R5 Guide

by Brian Fraser
Last update: Apr 1, 2025

Guide has been tested on

BeagleBone (Target):	**Debian 12.8**
PC OS (host):	**Debian 12.8**

Tested on Linux Kernel 6.1

This document guides the user through

1. Install Zephyr and R5 tools
2. Compiling sample Zephyr Blinky code for the R5
3. Run the app on the R5

Table of Contents

Formatting

1. Commands for the host Linux's console are show as:
   ```
   (host)$ echo "Hello PC world!"
   ```
2. Commands for the target (BeagleBone) Linux's console are shown as:
   ```
   (byai)$ echo "Hello embedded world!"
   ```

Revision History

- Mar 28, 2025: Created guide targeting R5.
- Mar 28, 2025: Corrected typo in "tar" command.
- Apr 1, 2025: Removed confusing initial section about starting/stopping R5.

1. R5 Intro

The BeagleY-AI has two user-programmable R5 processors integrated to the main system on chip (SoC). It allows us to run bare-metal code (without Linux getting in the way) to give us deterministic and fast timing.

1. Linux uses the Remote Processor framework to communicate with the R5:
   ```
   (byai)$ ls /sys/class/remoteproc/
   ```
   - `remoteproc0/` is the DSP #1
   - `remoteproc1/` is the DPS #2
   - **`remoteproc2/` is the R5-MCU (we'll target this one)**
   - `remoteproc3/` is for R5-WKUP (power management)
   - `remoteproc4/` is for R5-MAIN (we could run code on this one).
2. We can see if the R5 is running using:
   ```
   (byai)$ cd /sys/class/remoteproc/remoteproc2/
   (byai)$ cat ./state
   ```

2. Install Zephyr Tools on Host

Zephyr is a small real-time Operating System (RTOS) that can be used to build custom executables to run on the R5. We will cross-compile the Zephyr examples from the host and download them to the R5.

You will need ~3+ GB free on the host in order to follow these directions.

These directions are based on the Zephyr Getting Started guide, combined with information on minimizing the amount of tools installed.

If you find these directions don't work for you, you should likely switch to the full getting started guide which will take about 16GB of space.

1. On the host, install the necessary tools.
   ```
   (host)$ sudo apt update
   (host)$ sudo apt upgrade
   (host)$ sudo apt install wget xz-utils python3 device-tree-compiler cmake git
   ```
2. Install the correct version of the SDK. You may need to get a version other than the linux-x86_64 if you are not building on a x86-64-bit. If you are having challenges with this, try the full Getting Started guide.
   ```
   (host)$ cd ~
   (host)$ wget
   https://github.com/zephyrproject-rtos/sdk-ng/releases/download/v0.17.0/
   zephyr-sdk-0.17.0_linux-x86_64_minimal.tar.xz
   (host)$ tar xvf zephyr-sdk-0.17.0_linux-x86_64_minimal.tar.xz
   (host)$ rm zephyr-sdk-0.17.0_linux-x86_64_minimal.tar.xz
   ```
3. Install the Toolchain (ARM 32 bit). Change x86_64 to aarch64 if you are on an arm processor (Apple silicon, for example)
   ```
   (host)$ cd ~/zephyr-sdk-0.17.0
   (host)$ wget
   https://github.com/zephyrproject-rtos/sdk-ng/releases/download/v0.17.0/
   toolchain_linux-x86_64_arm-zephyr-eabi.tar.xz
   (host)$ tar xvf toolchain_linux-x86_64_arm-zephyr-eabi.tar.xz
   (host)$ rm toolchain_linux-x86_64_arm-zephyr-eabi.tar.xz
   ```

4. Ensure your SDK folder is correct:

```
(host)$ du -hs zephyr-sdk-0.17.0/
1.1G    zephyr-sdk-0.17.0/
```

   - If your zephyr-sdk folder is not this big, it likely means you did not correctly wget both files and extract them.

5. Setup base project for Python environment

```
(host)$ sudo apt install python3-venv
(host)$ python3 -m venv ~/zephyrproject/.venv
(host)$ source ~/zephyrproject/.venv/bin/activate
```

   - You will have to run the 'source...' command above on each terminal you create in the future to build your Zephry projects.

6. Create a manifest file to pick just the parts of Zephyr that we need (keeps this part of the install to about 1.5GB instead of 6.8GB).

```
(host)$ cd ~/zephyrproject
(host)$ mkdir manifest
(host)$ nano manifest/west.yml
```

   - Into `nano`, copy the following contents.
     NOTE: **Formatting is important!** See website for link to a file you can copy-and-paste from (PDF messes up formatting).

```
manifest:
  remotes:
    - name: zephyr
      url-base: https://github.com/zephyrproject-rtos

  projects:
    - name: zephyr
      remote: zephyr
      repo-path: zephyr
      revision: main
      import:
        name-allowlist:
          - cmsis
          - libmetal
          - open-amp
          - hal_ti
```

   - Save the file with Ctrl-X, and then Y.

