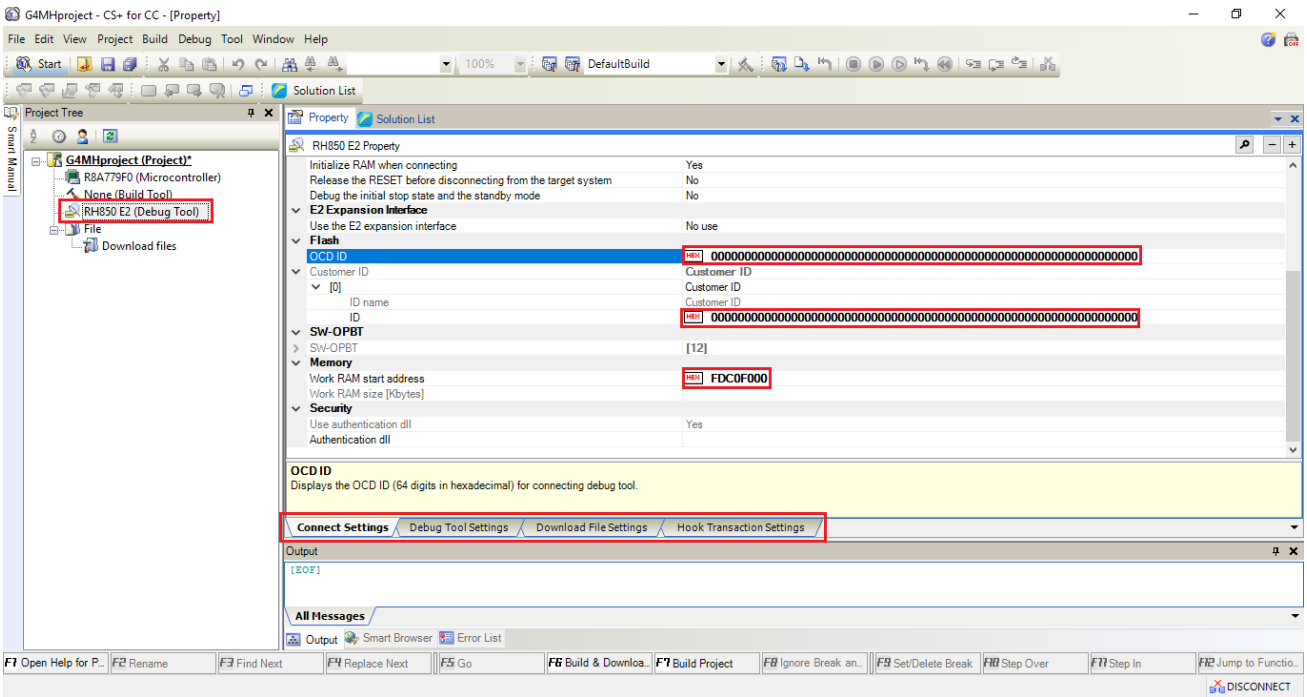

Figure 3-93 Setup for E2 Emulator1

(4-2) For **SW-OPBT** table, set to below value

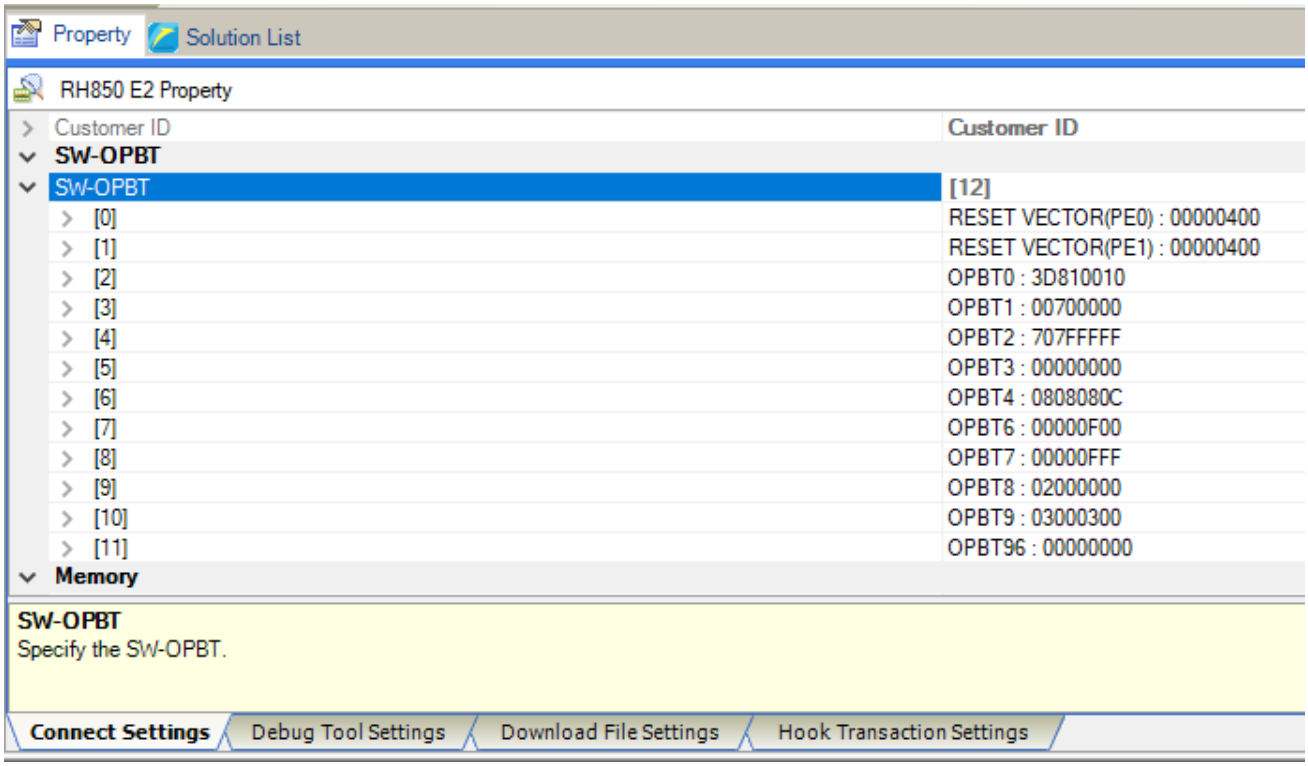

Figure 3-94 Setup for E2 Emulator2

(4-3) On the tab "**Debug Tool Settings**", setup **Memory** Verify on writing to memory set to **No**

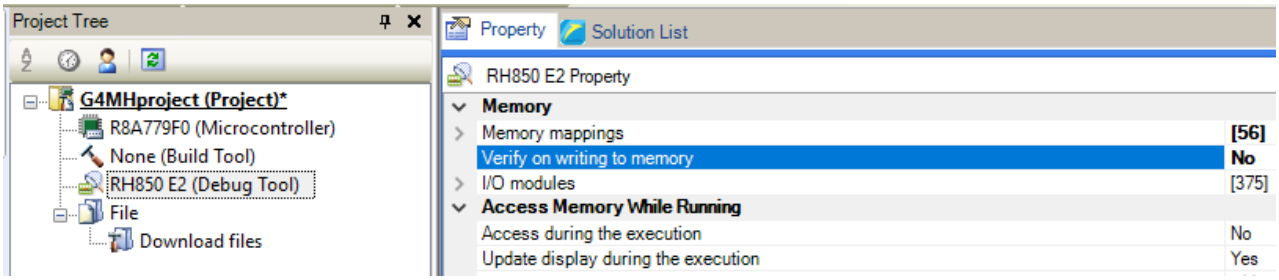

Figure 3-95 Setup for E2 Emulator3

(4-4) On the tab "**Download File Settings**",

- Setup Download -> **CPU Reset after download** to **No**
- Setup Debug Information -> **Execute to the specified sysbol after CPU Reset** to **No**

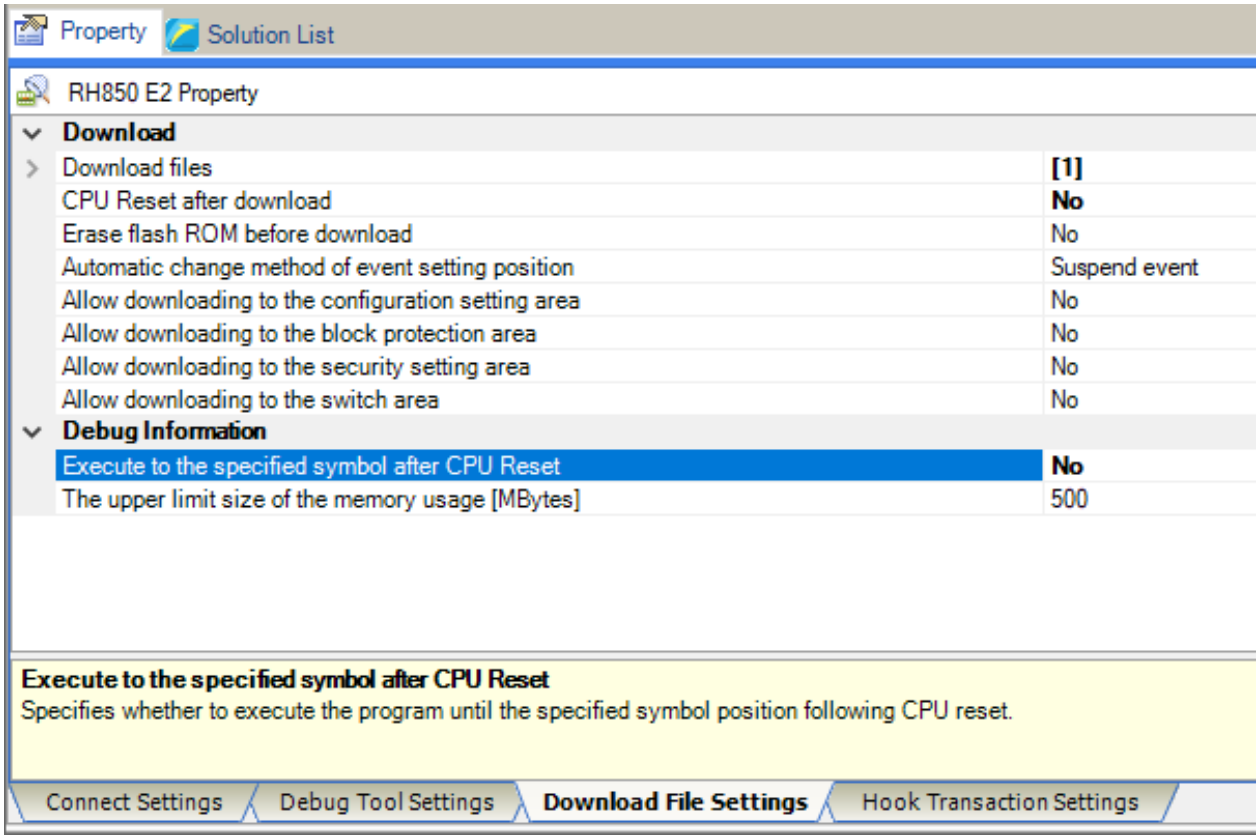

Figure 3-96 Setup for E2 Emulator4

Step 5: Drag Sample Application binary file (*.out) to the "Download files".

