XCORE RTOS Framework - Build System Guide

Release: 3.0.1

Publication Date: 2023/05/02

Table of Contents

1 Build System

This document describes the CMake-based build system used by applications based on the XMOS RTOS framework. The build system is designed so a user does not have to be an expert using CMake. However, some familiarity with CMake is helpful. You can familiarize yourself by reading the CMake Tutorial or CMake documentation. Reviewing these is optional and the reader should feel free to save that for later.

1.1 Overview

An xcore RTOS project can be seen as an integration of several modules. For example, for a FreeRTOS application that captures audio from PDM microphones and outputs it to a DAC, there could be the following modules:

- Several core modules (for debug prints, etc...)
- The FreeRTOS kernel and drivers
- PDM microphone array driver for receiving audio samples
- I^2C driver for configuring the DAC
- I^2S driver for outputting to the DAC
- Application code tying it all together

When a project is compiled, the build system will build all libraries and source files required for the application. For this to happen, your `CMakeLists.txt` file will need to specify:

- Application sources and include paths
- Compile flags
- Compile definitions
- Link libraries
- Link options

This is best illustrated with a commented *Example CMakeLists.txt*.

1.2 Aliases

Your *CMakeLists.txt* file will need to specify the target link libraries as shown in the following snippet:

```
target_link_libraries(my_target PUBLIC
    core::general
    rtos::freertos
    rtos::drivers::mic_array
    rtos::drivers::i2c
    rtos::drivers::i2s
    lib_mic_array
    lib_i2c
    lib_i2s
)
```

It is very common for target link alias libraries, like `rtos::freertos` in the snippet above, to include common sets of target link libraries. The snippet above could be simplified because the *rtos::freertos* alias includes many commonly used drivers and peripheral IO libraries as a dependency.

```
target_link_libraries(my_target PUBLIC
    core::general
    rtos::freertos
)
```

Application target link libraries can be further simplified using existing bsp_configs. These provide their dependent link libraries enabling applications to simplify their target link libraries list. The snippet above could be simplified because the *rtos::bsp_config::xcore_ai_explorer* alias includes *core::general*, *rtos::freertos*, and all required drivers and peripheral IO libraries used by the bsp_config. More information on bsp_configs can be found in the RTOS Programming Guide.

```
target_link_libraries(my_target PUBLIC
    rtos::bsp_config::xcore_ai_explorer
)
```

XMOS libraries and frameworks provide several target aliases. Being aware of the *Targets* will simplify your application `CMakeLists.txt`.

2 Example CMakeLists.txt

CMake is powerful tool that provides the developer a great deal of flexibility in how their projects are built. As a result, `CMakeLists.txt` files can accomplish the same function in multiple ways.

Below is an example `CMakeLists.txt` that shows both required and conventional commands for a basic FreeRTOS project. This example can be used as a starting point for your application, but it is recommended to copy a `CMakeLists.txt` from an XMOS reference design or other example application that closely resembles your application.

```
## Specify your application sources by globbing the src folder
file(GLOB_RECURSE APP_SOURCES src/*.c)

## Specify your application include paths
set(APP_INCLUDES src)

## Specify your compiler flags
set(APP_COMPILER_FLAGS
   -Os
   -report
   -fxscope
   -mcmodel=large
   ${CMAKE_CURRENT_SOURCE_DIR}/src/config.xscope
   ${CMAKE_CURRENT_SOURCE_DIR}/XCORE-AI-EXPLORER.xn
)

## Specify any compile definitions
set(APP_COMPILE_DEFINITIONS
   configENABLE_DEBUG_PRINTF=1
   PLATFORM_USES_TILE_0=1
   PLATFORM_USES_TILE_1=1
)

## Set your link libraries
set(APP_LINK_LIBRARIES
   rtos::bsp_config::xcore_ai_explorer
)

## Set your link options
set(APP_LINK_OPTIONS
   -report
   ${CMAKE_CURRENT_SOURCE_DIR}/XCORE-AI-EXPLORER.xn
   ${CMAKE_CURRENT_SOURCE_DIR}/src/config.xscope
)

## Create your targets

## Create the target for the portion of application code that will execute on tile[0]
set(TARGET_NAME tile0_my_app)
add_executable(${TARGET_NAME} EXCLUDE_FROM_ALL)
```

