RZ/V2H Robotic Development Kit User Manual

Release 0.1

Renesas Electronics Corporation

Oct 26, 2025

Contents

1 Getting Started

1.1 Overview

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

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

1.1.1 Software Environment

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

1.1.2 Hardware Environment

Items	Description

RZ/V2H	• **CPU**
	  – 4 × Arm Cortex-A55 (1.8GHz)
	  – 2 × Arm Cortex-R8 (800MHz)
	  – 1 × Arm Cortex-M33 (200MHz)
	• **DRP**
	  – Vision/Dynamically Reconfigurable Processor
	• **DRP-AI**
	  – Hardware AI Accelerator (8 dense TOPS, 80 sparse TOPS)
	• **Package**
	  – R9A09G057H44GBG: 1368-pin FCBGA

Memory	LPDDR4 1600MHz (8GB) × 2
SD Card	64GB SanDisk
QSPI Flash ROM	64MB
Interfaces	• DC Jack (12–24V / 2A) • JTAG (10-pin) • MIPI CSI-2 4-Lane ×2 (22-pin / 0.5mm) • HDMI • USB3.2 Type-A ×2 • USB Micro-B (SCIF) • 10/100/1000 Base-T RJ45 • Micro SD • PCIe 3.0 Root Complex (16-pin / 0.5mm) • CAN-FD ×2 • 40-pin RasPi GPIO Header

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

RZ/V2H RDK Board Image View:

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

1.2 Quick setup guide

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

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

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

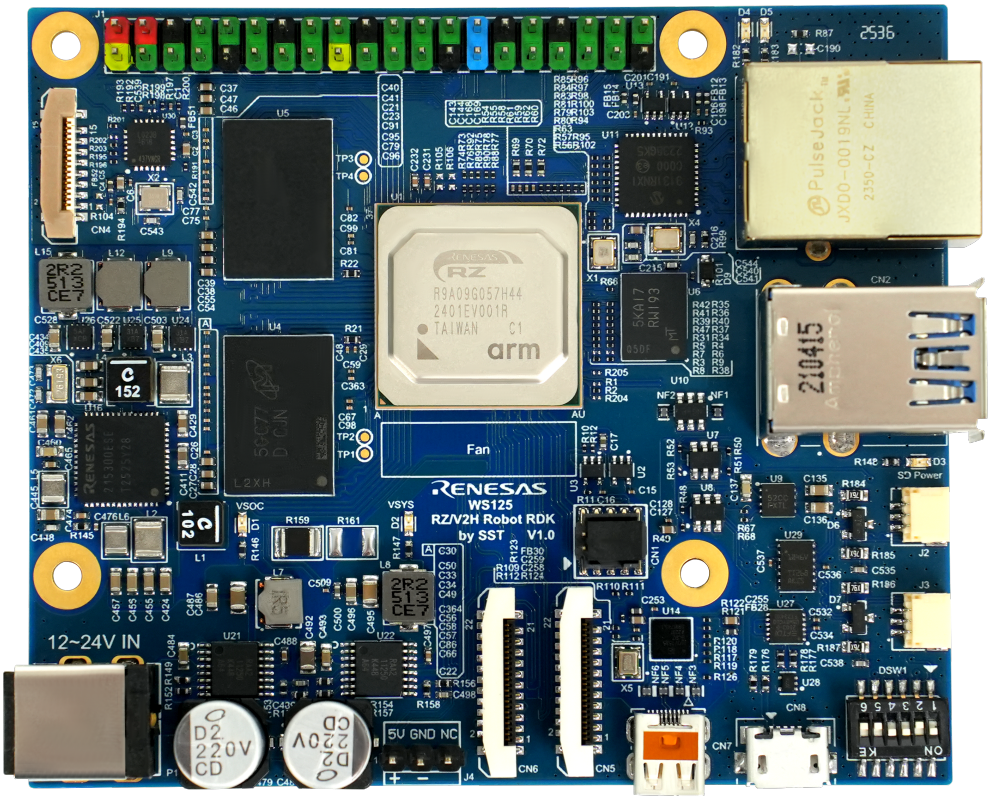

Fig. 1: RZ/V2H RDK Board Top View

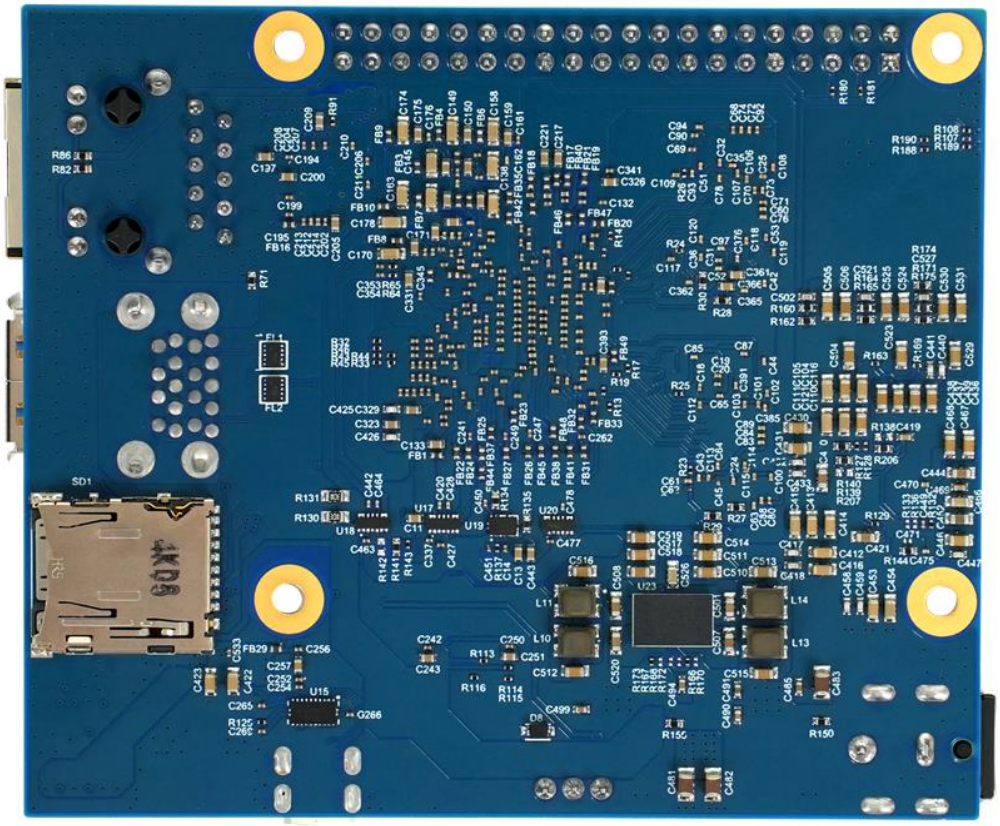

Fig. 2: RZ/V2H RDK Board Bottom View

1.2.1 Preparing the SD Card

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

Requirements

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

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

Flash using Balena Etcher

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

1. **Install Balena Etcher**

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

2. **Flashing the Image**
   - Once Etcher is open:

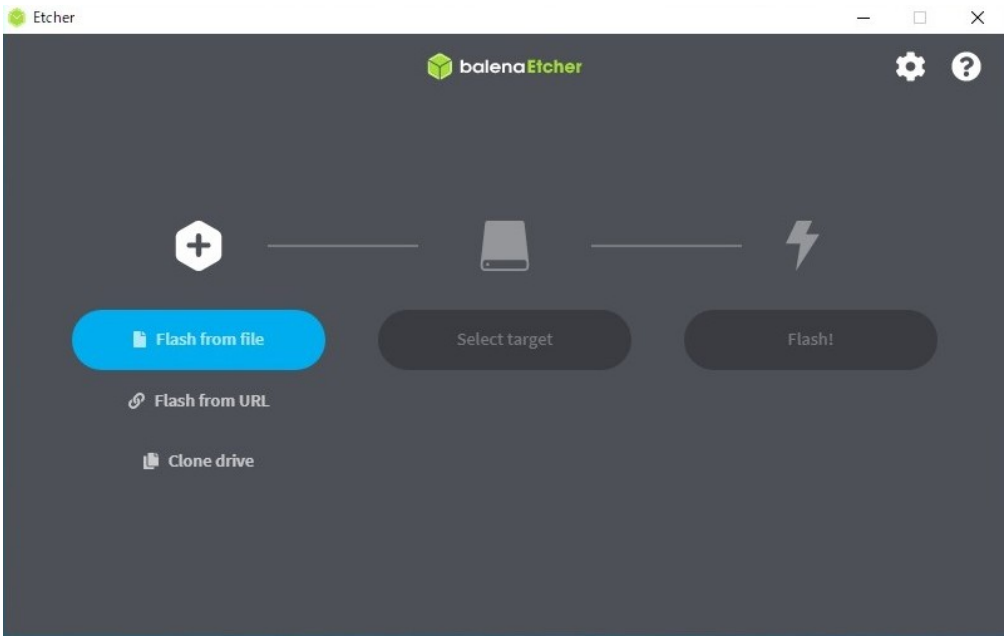

Fig. 3: Balena Etcher Application

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

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

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

1.2.2 Boot Mode Configuration (DIP Switch)

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

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

Attention

Always power off the board before changing boot switches.

1.2.3 Boot Mode Support

The board supports multiple boot options, including:

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

Tip

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

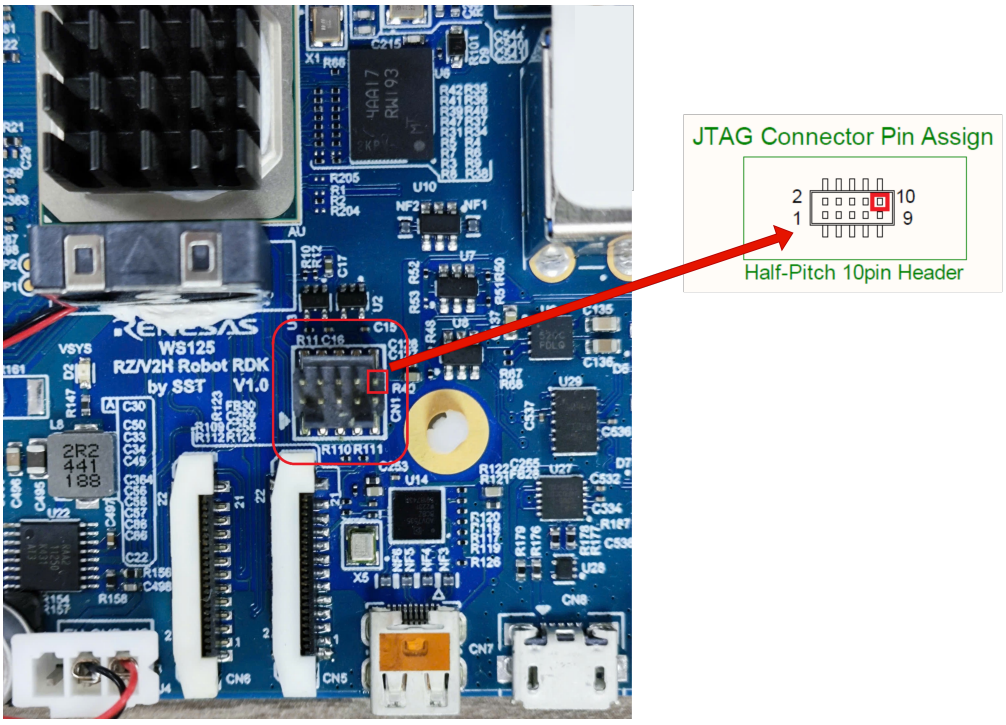

Fig. 4: JTAG Reset Pin Example

Option 1: SD Card Boot Mode

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

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

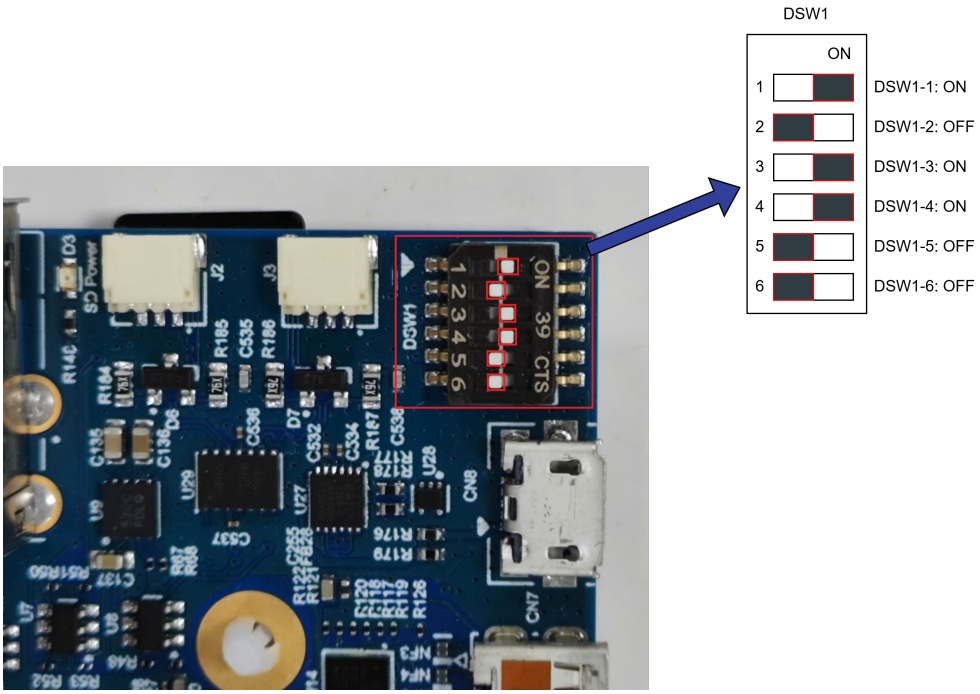

Fig. 5: DSW1 SD Card Boot Mode

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

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

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

The board will start the boot process.

Tip

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

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

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

Option 2: xSPI Boot Mode

Board Setup Procedure

Follow the instructions below to set up the board.

1. **Install Terminal Emulator**

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

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

2. **Install the Serial Port Driver**

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

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

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

3. **Write Bootloaders to the Board**

Copy the bootloaders file to your Windows PC.

File Name	Description
Flash_Writer_SCIF_RZV2H_DEV_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)**.
- Connect the power cable (**Max 24V/5A**).
- Open **Tera Term** and configure:

  Setup → Terminal:

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

  Setup → Serial Port:

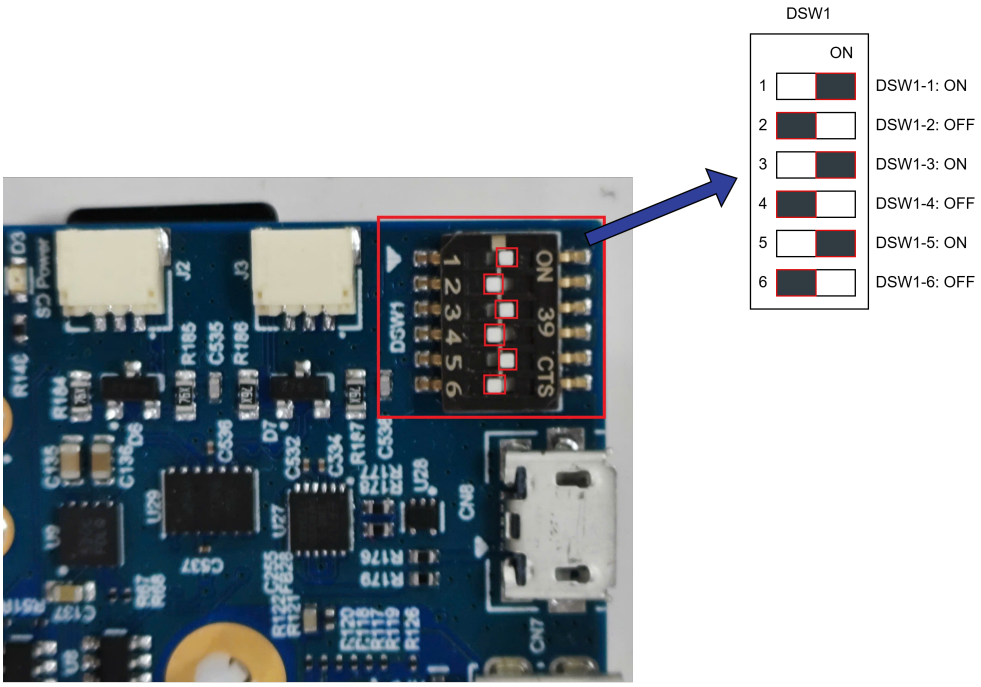

Fig. 6: DSW1 SCIF Download Mode

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

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

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

4. **Setup U-Boot Configuration**
   1. Insert the microSD card to the board.
   2. Change DSW1 to **Boot mode 2 (xSPI boot)**:
   3. Connect via **USB Serial to MicroUSB** cable.
   4. Power on the board.
   5. Open the terminal emulator and connect to the **COM** port (same configuration as before).
   6. The board will boot.