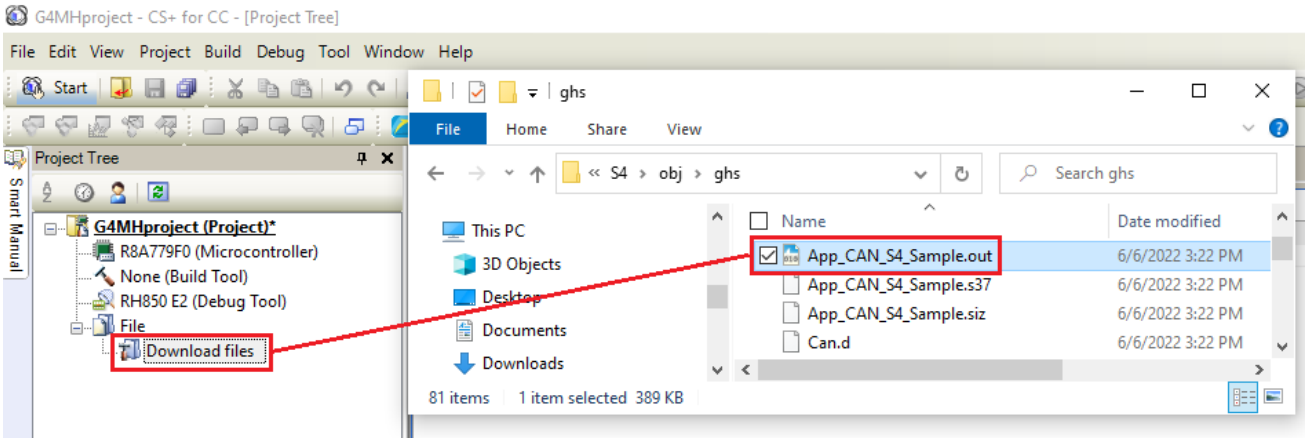

Figure 3-97 Setting of Sample App binay file

Step 6: Select "**RH850 E2 (Debug Tool)**", and press [Debug] -> [Connect to Debug Tool]

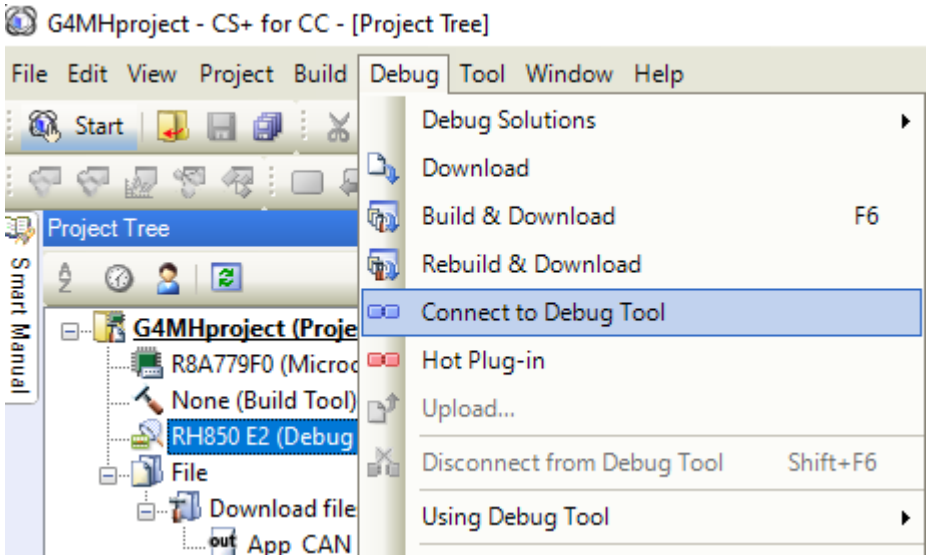

Figure 3-98 Connet to Debug Tool

Step 7: Build and Download binary file to RCar Evaluation board

After successful connect with E2 Emulator at **Step 6**, on the Toolbar select [Debug] -> [Build & Download] to download binary file

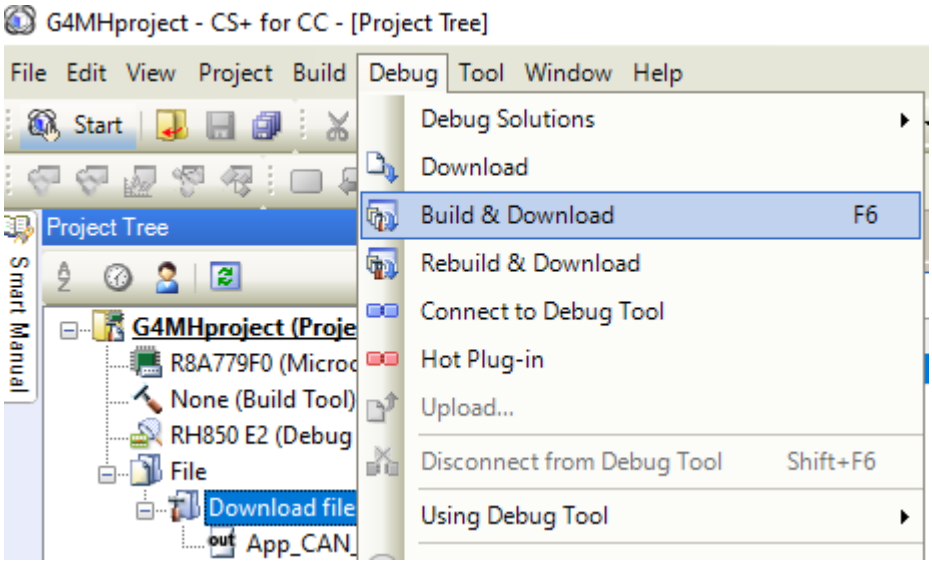

Figure 3-99 Build and Download binary file

Step 8: Run to MCAL main() function

(8-1) On the **Project Tree** panel, double-click to "App_<MSN>_Common_Sample.c" to open the MCAL module Sample Application source code.

(8-2) Set breakpoint in the beginning of main() function by double-click to this LoC.

(8-3) Click to **Go** button on the CS+ toolbar, the program will run immediately. After that it stops at the breakpoint same as figure below.

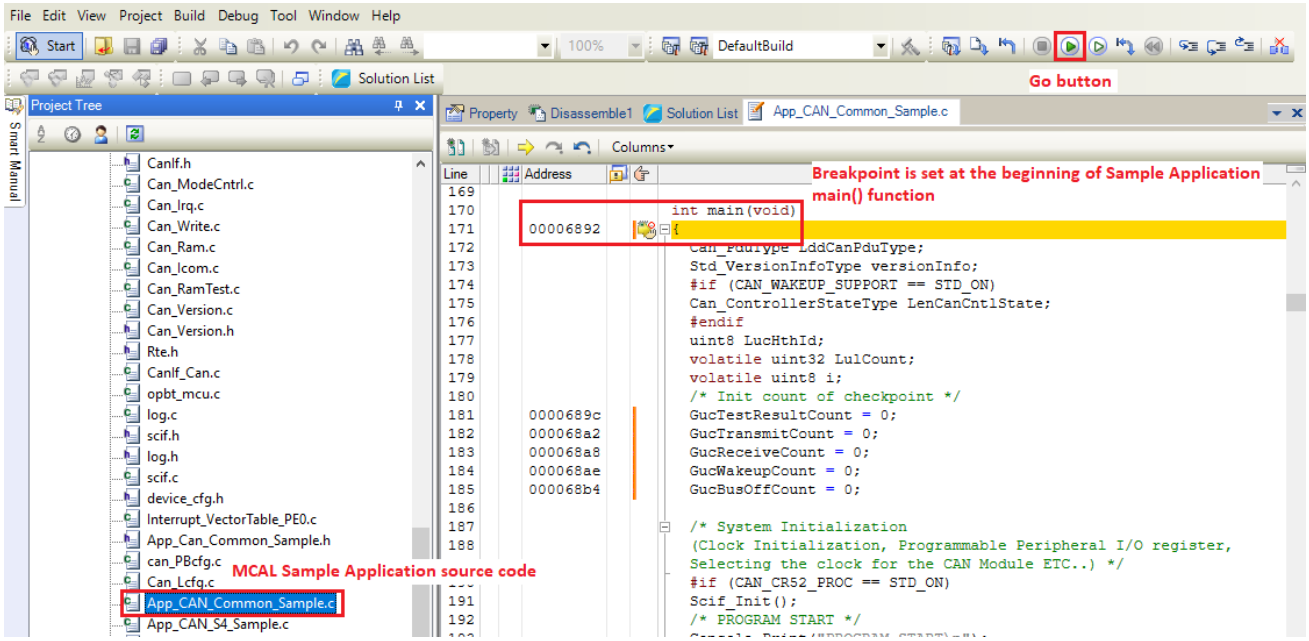

Figure 3-100 Run to MCAL main() on CS+

For the next step, refer to 'Sample Application' section of each module in Section *3.3 MCAL Module* and Section *3.4 CDD module* of this document.

Caution: Do not reset G4MH CPU after **Step 7**, the MCAL driver operation will not guarantee if G4MH CPU is reset. There is a workaround to reset G4MH CPU. It is disconnected Renesas E2 debugger in CS+ project and repeat **Step 6** to **Step 8** for safely reset G4MH CPU and run MCAL driver.

3.7.3.1.6 **How to load MCAL Sample Application binary file on the evaluation board**

Note:

<DeviceName> is S4, V4H, V4M.
<msn> is can, dio, … for MCAL module and cddiccom, cddem, … for CDD module.
<MSN> is CAN, DIO, … for MCAL module and CDD_ICCOM, CDD_EMM, … for CDD module.

Step 1: Following the guideline in section *3.7.2.2 How to build the Sample Application* to build MCAL Sample Application, the MCAL Sample Application binary file is located at

- ARM Compiler:

rel\modules\<msn>\sample_application\<DeviceName>\obj\arm\App_<MSN>_<DeviceName> _Sample.srec

- GHS Compiler:

rel\modules\<msn>\sample_application\<DeviceName>\obj\ghs\App_<MSN>_<DeviceName> _Sample.s37

Step 2: Edit MCAL binary file before flashing together with IPL image

- ARM Compiler's output:

Add below binary source to the top of 'App_<MSN>_<DeviceName>_Sample.srec'

```
S012000064756D6D795F72746F732E73726563BF
```

- GHS Compiler's output:

Rename ‘App_<MSN>_<DeviceName>_Sample.**s37**’ to ‘App_<MSN>_<DeviceName>_Sample.**srec**’

Step 3: Add the MCAL Sample Application into corresponding RCar <DeviceName> IPL package

- CR52 domain:

Rename ‘App_<MSN>_<DeviceName>_Sample.srec’ to ‘dummy_rtos.srec’ and replace ‘dummy_rtos.srec’ in RCar <DeviceName> IPL package.

- G4MH domain:

Rename ‘App_<MSN>_<DeviceName>_Sample.srec’ to ‘dummy_g4mh_case0.srec’ and replace ‘dummy_g4mh_case0.srec’ in RCar <DeviceName> IPL package.

Step 4: Flash the MCAL sample application and RCar <DeviceName> IPL to the RCar <DeviceName> evaluation board following the guidelines in section *3.7.3.2 Flash bootloaders.*

After the IPL flashing is complete, turn off the board. Switch settings of the evaluation board to normal mode. When the board is powered on, the IPL will start the MCAL sample application running.

4.Deviation List

The **Table 4-1** shows the AUTOSAR Deviation List for this MCAL package.

Table 4-1 AUTOSAR Deviation List

Sl. No.	Description
1.	Memory class does not conform to AUTOSAR standard [SWS_COMPILER_00040]. Used the following deprecated memory classes. <PREFIX> _APPL_CODE <PREFIX> _VAR_NOINIT <PREFIX> _VAR_FAST <PREFIX> _VAR
2.	The BSWMDT file should be modified by user. [SWS_BSW_00208], [SWS_BSW_00238] When MCAL driver used as multi-instance, interface name will be change as per configuration. The user should create BSWMDT file as per configuration for avoid redundant information.
3	AUTOSAR standard R19-11 SW-C and System Modeling Guide [TR_SWNR_00001] When configure for parameter and container for MCAL module, user should not use too long container short name or parameter value metioned in **Table 4-2** to avoid bad behavior of generation tool output.

For information on the Deviation list of each modules, refer to “Driver Component Embedded User’s Manual” document of each module.

Table 4-2 Deviation List for Generated Output

Description	Container's shortName or Parameter value shall be shorten	Module
#define CNAME value, with: CNAME length may be over 70 characters	All containers existed parameter have SYMBOLIC-NAME-VALUE is TRUE.	ICU, GPT, PWM, CAN, LIN, DIO, MCU, ETH, FR, SPI
	PortGroup container PortPin container	PORT
	DioConfig container DioChannel container	DIO
	McuRamSectorSettingConf container	MCU
#define CNAME value, with: value length must not be over 80 characters	DemEventParameter container	ICU, GPT, PWM, CAN, LIN, PORT, WDG, MCU, ETH, FR, SPI
	OsCounter container	CAN
	CanLPduReceiveCalloutFunction parameter	CAN

5.Required Operation Conditions

The required operation conditions of this product are described in this chapter.

