Automated Flashing and Testing for Continuous Integration

Igor Stoppa

Embedded Linux Conference
North America 2015

Automated Flasher Tester AFT

Tool for deploying and verifying a SW image on an appropriate target HW device.

- Easily integrate with existing setup.
- Minimize requirements of the setup (including cost).
- Easily scale both the number and the type of units.
- Minimal deployment & testing time.
- Keep close to real life conditions.
- Enable Developers to test before submitting, using identical means of verification.

Continuous Integration

Merge contributions <u>often</u>, even multiple times per day,
but <u>only</u> if they meet the following requirements:

1. Patches must apply cleanly.
2. Patched SW must build cleanly - at least not worse than pre-patching.
3. Generated SW image must deploy successfully to the Device Under Test (DUT).
4. Deployed SW image must boot.
5. Deployed image must pass a set of predefined test cases.

<u>Only what is tested can be expected to work.</u>

Why yet another tool?

- Several proprietary solutions available but:
  - focused only on one architecture
  - non open source - hinders sharing publicly testing methods / results

- Other public solutions, ex: LAVA (Linaro)
  - not alternative, rather complementary
  - LAVA provides lots of infrastructure (queuing, result visualization, etc.)
    AFT focuses on optimizing deployment and test execution.

- Let developers use locally exactly the same test configuration used by CI.
  - install and configure the tool painlessly

Woes of Continuous Integration

Speed: building and testing can take a long time, but it’s needed frequently

- parallelize sw build - throw in more servers
- parallelize testing - *but deployng the SW image can still take a long time!*

Multiple-targets: supporting multiple platform/arch is resources-intensive

- build infrastructure can adjust (run different cross compilers/qemu)
- each new type of HW needs ad-hoc work: expensive and slow.
- during the project the HW to support will likely change.

Optimizing Deployment & Testing can have high returns.

Key Features

Interaction with DUT
Standardize the interaction: do not use specific interfaces/APIs.

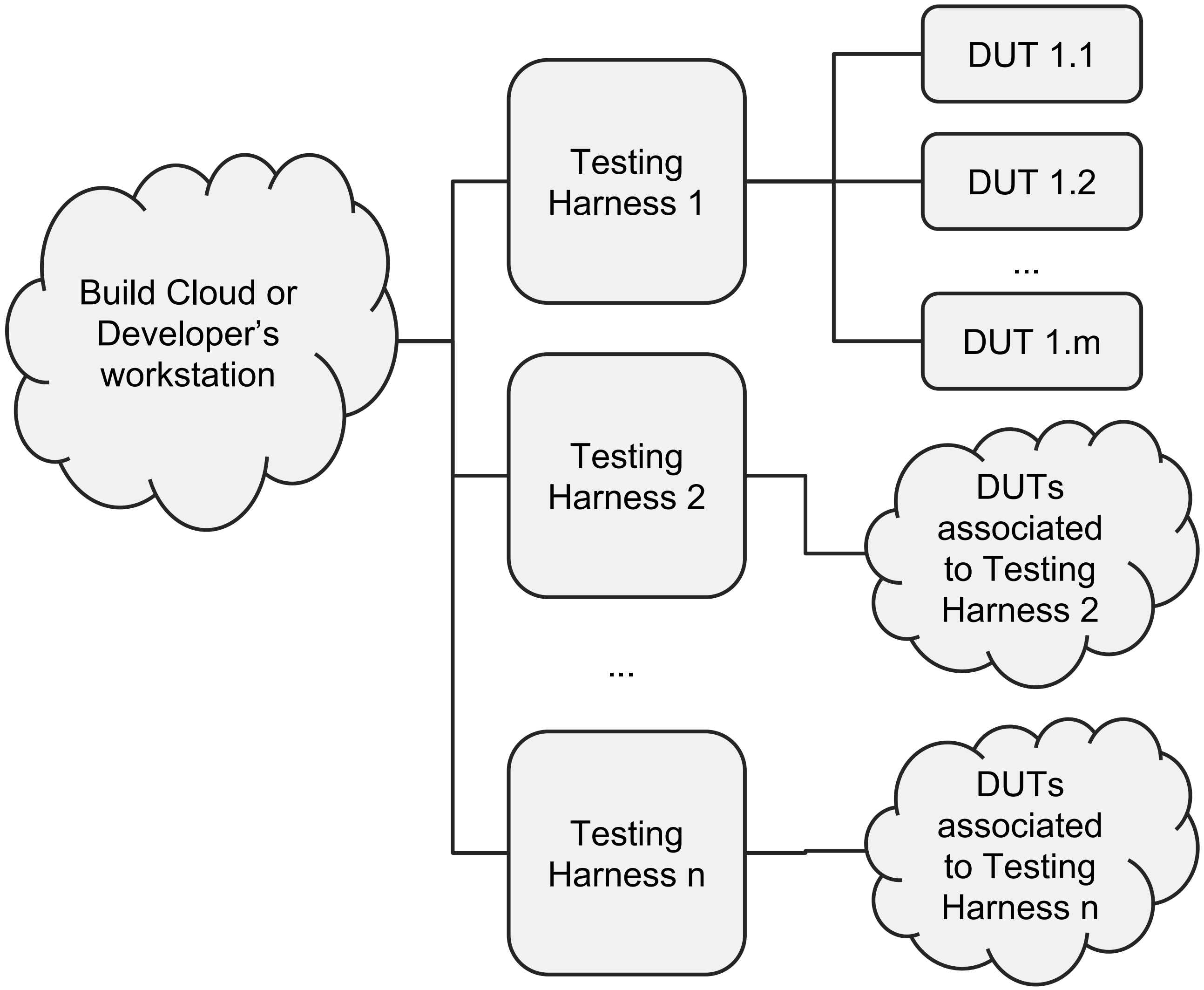
Rather, ***emulate the user***:

