Zephyr® Project
Developer Summit 2022

Twister

A Test Utility for Contributors

Aastha Grover, Intel Corporation

Summary

- Twister & Its Utility
  - How to Run Twister
  - Adding Tests to Zephyr & YAML Structure
- Features
  - Harness, Fixtures, Integration Mode & Quarantine
- Running Twister on Hardware
  - HW testing in Zephyr
  - Running Tests on Multiple HW Devices
  - Running Twister on Platforms out of Zephyr tree
- Other Twister Options
  - Twister Reporting & Performance Tuning
  - Differences between Samples & Tests
  - Testing Twister Itself

Aastha Grover

aasthagr

Intel Contributor to Zephyr Project.

What is Twister and its Utility?

- Test runner tool (written in Python)
  - Scans the Zephyr OS codebase for samples and tests and attempts to execute them.
- Helps Zephyr contributors focus on developing features while keeping Zephyr codebase coherent.
- By executing the testcases on various Platforms & architectures
  - Twister helps keep complete code tree buildable & avoid any regressions.

What is Twister and its Utility? (Cont.…)

- By default, twister builds & runs each test case on boards marked default in the board definition file.
- Default options builds most tests on defined set of boards and runs in an emulated environment. (if available for the architecture or configuration being tested).

How to Run Twister.

- Basic twister run. Will execute all testcases on all default architectures & platforms.

  Note: Full twister runs primarily done in CI.

  ```
  $scripts/twister
  ```

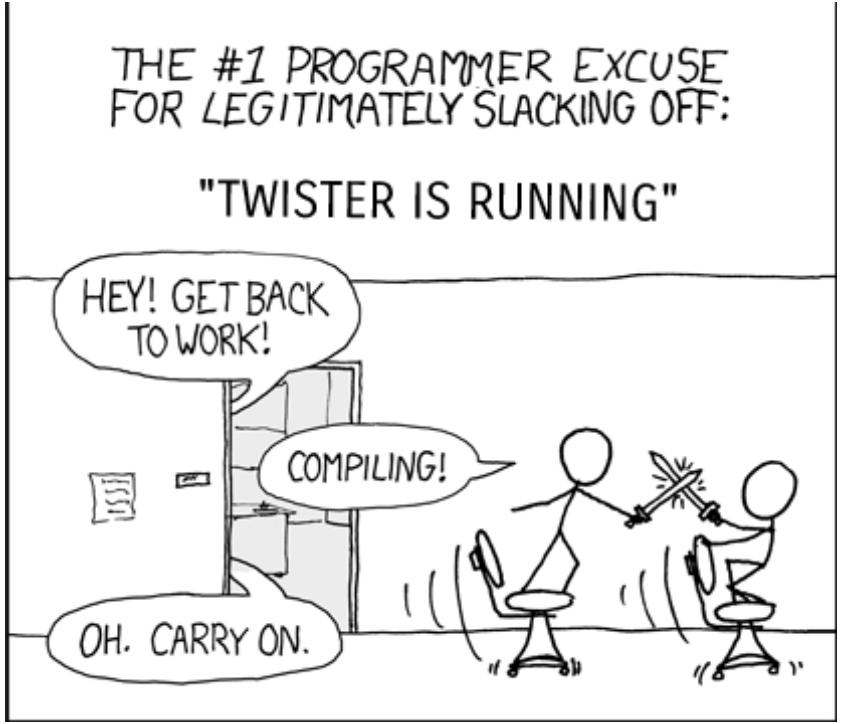
- Using twister with options leverages different capabilities of twister

  ```
  $scripts/twister –T tests/kernel -p qemu_x86
  ```

  where –T: Directory path & -p: Name of the board.

Problem Statement

- Twister helps you run 1000's of testcases and samples with ease, so we can be sure we got proper coverage and to avoid any regressions.
- Limited amount of time. Streamlining & standardizing the testing process is a priority.



Adding Tests/Samples to Zephyr

- Must have the following basic structure at minimum.

- Zephyr/tests
  - Testcase.yaml/sample.yaml - Test cases are detected by its presence
  - CMakeLists.txt - Build script to add source code to library targets.
  - prj.conf - Config file for testcase/application.
  - src/*.c - Source code for application or testcase.

Testcase YAML Structure

```
common:
  depends_on: sdhc
  tags: drivers sdhc
tests:
  subsys.sd.sdmmc:
    harness: ztest
    harness_config:
      fixture: fixture_sdhc
    filter: CONFIG_SD_STACK
    tags: sdhc
    min_ram: 32
    integration_platforms:
      - mimxrt1064_evk
```

- Testcase identifier
  - Starts with section followed by subsection separated by dot.
  - Example: subsys.sd.sdmmc.
- Tags : Functional domains. Command line invocations of this script filters set of tests to run based on tag
  - Useful feature in CI & for contributors.

Features

Harness Configuration

- If testcase defines harness , harness string needed to run tests successfully.
- As simple as a loopback wiring or complete hardware test setup for sensor and IO testing.
- Pertains to external dependency domains but can be anything such as console, sensor, net, keyboard, Bluetooth or pytest.

```
sample:
  description: Hello World sample, the simplest Zephyr
    application
  name: hello world
common:
    tags: introduction
    integration_platforms:
      - native_posix
    harness: console
    harness_config:
      type: one_line
      regex:
        - "Hello World! (.*)"
tests:
  sample.basic.helloworld:
    tags: introduction      Anas Nashif, 4 years ago •
```

Fixtures

- Fixtures specify a test case dependency on external device(e.g., sensor). An additional setup or special wiring specific to the test.
- Only one fixture can be defined per testcase.
- All tests which need extra fixtures configuration skipped in default
- Twister --fixture option, can be used if board has environment set up.

```
common:
  tags: drivers gpio
  depends_on: gpio
  filter: dt_compat_enabled("test-regulator-fixed")
  harness: ztest      Peter A. Bigot, 2 years ago •
  harness_config:
    fixture: regulator_loopback
  integration_platforms:
    - nrf52840dk_nrf52840
```

Integration Mode

- Narrows down scope of builds & tests.
- Defined under the integration keyword in testcase definition file. (testcase.yaml and sample.yaml).
- Mode activated using --integration option with twister.
- Used in CI to give developers fast feedbacks on changes introduced in Pull Request.

Quarantine

```
- scenarios:
    - sample.basic.helloworld
  platforms:
    - all
  comment:  "Link to the issue: https://github.com/zephyrproject-rtos/zephyr/pull/33287"


- scenarios:
    - kernel.common
    - kernel.common.misra
    - kernel.common.nano64
  platforms:
    - qemu_cortex_m3
    - native_posix
```

Failure of one test affects the execution of other tests.

- Useful when running larger test suits.

Use YAML files to define list of tests under quarantine.

- Sequence of dictionaries having combinations of scenarios & platforms.
- Optional: Comment can be used.
- --quarantine-list <PATH_TO_QUARANTINE_YAML>

Quarantine list of tests can be verified

- Adding --quarantine-verify to twister call.

Tests skipped & marked accordingly in output reports.

HW testing in Zephyr

To enable HW testing use --device-testing option with twister.

- *Single device Testing* can be done using --device-testing, --device-serial or --device-serial-pty options with Twister.
  - Example: twister --device-testing --device-serial /dev/ttyACM0 \ --device-serial-baud 9600 -p frdm_k64f -T tests/kernel
- Multiple device Testing is done using hardware map for parallel execution:
  - --generate-hardware-map: Generates the hardware map.

    Example: twister —generate-hardware-map map_name.yaml
  - --hardware-map: Loads hardware map from an existing file.

    Example: --hardware-map map_name.yaml
  - Hardware map is also required if one wants to use platforms with fixtures.

Running Tests on Multiple HW Devices

1. Creating the Hardware Map

- With all connected devices and their details such as serial device, baud and their IDs if available.
- Options marked as ‘unknown need to be replaced with values for connected hardware. Example: platform names and the runners.

```
twister –-generate-hardware-map  map_name.yaml
```

```
- connected: true
  id: OSHW000032254e4500128002ab98002784d1000097969900
  platform: unknown
  product: DAPLink CMSIS-DAP
  runner: pyocd
  serial: /dev/cu.usbmodem146114202
- connected: true
  id: 000683759358
  platform: unknown
  product: J-Link
  runner: unknown
  serial: /dev/cu.usbmodem0006837593581
```

```
- connected: true
  id: OSHW000032254e4500128002ab98002784d1000097969900
  platform: reel_board
  product: DAPLink CMSIS-DAP
  runner: pyocd
  serial: /dev/cu.usbmodem146114202
  baud: 9600
- connected: true
  id: 000683759358
  platform: nrf52840dk_nrf52840
  product: J-Link
  runner: nrfjprog
  serial: /dev/cu.usbmodem0006837593581
  baud: 9600
```

Running Tests on Multiple HW Devices (Contd..)

- If the map file already exists new entries are added, existing entries will be updated.

2. With the hardware map ready, you can run any tests by pointing to map file:

```
./scripts/twister --device-testing --hardware-map map.yml -T samples/hello_world/
```

Running Twister on Platforms out of Zephyr tree

- Zephyr supports testing using twister on platforms not defined in Zephyr tree.

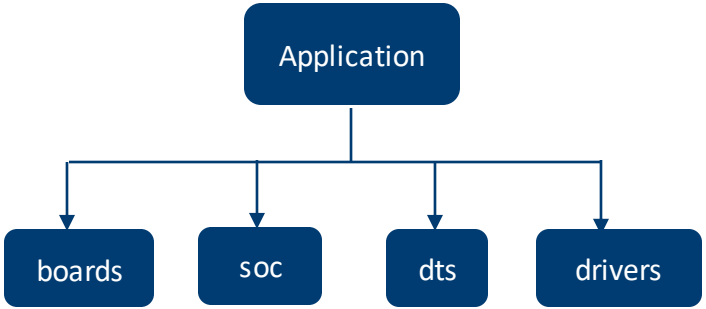
twister –p platform_name --device-testing

--device-serial serial -T testcases

-x=BOARD_ROOT= APPLICATION_PROJECT_DIR

-x=SOC_ROOT= APPLICATION_PROJECT_DIR

-x=DTS_ROOT= APPLICATION_PROJECT_DIR



Twister Reporting

- Twister reports generated by default in twister-out directory. Example:

```
scripts/twister -T samples/hello_world/
```

```
[aasthagr@aasthagr-mobl1 zephyr]$ ls twister-out
qemu_x86  qemu_x86_64  testplan.json  twister.json  twister.log  twister_report.xml  twister_suite_report.xml  twister.xml
```

- Reports generated in .json and .xml formats. Testplan.json is filtered list of testcases generated for every twister run .
- Platform specific directories contain each testcase result in detail.
- Example: twister-out/qemu_x86/samples/hello_world/sample.basic.helloworld/ directory contains :
  - build.log: all output messages of the build process
  - handler.log: output results of testcase.

Performance Tuning

- Parallelize jobs to reduce the twister execution time. Use "-j”/"--jobs” option
  - to specify the number of jobs for building.
- Running only failed testcases instead of whole testsuite.”
  - Using “-f”, “--only-failed” option only runs those tests that failed the previous twister run invocation.
- Re-use the outdir before building using “-n”/“--no-clean” option.
  - Builds will be incremental.

Difference between Samples & Tests

- Twister can run both Samples & Testcases. Samples & Testcases are not same.
- Sample: Working Application in Zephyr, briefly demonstrates functionality of domain/subsystem/feature.
- Testcases: Complete set of test suite, purpose of which is to flag twister run if anything breaks in Zephyr.
- Tests are implemented with a framework to report results of the testcases which were executed. Samples just run the output through pattern matching layer to decide if test succeeded or not.

Testing Twister Tool

- Twister contains various features and options scripted in Python. The runner tool needs to be tested - to flag unexpected behavior in twister.
- The testcases can be found in zephyr codebase under $ZEPHYR_BASE/scripts/tests/twister directory

  Data files: $ZEPHYR_BASE/scripts/test_data directory.
- Testcases are for TestInstance & Testplan class alongwith Twister script.

  Executing testsuite
  - From the root directory using `pytest $ZEPHYR_BASE/scripts/tests/twister`.
- Twister testsuite runs as part of CI when changes related to twister are introduced in PR.

Resources

- Zephyr Documentation
- Source Code
- Mailing Lists:
  - User List: users@lists.zephyrproject.org
  - Developer List: devel@lists.zephyrproject.org
- Discord
  - #testing
  - #ci
- Contributing to Zephyr: See the Contribution Guide

Questions?
Comments?

Aastha Grover
aasthagr

Thank You!!