User’s Manual

System Release Package

User’s Manual: Hardware and Software

Renesas Microprocessor
RZ G/V/H Series

Renesas Electronics
www.renesas.com

Rev.3.00 Sep.25.25

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Trademarks (continued)

For the “Cortex” notation, it is used as follows;
— Arm® Cortex®-A55
— Arm® Cortex®-M33
Note that after this page, they may be noted as Cortex-A55 and Cortex-M33 respectively.

Examples of trademark or registered trademark used in the RZ/G2L SMARC Module Board RTK9744L23C01000BE User’s Manual: Hardware;
CoreSight™: CoreSight is a trademark of Arm Limited.
MIPI®: MIPI is a registered trademark of MIPI Alliance, Inc.
eMMC™: eMMC is a trademark of MultiMediaCard Association.

Note that in each section of the Manual, trademark notation of ® and TM may be omitted.
All other trademarks and registered trademarks are the property of their respective owners.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

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

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Renesas RZ Family / RZ/G/V/H Series

Renesas System Release Package

Introduction

This user manual describes the unified system release package. The system release package contains supported hardware and software.

The result is a consistent experience across the different platforms. This streamlines the development effort for user applications.

Package Contents

The system release package contains the following:

- Multiple Images that are geared to general baseline use-cases.
- Yocto build scripts.
- Host side tools.
- Environmental files.
- SDKs for all images
- Documentation which includes
- User manual
- Copyright & License information

Features

The following are the general features of the system release package.

- Architected to support multiple platforms by the same image and tools over time.
- Common frameworks
- Open-source packages using GPLv2 and GPLv3 packages
- Carefully considered base images that allow for a quick starting point to build a product.
- Complete set of features working out of the box.
- Seamless out of box experience.
- Automated Yocto build scripts that can rebuild the entire package with only a few commands.
- Host tools to flash the firmware in multiple processes.
- Tools supporting both Linux and Windows workflows.
- Docker friendly build scripts.
- Extensive documentation covering the hardware, software and application development and deployment.

Contents

Glossary

Terms	Description
802.11 - Wi-Fi	The technical name of the standard specification for Wi-Fi is 802.11. This is also the working group that develops and maintains the standards for Wi-Fi that everyone conforms to.
ADC – Analog to digital converter	A hardware unit that converts an input analog signal to a digital value by measuring its immediate voltage at a fixed resolution.
BSP – Board Support Package	BSP is an essential software package that has bootloaders, Linux kernel, a minimal user space and programming tools, allowing the device to boot. This core software allows the system to boot into an operating system, enables all the features and allows application development.
CAN – Controller area network	This is a standardized communication protocol used widely on automotive and aerospace systems. It connects various ECU's known as nodes and uses two wires / lines as a pair carrying differential signals. This method of signaling allows long length cables to interface different systems on the machine with reliable signals. The CAN protocol has multiple specifications and is an ISO standard. It supports flexible data rates reaching as high as 8Mbps. Most automobiles have CAN networks in them, and it is a part of OBD-2 specification which is mandatory law in most of the world for automotive machines like cars.
DAC – Digital to analog converter	A hardware unit that takes digital value and exerts a corresponding analog voltage on an output line.
Firmware	For the scope of this document, the term 'firmware' refers to the low-level software that runs before an OS takes over. This includes arm trust zone, optee & u-boot at the very least. It also refers to the standalone binaries that run on the embedded real-time core like the CM33.
I2C - Inter Integrated circuit protocol:	This is a communication protocol used to implement digital communication between two devices (chips / board) using only two wires. It is a standardized specification and is used widely to implement low to medium data rate data transfers both among devices on the same circuit board as well as external add on peripheral boards. I2C can be implemented across a few meters in distance. I2C is half duplex meaning only one device can communicate at a time. Speeds range from 100 Kbps to 3Mbps while 100 / 400 Kbps are the typical operating mode. The other major advantage of this protocol is that it allows many devices to be on the same two lines reducing the cost of the interfacing. This is ideal when there are many devices like sensors that transfer limited amounts of data periodically. I2C can support up to 127 independent directly addressable devices on the same channel.
IEEE- Institute of Electrical and Electronics Engineers	IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity. It is a major technical organization covering vast fields of engineering and a major standards organization.
MCU – Micro controller unit	A micro controller unit is a self-contained unit that has the core processing as well as core memory within the same device. It often contains the core software programmed into the chip itself. This allows the device to start executing with minimal external devices / circuitry. Some microcontrollers can be powered on a mere breadboard.
MPU – Micro processing unit	An MPU is a processing unit: a CPU that contains only the processing core and interfaces for external peripherals. A microprocessor is usually a powerful CPU in its class. However, it requires a very large number of external circuitries to achieve its functionality like external memory, disk drives, etc.
PMIC – Power management IC	This is a specific chip on the board that manages multiple power supply lines at various levels. It manages the respective supplies along with sequences which control power on and power off cycles.

SBC – Single board computer	**It is a standard term that means a tiny computer in the form factor of a single circuit board usually just inches in area. This board is self-sufficient in every way and can give you a usable computer with just a power supply, keyboard, mouse, and display.**
SiP – System in Package	SiP is a device where multiple silicon IP's are combined to form a single device. It is one of the densest chips where the external devices like flash memory, DDR RAM and even Wi-Fi module are all packaged into a single chip. These are used in very niche application that require ultra small size and low thermal requirement.
SoC- System on Chip	A system on chip is a complete hardware platform packaged on to a single chip. It contains the CPU, internal fast memory, interrupt controllers, pin controllers, ROM memory, and a few other peripherals and sensors; all packaged into the same IC. An SoC despite the high level of integration does not necessarily power on and run by itself. Microcontrollers are often independent SoC's that can work on their own. However, SoC's often combine MPU's and MCU's into the same chip. This allows very powerful systems to be built in a compact form factor but requires external supporting peripherals like DDR RAM and flash memory and power management IC's.
SMARC	SMARC is a standardized interface to connect a core module to a common platform carrier board. It's basically a connector with specific pin outs. It allows the reuse of the carrier board to support a variety of modules for evaluation.
SPI - Serial Peripheral interface	SPI is another standard interface used to interface other devices on the board or attaching peripheral boards. It specifies 3 wires / lines to achieve fast full duplex data transfer. Two devices can send / receive data at the same time in this protocol. The protocol is also a high-speed protocol where typical operating speeds start at 5Mbps and go over 50Mbps. This high speed allows interfacing high speed devices like memory, Wi-Fi, subsystems made of independent microcontrollers, etc. While only 3 lines are needed to interface two devices, a fourth line is used as a device selector allowing multiple devices to share the same interface. However, only two devices may communicate at a time.

1. Overview

The Renesas System Release Package is a unified software package that aims to provide an easy-to-use yet comprehensive software platform for the Renesas RZ series of MPU-based boards. It aims to provide fully functional base images for supported reference designs, along with easy-to-use development and programming tools that allow the user to quickly get started on their application development. This package aims to provide a standardized and familiar workflow for a similar experience across a variety of Renesas RZ MPU-based product platforms.

This package provides comprehensive documentation, Quick start guides, multiple Linux-based distribution images, automated tools and scripts, and an ongoing expansion of supported products.

1.1 Supported Distributions

The System Release package supports a set of both Yocto images and custom images to enable the user to start quickly on their embedded end application. The large collection of images in prebuilt format provides a wide set of capabilities. This release focuses on Yocto images.

1.1.1 Yocto Images

This section lists the standard Yocto images, offering a variety of configurations that cater to different embedded use cases. From a minimal bootable environment to fully graphical systems, these images provide the essential building blocks for embedded Linux development.

Table 1. Yocto images

Distribution	Image file	Version	Description
Yocto minimal	core-image-minimal	styhead-5.1.4	A basic image that contains the minimal set of components required to boot the device. It focuses on essential system functions without extra tools or features.
Yocto BSP	core-image-bsp	styhead-5.1.4	Extends core-image-minimal with additional utilities and tools, providing a lightweight environment for system validation, hardware diagnostics, and basic development.
Yocto weston	core-image-weston	styhead-5.1.4	A standard graphical image with Wayland and Weston support for embedded GUI applications.

1.1.2 Renesas Custom Images

This section presents Renesas-specific custom images, which are customized and optimized for Renesas products. These images offer specialized features, including fast booting and tailored environments for both graphical and CLI-based applications. Table 2. Renesas custom images

Distribution	Image file	Version	Description
Renesas CLI Base	renesas-core-image-cli	styhead-5.1.4	Based on core-image-bsp, this image offers a CLI environment for Renesas hardware development without graphical interfaces. Besides the useful tools inherited from the core-image-bsp, this image also contains new packages for SBC (Single Board Computer) development. For example, package managers (apt, dpgk), network utilities for Bluetooth, Wi-Fi.
Renesas Quickboot CLI	renesas-quickboot-cli	styhead-5.1.4	This image has the same system functionality as the renesas-core-image-cli but with Quickboot enabled, allowing for faster boot times and efficient system validation on a CLI environment.
Renesas Weston (Qt6)	renesas-core-image-weston	styhead-5.1.4	Renesas customized core image based on the core-image-weston, with Qt6 framework support (no QT demo apps included). This image offers a full graphical environment for Renesas hardware development and all the useful tools from the renesas-core-image-cli.
Renesas Quickboot Wayland	renesas-quickboot-wayland	styhead-5.1.4	This image has the same system functionality as the renesas-core-image-weston but with Quickboot enabled, allowing for faster boot times and efficient system validation on a graphical environment.

Note: Quickboot is a trade term that refers to the specific optimizations that are performed to achieve ultra-low start-up times in specific images. Depending on the board architecture, the startup time can be as low as 2s. While there is no assurance of the startup time in these images for every platform, these images are the most optimized on our platforms.

1.1.3 Ubuntu Images

This section presents custom Ubuntu-based images tailored for embedded systems, offering a variety of configurations to suit both headless and graphical environments. These images are optimized for performance and ease of use, providing a solid foundation for deploying embedded applications on Renesas platforms.

Table 3. Renesas Ubuntu images

Distribution	Image file	Version	Description
Ubuntu Core	ubuntu-core-image	ubuntu-base-24.04	A minimal, headless Ubuntu image tailored for embedded systems.
Ubuntu LXDE	ubuntu-lxde-image	ubuntu-base-24.04	A lightweight Ubuntu image featuring the LXDE desktop environment, providing a graphical interface while maintaining low resource consumption. This image also includes Qt framework support for GUI development.

1.2 Supported Platforms

Table 4. Supported Platforms

Platform	MPU	OPN	Description
RZ/G2L-SBC	RZ/G2L	US157-G2LSBCPOCZ	RZ/G2L-based Pi-compatible SBC.
RZ/G2L-EVK	RZ/G2L	RTK9744L23S01000BE	Evaluation Board Kit for RZ/G2L MPU
RZ/V2L-EVK	RZ/V2L	RTK9754L23S01000BE	Evaluation Board Kit for RZ/V2L MPU
RZ/V2H-EVK	RZ/V2H	R9A09G057H48GBG#AC0 R9A09G057H48GBG#BC0	Evaluation Board Kit for RZ/V2H MPU

2. Introduction

The system release package provides a unified and consistent experience across multiple RZ platforms by providing prebuilt binaries that are as universal as possible, along with all the tools and documentation necessary to work with these platforms. To enable the platforms, the package contains a variety of images that provide the most common starting points for embedded application development. The workflow envisioned is provided below.

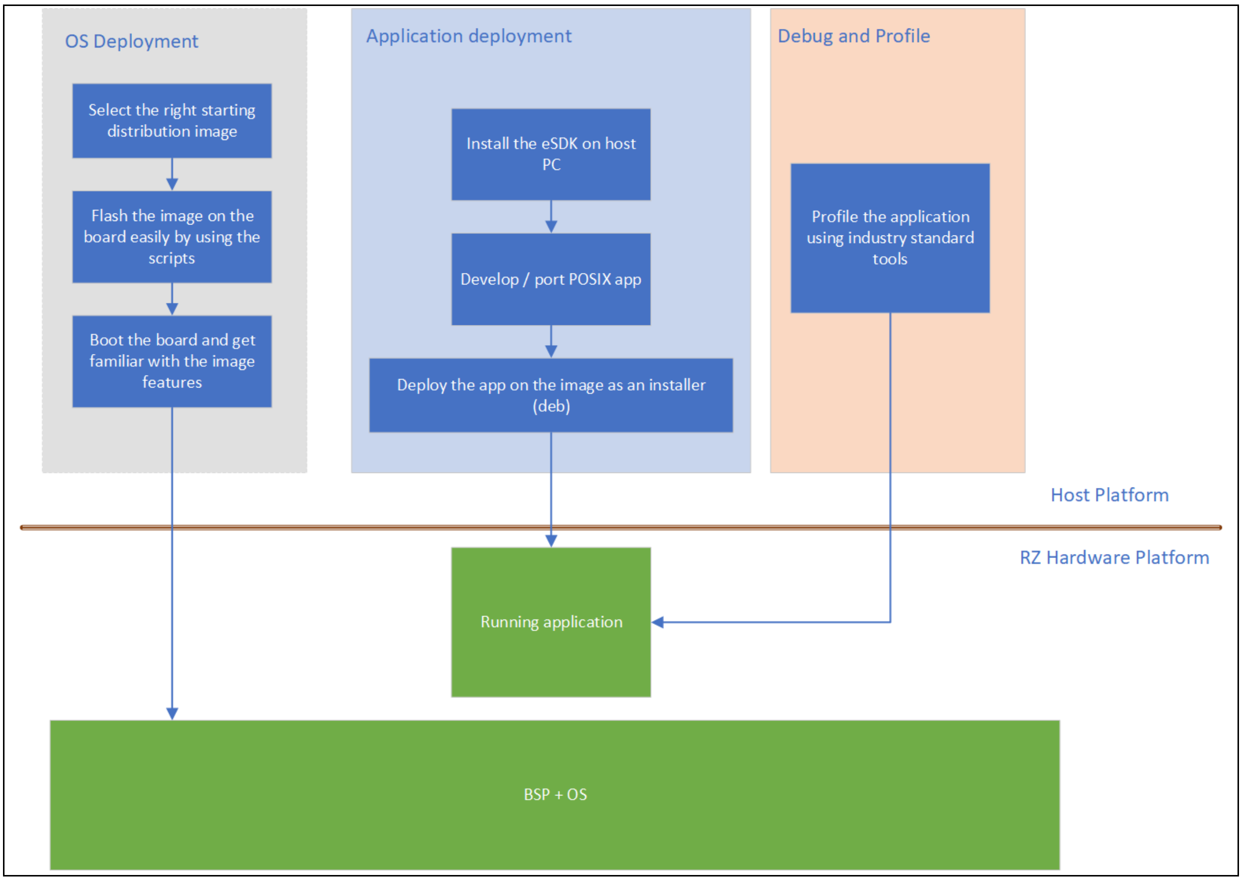

OS Deployment
Select the right starting distribution image
Flash the image on the board easily by using the scripts
Boot the board and get familiar with the image features
Application deployment
Install the eSDK on host PC
Develop / port POSIX app
Deploy the app on the image as an installer (deb)
Debug and Profile
Profile the application using industry standard tools
Host Platform
RZ Hardware Platform
Running application
BSP + OS

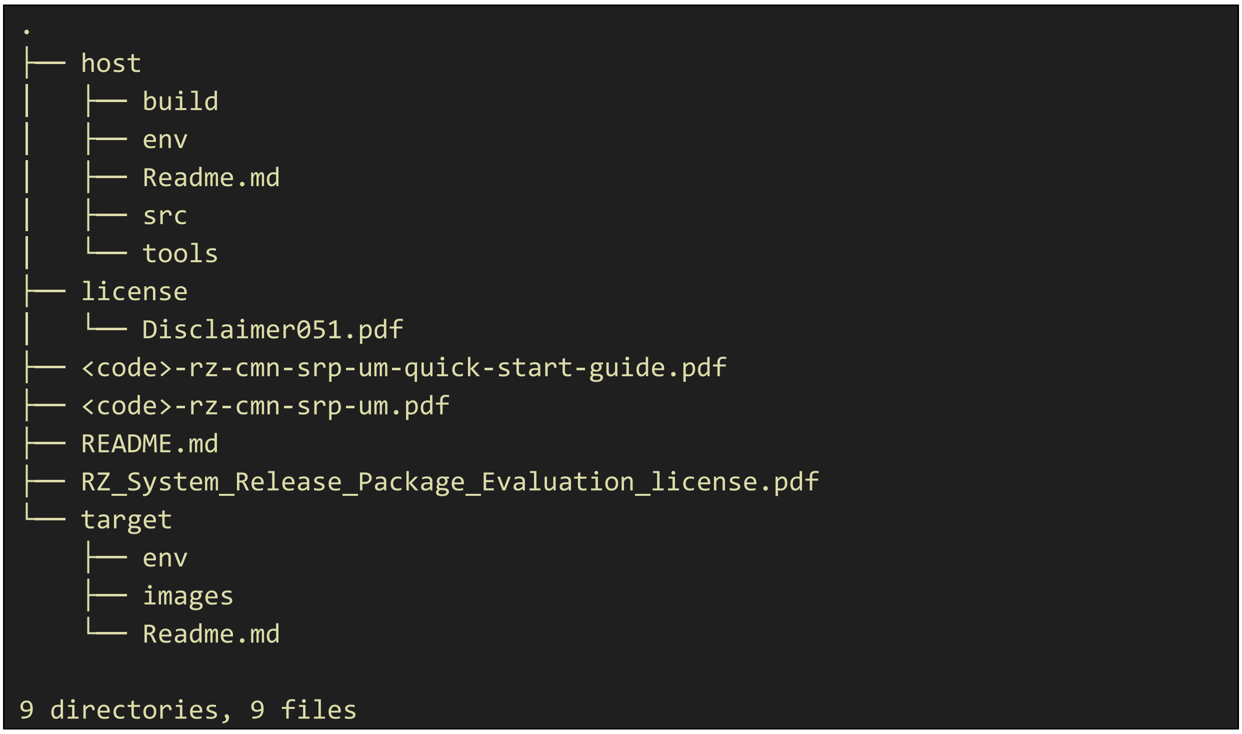
Figure 1. Embedded application workflow for RZ System release package

The package also comes with automated scripts that let users rebuild the entire package on the user end as well as modify the existing images as needed. The eSDK's and usage of the wic file formats for the images allow the tweaking and generation of new images without rebuilding the entire package. This lets users focus on the embedded applications instead of the platform's intricacies.

2.1 Package Hierarchy

The System Release Package is organized into two archives. The first archive is the primary package itself containing the images, build scripts, documentation, etc. The second archive is the SDK archive. The package is arranged into an intuitive file hierarchy that is easy to follow. There are ‘Readme.md’ files at every location to help with understanding of the contents. The Readme.md file at the root of the hierarchy is a comprehensive guide that gives an overview of the entire package and how to use it.

Below is an overview of the package hierarchy, followed by a description of the contents and purpose of each directory/file:



```
.
├── host
│   ├── build
│   ├── env
│   ├── Readme.md
│   ├── src
│   └── tools
├── license
│   └── Disclaimer051.pdf
├── <code>-rz-cmn-srp-um-quick-start-guide.pdf
├── <code>-rz-cmn-srp-um.pdf
├── README.md
├── RZ_System_Release_Package_Evaluation_license.pdf
└── target
    ├── env
    ├── images
    └── Readme.md

9 directories, 9 files
```

host/: This directory holds all the tools, scripts, and artifacts needed on the host machine for building and preparing the system images.

- **build/:** Contains build artifacts (manifests and test data).
  Key files:
  - Manifest file: Files like core-image-bsp-<timestamp>.rootfs.manifest lists the contents of the generated root file system.
  - Test data: Files with the *.testdata.json extension that contains metadata or test results of the said image.
- **env/:** Provides environment configuration files used during the build or runtime.
  Key files:
  - .env Files: Examples include core-image-bsp.env or core-image-minimal.env, which define variables and configuration parameters for different image variants.
- **src/:** Holds build scripts, source code, and patches that are used to build the package.
  Key files:
  - rz-cmn-srp/: The folder that contains artifacts to build Yocto and Ubuntu images.
    - Patches: Located in the patches/ subdirectory, these files (For example, 0001-...patch) apply for necessary modifications.

        - Build scripts: The master script rz_builder.sh automates the build process for both Ubuntu and Yocto packages, handling setup, configuration, and image generation based on user-selected build options.
        - Configuration files: site.conf, which is used to set up a specific build tag.
        - config.json: Contains the available build image options grouped by build type, including Yocto images, Ubuntu images, and static image collections (all-yocto-images, all-ubuntu-images, all-supported-images).
        - git_patch.json: Contains json keys and repository configuration such as: url, branch, tag, commit, repo type and patch paths to apply.
    - ubuntu/: Main folder for Ubuntu-based image generation for RZ boards .
        - config/: The folder that holds configuration files for different Ubuntu variants.
        - docs/: Contains documentation detailing supported features and usage instructions for each Ubuntu image variant.
        - Include/: Contains scripts related to Ubuntu (e.g., for creating WIC files, packaging the root filesystem, preparing the environment, etc.)
        - script/: The folder that contains all scripts related to Ubuntu image creation.
        - config.ini: Configuration file that defines key parameters for the Ubuntu image build process, such as the Ubuntu variant, base image, output filenames, and system settings.
        - setup_ubuntu_environment.sh: Main entry-point script (acts like a dispatcher/header). It sources and sequences logic from the modular scripts under script/. It does not build anything by itself.
- **tools/:** Provides utilities to assist with tasks such as bootloader flashing, uLoad bootloader flashing, and SD card image creation. All tools are designed to support both Linux and Windows environments.
  Key files:
    - bin/: This directory contains precompiled tools required for the firmware build and packaging process. They are organized by operating system to support cross-platform usage.
    - firmware_compile/: Contains a helper script for building Renesas RZ family firmware artifacts, including BL2, Boot Parameter (BP) files, U-Boot binaries, and FIP packages.
    - bootloader-flasher/: Contains script and README.md for flashing the main bootloader. Works on both Linux and Windows.
    - sd-creator/: Provides scripts to create SD card images from .wic files. Compatible with both Linux and Windows.
    - uload-bootloader: Includes automated host-side scripts to flash the qspi boot firmware (IPL) using the images from the SD card.
    - universal_flash.py: A master Python script that unifies all flashing workflows (bootloader, uLoad, SD card) under a single interface. Cross-platform and recommended for most use cases.
    - flash_images.json: Contains predefined image mappings for supported devices.
      User configuration should be defined in this file. All image files referenced for flashing must be located in: </path/to/your/yocto/package>/target/images>
    - config: Contains scripts uses a boards_flash_config.toml file to define flash layout and bootloader addresses for supported boards. The configuration is structured per board and boot type (e.g., qspi, emmc), and other information.

license/: Contains all the supporting legal documents and licensing agreements related to the release package.

target/: This directory includes all the files needed for deploying the system on target hardware.

- **env/:** Contains environment configuration files that are used during boot-up on the target device.
  Key file:

  - uEnv.txt: A file that holds boot configuration parameters.
- **images/:** Holds the final system images and associated files required for the target device. *Key files:*
  - atf: The RZ Common Arm Trusted-firmware (TF-A) directory contains BL2, BL31 binaries and TF-A configuration DTBs (FDTS).
  - u-boot/: The RZ Common U-Boot directory contains the U-Boot binaries and device tree blobs (DTBs) used across all supported boards.
  - linux/: The directory contains linux kernel and device trees for the target images.
  - System images: Files with the '.wic' extension corresponding to different build variants (BSP, minimal, Weston, Renesas images).
  - rootfs folder: Compressed archives (For example, core-image-bsp.tar.bz2) contain the root file system for each image.
- README.md (root level): This is the comprehensive guide that provides an overview of the entire release package, including instructions on how to use, build, and deploy the system.

2.2 Source Repositories

The system release package is maintained in public repositories that hold all the latest work that has been released. The table below describes the public repositories that are the basis for the system release package.

Table 5. Public repositories for the system release package

Name	Type	URL	Description
rz-build-scripts	Yocto build automation	Renesas-SST/rz-build-scripts: Build scripts for rz projects	Custom Yocto build script that downloads the base Yocto package and other downloaded zip files, arranges the layers, applies relevant meta layers, sets up the environment, and initiates a build.
meta-renesas	Yocto meta layer	Renesas-SST/meta-renesas: Yocto meta layer for Renesas System Solutions	Yocto meta layer supports RZ-based platforms.
linux-rz	Linux kernel	Renesas-SST/linux-rz: Linux kernel for System and Solutions Products	This repo contains the kernel fork with RZ-based patches.
u-boot	Boot loader	Renesas-SST/u-boot: A u-boot suporting System & Solutions Products	This repository contains the U-Boot bootloader source code customized for Renesas RZ devices. It includes board-specific configurations and patches required to boot Linux on supported RZ-based platforms.
rz-atf	Arm Trusted Firmware-A	Renesas-SST/rz-atf: Arm Trusted Firmware implementation for System & Solutions products	Arm trusted the firmware repo with RZ-based patches.
flash-writer	Firmware flashing tool	Renesas-SST/flash-writer: Serial flashing utility to load into blank boards supporting System & Solutions Products	This repository contains the code for an essential tool that is used as the base for flashing blank boards in the factory for the first time or recovering bricked boards.

rz-utils	Flashing tools	Renesas-SST/rz-utils: Collection of utilities for various workflows related Renesas RZ based devices	Collection of utilities for various workflows related to Renesas RZ-based devices

While the public repositories are mostly open source, the RZ MPUs contain IPs that are licensed differently, and those functionalities require specific packages to be downloaded from the Renesas website through login. These are mostly free-to-download packages and contain their licenses, which are non-standard. The build scripts can identify and point to the missing packages and their download URLs. You can download these scripts manually and copy them to the workspace.

3. Required Resources

3.1 Development Tools and Software

The following tools are used for development:

- SEGGER JLink software (SEGGER - The Embedded Experts - Downloads - J-Link / J-Trace).
- Tera Term (TeraTerm Project) on Windows PC for accessing UART.
- Minicom on the Ubuntu host machine for accessing the UART on Linux.
- Balena Etcher

3.2 Hardware

The hardware requirements vary slightly depending on the Renesas RZ board in use. Refer to the corresponding subsection below for board-specific details.

3.2.1 Common Hardware Requirements

These items are required for all supported Renesas RZ boards:

- Windows PC with Tera Term software and admin privileges.
- Ubuntu 24.04 host environment (native install, VM, or Docker) for working with Yocto distros.
- Ethernet cables for networking.

3.2.2 Board-Specific Hardware

3.2.2.1 RZ/G2L-SBC

The following hardware would be needed to work with the RZ/G2L-SBC:

- RZ/G2L-SBC - RZ/G2L Single Board Computer
- UART TTL cables (Raspberry Pi compatible) featuring FTDI chipset. We do not recommend PL2302-based UART TTL cables, as they have demonstrated issues with Windows drivers.
- Jumper wires/plugs.
- Mini-HDMI to HDMI display interface cable.
- Power supply that can provide 5V at 3 A with USB-C pins. (not included in the package).
- Waveshare 5” DSI display module with a capacitive touch interface (optional: not included in the hardware package).
- OV5640 camera module (optional: not included in the hardware package).

3.2.2.2 RZ/G2L-EVK- & RZ/V2L-EVK

The following hardware is required for working with the RZ/G2L-EVK or RZ/V2L-EVK. Most items are identical for both boards:

- RZ/G2L SMARC module board or RZ/V2L SMARC module board and common carrier board.
- USB Type-A to Micro USB Type-B cable
- Micro-HDMI to HDMI display interface cable.
- Power supply that can provide 5V at 3 A with USB-C pins. (not included in the package).
- OV5645 camera module (optional: not included in the hardware package).
- 3.5mm Audio Stereo Y Splitter extension cable (optional: not included in the hardware package).

3.2.2.3 RZ/V2H-EVK

The following hardware is needed to work with the RZ/V2H-EVK:

- RZ/V2H Quad-core Vision AI MPU Evaluation Kit

- USB Type-A to Micro USB Type-B cable.
- HDMI to HDMI display interface cable.
- Power supply that can provide up to 100W via USB-C PD (not included in the package).
- OV5645 camera module (optional: not included in the hardware package).
- Camera conversion 22-pin to 25-pin FPC adapter (optional: not included in the hardware package).
- 3.5mm Audio Stereo Y Splitter extension cable (optional: not included in the hardware package).

4. Quick Start

This section describes how to quickly get set up and start running the supported platforms with this release. The following are the essential steps for an SD-MMC card-based boot:

1. Select an image from the list of available images in Section 1.1. Supported Distributions
2. Prepare an SD MMC card that has the image programmed onto it.
3. Prepare the hardware with power and debug UART interface. Displaying the connection to one of the HDMI interfaces is highly recommended, but not essential.
4. Program the firmware using the appropriate scripts and process in the 'host/tools' directory of the package.
5. Boot normally with the SD MMC card.

4.1 SD-MMC Card Flashing

The Linux bootable SD card creation is a very simple process. The idea is to use any filesystem imaging tool (etcher) to burn the required image's '.wic' file (core-image-weston.wic for example) located in the 'target/images' directory of the release to the sd-mmc card. We recommend installing Balena etcher, which is available for Linux, MacOS, and Windows.

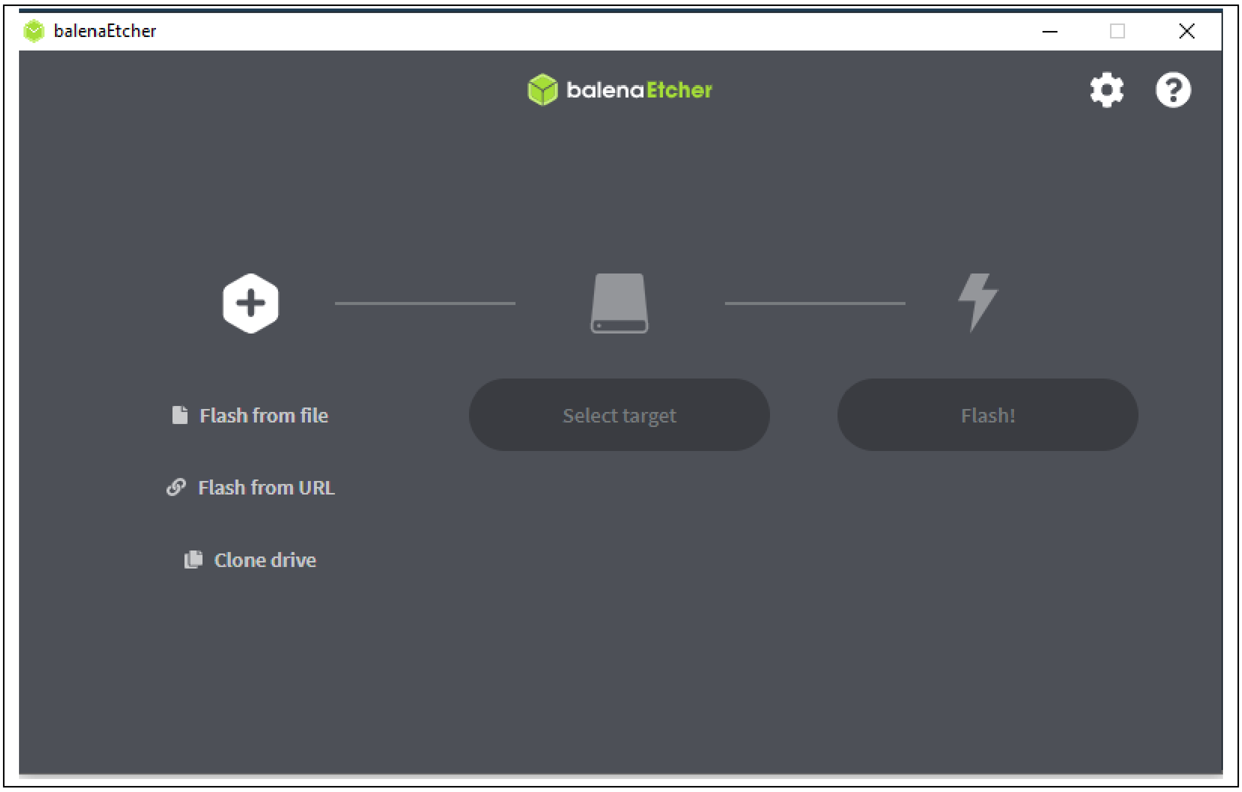


Figure 2. Balena etcher UI

Steps:

1. Select "Flash from File".
2. In the popup window, navigate to your release and select one of the chosen image files (core-image-weston.wic).
3. Then click on 'Select target,' and it will list all available devices.
4. Select your SD MMC card.
   Be mindful not to select your primary laptop/desktop hard drive.
5. Select 'Flash'.
6. When flashing is completed, it will automatically dismount the SD MMC card device.

7. Insert the SD card into the SD-MMC card slot on the RZ board.

4.2 RZ/G2L-SBC

This section describes the hardware-specific processes for the RZ/G2L-SBC (Single-board Computer).

Note:

- The release consists of images that have desktop and display support.
- At least one basic display, like a 1080p HDMI monitor, must be available for those images.
- You can also use the DSI touch panel described in the MIPI DSI Display Touch Panel.
- It is recommended to use an FTDI cable for the UART and not any other converter chip.

4.2.1 Hardware Requirements

The basic hardware setup consists of the following:

1. RZ/G2L-SBC
2. FTDI RS232 UART cable
3. USB-C 5V 3A+ power supply
4. SD-MMC card (minimum 8 GB)
5. 1080p HDMI display/Waveshare 5” MIPI DSI display touch panel
6. Ethernet cables.
7. OV5640 MIPI CSI camera
8. USB keyboard and mouse
9. 3.5mm Headphone with microphone

4.2.2 Essential Hardware Setup

Figure 3. Essential minimum interfaces show the basic essential hardware setup. We expect a UART cable and an HDMI display to be available.

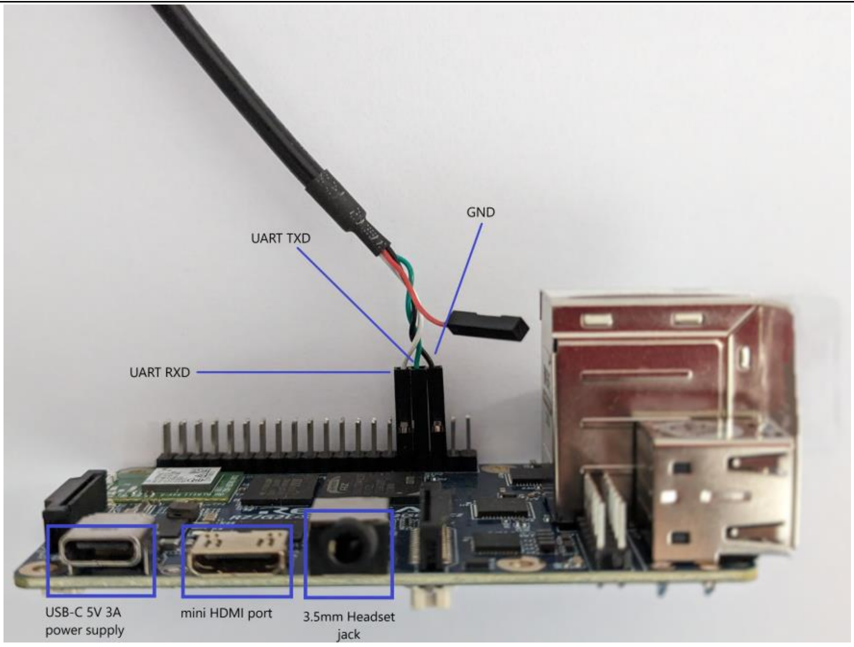


Figure 3. Essential minimum interfaces for RZ/G2L-SBC

4.2.3 Complete Hardware Setup

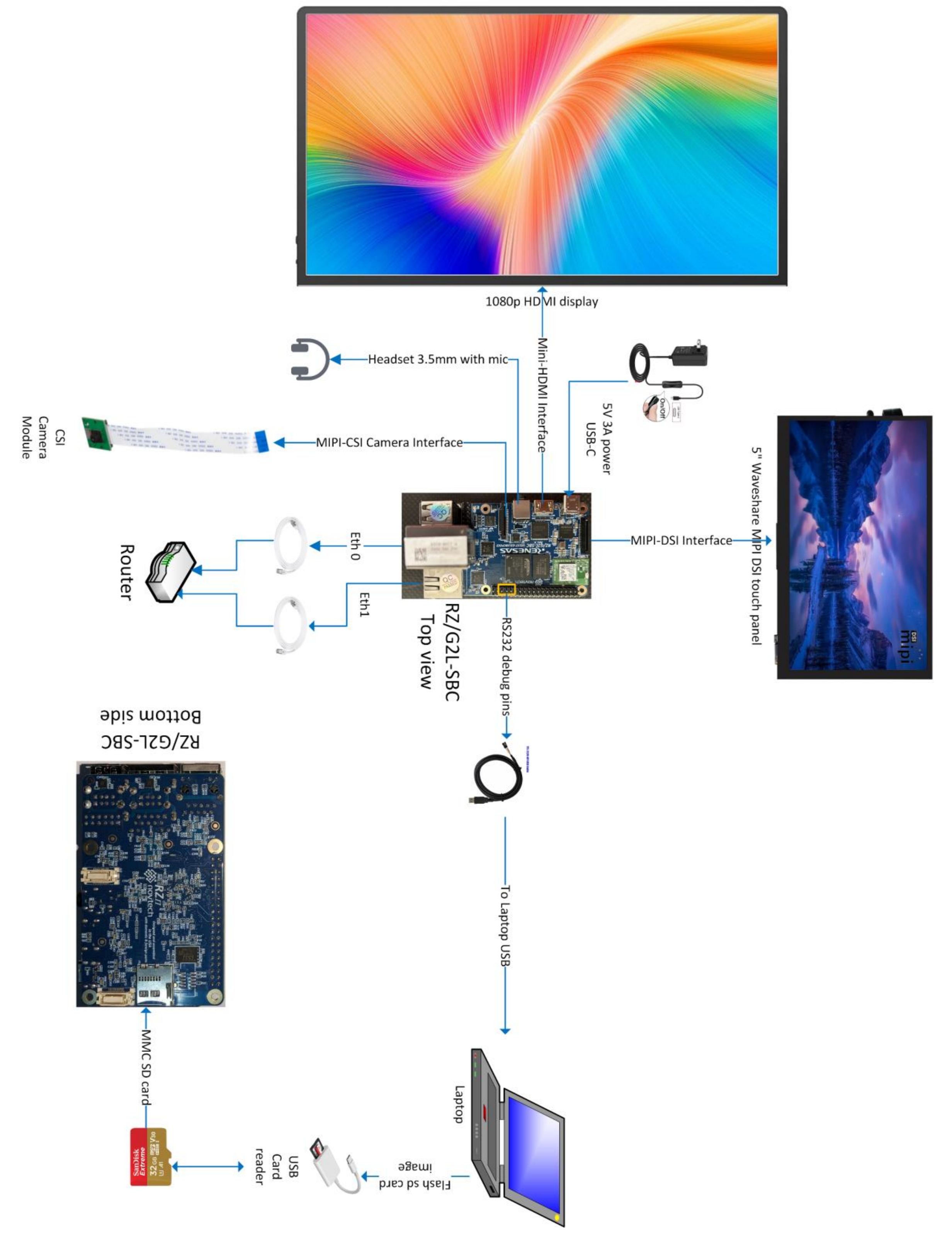


Figure 4. Complete setup for RZ/G2L-SBC board

4.2.4 Booting

The booting is straightforward.

1. Insert the MMC card into the MMC port on the bottom side of the RZ/G2L-SBC.
2. Connect the keyboard, mouse, and HDMI display; then insert the USB-C power supply and turn the power on.
3. You should see the boot log on the UART console and the Weston desktop on the HDMI screen.
4. Click on any of the applications and interact with them.

The image is fully featured and has powerful desktop-grade features. Read further to learn more about the features packed into the Linux image.

> Note: The default firmware shipped on the board may not recognize new images. If the board fails to boot, update the firmware using the Serial Download Mode (SCIF) procedure described in Section 16.1.1 RZ/G2L-SBC

4.2.5 Known Hardware and Functional Limitations on RZ/G2L-SBC

4.2.5.1 Linux (CA55) Side Known Issues

1. HDMI audio

- Status: Unverified
- Description: The functionality of the HDMI audio output has not been tested yet, and its behavior remains uncertain. Additional development and testing are required to assess its reliability and performance on the RZ/G2L-SBC.

2. Audio Sampling Rate Limitation

- Status: Currently limited to 48 kHz (validated)
- Description: The board's clock design deviates from the standard RZ reference, which prevents the existing driver frameworks from generating the proper clocks when the SoC operates as I²S/TDM master. As a result, only 48 kHz sampling has been successfully validated so far. This is not a fundamental hardware restriction — codec-master mode has not yet been evaluated, and wider sampling-rate support may be possible but remains unsupported.

3. Onboard Bluetooth (BT) Functionality

- Status: Non-functional (onboard BT only)
- Description: The onboard Bluetooth functionality is currently non-operational due to a schematic symbol error in the Laird Wi-Fi/BT module. The Bluetooth interface is missing from the module's schematic design, preventing Bluetooth connectivity. However, USB Bluetooth functionality remains operational. This issue requires a hardware revision to enable full Bluetooth functionality on the onboard module.

4.2.5.2 FreeRTOS/FSP (CM33) Side Known Issues

1. MIPI-CSI2 Camera and Peripherals Accessing Shared I2C1 Bus

In the RZ/G2L-SBC, the MIPI CSI Camera interface, HDMI Bridge, and MIPI DSI all share the same I2C1 channel. Due to this hardware constraint, controlling one of these devices may impact the functionality of the others.

Limitations:

- I2C1 can only be accessed by one core at a time, which can prevent both the camera and display from functioning simultaneously.
- Any device using I2C1 must be managed carefully to avoid conflicts with other peripherals.
- This limitation should be considered when designing the system to ensure both peripherals can operate as required.

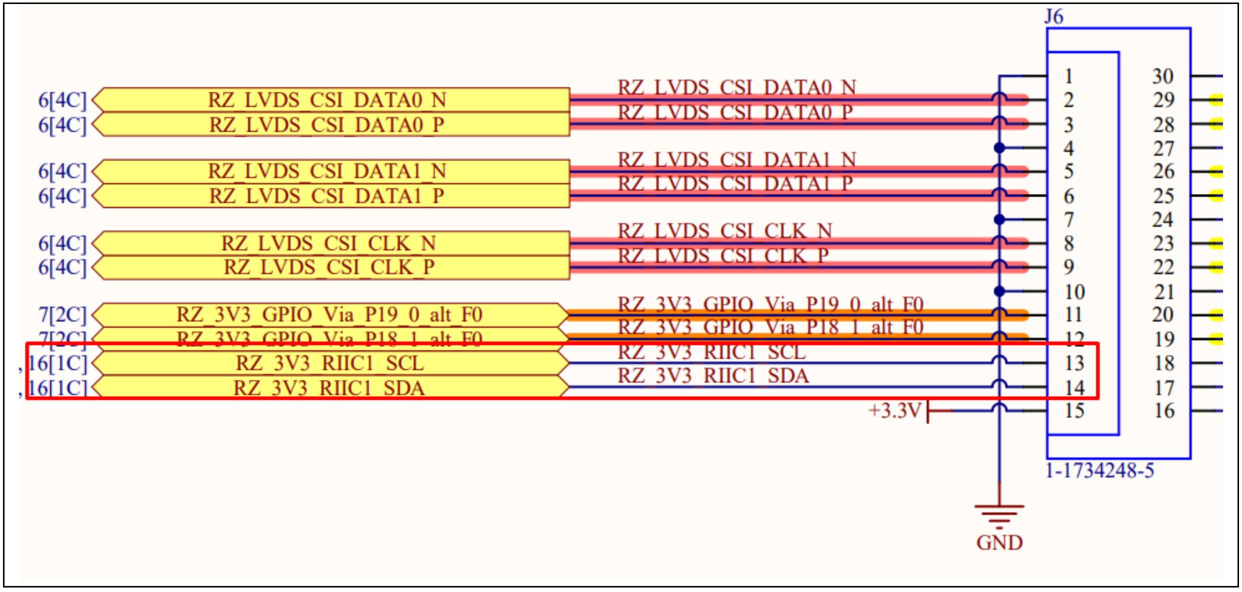


Figure 5. CSI using shared I2C1 bus

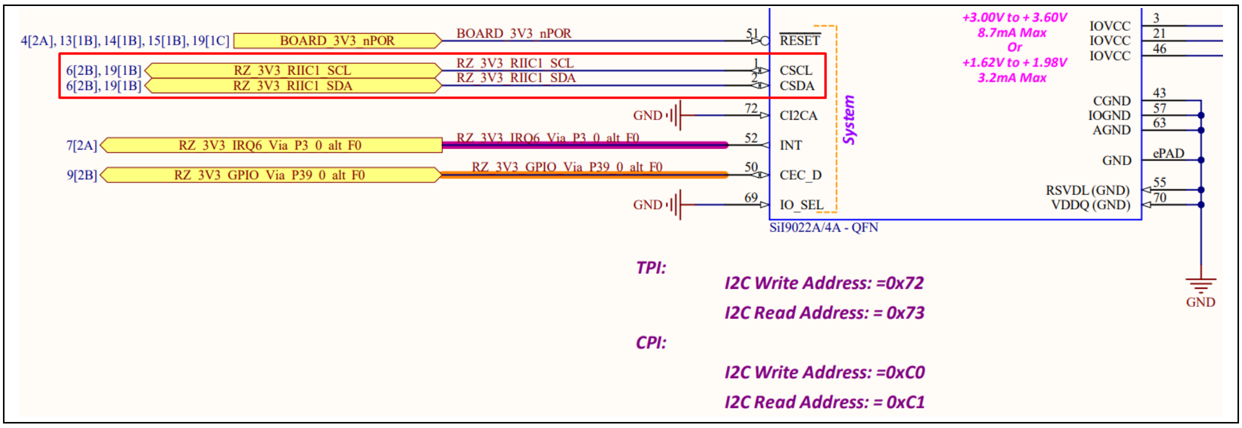


Figure 6. HDMI using shared I2C1 bus

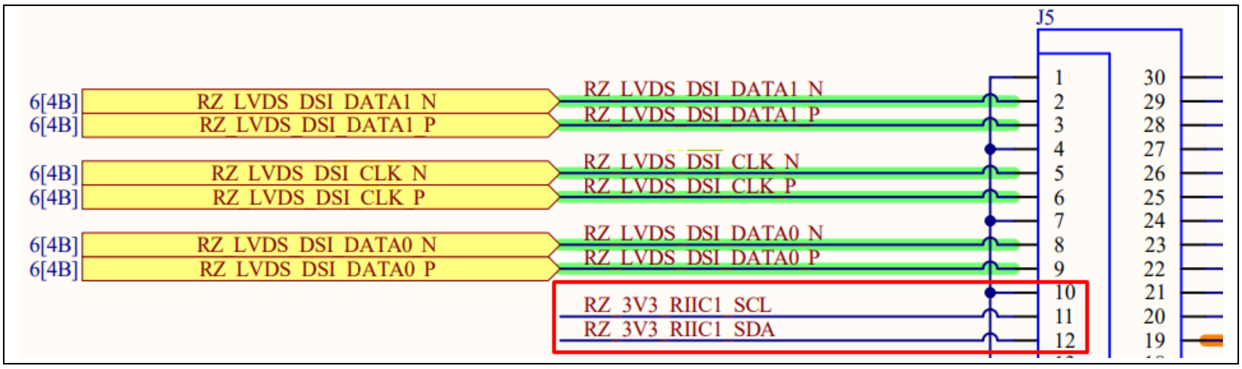


Figure 7. DSI using shared I2C1 bus

As shown in the three figures above, the shared I2C1 bus is used by multiple peripherals, which may lead to conflicts if both cores are used simultaneously. To avoid issues, users should ensure that only one core accesses I2C1 at a time or consider alternative methods for managing communication between peripherals.

2. Limited SCIF Availability for Multi-Core Development

However, a limitation exists in the number of available SCIF (Serial Communication Interface with FIFO) channels, which impacts debugging and logging functionality for multi-core development.

Limitations:

- Single SCIF Channel: Only SCIF0 is available for serial communication, and it is exclusively allocated to the CA55 core.
- Restricted logging for CM33: Since SCIF0 is dedicated to CA55, the CM33 core lacks direct access to an SCIF channel, making it challenging to perform independent serial logging or debugging.

This limitation should be considered when designing multi-core applications, especially those requiring real-time logging, debugging, or inter-core communications.

4.3 RZ/G2L-EVK and RZ/V2L-EVK

Both RZ/G2L-EVK and RZ/V2L-EVK platforms feature robust Linux images with desktop and display support. Their hardware setup and booting procedures are nearly identical, allowing a common approach for setup and evaluation.

Note:

- The release consists of images that have desktop and display support.
- At least one basic display, like a 1080p HDMI monitor, must be available for those images.

4.3.1 Hardware Requirements

The basic hardware setup consists of the following:

1. RZ/G2L-EVK or RZ/V2L-EVK
2. USB Type-A to Micro USB Type-B cable
3. USB-C 5V 3A+ power supply
4. SD-MMC card (minimum 8 GB)
5. HDMI display
6. Ethernet cables
7. OV5645 camera module (optional: not included in the hardware package)
8. USB keyboard and mouse
9. 3.5mm Audio Stereo Y Splitter extension cable (optional: not included in the hardware package)
10. 3.5mm Headphone with microphone

4.3.2 Essential Hardware Setup

The figure below shows the basic essential hardware setup for RZ/G2L-EVK & RZ/V2L-EVK. Follow these steps to prepare and power on the board:

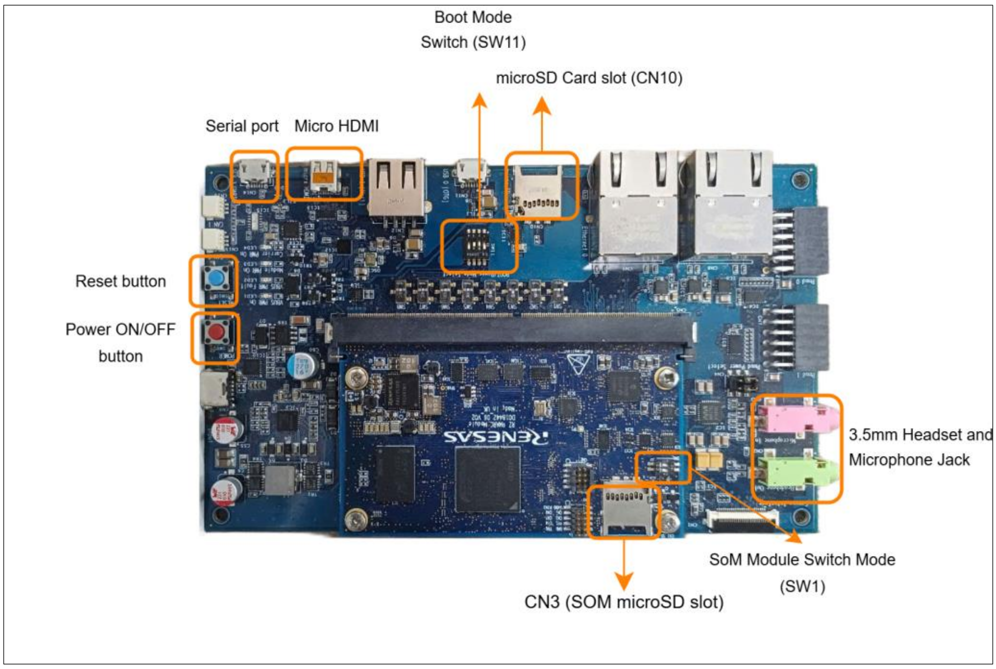


Figure 8. Essential minimum interfaces for RZ/G2L-EVK & RZ/V2L-EVK

Note:

- Boot source selection
  - In this release, all boot flows use QSPI or eMMC as selected by SW11.
  - eSD boot from CN3 is not supported.
- CN10 (Carrier microSD slot)
  - CN10 is not a boot device.
  - CN10 can be used only for rootfs or data storage after the system has booted.
  - No SW1 configuration is required for CN10.
- CN3 (SoM microSD slot)
  - BL2/FIP cannot be loaded from CN3 (eSD boot not supported).
  - When the board boots from QSPI or eMMC, CN3 is accessible in U-Boot/Linux (commonly as mmc device 0).
  - CN3 may be used as a rootfs or additional storage device if enabled via SW1-2.

1. **Prepare the microSD card**

   Flash the provided image to the microSD card using the procedure in Section 7.3.5. Flashing the SD Card Image or using a user-friendly tool such as Balena Etcher as describe in Section 4.1. SD-MMC Card Flashing for a simplified flashing experience.

2. **Insert the microSD card**

   Insert the prepared SD card into CN10. This slot is for rootfs/data only; it cannot be used to load BL2/FIP.

3. **Configure the boot mode**

   In this section, we focus on QSPI boot as the default boot method. The switch configuration below loads BL2 and FIP from QSPI flash, with the root filesystem provided from eMMC or the SD card in slot CN10/CN3.

   Default setup (QSPI boot):

- SW11 (on the carrier board): Set to QSPI boot.
- SW1-2 (on the SoM): Controls access to storage devices on the SOM (Optional)
  - OFF: Enables access to the onboard eMMC
  - ON: Enables access to the microSD slot CN3
- Root filesystem: Use the SD card in slot CN10 (carrier board). CN10 remains accessible regardless of SW1-2.

Alternative setup (eMMC boot):

- SW11 (on the carrier board): Set to eMMC boot
- SW1-2 (on the SoM): Same as above for QSPI boot.

Refer to Section 16.2 – Boot Mode Reference (RZ/G2L & RZ/V2L) for the correct switch positions. In these modes, the board fetches BL2 and FIP from the chosen device.

4. **Connect peripherals**

- Attach an HDMI display

5. **Connect UART**

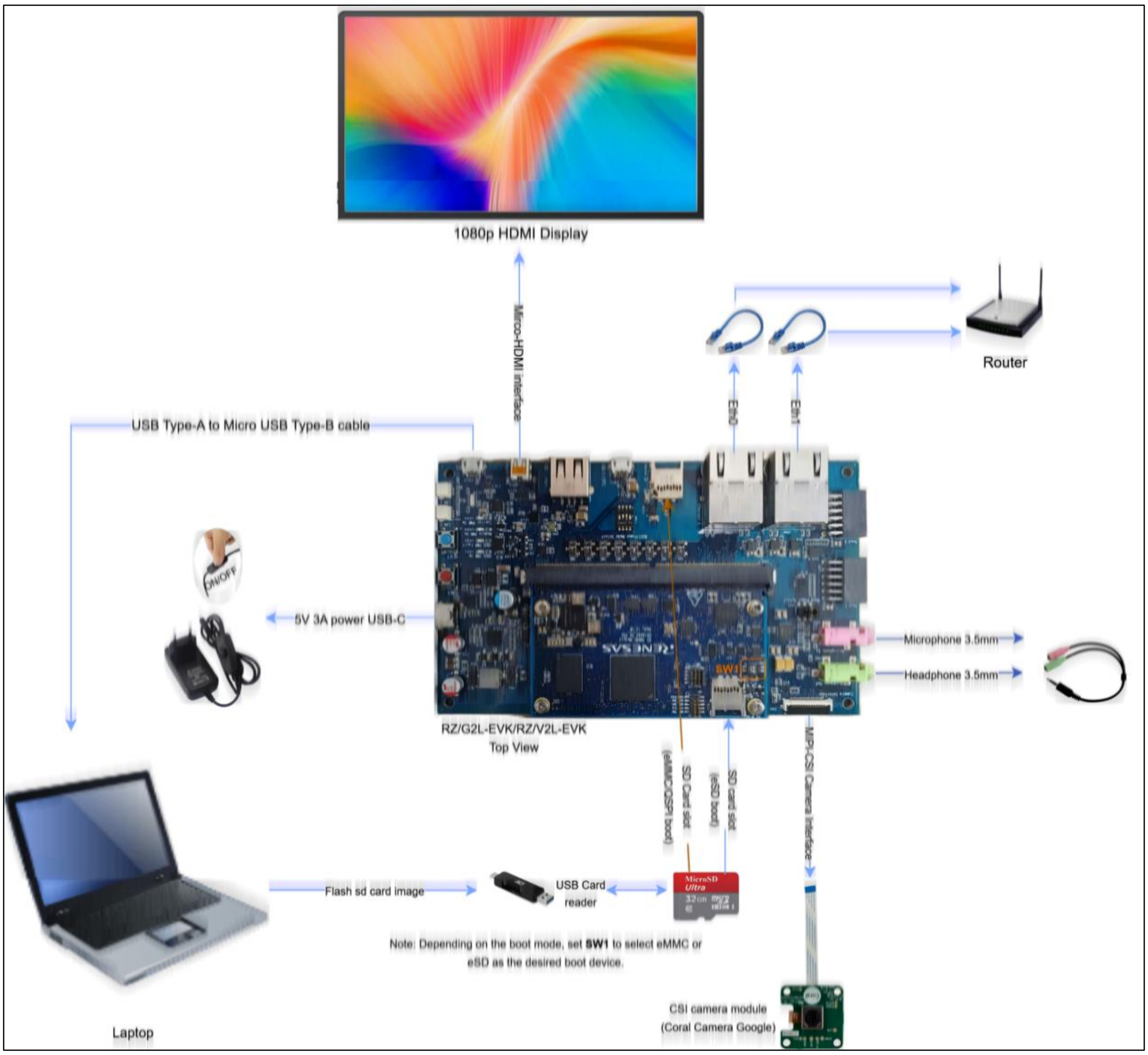
- Use a USB Type-A to Micro-USB Type-B cable to connect the board's UART console port to the host PC.
- Open a terminal program on the host PC to monitor the boot log.

6. **Power on the board**

- Connect the USB-C power supply (5 V, 3 A).
- Press the red power button to turn on the board.

Once powered on, the boot log should appear on the UART console, and the Weston desktop will display on the HDMI screen.

4.3.3 Complete Hardware Setup



Note: Depending on the boot mode, set **SW1** to select eMMC or eSD as the desired boot device.

Figure 9. Complete setup for RZ/G2L-EVK & RZ/V2L-EVK

4.3.4 Booting

The boot process is straightforward:

1. Insert the prepared SD card into CN10.
2. Connect a keyboard, mouse, HDMI display, and other devices. Then connect the USB-C power supply and press the red power button to turn the board on.
3. The boot log should appear on the UART console, and the Weston desktop should display on the HDMI screen.
4. Open any of the available applications to interact with the system.

The image provided is feature-complete, offering a desktop-grade experience.

If a different boot device is required such as QSPI, eMMC, or SD card — adjust the DIP switch settings as described in Section 16.2. Boot Mode Reference (RZ/G2L-EVK & RZ/V2L-EVK) before powering on. The selected boot device determines where on-chip ROM code loads the Boot Loader stage 2 (BL2) and Firmware Image Package (FIP):

- QSPI boot loads BL2 and FIP from the onboard QSPI flash.
- eMMC boot loads BL2 and FIP from the onboard eMMC device.
- eSD boot is not supported in this release.

The chosen boot device can be written or updated using the Universal Flashing Script; see Section 7.3. Universal Flashing Script for detailed instructions.

> Note: The default firmware shipped on the board may not recognize new images. If the board fails to boot, update the firmware using the Serial Download Mode (SCIF) procedure described in Section 16.1.2 RZ/G2L-EVK and RZ/V2L-EVK

4.3.4.1 Default Boot Behaviour

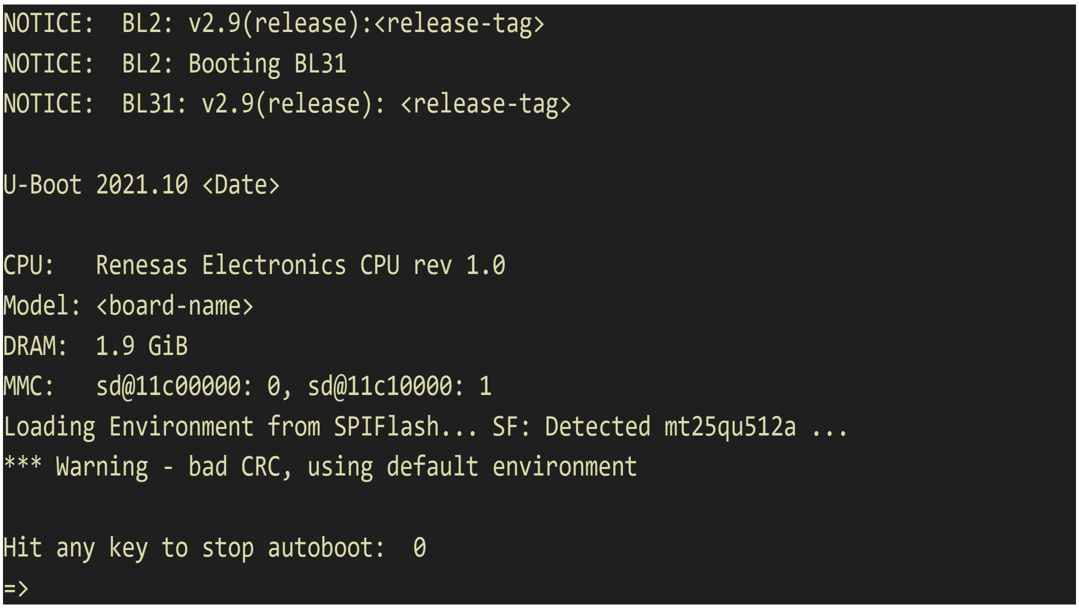
When booting from QSPI or eMMC, the U-Boot environment defaults to using the SD card (CN10) as the root filesystem.

- The SD card on CN10 is always available as mmc1.
- The onboard eMMC is always available as mmc0.
- Unless reconfigured, Linux will mount the rootfs from CN10 (e.g., /dev/mmcblk1p1).

4.3.4.2 Using an eMMC Root Filesystem

To use the onboard eMMC as the root filesystem instead of CN10:

1. Enable eMMC on the SoM: Set SW1-2 = OFF (see Section 16.1 Boot Mode Reference (Non-SCIF) RZ/G2L-EVK & RZ/V2L-EVK for SW1 description).
2. Prepare the eMMC rootfs. See Section 16.3 Prepare the eMMC root filesystem for detailed information
3. Reboot the board. During startup, output similar to the following will appear:



```
NOTICE:  BL2: v2.9(release):<release-tag>
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.9(release): <release-tag>

U-Boot 2021.10 <Date>

CPU:   Renesas Electronics CPU rev 1.0
Model: <board-name>
DRAM:  1.9 GiB
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from SPIFlash... SF: Detected mt25qu512a ...
*** Warning - bad CRC, using default environment

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

   - When the message “Hit any key to stop autoboot” appears, press Enter (or any key) to interrupt the countdown.
   - The => prompt indicates that are in the U-Boot console.
4. Now update the environment by changing the default device from 1 (SD – CN10) to 0 (eMMC) and then boot.

```
=> setenv mmcdev 0
=> saveenv
=> boot
```

4.3.4.3 Using CN3 (eSD slot) as Root Filesystem

eSD boot from CN3 is not supported in this release, meaning BL2/FIP cannot be loaded directly from this slot. However, once the board boots from QSPI, CN3 remains accessible in U-Boot and Linux as a standard MMC device. This allows CN3 to be used for the root filesystem or as additional data storage.

Steps to use CN3 as the root filesystem:

1. Insert the SD card into the CN3 slot
2. Enable eSD on the SoM: Set SW1-2 = ON (see Section 16.1 Boot Mode Reference (Non-SCIF) RZ/G2L-EVK & RZ/V2L-EVK for SW1 description).
3. Reboot the board. During startup, output similar to the following will appear:

```
NOTICE:  BL2: v2.9(release):<release-tag>
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.9(release): <release-tag>

U-Boot 2021.10 <Date>

CPU:   Renesas Electronics CPU rev 1.0
Model: <board-name>
DRAM:  1.9 GiB
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from SPIFlash... SF: Detected mt25qu512a ...
*** Warning - bad CRC, using default environment

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

   - When the message “Hit any key to stop autoboot” appears, press Enter (or any key) to interrupt the countdown.
   - The => prompt indicates that the system is now in the U-Boot console.

4. Update the U-Boot environment by changing the default device from 1 (CN10) to 0 (CN3):

```
=> setenv mmcdev 0
=> saveenv
=> boot
```

4.3.5 Known Hardware and Functional Limitation

1. Unstable UDP flashing

   UDP-based flashing on RZG2L-EVK and RZV2L-EVK is not stable, likely due to limitations in the on-board Ethernet interface. Use balenaEtcher or the dd command to flash root filesystems instead of relying on UDP mode.

2. No onboard Wi-Fi/Bluetooth

   These EVKs do not include an on-chip or onboard Wi-Fi/Bluetooth module. Wireless connectivity must be provided through external modules (e.g., USB or SDIO-based Wi-Fi dongles).

3. USB OTG is not supported in this release
4. eSD boot is not supported in this release.

4.4 RZ/V2H-EVK

This section describes the hardware-specific setup and booting process for the RZ/V2H-EVK

This EVK supports rich multimedia, multi-camera, and AI inference workloads. Advanced features may require DIP switch configuration. For full details, refer to the RZ/V2H Evaluation Board Kit Hardware Manual.

4.4.1 Hardware Requirements

The basic hardware setup consists of the following:

1. RZ/V2H-EVK
2. USB Type-A to Micro USB Type-B cable
3. Power supply that can provide up to 100W via USB-C PD (not included in the package).
4. SD-MMC card (minimum 8 GB)
5. 1080p HDMI display
6. Ethernet cables.
7. OV5645 camera module (optional: not included in the hardware package).
8. Camera conversion 22-pin to 25-pin FPC adapter
9. USB keyboard and mouse
10. 3.5mm Audio Stereo Y Splitter extension cable (optional: not included in the hardware package)
11. 3.5mm Headphone with microphone

4.4.2 Essential Hardware Setup and Booting

The figure below shows the basic essential hardware setup for RZ/V2H-EVK. Follow these steps to prepare and power on the board:

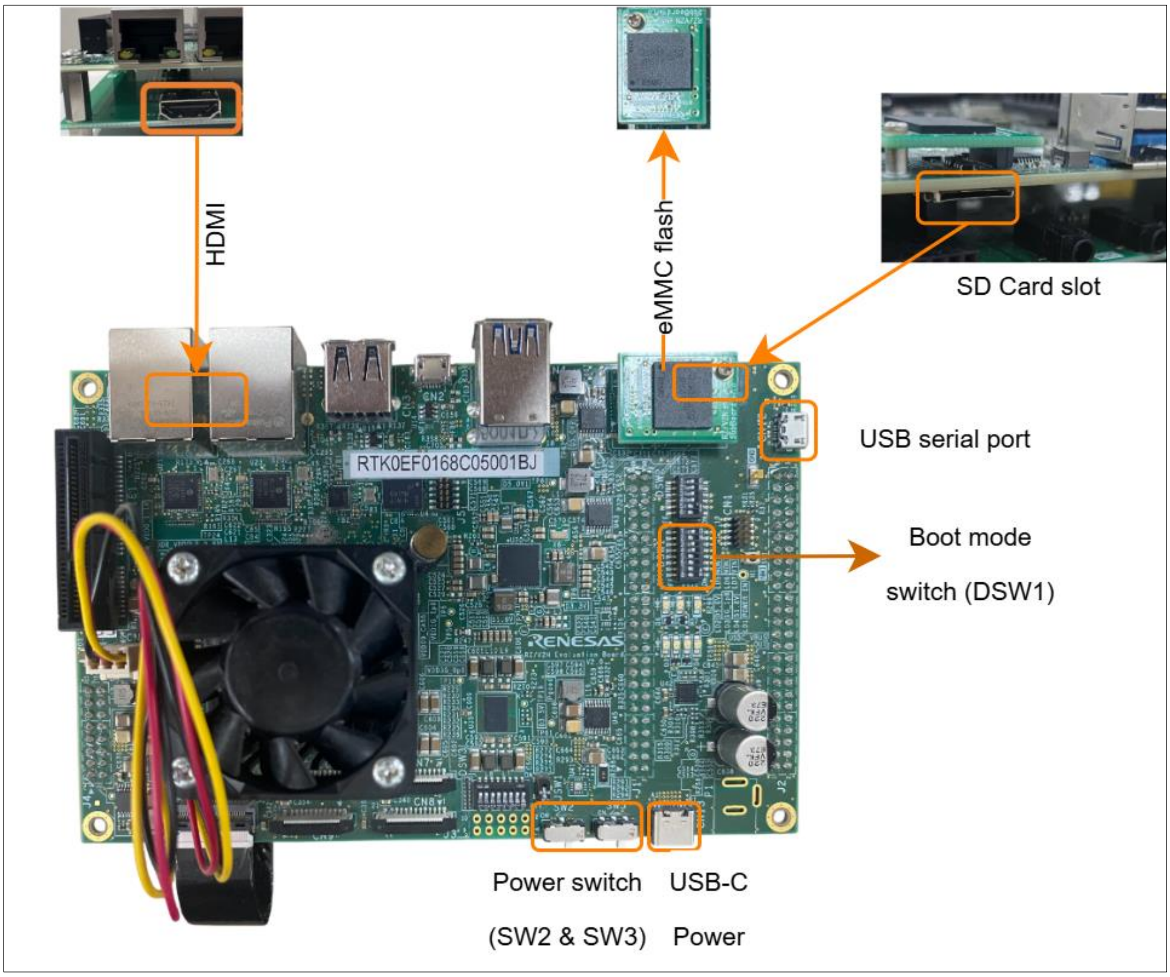


Figure 10. Essential minimum interfaces for RZ/V2H-EVK

1. **Prepare the microSD card**

   Flash the provided image to the microSD card using the procedure in Section 7.3.5. Flashing the SD Card Image or use a user-friendly tool such as Balena Etcher as described in Section 4.1. SD-MMC Card Flashing for a simplified flashing experience.

2. **Insert the microSD card**

   Insert the prepared microSD card into the SD card slot on the underside of the board.

3. **Configure the boot mode**

   Set the DIP switches (DSW1) to select the boot source. Section 16.2 – Boot Mode Reference (RZ/V2H-EVK) for the correct switch positions.

   For this activity, configure DSW1 to boot from QSPI flash:

   - BL2 and FIP are loaded from QSPI.
   - Rootfs can be located on eMMC or SD card, as defined in the U-Boot environment.

   Note: Booting from eMMC and eSD is currently not supported in this release. These options may be enabled in future updates.

4. **Connect peripherals**

   Attach an HDMI display.

5. **Connect UART**
    - Use a USB Type-A to Micro-USB Type-B cable to connect the board's UART console port to the host PC.
    - Open a terminal program on the host PC to monitor the boot log.
6. **Power on the board**
    - Connect the USB-C power supply (100W).
    - Turn on SW2 and SW3.

Once powered on, the boot log should appear on the UART console, and the Weston desktop will display on the HDMI screen.

4.4.3 Complete Hardware Setup

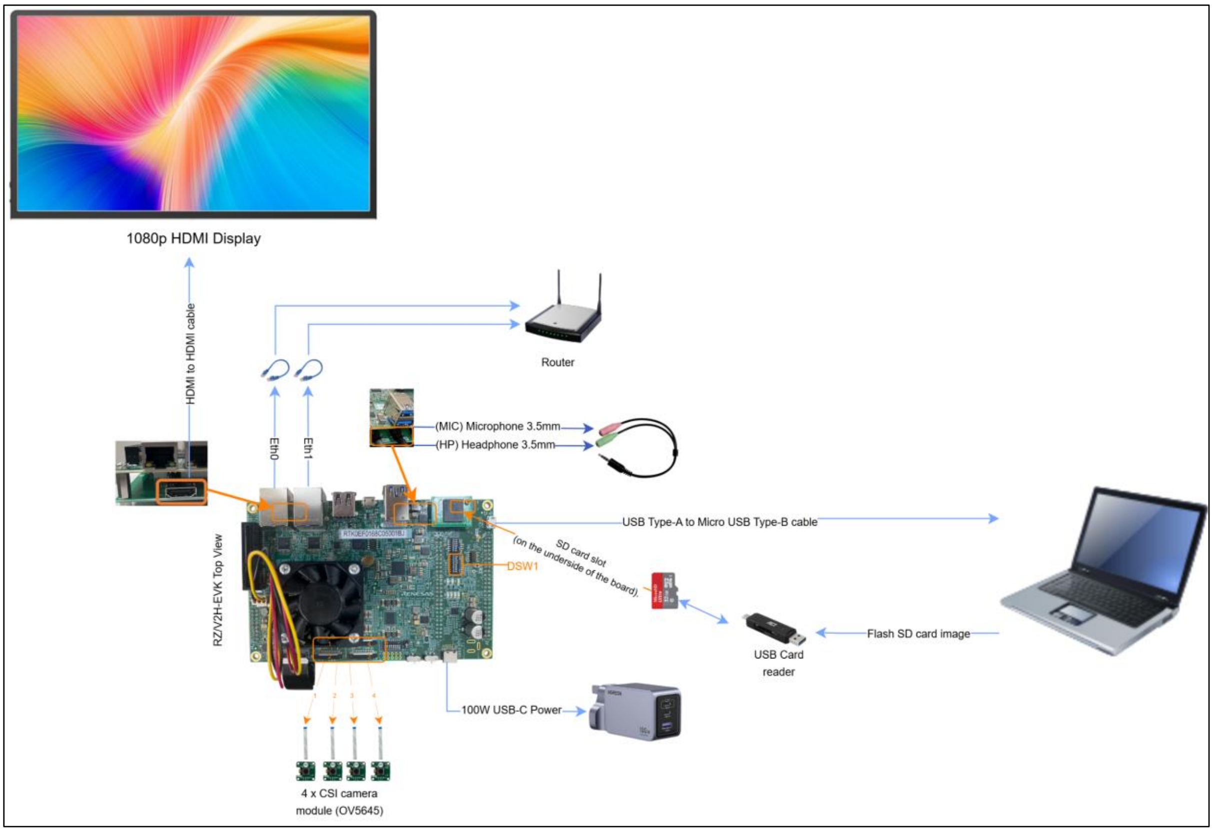


Figure 11. Complete setup for RZ/V2H-EVK board

Note: The RZ/V2H-EVK is available in two hardware versions, which differ in their storage options:

- **Version 1**: Equipped with two SD card slots and no onboard eMMC.
- **Version 2**: Equipped with one onboard eMMC (default boot device) and one SD card slot for alternate booting or data storage.

The complete setup shown above applies to Version 2. For Version 1, the setup procedure is the same as Version 2, with the only difference being the use of two SD card slots instead of one (and no onboard eMMC).

4.4.4 Booting

The boot process for RZ/V2H-EVK is very similar to RZ/G2L-EVK and RZ/V2L-EVK, as the boot mode must be selected using the DIP switches before power-on. Follow the steps below to boot the board:

1. Insert the SD card into the SD card slot (located on the underside of the carrier board; see the complete setup picture in the previous section).
2. Connect the keyboard, mouse, and HDMI display. Then connect the 100 W USB-C power supply and toggle two power switches next to the USB-C port (SW2 and SW3)
3. The boot log will appear on the UART console, and the Weston desktop will display on the HDMI screen.
4. Launch any of the available applications to interact with the system.

The provided image is feature-complete and offers desktop-grade user experience.

As with other RZ/G2L-EVK & RZ/V2L-EVK, the RZ/V2H-EVK requires selecting a boot device — such as xSPI, eMMC, or SD Card before powering on. Adjust the DIP switch settings as described in Section 16.2. Boot Mode Reference (RZ/V2H-EVK).

The chosen boot device can be written or updated using the Universal Flashing Script; see Section 7.3. Universal Flashing Script for detailed instructions.

> Note: The default firmware shipped on the board may not recognize new images. If the board fails to boot, update the firmware using the Serial Download Mode (SCIF) procedure described in Section 16.1.3 RZ/V2H-EVK

4.4.4.1 Default Boot Behaviour

When booting from QSPI or eMMC, the U-Boot environment defaults to using the SD card as the root filesystem.

- The SD card is always available as mmc0.
- The onboard eMMC is always available as mmc1.
- Unless reconfigured, Linux will mount the rootfs from SD card slot (e.g., /dev/mmcblk0p1).

4.4.4.2 Using an eMMC Root Filesystem

To use the onboard eMMC as the root filesystem instead of CN10:

1. Enable eMMC by setting DSW1 to eMMC boot mode (see Section 16.2 Boot Mode Reference (Non-SCIF) RZ/V2H-EVK for details)
2. Prepare the eMMC rootfs. Refer to Section 16.3 Prepare the eMMC root filesystem
3. Reboot the board. During startup, output similar to the following will appear:

```
NOTICE:  BL2: v2.9(release):<release-tag>
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.9(release): <release-tag>

U-Boot 2021.10 <Date>

CPU:   Renesas Electronics CPU rev 1.0
Model: <board-name>
DRAM:  1.9 GiB
MMC:   sd@11c00000: 0, sd@11c10000: 1
Loading Environment from SPIFlash... SF: Detected mt25qu512a ...
*** Warning - bad CRC, using default environment

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

- When the message “Hit any key to stop autoboot” appears, press Enter (or any key) to interrupt the countdown.
- The => prompt indicates that are in the U-Boot console.

4. Now update the environment by changing the default device from 0 (SD) to 1 (eMMC) and then boot.

```
=> setenv mmcdev 0
=> saveenv
=> boot
```

4.4.5 Known Hardware and Functional Limitations

1. No on-board Wi-Fi/Bluetooth:

   These EVKs do not include an on-chip or onboard Wi-Fi/Bluetooth module. Wireless connectivity must be provided through external modules (e.g., USB or SDIO-based Wi-Fi dongles).
2. USB OTG is not supported in this release
3. eSD boot is not supported in this release.
4. eMMC boot is not supported in this release.

   The RZ/V2H board exists in two hardware versions, one of which does not include eMMC. As a result, eMMC flash support was not provided.

5. General Operational Flow

The diagram below shows the operational flow of most of the RZ-based systems during power ON.

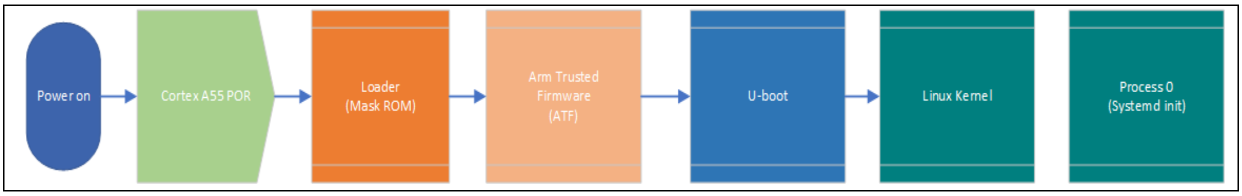


Figure 12. RZ-based systems boot operational flow

By default, the board is in the power OFF state. When the power is supplied, the PMIC immediately cycles power and puts the Cortex A55 into a POR state. This kickstarts the boot process with the Loader and ends with Linux booting into user space.

While u-boot passes full control to the Linux kernel, the ARM trust zone remains active along with op-tee within the ARM core's trust zone of operations.

The exact boot time depends on the boot environment and the number of services in the initialization process.

5.1 Arm Exception Levels

In order to explain the booting and running of software, it is necessary to understand the ARM core's exception levels. Exception levels refer to the different privilege modes of operation. These are different levels at which the CPU operates, and each level is a layer of software that remains active throughout the runtime of the SoC.

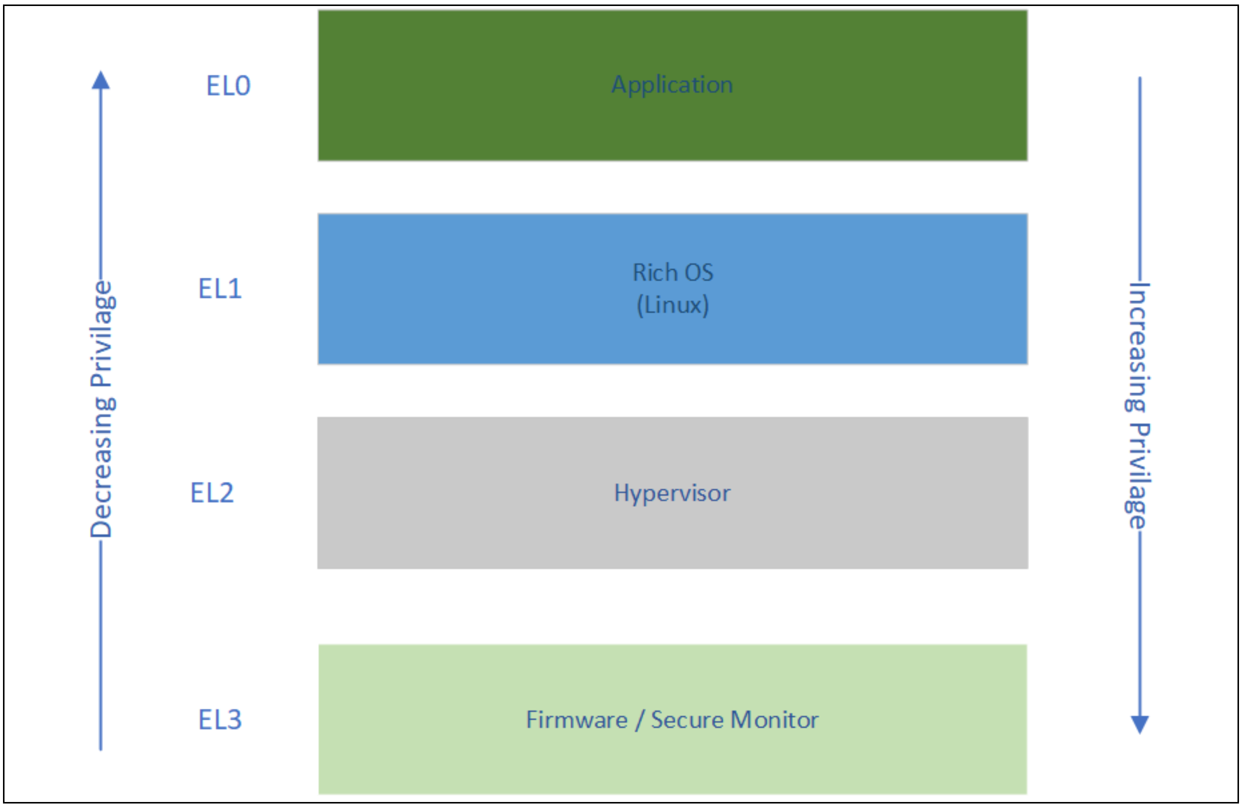


Figure 13. Arm Exception levels

The general layout of this is intuitive. However, each level has multiple implementations and layer stacks of its own. In most of our implementations, we do not deploy a Hypervisor. Hence, EL2 is bypassed. The levels we are concerned with are mostly EL3 and EL2. EL3 is complex, having its own stack of layers, which will be discussed subsequently in this section.

5.2 Secure and Non-Secure Runtime

The modern Arm64 has two execution modes, both active at the same time. These are called secure and non-secure worlds. The secure world comes with its own OS, storage management, exception handling, bootloaders, etc. The previous diagram shows the boot time model. The diagram below shows the runtime model of the system levels.

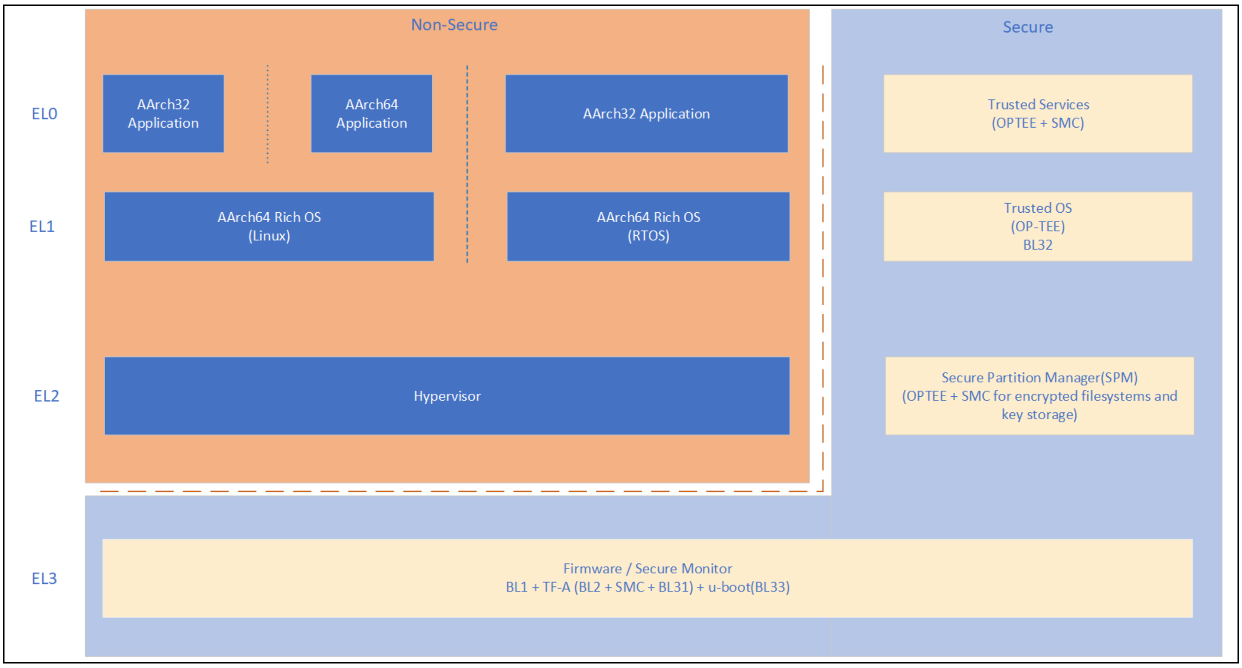

Non-Secure
Secure
EL0
EL1
EL2
EL3
AArch32 Application
AArch64 Application
AArch32 Application
Trusted Services (OPTEE + SMC)
AArch64 Rich OS (Linux)
AArch64 Rich OS (RTOS)
Trusted OS (OP-TEE) BL32
Hypervisor
Secure Partition Manager(SPM) (OPTEE + SMC for encrypted filesystems and key storage)
Firmware / Secure Monitor
BL1 + TF-A (BL2 + SMC + BL31) + u-boot(BL33)

Figure 14. Runtime Exception levels with their states

5.3 Arm Trusted Firmware-A (TF-A)

The Arm Trusted Firmware-A (TF-A project) is a reference implementation of the ARM architecture's EL3 layer. It consists of the entire stack of firmware in the EL3 level except for u-boot, which is the primary high-level boot loader.

Unlike in previous implementations of ARM, where there were just stage 1 and stage 2 bootloaders, the current generations operate with a more complex hierarchy that abstracts out hardware access and initializations into different components of the boot. Even DDR configurations, which were traditionally done by u-boot and later managed by the Linux kernel, are now done in the TF-A.

The TF-A project is also what is used by Renesas as its EL3 Firmware source. The forks of TF-A maintained by Renesas contain necessary hardware specific changes on top of the ARM's reference implementation.

The reference TF-A version we use in our current release is 2.9.

5.3.1 Components of Boot

This section provides an overview of the entire boot process used in the RZ-based system as per the TF-A paradigm. The process is divided into several distinct stages that collectively transition the hardware from a power-off state to a fully operational system running the operating system. Each stage is handled by a different component:

Table 6. RZ-specific TF-A implementation

Component	Description
BL1 (Boot Loader Stage 1 – AP Trusted ROM):	• BL1 is embedded in the Boot ROM of the SoC and is the first code executed when power is applied (POR). • It initializes the most basic hardware components (such as memory and essential peripherals) and sets the stage for a secure boot.
BL2 (Bootloader Stage 2 – Trusted boot firmware)	• BL2 is the next stage in the boot process, executed by the Trusted Boot Firmware that was loaded in BL1. • It loads two key components into DRAM: the EL3 Runtime Software (BL31) and U-Boot.
BL31 (Boot Loader stage 3-1 – EL3 Runtime Software)	• BL31 is the EL3 Runtime Software running at the highest privilege level (EL3). • It is part of the Arm Trusted Firmware-A and is responsible for configuring Arm TrustZone to create a secure execution environment.
BL32 (Boot Loader stage 3-2 – Secure-EL1 Payload)	• This is optional; it is used to load a secure payload (such as OP-TEE) that provides a Trusted Execution Environment. This is also where the encrypted filesystem is handled and contains the keys. This layer is also known as “Trust Zone” or “Trusted Execution Environment” (TEE), such as the ‘TEE’ in ‘OP-TEE’
BL33 (Non-Trusted-firmware – u-boot)	• The final boot stage where the u-boot is loaded, taking control of the system and booting the operating system into user space.

Note: TF-A is highly flexible and has plenty of optional layers. This section covers the specific implementation used in the RZ series of SoC’s reference implementation and hence does not use (bypasses) many of the optional levels such as BL3-0 SCP Firmware load. The components described here are the final implementation and supersede the reference ARM TF-A.

5.3.1.1 BL1

BL1 is the first program that runs on POR. While TF-A mentions all the layers and provides reference implementations till BL32, the BL1 is often a proprietary implementation that is strictly under NDA requirements.

The core functions of BL1 include:

- Execution of code from the POR reset vector, which contains an XIP ROM within the SoC.
- Determine between warm and cold reset boots. In the case of a warm boot, the secondary cores will go on a separate entry point while the primary boot core will continue cold boot. This is platform specific.
- Platform initialization:
  - Enable trusted watchdog.
  - Base Clock configuration and enable system timer.
  - Console initialization.
  - Enable SoC interconnect like AHB bridges.
  - Enable MMU and the memory map.
  - Enable the peripherals that are potential sources for the BL2 image.
- Check if there is a firmware update required, and if so, proceed with the firmware update process. This is usually the case with warm reset with a setting passed in.
- Read bootstrapping pins and cycle through boot sources.
- BL2 image load.
- Pass control to BL2.

Note: BL1 functions are platform dependent. In aarch64 architectures, BL1 contains a limited set of SMC call implementations, allowing some back and forth between EL1, BL2, and BL1.

Unlike traditional systems, modern systems have all layers of software remain active in their own quasi-virtual world and allow some interactions through the SMC call mechanism. These are critical for firmware updates and control passing across layers, along with initialization data like ID strings and timing. The Linux kernel has a secure component that interacts with all the EL3 components through SMC calls.

For more information, refer to TF-A's documentation for further details.

5.3.1.2 BL2

The BL2 layer is provided by Renesas TF-A implementation. The TF-A provides bl2.bin, which is the implementation used here.

Functions of BL2:

- Initialize console.
- Configure platform storage to load further images. This includes DDR, SD MMC, etc.
- MMU initialization and mapping.
- Set up security components.
- Populate shared memory to be passed to other components.
- Define memory regions and timing.
- Device tree load and address passing.
- BL2 loads multiple images to the DDR memory.
  - EL1 Runtime: This is the secure TEE OS that runs on the secure side of EL1.
  - U-boot: Primary BL33 bootloader.

5.3.1.3 BL31

BL31 is the EL3 runtime software responsible for managing secure monitor calls (SMCs) and switching between secure and non-secure execution worlds. It is provided as part of the TF-A implementation.

Functions of BL31:

- Initialize console.
- Configure the Interconnect to enable hardware coherency.
- Enable the MMU and map the memory it needs to access.

To initialize the generic interrupt controller (GIC):

- Initialize the power controller device.
- Detect the system topology.
- Manage execution handoff between secure (BL32) and non-secure (BL33) environments.
- If a Trusted OS (BL32) is present, transfer execution to BL32; otherwise, transfer execution directly to BL33.

5.3.1.4 BL33

BL33 refers to the non-secure world in ARM's TEE. It is the first non-secure code loaded by TF-A after the secure world (BL31) and optionally BL32 (Trusted OS).

In ARM-based systems, BL33 is typically the secondary stage bootloader (SSBL), responsible for further system initialization and loading the operating system.

For RZ-based systems, U-Boot is used as the default SSBL, preparing the hardware and loading the operating system into memory. The reference U-Boot version we use in our current release is 2021.10.

Functions of BL33 (U-Boot):

- Initializes key hardware components such as the CPU, memory, and storage devices.

- Configures system peripherals like UART, I2C, SPI, and others, ensuring they are ready for the OS.
- Loads essential images, including the kernel and device tree, into memory from boot sources such as SD cards or network locations using Network Boot and TFTP.
- Sets the necessary environment variables and passes boot arguments to the kernel.
- Finally, after loading the required images, BL33 hands off control to the operating system kernel to complete the boot process.

5.3.2 Trusted Boot Flow

The Trusted Boot process ensures secure initialization of the system through the following sequence:

- Power-on and BL1 execution: When power is applied, the boot ROM immediately executes BL1. It performs minimal hardware initialization and basic security checks.
- BL2 initialization: BL1 loads BL2 into memory. BL2 then verifies the system’s integrity, establishes additional secure boot mechanisms, and loads the next critical components (BL31 and U-Boot).
- Security setup via BL31: EL3 Runtime Software executes and sets up system-level security before handing off control.
- Transition to normal operation: U-Boot takes over from BL31 to finalize system initialization, prepare hardware interfaces, and manage device drivers as needed.
- Launching the operating system (BL33): Finally, U-Boot loads the Linux kernel (BL33). The kernel then starts the operating system, transitioning the system into full user-mode operation, and completes the boot process.

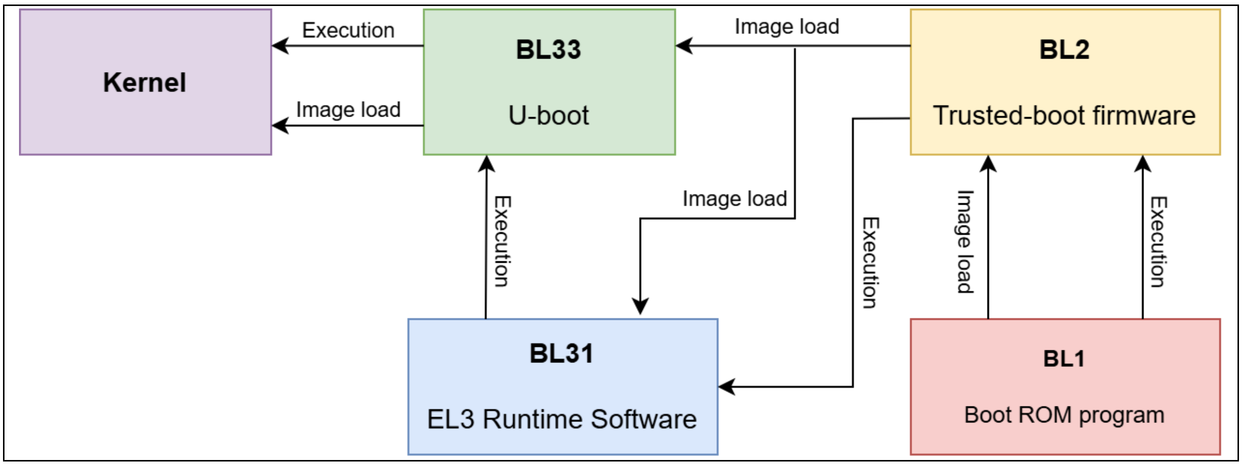


Figure 15. Trusted boot flow

6. Build System

The System release package comes with a copy of the build system that allows users to build the yocto as well as the ubuntu images on their own. This is located under the 'host/src/rz-cmn-srp/' path under the release package. The GitHub project 'rz-build-scripts' as noted in Table 5. Public repositories for the system release package is the original reference. The appropriate branch can be investigated for further reference. The branch names usually are of the format <yocto-name>/<machine-id>.

The repository hosts builder infra that allows the user to rebuild the release using one's own host machines. All projects concerned are available on GitHub and can be cloned for further development by end users restricted only by their respective end user Licenses.

One thing to be noted here is that the yocto is the basis for much of the build's internal workings. Hence the Yocto OE build must always be setup regardless of what else is being built.

6.1 Build Host Environment Setup

This section describes how to prepare a host system, install required dependencies, and perform a full Yocto build. The same environment is also used later when building Ubuntu images. The following sub-sections provide a step-by-step guide for setting up and completing a successful Yocto/Ubuntu build.

Requirements

- Ubuntu 24.04 LTS (64bit) is recommended as a build environment as we are building 'Styhead' version of yocto.
- The RZ System release package must be available on the host

Prerequisite files

The files listed in Table 7. Prerequisite files from the release package (Yocto build only) are part of the release package. These are essential files to be used for the RZ-based system Yocto build and are located under 'host/src/' in the release package.

Table 7. Prerequisite files from the release package (Yocto build only)

File	Description
rz-cmn-srp/	Main folder for Yocto/Ubuntu builds environment for RZ platforms
rz_builder.sh	Custom master build script that downloads required packages and ZIP files, configures meta layers, sets up the environment, and builds for both Yocto and Ubuntu target images. Execute it with no arguments for a helpful description.
site.conf	An override file that targets a specific build version.
patches	This is a folder that contains additional patches that are needed for Yocto eSDK build. The patches are organized as follows: • meta-summit-radio/ • 0001-rz-sbc-meta-summit-radio-Support-build-in-yocto-styh.patch • 0002-rz-sbc-summit-radio-support-eSDK-build.patch • meta-rz-features/ • 0001-support-codec-for-linux-6.10-and-yocto-styhead.patch
files_to_add	A folder containing additional files that need to be added to the meta layer. These files are specified in the add_files field of git_patch.json, which defines their source locations and target destinations. • meta-rz-features / • 0001-rzg2l-sbc-Bring-compat_alloc_user_space-back.patch • 0004-rzg2l-sbc-Get-interrupt-number.patch

config.json	Contains the config options including available build image options grouped by build type, including Yocto images, Ubuntu images, and static image collections (all-yocto-images, all-ubuntu-images, all-supported-images).
git_patch.json	A configuration file contains JSON keys and repository configuration such as: url, branch, tag, commit, repo type and patch paths to apply.
jq-linux-amd64	A lightweight and flexible shell tool that supports parsing of JSON data.
README.md	A README file describing all the necessary info about the building process.

Optional references:

- Yocto documentation (Yocto Project Documentation) – background and deeper details.
- Yocto quick build guide (Yocto Project Quick Build — The Yocto Project ®5.1.4 documentation) – alternative quick-start overview.

Install packages on Ubuntu Host.

1. Update the Ubuntu package manager.

