TF-M Build System Update

TF-M Open Tech Forum

June 11, 2020

Raef Coles, Anton Komlev

Modern CMake overview

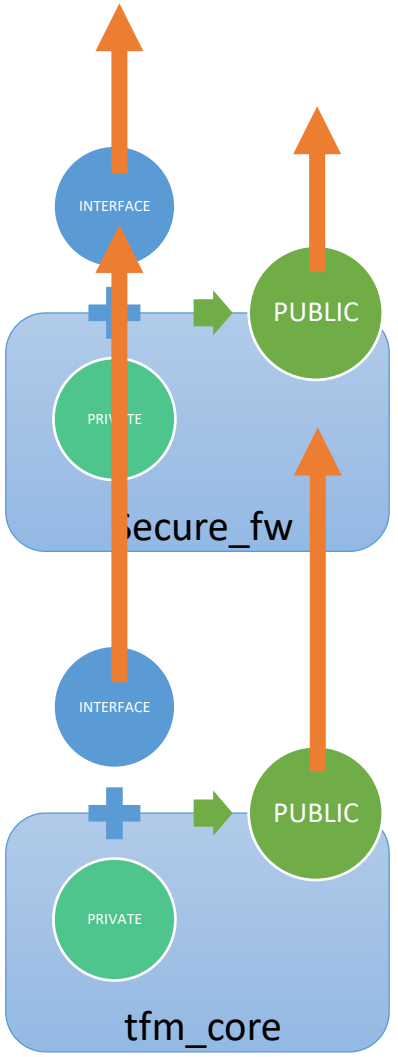
- Target oriented design
  - Parameters encapsulation vs Global variables
  - C / C++ analogy
- Public – private interfaces
  - Automatic properties propagation
- Supported widely but not widely used
- Helpful materials
  - https://www.youtube.com/watch?v=bsXLMQ6WgIk
  - https://www.youtube.com/watch?v=y7ndUhdQuU8
  - https://cliutils.gitlab.io/modern-cmake/
  - https://pabloariasal.github.io/2018/02/19/its-time-to-do-cmake-right/

Targets

- Basic unit of modern cmake
- Can have separate compile definitions, include paths etc.
- Executable / (static | shared | interface) Library
- Requires at least one source file
- Behave a bit like C++ classes in terms of isolation / inheritance

Public - Private - Interface

- Control the scope of target properties
- Compile definitions / include paths / compiler options
- Public
  - This target needs this, and all its dependents do as well
- Private
  - This target needs this, but it's dependents don't
- Interface
  - This target doesn't need this, but it's dependents do
- Propagates through dependency chains
- Private should be default