5.1 Product

Table 5-1 shows an overview of the Device.

Table 5-1 Product Overview

Device	R-CAR S4 G4MH	R-CAR S4 CR52	R-CAR V4H	R-CAR V4M
MCAL Product Release Version	Ver19.4.0	Ver19.4.0	Ver19.3.0	Ver19.0.2
AUTOSAR Specification Version	R19-11	R19-11	R19-11	R19-11
Device Manual Version	1.10	1.10	1.10	0.50
Operating Precautions	None	None	None	None
Device File Version	-	-	-	-
Device Tested On	R-Car S4	R-Car S4	R-Car V4H	R-Car V4M
Supported Devices	R-Car S4 (Includes S4N)	R-Car S4 (Includes S4N)	R-Car V4H	R-Car V4M
Supported Cores	G4MH	Cortex-R52	Cortex-R52	Cortex-R52
Supported OS	OS agnostic	OS agnostic	OS agnostic	OS agnostic

5.2 Compiler

Table 5-2 shows the compiler options.

Table 5-2 Compiler ‘s Option and Information

Tool	Version	Option
Green Hills C Compiler	Green Hills Multi V7.1.6 Compiler Version 2020.1.5	-c -Ogeneral -g -cpu=rh850g4mh -gsize -prepare_dispose -inline_prologue -sda=all -passsource -Wundef -no_callt -reserve_r2 --short_enum --prototype_errors --diag_error 193 -dual_debug -large_sda --no_commons -shorten_loads -shorten_moves -Wshadow -ignore_callt_state_in_interrupts -delete -additional_sda_reg=0 -rh850_abi=ghs2014 -MMD
ARM Compiler	6.16.2	❖ Compiler: CFLAGS= -c -O3 --target=arm-arm-none-eabi -mcpu=cortex-r52 -mfpu=neon-fp-armv8 -g -gdwarf-2 -fno-short-enums -mfloat-abi=hard -nostdlib -nostdlibinc -mno-unaligned-access -MD -MP -Wundef -std=c99 -pedantic-errors -Werror -Wall -Wextra -Wconversion -Wunused -Wuninitialized -DNDEBUG ❖ Linker: LFLAGS= --cpu=cortex-r52 --datacompressor off --no-startup --entry=Start --callgraph --map --info=sizes --info=totals --info=unused --info=compression --info=stack --info=inline --diag_error=warning ❖ Assembler: AFLAGS= -c -O3 --target=arm-arm-none-eabi -mcpu=cortex-r52 -mfpu=neon-fp-armv8 -g -gdwarf-2 -fno-short-enums -mfloat-abi=hard -nostdlib -nostdlibinc -mno-unaligned-access -MD -MP -Wundef -std=c99 -pedantic-errors -Werror -Wall -Wextra -Wconversion -Wunused -Wuninitialized -DNDEBUG -x assembler-with-cpp

5.3 Configuration Code Generator

Table 5-3 shows the Configuration Code Generator

Table 5-3 Configuration Code Generator’s Option and Information

Tool	Version	Option
DaVinci	5.23	-

5.4 RENESAS configuration

5.4.1 Configuration for Product

See Appendix section 2 and 3 for information on measuring RAM/ROM consumption, stack depth, execution time, and functional testing.

5.4.2 Additional Configuration for Functional Safety Usage (Only ASIL Products)

See MCAL SAN

6.Issue List

6.1 Issue List

Table 6-1 shows information of Issue List

Table 6-1 Information of Issue List

Issue Type	Issue Content
Fixed issues	Refer to the fixed issues list shared from Renesas.
Known Issues	Refer to the known issues list shared from Renesas.

Revision History		R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual	
Rev.	**Date**	**Description**	
		Page	**Summary**
4.00	Mar 27, 2025	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: - Update version of reference documents. Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H
3.02	Feb 28, 2025	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: - Update version of reference documents. Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H
3.01	Jan 22, 2025	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: - Update version of reference documents. Add section 3.5.5 Region ID Access Protection. Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H
3.00	Nov 27, 2024	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: -Update version of reference documents & Add (7.) Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H Chapter 3: Section 3.6: - Update title and content
2.11	Oct 29, 2024	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: -Update version of reference documents Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H. Section 5.2 Compiler: - Update ARM Compiler to 6.16.2

2.10	Nov 28, 2023	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: -Update version of reference documents Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H, R-CAR/V4M.
2.09	Oct 27, 2023	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: - Update version of reference documents Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H. Add V4M attribute and device for all requirements.
2.08	Aug 30, 2023	-	Cover, footer and colophon: - Update Rev and issue date. Chapter 2. Reference documents: Update version of reference documents Chapter 3: Section 3.3.3 GPT Driver Component: - Add information of parameter “GPT_E_INTERRUPT_CONTROLLER_FAILURE” to table 3-28 Section 3.3.5.3 Folder Structure - Header file: Mcu_CLK_LLDriver.h update to “-“ for V4H - Source Files: Mcu_CLK_LLDriver.c update to “-“ for V4H Remove file Mcu_Irq.c, Mcu_Cfg.c Section 3.3.8 SPI Driver Component: - Add information of parameter “SPI_E_INTERRUPT_CONTROLLER_FAILURE”, “SPI_E_WRITE_VERIFY_FAILURE” to table 3-67 Section 3.3.9.2 Module Dependency: - Correct name of MCU clock from WDTBTCLKI to McuWDTClk. Chapter 5: Section 5.1 Products: - Update MCAL Product Release Version for R-CAR/V4H.
2.07	Jul 25, 2023	-	Chapter 5: Section 5.1 Products: Update MCAL Product Release Version for R-CAR/V4H.
2.06	Jul 06, 2023	-	Chapter 5: Section 5.1 Products: Update MCAL Product Release Version for R-CAR/S4 G4MH, R-CAR/S4 CR52, R-CAR/V4H.

2.05	May 22, 2023	-	Chapter 2. Reference documents: Update version of reference documents Chapter 3: Section 3.3.x.9 and 3.4.x.9, (2) Recommended Environment (All except FlexRay): Add information for “Switch setting on the Board” between the Evaluation Board and Host PC (windows) lines. Section 3.3 and 3.4: Add section “Addition Error Handling”, “Restrictions”, “ROM/RAM Usage”, “Stack Depth” and “Throughput Details” for all modules. Add new section “How to build the Sample Application” for ICU. Section 3.3.5.9 Sample Appilcation Update the sequence to execute API Mcu_InitRamSection. Section 3.3.13.9.1 Sample Application Structure: Update description to add device name tag. Section 3.7.3: Add 3.7.3.1 Precondition in 3.7.3 How to run Sample Application The switch setting is a prerequisite for the factory setting. The following chapters explain the differences from the factory settings. Chapter 5: Section 5.1 Products: Update MCAL Product Release Version for R-CAR/S4 G4MH, R-CAR/S4 CR52, R-CAR/V4H. Add section 5.4 RENESAS Configuration
2.04	May 05, 2023	-	Chapter 2. Reference documents: Update version of reference document [3] , [4]. Chapter 5: Section 5.1 Products: Update MCAL Product Release Version for R-CAR/V4H
2.03	Apr 04, 2023	-	Chapter 2. Reference documents: Update version of reference document [4], [5]. Chapter 3: Section 3.2: In Table 3 3 File Category Table: Correct file path of MCALConfGen.exe. Section 3.5.3: Remove (Not Supported) in section name and the section content, add "Note: Not supported by R-Car". Section 3.6: Remove (Not Supported) in section name and the section content, add "Note: Not supported by R-Car". Chapter 5: Section 5.1 Products: Update MCAL Product Release Version for R-CAR/V4H Section 5.2: Update option for ARM Compiler. Remove section 5.4 RENESAS Configuration (Not Supported). Chapter 6: Remove (Not Supported) from chapter title

2.02	Feb 23, 2023	-	Chapter 2. Reference documents: Update version of reference document [3], [4]. Chapter 3: Section 3.4.1.7: Correct Sample Application status variable to GaaTestResult Section 3.4.6.7: - Add function CddEmm_SupportControlExternalErrorRequest and CddEmm_SetHoldMaskCounter to Sample App Section 5.1 Products: Update V4H Hardware User Manuals to Rev.0.70 Update MCAL Product Release Version for R-CAR/V4H
2.01	Jan 30, 2023	-	Chapter 2. Reference documents: Update version of reference document [1], [3]. Chapter 3: Section 3.3.10.7: In (2) Recommended Environment add note for V4H Environment Chapter 5: Section 5.1 Products: Update V4H Hardware User Manuals to Rev.0.55 Update MCAL Product Release Version for R-CAR/V4H
2.00	Jan 12, 2023	-	Section 5.1 Products: Update S4 Hardware User Manuals to Rev.0.82 Chapter 2. Reference documents: Update version of reference document [1], [2], and [4]
1.15	Dec 20, 2022	-	Chapter 3: Figure 3-2: + Update rel/common Folder Structure for devices which use GHS compiler + Add rel/common Folder Structure for devices which use ARM compiler. Table 3-2: + Add <compiler> for devices R-Car/S4_G4MH, R-Car/S4_CR52, R-Car/V4H. + Row <msn>: add prefix for CDD modules (from “iccom”, “rfso”, “iic”, “ipmmu”, “ths”, “emm”, “crc” to "cddiccom", "cddrfso", "cddiic", "cddipmmu", "cddths", "cddemm", "cddcrc"). + Remove “(not supported)” information for module “fr” Table 3-5: Update location of Translation Header Files to "rel\<Device_Name>\common_family\generator\" Table 3-6: BSWMDT Files: add note for <Device_Name> Table 3-9: + Update location for files App_<MSN>_<Device_Name>_Sample.h and App_<MSN>_<Device_Name>_Sample.c. + Add information for file dummy_g4mh_case0.srec Table 3-67: + Remove parameter SpiOsCounterRef Table 3-90: + Add information about S4N devices supporting. Chapter 5: Section 5.1: - Update MCAL Product Release Version for R-CAR/S4 G4MH, R-CAR/S4 CR52, R-CAR/V4H. Section 5.2: Update version of ARM compiler to 6.16.1

1.14	Nov 18, 2022	-	Chapter 2: Table 2-1: Update version for all Reference Document. Chapter 3 Unify the format of Module dependency parts for all modules. Unify format the path and execute batch command of Sample Application for all modules. Add table name for S4 G4MH/CR52/V4H Environment table for all modules. Add table Table 3-1 Files that deviate from the naming conventions for ETH(S4_CR52) in section 3.2. Update content of sample app for GPT (S4_CR52/V4H) at section 3.3.3.7 Sample Application. Update content of sample app for ETH(S4_CR52) in 3.3.13.7 Sample Application. Chapter 5: Section 5.1: - Update MCAL Product Release Version for V4H Section 5.2: - Update version of compiler to 6.16.2
1.13	Oct 11, 2022	-	Chapter 2: Table 2-1: Update version for No.4 Chapter 3: 3.2 Folder Structure - Update file name for EthIf, EthSwt and EthTrcv in Table 3-9 and 3-10 Stub Files. 3.3.8.4 SPI: - Update Path for SpiClockFrequencyRef. 3.3.2.3 FLS: - Remove Fls_Debug.h in Table 3-14 FLS Header Files 3.3.13 Ethernet - Update S4_CR52 information in 3.3.13.1 Module Overview, Table 3-69 Ethernet Header Files, Table 3-70 Ethernet Source Files and Table 3-71 ETH Parameter Definition Files. - Update dependency files in 3.3.13.5 Source Code Dependency. - Remove 3.3.13.8 De-Initialization Function Sample 3.4.4 IPMMU: - Correct information in section Sample Application Structure. Chapter 5: 5.1 Product: Update MCAL Product Release Version for V4H
1.12	Sep 23, 2022	-	Chapter 2: Table 2-1: Update version for No.3 Chapter 3: 3.3.10 CAN: - Remove CanWakeupSourceRef parameter out of table 3-60 CAN Driver Component Configuration Parameter Dependency. at Section 3.3.10.4: Chapter 5: 5.1 Product: Update MCAL Product Release Version for V4H

1.11	Aug 26, 2022	-	Chapter 3: 3.3.2 FLS: - Add information for Fls_Suspend(), Fls_Resume(), Fls_DDRCalibrate() API to overview and sample app description. - Update information for Fls_SendSpecificConfig() for additional to supports the send specific configurations feature with Hyper Flash device (Currently, Fls_SendSpecificConfig() can support send specific configurations to Serial Flash or Hyper Flash). 3.3.5 MCU - Add ECM and ECC work products and description to Table 3-29 to Table 3-30 MCU header file and Table 3-31 MCU source files - Add ECM parameter to Table 3 33 shows the list of configuration parameter dependency for MCU Driver component. 3.3.10 CAN: - Section 3.3.10.3: Added new source files Can_TSCapture.c and Can_TSCapture.h - Section 3.3.10.3: fill applicable for V4H at source Can_RamTest.c, Can_Icom.c, and header Can_Icom.h to coresponding table. 3.3.13 ETH: - Section 3.3.13.4.Configuration Parameter Dependency: + Remove ETH_E_INT_INCONSISTENT, ETH_E_TIMERINC_FAILED, ETH_E_TIMEROFFSET_FAILED, EthInputClockRef parameters for only V4H device. 3.4.3 IIC: Update Table 3-83: Replace CDDIIC_E_UNINTENDED_INTERRUPT_CHECK by CDDIIC_E_INTERRUPT_CONTROLLER_FAILURE. 3.4.4 IPMMU: Update Table 3-87: Replace CDDIPMMU_E_UNINTENDED_INTERRUPT_ FAILURE with CDDIPMMU_E_INTERRUPT_CONTROLLER_FAILURE. 3.5.4 Hypervisor Mode: - Added section 3.5.4 Hypervisor Mode 3.7.3 How to run Sample Application Update whole of section to add how to run MCAL sample application for both RCar S4 and RCar V4H devices. Chapter 5: 5.1 Product: -Table 5-1: Update Device Manual Version for S4(CR52/G4MH) and V4H.

1.10	Aug 22, 2022	-	Chapter 3: 3.3.2 FLS: - Add information for Fls_SendSpecificConfig(),Fls_DDRWritePattern() and Fls_DDRVerifyPattern() API to overview and sample app description. 3.3.5 MCU: - Section 3.3.5.3 Folder Structure: Update structure for files Mcu_VMN_LLDriver.h Mcu_VMN_LLDriver.c, Mcu_CLM_LLDriver.h, Mcu_STB_LLDriver.h, Mcu_CLM_LLDriver.c, Mcu_STB_LLDriver.c. - Section 3.3.5.4 Configuration Parameter Dependency: Add parameters MCU_E_VMON_DIAG_FAILURE and MCU_E_DMON_DIAG_FAILURE. 3.3.6 PORT: - Section 3.3.6.4 Configuration Parameter Dependency: + Add the parameter PORT_E_UNINTENDED_MODULE_STOP_FAILURE - Section 3.3.6.7 Sample Application: + Add the content of Port_UnintendedModuleStopCheck 3.3.8 SPI - Section 3.3.8.3 Folder Structure: Add file Spi_SYSDMAC_Irq.c, Spi_SYSDMAC_LLDriver.c, Spi_SYSDMAC_Irq.h, Spi_SYSDMAC_LLDriver.h 3.3.10 CAN: - Add src file Can_RamTest.c for Can Module at Table 3-57 CAN Source Files - Add new parameter “CAN_E_INTERRUPT_CONTROLLER_FAILURE” and remove “CAN_E_INT_INCONSISTENT” parameter in Table 3-60 “CAN Driver Component Configuration Parameter Dependency” - Add content invocation Can_SelfTestChannel API in sequence of Sample Application Structure at section 3.3.10.7 Sample Application 3.3.13.4 ETH: - Add new parameter “ETH_E_INTERRUPT_CONTROLLER_FAILURE” parameter for V4H in table “ETH Driver Component Configuration Parameter Dependency” 3.4.1 ICCOM Driver Component: - Section 3.4.1.7.1: Use System Up-Time Clock (SUCMT) replace TMU0 (Timer Unit 0) 3.4.4 IPMMU Driver Component: - Section 3.4.4.4: In “Table 3-87”, Change CDDIPMMU_E_WRITE_VERIFY to CDDIPMMU_E_WRITE_VERIFY_FAILURE. - Section 3.4.4.4: In “Table 3-87”, Change CDDIPMMU_E_UNINTENDED_INTERRUPT_CHECK to CDDIPMMU_E_UNINTENDED_INTERRUPT_FAILURE. - Section 3.4.4.7: Correct the guide to execute IPMMU Sample Application in (5) How to run IPMMU sample application 3.4.7 CRC - Section 3.4.7.2: Remove Port from Module Dependency. - Section 3.4.7.4: Add CDDCRC_E_INTERRUPT_CONTROLLER_FAILURE, CDDCRC_E_UNINTENDED_MODULE_STOP_FAILURE, CDDCRC_E_HARDWARE_ERROR - section 3.4.7.7: Update Sample Application Structure Common Update save address in eMMC memory in step6 of section '3.7.3. Flash bootloaders' 3.3.1 DIO

			- Section 3.3.1.7.1: Remove “configured” of Dio_WriteChannel API, Dio_WriteChannelGroup API, and Dio_WritePort API in Sample Application Structure.
1.09	Jul 27, 2022	-	Chapter 2: Update Reference Documents version. Chapter 3: 3.3.6 PORT: - Section 3.3.6.7 Sample Application : + Add the content of Port_RefreshPortDirection, Port_SetPinMode and Port_FUSEMonitoring in. - Section 3.3.6.4 Configuration Parameter Dependecy: + Add PORT_E_FUSE_MONITORING_FAILURE in Table 3-37. 3.3.13 ETH: - Added note to describe the configuration of PHY mode at section 3.3.13.7 (5) How to run ETH Sample app. 3.4.2 RFSO: - Section 3.4.2.3: In "Table 3.75 and Table 3.74", remove the bold format of word "and". - Section 3.4.2.5: Add file “Rte_CddRfso.h” into list of stub files - Section 3.4.2.7: Update content of "Periodical Check" and "Runtime test" to correspond with Sample Application. 3.4.5 THS: - Section 3.4.5.1: Remove "Not support" keyword for below feature: + Enable/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 + Clear temperature error status - Section 3.4.5.3: Table 3-85: + Remove CDD_Ths_Debug.h + Update structure for file CDD_Ths_THS_LLDriver.h Table 3-86: + Update structure for file CDD_Ths_THS_LLDriver.c 3.4.6 EMM: Section 3.4.6.4 - Table 3-92: + Change name of parameter CDDEMM_E_REG_WRITE_VERIFY by CDDEMM_E_WRITE_VERIFY_FAILURE + Change name of parameter CDDEMM_E_UNINTENDED_INTERRUPT_CHECK by CDDEMM_E_INTERRUPT_CONTROLLER_FAILURE 3.7 Sample Application: Section 3.7.2.2: Update MCAL Sample Application build guideline at (2)

1.08	Jun 16, 2022	-	Chapter 3: 3.3.1 DIO: - Section 3.3.1.2: Add DEM - Section 3.2.9 Apply file Dem_Cfg.h and Dem.h in Table 3-9 3.3.2 FLS: - Update information in section 3.3.2.1 and section 3.3.2.7 to add validating the flash memory's content information. - Add Note in section in 3.3.2.1. - Apply file Dem_Cfg.h and Dem.h in Table 3-9 in section 3.2.9. 3.4.2 RFSO: - Section 3.2.9 Apply file Dem_Cfg.h in Table 3-9 3.4.4 IPMMU: - Update information in section 3.4.4.4 3.4.5 THS: - Table 3-85: Remove Cdd_Ths_Cfg.h and Cdd_Ths_Reg.h due to they are not header file 3.7.3.2 Flash bootloaders: - Information list of release image when ICUMX IPL loads CX 2nd IPL (QSPI Flash): Program Top address in cert_header_sa9.srec changed to H'EB230000
1.07	May 27, 2022	-	Chapter 3 Section 3.2.9: Add new stub Fee. 3.3.2 FLS: - Section 3.3.2.2: Add Fee to module dependency. - Section 3.3.2.5: Add Fee.h to source code dependency. 3.3.9: - Add GPT as dependency module in section 3.3.9.2. 3.4.1 ICCOM: - Remove section (6) How to change the MCAL ICCOM sample application to run in combination with the CA76 ICCOM sample application for V4H. 3.4.6 EMM: - Section 3.4.6.7: Update Sample Appication sprint log. Section 3.4.7 CRC Driver Component: - Add source files and header files of low-level driver.

1.06	Apr 25, 2022	-	Chapter 3: - Update “(2) Preparation” in Sample Application section for all modules - Update How to run Sample Application for all modules - Secrion 3.3.1 DIO Driver Component: Remove “Not Supported” for descriptions related to new supported features of DIO in Sample Application Structure for V4H. - Secrion 3.3.8 SPI Driver Component: Update HW connection setting - Secrion 3.3.13 Ethernet Driver Component: Add note about building a sample app for the S4_CR52 Ethernet Driver Module. - Section 3.4.1 V4H ICCOM: + Update section Preparation for MCAL ICCOM Sample Application + Add section (6) How to change the MCAL ICCOM sample application to run in combination with the CA76 ICCOM sample application + Add section (7) How to change the serial console setting of MCAL ICCOM sample application + Add section 3.7.3.4 How to load MCAL Sample Application binary file on the evaluation board (MCAL ICCOM Sample Application only) - Secrion 3.4.5 THS Driver Component: Remove “Not Supported” for descriptions related to new supported features in Module Overview. - Add section 3.7.3 How to run V4H Sample Application Chapter 5: - Change MCAL Product Release Version of RCAR/V4H to Ver19.0.8 - Add Supported OS in the Table 5-1 Product Overview
1.05	Mar 28, 2022	119-120	Chapter 3: Section 3.3.10: Add S4_CR52 support “Can_Icom.h”, “Can_Icom.c”
		119-120	Add source "Can_Icom.c" and header "Can_Icom.h" to CAN structure S4
		-	Chapter 3: Section 3.4.7: Add CRC Driver Component to support V4H
		114-119	Chapter 3: Section 3.3.9: Remove Vendor ID (59) for all APIs and functions of S4 WDG and add support source file for WDG V4H.
		-	Chapter 3: Section 3.4.6: Add EMM Driver Component to support V4H
		-	Chapter 5: Apply new template: Update Sample Application sections for all Modules
		-	Chapter 3: Section 3.4.5: Add THS Driver Component to support V4H
		-	Chapter 3: Section 3.4.4: Add IPMMU Driver Component to support V4H
1.04	Jan 28, 2022	—	Chapter 5: Update Sample Application section for all Modules V4H Pre-Alpha2.
		116-118	Chapter 3: Section 3.3.10: Update CAN Driver Component to support V4H
		159-163	Chapter 3: Section 3.4.2: Add RFSO Driver Component to support V4H
		105-116	Chapter 3: Section 3.3.8: Update SPI Driver Component to support V4H

1.03	Jan 14, 2022	168	Chapter 5: Update ARM Compiler support to Version 6.16.1
1.02	Dec 17, 2021	—	Chapter 3: Add Sample Application section for all Modules
1.01	Oct 29, 2021	34-36	Chapter 3: Section 3.3.2: Update FLS Driver component to support V4H
	Nov 12, 2021	31-32	Chapter 3: Section 3.3.1: Update DIO Driver Component to support V4H
	Nov 22, 2021	63-67	Chapter 3: Section 3.3.6: Update PORT Driver Component to support V4H
	Nov 23, 2021	111-114	Chapter 3: Section 3.4.1: Update ICCOM Driver Component to support V4H
	Nov 25, 2021	106-109	Chapter 3: Section 3.3.13: Update ETH Driver Component to support V4H
1.00	Oct 08, 2021	—	First Edition issued

R-Car Gen4 AUTOSAR R19-11 MCAL User's Manual
Modules Overview

Publication Date: Rev.4.00 Mar 27, 2025

Published by: Renesas Electronics Corporation

R-Car Gen4 AUTOSAR R19-11 MCAL
User's Manual

Renesas Electronics Corporation