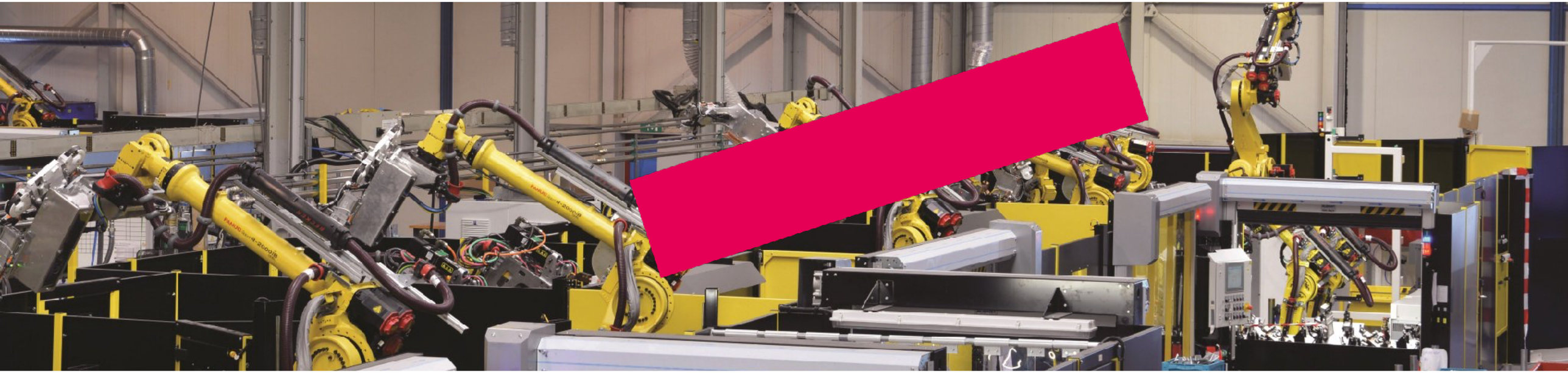
Programming 6



Workshop on CMake

johan.korten@han.nl

V1.0 Nov 2025

HAN_UNIVERSITY
OF APPLIED SCIENCES

Workshop Topics overview

Clean code (beyond SOLID)

CMake

Unit testing

Commentaar

Patterns (Clean architecture)

Workshop Topics for Today

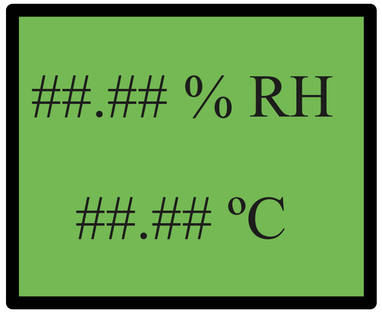
- Overcoming the Build Nightmare

- Cmake ins- and outs
- Cmake for Embedded targets
- Alternative to Make

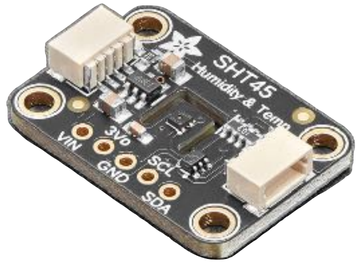
Example project for today

SHT45

- Temperature
- Humidity Sensor



Lab Environment Monitor



Pragmatic start - Stepping stone 1

Welcome | main.cpp ×

main.cpp > main()

```cpp
// Our starter project
// Cmake Workshop Prog 6
//
// HAN University of Applied Sciences
// Johan Korten - v1.0 2025
//
// main.cpp
// compile using: g++ main.cpp -o sensor_app

#include <iostream>

int main() {
    std::cout << "Reading SHT45 sensor..." << std::endl;
    return 0;
}
```

Pragmatic start - Stepping stone 1

Welcome | main.cpp ×

main.cpp > main()

```cpp
// Our starter project
// Cmake Workshop Prog 6
//
// HAN University of Applied Sciences
// Johan Korten - v1.0 2025
//
// main.cpp
// compile using: g++ main.cpp -o sensor_app

#include <iostream>

int main() {
    std::cout << "Reading SHT45 sensor..." << std::endl;
    return 0;
}
```

Compile and test:

```
jakorten@device-35 SimpleProject % g++ main.cpp -o sensor_app
jakorten@device-35 SimpleProject % ./sensor_app
Reading SHT45 sensor...
jakorten@device-35 SimpleProject %
```

Towards a library - Stepping stone 2

Welcome | main.cpp ×

main.cpp > main()

```cpp
// Our starter project
// Cmake Workshop Prog 6 - Step 2
//
// HAN University of Applied Sciences
// Johan Korten - v1.0 2025
//
// main.cpp
//

#include <iostream>
#include "sht45.hpp"

int main() {
    SHT45 sensor;
    std::cout << "Temperature: " << sensor.read_temperature() << std::endl;
    return 0;
}
```

Compile and test:

Towards a library - Stepping stone 2

Welcome | main.cpp

main.cpp > main()

```cpp
// Our starter project
// Cmake Workshop Prog 6 - Step 2
//
// HAN University of Applied Sciences
// Johan Korten - v1.0 2025
//
// main.cpp
//

#include <iostream>
#include "sht45.hpp"

int main() {
    SHT45 sensor;
    std::cout << "Temperature: " << sensor.read_temperature() << std::endl;
    return 0;
}
```

Compile and test: (Whoops…)

```
jakorten@device-35 SimpleLibrary % g++ main.cpp -o sensor_app
Undefined symbols for architecture arm64:
  "SHT45::read_temperature()", referenced from:
      _main in main-ecf635.o
ld: symbol(s) not found for architecture arm64
clang++: error: linker command failed with exit code 1 (use -v to see invocation
)
```

Towards a library - Stepping stone 2

Welcome | main.cpp

main.cpp > main()

