Trusted Firmware-A Tests

unknown

Dec 05, 2019

CONTENTS

1 User Guide 1

2 Porting Guide 11

3 Implementing Tests 19

4 Design 21

5 Change Log & Release Notes 25

6 License 35

CHAPTER

ONE

USER GUIDE

This document describes how to build the Trusted Firmware-A Tests (TF-A Tests) and run them on a set of platforms. It assumes that the reader has previous experience building and running the Trusted Firmware-A (TF-A).

1.1 Host machine requirements

The minimum recommended machine specification for building the software and running the FVP models is a dual-core processor running at 2GHz with 12GB of RAM. For best performance, use a machine with a quad-core processor running at 2.6GHz with 16GB of RAM.

The software has been tested on Ubuntu 16.04 LTS (64-bit). Packages used for building the software were installed from that distribution unless otherwise specified.

1.2 Tools

Install the required packages to build TF-A Tests with the following command:

```
sudo apt-get install device-tree-compiler build-essential make git perl libxml-libxml-
↪perl
```

Download and install the GNU cross-toolchain from Linaro. The TF-A Tests have been tested with version 6.2-2016.11 (gcc 6.2):

- AArch32 GNU cross-toolchain
- AArch64 GNU cross-toolchain

In addition, the following optional packages and tools may be needed:

- For debugging, Arm Development Studio 5 (DS-5).

1.3 Getting the TF-A Tests source code

Download the TF-A Tests source code using the following command:

```
git clone https://git.trustedfirmware.org/TF-A/tf-a-tests.git
```

1.4 Building TF-A Tests

- Before building TF-A Tests, the environment variable `CROSS_COMPILE` must point to the Linaro cross compiler.

  For AArch64:

  ```
  export CROSS_COMPILE=<path-to-aarch64-gcc>/bin/aarch64-linux-gnu-
  ```

  For AArch32:

  ```
  export CROSS_COMPILE=<path-to-aarch32-gcc>/bin/arm-linux-gnueabihf-
  ```

- Change to the root directory of the TF-A Tests source tree and build.

  For AArch64:

  ```
  make PLAT=<platform>
  ```

  For AArch32:

  ```
  make PLAT=<platform> ARCH=aarch32
  ```

  Notes:

  – If `PLAT` is not specified, `fvp` is assumed by default. See the *Summary of build options* for more information on available build options.

  – By default this produces a release version of the build. To produce a debug version instead, build the code with `DEBUG=1`.

  – The build process creates products in a `build/` directory tree, building the objects and binaries for each test image in separate sub-directories. The following binary files are created from the corresponding ELF files:

    * `build/<platform>/<build-type>/tftf.bin`
    * `build/<platform>/<build-type>/ns_bl1u.bin`
    * `build/<platform>/<build-type>/ns_bl2u.bin`
    * `build/<platform>/<build-type>/el3_payload.bin`
    * `build/<platform>/<build-type>/cactus_mm.bin`
    * `build/<platform>/<build-type>/cactus.bin`
    * `build/<platform>/<build-type>/ivy.bin`
    * `build/<platform>/<build-type>/quark.bin`

    where `<platform>` is the name of the chosen platform and `<build-type>` is either `debug` or `release`. The actual number of images might differ depending on the platform.

    Refer to the sections below for more information about each image.

- Build products for a specific build variant can be removed using:

  ```
  make DEBUG=<D> PLAT=<platform> clean
  ```

  ... where `<D>` is `0` or `1`, as specified when building.

  The build tree can be removed completely using:

```
make realclean
```

- Use the following command to list all supported build commands:

```
make help
```

1.4.1 TFTF test image

`tftf.bin` is the main test image to exercise the TF-A features. The other test images provided in this repository are optional dependencies that TFTF needs to test some specific features.

`tftf.bin` may be built independently of the other test images using the following command:

```
make PLAT=<platform> tftf
```

In TF-A boot flow, `tftf.bin` replaces the `BL33` image and should be injected in the FIP image. This might be achieved by running the following command from the TF-A root directory:

```
BL33=tftf.bin make PLAT=<platform> fip
```

Please refer to the TF-A User guide for further details.

1.4.2 NS_BL1U and NS_BL2U test images

`ns_bl1u.bin` and `ns_bl2u.bin` are test images that exercise the Firmware Update (FWU) feature of TF-A[1]. Throughout this document, they will be referred as the *FWU test images*.

In addition to updating the firmware, the FWU test images also embed some tests that exercise the FWU state machine implemented in the TF-A. They send valid and invalid SMC requests to the TF-A BL1 image in order to test its robustness.

NS_BL1U test image

The `NS_BL1U` image acts as the *Application Processor (AP) Firmware Update Boot ROM*. This typically is the first software agent executing on the AP in the Normal World during a firmware update operation. Its primary purpose is to load subsequent firmware update images from an external interface, such as NOR Flash, and communicate with `BL1` to authenticate those images.

The `NS_BL1U` test image provided in this repository performs the following tasks:

- Load FWU images from external non-volatile storage (typically flash memory) to Non-Secure RAM.
- Request TF-A BL1 to copy these images in Secure RAM and authenticate them.
- Jump to `NS_BL2U` which carries out the next steps in the firmware update process.

This image may be built independently of the other test images using the following command:

```
make PLAT=<platform> ns_bl1u
```

[1] Therefore, the Trusted Board Boot feature must be enabled in TF-A for the FWU test images to work. Please refer the TF-A User guide for further details.

NS_BL2U test image

The NS_BL2U image acts as the *AP Firmware Updater*. Its primary responsibility is to load a new set of firmware images from an external interface and write them into non-volatile storage.

The NS_BL2U test image provided in this repository overrides the original FIP image stored in flash with the backup FIP image (see below).

This image may be built independently of the other test images using the following command:

```
make PLAT=<platform> ns_bl2u
```

Putting it all together

The FWU test images should be used in conjunction with the TFTF image, as the latter initiates the FWU process by corrupting the FIP image and resetting the target. Once the FWU process is complete, TFTF takes over again and checks that the firmware was successfully updated.

To sum up, 3 images must be built out of the TF-A Tests repository in order to test the TF-A Firmware Update feature:

- ns_bl1u.bin
- ns_bl2u.bin
- tftf.bin

Once that's done, they must be combined in the right way.

- ns_bl1u.bin is a standalone image and does not require any further processing.
- ns_bl2u.bin must be injected into the FWU_FIP image. This might be achieved by setting NS_BL2U=ns_bl2u.bin when building the FWU_FIP image out of the TF-A repository. Please refer to the section Building FIP images with support for Trusted Board Boot in the TF-A User Guide.
- tftf.bin must be injected in the standard FIP image, as explained in section *TFTF test image*.

Additionally, on Juno platform, the FWU FIP must contain a SCP_BL2U image. This image can simply be a copy of the standard SCP_BL2 image if no specific firmware update operations need to be carried on the SCP side.

Finally, the backup FIP image must be created. This can simply be a copy of the standard FIP image, which means that the Firmware Update process will restore the original, uncorrupted FIP image.

1.4.3 EL3 test payload

el3_payload.bin is a test image exercising the alternative EL3 payload boot flow in TF-A. Refer to the EL3 test payload README file for more details about its behaviour and how to build and run it.

