linaro Connect 2025

Das U-Boot testing on a Linaro Automation Appliance (LAA)

Das U-Boot

linaro Connect

Das U-Boot is “a boot loader for Embedded boards based on PowerPC, ARM, MIPS and several other processors, which can be installed in a boot ROM and used to initialize and test the hardware or to download and run application code.”

U-Boot is highly relevant to the ARM ecosystem as it provides a well supported environment for a developer to deliver the code needed to boot a board in a reliable and flexible manner.

The pervasive use of U-Boot means that it now supports many CPUs, SOCs etc. and so has a need to not only be tested on emulators and virtual platforms but also on hardware to ensure the greatest possible code coverage.

Linaro Automation Appliance

A talk on the LAA was given at Linaro Connect Madrid 2024 which can be found on https://resources.linaro.org/

https://resources.linaro.org/en/resource/fwWCArmwQyjvLgnFMdZwwT

In essence the LAA gives you a predictable and repeatable way to interact with your device under test (DUT). This is normally done remotely as there is no need for physical access to the LAA or DUT when properly integrated.

Bakfleet

Bakfleet is the web application that is used to control each LAA.

https://lavacloud.io/

U-Boot testing

There are 3 main groups of tests that are run on the project's CI infrastructure.

- Does it build without error? This is run on many but not all supported platforms.
- Unit tests written in 'C' that are linked into the U-Boot code.
- Python tests using pytest. The unit tests can also be run through pytest.

U-Boot testing platforms

- QEMU
- Sandbox
- Hardware

U-Boot test hooks

linaro Connect

U-Boot allows the use of what is known as test hooks. This allows the generic test infrastructure in the main U-Boot code base to work with your local hardware configuration. You just need to provide the snippets of code to perform the operations such as power on or off the DUT and update wherever it is that holds the copy of the U-Boot binary used to boot the DUT.

It is likely that there will already be an example of the needed code snippets for controlling your DUT that will only need simple changes in order to work for your specific configuration.

U-Boot test hooks are maintained in a separate repository to the main U-Boot source code.

Initial trial setup without LAA

linaro Connect

Setup of Raspberry Pi 4B on my desk

- Power controlled by a radio controlled mains socket. Radio transmitter was attached to another Raspberry Pi which would respond to files copied to it via scp.
- Board reset controlled by a USB relay attached to my laptop.
- Console access to the Raspberry Pi via USB serial cable.
- Network attached to the local LAN.
- Tftp server on my laptop
- Bought SD-Mux to use for flashing boot image but never implemented this as the LAA turned up. The SD-Mux would have required another USB connection to my laptop.
- Implement necessary U-Boot test hooks

The above did allow to run the U-Boot pytest tests on the Raspberry Pi 4B but this was obviously never going to scale.

Overview of current implementation

linaro Connect

A Gitlab CI job builds U-Boot for the target platform and integrates the resulting binary into a disk image suitable for that platform.

This build also produces the helloworld.efi file used by some unit tests.

Finally the Gitlab CI job creates the LAVA job file and submits it to the relevant LAVA instance. The LAVA job is dynamically created so that it can contain information about the Gitlab CI job such as the job ID and also the hash of the commit that was used to build U-Boot. This allows the LAVA job to access the Gitlab CI job artifacts and to checkout the same version of the U-Boot source code.

Overview of current implementation (cont)

linaro Connect

The LAVA job downloads the disk image from Gitlab and uses it to boot the RPi4B into the U-Boot command line using a serial port to access the console. For the first test section LAVA directly runs some simple U-Boot commands via the console.

The second test section launches a docker container and passes control to a script that is running in the docker container. At this point LAVA is connected to the docker container and not to the DUT.

The script running in the docker container can now checkout the correct commit of the U-Boot source and perform a build so that all the test code is correctly set up.

Now the script runs pytest which will connect to the console of the DUT and run all applicable Python tests including running the unit tests.

The docker container needs to be used so LAVA can have its control connection active to ‘something’ while pytest can be connected to the DUT’s console. The docker container also contains the necessary tools and environment for testing U-Boot.

Targeting just my LAA/DUT

linaro Connect

Initially this was done by adding a tag to my LAA in Bakfleet. This has to be done by someone with admin access. This took a little while to achieve as it needed the tagging support in Bakfleet to be completed.

Now it seems that the tag has been removed from my LAA and now the job just matches on the LAA's name, so you may be able to target just your LAA/DUT without needing an admin to make changes to your LAA in Bakfleet.

Creating the RPi4B disk image with U-Boot

linaro Connect

This step was not specific to the use of the LAA although I had to complete it before I could automate the passing of the binary built by the CI job in Gitlab.

Most of the other boot files can be obtained from

https://github.com/raspberrypi/firmware/releases/download/1.20241126/raspi-firmware_1.20241126.orig.tar.xz

You may use whatever release suits your needs, just replace the '1.20241126' with the appropriate release.

You will only need a few of the included files, please see

https://github.com/raspberrypi/documentation/blob/develop/documentation/asciidoc/computers/configuration/boot_folder.adoc

Raspberry Pi 4B config.txt

linaro Connect

Along with the firmware files and the built u-boot.bin you also need a minimal config.txt file that will ensure that U-Boot is entered.

```
kernel=u-boot.bin
arm_64bit=1
arm_boost=1
[cm4]
otg_mode=1
[all]
enable_uart=1
```

Create the boot image with guestfish

```
disk-create disk.img raw 32M preallocation:sparse
add-drive disk.img
launch
part-init /dev/sda msdos
part-add /dev/sda primary 32 -1
part-set-mbr-id /dev/sda 1 14
part-set-bootable /dev/sda 1 true
mkfs fat /dev/sda1 label:boot
mount /dev/sda1 /
copy-in ${UBOOT_TRAVIS_BUILD_DIR}/boot_img/config.txt /
copy-in ${UBOOT_TRAVIS_BUILD_DIR}/boot_img/u-boot.bin /
copy-in ${UBOOT_TRAVIS_BUILD_DIR}/boot_img/bcm2711-rpi-4-b.dtb /
copy-in ${UBOOT_TRAVIS_BUILD_DIR}/boot_img/fixup4.dat /
copy-in ${UBOOT_TRAVIS_BUILD_DIR}/boot_img/start4.elf
```

Using API tokens

If your Gitlab repo is not public then you may want to use an API token to access it from the LAVA job. xxxxx is the Gitlab project ID. gitlab-private is the name of the access token as stored in LAVA.

```
- deploy:
        timeout:
                minutes: 5
        to: usbg-ms
                image:
                        url:
https://gitlab.com/api/v4/projects/xxxxx/jobs/${CI_JOB_ID}/artifacts/disk.img.gz
                        compression: gz
                        headers:
                                PRIVATE-TOKEN: gitlab-private
```

Docker container default user

The default user of the docker container provided by U-Boot is ‘uboot’. However LAVA creates directories for use in the container with ownership of ‘root’. This meant rebuilding the docker container with ‘root’ as the default user. Then the yaml file for the docker test shell changes ownership of the directories to be ‘uboot’ before running the main script as the user ‘uboot’.

Passing information into the container

linaro Connect

I used a separate access token to read the code from Gitlab and now this needed to be passed to the docker container as well as the commit hash.

```
- test:
        docker:
                image: linaro/uboot-arm64
        timeout:
                minutes: 15
        definitions:
        - repository: 'https://\${GITLAB_USER}:\${GITLAB_TOKEN}@gitlab.com/.../lava-tests.git'
                from: git
                path: laa-docker-tests.yaml
                name: laa-docker-tests
                parameters:
                        TEST_COMMIT: ${CI_COMMIT_SHA}
secrets:
        GITLAB_USER: gitlab-user
        GITLAB_TOKEN: gitlab-private-code
```

Interacting with the DUT from the container

linaro Connect

The docker container in use comes from the U-Boot project and knows nothing about an LAA so LAVA will provide environment variables for the needed commands. These commands come from the board dictionary in LAVA. This did not happen at first but the LAVA team quickly sorted the problem out.

```
job environment:
  - ANDROID_SERIAL=''
  - LAVA_BOARD_ID=''
  - LAVA_CONNECT_COMMAND='telnet localhost 2000'
  - LAVA_HARD_RESET_COMMAND='laacli power 3v3 on && laacli power 5v reset'
  - LAVA_POWER_ON_COMMAND='laacli power 3v3 on && laacli power 5v on'
  - LAVA_POWER_OFF_COMMAND='laacli power 5v off && laacli power 3v3 off'
```

Then it was found that the overlay for the laacli command etc was not present but that again was quickly sorted. These variables are used in the U-Boot test hooks.

Hostname of the container

The U-Boot test hooks are located by the scripts with the help of the hostname of the machine they are run on. Unfortunately LAVA does not provide a way to set the hostname of the container.

The solution to this was to add links in the test hooks repo from the container's host name to the directory with the relevant hooks.

Console access to DUT

linaro Connect

Console access to the DUT is provided by an instance of ‘ser2net’ running on the LAA.

There were two issues with this, firstly the value of LAVA_CONNECT_COMMAND='telnet localhost 2000' is obviously not correct for use within the docker container as that is not where ‘ser2net’ is running. This meant replacing ‘localhost’ with the suggested ‘lava-worker.internal’, once the DNS was made to resolve that address of course.

Secondly the invocation of ‘ser2net’ on the LAA needed to be modified so that it would listen on the docker virtual network IP address of the LAA as well as the ‘localhost’ address which is the only one it was listening on initially.

Files for http or tftp access by DUT

linaro Connect

If your DUT testing requires one or more files to be available for download over tftp or http then you can arrange for these files to be placed in a predictable folder using the ‘downloads’ deploy action in the LAVA job.

If the files are not in an archive they will be located in ${LAVA_JOB_ID}/downloads/common/

If the files are in an archive then there will be an extra element in the path which is the name of the download in the LAVA job, ${LAVA_JOB_ID}/downloads/common/gitlab_files/

```
- deploy:
    to: downloads
    images:
        gitlab_files:
            url: https://gitlab.com/api/v4/projects/.../files_from_gl.tar.xz
            compression: xz
            archive: tar
```


linaro
Connect
2025
Thank You!