YVR18-222: Testing Arm-with-Arm with Lava and 96Boards

Loic Poulain <loic.poulain@linaro.org>
Agenda

- What is LAVA?
- What is 96Boards?
- Yet Another Mini Lava Lab
- Existing Solutions
- Testing 96Boards with 96Boards
- LAVA Setup
- Adding a (96)Board
- Additional Infrastructure
- Architecture Overview
- Time for Testing
- Achievements
- Next Stages
What is LAVA?

- **Linaro Automated Validation Architecture**
- LAVA is a continuous integration system for deploying operating systems onto devices for running tests
  - Boot testing
  - System level testing
  - Power consumption testing
  - etc.
- Organized in a server-dispatcher model:
  - The lava-server (master) schedules jobs, administers devices and stores results
  - The lava-dispatcher (worker) processes the test jobs and deploys images on supported development boards
- Scalable, distributed
What is LAVA? Job in a nutshell

- **Deploy image:** prepare the device and infrastructure
  - Retrieve test image(s) (file, ftp, jenkins...)
  - Retrieve test cases (git, inline...), generate test scripts, install tests on test images (overlay)
  - Setup tftp/nfs server, flash image via fastboot...

- **Boot device:** boot test image on the DUT
  - Establish connection to the device (uart console)
  - Trigger device reboot/power-on (via command, power device unit, etc)

- **Test:**
  - Execute test(s) on the device
  - Parse output of the tests (success, failure)
  - Submit results
What is 96Boards?

- Range of **open specifications** defining standard form-factors for **SoC-agnostic** development platforms (CE, IoT, Enterprise editions).
- Suitable for evaluation, rapid prototyping, hobbyist projects...
- **Standardized expansion buses** for peripheral I/O allowing wide range of compatible add-on **mezzanine boards**
- Multiple vendor boards: DragonBoard 410c/820c (Qualcomm), Hikey (HiSilicon), Rock960 (Rockchip), Ultra96 (Xilinx), etc.
Yet Another Mini Lava Lab

Create my own automated testing home LAVA lab in order to perform wireless functional tests on 96Boards devices (Consumer/IoT boards).

Tests are performed on release, snapshot or personal builds/images.
Additional Goals

- **Keep the Lab simple**: no soldering, extra cables or board modification
- **Easily reproducible**: rapid setup, hardware agnostic
- **Autonomous**: All-in-One, do not rely on external infrastructure
- **Compact**: Tiny lab standing on a desk
- **Low cost**: less than $300
- **Fully running on ARM**: why not?
Existing solutions

https://github.com/chase-qi/lift-remote-lab

https://bootlin.com/blog/tag/kernelci/

https://baylibre.com/baylibre-lava-box/

https://baylibre.com/baylibre-lava-box/
Testing 96Boards with 96Boards

- **Choice:** *DragonBoard 820c*
- Powerful enough for All-in-One Lava master+worker instance (4-core, 2GB ram)
- Compact (100mm x 85mm)
- Ethernet controller for reliable web access/admin
- Possible storage extension via PCIe
- Available I/Os for control (e.g. relays) or/and testing (I2C, SPI...)
- Wireless chip (BT/WIFI) for DUT radio testing (no external infrastructure)
LAVA Setup: Dockerizing LAVA

- **Container**: A package of applications, libraries, and files isolated from the host system
- **Docker**: A command-line tool for programmatically defining the contents of a Linux container (via Dockerfile) which can be versioned, reproduced, shared, and modified easily

=> Using Docker to provide a lava-master and lava-worker package recipes
LAVA Setup: Dockerfile

```bash
# From debinstretch-backports
# Update package list and upgrade
RUN apt -q update
# Install prerequisites
RUN DEBIAN_FRONTEND=noninteractive apt-get -q -y install \\
    wget \\
    gnupg \\
    apt-transport-https
# Add Linaro Lava repository to apt sources, Register auth key
RUN echo "deb https://images.validation.linaro.org/production-repo stretch-backports main" >> /etc/apt/sources.list.d/lava.list
RUN wget -qO - https://images.validation.linaro.org/staging-regex-apt-key.asc | apt-key add -
# Add Linaro buster repository, some arm64 tools not available in standard source (e.g. sing2)
RUN echo "deb http://obs.linaro.org/linaro-overlay-buster/buster/" >> /etc/apt/sources.list.d/linaro.list
RUN wget -qO - http://obs.linaro.org/linaro-overlay-buster/buster/Release.key | apt-key add -
RUN apt -q update
# Install lava-dispatcher and required tools
RUN DEBIAN_FRONTEND=noninteractive apt-get -q -y install \\
    lava-dispatcher \\
    qemu-system \\
    qemu-system-arm \\
    qemu-system-1386 \\
    e2fsprogs \\
    ttftpd-hpa \\
    nfs-kernel-server \\
    python-netifaces \\
    sing2 \\
    tini \\
    microcom \\
    python-pip &
    pip install -pre -U pyrocd
COPY scripts/start.sh / 
COPY scripts/stop.sh / 
COPY scripts/git.sh / 
COPY config/ttftpd-hpa /etc/default/ttftpd-hpa 
COPY config/ser2net.conf /etc/ser2net.conf 
COPY config/lava-dispatcher-nfs.exports /etc/exports.d/lava-dispatcher-nfs.exports 
COPY config/lava-dispatcher.conf /etc/lava-dispatcher/lava-dispatcher.conf 
COPY config/lxc-net /etc/default/lxc-net 
COPY config/lxc-default /etc/lxc/default.conf 
COPY scripts/daqboard-ftdl.sh / 
COPY scripts/nitrogen-ftdl.sh / 
COPY patches /patches
# Apply patches
RUN find /patches -type f -name '*.patch' -print0 | xargs -0 -r patch -p 0 -p1 
RUN echo "MASTER_URL='http://lava-master:5556'" >> /etc/lava-dispatcher/lava-slave 
RUN echo "LOGGER_URL='http://lava-master:5555'" >> /etc/lava-dispatcher/lava-slave 
CMD [start.sh] & & sleep infinity
```

This mainly consists in packing installation steps from LAVA documentation

https://validation.linaro.org/static/docs/v2/installing_on_debian.html
LAVA Setup: One command setup

docker-compose is a tool* for defining and running multi-container Docker applications. It allows to build an run all our containers (master, worker…) with one command:

$ docker-compose up

Once all dependencies and layers are built, containers are instantiated.

$ docker ps

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>STATUS</th>
<th>NAMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>4e85278b304d</td>
<td>lava-master</td>
<td>Up 38 seconds</td>
<td>lava-master_1</td>
</tr>
<tr>
<td>ab9deb933337</td>
<td>lava-worker</td>
<td>Up 38 seconds</td>
<td>lava-worker_1</td>
</tr>
</tbody>
</table>

* https://docs.docker.com/compose

```
docker-compose.yml

version: "3"

services:
  lava-master:
    hostname: lava-master
    restart: always
    build:
      context: ./lava-master
      args:
        admin_username: "lola"
        admin_password: "password"
        admin_email: "admin@localhost.com"
    ports:
      - "80:80" # http
      - "5555:5555" # logger
      - "5556:5556" # master
    network_mode: "host"
    extra_hosts:
      - "worker0:127.0.0.1" # local-instance

  lava-worker:
    hostname: worker0
    dns: 127.0.0.1
    restart: always
    build:
      context: ./lava-worker
      args:
        board: "db828c"
    volumes:
      - /dev:/dev
      - /boot:/boot
      - /lib/modules:/lib/modules
      - /sys/class/gpio:/sys/class/gpio:rw # Power control via GPIO
      - /tmp:/tmp
      - /var/log:/var/log/
      - /var/cache:/var/cache/
      - /sys/fs/cgroup:/sys/fs/cgroup:rw # lxc
      - /sys/devices/virtual/net:/sys/devices/virtual/net:rw # Lxc network Interface
      - /images:/images:ro # temp, for local image download
    privileged: true
    ports:
      - "69:69" # tftp
      - "2049:2049" # nfs4
    network_mode: "host"
    extra_hosts:
      - "lava-master:127.0.0.1" # local-instance
```
LAVA Setup: Welcome to LAVA

Welcome to LAVA

LAVA is an automated validation architecture primarily aimed at testing deployments of systems based around the Linux kernel on ARM devices, specifically ARMv7 and later. The current range of boards (device types) supported by this LAVA instance can be seen on the scheduler status page which includes details of how many boards of each type are available for tests and currently running jobs.

LAVA components

- **Results** - viewing results of tests run by you or others.
- **Scheduler** - jobs are scheduled on available devices and the scheduler pages allow you to view current and past jobs as well as submit new jobs.
- **API** - information on how to interact with LAVA and export data from LAVA using XMLRPC.
- **Help** - documentation on using LAVA, worked examples and use cases, developing your own tests and how to administer a LAVA instance of your own.
- **Profile** - you are logged in as folc. Your profile provides access to jobs you have submitted or marked as favourites, your bundle streams and your filter or image report subscriptions.

Guides to LAVA

- Introduction to LAVA
- Administering a LAVA instance
- More about LAVA & Linaro
- Developing LAVA

Test using LAVA

- Use cases and worked examples.
- Logging into a LAVA device.
- Writing a LAVA test definition.

Your submissions

Your jobs

Your favourite jobs

Your results
Adding a Board: hardware requirements

In order to be autonomous, Lab needs to:

- Control power of the board (on/off)
  - Switched PDU
  - Controllable ATX power supply (PS_ON pin)
  - Relays
- Access devices debug console
  - UART/FTDI
- Provision and boot images (kernel, bootloaders)
  - Fastboot (usb)
  - U-boot (tftp/nfs...)

=> defined by board **device-type** and **device** configuration files, known as **device dictionary**.
Adding a Board: Device dictionary

Example: Dragonboard-410c from validation.linaro.org*:

{% extends 'dragonboard-410c.jinja2' %}
{% set connection_command = 'telnet localhost 7005' %}
{% set adb_serial_number = '1f58033c' %}
{% set fastboot_serial_number = '1f58033c' %}
{% set hard_reset_command = ['/usr/local/lab-scripts/usb_hub_control -p 7100 -m off -u 04',
'#/usr/local/lab-scripts/snmp_pdu_control --hostname pdu04 --command reboot --port 23 --delay 50',
'#/usr/local/lab-scripts/usb_hub_control -p 7100 -m sync -u 04'] %}
{% set pre_power_command = '/usr/local/lab-scripts/usb_hub_control -p 7100 -m sync -u 04' %}
{% set pre_os_command = '/usr/local/lab-scripts/usb_hub_control -p 7100 -m off -u 04' %}
{% set device_ip = "10.7.0.75" %}

* https://validation.linaro.org/scheduler/device/dragonboard-410c-02/devicedict
Adding a 96boards: LS Pinout

- 96Boards defines a standard Low-Speed expansion connector
- Common to all existing 96Boards specifications: Consumer, IoT, Enterprise, TV
- Allow to ‘debug’ (via UART) and control board state (via RST, PWR, and DCIN pins)
- All we need for test automation!
Adding a **96boards**: UART Mezzanine

- Is widely used for accessing 96Boards debug UART
- Integrates a FTDI (FTX) USB to Serial converter
- Exposed as a ttyUSB/COM port when connected to a Host
- A less known FTDI capability is **CBUS Bit Bang mode**
- Bit Bang mode allows to control CBUS pins as GPIOs over USB
Adding a 96 boards: FTDI CBUS Bit Bang Mode Support

- Only supported with user-space driver (via libfttdi and libusb)
- Does not coexist with standard Linux FTDI USB driver
- For automation we need to control board state and parse UART output at the same time (boot messages, u-boot control...)
- Solution: Implement FTDI CBUS GPIO driver in Linux kernel
- Upstream ongoing...
  - https://patchwork.kernel.org/patch/10555663/
Reminder: Adding a Board: Device dictionary

Example: Dragonboard-410c from validation.linaro.org*:

```python
{% extends 'dragonboard-410c.jinja2' %}
{% set connection_command = 'telnet localhost 7005' %}
{% set adb_serial_number = '1f58033c' %}
{% set fastboot_serial_number = '1f58033c' %}
{% set pre_power_command = '/usr/local/lab-scripts/usb_hub_control -p 7100 -m sync -u 04' %}
{% set pre_os_command = '/usr/local/lab-scripts/usb_hub_control -p 7100 -m off -u 04' %}
{% set hard_reset_command = ['/usr/local/lab-scripts/usb_hub_control -p 7100 -m off -u 04',
'    /usr/local/lab-scripts/snmp_pdu_control --hostname pdu04 --command reboot --port 23 --delay 50',
    '/usr/local/lab-scripts/usb_hub_control -p 7100 -m sync -u 04'] %}
{% set power_on_command = '/usr/local/lab-scripts/snmp_pdu_control --hostname pdu04 --command on --port 23' %}
{% set power_off_command = ['/usr/local/lab-scripts/snmp_pdu_control --hostname pdu04 --command off --port 23',
    '/usr/local/lab-scripts/usb_hub_control -p 7100 -m off -u 04'] %}
{% set device_info = [{'board_id': '1f58033c'}] %}
{% set flash_cmds_order = ['update', 'ptable', 'partition', 'hyp', 'modem',
    'rpm', 'sbl1', 'sbl2', 'sec', 'tz', 'aboot',
    'boot', 'rootfs', 'vendor', 'system', 'cache',
    'userdata',] %}
{% set device_ip = "10.7.0.75" %}
```

* [https://validation.linaro.org/scheduler/device/dragonboard-410c-02/devicedict](https://validation.linaro.org/scheduler/device/dragonboard-410c-02/devicedict)
Adding a 96boards: Device dictionary

The dragonboard-410c device dictionary is be simplified to:

```jinja2
{% extends 'dragonboard-410c.jinja2' %}
{% set hard_reset_command = '/dragonboard-ftdi.sh DA1WPWC7 reset-fastboot' %}
{% set pre_os_command = '/dragonboard-ftdi.sh DA1WPWC7 reset' %}
{% set power_off_command = '/dragonboard-ftdi.sh DA1WPWC7 off' %}
{% set power_on_command = '/dragonboard-ftdi.sh DA1WPWC7 on' %}
{% set connection_command = '/dragonboard-ftdi.sh DA1WPWC7 connect' %}
{% set fastboot_serial_number = '430d3c68' %}
{% set device_info = [{'board_id': '430d3c68'}] %}
{% set flash_cmds_order = ['update','ptable','partition','hyp','modem', 'rpm', 'sbl1','sbl2','sec','tz','aboot','boot','rootfs','vendor','system', 'cache','userdata'] %}
```

Note: dragonboard-ftdi.sh is a wrapper script which locates the USB FTDI device by its serial number and performs the appropriate action (gpio toggling, microcom exec ...)

Adding a **96** boards: Result

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Worker Host</th>
<th>Device Type</th>
<th>state</th>
<th>Health</th>
<th>Submissions restricted to</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>db410c-uboot_0</td>
<td>worker0</td>
<td>db410c-uboot</td>
<td>idle</td>
<td>Good</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dragonboard-410c_0</td>
<td>worker0</td>
<td>dragonboard-410c</td>
<td>idle</td>
<td>Unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nrf52-nitrogen_0</td>
<td>worker0</td>
<td>nrf52-nitrogen</td>
<td>idle</td>
<td>Good</td>
<td></td>
<td></td>
</tr>
<tr>
<td>guru_0</td>
<td>worker0</td>
<td>guru</td>
<td>idle</td>
<td>Good</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Additional Infrastructure

Usually, LAVA needs to be completed with external software/hardware infrastructure like switches, access points, file servers...

In our case, this infrastructure can be part of the tiny lab instance, three additional containers are created:

- **File Server**: A simple FTP/HTTP server allowing to push/pull images
- **WiFi Manager**: A soft-ap service creating WiFi access point, using host ieee80211 interface (wlan).
- **BT Manager**: A simple bluetooth service running bluez and using host Bluetooth interface (hci)
Architecture Overview
Time for Testing

Basic Bluetooth Scan test on 96Boards nitrogen

```
job_name: nrf52-nitrogen-bluetooth
device_type: nrf52-nitrogen
priority: medium
visibility: public

timeouts:
  job:
    minutes: 6

actions:
  - deploy:
    to: tmpfs
    images: zephyr
    url: http://file-server:8800/nitrogen/bluetooth.scan.hex
  - boot:
    method: pyocd
  - test:
    monitors:
      - name: test-bt-scan
        start: BOOTING ZEPHYR
        end: PROJECT EXECUTION SUCCESSFUL
        pattern: '(?P<test_case_id>\*) (?P<measurement>\*) tcs = [0-9]* nsec'
        fixupdict:
          PASS: pass
          FAIL: fail
```

Valid definition.
Achievements

- Keep the Lab simple ✔
- Easily reproducible ✔
- Autonomous ✔
- Compact ✔
- Low cost ✔
- Fully running on ARM ✔

Source: https://github.com/loicpoulain/96Lava
Next Stages

Ideas:

- Additional 96Boards tests: GPIO, SPI, I2C, Camera...
- Add new boards: Hikey960, rock960...
- LKFT remote lab
- Add missing CI components (Jenkins)
#YVR18
YVR18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org
Contact: support@linaro.org