7. Setup the West build system and the Zephyr project

```
(host)$ cd ~/zephyrproject
(host)$ pip install west
(host)$ west init -l manifest
(host)$ west update                     # Installs 1.5GB of modules.
(host)$ west zephyr-export
(host)$ west packages pip --install
```

8. Verify that your install is at least the expected size:
```
(host)$ du -hs zephyr*
1.1G    zephyr-sdk-0.17.0
1.5G    zephyrproject
```
9. Troubleshooting:
   - If your install folders are not the correct size, ensure that you are changing to the correct directories as needed, and that you are downloading the two files via `wget`.
   - If you get an error:
     “Could not find a package configuration file provided by "Zephyr" with any of the following names: ZephyrConfig.cmake, zephyr-config.cmake”
     - Ensure you have created the ~/zephyrproject folder and correctly initialized it with `` `west init ...` ``
   - If you get an error “warning: HAS_CMSIS_CORE ... has direct dependencies CPU_AARCH32_CORTEX_R ...” and “error: Aborting due to Kconfig warnings”
     - Then ensure you have run the `west update` command.

3. Build and Running R5 Blinky

1. **Download the provided sample code for the R5**
   - Download the sample `blinky` app (or other provided samples) from the course website: https://opencoursehub.cs.sfu.ca/bfraser/solutions/433/15-R5/
   - Extract the ZIP file into wherever you normally build your code from. Likely best to have this in your folder that is under Git
   - You should see a folder for the unzipped project, and in that folder there should be a couple scripts and a `CMakeLists.txt` along with C code and a `prj.conf` file.

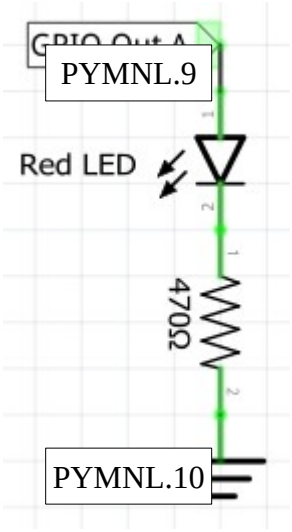
2. **Build R5 code on the host**
   - Open a terminal on the host and configure the environment for building:
     (host)$ source ~/zephyrproject/.venv/bin/activate
   - Change to the folder for the sample project (contains the `CMakeLists.txt` file).
   - Build the project:

```
(host)$ cmake -S . -B build -DBOARD=beagley_ai/j722s/mcu_r5f0_0
(host)$ cd build
(host)$ make
(host)$ cd ..
```

3. **Make executable available to target via NFS**
   - The above commands build `build/zephyr/zephyr.elf`
   - Copy this file to the shared folder (changing the firmware name to make it clear that it's built for the MCU R5 rather than the MAIN R5):

```
(host)$ cp build/zephyr/zephyr.elf ~/cmpt433/public/r5/zephyr_mcu.elf
```

4. **Wire up LED on target GPIO7**
   - `blinky` code toggles GPIO7 ("CE1") which is hat pin 26
     - GPIO7 is gpiochip0, pin 9
     - On the Zen Hat GPIO7 it is mapped to header PYMNL.9 (second from the top).



   - Wire the LED as follows:
     - Power off your target!
     - Connect PYMNL.9 (GPIO7) to your bread board.
     - To this, connect the long-end an LED.
     - To the short end of the LED, connect a resistor.
     - To the other end of the resistor, connect a wire to PYMNL.10 (to ground).
   - Power on your board and test your wiring is correct via the target's Linux prompt:
     - Turn LED on:

```
(host)$ gpioset gpiochip0 9=1
```

     - Turn LED off:

```
(host)$ gpioset gpiochip0 9=1
```

5. **Install on target**
   - Mount the NFS folder on the target and change to the R5 folder:
     `(byai)$ cd /mnt/remote/r5`
   - Copy the firmware to the necessary folder:
     `(byai)$ cp zephyr_mcu.elf /lib/firmware/`
   - Load the firmware on the MCU R5:

`(byai)$ echo zephyr_mcu.elf| sudo tee /sys/class/remoteproc/remoteproc2/firmware`

     - Done using `sudo tee` so that we can write to any files owned by root.
   - Start the MCU R5 running our code:
     `(byai)$ echo start | sudo tee /sys/class/remoteproc/remoteproc2/state`

6. **Run any necessary commands on the host to configure pins, such as**

```
(byai)$ gpioset gpiochip0 9=1
(byai)$ gpioget gpiochip0 10
```

   - Ideally this step should not be needed. However, as of March 2025 we have not figured out how to correctly configure the GPIO pins for the MCU R5 from within the R5 firmware. Instead this work-around does the pin configuration from Linux.

7. **Automating these steps**
   - As a developer, you must automate your process!
   - The sample projects have a `r5_mcu_build.sh` script which will build the project and copy the .elf executable to the NFS folder.
   - The sample projects also include `load_r5_mcu.sh` which should be run on the target to load the firmware. This script is copied to the NFS folder by the build script above.

8. **Reloading new firmware**
   - Currently (March 2025) there is no known way to stop the R5 and load new code without rebooting. So, reboot your target using:
     `(byai)$ sudo reboot -f`
   - Note: we would like to be able to stop the processor using
     `(byai)$ echo 'stop' | sudo tee ./state`

     However, this currently does not work.

9. **Troubleshooting**
   - If you echo '`start`' to `/sys/class/remoteproc/remoteproc2/state` and it returns:
     - '`No such file or directory`', it likely means that the string you wrote to to the `/sys/class/remoteproc/remoteproc2/firmware` file dose not match a file found in the folder `/lib/firmware/`.
     - '`Invalid argument`' may mean that the R5 is in the incorrect state for the operation requested, or that the file you are trying to load is not the correct format. It may have been built for the incorrect target R5, or have some other issue with building a correct file.
       - Sometimes the R5 can get in an incorrect state and then trying to load working firmware may fail with the 'Invalid argument' error even if the firmware you are trying to load is fine. If you see this error, try rebooting the target and trying to load

the firmware again.

- If the `west.yml` file is incomplete, it may allow you to build executables however they may fail to load with the 'Invalid argument' error. Double check that you have the full `west.yml` file as shown above. If you are trying to do something more than the sample R5 projects and get this error, then ensure that you can build the default `blinky` examples. If needed, you can install the full `zephyrproject` (takes ~8Gb!). Instead of running "`west init -l manifest`", just run "`west init`".

- If you get error "West not found", it may man you have not run
  `(host)$ source ~/zephyrproject/.venv/bin/activate`

4. R5 Coding and Troubleshooting Ideas

Programming on the R5 has very low visibility into its internal state. Therefore, you should:

1. Write a small bit of R5 code.
2. Test the code works (say by flashing an LED...)

Here are some common issues:

- In VS Code, if you see error-bars under the #include statements for the R5 (Zephyr specifically), those can be ignored because it does not know were to find the include files.
- When running your Linux program which tries to use shared memory (`mmap()`), if you get a permission denied on `/dev/mem` error, then run your app using `sudo`.
- If the GPIO input (a button) or output (an LED) is not working, try running the gpioset or gpioget commands on the target under Linux to set the data direction for the pin and configure it for GPIO.
- If your R5 code seems to start up and run but then does not work, try:
  - Change your R5 code to just toggle an LED forever using a busy wait (k_busy_wait(500000)) to ensure that your code is compiling and running.
  - If using mmap(), ensure you are not using a pointer to a struct (strangely seems to hang the R5 as of March 2025).
  - Ensure you are using k_busy_wait() for your delays rather than k_msleep().
- If you are not sure if your code is even compiling, add a #error definition to the code and try compiling. You should see a compile time error to prove your code is being compiled.
- Other debugging steps that are useful:
  - If possible, connect to the UART with a Raspberry Pie Debug Probe to view text output and use printf() statements to show what is happening.

Ideas for Future Improvements

- Shutting down the R5 to load new firmware.
  - Currently (March 2025) we must reboot the board to be able to reload new firmware.
  - Linux remoteproc requires a core to gracefully shutdown in order to reload firmware:
  - TI description of graceful shutdown: https://dev.ti.com/tirex/explore/node?a=7qm9DIS__LATEST&node=A__AdAyuKWUWVV5j4wBc7C6XA__AM64-ACADEMY__WI1KRXP__LATEST
  - TI code for graceful shutdown https://dev.ti.com/tirex/explore/node?a=7qm9DIS__LATEST&node=A__AVt2qZLTY3BgKr3D4YIv8w__AM64-ACADEMY__WI1KRXP__LATEST
  - Zephyr example for dong RP MSG https://github.com/zephyrproject-rtos/zephyr/tree/main/samples/subsys/ipc/rpmsg_service
- For debugging, it would be great if we could get a serial port loop-back from the R5 to Linux so we could view serial output.
  - Idea: Have R5 use hardware serial port (mapped to the UART header on the Zen Hat;

already works); Have Linux load an DT overlay for activating an unused pin as a UART Receive pin. May need to use bit-bang'd UART.
    - Could we use edge-triggered notifications from the gpiod library to "easily" implement bit-bang UART Rx?
- R5 MCU testing without custom LED:
    - Can we map the Zen Hat's LED, which is connected to the main domain, to the MUC R5?
- Other
    - Why does Zephyr hang on things like k_msleep()?
    - How can the MCU R5 correctly configure GPIO pins without needing Linux to step in?