(continues on next page)

(continued from previous page)

```
target_sources(${TARGET_NAME} PUBLIC ${APP_SOURCES})
target_include_directories(${TARGET_NAME} PUBLIC ${APP_INCLUDES})
target_compile_definitions(${TARGET_NAME} PUBLIC ${APP_COMPILE_DEFINITIONS} THIS_XCORE_
↪TILE=0)
target_compile_options(${TARGET_NAME} PRIVATE ${APP_COMPILER_FLAGS})
target_link_libraries(${TARGET_NAME} PUBLIC ${APP_LINK_LIBRARIES})
target_link_options(${TARGET_NAME} PRIVATE ${APP_LINK_OPTIONS})
unset(TARGET_NAME)

## Create the target for the portion of application code that will execute on tile[1]
set(TARGET_NAME tile1_my_app)
add_executable(${TARGET_NAME} EXCLUDE_FROM_ALL)
target_sources(${TARGET_NAME} PUBLIC ${APP_SOURCES})
target_include_directories(${TARGET_NAME} PUBLIC ${APP_INCLUDES})
target_compile_definitions(${TARGET_NAME} PUBLIC ${APP_COMPILE_DEFINITIONS} THIS_XCORE_
↪TILE=1)
target_compile_options(${TARGET_NAME} PRIVATE ${APP_COMPILER_FLAGS})
target_link_libraries(${TARGET_NAME} PUBLIC ${APP_LINK_LIBRARIES})
target_link_libraries(${TARGET_NAME} PRIVATE ${APP_LINK_OPTIONS} )
unset(TARGET_NAME)

## Merge tile[0] and tile[1] binaries into a single binary using an XMOS CMake macro
merge_binaries(my_app tile0_my_app tile1_my_app 1)

## Optionally create run and debug targets using XMOS CMake macros
create_run_target(my_app)
create_debug_target(my_app)
```

For more information, see the documentation for each of the CMake commands used in the example above.

- set
- add_executable
- target_sources
- target_include_directories
- target_compile_definitions
- target_compile_options
- target_link_libraries
- target_link_options

See *Macros* for more information on the XMOS CMake macros.

3 Targets

The following library target aliases can be used in your application *CMakeLists.txt*. An example of how to add aliases to your target link libraries is shown below:

```
target_link_libraries(my_app PUBLIC core::general rtos::freertos)
```

3.1 General

Several aliases are provided that specify a collection of libraries with similar functions. These composite target libraries provide a concise alternative to specifying all the individual targets that are commonly required.

Table 3.1: Composite Target Libraries

Target	Description
core::general	Commonly used core libraries
io::general	Commonly used peripheral libraries
io::audio	Commonly used peripheral libraries for audio applications
rtos::freertos	Commonly used RTOS libraries

3.2 Core

If you prefer, you can specify individual core library targets.

Table 3.2: Core Libraries

Target	Description
framework_core_clock_control	Clock control API
framework_core_utils	General utilities used by most applications
framework_core_legacy_compat	For compatibility with XC
lib_xcore_math	VPU-optimized math library

3.3 Peripherals

If you prefer, you can specify individual peripheral libraries.

Table 3.3: Peripheral Libraries

Target	Description
lib_i2c	I^2C library
lib_spi	SPI library
lib_uart	UART library
lib_qspi_io	QSPI library
lib_xud	XUD USB library
lib_i2s	I^2S library
lib_mic_array	Microphone Array library

3.4 RTOS

Several aliases are provided that specify a collection of RTOS libraries with similar functions. These composite target libraries provide a concise alternative to specifying all the individual targets that are commonly required.

Table 3.4: Composite RTOS Libraries

Target	Description
rtos::freertos	All libraries used my most FreeRTOS applications
rtos::drivers:all	All RTOS Driver libraries
rtos::freertos_usb	All libraries to support development with TinyUSB
rtos::sw_services::general	Most commonly used RTOS software service libraries
rtos::iot	All IoT libraries
rtos::wifi	All WiFi libraries

These board support libraries simplify development with a specific board.

Table 3.5: Board Support Libraries

Target	Description
rtos::bsp_config::xcore_ai_explorer	xcore.ai Explorer RTOS board support library

If you prefer, you can specify individual RTOS driver libraries.