```
$ sudo apt update
```

2. Install necessary packages and tools which are used by the Yocto build.

```
$ sudo apt install -y 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 \
rsync curl locales bash-completion zstd lz4
```

3. Configure local git account for the user.

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

4. Download the following packages provided by Renesas.

Table 8. List of packages to manually download for Yocto Build

File name	Version	Download Link	Comments
RTK0EF0045Z15001ZJ-v1.1.0_EN.zip	1.1.0	rz-mpu-video-codec-library-evaluation-version	Video codec package

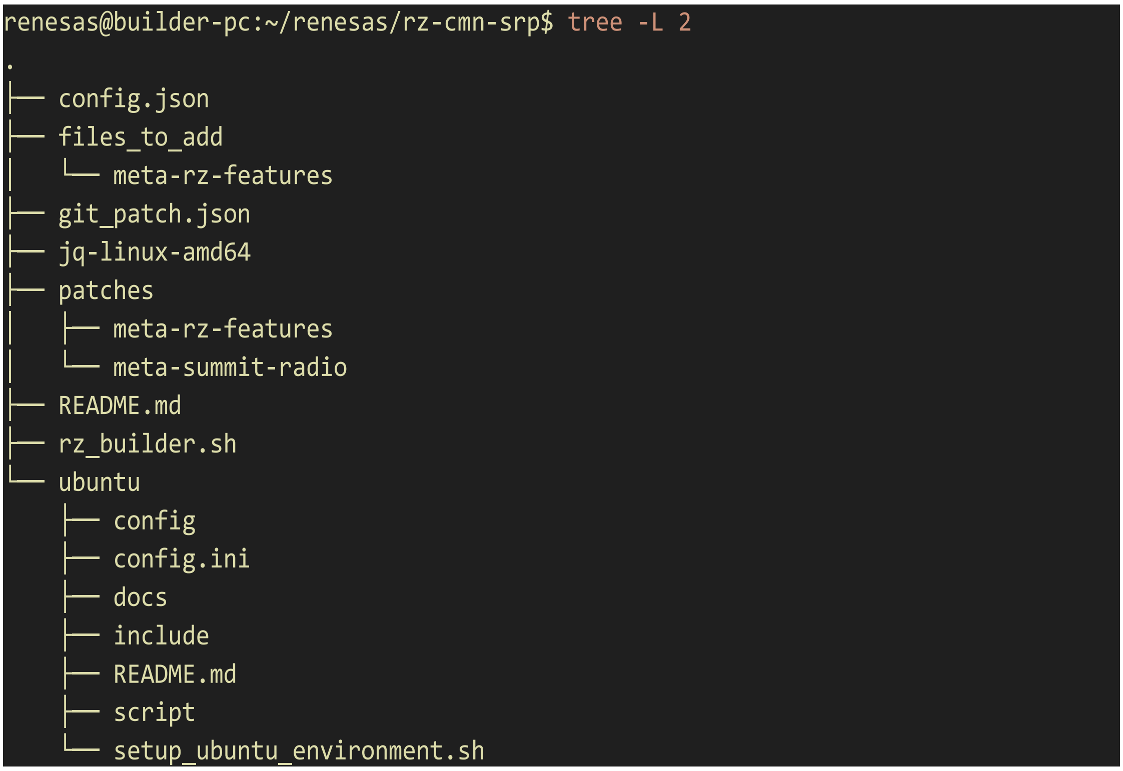
5. Assume all the downloaded zip files from Table 8. List of packages to manually download for Yocto Build are placed in ‘~/Downloads/renesas-yocto’. If locations differ, substitute the corresponding paths in subsequent steps.
6. Copy all the above downloaded zip files to a build folder (For example, ~/renesas/rz-cmn-srp as shown below) in Ubuntu Host PC.

```
$ cd ~/Downloads/renesas-yocto
$ mkdir -p ~/renesas/rz-cmn-srp
$ mv *.zip ~/renesas/rz-cmn-srp
```

7. Copy all the contents in folder ‘host/src/rz-cmn-srp/’ from the release package into ‘~/renesas/rz-cmn-srp’ folder. (This example assumes the pre-requisite files that are described in are located at package unpacked location ~/Downloads/renesas-yocto/rz-cmn-srp-3.0)

```
$ cd ~/Downloads/renesas-yocto/rz-cmn-srp-3.0/host/src/rz-cmn-srp
$ cp -r * ~/renesas/rz-cmn-srp/
```

8. Eventually, all the necessary files for the Yocto build should be present in '~/renesas/rz-cmn-srp folder as shown below.



```
renesas@builder-pc:~/renesas/rz-cmn-srp$ tree -L 2
.
├── config.json
├── files_to_add
│   └── meta-rz-features
├── git_patch.json
├── jq-linux-amd64
├── patches
│   ├── meta-rz-features
│   └── meta-summit-radio
├── README.md
├── rz_builder.sh
└── ubuntu
    ├── config
    ├── config.ini
    ├── docs
    ├── include
    ├── README.md
    ├── script
    └── setup_ubuntu_environment.sh
```

6.2 Builders

The release package provides a copy of the build system from rz-build-scripts repo that was used for creating this release. The general idea is to have a single script that takes inputs and performs all subsequent actions. While most of the build is automated, it is incapable of downloading some essential licensed components at the very beginning. It has a very good CLI and can is intelligent enough to point to the missing components and their download URLs. It is even able to identify the wrong checkouts and correct the branches to a limited capacity. It is governed by the parameters in the config.json along with the inputs provided on the command line while executing the builds.

The build infrastructure is split into two sections:

- **Yocto** – This part is the core foundation to build the BSP and all the yocto images. A build won't produce usable artifacts without it. Hence the yocto build is integrated into the primary build script rz_builder.sh itself.
- **Ubuntu** – This ubuntu build script / infra is placed under its own directory called 'ubuntu'. However, it should ideally be invoked using the master script rz_builder.sh to ensure that the BSP components are build in the yocto OE section and then transferred into the ubuntu image that is built under the ubuntu builder.

The key components are further described in the subsequent sections before heading over to the final build invocation.

6.2.1 rz_builder.sh

The rz_builder.sh script is the primary builder that performs all actions from setting up the build directory to collecting all the final build artifacts and packaging them together into a final release package.

It is capable of invoking ubuntu build as well.

Inputs:

- config.json — authoritative build menu:
  - Supported machines ("machine").
  - Supported images ("images.yocto" and "images.ubuntu").
  - Static groups ("images.static") that expand to multiple images (e.g., all-yocto-images).
  - Defaults ("defaults"), such as MACHINE and IMAGE.
- git_patch.json — repository set and customization plan:
  - Repository origin and revision (url / type / branch, tag, or commit)
  - Patch application list (paths under patches/)
  - File overlays (paths under files_to_add/ → destination in target repos)

Invoking with no parameters prints usage, supported machines/images/groups, and examples.

```
renesas@builder-pc:~/$ cd ~/renesas/rz-cmn-srp/
renesas@builder-pc:~/renesas/rz-cmn-srp$ chmod a+x rz_builder.sh
renesas@builder-pc:~/renesas/rz-cmn-srp$ ./rz_builder.sh
```

Typical operations:

Exact flags may vary by branch; rely on ./rz_builder.sh help for authoritative syntax.

```
# Environment and arguments
MACHINE=<machine> IMAGE=<image|group> DISTRO=<distro> ./rz_builder.sh <target-build>
<target-dir>
```

- MACHINE — target machine from config.json.machine.
- IMAGE — a single image (Yocto/Ubuntu) or a static group from config.json.images.*.
- DISTRO — defaults to poky for Yocto images; automatically set to ubuntu-tiny for Ubuntu images.
- <target_build> — build or build-sdk
- <build_dir> — build workspace name; default directory name (yocto_rzcmn_board) used if omitted.

Common examples:

```
# Yocto: single image (defaults: MACHINE=rz-cmn, DISTRO=poky, IMAGE=core-image-weston)
$ ./rz_builder.sh build ~/build-workspace

#Ubuntu: single image (DISTRO auto-overridden to ubuntu-tiny)
$ MACHINE=rz-cmn IMAGE=ubuntu-lxde \
  ./rz_builder.sh build ~/build-workspace

# Build all Yocto images (static group)
$ MACHINE=rz-cmn IMAGE=all-yocto-images \
  ./rz_builder.sh build ~/build-workspace

#Build all Ubuntu images (static group)
$ MACHINE=rz-cmn IMAGE=all-ubuntu-images \
  ./rz_builder.sh build ~/build-workspace

#Build the full set (Yocto then Ubuntu, in order)
$ MACHINE=rz-cmn IMAGE=all-supported-images \
  ./rz_builder.sh build ~/build-workspace

#SDK build for a Yocto image
$ MACHINE=rz-cmn IMAGE=renesas-core-image-cli \
  ./rz_builder.sh build-sdk ~/build-workspace

#Build full set of SDKs (static group)
$ MACHINE=rz-cmn IMAGE=all-supported-images \
  ./rz_builder.sh build-sdk ~/build-workspace
```

Note: When using build-sdk as <target_build>, the orchestrator builds the selected image first and then generates its matching SDK.

6.2.2 Ubuntu builder

The Ubuntu builder assembles Ubuntu images for RZ platforms by reusing BSP components exported from the Yocto build. It is a sourced helper that is loaded and driven by rz_builder.sh, ensuring the required Yocto artifacts exist and that image assembly proceeds in the correct sequence. It is not intended to operate as an independent CLI.

Location:

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ tree ubuntu -L 1
.
├── config                          # per-variant static configs
├── config.ini                      # configuration (image geometry, policy,..)
├── docs                            # Reference and usage for each Ubuntu image
├── include                         # common and variant functions
├── README.md                       # main README for Ubuntu build
├── script                          # chroot-run helpers script
└── setup_ubuntu_environment.sh     # sourced entry; orchestration helpers
```

6.2.2.1 Ubuntu Builder Configuration (ubuntu/config.ini)

This file defines the core parameters for Ubuntu image assembly. It controls partition sizing, output filenames, base image selection, timezone, user credentials, and optional features such as enabling or disabling the Weston compositor.

The builder automatically discovers and downloads the latest Ubuntu Base (arm64) tarball from the configured Ubuntu CDImage index using URL and filename patterns. If a suitable tarball is already present locally, it will be reused instead of downloading again.

Users may adjust config.ini before running a build to customize system behavior (for example: enlarge the boot partition, change default username and password, or set the correct timezone). These changes directly affect the generated Ubuntu Core or LXDE image.

Core parameters (summary):

- **UBUNTU_TYPE**: Type of target Ubuntu. Available types are "**CORE**", "**LXDE**" and "**ALL**". ("ALL" option will build all Ubuntu types)
- **BOOT_SIZE_MB**: Size of the boot partition in MB. It should be larger than 100MB.
- **WIC_ROOTFS_PARTITION_OVERHEAD_FACTOR**: Overhead factor for the rootfs partition in WIC. Default is 1.3 (30%) is a common default for WIC. Use 1.0 to disable overhead.
- **ROOTFS_INTERNAL_FREE_SPACE_MB**: Default extra free space to add inside the root filesystem (in MB). This space is available to the user/system after booting.
- **UBUNTU_BASE_URL**: The primary url to use for landing at the download page.
- **UBUNTU_BASE_URL_FILE_PATTERN**: The regex pattern used to filter out the requisite image file from the download page.
- **UBUNTU_BASE_FILE_PATTERN**: The file pattern used to search for local cdimage archive previously downloaded. It's passed on to a find command.
- **UBUNTU_LATEST_IMAGE**: A composite variable that automatically determines the image to download from those available at release page.
- **DOWNLOAD_DIR**: Specify the path to place the downloaded cdimage archive. DOWNLOAD_DIR can be set here but will default to './' during execution if left empty here.
- **LOCAL_FILE**: Short hand variable that points to where the previous cdimage archive might be present.
- **OUTPUT_ROOTFS**: The output file name for the rootfs.
- **OUTPUT_WIC**: The output file name for the wic image.
- **TIME_ZONE_AREA**: The time zone area (e.g., "Asia").
- **TIME_ZONE_CITY**: The time zone city (e.g., "Ho_Chi_Minh").
- **IS_WESTON_ENABLE**: Set to 0 to disable Weston compositor.
- **USERNAME:** The default username for logging into the system (e.g., "rz"). This account is used for user login during system access.
- **PASSWORD**: The password associated with the default **USERNAME** (e.g., "1"). This password is required to authenticate the user during login.

6.3 Yocto OE Build

6.3.1 Initiate Yocto Build

Add execute permission to rz_builder.sh

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ chmod a+x rz_builder.sh
```

Commence build.

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ IMAGE=<target-images> ./rz_builder.sh
build
```

See Section 1.1 Supported Distributions for the full list of available images. To build core-image-bsp, use the following command:

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ IMAGE=core-image-bsp ./rz_builder.sh
build
```

> Note: This build requires internet access and takes several hours. Use a build system with high core count, a lot of RAM memory, and a fast SSD to have quicker builds.

6.3.2 Collect the Build Output

After building Yocto with the ‘all-supported-images’ option, which builds all images at once, the output folder should be located at: `~/renesas/rz-cmn-srp/yocto_rzcmn_board/build/tmp/deploy/images/rz-cmn`

The output folder outline should look as follows:

```
renesas@builder-pc:~/renesas/rz-cmn-
srp/yocto_rzcmn_board/build/tmp/deploy/images/rz-cmn$ tree
.
├── host
│   ├── build
│   │   ├── <image-name>-<timestamp>.rootfs.manifest
│   │   ├── <image-name>-<timestamp>.testdata.json
│   │   ├── <image-name>.manifest -> <image-name>-<timestamp>.rootfs.manifest
│   │   └── <image-name>.testdata.json -> <image-name>-<timestamp>.testdata.json
│   ├── env
│   │   ├── core-image-bsp.env
│   │   ├── core-image-minimal.env
│   │   ├── core-image-weston.env
│   │   ├── Readme.md
│   │   ├── renesas-core-image-cli.env
│   │   ├── renesas-core-image-weston.env
│   │   ├── renesas-quickboot-cli.env
│   │   └── renesas-quickboot-wayland.env
│   ├── Readme.md
│   ├── src
│   │   └── rz-cmn-srp
│   │       ├── config.json
│   │       ├── files_to_add
│   │       │   └── meta-rz-features
│   │       │       ├── 0001-rzg2l-sbc-Bring-compat_alloc_user_space-back.patch
│   │       │       └── 0004-rzg2l-sbc-Get-interrupt-number.patch
│   │       ├── git_patch.json
│   │       ├── jq-linux-amd64
│   │       ├── patches
│   │       │   ├── meta-rz-features
│   │       │   │   └── 0001-support-codec-for-linux-6.10-and-yocto-styhead.patch
│   │       │   └── meta-summit-radio
│   │       │       ├── 0001-rz-sbc-meta-summit-radio-Support-build-in-yocto-
styh.patch
│   │       │       └── 0002-rz-sbc-summit-radio-support-eSDK-build.patch
│   │       ├── README.md
│   │       ├── rz_builder.sh
│   │       └── ubuntu
│   │           ├── config
│   │           │   ├── ubuntu_core
│   │           │   │   ├── audio-init-core.sh
│   │           │   │   ├── network_interfaces.conf
│   │           │   │   ├── NetworkManager.conf
│   │           │   │   └── resolved.conf
│   │           │   └── ubuntu_lxde
│   │           │       ├── audio-init-lxde.sh
│   │           │       ├── connman-gtk.desktop
│   │           │       ├── force-display-xorg.sh
│   │           │       ├── force-xorg-display.service
│   │           │       ├── interfaces
│   │           │       ├── lightdm.conf
│   │           │       ├── NetworkManager.conf
│   │           │       ├── panel
│   │           │       ├── rsyslog
│   │           │       ├── ttyS0.conf
│   │           │       └── v4l2-init.sh
│   │           ├── config.ini
│   │           ├── docs
│   │           │   ├── ubuntu_core
│   │           │   │   └── README.md
│   │           │   └── ubuntu_lxde
│   │           │       ├── Pictures
│   │           │       │   ├── audacity.png
│   │           │       │   ├── audio_settings.png
│   │           │       │   ├── bluetooth_0.png
│   │           │       │   ├── bluetooth_1.png
│   │           │       │   ├── bluetooth_2.png
│   │           │       │   ├── bluetooth_3.png
│   │           │       │   ├── bluetooth_4.png
│   │           │       │   ├── csi_0.png
│   │           │       │   ├── csi_1.png
│   │           │       │   ├── csi_2.png
│   │           │       │   ├── eth_1.png
│   │           │       │   ├── eth_2.png
│   │           │       │   ├── eth_3.png
│   │           │       │   ├── eth_4.png
│   │           │       │   ├── eth_5.png
│   │           │       │   ├── eth.png
│   │           │       │   ├── save_audio_0.png
│   │           │       │   ├── save_audio_1.png
│   │           │       │   ├── save_audio_2.png
│   │           │       │   ├── vlc_open_0.png
│   │           │       │   ├── vlc_open_1.png
│   │           │       │   ├── vlc_open_2.png
│   │           │       │   ├── vlc.png
│   │           │       │   ├── vlc_video_1.png
│   │           │       │   ├── vlc_video.png
│   │           │       │   ├── web_1.png
│   │           │       │   ├── web_2.png
│   │           │       │   ├── web_lxterm_htop.png
│   │           │       │   ├── web.png
│   │           │       │   └── wifi_0.png
│   │           │       └── README.md
│   │           ├── include
│   │           │   ├── common
│   │           │   │   ├── allow_empty_password.sh
│   │           │   │   ├── create_wic.sh
│   │           │   │   ├── install_gstreamer.sh
│   │           │   │   ├── install_weston.sh
│   │           │   │   ├── mount.sh
│   │           │   │   ├── prepare_env_rootfs.sh
│   │           │   │   ├── prepare_env.sh
│   │           │   │   ├── prepare_ubuntu_base.sh
│   │           │   │   └── yocto_working.sh
│   │           │   ├── ubuntu_core
│   │           │   │   ├── prepare_conf.sh
│   │           │   │   ├── prepare_env.sh
│   │           │   │   ├── prepare_rootfs.sh
│   │           │   │   └── setup_dns.sh
│   │           │   └── ubuntu_lxde
│   │           │       ├── create_swap.sh
│   │           │       ├── prepare_conf.sh
│   │           │       └── prepare_rootfs_qt.sh
│   │           ├── README.md
│   │           ├── script
│   │           │   ├── common
│   │           │   │   ├── dpkg-install-lock-fix.sh
│   │           │   │   └── setup_dns_and_time.sh
│   │           │   ├── ubuntu_core
│   │           │   │   ├── apt_install_base.sh
│   │           │   │   ├── link_to_leagcy_iptables.sh
│   │           │   │   └── set_root_password.sh
│   │           │   └── ubuntu_lxde
│   │           │       ├── apt_audio_video.sh
│   │           │       ├── apt_blueman.sh
│   │           │       ├── apt_install_base.sh
│   │           │       ├── apt_lxde_desktop.sh
│   │           │       ├── apt_wifi_ble.sh
│   │           │       ├── create_user.sh
│   │           │       ├── enable_service.sh
│   │           │       ├── set_root_password.sh
│   │           │       ├── set_swap_enable.sh
│   │           │       └── setup-set-permissions.sh
│   │           └── setup_ubuntu_environment.sh
│   └── tools
│       ├── bin
│       │   ├── linux
│       │   │   ├── bpgen
│       │   │   ├── fiptool
│       │   │   └── Readme.md
│       │   ├── Readme.md
│       │   └── windows
│       │       ├── bpgen.exe
│       │       ├── fiptool.exe
│       │       └── Readme.md
│       ├── bootloader_flasher
│       │   ├── bootloader_flash.py
│       │   └── README.md
│       ├── config
│       │   ├── boards_flash_config.toml
│       │   └── README.md
│       ├── firmware_compile
│       │   ├── firmware_compile.py
│       │   └── Readme.md
│       ├── flash_images.json
│       ├── README.md
│       ├── sd_creator
│       │   ├── README.md
│       │   ├── sd_flash.py
│       │   └── tools
│       │       ├── AdbWinApi.dll
│       │       └── fastboot.exe
│       ├── uload_bootloader
│       │   ├── README.md
│       │   └── uload_bootloader_flash.py
│       └── universal_flash.py
├── license
│   └── Disclaimer051.pdf
├── <code>-rz-cmn-srp-um-quick-start-guide.pdf
├── <code>-rz-cmn-srp-um.pdf
├── README.md
├── RZ_System_Release_Package_Evaluation_license.pdf
└── target
    ├── env
    │   ├── Readme.md
    │   └── uEnv.txt
    ├── images
    │   ├── atf
    │   │   ├── bl2-rz-cmn.bin
    │   │   ├── bl31-rz-cmn.bin
    │   │   ├── fdts
    │   │   │   ├──  <board-name>.dtb
    │   │   │   └──  Readme.md
    │   │   └── Readme.md
    │   ├── core-image-bsp.wic
    │   ├── core-image-minimal.wic
    │   ├── core-image-weston.wic
    │   ├── Flash_Writer_SCIF_<board-name>.mot
    │   ├── Flash_Writer_SCIF_<board-name>_PMIC.mot
    │   ├── linux
    │   │   ├── dtbs
    │   │   │   ├── overlays
    │   │   │   │   ├── Readme.md
    │   │   │   │   ├── rzg2l-sbc-can.dtbo
    │   │   │   │   ├── rzg2l-sbc-dsi.dtbo
    │   │   │   │   ├── rzg2l-sbc-ext-i2c.dtbo
    │   │   │   │   ├── rzg2l-sbc-ext-spi.dtbo
    │   │   │   │   └── rzg2l-sbc-ov5640.dtbo
    │   │   │   ├── <board-name>--<kernel-version>-rz-cmn-<timestamp>.dtbo
    │   │   │   ├── <board-name>.dtb -> <board-name>--<kernel-version>-rz-cmn-
<timestamp>.dtbo
    │   │   │   └── Readme.md
    │   │   ├── Image -> Image--<kernel-version>-rz-cmn-<timestamp>.bin
    │   │   ├── Image--<kernel-version>-rz-cmn-<timestamp>.bin
    │   │   └── Readme.md
    │   ├── Readme.md
    │   ├── renesas-core-image-cli.wic
    │   ├── renesas-core-image-weston.wic
    │   ├── renesas-quickboot-cli.wic
    │   ├── renesas-quickboot-wayland.wic
    │   ├── rootfs
    │   │   ├── core-image-bsp.tar.bz2
    │   │   ├── core-image-minimal.tar.bz2
    │   │   ├── core-image-weston.tar.bz2
    │   │   ├── Readme.md
    │   │   ├── renesas-core-image-cli.tar.bz2
    │   │   ├── renesas-core-image-weston.tar.bz2
    │   │   ├── renesas-quickboot-cli.tar.bz2
    │   │   └── renesas-quickboot-wayland.tar.bz2
    │   ├── <board>-<version>-platform-settings.bin
    │   ├── <board>-<version>-platform-settings.srec
    │   └── u-boot
    │       ├── dtbs
    │       │   ├── Readme.md
    │       │   └── <board-name>.dtb
    │       ├── Readme.md
    │       └── u-boot-nodtb-rz-cmn.bin
    └── Readme.md
```

6.4 Ubuntu OE Build

This section describes how to build a custom Ubuntu Core image for the RZ-based system. The process involves using the chroot method, which creates an isolated environment, and utilizes a compressed ubuntu-base file as the foundation. The overall build process relies on a Yocto-based environment to generate the necessary system files. The steps outlined below cover the entire process, including setting up the build host environment and creating the Ubuntu image for the RZ-based system.

What is chroot?

chroot (change root) is a Unix command that changes the apparent root directory for a process and its children, effectively isolating the environment for building or running applications. This is useful for creating custom system images, as it ensures that the environment is clean and separated from the host system.

6.4.1 Ubuntu Build Host Environment Setup

To begin the build process, it is essential to set up the build host environment. This involves configuring the necessary tools and dependencies to prepare for the Ubuntu image build.

The preparation steps for the Ubuntu build host are the same as those described for the Yocto build in 6.1. Build Host Environment Setup. The same workspace, dependencies, and prerequisite files must be in place.

Table 9. Prerequisite files from the release package for Ubuntu build

File	Description
rz-cmn-srp/	Main folder for Yocto/Ubuntu builds environment for RZ platforms.
rz_builder.sh	Custom master build script that downloads required packages and ZIP files, configures meta layers, sets up the environment, and builds for both Yocto and Ubuntu target images.
site.conf	An override file that targets a specific build version.

patches	This is a folder that contains additional patches that are needed for Yocto eSDK build. The patches are organized as follows: • meta-summit-radio/ • 0001-rz-sbc-meta-summit-radio-Support-build-in-yocto-styh.patch • 0002-rz-sbc-summit-radio-support-eSDK-build.patch • meta-rz-features/ • 0001-support-codec-for-linux-6.10-and-yocto-styhead.patch
files_to_add	A folder containing additional files that need to be added to the meta layer. These files are specified in the add_files field of git_patch.json, which defines their source locations and target destinations. • meta-rz-features / • 0001-rzg2l-sbc-Bring-compat_alloc_user_space-back.patch • 0004-rzg2l-sbc-Get-interrupt-number.patch
git_patch.json	A configuration file contains JSON keys and repository configuration such as: url, branch, tag, commit, repo type and patch paths to apply.
config.json	Contains the config options including available build image options grouped by build type, including Yocto images, Ubuntu images, and static image collections (all-yocto-images, all-ubuntu-images, all-supported-images).
jq-linux-amd64	A lightweight and flexible tool that supports parsing JSON file.
README.md	A README file describing all the necessary info about the building process.
ubuntu/	Main folder for Ubuntu-based image generation for RZ platform.
ubuntu/config	The folder that holds configuration files for different Ubuntu variants.
ubuntu/docs	Contains documentation detailing supported features and usage instructions for each Ubuntu image variant.
ubuntu/include	Contains scripts related to Ubuntu (e.g., for creating WIC files, packaging the root filesystem, preparing the environment, etc.)
ubuntu/script	The folder that contains all scripts related to Ubuntu image creation.
ubuntu/setup_ubuntu_environment.sh	Main entry-point script (acts like a dispatcher/header). It sources and sequences logic from the modular scripts under script/. It does not build anything by itself.
ubuntu/config.ini	Configuration file that defines key parameters for the Ubuntu image build process, such as the Ubuntu variant, base image, output filenames, and system settings.

Note: The Ubuntu build process relies on artifacts generated by the Yocto environment, such as the kernel, bootloader, and device tree. These are produced through the included Yocto build scripts and are required for creating a functional Ubuntu image for the RZ/G2L-SBC.

Eventually, all the necessary files for the Ubuntu build should be present in ‘renesas/rz-cmn-srp’ folder as shown below.

```
renesas@builder-pc:~/renesas/$ tree -L 3
.
└── rz-cmn-srp
    ├── files_to_add
    │   └── meta-rz-features
    ├── git_patch.json
    ├── images.json
    ├── config.json
    ├── jq-linux-amd64
    ├── patches
    │   ├── meta-summit-radio
    │   └── poky
    ├── README.md
    ├── rz_builder.sh
    └── ubuntu
        ├── config
        ├── config.ini
        ├── docs
        ├── include
        ├── script
        └── setup_ubuntu_environment.sh
```

6.4.2 Initial Ubuntu Build

Before starting the build, review the Ubuntu Builder Configuration (ubuntu/config.ini) file and adjust any required settings such as partition sizes, credentials, or base image selection. Once configured, add execute permission to rz_builder.sh and run it with the appropriate options to generate the desired Ubuntu image.

> Note: Host PC with Ubuntu 24.04 is recommended for the build. Prepare environment for building package and local build environment

To perform a build, first go to rz-cmn-srp folder

```
renesas@builder-pc:~/$ cd ~/renesas/rz-cmn-srp
```

Add execute permission to rz_builder.sh.

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ chmod a+x rz_builder.sh
```

Before running the build script, please ensure that this source belongs to a regular user (not root or a privileged user), and the user executing this must have sudo/root privileges.

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ IMAGE=<target-image> ./rz_builder.sh
build
```

Run the following command with the appropriate option:

- ubuntu-core: Build Ubuntu core image
- ubuntu-lxde: Build Ubuntu LXDE image (with graphic stacks).

- all-ubuntu-images: Build both Ubuntu LXDE and Ubuntu Core

For example to build ubuntu-lxde image:

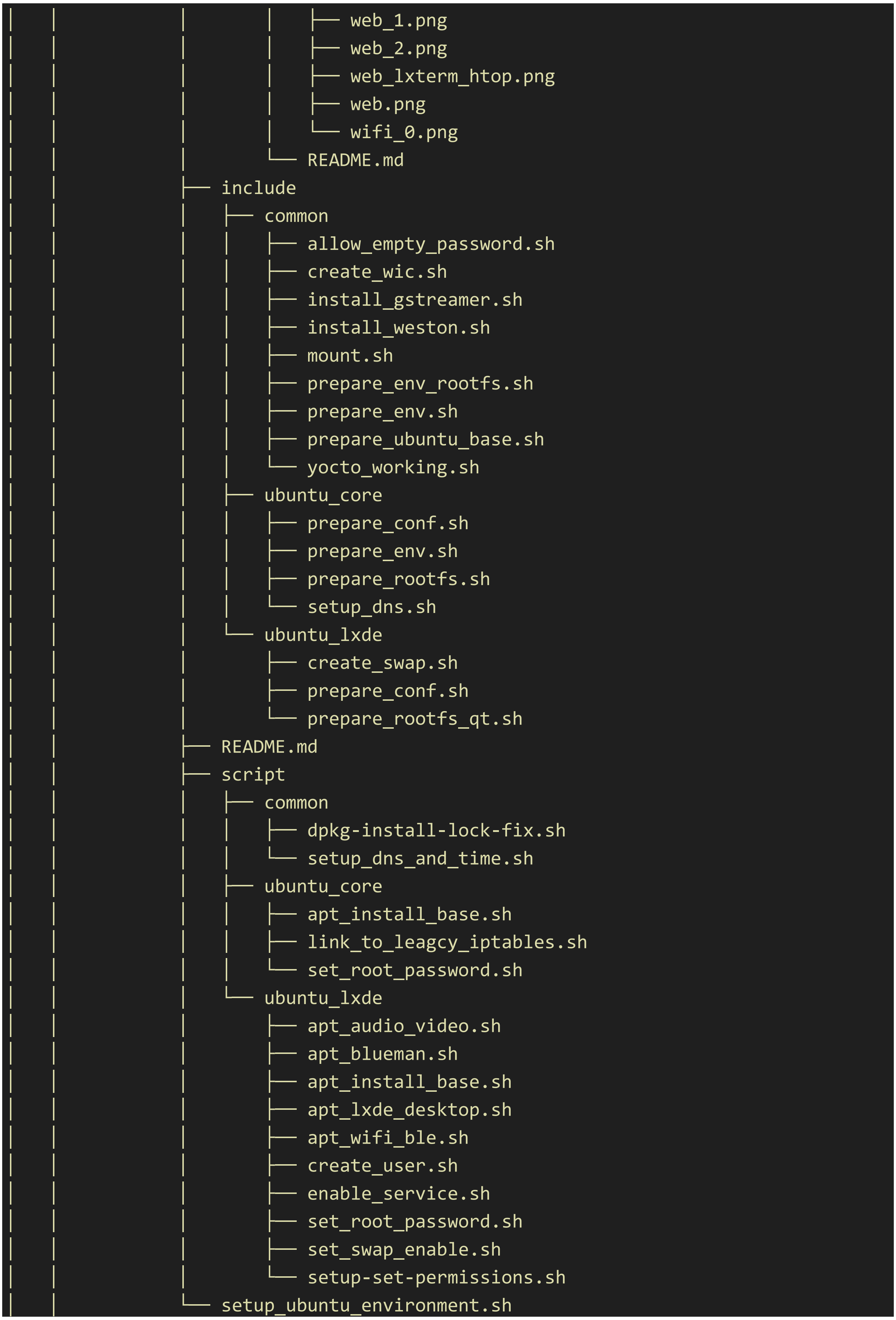
```
renesas@builder-pc:~/renesas/rz-cmn-srp$ IMAGE=ubuntu-lxde ./rz_builder.sh build
```

Note: Please note that this build requires internet access and will take several hours. Use a build system with high core count, lot of RAM memory and a fast SSD to have quicker builds.

6.4.3 Collect the Build Output

After building Ubuntu LXDE and Ubuntu Core, the output folder should be located at: `~/renesas/rz-cmn-srp/yocto_rzcmn_board/build/tmp/deploy/images/rz-cmn`
The output folder outline should look as follows:

```
renesas@builder-pc:~/renesas/rz-cmn-
srp/yocto_rzcmn_board/build/tmp/deploy/images/rz-cmn$ tree
.
├── host
│   ├── build
│   │   ├── <image-name>-<timestamp>.rootfs.manifest
│   │   ├── <image-name>-<timestamp>.testdata.json
│   │   ├── <image-name>.manifest -> <image-name>-<timestamp>.rootfs.manifest
│   │   └── <image-name>.testdata.json -> <image-name>-<timestamp>.testdata.json
│   ├── env
│   │   ├── Readme.md
│   │   └── renesas-ubuntu.env
│   ├── Readme.md
│   ├── src
│   │   └── rz-cmn-srp
│   │       ├── config.json
│   │       ├── files_to_add
│   │       │   └── meta-rz-features
│   │       │       ├── 0001-rzg2l-sbc-Bring-compat_alloc_user_space-back.patch
│   │       │       └── 0004-rzg2l-sbc-Get-interrupt-number.patch
│   │       ├── git_patch.json
│   │       ├── jq-linux-amd64
│   │       ├── patches
│   │       │   ├── meta-rz-features
│   │       │   │   └── 0001-support-codec-for-linux-6.10-and-yocto-styhead.patch
│   │       │   └── meta-summit-radio
│   │       │       ├── 0001-rz-sbc-meta-summit-radio-Support-build-in-yocto-
styh.patch
│   │       │       └── 0002-rz-sbc-summit-radio-support-eSDK-build.patch
│   │       ├── README.md
│   │       ├── rz_builder.sh
│   │       └── ubuntu
│   │           ├── config
│   │           │   ├── ubuntu_core
│   │           │   │   ├── audio-init-core.sh
│   │           │   │   ├── network_interfaces.conf
│   │           │   │   ├── NetworkManager.conf
│   │           │   │   └── resolved.conf
│   │           │   └── ubuntu_lxde
│   │           │       ├── audio-init-lxde.sh
│   │           │       ├── connman-gtk.desktop
│   │           │       ├── force-display-xorg.sh
│   │           │       ├── force-xorg-display.service
│   │           │       ├── interfaces
│   │           │       ├── lightdm.conf
│   │           │       ├── NetworkManager.conf
│   │           │       ├── panel
│   │           │       ├── rsyslog
│   │           │       ├── ttyS0.conf
│   │           │       └── v4l2-init.sh
│   │           ├── config.ini
│   │           ├── docs
│   │           │   ├── ubuntu_core
│   │           │   │   └── README.md
│   │           │   └── ubuntu_lxde
│   │           │       ├── Pictures
│   │           │       │   ├── audacity.png
│   │           │       │   ├── audio_settings.png
│   │           │       │   ├── bluetooth_0.png
│   │           │       │   ├── bluetooth_1.png
│   │           │       │   ├── bluetooth_2.png
│   │           │       │   ├── bluetooth_3.png
│   │           │       │   ├── bluetooth_4.png
│   │           │       │   ├── csi_0.png
│   │           │       │   ├── csi_1.png
│   │           │       │   ├── csi_2.png
│   │           │       │   ├── eth_1.png
│   │           │       │   ├── eth_2.png
│   │           │       │   ├── eth_3.png
│   │           │       │   ├── eth_4.png
│   │           │       │   ├── eth_5.png
│   │           │       │   ├── eth.png
│   │           │       │   ├── save_audio_0.png
│   │           │       │   ├── save_audio_1.png
│   │           │       │   ├── save_audio_2.png
│   │           │       │   ├── vlc_open_0.png
│   │           │       │   ├── vlc_open_1.png
│   │           │       │   ├── vlc_open_2.png
│   │           │       │   ├── vlc.png
│   │           │       │   ├── vlc_video_1.png
│   │           │       │   ├── vlc_video.png
```



```
│   │           │       │   ├── web_1.png
│   │           │       │   ├── web_2.png
│   │           │       │   ├── web_lxterm_htop.png
│   │           │       │   ├── web.png
│   │           │       │   └── wifi_0.png
│   │           │       └── README.md
│   │           ├── include
│   │           │   ├── common
│   │           │   │   ├── allow_empty_password.sh
│   │           │   │   ├── create_wic.sh
│   │           │   │   ├── install_gstreamer.sh
│   │           │   │   ├── install_weston.sh
│   │           │   │   ├── mount.sh
│   │           │   │   ├── prepare_env_rootfs.sh
│   │           │   │   ├── prepare_env.sh
│   │           │   │   ├── prepare_ubuntu_base.sh
│   │           │   │   └── yocto_working.sh
│   │           │   ├── ubuntu_core
│   │           │   │   ├── prepare_conf.sh
│   │           │   │   ├── prepare_env.sh
│   │           │   │   ├── prepare_rootfs.sh
│   │           │   │   └── setup_dns.sh
│   │           │   └── ubuntu_lxde
│   │           │       ├── create_swap.sh
│   │           │       ├── prepare_conf.sh
│   │           │       └── prepare_rootfs_qt.sh
│   │           ├── README.md
│   │           ├── script
│   │           │   ├── common
│   │           │   │   ├── dpkg-install-lock-fix.sh
│   │           │   │   └── setup_dns_and_time.sh
│   │           │   ├── ubuntu_core
│   │           │   │   ├── apt_install_base.sh
│   │           │   │   ├── link_to_leagcy_iptables.sh
│   │           │   │   └── set_root_password.sh
│   │           │   └── ubuntu_lxde
│   │           │       ├── apt_audio_video.sh
│   │           │       ├── apt_blueman.sh
│   │           │       ├── apt_install_base.sh
│   │           │       ├── apt_lxde_desktop.sh
│   │           │       ├── apt_wifi_ble.sh
│   │           │       ├── create_user.sh
│   │           │       ├── enable_service.sh
│   │           │       ├── set_root_password.sh
│   │           │       ├── set_swap_enable.sh
│   │           │       └── setup-set-permissions.sh
│   │           └── setup_ubuntu_environment.sh
│   └── tools
│       ├── bin
│       │   ├── linux
│       │   │   ├── bpgen
│       │   │   ├── fiptool
│       │   │   └── Readme.md
│       │   ├── Readme.md
│       │   └── windows
│       │       ├── bpgen.exe
│       │       ├── fiptool.exe
│       │       └── Readme.md
│       ├── bootloader_flasher
│       │   ├── bootloader_flash.py
│       │   └── README.md
│       ├── config
│       │   ├── boards_flash_config.toml
│       │   └── README.md
│       ├── firmware_compile
│       │   ├── firmware_compile.py
│       │   └── Readme.md
│       ├── flash_images.json
│       ├── README.md
│       ├── sd_creator
│       │   ├── README.md
│       │   ├── sd_flash.py
│       │   └── tools
│       │       ├── AdbWinApi.dll
│       │       └── fastboot.exe
│       ├── uload_bootloader
│       │   ├── README.md
│       │   └── uload_bootloader_flash.py
│       └── universal_flash.py
├── license
│   └── Disclaimer051.pdf
├── <code>-rz-cmn-srp-um-quick-start-guide.pdf
├── <code>-rz-cmn-srp-um.pdf
├── README.md
├── RZ_System_Release_Package_Evaluation_license.pdf
└── target
    ├── env
    │   ├── Readme.md
    │   └── uEnv.txt
    ├── images
    │   ├── atf
    │   │   ├── bl2-rz-cmn.bin
    │   │   ├── bl31-rz-cmn.bin
    │   │   ├── fdts
    │   │   │   ├──  <board-name>.dtb
    │   │   │   └──  Readme.md
    │   │   └── Readme.md
    │   ├── Flash_Writer_SCIF_<board-name>.mot
    │   ├── Flash_Writer_SCIF_<board-name>_PMIC.mot
    │   ├── linux
    │   │   ├── dtbs
    │   │   │   ├── overlays
    │   │   │   │   ├── Readme.md
    │   │   │   │   ├── rzg2l-sbc-can.dtbo
    │   │   │   │   ├── rzg2l-sbc-dsi.dtbo
    │   │   │   │   ├── rzg2l-sbc-ext-i2c.dtbo
    │   │   │   │   ├── rzg2l-sbc-ext-spi.dtbo
    │   │   │   │   └── rzg2l-sbc-ov5640.dtbo
    │   │   │   ├── <board-name>--<kernel-version>-rz-cmn-<timestamp>.dtbo
    │   │   │   ├── <board-name>.dtb -> <board-name>--<kernel-version>-rz-cmn-
<timestamp>.dtbo
    │   │   │   └── Readme.md
    │   │   ├── Image -> Image--<kernel-version>-rz-cmn-<timestamp>.bin
    │   │   ├── Image--<kernel-version>-rz-cmn-<timestamp>.bin
    │   │   └── Readme.md
    │   ├── Readme.md
    │   ├── rootfs
    │   │   ├── Readme.md
    │   │   ├── renesas-ubuntu.tar.bz2
    │   │   ├── ubuntu-core-image.tar.bz2
    │   │   └── ubuntu-lxde-image.tar.bz2
    │   ├── <board>-<version>-platform-settings.bin
    │   ├── <board>-<version>-platform-settings.srec
    │   ├── u-boot
    │   │   ├── dtbs
    │   │   │   ├── Readme.md
    │   │   │   └── <board-name>.dtb
    │   │   ├── Readme.md
    │   │   └── u-boot-nodtb-rz-cmn.bin
    │   ├── ubuntu-core-image.wic.gz
    │   └── ubuntu-lxde-image.wic.gz
    └── Readme.md
```

7. Programming / Flashing Images

This section explains how to program and flash various firmware and root file system images onto Renesas boards. It covers firmware components, prerequisites, hardware setup for each board, and usage of the universal flashing script for seamless flashing workflows.

This package contains only the following firmware components:

Table 10. Firmware description

Module	Binary	Stack layer	Description
ROM code	N/A	BL1	This is the internal ROM code that the Arm Cortex SoC's primary core executes at POR.
Flash writer	Flash_Writer_SCIF_ <board>.mot	BL2	This is meant for serial load in factory environments, which is directly loaded onto the SRAM by the BL1 (ROM code) through UART SCIF0. It is then executed to acquire another image on UART SCIF0 to directly flash onto qspi or emmc into the boot sector. It provides a command-based ui.
Arm trusted Firmware-A	bl2-rz-cmn.bin bl31-rz-cmn.bin <board>.dtb	BL2 and BL31	Minimal Trusted Firmware-A implementation without a device tree. The flashing script will combine bl2-rz-cmn.bin with device tree dynamically during flashing. It comes in only .bin format: • .bin – for raw flashing for native in-system flashing
U-Boot (BL33)	u-boot-nodtb-rz-cmn.bin <board>.dtb	BL33	U-Boot (nodtb) binary and matching device tree. The flashing script combines these into the FIP.
Board Identification	<board>-platform-settings.bin		This binary stores key platform settings for Renesas boards, like model IDs, revisions, memory locations, and image sizes; enabling firmware and bootloaders to identify hardware and locate boot components efficiently during startup or flashing.

Note: This release does **not** ship a prebuilt FIP. Instead, the flashing script will automatically build a valid FIP image at flash time by combining:

- bl31-rz-cmn.bin
- u-boot-nodtb-rz-cmn.bin
- <board>.dtb

It also dynamically merges bl2-rz-cmn.bin with <board>.dtb to create the BL2 binary that is flashed to the boot sector.

7.1 Prerequisites

Before flashing any images, ensure the following system requirements are met on your host PC and that necessary files and tools are available.

- Operating System:
    - Linux (Ubuntu 20.04 or newer recommened)
    - Windows 10 or newer
- Software Requirements:
    - Python 3.8 or later
    - GNU Binutils: Required for objcopy.
    - Firmware release package with images and tools
- Hardware requirements:
    - Required cables: USB, UART debug cables
    - SD card (8GB or larger)

7.1.1 Linux Setup

Follow these steps to prepare a Linux host:

1. Install Python, Binutils and build tools

```
renesas@builder-pc:~# sudo apt update
renesas@builder-pc:~# sudo apt install python3 python3-pip binutils build-essential libssl-dev
```

- python3, python3-pip: Required to run host scripts
- binutils: Provides objcopy for binary conversion
- build-essential (optional): Installs gcc, g++, make for rebuilding firmware
- libssl-dev: OpenSSL package

2. Install Python Dependencies

If Python 3.12 is in use: set up a virtual environment first.

```
renesas@builder-pc:~/rz-cmn-srp-3.0/host/tools# sudo apt install python3.12-venv
renesas@builder-pc:~/rz-cmn-srp-3.0/host/tools# python3 -m venv .venv
renesas@builder-pc:~/rz-cmn-srp-3.0/host/tools# source .venv/bin/activate
```

After the virtual environment is active, choose one of the two install methods:

- Option 1 – Use **requirements.txt** (recommended)

```
renesas@builder-pc:~# cd <path/to/the/package>/host/tools/
renesas@builder-pc:~/rz-cmn-srp-3.0/host/tools$ pip3 install -r requirements.txt
```

- Option 2 – Install manually

```
renesas@builder-pc:~# pip install pyserial
renesas@builder-pc:~# pip install dataclasses (if using python <3.7)
```

7.1.2 Windows Setup

Follow these steps to prepare a Windows host:

1. Install Python3
    - Download and install Python 3 from python.org
    - During installation, enable “Add Python to environment variables”
2. If pip is missing, repair your Python installation or download get-pip.py and run:

```
PS C:\Users\renesas> py get-pip.py
```

3. Install Python Dependencies: Open one of the following terminals with Administrator privileges:
   - PowerShell
   - Git Bash

   The example below uses PowerShell, but the same applies to other terminals
   - Option 1 – Use **requirements.txt** (recommended)

```
PS C:\Users\renesas> cd <path/to/the/package/host/tools>
PS C:\Users\renesas\rz-cmn-srp-3.0\host\tools> py -m pip install -r
requirements.txt
```

   - Option 2 – Install manually
     - Using the Python launcher:

```
PS C:\Users\renesas> py -m pip install pyserial
PS C:\Users\renesas> py -m pip install tomli
PS C:\Users\renesas> py -m pip install dataclasses       # Only if Python < 3.7
```

     - Or using pip directly (if already in PATH)

```
PS C:\Users\renesas> pip install pyserial
PS C:\Users\renesas> pip install tomli
PS C:\Users\renesas> pip install dataclasses # required only if using Python
versions older than 3.7
```

4. Environment and Tool Dependencies
   - GNU Binutils:
     - Download and install MinGW-w64
     - Install to default location (**C:/MinGW**)
     - Add the following path to the Windows Environment Variables → Path:

```
C:/MinGW/bin
```

   - OpenSSL (For MingW-w64):
     - Download the package from: MinGW-w64 OpenSSL
     - Extract the package into: **C:/mingw64**

Important Notice for Windows users: Executables such as fiptool.exe depend on OpenSSL runtime DLLs.

- Add this directory to your Environment Variables → Path:
  - ➢ **C:/mingw64/bin**
- Or copy the DLLs (C:\mingw64\bin\libcrypto-3-x64.dll) into:
  - ➢ **<path/to/package>/tools/bin/windows/**
- If skipped, running the tools will fail

The firmware_compile.py script also depends on objcopy (part of GNU binutils).

- Ensure **C:/MinGW/bin** is also in Windows Environment Variables Path so that objcopy.exe can be found.
- Without it, the script will fail during SREC/ELF conversions.

7.2 Hardware Setup

Flashing methods depend on the type of image being flashed. Follow the setup that matches the operation.

RZ boards are shipped with pre-programmed firmware. However, in scenarios such as factory flashing, image corruption recovery, or when deploying custom builds, re-flashing may be necessary. The Renesas System Package includes tools to perform firmware and root filesystem flashing across different host OS environments.

7.2.1 RZ/G2L-SBC

The RZ/G2L-SBC images consist of:

1. Trusted firmware
2. Multi-stage bootloaders.
3. Linux demo distribution.

The SBC board is designed to boot from QSPI EEPROM containing the trusted firmware and bootloaders. However, SBC does not have eMMC storage, and the Linux image is expected to be available on an SD card or a TFTP server.

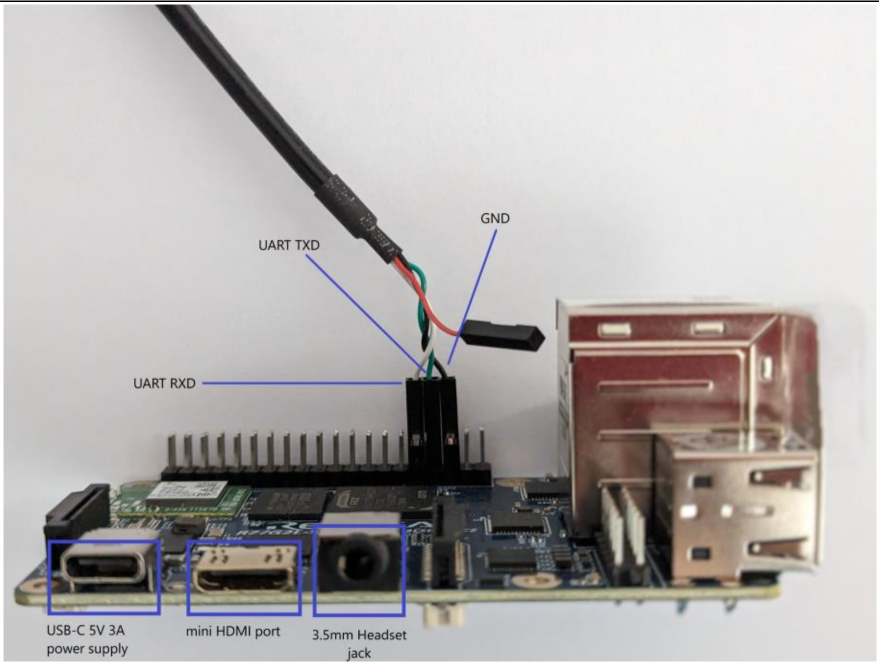


Figure 16. Cortex A55 debug UART cable interface

1. The SD card with the Linux boot image from the release.
2. A USB cable connected to the host PC for flashing and debugging
3. A 5V 3A USB-C power supply.

Other interfaces are not necessary for this purpose.

7.2.2 RZ/G2L-EVK and RZ/V2L-EVK

Both the RZ/G2L-EVK and RZ/V2L-EVK development kits ship with pre-programmed firmware and Linux images and include onboard eMMC storage. This allows both firmware and root filesystem images to be flashed directly onto the board, making them suitable for development and production scenarios where frequent updates or customization are required.

Firmware and root filesystem flashing can be done directly onto the eMMC, facilitating faster development cycles and easier updates.

Typical flashing requires:

- The SD card with the Linux boot image from the release.
- A USB Type-A to Micro USB Type-B cable connected to the host PC for flashing and debugging
- A USB type C charger 65W
  - Support USB PD (Power only)
  - Output specification: 5V3A,9V3A,15V3A,20V3.25A

Other interfaces are not necessary for this purpose.

7.2.3 RZ/V2H-EVK

The RZ/V2H-EVK development kit typically ships with pre-programmed firmware and Linux images. However, updating or customizing these images is common in development or production workflows.

Typical flashing requires:

- The SD card with the Linux boot image from the release.
- A USB Type-A to Micro USB Type-B cable connected to the host PC for flashing and debugging
- Power supply that can provide up to 100W via USB-C PD (not included in the package).

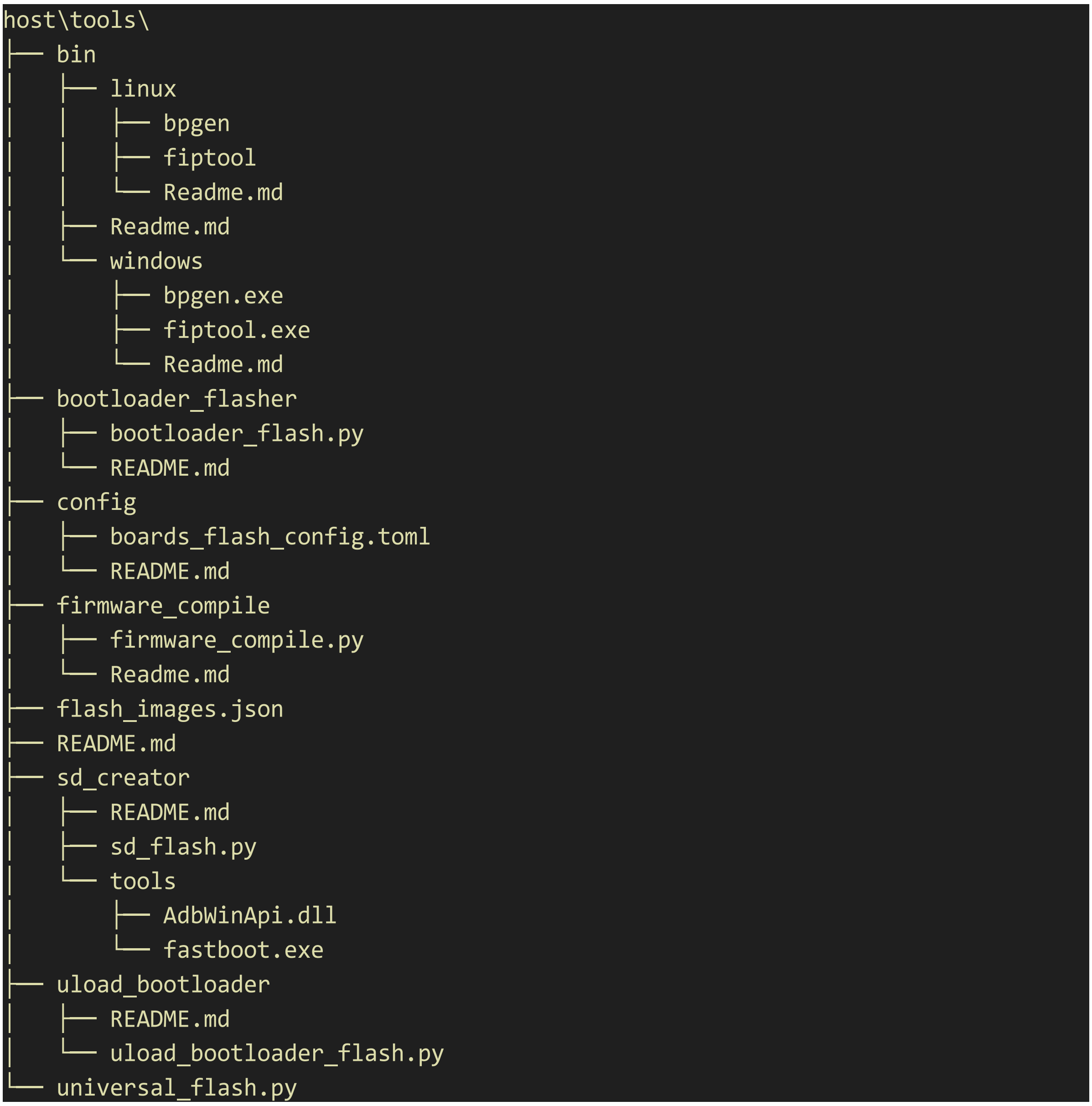
Other interfaces are not necessary for this purpose.

7.3 Universal Flashing Script

The universal_script.py is a cross-platform script designed to simplify various flashing workflows for Renesas boards. It utilizes a board configuration JSON file to manage image mappings and provides a unified interface for different flashing operations. This is the recommended tool for most flashing tasks.

It’s located in the Yocto build output: </path/to/your/yocto/package>/host/tools/universal_script.py

Below is the hierarchy of tools folder



```
host\tools\
├── bin
│   ├── linux
│   │   ├── bpgen
│   │   ├── fiptool
│   │   └── Readme.md
│   ├── Readme.md
│   └── windows
│       ├── bpgen.exe
│       ├── fiptool.exe
│       └── Readme.md
├── bootloader_flasher
│   ├── bootloader_flash.py
│   └── README.md
├── config
│   ├── boards_flash_config.toml
│   └── README.md
├── firmware_compile
│   ├── firmware_compile.py
│   └── Readme.md
├── flash_images.json
├── README.md
├── sd_creator
│   ├── README.md
│   ├── sd_flash.py
│   └── tools
│       ├── AdbWinApi.dll
│       └── fastboot.exe
├── uload_bootloader
│   ├── README.md
│   └── uload_bootloader_flash.py
└── universal_flash.py
```

7.3.1 flash_images.json – File Overview and Usage

The flash_images.json file is a board-to-image mapping configuration used by the flashing framework. It serves as a single source of truth that lists:

- Which images (BL2, FIP, U-Boot DTBs, rootfs, etc.) belong to each supported board
- Where those images are in the release package
- Which flashing methods (e.g., xSPI vs eMMC, UDP vs OTG) are applicable to that board

This file is mandatory for universal_script.py, which parses it to select the correct binaries and flashing procedure for a given board. In short, it links boards → binaries → flashing operations.

Location:

- The file itself must be in the same directory as universal_script.py.
- Referenced images are expected to be present under the release package with a path following the pattern <path_to_release_package>/target/images/ directory but may also be organized into subfolders. For example:

- target/images/
- target/images/atf/
- target/images/u-boot/dtbs/

The following section details which files are expected in each location.

7.3.1.1 JSON Structure (Schema)

The file is a JSON object whose keys are board identifiers (e.g., rzg2l-evk, rzg2l-sbc), and whose values are objects with the following fields:

Table 11: flash_images.json schema

Key	Description	Allowed / Example Values
soc	SoC/MPU family identifier	g2l, v2l, v2h
bl2	BL2 (Boot Loader stage 2) image. This BL2 must contain the FCONF device tree.	bl2_bp_<board-name>.srec
board_identification	Board-info binary from binmake tool (see Section 15.1. Board identification JSON Specification). Not the JSON itself.	<board-name>-platform-settings.bin
fip	FIP (TF-A package) image, contains BL31 and u-boot-nodtb.bin with its dtb.	fip_<board-name>.srec
atf_fdts	FCONF DTB for BL2, compiled from DTS (see Section 15.3. TF-A (BL2/BL31)).	<board-name>.dtb
uboot_dtb	U-Boot device tree blob	<board-name>.dtb
flash_writer	Flash Writer binary for low-level programming	Flash_Writer_SCIF_<board-name>.mot
Ipl_flash_method	IPL media used for flashing	xspi, emmc
rootfs	Root filesystem image	core-image-minimal.wic
rootfs_flash_method	Method to flash rootfs image.	udp, otg

The table below shows, for each supported board, which IPL flash method the IPL uses to write boot components (xspi or emmc) and which rootfs flash method is used to write the .wic image (udp or otg) as defined in flash_images.json is currently supported

Table 12: Supported IPL flash and rootfs method

Board	SoC/MPU	ipl_flash_method	Default	rootfs_flash_method	Default
rzg2l-sbc	g2l	xspi	xspi	udp	udp
rzg2l-evk	g2l	xspi, emmc	xspi	udp	udp
rzv2l-evk	v2l	xspi, emmc	xspi	udp	udp
rzv2h-evk	v2h	xspi	xspi	udp	udp

Note:

- IPL flash method: emmc for rzv2h-evk is not supported yet.
- IPL flash method: eSD for all boards is not supported yet.
- Rootfs flash method: OTG for all boards is not supported yet.
- Rootfs flashing via UDP on RZ/G2L-EVK and RZ/V2L-EVK is unreliable and not recommended. Use balenaEtcher or the dd command to write the image instead.

7.3.1.2 Adding a New Board

To add a custom board, create a new top-level entry following the schema described in Section 7.3.1.1 (JSON Structure)

- Provide the correct soc value.
- Generate the board-info binary via binmake and set board_identification to that binary filename (not the JSON source).
- Create/compile the FCONF DTB for BL2 and set atf_fdts accordingly (see Section 15.3).
- Provide the board's U-Boot DTB via uboot_dtb.
- Choose valid methods for ipl_flash_method (xspi/emmc) and rootfs_flash_method (udp/otg).
- Verify bl2, fip, flash_writer, and rootfs filenames match the delivered artifacts.

Example: A new board entry for RZ/G2L-CUSTOM may be defined as follows:

```
{
  "rzg2l-custom": {
    "soc": "g2l",
    "bl2": "bl2_bp_rzg2l-custom.srec",
    "board_identification": "rzg2l-custom-platform-settings.bin",
    "fip": "fip_rzg2l-custom.srec",
    "atf_fdts": "rzg2l-custom.dtb",
    "uboot_dtb": "rzg2l-custom.dtb",
    "flash_writer": "Flash_Writer_SCIF_rzg2l-custom.mot",
    "ipl_flash_method": "xspi",
    "rootfs": "core-image-minimal.wic",
    "rootfs_flash_method": "udp"
  }
}
```

7.3.2 Usage and Flashing Operations

The universal_script.py serves as the main entry point for performing all flashing operations. It uses the board configuration defined in flash_images.json to select the appropriate images and procedures.

All required images (bl2, bl31, u-boot-nodtb, DTBs, rootfs) must be placed under:

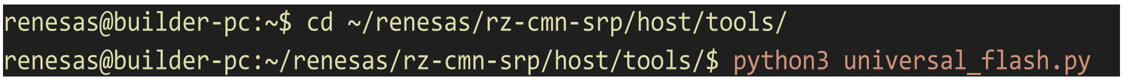
Table 13: Required Image Locations and Resolution Priority

Component	Expected Location	Purpose	Override (from flash_images.json)
BL2/BL31	• </path/to/yocto/package>/target/images/atf/bl2-rz-cmn.bin • </path/to/yocto/package>/target/images/atf/bl31-rz-cmn.bin	Bootloader stages are responsible for early initialization (BL2) and runtime services (BL31).	Not overridable – always taken from the default location.
BL2 FCONF Device Trees	</path/to/yocto/package>/target/images/atf/fdts/	Configuration device trees for BL2 (DDR, PFC, CPG, etc.).	**atf_fdts** can override with a per-board FCONF DTB
U-Boot (no DTB)	</path/to/yocto/package>/target/images/u-boot/u-boot-nodtb-rz-cmn.bin	Per-board U-Boot device trees. Selected automatically by the flashing script.	Not overridable – always taken from the default location.
U-Boot DTBs	</path/to/yocto/package>/target/images/u-boot/dtbs/	Per-board U-Boot device trees. Selected automatically by the flashing script.	**uboot_dtb** can override selects the exact DTB for the board.
Root filesystem	</path/to/yocto/package>/target/images/(e.g., core-image-weston.wic	Complete root file system image written to SD card or eMMC.	**rootfs** and **rootfs_flash_method** override the image and flashing method (e.g., udp, otg).
Flash writer	</path/to/yocto/package>/target/images/Flash-writer-<board>.mot	Initial loader for XSPI/eMMC flashing.	**flash_writer** must define the correct image for the board
Board identification	- </path/to/yocto/package>/target/images/<board>-platform-settings.srec - </path/to/yocto/package>/target/images/<board>-platform-settings.bin	Per-board platform/ID blob used during flashing.	**board_identification** can override selects the file.

After all, required images are placed in their expected locations (and flash_images.json is configured if needed), proceed to run the script.

In general, the release images already include all necessary files. Only when replacing a specific file or adding a new board is it necessary to follow the expected directory layout above and update flash_images.json accordingly.

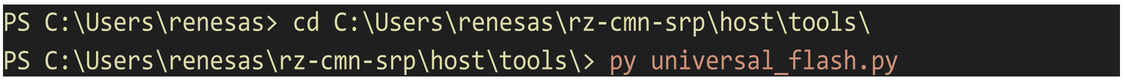
- **On Linux:** Open a terminal and run the following commands



```
renesas@builder-pc:~$ cd ~/renesas/rz-cmn-srp/host/tools/
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/$ python3 universal_flash.py
```

- **On Windows:** Open one of the following terminals with Administrator privileges:
  - PowerShell
  - Git Bash
  - MobaXterma

  Navigate to the host/tools/ directory inside the rz-cmn-srp folder. The example below uses PowerShell, but the same applies to other terminals

```
PS C:\Users\renesas> cd C:\Users\renesas\rz-cmn-srp\host\tools\
PS C:\Users\renesas\rz-cmn-srp\host\tools\> py universal_flash.py
```

Upon execution, the script will present an interactive menu to choose the desired flashing operation.

7.3.2.1 Workflow Diagram

The flow below reflects the interactive prompts in universal_script.py: board selection, serial port and baud rate selection, then choices to flash the root filesystem and/or the IPL.

To run the universal flashing proces, follow these steps:

1. Select the target board and press Enter.
2. Select the serial port and press Enter.
3. Select the baud rate – usually the default (115200). Press Enter to accept it.
4. When asked “Do you want to write the rootfs? (y/n)”, type n and press Enter.
   - y → to flash the root filesystem
   - n → to skip rootfs flashing
5. When asked “Write IPL method:”, choose:
   - 1 ‐ BootloaderFlash → standard bootloader flashing (recommended for Quick Start)
   - 2 ‐ UloadFlash → requires prerequisites (see Sections 7.3.4.1–7.3.4.2). Use this option only if the per-board BL2 and FIP images have been built and copied to the SD card.

   Confirm the choice and press Enter.

6. Reset the board to begin flashing:
   - If the board has a reset button, press it.
   - If it does not, power cycle the board.

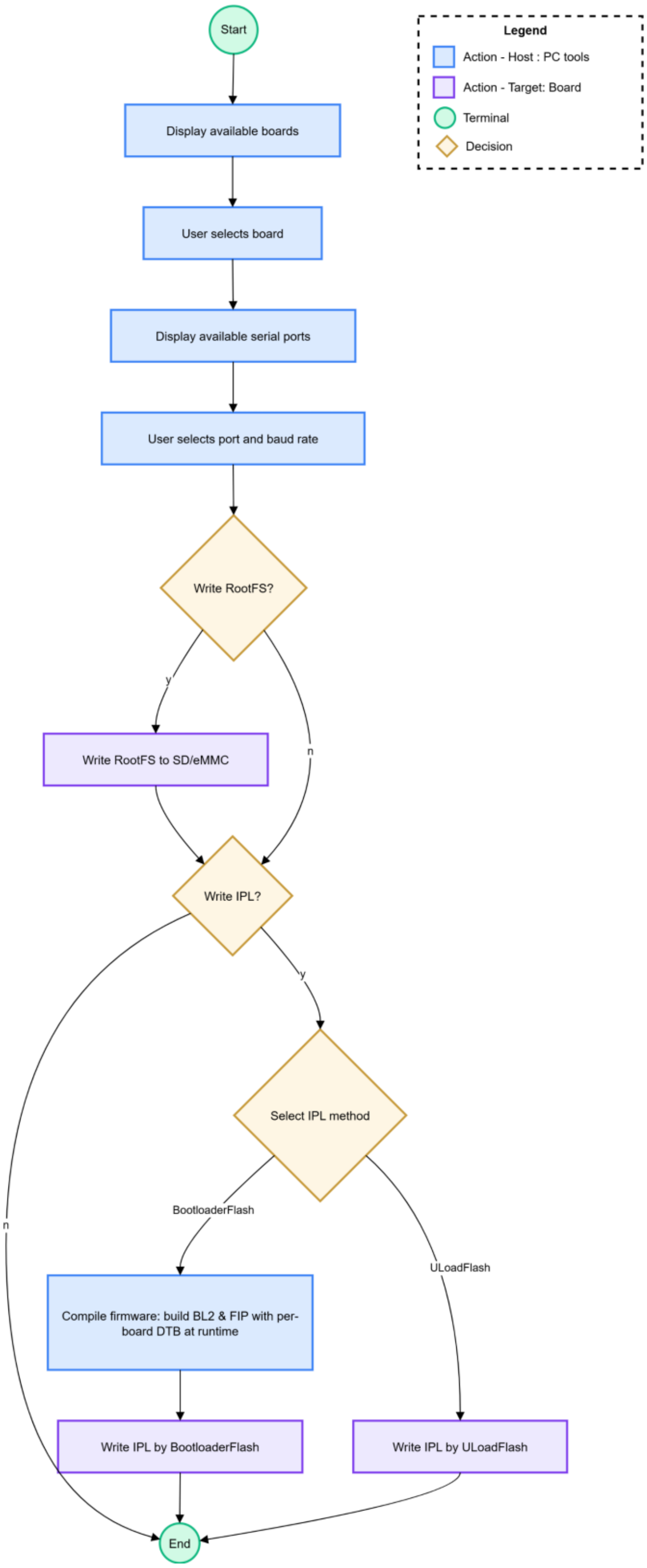


Figure 17: Universal Flashing Script — Interactive Workflow

7.3.3 Flashing the Bootloader

This section describes the dedicated bootloader flashing script, intended for writing the initial bootloader image to the board over a serial interface. It is suitable for first-time provisioning, factory recovery, or focused solely on IPL/bootloader updates.

The universal flashing script (see Section 7.3.2. Usage and Flashing Operations) is the master, multi-operation tool that handles rootfs, and IPL flashing. The bootloader-only script is a streamlined utility that performs bootloader flashing only, with minimal prompts and faster turnaround.

Location:

```
host/tools/bootloader_flasher/bootloader_flash.py
```

7.3.3.1 Hardware Connection

Before starting the flashing process, ensure that the required hardware interfaces are properly connected between the board and the host PC.

- Connect the debug serial port of the board to the host PC.
- Place the board into **SCIF (serial) download mode before flashing**. The exact DIP-switch or jumper settings are **board-specific**.
  - Refer to Section 16.1. Factory Firmware Flashing Using Serial Downloader (SCIF) Mode for the required switch positions and steps for the target board.

7.3.3.2 Flashing Procedure

To begin using the bootloader_flash.py, execute it from the host machine:

- **On Linux:** Open a terminal and run the following commands

```
renesas@builder-pc:~$ cd ~/renesas/rz-cmn-srp/host/tools/bootloader_flasher
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/bootloader_flasher$ python3
bootloader_flash.py
```

- **On Windows:** Open one of the following terminals with Administrator privileges:
  - PowerShell
  - Git Bash
  - MobaXterma

  Navigate to the host/tools/bootloader_flasher directory inside the rz-cmn-srp folder. The example below uses PowerShell, but the same applies to other terminals

```
PS C:\Users\renesas> cd C:\Users\renesas\rz-cmn-srp\host\tools\bootloader_flasher
PS C:\Users\renesas\rz-cmn-srp\host\tools\bootloader_flasher> py
bootloader_flash.py
```

If no arguments are provided, the script will use the following default values:

- Board name: rzg2l-sbc
- Flash method: xspi
- Serial port: most recently connected port (E.g: COM8 in Windows or /dev/ttyUSB0 in Linux)
- Serial port baud: 115200
- Flash Writer Image: /path/to/universal-scripts/target/images/Flash_Writer_SCIF_rzg2l-sbc.mot
- BL2 Image: /path/to/universal-scripts/target/images/bl2_bp_rzg2l-sbc.srec
- FIP Image: /path/to/universal-scripts/target/images/fip_rzg2l-sbc.srec

- Board identification Image: /path/to/universal-scripts/target/images/rzg2l-sbc-platform-settings.bin

Ensure that these files are present in the current directory before executing the script.

To specify custom file paths or override the defaults, the following arguments can be passed:

- **--board_name:** Board name to flash bootloader.
- **--flash_method:** Flash method to use (xspi or emmc).
- **--serial_port:** Serial port to use for communication with the board.
- **--serial_port_baud:** Baud rate for the serial port.
- **--image_writer:** Path to the Flash Writer image.
- **--image_bl2:** Path to the BL2 image.
- **--image_fip:** Path to the FIP image.
- **--image_bid:** Path to the board identification image.

Example command:

- **On Linux:**

```
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/bootloader_flasher$ py
bootloader_flash.py --board_name rzg2l-evk --flash_method emmc --serial_port
COM11 --serial_port_baud 9600 --image_writer
D:\custom_images\Flash_Writer_SCIF_rzg2l-sbc.mot --image_bl2
D:\custom_images\bl2_bp_rzg2l-sbc.srec --image_fip D:\custom_images\fip-rzg2l-
sbc.srec --image_bid D:\custom_images\rzg2l-evk-platform-settings.bin
```

- **On Windows:**

```
PS C:\Users\renesas\rz-cmn-srp\host\tools\bootloader_flasher> python3
bootloader_flash.py --board_name rzg2l-evk --flash_method emmc --serial_port
/dev/ttyUSB0 --serial_port_baud 9600 --image_writer
/home/renesas/custom_images/Flash_Writer_SCIF_rzg2l-sbc.mot --image_bl2
/home/renesas/custom_images/bl2_bp_rzg2l-sbc.srec --image_fip
/home/renesas/custom_images/fip-rzg2l-sbc.srec --image_bid
/home/renesas/custom_images/rzg2l-evk-platform-settings.bin
```

7.3.4 Flashing the uLoad-bootloader

This section describes the ULoad-bootloader flow for programming the bootloader from the U-Boot console. It supports QSPI/xSPI flashing and is intended for cases where the device can boot to U-Boot and program flash using images stored on the removable media.

Location:

```
host/tools/uload_bootloader/uload_bootloader_flash.py
```

7.3.4.1 Prerequisites

The release does not include prebuilt ULoad images on the SD card. The ULoad flow requires BL2 and FIP artifacts that are rebuilt for the selected board with the correct DTB/FCONF and configuration.

Run the firmware_compile.py script to generate these artifacts, then copy them to partition 1 (FAT32) under /uload-bootloader/ before running the ULoad flasher. This ensures the programmed bootloader matches the exact board and release configuration, minimizing risk of mismatch.

To begin with compiling uLoad images, execute it from the host machine:

- **On Linux:** Open a terminal and run the following commands

```
renesas@builder-pc:~$ cd ~/renesas/rz-cmn-srp/host/tools/firmware_compile
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/firmware_compile$ python3
firmware_compile.py
```

- **On Windows:** Open one of the following terminals with Administrator privileges:
  - PowerShell
  - Git Bash
  - MobaXterma

  Navigate to the host/tools/firmware_compile directory inside the rz-cmn-srp folder. The example below uses PowerShell, but the same applies to other terminals

```
PS C:\Users\renesas> cd C:\Users\renesas\rz-cmn-srp\host\tools\ firmware_compile
PS C:\Users\renesas\rz-cmn-srp\host\tools\firmware_compile> py
firmware_compile.py
```

If no CLI options are supplied, the script uses the following defaults:

Table 14: Firmware compiled CLI options

Option	Default	Description
--board	rzg2l-sbc	Target board name (must exist in boards_flash_config.toml and flash_images.json).
--soc	g2l	Target SoC family (g2l, v2l, v2h).
--method	xspi	Flash method (xspi or emmc).
--bl2	auto from images	Path to BL2 binary (override default).
--atf-fdts	auto from JSON	TF-A FDT(s) to append to BL2.
--uboot-dtbs	auto from JSON	U-Boot DTB(s) to append to U-Boot nodtb.
--bl31	auto from images	Path to BL31 binary (override default).
--u-boot-nodtb	auto from images	Path to U-Boot (nodtb) binary (override default).
--bootparameter	auto search	Path to bpgen tool (override search path).
--fiptool	auto search	Path to fiptool tool (override search path).
--objcopy	auto search	Path to objcopy tool (override search path).
--fip-align	16	FIP alignment.
--fip-vma	from TOML	Override virtual memory address (VMA) for FIP .srec.
--bl2-bp-vma	from TOML	Override VMA for BL2+BP .srec.

Notes:

- auto from images: uses the **common artifacts** produced in target/images/… (not per-board overrides).
- auto from JSON: resolves per-board filenames/DTBs via flash_images.json.
- from TOML: uses addresses/layout defined in boards_flash_config.toml for the selected --board and --method

7.3.4.2 Collect the Output and Prepare Binaries in SD Card

This step packages the artifacts built by firmware-compile.py and places them on the removable media so the ULoad-bootloader script (U-Boot console flow) can program QSPI/xSPI.

From <path/to/yocto/package/target/images/>, gather the per-board files:

- bl2: bl2_bp_<board>.bin.

- fip_<board>.bin
- <board>-<version>-platform-settings.bin

Place all files on partition 1 (FAT32) of the SD card under this directory.

```
/uload-bootloader
```

7.3.4.3 Hardware Connection

Before starting the flashing process, ensure that the required hardware interfaces are properly connected between the board and the host PC.

- Connect the debug serial port of the board to the host PC.
- Place the board into normal boot mode. The exact DIP-switch or jumper settings are board-specific.
  - Refer to Section 16.1. Factory Firmware Flashing Using Serial Downloader (SCIF) Mode for the required switch positions and steps for the target board.

7.3.4.4 Flashing Procedure

To begin using the uload_bootloader.py, execute it from the host machine:

- **On Linux:** Open a terminal and run the following commands

```
renesas@builder-pc:~$ cd ~/renesas/rz-cmn-srp/host/tools/uload_bootloader
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/uload_bootloader$ python3
uload_bootloader_flash.py
```

- **On Windows:** Open one of the following terminals with Administrator privileges:
  - PowerShell
  - Git Bash
  - MobaXterma

  Navigate to the host/tools/uload_bootloader directory inside the rz-cmn-srp folder. The example below uses PowerShell, but the same applies to other terminals

```
PS C:\Users\renesas> cd C:\Users\renesas\rz-cmn-srp\host\tools\uload_bootloader
PS C:\Users\renesas\rz-cmn-srp\host\tools\uload_bootloader> py
uload_bootloader_flash.py
```

If no arguments are provided, the script will use the following default values for RZ/G2L-SBC board.

- bl2 binary file: bl2_bp_<board-name>.bin
- fip file: fip_<board-name>.bin
- board identication file: <board-name>-platform-settings.bin

Ensure that these files are present in the current directory before executing the script.

To specify custom file paths or override the defaults, the following arguments can be passed:

- **--serial_port:** Serial port to use for communication with the board.
- **--serial_port_baud:** Baud rate for the serial port.
- **--image_bl2:** Path or filename of the BL2 image
- **--image_fip:** Path or filename of the FIP image
- **--image_bid:** Path or filename of the board-identification file

Example command:

- **On Linux:**

```
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/uload_bootloader$ python3
uload_bootloader_flash.py \
  --serial_port /dev/ttyUSB0 \
  --serial_port_baud 115200 \
  --image_bl2 /mnt/sd/uload-bootloader/bl2_bp-rzg2l-sbc.bin \
  --image_fip /mnt/sd/uload-bootloader/fip-rzg2l-sbc.bin \
  --image_bid /mnt/sd/uload-bootloader/rzg2l-sbc-platform-settings.bin
```

- **On Windows:**

```
PS C:\Users\renesas\rz-cmn-srp\host\tools\uload_bootloader> py
uload_bootloader_flash.py \
  --serial_port /dev/ttyUSB0 \
  --serial_port_baud 115200 \
  --image_bl2 /mnt/sd/uload-bootloader/bl2_bp-rzg2l-sbc.bin \
  --image_fip /mnt/sd/uload-bootloader/fip-rzg2l-sbc.bin \
  --image_bid /mnt/sd/uload-bootloader/rzg2l-sbc-platform-settings.bin
```

Note:

- If only a filename is provided (no path), the script searches the default directory (e.g., /uload-bootloader on partition 1, FAT32).
- Ensure the filenames match the board that was built with firmware-compile.py.

7.3.5 Flasing the SD Card Image

This section explains how to use the dedicated SD flashing script, sd_creator.py, which helps create bootable SD cards using a fastboot-like approach.

```
host/tools/sd_creator/
```

Hierarchy:

```
sd_creator
├── README.md
├── sd_flash.py
└── tools
    ├── AdbWinApi.dll
    └── fastboot.exe
```

7.3.5.1 Hardware Connection

Before starting the flashing process, ensure that the required hardware interfaces are properly connected between the board and the host PC. The type of connection depends on the flashing method selected.

- Connect the debug serial port of the board to the host PC for console access and monitoring.
- Connect the appropriate interface based on the fastboot method:
  - [UDP] Connect the Ethernet port of the board to the host PC.
  - [OTG] Connect the USB OTG port of the board to the host PC.

7.3.5.2 Flashing Procedure

To begin using the sd_creator.py, execute it from the host machine:

- **On Linux:** Open a terminal and run the following commands

```
renesas@builder-pc:~$ cd ~/renesas/rz-cmn-srp/host/tools/sd_creator
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/sd_creator$ python3
sd_flash.py
```

- **On Windows:** Open one of the following terminals with Administrator privileges:
  - PowerShell
  - Git Bash
  - MobaXterma

  Navigate to the host/tools/sd_creator directory inside the rz-cmn-srp folder. The example below uses PowerShell, but the same applies to other terminals

```
PS C:\Users\renesas> cd C:\Users\renesas\rz-cmn-srp\host\tools\sd_creator
PS C:\Users\renesas\rz-cmn-srp\host\tools\sd_creator> py sd_flash.py
```

If no arguments are provided, the script will use the following default values:

- Fastboot type: udp
- IP address: 169.254.187.89
- Serial port: most recently connected port (e.g: COM8 in Windows or /dev/ttyUSB0 in Linux)
- Serial port baud: 115200
- WIC file: </path/to/your/package>/target/images/core-image-minimal.wic

Ensure that these files are present in the current directory before executing the script.

To specify custom file paths or override the defaults, the following arguments can be passed:

- **--board_name**: Board name to flash bootloader. Default is rzg2l-sbc.
- **--fastboot_type**: Fastboot type to use (udp or otg). Default is udp.
- **--ether_port**: [Only used in fastboot UDP] Ethernet port used to board communication. Defaults to 1.
- **--ip_address**: [Only used in fastboot UDP] Ethernet IP address used to board communication. Defaults to 169.254.187.89.
- **--serial_port**: Serial port to use for communication with the board. Default is most recently connected port (E.g: COM8 in Windows or /dev/ttyUSB0 in Linux).
- **--serial_port_baud**: Baud rate for the serial port. Default is 115200.
- **--image_rootfs**: Path to the root filesystem image.

Example command:

- **On Linux:**

```
renesas@builder-pc:~/renesas/rz-cmn-srp/host/tools/sd_creator$ python3
sd_flash.py --board_name rzg2l-evk --fastboot_type udp --ip_address 169.254.187.9
--ether_port 1 --serial_port /dev/ttyUSB0 --serial_port_baud 9600 --image_rootfs
/home/renesas/custom_images/core-image-weston.wic
```

- **On Windows:**

```
PS C:\Users\renesas\rz-cmn-srp\host\tools\sd_creator> py sd_flash.py --board_name
rzg2l-evk --fastboot_type udp --ip_address 169.254.187.9 --ether_port 1 --
serial_port COM11 --serial_port_baud 9600 --image_rootfs D:\custom_images\core-
image-weston.wic
```

8. Accessing Supported Features

This section explores the key features and interfaces available across both Yocto and Ubuntu images on all supported platforms in this release.

8.1 Supported Features in Yocto Images

8.1.1 Common Yocto Features

8.1.1.1 Generic USB Bluetooth Framework

The RZ boards support the generic USB Bluetooth framework, which is back-ported from the Linux kernel mainline. TP-Link UB500 Bluetooth 5.0 Nano USB Adapter (Realtek chipset) has been tested and proven to work on the board.

(1) Establishing a Bluetooth Connection

Note: Ensure you have internet access before running the commands. If the firmware is downloaded for the first time, a reboot of the board is required to ensure the TP-Link UB500 adapter functions properly.

The following steps will guide you on how to enable the TP-Link UB500 adapter:

1. Download the appropriate firmware for the TP-Link UB500 adapter and store it on the target board. This will ensure it is loaded each time the board boots (one-time setup).

```
root@rz-cmn:~# mkdir -p /lib/firmware/rtl_bt
root@rz-cmn:~# curl -s https://raw.githubusercontent.com/Realtek-
OpenSource/android_hardware_realtek/rtk1395/bt/rtkbt/Firmware/BT/rtl8761b_fw -o
/lib/firmware/rtl_bt/rtl8761bu_fw.bin
```

2. By default, Bluetooth is blocked by RFKILL. To unblock it, use the command 'rfkill unblock bluetooth'

```
root@rz-cmn:~# rfkill list
0: hci0: Bluetooth
        Soft blocked: yes
        Hard blocked: no
root@rzg2l-sbc:~# rfkill unblock bluetooth
root@rzg2l-sbc:~# rfkill list
0: hci0: Bluetooth
        Soft blocked: no
        Hard blocked: no
```

3. Verify whether the TP-Link UB500 adapter is properly attached and is running.

Run the following command to ensure that the system has recognized the TP-Link UB500 adapter:

```
rz-cmn:~# hciconfig -a

hci0:   Type: Primary  Bus: USB
        BD Address: E8:48:B8:C8:20:00  ACL MTU: 1021:6  SCO MTU: 255:12
        UP RUNNING
        RX bytes:1773 acl:0 sco:0 events:142 errors:0
        TX bytes:13029 acl:0 sco:0 commands:142 errors:0
        Features: 0xff 0xff 0xff 0xfe 0xdb 0xfd 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH HOLD SNIFF PARK
        Link mode: PERIPHERAL ACCEPT
        Name: rz-cmn
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Version: 5.1 (0xa)  Revision: 0x97b
        LMP Version: 5.1 (0xa)  Subversion: 0xec43
        Manufacturer: Realtek Semiconductor Corporation (93)
```

The TP-Link UB500 adapter is now ready to connect.

3. Connect the Bluetooth device.

Use bluetoothctl to connect to a Bluetooth device:

```
rz-cmn:~# bluetoothctl

[bluetooth]# power on
[bluetooth]# pairable on
[bluetooth]# agent on
[bluetooth]# default-agent
```

Set the target board to be discoverable by other Bluetooth devices:

```
[bluetooth]# discoverable on
```

Enable and disable scan function:

```
[bluetooth]# scan on
[bluetooth]# scan off
```

Pair and connect the device:

```
[bluetooth]# pair FC:02:96:A5:80:97
[bluetooth]# trust FC:02:96:A5:80:97
[bluetooth]# connect FC:02:96:A5:80:97
```

‘FC:02:96:A5:80:97’ is the address of the Bluetooth device. Change it to match your device’s address.

Exit bluetoothctl.

```
[bluetooth]# exit
```

(2) Transferring Files over Bluetooth

To share files between the RZ board and another Bluetooth-enabled device (e.g., a laptop or smartphone), start the obexctl daemon on the RZ board and initiate a connection:

```
root@rz-cmn:~# export $(dbus-launch)

root@rz-cmn:~# /usr/libexec/bluetooth/obexd -r /home/root -a -d & obexctl
[1] 595
[NEW] Client /org/bluez/obex
[obex]#
[obex]# connect FC:02:96:A5:80:97
Attempting to connect to FC:02:96:A5:80:97
[NEW] Session /org/bluez/obex/client/session0 [default]
[NEW] ObjectPush /org/bluez/obex/client/session0
Connection successful
```

‘FC:02:96:A5:80:97’ is the address of the Bluetooth device. Change it to match your device’s address.

Then, to send files, use the ‘send’ command while connected to the OBEX Object Push profile.

```
[FC:02:96:A5:80:97]# send /boot/uEnv.txt
Attempting to send /boot/uEnv.txt to /org/bluez/obex/client/session0
[NEW] Transfer /org/bluez/obex/client/session0/transfer0
Transfer /org/bluez/obex/client/session0/transfer0
        Status: queued
        Name: uEnv.txt
        Size: 2069
        Filename: /boot/uEnv.txt
        Session: /org/bluez/obex/client/session0
[CHG] Transfer /org/bluez/obex/client/session0/transfer0 Status: complete
[DEL] Transfer /org/bluez/obex/client/session0/transfer0
[FC:02:96:A5:80:97]# quit
```

8.1.1.2 Onboard Audio Codec with Stereo Jack

Each RZ board features an onboard audio codec and may include a dedicated video codec chip, depending on the model. Audio playback and recording are supported via the 3.5mm stereo jack (connector J8, 6-pin).

- Audio Data Interface: Connected to DAI (SSI1) using the I^{2}S format.
- Control Interface: Managed via I^{2}C0.
- Headset Jack: Marked J8 on the board.

Step 1: Discover available audio interfaces

Before playback or recording, list all ALSA devices and their properties:

```
root@rz-cmn:~# aplay -l      # List available playback devices
root@rz-cmn:~# arecord -l    # List available recording devices
root@rz-cmn:~# aplay -L      # List all supported PCM devices and formats
```

This step ensures that the onboard codec is recognized and identifies the correct device index (e.g., hw:0,0).

Step 2: Prepare Audio files

Prepare the required audio files and copy them into the target filesystem (e.g., /home/root/audio/).

- The aplay tool supports only WAV (.wav) format.
- For additional formats such as MP3 and AAC, use the pre-installed GStreamer framework, which provides compatibility with multiple codecs.

Step 3: Playback

Examples:

- WAV playback (ALSA/PCM):

```
root@rz-cmn:~# aplay -D hw:0,0 /home/root/audios/test.wav
```

  - -D specifies the ALSA device to use.
  - hw:0,0 means card 0, device 0, which corresponds to the onboard audio codec (as shown in the aplay -l output).
  - If the board reports a different index, replace hw:0,0 with the correct value (e.g., hw:1,0).
- WAV, MP3, AAC playback (Gstreamer):

```
root@rz-cmn:~# gst-play-1.0 /home/root/audios/test.wav
root@rz-cmn:~# gst-play-1.0 /home/root/audios/test.mp3
root@rz-cmn:~# gst-play-1.0 /home/root/audios/test.aac
```

Step 4: Recording

To capture audio through the onboard codec:

```
root@rz-cmn:~# arecord -f S16_LE -r 48000 audio_capture.wav
```

In the above command:

- -f S16_LE : audio format (signed 16 bit little endian)
- -r 48000 : sample rate of the audio file (48KHz)

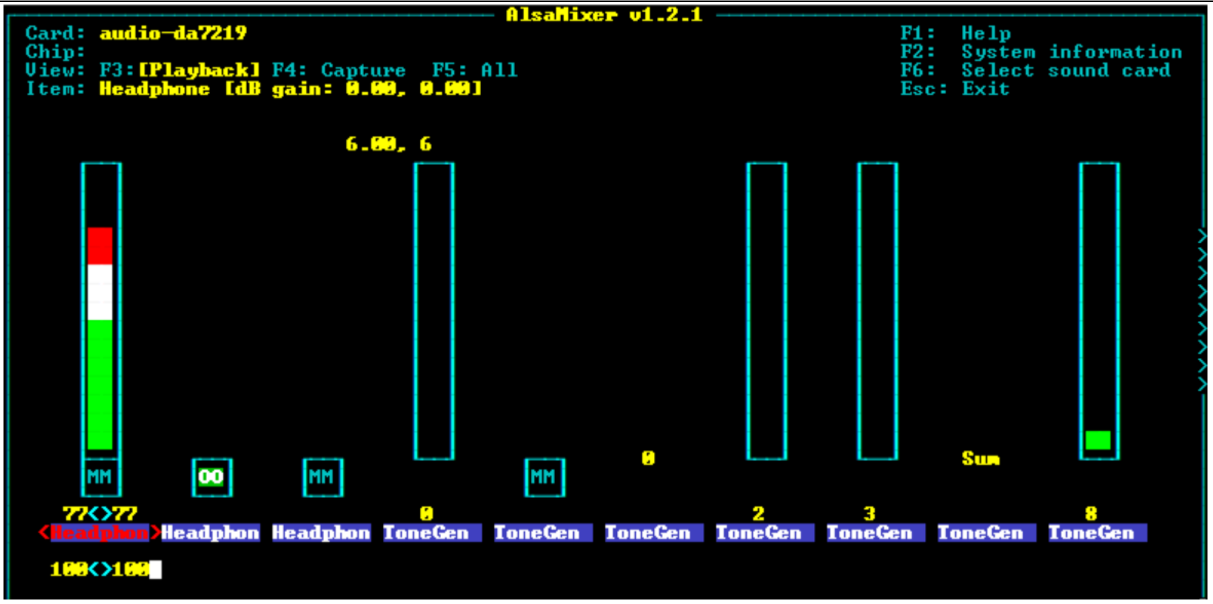
Press Ctrl+C to stop recording.

Verify the recording

```
root@rz-cmn:~# aplay audio_capture.wav
```

To adjust the level of the audio record/playback, use the following command to open the ALSA mixer GUI on the debug console:

```
root@rz-cmn:~# alsamixer
```



```
AlsaMixer v1.2.1
Card: audio-da7219                                                   F1:  Help
Chip:                                                                F2:  System information
View: F3:[Playback] F4: Capture  F5: All                             F6:  Select sound card
Item: Headphone [dB gain: 0.00, 0.00]                                Esc: Exit
```

Figure 18. Example ALSA Mixer GUI displayed on the RZ/G2L-SBC debug console

8.1.1.3 Quickboot Images and Network Configurations

Renesas provides custom Quickboot images optimized for faster boot times. These images include necessary systemd optimizations and a streamlined kernel to minimize boot delays.

By default, systemd services for networking, D-Bus, and other non-essential components are disabled, leaving only the core boot services active.

The details of these images are provided in section 1.1 Supported Distributions:

Table 15. Custom Quickboot images

Images	Description
renesas-quickboot-cli	A minimal Linux image with Quickboot enabled, offering only a CLI without a desktop environment. It supports HDMI/DSI output but lacks graphical components and a desktop, which makes it ideal for fast-booting command-line-based systems.
renesas-quickboot-wayland	A Quickboot-enabled Linux image with Wayland and Qt support, featuring the Weston graphical desktop environment. It provides a basic graphical desktop, allowing users to develop and integrate custom GUI applications. No desktop applications are included by

	default. However, the QT framework is available, allowing the user to install and run any QT application.

(1) Enable Networking Stack

For both Quickboot CLI and Quickboot Wayland images, networking (including Wi-Fi, Bluetooth, and SSH services) is disabled by default and must be enabled manually. The required scripts are in /home/root/network-management/.

To see available options before enabling any services, run the help command:

```
root@rz-cmn:~# cd network-management
root@rz-cmn:~/network-management# ./enable_networking_stack.sh help
```

This command displays the usage information along with the following options:

- wifi: Enable Wi-Fi services.
- bluetooth: Enable Bluetooth services.
- sshd: Enable SSH/SCP services.
- all: Enable all network-related services (wifi, bluetooth, sshd).

Run the following command with the appropriate option:

```
root@rz-cmn:~/network-management# ./enable_networking_stack.sh <service>
```

For example, to enable Wi-Fi, run:

```
root@rz-cmn:~/network-management# ./enable_networking_stack.sh wifi
```

To enable all networking services:

```
root@rz-cmn:~/network-management# ./enable_networking_stack.sh all
```

Note: Reboot the board for the changes to take effect or manually restart each service and its dependencies.

(2) Disable Networking Stack

To restore the default Quickboot behavior and disable unused network services, use the provided script. This removes systemd service symlinks and masks services related to networking, Wi-Fi, Bluetooth, and SSH.

Run the following command with the appropriate option to disable unused services:

```
root@rz-cmn:~/network-management# ./disable_networking_stack.sh <service>
```

For example, to disable Bluetooth:

```
root@rz-cmn:~/network-management# ./disable_networking_stack.sh bluetooth
```

To fully restore Quickboot’s default behavior by disabling all networking services:

```
root@rz-cmn:~/network-management# ./disable_networking_stack.sh all
```

Note: Reboot the board for the changes to take effect or manually restart each service and its dependencies.

(3) Kernel Optimization

By default, the release package does not optimize the kernel. This is purposefully done to allow kernel debugging and have more verbose logs.

If an optimized kernel is required, it becomes necessary to rebuild a kernel through the SDK or yocto. The optimization setting is configured in the local.conf file within the Yocto build environment (typically located under build/conf/local.conf.)

Set the variable OPTIMIZE_KERN in local.conf to enable kernel optimization. This configuration disables unused features and converts certain built-in modules (USB, touchscreen, CANFD, etc.) into loadable modules. The result is a smaller kernel, faster boot time, and improved resource utilization.

To optimize the kernel, follow these steps to modify the local.conf:

1. Open the local.conf file in Yocto build configuration.
2. Set the ‘OPTIMIZE_KERN’ from “0” to “1”.

   ```
   # Optimized Linux Kernel Support: Build with optimizations for the Linux kernel
   # Default: 0 - Disable
   # Set to: 1 - Enable
   OPTIMIZE_KERN = "1"
   ```

   This will ensure that unnecessary kernel features are disabled, and certain modules are built as loadable, leading to a more efficient system.
3. Rebuild and deploy the image to apply the changes.

8.1.1.4 Playing Video Files on RZ board

Use gst-launch-1.0 to play video files. The playbin element in GStreamer makes it easy to play multimedia content. Prepare an mp4 file and run the following command:

```
root@rz-cmn:~# gst-launch-1.0 playbin uri=file:///<path/to/your/video/path>
```

For example,

```
root@rz-cmn:~# gst-launch-1.0 playbin uri=file:///home/root/videos/h264-hd-30.mp4
```

This will start an MP4 video and display it on the screen.

Figure 21. Playing an MP4 video on the RZ/G2L-SBC

8.1.1.5 Package Management

The distribution comes with the Debian package manager ‘apt-get’ and ‘dpkg’ for binary package handling.

Follow the steps below to modify the Debian package repository and install packages according to your needs.

1. Add/modify sources.list file to address the packages repository:
   The ‘sources.list’ is a critical configuration file for package installation and updates used by package managers on Debian-based Linux distributions. The ‘sources.list’ file contains a list of URLs for repository addresses where the package manager can find software packages. These repositories may be maintained by the Linux distribution itself or by third-party individuals or organizations.

   Currently, the default `sources.list`, which is located in /etc/apt/sources.list.d/sources.list/ directory is as below.

