Reference Boards of RZ/G1H, RZ/G1M, RZ/G1N, and RZ/G1E

R01US0648EJ0103
Rev.1.03
May 31, 2024

Linux Start-up Guide

Introduction

This document provides a guide to prepare RZ/G1 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/G1 Group reference boards

- iWave RZ/G1H-PF Qseven Development Platform R2.1, R4.0
- iWave RZ/G1M-PF Qseven Development Platform R2.0, R5.0
- iWave RZ/G1N-PF Qseven Development Platform R3.4
- iWave RZ/G1E-PF SODIMM Development Platform R3.1, R4.0

These boards are provided by iWave Systems Technologies Pvt. Ltd

Target Software

- RZ/G Verified Linux Package version 3.0.6 or later. (hereinafter referred to as “VLP/G”)

Contents

1. Environment Requirement

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

A Linux PC is required for building the software.

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

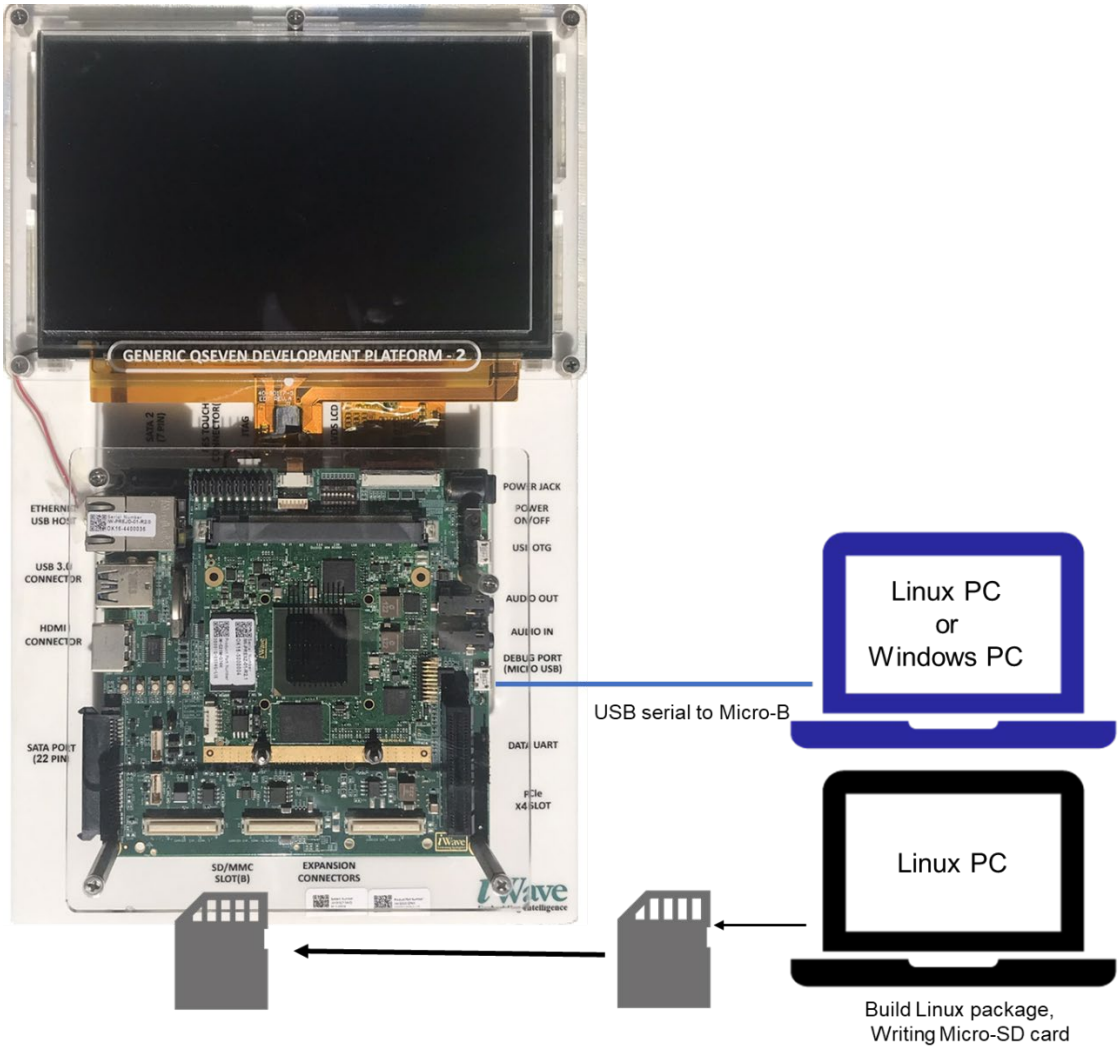


Figure 1. Recommend environment.

Note: The board shown in Figure 1 is RZ/G1H, but RZ/G1M, N, E has the same structure.

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

Equipment		Description
Linux Host PC		Used as build/debug environment 100GB free space on HDD 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
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/G supports can be built on an “offline” environment.
The word “offline” means an isolated environment which does not connect to any network. Since VLP/G includes all necessary source codes of OSS except for the Linux kernel, VLP/G 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

Below are not available in the “offline” environment. Please connect your Linux Host PC to the internet.