Table 3.6: Individual RTOS Driver Libraries

Target	Description
rtos::drivers::uart	UART RTOS driver library
rtos::drivers::i2c	I^2C RTOS driver library
rtos::drivers::i2s	I^2S RTOS driver library
rtos::drivers::spi	SPI RTOS driver library
rtos::drivers::qspi_io	QSPI RTOS driver library
rtos::drivers::mic_array	Microphone Array RTOS driver library
rtos::drivers::usb	USB RTOS driver library
rtos::drivers::dfu_image	RTOS DFU driver library
rtos::drivers::gpio	GPIO RTOS driver library
rtos::drivers::l2_cache	L2 Cache RTOS driver library
rtos::drivers::clock_control	Clock control RTOS driver library
rtos::drivers::trace	Trace RTOS driver library
rtos::drivers::swmem	SwMem RTOS driver library
rtos::drivers::wifi	WiFi RTOS driver library
rtos::drivers::intertile	Intertile RTOS driver library
rtos::drivers::rpc	Remote procedure call RTOS driver library

If you prefer, you can specify individual software service libraries.

Table 3.7: Individual Software Service Libraries

Target	Description
rtos::sw_services::fatfs	FatFS library
rtos::sw_services::usb	USB library
rtos::sw_services::device_control	Device control library
rtos::sw_services::usb_device_control	USB device control library
rtos::sw_services::wifi_manager	WiFi manager library
rtos::sw_services::tls_support	TLS library
rtos::sw_services::dhcp	DHCP library
rtos::sw_services::json	JSON library
rtos::sw_services::http	HTTP library
rtos::sw_services::sntpd	SNTP daemon library
rtos::sw_services::mqtt	MQTT library

The following libraries for building host applications are also provided by the SDK.

Table 3.8: Host (x86) Libraries

Target	Description
rtos::sw_services::device_control_host_usb	Host USB device control library

4 Macros

Several CMake macros and functions are provide to make building for XCORE easier. These macros are located in the file tools/cmake_utils/xmos_macros.cmake and are documented below.

To see what XTC Tools commands the macros and functions are running, add `VERBOSE=1` to your build command line. For example:

```
make run_my_target VERBOSE=1
```

4.1 Common Macros

4.1.1 merge_binaries

merge_binaries combines multiple xcore applications into one by extracting a tile elf and recombining it into another binary. This is used in multitile RTOS applications to enable building unique instances of the FreeRTOS kernel and task sets on a per tile basis. This macro takes an output target name, a base target, a target containing a tile to merge, and the tile number to merge. The resulting output will be a target named `<OUTPUT_TARGET_NAME>`, which contains the `<BASE_TARGET>` application with tile `<TILE_NUM_TO_MERGE>` replaced with the respective tile from `<OTHER_TARGET>`.

```
merge_binaries(<OUTPUT_TARGET_NAME> <BASE_TARGET> <OTHER_TARGET> <TILE_NUM_TO_MERGE>)
```

4.1.2 create_run_target

create_run_target creates a run target for `<TARGET_NAME>` with xscope output.

```
create_run_target(<TARGET_NAME>)
```

create_run_target allows you to run a binary with the following command instead of invoking `xrun --xscope`.

```
make run_my_target
```

4.1.3 create_debug_target

create_debug_target creates a debug target for `<TARGET_NAME>`.

```
create_debug_target(<TARGET_NAME>)
```

create_debug_target allows you to debug a binary with the following command instead of invoking `xgdb`. This target implicitly sets up the xscope debug interface as well.

```
make debug_my_target
```

4.1.4 create_filesystem_target

create_filesystem_target creates a filesystem file for <TARGET_NAME> using the files in the <FILESYSTEM_INPUT_DIR> directory. <IMAGE_SIZE> specifies the size (in bytes) of the filesystem. The filesystem output filename will end in _fat.fs. Optional argument <OPTIONAL_DEPENDS_TARGETS> can be used to specify other dependency targets, such as filesystem generators.

```
create_filesystem_target(<TARGET_NAME> <FILESYSTEM_INPUT_DIR> <IMAGE_SIZE> <OPTIONAL_
↪DEPENDS_TARGETS>)
```