```
deb [arch=arm64] http://ports.ubuntu.com/ oracular main multiverse universe
deb [arch=arm64] http://ports.ubuntu.com/ oracular-security main multiverse universe
deb [arch=arm64] http://ports.ubuntu.com/ oracular-backports main multiverse universe
deb [arch=arm64] http://ports.ubuntu.com/ oracular-updates main multiverse universe
```

2. Update the defined package index for apt-get.

```
root@rz-cmn:~# apt-get update
```

   Ensure you have internet access before running apt-get update.

   In the contents of sources.list file, each line has [arch=arm64]. This is because the RZ/G2L SoC is an ARM 64 (aarch64) core. This can be verified by the lscpu command:

```
root@rz-cmn:~# lscpu
Architecture:                   aarch64
CPU op-mode(s):                 32-bit, 64-bit
Byte Order:                     Little Endian
CPU(s):                         2
...
Vendor ID:                      ARM
```

   By specifying [arch=arm64] in sources.list file, apt-get will filter for the proper binary packages in the repository. This will limit the existing APT sources to arm64 only. However, if we use a repository which is entirely hosting ARM 64-bit (aarch64) packages, we do not need to specify [arch=arm64] in the sources.list entry. For example:

```
deb http://deb.debian.org/debian trixie main contrib non-free
```

   Remember that sources do not have to be a single origin. It is very common to add multiple repositories and sources for packages and manage them using keys.

   The source management is beyond the scope of this document.

3. Installing packages using apt-get:
   To install a package using apt-get, use the following command:

```
root@rz-cmn:~# apt-get install <package-name>
```

> Note: The release currently uses Ubuntu Oracular as the default APT repository source. Modifying the APT sources (e.g., switching to Debian or using third-party repositories) may break the boot or cause installation issues for some applications due to changes in package versions, availability, or dependencies. Proceed with caution if you plan to alter the default APT configuration.

(1) Docker Installation Setup

This guide walks you through enabling Docker support at the kernel level, installing Docker, configuring firewall compatibility, and verifying the installation.

Step 1: Enable Docker support in kernel build

Docker support is disabled by default for Yocto images. To enable Docker integration at the kernel level, set the following option in your `local.conf` build as below:

```
DOCKER_SUPPORT = "1"  # Set to "1" to enable; "0" to disable (default)
```

> Note: After enabling Docker support, you must rebuild the kernel and replace the existing kernel image with the newly built one for the changes to take effect.

Step 2: Install Docker

Ensure the device has internet access, then update package lists and install Docker:

```
root@rz-cmn:~# apt-get update
root@rz-cmn:~# apt-get install docker.io
```

Step 3: Configure firewall compatibility, Docker supports only iptables-legacy and iptables-nft. Directly using nftables firewall rules is incompatible. Switch to legacy iptables with:

```
root@rz-cmn:~# update-alternatives --set iptables /usr/sbin/iptables-legacy
root@rz-cmn:~# update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy
```

Restart Docker to apply these changes:

```
root@rz-cmn:~# systemctl restart docker
```

Step 4: Verify Docker Installation by running:

```
root@rz-cmn:~# docker run hello-world
```

A successful run will display a message confirming Docker is working properly.

(2) Using DPKG to Install Packages

The utility ‘dpkg’ is the low-level package manager for Debian-based systems. It is the local system-wide package manager. It handles installation, removal, provisioning, indexing, and other aspects of packages installed on the system. However, it does not perform any cloud operations. Dpkg also does not handle dependency resolution. This is another task handled by a high-level manager like ‘apt-get’. In fact, ‘dpkg’ is the backend for ‘apt-get’. While ‘apt-get’ handles fetching and indexing, the local installations and management of the packages are performed by the ‘dpkg’ manager.

Basic dpkg commands:

- dpkg -i <package.deb>: Installs a package.deb package.
- dpkg -r <package>: Removes a package.
- dpkg -l <pattern>: Lists installed packages matching <pattern>.
- dpkg -s <package>: Provides information about an installed package.

You can install any <package>.deb (where ‘<package>’ is a placeholder for the name of the real package being installed) using dpkg with the following command:

```
root@rz-cmn:~# dpkg -i <package>.deb
```

After installing a package using dpkg, if you need to resolve dependency issues, use the following command:

```
root@rz-cmn:~# apt-get install -f
```

8.1.1.6 Python GUI Programming with Tkinter

This section outlines the steps for creating a basic graphical user interface (GUI) application using Tkinter, the standard Python interface to the Tk GUI toolkit. Tkinter is included with Python by default, so no additional libraries are required. It offers a straightforward way to develop desktop applications, making it a practical option for many use cases.

Note: Running graphical applications such as Tkinter requires access to the X11 display server, which is provided by Xwayland in this setup. Therefore, the application must be run as the weston user (not as root), because only that user has permission to access the running Xwayland display session (DISPLAY=:0).

The following steps will show how to create a new Tkinter application:

1. Switch to user ‘weston’

```
root@rz-cmn:~# su - weston
```

2. Create a working directory on the RZ board to develop and store the Python application.

```
rz-cmn:~$ mkdir ~/python_apl
rz-cmn:~$ cd ~/python_apl
```

3. Create a new Python file (For example, main.py) in the work directory.

```
rz-cmn:~/python_apl$ vi main.py
```

4. Develop a Simple Python GUI Application with tkinter.

- Import the tkinter module:
  ```
  import tkinter as tk
  ```
  This statement imports the Tkinter module, allowing access to its classes and functions for creating GUI elements.
- Create a main window.
  ```
  root = tk.Tk()
  ```
  This creates the main application window.
- Change the window title and resolution as desired.
  ```
  root.title(“Sample application”)
  root.geometry(“200x100”)
  ```

- Create and place a label.

  ```
  label = tk.Label(root, text="Press the button", width=20, height=2)
  label.pack()
  ```

- Create and place a button.

  ```
  button = tk.Button(root, text="Click Me", command=on_button_click, width=10,height=2)
  button.pack()
  ```

  This creates a button with the text "Click Me" and associates it with the on_button_click function.

  When the button is pressed, the function is called.

- Define a user function which helps to handle on click event and shows “Hello, Tkinter!” on the application’s window.

  ```
  def on_button_click():
      label.config(text="Hello, Tkinter!")
  ```

- Run the application

  ```
  root.mainloop()
  ```

  This starts the Tkinter event loop, which waits for user interactions and updates the UI accordingly.

- The completed Python program: “main.py”.

  ```
  import tkinter as tk

  def on_button_click():
      label.config(text="Hello, Tkinter!")

  root = tk.Tk()
  root.title("Sample application")
  root.geometry("200x100")

  # Create a label
  label = tk.Label(root, text="Press the button", width=20, height=2)
  label.pack()

  # Create a button
  button = tk.Button(root, text="Click Me", command=on_button_click, width=10,height=2)
  button.pack()

  # Run the application
  root.mainloop()
  ```

4. Run the application

- Ensure the RZ board is connected to an external display. If the display is not set automatically, set the DISPLAY environment variable as follows:

-
  ```
  rz-cmn:~$ export DISPLAY=:0
  ```

- Run the Python application:

  ```
  rz-cmn:~$ python3 main.py
  ```

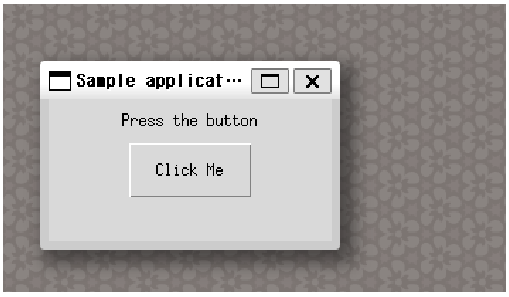


Figure 19. Initial GUI layout

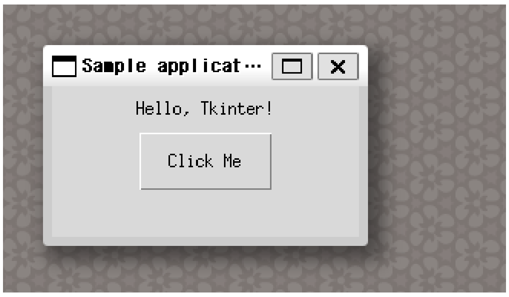


Figure 20. After the button ‘Click me’ is clicked

8.1.1.7 Install Packages Using Python3-Pip

The distribution includes Python 3 along with useful libraries/modules/packages such as Pip3, Numpy, Pandas, PySerial, Matplotlib, etc. This section will focus on using Pip3, the package installer for Python 3, to manage additional packages.

Python3-pip allows you to install, update, and manage Python packages from the Python Package Index (PyPI) and other repositories.

To install a new package using pip3, use the following command:

```
root@rz-cmn:~# pip3 install <package_name>
```

For example, to install the `requests` package, you would run:

```
root@rz-cmn:~# pip3 install requests
```

To verify that the `requests` package (or any other installed package) is correctly installed, you can use:

```
root@rz-cmn:~# pip3 show requests
```

This command provides details about the requests package, including its version and installation location.

Alternatively, you can list all installed packages and check if the `requests` package is included:

```
root@rz-cmn:~# pip3 list
```

This will confirm that the package is installed and available for use.

8.1.2 RZ/G2L-SBC Yocto Features

8.1.2.1 40-Pin IO Expansion Interface

The 40 IO Expansion Interface on RZ/G2L-SBC has support for:

- I2C channel 0
- I2C channel 3
- SPI channel 0
- SCIF channel 0
- CAN channel 0
- CAN channel 1
- GPIO pin-function (default).

Notes:

- The GPIO pin array is multiplexed with peripheral IO lines.
- By default, I2C channel 0 and SCIF channel 0 are enabled.
- The rest of the pins are GPIO's by default.
- Enable the other functions by editing the uEnv.txt on the SD card and enabling the appropriate device tree overlay file (DT overlays). This is also how some of the dedicated drivers are enabled, like the display.
- Reboot the board for the overlay to take effect.

(1) U-Boot Environment

The `uEnv.txt` file contains boot configuration settings for both the U-Boot bootloader and the Linux kernel.

This file resides in the boot partition (Partition 1) of the storage media. This partition is formatted as FAT32, which allows it to be easily read by almost any operating system, including Windows. When the device's Linux system is running, this FAT32 partition is typically mounted at the /boot directory.

While the U-Boot environment is extensive, this guide focuses on the essential settings for this SBC. The Table 11. Boot configuration settings below provide a list of all the overlay options available in the provided kernel.

Table 16. Boot configuration settings

Config	Value if set	Loading	Description
enable_overlay_i2c	1 or 'yes'	rzg2l-sbc-ext-i2c.dtbo	Enables the i2c driver enumeration and reconfigures the relevant IO pins to connect to the I2C peripheral.
enable_overlay_spi	1 or 'yes'	rzg2l-sbc-ext-spi.dtbo	Enables the SPI driver enumeration and reconfigures the relevant IO pins to connect to the SPI peripheral.
enable_overlay_can	1 or 'yes'	rzg2l-sbc-can.dtbo	Enables the CAN driver enumeration and reconfigures the relevant IO pins to connect to the CAN peripheral.
enable_overlay_dsi	1 or 'yes'	rzg2l-sbc-dsi.dtbo	Enables the Waveshare DSI to display touch panel driver enumeration and reroutes the video to DSI.
enable_overlay_csi_ov5640	1 or 'yes'	rzg2l-sbc-ov5640.dtbo	Enables the OV5640 CSI camera driver enumeration and loads the v4l2 pipelines.

A Readme.md file with potentially more up-to-date descriptions of the FDT overlays can also be found in the boot partition.

(A) HOW TO EDIT UENV.TXT

Because the 'uEnv.txt' file is on a FAT32 partition, it can be modified easily using one of two methods.

Method 1: Editing from a Windows PC (Recommend for simplicity)

This method is straightforward, following the steps below.

1. Access the SD Card: Power down the SBC and remove its SD card. Insert the card into a reader connected to a Windows PC.
2. Open the Boot Partition: Windows will automatically mount the FAT32 partition (Partition 1), which will appear as a removable drive. Open this drive.
3. Edit the File: Locate and open uEnv.txt with a plain text editor like Notepad or Notepad++.
4. Enable a Peripheral: To activate an overlay, set the value of enable_overlay_ variable to 1 or 'yes'.
5. Save and Eject: Save the changes. Use the "Safely Remove Hardware and Eject Media" option in Windows before physically removing the SD card to prevent data corruption.
6. Boot the Device: Return the SD card to the SBC and power it on.

Method 2: Editing from the Root File System (Linux)

This method ensures a reliable way to modify the file by manually mounting the boot partition.

1. Identify the boot partition

   Run the lsblk command to list your storage devices. Look for the small FAT32 partition on the SD card. Its name will typically be /dev/mmcblk0p1 or similar.

   ```
   root@rz-cmn:~# lsblk
   ```

2. Create a mount point: Make a temporary directory to mount the partition.

   ```
   root@rz-cmn:~# mkdir -p /mnt/boot_part
   ```

3. Mount the partition: Mount the boot partition you identified in the previous step. Replace /dev/mmcblk0p1 if your device has a different name.

   ```
   root@rz-cmn:~# mount /dev/mmcblk0p1 /mnt/boot_part
   ```

4. Modify the file: Edit the uEnv.txt file using a text editor.

   ```
   root@rz-cmn:~# vi /mnt/boot_part/uEnv.txt
   ```

5. Save and Sync: After saving the file, you must run the sync command to ensure the changes are written from memory to the SD card.

   ```
   root@rz-cmn:/mnt/boot_part# sync
   ```

6. Umount the partition: Umount the partition before rebooting

   ```
   root@rz-cmn:/mnt/boot_part# cd ~
   root@rz-cmn:~# umount
   root@rz-cmn:~# sync
   root@rz-cmn:~# reboot
   ```

Device tree file changes require the SBC to be rebooted to take effect.

(2) GPIO (General Purpose I/O pins) with libgpiod

By default, most pins are configured as GPIOs on the SBC's 40-pin GPIO pin header. This section details how to identify and control these pins using the libgpiod library and its associated command-line tools. The IO pins are explored in detail in Figure 94. 40 PIN GPIO map with orientation details.

Unlike the deprecated sysfs interface, libgpiod provides a standardized and kernel-integrated method for GPIO management. It interacts with GPIO character devices (e.g., /dev/gpiochip0, /dev/gpiochip1) to offer a more efficient and flexible control over individual GPIO lines.

All GPIO pins on the 40-pin header are exposed through /dev/gpiochip0.

Table 17. GPIO pins and functions

GPIO Line	Function	Group	Pin	J3 PINs		Pin	Group	Function	GPIO Line
				Left side	Right side				
	3.3V			**1**	**2**			5V	
370	I^{2}C3 SDA	46	2	**3**	**4**			5V	
371	I^{2}C3 SCL	46	3	**5**	**6**			GND	
184	GPIO	23	0	**7**	**8**	0	38	SCIF0 TX	304
	GND			**9**	**10**	1	38	SCIF0 RX	305
456	GPIO	42	0	**11**	**12**	2	7	GPIO	58
336	GPIO	27	0	**13**	**14**			GND	
225	GPIO	28	1	**15**	**16**	0	8	GPIO	64
	3.3V			**17**	**18**	0	15	GPIO	120
345	SPI0 MOSI	43	1	**19**	**20**			GND	
346	SPI0 MISO	43	2	**21**	**22**	1	14	GPIO	113
344	SPI0 CK	43	0	**23**	**24**	3	43	SPI0 CS	347
	GND			**25**	**26**	1	11	GPIO	89
	I^{2}C0 SDA			**27**	**28**			I^{2}C0 SCL	
32	GPIO	4	0	**29**	**30**			GND	
33	GPIO	4	1	**31**	**32**	0	32	GPIO	256
177	GPIO	22	1	**33**	**34**			GND	
337	CAN0 TX	42	1	**35**	**36**	1	23	GPIO	**185**
88	CAN0 RX	11	0	**37**	**38**	0	46	CAN1 TX	**368**
	GND			**39**	**40**	1	46	CAN1 RX	**369**

(A) UNDERSTANDING LIBGPIOD'S CONCEPTS: CHIPS AND LINES

Instead of a single, linear pin number system, libgpiod organizes GPIOs around two key concepts:

- GPIO Chips: These represent the physical GPIO controllers on your system. Each chip manages a specific set of GPIO lines. You'll typically see them identified as gpiochip0, gpiochip1, and so on.
- GPIO Lines: Each chip contains a number of individual GPIO lines, identified by an offset within that chip (e.g., line 0, line 1, line 2, etc.).

(B) IDENTIFYING GPIO CHIPS AND LINES

Before GPIO control can be initiated, the specific chip and line offset corresponding to the desired pin must be identified.

The gpiodetect command lists all GPIO controllers on the system:

```
root@rz-cmn:# gpiodetect
```

Output will be similar to

```
root@rz-cmn:# gpiochip0 [chip_name_0] (XX lines)
root@rz-cmn:# gpiochip1 [chip_name_1] (YY lines)
# ... additional chips
```

(C) **INSPECT LINES ON A SPECIFIC GPIO CHIP:**

Detailed information about individual lines on a chip is obtained using gpioinfo:

```
root@rz-cmn:# gpioinfo gpiochip0
```

Please replace gpiochip0 with the relevant chip name or number identified via gpiodetect. This command lists each line, including its offset, any assigned name, and its current state. This output is essential for mapping physical pins to libgpiod's chips and offset.

(3) Enabling I2C Function (Channel 3 – RIIC3)

Edit `uEnv.txt` and uncomment the line as follows to enable I2C channel 3 on the 40 IO expansion interface:

Change the following line:

```
#enable_overlay_i2c=1
```

To

```
enable_overlay_i2c=1
```

Then reboot the RZ/G2L-SBC.

To check if I2C channel three is enabled, run the following command and check the result:

```
root@rz-cmn:~# i2cdetect -l
i2c-3   i2c             Renesas RIIC adapter                I2C adapter
i2c-1   i2c             Renesas RIIC adapter                I2C adapter
i2c-4   i2c             i2c-1-mux (chan_id 0)               I2C adapter
i2c-0   i2c             Renesas RIIC adapter                I2C adapter
root@rz-cmn:~#
```

To map out all the devices present on the I2C bus, execute the following command:

```
root@rz-cmn:~# i2cdetect -y -r 3
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Any device present on the bus will be marked with the appropriate i2c device ID.

(4) SPI function (Channel 0 – RSPI0)

Edit `uEnv.txt` as follows to enable SPI channel 0 on the 40 IO expansion interface:

Change the following line:

```
#enable_overlay_spi=1
```

To

```
enable_overlay_spi=1
```

This will enable the SPI module.

Run the following command to configure the SPI:

```
root@rz-cmn:~# spi-config -d /dev/spidev0.0 -q
/dev/spidev0.0: mode=0, lsb=0, bits=8, speed=2000000, spiready=0
```

Connect Pin 19 (RSPI0 MOSI) to Pin 21 (RSPI0 MISO), then run the below command and check the result. The idea is to transmit on MOSI and read back on MISO to validate the transfer.

```
root@rzg2l-sbc:~# echo -n -e "1234567890" | spi-pipe -d /dev/spidev0.0 -s
10000000 | hexdump
0000000 3231 3433 3635 3837 3039
000000a
```

(5) CAN Function (Channel 0,1 - CAN 0, CAN 1)

Edit `uEnv.txt` as follows to enable CAN channel 0,1 on 40 IO expansion interface:

Change the following line:

```
#enable_overlay_can=1
```

To

```
enable_overlay_can=1
```

To verify that the CAN channels are enabled, run the following command and check the result:

```
root@rz-cmn:~# ip a | grep can
3: can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
4: can1: <NOARP,ECHO> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
root@rzg2l-sbc:~#
```

Then set up for CAN devices. Now you can go up/down the interface or send data over CAN channels.

The example below shows the communication between two CAN channels.

```
root@rz-cmn:~# ip link set can0 down
root@rz-cmn:~# ip link set can0 type can bitrate 500000
root@rz-cmn:~# ip link set can0 up
[   48.120419] IPv6: ADDRCONF(NETDEV_CHANGE): can0: link becomes ready
root@rz-cmn:~# ip link set can1 down
root@rz-cmn:~# ip link set can1 type can bitrate 500000
root@rz-cmn:~# ip link set can1 up
[   69.906039] IPv6: ADDRCONF(NETDEV_CHANGE): can1: link becomes ready
root@rz-cmn:~# candump can0 & cansend can1 123#01020304050607
[1] 271
  can0  123   [7]  01 02 03 04 05 06 07
root@rz-cmn:~# candump can1 & cansend can0 123#01020304050607
[2] 273
  can0  123   [7]  01 02 03 04 05 06 07
  can1  123   [7]  01 02 03 04 05 06 07
root@rz-cmn:~#
```

8.1.2.2 Wi-Fi 802.11 Module

RZ/G2L-SBC comes equipped with an onboard wireless 802.11 module. The image is ready with all the necessary tools to connect to Wi-Fi. The Wi-Fi can be configured on the command line, which can either be on the desktop UI or the UART tty from the host.

The following shows how to enable the 802.11 Wi-Fi module and connect to a network.

```
root@rz-cmn:~# connmanctl
connmanctl> enable wifi
Enabled wifi
connmanctl> agent on
Agent registered
connmanctl> scan wifi
Scan completed for wifi
connmanctl> services
    xDredme10zW         wifi_0025ca329da3_78447265646d6531307a57_managed_psk
                        wifi_0025ca329da3_hidden_managed_psk
    REL-GLOBAL          wifi_0025ca329da3_52454c2d474c4f42414c_managed_ieee8021x
    R-GUEST             wifi_0025ca329da3_522d4755455354_managed_none
    RVC-WLS             wifi_0025ca329da3_5256432d574c53_managed_ieee8021x
connmanctl> connect wifi_0025ca329da3_78447265646d6531307a57_managed_psk
Agent RequestInput wifi_0025ca329da3_78447265646d6531307a57_managed_psk
  Passphrase = [ Type=psk, Requirement=mandatory ]
Passphrase? nFjey48aT9pk
connmanctl> exit
```

To confirm the Wi-Fi is connected, ping to the outside world:

```
root@rz-cmn:~# ping www.google.com
PING www.google.com(hkg07s39-in-x04.1e100.net (2404:6800:4005:813::2004)) 56 data bytes
64 bytes from hkg07s39-in-x04.1e100.net (2404:6800:4005:813::2004): icmp_seq=1 ttl=57
time=43.2 ms
64 bytes from hkg07s39-in-x04.1e100.net (2404:6800:4005:813::2004): icmp_seq=2 ttl=57
time=81.1 ms
64 bytes from hkg07s39-in-x04.1e100.net (2404:6800:4005:813::2004): icmp_seq=3 ttl=57 time=124
ms
```

Note: The ethernet interfaces may potentially interfere with the routing of the communication through Wi-Fi. If issues start appearing, use the following commands to disable the ethernet ports.

```
root@rz-cmn:~# ifconfig end0 down
root@rz-cmn:~# ifconfig end1 down
```

8.1.2.3 MIPI DSI Display Touch Panel

The RZ/G2L-SBC has an MIPI DSI interface that supports both a display module and a touch interface. The DSI port supports dual-channel DSI and one I2C interface in the connector.

(1) Hardware Interfacing

Given below are pictures of Waveshare 5” DSI display panel with touch screen assembly. The pictures are self-explanatory.

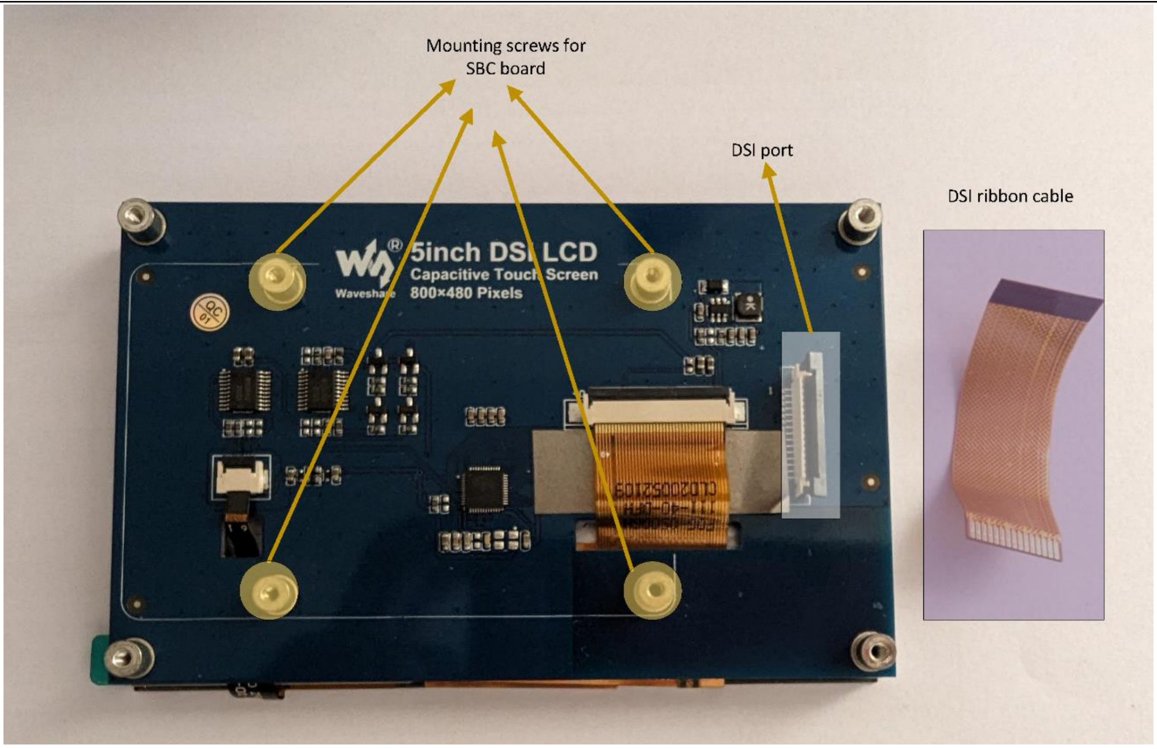


Figure 21. Waveshare 5" DSI touch panel read side picture with flat ribbon cable.

FPC connector locking and unlocking is done by pulling up the black notch or pushing it down. Unlock the connector by pulling up the notch, as shown below.

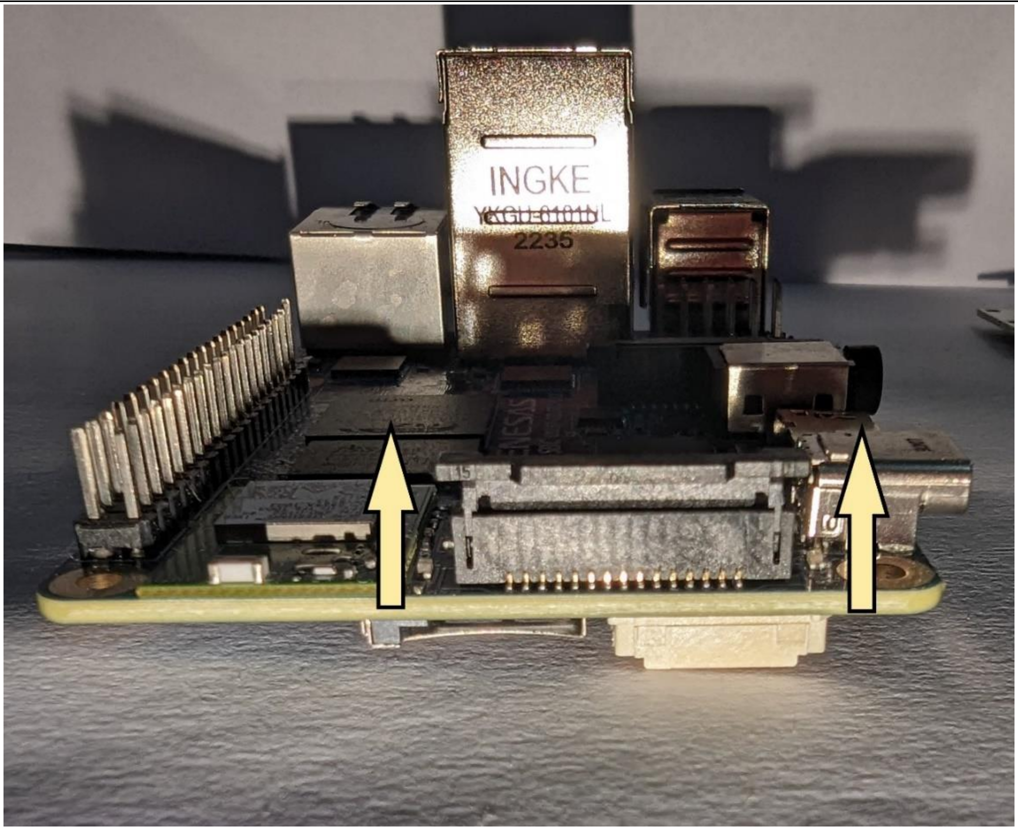


Figure 22. DSI port notch lock open by pulling it up

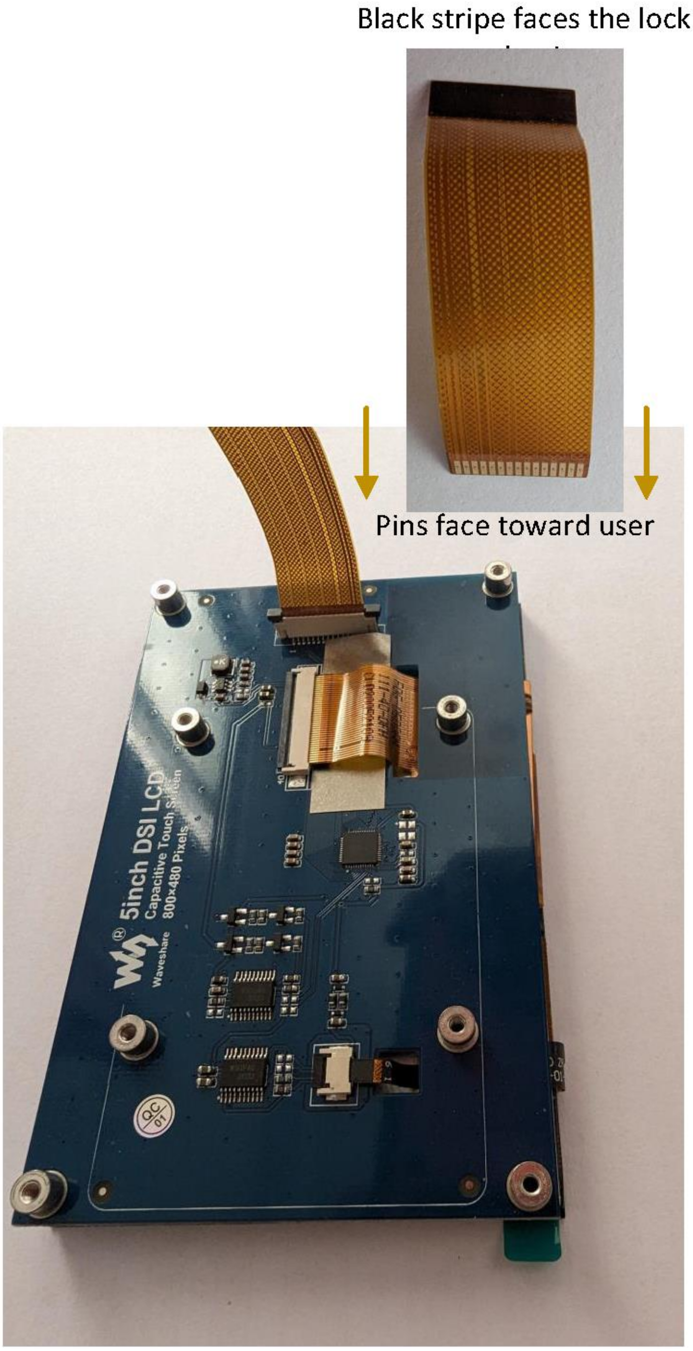


Figure 23. Waveshare DSI touch display DSI port interfacing cable orientation.

Mount the RZ/G2L-SBC onto the rear end of the display panel.

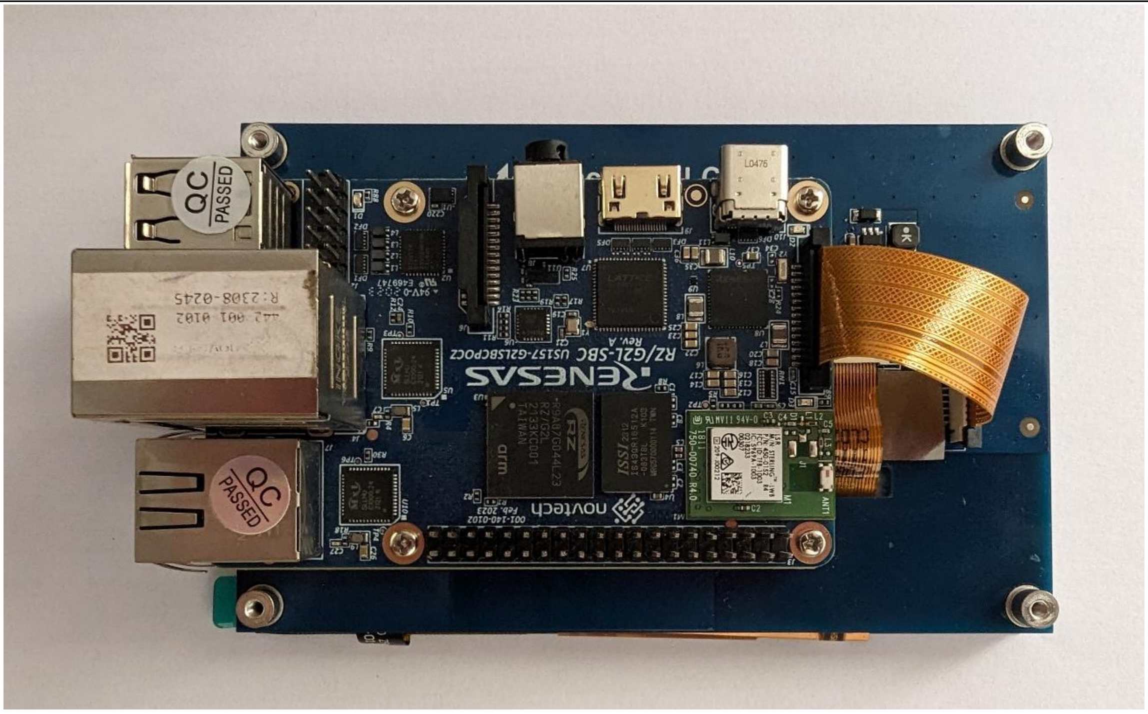


Figure 24. RZ/G2L-SBC mounted to the Waveshare DSI panel and interfaced.

Insert the other end of the FPC cable into the RZ/G2L-SBC DSI port and lock it. The locking mechanism is shown below.

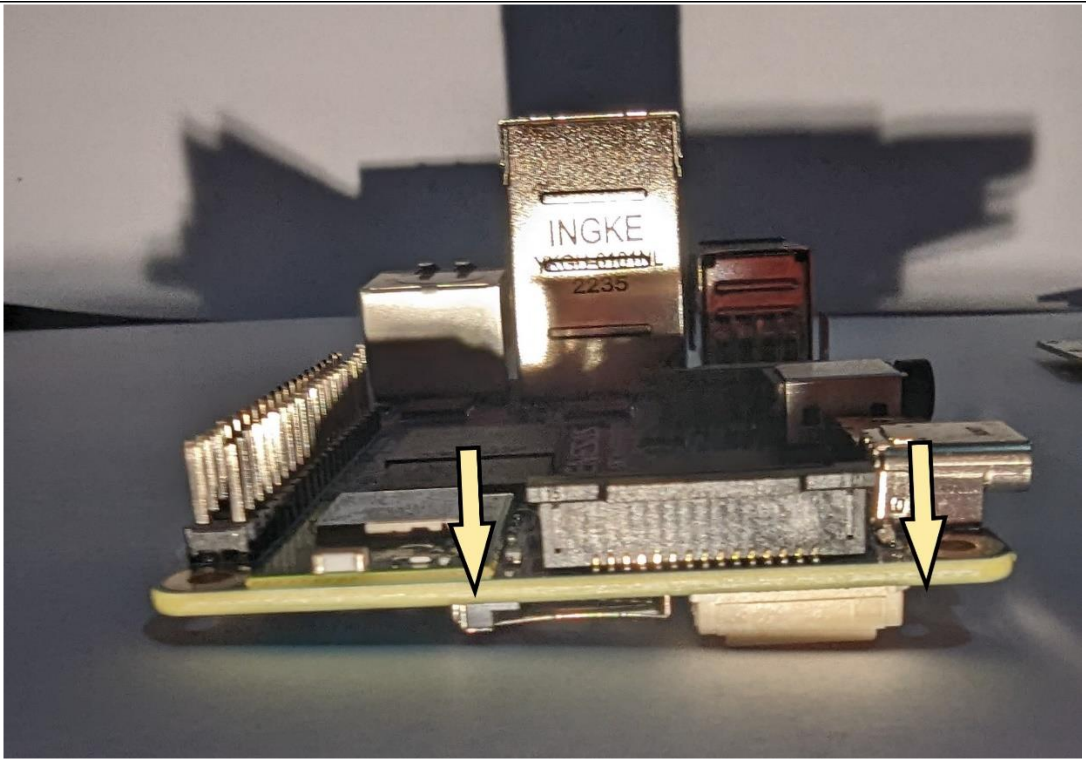


Figure 25. DSI port notch in the lock position. The cable is not shown to keep the notch in clear view.

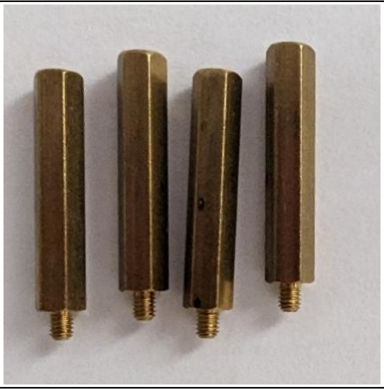


Figure 26. Metal support screws supplied by Waveshare

The Waveshare DSI display panel comes with four metal supports that raise the display, along with the rear-attached SBC, off the surface to provide sturdy support with clearance. However, these are not high enough for the RZ/G2L-SBC due to the SBC having dual Ethernet ports, where one port is too high, sitting on top of the two USB ports. We still recommend that you use a support stand, even an off-market custom one, to ensure that the DSI cable is off the ground.

Remember that the DSI port includes an I^{2}C two-wire interface that supports a touch panel interface without any extra cabling.

Note: The dark, solid stripe on the flat cable always faces the black locking mechanism of the connector. Do not insert the cable in reverse, as this could potentially damage the board due to incorrect electrical connections.

(2) Enabling DSI Panel Drivers

The Linux distribution supports the Waveshare 5-inch Touchscreen MIPI-DSI LCD capacitive touch panel.

By default, the video output is directed toward the mini-HDMI port. To enable the panel drivers and reroute the display to the DSI panel, you need to enable the panel driver DT overlay in uEnv.txt.

Open the `uEnv.txt` and change the following line:

```
#enable_overlay_dsi=1
```

To

```
enable_overlay_dsi=1
```

Reboot the SBC board.

Note: Enabling the MIPI DSI panel overlay disables the HDMI display. You can only use one at a time.

8.1.2.4 MIPI CSI2 with Arducam 5MP OV5640 Camera Module

RZ/G2L-SBC supports the MIPI CSI-2 camera interface. The Linux distribution supports the Arducam 5MP MIPI OV5640 image sensor-based module.

(1) Hardware Interfacing

The Arducam OV5640 camera module is easily installed into the RZ/G2L-SBC.

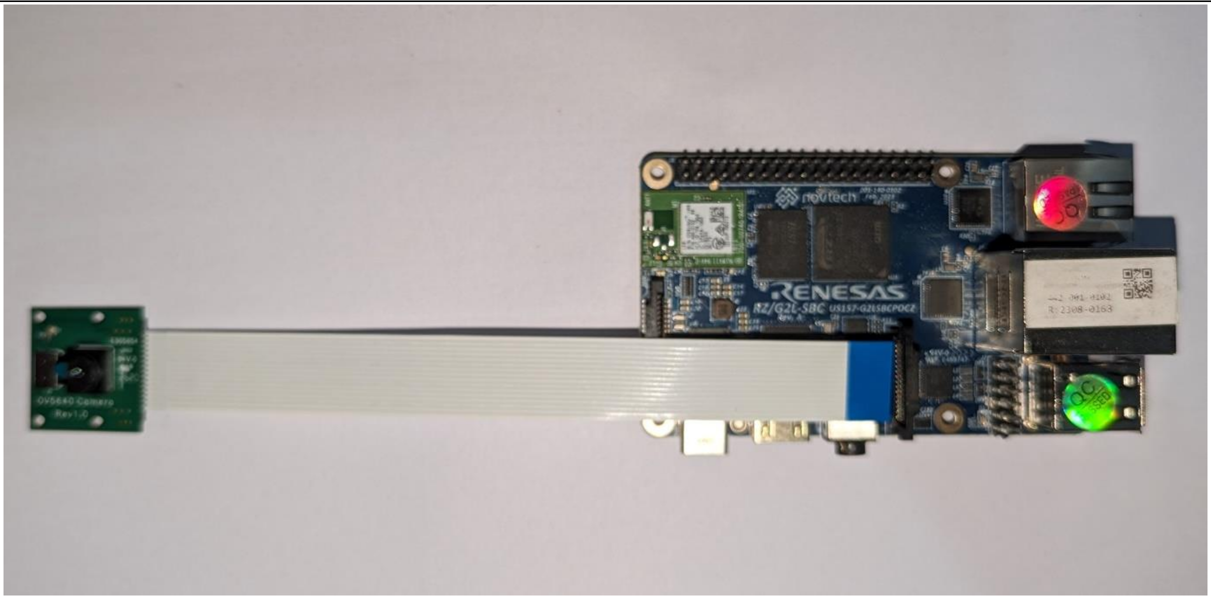


Figure 27. Orientation of the camera module. Blue stripe upward.

The black notch must be pulled up to unlock it.

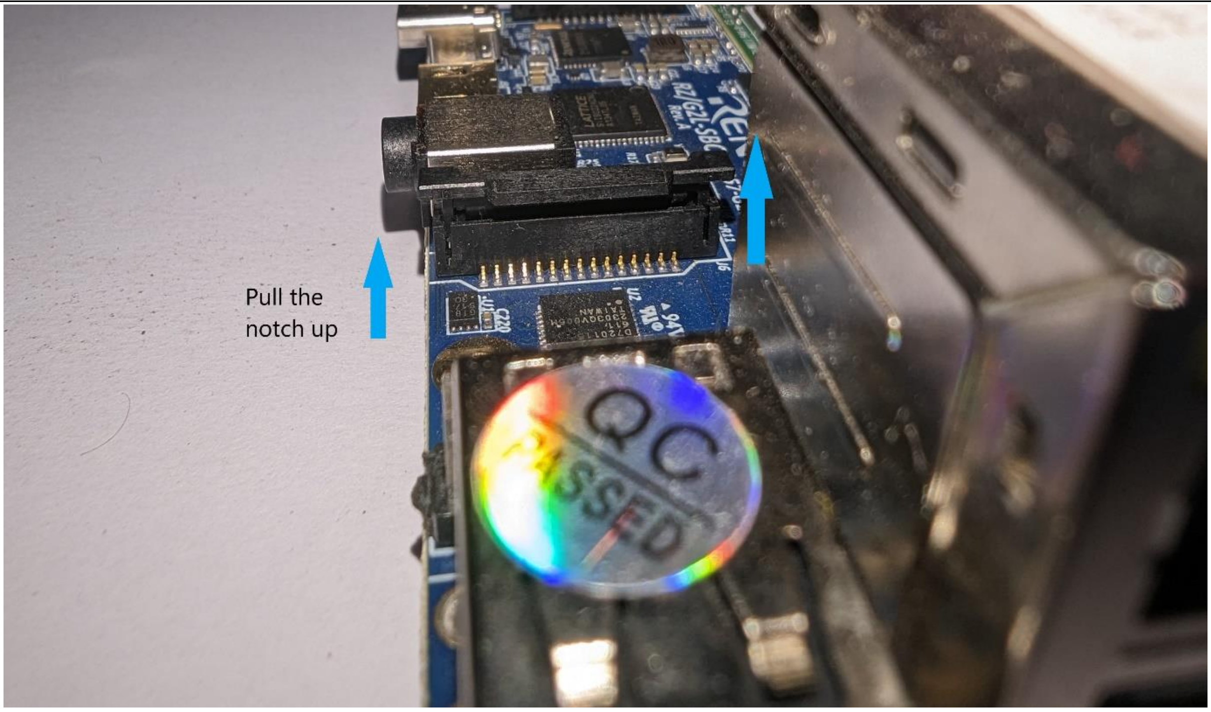


Figure 28. Pull the notch up to unlock it.

Insert the flat cable in the correct orientation, as depicted in the pictures.

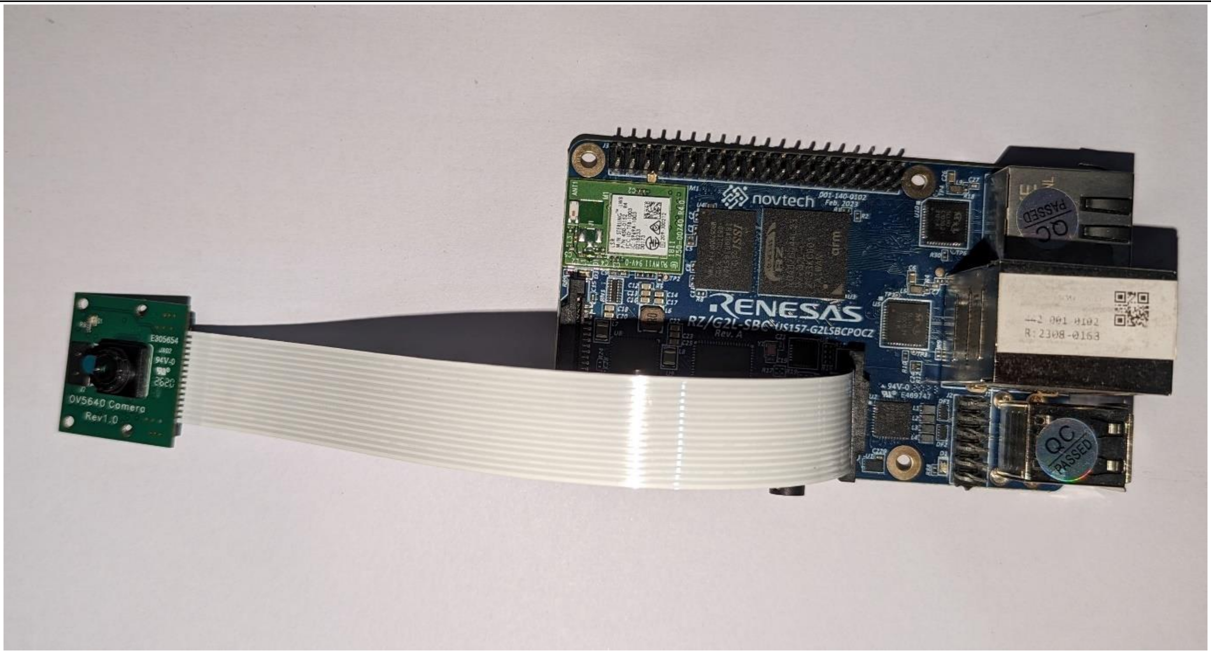


Figure 29. The CSI module is inserted.

Push down on the notch to lock it with the flat cable inserted.

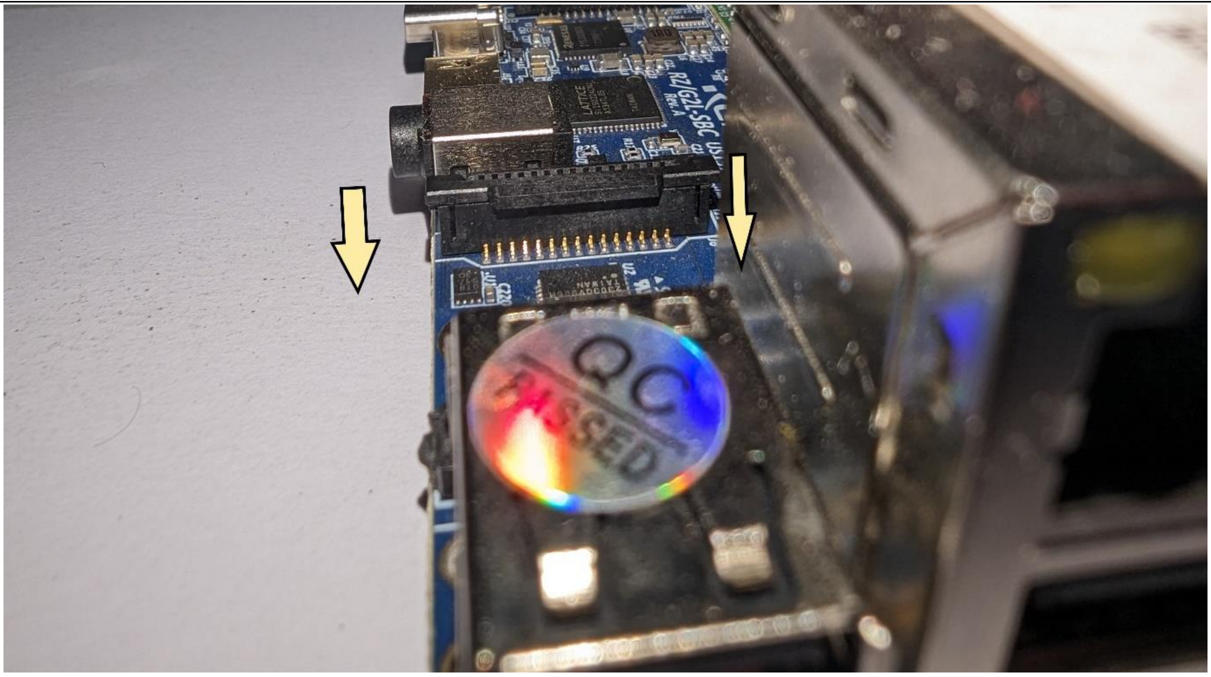


Figure 30. Push down the notch to lock it when you have inserted the flat cable

(2) Enabling CSI Camera Drivers

To enable the camera, edit the uEnv.txt and enable the following line:

```
#enable_overlay_csi_ov5640=1
```

To

```
enable_overlay_csi_ov5640=1
```

Reboot the board.

(3) Accessing the Camera

Before initializing the camera capture, it needs to be enabled and configured. The Linux distribution has a helper script (v4l2-init.sh) in the /home/root directory to enable and configure the camera.

```
root@rz-cmn:~# cd /home/root/
root@rz-cmn:~# ./v4l2-init.sh <resolution>
```

The argument <resolution> specifies the resolution for the camera. Valid resolutions are:

- 720x480
- 720x576
- 1024x768
- 1280x720
- 1920x1080
- 2592x1944

If no resolution is specified or an invalid resolution is provided, the default resolution, 1280x960, will be used. For example:

When using a valid resolution:

```
root@rz-cmn:~# ./v4l2-init.sh 1920x1080
Link CRU/CSI2 to ov5640 1-003c with format UYVY8_1X16 and resolution 1920x1080
```

When no resolution is specified:

```
root@rz-cmn:~# ./v4l2-init.sh
No resolution specified. Using default resolution: 1280x720
Link CRU/CSI2 to ov5640 1-003c with format UYVY8_1X16 and resolution 1280x720
```

The `v4l2-init.sh` script helps enable the CSI-2 module and select the camera's supported display resolution.

After running the script, initiate a video capture session using the matching width and height:

```
root@rz-cmn:~# gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-
raw,width=1280,height=720 ! videoconvert ! waylandsink
```

Ensure that the width and height values in the GStreamer pipeline match the resolution specified in v4l2-init.sh. This command starts a continuous stream of the camera feed to the active video display.

8.1.2.5 Accessing PWM Timers

The RZG2L-SBC provides PWM (Pulse Width Modulation) timers, which can be used for various applications, including motor control, LED dimming, and signal generation for external devices. PWM allows for precise control over voltage levels by adjusting the duty cycle, making it useful in scenarios requiring variable power output.

(1) Overview

The RZ/G2L-SBC's device tree source (DTS) includes two GPT channels by default, providing PWM functionality for three pins.

- GPT4: Supports two PWM channels (channel_A and channel_B).

- GPT5: Supports a signal PWM channel A.

However, these channels are disabled by default because the GPT4 pins are assigned to SPI, and the GPT5 pins are used for DSI. If these resources are repurposed for PWM, then the default functions (SPI or DSI) will no longer be available.

```
&gpt4 {
      pinctrl-0 = <&gpt4_pins>;
      pinctrl-names = "default";
      channel = "both_AB";
      poeg = <&poega &poegb &poegc &poegd>;
      status = "disabled";
};

&gpt5 {
      pinctrl-0 = <&gpt5_pins>;
      pinctrl-names = "default";
      channel="channel_A";
      poeg = <&poegd>;
      status = "disabled";
};
```

To enable the use of PWM, follow the steps in the next subsection:

(2) Enabling GPT Channels for PWM Use

This section explains how to enable GPT channels for PWM on the RZ/G2L-SBC. By default, the GPT channels are disabled in the device tree, so they need to be enabled manually.

Note: Ensure you have internet access before running the commands.

1. Install the device tree compiler tool.

```
root@rz-cmn:~# apt-get update
root@rz-cmn:~# apt-get install device-tree-compiler
```

2. Decompile the dtb file into a dts file.

The Device Tree Blob (DTB) is typically stored on a dedicated boot partition, which is often formatted as FAT32. This partition needs to be mounted to access its contents.

Create a temporary mount point and mount the boot partition (partition 1 – FAT32)

```
root@rz-cmn:~# mkdir -p /mnt/boot_partition
root@rz-cmn:~# mount /dev/mmcblk0p1 /mnt/boot_partition/
```

Following successful mounting, the specific DTB file can be located within `/mnt/boot_partition/``

```
root@rz-cmn:~# dtc -I dtb -O dts -f /mnt/boot_partition/dtb/renesas/rzg2l-sbc.dtb
-o rzg2l-sbc.dts
```

3. Modify the dts file.

Open the rzg2l-sbc.dts file in a text editor.

```
root@rz-cmn:~# vi rzg2l-sbc.dts
```

For GPT4, locate gpt@10048400

For GPT5, locate gpt@10048500

Change the status property of the node you want to enable from "disabled" to "okay". Save the file after making the changes.

4. Recompile the dts file back into a dtb file.

```
root@rz-cmn:~# dtc -I dts -O dtb -f rzg2l-sbc.dts -o new_rzg2l-sbc.dtb
```

5. Deploy the new dtb file:

Replace the original dtb file with the newly compiled one.

Note: It is recommended to back up the original DTB file beforehand. After recompiling the DTS into a DTB and deploying it to /mnt/boot_partition/dtb/renesas/rzg2l-sbc.dtb in partition 1, ensure that the file retains its original name. If the DTB file is missing or renamed, the boot process may fail.

```
root@rz-cmn:~# cp new_rzg2l-sbc.dtb /mnt/boot_partition/dtb/renesas/rzg2l-sbc.dtb
```

6. Sync and umount the partition

```
root@rz-cmn:~# cd ~
root@rz-cmn:~# sync
root@rz-cmn:~# umount /mnt/boot_partition
root@rz-cmn:~# sync
```

7. Reboot the system to apply the changes.

After booting up, if everything is configured correctly, the PWM device file will be automatically generated in /sys/class/pwm/pwmchipX, where X can be 0, 1, 2, and so on.

(3) Enable PWM channels

Before using PWM, the channels need to be exported to the system.

For example, to use PWM chip 0 and export channel 0, the following steps are required.

```
root@rz-cmn:~# cd /sys/class/pwm/pwmchip0/
root@rz-cmn:/sys/class/pwm/pwmchip0# echo 0 > export
```

(4) Configuring PWM

To configure a single PWM channel (For example, from GPT5), follow these steps:

In this example, the period is set to 1,000,000 nanoseconds, and the duty cycle is configured to 500,000 nanoseconds, which is 50% of the period. Adjust these values as needed to achieve the desired PWM output.

1. Modify the duty cycle and period.

Set the period (in nanoseconds).

```
root@rz-cmn:/sys/class/pwm/pwmchip0/# cd pwm0
root@rz-cmn:/sys/class/pwm/pwmchip0/pwm0# echo 1000000 > period
```

Set the duty cycle (in nanoseconds).

```
root@rzg2l-sbc:/sys/class/pwm/pwmchip0/pwm0# echo 500000 > duty_cycle
```

2. Enable the PWM to start output.

```
root@rzg2l-sbc:/sys/class/pwm/pwmchip0/pwm0# echo 1 > enable
```

For devices like GPT4 that provide two PWM channels (channel A and channel B), each channel needs to be configured separately.

1. Modify the period.

Define the period for both channels in nanoseconds. For example, to set the period to 100,000 nanoseconds, use the following command:

```
root@rzg2l-sbc:/sys/class/pwm/pwmchip0/pwm0# echo 1000000 > period
```

2. Enable the PWM to start output

```
root@rzg2l-sbc:/sys/class/pwm/pwmchip0/pwm0# echo 1 > enable
```

3. Modify the duty cycle for each channel.

Navigate to the device directory to configure the duty cycles for both channels.

```
root@rzg2l-sbc:/sys/class/pwm/pwmchip0/pwm0# cd /sys/class/pwm/pwmchip0/device
root@rzg2l-sbc:/sys/class/pwm/pwmchip0/device# echo 1000000 > buffA0
root@rzg2l-sbc:/sys/class/pwm/pwmchip0/device# echo 500000 > buffB0
```

In this example, channel A is set to a duty cycle of 1,000,000 nanoseconds, while channel B is set to 500,000 nanoseconds. Adjust these values as needed for the desired PWM output.

8.2 Supported Features in Ubuntu Images

Before accessing the features available in both the Ubuntu Core and Ubuntu LXDE images on the supported platforms, please log in using the default credentials:

- **Username:** rz
- **Password:** 1

After logging in, the supported features can be explored and interacted with, as detailed below.

8.2.1 Accessing Supported Features in Ubuntu LXDE

8.2.1.1 Selecting LXDE session

The LXDE desktop environment is enabled by default:

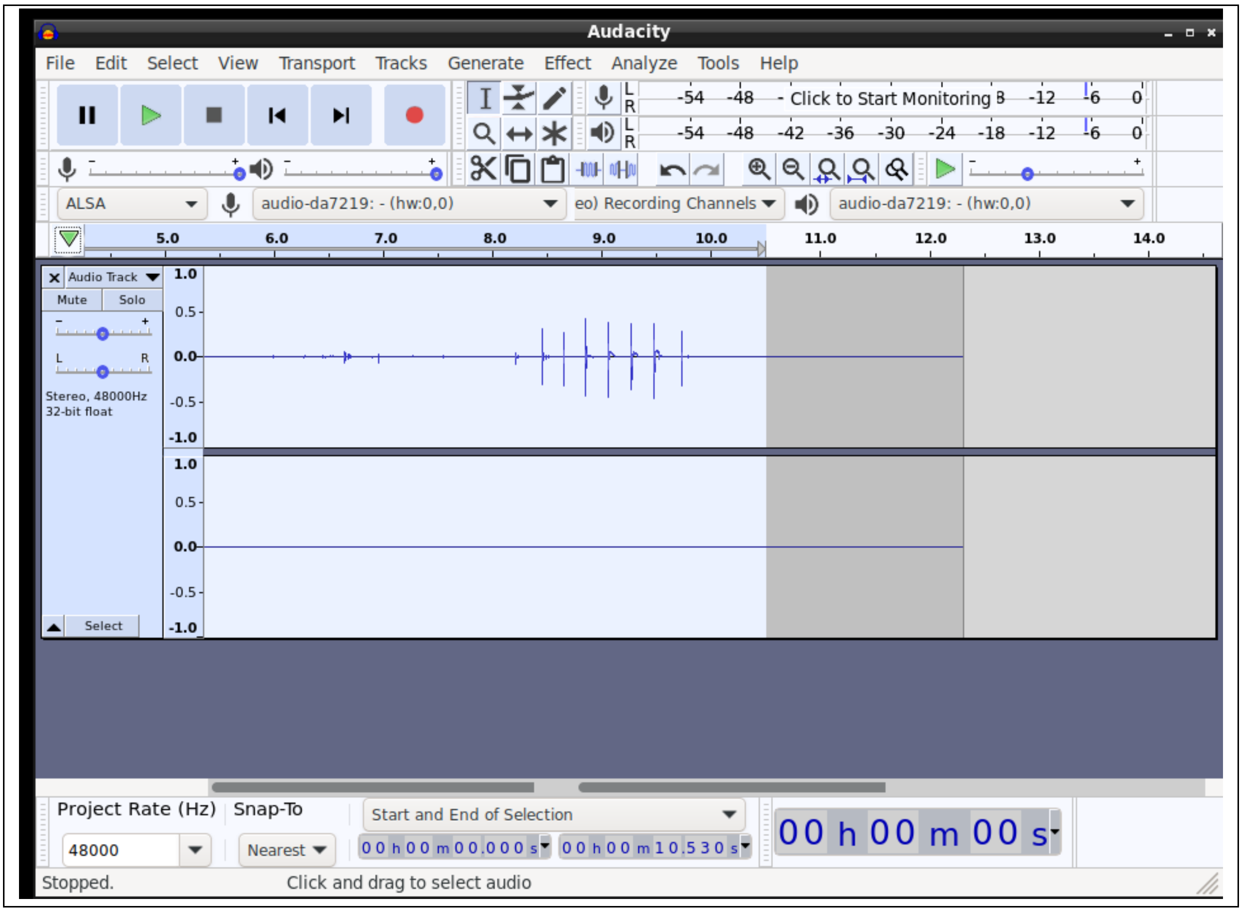
1. On first login, the system automatically launches LXDE as the desktop environment.
2. No manual selection is required, providing seamless user experience.

If you wish to use a different desktop environment, click the gear ⚙ icon in the bottom-right corner of the login screen and choose an alternative. However, note that this may result in a different experience from LXDE’s lightweight and responsive interface.

This default configuration ensures that users can immediately take advantage of LXDE’s performance and simplicity without requiring additional setup.

8.2.1.2 Audacity

Audacity is a free, open-source, cross-platform audio software that is used for recording, editing, and producing audio. It allows users to capture live audio, convert tapes and records into digital recordings, and edit audio files in a variety of formats. Audacity is widely used for tasks such as podcasting, music production, and audio analysis due to its user-friendly interface and powerful editing tools. It supports multi-track editing, numerous audio effects, and plugins, making it a popular choice for both amateurs and professionals.



ALSA | audio-da7219: - (hw:0,0) | eo) Recording Channels | audio-da7219: - (hw:0,0)

Stereo, 48000Hz
32-bit float

Project Rate (Hz): 48000
Snap-To: Nearest
Start and End of Selection: 00 h 00 m 00.000 s – 00 h 00 m 10.530 s
00 h 00 m 00 s

Stopped. Click and drag to select audio

Figure 31. Audacity graphic user interface

To properly configure Audacity for the system:

1. Open Audio Settings: In Audacity, click Audio Setup in the top-right corner, then select Audio Settings.
2. Select the correct playback and recording devices: In the window that opens, set both the Playback and Recording Device to the appropriate sound card for the current board in use (e.g., for the RZG2L-SBC, select the corresponding device).

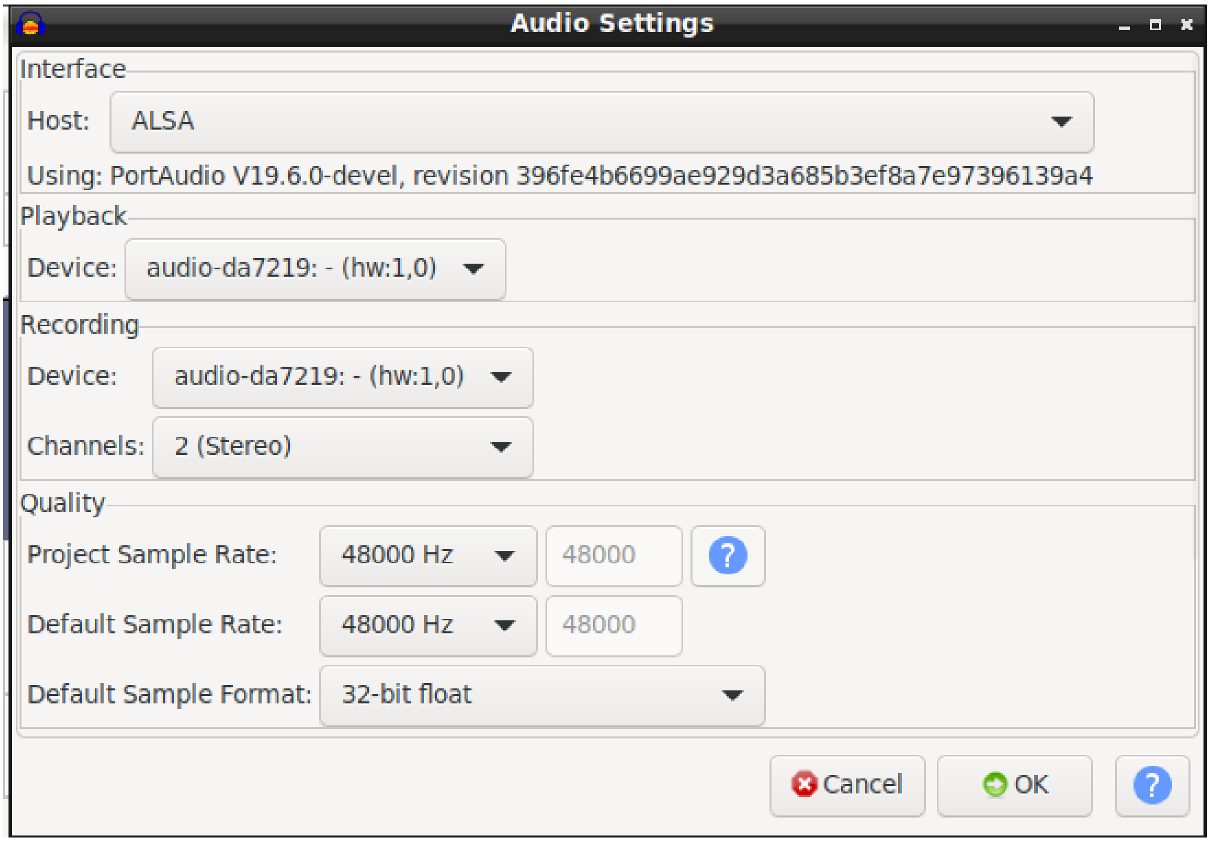

Audio Settings
Interface
Host: ALSA
Using: PortAudio V19.6.0-devel, revision 396fe4b6699ae929d3a685b3ef8a7e97396139a4
Playback
Device: audio-da7219: - (hw:1,0)
Recording
Device: audio-da7219: - (hw:1,0)
Channels: 2 (Stereo)
Quality
Project Sample Rate: 48000 Hz 48000
Default Sample Rate: 48000 Hz 48000
Default Sample Format: 32-bit float
Cancel
OK

Figure 32: Audacity - Audio Settings

3. Set the sample reate: At the bottom left of the main Audacity window, set both the Project Sample Rate and Default Sample Rate to 48000 to match the hardware requirements.
4. Increase buffer length if audio problems occur: If audio issues such as glitches, dropouts, or latency are encountered:
   - Open Edit → Preferences.
   - Increase the Buffer Length to a value greater than the default 100 ms (for example, 10000 ms is recommended).
   - A larger buffer allows more time for the system to process audio data, which can improve performance on embedded platforms, systems under high CPU load, or when using less optimized audio drivers.

Click OK to save the settings. Then, click the red circle button to begin recording.

To export the recording as an MP3, follow the steps outlined in the images below.

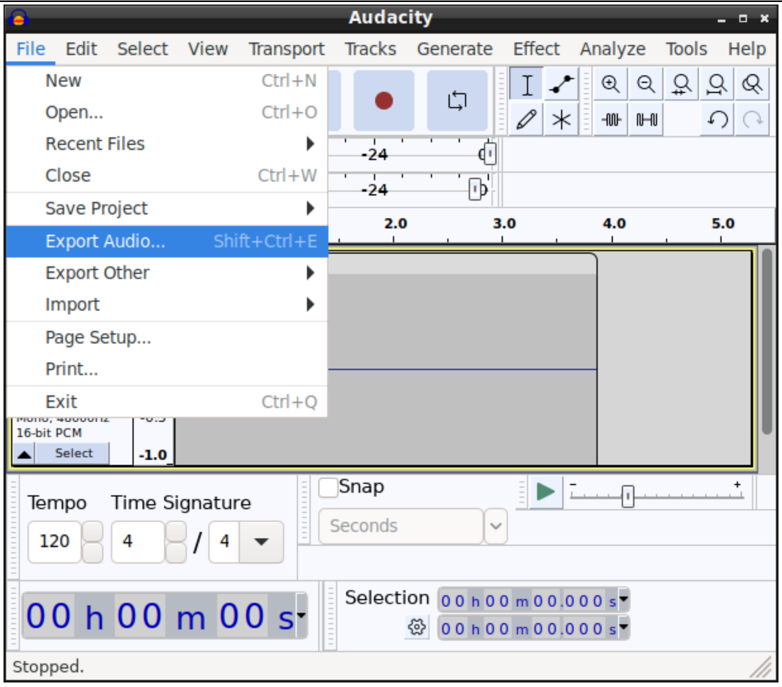
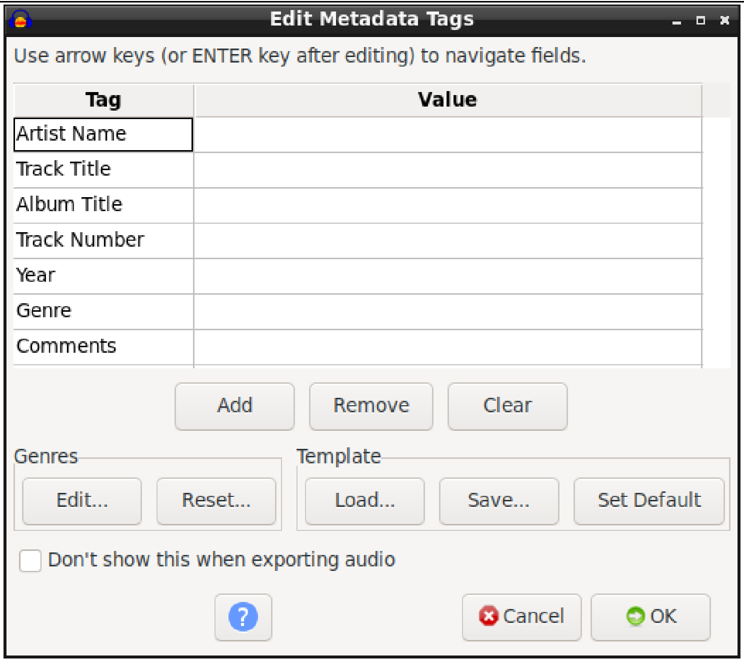


Figure 33. Audacity export video as MP3

Then, metadata can be filled for the audio as follows:



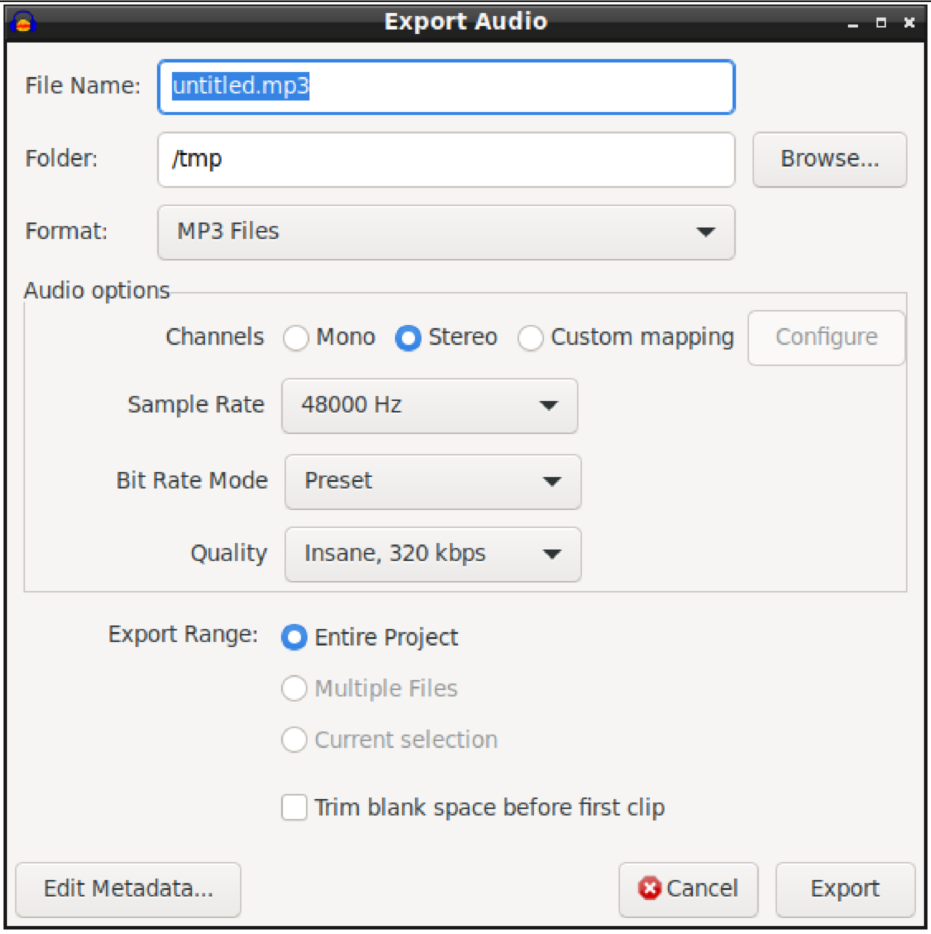
Use arrow keys (or ENTER key after editing) to navigate fields.

Tag	Value
Artist Name	
Track Title	
Album Title	
Track Number	
Year	
Genre	
Comments	

Figure 34. Audacity metadata tags to fill for the audio

Select OK to finish editing the metadata tags.

Once the audio file is edited, it can be renamed (e.g., song.mp3). Then, choose the desired directory and click Save to store the file.



Export Audio

File Name: untitled.mp3

Folder: /tmp

Format: MP3 Files

Audio options

Channels ○ Mono ● Stereo ○ Custom mapping

Sample Rate 48000 Hz

Bit Rate Mode Preset

Quality Insane, 320 kbps

Export Range: ● Entire Project

○ Multiple Files

○ Current selection

□ Trim blank space before first clip

Figure 35. Saving the recorded audio

8.2.1.3 VLC Media Player

VLC Media Player is a free and open-source multimedia player that supports a wide range of audio and video formats. To play music, simply open VLC and follow these steps:

1. Launch VLC Media Player

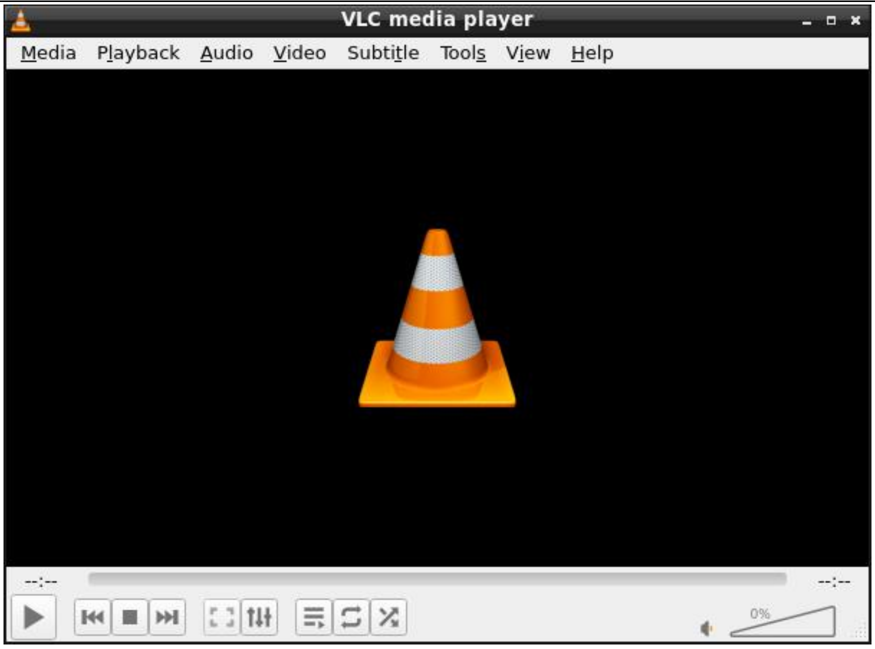


Figure 36. VLC Media Player

2. Click on “Media” in the top Menu, then select “Open File”

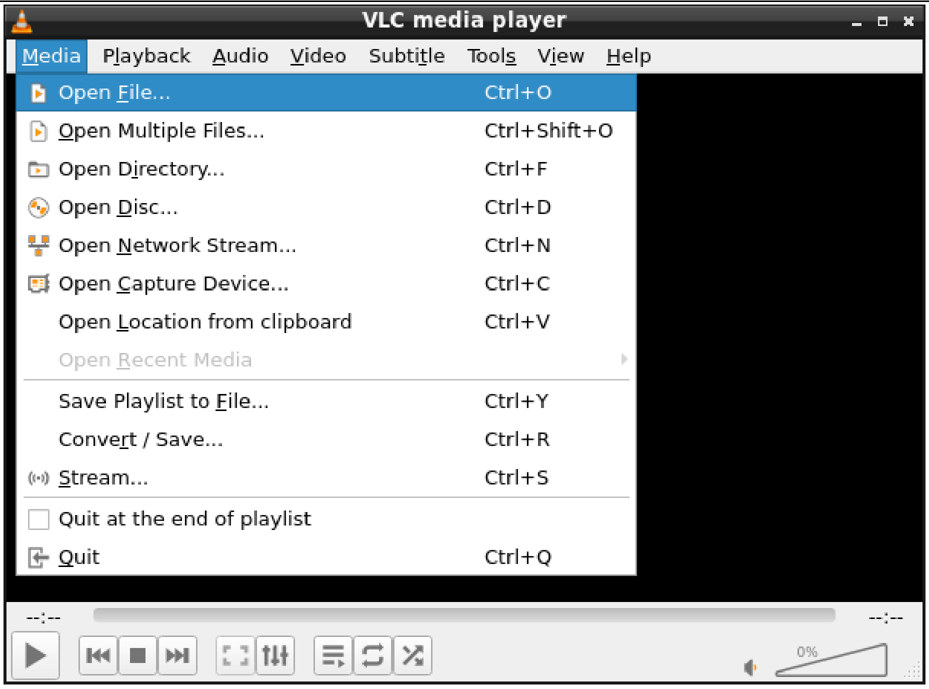
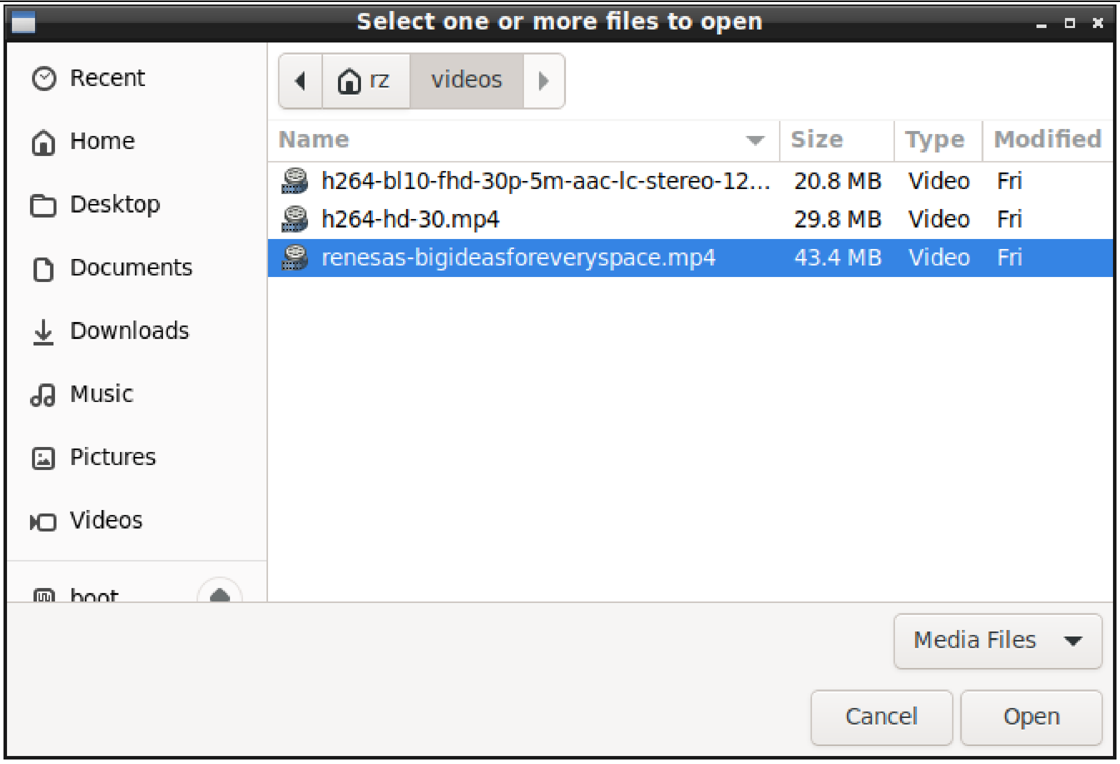

VLC media player
Media Playback Audio Video Subtitle Tools View Help
Open File... Ctrl+O
Open Multiple Files... Ctrl+Shift+O
Open Directory... Ctrl+F
Open Disc... Ctrl+D
Open Network Stream... Ctrl+N
Open Capture Device... Ctrl+C
Open Location from clipboard Ctrl+V
Open Recent Media
Save Playlist to File... Ctrl+Y
Convert / Save... Ctrl+R
Stream... Ctrl+S
Quit at the end of playlist
Quit Ctrl+Q
--:--
--:--
0%

Figure 37. Open file in VLC Media Player

3. Browse to the location of the MP3/MP4 file, select it, and click Open to start playing.



Name	Size	Type	Modified
h264-bl10-fhd-30p-5m-aac-lc-stereo-12...	20.8 MB	Video	Fri
h264-hd-30.mp4	29.8 MB	Video	Fri
renesas-bigideasforeveryspace.mp4	43.4 MB	Video	Fri

Figure 38. Select MP3/MP4 file to play in VLC Media Player

4. Now, the media can be played using VLC.

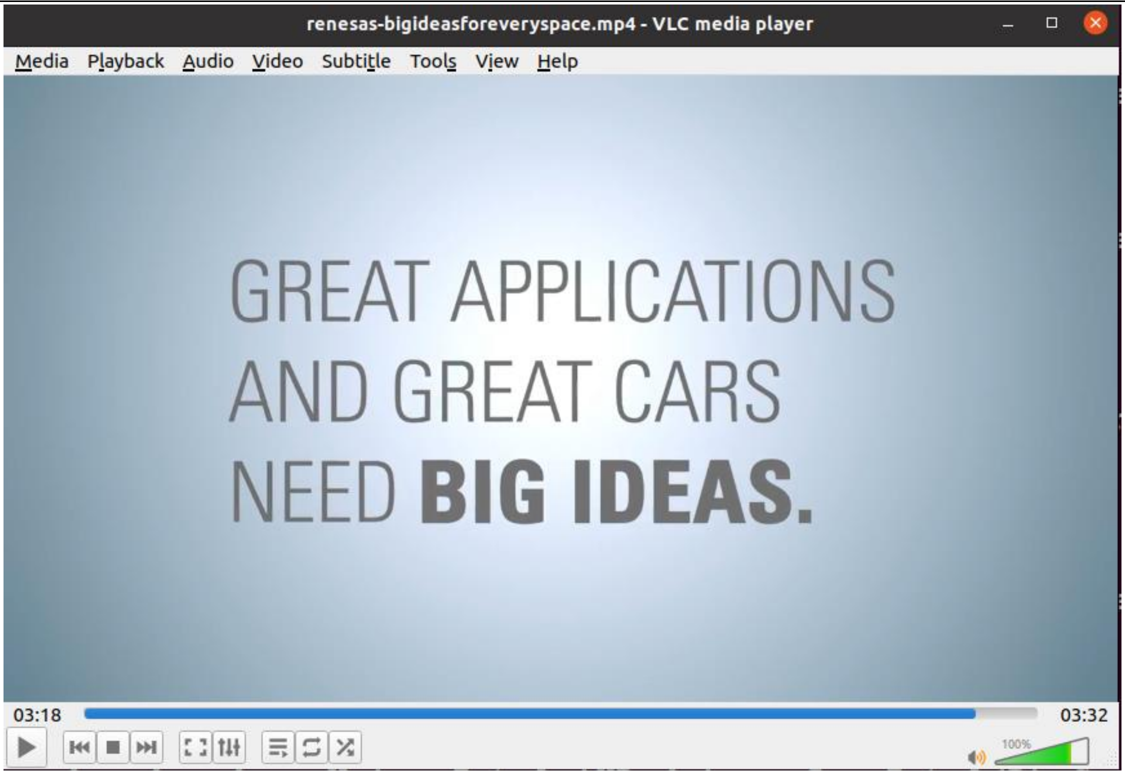

renesas-bigideasforeveryspace.mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
GREAT APPLICATIONS
AND GREAT CARS
NEED BIG IDEAS.
03:18
03:32
100%

Figure 39. Video playback in VLC Media Player

8.2.1.4 Using CSI Camera with VLC

CSI (Camera Serial Interface) is an interface standard used to connect cameras to a device, commonly used in embedded systems like Raspberry Pi and other single-board computers. It allows for high-speed data transfer between the camera and the system, enabling the capture of high-quality video and images.

You can use VLC Media Player to capture and view live videos from a CSI camera. Here's how you can do it:

1. Connect the Camera: Make sure your CSI camera is connected to the CSI port on your device.
2. Open VLC Media Player:
    - ○ Launch VLC from the application menu.

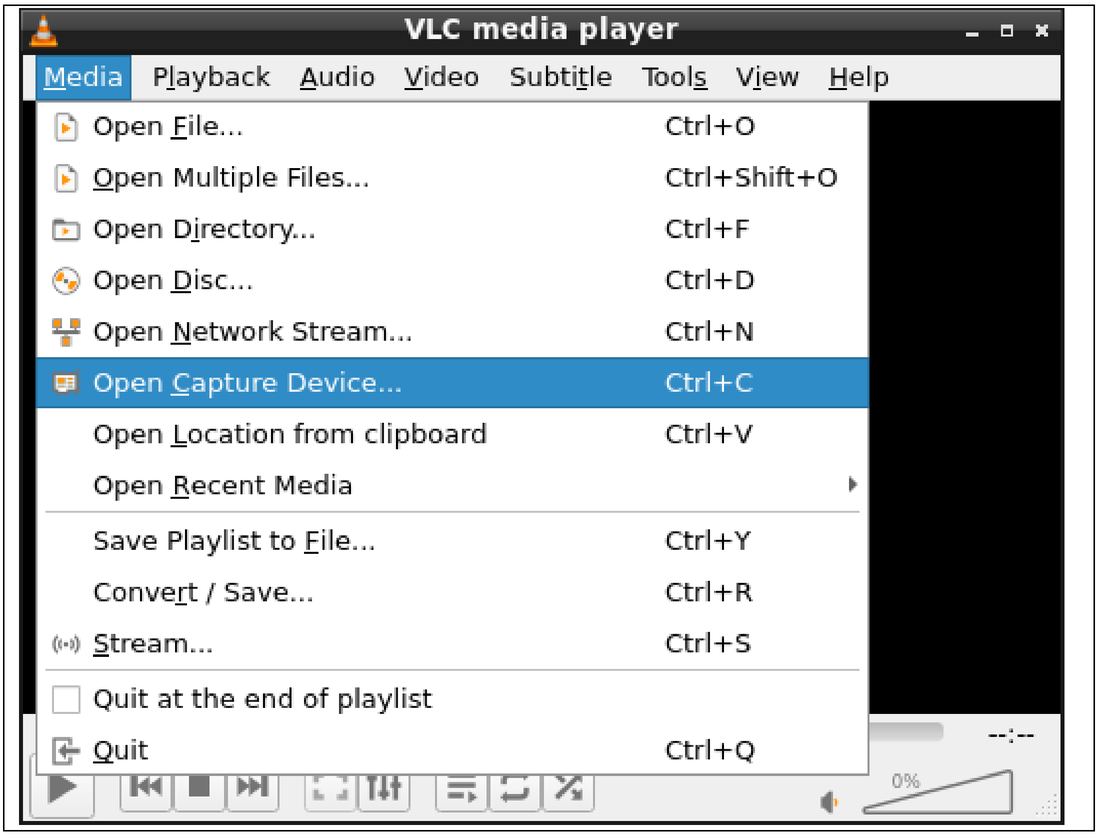


Figure 40. Capture Device in VLC Media Player

3. Open Capture Device:

- In VLC, click on the Media menu and select Open Capture Device....
- In the Capture Device tab, choose Video device name that corresponds to your CSI camera (it might be listed as /dev/video0 or something similar).

4. Configure the Capture Settings:

- Choose the desired video format (e.g., MJPEG or YUY2) and resolution (e.g., 640x480, 1280x720) based on your camera capabilities.

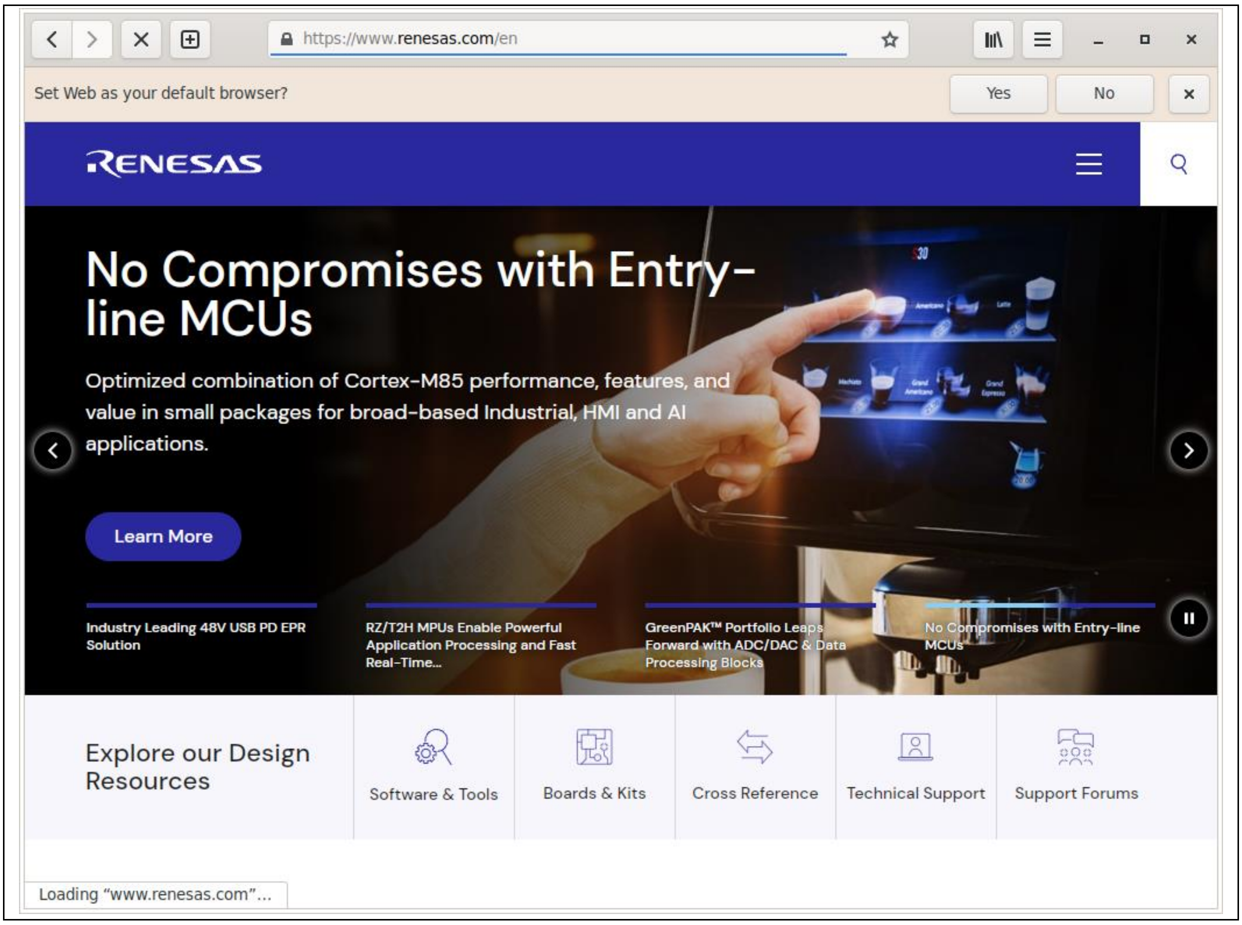
5. Click Play:

- Once you've selected the correct capture device and settings, click Play to start viewing the live video feed from your CSI camera.

Live video from the CSI camera should now be visible in VLC.

8.2.1.5 Web Browser

Ubuntu LXDE comes with a default web browser pre-installed. This browser provides essential features for browsing the internet and is lightweight, making it suitable for low-resource systems.



https://www.renesas.com/en

Set Web as your default browser?

No Compromises with Entry-line MCUs

Optimized combination of Cortex-M85 performance, features, and value in small packages for broad-based Industrial, HMI and AI applications.

Industry Leading 48V USB PD EPR Solution

RZ/T2H MPUs Enable Powerful Application Processing and Fast Real-Time...

GreenPAK™ Portfolio Leaps Forward with ADC/DAC & Data Processing Blocks

No Compromises with Entry-line MCUs

Explore our Design Resources

Loading "www.renesas.com"...

Figure 41. Ubuntu LXDE web browser pre-installed

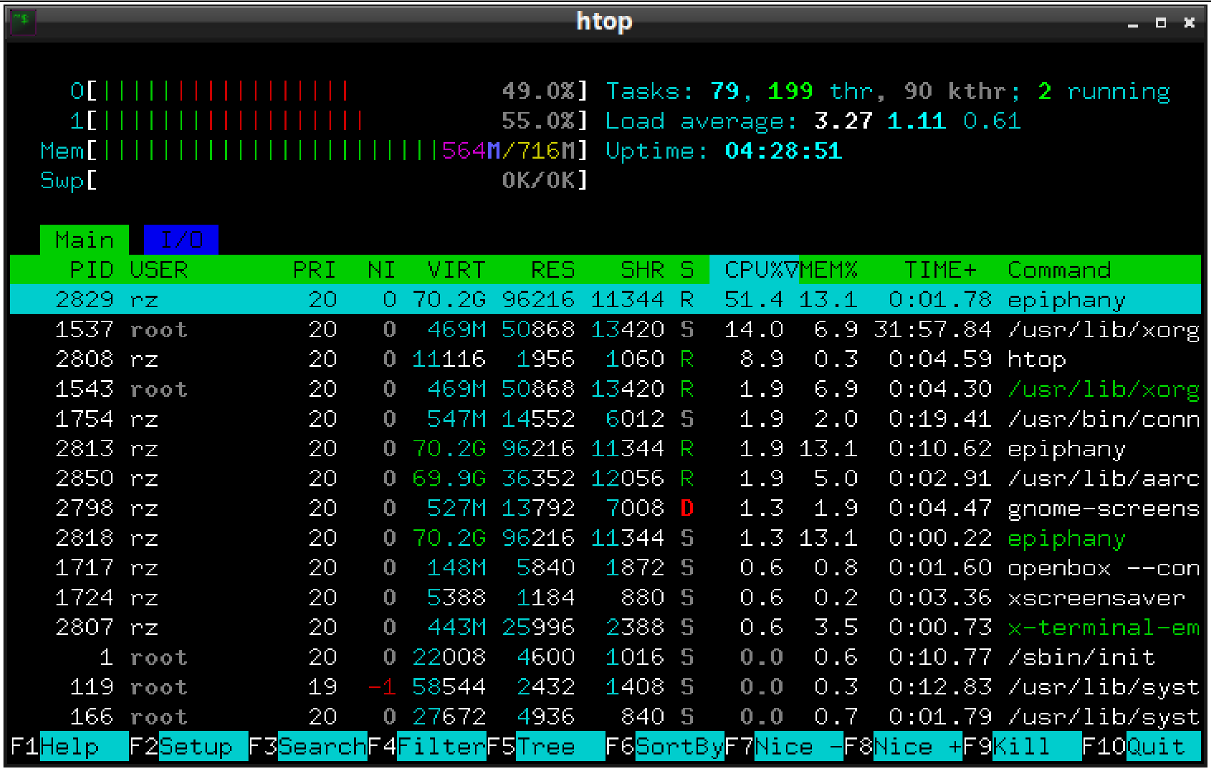
8.2.1.6 LXTerminal

LXTerminal is a VTE-based terminal emulator with support for multiple tabs. It is completely desktop-independent and does not have any unnecessary dependencies. In order to reduce memory usage and increase performance, all instances of the terminal share a single process.

Features:

- Lightweight and fast terminal emulator.
- Supports multiple tabs.
- Desktop-independent, reducing resource consumption.
- Optimized for performance with a single shared process for all instances.

Example Usage: Monitor Swap Memory with htop while browsing renesas.com



```
htop

    0[|||||||||||||||||                 49.0%]  Tasks: 79, 199 thr, 90 kthr; 2 running
    1[||||||||||||||||||                55.0%]  Load average: 3.27 1.11 0.61
  Mem[||||||||||||||||||||||||564M/716M]  Uptime: 04:28:51
  Swp[                                  0K/0K]

  Main  I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
   2829 rz          20   0 70.2G 96216 11344 R  51.4 13.1  0:01.78 epiphany
   1537 root        20   0  469M 50868 13420 S  14.0  6.9 31:57.84 /usr/lib/xorg
   2808 rz          20   0 11116  1956  1060 R   8.9  0.3  0:04.59 htop
   1543 root        20   0  469M 50868 13420 R   1.9  6.9  0:04.30 /usr/lib/xorg
   1754 rz          20   0  547M 14552  6012 S   1.9  2.0  0:19.41 /usr/bin/conn
   2813 rz          20   0 70.2G 96216 11344 R   1.9 13.1  0:10.62 epiphany
   2850 rz          20   0 69.9G 36352 12056 R   1.9  5.0  0:02.91 /usr/lib/aarc
   2798 rz          20   0  527M 13792  7008 D   1.3  1.9  0:04.47 gnome-screens
   2818 rz          20   0 70.2G 96216 11344 S   1.3 13.1  0:00.22 epiphany
   1717 rz          20   0  148M  5840  1872 S   0.6  0.8  0:01.60 openbox --con
   1724 rz          20   0  5388  1184   880 S   0.6  0.2  0:03.36 xscreensaver
   2807 rz          20   0  443M 25996  2388 S   0.6  3.5  0:00.73 x-terminal-em
      1 root        20   0 22008  4600  1016 S   0.0  0.6  0:10.77 /sbin/init
    119 root        19  -1 58544  2432  1408 S   0.0  0.3  0:12.83 /usr/lib/syst
    166 root        20   0 27672  4936   840 S   0.0  0.7  0:01.79 /usr/lib/syst
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 42. Htop in Ubuntu-LXDE

8.2.1.7 Ethernet

Ethernet, also known as a wired network, is a widely used method to connect a device to a Local Area Network (LAN) or the internet using physical cables. On Ubuntu LXDE, connecting to an Ethernet network can be easily done through the Network Manager, a powerful and user-friendly network management tool.

Follow these simple steps to connect to an Ethernet network using the Network Manager UI:

1. Open the Network Manager:

- At the bottom-right corner of the screen, click on the network icon, choose Edit connection....

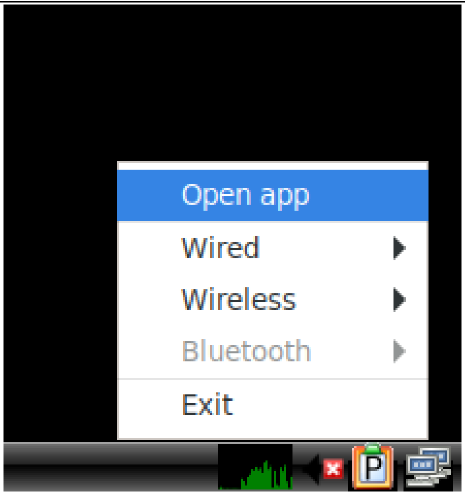
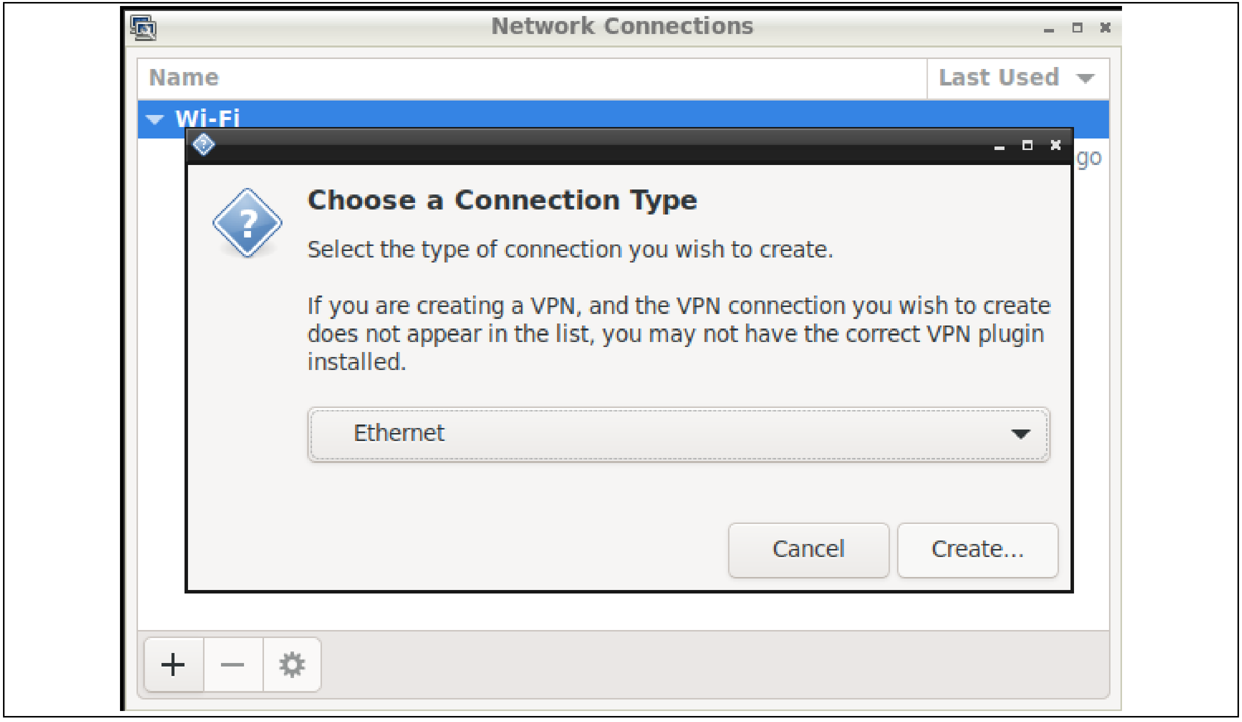


Figure 43. Bluetooth icon Ubuntu-LXDE taskbar

2. Choose Your Ethernet Network:

- In the Network Manager menu, you should see Wired Networks listed. Simply click on your Ethernet connection, or manually configure it as described below (if not automatically connected).



Choose a Connection Type

Select the type of connection you wish to create.

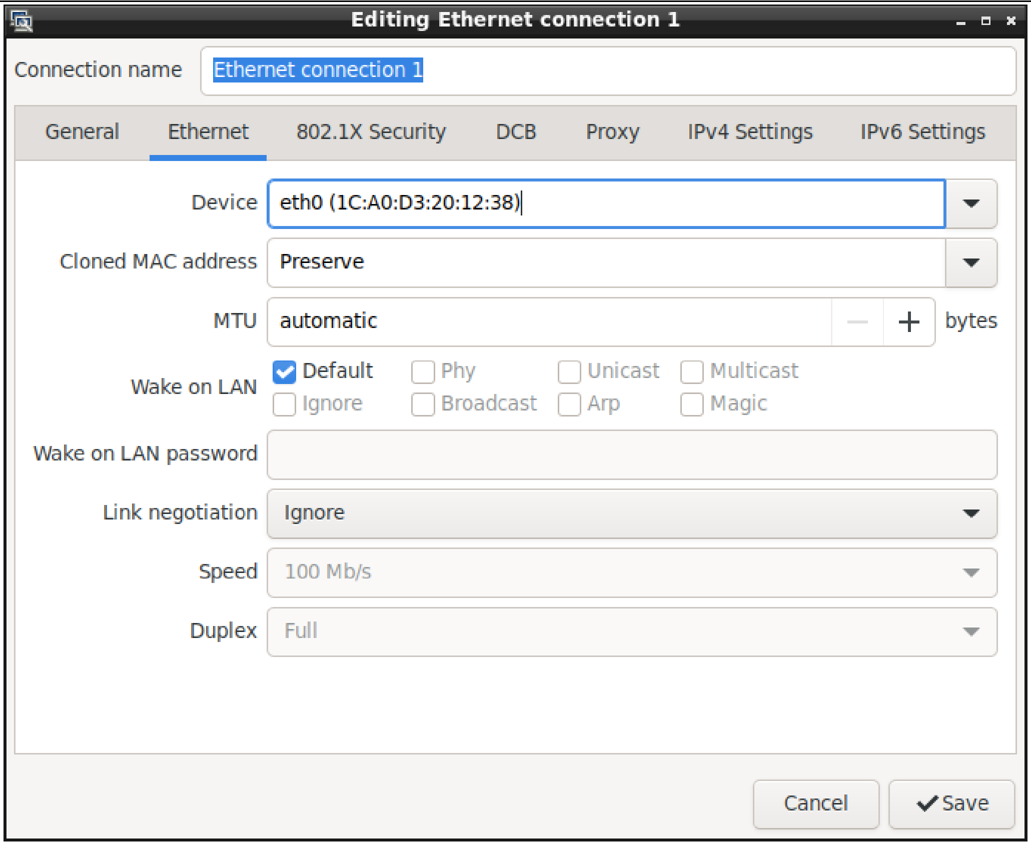
If you are creating a VPN, and the VPN connection you wish to create does not appear in the list, you may not have the correct VPN plugin installed.

Ethernet

Figure 44. Choose Network Connections

3. Configure the Connection:

- If the connection is not automatically established, you can configure network settings such as IP addresses, DNS servers, etc.



Editing Ethernet connection 1

Connection name: Ethernet connection 1

Device: eth0 (1C:A0:D3:20:12:38)

Cloned MAC address: Preserve

MTU: automatic bytes

Wake on LAN: ☑ Default □ Phy □ Unicast □ Multicast □ Ignore □ Broadcast □ Arp □ Magic

Wake on LAN password:

Link negotiation: Ignore

Speed: 100 Mb/s

Duplex: Full

Figure 45. Edit Ethernet connection 1

4. Connect: Once the connection settings are confirmed, the Ethernet connection should be ready for use. The network icon will update to indicate a successful connection.

8.2.1.8 Wi-Fi Network

Ubuntu LXDE provides an easy way to connect to Wi-Fi networks. Follow these simple steps to get connected:

1. Click on the Network Icon: In the lower-right corner of the screen, you will find the network icon. Click on this icon.

2. Choose Your Wi-Fi Network: A list of available Wi-Fi networks will appear. Find and click on your desired Wi-Fi network from the list.

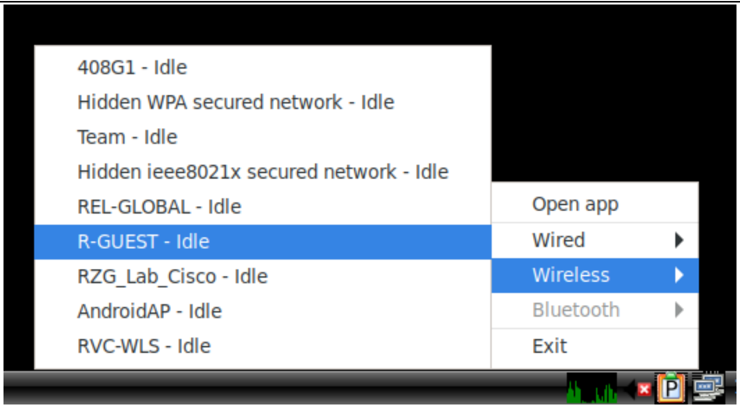


Figure 46. Wi-Fi selection in Ubuntu-LXDE

3. Enter the Password: After selecting the network, a prompt will appear asking for the Wi-Fi password. Type in the password and click Connect.
4. Connected: Once the password is verified, your system will be connected to the Wi-Fi network.

8.2.1.9 Bluetooth

Ubuntu LXDE provides an easy way to connect to Bluetooth devices. Follow these simple steps to get connected:

1. Open Bluetooth Manager: Click the LXDE icon in the lower-left corner of the screen, go to Preferences, and select Bluetooth Manager to access Bluetooth settings.

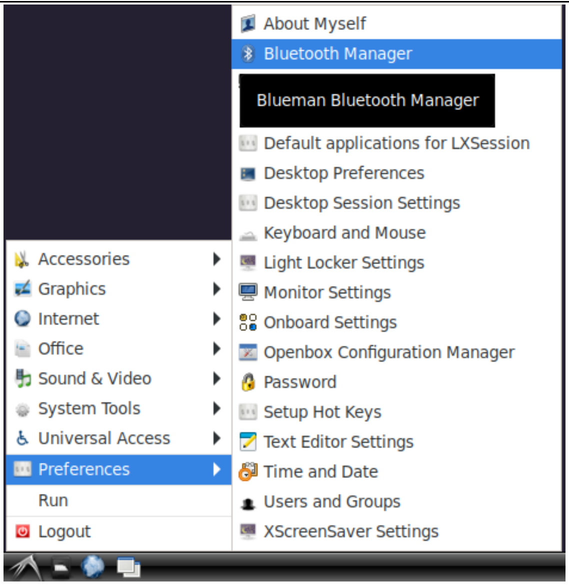


Figure 47. Bluetooth icon in Ubuntu LXDE taskbar

2. Enable Bluetooth: If Bluetooth is not already enabled, click the “Turn Bluetooth On” option to activate it.
3. Search for Devices: Select Adapter and click Search to view a list of available Bluetooth devices.

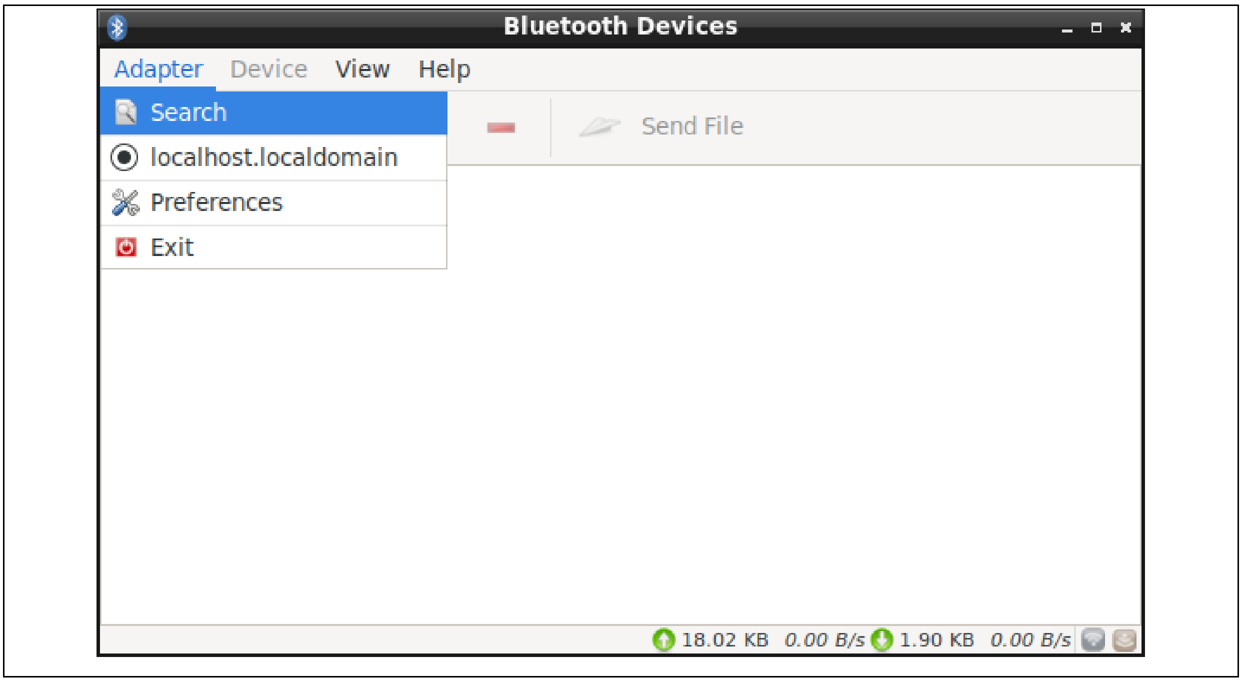
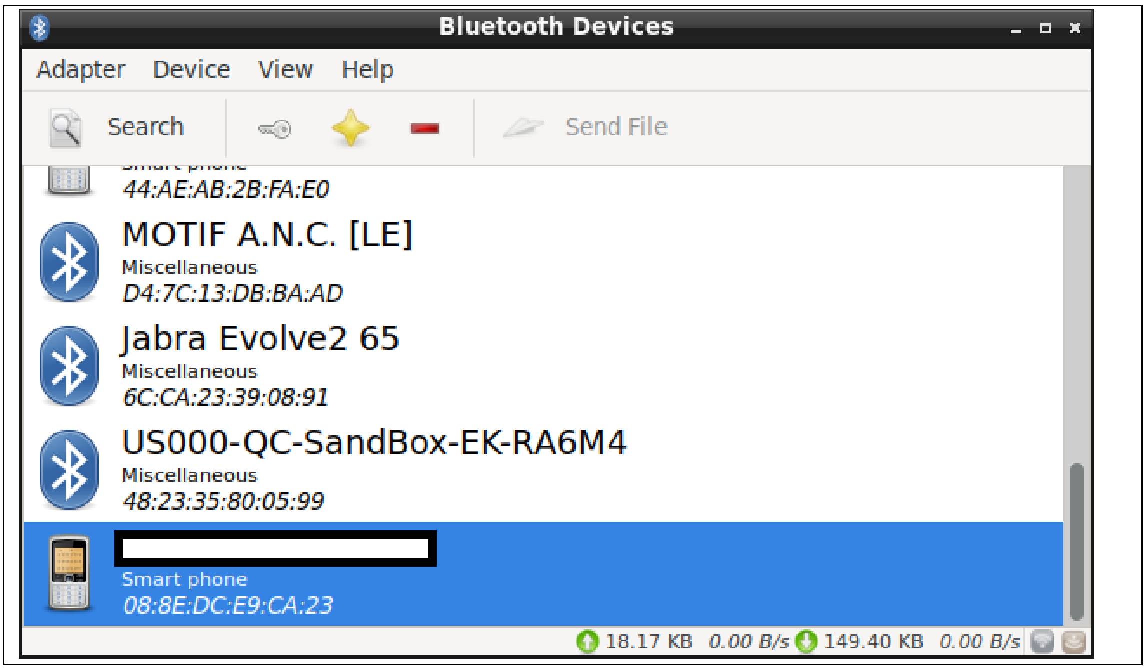


Figure 48. Search for available Bluetooth devices

4. Select the device: From the list of available Bluetooth devices, select the desired device to connect to.



44:AE:AB:2B:FA:E0

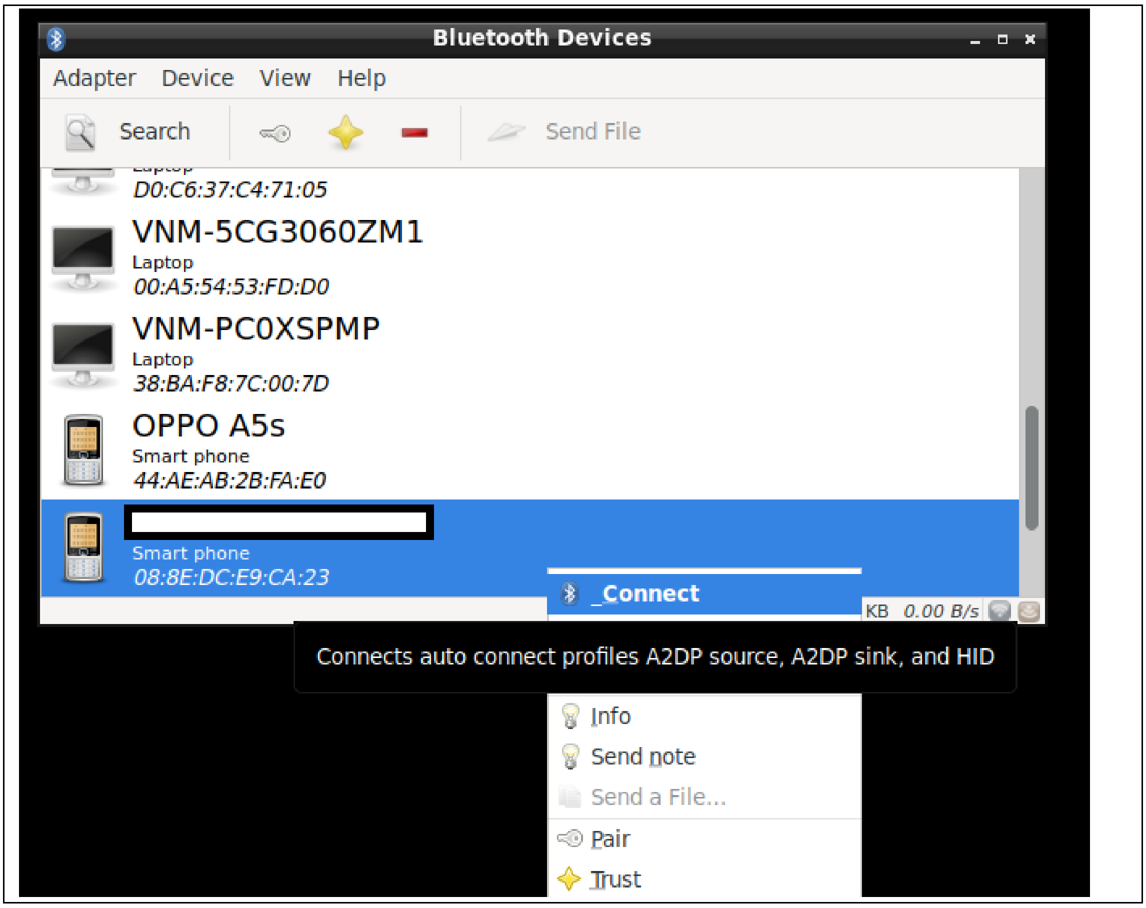
MOTIF A.N.C. [LE]
Miscellaneous
D4:7C:13:DB:BA:AD

Jabra Evolve2 65
Miscellaneous
6C:CA:23:39:08:91

US000-QC-SandBox-EK-RA6M4
Miscellaneous
48:23:35:80:05:99

[redacted]
Smart phone
08:8E:DC:E9:CA:23

Figure 49. Select a Bluetooth device



Laptop
D0:C6:37:C4:71:05

VNM-5CG3060ZM1
Laptop
00:A5:54:53:FD:D0

VNM-PC0XSPMP
Laptop
38:BA:F8:7C:00:7D

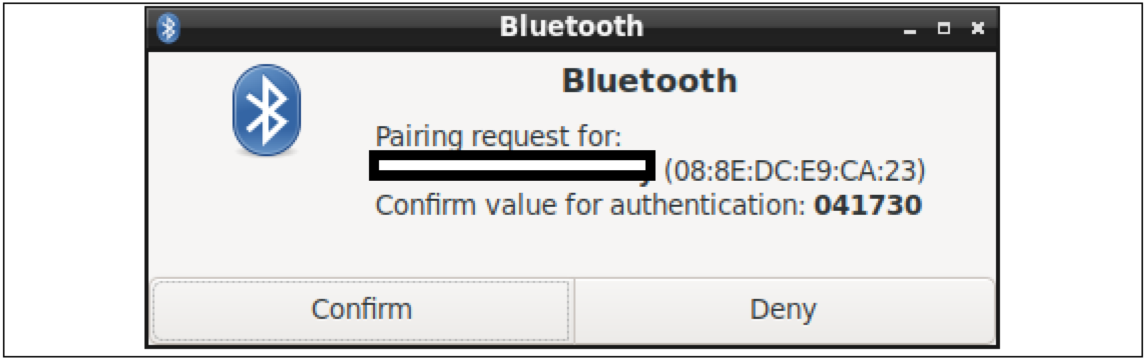
OPPO A5s
Smart phone
44:AE:AB:2B:FA:E0

[redacted]
Smart phone
08:8E:DC:E9:CA:23

Connects auto connect profiles A2DP source, A2DP sink, and HID

Figure 50. Connect to a Bluetooth device

5. Pair the Device: If prompted, confirm the pairing request and enter the required pairing code or PIN if necessary. After confirming, the devices will be paired.



Bluetooth

Pairing request for:
[redacted] (08:8E:DC:E9:CA:23)
Confirm value for authentication: **041730**

Figure 51. Bluetooth connection confirmation

6. Connection Established: Once the pairing process is complete, the device will be successfully connected.

8.2.2 Accessing Supported Features in Ubuntu Core

Ubuntu Core provides similar feature support to Yocto-based images, offering a headless environment for command-line operations. Feature usage and functionality align closely with those available in Yocto images. For details on supported features and their usage, refer to 9.1 Supported Features in Yocto Images

8.2.2.1 Configure the Network in Ubuntu Core

The Ubuntu installer configures the system to obtain network settings via DHCP by default. To switch to a static IP address, modify the network configuration using Netplan. The configuration file /etc/network/interfaces is no longer used. Instead, edit /etc/netplan/00-installer-config.yaml to set a static IP address. For example, the following configuration assigns the IP address 192.168.0.100 and specifies the DNS servers 8.8.4.4 and 8.8.8.8.

To open the network configuration file, use:

```
root@localhost:~# sudo vi /etc/netplan/00-installer-config.yaml
```

After installation, the system uses DHCP, and the network configuration file appears as follows:

```
# This is the network config written by 'subiquity'

network:
  ethernets:
    ens33:
      dhcp4: true
  version: 2
```

To assign a static IP address (192.168.0.100), modify the file as follows:

```
# This file describes the network interfaces available on your system

# For more information, see netplan(5).
network:
 version: 2
 renderer: networkd
 ethernets:
   ens33:
     dhcp4: no
     dhcp6: no
     addresses: [192.168.0.100/24]
     routes:
      - to: default
        via: 192.168.0.1
     nameservers:
       addresses: [8.8.8.8,8.8.4.4]
```

Then the hosts file needs to be updated to reflect the new hostname and IP address:

```
root@localhost:~# sudo vi /etc/hosts
```

Modify the file by adding the following entries:

```
127.0.0.1 localhost

192.168.0.100 rz-cmn.example.com rz-cmn

# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Next, change the hostname, run the following commands:

```
root@localhost:~# sudo echo rz > /etc/hostname
root@localhost:~# sudo hostname rz
```

The first command updates /etc/hostname, which is read during boot. The second command applies the change immediately without requiring a reboot.

As an alternative to the two commands above. Instead of manually updating the hostname file, the hostnamectl command (part of systemd) can be used:

```
root@localhost:~# sudo hostnamectl set-hostname rz
```

Afterward, run:

```
root@localhost:~# hostname
root@localhost:~# hostname -f
```

The first command returns the short hostname, while the second command shows the fully qualified domain name:

```
root@localhost:/home/root# hostname
localhost

root@localhost:/home/root# hostname -f
localhost.example.com

root@localhost:/home/root#
```

9. Network Boot and TFTP

This section outlines the process for network booting using TFTP (Trivial File Transfer Protocol). It includes configuration steps and commands necessary for a successful setup.

Network booting allows devices to boot from an image stored on a network server rather than relying on local storage. In this setup, the MMC SD card is not required for booting Linux.

9.1 TFTP Server Setup

This subsection covers the setup of a TFTP server on the host side. This is necessary for the device to retrieve the boot images over the network. We use an x86-based pc for this, but the instructions are the same for any server.

Prerequisites:

- x86 based PC or rack server (non x86 systems can be used at user discretion).
- An Ubuntu / Debian-based OS loaded onto the server.
- A router that performs DHCP.
- RZ/G2L-SBC, RZ/G2L-EVK, RZ/V2L-EVK, or RZ/V2H-EVK - depending on the board being used.
- FTDI-based USB-to-UART cable, or a USB Type-A to Micro USB cable (depending on the board’s UART interface).
- Ethernet cables.

Note: This requires a wired connection and cannot be performed on Wi-Fi.

1. Install a TFTP server using the following command:

```
$ sudo apt update
$ sudo apt install tftpd-hpa
```

2. Create a TFTP directory and set the appropriate permissions.

```
$ sudo mkdir /tftpboot
$ sudo chmod 755 /tftpboot
```

3. Edit the TFTP configuration file (typically found at /etc/default/tftpd-hpa) and set it up as follows:

```
$ vi /etc/default/tftpd-hpa
```

Paste the following content into the file.

```
# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"/tftpd-hpa
```

4. Restart the TFTP service to apply the changes.

```
$ sudo systemctl restart tftpd-hpa
```

Make sure the tftpd-hpa service is running:

```
$ sudo systemctl status tftpd-hpa
```

9.2 NFS Server Setup

NFS (Network File System) is a protocol that allows clients to access files over a network as if they were local. It enables multiple clients to share files from a central server, simplifying file management across machines.

In this setup, NFS will share the root file system (rootfs) with clients booting over the network. This allows client devices to dynamically retrieve their operating system files and configurations, making it ideal for embedded systems that require consistent file access without local storage.

1. Install the NFS server and NFS client packages if it is not already installed on your host PC:

```
$ sudo apt update
$ sudo apt install nfs-kernel-server nfs-common
```

2. Edit the /etc/exports file to specify the directories to be shared and their access permissions.

```
$ vi /etc/exports
```

For example, to share the /tftpboot directory, add the following line:

```
/tftpboot *(rw,no_root_squash,async)
```

Here, * allows access from any client. Consider replacing it with specific client IP addresses for better security.

3. After editing /etc/exports, run the following command to export the directories:

```
$ sudo exportfs -a
```

4. Start the NFS server and enable it to run at boot:

```
$ sudo systemctl start nfs-kernel-server
$ sudo systemctl enable nfs-kernel-server
```

9.3 U-Boot DHCP IP Configuration

In this subsection, the U-Boot environment will be configured for network settings, including the specification of the Ethernet device and the configuration of the server and device IP addresses.

1. Enter the U-Boot interactive command prompt for configuration by pressing any key when prompted with **Hit any key to stop autoboot**:

```
U-Boot 2021.10 (May 24 2024 - 07:26:08 +0000)

CPU:   Renesas Electronics CPU rev 1.0
Model: <Detected Board Model>
DRAM:  896 MiB
MMC:   sd@11c00000: 0
Loading Environment from SPIFlash... SF: Detected is25wp256 with page size 256
Bytes, erase size 4 KiB, total 32 MiB

In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
Net:   <Detected Ethernet Interface>
Hit any key to stop autoboot:  0
=>
```

2. Enter Specify the Ethernet device (eth1) to use for the network connection.
For example:

```
=> setenv ethact ethernet@11c30000
```

3. Configure server and device IPs:

```
=> setenv serverip <server_ip>
=> setenv ipaddr <device_ip>
```

For example:

```
=> setenv serverip 192.168.5.86
=> setenv ipaddr 192.168.5.30
```

9.4 -TFPT Boot

In this subsection, the boot arguments and commands for U-Boot will be configured to load the kernel image and device tree from the TFTP server.

Ensure all hardware connections are properly set up, as shown in Figure 52. TFTP boot setup. While the TFTP boot setup is shown using the RZ/G2L-SBC, the same procedure applies to other supported boards.

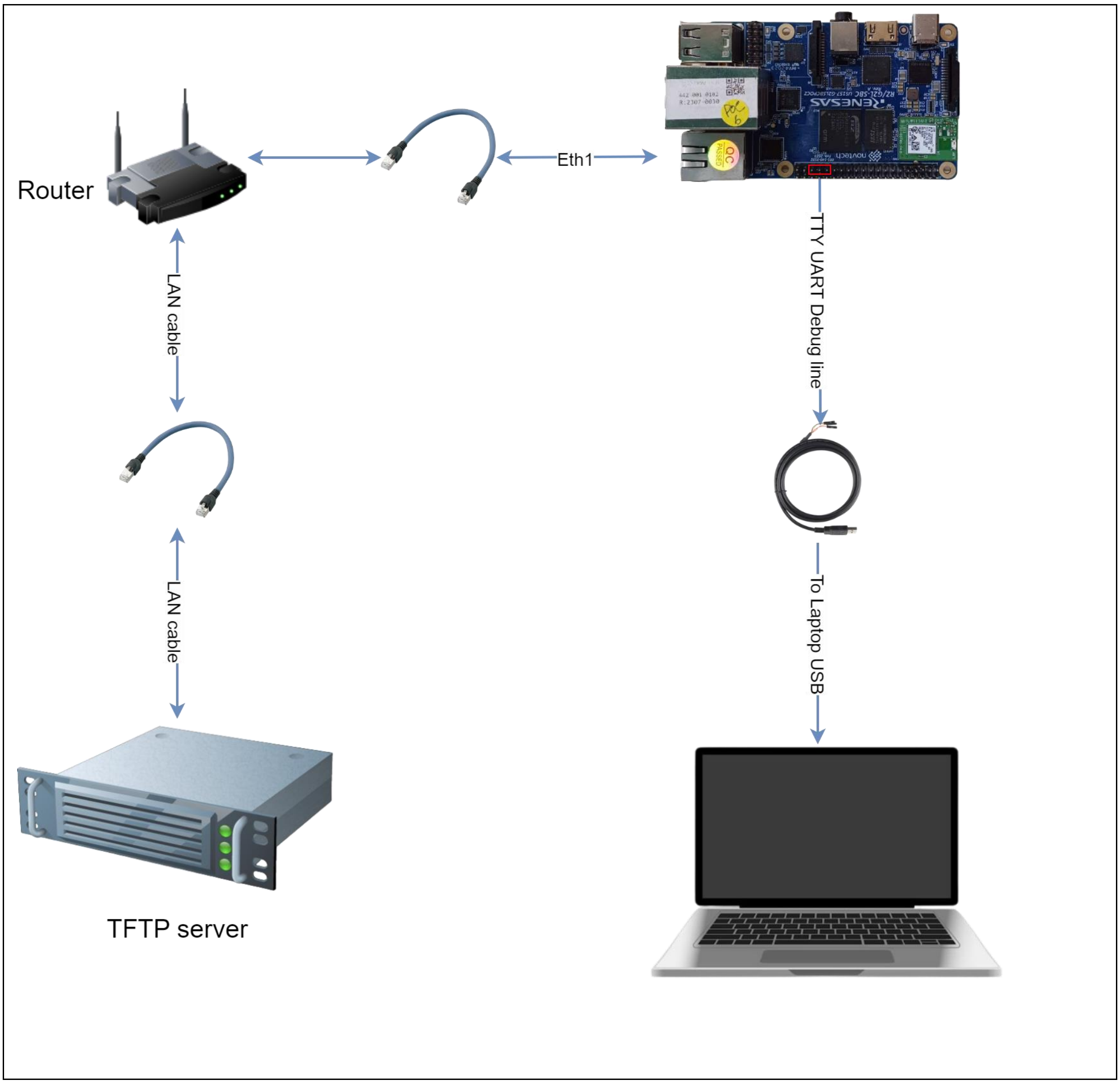


Figure 52. TFTP boot setup for RZ/G2L-SBC

1. After setting up the TFTP server and ensuring the hardware connections are correct, place the required boot images, such as the kernel image, device tree blob (DTB), device tree overlay (DTBO), and root file system in the TFTP directory (/tftpboot). These files will be loaded during the boot process.

```
renesas@builder-pc:/tftpboot/rz-cmn/$ tree -L 2
.
├── Image
├── overlays
│   ├── rzg2l-sbc-can.dtbo
│   ├── rzg2l-sbc-dsi.dtbo
│   ├── rzg2l-sbc-ext-i2c.dtbo
│   ├── rzg2l-sbc-ext-spi.dtbo
│   └── rzg2l-sbc-ov5640.dtbo
├── rootfs
│   ├── bin -> usr/bin
│   ├── boot
│   ├── dev
│   ├── etc
│   ├── home
│   ├── lib -> usr/lib
│   ├── media
│   ├── mnt
│   ├── opt
│   ├── proc
│   ├── root
│   ├── run
│   ├── sbin -> usr/sbin
│   ├── snap
│   ├── srv
│   ├── sys
│   ├── tmp
│   ├── usr
│   └── var
└── rzg2l-sbc.dtb
```

2. Define the boot arguments to specify the network and root file system settings:

```
=> setenv bootargs 'consoleblank=0 strict-devmem=0
ip=<device_ip>:<server_ip>::::<eth_device> root=/dev/nfs rw
nfsroot=<server_ip>:</path/to/your/rootfs>,v3,tcp'
```

For example:

```
=> setenv setenv bootargs 'consoleblank=0 strict-devmem=0
ip=192.168.5.30:192.168.5.86::::eth1 root=/dev/nfs rw
nfsroot=192.168.5.86:/tftpboot/rz-cmn/rootfs,v3,tcp'
```

3. Configure the boot command to load the kernel image and device tree files.

```
=> setenv bootcmd 'tftp <load_address_kernel> <path/to/kernel_image>; tftp
<load_address_dtb> <path/to/device_tree_blob>; tftp <load_address_dtbo>
<path/to/dtbo file>; booti <load_address_kernel> - <load_address_dtb> -
<load_address_dtbo>'
```

For example, load '**Image'**, **'rzg2l-sbc.dtb'** and '**rzg2l-sbc-ext-spi.dtbo'** files.

```
=> setenv bootcmd 'tftp 0x48080000 rz-cmn/Image; tftp 0x48000000 rz-cmn/rzg2l-
sbc.dtb; tftp 0x48010000 rz-cmn/overlays/rzg2l-sbc-ext-spi.dtbo; booti 0x48080000
- 0x48000000 - 0x48010000'
```

4. Save the changes to the environment variables so they persist across reboots:

```
=> saveenv
```

5. Initiate the boot progress by running **bootcmd**:

```
=> run bootcmd
```

If everything is set up correctly, the images will be booted from the network.

```
=> run bootcmd
Using ethernet@11c30000 device
TFTP from server 192.168.5.86; our IP address is 192.168.5.30
Filename rz-cmn/Image'.
Load address: 0x48080000
Loading: #################################################################
         #################################################################
         #################################################################
         19.6 MiB/s
done
Bytes transferred = 18035200 (1133200 hex)
Using ethernet@11c30000 device
TFTP from server 192.168.5.86; our IP address is 192.168.5.30
Filename rz-cmn/rzg2l-sbc.dtb'.
Load address: 0x48000000
Loading: ####
         8.6 MiB/s
done
Bytes transferred = 44855 (af37 hex)
Using ethernet@11c30000 device
TFTP from server 192.168.5.86; our IP address is 192.168.5.30
Filename rz-cmn/overlays/rzg2l-sbc-ext-spi.dtbo'.
Load address: 0x48010000
Loading: #
         455.1 KiB/s
done
Bytes transferred = 932 (3a4 hex)
Moving Image from 0x48080000 to 0x48200000, end=493a0000
## Flattened Device Tree blob at 48000000
   Booting using the fdt blob at 0x48000000
   Loading Device Tree to 000000007bf1a000, end 000000007bf27f36 ... OK

Starting kernel ...
...
```

10. Using SSH and SCP for Remote Access and File Transfers

This section explains how to use SSH (Secure Shell) for secure remote access to the RZ/G2L-SBC and how to utilize SCP (Secure Copy Protocol) for file transfers. By default, OpenSSH is employed as it is a feature-rich and widely used SSH implementation that offers advanced capabilities for secure communication. While OpenSSH serves as the default option, Dropbear SSH can be considered for lightweight, resource-constrained environments, making it particularly suitable for embedded systems.

10.1 Differences Between Dropbear and OpenSSH

- **Resource Usage**: Dropbear is optimized for lower resource usage, making it ideal for embedded systems.
- **Feature Set**: OpenSSH has a more extensive feature set, including advanced options for authentication and configuration.
- **Key Authentication**: OpenSSH requires the use of SSH keys for authentication, while Dropbear can operate with both keys and passwords.

10.2 Using OpenSSH

OpenSSH is a widely used, full-featured SSH implementation that provides encrypted communication between hosts. It supports advanced authentication methods and secure remote administration, making it ideal for robust network security.

The RZ/G2L-SBC supports both password and key-based authentication methods. To enhance security by enforcing SSH key-based login, follow these steps to switch to key-based authentication:

1. Generate an SSH key pair on the local machine. Run the following command to generate a secure SSH key pair:

```
$ ssh-keygen -t rsa -b 4096
```

2. Copying an SSH public Key to the board using SSH, transfer your public key to the board with this command:

```
$ cat ~/.ssh/id_rsa.pub | ssh username@remote_host "mkdir -p ~/.ssh && cat >> 
~/.ssh/authorized_keys"
```

For example:

```
$ cat ~/.ssh/id_rsa.pub | ssh root@192.168.5.30 "mkdir -p ~/.ssh && cat >> 
~/.ssh/authorized_keys"
```

3. Authenticate using SSH keys:

```
$ ssh root@192.168.5.30
```

If this is the first time connecting to this host (as mentioned in the previous method), a message like the following may appear:

```
$ The authenticity of host 192.169.5.30 (192.168.5.30)' can't be established.
ED25519 key fingerprint is SHA256:esQPI0Ip9HZH9A6dvTsA9+k7eLjT4sqzpiF7znl0tyw.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

This indicates that the local computer does not recognize the remote host. Type **yes** and press **ENTER key** to proceed.

Step 4: Disable password authentication. If logging in to your account using SSH is successful without a password, SSH key-based authentication has been correctly configured. However, password-based authentication remains active, which leaves the server vulnerable to brute-force attacks.

Once the SSH connection is established, open the SSH daemon's configuration file:

```
$ vi /etc/ssh/sshd_config
```

Inside the file, search for a directive called **PasswordAuthentication**. This may be commented out. Uncomment the line by removing any '#' at the beginning of the line, and set the value to **no**. This will disable the ability to log in through SSH using account passwords: /etc/ssh/sshd

```
PasswordAuthentication no
```

Step 5: Restart the SSH service to apply the changes:

```
$ systemctl restart ssh
```

10.3 SSH Access

After configuring the authentication key, access to the RZ/G2L-SBC via SSH can be achieved using various tools available on both Windows and Linux platforms.

10.3.1 SSH from Windows Host

1. Using Git Bash.
   - Install Git for Windows if you have not already.
   - Open and use the following command:

   ```
   $ ssh username@<device_ip>
   ```

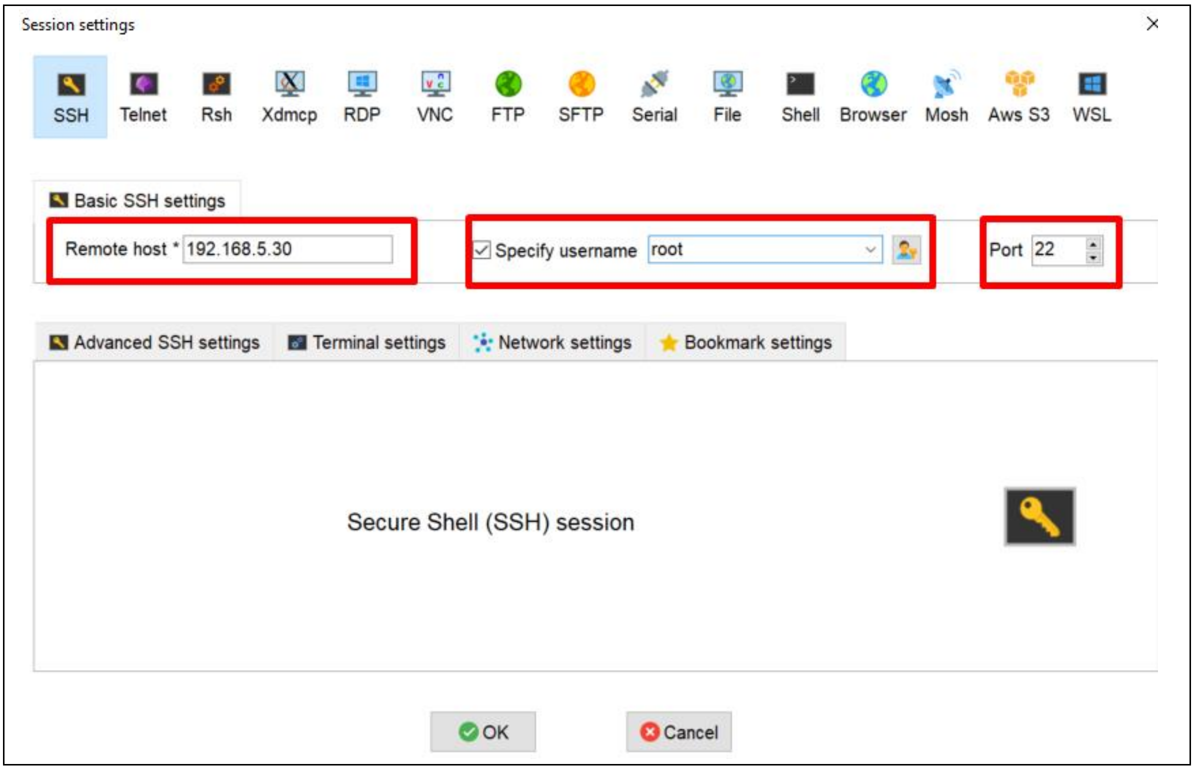
   For example:

   ```
   $ ssh root@192.168.5.30
   ```

   - Type **yes** to confirm the host's authenticity when prompted.

   ```
   $ ssh root@192.168.5.30
   The authenticity of host '192.168.5.30 (192.168.5.30)' can't be established.
   RSA key fingerprint is SHA256:v39PhjNp4F7HcQpwJmfNOYcC+ZZ3Yw8i1ICsL2mXUgg.
   This key is not known by any other names.
   Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
   Warning: Permanently added '192.168.5.30' (RSA) to the list of known hosts.
   ```

2. Use MobaXTerm.
   - Download and install MobaXterm if you have not already.
   - Select "Session" > "SSH" and enter the device's IP address



Remote host * 192.168.5.30

☑ Specify username root

Port 22

Figure 53. SSH settings in Mobaxterm

- Click 'OK' to save the setting.
- Click on the session to initiate an SSH connection.

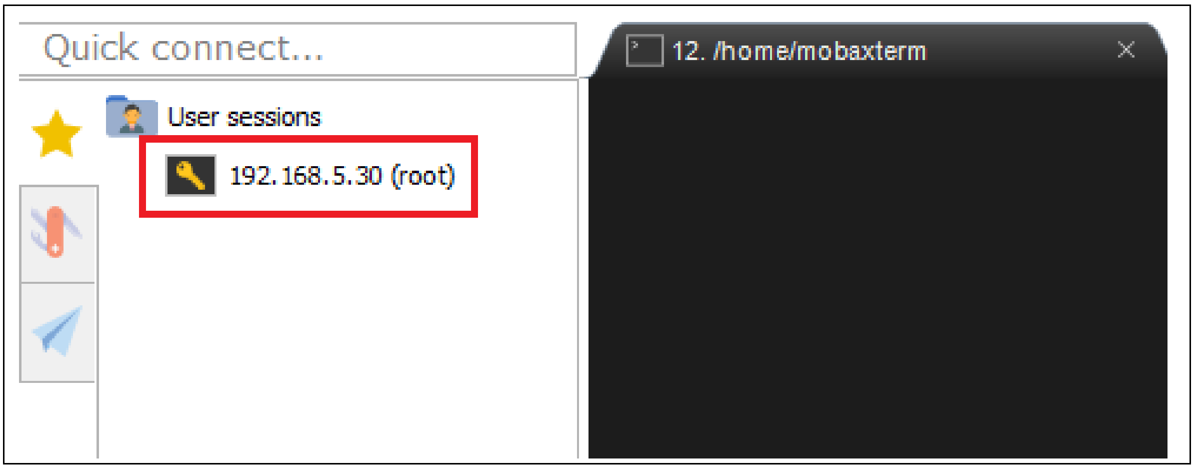


Figure 54. Connect to SSH session in Mobaxterm

10.3.2 SSH from Linux Host

1. Open a terminal and run.

```
$ ssh username@<device_ip>
```

For example:

```
$ ssh root@192.168.5.30
```

2. Type **yes** to confirm the host's authenticity when prompted.

```
$ ssh root@192.168.5.30
The authenticity of host '192.168.5.30 (192.168.5.30)' can't be established.
RSA key fingerprint is SHA256:v39PhjNp4F7HcQpwJmfNOYcC+ZZ3Yw8i1ICsL2mXUgg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.5.30' (RSA) to the list of known hosts.
```

10.4 SCP (Secure copy protocol)

To securely transfer files between local and remote systems, SCP can be used on both Windows and Linux.

10.4.1 SCP from Windows Host

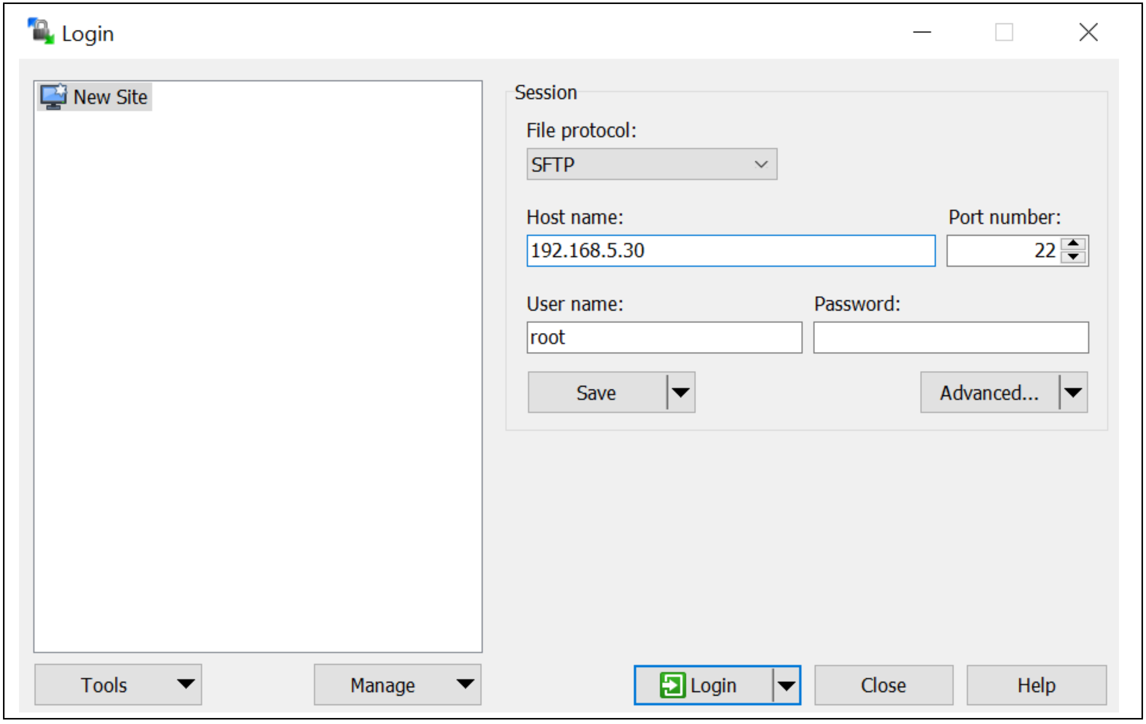
1. Using Git bash:
   - ○ Install Git for Windows if you have not already.
   - ○ Use the following command:

     ```
     $ scp <local_file> username@<device_ip>:<remote_path>
     ```

     For example:

     ```
     $ scp hello-world root@192.168.5.30:home/root
     ```

   - ○ Type **yes** to confirm the host's authenticity when prompted.
2. Use WinSCP.
   - ○ Open WinSCP and select "New Session".
   - ○ Choose SCP as the protocol, then enter the remote device's IP address and the hostname.



File protocol: SFTP

Host name: 192.168.5.30

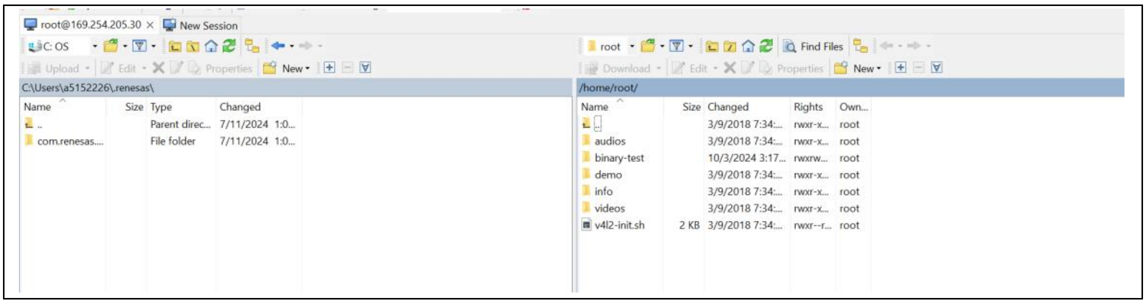
Port number: 22

User name: root

Password:

Figure 55. Setting up WinSCP for SSH session

   - ○ Click 'Login' and select yes to confirm the host's authenticity when prompted.
   - ○ Drag and drop files between your local machine (Left) and the target board (Right) to transfer.



C:\Users\a5152226\.renesas\

Name	Size	Type	Changed
..		Parent direc...	7/11/2024 1:0...
com.renesas....		File folder	7/11/2024 1:0...

/home/root/

Name	Size	Changed	Rights	Own...
..		3/9/2018 7:34:...	rwxr-x...	root
audios		3/9/2018 7:34:...	rwxr-x...	root
binary-test		10/3/2024 3:17...	rwxrw...	root
demo		3/9/2018 7:34:...	rwxr-x...	root
info		3/9/2018 7:34:...	rwxr-x...	root
videos		3/9/2018 7:34:...	rwxr-x...	root
v4l2-init.sh	2 KB	3/9/2018 7:34:...	rwxr--r...	root

Figure 56. Using WinSCP to transfer files

10.4.2 SCP from Linux Host

1. Open a terminal and run.

```
$ scp <local_file> username@<device_ip>:<remote_path>
```

For example:

```
$ scp hello-world root@192.168.5.30:/home/root
```

2. Type **yes** to confirm the host's authenticity when prompted.

```
$ scp hello-world root@192.168.5.30:/home/root
The authenticity of host '192.168.5.30 (192.168.5.30)' can't be established.
RSA key fingerprint is SHA256:v39PhjNp4F7HcQpwJmfNOYcC+ZZ3Yw8i1ICsL2mXUgg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.5.30' (RSA) to the list of known hosts.
```

11. Building the eSDK

The extensible SDK makes it easy to add new applications and libraries to an image, modify the source for an existing component, test changes on the RZ board, and ease integration into the rest of the OpenEmbedded Build System.

The eSDK build generates an installer, which you will use to install the eSDK on the same PC where your Yocto environment is set up.

In Section 6.1, the support script was copied from the release package, which helped commence the image build. The script can also support eSDK build, run the following command to start the build with specific images:

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ IMAGE=<target-images> ./rzsbc_builder.sh
build-sdk
```

For example, to build the core-image-weston eSDK:

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ IMAGE=core-image-
weston ./rzsbc_builder.sh build-sdk
```

To build the eSDK for all supported images, run the following command:

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ IMAGE=all-supported-
images ./rzsbc_builder.sh build-sdk
```

The resulting eSDK installer will be in `~/renesas/rz-cmn-srp/yocto_rzcmn_board/build/tmp/deploy/sdk`.

The eSDK installer will have the extension ".sh".

> Note: SDK build support is provided only for Yocto-based images. Ubuntu images are not associated with an SDK in the build system. For Ubuntu, the development environment must be set up manually using standard Ubuntu tools. The SDK generated by the build scripts applies exclusively to Yocto.

```
renesas@builder-pc:~/renesas/rz-cmn-srp/yocto_rzcmn_board/build/tmp/deploy/sdk$ ls
poky-glibc-x86_64-core-image-bsp-cortexa55-rz-cmn-toolchain-5.1.4.host.manifest
poky-glibc-x86_64-core-image-bsp-cortexa55-rz-cmn-toolchain-5.1.4.sh
poky-glibc-x86_64-core-image-bsp-cortexa55-rz-cmn-toolchain-5.1.4.target.manifest
poky-glibc-x86_64-core-image-bsp-cortexa55-rz-cmn-toolchain-5.1.4.testdata.json
poky-glibc-x86_64-core-image-minimal-cortexa55-rz-cmn-toolchain-5.1.4.host.manifest
poky-glibc-x86_64-core-image-minimal-cortexa55-rz-cmn-toolchain-5.1.4.sh
poky-glibc-x86_64-core-image-minimal-cortexa55-rz-cmn-toolchain-5.1.4.target.manifest
poky-glibc-x86_64-core-image-minimal-cortexa55-rz-cmn-toolchain-5.1.4.testdata.json
poky-glibc-x86_64-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.host.manifest
poky-glibc-x86_64-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh
poky-glibc-x86_64-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.target.manifest
poky-glibc-x86_64-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.testdata.json
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rz-cmn-toolchain-
5.1.4.host.manifest
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rz-cmn-toolchain-5.1.4.sh
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rz-cmn-toolchain-
5.1.4.target.manifest
```

```
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rz-cmn-toolchain-
5.1.4.testdata.json
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-
5.1.4.host.manifest
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-
5.1.4.testdata.json
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rz-cmn-toolchain-5.1.4.host.manifest
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rz-cmn-toolchain-5.1.4.sh
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rz-cmn-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rz-cmn-toolchain-5.1.4.testdata.json
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rz-cmn-toolchain-
5.1.4.host.manifest
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rz-cmn-toolchain-5.1.4.sh
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rz-cmn-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rz-cmn-toolchain-
5.1.4.testdata.json
```

Note: The SDK build may fail depending on the build environment. At that time, run the build again after a period of time.

12. Application Building, Packaging, and Running

The SDK allows you to develop and test custom applications for the RZ board on different systems. This section covers setting up your development environment and running your applications.

12.1 How to extract the eSDK

To get started, extract the eSDK and install the toolchain on your host PC. This step provides the necessary tools for cross compiling your applications.

To set up your environment:

1. Install the toolchain on a Host PC.

For example, to install the toolchain, run the following command.

```
renesas@builder-pc:~/renesas/rz-cmn-
srp/yocto_rzcmn_board$ sh ./build/tmp/deploy/sdk/poky-glibc-x86_64-core-image-
weston-cortexa55-rz-cmn-toolchain-5.1.4.sh
```

Note: You cannot install the eSDK as root because BitBake will not run with root privileges. Therefore, attempting to install the extensible SDK as root is counterproductive.

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

```
renesas@builder-pc:~/renesas/rz-cmn-srp/yocto_rzcmn_board
$ sh ./build/tmp/deploy/sdk/poky-glibc-x86_64-core-image-weston-cortexa55-rz-cmn-
toolchain-5.1.4.sh
Poky (Yocto Project Reference Distro) Extensible SDK installer version 5.1.4
==============================================================================
Enter target directory for SDK (default: ~/poky_sdk): ~/esdk/5.1.4
You are about to install the SDK to "/home/renesas/esdk/5.1.4". Proceed [Y/n]? Y
Extracting SDK.............done
Setting it up...
Extracting buildtools...
Preparing build system...
Parsing recipes: 100% |##########################################| Time: 0:00:52
Initialising tasks: 100% |#######################################| Time: 0:00:00
Checking sstate mirror object availability: 100% |###############| Time: 0:00:00
Loading cache: 100% |############################################| Time: 0:00:00
Initialising tasks: 100% |#######################################| Time: 0:00:00
done

SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
environment setup script e.g.
 $ . /home/renesas/esdk/5.1.4/environment-setup-cortexa55-poky-linux
```

2. Set up cross-compile environment. The following command assumes that you installed the SDK in `~/esdk/5.1.4`

```
renesas@builder-pc:~$ source ~/esdk/5.1.4/environment-setup-cortexa55-poky-linux
SDK environment now set up; additionally you may now run devtool to perform
development tasks.
Run devtool --help for further details.
```

Note: The user needs to run the above command once for each shell session. In addition, source' is a bash specific call. The POSIX convention is to use'. ~/esdk/5.1.4/environment-setup-cortexa55-poky-linux'. Bash equates 'source' to '.'.

To begin working with the Extensible Software Development Kit (eSDK) in Yocto, consult the official documentation provided by the Yocto Project. This guide offers comprehensive instructions on configuring and utilizing the eSDK effectively.

Access the official eSDK documentation by following this URL: Using the Extensible SDK.

12.2 Build a sample application using the eSDK with CMake

CMake is cross-platform, free, and open-source software for building automation, testing, packaging, and installation of software by using a compiler-independent method.

If the host PC does not have CMake installed, it can install using the following command:

```
renesas@builder-pc:~$ sudo apt-get install cmake
```

The following steps will include instructions for setting up the project, editing the CMakeLists.txt file, and performing the build and installation.

1. Create the project structure:

```
renesas@builder-pc:~$ mkdir ~/cmake_helloworld
renesas@builder-pc:~$ cd ~/cmake_helloworld
renesas@builder-pc:~/cmake_helloworld$ mkdir build src
```

2. Organize the project structure as shown below:

```
renesas@builder-pc:~/cmake_helloworld$ tree
.
├── build
├── CMakeLists.txt
└── src
    └── helloworld.c
```

`CMakeLists.txt` and `helloworld.c` will be created later.

3. Create your application.

```
renesas@builder-pc:~/cmake_helloworld$ vi src/helloworld.c
```

Then, copy the contents below to the file:

```
#include <stdio.h>

int main(int argc, char** argv)
{
    printf("\nHello World!\n");
    return 0;
}
```

4. Create CMake configuration file.

```
renesas@builder-pc:~/cmake_helloworld$ vi CMakeLists.txt
```

Edit the following configuration file to match the SDK paths, The eSDK installation as described in 13.1 How to extract the eSDK is a prerequisite for this operation.

```
cmake_minimum_required(VERSION 3.10)
project(HelloWorld C)

# Set the path to your C compiler
set(CMAKE_C_COMPILER /path/to/your/sdk/bin/gcc)

# Set the path to your C++ compiler (if needed)
set(CMAKE_CXX_COMPILER /path/to/your/sdk/bin/g++)

# Define the sysroot path for cross-compilation
set(CMAKE_SYSROOT /path/to/your/sysroot)

# Add the executable target “helloworld”
add_executable(helloworld src/helloworld.c)
```

For example, if the SDK is installed in `/home/renesas/esdk/5.1.4`, the completed configuration file will resemble the following:

```
cmake_minimum_required(VERSION 3.10)
project(HelloWorld C)

set(CMAKE_C_COMPILER
/home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-gcc)
set(CMAKE_CXX_COMPILER
/home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-g++)

# Sysroot path
set(CMAKE_SYSROOT /home/renesas/esdk/5.1.4/tmp/sysroots/rzg2l-sbc)

add_executable(helloworld src/helloworld.c)
```

5. Build the application.

```
renesas@builder-pc:~/cmake_helloworld$ cd build/
renesas@builder-pc:~/cmake_helloworld/build$ cmake ../
-- The C compiler identification is GNU 9.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/renesas/cmake_helloworld/build
renesas@builder-pc:~/cmake_helloworld/build$ cmake --build .
[ 50%] Building C object CMakeFiles/hello.dir/src/helloworld.c.o
[100%] Linking C executable helloworld
[100%] Built target helloworld
```

After completing, confirm that the execute application `helloworld` is generated in the build folder.

```
renesas@builder-pc:~/cmake_helloworld/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  CPackConfig.cmake
CPackSourceConfig.cmake  helloworld  Makefile
```

Also, this application must be cross-compiled for aarch64.

```
renesas@builder-pc:~/cmake_helloworld/build$ file helloworld
helloworld: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-aarch64.so.1,
BuildID[sha1]=436a40422c25d0eb57771b5cda061b49e5c197e7, for GNU/Linux 3.14.0,
with debug_info, not stripped
```

12.3 Package Programs with Cpack

This section provides a step-by-step guide on how to configure CMake to package your application into a .deb file, which is a Debian package file. You can install them on the RZ boards as an application. CPack is a CMake module that handles packaging.

12.3.1 Package a C Program

The following steps provide detailed instructions for using CPack to package a C program into a .deb file, including configuring CMake and preparing the necessary files for packaging.

1. Add CPack configuration to CMakeLists.txt from the previous chapter: 12.2 Build a sample application with Cmake.

```
renesas@builder-pc:~/cmake_helloworld$ vi CMakeLists.txt
```

Then, edit your CMakelists.txt file to include CPack configuration

```
cmake_minimum_required(VERSION 3.10)
project(HelloWorld C)

set(CMAKE_C_COMPILER
/home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-gcc)
set(CMAKE_CXX_COMPILER
/home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-g++)

# Sysroot path
set(CMAKE_SYSROOT /home/renesas/esdk/5.1.4/tmp/sysroots/rzg2l-sbc)

add_executable(helloworld src/helloworld.c)

# Specify the installation path
install(TARGETS helloworld DESTINATION /usr/local/bin)

# CPack configuration
set(CPACK_GENERATOR "DEB")
set(CPACK_PACKAGE_NAME "helloworld")
set(CPACK_PACKAGE_VERSION "1.0.0")
set(CPACK_DEBIAN_PACKAGE_ARCHITECTURE "arm64")
set(CPACK_PACKAGE_CONTACT "Your Name <your.email@example.com>")
set(CPACK_DEBIAN_PACKAGE_MAINTAINER "Your name")

include(CPack)
```

2. Package the C program into a Debian package installer.

```
renesas@builder-pc:~/cmake_helloworld$ cd build/
renesas@builder-pc:~/cmake_helloworld$ cmake ../
-- The C compiler identification is GNU 14.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /home/renesas/esdk/5.1.4/sysroots/x86_64-
pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done (0.1s)
-- Generating done (0.0s)
-- Build files have been written to: /home/tiennguyen/cmake/build
renesas@builder-pc:~/cmake_helloworld/build$ cpack
CPack: Create package using DEB
CPack: Install projects
CPack: - Run preinstall target for: HelloWorld
CPack: - Install project: HelloWorld []
CPack: Create package
-- CPACK_DEBIAN_PACKAGE_DEPENDS not set, the package will have no dependencies.
CPack: - package: /home/renesas/cmake_helloworld/build/helloworld-1.0.0-Linux.deb
generated.
```

After completing, confirm that the Debian package (.deb) is generated in the build folder.

```
renesas@builder-pc:~/cmake_helloworld/build$ ls
CMakeCache.txt  cmake_install.cmake  _CPack_Packages          helloworld
install_manifest.txt          CMakeFiles       CPackConfig.cmake
CPackSourceConfig.cmake  helloworld-1.0.0-Linux.deb  Makefile
```

3. Ship the Debian package installer to the target board.

The Debian package installer “helloworld-1.0.0-Linux.deb” can be transferred to the target board using the SCP tool as below, or other methods, such as a USB drive or NFS (Network File System).

```
renesas@builder-pc:~/cmake_helloworld/build$ scp helloworld-1.0.0-Linux.deb
root@<board_IP_address>:<destination>
```

For example:

```
renesas@builder-pc:~/cmake_helloworld/build$ scp helloworld-1.0.0-Linux.deb
root@192.168.5.58:/home/root
```

12.3.2 Package a Python Program

This section explains how to package Python scripts into a .deb file using CPack, focusing on the necessary configurations and packaging steps.

Two options are available for running a Python script:

1. Directly with Python: Use the command `python3 script.py` to execute the script directly.
2. By shell script: Use a shell script to run the Python script. This approach can be useful for adding additional setup or configuration steps.

To run the application without invoking the python3 command directly, create a shell script that contains the command to execute the Python script.

The steps below are similar to those for packaging a C program, with differences primarily in the source code and CPack configuration within the CMakeLists.txt file.

1. Create a workspace for CMake.

```
renesas@builder-pc:~$ mkdir ~/cmake_python
renesas@builder-pc:~$ cd ~/cmake_python
renesas@builder-pc:~/cmake_python$ mkdir build src
```

2. Organize the project structure as shown below:

```
renesas@builder-pc:~/cmake_python$ tree
.
├── build
├── CMakeLists.txt
└── src
    ├── tkinter_wrapper.sh
    └── main.py
```

`CMakeLists.txt`, `tkinter_wrapper.sh`, and `main.py` will be created later in the next steps.

3. Modify the Python program; this program is the same as Tkinter example in section 9.13 Python GUI programming with Tkinter.

Copy this example content and paste it into this Python file.

```
renesas@builder-pc:~/cmake_python$ vi src/main.py
```

4. Create a Tkinter wrapper shell script to run the application.

```
renesas@builder-pc:~/cmake_python$ vi src/tkinter_wrapper.sh
```

Then, copy the content below to the script.

```
#!/bin/bash

/usr/bin/python3 /usr/local/share/tkinter_example/main.py
```

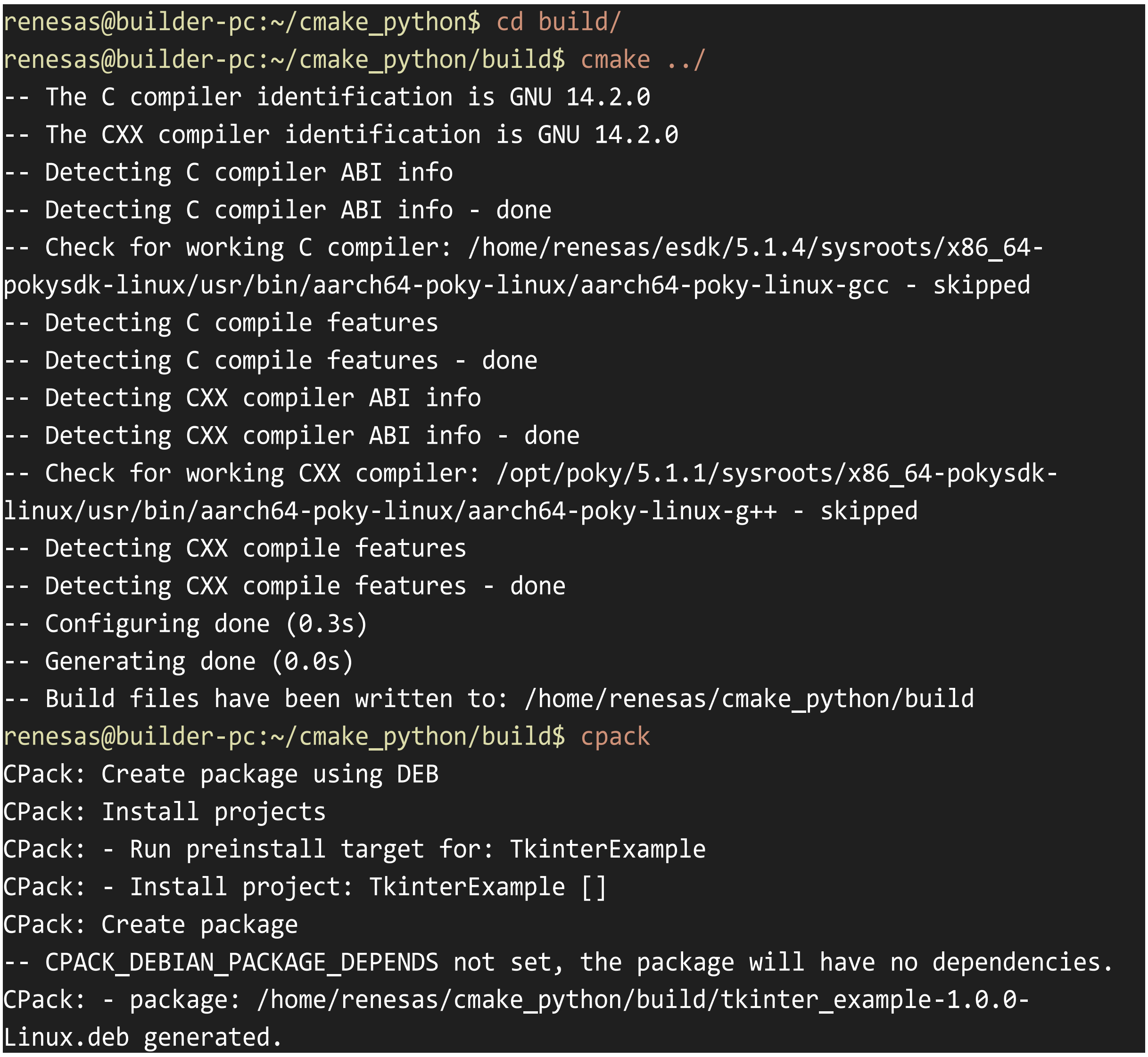
5. Configure the CMakeLists.txt for packaging a Python program.

```
renesas@builder-pc:~/cmake_python$ vi CMakeLists.txt
```

Then, copy the contents below to the file:

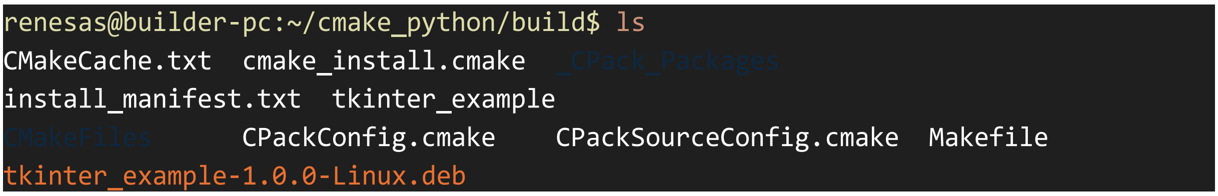
```
cmake_minimum_required(VERSION 3.10)
project(TkinterExample)

# Define script and wrapper
set(SCRIPT_NAME "src/main.py")
set(WRAPPER_SCRIPT "src/tkinter_wrapper.sh")
set(EXEC_NAME "tkinter_example")

# Define installation paths
set(INSTALL_DIR "/usr/local/bin")
set(INSTALL_SCRIPT_DIR "/usr/local/share/tkinter_example")

# Install the wrapper script
configure_file(${CMAKE_SOURCE_DIR}/${WRAPPER_SCRIPT} ${CMAKE_BINARY_DIR}/${EXEC_NAME} @ONLY)
install(PROGRAMS ${CMAKE_BINARY_DIR}/${EXEC_NAME} DESTINATION ${INSTALL_DIR})
install(FILES ${CMAKE_SOURCE_DIR}/${SCRIPT_NAME} DESTINATION ${INSTALL_SCRIPT_DIR})

# Packaging configuration
set(CPACK_GENERATOR "DEB")
set(CPACK_PACKAGE_NAME "tkinter_example")
set(CPACK_PACKAGE_VERSION "1.0.0")
set(CPACK_PACKAGE_CONTACT "your-email@example.com")
set(CPACK_DEBIAN_PACKAGE_ARCHITECTURE "arm64")
include(CPack)
```

6. Packaging the program.



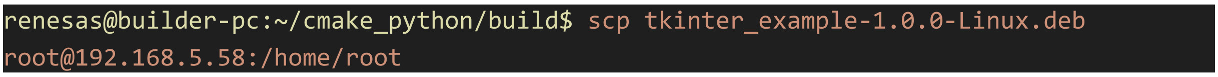
```
renesas@builder-pc:~/cmake_python$ cd build/
renesas@builder-pc:~/cmake_python/build$ cmake ../
-- The C compiler identification is GNU 14.2.0
-- The CXX compiler identification is GNU 14.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /home/renesas/esdk/5.1.4/sysroots/x86_64-
pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /opt/poky/5.1.1/sysroots/x86_64-pokysdk-
linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.3s)
-- Generating done (0.0s)
-- Build files have been written to: /home/renesas/cmake_python/build
renesas@builder-pc:~/cmake_python/build$ cpack
CPack: Create package using DEB
CPack: Install projects
CPack: - Run preinstall target for: TkinterExample
CPack: - Install project: TkinterExample []
CPack: Create package
-- CPACK_DEBIAN_PACKAGE_DEPENDS not set, the package will have no dependencies.
CPack: - package: /home/renesas/cmake_python/build/tkinter_example-1.0.0-
Linux.deb generated.
```

After completing, confirm that the Debian package (.deb) is generated in the build folder.



```
renesas@builder-pc:~/cmake_python/build$ ls
CMakeCache.txt  cmake_install.cmake  _CPack_Packages
install_manifest.txt  tkinter_example
CMakeFiles       CPackConfig.cmake    CPackSourceConfig.cmake  Makefile
tkinter_example-1.0.0-Linux.deb
```

Then, the Debian package installer “tkinter_example-1.0.0-Linux.deb” can be transferred to the target board using the SCP tool as below or other methods, such as a USB drive or NFS (Network File System).



```
renesas@builder-pc:~/cmake_python/build$ scp tkinter_example-1.0.0-Linux.deb
root@192.168.5.58:/home/root
```

12.4 Run Sample Applications

Power on the RZ/G2L-SBC and start the system. Once the system has booted, transfer the binary package that is built using the SDK with CMake, which is mentioned in chapter 13.2 Build a sample application with CMake. Then, run the sample application as follows:

```
NOTICE:  BL2: <version>
NOTICE:  BL2: Built : <date>
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: <version>
NOTICE:  BL31: Built : <date>
…
rzg2l-sbc login: root
root@rzpi:~# ./helloworld

Hello, World!
root@rzg2l-sbc:~#
```

12.5 Install and Run Debian application packages by using DPKG

After shipping the Debian package installer to the target board, the package can be installed using dpkg.

The steps are:

1. List out all available .deb files to make sure all .deb files have been shipped to the target board.
2. Install the C program by running `dpkg -i helloworld-1.0.0-Linux.deb`.
3. Install the Python program by running `dpkg -i tkinter_example-1.0.0-Linux.deb`.

```
NOTICE:  BL2: <version>
NOTICE:  BL2: Built : <date>
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: <version>
NOTICE:  BL31: Built : <date>
…
rz-cmn login: root
root@rz-cmn:~# ls
audios  demo  helloworld-1.0.0-Linux.deb  tkinter_example-1.0.0-Linux.deb  images
info  v4l2-init.sh  videos
root@rz-cmn:~# dpkg -i helloworld-1.0.0-Linux.deb
Selecting previously unselected package helloworld.
(Reading database ... 4 files and directories currently installed.)
Preparing to unpack helloworld-1.0.0-Linux.deb ...
Unpacking helloworld (1.0.0) ...
Setting up helloworld (1.0.0) ...
root@rz-cmn:~#
root@rz-cmn:~# dpkg -i tkinter_example-1.0.0-Linux.deb
Selecting previously unselected package tkinter_example.
(Reading database ... 4 files and directories currently installed.)
Preparing to unpack tkinter_example-1.0.0-Linux.deb ...
Unpacking tkinter_example (1.0.0) ...
Setting up tkinter_example (1.0.0) ...
```

After installation, confirm that the package is correctly installed by running the following.

```
root@rz-cmn:~# dpkg -l
ii  helloworld     1.0.0        arm64       HelloWorld built using CMake
ii  tkinter_example 1.0.0       arm64       TkinterExample built using CMake
```

This completes the installation. The applications are ready for use.

There are a few ways to run the application:

1. Directly from installation location.
2. Call it from /usr/local/bin/<your_application>.
3. Call it directly from anywhere if it is installed within the PATH search.

The following command lists all the files that were installed with their full paths:

```
root@rz-cmn:~# dpkg -L helloworld
/usr
/usr/local
/usr/local/bin
/usr/local/bin/hello
root@rz-cmn:~# /usr/local/bin/hello
Hello, World!
```

For applications that have a graphical interface, the display id needs to be set in the environment. For this reason, export the DISPLAY if you are using an environment where the display is not automatically set, as shown below:

Switch to the user 'weston' to run the Tkinter application

```
root@rz-cmn:~# su - weston
```

```
root@rz-cmn:~$ export DISPLAY=:0
root@rz-cmn:~$ dpkg -L tkinter_example
/usr
/usr/local
/usr/local/bin
/usr/local/bin/tkinter_example
/usr/local/share
/usr/local/share/tkinter_example
/usr/local/share/tkinter_example/main.py
root@rz-cmn:~$ tkinter_example
```

12.6 Using devtool in the Yocto eSDK

12.6.1 Overview

The devtool utility is part of the Yocto Project's Extensible SDK (eSDK). It provides an isolated workspace for modifying, testing, and maintaining recipes without altering upstream metadata.

This section focuses on the **core commands** used in day-to-day development, especially for Linux kernel, device trees, and driver modifications on Renesas RZ Common System.

12.6.2 Prerequisites

Before using devtool, ensure the following:

1. The Yocto eSDK has been installed and extracted, see Section 12.1. How to extract the eSDK for detailed information.
2. The environment setup script has been sourced:

   ```
   $ source /path/to/poky_sdk/environment-setup-<arch>-poky-linux
   ```

   For example:

   ```
   renesas@builder-pc:~$ source ~/poky_sdk/environment-setup-cortexa55-poky-linux
   SDK environment now set up; additionally you may now run devtool to perform
   development tasks.
   Run devtool --help for further details.
   ```

12.6.3 Common Usage Scenarios

This section explains how to use the main **devtool** commands in the Yocto Extensible SDK, along with examples based on Linux kernel development for Renesas RZ boards.

12.6.3.1 devtool modify – Preparing a Workspace

The devtool modify command checks out the source code for a recipe into a local workspace, allowing changes without touching upstream layers. This is usually the first step in customizing the kernel, device trees, or drivers.

Syntax:

```
renesas@builder-pc:~$ devtool modify <recipe>
```

Example (modify linux-yocto recipe):

```
renesas@builder-pc:~$ devtool modify linux-yocto
```

When executed, this:

- Creates a workspace copy of the kernel source under:

  ~/poky_sdk/workspace/sources/linux-yocto/
- Generates a .bbappend for linux-yocto in the workspace's recipe area.
- Prepares the environment for kernel modifications.

(1) Applying Patches for linux-yocto

Unlike most recipes, linux-yocto in this BSP is implemented as an out-of-tree kernel.

- Kernel modifications are stored as patches inside workspace/sources/linux-yocto/.kernel-meta/
- The kernel default configuration (renesas_defconfig) is also managed out-of-tree.
- To ensure the workspace matches the recipe, patches must be applied after running devtool modify.

Procedure, applying patches to the kernel linux-yocto source

```
renesas@builder-pc:~$ cd ~/poky_sdk/workspace/sources/linux-yocto
renesas@builder-pc:~/poky_sdk/workspace/sources/linux-yocto$ git am \
.kernel-meta/patches/*
```

This applies the patch series defined in .kernel-meta/patches/ to the kernel workspace.

After applying patches, developers may:

- Provide kernel configuration fragments (.cfg) to adjust features.
- Continue with devtool build linux-yocto to compile the kernel with modifications.

(2) Adding Kernel Configuration

There are two possible methods to add kernel configuration for linux-yocto:

1. Edit the out-of-tree defconfig directly:

   ~/poky_sdk/layers/meta-renesas/recipes-kernel/linux/rzg2l-sbc/rzg2l-sbc_defconfig
2. Adding a configuration fragment (.cfg) file
   - Create the append structure

   ```
   renesas@builder-pc:~$ mkdir -p ~/poky_sdk/workspace/appends/<recipe-name>/files
   ```

   For example, create an append structure for linux-yocto

   ```
   renesas@builder-pc:~$ mkdir -p ~/poky_sdk/workspace/appends/linux-yocto/files
   ```

   - Place the fragment in files/, e.g., usb-serial-ftdi.cfg:

   ```
   sCONFIG_USB_SERIAL=y
   CONFIG_USB_SERIAL_FTDI_SIO=y
   ```

   - Create/modify the bbappend

```
renesas@builder-pc:~$ vim ~/poky_sdk/workspace/appends/linux-yocto/\
linux-yocto_6.10.bbappend
```

Add the usb-serial.cfg file to SRC_URI to tell the kernel to apply this fragment

```
SRC_URI:append = " file://usb-serial-ftdi.cfg"
```

12.6.3.2 devtool build – Building the Recipe

After making changes in the workspace, use devtool build to compile the recipe.

Syntax

```
renesas@builder-pc:~$ devtool build <recipe>
```

Example:

```
renesas@builder-pc:~$ devtool build linux-yocto
```

What it does

- Uses the workspace sources for <recipe> (created by devtool modify <recipe>).
- Runs the recipe's standard BitBake tasks (e.g., do_compile, do_install, packaging) with dependency handling.
- Produces deployable artifacts for the recipe (binaries, libraries, firmware, kernel modules, etc.), depending on the recipe's class.

What it does not do

- Does not rebuild a complete image. To include changes in an image, use devtool build-image <image>
- To integrate changes into a bootable image, use devtool build-image <image>.

> **Note**: When artifacts generated by `devtool build` (for example, `Image`, DTBs, or kernel modules) are intended for use with Ubuntu-based root file systems (such as `ubuntu-core-image` or `ubuntu-lxde-image`), the build must be performed with `DISTRO=ubuntu-tiny`.
>
> Export the variable before running the build:
>
> $ export DISTRO=ubuntu-tiny
>
> $ devtool build <recipe>
>
> Otherwise, the generated artifacts will not be compatible with the Ubuntu image.

(1) Typical output (generic locations):

Work directory (per-recipe, per-machine):

- <sdk-root>/tmp/work/<machine>-poky-linux/<recipe>/<version>/
  Contains intermediate build results, the image/ directory with installed files, and logs.
- Deployed artifacts (if the recipe deploys output): <sdk-root>/tmp/deploy/

For example, when building linux-yocto, the artifacts can collect from these locations:

- Kernel modules (.ko files):~/poky_sdk/tmp/work/rz-cmn-poky-linux/linux-yocto/6.10.14+git/image/usr/lib/modules/6.10.14-yocto-standard/kernel/
- Device tree and Image:

  - In work directory: ~/poky_sdk/tmp/work/rz-cmn-poky-linux/linux-yocto/6.10.14+git/image/boot/
  - In deploy directory (as the recipe deploys output): ~/poky_sdk/tmp/deploy/images/rz-cmn/target/images/linux

12.6.3.3 devtool reset – Cleaning Up

When experimentation is complete, use devtool reset to remove the workspace copy and restore the recipe to its original state. This ensures that subsequent builds use the unmodified upstream recipe.

Command:

```
renesas@builder-pc:~$ devtool reset <recipe>
```

Example:

```
renesas@builder-pc:~$ devtool reset linux-yocto
```

This command deletes the workspace sources and temporary append files, cleaning up after testing (e.g., kernel debug options). Any changes not preserved with devtool update-recipe will be lost.

12.6.3.4 devtool build-image – Building Target Image

The devtool build-image command builds a complete target image that includes all recipes currently under modification. This is useful to verify integration of changes into a full root filesystem, not just individual components.

Command:

```
renesas@builder-pc:~$ devtool build-image <image>
```

Example:

```
renesas@builder-pc:~$ devtool build-image core-image-weston
```

When executed, this command:

- Rebuilds the specified image recipe.
- Automatically includes outputs from any recipes currently modified in the workspace.
- Produces a bootable image in the deploy directory:
  - .wic file (complete bootable image): <sdk-root>/tmp/deploy/images/<machine>/target/images/
  - Compressed root filesystem (for flashing or NFS root): <sdk-root>/tmp/deploy/images/<machine>/target/images/rootfs

Use devtool build-image after testing an individual recipe (e.g., linux-yocto) to confirm integration into the final image.

- The resulting image can be flashed to the target device for full system validation.
- This is equivalent to running bitbake <image>

13. Remote Debugging using GDBServer

GDBServer is utilized to facilitate remote debugging on the target board. GDBServer enables the debugging process to run on the target machine while being controlled from a different system (the host machine) via a network connection.

This setup is particularly beneficial for application development, as it allows the execution and debugging of programs on the target board while providing the capability to view and control the process from the host machine.

To ensure that all necessary tools and libraries for debugging are available, preparations must be made on both the host and target machines. With this preparation complete, the next step is to proceed with the remote debugging process.

13.1 Prepare GDB on the Host Machine

GDB has two components to work with. One is the host side ‘gdb’ debugger. The other is the target side ‘gdbserver’. The GDB (GNU debugger) is executed on the host side. It is executed on your host system to connect to the target system. It is always available within the eSDK. The eSDK installation as described in Section 12.1 is a prerequisite for this operation. To set up the environment that would use the GDB targeting the target board from the eSDK, simply run the poky environment script as follows:

```
renesas@builder-pc:~$ source ~/esdk/5.1.4/environment-setup-cortexa55-poky-linux
SDK environment now set up; additionally you may now run devtool to perform
development tasks.
Run devtool --help for further details.
```

Note: The user needs to run the above command once for each shell session. In addition, source’ is a bash specific call. The POSIX convention is to use. ~/esdk/5.1.4/environment-setup-cortexa55-poky-linux’. Bash equates ‘source’ to ‘.’.

To confirm GDB is ready to use, run the following command and check the result:

```
renesas@builder-pc:~$ echo ${GDB}

aarch64-poky-linux-gdb
```

13.2 Install GDBServer on RZ

By default, GDBServer is not installed on the target board. It is necessary to install it using APT. Execute the following commands to install GDBServer:

```
root@rz-cmn:~# apt-get update

root@rz-cmn:~# apt-get install gdbserver
```

Ensure that internet access is available before executing apt-get update.

This concludes the preparation of the basic host environment. The next section will discuss the remote debugging process.

13.3 Remote Debugging Example

13.3.1 Remote Debugging on CLI

CLI (Command Line Interface) is a text-based user interface used to interact with computer programs and operating systems. Unlike graphical user interfaces (GUIs), where users interact with visual elements (like buttons and icons), a CLI requires users to input commands in text form. This is basically a shell environment used in all operating systems as a foundational method of interacting with the system. For the purposes of this section, we assume Ubuntu bash as the interactive application.

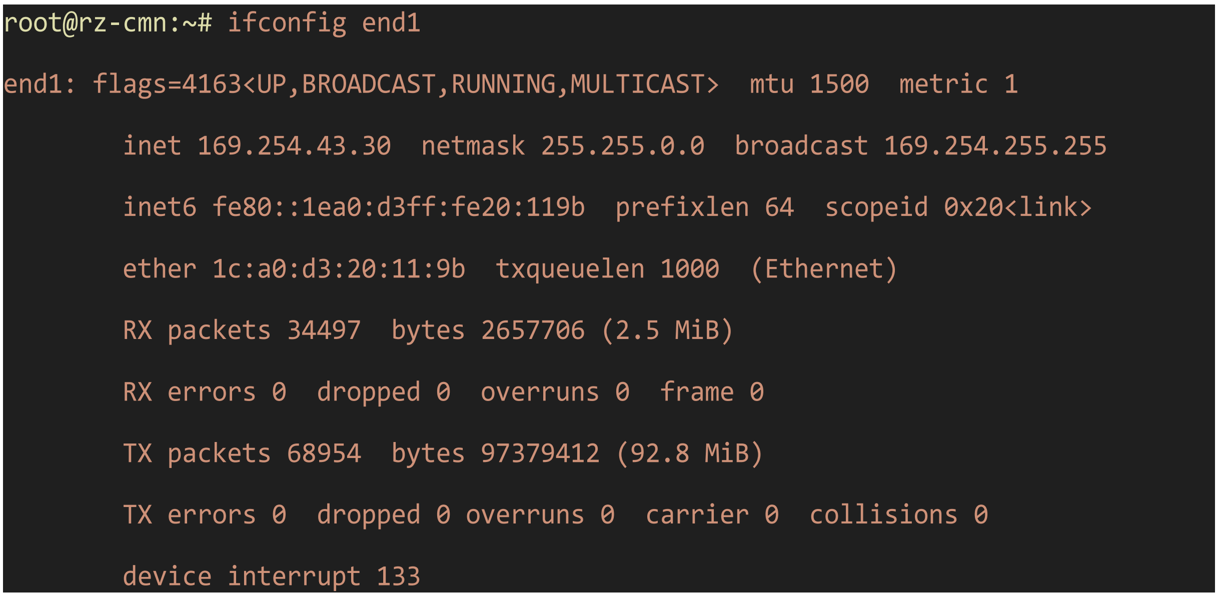
Firstly, run GDBServer with a specific network port (`2000` is the assigned port in this case) and the program `hello-gdbserver` as a parameter on the target as follows:

```
root@rz-cmn:~# gdbserver localhost:2000 hello-gdbserver

Process /home/root/hello-gdbserver created; pid = 358

Listening on port 2000
```

The content before compiling of the `hello-gdbserver` program:

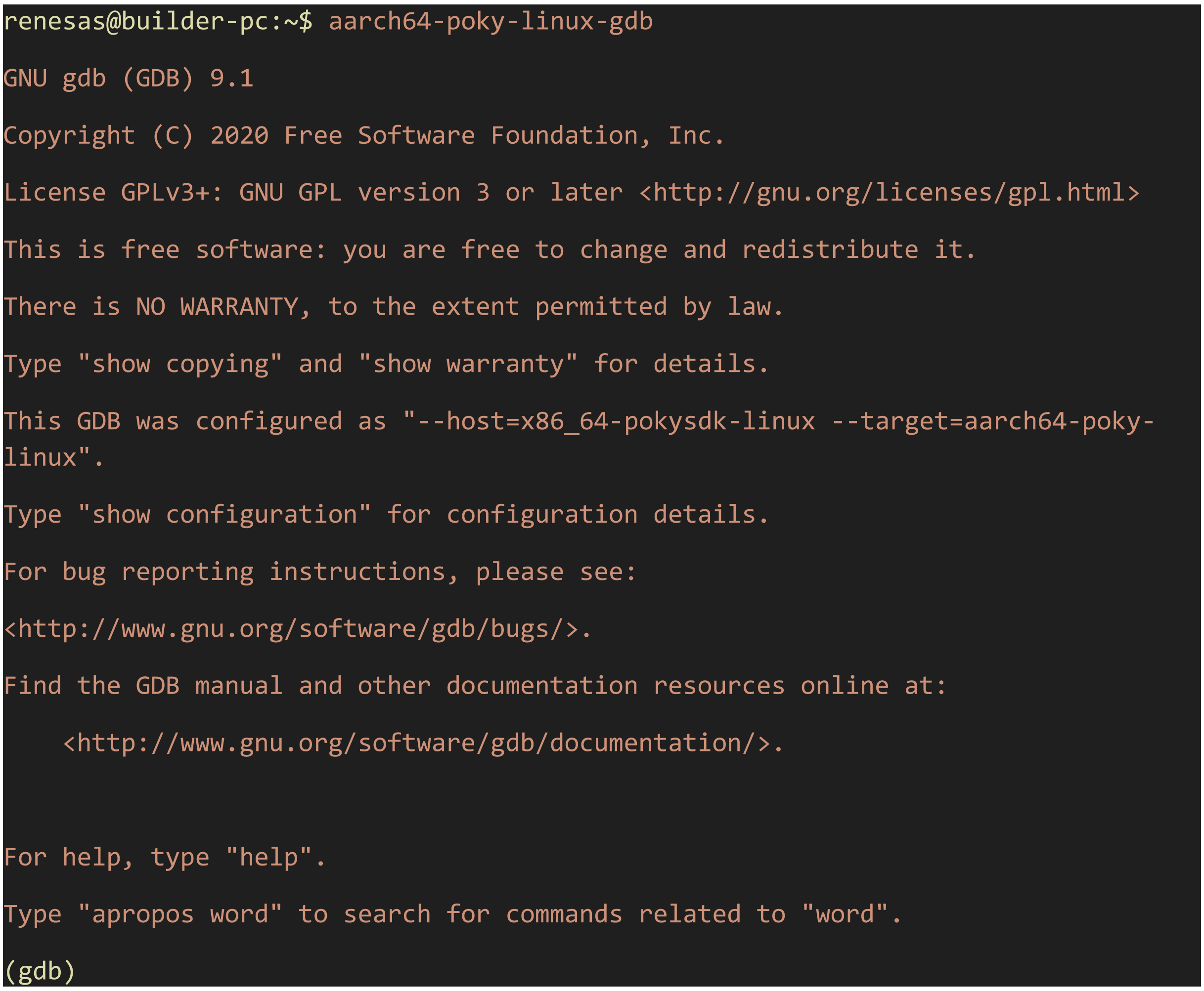
```
#include <stdio.h>

int main() {

        int i;

        printf("Program to demonstrate gdbserver debugging!\n");
        printf("Print from 1 to 10\n");

        for (i = 1;i <= 10;i++)
                printf("%d\n", i);

        printf("Program completed!\n");

        return 0;
}
```

The target's IP address is required for use on the host later. In this example, the IP address 169.254.43.30 will be used.



```
root@rz-cmn:~# ifconfig end1
end1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  metric 1
        inet 169.254.43.30  netmask 255.255.0.0  broadcast 169.254.255.255
        inet6 fe80::1ea0:d3ff:fe20:119b  prefixlen 64  scopeid 0x20<link>
        ether 1c:a0:d3:20:11:9b  txqueuelen 1000  (Ethernet)
        RX packets 34497  bytes 2657706 (2.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 68954  bytes 97379412 (92.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 133
```

Next, launch GDB on the host.



```
renesas@builder-pc:~$ aarch64-poky-linux-gdb
GNU gdb (GDB) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pokysdk-linux --target=aarch64-poky-
linux".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

Use `target remote` with the IP address and the assigned network port to connect to the target.

```
(gdb) target remote 169.254.43.30:2000
Remote debugging using 169.254.43.30:2000
Reading /home/root/hello-gdbserver from remote target...
warning: File transfers from remote targets can be slow. Use "set sysroot" to
access files locally instead.
Reading /home/root/hello-gdbserver from remote target...
Reading symbols from target:/home/root/hello-gdbserver...
Reading /lib64/ld-linux-aarch64.so.1 from remote target...
Reading /lib64/ld-linux-aarch64.so.1 from remote target...
Reading symbols from target:/lib64/ld-linux-aarch64.so.1...
Reading /lib64/ld-2.31.so from remote target...
Reading /lib64/.debug/ld-2.31.so from remote target...
Reading /lib64/.debug/ld-2.31.so from remote target...
Reading symbols from target:/lib64/.debug/ld-2.31.so...
0x0000fffff7fcd0c0 in _start () from target:/lib64/ld-linux-aarch64.so.1
```

Then, add a break point at `main` function to stop the program at that function in the next step:

```
(gdb) b main

Breakpoint 1 at 0xaaaaaaaa07cc: file hello-gdbserver.c, line 7.
```

At this point, the `continue` command can be used to resume execution and jump to the main function.

```
(gdb) continue

Continuing.

Reading /lib64/libc.so.6 from remote target...

Reading /lib64/libc-2.31.so from remote target...

Reading /lib64/.debug/libc-2.31.so from remote target...

Reading /lib64/.debug/libc-2.31.so from remote target...



Breakpoint 1, main () at hello-gdbserver.c:7

warning: Source file is more recent than executable.

7               printf("Program to demonstrate gdbserver debugging!\n");
```

Then, type `continue` to execute the remainder of the program.

```
(gdb) continue

Continuing.

[Inferior 1 (process 342) exited normally]
```

Eventually, run `quit` to exit GDB and stop the debugging section.

```
(gdb) quit
```

In parallel, the output can be monitored on the target device.

```
Remote debugging from host ::ffff:169.254.43.86, port 40666
Program to demonstrate gdbserver debugging!
Print from 1 to 10
1
2
3
4
5
6
7
8
9
10
Program completed!

Child exited with status 0
root@rzg2l-sbc:~#
```

13.3.2 Remote Debugging on Visual Studio Code

In the previous subsection, remote debugging using the command line was discussed, specifically with GDB and GDBServer. While this method is effective, it can be complex and challenging, particularly for developers who may not be familiar with command-line operations.

This section describes how to set up and use Visual Studio Code (VSCode) for remote debugging with the GDB (GNU Debugger) extension. Using VSCode simplifies the debugging process by providing a user-friendly graphical interface that streamlines the workflow, making it easier to troubleshoot and test C/C++ applications running on the target board.

Here’s how to get started:

1. Install the C/C++ Extension (If it has not been installed yet):
    - Open VSCode.
    - Go to the Extensions tab on the left side (or press Ctrl + Shift + X).
    - Search for C/C++.
    - Click Install to add the extension.

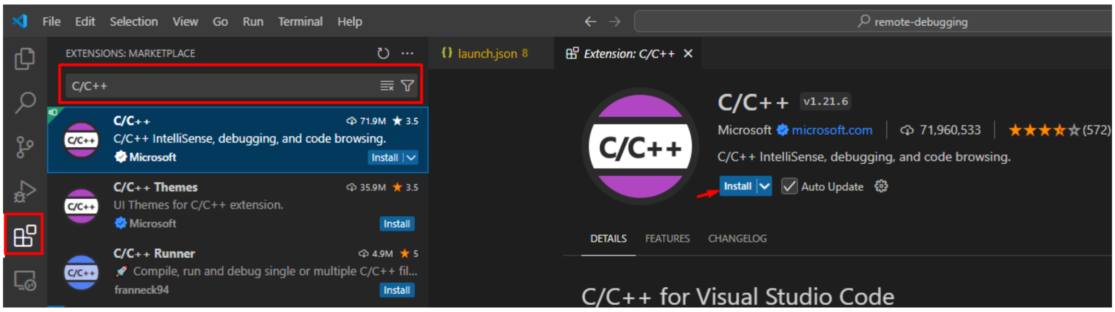

File Edit Selection View Go Run Terminal Help
remote-debugging
EXTENSIONS: MARKETPLACE
C/C++
C/C++ 71.9M 3.5
C/C++ IntelliSense, debugging, and code browsing.
Microsoft
Install
C/C++ Themes 35.9M 3.5
UI Themes for C/C++ extension.
Microsoft
Install
C/C++ Runner 4.9M 5
Compile, run and debug single or multiple C/C++ fil...
franneck94
Install
launch.json 8
Extension: C/C++
C/C++ v1.21.6
Microsoft microsoft.com 71,960,533 (572)
C/C++ IntelliSense, debugging, and code browsing.
Install
Auto Update
DETAILS FEATURES CHANGELOG
C/C++ for Visual Studio Code

Figure 57. C/C++ Extension in VSCode

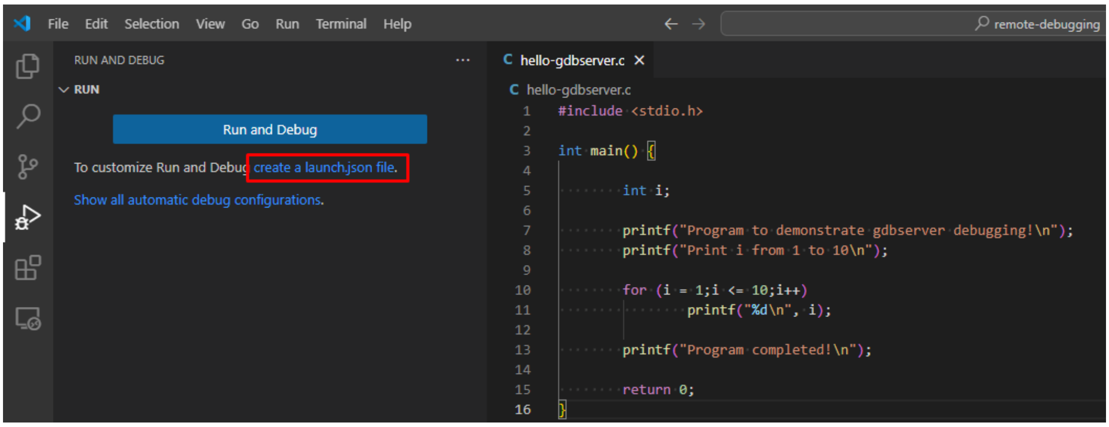
2. Create a Workspace:

- Create a new workspace (you can name it remote debugging).
- Create a folder within this workspace and place your program file, hello-gdbserver.c in it.
- Build the execution file using eSDK, we assume that you have sourced the environment.

```
renesas@builder-pc:~/remote-debugging/program$ $CC $CFLAGS hello-gdbserver.c -o
hello-gdbserver
```

3. Set Up Debug Configuration:

- Open the Run and Debug view in VSCode (or press Ctrl + Shift + D).
- Click on create a launch.json file to configure the debugger.


File Edit Selection View Go Run Terminal Help
remote-debugging
RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Show all automatic debug configurations.
hello-gdbserver.c
hello-gdbserver.c

```
#include <stdio.h>

int main() {
	int i;

	printf("Program to demonstrate gdbserver debugging!\n");
	printf("Print i from 1 to 10\n");

	for (i = 1;i <= 10;i++)
		printf("%d\n", i);

	printf("Program completed!\n");

	return 0;
}
```

Figure 58. Create a launch.json file in VSCode Debugger

- Select the C++ (GDB) option and customize the configuration as needed.

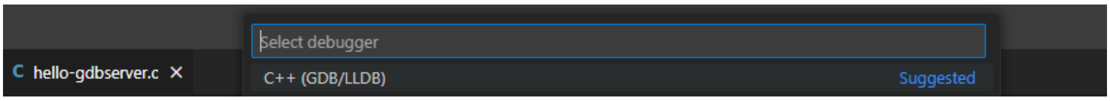


Figure 56. Select C++ GDB as Debugger

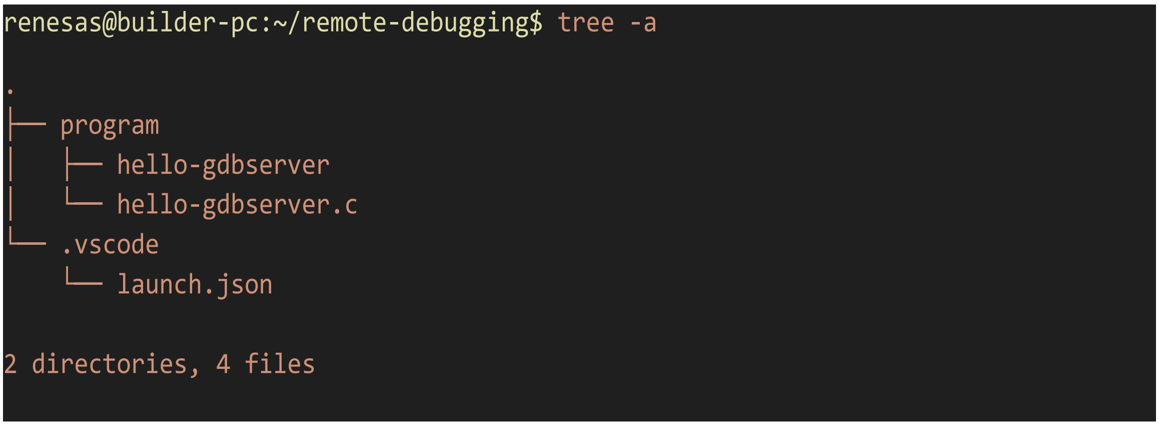
- Place the content below in launch.json file:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "gdb",
            "type": "cppdbg",
            "request": "launch",
            "program": "</local/path/to/the/executable>",
            "cwd": "${workspaceFolder}",
            "stopAtEntry": true,
            "stopAtConnect": true,
            "MIMode": "gdb",
            "miDebuggerPath": "</path/to/gdb>",
            "miDebuggerServerAddress": "<target_addr>:<port>",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        }
    ]
}
```

For example:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "gdb",
            "type": "cppdbg",
            "request": "launch",
            "program": "/home/renesas/remote-debugging/program/hello-gdbserver",
            "cwd": "${workspaceFolder}",
            "stopAtEntry": true,
            "stopAtConnect": true,
            "MIMode": "gdb",
            "miDebuggerPath":
"/home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-gdb",
            "miDebuggerServerAddress": "169.254.43.30:2000",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        }
    ]
}
```

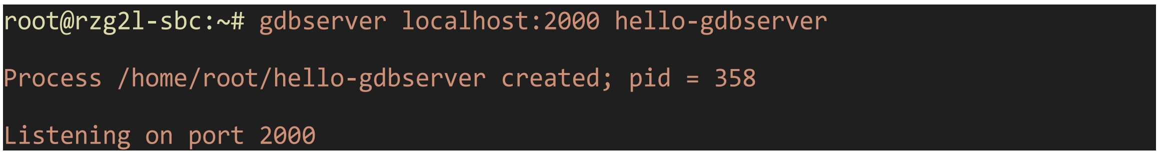
- Ensure your workspace appears as follows:



```
renesas@builder-pc:~/remote-debugging$ tree -a
.
├── program
│   ├── hello-gdbserver
│   └── hello-gdbserver.c
└── .vscode
    └── launch.json

2 directories, 4 files
```

4. Connect to the Remote Target:

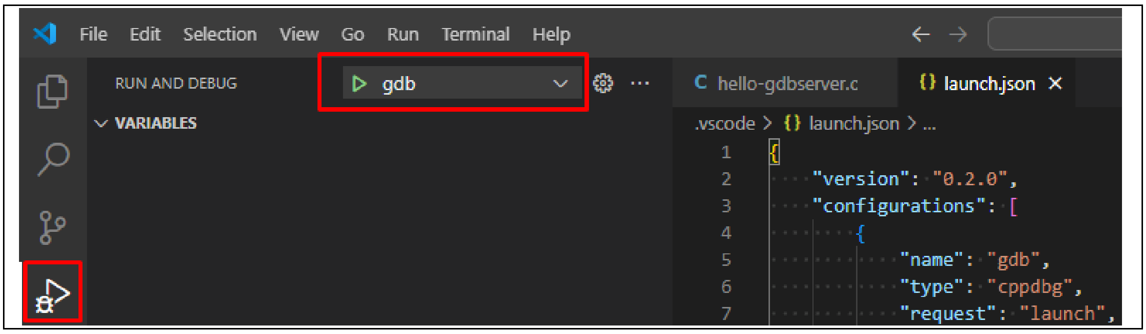
- As with the CLI section, start the GDBServer on the remote device and specify the target application.



```
root@rzg2l-sbc:~# gdbserver localhost:2000 hello-gdbserver
Process /home/root/hello-gdbserver created; pid = 358
Listening on port 2000
```

5. Start the debugging.

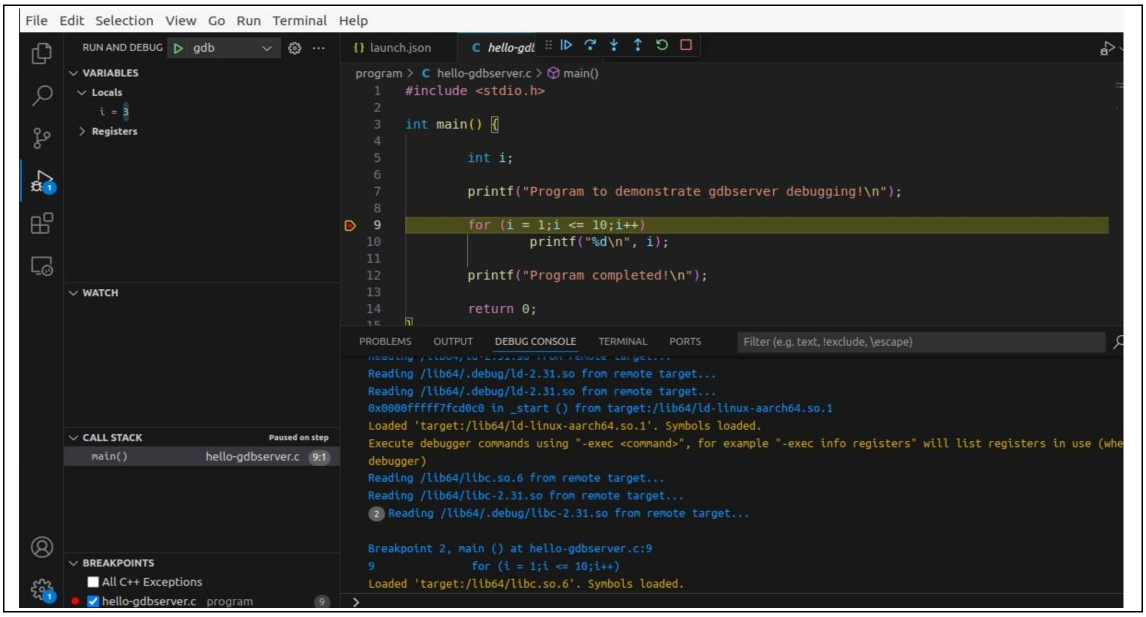
- Back in VSCode, select your launch configuration.
- Place the breakpoint within the hello-gdbserver.c file in VSCode.
- Click the Start Debugging button (green play icon) to begin the debugging session.



```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "gdb",
            "type": "cppdbg",
            "request": "launch",
```

Figure 59. Running the Debugger for Remote Debugging in VSCode

- Use F5 to continue execution, F10 to step over the current line, and F11 to step into functions.



File Edit Selection View Go Run Terminal Help
RUN AND DEBUG ▷ gdb
VARIABLES
Locals
i = 3
Registers
WATCH
CALL STACK Paused on step
main() hello-gdbserver.c 9:1
BREAKPOINTS
All C++ Exceptions
hello-gdbserver.c program 9
launch.json
hello-gdb
program > hello-gdbserver.c > main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Filter (e.g. text, !exclude, \escape)

```
#include <stdio.h>

int main() {

        int i;

        printf("Program to demonstrate gdbserver debugging!\n");

        for (i = 1;i <= 10;i++)
                printf("%d\n", i);

        printf("Program completed!\n");

        return 0;
}
```

```
Reading /lib64/.debug/ld-2.31.so from remote target...
Reading /lib64/.debug/ld-2.31.so from remote target...
0x0000fffff7fcd0c0 in _start () from target:/lib64/ld-linux-aarch64.so.1
Loaded 'target:/lib64/ld-linux-aarch64.so.1'. Symbols loaded.
Execute debugger commands using "-exec <command>", for example "-exec info registers" will list registers in use (whe
debugger)
Reading /lib64/libc.so.6 from remote target...
Reading /lib64/libc-2.31.so from remote target...
2 Reading /lib64/.debug/libc-2.31.so from remote target...

Breakpoint 2, main () at hello-gdbserver.c:9
9               for (i = 1;i <= 10;i++)
Loaded 'target:/lib64/libc.so.6'. Symbols loaded.
```

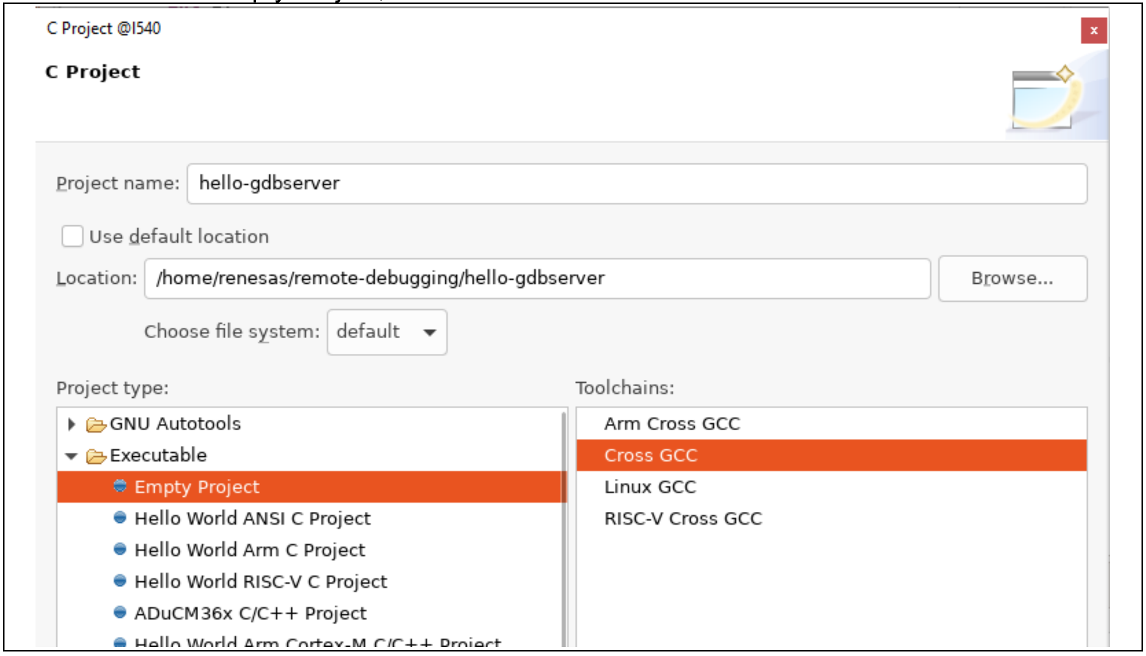
Figure 60. Step through each step in Debug Mode in VSCode

13.3.3 Remote Debugging on Eclipse IDE

In the previous section, the use of VSCode for remote debugging with GDB and GDBServer was discussed. While VSCode offers a modern and user-friendly environment, many developers prefer Eclipse IDE for its comprehensive toolset and robust support for C/C++ development. This section explains how to set up and use Eclipse IDE for remote debugging with GDB.

1. Install the Eclipse IDE (if not already installed) by following the official instructions on the Eclipse website: Eclipse Installer 2024-09 R | Eclipse Packages
2. Create a C/C++ project:

- Open Eclipse and navigate to File > New > C/C++ Project.
- Create a new C Empty Project, choose Cross GCC.



C Project @I540

C Project

Project name: hello-gdbserver

☐ Use default location

Location: /home/renesas/remote-debugging/hello-gdbserver

Choose file system: default

Project type: Executable > Empty Project

Toolchains: Cross GCC

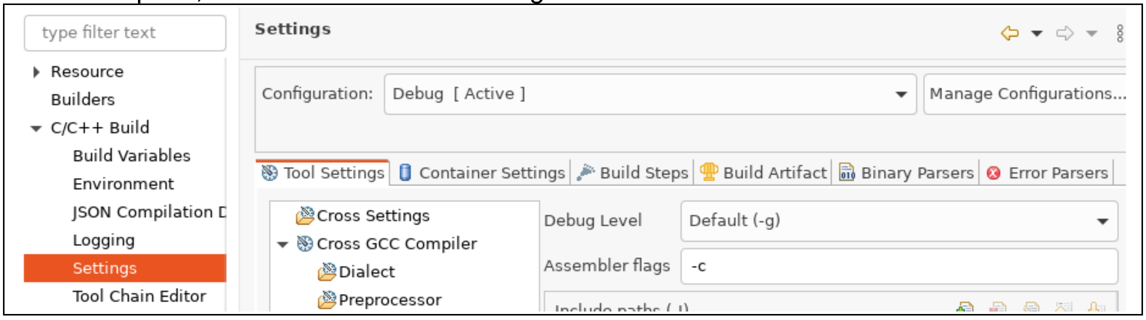
Browse...
GNU Autotools
Executable
Empty Project
Hello World ANSI C Project
Hello World Arm C Project
Hello World RISC-V C Project
ADuCM36x C/C++ Project
Hello World Arm Cortex-M C/C++ Project
Arm Cross GCC
Cross GCC
Linux GCC
RISC-V Cross GCC

Figure 61. Create a C Project in Eclipse

Click Next, then Finish and paste the content from hello-gdbserver.c into the C file.

3. Configure the Cross Toolchain.

- Go to Project > Properties.
- In the left pane, select C/C++ Build > Settings.



Configuration: Debug [Active]

Debug Level: Default (-g)

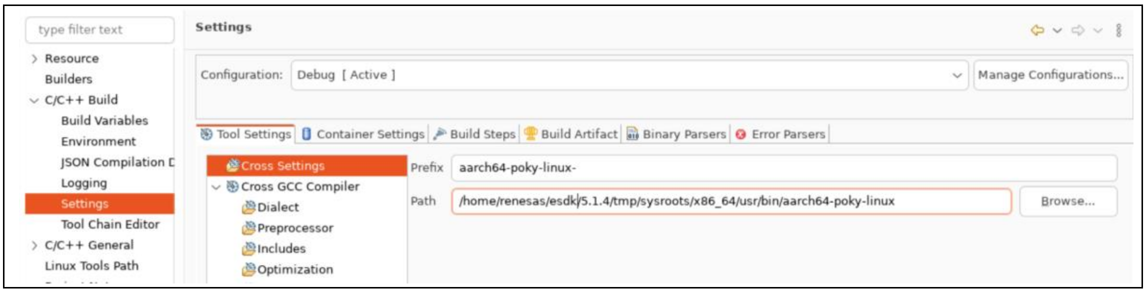
Assembler flags: -c

Figure 62. Configuring the cross toolchain in Eclipse project properties

- Under the Tool Settings tab, configure the Cross Settings as follows:
  - Prefix: aarch64-poky-linux-
  - Path: </path/to/your/aarch64-poky-linux>.

For example:

  - Prefix: aarch64-poky-linux-
  - Path: /home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux.



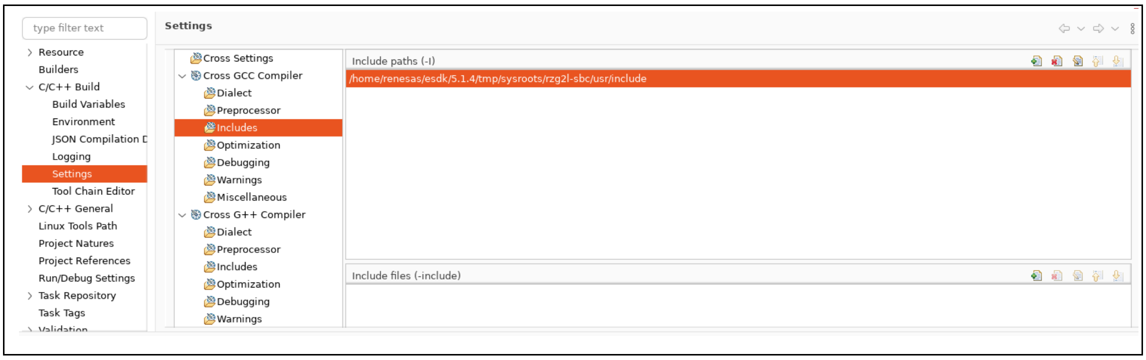
Configuration: Debug [Active]

Prefix: aarch64-poky-linux-

Path: /home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux

Figure 63. Configuring cross compiler settings in Eclipse tool settings

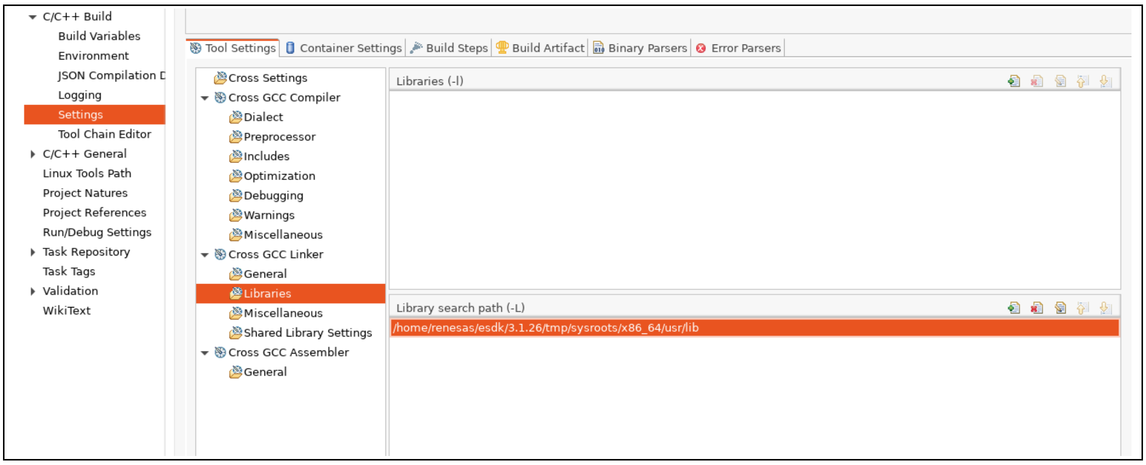
- In the Includes section, specify the include paths:
  - Include paths: /home/renesas/esdk/5.1.4/tmp/sysroots/rz-cmn/usr/include



Include paths (-I)
/home/renesas/esdk/5.1.4/tmp/sysroots/rzg2l-sbc/usr/include

Figure 64. Configuring includes path in Eclipse tool settings

- In the Cross GCC Linker section, go to Libraries and specify the library search path:
  - Library search path: /home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/lib



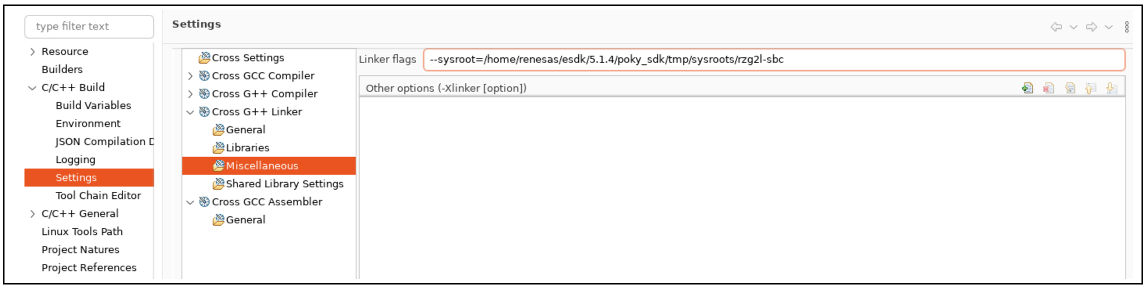
Libraries (-l)

Library search path (-L)

/home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/lib

Figure 65. Configuring library paths in Eclipse tool settings

- In the Miscellaneous section, specify the linker flags:
  - Linker flags: --sysroot=/home/renesas/esdk/5.1.4/poky_sdk/tmp/sysroots/rz-cmn



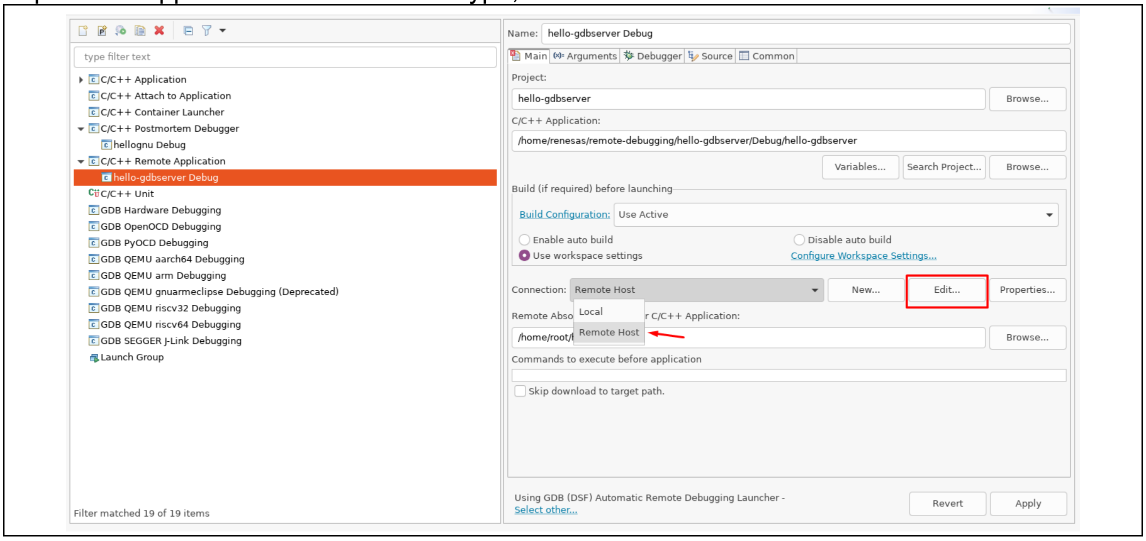
Linker flags --sysroot=/home/renesas/esdk/5.1.4/poky_sdk/tmp/sysroots/rzg2l-sbc

Other options (-Xlinker [option])

Figure 66. Configuring linker flags for Sysroot in Eclipse tool settings

4. Configure Eclipse to connect to the GDB Server:

- In Eclipse, go to the Run menu and select Debug Configurations.
- Under the Debugger tab, select C/C++ Remote Application.
- In the Main tab, click Edit to configure the target IP/hostname. After setting it up, the Remote option will appear under Connection Type, select Remote to enable the remote connection.



Name: hello-gdbserver Debug

Project:

hello-gdbserver

C/C++ Application:

/home/renesas/remote-debugging/hello-gdbserver/Debug/hello-gdbserver

Build (if required) before launching

Build Configuration: Use Active

○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

Connection: Remote Host

Remote Absolute File Path for C/C++ Application:

/home/root/

Commands to execute before application

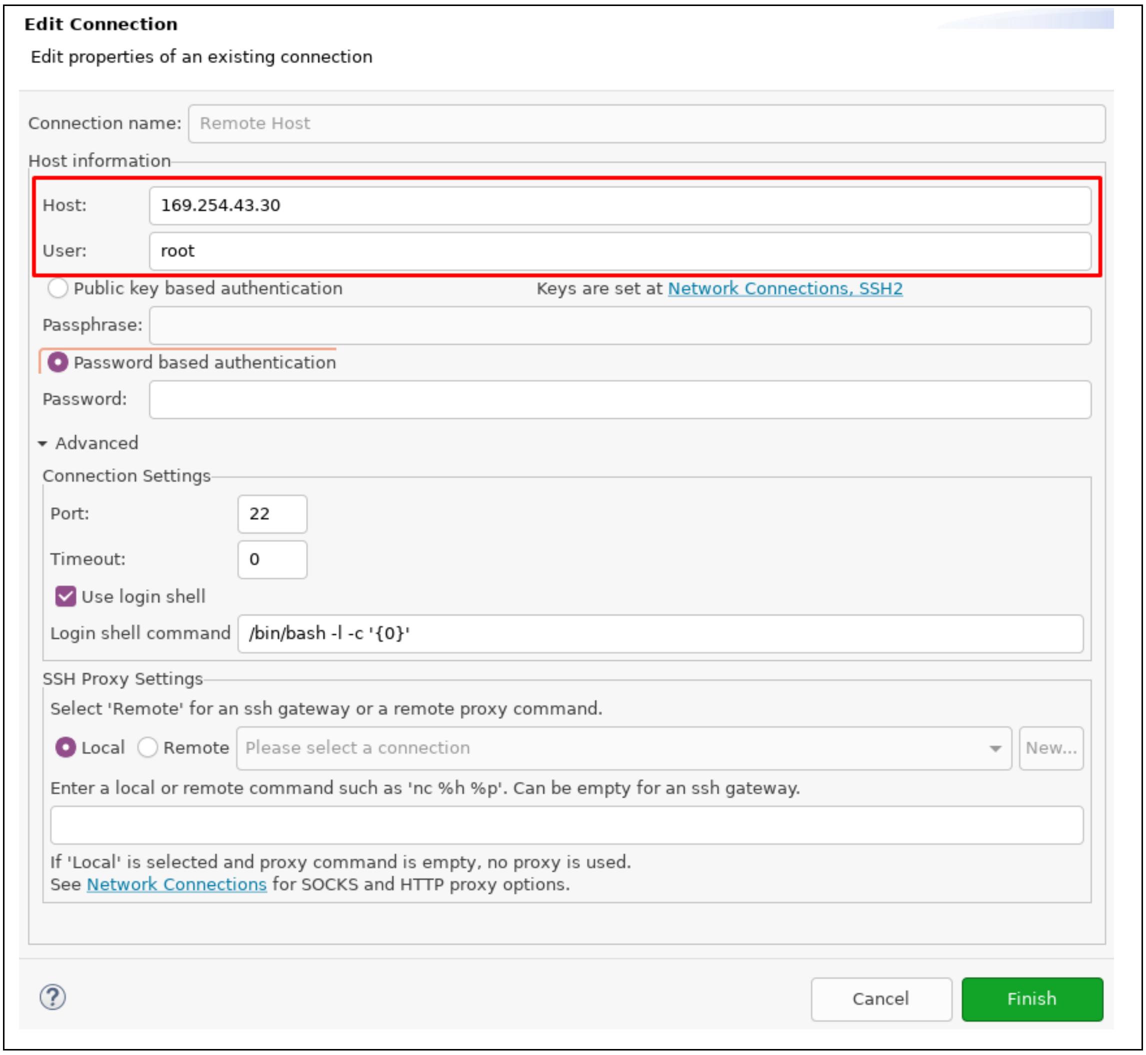
□ Skip download to target path.

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

Figure 67. Debug configuration settings in Eclipse

- Host: Enter the IP address of the RZ/G2L-SBC.
- User: Enter the username of the RZ/G2L-SBC (typically root).

- Authentication: Choose between key-based authentication or password-based authentication, depending on your preference.
- Finally, click Finish to complete the setup for the SSH session.



Edit Connection

Edit properties of an existing connection

Connection name: Remote Host

Host information

Host: 169.254.43.30

User: root

○ Public key based authentication Keys are set at Network Connections, SSH2

Passphrase:

● Password based authentication

Password:

Advanced

Connection Settings

Port: 22

Timeout: 0

☑ Use login shell

Login shell command /bin/bash -l -c '{0}'

SSH Proxy Settings

Select 'Remote' for an ssh gateway or a remote proxy command.

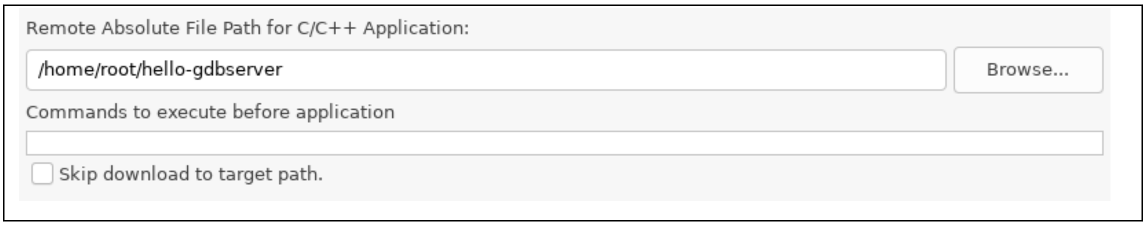
● Local ○ Remote Please select a connection New...

Enter a local or remote command such as 'nc %h %p'. Can be empty for an ssh gateway.

If 'Local' is selected and proxy command is empty, no proxy is used.
See Network Connections for SOCKS and HTTP proxy options.

Figure 68. Configuring SSH connection settings in debug configurations

- In the Remote Absolute File Path field, specify the location where Eclipse will copy the program on the target board Click Browse to connect via SSH and select the target location or manually enter the path on the target board.



Remote Absolute File Path for C/C++ Application:

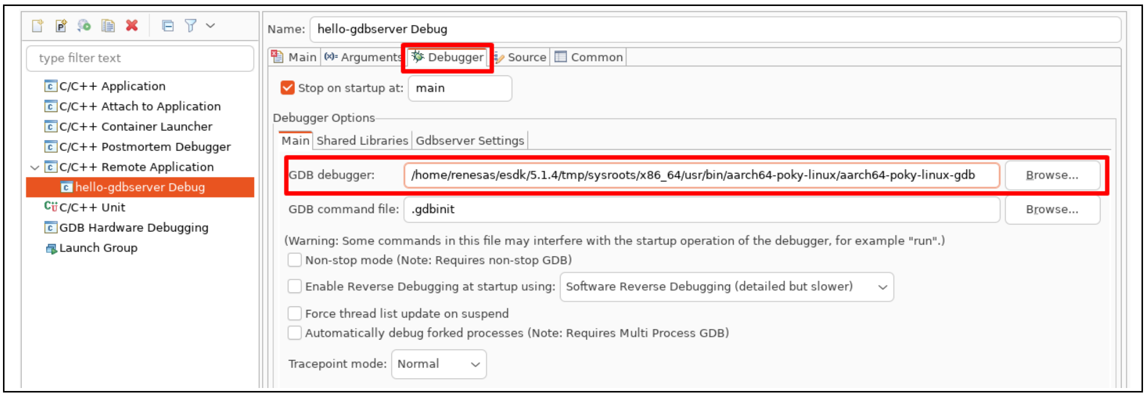
/home/root/hello-gdbserver

Commands to execute before application

□ Skip download to target path.

Figure 69. Configuring the remote absolute file path in debug configurations

- In the Debugger tab:
  - In GDB Debugger: Provide the path to your cross-compiled GDB (For example., /home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb).



Name: hello-gdbserver Debug

☑ Stop on startup at: main

GDB debugger: /home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

□ Non-stop mode (Note: Requires non-stop GDB)

□ Enable Reverse Debugging at startup using: Software Reverse Debugging (detailed but slower)

□ Force thread list update on suspend

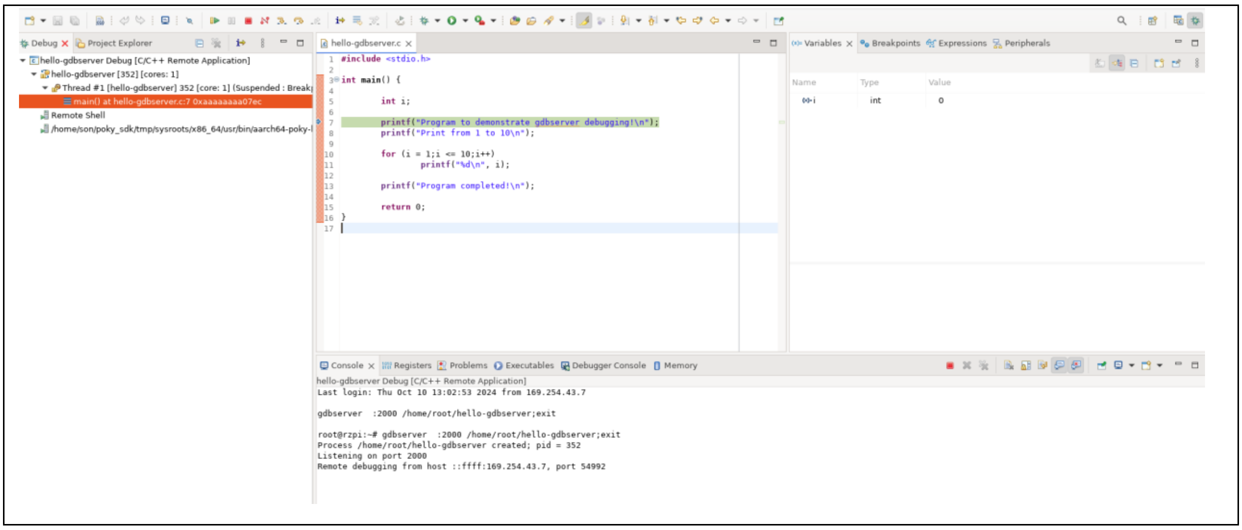
□ Automatically debug forked processes (Note: Requires Multi Process GDB)

Tracepoint mode: Normal

Figure 70. Configuring the GDB debugger in Eclipse debug configurations

5. Start the Debugging Session:

- After configuring the debug settings, click Apply and then Debug.
- Eclipse will attempt to connect to the GDB server running on the target device.
- If the connection is successful, it will be possible to set breakpoints, step through the code, and inspect variables just as in a local debugging session.



```
hello-gdbserver Debug [C/C++ Remote Application]
  hello-gdbserver [352] [cores: 1]
    Thread #1 [hello-gdbserver] 352 [core: 1] (Suspended : Breakp
      main() at hello-gdbserver.c:7 0xaaaaaaaa07ec
  Remote Shell
  /home/son/poky_sdk/tmp/sysroots/x86_64/usr/bin/aarch64-poky-l
```

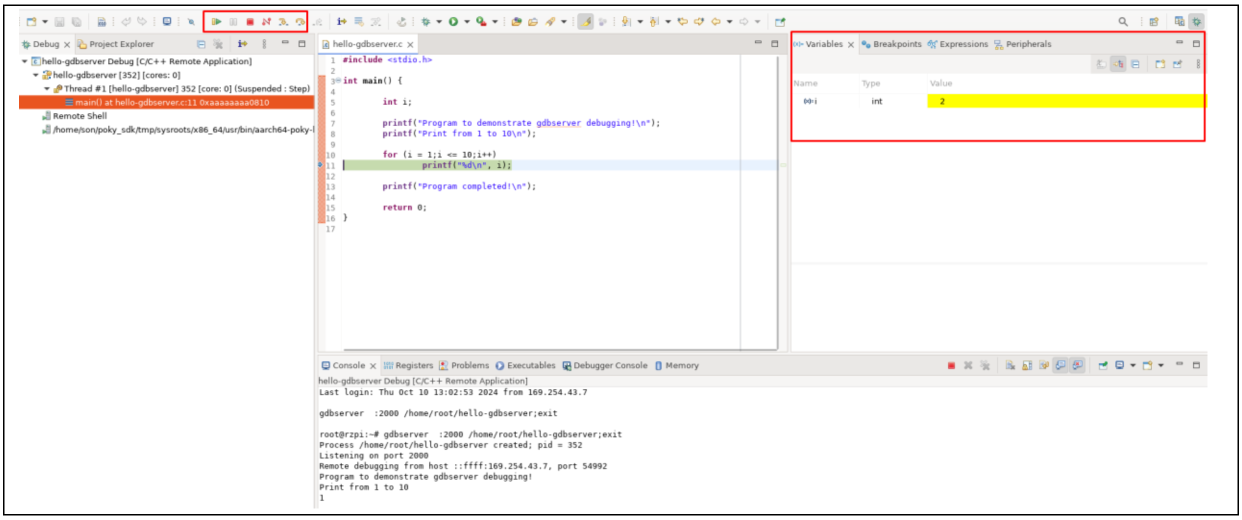
```c
#include <stdio.h>

int main() {

        int i;

        printf("Program to demonstrate gdbserver debugging!\n");
        printf("Print from 1 to 10\n");

        for (i = 1;i <= 10;i++)
                printf("%d\n", i);

        printf("Program completed!\n");

        return 0;
}
```

Name	Type	Value
i	int	0

```
hello-gdbserver Debug [C/C++ Remote Application]
Last login: Thu Oct 10 13:02:53 2024 from 169.254.43.7

gdbserver  :2000 /home/root/hello-gdbserver;exit

root@rzpi:~# gdbserver  :2000 /home/root/hello-gdbserver;exit
Process /home/root/hello-gdbserver created; pid = 352
Listening on port 2000
Remote debugging from host ::ffff:169.254.43.7, port 54992
```

Figure 71. Start the debugging session in Eclipse

Press F5 to step into, F6 to step over, or F8 to resume and monitor the variables.



```
hello-gdbserver Debug [C/C++ Remote Application]
  hello-gdbserver [352] [cores: 0]
    Thread #1 [hello-gdbserver] 352 [core: 0] (Suspended : Step)
      main() at hello-gdbserver.c:11 0xaaaaaaaa0810
  Remote Shell
  /home/son/poky_sdk/tmp/sysroots/x86_64/usr/bin/aarch64-poky-l
```

```
#include <stdio.h>

int main() {

        int i;

        printf("Program to demonstrate gdbserver debugging!\n");
        printf("Print from 1 to 10\n");

        for (i = 1;i <= 10;i++)
                printf("%d\n", i);

        printf("Program completed!\n");

        return 0;
}
```

Name	Type	Value
i	int	2

```
hello-gdbserver Debug [C/C++ Remote Application]
Last login: Thu Oct 10 13:02:53 2024 from 169.254.43.7

gdbserver  :2000 /home/root/hello-gdbserver;exit

root@rzpi:~# gdbserver  :2000 /home/root/hello-gdbserver;exit
Process /home/root/hello-gdbserver created; pid = 352
Listening on port 2000
Remote debugging from host ::ffff:169.254.43.7, port 54992
Program to demonstrate gdbserver debugging!
Print from 1 to 10
1
```

Figure 72. Go through each step in the debug mode in Eclipse

The path of the compiler may need to be adjusted to reflect the specific system configuration.

13.4 Postmortem Analysis Example

This section provides an overview of postmortem analysis, a critical process for diagnosing application crashes by examining core dump files. It details how developers can analyze these core dumps to pinpoint the exact lines of code that led to an error, allowing for effective troubleshooting and resolution of issues.

13.4.1 Postmortem Analysis on CLI

This subsection describes how to perform a postmortem analysis using the command-line interface (CLI). It emphasizes the steps for loading core dump files with CLI tools, enabling developers to navigate directly to the lines of code where errors occurred. The section highlights the efficiency of command-line tools for diagnosing issues quickly.

1. Create a simple C program that intentionally causes a segmentation fault. For example, the file name `segfault_example.` has the following content:

```
#include <stdio.h>

int main() {
        int *ptr = NULL;

        printf("Attempting to dereference a NULL pointer...\r\n");

        *ptr = 42;

        return 0;
}
```

2. Source the environment and compile the segfault_example.c program.

```
renesas@builder-pc:~$ source ~/esdk/5.1.4/environment-setup-cortexa55-poky-linux
SDK environment now set up; additionally you may now run devtool to perform
development tasks.
Run devtool --help for further details.
renesas@builder-pc:~/remote-debugging/segfault_program$ $CC $CFLAGS
segfault_example.c -o segfault_example
```

3. Transfer the program to RZ/G2L-SBC.

```
renesas@builder-pc:~/remote-debugging/segfault_program$ scp segfault_example
root@169.254.43.30:/home/root
```

4. Ensure that the system allows core dumps. To set the core dump size to unlimited, run the following command:

```
root@rz-cmn:~# ulimit -c unlimited
root@rz-cmn:~# echo core | tee /proc/sys/kernel/core_pattern
```

5. Run the program to generate a core dump file or use remote debugging to obtain it.

```
root@rz-cmn:~# ./segfault_example
Attempting to dereference a NULL pointer...
Segmentation fault (core dumped)
```

When the segmentation fault occurs, a core dump file will be generated, usually named core or core.<pid>, for example, core.880 in my case.

```
root@rz-cmn:~# ls core*
core.880
```

Transfer the core dump file back to your host machine.

6. Using GDB to analyze the core dump file. Return to the remote machine and use the following command.

```
renesas@builder-pc:~/remote-debugging/segfault_program$ aarch64-poky-linux-gdb
</path/to/local_program> </path/to/core/dump/file>
```

For example:

```
renesas@builder-pc:~/remote-debugging/segfault_program$ aarch64-poky-linux-gdb
segfault_example core.810

GNU gdb (GDB) 15.1
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux --target=aarch64-poky-linux".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
Reading symbols from segfault...
[New LWP 810]

warning: Could not load shared library symbols for 2 libraries, e.g.
/lib64/libc.so.6.
Use the "info sharedlibrary" command to see the complete listing.
Do you need "set solib-search-path" or "set sysroot"?
Core was generated by `./segfault.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000aaaae3340794 in main () at segfault_example.c:8
--Type <RET> for more, q to quit, c to continue without paging--
8               *ptr = 42;
(gdb)
(gdb) quit
```

The segmentation fault occurred because the program attempted to dereference a NULL pointer at line 8 in segfault_example.c, where it tried to assign 42 to *ptr, resulting in an invalid memory access.

13.4.2 Postmortem Analysis on Visual Studio Code

In this subsection, the process of analyzing core dump files using Visual Studio Code (VSCode) is explored. It explains how to load core dumps and utilize VSCode's debugging features to automatically jump to the lines of code that caused the application to crash.

If subsection Remote Debugging on Visual Studio Code has been followed, the next step is to analyze the core dump file. A key addition is to include a line in the `launch.json` file that specifies the path to the core dump file for analysis. This adjustment enables full utilization of VSCode's features for inspecting the crash details.

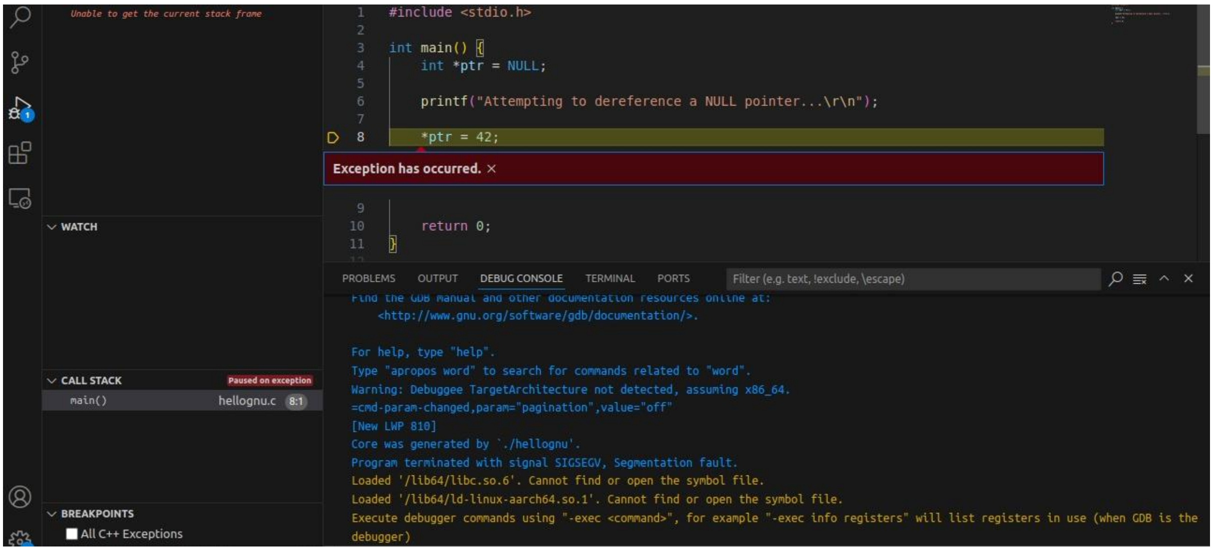
For example, in launch.json, add the following line to specify the core dump file path:

```
"coreDumpPath": "</path/to/core/dump/file>,
```

Here’s a complete example of a launch.json

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "gdb",
            "type": "cppdbg",
            "request": "launch",
            "program": "/home/renesas/remote-debugging/program/segfault_example",
            "cwd": "${workspaceFolder}",
            "stopAtEntry": true,
            "stopAtConnect": true,
            "MIMode": "gdb",
            "miDebuggerPath":
"/home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-gdb",
            "miDebuggerServerAddress": "169.254.43.30:2000",
            "coreDumpPath": "/home/renesas/remote-debugging/segfault/core.810",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        }
    ]
}
```

After running the debugging session with the core dump file, the IDE (Visual Studio Code) automatically points to the exact line in the source code where the crash occurred.



Unable to get the current stack frame

```
#include <stdio.h>

int main() {
    int *ptr = NULL;

    printf("Attempting to dereference a NULL pointer...\r\n");

    *ptr = 42;
```

Exception has occurred.

```

    return 0;
}
```

```
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
Warning: Debuggee TargetArchitecture not detected, assuming x86_64.
=cmd-param-changed,param="pagination",value="off"
[New LWP 810]
Core was generated by `./hellognu'.
Program terminated with signal SIGSEGV, Segmentation fault.
Loaded '/lib64/libc.so.6'. Cannot find or open the symbol file.
Loaded '/lib64/ld-linux-aarch64.so.1'. Cannot find or open the symbol file.
Execute debugger commands using "-exec <command>", for example "-exec info registers" will list registers in use (when GDB is the
debugger)
```

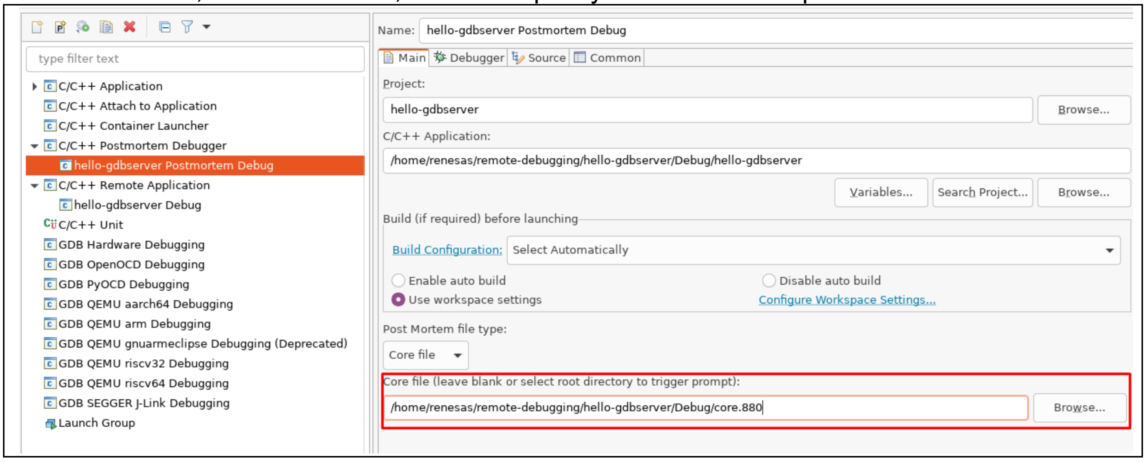
Figure 73. Starting analysis of the Core dump file in VSCode debug mode

13.4.3 Postmortem Analysis on Eclipse

This subsection describes postmortem analysis using Eclipse IDE. Similar to Visual Studio Code, Eclipse allows loading core dump to inspect the application's state at the time of a crash.

1. Configure Eclipse to connect to the GDB Server:

- In Eclipse, go to the Run menu and select Debug Configurations.
- Under the Debugger tab, select C/C++ Postmortem Debugger.
- In the Main tab, in Core file field, click and specify where core dump file is.



type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Container Launcher
- C/C++ Postmortem Debugger
  - hello-gdbserver Postmortem Debug
- C/C++ Remote Application
  - hello-gdbserver Debug
- C/C++ Unit
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB PyOCD Debugging
- GDB QEMU aarch64 Debugging
- GDB QEMU arm Debugging
- GDB QEMU gnuarmeclipse Debugging (Deprecated)
- GDB QEMU riscv32 Debugging
- GDB QEMU riscv64 Debugging
- GDB SEGGER J-Link Debugging
- Launch Group

Name: hello-gdbserver Postmortem Debug

Main | Debugger | Source | Common

Project: hello-gdbserver

C/C++ Application: /home/renesas/remote-debugging/hello-gdbserver/Debug/hello-gdbserver

Build (if required) before launching

Build Configuration: Select Automatically

○ Enable auto build ○ Disable auto build

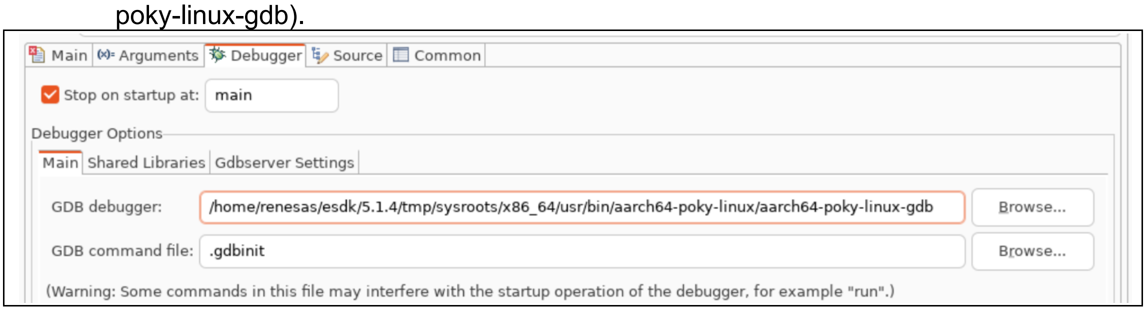
● Use workspace settings Configure Workspace Settings...

Post Mortem file type: Core file

Core file (leave blank or select root directory to trigger prompt): /home/renesas/remote-debugging/hello-gdbserver/Debug/core.880

Figure 74. Specifying the Core File in Eclipse Debugger Settings

- In the Debugger tab:
  - In GDB Debugger: Provide the path to your cross-compiled GDB (For example, /home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb).



Main | Arguments | Debugger | Source | Common

☑ Stop on startup at: main

Debugger Options

Main | Shared Libraries | Gdbserver Settings

GDB debugger: /home/renesas/esdk/5.1.4/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb

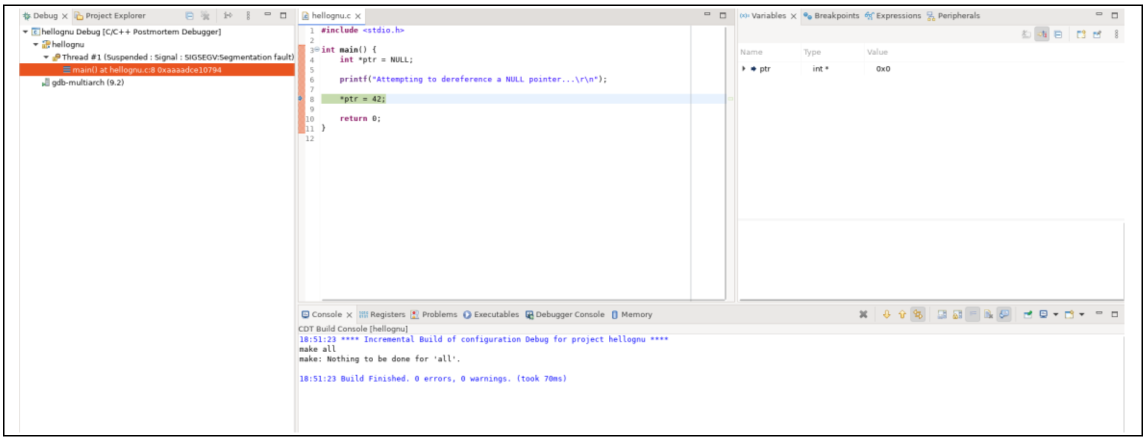
GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

Figure 75. Specifying the GDB Debugger in Eclipse Debugger settings

2. Start the Debugging Session:

- Once the debugging session starts, Eclipse will show the line of code that caused the segmentation fault, along with the call stack.



hellognu Debug [C/C++ Postmortem Debugger]
hellognu
Thread #1 (Suspended : Signal : SIGSEGV:Segmentation fault)
main() at hellognu.c:8 0xaaaadce10794
gdb-multiarch (9.2)

```
#include <stdio.h>

int main() {
    int *ptr = NULL;

    printf("Attempting to dereference a NULL pointer...\r\n");

    *ptr = 42;

    return 0;
}
```

Name	Type	Value
ptr	int *	0x0

CDT Build Console [hellognu]

```
18:51:23 **** Incremental Build of configuration Debug for project hellognu ****
make all
make: Nothing to be done for 'all'.

18:51:23 Build Finished. 0 errors, 0 warnings. (took 70ms)
```

Figure 76. Starting analysis of the Core dump file in Eclipse debug mode

- Inspect the values of variables at that point in time by hovering over them or using the Variables view.
- Utilize the Expressions view to evaluate any expressions or check the state of specific variables.
- Navigate through the call stack to see the sequence of function calls leading to the crash. This can provide insight into how the program reached the faulting line.

14. Functional Overview

This chapter presents a high-level overview of the RZ/G2L, RZV2L and RZV2H family of MPUs. It begins with a general introduction to the MPU architecture, followed by a detailed outline of the primary subsystems. Each subsection is designed to provide developers with foundational context prior to engaging in board-level integration or initiating software bring-up procedures.

14.1 MPU Architecture

This section describes the architectural foundation common across the RZ family. The MPU architecture defines the CPU clusters, interconnects, memory hierarchy, and peripheral integration that form the baseline of every device in the series.

14.1.1 RZ/G2L

The RZ/G2L MPU is a feature-packed SoC (System on Chip) that can support a variety of applications. Below is an overview of the MPU.

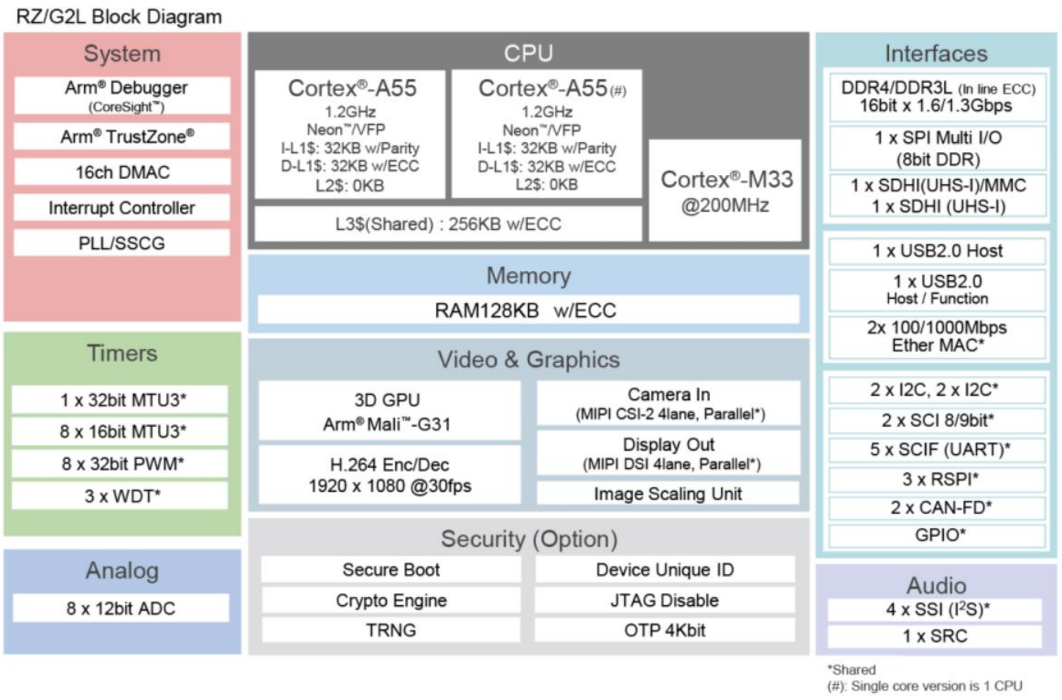


Figure 77. RZ/G2L Overview

14.1.2 RZ/V2L

RZ/V2L pairs a 1.2-GHz Arm® Cortex®-A55 with Renesas' DRP-AI vision engine (a combination of DRP and AI-MAC). It supports 16-bit DDR3L/DDR4, integrates an Arm Mali-G31 GPU, and includes an H.264 video codec.

The DRP-AI block delivers high inference efficiency, typically avoiding heatsinks or fans. Alongside neural-net acceleration, it provides real-time image-processing functions, e.g., color correction and noise reduction, so many camera systems can run without an external ISP. This keeps BOM and power low across consumer, industrial, and retail/POS designs.

RZ/V2L is package and pin compatible with RZ/G2L, enabling a straightforward upgrade path to add AI capability with minimal board changes and low migration cost.

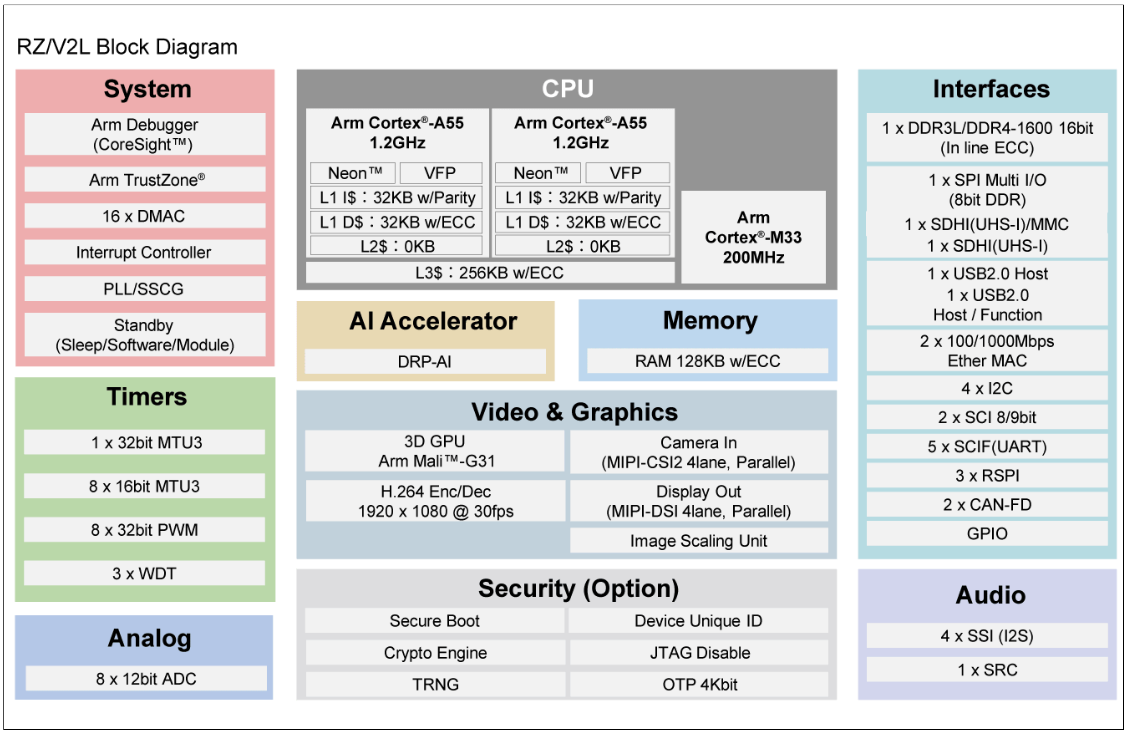


Figure 78: RZ/V2L Overview

14.1.3 RZ/V2H

The RZ/V2H is a high-performance MPU in the RZ/V series, optimized for AI inference and advanced multimedia while maintaining compatibility with the common RZ software stack. It expands on the G2L/V2L baseline with larger compute resources and dedicated accelerators.

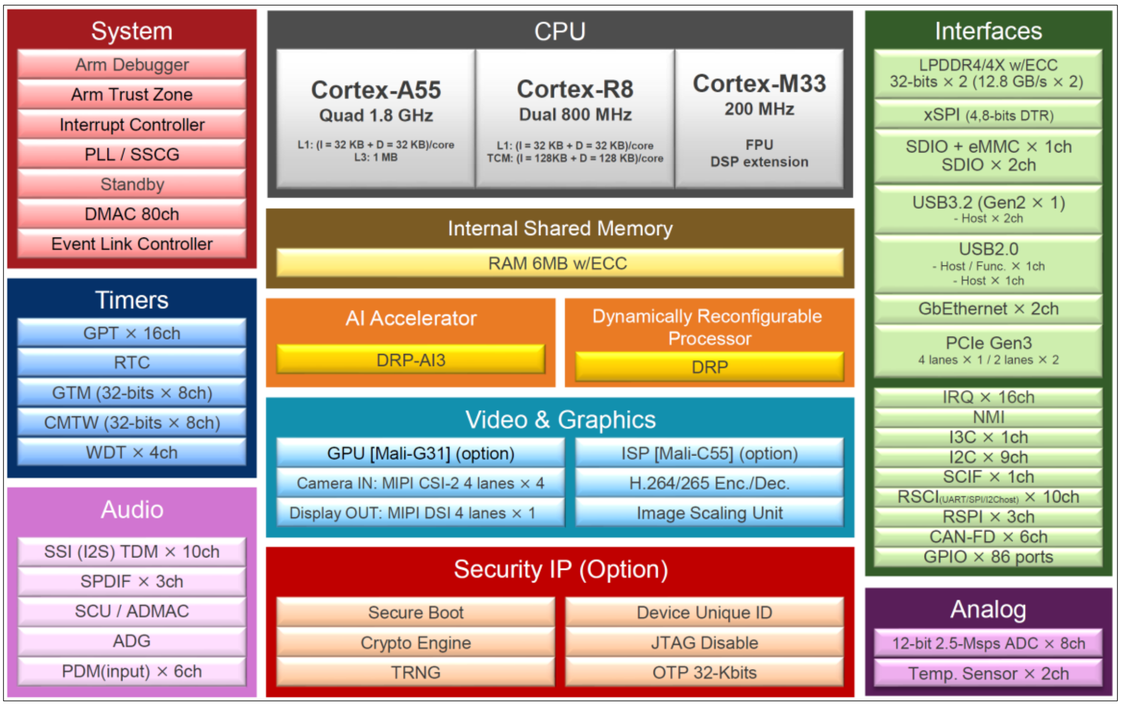


Figure 79: RZ/V2H Overview

14.2 RZ/G2L-SBC Board

This section delves into the functional and design aspects of the RZ/G2L-SBC. The image below highlights the key hardware components in the RZ/G2L SBC design.

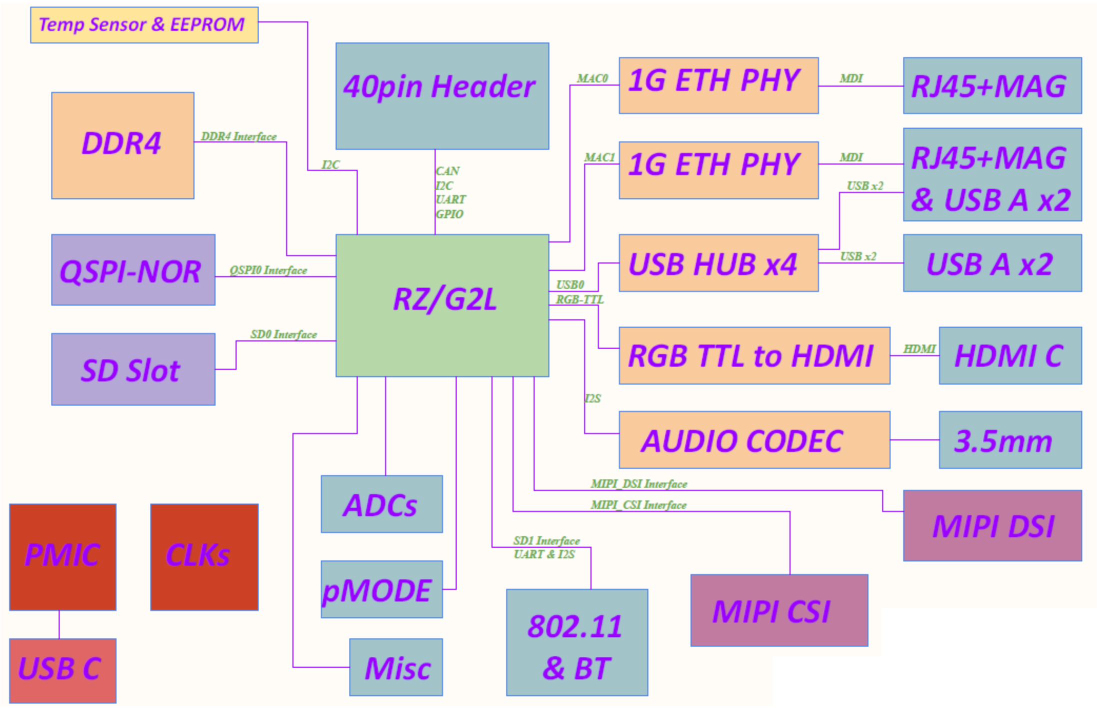


Figure 80: RZ/G2L-SBC System Overview

Table 18. Main Components on RZ/G2L-SBC

Component Number	Component Name	Type (Manufacturer)
U1	Temperature Sensor Digital, Local -55°C ~ 125°C 11 b 8-HWSON (2x3)	CAT34TS02 (Onsemi)
U2	USB Controller	UPD720115K8-611-BAK-A-ND (Renesas Electronics)
U3	MPU RZ/G2L	R9A07G044L23GBG (Renesas Electronics)
U4	DDR4 SDRAM 512MB	IS43QR16256A-093PBLI-TR (ISSI)
U5	Ethernet Phy 10BASE-TE, 100BASE-TX, 1GBASE-T	PEF7071VV16 (MaxLinear)
U6	VersaClock® Programmable Clock Generator	5P35023B-000NLGI8 (Renesas Electronics)
U7	HDMI Transmitter	SiI9022A/4A – QFN (SiliconImage)
U8	PMIC	RAA215300 (Renesas Electronics)
U9	AND GATE 2IN SOT-23-5 Vcc 1.65V to 5.5V	7UL1G08FS (Toshiba)
U10	Ethernet Phy 10BASE-TE, 100BASE-TX, 1GBASE-T	PEF7071VV16 (MaxLinear)
U11	Audio Codec with Advanced Accessory Detect	DA7219 (Renesas Electronics)

U12	Dual USB Port Power Supply Controller - Covering the Industrial Temperature Range of -40C to +85C	ISL61852FIRZ (Renesas Electronics)
U13	QSPI Flash 512MBIT SPI/QUAD 8WSON	S25FS512SDSNFB010 (Infineon)
U14	Dual USB Port Power Supply Controller - Covering the Industrial Temperature Range of -40C to +85C	ISL61852FIRZ (Renesas Electronics)
M1	Integrated 802.11 b/g/n Wi-Fi Module	iWi-L-WB (Laird)
Y1	Crystal resonator for XIN	XRCGB24M000F0L00R0 (Murata)
Y2	Crystal resonator for XIN	ST3215SB32768H5HPWAA (Kyocera-AVW)

Table 19. Primary connectors on RZ/G2L-SBC

Components Number	Component Name	Type (Manufacturer)
J1	USB 2 & 3	USB-A-D-RA (Adam Tech)
J2	PMOD	PPPC062LFBN-RC (Sullins)
J3	40-Pin Header (Raspberry Pi 3B compliant)	-
J4	USB 0 &1, 10/100/1000 Ethernet 2	YKGU-6101NL (Ingke)
J5	MIPI-CSI	1-1734248-5 (TE Connectivity)
J6	MIPI-DSI	1-1734248-5 (TE Connectivity)
J7	10/100/1000 Ethernet 1	YKGD-8069NL (Ingke)
J8	Audio I/O (Speaker/Microphone)	ASJ-192-Y (Adam Tech)
J9	Mini-HDMI	10029449-001RLF (FCI)
J10	USB-Type-C Power Input	C-ARA1-AK515 (CNC Tech)
J11	20-pin JTAG connector	3221-10-0300-00 (CNC Tech)
J12	Expansion Connecter to Display adapter & boot strapping pins	528850274 (Molex)
J13	Expansion Connecter to Display adapter & boot strapping pins	528850274 (Molex)
P1	microSD card slot	MEM2051-00-195-00-A (GCT)

14.2.1 Overview

The RZ/G2L-SBC is a power-efficient, graphics-enabled development board in a popular single-board computer format with well-supported expansion interfaces. This Renesas RZ/G2L processor-based platform is ideal for developing cost-efficient HMI, industrial, robotics, and a range of energy-efficient design applications. The RZ/G2L processor has two 1.2GHz Arm® Cortex®-A55 cores, a 200MHz Cortex-M33 core, a MALI 3D GPU, and an Image Scaling Unit. This processor SoC is equipped with an on-chip plus H.264 video (1920 x 1080) encode/decode function in silicon, making it ideal for implementing cost-effective embedded vision and display applications.

The RZ/G2L-SBC is engineered in a compact Raspberry Pi form factor with a versatile set of expansion interfaces, including Gigabit Ethernet, 801.11ac Wi-Fi, four USB 2.0 host ports, a MIPI DSI display with touch and CSI camera interfaces, a CANFD interface, a PMOD interface, a Pi-HAT-compatible 40-pin expansion header, and two expansion sockets for a daughter card.

The board supports analog audio applications via an audio codec and a stereo headphone jack. It also pins out five 12-bit ADC inputs for interfacing with analog sensors through an expansion module (not included). A 5V input power is sourced via a USB-C connector and managed via a single-chip Renesas RAA215300 PMIC device.

The onboard memory includes 1GB DDR4, 64 MiB QSPI NOR flash memory, and a microSD slot for removable boot media.

Software enablement includes a Linux 6.10 kernel-based BSP, along with reference designs showcasing demo implementations of HMI applications. Onboard 10-pin JTAG/SWD mini-SMT header (unpopulated) and 40-pin GPIO header enable the use of an external debugger and USB-serial cable

14.2.2 Physical View

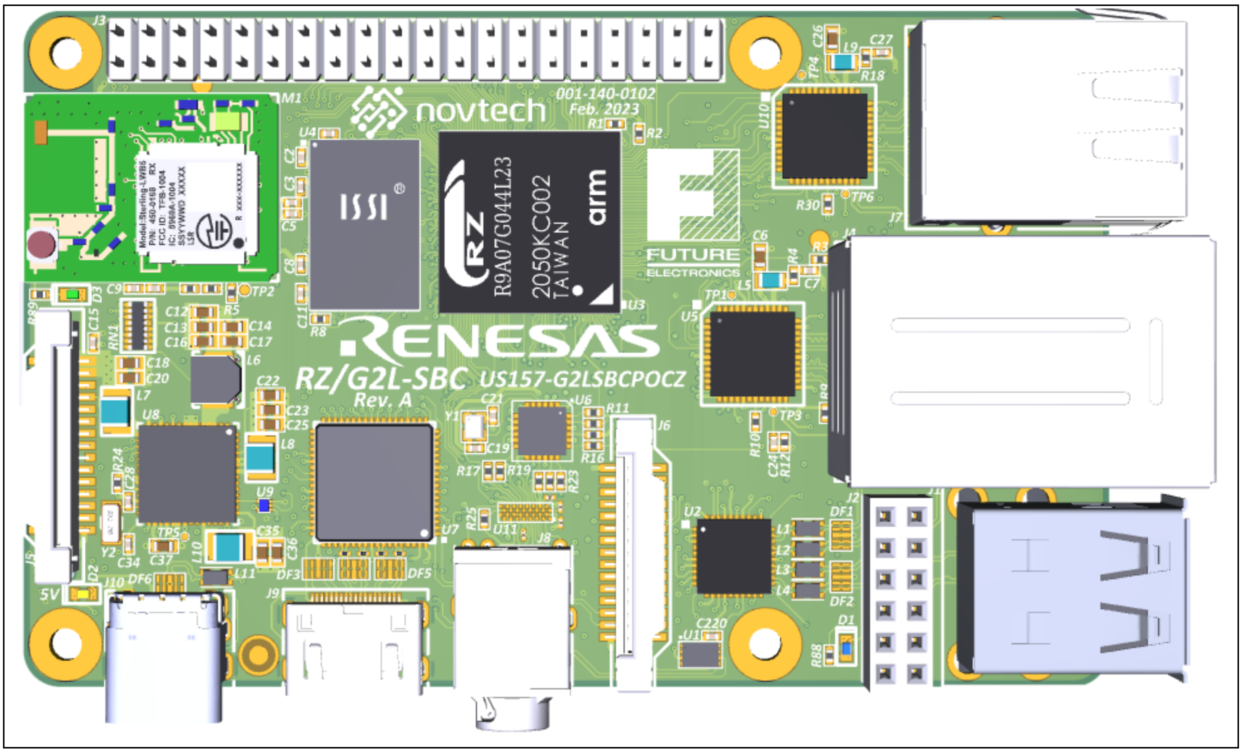
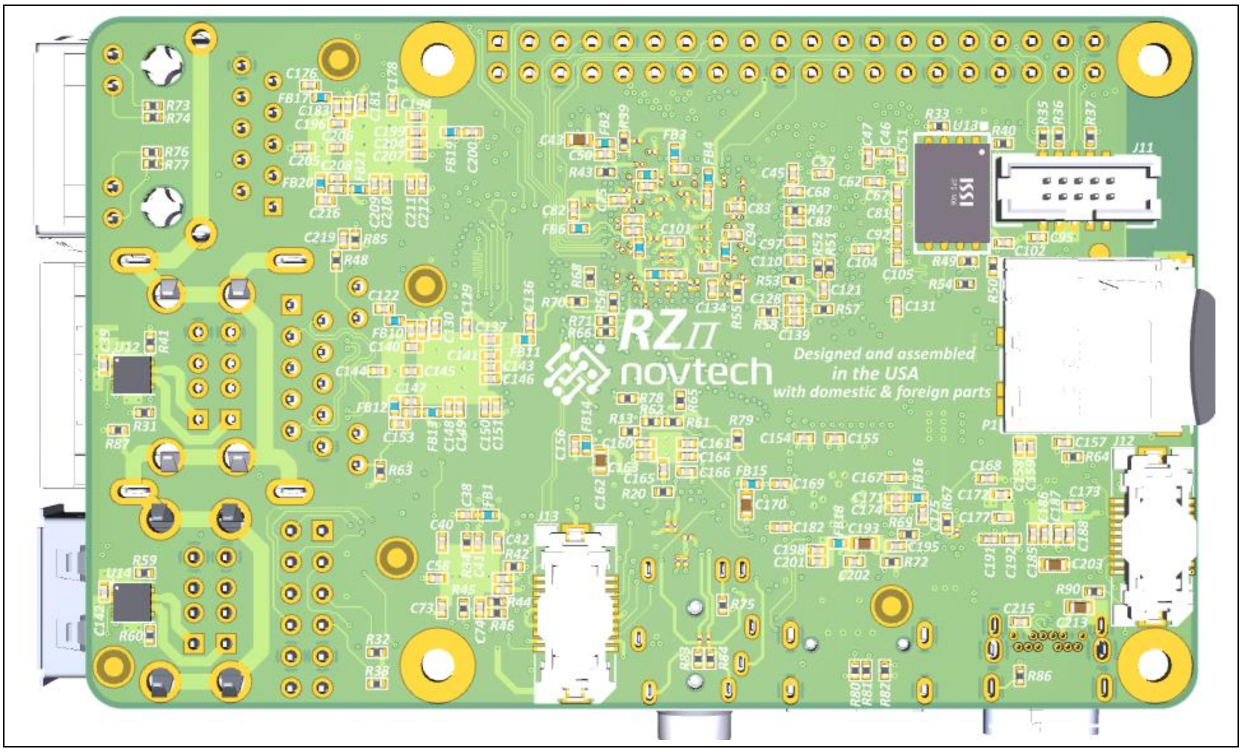


Figure 81. Top-side view of the RZ/G2L-SBC



RZ
novtech
Designed and assembled in the USA with domestic & foreign parts
J11
J12
J13
U12
U13
U14
P1

Figure 82. Bottom side of the RZ/G2L-SBC

14.2.3 Overview of Connectors

Given below is the basic positioning of the top-level connectors.

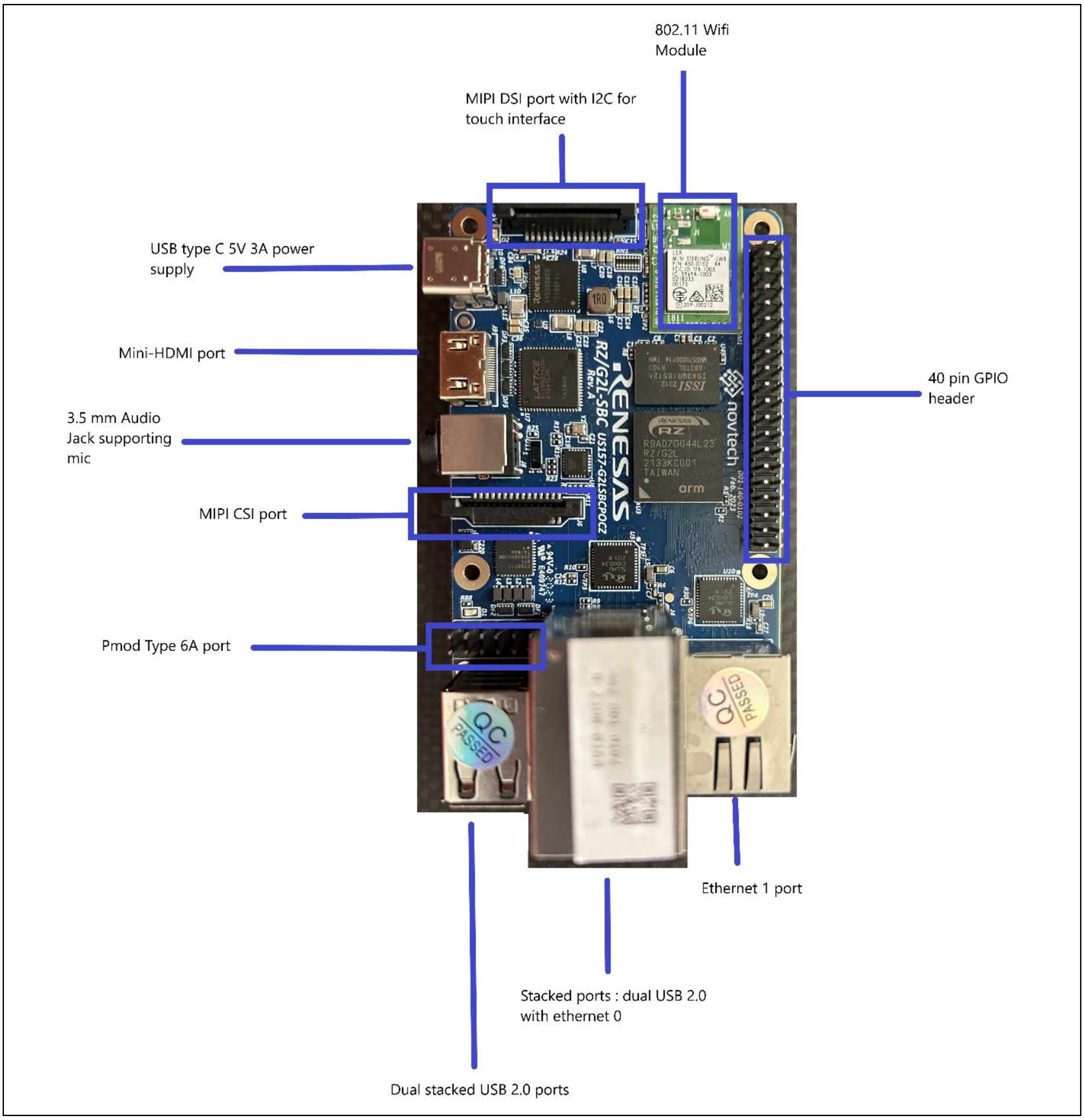

802.11 Wifi Module
MIPI DSI port with I2C for touch interface
USB type C 5V 3A power supply
Mini-HDMI port
3.5 mm Audio Jack supporting mic
MIPI CSI port
Pmod Type 6A port
40 pin GPIO header
Ethernet 1 port
Stacked ports : dual USB 2.0 with ethernet 0
Dual stacked USB 2.0 ports

Figure 83. RZ/G2L-SBC top side connectors.

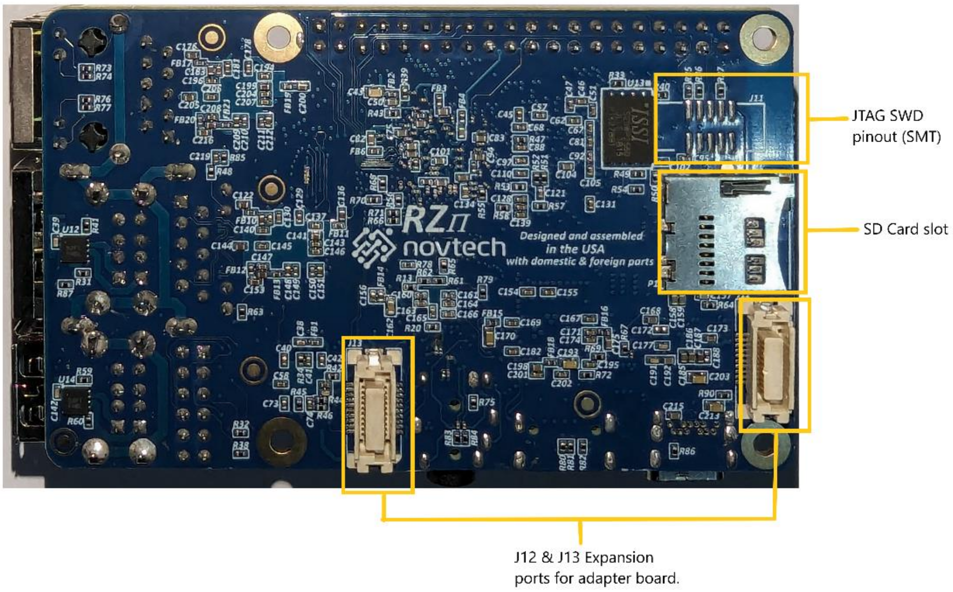


Figure 84. RZ/G2L-SBC Bottom view connectors.

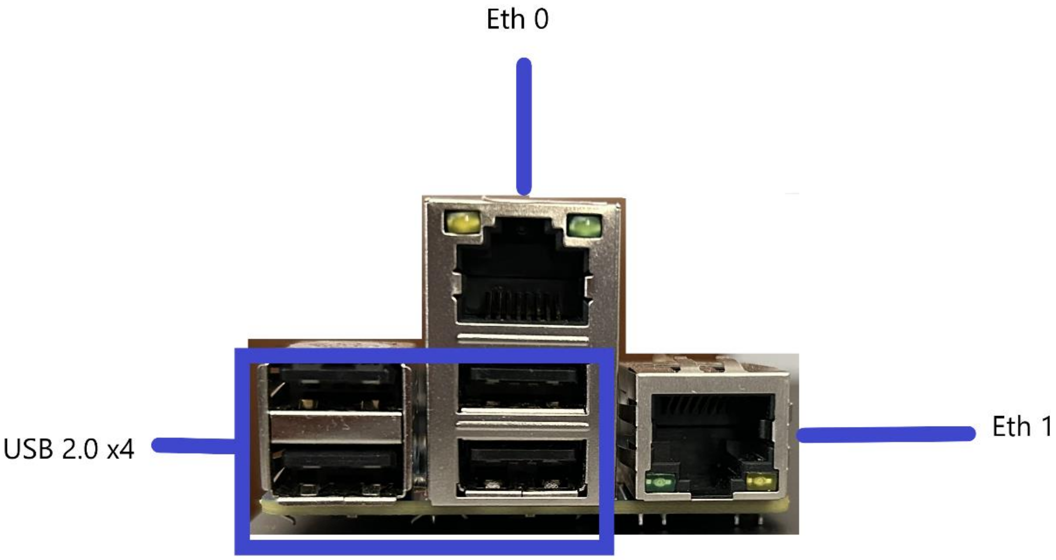


Figure 85. RZ/G2L-SBC side view I/O ports.

14.2.4 Power Supply

This section delves into the RZ/G2L-SBC's power supply architecture. The RZ/G2L-SBC uses a simple design with a 5V supply as the single external power source.

14.2.4.1 USB Type-C Power

This board has one USB Type-C receptacle for power input with USB Power Delivery. The USB Type–C power connector is meant to connect to a 5V power supply. The RZ/G2L-SBC requires a minimum of 3A power to prevent brownouts. However, we recommend a 4.5 -5A power supply as several ports support peripherals that consume substantial power.

14.2.4.2 Power Rails

Given below is the basic power supply design. It is a simple design that uses an input power supply from USB-C or one of the routed pins marked as 5V in the 40-pin GPIO or the adapter board and routes it through a series of converters to generate different power lines.

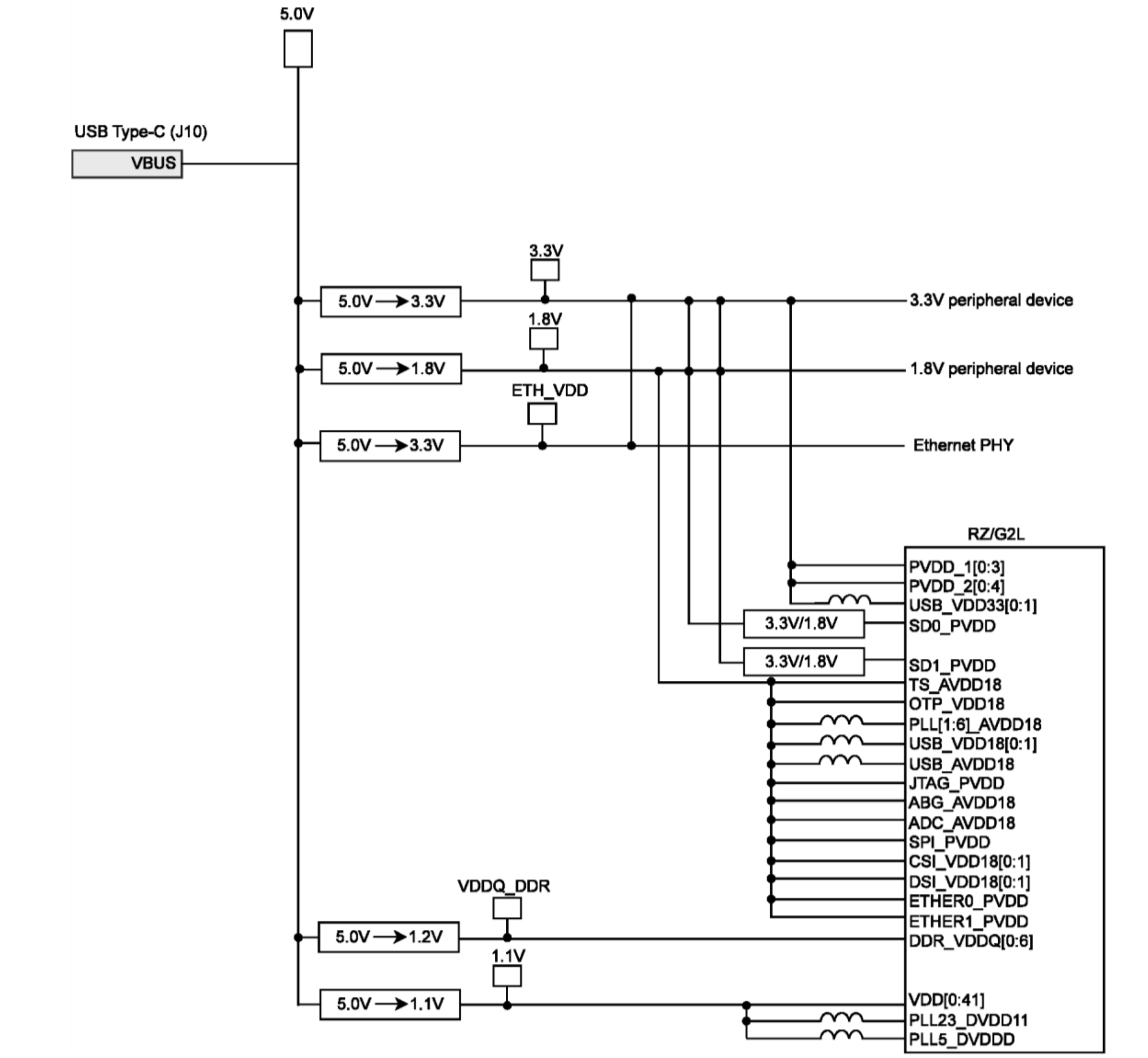


Figure 81. Power supply rails.

The Input power of 5V is used to generate five independent power lines:

- Two independent 3.3 V lines for peripherals and Ethernet.
- A 1.8V master supply line
- A 1.2V master supply line
- A 1.1V master supply line

The 1.2V line is used by the RZ/G2L SoC and the DDR4 SDRAM, while the 1.1V line is exclusively used by the RZ/G2L SoC. The RZ/G2L also draws power to its internal IP blocks from the 1.8V line.

This design is aimed at simplicity and hence omits the use of any power and reset switches. POR behavior is strictly controlled by the PMIC and its passives.

14.2.4.3 Power Supply Regulation

The power supply is regulated by Renesas RAA215300, a low-cost nine-channel PMIC IC.

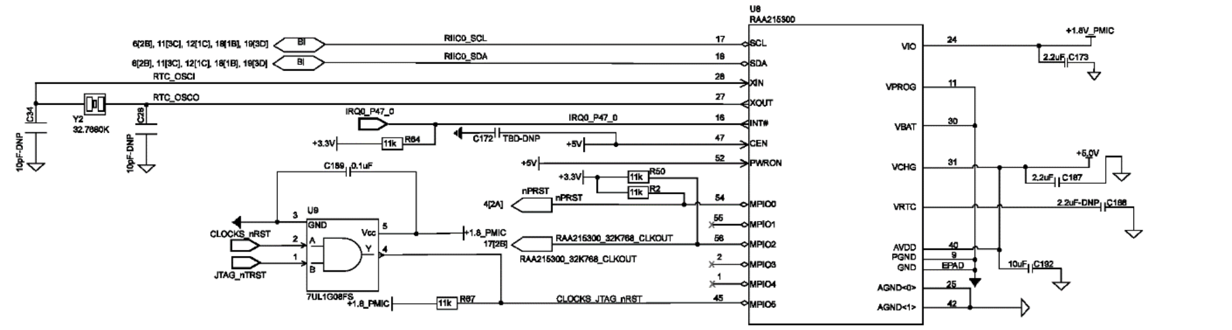


Figure 86. Block Diagram of Power Supply Regulation using RAA215300.

14.2.5 Power Management Integrated Circuit- PMIC

All LDOs are cycled as per the POR cycle. Any control is exercised by the RZ/G2L through the I2C interface. However, the LDOs are always turned on post-POR.