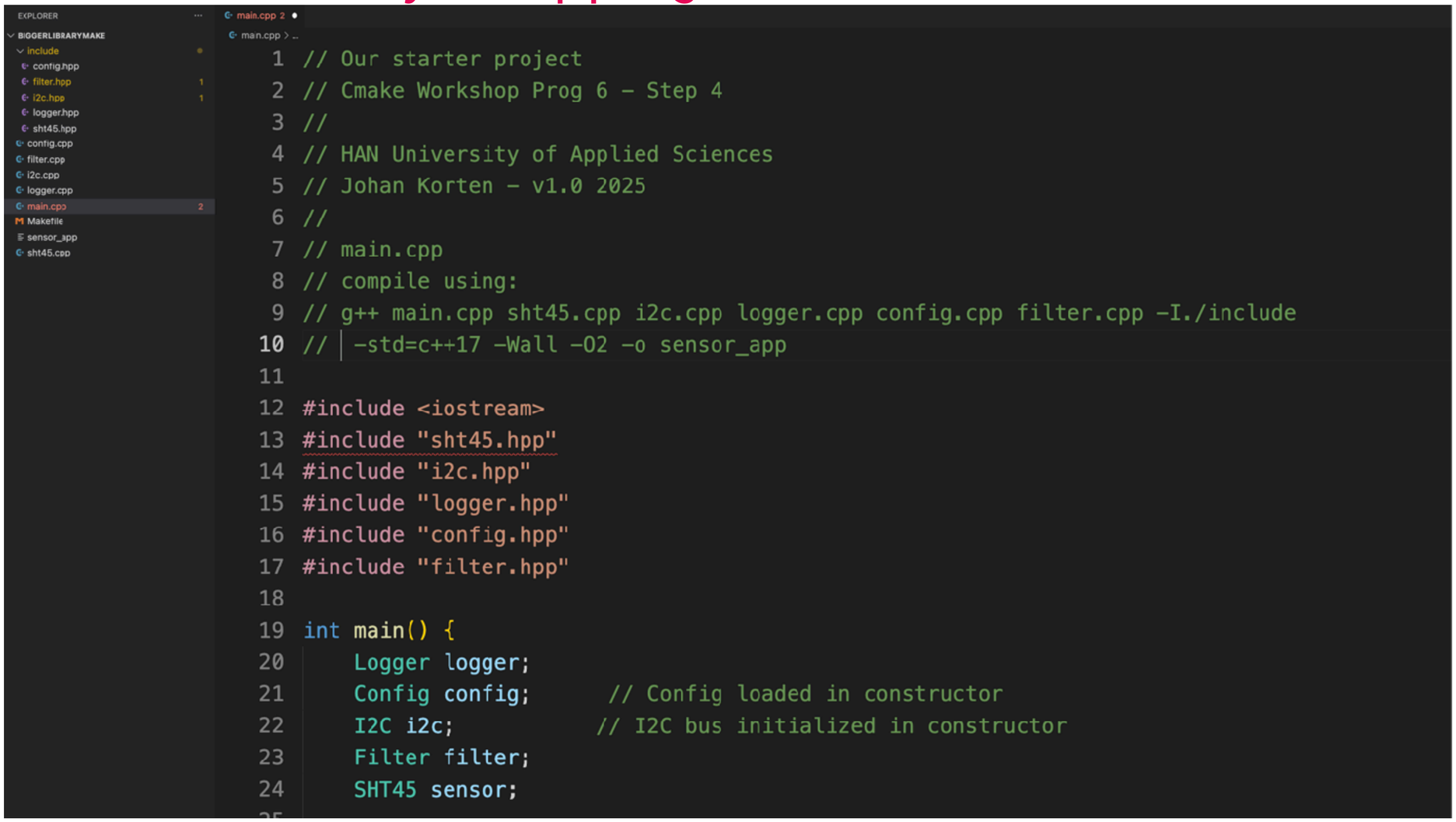
```cpp
// Our starter project
// Cmake Workshop Prog 6 - Step 2
//
// HAN University of Applied Sciences
// Johan Korten - v1.0 2025
//
// main.cpp
//

#include <iostream>
#include "sht45.hpp"

int main() {
    SHT45 sensor;
    std::cout << "Temperature: " << sensor.read_temperature() << std::endl;
    return 0;
}
```

Compile and test: (Pfwiew, we got away for now…)

```
jakorten@device-35 SimpleLibrary % g++ main.cpp -o sensor_app
Undefined symbols for architecture arm64:
  "SHT45::read_temperature()", referenced from:
      _main in main-ecf635.o
ld: symbol(s) not found for architecture arm64
clang++: error: linker command failed with exit code 1 (use -v to see invocation
)
jakorten@device-35 SimpleLibrary % g++ main.cpp sht45.cpp -o sensor_app
jakorten@device-35 SimpleLibrary %
```

