Improve embedded software unit testing efficiency

Version 1.0

Non-Confidential

Issue 02

102084_0100_02_en

Improve embedded software unit testing efficiency

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	20 April 2020	Non-Confidential	First release

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

In this guide, you will learn how to increase your unit testing throughput, by running more tests in less time. This efficiency improvement comes from using virtual platforms, instead of physical hardware, as a development platform. This guide will be useful for anyone developing or running unit tests for embedded software.

Accurate and efficient tests are essential when developing software for embedded systems, particularly for continuous integration and functional safety flows. Quick pass or fail results from a comprehensive regression test suite enable teams to develop quality software without wasting engineering resources waiting for tests to finish running. Unit tests are a key component of regression test suites. This is because unit tests ensure that the small code snippets, which are the building blocks of the product, are working as expected.

Streamlining unit testing is important. This is because unit testing can become a speed bottleneck with hundreds or even thousands of individual tests running to verify that each code snippet is working as expected. If running a regression test suite is too time-consuming:

- Engineering resources are unused while they wait for tests to finish
- Code quality decreases because code is tested less frequently
- Time-to-market increases

This guide describes a testing methodology that uses virtual platforms. Virtual platforms allow you to run unit tests faster and more efficiently than traditional approach. We run a suite of unit tests on virtual platforms, and then on physical hardware second. Then we will analyze the results.

Before you begin

We assume that you have a basic knowledge of embedded software development on Arm.

We also assume that you have a basic understanding of the following topics. If you are not familiar with these topics, we have provided some resources to help:

- Virtual platforms in general and Arm Fast Models in particular
- Python and the library PyCADI
- A high-level understanding of Mbed OS

To replicate the example in the guide, you will need:

- A Windows or Linux machine, or a Virtual Machine of Windows or Linux OS
- The Arch Pro hardware development board, available from online retailers
- From Arm, you will need:
    - MPS2+ Fixed Virtual Platforms (FVPs) – Arm Fast Model will give you access to the MPS2+ FVPs. For a free 30-day trial of Arm Fast Model, email license.support@arm.com and specify that you would like to trial the Cortex-M3 Fast Model.

- Arm Compiler – Arm Development Studio will give you access to Arm Compiler. For a free 30-day trial of Arm Development Studio, you will need an Arm account access the download page.

2. Virtual platforms

For tests like unit tests or integration tests, that need to run quickly on small sets of code, virtual platforms offer advantages over hardware, including:

- Speed - Virtual platforms have no overhead for flashing the application on physical hardware. This means that you can save time on small and fast unit tests.
- Scale - Virtual platforms can scale to run many tests in parallel. This makes virtual platforms more cost-effective than a farm of physical hardware.
- Maintenance – Unlike physical hardware, virtual platforms do not overheat, wear out from overuse, break from misuse, or use physical space and resources.
- Upgrades – You can change and reconfigure virtual platforms to match corresponding changes to the underlying hardware platform that is under development. These types of changes can be costly or impossible with physical hardware.

You can use both virtual platforms and hardware in your continuous integration regression test suite. Using both can provide an efficient software development workflow. Testing on virtual platforms is fast and accurate, but may not include specific components that are necessary for full system tests. Typically, there are a lower number of full system tests than small unit tests. This means that running system tests on hardware gives the benefits of a complete system performance check with fewer physical boards in the regression test suite.

Mbed OS is an open-source embedded operating system that provides a suite of tests to execute. These tests validate whether various parts of a platform, either hardware or virtual, work with the various Mbed OS capabilities. With the Mbed OS software base, you can quickly compare the results of virtual platforms and physical hardware as unit testing platforms.

Mbed OS is designed for Cortex-M microcontrollers and supports a wide range of Arm Cortex-M-based devices, including the Arm MPS2+ FPGA board. This FPGA can be programmed to use various Arm Cortex-M cores. Mbed OS also supports tests on the virtual platforms that represent the MPS2+ FPGA board, which are known as a Fixed Virtual Platforms (FVPs). Because unit tests often rely on only the core and memory of a processor, platforms with different peripherals can still be used to test the same code. This is another reason why virtual platforms are so powerful.

In the example in this guide, we test using an Arch Pro board and an MPS2+ FVP, both containing a Cortex-M3 core. The subset of unit tests that are used now require only the core and memory. This means that the MPS2+ FVP is suitable to run the same types of tests that are intended for the Arch Pro board. Using the free Mbed OS tools and a temporary license to Arm Compiler and the MPS2+ FVP, you can compare the test results of hardware compared to a virtual platform.

3. Install Fast Model and Arm Compiler components

In this section you will compile the MPS2+ FVP and route Arm Compiler 6 correctly. Follow these instructions to set up Arm Fast Models to run a simulation using the MPS2+ FVP representing a Cortex-M3 system:

1. Install Arm Fast Models if you have not done this already.
2. Navigate to the `MPS2+` folder representing the Cortex-M3 version.
3. Open the project file

   ```
   cd /ARM/FastModelsPortfolio_11<version>/examples/LISA/FVP_MPS2/Build_Cortex-M3/
   ./FVP_MPS2_Cortex-M3.sgproj
   ```

   With the GUI open, you can see the full FVP_MPS2 subsystem.
4. Click Settings on the top bar.
5. Uncheck the target SystemC integrated simulator and check the target CADI library.
6. Click Apply then click Ok to save changes.
7. Click Save All in the main top bar on the main GUI page, then click Build.

When the build is complete, you see a folder created named for your system type, for example Win64-Release-VC2015. This folder will be pointed to by the Mbed OS tools that we will install in Install Mbed OS components.

Next you will set up Arm Compiler 6 by following these steps:

1. Download Arm Development Studio, containing Arm Compiler 6, as described in Before you begin.
2. Install Arm Development Studio on your machine.
3. Launch the Development Studio IDE. Upon first opening you will be prompted to set up the license to enable the tool. You can also make changes to your license set up at any time using the Arm License Manager within the Help menu.
4. Select Obtain evaluation license and click on Next to generate an evaluation license. Use the login credentials for th Arm account that you used to download Arm Development Studio. You will be prompted to tie your license to a valid HOSTID, which on Windows can be the MAC address of any enabled networking device on your machine. If installing on Linux, this license, or any node-locked license, must be tied to the device that is identified as precisely eth0.
5. Add the path to Arm Compiler 6 to your overall PATH environmental variable, so that Mbed OS knows where the compiler executables are. These paths will be located here:

   ```
   <path-to-ArmDS-install>/Development Studio
   <version>/sw/ARMCompiler6.<version>/bin
   ```

4. Install Mbed OS components

This section explains how to install all the relevant Mbed OS components to run the unit test suite on Arm Fast Models. Follow these instructions to set up Mbed OS and related tools:

1. Create a directory.

   Note

   The default character limit for the path name on Windows 10 is 260 characters for any path. The path name cannot have any spaces in it, or compiling the Mbed OS tests fails.

2. Perform a git clone to install the main Mbed OS library that containing source code and test files. Verify that the version is at or above 1.8.2. Use these commands:

```
git clone https://github.com/ARMmbed/mbed-os
Mbed -version
```

3. Perform a git clone to install the Greentea test harness. Verify that the version is at or above 1.4.0, and that the `mbedgt --fm` output says that you need an argument. Use the following code:

```
git clone https://github.com/ARMmbed/mbed-os-tools/
cd mbed-os-tools/greentea/packages/mbed-greentea
sudo python setup.py install
cd ../../../..
mbedgt --version
mbedgt --fm
```

4. Download and run the installer to install the Mbed CLI interface on Windows. Run the following command on Linux and Mac to install with the Python pip:

```
sudo pip install mbed-cli
```

5. Perform a git clone with the following command to install the `mbed-fastmodel-agent`: `git clone` https://github.com/ARMmbed/mbed-fastmodel-agent The `mbed-fastmodel-agent` allows Arm Fast Models to be used as targets for the Mbed OS tests and software.

6. Edit the file `mbed-fastmodel-agent/fm_agent/settings.json` to set up the `mbed-fastmodel-agent` correctly. Change the following variables:

   - Change these variables under GLOBAL > Windows or GLOBAL >Linux, depending on your system type:

```
"model_lib_path": <path-to-FM-install>/examples/LISA/FVP_MPS2/Build_Cortex-M3/
Win64-Release-VC2015

"PyCADI_path": <path-to-FM-install>/lib/python27
```

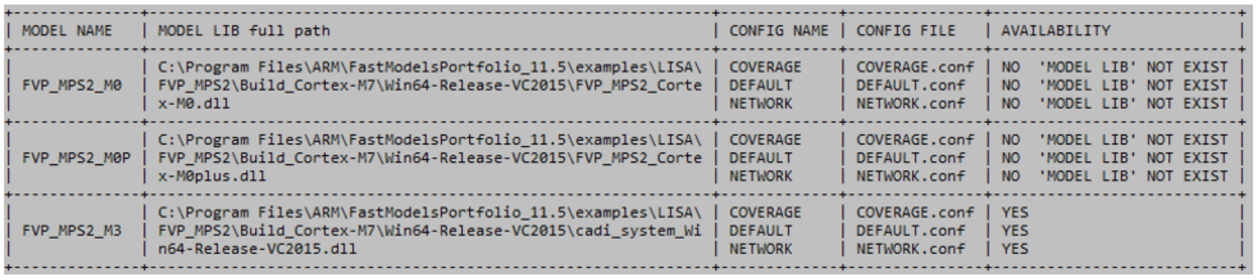
   - Change the `model_lib` variable for your OS under FVP_MPS2_M3 to match either the `.dll` name for Windows or `.so` name for Linux referring to the generated Fast Model binary. For example, on Windows use `"model_lib": "cadi_system_Win64-Release-VC2015.dll"`

7. Build the agent and verify that the program can find the Fast Model executable correctly. Use these commands:

```
cd mbed-fastmodel-agent
python setup.py install
cd ..
mbedfm
```

You should then see an output like what you can see in the following screenshot, indicating that the Mbed tools can locate the Cortex-M3 Fast Model executable:

Figure 4-1: Output



```
+--------------+--------------------------------------------------------------+-------------+---------------+---------------------------+
| MODEL NAME   | MODEL LIB full path                                          | CONFIG NAME | CONFIG FILE   | AVAILABILITY              |
+--------------+--------------------------------------------------------------+-------------+---------------+---------------------------+
|              | C:\Program Files\ARM\FastModelsPortfolio_11.5\examples\LISA\ | COVERAGE    | COVERAGE.conf | NO  'MODEL LIB' NOT EXIST |
| FVP_MPS2_M0  | FVP_MPS2\Build_Cortex-M7\Win64-Release-VC2015\FVP_MPS2_Corte | DEFAULT     | DEFAULT.conf  | NO  'MODEL LIB' NOT EXIST |
|              | x-M0.dll                                                     | NETWORK     | NETWORK.conf  | NO  'MODEL LIB' NOT EXIST |
+--------------+--------------------------------------------------------------+-------------+---------------+---------------------------+
|              | C:\Program Files\ARM\FastModelsPortfolio_11.5\examples\LISA\ | COVERAGE    | COVERAGE.conf | NO  'MODEL LIB' NOT EXIST |
| FVP_MPS2_M0P | FVP_MPS2\Build_Cortex-M7\Win64-Release-VC2015\FVP_MPS2_Corte | DEFAULT     | DEFAULT.conf  | NO  'MODEL LIB' NOT EXIST |
|              | x-M0plus.dll                                                 | NETWORK     | NETWORK.conf  | NO  'MODEL LIB' NOT EXIST |
+--------------+--------------------------------------------------------------+-------------+---------------+---------------------------+
|              | C:\Program Files\ARM\FastModelsPortfolio_11.5\examples\LISA\ | COVERAGE    | COVERAGE.conf | YES                       |
| FVP_MPS2_M3  | FVP_MPS2\Build_Cortex-M7\Win64-Release-VC2015\cadi_system_Wi | DEFAULT     | DEFAULT.conf  | YES                       |
|              | n64-Release-VC2015.dll                                       | NETWORK     | NETWORK.conf  | YES                       |
+--------------+--------------------------------------------------------------+-------------+---------------+---------------------------+
```

5. Build and run unit tests

In this section of the guide, we build and run the unit tests that are provided in our example. First, we compile the tests for both the Arch Pro hardware and MPS2+ FVP. These steps take some time. This is because Mbed OS compiles the entire library to run the unit tests properly on target.

Follow these steps:

1. Plug in your Arch Pro development board over USB.
2. Run the following commands for the Arch Pro:

   ```
   mbedls
   cd mbed-os
   mbed test -t ARM -m ARCH_PRO --compile
   ```

   `mbedls` will verify that you have an Mbed-enabled board connected to your computer. The `mbed test` command compiles all tests with the Arm Compiler.
3. Now we are ready to run the unit test suite on both hardware and virtual platforms to contrast the results. Certain tests run on one platform, and not another, because of differences in the internal hardware. For example, the Arch Pro cannot run some real-time operating system tests and driver tests that it does not support with its hardware. To ensure that the Arch Pro and MPS2+ FVP are running the same test suite, the `-n` switch is added to enumerate the tests to run. Wildcards (`*`) are used to include all of a certain class of test, and commas separate the tests. You must be in the `mbed-os` directory to perform these tests.
4. Run tests on the Arch Pro, using this command:

   ```
   mbedgt -m ARCH_PRO -n features-device_key*,features-frameworks*,features-
   storage*,tests-integration*,tests-mbed_platform*
   ```
5. Run tests on the MPS2+ FVP Cortex-M3, using this command:

   ```
   mbedgt --fm FVP_MPS2_M3:DEFAULT -n features-device_key*,features-
   frameworks*,features-storage*,tests-integration*,tests-mbed_platform*
   ```

   All test suites should pass, and a total time is displayed after completing all tests. The total test time for both the Arch Pro board and the MPS2+ FVP will vary, depending on the host computer bandwidth and performance. The test commands above run 33 test suites with a total of 124 tests on each platform. Different MBed OS versions contain different test modules. This means that you will run more tests, or fewer tests, depending on which version you have. Here are the times to complete all 124 tests for each platform on my machine – you may get different depending on your machine performance:

   - Arch Pro: 488 seconds
   - MPS2+ FVP: 259 seconds

6. Execute tests in parallel

In this section of the guide, we compare the virtual platform tests and hardware tests, and explain how to run tests in parallel to further increase test throughput.

In Build and run unit tests, we saw that running 124 unit tests on the two platforms took this amount of time on the an example system:

- Arch Pro: 488 seconds
- MPS2+ FVP: 259 seconds

These runs both test the same functionality. Because Arm virtual platforms are 100% functionally accurate, the total test time is cut almost in half while losing no information. The speed increase when using virtual platforms compared to hardware can be attributed to several factors, including:

- No flash time
- A faster clock speed on virtual platforms that run on a host OS than a development board

To speed up regression tests even more, you can run virtual platforms in parallel and reduce the testing time. This is another substantial benefit of using virtual platforms for running tests. Scaling with more simulations in parallel to speed up and run more tests is simple and easily maintained as opposed to buying and managing more hardware boards. Running simulations in parallel with Mbed OS is straightforward. You add the `--parallel` switch with the number of runs to perform in parallel.

For example, to run two tests in parallel on the previous test suite, use the following command:

```
mbedgt --fm FVP_MPS2_M3:DEFAULT -n features-device_key*,features-
frameworks*,features-storage*,tests-integration*,tests-mbed_platform* --parallel 2
```

Here are the results of the Arch Pro, MPS2+ FVP, and MPS2+ FVP running two tests in parallel:

Platform	# Tests	Time (seconds)	% Speed increase from Arch Pro
Arch Pro Dev Board	124	488	0%
MPS2+ FVP	124	259	88%
MPS2+ FVP (2 in parallel)	124	160	205%

The run time for a regression test suite decreases by half for each simulation running in parallel. This is limited by the computation power of the host machine and the size of the test suite. Running three tests in parallel results in a speed increase of approximately 300%, running four tests in parallel results in a speed increase of approximately 400%, and so on. Virtual platforms enable this regression suite execution speed increase to most embedded software development testing infrastructures, without sacrificing functional accuracy.

7. Related information

Here are some resources that are related to material in this guide:

- Arm Cortex-M3
- Arm Fast Models
    - Getting started
- Mbed OS
    - Greentea
    - Using Fast Models

8. Next steps

In this guide, we have looked at unit tests that only include the core and memory interaction. We have seen that it is valid to use a Fixed Virtual Platform (FVP) with the same core as the hardware in development, but with different peripherals wanted. If peripherals must be used or tested, however, we must customize these virtual platforms.

Arm Fast Models is the underlying technology for Arm FVPs. Arm Fast Models allows customization of system designs to match a target system. For full flexibility, the Arm Fast Models library comes with a multitude of Arm components that are already modelled. You also have the option to model custom components manually in SystemC. Refer to Arm Fast Models quick start for more information.