SMARC EVK of RZ/V2L

Linux Start-up Guide

R01US0617EJ0102
Rev.1.02
Feb. 29, 2024

Introduction

This document provides a guide to prepare RZ/V2L 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/V2L reference board

- RZ/V2L Evaluation Board Kit PMIC version (smarc-rzv2l-pmic) (*1)
  - RZ/V2L SMARC Module Board (P/N: RTK9754L23C01000BE)
  - RZ SMARC Series Carrier Board (P/N: RTK97X4XXXB00000BE)

Please refer to Appendix 8.2 for how to distinguish and replace each board.

(*1) “RZ/V2L Evaluation Board Kit PMIC” includes the RZ/V2L SMARC Module Board and the RZ SMARC Series Carrier Board.

Target Software

- RZ/V Verified Linux Package version 3.0.5-update3 or later. (hereinafter referred to as “VLP/V”)

Contents

1. Environment Requirement 4
2. Build Instructions 6
2.2 Notes 10
3. Preparing the SD Card 15
3.1 Write files to the microSD card 15
4. Reference Board Setting 20
4.1 Preparation of Hardware and Software 20
4.1.1 How to set boot mode and input voltage 21
4.1.2 How to set SW1 22
4.1.3 How to use debug serial (console ouput) 22
4.2 Startup Procedure 23
4.2.1 Power supply 23
4.2.2 Building files to write 24
4.2.3 Settings 24
4.3 Download Flash Writer to RAM 27
4.4 Write the Bootloader 29
4.5 Change Back to Normal Boot Mode 31
5. Booting and Running Linux 33
5.1 Power on the board and Startup Linux 33
5.2 Shutdown the Board 34
6. Building the SDK 35
7. Application Building and Running 36
7.1 Make an application 36
7.1.1 How to extract SDK 36
7.1.2 How to build Linux application 36
7.2 Run a sample application 38
8. Appendix 39
8.1 Preparing Flash Writer 39
8.1.1 Preparing cross compiler 39
8.1.2 Building Flash Writer 40
8.2 How to replace the SMARC Module Board 41
8.3 How to boot from eMMC 42
8.3.1 Rebuild rootfs 42
8.3.2 Writing Bootloader for eMMC Boot 42
8.3.3 Create a microSD card to boot linux for eMMC boot 44
8.3.4 Setting U-boot and Writing rootfs to eMMC 45

8.3.5 Setting U-boot for eMMC boot 47
8.4 How to boot from eSD 48
8.4.1 Steps for eSD boot 48
8.4.2 Prepare micro SD card 48
8.4.3 Copy the Kernel image and DTB and Extract the root file system 48
8.4.4 Writing Bootloader for eSD Boot 49
8.4.5 Set SMARC EVK board for eSD boot 50
8.4.6 Power on and boot 50

9. Revision History 52

Website and Support 53

1. Environment Requirement

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

0 shows the recommended environment for this package.

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.

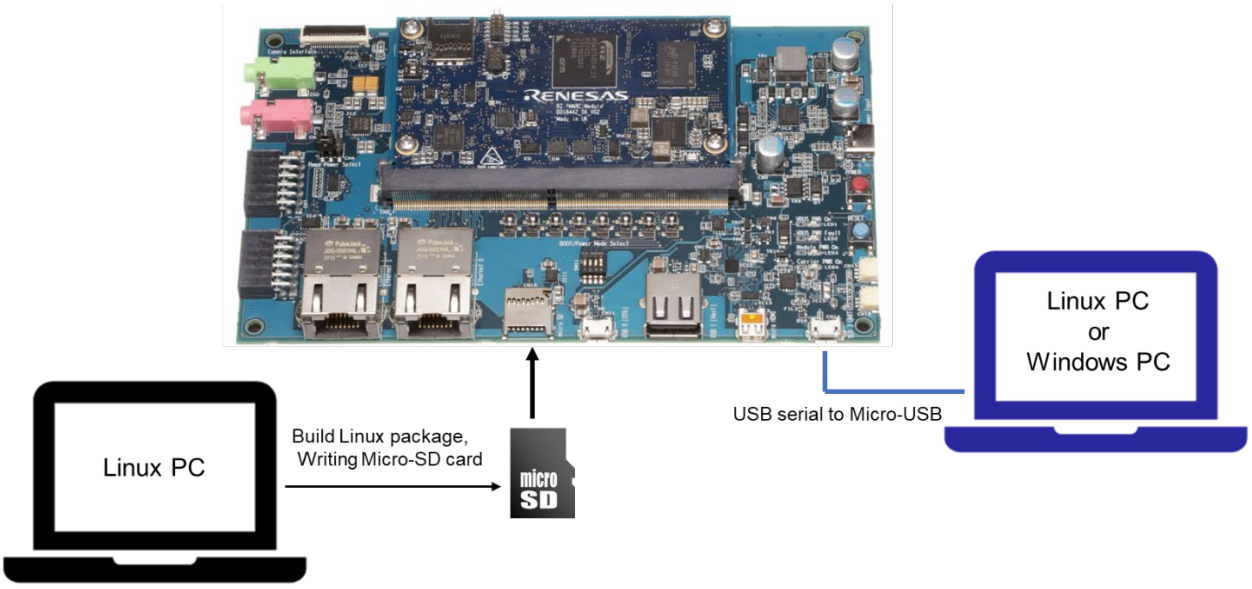


Figure 1. Recommend environment.

Table 1. Equipment and Software Necessary for Developing Environments of RZ/V2L Linux Platform

Equipment	Description
Linux Host PC	Used as build/debug environment 100GB free space on HDD or SSD is necessary
OS	Ubuntu 20.04 LTS 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 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 Please install the VCP Driver corresponding to the target board.
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.
micro–SD Card	Use to boot the system, and store applications. Note that use a micro-SDHC card for the flash writer.

Most bootable images VLP/V supports can be built on an “offline” environment.
The word “offline” means an isolated environment which does not connect to any network. Since VLP/V includes all necessary source codes of OSS except for the Linux kernel, VLP/V 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 (including the SDK build)
- core-image-weston (including the SDK build)
- core-image-qt (including the SDK build)

2. Build Instructions

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 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/3.1.26/brief-yoctoprojectqs/brief-yoctoprojectqs.html

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.

(1) 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 ~/rzv_vlp_<package version>
$ cd ~/rzv_vlp_<package version>
$ cp ../<package download directory>/*.zip .
$ unzip ./RTK0EF0045Z0024AZJ-<package version>.zip
$ tar zxvf ./RTK0EF0045Z0024AZJ-v3.0.5-update3/rzv_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-update3/

(2) Enable Graphics and Video Codec

The graphics package and the video codec package can be used at the same time. And also, one of the packages can be used.
Please check the website page below for available combination with Yocto recipe package.
RZ/V Verified Linux Package [5.10-CIP] | Renesas

If you want to enable the Graphics on RZ/V2L when building **core-image-weston**, please copy the Graphics package (RTK0EF0045Z13001ZJ-<version>_EN.zip or RTK0EF0045Z13001ZJ-<version>_JP.zip) to working directory and run the commands below. If you build core-image-minimal, please ignore this step.

```
$ unzip ./RTK0EF0045Z13001ZJ-<version>_EN.zip
$ tar zxvf ./RTK0EF0045Z13001ZJ-<version>_EN/meta-rz-features_graphics_<version>. \
tar.gz
```

If you want to enable the video codec on RZ/V2L when building **core-image-weston** or **core-image-bsp**, please copy the video codec package (RTK0EF0045Z15001ZJ--<version>_EN.zip or RTK0EF0045Z15001ZJ--<version>_JP.zip) to working directory and run the commands below.

```
$ unzip ./RTK0EF0045Z15001ZJ-<version>_EN.zip
$ tar zxvf ./RTK0EF0045Z15001ZJ-<version>_EN/meta-rz-features_codec_<version>.tar.gz
```

(3) 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-rzv2l/docs/template/conf/ source \
poky/oe-init-build-env build
```

(4) Add layers (Optional)

Please follow the below steps to add the layers you need. The steps add the settings to bblayers.conf.

- **Graphics**: Please run the below command if you need the Graphics library.

