Open CI for Trusted Firmware

Good practices and design considerations

System overview

Trusted firmware build configurations

Default

- Default is the production trusted firmware build, intended to be integrated with external applications.
- It does not run any tests.

Core Test

- Core tests are designed to verify the integrity of the trusted firmware core module.
- Core tests must be carefully ported to each new platform.

Regression

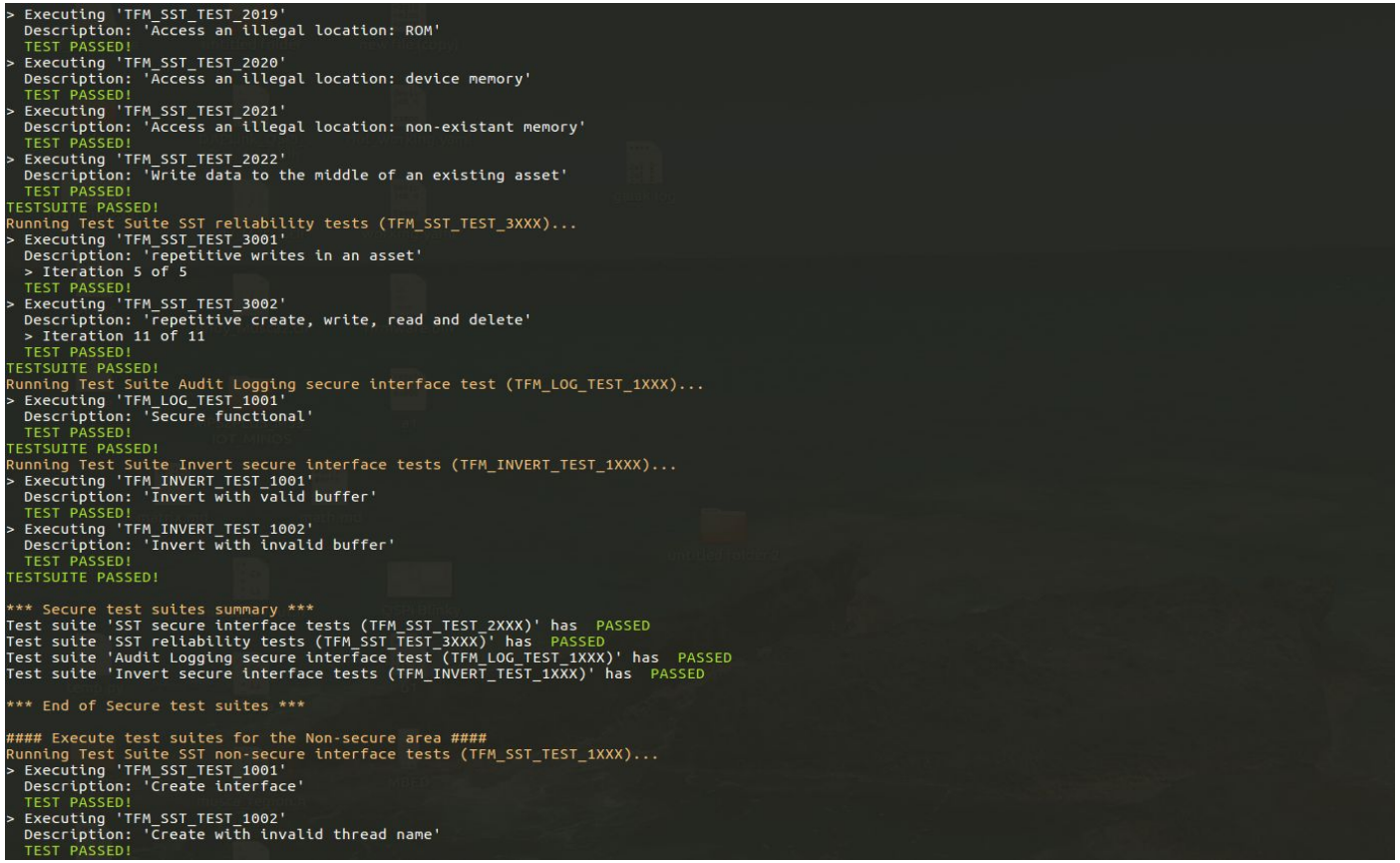
- Regression tests will perform a series of secure calls to verify each of the supported trusted firmware services.
- They are platform agnostic, with few exceptions on hardware feature related services.

Trusted Firmware tests

Regression Tests example

During regression testing, the suite verifies the design on both secure and non-secure sides.

- TFM Tests only provide PASSED/FAILED status.
- ANSI colors are used to produce a human readable output.
- A summary is appended on the end of each suite



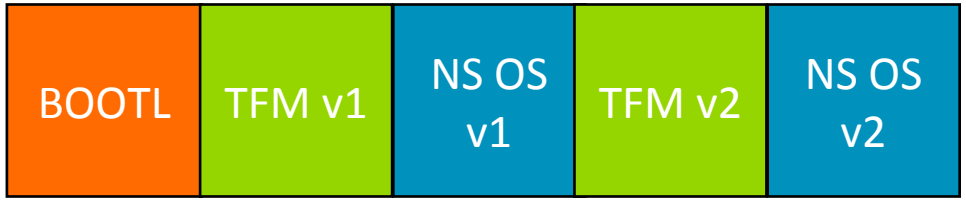
arm

Trusted firmware partition variants

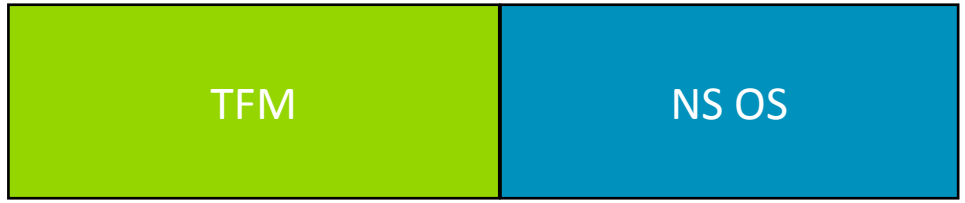
Planning for bootloader and OTA upgrades

- By default TF-M creates a five-part partition configuration. A secure and non-secure image are merged together and signed.
- Two slots are provisioned to enable firmware upgrades.
- MCUBoot is used as a second stage bootloader to verify the images, and chain load to normal operation, or perform upgrade.
- It is possible to strip that functionality or create user defined partition tables.

MCUBoot with TFM and Non-Secure OS



TFM with Non-Secure OS



arm

Controlling the hardware

It is imperative that the testing software can control and validate the state of hardware.

- The software needs to be able to power-cycle, reset, upload firmware and communicate with the board during the test.
- Software needs to be able to control the flow of hardware boot sequence and interrupt it if required.
- Need for specialized equipment requirements such as JTAG debuggers shall be kept to a minimum. Open-source alternatives preferred (i.e. mbed DAPLink)
- The hardware should be able to provide state information to the software.
- The design should account for hardware specific variants and enable easily extensible functionality.

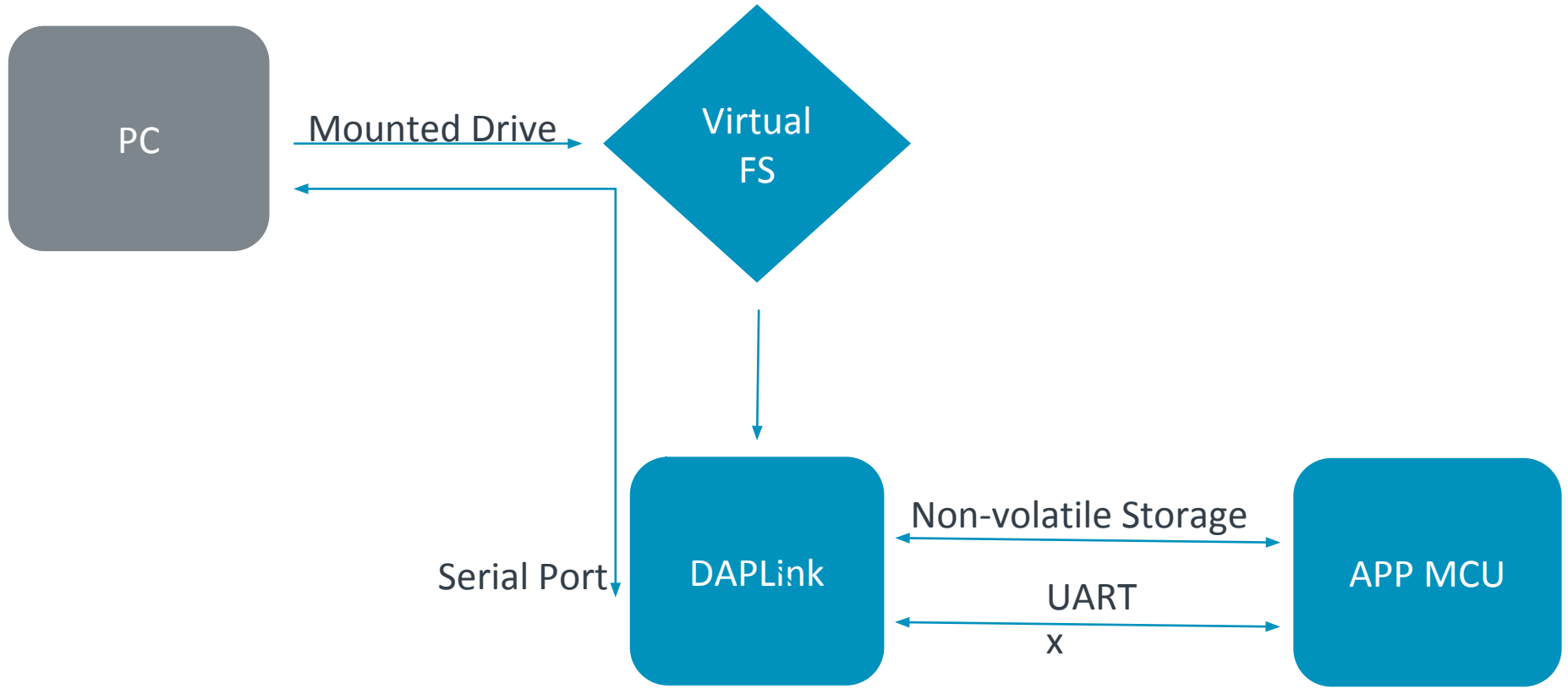
Arm Mbed DAPLink

DAPLink is an interface firmware that enables firmware flashing and debugging of Arm Cortex CPUs. In an Arm IoT boards it runs on a secondary controller that is attached to the application MCU.

- DAPLink provides a virtual filesystem enabling flashing by drag & drop of the firmware.
- It creates a virtual serial port, attached to one of the UART provided by the application MCU.
- It enables rudimentary control of the power state of the application MCU, through serial and virtual files.
- Going forward it will need to have its functionality extended, adding more CI friendly instructions.

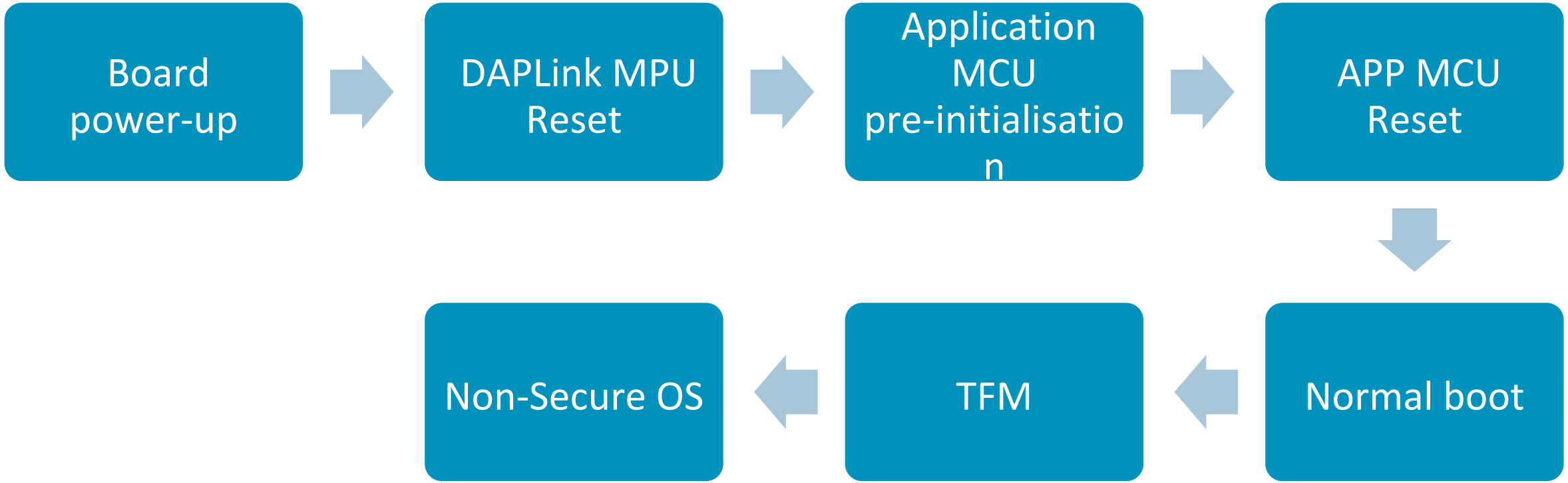
Arm Mbed DAPLink

Functional overview



Hardware Boot Sequence

ARM IoT enabled board



arm

Linaro Automated Validation Architecture (LAVA)

LAVA is a fully customisable solution for deploying software onto physical and virtual hardware and running tests. Its core modules are the lava-server and the lava-dispatcher. The dispatcher is responsible for wrapping around the physical hardware. For DAPLink enabled Arm Cortex-M devices, the lava-dispatcher can be used as is.

- Uploading the firmware is a simple copy operation.
- Lava will attach to the virtual serial and parse all the produced output.
- Lava uses a monitor window functionality, triggered by start and stop text strings, and applies regex matching between the triggered area, to verify PASSED|FAILED state of a test.

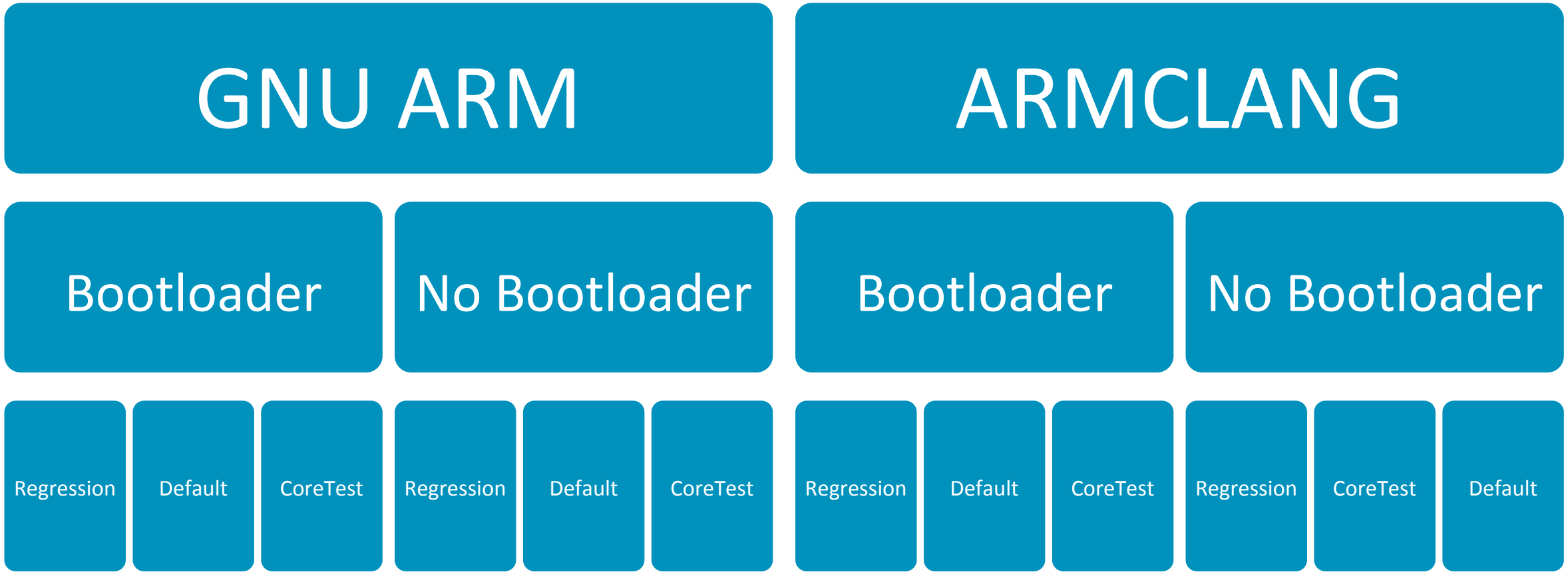
Open CI design challenges

We aim to develop an open robust system with versatility, scalability, extensibility. Several challenges arise, including

- Merging multiple binaries and verifying their integrity.
- Testing on hardware adds a new point of failure (flashing controller).
- Managing the different build configurations.
- Time required to check each patch.
- Create a simple and easy to use suite for the user.
- Provide standardized hooks for existing open CI solutions.

Build configurations

12 configuration combinations per platform



arm

Platform specific constrains

Different platforms variants can offer, or limit provided services.

- Specific tests may fail.
- Test execution time is platform dependent. Timeouts need to be adjusted.
- The flow of flashing and running the tests can differ among boards.
- The code size may vary among compilers, and exceed allocated code space for a specific platform.
- Some platforms may use discrete UART interfaces, while others multiplex them in a single port.

Time

Everything should be tested, but is there enough time to do so? A typical Regression/Debug test on a board running from QSPI with execute in place can take over 10 minutes. Assuming 24 combinations, full testing would require more than 4 hours.

- Release builds should be prioritised.
- Which platforms does the user need to test locally before upstreaming?
- Running tests in parallel increases the hardware requirements of the CI system, and adds synchronization issues.
- How much testing is considered adequate?

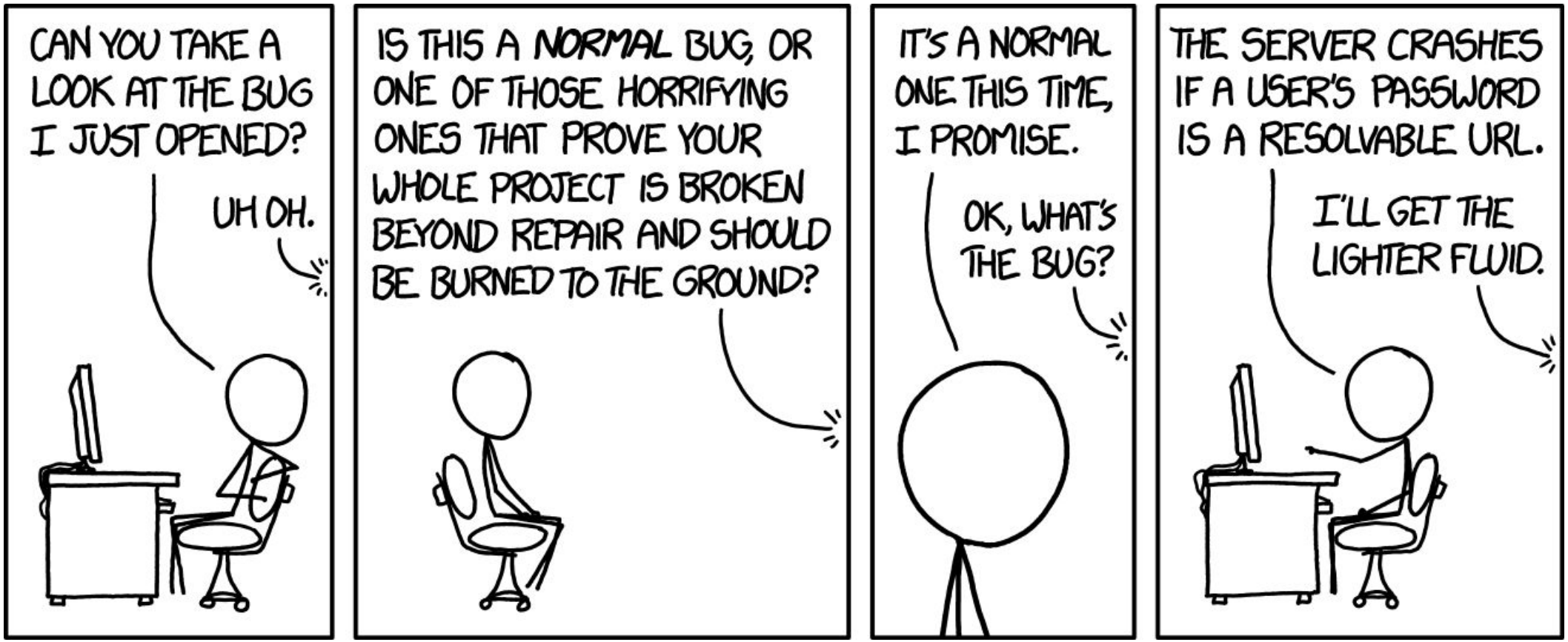
Synchronicity

One of the biggest challenges of live hardware testing is verifying that code running on the board is the one that is meant to be tested.

- When a board automatically powers up during the deploy (programming) stage, it may produce some text from the previous application already on the device.
- This text can corrupt the serial buffers, and cause the monitor state to trigger in the wrong state, or pass a test based on the previous build.
- If a board configuration is faster during a reboot cycle, it may miss a portion of test text that is required for a test to pass.
- DAPLink could be adjusted to either power down the application MCU during flashing, or automatically disable the Serial output at this state.
- Ideally the test binaries should block and wait for the testing software to send a launch trigger.

Open Continuous Integration System

Hunting for bugs



New Bug, from Randall Munroe, xkcd.com

arm

Open CI reference design.

A flexible high-level scripted language should be used to provide a test framework, combined with a virtual environment for automated testing on user side.

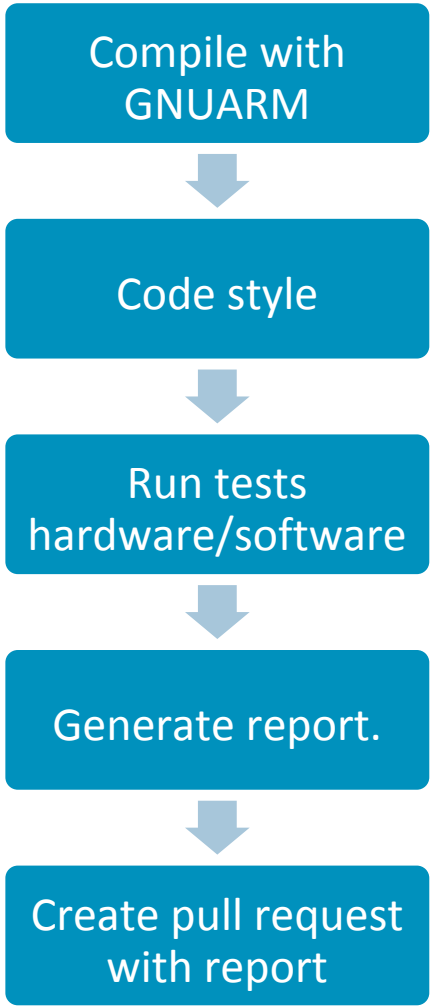
- Python3+ is mature, provides **virtualenv**, and is widely used.
- The process should be structured in discrete steps which will perform a single task:
  - Build configuration combinations and create an artifact list.
  - Run the binaries and store the output to a log file.
  - Parse the log using regex and determine failure/success.
- Hardware and Simulation control should be wrapped around and abstracted.
- The framework should provide a standardized modular support for third party CI solutions (Travis/ Jenkins/ Lava).
- The test framework, should validate the logic as well as capture and record code size or performance affecting changes.
- A report should be produced in an open and standard documentation format.

A Hybrid Development flow

Local testing first

Users should validate that the patch does not break the platform they are targeting before opening a pull request.

- Testing can happen in simulation or hardware.
- It is required to only test GNUARM compiler.
- Code style is tested using regex linting (cppcheck/checkpatch).
- A report should be generated, and attached with the pull request.
- If more than a single combination of compiler/platform is tested it should be included in the report.



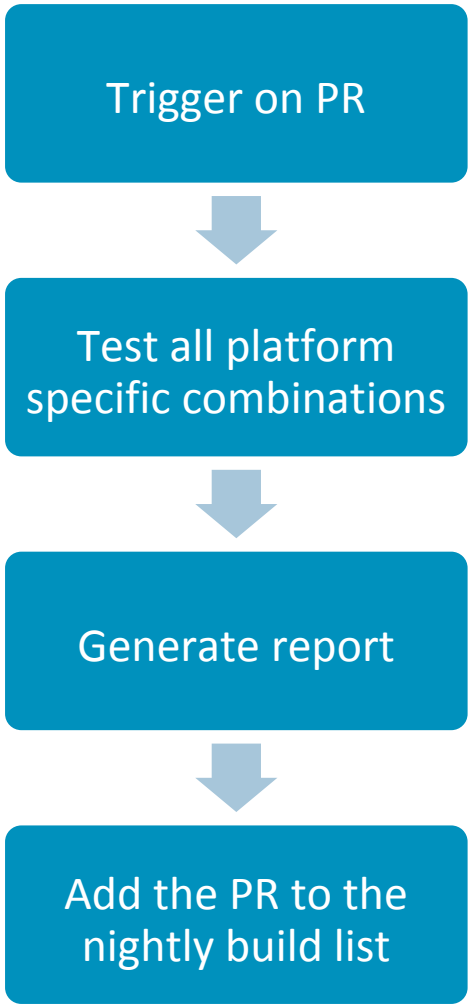
arm

A Hybrid Development flow

Tested on the backend(Travis/Jenkins)

After a pull request has been triggered, the system should attempt to build/test multiple configurations.

- Per patch testing should pass for every platform related configuration. Success will give a provisional +1 score.
- A nightly build will test every patch submitted on the day and will give a final +2 to the score.
- Manually triggered extensive testing for priority patches, should be possible.
- When the remote system is overloaded, it may deprioritise tests that have been performed on the user side.



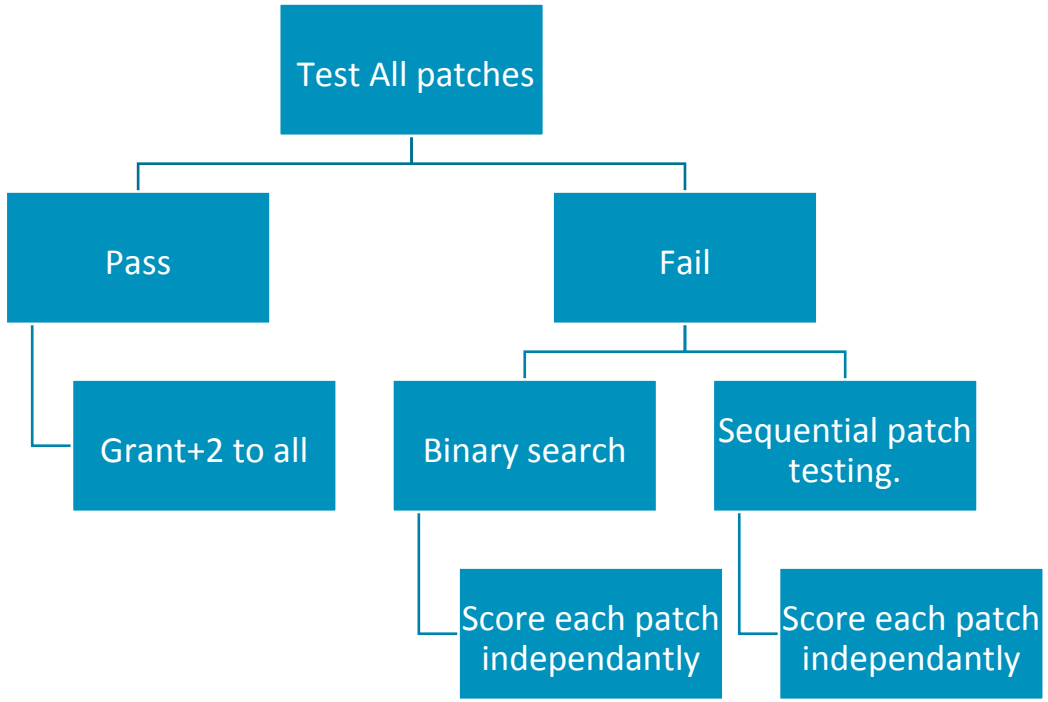
arm

A Hybrid Development flow

Nightly Testing

Out of working hours the system should check out daily changes and test them as a set if possible. Success would be passing all tests.

- If the number of patches allows it, testing should be done in a per patch basis.
- A nightly build will test every patch submitted on the day and will give a final +2 to CI score.
- When the combined patch fails, depending on the load the system should perform per patch testing, either sequentially or following binary search pattern.
- When the remote system is overloaded, it may deprioritise tests that have been performed on the user side.



arm

Testing the binaries

On hardware and emulation

Hardware

On hardware

- The board is assumed to be running a DAPLink firmware, or a functionality mimicking variant.
- Flashing the firmware using simple copy, and reset over serial or command file should be supported.
- Serial output will be stored into a file and regex parsed to determine the state of the tests.

Simulation

Qemu 3.0 can be used to verify the current iteration of trusted firmware for AN521 using platform AN505.

- Platform support will be extended.
- Log file should be stored and parsed.
- Synchronicity is not a problem with emulated testing.
- Recommended for functional testing, but may miss timing related faults.

Questions?

arm

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद