CEC Continuing Education Center

DesignNews

Introduction to Build Systems and CMake

DAY 2 : CMake Fundamentals

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

01

Review: The Problem

The Problem

There are several problems that teams are facing:

- Managing multiple build configurations
- Slow builds
- Software quality issues
- Inability to use modern techniques like DevOps, Simulation, TDD, etc, effectively
- Productivity issues (time to market, product quality)

The Solution

A carefully designed CMake build system will:

- Simplify build configurations with better dependency management
- Allow for faster, cross-platform builds
- Enable consistency across different development environments
- Unlock modern development processes and tools like DevOps, Simulation, and TDD
- Increase productivity

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

THE SPEAKER

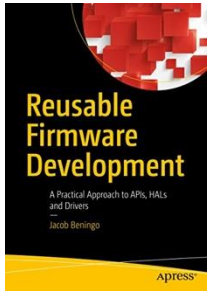
Jacob Beningo

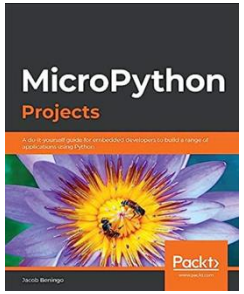
Jacob@beningo.com

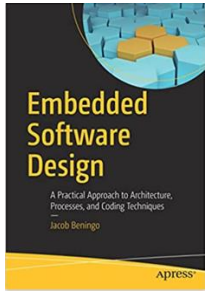
Beningo Embedded Group – CEO / Founder

Focus: Embedded Software Consulting and Training

Help teams deliver higher-quality embedded software faster. We specialize in creating and promoting embedded software excellence in businesses around the world.







Blogs for:

- DesignNews.com
- Embedded.com
- EmbeddedRelated.com
- MLRelated.com

Visit **www.beningo.com** to learn more

The Plan

Transform Your Build Process: Streamline, Modernize, and Boost Productivity with CMake

Step 1

Learn the Technology

Step 2

Design the Solution

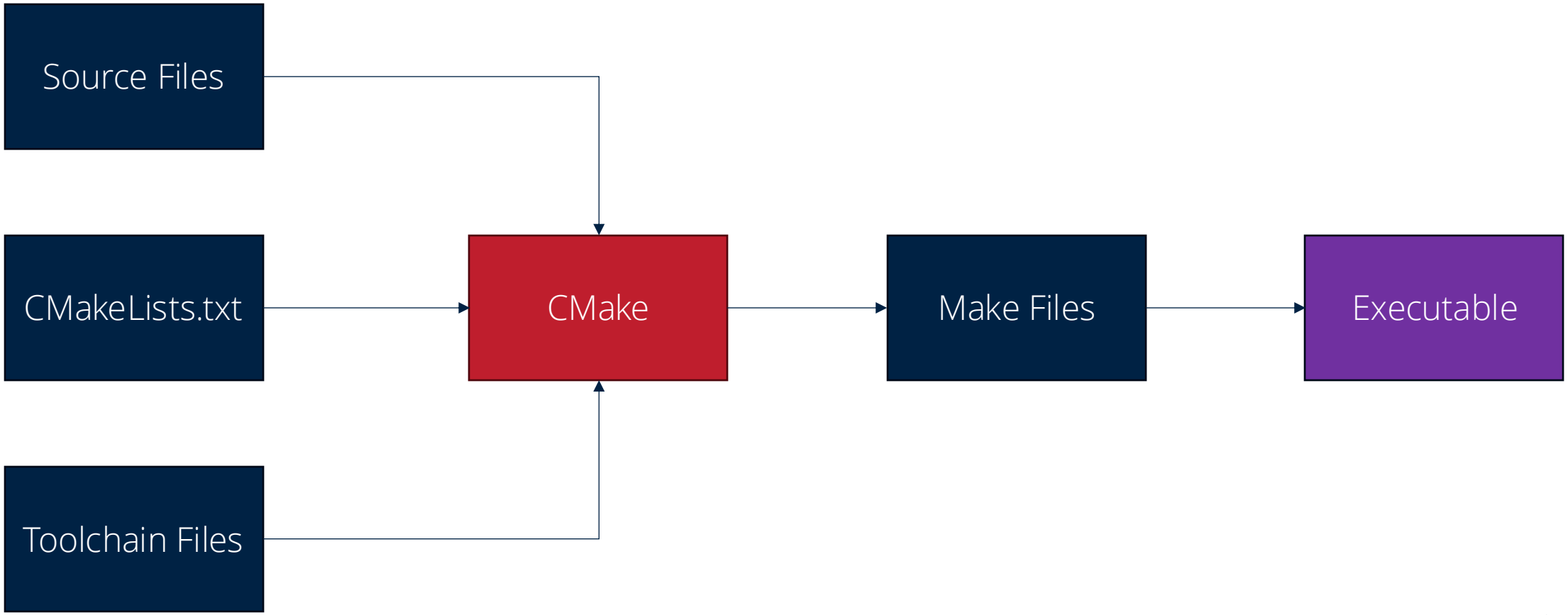
Step 3

Adopt Modern Practices

02

CMake Fundamentals

CMake Fundamentals - Overview



CMake Fundamentals – CMakeLists.txt

- CMakeLists.txt contains
  - Commands
    - cmake_minimum_required(VERSION 3.12)
    - add_executable()
    - add_library()
  - Variables
    - set(TARGET controller)
    - set(ALL_C_SOURCES "")

```
MyEmbeddedProject/
├── CMakeLists.txt
├── src/
│   ├── CMakeLists.txt
│   ├── main.c
│   ├── utils.c
│   └── device_drivers/
│       ├── CMakeLists.txt
│       ├── gpio_driver.c
│       └── uart_driver.c
├── include/
│   ├── gpio_driver.h
│   ├── uart_driver.h
│   └── utils.h
└── third_party/
    ├── CMakeLists.txt
    └── some_library/
        ├── CMakeLists.txt
        ├── some_library.c
        └── some_library.h
```

CMake Fundamentals – An Example CMakeLists.txt

```
1   cmake_minimum_required(VERSION 3.10)
2   project(controller C CXX ASM)
3
4   # Set target
5   set(TARGET controller)
6
7   # Set the output directory for the executables
8   set(EXE_DIR ${CMAKE_BINARY_DIR}/bin)
9   set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${EXE_DIR})
10
11  # Initialize lists for sources and include directories
12  set(ALL_C_SOURCES "")
13  set(ALL_CPP_SOURCES "")
14  set(ALL_ASM_SOURCES "")
15  set(ALL_INCLUDE_DIRS "")
16
17  # Add subdirectories
18  add_subdirectory(firmware/app)
19  add_subdirectory(firmware/bsp)
20  add_subdirectory(firmware/drv)
21  add_subdirectory(firmware/hal)
22  add_subdirectory(firmware/lib)
```

```
29  # Add the executable target
30  add_executable(${PROJECT_NAME} ${ALL_C_SOURCES} ${ALL_CPP_SOURCES} ${ALL_ASM_SOURCES})
31
32  set_target_properties(${PROJECT_NAME} PROPERTIES OUTPUT_NAME ${PROJECT_NAME}.elf)
33
34  # Set the include directories for the target
35  target_include_directories(${PROJECT_NAME} PRIVATE ${ALL_INCLUDE_DIRS})
36
37  # Detect if we are inside a Docker container
38  if(EXISTS "/.dockerenv")
39      set(SOURCE_PATH_PREFIX "/home/app/")
40  else()
41      set(SOURCE_PATH_PREFIX "")
42  endif()
43
44  # The following line is needed to make the debugger work. It removes the docker image
45  # path /home/app from the source file paths.
46  target_compile_options(controller PRIVATE "-fdebug-prefix-map=${CMAKE_SOURCE_DIR}=/")
```

Audience POLL Question

Which is your biggest concern with using a build system?

a) Compiling mixed projects for C, C++ and Assembly
b) Scalability for long-term support
c) Compilation speed
d) Enabling support for modern techniques like DevOps, TDD, etc
e) Other (Specify in the chat)

03

Target Management

Target Management – Building with CMake and Ninja

```
cmake -Bbuild -GNinja
```

```
ninja -C build
```

Build Targets:

1) Debug
2) Release
3) Test
4) Simulate
5) Analyze

Target Management – "Project Manager"

- project.sh / devManager.sh
  - Shell script that invokes CMake for the desired build type

```
./project.sh docker_run
./project.sh debug
./project.sh release
./project.sh simulation
```

Target Management – Example script

```
#!/bin/bash

BUILD_ROOT_DIR="build"
TOOLCHAIN_DIR="config" # Directory where toolchain files are now located
TOOLCHAIN_FILE="${TOOLCHAIN_DIR}/toolchain-arm.cmake" # Default toolchain

if [ "$1" = "erase-all" ]; then
    rm -rf $BUILD_ROOT_DIR
    exit 0
fi

case "$1" in
    release)
        BUILD_SUB_DIR="release"
        BUILD_TYPE="Release"
        ;;
    simulation)
        BUILD_SUB_DIR="simulation"
        BUILD_TYPE="Simulation"
        TOOLCHAIN_FILE="${TOOLCHAIN_DIR}/toolchain-host.cmake" # Use the host toolchain for simulation
        ;;
    debug|*)
        BUILD_SUB_DIR="debug"
        BUILD_TYPE="Debug"
        ;;
esac
```

Target Management – Example script

```
28  BUILD_DIR="$BUILD_ROOT_DIR/$BUILD_SUB_DIR"
29
30  case "$2" in
31      clean)
32          ninja -C $BUILD_DIR clean
33          ;;
34      erase)      You, 3 months ago • Added the latest baseline build system files. L…
35          rm -rf $BUILD_DIR
36          ;;
37      build|*)
38          # Always run CMake to ensure configuration is up to date.
39          cmake -DCMAKE_TOOLCHAIN_FILE=$TOOLCHAIN_FILE -G Ninja -B $BUILD_DIR -S . -DCMAKE_BUILD_TYPE=$BUILD_TYPE
40          ninja -C $BUILD_DIR
41          ;;
42      verbose)
43          # For the verbose flag, make sure cmake configuration is also considered.
44          cmake -DCMAKE_TOOLCHAIN_FILE=$TOOLCHAIN_FILE -G Ninja -B $BUILD_DIR -S . -DCMAKE_BUILD_TYPE=$BUILD_TYPE
45          ninja -C $BUILD_DIR -v
46          ;;
47  esac
```

Audience POLL Question

What does the script file do for you?

a) Simplifies building a CMake project
b) Removes the need to memorize command line sequences
c) Provides scalable interface for managing a software project
d) All the above
e) None of the above

Patterns and Antipatterns

04

Patterns & Antipatterns – "Good Practices"

- Treat CMake as code (Keep it readable and clean)
- Think in targets
- Make ALIAS targets to keep usage consistent
  - add_subdirectory and find_package should provide the same targets
- Use lowercase function names (uppercase is for variables)
- Select a minimum version (base it on the features you need)

Patterns & Antipatterns – "Poor Practices"

- Don't add unneeded public requirements (like –Wall). Make them PRIVATE
- Minimize or don't use global functions
- Link to built files directly (link to the targets)
- Use set() with scope
- Ignoring Build Type (CMAKE_BUILD_TYPE)
  - Can lead to non-optimized or debug builds in production
- Overcomplicating CMakeLists.txt files

Next Steps

05

Embedded Build System

Transform your build system with the free Beningo Embedded Build System example:

- Docker container build system
- Makefile-based
- CMake with Ninja Example
- Compilation scripts
- Integrated tools like cpputest

https://mailchi.mp/beningo/beningo-devops

Additional Resources

Please consider the resources below:

- Jacob's Blogs
- Jacob's CEC courses
- Embedded Software Academy

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

Consulting | Coaching | Training

www.beningo.com

Next Steps

- ✓ Introduction to Embedded Build Systems
- ✓ CMake Fundamentals
- CMake for Embedded Systems
- Designing your Build System
- Adopting Modern Practices

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets