User's Manual

System Release Package

User's Manual: Hardware and Software

Renesas Microprocessor
RZ Series

Renesas Electronics
www.renesas.com

Rev.2.00 Jul.09.25

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 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
- Comprehensive documentation includes a quick start guide, user manual, hierarchical Readme's at every level and licensing information.

Features

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

- Architected to support multiple platforms with 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-the-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.
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 SoC-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 SoC-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-24.04.2-base	A minimal, headless Ubuntu image tailored for embedded systems.

Ubuntu LXDE	ubuntu-lxde-image	ubuntu-base-24.04.2-base	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

Platform	SoC	OPN	Description
RZ/G2L-SBC	RZ/G2L	US157-G2LSBCPOCZ	RZ/G2L-based Pi-compatible SBC.

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.

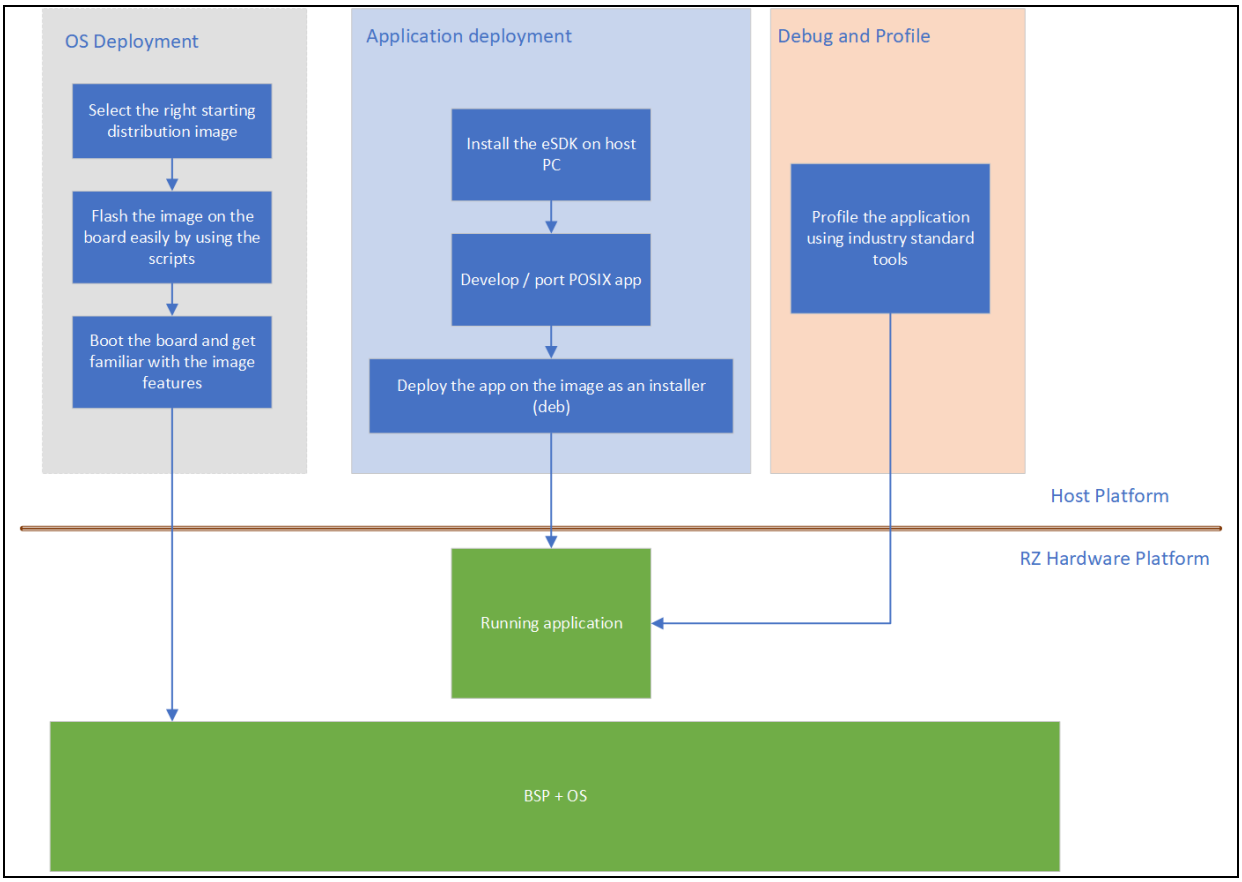


Figure 1. Embedded application workflow for RZ System release package

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 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-srp-<yocto-version>-um-quick-start-guide.pdf
├── <code>-rz-srp-<yocto-version>-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-rzg2l-sbc-<timestamp>.rootfs.manifest lists the contents of the generated root file system.
  - Test data: Files with the *.testdata.json extension that contain 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 rzsbc_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.
        - images.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/G2L-SBC.
        - 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.
        - 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, or SD card image creation.
  Key files:
    - bootloader-flasher: Contains scripts for flashing the bootloader (with sub-directories for Linux and Windows, each with their own instructions via Readme.md files).
    - sd-creator: Utilities for creating SD card images for Linux and Windows.
    - uload-bootloader: Includes automated host-side scripts to flash the qspi boot firmware (IPL) using the images from the SD card.

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:
    - Firmware files: Files like bl2_bp-rzg2l-sbc.bin and bl2-rzg2l-sbc.bin are used to boot the device.
    - System images: Files with the ‘.wic’ extension corresponding to different build variants (BSP, minimal, Weston, Renesas images).
    - dtbs folder: Directory containing ‘.dtb’ and ‘.dtbo’ files necessary for hardware configuration.
    - rootfs folder: Compressed archives (For example, core-image-bsp-rzg2l-sbc.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 4. 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 SoC platforms.
linux-rz	Linux kernel	Renesas-SST/linux-rz: Linux kernel for System and Solutions Products	This repo contains the kernel fork with RZ SoC 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 the RZ SoC patch.
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.

While the public repositories are mostly open source, the RZ SoCs 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 following hardware would be needed to work with the RZ/G2L-SBC:

- Renesas RZ family SoC-based board from the supported list.
- Windows PC with Tera Term software and admin privileges.
- Ubuntu 24.04 host environment as native install, VM, or Docker environment: For working on Yocto distros.
- 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.
- Micro USB cables to interface with a host machine.
- Jumper wires/plugs.
- Mini-HDMI to HDMI display interface cable.
- Ethernet cables for networking.
- 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).

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 the section 1.1.
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-rzg2l-sbc.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.

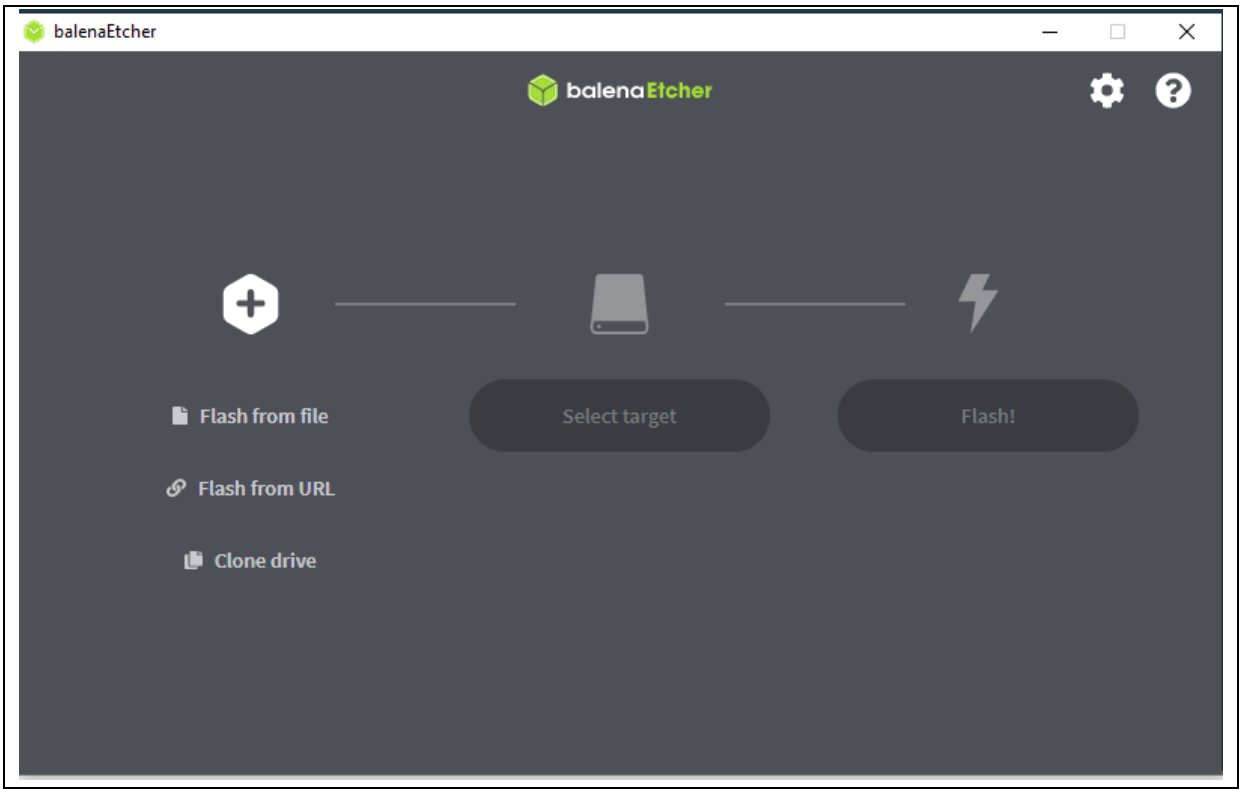


Figure 2. Balena etcher UI

Steps:

1. Select “Flash from File”.
2. In the pop-up window, navigate to your release and select one of the chosen image files (core-image-weston-rzg2l-sbc.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 MMC card into the RZ/G2L-SBC bottom SD MMC card connector.

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.

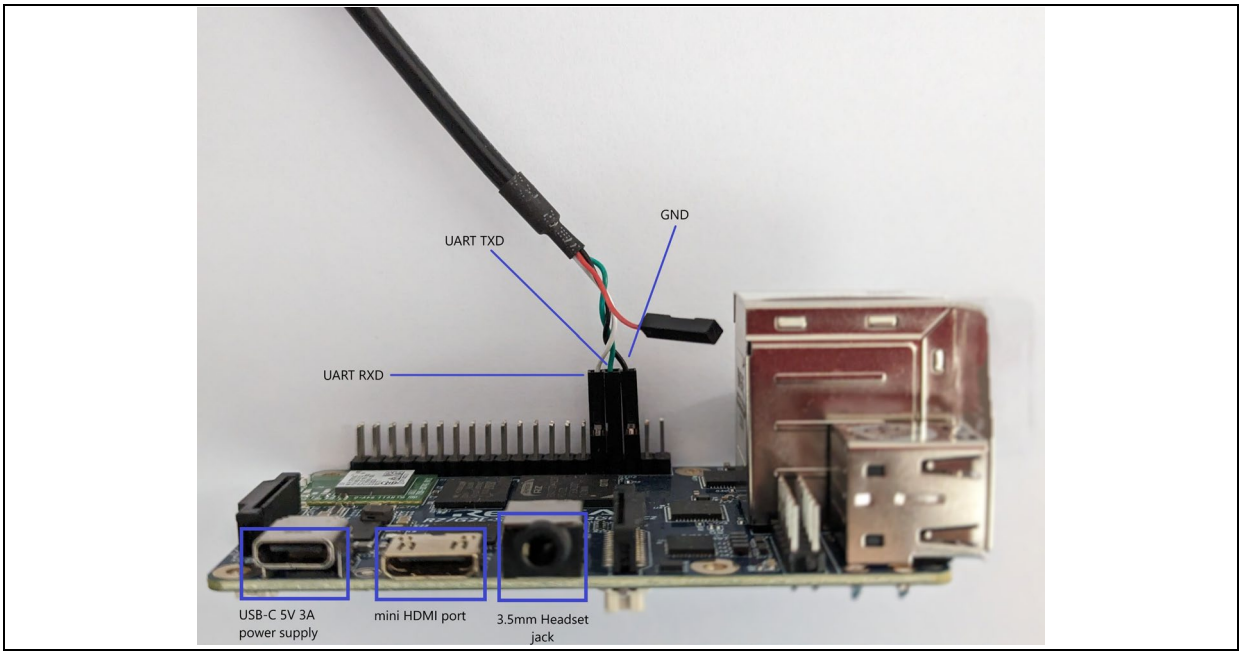


Figure 3. Essential minimum interfaces

4.2.3 Complete Hardware Setup

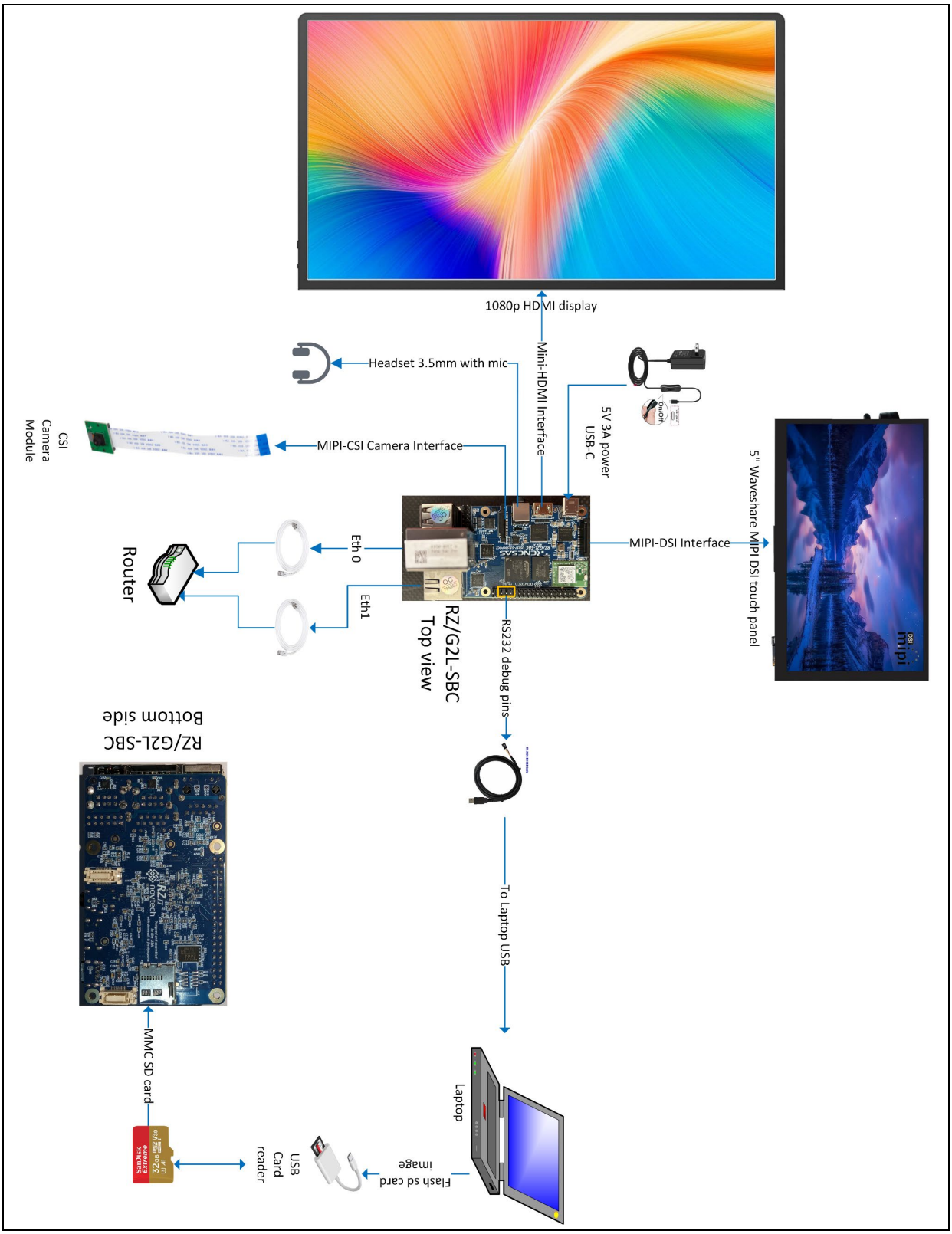


Figure 4. Complete setup

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.

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: Limited to 48 kHz
- Description: Due to hardware limitations, the audio subsystem on the A55 side can only support a 48 kHz sampling rate. This restriction is inherent in the hardware design, preventing the use of any other audio sampling rates.

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.

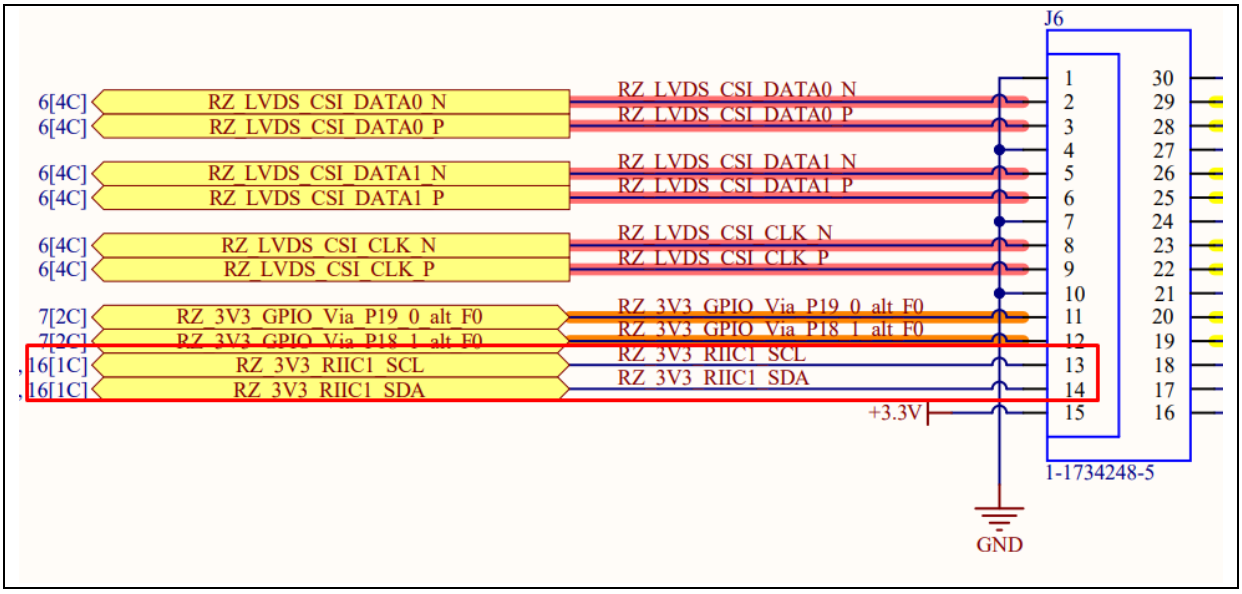


Figure 5. CSI using shared I2C1 bus

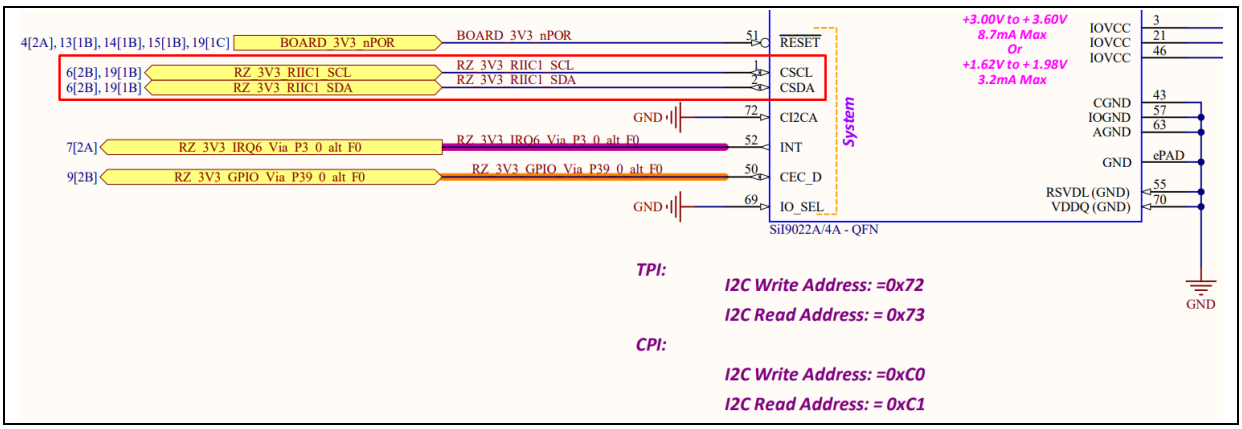


Figure 6. HDMI using shared I2C1 bus

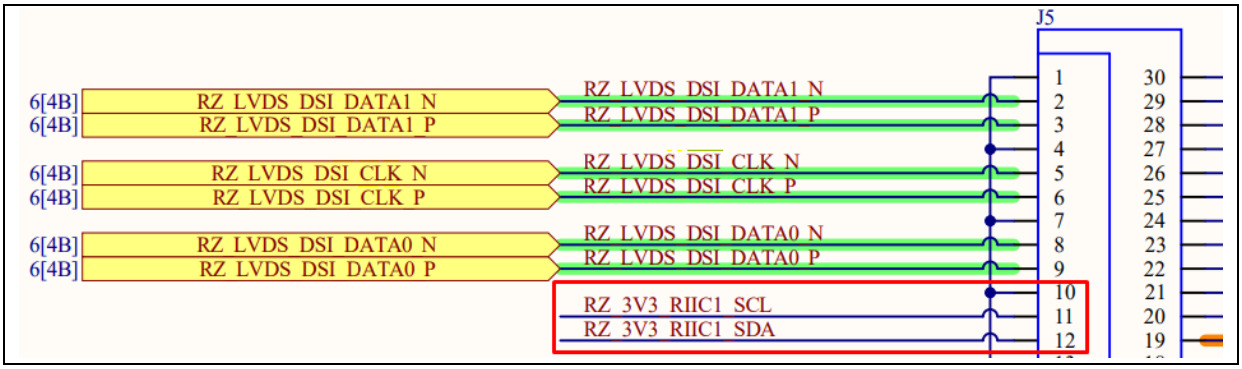


Figure 7. DSI using shared I2C1 bus

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.

5. General Operational Flow

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

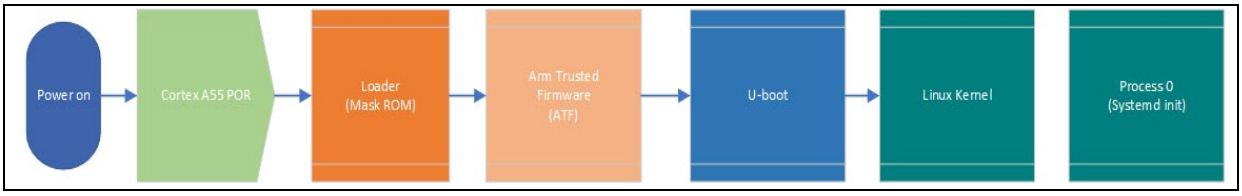


Figure 8. RZ/G2L-SBC boot operational flow

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.

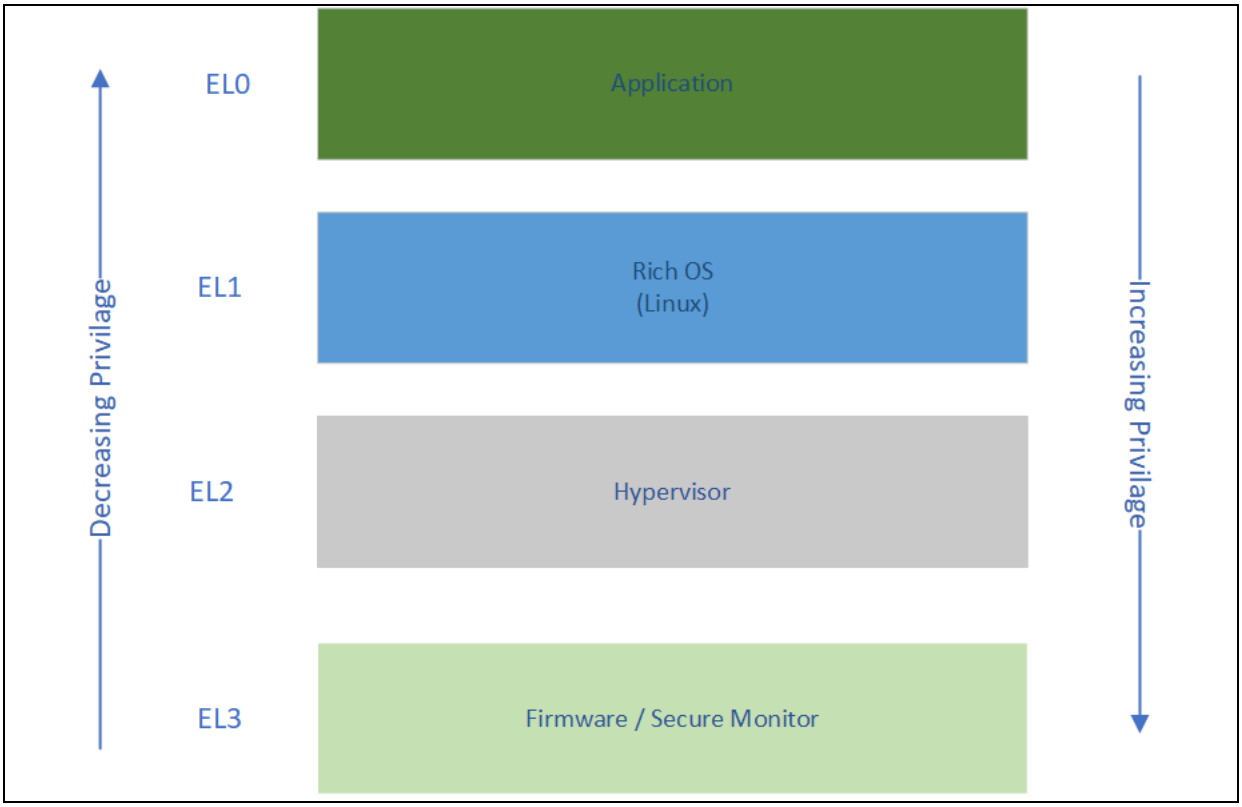


Figure 9. Arm Exception levels

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.

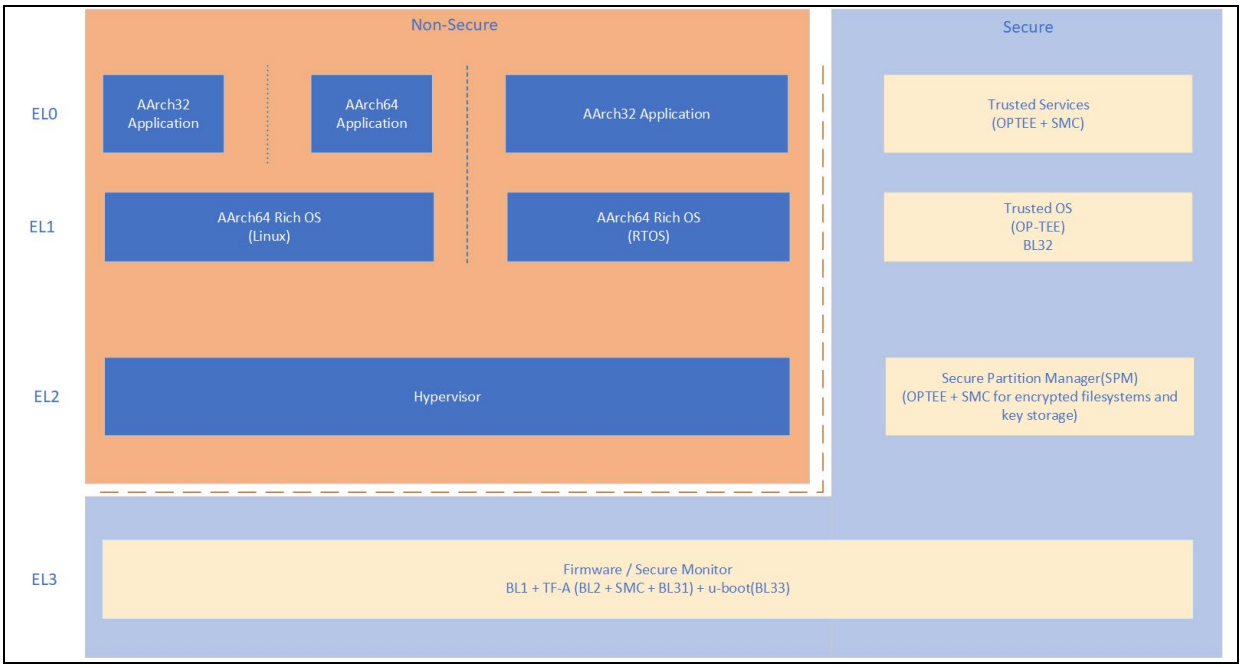


Figure 10. Runtime Exception levels with their states

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 5. 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 SoCs’ 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 the 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 the 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.

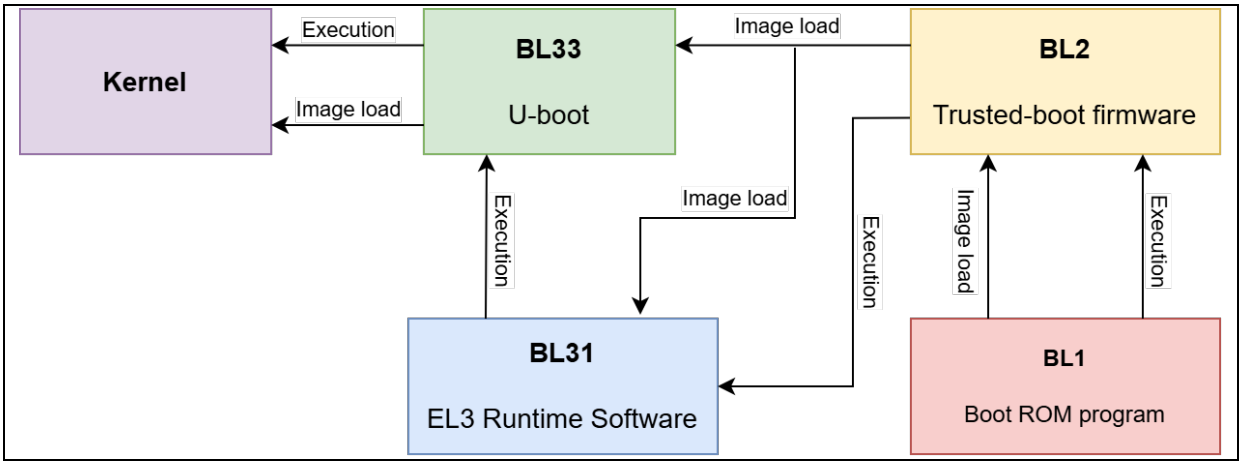


Figure 11. Trusted boot flow

6. OE Build

6.1 Yocto OE Build

This section describes how to prepare a host system, download dependencies, and then perform a full Yocto build. The following sub-sections are step-by-step processes of performing a successful Yocto build.

6.1.1 Yocto Build Host Environment Setup

Requirements

- Ubuntu 24.04 LTS (64bit) is recommended as a build environment as we are building 'Styhead' version of yocto.
- Development packages for Yocto:
  Refer to official Yocto documentation (Yocto Project Documentation) to get started.
  Refer to the official Yocto quick build guide (Yocto Project Quick Build — The Yocto Project ®5.1.4 documentation) for a quick start.

The files listed in Table 6. Prerequisite files from release package are part of the release package. These are essential files to be used for the RZ/G2L-SBC Yocto build. You should find them located under the path 'host/src/' of the release package.

Table 6. Prerequisite files from the release package (Yocto build only)

File	Description
rz-cmn-srp/	Main folder for Yocto/Ubuntu build environment for RZ-G2L/SBC.
rzsbc_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 help 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
images.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	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 the parsing of JSON data.
README.md	A README file describing all the necessary information about the building process.

Install packages on the 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 7. 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. We assume that all the downloaded zip files from Table 7. List of Packages to Manually Download for Yocto Build are collected at the path ‘Downloads/renesas-yocto’ in the user’s home directory creating paths ‘~/Downloads/renesas-yocto/*.zip’. If your locations are different, you must substitute the appropriate paths in the following 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 the package unpacked location ~/Downloads/renesas-yocto/rz-cmn-srp-2.0)

```
$ cd ~/Downloads/renesas-yocto/rz-cmn-srp-2.0/host/src/rz-cmn-srp
$ cp README.md ~/renesas/rz-cmn-srp
$ cp rzsbc_builder.sh ~/renesas/rz-cmn-srp
$ cp site.conf ~/renesas/rz-cmn-srp
$ cp jq-linux-amd64 ~/renesas/rz-cmn-srp
$ cp git_patch.json ~/renesas/rz-cmn-srp
$ cp images.json ~/renesas/rz-cmn-srp
$ cp -r patches ~/renesas/rz-cmn-srp
$ cp -r files_to_add ~/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
.
├── 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
├── images.json
├── 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
├── site.conf       /* (optional) */
└── rzsbc_builder.sh

5 directories, 10 files
```

6.1.2 Initiate Yocto Build

Add execute permission to rzsbc_builder.sh

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

Commence build.

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

There are eight available build targets. For detailed information, refer to 1.1 Supported Distributions.
To build core-image-bsp, use the following command:

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

6.1.3 Build configuration

Building certain recipes or the overall Yocto image can be resource-intensive for the build host (in terms of CPU and memory). To contribute to a stable and efficient build process and to assist in preventing Out-Of-Memory (OOM) conditions, the configuration of BitBake's resource pressure monitoring is available.

BitBake utilizes Linux Kernel's Pressure Stall Information (PSI), which provides insights into CPU, I/O, and Memory resource contention. PSI data is exposed via `/proc/pressure` and is supported in Linux kernels from version 4.20 onwards. If resource pressure exceeds configured thresholds, BitBake's scheduler may pause the initiation of new tasks, which can assist in preventing system unresponsiveness or Out-Of-Memory (OOM) conditions.

The following variables can be set in the `conf/local.conf` file to control BitBake's behavior under resource pressure:

- **BB_PRESSURE_MAX_CPU**: Configures the maximum tolerable CPU pressure threshold.
- **BB_PRESSURE_MAX_MEMORY**: Configures the maximum tolerable Memory pressure threshold.
- **BB_PRESSURE_MAX_IO**: (Optional) Configures the maximum tolerable I/O pressure threshold.

The values assigned to these variables are expressed in internal "pressure units," which are not direct measurements (e.g., MiB/GiB). They represent the difference in "total" pressure from the preceding second. If the actual resource pressure on the build host surpasses the configured threshold, BitBake's scheduler is designed to temporarily suspend the commencement of new tasks until the pressure alleviates.

Based on Yocto Project documentation BitBake User Manual - Reference Variables, the following values are used as initial configurations, intended to balance build execution speed with the stability of the build host:

```
# In build/conf/local.conf
BB_PRESSURE_MAX_MEMORY = "100000"
#BB_PRESSURE_MAX_CPU = "15000"
#BB_PRESSURE_MAX_IO = "15000" # May be included for disk-intensive builds
```

In the default `local.conf` configuration provided with this project, `BB_PRESSURE_MAX_MEMORY` is enabled with a value of "100000". `BB_PRESSURE_MAX_CPU` and `BB_PRESSURE_MAX_IO` are commented out by default.

> Note: To enable or adjust any of these resource pressure variables, modify the corresponding lines in the `conf/local.conf` file by uncommenting them (if necessary) and setting the desired values. Please adjust these values as needed based on your specific build host and workload requirements.

6.1.4 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_rzsbc_board/build/tmp/deploy/images/rzg2l-sbc`
The output folder outline should look as follows:

```
renesas@builder-pc:~/renesas/rz-cmn-
srp/yocto_rzsbc_board/build/tmp/deploy/images/rzg2l-sbc$ tree
.
├── host
│   ├── build
│   │   ├── core-image-bsp-rzg2l-sbc-<timestamp>.rootfs.manifest
│   │   ├── core-image-bsp-rzg2l-sbc-<timestamp>.testdata.json
```

```
|   |   ├── core-image-bsp-rzg2l-sbc.manifest -> core-image-bsp-rzg2l-sbc-
<timestamp>.rootfs.manifest
|   |   ├── core-image-bsp-rzg2l-sbc.testdata.json -> core-image-bsp-rzg2l-sbc-
<timestamp>.testdata.json
|   |   ├── core-image-minimal-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── core-image-minimal-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── core-image-minimal-rzg2l-sbc.manifest -> core-image-minimal-rzg2l-sbc-
<timestamp>.rootfs.manifest
|   |   ├── core-image-minimal-rzg2l-sbc.testdata.json -> core-image-minimal-
rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── core-image-weston-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── core-image-weston-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── core-image-weston-rzg2l-sbc.manifest -> core-image-weston-rzg2l-sbc-
<timestamp>.rootfs.manifest
|   |   ├── core-image-weston-rzg2l-sbc.testdata.json -> core-image-weston-rzg2l-
sbc-<timestamp>.testdata.json
|   |   ├── renesas-core-image-cli-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-core-image-cli-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-core-image-cli-rzg2l-sbc.manifest -> renesas-core-image-cli-
rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-core-image-cli-rzg2l-sbc.testdata.json -> renesas-core-image-
cli-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-core-image-weston-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-core-image-weston-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-core-image-weston-rzg2l-sbc.manifest -> renesas-core-image-
weston-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-core-image-weston-rzg2l-sbc.testdata.json -> renesas-core-
image-weston-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-quickboot-cli-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-quickboot-cli-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-quickboot-cli-rzg2l-sbc.manifest -> renesas-quickboot-cli-
rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-quickboot-cli-rzg2l-sbc.testdata.json -> renesas-quickboot-
cli-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-quickboot-wayland-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-quickboot-wayland-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-quickboot-wayland-rzg2l-sbc.manifest -> renesas-quickboot-
wayland-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-quickboot-wayland-rzg2l-sbc.testdata.json -> renesas-
quickboot-wayland-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-ubuntu-rzg2l-sbc-<timestamp>.rootfs.manifest
|   |   ├── renesas-ubuntu-rzg2l-sbc-<timestamp>.testdata.json
|   |   ├── renesas-ubuntu-rzg2l-sbc.manifest -> renesas-ubuntu-rzg2l-sbc-
<timestamp>.rootfs.manifest
|   |   └── renesas-ubuntu-rzg2l-sbc.testdata.json -> renesas-ubuntu-rzg2l-sbc-
<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
|   |       ├── 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
|   |       ├── images.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
|   |       ├── rzsbc_builder.sh
|   |       └── ubuntu
|   |           ├── config
|   |           |   ├── ubuntu_core
|   |           |   |   ├── network_interfaces.conf
|   |           |   |   ├── NetworkManager.conf
|   |           |   |   └── resolved.conf
|   |           |   └── ubuntu_lxde
|   |           |       ├── connman-gtk.desktop
|   |           |       ├── interfaces
|   |           |       ├── lightdm.conf
|   |           |       ├── NetworkManager.conf
|   |           |       ├── panel
|   |           |       ├── rsyslog
|   |           |       ├── ttyS0.conf
|   |           |       └── v4l2-init.sh
|   |           ├── config.ini
|   |           ├── docs
|   |           |   ├── ubuntu_core
|   |           |   |   └── README.md
|   |           |   └── ubuntu_lxde
```

```
│   │           │       ├── Pictures
│   │           │       │   ├── audacity.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
|   |           ├── 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
|   |           |       ├── set_root_password.sh
|   |           |       ├── set_swap_enable.sh
|   |           |       └── setup-set-permissions.sh
|   |           └── setup_ubuntu_environment.sh
|   └── tools
|       ├── bootloader-flasher
|       |   ├── linux
|       |   |   ├── bootloader_flash.py
|       |   |   └── Readme.md
|       |   ├── Readme.md
|       |   └── windows
|       |       ├── config.ini
|       |       ├── flash_bootloader.bat
|       |       ├── Readme.md
|       |       └── tools
|       |           ├── cygterm.cfg
|       |           ├── flash_bootloader.ttl
|       |           ├── TERATERM.INI
|       |           ├── ttermpro.exe
|       |           ├── ttpcmn.dll
|       |           ├── ttpfile.dll
|       |           ├── ttpmacro.exe
|       |           ├── ttpset.dll
|       |           └── ttxssh.dll
|       ├── Readme.md
|       ├── sd-creator
|       |   ├── linux
|       |   |   ├── Readme.md
```

```
│       │   │   └── sd_flash.sh
│       │   ├── Readme.md
│       │   └── windows
│       │       ├── config.ini
│       │       ├── flash_filesystem.bat
│       │       ├── Readme.md
│       │       └── tools
│       │           ├── AdbWinApi.dll
│       │           ├── cygterm.cfg
│       │           ├── fastboot.bat
│       │           ├── fastboot.exe
│       │           ├── flash_system_image.ttl
│       │           ├── TERATERM.INI
│       │           ├── ttermpro.exe
│       │           ├── ttpcmn.dll
│       │           ├── ttpfile.dll
│       │           ├── ttpmacro.exe
│       │           ├── ttpset.dll
│       │           └── ttxssh.dll
│       └── uload-bootloader
│           ├── linux
│           │   ├── Readme.md
│           │   └── uload_bootloader_flash.py
│           ├── Readme.md
│           └── windows
│               ├── config.ini
│               ├── Readme.md
│               ├── tools
│               │   ├── cygterm.cfg
│               │   ├── TERATERM.INI
│               │   ├── ttermpro.exe
│               │   ├── ttpcmn.dll
│               │   ├── ttpfile.dll
│               │   ├── ttpmacro.exe
│               │   ├── ttpset.dll
│               │   ├── ttxssh.dll
│               │   └── uload-flash_bootloader.ttl
│               └── uload-flash_bootloader.bat
├── license
│   └── Disclaimer051.pdf
├── <code>-rz-srp-<yocto-version>-um-quick-start-guide.pdf
├── <code>-rz-srp-<yocto-version>-um.pdf
├── README.md
├── RZ_System_Release_Package_Evaluation_license.pdf
└── target
    ├── env
    │   ├── Readme.md
```

```
    │   └── uEnv.txt
    ├── images
    │   ├── bl2_bp-rzg2l-sbc.bin
    │   ├── bl2_bp-rzg2l-sbc.srec
    │   ├── bl2-rzg2l-sbc.bin
    │   ├── core-image-bsp-rzg2l-sbc.wic
    │   ├── core-image-minimal-rzg2l-sbc.wic
    │   ├── core-image-weston-rzg2l-sbc.wic
    │   ├── 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
    │   │   ├── Readme.md
    │   │   ├── rzg2l-sbc--6.10.14+git0+<commit-hash>-r0-rzg2l-sbc-
<timestamp>.dtbo
    │   │   └── rzg2l-sbc.dtb -> rzg2l-sbc--6.10.14+git0+<commit-hash>-r0-rzg2l-
sbc-<timestamp>.dtbo
    │   ├── fip-rzg2l-sbc.bin
    │   ├── fip-rzg2l-sbc.srec
    │   ├── Flash_Writer_SCIF_rzg2l-sbc.mot
    │   ├── Flash_Writer_SCIF_rzg2l-sbc_PMIC.mot
    │   ├── Image -> Image--6.10.14+git0+<commit-hash>-r0-rzg2l-sbc-
<timestamp>.bin
    │   ├── Image--6.10.14+git0+<commit-hash>-r0-rzg2l-sbc-<timestamp>.bin
    │   ├── Readme.md
    │   ├── renesas-core-image-cli-rzg2l-sbc.wic
    │   ├── renesas-core-image-weston-rzg2l-sbc.wic
    │   ├── renesas-quickboot-cli-rzg2l-sbc.wic
    │   ├── renesas-quickboot-wayland-rzg2l-sbc.wic
    │   └── rootfs
    │       ├── core-image-bsp-rzg2l-sbc.tar.bz2
    │       ├── core-image-minimal-rzg2l-sbc.tar.bz2
    │       ├── core-image-weston-rzg2l-sbc.tar.bz2
    │       ├── Readme.md
    │       ├── renesas-core-image-cli-rzg2l-sbc.tar.bz2
    │       ├── renesas-core-image-weston-rzg2l-sbc.tar.bz2
    │       ├── renesas-quickboot-cli-rzg2l-sbc.tar.bz2
    │       ├── renesas-quickboot-wayland-rzg2l-sbc.tar.bz2
    │       └── renesas-ubuntu-rzg2l-sbc.tar.bz2
    └── Readme.md
```

6.2 Ubuntu OE Build

This section describes how to build a custom Ubuntu Core image for the RZ/G2L-SBC. 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/G2L-SBC.

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.

6.2.1 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 provided build script will handle all necessary steps, including Yocto and do the Ubuntu root filesystem setup. Refer to section Yocto OE Build only for additional details on the Yocto environment, if needed.

Table 8. Prerequisite files from the release package for the Ubuntu build

File	Description
rz-cmn-srp/	Main folder for Yocto/Ubuntu build environment for RZ-G2L/SBC.
rzsbc_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.
images.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).
jq-linux-amd64	A lightweight and flexible tool that supports parsing JSON files.
README.md	A README file describing all the necessary information about the building process.
ubuntu/	Main folder for Ubuntu-based image generation for RZ/G2L-SBC.
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.

Install packages on Ubuntu Host.

1. Ensure that all Yocto build tools and packages described in Section 6.1.1 Yocto build Host Environment Setup have been completed.
2. Create folders for Ubuntu and Yocto workspace, then 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/
$ mkdir -p ~/renesas/rz-cmn-srp/ubuntu
$ mv *.zip ~/renesas/rz-cmn-srp
```

3. Copy the files ‘rzsbc_builder.sh’, ‘site.conf’, ‘README.md’, ‘jq-linux-amd64’ and ‘patches’ folder from the release package into ‘~/renesas/rz-cmn-srp’ folder. (This example assumes the pre-requisite files that are described in Table 5. RZ-specific TF-A implementation are located at package unpacked location ~/Downloads/renesas-yocto/rz-cmn-srp-2.0) for Yocto build.

```
$ cd ~/Downloads/renesas-yocto/rz-cmn-srp-2.0/host/src/rz-cmn-srp
$ cp README.md ~/renesas/rz-cmn-srp
$ cp rzsbc_builder.sh ~/renesas/rz-cmn-srp
$ cp site.conf ~/renesas/rz-cmn-srp
$ cp jq-linux-amd64 ~/renesas/rz-cmn-srp
$ cp git_patch.json ~/renesas/rz-cmn-srp
$ cp images.json ~/renesas/rz-cmn-srp
$ cp -r patches ~/renesas/rz-cmn-srp
$ cp -r files_to_add ~/renesas/rz-cmn-srp
```

4. Copy the files ‘setup_environment_ubuntu.sh’, ‘config.ini’, ‘README.md’, and the ‘scripts’, ‘include’, ‘docs’, and ‘config’ folders from the release package into the ‘~/renesas/rz-cmn-srp/ubuntu’ folder for the Ubuntu build process.

```
$ cd ~/Downloads/renesas-yocto/rz-cmn-srp-2.0/host/src/rz-cmn-srp/ubuntu
$ cp -r config/ ~/renesas/rz-cmn-srp/ubuntu/
$ cp -r docs/ ~/renesas/rz-cmn-srp/ubuntu/
$ cp -r include/ ~/renesas/rz-cmn-srp/ubuntu/
$ cp -r script/ ~/renesas/rz-cmn-srp/ubuntu/
$ cp setup_ubuntu_environment.sh ~/renesas/rz-cmn-srp/ubuntu/
$ cp config.ini ~/renesas/rz-cmn-srp/ubuntu/
$ cp README.md ~/renesas/rz-cmn-srp/ubuntu/
```

Eventually, all the necessary files for the Ubuntu build should be present in ‘renesas’ 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
    ├── jq-linux-amd64
    ├── patches
    │   ├── meta-summit-radio
    │   └── poky
    ├── README.md
    ├── rzsbc_builder.sh
    └── ubuntu
        ├── config
        ├── config.ini
        ├── docs
        ├── include
        ├── script
        └── setup_ubuntu_environment.sh

10 directories, 7 files
```

6.2.2 Initial Ubuntu Build

The **config.ini** file is used for configuring the script that builds an Ubuntu image for ARM systems. It includes essential parameters for partition sizes, the Ubuntu base file, and other configurations needed to create the rootfs and wic image. Here are the parameters that need to be configured before starting the script:

- **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_FILE_NAME**: The file name of the Ubuntu base that will be downloaded.
- **UBUNTU_BASE_LINK**: The link to download the Ubuntu base file.
- **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.

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 ubuntu folder

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

Add execute permission to rzsbc_builder.sh.

```
renesas@builder-pc:~/renesas/rz-cmn-srp$ chmod a+x rzsbc_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> ./rzsbc_builder.sh build
```

Run the following command with the appropriate option:

- ubuntu-core: Build Ubuntu core image
- ubuntu-lxde: Build Ubuntu LXDE image (with graphics 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 ./rzsbc_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.

6.2.3 Collect the Build Output

After building Ubuntu LXDE and Ubuntu Core, the output folder should be located at: `~/renesas/rz-cmn-srp/yocto_rzsbc_board/build/tmp/deploy/images/rzg2l-sbc`
The output folder outline should look as follows:

```
renesas@builder-pc:~/renesas/rz-cmn-
srp/yocto_rzsbc_board/build/tmp/deploy/images/rzg2l-sbc$ tree
.
├── host
│   ├── build
│   ├── env
│   │   └── Readme.md
│   ├── Readme.md
│   ├── Readme.md
│   ├── src
│   │   └── rz-cmn-srp
│   │       ├── 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
│   │       ├── images.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
│   │       ├── rzsbc_builder.sh
│   │       └── ubuntu
│   │           ├── config
│   │           │   ├── ubuntu_core
│   │           │   │   ├── network_interfaces.conf
│   │           │   │   ├── NetworkManager.conf
│   │           │   │   └── resolved.conf
│   │           │   └── ubuntu_lxde
│   │           │       ├── connman-gtk.desktop
│   │           │       ├── interfaces
│   │           │       ├── lightdm.conf
│   │           │       ├── NetworkManager.conf
│   │           │       ├── panel
│   │           │       ├── rsyslog
│   │           │       ├── ttyS0.conf
│   │           │       └── v4l2-init.sh
```

```
|   |           ├── config.ini
|   |           ├── docs
|   |           |   ├── ubuntu_core
|   |           |   |   └── README.md
|   |           |   └── ubuntu_lxde
|   |           |       ├── Pictures
|   |           |       |   ├── audacity.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
│   │           ├── 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
│   │           │       ├── set_root_password.sh
│   │           │       ├── set_swap_enable.sh
│   │           │       └── setup-set-permissions.sh
│   │           └── setup_ubuntu_environment.sh
│   └── tools
│       ├── bootloader-flasher
│       │   ├── linux
│       │   │   ├── bootloader_flash.py
│       │   │   └── Readme.md
│       │   ├── Readme.md
│       │   └── windows
│       │       ├── config.ini
│       │       ├── flash_bootloader.bat
│       │       ├── Readme.md
│       │       └── tools
│       │           ├── cygterm.cfg
│       │           ├── flash_bootloader.ttl
│       │           ├── TERATERM.INI
│       │           ├── ttermpro.exe
│       │           ├── ttpcmn.dll
│       │           ├── ttpfile.dll
│       │           ├── ttpmacro.exe
│       │           ├── ttpset.dll
```

```
│       │           └── ttxssh.dll
│       ├── Readme.md
│       ├── sd-creator
│       │   ├── linux
│       │   │   ├── Readme.md
│       │   │   └── sd_flash.sh
│       │   ├── Readme.md
│       │   └── windows
│       │       ├── config.ini
│       │       ├── flash_filesystem.bat
│       │       ├── Readme.md
│       │       └── tools
│       │           ├── AdbWinApi.dll
│       │           ├── cygterm.cfg
│       │           ├── fastboot.bat
│       │           ├── fastboot.exe
│       │           ├── flash_system_image.ttl
│       │           ├── TERATERM.INI
│       │           ├── ttermpro.exe
│       │           ├── ttpcmn.dll
│       │           ├── ttpfile.dll
│       │           ├── ttpmacro.exe
│       │           ├── ttpset.dll
│       │           └── ttxssh.dll
│       └── uload-bootloader
│           ├── linux
│           │   ├── Readme.md
│           │   └── uload_bootloader_flash.py
│           ├── Readme.md
│           └── windows
│               ├── config.ini
│               ├── Readme.md
│               ├── tools
│               │   ├── cygterm.cfg
│               │   ├── TERATERM.INI
│               │   ├── ttermpro.exe
│               │   ├── ttpcmn.dll
│               │   ├── ttpfile.dll
│               │   ├── ttpmacro.exe
│               │   ├── ttpset.dll
│               │   ├── ttxssh.dll
│               │   └── uload-flash_bootloader.ttl
│               └── uload-flash_bootloader.bat
├── license
│   └── Disclaimer051.pdf
├── <code>-rz-srp-<yocto-version>-um-quick-start-guide.pdf
├── <code>-rz-srp-<yocto-version>-um.pdf
```

```
├── README.md
├── RZ_System_Release_Package_Evaluation_license.pdf
└── target
    ├── env
    │   ├── Readme.md
    │   └── uEnv.txt
    ├── images
    │   ├── bl2_bp-rzg2l-sbc.bin
    │   ├── bl2_bp-rzg2l-sbc.srec
    │   ├── bl2-rzg2l-sbc.bin
    │   ├── 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
    │   │   ├── Readme.md
    │   │   ├── rzg2l-sbc--6.10.14+git0+<commit-hash>-r0-rzg2l-sbc-
<timestamp>.dtbo
    │   │   └── rzg2l-sbc.dtb -> rzg2l-sbc--6.10.14+git0+<commit-hash>-r0-rzg2l-
sbc-<timestamp>.dtbo
    │   ├── fip-rzg2l-sbc.bin
    │   ├── fip-rzg2l-sbc.srec
    │   ├── Flash_Writer_SCIF_rzg2l-sbc.mot
    │   ├── Image -> Image--6.10.14+git0+af06ad75b8_bbe3d1be4e-r0-rzg2l-sbc-
20250703132240.bin
    │   ├── Image--6.10.14+git0+af06ad75b8_bbe3d1be4e-r0-rzg2l-sbc-
20250703132240.bin
    │   ├── Readme.md
    │   ├── rootfs
    │   │   ├── Readme.md
    │   │   ├── renesas-ubuntu-rzg2l-sbc.tar.bz2
    │   │   ├── ubuntu-core-image-rzg2l-sbc.tar.bz2
    │   │   └── ubuntu-lxde-image-rzg2l-sbc.tar.bz2
    │   ├── ubuntu-core-image-rzg2l-sbc.wic.gz
    │   └── ubuntu-lxde-image-rzg2l-sbc.wic.gz
    └── Readme.md
```

7. Creating A Bootable SD Card On the Host Machine

This section describes all the tools and methods for creating a Linux bootable SD card under different environments of host machines, such as laptops.

7.1 Linux Host

This section explores the SD-flashing tools available in the Linux environment.
There is a helper script `sd_flash.sh` in the `host/tools/sd-creator/linux` folder of the Yocto build output/release directory for this purpose.

Run the following command to learn how to use the script:

```
$ ./sd_flash.sh
```

The script needs an argument to run successfully. The argument is the device to be flashed with the image. In this case, the device needs to flash its SD card. You will have to identify the correct device name that represents the SD card on Linux.

The example below shows how to identify an SD card on Ubuntu 24.04. The command 'lsblk' is executed to check all available storage devices. You can see that the 32 GB SD card is represented under the device name 'sdb' in the result (its full name is /dev/sdb). The command also shows you where the drive partitions are mounted in the filesystem.

```
$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda       8:0    0 119.2G  0 disk
├─sda1    8:1    0   976M  0 part /boot
├─sda2    8:2    0   977M  0 part [SWAP]
├─sda3    8:3    0   977M  0 part /boot/efi
└─sda4    8:4    0 116.4G  0 part /var/snap/firefox/common/host-hunspell
                                  /
sdb       8:16   1  31.6G  0 disk
nvme0n1 259:0    0   1.1T  0 disk /data1
```

Identify the device name of the SD card to be flashed. Then, pass it to the script as an argument, as shown in the example here:

```
$ ./sd_flash.sh /dev/sdb
```

After executing the SD card flashing script successfully, the SD card is automatically unmounted.

Note: Since the various Linux distributions have different disk management arrangements, the script may fail to create the card. Hence, we are unable to assure that the script works in every Linux environment. In the case it fails, you may modify the call for creating the filesystem, like the calls to ext4fs in the script. Pay attention to the script and ensure that it succeeds.The script is tested on Ubuntu 24.04.

The full command allows you to specify the rootfs you want to flash:

```
$ ./sd_flash.sh /dev/sdb <full-path-to-your-rootfs.tar.bz2 file>
```

For example, to flash the Renesas cli image, you can try:

```
$ ./sd_flash.sh /dev/sdb /home/renesas/rz-cmn-srp-
2.0/target/images/rootfs/renesas-core-image-cli-rzg2l-sbc.tar.bz2
```

7.2 Windows Host

The preferred way to flash the image onto the SD card is to use Balena Etcher. For Yocto images, flash the .wic file, such as renesas-core-image-weston-rzg2l-sbc.wic, onto the SD card.

For Ubuntu images, the file type is different; you will use a .wic.gz file, such as ubuntu-lxde-image-rzg2l-sbc.wic.gz. Balena Etcher can directly flash the .wic.gz file onto the SD card without any need for extraction.

8. Programming / Flashing Firmware

The firmware part of this release contains the secure world Trusted firmware images:

Table 9. 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_rzg2l-sbc.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_bp-<board>.bin bl2_bp-<board>.srec	BL2	Trusted Firmware-A implementation binary. Its job is to load BL31, BL32, and u-boot (BL33) binaries into memory. It comes in two formats: • .bin – for raw flashing for native in-system flashing • .srec – motorola srec format for flash writer
Firmware Image Package (FIP)	fip-<board>.bin fip-<board>.srec	BL3 to EL3	This image is also a standard trusted firmware package that is a unified image containing: • BL31 – Trusted Firmware-A Secure monitor • BL32 – Trusted Firmware-A Optee • BL33 – U-boot It comes in two formats: • .bin – for raw flashing for native in-system flashing • .srec – motorola srec format for flash writer

8.1 RZ/G2L-SBC

The RZ/G2L-SBC comes with the most recent firmware images. However, there might be cases where a firmware update may be needed, such as in a factory setting where volume flashing is performed, or a custom version is designed by the end user. Renesas BSP provides firmware update tools to make it seamless to perform these tasks under multiple OS environments.

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.

8.1.1 Hardware Setup

To perform a firmware flashing:

1. The board has the UART console connected to the host PC.

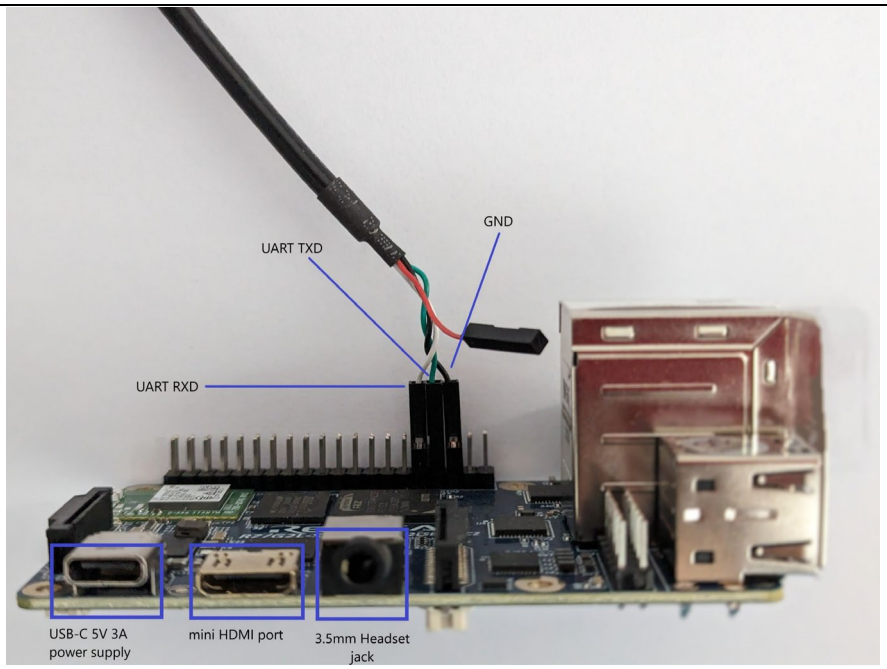


Figure 12. Cortex A55 debug UART cable interface

2. The SD card with the Linux boot image from the release.
3. A 5V 3A USB-C power supply.

Other interfaces are not necessary for this purpose.

8.1.2 Flash Bootloader on U-Boot Console

If users want to update the Bootloader without touching the hardware setup, we support a method for flashing the Bootloader on the U-Boot console. This is especially useful when end customers need to update firmware as part of a field service. This is a straightforward method.

The sub-directory `host/tools/uload-bootloader` in Yocto build output/release folder contains the toolset for SD card flashing. The sub-directory contains its ReadMe (Readme.md) file with the flashing procedure.

Note: Default bootloader images (.bin) are in the subdirectory `/boot/uload-bootloader` of the root filesystem in the SD card. You can put your own bootloader images there and perform a flashing.

Before performing the flashing:

- ✓ Make sure the board is powered off.
- ✓ Connect the debug serial port (SCIF0 - TXD, RXD, GND) to your Linux PC.
- ✓ Insert the SD card with the Linux image (you do not need a separate image for this).
- ✓ Ensure that Teraterm application is installed on your windows pc.
- ✓ Ensure that the minicom and FTDI drivers are loaded properly on the Linux host pc.
- ✓ Ensure that the scripts in the process have executed permissions.

8.1.2.1 Linux Host

The Linux flashing script is named uload_bootloader_flash.py under the uload-bootloader/linux folder.

The script has options, and the details of using it are provided in the Readme.md file at the same location. You know more about the command by issuing a `-h` option while invoking the script.

```
$./uload_bootloader_flash.py -h
```

Note: The script, by default, tries to access /dev/ttyUSB0 without passing any arguments. This works on most systems that have a single FTDI cable attached to a single USB port.

Here are the steps to flash:

1. Ensure that the hardware setup is accurate, as described above.

2. Start the script uload_bootloader_flash.py.

```
renesas@builder-pc:~/renesas/rz-cmn-srp/build/tmp/deploy/images/rzg2l-sbc$ cd
host/tools/uload-bootloader/linux
renesas@builder-pc:~/renesas/rz-cmn-srp/build/tmp/deploy/images/rzg2l-
sbc/host/tools/uload-bootloader/linux$ ./uload_bootloader_flash.py
```

3. Power on the board. The flashing should automatically start and complete.

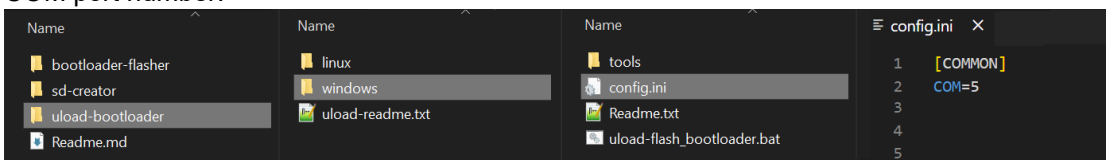
4. Once the flashing is complete, power-cycle the board.

8.1.2.2 Windows Host

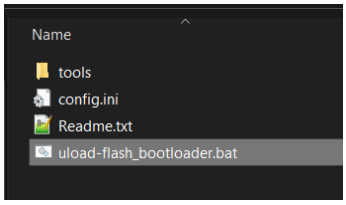
Windows host uses its own script that is cleanly tucked into the sub directory of uload-bootloader called windows. The sub-directory has its own Readme.md that describes everything that is needed.

The Windows script is also a script that only depends on the teraterm TTL scripting tool.

1. Navigate through the release to the Windows utility directory and update the config.ini with the COM port number.



2. Execute the uload-flash_bootloader.bat.



3. Notice application windows open and perform flashing. Once the flashing is completed, it will disconnect from the UART port. Power-cycle the board.

9. Accessing Supported Features

This section explores the key features and interfaces available across Yocto and Ubuntu images on the supported platforms, beginning with the RZ/G2L-SBC.

9.1 Supported Features in Yocto Images

9.1.1 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 10. 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.

9.1.1.1 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@rzg2l-sbc:~# cd network-management
root@rzg2l-sbc:~/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@rzg2l-sbc:~/network-management# ./enable_networking_stack.sh <service>
```

For example, to enable Wi-Fi, run:

```
root@rzg2l-sbc:~/network-management# ./enable_networking_stack.sh wifi
```

To enable all networking services:

```
root@rzg2l-sbc:~/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.

9.1.1.2 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@rzg2l-sbc:~/network-management# ./disable_networking_stack.sh <service>
```

For example, to disable Bluetooth:

```
root@rzg2l-sbc:~/network-management# ./disable_networking_stack.sh bluetooth
```

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

```
root@rzg2l-sbc:~/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.

9.1.1.3 Confirming Interface Names

In Quickboot images, system/udev is disabled in Quickboot images by default, predictable network interface naming (e.g., end0) does not apply at first boot. Instead, interfaces retain traditional names like eth0, eth1, etc.

However, when running enable_networking_stack.sh, system/udevd is re-enabled and restarted. This triggers predictable naming rules, and interface names may change (e.g., from eth0 to end0).

To determine the actual interface name before or after running the script, use:

```
root@rzg2l-sbc:~# ip link
```

Example output:

```
root@rzg2l-sbc:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> ...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
link/ether 1c:a0:d3:20:12:38 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
link/ether 1c:a0:d3:20:12:39 brd ff:ff:ff:ff:ff:ff
```

In this example, ethx is the Ethernet interface name. After the script runs, this interface might be renamed to endx, depending on the system's naming rules.

9.1.1.4 Kernel Optimization

By default, the release package does not optimize the kernel. If you want to optimize it, you can enable kernel optimization by setting the appropriate variable in local.conf.

When enabled, this optimization disables unused features and converts certain built-in modules into loadable modules (USD, touchscreen, CANFD,..). This helps reduce kernel size, improve boot time, and free up system resources.

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.

9.1.2 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.

9.1.2.1 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 11. Boot configuration settings

Config	Value if set	Loading	Description
fdtfile	rzg2l-sbc.dtb	rzg2l-sbc.dtb	Main device tree file to be loaded from the filesystem
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.

(1) 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 (Recommended 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@rzg2l-sbc:~# lsblk
   ```

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

   ```
   root@rzg2l-sbc:~# 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@rzg2l-sbc:~# mount /dev/mmcblk0p1 /mnt/boot_part
   ```

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

   ```
   root@rzg2l-sbc:~# 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@rzg2l-sbc:/mnt/boot_part# sync
   ```

6. Umount the partition: Umount the partition before rebooting

   ```
   root@rzg2l-sbc:/mnt/boot_part# cd ~
   root@rzg2l-sbc:~# umount /mnt/boot_part
   root@rzg2l-sbc:~# sync
   root@rzg2l-sbc:~# reboot
   ```

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

9.1.2.2 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 12. 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^2C3 SDA	46	2	**3**	**4**			5V	
371	I^2C3 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^2C0 SDA			**27**	**28**			I^2C0 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**

(1) 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.).

(2) 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@rzg2l-sbc:# gpiodetect
```

Output will be similar to

```
root@rzg2l-sbc:# gpiochip0 [chip_name_0] (XX lines)
root@rzg2l-sbc:# gpiochip1 [chip_name_1] (YY lines)
# ... additional chips
```

(3) Inspect Lines on a Specific GPIO Chip:

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

```
root@rzg2l-sbc:# 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.

9.1.2.3 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@rzg2l-sbc:~# 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@rzg2l-sbc:~#
```

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

```
root@rzg2l-sbc:~# 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.

9.1.2.4 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@rzg2l-sbc:~# 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
```

9.1.2.5 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@rzg2l-sbc:~# 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@rzg2l-sbc:~# ip link set can0 down
root@rzg2l-sbc:~# ip link set can0 type can bitrate 500000
root@rzg2l-sbc:~# ip link set can0 up
[   48.120419] IPv6: ADDRCONF(NETDEV_CHANGE): can0: link becomes ready
root@rzg2l-sbc:~# ip link set can1 down
root@rzg2l-sbc:~# ip link set can1 type can bitrate 500000
root@rzg2l-sbc:~# ip link set can1 up
[   69.906039] IPv6: ADDRCONF(NETDEV_CHANGE): can1: link becomes ready
root@rzg2l-sbc:~# candump can0 & cansend can1 123#01020304050607
[1] 271
  can0  123   [7]  01 02 03 04 05 06 07
root@rzg2l-sbc:~# 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@rzg2l-sbc:~#
```

9.1.3 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.

9.1.3.1 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:

9.1.3.2 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@rzg2l-sbc:~# apt-get update
root@rzg2l-sbc:~# 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@rzg2l-sbc:~# mkdir -p /mnt/boot_partition
root@rzg2l-sbc:~# mount /dev/mmcblk0p1 /mnt/boot_partition/
```

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

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

6. Sync and umount the partition

```
root@rzg2l-sbc:~# cd ~
root@rzg2l-sbc:~# sync
root@rzg2l-sbc:~# umount /mnt/boot_partition
root@rzg2l-sbc:~# 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.

9.1.3.3 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@rzg2l-sbc:~# cd /sys/class/pwm/pwmchip0/
root@rzg2l-sbc:/sys/class/pwm/pwmchip0# echo 0 > export
```

9.1.3.4 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@rzg2l-sbc:/sys/class/pwm/pwmchip0/# cd pwm0
root@rzg2l-sbc:/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.

9.1.4 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@rzg2l-sbc:~# 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@rzg2l-sbc:~# 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@rzg2l-sbc:~# ifconfig end0 down

root@rzg2l-sbc:~# ifconfig end1 down
```

9.1.4.1 Generic USB Bluetooth Framework

The RZG2L-SBC supports the generic USB Bluetooth framework, which is backported 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 RZG2L-SBC. This will ensure it is loaded each time the board boots (one-time setup).

```
root@rzg2l-sbc:~# mkdir -p /lib/firmware/rtl_bt
root@rzg2l-sbc:~# 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@rzg2l-sbc:~# 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:

```
root@rzg2l-sbc:~# 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: 'rzg2l-sbc'
        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:

```
root@rzg2l-sbc:~# bluetoothctl

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

Set the RZG2L-SBC 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 RZG2L-SBC and the target Bluetooth device, run the obexctl daemon and connect:

```
root@rzg2l-sbc:~# export $(dbus-launch)

root@rzg2l-sbc:~# /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
```

9.1.5 Onboard Audio Codec with Stereo Jack

The RZ/G2L-SBC features an onboard audio codec, Renesas DA7219, enabling audio playback and recording through a 3.5mm stereo jack (connector J8, 6-pin).

- Audio Data Interface: Connected to DAI (SSI1) using the I2S format.
- Control Interface: Managed via I2C0.
- Headset Jack: Marked J8 on the board.

Audio playback and recording are supported through ALSA tools with PCM WAV files. For other formats such as MP3, the pre-installed GStreamer framework provides compatibility.

Prepare the required audio files and place them in the target directory before executing the following commands. Example commands are shown below:

```
root@rzg2l-sbc:~# aplay
/home/root/audios/04_16KH_2ch_bgm_maoudamashii_healing01.wav
root@rzg2l-sbc:~# gst-play-1.0 /home/root/audios/COMMON6_MPEG2_L3_24KHZ_160_2.mp3
```

`aplay` command supports only `wav` format audio files.

`gst-play-1.0` command supports `wav`, `mp3`, and `aac` formats.

The following shows commands to record audio.

```
root@rzg2l-sbc:~# arecord -f S16_LE -r 48000 audio_capture.wav
```

Press Ctrl+C if you want to stop recording.

In the above command:

-f S16_LE : audio format (signed 16-bit little endian)

-r 48000 : sample rate of the audio file (48KHz)

To verify the recorded file, you can play it with the following command:

```
root@rzg2l-sbc:~# aplay audio_capture.wav
```

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

```
root@rzg2l-sbc:~# alsamixer
```

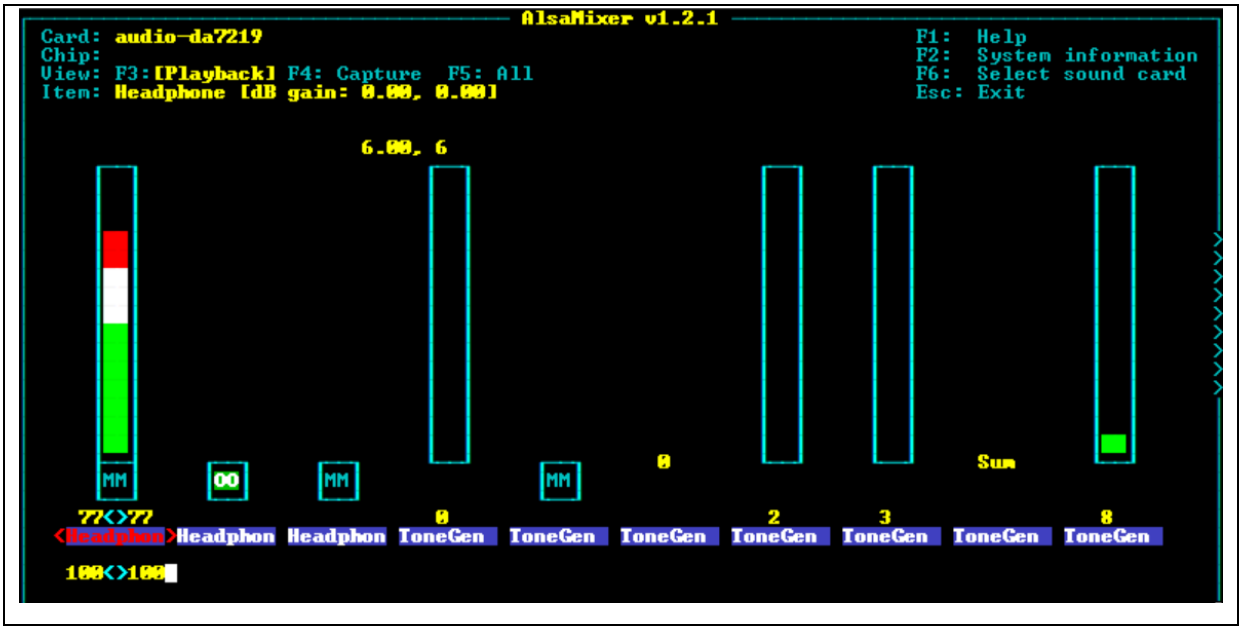


Figure 14. ALSA Mixer GUI on RZ/G2L-SBC

9.1.6 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.

9.1.6.1 Hardware Interfacing

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

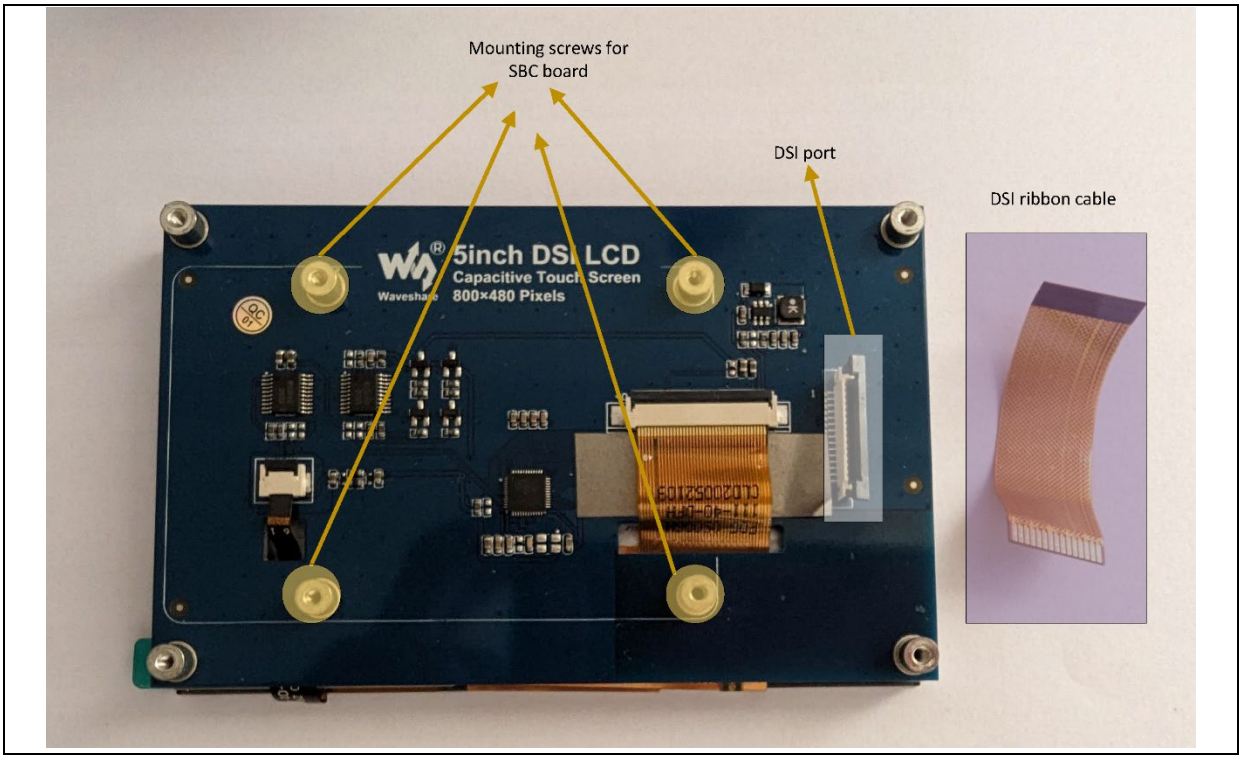


Figure 15. Waveshare 5" DSI touch panel read side picture with flat ribbon cable.

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.

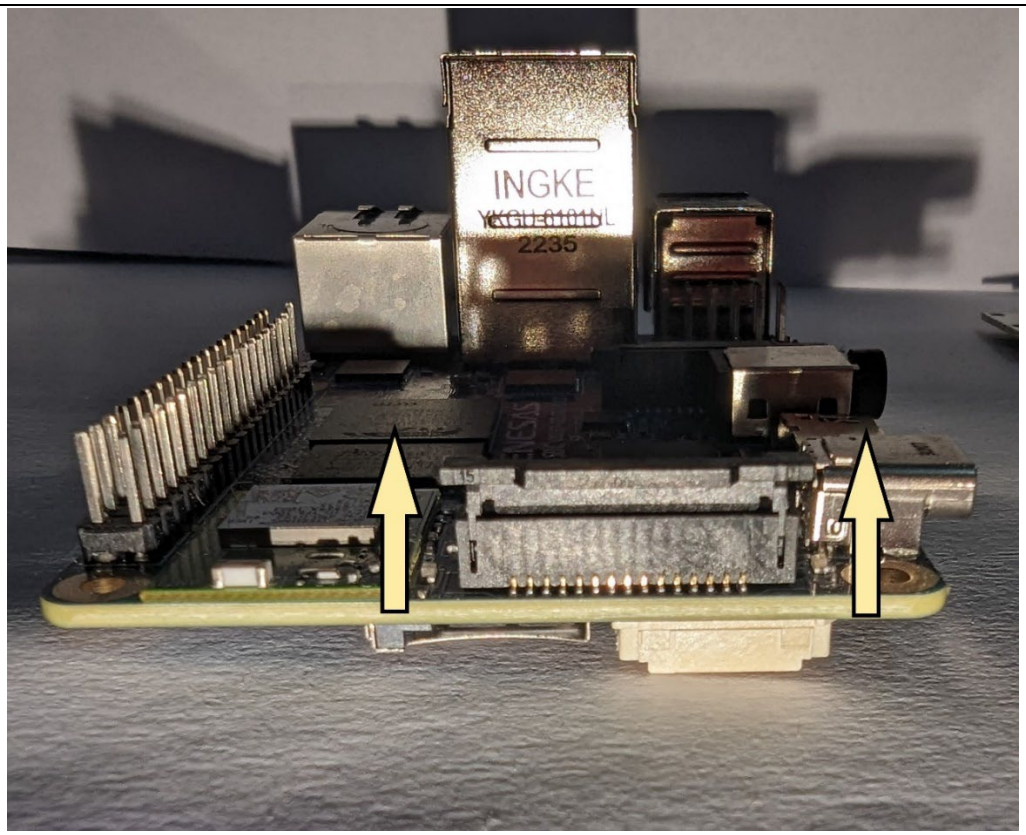


Figure 16. DSI port notch lock open by pulling it up

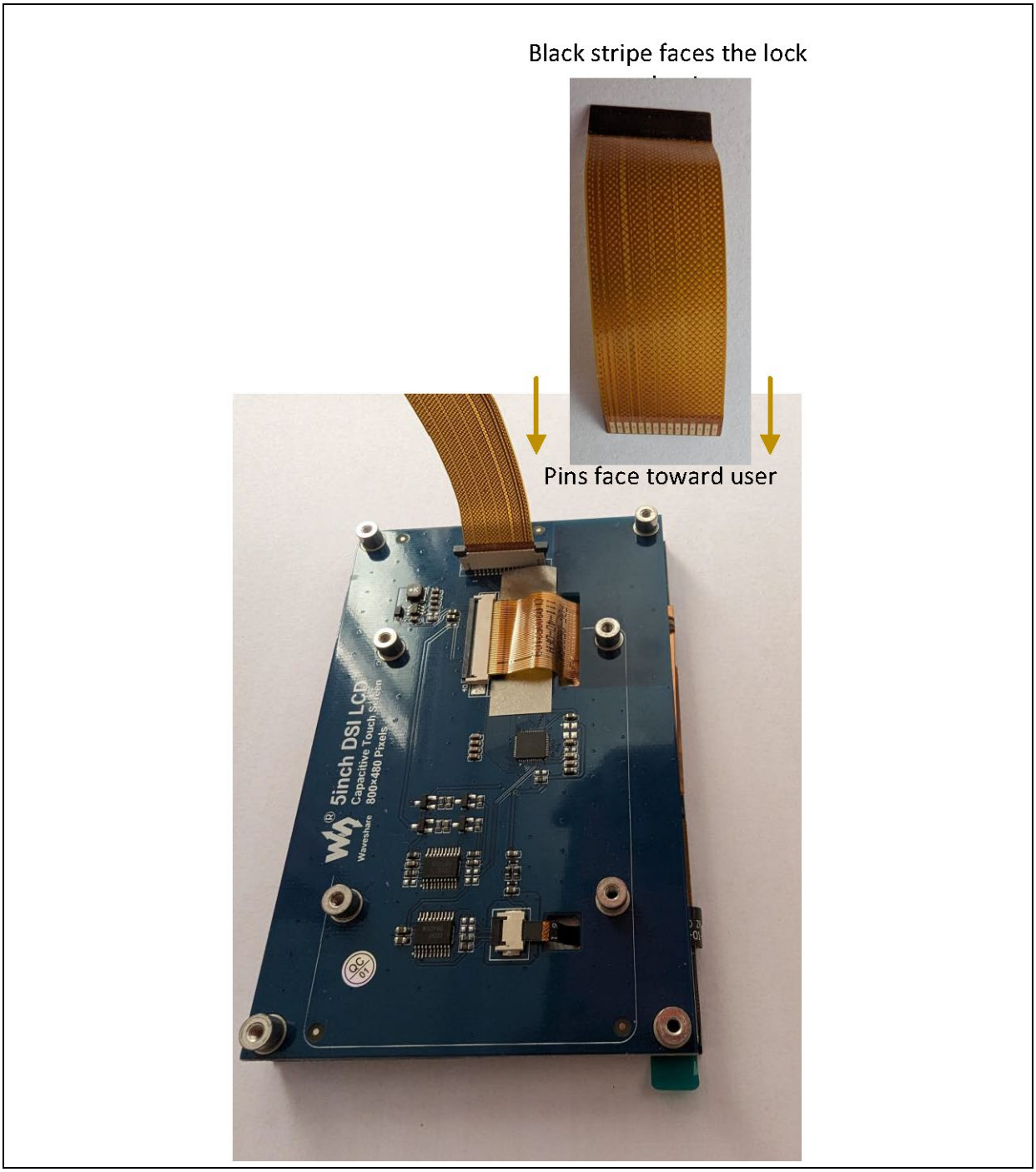


Figure 17. Waveshare DSI touch display DSI port interfacing cable orientation.

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

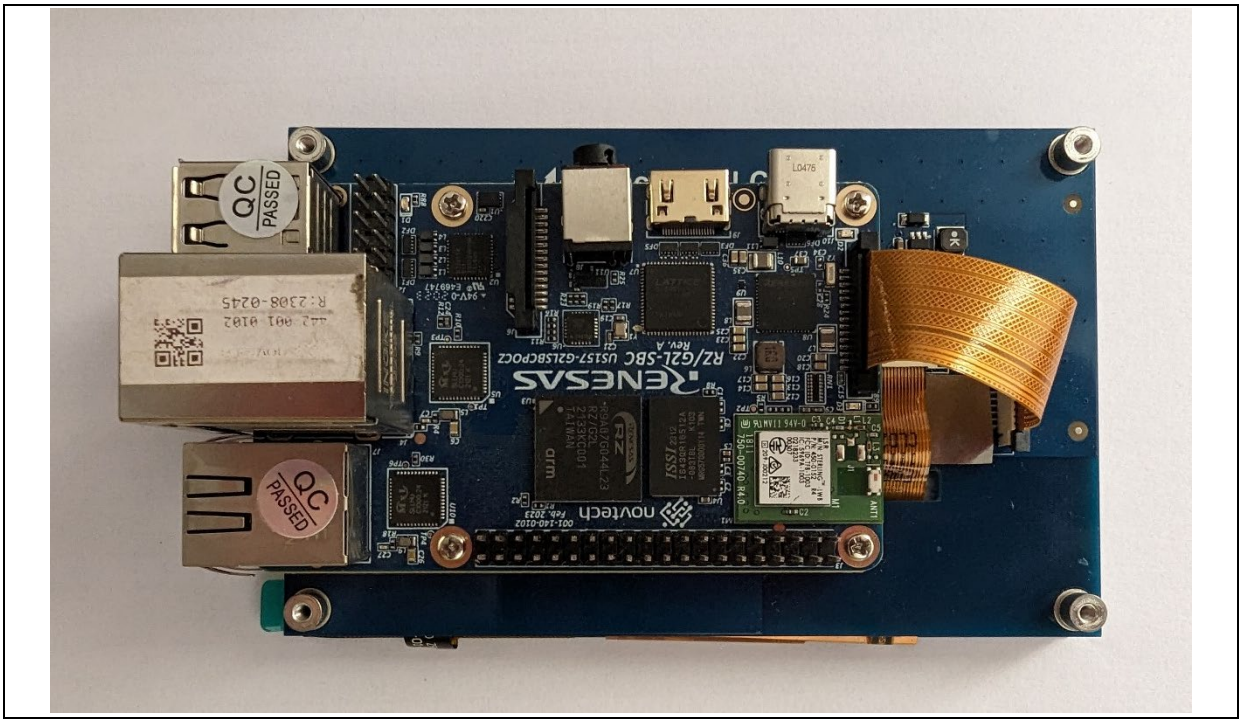


Figure 18. RZ/G2L-SBC mounted to the Waveshare DSI panel and interfaced.

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

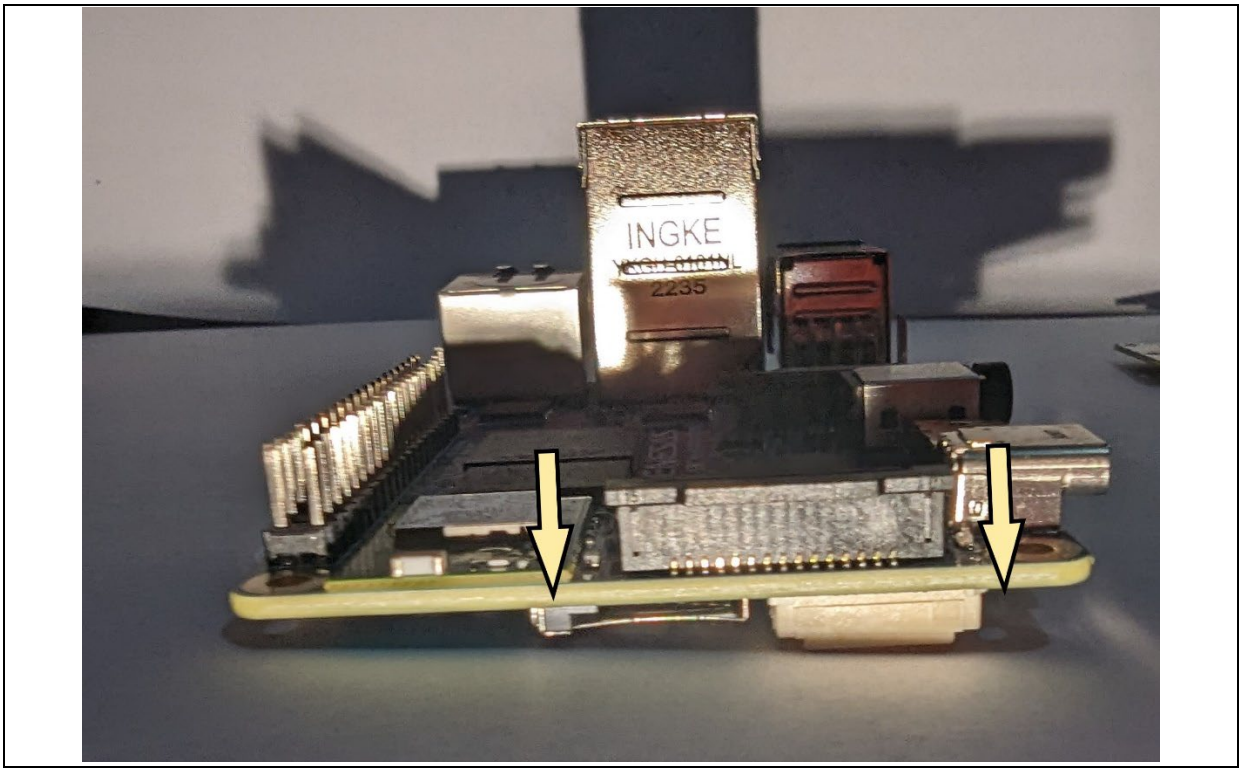


Figure 19. DSI port notch in the lock position. The cable is not shown to keep the notch in clear view.

Figure 20. 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 I²C 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.

9.1.6.2 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.

9.1.7 Playing Video Files on RZ/G2L-SBC

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@rzg2l-sbc:~# gst-launch-1.0 playbin uri=file:///<path/to/your/video/path>
```

For example,

```
root@rzpi:~# 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

9.1.8 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.

9.1.8.1 Hardware Interfacing

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

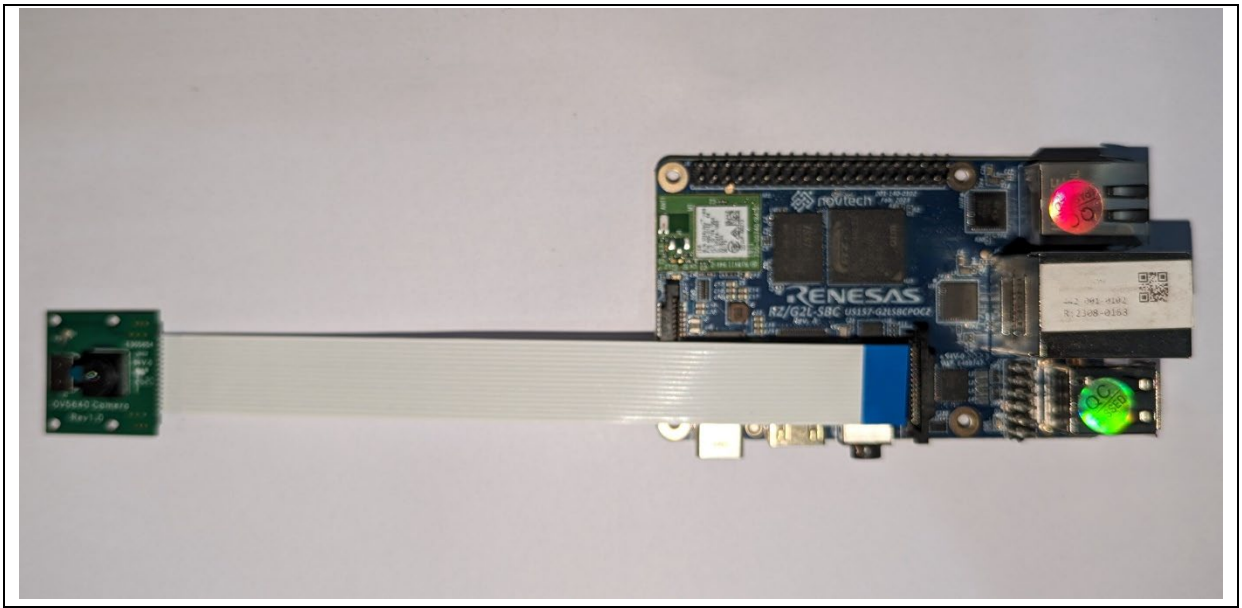


Figure 22. Orientation of the camera module. Blue stripe upward.

The black notch must be pulled up to unlock it.

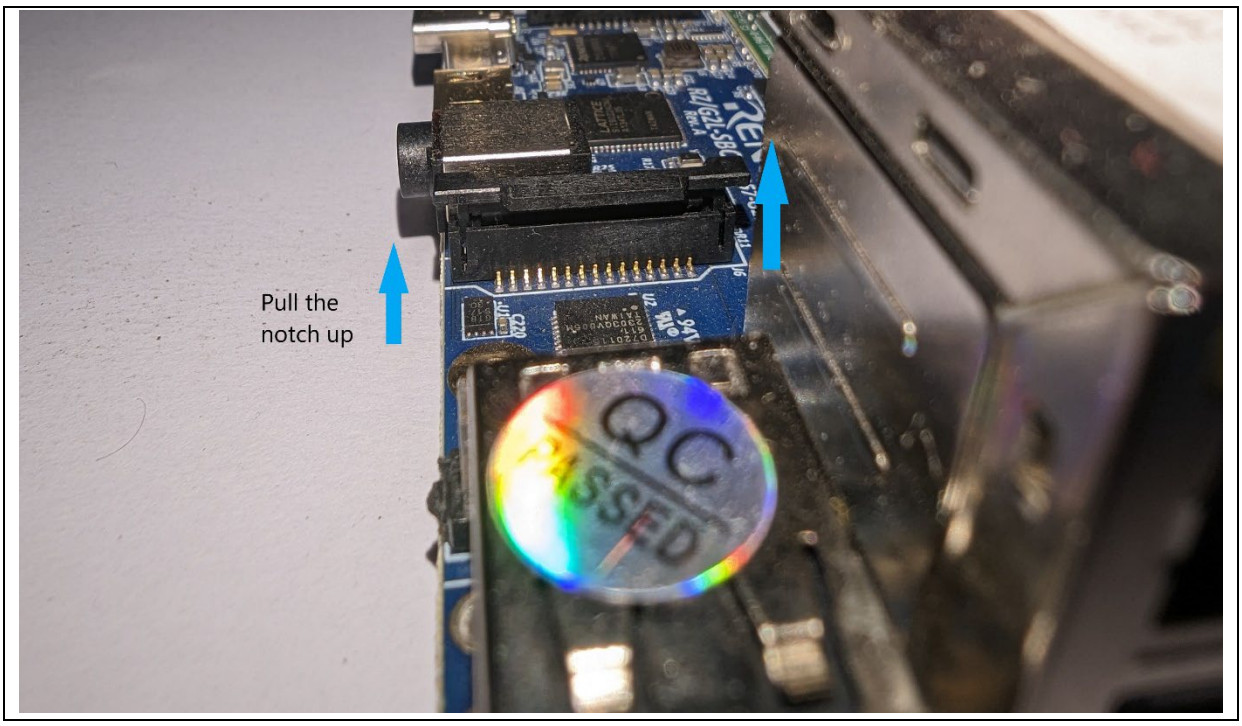


Figure 23. Pull the notch up to unlock it.

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

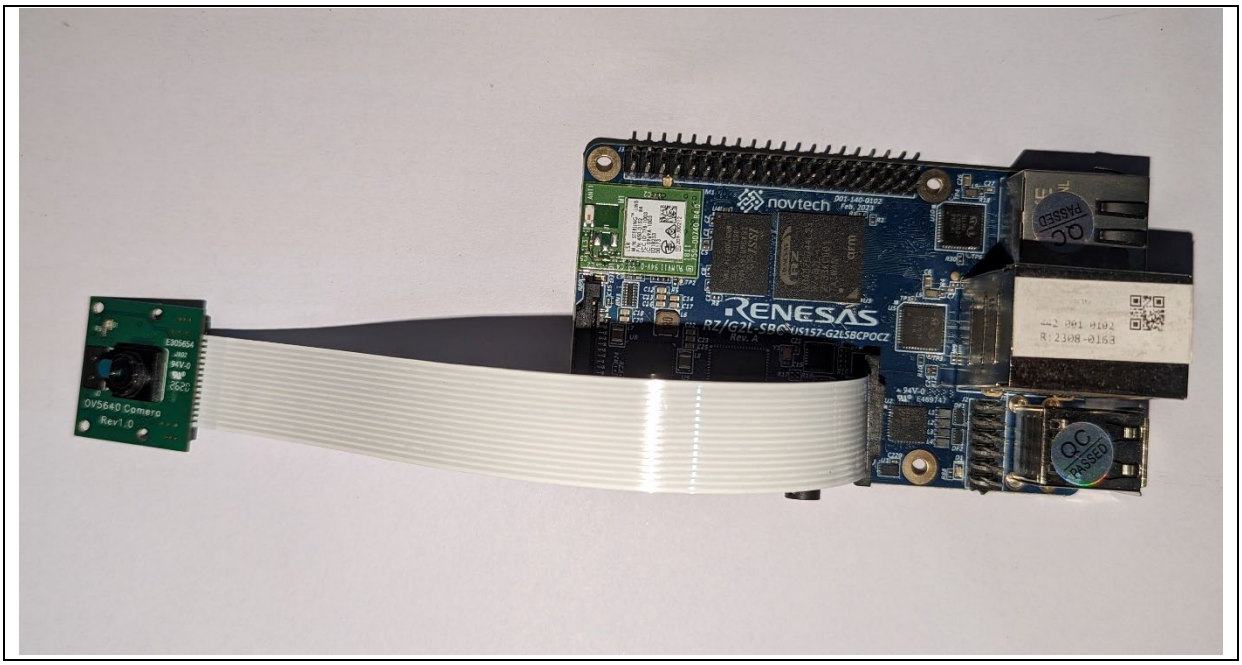


Figure 24. The CSI module is inserted.

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

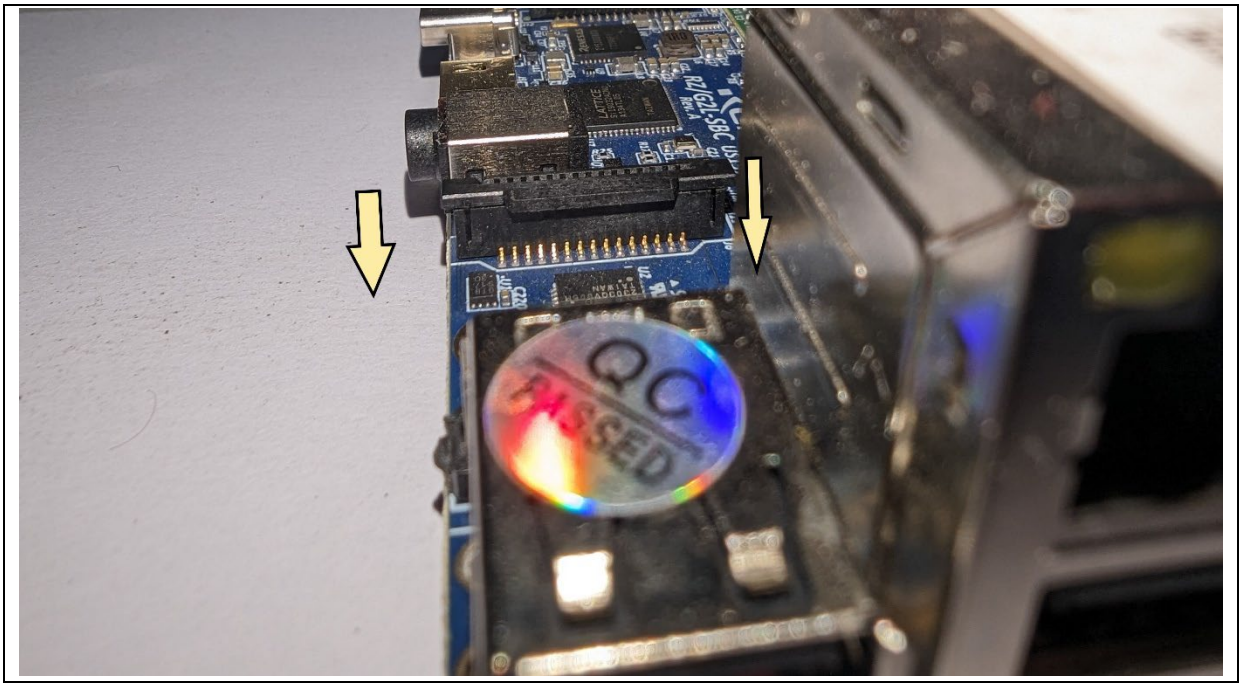


Figure 25. Push down the notch to lock it when you have inserted the flat cable

9.1.8.2 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.

9.1.8.3 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@rzg2l-sbc:~# cd /home/root/
root@rzg2l-sbc:~# ./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@rzg2l-sbc:~# ./v4l2-init.sh 1920x1080
Link CRU/CSI2 to ov5640 1-003c with format UYVY8_1X16 and resolution 1920x1080
```

When no resolution is specified:

```
root@rzg2l-sbc:~# ./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@rzg2l-sbc:~# 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.

9.1.9 Package Management

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

9.1.9.1 Setting Up Debian as A Backend Source

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@rzg2l-sbc:~# 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@rzg2l-sbc:~# 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 that entirely hosts 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@rzg2l-sbc:~# 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.

9.1.9.2 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@rzg2l-sbc:~# apt-get update
root@rzg2l-sbc:~# 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@rzg2l-sbc:~# update-alternatives --set iptables /usr/sbin/iptables-legacy
root@rzg2l-sbc:~# update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy
```

Restart Docker to apply these changes:

```
root@rzg2l-sbc:~# systemctl restart docker
```

Step 4: Verify Docker Installation by running:

```
root@rzg2l-sbc:~# docker run hello-world
```

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

9.1.9.3 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@rzg2l-sbc:~# dpkg -i <package>.deb
```

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

```
root@rzg2l-sbc:~# apt-get install -f
```

9.1.10 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@rzg2l-sbc:~# pip3 install <package_name>
```

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

```
root@rzg2l-sbc:~# pip3 install requests
```

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

```
root@rzg2l-sbc:~# 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@rzg2l-sbc:~# pip3 list
```

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

9.1.11 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@rzg2l-sbc:~# su - weston
```

2. Create a working directory on the RZ/G2L-SBC to develop and store the Python application.

```
rzg2l-sbc:~$ mkdir ~/python_apl
rzg2l-sbc:~$ cd ~/python_apl
```

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

```
rzg2l-sbc:~/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/G2L SBC is connected to an external display. If the display is not set automatically, set the DISPLAY environment variable as follows:
-
```
rzg2l-sbc:~$ export DISPLAY=:0
```

- Run the Python application:

```
rzg2l-sbc:~$ python3 main.py
```

Figure 26. Initial GUI layout

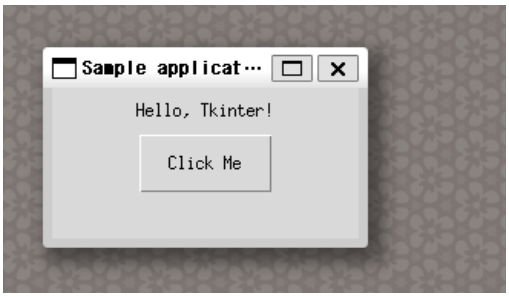


Figure 27. After the button 'Click me' is clicked

9.2 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.

9.2.1 Accessing Supported Features in Ubuntu LXDE

9.2.1.1 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 a 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.

9.2.1.2 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.

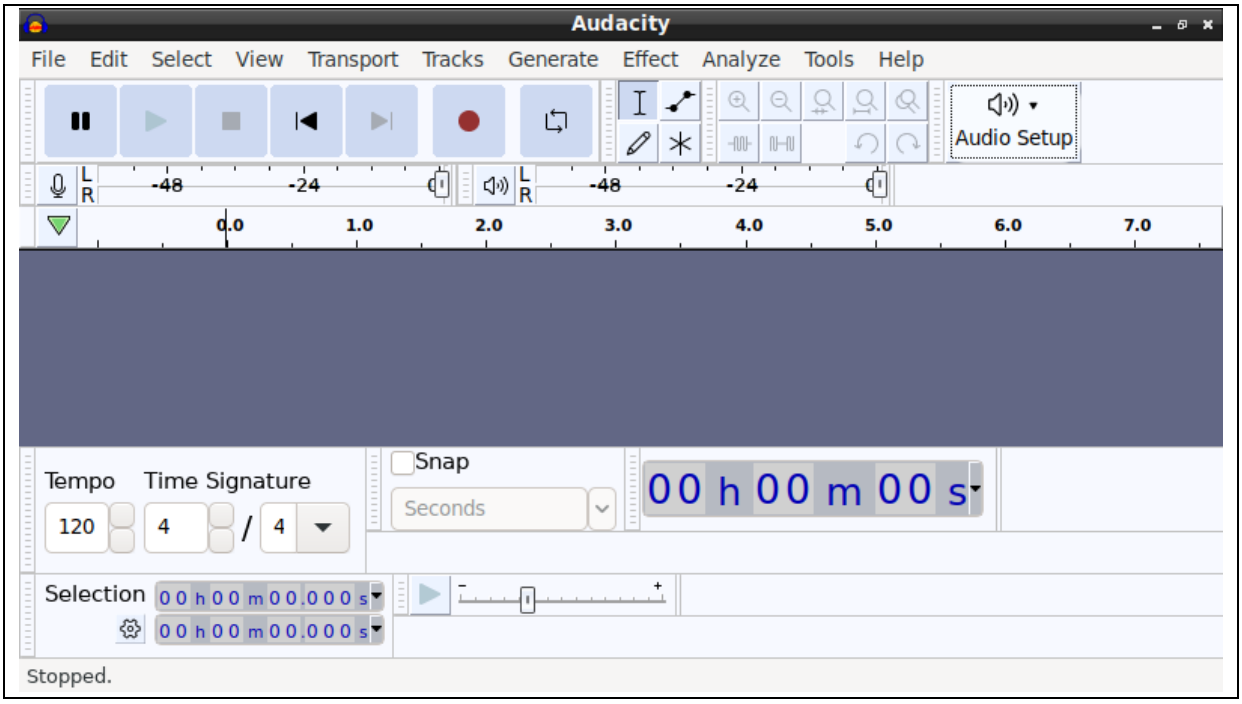


Figure 29. Audacity graphic user interface

To properly configure Audacity for the system:

1. In Audacity, click Audio Setup in the top-right corner, then select Audio Settings.
2. In the window that opens, set both the Playback and Recording Device to audio-da7219.
3. Set the Project Sample Rate and Default Sample Rate at the bottom left to 48000 to match the hardware's requirements.

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.

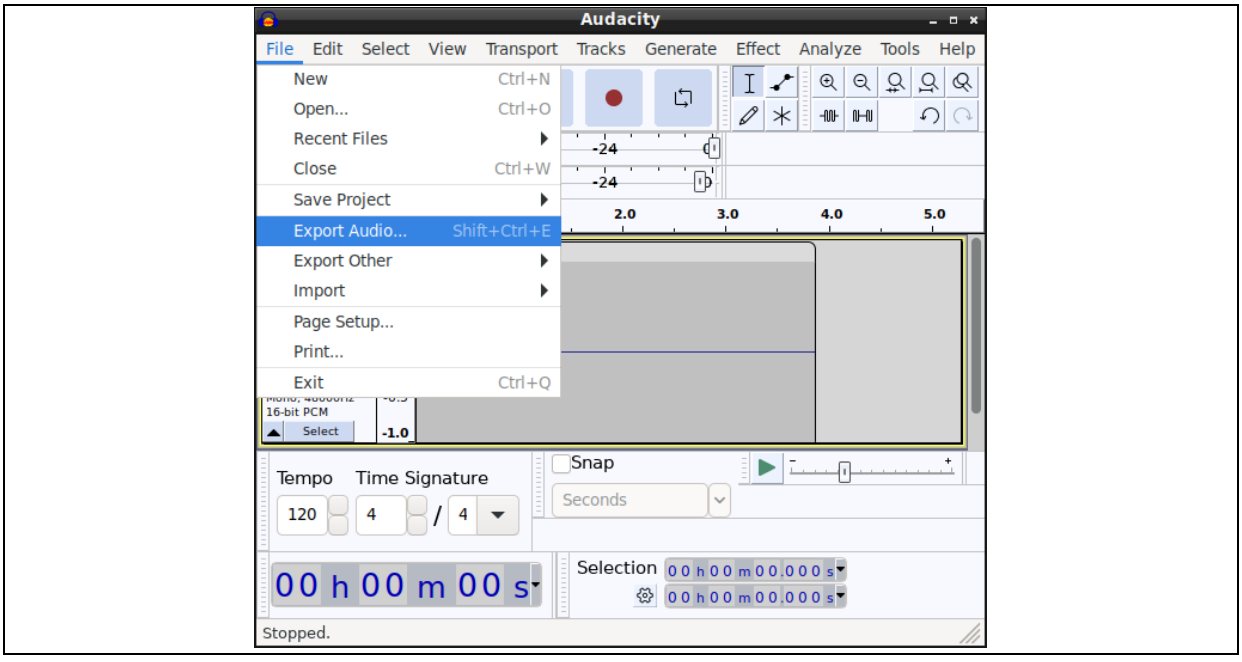


Figure 30. Audacity exports video as MP3

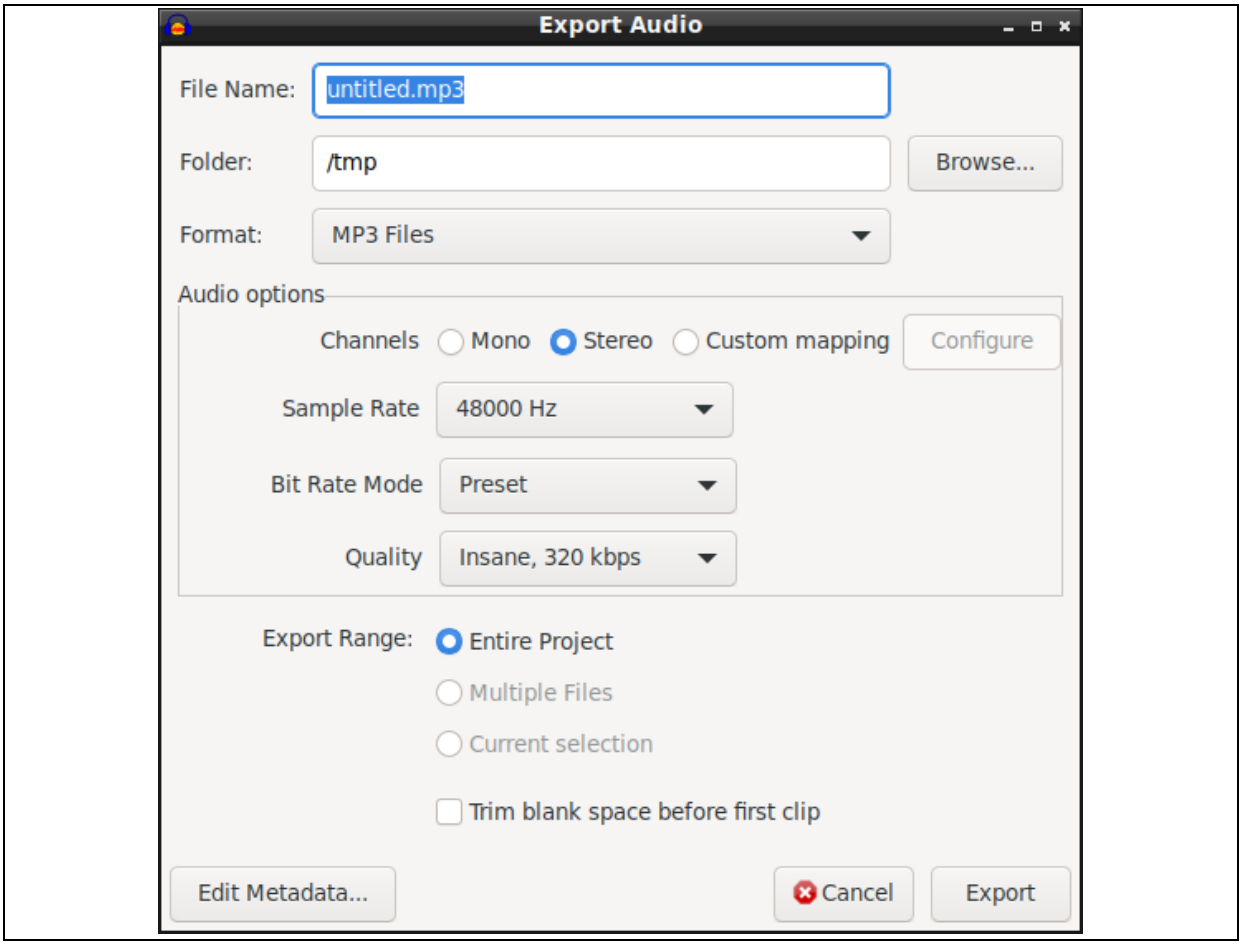


Figure 31. Saving the recorded audio

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

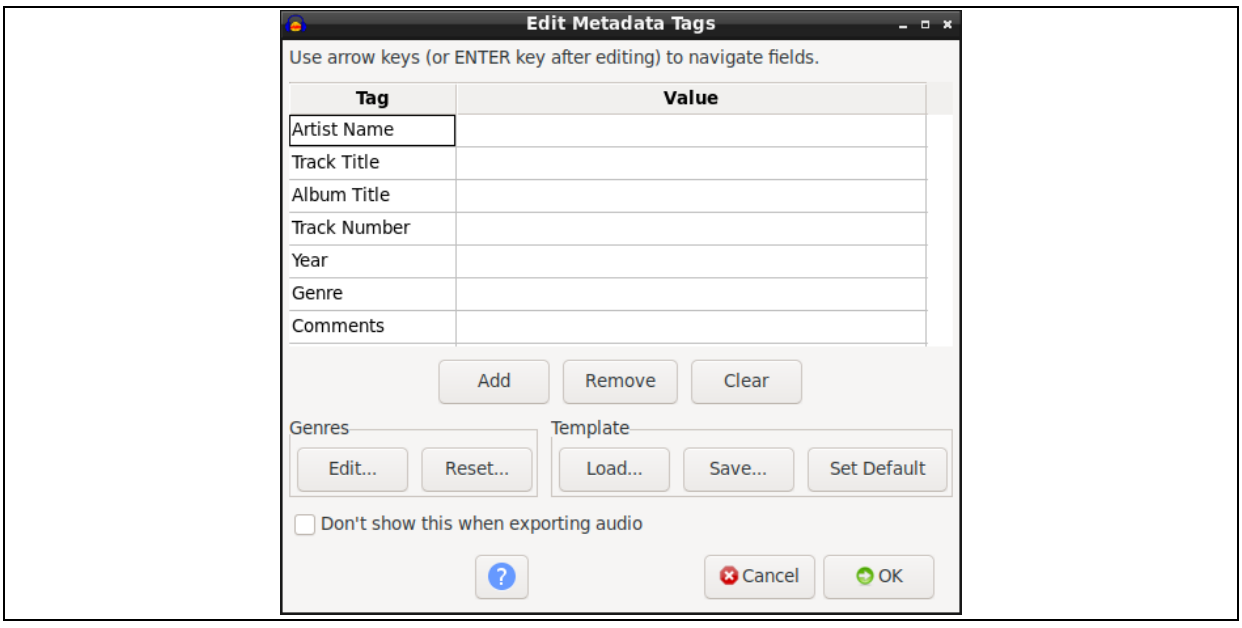


Figure 32. Audacity metadata tags to fill in for the audio

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.

9.2.1.3 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

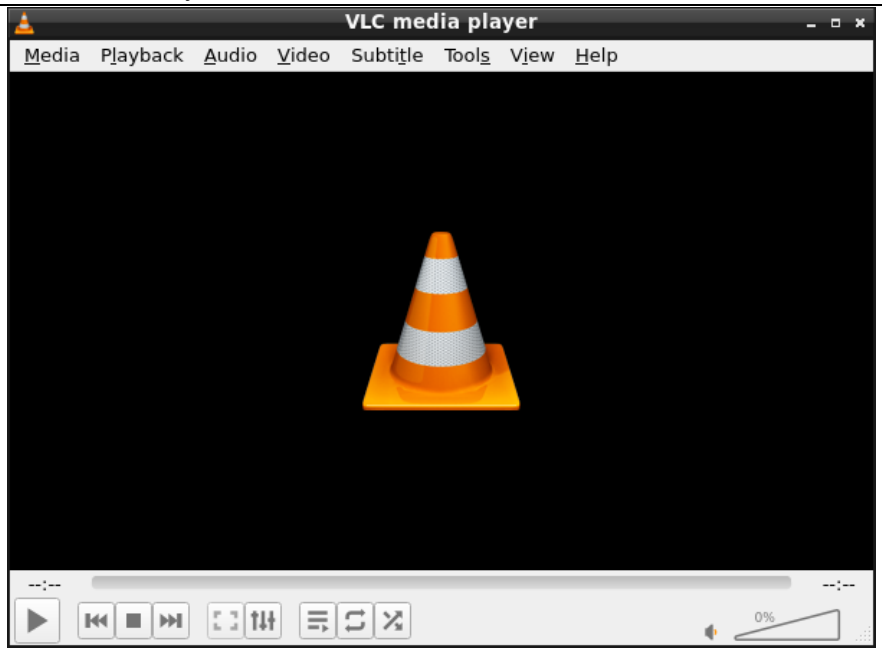


Figure 33. VLC Media Player

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

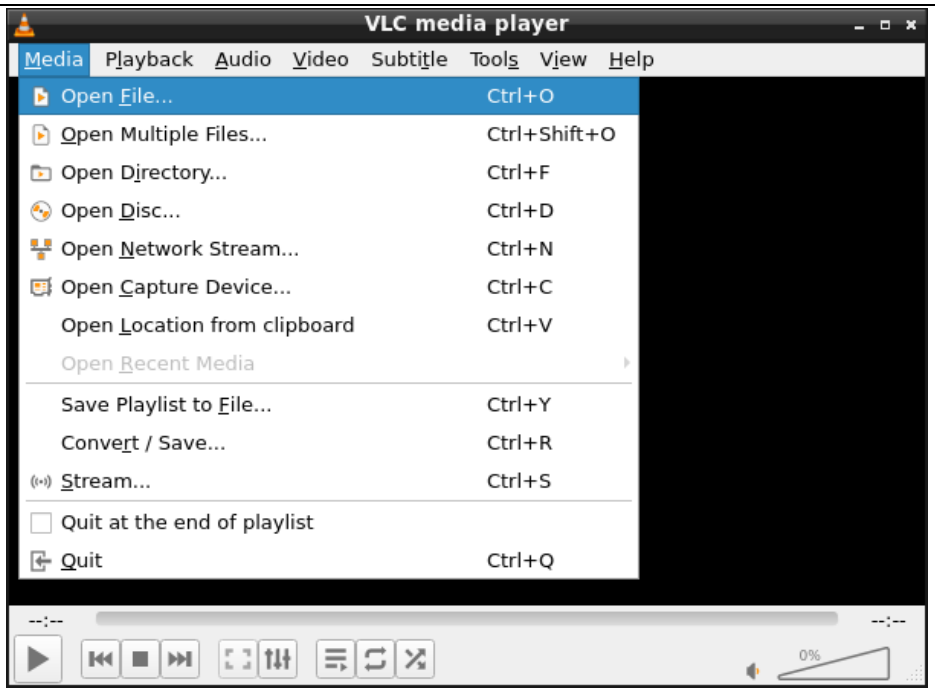


Figure 34. Open file in VLC Media Player

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

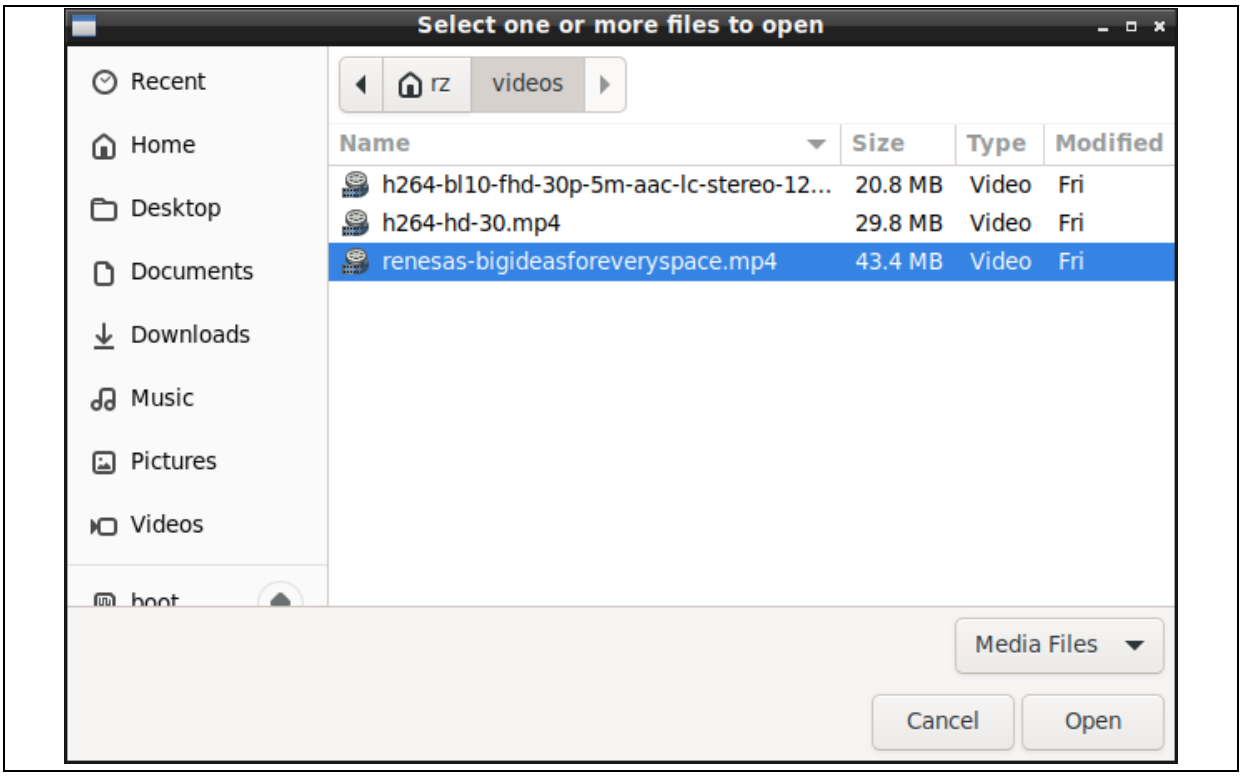


Figure 35. Select MP3/MP4 file to play in VLC Media Player

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

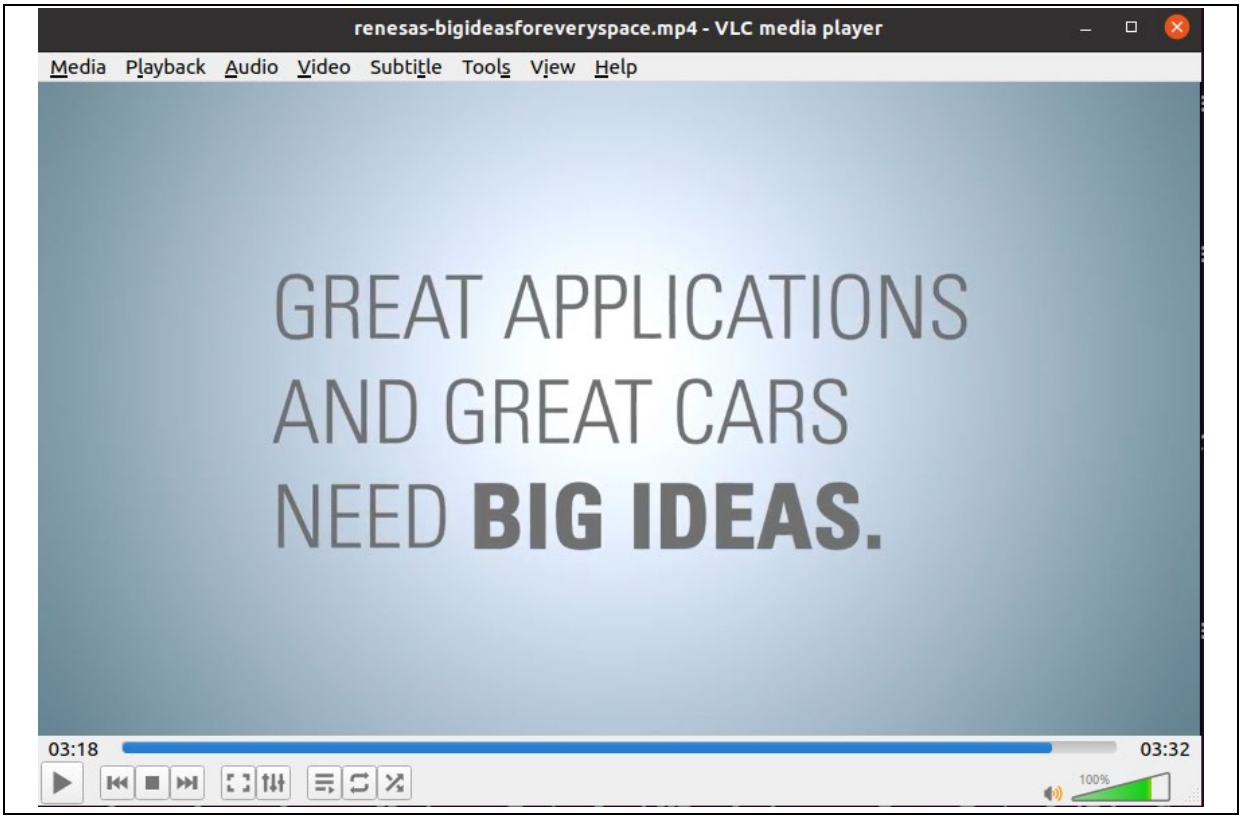


Figure 36. Video playback in VLC Media Player

9.2.1.4 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.

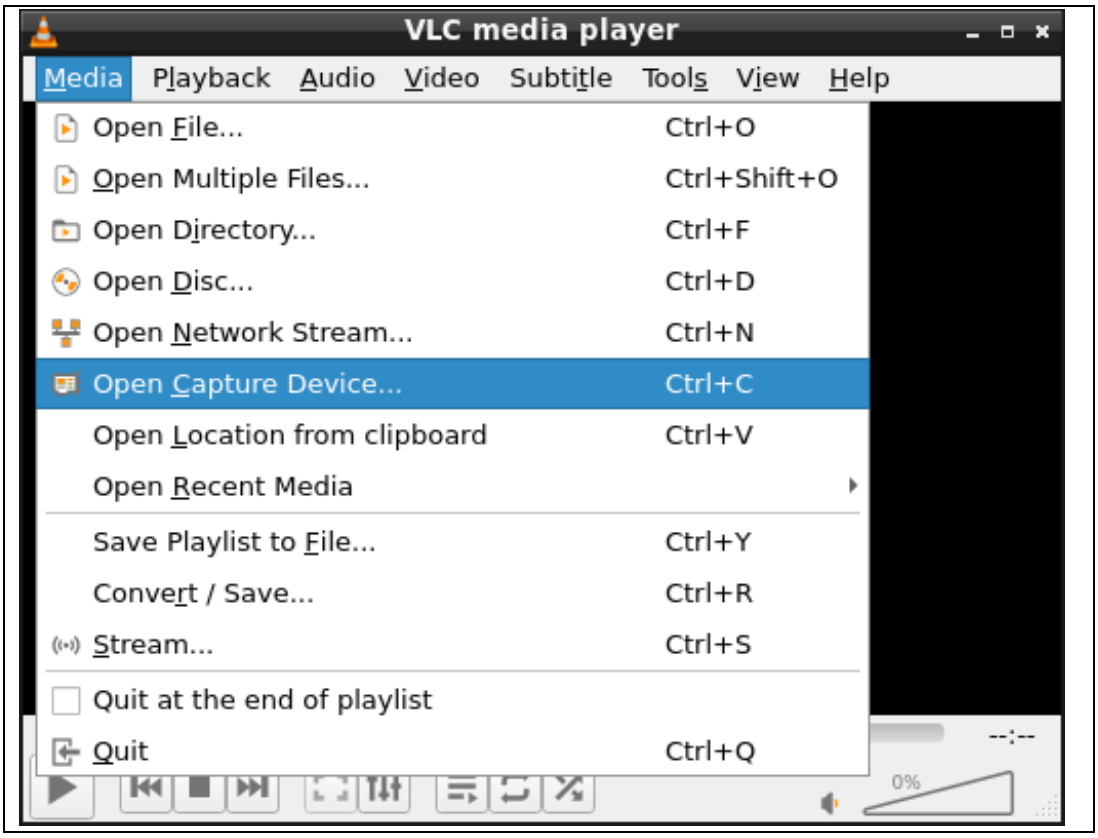


Figure 37. Capture Device in VLC Media Player

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.

9.2.1.5 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.

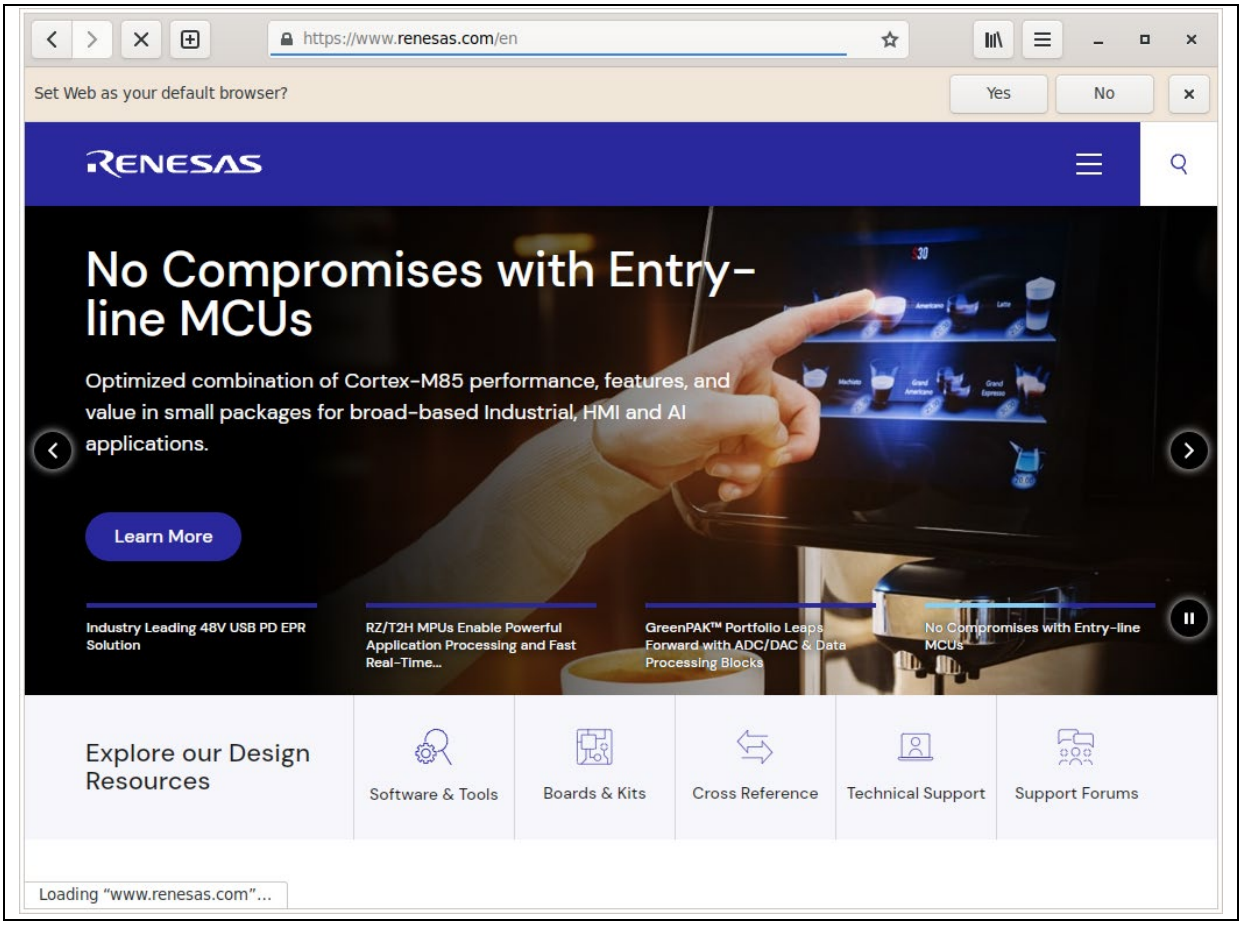


Figure 38. Ubuntu LXDE web browser pre-installed

9.2.1.6 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

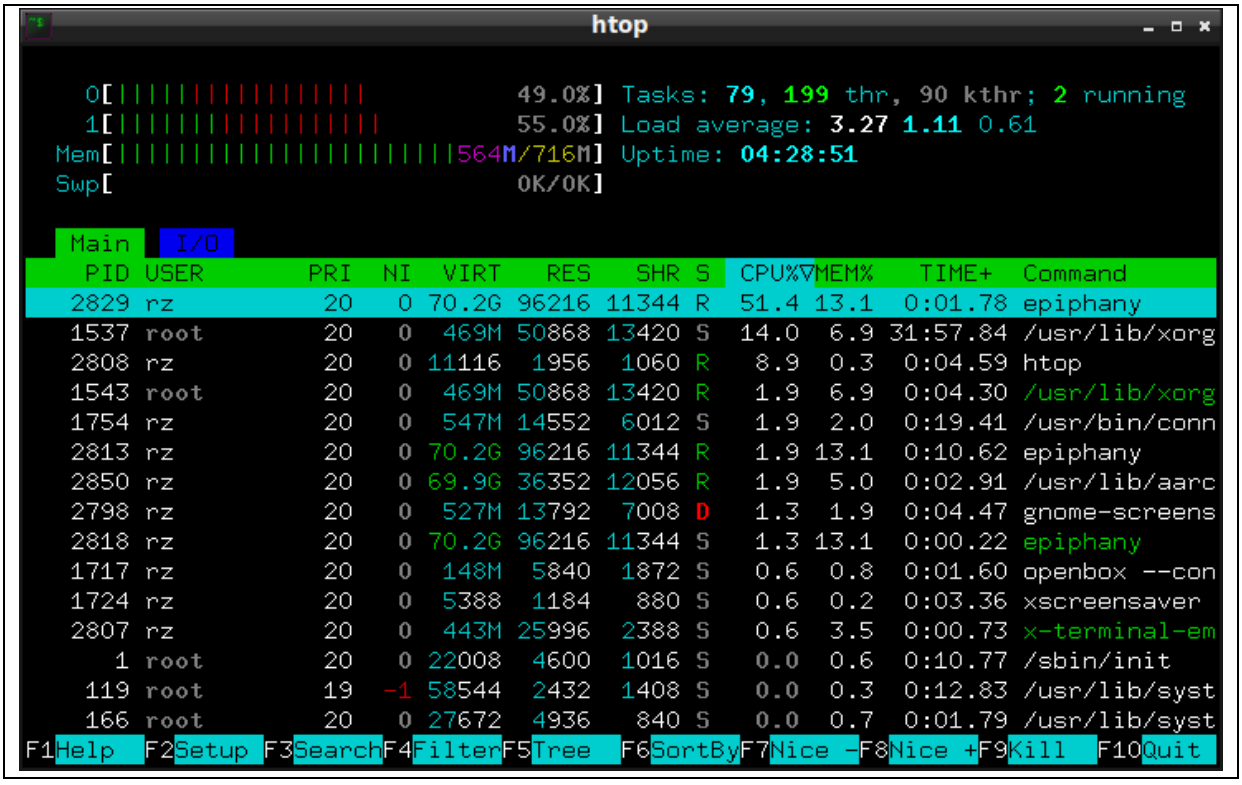


Figure 39. Htop in Ubuntu-LXDE

9.2.1.7 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....

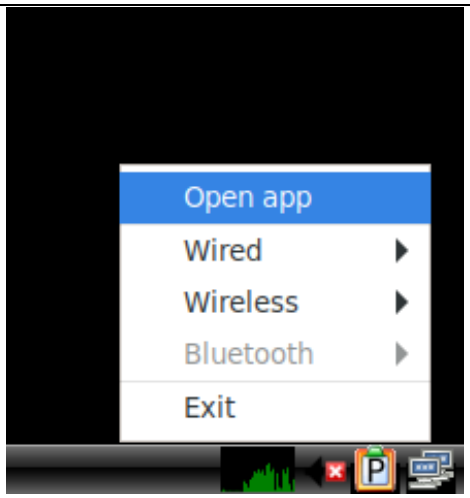


Figure 40. Bluetooth icon Ubuntu-LXDE taskbar

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).

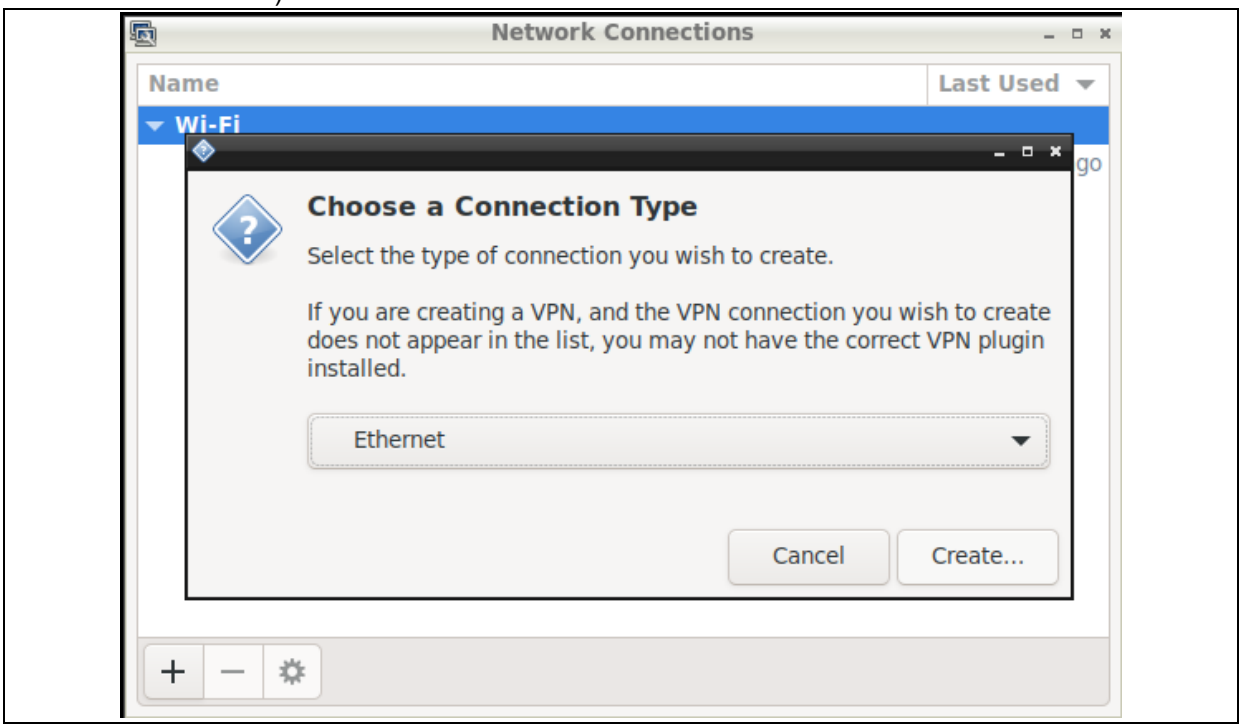


Figure 41. Choose Network Connections

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

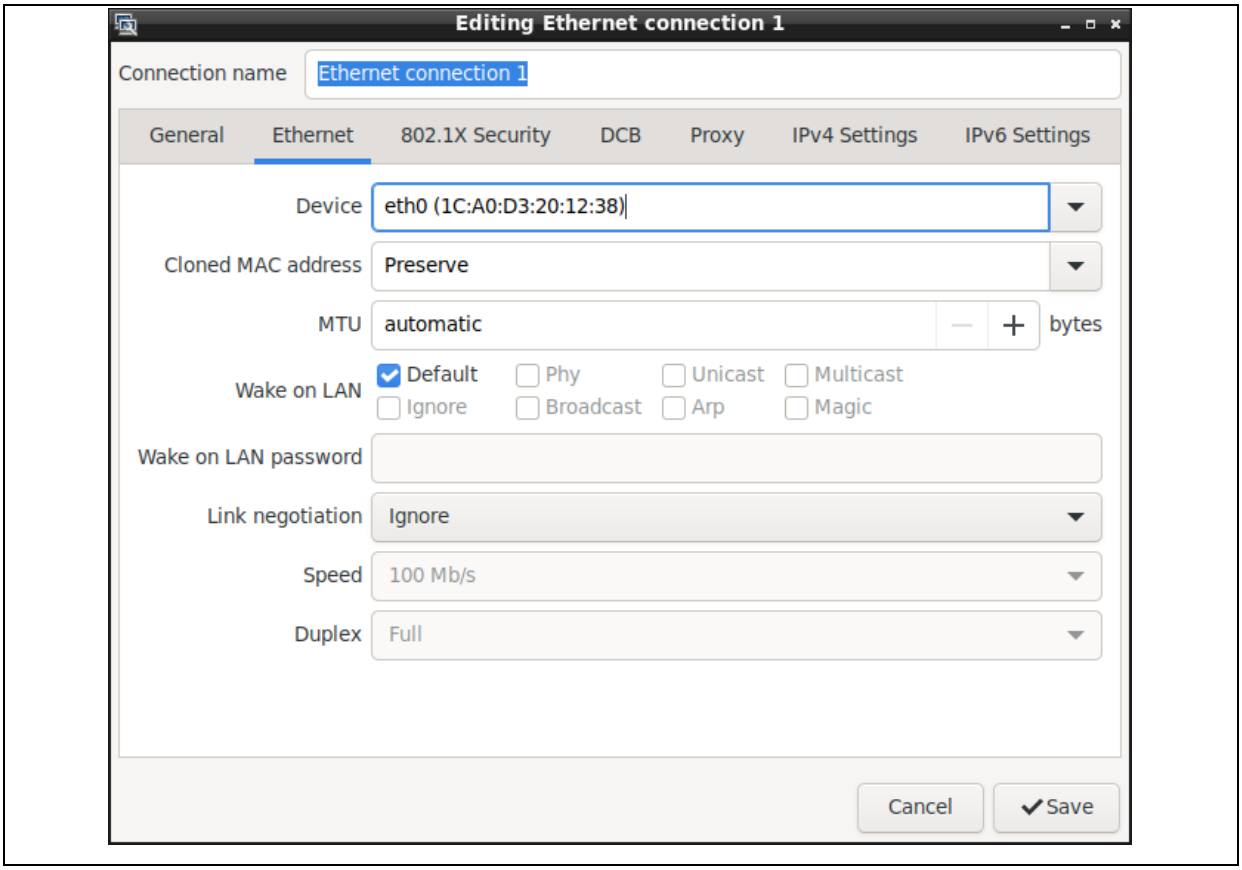


Figure 42. Edit Ethernet connection 1

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.

9.2.1.8 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.

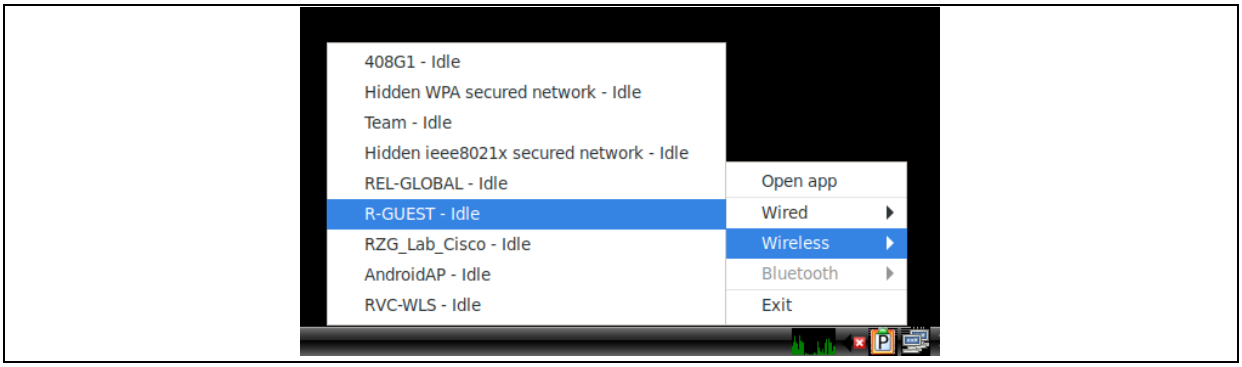


Figure 43. Wi-Fi selection in Ubuntu-LXDE

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.

9.2.1.9 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.

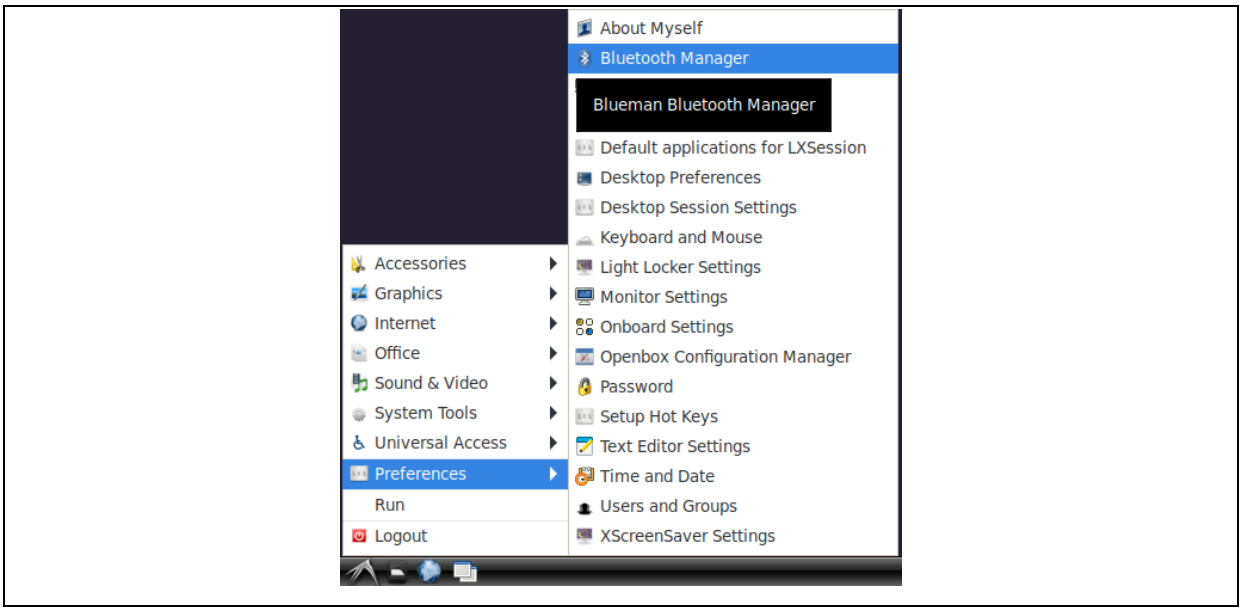


Figure 44. Bluetooth icon in Ubuntu LXDE taskbar

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.

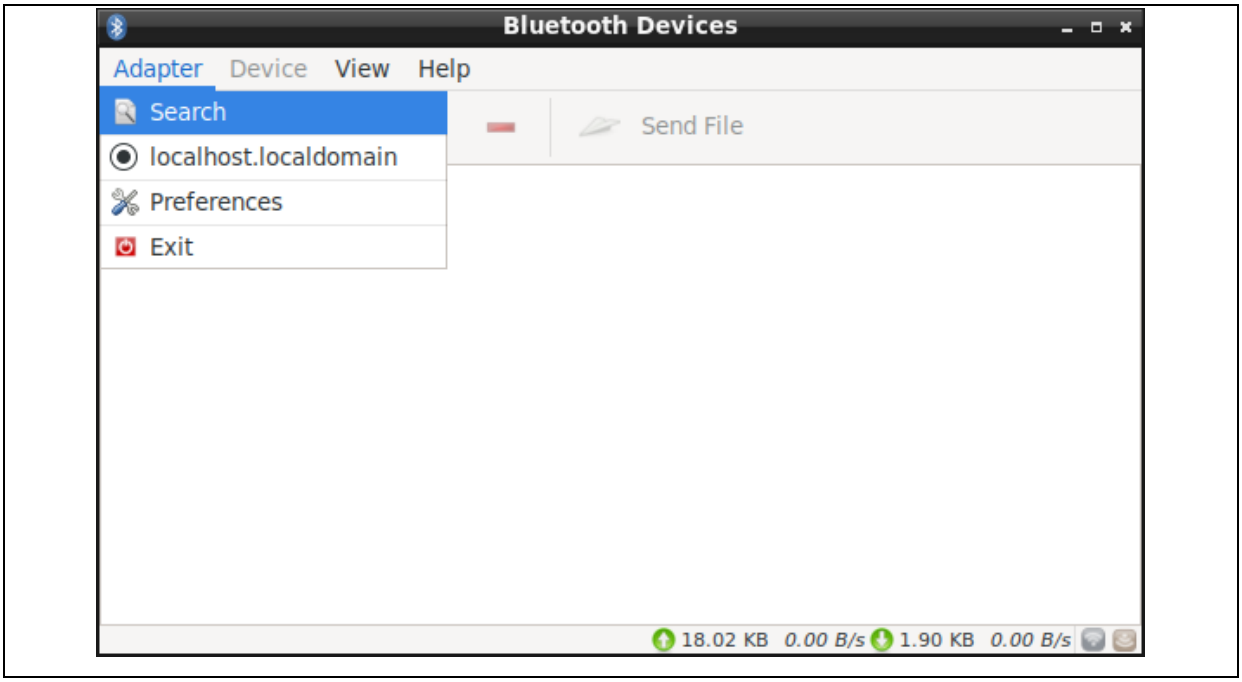


Figure 45. Search for available Bluetooth devices

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

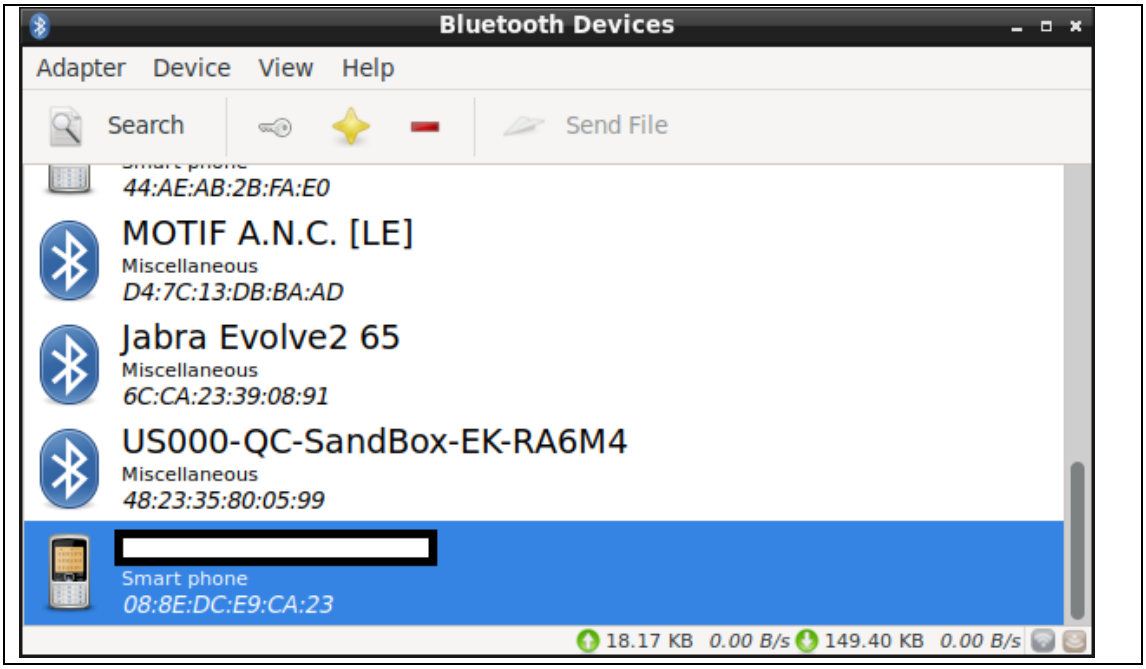


Figure 46. Select a Bluetooth device

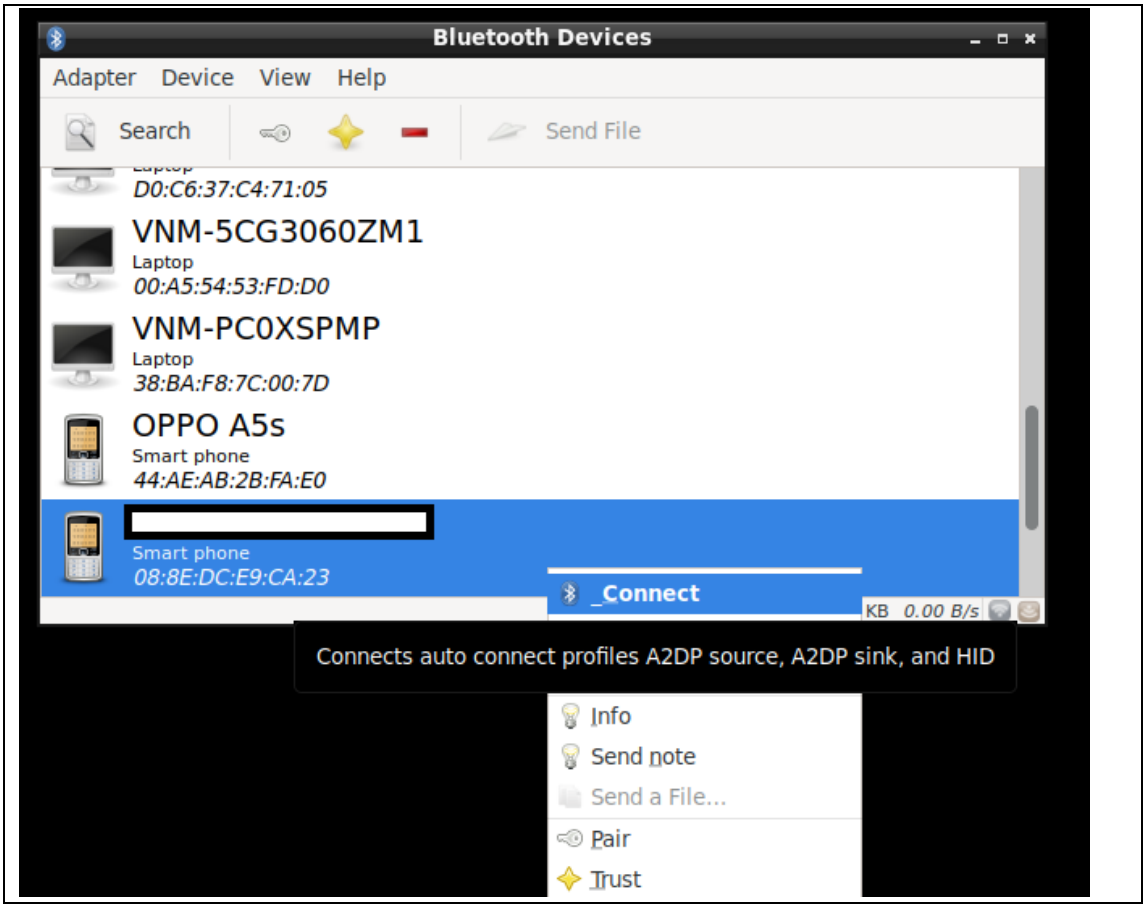


Figure 47. Connect to a Bluetooth device

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.

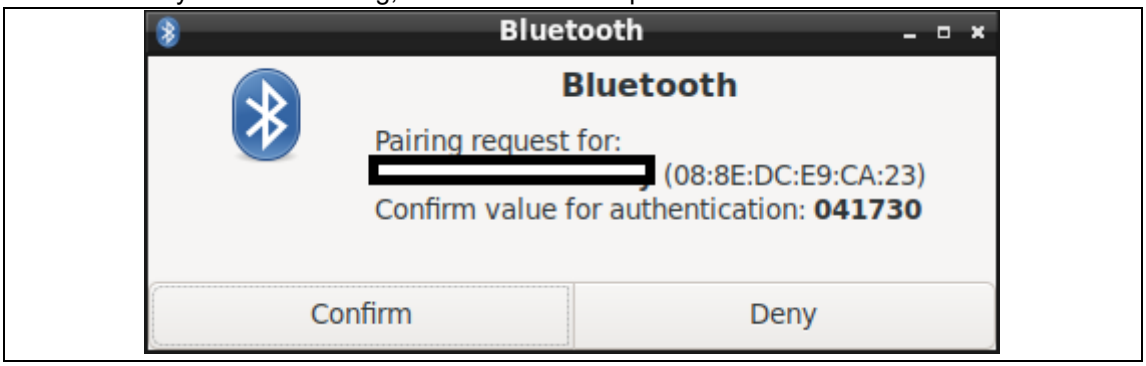


Figure 48. Bluetooth connection confirmation

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

9.2.2 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

9.2.2.1 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 rzpi.example.com rzpi

# 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#
```

10. 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.