1.4.4 SPM test images

This repository contains 3 Secure Partitions that exercise the Secure Partition Manager (SPM) in TF-A[2]. Cactus-MM is designed to test the SPM implementation based on the ARM Management Mode Interface (MM), while Cactus and Ivy can test the SPM implementation based on the SPCI and SPRT draft specifications. Note that it isn't possible to use both communication mechanisms at once: If Cactus-MM is used Cactus and Ivy can't be used.

They run in Secure-EL0 and perform the following tasks:

[2] Therefore, the Secure Partition Manager must be enabled in TF-A for any of the test Secure Partitions to work. Please refer to the TF-A User guide for further details.

- Test that TF-A has correctly setup the secure partition environment: They should be allowed to perform cache maintenance operations, access floating point registers, etc.
- Test that TF-A accepts to change data access permissions and instruction permissions on behalf of the Secure Partitions for memory regions the latter owns.
- Test communication with SPM through either MM, or both SPCI and SPRT.

They are only supported on AArch64 FVP. They can be built independently of the other test images using the following command:

```
make PLAT=fvp cactus ivy cactus_mm
```

In the TF-A boot flow, the partitions replace the `BL32` image and should be injected in the FIP image. To test SPM-MM with Cactus-MM, it is enough to use `cactus_mm.bin` as BL32 image. To test the SPM based on SPCI and SPRT, it is needed to use `sp_tool` to build a Secure Partition package that can be used as BL32 image.

To run the full set of tests in the Secure Partitions, they should be used in conjunction with the TFTF image.

For SPM-MM, the following commands can be used to build the tests:

:: # TF-A-Tests repository:

make PLAT=fvp TESTS=spm-mm tftf cactus_mm

TF-A repository:

make BL33=path/to/tftf.bin BL32=path/to/cactus_mm.bin PLAT=fvp EL3_EXCEPTION_HANDLING=1 ENABLE_SPM=1 all fip

For SPM based on SPCI and SPRT:

:: # TF-A-Tests repository:

make PLAT=fvp TESTS=spm tftf cactus ivy

TF-A repository:

make sptool

tools/sptool/sptool -o sp_package.bin -i path/to/cactus.bin:path/to/cactus.dtb -i path/to/ivy.bin:path/to/ivy.dtb

make BL33=path/to/tftf.bin BL32=path/to/sp_package.bin PLAT=fvp ENABLE_SPM=1 SPM_MM=0 ARM_BL31_IN_DRAM=1 all fip

Please refer to the TF-A User guide for further details.

1.4.5 Summary of build options

As much as possible, TF-A Tests dynamically detect the platform hardware components and available features. There are a few build options to select specific features where the dynamic detection falls short. This section lists them.

Unless mentioned otherwise, these options are expected to be specified at the build command line and are not to be modified in any component makefiles.

Note that the build system doesn't track dependencies for build options. Therefore, if any of the build options are changed from a previous build, a clean build must be performed.

Build options shared across test images

Most of the build options listed in this section apply to TFTF, the FWU test images and Cactus, unless otherwise specified. These do not influence the EL3 payload, whose simplistic build system is mostly independent.

- ARCH: Choose the target build architecture for TF-A Tests. It can take either aarch64 or aarch32 as values. By default, it is defined to aarch64. Not all test images support this build option.
- ARM_ARCH_MAJOR: The major version of Arm Architecture to target when compiling TF-A Tests. Its value must be numeric, and defaults to 8.
- ARM_ARCH_MINOR: The minor version of Arm Architecture to target when compiling TF-A Tests. Its value must be a numeric, and defaults to 0.
- DEBUG: Chooses between a debug and a release build. A debug build typically embeds assertions checking the validity of some assumptions and its output is more verbose. The option can take either 0 (release) or 1 (debug) as values. 0 is the default.
- ENABLE_ASSERTIONS: This option controls whether calls to assert() are compiled out.
  - For debug builds, this option defaults to 1, and calls to assert() are compiled in.
  - For release builds, this option defaults to 0 and calls to assert() are compiled out.

  This option can be set independently of DEBUG. It can also be used to hide any auxiliary code that is only required for the assertion and does not fit in the assertion itself.
- LOG_LEVEL: Chooses the log level, which controls the amount of console log output compiled into the build. This should be one of the following:

```
0  (LOG_LEVEL_NONE)
10 (LOG_LEVEL_ERROR)
20 (LOG_LEVEL_NOTICE)
30 (LOG_LEVEL_WARNING)
40 (LOG_LEVEL_INFO)
50 (LOG_LEVEL_VERBOSE)
```

  All log output up to and including the selected log level is compiled into the build. The default value is 40 in debug builds and 20 in release builds.
- PLAT: Choose a platform to build TF-A Tests for. The chosen platform name must be a subdirectory of any depth under plat/, and must contain a platform makefile named platform.mk. For example, to build TF-A Tests for the Arm Juno board, select PLAT=juno.
- V: Verbose build. If assigned anything other than 0, the build commands are printed. Default is 0.

TFTF build options

- ENABLE_PAUTH: Boolean option to enable ARMv8.3 Pointer Authentication (ARMv8.3-PAuth) support in the Trusted Firmware-A Test Framework itself. If enabled, it is needed to use a compiler that supports the option -mbranch-protection (GCC 9 and later). It defaults to 0.
- NEW_TEST_SESSION: Choose whether a new test session should be started every time or whether the framework should determine whether a previous session was interrupted and resume it. It can take either 1 (always start new session) or 0 (resume session as appropriate). 1 is the default.
- TESTS: Set of tests to run. Use the following command to list all possible sets of tests:

```
make help_tests
```

  If no set of tests is specified, the standard tests will be selected (see tftf/tests/tests-standard.xml).
- USE_NVM: Used to select the location of test results. It can take either 0 (RAM) or 1 (non-volatile memory like flash) as test results storage. Default value is 0, as writing to the flash significantly slows tests down.

FWU test images build options

- `FIRMWARE_UPDATE`: Whether the Firmware Update test images (i.e. `NS_BL1U` and `NS_BL2U`) should be built. The default value is 0. The platform makefile is free to override this value if Firmware Update is supported on this platform.

Arm FVP platform specific build options

- `FVP_MAX_PE_PER_CPU`: Sets the maximum number of PEs implemented on any CPU in the system. It can take either 1 or 2 values. This option defaults to 1.

1.5 Checking source code style

When making changes to the source for submission to the project, the source must be in compliance with the Linux style guide. To assist with this, the project Makefile provides two targets, which both utilise the `checkpatch.pl` script that ships with the Linux source tree.

To check the entire source tree, you must first download copies of `checkpatch.pl`, `spelling.txt` and `const_structs.checkpatch` available in the Linux master tree scripts directory, then set the `CHECKPATCH` environment variable to point to `checkpatch.pl` (with the other 2 files in the same directory).

Then use the following command:

```
make CHECKPATCH=<path-to-linux>/linux/scripts/checkpatch.pl checkcodebase
```

To limit the coding style checks to your local changes, use:

```
make CHECKPATCH=<path-to-linux>/linux/scripts/checkpatch.pl checkpatch
```

By default, this will check all patches between `origin/master` and your local branch. If you wish to use a different reference commit, this can be specified using the `BASE_COMMIT` variable.

1.6 Running the TF-A Tests

Refer to the sections Running the software on FVP and Running the software on Juno in TF-A User Guide. The same instructions mostly apply to run the TF-A Tests on those 2 platforms. The difference is that the following images are not needed here:

- Normal World bootloader. The TFTF replaces it in the boot flow;
- Linux Kernel;
- Device tree;
- Filesystem.

In other words, only the following software images are needed:

- `BL1` firmware image;
- `FIP` image containing the following images:
  - `BL2`;
  - `SCP_BL2` if required by the platform (e.g. Juno);

- `BL31`;
- `BL32` (optional);
- `tftf.bin` (standing as the BL33 image).

1.6.1 Running the manual tests on FVP

The manual tests rely on storing state in non-volatile memory (NVM) across reboot. On FVP the NVM is not persistent across reboots, so the following flag must be used to write the NVM to a file when the model exits.

:: -C bp.flashloader0.fnameWrite=[filename]

To ensure the model exits on shutdown the following flag must be used:

:: -C bp.ve_sysregs.exit_on_shutdown=1

After the model has been shutdown, this file must be fed back in to continue the test. Note this flash file includes the FIP image, so the original fip.bin does not need to be passed in. The following flag is used:

```
-C bp.flashloader0.fname=[filename]
```

1.6.2 Running the FWU tests

As previously mentioned in section *Putting it all together*, there are a couple of extra images involved when running the FWU tests. They need to be loaded at the right addresses, which depend on the platform.

FVP

In addition to the usual BL1 and FIP images, the following extra images must be loaded:

- `NS_BL1U` image at address `0x0BEB8000` (i.e. NS_BL1U_BASE macro in TF-A)
- `FWU_FIP` image at address `0x08400000` (i.e. NS_BL2U_BASE macro in TF-A)
- `Backup FIP` image at address `0x09000000` (i.e. FIP_BKP_ADDRESS macro in TF-A tests).

An example script is provided in scripts/run_fwu_fvp.sh.

Juno

The same set of extra images and load addresses apply for Juno as for FVP.

The new images must be programmed in flash memory by adding some entries in the `SITE1/HBI0262x/images.txt` configuration file on the Juno SD card (where `x` depends on the revision of the Juno board). Refer to the Juno Getting Started Guide, section 2.3 "Flash memory programming" for more information. Users should ensure these do not overlap with any other entries in the file.

Addresses in this file are expressed as an offset from the base address of the flash (that is, `0x08000000`).

```
NOR10UPDATE: AUTO                       ; Image Update:NONE/AUTO/FORCE
NOR10ADDRESS: 0x00400000                ; Image Flash Address
NOR10FILE: \SOFTWARE\fwu_fip.bin        ; Image File Name
NOR10LOAD: 00000000                     ; Image Load Address
NOR10ENTRY: 00000000                    ; Image Entry Point
```

(continues on next page)

(continued from previous page)

```
NOR11UPDATE: AUTO                       ; Image Update:NONE/AUTO/FORCE
NOR11ADDRESS: 0x03EB8000                ; Image Flash Address
NOR11FILE: \SOFTWARE\ns_bl1u.bin        ; Image File Name
NOR11LOAD: 00000000                     ; Image Load Address
NOR11ENTRY: 00000000                    ; Image Load Address

NOR12UPDATE: AUTO                       ; Image Update:NONE/AUTO/FORCE
NOR12ADDRESS: 0x01000000                ; Image Flash Address
NOR12FILE: \SOFTWARE\backup_fip.bin     ; Image File Name
NOR12LOAD: 00000000                     ; Image Load Address
NOR12ENTRY: 00000000                    ; Image Entry Point
```

CHAPTER

TWO

PORTING GUIDE

2.1 Introduction

Please note that this document is incomplete.

Porting the TF-A Tests to a new platform involves making some mandatory and optional modifications for both the cold and warm boot paths. Modifications consist of:

- Implementing a platform-specific function or variable,
- Setting up the execution context in a certain way, or
- Defining certain constants (for example #defines).

The platform-specific functions and variables are all declared in `include/plat/common/platform.h`. The framework provides a default implementation of variables and functions to fulfill the optional requirements. These implementations are all weakly defined; they are provided to ease the porting effort. Each platform port can override them with its own implementation if the default implementation is inadequate.

2.2 Platform requirements

The TF-A Tests rely on the following features to be present on the platform and accessible from Normal World.

- Watchdog
- Non-Volatile Memory
- System Timer

This also means that a platform port of the TF-A Tests must include software drivers for those features.

2.3 Mandatory modifications

2.3.1 File : platform_def.h [mandatory]

Each platform must ensure that a header file of this name is in the system include path with the following constants defined. This may require updating the list of `PLAT_INCLUDES` in the `platform.mk` file. In the ARM FVP port, this file is found in `plat/arm/board/fvp/include/platform_def.h`.

- **#define : PLATFORM_LINKER_FORMAT**

  Defines the linker format used by the platform, for example *elf64-littleaarch64* used by the FVP.

- **#define : PLATFORM_LINKER_ARCH**

  Defines the processor architecture for the linker by the platform, for example *aarch64* used by the FVP.

- **#define : PLATFORM_STACK_SIZE**

  Defines the stack memory available to each CPU. This constant is used by `plat/common/aarch64/platform_mp_stack.S`.

- **#define : PLATFORM_CLUSTER_COUNT**

  Defines the total number of clusters implemented by the platform in the system.

- **#define : PLATFORM_CORE_COUNT**

  Defines the total number of CPUs implemented by the platform across all clusters in the system.

- **#define : PLATFORM_NUM_AFFS**

  Defines the total number of nodes in the affinity hierarchy at all affinity levels used by the platform.

- **#define : PLATFORM_MAX_AFFLVL**

  Defines the maximum number of affinity levels in the system that the platform implements. ARMv8-A has support for 4 affinity levels. It is likely that hardware will implement fewer affinity levels. For example, the Base AEM FVP implements two clusters with a configurable number of CPUs. It reports the maximum affinity level as 1.

- **#define : PLAT_MAX_SPI_OFFSET_ID**

  Defines the offset of the last Shared Peripheral Interrupt supported by the TF-A Tests on this platform. SPI numbers are mapped onto GIC interrupt IDs, starting from interrupt ID 32. In other words, this offset ID corresponds to the last SPI number, to which 32 must be added to get the corresponding last GIC IRQ ID.

  E.g. If `PLAT_MAX_SPI_OFFSET_ID` is 10, this means that IRQ #42 is the last SPI.

- **#define : PLAT_LOCAL_PSTATE_WIDTH**

  Defines the bit-field width of the local state in State-ID field of the power-state parameter. This macro will be used to compose the State-ID field given the local power state at different affinity levels.

- **#define : PLAT_MAX_PWR_STATES_PER_LVL**

  Defines the maximum number of power states at a power domain level for the platform. This macro will be used by the `PSCI_STAT_COUNT/RESIDENCY` tests to determine the size of the array to allocate for storing the statistics.

- **#define : TFTF_BASE**

  Defines the base address of the TFTF binary in DRAM. Used by the linker script to link the image at the right address. Must be aligned on a page-size boundary.

- **#define : IRQ_PCPU_NS_TIMER**

  Defines the IRQ ID of the per-CPU Non-Secure timer of the platform.

- **#define : IRQ_CNTPSIRQ1**

  Defines the IRQ ID of the System timer of the platform.

- **#define : TFTF_NVM_OFFSET**

  The TFTF needs some Non-Volatile Memory to store persistent data. This defines the offset from the beginning of this memory that the TFTF can use.

- **#define : TFTF_NVM_SIZE**

  Defines the size of the Non-Volatile Memory allocated for TFTF usage.

If the platform port uses the ARM Watchdog Module (SP805) peripheral, the following constant needs to be defined:

- **#define : SP805_WDOG_BASE**

  Defines the base address of the SP805 watchdog peripheral.

If the platform port uses the IO storage framework, the following constants must also be defined:

- **#define : MAX_IO_DEVICES**

  Defines the maximum number of registered IO devices. Attempting to register more devices than this value using `io_register_device()` will fail with `IO_RESOURCES_EXHAUSTED`.

- **#define : MAX_IO_HANDLES**

  Defines the maximum number of open IO handles. Attempting to open more IO entities than this value using `io_open()` will fail with `IO_RESOURCES_EXHAUSTED`.

If the platform port uses the VExpress NOR flash driver (see `drivers/io/vexpress_nor/`), the following constants must also be defined:

- **#define : NOR_FLASH_BLOCK_SIZE**

  Defines the largest block size as seen by the software while writing to NOR flash.

2.3.2 Function : tftf_plat_arch_setup() [mandatory]

```
Argument : void
Return   : void
```

This function performs any platform-specific and architectural setup that the platform requires.

In both the ARM FVP and Juno ports, this function configures and enables the MMU.

2.3.3 Function : tftf_early_platform_setup() [mandatory]

```
Argument : void
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU. It is used to perform platform-specific actions very early in the boot.

In both the ARM FVP and Juno ports, this function configures the console.

2.3.4 Function : tftf_platform_setup() [mandatory]

```
Argument : void
Return   : void
```

This function executes with the MMU and data caches enabled. It is responsible for performing any remaining platform-specific setup that can occur after the MMU and data cache have been enabled.

This function is also responsible for initializing the storage abstraction layer used to access non-volatile memory for permanent storage of test results. It also initialises the GIC and detects the platform topology using platform-specific means.

2.3.5 Function : plat_get_nvm_handle() [mandatory]

```
Argument : uintptr_t *
Return   : void
```

It is needed if the platform port uses IO storage framework. This function is responsible for getting the pointer to the initialised non-volatile memory entity.

2.3.6 Function : tftf_plat_get_pwr_domain_tree_desc() [mandatory]

```
Argument : void
Return   : const unsigned char *
```

This function returns the platform topology description array in a suitable format as expected by TFTF. The size of the array is expected to be `PLATFORM_NUM_AFFS - PLATFORM_CORE_COUNT + 1`. The format used to describe this array is :

1. The first entry in the array specifies the number of power domains at the highest power level implemented in the platform. This caters for platforms where the power domain tree does not have a single root node e.g. the FVP which has two cluster power domains at the highest level (that is, 1).
2. Each subsequent entry corresponds to a power domain and contains the number of power domains that are its direct children.

The array format is the same as the one used by Trusted Firmware-A and more details of its description can be found in the Trusted Firmware-A documentation: docs/psci-pd-tree.rst.

2.3.7 Function : tftf_plat_get_mpidr() [mandatory]

```
Argument : unsigned int
Return   : uint64_t
```

This function converts a given *core_pos* into a valid MPIDR if the CPU is present in the platform. The *core_pos* is a unique number less than the `PLATFORM_CORE_COUNT` returned by `platform_get_core_pos()` for a given CPU. This API is used by the topology framework in TFTF to query the presence of a CPU and, if present, returns the corresponding MPIDR for it. If the CPU referred to by the *core_pos* is absent, then this function returns `INVALID_MPID`.

2.3.8 Function : plat_get_state_prop() [mandatory]

```
Argument : unsigned int
Return   : const plat_state_prop_t *
```

This functions returns the `plat_state_prop_t` array for all the valid low power states from platform for a specified affinity level and returns `NULL` for an invalid affinity level. The array is expected to be NULL-terminated. This function is expected to be used by tests that need to compose the power state parameter for use in `PSCI_CPU_SUSPEND` API or `PSCI_STAT/RESIDENCY` API.

2.3.9 Function : plat_fwu_io_setup() [mandatory]

```
Argument : void
Return   : void
```

This function initializes the IO system used by the firmware update.

2.3.10 Function : plat_arm_gic_init() [mandatory]

```
Argument : void
Return   : void
```

This function initializes the ARM Generic Interrupt Controller (GIC).

2.3.11 Function : platform_get_core_pos() [mandatory]

```
Argument : u_register_t
Return   : unsigned int
```

This function returns a linear core ID from a MPID.

2.3.12 Function : plat_crash_console_init() [mandatory]

```
Argument : void
Return   : int
```

This function initializes a platform-specific console for crash reporting.

2.3.13 Function : plat_crash_console_putc() [mandatory]

```
Argument : int
Return   : int
```

This function prints a character on the platform-specific crash console.

2.3.14 Function : plat_crash_console_flush() [mandatory]

```
Argument : void
Return   : int
```

This function waits until all the characters of the platform-specific crash console have been actually printed.

2.4 Optional modifications

The following are helper functions implemented by the test framework that perform common platform-specific tasks. A platform may choose to override these definitions.

2.4.1 Function : platform_get_stack()

```
Argument : unsigned long
Return   : unsigned long
```

This function returns the base address of the memory stack that has been allocated for the CPU specified by MPIDR. The size of the stack allocated to each CPU is specified by the platform defined constant `PLATFORM_STACK_SIZE`.

Common implementation of this function is provided in `plat/common/aarch64/platform_mp_stack.S`.

2.4.2 Function : tftf_platform_end()

```
Argument : void
Return   : void
```

This function performs any operation required by the platform to properly finish the test session.

The default implementation sends an EOT (End Of Transmission) character on the UART. This can be used to automatically shutdown the FVP models. When running on real hardware, the UART output may be parsed by an external tool looking for this character and rebooting the platform for example.

2.4.3 Function : tftf_plat_reset()

```
Argument : void
Return   : void
```

This function resets the platform.

The default implementation uses the ARM watchdog peripheral (SP805) to generate a watchdog timeout interrupt. This interrupt remains deliberately unserviced, which eventually asserts the reset signal.

2.5 Storage abstraction layer

In order to improve platform independence and portability a storage abstraction layer is used to store test results to non-volatile platform storage.

Each platform should register devices and their drivers via the Storage layer. These drivers then need to be initialized in `tftf_platform_setup()` function.

It is mandatory to implement at least one storage driver. For the FVP and Juno platforms the NOR Flash driver is provided as the default means to store test results to storage. The storage layer is described in the header file `include/lib/io_storage.h`. The implementation of the common library is in `drivers/io/io_storage.c` and the driver files are located in `drivers/io/`.

2.6 Build Flags

- **PLAT_TESTS_SKIP_LIST**

This build flag can be defined by the platform to control exclusion of some testcases from the default test plan for a platform. If used this needs to point to a text file which follows the following criteria:

- Contain a list of tests to skip for this platform.

- Specify 1 test per line, using the following format:

```
testsuite_name/testcase_name
```

  where `testsuite_name` and `testcase_name` are the names that appear in the XML tests file.

- Alternatively, it is possible to disable a test suite entirely, which will disable all test cases part of this test suite. To do so, only specify the test suite name, omitting the `/testcase_name` part.

CHAPTER

THREE

IMPLEMENTING TESTS

This document aims at providing some pointers to help implementing new tests in the TFTF image.

3.1 Structure of a test

A test might be divided into 3 logical parts, detailed in the following sections.

3.1.1 Prologue

A test has a main entry point function, whose type is:

```
typedef test_result_t (*test_function_t)(void);
```

See tftf/framework/include/tftf.h.

Only the primary CPU enters this function, while other CPUs are powered down.

First of all, the test function should check whether this test is applicable to this platform and environment. Some tests rely on specific hardware features or firmware capabilities to be present. If these are not available, the test should be skipped. For example, a multi-core test requires at least 2 CPUs to run. Macros and functions are provided in include/common/test_helpers.h to help test code verify that their requirements are met.

3.1.2 Core

This is completely dependent on the purpose of the test. The paragraphs below just provide some useful, general information.

The primary CPU may power on other CPUs by calling the function `tftf_cpu_on()`. It provides an address to which secondary CPUs should jump to once they have been initialized by the test framework. This address should be different from the primary CPU test function.

Synchronization primitives are provided in include/lib/events.h in case CPUs' execution threads need to be synchronized. Most multi-processing tests will need some synchronisation points that all/some CPUs need to reach before test execution may continue.

Any CPU that is involved in a test must return from its test function. Failure to do so will put the framework in an unrecoverable state, see the TFTF known limitations. The return code indicates the test result from the point of view of this CPU. At the end of the test, individual CPU results are aggregated and the overall test result is derived from that. A test is considered as passed if all involved CPUs reported a success status code.

3.1.3 Epilogue

Each test is responsible for releasing any allocated resources and putting the system back in a clean state when it finishes. Any change to the system configuration (e.g. MMU setup, GIC configuration, system registers, ...) must be undone and the original configuration must be restored. This guarantees that the next test is not affected by the actions of the previous one.

One exception to this rule is that CPUs powered on as part of a test must not be powered down. As already stated above, as soon as a CPU enters the test, the framework expects it to return from the test.

3.2 Template test code

Some template test code is provided in tftf/tests/template_tests. It can be used as a starting point for developing new tests. Template code for both single-core and multi-core tests is provided.

3.3 Plugging the test into the build system

All test code is located under the tftf/tests directory. Tests are usually divided into categories represented as sub-directories under `tftf/tests/`.

The source file implementing the new test code should be added to the appropriate tests makefile, see *.mk* files under tftf/tests.

The new test code should also appear in a tests manifest, see `*.xml` files under tftf/tests. A unique name and test function must be provided. An optional description may be provided as well.

For example, to create a test case named "`Foo test case`", whose test function is `foo()`, add the following line in the tests manifest:

```
<testcase name="Foo test case" function="foo" />
```

A testcase must be part of a testsuite. The `testcase` XML node above must be inside a `testsuite` XML node. A unique name and a description must be provided for the testsuite.

For example, to create a test suite named "`Bar test suite`", whose description is: "`An example test suite`", add the following 2 lines:

```
<testsuite name="Bar test suite" description="An example test suite">
</testsuite>
```

See the template test manifest for reference: tftf/tests/tests-template.xml.

CHAPTER

FOUR

DESIGN

This document provides some details about the internals of the TF-A Tests design. It is incomplete at the moment.

4.1 Global overview of the TF-A tests behaviour

The EL3 firmware is expected to hand over to the TF-A tests with all secondary cores powered down, i.e. only the primary core should enter the TF-A tests.

The primary CPU initialises the platform and the TF-A tests internal data structures.

Then the test session begins. The TF-A tests are executed one after the other. Tests results are saved in non-volatile memory as we go along.

Once all tests have completed, a report is printed over the serial console.

4.2 Global Code Structure

The code is organised into the following categories (present as directories at the top level or under the `tftf/` directory):

- **Drivers.**

  Some examples follow, this list might not be exhaustive.

  - Generic GIC driver.

    `arm_gic.h` contains the public APIs that tests might use. Both GIC architecture versions 2 and 3 are supported.

  - PL011 UART driver.

  - VExpress NOR flash driver.

    Note that tests are not expected to use this driver in most cases. Instead, they should use the `tftf_nvm_read()` and `tftf_nvm_write()` wrapper APIs. See definitions in `tftf/framework/include/nvm.h`. See also the NVM validation test cases (`tftf/tests/framework_validation_tests/test_validation_nvm.c`) for an example of usage of these functions.

  - SP805 watchdog.

    Used solely to generate an interrupt that will reset the system on purpose (used in `tftf_plat_reset()`).

  - SP804 timer.

    This is used as the system timer on Juno. It is configured such that an interrupt is generated when it reaches 0. It is programmed in one-shot mode, i.e. it must be rearmed every time it reaches 0.

- **Framework.**

  Core features of the test framework.

- **Library code.**

  Firstly, there is `include/libc/` which provides standard C library functions like `memcpy()`, `printf()` and so on. Additionally, various other APIs are provided under `include/lib/`. The below list gives some examples but might not be exhaustive.

  - `aarch64/`

    Architecture helper functions for e.g. system registers access, cache maintenance operations, MMU configuration, ...

  - `events.h`

    Events API. Used to create synchronisation points between CPUs in tests.

  - `irq.h`

    IRQ handling support. Used to configure IRQs and register/unregister handlers called upon reception of a specific IRQ.

  - `power_management.h`

    Power management operations (CPU ON/OFF, CPU suspend, etc.).

  - `sgi.h`

    Software Generated Interrupt support. Used as an inter-CPU communication mechanism.

  - `spinlock.h`

    Lightweight implementation of synchronisation locks. Used to prevent concurrent accesses to shared data structures.

  - `timer.h`

    Support for programming the timer. Any timer which is in the *always-on* power domain can be used to exit CPUs from suspend state.

  - `tftf_lib.h`

    Miscellaneous helper functions/macros: MP-safe `printf()`, low-level PSCI wrappers, insertion of delays, raw SMC interface, support for writing a string in the test report, macros to skip tests on platforms that do not meet topology requirements, etc.

  - `io_storage.h`

    Low-level IO operations. Tests are not expected to use these APIs directly. They should use higher-level APIs like `tftf_nvm_read()` and `tftf_nvm_write()`.

- **Platform specific.**

  Note that `include/plat/common/plat_topology.h` provides the interfaces that a platform must implement to support topology discovery (i.e. how many CPUs and clusters there are).

- **Tests.**

  The tests are divided into the following categories (present as directories in the `tftf/tests/` directory):

- **Framework validation tests.**

  Tests that exercise the core features of the framework. Verify that the test framework itself works properly.

- **Runtime services tests.**

  Tests that exercise the runtime services offered by the EL3 Firmware to the Normal World software. For example, this includes tests for the Standard Service (to which PSCI belongs to), the Trusted OS service or the SiP service.

- **CPU extensions tests.**

  Tests some CPU extensions features. For example, the AMU tests ensure that the counters provided by the Activity Monitor Unit are behaving correctly.

- **Firmware Update tests.**

  Tests that exercise the Firmware Update feature of TF-A.

- **Template tests.**

  Sample test code showing how to write tests in practice. Serves as documentation.

- **Performance tests.**

  Simple tests measuring the latency of an SMC call.

- **Miscellaneous tests.**

  Tests for RAS support, correct system setup, ...

All assembler files have the `.S` extension. The linker source file has the extension `.ld.S`. This is processed by GCC to create the linker script which has the extension `.ld`.

4.3 Detailed Code Structure

The cold boot entry point is `tftf_entrypoint` (see `tftf/framework/aarch64/entrypoint.S`). As explained in section *Global overview of the TF-A tests behaviour*, only the primary CPU is expected to execute this code.

Tests can power on other CPUs using the function `tftf_cpu_on()`. This uses the PSCI `CPU_ON` API of the EL3 Firmware. When entering the Normal World, execution starts at the warm boot entry point, which is `tftf_hotplug_entry()` (see `tftf/framework/aarch64/entrypoint.S`).

Information about the progression of the test session and tests results are written into Non-Volatile Memory as we go along. This consists of the following data (see struct `tftf_state_t` typedef in `tftf/framework/include/nvm.h`):

- `test_to_run`

  Reference to the test to run.

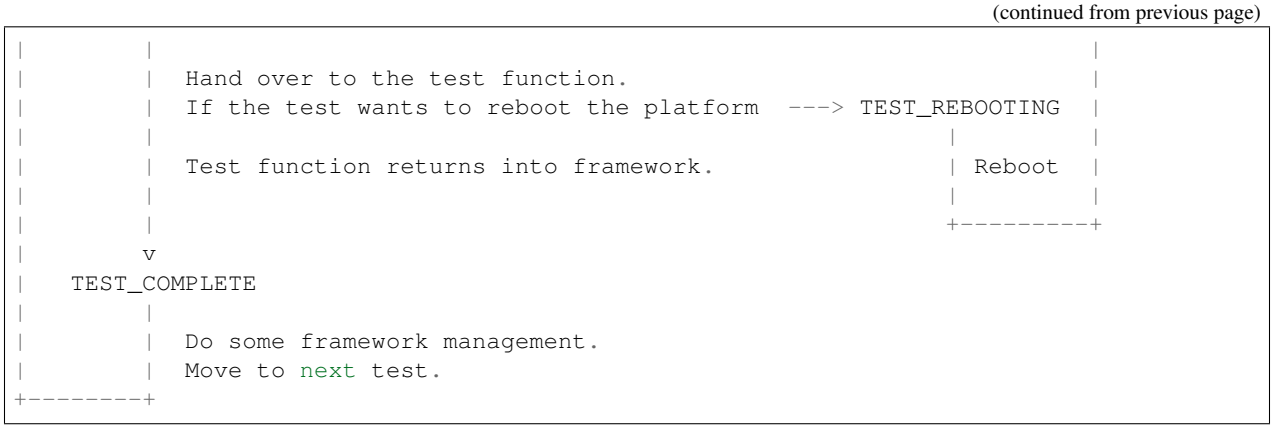
- `test_progress`

  Progress in the execution of `test_to_run`. This is used to implement the following state machine:

```
+-> TEST_READY (initial state of the test)                    <---------------+
|         |                                                                     |
|         |  Test framework prepares the test environment.                      |
|         |                                                                     |
|         v                                                                     |
|    TEST_IN_PROGRESS                                                           |
```

(continues on next page)

(continued from previous page)



- `testcase_buffer`

  A buffer that the test can use as a scratch area for whatever it is doing.

- `testcase_results`
- `result_buffer_size`
- `result_buffer`

  Buffer holding the tests output. Tests output are concatenated.

4.4 Interrupts management

The TF-A tests expect SGIs #0 to #7 to be available for their own usage. In particular, this means that Trusted World software must configure them as non-secure interrupts.

SGI #7 has a special status. It is the SGI that the timer management framework sends to all CPUs when the system timer fires off (see the definition of the constant `IRQ_WAKE_SGI` in the header file `include/lib/irq.h`). Although test cases can use this specific SGI - e.g. they can register an IRQ handler for it and use it as an inter-CPU communication mechanism - they have to be aware of the underlying consequences. Some tests, like the PSCI `CPU_SUSPEND` tests, rely on this SGI to be enabled in order to wake up CPUs from their suspend state. If it is disabled, these tests will leave the system in an unresponsive state.

CHAPTER

FIVE

CHANGE LOG & RELEASE NOTES

Please note that the Trusted Firmware-A Tests version follows the Trusted Firmware-A version for simplicity. At any point in time, TF-A Tests version *x.y* aims at testing TF-A version *x.y*. Different versions of TF-A and TF-A Tests are not guaranteed to be compatible. This also means that a version upgrade on the TF-A-Tests side might not necessarily introduce any new feature.

5.1 Version 2.2

5.1.1 New features

- A wide range of tests are made available in this release to help validate the functionality of TF-A.
- Various improvements to test framework and test suite.

TFTF

- Enhancement to xlat table library synchronous to TF-A code base.
- Enabled strict alignment checks (SCTLR.A & SCTLR.SA) in all images.
- Support for a simple console driver. Currently it serves as a placeholder with empty functions.
- A topology helper API is added in the framework to get parent node info.
- Support for FVP with clusters having upto 8 CPUs.
- Enhanced linker script to separate code and RO data sections.
- Relax SMC calls tests. The SMCCC specification recommends Trusted OSes to mitigate the risk of leaking information by either preserving the register state over the call, or returning a constant value, such as zero, in each register. Tests only allowed the former behaviour and have been extended to allow the latter as well.
- Pointer Authentication enabled on warm boot path with individual APIAKey generation for each CPU.
- New tests:
  - Basic unit tests for xlat table library v2.
  - Tests for validating SVE support in TF-A.
  - Stress tests for dynamic xlat table library.
  - PSCI test to measure latencies when turning ON a cluster.
  - Series of AArch64 tests that stress the secure world to leak sensitive counter values.
  - Test to validate PSCI SYSTEM_RESET call.

  - Basic tests to validate Memory Tagging Extensions are being enabled and ensuring no undesired leak of sensitive data occurs.

- Enhanced tests:
  - Improved tests for Pointer Authentication support. Checks are performed to see if pointer authentication keys are accessible as well as validate if secure keys are being leaked after a PSCI version call or TSP call.
  - Improved AMU test to remove unexecuted code iterating over Group1 counters and fix the conditional check of AMU Group0 counter value.

Secure partitions

A new Secure Partition Quark is introduced in this release.

Quark

The Quark test secure partition provided is a simple service which returns a magic number. Further, a simple test is added to test if Quark is functional.

5.1.2 Issues resolved since last release

- Bug fix in libc memchr implementation.
- Bug fix in calculation of number of CPUs.
- Streamlined SMC WORKAROUND_2 test and fixed a false fail on Cortex-A76 CPU.
- Pointer Authentication support is now available for secondary CPUs and the corresponding tests are stable in this release.

5.1.3 Known issues and limitations

The sections below list the known issues and limitations of each test image provided in this repository. Unless and otherwise stated, issues and limitations stated in previous release continue to exist in this release.

TFTF

- Multicore spurious interrupt test is observed to have unstable behavior. As a temporary solution, this test is skipped for AArch64 Juno configurations.
- Generating SVE instructions requires *O3* compilation optimization. Since the current build structure does not allow compilation flag modification for specific files, the function which tests support for SVE has been pre-compiled and added as an assembly file.

5.2 Version 2.1

5.2.1 New features

- Add initial support for testing Secure Partition Client Interface (SPCI) and Secure Partition Run-Time (SPRT) standards.

Exercise the full communication flow throughout the software stack, involving:

  - A Secure-EL0 test partition as the Trusted World agent.
  - TFTF as the Normal World agent.
  - The Secure Partition Manager (SPM) in TF-A.

- Various stability improvements, code refactoring and clean ups.

TFTF

- Reorganize tests build infrastructure to allow the selection of a subset of tests.
- Reorganize the platform layer for improved clarity and simplicity.
- Sanitise inclusion of drivers header files.
- Enhance the test report format for improved clarity and conciseness.
- Dump CPU registers when hitting an unexpected exception. Previously, this would silently loop forever.
- Import libc from TF-A to better align the two code bases.
- New tests:
  - SPM tests for exercising communication through either the MM or SPCI/SPRT interfaces.
  - SMC calling convention tests.
  - Initial tests for Armv8.3 Pointer Authentication support (experimental).
- New platform ports:
  - Arm SGI-575 FVP.
  - Hikey960 board (experimental).
  - Arm Neoverse Reference Design N1 Edge (RD-N1-Edge) FVP (experimental).

Secure partitions

We now have 3 Secure Partitions to test the SPM implementation in TF-A.

Cactus-MM

The Cactus test secure partition provided in version 2.0 has been renamed into "*Cactus-MM*". It is still responsible for testing the SPM implementation based on the Arm Management Mode Interface.

Cactus

This is a new test secure partition (as the former "*Cactus*" has been renamed into "*Cactus-MM*", see above).

Unlike *Cactus-MM*, this image tests the SPM implementation based on the SPCI and SPRT draft specifications.

It runs in Secure-EL0 and performs the following tasks:

- Test that TF-A has correctly setup the secure partition environment (access to cache maintenance operations, to floating point registers, etc.)

- Test that TF-A accepts to change data access permissions and instruction permissions on behalf of Cactus for memory regions the latter owns.
- Test communication with SPM through SPCI/SPRT interfaces.

Ivy

This is also a new test secure partition. It is provided in order to test multiple partitions support in TF-A. It is derived from Cactus and essentially provides the same services but with different identifiers at the moment.

EL3 payload

- New platform ports:
  - Arm SGI-575 FVP.
  - Arm Neoverse Reference Design N1 Edge (RD-N1-Edge) FVP (experimental).

5.2.2 Issues resolved since last release

- The GICv2 spurious IRQ test is no longer Juno-specific. It is now only GICv2-specific.
- The manual tests in AArch32 state now work properly. After investigation, we identified that this issue was not AArch32 specific but concerned any test relying on state information persisting across reboots. It was due to an incorrect build configuration.
- Cactus-MM now successfully links with GNU toolchain 7.3.1.

5.2.3 Known issues and limitations

The sections below lists the known issues and limitations of each test image provided in this repository.

TFTF

The TFTF test image might be conceptually sub-divided further in 2 parts: the tests themselves, and the test framework they are based upon.

Test framework

- Some stability issues.
- No mechanism to abort tests when they time out (e.g. this could be implemented using a watchdog).
- No convenient way to include or exclude tests on a per-platform basis.
- Power domains and affinity levels are considered equivalent but they may not necessarily be.
- Need to provide better support to alleviate duplication of test code. There are some recurrent test patterns for which helper functions should be provided. For example, bringing up all CPUs on the platform and executing the same function on all of them, or programming an interrupt and waiting for it to trigger.
- Every CPU that participates in a test must return from the test function. If it does not - e.g. because it powered itself off for testing purposes - then the test framework will wait forever for this CPU. This limitation is too restrictive for some tests.

- No protection against interrupted flash operations. If the target is reset while some data is written to flash, the test framework might behave incorrectly on reset.
- When compiling the code, if the generation of the `tests_list.c` and/or `tests_list.h` files fails, the build process is not aborted immediately and will only fail later on.
- The directory layout requires further improvements. Most of the test framework code has been moved under the `tftf/` directory to better isolate it but this effort is not complete. As a result, there are still some TFTF files scattered around.
- Pointer Authentication testing is experimental and incomplete at this stage. It is only enabled on the primary CPU on the cold boot.

Tests

- Some tests are implemented for AArch64 only and are skipped on AArch32.
- Some tests are not robust enough:
  - Some tests might hang in some circumstances. For example, they might wait forever for a condition to become true.
  - Some tests rely on arbitrary time delays instead of proper synchronization when executing order-sensitive steps.
  - Some tests have been implemented in a practical manner: they seem to work on actual hardware but they make assumptions that are not guaranteed by the Arm architecture. Therefore, they might fail on some other platforms.
- PSCI stress tests are very unreliable and will often hang. The root cause is not known for sure but this might be due to bad synchronization between CPUs.
- The GICv2 spurious IRQ test sometimes fails with the following error message:

  ```
  SMC @ lead CPU returned 0xFFFFFFFF 0x8 0xC
  ```

  The root cause is unknown.
- The FWU tests take a long time to complete. This is because they wait for the watchdog to reset the system. On FVP, TF-A configures the watchdog period to about 4 min. This limit is excessive for an automated testing context and leaves the user without feedback and unable to determine if the tests are proceeding properly.
- The test "Target timer to a power down cpu" sometimes fails with the following error message:

  ```
  Expected timer switch:  4 Actual:  3
  ```

  The root cause is unknown.

FWU images

- The FWU tests do not work on the revC of the Base AEM FVP. They only work on the revB.
- NS-BL1U and NS-BL2U images reuse TFTF-specific code for legacy reasons. This is not a clean design and may cause confusion.

Test secure partitions (Cactus, Cactus-MM, Ivy)

- This is experimental code. It's likely to change a lot as the secure partition software architecture evolves.
- Supported on AArch64 FVP platform only.

All test images

- TF-A Tests are derived from a fork of TF-A so:
  - they've got some code in common but lag behind on some features.
  - there might still be some irrelevant references to TF-A.
- Some design issues. E.g. TF-A Tests inherited from the I/O layer of TF-A, which still needs a major rework.
- Cannot build TF-A Tests with Clang. Only GCC is supported.
- The build system does not cope well with parallel building. The user should not attempt to run multiple jobs in parallel with the `-j` option of *GNU make*.
- The build system does not properly track build options. A clean build must be performed every time a build option changes.
- UUIDs are not compliant to RFC 4122.
- No floating point support. The code is compiled with GCC flag `-mgeneral-regs-only`, which prevents the compiler from generating code that accesses floating point registers. This might limit some test scenarios.
- The documentation is too lightweight.
- Missing instruction barriers in some places before reading the system counter value. As a result, the CPU could speculatively read it and any delay loop calculations might be off (because based on stale values). We need to examine all such direct reads of the `CNTPCT_EL0` register and replace them with a call to `syscounter_read()` where appropriate.

5.3 Version 2.0

5.3.1 New features

This is the first public release of the Trusted Firmware-A Tests source code.

TFTF

- Provides a baremetal test framework to exercise TF-A features through its `SMC` interface.
- Integrates easily with TF-A: the TFTF binary is packaged in the FIP image as a `BL33` component.
- Standalone binary that runs on the target without human intervention (except for some specific tests that require a manual target reset).
- Designed for multi-core testing. The various sub-frameworks allow maximum parallelism in order to stress the firmware.
- Displays test results on the UART output. This may then be parsed by an external tool and integrated in a continuous integration system.
- Supports running in AArch64 (NS-EL2 or NS-EL1) and AArch32 states.
- Supports parsing a tests manifest (XML file) listing the tests to include in the binary.
- Detects most platform features at run time (e.g. topology, GIC version, ...).
- Provides a topology enumeration framework. Allows tests to easily go through affinity levels and power domain nodes.
- Provides an event framework to synchronize CPU operations in a multi-core context.

- Provides a timer framework. Relies on a single global timer to generate interrupts for all CPUs in the system. This allows tests to easily program interrupts on demand to use as a wake-up event source to come out of CPU suspend state for example.
- Provides a power-state enumeration framework. Abstracts the valid power states supported on the platform.
- Provides helper functions for power management operations (CPU hotplug, CPU suspend, system suspend, . . .) with proper saving of the hardware state.
- Supports rebooting the platform at the end of each test for greater independence between tests.
- Supports interrupting and resuming a test session. This relies on storing test results in non-volatile memory (e.g. flash).

FWU images

- Provides example code to exercise the Firmware Update feature of TF-A.
- Tests the robustness of the FWU state machine implemented in the TF-A by sending valid and invalid authentication, copy and image execution requests to the TF-A BL1 image.

EL3 test payload

- Tests the ability of TF-A to load an EL3 payload.

Cactus test secure partition

- Tests that TF-A has correctly setup the secure partition environment: it should be allowed to perform cache maintenance operations, access floating point registers, etc.
- Tests the ability of a secure partition to request changing data access permissions and instruction permissions of memory regions it owns.
- Tests the ability of a secure partition to handle StandaloneMM requests.

5.3.2 Known issues and limitations

The sections below lists the known issues and limitations of each test image provided in this repository.

TFTF

The TFTF test image might be conceptually sub-divided further in 2 parts: the tests themselves, and the test framework they are based upon.

Test framework

- Some stability issues.
- No mechanism to abort tests when they time out (e.g. this could be implemented using a watchdog).
- No convenient way to include or exclude tests on a per-platform basis.
- Power domains and affinity levels are considered equivalent but they may not necessarily be.

- Need to provide better support to alleviate duplication of test code. There are some recurrent test patterns for which helper functions should be provided. For example, bringing up all CPUs on the platform and executing the same function on all of them, or programming an interrupt and waiting for it to trigger.
- Every CPU that participates in a test must return from the test function. If it does not - e.g. because it powered itself off for testing purposes - then the test framework will wait forever for this CPU. This limitation is too restrictive for some tests.
- No protection against interrupted flash operations. If the target is reset while some data is written to flash, the test framework might behave incorrectly on reset.
- When compiling the code, if the generation of the tests_list.c and/or tests_list.h files fails, the build process is not aborted immediately and will only fail later on.
- The directory layout is confusing. Most of the test framework code has been moved under the `tftf/` directory to better isolate it but this effort is not complete. As a result, there are still some TFTF files scattered around.

Tests

- Some tests are implemented for AArch64 only and are skipped on AArch32.
- Some tests are not robust enough:
  - Some tests might hang in some circumstances. For example, they might wait forever for a condition to become true.
  - Some tests rely on arbitrary time delays instead of proper synchronization when executing order-sensitive steps.
  - Some tests have been implemented in a practical manner: they seem to work on actual hardware but they make assumptions that are not guaranteed by the Arm architecture. Therefore, they might fail on some other platforms.
- PSCI stress tests are very unreliable and will often hang. The root cause is not known for sure but this might be due to bad synchronization between CPUs.
- The GICv2 spurious IRQ test is Juno-specific. In reality, it should only be GICv2-specific. It should be reworked to remove any platform-specific assumption.
- The GICv2 spurious IRQ test sometimes fails with the following error message:

  ```
  SMC @ lead CPU returned 0xFFFFFFFF 0x8 0xC
  ```

  The root cause is unknown.
- The manual tests in AArch32 mode do not work properly. They save some state information into non-volatile memory in order to detect the reset reason but this state does not appear to be retained. As a result, these tests keep resetting infinitely.
- The FWU tests take a long time to complete. This is because they wait for the watchdog to reset the system. On FVP, TF-A configures the watchdog period to about 4 min. This is way too long in an automated testing context. Besides, the user gets not feedback, which may let them think that the tests are not working properly.
- The test "Target timer to a power down cpu" sometimes fails with the following error message:

  ```
  Expected timer switch:  4 Actual:  3
  ```

  The root cause is unknown.

FWU images

- The FWU tests do not work on the revC of the Base AEM FVP. They only work on the revB.
- NS-BL1U and NS-BL2U images reuse TFTF-specific code for legacy reasons. This is not a clean design and may cause confusion.

Cactus test secure partition

- Cactus is experimental code. It's likely to change a lot as the secure partition software architecture evolves.
- Fails to link with GNU toolchain 7.3.1.
- Cactus is supported on AArch64 FVP platform only.

All test images

- TF-A Tests are derived from a fork of TF-A so:
  - they've got some code in common but lag behind on some features.
  - there might still be some irrelevant references to TF-A.
- Some design issues. E.g. TF-A Tests inherited from the I/O layer of TF-A, which still needs a major rework.
- Cannot build TF-A Tests with Clang. Only GCC is supported.
- The build system does not cope well with parallel building. The user should not attempt to run multiple jobs in parallel with the `-j` option of *GNU make*.
- The build system does not properly track build options. A clean build must be performed every time a build option changes.
- SMCCC v2 is not properly supported.
- UUIDs are not compliant to RFC 4122.
- No floating point support. The code is compiled with GCC flag `-mgeneral-regs-only`, which prevents the compiler from generating code that accesses floating point registers. This might limit some test scenarios.
- The documentation is too lightweight.

CHAPTER

SIX

LICENSE

The software is provided under a BSD-3-Clause *license*. Contributions to this project are accepted under the same license with developer sign-off as described in the **'Contributing Guidelines'_**.

```
Copyright (c) <year> <owner>. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

-  Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

-  Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

-  Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from this
software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

6.1 SPDX Identifiers

Individual files contain the following tag instead of the full license text.

```
SPDX-License-Identifier:    BSD-3-Clause
```

This enables machine processing of license information based on the SPDX License Identifiers that are here available: http://spdx.org/licenses/

6.2 Other Projects

This project contains code from other projects as listed below. The original license text is included in those source files.

- The libc source code is derived from FreeBSD and SCC. FreeBSD uses various BSD licenses, including BSD-3-Clause and BSD-2-Clause. The SCC code is used under the BSD-3-Clause license with the author's permission.
- The LLVM compiler-rt source code is disjunctively dual licensed (NCSA OR MIT). It is used by this project under the terms of the NCSA license (also known as the University of Illinois/NCSA Open Source License), which is a permissive license compatible with BSD-3-Clause. Any contributions to this code must be made under the terms of both licenses.
