

# SMARC EVK of RZ/Five

## Linux Start-up Guide

R01US0618EJ0101

Rev.1.01

Oct. 31, 2023

### Introduction

This document provides a guide to prepare RZ/Five reference boards to boot up with the Verified Linux Package.

This guide provides the following information:

- Building procedure
- Preparation for use
- Boot loader and U-Boot
- How to run this Linux package on the target board
- How to create a software development kit (SDK)

### Target Reference Board

RZ/Five reference board

- RZ/Five Evaluation board Kit (smarc-rzfive) (\*1)
  - RZ/Five SMARC Module Board (P/N: RTK9743F01C01000BE)
  - RZ SMARC Series Carrier Board (P/N: RTK97X4XXXB00000BE)

(\*1) “RZ/Five Evaluation board Kit” includes the RZ/Five SMARC Module Board and the RZ SMARC Series Carrier Board.

The “Evaluation board Kit for RZ/Five MPU” will be called “RZ/Five Evaluation Kit” in the next section.

### Target Software

- RZ/Five Verified Linux Package version 3.0.5 or later. (hereinafter referred to as “VLP/F”)

## Contents

1. Environment Requirement.....	4
2. Build Instructions.....	6
2.1 Notes .....	9
3. Preparing the SD Card .....	12
3.1 Write files to the microSD card (used wic image).....	12
3.2 Write files to the microSD card (not used wic image).....	14
3.2.1 Create a microSD card for boot Linux .....	14
3.2.2 Write files to the microSD card.....	18
4. Reference Board Setting.....	19
4.1 Preparation of Hardware and Software.....	19
4.1.1 How to set boot mode and input voltage .....	20
4.1.2 How to set SW1 .....	21
4.1.3 How to use debug serial (console output) .....	21
4.1.4 Power supply .....	22
4.1.5 Building files to write.....	23
4.1.6 Settings .....	23
4.2 Download Flash Writer to RAM.....	26
4.3 Write the Bootloader.....	28
4.4 Change Back to Normal Boot Mode .....	30
5. Booting and Running Linux.....	32
5.1 Power on the board and Startup Linux.....	32
5.2 Shutdown the Board .....	33
6. Building the SDK .....	34
7. Application Building and Running .....	35
7.1 Make an application .....	35
7.1.1 How to extract SDK.....	35
7.1.2 How to build Linux application.....	35
7.2 Run a sample application .....	37
8. Appendix.....	38
8.1 Preparing Flash Writer.....	38
8.1.1 Preparing cross compiler.....	38
8.1.2 Building Flash Writer .....	38
8.2 How to replace the SMARC Module Board.....	39
8.3 Device drivers.....	40
9. Revision History .....	41

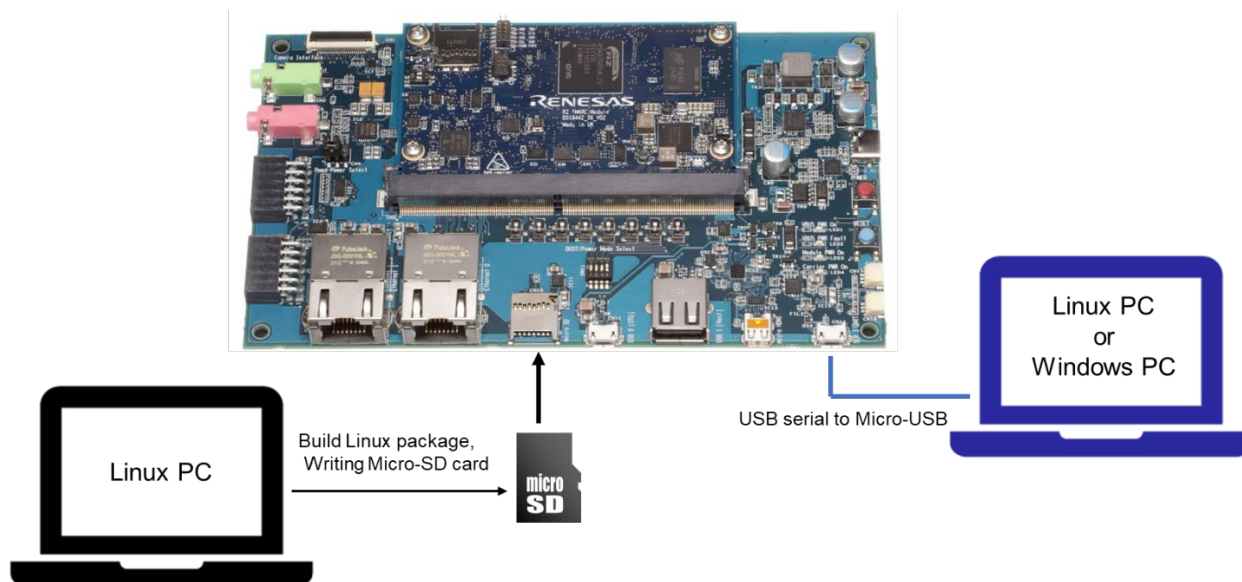
**Website and Support.....42**

## 1. Environment Requirement

The environment for building the Board Support Package (hereinafter referred to as “BSP”) is listed in the Table 1. Please refer to the below documents for details about setting up the environment.

A Linux PC is required for building the software.

A Windows PC can be used as the serial terminal interface with software such as TeraTerm.



**Figure 1 Recommend environment**

Note: The board shown in Figure 1 is RZ/V2L, but RZ/Five has the same structure.

**Table 1. Equipment and Software for Developing Environments of RZ/Five Linux Platform**

Equipment	Description
Linux Host PC	Used as build/debug environment 100GB free space on HDD or SSD is necessary
OS	<b>Ubuntu 20.04 LTS</b> 64 bit OS must be used. 20.04 inside a docker container also OK.
Windows Host PC	Used as debug environment, controlling with terminal software
OS	Windows 10 or Windows 11
Terminal software	Used for controlling serial console of the target board Tera Term (latest version) is recommended Available at <a href="https://ttssh2.osdn.jp/index.html.en">https://ttssh2.osdn.jp/index.html.en</a>
VCP Driver	Virtual COM Port driver which enables to communicate Windows Host PC and the target board via USB which is virtually used as serial port. Available at: <a href="http://www.ftdichip.com/Drivers/VCP.htm">http://www.ftdichip.com/Drivers/VCP.htm</a>
USB serial to micro-USB Cable	Serial communication (UART) between the Evaluation Board Kit and Windows PC. The type of USB serial connector on the Evaluation Board Kit is Micro USB type B.
microSD Card	Use to boot the system, and store applications. <b>Note that use a micro-SDHC card for the flash writer.</b>

Most bootable images VLP/F supports can be built on an “offline” environment.

The word “offline” means an isolated environment which does not connect to any network. Since VLP/F includes all necessary source codes of OSS except for the Linux kernel, VLP/F can always build images in this “offline” environment without affected from changes of repositories of OSS. Also, this “offline” environment reproduces the same images as the images which were verified by Renesas.

Below images can be built “offline”.

- core-image-minimal
- core-image-bsp

## 2. Build Instructions

### 2.1 Building images

This section describes the instructions to build the Board Support Package.

Before starting the build, run the command below on the Linux Host PC to install packages used for building the BSP.

```
$ sudo apt-get update
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping libssl1.2-dev xterm p7zip-full libyaml-dev \
libssl-dev bmap-tools
```

Please refer to the URL below for detailed information:

- <https://docs.yoctoproject.org/3.1.26/brief-yoctoprojectqs/brief-yoctoprojectqs.html>

#### (1) Create a working directory at your home directory, and decompress Yocto recipe package

Run the commands below and set the user name and email address before starting the build procedure. **Without this setting, an error occurs when building procedure runs git command to apply patches.**

```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

**Copy all files obtained from Renesas into your Linux Host PC prior to the steps below. The directory which you put the files in is described as <package download directory> in the build instructions.**

Create a working directory at your home directory, and decompress Yocto recipe package

Run the commands below. The name and the place of the working directory can be changed as necessary.

```
$ mkdir ~/rzfive_vlp_<package version>
$ cd ~/rzfive_vlp_<package version>
$ cp ../<package download directory>/*.zip .
$ unzip ./RTK0EF0045Z0025AZJ-<package version>.zip
$ tar zxvf ./RTK0EF0045Z0025AZJ-<package version>/rzfive_vlp_<package version>.tar.\
gz
```

Note) Please note that your build environment must have 100GB of free hard drive space in order to complete the minimum build. The Yocto BSP build environment is very large. Especially in case you are using a Virtual Machine, please check how much disk space you have allocated for your virtual environment.

<package version>: e.g v3.0.5

#### (2) Build Initialize

Initialize a build using the 'oe-init-build-env' script in Poky and point TEMPLATECONF to platform conf path.

```
$ TEMPLATECONF=$PWD/meta-renesas/meta-rzfive/docs/template/conf/ source \
poky/oe-init-build-env build
```

### (3) Add layers (Optional)

- **Docker:** Please run the below commands if you want to include Docker. This means running Docker on the RZ board, not as using Docker as part of your build environment.

```
$ bitbake-layers add-layer ../meta-openembedded/meta-fileformats
$ bitbake-layers add-layer ../meta-openembedded/meta-networking
$ bitbake-layers add-layer ../meta-virtualization
```

### (4) Decompress OSS files to “build” directory (Optional)

Run the commands below. This step is not mandatory and able to go to the step (5) in case the “offline” environment is not required. All OSS packages will be decompressed with this '7z' command.

```
$ cp ../../<package download directory>/*.7z .
$ 7z x oss_pkg_rzfive_<package version>.7z
```

Note) If this step is omitted and BB\_NO\_NETWORK is set to “0” in next step, all source codes will be downloaded from the repositories of each OSS via the internet when running bitbake command. Please note that if you do not use an “offline” environment, a build may fail due to the implicit changes of the repositories of OSS.

After the above procedure is finished, the “offline” environment is ready. If you want to prevent network access, please change the line in the “~/rzfive\_vlp\_<package version>/build/conf/local.conf” as below:

```
BB_NO_NETWORK = "1"
```

To change BB\_NO\_NETWORK from “0” to “1”.

Note) Open source software packages contain all source codes of OSSs. These are the same versions of OSSs used when VLP/F was verified.  
If you are just evaluating VLP/F and RZ/Five series, open source software packages are not mandatory to use.  
Usually, all the software can be built without using these files if your build machine is connected to the Internet.

Open source software packages are required for an “offline” environment. The word “offline” means an isolated environment which does not connect to any network. VLP/F can always build images in this “offline” environment by using these packages without affected from changes of original repositories of OSSs. Also, this “offline” environment always reproduces the same images as the images which were verified by Renesas. Note that if you build without using open source software packages, there are possibilities to use different source codes than Renesas used due to the implicit changes of the repositories of OSSs.

**(5) Start a build**

Run the commands below to start a build. Building an image can take up to a few hours depending on the user's host system performance.

Build the target file system image using bitbake

```
$ MACHINE=smarc-rzfive bitbake core-image-<target>
```

<target> can be selected in below. Please refer to the Table 2 for supported image details.

- core-image-minimal
- core-image-bsp

After the build is successfully completed, a similar output will be seen, and the command prompt will return.

NOTE: Tasks Summary: Attempted 3795 tasks of which 8 didn't need to be rerun and all succeeded.

All necessary files listed in Table 3 will be generated by the bitbake command and will be located in the **build/tmp/deploy/images** directory.

VLP/F can build a few types of images listed in the Table 2.

**Table 2. Supported images of VLP/F**

Image name	Purpose
core-image-minimal	Minimal set of components
core-image-bsp	Minimal set of components plus audio support and some useful tools

**Table 3. Image files for RZ/Five**

<b>RZ/Five</b>	<b>Linux kernel</b>	Image-smarc-rzfive.bin
	<b>Device tree file</b>	Image-r9a07g043f01-smarc.dtb
	<b>root filesystem</b>	<image name>-smarc-rzfive.tar.bz2
	<b>Boot loader</b>	<ul style="list-style-type: none"> <li>• fit-smarc-rzfive.srec</li> <li>• spl-smarc-rzfive.srec</li> </ul>
	<b>Flash Writer</b>	Flash_Writer_SCIF_RZFIVE_SMARC.mot
	<b>SD image</b>	core-image-<image name>-smarc-rzfive.wic.gz core-image-<image name>-smarc-rzfive.wic.bmap



## 2.2 Notes

### (1) GPLv3 packages

In this release, the GPLv3 packages are disabled as default in *build/conf/local.conf*:

```
INCOMPATIBLE_LICENSE = "GPLv3 GPLv3+"
```

If you want to use GPLv3, just hide this line:

```
#INCOMPATIBLE_LICENSE = "GPLv3 GPLv3+"
```

### (2) CIP Core Packages

VLP/F includes Debian 10 (Buster) based CIP Core Packages and is enabled by the default settings. These packages can be changed.

Note that network access is required to start the build process when you enable these packages except for Buster which is set as the default setting.

CIP Core Packages are going to be maintained by the Civil Infrastructure Platform project. For more technical information, please contact Renesas.

#### 1. Buster (default):

The following lines are added as default in the *local.conf*:

```
# Select CIP Core packages
CIP_CORE = "1"
```

#### 2. Bullseye:

Please change "CIP\_MODE" in the *local.conf* to change from Buster to Bullseye:

```
# Select CIP Core packages by switching between Buster and Bullseye.
# - Buster (default) : build all supported Debian 10 Buster recipes
# - Bullseye         : build all supported Debian 11 Bullseye recipes
# - Not set (or different with above): not use CIP Core, use default packages
version in Yocto

CIP_MODE = "Bullseye"
```

#### 3. No CIP Core Packages:

If the CIP Core Packages are unnecessary, comment out and add the following lines to disable CIP Core Packages in the *local.conf*:

```
# Select CIP Core packages
#CIP_CORE = "1"
```

Note) The above 2 settings disable GPLv3 packages as default. In case the GPLv3 packages are required, please comment out the following line in the *local.conf*.

```
# INCOMPATIBLE_LICENSE = "GPLv3 GPLv3+"
```

By building the VLP/F, the packages will be replaced as below in the Table 4.

**Table 4. Versions of all Buster Debian packages**

Package	Buster Debian	Bullseye Debian
Attr	2.4.48	2.4.48
busybox	1.30.1	1.30.1
coreutils	6.9	6.9
GCC	8.3.0	-
glib-2.0	2.58.3	2.62.2
glibc	2.28	2.31
kbd	2.0.4	2.2.0
libgcrypt	1.8.4	1.8.5
openssh	7.9p1	8.2p1
Perl	5.30.1	5.30.1
pkgconfig	0.29	0.29.2
Quilt	0.65	0.66

**(3) ECC**

If you want to use ECC, see [r01us0647ej0100-rz-mpu\\_ECC\\_UME.pdf](#) included in the BSP manual set on the Renesas Web. Please also check the section 8.3.

**(4) WIC image**

The name “WIC” is derived from OpenEmbedded Image Creator (oeic). It includes image that system can boot it in hardware device.

From VLP/G v3.0.5, WIC is supported and below guidelines are shown how to use it to boot Renesas RZ/Five devices.

- Enable building WIC image in local.conf (default is enabled) by setting “**WKS\_SUPPORT**” to 1:

```
WKS_SUPPORT ?= "1"
```

- Defines additional free disk space created in the image in Kbytes (keep default value if unsure):

```
IMAGE_ROOTFS_EXTRA_SPACE = "1048576"
```

- Select wks file to be built by setting “**WKS\_DEFAULT\_FILE**” (keep default value if unsure). Currently, there are 2 types of wks defined in “meta-renesas/meta-rz-common/wic” for uSD/eMMC (channel 0) and uSD (channel 1).
- Building your desired core-image by running “bitbake core-image-x”. “core-image-x” should be one of following options:
  - core-image-minimal
  - core-image-bsp
- There are 2 files \*wic.bmap and \*wic.gz in deploy folder after building successfully. Example:
  - core-image-minimal-smarc-rzfive.wic.bmap
  - core-image-minimal-smarc-rzfive.wic.gz

**(5) Software bill of materials (SBoM)**

Software package data exchange (SPDX) is an open standard for SBoM that identifies and catalogs components, licenses, copyrights, security references, and other metadata relating to software.

From VLP/G v3.0.5, creating SPDX is supported and below guidelines are shown how to use it:

- Enable creating SPDX in local.conf (default is disabled) by uncommenting out below line:

```
#INHERIT += "create-spdx"
```

- Select below optional features to be supported for SDPX by enable in local.conf (all is disabled by default):

- **SPDX\_PRETTY**: Make generated files more human readable (newlines, indentation)

```
SPDX_PRETTY = "1"
```

- **SPDX\_ARCHIVE\_PACKAGED**: Add compressed archives of the files in the generated target packages in tar.gz files.

```
SPDX_ARCHIVE_PACKAGED = "1"
```

- SPDX is created and deployed in “tmp/deploy/spdx/\$MACHINE”. All information can be checked [here](#).s

**Note:** There is an issue when building SDK (example bitbake core-image-weston -c populate\_sdk) with SBoM SDPX support:

```
| ERROR: core-image-weston-1.0-r0 do_populate_sdk: Error executing a python function
| in exec_func_python() autogenerated:
| *** 1078:         return self._accessor.open(self, flags, mode)
|      1079:
|      1080:     def _raw_open(self, flags, mode=0o777):
|      1081:         """
|      1082:         Open the file pointed by this path and return a file descriptor,
| Exception: FileNotFoundError:
| [Errno 2] No such file or directory: 'tmp/work/smarc_rzg2l-poky-linux/core-image-
| weston/1.0-r0/spdx/sdk-work/poky-glibc-x86_64-core-image-weston-aarch64-smarc-rzg2l-
| target.spdx.json'
```

To fix this, please apply below change in “poky/meta/classes/populate\_sdk\_base.bbclass”:

```
-do_populate_sdk[cleandirs] = "${SDKDEPLOYDIR}"
+do_populate_sdk[cleandirs] += "${SDKDEPLOYDIR}"
```

### 3. Preparing the SD Card

You can prepare the microSD card by the following 2 methods. **Please select one of them and follow the steps.**

- 3.1 Write files to the microSD card (used wic image)
- 3.2 Write files to the microSD card (not used wic image)

#### 3.1 Write files to the microSD card (used wic image)

Set microSD card to Linux PC. And check the mount device name with fdisk command.

```
$ sudo fdisk -l
Disk /dev/sdb: 3.74 GiB, 3997171712 bytes, 7806976 sectors
Disk model: Storage Device
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xxxxxxxxx
$ umount /dev/sdb1
$ umount /dev/sdb2
```

Expand disk image.

```
$ sudo bmaptool copy <wic image>.wic.gz /dev/sdb
```

The file names of <wic image> is listed in the **Table 3**.

When you complete the above commands, the micro SD card is prepared correctly. Please skip the section 3.2.

**Table 5. File and directory in the micro SD card**

Type/Number	Filesystem	Contents
Primary #1	FAT32	Flash_Writer_SCIF_RZFIVE_SMARC.mot fit-smarc-rzfive.srec spl-smarc-rzfive.srec
Primary #2	Ext4(*1)	./  — bin  — boot      — Image     └─ r9a07g043f01-smarc.dtb  — dev  — etc  — home  — lib  — media  — mnt  — proc  — run  — sbin  — sys  — tmp  — usr  — var

Note \*1) Please refer to 2.2(4) WIC image for partition size specifications.

## 3.2 Write files to the microSD card (not used wic image)

### 3.2.1 Create a microSD card for boot Linux

To boot from SD card, over 4GB capacity of blank SD card is needed. You can use Linux Host PC to expand the kernel and the rootfs using USB card reader or other equipment.

Please format the card according to the following steps before using the card:

#### (1) Non-connect microSD card to Linux Host PC

```
$ lsblk
NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0 30.9G  0 disk
├─sda1    8:1    0  512M  0 part /boot/efi
├─sda2    8:2    0    1K    0 part
└─sda5    8:5    0 30.3G  0 part /
sr0     11:0    1 1024M  0 rom
```

#### (2) Connect microSD card to Linux Host PC with USB adapter

#### (3) Check the device name which is associated to the microSD card.

```
$ lsblk
NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0 30.9G  0 disk
├─sda1    8:1    0  512M  0 part /boot/efi
├─sda2    8:2    0    1K    0 part
└─sda5    8:5    0 30.3G  0 part /
sdb      8:16    1 29.7G  0 disk
└─sdb1    8:17    1 29.7G  0 part
sr0     11:0    1 1024M  0 rom
```

The message above shows the card associated with the `/dev/sdb`. **Be careful not to use the other device names in the following steps.**

#### (4) Unmount automatically mounted microSD card partitions

If necessary, unmount all mounted microSD card partitions.

```
$ df
Filesystem      1K-blocks      Used Available Use% Mounted on
udev              745652         0    745652   0% /dev
:
: snip
:
/dev/sdb1        511720      4904    506816   1% /media/user/A8D3-393B
$ sudo umount /media/user/A8D3-393B
```

If more than one partition has already been created on microSD card, unmount all partitions.

**(5) Change the partition table**

microSD card needs two partitions as listed in Table 6 .

**Table 6. Partitions of microSD card**

Type/Number	Size	Filesystem	Contents
Primary #1	500MB (minimum 128MB)	FAT32	Linux kernel Device tree
Primary #2	All remaining	Ext4	root filesystem

Set the partition table using the fdisk command like this.

```
$ sudo fdisk /dev/sdb
Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): o

Created a new DOS disklabel with disk identifier 0x6b6aac6e.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-62333951, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-62333951, default 62333951): +500M

Created a new partition 1 of type 'Linux' and of size 500 MiB.
Partition #1 contains a vfat signature.

Do you want to remove the signature? [Y]es/[N]o: Y

The signature will be removed by a write command.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): (Push the enter key)
First sector (1026048-62333951, default 1026048): (Push the enter key)
Last sector, +/-sectors or +/-size{K,M,G,T,P} (1026048-62333951, default 62333951): (Push the enter key)

Created a new partition 2 of type 'Linux' and of size 29.2 GiB.

Command (m for help): p
Disk /dev/sdb: 29.74 GiB, 31914983424 bytes, 62333952 sectors
Disk model: Transcend
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
```

Disk identifier: 0x6b6aac6e

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sdb1		2048	1026047	1024000	500M	83	Linux
/dev/sdb2		1026048	62333951	61307904	29.2G	83	Linux

Filesystem/RAID signature on partition 1 will be wiped.

Command (m for help): **t**

Partition number (1,2, default 2): **1**

Hex code (type L to list all codes): **b**

Changed type of partition 'Linux' to 'W95 FAT32'.

Command (m for help): **w**

The partition table has been altered.

Syncing disks.

Then, check the partition table with the commands below:

```
$ partprobe
$ sudo fdisk -l /dev/sdb
Disk /dev/sdb: 29.74 GiB, 31914983424 bytes, 62333952 sectors
Disk model: Maker name etc.
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x6b6aac6e

Device      Boot    Start        End    Sectors    Size Id Type
/dev/sdb1             2048    1026047    1024000    500M  b W95 FAT32
/dev/sdb2          1026048  62333951  61307904  29.2G  83 Linux
```



**(6) Format and mount the partitions**

If the partitions were automatically mounted after the step (4), please unmount them according to the step (3).

Then format the partitions using the command below:

```
$ sudo mkfs.vfat -v -c -F 32 /dev/sdb1
mkfs.fat 4.1 (2017-01-24)
/dev/sdb1 has 64 heads and 32 sectors per track,
hidden sectors 0x0800;
logical sector size is 512,
using 0xf8 media descriptor, with 1024000 sectors;
drive number 0x80;
filesystem has 2 32-bit FATs and 8 sectors per cluster.
FAT size is 1000 sectors, and provides 127746 clusters.
There are 32 reserved sectors.
Volume ID is a299e6a6, no volume label.
Searching for bad blocks 16848... 34256... 51152... 68304... 85072... 102096... 11937
6... 136528... 153552... 170576... 187472... 204624... 221648... 238928... 256208... 2
73744... 290768... 308048... 325328... 342480... 359504... 376656... 393680... 41057
6... 427216... 444624... 462032... 479184... 495952...

$ sudo mkfs.ext4 -L rootfs /dev/sdb2
mke2fs 1.45.5 (07-Jan-2020)
Creating filesystem with 7663488 4k blocks and 1916928 inodes
Filesystem UUID: 63dddb3f-e268-4554-af51-1c6e1928d76c
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

**(7) Remount microSD card**

```
$ umount /dev/sdb1
$ umount /dev/sdb2
```

After format, **remove the card reader and connect it again** to mount the partitions.

### 3.2.2 Write files to the microSD card

Check the mount point name with df command.

```
$ df
Filesystem      1K-blocks      Used Available Use% Mounted on
udev            745652          0    745652   0% /dev
:
snip
:
/dev/sdb1        510984         16    510968   1% /media/user/A299-E6A6
/dev/sdb2       30041556       45080   28447396   1% /media/user/rootfs
```

When you use RZ/Five Evaluation board Kit, copy kernel and device tree file to the first partition like as below.

```
$ cp $WORK/build/tmp/deploy/images/smarc-rzfive/<Linux kernel> /media/user/A299-E6A6
$ cp $WORK/build/tmp/deploy/images/smarc-rzfive/<Devise tree> /media/user/A299-E6A6
```

When you use RZ/Five Evaluation board Kit, expand rootfs to the second partition like as below.

```
$ cd /media/user/rootfs
$ sudo tar jxvf $WORK/build/tmp/deploy/images/smarc-rzfive/<root filesystem>
```

The file names of [<Linux kernel>](#), [<Devise tree>](#), and [<root filesystem>](#) are listed in the Table 3.

**Table 7. File and directory in the microSD card**

Type/Number	Size	Filesystem	Contents
Primary #1	Size specified when the partition was created. (minimum 128MB)	FAT32	Image-smarc-rzfive.bin Image-r9a07g043f01-smarc.dtb
Primary #2	Size specified when the partition was created.	Ext4	./  — bin  — boot  — dev  — etc  — home  — lib  — media  — mnt  — proc  — run  — sbin  — sys  — tmp  — usr  — var

## 4. Reference Board Setting

### 4.1 Preparation of Hardware and Software

The following environment of Hardware and Software is used in the evaluation.

Hardware preparation (Users should purchase the following equipment.):

- USB Type-C cable compatible with USB PD. (e.g. AK-A8485011 (manufactured by Anker))
- USB PD Charger 15W (5V 3.0A) or more. (e.g. PowerPort III 65W Pod (manufactured by Anker))
- USB Type-microAB cable (Any cables)
- PC Installed FTDI VCP driver and Terminal software (Tera Term) (\*1)

(\*1) Please install the FTDI driver that can be following website (<https://www.ftdichip.com/Drivers/VCP.htm>).