```
$ bitbake-layers add-layer ../meta-rz-features/meta-rz-graphics
```

- **Video Codec**: Please run the below command if you need the video codec library.

```
$ bitbake-layers add-layer ../meta-rz-features/meta-rz-codecs
```

- **Qt**: Please run the below commands if you want to include Qt.

```
$ bitbake-layers add-layer ../meta-qt5
$ bitbake-layers add-layer ../meta-rz-features/meta-rz-graphics
$ bitbake-layers add-layer ../meta-rz-features/meta-rz-codecs
```

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

(5) Decompress OSS files to “build” directory (Optional)

Run the commands below. This step is not mandatory and able to go to the step (6) 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_rzv_<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.

Open source software packages contain all source codes of OSSs. These are the same versions of OSSs used when VLP/V was verified.
If you are just evaluating VLP/V and RZ/V2L 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/V 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.

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

```
BB_NO_NETWORK = "1"
```

To change BB_NO_NETWORK from “0” to “1”.

(6) 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-rzv2l bitbake core-image-<target>
```

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

- core-image-minimal
- core-image-bsp
- core-image-weston
- core-image-qt

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

```
NOTE: Tasks Summary: Attempted 7427 tasks of which 16 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/V can build a few types of images listed in Table 2.

Table 2. Supported images of VLP/V

Image name	Purpose
core-image-minimal	Minimal set of components
core-image-bsp	Minimal set of components plus audio support and some useful tools
core-image-weston	Standard image with graphics support
core-image-qt	Enable Qt LGPL version

Table 3. Image files for RZ/V2L

RZ/V2L	**Linux kernel**	Image-smarc-rzv2l.bin
	Device tree file	Image-r9a07g054l2-smarc.dtb
	root filesystem	*<image name>*-smarc-rzv2l.tar.bz2
	Boot loader	・bl2_bp-smarc-rzv2l_pmic.srec ・fip-smarc-rzv2l_pmic.srec
	Flash Writer	Flash_Writer_SCIF_RZV2L_SMARC_PMIC_DDR4_2GB_1PCS.mot
	SD image	core-image-*<image name>* -smarc-rzv2l.wic.gz core-image-*<image name>* -smarc-rzv2l.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) Disable libraries of Graphics and Video Codec

When you want to disable the functions of the libraries of the graphics and the video codec, please add the following lines in `build/conf/local.conf`:

- Disable OpenGL ES library in the graphics package (*1)

```
DISTRO_FEATURES_remove = " opengles"
```

- Disable OpenCL library in the graphics package (*1)

```
DISTRO_FEATURES_remove = " opencl"
```

- Disable OpenMAX library for decode in the video codec package (*2)

```
DISTRO_FEATURES_remove = " hwh264dec hwh265dec"
```

- Disable OpenMAX library for encode in the video codec package (*2)

```
DISTRO_FEATURES_remove = " hwh264enc"
```

(*1) This library is included in RTK0EF0045Z13001ZJ-v1.2.0_EN.zip and RTK0EF0045Z13001ZJ-v1.2.0_JP.zip
(*2) This library is included in RTK0EF0045Z15001ZJ-v1.2.0_EN.zip and RTK0EF0045Z15001ZJ-v1.2.0_JP.zip

(3) Real time performance

If you want to use the kernel which improves the real-time performance, please add the line below to the file "`~/rzv_vlp_v3.0.x/build/conf/local.conf`".

```
IS_RT_BSP="1"
```

(4) USB Video Class

USB Video Class (UVC) driver is not installed with the default settings of VLP/V due to its large size.

In case UVC devices such as USB cameras are necessary, please install the driver by adding the line below to `local.conf`.

```
IMAGE_INSTALL_append = " kernel-module-uvcvideo "
```

(5) CIP Core Packages

VLP/V includes Debian 10 (Buster) and Debian 11 (Bullseye) based CIP Core Packages and Buster is enabled by the default settings. These packages can be replaced with other versions of packages.

Note that network access is required to start the build process when you enable these packages except for Buster (or Bullseye) 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 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 = "Buster"
```

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 `local.conf`:

```
# 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 = "Buster"
```

Note) The above 3 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 BSP, the packages will be replaced as below in the **Table 4**.

Table 4. Versions of all Buster and Bullseye Debian packages

Package	Buster Debian	Bullseye Debian
attr	2.4.48	2.4.48
busybox	1.30.1	1.30.1
coreutils	6.9	-
gcc	8.3.0	-
glib-2.0	2.58.3	-
glibc	2.28	2.31
gnupg	1.4.7	-
kbd	2.0.4	-
libassuan0	2.5.2	2.5.3
libgcrypt	1.8.4	-
libunistring	0.9.10	0.9.10
libnss	0.14.1	-
openssh	7.9p1	-
perl	5.30.1	-
pkgconfig	0.29	-
quilt	0.65	-

Note)
(-) These packages are not supported with Bullseye Debian version, so they used No CIP CORE version.

(6) 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.

(7) 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/V v3.0.5, WIC is supported and below guidelines are shown how to use it to boot Renesas RZ/G 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
  - core-image-weston

- core-image-qt

- There are 2 files *wic.bmap and *wic.gz in deploy folder after building successfully. Example:
  - core-image-minimal-smarc-rzv2l.wic.bmap
  - core-image-minimal-smarc-rzv2l.wic.gz

(8) 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/V 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 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_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 method. Please select one of them and follow the steps.

There is another method using a script. When you want to use the script, please see below.

rzg2_bsp_scripts/usb_sd_partition at master · renesas-rz/rzg2_bsp_scripts (github.com)

3.1 Write files to the microSD card

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 5. 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): (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

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

(8) 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
```

Copy kernel and device tree file to the first partition.

```
$ cp $WORK/build/tmp/deploy/images/smarc-rzv2l/<Linux kernel> /media/user/A299-E6A6
$ cp $WORK/build/tmp/deploy/images/smarc-rzv2l/<Devise tree> /media/user/A299-E6A6
```

Expand rootfs to the second partition.

