SMARC EVK of RZ/Five

Linux Start-up Guide

R01US0618EJ0106
Rev.1.06
Mar. 31, 2025

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 3.0.7 or later. (hereinafter referred to as "VLP/F")

Contents

1. Environment Requirement

The environment for building the Verified Linux Package (hereinafter referred to as "VLP") is listed in the Table 1. Please refer to the documents below for details about setting up the environment.

A Linux PC is required for building the software.

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

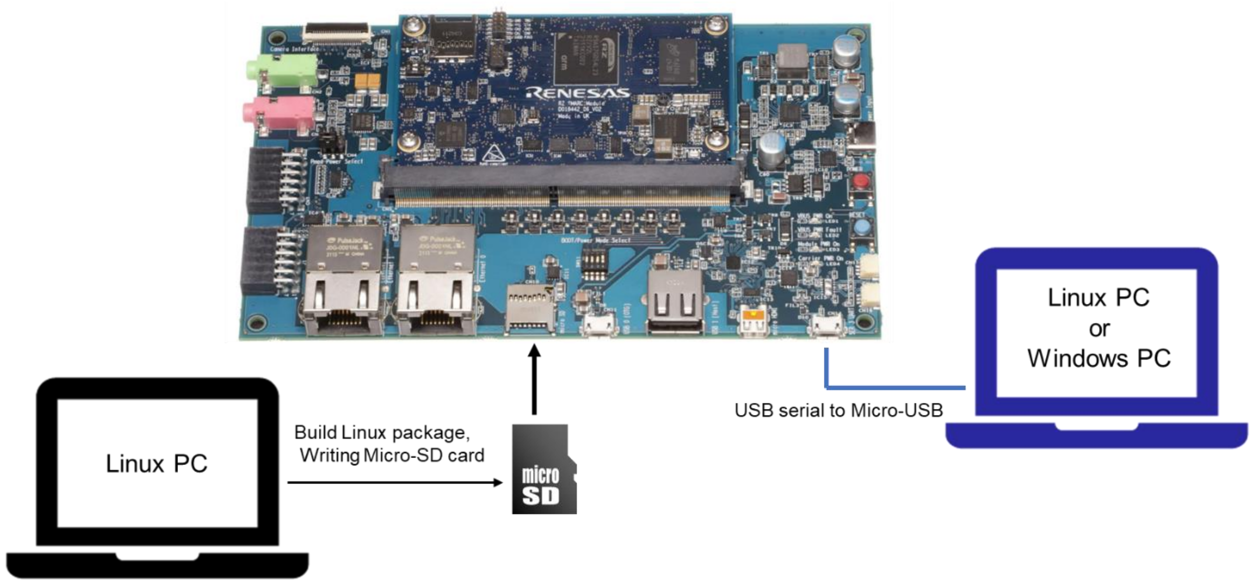


Figure 1 Recommend environment

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	Ubuntu 22.04 LTS 64 bit OS must be used. 22.04 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 controling serial console of the target board Tera Term (latest version) is recommended Available at Releases TeraTermProject/teraterm(github.com)
	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: http://www.ftdichip.com/Drivers/VCP.htm
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. Note that use a micro-SDHC card for the flash writer.

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 Required Host OS

⚠ The VLP/F is only built in **Ubuntu 22.04**

Ubuntu 22.04 is required to build the VLP/F. This is because it was the only host operating system tested and is a specific requirement for Yocto 3.1 (dunfell). Using Ubuntu 24.04 is not supported.

2.2 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 VLP.

```
$ sudo 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-ping libsdl1.2-dev xterm p7zip-full libyaml-dev libssl-dev \
bmap-tools
```

Please refer to the URL below for detailed information:

- https://docs.yoctoproject.org/3.1.33/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.

```
$ PACKAGE_VERSION=v3.0.7
$ 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 building environment must have 100GB of free hard drive space in order to complete the minimum build. The Yocto VLP 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.

Note) `${PACKAGE_VERSION}: e.g v3.0.7`

(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 commands below 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-filesystems
$ 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 <image name>
```

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

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

After the building 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 s
ucceeded.
```

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

RZ/Five	**Linux kernel**	Image-smarc-rzfive.bin
	Device tree file	Image-r9a07g043f01-smarc.dtb
	root filesystem	*<image name>*-smarc-rzfive.tar.bz2
	Boot loader	• fit-smarc-rzfive.srec • spl-smarc-rzfive.srec
	Flash Writer	Flash_Writer_SCIF_RZFIVE_SMARC.mot
	SD image (wic)	*<image name>* -smarc-rzfive.wic.gz *<image name>* -smarc-rzfive.wic.bmap

2.3 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+"
```

If you want to change this setting after completing step (5) of section 2.2, create a new working directory and prepare the new building environment. Note that not doing this may cause a build error.

(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.
If you want to change this setting after completing step (5) of section 2.2, create a new working directory and prepare the new building environment. Note that not doing this may cause a build error.

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 betwwen 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 on them 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	9.5.0
glib-2.0	2.58.3	2.62.6
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

(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 8.6.

(4) WIC image

The name “WIC” is derived from OpenEmbedded Image Creator (oeic). It includes images that system can boot it in hardware device.
WIC is supported and below guidelines show 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.

SPDX is supported and below guidelines show 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.

<u>**Note**</u>: There is an issue when building SDK (example bitbake core-image-weston -c populate_sdk) with SBoM SPDX 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_rzfive-poky-linux/core-image-w
eston/1.0-r0/spdx/sdk-work/poky-glibc-x86_64-core-image-weston-aarch64-smarc-rzfive-
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}"
```

(6) Real time performance

If you want to use the kernel which improves the real-time performance, please add the line below to the `local.conf`.

