TF-A CMake build system

Javier Almansa Sobrino

7 May 2020

Table of contents

Motivation

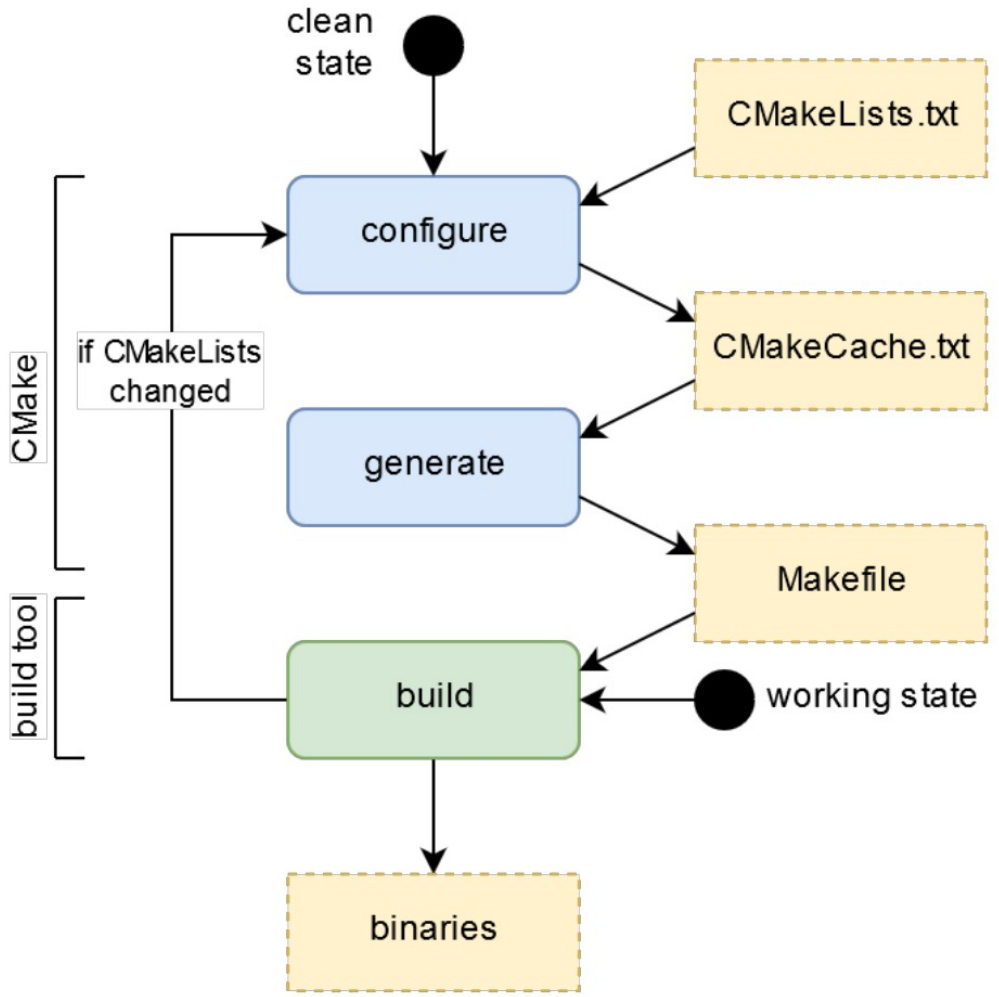
- Current build system based on GNU Make
- As the project grows, the current build system is getting hard to scale
  - Large amount of options and dependencies
  - It makes difficult not to break some parts of the system when adding support to others
  - The current build system is unable to detect changes on the configuration: the workspace needs to be cleaned in the case we need to rebuild with different options
- CMake has been successfully used on several other projects at ARM
  - More scalable
  - Able to handle dependencies easier
  - Detects changes on the configuration
  - More portable
  - Richer feature set compared to the current build system

Introduction to CMake

- CMake is a tool to describe and generate buildsystems.
- Describes a project in the CMake language
  - OS, compiler and target independent
- CMake generates a buildsystem using a generator
  - Many generators available (Makefile, Ninja, VS, etc)
- Cons
  - CMake language