4.1.5 create_data_partition_directory

create_data_partition_directory creates a directory populated with all components related to the data partition. The data partition output folder will end in _data_partition Optional argument <OPTIONAL_DEPENDS_TARGETS> can be used to specify other dependency targets.

```
create_data_partition_directory(<TARGET_NAME> <FILES_TO_COPY> <OPTIONAL_DEPENDS_TARGETS>)
```

4.1.6 create_flash_app_target

create_flash_app_target creates a debug target for <TARGET_NAME> with optional arguments <BOOT_PARTITION_SIZE>, <DATA_PARTITION_CONTENTS>, and <OPTIONAL_DEPENDS_TARGETS>. <BOOT_PARTITION_SIZE> specificies the size in bytes of the boot partition. <DATA_PARTITION_CONTENTS> specifies the optional binary contents of the data partition. <OPTIONAL_DEPENDS_TARGETS> specifies CMake targets that should be dependencies of the resulting create_flash_app_target target. This may be used to create recipes that generate the data partition contents.

```
create_flash_app_target(<TARGET_NAME> <BOOT_PARTITION_SIZE> <DATA_PARTITION_CONTENTS>
↪<OPTIONAL_DEPENDS_TARGETS>)
```

create_flash_app_target allows you to flash a factory image binary and optional data partition with the following command instead of invoking `xflash`.

```
make flash_app_my_target
```

4.2 Less Common Macros

4.2.1 create_install_target

create_install_target creates an install target for <TARGET_NAME>.

```
create_install_target(<TARGET_NAME>)
```

create_install_target will copy <TARGET_NAME>.xe to the ${PROJECT_SOURCE_DIR}/dist directory.

```
make install_my_target
```

4.2.2 create_run_xscope_to_file_target

create_run_xscope_to_file_target creates a run target for `<TARGET_NAME>`. `<XSCOPE_FILE>` specifies the file to save to (no extension).

```
create_run_xscope_to_file_target(<TARGET_NAME> <XSCOPE_FILE>)
```

create_run_xscope_to_file_target allows you to run a binary with the following command instead of invoking `xrun --xscope-file`.

```
make run_xscope_to_file_my_target
```

4.2.3 create_upgrade_img_target

create_upgrade_img_target creates an xflash image upgrade target for a provided binary for use in DFU

```
create_data_partition_directory(<TARGET_NAME> <FACTORY_MAJOR_VER> <FACTORY_MINOR_VER>)
```

4.2.4 create_erase_all_target

create_erase_all_target creates an xflash erase all target for `<TARGET_FILEPATH>` target XN file. The full filepath must be specified for XN file

```
create_filesystem_target(<TARGET_NAME> <TARGET_FILEPATH>)
```

create_erase_all_target allows you to erase flash with the following command instead of invoking `xflash`.

```
make erase_all_my_target
```

4.2.5 query_tools_version

query_tools_version populates the following CMake variables:

`XTC_VERSION_MAJOR` `XTC_VERSION_MINOR` `XTC_VERSION_PATCH`

```
query_tools_version()
```

5 Copyright & Disclaimer

6 Licenses

6.1 XMOS

All original source code is licensed under the XMOS License.

6.2 Third-Party

Additional third party code is included under the following copyrights and licenses:

Table 6.1: Third Party Module Copyrights & Licenses

Module	Copyright & License
Argtable3	Copyright (C) 1998-2001,2003-2011,2013 Stewart Heitmann, licensed under LICENSE
FatFS	Copyright (C) 2017 ChaN, licensed under a BSD-style license
FreeRTOS	Copyright (c) 2017 Amazon.com, Inc., licensed under the MIT License
HTTP Parser	Copyright (c) Joyent, Inc. and other Node contributors, licensed under the MIT license
JSMN JSON Parser	Copyright (c) 2010 Serge A. Zaitsev, licensed under the MIT license
Mbed TLS library	Copyright (c) 2006-2018 ARM Limited, licensed under the Apache License 2.0
Paho MQTT C/C++ client for Embedded platforms	Copyright (c) 2020 The TensorFlow Authors, licensed under the Apache License
TinyUSB	Copyright (c) 2018 hathach (tinyusb.org), licensed under the MIT license