```
IS_RT_BSP="1"
```

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: 0xxxxxxxxxx
$ 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 Table 3.

When you complete the above commands, the micro SD card is prepared correctly. Please skip 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 2.3(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 with 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 the following table.

Table 6. Partitions of microSD card

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

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): (Push the enter key)
First sector (2048-62333951, default 2048): (Push the enter key)
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): (P
ush 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, expand the roots 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 nemes 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		FAT32	
Primary #2	Size specified when the partition was created.	Ext4	./ ├──── bin ├──── boot ├──Image └──r9a07g043f01-smarc.dtb ├──── 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).

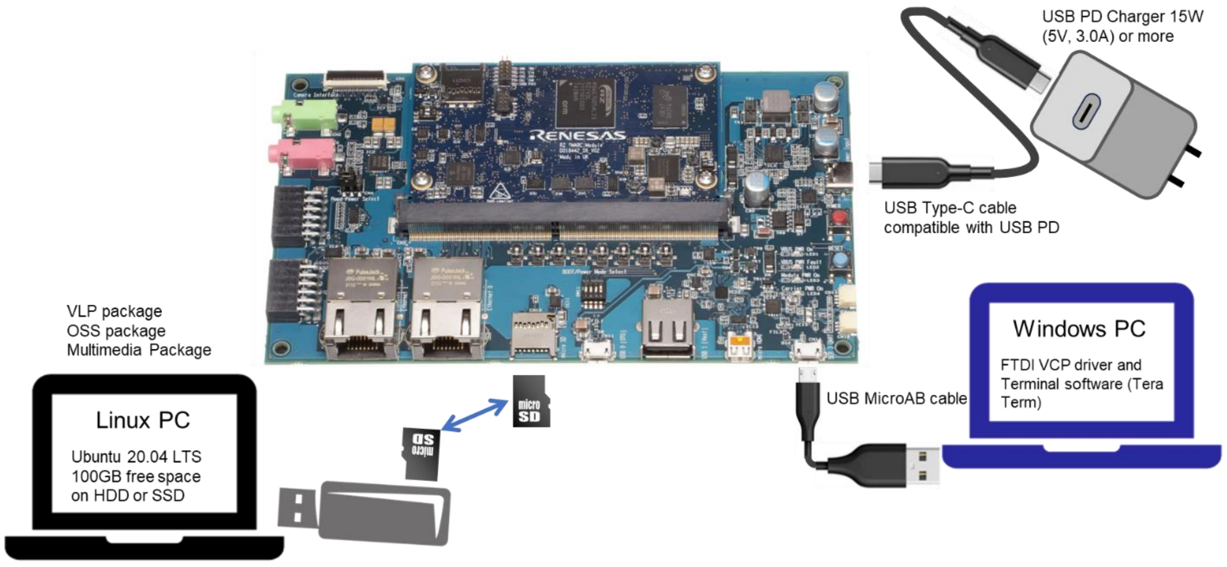
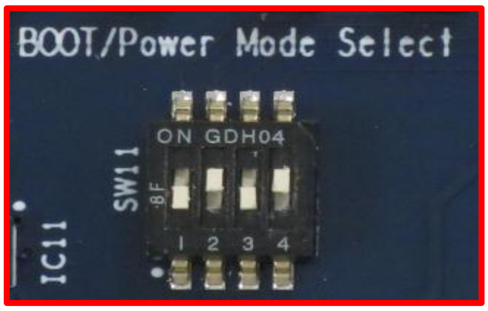


Figure 2. Operating environment

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 the 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)

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

QSPI Boot (1.8V) Mode

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

eMMC Boot (1.8V) Mode

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

eSD Boot Mode