- plug/unplug power
- enter data from a (USB) keyboard
- issue commands to transfer data over a network connection.

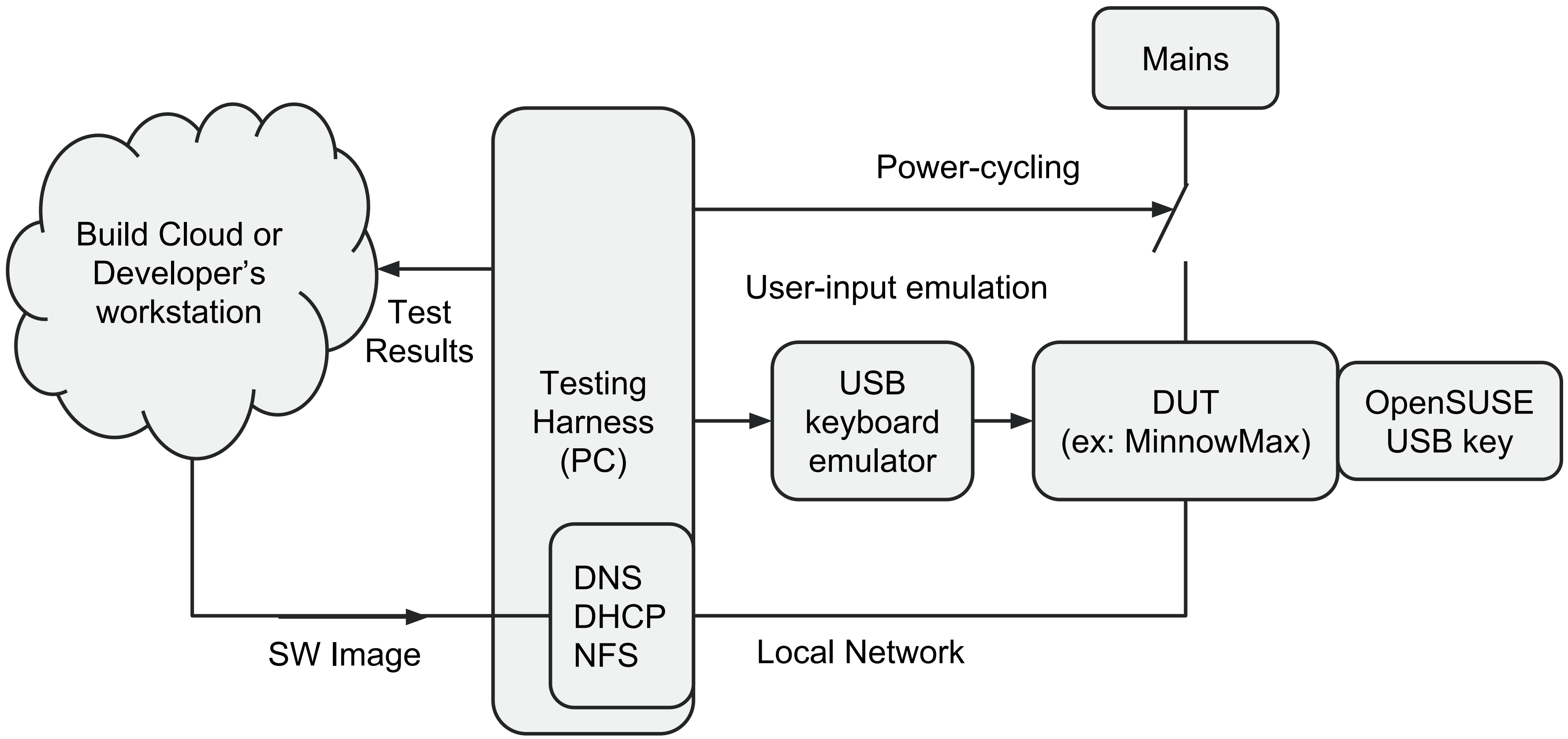
Deployment of SW image to test
Deploy the same image produced by the build system, directly to the DUT.

Allow passwordless login by injecting the ssh public key of the testing harness, for the root user.

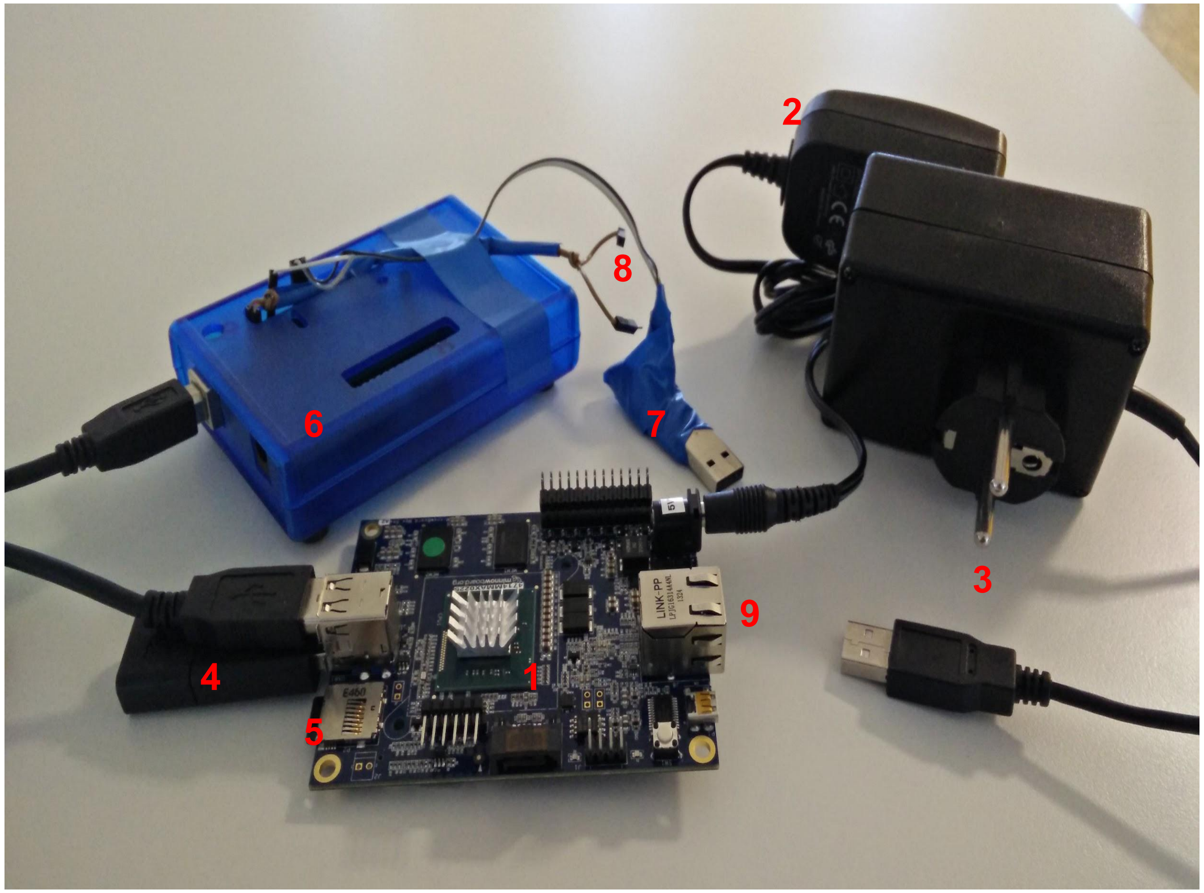
High level view


Build Cloud or Developer's workstation
Testing Harness 1
Testing Harness 2
...
Testing Harness n
DUT 1.1
DUT 1.2
...
DUT 1.m
DUTs associated to Testing Harness 2
DUTs associated to Testing Harness n