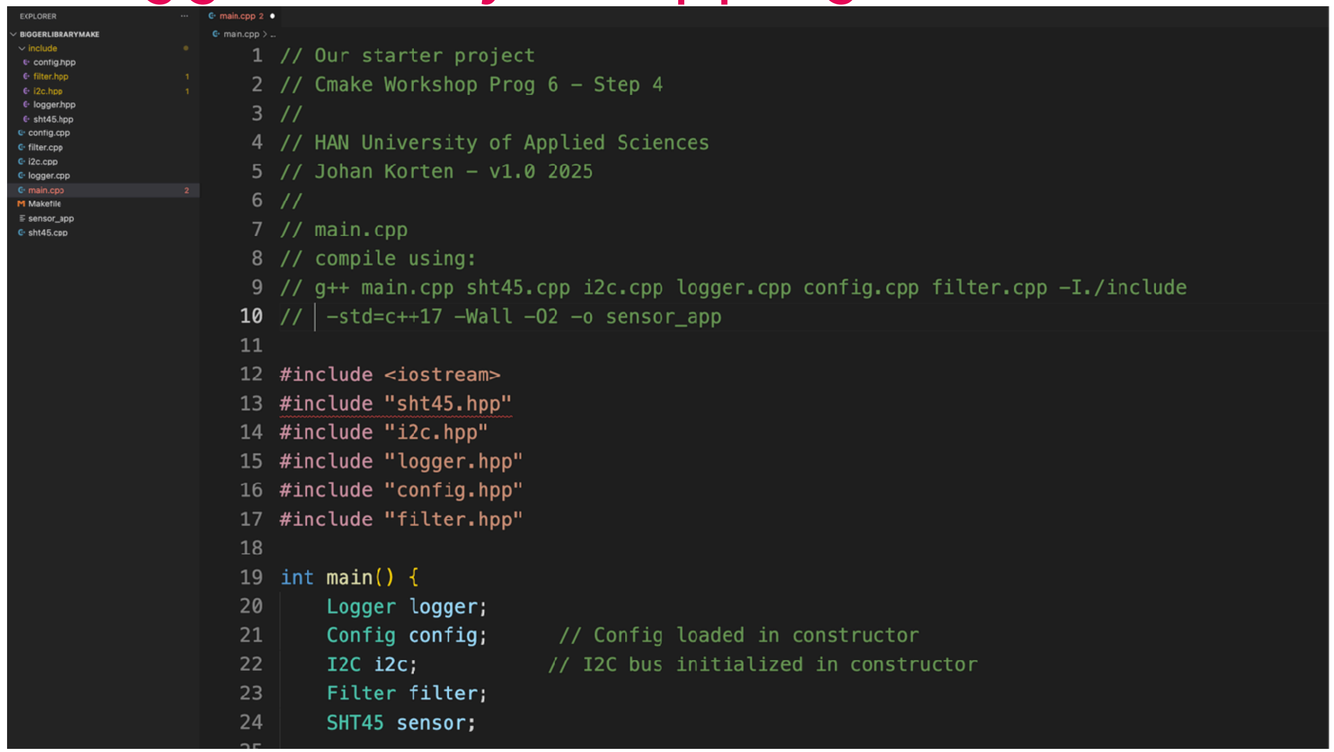
Towards a library - Stepping stone 3

Build Nightmare



A bigger library - Stepping stone 4

Build Nightmare




```
g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include -std=c++17 -Wall -O2 -o sensor_app
```

Fighting Commandline Nightmares - Stepping stone 5

Build Nightmare

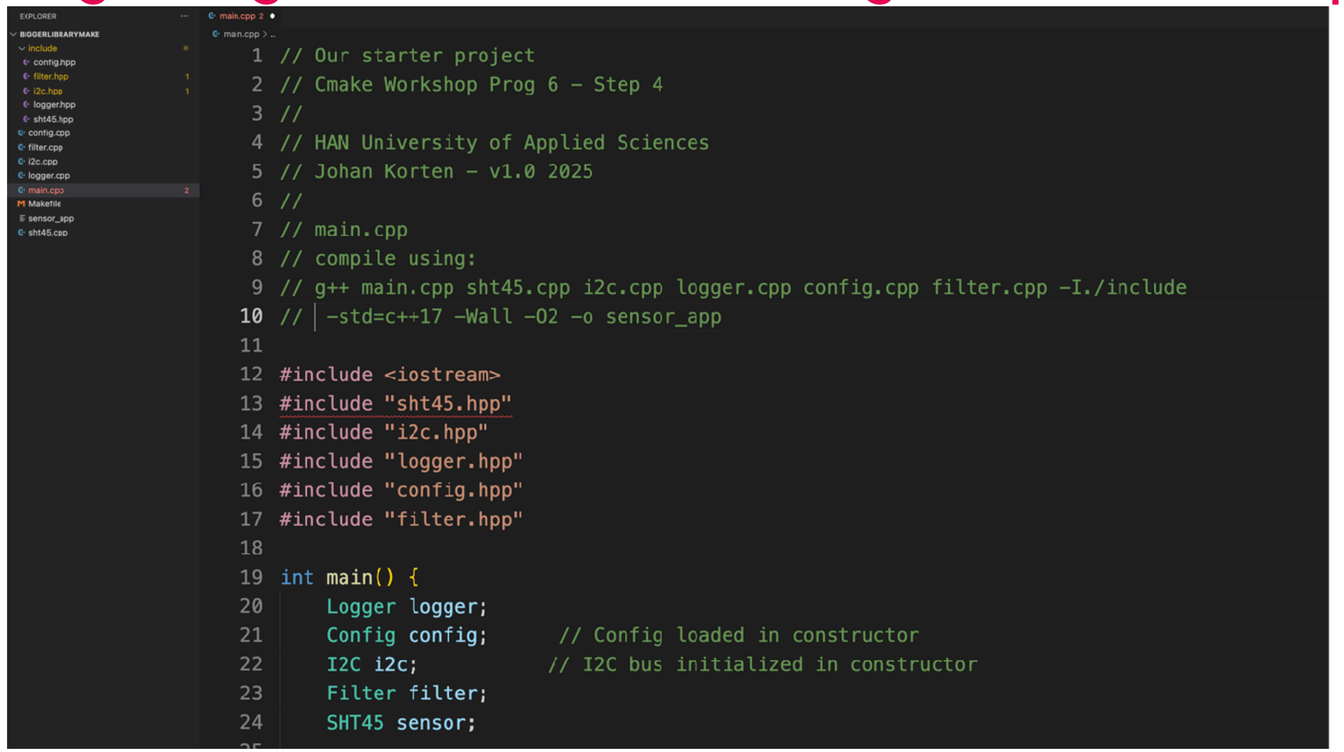
```
g++ main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp -I./include -std=c++17 -Wall -O2 -o sensor_app
```