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

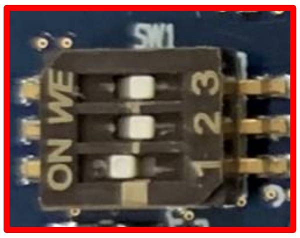
Please select input voltage setting as below.

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.

- SW1-1 is used to select the JTAG debug mode or not.
  JTAG is not used 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, RSPI1 selection
OFF	CAN0, CAN1, PMOD0 Type-2A, PMOD Type-6
ON	Ether0

4.1.3 How to use debug serial (console ouput)

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

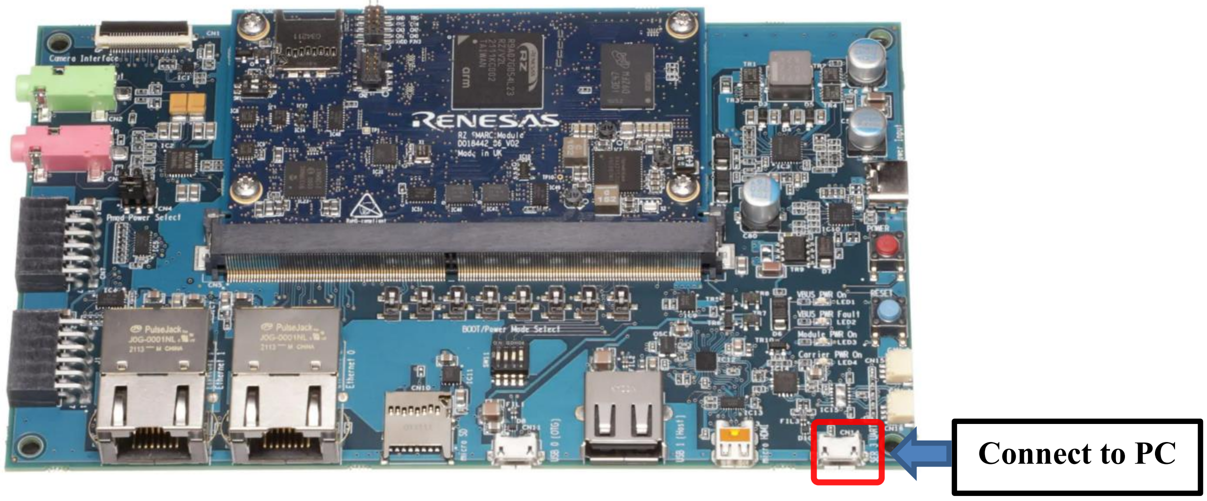


Figure 3. Connecting console for debug

4.2 Startup Procedure

This section describes how to write Flash writer and bootloaders using Windows PC. For describes how to write using Ubuntu PC, please refer to 8.5.

4.2.1 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.

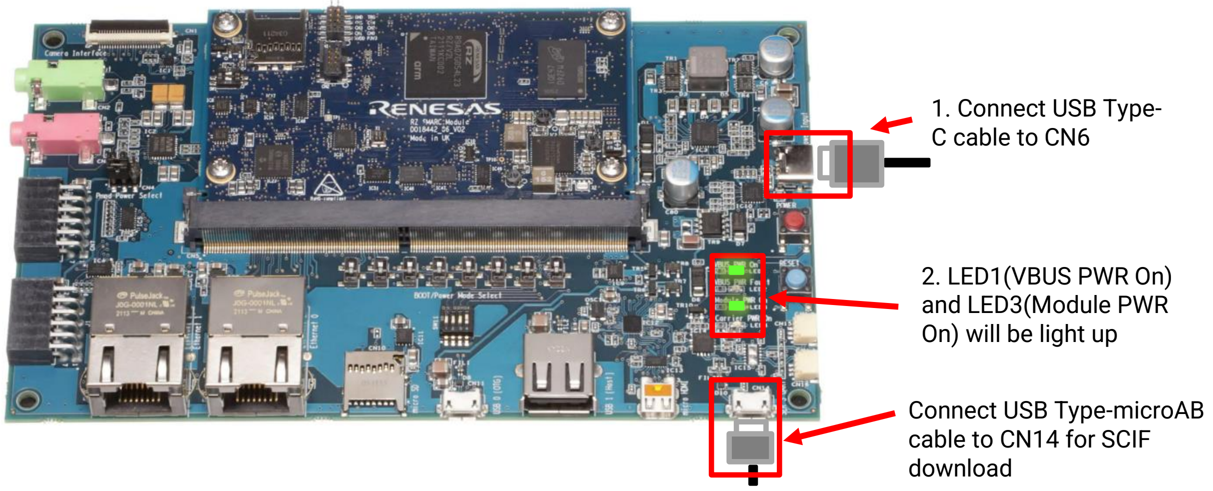

1. Connect USB Type-C cable to CN6
2. LED1(VBUS PWR On) and LED3(Module PWR On) will be light up
Connect USB Type-microAB cable to CN14 for SCIF download

Figure 4. Connecting Power Supply

3. Press the power button(SW9) to turn on the power.
   Note: When turning 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.

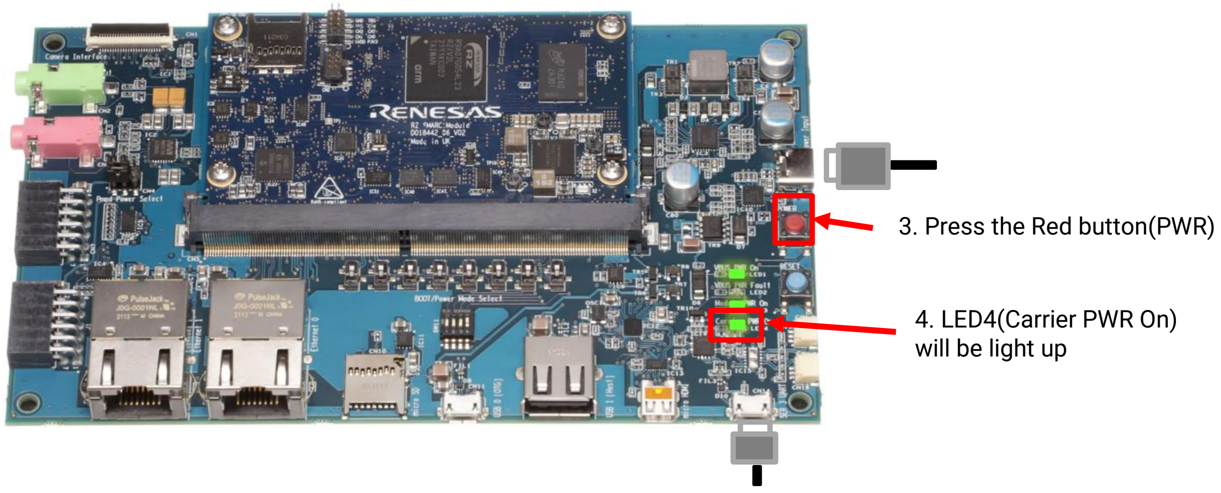

3. Press the Red button(PWR)
4. LED4(Carrier PWR On) will be light up

Figure 5. Power ON

4.2.2 Building files to write

The evaluation boards use the files in the Table 8 as boot loaders. The boot loaders files are generated by the build instruction in the section 2.2. Please refer to the Table 3. Please copy these files to a PC which runs serial terminal software.

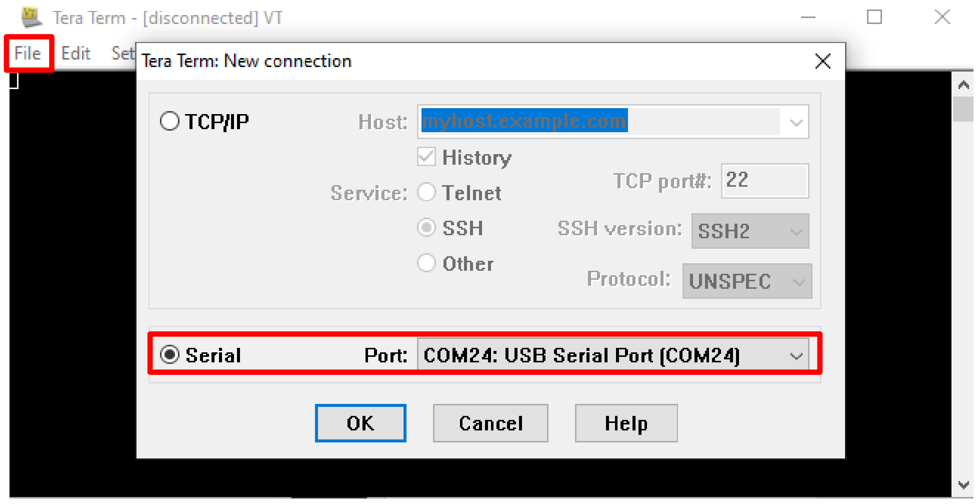
Table 8. File names of Boot loader

Board	File name of Boot loader
RZ/Five Evaluation Board Kit	• fit-smarc-rzfive.srec • spl-smarc-rzfive.srec

4.2.3 Settings

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

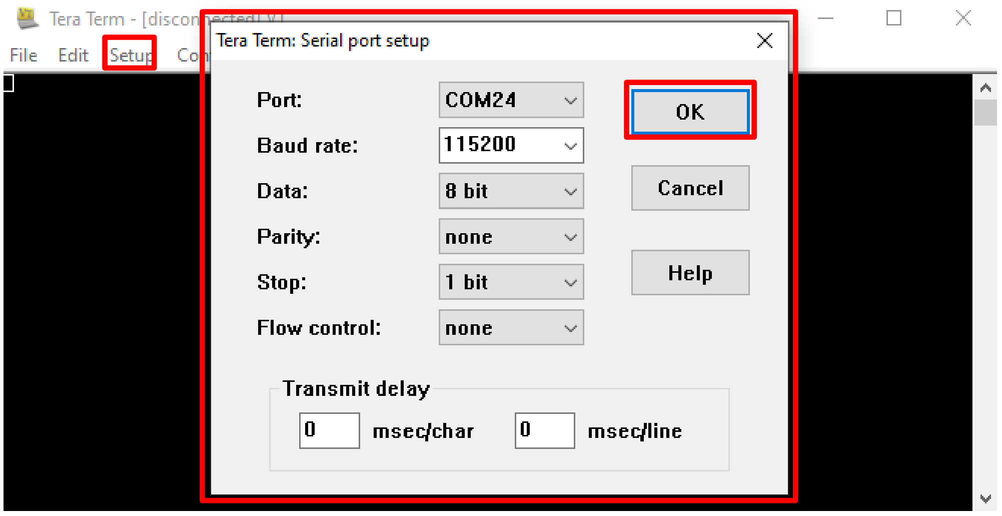
(1) Bring up the terminal software and select the "File" > "New Connection" to set the connection on the software.



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

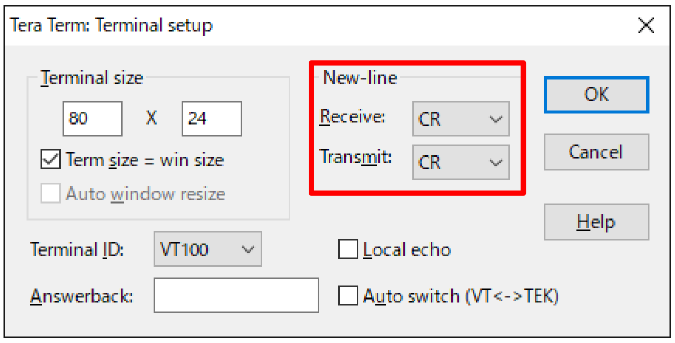
Set the settings about serial communication protocol on a terminal software as below:

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


Tera Term - [disconnected] VT
File Edit Setup Co
Tera Term: Serial port setup
Port: COM24
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay
0 msec/char 0 msec/line

Select the "Setup" > "Terminal" to set the new-line code.

- New-line:"CR" or "AUTO"


Tera Term: Terminal setup
Terminal size
80 X 24
☑ Term size = win size
☐ Auto window resize
New-line
Receive: CR
Transmit: CR
Terminal ID: VT100
Answerback:
☐ Local echo
☐ Auto switch (VT<->TEK)

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

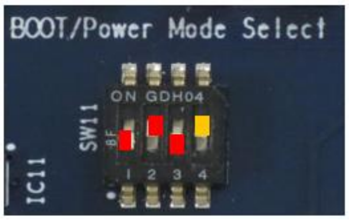


Table 9. SW11

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.

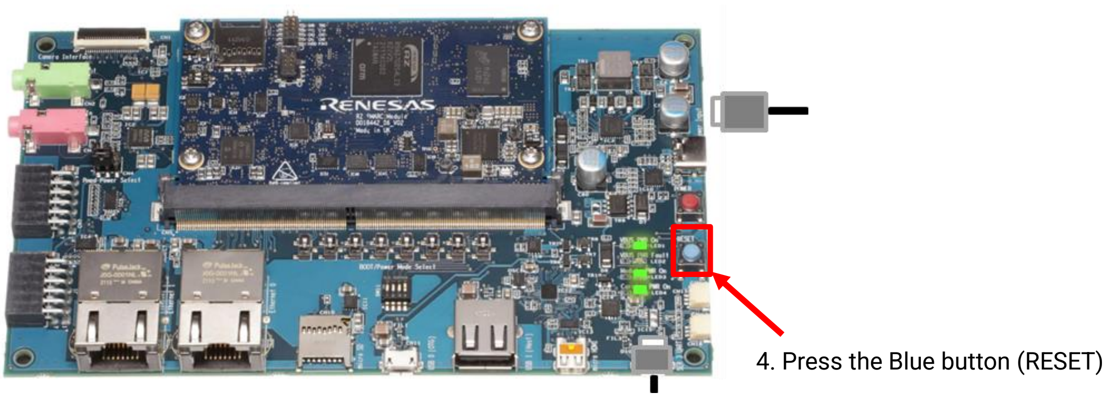
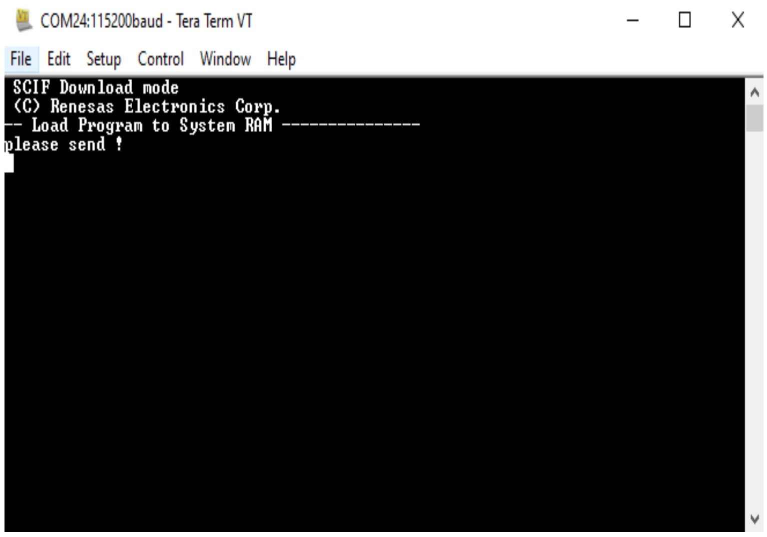


Figure 6. Press the RESET button


COM24:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
 SCIF Download mode
 (C) Renesas Electronics Corp.
-- Load Program to System RAM ---------------
please send !
```

4.3 Download Flash Writer to RAM

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

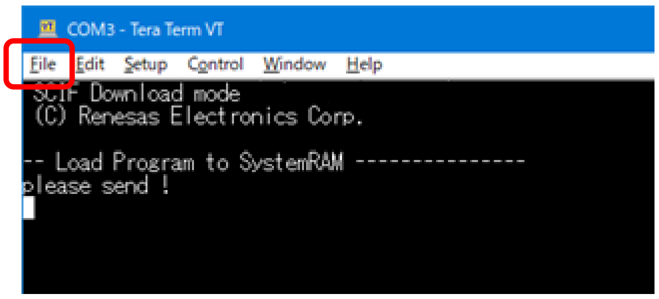
```
 SCIF Download mode
 (C) Renesas Electronics Corp.

-- 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.

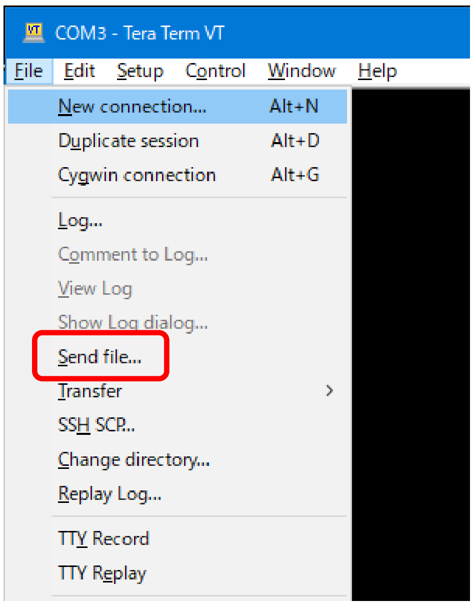
Below is a sample procedure with Tera Term.

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

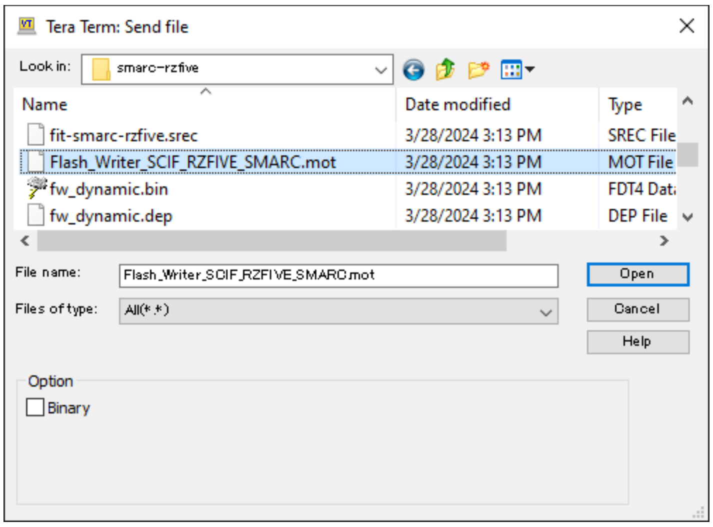


```
 SCIF Download mode
 (C) Renesas Electronics Corp.

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



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


Tera Term: Send file

Look in: smarc-rzfive

Name	Date modified	Type
fit-smarc-rzfive.srec	3/28/2024 3:13 PM	SREC File
Flash_Writer_SCIF_RZFIVE_SMARC.mot	3/28/2024 3:13 PM	MOT File
fw_dynamic.bin	3/28/2024 3:13 PM	FDT4 Dat
fw_dynamic.dep	3/28/2024 3:13 PM	DEP File

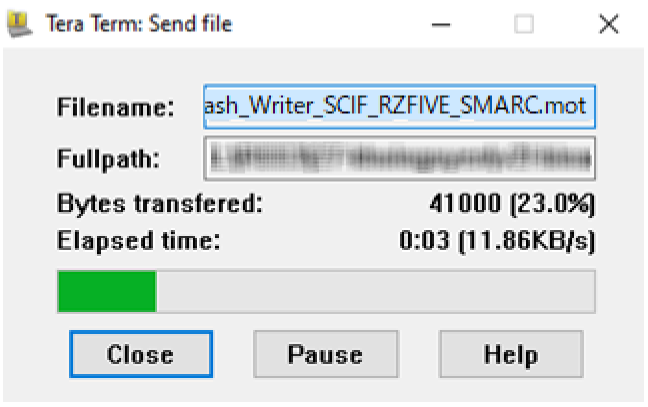
File name: Flash_Writer_SCIF_RZFIVE_SMARC.mot

Files of type: All(*.*)

Option

Binary

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


Tera Term: Send file

Filename: ash_Writer_SCIF_RZFIVE_SMARC.mot

Fullpath: [redacted]

Bytes transfered: 41000 (23.0%)

Elapsed time: 0:03 (11.86KB/s)

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
>
```

4.4 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 depend on each target board as described in **Table 10** .

Before writing the loader files, change the Flash Writer transfer rate from default (115200bps) to high speed (921600bps) with "SUP" command of Flash Writer.

```
>SUP
Scif speed UP
Please change to 921.6Kbps baud rate setting of the terminal.
```

After "SUP" command, change the serial communication protocol speed from 115200bps to 921600bps as well by following the steps described in 4.2.3, and push the enter key.

Next, use "XLS2" command of Flash Writer 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 downloading 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 downloading 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, change the serial communication protocol speed from 921600 bps to 115200 bps by following the steps described in 4.2.3.

Finally, turn off the power of the board by pressing SW9.

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

4.5 Change Back to Normal Boot Mode

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

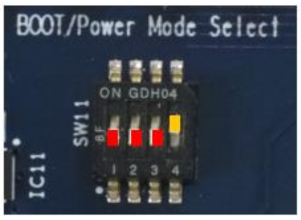
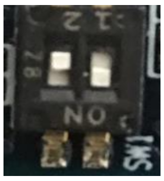


Table 11. SW11

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
=>
```

5. Booting and Running Linux

Set microSD card to slot on carry board.

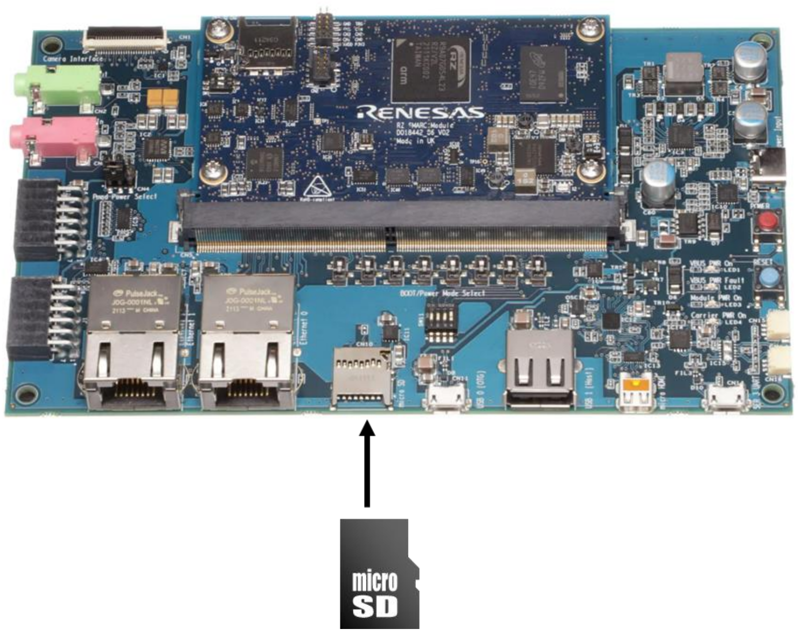


Figure 7.Set microSD card to SMARC-EVK

Now the board can boot up 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 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) 3.1.33 smarc-rzfive ttySC0

BSP: RZFive/RZFive-SMARC-EVK/3.0.7
LSI: RZFive
Version: 3.0.7
smarc-rzfive 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-rzfive:~#
```

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-rzfive:~# 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 2.2 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 building again after a period of time. Or build it again from scratch with the commands below.

```
$ cd ~/rzfve_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-rzfive-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 ./rzfive_vlp_${PACKAGE_VERSION}/build/tmp/deploy/sdk/poky-glibc-x86_64-core-
image-bsp-aarch64-smarc-rzfive-toolchain-<version>.sh
Poky (Yocto Project Reference Distro) SDK installer version 3.1.33
=================================================================
Enter target directory for SDK (default: /opt/poky/3.1.33): ~/workspace/rzfive/vlpf3.
0.7-build-test/build/tmp/deploy/sdk/temp
You are about to install the SDK to "/home/renesas/workspace/rzfive/vlpf3.0.7-build-te
st/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 envir
onment setup script e.g.
$ ./opt/poky/3.1.33/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/3.1.33/environment-setup-riscv64-poky-linux
```

7.1.2 How to build Linux application

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

```
$ mkdir ~/hello_apl
$ cd ~/hello_apl
```

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

$(APP):
<-tab->$(CC) -o $(APP) $(SRC) $(CFLAGS)

install:
<-tab->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 -Wfo
rmat-security -Werror=format-security --sysroot=/opt/poky/3.1.33/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.

Step4. 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 ~/hello_apl/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-rzfive login: root
root@smarc-rzfive:~#
root@smarc-rzfive:~# /usr/bin/linux-helloworld

Hello World
root@smarc-rzfive:~#
```

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 VLP by bitbake command. Please refer to the Release Note of the RZ/Five VLP to obtain a binary file of Flash Writer.

If you need the 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 the latest Flash Writer.

8.1.1 Preparing cross compiler

FlashWriter runs on target boards. Please get a 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/oecore-x86_64/environment-setup-riscv64-oe-linux
```

8.1.2 Building Flash Writer

Get source codes of Flash Writer from the GitHub repository and check out 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.

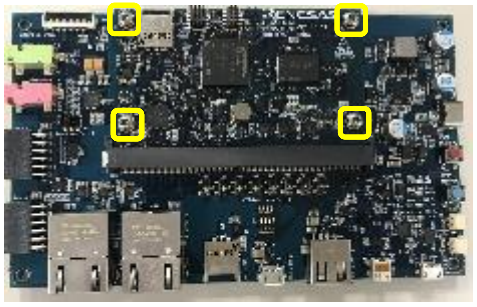
8.2 How to replace the SMARC Module Board

Please be careful when replacing the board as follows.

1. Remove the four screws.

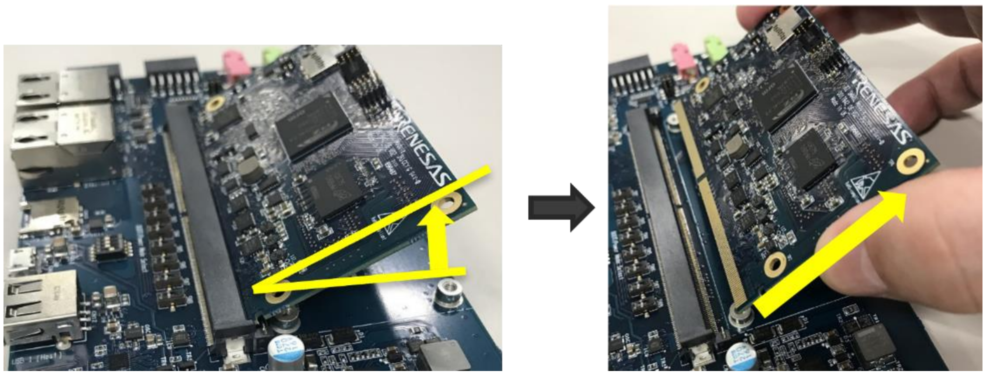
Note: The screw thread is a special shape, so be careful not to crush the screw thread.

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

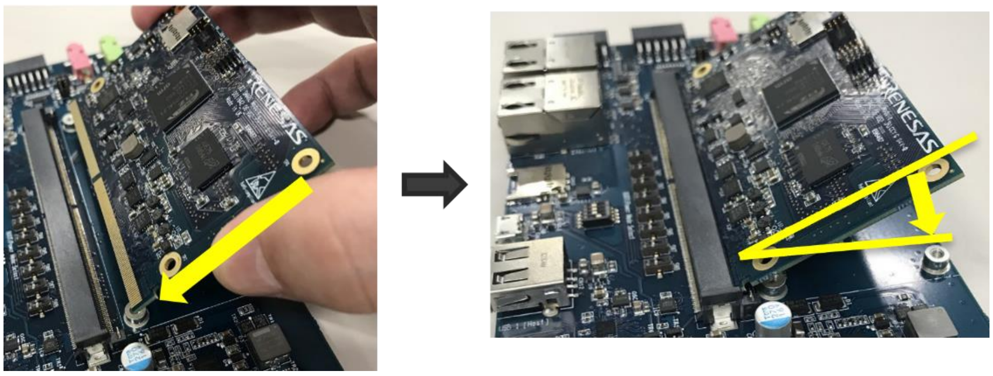


Specially shaped screw threads

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 How to boot from eMMC

In this section, the steps to boot from eMMC on RZ/Five is described.

8.3.1 Rebuild rootfs

Build a rootfs according to Release Note. After that, please rebuild the rootfs with the command below.

```
$ cd ${your build directory} (ex.$ cd ~/rzfive_vlp_v3.0.7)
$ source poky/oe-init-build-env
$ echo 'IMAGE_INSTALL_append = " e2fsprogs-mke2fs"'>> conf/local.conf
$ bitbake core-image-xxxxx (ex. core-image-weston)
```

8.3.2 Writing Bootloader for eMMC Boot

For the boot operation, EXT_CSD register of eMMC needs to be modified and two boot loader files need to be written to the target board. Modifying register address and value information are described in **Table 12.**, and corresponding bootloader files and specified address information are depending on each target board as described in **Table 13.**.

After booting the Flash Writer (Refer to Section 4.3), “EM_SECSD” command of Flash Writer is used to modify EXT_CSD register of eMMC to enable eMMC boot.

Then, “EM_W” 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 eMMC with information where the data should be loaded on the address of the main memory.

```
>EM_SECSD
  Please Input EXT_CSD Index(H'00 - H'1FF) :b1
  EXT_CSD[B1] = 0x00
  Please Input Value(H'00 - H'FF) :2
  EXT_CSD[B1] = 0x02
>EM_SECSD
  Please Input EXT_CSD Index(H'00 - H'1FF) :b3
  EXT_CSD[B3] = 0x00
  Please Input Value(H'00 - H'FF) :8
  EXT_CSD[B3] = 0x08
```

```
>EM_W
EM_W Start --------------
---------------------------------------------------------
Please select,eMMC Partition Area.
 0:User Partition Area   : 62160896 KBytes
  eMMC Sector Cnt : H'0 - H'0768FFFF
 1:Boot Partition 1      : 32256 KBytes
  eMMC Sector Cnt : H'0 - H'0000FBFF
 2:Boot Partition 2      : 32256 KBytes
  eMMC Sector Cnt : H'0 - H'0000FBFF
---------------------------------------------------------
  Select area(0-2)>1
-- Boot Partition 1 Program -----------------------------
Please Input Start Address in sector :1
Please Input Program Start Address : 11e00
Work RAM(H'50000000-H'50FFFFFF) Clear....
please send ! ('.' & CR stop load)
```

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

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

```
SAVE -FLASH.......
EM_W Complete!
```

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

```
> EM_W
EM_W Start --------------
---------------------------------------------------------
Please select,eMMC Partition Area.
 0:User Partition Area   : 62160896 KBytes
  eMMC Sector Cnt : H'0 - H'0768FFFF
 1:Boot Partition 1      : 32256 KBytes
  eMMC Sector Cnt : H'0 - H'0000FBFF
 2:Boot Partition 2      : 32256 KBytes
  eMMC Sector Cnt : H'0 - H'0000FBFF
---------------------------------------------------------
  Select area(0-2)>1
-- Boot Partition 1 Program -----------------------------
Please Input Start Address in sector :100
Please Input Program Start Address : 0
Work RAM(H'50000000-H'50FFFFFF) 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 downloading the binary, messages like below are shown on the terminal.

```
SAVE -FLASH.......
EM_W Complete!
```

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

Table 12. Address of EXT_CSD register of eMMC for eMMC boot

Address	Value to write
0xB1	0x02
0xB3	0x08

Table 13. Address for sending each loader binary file for eMMC boot

RZ/Five Evaluation Board Kit

File name	Partition to save to eMMC	Address to save to eMMC	Address to load to RAM
spl_bp-smarc-rzfive.srec	1	00000001	11E00
fit-smarc-rzfive.srec	1	00000100	00000

8.3.3 Create a microSD card to boot linux for eMMC boot

Create a microSD card by chapter 3. After that, please go back to following instructions before un-mounting a microSD card.

Copy a kernel image, a device tree file and rootfs to the second partition of the microSD card.

```
$ cd /media/user/rootfs/home/root/
$ sudo cp $WORK/build/tmp/deploy/images/<board>/<Linux kernel> ./
$ sudo cp $WORK/build/tmp/deploy/images/<board>/<Devise tree> ./
$ sudo cp $WORK/build/tmp/deploy/images/<board>/<root filesystem> ./
$ cd $WORK
$ sudo umount /media/user/rootfs
```

8.3.4 Setting U-boot and writing rootfs to eMMC

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

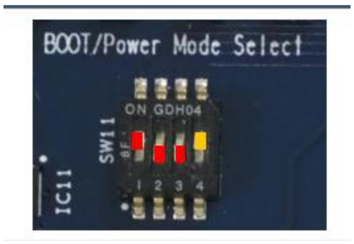
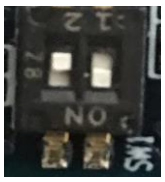


Table 14 SW11

1	2	3	4
ON	OFF	OFF	ON

Note:-

Set the SW1 on SoM module to eMMC mode. Please refer to Section 4.1.2.



Refer to Section 4.5 and set the u-boot environment variables (please ignore switch settings in that section). Then, turn off and on the power of the board by pressing the reset button SW10.

After booting Linux, please login as root and create partitions on eMMC

```
root@smarc-rzfive2l:~# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.35.1).
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 0xf3d53104.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): (Push the enter key)
Partition number (1-4, default 1): (Push the enter key)
First sector (2048-124321791, default 2048): (Push the enter key)
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-124321791, default 124321791): +50
0M

Created a new partition 1 of type 'Linux' and of size 500 MiB.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
```

```
Select (default p): (Push the enter key)

Using default response p.
Partition number (2-4, default 2): (Push the enter key)
First sector (1026048-124321791, default 1026048): (Push the enter key)
Last sector, +/-sectors or +/-size{K,M,G,T,P} (1026048-124321791, default 124321791):
(Push the enter key)

Created a new partition 2 of type 'Linux' and of size 58.8 GiB.

Command (m for help): p
Disk /dev/mmcblk0: 59.29 GiB, 63652757504 bytes, 124321792 sectors
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: 0xf3d53104

Device         Boot   Start       End   Sectors  Size Id Type
/dev/mmcblk0p1         2048   1026047   1024000  500M 83 Linux
/dev/mmcblk0p2      1026048 124321791 123295744 58.8G 83 Linux

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

root@smarc-rzfive:~#
```

Format eMMC.

```
root@smarc-rzfive:~# mkfs.ext4 /dev/mmcblk0p1
root@smarc-rzfive:~# mkfs.ext4 /dev/mmcblk0p2
```

Format eMMC and write the kernel, the device tree, and the rootfs.

```
root@smarc-rzfive:~# mount /dev/mmcblk0p1 /mnt/
root@smarc-rzfive:~# cp Image-smarc-rzg2l.bin /mnt/
root@smarc-rzfive:~# cp Image-r9a07g043f01-smarc.dtb /mnt/
root@smarc-rzfive:~# umount /dev/mmcblk0p1
root@smarc-rzfive:~# mount /dev/mmcblk0p2 /mnt/
root@smarc-rzfive:~# tar xf /home/root/core-image-weston-smarc-rzg2l.tar.bz2 \
-C /mnt/
root@smarc-rzfive:~# umount /dev/mmcblk0p2
```

8.3.5 Setting U-boot for eMMC boot

Reset the board by pressing the reset button SW10.

```
U-Boot 2021.10 (Apr 22 2022 - 03:04:59 +0000)

CPU:   Renesas Electronics K rev 16.15
Model: smarc-rzfive
DRAM:  1.9 GiB
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from MMC... OK
In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
Net:   eth0: ethernet@11c20000
Hit any key to stop autoboot:  0
=>
```

Set environment variables to boot from eMMC. The following variables are for the RZ/Five board. Please replace the file names in "bootcmd" according to the Release Note when you use other boards and set IP address of your Linux Host PC.

```
=> setenv bootargs 'root=/dev/mmcblk0p2 rootwait'
=> setenv bootcmd 'mmc dev 1; ext4load mmc 0:1 0x48080000 Image-smarc-rzfive.bin; ext4
load mmc 0:1 0x48000000 Image-r9a07g043f01-smarc.dtb; booti 0x48080000 - 0x48000000'
=> saveenv
Saving Environment to MMC... Writing to MMC(0)….OK
```

Please reset the board again for eMMC boot.

8.4 Docker

This section explains how to build the VLP with Docker enabled and how to obtain and run the "hello-world" image.

(1) Add layers to enable Docker

After completing step (2) in section 2.2, run the following commands. Once executed, proceed until step (5) in section 2.2 to build the VLP with Docker enabled.

```
$ cd ~/rzfive_vlp_${PACKAGE_VERSION}/build
$ bitbake-layers add-layer ../meta-openembedded/meta-filesystems
$ bitbake-layers add-layer ../meta-openembedded/meta-networking
$ bitbake-layers add-layer ../meta-virtualization
```

(2) Boot the evaluation board

Follow section 4 and 5, boot the evaluation board, and proceed to the next step.

(3) Check Docker on the evaluation board (Optional)

After booting the board, optionally run the following commands to display the status of the Docker Daemon and its version.

```
root@smarc-rzfive:~# systemctl status docker.service
root@smarc-rzfive:~# docker version
```

(4) Hello world

Run the commands below to get the docker image of hello-world and run the image.

```
root@smarc-rzfive:~# docker pull hello-world
root@smarc-rzfive:~# docker run hello-world
Hello from Docker!
```

8.5 Booting Setup with Ubuntu PC

This section describes how to write Flash writer and bootloaders using Ubuntu PC. Here is an example using an Ubuntu PC and minicom.

Please connect the reference board to your Ubuntu PC.

(1) Check the serial connected to the Ubuntu PC

```
$ ls -l /dev/serial/by-id
```

Assuming that the serial device is connected to "ttyUSB0", the following procedure is introduced.

(2) Allow access to the serial device

```
$ sudo chmod 666 /dev/ttyUSB0
```

(3) Get a minicom communication app

```
$ sudo apt-get install minicom
```

(4) Configure settings that can be executed by the logged-in user, and reboot

```
$ sudo usermod -a -G dialout $USER
$ sudo shutdown -r now
```

(5) Connect the minicom communication app to the serial device

```
$ minicom -D /dev/ttyUSB0
```

To change the settings, press "Ctrl+A" -> "Z" to display the HELP screen and select "Other Function(O)"-> "Serial port setup".

Normal communication is now possible.

(6) Send a file

This section describes the case of rewriting U-boot.

```
SCIF Download mode (w/o verification)
(C) Renesas Electronics Corp.

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

“Ctrl + A” -> “Z”

```
+-------------------------------------------------------------------+
|                      Minicom Command Summary                      |
|                                                                   |
|              Commands can be called by CTRL-A <key>               |
|                                                                   |
|               Main Functions                  Other Functions    |
|                                                                   |
| Dialing directory..D  run script (Go)....G | Clear Screen.......C |
| Send files.........S  Receive files......R | cOnfigure Minicom..O |
| comm Parameters....P  Add linefeed.......A | Suspend minicom....J |
| Capture on/off.....L  Hangup.............H | eXit and reset.....X |
| send break.........F  initialize Modem...M | Quit with no reset.Q |
| Terminal settings..T  run Kermit.........K | Cursor key mode....I |
| lineWrap on/off....W  local Echo on/off..E | Help screen........Z |
| Paste file.........Y  Timestamp toggle...N | scroll Back........B |
| Add Carriage Ret...U                                             |
|                                                                   |
|             Select function or press Enter for none.             |
+-------------------------------------------------------------------+
```

Please select Senf files "S" and ascii.

```
+-[Upload]--+
| zmodem    |
| ymodem    |
| xmodem    |
| kermit    |
| ascii     |
+-----------+--
```

At first please select Flash Write file.

```
+------------------------[Select a file for upload]------------------------+
|Directory: /home/user                                                    |
| .sudo_as_admin_successful                                               |
| Flash_Writer_SCIF_RZFIVE_SMARC.mot                                      |
| fit-smarc-rzfive.srec                                                   |
| spl-smarc-rzfive.srec                                                   |
|              ( Escape to exit, Space to tag )                           |
+-------------------------------------------------------------------------+

               [Goto]  [Prev]  [Show]   [Tag]  [Untag] [Okay]
```

Start uploading. Press any key when upload completed.

```
+------------[ascii upload - Press CTRL-C to quit]------------+
|ASCII upload of " Flash_Writer_SCIF_RZFIVE_SMARC.mot          |
|                                                              |
|82.5 Kbytes transferred at 11234 CPS... Done.                 |
|                                                              |
| READY: press any key to continue...                          |
|                                                              |
+--------------------------------------------------------------+
```

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

Next is writing U-boot step. Please refer to 4.4Write the Bootloader. Upload with the "ascii" command in the same way as a Flash Writer file.

8.6 Device drivers

The following drivers are supported: For detailed information on how to use it, 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 15. Support device drivers

Device Driver	Documents
Kernel Core	r01us0468ej0xxx-rz-g_Kernel_Core_UME.pdf
Direct Memory Access Controller (DMAC)	r01us0480ej0xxx-rz-g_DMAC_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.
1.01	Oct. 31, 2023	-	Add the patch file of update1.
1.02	Apr. 24, 2024	5	Add “2.1 Required Host OS” section.
		39	Add “8.3 How to boot from eMMC” section.
		45	Add “8.4 Docker” section.
1.03	May 31, 2024		VLP/F v3.0.6-update1
		46	Add “8.5 Booting Setup with Ubuntu PC” section.
1.04	Jul. 31, 2024	17	Update “3.2.2 Write files to the microSD card”.
1.05	Jan. 31, 2025	34	Display “<-tab->” code in the Makefile.
1.06	Mar. 31, 2025		VLP/F v3.0.7

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/