```
$ cd /media/user/rootfs
$ sudo tar jxvf $WORK/build/tmp/deploy/images/smarc-rzv2l/<root filesystem>
```

The file nemes of *<Linux kernel>*, *<Devise tree>*, and *<root filesystem>* are listed in the Table 3. Image files for RZ/V2L

Table 6. File and directory in the micro SD card

Type/Number	Size	Filesystem	Contents
Primary #1	500MB (minimum 128MB)	FAT32	Image-smarc-rzv2l.bin Image-r9a07g054l2-smarc.dtb
Primary #2	All remaining	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)
- micro HDMI cable (Any cables) (*1)
- PC Installed FTDI VCP driver and Terminal software (Tera Term) (*2)

(*1) Please prepare this cable if you use core-image-weston or core-image-qt images.
(*2) Please install the FTDI driver that can be following website (https://www.ftdichip.com/Drivers/VCP.htm).

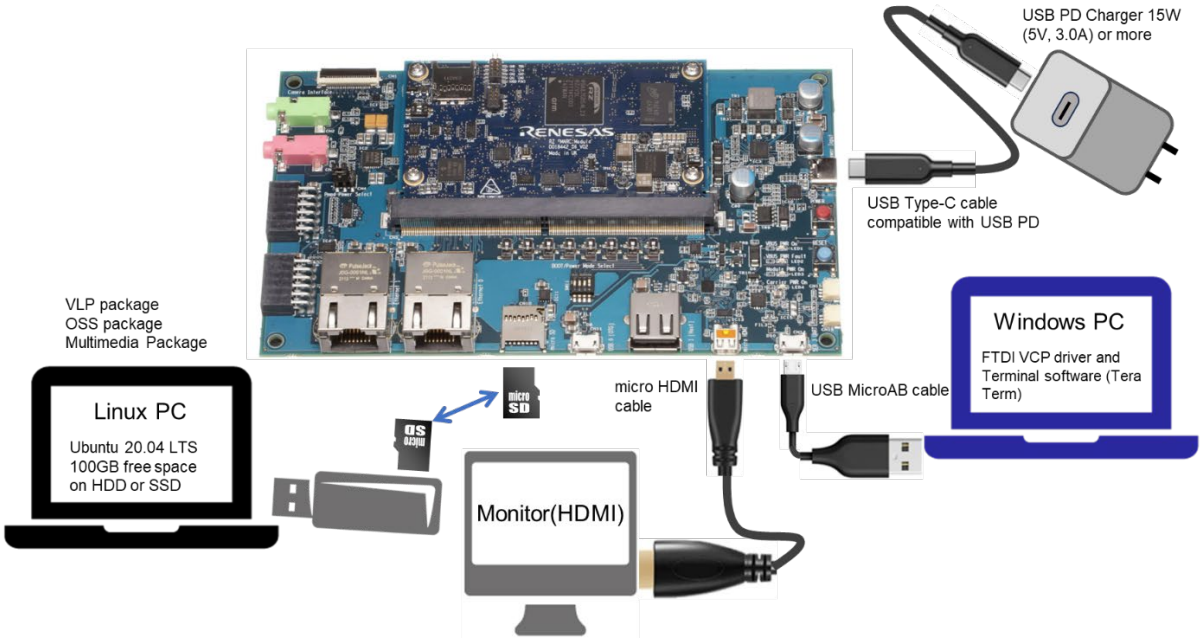


Figure 2. Operating environment

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/V2L.
- 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)



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.

- Pin no1 of the SW1 is used to select the JTAG debug mode or not.
  JTAG is not used, so set SW1-1 to normal operation mode.
- Pin no2 of the SW1 is used to select the eMMC or microSD mode. Please set SW1-2 to eMMC mode.



SW1-1	ON
SW1-2	OFF



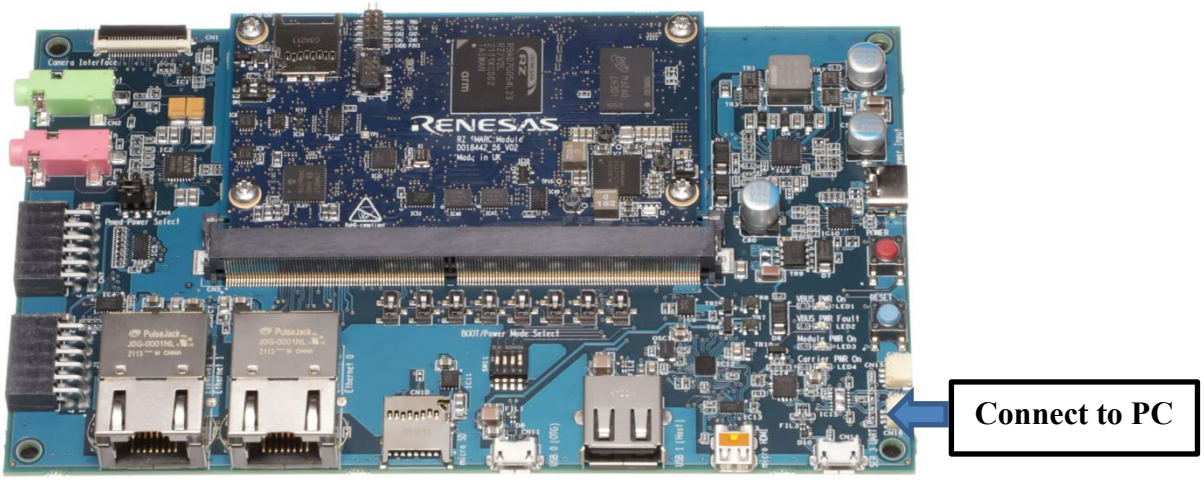
SW1-1	DEBUGEN
OFF	JTAG debug mode
ON	Normal operation

SW1-2	microSD/eMMC selection
OFF	Select eMMC on RTK9744L23C01000BE
ON	Select microSD slot on RTK9744L23C01000BE

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

4.1.3 How to use debug serial (console ouput)

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



CN14:USB Type-microAB Connector

Figure 3. Connecting console for debug

4.2 Startup Procedure

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.

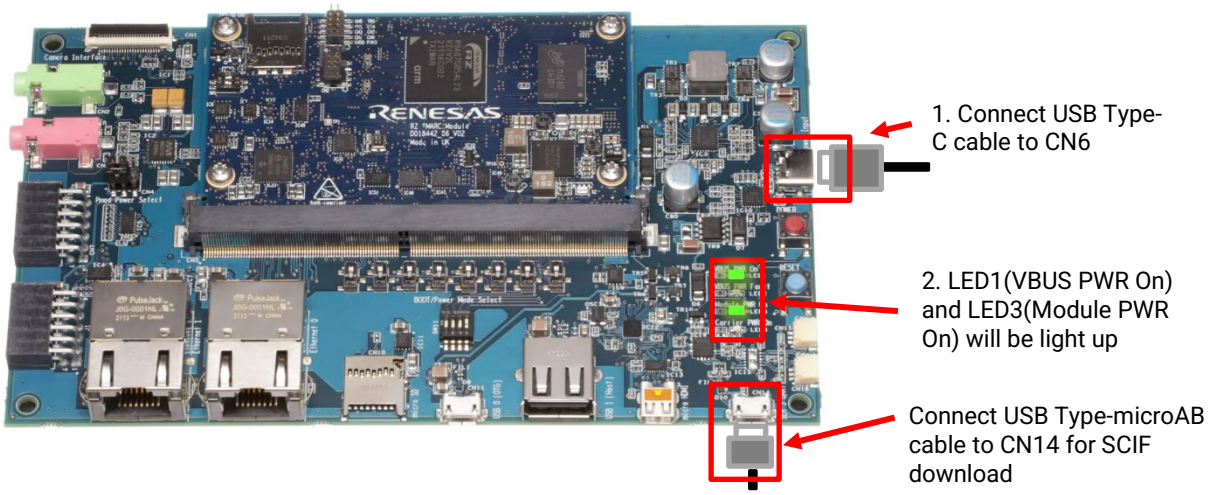


Figure 4. Connecting Power Supply

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.

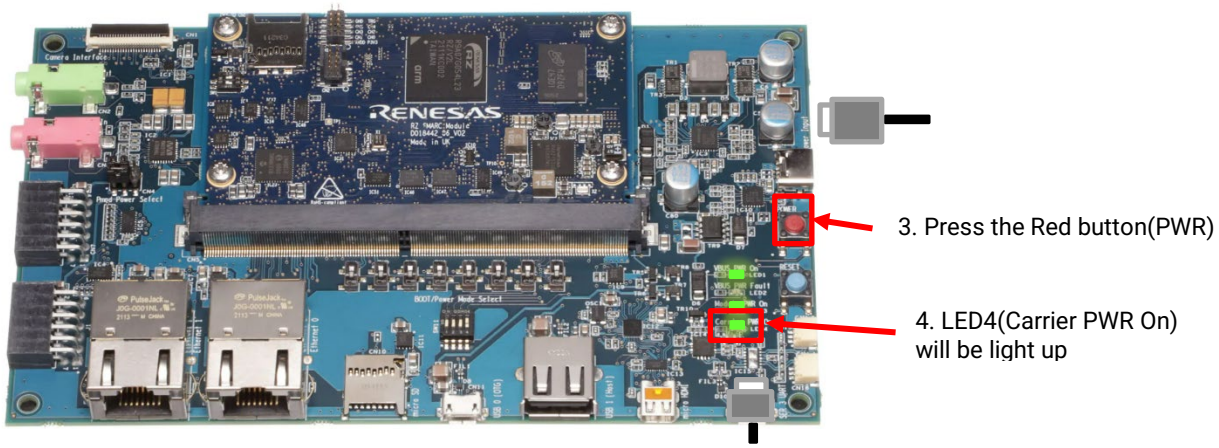


Figure 5. Power ON

4.2.2 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 following the build instruction. Please refer Table 2. Once builded copy these files to a PC which runs serial terminal software.

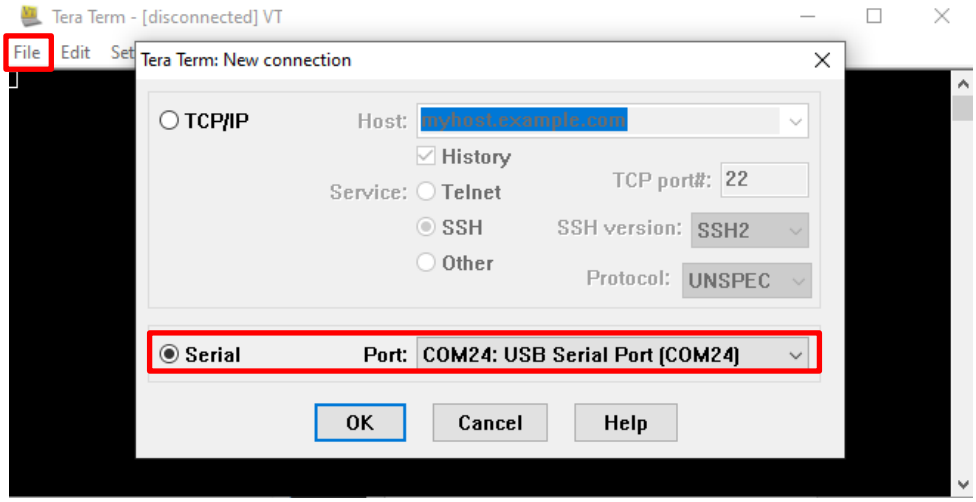
Table 7. File names of Boot loader

Board	File name of Boot loader
RZ/V2L Evaluation Board Kit PMIC version	• bl2_bp-smarc-rzv2l_pmic.srec • fip-smarc-rzv2l_pmic.srec

4.2.3 Settings

Connect between the board and a control Windows PC by USB serial cable according to the Release Note.

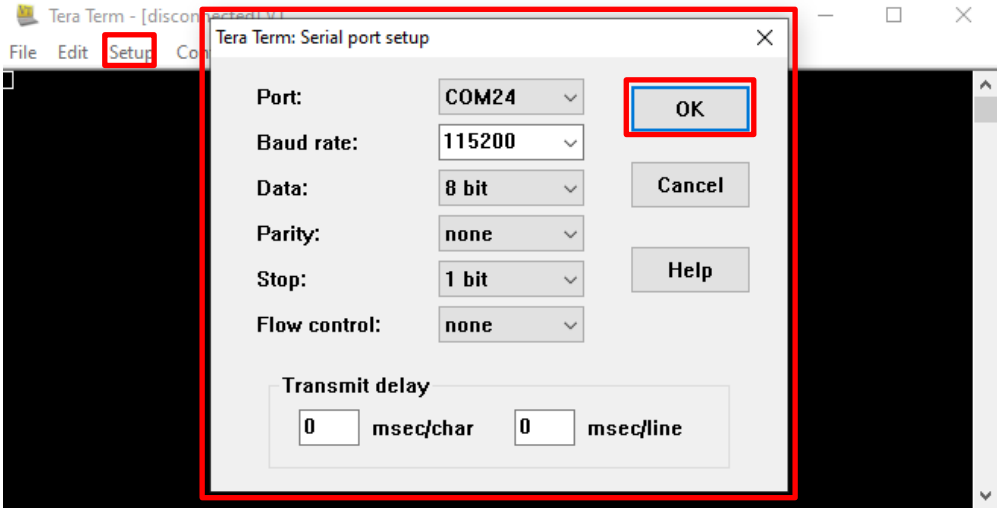
1. Bring up the terminal software and select the “File” > “New Connection” to set the connection on the software.



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:

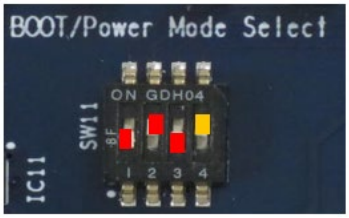


Table 8. SW11

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.

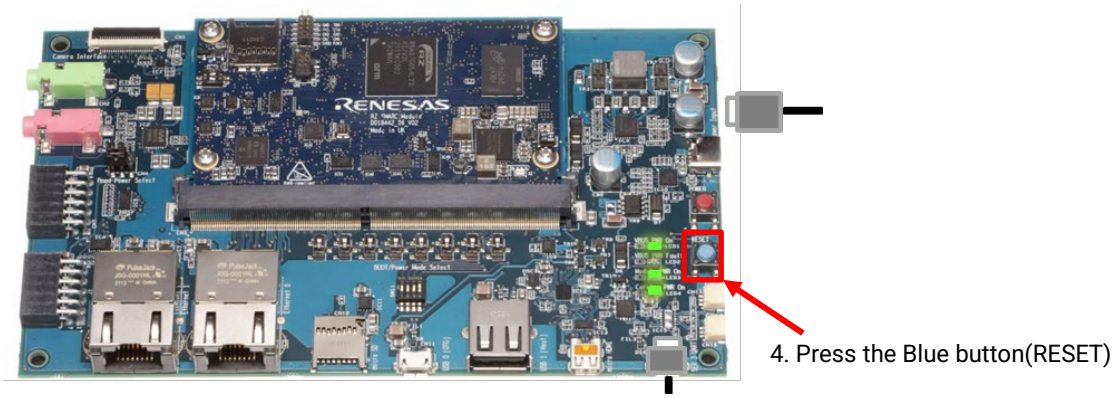
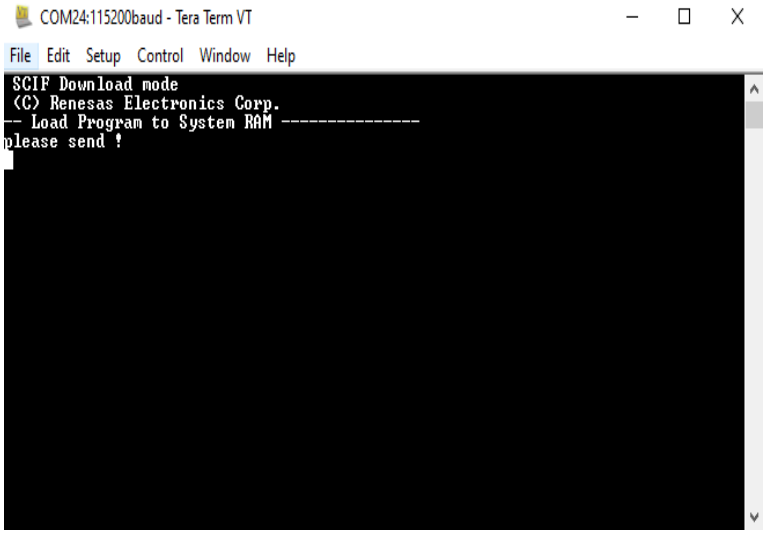


Figure 6. Press the RESET button



If you use a Linux PC instead of a Windows PC, follow the steps below to set the serial communication speed.

1. Determine which number on the USB the SMARC EVK is connected to as a tty.