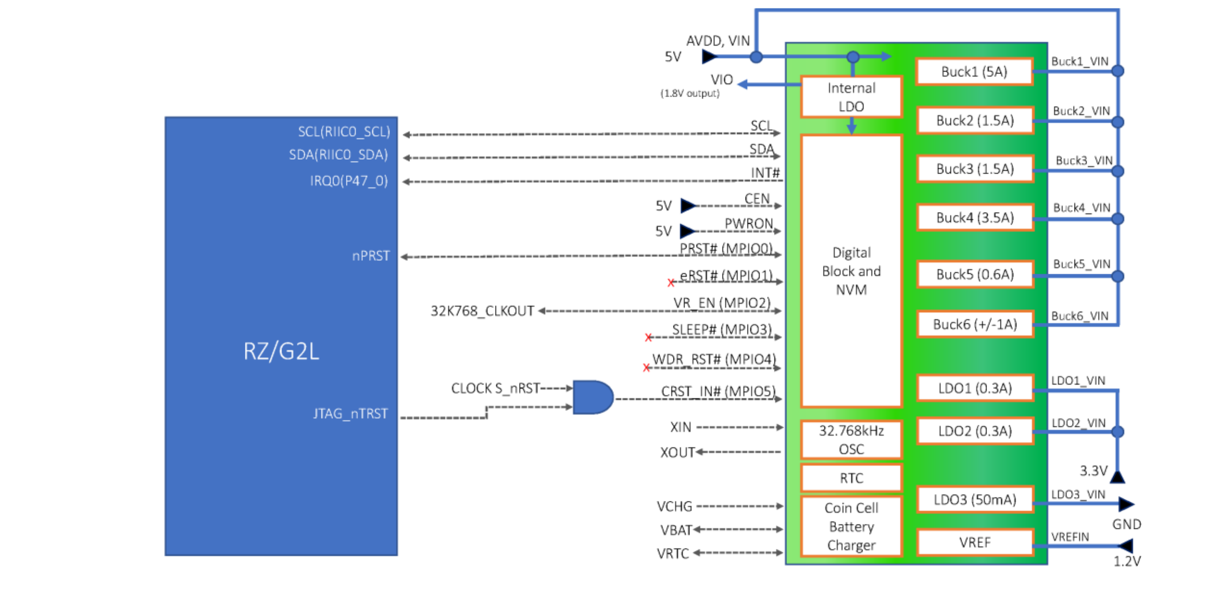


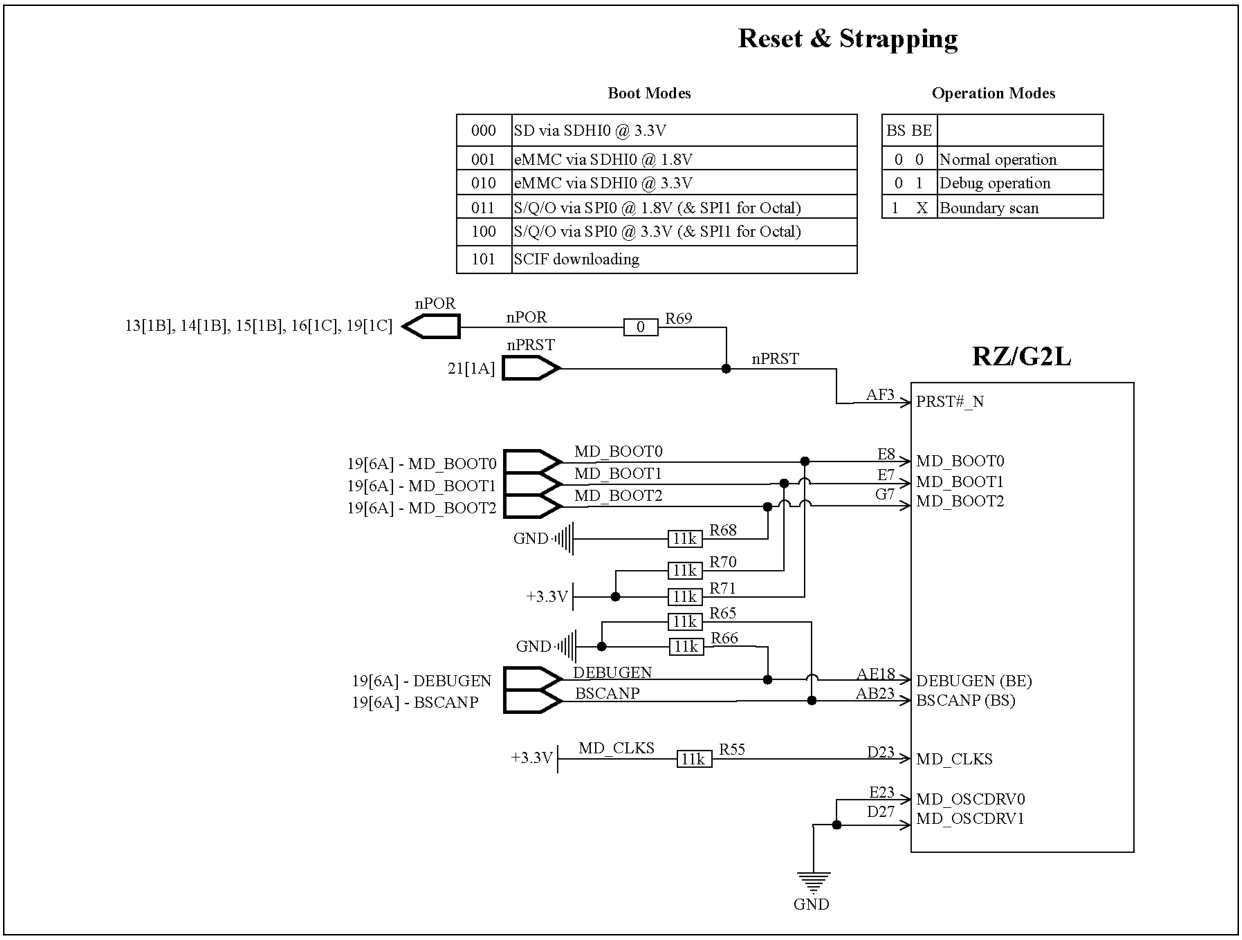
Figure 87. Block diagram of PMIC interface to RZ/G2L

14.2.6 RESET Control

The RZ/G2L-SBC has simplified POR behavior. It is by default set up to boot from QSPI0, which is achieved through external pull-up and pull-down resistors to a default code of 011. The default boot mode of 011 is for booting from QSPI0 but setting the operating voltage to 1.8V. The bootstrapping lines can be accessed externally through the J12 port at the bottom (through an adapter board). This makes it possible to alter the boot flow using these pins.

In addition to the boot order, the SoC has two more lines: DEBUGEN (BE) & BSCANP (BS). These lines control the boot mode, which can be JTAG boundary scan or debug mode. Figure 84. Reset Control Logic below shows all the necessary information.

Note**:** Debug mode and boundary scan are mutually exclusive. Only one of DEBUGEN / BSCANP can be active at a time. Do not enable both.



Boot Modes

000	SD via SDHI0 @ 3.3V
001	eMMC via SDHI0 @ 1.8V
010	eMMC via SDHI0 @ 3.3V
011	S/Q/O via SPI0 @ 1.8V (& SPI1 for Octal)
100	S/Q/O via SPI0 @ 3.3V (& SPI1 for Octal)
101	SCIF downloading

Operation Modes

BS BE	
0 0	Normal operation
0 1	Debug operation
1 X	Boundary scan

Figure 88. Reset Control Logic

14.2.7 Clock Configuration

The RZ/G2L-SBC design uses a Renesas VersaClock-3S as a singular programmable clock generator as the master clock source for the entire board. It drives the source clock for not just the RZ/G2L-SoC but all other devices that use an external clock input. This reduces the design complexity by reducing the use of passives and PLLs per peripheral while using a single 24 MHz crystal XTALL.

Notes:

1. MIPI DSI interface supports operations up to FHD@60 fps rates.
2. SD Interface supports UHS-I mode of 50MBps (SDR50) and 104MBps (SDR104)

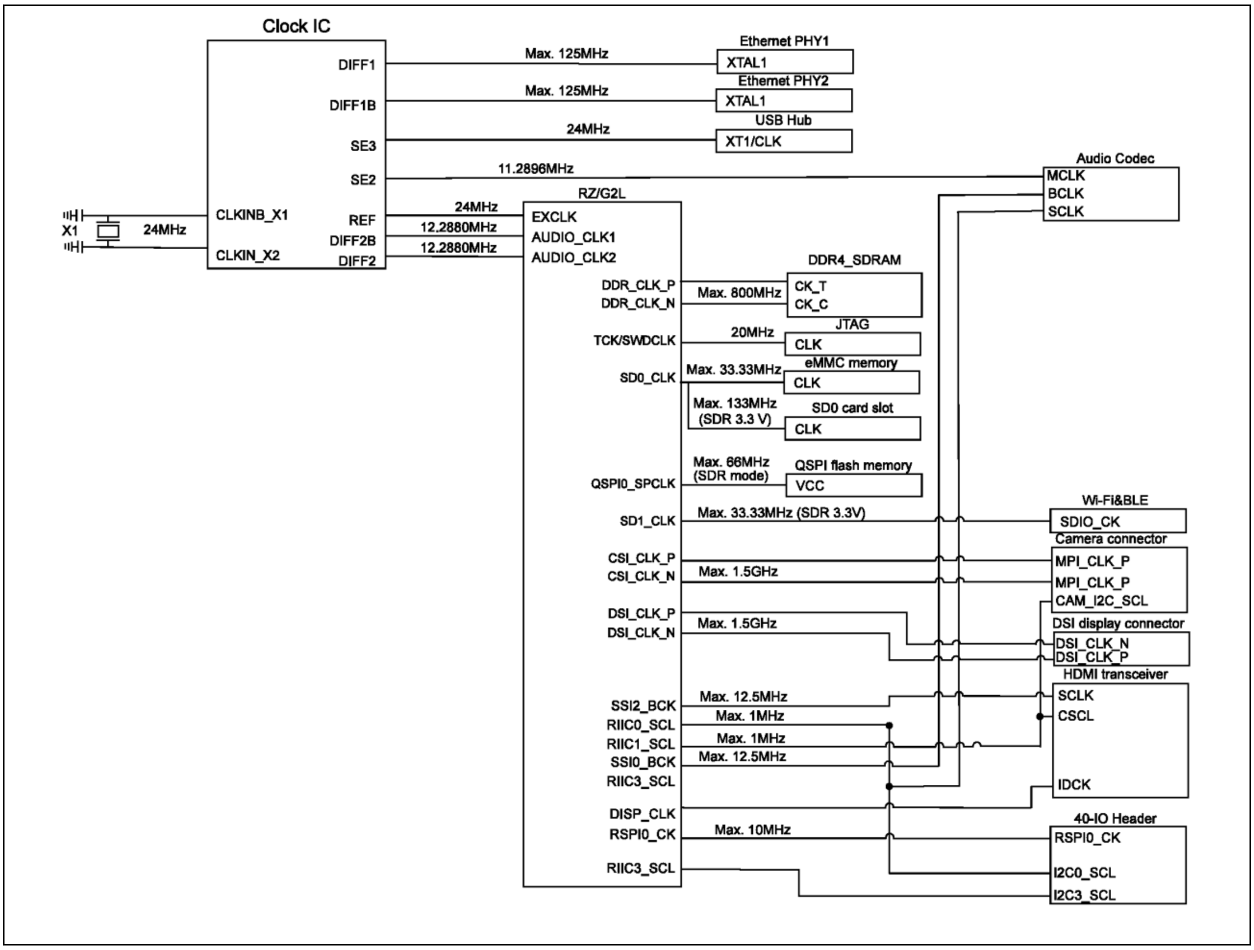


Figure 89. Block diagram of Clock interfacing.

14.2.8 Peripheral Interface

14.2.8.1 Gigabit Ethernet

The RZ/G2L-SBC comes with two Gigabit Ethernet ports. They are identified as Eth 0 and Eth 1 in the Linux environment. They are both gigabit-capable. The Gigabit Ethernet Interface is controlled by the Ethernet controller (E-MAC) that conforms to the definition of the MAC (Media Access Control) layer that is built into the RZ/G2L. The Ethernet clock is sourced from a clock generator connected to the Ethernet PHY.

This interface complies with IEEE802.3 PHY RGMII.

ETH0 is connected to PHY 2, and ETH1 is connected to PHY1. Take note of the order.

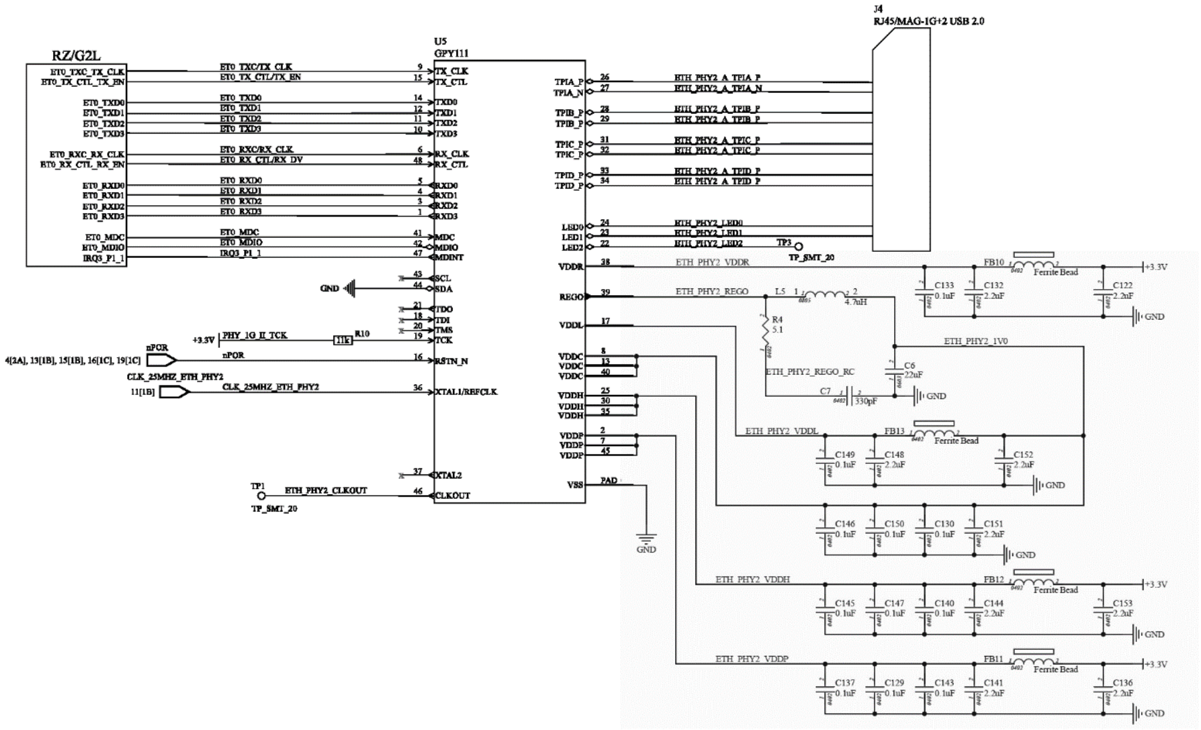


Figure 90. Ethernet 0 PHY interfacing.

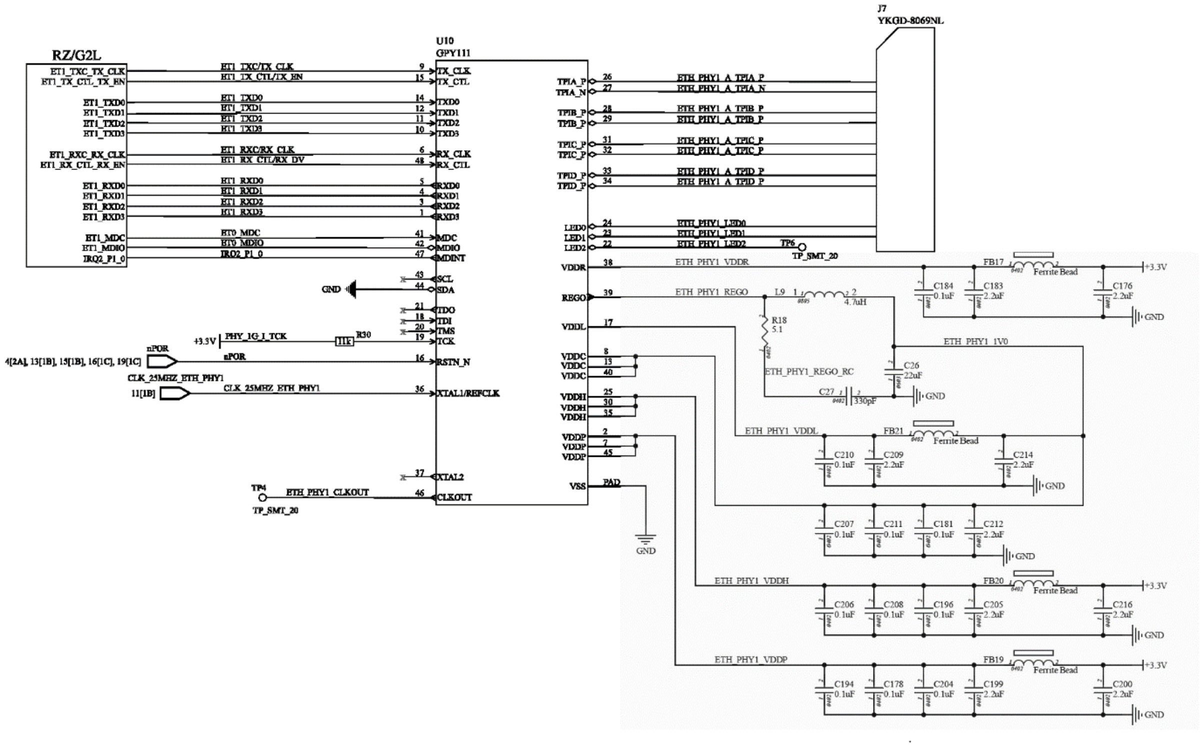


Figure 91. Ethernet 1 PHY interfacing.

14.2.8.2 USB 2.0 Ports

The SBC has 4 USB 2.0 ports, which are of type A. The primary USB hub is the Renesas UPD720115 (µPD720115), which is a 4-port hub conforming to USB battery charging specification version 1.2. It has one upstream port and four downstream ports. The USB hub is connected directly to the RZ/G2L SoC's USB 1 data ports. The RZ/G2L SoC has a single USB 2.0 Host Interface channel.

The USB 0 channel (OTG interface) is routed to the USB-C power supply port. However, the actual OTG lines are not connected, and only the data lines are routed to the USB-C port. When the board is powered through the 40-pin IO or the bottom expansion connectors, it frees up the USB-C port. It can then be used for connecting peripherals.

Note: The USB-C has not been tested as a peripheral interface so far.

The power supply to the four USB 2.0 ports downstream is controlled through two external power regulators: Renesas ISL6185. The ISL 6185 isolates and protects the internal circuit from the external USB peripheral while providing higher levels of 5V power through supply sourcing.

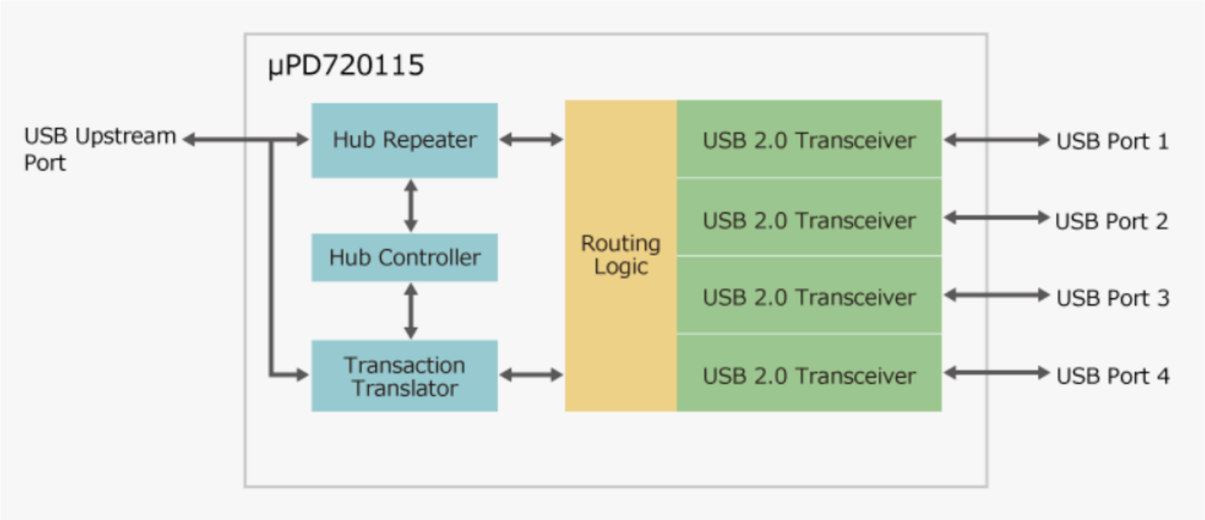


Figure 92. UPD720115 block diagram.

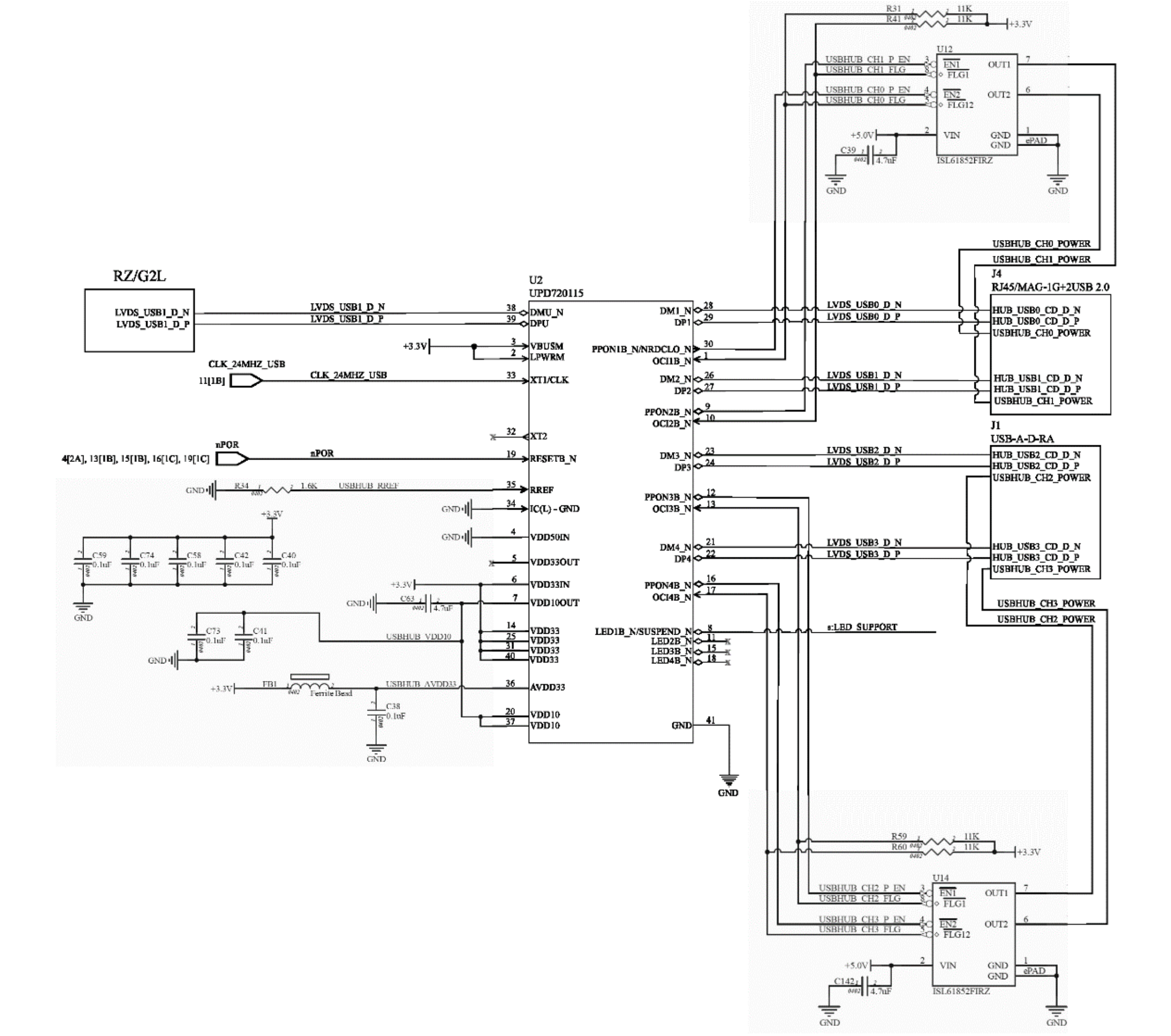


Figure 93. USB 2.0 Hub Block Diagram

14.2.8.3 MIPI CSI Interface

The RZ/G2L-SBC comes with a dual-channel MIPI CSI port labelled as J6. It is located right next to the 3.5 mm audio jack. The CSI port 15-pin camera port is verified to work with the OV5640 camera module. It supports two data channels and one I2C channel. It is directly interfaced to the RZ/G2L SoC.

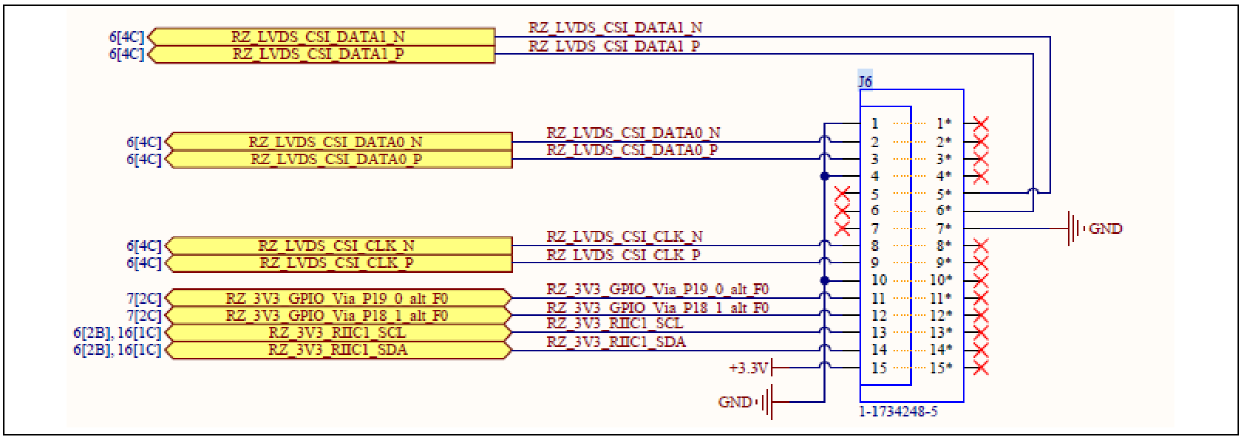


Figure 94. CSI Interface Schematic

14.2.8.4 MIPI DSI Interface

The RZ/G2L-SBC comes with a dual-channel MIPI DSI port labelled as J5. It is located toward the edge of the board next to the Wi-Fi chipset. The 15-pin display port is verified to work with Waveshare 5” DSI display with a capacitive touch interface module. It supports two data channels and one I2C channel. It is directly interfaced to the RZ/G2L SoC.

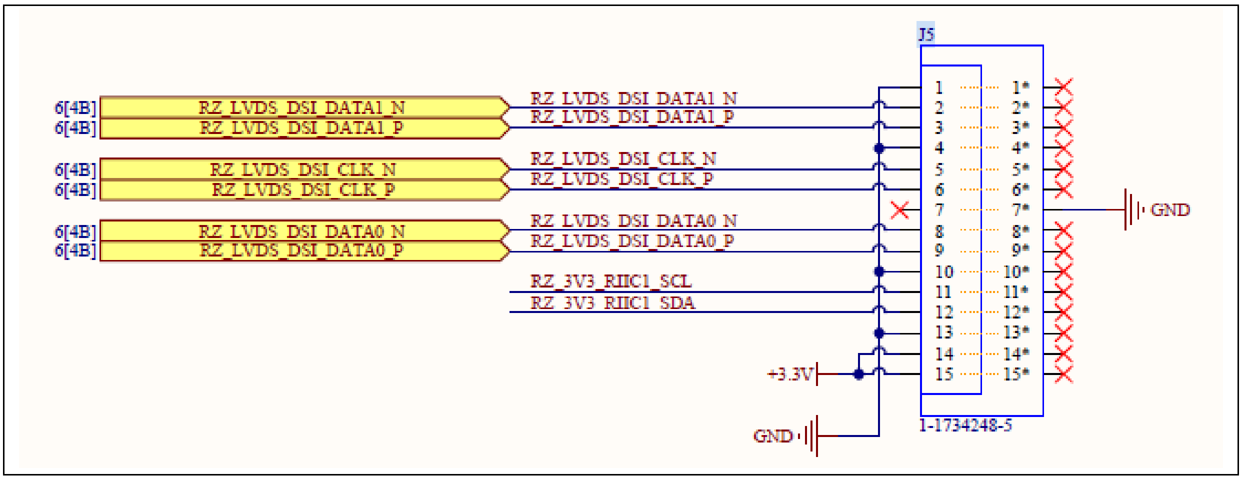


Figure 95. DSI Schematic

14.2.8.5 Audio DAC with 3.5mm Jack

The RZ/G2L-SBC comes with an onboard audio DAC from Renesas: DA7219. The audio DAC is interfaced to the RZ/G2L SoC to its SSI1 and I2C 0. The SSI 1 is used for audio streaming of I2S data, while the I2C interface is used for mux and peripheral control.

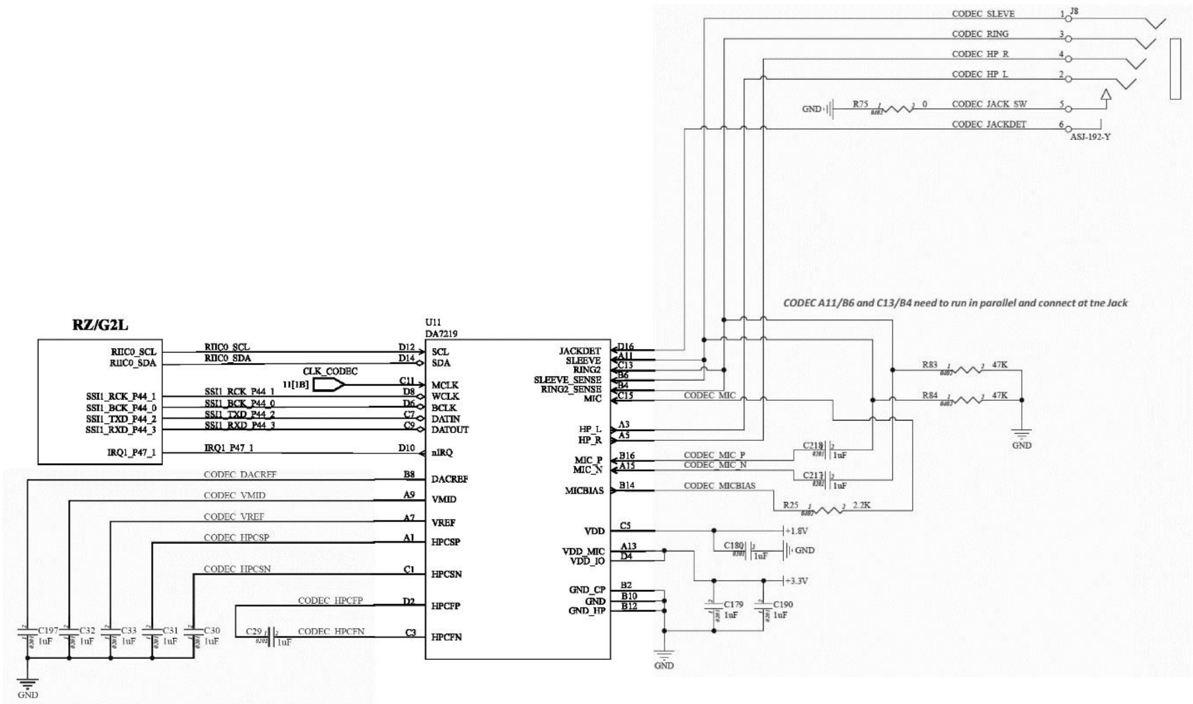


Figure 96. Audio CODEC Interface Block Diagram

14.2.8.6 HDMI Display Subsystem

The RZ/G2L-SBC comes with an HDMI display output, which is derived from the RGB parallel interface from RZ/G2L SoC through an RGB to HDMI converter interface IC. The physical HDMI port is a mini-HDMI type (not micro). The HDMI signal source is the RGB parallel LVDS interface. An RGB to HDMI bridge IC is used to convert RGB to the HDMI protocol. The bridge is fully supported, and the HDMI is enabled with the EDID feature.

Note: The LVDS interface is the source for both the external HDMI bridge and the on-chip DSI IP blocks. So, only one interface may be active at a time. Under no circumstances should both interfaces be turned on at the same time, as there is a limitation regarding the ISP unit.

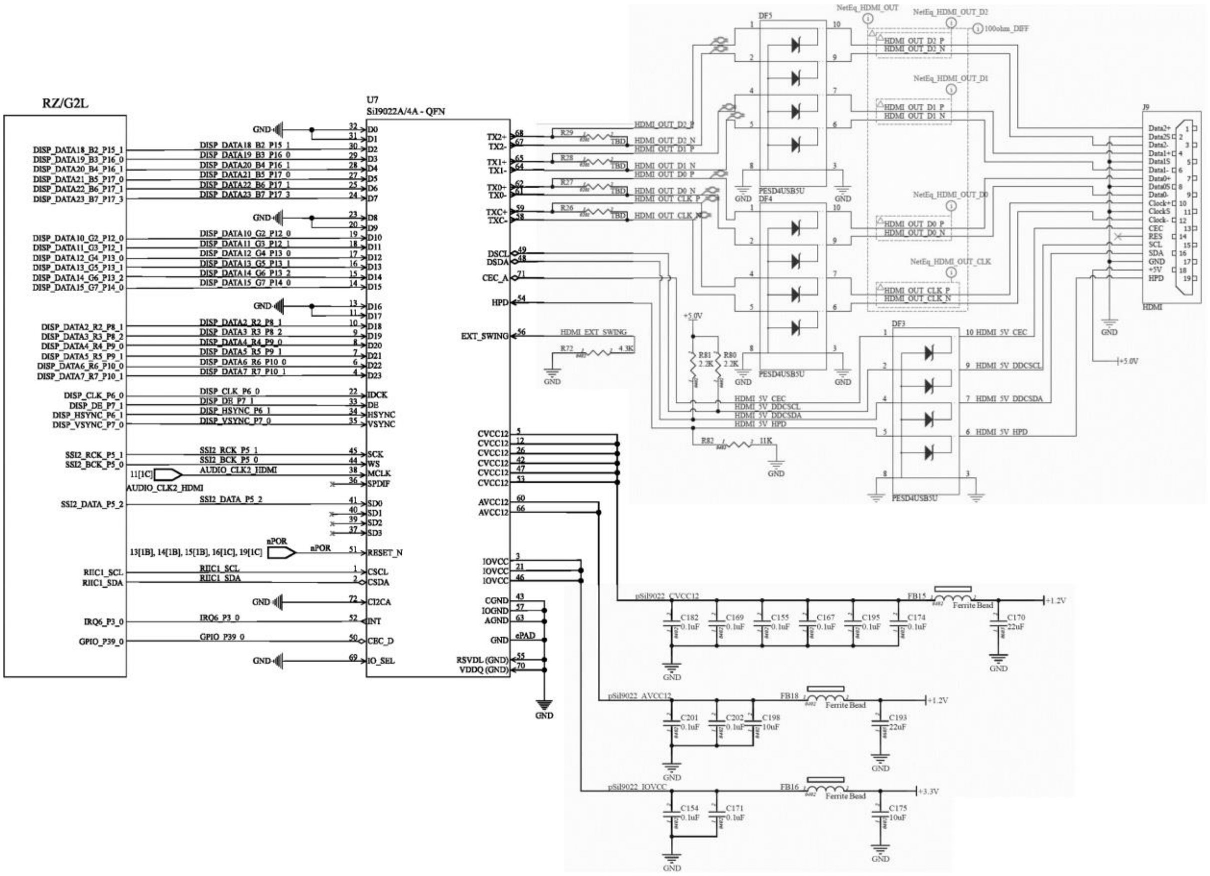


Figure 97. HDMI Bridge and mini-HDMI port interfacing.

14.2.8.7 40-pin I/O Header

The RZ/G2L-SBC comes with a 40-pin GPIO interface, which is broadly compliant with the Raspberry Pi 3 40-pin GPIO interface and provides additional interfaces like two CAN ports. The diagram below shows the pin configuration along with marking of the bottom I/O ports for reference to the orientation of the board

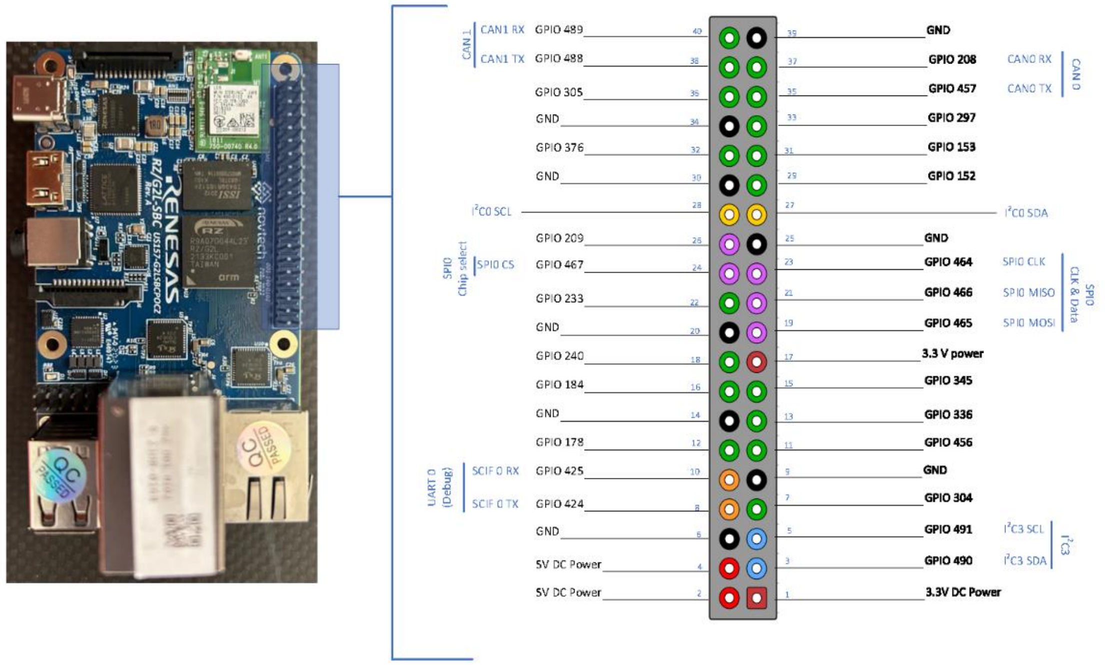


Figure 98. 40 PIN GPIO map with orientation details.

14.2.8.8 PMOD Type 6A Standard Interface

The RZ/G2L-SBC is equipped with a 2x6-pin header routed to the PMOD Type-6A interface conforming to the 1.3.0 specification of PMOD. It includes the alternate pin functions from the specification.

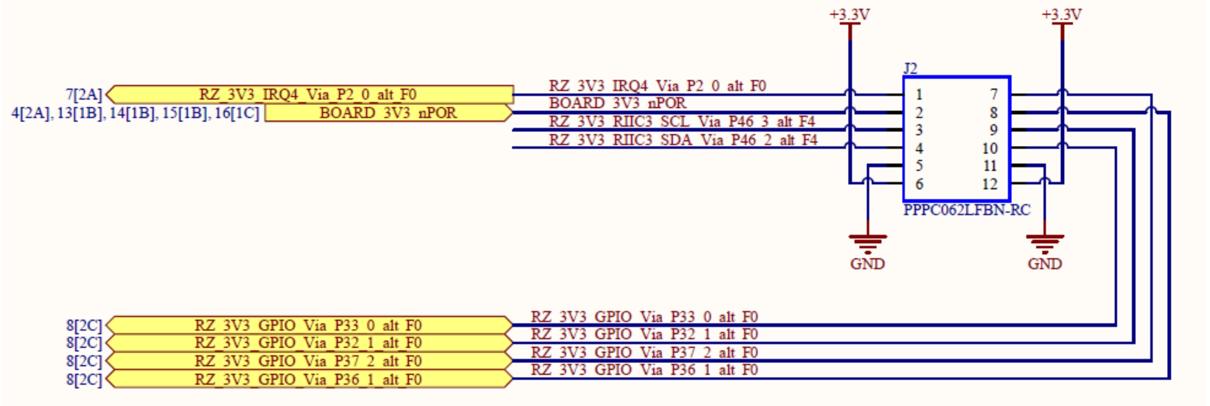

+3.3V
+3.3V
J2
7[2A]
RZ_3V3_IRQ4_Via_P2_0_alt_F0
4[2A], 13[1B], 14[1B], 15[1B], 16[1C]
BOARD_3V3_nPOR
RZ_3V3_IRQ4_Via_P2_0_alt_F0
BOARD_3V3_nPOR
RZ_3V3_RIIC3_SCL_Via_P46_3_alt_F4
RZ_3V3_RIIC3_SDA_Via_P46_2_alt_F4
1 2 3 4 5 6
7 8 9 10 11 12
PPPC062LFBN-RC
GND
GND
8[2C] RZ_3V3_GPIO_Via_P33_0_alt_F0
8[2C] RZ_3V3_GPIO_Via_P32_1_alt_F0
8[2C] RZ_3V3_GPIO_Via_P37_2_alt_F0
8[2C] RZ_3V3_GPIO_Via_P36_1_alt_F0
RZ_3V3_GPIO_Via_P33_0_alt_F0
RZ_3V3_GPIO_Via_P32_1_alt_F0
RZ_3V3_GPIO_Via_P37_2_alt_F0
RZ_3V3_GPIO_Via_P36_1_alt_F0

Figure 99. Schematic of PMOD Type 6 A pin header J2.

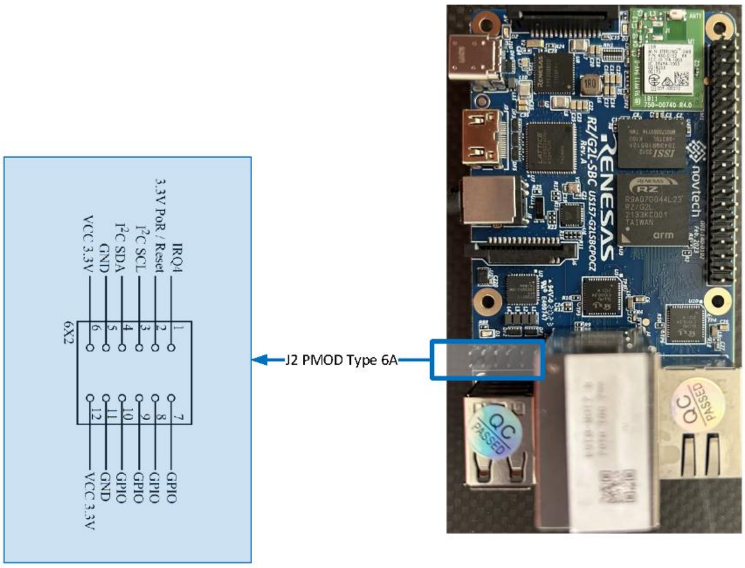


Figure 100. PMOD Type 6A 2x6 0.1mm pin out with orientation details.

14.2.8.9 uSD-Card Interface

The RZ/G2L-SBC comes with a spring-loaded micro-SD card slot. This is intended to be the primary storage as well as the OS boot device. The SD card is connected to channel 0 of the RZ/G2L SoC SD/MMC interface. The SoC SDIO interface is compliant with memory card standard version 3.0 and supports UHS-1 mode of 50 MB/s (SDR50) and 104 MB/s (SDR104).

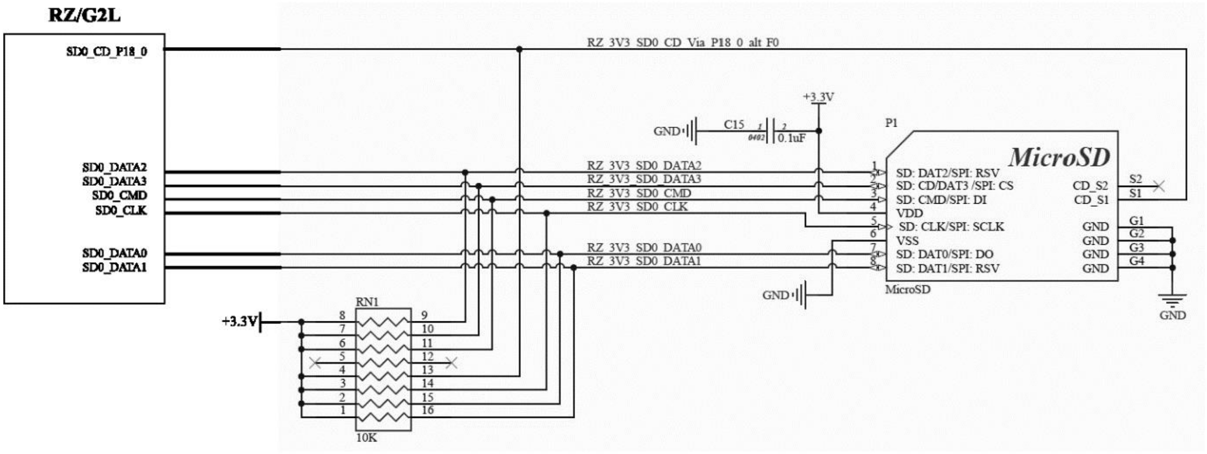


Figure 101. uSD-Card interface block diagram.

14.2.8.10 JTAG SWD Debug

The JTAG/SWD interface is an SMT pin-out on the bottom side of the board marked J11. It uses the standard 10-pin interface when populated. By default, this is not populated on the board. In addition to populating the pins of J11, the use of the J12 port to set BSCANP is necessary to trigger the JTAG boundary scan of the RZ/G2L SoC. The SBC by itself will not be able to initiate the JTAG boundary scan mode. All the interface lines have pull-ups.

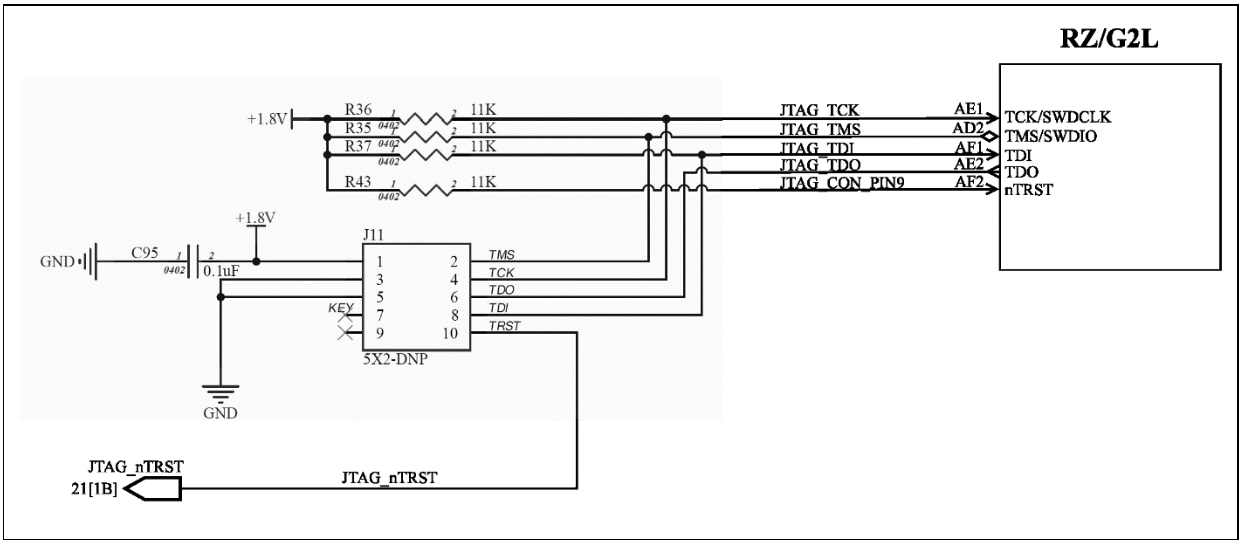


Figure 102. JTAG/SWD Block Diagram

14.2.8.11 Expansion Connector

The RZ/G2L-SBC has two connectors in the bottom, J12 and J13, that contain pinouts for the ADC inputs, Bootstrapping (boot mode selection), and the QSPI1 interface, in addition to a few GPIOs. This is meant to be used in conjunction with an adapter/daughter board. The primary uses of this are mostly custom versions, where factory flashing and other manufacturing functions are controlled by these lines. The ADC input lines are also mapped to the J13 connector.

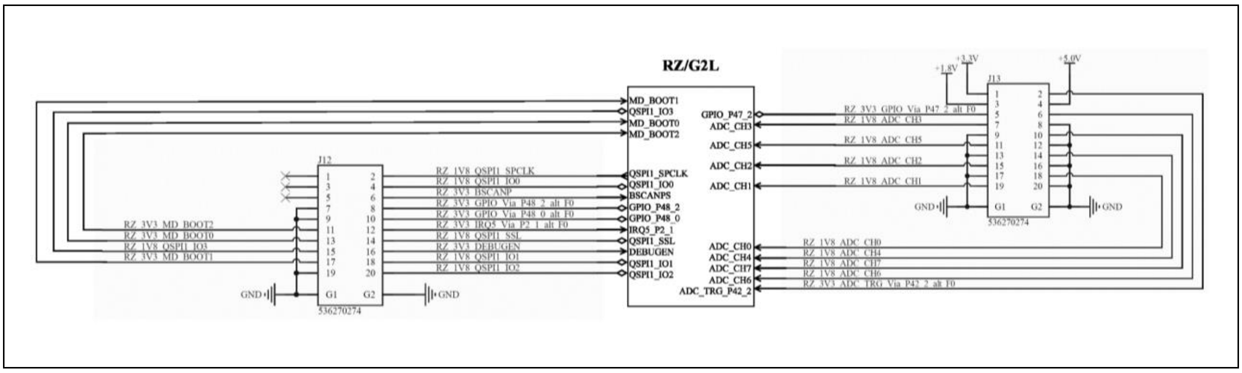


Figure 103. Block diagram of Bottom Connectors.

Refer to the appendix for details on the adapter board and flashing tools.

14.2.9 Memory

The RZ/G2L-SBC design uses four types of memory.

1. QSPI NOR Flash
2. DDR4 SDRAM
3. EEPROM
4. SD-Card

14.2.9.1 QSPI Flash

The QSPI flash memory is controlled by the SPI multi-I/O bus controller (SPIBSC) that is built into the RZ/G2L. This memory supports both single data rate (SDR) and double data rate (DDR) transfers at 66MHz and 50MHz clock frequency. QSPI0 interfaces to a Cyprus S25FS512SDSNFB010 64MiB NOR

Flash module. The QSPI is the default boot device, which contains the firmware: Arm Trusted Firmware (ATF), OPTEE (loaded but disabled by default), and U-Boot.

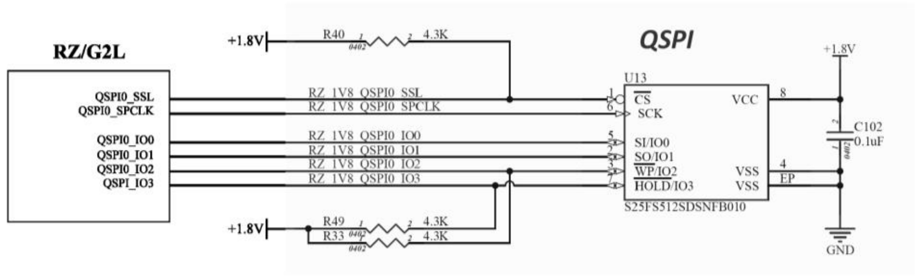


Figure 104. QSPI interface.

Note: The pull-up resistor on the clock line “QSPI0_SPCLK” is optional and is omitted in this design.

14.2.9.2 DDR4 SDRAM

The DDR4 SDRAM is controlled by the DDD3L/DDR4 SDRAM Memory Controller (MEMC) that is built into the RZ/G2L. This interface supports up to DDR4-1600 SDRAM, a data bus width of 16-bit, and inline ECC.

This interface complies with JEDEC STANDARD JESD79-4C.

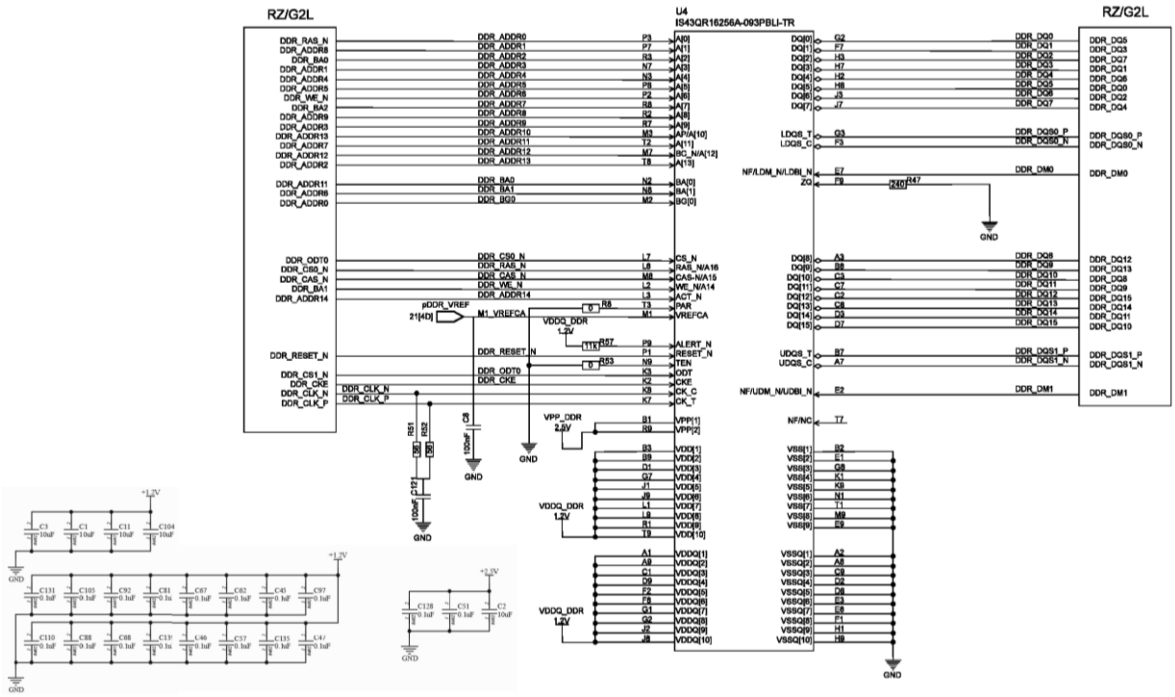
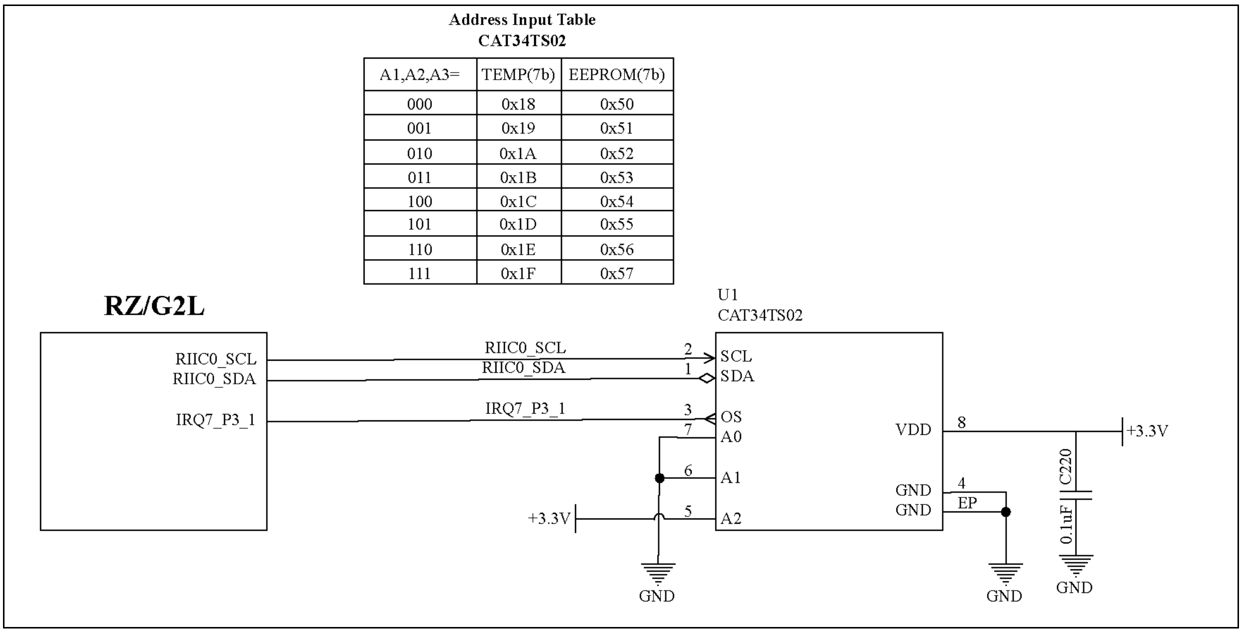


Figure 105. DDR4 SDRAM Interface

14.2.9.3 EEPROM with Temperature Sensor.

The RZ/G2L-SBC has an onboard CAT34TS02 I^2C Temperature sensor with on-chip EEPROM, which is meant to hold factory data like Serial number, manufacturer name, etc. It is currently only used to hold the Ethernet MAC ID's. Each board has its own registered MAC ID, which is stored on the EEPROM and read by u-boot during bootup. The EEPROM also has a built-in temperature sensor that can be read over the I^{2}C interface. The EEPROM is configured as 16 pages of 16 bytes each for a total of 256 KiB (2 kilobits) of memory. Currently, two MAC IDs occupy 6 bytes of memory each for a total of 12 bytes.



Address Input Table
CAT34TS02

A1,A2,A3=	TEMP(7b)	EEPROM(7b)
000	0x18	0x50
001	0x19	0x51
010	0x1A	0x52
011	0x1B	0x53
100	0x1C	0x54
101	0x1D	0x55
110	0x1E	0x56
111	0x1F	0x57

Figure 106. I2C EEPROM Block Diagram

Table 20. EEPROM Parameters

Parameter	Value	Description
I^{2}C speed	100KHz / 400 KHz	It supports the standard and fast modes of operation.
EEPROM memory size	2 kib / 256 bytes	
EEPROM memory ordering	16 pages of 16 bytes each	Page bank array configuration
Temperature range	-20 °C to +125 °C	
Operating Voltage	3.3V	
Temperature alarm	Programmable over I^{2}C	Three programmable trigger settings for high, low, and critical temperatures to raise interrupt over line IRQ 7.

14.2.10 GPIO Internals

The RZ/G2L SoC has a unique way of GPIO organization. It is not the typical banked GPIO interface that one might be used to. The RZ/G2L has individual GPIO LSI logic directly attached to the register outputs. This creates a notation for GPIO pins attached to ports, which are basically bits in a register.

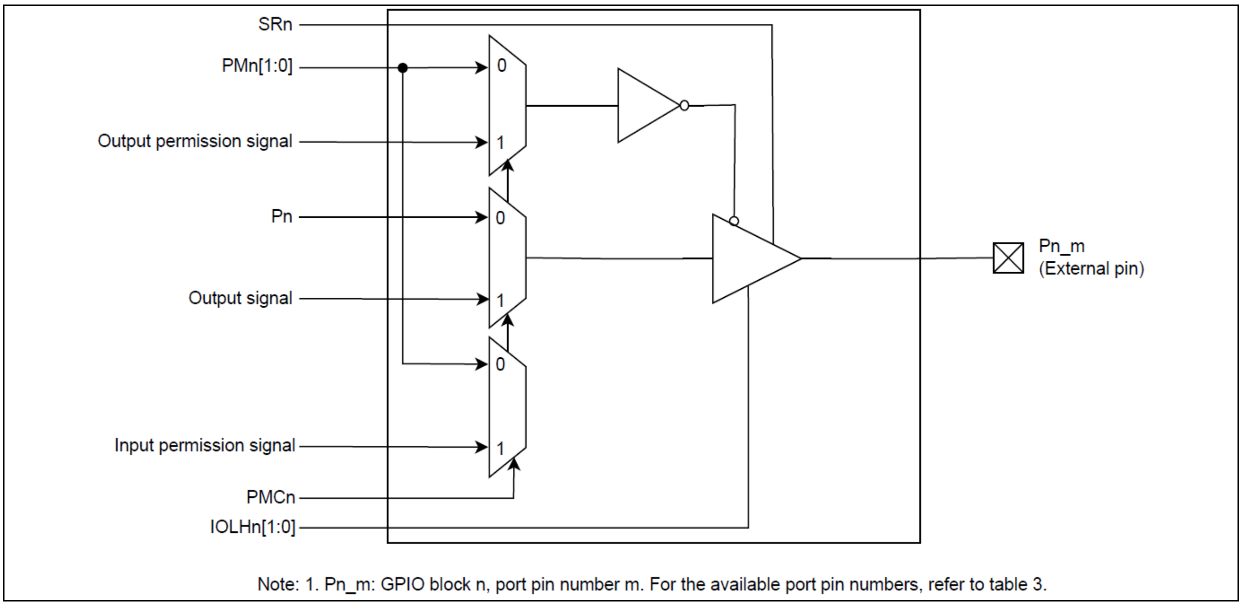
Px_y :

P= port a.k.a 8 bit register set number.

x= port number

y= port bit

Each bit in a port control register corresponds to a single gpio logic module. While each port has 8 bits, most of the ports (registers) are only using the lower two to three bits for gpio line outs. The upper bits are used for other special functions at times. Table 16. GPIO-supported pins in RZ/G2L maps all the available ports to bits.



Note: 1. Pn_m: GPIO block n, port pin number m. For the available port pin numbers, refer to table 3.

Figure 107. Multiplexed peripheral functions configuration diagram for GPIO pins

RZ/G2L can support up to 123 general-purpose I/O pins from 49 ports in Table 16. GPIO-supported pins in RZ/G2L:

Table 21. GPIO-supported pins in RZ/G2L

Port name	External Terminal Name					
	Bit7-5	Bit4	Bit3	Bit2	Bit1	Bit0
PORT 10	-	-	-	-	P0_1	P0_0
PORT 11	-	-	-	-	P1_1	P1_0
PORT 12	-	-	-	-	P2_1	P2_0
PORT 13	-	-	-	-	P3_1	P3_0
PORT 14	-	-	-	-	P4_1	P4_0
PORT 15	-	-	-	P5_2	P5_1	P5_0
PORT 16	-	-	-	-	P6_1	P6_0
PORT 17	-	-	-	P7_2	P7_1	P7_0
PORT 18	-	-	-	P8_2	P8_1	P8_0
PORT 19	-	-	-	-	P9_1	P9_0
PORT 1A	-	-	-	-	P10_1	P10_0
PORT 1B	-	-	-	-	P11_1	P11_0
PORT 1C	-	-	-	-	P12_1	P12_0
PORT 1D	-	-	-	P13_2	P13_1	P13_0
PORT 1E	-	-	-	-	P14_1	P14_0
PORT 1F	-	-	-	-	P15_1	P15_0
PORT 20	-	-	-	-	P16_1	P16_0
PORT 21	-	-	-	P17_2	P17_1	P17_0
PORT 22	-	-	-	-	P18_1	P18_0
PORT 23	-	-	-	-	P19_1	P19_0
PORT 24	-	-	-	P20_2	P20_1	P20_0
PORT 25	-	-	-	-	P21_1	P21_0
PORT 26	-	-	-	-	P22_1	P22_0
PORT 27	-	-	-	-	P23_1	P23_0
PORT 28	-	-	-	-	P24_1	P24_0
PORT 29	-	-	-	-	P25_1	P25_0
PORT 2A	-	-	-	-	P26_1	P26_0
PORT 2B	-	-	-	-	P27_1	P27_0

PORT 2C	-	-	-	-	P28_1	P28_0
PORT 2D	-	-	-	-	P29_1	P29_0
PORT 2E	-	-	-	-	P30_1	P30_0
PORT 2F	-	-	-	-	P31_1	P31_0
PORT 30	-	-	-	-	P32_1	P32_0
PORT 31	-	-	-	-	P33_1	P33_0
PORT 32	-	-	-	-	P34_1	P34_0
PORT 33	-	-	-	-	P35_1	P35_0
PORT 34	-	-	-	-	P36_1	P36_0
PORT 35	-	-	-	P37_2	P37_1	P37_0
PORT 36	-	-	-	-	P38_1	P38_0
PORT 37	-	-	-	P39_2	P39_1	P39_0
PORT 38	-	-	-	P40_2	P40_1	P40_0
PORT 39	-	-	-	-	P41_1	P41_0
PORT 3A	-	P42_4	P42_3	P42_2	P42_1	P42_0
PORT 3B	-	-	P43_3	P43_2	P43_1	P43_0
PORT 3C	-	-	P44_3	P44_2	P44_1	P44_0
PORT 3D	-	-	P45_3	P45_2	P45_1	P45_0
PORT 3E	-	-	P46_3	P46_2	P46_1	P46_0
PORT 3F	-	-	P47_3	P47_2	P47_1	P47_0
PORT 40	-	P48_4	P48_3	P48_2	P48_1	P48_0

-: unused pins

Note: The RZ/G2L has only one GPIO chip interface to control all the supported pins mentioned in Section 8.1.2.1.(2). GPIO (General Purpose I/O pins) with libgpiod

14.3 RZ/G2L-EVK & RZ/V2L-EVK Board

The RZ/G2L-EVK and RZ/V2L-EVK share an identical hardware design. Both evaluation kits use the same PCB layout, connectors, and peripheral configuration, differing only in the SoC mounted on the board. Unless otherwise specified, the descriptions in this section apply equally to both EVKs.

14.3.1 Overview

The RZ/G2L SMARC Module Board provides a compact evaluation platform for the Renesas RZ/G2L microprocessor (R9A07G044L23GBG). Its design follows the SMARC v2.1 standard and is intended for functional evaluation, performance testing, and application software development.

The companion RZ/V2L SMARC Module Board (RTK9754L23C01000BE) is identical in hardware layout but mounts the R9A07G054L23GBG device. Since both processors are pin-compatible, the two boards are functionally equivalent.

Processor

- RZ/G2L microprocessor (R9A07G044L23GBG)
- Arm® Cortex®-A55 × 2 cores, Arm® Cortex®-M33 × 1 core
- Multimedia and peripheral subsystems integrated

Memory and Storage

- DDR4 SDRAM: 2 GB × 1 device
- QSPI NOR Flash: 512 Mb × 1 device
- eMMC: 64 GB × 1 device
- microSD card slot: supported, usable for booting as eSD

Clock and Power

- Clock Generator: 5-output programmable clock generator (5P35023)
- PMIC: Renesas RAA215300, supplies all SoC and board rails

Connectivity

- Ethernet PHY: integrated, supporting 10/100/1000 Mbps
- SMARC Edge Connector: 314-pin, 0.5 mm pitch; exposes unused SoC signals for carrier board expansion
- The Micro-HDMI connector via DSI/HDMI conversion module is mounted as standard for connection to high-speed serial interface for digital video module.
- The audio codec is mounted as standard for advanced development of audio system. A 3.5 mm audio jack is provided for connection to audio equipment.
- MIPI-CSI interface: supports external camera modules (e.g., Google Coral Camera). Image recognition processing can be used with images input with MIPI camera.

Debug and Expansion

- Debugging: 10-pin header for ARM Cortex debug interface
- Analog: 10-pin header for ADC signals
- Carrier board support: SMARC connector enables use with standard or custom carrier boards

14.3.2 Layout Parts

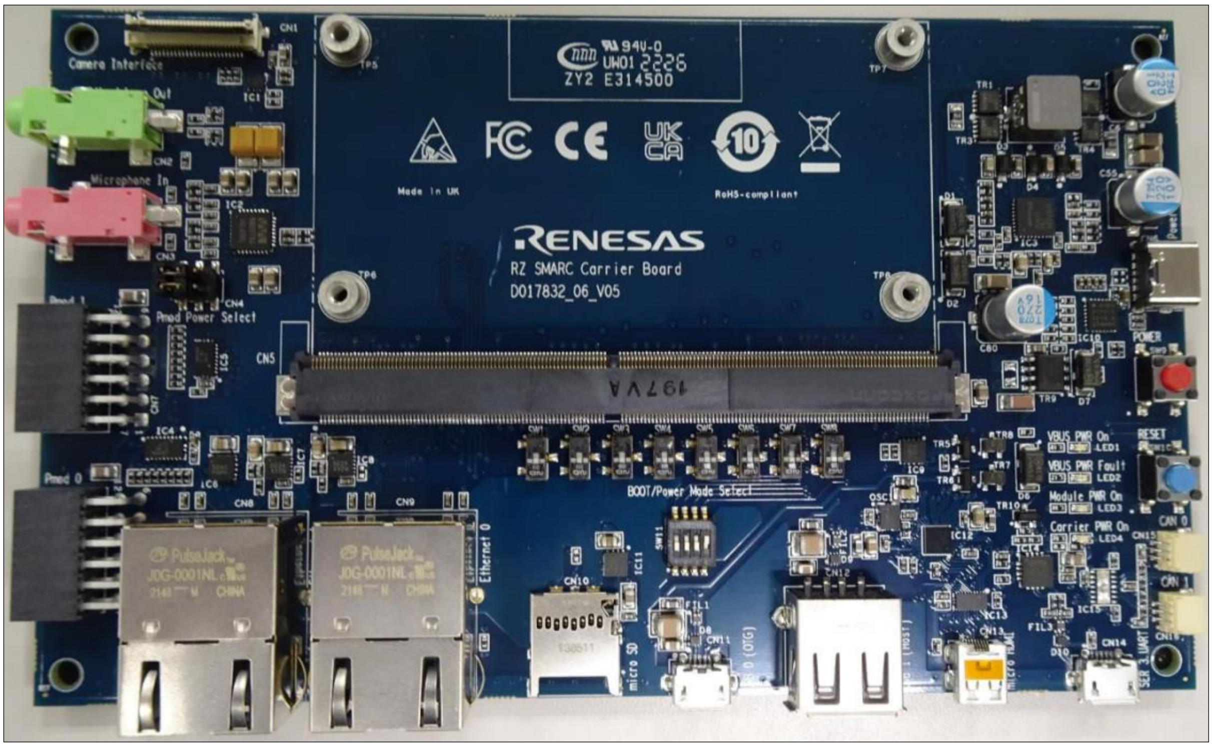


Figure 108. Top View of the RTK97X4XXXB00000BE

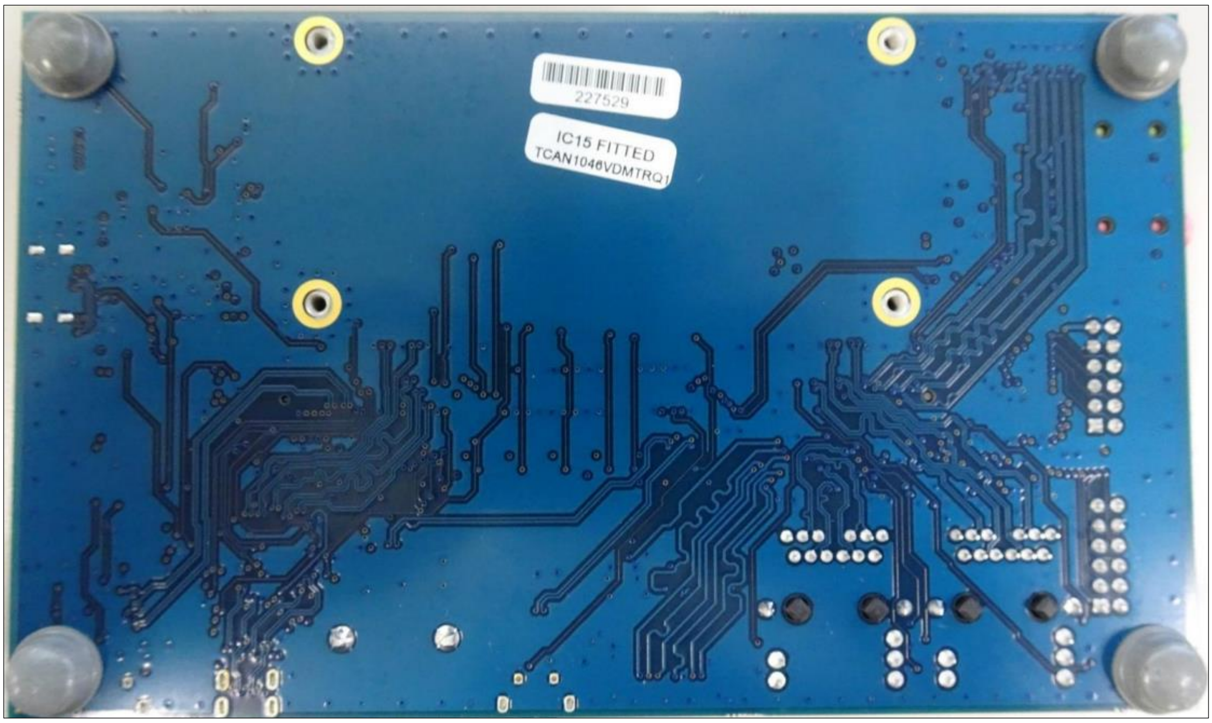


Figure 109. Bottom View of the RTK97X4XXXB00000BE

14.3.3 Overview of Connectors

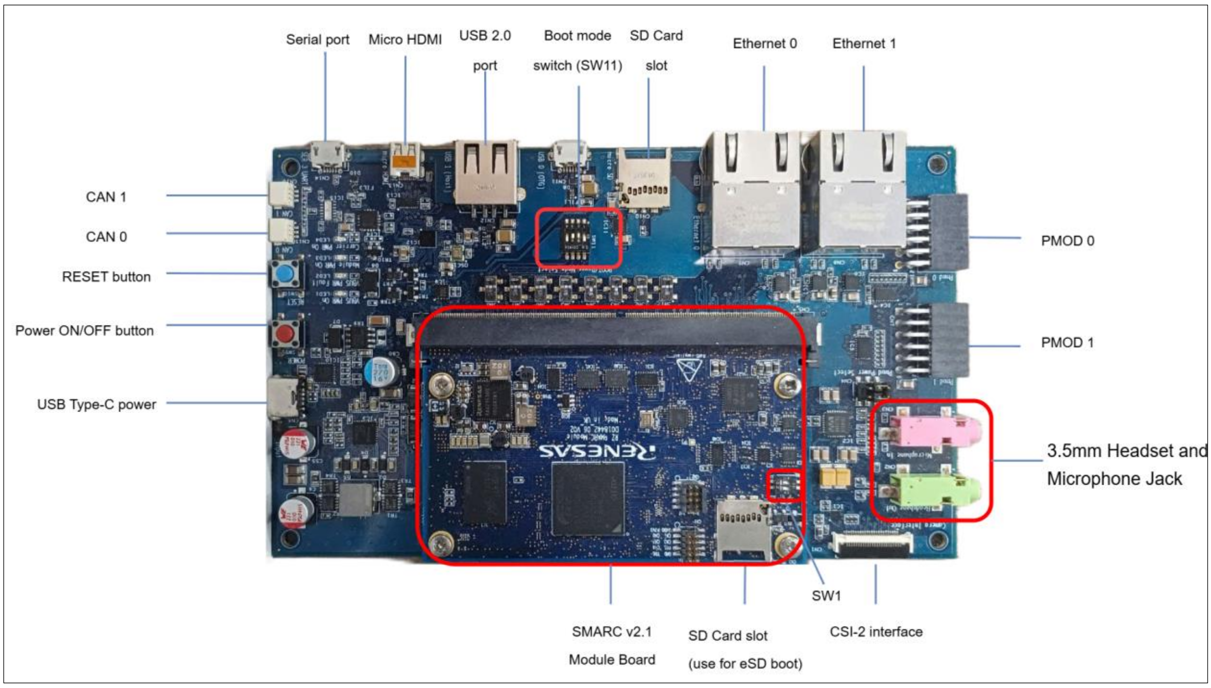


Figure 110. RZ/G2L-EVK Overview of Connectors

14.3.4 Power Supply

14.3.4.1 USB Type-C Power

The board includes a USB Type-C receptacle for power input with USB Power Delivery support. The connector is designed for connection to a 5 V power source. A minimum current supply of 3 A is required to ensure stable operation and avoid brownouts. For systems using multiple peripherals, a power source rated at 4.5–5 A is recommended to provide sufficient margin.

14.3.4.2 Power Rails

This section describes how the RZ/G2L-EVK and RZ/V2L-EVK boards distribute power from the USB Type-C input into the different voltage rails required by the SoC and peripheral devices. Each functional block of the SoC is supplied by a dedicated regulator output to ensure stable operation.

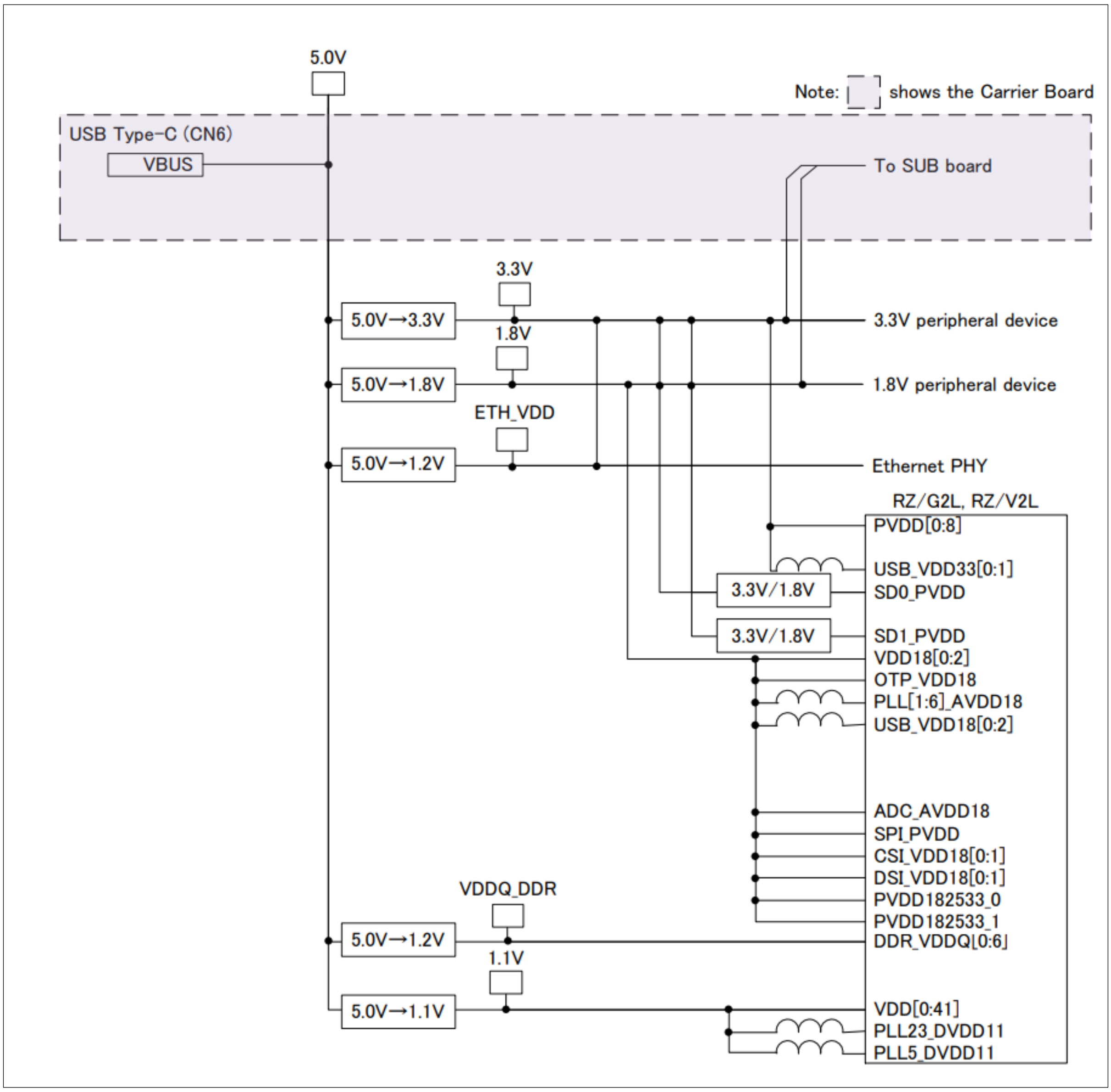


Figure 111. RZG2L-EVK Block Diagram of Power rails

Input source

- The board is powered through a USB Type-C connector (CN6).
- VBUS 5 V from USB Power Delivery serves as the main input supply.

Regulated rails

From the 5 V input, multiple voltage rails are generated by on-board DC-DC converters and LDOs:

- 3.3V rail:
  - Supplies 3.3 V peripheral devices and SoC I/O domains.
  - Also, powers USB_VDD33, SD0_PVDD, and other 3.3 V blocks.
- 1.8 V rail:
  - Supplies low-voltage peripherals and interface logic.
  - Provides power to SD1_PVDD, OTP_VDD18, USB_VDD18, and Ethernet PHY.
- 1.2 V rail
  - Used by SoC internal PVDD domains and some PLL circuits.
  - Also feeds DDR VDDQ and selected core logic.
- 1.1 V rail
  - Supplies SoC digital cores (VDD11, DVDD11).
  - Used for PLL and analog domains requiring 1.1 V.

14.3.4.3 Power Supply Regulation

This section describes the on-board power management and regulation for the RZ/G2L-EVK and RZ/V2L-EVK. The board uses a dedicated Power Management IC (PMIC) to generate and sequence the supply rails required by the SoC and peripherals.

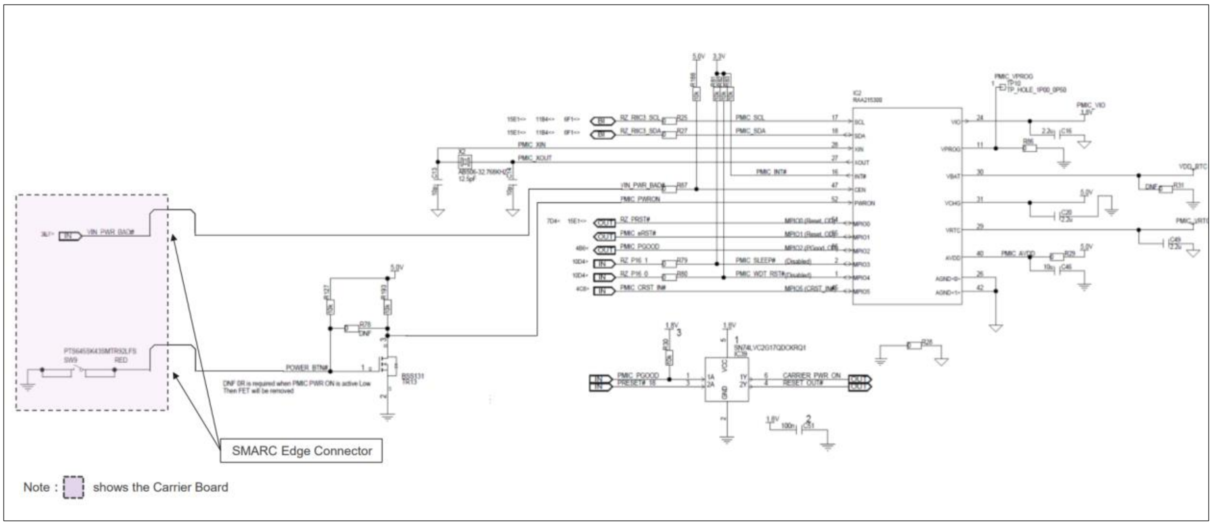


Figure 112. RZG2L-EVK Block Diagram of Power Supply Regulation

14.3.5 Power Management Integrated Circuit (PMIC)

LDO1 and LDO2 provide low-current regulated outputs (up to 0.3 A each) used to supply SoC subsystems or peripherals that require stable voltage rails. Their output voltages are not fixed in hardware but are determined by the state of configuration pins on the SoC.

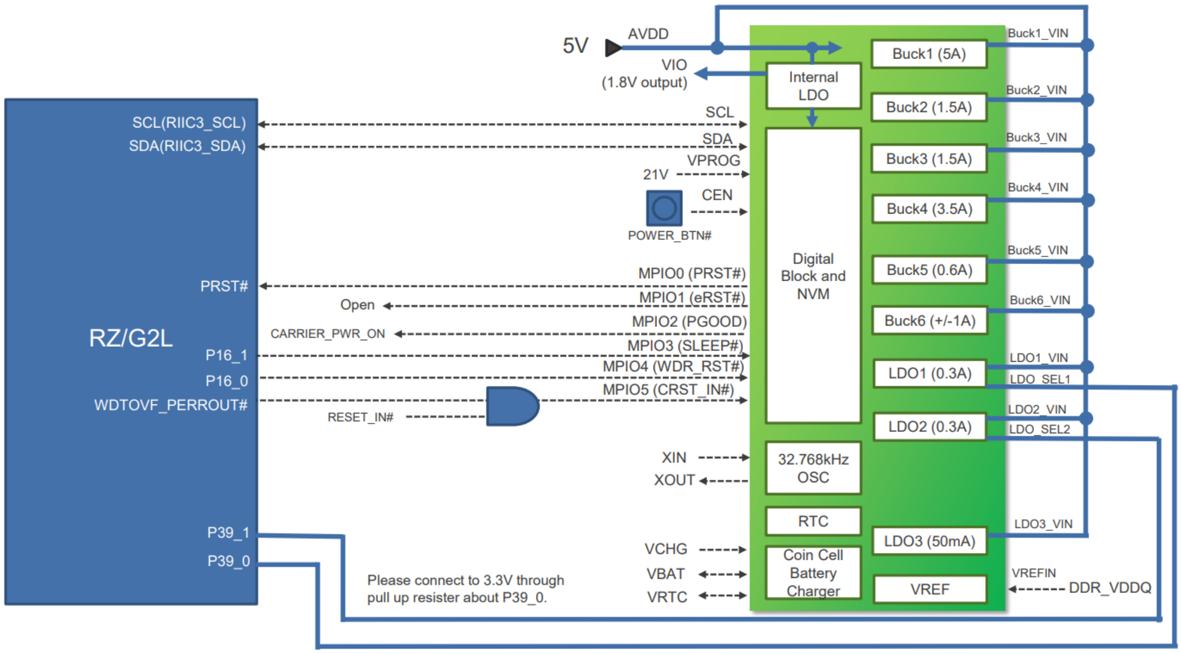


Figure 113. RZG2L-EVK Block diagram of PMIC

14.3.6 RESET Control

This section explains how reset signals are managed on the RZ/G2L-EVK and RZ/V2L-EVK boards. Reset control ensures that the SoC, memory, and peripheral devices start in a defined state during power-up or after a manual reset.

Reset sources.

- Power-on reset — generated automatically by the PMIC (RTK9744L23C01000BE / RAA215300) when the 5 V input supply is applied and stable.
- Manual reset — initiated by pressing the reset button (RESET_BTN#), which asserts RESET_IN#.

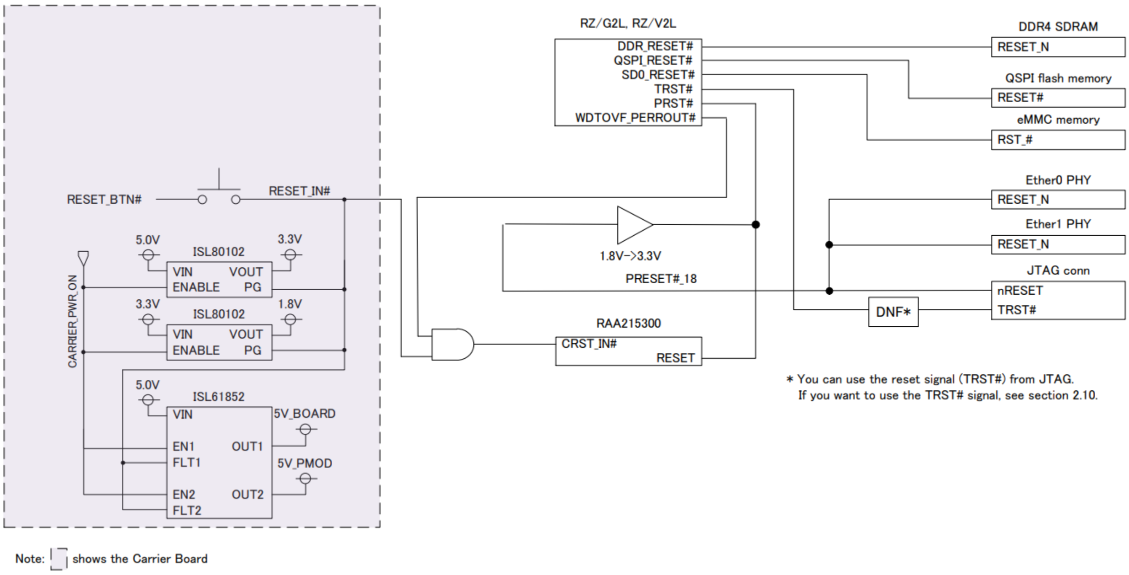


Figure 114. RZG2L-EVK Block Diagram of RESET Control

14.3.7 Clock Configuration

The RZ/G2L-EVK provides a flexible clock distribution network to support the SoC, DDR, storage, and peripheral interfaces. A 24 MHz reference oscillator feeds a clock generator IC, which distributes clocks to the SoC, Ethernet PHYs, audio codec, and high-speed interfaces.

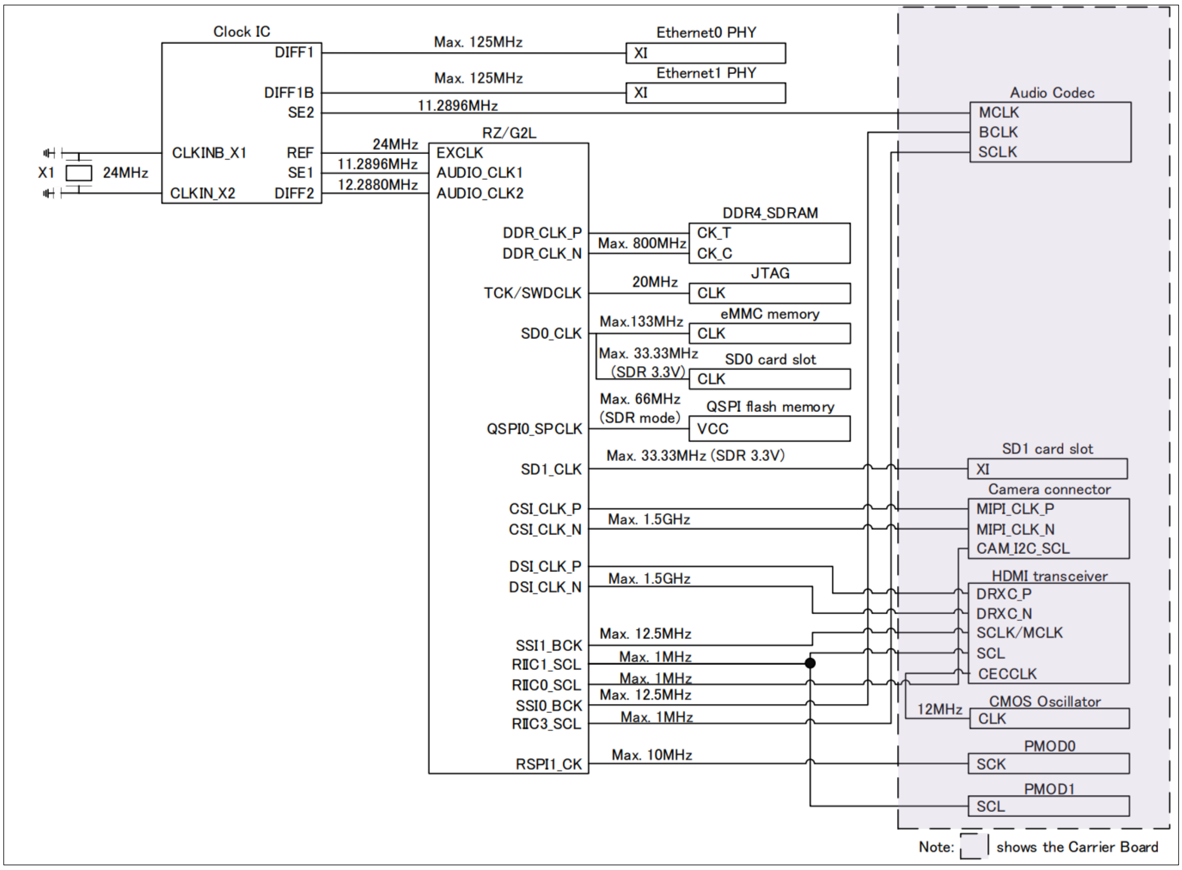


Figure 115. RZG2L-EVK Block Diagram of Clock Configuration

14.3.8 Peripheral Interface

14.3.8.1 Gigabit Ethernet

The RZ/G2L-EVK integrates a Gigabit Ethernet interface to provide high-speed network connectivity. The SoC contains a built-in Ethernet Media Access Controller (E-MAC) that implements the MAC layer functions. This controller connects externally to a dedicated Ethernet PHY, which handles the physical layer signaling.

The Ethernet subsystem requires a reference clock, which is supplied from an external clock generator to the PHY. This ensures stable timing for both transmission and reception of data.

Standards compliance.

- The interface follows the IEEE 802.3 specification for Ethernet operation.
- The link between the SoC and the PHY uses the Reduced Gigabit Media-Independent Interface (RGMII) standard.
- Supported data rates include 10 Mbps, 100 Mbps, and 1000 Mbps.

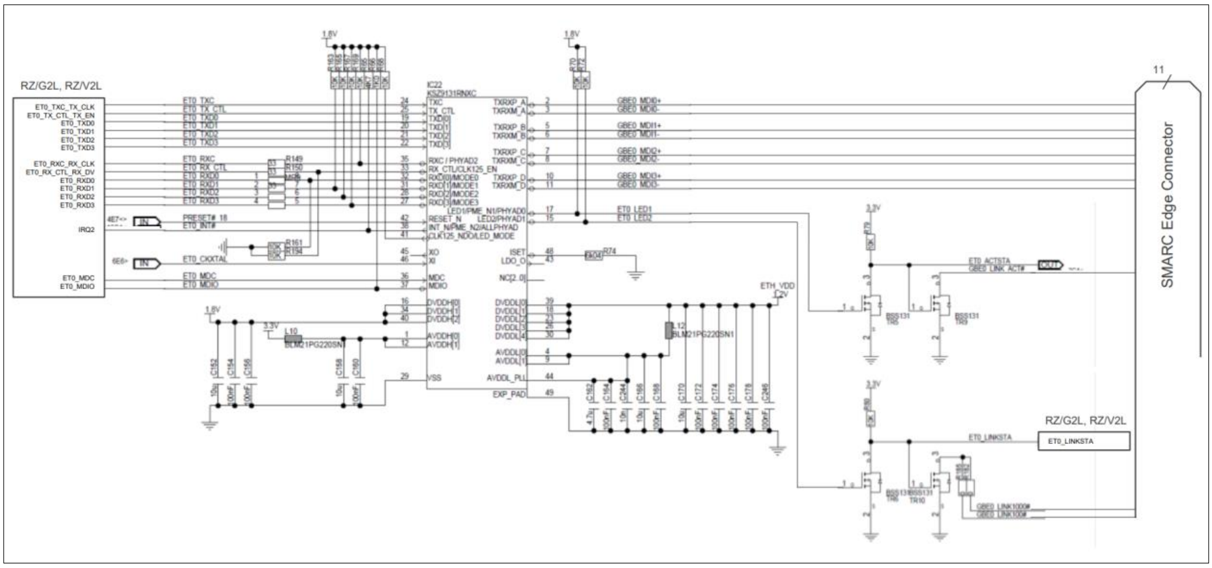


Figure 116. RZG2L-EVK Block Diagram of Ethernet

14.3.8.2 USB 2.0 Ports

The RZ/G2L-EVK provides two USB 2.0 interfaces:

- USB0 (OTG Port)
  - Configurable as either Host or Device (On-The-Go mode).
  - Supports VBUS detection and ID pin control for role switching.
  - Exposed through a Micro-USB connector for direct peripheral or host connection.
- USB1 (Host Port)
  - Dedicated Host-only port, intended for USB peripherals such as keyboards, storage devices, or Wi-Fi dongles.
  - Exposed through a standard USB Type-A connector.

Both ports comply with the USB 2.0 specification, supporting high-speed (480 Mbps) and full-speed (12 Mbps) operation. Protection circuitry is implemented with ESD diodes and EMI filters to ensure stable operation and compliance with EMI/ESD requirements.