- Preparing a Linux Host PC

2. Build Instructions

2.1 Required Host OS

⚠ The VLP/G is only built in **Ubuntu 20.04**

Ubuntu 20.04 is required to build the VLP/G. This is because it was the only host operating system tested and is a specific requirement for Yocto 3.1 (dunfell). Using Ubuntu 22.04 is not supported.

However, you can build the BSP inside a Docker container running Ubuntu 20.04. Instructions for creating this Docker container can be found here:

- https://github.com/renesas-rz/docker_setup

2.2 Building images to run on the board

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 \
bmap-tools
```

Please refer to the URL below for detailed information:

- https://docs.yoctoproject.org/3.1.31/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 ~/rzg_vlp_<package version>
$ cd ~/rzg_vlp_<package version>
$ cp ../<package download directory>/*.zip .
$ unzip ./RTK0EF0045Z0021AZJ-<package version>.zip
$ tar zxvf ./RTK0EF0045Z0021AZJ-<package version>/rzg_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.

Note) <package version>: e.g v3.0.6

(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-rzg1/docs/template/conf/ source \
poky/oe-init-build-env build
```

(3) 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_rzg_<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 "~/rzg_vlp_<package version>/build/conf/local.conf" as below:

```
BB_NO_NETWORK = "1"
```

To change BB_NO_NETWORK from "0" to "1".

(4) 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=<board> bitbake core-image-<target>
```

<board> can be selected by referring to the **Table** 2.

Table 2. List of the platforms and the boards

Renesas MPU	Board
RZ/G1H	iwg21m
RZ/G1M	iwg20m-g1m
RZ/G1N	iwg20m-g1n
RZ/G1E	iwg22m

<target> can be selected in below. Please refer to the **Table** 3 for supported image details.

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

Table 3. Supported images of VLP/G

Image name	Target devices	Purpose
core-image-minimal	RZ/G1H, M, N, E	Minimal set of components
core-image-bsp	RZ/G1H, M, N, E	Minimal set of components plus audio support and some useful tools

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 the **Table** 4 will be generated by the bitbake command and will be located in the **`build/tmp/deploy/images`** directory.

Table 4. Image files for RZ/G1H, RZ/G1M, RZ/G1N, and RZ/G1E

RZ/G1H V4.0 (*)	**Linux kernel**	uImage-iwg21m.bin
	root filesystem	*<image name>*-iwg21m.tar.bz2
	u-boot	u-boot-iwg21m.bin
	Device tree file	uImage-r8a7742-iwg21d-q7.dtb uImage-r8a7742-iwg21d-q7-dbcm-ca.dtb
RZ/G1M V5.0 (*)	**Linux kernel**	uImage-iwg20m-g1m.bin
	root filesystem	*<image name>*-iwg20m-g1m.tar.bz2
	u-boot	u-boot-iwg20m-g1m.bin
	Device tree file	uImage-r8a7743-iwg20d-q7.dtb uImage-r8a7743-iwg20d-q7-dbcm-ca.dtb
RZ/G1N V5.0 (*)	**Linux kernel**	uImage-iwg20m-g1n.bin
	root filesystem	*<image name>*- iwg20mg1n.tar.bz2
	u-boot	u-boot-iwg20m-g1n.bin
	Device tree file	uImage-r8a7744-iwg20d-q7.dtb uImage-r8a7744-iwg20d-q7-dbcm-ca.dtb
RZ/G1E V4.0 (*)	**Linux kernel**	uImage-iwg20m-g1m.bin
	root filesystem	*<image name>*- iwg22m.tar.bz2
	u-boot	u-boot-iwg22m.bin
	Device tree file	uImage-r8a7745-iwg22dsodimm.dtb

<image name> will be the name used in the step (5).

2.3 Note

(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 the step (4) of the section 2.2, create a new working directory and prepare the new build 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 the step (4) of the section 2.2, create a new working directory and prepare the new build 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 out and add the following lines to disable CIP Core Packages in the `local.conf`:

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

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

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

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

Table 5. Versions of all Buster Debian packages

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

(3) 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 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

Please prepare a micro SD card of 4GB or more.

You can prepare the micro SD card by the following methods in this section.

3.1 Create a microSD card for boot Linux

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

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

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

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

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

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

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

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

(4) Unmount automatically mounted microSD card partitions

If necessary, unmount all mounted microSD card partitions.

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

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

(5) Change the partition table

microSD card needs two partitions as listed in Table 6.

Table 6. Partitions of microSD card

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

Set the partition table using the fdisk command like this.

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

Command (m for help): o

Created a new DOS disklabel with disk identifier 0x6b6aac6e.

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

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

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

The signature will be removed by a write command.

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

3.2 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/G1 Evaluation board Kit, copy kernel and device tree file to the first partition like as below.