```
$ lsusb
  :
  Bus 001 Device 009: ID 0403:6015 Future Technology Devices International, Ltd Bridge
(I2C/SPI/UART/FIFO)
  :
$ ls /dev
  :
  ttyUSB0
  :
```

2. Allow access to the appropriate ttyUSBx.

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

3. Start communication with the “cu” command.

```
$ cu -l /dev/ttyUSB0 -s 115200
```

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 !
```

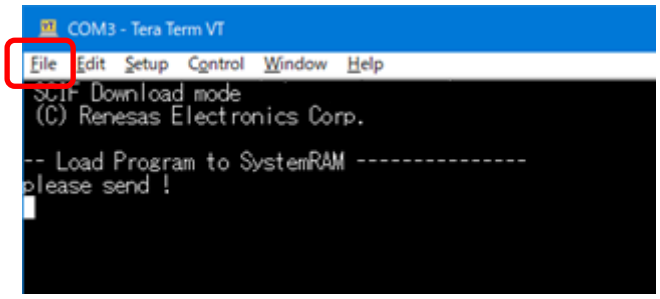
Send an image of Flash Writer using terminal software after the message “please send !” is shown. Please refer to the Table 10 below to know which file name of Flash Writer should be sent.

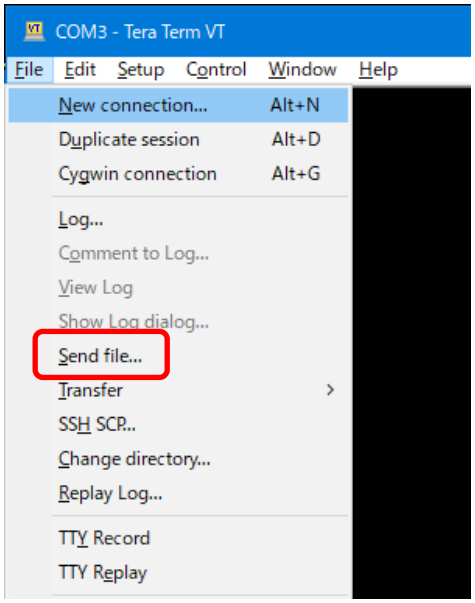
Table 9. File names of Flash Writer

Board name	File name of Flash Writer
RZ/V2L Evaluation Board Kit PMIC version	Flash_Writer_SCIF_RZV2L_SMARC_PMIC_DDR4_2GB_1PCS.mot

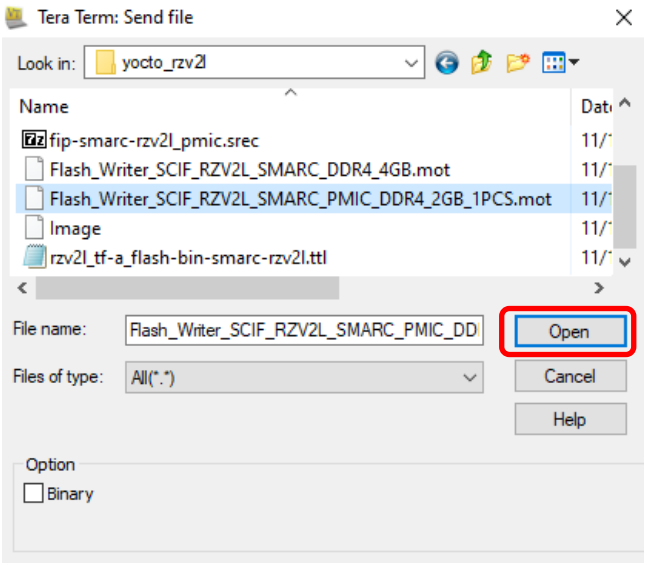
Below is a sample procedure with Tera Term.

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

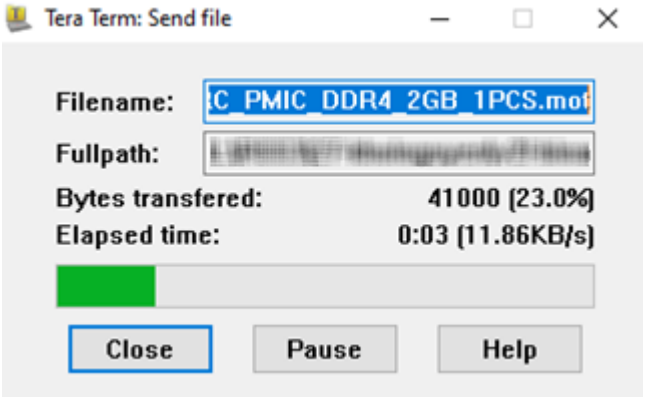




Then, select the image to be send and click “Open” button.



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/V2 Series V1.02 Nov.15,2021
 Product Code : RZ/V2L
>
```

When you use Linux PC, please open another terminal.

Use the “cat” command to output the file to ttyUSB0.

```
$ cat Flash_Writer_SCIF_RZV2L_SMARC_PMIC_DDR4_2GB_1PCS.mot > /dev/ttyUSB0
```

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 are depending on each target board as described in Table 8.

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/V2L Evaluation Board Kit PMIC version.:

```
>XLS2
===== Qspi writing of RZ/V2 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 “bl2_bp-smarc-rzv2l_pmic.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/V2 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'1D200
Work RAM(H'50000000-H'53FFFFFF) Clear....
please send ! ('.' & CR stop load)
```

Send the data of “fip-smarc-rzv2l_pmic.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, change the serial communication protocol speed from 921600 bps to 115200 bps by following the steps described in 4.2.3.

At last, turn off the power of the board by pressing the SW9.

Table 10. Address for sending each loader binary file

File name	Address to load to RAM	Address to save to ROM
bl2_bp-smarc-rzv2l_pmic.srec	11E00	00000
fip-smarc-rzv2l_pmic.srec	00000	1D200

4.5 Change Back to Normal Boot Mode

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

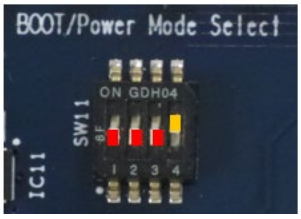


Table 11. SW11

1	2	3	4
OFF	OFF	OFF	ON

Note:-

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



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

```
root@smarc-rzv2l:~# NOTICE:  BL2: v2.7(release):v2.5/rzg2l-1.00-2007-g203c78323-dirty
NOTICE:  BL2: Built : 06:57:02, Nov 10 2022
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.7(release):v2.5/rzg2l-1.00-2007-g203c78323-dirty
NOTICE:  BL31: Built : 06:57:02, Nov 10 2022


U-Boot 2021.10 (Mar 15 2023 - 07:34:17 +0000)

CPU:   Renesas Electronics K rev 16.15
Model: smarc-rzv2l
DRAM:  1.9 GiB
WDT:   watchdog@0000000012800800
WDT:   Started with servicing (60s timeout)
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from MMC... OK
In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
U-boot WDT started!
Net:
Error: ethernet@11c20000 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
=>
```

In case booting from micro SD card on SMARC carrier board, set environment variables using the commands below. The commands below are for the RZ/V2L board.

```
=> setenv bootargs 'root=/dev/mmcblk1p2 rootwait'
=> setenv bootcmd 'mmc dev 1;fatload mmc 1:1 0x48080000 Image-smarc-rzv2l.bin; fatload
 mmc 1:1 0x48000000 Image-r9a07g054l2-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 SD card has two partitions and stores data as below:
First partition: formatted as FAT, includes Image-smarc-rzv2l.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.

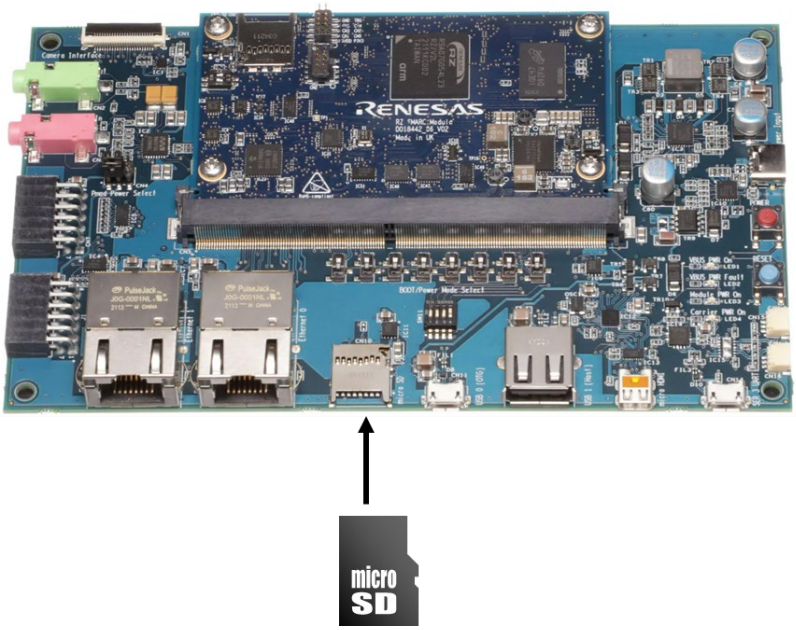


Figure 7. Set micro SD card to SMARC-EVK

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.

```
CPU:   Renesas Electronics K rev 16.15
Model: smarc-rzv2l
DRAM:  1.9 GiB
WDT:   watchdog@0000000012800800
WDT:   Started with servicing (60s timeout)
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from MMC... OK
In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
U-boot WDT started!
Net:
Error: ethernet@11c20000 address not set.
No ethernet found.

Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc1 is current device
17486336 bytes read in 1458 ms (11.4 MiB/s)
39443 bytes read in 7 ms (5.4 MiB/s)
```

```
Moving Image from 0x48080000 to 0x48200000, end=49310000
## Flattened Device Tree blob at 48000000
   Booting using the fdt blob at 0x48000000
   Loading Device Tree to 0000000057ff3000, end 0000000057fffa12 ... OKU-Boot 2021.10
(Mar 31 2022 - 03:57:20 +0000)

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
 :
 :
Poky (Yocto Project Reference Distro) 3.1.17 smarc-rzv2l ttySC0

BSP: RZV2L/RZV2L-SMARC-EVK/3.0.2
LSI: RZV2L
Version: 3.0.2
smarc-rzv2l login: root
Last login: Sun Sep 20 10:44:05 UTC 2020
root@smarc-rzv2l:~#
```

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-rzv2l:~# 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) – (6) of 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 ~/rzv_vlp_<package version>/build
$ MACHINE=smarc-rzv2l bitbake core-image-weston -c populate_sdk
```

For building Qt applications:

```
$ cd ~/rzv_vlp_<package version>/build
$ MACHINE=smarc-rzv2l bitbake core-image-qt -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-weston-aarch64-smarc-rzv2l-toolchain-<version>.
sh
```

Or

```
$ sudo sh poky-glibc-x86_64-core-image-qt-aarch64-smarc-rzv2l-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 ~/rzv_vlp_<package version>/build
$ MACHINE=smarc-rzv2l bitbake core-image-weston -c cleanall
$ MACHINE=smarc-rzv2l bitbake core-image-weston
```

For building general applications:

```
$ MACHINE=smarc-rzv2l bitbake core-image-weston -c populate_sdk
```

For building Qt applications:

```
$ MACHINE=smarc-rzv2l bitbake core-image-qt -c populate_sdk
```

7. Application Building and Running

This chapter explains how to make and run an application for RZ/V2L 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-weston-sdk-aarch64-smarc-rzv2l-toolchain-<ver
sion>.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 ./rzv_vlp_<package version>/build/tmp/deploy/sdk/poky-glibc-x86_64-core-imag
e-weston-aarch64-smarc-rzv2l-toolchain-<version>.sh
Poky (Yocto Project Reference Distro) SDK installer version 3.1.21
==================================================================
Enter target directory for SDK (default: /opt/poky/3.1.21):
You are about to install the SDK to "/opt/poky/3.1.21". 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/3.1.21/environment-setup-aarch64-poky-linux
 $ . /opt/poky/3.1.21/environment-setup-armv7vet2hf-neon-vfpv4-pokymllib32-linux-gnuea
bi
```

Step 2. Set up cross-compile environment:

```
$ source /<Location in which SDK is extracted>/environment-setup-aarch64-poky-linux
```

Note:

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

```
$ source /opt/poky/3.1.21/environment-setup-aarch64-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
aarch64-poky-linux-gcc  -mtune=cortex-a55 -fstack-protector-strong  -D_FORTIFY_SOURCE=
2 -Wformat -Wformat-security -Werror=format-security --sysroot=/opt/poky/3.1.21/sysroo
ts/aarch64-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. Also, this application must be cross-compiled for AArch64. Mar.17.2023 4.2

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 .
$ sudo chmod +x hello
```

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/V2L Evaluation Board Kit and start the system. After booting, run the sample application with the following command.

```
BSP: RZ/V2L-SMARC-EVK/<package version>
LSI: RZV2L
Version: <package version>
smarc-rzv2l login: root
root@smarc-rzv2l:~# ls
linux-helloworld  v4l2-init.sh
root@smarc-rzv2l:~# ./linux-helloworld

Hello World
root@smarc-rzv2l:~#
```

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/V VLP 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 Linaro or setup a Yocto SDK.

- **ARM toolchain:**

```
$ cd ~/
$ wget https://developer.arm.com/-/media/Files/downloads/gnu-a/10.2-2020.11/binrel/gcc
-arm-10.2-2020.11-x86_64-aarch64-none-elf.tar.xz
$ tar xvf gcc-arm-10.2-2020.11-x86_64-aarch64-none-elf.tar.xz
```

- **Yocto SDK:**

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

```
$ source /opt/poky/x.x.xx/environment-setup-aarch64-poky-linux
```

8.1.2 Building Flash Writer

Get source codes of Flash Writer from the GitHub repository and checkout the branch rz_v2l.

```
$ cd ~/
$ git clone https://github.com/renesas-rz/rzg2_flash_writer.git
$ cd rzg2_flash_writer
$ git checkout -b tmp 4168466783f06fa7f2aa5782c597803a6882ed2f
```

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

- **ARM toolchain:**

```
$ export PATH=$PATH:~/gcc-arm-10.2-2020.11-x86_64-aarch64-none-elf/bin
$ export CROSS_COMPILE=aarch64-none-elf-
$ export CC=${CROSS_COMPILE}gcc
$ export AS=${CROSS_COMPILE}as
$ export LD=${CROSS_COMPILE}ld
$ export AR=${CROSS_COMPILE}ar
$ export OBJDUMP=${CROSS_COMPILE}objdump
$ export OBJCOPY=${CROSS_COMPILE}objcopy

$ make clean
$ make BOARD=RZV2L_SMARC_PMIC
```

- **Yocto SDK:**

```
$ make clean
$ make BOARD=RZV2L_SMARC_PMIC
```

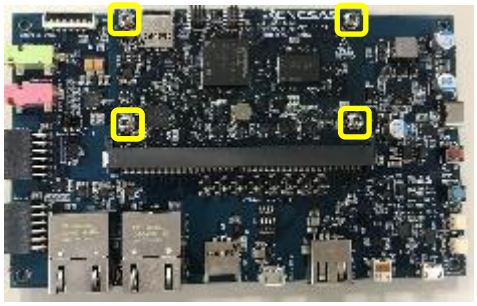
8.2 How to replace the SMARC Module Board

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.

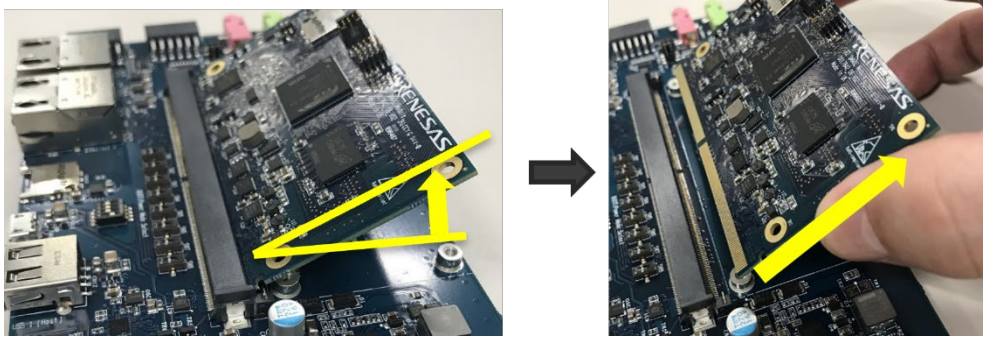
Please recommend to prepare a torx screwdriver which is a "T6" head size.



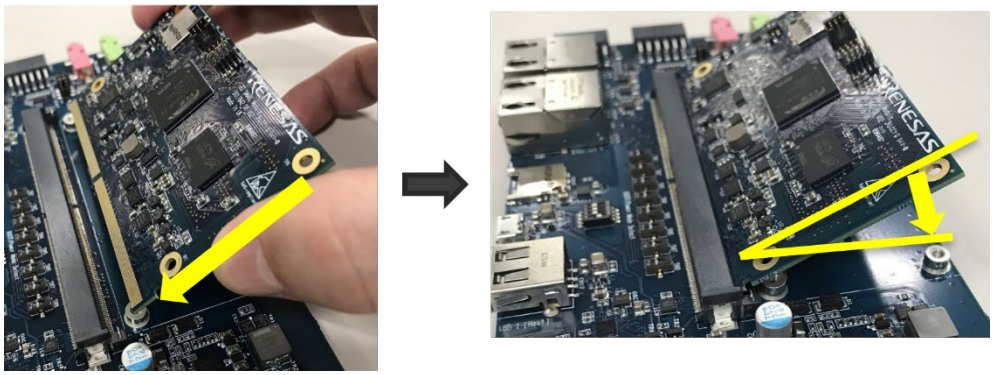


Specially shaped screw threads

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/V2L is described.

eMMC Boot refers to the SoC booting directly from eMMC flash after RESET. This means SPI Flash is not used.

Note that eMMC Flash is always used to store the u-boot environment variables, even when boot from SPI Flash is enabled. SPI Flash is never used to store the u-boot environment variables as there is not code to support that. This means that when you switch between SPI Flash booting and eMMC Flash booting, your u-boot environment variables will remain the same.

But default, the boot loader (TrustedFirmware-A) contains both support to boot from both SPI or eMMC Flash. During boot, the code will read a boot register to determine if it should load from SPI Flash or eMMC Flash. This means you can use the same default Yocto output for either SPI Flash boot or eMMC boot.

To program the eMMC Flash device on the SOM board, you can follow the instructions below.

You will need to boot with SD card first and partition eMMC Flash device. Then you can load the files from SD card to eMMC Flash.

Step1. Copy listed files to “/home/root” of the SD Card that you use to boot the system.

RZV2L : core-image-weston-smarc-rzv2l.tar.gz, Image and r9a07g054l2-smarc.dtb

Step2. Boot the board with the SD card and log into Linux.

Step3. Partition the eMMC Flash using fdisk.

Step4. Format the partition as ext4

Step5. Mount the ext4 partition to "/mnt".

Step6. Extract the file system to the ext4 partition

Step7. Unmount “/mnt”

Step8. Reboot and change the boot parameter in u-boot to boot from eMMC

8.3.1 Rebuild rootfs

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

```
$ cd ${your build directory} (ex.$ cd ~/rzv_vlp _v3.0.0)
$ 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 need 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 13, and corresponding bootloader files and specified address information are depending on each target board as described in Table 14.

After booting the Flash Writer (Refer to **4.3 Download Flash Writer to RAM**), “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.

For example, this part describes how to modify EXT_CSD register and write boot loader files in the case of RZ/V2L Evaluation Board Kit PMIC version.:

```
>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 “bl2_bp-smarc-rzv2l_pmic.srec” from terminal software after the message “please send !” is shown.

After successfully download 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 “fip-smarc-rzv2l_pmic.srec” from terminal software after the message “please send !” is shown.

After successfully download 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

File name	Partition to save to eMMC	Address to save to eMMC	Address to load to RAM
bl2_bp-smarc-rzv2l_pmic.srec	1	00000001	11E00
fip-smarc-rzv2l_pmic.srec	1	00000100	00000

8.3.3 Create a microSD card to boot linux for eMMC boot

Create a microSD card by the section 3 Preparing the SD Card. After finishing the steps, please 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:

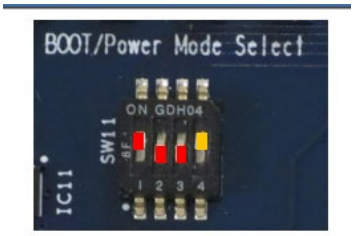


Table 14. SW11

1	2	3	4
ON	OFF	OFF	ON

Note:-

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



Refer to 4.5 Change Back to Normal Boot Mode and set the u-boot environment variables (please ignore switch settings). Then, turn 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-rzv2l:~# 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-rzv2l:~#
```

Format eMMC.

```
root@smarc-rzv2l:~# mkfs.ext4 /dev/mmcblk0p1
root@smarc-rzv2l:~# mkfs.ext4 /dev/mmcblk0p2
```

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

```
root@smarc-rzv2l:~# mount /dev/mmcblk0p1 /mnt/
root@smarc-rzv2l:~# cp Image-smarc-rzv2l.bin /mnt/
root@smarc-rzv2l:~# cp Image-r9a07g054l2-smarc.dtb /mnt/
root@smarc-rzv2l:~# umount /dev/mmcblk0p1
root@smarc-rzv2l:~# mount /dev/mmcblk0p2 /mnt/
root@smarc-rzv2l:~# tar xf /home/root/core-image-weston-smarc-rzv2l.tar.bz2 \
-C /mnt/
root@smarc-rzv2l:~# 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-rzv2l
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/V2L 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-rzv2l.bin; ext4l
oad mmc 0:1 0x48000000 Image-r9a07g054l2-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 How to boot from eSD

In this section, the steps to boot from eSD on RZ/V2L is described.

RZ/V2L can also boot from a micro-SD card, this is boot mode 0 called "Booting from eSD" in the User Manual.

8.4.1 Steps for eSD boot

The steps are as follows.

Step1. Create two Primary partitions on the micro-SD card and format it with FAT and ext4.

Step2. Copy the Kernel image and DTB to #1 partition. Extract the root file system to the #2 partition.

Step3. Write the bootloader to the area of the micro-SD card with the specified seek number.

Step4. Set SW1 and SW11 on the SMARC EVK board for eSD boot.

Step5. Set micro-SD card to the socket on the SMARC module card.

Step6. Power on and boot.

Table 15. eSD boot files for RZ/V2L

RZ/V2L	**Linux kernel**	Image-smarc-rzv2l.bin
	Device tree file	Image-r9a07g054l2-smarc.dtb
	root filesystem	*<image name>*-smarc-rzv2l.tar.bz2
	Boot loader	・bl2_bp_esd-smarc-rzv2l_pmic.bin ・bl2-smarc-rzv2l_pmic.**bin** ・fip-smarc-rzv2l_pmic.bin
	Flash Writer	Flash_Writer_SCIF_RZV2L_SMARC_PMIC_DDR4_2GB_1PCS.mot

8.4.2 Prepare micro SD card

Create two Primary partitions on the micro-SD card and format it with FAT and ext4. Please refer 3.2Write files to the microSD card (not used wic image)

8.4.3 Copy the Kernel image and DTB and Extract the root file system

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
```

Copy kernel and device tree file to the first partition.

```
$ cp $WORK/build/tmp/deploy/images/smarc-rzv2l/<Linux kernel> /media/user/A299-E6A6
$ cp $WORK/build/tmp/deploy/images/smarc-rzv2l/<Devise tree> /media/user/A299-E6A6
```

Expand rootfs to the second partition.

```
$ cd /media/user/rootfs
$ sudo tar jxvf $WORK/build/tmp/deploy/images/smarc-rzv2l/<root filesystem>
```

The file nemes of *<Linux kernel>*, *<Devise tree>*, and *<root filesystem>* are listed in the Table 16.

8.4.4 Writing Bootloader for eSD Boot

Run the following commands to write these files to the microSD card to boot for eSD.

```
$ sudo dd if=bl2_bp_esd-smarc-rzv2l_pmic.bin of=/dev/sdb seek=1 count=1
$ sudo dd if=bl2-smarc-rzv2l_pmic.bin of=/dev/sdb seek=8
$ sudo dd if=fip-smarc-rzv2l_pmic.bin of=/dev/sdb seek=128
```

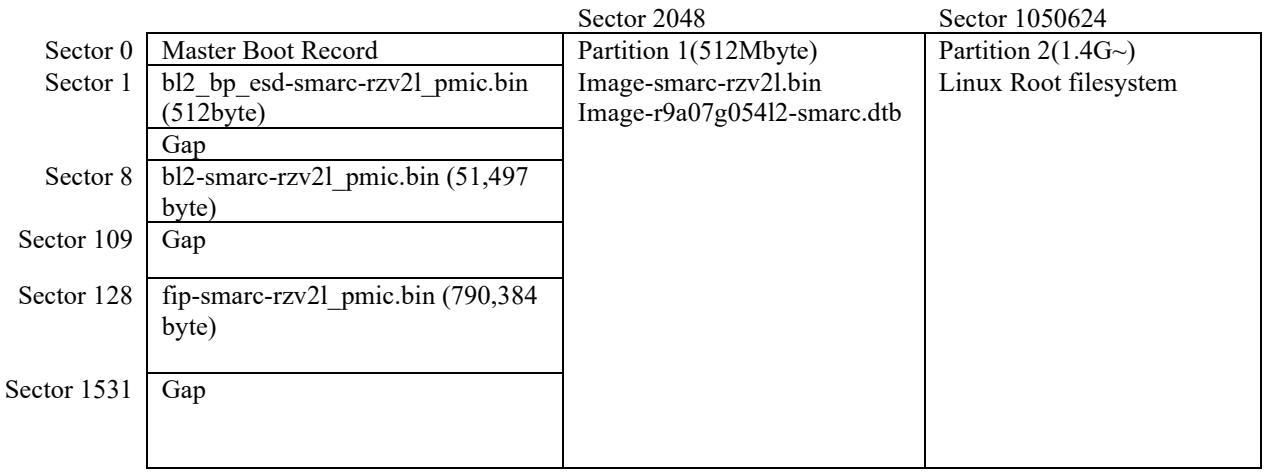


Figure 8. File structure in micro SD card

8.4.5 Set SMARC EVK board for eSD boot

Change the switches to eSD boot on the carrier board acting on SW11 (SW11-1 ON, SW11-2 ON, SW11-3 OFF, SW11-4 ON)



Change SW1 on the module to select micro SD card slot (SW1-1 ON, SW1-2 ON):



Please insert the micro SD card in the SOM module slot.

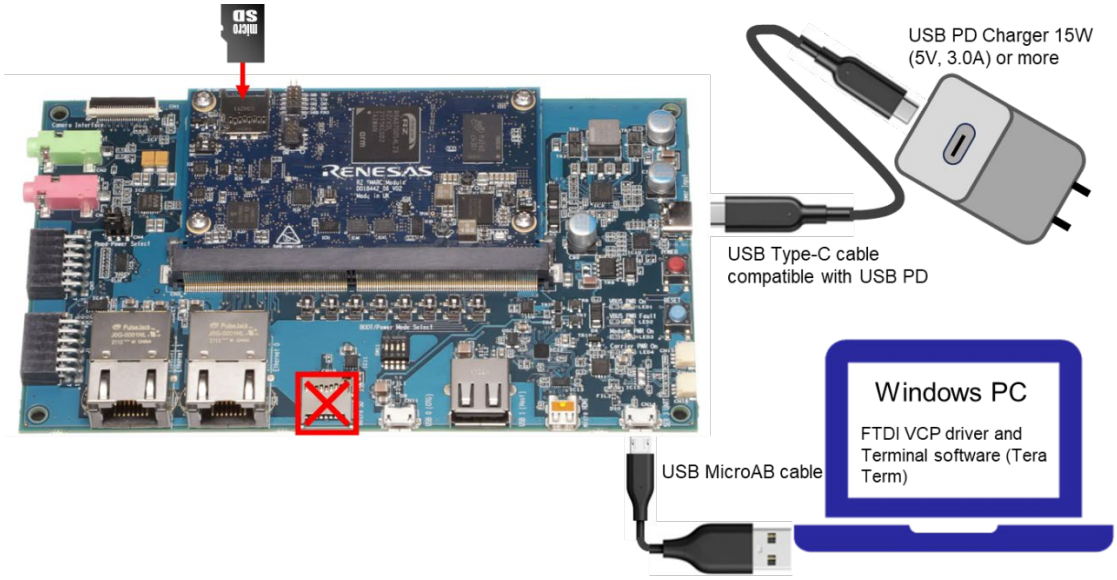


Figure 9. Operating environment for eSD boot

8.4.6 Power on and boot

After press the POWER button(SW9) to turn on the power and RESET button(SW10), Linux will be booted from eSD.

```
NOTICE:  BL2: v2.9(release):cc18695-dirty
NOTICE:  BL2: Built : 04:12:39, Dec 14 2023
NOTICE:  BL2: SD boot from partition 0
NOTICE:  BL2: Load dst=0x1f840 src=(p:0)0x10000(128) len=0x10(1)
NOTICE:  BL2: SD boot from partition 0
NOTICE:  BL2: Load dst=0x1f9a0 src=(p:0)0x10010(128) len=0x28(1)
NOTICE:  BL2: SD boot from partition 0
NOTICE:  BL2: Load dst=0x44000000 src=(p:0)0x10090(128) len=0x6069(49)
NOTICE:  BL2: SD boot from partition 0
```

```
NOTICE:  BL2: Load dst=0x1f840 src=(p:0)0x10000(128) len=0x10(1)
NOTICE:  BL2: SD boot from partition 0
NOTICE:  BL2: Load dst=0x1f9a0 src=(p:0)0x10010(128) len=0x28(1)
NOTICE:  BL2: Load dst=0x1f9a0 src=(p:0)0x10038(128) len=0x28(1)
NOTICE:  BL2: SD boot from partition 0
NOTICE:  BL2: Load dst=0x50000000 src=(p:0)0x16100(176) len=0xbae68(1496)
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.9(release):cc18695-dirty
NOTICE:  BL31: Built : 04:12:39, Dec 14 2023


U-Boot 2021.10 (Dec 15 2023 - 06:47:44 +0000)

CPU:   Renesas Electronics CPU rev 1.0
Model: smarc-rzv2l
DRAM:  1.9 GiB
WDT:   watchdog@0000000012800800
WDT:   Started with servicing (60s timeout)
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from MMC... *** Warning - bad CRC, using default environment

In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
U-boot WDT started!
Net:
Error: ethernet@11c20000 address not set.
No ethernet found.

Hit any key to stop autoboot:  0
## Resetting to default environment
Card did not respond to voltage select! : -110
20070912 bytes read in 1655 ms (11.6 MiB/s)
39353 bytes read in 6 ms (6.3 MiB/s)
Error: Bad gzipped data
Moving Image from 0x48080000 to 0x48200000, end=495a0000
## Flattened Device Tree blob at 48000000
   Booting using the fdt blob at 0x48000000
   Loading Device Tree to 0000000057ff3000, end 0000000057fff9b8 ... OK
Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
[    0.000000] Linux version 5.10.184-cip36-yocto-standard (oe-user@oe-host) (aarc
 :
 :
Poky (Yocto Project Reference Distro) 3.1.21 smarc-rzv2l ttySC0

BSP: RZV2L/RZV2L-SMARC-EVK/3.0.5-update3
LSI: RZV2L
Version: 3.0.5-update3
smarc-rzv2l login: root
root@smarc-rzv2l:~#
```

9. Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Jul. 31, 2023	-	First edition issued.
1.01	Dec. 27, 2023	15	Add how to use wic image file.
		50	Add how to boot from eSD.

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.