CMake workflow

- Files
  - CMakeLists
    - Project description in CMake language
  - CMakeCache
    - Text file with cached CMake variables
    - Persistent across multiple runs
- Steps
  - Configuration
    - Build cache based on CmakeLists
    - CMake scripts are parsed/run
    - Create native build tool files
  - Build
    - The actual build tool is ran and the compiler and other tools get invoked



arm

CMake integration into TF-A

- Solutions to CMake cons
  - CMake framework
    - Hosted on its own repository on TF-A
    - Shared portion of CMake scripts
    - Project independent
  - Built-definitions
    - Project specific CMake scripts
    - Merged into TF-A
    - Rely on functions and macros implemented in the CMake framework

Two phase approach

1. Without code refactor (now)
   - No source code modification
   - Project structure and modularization untouched
   - Buildsystem logic similar to Makefile
   - CMake language with not all features used.
2. Code refactoring (future)
   - Refactor TF-A source code where necessary
   - Better modularization, clear APIs/dependencies
   - Separate include paths
   - Use all CMake features, such as transitive dependency propagation

Needed features for the framework

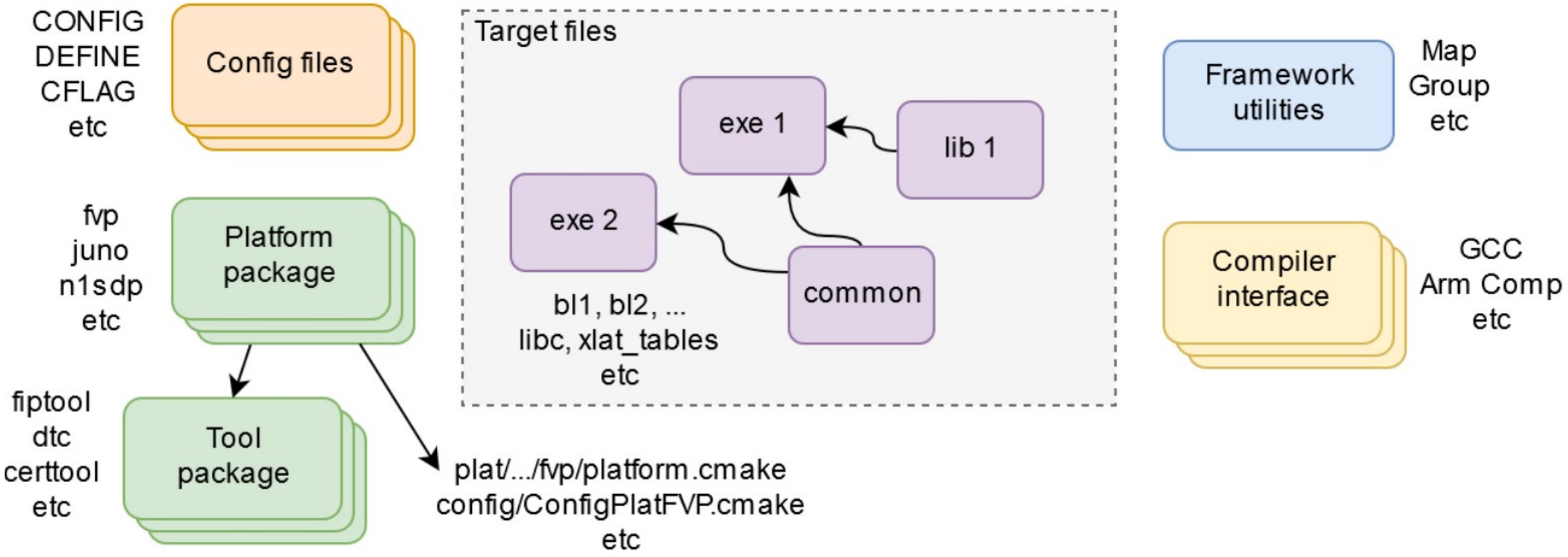
Features

- Structured configuration description
  - Build options
  - Defines, flags, etc.
- Target description
  - What are we building
  - Source files, linked libraries
  - Liker script, etc.
- Compiler abstraction
- External tools

Solutions

- Utilities
  - Map: Key-value pairs
  - Groups: Collection of maps
  - Config files
- STGT API
  - Wrap CMake functions
  - Use setting groups
- Compiler_functions for common tasks
  - Preprocess, set linker script, etc.
- find_package modules
- For fiptool, dtc, etc.