Alternative: Makefile

```
# Makefile for Sensor Application
# CMake Workshop - HAN University of Applied Sciences
# Johan Korten - v1.0 2025

# Compiler and flags
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -O2 -I./include
LDFLAGS =

# Target executable
TARGET = sensor_app

# Source files
SOURCES = main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp

# Object files (replace .cpp with .o)
OBJECTS = main.o sht45.o i2c.o logger.o config.o filter.o

# Header files (for dependency tracking)
HEADERS = include/sht45.hpp include/i2c.hpp include/logger.hpp include/config.hpp include/filter.hpp

# Default target
all: $(TARGET)

# Link object files to create executable
$(TARGET): $(OBJECTS)
        $(CXX) $(OBJECTS) $(LDFLAGS) -o $(TARGET)
        @echo "Build complete: $(TARGET)"

# Compile main.cpp
main.o: main.cpp $(HEADERS)
        $(CXX) $(CXXFLAGS) -c main.cpp

# Compile sht45.cpp
sht45.o: sht45.cpp include/sht45.hpp
        $(CXX) $(CXXFLAGS) -c sht45.cpp

# Compile i2c.cpp
i2c.o: i2c.cpp include/i2c.hpp
        $(CXX) $(CXXFLAGS) -c i2c.cpp

# Compile logger.cpp
logger.o: logger.cpp include/logger.hpp
        $(CXX) $(CXXFLAGS) -c logger.cpp

# Compile config.cpp
config.o: config.cpp include/config.hpp
        $(CXX) $(CXXFLAGS) -c config.cpp

# Compile filter.cpp
filter.o: filter.cpp include/filter.hpp
        $(CXX) $(CXXFLAGS) -c filter.cpp

# Clean build artifacts
clean:
        rm -f $(OBJECTS) $(TARGET)
        @echo "Clean complete"

# Rebuild everything from scratch
rebuild: clean all

# Run the application
run: $(TARGET)
        ./$(TARGET)

# Show help
help:
        @echo "Available targets:"
        @echo "  all     - Build the application (default)"
        @echo "  clean   - Remove all build artifacts"
        @echo "  rebuild - Clean and build from scratch"
        @echo "  run     - Build and run the application"
        @echo "  help    - Show this help message"

.PHONY: all clean rebuild run help
```

Fighting Commandline Nightmares - Stepping stone 5

Build Nightmare




```
jakorten@device-35 BiggerLibraryMake % make
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c main.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c sht45.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c i2c.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c logger.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c config.cpp
g++ -std=c++17 -Wall -Wextra -O2 -I./include -c filter.cpp
g++ main.o sht45.o i2c.o logger.o config.o filter.o  -o sensor_app
Build complete: sensor_app
jakorten@device-35 BiggerLibraryMake % make run
./sensor_app
[LOG] Starting sensor application...
Temperature: 20.25
[LOG] Application complete
jakorten@device-35 BiggerLibraryMake %
```

```
jakorten@device-35 BiggerLibraryMake % make clean
rm -f main.o sht45.o i2c.o logger.o config.o filter.o sensor_app
Clean complete
jakorten@device-35 BiggerLibraryMake %
```

```
cd BiggerLibraryMake
make              # Build the project
make run          # Build and run
make clean        # Clean up
make help         # See available commands
```

The problem of makefiles

Build Nightmare

‘Makefile hell’: 200+ lines for a simple project

IDE lock-in: Can't share code between STM32CubeIDE / VS Code / CLion / whatever.

Fighting Commandline Nightmares - Stepping stone 5

From the cli nightmare to the make nightmare…

Pro's:

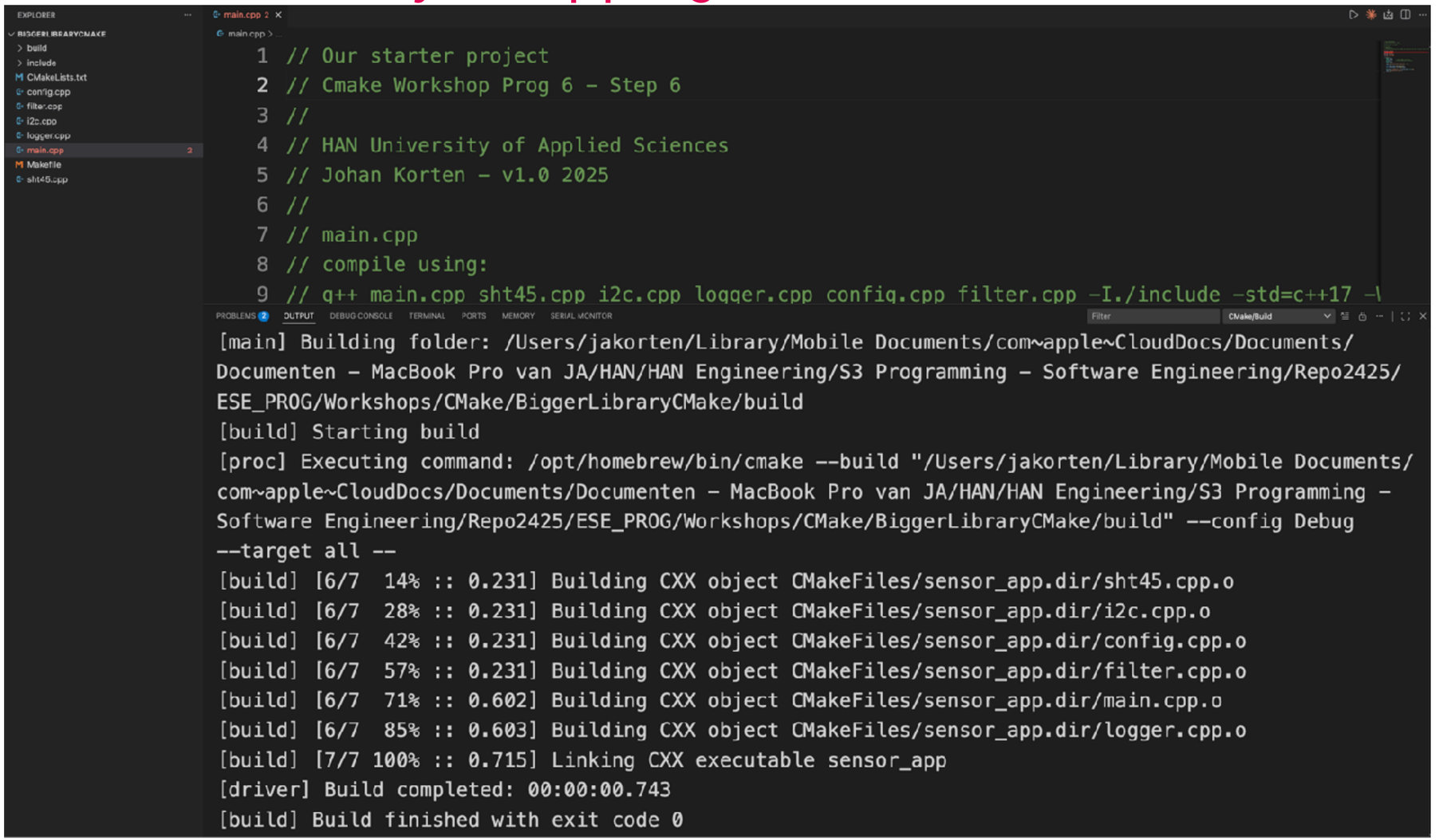
1. Incremental compilation - Only recompiles changed files
2. Dependency tracking - Recompiles when headers change
3. Clean separation - Variables for compiler, flags, sources
4. Helper targets - clean, rebuild, run, help
5. Standard conventions - Uses CXX, CXXFLAGS, etc.

Con's:

1. Manual listing - Every .cpp and .o file must be listed
2. Repetitive rules - 6 nearly identical compilation rules
3. Manual dependency tracking - Have to specify which .cpp depends on which .hpp
4. Platform-specific - Won't work on Windows without changes
5. Hard-coded compiler - g++ might not be available everywhere
6. Tedious maintenance - Add new file = edit 3+ places in Makefile

Towards a library - Stepping stone 6

Build Nightmare




```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..          # Generate build system
make              # Build (or 'cmake --build .')
./sensor_app      # Run
```

Towards a library - Stepping stone 6

Build Nightmare

Makefile

```
# Makefile for Sensor Application
# CMake Workshop – HAN University of Applied Sciences
# Johan Korten – v1.0 2025

# Compiler and flags
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -O2 -I./include
LDFLAGS =

# Target executable
TARGET = sensor_app

# Source files
SOURCES = main.cpp sht45.cpp i2c.cpp logger.cpp config.cpp filter.cpp

# Object files (replace .cpp with .o)
OBJECTS = main.o sht45.o i2c.o logger.o config.o filter.o
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..        # Generate build system
make            # Build (or 'cmake --build .')
./sensor_app    # Run
```

Makefile 1/5 1..17

Towards a library - Stepping stone 6

Build Nightmare

Makefile

```
18
19 # Header files (for dependency tracking)
20 HEADERS = include/sht45.hpp include/i2c.hpp include/logger.hpp include/config.hpp include/
21
22 # Default target
23 all: $(TARGET)
24
25 # Link object files to create executable
26 $(TARGET): $(OBJECTS)
27     $(CXX) $(OBJECTS) $(LDFLAGS) -o $(TARGET)
28     @echo "Build complete: $(TARGET)"
29
30 # Compile main.cpp
31 main.o: main.cpp $(HEADERS)
32     $(CXX) $(CXXFLAGS) -c main.cpp
33
34 # Compile sht45.cpp
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..        # Generate build system
make            # Build (or 'cmake --build .')
./sensor_app    # Run
```

Makefile 2/5 18..33

Towards a library - Stepping stone 6

Build Nightmare

Makefile

```
34 # Compile sht45.cpp
35 sht45.o: sht45.cpp include/sht45.hpp
36     $(CXX) $(CXXFLAGS) -c sht45.cpp
37
38 # Compile i2c.cpp
39 i2c.o: i2c.cpp include/i2c.hpp
40     $(CXX) $(CXXFLAGS) -c i2c.cpp
41
42 # Compile logger.cpp
43 logger.o: logger.cpp include/logger.hpp
44     $(CXX) $(CXXFLAGS) -c logger.cpp
45
46 # Compile config.cpp
47 config.o: config.cpp include/config.hpp
48     $(CXX) $(CXXFLAGS) -c config.cpp
49
50 # Compile filter.cpp
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..          # Generate build system
make              # Build (or 'cmake --build .')
./sensor_app      # Run
```

Makefile 3/5 34..49

Towards a library - Stepping stone 6

Build Nightmare

Makefile

```
47 config.o: config.cpp include/config.hpp
50 # Compile filter.cpp
51 filter.o: filter.cpp include/filter.hpp
52     $(CXX) $(CXXFLAGS) -c filter.cpp
53
54 # Clean build artifacts
55 clean:
56     rm -f $(OBJECTS) $(TARGET)
57     @echo "Clean complete"
58
59 # Rebuild everything from scratch
60 rebuild: clean all
61
62 # Run the application
63 run: $(TARGET)
64     ./$(TARGET)
65
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..        # Generate build system
make            # Build (or 'cmake --build .')
./sensor_app    # Run
```

Makefile 4/5 47..64

Towards a library - Stepping stone 6

Build Nightmare

Makefile

```
63 run: $(TARGET)
65
66 # Show help
67 help:
68     @echo "Available targets:"
69     @echo "  all     - Build the application (default)"
70     @echo "  clean   - Remove all build artifacts"
71     @echo "  rebuild - Clean and build from scratch"
72     @echo "  run     - Build and run the application"
73     @echo "  help    - Show this help message"
74
75 .PHONY: all clean rebuild run help
76
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..        # Generate build system
make            # Build (or 'cmake --build .')
./sensor_app    # Run
```

Makefile 5/5 47..64

Towards a library - Stepping stone 6

Build Nightmare

CMakeLists.txt

```
# CMakeLists.txt for Sensor Application
# CMake Workshop – HAN University of Applied Sciences
# Johan Korten – v1.0 2025

# Minimum CMake version required
cmake_minimum_required(VERSION 3.10)

# Project name and version
project(SensorApp VERSION 1.0 LANGUAGES CXX)

# Set C++ standard
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)

# Compiler flags
add_compile_options(-Wall -Wextra -O2)
```

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..        # Generate build system
make            # Build (or 'cmake --build .')
./sensor_app    # Run
```

CMakeLists.txt
1/2 1..17

Towards a library - Stepping stone 6

Build Nightmare

```
cd BiggerLibraryCMake
mkdir build
cd build
cmake ..        # Generate build system
make            # Build (or 'cmake --build .')
./sensor_app    # Run
```

CMakeLists.txt

```
17 add_compile_options(-Wall -Wextra -O2)
18
19 # Include directories
20 include_directories(include)
21
22 # Source files
23 set(SOURCES
24     main.cpp
25     sht45.cpp
26     i2c.cpp
27     logger.cpp
28     config.cpp
29     filter.cpp
30 )
31
32 # Create executable
33 add_executable(sensor_app ${SOURCES})
34
35 # Print build information
36 message(STATUS "Building ${PROJECT_NAME} version ${PROJECT_VERSION}")
37 message(STATUS "C++ Standard: C++${CMAKE_CXX_STANDARD}")
38
```

CMakeLists.txt
2/2 17..37

Makefile versus CMakeLists.txt

1. No manual .o file listing - CMake figures it out
2. No repetitive rules - One add_executable() does it all
3. Automatic dependency tracking - CMake scans headers
4. Cross-platform - Same file works on Windows/Linux/Mac
5. Modern build system - Generates Makefiles, Visual Studio projects, Ninja, etc.

Makefile versus CMakeLists.txt


```
# Just list the sources
set(SOURCES
    main.cpp
    sht45.cpp
    i2c.cpp
    logger.cpp
    config.cpp
    filter.cpp
)

# CMake figures out the rest!
add_executable(sensor_app ${SOURCES})
```

Workshop Topics for Today

- Overcoming the Build Nightmare
- **Cmake ins- and outs**
- Cmake for Embedded targets
- Alternatives to make

Towards a library - Stepping stone 6

BiggerLibraryCMakeV2:

Stage 1 - What we currently have

Stage 2 - Updated CMake

Stage 3 - Separate libraries for reusability

Stage 4 - FetchContent for external libraries

Stage 5 - How real projects configure CMake

Stage 6 - CMake for Embedded Systems targets

CMake V2 stage2

BiggerLibraryCMakeV2:

3.20+ to use ‘modern’ features.

```
target_include_directories(sht45_lib PUBLIC include)   # vs
target_include_directories(sht45_lib PRIVATE include)  # vs
target_include_directories(sht45_lib INTERFACE include)
```

Tip: always use “PRIVATE” unless you need INTERFACE or PUBLIC

```
8  cmake_minimum_required(VERSION 3.20)  # Updated for modern features
9  project(SensorApp VERSION 1.0 LANGUAGES CXX)
10
11 # Set C++ standard
12 set(CMAKE_CXX_STANDARD 17)
13 set(CMAKE_CXX_STANDARD_REQUIRED ON)
14 set(CMAKE_CXX_EXTENSIONS OFF)
15
16 # Source files
17 set(SOURCES
18     main.cpp
19     sht45.cpp
20     i2c.cpp
21     logger.cpp
22     config.cpp
23     filter.cpp
24 )
25
26 # Create executable
27 add_executable(sensor_app ${SOURCES})
28
29 # MODERN: Use target-specific commands with PRIVATE visibility
30 # This means ONLY sensor_app uses these settings, not everything in the project!
31 target_include_directories(sensor_app PRIVATE include)
32 target_compile_options(sensor_app PRIVATE -Wall -Wextra -O2)
33
34 # Print build information
35 message(STATUS "Building ${PROJECT_NAME} version ${PROJECT_VERSION}")
36 message(STATUS "C++ Standard: C++${CMAKE_CXX_STANDARD}")
37 message(STATUS "Using modern target-specific commands!")
38
```

CMake V2 stage3

Up-to-date CMake

BiggerLibraryCMakeV2:

Using Libraries:

```
# Create sensor hardware library (SHT45 + I2C)
add_library(sht45_lib STATIC
    sht45.cpp
    i2c.cpp
)
target_include_directories(sht45_lib PUBLIC include)
target_compile_options(sht45_lib PRIVATE -Wall -Wextra -O2)
```

```
# Link libraries to executable
target_link_libraries(sensor_app
    PRIVATE
        sht45_lib
        utils_lib
)
```

```
8  cmake_minimum_required(VERSION 3.20)
9  project(SensorApp VERSION 1.0 LANGUAGES CXX)
10
11 set(CMAKE_CXX_STANDARD 17)
12 set(CMAKE_CXX_STANDARD_REQUIRED ON)
13 set(CMAKE_CXX_EXTENSIONS OFF)
14
15 # Create sensor hardware library (SHT45 + I2C)
16 add_library(sht45_lib STATIC
17     sht45.cpp
18     i2c.cpp
19 )
20 target_include_directories(sht45_lib PUBLIC include)
21 target_compile_options(sht45_lib PRIVATE -Wall -Wextra -O2)
22
23 # Create utilities library (Logger, Config, Filter)
24 add_library(utils_lib STATIC
25     logger.cpp
26     config.cpp
27     filter.cpp
28 )
29 target_include_directories(utils_lib PUBLIC include)
30 target_compile_options(utils_lib PRIVATE -Wall -Wextra -O2)
31
32 # Create main executable
33 add_executable(sensor_app main.cpp)
34
35 # Link libraries to executable
36 target_link_libraries(sensor_app
37     PRIVATE
38         sht45_lib
39         utils_lib
40 )
41
42 # Compile options for the executable
43 target_compile_options(sensor_app PRIVATE -Wall -Wextra -O2)
```

CMake V2 stage4

BiggerLibraryCMakeV2:

External Dependencies
(Automatic Dependency Management)

```
16 # External dependencies with FetchContent
17 include(FetchContent)
18
19 # Add fmt library for better output formatting
20 FetchContent_Declare(
21     fmt
22     GIT_REPOSITORY https://github.com/fmtlib/fmt.git
23     GIT_TAG 10.1.1
24 )
25 FetchContent_MakeAvailable(fmt)
```

CMake V2 stage5

BiggerLibraryCMakeV2:

More advanced options including conditions

```
# Build type with sensible default
if(NOT CMAKE_BUILD_TYPE)
    set(CMAKE_BUILD_TYPE Release)
    message(STATUS "Build type not specified, defaulting to Release")
endif()
```

```
# Compiler-specific warnings
if(CMAKE_CXX_COMPILER_ID MATCHES "GNU|Clang")
    add_compile_options(-Wall -Wextra -Wpedantic)
elseif(CMAKE_CXX_COMPILER_ID MATCHES "MSVC")
    add_compile_options(/W4)
endif()
```

CMake V2 stage5

BiggerLibraryCMakeV2:

More advanced options including conditions

```
37 # Project options (user-configurable)
38 option(BUILD_TESTS "Build unit tests" OFF)
39 option(USE_FMT "Use fmt library for formatting" ON)
40 option(ENABLE_INSTALL "Enable installation rules" ON)
```

```
42 # External dependencies (conditional)
43 if(USE_FMT)
44     include(FetchContent)
45     message(STATUS "Fetching fmt library...")
46     FetchContent_Declare(
47         fmt
48         GIT_REPOSITORY https://github.com/fmtlib/fmt.git
49         GIT_TAG 10.1.1
50     )
51     FetchContent_MakeAvailable(fmt)
52 endif()
```

CMake V2 stage5

BiggerLibraryCMakeV2:

More advanced options including conditions

```
37 # Project options (user-configurable)
38 option(BUILD_TESTS "Build unit tests" OFF)
39 option(USE_FMT "Use fmt library for formatting" ON)
40 option(ENABLE_INSTALL "Enable installation rules" ON)
```

```
42 # External dependencies (conditional)
43 if(USE_FMT)
44     include(FetchContent)
45     message(STATUS "Fetching fmt library...")
46     FetchContent_Declare(
47         fmt
48         GIT_REPOSITORY https://github.com/fmtlib/fmt.git
49         GIT_TAG 10.1.1
50     )
51     FetchContent_MakeAvailable(fmt)
52 endif()
```

```
97 # Link fmt if enabled
98 if(USE_FMT)
99     target_link_libraries(utils PUBLIC fmt::fmt)
100    target_compile_definitions(utils PUBLIC USE_FMT)
101 endif()
```

BiggerLibraryCMakeV2

CMake V2 stage5

BiggerLibraryCMakeV2:

Useful for e.g. CI/CD:

```
121 #
122 # Installation rules
123 #
124 if(ENABLE_INSTALL)
125     include(GNUInstallDirs)
126 
127     # Install executable
128     install(TARGETS ${PROJECT_NAME}
129         RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
130         COMPONENT Runtime
131     )
132 
133     # Install libraries
134     install(TARGETS sht45 utils
135         ARCHIVE DESTINATION ${CMAKE_INSTALL_LIBDIR}
136         LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
137         COMPONENT Development
138     )
139 
140     # Install headers
141     install(DIRECTORY include/
142         DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}/${PROJECT_NAME}
143         COMPONENT Development
144         FILES_MATCHING PATTERN "*.hpp"
145     )
146 
147     # Install README and LICENSE (if they exist)
148     install(FILES
149         ${CMAKE_CURRENT_SOURCE_DIR}/README.md
150         DESTINATION ${CMAKE_INSTALL_DOCDIR}
151         COMPONENT Documentation
152         OPTIONAL
153     )
154 endif()
```

Up-to-date CMake

CMake V2 stage5

BiggerLibraryCMakeV2:

Build config summary:

```
165 #
166 # Build configuration summary
167 #
168 message(STATUS "")
169 message(STATUS "================================")
170 message(STATUS "${PROJECT_NAME} v${PROJECT_VERSION}")
171 message(STATUS "================================")
172 message(STATUS "Build type:        ${CMAKE_BUILD_TYPE}")
173 message(STATUS "C++ Standard:      C++${CMAKE_CXX_STANDARD}")
174 message(STATUS "Compiler:          ${CMAKE_CXX_COMPILER_ID} ${CMAKE_CXX_COMPILER_VERSION}")
175 message(STATUS "Build tests:       ${BUILD_TESTS}")
176 message(STATUS "Use fmt:           ${USE_FMT}")
177 message(STATUS "Enable install:    ${ENABLE_INSTALL}")
178 message(STATUS "Install prefix:    ${CMAKE_INSTALL_PREFIX}")
179 message(STATUS "================================")
180 message(STATUS "")
```

CMake V2 stage5

BiggerLibraryCMakeV2:

More on this next week:

```
156 #
157 # Testing support
158 #
159 if(BUILD_TESTS)
160     enable_testing()
161     message(STATUS "Building with tests enabled")
162     # add_subdirectory(tests)  # Uncomment when tests exist
163 endif()
164
```

Workshop Topics for Today

- Overcoming the Build Nightmare
- Cmake ins- and outs
- **Cmake for Embedded targets**
- Alternatives to Make

CMake V2 stage6

BiggerLibraryCMakeV2:

arm-toolchain.cmake

```
# arm-toolchain.cmake
# ARM GCC Toolchain for STM32 Cross-Compilation
# CMake Workshop - HAN University of Applied Sciences

# Tell CMake we're cross-compiling (not building for host system)
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_SYSTEM_PROCESSOR arm)

# Specify the cross compiler
# These tools are from ARM GCC toolchain (arm-none-eabi-gcc)
set(CMAKE_C_COMPILER arm-none-eabi-gcc)
set(CMAKE_CXX_COMPILER arm-none-eabi-g++)
set(CMAKE_ASM_COMPILER arm-none-eabi-gcc)
set(CMAKE_OBJCOPY arm-none-eabi-objcopy)
set(CMAKE_OBJDUMP arm-none-eabi-objdump)
set(CMAKE_SIZE arm-none-eabi-size)
set(CMAKE_DEBUGGER arm-none-eabi-gdb)

# Don't try to link during CMake configuration
# (embedded targets need special linker scripts)
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

# Search for programs only in the build host directories
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)

# Search for libraries and headers only in the target directories
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)

```

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
# CMakeLists.txt - Stage 6: Real-World STM32 Project
# CMake Workshop - HAN University of Applied Sciences
# Johan Korten - v1.0 2025
#
# STAGE 6: Real Embedded Example (STM32 ARM Cortex-M)
# Shows how workshop concepts apply to actual embedded development

cmake_minimum_required(VERSION 3.20)

# CRITICAL: Toolchain file MUST be set before project()
# This tells CMake we're cross-compiling for ARM, not the host system
set(CMAKE_TOOLCHAIN_FILE ${CMAKE_CURRENT_SOURCE_DIR}/arm-toolchain.cmake)

# Project definition
project(STM32_SHT45
    VERSION 1.0.0
    DESCRIPTION "SHT45 Sensor on STM32F4"
    HOMEPAGE_URL "https://github.com/AEAEmbedded/ESE_PROG"
    LANGUAGES C CXX ASM  # ASM needed for startup code!
)

#
# MCU-Specific Settings
#
set(MCU_FAMILY STM32F4xx)
set(MCU_MODEL STM32F411xE)
set(MCU_LINKER_SCRIPT ${CMAKE_CURRENT_SOURCE_DIR}/STM32F411RETx_FLASH.ld)

