CMake FrameWork
User Manual

Document Revision Details

sl no	Change Details	Document Revisions
1	Initial Documentation of CMake Framework	rtss_cmake_scripts_v0.1
2	Cmake support for GNSS-M55 core	rtss_cmake_scripts_v0.2
3	Contents of scripts directory moved to base directory	rtss_cmake_scripts_v0.3
4	Additional Compiler Arguments support	rtss_cmake_scripts_v0.4
5	GCC compiler Support	rtss_cmake_scripts_v0.5
6	Support for Windows Host machine	rtss_cmake_scripts_v0.6
7	Multi test applications build support	rtss_cmake_scripts_v0.7
8	Optimizing running steps and feature enhancement	rtss_cmake_scripts_v0.8
9	Update running steps	rtss_cmake_scripts_v0.9
10	Adding support to build from pack	rtss_cmake_scripts_v1.0
11	Device Selection support added	rtss_cmake_scripts_v1.1
12	Help Option is added	rtss_cmake_scripts_v1.2
13	Revamped configuration of CMake using preset	rtss_cmake_scripts_v2.0

Table of Contents

Contents

rtss_cmake_scripts

This document contains detailed information on how to use the CMake scripts and helper shell scripts, to clone git repos or pack setup, compile test applications and generate binary files with respect to **Cortex-M55**.

System Requirements to run the Scripts

Ubuntu

1. CMake version on the Linux Host system should be having the latest version **3.25** and above**.**
2. Ubuntu OS version on the Host system should be at least **18.4 and above.**

Steps to update the CMake version on the Host

Run the below commands on the terminal to update the Cmake version to the latest version.

1. *sudo apt purge --auto-remove cmake*
2. *wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc 2>/dev/null | gpg --dearmor - | sudo tee /etc/apt/trusted.gpg.d/kitware.gpg >/dev/null*

For Ubuntu 18.04

1. *sudo apt-add-repository 'deb https://apt.kitware.com/ubuntu/ bionic main'*
2. sudo apt update
3. sudo apt install cmake

Windows

1. CMake version on the Windows Host system should be having the latest version **3.25** and above**.**
2. MingW64 should be installed, which provides the application named "GitBash".
3. Git should be installed.
4. Git bash application should be used as Terminal console while using this CMake Framework.

Contents of the ‘cmake folder

1. **rtss** → This directory contains all the CMake files, related to tool-chain and projects source files with respect to rtss.
2. **Include_RTE_Comp** → This directory contains "RTE_Components.h" header file specifically for Ensemble pack.
3. **Cmake Presets and Shell Scripts** → Cmake Presets will help to set the configuration/build/PresetName using cmake. Run.sh is wrapper script over presets which will run all available preset.

rtss directory

1. **device_cmake** → This contains cmake file which has the details related to devices bolt_extsys_driver or extsys0 or extsys1 for HP and HE.
2. **drivers_cmake** → This contains cmake file which has the details of source files and header files of all the supporting drivers for cortex_m55.
3. **ensemble_cmake** → This contains cmake file which has the details of source files and header files with respect to Ensemble package.
4. **os_cmake** → This contains cmake files which have the details of source files and header files to include with respect to different Operating Systems. It also has details of test applications to be built w.r.t each OS.
5. **toolchains** → This contains cmake file which has the compiler toolchain related details with respect to Cortex-M55 and default configurable compiler argument variables for *Cortex-M55*
6. **CMakeLists.txt** → This is the main cmake file, which links all the above cmake files based on the configuration and requirement.
7. **utilities_func.cmake** → This cmake file has helper functions and macro definitions required for the build.

Include_RTE_Comp

1. **ensemble** → This directory consists of **RTE_Components.h** header file specifically for **Ensemble pack**.

Preset files and Shell Script

1. **CMakePresets.json** → This Json basically setups the environment variables and other configuration parameters for user. User can have multiple entry inside *configurePresets, buildPresets/ testPresets/workflowPresets.*In workflow user can have their own custom flow.

   Note: Users need to add proper paths to set environments variable in global preset environment setting.

```
    {
        "name": "global-settings",
        "description": "Common settings shared by all configure presets",
        "hidden": true,
        "environment": {
            "COMPILER_BIN_PATH": "/opt/arm/developmentstudio-2022.0/sw/ARMCompiler6.18/bin",
            "ARM_PRODUCT_DEF": "/opt/arm/developmentstudio-2022.0/sw/mappings/gold.elmap",
            "ARMLMD_LICENSE_FILE": "27000@10.10.2.99",
            "CMSIS_PACK_PATH": "/mnt/c/Users/RajRanjan/AppData/Local/Arm/Packs/ARM/CMSIS/5.9.0",
            "CMSIS_COMPILER_PATH": "/mnt/c/Users/RajRanjan/AppData/Local/Arm/Packs/ARM/CMSIS-Compiler/2.1.0"
        }
    }
```

2. **run.sh** → This script is wrapper shell script over CmakePresets. By running this all combination of available/added configPreset and buildPreset can be run.

Steps to run the Preset and Shell Script.

Compilation without ./run.sh

- To check available configuration, we can run command "**cmake --list-presets" or "cmake --list-presets=configure" or "cmake --list-presets=configure"**

```
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$ cmake --list-presets
Available configure presets:

  "armclang" - Armclang Build Config
  "gcc"      - GCC Build Config
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$
```

- To check available build configuration, we can run command "**cmake --build --list-presets" or "cmake --list-presets=build"**

```
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$ cmake --build --list-presets
Available build presets:

  "armclang_build"
  "gcc_build"
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$
```

- To check available test configuration, we can run command "**ctest --list-presets" or "cmake --list-presets=test"**

```
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$ ctest --list-presets
Available test presets:

  "armclang_build_test"
  "gcc_build_test"
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$
```

- “*cmake --fresh --preset armclang*” ➔ To configure cmake with armclang preset. Users can set variables e.g. compiler, device etc in cache section.

  “cmake --fresh --preset armclang -DCOMPILER=GCC” ➔ Change/Override the COMPILER variable with GCC with command line.

  “cmake --fresh --preset armclang -DENABLE_USART=1” ➔ Enable USART driver with command line.

- “*cmake --build --preset armclang_build*” ➔ To compile configured cmake.

- “*cmake --workflow --preset armclang_workflow*” ➔ To run complete work flow i.e. configuration + compile + test + package.

```
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$ cmake --workflow --preset armclang_workflow
Executing workflow step 1 of 4: configure preset "armclang"

Preset CMake variables:

  BOARD_NAME="DevKit-e7"
  BOARD_REV="B0"
  BOOT="TCM"
  CMAKE_EXPORT_COMPILE_COMMANDS="ON"
  COMPILER="ARMCLANG"
  CPU="M55"
  DEVICE="AE722F80F55D5XX"
  DEVICE_SERIES="E7"
  OS="NONE"
  RTSS="HE"
  TEST_APP="ALL"

Preset environment variables:

  ARMLMD_LICENSE_FILE="27000@10.10.2.99"
  ARM_PRODUCT_DEF="/opt/arm/developmentstudio-2022.0/sw/mappings/gold.elmap"
  CMSIS_COMPILER_PATH="/mnt/c/Users/RajRanjan/AppData/Local/Arm/Packs/ARM/CMSIS-Compiler/2.1.0"
  CMSIS_PACK_PATH="/mnt/c/Users/RajRanjan/AppData/Local/Arm/Packs/ARM/CMSIS/6.0.0"
  COMPILER_BIN_PATH="/opt/arm/developmentstudio-2022.0/sw/ARMCompiler6.18/bin"
  PATH="/opt/arm/developmentstudio-2022.0/sw/ARMCompiler6.18/bin:/home/rajranjan/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr
/mnt/c/WINDOWS/system32:/mnt/c/WINDOWS:/mnt/c/WINDOWS/System32/Wbem:/mnt/c/WINDOWS/System32/OpenSSH/:/mnt/c/Program Files/Microsoft SQL S
nt/c/Program Files/dotnet/:/mnt/c/Users/RajRanjan/AppData/Roaming/Python/Python311/Scripts:/mnt/c/Program Files/Git/cmd:/mnt/c/Program Fi
t/c/Program Files/CMake/bin:/mnt/c/ProgramData/chocolatey/bin:/mnt/c/Program Files/7-Zip:/mnt/c/Users/RajRanjan/AppData/Local/Programs/Py
gram Files/GitHub CLI/:/mnt/c/Users/RajRanjan/AppData/Local/Programs/Python/Python311/Scripts/:/mnt/c/Users/RajRanjan/AppData/Local/Progr
```

Compilation with ./run.sh

“./run.sh “ ➔ By default, we have added two presets (armclang & gcc) and their build + test configuration (initially, later more presets can be added). With this command all presets, i.e. configure, build will run back-to-back.

```
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$
rajranjan@ALIFSEMI:/mnt/d/hhh/github/rajranjan/final/ensemble-cmsis-dfp_DEV/scripts/cmake$ ./run.sh


Running CMake with workflow preset:  "armclang_workflow"
Executing workflow step 1 of 4: configure preset "armclang"

Preset CMake variables:

  BOARD_NAME="DevKit-e7"
  BOARD_REV="B0"
  BOOT="TCM"
  CMAKE_EXPORT_COMPILE_COMMANDS="ON"
  COMPILER="ARMCLANG"
  CPU="M55"
  DEVICE="AE722F80F55D5XX"
  DEVICE_SERIES="E7"
  OS="NONE"
  RTSS="HE"
  TEST_APP="ALL"

Preset environment variables:

  ARMLMD_LICENSE_FILE="27000@10.10.2.99"
  ARM_PRODUCT_DEF="/opt/arm/developmentstudio-2022.0/sw/mappings/gold.elmap"
```

Change Configuration Parameter with Existing Presets

- ***"./run.sh -c -DRTSS=HE -DENABLE_RTC=1"*** ➔ With existing presets these parameters will be added/override in configuration and compiled.
- ***"./run.sh -b –jobs=32"*** ➔ With existing presets these parameters will be added/override in configuration and compiled.
- ***"./run.sh -p -DRTSS=HE"*** ➔ Default preset will be selected and RTSS configuration will be added/override in configuration, compiled and tested.
- ***"./run.sh -p armclang,armclang_build,armclang_build_test"*** ➔ armclang preset will be selected as configuration preset, armclang_build will be used as build preset, and armclang_build_test will be used as test preset.
- ***"./run.sh -p armclang,armclang_build,armclang_build_test -DRTSS=HE"*** ➔ armclang preset will be selected as configuration preset, armclang_build will be used as build preset, and armclang_build_test will be used as test preset. Also, configuration variable RTSS will be added/override in configuration, compiled and tested.
- ***"./run.sh -l"*** ➔ It will display all available config,build,test and workflow presets.
- ***"./run.sh -f"*** ➔ It will be used to force fresh and clean build.

Modifying CMake scripts

When it comes to modifying cmake script files, care should be taken as wrong changes would result in compilation error or even make file generation fail. There are different steps involved in modifying any cmake scripts.

Please follow the below steps, if cmake script modification is necessary.

Adding New Driver Files

1. The **drivers.cmake** script file is present in the directory path **scripts/cmake/rtss/driver_cmake/**.
2. This cmake file has each driver source file and headers files included with respect to each driver available.
3. Each driver related file is included to the build, based on respective macros which are enabled in the **RTE_Components.h** header file.

```
/* AlifSemiconductor::CMSIS Driver.SOC Peripherals.PINCONF */
#define RTE_Drivers_LL_PINCONF                  1          /* Driver PinPAD and PinMux          */
/* AlifSemiconductor::CMSIS Driver.USART */
#define RTE_Drivers_USART                       1          /* Driver UART                       */
/* AlifSemiconductor::CMSIS Driver.SOC Peripherals.GPIO */
#define RTE_Drivers_GPIO                        1          /* Driver GPIO                       */
```

4. When a new driver is added to the driver's folder, please make sure to add the same files in this cmake file, so that it will be compiled for the new build.

```
target_sources(${DRIVER_LIB} PRIVATE
    #Pin Config
    $<$<BOOL:${ENABLE_PIN_CONF}>:${ALIF_ENSEMBLE_DRIVERS_SRC_DIR}/pinconf.c>

    #ADC Driver
    $<$<BOOL:${ENABLE_ADC}>:${ALIF_ENSEMBLE_DRIVERS_SRC_DIR}/adc.c>
    $<$<BOOL:${ENABLE_ADC}>:${ALIF_CMSIS_DRIVER_SRC_DIR}/Driver_ADC.c>
```

With **ENABLE_ADC** flag add all required files needed for specific driver with proper file path.

Editing/Adding Test Applications in cmake for the Build

1. While cmake is being built, the test applications are considered according to the OS and name given in variable **TEST_APP**.
2. The cmake files for each OS type are available in the directory **scripts/cmake/rtss/os_cmake**
3. For each OS, specific cmake script has been added and it will contain all relevant commands.
4. Specific test app is added as follows with relevant flags

```
51  COND_FILE_ADD(${BARE_METAL_APP_DIR}/UART4_Baremetal.c          ENABLE_USART    TEST_APP_SRCS   "test-apps")
52  COND_FILE_ADD(${BARE_METAL_APP_DIR}/UART2_Baremetal.c          ENABLE_USART    TEST_APP_SRCS   "test-apps")
53  COND_FILE_ADD(${BARE_METAL_APP_DIR}/LPUART_Baremetal.c         ENABLE_USART    TEST_APP_SRCS   "test-apps")
```

If any supporting file is needed to compile specific testapp, user can add as follows

```
COND_FILE_ADD(${BARE_METAL_APP_DIR}/i2c_using_i3c_Baremetal.c       ENABLE_I2C          TEST_APP_SRCS   "test-apps")

eval_flags(TMP_FLAG     AND     ENABLE_RTC          ENABLE_LPTIMER)
if(${TMP_FLAG})
    COND_FILE_ADD(${BARE_METAL_APP_DIR}/Demo_pm_baremetal.c         ${TMP_FLAG}         TEST_APP_SRCS   "test-apps")
else()
    list(APPEND     RM_TEST_APPS_LIST       "Demo_pm_baremetal")
endif()
```

Note: Add test-app with flag/s and if that flag is not enabled, include in *RM_TEST_APPS_LIST* list.

A]. If any test-app depends on files that need to add as dependency, user has to do 2 things

```
COND_FILE_ADD(${BARE_METAL_APP_DIR}/Demo_SD_fatFS_Baremetal.c       ENABLE_SD           TEST_APP_SRCS       "test-apps")
COND_FILE_ADD(${BARE_METAL_APP_DIR}/FatFS/diskio.c                  ENABLE_SD       SD_TEST_APP_DEP_SRCS    "dependency")
COND_FILE_ADD(${BARE_METAL_APP_DIR}/FatFS/ff.c                      ENABLE_SD       SD_TEST_APP_DEP_SRCS    "dependency")
COND_FILE_ADD(${BARE_METAL_APP_DIR}/FatFS/ffsystem.c                ENABLE_SD       SD_TEST_APP_DEP_SRCS    "dependency")
COND_FILE_ADD(${BARE_METAL_APP_DIR}/FatFS/ffunicode.c               ENABLE_SD       SD_TEST_APP_DEP_SRCS    "dependency")
```

B]. Based-on test-app name add specific variable for example *SD_TEST_DEP_SRCS* to *addonsourcefiles* and reset for others.

```
foreach (testApp ${TEST_APPS})
    set(TEST_FOUND_FLAG "Not Found")
    foreach (testsourcefile ${TEST_APP_SRCS})

        # Collectiong Test app names one by one
        get_filename_component (tempname "${testsourcefile}" NAME)
        string (REPLACE ".c" "" testname ${tempname} )

        if("${testname}" STREQUAL "Demo_SD_fatFS_Baremetal")
            set(addonsourcefiles        ${SD_TEST_APP_DEP_SRCS})
        else()
            set(addonsourcefiles        "")
        endif()
```

5. If ***ALL*** is passed as the test application name i.e. TEST_APP, all the test applications under that given OS type will be considered.

Editing/Adding RTE_Components.h header file for the Build

1. There are two **RTE_Components.h** header files in the directory **scripts/cmake.Include_RTE_Comp**, one for **Ensemble** pack .
2. Based on the values passed with **RTSS (HE/HP)** and CPU **(M55)** as arguments/macros to **run.sh** script, the header file will be modified by the cmake script on the run time, with respect to the given configuration.
3. If there are any new Drivers added in the git repo source, then driver cmake is updated with the same. And make sure to add the required macro definitions in **RTE_Components.h** to enable the particular drivers for the compilation.

```
/* AlifSemiconductor::CMSIS Driver.SOC Peripherals.PINCONF */
#define RTE_Drivers_LL_PINCONF                  1          /* Driver PinPAD and PinMux          */
/* AlifSemiconductor::CMSIS Driver.USART */
#define RTE_Drivers_USART                       1          /* Driver UART                       */
/* AlifSemiconductor::CMSIS Driver.SOC Peripherals.GPIO */
#define RTE_Drivers_GPIO                        1          /* Driver GPIO                       */
```

4. To add specific driver users must do following thing
   - Add specific define in **RTE_Components.h** and that processing in **get_rte_macros** function in file (**rtss/utilities_func.cmake**)

```
GET_MACRO_VALUE("${RTEcomponentFile}"   RTE_Drivers_WDT         ENABLE_WDT)
GET_MACRO_VALUE("${RTEcomponentFile}"   RTE_Drivers_WIFI        ENABLE_WIFI)
GET_MACRO_VALUE("${RTEcomponentFile}"   RTE_Drivers_DMA         ENABLE_DMA)
GET_MACRO_VALUE("${RTEcomponentFile}"   RTE_Drivers_I2C         ENABLE_I2C)
```

   - Based on variable *for example ENABLE_I2C* is Boolean type and based on this flag in

drivers_cmake /drivers.cmake corresponding driver files should be added.

```
target_sources(${DRIVER_LIB} PRIVATE
    #MHU Driver
    $<$<BOOL:${ENABLE_MHU}>:${ALIF_ENSEMBLE_DRIVERS_SRC_DIR}/mhu_driver.c>
    $<$<BOOL:${ENABLE_MHU}>:${ALIF_ENSEMBLE_DRIVERS_SRC_DIR}/mhu_receiver.c>
    $<$<BOOL:${ENABLE_MHU}>:${ALIF_ENSEMBLE_DRIVERS_SRC_DIR}/mhu_sender.c>

    #Pin Config
    $<$<BOOL:${ENABLE_PIN_CONF}>:${ALIF_ENSEMBLE_DRIVERS_SRC_DIR}/pinconf.c>

    #UART Driver
    $<$<BOOL:${ENABLE_USART}>:${ALIF_ENSEMBLE_DRIVERS_SRC_DIR}/uart.c>
    $<$<BOOL:${ENABLE_USART}>:${ALIF_CMSIS_DRIVER_SRC_DIR}/Driver_USART.c>

    #GPIO Driver
    $<$<BOOL:${ENABLE_GPIO}>:${ALIF_CMSIS_DRIVER_SRC_DIR}/Driver_GPIO.c>
```