User's Manual

# RZ/G2L-SBC, Single Board Computer

## User's Manual: Hardware and Software

Renesas Microprocessor
RZ Family RZ/G Series

OPN
US157-G2LSBCPOCZ



Renesas Electronics
www.renesas.com

Rev.1.20 OCT, 2024

# 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.

## Introduction

This user manual describes the RZ/G2L based single board computer. This system architecture is of a generic single-board computer based on the Renesas RZ/G2L series SoC. It is a fully capable general purpose computer module aimed at HMI, industrial, and robotics applications.

This SBC features a Renesas RZ/G2L MPU as the main processor and runs a Linux distro built on Yocto OE using the Renesas VLP 3.0.5 package as its source.

One of the key highlights is the ease at which the board can be used to quickly create a PoC using a wide array of peripheral ports and proven accessories / modules. It comes equipped with an extensive set of features and interfaces, including onboard Wi-Fi, PMOD interface and dual ethernet ports.

## Features

The RZ/G2L-SBC board contains the following features:

- RZ/G2L or Dual core Cortex®A55 SoC with on-chip Cortex M33 core for real time applications.
- 1 GiB DDR4 (single chip of 4 Gbit)
- Micro SD card socket for OS image and rootfs
- QSPI for boot
- Temperature sensor with on-chip EEPROM holding board configuration data.
- Onboard Laird 802.11 Wi-Fi/BT module
- 40-pin Header connector (Raspberry Pi 3B compatible)
- Four USB 2.0 Type-A ports
- Dual Gigabit Ethernet ports
- 3.5mm Audio Port
- Mini- HDMI supporting full HD displays.
- MIPI-CSI port (Arduino compatible)
- MIPI-DSI port (Raspberry Pi compatible)
- Dual expansion ports for adapter board interfacing:
    - 40-pin DSI display modules.
    - 6 pin I2C touch modules.
    - ADC.
    - Bootstrapping.
    - External power and ground.
- USB Type-C Power connector
- Status LED indicators
- Board dimensions: 82 mm * 50 mm
- Mount: Double-sided mounting (10 layers)

## 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 cable 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. |
| 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 is 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's one of the densest chips where all the typically external devices like flash memory, DDR RAM and even Wi-Fi module are all packaged into a single chip. These usually 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 number of other peripherals and even 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 do require 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. |

# RZ/G2L-SBC, Single Board Computer

## 1. Overview

RZ/G2L-SBC Board 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.

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 audio codec and 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 CIP Kernel-based Linux BSP (maintained for 10 years+) plus reference designs that highlight 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.

## 1.1 Physical View

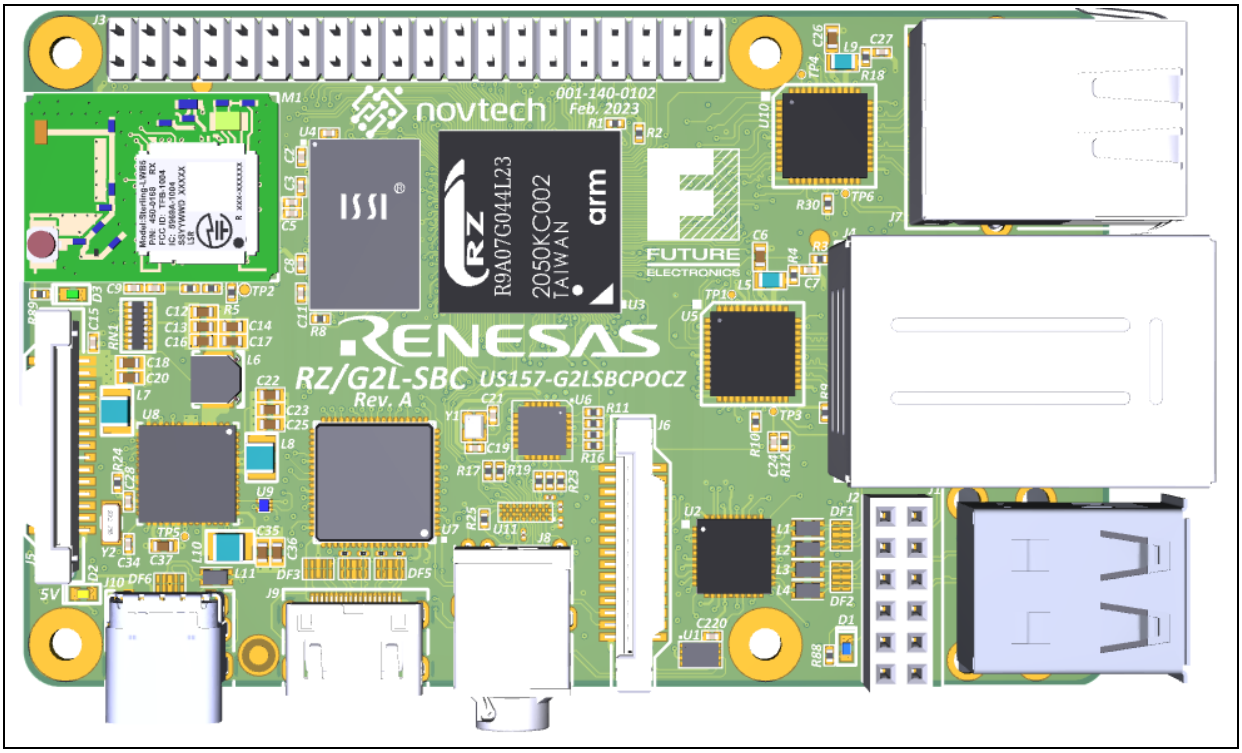



**Figure 1: Top side view of the RZ/G2L-SBC**

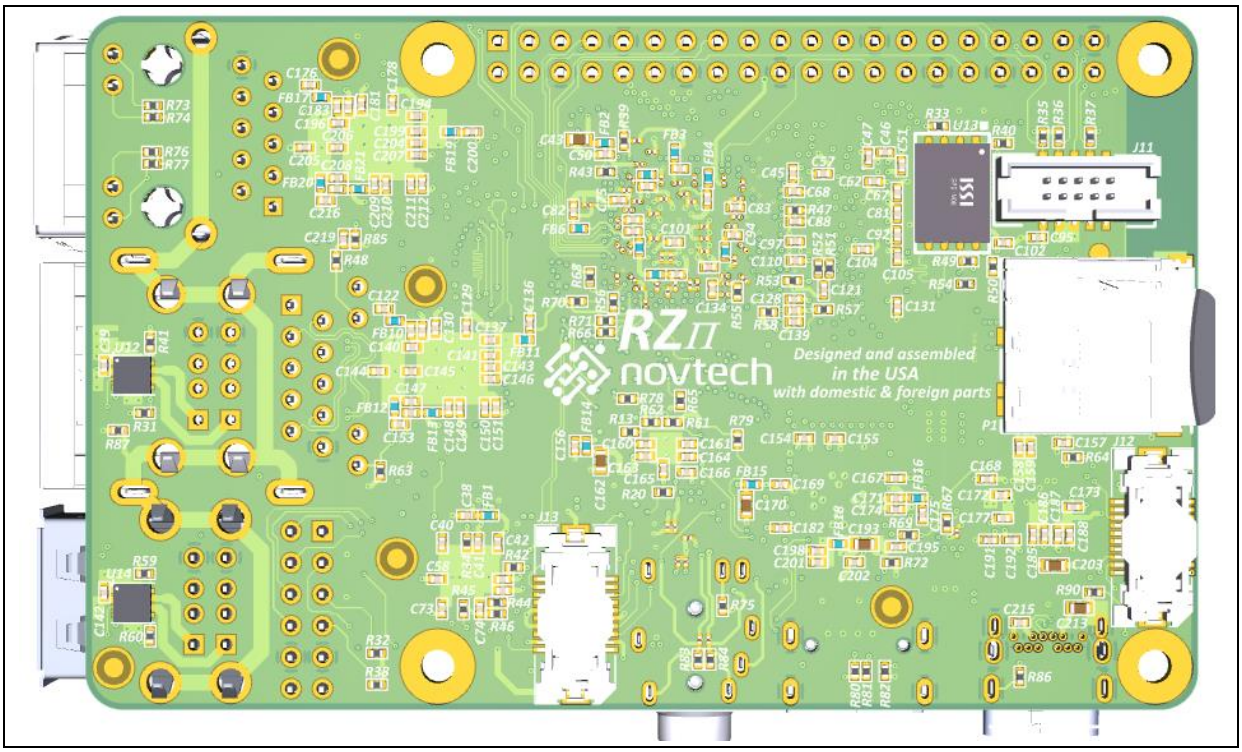



**Figure 2: Bottom side of the RZ/G2L-SBC**

# 2. Required Resources

## 2.1 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 (Download File List - Tera Term - OSDN) on Windows PC for accessing UART.
- Minicom on Ubuntu host machine for accessing UART on Linux.
- Balena Etcher

## 2.2 Hardware

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

- Renesas RZ/G2L Board.
- Windows PC with Tera Term software and admin privileges.
- Ubuntu 20.04 host environment as native install or 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 package).
- OV5640 camera module (optional: not included in the package).

# 3. 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.

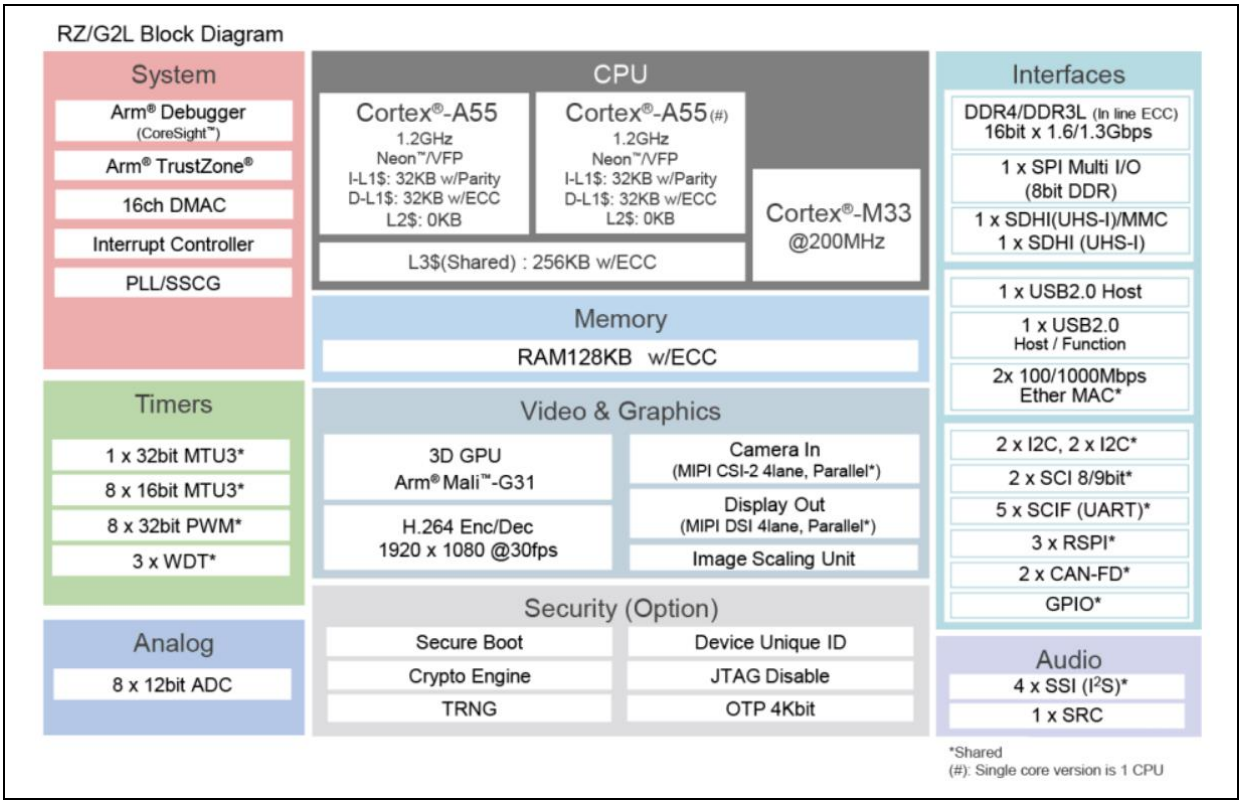



**Figure 3: RZ/G2L SoC (System on Chip) Overview**

## 3.1 Operational Flow

The diagram below will show the operational flow of the RZ/G2L-SBC system during power ON.

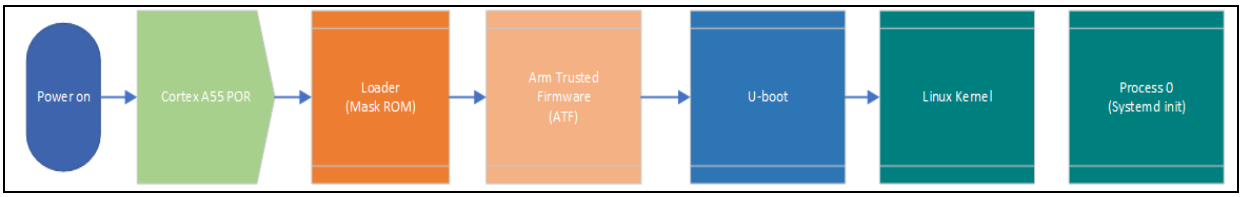



**Figure 4: RZ/G2L-SBC boot operational flow**

By default, the main processor will be in power OFF state to conserve battery. 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 the Linux booting into user space.

While u-boot passes full control to the Linux kernel, 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.

# 4. Functional Overview

This section delves into the functional and design aspects of the RZ/G2L-SBC. The below image highlights the key hardware components in the RZ/G2L SBC design.

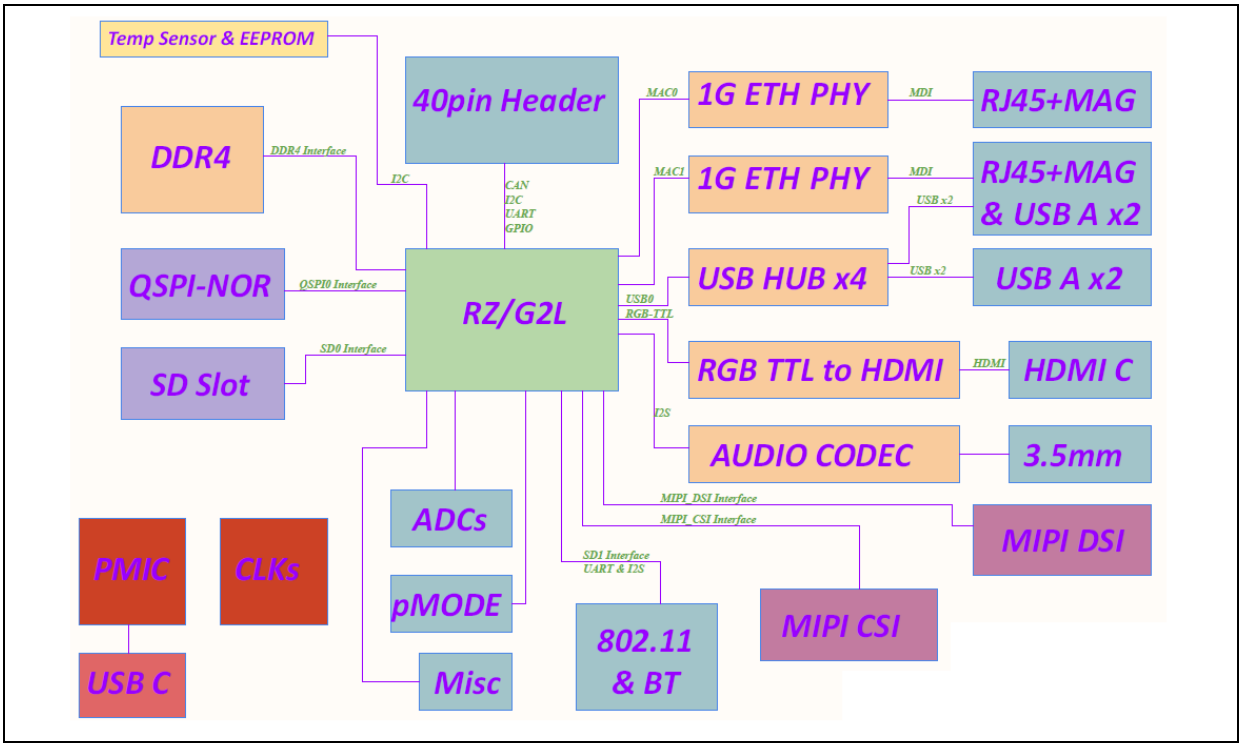



**Figure 5: RZ/G2L SBC System Overview**

**Table 1: Main Components on RZ/G2L-SBC**

| Component Number | Component Name | Type (Manufacturer) |
|---|---|---|
| U1 | Temperature Sensor Digital, Local -55°C ~ 125°C 11 b 8-HWSON (2x3) | CAT34TS02 (Onsemi) |
| U2 | USB Controller | UPD720115K8-611-BAK-A-ND (Renesas Electronics) |
| U3 | MPU RZ/G2L | R9A07G044L23GBG (Renesas Electronics) |
| U4 | DDR4 SDRAM 512MB | IS43QR16256A-093PBLI-TR (ISSI) |
| U5 | Ethernet Phy 10BASE-TE, 100BASE-TX, 1GBASE-T | PEF7071VV16 (MaxLinear) |
| U6 | VersaClock® Programmable Clock Generator | 5P35023B-000NLGI8 (Renesas Electronics) |
| U7 | HDMI Transmitter | SiI9022A/4A – QFN (SiliconImage) |
| U8 | PMIC | RAA215300 (Renesas Electronics) |
| U9 | AND GATE 2IN SOT-23-5 Vcc 1.65V to 5.5V | 7UL1G08FS (Toshiba) |
| U10 | Ethernet Phy 10BASE-TE, 100BASE-TX, 1GBASE-T | PEF7071VV16 (MaxLinear) |
| U11 | Audio Codec with Advanced Accessory Detect | DA7219 (Renesas Electronics) |
| U12 | Dual USB Port Power Supply Controller - Covering the Industrial Temperature Range of -40C to +85C | ISL61852FIRZ (Renesas Electronics) |
| U13 | QSPI Flash 512MBIT SPI/QUAD 8WSON | S25FS512SDSNFB010 (Infineon) |

| | | |
|---|---|---|
| U14 | Dual USB Port Power Supply Controller - Covering the Industrial Temperature Range of -40C to +85C | ISL61852FIRZ (Renesas Electronics) |
| M1 | Integrated 802.11 b/g/n Wi-Fi Module | iWi-L-WB (Laird) |
| Y1 | Crystal resonator for XIN | XRCGB24M000F0L00R0 (Murata) |
| Y2 | Crystal resonator for XIN | ST3215SB32768H5HPWAA (Kyocera-AVW) |

**Table 2: 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) |

## 4.1 Overview of Connectors

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

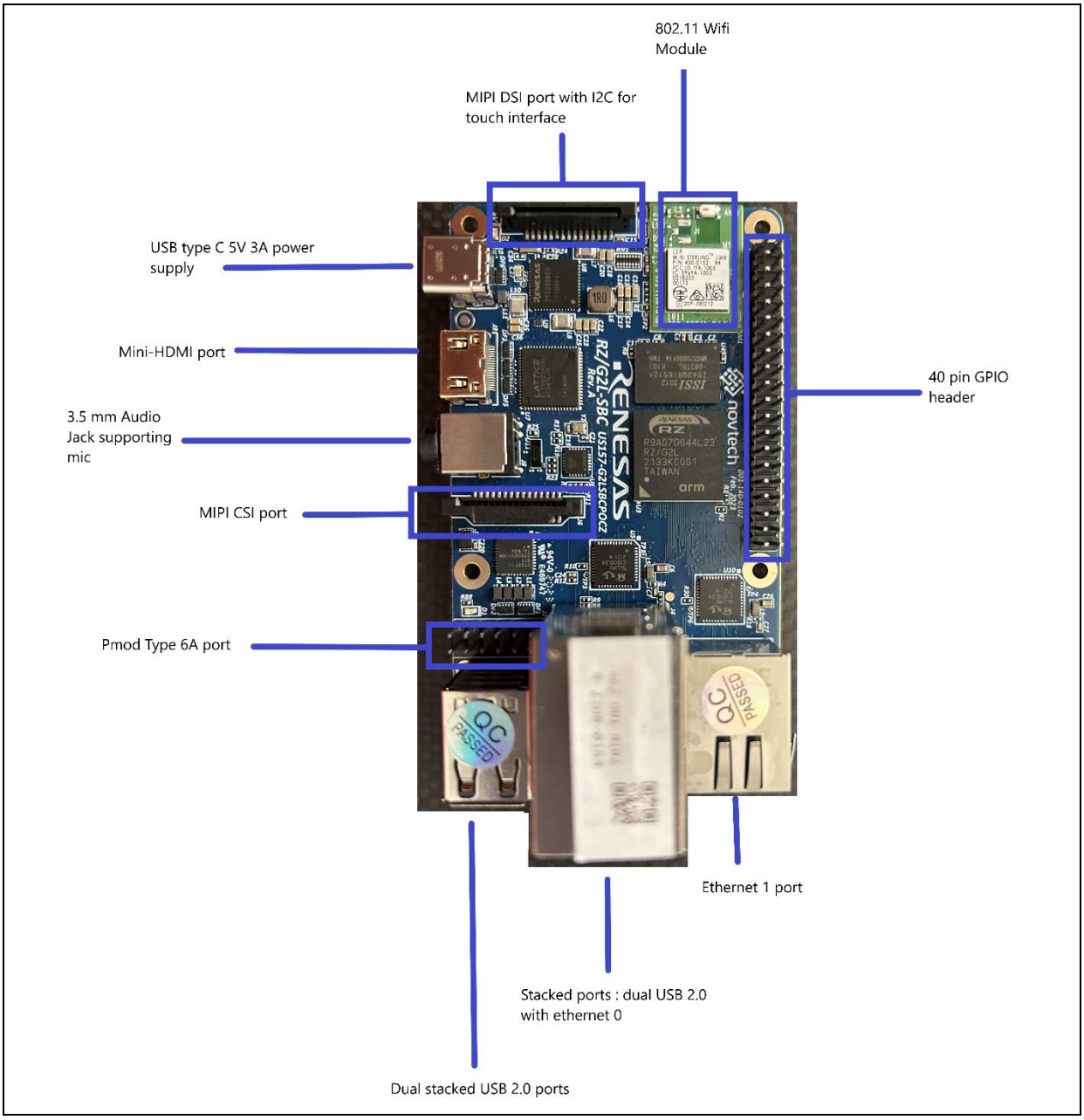



**Figure 6: RZ/G2L-SBC top side connectors.**

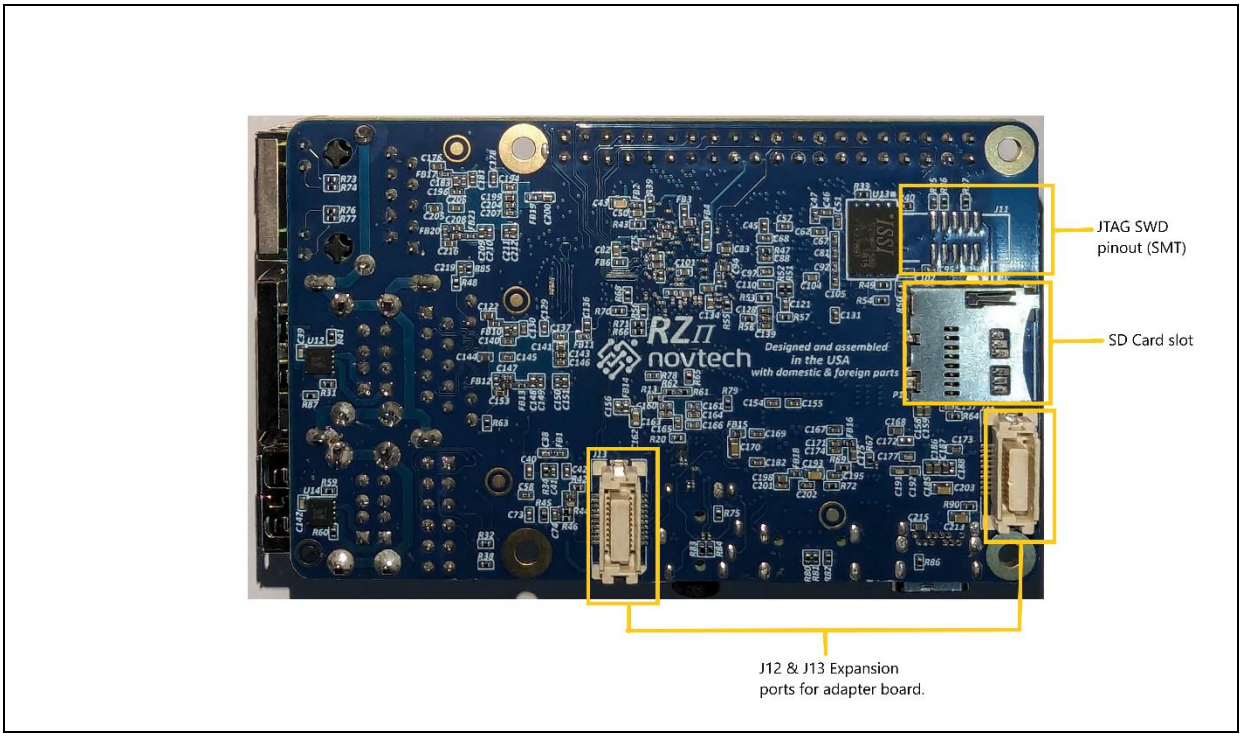



**Figure 7: RZ/G2L-SBC Bottom view connectors.**

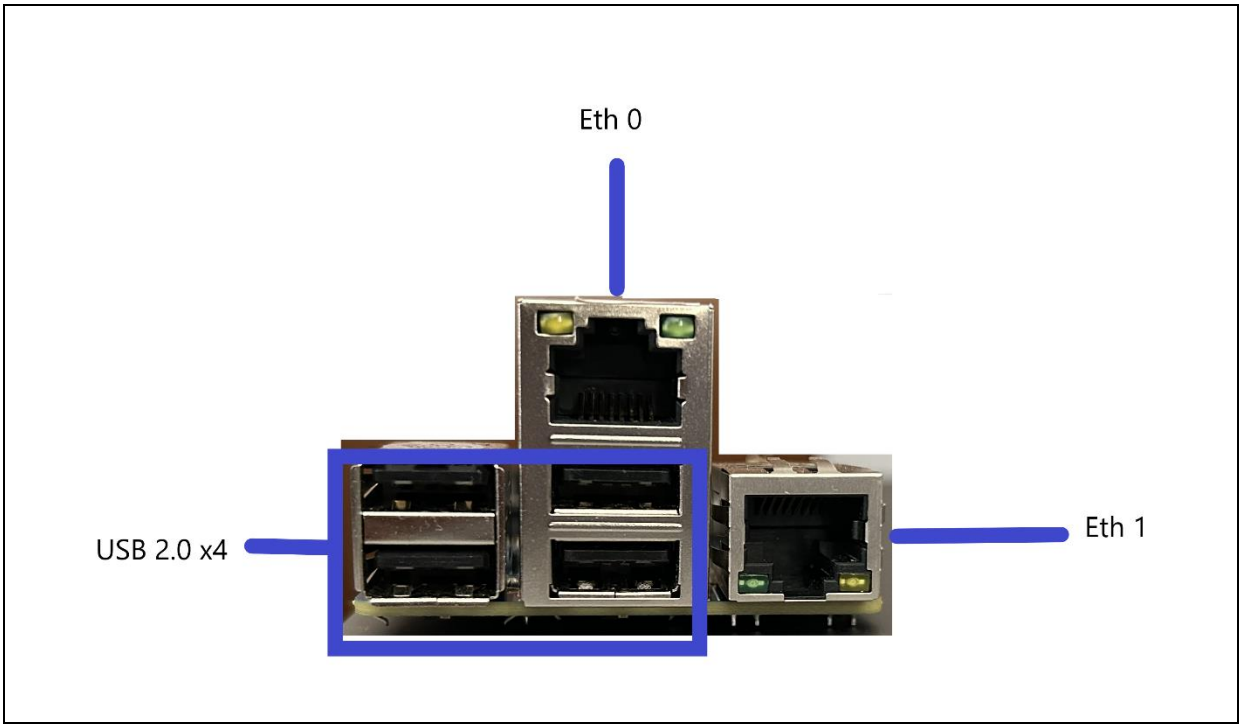



**Figure 8: RZ/G2L-SBC side view I/O ports.**

## 4.2 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.

### 4.2.1 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 consuming substantial power.

### 4.2.2 Power rails

Given below is the basic power supply design. It’s 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.

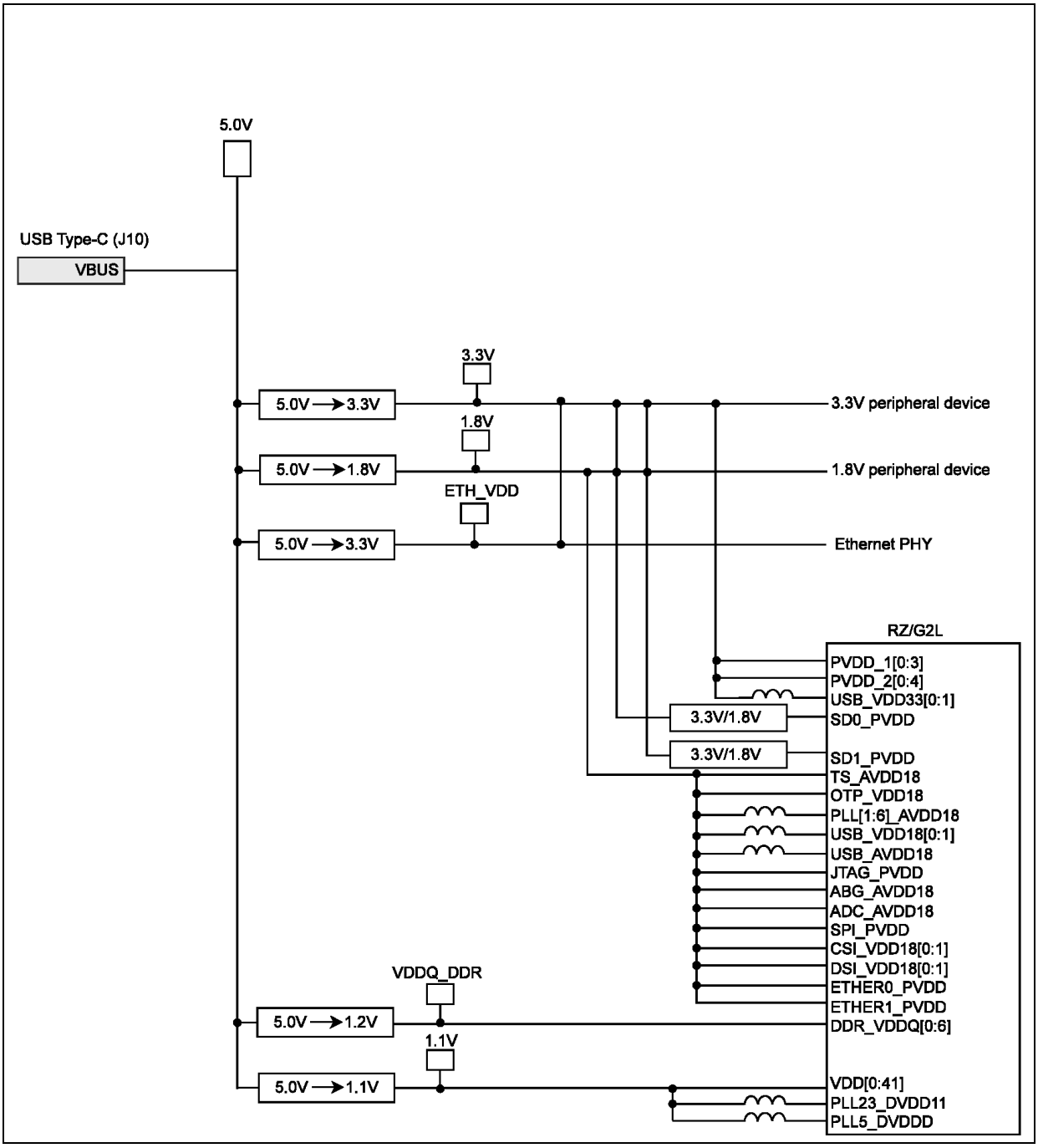



**Figure 9: Power supply rails.**

The Input power of 5V is used to generate 5 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. The POR behavior is strictly controlled by the PMIC and its passives.

### 4.2.3 Power Supply Regulation

The power supply is regulated by Renesas RAA215300 low-cost 9 channel PMIC IC.

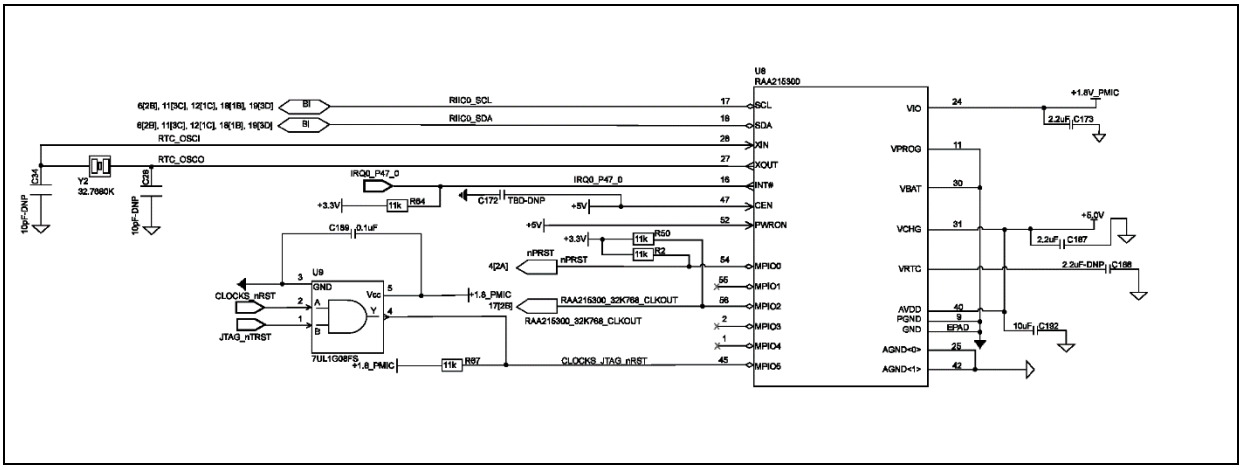



**Figure 10: Block Diagram of Power Supply Regulation using RAA215300.**

## 4.3 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 LDO's are always turned-on post PoR.

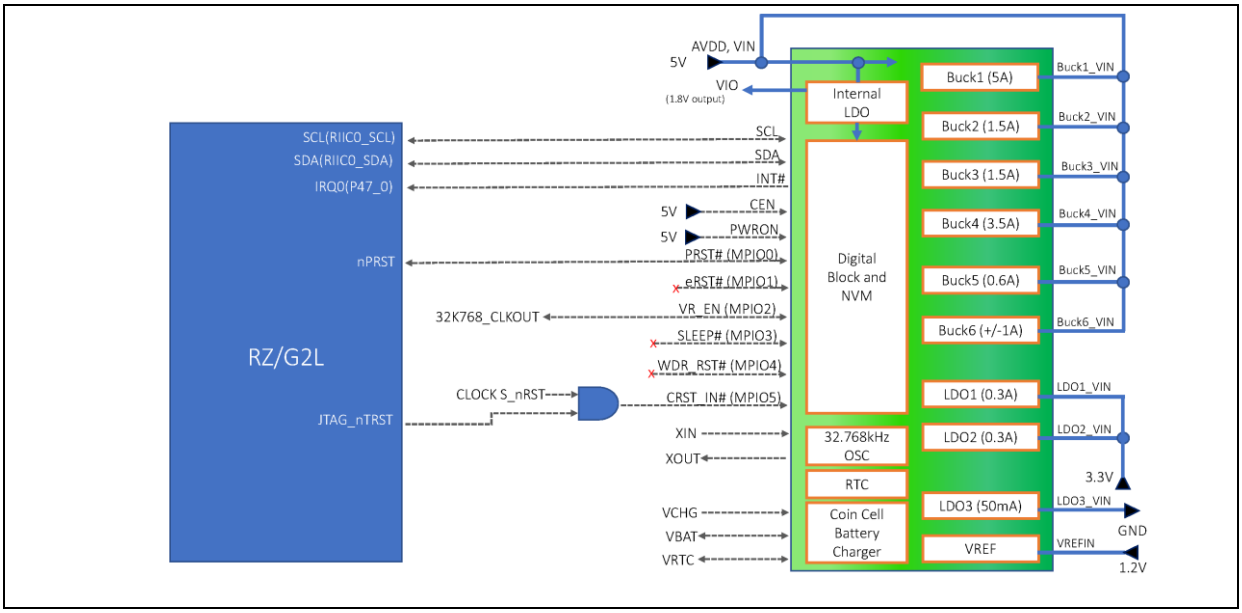



**Figure 11: Block diagram of PMIC interface to RZ/G2L**

## 4.4 RESET Control

The RZ/G2L-SBC has a simplified PoR behavior. It's 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 J12 port in 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 a debug mode. The figure 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. Never enable both.

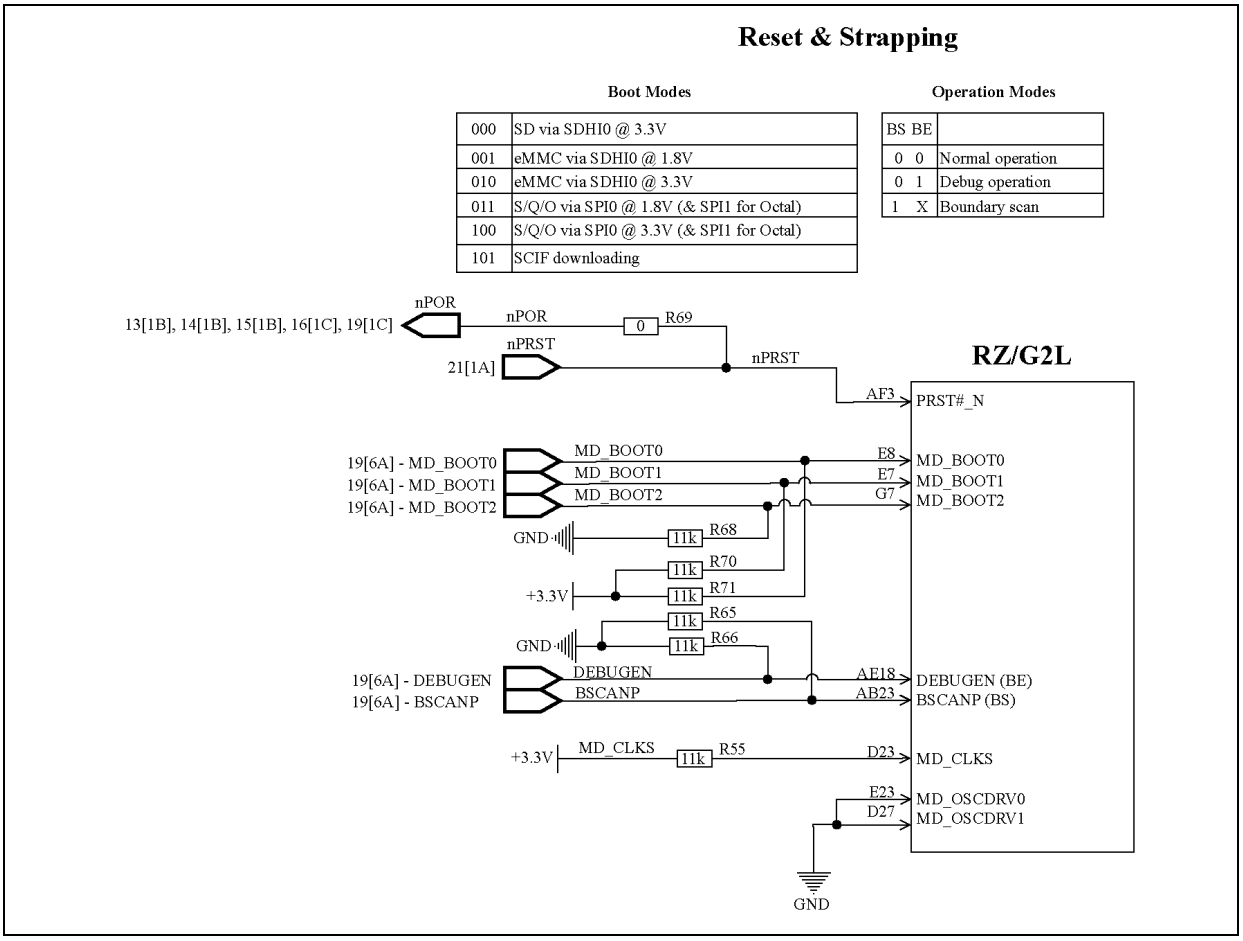



**Figure 12: Reset Control Logic**

## 4.5 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 PLL's per peripheral while using a single 24 MHz crystal XTALL.

Note:

- ✓ MIPI DSI interface supports operation up to FHD @60 fps rates.
- ✓ SD Interface supports UHS-I mode of 50MBps (SDR50) and 104MBps (SDR104)

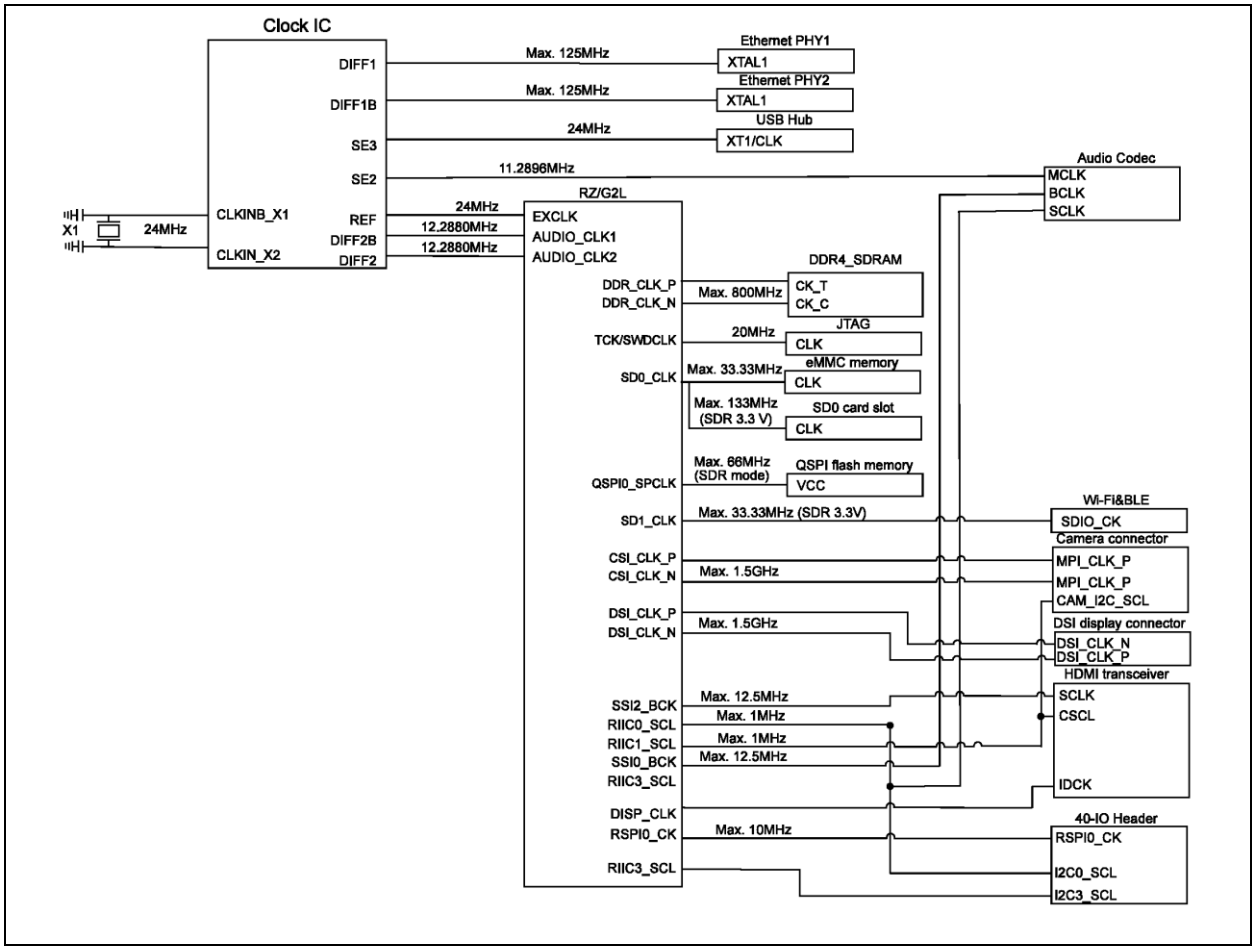



Figure 13: Block diagram of Clock interfacing.

## 4.6 Peripheral Interface

### 4.6.1 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 gigabits 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-in to 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. Please be mindful of the order.

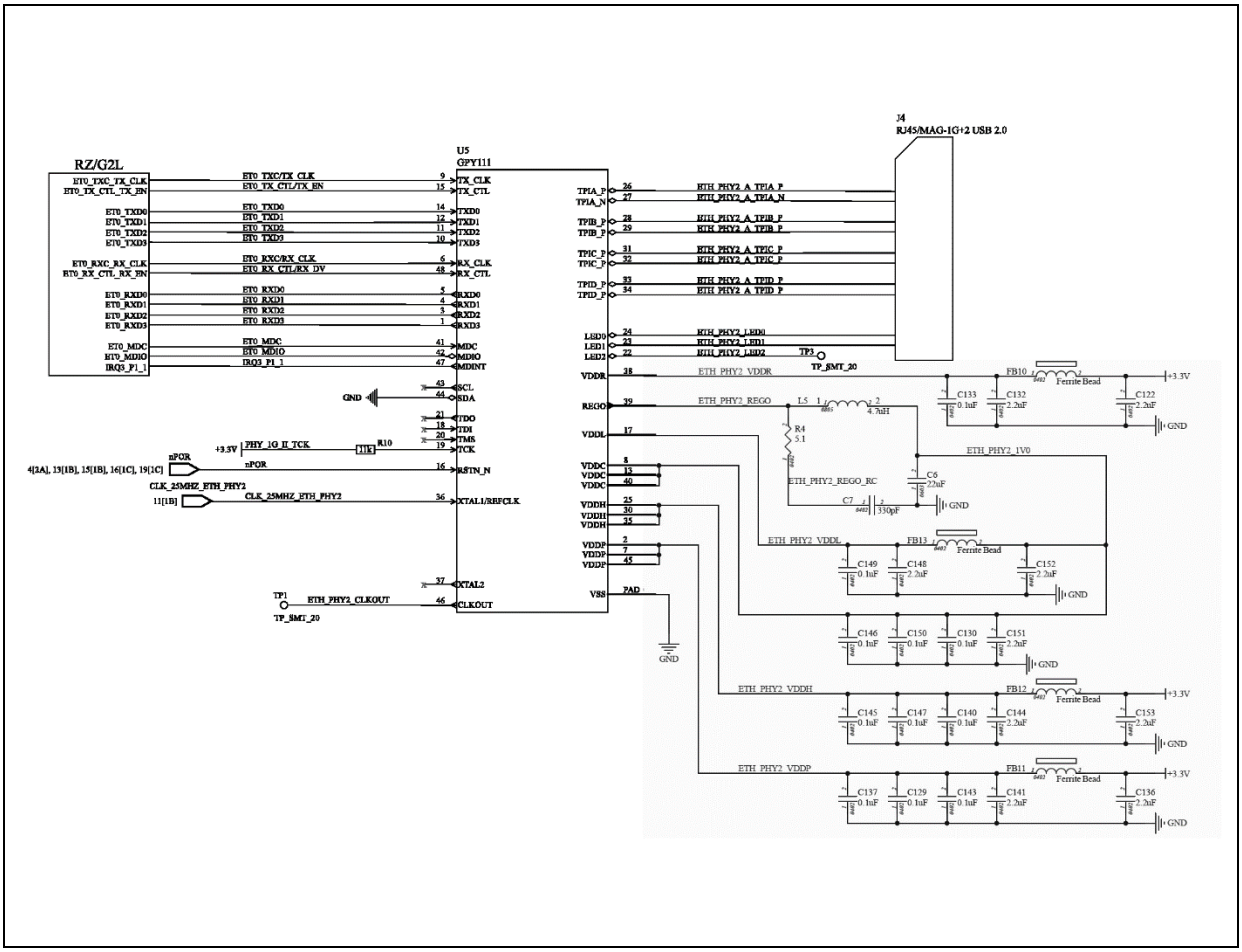



**Figure 14: Ethernet 0 PHY interfacing.**

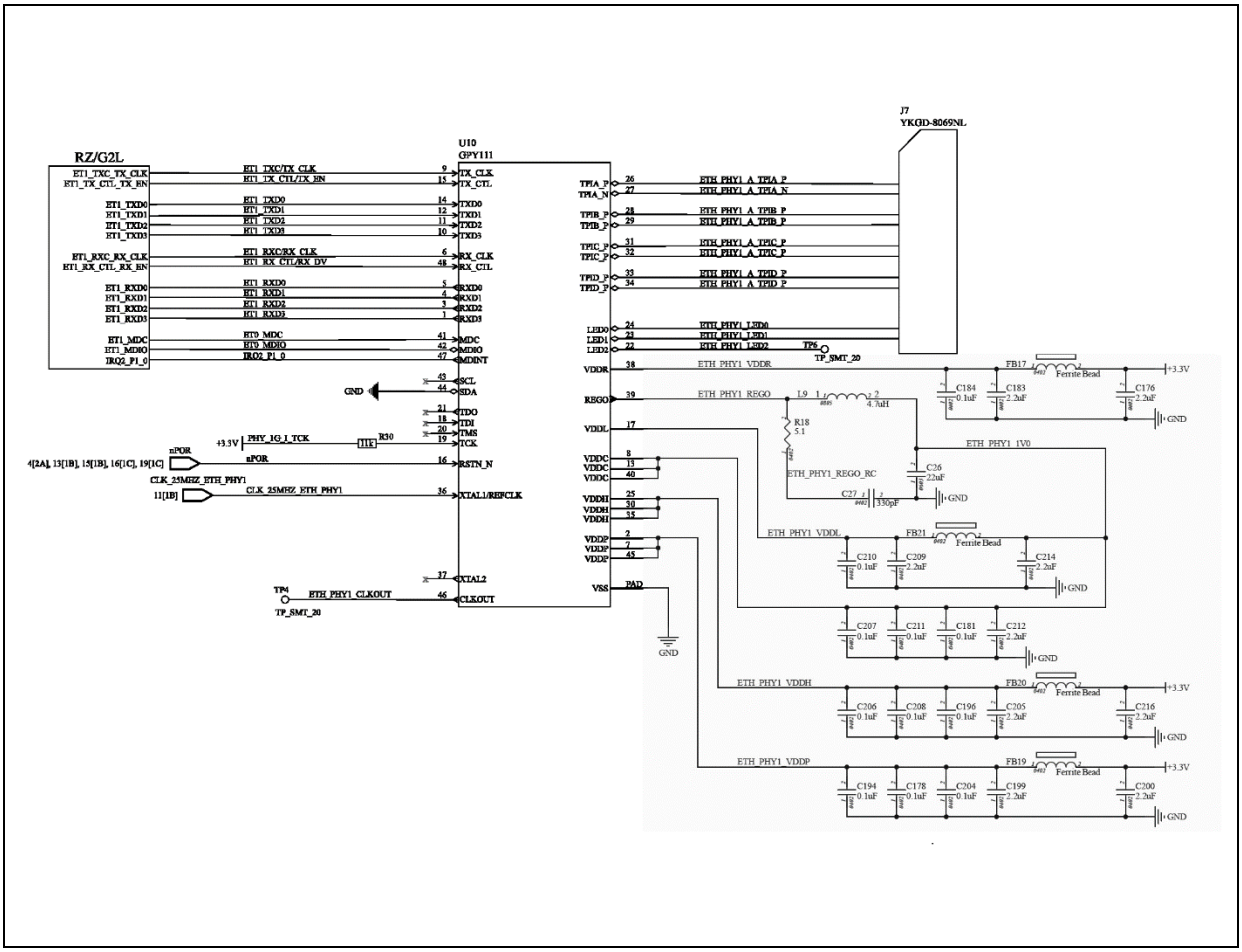



**Figure 15: Ethernet 1 PHY interfacing.**

## 4.6.2 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 4 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 is 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. Please note that 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.

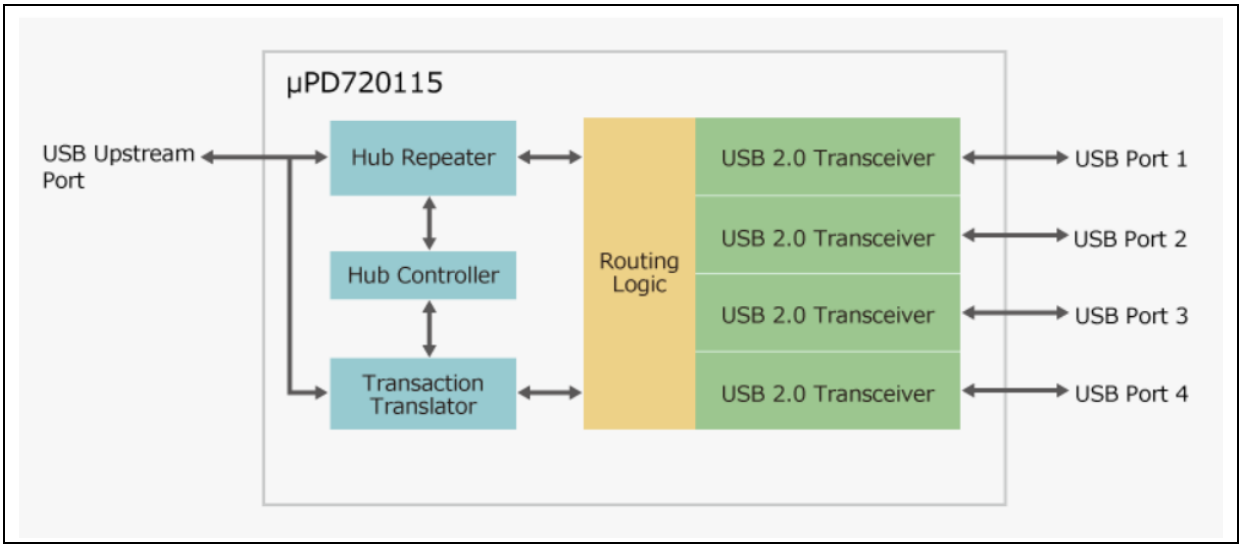



**Figure 16: UPD720115 block diagram.**

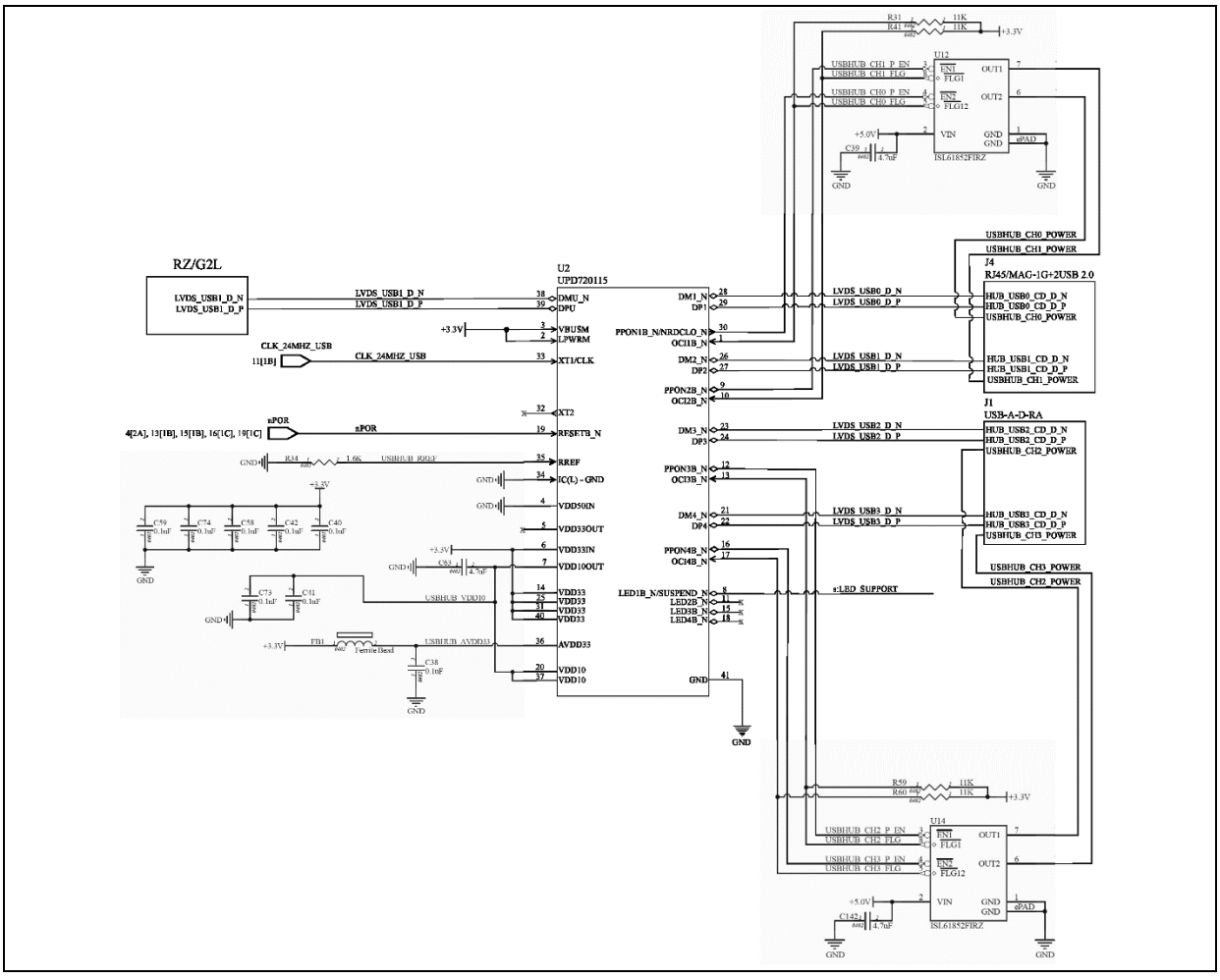



**Figure 17: USB 2.0 Hub Block Diagram**

### 4.6.3 MIPI CSI Interface

The RZ/G2L-SBC comes with a dual channel MIPI CSI port labelled as J6. It's located right next to the 3.5 mm audio jack. The CSI port 15 pin camera port is verified to work with OV5640 camera module. It supports two data channels and One I2C channel. It is directly interfaced to the RZ/G2L SoC.

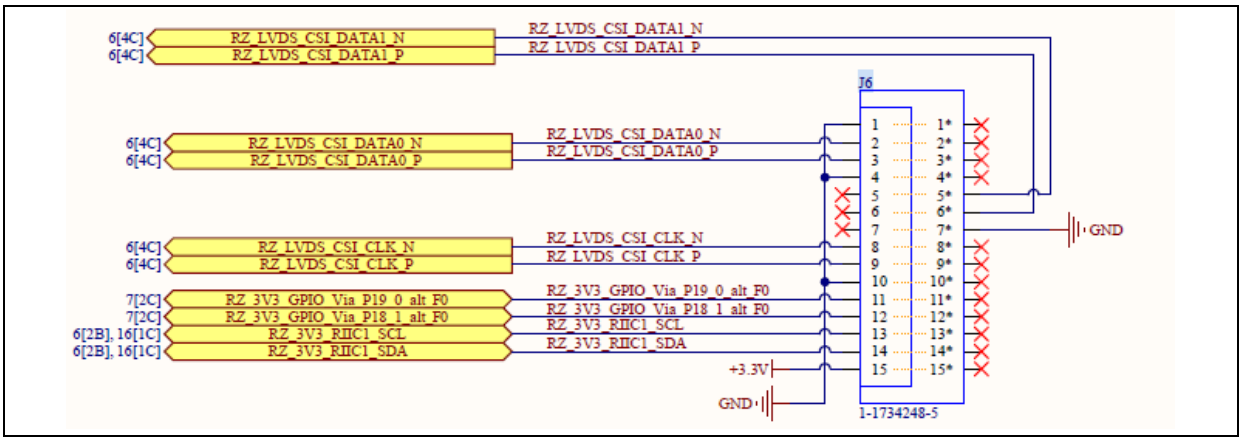



**Figure 18: CSI Interface Schematic**

### 4.6.4 MIPI DSI Interface

The RZ/G2L-SBC comes with a dual channel MIPI DSI port labelled as J5. It's 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.

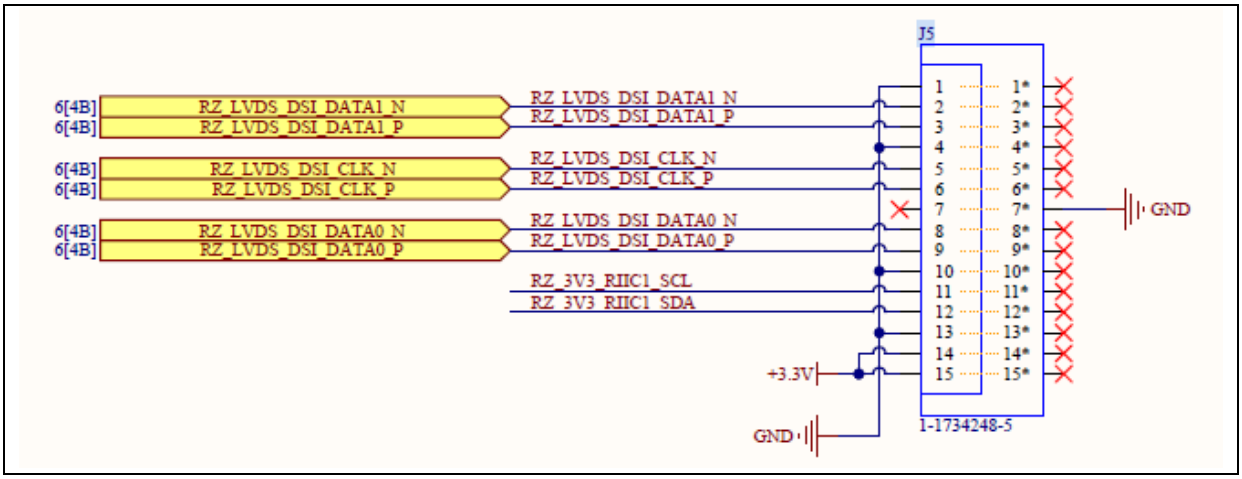



**Figure 19: DSI Schematic**

### 4.6.5 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 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.

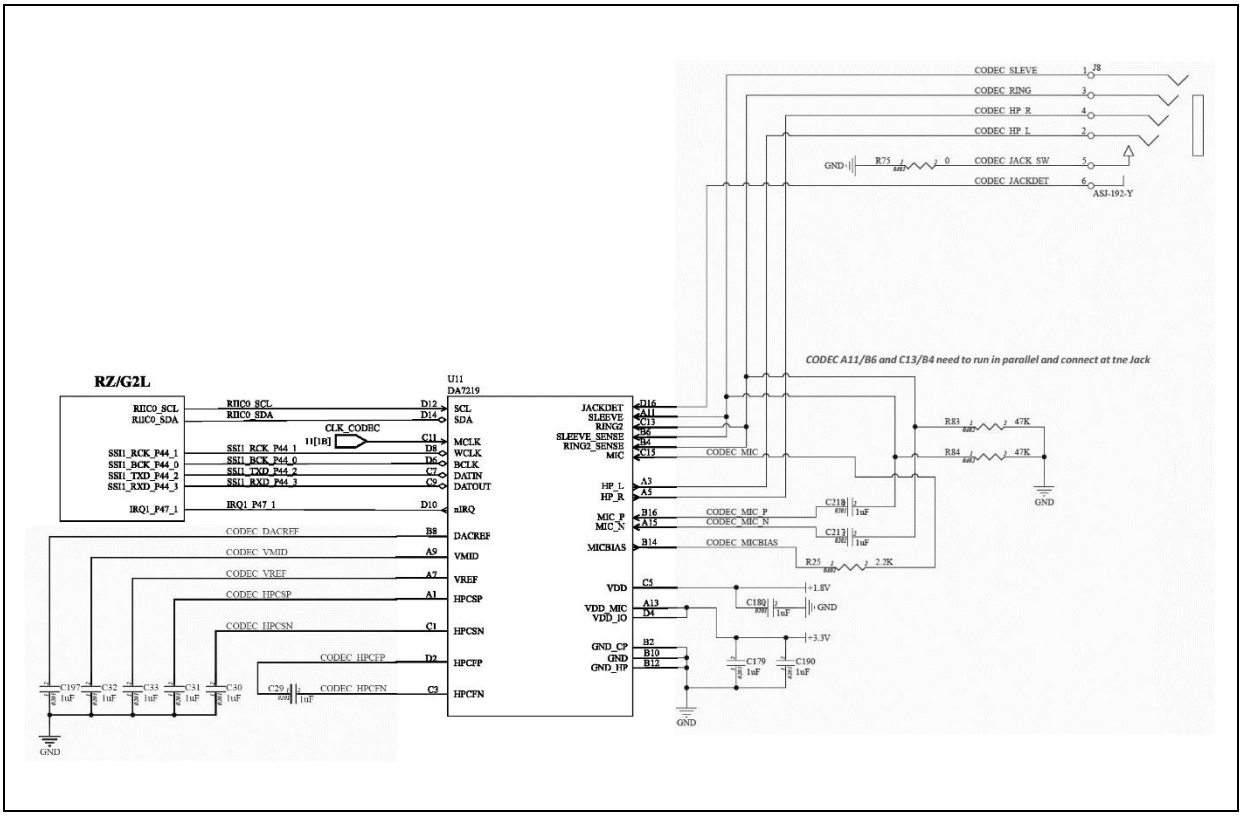



**Figure 20: Audio CODEC Interface Block Diagram**

### 4.6.6 HDMI Display Subsystem

The RZ/G2L-SBC comes with a 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. A RGB to HDMI bridge IC is used to convert RGB to HDMI protocol. The bridge is fully supported, and the HDMI is enabled with EDID feature.

Note: The LVDS interface is source for both the external HDMI bridge as well as 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 limitation with regards to the ISP unit.

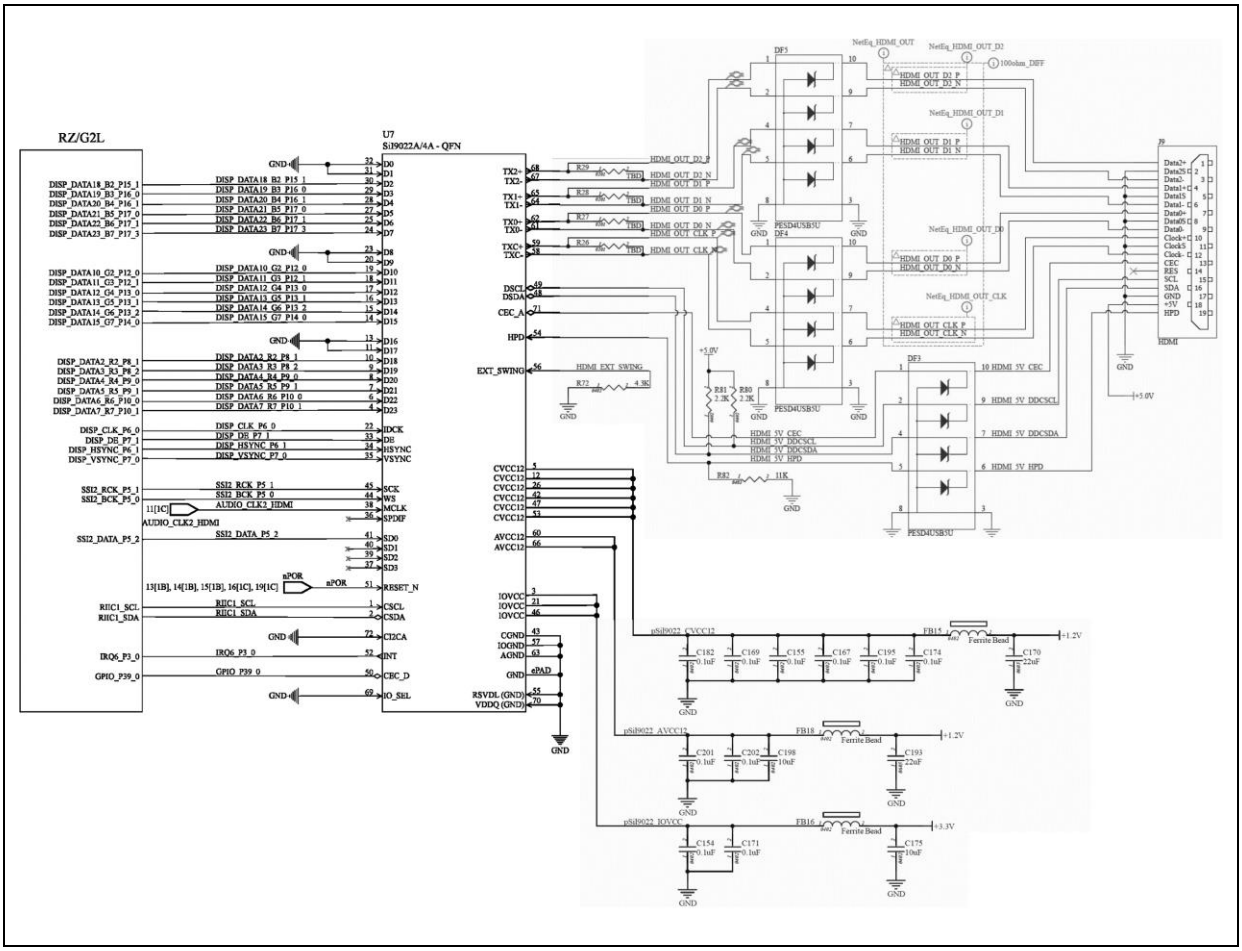



**Figure 21: HDMI Bridge and mini HDMI port interfacing.**

### 4.6.7 40-pin I/O Header

The RZ/G2L-SBC comes with a 40-pin GPIO interface which is broadly compliant with 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 of the orientation of the board.

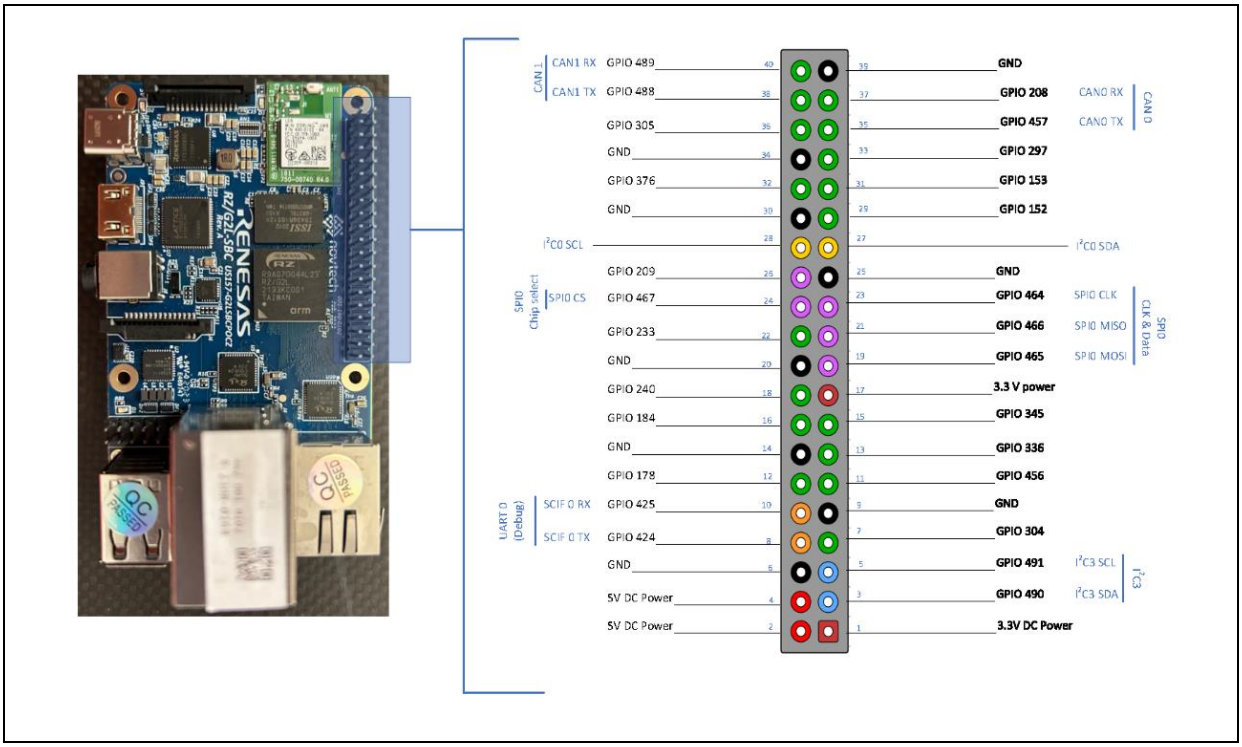



**Figure 22: 40 PIN GPIO map with orientation details.**

## 4.6.8 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.

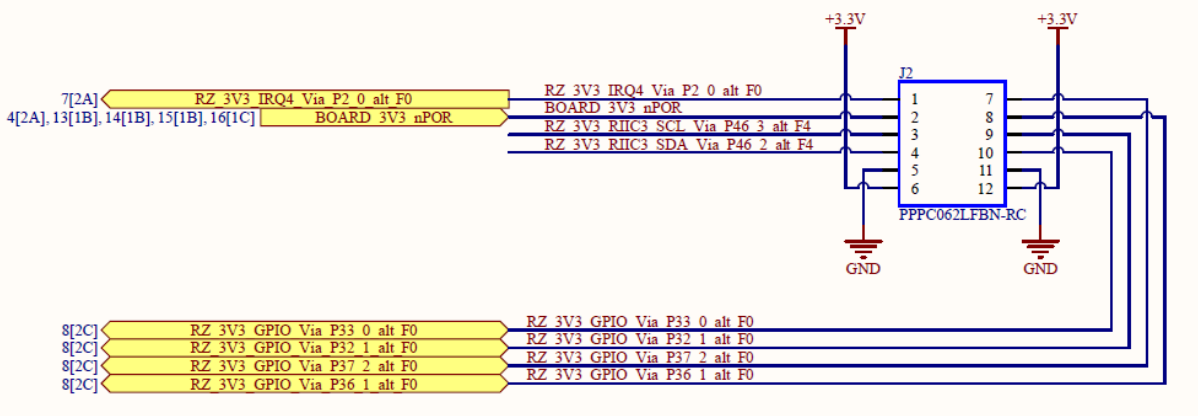



**Figure 23: Schematic of PMOD Type 6 A pin header J2.**

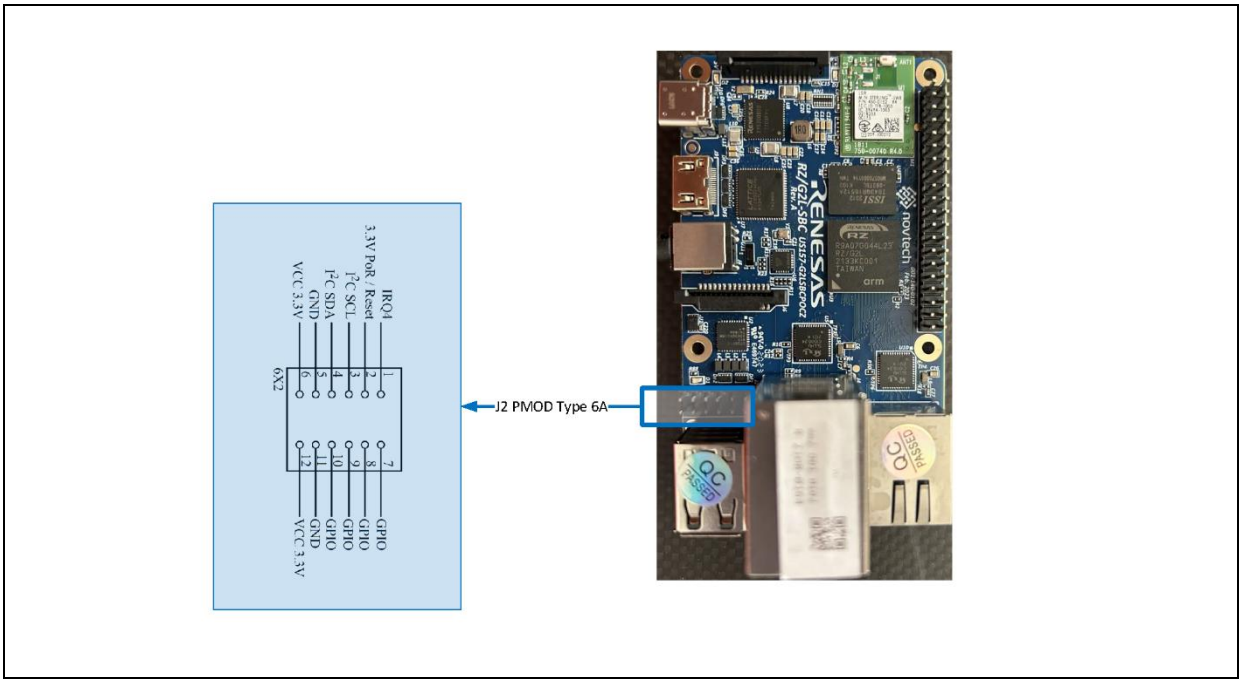



**Figure 24: PMOD Type 6A 2x6 0.1mm pin out with orientation details.**

### 4.6.9 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).

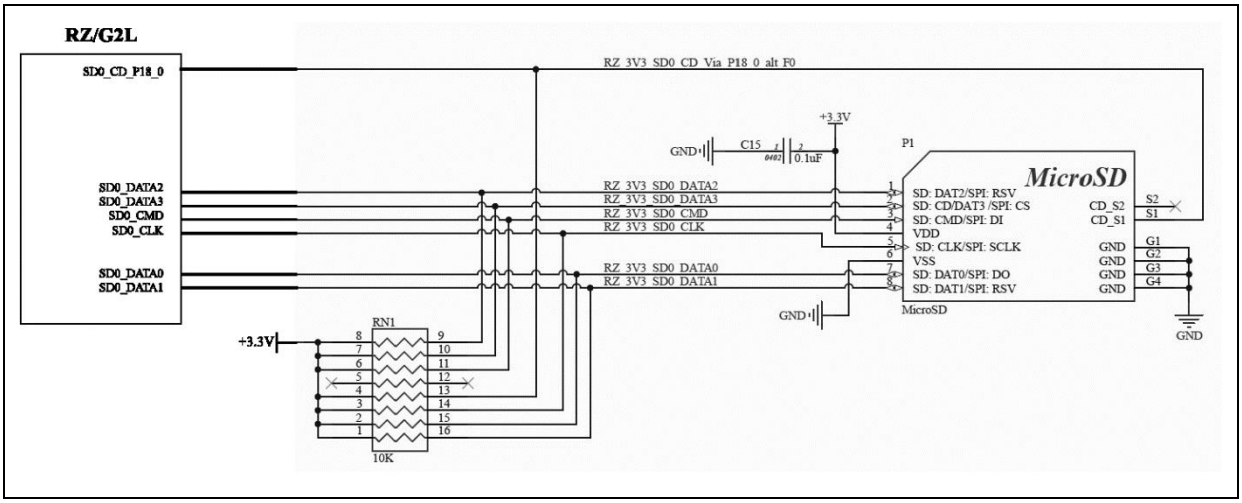



**Figure 25: uSD-Card interface block diagram.**

### 4.6.10 JTAG SWD Debug

The JTAG/SWD interface is an SMT pin out on the bottom side of the board marked as J11. It uses the standard 10-pin interfacing when populated. By default, this is not populated on the board. In addition to populating the pins of J11, the use of J12 port to set BSCANP is necessary to trigger 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 pullups.

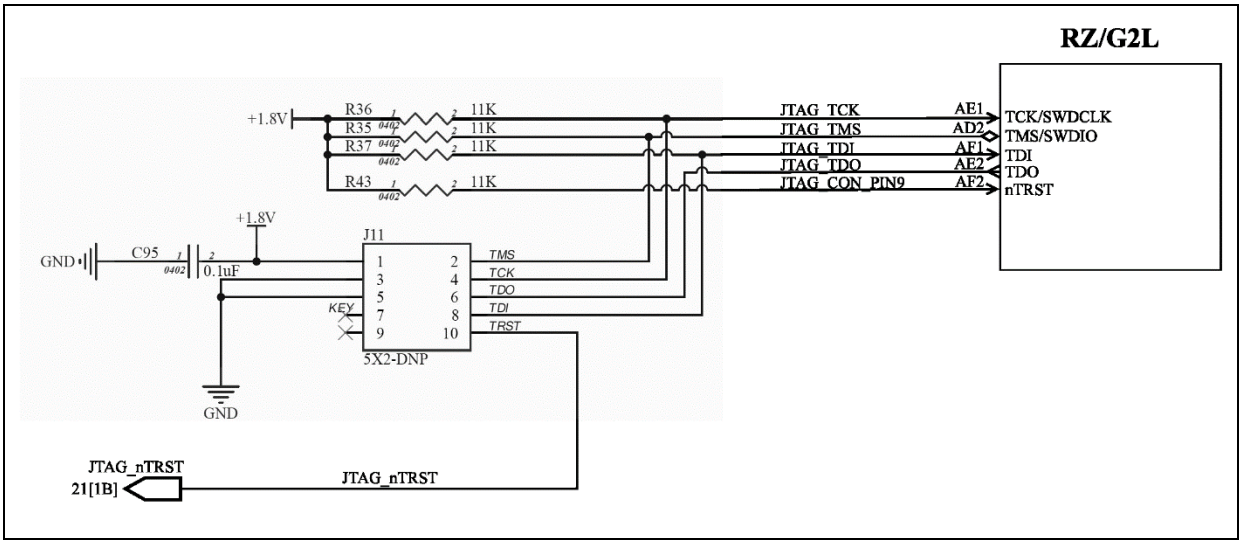



**Figure 26: JTAG/SWD Block Diagram**

### 4.6.11 Expansion Connector

The RZ/G2L-SBC has two connectors in the bottom J12 and J13 that contain pin outs for the ADC inputs, Bootstrapping (boot mode selection), and the QSPI1 interface in addition to a few GPIO's. This is meant to be used in conjunction with an adapter/daughter board. The primary uses of this are mostly on custom versions where factory flashing, and other manufacturing functions are controlled by these lines. The ADC input lines are also mapped to J13 connector.

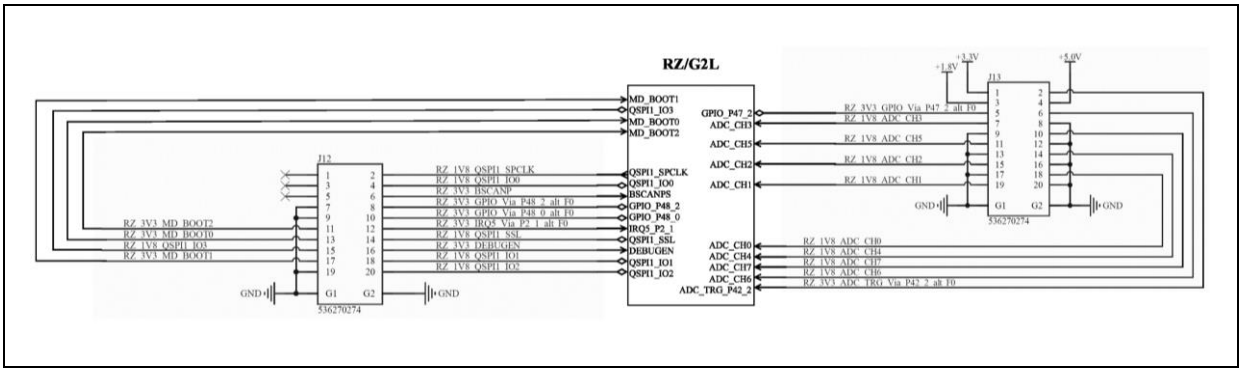



**Figure 27: Block diagram of Bottom Connectors.**

Please refer to the appendix for details on the adaptor board and flashing tools.

## 4.7 Memory

The RZ/G2L-SBC design uses 4 types of memory.

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

### 4.7.1 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.

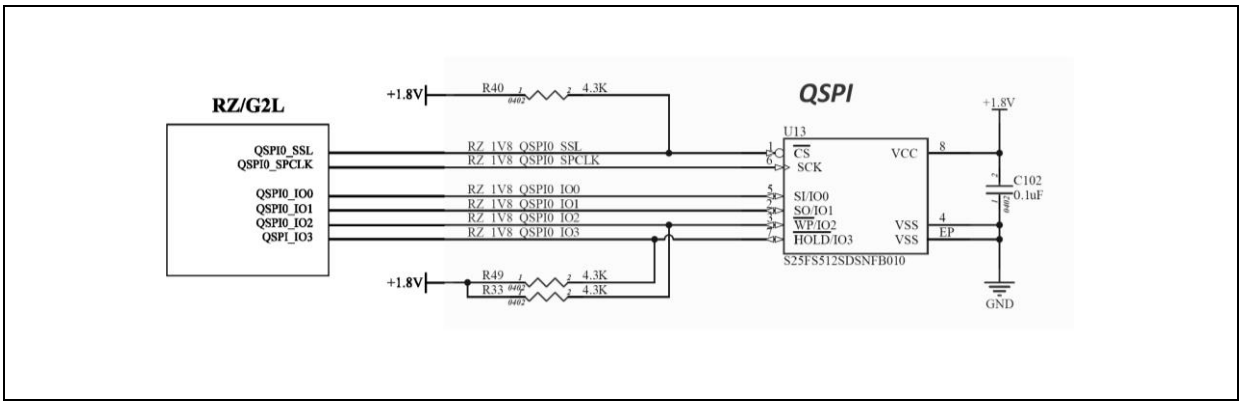



**Figure 28: QSPI interface.**

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

## 4.7.2 DDR4 SDRAM

The DDR4 SDRAM is controlled by the DDD3L/DDR4 SDRAM Memory Controller (MEMC) that is built-in to 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.

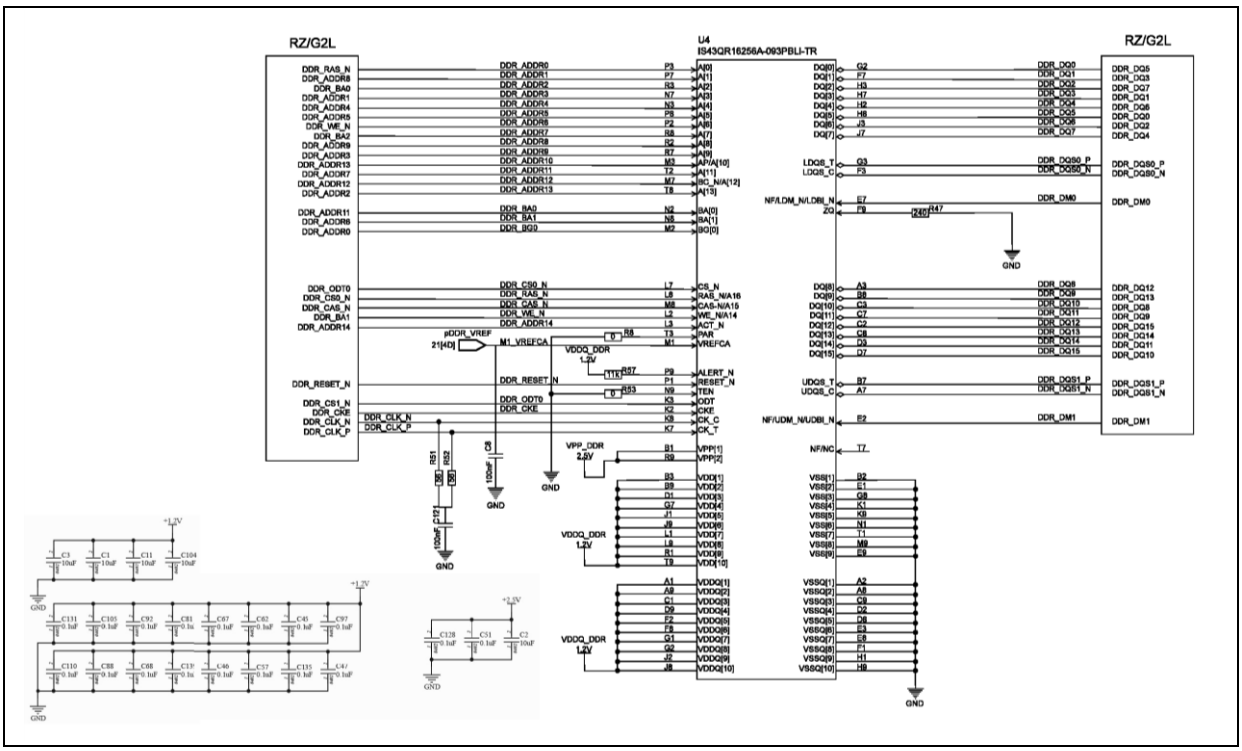



**Figure 29: DDR4 SDRAM Interface**

### 4.7.3 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. Please note that 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 kilobits) of memory. Currently, two MAC IDs occupy 6 bytes of memory each for a total of 12 bytes.

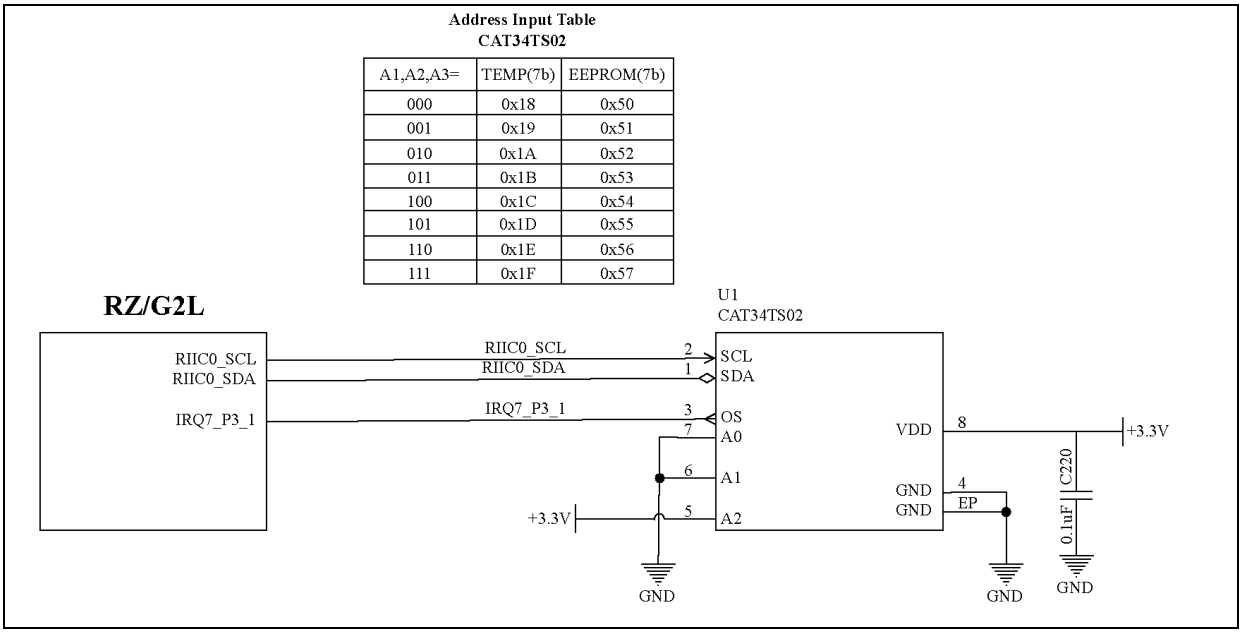



**Figure 30: I2C EEPROM Block Diagram**

| 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. |

## 4.8 GPIO Internals

The RZ/G2L SoC has a unique way of GPIO organization. It's 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. The table below maps all the available ports to bits.

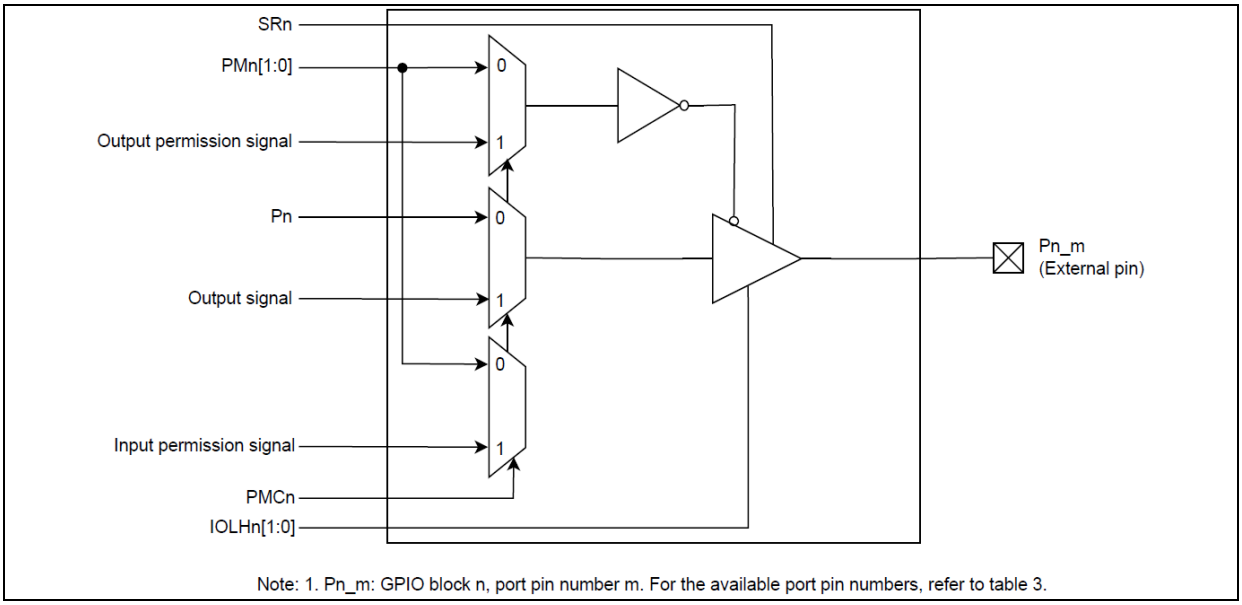



**Figure 31: Multiplexed peripheral functions configuration diagram for GPIO pins**

RZ/G2L can support up to 123 general-purpose I/O pins from 49 ports in the following table:

**Table 3: 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: Please note that the RZ/G2L has only 1 gpiochip interface to control all the supported pins mentioned in the table 3.

# 5. Quick Start

## 5.1 Hardware requirement

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

## 5.2 Essential Hardware Setup

Given below is the basic essential hardware setup. We expect at least the UART cable and an HDMI display to be available.

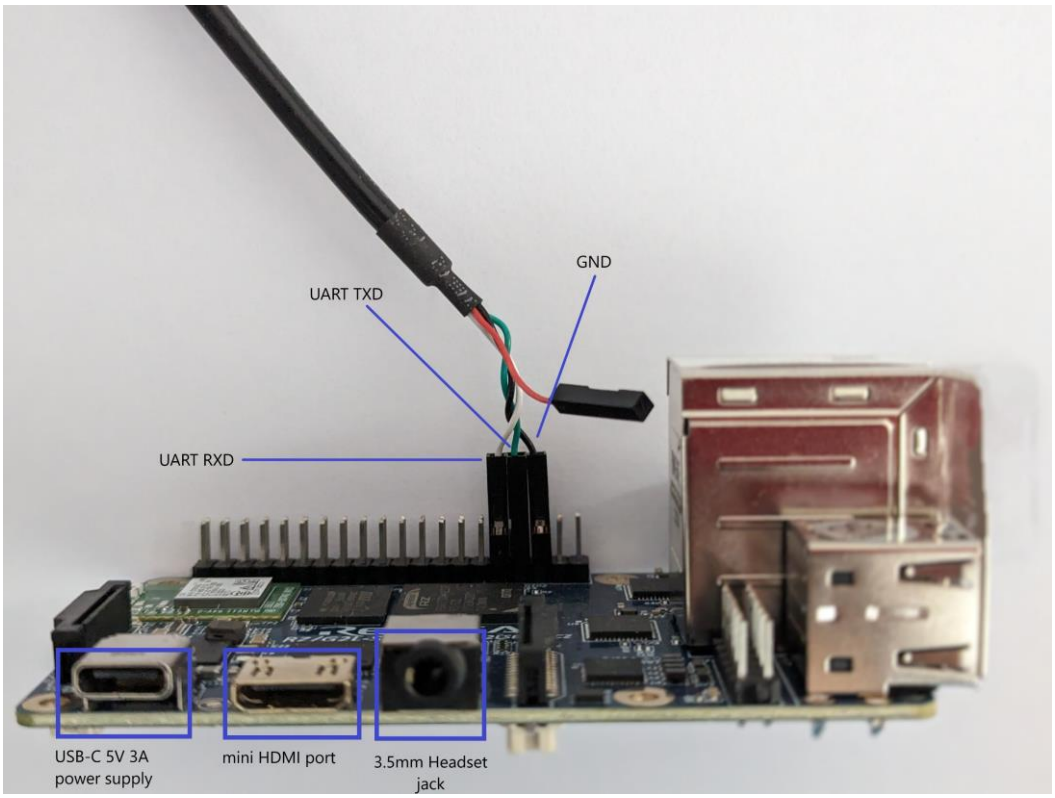



**Figure 32: Essential minimum interfaces needed**

Note: Please note that the release consists of a QT demo image. Due to this we expect at least one display to be available which is the basic 1080p HDMI monitor. However, you may also use the DSI touch panel as described in **MIPI DSI Display Touch Panel** section.

We also highly recommend that you use an FTDI cable for the UART and not any other converter chip.

## 5.3 Complete Hardware Setup

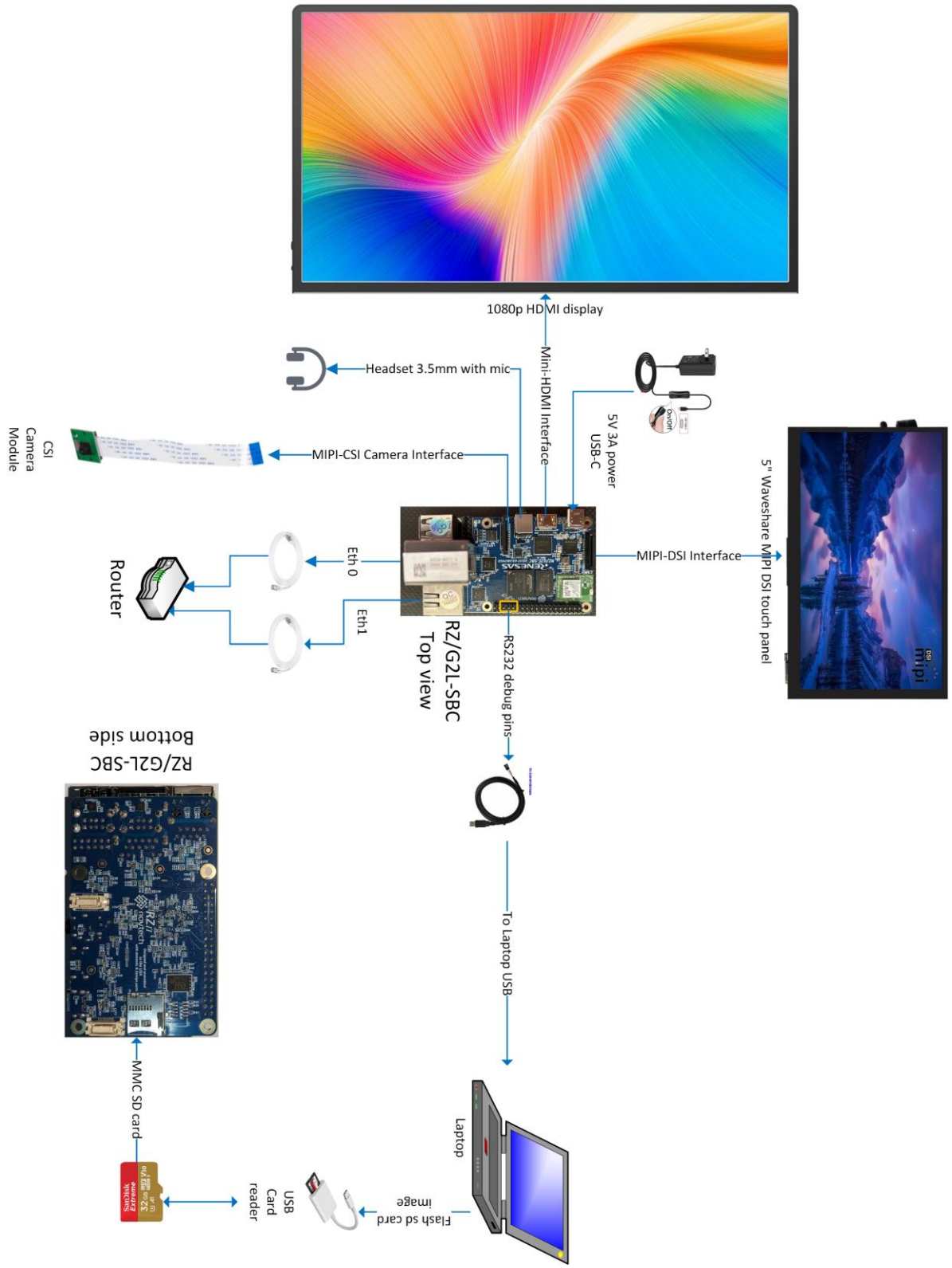



**Figure 33: Complete setup**

## 5.4 Linux SD Card Creation

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

The UI is Straight forward.

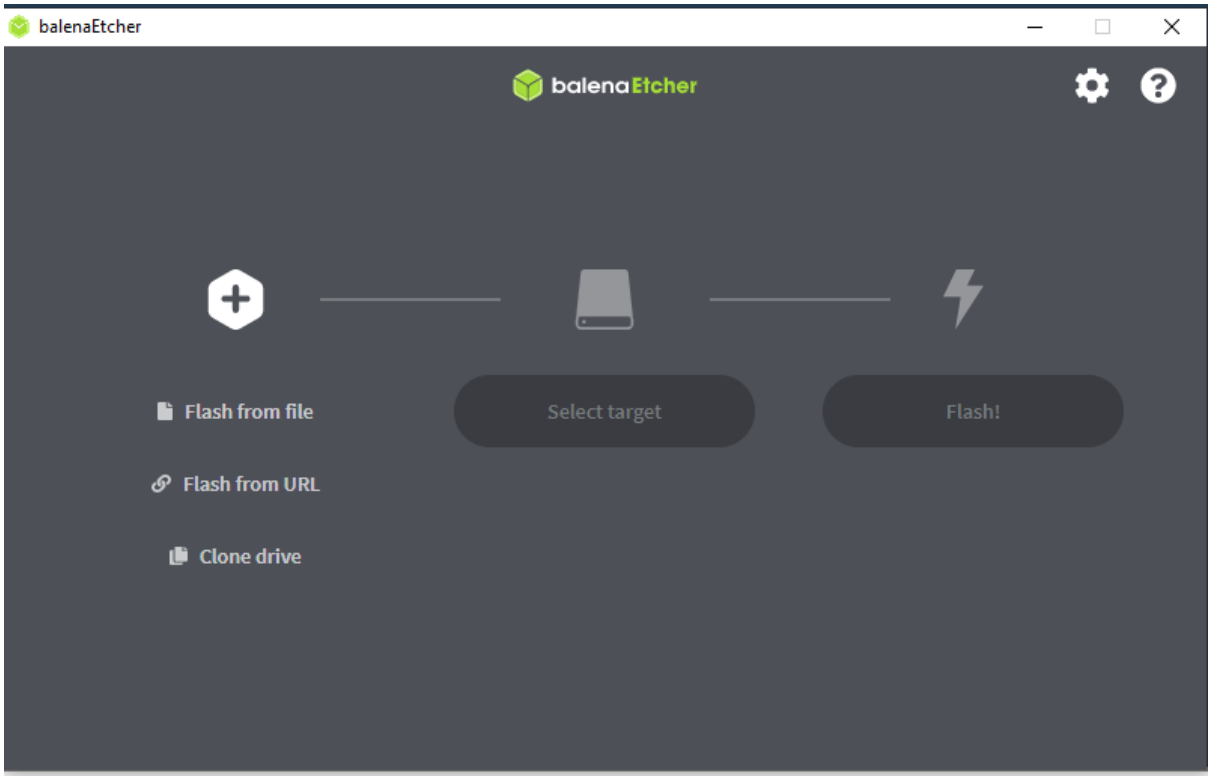



**Figure 34: Balena etcher UI**

Steps:

1. Select "Flash from File".
2. In popup window, navigate to your release and select the file '(core-image-qt-rzpi.wic'.
3. Then click on 'Select target' and it will list all available devices. Select your sd card. Be mindful not to select your primary laptop hard drive.
4. Select 'Flash'.
5. When Flashing is completed, it will automatically dismount the sd-card device.
6. Insert the sd-card into the RZ/G2L-SBC bottom sd-card connector.

## 5.5 Booting

The booting is straight forward. Insert the MMc card to the mmc port in the bottom side of the RZ/G2L-SBC. Connect keyboard, mouse, hdmi display; then insert the USB-C power supply and turn the power on. You should see the boot log on the UART console and the Weston desktop with qt apps on the HDMI screen. You can now click on any of the applications and interact with it.

The image is fully featured and has powerful desktop grade features. Explore the rest of the document to learn about all the features packed into the Linux image.

# 6. 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 process of performing a successful yocto build.

## 6.1 Build Host Environment Setup

**Requirements**

- Ubuntu 20.04 LTS (64bit) is recommended as a build environment as we are using 'Dunfell' version.
- 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 ® 3.1.27 documentation) for a quick start.

The files listed in the table below are part of the release package. These are essential files to be used for the RZ/G2L-SBC Yocto build.

| File | Description |
|---|---|
| rzsbc_yocto.sh | 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. |
| site.conf | An override file that targets for a specific build version. |
| patches | This folder contains additional patches that are needed for Yocto eSDK build. The patches are organized based on the layer to be patched. |
| 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 tool that supports parsing json file. |
| README.md | A readme file describing all the necessary info about the build process. |

**Table 4: Pre-requisite files from release package**

**Install packages on Ubuntu Host.**

1. Update the ubuntu package manager.
```
$ sudo apt update
```
2. Install necessary packages and tools which are used by the yocto build.
```
$ sudo apt install -y gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping libsdl1.2-dev xterm p7zip-full libyaml-dev \
rsync curl locales bash-completion
```
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.

| File name | Version | Download Link | Comments |
|---|---|---|---|
| RTK0EF0045Z13001ZJ-v1.1.2_EN.zip | 1.1.2 | rz-mpu-graphics-library-evaluation-version | This is the Mali driver and graphics package that enables the Mali GPU in the SoC. |
| RTK0EF0045Z15001ZJ-v1.1.0_EN.zip | 1.1.0 | rz-mpu-video-codec-library-evaluation-version | Video codec package |

**Table 5: List of packages to manually download for Yocto Build**

5. We assume that all the downloaded zip files from Table 5 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 e.g., '~/yocto' as shown below) in Ubuntu Host PC.

```
$ cd ~/Downloads/renesas-yocto
$ mkdir ~/yocto
$ mv *.zip ~/yocto
```

7. This example assumes the pre-requisite files that are described in Table 4 are located at ' ~/Downloads/renesas-yocto/ rz-sbc-qt-v1.1.1' path after unpacking. Alternatively, one can initiate a build directly in `host/src` by copying the zip files listed in Table 5 to the 'host/src' location.
   Copy all the files and directories from the release package's 'host/src' directory into '~/yocto' folder.

```
$ cd ~/Downloads/renesas-yocto/rz-sbc-qt-v1.1.1/
$ cp -r ./host/src/* ~/yocto
```

8. Eventually, all the necessary files for the yocto build should be present in '~/yocto' folder as shown below.

```
renesas@builder-pc:~/Downloads/renesas-yocto$ cd ~/yocto/
renesas@builder-pc:~/yocto$ tree
.
├── git_patch.json
├── jq-linux-amd64
├── patches
│   ├── meta-summit-radio
│   │   └── 0001-rzsbc-summit-radio-pre-3.4-support-eSDK-build.patch
│   └── poky
│       └── 0001-meta-classes-esdk-explicitly-address-the-location-of.patch
├── README.md
├── RTK0EF0045Z13001ZJ-v1.1.2_EN.zip
├── rzsbc_yocto.sh
└── site.conf

4 directories, 8 files
```

## 6.2 Initiate Yocto Build

Add execute permission to rzsbc_yocto.sh and to jq-linux-amd64.

```
renesas@builder-pc:~/yocto$ chmod a+x rzsbc_yocto.sh
renesas@builder-pc:~/yocto$ chmod a+x jq-linux-amd64
```

Commence build:

```
renesas@builder-pc:~/yocto$ ./rzsbc_yocto.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.

## 6.3 Collect the build output

After building Yocto, the output folder should be located at:
`~/yocto/yocto_rzsbc_board/build/tmp/deploy/images/rzpi`
The output folder outline should look as follows:

```
renesas@builder-
pc:~/yocto/yocto_rzsbc_board/build/tmp/deploy/images/rzpi$ tree
.
├── host
│   ├── build
│   │   ├── core-image-qt-rzpi-20240918080332.rootfs.manifest
│   │   ├── core-image-qt-rzpi-20240918080332.testdata.json
│   │   ├── core-image-qt-rzpi.manifest -> core-image-qt-rzpi-
20240918080332.rootfs.manifest
│   │   └── core-image-qt-rzpi.testdata.json -> core-image-qt-rzpi-
20240918080332.testdata.json
│   ├── env
│   │   ├── core-image-qt.env
│   │   └── Readme.md
│   ├── Readme.md
│   ├── src
│   │   ├── git_patch.json
│   │   ├── jq-linux-amd64
│   │   ├── patches
│   │   │   ├── meta-summit-radio
│   │   │   │   └── 0001-rzsbc-summit-radio-pre-3.4-support-eSDK-build.patch
│   │   │   └── poky
│   │   │       └── 0001-meta-classes-esdk-explicitly-address-the-location-
of.patch
│   │   ├── README.md
│   │   ├── rzsbc_yocto.sh
│   │   └── site.conf
│   └── 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
│   └── Disclaimer052.pdf
├── r12uz0158eu0101-rz-g2l-sbc-single-board-computer.pdf
├── README.md
```

```
├── RZG2L-SBC_Evaluation_license.pdf
└── target
    ├── env
    │   ├── Readme.md
    │   └── uEnv.txt
    ├── images
    │   ├── bl2_bp-rzpi.bin
    │   ├── bl2_bp-rzpi.srec
    │   ├── bl2-rzpi.bin
    │   ├── core-image-qt-rzpi.wic
    │   ├── dtbs
    │   │   ├── overlays
    │   │   │   ├── Readme.md
    │   │   │   ├── rzpi-can.dtbo
    │   │   │   ├── rzpi-dsi.dtbo
    │   │   │   ├── rzpi-ext-i2c.dtbo
    │   │   │   ├── rzpi-ext-spi.dtbo
    │   │   │   └── rzpi-ov5640.dtbo
    │   │   ├── Readme.md
    │   │   ├── rzpi--5.10.184-cip36+gitAUTOINC+5f065ec41b-r1-rzpi-
20240918080332.dtb
    │   │   └── rzpi.dtb -> rzpi--5.10.184-cip36+gitAUTOINC+5f065ec41b-r1-
rzpi-20240918080332.dtb
    │   ├── fip-rzpi.bin
    │   ├── fip-rzpi.srec
    │   ├── Flash_Writer_SCIF_rzpi.mot
    │   ├── Image -> Image--5.10.184-cip36+gitAUTOINC+5f065ec41b-r1-rzpi-
20240918080332.bin
    │   ├── Image--5.10.184-cip36+gitAUTOINC+5f065ec41b-r1-rzpi-
20240918080332.bin
    │   ├── Readme.md
    │   └── rootfs
    │       ├── core-image-qt-rzpi.tar.bz2
    │       └── Readme.md
    └── Readme.md

28 directories, 92 files
```

# 7. Creating bootable SD card

This section describes all the tools and methods for creating the Linux bootable SD card under different environments.

## 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 the rootfs into. In this case, the device needs to flash its SD card. You will have to identify the correct device name which represents the SD card on Linux.

The below example shows how to identify an SD card on Ubuntu 22.0.4. The command 'lsblk' is executed to check all available storage devices. You can see that the 32GB SD card is being 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
```

Please identify the device name of the sd-card to be flashed. Then, pass it to the script as argument as shown in the example here:

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

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

> Note: Due to the various Linux distributions having different disk management arrangements, the script may fail to create the card. Hence, we are unable to assure that the script will work in every Linux environment. In the case where it fails, you might need to modify the call for creating the filesystem like the calls to ext4fs in the script. Please pay attention to the script and ensure that it succeeds.
>
> The script is tested on ubuntu 22.0.4.

## 7.2 Windows Host

The preferred way to flash the image onto the SD card is to simply use Balaena etcher to flash the: core-image-qt-rzpi.wic image file onto the SD-card. You can use any etcher that can create bootable media.

# 8. Programming / Flashing Firmware to 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 being performed or a custom version designed by the end user. The 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, the SBC does not have an emmc storage and the Linux image is expected to be available on an SD card or on a TFTP server.

## 8.1 Hardware Setup

To perform a firmware flashing, you need to ensure the following:

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

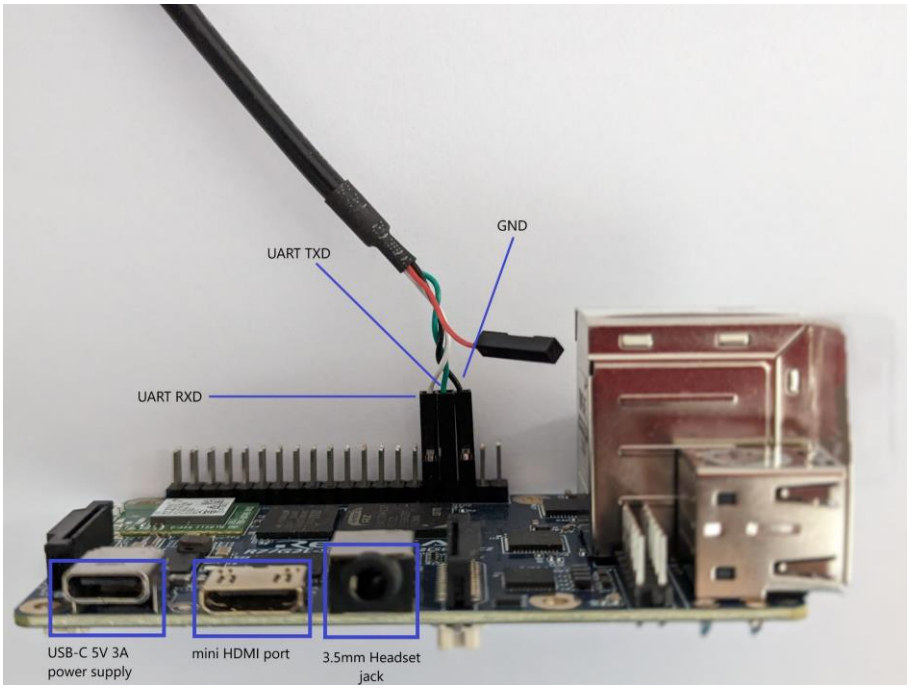



**Figure 35: Cortex A55 debug UART cable interface**

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.

## 8.2 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.

Default bootloader images (.bin) are in the subdirectory `/boot/uload-bootloader` of the root filesystem in 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 don't need a separate image for this).
- ✓ Ensure that Teraterm application is installed on your windows pc.
- ✓ Ensure that minicom and FTDI drivers are loaded properly on Linux host pc.
- ✓ Ensure that the scripts in the process have execute permissions.

### 8.2.1 Linux Host

The Linux flashing script is named: uload_bootloader_flash.py under 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
```

Please note that the script by default tries to access /dev/ttyUSB0 without any arguments passed. This works on most systems which have a single FTDI cable attached to a single USB port.

Here are the simplest steps to flash:

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

Step 2. Start the script uload_bootloader_flash.py.

```
renesas@builder-pc:~/yocto/yocto_rzsbc_board/build/tmp/deploy/images/rzpi$ cd
host/tools/uload-bootloader/linux
renesas@builder-pc:~/yocto/yocto_rzsbc_board/build/tmp/deploy/im-
ages/rzpi/host/tools/uload-bootloader/linux$ ./uload_bootloader_flash.py
```

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

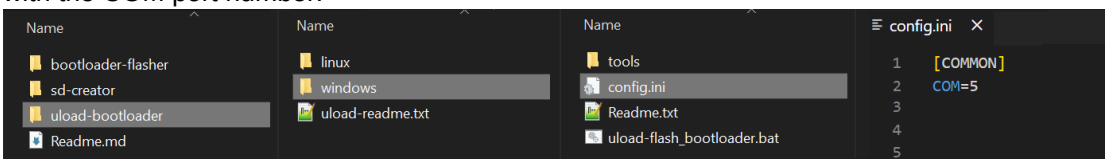
Step 4. Once the flashing is complete, power-cycle the board.

### 8.2.2 Windows Host

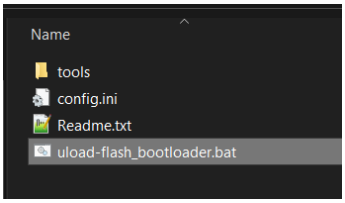
Windows host uses its own script that is cleanly tucked into the sub directory of uload-bootloader called windows. The sub-directory has its own Readme.md describing everything that's needed.

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

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




Execute the uload-flash_bootloader.bat




Step 2. 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

In this section, we will explore the features and interfaces available on the RZ/G2L-SBC.

## 9.1 QT Demo Applications

The Linux root file system contains a few QT applications for demo purposes. They can be launched from the taskbar at the top of the screen. They can also be launched through the UART console. When you login to the console, you will find all the demo apps in the home directory of root user.

```
root@rzpi:~# cd /home/root/demo/scripts/
root@rzpi:~/demo/scripts# ls
Help.sh  Qmlvideofx-demo.sh  QtCinematicExperience-demo.sh  Qteverwhere-demo.sh
Qt-launch-demo.sh  QtSmarthome-demo.sh
```

Most of the demo apps are launched through their corresponding shell script or the UI launchers on the taskbar.

For example, QT smart home demo application can be executed as follows:

```
root@rzpi:~# cd /home/root/demo/scripts/
root@rzpi:~/demo/scripts# ./QtSmarthome-demo.sh
```

The following figure shows all the demo apps on the taskbar:

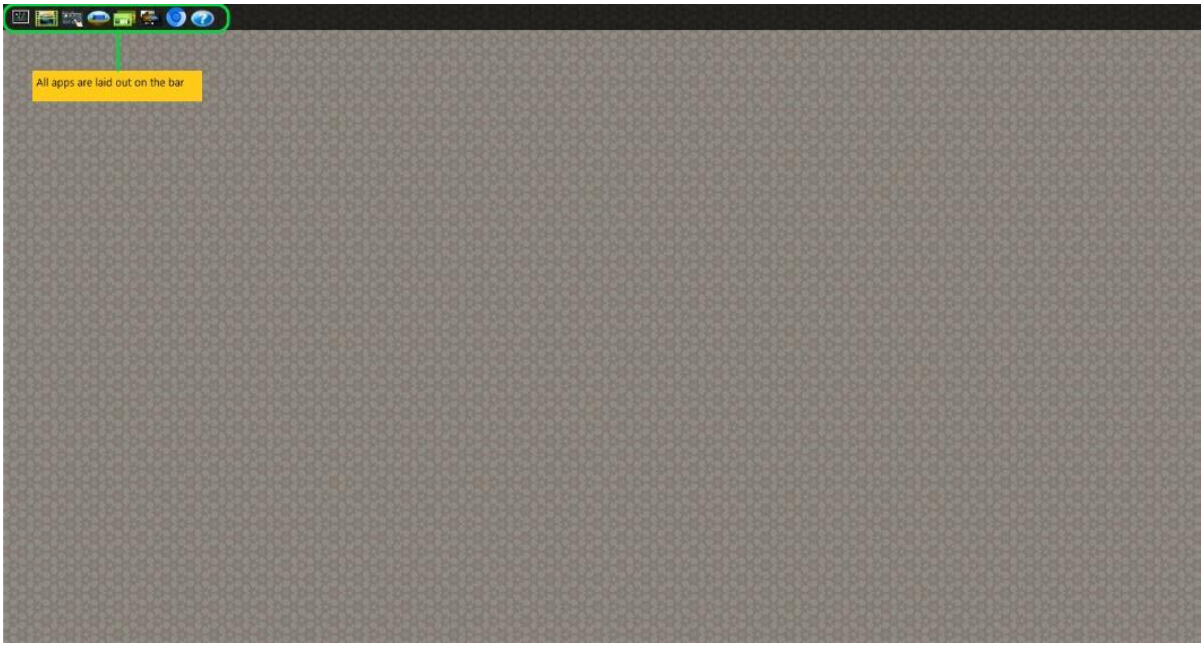
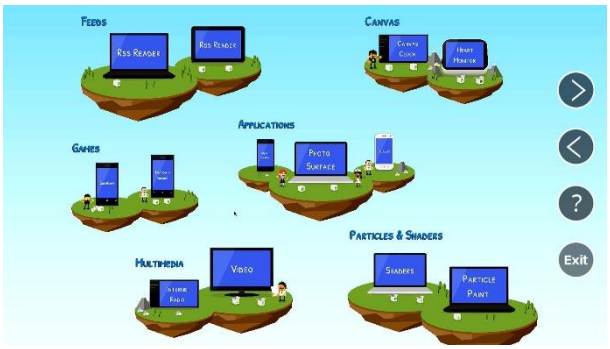



**Figure 36: All the demo apps are on the taskbar on the main screen.**

Each demo app offers a unique blend of functionality and user interface, catering to diverse needs and preferences. However, due to the main memory limitation, not all of them can run successfully on the RZ/G2L-SBC.

The following table describes the details for each demo app.

| Qt demo application name | Screenshot | Description |
|---|---|---|
| Qt Smart Home (QtSmarthome-demo.sh) |  | - This application shows how you can control and adjust various home operations. Some activities are the control of windows, blinds, heating, and lighting.<br>- The operations are activated by a change in weather conditions, and you can also adjust the weather as you like in the "weather god control" mode. |
| Qt Graphical Effects (Qmlvideofx-demo.sh) | This demo application does not run properly when rendering videos due to its heavy size. There is no screenshot for it. | |
| Qt everywhere (Qteverwhere-demo.sh) |  | - This application contains several Qt Quick 2 applications which you can launch by tapping the devices.<br>- The applications are separated into several areas such as Games, Multimedia, Feeds, Canvas, Applications and Particles & Shaders. |
| Qt Quick (Qt-launch-demo.sh) |  | - This demo application shows new features in QT Quick 2.0.<br>- There are “Qt Quick - Front” where font rendering performs, “Qt Quick - Canvas” where shapes are created visually, “Qt Quick – Particle System” where special effects follow your cursor. |

| | | |
|---|---|---|
| Qt Cinematic Experience (QtCinematic Experience-demo.sh) |  | - This UX demo application presents some graphical features of Qt5.<br>- The name 'Cinematic Experience' reflects how it's possible to build user interfaces with increased dynamics. |

## 9.2 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).

Note:

The GPIO pin array is multiplexed with peripheral IO lines. However, by default, they are mostly GPIO's.
By default, I2C channel 0 and SCIF channel 0 are enabled.
The rest of the pins are GPIO's by default. Other functions are enabled 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 display. Please ensure that you reboot the board for the overlay to take effect.

### 9.2.1 U-Boot Environment

The u-boot environment file is named 'uEnv.txt' and is present in the 'boot' directory. It contains boot configuration settings to be processed by the u-boot and configuration to be passed on to the Linux kernel. The full description of the U-boot environment is beyond the scope of this document. However, we cover the necessary aspects and settings that are relevant to the SBC and most frequently used.

The table below provides a list of all the overlay options available in the provided kernel.

| Config | Value if set | Loading | Description |
|---|---|---|---|
| **fdtfile** | rzpi.dtb | rzpi.dtb | Main device tree file to be loaded from the filesystem |
| **enable_overlay_i2c** | 1 or 'yes' | rzpi-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' | rzpi-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' | rzpi-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' | rzpi-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' | rzpi-ov5640.dtbo | Enables the OV5640 CSI camera driver enumeration and loads the v4l2 pipelines. |

There is a `readme.txt` file in `/boot` folder with the descriptions of the FDT overlay information. This is usually more up-to-date with the build.

Note:

The Linux shell command ‘sync’ needs to be run after changing files on the rootfs to ensure that the data is flushed to the actual physical storage. Without it there is a possibility that the changes may not take effect in the actual file.

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

### 9.2.2 GPIO (General Purpose I/O pins)

By default, most pins are configured as GPIO’s on the SBC’s 40-pin GPIO pin header. This section describes what those pins are and how to access them. The io pins are explored in detail in Figure 22: 40 PIN GPIO map with orientation details. The explanation of the Linux GPIO framework is beyond the scope of this document. In this section, we mostly deal with the identification of pin and port numbers and how to access them.

Linux sysfs uses /sys/class/gpio entries to control the GPIO bank. The following table maps out the pins and their functions to the IO port header:

| GPIO Pin number | Function | group | pin | J3 PINs | | pin | group | Function | GPIO Pin Number |
|---|---|---|---|---|---|---|---|---|---|
| | | | | **Left side** | **Right side** | | | | |
| | 3.3V | | | **1** | **2** | | | 5V | |
| **490** | I²C3 SDA | 46 | 2 | **3** | **4** | | | 5V | |
| **491** | I²C3 SCL | 46 | 3 | **5** | **6** | | | GND | |
| **304** | GPIO | 23 | 0 | **7** | **8** | 0 | 38 | SCIF0 TX | **424** |
| | GND | | | **9** | **10** | 1 | 38 | SCIF0 RX | **425** |
| **456** | GPIO | 42 | 0 | **11** | **12** | 2 | 7 | GPIO | **178** |
| **336** | GPIO | 27 | 0 | **13** | **14** | | | GND | |
| **345** | GPIO | 28 | 1 | **15** | **16** | 0 | 8 | GPIO | **184** |
| | 3.3V | | | **17** | **18** | 0 | 15 | GPIO | **240** |
| **465** | SPI0 MOSI | 43 | 1 | **19** | **20** | | | GND | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **466** | SPI0 MISO | 43 | 2 | **21** | **22** | 1 | 14 | GPIO | **233** |
| **464** | SPI0 CK | 43 | 0 | **23** | **24** | 3 | 43 | SPI0 CS | **467** |
| | GND | | | **25** | **26** | 1 | 11 | GPIO | **209** |
| | $I^2C0$ SDA | | | **27** | **28** | | | $I^2C0$ SCL | |
| **152** | GPIO | 4 | 0 | **29** | **30** | | | GND | |
| **153** | GPIO | 4 | 1 | **31** | **32** | 0 | 32 | GPIO | **376** |
| **297** | GPIO | 22 | 1 | **33** | **34** | | | GND | |
| **457** | CAN0 TX | 42 | 1 | **35** | **36** | 1 | 23 | GPIO | **305** |
| **208** | CAN0 RX | 11 | 0 | **37** | **38** | 0 | 46 | CAN1 TX | **488** |
| | GND | | | **39** | **40** | 1 | 46 | CAN1 RX | **489** |

The SoC uses bank ID and io line number to identify the GPIO port. The pin mux uses a unique Px_y notation for the physical pins. Linux, however, uses a linear GPIO pin number list and internally maps the GPIO numbers to the appropriate GPIO line.

The following method is used to identify the correct Linux pin number:

Step 1: We start with the Px_y io pin from the schematic. Identify the port values as per the table below.

**Table 6: Symbol definition for GPIO Px_y Notation**

| Symbol / variable in notation | Description |
|---|---|
| **x** | Group number / port number |
| **y** | Pin number in port (0:8) |
| **G** | Group in (always 8 bit which is the size of the port register). Constant 8. |
| $\mathbf{p_{base}}$ | Pin base: starting io pin number (constant 120). All external Linux gpio pins start from 120. |

Step 2: Calculate the Linux port ID using the following formula:

$$\mathbf{Linux\_pin\_number = (x * G) + y + P_{base}}$$

Example for J3 PIN 7:

$$(23*8) + 0 + 120 = 304 = pinum$$

To set the GPIO pin, change the directory to the GPIO sysfs directory and set values as shown below:

```
root@rzpi:~# cd /sys/class/gpio/
root@rzpi:/sys/class/gpio# echo 304 > export
```

There will be a new directory that represents the GPIO pin. In this example, it will be the P23_0 directory.

```
root@rzpi:/sys/class/gpio# ls
P23_0  export  gpiochip120  unexport
```

Inside the P23_0 directory, some control interfaces are created by the Linux sysfs to manage the GPIO pin:

```
root@rzpi:/sys/class/gpio# cd P23_0
root@rzpi:/sys/class/gpio/P23_0# ls
active_low  device  direction  edge  power  subsystem  uevent  value
```

Note:

The Linux sysfs is not populated with all the gpio's. They are usually mapped for use within the kernel. So, to get the gpio handle, its necessary to call an export on it so that the kernel driver makes it available by populating a new directory with the pin number and, control handles placed in it. For detail, refer to the official document: https://docs.kernel.org/5.10/admin-guide/gpio/sysfs.html

#### 9.2.2.1 Setting I/O pin direction

To control the input/output of the GPIO pin, either "in" or "out" should be written to the "direction" interface. Writing as "out" defaults to initializing the value as low.

```
root@rzpi:/sys/class/gpio# echo out > P23_0/direction
```

#### 9.2.2.2 Reading the GPIO

To read the state high/low of the GPIO pin, print out the value of the "value" interface.

```
root@rzpi:/sys/class/gpio# cat P23_0/value
0
```

Value 0 means the I/O pin is low; Value 1 means the I/O pin is high.

#### 9.2.2.3 Setting the GPIO

The ability to control the pin's output is only available when the direction is set to 'out' / output mode. To set the high/low value of the GPIO pin (output pin), either "1" or "0" should be written to the "value" interface. Any nonzero value is treated as high.

```
root@rzpi:/sys/class/gpio# echo 1 > P23_0/value
root@rzpi:/sys/class/gpio# cat P23_0/value
1
root@rzpi:/sys/class/gpio# echo 0 > P23_0/value
root@rzpi:/sys/class/gpio# cat P23_0/value
0
```

You can always read the current state of the port by reading back the value interface.

### 9.2.3 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 the I2C channel 3 is enabled, run the following command, and check the result:

```
root@rzpi:~# 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@rzpi:~#
```

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

```
root@rzpi:~# 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.

### 9.2.4 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 config the SPI:

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

### 9.2.5 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@rzpi:~# 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@rzpi:~#
```

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

The below example shows the communication between two CAN channels.

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

## 9.3 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@rzpi:~# 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@rzpi:~# 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 the communication through the Wi-Fi. If issues start appearing, use the following to disable the ethernet ports.

```
root@rzpi:~# ifconfig eth0 down

root@rzpi:~# ifconfig eth1 down
```

## 9.4 On-board Audio Codec with Stereo Jack

The RZ/G2L-SBC comes equipped with an onboard audio codec: Renesas DA7219. The audio codec is connected to the DAI interface (SSI 1) of the SoC configured to I2S data format for the audio data while the control interface is on the I2C 0 interface.

The SBC board has a 3.5mm headset Jack labeled J8. It uses a 6-pin connector.

You can play and record audio directly using ALSA tools. However, it's restricted to PCM wave files only. The image comes equipped with a fully configured GStreamer that lets you play other types of audio files like MP3.

The following shows the two commands to play audio files.

```
root@rzpi:~# aplay /home/root/audios/04_16KH_2ch_bgm_maoudamashii_healing01.wav
root@rzpi:~# 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 an audio.

```
root@rzpi:~# 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 by the following command:

```
root@rzpi:~# aplay audio_capture.wav
```

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

```
root@rzpi:~# alsamixer
```

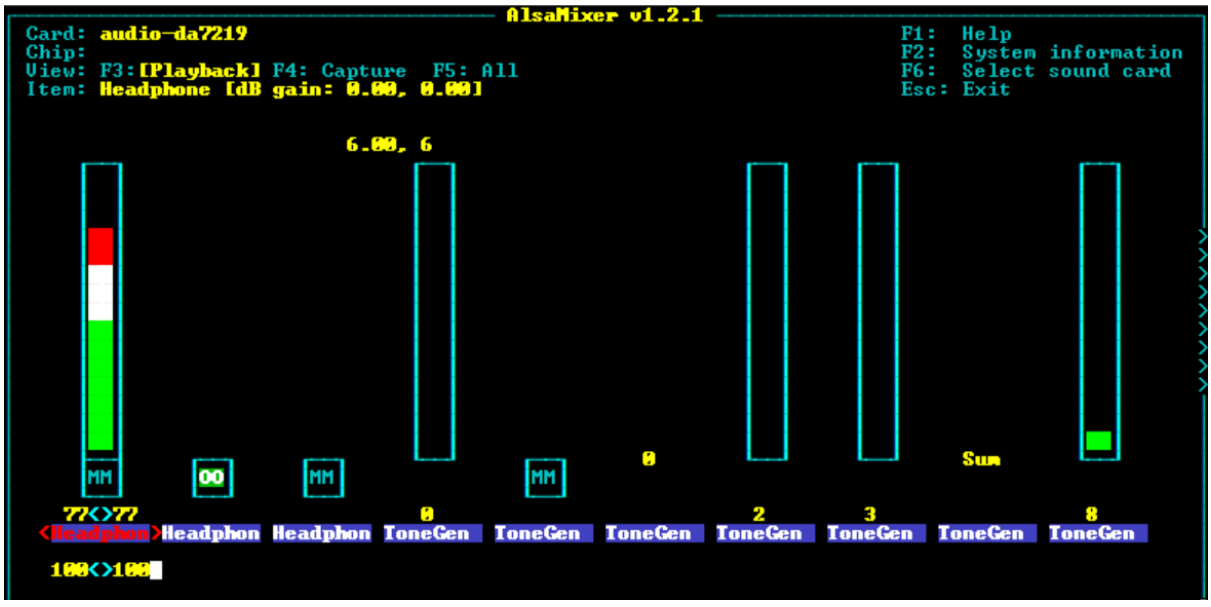



**Figure 37: ALSA Mixer GUI on RZ/G2L-SBC**

## 9.5 MIPI DSI Display Touch Panel

RZ/G2L-SBC has a 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.

### 9.5.1 Hardware Interfacing

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

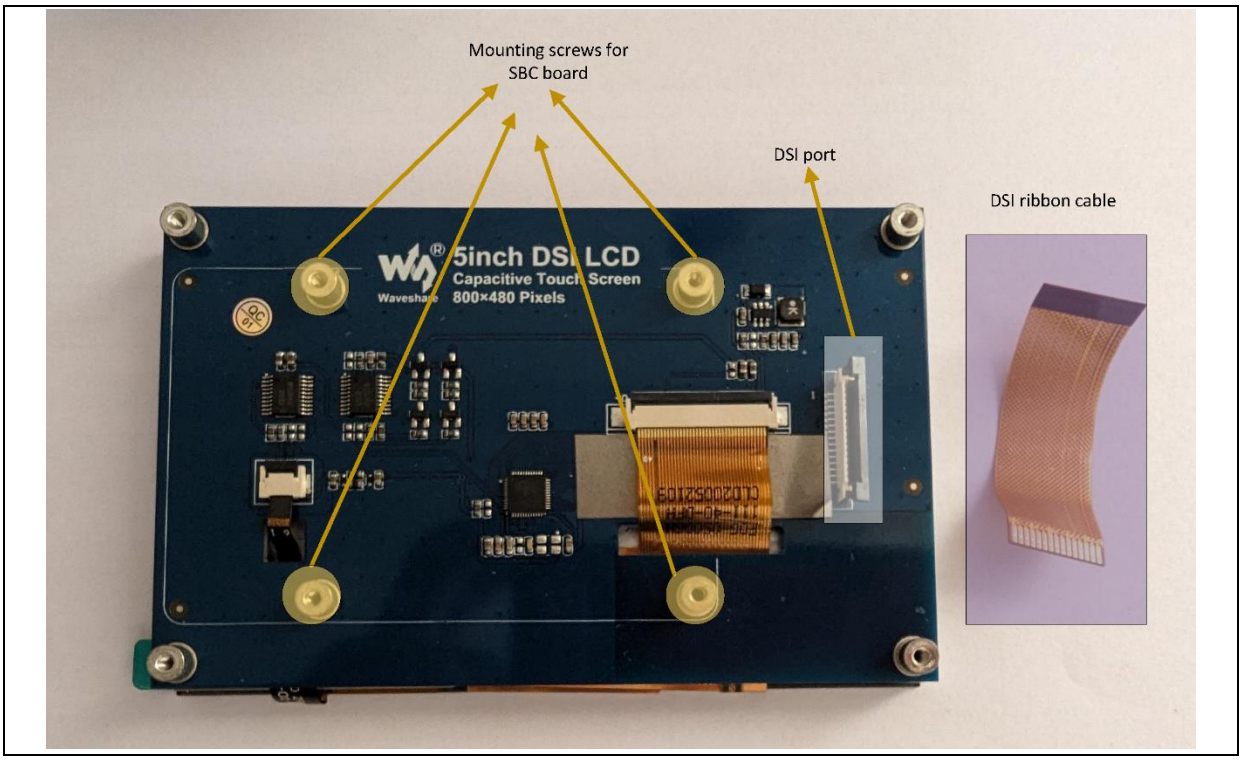



**Figure 38: Waveshare 5" DSI touch panel read side picture with flat ribbon cable.**

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.

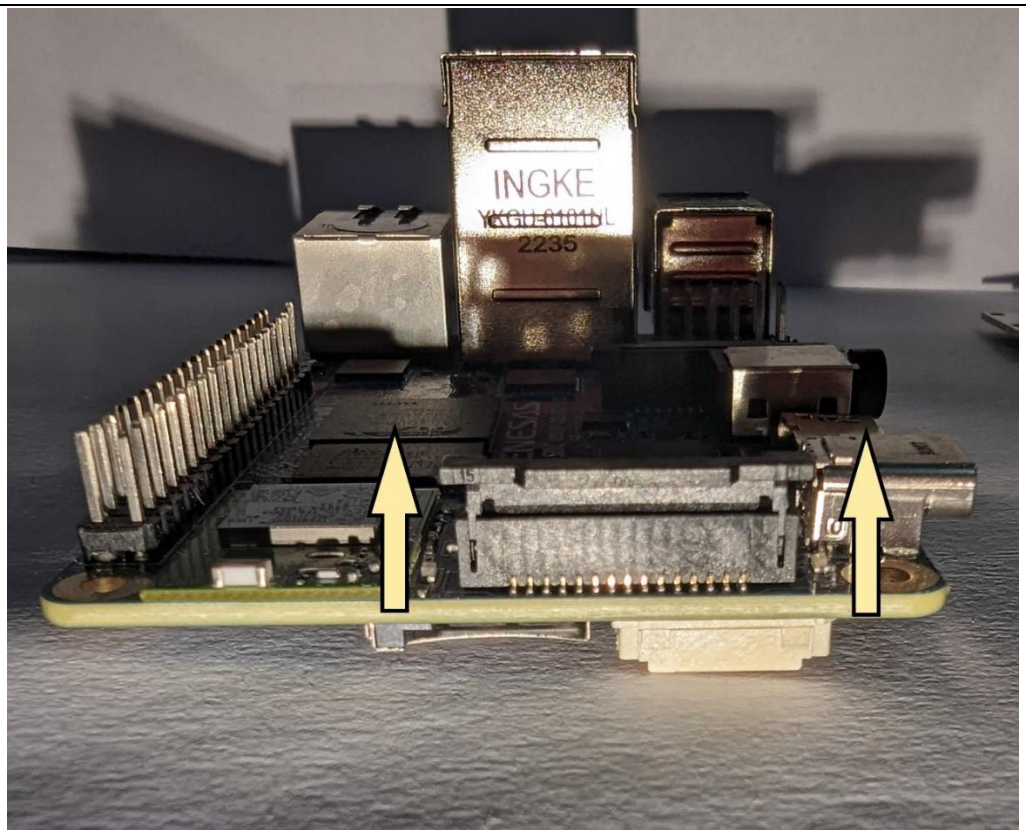



**Figure 39: DSI port notch lock open by pulling it up.**

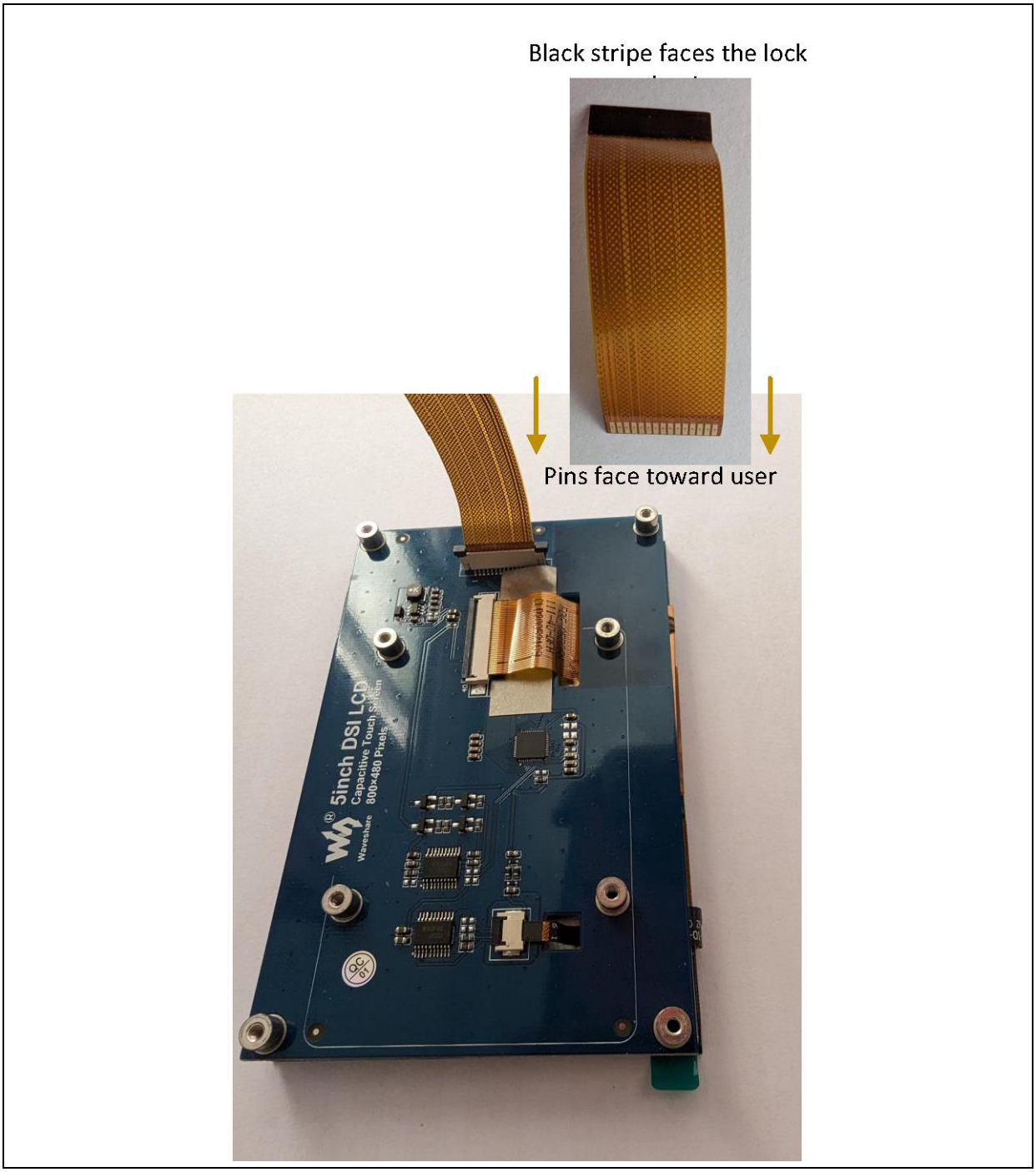



**Figure 40: Waveshare DSI touch display DSI port interfacing cable orientation.**

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

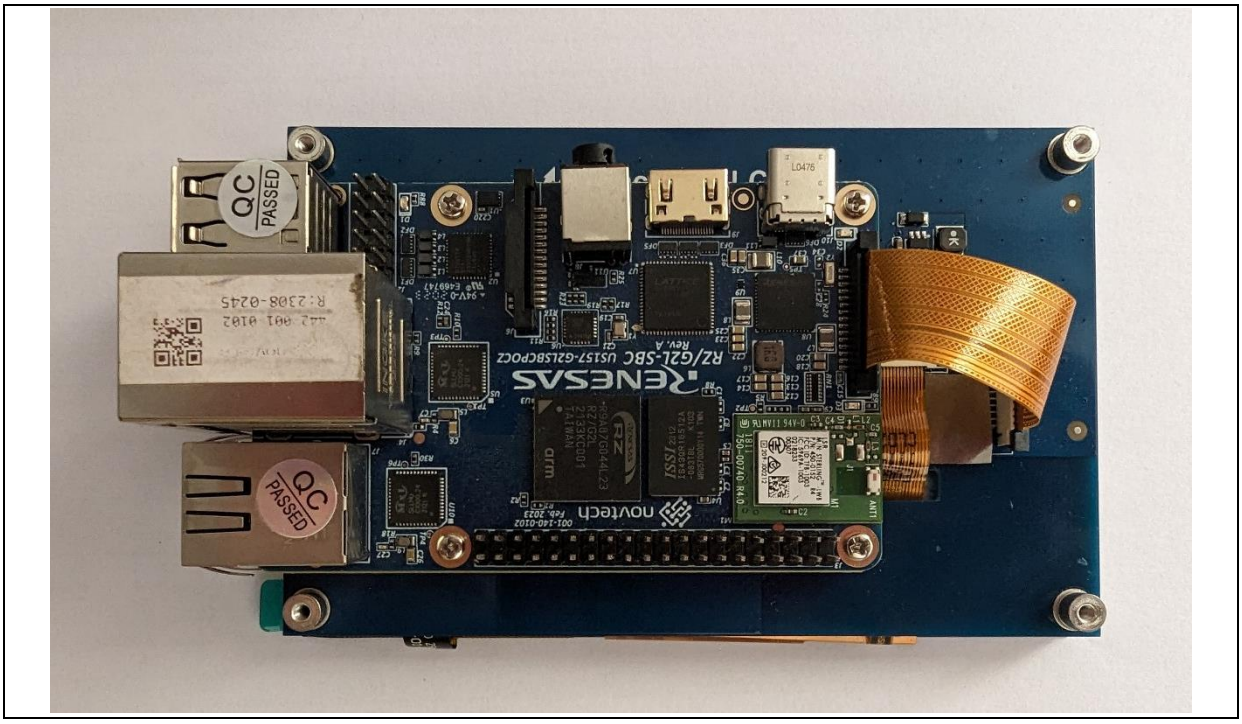



**Figure 41: RZ/G2L-SBC mounted to the Waveshare DSI panel and interfaced.**

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

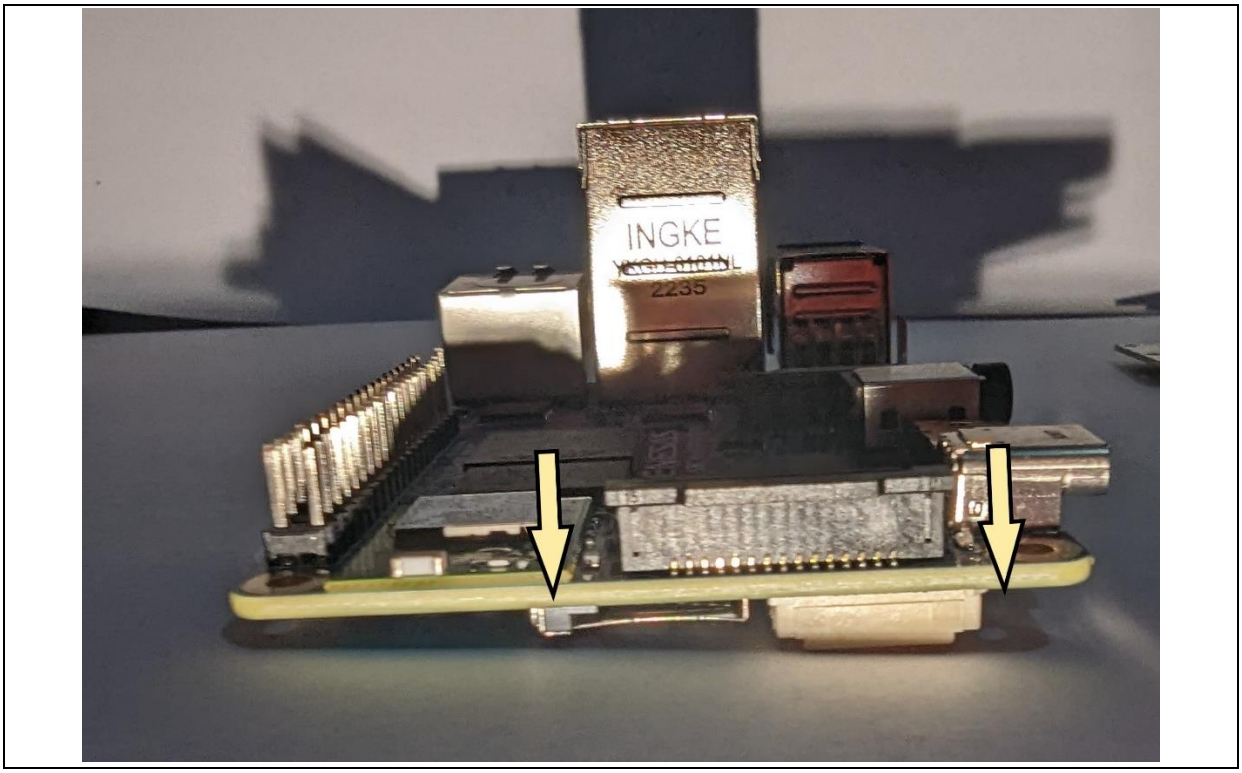



**Figure 42: DSI port notch in the lock position. The cable is not shown to keep the notch in clear view.**

**Figure 43: 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 the two USB ports. We still recommend that you use a support stand, even an off-market custom one, to ensure that the DSI cable is off the ground.

Remember that the DSI port includes an $I^2C$ 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 wrong electricals.

### 9.5.2 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.

## 9.6 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. Run the following command:

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

We have prepared some test videos in the /home/root/videos folder. You can use these for testing. 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 44: Playing an MP4 video on the RZ/G2L-SBC**

## 9.7 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.

### 9.7.1 Hardware Interfacing

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

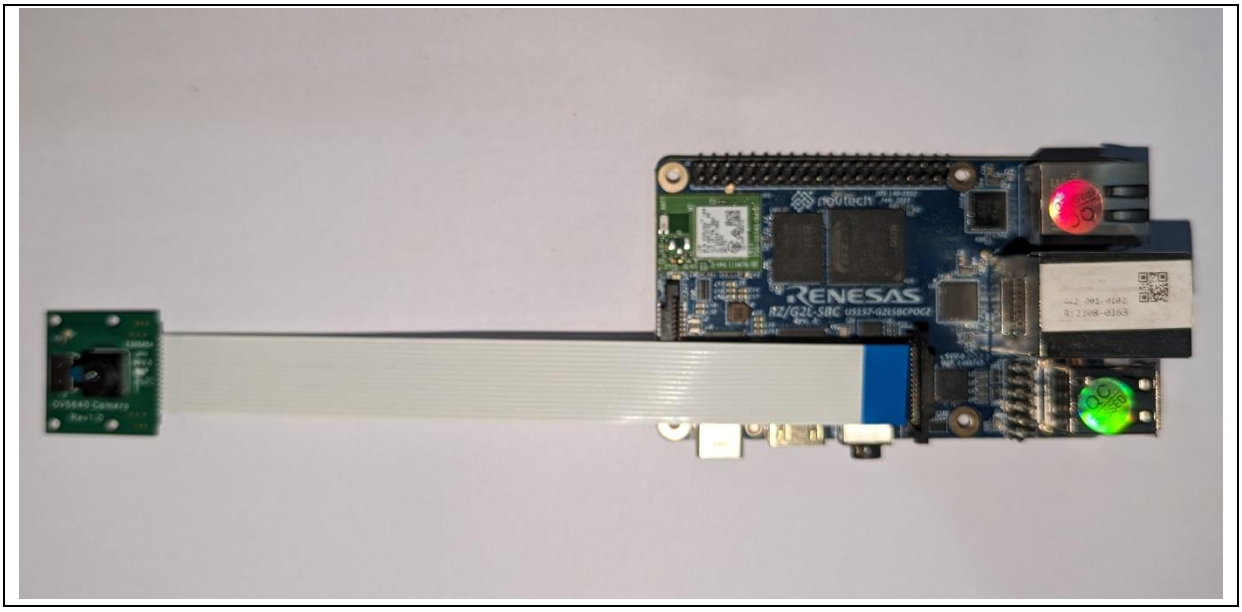



**Figure 45: Orientation of the camera module. Blue stripe upward.**

The black notch must be pulled up to unlock it.

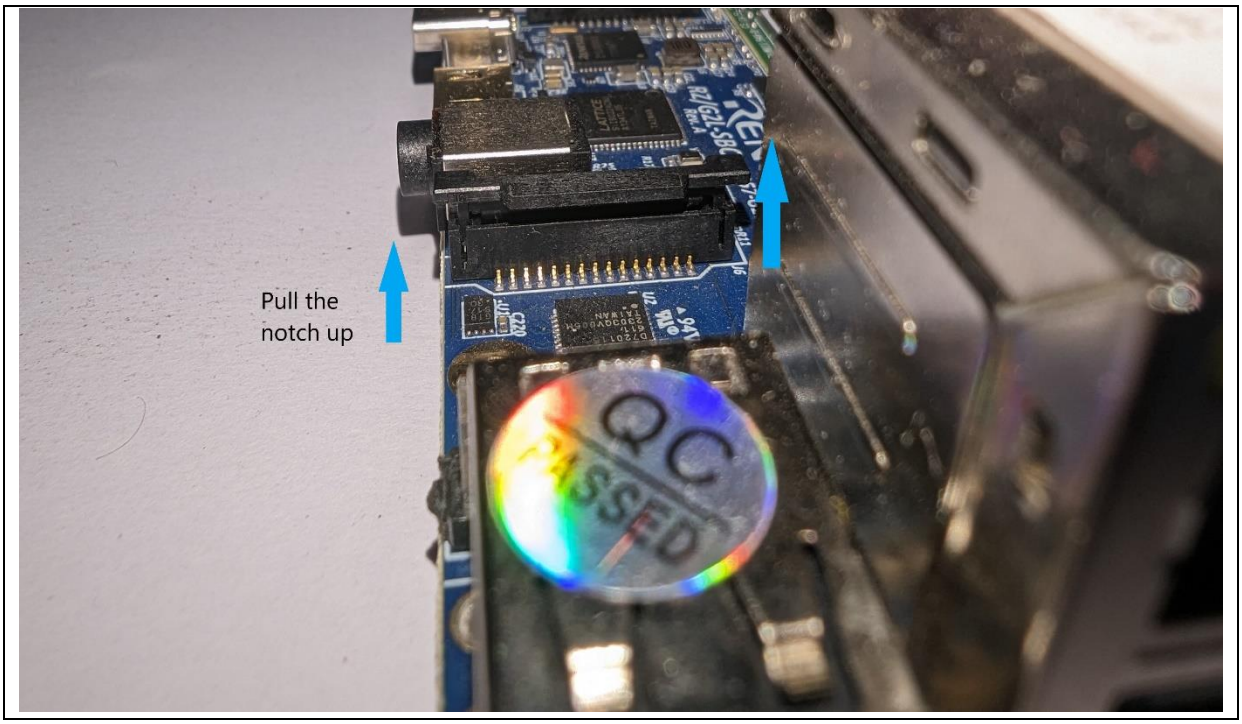



**Figure 46: Pull the notch up to unlock it.**

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

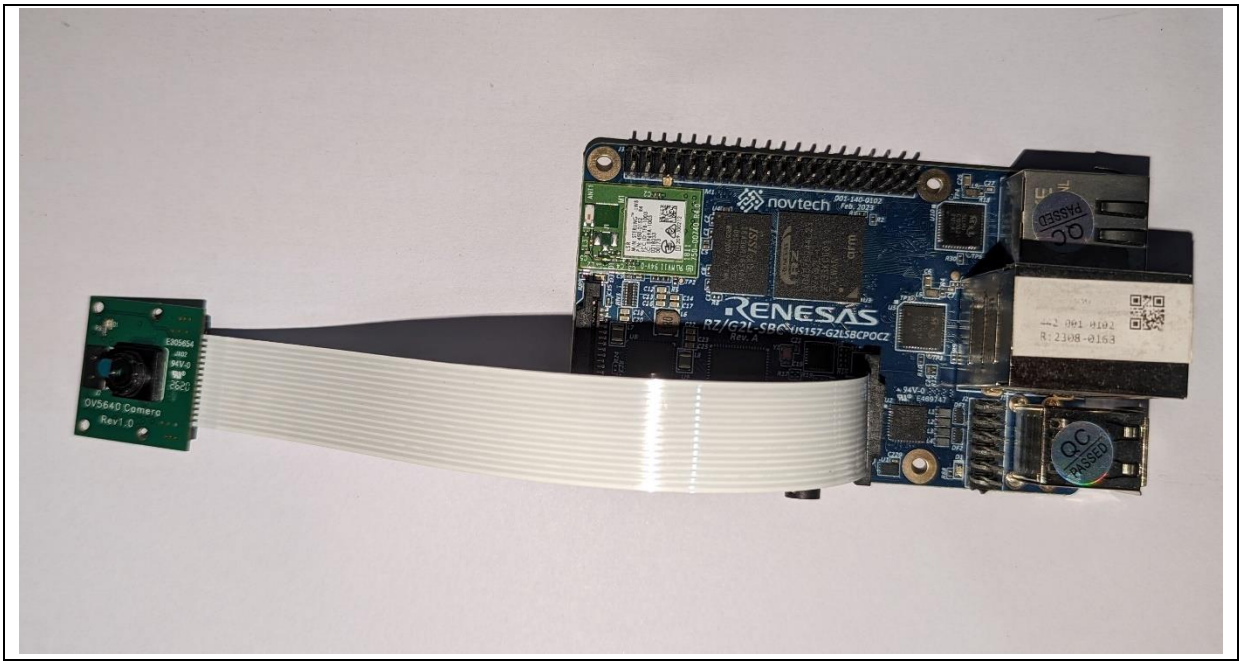



**Figure 47: CSI module inserted.**

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

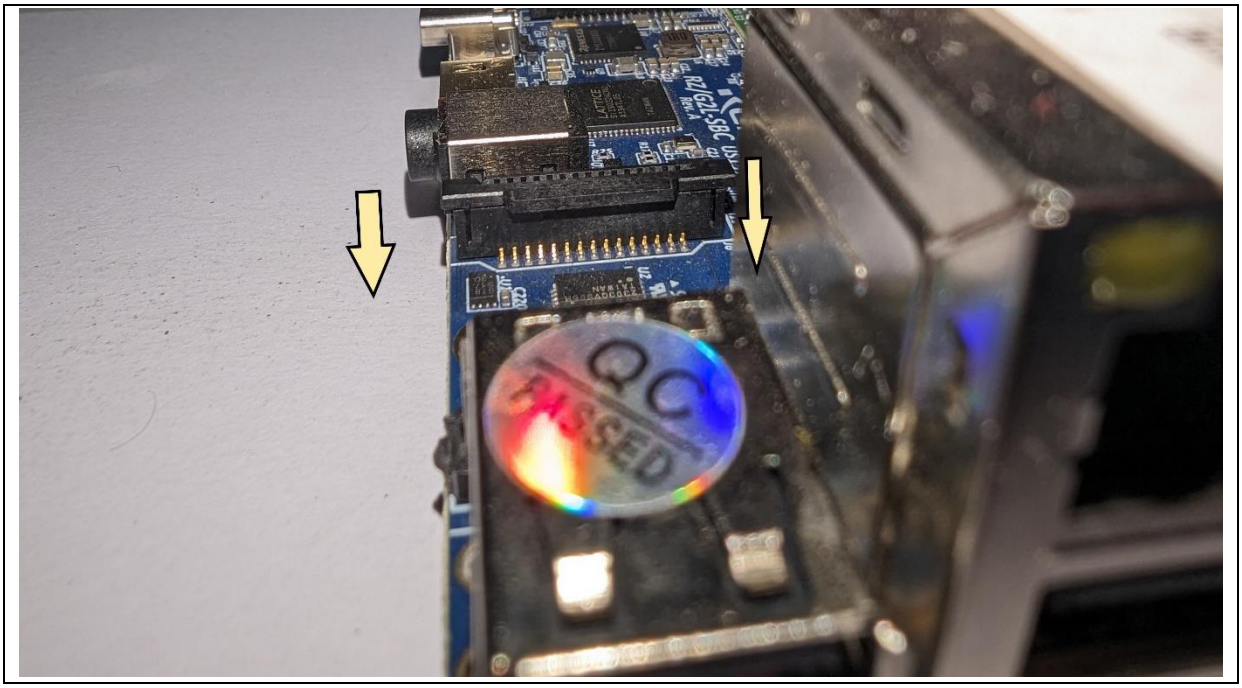



**Figure 48: Push down the notch to lock it when you have inserted the flat cable**

### 9.7.2 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.

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

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

- 1280x720
- 1280x960
- 1600x900
- 1920x1080
- 1920x1200
- 2560x1080

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

When use a valid resolution:

```
root@rzpi:~# ./v4l2-init.sh 1920x1080
Link CRU/CSI2 to ov5640 1-003c with format UYVY8_2X8 and resolution 1920x1080
```

When no resolution is specified:

```
root@rzpi:~# ./v4l2-init.sh
No resolution specified. Using default resolution: 1280x960
Link CRU/CSI2 to ov5640 1-003c with format UYVY8_2X8 and resolution 1280x960
```

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

Run the following to initiate a video capture session and preview the video on the screen.

```
root@rzpi:~# gst-launch-1.0 v4l2src device=/dev/video0 ! videoconvert !
waylandsink
```

This will start a continuous stream of camera feed to the active video display.

## 9.8 Package Management

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

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

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

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

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

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

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

   Remember that sources don’t have to be a single origin. It’s 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@rzpi:~# apt-get install <package-name>
```

### 9.8.2 Using DPKG to install packages

The utility ‘dpkg’ is the low-level package manager for Debian-based systems. It is the local system wide package manager. It handles installation, removal, provisioning, indexing and other aspects of packages installed on the system. However, it does not perform any cloud operations. Dpkg also doesn’t 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@rzpi:~# dpkg -i <package>.deb
```

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

```
root@rzpi:~# apt-get install -f
```

## 9.9 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@rzpi:~# pip3 install <package_name>
```

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

```
root@rzpi:~# pip3 install requests
```

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

```
root@rzpi:~# 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@rzpi:~# pip3 list
```

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

## 9.10 Python GUI programming with Tkinter

This section provides a step-by-step guide on creating a basic graphical user interface (GUI) application using Tkinter; the standard Python interface to the Tk GUI toolkit. Tkinter is included with Python, so

you don't need to install any additional libraries. It's a great choice for building desktop applications due to its simplicity and ease of use.

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

Step 1. Create a working directory on the RZ/G2L-SBC where you will develop and store your Python application.

```
root@rzpi:~# mkdir ~/python_apl
root@rzpi:~# cd ~/python_apl
```

Step 2. Create a new Python file (e.g., main.py) in your work directory.

```
root@rzpi:~/python_apl# vi main.py
```

Step 3. Develop a Simple Python GUI Application with tkinter

- Import the tkinter module:
  ```
  import tkinter as tk
  ```
  This imports the Tkinter module and gives you access to its classes and functions.
- 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()
```

Step 4. Run the application

- Ensure the RZ/G2L SBC is connected to an external display. If you're using an environment where the display is not automatically set, you may need to set the DISPLAY environment variable as follows:

```
root@rzpi:~# export DISPLAY=:0
```

- Run the Python application:

```
root@rzpi:~# python3 main.py
```

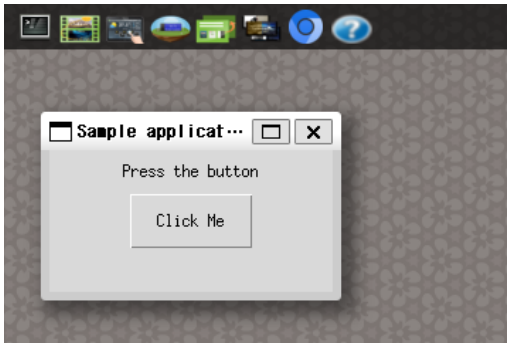



**Figure 49: Initial GUI layout**

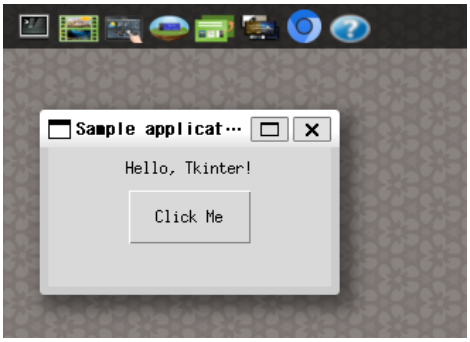



**Figure 50: After the button 'Click me' is clicked**

## 9.11 Chromium web browser

The distro image in this release comes with open-source chromium browser. It is a fully featured version.

You can use the following command line to launch a chromium window with a url it will load on launch.

```
root@rzpi:~# chromium --no-sandbox --in-process-gpu https://google.com
```

Please note that

Note:

It is a must to have an input device (USB mouse or touchscreen) plugged in before you start the browser. The lack of an input device will cause a "Segmentation fault".

Chromium can be launched from the taskbar at the top of the screen as shown below:

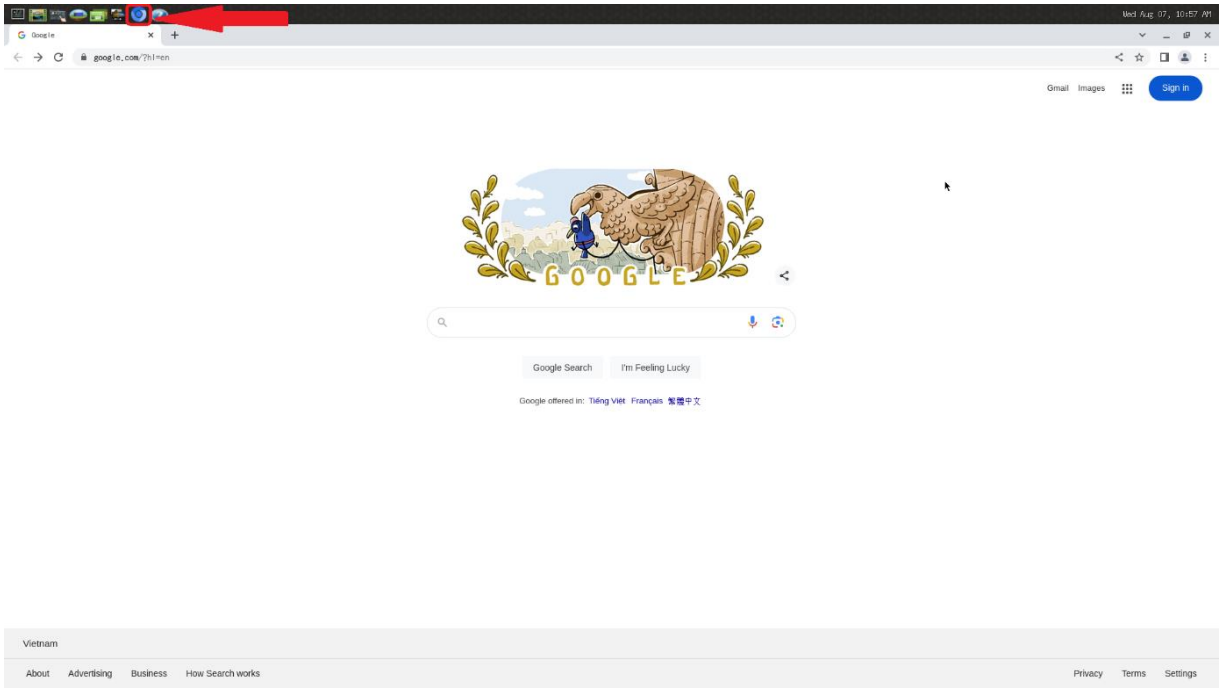



**Figure 51: Chromium web browser on RZ/G2L-SBC**

# 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.

Prerequisite:

- x86 based PC or rack server (non x86 systems can be used at user discretion).
- An Ubuntu / Debian based OS loaded on to the server.
- Router that performs DHCP.
- RZ/G2L-SBC
- FTDI based USB-UART cable.
- Ethernet cables.

Please note that this requires a wired connection and cannot be performed on Wi-Fi.

Step 1: Install a TFTP server using the following command:

```
$ sudo apt update
$ sudo apt install tftpd-hpa
```

Step 2: Create a TFTP directory and set the appropriate permissions.

```
$ sudo mkdir /tftpboot
$ sudo chmod 755 /tftpboot
```

Step 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 to the file.

```
# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"/tftpd-hpa
```

Step 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 filesystem (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.

Step 1: Install NFS server and NFS client package if it's not already installed on your host PC:

```
$ sudo apt update
$ sudo apt install nfs-kernel-server nfs-common
```

Step 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.

Step 3: After editing /etc/exports, run the following command to export the directories:

```
$ sudo exportfs -a
```

Step 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, you will configure the U-Boot environment for network settings. This includes specifying the Ethernet device and setting the server and device IP addresses.

Step 1: Enter U-Boot’s interactive command prompt for configuration.

You can achieve this 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: RZpi
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
=>
```

Step 2: Enter Specify the Ethernet device (eth1) to use for the network connection. For example,

```
=> setenv ethact ethernet@11c30000
```

Step 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 the figure below:

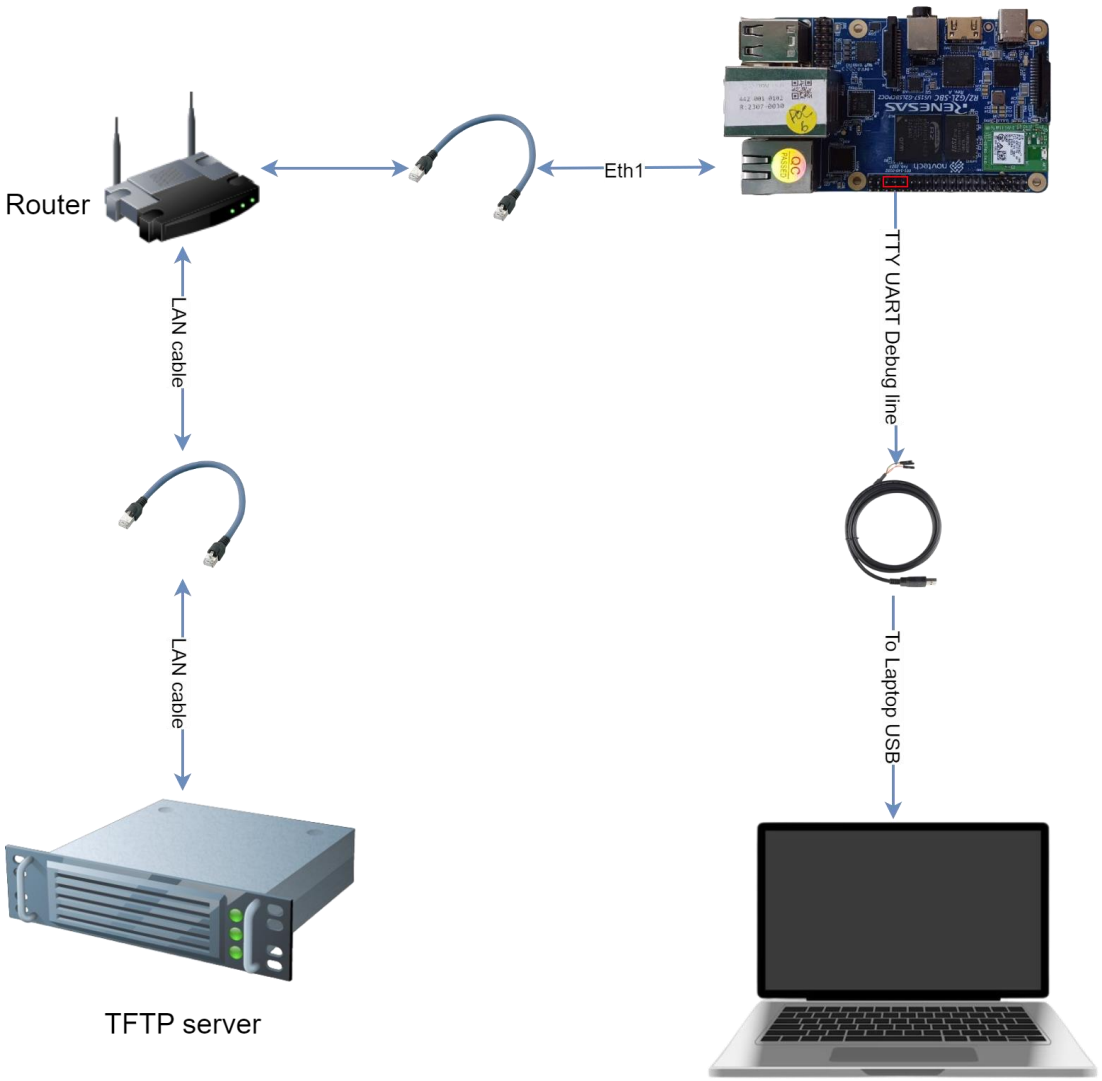



**Figure 52: TFTP boot setup**

Step 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
│   ├── rzpi-can.dtbo
│   ├── rzpi-dsi.dtbo
│   ├── rzpi-ext-i2c.dtbo
│   ├── rzpi-ext-spi.dtbo
│   └── rzpi-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
└── rzpi.dtb
```

Step 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'
```

Step 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', 'rzpi.dtb'** and '**rzpi-ext-spi.dtbo'** files.

```
=> setenv bootcmd 'tftp 0x48080000 rzsbc/Image; tftp 0x48000000 rzsbc/rzpi.dtb;
tftp 0x48010000 rzsbc/overlays/rzpi-ext-spi.dtbo; booti 0x48080000 - 0x48000000 -
0x48010000'
```

Step 4: Save the changes to the environment variables so they persist across reboots:

```
=> saveenv
```

Step 5: Initiate the boot progress by running **bootcmd**:

```
=> run bootcmd
```

If you set everything up correctly, you will be able to boot the images 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/rzpi.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/rzpi-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

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:

Step 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
```

Step 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"
```

Step 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 similar to 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 login 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 haven't 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. Using MobaXTerm
   - Download and install MobaXterm if you haven't already.
   - Select "Session" > "SSH" and enter the device's IP address.

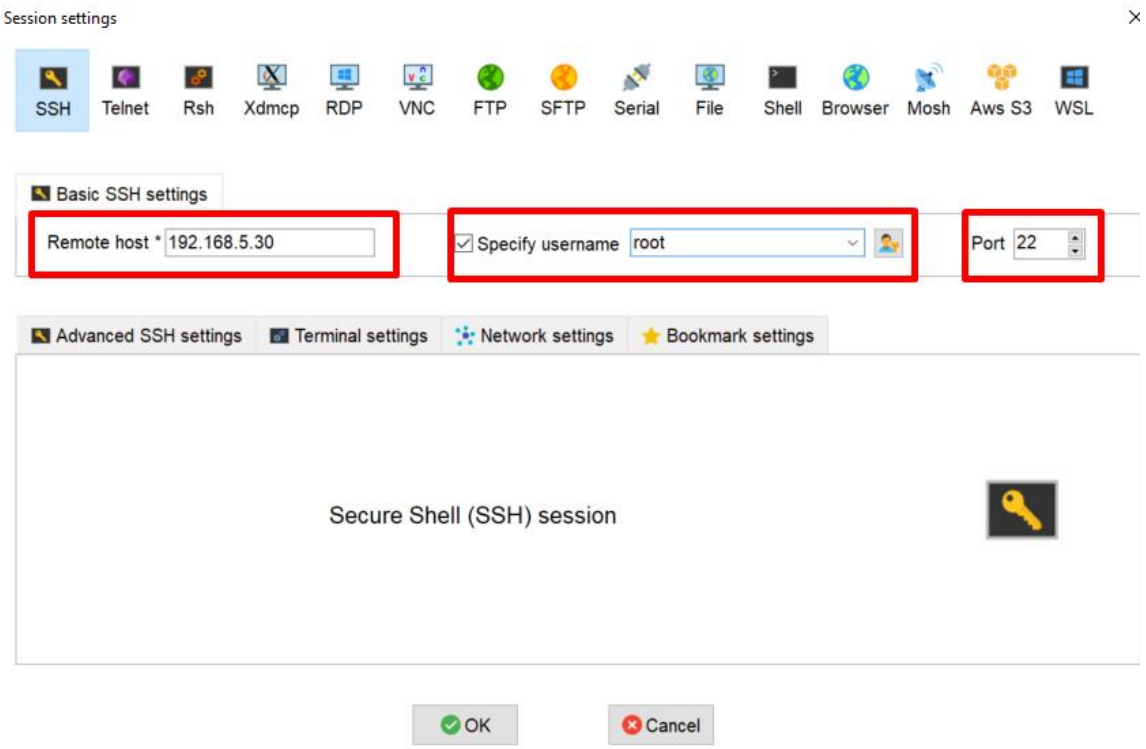



**Figure 53: SSH in settings in Mobaxterm**

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

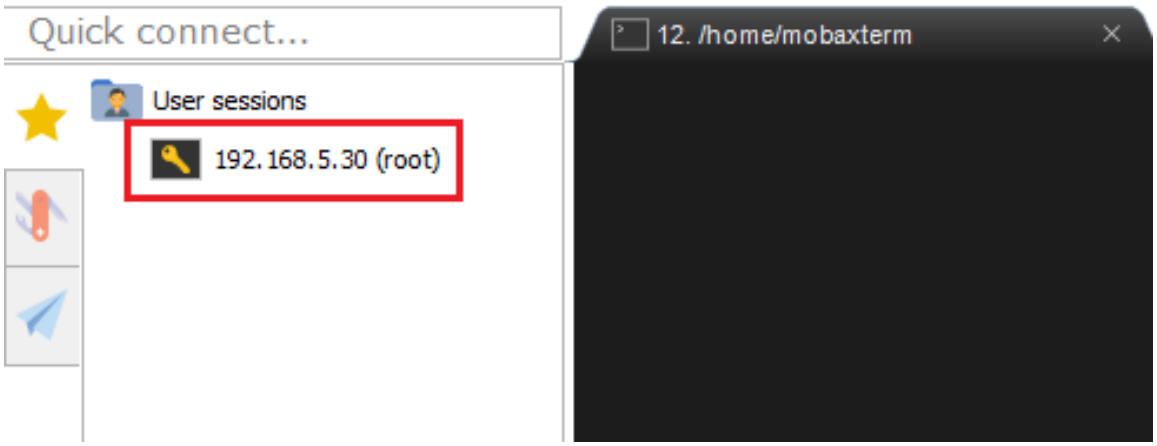



**Figure 54: Connect to SSH session in Mobaxterm**

### 11.3.2 SSH from Linux host

Step 1: Open a terminal and run

```
$ ssh username@<device_ip>
```

For example:

```
$ ssh root@192.168.5.30
```

Step 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 haven't 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. Using WinSCP
   - Open WinSCP and select “New Session”
   - Choose SCP as protocol then enter the remote device's IP address and the host name.

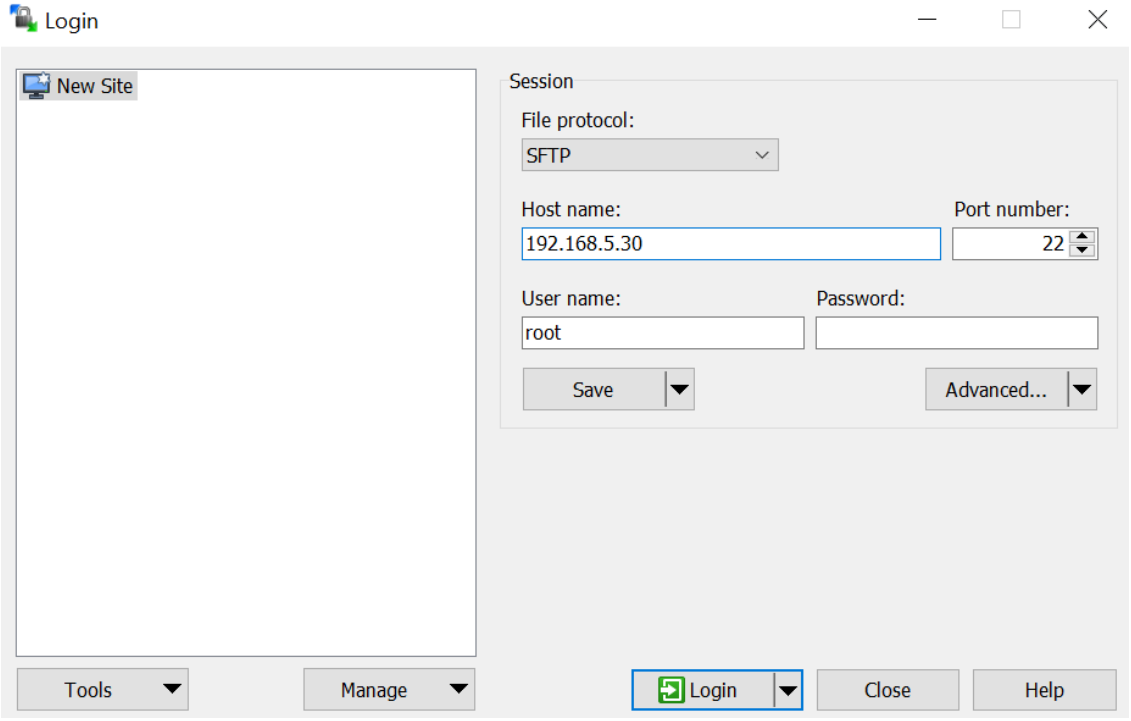



**Figure 55: Setting up WinSCP for SSH session**

   - Click ‘Login’ and choose 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.

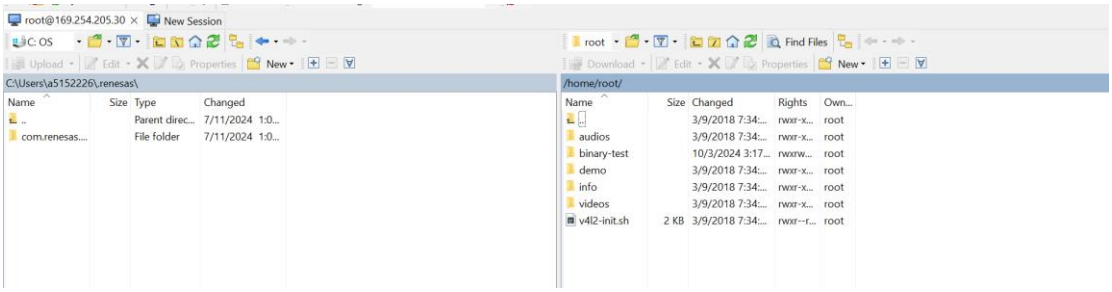



**Figure 56: Using WinSCP to transfer files**

### 11.4.2 SCP from Linux host

Step 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
```

Step 2: Type **yes** to confirm the host's authenticity when prompted.

```
$ scp hello-world root@192.168.5.30:/home/root
The authenticity of host '192.168.5.30 (192.168.5.30)' can't be established.
RSA key fingerprint is SHA256:v39PhjNp4F7HcQpwJmfNOYcC+ZZ3Yw8i1ICsL2mXUgg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.5.30' (RSA) to the list of known hosts.
```

## 11.5 Switching from OpenSSH to Dropbear

By default, the RZ/G2L-SBC uses OpenSSH. To switch from OpenSSH to Dropbear, follow these steps to modify the local.conf:

Step 1: Open the local.conf file in Yocto build configuration

Step 2: Comment the following lines in the local.conf

```
IMAGE_FEATURES_remove = "ssh-server-dropbear"
IMAGE_FEATURES_append = "ssh-server-openssh"
```

This will remove OpenSSH and enable Dropbear for the board.

Step 3: Rebuild and deploy the image to apply the changes.

# 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 5.2, the support script was copied from release package which helped commencing images build. The script can also support eSDK build, run the following command to start the build:

```
renesas@builder-pc:~/yocto$ ./rzsbc_yocto.sh build-sdk
```

The resulting eSDK installer will be located in `~/yocto/yocto_rzsbc_board/build/tmp/deploy/sdk`.

The eSDK installer will have the extension ".sh".

```
renesas@builder-pc:~/yocto/yocto_rzsbc_board/build/tmp/deploy/sdk$ ls
poky-glibc-x86_64-core-image-qt-aarch64-rzpi-toolchain-ext-3.1.26.sh
poky-glibc-x86_64-core-image-qt-aarch64-rzpi-toolchain-ext-3.1.26.host.manifest
poky-glibc-x86_64-core-image-qt-aarch64-rzpi-toolchain-ext-3.1.26.testdata.json
poky-glibc-x86_64-core-image-qt-aarch64-rzpi-toolchain-ext-3.1.26.target.manifest
```

Note:

The SDK build may fail depending on the build environment. At that time, please 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.

Follow the steps below to set up your environment:

Step 1. Install toolchain on a Host PC.

```
renesas@builder-pc:~/yocto$ sh ./build/tmp/deploy/sdk/poky-glibc-x86_64-core-
image-qt-aarch64-rzpi-toolchain-ext-3.1.26.sh
```

**Note:** You cannot install the eSDK as root because BitBake won't 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:~/yocto$ sh ./build/tmp/deploy/sdk/poky-glibc-x86_64-core-
image-qt-aarch64-rzpi-toolchain-ext-3.1.26.sh
Poky (Yocto Project Reference Distro) Extensible SDK installer version 3.1.26
==========================================================================
Enter target directory for SDK (default: ~/poky_sdk): ~/esdk/3.1.26
You are about to install the SDK to "/home/renesas/esdk/3.1.26". 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.
$ . ~/esdk/3.1.26/environment-setup-aarch64-poky-linux
$ . ~/esdk/3.1.26/environment-setup-armv7vet2hf-neon-vfpv4-pokymllib32-linux-
gnueabi
```

Step 2. Set up cross-compile environment. The following command assumes that you installed the SDK in `~/esdk/3.1.26`.

```
renesas@builder-pc:~$ source ~/esdk/3.1.26/environment-setup-aarch64-poky-linux
SDK environment now set up; additionally you may now run devtool to perform
development tasks.
Run devtool --help for further details.
```

> Note:
>
> 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/3.1.26/environment-setup-aarch64-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 build automation, testing, packaging and installation of software by using a compiler-independent method

If the host PC doesn't 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.

Step 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
```

Step 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.

Step 3: Create your application

```
renesas@builder-pc:~/cmake_helloworld$ vi src/helloworld.c
```

Then, copy the below contents to the file:

```
#include <stdio.h>

int main(int argc, char** argv)
{
    printf("\nHello World!\n");
    return 0;
}
```

Step 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/3.1.26`, the completed configuration file will resemble the following:

```
cmake_minimum_required(VERSION 3.10)
project(HelloWorld C)

set(CMAKE_C_COMPILER
/home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gcc)
set(CMAKE_CXX_COMPILER
/home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-g++)

# Sysroot path
set(CMAKE_SYSROOT /home/renesas/esdk/3.1.26/tmp/sysroots/rzpi)

add_executable(helloworld src/helloworld.c)
```

Step 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 complete, 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. And you can then 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.

Step 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/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-gcc)
set(CMAKE_CXX_COMPILER
/home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-g++)

# Sysroot path
set(CMAKE_SYSROOT /home/renesas/esdk/3.1.26/tmp/sysroots/rzpi)

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)
```

Step 2: Package the C program into Debian package installer

```
renesas@builder-pc:~/cmake_helloworld$ cd build/
renesas@builder-pc:~/cmake_helloworld$ cmake ../
-- Toolchain file defaulted to
'/home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/share/cmake/OEToolchainConfig.
cmake'
-- The C compiler identification is GNU 9.5.0
-- Check for working C compiler:
/home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-gcc
-- Check for working C compiler:
/home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-
poky-linux-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- 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$ 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 complete, 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
```

Step 3: Ship the Debian package installer to RZ/G2L-SBC.

The Debian package installer “helloworld-1.0.0-Linux.deb” can be transfered to the RZ/G2L-SBC using SCP tool as below or other methods, such as an 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.

Step 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
```

Step 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.

Step 3: Modify the python program, this program same as Tkinter example in section 8.9 Python GUI programming with Tkinter

Copy this example content and paste to this python file

```
renesas@builder-pc:~/cmake_python$ vi src/main.py
```

Step 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

python3 /usr/local/share/tkinter_example/main.py
```

Step 5: Configure the CMakeLists.txt for packaging Python program

```
renesas@builder-pc:~/cmake_python$ vi CMakeLists.txt
```

Then, copy the below contents 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)
```

Step 6: Packaging the program

```
renesas@builder-pc:~/cmake_python$ cd build/
renesas@builder-pc:~/cmake_python/build$ cmake ../
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- 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 complete, 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 SCP tool as below or other methods, such as an 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 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>
…
rzpi login: root
root@rzpi:~# ./helloworld

Hello, World!
root@rzpi:~#
```

## 13.5 Install and Run Debian application packages by using DPKG

After shipping the Debian package installer to the RZ/G2L-SBC, the package can be installed using dpkg.

The steps are:

Step 1: List out all available .deb files to make sure all .deb file have been shipped to the RZ/G2L-SBC

Step 2: Install the C program by running `dpkg -i helloworld-1.0.0-Linux.deb`

Step 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>
…
rzpi login: root
root@rzpi:~# ls
audios  demo  helloworld-1.0.0-Linux.deb  tkinter_example-1.0.0-Linux.deb  images
info  v4l2-init.sh  videos
root@rzpi:~# 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@rzpi:~#
root@rzpi:~# 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@rzpi:~# dpkg -l
ii  helloworld     1.0.0        arm64        HelloWorld built using CMake
ii  tkinter_example 1.0.0       arm64        TkinterExample built using CMake
```

This completes the installation. The applications are ready for use.

There are a few ways to run the application:

1. directly from installation location
2. call it from /usr/local/bin/<your_application>.
3. Call it directly from anywhere, if it's installed within the PATH search.

The following command lists all the files that were installed with their full paths:

```
root@rzpi:~# dpkg -L helloworld
/usr
/usr/local
/usr/local/bin
/usr/local/bin/hello
root@rzpi:~# /usr/local/bin/hello
Hello, World!
```

For applications that have a graphical interface, the display id needs to be set in the environment. For this reason, export the DISPLAY if you're using an environment where the display is not automatically set as shown below:

```
root@rzpi:~# export DISPLAY=:0
root@rzpi:~# dpkg -L tkinter_example
/usr
/usr/local
/usr/local/bin
/usr/local/bin/tkinter_example
/usr/local/share
/usr/local/share/tkinter_example
/usr/local/share/tkinter_example/main.py
root@rzpi:~# /usr/local/bin/tkinter_example
```

# 14. Remote debugging using GDBServer

In this section, GDBServer will be 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/3.1.26/environment-setup-aarch64-poky-linux
SDK environment now set up; additionally you may now run devtool to perform
development tasks.
Run devtool --help for further details.
```

> Note:
>
> 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/3.1.26/environment-setup-aarch64-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
```

Please 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@rzpi:~# 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@rzpi:~# ifconfig eth1
eth1: 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 0xaaaaaaaaa07cc: 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@rzpi:~#
```

### 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 RZ-G2L/SBC.

Here's how to get started:

1. Install the C/C++ Extension (If have not 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.

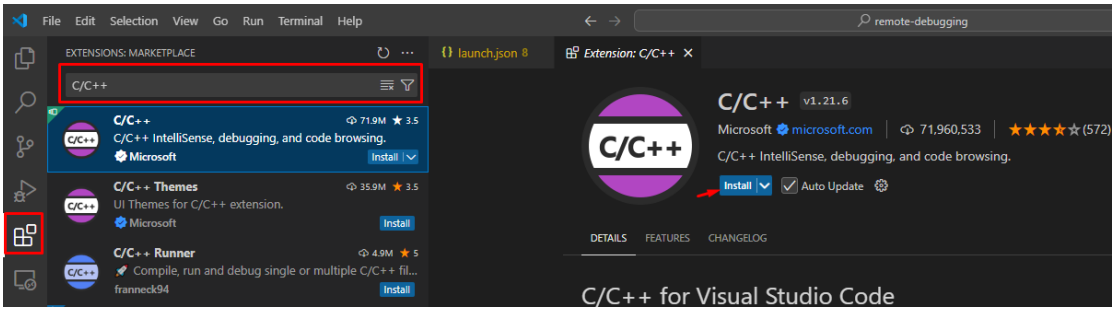



**Figure 57: C/C++ Extension in VSCode**

2. Create a Workspace:
   - Create a new workspace (you can name it remote-debugging).
   - Create a folder within this workspace and place your program file, hello-gdbserver.c in it.
   - Build the execution file using eSDK, we assume that you have source 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.

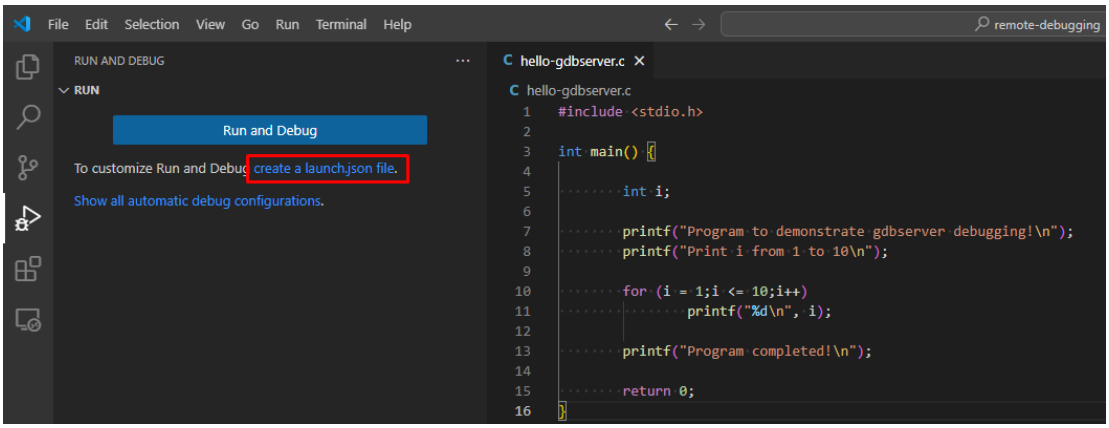



**Figure 58: Create a launch.json file in VSCode Debugger**

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

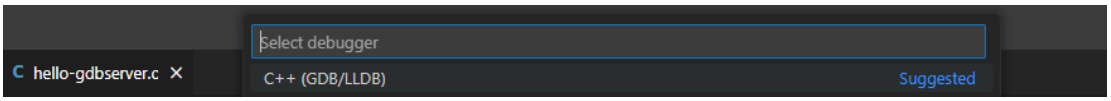



**Figure 59: Select C++ GDB as Debugger**

- Place the content as below to 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/3.1.26/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@rzpi:~# 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 breakpoint within the hello-gdbserver.c file in VSCode.
   - Click the Start Debugging button (green play icon) to begin the debugging session.

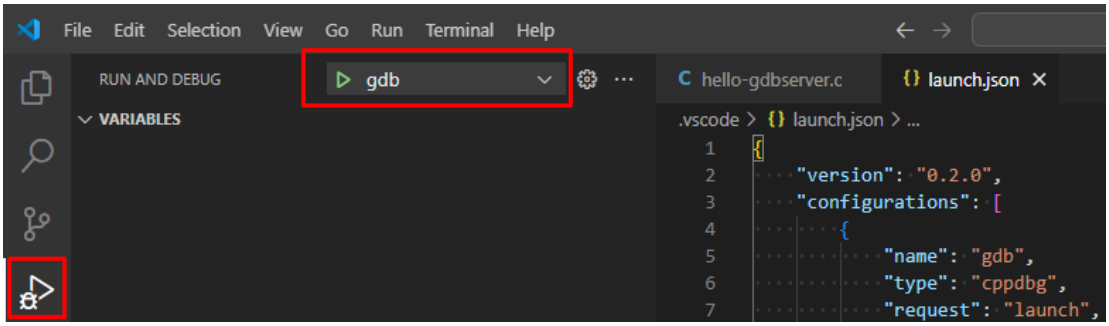



**Figure 60: Running the Debugger for Remote Debugging in VSCode**

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

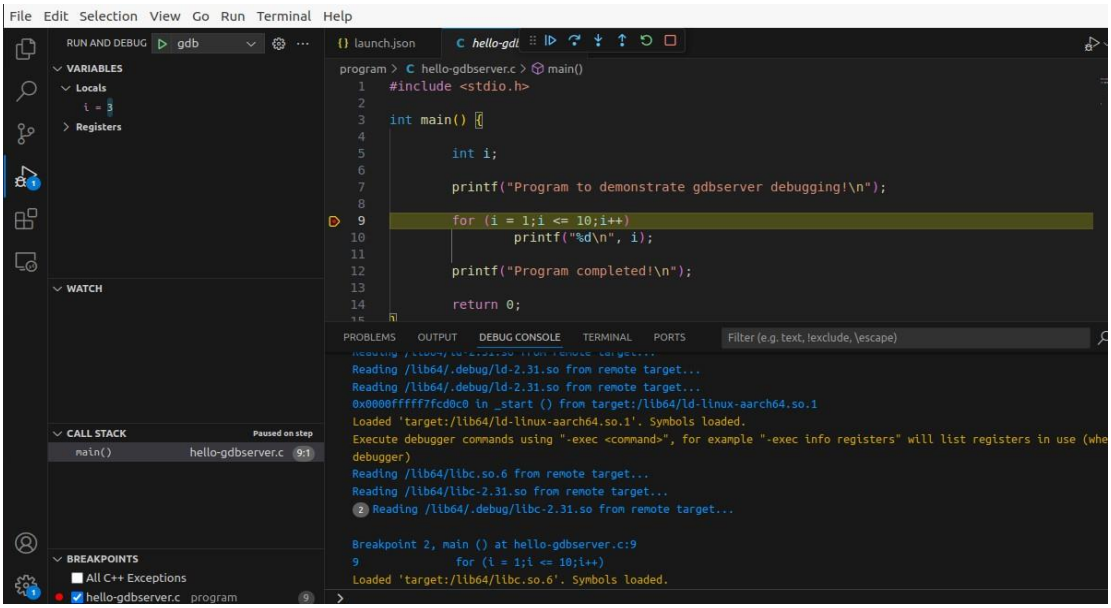



**Figure 61: Step through each step in Debug Mode in VSCode**

### 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.

Step 1: Install the Eclipse IDE (if not already installed), you can follow these official steps on Eclipse website: Eclipse Installer 2024-09 R | Eclipse Packages

Step 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

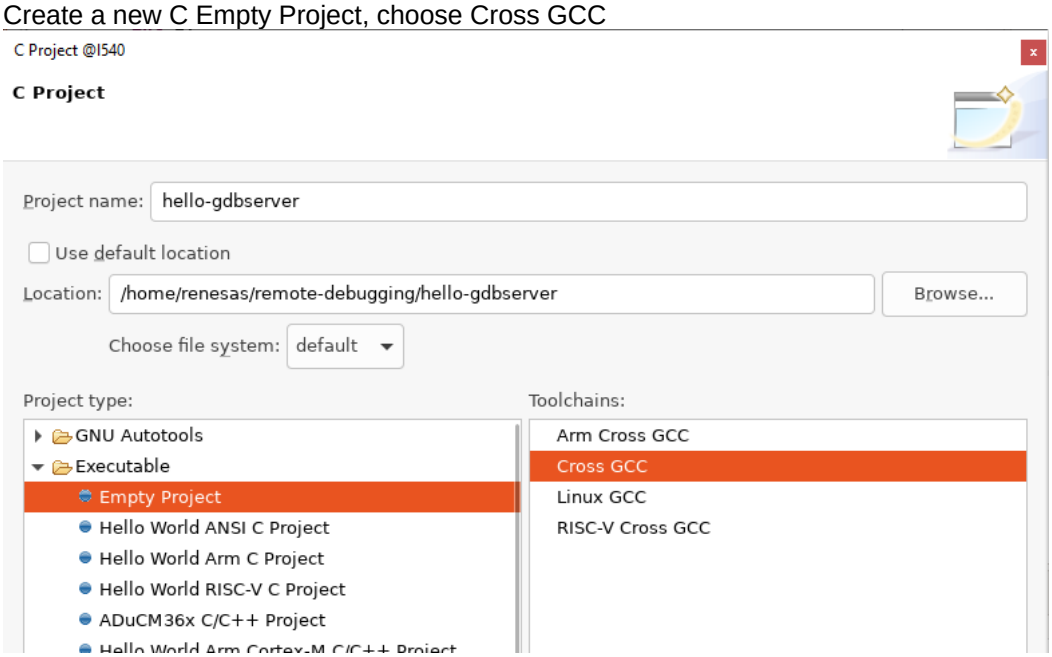



**Figure 62: Create a C Project in Eclipse**

Click Next then Finish and paste the content from hello-gdbserver.c into the C file.

Step 3: Configure the Cross Toolchain

- Go to Project > Properties.
- In the left pane, select C/C++ Build > Settings.

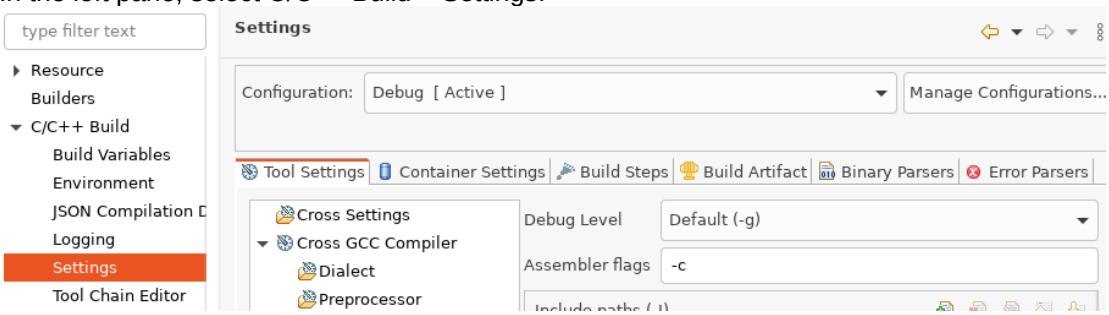



**Figure 63: Configuring the cross toolchain in Eclipse project properties**

- 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/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux

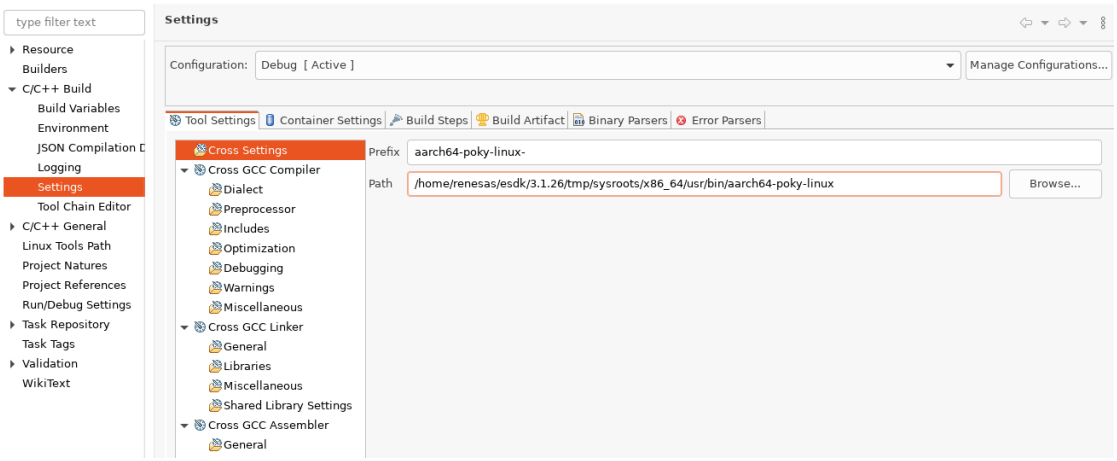



**Figure 64: Configuring cross compiler settings in Eclipse tool settings**

- In the Includes section, specify the include paths:
  - Include paths: /home/renesas/esdk/3.1.26/tmp/sysroots/rzpi/usr/include

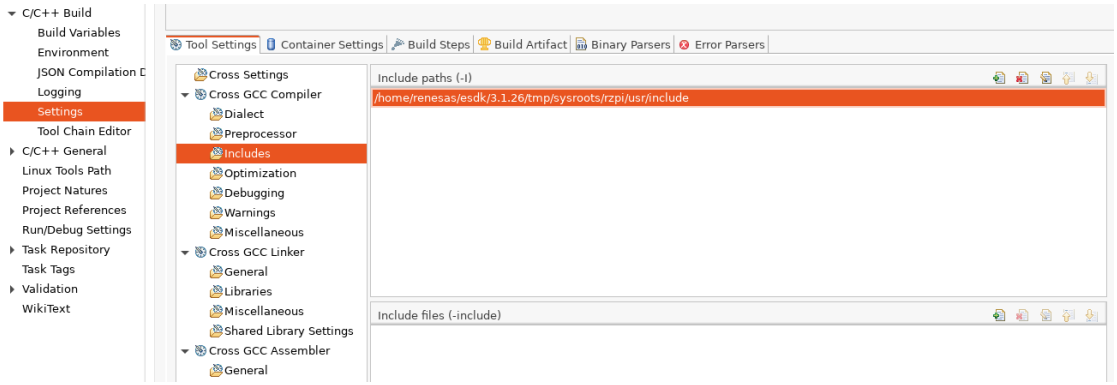



**Figure 65: Configuring includes path in Eclipse tool settings**

- In the Cross GCC Linker section, go to Libraries and specify the library search path:
  - Library search path: /home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/lib

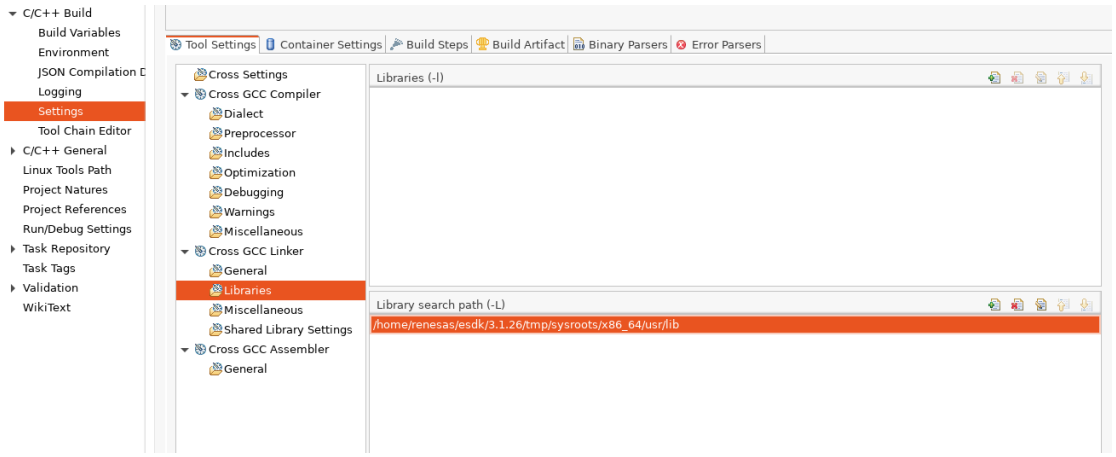



**Figure 66: Configuring libraries path in Eclipse tool settings**

- In the Miscellaneous section, specify the linker flags:
  - Linker flags: --sysroot=/home/renesas/esdk/3.1.26/poky_sdk/tmp/sysroots/rzpi

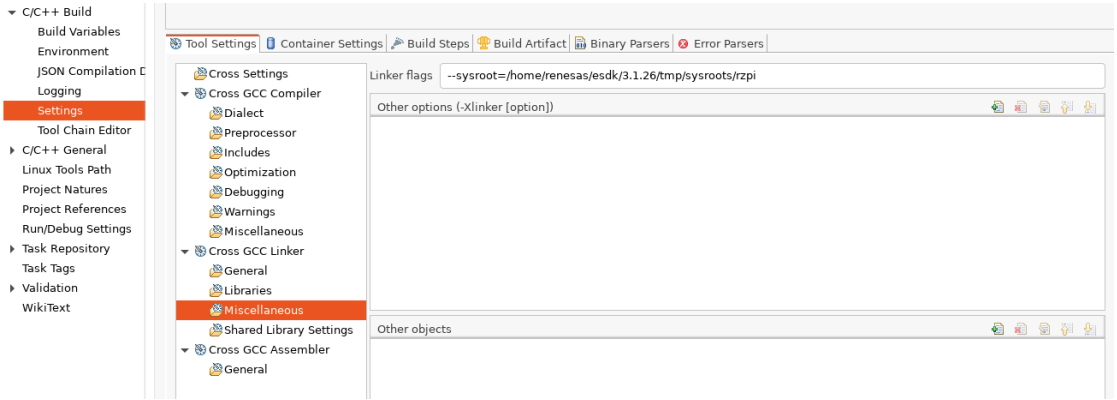



**Figure 67: Configuring linker flags for Sysroot in Eclipse tool settings**

Step 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, in Connection Type, select Remote and click Edit

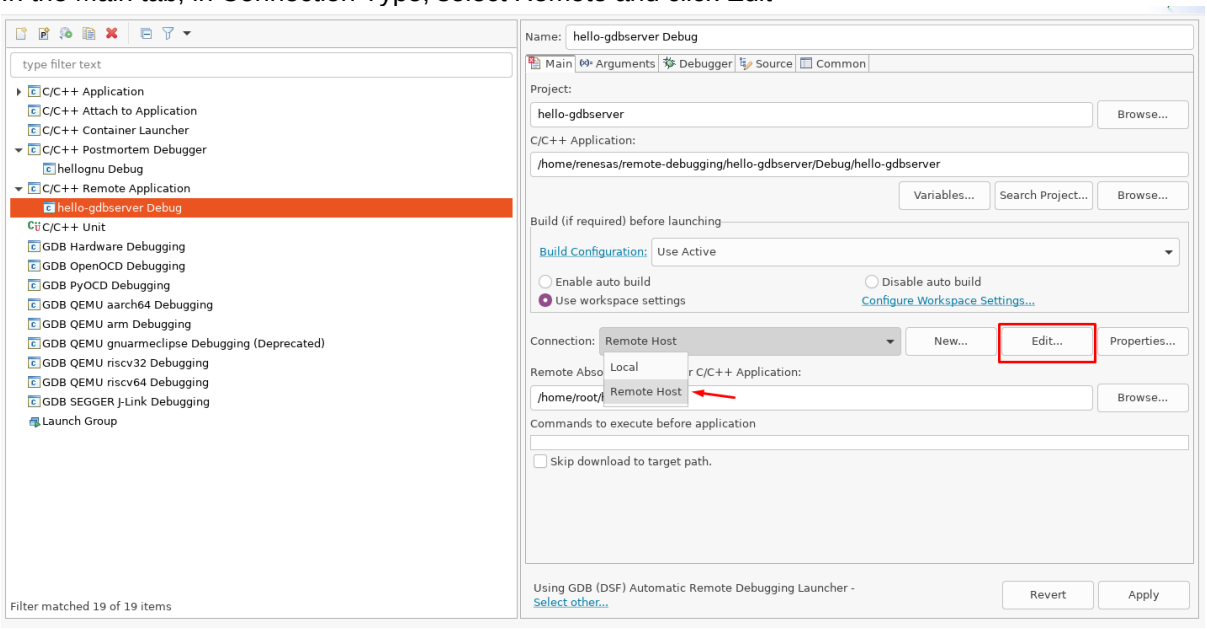



**Figure 68: Debug configuration settings in Eclipse**

- Host: Enter the IP address of RZ/G2L-SBC.
- User: Enter the user name of 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.

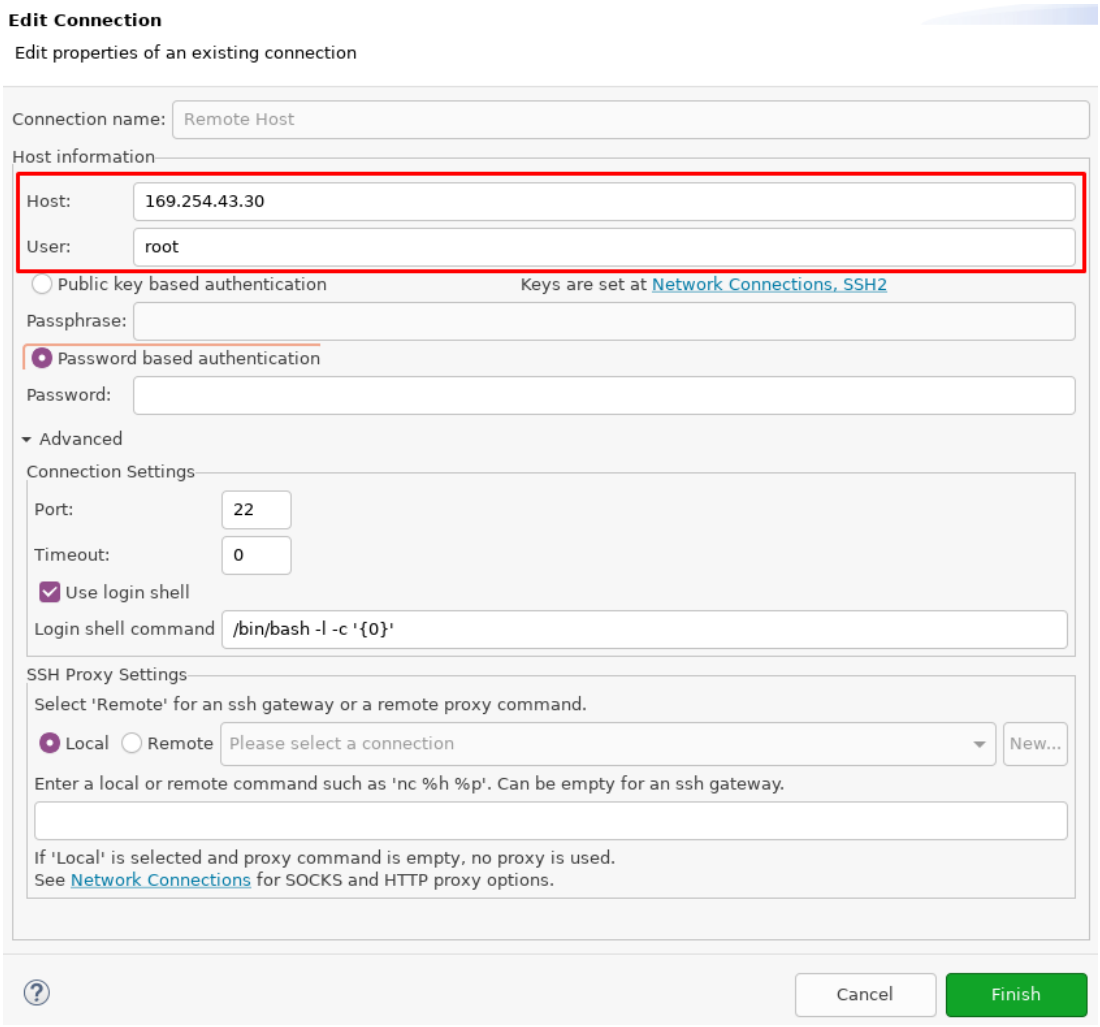



**Figure 69: Configuring SSH connection settings in debug configurations**

- 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 70: Configuring remote absolute file path in debug configurations**

- In the Debugger tab:
  - In GDB Debugger: Provide the path to your cross-compiled GDB (e.g., /home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb).

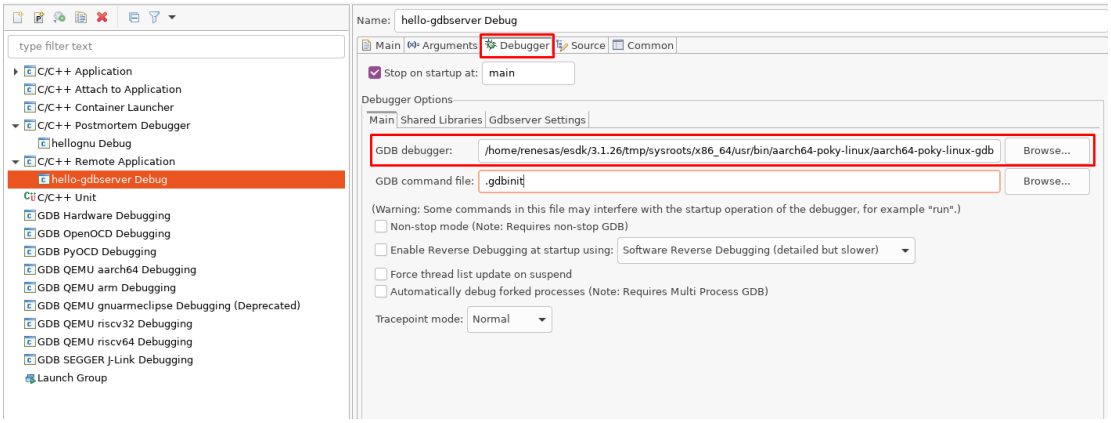



**Figure 71: Configuring GDB debugger in Eclipse debug configurations**

Step 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.

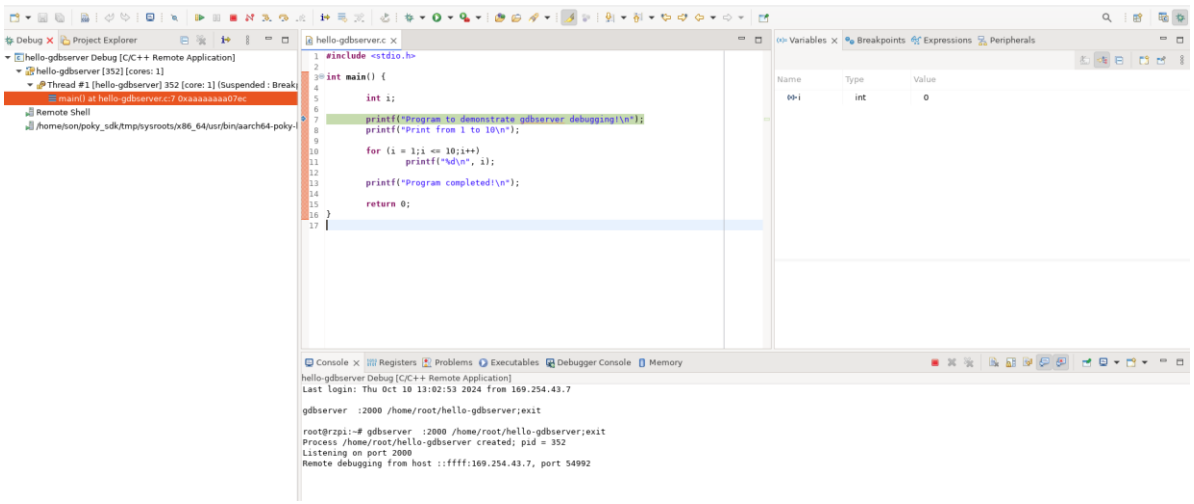



**Figure 72: Start the debugging session in Eclipse**

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

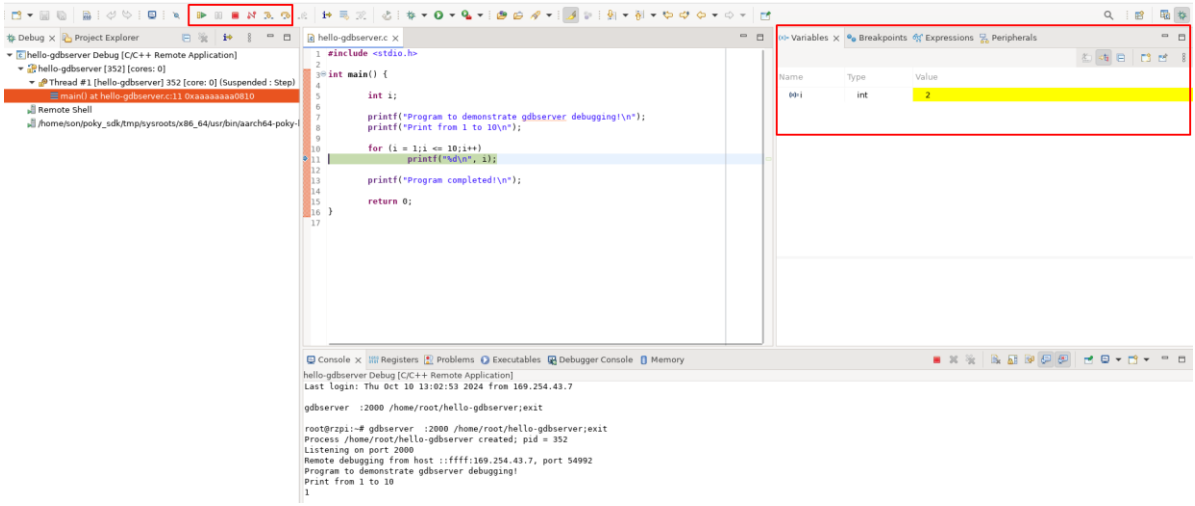



**Figure 73: Step through each step in the Debug Mode in Eclipse**

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 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.

Step 1: Create a simple C program that intentionally causes a segmentation fault. For example, the file name `segfault_example.` has below content:

```
#include <stdio.h>

int main() {
        int *ptr = NULL;

        printf("Attempting to dereference a NULL pointer...\r\n");

        *ptr = 42;

        return 0;
}
```

Step 2: Source the environment and compile the segfault_example.c program

```
renesas@builder-pc:~$ source ~/esdk/3.1.26/environment-setup-aarch64-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
```

Step 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
```

Step 4: Ensure that the system allows core dumps. To set the core dump size to unlimited, run the following command:

```
root@rzpi:~# ulimit -c unlimited
```

Step 5: Run the program to generate a core dump file or use remote debugging to obtain it.

```
root@rzpi:~# ./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@rzpi:~# ls core*
core.880
```

Transfer the core dump file back to your host machine.

Step 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) 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-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".
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 in this example

```
{
    "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/3.1.26/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) will automatically point to the exact line in the source code where the crash occurred.

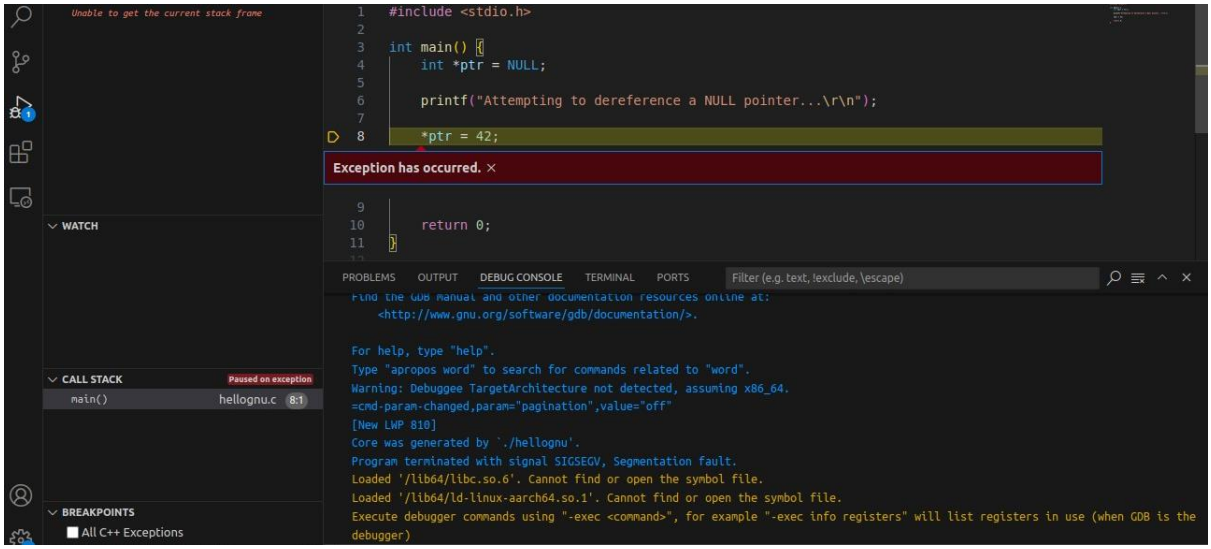



**Figure 74: Starting analysis of the Core dump file in VSCode debug mode**

### 14.4.3 Postmortem analysis on Eclipse

This subsection describes postmortem analysis using Eclipse IDE. Similar with Visual Studio Code, Eclipse allows loading core dump to inspect the application's state at the time of a crash.

Step 1: Configure Eclipse to connect to the GDB Server:

- In Eclipse, go to the Run menu and select Debug Configurations.
- Under the Debugger tab, select C/C++ Postmortem Debugger
- In the Main tab, in Core file field, click and specify where core dump file is.

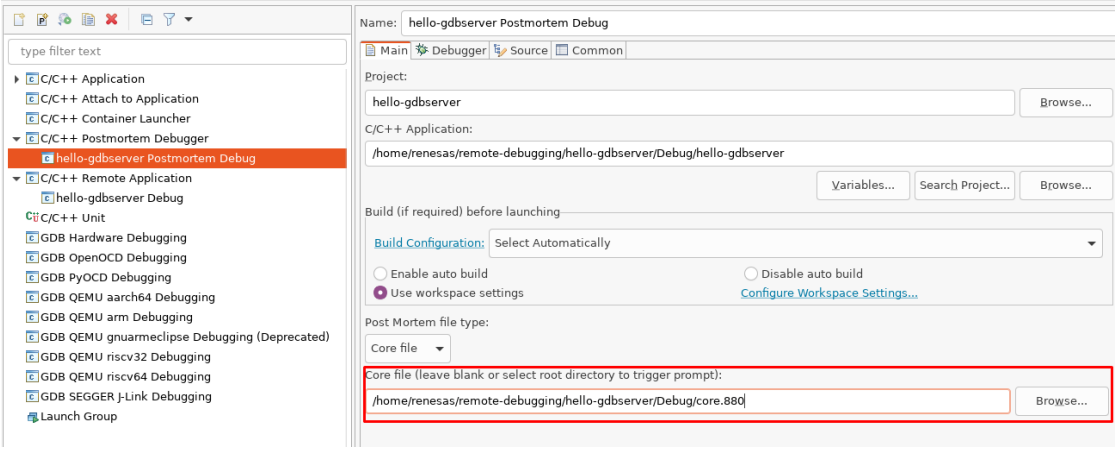



**Figure 75: Specifying the Core File in Eclipse Debugger Settings**

- In the Debugger tab:
  - In GDB Debugger: Provide the path to your cross-compiled GDB (e.g., /home/renesas/esdk/3.1.26/tmp/sysroots/x86_64/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb).

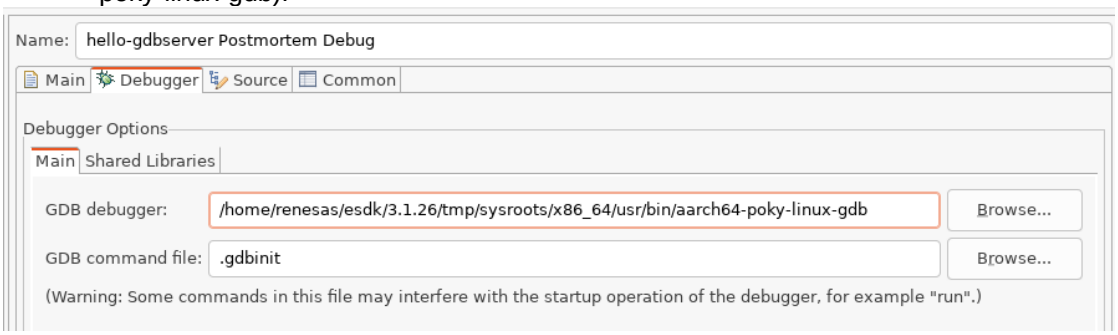



**Figure 76: Specifying the GDB Debugger in Eclipse Debugger settings**

Step 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.

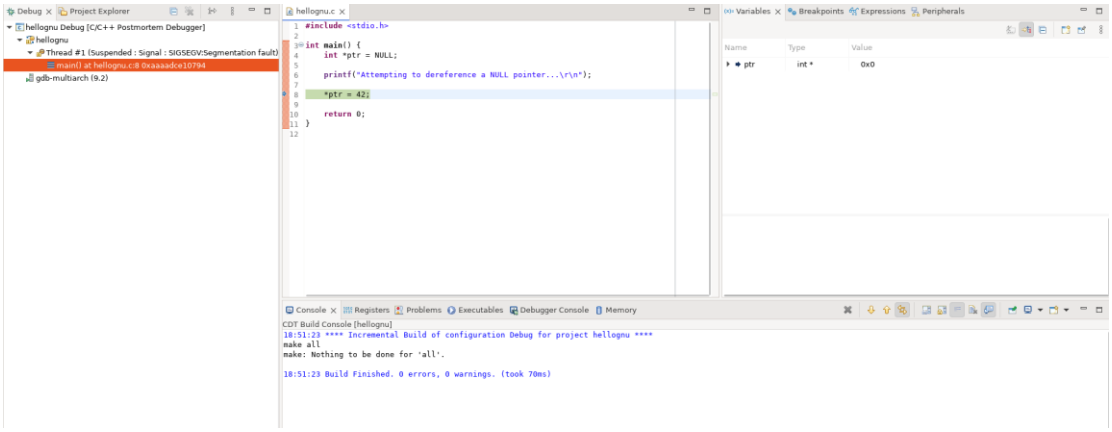



**Figure 77: Starting analysis of the Core dump file in Eclipse debug mode**

- 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. Appendix

## 15.1 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 Programming / Flashing Firmware to RZ/G2L-SBC.

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 SCIF a.k.a serial download mode by altering the bootstrapping pins.

> Note:
>
> The RZ/G2L-SBC does not have any interfaces in the main board to alter the boot mode. The boot strapping 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.

### 15.1.1 Required Hardware

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

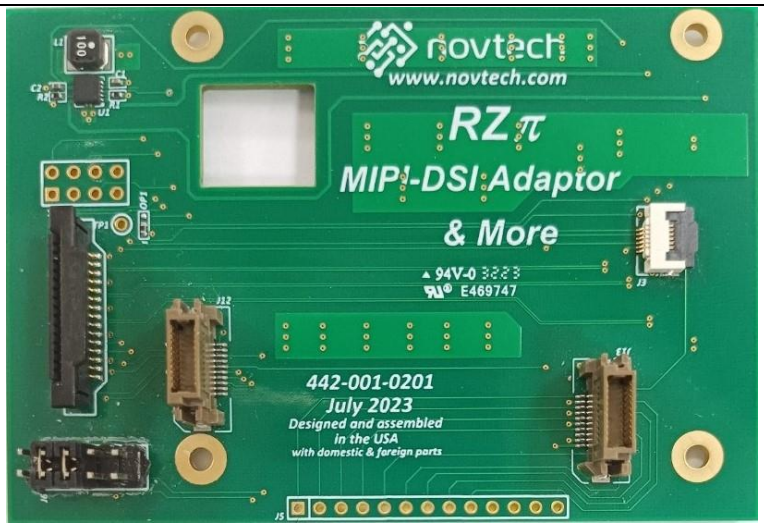



**Figure 78: Adaptor board**

### 15.1.2 Flashing Bootloader/Firmware using Linux host

The build contains a support script `bootloader_flash.py` for flashing 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.

Please 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 jumpers set to serial load boot mode.

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

Here are the steps:

Step 1. Ensure that the hardware setup is accurate.

Step 2. Start the script.

Step 3. Power on the board

```
renesas@builder-
pc:~/yocto/yocto_rzsbc_board/build/tmp/deploy/images/rzpi/host/tools/bootloade
r-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 completes.

### 15.1.3 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 jumpers set to serial load boot mode.
- ✓ Ensure that Teraterm application is installed on your windows pc.

Here are the steps:

Step 1. Ensure that the hardware setup is accurate.

Step 2. Edit config.ini and set the correct com port number.

Step 3. Start the script flash_bootloader.bat.

Step 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.

## 15.2 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 79: Root login of Linux console over UART 0.**

# 16. Troubleshooting

## 16.1 Unable to run 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.

## 16.2 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 a.k.a ‘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’.

## 16.3 Running many Qt demo apps slow down the system

QT applications are generally RAM-heavy, and their memory requirement scales up with display characteristics and object complexity. The Qt demo applications in the Linux distro image have been validated to work on the RZ/G2L-SBC over a 10” 1080p HDMI and the Waveshare 5” DSI touch display units. However, some applications can freeze or stutter, especially when other processes are running, the screen size is large, or the framerates are high. One of the limitations in this regard is the 1GiB DDR memory, which limits usable memory for GFX.

Methods to enhance QT application performance:

1. Reduce the application's memory consumption by optimizing QT for using the MALI GPU for animations and reducing the number of objects to be rendered. Simply reducing the framerate can often achieve better performance.
2. Custom board for a custom application: The RZ/G2L SoC supports 2GiB DDR4 SDRAM. If the application requires it, we recommend a custom version of the board with 2GiB DDR SDRAM memory. Please be advised that the existing board is still highly capable of running high GFX applications, as seen in the demos.

## 16.4 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, please recheck with eth1. Your individual network topology will affect the board's ability to get an IP address through DHCP.

## 16.5 '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 the section 'Wi-Fi 802.11 Module'.

## 16.6 IP configuration

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

## 16.7 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 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: RZpi
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!
=>
```

Congratulations on receiving a factory board. This is a board not updated with the newest u-boot. This is also your chance to try the steps from section **Flash bootloader on u-boot console**. Once "u-boot" is updated, this issue will be resolved.

# 17. References

## 17.1 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)

## 17.2 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

## 17.3 External resources

### 17.3.1 QT development

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

Qt documentation: Qt Documentation | Home

### 17.3.2 Yocto Project

Official Yocto manual: Yocto Project Reference Manual — The Yocto Project ® 4.3.999 documentation

### 17.3.3 Linux Kernel Documentation

The Linux Kernel documentation — The Linux Kernel documentation

### 17.3.4 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

### 17.3.5 JEDEC DDR4

DDR4 SDRAM STANDARD | JEDEC

### 17.3.6 PMOD Specification

Wiki: Pmod Interface - Wikipedia

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

### 17.3.7 Essential Linux Tutorial

Linux/Unix Tutorial (geeksforgeeks.org)

Linux/Unix Tutorial - javatpoint

UNIX / LINUX Tutorial (tutorialspoint.com)

### 17.3.8 Packaging

CMake Reference Documentation — CMake 3.30.2 Documentation

CPack — CMake 3.30.2 Documentation

### 17.3.9 Using the Extensible SDK

Using the Extensible SDK

### 17.3.10 Install Eclipse IDE

Eclipse Installer 2024-09 R | Eclipse Packages

### 17.3.11 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

### 17.3.12 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 | Jul.12.24 | — | Initial release |
| 1.1 | Sep.20.24 | — | - Update:<br>o Yocto build output hierarchy<br>o Add ALSA Mixer figure for Audio feature<br>o New Arducam 5MP OV5640 resolutions supported<br>o Chromium web browser<br>o Debian Package Manager<br>- Add new sections:<br>o Playing videos<br>o Python3-pip installation<br>o Python Tkinter programming<br>o Building the eSDK<br>o Example for using the eSDK, build and run a sample application |
| 1.2 | Oct.08.24 | — | - Add new sections:<br>o Network Boot and TFTP<br>o Using SSH and SCP for Remote Access and File Transfers<br>o Remote debugging using GDBServer |

RZ/G2L-SBC, Single Board Computer – User Manual

Publication Date: Oct.08.2024

Published by: Renesas Electronics Corporation

# RZ Family/ RZ/G Series

Renesas Electronics Corporation

R12UZxxxxEU010x