R-Car S4 Series
Guide to Inter-Core Communications with the Use of a G4MH Core of an R-Car S4

Introduction

In addition to the Cortex-A55 application cores and the Cortex-R52 realtime core, an R-Car S4 device also has two G4MH cores from the range of RH850 microcontrollers. The G4MH cores are intended for general vehicle control. Moreover, the R-Car S4 has a multifunctional interface (MFIS) controller as hardware to provide support for inter-core data communications. This application note describes how one G4MH core receives data from a CAN port and transfers them to one of the application cores, after which the application core externally outputs the information.

Target Board

R-car S4 evaluation board RTP8A779F0ASKB0SC1S or RTP8A779F0ASKB0SB0S
We simply refer to it as "evaluation board" in this document.
Contents

1. Overview ......................................................................................................................... 4
   1.1 Purpose ......................................................................................................................... 4

2. Contents of the Sample Program Package ................................................................. 5

3. Environment for Confirming Operation ........................................................................ 6

4. Overview of the Software ............................................................................................. 8

5. Configuration of Software ............................................................................................ 9
   5.1 Startup Module (for the G4MH) ................................................................................. 9
   5.2 MCAL CAN Driver (for the G4MH) .......................................................................... 9
   5.3 MCAL ICCOM Driver (for the G4MH) ...................................................................... 10
   5.4 Application Software (for the G4MH) ...................................................................... 10
   5.5 ICCOM Driver (for Linux on CA55) ......................................................................... 10
   5.6 libwebsockets (for Linux on CA55) .......................................................................... 10
   5.7 Application Software (for Linux on CA55) ................................................................. 10

6. Details of the Software ................................................................................................. 11
   6.1 Inter-Core Communications ..................................................................................... 11
   6.2 Processing by the G4MH Core ................................................................................. 13
       6.2.1 Startup Processing ........................................................................................... 13
       6.2.1.1 Initializing the Registers .............................................................................. 13
       6.2.1.2 Judging the Core Number .......................................................................... 13
       6.2.1.3 Setting the Reset Vector .......................................................... 13
       6.2.1.4 Initializing the RAM Areas ........................................................................ 13
       6.2.1.5 Enabling FPUs .......................................................................................... 14
       6.2.1.6 To the Main Function ................................................................................. 14
       6.2.2 Initializing the MCAL CAN Driver ................................................................. 14
           6.2.2.1 Setting Input of the Clock Signal to the CAN Module ......................... 14
           6.2.2.2 Releasing the CAN Module from Standby ....................................... 14
           6.2.2.3 Setting the Interrupt Method ............................................................... 14
           6.2.2.4 Setting the Pin Functions ...................................................................... 14
           6.2.2.5 Initializing the CAN port ................................................................. 15
           6.2.2.6 Setting the CAN Bit Rate ..................................................................... 16
           6.2.2.7 Setting the CAN Controller ................................................................. 16
           6.2.3 Initializing the MCAL ICCOM Driver ...................................................... 17
               6.2.3.1 Enabling the Bus ............................................................................... 17
               6.2.3.2 Releasing Protection of the MFIS Control Registers ..................... 17
               6.2.3.3 Enabling the Interrupt ...................................................................... 18
               6.2.3.4 Initializing the MCAL ICCOM Driver ......................................... 18
               6.2.4 Processing of the G4MH Application ................................................. 19
       6.3 Processing by the CA55 Core ................................................................................. 21
       6.3.1 Initializing the ICCOM Driver .......................................................................... 21
       6.3.2 Activating the Web Server Thread ................................................................. 22
The following registered trademarks and trademarks of third parties, as well as company names, system names, and product names mentioned in this document are generally registered trademarks or trademarks of their respective companies.™ mark and ® mark for companies trademark or registered trademark are omitted in this document.

- Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. (https://www.arm.com/ja/company/policies/trademarks)
1. **Overview**

1.1 **Purpose**

R-Car S4 devices support high performance and various high-speed networks which are required for the evolution of electrical/electronic (E/E) architectures into domain-oriented and zone-oriented E/E architectures, thus enabling the employment of a car server/communication gateway (CoGW) with strong security functionality and high-level functional safety. The R-Car S4 enables the reuse of up to 88% of software developed for third-generation R-Car series products and RH850 microcontrollers.

Figure 1  Schematic View of Reuse of Software for the R-Car S4 and Integration of the Chips

This application note supports the effective utilization of assets from the RH850 microcontrollers and the development of a car server by giving examples where a G4MH core from the RH850 range receives CAN data and transfers them to an application core through inter-core communications, after which the application core outputs the data to an external display via an Ethernet interface. The following figure is an overview of the sample system.

Figure 2  Overview of the Sample System
2. Contents of the Sample Program Package

This section describes the contents of the sample program used in this application note. Download it from the following page on the Renesas official site.

Renesas Official Web site

The following shows the software package required for the software configuration described in section 4, Overview of the Software. It includes the modules framed by red rectangles in the following figure.

---

<table>
<thead>
<tr>
<th>Target</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G4MH</td>
<td>Application software and MCAL patch</td>
<td>Application software, MCAL generator file and patch for the CC-RH compiler</td>
</tr>
<tr>
<td></td>
<td>Startup, CS+ project file</td>
<td>Code for starting up the G4MH, CS+ project file</td>
</tr>
<tr>
<td>CA55</td>
<td>Application software</td>
<td>Application software and file for controlling the Web display</td>
</tr>
<tr>
<td>PC (Ubuntu Linux)</td>
<td>Application software for the CARLA simulator</td>
<td>Application software for CAN output</td>
</tr>
</tbody>
</table>

The folders of this package are structured as follows.

---

Figure 3 Sample Program Package for Data Transfer with the Use of a G4MH Core of an R-Car S4

Figure 4 Structure of Folders in the Package
3. Environment for Confirming Operation

The following lists the items of the operating environment in which operation of the sample program was confirmed.

<table>
<thead>
<tr>
<th>Table 2 Operating Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
</tr>
<tr>
<td>Evaluation board</td>
</tr>
<tr>
<td>Boot loader</td>
</tr>
<tr>
<td>CAN USB cable</td>
</tr>
<tr>
<td>D-sub 9-pin cable</td>
</tr>
<tr>
<td>G4MH development environment</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CA55 development environment</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Operating and development</td>
</tr>
<tr>
<td>environment for the CARLA</td>
</tr>
<tr>
<td>simulator(^2)</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The figure below shows the connections in the environment where operation was confirmed.

---

\(^1\) Download the CS+ device custom file for the G4MH from the R-Car Market Place (see footnote エラー！ブックマークが定義されていません。).

