Quick Start Guide

Renesas Smart Power Controller

Webserver Application

User Manual

Renesas RZ Application
RZ/G Series

Renesas Electronics
www.renesas.com

Rev.2.1.0 Dec.05.25

Trademarks (continued)

For the “Cortex” notation, it is used as follows;
— Arm® Cortex®-A55
— Arm® Cortex®-M33
Note that after this page, they may be noted as Cortex-A55 and Cortex-M33 respectively.

Examples of trademark or registered trademark used in the RZ/G2L SMARC Module Board RTK9744L23C01000BE User’s Manual: Hardware;
CoreSight™: CoreSight is a trademark of Arm Limited.
MIPI®: MIPI is a registered trademark of MIPI Alliance, Inc.
eMMC™: eMMC is a trademark of MultiMediaCard Association.

Note that in each section of the Manual, trademark notation of ® and TM may be omitted.
All other trademarks and registered trademarks are the property of their respective owners.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Contents

Renesas RZ Family

Renesas System Release Package

Introduction

The **Smart Power Controller** is a Debian-packaged utility designed to control GPIO-based relay circuits for power cycling, deployed on embedded devices such as Renesas RZ boards. It provides both command-line and REST API interfaces and supports integration with web dashboards or automation pipelines.

Features

The following are the general features of the release package:

- Control up to 8 relays via GPIO using Python
- REST API for integration with automation tools
- Web UI with live relay toggles and feedback
- Basic HTTP authentication for secure access
- JSON configuration for GPIO mapping
- Easy installation/uninstallation via .deb
- Terminal interface for embedded shell use
- Verified on RZ/G2L platform with relay modules

It is especially useful in embedded labs where remote rebooting or hardware toggling is required frequently during testing.

Package Contents

The .deb package installs the following files:

Table 1. Package Contents

File Path (Destination)	Description
/usr/local/bin/gpio_api_c.py	Main Python program for REST + GPIO control
/usr/local/bin/gpio_terminal.sh	Shell wrapper to run the terminal interface
/usr/local/etc/at-powercycling/gpio_config.json	User-defined GPIO pin map
/usr/local/etc/at-powercycling/requirements.txt	pip dependencies (Flask, Flask-HTTPAuth)
/usr/local/share/at-powercycling/templates/	HTML files for Flask dashboard
/usr/local/share/doc/at-powercycling/README.md	Readme with example usage

Glossary

Terms	Description
GPIO	General Purpose Input/Output — Programmable pins on the SBC that can be set to HIGH (on) or LOW (off) to control external devices like relays.
Relay	An electrically operated switch that can turn devices on/off by opening or closing circuits, controlled via GPIO signals.
Relay Module	A board containing one or more relays, often with driver circuits, that interfaces between low-voltage GPIO control and higher-voltage loads.
SBC	Single Board Computer — A complete computer built on a single circuit board, such as the Renesas RZ/G2L or RZ/V2L.
Target Device	The hardware you are controlling or power cycling using the relay (e.g., another SBC, embedded device, or test equipment).
Sysfs Path	A Linux virtual filesystem path (e.g., /sys/class/gpio/...) used to control and monitor hardware interfaces from user space.
Python Flask	A lightweight Python web framework used to create the REST API and Web UI for the Smart Power Controller.
REST API	Representational State Transfer Application Programming Interface — A set of HTTP endpoints allowing control of relays from software or scripts.
Web UI	A browser-based graphical dashboard for controlling relays and viewing system status.
Debian Package (.deb)	A Linux package format used by Debian and Ubuntu-based systems to distribute and install software.
Basic Auth	A simple authentication method that uses a username and password sent with each HTTP request.
SSH	Secure Shell — A network protocol for securely logging into a remote machine and running commands.
UART	Universal Asynchronous Receiver-Transmitter — A hardware communication protocol used for serial console connections to the SBC.
Minicom	A terminal program for communicating with a device via a serial connection (e.g., USB-to-UART).
SCP	Secure Copy Protocol — A way to securely transfer files between your PC and the SBC over a network.
IP Address	A unique identifier for a device on a network, needed to access the Web UI or REST API remotely.
Yocto	A Linux build system is often used for creating minimal or customized Linux images for embedded devices.
VCC	Voltage Common Collector — The power supply voltage input for electronic components.
GND	Ground — The reference voltage level (0V) in an electrical circuit, often common to all devices in the setup.
High/Low Signal	In GPIO control, High means voltage applied (logic 1) and Low means no voltage (logic 0).

1. Introduction

The RZ smart power controller application is an implementation of a generic IO controller based on a standard mechanical relay interface. At its core, it's a web server and tools implementation that toggles specified GPIO lines that are expected to control a relay.

The web server is basically a Python Flask application and contains command-line tools that allow the user fine-grained control of the IO lines and the web server.

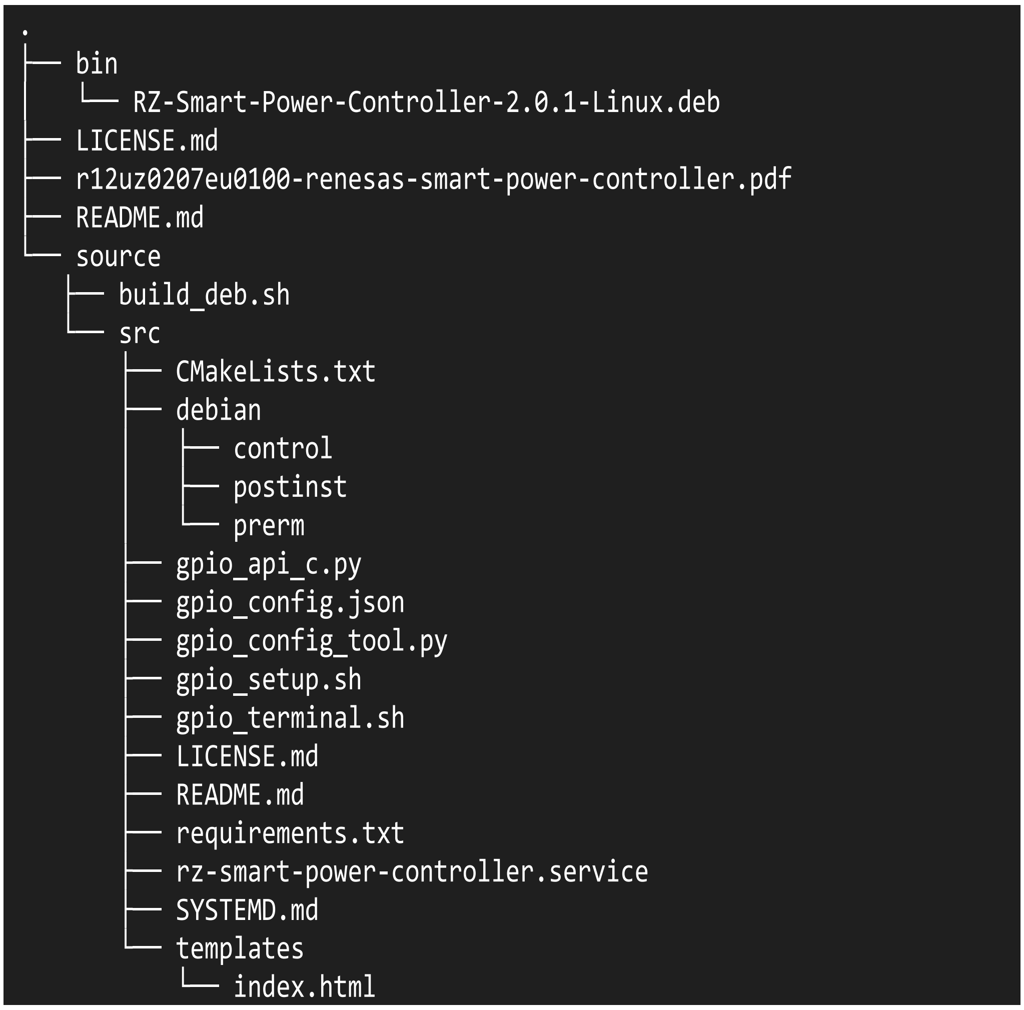
1.1 Features

The following are the features of this application.

- Python Flask-based webserver.
- JSON-based dynamic IO line configuration.
- Libgpiod and sysfs support (Kernel 5.10 & 6.10 support).
- Systemd services to manage the web server dynamically.
- .deb installer for easy installation, removal, and package management.
- Config and support tools.
- Shell interface for local control.

1.2 Release Contents

The following is the package hierarchy.



```
.
├── bin
│   └── RZ-Smart-Power-Controller-2.0.1-Linux.deb
├── LICENSE.md
├── r12uz0207eu0100-renesas-smart-power-controller.pdf
├── README.md
└── source
    ├── build_deb.sh
    └── src
        ├── CMakeLists.txt
        ├── debian
        │   ├── control
        │   ├── postinst
        │   └── prerm
        ├── gpio_api_c.py
        ├── gpio_config.json
        ├── gpio_config_tool.py
        ├── gpio_setup.sh
        ├── gpio_terminal.sh
        ├── LICENSE.md
        ├── README.md
        ├── requirements.txt
        ├── rz-smart-power-controller.service
        ├── SYSTEMD.md
        └── templates
            └── index.html
```

Prebuilt package: bin/RZ-Smart-Power-Controller-2.0.1-Linux.deb is the prebuilt Debian package that can be directly installed on a target arm64 board.
License.md: Package license text.
r12uz0207eu01xx-renesas-smart-power-controller.pdf: Comprehensive user manual.
build_deb.sh: Build script that can build the package from source.
README.md: Primary readme for information.
Source: It contains a copy of the tested source code.
source/src/gpio_config.json: This file is the template configuration file for the RZ/G2L-SBC.

1.3 Source Repository

The release package contains the currently tested source code under the 'source' directory . The code is also hosted on a public GitHub repository, Renesas-SST/poc-rz-smart-power-controller (Renesas-SST/poc-rz-smart-power-controller).

2. System Overview

The Smart Power Controller Tool is a lightweight utility designed to remotely toggle relays using GPIO on embedded Linux platforms like the Renesas RZ/G2L. It is especially useful in test labs where remote reboots or cycling the power of boards are frequently required.

This tool offers multiple interfaces:

- **Web UI** – A browser-based dashboard to toggle relays
- **REST API** – Programmatic relay control using HTTP requests
- **Terminal Interface** – Local shell interface for manual or script-based control

2.1 Compatible Boards

The tool is tested and verified on the following Renesas platforms:

Board Model	Notes
RZ/G2L-SBC	Official platform; supports GPIO output via 40 40-pin GPIO port.
RZ/V2L-SBC	Also supports GPIOs through similar sysfs interfaces on 40-pin GPIO.
RZ/G2L-EVK	Supports through pmod ports, GPIO pins
RZ/V2L-EVK	Supports through pmod ports, GPIO pins

GPIO control is performed using ‘libgpiod’ on kernel 6.x and above while falling back to sysfs interface on older kernels. The implementation is generic enough that it can be configured for any board of the same class as the ones mentioned here.

2.2 Supported Platforms

The .deb package is intended for installation on the following Debian-based Linux distributions:

OS	Version	Kernel Version	Notes
Ubuntu	24.04 (Noble), 22.04 (Jammy), 20.04 (Focal)	5.10 CIP	Uses ‘sysfs’ interface
	24.04 (Noble), 22.04 (Jammy), 20.04 (Focal)	6.10	Uses ‘libgpiod’ interface
Debian	12 (Bookworm)	6.10	Uses ‘libgpiod’ interface
	11 (Bullseye)	5.10	Uses ‘sysfs’ interface

While the tool may run on other systems with Python 3.6+ and Flask, only the above configurations are officially tested.

Note: Yocto-based minimal images may require manual installation of Python3, pip, and networking tools to support the tool’s dependencies.

2.3 System Requirements

Component	Requirement
Python	Version 3.6 or newer
Disk Space	At least 10 MB free space
Network	Ethernet access (for API/web usage)
GPIO Access	Board user must have GPIO privileges
Additional	Relay board powered separately

2.4 Architecture Diagram

Figure 1. Architecture Diagram shows the high-level architecture of how the Smart Power Controller works