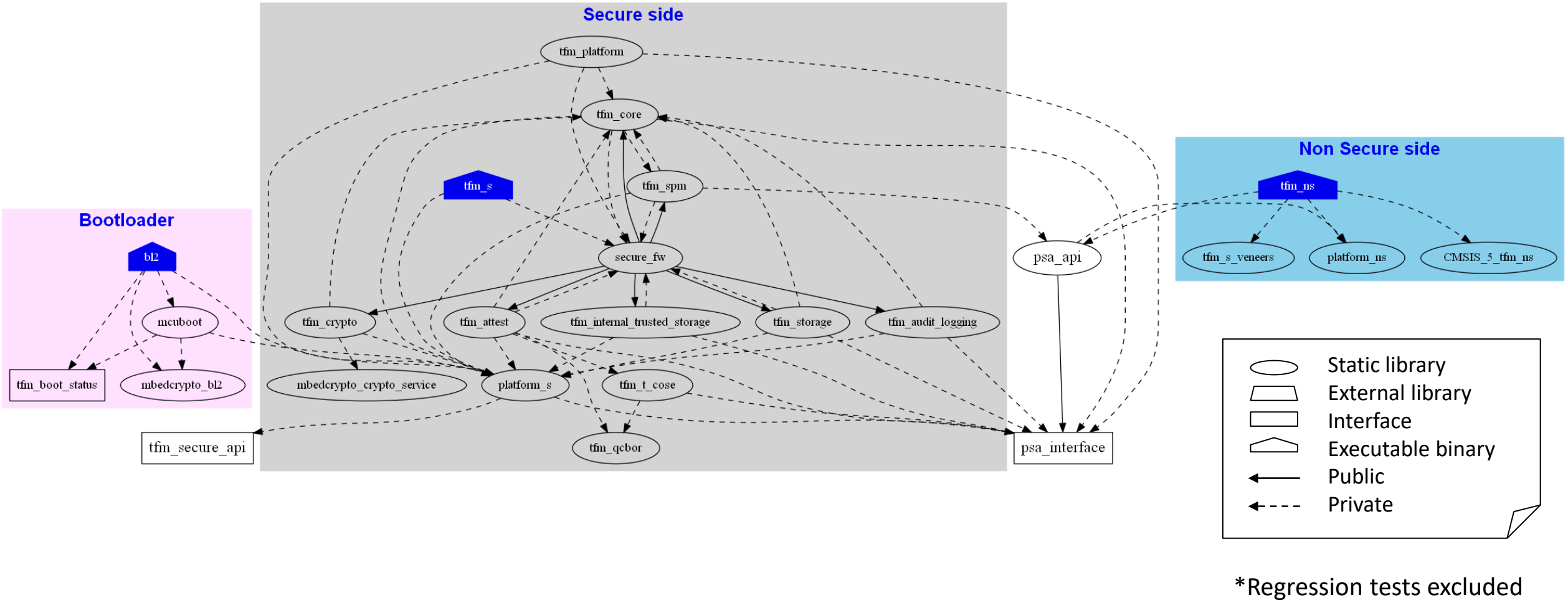


Inheritance example

- Set CMSE flag on in only platform_s
- Propagates to everything that links to platform_s:
  - All secure partitions
  - SPM
- Does not get set for things in the NS world or bootloader
- Single statement – all propagation automatic

```
target_compile_options(platform_s
    PUBLIC
        ${COMPILER_CMSE_FLAG}
)
```

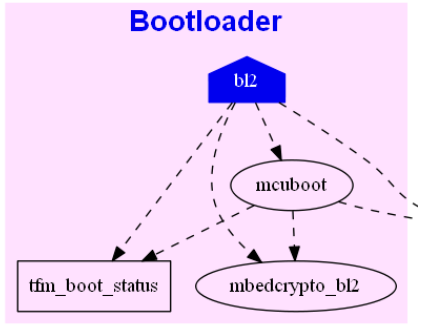
Dependencies (on June 5)

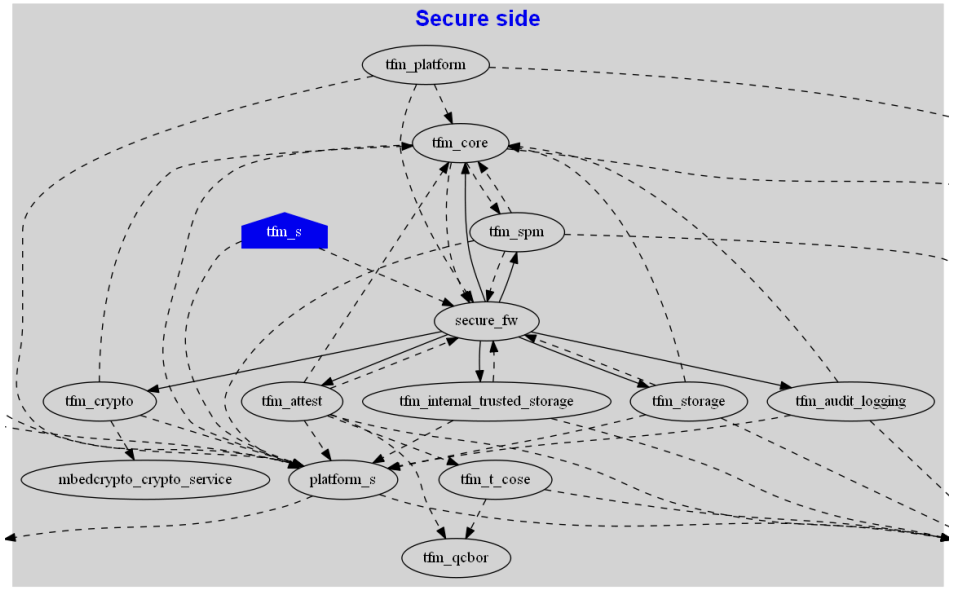


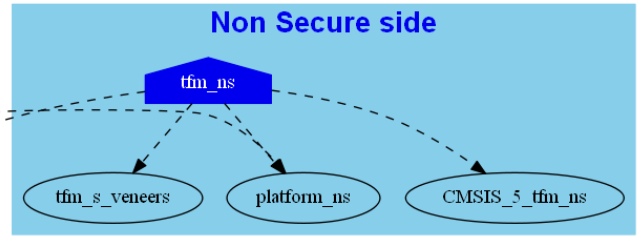
*Regression tests excluded

Important TF-M Targets

- Most targets built as static libraries
- Main targets:
  - secure_fw – The secure side / actual TFM code
  - bl2 – the level 2 bootloader
  - app – The nonsecure (example) app
- Other targets
  - TFM partitions
  - TFM veneers
  - PSA interface
  - Platform (Secure | non-secure | bl2)
  - External libraries







External libraries

- Automatically download the correct versions and build them
- Automatically patch if necessary
- Can be overridden by setting a cmake variable
  - -DMBEDCRYPTO_PATH=…

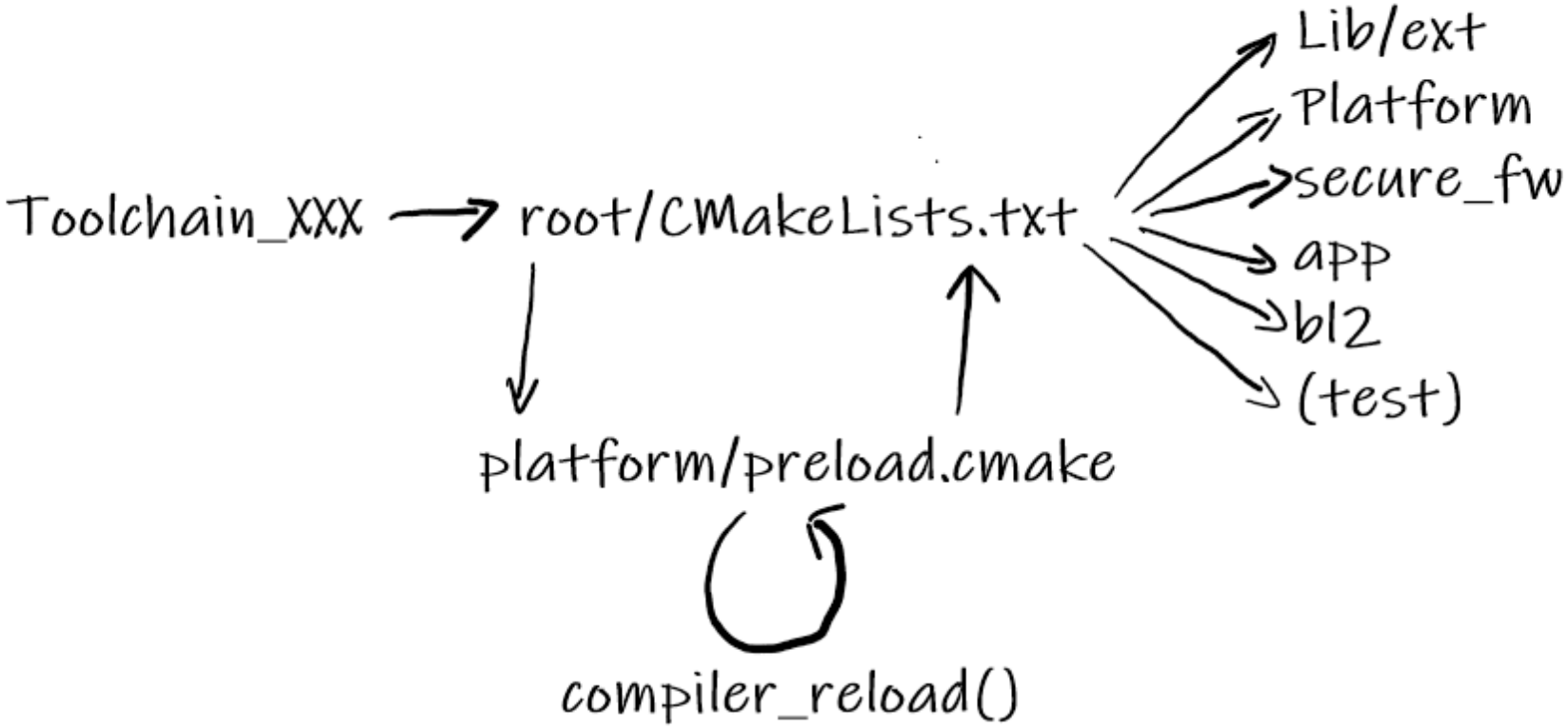
HW Platform support

- Specified by user externally (similar as before but not the same)
  - platform name -> `platform/ext/target/${TFM_PLATFORM}/`
  - Every platform provides two files:
    - CMakeLists.txt
    - preload.cmake
- Cmakelists.txt
  - Sets what files are built by the three targets platform_(s | ns | bl2)
  - Sets startup and scatter files
  - (Maybe) sets cmake setting flags
- Preload.cmake
  - Sets CPU and architecture
  - Sets details of any hardware accelerators

Compiler support / Toolchains

- Supported toolchains
  - GCC, CLANG, [IAR]
- Compiler support entirely in one toolchain file
  - Define variables and macros for things that are compiler specific
  - `${COMPILER_CMSE_FLAG}`
- Select compiler by selecting toolchain file
  - `-DCMAKE_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake`
- Currently compiler support is ~3 macros, 2 variables and default compiler / linker flags

Startup sequence



Benefits

- Currently removes 4000 lines of cmake, replacing it with 2800 lines of much maintainable code
- Removes most logic from platform cmake = easier platform porting
- Full incremental build support = much faster build times
- Should be able to plug TFM into existing cmake build systems
  - Link to targets like psa_api in one cmake statement

Visible changes

- `make install` replaced by just `make`
  - Will be disabled by time of release
- Armclang support requires cmake v15.0 +
  - Earliest version that supports armclang
  - Previously backported but that has been removed
  - Not in repos for Ubuntu
  - Takes ~5 mins to install from source
- Changes to cmake command-line parameters
  - `DIR -> PATH`

Example: Musca S1 platform_s

```
set_target_properties(platform_s PROPERTIES
    PLATFORM_SCATTER "${PLATFORM_DIR}/ext/common/armclang/tfm_common_s.sct"
    PLATFORM_STARTUP "${CMAKE_CURRENT_SOURCE_DIR}/Device/Source/armclang/startup_cmsdk_musca_s.s"
)
target_include_directories(platform_s
    PUBLIC
        .
        CMSIS_Driver
        CMSIS_Driver/Config
        Device/Config
        Device/Include
        Native_Driver
        partition
        services/include
)

target_sources(platform_s
    PRIVATE
        CMSIS_Driver/Driver_Flash_MRAM.c
        CMSIS_Driver/Driver_MPC.c
        CMSIS_Driver/Driver_PPC.c
        CMSIS_Driver/Driver_USART.c
        Device/Source/device_definition.c
        Device/Source/system_core_init.c
        Native_Driver/mpc_sie200_drv.c
        Native_Driver/mpu_armv8m_drv.c
        Native_Driver/ppc_sse200_drv.c
        spm_hal.c
        target_cfg.c
        $<$<BOOL:BUILD_NATIVE_DRIVERS>:${CMAKE_CURRENT_SOURCE_DIR}/Native_Driver/ppc_sse200_drv.c>
        $<$<BOOL:BUILD_NATIVE_DRIVERS>:${CMAKE_CURRENT_SOURCE_DIR}/Native_Driver/uart_pl011_drv.c>
        $<$<BOOL:BUILD_PLAT_TEST>:${CMAKE_CURRENT_SOURCE_DIR}/plat_test.c>
        $<$<BOOL:BUILD_TIME>:${CMAKE_CURRENT_SOURCE_DIR}/Native_Driver/timer_cmsdk_drv.c>
        $<$<BOOL:TFM_PARTITION_PLATFORM>:${CMAKE_CURRENT_SOURCE_DIR}/services/src/tfm_platform_system.c>
)
```

TODO

- Move Non-Secure app into different repo
- Port and minimise all configuration options
- IAR compiler support
- Port tests
- Port remaining platforms

The End

Thank you