```
$ cp $WORK/build/tmp/deploy/images/<board>/uImage /media/user/A299-E6A6
$ cp $WORK/build/tmp/deploy/images/<board>/uImage-<Devise tree> /media/user/A299-E6A6
$ cp $WORK/build/tmp/deploy/images/<board>/uboot.bin /media/user/A299-E6A6
```

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

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

The file names of Linux file is listed in the Table 4.

Table 7. File and directory in the micro SD card

Type/Number	Size	Filesystem	Contents
Primary #1	500MB (minimum 128MB)	FAT32	u-boot.bin uImage-<board>.bin uImage-<Device tree>.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

This section describes how to setup using Windows PC. For describes how to setup using Linux PC, please refer to 8.1.

4.1 Preparation of Hardware and Software

4.1.1 Preparation of Hardware

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

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

- AC Adapter 12V2A with EIAJ3 connector(Any AC Adapter)
- USB Type-microB cable (Any cables)
- HDMI cable (Any cables)

PC Installed Silicon Labs VCP driver and Terminal software (Tera Term)

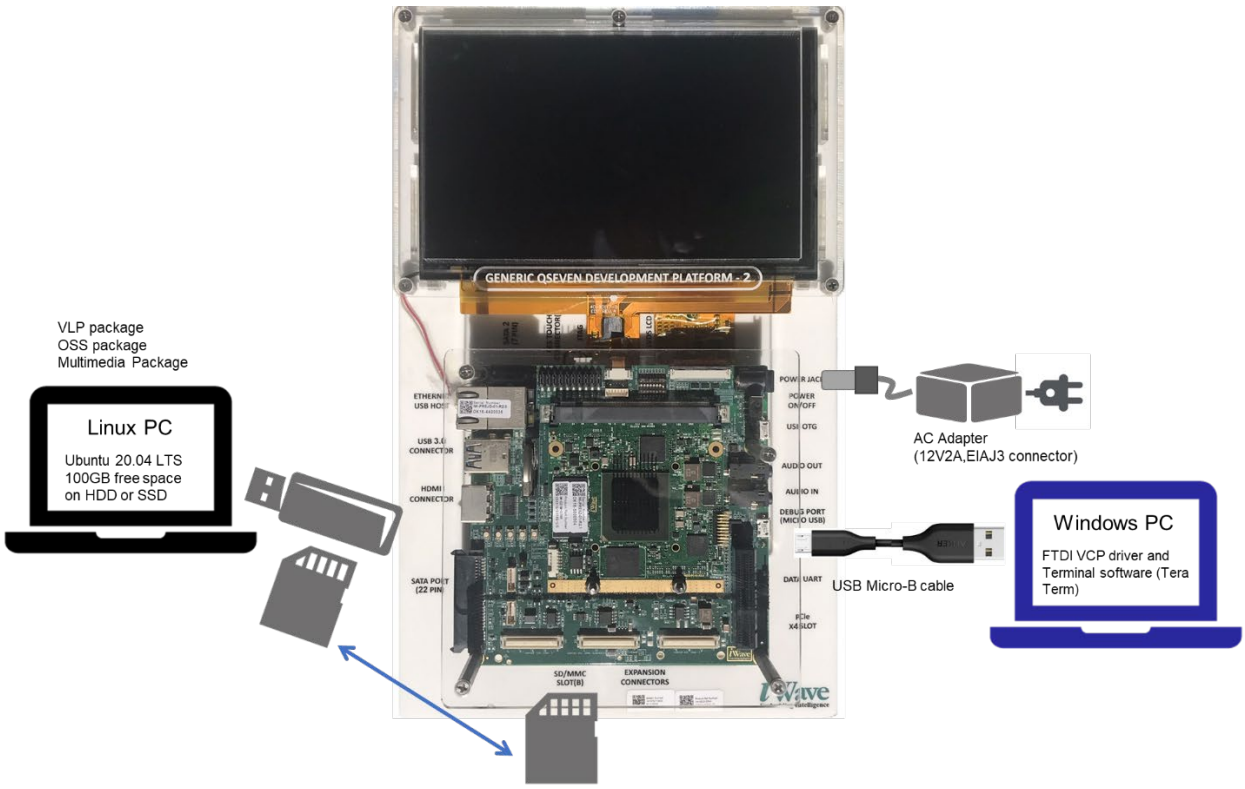


Figure 2. Operating environment

4.1.2 Building files to write

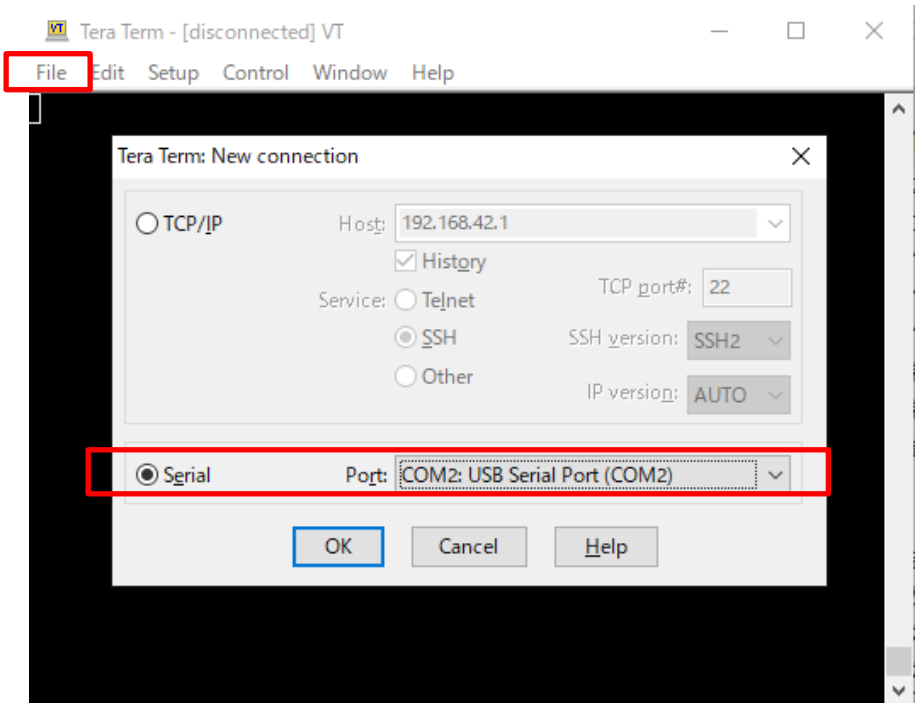
This board uses the files below as a bootloader. Please build them according to the Release Note and copy these files to the PC which runs a serial terminal software.

- u-boot.bin

4.1.3 Settings

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

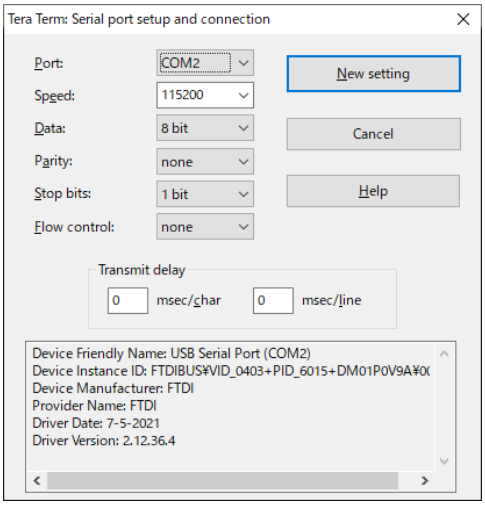
Set the settings about serial communication protocol on a terminal software as below:



1. 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

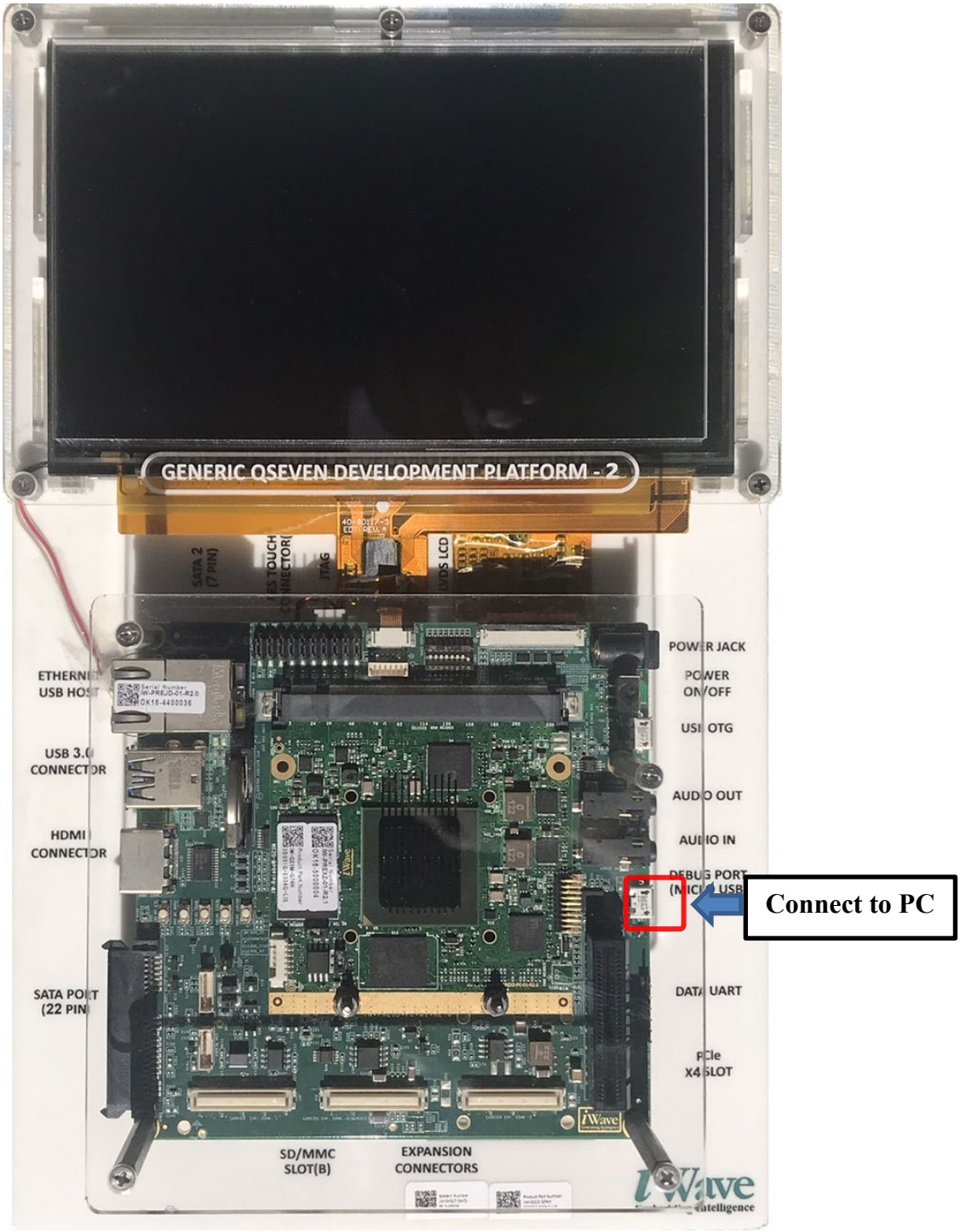
Select the “Setup” > “Terminal” to set the new-line code.

- New-line:”CR” or “AUTO”



4.1.4 How to use debug serial (console ouput)

Please connect USB Type-microB cable to J3.



J3:USB Type-microB Connector

Figure 3. Connecting console for debug

4.2 Startup Procedure

4.2.1 Power supply

1. Connect AC adapter to Power Jack Connector (J4).

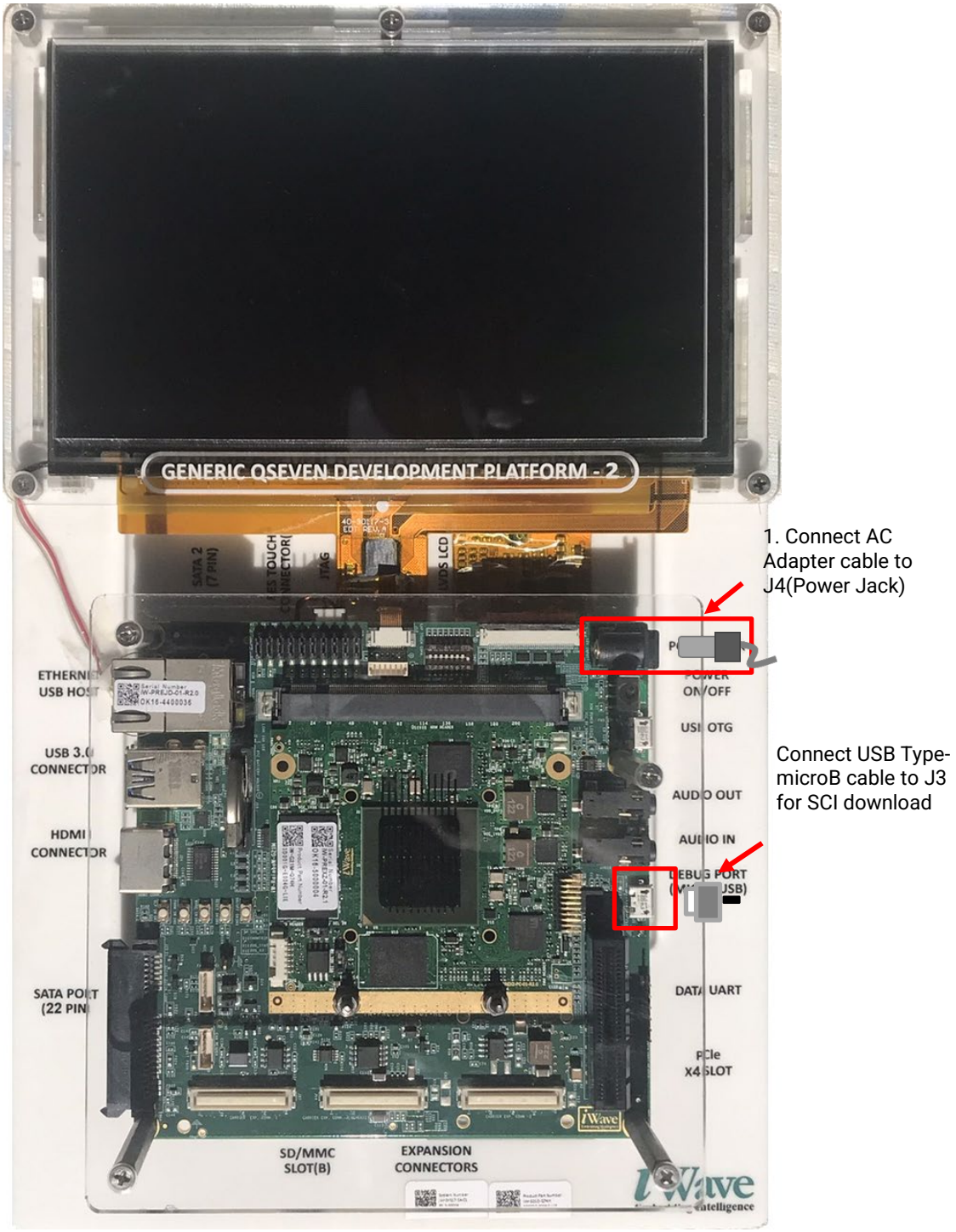


Figure 4. Connecting Power Supply

2. Toggle the power switch (SW1) to turn on the power.
3. Power LED lighs up.

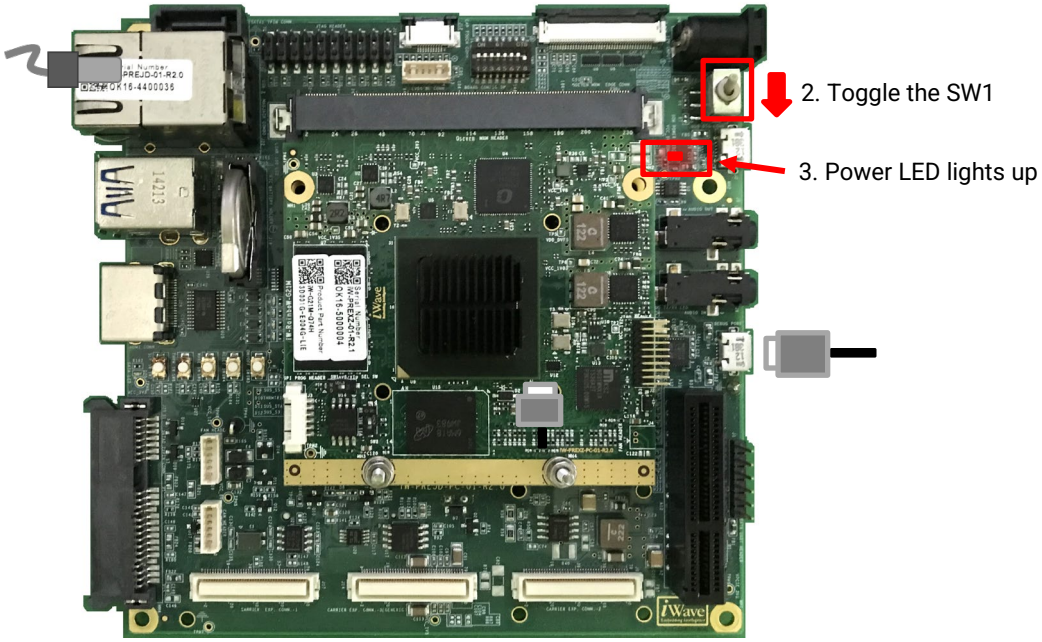


Figure 5. Power ON

```
iW-RainboW-G21M SPI_LOADER V0.31 2015.11.10
 DEVICE SST25VF016B

U-Boot 2013.01.01-gbc72a74 (Dec 17 2020 - 05:09:31)

CPU: Renesas Electronics R8A7742 rev 3.0
Board: RZ/G1H iW-RainboW-G21M-Q7

Reset Cause : POR
DRAM:  2 GiB
MMC:   sh-sdhi: 0, sh_mmcif: 1
SF: Detected SST25VF016B with page size 4 KiB, total 2 MiB
In:    serial
Out:   serial
Err:   serial

Board Info:
        SOM Version     : iW-PREXZ-AP-01-R2.1

Net:   ether_avb
Hit any key to stop autoboot:  0
sh-sdhi: Cmd(d'1) err
sh-sdhi: cmdidx = 1
Card did not respond to voltage select!
** Bad device mmc 0 **
sh-sdhi: Cmd(d'1) err
sh-sdhi: cmdidx = 1
Card did not respond to voltage select!
** Bad device mmc 0 **
Wrong Image Format for bootm command
ERROR: can't get kernel image!
iWave-G21M >
```

4.2.2 Replacing boot loader

RZ/G evaluation kit boards ship with boot loaders are written. RZ/G series use SPI loader and u-boot as boot loaders. Generally, SPI loader is not necessary to be replaced. Please replace u-boot to the file you built to enable some devices. Note that this procedure overwrites old u-boot. Be sure not to use wrong image file or wrong address. Once you fail to write new image by mistake, you can't do again the same procedure.

(1) Load u-boot image to RAM

Load the image like the steps to load kernel image. Put the image prior to this step to the root directory of the same partition as the kernel on an SD card.

```
iWave-G21M > fatload mmc 0:1 0x50000000 u-boot.bin
```

Note) If "mmc 0:1" does not work, try to use "mmc 1:1" instead. (<device number>:<partition number>)

(2) Write the image

Write the image from RAM to SPI FLASH.