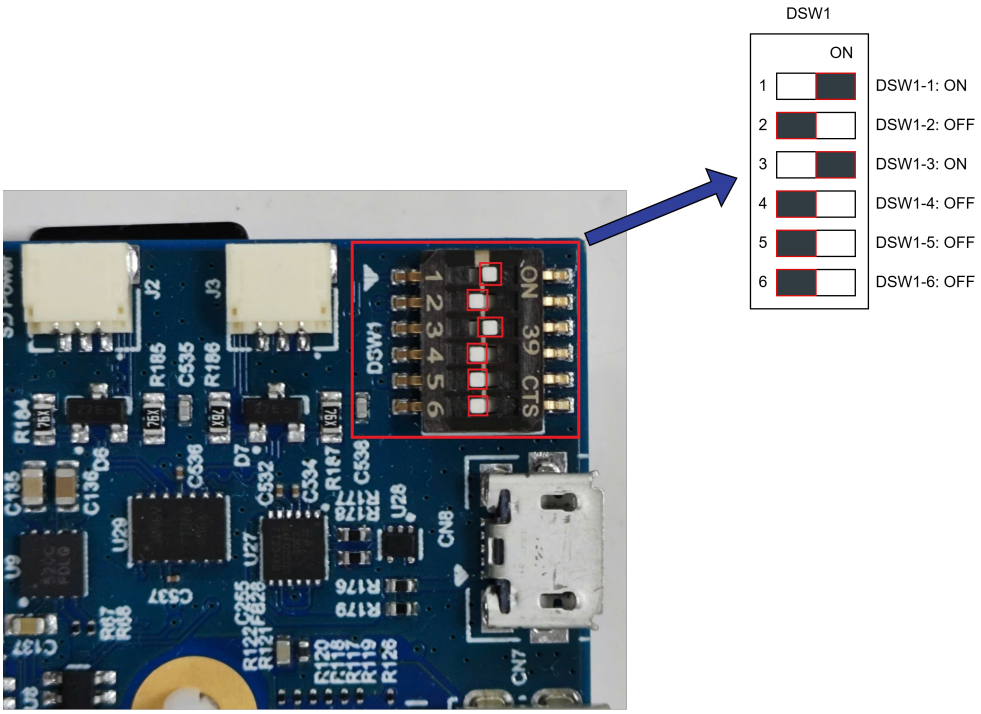

Fig. 7: DSW1 xSPI Boot Mode

Tip

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

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

1.2.4 First Time Boot Setup

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

Table 1: Default Login Information

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

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

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

Connect an Ethernet cable to the board and run:

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

1. Perform apt update and resize the SD card:

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

   Inside parted terminal:

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

   Resize root filesystem:

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

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

   ```
   sudo rosdep init
   rosdep update
   ```

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

1.2.5 Reference

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

1.3 RZ/V ROS2 Demos

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

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

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

2 System Configuration

2.1 Common system configuration

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

2.1.1 Overview

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

The Yocto Project eSDK provides tools that allow developers to:

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

To begin working with the Extensible Software Development Kit (eSDK) in Yocto, consult the official documentation provided by the Yocto Project.

This guide offers comprehensive instructions on configuring and utilizing the eSDK effectively.

Access the official eSDK documentation by following this URL: Using the Extensible SDK.

2.1.2 Prerequisites

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

Item	Description / Link
Docker	Must be installed on the Host PC. Refer to the Docker Official Installation Guide.
Yocto eSDK for RZ/V2H RDK	Required for extensible development and rebuilding components. Obtain the installer: `poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-ext-5.1.4.sh` *(TODO: Add link to Yocto eSDK package)*
Yocto SDK for RZ/V2H RDK	Standard SDK tool-chain for building applications. Obtain the installer: `poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.1.4.sh` *(TODO: Add link to Yocto SDK package)*
RZ/V2H RDK X Compile Docker	Preconfigured cross-compilation Docker environment. Refer to the *RZ/V2H RDK X Compile Docker repository. (TODO: Add repository link)*

2.1.3 Quick set up guide

Common docker environment setup

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

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

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

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

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

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

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

   Hint

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

   This step is required to prepare the cross-compilation environment for cross-compiling the ROS2 applications targeting the RZ/V2H platform.

   The setup of this cross-compilation workflow will be introduced in a later section.

3. Build the Docker Container

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

```
renesas@builder-pc:~$ cd x_compilation_docker/
renesas@builder-pc:~/x_compilation_docker$ ./setup_ros2_cross_env.
sh <path_to_ros2_workspace> [name_of_docker_container]
```

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

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

What does this script do?

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

4. Enter the Docker Container

   Use the following command to enter the Docker container:

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

eSDK Setup

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

Please replace the paths below with your actual file locations.

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

To set up your environment:

1. Install the Yocto eSDK tool-chain.

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

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

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

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

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

2. Set up cross-compile environment.

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

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

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

2.1.4 Using devtool in the Yocto eSDK

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

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

1. devtool modify

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

   Example: To modify the Linux kernel recipe:

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

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

2. devtool build

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

   Example: To build the modified Linux kernel:

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

   This command compiles the changes and prepares them for deployment.

3. devtool reset

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

   Example: To reset the Linux kernel recipe:

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

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

4. devtool build-image

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

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

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

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

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

Known Issue

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

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

The ros2-control:do_package_qa sig is computed to be
ea8f7e910d566912b932cbe602d93b93502064e293d1f4f1f569a67ee49f1c72,but the sig
is locked to fd89fc1eb9961fd4ccddf16ea2ca1b73b5480ce1670ebb07a8075603bb645bc8
in SIGGEN_LOCKEDSIGS_t-cortexa55
```

Note

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

2.1.5 Custom Linux Kernel and Device Tree

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

Note

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

1. modify the Linux kernel recipe:

   Setup the eSDK environment in the current terminal session:

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

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

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

When executed, this:

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

2. Applying Patches for linux-yocto

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

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

Procedure, applying patches to the kernel linux-yocto source

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

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

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

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

3. Adding Kernel Configuration Fragments

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

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

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

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

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

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

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

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

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

  - Add the following line to the bbappend file:

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

4. Modify the DTS file:

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

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

5. Build the Modified Kernel

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

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

Attention

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

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

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

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

6. Collect the Built Kernel Artifacts

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

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

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

Tip

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

```
$ sudo depmod -a
```

2.2 Ubuntu System with RZ/V2H RDK

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

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

2.2.1 Overview

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

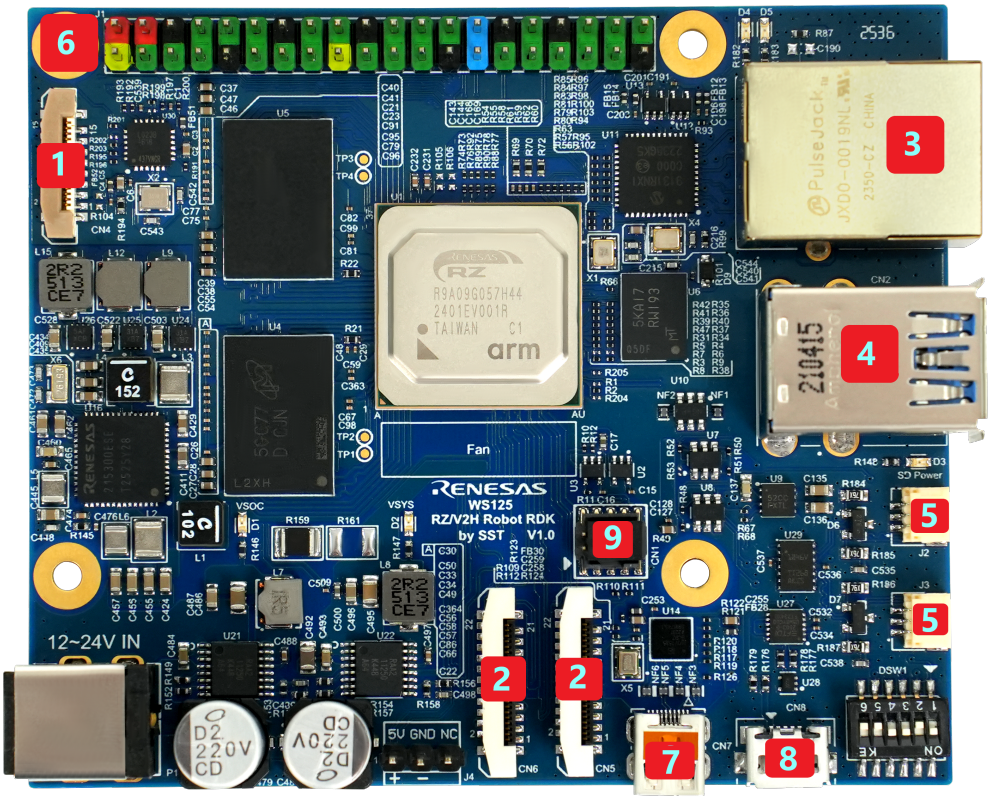

Fig. 1: RZ/V2H Robotic Development Kit Interfaces

2.2.2 Main Interfaces

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

Table 1: High-Speed Interfaces

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

Table 2: Communication Interfaces

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

Table 3: Other Interfaces

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

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

High-Speed Interface

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

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

1. PCIe 3.0 16-pin connector

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

Important

The PCIe interface is not available in this release.

2. MIPI-CSI 22-pin connector ×2

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

See also

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

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

Hint

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

Set up the MIPI-CSI interface

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

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

```
#!/bin/bash

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

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

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

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

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

# Check for no input
if [ -z "$1" ]; then
    echo "Detected camera: $default_camera"
    echo "No resolution specified. Using default resolution: ${camera_default_
resolution[$default_camera]}"
    ov564x_res="${camera_default_resolution[$default_camera]}"
```

(continues on next page)

(continued from previous page)

```
else
    ov564x_res="$1"
    valid_resolutions=(${camera_valid_resolutions[$default_camera]})
    # Check if the given resolution is valid
    if [[ ! " ${valid_resolutions[@]} " =~ " ${ov564x_res} " ]]; then
        echo "Invalid resolution: $ov564x_res for camera $default_camera"
        ov564x_res="${camera_default_resolution[$default_camera]}"
        echo "Input resolution is not available. Using default resolution: $
{camera_default_resolution[$default_camera]}"
    fi
fi

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

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

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

For other camera modules, please modify the script accordingly.

Usage example with v4l2-ctl

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

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

List all connected cameras:

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

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

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

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

3. 1000M RJ45 - Gigabit Ethernet Port

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

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

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

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

To list all network interfaces and their IP addresses:

```
$ ifconfig
```

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

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

Set a static IP address

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

- Open the network configuration file with vim:

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

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

  ```
  # This file describes the network interfaces available on your system
  # For more information, see netplan(5).
  network:
  version: 2
  renderer: NetworkManager
  ethernets:
      end0:
      dhcp4: no
      dhcp6: no
      addresses: [169.254.43.99/24]
      routes:
        - to: default
          via: 169.254.43.86
      nameservers:
          addresses: [8.8.8.8, 8.8.4.4]
  ```

- Apply the changes with the following command:

```
$ sudo netplan apply
```

Set DHCP

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

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

- Apply the changes with the following command:

```
$ sudo netplan apply
```

4. USB 3.0 Type A ×2

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

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

Verify USB 3.0 functionality

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

```
$ lsusb -t
```

Example output:

```
root@localhost:~# lsusb -t
/:  Bus 001.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
480M
/:  Bus 002.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
20000M/x2
/:  Bus 003.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
480M
/:  Bus 004.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
20000M/x2
root@localhost:~#
```

Communication Interfaces

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

5. CAN-FD ×2

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

Tip

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

We recommend using the: **Bus Cable, 1 Female Connector, 3 Contacts, 1.0 mm Pitch, 10 cm Length**.

The RZ/V2H RDK is equipped with an onboard CAN transceiver (**TCAN1046VDMTRQ1**) and an integrated **120 Ω termination resistor**, eliminating the need for any external CAN transceiver circuitry.

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

Usage examples:

Verify that the CAN interfaces are recognized:

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

Example output:

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

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

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

Check the interface status:

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

Send and Receive CAN Messages

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

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

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

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

```
$ candump can0
```

3. On another terminal, send a test frame:

```
$ cansend can0 123##01122334455667788
```

6. RasPi GPIO 40-pin Header

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

Support the following communication protocols:

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

Pin Out Diagram

I2C (Inter-Integrated Circuit)

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

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

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

Table 4: I2C7 Interface Pins

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

Usage example with i2c-tools

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

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

List all I2C buses available on the system:

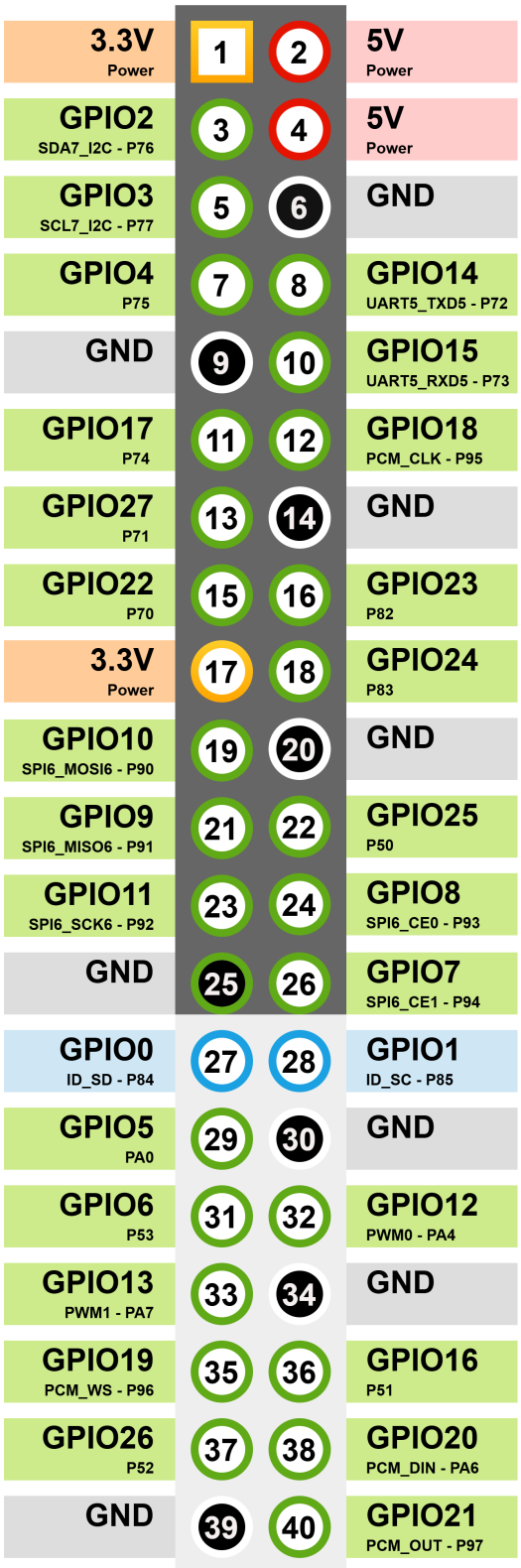

Fig. 2: RZ/V2H RDK Raspberry Pi GPIO 40-pin Header Pin Out

```
$ i2cdetect -l
```

Example output:

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

Find the correct bus number for I2C7:

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

In this example, I2C7 corresponds to bus number 10.

Hint

How to identify the correct I2C bus number for I2C7?

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

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

Scan for I2C devices on bus 10:

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

SPI (Serial Peripheral Interface)

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

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

Table 5: SPI6 Interface Pins

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

Usage example

List SPI devices:

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

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

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

UART (Universal Asynchronous Receiver/Transmitter)

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

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

Table 6: UART5 Interface Pins

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

Usage example with minicom

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

```
$ sudo apt install minicom
```

List available serial ports:

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

Example output:

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

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

Tip

How to identify the correct UART port for UART5?

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

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

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

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

GPIO (General Purpose Input/Output)

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

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

Usage example with gpiod:

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

```
$ sudo apt install gpiod
```

List available GPIO chips:

```
$ gpiodetect
```

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

```
$ gpioinfo gpiochip1
```

Set a GPIO line as output and change its value:

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

Read the value of a GPIO line:

```
$ gpioget gpiochip1 92
```

PCM (Pulse Code Modulation)

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

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

Table 7: PCM Audio Interface Pins

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

TODO: Add usage examples for PCM interface.

Other interfaces

7. Mini-HDMI

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

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

8. USB-UART

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

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

The configuration settings for the serial connection are as follows:

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

9. JTAG 10-pin

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

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

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

TODO: Add link to Multi-OS Development Section

3 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.

3.1 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.

3.1.1 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.

3.1.2 Concepts

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:

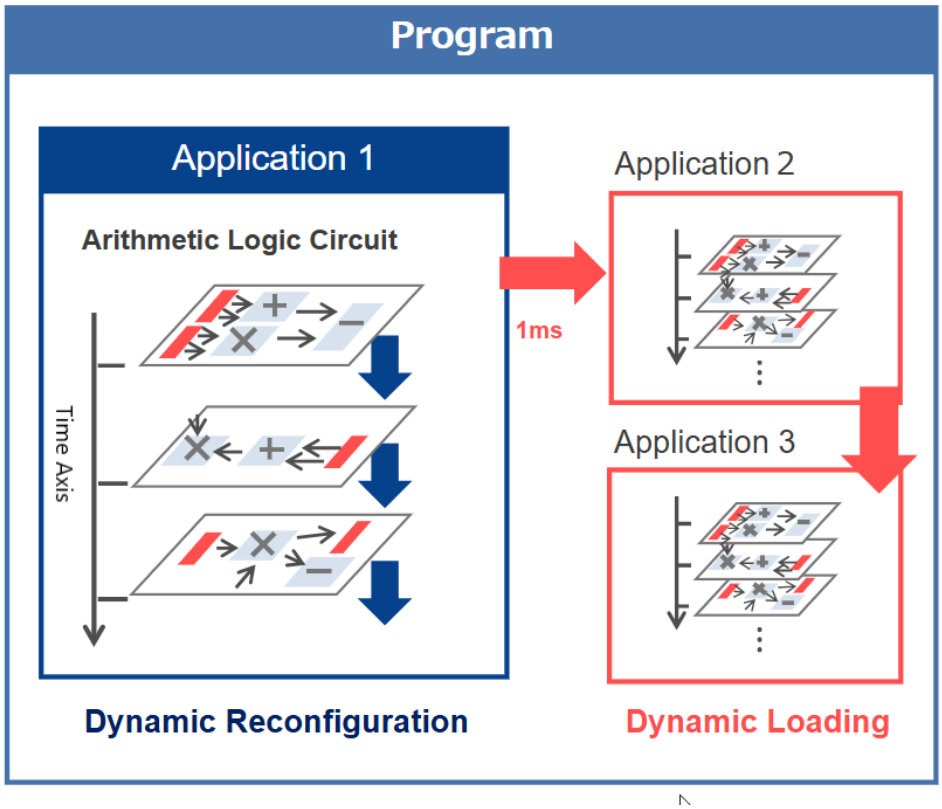

Fig. 1: DRP Dynamic Reconfiguration

What is DRP-AI?

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

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

> **See also**
>
> 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:
>
> **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)

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.

> **See also**
>
> For more detail information about the DRP-AI Driver, refer to the RZ/V2H DRP-AI Driver.
>
> It provide API functions to help you get started with DRP-AI Driver development on the RZ/V2H platform.

3.1.3 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.

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.

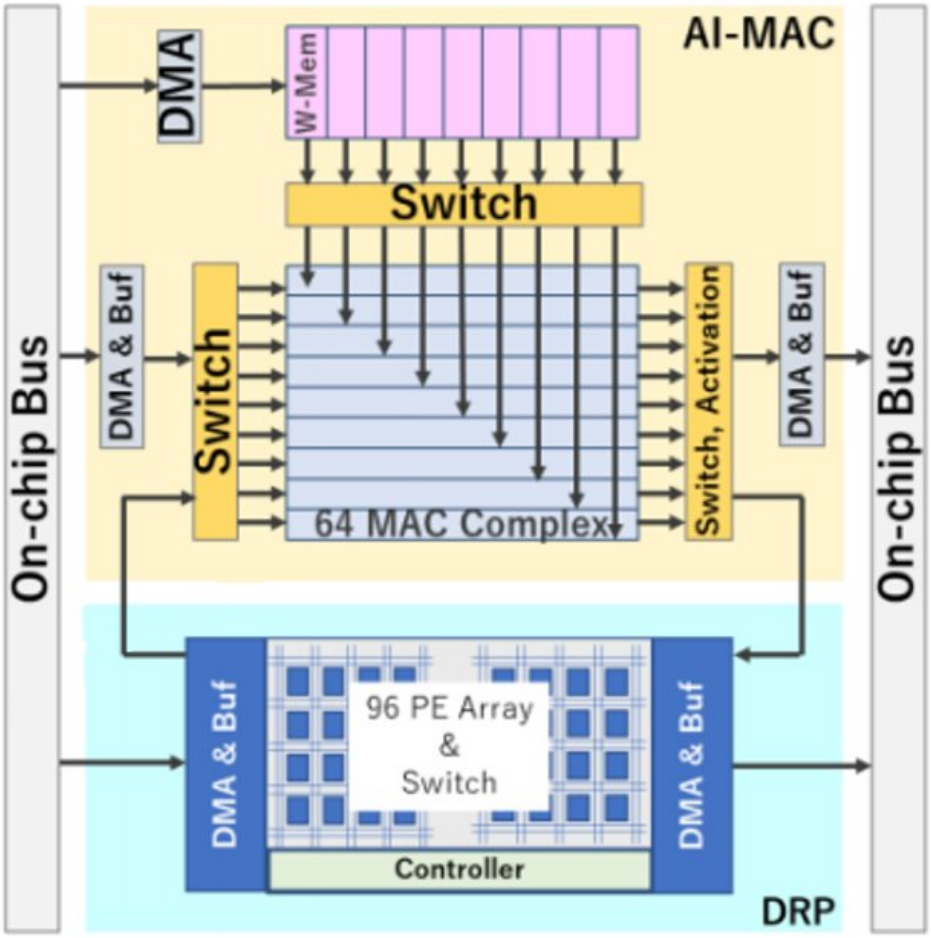

Fig. 2: DRP-AI Driver Architecture

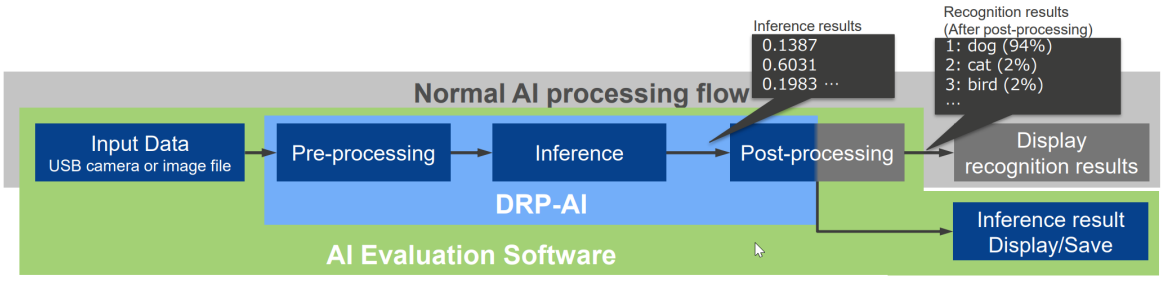

Fig. 3: DRP-AI Driver Execution Flow

See also

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

- DRP-AI TVM Extension Package.
- DRP-AI TVM on RZ/V series.

Install the DRP-AI TVM extension package

To install the DRP-AI TVM extension package, follow the instructions provided in the package's GitHub repository: Installation.

We recommend installing DRP-AI TVM with Docker for ease of setup and to avoid dependency issues.

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:

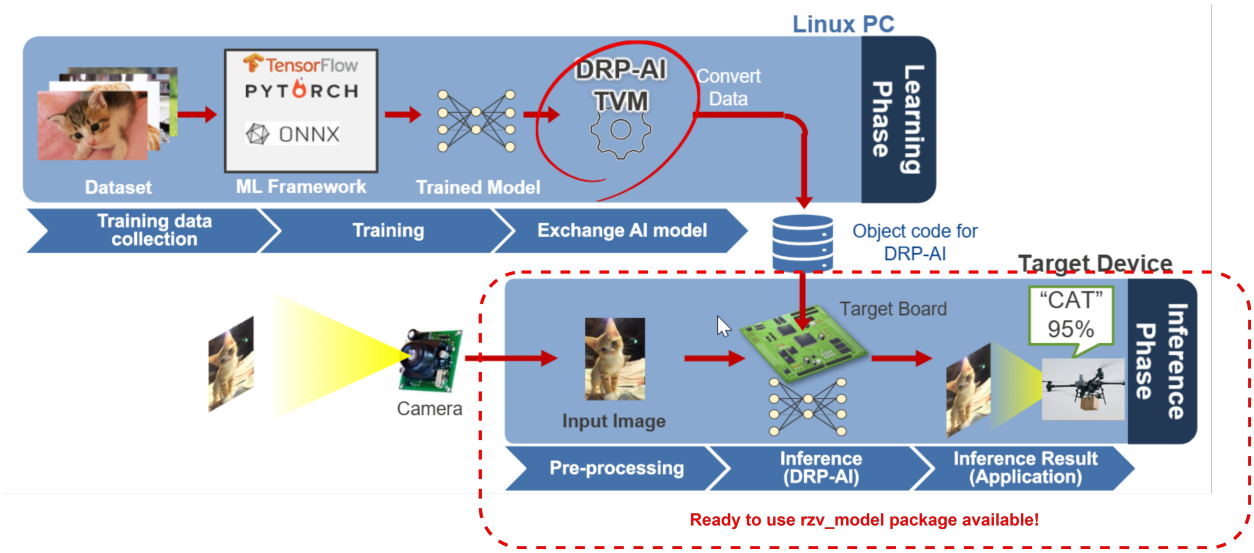

Fig. 4: BYOM Development Flow

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.

Hint

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.

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.

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.

See also

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

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

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.

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/V MPU 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.

3.1.4 The rzv_model package

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.

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

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.

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.

How to use the rzv_model package

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 subdirectories representing different inference stages (executed by DRP or CPU) that each contain an `addr_map.txt` file.

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

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.

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.

Hint

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

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.

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);
```

3.1.5 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.

Prequisites

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.

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 *Prequisites step of the tutorial* before proceeding.

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

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
```

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*.

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 *Prequisites step of the tutorial* before proceeding.

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.

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.

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).

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
```

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*.

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 *Prequisites step of the tutorial* before proceeding.

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

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
```

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
```

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_
```

(continues on next page)

(continued from previous page)

```
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
```

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*.

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 *Prequisites step of the tutorial* before proceeding.

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

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 \
```

(continues on next page)

(continued from previous page)

```
      -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"
```

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*.

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 *Prequisites step of the tutorial* before proceeding.

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

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_freihand \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/freihand_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"
```

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*.

3.2 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*.

See also

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.

3.3 OpenCV Accelerator

3.3.1 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.

See also

For more detail information about OpenCVA, refer to the RZ/V2H OpenCV Accelerator.

3.3.2 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 1: 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 convolving.
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.

Note

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

*/usr/lib/aarch64-linux-gnu/renesas/libopencv**

/usr/include/opencv4/renesas/opencv4

3.4 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.

3.4.1 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

Important

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

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

Attention

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.

TODO: Update the image with the SW1-6 is set to "ON".

3.4.2 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.

3.4.3 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.

3.4.4 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 rzv2h_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),
```

(continues on next page)

(continued from previous page)

```
+    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*.

3.4.5 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 Templete 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.

3.4.6 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.

Hardware supported

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

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

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

Installation Guide

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 prebuilt 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.

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
```

Usage Guide

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
```

(continues on next page)

(continued from previous page)

```
[xxx] 43000000.vring-ctl0 closed
...
```

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
...
```

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**.