

RZ/V2H Robotic Development Kit

User's Manual: Software

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (<http://www.renesas.com>).

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.

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 implants; 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

Trademarks

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

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/.

Contents

1. Overview	5
1.1. Software Environment	5
1.2. Hardware Environment	5
2. Quick setup guide	8
2.1. Preparing the SD Card	8
2.1.1. Requirements	8
2.1.2. Flash using Balena Etcher	8
2.2. Boot Mode Configuration (DIP Switch)	9
2.3. Boot Mode Support	10
2.3.1. Option 1: SD Card Boot Mode	11
2.3.2. Option 2: xSPI Boot Mode	12
2.4. First Time Boot Setup	17
2.5. Reference	20
3. Common system configuration	21
3.1. Overview	21
3.2. Prerequisites	21
3.3. Quick set up guide	22
3.3.1. Common docker environment setup	22
3.3.2. eSDK Setup	24
3.4. Using devtool in the Yocto eSDK	25
3.5. Custom Linux Kernel and Device Tree	27
3.6. Custom Linux Real-time Kernel and Device Tree	31
4. Ubuntu System with RZ/V2H RDK	32
4.1. Overview	33
4.2. Main Interfaces	33
4.3. High-Speed Interface	34
4.3.1. PCIe 3.0 16-pin connector	34
4.3.2. MIPI-CSI 22-pin connector x2	36
4.3.3. 1000M RJ45 - Gigabit Ethernet Port	37
4.3.4. USB 3.0 Type A x2	39
4.4. Communication Interfaces	42
4.4.1. CAN-FD x2	42
4.4.2. RasPi GPIO 40-pin Header	44
4.5. Other interfaces	50
4.5.1. Micro-HDMI	50

4.5.2. USB-UART	51
4.5.3. JTAG 10-pin	51
5. RZ/V2H Advance Features	52
6. DRP-AI	52
6.1. Overview	52
6.2. Concepts	52
6.2.1. What is Dynamically Reconfigurable Processor (DRP)?	52
6.2.2. What is DRP-AI?	53
6.3. BYOM AI model support	55
6.3.1. Getting Started	55
6.3.2. Install the DRP-AI TVM extension package	55
6.3.3. BYOM Development Flow	55
6.3.4. Training data collection and Model training	56
6.3.5. Exchange AI model	57
6.3.6. Deployment	57
6.4. The rzv_model package	57
6.4.1. Base Framework	57
6.4.2. Implemented Models	57
6.4.3. Package structure	58
6.4.4. Architecture	59
6.4.5. How to use the rzv_model package	59
6.5. DRP-AI with rzv_model tutorials	62
6.5.1. YOLOX Object Detection Tutorial	63
6.5.2. YOLOv8 Object Detection Tutorial	64
6.5.3. MediaPipe Hand Landmark Tutorial	66
6.5.4. RTMPose Hand Landmark Tutorial	69
6.5.5. HRNetV2 Hand Landmark Tutorial	71
6.6. Renesas AI Applications	73
6.6.1. Prerequisites	73
6.6.2. Available AI Applications	74
6.6.3. Building Renesas AI Applications	74
7. Video Codec Library	83
8. OpenCV Accelerator	83
8.1. Overview	83
8.2. How to use OpenCVA	83
9. RZ/V Multi-OS	85
9.1. Overview	85

9.2. Useful References	86
9.3. Requirement	86
9.4. Note for integration	87
9.5. How to create the new project for RZ/V2H RDK	88
9.6. RZV2H RDK Multi-OS Example Packages	88
9.6.1. Hardware supported	89
9.6.2. Software supported	89
9.6.3. Package Specification	89
9.6.4. Installation Guide	90
9.6.5. Usage Guide	92
9.6.6. Troubleshooting	95
10. ROS2 Application Development with RZ/V2H RDK	96
11. ROS2 Application Development	96
11.1. Cross-build the ROS2 Application	96
11.1.1. Cross-build with Yocto SDK	97
11.1.2. Cross-build with Docker and QEMU	101
11.2. Useful tool for development of ROS2 applications	103
11.2.1. Prerequisites	104
11.2.2. VS Code Workspace Configuration	105
11.2.3. Supported Features	105
11.2.4. Prepare for Deployment	106
11.2.5. Start Deployment	107
11.2.6. Start Running / Remote Debugging	108
11.2.7. Example: Workflow	108
11.2.8. Troubleshooting	109
12. Other Concepts	109
12.1. Foxglove Visualization	110
12.2. MuJoCo Visualization	111
12.3. Appendix	113
12.3.1. What is ABI mismatch error?	113
12.3.2. How to avoid ABI mismatch error?	114
12.3.3. How to update/add the Yocto SDK with the correct ROS2 package versions with eSDK?	114
12.3.4. Common ROS 2 topics	116
13. FAQs and Known Issues	118
13.1. Frequency Asked Questions	118
13.2. Known Issues	121

14. Revision History	122
----------------------------	-----

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. Software Environment

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

1.2. Hardware Environment

Items	Description
RZ/V2H	<p>CPU:</p> <ul style="list-style-type: none"> • 4 x Arm Cortex-A55 (1.8GHz) • 2 x Arm Cortex-R8 (800MHz) • 1 x Arm Cortex-M33 (200MHz) <p>DRP:</p> <ul style="list-style-type: none"> • Vision/Dynamically Reconfigurable Processor <p>DRP-AI:</p> <ul style="list-style-type: none"> • Hardware AI Accelerator (8 dense TOPS, 80 sparse TOPS) <p>Package:</p> <ul style="list-style-type: none"> • R9A09G057H44GBG: 1368-pin FCBGA
Memory	LPDDR4 1600MHz (8GB) x 2
SD Card	64GB SanDisk
QSPI Flash ROM	64MB

Items	Description
Interfaces	<ul style="list-style-type: none"> • DC Jack (12-24V / 2A) • JTAG (10-pin) • MIPI CSI-2 4-Lane x2 (22-pin / 0.5mm) • Micro-HDMI • USB3.2 Type-A x2 • USB Micro-B (SCIF) • 10/100/1000 Base-T RJ45 • Micro SD • PCIe 3.0 Root Complex (16-pin / 0.5mm) • CAN-FD x2 • 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.

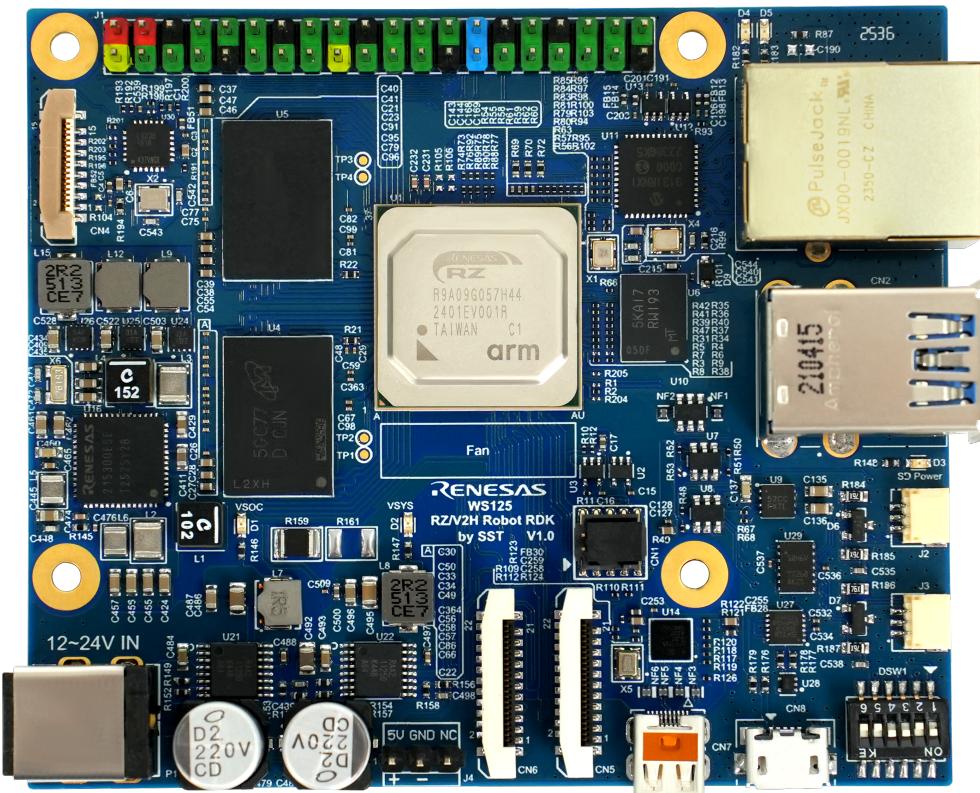


Figure 1. RZ/V2H RDK Board Top View

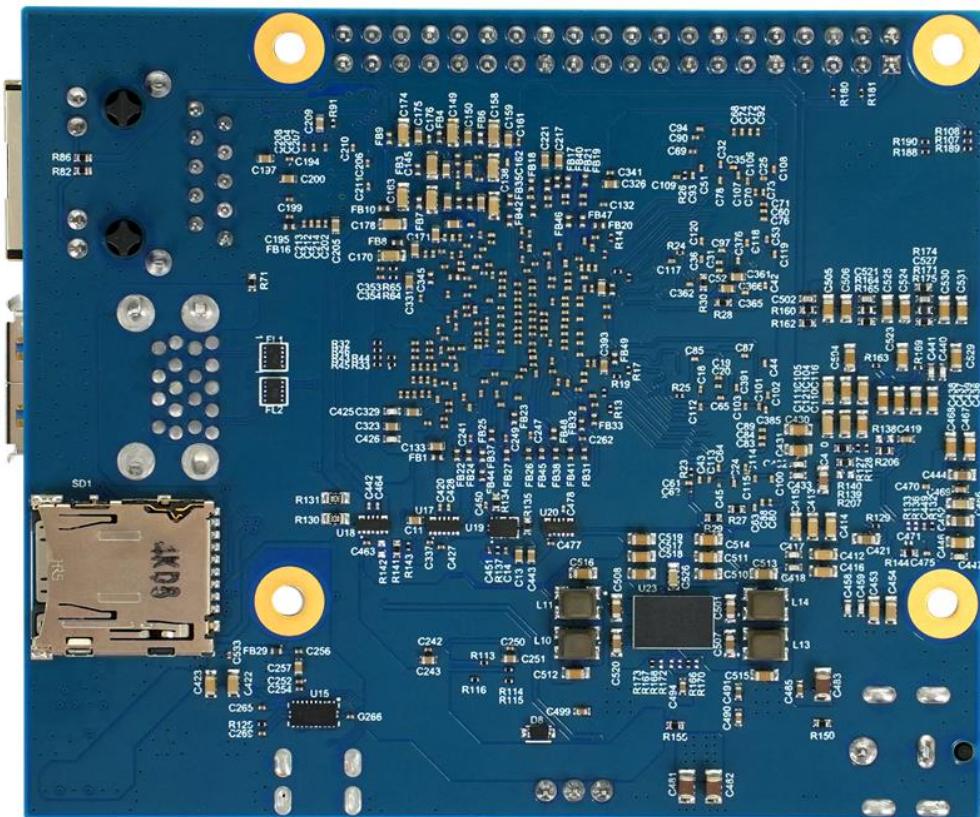


Figure 2. RZ/V2H RDK Board Bottom View

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.

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.

2.1.1. 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
renesas-core-image-weston.wic.gz	Yocto Linux based Weston Image	Windows / macOS / Linux
ubuntu-core-image.wic.gz	Ubuntu 24.04 headless	Windows / macOS / Linux

2.1.2. 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:

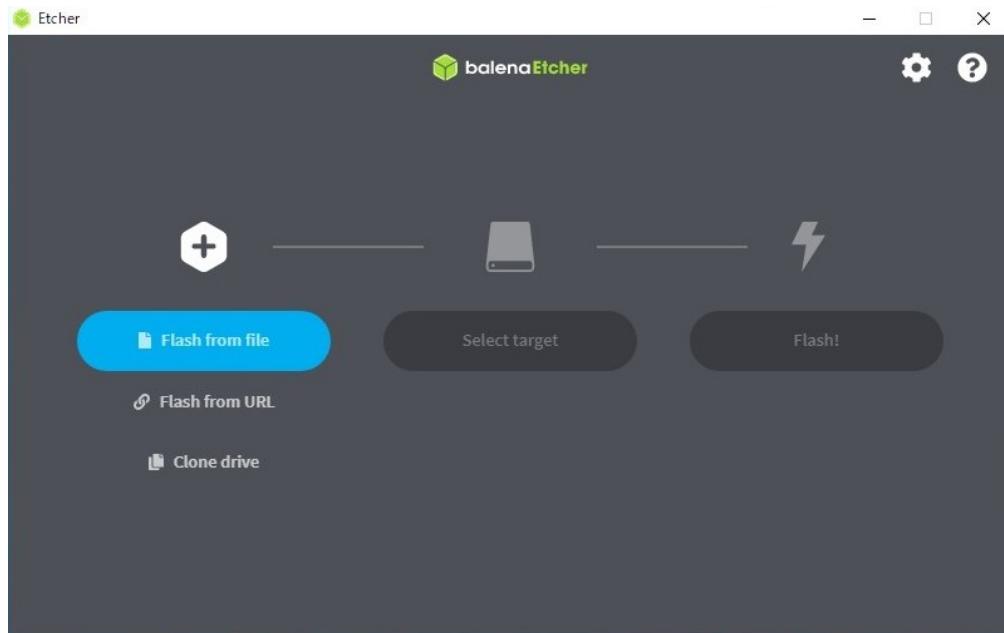


Figure 3. Balena Etcher Application

- **Select Image:** Click “Flash from file” and choose your image file (e.g., ubuntu-core-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.

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: <ul style="list-style-type: none"> • High: CA55 (default) • Low: CM33

DSW1	RZ/V2H Pin	Default Setting	Operation
2, 3	BOOTPLLCA_1 BOOTPLLCA_0	OFF = High: 1 ON = High: 1	<p>Input the CA55 frequency at CA55 coldboot.</p> <p>BOOT_PLLCA[1:0]:</p> <ul style="list-style-type: none"> • 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) (<i>default</i>)
4	MD_BOOT1	ON = Low: 0	Input boot mode select signal.
5	MD_BOOT0	OFF = Low: 0	<p>MD_BOOT[1:0]:</p> <ul style="list-style-type: none"> • Low:Low → SD (<i>default</i>) • Low:High → eMMC • High:Low → xSPI • High:High → SCIF download
6	MD_BOOT3	OFF = Low: 0	<p>Select JTAG debug mode:</p> <ul style="list-style-type: none"> • Low: normal mode (<i>default</i>) • High: JTAG

WARNING Always power off the board before changing boot switches.

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.

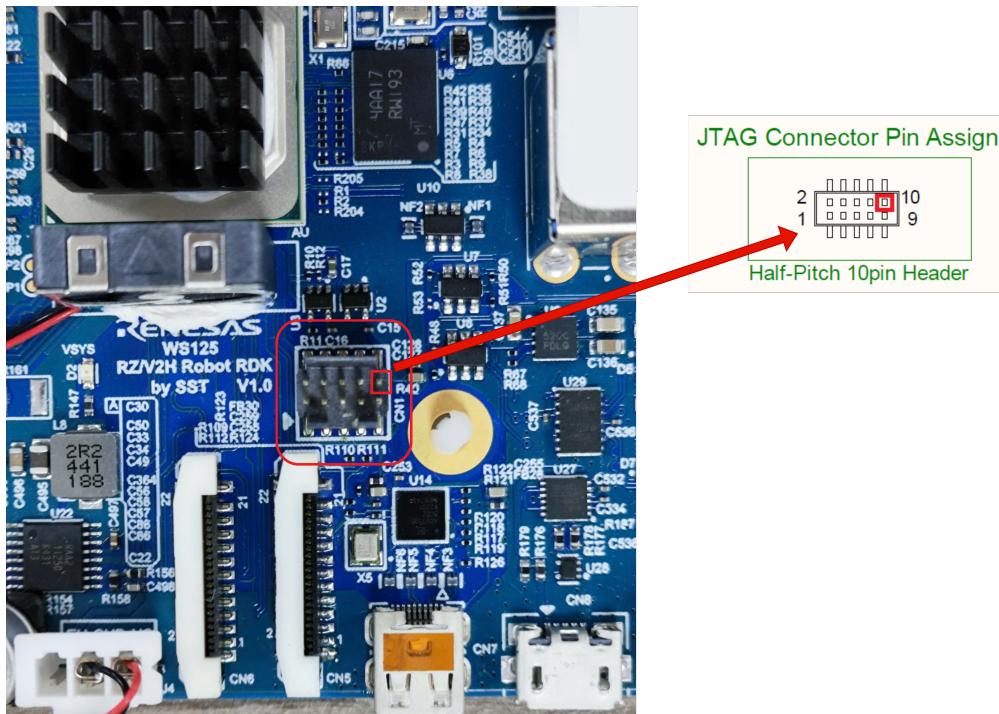


Figure 4. JTAG Reset Pin Example

Before proceeding, ensure that your machine has the necessary drivers and a terminal emulator (MobaXterm, TeraTerm, etc.) installed.

NOTE 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).

The power supply for the RZ/V2H RDK board should satisfy the maximum requirement of 24V / 5A.

IMPORTANT The common DC power adapter specifications are:

- DC power adapter 12V, 2A.
- DC power adapter 24V, 1A.

2.3.1. 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:

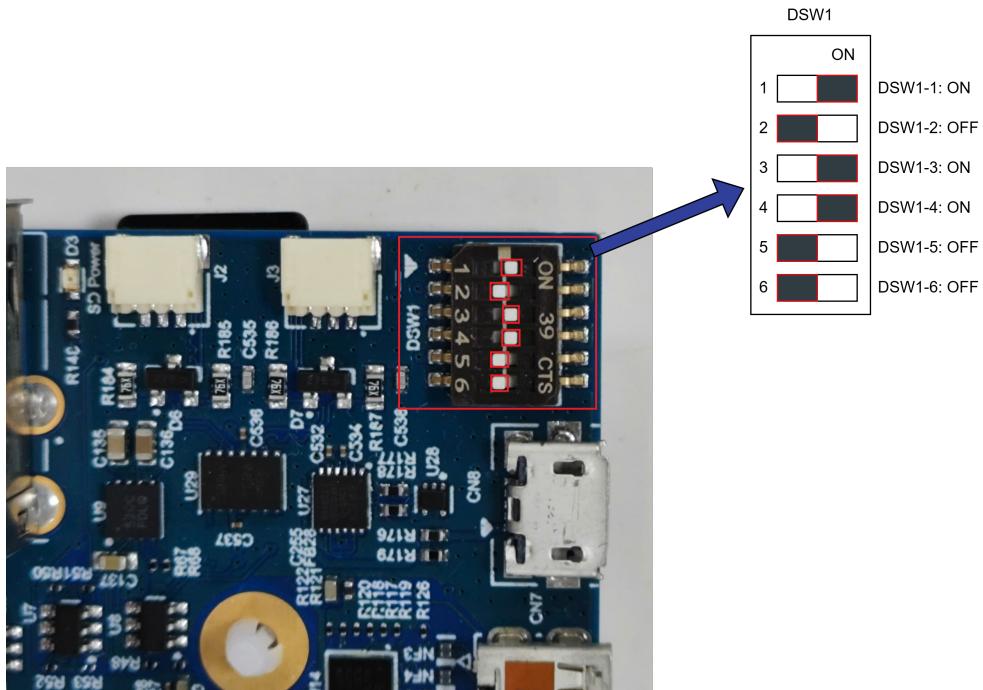


Figure 5. DSW1 SD Card Boot Mode

After that, insert the SD card and connect the power supply 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.

If there is no output from the terminal, do [the JTAG reset tip](#) first, then reset the U-Boot environment variables:

TIP

```
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.

2.3.2. 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. Write Bootloaders to the Board

Copy the bootloaders file to your Windows PC.

File Name	Description
Flash_Writer_SCIF_RZV2H_DEV_INTERNAL_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)**.

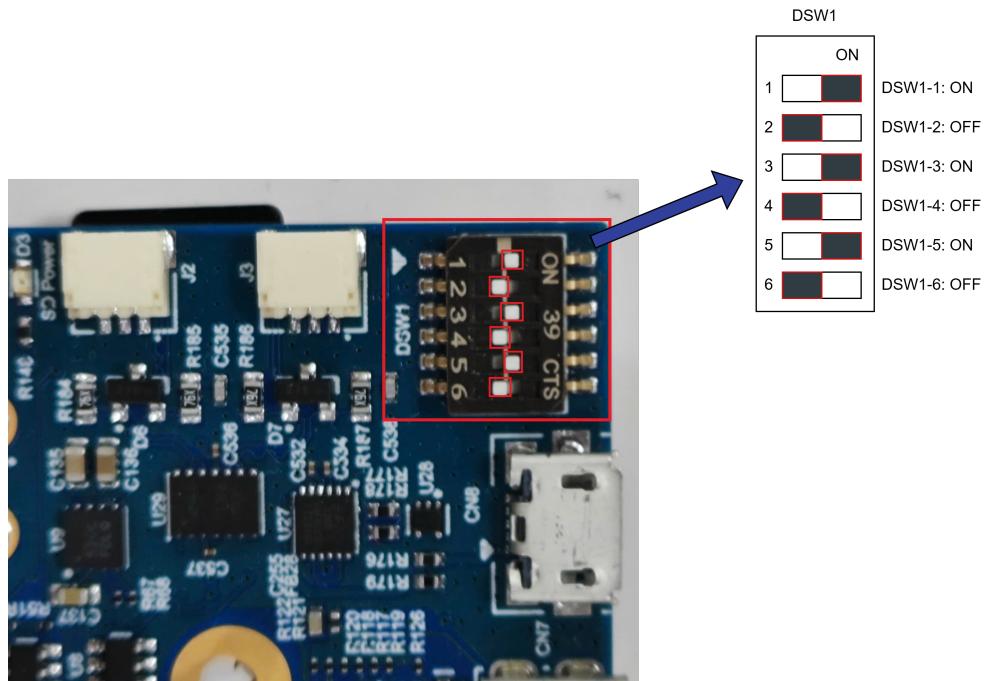


Figure 6. DSW1 SCIF Download Mode

- Connect the power cable.
- 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

- Open **File → Send file...** and send the **Flash Writer** file (.mot) as text.

If the following message is displayed, the file transfer was successful:

```
Flash writer for RZ/V2x Series Vx.xx xxx.xx,20xx
Product Code : RZ/V2x
```

- Next, enter the **XLS2** command in the terminal:

```
> XLS2
===== Qspi writing of RZ/V2x Board Command =====
Load Program to Spiflash
Writes to any of SPI address.
Program size & Qspi Save Address
===== Please Input Program Top Address =====
Please Input : H'
```

- Enter **8101e00**. The log continues:

```
Please Input : H'8101e00
===== Please Input Qspi Save Address ===
Please Input : H'
```

- Enter **00000**. The log continues:

```
Please Input : H'00000
```

please send ! ('.' & CR stop load)

- After the "please send!" message, open **File** → **Send file...** and send the bl2_bp_spi-rzv2*.srec file as text from the terminal software.
- If prompted to clear data, enter **y**:

SPI Data Clear(H'FF) Check : H'00000000-0000FFFF, Clear OK?(y/n)

- The following log will be displayed. The end address depends on the version of IPL:

```
Write to SPI Flash memory.
===== Qspi Save Information =====
SpiFlashMemory Stat Address : H'00000000
SpiFlashMemory End Address : H'00036D17
=====
```

- Enter **XLS2** on the terminal again to get the following messages:

```
> XLS2
===== Qspi writing of RZ/V2x Board Command =====
Load Program to Spiflash
Writes to any of SPI address.
Program size & Qspi Save Address
===== Please Input Program Top Address =====
Please Input : H'
```

- Enter **00000**. The log continues:

```
Please Input : H'00000
===== Please Input Qspi Save Address ===
Please Input : H'
```

- Enter **60000**. The log continues:

```
Please Input : H'60000
please send ! ('.' & CR stop load)
```

- After the "please send!" message, open **File** → **Send file...** and send the fip-rzv2*.srec file as text from the terminal software.

- If prompted to clear data, enter **y**:

```
SPI Data Clear(H'FF) Check : H'00000000-0000FFFF, Clear OK?(y/n)
```

- The following log will be displayed. The end address depends on the version of IPL:

```
Write to SPI Flash memory.
===== Qspi Save Information =====
SpiFlashMemory Stat Address : H'00060000
SpiFlashMemory End Address : H'0011C2EE
=====
```

- Power off the board and change DSW1 to configure the boot mode.

3. Setup U-Boot Configuration

- Insert the microSD card to the board.
- Change DSW1 to **Boot mode 2 (xSPI boot)**:

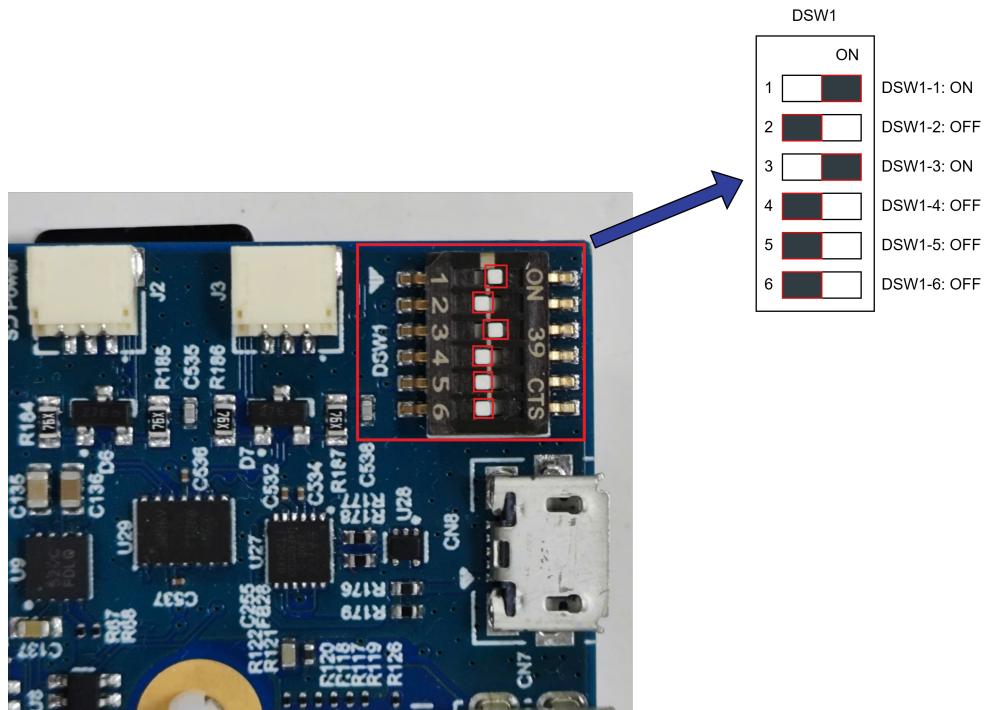


Figure 7. DSW1 xSPI Boot Mode

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

If there is no output from the terminal, do [the JTAG reset tip](#) first, then reset the U-Boot environment variables:

TIP

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

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)
Yocto Linux based Weston Image (renesas core image weston)	root	(none)	(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.
- For Yocto Linux based Weston images, perform **only Step 1** below. **Do not perform Steps 2 to 5.**
- For Ubuntu images, perform **all steps (1 through 5)** below.

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
```

For example, this is the sample output after resizing:

```
$ sudo parted /dev/mmcblk0  
sudo: unable to resolve host localhost.localdomain: Name or service not known  
GNU Parted 3.6  
Using /dev/mmcblk0  
Welcome to GNU Parted! Type 'help' to view a list of commands.  
(parted) print  
Model: SD SN64G (sd/mmc)  
Disk /dev/mmcblk0: 63.9GB  
Sector size (logical/physical): 512B/512B  
Partition Table: msdos  
Disk Flags:  
  
Number  Start   End     Size    Type      File system  Flags  
1        1049kB  211MB   210MB   primary                lba  
2        211MB   4855MB  4644MB  primary   ext4  
  
(parted) resizepart 2 100%  
(parted) print  
Model: SD SN64G (sd/mmc)  
Disk /dev/mmcblk0: 63.9GB  
Sector size (logical/physical): 512B/512B  
Partition Table: msdos  
Disk Flags:  
  
Number  Start   End     Size    Type      File system  Flags  
1        1049kB  211MB   210MB   primary                lba  
2        211MB   63.9GB  63.7GB  primary   ext4  
  
(parted) quit  
Information: You may need to update /etc/fstab.
```

```
$ rz@localhost:~$ sudo resize2fs /dev/mmcblk0p2
sudo: unable to resolve host localhost.localdomain: Name or service not known
resize2fs 1.47.0 (5-Feb-2023)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 8
The filesystem on /dev/mmcblk0p2 is now 15540480 (4k) blocks long.
```

2. Install the ROS2 Jazzy:

We provide the script called: [common_utils/ros2_utils/apt_install_ros2.sh](#) to install ROS2 Jazzy packages on the target system.

Please download the script to the target system and run the following commands:

```
chmod +x apt_install_ros2.sh
sudo ./apt_install_ros2.sh
```

For more details about the ROS2 Jazzy installation, please refer to the [ROS2 Jazzy Installation Guide](#).

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

```
sudo rosdep init
rosdep update
```

4. Setup user groups for: serial port and video access, otherwise some applications may not work properly due to insufficient permissions:

```
sudo usermod -aG dialout $USER
sudo usermod -aG video $USER
```

NOTE

After executing the above commands, please log out and log back in for the group changes to take effect.

5. (Optional) Add ROS2 workspace setup to bashrc:

```
echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

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

2.5. Reference

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

3. 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.

3.1. 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](#).

3.2. 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-cortexa55-rz-cmn-toolchain-ext-5.1.4.sh
Yocto SDK for RZ/V2H RDK	Standard SDK tool-chain for building applications. Obtain the installer: poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh
RZ/V2H RDK X Compile Docker	Preconfigured cross-compilation Docker environment. Refer to the RZ/V2H RDK X Compile Docker repository.

3.3. Quick set up guide

3.3.1. 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/
```

Why copy the Yocto SDK installer into the Docker build context directory?

TIP 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.

TIP In the [Common docker environment setup](#) section, note the following:

- We can create a new Docker container by replacing the [name_of_docker_container] placeholder with a custom name.
- In case you need to rebuild the Docker image (for example, after modifying the SDK), use the following command to rebuild the image:

```
renesas@builder-pc:~/x_compilation_docker$ ./setup_ros2_cross_env.sh .
[name_of_docker_container]
Docker image 'rzv2h_ros_xbuild:latest' already exists.
Do you want to create the container based on this image or create a new Docker
image?
1) Use existing image
2) Rebuild new image
Enter your choice [1 or 2]:
```

Enter 2 and provide a tag name to rebuild the image. The script will back up the existing image using the provided tag before rebuilding.

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
```

3.3.2. eSDK Setup

The purpose of the Yocto eSDK (Extended Software Development Kit) is as follows:

- **Rebuild the Linux kernel and device tree** to customize or modify system behavior for the RZ/V2H platform.
- **Add or update kernel configurations** to enable additional hardware drivers or features.
- **Install missing ROS 2 packages** required for the cross-build workflow to ensure compatibility and completeness of the development environment.

NOTE

If you do not require any of the purposes listed above, you may **skip the eSDK setup** for now. You can always return and perform the eSDK setup later when kernel rebuilding, driver configuration, or additional ROS 2 package installation becomes necessary.

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

Please replace the paths below with your actual file locations.

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.

TIP

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]:/home/ubuntu
```

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% |#####
Initialising tasks: 100% |#####
Checking sstate mirror object availability: 100% |#####
Loading cache: 100% |#####
Initialising tasks: 100% |#####
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.

3.4. 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  
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.

3.5. 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/.kernel-meta
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

If you're unsure whether the config has been added to the kernel configuration,

TIP 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`:

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

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

- Create a configuration fragment file, e.g., myconfig.cfg, inside the ~/poky_sdk/workspace/appends/linux-yocto/ directory. For example:

```
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_FTDI_SI0=y
```

NOTE Please include all dependency configs as well.

- Edit the ~/poky_sdk/workspace/appends/linux-yocto_6.10.bbappend file to include the new configuration fragment at the end of the file:

```
renesas@docker-pc:~$ vi ~/poky_sdk/workspace/appends/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 linux-yocto
renesas@docker-pc:~$ export DISTRO=ubuntu-tiny
renesas@docker-pc:~$ devtool build linux-yocto
```

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

WARNING

```
renesas@docker-pc:~$ devtool build -c 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.

1. Make sure to copy the updated kernel image and device tree blob to the correct locations in the root file system on the SD card.

For example:

Table 2. File Mapping Between eSDK Build Output and Target RootFS

eSDK Build Output Path	Target RootFS Location
<code>~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image/boot/Image-6.10.14-yocto-standard</code>	<code>/boot/Image-6.10.14-yocto-standard</code>
<code>~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image/boot/dtb/renesas/r9a09g057h4-rdk-ver1.dtb</code>	<code>/boot/dtb/renesas/r9a09g057h4-rdk-ver1.dtb</code>
<code>~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image/usr/lib/modules</code>	<code>/usr/lib/modules</code>

TIP

2. On the target device, run the following command to update the module dependencies after deployment:

```
$ sudo depmod -a
```

On the original RZ/V2H RDK image, the kernel image is named `Image` which links to the `Image-6.10.14-yocto-standard` file.

IMPORTANT

The u-boot configuration on the RZ/V2H RDK board loads the kernel image with the name `Image` from the `/boot` directory.

Please make sure to maintain this naming convention when copying the new kernel image to the target root filesystem.

If your new kernel image has a different name, please rename it to [Image](#) after copying it to the target root filesystem.

3.6. Custom Linux Real-time Kernel and Device Tree

This section describes how to customize and rebuild the **Linux Real-time kernel** for the RZ/V2H RDK board using the Yocto eSDK and devtool.

Before proceeding, ensure that you have deployed [board_setup/rt_kernel](#) to your RZ/V2H RDK board as described in the [board_setup/rt_kernel/Readme.md](#). You can find the [board_setup/rt_kernel](#) in the software release package.

IMPORTANT

This image includes the real-time kernel and necessary configurations.

If you want to build your own real-time kernel, follow the steps below.

To enable support for Linux real-time kernel, you need to patch the kernel with the necessary patch. Follow these steps on the host machine where the Yocto eSDK is set up:

1. Download the following real-time kernel patch files from [Common Utils](#) repository:
 - [0000-rz-cmn-rz-Add-realtime-kernel-patch.patch](#)
 - [rt_kernel.cfg](#)
2. Complete the [Step 1 and 2: Applying Patches for linux-yocto](#) in the [Custom Linux Kernel and Device Tree](#) section.
3. Copy the [rt_kernel.cfg](#) and [0000-rz-cmn-rz-Add-realtime-kernel-patch.patch](#) files to the `~/poky_sdk/workspace/appends/linux-yocto/` directory.

```
renesas@docker-pc:~$ mkdir -p ~/poky_sdk/workspace/appends/linux-yocto/
renesas@docker-pc:~$ cd ~/downloads # Change to the directory where you downloaded
the files
```

```
renesas@docker-pc:~$ cp rt_kernel.cfg ~/poky_sdk/workspace/appends/linux-yocto/
renesas@docker-pc:~$ cp 0000-rz-cmn-rz-Add-realtime-kernel-patch.patch
~/poky_sdk/workspace/appends/linux-yocto/
```

4. Edit the `~/poky_sdk/workspace/appends/linux-yocto_6.10.bbappend` file to include the new configuration fragment at the end of file:

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

- Add the following line to the bbappend file:

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

5. Apply the real-time kernel patch to the kernel source:

```
renesas@docker-pc:~$ cd ~/poky_sdk/workspace/sources/linux-yocto
renesas@docker-pc:~/poky_sdk/workspace/sources/linux-yocto$ git am
~/poky_sdk/workspace/appends/linux-yocto/0000-rz-cmn-rz-Add-realtime-kernel-
patch.patch
```

6. **Skip step 3 and 4**, continue build the kernel and deploy it by following **Step 5: Build the Modified Kernel** in the [Custom Linux Kernel and Device Tree](#) section.

For deploying real-time kernel step:

- The [~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image/boot/Image-6.10.14-rt14-yocto-standard](#) should be copied to the target board's `/boot` directory as `Image` not `Image-6.10.14-rt14-yocto-standard`.
- Please don't remove the existing real-time kernel module driver under [/lib/modules/6.10.14-rt14-yocto-standard/kernel/drivers/](#) on the target board.

IMPORTANT

As there are some kernel module drivers (build with 'm' type) are built with the real-time kernel image. Building the linux-yocto recipe will not build these module drivers again.

If you want to add new module drivers, please copy the new module drivers only to the target board.

If you want to switch back to the standard kernel, remove the commit added in [Step 5](#) from the linux-yocto source tree in your workspace, and delete the kernel configuration added in [Step 4](#) from the `linux-yocto_6.10.bbappend` file.

4. 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](#).

4.1. 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:

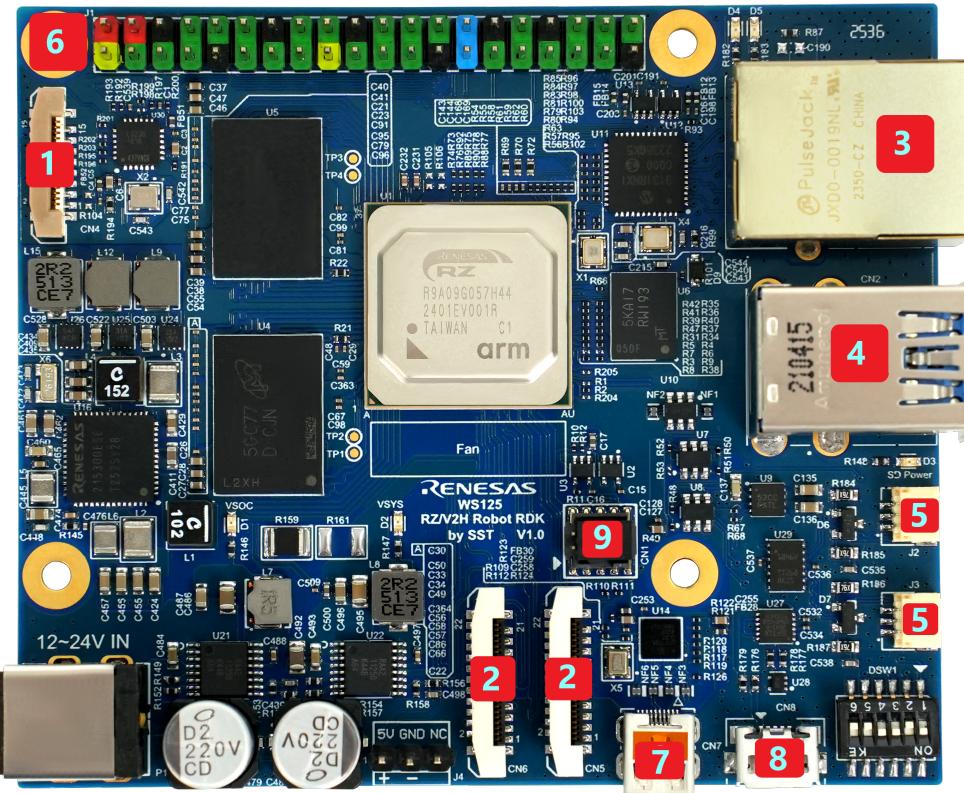


Figure 8. RZ/V2H Robotic Development Kit Interfaces

4.2. Main Interfaces

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

Table 3. 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 x2	Dual camera input interface for image sensors.
3	1000M RJ45	Gigabit Ethernet LAN port for network connectivity.

No	Interface Name	Description
4	USB 3.0 Type A x2	USB host ports for external devices such as mouse, keyboard, or USB camera.

Table 4. Communication Interfaces

No	Interface Name	Description
5	CAN-FD x2	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 5. Other Interfaces

No	Interface Name	Description
7	Micro-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.

4.3. 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 section describes the High-Speed Interface unit of this Kit.

4.3.1. 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.

For example, you can connect a PCIe NVMe SSD to enhance storage performance. Following are the steps to set up and use a PCIe NVMe SSD with the RZ/V2H RDK.

- Hardware Requirements (recommended):
 - [PCIe TO M.2 Board \(D\)](#).
 - M.2 NVMe SSD.
- Hardware Setup:
 1. Power off the RZ/V2H RDK.

2. Connect the PCIe TO M.2 Board to the PCIe 3.0 16-pin connector on the RZ/V2H RDK.
3. Insert the M.2 NVMe SSD into the PCIe TO M.2 Board.
4. Connect 5V power and GND (from GPIO 40 Pins) to the PCIe TO M.2 Board.
5. Power on the RZ/V2H RDK.

IMPORTANT

- Ensure that the PCIe TO M.2 Board is properly powered, as the RZ/V2H RDK does not supply power to PCIe devices.
- Make sure to handle the M.2 NVMe SSD with care to avoid damage from static electricity.
- Make sure you connect the PCIe TO M.2 Board to the correct PCIe 3.0 16-pin connector on the RZ/V2H RDK.

Usage example with pciutils:

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

```
$ sudo apt install pciutils
```

To list all PCIe devices connected to the system, use the following command:

```
$ lspci
```

Example output:

```
root@localhost:~# lspci
00:00.0 PCI bridge: Renesas Technology Corp. Device 003b
01:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd NVMe SSD Controller
980 (DRAM-less)
root@localhost:~#
```

To check the NVMe SSD is recognized by the system, use the following command:

```
$ lsblk
```

Example output:

```
root@localhost:~# lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
mtdblock0  31:0    0 116.5K 1 disk
```

```
mtdblock1 31:1 0 1.8M 1 disk
mtdblock2 31:2 0 128K 1 disk
mtdblock3 31:3 0 14M 0 disk
mmcblk0 179:0 0 29.7G 0 disk
|-mmcblk0p1 179:1 0 100M 0 part
`-mmcblk0p2 179:2 0 2.4G 0 part /
nvme0n1 259:0 0 465.8G 0 disk
```

Mount the NVMe SSD:

```
$ sudo mkdir /mnt/nvme
$ sudo mount /dev/nvme0n1 /mnt/nvme
```

Unmount the NVMe SSD:

```
$ sudo umount /mnt/nvme
$ sudo rmdir /mnt/nvme
```

4.3.2. MIPI-CSI 22-pin connector x2

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

NOTE

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.

TIP

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.

4.3.2.1. 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 `v4l2_init.sh` script in the terminal:

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

For other camera modules, please modify the script accordingly.

4.3.2.2. 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
```

To capture an image from the camera using **Renesas Core Image Weston**, use the following command:

```
$ gst-launch-1.0 v4lsrc device=/dev/video0 ! videoconvert ! waylandsink
```

4.3.3. 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
```

4.3.3.1. 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:
    eno0:
      dhcp4: no
      dhcp6: no
      addresses: [169.254.43.99/24]
      routes:
        - to: default
          via: 169.254.43.86
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

NOTE

Make sure to replace the `addresses` and `routes` with the appropriate values for your router network.

- Apply the changes with the following command:

```
$ sudo netplan apply
```

4.3.3.2. 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
```

4.3.4. USB 3.0 Type A x2

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.

4.3.4.1. 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/1p, 480M
/: Bus 004.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p, 20000M/x2
root@localhost:~#
  
```

4.3.4.2. USB-WIFI Adapter Support

The following USB-WIFI adapters have been tested and are compatible with the RZ/V2H RDK:

- Ralink Technology, Corp. MT7601U Wireless Adapter
- AC1300 TP-Link T3U Nano

Please refer to the [USB-WIFI Adapter Support](#) section for more detailed instructions on driver and

firmware installation for these adapters.

4.3.4.3. Usage example

- Check USB Devices

First, connect the USB-WIFI adapter to the RZ/V2H RDK.

Then, run the following command to list all connected USB devices:

```
$ lsusb
```

Example output:

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 003: ID 2357:0138 TP-Link 802.11ac NIC
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 003: ID 148f:7601 Ralink Technology, Corp. MT7601U Wireless Adapter
```

- Check Interface Name

```
$ ip a | grep wl
```

Example output:

```
7: wlx98ba5f1918cf: <BROADCAST,MULTICAST,DYNAMIC> mtu 1500 qdisc noqueue state DOWN
group default qlen 1000
```

- Check Device Status

```
$ nmcli dev status
```

Example output:

DEVICE	TYPE	STATE	CONNECTION
end0	ethernet	connected	Wired connection 1
lo	loopback	connected (externally)	lo
wlx98ba5f1918cf	wifi	unavailable	--
can0	can	unmanaged	--

can1	can	unmanaged	--
dummy0	dummy	unmanaged	--
sit0	iptunnel	unmanaged	--

- Enable the WiFi Interface

If the WiFi interface is disabled, the STATE will show as "unavailable".

We have to enable the WiFi interface first:

```
$ sudo nmcli radio wifi on
```

- Recheck Device Status

```
$ nmcli dev status
```

Example output:

DEVICE	TYPE	STATE	CONNECTION
end0	ethernet	connected	Wired connection 1
lo	loopback	connected (externally)	lo
wlx98ba5f1918cf	wifi	disconnected	--
can0	can	unmanaged	--
can1	can	unmanaged	--
dummy0	dummy	unmanaged	--
sit0	iptunnel	unmanaged	--

- List Available WiFi Connections

```
$ nmcli dev wifi list
```

Example output:

IN-USE SECURITY	SSID	MODE	CHAN	RATE	SIGNAL	BARS
WPA2	C8:7F:54:E1:45:10 MY_SSID	Infra	8	270 Mbit/s	100	██████

- Connect to WiFi Network

```
# Replace MY_SSID and yourpassword with the actual WiFi SSID and password.
$ sudo nmcli dev wifi connect MY_SSID password "yourpassword"
```

Example output:

```
sudo: unable to resolve host localhost.localdomain: Name or service not known
Device 'wlx98ba5f1918cf' successfully activated with '0fde1c53-876d-4752-adba-8b6f758d9c51'.
```

- Test Network Connectivity

```
$ ping -I wlx98ba5f1918cf bing.com
```

Example output:

```
PING bing.com (150.171.27.10) from 192.168.19.177 wlx98ba5f1918cf: 56(84) bytes of
data.
64 bytes from 150.171.27.10: icmp_seq=1 ttl=120 time=420 ms
64 bytes from 150.171.27.10: icmp_seq=2 ttl=120 time=482 ms
--- bing.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 419.837/451.166/482.495/31.329 ms
```

- Disconnect from WiFi Network

```
$ sudo nmcli con down id MY_SSID
```

4.4. Communication Interfaces

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

4.4.1. CAN-FD x2

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.

To use the CAN-FD interfaces, ensure that you have the required CAN connectors.

TIP

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.

Follow the steps below to use the CAN-FD interfaces on the RZ/V2H RDK running the Ubuntu system.

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##01122334455667788
```

4.4.2. 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)

4.4.2.1. Pin Out Diagram

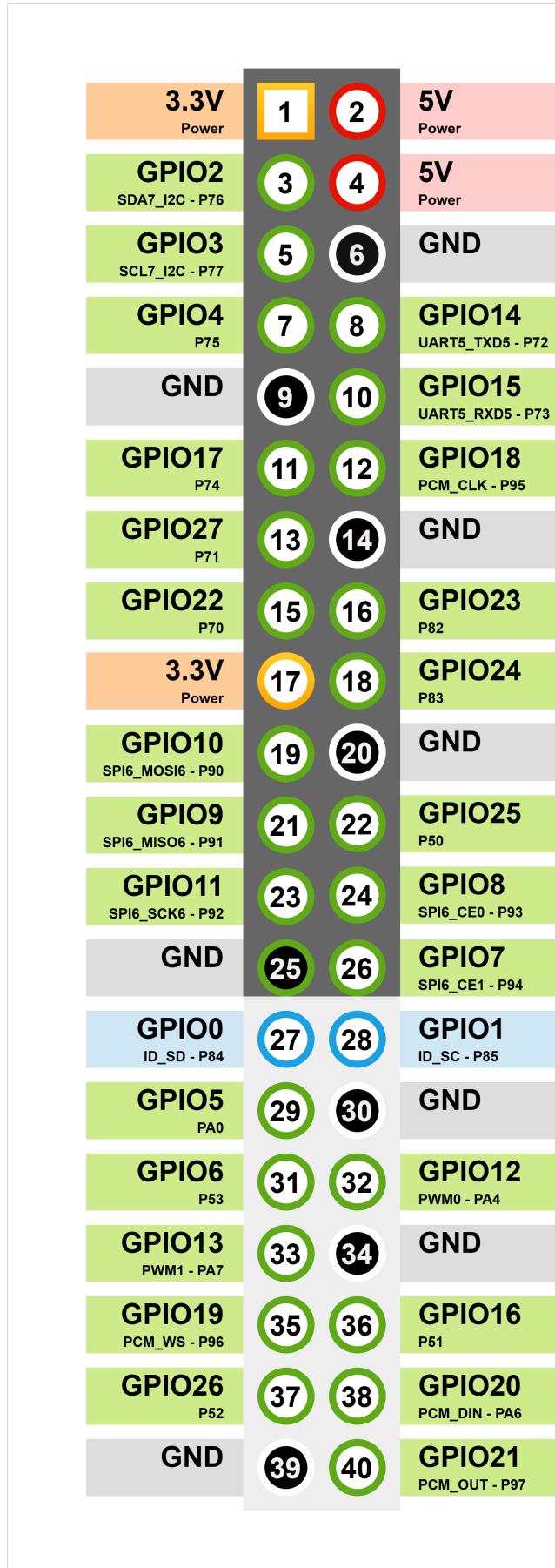


Figure 9. RZ/V2H RDK Raspberry Pi GPIO 40-pin Header Pin Out

4.4.2.2. 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 6. 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:

```
$ 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.

How to identify the correct I2C bus number for I2C7?

TIP 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
```

4.4.2.3. 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 7. 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
00000000 3231 3433 3635 3837 3039
000000a
```

4.4.2.4. 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 8. 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
```

4.4.2.5. 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
```

4.4.2.6. 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 9. 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.

NOTE This feature is not available for this release.

4.5. Other interfaces

The RZ/V2H Robotic Development Kit (RDK) is equipped with several additional interfaces to enhance its functionality and connectivity options. This section provides an overview of these interfaces, including Micro-HDMI, USB-UART, and JTAG 10-pin.

4.5.1. Micro-HDMI

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

- The display supports a maximum resolution of 1920x1920.

NOTE

- Before powering on the board, ensure that the HDMI cable is securely connected to avoid any display issues.

4.5.2. 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

4.5.3. 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](#).

5. RZ/V2H Advance Features

The RZ/V2H Advance Features section provides detailed documentation on the advanced functionalities available on the RZ/V2H platform, including hardware accelerators and multi-OS capabilities.

6. DRP-AI

This section provides an overview of the DRP-AI (Dynamically Reconfigurable Processor for AI) Driver available on the RZ/V2H platform, along with instructions on how to utilize its features effectively.

6.1. Overview

The RZ/V2H DRP-AI Driver enables the use of the DRP-AI (Dynamically Reconfigurable Processor for AI) hardware accelerator on the RZ/V2H platform.

This driver facilitates efficient execution of AI inference tasks by offloading computations to the DRP-AI, thereby improving performance and reducing CPU load and high power efficiency.

The DRP-AI device driver provides an interface to easily handle the AI inference execution function of DRP-AI. So that there is no hardware knowledge required for the user.

6.2. Concepts

6.2.1. What is Dynamically Reconfigurable Processor (DRP)?

DRP is the hardware IP (Intellectual Property) that can dynamically change its hardware (arithmetic logic circuit) configuration.

Its main advantages are the ability to reduce surface area and power consumption whilst maintaining high performance.

Dynamic Reconfiguration can change the configuration of the arithmetic circuit in execution.

This image below shows an example of dynamic reconfiguration:

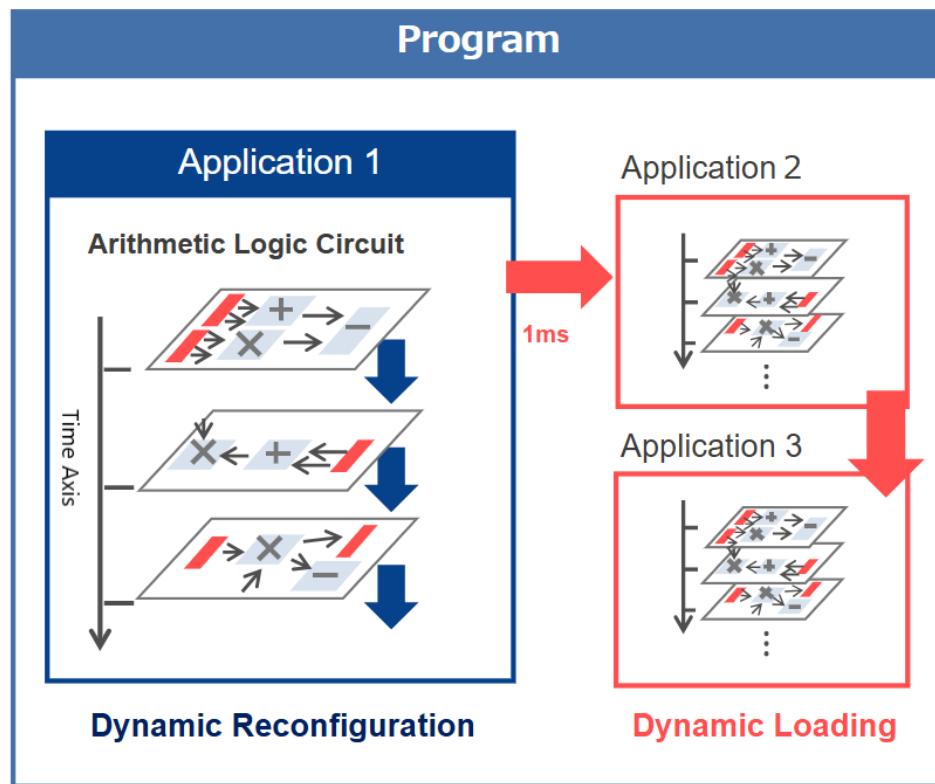


Figure 10. DRP Dynamic Reconfiguration

6.2.2. What is DRP-AI?

DRP-AI is a specialized version of DRP designed specifically for AI (Artificial Intelligence) processing tasks.

By combining the DRP and AI-MAC (AI Matrix Arithmetic Circuit) to accelerate AI inference tasks efficiently.

On the RZ/V2H platform, the DRP-AI3 is used as the DRP-AI hardware accelerator. For more information about the DRP-AI3, refer to the:

NOTE **DRP-AI3 White Paper:** [Next Generation Highly Power-Efficient AI Accelerator \(DRP AI3\)](#).

Software Package: [AI Accelerator: DRP-AI](#).

DRP-AI Driver Architecture:

- Buffer to reuse input data
- Switches to avoid zero data processing
- Controller to optimize operation flow (scheduling)

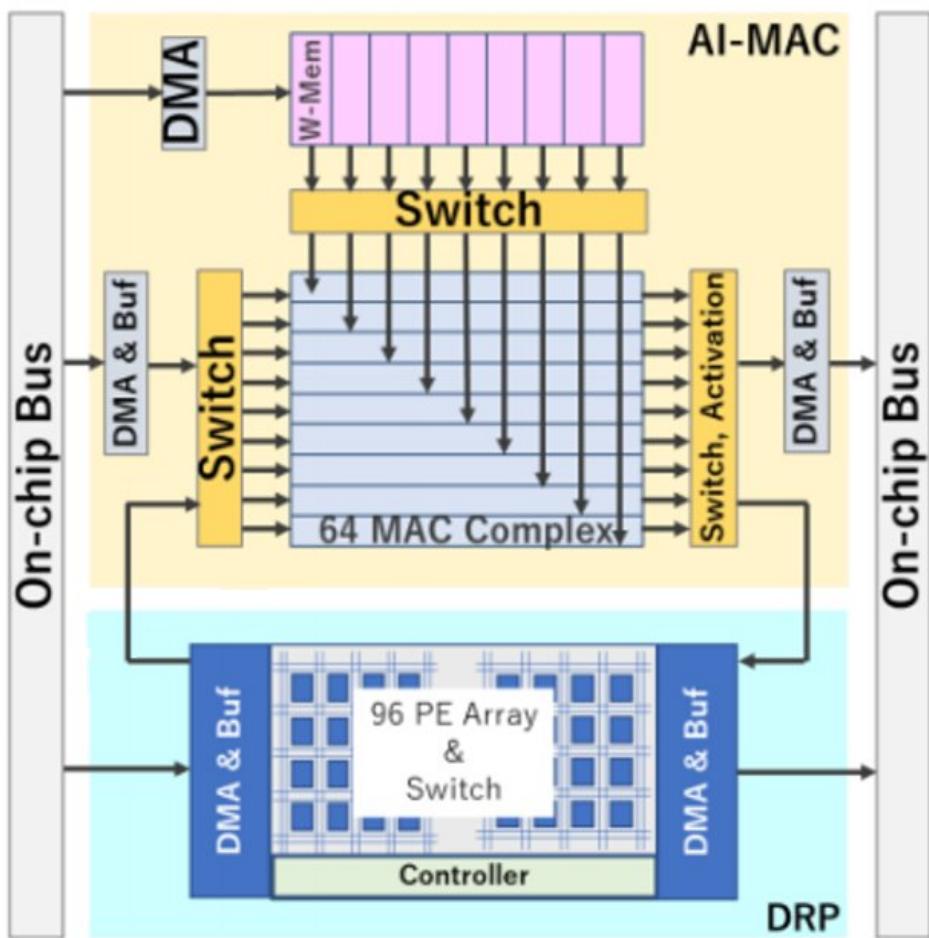


Figure 11. DRP-AI Driver Architecture

DRP-AI Driver Execution Flow:

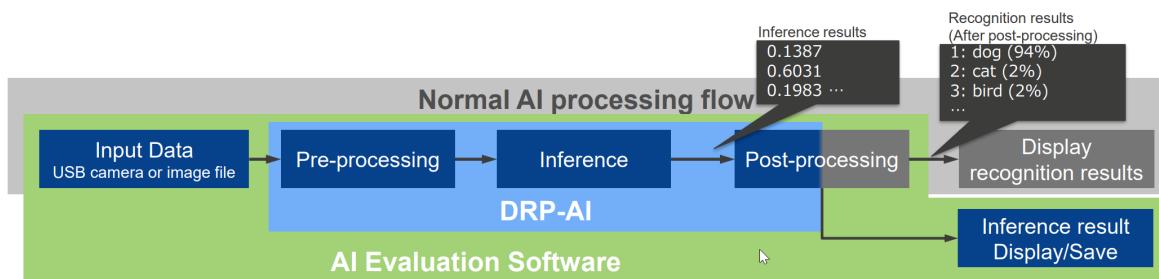


Figure 12. DRP-AI Driver Execution Flow

The DRP-AI Driver handles the following tasks to execute AI inference on the DRP-AI:

1. Pre-processing: Prepares input data for DRP-AI processing, including format conversion, crop the image, normalization, etc...
2. Inference execution: Manages the execution of the AI model on the DRP-AI hardware.

3. Post-processing: Processes the output data from DRP-AI to obtain the final inference results.

For more detail information about the DRP-AI Driver, refer to the [RZ/V2H DRP-AI Driver](#).

NOTE

It provide API functions to help you get started with DRP-AI Driver development on the RZ/V2H platform.

6.3. BYOM AI model support

The DRP-AI supports BYOM (Bring Your Own Model) AI models, allowing users to deploy their custom-trained AI models on the RZ/V2H platform.

6.3.1. Getting Started

To enable BYOM support, users need to convert their AI models into a format compatible with the DRP-AI using the:

Extension package of TVM Deep Learning Compiler for Renesas DRP-AI accelerators powered by EdgeCortix MERA™ (DRP-AI TVM).

This package provides the necessary tools and libraries to facilitate the conversion process, ensuring that the models can leverage the DRP-AI's capabilities effectively.

For more information about the DRP-AI TVM extension package, refer to the:

NOTE

- [DRP-AI TVM Extension Package](#).
- [DRP-AI TVM on RZ/V series](#).

6.3.2. Install the DRP-AI TVM extension package

To install the DRP-AI TVM extension package, follow the instructions provided in the [rzv_ai_toolchain_docker repository](#).

IMPORTANT

Please use the RZ/V2H SDK environment when installing the DRP-AI TVM extension package. Do not use SDKs provided by other platforms.

6.3.3. BYOM Development Flow

The typical development flow for deploying BYOM AI models on the RZ/V2H platform using the DRP-AI TVM extension package involves the following steps:

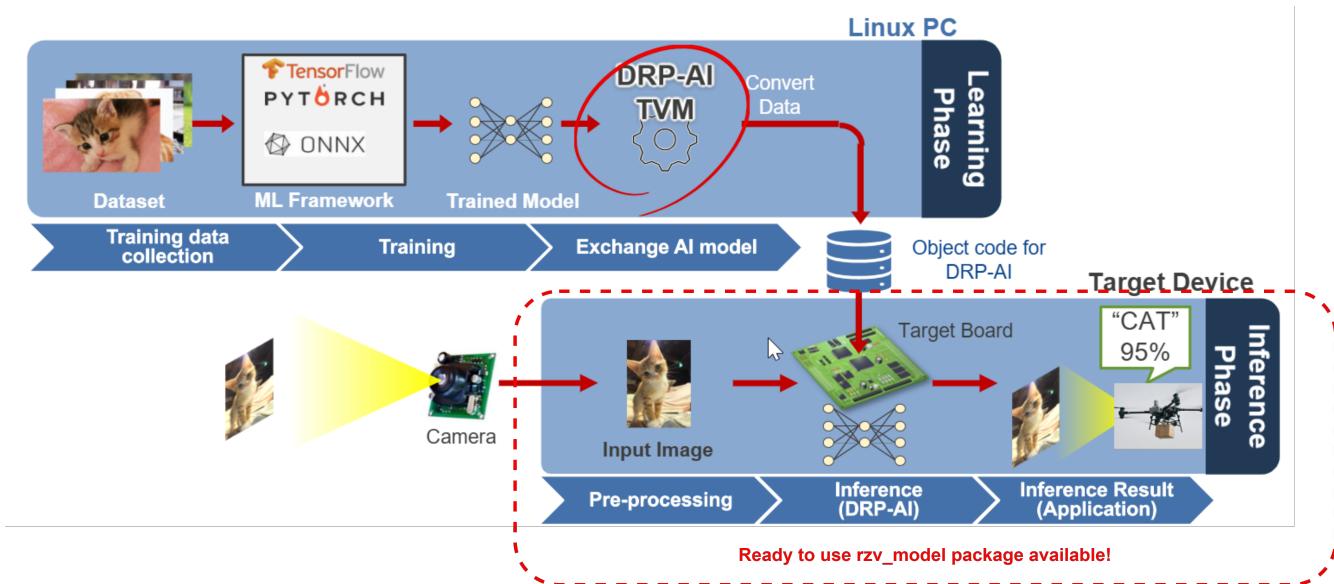


Figure 13. BYOM Development Flow

1. Training data collection: Gather and prepare the dataset required for training the AI model.
2. Model training: Use a deep learning framework (e.g., TensorFlow, PyTorch, ONNX) to train the AI model on the collected dataset.
3. Exchange AI model: Convert the trained AI model into a format compatible with the DRP-AI using the DRP-AI TVM extension package.
4. Deployment: Deploy the converted model onto the RZ/V2H platform and integrate it with the DRP-AI Driver for inference execution.