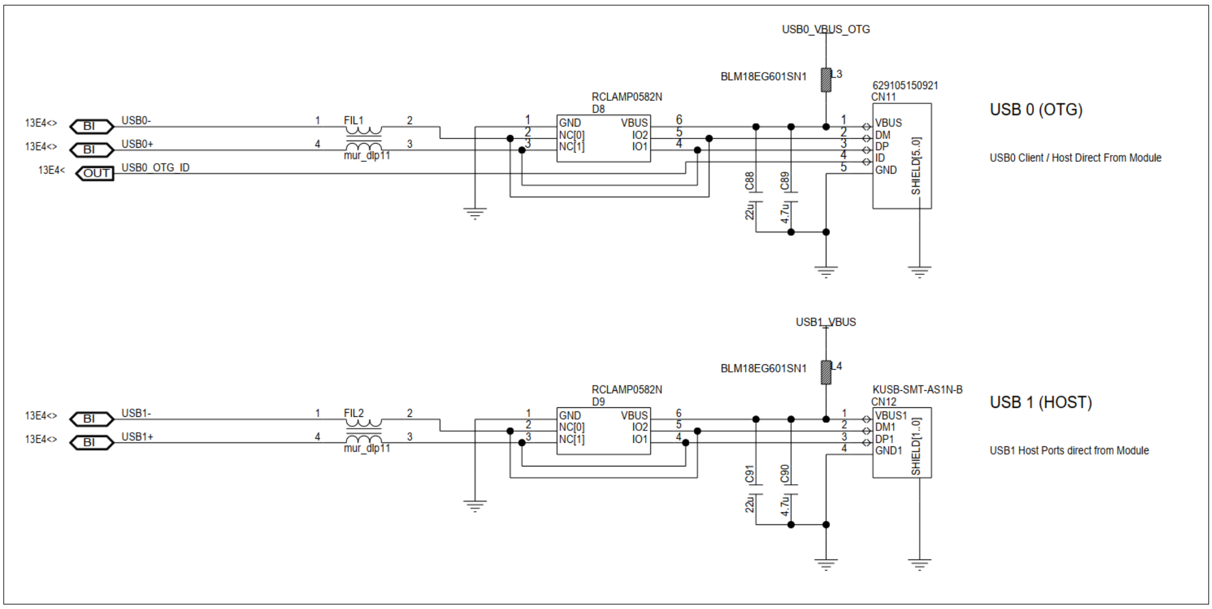


Figure 117. RZG2L-EVK Block Diagram of USB 2.0 Ports

14.3.8.3 MIPI CSI Interface

The RZ/G2L-EVK includes a MIPI CSI-2 camera interface for attaching external image sensors. The interface provides high-speed serial lanes optimized for video capture and vision processing tasks. The interface is designed to work with standard MIPI CSI-2 camera modules. For example, the Google Coral Camera Module can be connected directly, enabling rapid prototyping of computer vision and AI workloads.

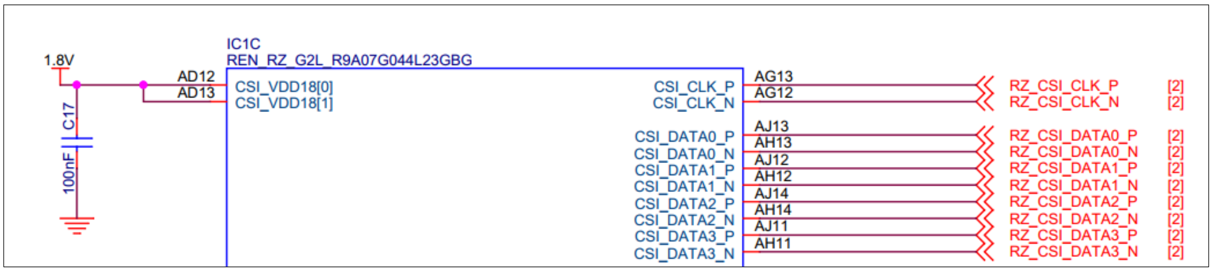


Figure 118. RZG2L-EVK Schematic MIPI CSI

14.3.8.4 Audio Subsytem with 3.5mm Jack

The RZ/G2L-EVK integrates an audio subsystem based on the WM8978 audio codec, providing both input and output capabilities.

- Microphone Input (MIC):
  - A dedicated microphone interface is available through a 3.5 mm jack, with biasing and filtering circuitry for stable audio capture.
- Headphone Output:
  - The codec drives a stereo headphone output via a 3.5 mm jack, supporting audio playback for development and testing.
- Codec Integration:
  - The audio codec is powered by 1.8 V and 3.3 V rails and interfaces with the SoC over the I²S bus. Control is performed via an I²C interface, allowing configuration of audio paths and gain settings.

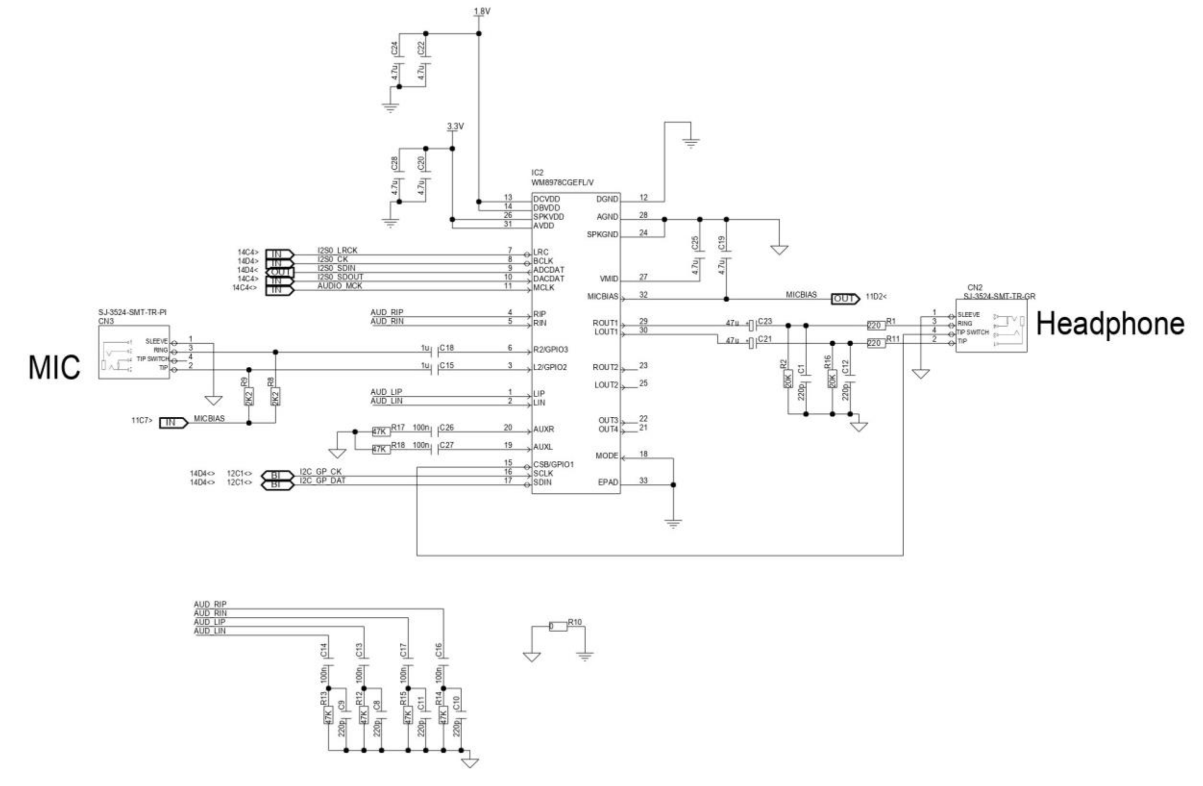


Figure 119. RZG2L-EVK Audio Subsystem

14.3.8.5 HDMI Display Subsystem

The RZ/G2L-EVK includes a micro-HDMI Type-D connector that provides display output through a dedicated HDMI transmitter (ADV7533). The interface uses the SoC’s MIPI-DSI signals, which are converted to HDMI TMDS by the transmitter.

The HDMI block is supported by a 27 MHz reference clock, I²C control lines for configuration, and hot-plug detection. Power is supplied through multiple rails (1.8 V, 3.3 V, and 5 V), with appropriate filtering and decoupling to ensure signal integrity.

This implementation complies with HDMI 1.4 specifications, enabling resolutions up to 1080p at 60 Hz and supporting features such as audio over HDMI and Consumer Electronics Control (CEC).

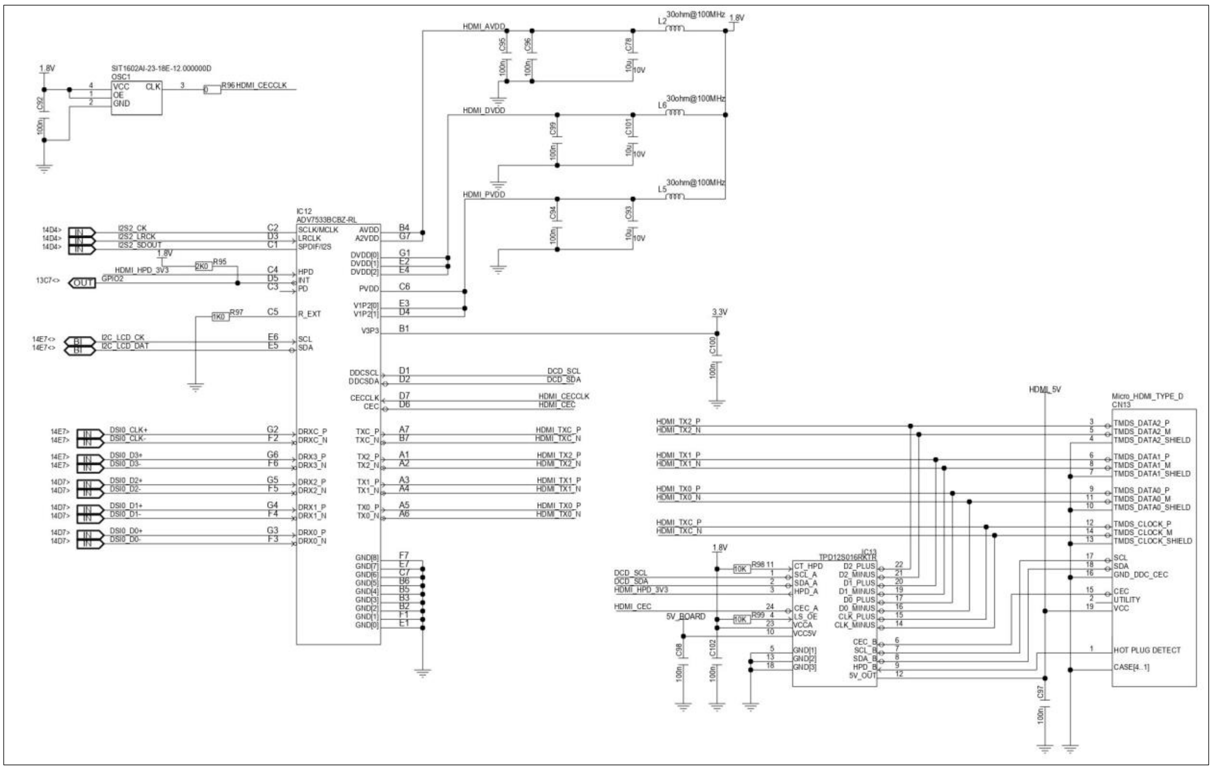


Figure 120. RZG2L-EVK HDMI and Micro HDMI interface

14.3.8.6 uSD-card Interface

The evaluation board includes a microSD slot connected to channel 0 of the SoC's SD/MMC interface. This interface allows the use of a microSD card alongside the onboard eMMC device.

For boot mode switches, please refer to the Section 16.2 Boot Mode Reference (Non-SCIF) RZG2L-EVK & RZV2L-EVK for more detailed information.

Standards compliance.

- Conforms to the SD Memory Card Standard v3.0.
- Supports UHS-I transfer modes, including:
  - SDR50: up to 50 MB/s
  - SDR104: up to 104 MB/s

The microSD interface is typically used for removable storage, firmware updates, or system boot in development setups where eMMC is not required.

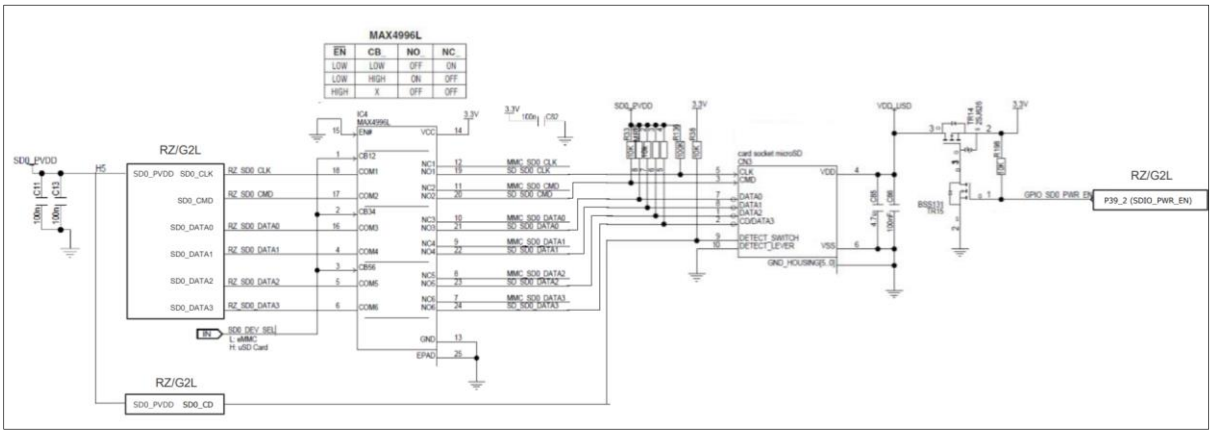


Figure 121. RZG2L-EVK Block Diagram of uSD-card

14.3.8.7 PMOD Interface

The RZ/G2L-EVK provides two PMOD connectors (PMOD0 and PMOD1) to allow flexible peripheral expansion. These connectors follow the Digilent PMOD standard and support different functional configurations such as SPI, UART, and I²C, enabling engineers to attach a wide variety of external modules.

PMOD0 – Type 2A (Expanded SPI)

- Provides an SPI interface with CS, MOSI, MISO, and SCK signals.
- Includes additional GPIO pins for interrupts, reset, and chip select extensions.
- Suitable for SPI-based expansion modules such as sensors, displays, or communication ICs.

PMOD1 – Type 3A / Type 6A (Expanded UART/I²C)

- Configurable as Type 3A (UART) with TXD, RXD, CTS, RTS signals plus general-purpose GPIO.
- Alternatively supports Type 6A (I²C) with SCL, SDA, interrupt, reset, and extra GPIO lines.
- Offers flexibility for modules requiring serial communication or I²C-based expansion devices.

Power and Pinout

- Both PMOD connectors provide 3.3 V and 5 V power supply options, selectable via jumpers.
- Pinouts conform to PMOD specifications, ensuring compatibility with a wide range of third-party modules.
- Each connector exposes up to 12 pins, combining data, control, and power signals.

Sss

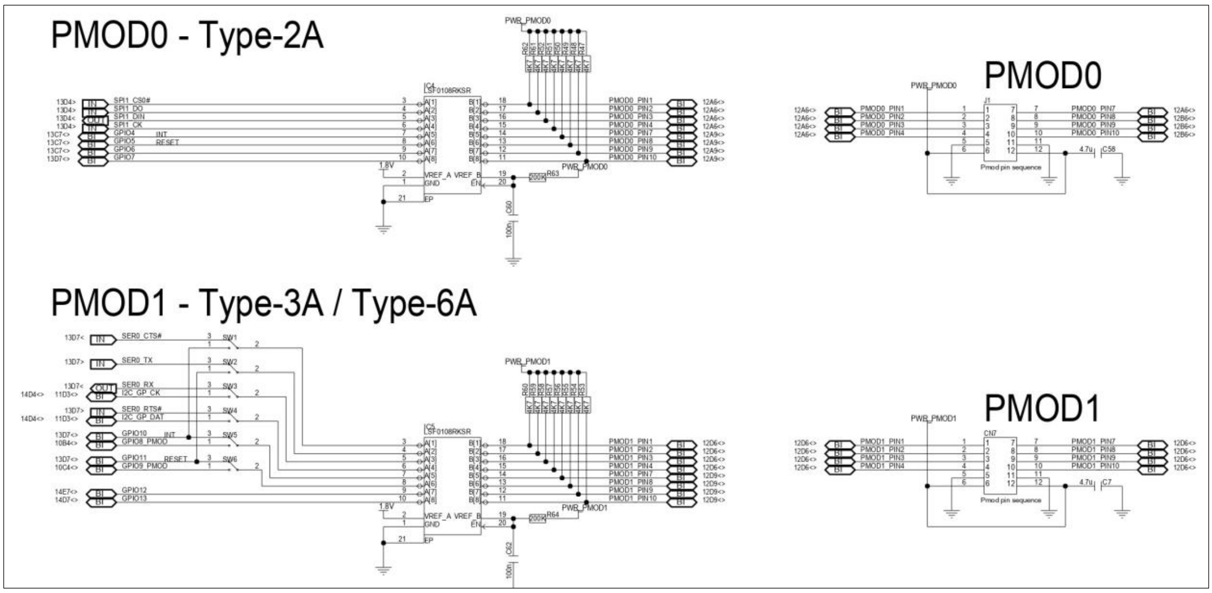


Figure 122. RZG2L-EVK PMOD Type 2A, 3A and 6A Interface

14.3.8.8 Analog to Digital Converter

The RZ/G2L SMARC Module Board (RTK9744L23C01000BE) provides access to the on-chip Analog-to-Digital Converter (ADC) through a 10-pin header. This interface allows developers to evaluate analog signal acquisition features of the RZ/G2L processor.

Features:

- Up to six ADC input channels (CH0–CH5) are available via the header.
- Each channel supports 1.8 V analog input (ADC_AVDD domain).
- Signals are routed directly from the RZ/G2L SoC pins to the connector for external measurement or application-specific wiring.
- Additional pins are provided for ADC trigger input and power references (1.8 V and 3.3 V).

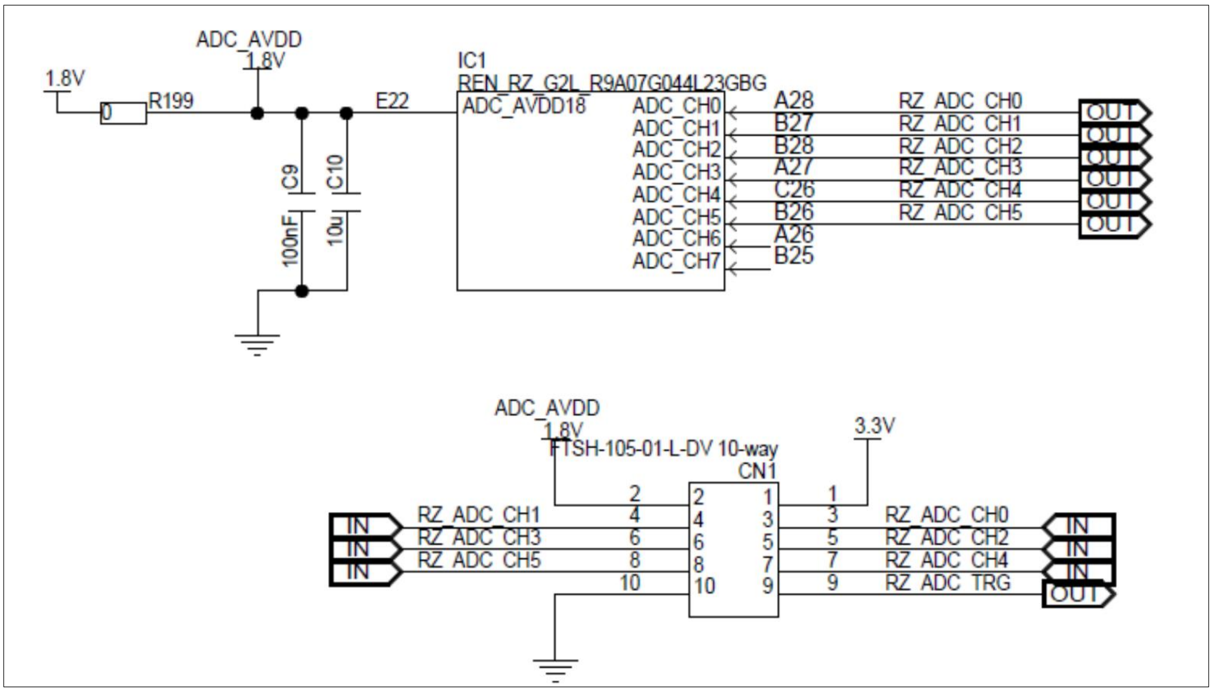


Figure 123. RZG2L-EVK Block Diagram of ADC

14.3.8.9 JTAG SWA Debug

The RZ/G2L-EVK provides a 10-pin header (CN2) for connection to an external debugger. This interface enables hardware-level debugging and supports both JTAG and Serial Wire Debug (SWD) protocols. Debug access is available for the Cortex-A55 application cores and the Cortex-M33 real-time core.

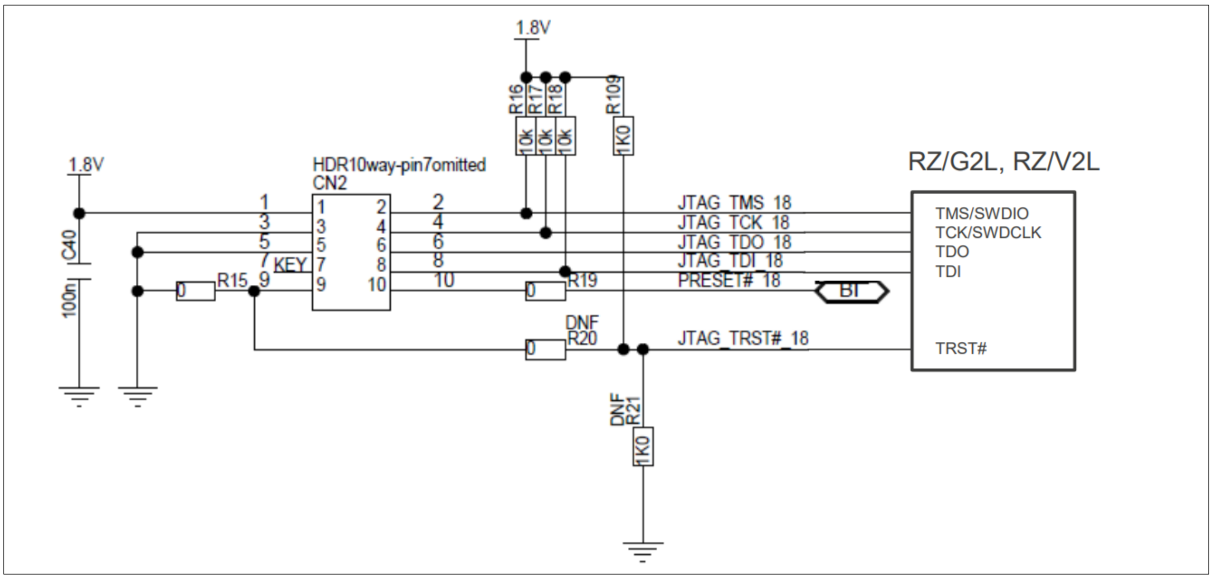


Figure 124. RZG2L-EVK Block Diagram of JTAG SWA Debug

14.3.9 Memory

The EVK's boards design uses four types of memory.

1. QSPI NOR Flash
2. eMMC Flash
3. DDR4 SDRAM

4. SD-Card

14.3.9.1 QSPI NOR Flash

The QSPI flash device is managed by the integrated SPI multi-I/O bus controller (SPIBSC) in the RZ/G2L. It is capable of operating in both single data rate (SDR) and double data rate (DDR) modes, supporting transfer speeds of up to 66 MHz in SDR and 50 MHz in DDR.

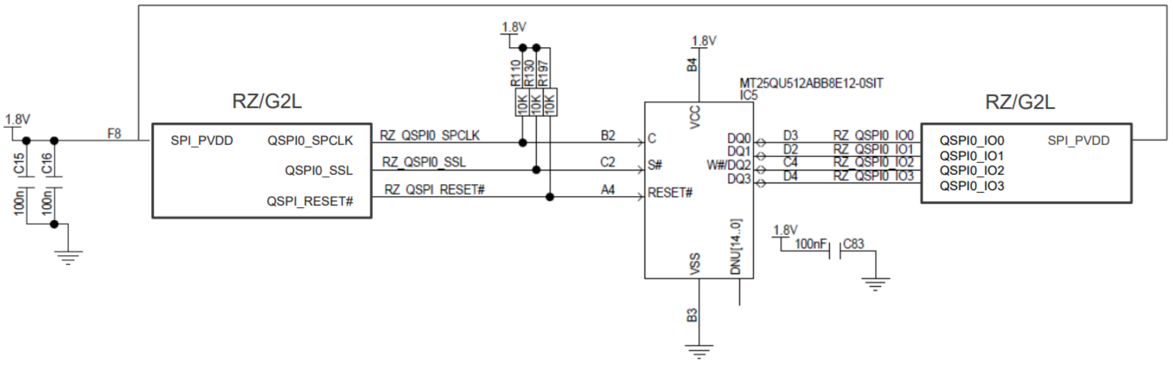


Figure 125. RZG2L-EVK Block Diagram of QSPI NOR Flash

14.3.9.2 DDR4 SDRAM

The DDR4 SDRAM is managed by the integrated DDR3L/DDR4 Memory Controller (MEMC) within the RZ/G2L. The controller supports DDR4-1600 devices with a 16-bit data bus and inline ECC capability. The interface is fully compliant with the JEDEC JESD79-4C standard.

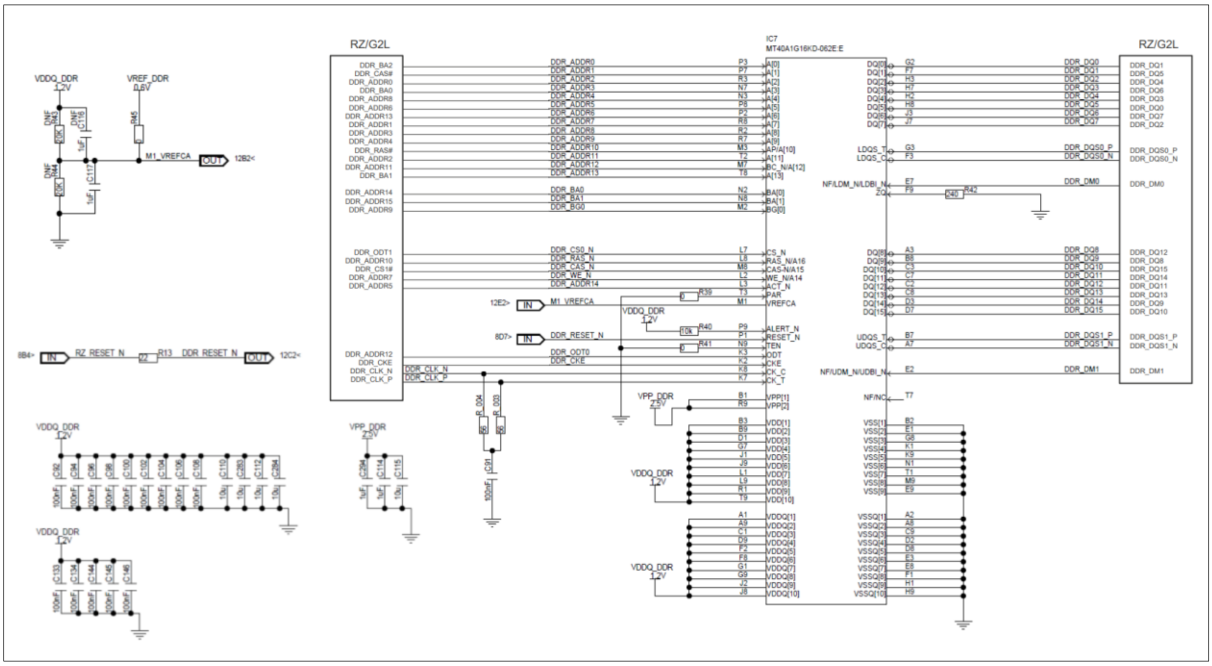


Figure 126. RZG2L-EVK Block Diagram of DDR4 SDRAM

14.3.9.3 eMMC Flash

The on-board eMMC device is connected to channel 0 of the integrated SD/MMC interface on the RZ/G2L. It operates alongside the microSD card interface as an alternative storage option. The eMMC can be selected when the SW_SD0_DEV_SEL switch is enabled (SW1-2 set to OFF, selecting SD/MMC). This interface follows the JEDEC v4.51 specification and supports HS200 transfer mode.

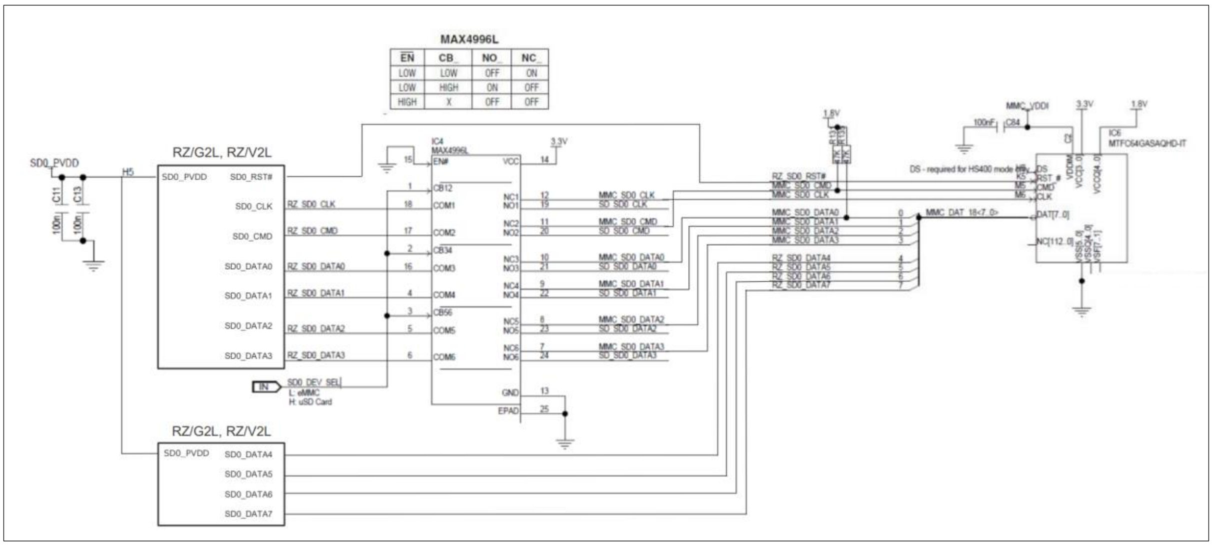


Figure 127. RZG2L-EVK eMMC Flash

14.4 RZ/V2H-EVK Board

14.4.1 Overview

Evaluation platform for the Renesas RZ/V2H, intended for functional evaluation, performance testing, and application software development. The kit follows a module-plus-carrier approach: a Secure Evaluation Board (CPU board) hosts the SoC and core functions, and an EVK Expansion Board (EXP board) adds user I/O (HDMI, audio, Pmod, etc.).

Processor

- RZ/V2H application MPU (Arm® Cortex®-A55 cluster with integrated peripheral subsystems and security features per device option).
- Suitable for vision/HMI/industrial use cases.

Memory and Storage

- System DRAM: LPDDR4X, 64 Gb × 2 on the CPU board.
- Boot/firmware: NOR flash 512 Mb.
- Removable: dual microSD card slots on the EXP board.
- eMMC (if fitted by variant): supported via SDHI.

Display and Camera

- Camera: up to 4× MIPI® CSI-2® connectors for image sensors.
- Display: HDMI® Type-A ×1 on the EXP board; MIPI-DSI via module (if supported by variant/bridge).

Connectivity

- Ethernet: 2× Gigabit Ethernet connectors (RGMII/RMII per wiring and PHY).
- USB:
    - 2× USB 3.2 Gen 2 Type-A

  - 1× USB 2.0 micro-AB
  - 1× USB 2.0 Type-A
- Serial: USB micro-B debug UART.
- I^{2}C/SPI/GPIO: exposed on board headers and via SMARC edge interconnect.

High-Speed Expansion

- PCIe® x4 slot ×1 on the EXP board for add-in cards or accelerators.

Audio

- MIC input ×1, Headphone output ×1, AUX line input ×1 on the EXP board.
- Codec and jack population follow the kit BOM.

Power and Clocks

- Power input: USB-PD Type-C ×1 (kit default).
- PMIC: generates SoC rails (core/IO/analog) and peripheral supplies.
- Clocking: programmable clock generator provides system/peripheral references.

Debug and Expansion

- Debugger interface: 10-pin Arm® Cortex® JTAG/SWD header.
- Pmod™ connectors: 4× general-purpose expansion headers.
- SMARC edge connector exposes additional SoC interfaces for custom carriers.

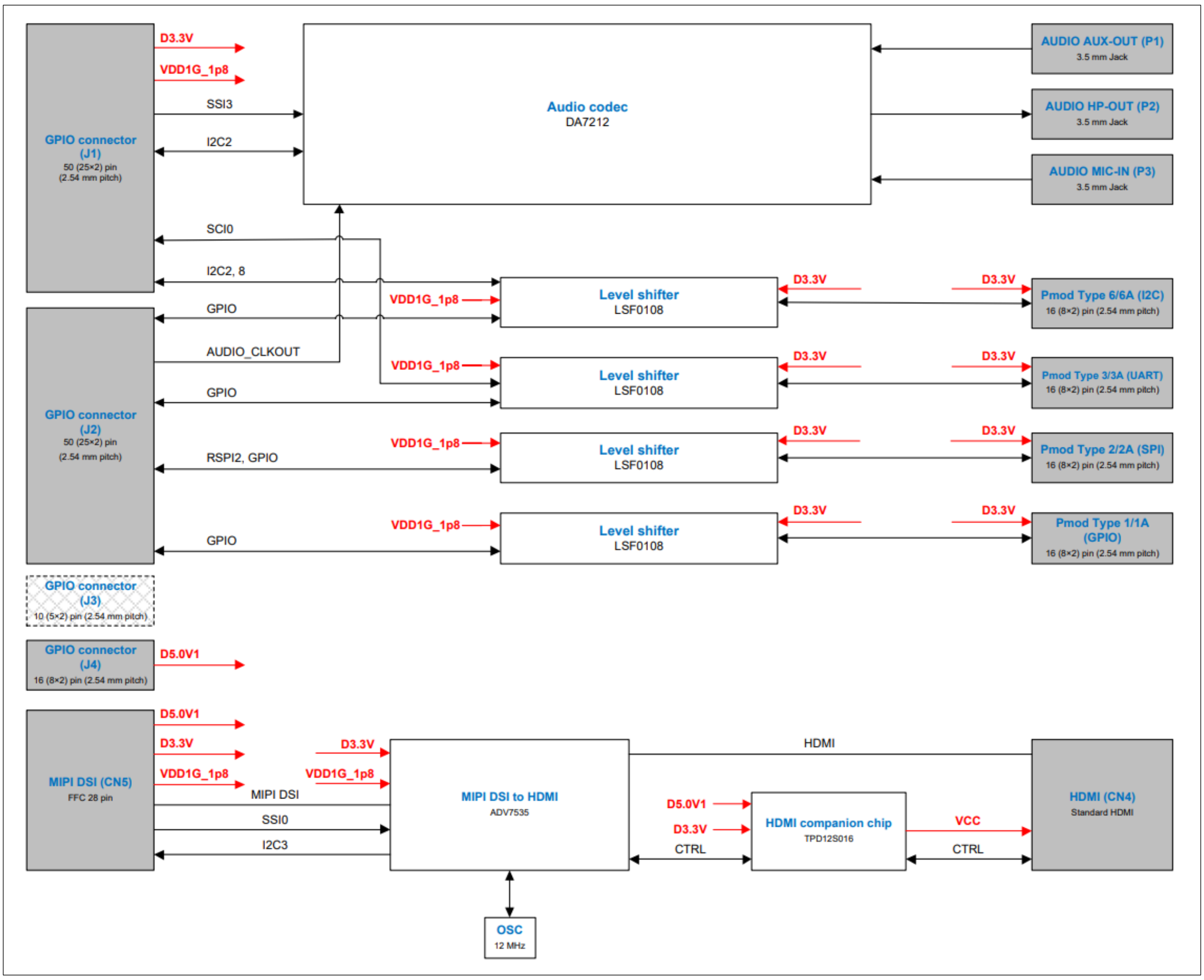


Figure 128. RZ/V2H-EVK Board Block Diagram (Source: *RZ/V2H Evaluation Board Kit (Secure Type) Hardware Manual*, p. 12)

14.4.2 Part Layout

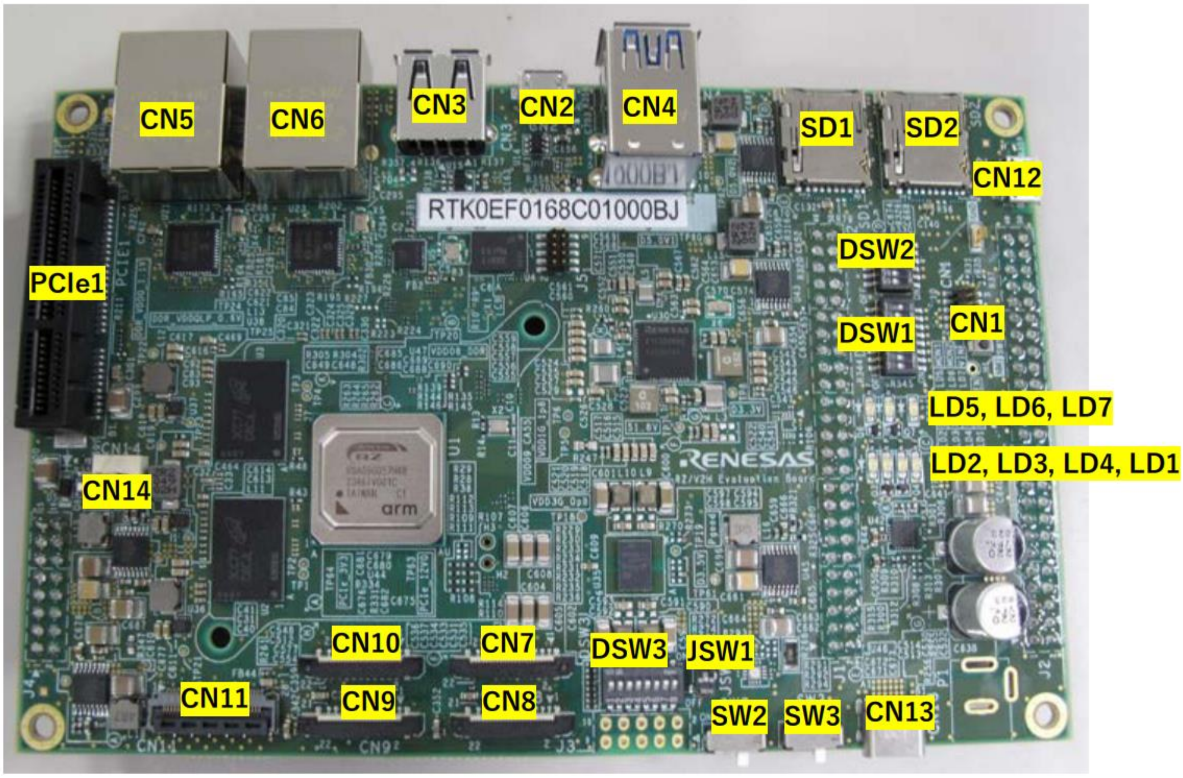


Figure 129. RZ/V2H-EVK Part Side

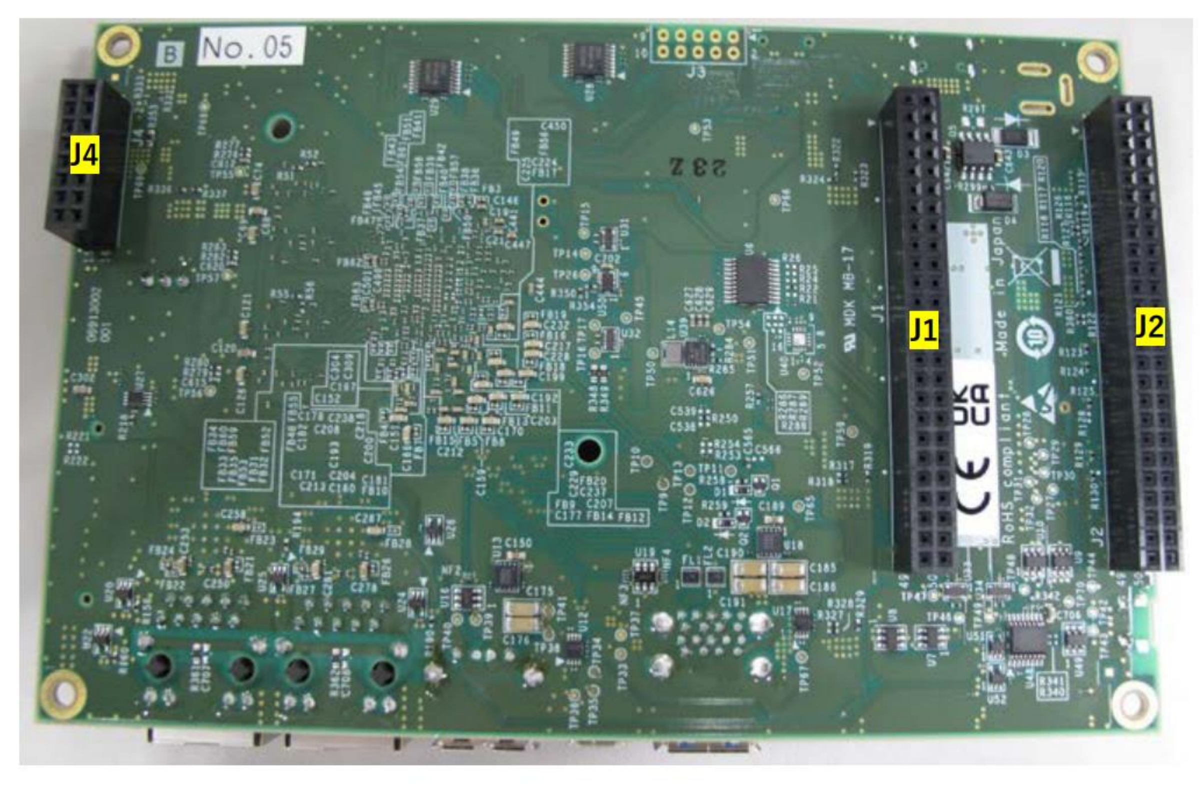


Figure 130. RZ/V2H-EVK Solder Side

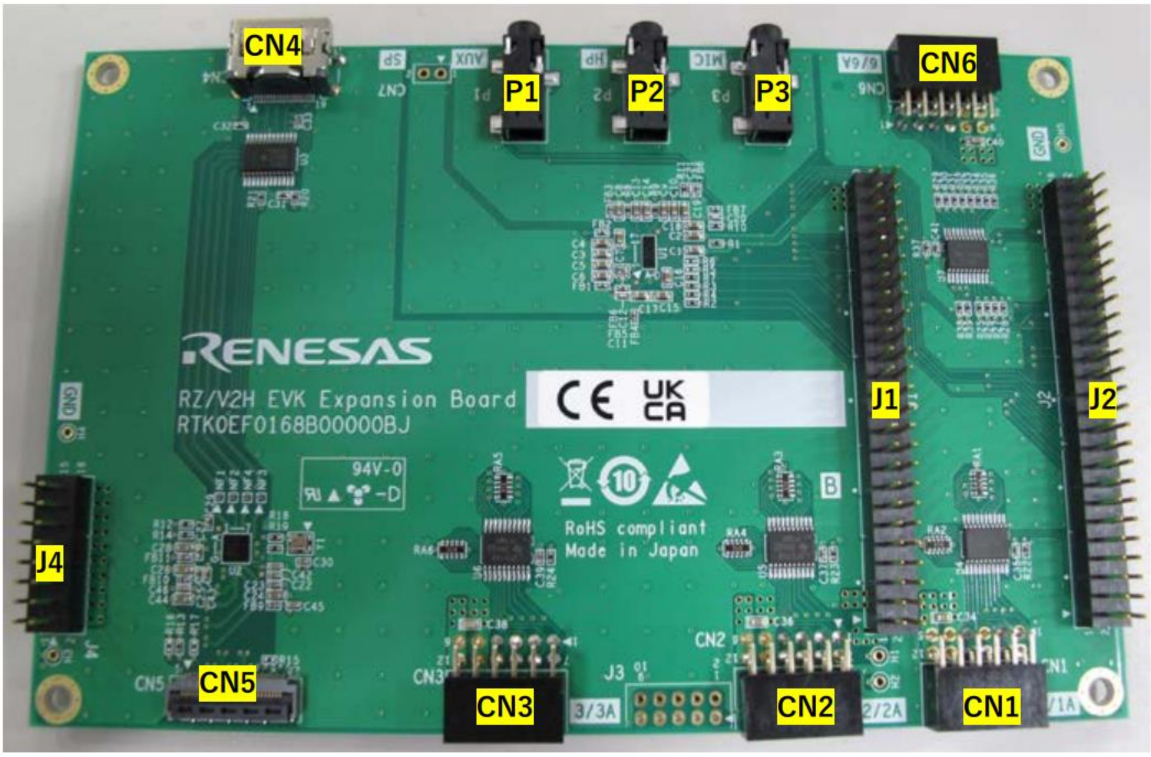


Figure 131. RZ/V2H-EVK Part Side

14.4.3 Overview of Connectors

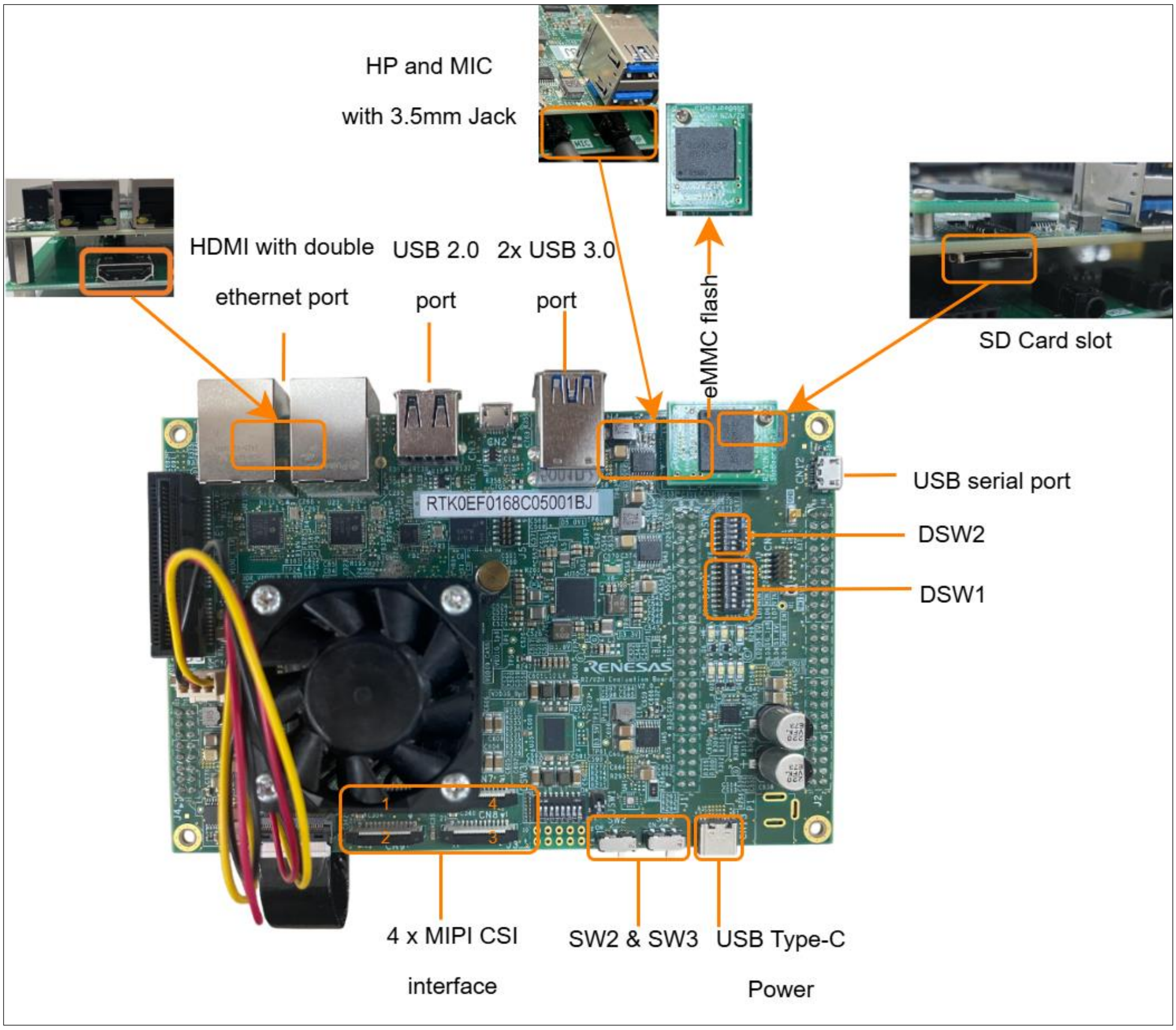


Figure 132. RZ/V2H-EVK Overview of Connectors

14.4.4 Power Supply

14.4.4.1 USB Type-C Power

The RZ/V2H-EVK is powered through a USB Type-C receptacle controlled by an on-board USB Power Delivery (PD) controller that negotiates the input profile before enabling rails. For full-feature operation (PCIe card, dual GbE, multi-camera, HDMI, multiple USB3 devices), use a USB-PD 100 W adapter (20 V × 5 A) with a 5 A e-marked USB-C cable.

14.4.4.2 Power Supply Voltage

Table 22: RZ/V2H-EVK Power Supply Voltage

Item	Description
Power supply voltage	• VDD (core): 0.8 V • VDD (CA55): 0.8 V or 0.9 V • VDD (ADC, TSU, OTP): 1.8 V • VDD (DDR IO): 1.1 V, 0.6 V (only 0.6 V: for LPDDR4X) • VDD (MIPI DPHY): 1.2 V, 1.8 V (only 1.8 V: for MIPI CSI-2) • VDD (others): 1.8 V, 3.3 V

14.4.4.3 Power Supply Regulation

The RZ/V2H-EVK board uses a dedicated Power Management IC (PMIC) to generate and sequence the supply rails required by the SoC and peripherals. See Section 14.4.5 Power Management Integrated Circuit (PMIC) for more detailed information.

14.4.5 Power Management Integrated Circuit (PMIC)

The RAA215300 is a high-performance, low-cost, 9-channel PMIC designed for 32-bit and 64-bit MCU and MPU applications. It supports DDR3, DDR3L, DDR4, and LPDDR4 memory power interfaces. The internally compensated regulators, built-in Real-Time Clock (RTC), 32kHz crystal oscillator, and coin cell battery charger provide a highly integrated, small footprint power solution ideal for System-On-Module (SOM) applications.

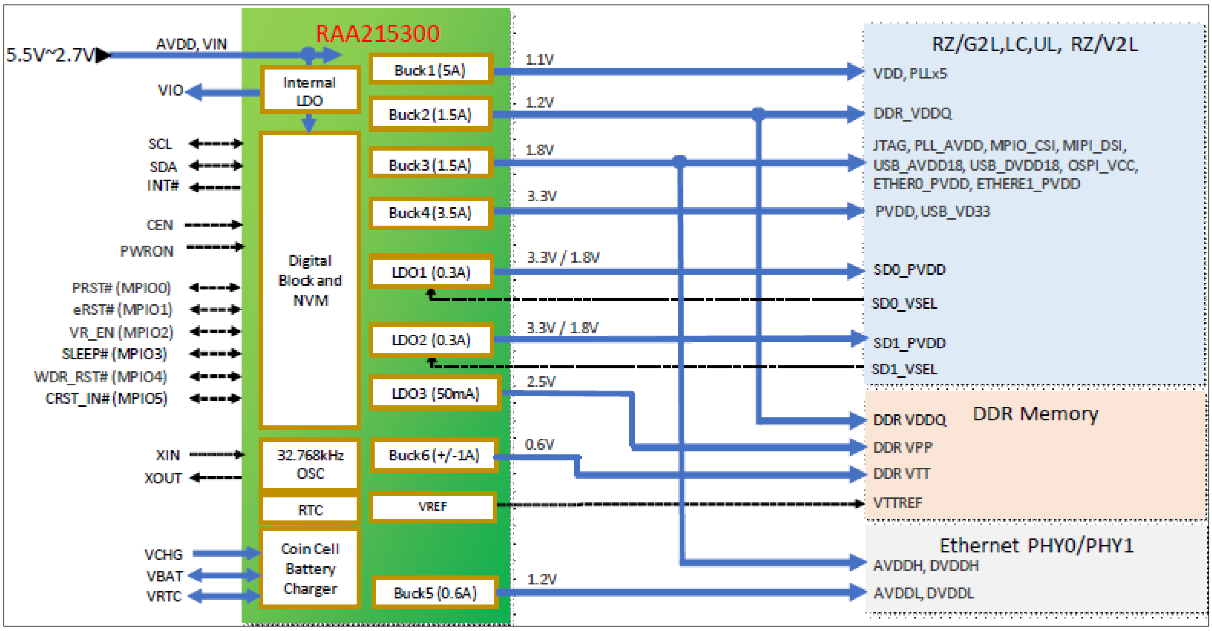


Figure 133: RAA215300 Block Diagram

14.4.6 RESET Control

Describe the power/reset sequence and the roles of the on-board slide switches and indicators on the RZ/V2H Secure Evaluation Board (CPU board) when used with the EVK Expansion Board.

Power/Reset Sequence

1. Set DIP switches to the desired boot configuration before applying power.
2. Verify the power slide switches SW2 and SW3 are OFF.
3. Connect the USB Type-C power cable to CN13 on the CPU board.
4. Turn SW3 = ON to enable USB-PD input. LD2 and LD7 should illuminate.
5. Turn SW2 = ON to enable the PMIC outputs. LD1, LD3, and LD4 illuminate and the RZ/V2H boots.

Table 23: RZ/V2H-EVK Switches Function

Switch	Name/Function	ON	OFF
SW2	PMIC (RAA215300) Enable	Enables PMIC outputs (rails sequenced and released).	Disables PMIC outputs (rails off).
SW3	USB-PD Input Enable	Enables USB-PD power input from the Type-C port	Disables USB-PD power input.

LED cues:

- LD2, LD7 = USB-PD power path active (after SW3 = ON).
- LD1, LD3, LD4 = PMIC rails enabled and system starting (after SW2 = ON).

14.4.7 Clock Configuration

The RZ/V2H integrates a Clock Pulse Generator (CPG) and a Power Management Unit (PMU) to manage all timing and power sequencing across the SoC. Together, they provide:

- Generation and distribution of system and peripheral clocks.
- Reset sequencing for reliable initialization.
- Boot-time configuration of CPU and peripheral frequencies.
- Dynamic power management, including clock gating and domain-level power control.

14.4.7.1 Clock Generating and Control Functions

The following functions are provided.

- The clocks to be supplied to various units are generated from external input clocks or PLL output clocks.
- Especially when the CA55 is cold boot, the CA55 operating frequency by the external pin is set.
- The selection of frequency divider ratio is implemented (the setting can be changed by the register).
- The clock path is selected by the clock selector (the setting can be changed by the register).
- Clock supply ON/OFF control (the setting can be changed by register)
- SSCG control of PLL, multiplication ratio setting, ON/OFF control (the setting can be changed by the register)

Table 24: PLL Table Overview

PLL	Details	Initial Frequency	SSCG Default Value
PLLCM33	For the MCPU bus and various units (PD_AWO area)	1,600 MHz	OFF (fixed)
PLLCLN	For buses other than the MCPU bus and various units	1,600 MHz	OFF (fixed)
PLLDTY	For buses other than the MCPU bus and various units	1,600 MHz	Depends on MD_CLKS
PLLCA55	CA55	1,700 MHz	Depends on MD_CLKS
PLLVDO	CRU / ISP / GE3D / CA55	1,260 MHz	Depends on MD_CLKS
PLLETH	GBETH / DRP1 / DRP-AI / DSI	1,000 MHz	OFF (fixed)
PLLDSI	DSI / LCDC	297 MHz	OFF (fixed)
PLLDDR0	For DDR ch. 0	800 MHz	OFF (fixed)
PLLDDR1	For DDR ch. 1	800 MHz	OFF (fixed)
PLLGPU	GE3D	1,260 MHz	ON (fixed)
PLLDRP	DRP1 / DRP-AI / CA55	1,260 MHz	Depends on MD_CLKS

14.4.7.2 Clock Pin Specification

The RZ/V2H device provides a wide range of dedicated clock input and output pins to support system timing, peripheral interfaces, and external device synchronization.

This table lists all clock-related pins on RZ/V2H—showing pin name, I/O direction, function, maximum supported frequency, and the supplying/receiving unit

Table 25: RZ/V2H-EVK Clock Pin Specification

Pin Name	Input	Output	Frequency[*1]	Supply Source/ Destination Unit
LSI Clock				
QXTAL	Output	24 MHz main clock	24 MHz	CPG
QEXTAL	Input	24 MHz main clock	24 MHz	CPG
RTXOUT	Output	32.768 kHz real-time clock	32.768 kHz	RTC
RTXIN	Input	32.768 kHz real-time clock	32.768 kHz	RTC
AUDIO_XTAL	Output	4 to 48 MHz audio clock (Internal OSC mode)	48 MHz	ADG
AUDIO_EXTAL	Input	4 to 48 MHz audio clock (Internal OSC mode) 4 to 50 MHz audio clock (Input clock from the external device)	50 MHz	ADG
AUDIO_CLKB[*2]	Input	4 to 50 MHz audio clock-in	50 MHz	ADG
AUDIO_CLKC[*2]	Input	4 to 50 MHz audio clock-in	50 MHz	ADG
AUDIO_CLKOUT[*2]	Output	2 to 25 MHz audio clock-out (Half frequency of input clock)	25 MHz	ADG
Debug Interface				
TCK_SWCLK	Input	Test clock pin for the on-chip emulator functions as the SWCLK pin in serial wire debug (SWD) mode	20 MHz	CST
Expanded serial peripheral interface (xSPI)				
XSPI0_CKP[*3]	Output	Clock output pins (positive)	133 MHz	xSPI
XSPI0_CKN[*3]	Output	Clock output pins (negative)	133 MHz	xSPI
SD/MMC interface				
SD0CLK	Output	Output the clock signal to external SD/MMC device	200 MHz	SD0
SD Interface				
SD1CLK	Output	Output the clock signal to external SD device	200 MHz	SD1
SD2CLK	Output	Output the clock signal to external SD device	200 MHz	SD2
PCIe Gen3				
PCIE_REFCLKP0	I/O	Differential reference clock (positive), 100 MHz ±300 ppm	100 MHz	PCIE0
PCIE_REFCLKN0	I/O	Differential reference clock (negative), 100 MHz ±300 ppm	100 MHz	PCIE0
PCIE_REFCLKP1	I/O	Differential reference clock (positive), 100 MHz ±300 ppm	100 MHz	PCIE1
PCIE_REFCLKN1	I/O	Differential reference clock (negative), 100 MHz ±300 ppm	100 MHz	PCIE1
Gbit Ethernet ch. 0				
ET0_MDC	Output	Output management data clocks	2.5 MHz	GBETH0
ET0_RXC_RXCLK	Input	RX Clocks	125 MHz	GBETH0
ET0_TXC_TXCLK	I/O	TX Clocks	125 MHz	GBETH0
Gbit Ethernet ch. 1				
ET1_MDC	Output	Output management data clocks	2.5 MHz	GBETH1
ET1_RXC_RXCLK	Input	RX Clocks	125 MHz	GBETH1
ET1_TXC_TXCLK	I/O	TX Clocks	125 MHz	GBETH1

MIPI DSI				
DSI_DPCLK	Output	Output clocks (positive)	750 MHz	DSI
DSI_DNCLK	Output	Output clocks (negative)	750 MHz	DSI
MIPI CSI-2 ch. n (n = 0 to 3)				
CSIn_CLKP	Input	Input clocks (positive)	1050 MHz	CRUn
CSIn_CLKN	Input	Input clocks (negative)	1050 MHz	CRUn
Serial peripheral interface (RSPI) ch. n (n = 0, 1, 2)				
RSPCKn	I/O	Synchronous clock signal	50 MHz	RSPIn
Serial communication interface (RSCI) ch. n (n = 0 to 9)				
SCKn	I/O	Clock pins (simple SPI mode)	37.5 MHz	RSCIn
SCLn	I/O	I2C clocks (simple I2C mode)	400 kHz	RSCIn
I2C bus interface (RIIC) ch. n (n = 0 to 8)				
SCLn	I/O	Clock pins with N-ch open drain	1 MHz	RIICn
I3C bus interface (I3C)				
SCL30	I/O	Clock Pin	3 MHz	I3C0
Pulse density modulation interface (PDM) ch. n (n = 0 to 5)				
PDMCLK00	Output	Output PDM sampling clocks	2.4 MHz	PDM0
PDMCLK01	Output	Output PDM sampling clocks	2.4 MHz	PDM0
PDMCLK02	Output	Output PDM sampling clocks	2.4 MHz	PDM0
PDMCLK10	Output	Output PDM sampling clocks	2.4 MHz	PDM0
PDMCLK11	Output	Output PDM sampling clocks	2.4 MHz	PDM0
PDMCLK12	Output	Output PDM sampling clocks	2.4 MHz	PDM0
Serial sound interface (SSIU) ch. n (n = 0 to 9)				
SSIn_SCK	I/O	Output clocks	12.5 MHz	SSIU

- Note 1. The clock frequencies are either variable or fixed. In the case of the variable clocks, the maximum frequencies which can be output are listed for the output clocks and the maximum frequencies which can be input to this LSI are listed for the input clocks.
- Note 2. For use of these pins, the function selection of multiplexed function pins is required.
- Note 3. CKP and CKN waves have opposite phases.

14.4.8 Peripheral Interface

14.4.8.1 Gigabit Ethernet

The CPU board of the V2HEVK has two Ethernet interfaces. The RJ-45 connectors (CN5 and CN6) are connected to the Ethernet interface of the RZ/V2H via the Ethernet PHY IC. Connecting this interface to a public line is prohibited.

14.4.8.2 USB 2.0 Ports

USB 2.0 Host (CH1): Type-A connector CN3 is wired to USB2.0 channel 1 on the RZ/V2H. Use this port to connect full-/high-speed USB peripherals (flash drives, keyboards, Wi-Fi dongles, etc.).

USB 2.0 DRD (CH0): Micro-AB connector CN2 is wired to USB2.0 channel 0 and operates as a Dual-Role Device (host/device) depending on cable/ID pin. Suitable for OTG use, device mode console, or host mode with an adapter.

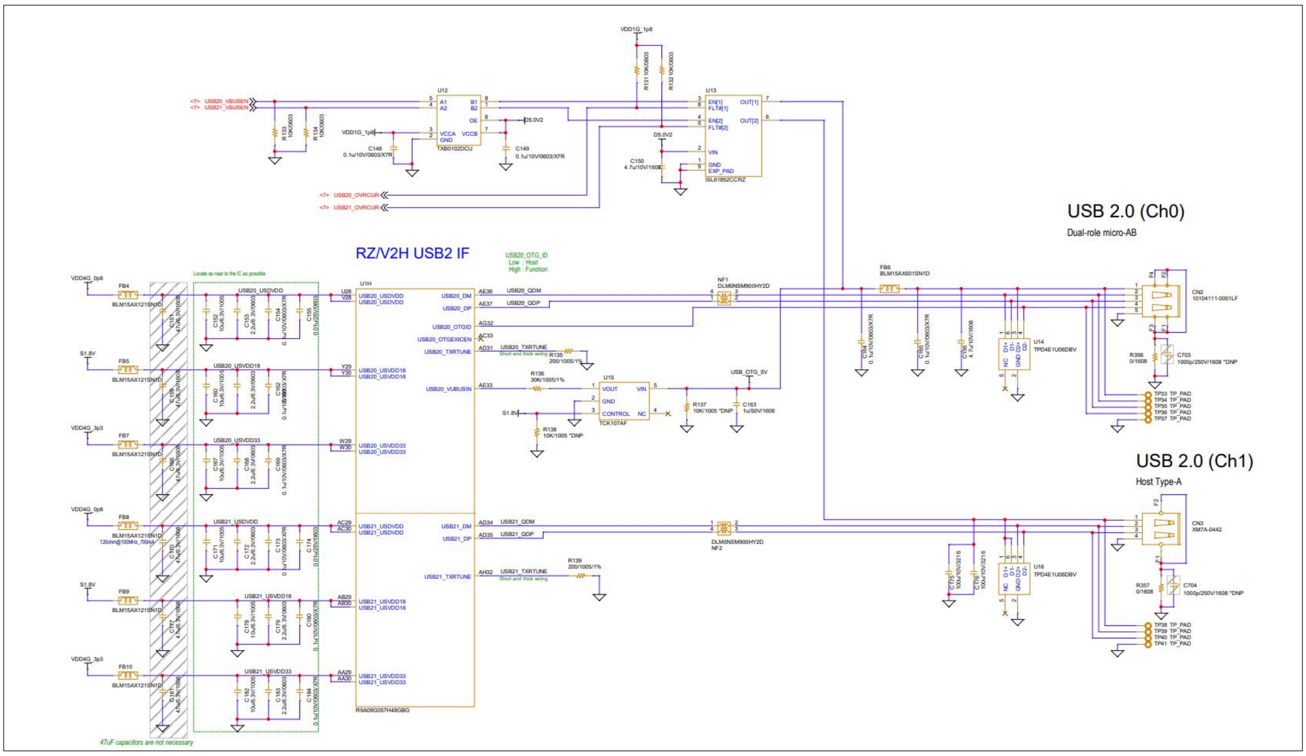


Figure 134. RZ/V2H-EVK USB 2.0 Port Schematic

14.4.8.3 USB 3.2 Ports

The RZ/V2H-EVK provides one USB 3.2 Gen2 Type-A host connector (CN4), supporting Superspeed+ (10 Gbps) signaling. This interface allows connection of high-speed external devices such as SSDs, cameras, and expansion adapters.

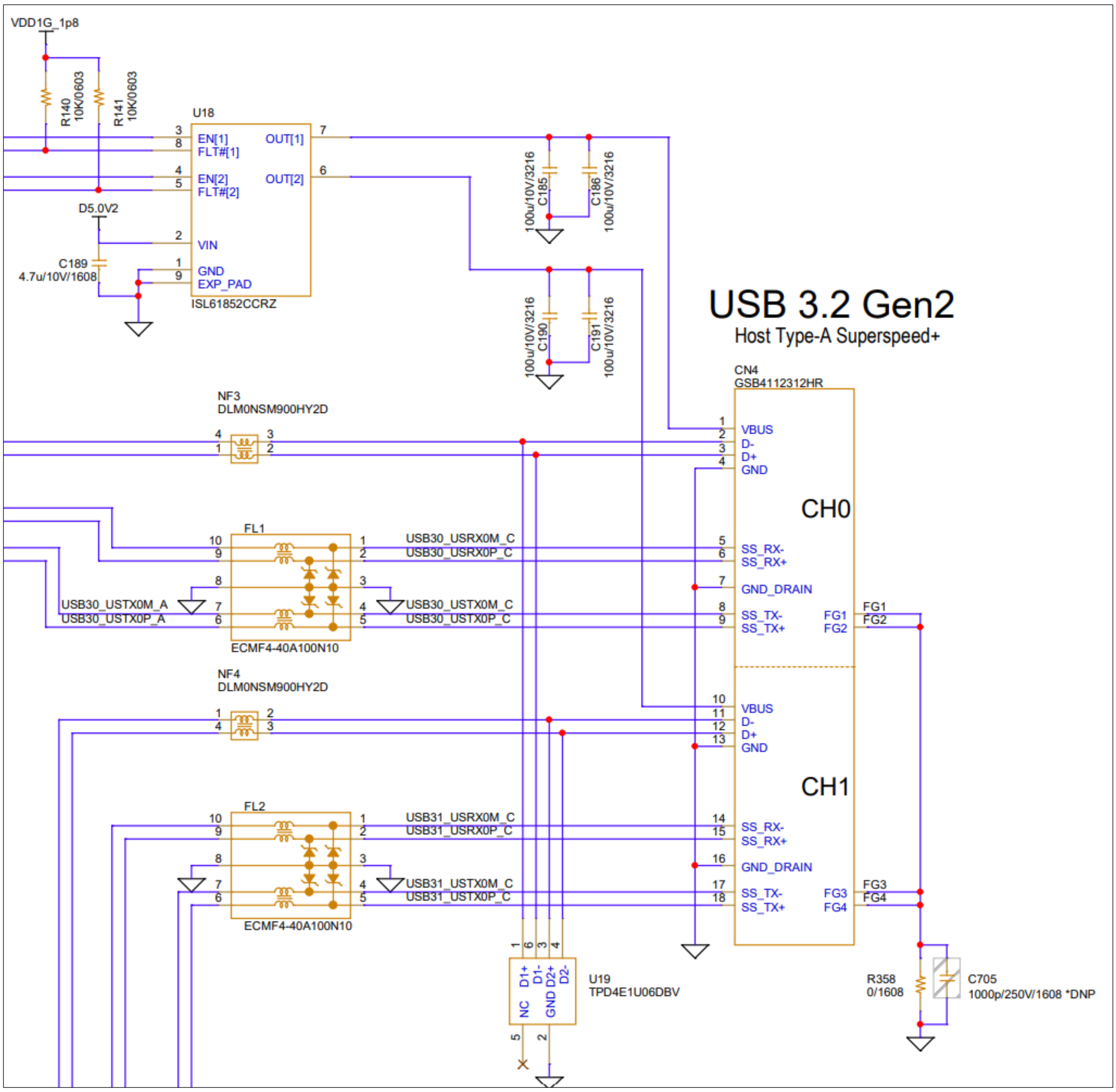


Figure 135. RZ/V2H-EVK USB 3.2 Module

14.4.8.4 MIPI CSI Interface

The RZ/V2H-EVK exposes four independent MIPI-CSI-2 ports (CH0–CH3) for image sensors. Each channel is brought to its own 22-pin camera connector (CN7/CN8/CN9/CN10) and is protected with ESD arrays and common-mode chokes. High-speed pairs are length-matched and routed as 100-Ω differential lines.

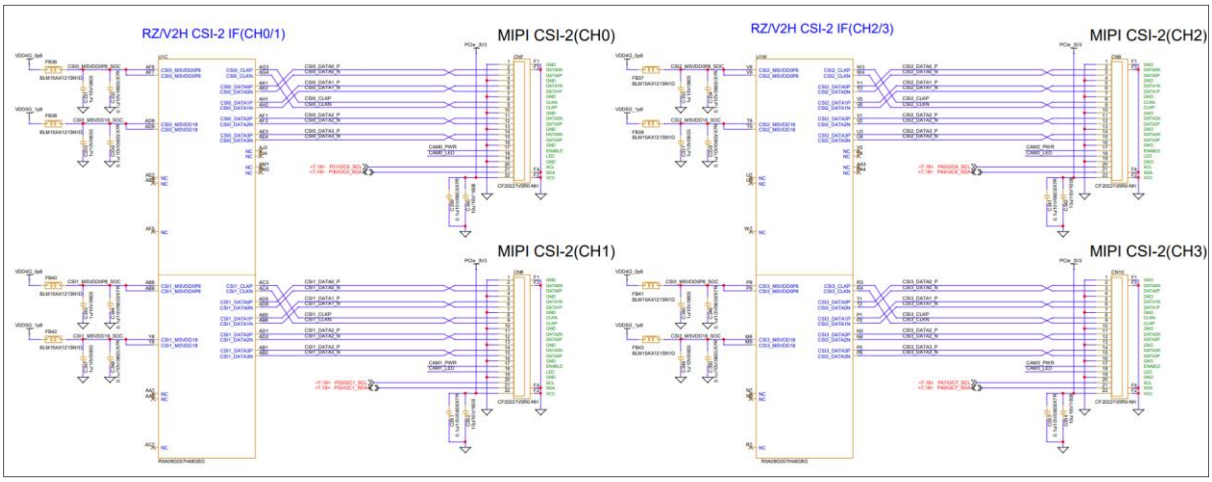


Figure 136. RZ/V2H MIPI CSI-2 Interface Schematic

14.4.8.5 Audio Subsytem with 3.5mm Jack

The EVK's audio path is codec-centric: a DA7212 codec on the EXP board connects to the SoC over I²C (control) and SSI3 (I²S data), with AUDIO_CLKx from the ADG providing the master clock. It exposes three 3.5-mm jacks—MIC-IN, HP-OUT, and AUX-OUT—and can be extended with PDM mics or S/PDIF as needed.

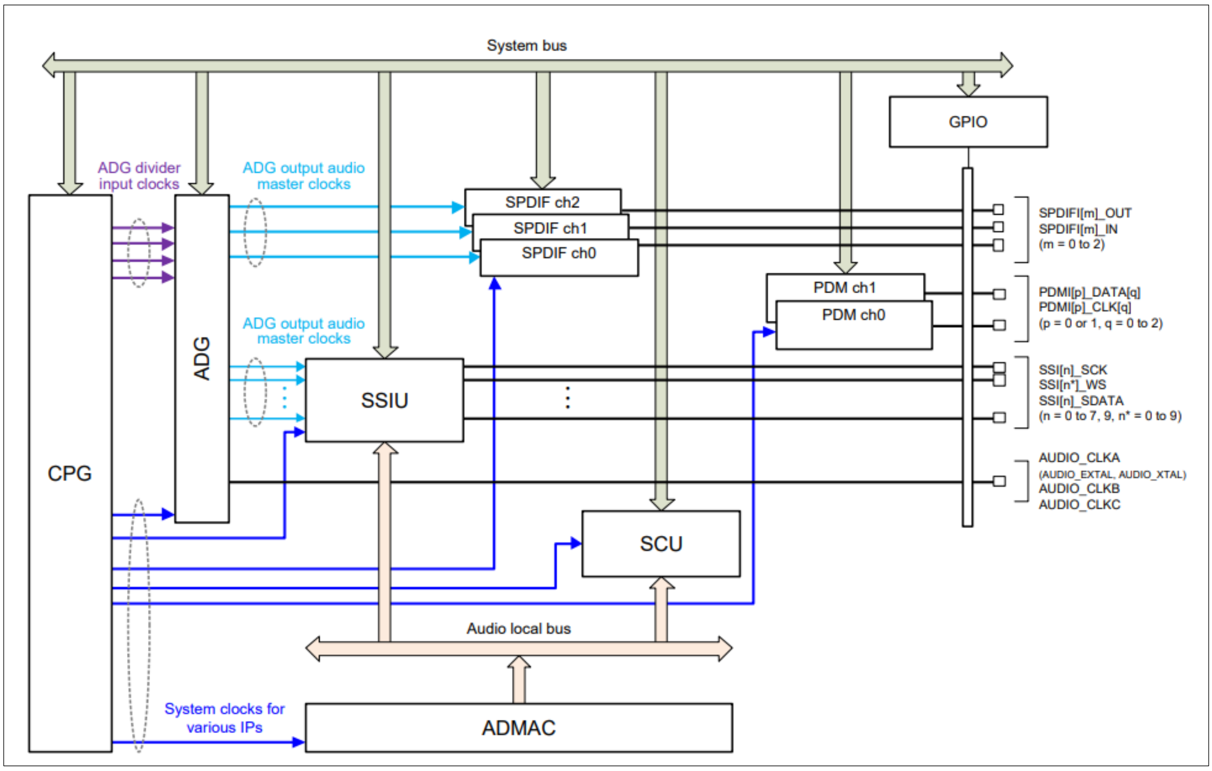


Figure 137. RZ/V2H-EVK Block Diagram of Audio Module

14.4.8.6 HDMI Display Subsystem

RZ/V2H-EVK drives HDMI through a MIPI-DSI → HDMI bridge. The SoC' s MIPI-DSI interface connects to an ADV7535 bridge, which converts DSI video to TMDS (three data pairs plus one clock pair). These TMDS pairs route to the micro-HDMI Type-D connector.

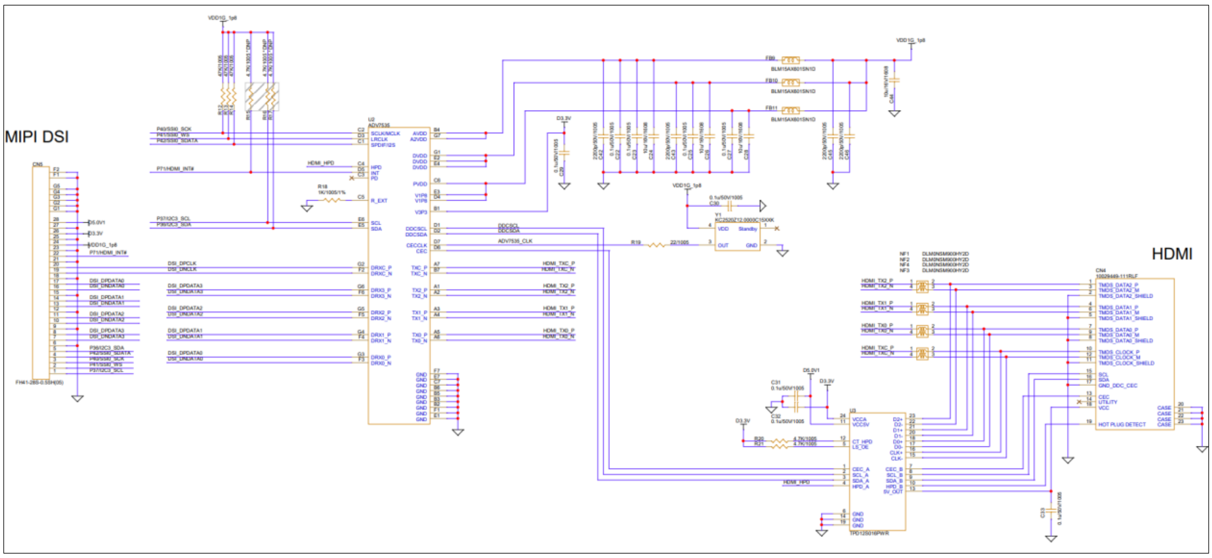


Figure 138. RZ/V2H-EVK HDMI Interface

14.4.8.7 uSD-card Interface

The interface shown below reflects RZ/V2H-EVK Rev.1: two microSD connectors

- SD1 → SDHI0 (boot-capable). This interface supports ROM boot. Insert the card before power-up/reset and select SD boot with the board's boot DIP settings. Refer to Section 16.2. Boot Mode Reference (Non-SCIF) of RZV2H-EVK for more detailed information.
- SD2 → SDHI1 for removable storage.

In EVK Rev.2, the layout changes to one microSD slot plus one on-board eMMC device (boot-capable), replacing the second microSD.

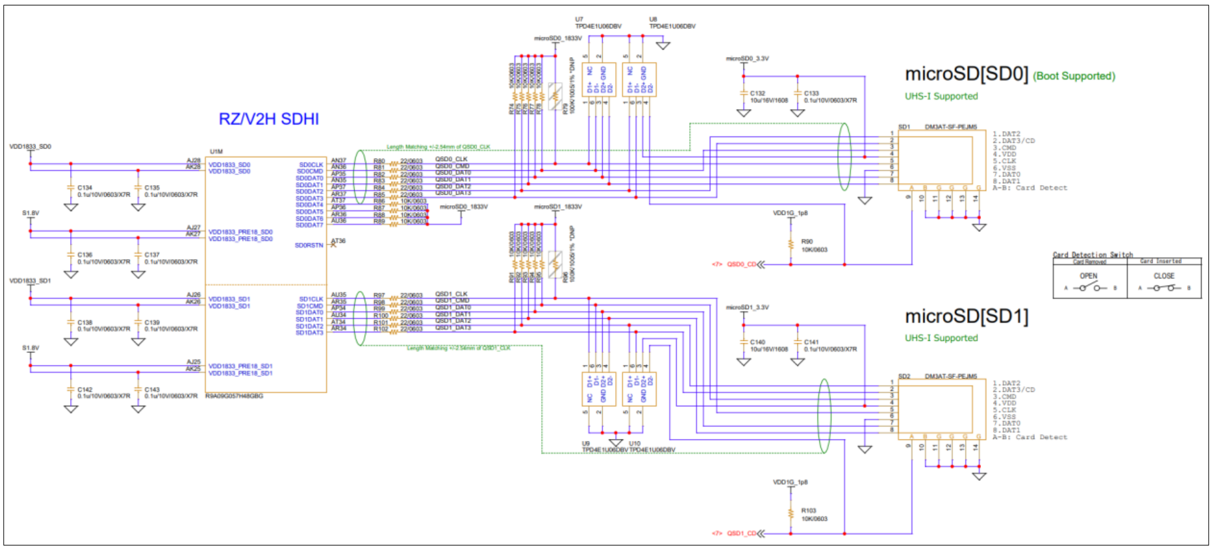


Figure 139. RZ/V2H-EVK microSD interface

14.4.8.8 PMOD Interface

The expansion board provides four PMOD headers, each translated from the SoC's 1.8 V I/O to 3.3 V through LSF0108 level shifters. All headers follow the 2×6 PMOD pin format with +3.3 V and GND on each row.

- PMOD Type-1A (GPIO) – General-purpose digital I/O. Eight bidirectional lines for LEDs, buttons, or simple logic control.
- PMOD Type-2A (SPI) – SPI host interface (SCLK, MOSI, MISO, CS) plus two auxiliary GPIOs for IRQ/RESET. Suitable for ADC/DAC, sensors, and small displays.
- PMOD Type-3A (UART) – Asynchronous serial interface (TXD, RXD, CTS, RTS) at 3.3 V levels; remaining pins available as GPIO.
- PMOD Type-6/6A (I²C) – I^{2}C host interface (SCL, SDA) with 3.3 V pull-ups and extra pins for INT/RESET or GPIO on many modules.

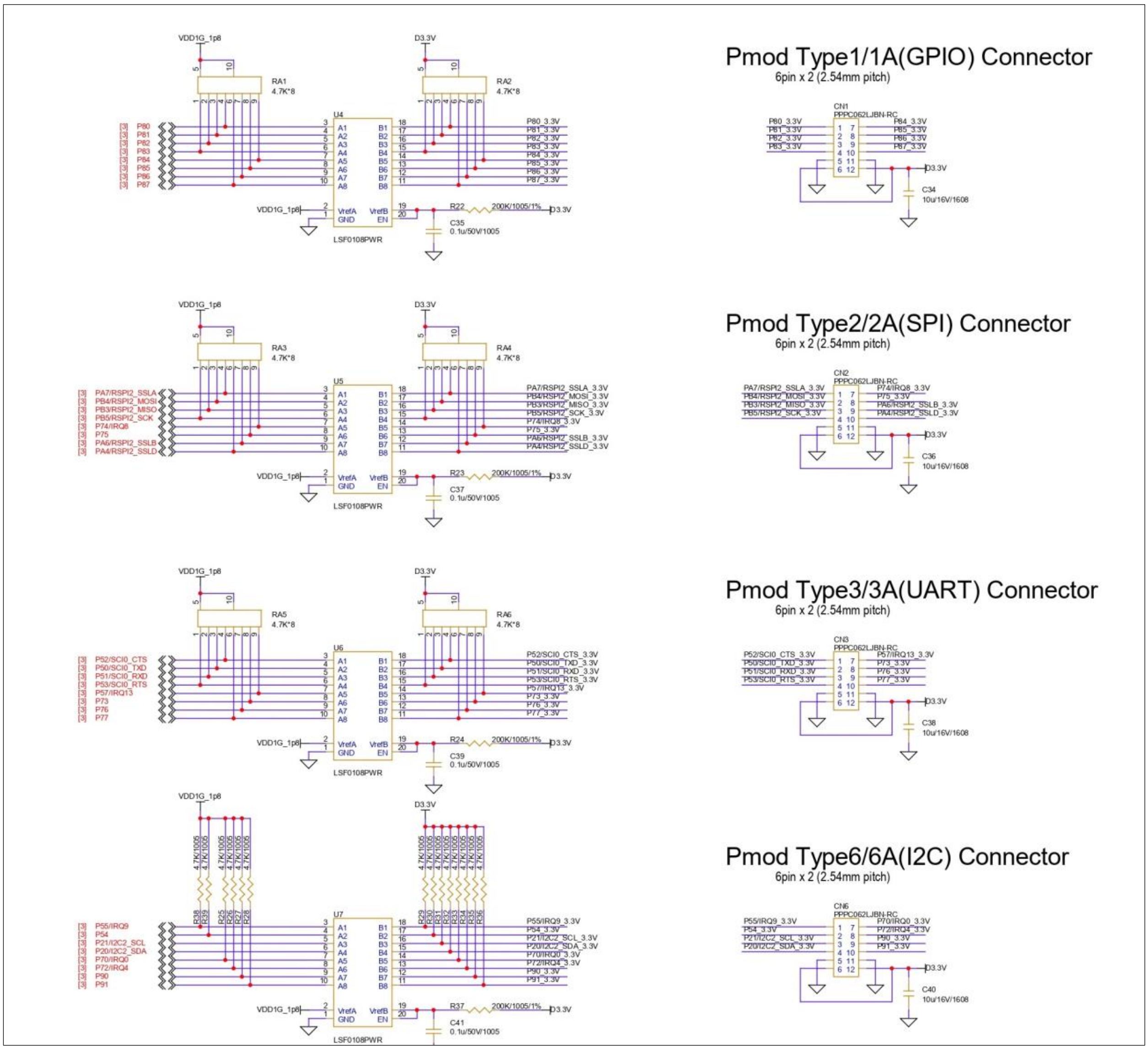


Figure 140. RZ/V2H-EVK PMOD Interface

14.4.8.9 Analog to Digital Converter

RZ/V2H-EVK exposes the SoC's on-chip SAR ADC for basic voltage sensing and evaluation. The CPU board routes up to 8 single-ended input channels (ANI000–ANI007).

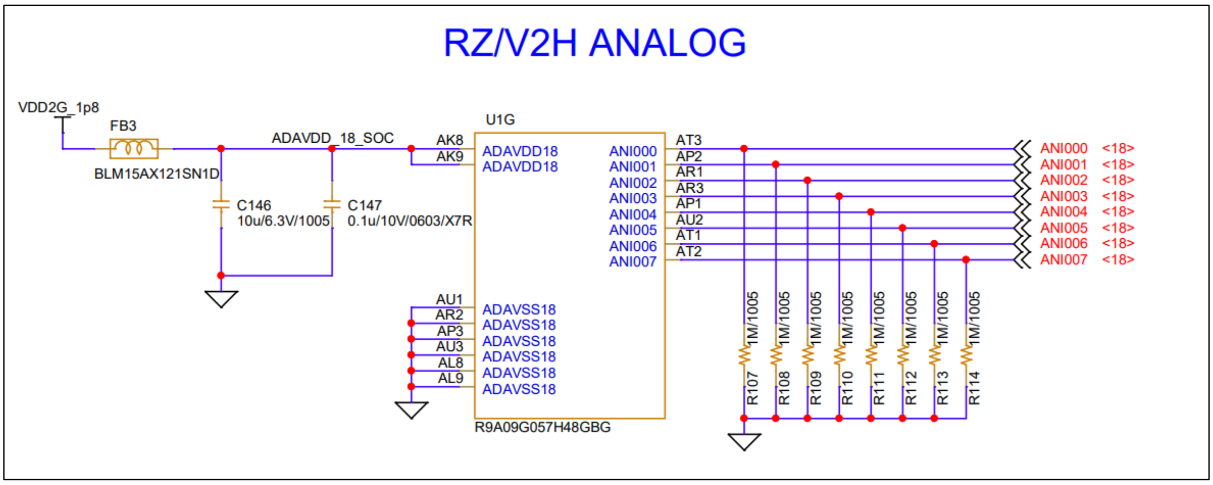


Figure 141. RZ/V2H Analog Module

14.4.8.10 Debug Interface

Connect the ICE to the Arm JTAG connector (CN1). For the ICE having confirmed the connection, contact Renesas Electronics' sales representative.

Connector

- 10-pin, 1.27 mm (0.05") Arm Cortex Debug header (half-pitch).
- Voltage reference: 1.8 V (VDD1G_1p8 on the board). Use probes/cables that support 1.8 V I/O.

Supported modes

- SWD (2-wire): TMS/SWDIO, TCK/SWCLK, nRESET, VTref, GND.
- Full JTAG (4-wire): adds TDI and TDO.
- nTRST is brought out via NTRST_CN; it's optional and may be left open by default.

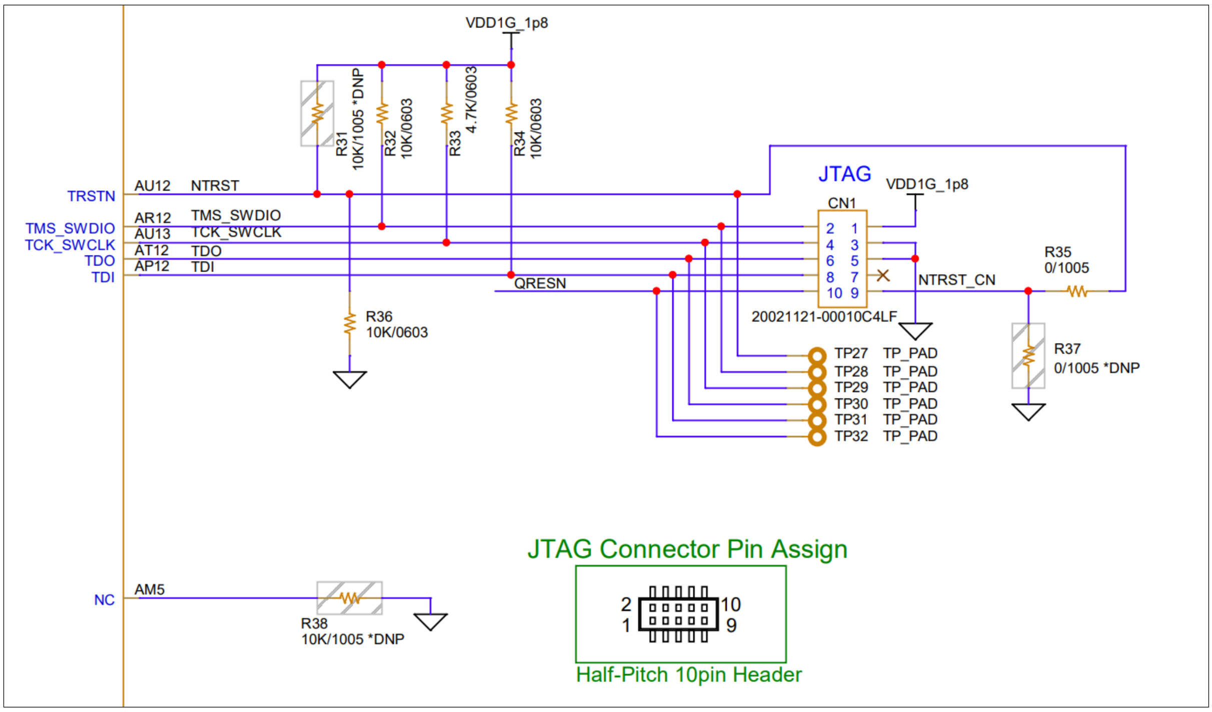


Figure 142. RZ/V2H-EVK JTAG Connector Schematic

14.4.9 Memory

14.4.9.1 xSPI Flash

The CPU board wires xSPI0 (octal/DDR-capable Serial Flash interface) to a 512-Mbit (64-MB) SPI-NOR device. The interface is powered at 1.8 V (VDD1G_1p8) and is ROM-boot supported—the SoC can fetch BL2/FIP directly from this flash.

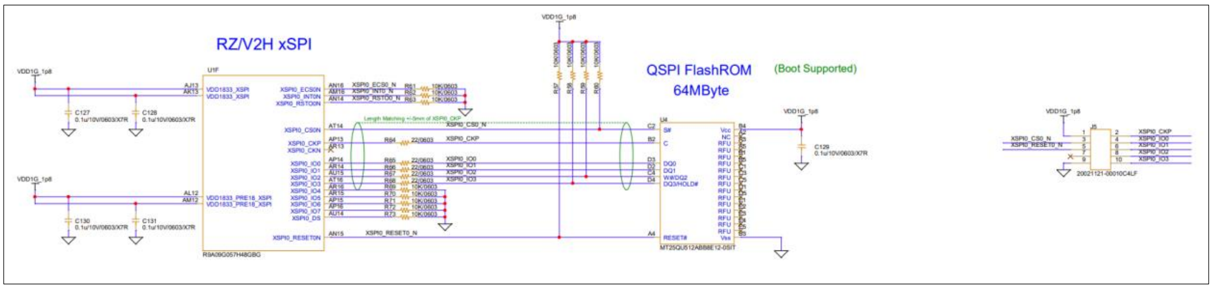


Figure 143: RZ/V2L xSPI Flash

14.4.9.2 LPDDR4

This block is the external memory subsystem for LPDDR4/LPDDR4X devices. It runs at 3200 Mb/s (1600 MHz) and uses a 32-bit data bus (two x 16-bit channels). The subsystem is split into a DDR Controller (MC) and a DDR PHY. Initialization sequences and timing tables are provided in the Linux package for this device.

Supported standards

- LPDDR4 — JEDEC JESD209-4D
- LPDDR4X — JEDEC JESD209-4-1A

DRAM interface

- Data rate: 3200 Mb/s (LPDDR4/LPDDR4X)
- Bus width: 32-bit (2 × 16-bit channels)
- Ranks: 1 or 2
- Capacity: up to 8 GB (byte-mode not supported)

Controller (MC) highlights

- Fully pipelined command/read/write paths
- Bank look-ahead to maximize throughput
- Programmable registers for timings/protocol (include auto-precharge)
- Full DRAM bring-up on controller reset
- Weighted round-robin arbitration across ports
- ECC: single-bit correction; single/double-bit error detect/report; selectable ECC storage (not on #AC0/#BC0)
- Built-in self-test (BIST) for external DRAM

Low-power features

- Power-down, self-refresh, and I/O retention modes
- Automatic entry/exit or software-controlled operation

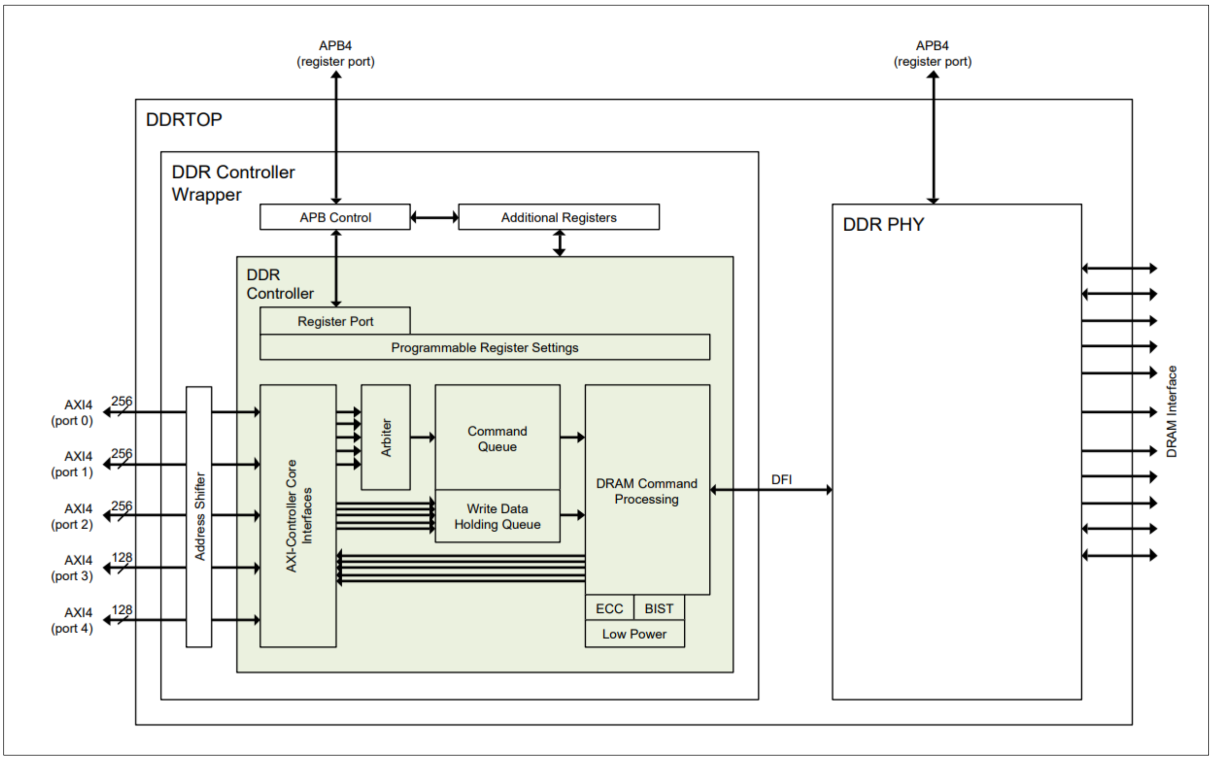


Figure 144: RZ/V2H-EVK Block Diagram of LPDDR4 (Source *RZ/V2H Group User's Manual: Hardware (Non-Agreement)*, p.278)

14.4.9.3 eMMC Flash

Support for on-board eMMC depends on the RZ/V2H-EVK hardware revision.

Board variants

- Version 1 – No eMMC fitted. The CPU board provides two microSD slots (SD1 on SDHI0, boot-capable; SD2 on SDHI1 for data).
- Version 2 – One eMMC device is populated (wired to the MMC/SDHI host), plus one microSD slot for removable media.

When eMMC is present, it may be used as the primary boot/storage device; otherwise, boot from SD1 (SDHI0).

15. Adding a New Board (Porting Guide)

This section describes how to add a new board to the RZ Common System. It covers naming, device trees, Flash-writer, TF-A, U-Boot, Linux Kernel settings, Yocto/Ubuntu integration.

For boards in the same RZ series (RZG2L / RZV2L / RZV2H), changes are typically device tree, platform-specific features, and packaging. Adding new MPUs to the RZ Common System often requires driver or platform code updates.

The overall process includes:

- Defining board metadata (JSON and board-info binary).
- Porting Flash Writer for storage and DDR initialization.
- Supplying TF-A (BL2/BL31) with a board configuration via FCONF DT.
- Creating a U-Boot device tree and minimal board hooks.
- Creating a Linux device tree (.dts) and enabling required drivers.
- Updating flash scripts and build recipes to package the new board.

Each subsection below describes these steps in detail.

15.1 Board Identification JSON Specification

This file defines metadata for each supported Renesas RZ-based board. It provides a unified description of model IDs, revision numbers, boot device locations, and boot parameters, enabling the firmware pipeline to identify and load the correct binaries (BL2, U-Boot, Kernel, etc.).

This board information will be flashed into persistent storage such as QSPI, eMMC, or SD Card. At boot time, Trusted Firmware-A (ATF) and U-Boot read this board-info binary to detect the board, select the correct device tree, and use the correct parameters for BL2, U-Boot, and the Linux Kernel.

15.1.1 JSON Field Description

Table 26: JSON specification table

Field		Description	Example	Usage
model_id		Unique numeric identifier for the board. Used by TF-A and U-Boot to select the right configuration.	0x10 = RZ/G2L-EVK 0x11 = RZ/G2L-SBC 0x20 = RZ/V2L-EVK 0x30 = RZ/V2H-EVK	Used
Revision number	Major	Major PCB revision number. Increment when the board design changes significantly.	Used	Used
	Minor	Minor PCB revision number. Increment for small changes or fixes.	Used	Used
model_string		Human-readable string for the board name. Must uniquely match model_id.	"rzg2l-evk"	Used
mfg_name		Manufacturer name.	"Renesas"	Reserved/ Not Used
bl2_loc		Storage class (device type) where BL2 is located	0 = QSPI 1 = MMC	Reserved/ Not Used
bl2_dtb_loc		Storage class (device type) where BL2 DTB is located. It must be the same as bl2_loc	0 = QSPI 1 = MMC	Reserved/ Not Used
u_boot_loc		Storage class where U-Boot is located.	0 = QSPI 1 = MMC	Reserved/ Not Used
u_boot_dtb_loc		Storage class where U-Boot DTB is located. It must be the same as u_boot_loc	0 = QSPI 1 = MMC	Reserved/ Not Used
kernel_loc		Storage class where the Linux kernel is located.	0 = QSPI 1 = MMC	Reserved/ Not Used

kernel_dtb_loc	Storage class where the Linux DTB is located. It must be the same as kernel_loc	0 = QSPI 1 = MMC	Reserved/ Not Used
bl2_id	Device number within the selected storage class for BL2 and BL2 DTB.	If bl2_loc=1 (MMC) and bl2_id=0, BL2 is on mmc0.	Reserved/ Not Used
bl2_dtb_id	Device number within the selected storage class for BL2 DTB.	If bl2_loc=1 (MMC) and bl2_dtb_id=0, BL2 DTB is on mmc0.	Reserved/ Not Used
u_boot_id	Device number for U-Boot and its DTB.	0 (mmc0)	Reserved/ Not Used
u_boot_dtb_id	Device number for U-Boot DTB.	0 (mmc0)	Reserved/ Not Used
kernel_id	Device number for Linux kernel.	0 (mmc0)	Reserved/ Not Used
kernel_dtb_id	Device number for Linux DTB.	0 (mmc0)	Reserved/ Not Used
bl2_desc	Extra parameters for BL2 (offsets, load addresses). Usually, empty.	[]	Reserved/ Not Used
bl2_dtb_desc	Extra parameters for BL2 DTB (offsets, load addresses). Usually, empty.	[]	Reserved/ Not Used
u_boot_desc	U-Boot-specific parameters. Indexed values: [0] mmcdev [1] mmcpart [2] rootfs_mmcdev [3] rootfs_mmcpart [4] image load address [5] environment load address	["0","1","0","2","0x48080000","0x58000000"]	Used
u_boot_dtb_desc	U-Boot DTB and DTBO load addresses. First entry = DTB, optional second entry = DTBO. [0] u-boot DTB address [1] u-boot DTBO address	["0x48000000","0x48010000"]	Used
kernel_desc	Reserved for Linux kernel extra parameters. Usually left empty.	[]	Reserved/ Not Used
kernel_dtb_desc	Reserved for Linux DTB extra parameters. Usually left empty.	[]	Reserved/ Not Used

Note: Reserved/Not Used fields exist for forward compatibility with legacy layouts and future boot paths. Leave them at defaults or empty [] unless a future release marks them Used.

Example JSON for a new board

```
    "rzg2l-sbc": {
        "model_id": 17,
        "revision_minor": 0,
        "revision_major": 1,
        "model_string": "rzg2l-sbc",
        "mfg_name": "Renesas",

        "bl2_loc": 0,
        "bl2_dtb_loc": 0,
        "u_boot_loc": 3,
        "u_boot_dtb_loc": 3,
        "kernel_loc": 1,
        "kernel_dtb_loc": 1,

        "bl2_id": 2,
        "bl2_dtb_id": 3,
        "u_boot_id": 6,
        "u_boot_dtb_id": 7,
        "kernel_id": 14,
        "kernel_dtb_id": 15,

        "bl2_desc": [],
        "bl2_dtb_desc": [],
        "u_boot_desc": ["0", "1", "0", "2", "0x48080000", "0x58000000"],
        "u_boot_dtb_desc": ["0x48000000", "0x48010000"],
        "kernel_desc": [],
        "kernel_dtb_desc": []
    }
```

15.1.2 Build and Flash Procedure

Once the JSON configuration has been updated with a new board entry, it must be converted into a binary format and then flashed into persistent storage (QSPI, eMMC, or SD card). At boot time, Trusted Firmware-A (ATF) and U-Boot can read the board information directly from hardware and correctly identify the platform, load the right device tree, and initialize BL2, U-Boot, and Linux Kernel with the proper parameters.

The following steps describe how to generate the board-info binary, integrate it into the build system, and flash it onto the target device. The **binmake** tool is used to convert the JSON configuration file into a binary format.

1. Create the JSON
   Define a new board entry in the JSON file. See Section 15.1.1 JSON Field Descriptions for details on the required fields and examples.
2. Build **binmake** tool
   - Prerequisites
     - On Windows
       - CMake: To install, please refer to this page CMake to download the setup file for specific platform.

    - gcc
      - Download and install MinGW-w64
      - Install to default location (**C:/MinGW**)
      - Add the following path to the Windows Environment Variables → Path: **C:/MinGW/bin**
  - On Linux

```
sudo apt install build-essential cmake
```

- Build tool
  - **On Linux**: Open a terminal and run the following commands to clone the repository

```
renesas@builder:~/$ git clone git@github.com:Renesas-SST/rz-utils.git -b \
styhead/rz-cmn
renesas@builder:~/$ cd rz-utils/tools/binmake
renesas@builder:~/rz-utils/tools/binmake$ mkdir build
renesas@builder:~/rz-utils/tools/binmake$ cd build
```

Then compile the binary

```
renesas@builder:~/rz-utils/tools/binmake/build$ cmake ..
-- The C compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/renesas/rz-utils/tools/binmake/build
renesas@builder:~/rz-utils/tools/binmake/build$ make
[ 16%] Building C object CMakeFiles/binmake.dir/src/binmake.c.o
[ 33%] Building C object CMakeFiles/binmake.dir/src/revision.c.o
[ 50%] Building C object CMakeFiles/binmake.dir/src/json_utils.c.o
[ 66%] Building C object CMakeFiles/binmake.dir/src/output_utils.c.o
[ 83%] Building C object CMakeFiles/binmake.dir/cjson/cJSON.c.o
[100%] Linking C executable binmake
[100%] Built target binmake
```

- **On Windows:** Open one of the following terminals with Administrator privileges:
  - PowerShell
  - Git Bash
  - MobaXterma

  The step is similar as Linux users. First clone the repository and then navigate to the rz-utils/tools/binmake directory. The example below uses PowerShell, but the same applies to other terminals

```
PS C:\Users\renesas> git clone git@github.com:Renesas-SST/rz-utils.git -b \
styhead/rz-cmn
PS C:\Users\renesas> cd rz-utils/tools/binmake
PS C:\Users\renesas\rz-utils\tools\binmake> mkdir build
PS C:\Users\renesas\rz-utils\tools\binmake> cd build
```

Then compile the binary

```
PS C:\Users\renesas\rz-utils\tools\binmake\build> cmake -G "MinGW Makefiles" ..
-- The C compiler identification is GNU 6.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/MinGW/bin/gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done (2.0s)
-- Generating done (0.0s)
PS C:\Users\renesas\rz-utils\tools\binmake\build> make
-- Configuring done (0.2s)
-- Generating done (0.1s)
-- Build files have been written to: C:/Users/renesas/rz-
utils/tools/binmake/build
[ 33%] Building C object CMakeFiles/binmake.dir/binmake.c.obj
[ 66%] Building C object CMakeFiles/binmake.dir/cjson/cJSON.c.obj
[100%] Linking C executable binmake.exe
[100%] Built target binmake
```

3. Basic usage

   Depending on the OS, the compiled binary in the build folder will be

   - Windows: binmake.exe
   - Linux: binmake

   Quick check after building, verify that the tool runs correctly, if it appears the usage or help, then the binary is compiled and run properly.

```
# Linux
./binmake –help

# Windows
./binmake.exe --help
```

- Single board / single revision builds

```
# Linux
./binmake --input=../platform_info.json --board=rzg2l-sbc --output=board-info.bin

# Windows (PowerShell)
./binmake.exe --input=../platform_info.json --board=rzg2l-sbc --output=board-
info.bin
```

- All revisions of a board family: If the board has multiple revisions, use all-revisions flag to generate one binary for each revision.

```
# Linux
./binmake --input=../platform_info.json --board=rzg2l-sbc --all-revisions

# Windows
./binmake.exe --input=../platform_info.json --board=rzg2l-sbc --all-revisions
```

After generating the board identification file (board-info.bin), this binary file can be passed to the universal flashing script to write it into the appropriate flash region.

Refer to Section 7.3.1. JSON Configuration for A New Board for detailed steps on how to use the universal flashing script with this file specifically how to define it in flash_images.json.

15.2 Flash Writer

The Flash Writer is a small, self-contained utility used to erase, partition, and program non-volatile storage (QSPI/OSPI via RPC, eMMC/SD via SDHI) before TF-A, U-Boot, and Linux are in place. It operates without a board DTB; instead, it relies on board-specific C definitions (pins, clocks, voltages, storage layout). Only the minimum set of blocks required to communicate on the console and access one storage path must be enabled.

15.2.1 What must be ported for a new board

1. New board on an already-supported SoC
   - Serial console (SCIF)
     - Usually reused from the existing SoC support.
     - Only change if the board uses a different SCIF instance or pins.
   - Storage path (RPC for QSPI/OSPI, or SDHI for eMMC/SD)
     - If the board uses a new flash device (such as a new SPI-NOR part or a different eMMC vendor), a device profile must be added if one does not already exist.
   - DDR initialization
     - Required in most cases, since each board usually uses a different DDR device and layout. Port the DDR parameter tables and training code for the specific DRAM on the board.
     - For very small images, the writer can run in SRAM-only mode, in which case DDR initialization can be skipped.

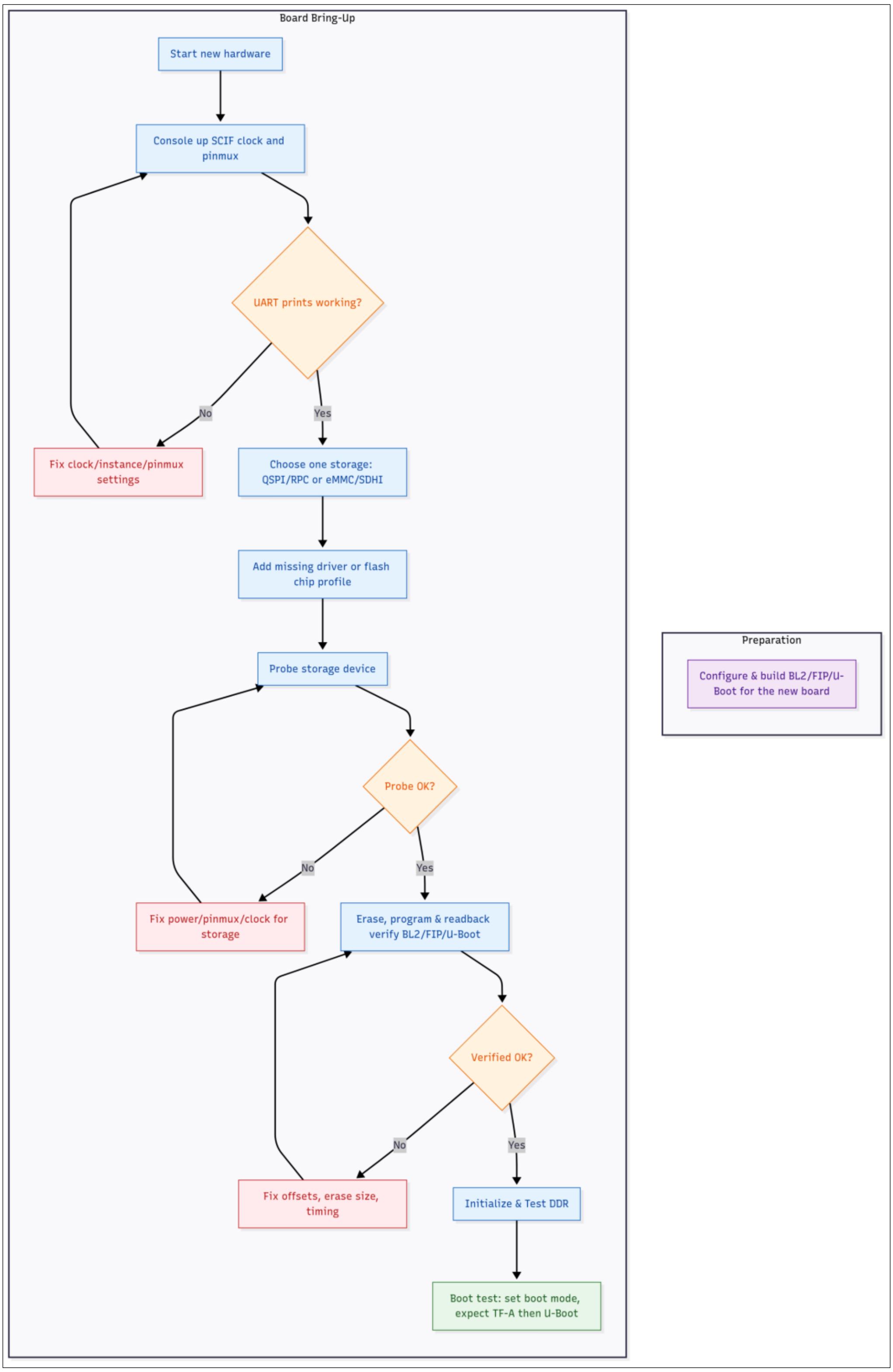

Board Bring-Up
Start new hardware
Console up SCIF clock and pinmux
UART prints working?
No
Yes
Fix clock/instance/pinmux settings
Choose one storage: QSPI/RPC or eMMC/SDHI
Add missing driver or flash chip profile
Probe storage device
Probe OK?
No
Yes
Fix power/pinmux/clock for storage
Erase, program & readback verify BL2/FIP/U-Boot
Verified OK?
No
Yes
Fix offsets, erase size, timing
Initialize & Test DDR
Boot test: set boot mode, expect TF-A then U-Boot
Preparation
Configure & build BL2/FIP/U-Boot for the new board

Figure 145. Flash-writer porting sequence

2. First board on a new SoC (SoC not yet supported):

   This is the more comprehensive porting process and should be performed once per SoC. Individual boards can then reuse the resulting implementation. All items from case (1) plus SoC-level bring-up

- Base addresses & module IDs
  - Define SCIF, RPC/OSPI, SDHI, GPIO, reset, and clock registers for this SoC.
- Clock tree & resets
  - Implement CPG/Module Stop control for SCIF and storage blocks.
  - Configure dividers/sources required for console and storage.
- Low-level init
  - Provide delay/timer sources, watchdog handling, and cache/MMU policy.
  - Add handling for SoC-specific quirks (e.g., RPC PHY toggles, SDHI tuning/voltage switching).
- DDR init framework
  - First set of DDR parameter tables for this SoC family.

15.2.2 DDR bring-up for the Flash Writer

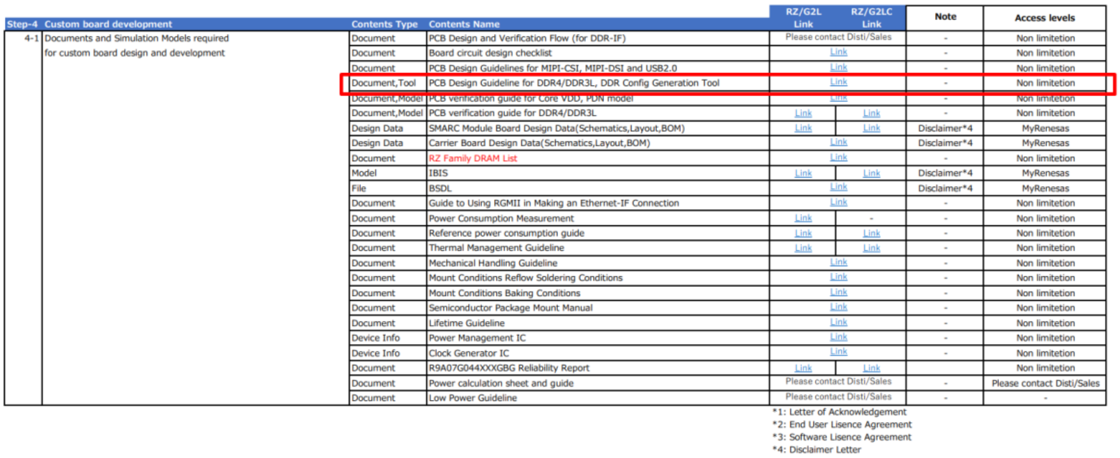
Flash Writer includes a DDR initialization library (e.g., drivers/ddr/common/ddr.c) that pulls in pre-generated parameter files. When external DRAM is required, add board-specific DDR parameter files and select them via preprocessor switches.

Unless the DDR connection and topology are identical to those on a Renesas reference board, parameters must be adapted. The DDR configuration generation tool, provided by Renesas as an Excel workbook, is used for this purpose.

> Note: On some boards (e.g., RZ/V2H-EVK), Flash Writer operates entirely in internal SRAM. In these cases, DDR initialization is not required, and no DDR parameter files are needed. This section can be skipped when porting for such boards.

15.2.2.1 How to get DDR tools and Documentation for Flash Writer DDR

1. Go to renesas.com and navigate to the product page for the target board.
2. Look for the Easy Download Guide on the product or evaluation board page.
3. Inside the Easy Download Guide look specifically for documents such as: DDR Configuration Generation Tool (usually an Excel workbook)



Step-4	Custom board development	Contents Type	Contents Name	RZ/G2L Link	RZ/G2LC Link	Note	Access levels
4-1	Documents and Simulation Models required for custom board design and development	Document	PCB Design and Verification Flow (for DDR-IF)	Please contact Disti/Sales		-	Non limitetion
		Document	Board circuit design checklist	Link		-	Non limitetion
		Document	PCB Design Guidelines for MIPI-CSI, MIPI-DSI and USB2.0	Link		-	Non limitetion
		Document,Tool	PCB Design Guideline for DDR4/DDR3L, DDR Config Generation Tool	Link		-	Non limitetion
		Document,Model	PCB verification guide for Core VDD, PDN model	Link		-	Non limitetion
		Document,Model	PCB verification guide for DDR4/DDR3L	Link	Link	-	Non limitetion
		Design Data	SMARC Module Board Design Data(Schematics,Layout,BOM)	Link	Link	Disclaimer*4	MyRenesas
		Design Data	Carrier Board Design Data(Schematics,Layout,BOM)	Link		Disclaimer*4	MyRenesas
		Document	RZ Family DRAM List	Link		-	Non limitetion
		Model	IBIS	Link	Link	Disclaimer*4	MyRenesas
		File	BSDL	Link		Disclaimer*4	MyRenesas
		Document	Guide to Using RGMII in Making an Ethernet-IF Connection	Link		-	Non limitetion
		Document	Power Consumption Measurement	Link	-	-	Non limitetion
		Document	Reference power consumption guide	Link	Link	-	Non limitetion
		Document	Thermal Management Guideline	Link	Link	-	Non limitetion
		Document	Mechanical Handling Guideline	Link		-	Non limitetion
		Document	Mount Conditions Reflow Soldering Conditions	Link		-	Non limitetion
		Document	Mount Conditions Baking Conditions	Link		-	Non limitetion
		Document	Semiconductor Package Mount Manual	Link		-	Non limitetion
		Document	Lifetime Guideline	Link		-	Non limitetion
		Device Info	Power Management IC	Link		-	Non limitetion
		Device Info	Clock Generator IC	Link		-	Non limitetion
		Document	R9A07G044XXXGBG Reliability Report	Link	Link		Non limitetion
		Document	Power calculation sheet and guide	Please contact Disti/Sales		-	Please contact Disti/Sales
		Document	Low Power Guideline	Please contact Disti/Sales		-	-

*1: Letter of Acknowledgement
*2: End User Lisence Agreement
*3: Software Lisence Agreement
*4: Disclaimer Letter

Figure 146. Example DDR generation tool of RZ/G2L-EVK

Download the ZIP file and open the Excel workbook to generate DDR parameter files (e.g., param_mc.c, param_swizzle.c) using the Renesas-style generator. Follow the next section for details.

15.2.2.2 DDR parameter generation with the Excel Workbook

The workbook generates C source files that encode DDR initialization parameters for a specific hardware design. These files are compiled into Flash Writer to apply the exact electrical and mapping settings expected by the board.

(1) Prerequisites

- Schematic and PCB layout finalized (or at least stable DDR routing and terminations).
- Known DDR device configuration (bus width, density, rank count, timing grade).
- Results or guidance from signal-integrity/layout analysis for drive strength and impedance.
- A target SoC family path in Flash Writer (ddr/g2l, ddr/v2l, or ddr/v2h).

(2) Configuration Model (how selections work)

There are two mutually exclusive ways to define the configuration:

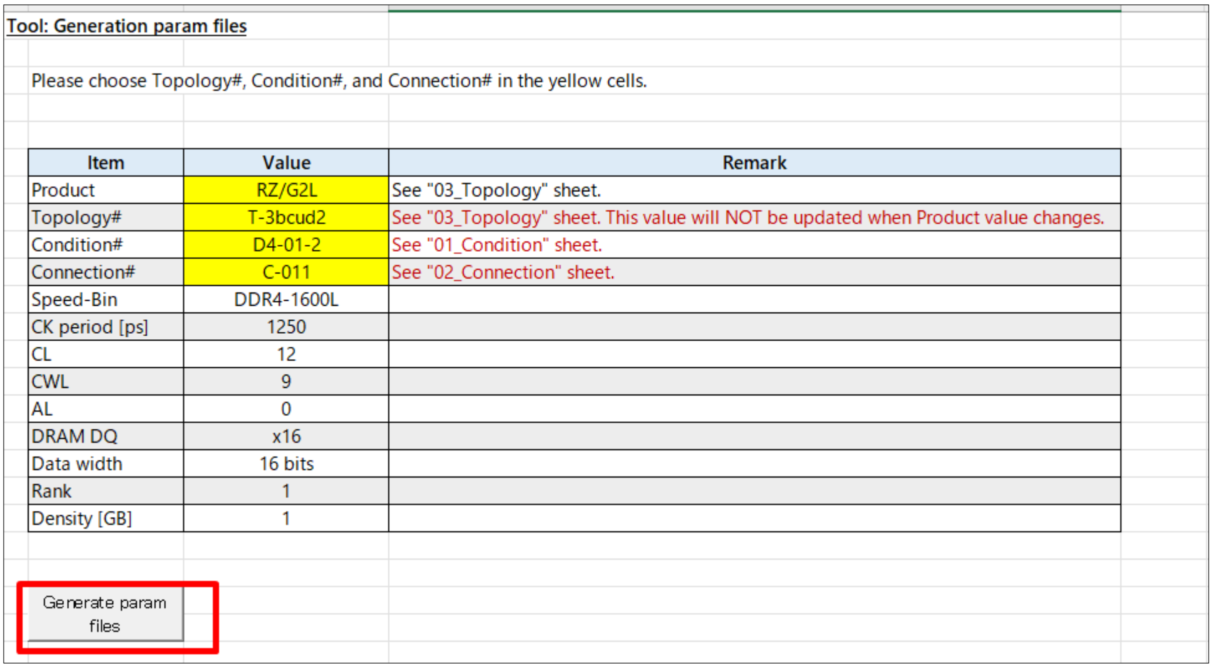
4. Condition/Connection pairing
   - Condition: Describes operating constraints and memory component characteristics.
   - Connection: Describes board-level wiring (lane usage, byte lanes, swap patterns).
   - Use this path when a published Condition/Connection pair exactly matches the hardware.
5. Topology
   - A consolidated preset that captures the wiring pattern when a matching Condition/Connection pair is unavailable.
   - Use this path when the design deviates from predefined Condition/Connection entries.

Note: choose either a Condition/Connection pair or a Topology, not both.

(3) Tab-by-Tab Guide

1. guide (usage notes) tab
   - Provides general workbook instructions.
   - Code-change examples on this tab target Trusted Firmware-A (TF-A). Integration steps for Flash Writer differ; follow the Flash Writer placement guidance in this document when adding the generated files.
2. 01_Condition (reference) tab
   - Lookup sheet for valid Condition# values.
   - No edits are expected on this tab.
3. 02_Connection (reference) tab
   - Lookup sheet for valid Connection# values.
   - No edits are expected on this tab.
4. 03_Topology (reference) tab
   - Lookup sheet for valid Topology# values.
   - No edits are expected on this tab.
5. 04_Analog (board-specific electricals) tab
   - Editable fields for drive strength, impedance, ODT, and related analog parameters.
   - Values should mirror what the layout/SI analysis determined for the design so that DDR init writes consistent register settings.
   - Modify only the column corresponding to the selected entry in GenParam; other columns are ignored during generation.
6. 05_CA_Remap (command/address mapping) tab
   - Adjust when command/address pin mapping differs from Renesas EVK examples
   - Update only the column matching the active Condition/Connection or Topology selection in GenParam.
   - Keep mappings for pins that are unused on the board (e.g., smaller DDR device). DDR pins are dedicated; blank cells can trigger validation errors in the workbook.

7. GenParam (control panel) tab
   - All selections are made here via dropdowns in highlighted cells.
   - For Condition/Connection flow: set Topology = “Other”, then pick Condition# and Connection# entries.
   - For Topology flow: select a Topology#; the Condition/Connection indicators on the page update automatically for reference.
   - Press Generate param files to export the C sources to the workbook’s folder once all the configuration is complete



Tool: Generation param files

Please choose Topology#, Condition#, and Connection# in the yellow cells.

Item	Value	Remark
Product	RZ/G2L	See "03_Topology" sheet.
Topology#	T-3bcud2	See "03_Topology" sheet. This value will NOT be updated when Product value changes.
Condition#	D4-01-2	See "01_Condition" sheet.
Connection#	C-011	See "02_Connection" sheet.
Speed-Bin	DDR4-1600L	
CK period [ps]	1250	
CL	12	
CWL	9	
AL	0	
DRAM DQ	x16	
Data width	16 bits	
Rank	1	
Density [GB]	1	

Figure 147. DDR GenParam sheet

15.2.2.3 Collect output and integrate into Flash-writer sources

When **Generate param files** button is pressed, two C files are produced in the workbook’s directory:

- param_mc.c – memory controller configuration (timings, drive/ODT, controller registers).
- param_swizzle.c – lane/bit mapping and any address/command swizzles.

(1) Recommended naming (follow guide tab)

Use identifiers from GenParam to make variants explicit:

- param_mc.c → param_mc_{Connection#}_{Condition#}.c
  Example: param_mc_C-011_D4-01-2.c
- param_swizzle.c → param_swizzle_{Topology#}.c
  Examples: param_swizzle_T3bcud.c, param_swizzle_other.c

Place the generated files in the Flash-Writer source tree; reuse an existing equivalent file if it matches the required parameters, otherwise add a new, distinctly named file (avoid overwriting unless intentionally replacing it for all boards).

- param_mc_*.c → flash-writer/drivers/ddr/g2l/ (or ddr/v2l or ddr/v2h, matching the SoC family)
- param_swizzle_*.c → flash-writer/drivers/ddr/common/

(2) Add a new board to the Flash-Writer Makefile

Open the Flash-Writer makefile and add a board stanza that sets the device family, flash type, DDR type/size, and swizzle/topology label.

Example inputs

- Board name: RZV2L_EVK_A1
- DEVICE (SoC family): RZV2L
- FLASH_TYPE: QSPI
- DDR_TYPE: DDR4
- DDR_SIZE: 2GB
- SWIZZLE: T3BCUD

If an identical board stanza already exists in the makefile, reuse it; otherwise, add a new stanza with the correct values for the hardware.

Open makefile

```
renesas@builder-pc ~/flash-writer$ vim makefile
```

makefile stanza (example)

```
# Default
ifeq ("$(BOARD)", "")
BOARD = RZG2L_SMARC
endif

# --------------------------------------
# NEW BOARD (example): RZV2L_EVK_A1
# --------------------------------------
ifeq ("$(BOARD)", "RZV2L_EVK_A1")
FILENAME_ADD = _RZV2L_EVK_A1
DEVICE      = RZV2L         # SoC family → controls param_mc path (ddr/v2l)
FLASH_TYPE  = QSPI          # or EMMC, as used by the board
DDR_TYPE    = DDR4          # memory type
DDR_SIZE    = 2GB           # 512MB | 1GB | 2GB | 4GB | …
SWIZZLE     = T3BCUD        # must correspond to an available param_swizzle_*.c
Endif

# Map to ddr.c macros
ifeq ($(DDR_TYPE),DDR4)
CFLAGS += -DDDR4=1
endif

# DDR size
ifeq ($(DDR_SIZE),2GB)
CFLAGS += -DDDR_SIZE_2GB=1
endif

# Swizzle/topology
ifeq ($(SWIZZLE),T3BCUD)
CFLAGS += -DSWIZZLE_T3BCUD=1
endif
```

(3) Map Makefile variables to the C preprocessor macros used by ddr.c

Ensure the build defines exactly one DDR size macro and one swizzle macro that match cases in ***drivers/common/ddr.c.***

Before adding new entries, check whether the required size and swizzle/topology are already supported:

- If an existing definition and file already exist
  - ○ Confirm that ddr.c contains a matching macro (e.g., DDR_SIZE_2GB, SWIZZLE_T3BCUD) and the corresponding #include "param_mc_….c" / #include "param_swizzle_….c".
  - ○ In the makefile, enable exactly one matching size macro and one swizzle macro (via CFLAGS or variable-to-macro mapping).
  - ○ Generated files may be omitted if the existing shipped files match the design.
  - ○ No changes to ddr.c are required in this case.
- If no matching definition/file exists

- Extend ddr.c by adding one #elif in the size selector and one #elif in the swizzle selector to include the new filenames.

```
/* Size selector (add near other DDR_SIZE_* cases) */
#if (DDR4 == 1)
#if (DDR_SIZE_2GB == 1)
#include "param_mc_C-010_D4-01-2.c" // controller params for 2GB
#endif
#if (SWIZZLE_T3BCUD == 1)
#include "param_swizzle_T3bcud.c" // swizzle/topology T3BCUD
#endif
#endif
```

Perform a clean rebuild of Flash Writer for a new example board named RZV2L_EVK_A1 to ensure the new parameters are compiled in.

```
renesas@builder-pc ~/flash-writer$ make BOARD=RZV2L_EVK_A1 -j"$(nproc)"
```

Before running the build command, ensure the cross-compile environment is set up. Please follow the

Section. 12.1 How to extract the eSDK

15.2.3 Add a new SPI Flash

This subsection outlines the minimal changes required to register an additional SPI/QSPI device.

15.2.3.1 Header updates

Confirm that the manufacturer identifier for the target flash vendor is present in the file **(include/dg_modul4.h)**. Typical entries include:

```
#define CYPRESS_MANUFACTURER_ID         0x01    /* Cypress      */
#define WINBOND_MANUFACTURER_ID         0xEF    /* Winbond      */
#define MACRONIX_MANUFACTURER_ID        0xC2    /* Macronix     */
#define MICRON_MANUFACTURER_ID          0x20    /* Micron       */
#define DIALOG_MANUFACTURER_ID          0x1F    /* Dialog       */
```

If the vendor is not listed, add:

- A manufacturer ID macro: *_MANUFACTURER_ID
- A device ID macro: DEVICE_ID_*
- A sector-erase size macro (SA) if a new granularity is required

```
/* Vendor and device identifiers */
#define ISS_MANUFACTURER_ID 0x9D         /* Example vendor ISS */
#define DEVICE_ID_IS25WP512 0x701A        /* Example Device ID */

/* Sector erase size (SA) definitions */
#define SA_4KB 0x01000 /* 4 KiB = 4096 bytes */
```

15.2.3.2 Register the device (table-driven flow)

Once the manufacturer and device IDs are defined (see Section 15.2.3.1. Header updates), the device must be registered in the Flash Writer. Registration is done by updating the vendor's device table in

dg_modul4.c. If the vendor is not yet supported, create a new device table and add an entry to supportedManufacturers[].

Check if the vendor already exists

Look in the SPI flash driver sources for an existing device table such as winbondDevices, macronixDevices, micronDevices, or issDevices. The currently supported vendors are:

- Cypress
- Winbond
- Macronix
- Micron
- Dialog
- ISS (Integrated Silicon Solution, Inc.)

(1) Add a Device to an Existing Vendor

If the vendor is already listed above, extend the corresponding device table by adding one line for the new device. Example (adding a new ISS device):

```
// Example: extend ISS table with an additional part
const FlashDeviceConfig issDevices[] __attribute__((section(".rodata"))) = {
    { DEVICE_ID_IS25WP256, "IS25WP256", SA_4KB,  TOTAL_SIZE_32MB },
    { DEVICE_ID_IS25WP512, "IS25WP512", SA_4KB,  TOTAL_SIZE_64MB }  // ← new device
};
```

- When adding a device, choose values according to the datasheet:
  - Sector erase size: one of SA_* macros, per datasheet.
  - Total size: one of TOTAL_SIZE_* macros, per datasheet.
  - Name: short, human-readable part name for logs (e.g., "IS25WP512").

(2) Add a New Vendor

If the vendor does not exist in the list above, create a new device table and register it in supportedManufacturers[].

Example (new vendor “AcmeSemi”):

```
// New device table
const FlashDeviceConfig acmesemiDevices[] __attribute__((section(".rodata"))) = {
    { DEVICE_ID_ACME128, "ACME128", SA_64KB, TOTAL_SIZE_16MB },
    { DEVICE_ID_ACME256, "ACME256", SA_64KB, TOTAL_SIZE_32MB },
};

// Add to supportedManufacturers[]
const Manufacturer supportedManufacturers[] __attribute__((section(".rodata"))) = {
    { CYPRESS_MANUFACTURER_ID,  "Cypress",  cypressDevices,  sizeof(cypressDevices)/sizeof(cypressDevices[0]) },
    { WINBOND_MANUFACTURER_ID,  "Winbond",  winbondDevices,  sizeof(winbondDevices)/sizeof(winbondDevices[0]) },
    { MACRONIX_MANUFACTURER_ID, "Macronix", macronixDevices, sizeof(macronixDevices)/sizeof(macronixDevices[0]) },
    { MICRON_MANUFACTURER_ID,   "Micron",   micronDevices,   sizeof(micronDevices)/sizeof(micronDevices[0]) },
    { DIALOG_MANUFACTURER_ID,   "Dialog",   dialogDevices,   sizeof(dialogDevices)/sizeof(dialogDevices[0]) },
    { ISS_MANUFACTURER_ID,      "ISS",      issDevices,      sizeof(issDevices)/sizeof(issDevices[0]) },
    { ACMESEMI_MANUFACTURER_ID, "AcmeSemi", acmesemiDevices, sizeof(acmesemiDevices)/sizeof(acmesemiDevices[0]) } // new
vendor
};
```

After rebuilding, the Flash Writer will automatically detect and configure devices from this vendor or new device.

15.3 TF-A (BL2/BL31)

As described in Section 5.3. Arm Trusted Firmware (TF-A), TF-A provides the BL2 and BL31 stages of the boot flow. In the RZ Common System, TF-A ensures early memory, clock, and boot device initialization before handing control to U-Boot and Linux.

The RZ Common BL2/BL31 binary supports multiple SoCs (RZ/G2L, RZ/V2L, RZ/V2H) and is designed to remain board-agnostic. Board-specific details such as DDR initialization, SPI/xSPI flash parameters, and system configuration are provided at runtime through a Firmware Configuration Framework (FCONF) device tree.

When adding a new board, the task is to provide these parameters in a structured way. To simplify this process, TF-A includes the dts_creator tool, which reads a YAML board configuration, applies the appropriate SoC template, extracts DDR and SPI parameters, and generates a valid DTS for BL2.

15.3.1 Firmware Configuration Framework

FCONF (Firmware Configuration Framework) is a TF-A mechanism that allows platform settings to be passed at runtime through a structured device tree blob (DTB) rather than being hard-coded into firmware binaries. Instead of recompiling BL2 for every new board, generic binaries (BL2/BL31) remain board-agnostic, while the FCONF DT provides:

- Memory parameters (DDR type, initialization tables)
- Boot device parameters (SPI/xSPI, eMMC, SD offsets)
- Clock and pinctrl defaults
- Security and system controller options

At boot, BL2 loads the FCONF DTB and uses its properties to configure hardware before handing off to BL31 and U-Boot. This makes it possible to support many boards within the same SoC family using one common BL2 binary — only the FCONF DT changes per board.

15.3.2 Tool Location and Layout

The dts_creator tool is provided under rz-atf/tools/renesas/ and automates the generation of board device tree files for BL2. Instead of manually editing complex DTS templates, users describe their board in a YAML config, and the tool fills in the correct values.

```
rz-atf/tools/renesas/dts_creator
├── configs/              # Board configuration YAMLs
│   ├── rzg2l_evk.yaml
│   ├── rzg2l_sbc.yaml
│   ├── rzv2h_evk.yaml
│   └── rzv2l_evk.yaml
├── dts_creator.py        # Main generator script
├── schemas/              # YAML schema for validation
│   └── rz-boards.yaml
└── template/             # SoC template DTS files
    ├── rzg2l_template.dts
    ├── rzv2l_template.dts
    └── rzv2h_template.dts
```

Each part has a defined role:

- configs: describe boards
- schemas: validate generated DTS
- template: provide SoC boilerplate
- Python script merges everything into the final DTS.

15.3.3 Prerequisites

Before using dts_creator, the build environment must have supported Python runtime and required libraries as follows:

- Python 3.8 or later – required to run the script
- PyYAML – used for parsing YAML configs.
- Core requirements:
  - **On Windows**, MSYS2 is required to build and run dtc and related tools.
    - Download and install MSYS2 from MSYS2 (check Installation section and download the .exe installer)
    - Follow the setup steps to complete the installation
    - Once the installation is complete, open the MSYS2 shell and run:

      ```
      PS C:\Users\renesas> pacman -S base-devel mingw-w64-x86_64-gcc
      PS C:\Users\renesas> git clone https://git.kernel.org/pub/scm/utils/dtc/dtc.git
      PS C:\Users\renesas> cd dtc
      PS C:\Users\renesas> make dtc
      ```

      After building, dtc.exe will be in the dtc directory; you can run it from Windows CMD/PowerShell or add its location to your PATH.
    - dtschema for dt-validate. First, install Microsoft Visual C++ Build Tools
      - Download the installer from: Microsoft C++ Build Tools - Visual Studio
      - Run the installer and select the Desktop development with C++ workload.
      - After installation, verify by running:

      ```
      PS C:\Users\renesas> cl.exe
      ```

      It should print the Microsoft compiler version (14.0 or newer).

- Install Python packages. From Powershell, run:

```
PS C:\Users\renesas> pip install pyyaml dtschema
```

- **On Linux**, install system packages first. Open the terminal and run the following commands. If Python 3.12 is in use: set up a virtual environment first.

```
renesas@builder-pc:~/$ sudo apt install python3.12-venv
renesas@builder-pc:~/$ python3 -m venv .venv
renesas@builder-pc:~/$ source .venv/bin/activate
```

Then run the following commands

```
renesas@builder-pc:~$ sudo apt install device-tree-compiler swig python3-dev
renesas@builder-pc:~$ pip3 install pyyaml dtschema
```

15.3.4 Adding a New Board

When adding a new board, the starting point is a YAML configuration file under tools/renesas/dts_creator/configs/. This file describes the SoC family, board name, memory type, and boot flash device. dts_creator uses these fields to select the right DTS template and fill in hardware parameters.

3. Create a new configuration file (myboard.yaml) for your board under:

rz-atf/tools/renesas/dts_creator/configs

Replace myboard.yaml with a descriptive name for the actual board (e.g. rzg2l_sbc.yaml, rzv2h_custom.yaml). By default, the DTS generated will be written to:

```
rz-atf/fdts/<soc>-<board>.dts
```

4. Field in the required fields:

```
soc: rzg2l            # SoC family: rzg2l | rzv2l | rzv2h
board: sbc            # Board identifier: evk | sbc
# output: fdts/myboard.dts   # Optional override of the output path

spi:
  protocol: spi_multi # Boot flash protocol: spi_multi (QSPI) | xspi (Octa)
  flash_chip: IS25WP256  # Exact flash family name supported by TF-A

ddr:
  type: ddr4            # Memory type: ddr4 | lpddr4
  ddr sources:
    - plat/renesas/rz/soc/g2l/drivers/ddr/param_mc_myboard.c
    - plat/renesas/rz/common/drivers/ddr/param_swizzle_myboard.c
```

This example defines a board based on RZ/G2L, booting from SPI flash, with DDR4 memory, and DDR parameters coming from two .c files under the TF-A source tree.

YAML Fields

- soc: Defines the SoC family of the board. Supported values are:
  - rzg2l — RZ/G2L family (includes G2L, G2LC, G2UL variants)
  - rzv2l — RZ/V2L family
  - rzv2h — RZ/V2H family

- board: A simple identifier for the board type, e.g. evk, sbc, or a custom name. Used to name the generated DTS.
- output (optional): Path where the generated .dts will be written. If omitted, a default path under fdts/ is used.
- spi.protocol:
  Specifies the boot flash protocol:
  - spi_multi — Standard QSPI/quad-SPI flash
  - xspi — Octa-SPI flash (used in RZ/V2H and some RZ/V2L designs)
- spi.flash_chip: The specific flash device part number (e.g., IS25WP256). This must match the supported flash driver names in TF-A. For new parts, add a matching header with spi_multi_reg_values.h under plat/renesas/rz/common/include/drivers/spi_multi.
- ddr.type:
  Memory type used on the board:
  - ddr4 — DDR4 SDRAM
  - lpddr4 — Low-power DDR4
- ddr sources: A list of .c files containing DDR configuration tables (memory controller init, PHY init, swizzle settings). These files are parsed by dts_creator and embedded into the generated DTS.
  - Each new board with a unique memory device must provide its own DDR parameter sources under plat/renesas/rz/.../drivers/ddr/.
  - Refer to Section 15.3.2. DDR bring-up for the Flash Writer for detailed guidance on how to create new DDR parameter files for a new board.

15.3.5 Usage

After preparing a board YAML configuration, the next step is to run the dts_creator tool. The tool parses the YAML file, selects the appropriate SoC template, extracts DDR and SPI parameters from the specified sources, and generates a complete DTS for BL2 FCONF.

From the root of the Linux TF-A repository:

On Linux: Open terminal and run the following commands:

```
renesas@builder:~/$ cd rz-atf/tools/renesas/dts_creator
renesas@builder:~/rz-atf/tools/renesas/dts_creator$ python3 dts_creator.py -c \
configs/myboard.yaml
```

On Windows: Open one of the following terminals with Administrator privileges:

- PowerShell
- Git Bash
- MobaXterma

Navigate to the host/tools/bootloader_flasher directory inside the rz-cmn-srp folder. The example below uses PowerShell, but the same applies to other terminals

```
PS C:\Users\renesas\Desktop> cd ~/rz-atf\tools\renesas\dts_creator
PS C:\Users\renesas\rz-atf\tools\renesas\dts_creator$ py dts_creator.py -c \
configs\myboard.yaml
```

Optional: -o <output-path> overrides the output declared in the config.

After generating a DTS, validate it against the Renesas schema:

```
$ cd ~/rz-atf/fdts
$ dtc -I dts -O dtb -o <temp>.dtb <path-to-generated.dts>
$ dt-validate --schema tools/renesas/dts_creator/schemas/rz-boards.yaml\
<temp>.dtb
```

The DTS should pass without warnings or errors before it is committed.

Example:

```
# On Linux
renesas@builder-pc:~$ cd ~/rz-atf/fdts
renesas@builder-pc:~/rz-atf/fdts$ dtc -I dts -O dtb -o myboard.dtb myboard.dts
renesas@builder-pc:~/rz-atf/fdts$ dt-validate \
--schema ../tools/renesas/dts_creator/schemas/rz-boards.yaml myboard.dtb


# On Windows
PS C:\Users\renesas> cd ~/rz-atf/fdts
PS C:\Users\renesas\rz-atf/fdts> dtc -I dts -O dtb -o myboard.dtb myboard.dts
PS C:\Users\renesas\rz-atf\fdts> dt-validate \
--schema ../tools/renesas/dts_creator/schemas/rz-boards.yaml myboard.dtb
```

After generating the device tree, it can be passed to the universal flashing script, which merges it into the common BL2 at flash time. Refer to Section 7.3.1. JSON Configuration for A New Board for detaied steps on how to use the universal flashing script with this file specifically how to define it in flash_images.json (atf_dtb) field.

15.4 U-Boot (BL33)

This release ships a SoC-common U-Boot binary without a DTB (u-boot-nodtb.bin). Adding a new board usually requires only minor modifications to the U-Boot source code, mainly in the early initialization phase (e.g., board_early_init_f(), misc_init_r(), or ft_board_setup()) to handle board-specific logic such as power sequencing or clock pinmuxing.

U-Boot separates responsibilities into three scopes, aligned with initialization phases:

- SoC (early init): minimal, silicon-wide steps to reach a console and parse the Device Tree.
- BSP / drivers (late init): device behavior configured by the Device Tree.
- Board (late init): wiring- and policy-specific choices read from the Device Tree.

Because most configurations are expressed in the Device Tree, the U-Boot binary remains portable across boards, and only minimal source changes are needed for unique initialization requirements.

15.4.1 Initialization model

15.4.1.1 SoC-specific - early init (s_init)

Purpose: stabilize silicon so that U-Boot can run, print, and parse DT.

Typical content

- Minimal clocks/resets and bus enables.

- Essential UART pinmux (only if required for early console).

- Watchdog safety (SoC-level).

- Mandatory SoC errata/workarounds that must precede any driver's access.

15.4.1.2 BSP-specific (drivers)

Purpose: reusable device support for multiple boards.

Typical content

- SPI-NOR/PMIC/PHY drivers and generic quirks.
- DT-configurable features shared across platforms.

15.4.1.3 Board-specific — late init (board_late_init / last_stage_init)

Apply behavior that depends on wiring or fitted options and can wait until devices are probed.

Typical content

- Read board EEPROM (e.g., MAC/IDs) and set env.
- PMIC/I²C sequences defined by DT.
- Enable USB/Ethernet once pinctrl/rails.
- Optional PHY tuning, LEDs, diagnostics (often in last_stage_init).

15.4.2 How soc_id and board_id are used

TF-A provides two identifiers that U-Boot consumes:

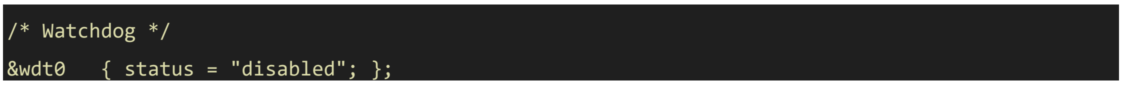
- extern u64 soc_id; identifies RZ/G2L, RZ/V2L, or RZ/V2H.
  - RZ_SOC_RZG2L: RZ/G2L SoC
  - RZ_SOC_RZV2L: RZ/V2L SoC
  - RZ_SOC_RZV2H: RZ/V2H SoC
- extern u64 board_id; identifies the specific board (EVK, SBC, custom, etc.). This value is organates from the Board Identification data corresponds to the model_id field defined in the JSON file. See Section 15.1. Board Identification JSON Specification

board/renesas/rz-cmn/rz-cmn.c branches on these IDs to select the minimal per-board actions.

15.4.3 Steps to add a board

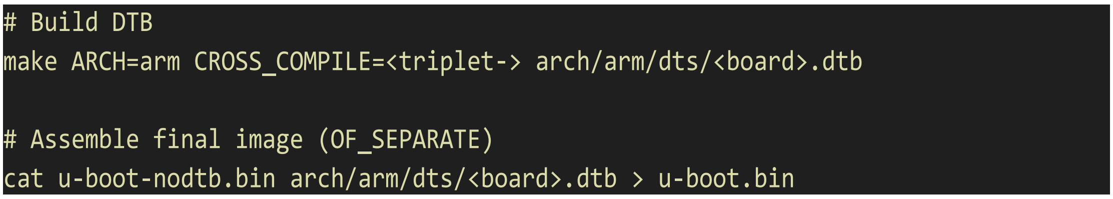
1. Create a board device tree for the board
   - Add arch/arm/dts/<board>.dts
   - Include the correct SoC .dtsi and any common carrier/module *-u-boot.dtsi
2. Describe the hardware
   - Enable the console UART (chosen.stdout-path).
   - Enable only peripherals that exist on the board (SDHI/eMMC, Ethernet, I²C, USB, RPC/SPI-NOR, etc.).
   - Provide pinctrl, clocks, and regulators as needed.
3. Example: Minimal board DT template

```
// SPDX-License-Identifier: GPL-2.0
/dts-v1/;

/* TODO: pick the right SoC include:
 *   RZ/G2L: r9a07g044l.dtsi
 *   RZ/V2L: r9a07g054l.dtsi
 *   RZ/V2H: r9a09g057h.dtsi
 *   .. other SoCs
 */
#include "r9a07g044l.dtsi"
#include "r9a07g044l-u-boot.dtsi"

 / {
        model = "<board name>";
        /* Recommended: "<vendor-prefix>, <board-name>", then the SoC compatible */
        /* Example */
        compatible = "acme,foo-rzg2l", "renesas,r9a07g044l", "renesas,rzg2l";

        chosen {
                /* TODO: set to the UART wired (serial0/1/etc.) and baud */
                stdout-path = "serial0:115200n8";
        };

        /* TODO: set actual DRAM base/size for the board
         * Replace the Marco with actual DDR base and size
        /*
        memory@48000000 {
                device_type = "memory";
                reg = <0 DDR_BASE_ADDR 0 DDR_AVAILABLE_SIZE>;
        };
};

/* ---------- Enable and adjust only what is actually present on the board ---------- */

/* Console UART */
&scif0  { status = "okay"; };

/* SD/eMMC */
&sdhi0  { status = "disabled"; /* bus-width = <4>; or <8>; */ };
&sdhi1  { status = "disabled"; };

/* Ethernet (add pinctrl, phy-mode, and phy node if used) */
&eth0   { status = "disabled"; };

/* I2C (add devices if used) */
&i2c0   { status = "disabled"; };

/* RPC/SPI-NOR (add flash child if used) */
&spibsc { status = "disabled"; };

/* USB (EHCI) */
&ehci0  { status = "disabled"; };
&ehci1  { status = "disabled"; };
```



```
/* Watchdog */
&wdt0   { status = "disabled"; };
```

4. Build and assembled



```
# Build DTB
make ARCH=arm CROSS_COMPILE=<triplet-> arch/arm/dts/<board>.dtb

# Assemble final image (OF_SEPARATE)
cat u-boot-nodtb.bin arch/arm/dts/<board>.dtb > u-boot.bin
```

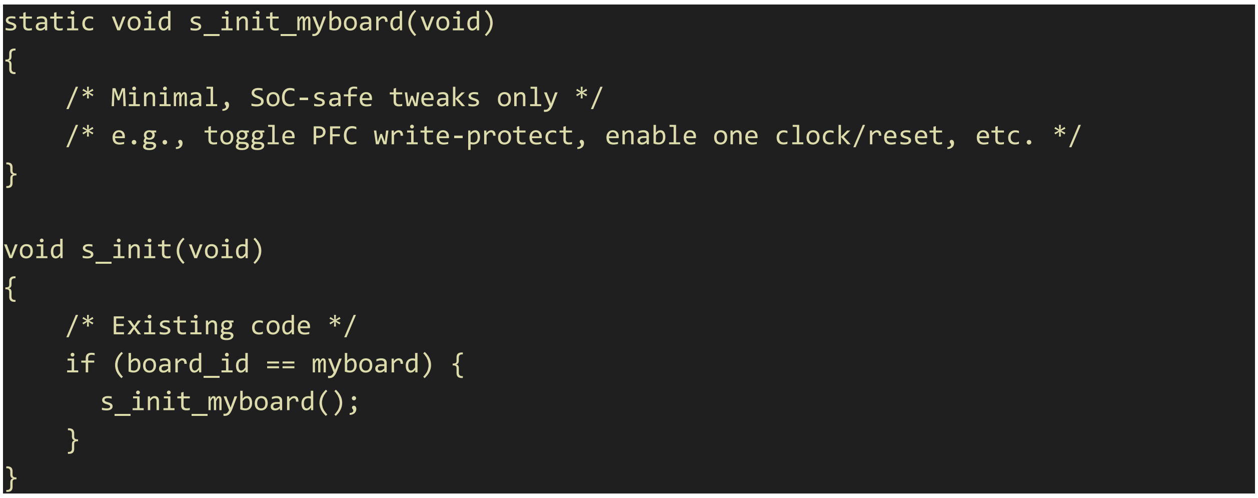
15.4.4 Board-specific code hooks

This section describes the three optional places to add minimal board code in board/renesas/rz-cmn/rz-cmn.c.

15.4.4.1 Early init – s_init()

Do the minimum register work required so U-Boot can run reliably, print to console (if needed), and parse the device tree—before any driver accesses hardware.

- A clock/reset/pinmux must be applied before DM/driver probe (e.g., gating a domain that holds bus fabric).
- SoC errata/workarounds that must precede any peripheral access.
- Early console pinmux only if necessary for debug.



```
static void s_init_myboard(void)
{
    /* Minimal, SoC-safe tweaks only */
    /* e.g., toggle PFC write-protect, enable one clock/reset, etc. */
}

void s_init(void)
{
    /* Existing code */
    if (board_id == myboard) {
      s_init_myboard();
    }
}
```

15.4.4.2 Board Init – board_init()

Perform concise per-board setup after relocation and basic subsystems are ready, but before late policies. Typical use: a PHY/USB/PMIC enables sequence that drivers can't (or shouldn't) do automatically.

```
int board_init(void)
{
    gd->bd->bi_boot_params = CONFIG_SYS_TEXT_BASE + 0x50000;

    if (board_id == BOARD_ID_MYBOARD) {
        // board_usb_init_myboard();   /* short bring-up sequence */
        // board_pmic_i2c_init();      /* minimal sanity config */
    }
    return 0;
}
```

15.4.4.3 Board_late_init() – policies & ID after drivers' probe

This is an optional stage that applies board policies and wiring-dependent actions after devices and buses exist under driver model. If the added board does not require any late policies beyond what the device tree and drivers already provide, this stage can be omitted entirely for a new board.

Common uses.

- Read board EEPROM (e.g., MAC addresses) and export to env
- Watchdog re-init
- GPIO indicators or simple diagnostics

```
int board_late_init(void)
{
    /* Existing code */
    if (board_id == BOARD_ID_MYBOARD) {
        /* Final, board-specific tweaks (e.g., PHY delay configuration). */
        /* TODO: add your writes/calls here */
    }
    return 0;
}
```

15.4.4.4 last_stage_init() – Final tweaks

last_stage_init() is an optional U-Boot board hook that runs near the end of BL33 bring-up, after driver model devices have been probed and board_late_init() has executed.

- PHY register tuning or strap fixes
- Enabling auxiliary clocks/peripherals not strictly required for boot
- Final diagnostics

```
int last_stage_init(void)
{
    /* Existing code */
    if (board_id == BOARD_ID_MYBOARD) {
        /* Example: PHY delay configuration */
    }
    return 0;
}
```

15.4.5 Adding BSP Drivers

Add or modify BSP components only when the hardware cannot be supported by existing drivers and Device Tree (DT) alone.

When a driver update/addition is required

- New or unsupported device: SPI-NOR without usable SFDP/ID, new PMIC/regulator, clock chip, Ethernet PHY, USB/bridge device.
- Mandatory behaviors that cannot be modeled in DT: required register sequences, timing workarounds, addressing modes, or errata not exposed via DT properties.
- Probe succeeds but operation fails; device binds yet needs a quirk/opcode/map fix to function reliably.

15.5 Linux kernel

Linux on the RZ Common System is device tree (DT) centric. The SoC .dtsi file describes common silicon blocks, while each board contributes a small .dts file that wires those blocks together (pins, voltages, clocks, PHYs, GPIOs, supplies). Most bring-up work involves authoring a board device tree and enabling the correct drivers.

Direct register writings are not used in Linux DT. Instead, standard bindings and driver properties should be used.

15.5.1 Steps to add a board

1. **Create a board DT**
    - Create arch/arm64/boot/dts/renesas/<board>.dts (path may vary if using an out-of-tree DTS package).
    - Include the correct SoC .dtsi (e.g., r9a07g044l.dtsi, r9a07g054l.dtsi, …) and any board-shared .dtsi you maintain.
2. **Describe the hardware**
    - **Memory:** add a memory@… node with the actual base/size; declare any reserved-memory regions needed by firmware.
    - **Chosen/console:** set /chosen { stdout-path = "serialX:115200n8"; }; to match the wired UART.
    - **Pin control:** define pin groups under &pinctrl and reference them from each peripheral via pinctrl-0.
    - **Clocks/resets:** reference SoC clock providers via clocks = <&…> (no hardcoded clock writes).
    - **Regulators:** model rails (fixed or PMIC-based) and hook them up via vmmc-supply, vqmmc-supply, vdd-supply, phy-supply, etc.
    - **Peripherals:** enable only what exists on the board; provide properties the drivers expect:
        - **MMC/SDHI:** bus-width, cd-gpios, wp-gpios, vmmc-supply, vqmmc-supply, non-removable (for eMMC), timing caps (cap-sd-highspeed, mmc-hs200-1_8v, etc.).
        - **Ethernet:** phy-mode (rgmii, rmii, …), phy-handle (or fixed-link), optional internal delay properties (rx-internal-delay-ps, tx-internal-delay-ps) depending on PHY/MAC, and MDIO bus if external PHYs are present.

- **I²C:** mark controllers okay; add on-bus devices (EEPROMs, PMICs, sensors) with their compatible and reg.
- **USB:** enable controller(s); add vbus-supply if power switching is controlled by a regulator/GPIO.
- **SPI/RPC (xSPI/NOR):** enable the bus and add the flash node with compatible = "jedec,spi-nor" (or vendor-specific); configure bus widths and modes per binding.
- **Watchdog, GPIO LEDs/keys, UARTs:** enable and wire as needed.

3. **Add to Makefile**
   - Kconfig/SoC enablement
     - Ensure the correct CONFIG_ARCH_* option is enabled for the SoC family (e.g., CONFIG_ARCH_R9A07G044, CONFIG_ARCH_R9A07G057). This ensures the kernel builds DTS files for the right SoC.
   - Device tree Makefile
     - Edit the arch/arm64/boot/dts/renesas/Makefile
     - Add new .dts so it is compiled into a .dtb
     - Example

```
# For R9A07G044 SoC
dtb-$(CONFIG_ARCH_R9A07G044) += <new-board>.dtb
```

4. **Build the DTBs**
   Run the kernel build to generate device trees

```
renesas@builder-pc:~/linux-rz$ make renesas_defconfig
renesas@builder-pc:~/linux-rz$ make ARCH=arm64 \
CROSS_COMPILE=aarch64-poky-linux- \
dtbs
```

   This way, when building the kernel, the board's DTB will be automatically generated in the arch/arm64/boot/dts/renesas/ output directory.

15.5.2 Kernel Device Tree Selection

After completing all BSP steps, the final task is to let U-Boot know which board's device tree to load. U-Boot's environment already supports user-defined hooks. To load a new device tree for a new board without rebuilding U-Boot, add a rule in uEnv.txt that sets fdtfile when the board's identifiers match.

Open the uEnv.txt placed under partition 1 (FAT32) and add the following condition

```
# Set fdtfile to "new-board.dtb" when the board model and revision match.
# This tells U-Boot which device tree to load and pass to the kernel at boot.
# Replace "model-string", "1", and "0" with your board's actual identifiers.

fdt_user_cases=if test "${model_string}" = "model-string" -a "${revision_major}"
= "1" -a "${revision_minor}" = "0"; then setenv fdtfile new-board.dtb; fi
```

How it works

- fdt_user_cases: Executed after the built-in selection table. When fdtfile remains unset, this hook can be used to assign a DTB.
- Identifiers: model_string, revision_major, and revision_minor come from the board-info blob generated from platform_info.json in the build system (see Section 15.1.1. JSON Field Description). These values must match the identifiers for the target hardware.

Note: If the condition matches the board identification, then new-board.dtb will be loaded during kernel boot. The new-board.dtb file should be placed under partition 1 (FAT32) at /dtb/renesas/new-board.dtb.

16. Appendix

16.1 Factory Firmware Flashing Using Serial Downloader (SCIF) Mode

In most cases, the RZ boards come preloaded with the latest firmware. The preferred method of updating the firmware is through the SD card flashing method, as described in section 7. Programming / Flashing .

However, there are cases where you might require the use of a serial downloader. This is more common in a factory environment where the boards are being programmed for the first time or in cases where the board is bricked.

This is considered hardware flashing because it requires the board to be put into the serial download mode (called SCIF mode), by altering the bootstrapping pins.

16.1.1 RZ/G2L-SBC

This section describes the firmware flashing process for the RZ/G2L-SBC board.

Note: The RZ/G2L-SBC does not have any interfaces on the main board to alter the boot mode. The bootstrapping pins are routed through the bottom connectors J12 & J13. Hence, the process requires the use of an adapter board, which is not included in the package.

16.1.1.1 Required Hardware

This flashing process requires the use of boot mode change, which is achieved using an adapter board which is not included in the package.

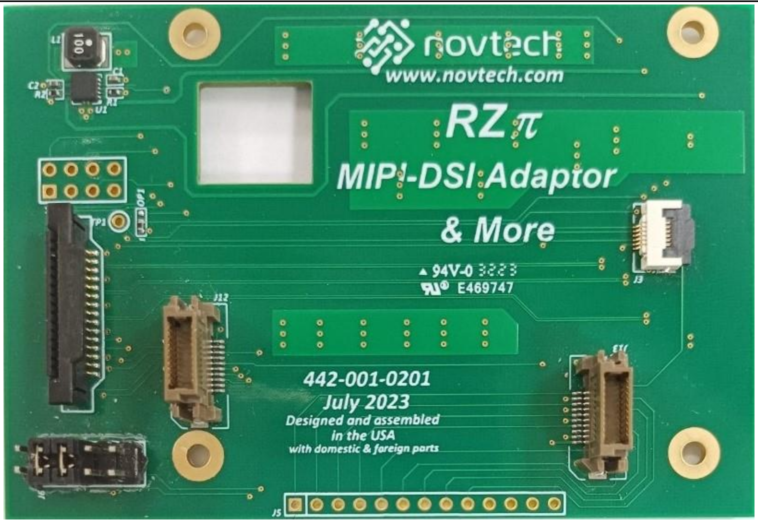


Figure 148. Adaptor board

After setting up the required hardware and configuring the boot mode through connectors J12 and J13 on the RZ/G2L-SBC, refer to Section 7.3.3. Flashing the Bootloader or using the universal script in Section 7.3.2. Usage and Flashing Operations to flash the firmware

16.1.2 RZ/G2L-EVK and RZ/V2L-EVK

This section describes the firmware flashing process and boot mode configuration for each supported board. Firmware flashing is required to write bootloaders (BL2 and FIP) to the onboard flash memory. The process uses Renesas' Flash Writer tool and requires setting the board into SCIF Download Mode.

Unlike the RZ/G2L-SBC, which requires external hardware to configure boot mode, the RZ/G2L-EVK and RZ/V2L-EVK feature onboard DIP switches that allow direct selection of boot modes. This simplifies the flashing process and eliminates the need for manual signal strapping.

16.1.2.1 DIP Switch Settings

Use the DIP switch SW11 to configure the boot mode:

Table 27: SCIF Download Mode - RZ/G2L-EVK & RZ/V2L-EVK

Switch	SCIF Download Mode
SW11-1	OFF
SW11-2	ON
SW11-3	OFF
SW11-4	ON

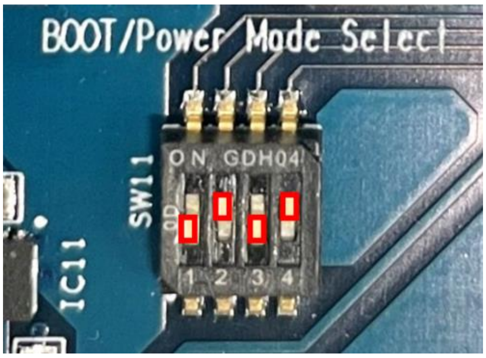


Figure 149. SW11 SCIF Download Mode - RZ/G2L-EVK & RZ/V2L-EVK

Use the DIP switch1 (SW1) to select eMMC as boot device:

Table 28: Select eMMC as boot device

Switch	Select eMMC
SW1-1	ON
SW1-2	OFF

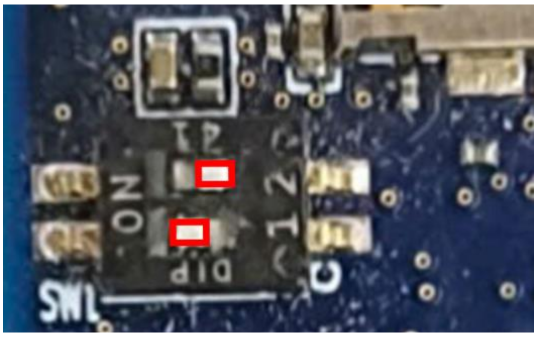


Figure 150: SW1 Settings for eMMC Boot - RZ/G2L-EVK & RZ/V2L-EVK

After setting up the required hardware and configuring the SCIF download mode for the RZ/G2L-EVK or RZ/V2L-EVK, refer to Section 7.3.3. Flashing the Bootloader or using the universal script in Section 7.3.2. Usage and Flashing Operations to flash the firmware

16.1.3 RZ/V2H-EVK

Use the DIP switch DSW1 to configure the boot mode:

Table 29: SCIF Download Mode - RZ/V2H-EVK

Switch	Status	Function
DSW1-1	ON	Select the cold boot CPU - OFF: CM33 - ON: CA55 (default)
DSW1-2	OFF	Input the CA55 frequency at the CA55 cold boot - [OFF: OFF]: 1.6GHz - [OFF: ON]: 1.7GHz (default) - [ON: OFF]: 1.1GHz - [ON: ON]: 1.5GHz
DSW1-3	ON	
DSW1-4	OFF	- [OFF: OFF]: xSPI - [OFF: ON]: SCIF - [ON: OFF]: SD (default) - [ON: ON]: eMMC
DSW1-5	ON	
DSW1-6	OFF	OFF: SSCG ON (default), ON: SSCG OFF
DSW1-6	OFF	OFF: Normal mode, ON: Debug mode
DSW1-7	OFF	Fixed to OFF

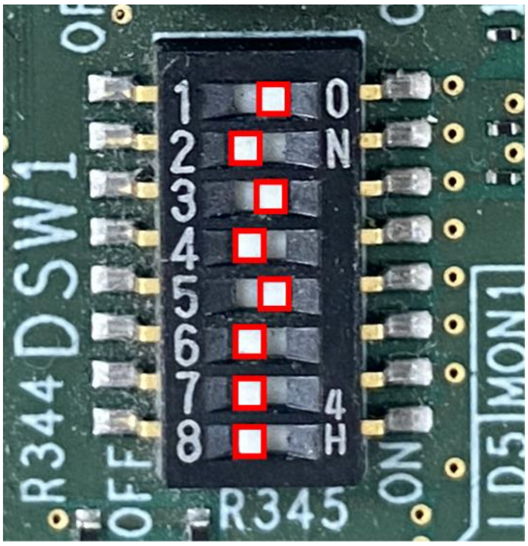


Figure 151. DSW1 - SCIF Boot Mode

To enable SCIF Download Mode, set DSW1-4 and DSW1-5 according to the SCIF configuration in the Table SCIF Download Mode

Other switches (DSW1-1, DSW1-2, DSW1-3, DSW1-6, and DSW1-7) should remain in their default positions unless you need to change CPU selection, boot frequency, SSCG, or debug mode as described in the table.

After configuring the DIP switches for SCIF Download Mode, please refer to Section 7.3.3. Flashing the Bootloader or using the universal script in Section 7.3.2. Usage and Flashing Operations to flash the firmware

16.2 Boot Mode Reference (Non-SCIF)

This section summarizes the switch/strap settings required for normal boot and boot-device selection across supported boards. Use these settings after completing factory flashing or when switching the boot device during bring-up.

16.2.1 RZ/G2L-EVK & RZ/V2L-EVK

These EVKs provide on-board DIP switches for boot mode and boot device selection.

The settings below tell the BootROM which device to read the initial firmware from, i.e., where to fetch BL2 (and Boot Parameter, if used) and subsequently the FIP.

Use these settings after factory flashing to boot from the programmed device.

Table 30: SW11 – Boot Device Selection (Normal Boot)

Boot device	SW11-1	SW11-2	SW11-3	SW11-4	Description
eMMC	ON	OFF	OFF	ON	Boot from on-board eMMC (BootROM loads BL2/BL2+BP from eMMC, then FIP).
QSPI	OFF	OFF	OFF	ON	Boot from QSPI NOR flash
SD / eSD	ON	ON	OFF	ON	Boot from SD/eSD card (slot media)

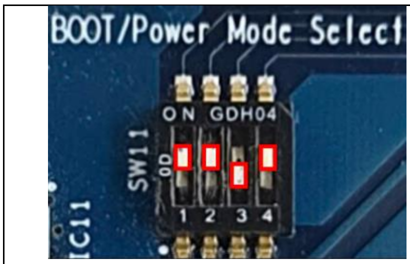


Figure 152: SW11 - eSD Boot Mode

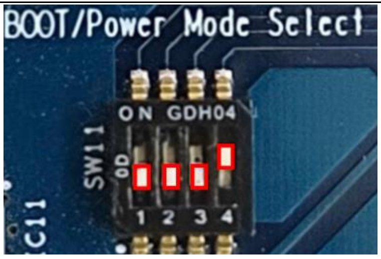


Figure 153. SW11 – QSPI Boot Mode

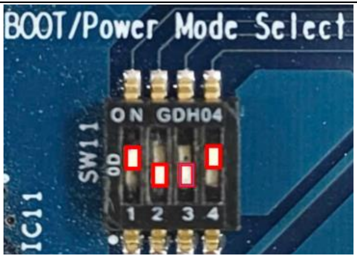


Figure 154. SW11 – eMMC Boot Mode

On the SOM module, SW1 selects eMMC or microSD boot mode. Please refer to the table below for the boot-mode options for each switch setting.

Table 31: SW1 – SOM module Switch mode

Boot device	ON	OFF
SW1-1	Normal Operation	JTAG debug mode
SW1-2	Select microSD slot on RTK9744L23C01000BE	Select eMMC on RTK9744L23C01000BE

16.2.2 RZ/V2H-EVK

The RZ/V2H-EVK also provide on-board DIP switches for boot mode and boot device selection.

Use these settings after factory flashing to boot from the programmed device.

Table 32: DSW1 – Boot Device Selection (Normal Boot)

Boot device	DSW1-1	DSW1-2	DSW1-3	DSW1-4	DSW1-5	DSW1-6	DSW1-7	DSW1-8
eMMC	ON	OFF	ON	ON	ON	OFF	OFF	OFF
xSPI	ON	OFF	ON	OFF	OFF	OFF	OFF	OFF
SD / eSD	ON	OFF	ON	ON	OFF	OFF	OFF	OFF

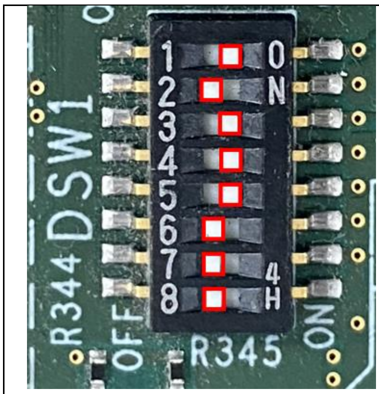


Figure 155. DSW1 - eMMC Boot Mode

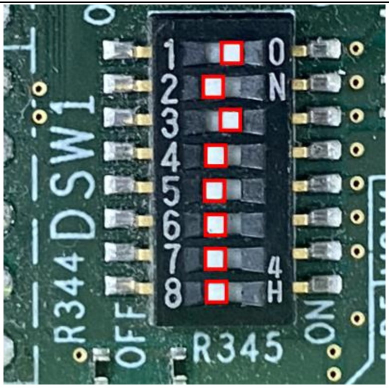


Figure 156. DSW1 – xSPI Boot Mode

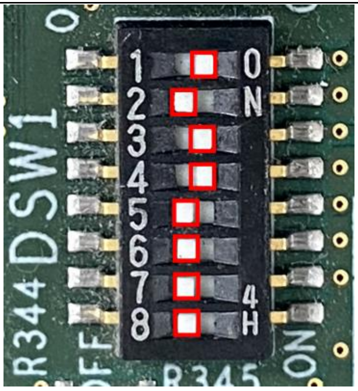


Figure 157. DSW1 – eSD Boot Mode

DSW2 controls output from the on-board clock generator (5P35023B) and one protected utility signal.

Table 33: DSW2 - Audio Clock / Utility DIP

Switch	Signal	OFF (default)	ON
1	Audio_CLKB_OE	Disables 5P35023B Audio_CLKB output	Enables Audio_CLKB output
2	Audio_CLKB	Audio_CLKB not supplied	Audio_CLKB is driven
3	Audio_CLKC_OE	Disables 5P35023B Audio_CLKC output	Enables Audio_CLKC output
4	Audio_CLKC	Audio_CLKC not supplied	Audio_CLKC is driven
5	NEN_VPROG	Must remain OFF	Prohibited — do not set ON
6	—	—	—

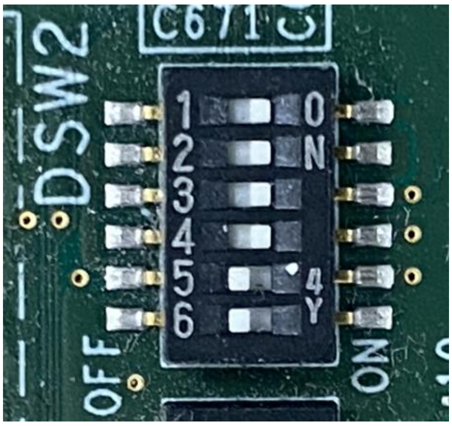


Figure 158. DSW2 - Audio Clock / Utility DIP

The table below lists the settings of the DIP switch (JSW1 on the RZ/V2H Secure Evaluation Board) and its functions.

Table 34: JSW1 Functions

Switch	Function
1-2	MIPI CSI-2 camera interface voltage: 1.8 V
2-3	MIPI CSI-2 camera interface voltage: 3.3 V (default)

Note: Set this switch according to the interface voltage of the camera module to be connected.

16.3 Prepare the eMMC root filesystem

The onboard eMMC can be used as the main root filesystem, but it must first be initialized with a valid Linux rootfs. Since the eMMC is initially empty or unformatted, it cannot be used directly.

During preparation, the system must boot Linux using an SD card rootfs, while BL2/FIP are loaded from either QSPI or eMMC depending on the boot mode. Once Linux is running, the onboard eMMC (e.g. /dev/mmcblk0) becomes accessible and can be partitioned, formatted, and written with the rootfs.

Step 1. Prepare the rootfs archive

- Obtain the provided root filesystem archive (e.g core-image-weston.tar.bz2). Obtain the provided root filesystem archive, which is included in the release images under:
  ```
  rz-cmn-srp/target/images/rootfs
  ```
- Copy the archive to an SD card. This SD card will later be used as the source for writing the rootfs into the onboard eMMC.

Step2. Boot the board in QSPI/eMMC mode

- Set DIP switch to select the boot source: QSPI or eMMC. Refer to Section 16.2. Boot Mode Reference (Non-SCIF) for the correct switch positions for the target boards.
- Insert the prepared SD card (from Step 1) into the correct slot.

Table 35. SD card slot used for rootfs preparation

Board	SD card slot used for rootfs preparation	Notes
RZ/G2L-EVK	CN10 (Carrier microSD slot)	SW1-2 must be set to **ON (eMMC)**
RZ/V2L-EVK	CN10 (Carrier microSD slot)	SW1-2 must be set to **ON (eMMC)**
RZ/V2H-EVK	SoM microSD slot (on CPU board, underneath the module)	--

Note: Some early RZ/V2H-EVK revisions included two SD slots, and either slot could be used for rootfs preparation. The production version provides only the SoM slot underneath the CPU board. Always check the hardware manual for the board revision in use.

- Power on the board

Step 3. Identify the eMMC device

Determine which MMC index corresponds to the onboard eMMC. This can be checked in either U-Boot or Linux.

- In U-boot: Run the following commands:
  1. Reboot or power on the board
  2. When the message “Hit any key to stop autoboot” appears, press a key to interrupt boot and enter the U-Boot console (=>).
  3. Run the following commands:

```
=> mmc rescan
=> mmc list
```

After rescanning, the list will show which device is the eMMC. The example below is for the RZ/G2L-EVK, where **mmc0** is identified as the eMMC device and **mmc1** as the SD card (CN10).

```
=> mmc list
sd@11c00000: 0
sd@11c10000: 1

=> mmc rescan
=> mmc list
sd@11c00000: 0 (eMMC)
sd@11c10000: 1
```

- Run *lsblk* to list all block devices. The numbering in Linux matches the U-Boot mapping: for example, if U-Boot shows mmc0 = eMMC, then in Linux the eMMC will appear as /dev/mmcblk0.

```
root@rz-cmn:~# lsblk
```

Example output:

```
mmcblk0      179:0    0 59.3G  0 disk
|-mmcblk0p1  179:1    0  100M  0 part
`-mmcblk0p2  179:2    0  2.5G  0 part
```

Step 4. Create the partition table

```
root@rz-cmn:~# fdisk /dev/mmcblk0
```

Inside fdisk, create the partition table as follows:

1. Press o → create a new DOS disklabel.
2. Press n, then p, then ENTER for defaults, then type +500M → creates Partition 1 (boot).
3. Press n, then p, then ENTER twice for defaults → creates Partition 2 (rootfs).
4. Press t, then select partition 1, then type b → set Partition 1 type to W95 FAT32.
5. Press w → write and exit.

Afterward, check the partition table with the command below:

```
root@rz-cmn:~# fdisk -l /dev/mmcblk0
```

Expected table:

```
root@rz-cmn:~# fdisk -l  /dev/mmcblk0
Disk /dev/mmcblk0: 59.28 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: 0x076c4a2a

Device         Boot  Start     End Sectors  Size Id Type
/dev/mmcblk0p1 *        32  204831  204800  100M  c W95 FAT32 (LBA)
/dev/mmcblk0p2      204832 5376159 5171328  2.5G 83 Linux
```

Step 5. Format the partitions

Format Partition 1 as FAT32 (boot):

```
root@rz-cmn:~# mkfs.vfat -F 32 -n boot /dev/mmcblk0p1
```

Format Partition 2 as ext4 (rootfs)

```
root@rz-cmn:~# mkfs.ext4 -L rootfs /dev/mmcblk0p2
```

Step 6. Populate the rootfs and kernel Image

Once the partitions are formatted, the eMMC must be populated with the Linux root filesystem and the kernel image.

4. Mount the partitions

   ```
   root@rz-cmn:~# mkdir -p /mnt/boot
   root@rz-cmn:~# mkdir -p /mnt/rootfs

   root@rz-cmn:~# mount /dev/mmcblk0p1 /mnt/boot
   root@rz-cmn:~# mount /dev/mmcblk0p2 /mnt/rootfs
   ```

5. Extract the rootfs archive into the rootfs partition

   ```
   root@rz-cmn:~# cd /mnt/rootfs
   root@rz-cmn:~# tar xf /home/root/core-image-weston.tar.bz2 .
   root@rz-cmn:~# sync
   ```

6. Copy the kernel Image, DTBs and user environment (uEnv.txt) to the boot partition

   ```
   root@rz-cmn:~# cp -rf /boot/* /mnt/boot/
   ```

7. Unmount the partitions

   ```
   root@rz-cmn:~# umount /mnt/boot
   root@rz-cmn:~# umount /mnt/rootfs
   ```

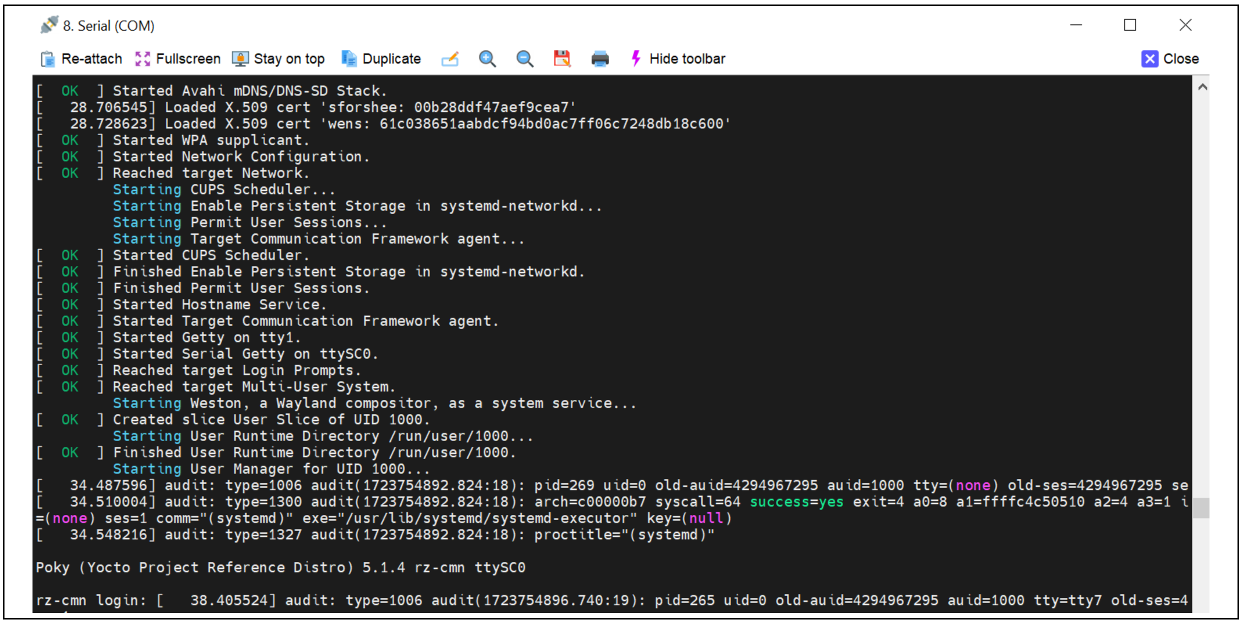
At this point, the eMMC contains:

- Partition 1 (boot): Kernel Image + DTBs + user environment (uEnv.txt)
- Partition 2 (rootfs): Full Linux root filesystem

The board can now boot using eMMC as the root filesystem.

16.4 How To Get the Console After Bootup

Once the RZ boards has booted, on the UART terminal, you will be able to login using the default user ‘root’. There is no password. Leave the password field empty and just hit the return / enter key.



```
[  OK  ] Started Avahi mDNS/DNS-SD Stack.
[   28.706545] Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[   28.728623] Loaded X.509 cert 'wens: 61c038651aabdcf94bd0ac7ff06c7248db18c600'
[  OK  ] Started WPA supplicant.
[  OK  ] Started Network Configuration.
[  OK  ] Reached target Network.
         Starting CUPS Scheduler...
         Starting Enable Persistent Storage in systemd-networkd...
         Starting Permit User Sessions...
         Starting Target Communication Framework agent...
[  OK  ] Started CUPS Scheduler.
[  OK  ] Finished Enable Persistent Storage in systemd-networkd.
[  OK  ] Finished Permit User Sessions.
[  OK  ] Started Hostname Service.
[  OK  ] Started Target Communication Framework agent.
[  OK  ] Started Getty on tty1.
[  OK  ] Started Serial Getty on ttySC0.
[  OK  ] Reached target Login Prompts.
[  OK  ] Reached target Multi-User System.
         Starting Weston, a Wayland compositor, as a system service...
[  OK  ] Created slice User Slice of UID 1000.
         Starting User Runtime Directory /run/user/1000...
[  OK  ] Finished User Runtime Directory /run/user/1000.
         Starting User Manager for UID 1000...
[   34.487596] audit: type=1006 audit(1723754892.824:18): pid=269 uid=0 old-auid=4294967295 auid=1000 tty=(none) old-ses=4294967295 se
[   34.510004] audit: type=1300 audit(1723754892.824:18): arch=c00000b7 syscall=64 success=yes exit=4 a0=8 a1=ffffc4c50510 a2=4 a3=1 i
=(none) ses=1 comm="(systemd)" exe="/usr/lib/systemd/systemd-executor" key=(null)
[   34.548216] audit: type=1327 audit(1723754892.824:18): proctitle="(systemd)"

Poky (Yocto Project Reference Distro) 5.1.4 rz-cmn ttySC0

rz-cmn login: [   38.405524] audit: type=1006 audit(1723754896.740:19): pid=265 uid=0 old-auid=4294967295 auid=1000 tty=tty7 old-ses=4
```

Figure 105. Root login of Linux console over UART.

16.5 BSP Interfaces

Renesas provides a dedicated BSP Manual Set for the RZ/G2L, RZ/V2L and RZ/V2H Evaluation Kits (EVKs), offering technical guidance on SoC configuration, supported drivers, and Linux system integration.

It is a key reference for developers working with the Verified Linux Package (VLP) on these platforms.

- Download the BSP Manual Set here: RZ/G2L, RZ/Five, RZ/V2L BSP Manual Set (v4.00)
- Download the RZ/V2H BSP Manual Set: RZ/V2H BSP Manual Set (v1.01)

17. Troubleshooting

17.1 Unable To Support Scripts for Bootloader/Firmware Flashing on Linux

Not all Linux distributions ship with the Python3 package and its modules, which are required to run the support scripts described in the Section 7. Programming / Flashing Firmware

To make sure your Linux machine can run the support scripts successfully, execute the following commands (This example is for Ubuntu 20.04):

```
$ sudo apt update
$ sudo apt install -y python3 python3-pip
$ pip3 install pyserial==3.5 argparse==1.4.0
```

These commands update system packages, install Python 3 and its pip tool, and then install the required Python modules (‘pyserial’ and ‘argparse’) at the specified versions for the support scripts.

17.2 Flashing Tools Failing Halfway

The flashing tools are used to update the core firmware in the QSPI memory, which forms the core part of the booting process. This should never fail. When a firmware flashing tool fails, the result is often an unbootable ‘bricked’ device. The only way to recover from this is to use a SCIF boot and the respective flashing process described in the appendix section Factory Firmware Flashing Using Serial Downloader (SCIF) Mode

17.3 DHCP Failure

DHCP depends on the network infrastructure and sometimes takes over 30 seconds or fails completely. When the DHCP fails, the SBC will self-assign an IP address from the address range 169.254.x.y pattern series. This series of addresses is called the automatic private IP addresses.

This is often a network issue. At times, eth0 can take longer to get the IP address. If eth0 is not responding, recheck with eth1. Your individual network topology will affect the board's ability to get an IP address through DHCP.

17.4 ‘Ifconfig’ doesn’t list the Wi-Fi interface

The Wi-Fi is not active by default at boot. While all the drivers and subsystems are loaded, the Wi-Fi must be enabled with the command ‘enable Wi-Fi’ in conmanctl utility as described in Wi-Fi 802.11 Module.

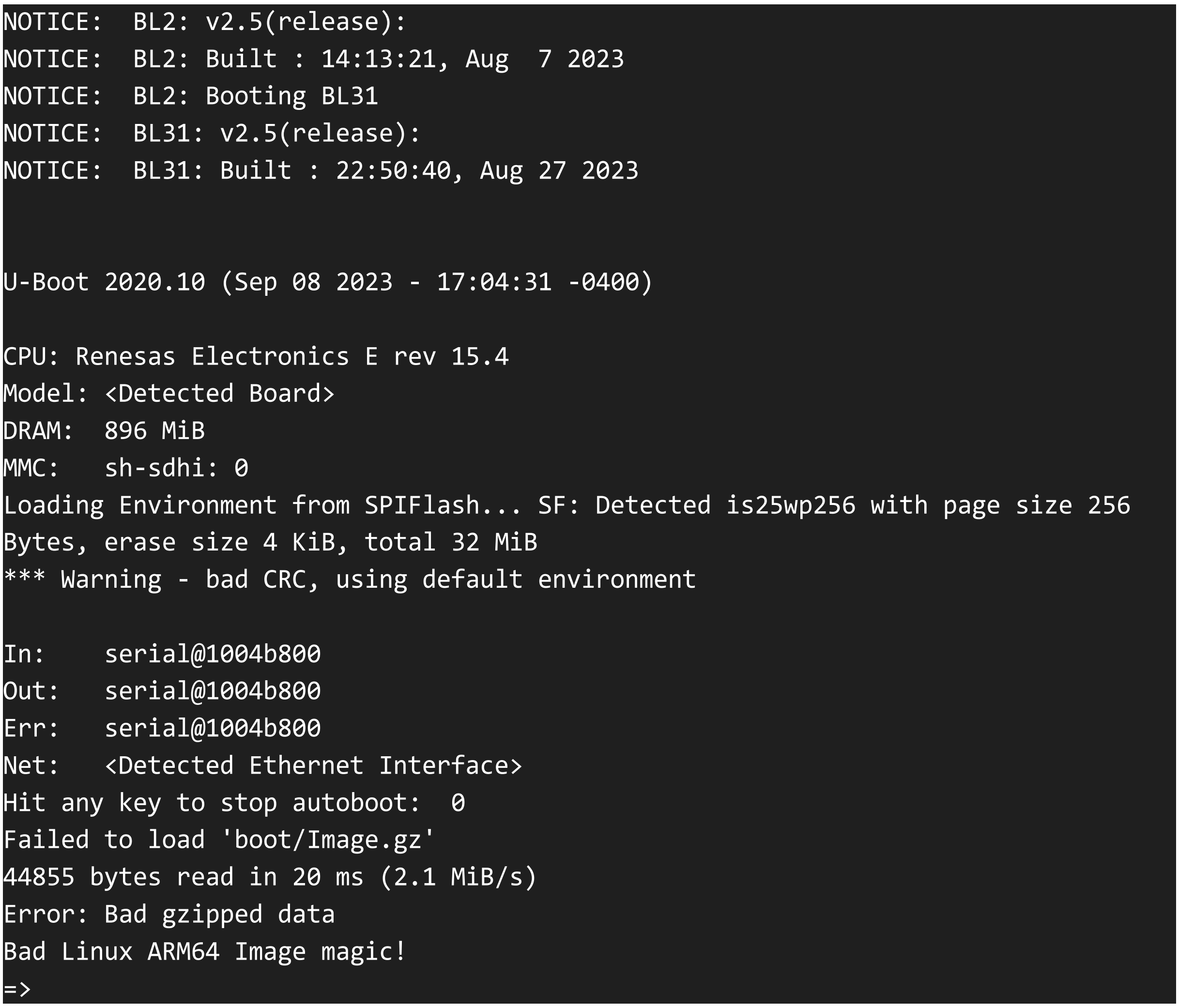
17.5 IP Configuration

An IP address is a bit tricky to get right. It often won’t show up unless the port is powered up, and it gets complicated to identify the interface name and ensure there is an address on it. There is some trial and error involved in this step for flashing the system image. You can manually assign the IP address to your host if necessary. Refer to the following for more info on Windows IP settings:

1. How to configure a static IP on Windows 10 or 11 | Windows Central
2. Change TCP/IP settings - Microsoft Support

17.6 Stuck in U-boot with error “Bad Linux ARM64 Image magic!”

There is a very rare situation in which a board might refuse to boot the Linux image. It usually displays the following in the UART:



```
NOTICE:  BL2: v2.5(release):
NOTICE:  BL2: Built : 14:13:21, Aug  7 2023
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.5(release):
NOTICE:  BL31: Built : 22:50:40, Aug 27 2023


U-Boot 2020.10 (Sep 08 2023 - 17:04:31 -0400)

CPU: Renesas Electronics E rev 15.4
Model: <Detected Board>
DRAM:  896 MiB
MMC:   sh-sdhi: 0
Loading Environment from SPIFlash... SF: Detected is25wp256 with page size 256
Bytes, erase size 4 KiB, total 32 MiB
*** Warning - bad CRC, using default environment

In:    serial@1004b800
Out:   serial@1004b800
Err:   serial@1004b800
Net:   <Detected Ethernet Interface>
Hit any key to stop autoboot:  0
Failed to load 'boot/Image.gz'
44855 bytes read in 20 ms (2.1 MiB/s)
Error: Bad gzipped data
Bad Linux ARM64 Image magic!
=>
```

This is a board not updated with the newest U-Boot. This is also your chance to try the steps from section 7.3.4. Flashing the uLoad-Bootloader. Once “u-boot” is updated, this issue will be resolved.

18. References

18.1 Git Repositories

Build scripts: Renesas-SST/rz-build-scripts: Build scripts for rz projects (github.com)

Yocto board meta layer: Renesas-SST/meta-renesas: Yocto meta layer for Renesas System Solutions (github.com)

Linux Kernel: Renesas-SST/linux-rz: Linux kernel for System and Solutions Products (github.com)

Arm trusted firmware – A: Renesas-SST/rz-atf: Arm Trusted Firmware implementation for System & Solutions products (github.com)

u-boot: Renesas-SST/u-boot: A u-boot suporting System & Solutions Products (github.com)

flash-writer: Renesas-SST/flash-writer: Serial flashing utility to load into blank boards supporting System & Solutions Products (github.com)

rz-utils: Renesas-SST/rz-utils: Collection of utilities for various workflows related Renesas RZ based devices (github.com)

18.2 RZ/G MPU Series

Product page: RZ/G Series (Linux-based MPU) | Renesas

Wiki: RZ/G - Renesas-wiki - Confluence

Other RZ topics: RZ Topics - Renesas-wiki - Confluence

18.3 RZ/V MPU Series

Product page: RZ/V Embedded AI MPUs | Renesas

Wiki: RZ/V - Renesas-wiki - Confluence

Other RZ topics: RZ Topics - Renesas-wiki - Confluence

18.4 External Resources

18.4.1 QT Development

Qt official page: Qt | Tools for Each Stage of Software Development Lifecycle

Qt documentation: Qt Documentation | Home

18.4.2 Yocto Project

Official Yocto manual: Yocto Project Reference Manual — The Yocto Project ® 5.1.4 documentation

18.4.3 Linux Kernel Documentation

The Linux Kernel Documentation — The Linux Kernel documentation

18.4.4 Arm Developer Documentation

Main page: https://developer.arm.com/documentation/

Armv8 Architecture manual: Arm Architecture Reference Manual for A-profile architecture

Generic Interrupt Controller (GIC) architecture specification: Arm Generic Interrupt Controller (GIC) Architecture Specification

Armv8-A Register manual: Arm Armv8-A Architecture Registers

Armv8-A Known issues: Arm Architecture Reference Manual for A-profile architecture: Known issues

Arm Yocto SystemReady IR implimetnation: Deploying Yocto on SystemReady IR compliant hardware (arm.com)

Arm TrustZone SMCC protocol: SMC Calling Convention (SMCCC) (arm.com)

Arm 64-bit ISA architecture: Arm A64 Instruction Set Architecture

18.4.5 JEDEC DDR4

DDR4 SDRAM STANDARD | JEDEC

18.4.6 PMOD Specification

Wiki: Pmod Interface - Wikipedia

Specification document: pmod-interface-specification-1_3_1.pdf (digilent.com)

18.4.7 Essential Linux Tutorial

Linux/Unix Tutorial (geeksforgeeks.org)

UNIX / LINUX Tutorial (tutorialspoint.com)

18.4.8 Packaging

CMake Reference Documentation — CMake 3.30.2 Documentation

CPack — CMake 3.30.2 Documentation

18.4.9 Using Extensible SDK

Using the Extensible SDK

18.4.10 Install Eclipse IDE

Eclipse Installer 2024-09 R | Eclipse Packages

18.4.11 Linux Kernel Development

HOWTO do Linux kernel development — The Linux Kernel documentation

Linux Kernel - GeeksforGeeks

The Linux Kernel Module Programming Guide (sysprog21.github.io)

A Beginner’s Guide to Linux Kernel Development (LFD103) - Linux Foundation - Training

18.4.12 Linux Kernel Driver Development

Basic intro: Device Drivers in Linux - GeeksforGeeks

Drive docs: Driver Basics — The Linux Kernel documentation

Kernel docs: Device Drivers — The Linux Kernel documentation

Lab: Character device drivers — The Linux Kernel documentation (linux-kernel-labs.github.io)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.03.25	—	Initial release
1.10	Jun.04.25	—	Ubuntu release
2.00	Jul.02.25	—	Yocto Styhead release
3.00	Sep.25.25	—	Expanded release with support for multiple boards

System Release Package, RZ Series – User Manual

Publication Date: Sep.25.25

Published by: Renesas Electronics Corporation

RZ Family/ RZ/G Series

Renesas Electronics Corporation