10.1 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
- FTDI-based USB-UART cable.
- 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
```

10.2 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
```

10.3 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: RZ/G2L-SBC
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:   eth0: ethernet@11c20000, eth1: ethernet@11c30000
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
```

10.4 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 49. TFTP boot setup:

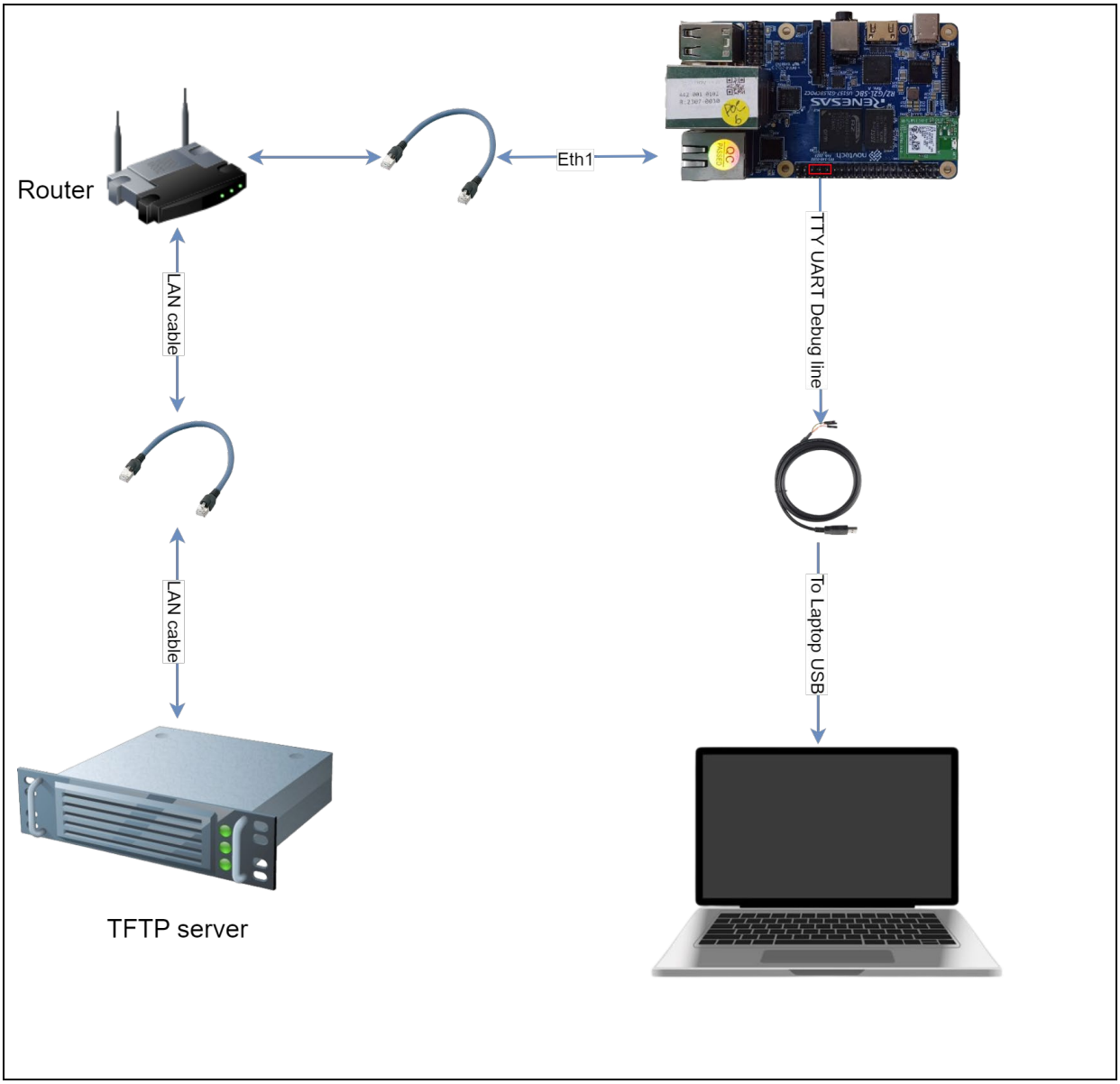


Figure 49. TFTP boot setup

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/rzsbc/$ 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/rzsbc/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 rzsbc/Image; tftp 0x48000000 rzsbc/rzg2l-
sbc.dtb; tftp 0x48010000 rzsbc/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 rzsbc/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 'rzsbc/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 'rzsbc/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 ...
...
```

11. 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.

11.1 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.

11.2 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
```

11.3 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.

11.3.1 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

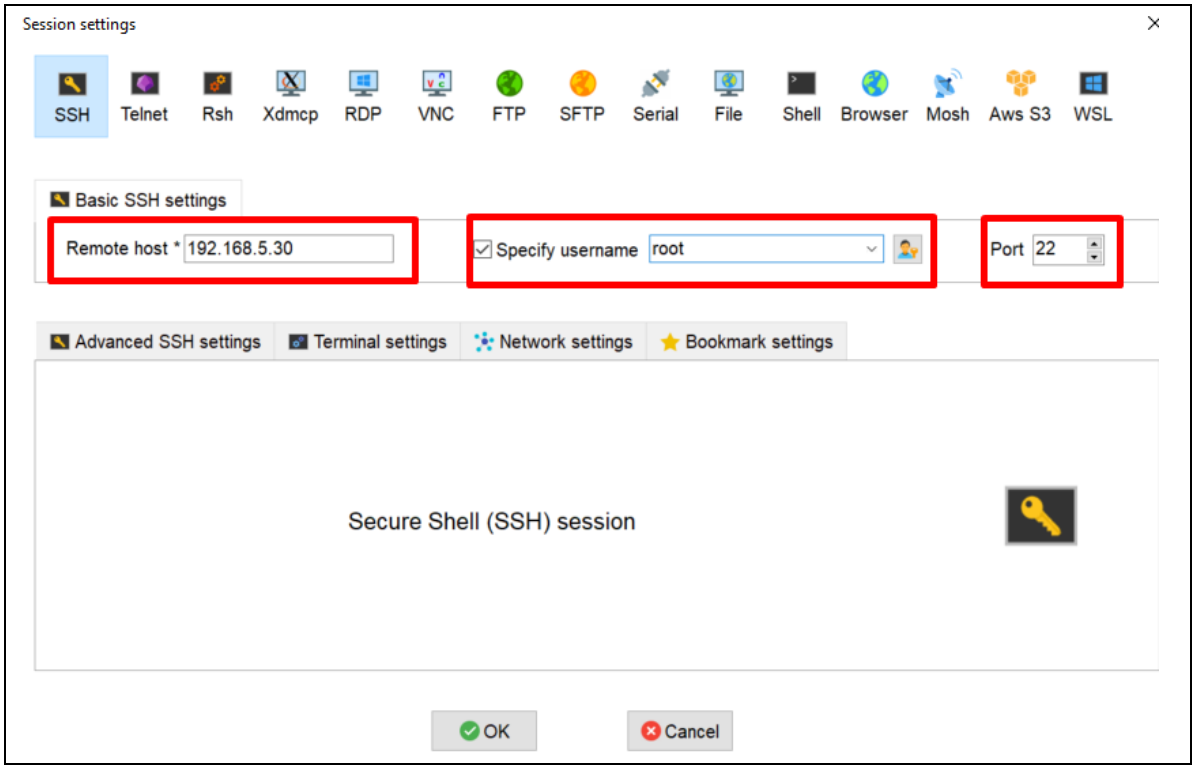


Figure 50. SSH settings in Mobaxterm

- Click 'OK' to save the setting.
- Click on the session to initiate an SSH connection.

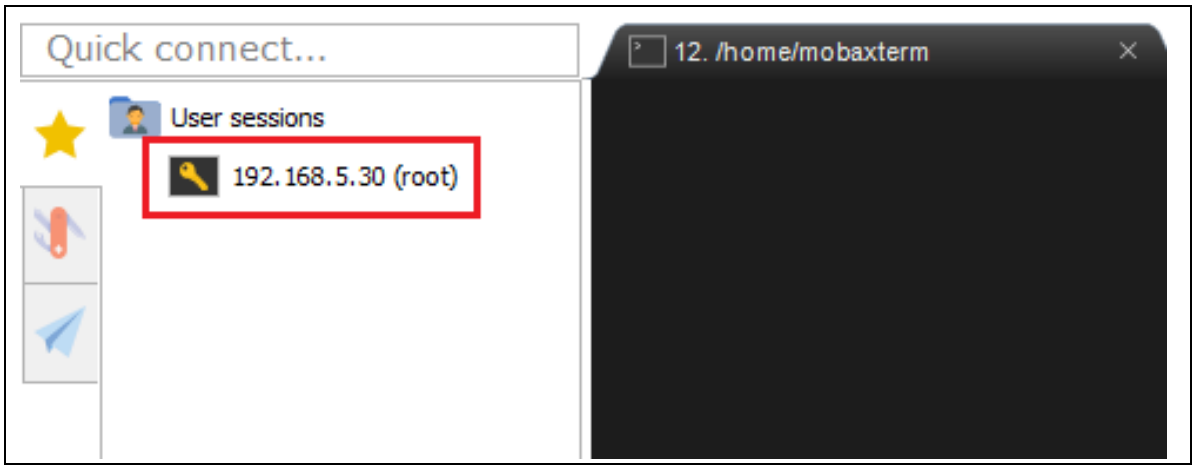


Figure 51. Connect to an SSH session in Mobaxterm

11.3.2 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.
```

11.4 SCP (Secure copy protocol)

To securely transfer files between local and remote systems, SCP can be used on both Windows and Linux.

11.4.1 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.

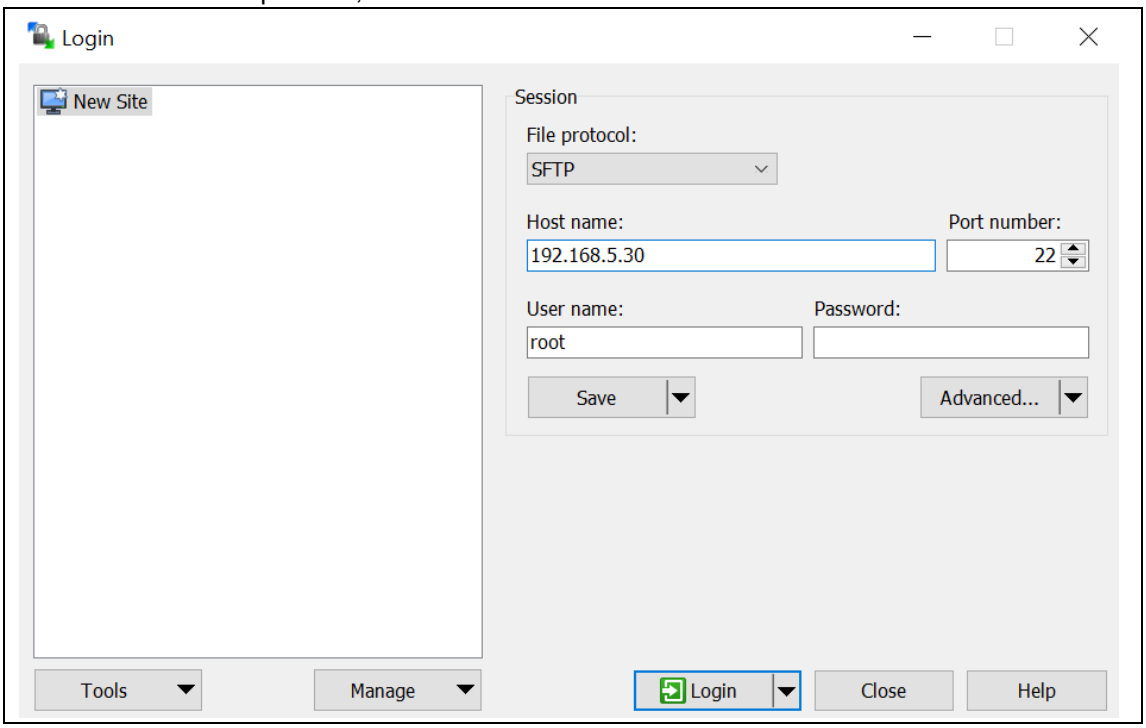


Figure 52. Setting up WinSCP for SSH session

   - 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.

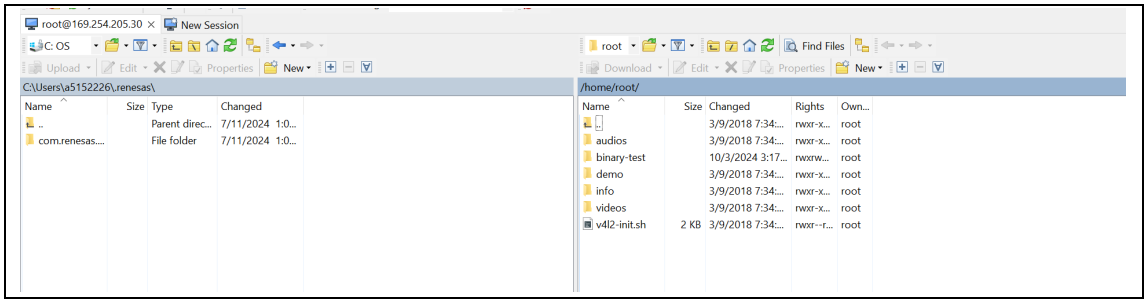


Figure 53. Using WinSCP to transfer files

11.4.2 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.
```

12. 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/G2L-SBC, 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 6.2, 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_rzsbc_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_rzsbc_board/build/tmp/deploy/sdk$ ls
poky-glibc-x86_64-core-image-bsp-cortexa55-rzg2l-sbc-toolchain-5.1.4.host.manifest
poky-glibc-x86_64-core-image-bsp-cortexa55-rzg2l-sbc-toolchain-5.1.4.sh
poky-glibc-x86_64-core-image-bsp-cortexa55-rzg2l-sbc-toolchain-5.1.4.target.manifest
poky-glibc-x86_64-core-image-bsp-cortexa55-rzg2l-sbc-toolchain-5.1.4.testdata.json
poky-glibc-x86_64-core-image-minimal-cortexa55-rzg2l-sbc-toolchain-5.1.4.host.manifest
poky-glibc-x86_64-core-image-minimal-cortexa55-rzg2l-sbc-toolchain-5.1.4.sh
poky-glibc-x86_64-core-image-minimal-cortexa55-rzg2l-sbc-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-core-image-minimal-cortexa55-rzg2l-sbc-toolchain-5.1.4.testdata.json
poky-glibc-x86_64-core-image-weston-cortexa55-rzg2l-sbc-toolchain-5.1.4.host.manifest
poky-glibc-x86_64-core-image-weston-cortexa55-rzg2l-sbc-toolchain-5.1.4.sh
poky-glibc-x86_64-core-image-weston-cortexa55-rzg2l-sbc-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-core-image-weston-cortexa55-rzg2l-sbc-toolchain-5.1.4.testdata.json
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rzg2l-sbc-toolchain-
5.1.4.host.manifest
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rzg2l-sbc-toolchain-5.1.4.sh
```

```
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rzg2l-sbc-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-renesas-core-image-cli-cortexa55-rzg2l-sbc-toolchain-
5.1.4.testdata.json
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rzg2l-sbc-toolchain-
5.1.4.host.manifest
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rzg2l-sbc-toolchain-5.1.4.sh
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rzg2l-sbc-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rzg2l-sbc-toolchain-
5.1.4.testdata.json
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rzg2l-sbc-toolchain-
5.1.4.host.manifest
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rzg2l-sbc-toolchain-5.1.4.sh
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rzg2l-sbc-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-renesas-quickboot-cli-cortexa55-rzg2l-sbc-toolchain-
5.1.4.testdata.json
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rzg2l-sbc-toolchain-
5.1.4.host.manifest
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rzg2l-sbc-toolchain-5.1.4.sh
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rzg2l-sbc-toolchain-
5.1.4.target.manifest
poky-glibc-x86_64-renesas-quickboot-wayland-cortexa55-rzg2l-sbc-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.

13. Application Building, Packaging, and Running

The SDK allows you to develop and test custom applications for the RZ/G2L-SBC on different systems. This section covers setting up your development environment and running your applications.

13.1 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_rzsbc_board
$ sh ./build/tmp/deploy/sdk/poky-glibc-x86_64-core-image-weston-cortexa55-rzg2l-sbc-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_rzsbc_board
$ sh ./build/tmp/deploy/sdk/poky-glibc-x86_64-core-image-weston-cortexa55-rzg2l-sbc-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.

13.2 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
```

13.3 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/G2L SBC as an application. CPack is a CMake module that handles packaging.

13.3.1 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: 13.2 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 RZ/G2L-SBC.

The Debian package installer “helloworld-1.0.0-Linux.deb” can be transferred to the RZ/G2L-SBC 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
```

13.3.2 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 RZ/G2L-SBC 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
```

13.4 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:~#
```

13.5 Install and Run Debian application packages by using DPKG

After shipping the Debian package installer to RZ/G2L-SBC, 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 RZ/G2L-SBC.
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>
…
rzg2l-sbc login: root
root@rg2l-sbc:~# ls
audios  demo  helloworld-1.0.0-Linux.deb  tkinter_example-1.0.0-Linux.deb  images
info  v4l2-init.sh  videos
root@rzg2l-sbc:~# 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@rzg2l-sbc:~#
root@rzg2l-sbc:~# 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@rzg2l-sbc:~# 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 the 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@rzg2l-sbc:~# dpkg -L helloworld
/usr
/usr/local
/usr/local/bin
/usr/local/bin/hello
root@rzg2l-sbc:~# /usr/local/bin/helloworld
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@rzg2l-sbc:~# su - weston
```

```
rzg2l-sbc:~$ export DISPLAY=:0
rzg2l-sbc:~$ 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
rzg2l-sbc:~$ tkinter_example
```

14. Remote Debugging using GDBServer

GDBServer is utilized to facilitate remote debugging on the RZ/G2L-SBC. GDBServer enables the debugging process to run on the RZ/G2L-SBC (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 RZ/G2L-SBC 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.

14.1 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 13.1, is a prerequisite for this operation. To set up the environment that would use the GDB targeting the RZ/G2L-SBC 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
```

14.2 Install GDBServer on RZ/G2L-SBC

By default, GDBServer is not installed on the RZ/G2L-SBC. It is necessary to install it using APT. Execute the following commands to install GDBServer:

```
root@rzpi:~# apt-get update

root@rzpi:~# 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.

14.3 Remote Debugging Example

14.3.1 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@rzg2l-sbc:~# 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@rzg2l-sbc:~# 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:~#
```

14.3.2 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 RZ-G2L/SBC.

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.

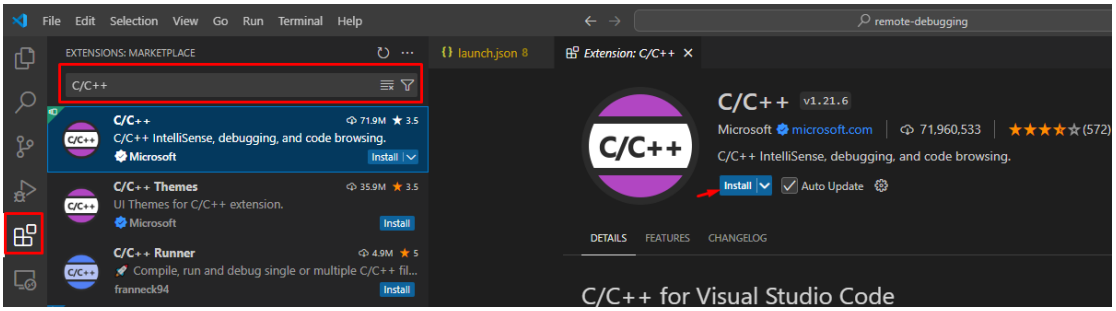


Figure 54. C/C++ Extension in VSCode

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.

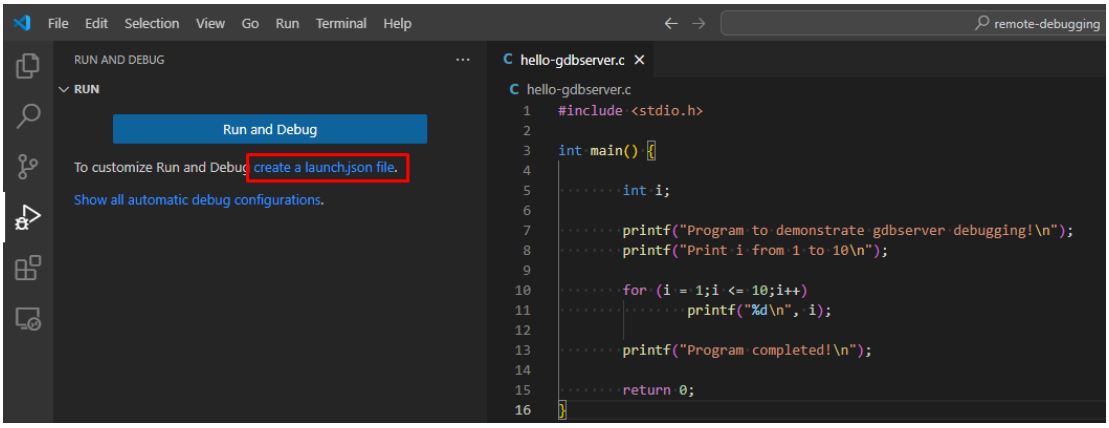


Figure 55. Create a launch.json file in VSCode Debugger

- Select the C++ (GDB) option and customize the configuration as needed.

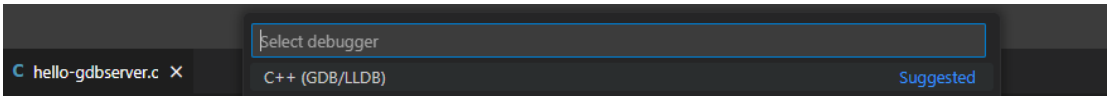


Figure 56. Select C++ GDB as Debugger

- 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.

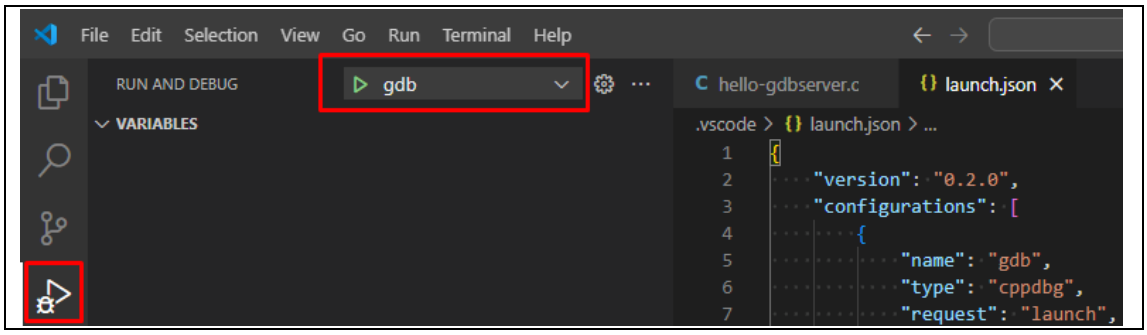


Figure 57. Running the Debugger for Remote Debugging in VSCode

- Use F5 to continue execution, F10 to step over the current line, and F11 to step into functions.

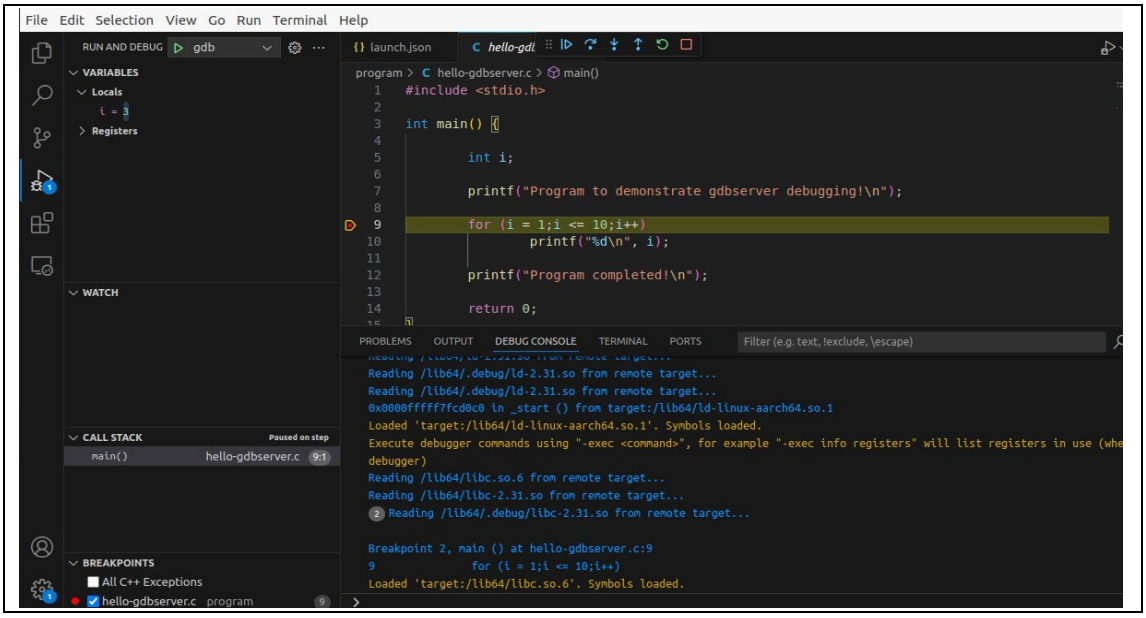


Figure 58. Step through each step in Debug Mode in VSCode

14.3.3 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.

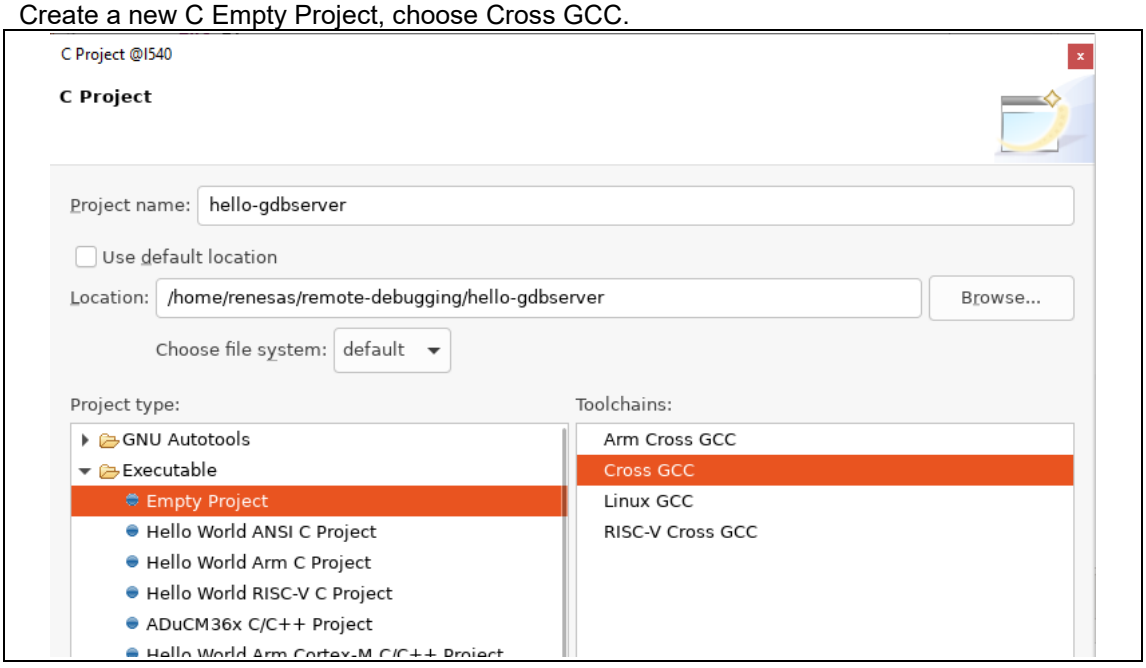


Figure 59. Create a C Project in Eclipse

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.

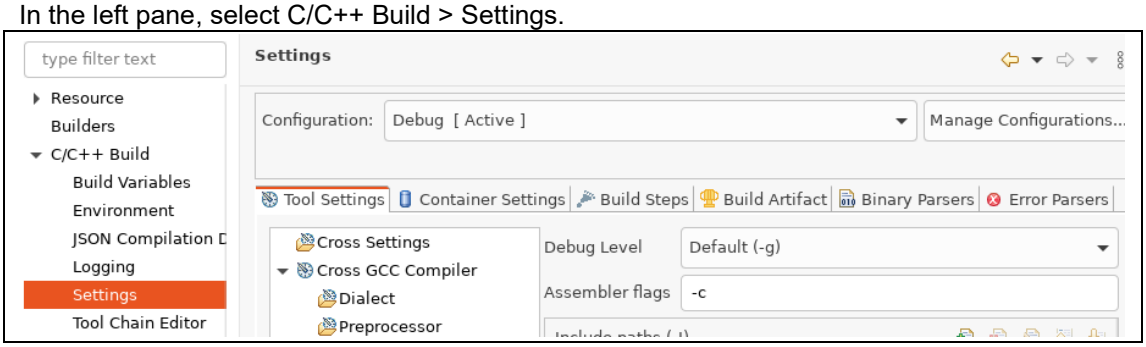


Figure 60. Configuring the cross toolchain in Eclipse project properties

- 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.

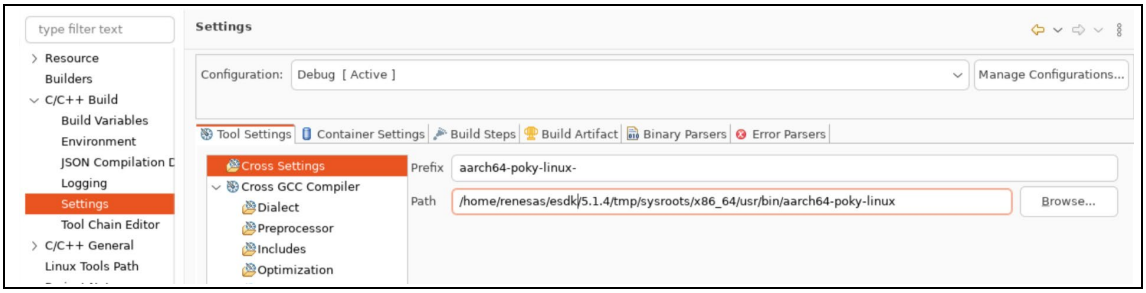


Figure 61. Configuring cross compiler settings in Eclipse tool settings

- In the Includes section, specify the include paths:
  - Include paths: /home/renesas/esdk/5.1.4/tmp/sysroots/rzg2l-sbc/usr/include

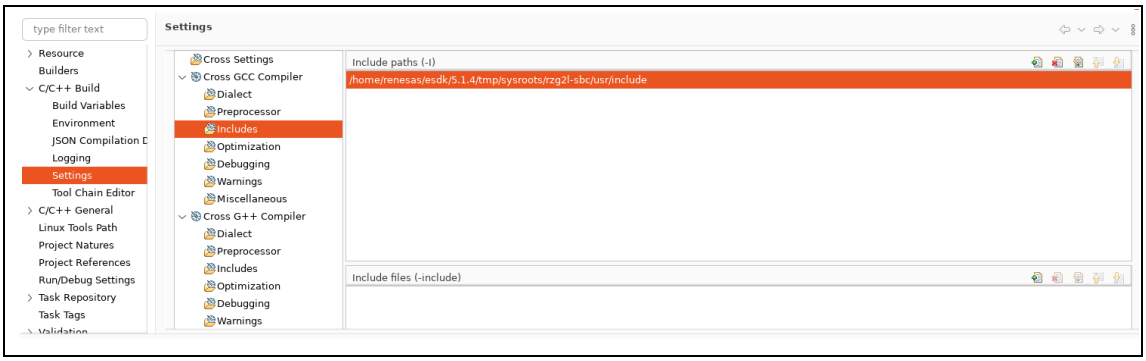


Figure 62. Configuring includes path in Eclipse tool settings

- 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

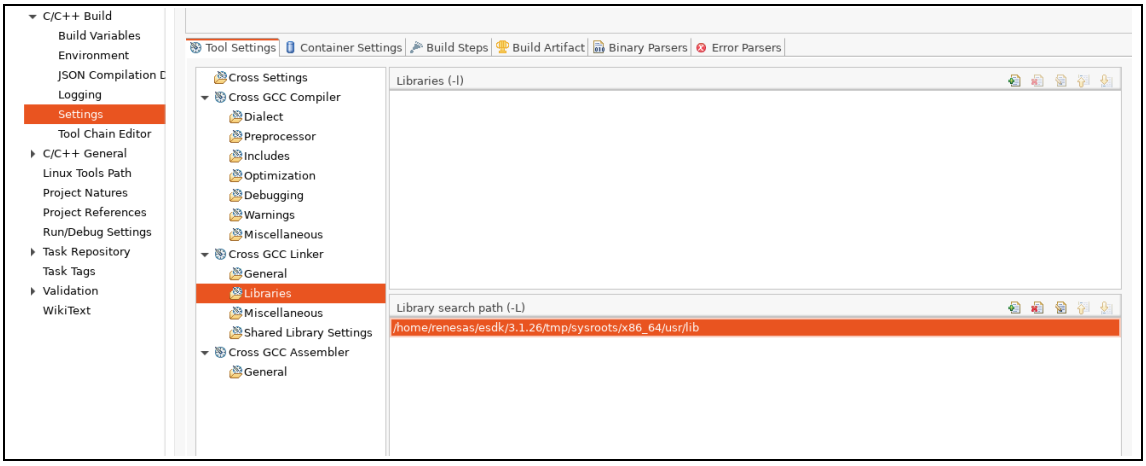


Figure 63. Configuring library paths in Eclipse tool settings

- In the Miscellaneous section, specify the linker flags:
  - Linker flags: --sysroot=/home/renesas/esdk/5.1.4/poky_sdk/tmp/sysroots/rzg2l-sbc

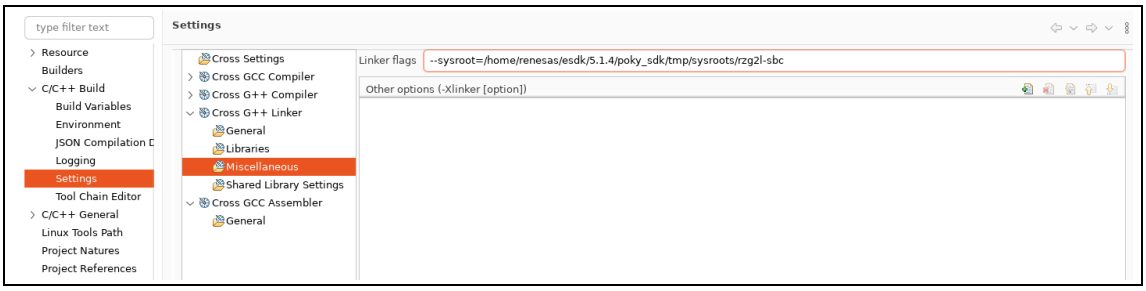


Figure 64. Configuring linker flags for Sysroot in Eclipse tool settings

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.

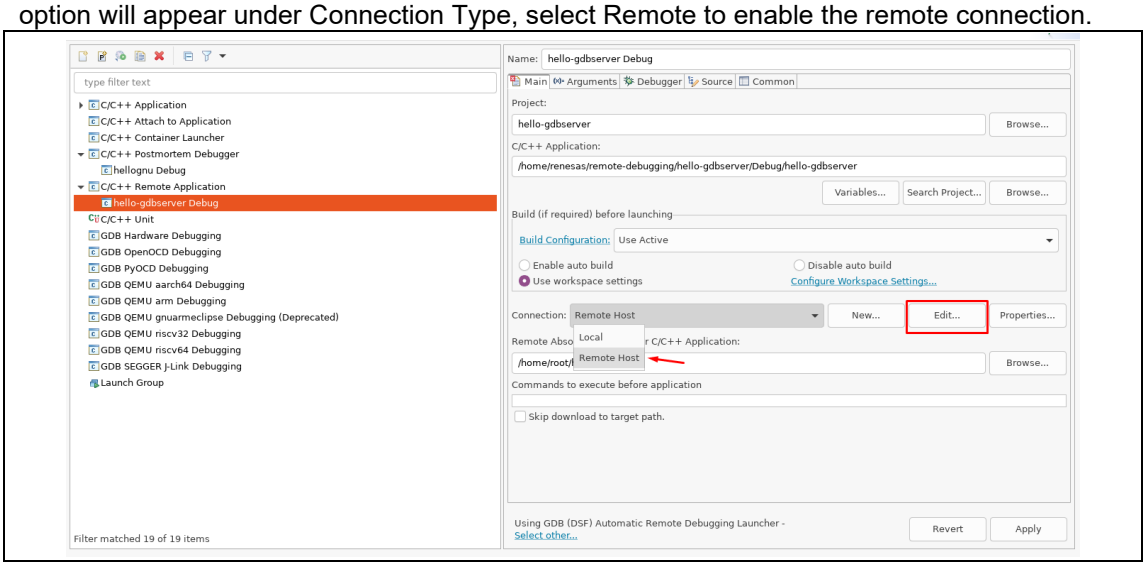


Figure 65. Debug configuration settings in Eclipse

- 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.

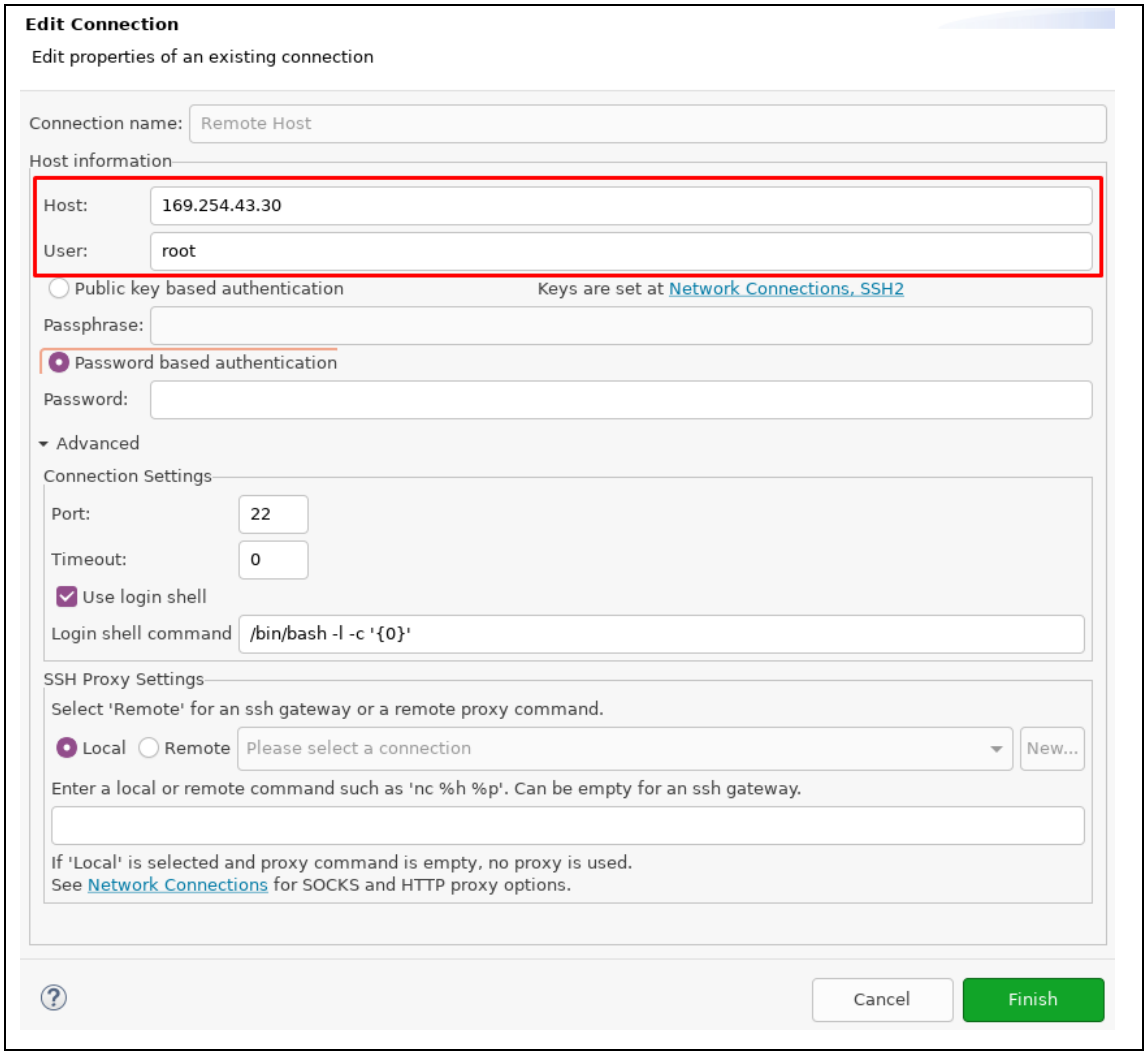


Figure 66. Configuring SSH connection settings in debug configurations

- In the Remote Absolute File Path field, specify the location where Eclipse will copy the program on the RZ/G2L-SBC. Click Browse to connect via SSH and select the target location or manually enter the path on the RZ/G2L-SBC.

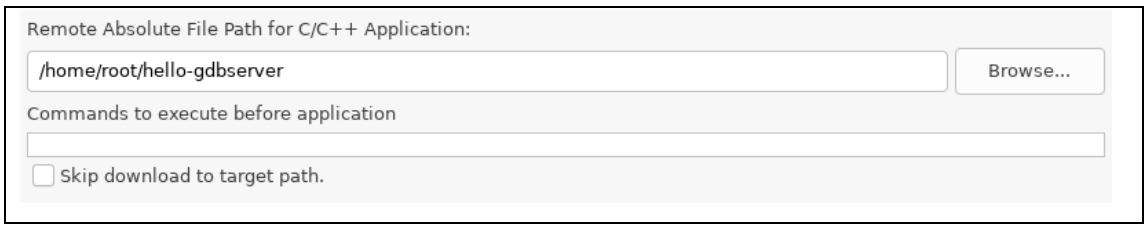


Figure 67. Configuring the remote absolute file path in debug configurations

- 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).

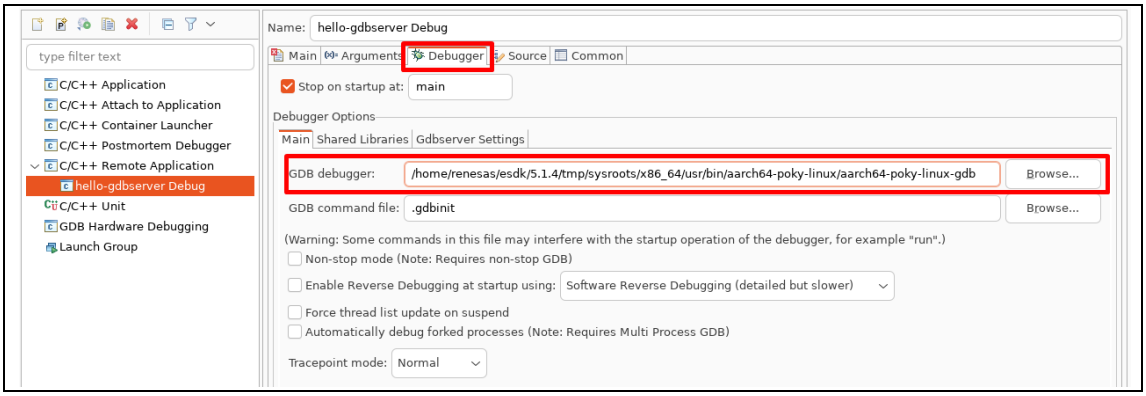


Figure 68. Configuring the GDB debugger in Eclipse debug configurations

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.

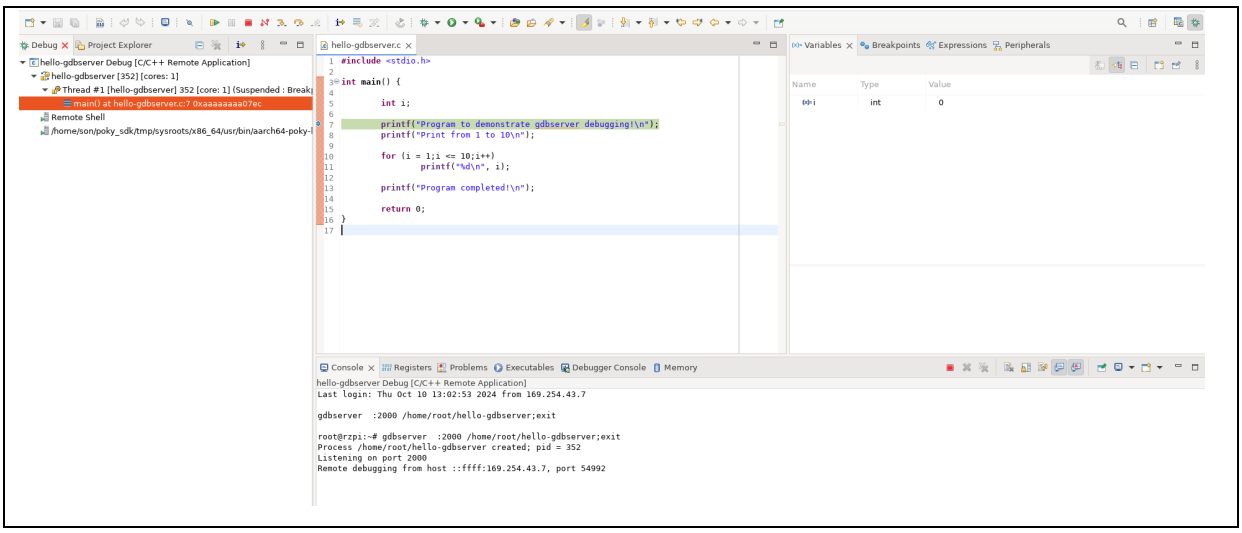


Figure 69. Start the debugging session in Eclipse

Press F5 to step into, F6 to step over, or F8 to resume and monitor the variables.

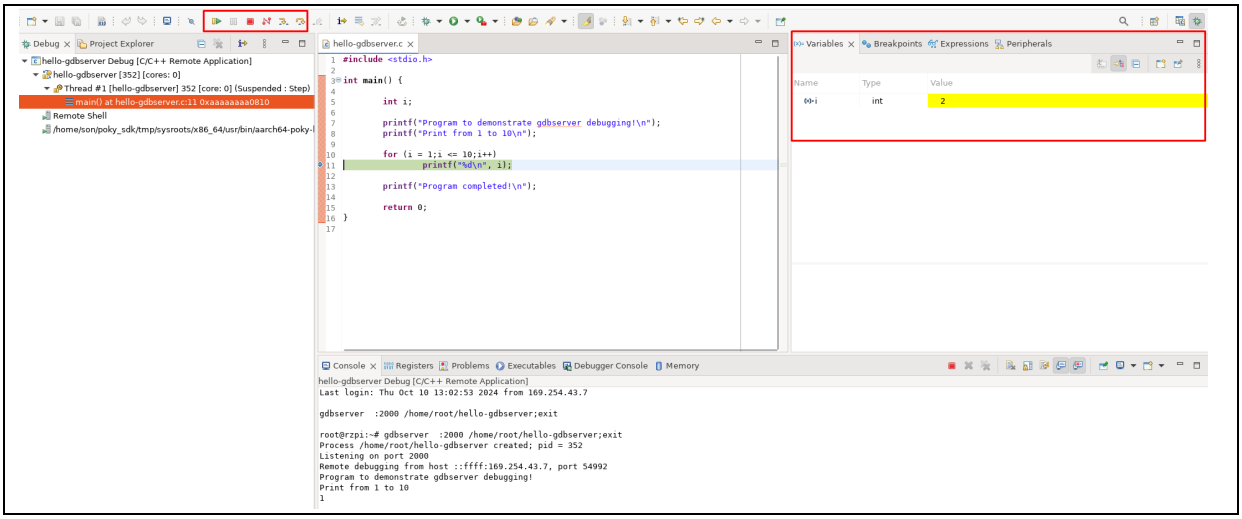


Figure 70. Go through each step in the debug mode in Eclipse

The path of the compiler may need to be adjusted to reflect the specific system configuration.

14.4 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.

14.4.1 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@rzg2l-sbc:~# ulimit -c unlimited
root@rzg2l-sbc:~# 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@rzg2l-sbc:~# ./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@rzg2l-sbc:~# 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.

14.4.2 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 14.3.2 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.

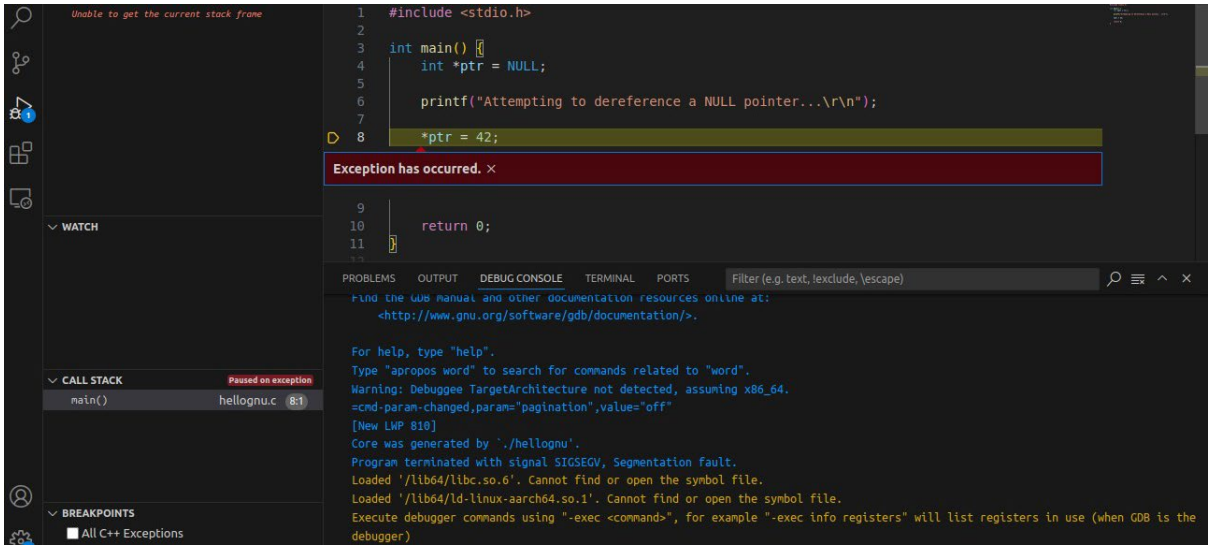


Figure 71. Starting analysis of the Core dump file in VSCode debug mode

14.4.3 Postmortem Analysis on Eclipse

This subsection describes postmortem analysis using the Eclipse IDE. Similar to Visual Studio Code, Eclipse allows loading a 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 the Core file field, click and specify where the core dump file is.

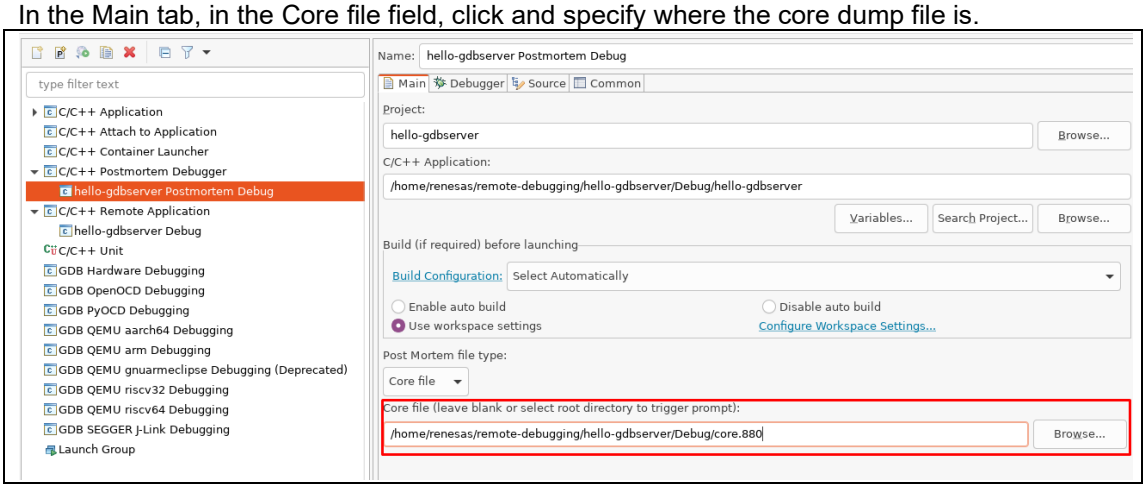


Figure 72. Specifying the Core File in Eclipse Debugger Settings

- 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).

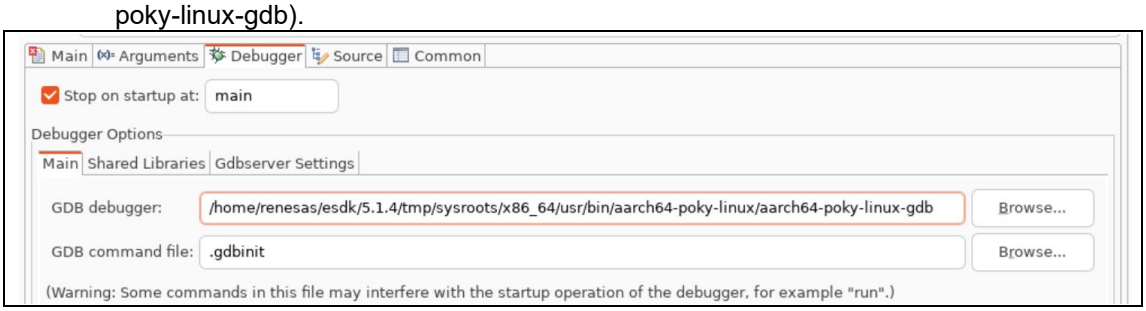


Figure 72. Specifying the GDB Debugger in Eclipse Debugger settings

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.

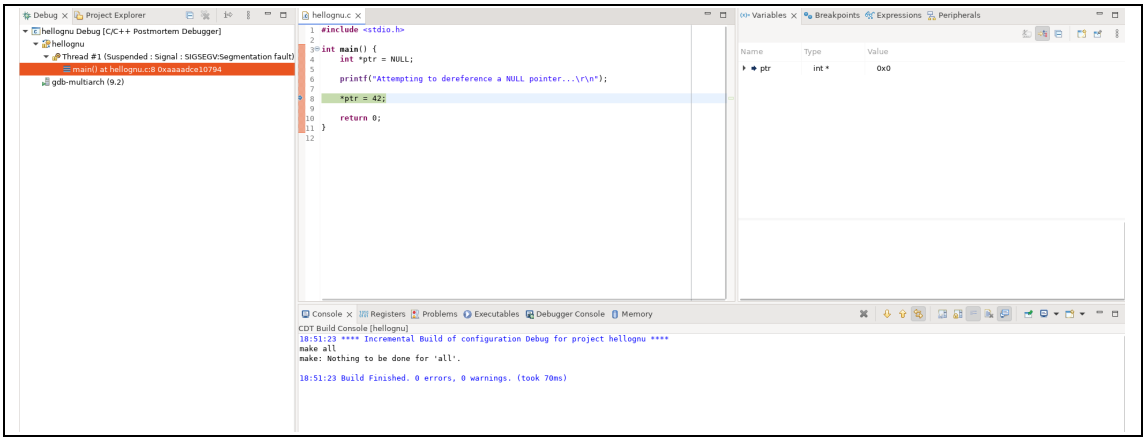


Figure 73. Starting analysis of the Core dump file in Eclipse debug mode

- 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.

15. Functional Overview

15.1 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.

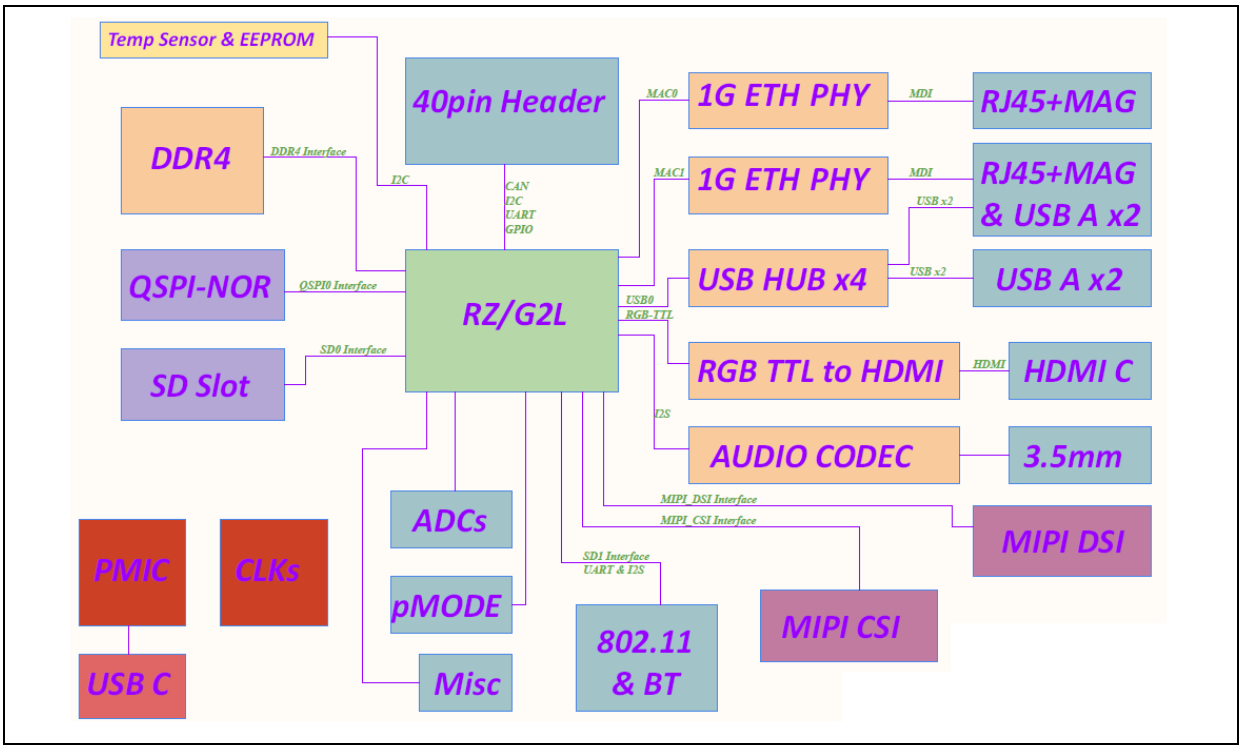


Figure 74. RZ/G2L SBC System Overview

Table 13. 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 -40 °C°C to + 85 °C	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 -40 °C to + 85 °C	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 14. 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)

15.1.1 RZ/G2L SoC MPU Architecture

The RZ/G2L MPU is a feature-packed SoC (System on Chip) that can support a variety of applications. Below is an overview of SoCs.

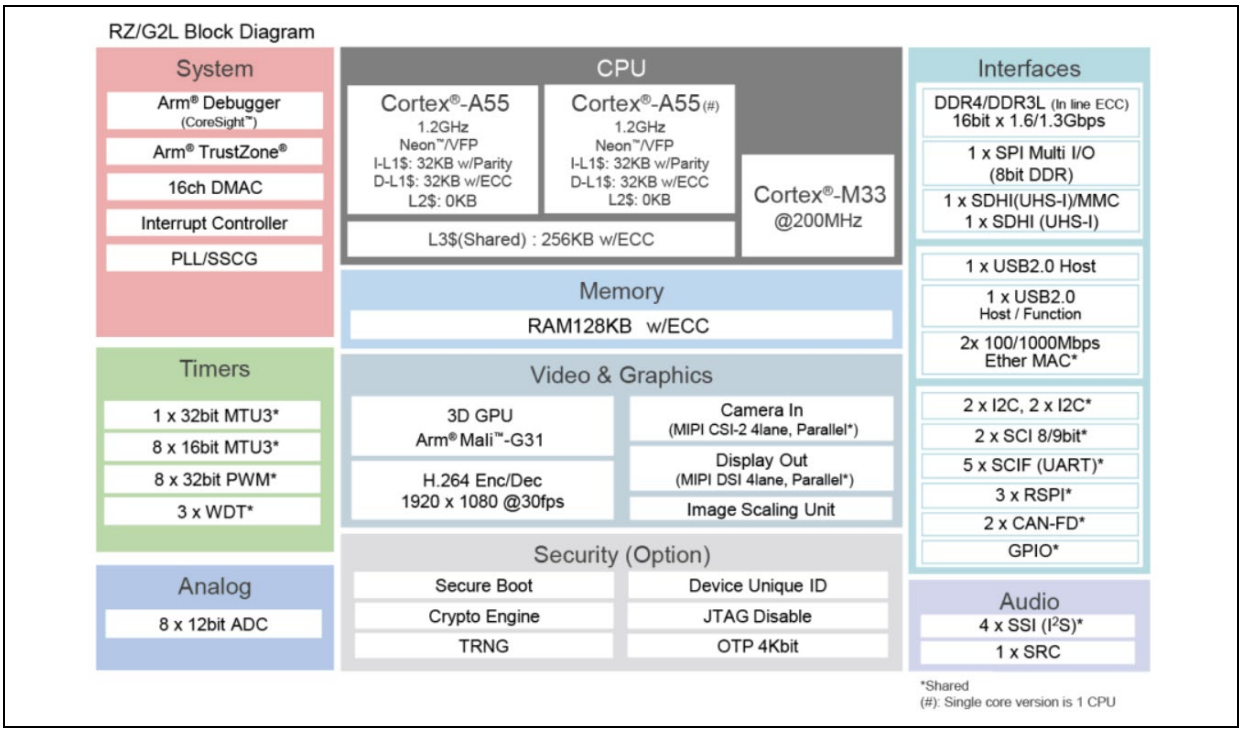


Figure 75. RZ/G2L SoC (System on Chip) Overview

15.1.2 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.

15.1.3 Physical View

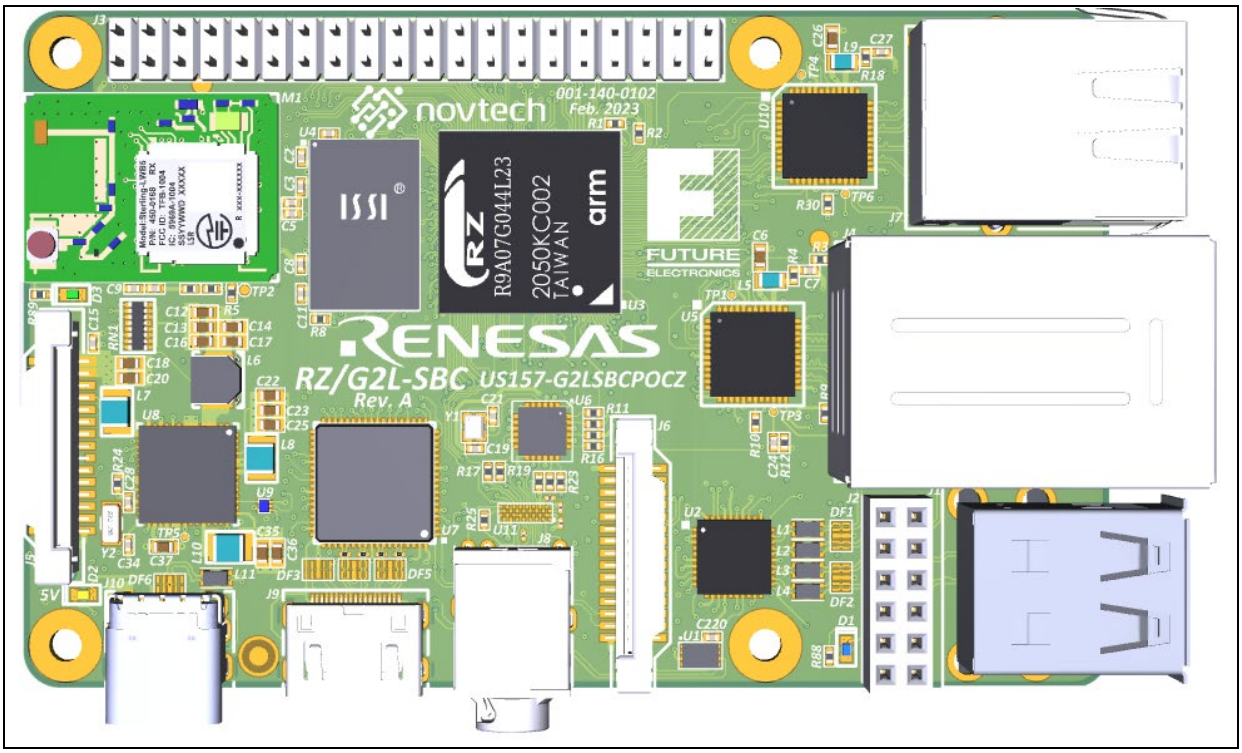


Figure 76. Top-side view of the RZ/G2L-SBC

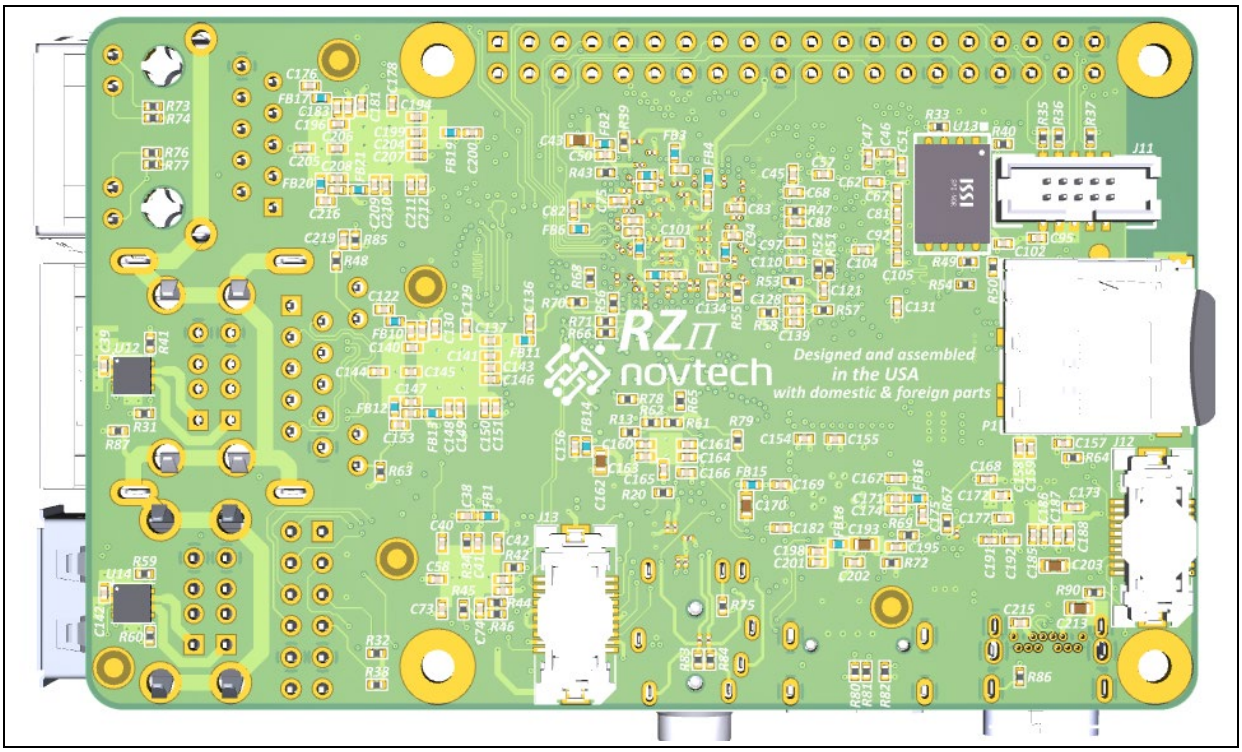


Figure 77. Bottom side of the RZ/G2L-SBC

15.1.4 Overview of Connectors

Given below is the basic positioning of the top-level connectors.

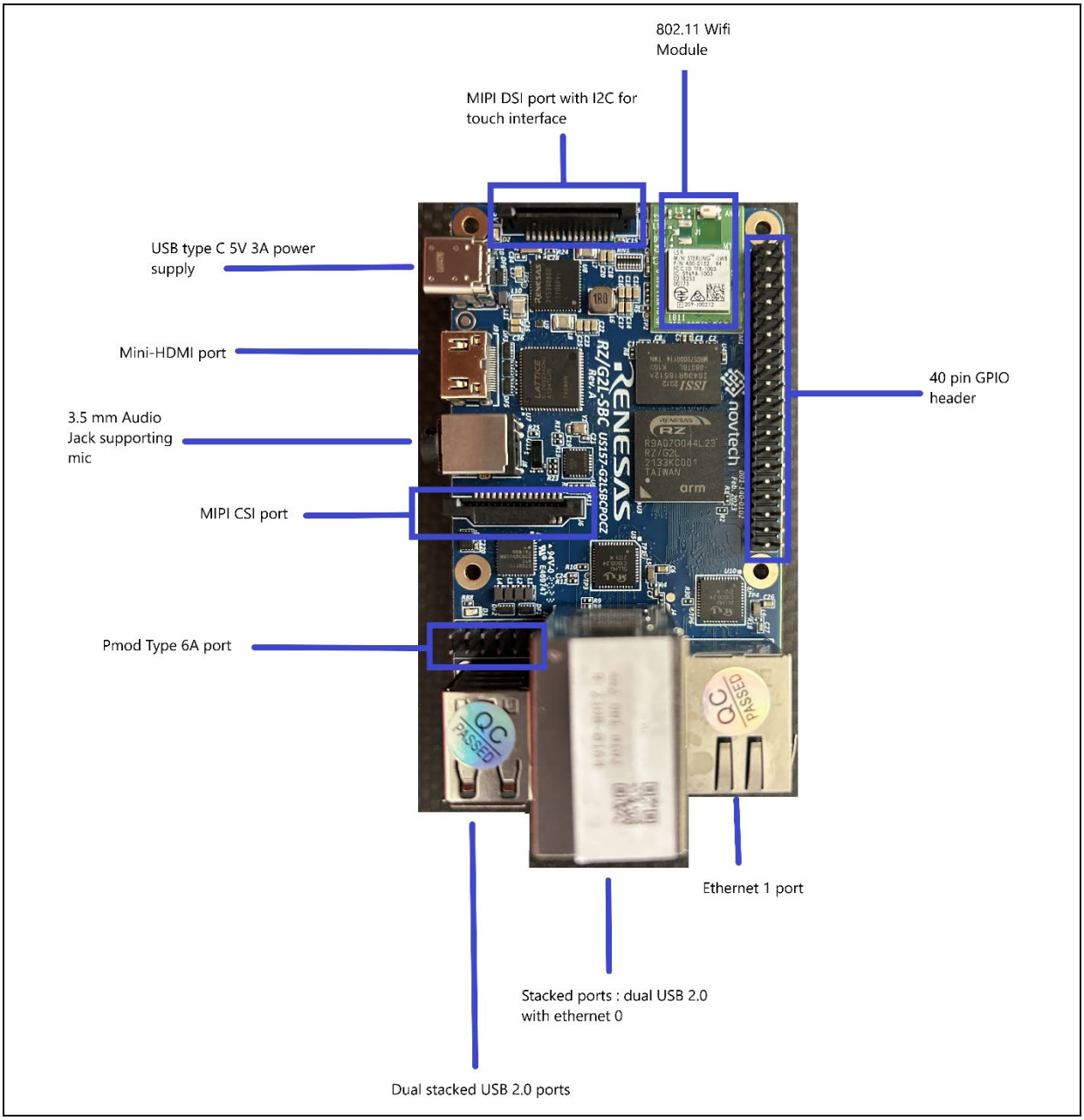


Figure 78. RZ/G2L-SBC top side connectors.

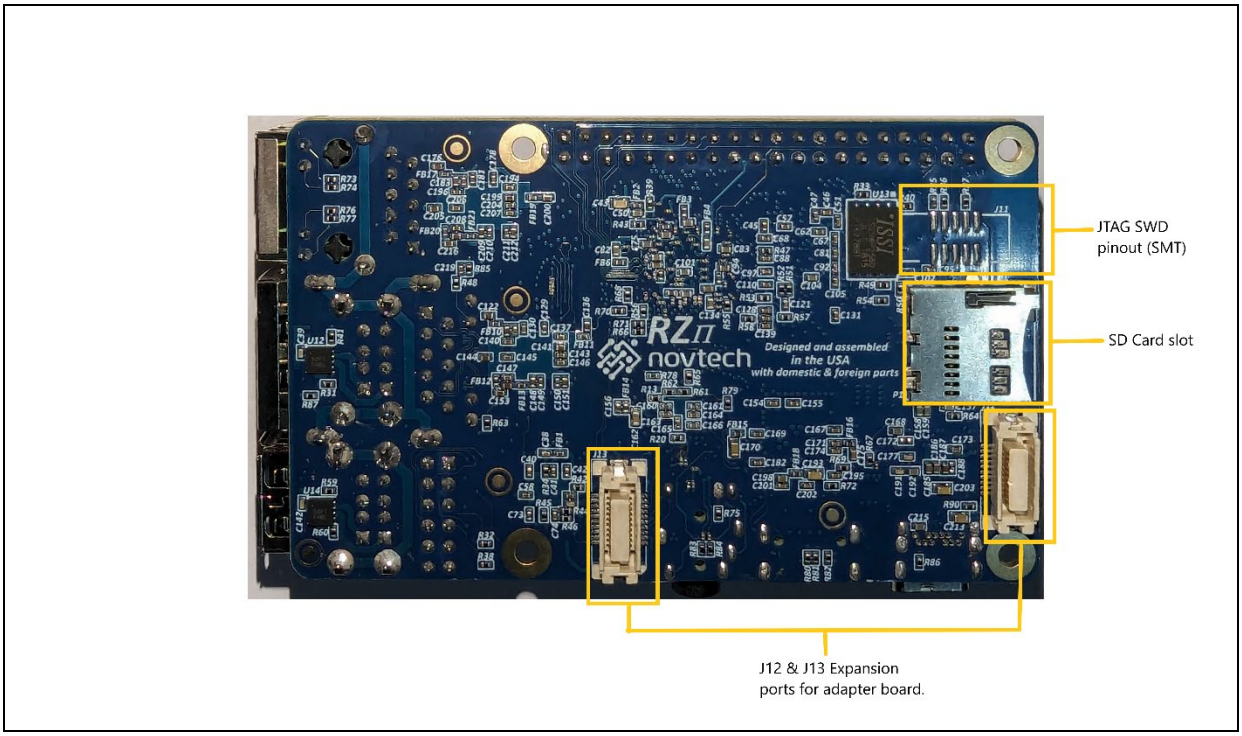


Figure 79. RZ/G2L-SBC Bottom view connectors.

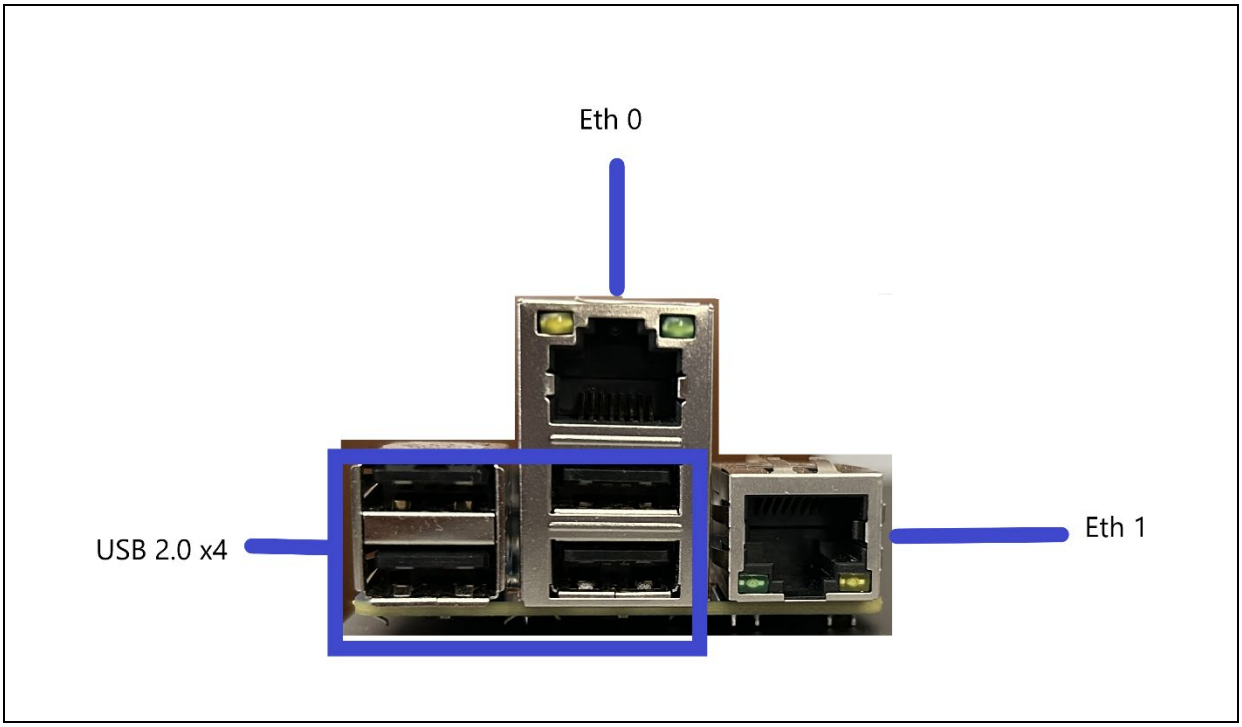


Figure 80. RZ/G2L-SBC side view I/O ports.

15.1.5 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.

15.1.5.1 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.

15.1.5.2 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.

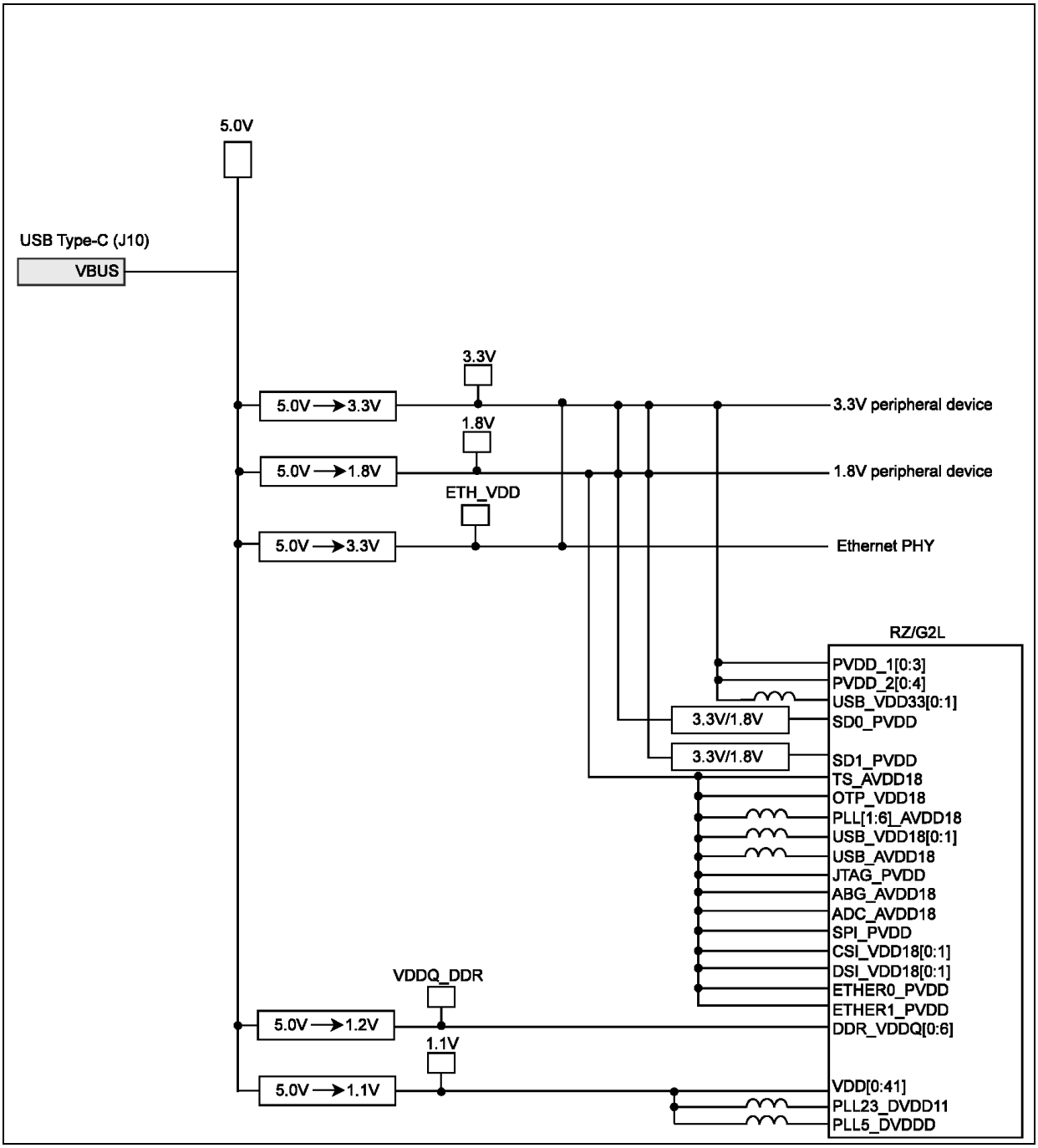


Figure 81. Power supply rails.

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.

15.1.5.3 Power Supply Regulation

The power supply is regulated by Renesas RAA215300, a low-cost nine-channel PMIC IC.

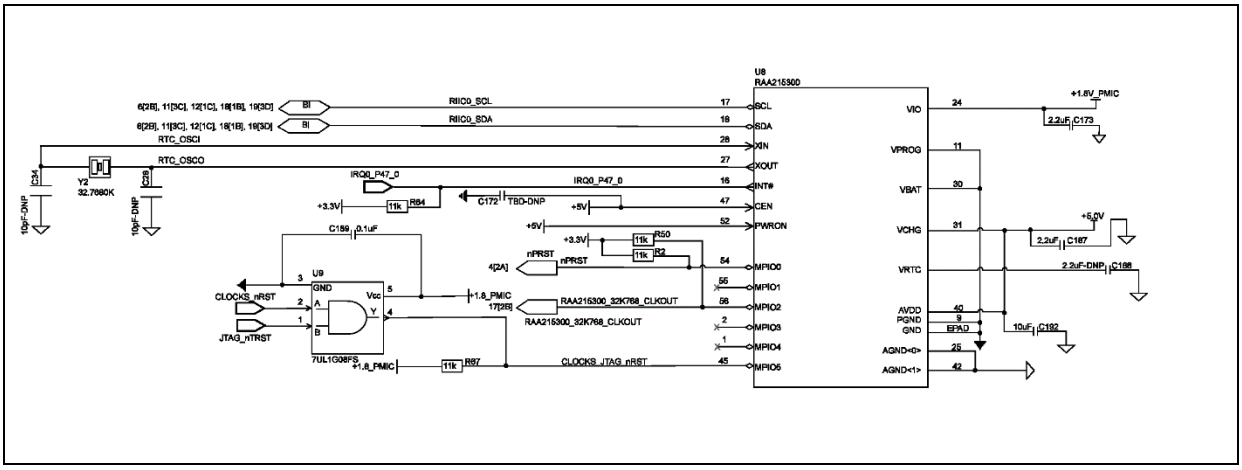


Figure 82. Block Diagram of Power Supply Regulation using RAA215300.

15.1.6 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.

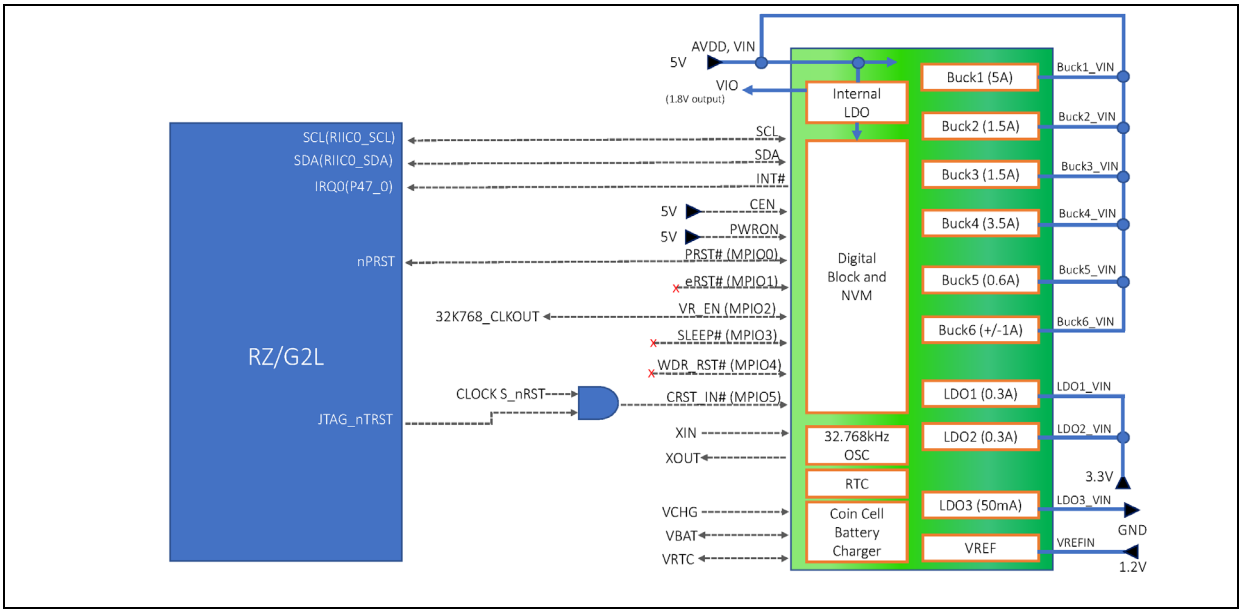


Figure 83. Block diagram of PMIC interface to RZ/G2L

15.1.7 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.

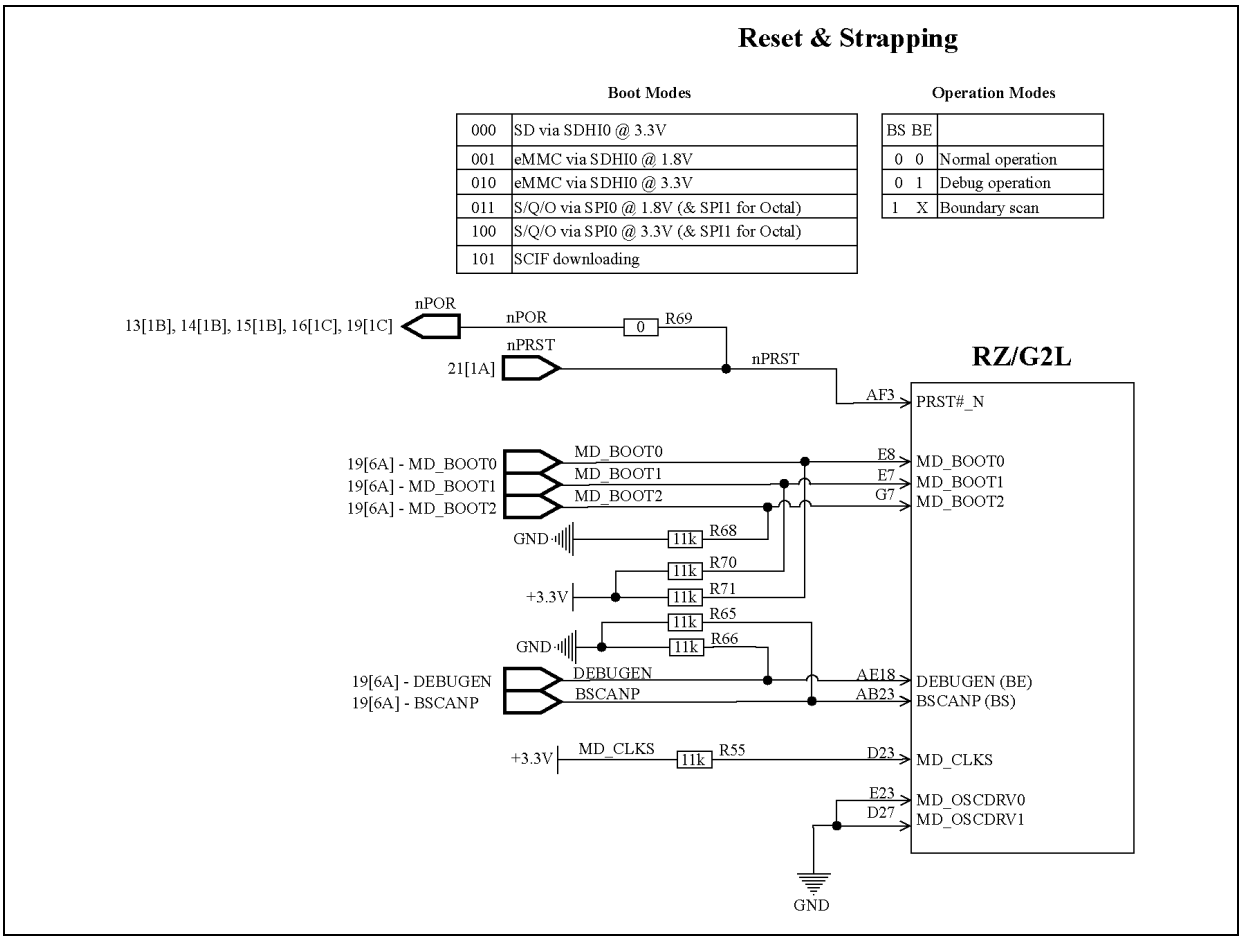


Figure 84. Reset Control Logic

15.1.8 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)

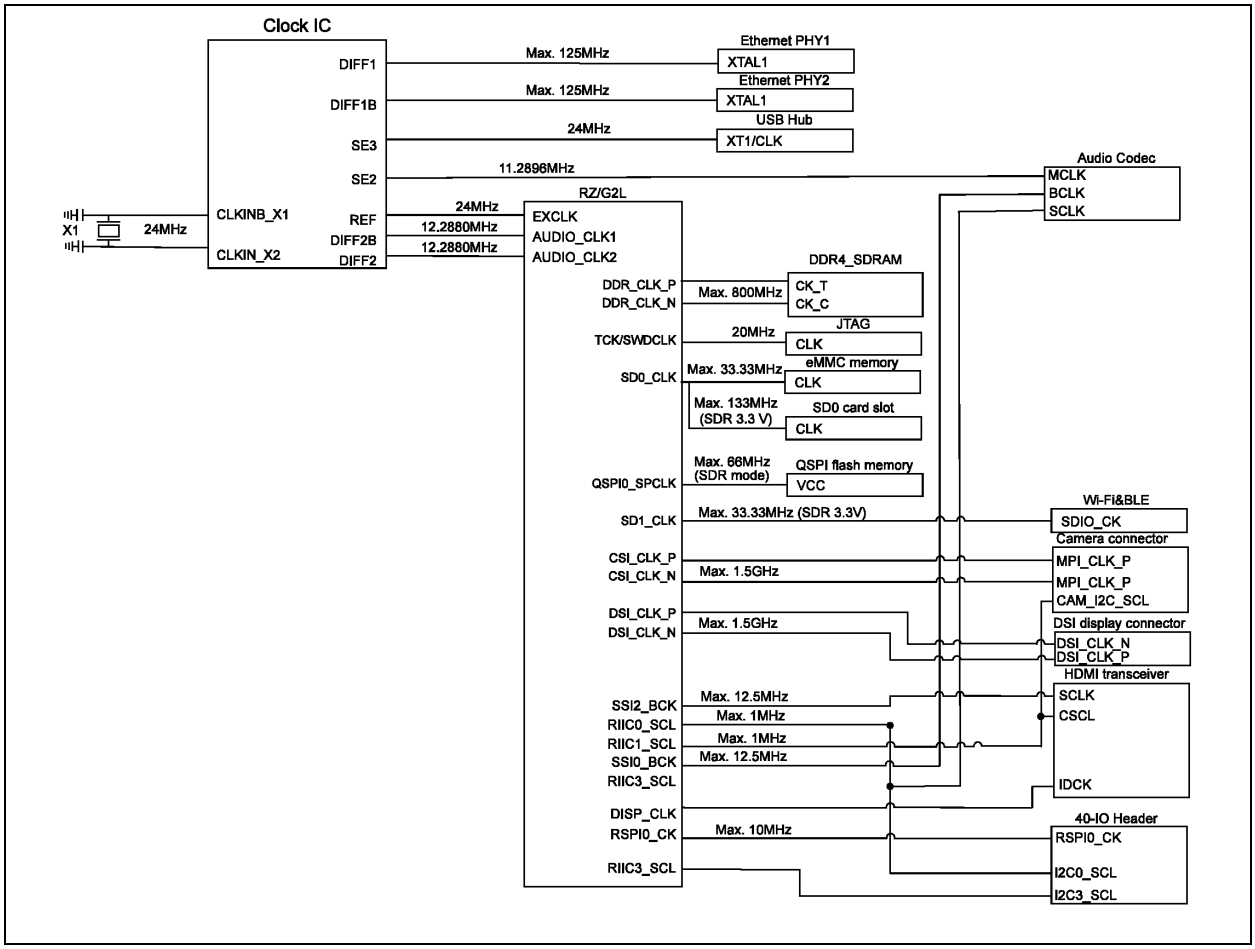


Figure 85. Block diagram of Clock interfacing.

15.1.9 Peripheral Interface

15.1.9.1 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.

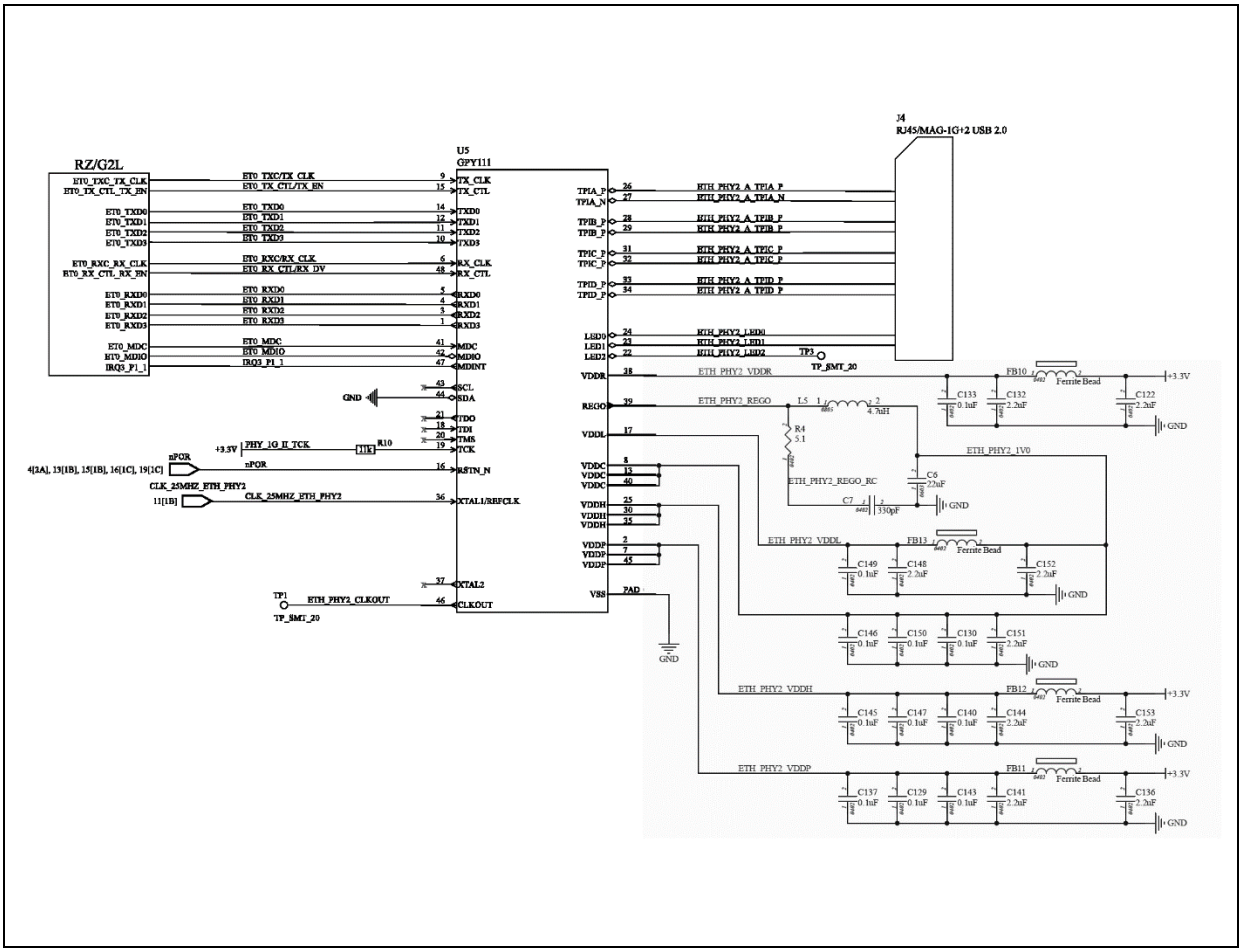


Figure 86. Ethernet 0 PHY interfacing.

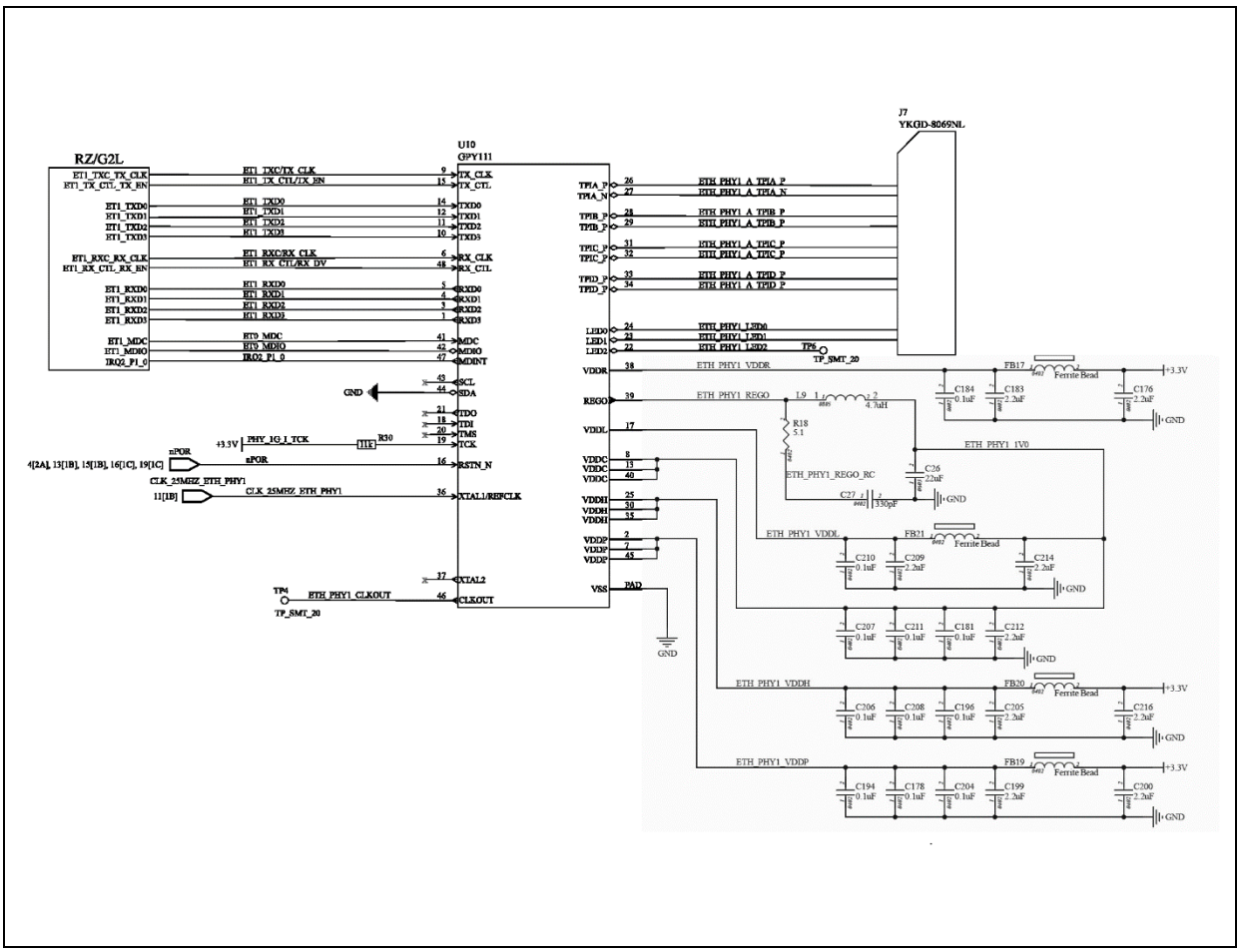


Figure 87. Ethernet 1 PHY interfacing.

15.1.9.2 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.

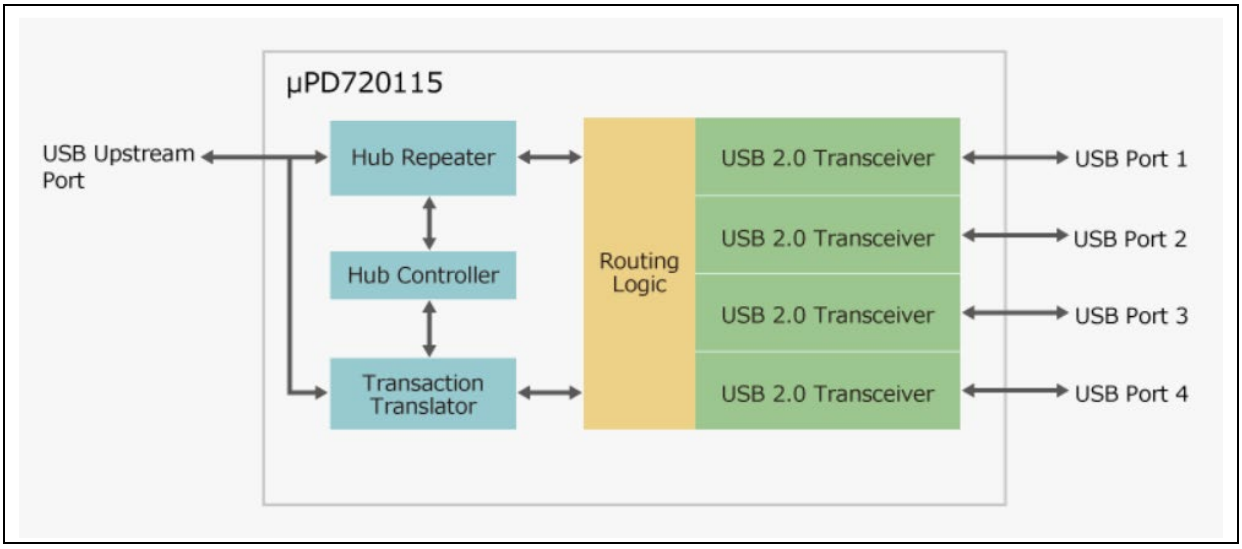


Figure 88. UPD720115 block diagram.

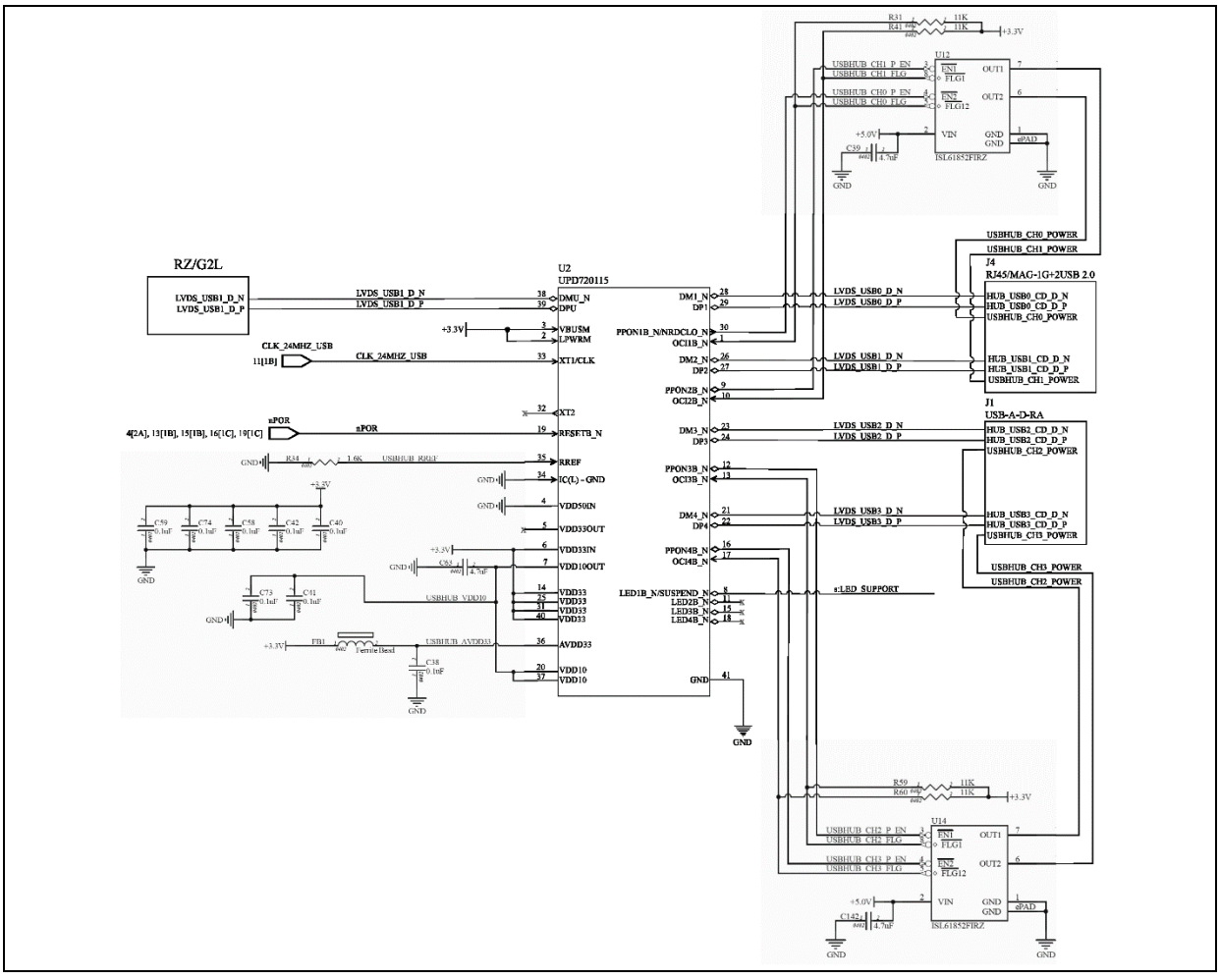


Figure 89. USB 2.0 Hub Block Diagram

15.1.9.3 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.

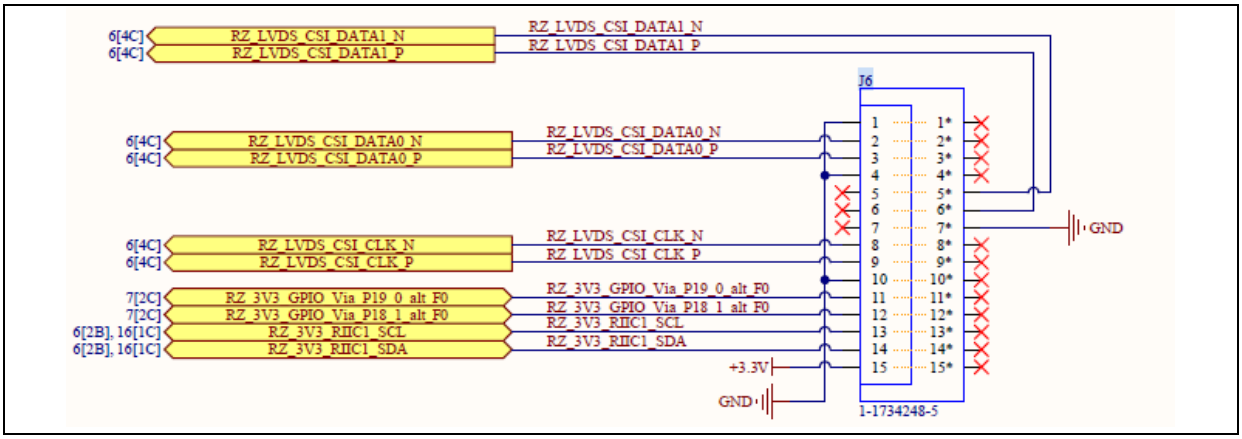


Figure 90. CSI Interface Schematic

15.1.9.4 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.

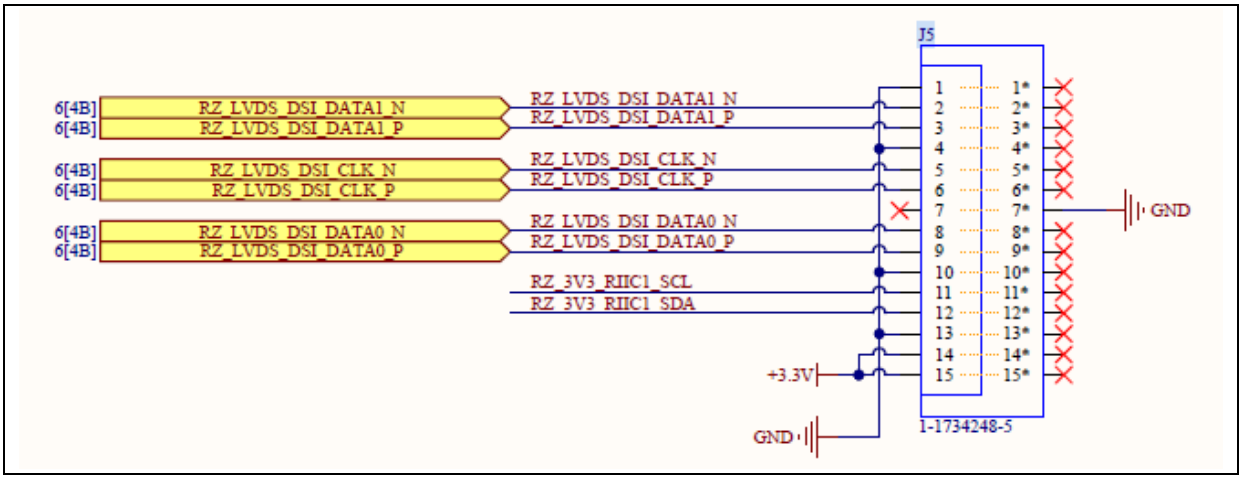


Figure 91. DSI Schematic

15.1.9.5 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.

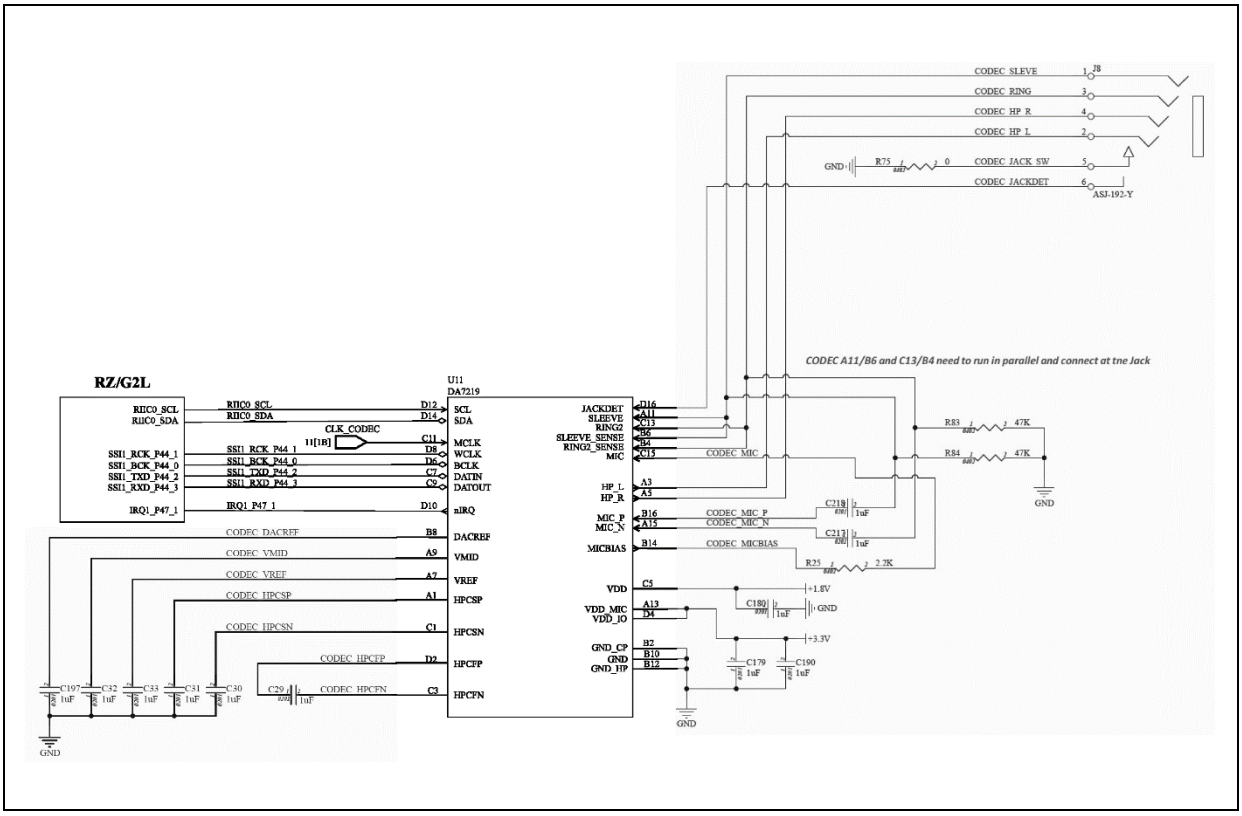


Figure 92. Audio CODEC Interface Block Diagram

15.1.9.6 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.

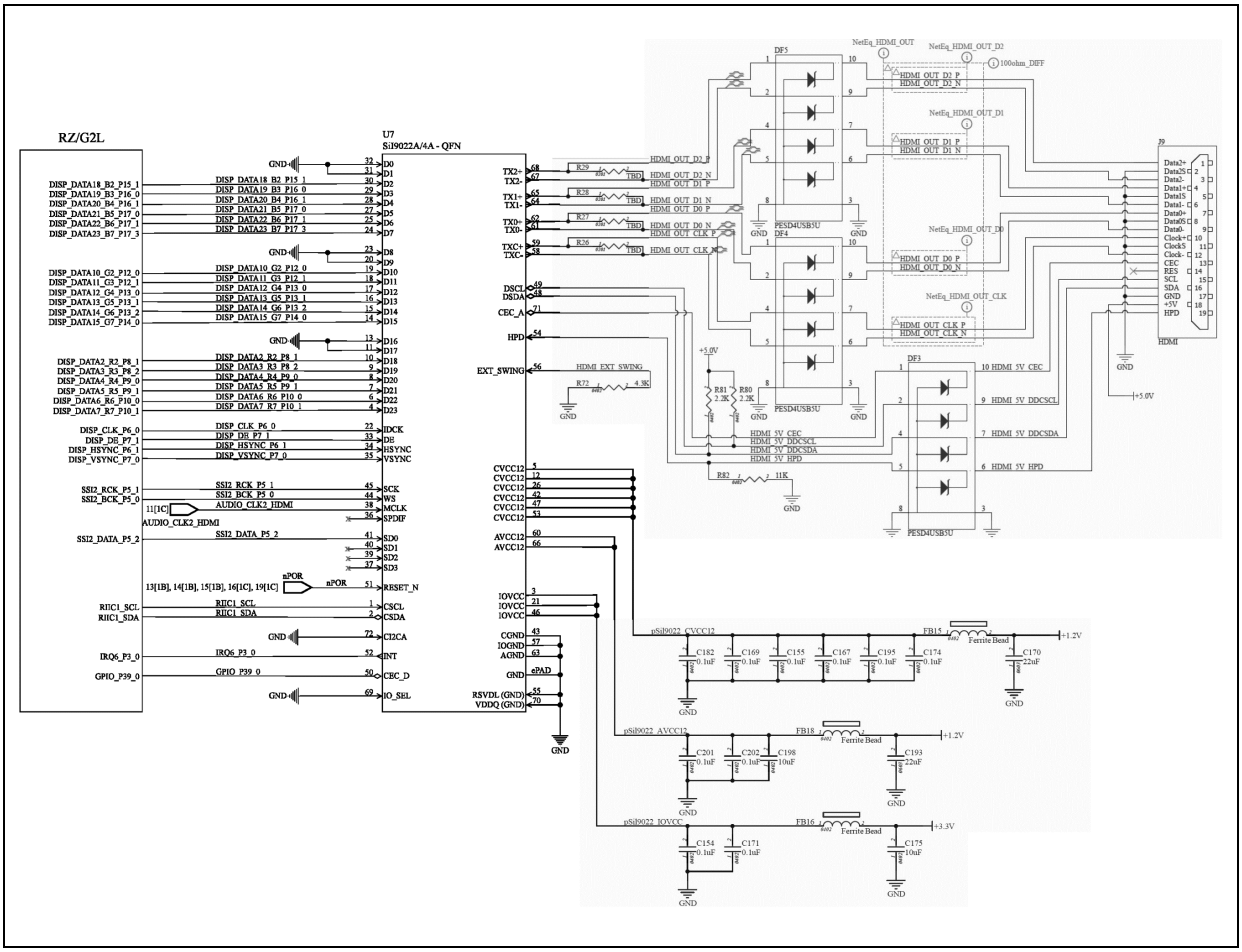


Figure 93. HDMI Bridge and mini-HDMI port interfacing.

15.1.9.7 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

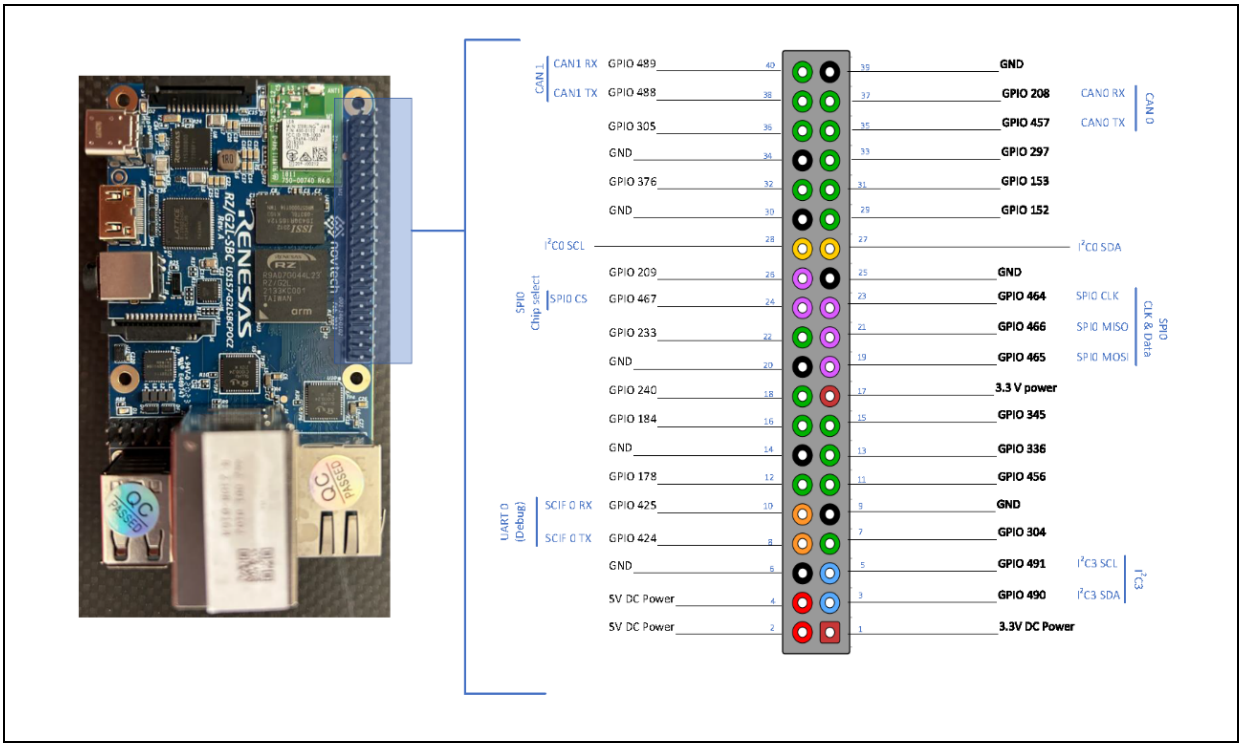


Figure 94. 40 PIN GPIO map with orientation details.

15.1.9.8 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.

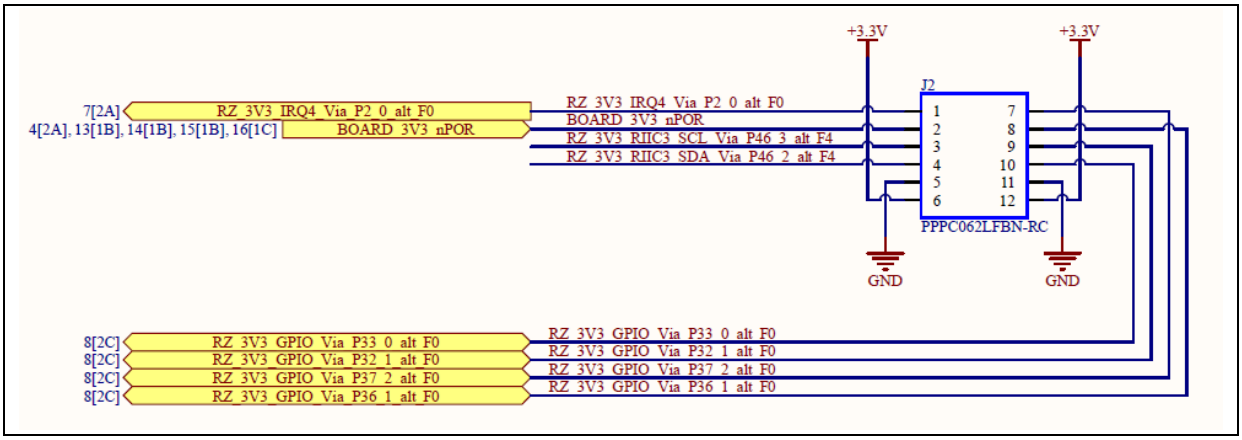


Figure 95. Schematic of PMOD Type 6 A pin header J2.

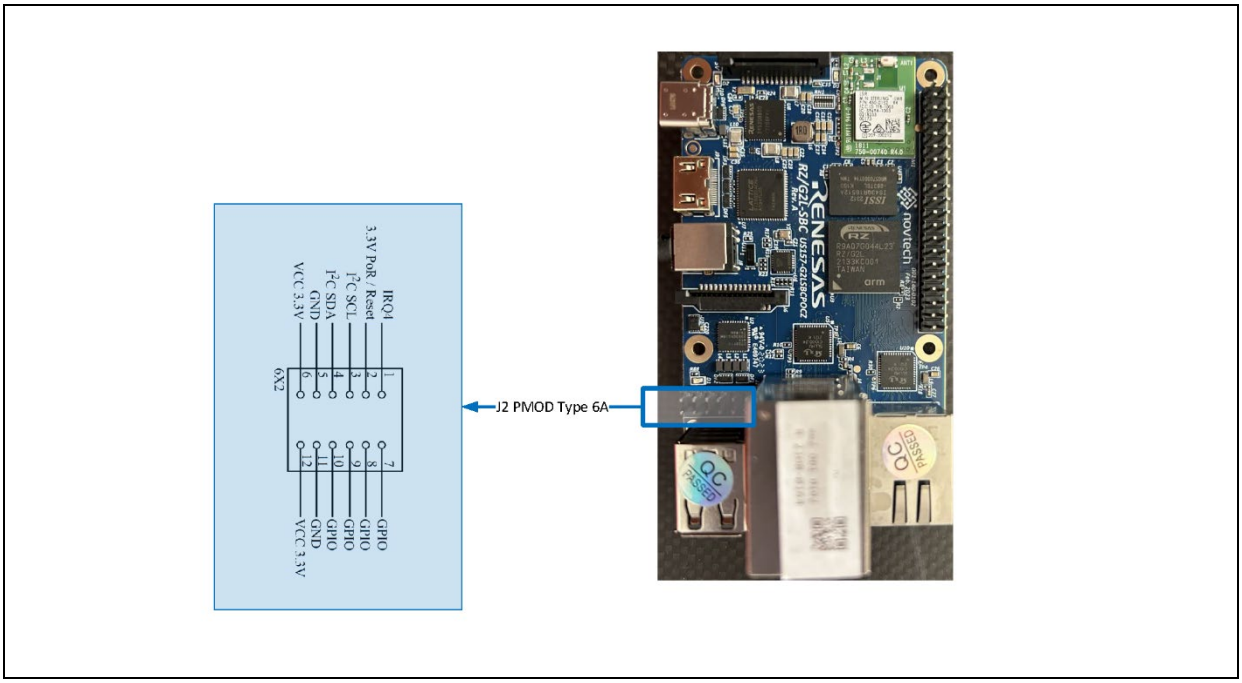


Figure 96. PMOD Type 6A 2x6 0.1mm pin out with orientation details.

15.1.9.9 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).

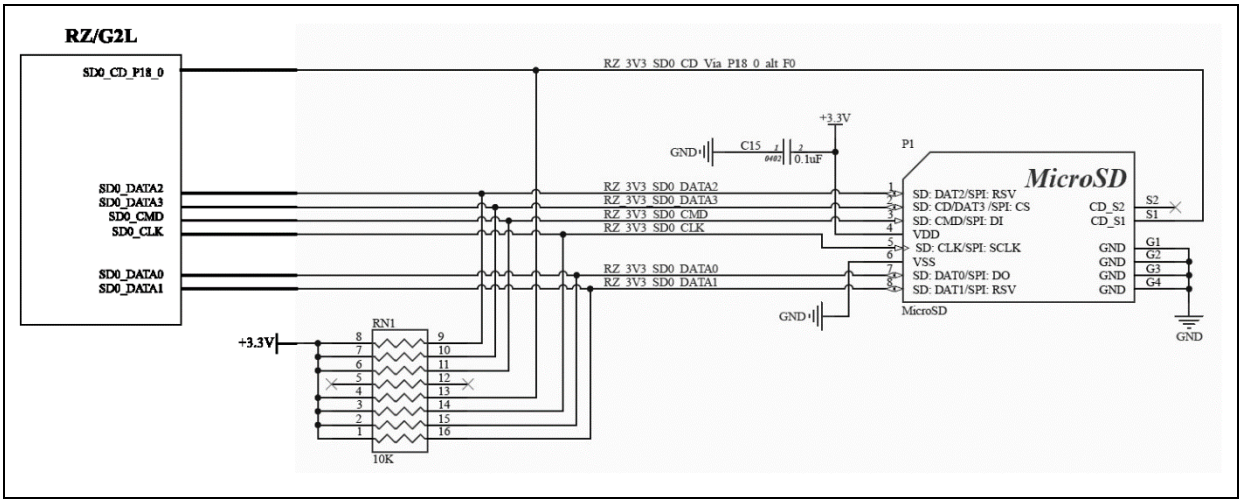


Figure 97. uSD-Card interface block diagram.

15.1.9.10 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.

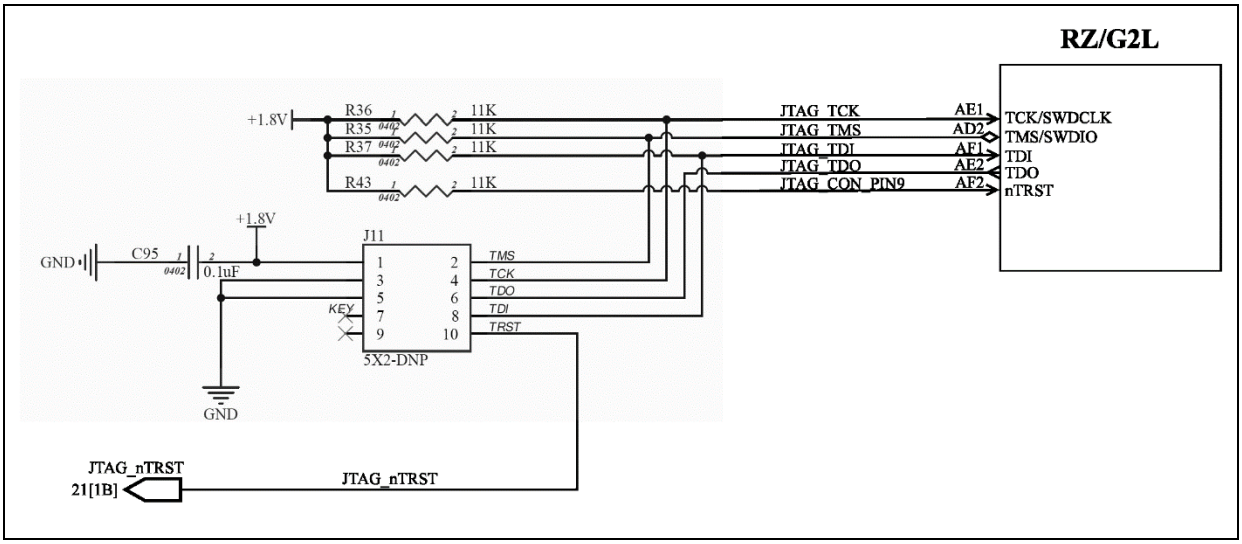


Figure 98. JTAG/SWD Block Diagram

15.1.9.11 Expansion Connector

The RZ/G2L-SBC has two connectors at 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.

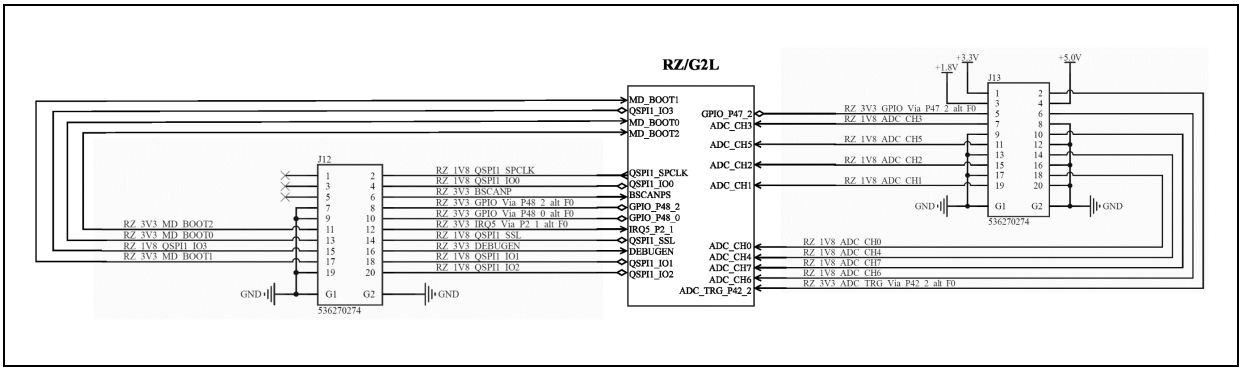


Figure 99. Block diagram of Bottom Connectors.

Refer to the appendix for details on the adapter board and flashing tools.

15.1.10 Memory

The RZ/G2L-SBC design uses four types of memory.

1. QSPI NOR Flash
2. DDR4 SDRAM
3. EEPROM
4. SD-Card

15.1.10.1 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.

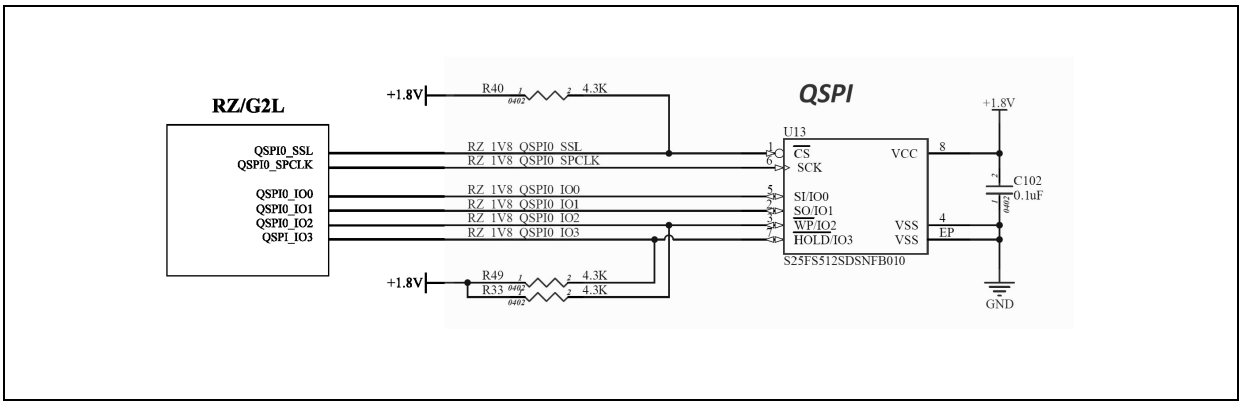


Figure 100. QSPI interface.

Note: The pull-up resistor on the clock line “QSPI0_SPCLK” is optional and is omitted in this design.

15.1.10.2 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.

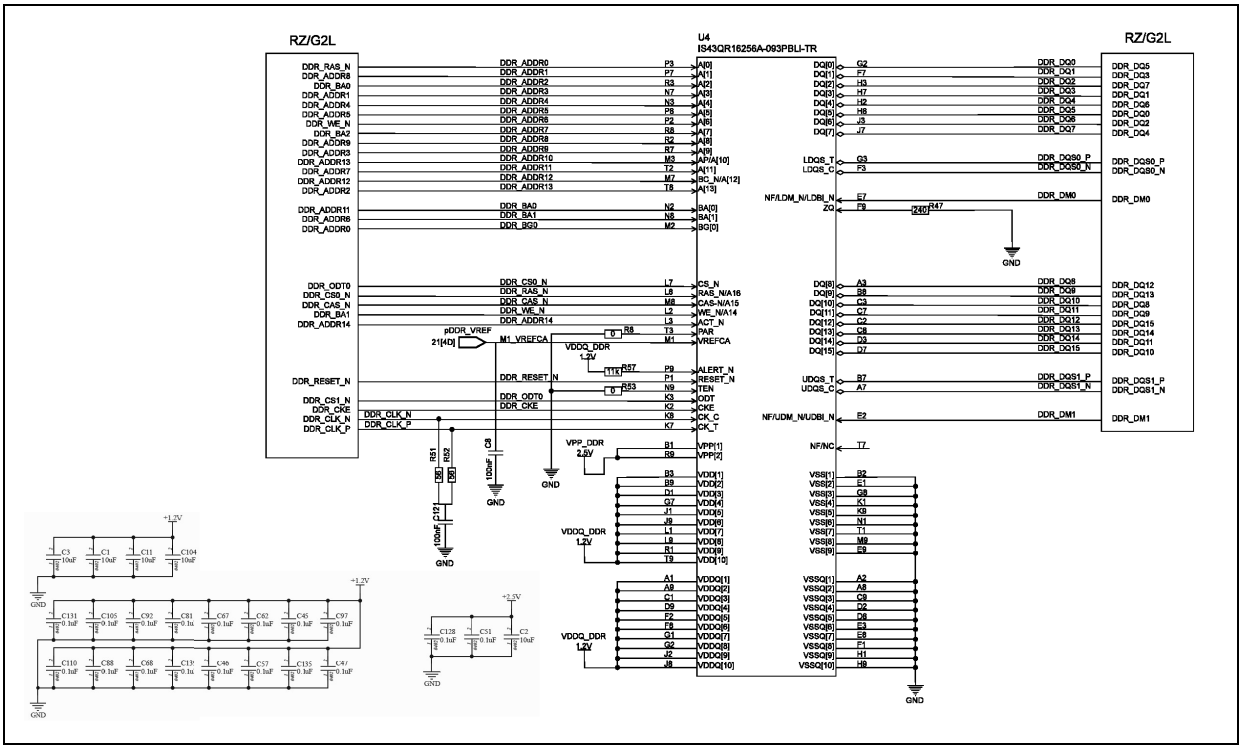


Figure 101. DDR4 SDRAM Interface

15.1.10.3 EEPROM with Temperature Sensor.

The RZ/G2L-SBC has an onboard CAT34TS02 I²C 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 I²C interface. The EEPROM is configured as 16 pages of 16 bytes each for a total of 256 KiB (2 kilobytes) of memory. Currently, two MAC IDs occupy 6 bytes of memory each for a total of 12 bytes.

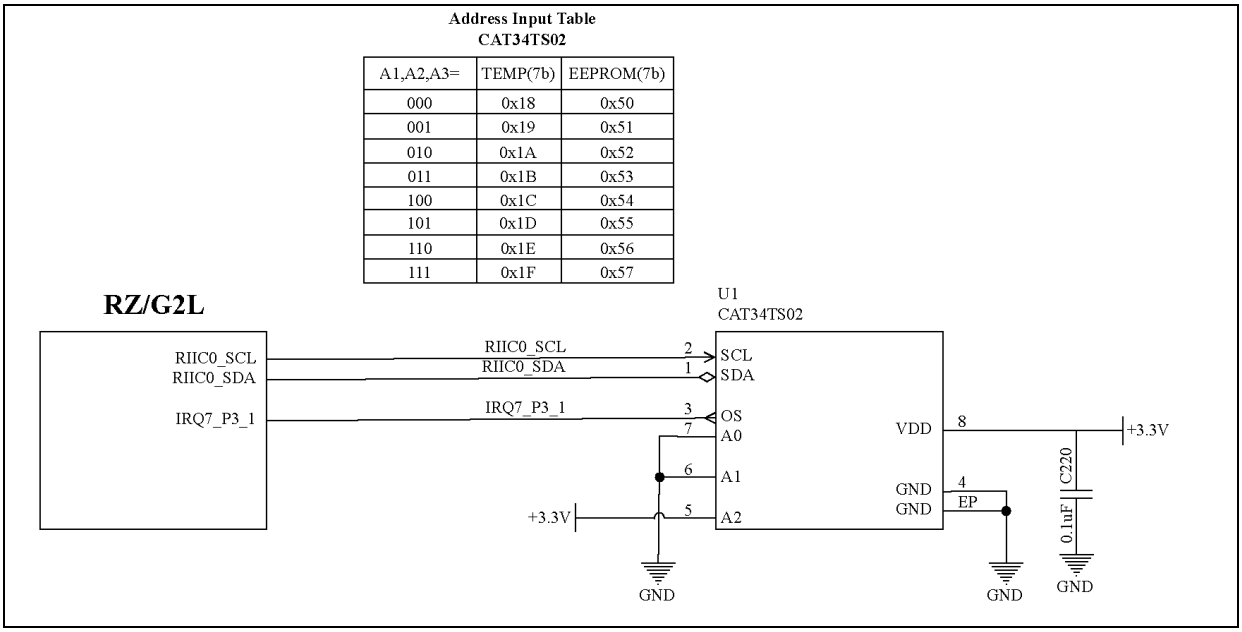


Figure 102. I2C EEPROM Block Diagram

Table 15. EEPROM Parameters

Parameter	Value	Description
I²C 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 I²C	Three programmable trigger settings for high, low, and critical temperatures to raise interrupt over line IRQ 7.

15.1.11 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.

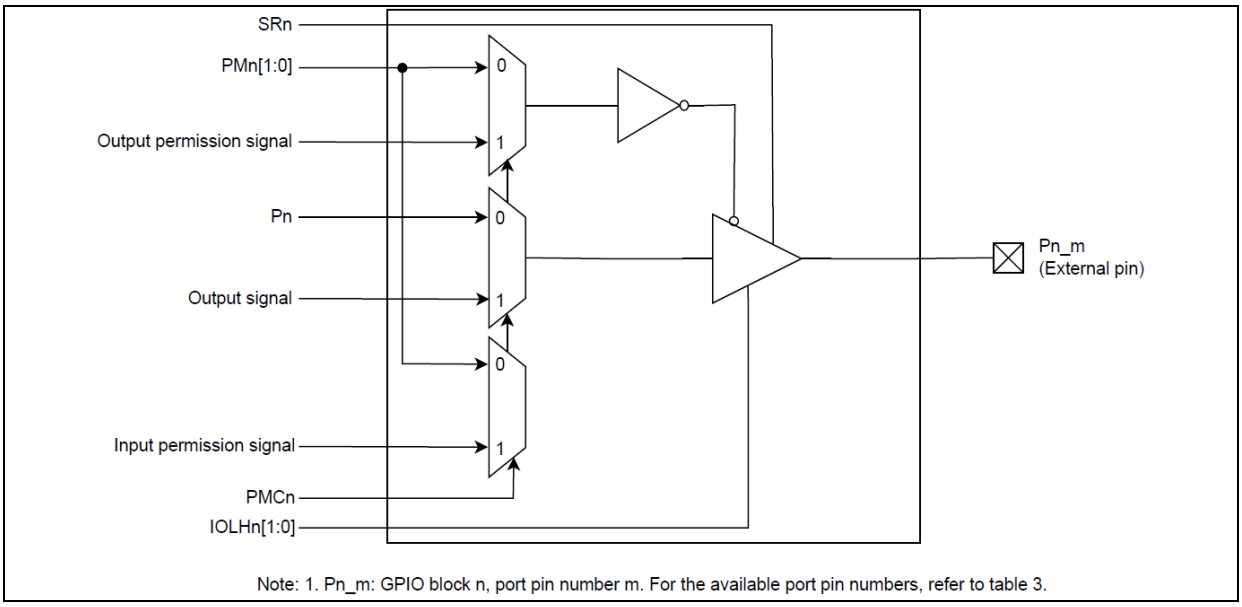


Figure 103. Multiplexed peripheral functions configuration diagram for GPIO pins

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 16. 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 9.1.2. .

16. Appendix

16.1 Factory Firmware Flashing Using Serial Downloader (SCIF) Mode

In most cases, the RZ/G2L-SBC comes preloaded with the latest firmware. The preferred method of updating the firmware is through the SD card flashing method, as described in section 8. .

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.

16.2 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.

16.2.1 Required Hardware

This flashing process requires the use of boot mode change, which is achieved using an adapter board that is not included in the package.

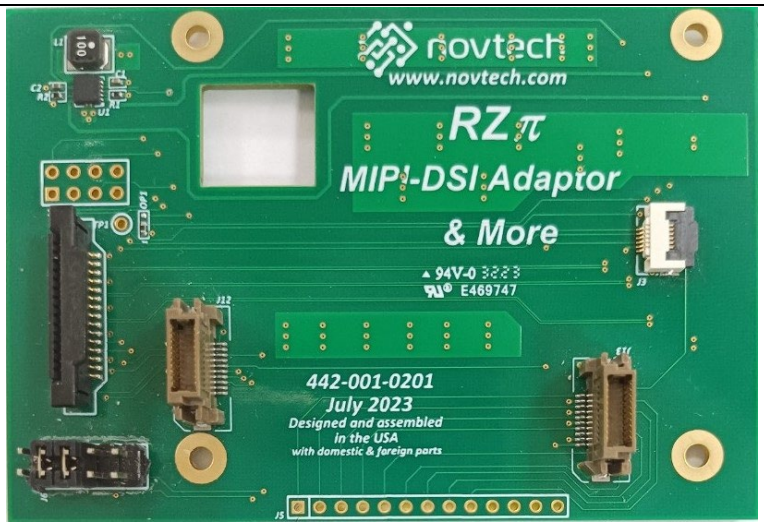


Figure 104. Adaptor board

16.2.2 Flashing Bootloader/Firmware Using Linux Host

The building contains a support script, `bootloader_flash.py`, for flashing the bootloader on Linux. The script is part of the Yocto build. The official release is a qualified Yocto build from Renesas and is a full package with all tools and scripts.

The Python script is present at the root of the release directory.

Run the following command to learn how to use the script:

```
$ ./bootloader_flash.py -h
```

Before performing a flashing:

- ✓ Make sure the board is powered off,
- ✓ Connect the debug serial port (SCIF0 - TXD, RXD, GND) to your Linux PC
- ✓ Connect the adapter board with the jumpers set to serial load boot mode.

By default, the script uses /dev/ttyUSB0 when no arguments are passed.

Here are the steps:

1. Ensure that the hardware setup is accurate.
2. Start the script.
3. Power on the board

```
renesas@builder- pc:~/renesas/rz-cmn-
srp/yocto_rzsbc_board/build/tmp/deploy/images/rzg2l-sbc/host/tools/bootloader-
flasher/linux$ ./bootloader_flash.py
Please power on board. Make sure you changed switches to SCIF download mode.
 SCIF Download mode
 (C) Renesas Electronics Corp.
-- Load Program to System RAM ---------------
please send !
Writing Flash Writer application...


Flash writer for RZ/G2 Series V1.06 Aug.10,2022
 Product Code : RZ/G2L
>
Elapsed time: Flash Writer: 23.976105 seconds
true
command not found
>
SUP
Scif speed UP
Please change to 921.6Kbps baud rate setting of the terminal.

>

>
>XLS2
===== Qspi writing of RZ/G2 Board Command =============
Load Program to Spiflash
Writes to any of SPI address.
 ISS : IS25WP256
Program Top Address & Qspi Save Address
===== Please Input Program Top Address ============
  Please Input : H
```

```
'11E00
===== Please Input Qspi Save Address ===
  Please Input : H
'
  Please Input : H'00000
Work RAM(H'50000000-H'53FFFFFF) Clear....
please send !
Writing BL2...
 ('.' & CR stop load)
SPI Data Clear(H'FF) Check :H'00000000-0000CFFF,Clear OK
H'00000000-0000CFFF Erasing..............Erase Completed
SAVE SPI-FLASH.......
======= Qspi  Save Information  =================
 SpiFlashMemory Stat Address : H'00000000
 SpiFlashMemory End Address  : H'0000CB28
============================================================

>

>XLS2
===== Qspi writing of RZ/G2 Board Command =============
Load Program to Spiflash
Writes to any of SPI address.
 ISS : IS25WP256
Program Top Address & Qspi Save Address
===== Please Input Program Top Address ============
  Please Input : H
'00000
===== Please Input Qspi Save Address ===
  Please Input : H
'1D200
Work RAM(H'50000000-H'53FFFFFF) Clear....
please send !
Writing fip ...
 ('.' & CR stop load)
SPI Data Clear(H'FF) Check :H'0001D000-000D7FFF,Clear OK
H'0001D000-000D7FFF
Erasing.....................................................................
.............................................................
Closed serial port.
Elapsed time: 81.550220 seconds
```

Power cycle the board after the script is completed.

16.2.3 Flashing Bootloader/Firmware Using Windows Host

The subdirectory `windows` from Yocto build output/release directory contains the Windows scripts. The Windows tool has its own `Readme.md` file with the necessary information about the scripts.

Before performing a flashing:

- ✓ Make sure the board is powered off,
- ✓ Connect the debug serial port (SCIF0 - TXD, RXD, GND) to your Linux PC
- ✓ Connect the adapter board with the jumpers set to serial load boot mode.
- ✓ Ensure that Tera-Term application is installed on your Windows PC.

To boot firmware using Windows Host:

1. Ensure that the hardware setup is accurate.
2. Edit config.ini and set the correct com port number.
3. Start the script flash_bootloader.bat.
4. Power on the board.

The script uses Tera Term's TTL to complete the flashing of the firmware. Upon completion, it will disconnect the port.

Power cycle the SBC to boot new firmware.

16.3 How To Get the Console After Bootup

Once the RZ/G2L-SBC 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.

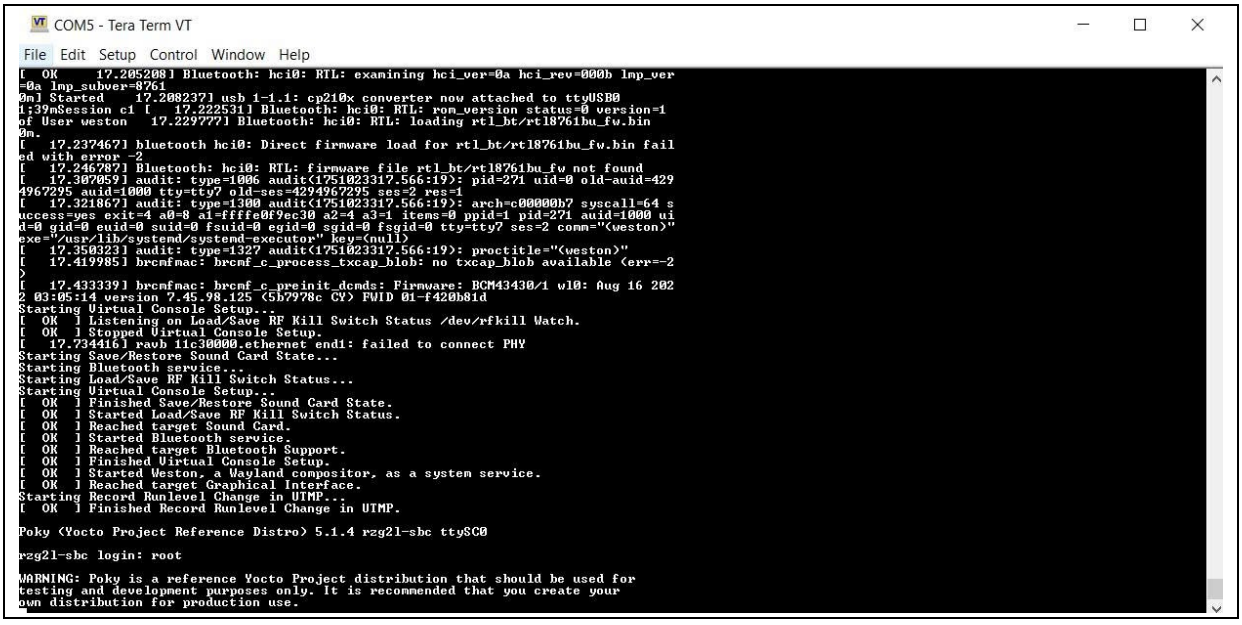


Figure 105. Root login of Linux console over UART 0.

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 Programming/Flashing Firmware to RZ/G2L-SBC section 'Flash bootloader on u-boot console and in the appendix section 'Flashing Bootloader/Firmware using Linux host'.

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
```

The above commands try to update packages on your Linux machine to the latest. Then, they install the python3 package and the python3 pip tool, which is used to install python3's modules. And finally, they install the necessary modules ('pyserial' and 'argparse') with the specific versions for running 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: RZ/G2L-SBC
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:   eth0: ethernet@11c20000, eth1: ethernet@11c30000
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 8.1.2 Flash bootloader on u-boot console. 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)

18.2 RZ/G2L SoC

Product page: RZ/G Series (Linux-based MPU) | Renesas

Wiki: RZ/G Series 32/64-bit MPU - Renesas-wiki

Other RZ topics: RZ Topics - Renesas-wiki

18.3 External Resources

18.3.1 QT Development

Qt official page: Qt | Tools for Each Stage of Software Development Lifecycle

Qt documentation: Qt Documentation | Home

18.3.2 Yocto Project

Official Yocto manual: Yocto Project Reference Manual — The Yocto Project ® 5.1.4 documentation

18.3.3 Linux Kernel Documentation

The Linux Kernel Documentation — The Linux Kernel documentation

18.3.4 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

18.3.5 JEDEC DDR4

DDR4 SDRAM STANDARD | JEDEC

18.3.6 PMOD Specification

Wiki: Pmod Interface - Wikipedia

Specification document: pmod-interface-specification-1_3_1.pdf (digilent.com)

18.3.7 Essential Linux Tutorial

Linux/Unix Tutorial (geeksforgeeks.org)

UNIX / LINUX Tutorial (tutorialspoint.com)

18.3.8 Packaging

CMake Reference Documentation — CMake 3.30.2 Documentation

CPack — CMake 3.30.2 Documentation

18.3.9 Using Extensible SDK

Using the Extensible SDK

18.3.10 Install Eclipse IDE

Eclipse Installer 2024-09 R | Eclipse Packages

18.3.11 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

18.3.12 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.09.25	—	Yocto Styhead release

System Release Package, RZ Series – User Manual

Publication Date: Jul.09.25

Published by: Renesas Electronics Corporation

RZ Family/ RZ/G Series

Renesas Electronics Corporation

R12UZ0195EU0200