In the deployment step, the ready to use `rzv_model` package is provided to simplify the integration of compiled models with the DRP-AI Driver.

TIP

We also provide some example of complete flow about developing some popular AI models with DRP-AI TVM extension package. Refer to the [DRP-AI with `rzv_model` tutorials](#) for more details.

6.3.4. Training data collection and Model training

For training data collection and model training, users can utilize popular deep learning frameworks such as [TensorFlow](#), [PyTorch](#), or [ONNX](#).

List of AI models that Renesas has verified for conversion with the DRP-AI TVM:
[Model list for RZ/V2H](#).

NOTE

Note that, the above list is not exhaustive, and users can attempt to convert other models as well.

6.3.5. Exchange AI model

To convert the trained AI model into a format compatible with the DRP-AI for V2H target device, follow the instructions provided in the DRP-AI TVM extension package repository: [Compile with Sample Scripts](#).

TIP Follow the convert tips to easily convert your AI model: [How to convert](#).

6.3.6. Deployment

For deploying the converted AI model onto the RZ/V2H platform, the rzv_model package is provided to facilitate the integration process.

The rzv_model package providing AI model abstractions and implementations for Renesas RZ/VMPU platforms.

This package implements various models for computer vision tasks using the DRP-AI accelerator.

Refer to the next section for more details about the rzv_model package and its usage.

6.4. The rzv_model package

The **rzv_model** package is a C++ library designed to facilitate the deployment of AI models on the RZ/V2H platform using the DRP-AI driver.

It provides a unified interface for loading, running, and managing various AI models optimized for the RZ/V2H architecture.

6.4.1. Base Framework

The **rzv_model** package provides a flexible and modular framework for deploying AI models optimized on DRP-AI driver for the RZ/V2H platform. It includes the following core features:

- Abstracted model interface with hardware acceleration support.
- Common pre-processing and post-processing utilities for image-based inference.
- DRP-AI runtime integration for efficient inference execution.
- Support multiple AI model running at the same time with DRP-AI driver.
- Support for both YUV422 and RGB image formats.

6.4.2. Implemented Models

The following AI models are implemented, optimized and ready to use with DRP-AI acceleration.

Object Detection Models

- YOLOX Base Model
- YOLOX Hand Detection
- YOLOX Pascal VOC Detection
- Gold YOLOX Hand Detection
- YOLOv8 Base Model
- YOLOv8n Rock Paper Scissors Gesture Detection

Pose Estimation Models

- HRNetV2 Base Model
- HRNetV2 Hand Landmark Model
- RTMPose Base Model
- RTMPose Hand Model
- MediaPipe Hand Landmark Model

6.4.3. Package structure

The `rzv_model` package is organized into the following structure:

```

rzv_model/
├── CMakeLists.txt
├── config
│   └── models
└── include
    └── rzv_model
        ├── base_model.hpp
        ├── model_specific.hpp
        └── utils.hpp
├── package.xml
└── README.md
└── src
    ├── base_model.cpp
    ├── model_specific.cpp
    └── platform
        ├── MeraDrpRuntimeWrapper.cpp
        ├── MeraDrpRuntimeWrapper.h
        ├── PreRuntime.h
        └── PreRuntimeV2H.cpp
    └── utils.cpp

```

- The `config/models` directory contains configuration files for each supported AI model,

including the output from the DRP-AI TVM conversion step.

- The `include/rzv_model` directory contains the header files defining the base model class and utility functions.
- The `src` directory contains the implementation of the base model, platform-specific runtime wrappers, and utility functions.
- The `CMakeLists.txt` and `package.xml` files are used for building and packaging the library.

6.4.4. Architecture

The `rzv_model` package follows a **modular architecture** designed for extensibility, maintainability, and efficient deployment on DRP-AI.

- **Base Model:** Provides the `BaseModel` class, which implements shared functionalities such as model loading, pre-processing, inference execution, and result handling.
- **Model-Specific Implementations:** Each AI model (e.g., YOLOX, YOLOv8, HRNet, RTMPose) inherits from the base class and extends it with task-specific logic such as detection parsing or key point extraction.
- **Utility Modules:** Contain helper functions for image pre-processing, tensor conversion, normalization, and post-processing visualization.

This modular design enables developers to easily integrate new AI models and customize pre-processing or inference pipelines for various use cases on the RZ/V2H platform.

6.4.5. How to use the `rzv_model` package

To use the `rzv_model` package, you need to prepare the model configuration files, including the compiled model files from the DRP-AI TVM conversion process.

Please follow the steps below to set up and use the `rzv_model` package effectively.

6.4.5.1. Input requirements files

After complete the [Exchange AI model](#) step, the output should contain the compiled model files including:

```
Ouput_Directory/
  |-- deploy.json
  |-- deploy.params
  |-- deploy.so
  |-- input_0.bin
  \-- preprocess
    |-- addr_map.txt
    |-- aimac_cmd.bin
```

```

|-- aimac_desc.bin
|-- aimac_param_cmd.bin
|-- aimac_param_desc.bin
|-- drp_config.mem
|-- drp_desc.bin
|-- drp_param.bin
|-- drp_param_info.txt
`-- weight.bin

```

Also, to enable multiple AI models running simultaneously with the DRP-AI driver, a special file called **addr_map.txt** is required.

This **addr_map.txt** file is for the model in the inference phase. The purpose is to get the memory size used as multiple models scenario each model shall be allocated a memory block in advance before running.

This file is different from the one generated in the **Ouput_Directory** folder.

To obtain the **addr_map.txt** file, locate it in the **temp** folder created during the model conversion process.

For example, if you compile the model in the **/drp-ai_tvm/tutorials/** directory, a **temp** folder will be generated automatically.

You can find the **addr_map.txt** file in the following path:

```
/drp-
ai_tvm/tutorials/temp/<date_time>/tvmgen_default_tvmgen_default_mera_drp_main_*/drp_compilation
_output/
```

There might be several sub-directories representing different inference stages (executed by DRP-AI or CPU) that each contain an **addr_map.txt** file.

The correct file to use is the one with the largest memory address allocation, corresponding to the **maximum drp_desc value**, as it represents the final and complete memory size used by the AI Model with DRP-AI driver.

How to calculate the memory size used by the model?

- Open the **addr_map.txt** file with a text editor.

This file contains several lines, each representing a memory block with its start address and size. For example:

```

data_in 32ec4c0 dc00
data 32fa0c0 41040
data_out 333b100 12c00

```

```

work 334dd00 80
weight 334dd80 bd80
drp_config 3359b00 87480
aimac_param_cmd 33e0f80 140
aimac_param_desc 33e10c0 50
aimac_cmd 33e1140 1300
aimac_desc 33e2440 170
drp_param 33e2600 350
drp_desc 33e2980 380

```

- Calculate the total memory size by summing the sizes of all `drp_desc` entries:

In this example, the description of `drp_desc` is: start address `33e2980` and size `380`.

Therefore, the total memory size for `drp_desc` is: `33e2980` (start address) + `380` (size) = `33e2d00`.

- Find the correct `addr_map.txt` file that contains the largest `drp_desc` value, corresponding to the total memory size used by the AI model with the DRP-AI driver.

6.4.5.2. Construct the Model configuration files

To construct the model, import the **Input requirements files** above with the following structure:

```

Model_name/
|-- addr_map.txt
|-- deploy.json
|-- deploy.params
|-- deploy.so
`-- preprocess
    |-- addr_map.txt
    |-- aimac_cmd.bin
    |-- aimac_desc.bin
    |-- aimac_param_cmd.bin
    |-- aimac_param_desc.bin
    |-- drp_config.mem
    |-- drp_desc.bin
    |-- drp_param.bin
    |-- drp_param_info.txt
    `-- weight.bin

```

Note that, the top-level `addr_map.txt` file is required for multiple models running with DRP-AI driver.

This `Model_name` folder will place under the `config/models` directory of the `rzv_model` package.

6.4.5.3. Post-processing configuration

Each model may have different post-processing requirements based on its specific task (e.g., object detection, pose estimation).

To customize the post-processing behavior, you can modify the corresponding model-specific implementation files located in the **src/** directory of the **rzv_model** package.

The details of post-processing configuration are not covered in this section. Please refer to the example in the next section for a clearer understanding.

There are some sample applications from Renesas, which have the custom post-processing for the specific models.

TIP You can refer to these applications for reference on implementing the AI model post-processing logic:

- [RZ/V AI Applications Repository](#).
- [Ignitarium Renesas - RZ/V AI Applications](#).

6.4.5.4. Example of using rzv_model package

Load the model and perform inference using the **rzv_model** package with the following code snippet:

```
// Example using HRNetV2 Hand Landmark model
auto model = std::make_unique<rzv_model::HRNetV2HandLandmarkModel>();
model->load(model_path);

// Prepare input
rzv_model::ModelInput input{image, roi};

// Run inference
auto result = model->run<rzv_model::KeyPointResult>(input);
```

6.5. DRP-AI with rzv_model tutorials

This section provides tutorials on how to use the **rzv_model** package to deploy AI models on the RZ/V2H platform with DRP-AI acceleration.

Make sure you have completed the steps in the [BYOM AI model support](#) section to set up the DRP-AI TVM extension package in your development machine.

Prerequisites

On the development machine, which has the DRP-AI TVM environment, clone the [Hand Models](#) repository, which contains the necessary model files and conversion scripts for the tutorials.

```
$ git clone
https://partnergitlab.renesas.solutions/sst1/industrial/ws078/ai/hand_models.git
```

Then, set the hand models directory environment variable:

```
$ export HAND_MODELS_DIR=<path_to_cloned_hand_models_directory>
```

Next, change to the TVM tutorials directory and copy the compilation script:

```
$ cd $TVM_ROOT/tutorials
$ cp ${HAND_MODELS_DIR}/compilation/compile_onnx_model_quant_<model>.py .
```

Follow the specific tutorial instructions below to convert and deploy different hand detection and landmark models using the **rzv_model** package.

6.5.1. YOLOX Object Detection Tutorial

This tutorial describes how to use the **YOLOX Hand Detection** model for object detection with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the [Prerequisites step of the tutorial](#) before proceeding.

For this tutorial, we will utilize the pre-trained YOLOX model.

6.5.1.1. Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the hand_models/compilation directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- YOLOX Hand

```
$ python3 compile_onnx_model_quant_yolox_hand_fhd_crop.py \
$HAND_MODELS_DIR/palm_detection/yolox/yolox_hand.onnx \
-o ../data/yolox_hand_fhd_crop \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,640,640 \
-v 100 \
```

```
--images $HAND_MODELS_DIR/dataset/scripts/selected_hands/
```

- Gold YOLO Nano

```
$ python3 compile_onnx_model_quant_gold_yolo_fhd_crop.py \
$HAND_MODELS_DIR/palm_detection/gold_yolo/gold_yolo_n_hand_0303_0.4172_1x3x480x640.
onnx \
-o ../data/gold_yolo_n_fhd_crop \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,480,640 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
-v 100
```

6.5.1.2. Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to [the rzv_model package usage](#).

6.5.2. YOLOv8 Object Detection Tutorial

This tutorial describes how to use the **YOLOv8n Rock Paper Scissors Gesture Detection** model for object detection with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the [Prerequisites step of the tutorial](#) before proceeding.

6.5.2.1. Installation requirements

- Ultralytics YOLOv8
- Dataset for Rock Paper Scissors Gesture Detection

You can use the [Rock-Paper-Scissors Dataset](#) from roboflow.

Please download and prepare the dataset as YOLOv8 format for training.

6.5.2.2. Train the Model using Ultralytics YOLOv8 and transfer learning method

Please download the dataset above with the YOLOv8 format and locate the **data.yaml** file, replace the path in the below Python script and save as **train_model_yolov8.py**

```

from ultralytics import YOLO

# Loading pre-trained model
# Can select between: yolov8n.pt yolov8s.pt yolov8m.pt yolov8l.pt yolov8x.pt
model = YOLO('yolov8n.pt')

# Display model information
model.info()

# Train the model with 80 epochs and set the input image size is 640x640
# The number of epochs can be changed to adapt to your requirements
results = model.train(data="<path-to-the-dataset-folder>/rock-paper-scissors-14/data.yaml", epochs=80, imgsz=640)

```

Run the training process:

```
$ python3 train_model_yolov8.py
```

For faster training time, the GPU with appropriate driver should be installed.

Result will be located at: <workspace-directory>/runs/detect/trainX/weights/best.pt where X will increase each time you run the training process.

6.5.2.3. Export and cut the model

Since some parts of the post-processing phase of YOLOv8 model cannot be handled by the DRP-AI hardware, we need to remove certain steps to ensure it can run on the DRP-AI hardware.

Please follow this documentation to learn how to export the model to ONNX format and cut the model:

- [How to convert yolov8 onnx models V2H.md](#)

After exporting and cutting the model, you will get the ONNX model file named [yolov8n_rps_cut.onnx](#) (the name can be different based on your configuration).

6.5.2.4. Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the hand_models/compilation directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- YOLOv8n Rock Paper Scissors Gesture Detection

```
$ python3 compile_onnx_model_quant_yolov8_fhd_crop.py \
```

```
$HAND_MODELS_DIR/palm_detection/yolov8/yolov8n_rps_cut.onnx \
-o yolov8_rps \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,640,640 \
-v 100 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_rps_hands
```

6.5.2.5. Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to [the rzv_model package usage](#).

6.5.3. MediaPipe Hand Landmark Tutorial

This guide describes the process for converting **MediaPipe Hand Landmark Detection** models from **TFLite** to **ONNX** format for compatibility with **DRP-AI**.

Please complete the [Prerequisites step of the tutorial](#) before proceeding.

6.5.3.1. Download Models

- Reference: [MediaPipe Models List](#)
- [MediaPipe Hand Landmarker Models Documentation](#)
- [Hand Landmark Full Model](#)
- [Hand Landmark Lite Model](#)
- [Palm Detection Full Model](#)
- [Palm Detection Lite Model](#)

6.5.3.2. Set Up Conversion Environment

- Reference: [tensorflow-onnx GitHub Repository](#) — Convert TensorFlow, Keras, TensorFlow.js, and TFLite models to ONNX.
- Create a dedicated conda environment with Python 3.10:

```
$ conda create --name tf2onnx python=3.10
$ conda activate tf2onnx
```

6.5.3.3. Conversion Process

- **Important:** The supported ONNX opset for the DRP-AI translator is **v12**.
- Install dependencies (downgrade NumPy to resolve compatibility issues):

```
$ pip install tensorflow-onnx
$ pip install numpy==1.26.4
```

- Convert the models to ONNX format (using NCHW layout):

```
$ python -m tf2onnx.convert \
--tflite hand_landmark_full.tflite \
--opset 12 \
--output hand_landmark_full.onnx \
--inputs-as-nchw input_1
$ python -m tf2onnx.convert \
--tflite palm_detection_full.tflite \
--opset 12 \
--output palm_detection_full.onnx \
--inputs-as-nchw input_1
```

- For the Lite version, use:

```
$ python -m tf2onnx.convert \
--tflite hand_landmark_lite.tflite \
--opset 12 \
--output hand_landmark_lite.onnx \
--inputs-as-nchw input_1
$ python -m tf2onnx.convert \
--tflite palm_detection_lite.tflite \
--opset 12 \
--output palm_detection_lite.onnx \
--inputs-as-nchw input_1
```

6.5.3.4. Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the hand_models/compilation directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

Best performance and accuracy:

```
$ SPARSE_ENABLE=false python3 compile_onnx_model_quant_mp_handlandmark_fhd_crop.py \
```

```
$HAND_MODELS_DIR/hand_landmark_estimation/mp_hand_landmark/hand_landmark_full.onnx \
-o ../data/mp_handlandmark_fhd_crop \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,224,224 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
-v 100 \
-p "--calibrate_method Entropy"
```

MediaPipe Evaluation model:

- Full model:

```
$ SPARSE_ENABLE=false \
python3 compile_onnx_model_quant_mediapipe.py \

$HAND_MODELS_DIR/hand_landmark_estimation/mp_hand_landmark/hand_landmark_full.onnx \
-o ../data/mediapipe_entropy \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,224,224 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
-v 100 \
-p "--calibrate_method Entropy"
```

- Lite model:

```
$ SPARSE_ENABLE=false \
python3 compile_onnx_model_quant_mediapipe.py \

$HAND_MODELS_DIR/hand_landmark_estimation/mp_hand_landmark/hand_landmark_lite.onnx \
-o ../data/mediapipe_lite_entropy \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,224,224 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
-v 100 \
-p "--calibrate_method Entropy"
```

Optional: MediaPipe Palm Detection

```
$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false python3
compile_onnx_model_quant_mp_palm_det.py \
$HAND_MODELS_DIR/palm_detection/mediapipe/palm_detection_full.onnx \
-o ../data/mp_palm_det \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,192,192 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
-v 100
```

6.5.3.5. Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to [the rzv_model package usage](#).

6.5.4. RTMPose Hand Landmark Tutorial

This tutorial describes how to use the **RTMPose Hand Landmark** model for hand landmark estimation with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the [Prerequisites step of the tutorial](#) before proceeding.

For this tutorial, we will utilize the pre-trained RTMPose model.

6.5.4.1. Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the hand_models/compilation directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- RTMPose Hand Landmark

```
$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false \
python3 compile_onnx_model_quant_rtmpose_fhd_crop.py \
$HAND_MODELS_DIR/hand_landmark_estimation/rtmpose/rtmpose.onnx \
-o ../data/rtmpose_fhd_crop \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,256,256 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
```

```

-v 100 \
-p "--calibrate_method Entropy \
    --node_to_exclude \
    /Shape,/Slice,/Concat,/Reshape,\
    /mlp/mlp.0/ReduceL2,/mlp/mlp.0/Mul,\
    /mlp/mlp.0/Clip,/mlp/mlp.0/Div,\
    /mlp/mlp.0/Mul_1,/mlp/mlp.1/MatMul,\
    /gau/ln/ReduceL2,/gau/ln/Mul,\
    /gau/ln/Clip,/gau/ln/Div,\
    /gau/ln/Mul_1,/gau/uv/MatMul,\
    /gau/act_fn/Sigmoid,/gau/act_fn/Mul,\
    /gau/Split,/gau/Unsqueeze,\
    /gau/Mul,/gau/Add,/gau/Split_1,\
    /gau/Squeeze_1,/gau/Transpose,\
    /gau/Squeeze,/gau/MatMul,\
    /gau/Div,/gau/Relu,/gau/Mul_1,\
    /gau/MatMul_1,/gau/Mul_2,\
    /gau/o/MatMul,/gau/res_scale/Mul,\
    /gau/Add_1,/cls_x/MatMul,\
    /cls_y/MatMul"

```

- RTMPose Evaluation with basic compilation

```

$ python3 compile_onnx_model_quant_rtmpose.py \
$HAND_MODELS_DIR/hand_landmark_estimation/rtmpose/rtmpose.onnx \
-o ../data/rtmpose_freihand \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,256,256 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
-v 100

```

- RTMPose Evaluation with Optimized Compilation (Recommended)

```

$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false \
python3 compile_onnx_model_quant_rtmpose.py \
$HAND_MODELS_DIR/hand_landmark_estimation/rtmpose/rtmpose.onnx \
-o ../data/rtmpose_optimized \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,256,256 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
-v 100 \

```

```
-p "--calibrate_method Entropy \
    --node_to_exclude \
    /Shape,/Slice,/Concat,/Reshape,\
    /mlp/mlp.0/ReduceL2,/mlp/mlp.0/Mul,\
    /mlp/mlp.0/Clip,/mlp/mlp.0/Div,\
    /mlp/mlp.0/Mul_1,/mlp/mlp.1/MatMul,\
    /gau/ln/ReduceL2,/gau/ln/Mul,\
    /gau/ln/Clip,/gau/ln/Div,\
    /gau/ln/Mul_1,/gau/uv/MatMul,\
    /gau/act_fn/Sigmoid,/gau/act_fn/Mul,\
    /gau/Split,/gau/Unsqueeze,\
    /gau/Mul,/gau/Add,/gau/Split_1,\
    /gau/Squeeze_1,/gau/Transpose,\
    /gau/Squeeze,/gau/MatMul,\
    /gau/Div,/gau/Relu,/gau/Mul_1,\
    /gau/MatMul_1,/gau/Mul_2,\
    /gau/o/MatMul,/gau/res_scale/Mul,\
    /gau/Add_1,/cls_x/MatMul,\
    /cls_y/MatMul"
```

6.5.4.2. Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to [the rzv_model package usage](#).

6.5.5. HRNetV2 Hand Landmark Tutorial

This tutorial describes how to use the **HRNetV2 Hand Landmark** model for hand landmark estimation with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the [Prerequisites step of the tutorial](#) before proceeding.

For this tutorial, we will utilize the pre-trained HRNetV2 model.

6.5.5.1. Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the hand_models/compilation directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- HRNetv2 Hand Landmark

```
$ python3 compile_onnx_model_quant_hrnetv2_fhd_crop.py \
$HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.onnx \
-o ../data/hrnetv2_hands_fhd_crop \
```

```
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,256,256 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
-v 100
```

- HRNetv2 Evaluation with Selected Hands Dataset

```
$ python3 compile_onnx_model_quant_hrnetv2.py \
$HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.onnx \
-o ../data/hrnetv2_selected_hands \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,256,256 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
-v 100
```

- HRNetv2 Evaluation with FreiHand Dataset

```
$ python3 compile_onnx_model_quant_hrnetv2.py \
$HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.onnx \
-o ../data/hrnetv2_freibhand \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,256,256 \
--images $HAND_MODELS_DIR/dataset/scripts/freibhand_rois/ \
-v 100
```

- HRNetv2 Evaluation with Entropy Calibration

```
$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false \
python3 compile_onnx_model_quant_hrnetv2.py \
$HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.onnx \
-o ../data/hrnetv2_selected_hands_entropy \
-t $SDK \
-d $TRANSLATOR \
-c $QUANTIZER \
-s 1,3,256,256 \
--images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
-v 100 \
```

```
-p "--calibrate_method Entropy"
```

6.5.5.2. Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to [the rzv_model package usage](#).

6.6. Renesas AI Applications

AI Applications are sets of application source code, pre-build application binary and pre-trained AI model objects, which allow users to run AI Applications easily and quickly.

Users can select the category of applications and access the applications provided on GitHub.

This section introduces the Renesas AI Applications provided by Renesas and third-party developers for the RZ/V2H RDK platform.

For more details, please visit the [Renesas AI Applications](#) page.

6.6.1. Prerequisites

Before running the Renesas AI Applications, ensure that you have completed the following prerequisites:

- Set up the RZ/V2H RDK with **renesas-core-image-weston.wic.gz** image flashed on the microSD card. Follow the instructions in the [Quick Setup Guide](#) to set up the environment.
- Building the DRP-AI TVM toolchain docker, follow the instructions provided in the [rzv_ai_toolchain_docker repository](#).

IMPORTANT

Please use the RZ/V2H SDK when installing the DRP-AI TVM extension package. **Do not use SDKs provided by other platforms.**

- Install the **libtvm_runtime.so** library to the target RZ/V2H RDK platform:

```
$ wget https://github.com/renesas-rz/rzv_drp-
ai_tvm/raw/refs/heads/v2.5.1/obj/build_runtime/V2H/libtvm_runtime.so
$ mv libtvm_runtime.so /usr/lib/
$ ldconfig
```

6.6.2. Available AI Applications

The following AI Applications are available, developed by Renesas and third-party developers:

- [RZ/V AI Applications Repository](#).
- [RZ/V Sample Applications Repository](#).
- [Ignitarium Renesas - RZ/V AI Applications](#).
- [Computermind corporation - DRP-AI Demo App](#).

6.6.3. Building Renesas AI Applications

Select the desired AI Application from the repositories mentioned above and clone the repository to your DRP-AI TVM toolchain docker.

Based on the repository, follow the specific porting instructions below to build the application.

The following common steps need to be performed for all the applications that use wayland functions:

TIP

- Update the wayland initialization function in the wayland source file.
- Update the CMakeLists.txt file to include the correct GStreamer version and wayland source file.

6.6.3.1. Common step

Inside the DRP-AI toolchain docker, update the `/drp-ai_tvm/how-to/sample_app_v2h/common_files/wayland_scarthgap.cpp` file to include `wl_display_roundtrip` after `wl_surface_commit` as shown below:

```
diff --git a/how-to/sample_app_v2h/common_files/wayland_scarthgap.cpp b/how-to/sample_app_v2h/common_files/wayland_scarthgap.cpp
index 20a9236..10cc132 100644
--- a/how-to/sample_app_v2h/common_files/wayland_scarthgap.cpp
+++ b/how-to/sample_app_v2h/common_files/wayland_scarthgap.cpp
@@ -293,6 +293,9 @@ static int8_t initWaylandDisplay(struct wl_display** wlDisplay,
 struct wl_surfac
     xdg_surface_add_listener(xdg_surface, &xdg_surface_listener, NULL);

     wl_surface_commit(*wlSurface);
+
+    wl_display_roundtrip(*wlDisplay);
+
     return 0;
 }
```

```
}
```

6.6.3.2. How to port and build [rzv_drp-ai_tvm/how-to/sample_app_v2h](#) applications

The following step by step guide shows how to port [rzv_drp-ai_tvm/how-to/sample_app_v2h/app_yolox_cam](#) application as an example.

You can follow the same steps to port other applications available in the [rzv_drp-ai_tvm/how-to/sample_app_v2h](#) folder.

Inside the DRP-AI toolchain docker, navigate to the [/drp-ai_tvm/how-to/sample_app_v2h/app_yolox_cam/src](#) folder.

- Update the [CMakeLists.txt](#) file:

Before change (line 21-31 of CMakeLists.txt file):

```
if(NOT EXISTS "$ENV{SDK}/sysroots/cortexa55-poky-linux")
    list(REMOVE_ITEM SOURCE ${TVM_ROOT}/../how-
to/sample_app_v2h/common_files/wayland_scarthgap.cpp)
    add_definitions(-DV2H)
    message("-- For Dunfell or older")
else()
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.22.12/gst-libs/gst/gl)
    list(REMOVE_ITEM SOURCE ${TVM_ROOT}/../how-
to/sample_app_v2h/common_files/wayland.cpp)
    list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.22.12/gst-libs/gst/gl/xdg-shell-client-protocol.c)
    add_definitions(-DV2H -DV2N)
    message("-- For Scarthgap or lator")
endif()
```

After modification:

```
if(NOT EXISTS "$ENV{SDK}/sysroots/cortexa55-poky-linux")
    list(REMOVE_ITEM SOURCE ${TVM_ROOT}/../how-
to/sample_app_v2h/common_files/wayland_scarthgap.cpp)
    add_definitions(-DV2H)
    message("-- For Dunfell or older")
else()
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl)
    list(REMOVE_ITEM SOURCE ${TVM_ROOT}/../how-
to/sample_app_v2h/common_files/wayland.cpp)
```

```

list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl/xdg-shell-client-protocol.c)
add_definitions(-DV2H -DV2N)
message("-- For Scarthgap or lator")
endif()

```

- Refer to the [README.md](#) file in the [app_yolox_cam](#) folder for further build instructions to compile and run the application on the RZ/V2H RDK platform.

6.6.3.3. How to port and build Renesas AI Applications: [RZ/V AI Applications Repository](#)

Clone the `rzv_ai_sdk` repository to your DRP-AI TVM toolchain docker.

```

$ cd /drp-ai_tvm/data
$ git clone https://github.com/renesas-rz/rzv_ai_sdk.git
$ cd rzv_ai_sdk
$ git checkout 7362488f2b59c46c337c797e9412acef6ed9dd7c

```

The following step by step guide shows how to port Renesas AI Applications [Q01_footfall_counter](#) as an example.

You can follow the same steps to port other Renesas AI Applications available in the repository.

- Update the `rzv_ai_sdk/Q01_footfall_counter/src/CMakeLists.txt` file:

Before change (line 16-36 of CMakeLists.txt file):

```

if(PRODUCT STREQUAL "V2H")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.h)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland_scarthgap.cpp)
    add_definitions(-DV2H)
    message("-- PRODUCT [V2H]")
elseif(PRODUCT STREQUAL "V2N")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.h)
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.22.12/gst-libs/gst/gl)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland.cpp)
    list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.22.12/gst-libs/gst/gl/xdg-shell-client-protocol.c)
    add_definitions(-DV2H -DV2N)
    message("-- PRODUCT [V2N]")

```

```

else()
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.h)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland_scarthgap.cpp)
    add_definitions(-DV2L)
    message("-- PRODUCT [V2L]")
endif()

```

After modification:

```

if(PRODUCT STREQUAL "V2H")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.h)
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland.cpp)
    list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl/xdg-shell-client-protocol.c)
    add_definitions(-DV2H -DV2N)
    message("-- PRODUCT [V2H]")
endif()

```

IMPORTANT

Because this application don't use wayland functions, there is no need to update the [rzv_ai_sdk/Q01_footfall_counter/src/wayland_scarthgap.cpp](#) file.

However, if you are porting other applications that use wayland functions ([src](#) folder have the [wayland_scarthgap.cpp](#) file), please make sure to update the corresponding wayland file as mentioned in the [Common Step](#) section above.

For example, with [rzv_ai_sdk/Q04_fish_classification](#) application, we have to update the [/drp-ai_tvm/data/rzv_ai_sdk/Q04_fish_classification/src/wayland_scarthgap.cpp](#) file.

- Continue with the **step 3** in the build instructions provided in the repository to compile and run the application on the RZ/V2H RDK platform: [Q01_footfall_counter Application File Generation](#).

6.6.3.4. How to port and build and build RZ/V Sample Applications: [RZ/V Sample Applications Repository](#).

Clone the `rzv_sample_apps` repository to your DRP-AI TVM toolchain docker.

```
$ cd /drp-ai_tvm/data
$ git clone https://github.com/renesas-rz/rzv_sample_apps.git
$ cd rzv_sample_apps
$ git checkout d760587ea766830d8ca224fc67a05941194358fd
```

The following step by step guide shows how to port RZ/V Sample Applications `rzv_sample_apps` as an example.

You can follow the same steps to port other RZ/V Sample Applications available in the repository.

- Update the `rzv_sample_apps/S01_face_mosaic/src/CMakeLists.txt` file:

Before change (line 16-24 of CMakeLists.txt file):

```
if(V2H STREQUAL "ON")
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/PreRuntime.cpp)
    add_definitions(-DV2H)
    message("-- PRODUCT [V2H]")
else()
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/PreRuntimeV2H.cpp)
    add_definitions(-DV2L)
    message("-- PRODUCT [V2L]")
endif()
```

After modification:

```
if(V2H STREQUAL "ON")
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/PreRuntime.cpp)
    add_definitions(-DV2H -DV2N)
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl)
    list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl/xdg-shell-client-protocol.c)
    message("-- PRODUCT [V2H]")
else()
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/PreRuntimeV2H.cpp)
    add_definitions(-DV2L)
    message("-- PRODUCT [V2L]")
endif()
```

- Replace the `rzv_sample_apps/S01_face_mosaic/src/wayland.cpp` and the `rzv_sample_apps/S01_face_mosaic/src/wayland.h` files

Please copy the `/drp-ai_tvm/how-to/sample_app_v2h/common_files/wayland_scarthgap.cpp` and `/drp-ai_tvm/how-to/sample_app_v2h/common_files/wayland_scarthgap.h` files respectively to replace them.

- Continue with the **step 3** in the build instructions provided in the repository to compile and run the application on the RZ/V2H RDK platform: [S01_face_mosaic Application File Generation](#).

6.6.3.5. How to port and build and build Ignitarium Renesas RZ/V AI Applications: [Ignitarium Renesas - RZ/V AI Applications](#).

Clone the Ignitarium-Renesas/rzv_ai_apps repository to your DRP-AI TVM toolchain docker.

```
$ cd /drp-ai_tvm/data
$ git clone https://github.com/Ignitarium-Renesas/rzv_ai_apps.git
$ cd rzv_ai_apps
$ git checkout d607cba6e24ffd8a5093ecab45f7bedd5e010b5d
```

The following step by step guide shows how to port Renesas AI Applications [01_Head_count](#) as an example.

You can follow the same steps to port other Renesas AI Applications available in the repository.

- Update the `rzv_ai_apps/01_Head_count/src/CMakeLists.txt` file:

Before change (line 16-39 of CMakeLists.txt file):

```
if(PRODUCT STREQUAL "V2H")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.cpp
        ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.h)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/PreRuntime.cpp)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland_scarthgap.cpp)
    add_definitions(-DV2H)
    message("-- PRODUCT [V2H]")
elseif(PRODUCT STREQUAL "V2N")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.cpp
        ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.h)
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
        base/1.22.12/gst-libs/gst/gl)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/PreRuntime.cpp)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland.cpp)
```

```

list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.22.12/gst-libs/gst/gl/xdg-shell-client-protocol.c)
add_definitions(-DV2H -DV2N)
message("-- PRODUCT [V2N]")
else()
include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230)
file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.h)
list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/PreRuntimeV2H.cpp)
list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland_scarthgap.cpp)
add_definitions(-DV2L)
message("-- PRODUCT [V2L]")
endif()

```

After modification:

```

if(PRODUCT STREQUAL "V2H")
include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251)
file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.h)
include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl)
list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/PreRuntime.cpp)
list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland.cpp)
list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl/xdg-shell-client-protocol.c)
add_definitions(-DV2H -DV2N)
message("-- PRODUCT [V2H]")
endif()

```

- Update the function `initWaylandDisplay` in the `rzv_ai_apps/01_Head_count/src/wayland_scarthgap.cpp` file, please add `wl_display_roundtrip` after `wl_surface_commit` as shown below:

```

diff --git a/01_Head_count/src/wayland_scarthgap.cpp
b/01_Head_count/src/wayland_scarthgap.cpp
index 0fb92a4..b0a6d11 100644
--- a/01_Head_count/src/wayland_scarthgap.cpp
+++ b/01_Head_count/src/wayland_scarthgap.cpp
@@ -323,6 +323,9 @@ static int8_t initWaylandDisplay(struct wl_display** wlDisplay,
struct wl_surfac
    xdg_surface_add_listener(xdg_surface, &xdg_surface_listener, NULL);

    wl_surface_commit(*wlSurface);

```

```

+
+     wl_display_roundtrip(*wlDisplay);
+
+     return 0;
}

```

- Continue with the **step 3** in the build instructions provided in the repository to compile and run the application on the RZ/V2H RDK platform: [01_Head_count Application File Generation](#).

6.6.3.6. How to port and build Computermind corporation DRP-AI Demo App: [Computermind corporation - DRP-AI Demo App](#).

Clone the ComputermindCorp/drp-ai-demo-app repository to your DRP-AI TVM toolchain docker.

```

$ cd /drp-ai_tvm/data
$ git clone https://github.com/ComputermindCorp/drp-ai-demo-app.git
$ cd drp-ai-demo-app
$ git checkout fe24fc8e8b94028f8cf60cd670e5df12665ed2d9

```

The following step by step guide shows how to port Computermind corporation DRP-AI Demo App [C01_river_area_monitoring](#) as an example.

You can follow the same steps to port other Computermind corporation DRP-AI Demo Apps available in the repository.

- Update the [drp-ai-demo-app/C01_river_area_monitoring/src/CMakeLists.txt](#) file:

Before change (line 20-34 of CMakeLists.txt file):

```

if(PRODUCT STREQUAL "V2H")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v230/*.h)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland_scarthgap.cpp)
    add_definitions(-DV2H)
    message("-- PRODUCT [V2H]")
elseif(PRODUCT STREQUAL "V2N")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.h)
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.22.12/gst-libs/gst/gl)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland.cpp)
    list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.22.12/gst-libs/gst/gl/xdg-shell-client-protocol.c)

```

```

add_definitions(-DV2H -DV2N)
message("-- PRODUCT [V2N]")
endif()

```

After modification:

```

if(PRODUCT STREQUAL "V2H")
    include_directories(${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251)
    file(GLOB SOURCE *.cpp *.h ${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.cpp
${CMAKE_SOURCE_DIR}/drp-ai_tvm_v251/*.h)
    include_directories(${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl)
    list(REMOVE_ITEM SOURCE ${CMAKE_SOURCE_DIR}/wayland.cpp)
    list(APPEND SOURCE ${CMAKE_SYSROOT}/usr/src/debug/gstreamer1.0-plugins-
base/1.24.10/gst-libs/gst/gl/xdg-shell-client-protocol.c)
    add_definitions(-DV2H -DV2N)
    message("-- PRODUCT [V2H]")
endif()

```

- Update the function `initWaylandDisplay` in the `drp-ai-demo-app/C01_river_area_monitoring/src/wayland_scarthgap.cpp` file, please add `wl_display_roundtrip` after `wl_surface_commit` as shown below:

```

diff --git a/C01_river_area_monitoring/src/wayland_scarthgap.cpp
b/C01_river_area_monitoring/src/wayland_scarthgap.cpp
index 03af867..5c2ff08 100644
--- a/C01_river_area_monitoring/src/wayland_scarthgap.cpp
+++ b/C01_river_area_monitoring/src/wayland_scarthgap.cpp
@@ -293,6 +293,9 @@ static int8_t initWaylandDisplay(struct wl_display** wlDisplay,
struct wl_surfac
    xdg_surface_add_listener(xdg_surface, &xdg_surface_listener, NULL);

    wl_surface_commit(*wlSurface);
+
+    wl_display_roundtrip(*wlDisplay);
+
    return 0;
}

```

- Continue with the **step 3** in the build instructions provided in the repository to compile and run the application on the RZ/V2H RDK platform: [C01_river_area_monitoring Application: Build Stage](#).

7. Video Codec Library

The RZ/V2H Video Codec Library provides hardware-accelerated (combined with [DRP IP](#)) video encoding and decoding capabilities.

This software supports the following features:

- Support for H.264 decoding and encoding
- Support for H.265 decoding and encoding

IMPORTANT

In this release, the Video Codec Library is available only for the Yocto image: Renesas Core Image Weston.

NOTE

For more information on how to use the Video Codec Library, refer to the [Linux Interface Specification GStreamer](#).

It provide GStreamer plugins for easy integration with available Video Codec functionalities.

8. OpenCV Accelerator

The RZ/V2H MPU platform features the OpenCV Accelerator (OpenCVA) that utilizes the DRP IP to accelerate specific OpenCV functions for enhanced performance in computer vision applications.

8.1. Overview

The RZ/V2H OpenCV Accelerator (OpenCVA) leverages the OpenCV library to provide optimized computer vision functionalities on the RZ/V2H platform by using the [DRP IP](#).

Based on the [OpenCV version 4.10.0](#), this feature enables efficient image processing and computer vision tasks by offloading some computations to the DRP, thereby enhancing performance and reducing CPU load.

NOTE

For more detail information about OpenCVA, refer to the [RZ/V2H OpenCV Accelerator](#).

8.2. How to use OpenCVA

OpenCVA leverages the DRP's processing capability to enhance specific functions of the OpenCV library. You can use OpenCVA same as OpenCV as usual and you do not need to consider of OpenCVA architecture.

OpenCVA is automatically executed by DRP as follows if it matches the conditions under which DRP can be used.

For the DRP using conditions, see [OpenCVA API specification and condition for using DRP](#).

OpenCVA can disable DRP, for each function. See [API functions to control OpenCVA](#) for details.

The following table lists the OpenCV functions that can be executed using DRP in the OpenCVA:

Table 10. OpenCV Functions Supported by DRP

OpenCV Function Name	Function
resize	Image resize.
cvtColor	Change color space.
cvtColorTwoPlane	Change color space.
GaussianBlur	Gaussian filter process.
dilate	Areas of bright regions grow.
erode	Areas of dark regions grow.
morphologyEX	Combination of dilate and erode.
filter2D	Image convolution..
Sobel	Extracting image edges.
adaptiveThreshold	Transforms a grayscale image to a binary image according to the formula.
matchTemplate	Compares a template against overlapped image regions.
wrapAffine	Transforms the source image using the 2x3 matrix.
wrapPerspective	Transforms the source image using the 3x3 matrix.
pyrDown	Downsampling step of the Gaussian pyramid construction.
pyrUp	Upsampling step of the Gaussian pyramid construction.
FAST	Detects corners using the FAST algorithm.
remap	Applies a generic geometrical transformation to an image.

On Ubuntu OS, OpenCVA library is installed in the following path:

NOTE /usr/lib/aarch64-linux-gnu/renesas/libopencv*
 /usr/include/opencv4/renesas/opencv4

9. RZ/V Multi-OS

The RZ/V2H RDK equipped with the following CPUs:

- Cortex-A55 (4 cores): Runs Linux OS
- Cortex-M33 (1 core): Runs MCU based OS such as FreeRTOS, Zephyr, or BareMetal
- Cortex-R8 (2 cores): Runs MCU based OS such as FreeRTOS, Zephyr, or BareMetal

This package provides the necessary software components and examples to enable and manage multiple operating systems running concurrently on the RZ/V2H platform.

9.1. Overview

The RZ/V Multi-OS feature allows the RZ/V2H platform to run multiple operating systems concurrently.

This capability is particularly useful for applications that require separation of tasks, such as running a real-time operating system (RTOS) alongside a general-purpose OS like Linux

On our default IPL, the following features are enabled by default:

IMPORTANT

- Remoteproc support
- CM33 and CR8 invocation from U-Boot

WARNING

When using the J-Link debugger, ensure that the SW1-6 switch on the RZ/V2H RDK board is set to the "ON" position to enable JTAG debug functionality.

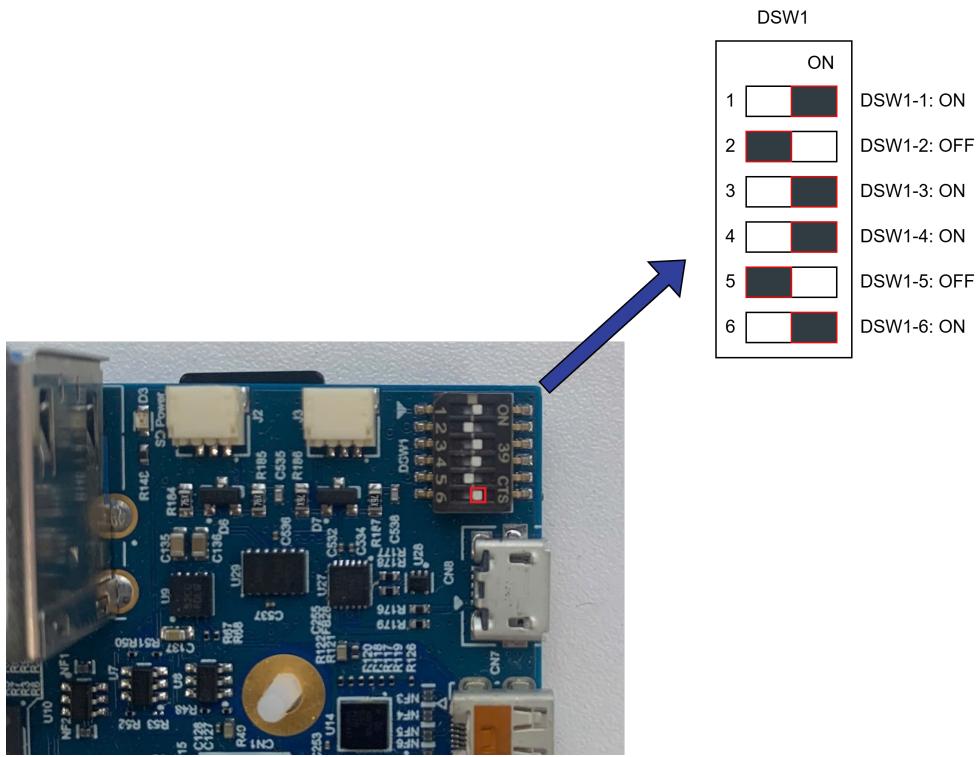


Figure 14. RZ/V2H RDKit JTAG Switch 6 ON

9.2. Useful References

The following documents and repositories are recommended for users who want to explore more about multi-OS integration and RZ/V FSP software development:

- [RZ/V Multi-OS Package v3.2.0](#): Official software package enabling Linux + FreeRTOS/BareMetal multi-OS solutions on RZ/V2H.
- [RZ/V2H Quick Start Guide for RZ/V Multi-OS Package](#): Step-by-step guide to integrate and configure Multi-OS environments.
- [Getting Started with RZ/V Flexible Software Package \(FSP\)](#): Instructions for developing and managing applications using RZ/V FSP.
- [FSP Example Project Usage Guide](#): Detailed examples demonstrating peripheral control, communication, and middleware setup.
- [micro_ros_renesas_demos](#): Demonstrations and documentation for using Micro-ROS with Renesas platforms, including agent setup and communication examples.

We recommend reviewing these resources to gain a deeper understanding of multi-OS capabilities and how to effectively utilize them in your projects.

9.3. Requirement

- Ubuntu or Windows machines that can install the e2Studio and Flexible Software Package (FSP)

for the Renesas RZ/V MPU Series.

- Segger J-Link (**firmware version 7.96e**): JTAG debugger for flashing firmware and debugging.
- [RZ/V FSP version 3.1](#) installed on the development machine. Follow the instructions in the [RZ/V FSP Getting Started Guide](#) to set up the FSP.

9.4. Note for integration

The peripherals which are NOT enabled enter Module Standby Mode after Linux kernel is booted up. That means the peripherals used on CM33 side might stop working at that time.

To avoid such a situation, Multi-OS Package incorporates the patch below:

- [0003-Set-OSTM-for-MCPU-and-RCPU.patch](#)

This patch prevents GTM used in RPMsg demo program from entering Module Standby Mode. If you have any other peripherals which you would like to stop entering Module Standby implicitly, please update the patch as shown below:

```
diff --git a/drivers/clk/renesas/r9a09g057-cpg.c b/drivers/clk/renesas/r9a09g057-cpg.c
index 0468eb17305b..146d5b444e8c 100644
--- a/drivers/clk/renesas/r9a09g057-cpg.c
+++ b/drivers/clk/renesas/r9a09g057-cpg.c
@@ -302,14 +302,14 @@ static const struct rзв2h_mod_clk r9a09g057_mod_clks[]
 __initconst = {
     DEF_MOD("rcpu_cmtw1_clkm", CLK_PLLCLN_DIV32, 4, 0, 2, 0),
     DEF_MOD("rcpu_cmtw2_clkm", CLK_PLLCLN_DIV32, 4, 1, 2, 1),
     DEF_MOD("rcpu_cmtw3_clkm", CLK_PLLCLN_DIV32, 4, 2, 2, 2),
-    DEF_MOD("gtm_0_pclk", CLK_PLLCM33_DIV16, 4, 3, 2, 3),
-    DEF_MOD("gtm_1_pclk", CLK_PLLCM33_DIV16, 4, 4, 2, 4),
+    DEF_MOD_CRITICAL("gtm_0_pclk", CLK_PLLCM33_DIV16, 4, 3, 2, 3),
+    DEF_MOD_CRITICAL("gtm_1_pclk", CLK_PLLCM33_DIV16, 4, 4, 2, 4),
     DEF_MOD("gtm_2_pclk", CLK_PLLCLN_DIV16, 4, 5, 2, 5),
     DEF_MOD("gtm_3_pclk", CLK_PLLCLN_DIV16, 4, 6, 2, 6),
-    DEF_MOD("gtm_4_pclk", CLK_PLLCLN_DIV16, 4, 7, 2, 7),
-    DEF_MOD("gtm_5_pclk", CLK_PLLCLN_DIV16, 4, 8, 2, 8),
-    DEF_MOD("gtm_6_pclk", CLK_PLLCLN_DIV16, 4, 9, 2, 9),
-    DEF_MOD("gtm_7_pclk", CLK_PLLCLN_DIV16, 4, 10, 2, 10),
+    DEF_MOD_CRITICAL("gtm_4_pclk", CLK_PLLCLN_DIV16, 4, 7, 2, 7),
+    DEF_MOD_CRITICAL("gtm_5_pclk", CLK_PLLCLN_DIV16, 4, 8, 2, 8),
+    DEF_MOD_CRITICAL("gtm_6_pclk", CLK_PLLCLN_DIV16, 4, 9, 2, 9),
+    DEF_MOD_CRITICAL("gtm_7_pclk", CLK_PLLCLN_DIV16, 4, 10, 2, 10),
     DEF_MOD("wdt_0_clkp", CLK_PLLCM33_DIV16, 4, 11, 2, 11),
     DEF_MOD("wdt_0_clk_loco", CLK_QEXTAL, 4, 12, 2, 12),
     DEF_MOD("wdt_1_clkp", CLK_PLLCLN_DIV16, 4, 13, 2, 13),
```

--

Please edit the `r9a09g057-cpg.c` file accordingly and re-build the Linux Image to ensure the desired peripherals remain active during Multi-OS operation.

We recommend using the eSDK to build the Linux image to do it. For more details, please refer to the [Custom Linux Kernel and Device Tree](#).

9.5. How to create the new project for RZ/V2H RDK

To create a new project for the RZ/V2H RDK board using the RZ/V Flexible Software Package (FSP) and e2Studio, follow these steps:

1. Open e2Studio and create a new Renesas FSP project.
2. Choose **File > New > Renesas C/C Project*** > ***Renesas RZ*** > ***Renesas RZ/V C/C FSP Project**.
3. Enter the Project name and location.
4. Select the **Custom User Board (for RZ/V2H)**, the Device **R9A09G057H44GBG**, and the target core: **CM33 or CR8**.
5. If you select the CR8 core, please select the appropriate **Preceding Project/Smart Bundle Details**.

This setup is necessary to ensure that there are no conflicts in hardware resource usage between the CM33 and CR8 cores when both are running on the RZ/V2H platform.

6. Configure the **Build artifact type, RTOS selection and Sub-core selection** as needed.
7. Configure the **Project Template Selection**, and click on the **Finish** button to create the project.
8. Once the project is created, you can start adding your application code and configuring the necessary peripherals using the FSP.

We also provide a sample project for RZ/V2H RDK board with Multi-OS Package.

Please follow the next section to import the sample project.

9.6. RZV2H RDK Multi-OS Example Packages

This section contains a collection of Multi-OS packages designed for applications on Renesas RZ/V MPU platforms, specifically targeting the RZ/V2H RDK.

These packages provide practical examples demonstrating how to operate and integrate Multi-OS environments on the RZ/V2H RDK, helping developers understand cross-core communication, system setup, and interaction between Linux and RTOS components.

Additionally, a demo showcasing Micro-ROS (uROS) running on the real-time CR8 core is supported. It demonstrates the implementation of Micro-ROS on an MCU-class core within the device.

9.6.1. Hardware supported

- Platform: Renesas RZ/V2H MPU
- Development Board: RZ/V2H RDK (SoC: R9A09G057H44GBG)

9.6.2. Software supported

- Target Operating System: Ubuntu 24.04
- RZ/V Multi-OS Package version 3.2
- RZ/V FSP version 3.1
- Micro XRCE-DDS Agent version 3.0.1
- Micro ROS Client Jazzy
- ROS2 Distribution: ROS2 Jazzy

9.6.3. Package Specification

Package	Target Core	Purpose / Description
Micro XRCE-DDS Agent	CA55 (Linux)	Provides the middleware agent running on the Linux core (CA55) for communication between Micro-ROS clients (running on RTOS CR8_0 core) and the ROS 2 environment on Linux via the XRCE-DDS protocol.
RZ/V2H RDK Blinky	CM33 (RTOS)	A simple LED blinking demo running on the CM33 core that verifies basic GPIO functionality and confirms that the RTOS environment is running correctly on the RZ/V2H RDK.
RZ/V2H RDK CM33 RPMsg Linux-RTOS Demo	CM33 (RTOS)	Demonstrates inter-core communication (RPMsg) between the Linux core (CA55) and the CM33 RTOS core, showing message exchange and synchronization.
RZ/V2H RDK CR8 Core0 RPMsg Linux-RTOS Demo	CR8_0 (RTOS)	Demonstrates RPMsg-based communication between the Linux core (CA55) and the CR8_0 real-time core, validating message passing and core coordination.

Package	Target Core	Purpose / Description
RZ/V2H RDK CR8 Core0	CR8_0 (RTOS)	Showcases Micro-ROS running on the CR8_0 real-time core, integrating the uROS client
RPMsg Demo	Micro-ROS	with the custom RPMsg transport layer for communication with Linux and ROS 2.

9.6.4. Installation Guide

To set up and use the Multi-OS example packages on the RZ/V2H RDK, follow the steps below:

9.6.4.1. Firmware Code for CM33/CR8

This section describes how to build and flash the firmware for the CM33/CR8 core using e² studio and the provided sample project.

1. Clone the CM33/CR8 project into your host machine.
2. Open **e² studio** and import the above project using "**Import Existing Project**".
3. Open the configuration file.
4. Click "**Generate**" to generate configuration files.
5. Click "**Build Project**" and wait for the build process to complete.
6. Flash the firmware to the CM33/CR8 core using your preferred method (e.g., J-Link).

IMPORTANT

The preceding project for all of CR8_0 packages is [RZ/V2H RDK CM33 RPMsg Linux-RTOS Demo](#). Please import this CM33 project into your e² studio workspace and build it before using the CR8_0 packages.

Special Note for [RZ/V2H RDK CR8 Core0 RPMsg Micro-ROS Demo Package](#)

1. This demo includes a pre-built libmicroros library. If you want to rebuild this library, use this project on the Ubuntu machine and perform the following steps:

Go to **Project** → **Properties** → **C/C++ Build** → **Settings** → **Build Steps** tab and in **Pre-build steps**, add the command:

```
cd ../../micro_ros_renesas2estudio_component/library_generation &&
./library_generation.sh "${cross_toolchain_flags}"
```

Then click **Apply and Close** → **Build the project**.

2. The code includes a 30-second delay (in [main_task_entry.c](#) line 168) before initializing MCU tasks to prevent issues with PWM and I2C pin control on the CR8 core. By default, this delay is

commented out.

- If flashing via **J-Link**, this delay can be skipped.
- However, when invoking the firmware from **U-Boot**, please enable this delay.

9.6.4.2. Cross Compile the Micro XRCE-DDS Agent

This section describes how to deploy and build the custom OpenAMP Micro-ROS agent application running on the CA55 core of the RZ/V2H platform.

Prerequisites: The Linux local host machine must have the Yocto SDK installed. It is recommended to use the [ROS2 cross-build Docker container](#), since the environment is already fully set up.

1. Clone the [Micro-XRCE-DDS-Agent](#) to your local machine.
2. The custom OpenAMP agent source code is located at: [Micro-XRCE-DDS-Agent/examples/custom_agent](#).
3. Navigate to the directory [Micro-XRCE-DDS-Agent/](#).
4. Build the project:

```
mkdir build && cd build

# We have to disable the UAGENT_LOGGER_PROFILE because the current SDK cannot
# compile spdlog.
# For native builds, we can remove the DUAGENT_LOGGER_PROFILE flag.
cmake .. -DCMAKE_TOOLCHAIN_FILE=../coretexa55-toolchain.cmake \
    -DUAGENT_BUILD_USAGE_EXAMPLES=ON \
    -DUAGENT_LOGGER_PROFILE=OFF \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_INSTALL_PREFIX=../arm64-install

make -j$(nproc)
make install
```

5. Wait until the build process completes.
6. Deploy the output to the target board by copying the output artifact:

- On the **Host machine**:

```
# Copy the built CustomXRCEAgent binary to the arm64-install folder for
# deployment
$ cp ./examples/custom_agent/CustomXRCEAgent arm64-install/bin

# Compress the arm64-install folder
```

```
$ tar -cf libdds_agent.tar.bz2 -C arm64-install .
```

- Then copy `libdds_agent.tar.bz2` to the target board using `scp` or another file transfer method.
- On the **Target machine**:

```
# Extract the archive
$ mkdir tmp-install
$ sudo tar -xf libdds_agent.tar.bz2 -C tmp-install

# Install libdds_agent to the system
$ cd tmp-install
$ sudo cp -r * /usr/local/
$ sudo ldconfig
```

9.6.5. Usage Guide

To run the Multi-OS example packages on the RZ/V2H RDK, follow the instructions below for each package.

9.6.5.1. RPMsg Linux-RTOS Demo

This demo behaves identically to the version released in the **RZ/V Multi-OS Package**. For more details, refer to the [RZ/V2H Quick Start Guide](#) for the RZ/V Multi-OS Package.

1. Flash the **RPMsg Linux-RTOS Demo** firmware to the target board.
2. On the board's terminal, run the `rpmsg_sample_client` with sudo privilege:

```
root@localhost:~# rpmsg_sample_client
```

Example output:

```
root@localhost:~# rpmsg_sample_client
[694] proc_id:0 rsc_id:0 mbx_id:1
metal: warning: metal_linux_irq_handling: Failed to set scheduler: -1.
metal: info: metal_uio_dev_open: No IRQ for device 10480000.mbox-uio.
[694] Successfully probed IPI device
...
```

3. Based on the firmware you have flashed, select the corresponding option below and press **Enter** when prompted.

Input Option	Target Core / Firmware	Description
1	CM33 (Linux <⇒ RTOS RPMsg Demo)	Select this if you have flashed the RZ/V2H RDK CM33 RPMsg Linux-RTOS Demo firmware.
4	CR8_0 (Linux <⇒ RTOS RPMsg Demo)	Select this if you have flashed the RZ/V2H RDK CR8 Core0 RPMsg Linux-RTOS Demo firmware.

NOTE Ensure that the firmware on your target board matches the selected option to avoid communication errors.

4. Example output:

- If the Input Option is 1:

```
[CM33] received payload number 469 of size 486
[CM33] sending payload number 470 of size 487
[828] cond signal 1 sync:0
...
...
```

- If the Input Option is 4:

```
[CR8_0 ] received payload number 469 of size 486
[CR8_0 ] sending payload number 470 of size 487
[790] cond signal 2 sync:0
...
...
```

5. By typing e, the sample program should terminate with the message shown below:

```
please input
> e
[xxx] 42f00000.rsctbl closed
[xxx] 43000000.vring-ctl0 closed
...
...
```

9.6.5.2. uROS and Custom Micro XRCE-DDS Agent

This section describes how to run the Micro-ROS Client demo and the custom XRCE-DDS RPMsg Agent.

1. (Optional) Connect the UART-to-TTL cable to **GPIO pin 40** on the RDK board to view log output from the CR8 core over UART channel 5 (P72–TXD5 / P73–RXD5).

2. Flash the [RZ/V2H RDK CR8 Core0 RPMsg Micro-ROS Demo](#) firmware to the target board.
3. On the board's terminal, run the CustomXRCEAgent with sudo privilege:

```
root@localhost:~# CustomXRCEAgent
```

Example output:

```
root@localhost:~# CustomXRCEAgent
[787] proc_id:0 rsc_id:0 mbx_id:1
metal: warning: metal_linux_irq_handling: Failed to set scheduler: -1.
...
```

4. On another terminal, use the following ROS 2 commands to verify communication:

```
source /opt/ros/jazzy/setup.bash
ros2 topic list
ros2 topic echo /cr8/heartbeat
```

Behavior:

- The CR8 firmware creates the topic [/cr8/heartbeat](#) and continuously publishes data to it.
- The custom Micro XRCE-DDS Agent makes this topic available in the ROS 2 environment running on the CA55 core.
- From the CA55 core, you can subscribe to and retrieve data from the [/cr8/heartbeat](#) topic.

Example output:

```
rz@localhost:~$ source /opt/ros/jazzy/setup.bash
rz@localhost:~$ ros2 topic list
/cr8/heartbeat
/parameter_events
/rosout
rz@localhost:~$ ros2 topic echo /cr8/heartbeat
data: 328
---
data: 329
---
data: 330
...
```

9.6.6. Troubleshooting

1. Can't open the configuration.xml of CR8 e² studio project?

Confirm the RZ/V FSP version is 3.1 and import the **CM33 project** into the workspace and build it first, then try opening the CR8 project again.

2. The behavior of the RPMsg demo is strange?

Restart the **RDK board** to reset the RPMsg endpoint.

3. Unknown status of the micro-ROS demo?

Use a **USB-to-TTL** module to read logs from the UART interface of the RDK board (baud rate: **115200**).

You should see output similar to the following:

```
[CR8] Start main_task_entry
[CR8] RPMsg endpoint ready
[CR8] Heartbeat publisher ready on /cr8/heartbeat
[CR8] Heartbeat #50 (uptime=16061 ms)
```

You should run the **CustomXRCEAgent** only after the message **[CR8] RPMsg endpoint ready** appears on the UART log.

4. Can't flash the firmware over J-Link?

Make sure you are using the correct **J-Link firmware version** and that **DIP switch SW1-6** is turned **ON**.

10. ROS2 Application Development with RZ/V2H RDK

This section covers the development of ROS2 applications using the Renesas RZ/V2H RDK board.

Target [ROS2 Jazzy](#) running on Ubuntu 24.04 LTS and RZ/V2H RDK.

When developing ROS2 applications for the Renesas RZ/V2H RDK platform, it is essential to set up the development environment correctly.

We provide a guide to help you get started with ROS2 development, including setting up the necessary tools, **cross-building applications**, and deploying them to the RZ/V2H RDK board.

The following topics are covered in this section:

- **Sample Applications:** Examples of ROS2 applications that can be built and executed on the RZ/V2H RDK.
- **Repositories and Packages:** Information about available RZ/V ROS2 packages.
- **ROS2 Development:** Instructions on the development workflow for ROS2 applications, including cross-build and deployment.
- **Other Concepts:** Additional concepts and advanced topics related to ROS2 development.
- **Appendix:** Supplementary information.

11. ROS2 Application Development

This section covers the development of ROS2 applications on the Renesas RZ/V2H RDK platform, including setup, cross-build, and deploy the application.

11.1. Cross-build the ROS2 Application

This section provides instructions on how to cross-build ROS2 applications for the Renesas RZ/V2H RDK platform.

What is Cross-building?

Cross-building is the process of compiling software on a host system to run on a different target system.

The advantages of cross-building is that it allows developers to build applications for embedded systems without needing to compile directly on the target device, which have limited resources causing longer build times.

In this case, we will be cross-building ROS2 applications on a development machine (host) to run on the Renesas RZ/V2H RDK board (target).

We provide two methods for cross-building ROS2 applications for the RZ/V2H RDK platform:

- **Cross-build with Yocto SDK:** Uses the Yocto-provided SDK for cross-compiling ROS 2 applications targeting the RZ/V2H platform.
- **Cross-build with QEMU Docker:** Utilizes a Docker container with QEMU emulation to create a cross-compilation environment for building ROS2 applications.

Table 11. Comparison Between Build Methods

Method	Advantages	Disadvantages
Yocto SDK	Very fast compilation.	Possible ABI version mismatch error , if package versions differ between Yocto SDK and Ubuntu.
QEMU Docker	No ABI mismatch , as the container matches the target runtime.	Slower build time due to QEMU emulation.
Native build	Simplest setup, directly compiles on the RZ/V2H target.	Very slow build time and requires sufficient storage and memory on the board.

Follow the respective sections for detailed instructions on each cross-build method.

For most use cases, the **Cross-build with Yocto SDK** method is recommended for its speed and efficiency.

TIP

But if you encounter [ABI mismatch errors](#), consider using the **Cross-build with QEMU Docker** method.

11.1.1. Cross-build with Yocto SDK

CAUTION Be aware of potential **ABI mismatch errors** when using the Yocto SDK method.

Refer to the [ABI mismatch appendix](#) for more details.

To cross-build ROS2 applications for the RZ/V2H RDK platform, we will use the Yocto SDK provided with the RZ/V2H RDK Linux image.

11.1.1.1. Prerequisites

Complete the [Common docker environment setup](#) step as described in the **System Configuration** section.

After this step, the Docker container is expected to be fully set up with the Yocto SDK installed.

Access the Docker container terminal:

```
$ docker exec -it [name_of_docker_container] bash
```

Inside the container, go to your ROS2 workspace:

```
$ cd <path-to-your-ROS2-workspace>
```

\$ROS2_WS is the default workspace directory set by the Dockerfile.

```
$ cd $ROS2_WS
```

TIP

Environment variables set in the Docker container:

- **\$ROS2_WS**: Default ROS2 workspace directory.
- **\$TOOLCHAINS_WS**: Directory for cross-compilation toolchain files.

Copy cross.cmake file to your workspace:

```
$ cp $TOOLCHAINS_WS/cross.cmake $ROS2_WS/
```

NOTE

On the first time create the container, the default toolchain files are already installed to the **\$ROS2_WS**

11.1.1.2. Requirements

Before cross-building ROS 2 applications, ensure that the ROS 2 workspace is fully set up with all necessary dependency packages.

Refer to the [Common ROS2 Workspace structure](#) to gain some tips for setting up the ROS2 workspace for your development.

Make sure to include all required ROS2 packages in the **src/** directory of your workspace.

The workspace structure:

```
ros2_ws/
├── src/          # Source code for ROS 2 packages
│   ├── package_1/
│   ├── package_2/
│   └── ...
├── build/        # Build output directory (generated)
├── install/      # Installation directory (generated)
├── cross.cmake   # CMake toolchain file for cross-compilation
└── log/          # Log files (generated)
```

11.1.1.3. Cross-build Steps

To cross-build ROS2 applications for the RZ/V2H RDK platform, follow the steps below.

11.1.1.3.1. Using `cross-colcon-build`

Use the `cross-colcon-build` command to build ROS 2 packages for the RZ/V2H RDK.

The `cross-colcon-build` script is a wrapper around the standard `colcon build` command that performs the following steps:

- Checks if a `cross.cmake` toolchain file exists in the current working directory.
- Unsets the `LD_LIBRARY_PATH` variable to avoid conflicts with the cross-compilation environment.
- Loads the cross-compilation environment by sourcing the appropriate SDK setup script.
- Runs `colcon build` with the necessary CMake arguments for cross-compilation, along with any user-provided arguments.

The `cross-colcon-build` command functions similarly to the standard `colcon build` command, but with key differences tailored for cross-compilation.

Instead of building binaries for your current host architecture, it targets the **RZ/V2H (arm64)** platform using the configured **Yocto SDK** and **toolchain**.

11.1.1.3.2. Usage Guide

Simply replace `colcon build` with `cross-colcon-build` in your workflow.

You can pass any additional arguments supported by `colcon build` directly to `cross-colcon-build`.

Example:

```
$ cd <path-to-your-ROS2-workspace>
```

```
$ cross-colcon-build --packages-select rzv_demo_dexhand
```

This command ensures that all necessary environment variables and toolchain settings are correctly applied for cross-compiling your ROS 2 packages for the RZ/V2H platform.

For more details, refer to the `env.conf` file to understand exactly what `cross-colcon-build` is doing under the hood.

NOTE

Make sure your `ros2_ws` contain the `cross.cmake` file. To obtain the default `cross.cmake` file, copy it from the `$TOOLCHAIN_WS` directory.

11.1.1.4. Deployment

After successfully cross-building the ROS2 applications, deploy the built packages to the RZ/V2H RDK target device.

Copy the contents of the `install/` directory to the target device using the `scp` command or using the **ROS2: Deploy to Target** feature in VS Code workspace.

Refer to the [ROS2 Deployment](#) section for detailed instructions on deploying the applications to the target device.

After deployment, please install any additional dependencies on the target device using:

```
$ source /opt/ros/jazzy/setup.bash
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

11.1.1.5. Known issue

During cross-build, you will see:

```
WARNING:colcon.colcon_ros.prefix_path.ament:The path
'/opt/poky/5.1.4/sysroots/cortexa55-poky-linux/opt/ros/jazzy' in the environment
variable AMENT_PREFIX_PATH doesn't contain any 'local_setup.*' files.
```

The warning message during cross-build is expected and does not indicate a problem with your setup. It results from intentionally removing ROS2 setup files from the SDK sysroot to prevent hard code paths that would break execution on the target board.

As long as your application runs correctly on the RZ/V2H device, you can safely ignore this warning.

Without this, this error will be occur on the target board:

```
rz@localhost:/home/rz# source install/setup.bash
not found: "/opt/poky/5.1.4/sysroots/cortexa55-poky-
linux/opt/ros/jazzy/local_setup.bash"
rz@localhost:/home/rz#
```

11.1.2. Cross-build with Docker and QEMU

This method currently supports cross-compiling ROS2 applications for the RZ/V2H platform using Docker Buildx.

The provided Dockerfile and scripts have been tested with the RZ/V2H AI SDK and are intended for development and evaluation purposes.

Supported: ROS2 Jazzy cross-compilation for arm64 (RZ/V2H) using the SDK and Docker Buildx.

NOTE

Limitations: Only RZ/V2H (arm64) builds are supported. Additional ROS2 packages or custom dependencies may need to be installed manually via:

```
$ sudo apt install ros-<distro>-<package-name>
```

11.1.2.1. Prerequisites

1. Install Docker. Refer to the [Docker Official Installation Guide](#).

Ensure that Docker is installed on your host machine with Buildx and QEMU support for cross-platform builds.

2. Clone the X Compilation QEMU Docker repository to your Host PC.

```
$ git clone
https://partnergitlab.renesas.solutions/sst1/industrial/ws078/utility/x_compilation
_qemu_docker.git
```

3. RZ/V2H RDK Yocto SDK installer.

Obtain the installer:

[poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh](#)

4. Copy the Yocto SDK installer to the Docker build context directory. Please replace the paths

below with your actual file locations.

```
$ cp poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh ~/x_compilation_qemu_docker/
```

After completing the prerequisites, the project structure should be:

```
x_compilation_qemu_docker/
├── Dockerfile
├── setup_docker_builx.sh
├── poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh
└── README.md
```

11.1.2.2. Build the Docker Image

Run `./setup_docker_builx.sh <path_to_ros2_workspace> [name_of_docker_container]` to set up the environment:

```
$ cd x_compilation_qemu_docker/
$ ./setup_docker_builx.sh <path_to_ros2_workspace> [name_of_docker_container]
```

- `<path_to_ros2_workspace>`: Path to your ROS 2 workspace on your machine. It will be mounted inside the Docker container.
- `[name_of_docker_container]`: (Optional) Name for the Docker container. Default is `rzv2h_ros_sdk`.

To start working inside the Docker container, use:

```
$ docker exec -it [name_of_docker_container] bash
```

11.1.2.3. Requirements

Before cross-building ROS 2 applications, ensure that the ROS 2 workspace is fully set up with all necessary dependency packages.

Refer to the [Common ROS2 Workspace structure](#) to gain some tips for setting up the ROS2 workspace for your development.

Make sure to include all required ROS2 packages in the `src/` directory of your workspace.

The workspace structure:

```

ros2_ws/
├── src/          # Source code for ROS 2 packages
│   ├── package_1/
│   ├── package_2/
│   └── ...
└── build/        # Build output directory (generated)
└── install/      # Installation directory (generated)
└── log/          # Log files (generated)

```

11.1.2.4. Cross-build Steps

Inside the Docker container, navigate to your ROS2 workspace:

```
$ cd <path-to-your-ROS2-workspace>
```

Build the ROS2 packages using the **colcon build** command:

```
$ colcon build
```

NOTE

For more details on colcon to build the ROS2 applications, refer to the [Using colcon to build packages](#).

11.1.2.5. Deployment

After successfully cross-building the ROS2 applications, deploy the built packages to the RZ/V2H RDK target device by copying the contents of the **install/** directory to the target.

```
$ scp -r <path/to>/ros2_ws/install/* rz@<RZ/V2H-RDK-IP-address>:/path/to/deployment/directory/
```

Install any additional dependencies on the target device using:

```
$ source /opt/ros/jazzy/setup.bash
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

11.2. Useful tool for development of ROS2 applications

This section provides instructions on deploying, debugging ROS2 applications to the Renesas RZ/V2H RDK platform.

WARNING This method support the cross-build method using Yocto SDK only.

11.2.1. Prerequisites

1. Complete the [Common docker environment setup](#) step as described in the **System Configuration** section.
2. Also, complete the [Cross-build the ROS2 Application using Yocto SDK](#) step as described in the **ROS2 Development** section.
3. The following structure is expected after cross-building the ROS2 application:

```
ros2_ws/
├── build
├── .clang-format # Clang format configuration file
├── cross.cmake
├── install       # Installation directory (generated by cross-colcon-build
command)
├── log
├── src
└── .vscode       # VS Code workspace configuration files
```

In case your workspace does not contain the `.vscode/` directory, you can copy it manually from the `$TOOLCHAIN_WS` directory.

4. Ensure your development machine has the following:

- Visual Studio Code (VS Code)
- Remote Development extension for VS Code
- C/C++ extension installed inside the development container
 - Your **development machine** can establish an SSH connection to the target board.
 - Please establish the first SSH connection using the `ssh` command to accept the host key.

WARNING

```
$ ssh rz@<target_ip_address>
```

- Your **target board** has both gdbserver and rsync installed. If not, you can install it using:

```
$ sudo apt install gdbserver rsync
```

11.2.2. VS Code Workspace Configuration

With the VS Code workspace configuration, you can easily deploy and remote debug your ROS2 applications on the RZ/V2H RDK platform.

VS Code configuration files are located in the `.vscode/` directory of your ROS2 workspace:

```
.vscode/
├── c_cpp_properties.json      # C/C++ extension configuration file
├── deploy.sh                 # Deployment script to copy files to target
├── launch.json               # Debugger launch configuration file
├── run_program.sh            # Script to run the program on target
├── settings.json              # VS Code workspace settings file
├── settings.linux.json        # Linux-specific settings overrides
├── start_target_gdbserver.sh  # Script to start gdbserver on target
└── tasks.json                 # Task runner configuration file
```

By default, the configuration files are set up by the Dockerfile during the Docker container creation.

11.2.3. Supported Features

The VS Code workspace configuration supports the following features:

Target for ROS2 Jazzy Remote Run/Debug for arm64 (RZ/V2H) with cross-build using Yocto SDK method.

11.2.3.1. VS Code Tasks

- **ROS2: SSH to Target** - Start an SSH session to the target board.
- **ROS2: Debug Run (GDB)** - Start remote debugging a ROS2 node using the `ros2 run` command with the GDB server.
- **ROS2: Debug Launch (GDB)** - Start remote debugging a ROS2 node within a launch file using the `ros2 launch` command with the GDB server.
- **ROS2: Run Package Executable** - Run a ROS2 node using the `ros2 run` command.
- **ROS2: Launch Package LaunchFile** - Run ROS2 nodes using the `ros2 launch` command.
- **ROS2: Deploy to Target** - Deploy the `install` folder to the target board.
- **ROS2: Build Debug** - Build the workspace with the *Debug* configuration.
- **ROS2: Build Release** - Build the workspace with the *Release* configuration.
- **ROS2: Clean All** - Clean all build artifacts (`build`, `install`, `log` folders).

11.2.3.2. VS Code Launch Configurations

- **GDB for ROS2 Run** - Configure debugging in VS Code compatible with ROS2 *Run* debugging using GDB (`ros2 run`).
- **GDB for ROS2 Launch** - Configure debugging in VS Code compatible with ROS2 *Launch* debugging using GDB (`ros2 launch`).

NOTE All tasks support running with custom arguments.

A prompt will be displayed for argument input on each execution.

TIP You can use the hot key `Ctrl+Shift+P` in VS Code and search for **Tasks: Run Tasks** to execute the tasks listed above.

11.2.3.3. Limitations:

- Only RZ/V2H (arm64) builds are supported.
- Only support debug one ROS node at a time (both `ros2 run` and `ros2 launch`).
- The speed can be slow if the install folder is large or the network connection is poor.

11.2.4. Prepare for Deployment

1. Use the `Remote Development` plugin for VS Code to start working inside the development Docker container.
2. Prepare the workspace following the [VS Code Workspace Configuration](#).
3. Modify the contents of `c_cpp_properties.json` in your workspace. For example, modify the `includePath` or related configuration in `settings.json` to match your workspace setup. This file is provided as a reference example only.
4. Open the `settings.json` file, then edit the following variables to match your development environment:

WARNING The variables listed below are essential for the remote debugging and deployment workflow.

Table 12. Debug Configuration Variables

Variable Name	Required	Description	<code>ros2 run</code> required	<code>ros2 launch</code> required
<code>HOST_GDB_PATH</code>	Optional	Path to the GDB executable on the host machine.		

Variable Name	Required	Description	ros2 run required	ros2 launch required
TARGET_LOCAL_SYSROOT	Optional	Path to the sysroot of the target device SDK (used for debugging).		
TARGET_IP	Yes	IP address of the target device.	Yes	Yes
TARGET_GDB_PORT	Optional	GDB server port used on the target device.		
TARGET_USER	Optional	Username used to connect to the target device.		
TARGET_PASSWORD	Optional	Password used to authenticate the target connection.		
TARGET_ROS2_WS	Optional	Target ROS2 workspace path (used for deployment).		
NODE_PACKAGE_NAME	Yes	The ROS 2 package name that contains the executable.	Yes	For debug only
NODE_EXECUTABLE_NAME	Yes	The name of the executable to be launched.	Yes	For debug only
LAUNCH_PACKAGE_NAME	Yes	The ROS 2 package name that contains the launch file.	No	Yes
LAUNCH_FILE_NAME	Yes	The launch file used to start the ROS 2 application.	No	Yes

NOTE

- Variables marked as *Optional* should be modified only if your development environment differs from the default one generated by the [Dockerfile](#).
- Variables marked as **Required** must be set correctly each time you run a remote debug session; otherwise, the workflow will not operate properly.

When using this configuration, it will mimic the following commands on the target board:

```
ros2 run <NODE_PACKAGE_NAME> <NODE_EXECUTABLE_NAME>
```

```
ros2 launch <LAUNCH_PACKAGE_NAME> <LAUNCH_FILE_NAME>
```

11.2.5. Start Deployment

Run the **ROS2: Deploy to Target** task in VS Code to deploy the [install](#) folder to the target board.

After deployment, please install any additional dependencies on the target device using:

```
$ source /opt/ros/jazzy/setup.bash
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

11.2.6. Start Running / Remote Debugging

1. Update the variables `NODE_PACKAGE_NAME`, `NODE_EXECUTABLE_NAME`, `LAUNCH_PACKAGE_NAME` and `LAUNCH_FILE_NAME` in the `settings.json` file to match the node you want to debug.
2. Build your workspace using the *Debug* build type (required for debugging):

```
cross-colcon-build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

3. Use **Run Task** or **Launch** to start your workflow.
4. Enter any custom arguments if prompted.
5. Begin debugging or running the application.

11.2.7. Example: Workflow

Debugging `foxglove_keypoint_publisher_node` in `demo_virtual_inspire_rh56_hands.launch.py` from the RZ/V ROS Package

- In `settings.json`, set the following variables:

```
{
  "NODE_PACKAGE_NAME": "foxglove_keypoint_publisher",
  "NODE_EXECUTABLE_NAME": "foxglove_keypoint_publisher_node",
  "LAUNCH_PACKAGE_NAME": "rzv_demo_dexhand",
  "LAUNCH_FILE_NAME": "demo_virtual_inspire_rh56_hands.launch.py"
}
```

- Build the workspace with *Debug* configuration using the VS Code task **ROS2: Build Debug** or the command:

```
cross-colcon-build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

- In VS Code, press **Ctrl+Shift+D** and select **GDB for ROS2 Launch**.

- Enter the custom arguments when prompted:

```
video_device:=/dev/video0
```

- Press **Enter** to start the debugging session. The following command will be executed on the target board:

```
$ ros2 launch --launch-prefix 'gdbserver localhost:2345' --launch-prefix-filter
  'foxglove_keypoint_publisher_node' rzv_demo_dexhand
  demo_virtual_inspire_rh56_hands.launch.py
```

When debugging in **ros2 launch** mode:

NOTE

- Set **NODE_PACKAGE_NAME** to the package containing the target executable.
- Set **NODE_EXECUTABLE_NAME** to the name of the executable to be debugged.

11.2.8. Troubleshooting

- Can't deploy / remote run / remote debug:
 - Check the target IP, username, and password in the **settings.json** file.
 - Ensure the target board is powered on and connected to the network.
 - Verify that **gdbserver** and **rsync** are installed on the target board.
 - Establish the first SSH connection to the target board using the **ssh** command to accept the host key.
 - Delete the known hosts entry for the target IP in the **~/.ssh/known_hosts** file if the target board's host key has changed.
- Remote debug fails to start or disconnects immediately:
 - Ensure the workspace is built with the *Debug* configuration.
 - Verify that the correct package and executable names are set in the **settings.json** file.
 - Check network stability between the host and target board.
 - Make sure the GDB server port is not blocked by a firewall or not used by other process.

12. Other Concepts

High-level overview of important concepts relevant to RZ/V2H RDK ROS2 development.

12.1. Foxglove Visualization

Foxglove Studio is a popular framework to visualize ROS running applications similar to [Rviz](#).

It provides a web-based visualization client and a desktop app on both Window and Ubuntu machine.

Foxglove Setup

Use the following steps to visualize a ROS application in Foxglove.

1. Install Foxglove Studio on your Host PC

[Download the Foxglove Studio package](#) to install Foxglove Studio on your Host PC.

2. Start Foxglove Studio on your Host PC.
3. Create a new connection by clicking on the "Open connection".

Select "Foxglove WebSocket" as the connection type.

4. In the WebSocket URL field, enter the following URL to connect to a ROS2 application running on the RZ/V2H RDK board:

```
ws://<board-IP>:8765
```

Replace <board-IP> with the actual IP address of your RZ/V2H RDK board.

Make sure that the RZ/V2H RDK board is running a ROS2 application with the Foxglove WebSocket server node active, and your machine can reach the board over the network.

By default, the [Foxglove Bridge](#) runs on the RZ/V2H RDK.

It can also be deployed to any [ROS2 device on the same network](#). For example a laptop or another embedded device.

TIP

Keep in mind that running the Foxglove Bridge consumes some CPU resources on the device, so it is recommended to monitor system load when enabling it alongside other ROS2 nodes.

5. Click on the "Open" button to establish the connection.
6. Once connected, you can add various visualization panels to display data from your ROS2 application.
7. To add a panel, click on the "+" button in the top-right corner of the Foxglove Studio window and select the desired panel type (e.g., 3D View, Image, Plot, etc.).

Some tutorials also provide a pre-configured Foxglove layout in the form of a JSON file, which is typically located under `package_name/config/foxbglove`.

TIP To use those make sure you have the repository containing the tutorial cloned on your visualization machine.

Import the layout file by clicking on the **Import from file...** in the layout drop-down menu and choosing the layout file from the correct folder.

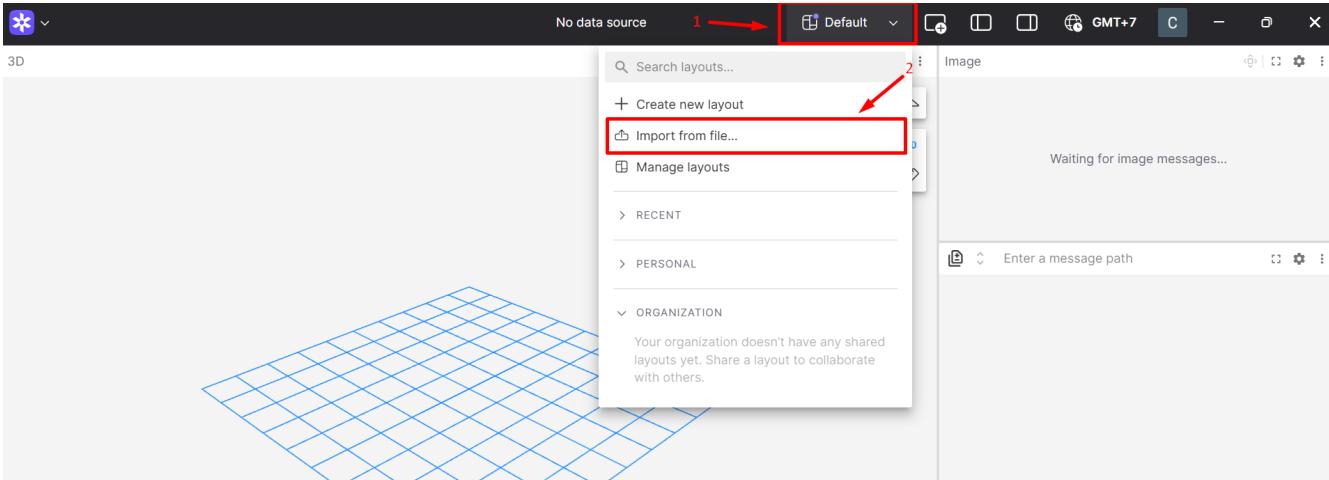


Figure 15. Foxglove Import Layout

12.2. MuJoCo Visualization

[MuJoCo Visualization](#) is a simulation platform designed for robotics development, particularly in the context of ROS2.

It provides a virtual environment where developers can test and validate their robotic applications without the need for physical hardware.

MuJoCo Setup

Use the following steps to visualize a ROS application in MuJoCo.

1. Install the ROS2 Jazzy on the host PC as per [ROS2 Jazzy installation guide](#).
2. Install MuJoCo and the necessary ROS2 MuJoCo packages on your Host PC.

The required packages are, please install them into your **Host PC's ROS2 Jazzy workspace**:

- [mujoco_sim_ros2](#)
- [mujoco_ros2_control](#)
- [mujoco](#)

Additional, some other ROS2 packages are also required on the **Host PC's ROS2 Jazzy workspace** in

order to run the MuJoCo simulation for the RZ/V Demo Arm Teleoperation application:

- [agilex_piper_arm_description](#)
- [agilex_piper_mujoco](#)
- [cartesian_controllers](#)

For other applications, please refer to the respective application documentation for any additional packages that may be required.

Those packages can be cloned into the `src/` directory of your [ROS2 Jazzy workspace](#) on the Host PC.

3. Build the ROS2 workspace on the Host PC.

```
$ cd <path-to-your-ROS2-Jazzy-workspace>
$ rosdep update
$ rosdep install --from-paths src --ignore-src -r -y
$ colcon build --symlink-install
```

4. Load the workspace environment:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path-to-your-ROS2-Jazzy-workspace>/install/setup.bash
```

5. Launch the MuJoCo simulation, for example the RZ/V Demo Arm Teleoperation application.

```
$ ros2 launch agilex_piper_mujoco
bringup_mujoco_cartesian_motion_controller.launch.py
```

The MuJoCo simulator window should open, displaying the robotic arm like below:

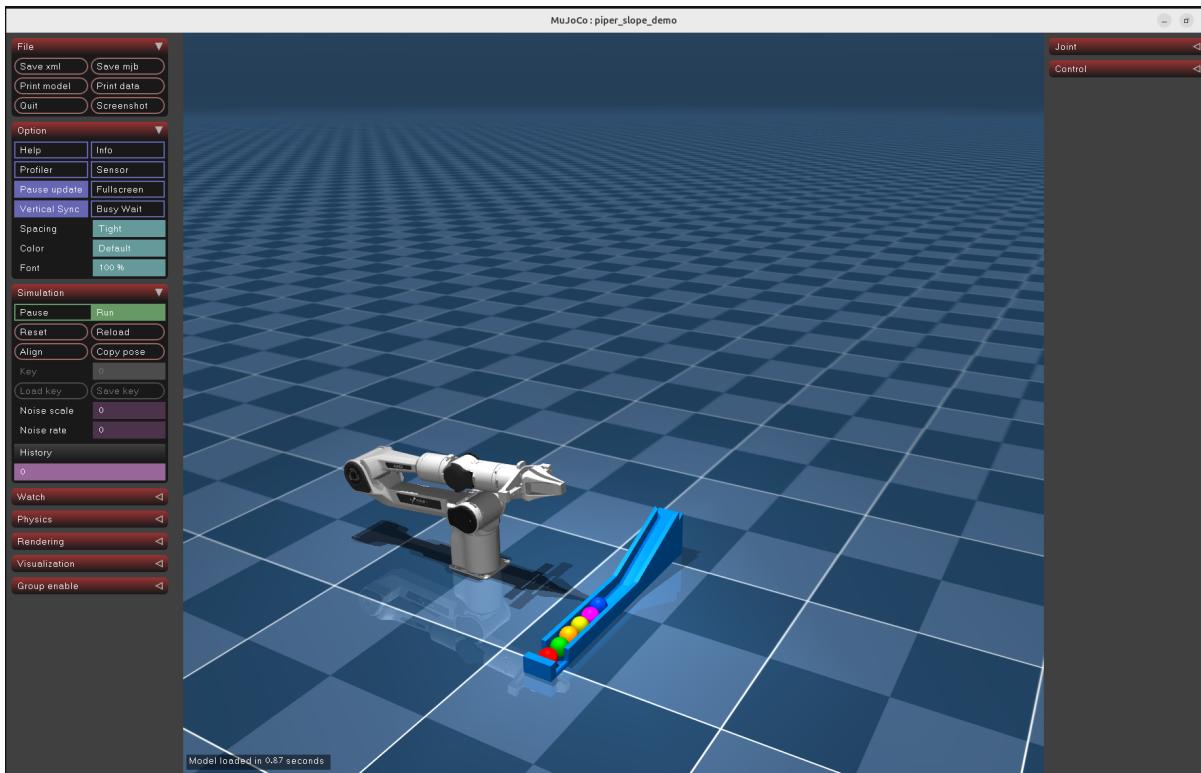


Figure 16. MuJoCo Simulation of RZ/V Demo Arm Teleoperation Application

12.3. Appendix

12.3.1. What is ABI mismatch error?

ABI (Application Binary Interface) mismatch occurs when the compiled application expects certain library versions or system interfaces that differ from those available on the target system.

It can lead to runtime errors or crashes when the application is executed on the target device.

In the context of cross-building ROS2 applications for the Renesas RZ/V2H RDK platform by using the Yocto SDK, an ABI mismatch error may arise if there are discrepancies between the versions of ROS2 packages and libraries used during the cross-compilation process (**installed in the SDK sysroot**) and those present on the target RZ/V2H RDK board (**installed over apt repository**).

Example output indicating ABI mismatch:

```
[ros2_control_node-3] Stack trace (most recent call last):
[ros2_control_node-3] #17  Object "/usr/lib/aarch64-linux-gnu/ld-linux-aarch64.so.1",
at 0xfffffffffffffff, in
[ros2_control_node-3] #16  Object
"/opt/ros/jazzy/lib/controller_manager/ros2_control_node", at 0xaaaad5ad5d6f, in
_start
[spawner-5] [INFO] [1756346496.296861080] [spawner_joint_state_broadcaster]: waiting
for service /controller_manager/list_controllers to become available...
```

```
[ros2_control_node-3] #15  Source "../csu/libc-start.c", line 360, in
__libc_start_main_impl [0xfffff9b068597]
[ros2_control_node-3] #14  Source "../sysdeps/nptl/libc_start_call_main.h", line 58,
in __aarch64_ldadd4_relax [0xfffff9b0684c3]
[ros2_control_node-3] #13  Object
"/opt/ros/jazzy/lib/controller_manager/ros2_control_node", at 0xaaaad5ad4e07, in main
[ros2_control_node-3] #12  Object "/opt/ros/jazzy/lib/librclcpp.so", at
0xfffff9b6f342b, in rclcpp::executors::MultiThreadedExecutor::spin()
[ros2_control_node-3] #11  Object "/opt/ros/jazzy/lib/librclcpp.so", at
0xfffff9b6f30b3, in rclcpp::executors::MultiThreadedExecutor::run(unsigned long)
[ros2_control_node-3] #10  Object "/opt/ros/jazzy/lib/librclcpp.so", at
0xfffff9b6dd8a3, in rclcpp::Executor::execute_any_executable(rclcpp::AnyExecutable&)
.....
[ros2_control_node-3] #0    Object "/opt/ros/jazzy/lib/libhardware_interface.so", at
0xfffff9af7e26c, in
[ros2_control_node-3] Bus error (Invalid address alignment [0x9])
```

12.3.2. How to avoid ABI mismatch error?

Using the [check_cross_build_versions.sh](#) script to check for version mismatches between SDK and target.

NOTE

Make sure to update the `PROJECT_ROOT` variable in the script to point to your ROS2 workspace `src` directory before running it.

If the script reports any mismatches, you may need to take one of the following actions:

- Try to run the application on the target system and see if it works despite the version mismatches.
- Rebuild the Yocto SDK with the correct versions of ROS2 packages that match those on the target system.
- Manual update the ROS2 packages on the SDK sysroot to match the target system versions. Refer to the next section for detailed steps.

12.3.3. How to update/add the Yocto SDK with the correct ROS2 package versions with eSDK?

The procedure below outlines the steps to update or add ROS2 packages in the Yocto SDK sysroot to resolve ABI mismatch issues or missing packages.

If you plan to update the version of an existing ROS2 package, please follow the steps under the **Optional** section. Otherwise, you can skip to the main steps.

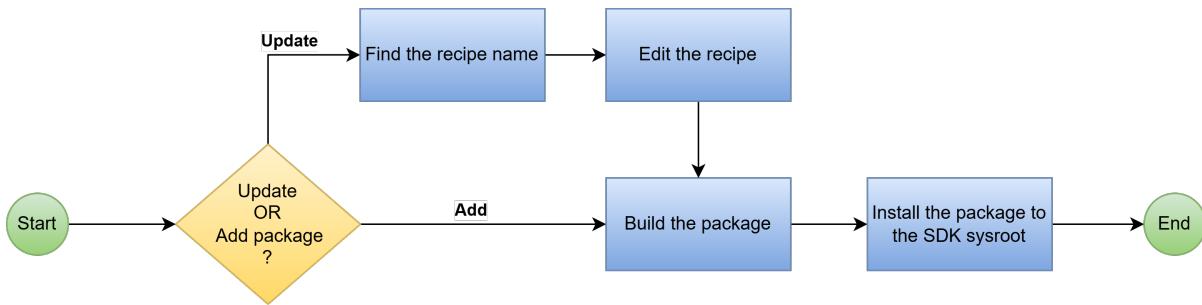


Figure 17. Updating/Adding ROS2 packages in Yocto SDK sysroot

NOTE

This section assumes that you have already set up the Yocto eSDK for RZ/V2H RDK as per the instructions in the [RZ/V2H eSDK Setup](#) section.

- Open a **new terminal** and source the Yocto eSDK environment:

```
$ source ~/poky_sdk/environment-setup-cortexa55-poky-linux
```

Please replace the path with the actual path to your eSDK working folder.

- **Optional:** If you want to update version of an existing ROS2 package in the SDK sysroot, please edit the correct recipe file in the [~/poky_sdk/layers/meta-ros/meta-ros2-jazzy](#).

For example, to update the **pinocchio** package:

1. Find the corresponding recipe file located at: [~/poky_sdk/layers/meta-ros/meta-ros2-jazzy](#)

```
ubuntu@ros-xbuild:~/poky_sdk/layers/meta-ros/meta-ros2-jazzy$ find . -name
"*\*pinocchio*"
./generated-recipes/pinocchio
./generated-recipes/pinocchio/pinocchio_3.6.0-1.bb
./generated-recipes/kinematics-interface-pinocchio
./generated-recipes/kinematics-interface-pinocchio/kinematics-interface-
pinocchio_0.0.1-1.bb
./recipes-bbappends/pinocchio
./recipes-bbappends/pinocchio/pinocchio_%.bbappend
ubuntu@ros-xbuild:~/poky_sdk/layers/meta-ros/meta-ros2-jazzy$
```

2. Edit the recipe file. For example, to update to version **3.8.0**, modify the relevant fields:

- Find the correct new **SRCREV** for newer version available at the package's official repository.

TIP

You can find the correct **SRCREV** by visiting the package's GitHub repository which is **SRC_URI** combine with **ROS_BRANCH**. Open the **pinocchio_3.6.0-1.bb** file and check the following fields:

```
# matches with: https://github.com/ros2-gbp/pinocchio-
release/archive/release/jazzy/pinocchio/3.6.0-1.tar.gz
ROS_BRANCH ?= "branch=release/jazzy/pinocchio"
SRC_URI = "git://github.com/ros2-gbp/pinocchio-
release;${ROS_BRANCH};protocol=https"
SRCREV = "38cb5592495c829a7e41c9b796ff9e617c797b27"
S = "${WORKDIR}/git"
```

The **SRCREV** is usually the commit hash or tag corresponding to the desired version. For example, for **pinocchio**, you can check its GitHub releases for ROS2 Jazzy: [Pinocchio Releases Jazzy](#).

And the correct **SRCREV** for version **3.8.0** is **89e88cb7ac670c41dd86f48e5b4ac4c697579811** which is the commit hash of the release tag: [release/jazzy/pinocchio/3.8.0-1](#).

- Update the **SRCREV** field in the recipe file - **pinocchio_3.6.0-1.bb** accordingly.
- Rename the recipe file to reflect the new version, e.g., **pinocchio_3.8.0-1.bb**. Also, update the name of bbappend files if necessary.
- Build the package.

```
$ devtool modify pinocchio
$ devtool build pinocchio
```

- Install the package into the SDK sysroot:

```
$ cd ~/poky_sdk/workspace/sources/pinocchio/oe-workdir/image/opt/ros/jazzy
$ sudo cp -r * /opt/poky/5.1.4/sysroots/cortexa55-poky-linux/opt/ros/jazzy/
```

12.3.4. Common ROS 2 topics

1. A typical ROS 2 workspace has the following structure:

```
ros2_ws/
├── src/                      # Source code for ROS 2 packages
│   ├── package_1/
│   ├── package_2/
│   └── ...
└── build/                     # Build output directory (generated)
    └── install/                # Installation directory (generated)
```

```
└── log/          # Log files (generated)
```

2. Code style should follow the official documentation: [Code style and language versions](#).
3. Organizing Files and Folders Inside a ROS 2 Package: [Package Organization For a ROS Stack \[Best Practices\]](#).

13. FAQs and Known Issues

This section contains frequently asked questions (FAQs) and known issues related to the RZ/V2H RDK.

13.1. Frequency Asked Questions

1. Does the RZ/V2H RDK support multiple operating systems?

Yes, the RZ/V2H RDK supports multiple operating systems running concurrently on its different CPU cores. For more details, refer to the [RZ/V Multi-OS](#) section.

2. How can I set up the USB-UART interface on the RZ/V2H RDK?

Refer to the [Other interfaces - USB-UART](#) section for detailed instructions on setting up the USB-UART interface for serial communication and debugging.

3. How do I configure a static IP address for high-speed interfaces on the RZ/V2H RDK?

Refer to the [Set Static IP](#) section for step-by-step instructions on configuring a static IP address.

4. I build the Ruiyan related ROS2 packages, but encounter the following error:

```
--- stderr: rh6_ctrl
CMake Error at CMakeLists.txt:47 (message):
library Path: /home/ubuntu/ros2_ws/src/ruiyan_rh2_controller/rh6_ctrl/lib
not found!

---
Failed  <<< rh6_ctrl [0.97s, exited with code 1]
```

Please make sure to copy the [libRyhandArm64.so](#) file to the SDK sysroot before cross-compilation as mentioned in the [Dexterous Hand Control Application](#) and [Rock Paper Scissors Application](#) sections.

5. Does the RZ/V2H RDK support Docker?

Yes, the RZ/V2H RDK supports Docker. You can install Docker on the RZ/V2H RDK by following steps:

```
$ sudo apt update
# Make sure you can access the internet
$ ping bing.com
# Install docker
$ curl -fsSL https://get.docker.com | sudo sh
```

```
# Add user to docker group
$ sudo usermod -aG docker $USER
# Logout and Login back to make it effects
# Simple check
$ docker --version
$ docker run hello-world
```

6. How to avoid ABI mismatch error when cross-compiling ROS2 applications?

Please refer to the [How to avoid ABI mismatch error?](#) section for detailed instructions.

7. How to update/add the Yocto SDK with the correct ROS2 package versions with eSDK?

Please refer to the [How to update/add the Yocto SDK with the correct ROS2 package versions with eSDK?](#) section for detailed instructions.

8. What USB-WIFI adapters are compatible with the RZ/V2H RDK?

The following USB-WIFI adapters have been tested and are compatible with the RZ/V2H RDK:

- Ralink Technology, Corp. MT7601U Wireless Adapter Linux firmware download link: [mt7601u.bin](#) Firmware path: [/lib/firmware/mt7601u.bin](#)
- AC1300 TP-Link T3U Nano Linux firmware download link: [rtw8822b_fw.bin](#) Firmware path: [/lib/firmware/rtw88/rtw8822b_fw.bin](#)

IMPORTANT

By default, no firmware is installed. Please install the correct **Linux firmware file** into the appropriate **firmware path** on the **RZ/V2H RDK root file system**, and then reboot the device to ensure proper driver support for these adapters.

If you have other USB-WIFI adapters, please refer to the manufacturer's documentation for compatibility and driver installation instructions. You can take the following steps as general guidance to check and install the required drivers and firmware for other USB-WIFI adapters:

- Enable the corresponding driver in the [Linux kernel configuration](#) (refer to [Custom Linux Kernel and Device Tree](#)). Recommend enable the driver as a module (m type) and copy the built module to the RZ/V2H RDK. For example, for the two adapters listed above, the following kernel configurations have already been enabled:

```
CONFIG_MT7601U
CONFIG_MT76x0U
CONFIG_RTL8XXXU
CONFIG_RTW88
CONFIG_RTW88_8822BU
```

- Build and install the driver module on the RZ/V2H RDK.
- Find the correct **firmware files** at: [Linux firmware repository](#) and place them in the appropriate **directory path** within the **RZ/V2H RDK root file system** to ensure proper device operation. Typically, firmware files are located in the `/lib/firmware/` directory.
- Reboot the RZ/V2H RDK and check if the USB-WIFI adapter is recognized and functioning properly.

TIP All firmware file can be found in the [Linux firmware repository](#). Please searching the repository with the USB-WIFI adapter model name or chip-set can help to locate the correct firmware file.

How to know which driver and firmware is needed for your USB-WIFI adapter?

- Connect the USB-WIFI adapter to the RZ/V2H RDK.
- Open a terminal and run the command: `lsusb` to list all connected USB devices.
- Identify your USB-WIFI adapter from the list and note its Vendor ID and Product ID.
- Use the Vendor ID and Product ID to search online for the specific driver and firmware required for your USB-WIFI adapter model.
- Refer to the manufacturer's website or Linux community forums for additional information on driver and firmware installation.
- Using `dmesg` command after plugging in the USB-WIFI adapter can also provide useful information about the required driver and firmware. For example, run the command: `dmesg | grep -i firmware` to check for any firmware-related messages.

```
rz@localhost:~$ dmesg | grep firmware
[ 17.946635] rtw_8822bu 3-1:1.0: Direct firmware load for rtw88/rtw8822b_fw.bin
failed with error -2
```

This log indicates that the `rtw8822b_fw.bin` firmware file is required at `/lib/firmware/rtw88/rtw8822b_fw.bin` for the adapter to function properly.

- Once you have identified the required driver and firmware, follow the steps mentioned above to install them on the RZ/V2H RDK.

9. Failed opening device /dev/video0 or /dev/tty: Permission denied

Please complete the First Time Boot Setup as described in the [First Time Boot Setup](#) section to add your user to the necessary groups (e.g., video, dialout) to gain access to these device files.

10. No space left on device error

Please complete the First Time Boot Setup as described in the [First Time Boot Setup](#) section to

expand the root filesystem to utilize the full storage capacity of the RZ/V2H RDK.

13.2. Know Issues

1. USB camera warning and error messages when running AI applications

The following warning and error messages may appear in the terminal when running AI applications that use USB cameras 2.0 on the RZ/V2H RDK:

```
[ 3495.792814] xhci-renesas-hcd 15860000.usb: WARN: HC couldn't access mem fast enough  
for slot 1 ep 2  
[ 3495.801878] xhci-renesas-hcd 15860000.usb: WARN: HC couldn't access mem fast enough  
for slot 1 ep 2  
[ 3495.810911] xhci-renesas-hcd 15860000.usb: ERROR Transfer event TRB DMA ptr not  
part of current TD ep_index 2 comp_code 13  
[ 3495.821917] xhci-renesas-hcd 15860000.usb: Looking for event-dma 000000004b0b1190  
trb-start 00000004b0b11a0 trb-end 00000004b0b11a0 seg-start 00000004b0b1000 seg-end  
00000004b0b1ff0  
[ 3496.032127] xhci-renesas-hcd 15860000.usb: WARN: HC couldn't access mem fast enough  
for slot 1 ep 2  
[ 3496.041226] xhci-renesas-hcd 15860000.usb: WARN: HC couldn't access mem fast enough  
for slot 1 ep 2  
[ 3496.299376] xhci-renesas-hcd 15860000.usb: WARN: HC couldn't access mem fast enough  
for slot 1 ep 2
```

We have observed that these warning and error messages do not impact the functionality of the USB camera or the performance of the AI applications.

The applications continue to operate as expected despite the presence of these messages.

There is no issue if you use the USB camera 3.0 on the RZ/V2H RDK.

We are actively investigating the root cause of these messages and working towards a resolution.

2. Some SD cards may not work properly with the RZ/V2H RDK

Some SD cards may not function correctly with the RZ/V2H RDK, leading to issues such as failure to boot or read/write errors.

To ensure compatibility, we recommend using **SD cards which accept the high-speed mode** and are from reputable brands, such as SanDisk, Samsung, or Kingston.

14. Revision History

Revision	Date	Description
1.00	Mar 31, 2026	Initial release.