Bootloader Testing in LAVA
Rémi Duraffort, Loïc Poulain
Why Test Bootloaders?

● As any other software, subject to bugs or regressions.

● The bootloader is a critical component
  ○ Can prevent the main OS/function to start

● Bootloader security
  ○ Chain of trust element (secure boot)
  ○ Privileged hardware access

● Second/Third Stage Bootloaders are complex
  ○ Small OS (u-boot, LK)
  ○ Various protocols (fastboot, DFU...) and drivers (FS, SDIO, USB...)
  ○ Network stack (TCP/IP, TFTP)
  ○ New standardized interfaces (EBBR, UEFI...)

● Bootloaders are usually much less tested
  ○ Can include non-upstream code (e.g. vendor/oem customizations)
LAVA

- Linaro Automated Validation Architecture
- Test execution system for **testing software** on **real hardware**
  - Boot testing
  - System level testing
  - Power consumption testing
  - etc.
- Organized in a server-dispatcher model:
  - **lava-server** (master): schedules jobs, administers devices and stores results
  - **lava-dispatcher** (worker): processes the test **jobs** and deploys images on DUT
- Scalable, distributed
LAVA architecture

- Job Configuration
- dispatcher
- Power control
- Serial relay
- LAVA
- LAVA
- kernel
- dtb
- rootfs
- tftp&nfs server
LAVA architecture (full lab)

- Users
- LAVA
- server
- dispatcher 1
- dispatcher 2
- dispatcher N
- Power control
- Serial relay
- tftp&nfs server
LAVA Job

1. **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...

2. **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)

3. **Test**:
   - Execute test(s) in the device shell
   - Parse output of the tests (success, failure)
   - Collect results
Example: Standard RPI3 LAVA Job

**device_type**: rpi3-b-32

**job_name**: rpi3-b-32 standard health check

**timeouts**:
- **job**: minutes: 10

**priority**: medium

**visibility**: public

**actions**:

- **deploy**:
  - **timeout**:
    - minutes: 6
  - **to**: tftp
  - **kernel**:
    - **url**: http://.../r0-raspberrypi3.bin
    - **type**: uimage
  - **modules**:
    - **url**: http://../raspberrypi3.tgz
    - **compression**: gz
  - **nfsrootfs**:
    - **url**: http://.../rootfs.tar.xz
    - **compression**: xz
  - **dtb**:
    - **url**: http://../bcm2710-rpi-3-b.dtb
Example: Standard RPI3 LAVA Job

- **boot:**
  - timeout:
    - minutes: 4
  - method: u-boot
  - commands: nfs
  - auto_login:
    - login_prompt: 'login:'
    - username: root
  - prompts:
    - 'root@raspberrypi3:~#'

- **test:**
  - timeout:
    - minutes: 5
  - definitions:
    - repository: http://.../tests.git
      - from: git
        - path: smoke-tests-basic.yaml
        - name: smoke-tests
    - repository: http://.../qa.git
      - from: git
        - path: ltp.yaml
        - name: ltp
Usual Assumptions (OS testing)

- **Deploying Images**
  - Known filesystem (e.g. ext4, fat...) in which test scripts are injected
  - Known and widely adopted provisioning protocol (fastboot, tftp, nfs, etc)
  - Stable and unique DUT identifier (USB ID, mac address, etc)

- **Booting Device**
  - UART/Console availability
  - Available recovery method in case of non functional OS

- **Tests**
  - POSIX OS/shell for test execution

And for sure, a **working bootloader** is the root condition of all these assumptions.
Bootloader issues

- No POSIX shell/system
  - Can not run standard test scripts

- No filesystem
  - No easy way to inject tests (overlay) into tested system image

- Harder to deploy/recover
  - The bootloader is usually responsible for system update (e.g. fastboot, TFTP...)
  - Need a way to provision a new bootloader without relying on an operational one.
    - Recovery mode
    - Out-of-band provisioning

- Identifiers/Uniqueness
  - E.g. empty DFU serial number
Testing u-boot on Raspberry Pi

● Flashing bootloader
  ● Boot from SDCard
  ● Automatic SD flash
    ○ SDMux/SDWire

● Interactive test
  ● New in LAVA 2019.03
  ● Commands sent over UART
  ● Parse output (successes and failures)
  ● Example:
    ○ https://staging.validation.linaro.org/scheduler/job/248753/definition
Provisioning with SD MUX

- Don’t rely on a minimal functional system or recovery mode.
- Multiplexing the SD-Card to either DUT, Host or disconnect.
- Every USB-SD-Mux should be identified by a unique serial number at USB-level.
- Full control on memory layout, no bootloader partition/blob writing restriction.
Interactive Testing

interactive:
- name: memory-test
prompts: ["=> ", "/ # "]

script:
# set and test network interface
- command: dhcp
name: dhcp
  successes:
  - message: "client bound to address"
  failures:
  - message: "TIMEOUT"
    exception: InfrastructureError
    error: "dhcp failed"

[...]
- command: base
name: default-base-address-offset
  successes:
  - message: "Base Address: 0x"
[...]

- command: base B0000000
  name: set-address-offset-0xb0000000
  successes:
  - message: "Base Address: 0xb0000000"

# mw - memory write
# md - memory display
- command: mw 100000 aabbccdd 10
- command: md 100000 10
  name: mw-md-100000
  successes:
  - message: "aabbccdd"

# md - memory display
# cp - memory copy
- command: cp 100000 200000 10
- command: md 200000 10
  name: cp-md-200000
  successes:
  - message: "aabbccdd"
Demo
Provisioning without SD MUX

- Depends on the board features
- Remotely control boot media (relay, ethernet, ...)
- Two boot medias:
  - Select boot to NAND
  - Boot on NAND
    - Stable/recovery u-boot
    - Flash NOR (dfu, ...)
    - Reboot
  - Select boot to NOR
  - Boot on NOR
    - Test bootloader
    - Boot to Linux

See rzn1d-dfu.yaml
Thank you
Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contact@linaro.org