RZ/V2H Robotic Development Kit User Manual

Release 0.1

Renesas Electronics Corporation

Oct 23, 2025

Contents

1 Getting Started

1.1 Overview

WS125 Robotic Development Kit is a solution with Renesas new generation RZ/V2H MPU for AI application, which has AI inference processing performance of up to 80TOPS with multi-core CPU to run multiple OS simultaneously for high performance AI image processing.

It is also equipped with many interfaces that make it suitable for development and integration into a variety of robotic applications.

1.1.1 Software Environment

Category	Description
OS Support	Yocto 5.1 (Styhead) and Ubuntu 24.04 (available in headless and LXDE versions).
ROS 2 Distribution	ROS 2 Jazzy

1.1.2 Hardware Environment

Items	Description
RZ/V2H	• **CPU** – 4 × Arm Cortex-A55 (1.8GHz) – 2 × Arm Cortex-R8 (800MHz) – 1 × Arm Cortex-M33 (200MHz) • **DRP** – Vision/Dynamically Reconfigurable Processor • **DRP-AI** – Hardware AI Accelerator (8 dense TOPS, 80 sparse TOPS) • **Package** – R9A09G057H44GBG: 1368-pin FCBGA
Memory	LPDDR4 1600MHz (8GB) × 2
SD Card	64GB SanDisk
QSPI Flash ROM	64MB
Interfaces	• DC Jack (12–24V / 2A) • JTAG (10-pin) • MIPI CSI-2 4-Lane ×2 (22-pin / 0.5mm) • HDMI • USB3.2 Type-A ×2 • USB Micro-B (SCIF) • 10/100/1000 Base-T RJ45 • Micro SD • PCIe 3.0 Root Complex (16-pin / 0.5mm) • CAN-FD ×2 • 40-pin RasPi GPIO Header

For more details about RZ/V2H RDK's specification, visit the WS125 Robotic Development Kit Hardware Manual.

RZ/V2H RDK Board Image View:

The following image shows the top/bottom view of the RZ/V2H Robotics Development Kit (RDK) board, highlighting its main connectors and interfaces.

1.2 Quick setup guide

This quick start guide focuses on booting the board using a **microSD card**, which is the most straightforward method.

Other advanced boot methods, such as **xSPI flash**, are also supported.

The **TFTP + NFS boot** method is supported as well but is not covered in detail here.

1.2.1 Preparing the SD Card

To boot the RZ/V2H RDK board using a microSD card, you must first flash a bootable Linux image onto it.

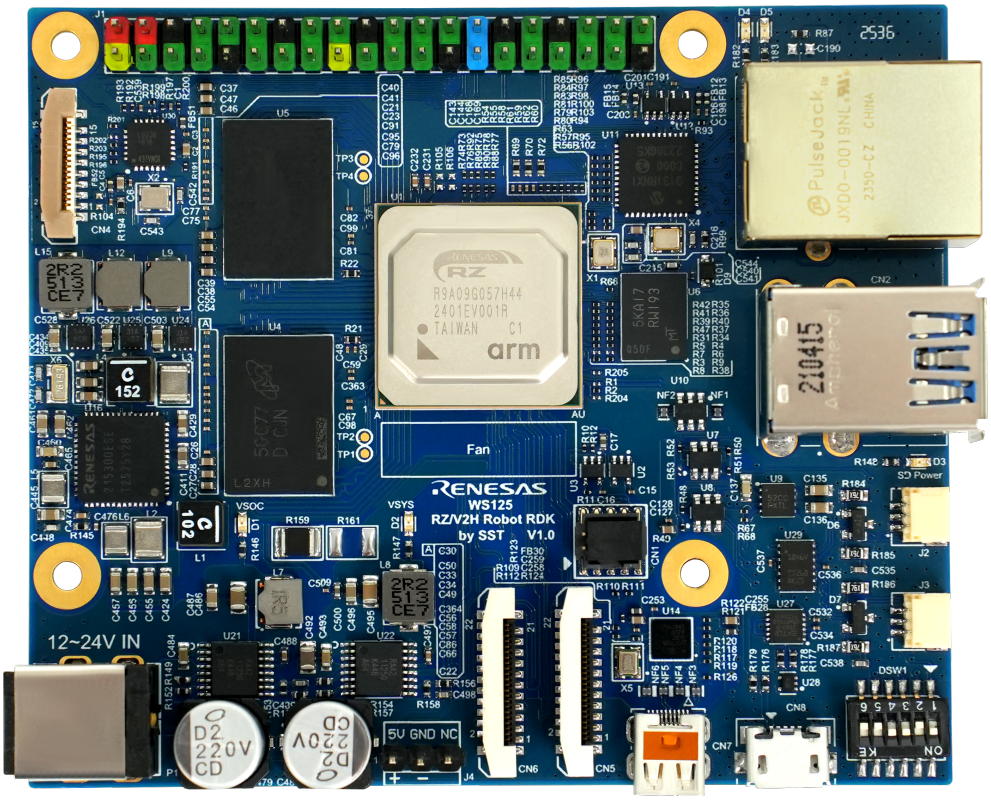

Fig. 1: RZ/V2H RDK Board Top View

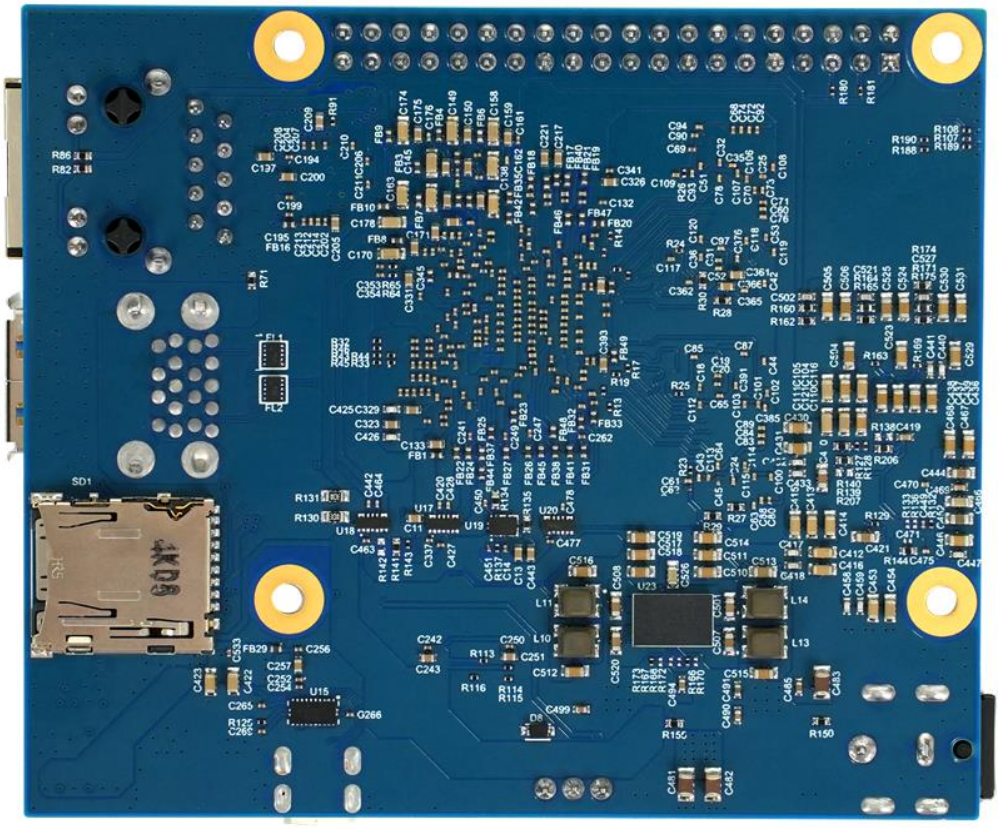

Fig. 2: RZ/V2H RDK Board Bottom View

Requirements

- **Balena Etcher:** GUI-based tool to flash image
- **microSD card:** at least 16 GB recommended
- **Provided bootable Linux images:**

File name	Target OS	Host platform support
ubuntu-lxde-image.wic.gz	Ubuntu 24.04 with GUI LXDE support	Windows / macOS / Linux
ubuntu-core-image.wic.gz	Ubuntu 24.04 headless	Windows / macOS / Linux

Flash using Balena Etcher

Balena Etcher is a user-friendly GUI tool to flash OS images to SD cards and USB drives. It provides a simple and safe method.

1. **Install Balena Etcher**

   Download and install the software from the Balena Etcher Official Website.

2. **Flashing the Image**

   - Once Etcher is open:

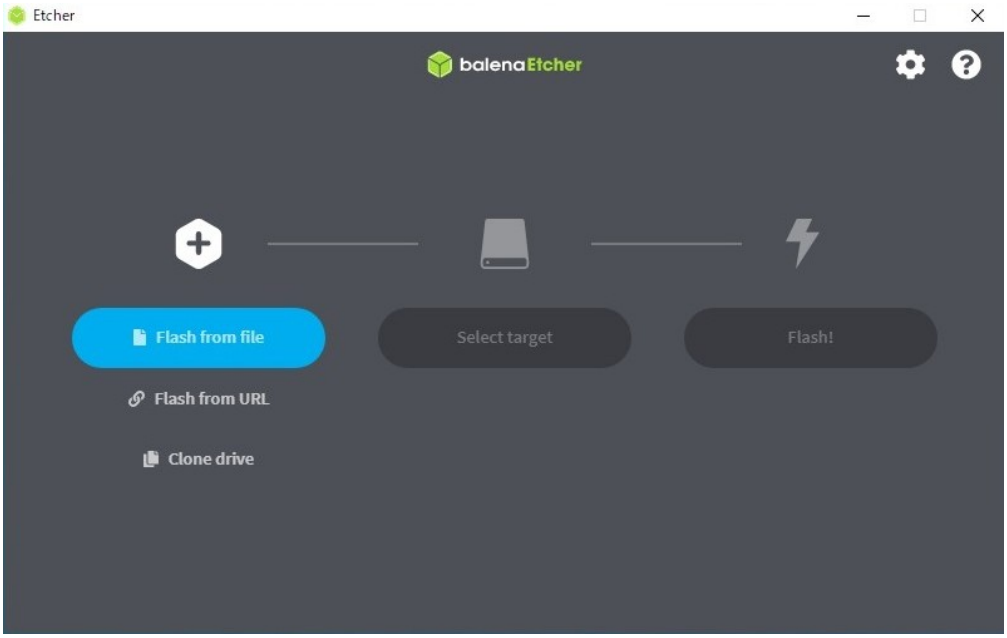

Fig. 3: Balena Etcher Application

   - **Select Image:** Click “Flash from file” and choose your image file (e.g., ubuntu-lxde-image.wic.gz)
   - **Select Target:** Insert your SD card into the host machine and choose the correct device.

> **Note**
>
> Please confirm the SD card device name carefully. Double-check to avoid overwriting your main disk.

- **Flashing:** Click “Flash” to begin. Etcher will:
  - Write the image
  - Validate the image
  - Automatically unmount the SD card
- **Finish:** Remove the SD card safely after Etcher reports successful completion.

1.2.2 Boot Mode Configuration (DIP Switch)

Before powering up the RZ/V2H RDK, make sure the board's boot mode is configured correctly using the DIP switches.

DSW1	RZ/V2H Pin	Default Setting	Operation
1	BTSEL (BOOSTSELCPU)	ON = High: 1	Select the coldboot CPU: • High: **CA55** (*default*) • Low: **CM33**
2, 3	BOOTPLLCA_1 BOOTPLLCA_0	OFF = High: 1 ON = High: 1	Input the CA55 frequency at CA55 coldboot. **BOOT_PLLCA[1:0]:** • Low:Low → 1.1 GHz • Low:High → 1.5 GHz (0.9 V) • High:Low → 1.6 GHz (0.9 V) • High:High → 1.7 GHz (0.9 V) (*default*)
4 5	MD_BOOT1 MD_BOOT0	ON = Low: 0 OFF = Low: 0	Input boot mode select signal. **MD_BOOT[1:0]:** • Low:Low → SD (*default*) • Low:High → eMMC • High:Low → xSPI • High:High → SCIF download
6	MD_BOOT3	OFF = Low: 0	Select JTAG debug mode: • Low: normal mode (*default*) • High: JTAG

Attention

Always power off the board before changing boot switches.

1.2.3 Boot Mode Support

The board supports multiple boot options, including:

Boot Source	Description	DSW1 Setting
microSD	Boot from SD card	SD mode
xSPI	Boot from xSPI flash	xSPI mode

> **Tip**
>
> The serial port is powered by the **board's power supply**, not by the **USB port** from the PC. Early boot messages might not appear automatically in the terminal (including U-Boot console and SCIF terminal). To view them, manually reset the board by connecting **JTAG QRESN (PIN10)** to **GND**, as shown below.

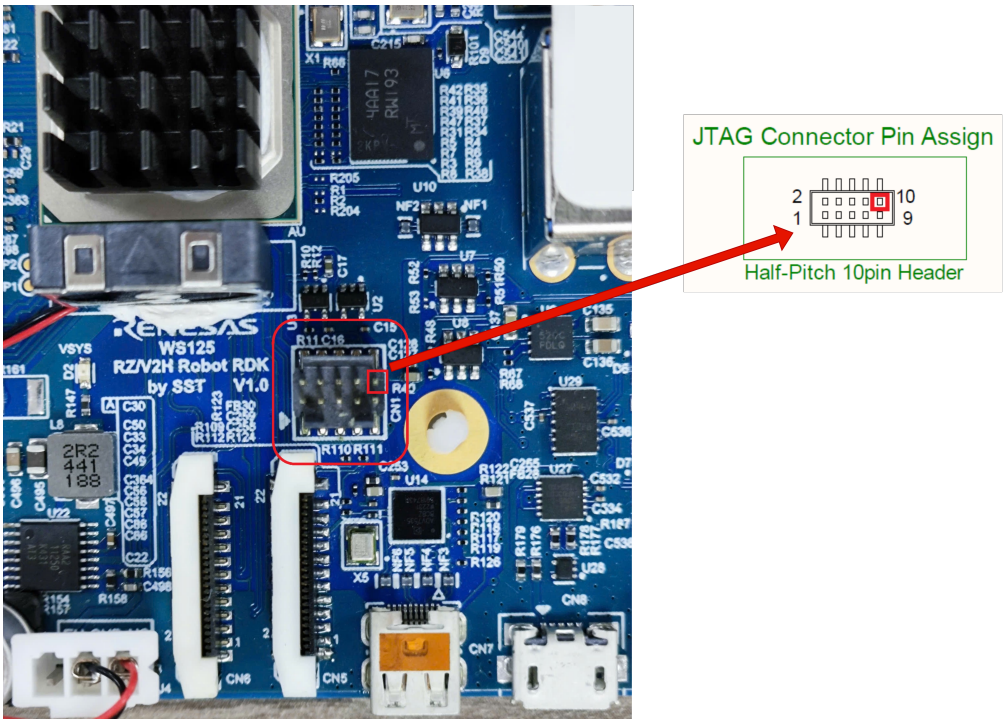

Fig. 4: JTAG Reset Pin Example

Option 1: SD Card Boot Mode

For **SD card boot mode**, the IPLs are already written to the SD card when flashing the image using Balena Etcher.

On the RZ/V2H RDK board, configure the **DSW1** switches as shown below:

After that, insert the SD card and connect the power supply (**Max 24V/5A**) to the board.

Open a terminal emulator (e.g., **Tera Term**) and connect to the **COM** port.

The COM port settings are the same as described in **Step 3** of *Write bootloaders to board*.

The board will start the boot process.

> **Tip**
>
> If there is no output from the terminal, do *the JTAG reset tip* first, then reset the U-Boot environment variables:

```
env default -a
saveenv
boot
```

If you intend to use **SD card boot mode only**, proceed to *First Time Boot Setup* to complete the setup.

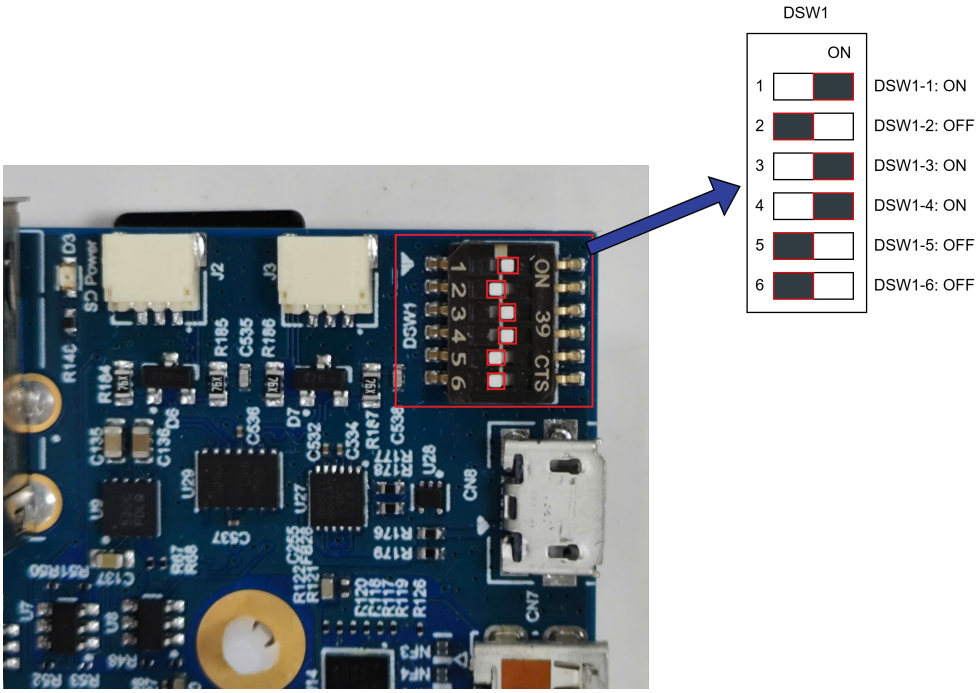

Fig. 5: DSW1 SD Card Boot Mode

Option 2: xSPI Boot Mode

Board Setup Procedure

Follow the instructions below to set up the board.

1. **Install Terminal Emulator**

> **Note**
>
> If already installed, skip this step.

- **Terminal Emulator:** Tera Term
- **Operating Environment:** Windows

2. **Install the Serial Port Driver**

> **Note**
>
> If already installed, skip this step.

- The serial communication between the Windows PC and **RZ/V2H RDK** requires: FTDI Virtual COM Port (VCP) driver

  Download and install the Windows version (*.exe*).

3. **Write Bootloaders to the Board**

Copy the bootloaders file to your Windows PC.

File Name	Description
Flash_Writer_SCIF_RZV2H_DEV_INTER-NAL_MEMORY.mot	Flash writer for RZ/V2H (used in SCIF download mode)
bl2_bp_spi-rzv2h-rdk.srec	Boot loader stage 2 binary
fip-rzv2h-rdk.srec	Firmware Image Package for RZ/V2H

- Connect the **Windows PC** and **Board** using a **Serial-to-MicroUSB** cable.
- Change the **DSW1** setting to **Boot Mode 3 (SCIF download)**.

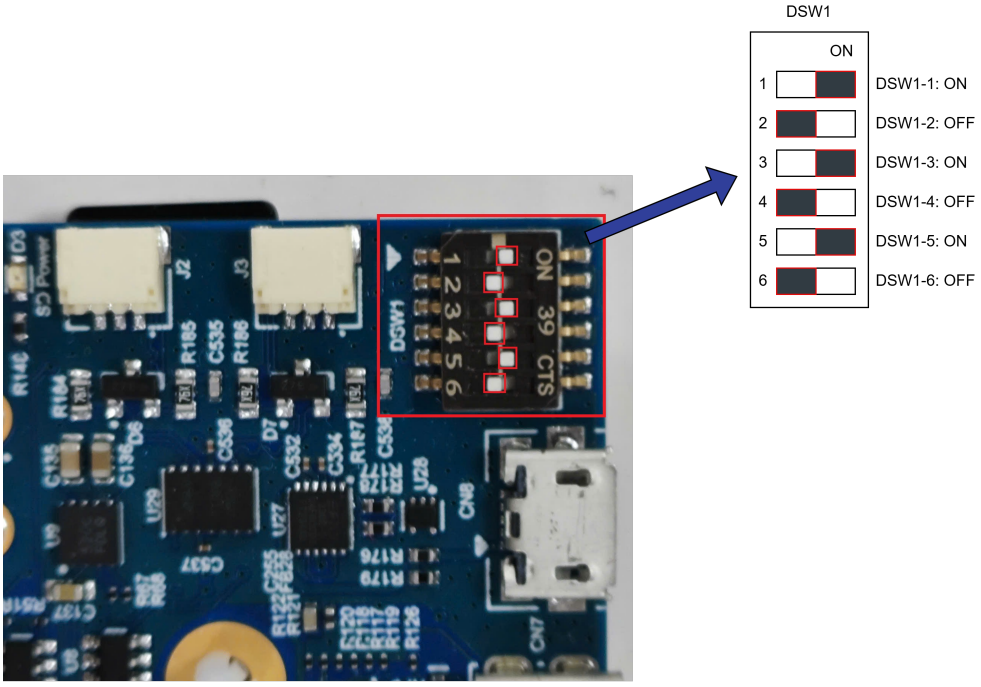

Fig. 6: DSW1 SCIF Download Mode

- Connect the power cable (**Max 24V/5A**).
- Open **Tera Term** and configure:

  Setup → Terminal:

Item	Value
New-line	Receive: Auto / Transmit: CR

  Setup → Serial Port:

Item	Value
Baud rate	115200
Data	8-bit
Parity	None
Stop	1-bit
Flow control	None
Transmit delay	0 msec/char

- Send files using “File → Send file...” and follow on-screen messages.

  (Keep the original command sequences as-is for flashing.)

4. **Setup U-Boot Configuration**
   1. Insert the microSD card to the board.
   2. Change DSW1 to **Boot mode 2 (xSPI boot)**:

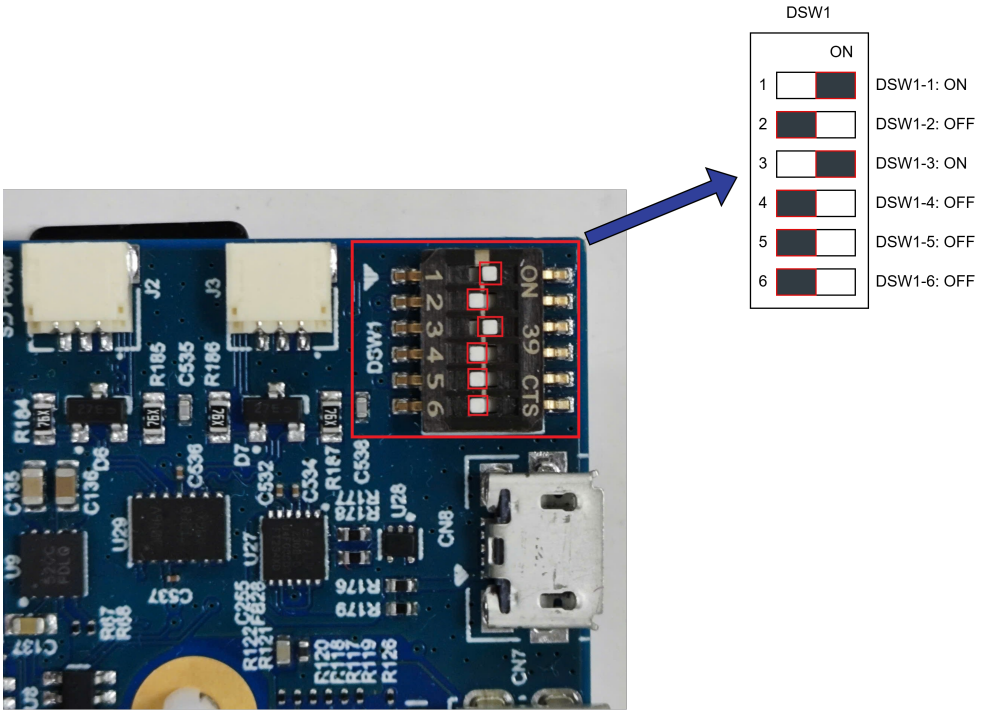

Fig. 7: DSW1 xSPI Boot Mode

   3. Connect via **USB Serial to MicroUSB** cable.
   4. Power on the board.
   5. Open the terminal emulator and connect to the **COM** port (same configuration as before).
   6. The board will boot.

> **Tip**
>
> If there is no output from the terminal, do *the JTAG reset tip* first, then reset the U-Boot environment variables:
> ```
> env default -a
> saveenv
> boot
> ```

1.2.4 First Time Boot Setup

The default user credentials for the provided Ubuntu images are as follows:

Table 1: Default Login Information

Image Type	Username	Password	Root Password
Ubuntu 24.04 (Headless)	rz	*(none)*	*(none)*
Ubuntu 24.04 (with GUI LXDE)	rz	1	*(none)*

After powering on the board **for the first time**, connect to the serial console and check the boot log to verify that Ubuntu boots successfully.

> **Note**
>
> This operation is required **only once**, immediately after flashing the root filesystem and booting the board for the first time.

Connect an Ethernet cable to the board and run:

```
# Check network
ping 8.8.8.8 -c 3
ping bing.com -c 3
```

1. Perform apt update and resize the SD card:

   ```
   sudo apt update
   sudo apt install parted
   sudo parted /dev/mmcblk0
   ```

   Inside parted terminal:

   ```
   > print
   > resizepart 2 100%
   > print
   > quit
   ```

   Resize root filesystem:

   ```
   sudo resize2fs /dev/mmcblk0p2
   ```

2. Setup **rosdep** for ROS2 package dependency management:

```
sudo rosdep init
rosdep update
```

This completes the **Quick Setup Guide** for the RZ/V2H RDK board.

1.2.5 Reference

- Advanced Boot Options (xSPI): Renesas RZ/V AI SDK Developer Guide
- Balena Etcher Official Website: https://www.balena.io/etcher

1.3 RZ/V ROS2 Demos

ROS2 packages designed for computer vision applications on Renesas RZ/V MPU platforms, specifically targeting the RZ/V2H.

The packages provide AI-accelerated vision capabilities including object detection, pose estimation, and visualization tools.

TODO: Add links to individual demo documentation pages when available.

2 System Configuration

2.1 Common system configuration

This section describes how to customize and rebuild the Linux system for the RZ/V2H RDK board, including kernel, device tree, and loadable modules.

2.1.1 Overview

The RZ/V2H RDK uses a Linux kernel **version 6.10**, which is built from the **Yocto Styhead Project**. This section provides instructions on how to customize and rebuild various components of the system by using the **Yocto Extensible SDK (eSDK)** environment.

The Yocto Project eSDK provides tools that allow developers to:

- Add new applications and libraries to an image
- Modify and rebuild the source of existing components
- Test software changes directly on the target hardware

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.

2.1.2 Prerequisites

Before proceeding, ensure that the following prerequisites are in place:

Item	Description / Link
Docker	Must be installed on the Host PC. Refer to the Docker Official Installation Guide.
Yocto eSDK for RZ/V2H RDK	Required for extensible development and rebuilding components. Obtain the installer: `poky-glibc-x86_64-renesas-core-image-weston-cor-texa55-rz-cmn-toolchain-ext-5.1.4.sh` *(TODO: Add link to Yocto eSDK package)*
Yocto SDK for RZ/V2H RDK	Standard SDK tool-chain for building applications. Obtain the installer: `poky-glibc-x86_64-renesas-core-image-weston-cor-texa55-rz-cmn-toolchain-5.1.4.sh` *(TODO: Add link to Yocto SDK package)*
RZ/V2H RDK X Compile Docker	Preconfigured cross-compilation Docker environment. Refer to the *RZ/V2H RDK X Compile Docker repository. (TODO: Add repository link)*

2.1.3 Quick set up guide

Common docker environment setup

To streamline the development process, it is recommended to use a Docker container that has been preconfigured for cross-compiling applications as well as eSDK development for the RZ/V2H RDK board.

This section provides a quick guide to setting up the Docker environment.

Obtain all files from the Prerequisites section, and following the step below to set up the Docker environment:

1. Clone the *x_compilation_docker* repository to your Host PC.

```
renesas@builder-pc:~$ git clone https://partnergitlab.renesas.
solutions/sst1/industrial/ws078/utility/x_compilation_docker.git
```

2. Copy the **Yocto SDK** installer to the Docker build context directory. Please replace the paths below with your actual file locations.

```
renesas@builder-pc:~$ cp poky-glibc-x86_64-renesas-core-
image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh ~/x_
compilation_docker/
```

> **Hint**
>
> *Why copy the Yocto SDK installer into the Docker build context directory?*
>
> This step is required to prepare the cross-compilation environment for cross-compiling the ROS2 applications targeting the RZ/V2H platform.
>
> The setup of this cross-compilation workflow will be introduced in a later section.

3. Build the Docker Container

   Navigate to the x_compilation_docker directory and build the Docker image using the following command:

```
renesas@builder-pc:~$ cd x_compilation_docker/
renesas@builder-pc:~/x_compilation_docker$ ./setup_ros2_
cross_env.sh <path_to_ros2_workspace> [name_of_docker_
container]
```

- <path_to_ros2_workspace>: Path to your ROS2 workspace, mounted inside the container.
- [name_of_docker_container]: Optional container name (default: rzv2h_ros_xbuild).

After complete this step, the Docker image (name: rzv2h_ros_xbuild:latest) and container will be created.

What does this script do?

- Copying the Yocto SDK tool-chain scripts into the build directory lets the Dockerfile install the Yocto SDK inside the image.
- The setup script usually runs docker build (to produce the image) and docker run (to create/start a container) with proper mounts, environment variables and volumes.
- Inside the container the image will have the cross compilers, sysroot and a configured environment (CMake toolchain file, sourced toolchain setup) so colcon/CMake can cross-compile ROS2 for the target.

4. Enter the Docker Container

   Use the following command to enter the Docker container:

```
renesas@builder-pc:~$ docker exec -it [name_of_docker_container] /
bin/bash
```

eSDK Setup

To get started, extract the eSDK and install the tool-chain.

Please replace the paths below with your actual file locations.

> **Tip**
>
> There is no requirement to use the Yocto eSDK within the Docker environment.
>
> However, using Docker can simplify the setup process and ensure a consistent build environment. Since Yocto SDK/eSDK installations can be complex, Docker provides a preconfigured environment that minimizes setup time and potential configuration issues.
>
> Additional, ensure that the **Yocto eSDK installer file** has been **copied into the container** so it can be executed from within that environment.
>
> ```
> renesas@builder-pc:~$ docker cp poky-glibc-x86_64-renesas-core-image-
> weston-cortexa55-rz-cmn-toolchain-ext-5.1.4.sh [name_of_docker_
> container]:~
> ```

To set up your environment:

1. Install the Yocto eSDK tool-chain.

For example, to install the tool-chain, run the following command.

```
renesas@docker-pc:~$ sh ./poky-glibc-x86_64-renesas-core-image-weston-
cortexa55-rz-cmn-toolchain-ext-5.1.4.sh
```

> **Note**
>
> You **cannot install the eSDK as the root user** because BitBake does not run with root privileges. Attempting to install the extensible SDK as root will cause the installation to fail or behave unpredictably.

If the installation is successful, the following messages will appear in the terminal output.

```
renesas@docker-pc:~$ sh ./poky-glibc-x86_64-renesas-core-image-weston-
cortexa55-rz-cmn-toolchain-ext-5.1.4.sh
Poky (Yocto Project Reference Distro) Extensible SDK installer version
5.1.4
==========================================================================
Enter target directory for SDK (default: ~/poky_sdk):
You are about to install the SDK to "/home/renesas/poky_sdk/5.1.4".
Proceed [Y/n]? Y
Extracting SDK..............done
Setting it up...
Extracting buildtools...
Preparing build system...
Parsing recipes: 100% |#########################################| Time:
0:00:52
Initialising tasks: 100% |#######################################| Time:
0:00:00
Checking sstate mirror object availability: 100% |###############| Time:
0:00:00
Loading cache: 100% |############################################| Time:
0:00:00
Initialising tasks: 100% |#######################################| Time:
0:00:00
done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to
source the
environment setup script e.g.
$ . /home/renesas/poky_sdk/5.1.4/environment-setup-cortexa55-poky-linux
```

2. Set up cross-compile environment.

The following command assumes that you installed the SDK in *~/poky_sdk/5.1.4*

```
renesas@docker-pc:~$ source ~/poky_sdk/5.1.4/environment-setup-
cortexa55-poky-linux
SDK environment now set up; additionally you may now run devtool to
perform development tasks.
Run devtool --help for further details.
```

> **Note**
>
> The user needs to run the above command once for each shell session.

2.1.4 Using devtool in the Yocto eSDK

The devtool utility is part of the Yocto Project's Extensible SDK (eSDK). It provides an isolated workspace for modifying, testing, and maintaining recipes without altering upstream metadata.

This section focuses on the core commands used in day-to-day development, especially for Linux kernel, device trees, and driver modifications on Renesas RZ Common System.

1. devtool modify

   The devtool modify command checks out the source code for a recipe into a local workspace, allowing changes without touching upstream layers. This is usually the first step in customizing the kernel, device trees, or drivers.

   Example: To modify the Linux kernel recipe:

   ```
   renesas@docker-pc:~$ devtool modify linux-yocto
   ```

   This command sets up a workspace where you can make changes to the Linux kernel source code.

2. devtool build

   After making changes to the source code, use this command to build the modified recipe.

   Example: To build the modified Linux kernel:

   ```
   renesas@docker-pc:~$ devtool build linux-yocto
   ```

   This command compiles the changes and prepares them for deployment.

3. devtool reset

   If you want to discard your changes and revert to the original recipe, use this command.

   Example: To reset the Linux kernel recipe:

   ```
   renesas@docker-pc:~$ devtool reset linux-yocto
   ```

   This command removes your modifications and restores the original source code.

4. devtool build-image

   The devtool build-image command builds a complete target image that includes all recipes currently under modification.

   This is useful to verify integration of changes into a full root filesystem, not just individual components.

   Example: To build a new renesas-core-image-weston:

   ```
   renesas@docker-pc:~$ devtool build-image renesas-core-image-weston
   ```

   This command generates an updated image that can be deployed to the target hardware.

Known Issue

When working with the **Yocto eSDK**, you might encounter the following warning:

```
WARNING: You are using a local hash equivalence server but have
configured an sstate mirror.
This will likely mean no sstate will match from the mirror.
You may wish to disable the hash equivalence use (BB_HASHSERVE), or use
a hash equivalence server alongside the sstate mirror.

The ros2-control:do_package_qa sig is computed to be
```

(continues on next page)

(continued from previous page)

```
ea8f7e910d566912b932cbe602d93b93502064e293d1f4f1f569a67ee49f1c72,but the
sig is locked to
fd89fc1eb9961fd4ccddf16ea2ca1b73b5480ce1670ebb07a8075603bb645bc8 in
SIGGEN_LOCKEDSIGS_t-cortexa55
```

Note

This warning can be **safely ignored**. It does not affect the build process or output artifacts when using the Yocto eSDK.

2.1.5 Custom Linux Kernel and Device Tree

This section describes how to customize and rebuild the Linux kernel or device tree blob for the RZ/V2H RDK board using the Yocto eSDK and devtool.

Note

Before proceeding, ensure that you have set up the eSDK environment as described in the *eSDK Setup* section above.

1. modify the Linux kernel recipe:

   Setup the eSDK environment in the current terminal session:

   ```
   renesas@docker-pc:~$ source ~/poky_sdk/environment-setup-cortexa55-
   poky-linux
   ```

   Use the devtool modify command to check out the linux-yocto recipe for modification:

   ```
   renesas@docker-pc:~$ devtool modify linux-yocto
   ```

When executed, this:

- Creates a workspace copy of the kernel source under: *~/poky_sdk/workspace/sources/linux-yocto/*
- Generates a .bbappend for linux-yocto in the workspace's recipe area.
- Prepares the environment for kernel modifications.

2. Applying Patches for linux-yocto

Unlike most recipes, linux-yocto in this BSP is implemented as an out-of-tree kernel.

- Kernel modifications are stored as patches inside workspace/sources/linux-yocto/.kernel-meta/
- The kernel default configuration (renesas_defconfig) is also managed out-of-tree.
- To ensure the workspace matches the recipe, patches must be applied after running devtool modify.

Procedure, applying patches to the kernel linux-yocto source

```
renesas@docker-pc:~$ cd ~/poky_sdk/workspace/sources/linux-yocto/
renesas@docker-pc:~/poky_sdk/workspace/sources/linux-yocto$ git am $(cat
patch.queue)
```

This applies the patch series defined in .kernel-meta/patches/ to the kernel workspace.

After applying the patches, developers may perform the following steps:

- Provide kernel configuration fragments (`.cfg` files) to adjust features or enable additional built-in kernel drivers.
- Modify the kernel source code as needed for custom functionality.
- Continue with the `devtool build linux-yocto` command to compile the kernel with the applied modifications.

3. Adding Kernel Configuration Fragments

> **Tip**
>
> If you're unsure whether the config has been added to the kernel configuration,
>
> you can use *zcat /proc/config.gz | grep <CONFIG_NAME>* in the target machine to check whether it is enabled.

There are two possible methods to add kernel configuration for linux-yocto:

- Edit the out-of-tree defconfig directly: *~/poky_sdk/layers/meta-renesas/recipes-kernel/linux/rz-cmn/common/kernel-common.cfg*
- **Adding a configuration fragment (.cfg) file**
  - Create the append structure

```
renesas@docker-pc:~$ mkdir -p ~/poky_sdk/workspace/recipes-
kernel/linux/linux-yocto/files/
```

  - Create a configuration fragment file, e.g., *myconfig.cfg*, inside the *files/* directory. For example:

```
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_FTDI_SIO=y
```

> **Note**
>
> Please include all dependency configs as well.

  - Edit the *linux-yocto_%.bbappend* file in the workspace's recipe area to include the new configuration fragment:

```
renesas@docker-pc:~$ vim ~/poky_sdk/workspace/appends/linux-
yocto/linux-yocto_6.10.bbappend
```

  - Add the following line to the bbappend file:

```
SRC_URI:append = " file://myconfig.cfg"
```

4. Modify the DTS file:

```
renesas@docker-pc:~$ cd ~/poky_sdk/workspace/sources/linux-yocto/
renesas@docker-pc:~$ vi arch/arm64/boot/dts/renesas/r9a09g057h4-rdk-
ver1.dts
```

   Make the necessary changes to the Device Tree Source file (`.dts`) according to your requirements.

5. Build the Modified Kernel

After making changes in the workspace, use devtool build to compile the recipe.

```
renesas@docker-pc:~$ devtool build -c clean linux-yocto
renesas@docker-pc:~$ export DISTRO=ubuntu-tiny
renesas@docker-pc:~$ devtool build linux-yocto
```

Attention

Before building, always **clean the previous build artifacts** to avoid conflicts.

```
renesas@docker-pc:~$ devtool build -c clean linux-yocto
```

Always set the `DISTRO` variable to `ubuntu-tiny` to ensure compatibility with the Ubuntu-based root filesystem.

Otherwise, the generated artifacts may **not be compatible** with the Ubuntu image.

6. Collect the Built Kernel Artifacts

Once the build is complete, collect the built kernel artifacts for deployment to the target hardware.

When building `linux-yocto`, the generated artifacts can be collected from the following locations: *~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image*

Please copy those files to the appropriate boot media (e.g., SD card) as per your deployment requirements.

Tip

On the target device, make sure to run the following command to update the module dependencies after deployment:

```
$ sudo depmod -a
```

2.2 Ubuntu System with RZ/V2H RDK

This section provides usage information about the interfaces available on the RZ/V2H RDK when running the Ubuntu system.

For more details about specification of each interface, refer to the RZ/V2H Group User's Manual: Hardware.

2.2.1 Overview

The RZ/V2H RDK supports multiple peripheral interfaces that allow users to connect and control external devices for various robotic and industrial applications. These interfaces include:

2.2.2 Main Interfaces

The main interfaces available on the RZ/V2H Robotic Development Kit are listed below.

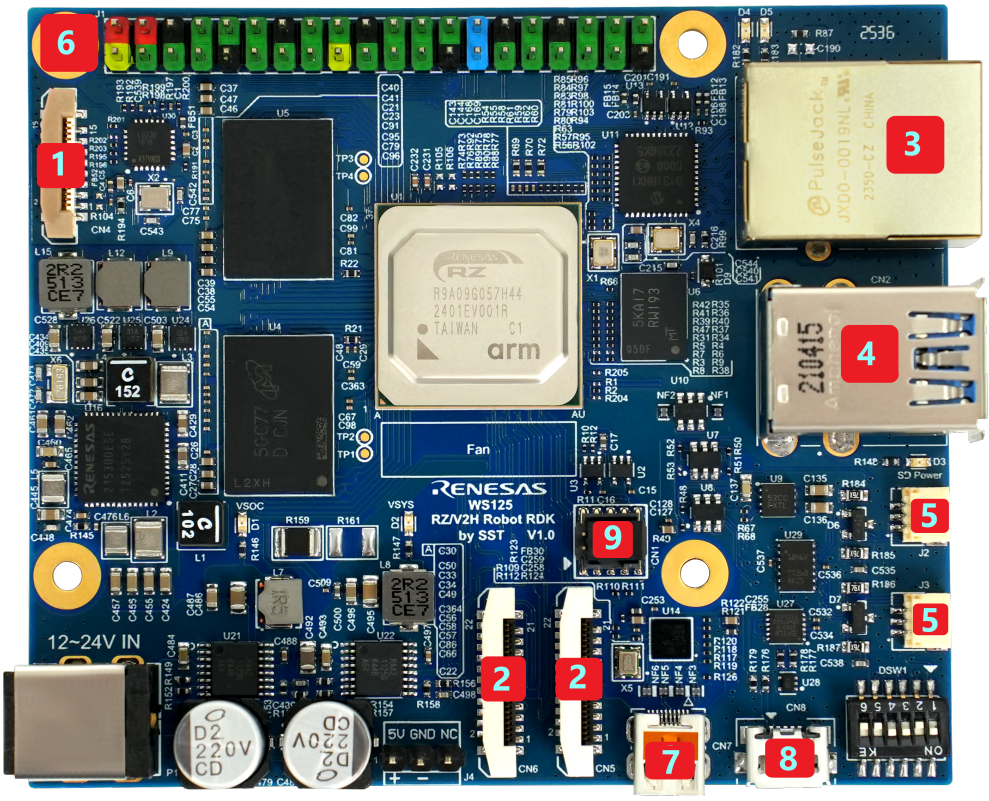

Fig. 1: RZ/V2H Robotic Development Kit Interfaces

Table 1: High-Speed Interfaces

No	Interface Name	Description
1	PCIe 3.0 16-pin connector	High-speed peripheral expansion interface.
2	MIPI-CSI 22-pin connector ×2	Dual camera input interface for image sensors.
3	1000M RJ45	Gigabit Ethernet LAN port for network connectivity.
4	USB 3.0 Type A ×2	USB host ports for external devices such as mouse, keyboard, or USB camera.

Table 2: Communication Interfaces

No	Interface Name	Description
5	CAN-FD ×2	Controller Area Network Flexible Data-Rate communication ports.
6	RasPi GPIO 40-pin Header (I2C, SPI, UART, GPIO, PCM)	General purpose I/O interface compatible with Raspberry Pi pin layout.

Table 3: Other Interfaces

No	Interface Name	Description
7	Mini-HDMI	Video output interface for external display.
8	USB-UART	Serial console interface for debugging.
9	JTAG 10-pin	JTAG interface for debugging and programming.

Each subsection provides details on how to identify, configure, and access these interfaces within the Ubuntu environment.

High-Speed Interface

The RZ/V2H Robotic Development Kit is equipped with several high-speed interfaces that enable users to connect a variety of peripherals and expansion modules.

This sections describes the High-Speed Interface unit of this Kit.

1. PCIe 3.0 16-pin connector

The PCIe 3.0 interface on the RZ/V2H RDK allows for high-speed data transfer and connectivity with compatible PCIe devices.

Important

The PCIe interface is not available in this release.

2. MIPI-CSI 22-pin connector ×2

The RZ/V2H RDK features dual MIPI-CSI connectors that support camera input for applications requiring image capture and processing.

See also

For information about available partner Camera module list on RZ/V2H, see [RZ/V2H] AVAILABLE PARTNER CAMERA MODULE LIST.

The default RZ/V2H RDK device tree supports the OV5645 camera module connected to the MIPI-CSI interface.

Hint

It is required to change the dts file to use the MIPI-CSI interface with other camera module. Refer to: *Modify the DTS file* section in the Build Kernel chapter for more details about customize the dts file.

Set up the MIPI-CSI interface

Before using the MIPI-CSI interface, we have to configure the property of the camera first.

For example, to use the OV5645 camera module, create and run the following script in the terminal:

```
#!/bin/bash

cru=$(cat /sys/class/video4linux/v4l-subdev*/name | grep "cru-ip")
csi2=$(cat /sys/class/video4linux/v4l-subdev*/name | grep "csi2")
ov=$(cat /sys/class/video4linux/v4l-subdev*/name | grep "ov")
default_camera=$(echo "$ov" | awk '{print $1}')

declare -A camera_default_resolution
camera_default_resolution["ov5640"]="1280x720"
camera_default_resolution["ov5645"]="1280x960"

declare -A camera_valid_resolutions
camera_valid_resolutions["ov5640"]="720x480 720x576 1024x768 1280x720
1920x1080 2592x1944"
camera_valid_resolutions["ov5645"]="1280x960 1920x1080 2592x1944"

# Usage information function
function print_usage {
    echo "Usage: $0 <resolution>"
    echo "Detected camera: $default_camera"
    echo "Available resolutions for $default_camera: ${camera_valid_
resolutions[$default_camera]}"
    echo ""
    echo "Example: $0 1920x1080"
    echo "If no resolution is specified, $default_camera will used the
default resolution '${camera_default_resolution[$default_camera]}'."
}

# Check if help is requested
if [[ "$1" == "-h" ]] || [[ "$1" == "--help" ]]; then
    print_usage
    exit 0
fi
```

(continues on next page)

(continued from previous page)

```
# Check for no input
if [ -z "$1" ]; then
    echo "Detected camera: $default_camera"
    echo "No resolution specified. Using default resolution: ${camera_
default_resolution[$default_camera]}"
    ov564x_res="${camera_default_resolution[$default_camera]}"
else
    ov564x_res="$1"
    valid_resolutions=(${camera_valid_resolutions[$default_camera]})
    # Check if the given resolution is valid
    if [[ ! " ${valid_resolutions[@]} " =~ " ${ov564x_res} " ]]; then
        echo "Invalid resolution: $ov564x_res for camera $default_camera
"
        ov564x_res="${camera_default_resolution[$default_camera]}"
        echo "Input resolution is not available. Using default
resolution: ${camera_default_resolution[$default_camera]}"
    fi
fi

if [ -z "$cru" ]
then
    echo "No CRU video device founds"
else
    media-ctl -d /dev/media0 -r
    if [ -z "$csi2" ]
    then
        echo "No MIPI CSI2 sub video device founds"
    else
        media-ctl -d /dev/media0 -V "'$csi2':0 [fmt:UYVY8_1X16/$ov564x_
res field:none]"
        media-ctl -d /dev/media0 -V "'$csi2':1 [fmt:UYVY8_1X16/$ov564x_
res field:none]"
        media-ctl -d /dev/media0 -V "'$ov':0 [fmt:UYVY8_1X16/$ov564x_res
field:none]"
        media-ctl -d /dev/media0 -V "'$cru':0 [fmt:UYVY8_1X16/$ov564x_
res field:none]"
        media-ctl -d /dev/media0 -V "'$cru':1 [fmt:UYVY8_1X16/$ov564x_
res field:none]"

        width=${ov564x_res%x*}
        height=${ov564x_res#*x}
        v4l2-ctl -d /dev/video0 --set-fmt-video=width=${width},height=$
{height},pixelformat=UYVY
        echo "Link CRU/CSI2 to $ov with format UYVY8_1X16 and resolution
${ov564x_res}"
    fi
fi
```

This script detects the connected camera module and sets the desired resolution.

For other camera modules, please modify the script accordingly.

Usage example with v4l2-ctl

First, install the v4l-utils package if it is not already installed:

```
$ sudo apt install v4l-utils
```

List all connected cameras:

```
$ v4l2-ctl --list-devices
```

List all supported formats for selected camera /dev/video0:

```
$ v4l2-ctl -d /dev/video0 --list-formats-ext
```

TODO: Update the command about capturing the image from the camera module.

3. 1000M RJ45 - Gigabit Ethernet Port

The Gigabit Ethernet (1000M RJ45) port on the RZ/V2H RDK provides high-speed network connectivity for data communication and internet access.

Connect the network cable to (3) before using the Ethernet interface.

The Ubuntu installer configures the system to obtain its network settings via DHCP.

After connecting the Ethernet cable, use the following command to confirm the network configuration:

To list all network interfaces and their IP addresses:

```
$ ifconfig
```

To test network connectivity reach an external server, use the ping command:

```
$ ping -c 4 bing.com
$ ping -c 4 8.8.8.8
```

Set a static IP address

In Ubuntu, the network is configured with Netplan, if you need to set a static IP address for the Ethernet interface (example: use a static IP address 192.168.0.100), follow these steps:

- Open the network configuration file with vim:

```
$ sudo vi /etc/netplan/01-netcfg.yaml
```

- Modify the file to set a static IP address. For example:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
version: 2
renderer: NetworkManager
ethernets:
    end0:
    dhcp4: no
    dhcp6: no
    addresses: [169.254.43.99/24]
```

(continues on next page)

(continued from previous page)

```
    routes:
      - to: default
        via: 169.254.43.86
    nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

- Apply the changes with the following command:

```
$ sudo netplan apply
```

Set DHCP

In case you want to revert back to DHCP configuration, modify the `/etc/netplan/01-netcfg.yaml` file as follows:

```
# This file describes the network interfaces available on your
system
# For more information, see netplan(5).
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    end0:
      dhcp4: yes
      dhcp6: yes
```

- Apply the changes with the following command:

```
$ sudo netplan apply
```

4. USB 3.0 Type A ×2

The RZ/V2H RDK includes two USB 3.0 Type-A ports that support high-speed data transfer for connecting various USB peripherals, such as external storage devices, cameras, and input devices.

To use these devices, simply connect them to the USB 3.0 Type-A ports.

Verify USB 3.0 functionality

To verify that the USB 3.0 ports are functioning correctly, you can use the following command to list USB devices and check their connection speed:

```
$ lsusb -t
```

Example output:

```
root@localhost:~# lsusb -t
/:  Bus 001.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/
1p, 480M
/:  Bus 002.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/
1p, 20000M/x2
/:  Bus 003.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/
```

(continues on next page)

(continued from previous page)

```
1p, 480M
/:  Bus 004.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/
1p, 20000M/x2
root@localhost:~#
```

Communication Interfaces

This section provides usage examples of the communication interfaces available on this kit.

5. CAN-FD ×2

The RZ/V2H RDK is equipped with two CAN-FD (Controller Area Network Flexible Data-Rate) ports that enable high-speed communication for automotive and industrial applications.

> **Tip**
>
> To use the CAN-FD interfaces, ensure that you have the required CAN connectors.
>
> We recommend using the: **Bus Cable, 1 Female Connector, 3 Contacts, 1.0 mm Pitch, 10 cm Length**.
>
> The RZ/V2H RDK is equipped with an onboard CAN transceiver (**TCAN1046VDMTRQ1**) and an integrated **120 Ω termination resistor**, eliminating the need for any external CAN transceiver circuitry.

Connect the CAN-FD ports to your CAN network using appropriate connectors and cables. CAN-H and CAN-L lines should be connected accordingly.

Usage examples:

Verify that the CAN interfaces are recognized:

```
$ ip link show | grep can
```

Example output:

```
5: can0: <NOARP,ECHO> mtu 72 qdisc noop state DOWN mode DEFAULT group
default qlen 10 link/can
6: can1: <NOARP,ECHO> mtu 72 qdisc noop state DOWN mode DEFAULT group
default qlen 10 link/can
```

Bring up the CAN interface (e.g., 500 kbps nominal, 2 Mbps data):

```
$ sudo ip link set can0 down
$ sudo ip link set can0 type can bitrate 500000 dbitrate 2000000 fd on
$ sudo ip link set can0 up
```

Check the interface status:

```
$ ip -details link show can0
```

Send and Receive CAN Messages

You can use the **can-utils** package for testing CAN-FD communication.

1. Install can-utils (if not already installed):

```
$ sudo apt install can-utils
```

2. On one terminal, listen for incoming CAN-FD frames:

```
$ candump can0
```

3. On another terminal, send a test frame:

```
$ cansend can0 123##0112233445566778 8
```

6. RasPi GPIO 40-pin Header

The Raspberry Pi GPIO 40-pin header on the RZ/V2H RDK provides a versatile interface for connecting various peripherals and expansion boards compatible with the Raspberry Pi pin layout. This header includes multiple communication protocols such as I2C, SPI, UART, GPIO, and PCM.

Support the following communication protocols:

- I2C (Inter-Integrated Circuit)
- SPI (Serial Peripheral Interface)
- UART (Universal Asynchronous Receiver/Transmitter)
- GPIO (General Purpose Input/Output)
- PCM (Pulse Code Modulation)

Pin Out Diagram

I2C (Inter-Integrated Circuit)

The I2C interface allows communication with multiple slave devices using just two wires: SDA (data line) and SCL (clock line).

It is commonly used for connecting sensors, displays, and other peripherals.

On the RZ/V2H RDK, the I2C pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 4: I2C7 Interface Pins

Pin Name	Function	Description
P76	SDA7	I2C7 data line - Serial Data (connected with 2.2K pull-up resistor).
P77	SCL7	I2C7 clock line - Serial Clock (connected with 2.2K pull-up resistor).

Usage example with i2c-tools

First, install the i2c-tools package if it is not already installed:

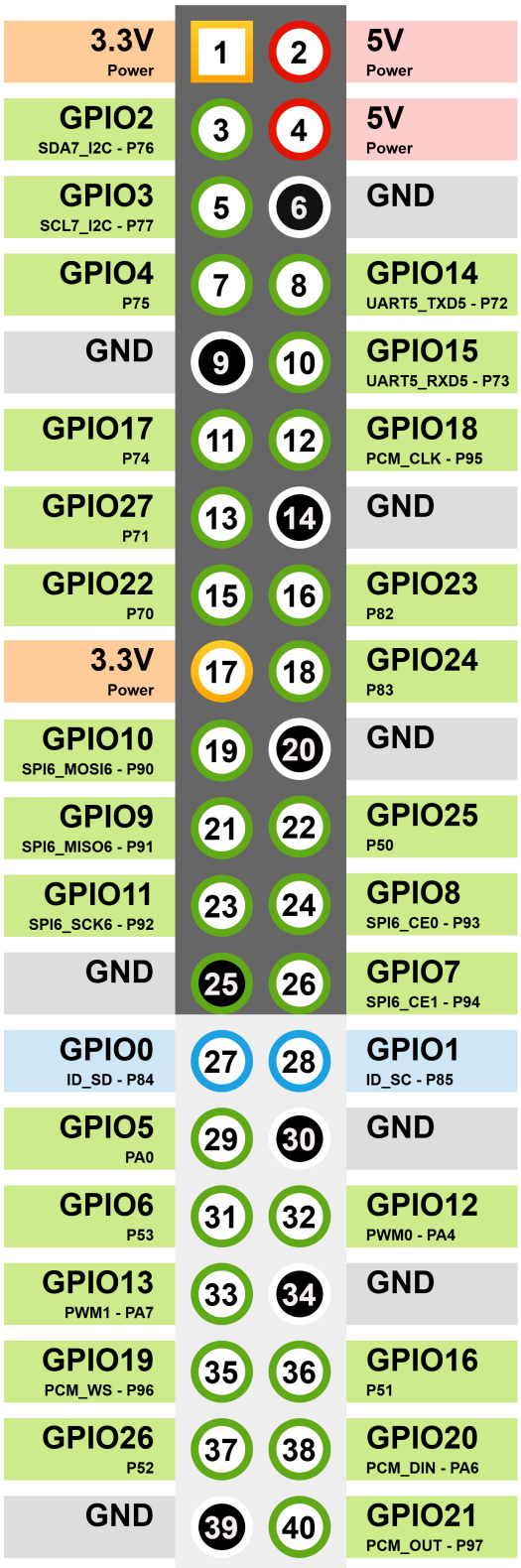

Fig. 2: RZ/V2H RDK Raspberry Pi GPIO 40-pin Header Pin Out

```
$ sudo apt install i2c-tools
```

List all I2C buses available on the system:

```
$ i2cdetect -l
```

Example output:

```
i2c-3   i2c             Renesas RIIC adapter                    I2C
adapter
i2c-4   i2c             Renesas RIIC adapter                    I2C
adapter
i2c-8   i2c             Renesas RIIC adapter                    I2C
adapter
i2c-9   i2c             Renesas RSCI I2C adapter                I2C
adapter
i2c-10  i2c             Renesas RSCI I2C adapter                I2C
adapter
```

Find the correct bus number for I2C7:

```
root@localhost:~# ls -l /sys/class/i2c-dev/
total 0
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-10 -> ../../devices/platform/
soc/12802800.i2c/i2c-10/i2c-dev/i2c-10
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-3 -> ../../devices/platform/
soc/14401000.i2c/i2c-3/i2c-dev/i2c-3
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-4 -> ../../devices/platform/
soc/14401400.i2c/i2c-4/i2c-dev/i2c-4
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-8 -> ../../devices/platform/
soc/11c01000.i2c/i2c-8/i2c-dev/i2c-8
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-9 -> ../../devices/platform/
soc/12802400.i2c/i2c-9/i2c-dev/i2c-9
root@localhost:~#
```

In this example, I2C7 corresponds to bus number 10.

Hint

How to identify the correct I2C bus number for I2C7?

You can identify the correct I2C bus number by checking the device tree source (DTS) file for the RZ/V2H RDK or by referring to the system documentation.

In this case, the device tree of RZ/V2H RDK defines the IC27 interface as **12802800.i2c**, which is mapped to **I²C bus number 10**.

Scan for I2C devices on bus 10:

```
$ sudo i2cdetect -y 10
```

SPI (Serial Peripheral Interface)

The SPI interface enables high-speed communication with peripheral devices using a master-slave architecture. It uses separate lines for data in, data out, clock, and chip select.

On the RZ/V2H RDK, the SPI pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 5: SPI6 Interface Pins

Pin Name	Function	Description
P93	SS0	Slave Select 0 signal for SPI6.
P94	SS1	Slave Select 1 signal for SPI6.
P90	MOSI6	Master Out Slave In (data output from master).
P91	MISO6	Master In Slave Out (data input to master).
P92	SCK6	Serial Clock signal for SPI6.

Usage example

List SPI devices:

```
$ ls /dev/spidev*
/dev/spidev1.0
```

Test SPI communication (please connect an appropriate SPI device to test):

```
$ spi-config -d /dev/spidev1.0 -q
/dev/spidev1.0: mode=0, lsb=0, bits=8, speed=2000000, spiready=0
$ echo -n -e "1234567890" | spi-pipe -d /dev/spidev1.0 -s 10000000 |
hexdump
0000000 3231 3433 3635 3837 3039
000000a
```

UART (Universal Asynchronous Receiver/Transmitter)

The UART interface provides serial communication capabilities, allowing data exchange between the RZ/V2H RDK and other devices such as micro-controllers, GPS modules, or serial consoles.

On the RZ/V2H RDK, the UART pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 6: UART5 Interface Pins

Pin Name	Function	Description
P72	TXD5	UART5 transmit data (TX) signal.
P73	RXD5	UART5 receive data (RX) signal.

Usage example with minicom

First, install the minicom package if it is not already installed:

```
$ sudo apt install minicom
```

List available serial ports:

```
$ ls /dev/ttySC*
```

Example output:

```
/dev/ttySC0  /dev/ttySC1
```

Open a serial connection using minicom (replace /dev/ttySC1 with the appropriate device):

Tip

How to identify the correct UART port for UART5?

You can identify the correct UART port by checking the device tree source (DTS) file for the RZ/V2H RDK or by referring to the system documentation.

```
root@localhost:~# ls -l /sys/class/tty/ | grep ttySC
lrwxrwxrwx 1 root root 0 Jul 12 01:53 ttySC0 -> ../../devices/
platform/soc/11c01400.serial/11c01400.serial:0/11c01400.
serial:0.0/tty/ttySC0
lrwxrwxrwx 1 root root 0 Jul 12 01:53 ttySC1 -> ../../devices/
platform/soc/12802000.serial/12802000.serial:0/12802000.
serial:0.0/tty/ttySC1
root@localhost:~#
```

In this case, the device tree of RZ/V2H RDK defines the UART5 interface as **12802000.serial**, which is mapped to **/dev/ttySC1**.

```
$ sudo minicom -D /dev/ttySC1 -b 115200
```

GPIO (General Purpose Input/Output)

The GPIO pins allow for digital input and output operations, enabling interaction with various sensors, actuators, and other electronic components.

Please refer to the RZ/V2H RDK GPIO pinout documentation for detailed information on each GPIO pin's capabilities and functions.

Usage example with gpiod:

First, install the gpiod package if it is not already installed:

```
$ sudo apt install gpiod
```

List available GPIO chips:

```
$ gpiodetect
```

List lines for a specific GPIO chip (e.g., gpiochip1):

```
$ gpioinfo gpiochip1
```

Set a GPIO line as output and change its value:

```
# Set GPIO line 92 high - turn on LED D4
$ gpioset --mode=signal gpiochip1 92=1
```

Read the value of a GPIO line:

```
$ gpioget gpiochip1 92
```

PCM (Pulse Code Modulation)

The PCM interface is used for audio data transmission, allowing the RZ/V2H RDK to connect with audio codecs and other audio peripherals.

On the RZ/V2H RDK, the PCM pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 7: PCM Audio Interface Pins

Pin Name	Function	Description
PA6	PCM_DIN	PCM data input (from external audio device to RZ/V2H).
P97	PCM_OUT	PCM data output (from RZ/V2H to external audio device).
P96	PCM_WS	PCM word select (frame sync) signal.
P95	PCM_CLK	PCM bit clock signal.

TODO: Add usage examples for PCM interface.

Other interfaces

7. Mini-HDMI

The RZ/V2H Robotic Development Kit features a Mini-HDMI port for video output to an external display. To use this interface, connect a Mini-HDMI cable from the board to a compatible monitor.

TODO: Add more information, with Ubuntu Core, there is no display output by default.

8. USB-UART

The RZ/V2H RDK includes a USB-UART interface for serial communication and debugging purposes. This interface allows you to connect the board to a host computer via a USB cable and access the serial console.

We recommend using a terminal emulator such as *minicom* or *Tera Term* to connect to the USB-UART interface.

The configuration settings for the serial connection are as follows:

- Baud Rate: 115200
- Data Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

9. JTAG 10-pin

The RZ/V2H RDK provides a JTAG 10-pin interface for debugging and programming CM33 and two CR8 cores.

This interface is essential for low-level debugging and development tasks in Multi-Core applications.

For more information on using the JTAG interface, please refer to the RZ/V2H RDK *Multi-OS Development Section*.

TODO: Add link to Multi-OS Development Section