\(^2\) CARLA is an open-source simulator for advanced driver-assistance systems (ADAS) and autonomous vehicle applications. For details, see [https://carla.org](https://carla.org). For details on the recommended environment, see section 7.5, Building the CARLA Development Environment.
The figures below show normal settings of SW1, SW2 and switches on the switch board when booting up is to be from flash ROM instead of with use of the E2 emulator.

![Figure 6 Settings of SW1 and SW2](image1.png)

![Figure 7 Switch Board Settings](image2.png)

The following lists the settings of the serial port for the USB serial cable (for use with the console).

<table>
<thead>
<tr>
<th>Item</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit rate</td>
<td>1,843,200 bps</td>
</tr>
<tr>
<td>Data</td>
<td>8 bits</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
</tr>
<tr>
<td>Stop bit</td>
<td>1 bit</td>
</tr>
<tr>
<td>Flow control</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 3  Serial Port Settings
4. Overview of the Software

The sample program used in this application note consists of application software for use with the two types of core on which they are intended to run, a G4MH and CA55, and application software for the output of CAN data, which runs on the CARLA simulator. The G4MH core only receives the required CAN data from among those output from the CARLA simulator after internally filtering by the CAN module of the R-Car S4. The G4MH core transfers the received CAN data to the CA55 core through the MFIS module, which supports inter-core communications. The CA55 core generates the window control information based on the received CAN data and updates the display in the Web browser running on the external PC. Repeatedly cycling through these steps allows operation by the CARLA simulator to update the browser display in real time. The following figure gives an overview of the operations of the applications.

Figure 8  Overview of Application Operations
5. Configuration of Software

The G4MH software consists of the application software, MCAL CAN driver, MCAL ICCOM driver, and startup module. The CA55 software consists of the application software, ICCOM driver, libwebsockets, and Linux OS. The figure below illustrates the configuration of the software.

![Configuration of Software](image)

### 5.1 Startup Module (for the G4MH)

After turning on the power supply to the R-Car S4, the G4MH cores are activated before the CA55 core. The startup module begins operations from program counter address 0 and proceeds with initialization of registers, setting of an exception vector table, clearing of the RAM area that doesn’t have initial values, and setting up the RAM area that does have initial values. After that, the module runs the main function of the application software. Note that the sample program described in this application note only uses core 0 of the two G4MH cores in the R-Car S4 and simply places core 1 in an endless loop after initialization of the registers.

### 5.2 MCAL CAN Driver (for the G4MH)

MCAL is provided as part of the R-Car S4 SDK from Renesas. MCAL is software for a layer which directly controls functions in the MCU among the software layers stipulated in AUTOSAR, a set of standards to specify two kinds of platform for in-vehicle software. Though MCAL is originally part of the basic software (BSW) layer of AUTOSAR, it can be used for independent applications, which enables the building of software that is not MCU-dependent. MCAL includes various types of software that are applicable to the peripherals in the R-Car S4, one of which is the MCAL CAN driver. Note that the base version of the MCAL software is for compilers from Green Hills Software, but the version in the sample program for use with this application note has been modified to be usable with the CC-RH compiler.
5.3 MCAL ICCOM Driver (for the G4MH)

The MCAL ICCOM driver is also part of the MCAL software in the same way as the CAN driver stated in section 5.2, MCAL CAN Driver (for the G4MH), and provides the API functions for inter-core communications. Use of the MCAL ICCOM driver enables the implementation of inter-core communications without the need to take the register configuration of the MFIS module into account.

5.4 Application Software (for the G4MH)

The G4MH application software initializes the MCAL CAN and ICCOM drivers without using an OS and then repeats the reception of CAN data and their transfer to the CA55 core. Filtering the CAN data by using ID values is possible. This is described in section 6, Details of the Software.

5.5 ICCOM Driver (for Linux on CA55)

The ICCOM driver for the CA55 is provided as part of the R-Car S4 SDK from Renesas and provides the API functions for inter-core communications. This driver is capable of communications with the Cortex-R52 core (CR52) in the R-Car S4 in addition to the G4MH cores. Note that the driver only handles communications between one CA55 core and one G4MH core in the sample program used with this application note. Use of the ICCOM driver enables the implementation of inter-core communications without the need to take the register configuration of the MFIS module into account in the same way as the MCAL ICCOM driver.

5.6 libwebsockets (for Linux on CA55)

libwebsockets is a C library for building a Web server. Use of libwebsockets enables the generation of flexible Web application software in accord with the WebSocket protocol.

5.7 Application Software (for Linux on CA55)

The CA55 application software initializes the ICCOM driver and libwebsockets and then behaves as a Web server. In the reception of CAN data transferred from the G4MH core, the ICCOM driver calls a callback function that was registered at the time of initialization of the ICCOM driver. The callback function analyzes the received CAN data, generates commands to control the Web browser display, and then requests a WebSocket transmission to a Web server thread. On reception of a request to transmit a command, the Web server thread transmits the command to the Web browser through the WebSocket communications path connected to the Web browser.
6. Details of the Software

This section describes the details of how communications between the two cores proceed and then describes processing for the initialization and applications of each core.

6.1 Inter-Core Communications

The MFIS module supports inter-core communications for the eight Cortex-A55 AP-system cores, one Cortex-R52 realtime core, and two G4MH control domain cores in the R-Car S4. Note that it does not cover communications between cores of the same type, for example, transfer between G4MH cores 0 and 1. An ICCOM driver is control software for the MFIS module and is provided for each type of core. Use of these drivers by the application software enables inter-core communications without the need to take the hardware into account. How to proceed with inter-core communications is described in terms of the behavior of the sample program below and on the following page.

Description:

<1> The G4MH core uses the Cddlccom_Ch1SendRun function to transmit CAN data specified as the argument to the CA55 core. Note that “Ch1” in the function name indicates that a G4MH core is to be the source of the transmission. The Cddlccom_Ch0SendRun function is the transmission function for use with the CR52 core. The MCAL driver is designed for the application of common driver code in this way.
<2> When the CddIccom_Ch1SendRun function of the MCAL ICCOM driver is called, the G4MH core stores the data in the shared memory area (CTA: common transmission area) to which cores of all three types have access and the data length in a general-purpose register for inter-core communications in the MFIS module, respectively. After that, the G4H core asserts an interrupt signal for the CA55 core.

<3> Though the set of MFIS registers in the figure above only consists of four registers, the MFIS module actually has the same number of register sets as the number of combinations of cores.

Number of MFIS register sets = CA55 (8) x CR52 (1) + CA55 (8) x G4MH (2) + CR52(1) x G4MH (2) = 26

Specifically, with the sample software, the ICCOM driver only uses the register set for the combination of G4MH core 0 and CA55 core 0 from among the sets. Since the CTA size is specified as 2 Kbytes x 2 (for a double buffer) per register set in the ICCOM driver, the amount of data in a single transfer is up to 2 Kbytes. The table below describes the interrupt registers in detail.

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:16]</td>
<td>Reserved. These bits are always read as 0. When writing, always write 0 to these bits.</td>
</tr>
<tr>
<td>[15:1]</td>
<td>Freely writing to and reading from these bits by software is possible. This does not affect the destination core for interrupt. The ICCOM driver uses these bits to convey the type of interrupt (data or acknowledgement), and other information. For details, see “Control value” in CDD_Iccom_PBTypes.h.</td>
</tr>
<tr>
<td>[0]</td>
<td>Interrupt request bit. Writing 1 to this bit asserts the interrupt signal for the other-party core.</td>
</tr>
</tbody>
</table>

<4> Writing to the interrupt register for an interrupt from the G4MH core to the CA55 core asserts an interrupt signal for the CA55 core. The interrupt function of the CA55 core clears the interrupt register and reads the data length. After that, the interrupt function calls the callback function specified by the initialization function with the data length and address in the CTA area specified as the arguments. When the processing of the callback function is completed, the CA55 core writes to the interrupt register for an interrupt to the G4MH core to notify the G4MH core that transmission has been completed.

<5> The callback function of the Linux application receives the CAN data passed as the argument, stores them in a temporary buffer, and returns at once. Note that this callback function is handled within the interrupt processing.

<6> On receiving the interrupt from the MFIS module, the G4MH core judges the interrupt type by reading the value of the interrupt register. If the interrupt is a notification of the completion of transmission, the G4MH core modifies the transmission-completed flag to “true”. The application program refers to the transmission-completed flag to detect the completion of transmission to the CA55 core.
6.2 Processing by the G4MH Core

This subsection gives an overview of the processing by the G4MH sample program.

![Diagram of G4MH sample program]

6.2.1 Startup Processing

After turning on the power supply to the R-Car S4, the IMX boot loader transfers the code for the G4MH cores to RAM then activates the cores. The program is started from address 0 to proceed with the startup processing. The startup processing is written in assembler. Detailed descriptions of the startup processing in the order of its handling are given below.

6.2.1.1 Initializing the Registers

The startup module clears general-purpose registers r1 to r31 to 0.

6.2.1.2 Judging the Core Number

The startup module reads the HTCFG0 system register to obtain its own core number. For core 0, the startup module proceeds with the processing at the next reset vector setting. The startup module places core 1 in an endless loop.

6.2.1.3 Setting the Reset Vector

The startup module sets the EBV bit of the PSW register to 1 to select the reset vector method. The RBASE register is fixed to 400h by hardware. The startup module sets the EBASE register to 0.

6.2.1.4 Initializing the RAM Areas

The startup module sets up the RAM area that has initial values [.data] and clears the RAM area that does not have initial values [.bss]. This sample program uses the _INITSCT_RH function within CS+ to initialize the RAM areas.
6.2.1.5 Enabling FPUs

Though this sample program does not use FPUs, if any were to be used, the startup module would make the setting to enable them.

6.2.1.6 To the Main Function

Processing makes the transition from the startup module to the main function of the application program.

6.2.2 Initializing the MCAL CAN Driver

Initialization of the MCAL driver is described in the order of its handling below.

6.2.2.1 Setting Input of the Clock Signal to the CAN Module

The G4MH core sets clock input to the CAN module.
For details, see the Can_Clock_Init function in App_CAN_S4_Sample.c.

6.2.2.2 Releasing the CAN Module from Standby

The G4MH core releases the CAN-FD and OS timer modules from standby.

6.2.2.3 Setting the Interrupt Method

From the direct and vector methods, the vector method is selected as the interrupt method for the G4MH core and the interrupt priority level is set to 7, a medium level.

6.2.2.4 Setting the Pin Functions

The G4MH core makes the settings to connect the control signal of CAN port 0 to an external pin. The processing required before calling an API function of the MCAL CAN driver is completed at this point. After that, the G4MH core uses an API function of the MCAL CAN driver to initialize the CAN port.
6.2.2.5 Initializing the CAN port

From among the MCAL CAN driver API functions, the G4MH core calls the Can_Init function to initialize the driver. The Can_Init function uses a static setting for the CAN port generated by the MCAL generator. Since the CAN data used in this sample program cannot be received due to filtering with the default setting, change Can_PBcfg.c to receive the data corresponding to the intended CAN ID values. The Can_PBcfg.c file is generated by the MCAL generator. Though the automatically generated MCAL source code should originally not been changed, we dare to change Can_PBcfg.c to run the sample program. The tables below list the CAN data to be received and the values to be set to Can_PBcfg.c.

<table>
<thead>
<tr>
<th>No.</th>
<th>CAN ID Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xB4 (180)</td>
<td>Speed information (km/h)</td>
</tr>
<tr>
<td>2</td>
<td>0x25 (37)</td>
<td>Steering wheel angle (rad)</td>
</tr>
<tr>
<td>3</td>
<td>0x127 (295)</td>
<td>Gear position</td>
</tr>
</tbody>
</table>

Note that the CAN data to be received are converted according to the statements in the attached sample DBC file.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Corresponding RS-CAN FD Register Name</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usNoOfFilters</td>
<td>uint16</td>
<td>—</td>
<td>3</td>
<td>Total number of filters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Since this variable is only for use as a loop counter, it does not have a corresponding register.</td>
</tr>
<tr>
<td>Can_GaaGAFLCFGConfig0[0]</td>
<td>uint32</td>
<td>CFDGAFLCFGw</td>
<td>3</td>
<td>Number of filters for channel 0</td>
</tr>
<tr>
<td>Can_GaaFilterConfig0[0]. ulGAFLID</td>
<td>uint32</td>
<td>CFDGAFLIDr</td>
<td>180</td>
<td>CAN ID value for the speed information</td>
</tr>
<tr>
<td>Can_GaaFilterConfig0[0]. ulGAFLM</td>
<td>uint32</td>
<td>CFDGAFLMr</td>
<td>0x1fffffff</td>
<td>Bit mask for use in comparison of the CAN ID value for the speed information (all-bit comparison)</td>
</tr>
<tr>
<td>Can_GaaFilterConfig0[1]. ulGAFLID</td>
<td>uint32</td>
<td>CFDGAFLIDr</td>
<td>37</td>
<td>CAN ID value for the steering wheel angle information</td>
</tr>
<tr>
<td>Can_GaaFilterConfig0[1]. ulGAFLM</td>
<td>uint32</td>
<td>CFDGAFLMr</td>
<td>0x1fffffff</td>
<td>Bit mask for use in comparison of the CAN ID value for the steering wheel angle information (all-bit comparison)</td>
</tr>
<tr>
<td>Can_GaaFilterConfig0[2]. ulGAFLID</td>
<td>uint32</td>
<td>CFDGAFLIDr</td>
<td>295</td>
<td>CAN ID value for the gear position information</td>
</tr>
<tr>
<td>Can_GaaFilterConfig0[2]. ulGAFLM</td>
<td>uint32</td>
<td>CFDGAFLMr</td>
<td>0x1fffffff</td>
<td>Bit mask for use in comparison of the CAN ID value for the gear position information (all-bit comparison)</td>
</tr>
</tbody>
</table>

For details, see the Can_PBcfg.c file.

3 For details, see https://docs.fileformat.com/database/dbc.
Setting the values in the table on the previous page enables reception of the CAN data related to the vehicle speed, steering wheel angle, and gear position. Reception of any other CAN data is denied through filtering by the RS-CAN FD.

6.2.2.6 Setting the CAN Bit Rate

From among the MCAL CAN driver API functions, the G4MH core calls the Can_SetBaudrate function to set the CAN port to 250 Kbps. The bit rate is set to a fixed value by the MCAL driver. Changing the rate requires changes to the setting and code output by the MCAL generator.

6.2.2.7 Setting the CAN Controller

From among the MCAL CAN driver API functions, the G4MH core calls the Can_SetControllerMode function to enable the CAN port. The CAN port becomes ready for transmission or reception at this time.
6.2.3 Initializing the MCAL ICCOM Driver

The procedure for inter-core communications is described in section 6.1, Inter-Core Communications. This subsection describes the procedure for initialization of the inter-core communications driver (MCAL ICCOM driver) by the G4MH core. Before calling the initialization function for the MCAL ICCOM driver, the G4MH core proceeds with the processing for enabling the bus, releasing protection of the MFIS control registers, and enabling the interrupt from the CA55.

6.2.3.1 Enabling the Bus

Since the default setting is for the bus between the CA55 and G4MH cores to be disabled, communications between them are initially not possible. The G4MH core thus enables the bus.

```c
#define STBY_CTRL (*(volatile uint32 *)(0xF8F18234))
STBY_CTRL = 1;
```

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:1]</td>
<td>Reserved</td>
</tr>
<tr>
<td>[0]</td>
<td>0: Enables bus stop requests. 1: Disables bus stop requests.</td>
</tr>
</tbody>
</table>

For details, see section 51.3.5, MCCR Standby Control Register in document 1 under References in this application note.

6.2.3.2 Releasing Protection of the MFIS Control Registers

Since writing to the MFIS registers is prohibited with the default setting, the G4MH core releases this protection.

```c
#define MFISWPCNTR (*((volatile uint32 *)(0xD6260900)))
MFISWPCNTR = 0xACCE0001;
```

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:16]</td>
<td>Key code 0xACCE</td>
</tr>
<tr>
<td>[15:1]</td>
<td>Reserved</td>
</tr>
<tr>
<td>[0]</td>
<td>0: Applies protection. 1: Releases protection.</td>
</tr>
</tbody>
</table>
6.2.3.3 Enabling the Interrupt

The G4MH core enables the interrupt from CA55 core 0 to G4MH core 0.

```c
#define INTC2_PE0_BASE 0xFFF80000UL
#define INTC2_PE0(ID) (*((volatile uint16*)(INTC2_PE0_BASE+(uint32)(2U*ID)))

irqno = 80;        // Number of the interrupt from CA55 core 0 to G4MH core 0
INTC2_PE0(irqno) = 0x0045;
```

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15]</td>
<td>Read-only</td>
</tr>
<tr>
<td>[15]</td>
<td>Interrupt detection: 0: Edge-sensed detection; 1: Level-sensed detection</td>
</tr>
<tr>
<td>[7]</td>
<td>Interrupt mask: 0: Interrupt processing is enabled. 1: Interrupt processing is disabled.</td>
</tr>
<tr>
<td>[6]</td>
<td>Interrupt vector method: 0: Direct vector method; 1: Table reference method</td>
</tr>
<tr>
<td>[5]</td>
<td>Interrupt overflow setting: 0: No interrupt overflow; 1: Dependent on the interrupt detection method: (For details, see Table 6.19 in section 6 of 158, Control Domain Parts, in document 1 under References in this application note.)</td>
</tr>
<tr>
<td>[3:0]</td>
<td>Interrupt priority level setting from 0 to 15: 0 = Highest priority</td>
</tr>
</tbody>
</table>

6.2.3.4 Initializing the MCAL ICCOM Driver

The G4MH core initializes the MCAL ICCOM driver after the processing described from section 6.2.3.1, Enabling the Bus, to section 6.2.3.3, Enabling the Interrupt.

```c
CddIccom_Init(Pointer to the CddIccom_ConfigType structure data);
while(CDDICCOM_CH_READY != CddIccom_GpChannelStatus[1].enChSta) {
    /* Do nothing */
}
```

The CddIccom_ConfigType structure data are global structure data in which static parameters are set and generated by the MCAL generator. This sample program uses the CddIccom_ConfigType structure data generated by the MCAL generator provided as part of the MCAL SDK. After the initialization processing has been completed, the G4MH core is placed in the wait state until the

```
CddIccom_GpChannelStatus[1].enChSta global variable has become CDDICCOM_CH_READY. The array number “1” in CddIccom_GpChannelStatus[1] indicates that the core which is currently operating is the G4MH. Similarly, the array number “0” indicates the CR52. Note that although array numbers are used to distinguish between the G4MH and CR52 cores in the MCAL sample program, this is not always required. After the CA55 Linux application has initialized the ICCOM driver, inter-core communications are established, the above-mentioned variable becomes CDDICCOM_CH_READY, and processing by the G4MH leaves the loop shown in the code above. The processing for initializing the MCAL ICCOM driver is completed at this point.
6.2.4 Processing of the G4MH Application

After the initialization processing has been completed, processing makes the transition to the G4MH application. In the application processing of this sample program, the G4MH core that handles it repeatedly monitors reception through the CAN port. On detecting the reception of CAN data, the G4MH core reads and transmits the received data to the CA55 core. The listing below shows the processing by the sample program.

```c
struct candata_t {
    uint32_t id;
    uint8_t data[8];
};
struct canctl_t {
    struct candata_t d;
    uint8_t valid;
};
static struct canctl_t g_candata;

main(void) {
    g_candata.valid = FALSE;
    while(1) {
        Can_MainFunction_Read();
        if (g_candata.valid == TRUE) {
            g_candata.valid = FALSE;
            Send2Linux(&g_candata);
        }
    }
}

uint8 UserCalloutFunction(uint16 Hrh, Can_IdType CanId, uint8 CanDataLength, uint8 *CanSduPtr)
{
    g_candata.d.id = (uint32_t)CanId;
    memcpy(g_candata.d.data, CanSduPtr, sizeof(g_candata.d.data));
    g_candata.valid = TRUE;
    return TRUE;
}
```

The `Can_MainFunction_Read` function is an MCAL API function and checks whether CAN data have or have not been received. After detecting received data, the function calls the `UserCalloutFunction` callback function with the received CAN data as the argument. If no data have been received, return from the function proceeds at once.

The function transmits the CAN data specified as the argument to the CA55 core (described later).

The name of the callback function is specified in the header file (Can_Cfg.h) generated by the MCAL generator. Therefore, use the specified name to implement the function.
The processing by the Send2Linux function is as shown in the listing below.

```c
void Send2Linux(struct canctl_t *candata)
{
    uint32 ret;

    /* Waiting for data on Channel 1 buffer is sent */
    while((CDDICCOM_SND_IDLE!=CddIccom_GpChannelStatus[1].enSndSta[CDDICCOM_CTA_BOTTOM])&&
         (CDDICCOM_SND_IDLE!=CddIccom_GpChannelStatus[1].enSndSta[CDDICCOM_CTA_UPPER]))
    {
        /* Do nothing */
    }
    ret = CddIccom_Ch1SendRun(&candata->d, sizeof(candata->d));
    if (ret != E_OK) {
        Error processing;
    }
}
```

CddIccom_GpChannelStatus[1].enSndSta[] is a global variable indicating the state of the transmission buffer. Since the transmission buffer for MCAL is a double-buffer, the function monitors the states of the two buffers. Processing by the function leaves the loop on detection of either of the two being in the idle state and the CddIccom_Ch1SendRun function transmits the CAN data. The array number “1” in CddIccom_GpChannelStatus[1] indicates that the core which is currently operating is the G4MH. Similarly, the array number “0” indicates the CR52. Note that although array numbers are used to distinguish between the G4MH and CR52 cores in the MCAL sample program, this is not always required.

For details on the CddIccom_Ch1SendRun function, see section 6.1, Inter-Core Communications.
6.3 Processing by the CA55 Core

After the power supply to the R-Car S4 is turned on, the IMX boot loader transfers the code for the CA55 core to RAM then activates the core. After that, activation proceeds in the order U-Boot and then Linux. This subsection gives an overview of the processing by the CA55 sample program.

![CA55 sample program](image)

**Figure 12** Overview of the Processing by the CA55 Sample Program

6.3.1 Initializing the ICCOM Driver

The three functions for the ICCOM driver shown in the listing below are called to initialize the driver.

```
static void init_iccom(void)
{
    iccom_channel_t pChannelHandle;
    int32_t result = ICCOM_OK;
    uint32_t user_MsgSize = 2048;
    result = R_ICCOM_SetBuffSize(user_MsgSize);
    assert (result == ICCOM_OK);
    result = iccom_driver_init();
    assert (result == ICCOM_OK);
    result = iccom_library_init(&pChannelHandle, cb_iccom_recv);
    printf("R_ICCOM_Open: %d
\n", result);
    assert (result == ICCOM_OK);
}
```

This function specifies the size of the buffer to be used for inter-core communications. The maximum size is 2048 bytes.

This function initializes the ICCOM driver.

This function establishes communications with the other-party core. It also specifies the callback function at the time reception is completed.

After a successful call of the iccom_library_init function, communications with the G4MH core are established. The processing to make the G4MH core wait for the initialization processing of the CA55 core described in section 6.2.3.4, Initializing the MCAL ICCOM Driver, leaves its loop.
6.3.2 Activating the Web Server Thread

The CA55 core uses libwebsockets to activate the Web server thread. Running libwebsockets requires preparation of the callback function for libwebsockets in advance. Firstly, the Web server directs initialization through the callback function for libwebsockets. In this sample program, the CAN data analysis thread (<2> in the figure below) is activated within the initialization processing of the callback function. The flow from using inter-core communications to obtain CAN data to control over the Web browser display is given below.

**Figure 13  Flow from Inter-Core Communications to Control over the Web Display**

<1> In the reception of CAN data transferred from the G4MH core, the ICCOM driver, which supports inter-core communications, calls the ICCOM reception callback function (db_iccom_recv) that was specified at the time of initialization of the driver. The ICCOM reception callback function stores the data in a temporary storage buffer, awakens the CAN data analysis thread that has been placed in the wait state, and return from the function immediately follows that. Note that this callback function is handled within the interrupt processing.

<2> The CAN data analysis thread captures the CAN data from the temporary storage buffer, analyzes the data, generates commands to control the Web display, and stores the commands in the ring buffer for libwebsockets. After that, the CAN data analysis thread requests that the Web server thread transmit a command.

<3> On receiving a request to transmit a command, the Web server thread informs the callback function for libwebsockets indicated by <4> in the above flow of a request for transmission. In addition, on receiving a request to confirm data for transmission, the Web server thread captures the data from the ring buffer and informs the callback function <4> again of being ready for transmission.

<4> The processing corresponding to the instruction from the Web server is written in the callback function for libwebsockets. In response to the instruction being a request for transmission, the callback function requests confirmation by the Web server of the readiness of data for transmission. In response to an instruction to convey readiness for transmission, the callback function uses the lws_write function to
transmit the command to the Web browser running on the PC through the WebSocket communications path.
7. Building the Software from the Sample Code

The following shows the procedures for building the executable modules.

• Procedure for building the executable modules
1. Build the G4MH development environment.
2. Build and install the G4MH sample program.
3. Build the CA55 development environment.
4. Build and install the CA55 sample program.
5. Build the CARLA development environment.
6. Install the sample program for CAN output for use by the CARLA simulator.

7.1 Building the G4MH Development Environment

Download CS+ for CC V8.07.00 under Windows 10 or Windows 11 from the following URL.
https://www.renesas.com/products/software-tools/tools/ide/csplus.html#downloads

Decompress and install the zip file. Make a note on the destination directory for installation at this time.

Copy the “CS+” folder in the destination directory for installation to any folder other than “C:\Program Files” and rename the CS+ folder (example: C:\CS+forPre_i).

If the “C:\CS+forPre_i\CC” (used above as an example) folder contains the Device_Custom folder, rename the Device_Custom folder (example: C:\CS+forPre_i\CC\Device_Custom → C:\CS+forPre_i\CC\_Device_Custom).

Copy the “Device_Custom” folder obtained from the CS+ device custom file for the G4MH (see footnote 1) to the C:\CS+forPre_i\CC\folder.

Run CS+forPre_i(for example)\CC\CubeSuiteW+.exe.

Activation of the CS+ window indicates completion of building the G4MH development environment.

Note: Attempting to run the program by double-clicking on *.mtpj or from the Windows menu leads to incorrect operation.
7.2 Building and Installing the G4MH Sample Program

Build the G4MH sample program by following the procedure below.

Install Git.
Deploy the sample program.
Copy the MCAL code.
Apply the patch to the MCAL code.

Building Installation

Installing Git
Download Git for Windows from https://gitforwindows.org/ and install it.

Deploying the sample program

Create a desired working folder. In this description, the sample program is deployed to the C:¥g4mh folder as an example.
Create the C:¥g4mh folder, decompress the attached RCarS4_G4MH_ICCOM_Sample.zip file, and copy the g4mh¥sample folder to the C:¥g4mh¥ folder.

Copying the MCAL code
Copy the mcal¥v3.14.0¥sw_src¥g4mh_mcal folder in RCar SDK MCAL v3.14.0 to the C:¥g4mh¥ folder.

Applying the patch to the MCAL code
Decompress the attached RCarS4_G4MH_ICCOM_Sample.zip file, and copy the g4mh¥mcal314.patch file to the C:¥g4mh¥ folder.
Activate Git BASH.
Executed the commands below.
  $ cd /c/g4mh
  $ patch -p2 < mcal314.patch

Building
Follow the procedure described in section 7.1, Building the G4MH Development Environment, up to the activation of CS+.
Note that, as stated in section 7.1, this is not done from the Windows menu nor mtpj. file.
Select [File (F)] → [Open a file (O) … ] → C:¥g4mh¥sample¥apn_g4mh.mtpj in the dialog box for opening a file, and the project will open.
Select [Build (B)] → [Build apn_g4mh (U)].
After the successful completion of these operations, the apn_g4mh.mot file will have been generated in the c:¥g4mh¥sample¥DefaultBuild folder.
Installation

The G4MH cores can be activated in the following two ways: by the E2 emulator or from flash ROM.
Installation processing is not required if the E2 emulator is used. Activation of the G4MH cores from the flash ROM is described below.

Connect the CN20 connector of the evaluation board to the PC via a USB serial cable and activate TeraTerm. For details on the settings of the serial port, see Table 3, Serial Port Settings.

![Figure 14 Connections in Writing to Flash ROM](image)

The figure below shows changes to the switch settings on the switch board from those shown in Figure 7, Switch Board Settings.
Set switches 5, 7, and 8 of the SW1 switch block to OFF. The settings of the switches other than those on the SW1 switch block are not changed.

![Figure 15 Settings of the SW1 Switch Block on the Switch Board When Writing to Flash ROM](image)
Turn on the power to the board; the message shown in the figure below appears and the board waits for downloading of the program.

Drag and drop the ICUMXA_Flash_writer_SCIF_DUMMY_CERT EB203000_S4.mot file, which is provided as part of the RCar XOS SDK-Gateway V3.14, from the explorer to TeraTerm or open the file by selecting [File] → [Send file] from the TeraTerm menu bar.

After the completion of downloading, the program for writing to the flash ROM outputs the text shown in the figure below and waits for input by the user. Enter the command and values in red in the figure. Note that after “1” is entered in response to “Select (1-3)>”, the user is prompted to check the settings of SW1 and SW2. When the settings are correct, enter “y”. The line where “y” was entered is overwritten by the next message.

The flash writer waits for downloading.
Drag and drop the `apn_g4mh.mot` file from the explorer to TeraTerm or open the file by selecting [File] → [Send file] from the TeraTerm menu bar.

![Figure 18 Flash Writer Message to Request Confirmation](image)

After the file has been transferred, a message to request confirmation of the erasure of flash ROM is output. Check the storage addresses and enter "y".

```
SPI Data Clear(H'FF) Check :H'00900000-0093FFFF, Clear OK?(y/n)
```

![Figure 19 Flash Writer Message for Completion of Writing](image)

Display of the "SAVE SPI-FLASH....... complete!" message indicates completion of writing to the flash ROM.

Turn off the power supply and restore the settings of the SW1 switch block on the switch board to those shown in Figure 7.

Turn on the power supply again, and the G4MH sample program will be activated. Just LED9 described in Figure 10 in 8.1.2 turns on in case the program runs correctly.
7.3 Building the CA55 Development Environment

7.3.1 Building the Basic Yocto

Use build_yocto.sh included in the software development kit (SDK) 3.8 provided by Renesas to build the Yocto development environment.

Environment:

<table>
<thead>
<tr>
<th>Host PC</th>
<th>Recommended OS: Ubuntu 20.04 LTS (64 bits).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The host must be connected to the Internet.</td>
</tr>
<tr>
<td></td>
<td>Free disk space: Approximately 100 Gbytes</td>
</tr>
</tbody>
</table>

It can be deployed in any directory; the following shows an example of deployment in the $HOME/rcars4sdk directory.

```
$ mkdir $HOME/rcars4sdk
$ cd $HOME/rcars4sdk
$ sudo -E apt-get update
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils python3-jinja2 libegl1-mesa libsdll1.2-dev pylint3 xterm libarchive-zip-perl
```

# Copy the build_yocto.sh file to the current directory.

```
$ ./build_yocto.sh spider gateway
```

# Building takes several hours. When building the basic Yocto is completed normally,
# the kernel image, root file system, and device tree will have been generated in the
# $HOME/rcars4sdk/build-spider-gateway/tmp/deploy/images/spider/ directory.

Building the basic Yocto is completed at this time.
7.3.2 Building Yocto for the Application

Though the basic Linux for the R-Car S4 is generated by following the procedure described in section 7.3.1, Building the Basic Yocto, that Yocto requires rebuilding because it does not include libwebsockets, which the sample program requires. The procedure for building is continued from section 7.3.1, Building the Basic Yocto.

```sh
$ cd $HOME/rcars4sdk
$ source poky/oe-init-build-env build-spider-gateway
$ vi conf/local.conf
Add libwebsockets by adding the following line at the end of local.conf.

```

IMAGE_INSTALL_append = " libwebsockets"
```

Note: Insert a space before libwebsockets.

$ bitbake rcar-image-gateway

After the successful completion, the root file system will have been generated in the following file.

tmp/deploy/images/spider/rcar-image-gateway-spider.tar.bz2

Deploy the generated root file system to an eMMC on the R-Car S4 evaluation board and then activate Linux.

The processing for building Linux for the application has been completed at this point.
7.3.3 Building Yocto SDK

After having built Yocto, build the SDK for use in building the application. The procedure for building the SDK is given below.

```
$ cd $HOME/rcars4sdk
$ source poky/oe-init-build-env build-spider-gateway
$ bitbake rcar-image-gateway -c populate_sdk
  # Building takes several hours.
  # After having built the SDK, install it.
  $ sudo ./tmp/deploy/sdk/poky-glibc-x86_64-rcar-image-gateway-aarch64-spider-toolchain-3.1.11.sh
  # A message "Enter target directory for SDK", which requires entering the destination for installation of
  # the SDK, appears during the installation process.
  # If the default directory setting /opt/poky/3.1.11 does not create a problem, press the Enter key without
  # changing it.
```

Building the CA55 development environment is completed at this time.

7.4 Building and Installing the CA55 Sample Program

Use the SDK for the CA55 that was generated in the previous subsection to build the CA55 sample program. The procedure for building and installing the sample program is given below.

It can be deployed in any directory; the following shows an example of deployment in the $HOME/rcars4app directory.

```
$ mkdir $HOME/rcars4app
$ unzip {…}/RCarS4_G4MH_ICCOM_Sample.zip 'ca55/*'
$ mv ca55/*.
$ rmdir ca55
$ ls -F
  Makefile
  sample/

Copy rcar-xos¥v3.14.0¥sw_src¥renesas folder included in “RCar SDK MCAL v3.14.0”.
$ cp -a (SDK directory)/rcar-xos/v3.14.0/sw_src/Renesas .

Set up the SDK environment. Specify the directory specified as the destination for installation of the SDK in section 7.3.3, Building Yocto SDK. The following command is with the default setting.
$ source /opt/poky/3.1.11/environment-setup-aarch64-poky-linux
$ make

After successful completion of the process, the iccsample file and html/ folder will have been generated in the build/bin/ directory.

Copy the generated file and folder to /home/root/ on the R-Car S4 evaluation board.
The following is an example with scp in use.
$ cd build/bin
$ scp -r iccsample html/ root@{IP address of the R-Car S4 evaluation board}:

Close the working terminal window to reset SDK environment. Open new terminal window and proceed
following build steps on it.

7.5 Building the CARLA Development Environment

The table below shows the elements of a recommended environment for running the CARLA simulator.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS and version</td>
<td>Ubuntu 20.04</td>
</tr>
<tr>
<td>Free disk space</td>
<td>130 Gbytes</td>
</tr>
<tr>
<td>GPU</td>
<td>GPU card with at least 6 Gbytes of memory</td>
</tr>
</tbody>
</table>


Note that operation had also been confirmed in an environment for confirming operation in which the GPU had a specification inferior to that in the table above and was found to be slower than expected.

Any directory can be used as a working directory; the $HOME/work directory is used as an example in the following description.

```
$ cd $HOME/work
$ sudo -E apt update
$ sudo -E apt install can-utils libomp5 python3-pip
$pip3 install --upgrade pip
$ echo "export PATH=$HOME/.local/bin:$PATH" >> ~/.bashrc
$ source ~/.bashrc
$ pip3 -V ; Confirm that pip23.1.2 or above has been installed.
$ pip3 install --user numpy ; Install numpy.
$ pip3 install --user pygame ; Install pygame.
$ pip3 install --user cantools ; Install the CAN module.
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 1AF1527DE64CB8D9
```

```
Proceed following steps in case of fail with above steps.

Access to http://keyserver.ubuntu.com with web browser

Input “0x1AF1527DE64CB8D9” in “Search for an OpenPGP Public Key, ..” box

and then click “Search Key” button.

Click “pub rsa4096...”

Save all of output PGP Public Key to any file. (i.e. save as key.txt)
$ cat key.txt | sudo apt-key add –
```

```
$ sudo -E add-apt-repository $y
"deb [arch=amd64] http://dist.carla.org/carla $(lsb_release -sc) main"
$ sudo -E apt update
$ sudo -E apt-get install carla-simulator
$ echo "export PYTHONPATH=/opt/carla-simulator/PythonAPI/carla/dist/carla-0.9.13-py3.7-linux-x86_64.egg" >> ~/.bashrc
$ source ~/.bashrc
```

```
sudo vi /etc/modules

can

can_raw

slan
```

The processing for preparation of the CARLA simulator has been completed at this point.
7.6 Installing the Sample Program for CAN Output for Use by the CARLA Simulator

Deploy the sample program for CAN output for use with the CARLA simulator in the RCarS4_G4MH_ICCOM_Sample.zip file of the sample software package.

```bash
$ cd /opt/carla-simulator/PythonAPI/examples
$ sudo unzip {…}/RCarS4_G4MH_ICCOM_Sample.zip 'pc/carla/*'
$ sudo mv pc/carla/* .
$ sudo rm -rf pc/
```

8. Executing the Sample Program

Execute the sample program by following the procedure listed below.

- Execute the G4MH sample program.
- Execute the CA55 sample program.
- Connect the sample display through a Web browser.
- Execute the sample program for the CARLA simulator.
8.1 Executing the G4MH Sample Program

The G4MH sample program can run by a connected E2 emulator or by booting up from flash ROM. Execution by booting up from flash ROM is obtained by simply turning on the power supply after having configured the switch settings shown in Figure 7, Switch Board Settings, which leads to automatic activation of the program. Note that this case also requires the operations described in section 8.2.1, Setting up U-Boot. On normal completion of activating the program, the LEDs provide confirmation of the state of the program. For details, see section 8.1.2, State Indication with Use of the LEDs.

The following shows the procedure for execution with the use of the E2 emulator.

![E2 Emulator Connection Diagram](image)

The figure below shows changes to the switch settings on the switch board from those shown in Figure 7, Switch Board Settings.

The settings of the switches other than those on the SW2 switch block are not changed.

![Settings of SW2 on the Switch Board When the E2 Emulator Is to Be Connected](image)
8.1.1 Operations with CS+

Since execution with the use of the E2 emulator leads to the CA55 core being activated, the hot plug-in function is required for connection. For details, see documents 6 and 7 under References in this application note. Activate CS+ by following the procedure described in section 7.1, Building the G4MH Development Environment.

Note that, as stated in section 7.1, this is not done from the Windows menu nor mtpj file.

Open the file.

![Figure 22 Window Displayed after Activation of CS+](image)

1. Selecting C:\g4mh\apn_g4mh.mtpj specified as an example in section 7.2, Building and Installing the G4MH Sample Program, opens the project.
2. Turn on the power supply to the R-Car S4 evaluation board.
3. Select [Debug (D)] and then [Hot plug-in (H)] while the flat cable connected to the E2 emulator is disconnected from the board.
4. After a message indicating that preparation of the hot plug-in function has been completed, connect the flat cable connected to the E2 emulator and click on the [OK] button in the dialog box.
5. Select [Debug (D)] and then [Stop (S)] to stop execution.
6. Download the program to code SRAM for the G4MH cores by selecting [Debug (D)] and then selecting [Rebuild and download to debug tool (W)] or by pressing the F6 key.
7. Enter __start in the program counter (PC) column in the CPU register window.
8. Selecting [Debug (D)] and then selecting [Run (G)] or pressing the F5 key starts operation of the G4MH sample program.

The CA55 core also starts operating at the same time as activation of the G4MH cores. Since setting up U-Boot as described in section 8.2.1, Setting up U-Boot, is required, place the CA55 core in the state of waiting for command input in U-Boot mode by pressing any key in the TeraTerm window within the time of waiting for keyboard input.
8.1.2 State Indication with Use of the LEDs

Whether the G4MH program is running normally can be confirmed by checking the LEDs. The following describes the positions of the LEDs and details of the conditions under which each of the LEDs is turned on.

![Positions of the LEDs](image)

Figure 23 Positions of the LEDs

<table>
<thead>
<tr>
<th>LED No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>This LED lights up after initialization.</td>
</tr>
<tr>
<td>10</td>
<td>This LED lights up after the establishment of ICCOM communications between the G4MH and CA55 (Linux) cores.</td>
</tr>
<tr>
<td>11</td>
<td>This LED outputs 4-bit code values to indicate the state by repeatedly being turned on and off. Turned on for a short time: 0; turned on for a longer time: 1; output order: MSB to LSB 0001: The G4MH core is in the state of waiting to receive CAN data (the G4MH core has received no CAN data). 0010: The G4MH core has received CAN data (the G4MH core has received CAN data and transmitted them to the CA55 core at least once).</td>
</tr>
</tbody>
</table>
8.2 Executing the CA55 Sample Program

8.2.1 Setting up U-Boot

Add the following parameter to the existing bootargs variables of U-Boot.

```
bootargs=... cma=256M clk_ignore_unused
```

Enter the following command to activate Linux.

```
=> run bootcmd
```

When the G4MH sample program is to be run by booting up from flash ROM.
Turning on the power supply automatically activates the G4MH and CA55 cores.

The following describes operations after booting-up Linux.

```
spider login: root ; Log in as the root user (no password).
root@spider:~# ./iccsample ; Execute the sample program.
```

The state of waiting for connection with the sample display through a Web browser and the reception of CAN data is entered at this time.

If initialization of inter-core communications by the G4MH core failed, the sample program will stop running.
In such cases, check the procedure again and re-execute it from turning on the power supply.

8.3 Connection to the Sample Display through a Web Browser

Activate a Web browser to run on the PC and enter the IP address of the R-Car S4.
After access to http://[IP address of the R-Car S4]:7681, a virtual vehicle console is displayed.
(Import again if following image are not displayed.)
8.4 Executing the Sample Program for the CARLA Simulator

Activation of the sample program for the CARLA simulator on the PC in the CARLA development environment is described below.

Connect a CANUSB cable to the PC.

```
$ sudo modprobe can
$ sudo modprobe can_raw
$ sudo modprobe slcan
$ sudo slcand -o -c -f -s5 /dev/ttyUSB{n} slcan0
```

Note that {n} is replaced with a number starting at 0 that is allocated to each USB device. Specify the number allocated to the CANUSB connection.

```
$ sudo ip link set slcan0 up
$ cd /opt/carla-simulator
$ ./CarlaUE4.sh &
```

Execute the CARLA server program in the background.

Install of appropriate GPU driver is required for start of CarlaUE4.sh.

The CARLA simulator consists of the server software and simulator software for a client.

Executing CarlaUE4.sh activates the CARLA server and following display appears.

The CARLA server supports the CARLA simulator software in terms of simulation processing such as physics calculation. After that, activate the CARLA simulator.

![Figure 25 Display by the CARLA Server](image-url)
Activating the CARLA simulator

$ cd /opt/carla-simulator/PythonAPI/examples
$ python3 can_control.py

; Activate the CARLA simulator.

Activating the CARLA simulator displays a car on the map. The output of CAN data starts at the same time. Note that displayed car image is changed according to the starts.

![Window Handled by CARLA](image.png)

Figure 26 Window Handled by CARLA

Keyboard operations can control the gear shift, accelerator, steering wheel, opening and closing of doors, etc. Pressing the w key repeatedly makes the car in the display run forwards and the speedometer displayed in the Web browser changes in synchronization with the action on the display. For details on how to operate the car with a keyboard, refer to the output when the CARLA simulator is booted up.
References

   (Obtain the latest version from the Renesas Web site.)

2. R-Car S4 Series Linux Interface Specification Yocto recipe Start-Up Guide
   (Obtain SDK 3.14.0 from the R-Car Market Place or your local Renesas Electronics sales representative.)

3. xOS3 ICCOM User’s Manual
   (Obtain SDK 3.14.0 from the R-Car Market Place or your local Renesas Electronics sales representative.)

   (Obtain SDK 3.14.0 from the R-Car Market Place or your local Renesas Electronics sales representative.)

   (Obtain SDK 3.14.0 from the R-Car Market Place or your local Renesas Electronics sales representative.)

6. CS+: Setting Up an Environment for Debugging the G4MH Cores for the R-Car S4 (alpha version of the document)
   (Obtain the latest version from the Renesas Web site.)

   (Setting up the Hardware Environment when Using the R-Car S4)
   (Obtain the latest version from the Renesas Web site.)

   (Obtain SDK 3.14.0 from the R-Car Market Place or your local Renesas Electronics sales representative.)

   (Obtain SDK 3.14.0 from the R-Car Market Place or your local Renesas Electronics sales representative.)
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>May 23, 2023</td>
<td>All</td>
<td>Initial version</td>
</tr>
<tr>
<td>1.01</td>
<td>Aug. 23, 2023</td>
<td>All</td>
<td>Modified according to change of supported SDK version</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.
   Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.).

7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
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 in any other use of the circuits, software, and information in the design of your products. 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 infringement and liability for 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.) as a reference. 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.

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.