Simplified Model

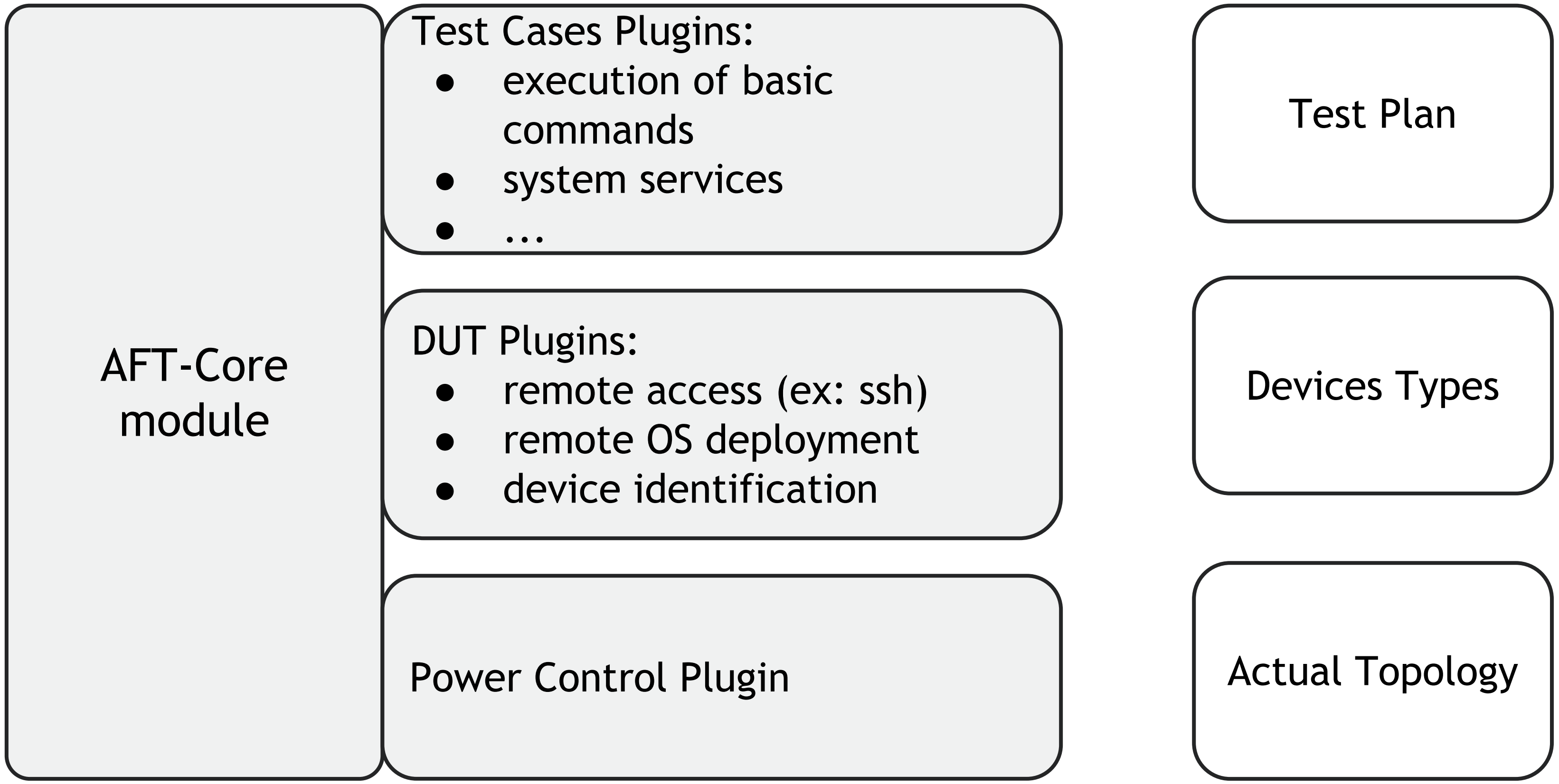


HW configuration



1. MinnowBoard Max
2. Minnow Power Supply
3. USB-controlled power cutter.
4. OpenSUSE Thumb drive
5. SD Card (target media)
6. Arduino UNO R3
7. USB to Serial port Control interface for UNO
8. Programming toggle for UNO USB port.
9. Ethernet port

SW Stack & Data



Main Steps
(ex: MinnowBoard Max)

1. Verify compatibility: is the image supported and is there a suitable device?
2. Allocate device compatible with the SW image.
3. Power-cycle and boot the DUT into service mode from the OpenSUSE USB key.
4. Deploy the test SW image to the target storage - it is taken from an nfs share.
5. Install the public ssh key used by the Testing Harness for the DUT root user.
6. Power-cycle and boot the DUT into testing mode, from the image just deployed.
7. Check for availability and run the selected test plan.
8. Report back the test results, in xUnit format.

But it was not always easy ...

System Partitioning: many SW components are specific to the HW they drive (ex: power switch) and the interface used must support various models from different vendors.

BIOS configuration: some (most?) BIOSes rearrange the sequence of the boot devices, others can completely lose their settings when the DUT is power-cycled - the only foolproof solution is to reconfigure the BIOS at each iteration.

More BIOS blues: at some point one device changed autonomously MAC address - reason still unknown.

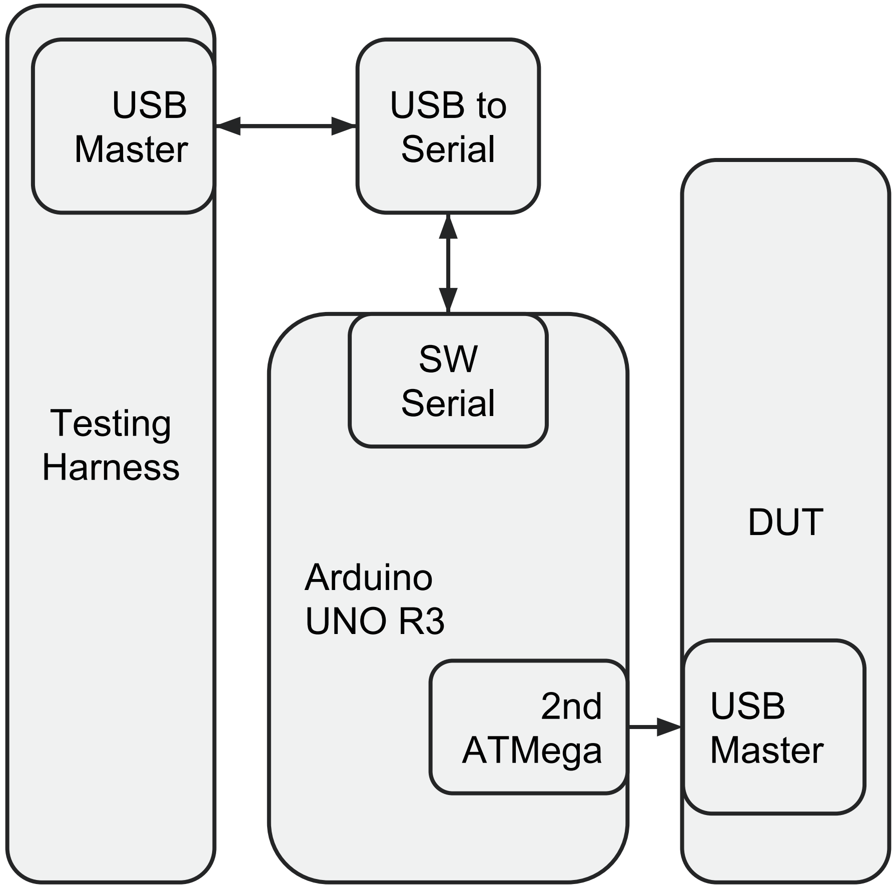
System Architecture: initial idea - Testing Harness as VM - qemu cannot reliably pass to the VM 2+ USB devices with same VendorID:DeviceID - Doh.

Network Interfaces: OpenSUSE network manager (wicked) cannot bring up interfaces with static IP and no carrier.

USB thumb drives: some brands dropped like flies, after undergoing only a few power-cycles.

USB Keyboard Emulation

- Based on Arduino UNO R3, the only one currently capable of interfacing with BIOSes
- Uses LUFA FW to emulate the USB protocol.
- Uses only libraries with GPLv2 compatible license.
- Messaging protocol to detect data losses
- It costs ~ $7, while the cheapest commercial alternative is above $100
- Supports:
  - record/playback of keys
  - LibreOffice Calc sheet
  - sequence generated on-the-fly



Does it meet the requirements?

All the requirements were met:

- **The only requirement toward the target OS is to provide means for login access.**
- **Adding support for new OS (Yocto) and new HW (NUC, MinnowBoardMax) took less than a week.**
- **Depending on the performance of the DUT, a deployment & basic testing session can last between 3 and 10 minutes.**
- **The BOM per-device is fairly frugal and relies solely on inexpensive off-the shelf CE devices, easily obtainable in large amounts, worldwide.**
- **Everything, from the SW to the HW setup, is public and can be reproduced anywhere.**

Ideas for future development

- Support more architectures/boards.

  Ex: Edison, Quark, BeagleBone Black, WandBoard Quad, ODroid.
- Support more flavors of power cutter.

  Ex: ethernet controlled.
- Replace USB-Serial interface with ethernet.
- Use Edison as testing harness, rather than a PC.
- Support test cases run with fMBT (https://01.org/fmbt)

References

Automated Flasher Tester:

https://github.com/igor-stoppa - all the aft-* projects

USB Keyboard Emulator (Peripheral EMulator):

https://github.com/igor-stoppa/pem

Q & A