Framework overview



arm

Config example

Groups

- Groups allow to define sets of related flags, build options or definitions.

```
 8  group_new(NAME compiler)
 9
10  if(CMAKE_BUILD_TYPE STREQUAL Debug)
11          group_add(NAME compiler TYPE CFLAG KEY -g)
12          group_add(NAME compiler TYPE ASFLAG KEY -g)
13          group_add(NAME compiler TYPE ASFLAG KEY -Wa,--gdwarf-2)
14  endif()
15
16  group_add(NAME compiler TYPE CFLAG KEY -Os)
17  group_add(NAME compiler TYPE CFLAG KEY -march VAL armv8-a)
18  group_add(NAME compiler TYPE CFLAG KEY -mgeneral-regs-only)
19  group_add(NAME compiler TYPE CFLAG KEY -mstrict-align)
```

```
19  # Use the GICv3 driver on the FVP by default
20  group_add(NAME hw_plat TYPE CONFIG DEFINE KEY FVP_USE_GIC_DRIVER VAL FVP_GICV3)
21
22  # Use the SP804 timer instead of the generic one
23  group_add(NAME hw_plat TYPE CONFIG DEFINE KEY FVP_USE_SP804_TIMER VAL 0)
24
25  # Default cluster count for FVP
26  group_add(NAME hw_plat TYPE CONFIG DEFINE KEY FVP_CLUSTER_COUNT VAL 2)
27
28  # Default number of CPUs per cluster on FVP
29  group_add(NAME hw_plat TYPE DEFINE KEY FVP_MAX_CPUS_PER_CLUSTER VAL 4)
30
31  # Default number of threads per CPU on FVP
32  group_add(NAME hw_plat TYPE DEFINE KEY FVP_MAX_PE_PER_CPU VAL 1)
```

Config example

Targets

- Groups all the artifacts needed to build a binary:
  - Src files
  - Includes
  - Libraries
- Allows for conditional inclusion of srcs

```
12  stgt_create(NAME bl31)
13  stgt_add_setting(NAME bl31 GROUPS default compiler hw_plat bl31_specific)
14  stgt_set_target(NAME bl31 TYPE exe)
15
16  stgt_add_src(NAME bl31 SRC
17          ${CMAKE_CURRENT_LIST_DIR}/bl31_main.c
18          ${CMAKE_CURRENT_LIST_DIR}/interrupt_mgmt.c
19          ${CMAKE_CURRENT_LIST_DIR}/aarch64/bl31_entrypoint.S
20          ${CMAKE_CURRENT_LIST_DIR}/aarch64/crash_reporting.S
21          ${CMAKE_CURRENT_LIST_DIR}/aarch64/ea_delegate.S
22          ${CMAKE_CURRENT_LIST_DIR}/aarch64/runtime_exceptions.S
23          ${CMAKE_CURRENT_LIST_DIR}/bl31_context_mgmt.c
24  )
```

