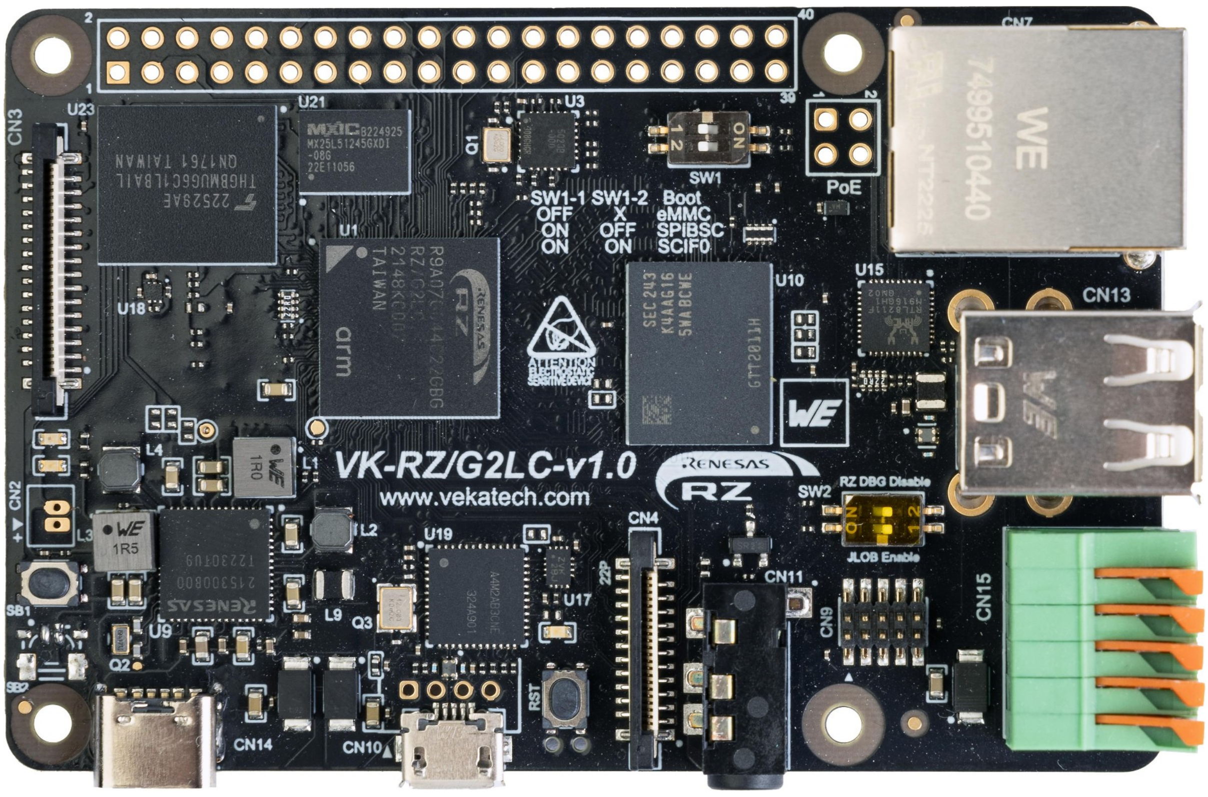
VK-RZ/G2LC How to run Flutter



VK-RZ/G2LC v1.0 Board

Content:

1. Introduction

VK-RZ/G2LC is industrial oriented board, compatible with Raspberry Pi 4 shields. It is based on Renesas **R9A07G044C22GBG**, **Dual ARM Cortex-A55 + Cortex-M33 MCU**. The main purpose of this manual is to show how to install Flutter on a host PC and run & debug applications remotely (on the board). For more info about this board, please read the full manual.

2. What is Flutter

Flutter is a cross-platform **S**oftware **D**evelopment **K**it which tremendously simplifies multi-platform applications development. Every app designed with Flutter has a single codebase, regardless of where it will be executed on (Android, iOS, web, Windows, macOS or Linux).

Direct development with Flutter on the VK-RZ/G2LC is quite slow, so a remote target debugging is the feature, that will be heavily used here.

2.1 Instalation

- Install Flutter eLinux on the **dev**elopment **PC** (Ubuntu 22.04 LTS in this case).

  → On the dev PC, get the tools you will need for the SDK:

  `sudo apt-get install git unzip curl clang cmake pkg-config`.

  → On the dev PC, get the tools you will need for the remote debugging:

  => If the target runs **debian-bookworm-vkrzg2lc-wl** image, install this set of tools:

  `sudo apt-get install sshfs gcc-aarch64-linux-gnu binutils-aarch64-linux-gnu`.

  => If the target runs **core-image-(weston/qt)-vkrzg2lc** image, install yocto's SDK.

  `wget https://vekatech.com/VK-RZ_G2LC_docs/Demo/get_SDK.sh`.

  `chmod +x get_SDK.sh` && `./get_SDK.sh`.

  → On the dev PC, Install **Flutter eLinux**:

  `git clone https://github.com/sony/flutter-elinux.git`.

  `sudo mv flutter-elinux /opt/`.

  `echo 'export PATH="$PATH:/opt/flutter-elinux/bin"' >> ~/.bashrc`.

  `source ~/.bashrc`.

  To complete the installation, make a call to flutter, let's say `flutter-elinux doctor`.

2.2 Configuration

- Configure Flutter eLinux to use VK-RZ/G2LC custom device.
  → On the dev PC, enable: `flutter-elinux config --enable-custom-devices`.
  → On the dev PC, get the VK-RZ/G2LC config file: `wget https://vekatech.com/VK-RZ_G2LC_docs/Demo/Flutter/.flutter_custom_devices.json`.
  → On the target vkrzg2lc, plug the ethernet & obtain the board's IP.
  → On the dev PC, open json file, edit the IP to match with the target's IP & save the file.

- Check if Flutter eLinux sees the **VK-RZ/G2LC** custom device.
  → On the dev PC, type `flutter-elinux devices` and you should see one of these:

```
eLinux (mobile) • debian-wayland • flutter-tester • Debian GNU/Linux 12 (bookworm) …
eLinux (mobile) • yocto-wayland • flutter-tester • Poky (Yocto …) 3.1.26 (dunfell) …
```

  along with the standard devices:

```
Linux  (desktop)  • linux           • linux-x64      • Ubuntu 22.04.4 LTS 6.5.0-41-generic
eLinux (desktop) • elinux-wayland • flutter-tester • Ubuntu 22.04.4 LTS 6.5.0-41-generic
eLinux (desktop) • elinux-x11        • flutter-tester • Ubuntu 22.04.4 LTS 6.5.0-41-generic
```

  depending on what is running on the target (debian or yocto) !

2.3 Run Test Application

- Create a sample Flutter application:
  → On the dev PC, type: `flutter-elinux create ~/sample`.

- Build sample Flutter application:
  → On the dev PC, cross compile the sample application:
  => If you building for debian:
  → Make a folder and mount the debian's root file system in it:
  `mkdir ~/rootfs` && `sshfs vkrz@<bord's IP>:/ ~/rootfs`.
  → Build the sample: `cd ~/sample` && `flutter-elinux build elinux --debug --target-arch=arm64 -target-compiler-triple=aarch64-linux-gnu -target-sysroot=$HOME/rootfs`.

=> If you building for yocto:

→ Make Yocto's SDK available:

`source /opt/poky/3.1.26/environment-setup-aarch64-poky-linux`.

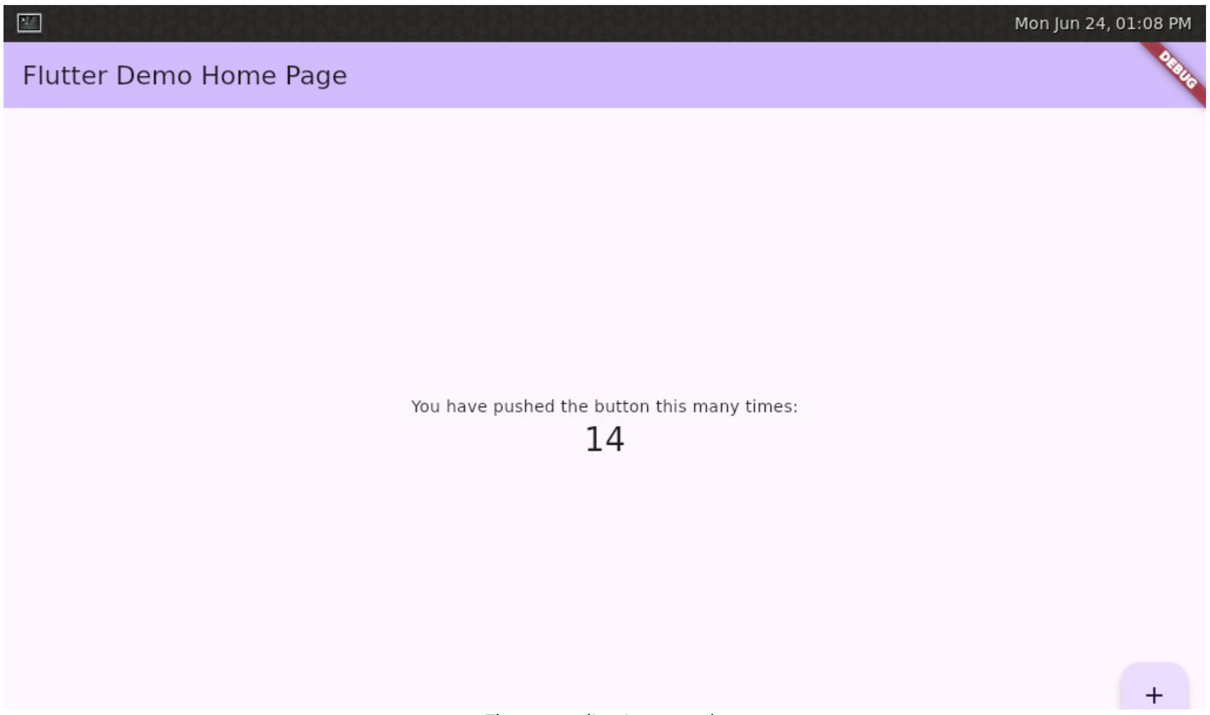
→ Build the sample: `cd ~/sample` && `flutter-elinux build elinux --debug --target-arch=$ARCH --target-compiler-triple=${TARGET_PREFIX%-} --target-sysroot=$SDKTARGETSYSROOT`.

- Run sample Flutter application:

  → On the dev PC, make sure passwordless ssh connection can be established and type:

  => If the target runs on debian : `flutter-elinux run -d debian-wayland`.

  => If the target runs on yocto : `flutter-elinux run -d yocto-wayland`.



You have pushed the button this many times:
14

Flutter application sample

2.4 Use Docker container (optional)

If you don't want to clog your system with additional software, you can use a ready made docker container with flutter-elinux preinstalled and everything it needs.

- Install Docker.
  → On the dev PC, you need to have Docker installed.
  → On the dev PC, make sure you can use Docker as reguler user.

- Build Docker Image.
  → On the dev PC, get the Docker file:
  `wget https://vekatech.com/VK-RZ_G2LC_docs/Demo/Flutter/Dockerfile`.
  → On the dev PC, build Docker image:
  `wget https://vekatech.com/VK-RZ_G2LC_docs/Demo/Flutter/build_docker`.
  `chmod +x build_docker` && `./build_docker`.

- Run Docker container.
  → On the dev PC, launch the image in a container:
  `wget https://vekatech.com/VK-RZ_G2LC_docs/Demo/Flutter/run_docker`.
  `chmod +x run_docker` && `./run_docker`.

3. Use VS code with Flutter

- Get VS Code.
  → On the dev PC, get the .deb package from here and install VS code:
  → On the dev PC, execute `sudo apt-get install ./<file>.deb`.
  → On the VS code, install **Flutter** Extension.
  → On the VS code, install **Dev Containers** Extension (only if you use docker container)

3.1 Use VS code with the native elinux

- Open the sample project.
  → On VS Code, locate the sample project and open it's folder in VS code's explorer.

- Setup VS Code to work with **dev PC's** flutter-elinux.
  → On VS Code, follow the guidance of flutter-elinux's creators & create **launch.json** file.

- Build the sample project
  → On the VS code, open terminal (View → Terminal) and execute the same commands as in **2.3**: `flutter-elinux build elinux …` (where … is different for yocto & debian !)

- Debug the sample project

  → On the dev PC, make sure you edited **.flutter_custom_targets.json** file, so the IP of the target board to match the IP in the file.

  → On the VS code, in it's terminal execute the same command as in **2.3**: `flutter-elinux run -d (debian/yocto)-wayland`.

  → On the VS code, once the sample is running, edit the launch.json file, so the **observatoryUri** to match with the **VM Service URL** from the terminal.

  → On the VS code, go to the **Run and Debug** tab and start the debugger (hit **|>**)

3.2 *Use VS code with the container's elinux*

- Run the container.

  → On the dev PC, execute `./run_docker` (from **2.4**).

- Make the sample project accessible for the container.

  → On the dev PC, copy the **sample** folder & place it where the `run_docker` is located.

- Open the sample project.

  → On VS Code, go to **Open a Remote Window** → **Attach to Running Container…** .

  → On VS Code, locate the sample project (the **parrent** folder of **run_docker** file) and open **sample** folder in VS code's explorer.

- Setup VS Code to work with **container's** flutter-elinux.

  → On VS Code, follow the guidance of flutter-elinux's creators & create **launch.json** file.

- Build the sample project (On the container only **building for Yocto** is setuped !)

  → On the VS code, open terminal (View → Terminal) and execute the build command: `flutter-elinux build elinux --debug --target-arch=$ARCH -target-compiler-triple=${TARGET_ARCH} -target-sysroot=$SDKTARGETSYSROOT`.

  If you want to build for **debian** you will have to install the SDK tools on the **container**: i.e. `sudo apt-get install git unzip curl clang cmake pkg-config`. After that the build commands are the same as **building for debian** in **2.3**:

- Debug the sample project.

  → On the dev PC, make sure you edited container's **~/.flutter_custom_targets.json** file (it is different from dev PC's .flutter_custom_targets.json), so the ip of the target board should match with the ip in the **ping** section of the .json file. The easiest way is to alter it with nano in the VS code's terminal.

  → On the VS code, in it's terminal alter the container's **~/.ssh/config** file with nano: It's HostName section should match with the target's board IP as well.

  → On the VS code, in it's terminal execute the command:

  `flutter-elinux run -v -d yocto-wayland`.

  → On the VS code, once the sample is running, edit the launch.json file, so the **observatoryUri** to match with the **VM Service URL** from the terminal.

  → On the VS code, go to the **Run and Debug** tab and start the debugger (hit **|>**)

If you want to debug for **debian** everything is the same except run command:

i.e. `flutter-elinux run -v -d debian-wayland`.

Revision overview list

Revision number	Description changes
0.1	Initial
0.2	Added chapter: Use VS code with Flutter