```
iWave-G21M > sf probe
iWave-G21M > sf erase 0x20000 0x40000
iWave-G21M > sf write 0x50000000 0x20000 0x40000
```

(3) Set environment

```
iWave-G21M > env default -a
iWave-G21M > setenv loadaddr 0x48008000
iWave-G21M > setenv fdt_addr 0x4a000000
iWave-G21M > setenv fdt_high 0xffffffff
iWave-G21M > setenv bootargs_base 'console=ttySC2,115200n8 debug ignore_loglevel vmall
oc=384M'
iWave-G21M > setenv bootargs '${bootargs_base} root=/dev/mmcblk0p2 rootwait rootfstype
=ext4 rw'
iWave-G21M > setenv bootcmd 'mmc dev 0 ; fatload mmc 0:1 0x48008000 <KERNEL IMAGE> ; f
atload mmc 0:1 0x4a000000 <DEVICE TREE> ; bootm 0x48008000 - 0x4a000000'
iWave-G21M > saveenv
```

Note) Please replace <KERNEL IMAGE> and <DEVICE TREE> with the actual file names.

Once you've finished writing, you can boot Linux using the "boot" command or by simply powering off and then back on the system.

5. Booting and Running Linux

Set SD card to slot on carry board.

[Top view] [Bottom view]

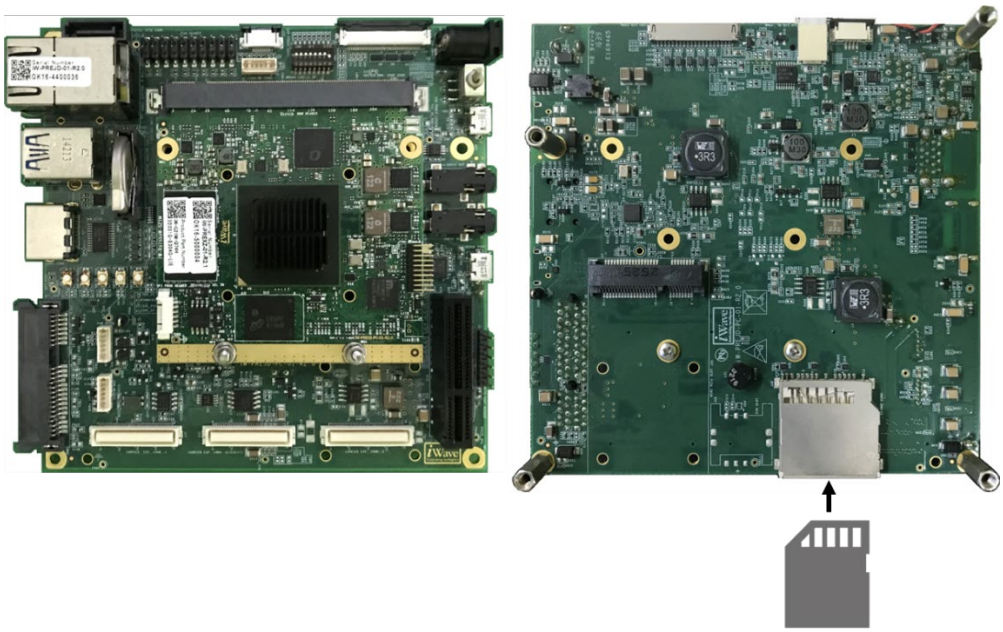


Figure 6. Set SD card to carry board

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.

```
iW-RainboW-G21M SPI_LOADER V0.31 2015.11.10
 DEVICE SST25VF016B


U-Boot 2013.01.01-g06622d5 (Jun 16 2023 - 15:19:03)

CPU: Renesas Electronics R8A7742 rev 3.0
Board: RZ/G1H iW-RainboW-G21M-Q7

Reset Cause : POR
DRAM:  2 GiB
MMC:   sh-sdhi: 0, sh_mmcif: 1
SF: Detected SST25VF016B with page size 4 KiB, total 2 MiB
In:    serial
Out:   serial
Err:   serial
```

```
:
:
Poky (Yocto Project Reference Distro) 3.1.31 iwg21m ttySC2

BSP: RZG1H/iWave RZ/G1H-PF Development Kit/3.0.6
LSI: RZG1H
Version: 3.0.6
iwg21m login:root
root@iwg21m:~#
```

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@iwg21m:~# 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, you can see "reboot: System halted".

Step 3. Turn off the power switch on the board

Turn off SW1.

6. Building the SDK

To build Software Development Kit (SDK), run the commands below after the steps (1) – (6) of section **2.1** 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 bsp applications:

```
$ cd ~/rzg_vlp_<package version>/build
$ MACHINE=<board> 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-cortexa15hf-neon-vfpv4-<board>-toolchain-3.
1.31.sh
```

- iWave RZ/G1H-PF iwg21m
- iWave RZ/G1M-PF iwg20m-g1m
- iWave RZ/G1N-PF iwg20m-g1n
- iWave RZ/G1E-PF iwg22m

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 ~/rzg_vlp_<package version>/build
$ MACHINE=<board> bitbake core-image-bsp -c cleanall
$ MACHINE=<board> bitbake core-image-bsp
```

For building general applications:

```
$ MACHINE=<board> bitbake core-image-bsp -c populate_sdk
```

7. Application Building and Running

This chapter explains how to make and run an application for RZ/G1H,M,N,E 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-cortexa15hf-neon-vfpv4-<board>-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 ./rzg_vlp_<package version>/build/tmp/deploy/sdk/poky-glibc-x86_64-core-imag
e-bsp-cortexa15hf-neon-vfpv4-iwg21m-toolchain-x.x.xx.sh
Poky (Yocto Project Reference Distro) SDK installer version x.x.xx
==========================================================================
Enter target directory for SDK (default: /opt/poky/x.x.xx):
You are about to install the SDK to "/opt/poky/x.x.xx". 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/x.x.xx/environment-setup-cortexa15hf-neon-vfpv4-poky-linux-gnueabi
```

Step 2. Set up cross-compile environment:

```
$ source /<Location in which SDK is extracted>/environment-setup-cortexa15hf-neon-vfpv
4-poky-linux-gnueabi
```

Note:

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

```
$ source /opt/poky/x.x.xx/environment-setup-cortexa15hf-neon-vfpv4-poky-linux-gnueabi
```

7.1.2 How to build Linux application

Step 1. Make a work 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):
  $(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
arm-poky-linux-gnueabi-gcc  -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a15 -fstack
-protector-strong  -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-secur
ity --sysroot=/opt/poky/3.1.31/sysroots/cortexa15hf-neon-vfpv4-poky-linux-gnueabi -o l
inux-helloworld main.c -g -O0 -Wall -DDEBUG_LOG
renesas@49d1d5882435:/mnt/host/hello$ ls -l
total 24
-rwxr-xr-x 1 renesas renesas 13872 Sep 27 22:14 linux-helloworld
-rw-r--r-- 1 renesas renesas   149 Aug 30 23:25 main.c
-rw-r--r-- 1 renesas renesas   253 Aug 30 23:30 makefile
```

Step 4. 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/G2H,M,N,E Evaluation Board Kit and start the system. After booting, run the sample application with the following command.

```
BSP: RZG1H/iWave/3.0.6
LSI: RZG1H
Version: 3.0.6
iwg21m login: random: nonblocking pool is initialized
root
Simple mixer control 'DVC In',0
  Capabilities: cvolume
  Capture channels: Front Left - Front Right
  Limits: Capture 0 - 8388607
  Front Left: Capture 4194304 [50%]
  Front Right: Capture 4194304 [50%]
Simple mixer control 'DVC Out',0
  Capabilities: pvolume
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 8388607
  Mono:
  Front Left: Playback 4194304 [50%]
  Front Right: Playback 4194304 [50%]
root@iwg21m:~# /usr/bin/linux-helloworld

Hello World
root@iwg21m:~#
```

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

8. Appendix

8.1 Booting Setup with 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.

Next is replacing boot loader step. Please refer to 4.2.2Replacing boot loader.

8.2 Device drivers

The following drivers are supported: For details, refer to each manual included in the BSP manual set.

Table 8. Support device drivers

Device Driver	Documents
Kernel Core	R01US0656EJxxxx_KernelCore_UME.pdf
GPIO	R01US0657EJxxxx_GPIO_UME.pdf
Direct Memory Access Controller (DMAC)	R01US0658EJxxxx_DMAE_UME.pdf
Boot loader for iW-RainboW Platform	R01US0659EJxxxx_UBOOT_UME.pdf
Audio Interface (Serial Sound Interface Unit (SSIU))	R01US0666EJxxxx_AUDIO_UME.pdf
Gigabit Ethernet Interface	R01US0667EJxxxx_GEther_UME.pdf
Ethernet Interface	R01US0668EJxxxx_Ether_UME.pdf
PCI Express Controller	R01US0669EJxxxx_PCIEC_UME.pdf
Serial Communication Interface with FIFO (SCIF)	R01US0670EJxxxx_SCIF_UME.pdf
High Speed Serial Communication Interface with FIFO (HSCI-F)	R01US0671EJxxxx_HSCIF_UME.pdf
I2C Bus Interface	R01US0672EJxxxx_I2C_UME.pdf
Clock-Synchronized Serial Interface with FIFO (MSIOF)	R01US0673EJxxxx_MSIOF_UME.pdf
Quad Serial Peripheral Interface	R01US0674EJxxxx_QSPI_UME.pdf
SD Card Interface	R01US0675EJxxxx_SD_UME.pdf
Multimedia Card Interface	R01US0676EJxxxx_MMC_UME.pdf
Serial-ATA Interface	R01US0677EJxxxx_SATA_UME.pdf
USB 2.0/3.0 Host	R01US0678EJxxxx_USB_UME.pdf
USB 2.0 Function	R01US0679EJxxxx_USBF_UME.pdf

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

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sep. 29, 2023	–	First edition VLP/G v3.0.5.
1.01	Nov. 30, 2023	5	Change “TEMPLATECONF” command parameters.
1.02	Jan. 31, 2024	22	Change boot description after bootloader update.
1.03	Apr. 24, 2024	5	Add “2.1 Required Host OS” section.
1.04	May 31, 2024	-	VLP/G v3.0.6-update1
		29	Add “8.1 Booting Setup with Ubuntu PC” section.
1.05	Jul. 31, 2024	-	VLP/G v3.0.6-update3

Website and Support

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

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