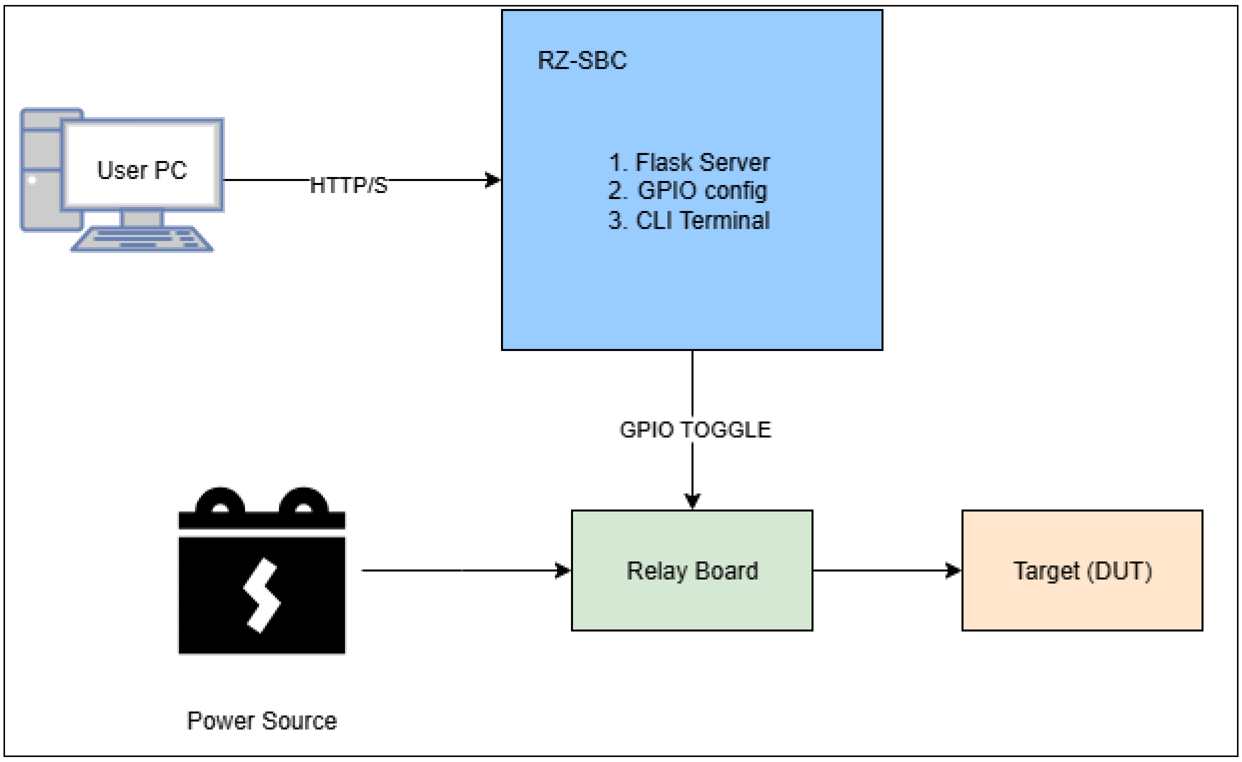


Figure 1. Architecture Diagram

2.5 How It Works

1. The .deb package installs the web server, dependencies like Python, and shell scripts on the target board. The web server is managed by the system and is automatically started after installation and bootup.

2. The user can control the service through the name ‘rz-smart-power-controller’ systemd service name.

```
root@localhost:~# systemctl status rz-smart-power-controller
● rz-smart-power-controller.service - RZ Smart Power Controller Service
   Loaded: loaded (/usr/local/lib/systemd/system/rz-smart-power-controller.service;
enabled; preset: enabled)
   Active: active (running) since Thu 2025-12-04 04:50:16 +07; 10h ago
  Main PID: 401 (python3)
    Tasks: 1 (limit: 699)
   Memory: 32.2M (peak: 42.1M)
      CPU: 26.487s
   CGroup: /system.slice/rz-smart-power-controller.service
           └─401 /usr/bin/python3 /usr/local/bin/gpio_api_c.py

Dec 04 08:42:11 localhost.localdomain python3[401]: Setting initial value to inactive (1) for
active low
Dec 04 08:42:11 localhost.localdomain python3[401]: Setting initial value to inactive (1) for
active low
Dec 04 08:42:11 localhost.localdomain python3[401]: Setting initial value to inactive (1) for
active low
Dec 04 08:42:11 localhost.localdomain python3[401]:  * Serving Flask app 'gpio_api_c'
Dec 04 08:42:11 localhost.localdomain python3[401]:  * Debug mode: off
Dec 04 08:42:11 localhost.localdomain python3[401]: INFO:werkzeug:WARNING: This is a
development server. Do not use it in a production deployment. Use a production WSGI
server instead.
Dec 04 08:42:11 localhost.localdomain python3[401]:  * Running on all addresses (0.0.0.0)
Dec 04 08:42:11 localhost.localdomain python3[401]:  * Running on http://127.0.0.1:5000
Dec 04 08:42:11 localhost.localdomain python3[401]:  * Running on
http://192.168.1.208:5000
Dec 04 08:42:11 localhost.localdomain python3[401]: INFO:werkzeug:Press CTRL+C to
quit
```

3. Most of the installation is done on /usr/local path. The tool reads relay-to-GPIO mapping from /usr/local/etc/rz-smart-power-controller/gpio_config.json.
4. When a relay toggle is triggered, it uses the appropriate interface to activate/deactivate the relay.
5. The relay module switches power to the connected embedded device(s).

3. Quick Start Instructions

This section provides comprehensive steps to install, configure, and validate the Smart Power Controller tool on a supported embedded Linux board such as the RZ/G2L. It assumes the board has access to a terminal (serial, SSH, or desktop) and supports .deb packages and Python3.

3.1 Prerequisites

Before installing and running the Smart Power Controller, ensure that the following hardware, software, and information are handy. This will help prevent delays or issues during setup.

3.1.1 Hardware Requirements

- Renesas RZ/G2L or RZ/V2L SBC (or any other supported platform)
- Relay module (up to 8 channels) with separate power supply
- Target device(s) to be powered
- Required cables:
  1. GPIO jumper wires (male–female or female–female as needed)
  2. UART cable for serial console access
  3. Ethernet cable or Wi-Fi connection for network control

3.1.2 Software Requirements

- Debian-based OS installed on the SBC (Ubuntu 20.04+/Debian 11+ recommended)
- .deb package for Smart Power Controller
- Python 3.6 or newer
- Network access between the control PC and the SBC

3.1.3 Identification and Credentials

- The SBC's IP address on the network
- Default Web UI login credentials:
  1. Username: **admin**
  2. Password: **password123** (should be changed after initial setup)

3.1.4 Methods for Accessing the Board

All necessary commands required for installing and configuring the Smart Power Controller directly on the SBC are available through terminal sessions. There are two main ways to access the SBC terminal:

1. Via Serial Console (Recommended for first setup)

   Connect a USB-to-UART cable between the PC and the SBC's serial console port, as explained in RZ/G2L-SBC.
   On PC, open a terminal and run:

   ```
   $ sudo minicom -D /dev/ttyUSB0 -b 115200
   ```

   Press Enter to get the SBC login prompt.

2. Via SSH

   Find SBC's IP address from the router or via the serial console.
   From the PC terminal:

```
$ ssh user@<board-ip>
```

Replace 'user' with SBC's username (e.g., root or configured user).

Refer to SBC's documentation for setting up network connectivity.

3.2 Hardware Setup

Before running the software, connect the hardware as shown in the figure below. Ensure all connections are correct before powering on the SBC.

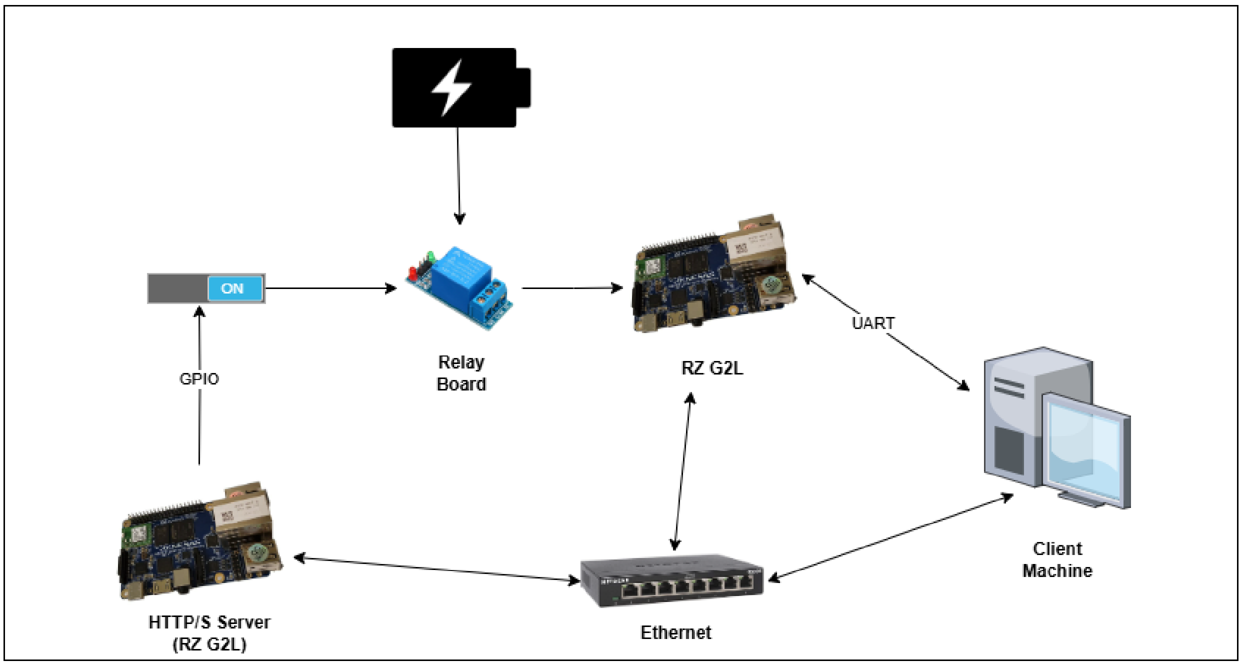


Figure 2. Example Setup with DUT

3.2.1 Steps

1. Connect the SBC GPIO pins to the relay module IN pins based on the mapping in /usr/bin/gpio_config.py.
2. Connect the relay module power (VCC, GND) to either the SBC or an external power source as per relay board requirements.
3. Connect the relay output terminals in series with the target device's power line.
4. Ensure all grounds are common between the SBC, relay module, and target device.
5. Power on the SBC and relay module.

3.2.2 Example Mapping (Default Configuration for RZ/G2L)

Table 2 shows the initial configuration of GPIO mapping and respective header labels. Figure 3. GPIO Layout for RZ/G2L shows the accessible pins and mapping on RZ/G2L. Users are free to choose any pin mapping based on their needs and pin availability.

Table 2. Pin Mapping for RZ/G2L

Relay No	SBC Linux Sysfs GPIO No	SBC Pin Header Label	SBC Linux libgpiod No
1	304	P23_0	184
2	456	P42_0	336

3	336	P27_0	216
4	345	P28_1	225
5	490	P46_2	370
6	491	P46_3	371
7	465	P43_1	345
8	466	P43_2	346

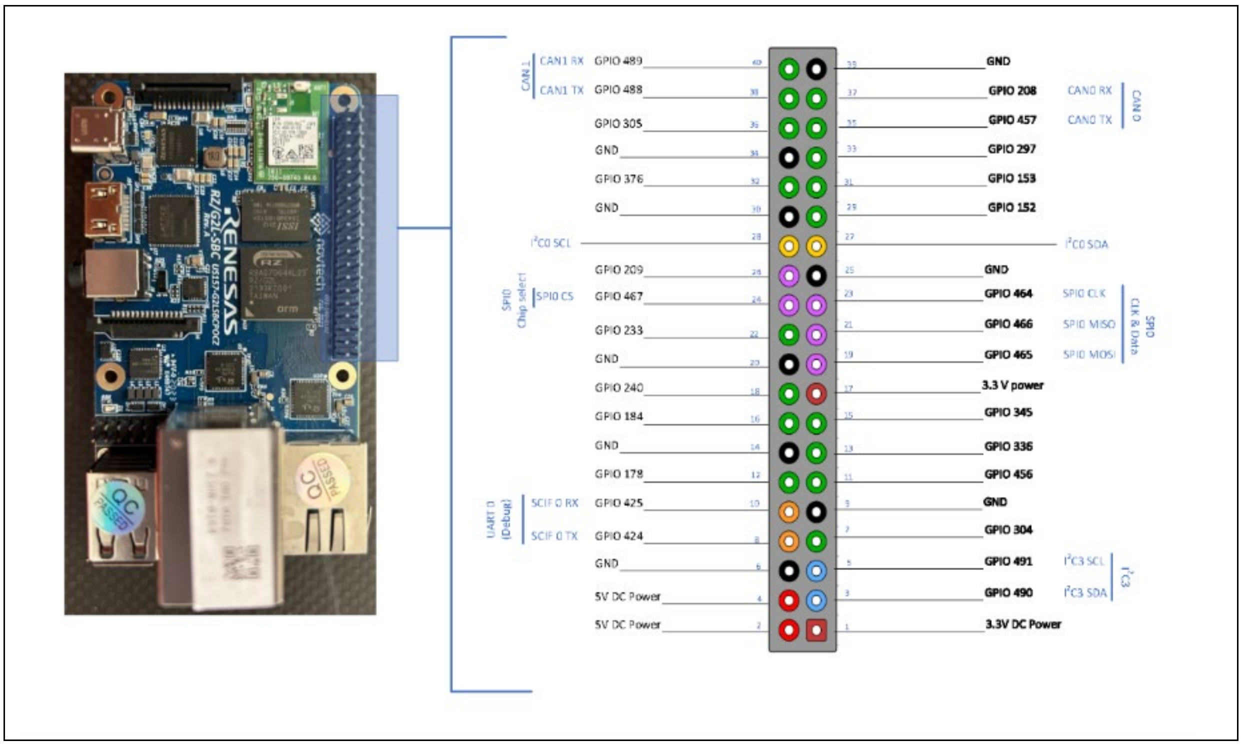


Figure 3. GPIO Layout for RZ/G2L

3.2.3 Safety Notes

- Always check relay voltage/current ratings before connecting the target device.
- If switching mains power, follow proper electrical safety practices.
- Avoid touching live circuits while the system is powered.

3.3 Installing the .deb Package

The tool is distributed as a prebuilt .deb package in the bin directory to simplify deployment. This allows the tool to be installed using native Debian utilities like dpkg or apt.

Step 1. Transfer the Package to the Board

The .deb package can be transferred using **SCP** (secure copy) over the network:
On the host machine, type the following command.

```
$ scp RZ-Smart-Power-Controller-2.1.0-Linux.deb user@<board-ip>:/tmp
```

Note: It can also be copied over the USB flash drive by accessing the flash drive on /media or /mnt. You can check the available storage media using the '**lsblk**' command. You can list all mount points using the **'mount'** command.

Step 2. Install the Package

Run the following command on the target machine:

```
$ apt-get update && apt-get upgrade
```

```
$ cd /tmp
$ apt-get install -y --allow-unauthenticated ./RZ-Smart-Power-Controller-2.1.0-
Linux.deb
```

This installs:

- Executables in /usr/local/bin/
- Configuration files in /usr/local/etc/rz-smart-power-controller/
- Web templates and README in /usr/local/share/rz-smart-power-controller/ and /usr/local/share/doc/

GPIOs are automatically set up based on the JSON file.

3.4 Verifying Installation

Once the package and dependencies are installed, verify the tool is working correctly:

3.4.1 Check Installed Files

Verify presence of files:

```
root@localhost:~# ls -l /usr/local/bin/
total 32
drwxrwxr-x  2 root root  4096 Dec  6 11:33 ./
drwxr-xr-x. 10 root root  4096 Dec  6 11:33 ../
-rwxr-xr-x  1 root root 11403 Dec  5 17:39 gpio_api_c.py*
-rwxr-xr-x  1 root root  2334 Dec  5 17:39 gpio_config_tool.py*
-rwxr-xr-x  1 root root  3247 Dec  5 17:39 gpio_setup.sh*
-rwxr-xr-x  1 root root  2075 Dec  5 17:39 gpio_terminal.sh*

root@localhost:~# ls -l /usr/local/etc/rz-smart-power-controller/
total 8
-rw-r--r-- 1 root root 1241 Dec  5 17:39 gpio_config.json
-rw-r--r-- 1 root root   20 Dec  5 17:39 requirements.txt
```

3.4.2 Check Webserver Service

The web server is managed by a system service file. You can use the standard system interface to interact with it.

```
root@localhost:~# systemctl status rz-smart-power-controller
● rz-smart-power-controller.service - RZ Smart Power Controller Service
     Loaded: loaded (/usr/local/lib/systemd/system/rz-smart-power-controller.service; enabled;
preset: enabled)
     Active: active (running) since Sat 2025-12-06 11:33:53 +07; 12min ago
   Main PID: 2606 (python3)
      Tasks: 1 (limit: 699)
     Memory: 25.1M (peak: 34.4M)
        CPU: 4.177s
     CGroup: /system.slice/rz-smart-power-controller.service
             └─2606 /usr/bin/python3 /usr/local/bin/gpio_api_c.py

Dec 06 11:36:42 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:42] "GET /relay/2/0 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay/1/0 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/1 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/3 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/2 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/6 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/5 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/4 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/7 HTTP/1.1" 200 -
Dec 06 11:36:44 localhost.localdomain python3[2606]: INFO:werkzeug:192.168.1.66 - -
[06/Dec/2025 11:36:44] "GET /relay_state/8 HTTP/1.1" 200 -
```

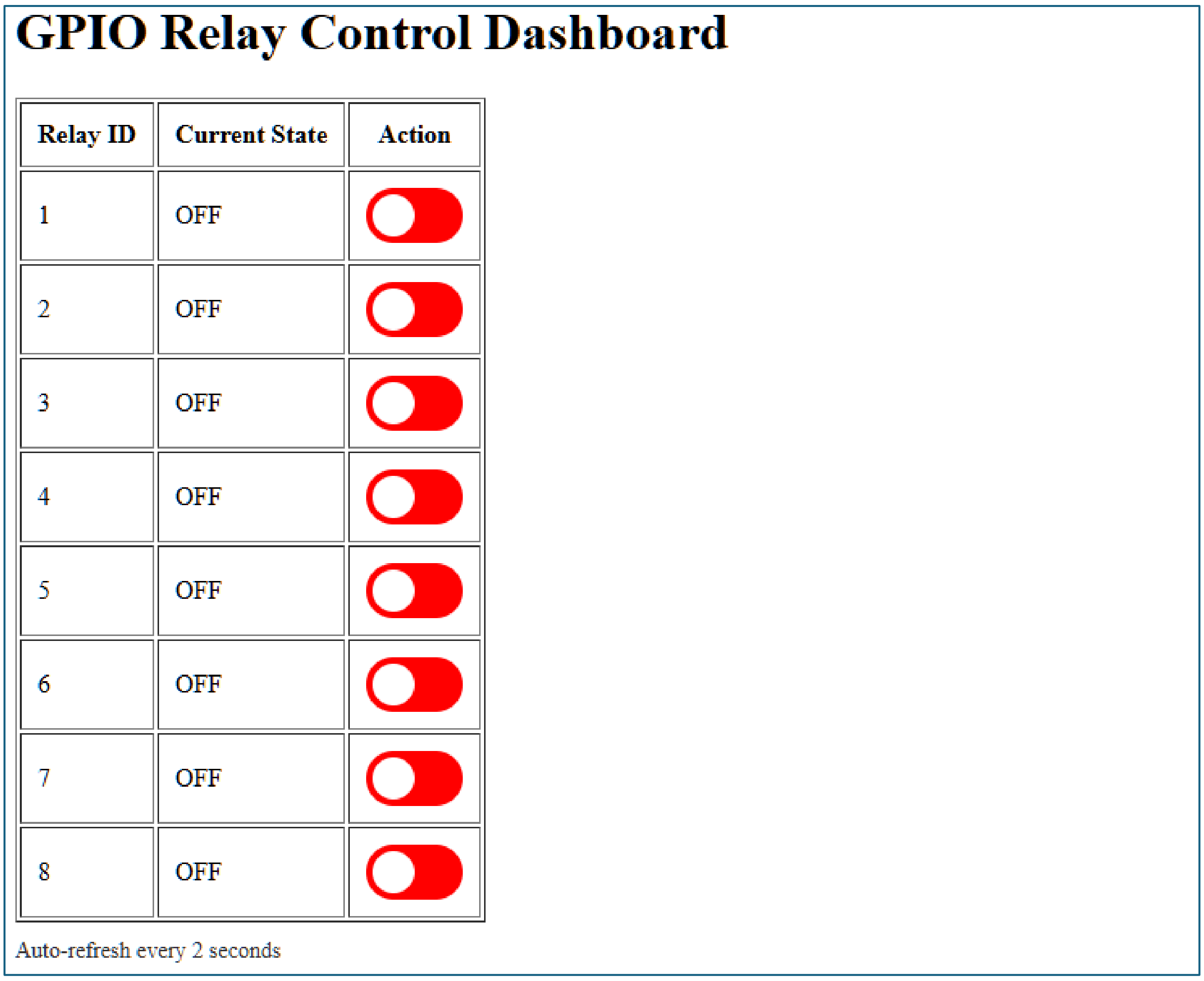
3.4.3 Access Web UI

From another system on the same network:

1. Open a browser.
2. Navigate to http://<board-ip>:5000
3. Use default credentials:
    - **Username**: admin
    - **Password**: password

    The actual username and password will be from the gpio_config.json file.

This will open a relay control interface with toggle buttons.



GPIO Relay Control Dashboard

Relay ID	Current State	Action
1	OFF	●
2	OFF	●
3	OFF	●
4	OFF	●
5	OFF	●
6	OFF	●
7	OFF	●
8	OFF	●

Auto-refresh every 2 seconds

Figure 4. Web UI Dashboard

4. Configuration

The Smart Power Controller tool uses a Python-based GPIO configuration that maps each relay number to a specific GPIO pin and its corresponding sysfs path. This section explains how to define or modify that mapping to match the board's hardware wiring. The reference hardware is the RZ/G2L-SBC.

4.1 RZ/G2L-SBC

This section describes the hardware-specific processes for the RZ/G2L SBC single-board computer. This board is a Pi-compatible board used for reference here.

4.1.1 Hardware Setup

The basic hardware setup consists of the following:

Figure 4. Web UI Dashboard shows the essential hardware setup. We expect a UART cable or an HDMI display to be available.

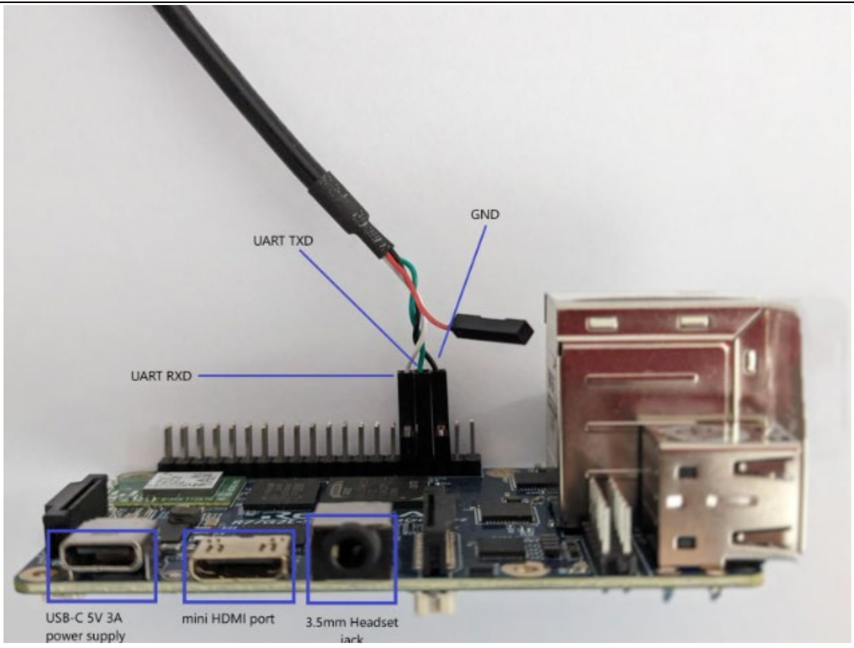


Figure 5. Essential Minimum Interfaces

4.2 File Location

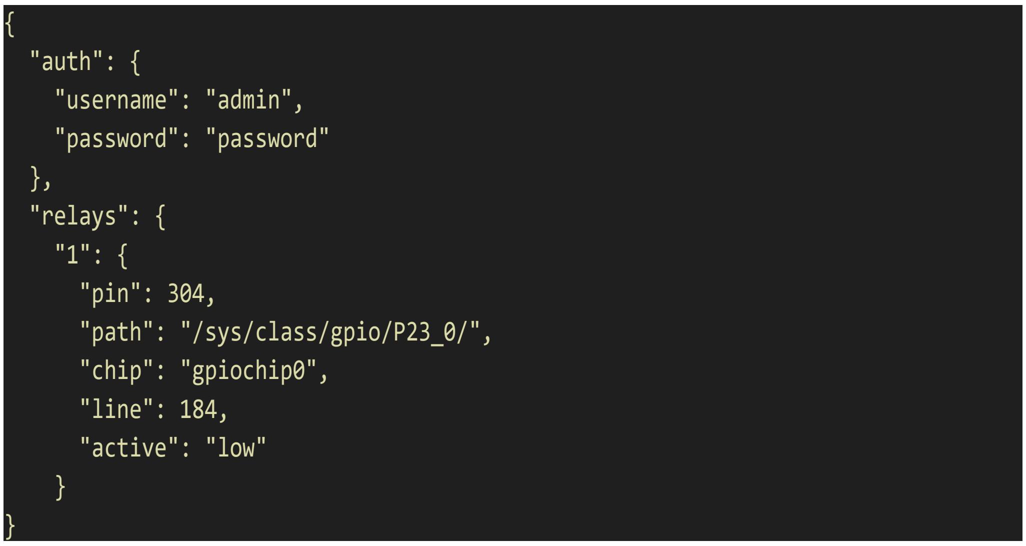
The default GPIO relay mapping is embedded in:

```
/usr/local/etc/rz-smart-power-controller/gpio_config.json
```

This file is imported at runtime by both the REST API server and the terminal tool. It contains a JSON map of relay configuration regarding the GPIO pins of the hardware, as seen in the Linux sysfs config. The entire data from the JSON file is loaded to a Python variable named GPIO_CONFIG, which defines the GPIO setting for the rest of the application.

4.3 Understanding the Configuration Format

The config is structured as follows:



```
{
  "auth": {
    "username": "admin",
    "password": "password"
  },
  "relays": {
    "1": {
      "pin": 304,
      "path": "/sys/class/gpio/P23_0/",
      "chip": "gpiochip0",
      "line": 184,
      "active": "low"
    }
}
```

Each relay entry includes:

- **pin**: Linux GPIO sysfs number used to control that relay in older kernels with sysfs
- **path**: Sysfs path pointing to the GPIO value file, used for toggling
- **chip**: GPIO chip ID used by the newer kernels running libgpiod
- **line**: GPIO line number corresponding to libgpiod entry
- **active**: Can be “low” or “high” based on whether the pin is active high or low for ‘on’ behavior.

This allows fine-grained control over boards that may not follow standard GPIO numbering.

Note: GPIOs must be exported and properly initialized in the system before use (via device tree or manual echo commands). If the pins are not configured as GPIO in the kernel device tree, they will fail to export and interact with them.

4.4 Mapping Reference Table

The table below shows the default GPIO mapping that comes configured with the application. These mappings can be changed based on user needs and pin availability, depending on the type of board used.

Table 3. Default Pin Mapping with Sysfs Path for RZ/G2L

Relay No	Sysfs GPIO Number	Sysfs Path	Libgpiod number	Libgpiod chip number
1	304	/sys/class/gpio/P23_0/value	184	gpiochip0
2	456	/sys/class/gpio/P42_0/value	336	gpiochip0
3	336	/sys/class/gpio/P27_0/value	216	gpiochip0
4	345	/sys/class/gpio/P28_1/value	225	gpiochip0
5	490	/sys/class/gpio/P46_2/value	370	gpiochip0
6	491	/sys/class/gpio/P46_3/value	371	gpiochip0
7	465	/sys/class/gpio/P43_1/value	345	gpiochip0
8	466	/sys/class/gpio/P43_2/value	346	gpiochip0

4.5 How to Modify GPIO Mappings

To change the GPIO assignments, the user only needs to modify the gpio_config.json file and load it on the target board. The specific method is flexible.

To change the relay-to-GPIO mapping on the target:

1. Ensure that you are logged in as root or another user who has admin and file access.
2. Edit the file:

```
# nano /usr/local/etc/at-powercycling/gpio_config.json
```

3. Update the "pin" and "path" fields as required.
4. Save and exit.
5. Rerun the GPIO setup script to reinitialize all the IO pins in the hardware.

```
# /usr/local/bin/gpio_setup.sh
```

Note: Always validate that the sysfs path exists on the board using the ls command. The GPIO interface is beyond the scope of this document. The user is expected to refer to the board's user manual for details on it.

4.6 Applying Changes

When the API server is already running, the user is required to restart it to apply the new configuration:

```
$ sudo pkill -f gpio_api_c.py
$ python3 /usr/local/bin/gpio_api_c.py
```

This will reload the Python dictionary with updated mappings.

5. Usage Instructions

The Smart Power Controller tool supports three modes of operation to control GPIO relays:

1. Web UI (Browser Dashboard)
2. REST API (HTTP-based control)
3. Terminal Interface (Shell-based local control)

5.1 Web UI

The Application comes with an intuitive and interactive web user interface for easy access and control of the pins. Follow the steps below to launch access to the UI.

5.1.1 Launching the Server

If the installation works fine, systemd would have started the service already and bound itself to one of the IP addresses. There is no need to explicitly start the service. However, standard system commands can control the service through the service name “rz-smart-power-controller”. Logs should also be visible on journald.

There might be times when there is a need for manual and limited control. This section tells how to stop the system service and directly start the web server.

```
# systemctl stop rz-smart-power-controller
# python3 /usr/local/bin/gpio_api_c.py
```

If successful, the given log below can be seen:

```
* Running on http://<0.0.0.0>:5000/ (Press CTRL+C to quit)
```

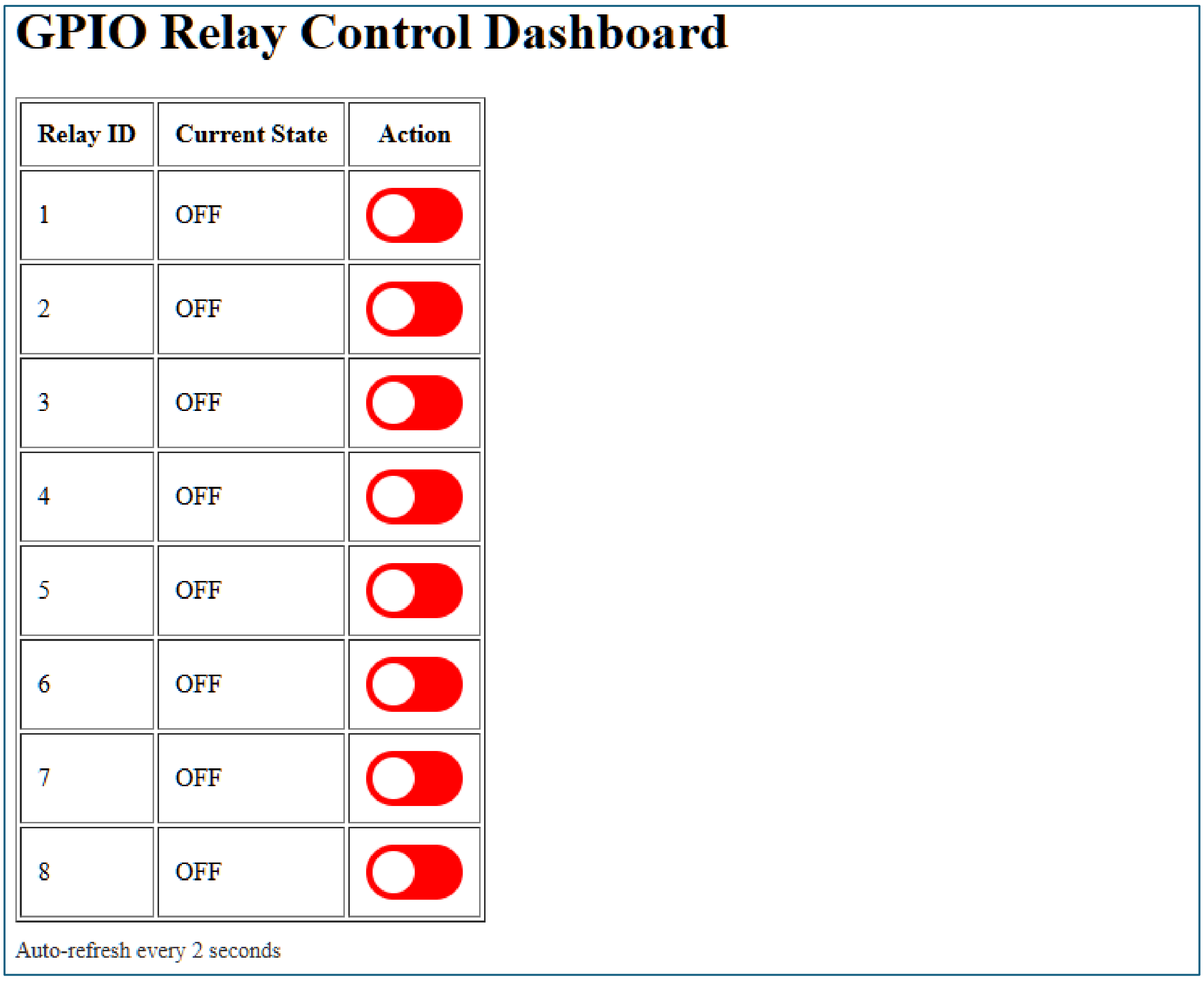
This launches a Flask web server on port 5000.

5.1.2 Accessing the UI website

Use the following URL on your PC browser:

```
http://<board-ip>:5000/
```

The figure below shows the main UI page with the pin control and the status of the pins



GPIO Relay Control Dashboard

Relay ID	Current State	Action
1	OFF	🟥
2	OFF	🟥
3	OFF	🟥
4	OFF	🟥
5	OFF	🟥
6	OFF	🟥
7	OFF	🟥
8	OFF	🟥

Auto-refresh every 2 seconds

Figure 6. UI Screen for the Application

5.2 Terminal Mode

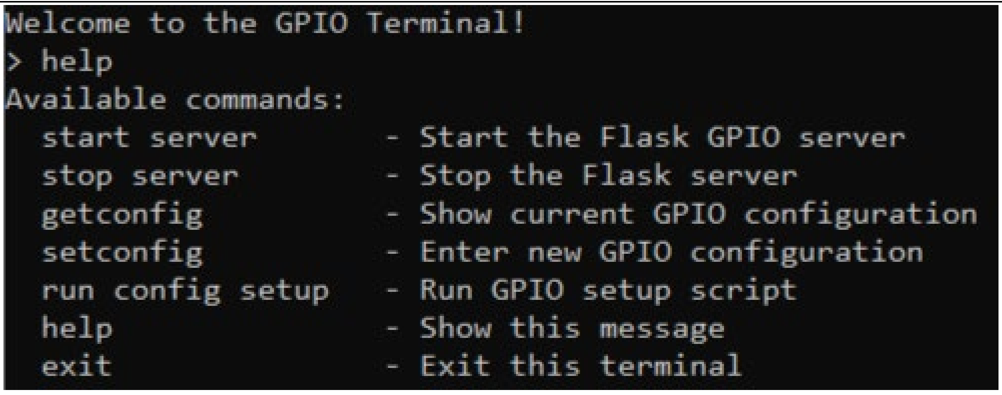
Terminal Mode offers a user-friendly method to access the application through the command line. Follow the steps below to launch and access the CLI mode

5.2.1 Starting Terminal Interface

Run:

```
$ gpio_terminal.sh
```

This will bring up the menu as shown below:



```
Welcome to the GPIO Terminal!
> help
Available commands:
  start server       - Start the Flask GPIO server
  stop server        - Stop the Flask server
  getconfig          - Show current GPIO configuration
  setconfig          - Enter new GPIO configuration
  run config setup   - Run GPIO setup script
  help               - Show this message
  exit               - Exit this terminal
```

Figure 7. GPIO Terminal

This mode is useful when:

- Users are logged into the board through a remote secure shell (SSH) connection.
- To do scripting control locally

Note: Type in' **run config setup'** to initialize the pins for the first time

5.3 REST API

The Flask app also exposes a REST API for controlling GPIO relays via HTTP.

5.3.1 Base URL

```
http://<board-ip>:5000
```

Flask uses **Basic Auth** (admin/password by default).

5.3.2 API Endpoints

Table 4 lists the available APIs to control the available pins. All relay IDs must be integers from 1 to 8.

Table 4. Base URL for available APIs

Endpoint	Method
/relay/<int:relay_id>/<int:state>	GET
/relay_state/<int:relay>	GET
/reload_config	POST

5.3.3 API Endpoints Usage

Example Commands

1. curl -u admin: password -X GET http://<board-ip>:5000/relay/1/1

   → Turns OFF Relay 1

2. curl -u admin:password123 -X GET http://<board-ip>:5000/relay/2/0

   → Turns ON Relay 2

6. Build

The complete source code is provided in the release package under the source directory. The user can build it on a Linux host environment.

6.1 Build system

The build system comprises the following:

build_deb.sh: single-shot script that performs all build and packaging.
src/CMakeLists.txt: CMake file that sets up the installation and packaging operations.

The project mostly consists of POSIX shell and Python code. It does not require a compilation step. However, it does require the underlying packages to reference the target architecture, which is handled automatically for arm64 (default) by build_deb.sh. It passes the architecture in the invocation of the appropriate commands.

6.2 Build commands

The build process is very simple. The user only needs to invoke the build_deb.sh script and it will create the final Debian package in a ‘build’ directory.

```
$ cd source/
$ ./build_deb.sh
Building .deb package for architecture: arm64
CMake Warning (dev) at /usr/share/cmake-3.30/Modules/GNUInstallDirs.cmake:253
(message):
  Unable to determine default CMAKE_INSTALL_LIBDIR directory because no
  target architecture is known.  Please enable at least one language before
  including GNUInstallDirs.
Call Stack (most recent call first):
  CMakeLists.txt:11 (include)
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Configuring done (0.0s)
-- Generating done (0.0s)
-- Build files have been written to: /home/preetam/workspace/PoC/Renesas-Smart-
Power-Controller/source/build
CPack: Create package using DEB
CPack: Install projects
CPack: - Run preinstall target for: RZ-Smart-Power-Controller
CPack: - Install project: RZ-Smart-Power-Controller []
CPack: Create package
CPack: - package: /home/preetam/workspace/PoC/Renesas-Smart-Power-
Controller/source/build/RZ-Smart-Power-Controller-2.1.0-Linux.deb generated.
Build complete. Package location: /home/preetam/workspace/PoC/Renesas-Smart-
Power-Controller/source/build
-rw-rw-r-- 1 preetam preetam 10338 Dec  5 23:02
/home/preetam/workspace/PoC/Renesas-Smart-Power-Controller/source/build/RZ-Smart-
Power-Controller-2.1.0-Linux.deb

To check package contents:
  dpkg -c /home/preetam/workspace/PoC/Renesas-Smart-Power-
Controller/source/build/*.deb

To check package metadata (including architecture):
  dpkg -I /home/preetam/workspace/PoC/Renesas-Smart-Power-
Controller/source/build/*.deb

To install (when running on target hardware):
  sudo dpkg -i /home/preetam/workspace/PoC/Renesas-Smart-Power-
Controller/source/build/*.deb
  sudo apt-get -f install  # installs any missing dependencies
```

The final Debian package will be available in the build directory:

```
$ cd build/
$ ls -1
CMakeCache.txt
CMakeFiles
cmake_install.cmake
CPackConfig.cmake
_CPack_Packages
CPackSourceConfig.cmake
install_manifest.txt
Makefile
RZ-Smart-Power-Controller-2.1.0-Linux.deb
rz-smart-power-controller.service
```

The CMake build system and platform architecture tuning are beyond the scope of this document. However, the code should be mostly self-explanatory.

7. Troubleshooting & Common Issues

This section helps to quickly identify and resolve common problems one may encounter when installing or using the Smart Power Controller Tool.

7.1 Installation Issues

This section includes possible issues during Application Installation and their corresponding solutions.

Table 6. Installation Issues and Fixes

Problem	Cause	Solution
dpkg: dependency problems prevent configuration	Required Python packages or system dependencies are missing	Run: sudo apt --fix-broken install
ModuleNotFoundError: flask	Python dependencies not installed	Install with: sudo pip3 install -r /etc/at-powercycling/requirements.txt
sudo: pip3: command not found	pip is not installed	Install pip: sudo apt install python3-pip

7.2 Network / UI Access Issues

This section includes possible issues with Network and UI access and their corresponding solutions.

Table 7. Network/UI Access Issues and Fixes

Problem	Cause	Solution
Web UI is not loading in the browser	Server not running or firewall blocking port 5000	Ensure gpio_api_c.py is running and try: http://<board-ip>:5000
Connection refused when accessing the API	Wrong IP or port, or service stopped	Check your board's IP with ip addr and verify process with ps aux | grep gpio_api_c.py
UI loads, but toggles don't work	Wrong GPIO mapping	Check /usr/bin/gpio_config.py matches your board's wiring

7.3 GPIO / Relay Issues

This section includes possible issues during GPIO/Relay usage and their corresponding solutions.

Table 8. GPIO/Relay Issues and Fixes

Problem	Cause	Solution
Relays don't toggle at all	GPIO not exported or wrong pin number	Verify sysfs path exists: ls /sys/class/gpio/ and adjust config
Permission denied error	GPIO access requires root	Run commands with sudo
Relay stays stuck ON or OFF	Incorrect wiring or pin direction	Ensure correct wiring, set direction: echo out > /sys/class/gpio/gpio<pin>/direction

7.4 REST API Issues

This section includes possible issues during API usage and their corresponding solutions.

Table 9. API Issues and Fixes

Problem	Cause	Solution
API works locally but not remotely	Board firewall or network restriction	Ensure both devices are on the same network and port 5000 is open
API call gives 404	Wrong endpoint format	Correct format: /relay/<id>/<state> where id is 1–8, and state is 0 or 1

7.5 General Issues

- Always check the board's IP address after network changes.
- Restart the server after editing gpio_config.py:

```
$ sudo pkill -f gpio_api_c.py
$ python3 /usr/local/bin/gpio_api_c.py
```

- For persistent usage, consider adding the server to the system startup using systemd.

Revision History

Rev.	Date	Description	
		Page	Summary
2.0.0	Sep.25.25	—	Initial release
2.1.0	Dec.05.25	—	Feature complete release.

Smart Power Controller - User Manual

Publication Date: Sep.25.25

Published by: Renesas Electronics Corporation

RZ Family/ RZ/G Series

Renesas Electronics Corporation

R12UZ0212EU0210