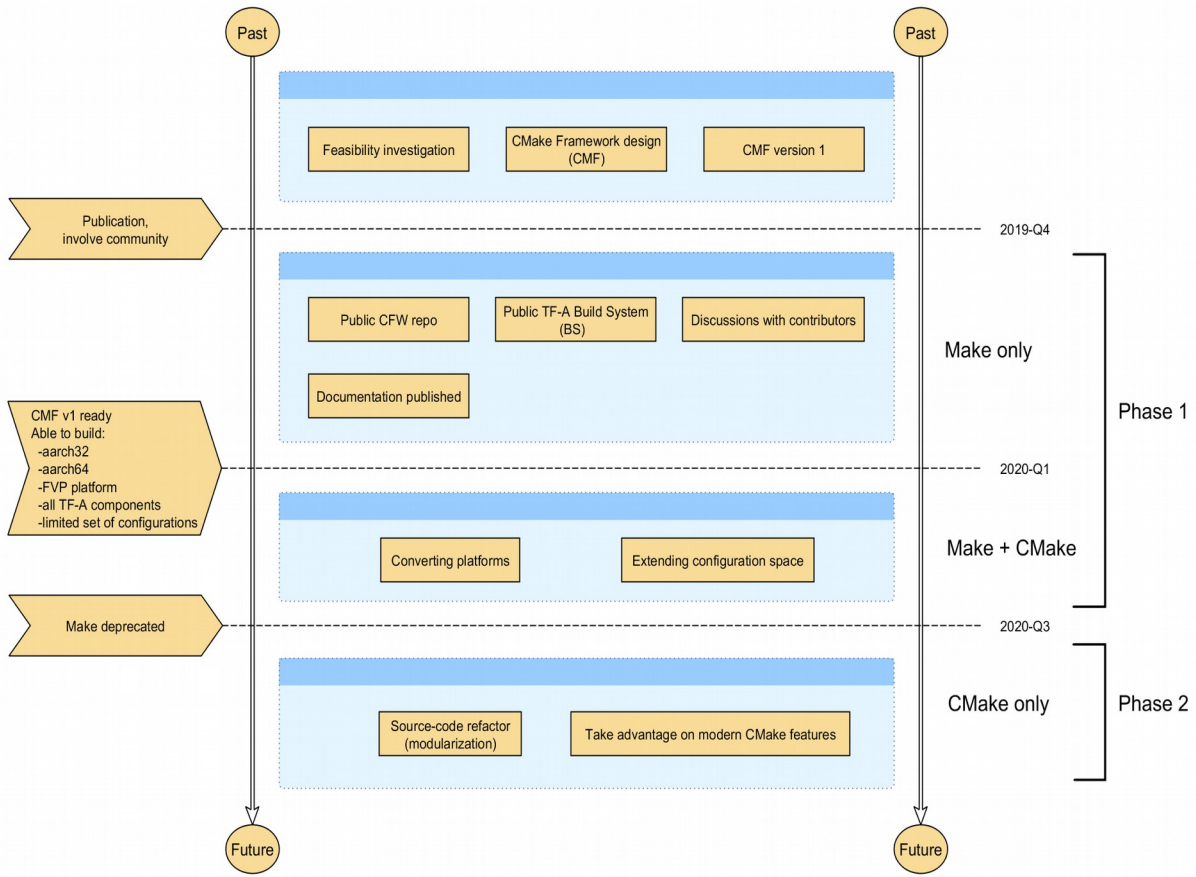
```
62  stgt_link_libraries(NAME bl31 LIBS libc xlat_tables libfdt)
63  stgt_link_build_messages(NAME bl31 LIBS build_message)
64
65  get_target_property(_defs bl31 COMPILE_DEFINITIONS)
66  get_target_property(_inc bl31 INCLUDE_DIRECTORIES)
```

Current status

- Current framework is work in progress.
  - More features will need to be added as migration to the new build system progresses
- Basic support for FVP is available – Internal, WIP
  - Some libraries with basic support

Future plan/Roadmap

- CMake build system is in a very early stage. Still a lot of work to do:
  - Finish FVP port
  - Add support for missing components and configurations
  - sp_min, support for 32bit build, etc.
  - Extend the framework as needed
  - Support for armclang and for KConfig, among others
  - Prepare porting over platforms
  - TF-A CI integration
  - Documentation
  - Make and CMake coexistence
  - Both would have to coexist for a long period of time
  - Make deprecation



The current plan is still under development and the deadline for all the milestones are TBD. Their order of implementation may vary with regards to the one exposed here.

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה