C++ development on bare metal embedded

Demonstrated on STM32

Matej Blagšič

January 1, 2022

Why C++

Started learning C++ for work to help on projects

Points of interest

- Compatible with C code
- Code organisation
- Type safety
- Don't write it twice with templates and wrappers
- More warnings and errors preventing runtime bugs
- Tools for safer and more managable code
- Shared code between Application and Embedded device

Tools

Provided tools and configurations

- ✓ Cross compiler (arm-none-eabi-[gcc,g++], clang?)
- ✓ Startup file (usually in assembler) and linker script
- ✓ Flash tools: stm32flash, jlinkexe, openocd, st-flash
- ✓ Build tools: make, cmake

Project design

`.git`	Git related
`.gitignore`	
`Core`	STM32 Core folders
`Drivers`	and project folder
`CubeMX`	
`Project`	
`Makefile`	Building tools
`CMakeLists.txt`	
`gcc-arm-none-eabi.cmake`	
`STM32-project-template.ioc`	STM32CubeMX files
`.mxproject`	
`default.nix`	Portable environment for nix
`.envrc`	

Tools

Toolchain file: `gcc-arm-none-eabi.cmake`

```
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_SYSTEM_PROCESSOR arm)

set(TOOLCHAIN_PREFIX arm-none-eabi-)
set(FLAGS
    "-fdata-sections -ffunction-sections --specs=nano.specs -Wl,--gc-sections")
set(CPP_FLAGS
    "-fno-rtti -fno-exceptions -fno-threadsafe-statics")

set(CMAKE_C_COMPILER ${TOOLCHAIN_PREFIX}gcc ${FLAGS})
set(CMAKE_ASM_COMPILER ${CMAKE_C_COMPILER})
set(CMAKE_CXX_COMPILER ${TOOLCHAIN_PREFIX}g++ ${FLAGS} ${CPP_FLAGS})
set(CMAKE_OBJCOPY ${TOOLCHAIN_PREFIX}objcopy)
set(CMAKE_SIZE ${TOOLCHAIN_PREFIX}size)
set(CMAKE_OBJDUMP ${TOOLCHAIN_PREFIX}objdump)

set(CMAKE_EXECUTABLE_SUFFIX_ASM ".elf")
set(CMAKE_EXECUTABLE_SUFFIX_C ".elf")
set(CMAKE_EXECUTABLE_SUFFIX_CXX ".elf")

set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)
```

Tools

CMake (1): CMakeLists.txt

```
project(sample-f407vg)
set(PROJECT_DIR ${CMAKE_CURRENT_SOURCE_DIR}/PROJECT)

set(MCU_FAMILY STM32F4xx)
set(MCU_MODEL STM32F407xx)
set(CPU_PARAMETERS
    -mcpu=cortex-m4
    -mthumb
    -mfpu=fpv4-sp-d16
    -mfloat-abi=hard)

set(STARTUP_SCRIPT ${CMAKE_CURRENT_SOURCE_DIR}\
    /CubeMX/startup_stm32f407xx.s)
set(MCU_LINKER_SCRIPT ${CMAKE_CURRENT_SOURCE_DIR}\
    /CubeMX/STM32F407VGTx_FLASH.ld)

set(EXECUTABLE ${CMAKE_PROJECT_NAME})

enable_language(C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_C_STANDARD_REQUIRED ON)
set(CMAKE_C_EXTENSIONS ON)
set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS ON)
```

```
set(STM32CUBEMX_INCLUDE_DIRECTORIES
    ${CMAKE_CURRENT_SOURCE_DIR}/Core/Inc
    ${CMAKE_CURRENT_SOURCE_DIR}/Drivers\
        /${MCU_FAMILY}_HAL_Driver/Inc
    ${CMAKE_CURRENT_SOURCE_DIR}/Drivers\
        /${MCU_FAMILY}_HAL_Driver/Inc/Legacy
    ${CMAKE_CURRENT_SOURCE_DIR}/Drivers/CMSIS\
        /Device/ST/${MCU_FAMILY}/Include
    ${CMAKE_CURRENT_SOURCE_DIR}/Drivers/CMSIS/Include)

set(PROJECT_INCLUDE_DIRECTORIES
    ${CMAKE_CURRENT_SOURCE_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/${PROJECT_DIR})

file(GLOB_RECURSE STM32CUBEMX_SOURCES
    ${CMAKE_CURRENT_SOURCE_DIR}/Core/*.c
    ${CMAKE_CURRENT_SOURCE_DIR}/Drivers/*.c)

file(GLOB_RECURSE PROJECT_SOURCES FOLLOW_SYMLINKS
    ${PROJECT_DIR}/*.cpp
    ${PROJECT_DIR}/*.c)

add_executable(${EXECUTABLE}
    ${STM32CUBEMX_SOURCES}
    ${PROJECT_SOURCES}
    ${STARTUP_SCRIPT})

target_compile_definitions(${EXECUTABLE} PRIVATE
    $<$<CONFIG:Debug>:DEBUG>
    ${MCU_MODEL}
    USE_HAL_DRIVER)

target_include_directories(${EXECUTABLE} PRIVATE
    ${STM32CUBEMX_INCLUDE_DIRECTORIES}
    ${PROJECT_INCLUDE_DIRECTORIES})
```

Tools

CMake (2): CMakeLists.txt

```
target_compile_options(${EXECUTABLE} PRIVATE
${CPU_PARAMETERS}
    -Wall
    -Wextra
    -Wpedantic
    -Wno-unused-parameter
    $<$<COMPILE_LANGUAGE:CXX>:
        -Wno-volatile
        -Wold-style-cast
        -Wuseless-cast
        -Wsuggest-override>
    $<$<CONFIG:Debug>:-Og -g3 -ggdb>
    $<$<CONFIG:Release>:-Og -g0>)

target_link_options(${EXECUTABLE} PRIVATE
    -T${MCU_LINKER_SCRIPT}
    ${CPU_PARAMETERS}
    -Wl,-Map=${CMAKE_PROJECT_NAME}.map
    --specs=nosys.specs
    -Wl,--start-group
    -lc
    -lm
    -lstdc++
    -lsupc++
    -Wl,--end-group
    -Wl,--print-memory-usage)

add_custom_command(TARGET ${EXECUTABLE} POST_BUILD
    COMMAND ${CMAKE_SIZE} $<TARGET_FILE:${EXECUTABLE}>
    COMMAND ${CMAKE_OBJCOPY} -O ihex $<TARGET_FILE:${EXECUTABLE}> ${EXECUTABLE}.hex
    COMMAND ${CMAKE_OBJCOPY} -O binary $<TARGET_FILE:${EXECUTABLE}> ${EXECUTABLE}.bin
    COMMAND ${CMAKE_OBJDUMP} -D ${EXECUTABLE} > ${EXECUTABLE}.s)
```

Tools

Makefile

```
.PHONY: all build cmake clean format

BUILD_DIR := build
BUILD_TYPE ?= Debug

all: build

${BUILD_DIR}/Makefile:
    cmake \
        -B${BUILD_DIR} \
        -DCMAKE_BUILD_TYPE=${BUILD_TYPE} \
        -DCMAKE_TOOLCHAIN_FILE=gcc-arm-none-eabi.cmake \
        -DCMAKE_EXPORT_COMPILE_COMMANDS=ON

cmake: ${BUILD_DIR}/Makefile

build: cmake
    $(MAKE) -C ${BUILD_DIR} --no-print-directory

SRCS := $(shell find . -name '*.[ch]' -or -name '*.[ch]pp')
%.format: %
    clang-format -i $<
format: $(addsuffix .format, ${SRCS})

clean:
    rm -rf $(BUILD_DIR)
```

Tools

Build result

```
[100%] Linking C executable sample-f407vg.elf
Memory region    Used Size    Region Size    %age Used
RAM:             1800 B       128 KB          1.37%
CCMRAM:          0 GB         64 KB           0.00%
FLASH:           9904 B       1 MB            0.94%
text          data      bss      dec      hex      filename
9776          120       1688     11584          2d40
/<path>/sample-f407vg/build/sample-f407vg.elf
[100%] Built target sample-f407vg
```

Connect C and C++

Name mangling

C differentiates functions and variables by their name only
C++ uses name mangling to encode function and variable names to facilitate overloading and visibility in different scopes
main is required to be defined in global namespace

Connect C and C++

Name mangling example

Use `extern "C" {...}` to declare a C-style naming section

```
#ifdef __cplusplus
extern "C"{
#endif
int foo(int k)
{
    return k + 5;
}
#ifdef __cplusplus
}
#endif

template <typename T>
T bar(T k)
{
    return k;
}

int main()
{
    auto t = foo(5);
    auto k = bar(5.5f);
    auto j = bar(5.5);
    auto l = bar(55);
}
```

```
foo:

main:

_Z3barIfET_S0_:

_Z3barIdET_S0_:

_Z3barIiET_S0_:
```

Connect C and C++

Name mangling example

Compiler explorer can demangle identifiers `www.godbolt.org`

```
foo:

main:

float bar<float>(float):

double bar<double>(double):

int bar<int>(int):
```

```
foo:

main:

_Z3barIfET_S0_:

_Z3barIdET_S0_:

_Z3barIiET_S0_:
```

Connect C and C++

"New" main

```cpp
// file: projectMain.cpp
#include <PROJECT/projectMain.h"
#include <PROJECT/Drivers/drivers.h"
#include <PROJECT/Message/message.h"

void projectMain()
{
    using namespace PROJ;
    Drivers::init();
    while (true)
    {
        Message::messageParse();
        Drivers::update();
    }
}
```

```cpp
// file: projectMain.h
#include <stdint.h>

#ifdef __cplusplus
extern "C" {
#endif

void projectMain();

#ifdef __cplusplus
}
#endif
```

Connect C and C++

C libraries

This includes user libraries or system drivers
Use `extern "C"` guard on include or inside headers (STM32 have headers guarded)

```
// file: someUserFile.[cpp,h]

#ifdef __cplusplus
extern "C"{
#endif

#include "someCLibrary.h"

#ifdef __cplusplus
}
#endif
```

```
// file: someCLibrary.h
#pragma once

#ifdef __cplusplus
extern "C"{
#endif

...

#ifdef __cplusplus
}
#endif
```

Connect C and C++

Calling "new main"

```
// file: main.c
#include "main.h"
...

#include <PROJECT/projectMain.h>

int main()
{
    // System init
    ...

    projectMain();

    while (1)
    {
    }
}
```

How about STM32CubeIDE?