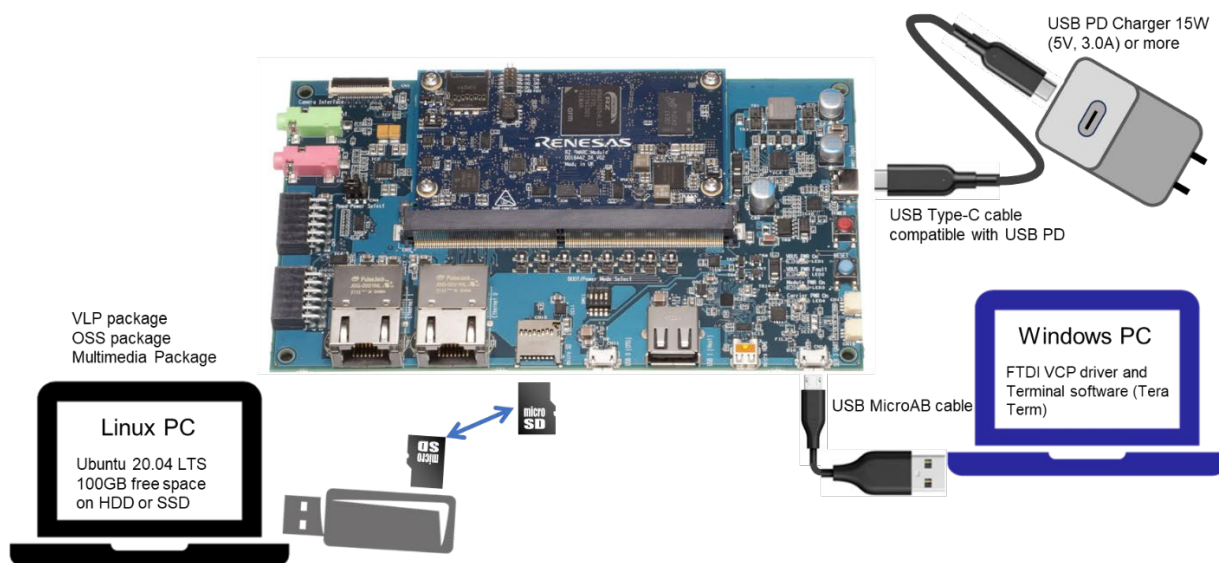
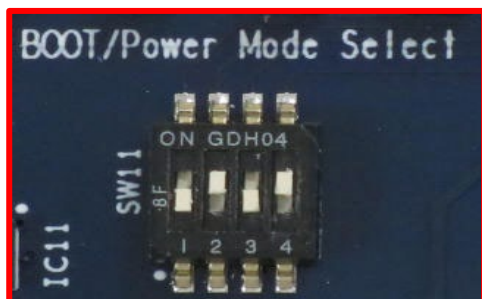


Figure 2. Operating environment

#### 4.1.1 How to set boot mode and input voltage

Please set the SW11 settings as follows.

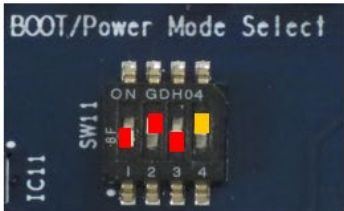



- Pin no1 to no3 of the SW11 is used to control boot mode of RZ/Five.
- Pin no4 of the SW11 is used to control the input voltage from power charger to 5V or 9V. Please use a 5V setting as initial setting.



SW11-1	OFF
SW11-2	ON
SW11-3	OFF
SW11-4	ON

Please select boot mode as below figures.

 Currently we support 2 modes in 4 modes: SCIF Download mode and QSPI Boot mode. eSD Boot mode will be supported by the future update.

SCIF Download Mode(*1)	QSPI Boot (1.8V) Mode																
 <table> <tr><td>SW11-1</td><td>OFF</td></tr> <tr><td>SW11-2</td><td>ON</td></tr> <tr><td>SW11-3</td><td>OFF</td></tr> <tr><td>SW11-4</td><td>ON</td></tr> </table>	SW11-1	OFF	SW11-2	ON	SW11-3	OFF	SW11-4	ON	 <table> <tr><td>SW11-1</td><td>OFF</td></tr> <tr><td>SW11-2</td><td>OFF</td></tr> <tr><td>SW11-3</td><td>OFF</td></tr> <tr><td>SW11-4</td><td>ON</td></tr> </table>	SW11-1	OFF	SW11-2	OFF	SW11-3	OFF	SW11-4	ON
SW11-1	OFF																
SW11-2	ON																
SW11-3	OFF																
SW11-4	ON																
SW11-1	OFF																
SW11-2	OFF																
SW11-3	OFF																
SW11-4	ON																
eMMC Boot (1.8V) Mode	eSD Boot Mode																
 <table> <tr><td>SW11-1</td><td>ON</td></tr> <tr><td>SW11-2</td><td>OFF</td></tr> <tr><td>SW11-3</td><td>OFF</td></tr> <tr><td>SW11-4</td><td>ON</td></tr> </table>	SW11-1	ON	SW11-2	OFF	SW11-3	OFF	SW11-4	ON	 <table> <tr><td>SW11-1</td><td>ON</td></tr> <tr><td>SW11-2</td><td>ON</td></tr> <tr><td>SW11-3</td><td>OFF</td></tr> <tr><td>SW11-4</td><td>ON</td></tr> </table>	SW11-1	ON	SW11-2	ON	SW11-3	OFF	SW11-4	ON
SW11-1	ON																
SW11-2	OFF																
SW11-3	OFF																
SW11-4	ON																
SW11-1	ON																
SW11-2	ON																
SW11-3	OFF																
SW11-4	ON																

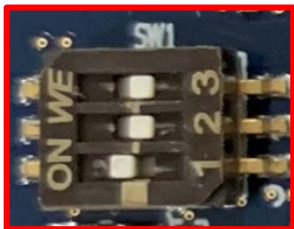
Please select input voltage setting as below.

SW11-4	Input voltage selection
OFF	Input 9V
ON	Input 5V

4.1.2 How to set SW1

Please set the SW1 settings to follows.

- The SW1-1 is used to select the JTAG debug mode or not.  
JTAG is not used to in the procedure for booting up RZ/Five Group Board Support Package, so please set SW1-1 to normal operation mode.
- The SW1-2 is used to select the eMMC or microSD mode. Please set SW1-2 to eMMC mode.
- The SW1-3 is used to select the Ethernet or multiple functions mode. The multiple functions are enabled CAN0, CAN1, PMOD0 and PMOD1. Please set SW1-3 to other function mode.



SW1-3	OFF
SW1-2	OFF
SW1-1	ON

SW1-1	DEBUGEN
OFF	JTAG debug mode
ON	Normal operation

SW1-2	SD/MMC selection
OFF	Select eMMC memory
ON	Select microSD card

The selection of microSD slot and eMMC on the SMARC module is exclusive

SW1-3	Ether0/CAN0, CAN1, SSI1, RSP11 selection
OFF	CAN0, CAN1, PMOD0 Type-2A, PMOD Type-6
ON	Ether0

4.1.3 How to use debug serial (console ouput)

Please connect USB Type-micro-B cable to CN14.

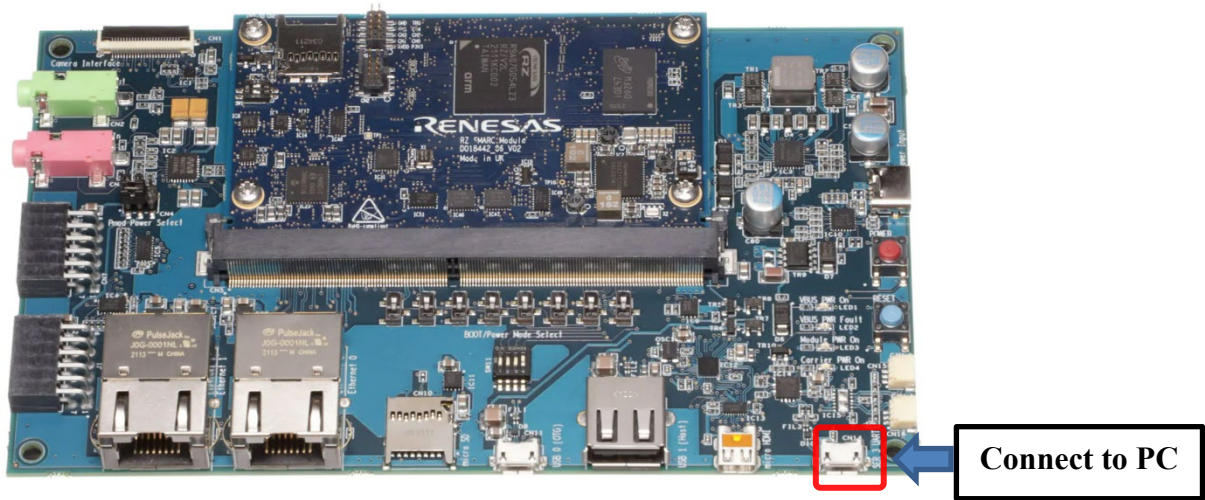


Figure 3. Connecting console for debug



## Startup Procedure

## 4.1.4 Power supply

1. Connect USB-PD Power Charger to USB Type-C Connector (CN6).
2. LED1(VBUS Power ON) and LED3 (Module PWR On) lights up.

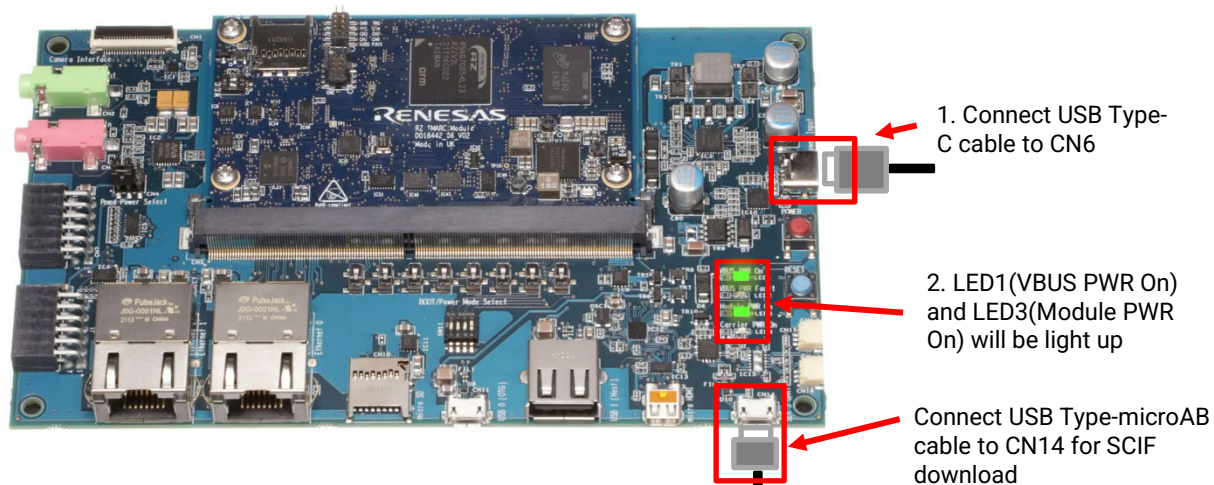


Figure 4. Connecting Power Supply

3. Press the power button(SW9) to turn on the power.  
*Note: When turn on the power, press and hold the power button for 1 second.*  
*When turn off the power, press and hold the power button for 2 seconds*
4. LED4(Carrier PWR On) lights up.

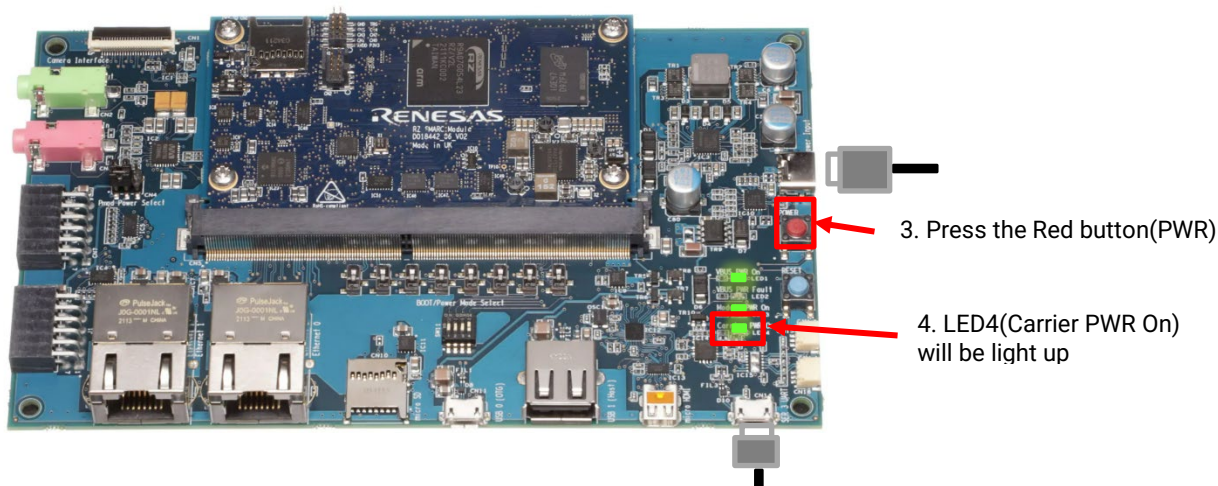


Figure 5. Power ON

#### 4.1.5 Building files to write

The evaluation boards use the files in the Table 8 as the boot loaders. The boot loaders files are generated by the build instruction in the section 2. Please refer to the Table 3. Please copy these files to a PC which runs serial terminal software.

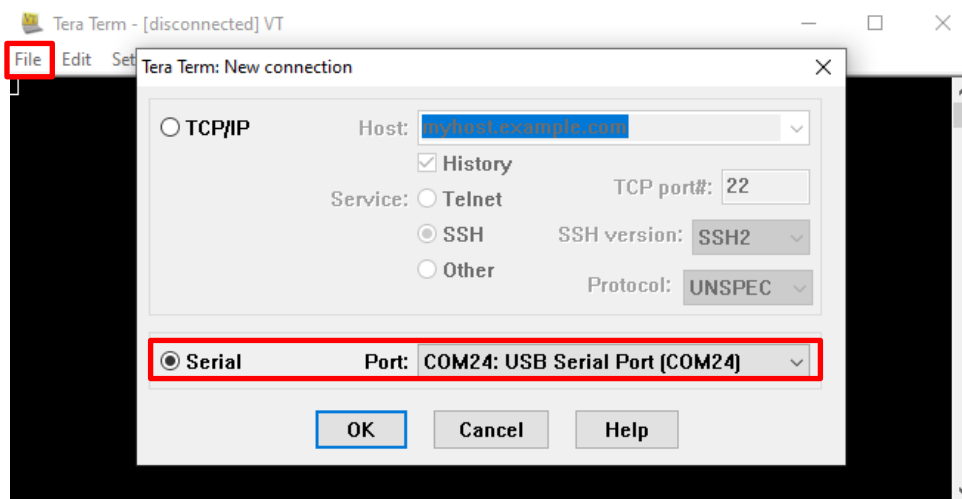
**Table 8. File names of Boot loader**

Board	File name of Boot loader
RZ/Five Evaluation Board Kit	<ul style="list-style-type: none"> <li>fit-smarc-rzfive.srec</li> <li>spl-smarc-rzfive.srec</li> </ul>

#### 4.1.6 Settings

Connect between the board and a control PC by USB serial cable.

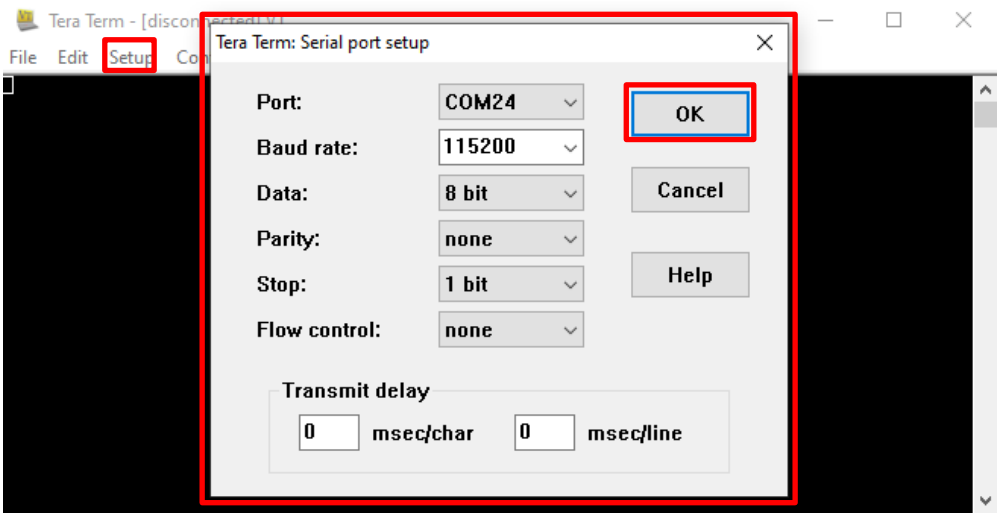
- (1) Bring up the terminal software and select the “File” > “New Connection” to set the connection on the software.



- (2) Select the “Setup” > “Serial port” to set the settings about serial communication protocol on the software.

Set the settings about serial communication protocol on a terminal software as below:

- Speed: 115200 bps
- Data: 8bit
- Parity: None
- Stop bit: 1bit
- Flow control: None



(3) To set the board to SCIF Download mode, set the SW11 as below:

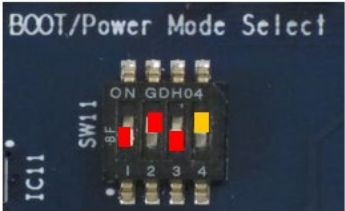
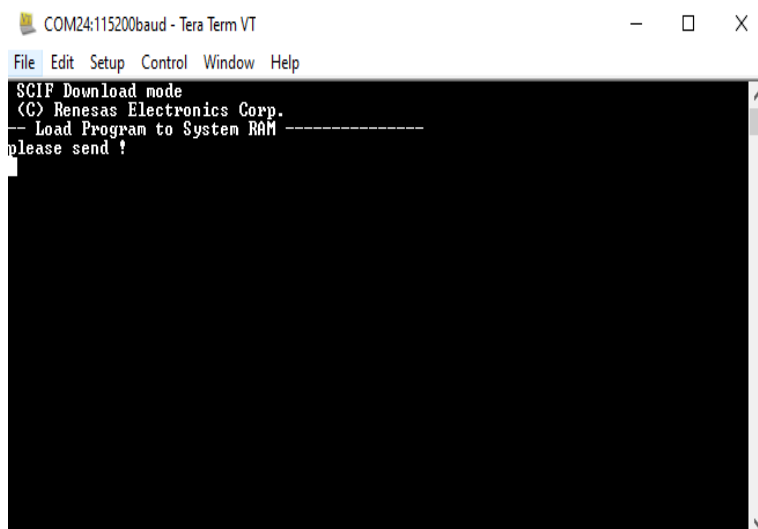
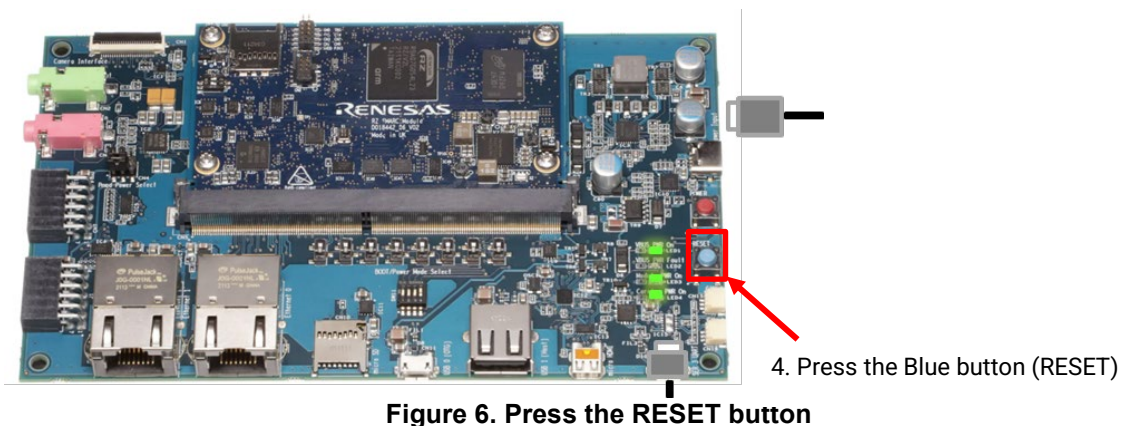


Table 9. SW11

1	2	3	4
OFF	ON	OFF	ON



- (4) After finished all settings, when pressed the reset button SW10, the messages below are displayed on the terminal.



## 4.2 Download Flash Writer to RAM

Turn on the power of the board by pressing SW9. The messages below are shown on the terminal.

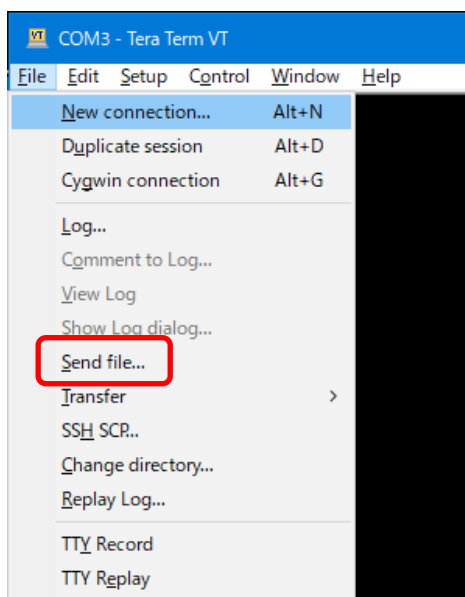
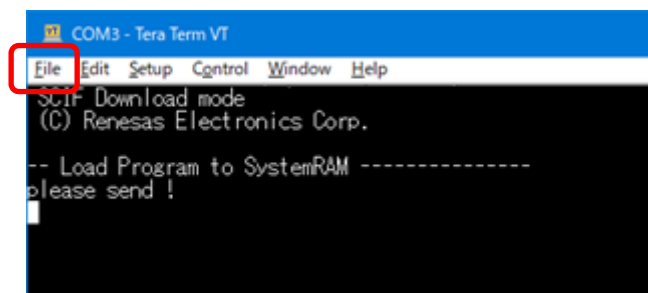
```
SCIF Download mode
(C) Renesas Electronics Corp.

-- Load Program to SystemRAM -----
please send !
```

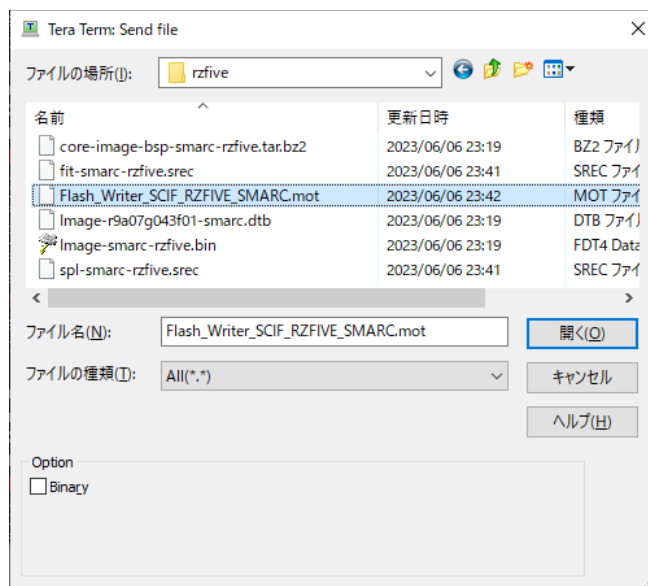
Send an image of Flash Writer (Flash\_Writer\_SCIF\_RZFIVE\_SMARC.mot) using terminal software after the message “please send !” is shown.

Below is a sample procedure with Tera Term.

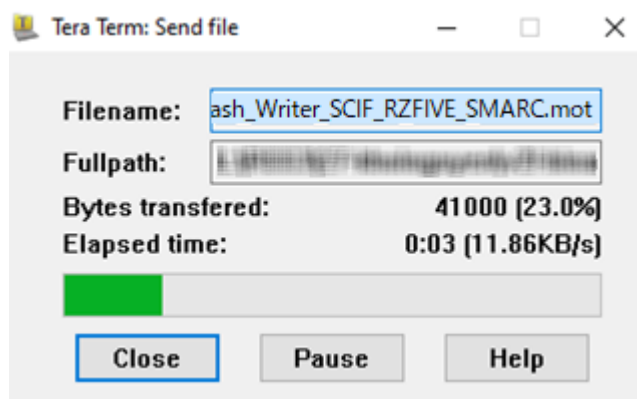
Open a “Send file” dialog by selecting “File” → “Sendfile” menu.



Then, select the image to be send and click “Open” button.



The image will be sent to the board via serial connection.



After successfully downloading the binary, Flash Writer starts automatically and shows a message like below on the terminal.

```
Flash writer for RZ/Five Series V1.02 Nov.15,2021
Product Code : RZ/Five
>
```

### 4.3 Write the Bootloader

For the boot operation, two boot loader files need to be written to the target board. Corresponding bootloader files and specified address information are depending on each target board as described in **Table 10**.

“XLS2” command of Flash Writer is used to write boot loader binary files. This command receives binary data from the serial port and writes the data to a specified address of the Flash ROM with information where the data should be loaded on the address of the main memory.

For example, this part describes how to write boot loader files in the case of RZ/Five Evaluation Board Kit PMIC version.:

```
>XLS2
===== Qspi writing of RZ/Five Board Command =====
Load Program to Spiflash
Writes to any of SPI address.
  Micron : MT25QU512
Program Top Address & Qspi Save Address
===== Please Input Program Top Address =====
  Please Input : H'11E00

===== Please Input Qspi Save Address ===
  Please Input : H'00000
Work RAM(H'50000000-H'53FFFFFF) Clear....
please send ! ( '.' & CR stop load)
```

Send the data of “spl-smarc-rzfive.srec” from terminal software after the message “please send !” is shown.

After successfully download the binary, messages like below are shown on the terminal.

```
SPI Data Clear(H'FF) Check :H'00000000-0000FFFF Erasing..Erase Completed
SAVE SPI-FLASH.....
===== Qspi Save Information =====
  SpiFlashMemory Stat Address : H'00000000
  SpiFlashMemory End Address  : H'00009A80
=====
```

```
SPI Data Clear(H'FF) Check : H'00000000-0000FFFF,Clear OK?(y/n)
```

In case a message to prompt to clear data like above, please enter “y”.

Next, write another loader file by using XLS2 command again.

```
>XLS2
===== Qspi writing of RZ/Five Board Command =====
Load Program to Spiflash
Writes to any of SPI address.
  Micron : MT25QU512
Program Top Address & Qspi Save Address
===== Please Input Program Top Address =====
  Please Input : H'00000

===== Please Input Qspi Save Address ===
  Please Input : H'20000
Work RAM(H'50000000-H'53FFFFFF) Clear....
please send ! ( '.' & CR stop load)
```

Send the data of “fit-smarc-rzfive.srec” from terminal software after the message “please send !” is shown.

After successfully download the binary, messages like below are shown on the terminal.

```
SPI Data Clear(H'FF) Check :H'00000000-0000FFFF Erasing..Erase Completed
SAVE SPI-FLASH.....
===== Qspi Save Information =====
SpiFlashMemory Stat Address : H'0001D200
SpiFlashMemory End Address  : H'000CC73F
=====
```

```
SPI Data Clear(H'FF) Check : H'00000000-0000FFFF,Clear OK?(y/n)
```

In case a message to prompt to clear data like above, please enter “y”.

After writing two loader files normally, turn off the power of the board by changing the SW11.

**Table 10. Address for sending each loader binary file**

File name	Address to load to RAM	Address to save to ROM
spl-smarc-rzfive.srec	11E00	00000
fit-smarc-rzfive.srec	00000	20000

## 4.4 Change Back to Normal Boot Mode

To set the board to SPI Boot mode, set the SW11 as below:

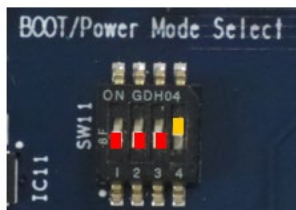
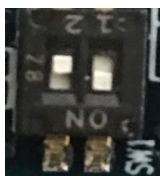


Table 11. SW11

1	2	3	4
OFF	OFF	OFF	ON

Note:-

Set the SW1 on SoM module to eMMC mode.



Turn on the power of the board by pressing the reset button SW10.

```
U-Boot 2021.10 (Dec 20 2022 - 07:08:13 +0000)
```

```
CPU:   rv64imafdc
Model: smarc-rzf
DRAM:  896 MiB
SW_ET0_EN: OFF
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from MMC... OK
In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
Net:
Error: ethernet@11c30000 address not set.
No ethernet found.
```

```
Hit any key to stop autoboot:  0
=>
```

Following the messages above, many warning messages will be shown. These warnings are eliminated by setting correct environment variables. Please set default value and save them to the Flash ROM.

```
=> env default -a
## Resetting to default environment
=> saveenv
Saving Environment to MMC... Writing to MMC(0)...OK
=>
```

If you created the microSD card according to the **step 3.2**, perform the following settings. If you created the microSD card according to the **step 3.1**, this setting is not required.

In case booting from microSD card on SMARC carrier board, set environment variables using the commands below. The commands below are for the RZ/Five board. Please replace the file names in “bootcmd” according to the Release Note when you use other boards.

```
=> setenv bootargs 'root=/dev/mmcblk1p2 rootwait'
=> setenv bootcmd 'mmc dev 1;fatload mmc 1:1 0x48080000 Image-smarc-rzfive.bin; fatload mmc 1:1 0x48000000 Image-r9a07g043f01-smarc.dtb; booti 0x48080000 - 0x48000000'
=> saveenv
Saving Environment to MMC... Writing to MMC(0)...OK
```

```
bootargs:      'root=/dev/mmcblk1p2 rootwait'
```

```
              root filesystem is partition 2 of block 1 on microSD card.
```

```
bootcmd:      'mmc dev 1;fatload mmc 1:1 0x48080000 <Linux kernel>;
              fatload mmc 1:1 0x48000000 <Device tree>;
              booti 0x48080000 - 0x48000000'
```

Note) The setting above assumes the microSD card has two partitions and stores data as below:

**First partition:** formatted as FAT, includes Image-smarc-rzfive.bin and Image-r9a07g054l2-smarc.dtb

**Second partition:** formatted as ext4, rootfs image is expanded

## 5. Booting and Running Linux

Set microSD card to slot on carry board.

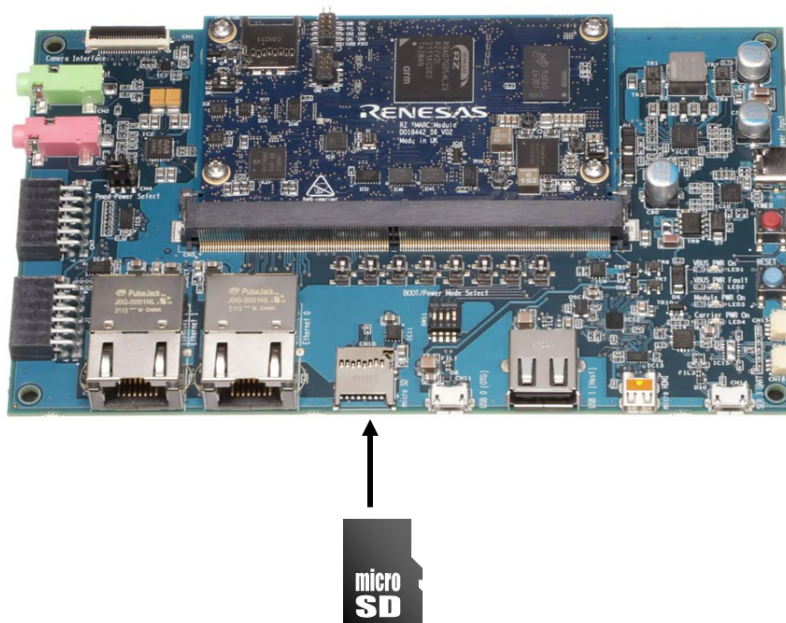


Figure 7. Set microSD card to SMARC-EVK

Now the board can bootup normally. Please turn off and on the power again to boot up the board.

### 5.1 Power on the board and Startup Linux

After obtaining your reference board, please be sure to follow the document and write the bootloaders to the Flash ROM before starting the evaluation.

Before booting the board, please be sure to confirm the bootloaders which are built with your BSP/VLP are written to your board.

```
U-Boot 2021.10 (Dec 20 2022 - 07:08:13 +0000)

CPU:   rv64imafdc
Model: smarc-rzf
DRAM:  896 MiB
SW_ET0_EN: OFF
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from MMC... OK
In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
Net:

Error: ethernet@11c30000 address not set.
No ethernet found.

Hit any key to stop autoboot: 0
17666560 bytes read in 1845 ms (9.1 MiB/s)
22811 bytes read in 7 ms (3.1 MiB/s)
Moving Image from 0x48080000 to 0x48200000, end=49334000
## Flattened Device Tree blob at 48000000
```



```
Booting using the fdt blob at 0x48000000
Loading Device Tree to 0000000057ff7000, end 0000000057fff91a ... OK

Starting kernel ...

[ 0.000000] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
[ 0.000000] Linux version 5.10.83-cip1-yocto-standard (oe-user@oe-host) (aarc
:
:
oky (Yocto Project Reference Distro) 3.1.17 smarc-rzf5 ttySC0

BSP: RZFive/RZFive-SMARC-EVK/3.0.2
LSI: RZFive
Version: 3.0.2
smarc-rzf5 login: root
[ 40.652896] audit: type=1006 audit(1600598675.179:2): pid=158 uid=0 old-auid=429496
7295 auid=0 tty=(none) old-ses=4294967295 ses=1 res=1
root@smarc-rzf5:~#
```

## 5.2 Shutdown the Board

To power down the system, follow the step below.

Step 1. Run shutdown command

Run shutdown command on the console as below. After that, the shutdown sequence will start.

```
root@smarc-rzf5:~# shutdown -h now
```

Note: Run this command during the power-off sequence on rootfs.

Step 2. Confirm the power-off

After executing the shutdown command, confirm that LED302, LED304, and D305 are off.

Step 3. Turn off the power switch on the board

After checking the above LEDs, turn SW303 off.

## 6. Building the SDK

To build Software Development Kit (SDK), run the commands below after the steps (1) – (5) of the section 2 are finished.

The SDK allows you to build custom applications outside of the Yocto environment, even on a completely different PC. The results of the commands below are ‘installer’ that you will use to install the SDK on the same PC, or a completely different PC.

For building general applications:

```
$ cd ~/rzfive_vlp_<package version>/build
$ MACHINE=smarc-rzfive bitbake core-image-minimal -c populate_sdk
```

Or

```
$ cd ~/rzfive_vlp_<package version>/build
$ MACHINE=smarc-rzfive bitbake core-image-bsp -c populate_sdk
```

The resulting SDK installer will be located in **build/tmp/deploy/sdk/**

The SDK installer will have the extension .sh

To run the installer, you would execute the following command:

```
$ sudo sh poky-glibc-x86_64-core-image-bsp-riscv64-smarc-rzfive-toolchain-<version>.sh
```

Note) The SDK build may fail depending on the build environment. At that time, please run the build again after a period of time. Or build it again from scratch with the below commands.

```
$ cd ~/rzfive_vlp_<package version>/build
$ MACHINE=smarc-rzfive bitbake core-image-bsp -c cleanall
$ MACHINE=smarc-rzfive bitbake core-image-bsp
```

For building general applications:

```
$ MACHINE=smarc-rzfive bitbake core-image-bsp -c populate_sdk
```

## 7. Application Building and Running

This chapter explains how to make and run an application for RZ/Five with this package.

### 7.1 Make an application

Here is an example of how to make an application running on VLP. The following steps will generate the “Hello World” sample application.

Note that you must build (bitbake) a core image for the target and prepare SDK before making an application. Refer to the start-up guide on how to make SDK.

#### 7.1.1 How to extract SDK

Step 1. Install toolchain on a Host PC:

```
$ sudo sh ./poky-glibc-x86_64-core-image-bsp-riscv64-smarc-rzf5-toolchain-\  
<version>.sh
```

Note:

sudo is optional in case user wants to extract SDK into a restricted directory (such as: /opt/).

If the installation is successful, the following messages will appear:

```
$ sudo sh ./rzf5_vlp_<package version>/build/tmp/deploy/sdk/poky-glibc-x86_64-core-i  
mage-bsp-aarch64-smarc-rzf5-toolchain-<version>.sh  
Poky (Yocto Project Reference Distro) SDK installer version 3.1.26  
=====
```

Enter target directory for SDK (default: /opt/poky/3.1.26): ~/workspace/rzf5/vlpf3.0.5-build-test/build/tmp/deploy/sdk/temp

You are about to install the SDK to "/home/renesas/workspace/rzf5/vlpf3.0.5-build-test/build/tmp/deploy/sdk/temp". Proceed [Y/n]? Y

Extracting SDK.....

.....done

Setting it up...done

SDK has been successfully set up and is ready to be used.

Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.

```
$ ./opt/poky/3.1.26/environment-setup-riscv64-poky-linux
```

Step 2. Set up cross-compile environment:

```
$ source /<Location in which SDK is extracted>/environment-setup-riscv64-poky-linux
```

Note:

User needs to run the above command once for each login session.

```
$ source /opt/poky/3.1.26/environment-setup-riscv64-poky-linux
```

#### 7.1.2 How to build Linux application

Step 1. Go to linux-helloworld directory:

```
$ cd $WORK/linux-helloworld
```

Step 2. Cross-compile:

```
$ make
```

Step 3. Copy all files inside this directory to /usr/share directory on the target board:

```
$ scp -r $WORK/linux-helloworld/<username>@<board IP>:/usr/share/
```

Step 4. Run the application:

```
# /usr/share/linux-helloworld/linux-helloworld
```

How to extract SDK

Step 1. Make a work directory for the application on the Linux host PC.

```
$ mkdir ~/linux-helloworld
$ cd ~/linux-helloworld
```

Step 2. Make the following three files (an application file, Makefile, and configure file) in the directory for the application.

Here, the application is made by automake and autoconf.

- main.c

```
#include <stdio.h>
/* Display "Hello World" text on terminal software */
int main(int argc, char** argv)
{
    printf("\nHello World\n");
    return 0;
}
```

- Makefile

```
APP = linux-helloworld
SRC = main.c

all: $(APP)

CC ?= gcc

# Options for development
CFLAGS = -g -O0 -Wall -DDEBUG_LOG

$(APP):
    $(CC) -o $(APP) $(SRC) $(CFLAGS)

install:
    install -D -m755 $(APP) $(DESTDIR)/home/root/$(APP)

clean:
    rm -rf $(APP)
```

Step 3. Make the application by the generated makefile.

```
$ make
```

```
$ make
riscv64-poky-linux-gcc      -fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-security --sysroot=/opt/poky/3.1.26/sysroots/riscv64-poky-linux -o linux-helloworld main.c -g -O0 -Wall -DDEBUG_LOG
```

After making, confirm that the execute application (the sample file name is “hello”) is generated in the hello\_apl folder.

Store a sample application

The sample application could be written by the following procedure. The application should be stored in the ext3 partition.

```
$ sudo mount /dev/sdb2 /media/  
$ cd /media/usr/bin  
$ sudo cp /linux-helloworld .  
$ sudo chmod +x linux-helloworld
```

Notes: 1. “sdb2” (above in red) may depend on using system.  
2. is an optional directory name to store the application.

## 7.2 Run a sample application

Power on the RZ/Five Evaluation Board Kit and start the system. After booting, run the sample application with the following command.

```
BSP: RZFIVE/RZFIVE-SMARC-EVK/<package version>  
LSI: RZFIVE  
Version: <package version>  
smarc-rzf5 login: root  
root@smarc-rzf5:~# ls  
linux-helloworld v4l2-init.sh  
root@smarc-rzf5:~# ./linux-helloworld  
  
Hello World  
root@smarc-rzf5:~#
```

Note: Refer to the start-up guide for the method of how to boot the board and system.

## 8. Appendix

### 8.1 Preparing Flash Writer

Flash Writer is built automatically when building BSP by bitbake command. Please refer to the Release Note of the RZ/Five BSP to obtain a binary file of Flash Writer.

If you need latest one, please get source code from the GitHub repository and build it according to the following instructions. In general, new revision of reference boards requires latest Flash Writer.

#### 8.1.1 Preparing cross compiler

FlashWriter runs on target boards. Please get cross compiler built by GNU toolchain or setup a Yocto SDK.

- **GNU toolchain:**

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
$ cd riscv-gnu-toolchain
$ ./configure --prefix=/opt/riscv
$ sudo make linux -j
```

- **Yocto SDK:**

Build an SDK according to Release Notes and install it to a Linux Host PC. Then, enable the SDK as below.

```
$ source /usr/local/oe-core-x86_64/environment-setup-riscv64-oe-linux
```

#### 8.1.2 Building Flash Writer

Get source codes of Flash Writer from the GitHub repository and checkout the branch rz\_five.

```
$ cd ~/
$ git clone https://github.com/renesas-rz/rzg2_flash_writer.git
$ cd rzg2_flash_writer
$ git checkout rz_five
```

Build Flash Writer as an s-record file by the following commands. Please specify a target board by “BOARD” option.

```
$ export PATH=$PATH:/opt/riscv/bin
$ make clean
$ make BOARD=RZFIVE_SMARC
```

After above steps, “Flash\_Writer\_SCIF\_RZFIVE\_SMARC.mot” is generated.

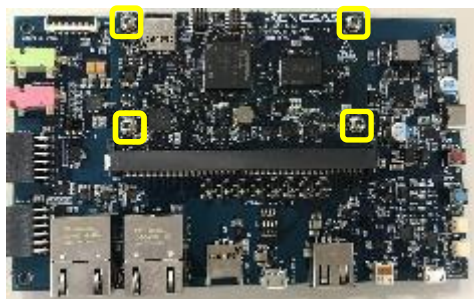
## 8.2 How to replace the SMARC Module Board

Please be careful when replacing the board as follows.

1. Remove the four screws.

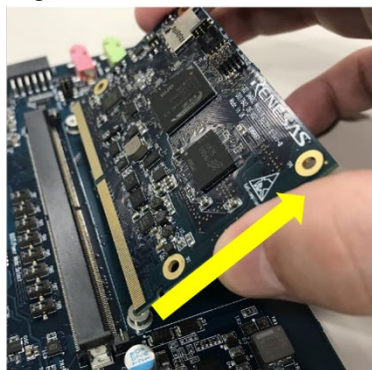
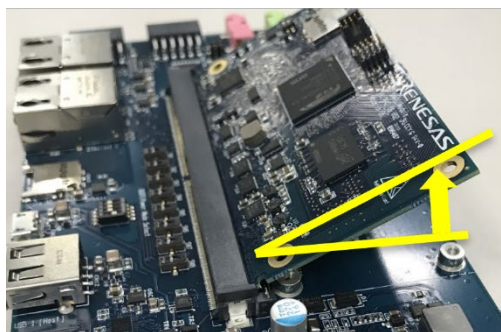
*Note: The screw thread is a special shape, so be careful not to crush the screw thread.*

Please recommend to prepare a torx screwdriver which is a "T6" head size.

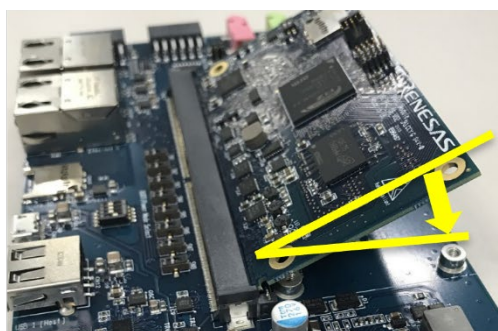
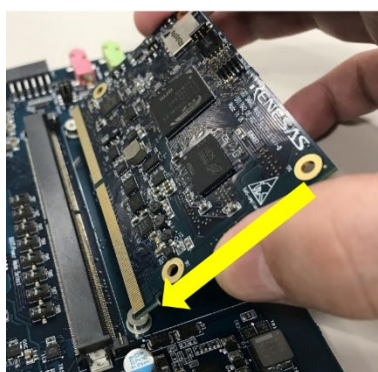


Specially shaped screw threads

2. Remove the screw and the board will stand up at an angle. Slide it out.



3. Insert the replacement the board diagonally, then roll the SMARC board parallel to the board and fix it with screws.



### 8.3 Device drivers

The following drivers are supported: For detail information on how to use, please refer to these documents in the BSP manual set.

File	Description
RTK0EF0045Z9006AZJ-v3.0.x.zip	BSP Manual Set for RZ/Five.

Note) “x” is the version of the file. Please refer to the latest one.

**Table 12. Support device drivers**

Device Driver	Documents
Kernel Core	r01us0468ej0xxx-rz-g_Kernel_Core_UME.pdf
Direct Memory Access Controller (DMAC)	r01us0480ej0xxx-rz-g_DMAMC_UME.pdf
Multi-function Timer Unit 3a (MTU3a)	r01us0476ej0xxx-rz-g_MTU3a_UME.pdf
Port Output Enable 3 (POE3)	r01us0552ej0xxx-rz-g_POE3_UME.pdf
Watchdog Timer (WDT)	r01us0479ej0xxx-rz-g_WDT_UME.pdf
Serial Communication Interface with FIFO A (SCIFA)	r01us0483ej0xxx-rz-g_SCIFA_UME.pdf
Renesas Serial Peripheral Interface (RSPI)	r01us0481ej0xxx-rz-g_SPI_UME.pdf
SPI Multi IO Bus Controller	r01us0482ej0xxx-rz-g_SPI_Multi IO_UME.pdf
I2C Bus Interface	r01us0477ej0xxx-rz-g_I2C_UME.pdf
Serial Sound Interface	r01us0490ej0xxx-rz-g_SSI_UME.pdf
RS-CANFD Interface	r01us0478ej0xxx-rz-g_CANFD_UME.pdf
Gigabit Ethernet Interface	r01us0475ej0xxx-rz-g_Gigabit_Ethernet_UME.pdf
A/D Converter	r01us0487ej0xxx-rz-g_AD_Converter_UME.pdf
USB 2.0 Host	r01us0485ej0xxx-rz-g_USB2.0_Host_UME.pdf
USB 2.0 Function	r01us0491ej0xxx-rz-g_USB2.0_Function_UME.pdf
SD/MMC Host Interface	r01us0474ej0xxx-rz-g_SD_MMC_UME.pdf
GPIO	r01us0488ej0xxx-rz-g_GPIO_UME.pdf
Power Management	r01us0605ej0xxx-rz-g_Power_Management_UME.pdf
Thermal Sensor Unit (TSU)	r01us0486ej0xxx-rz-g_Thermal_Sensor_UME.pdf
Error Correction Code (ECC)	r01us0647ej0xxx-rz-mpu_ECC_UME.pdf

Note) “xxx” is the revision of the file. Please refer to the latest one.



## 9. Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jun. 30, 2023	-	First edition issued.

**Website and Support**

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/contact/>

All trademarks and registered trademarks are the property of their respective owners.