```

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
# ARM Cortex-M4 compiler flags
set(MCU_FLAGS
    -mcpu=cortex-m4        # Target CPU
    -mthumb                # Thumb instruction set
    -mfpu=fpv4-sp-d16      # Hardware FPU
    -mfloat-abi=hard       # Use hardware FP
)

# Common compile definitions
add_compile_definitions(
    ${MCU_MODEL}
    USE_HAL_DRIVER
    ARM_MATH_CM4
)

#
# STM32 HAL Library (Hardware Abstraction Layer)
#
add_library(stm32_hal STATIC
    # HAL Core
    Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal.c
    Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_cortex.c
    Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc.c
    Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_gpio.c

    # I2C for sensor communication
    Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_i2c.c

    # DMA for efficient transfers (optional)
    Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_dma.c

    # Add more HAL modules as needed:
    # Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_uart.c
    # Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_tim.c
)
```

BiggerLibraryCMakeV2

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
65 target_include_directories(stm32_hal PUBLIC
66     Drivers/STM32F4xx_HAL_Driver/Inc
67     Drivers/CMSIS/Device/ST/${MCU_FAMILY}/Include
68     Drivers/CMSIS/Include
69     Core/Inc  # User configuration headers
70 )
71
72 target_compile_options(stm32_hal PRIVATE ${MCU_FLAGS} -Wall -O2)
73
74 #
75 # Sensor Library (Same concepts from workshop!)
76 #
77 add_library(sht45 STATIC
78     src/sht45.cpp
79     src/i2c_wrapper.cpp  # Wraps STM32 HAL I2C functions
80 )
81
82 target_include_directories(sht45 PUBLIC
83     $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
84 )
85
86 # Link to HAL (sensor needs I2C)
87 target_link_libraries(sht45 PUBLIC stm32_hal)
88 target_compile_options(sht45 PRIVATE ${MCU_FLAGS} -Wall -Wextra)
89
90 #
91 # System Files
92 #
93 add_library(system OBJECT
94     Core/Src/system_stm32f4xx.c     # System clock configuration
95     Core/Src/stm32f4xx_it.c         # Interrupt handlers
96     startup_stm32f411xe.s            # Assembly startup code
97 )
```

BiggerLibraryCMakeV2

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
98
99  target_include_directories(system PUBLIC Core/Inc)
100 target_compile_options(system PRIVATE ${MCU_FLAGS})
101
102 #
103 # Main Firmware Executable
104 #
105 add_executable(${PROJECT_NAME}.elf
106     src/main.cpp
107     $<TARGET_OBJECTS:system>  # Include system files
108 )
109
110 # Link all libraries
111 target_link_libraries(${PROJECT_NAME}.elf
112     PRIVATE
113         sht45        # Our sensor library
114         stm32_hal    # STM32 HAL
115 )
116
117 target_include_directories(${PROJECT_NAME}.elf PRIVATE
118     Core/Inc
119     include
120 )
121
122 target_compile_options(${PROJECT_NAME}.elf PRIVATE
123     ${MCU_FLAGS}
124     -Wall
125     -Wextra
126     -fdata-sections    # Enable garbage collection
127     -ffunction-sections
128 )
```

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
#
# Linker Settings (Critical for embedded!)
#
target_link_options(${PROJECT_NAME}.elf PRIVATE
    -T${MCU_LINKER_SCRIPT}          # Memory layout
    -Wl,-Map=${PROJECT_NAME}.map    # Generate map file
    -Wl,--gc-sections               # Remove unused code
    -Wl,--print-memory-usage        # Show memory usage
    ${MCU_FLAGS}
    --specs=nano.specs              # Use smaller newlib-nano
    --specs=nosys.specs             # No system calls
)

#
# Post-Build: Generate Binary Files for Flashing
#
find_program(ARM_SIZE arm-none-eabi-size REQUIRED)
find_program(ARM_OBJCOPY arm-none-eabi-objcopy REQUIRED)

add_custom_command(TARGET ${PROJECT_NAME}.elf POST_BUILD
    # Generate Intel HEX format
    COMMAND ${ARM_OBJCOPY} -O ihex
        ${PROJECT_NAME}.elf
        ${PROJECT_NAME}.hex
    COMMENT "Generating HEX file..."

    # Generate binary format
    COMMAND ${ARM_OBJCOPY} -O binary
        ${PROJECT_NAME}.elf
        ${PROJECT_NAME}.bin
    COMMENT "Generating BIN file..."

    # Show size information
    COMMAND ${ARM_SIZE} --format=berkeley ${PROJECT_NAME}.elf
    COMMENT "Size analysis:"
)
```

BiggerLibraryCMakeV2

CMake V2 stage6

BiggerLibraryCMakeV2:

CMakeLists.txt for an embedded target:

```
167 #
168 # Flash Target (Optional: Program the MCU)
169 #
170 find_program(OPENOCD openocd)
171 if(OPENOCD)
172     add_custom_target(flash
173         COMMAND ${OPENOCD}
174             -f interface/stlink.cfg
175             -f target/stm32f4x.cfg
176             -c "program ${PROJECT_NAME}.elf verify reset exit"
177         DEPENDS ${PROJECT_NAME}.elf
178         COMMENT "Flashing ${PROJECT_NAME}.elf to STM32..."
179     )
180 endif()
181
182 #
183 # Debug Target (Optional: Start GDB server)
184 #
185 if(OPENOCD)
186     add_custom_target(debug
187         COMMAND ${OPENOCD}
188             -f interface/stlink.cfg
189             -f target/stm32f4x.cfg
190         COMMENT "Starting OpenOCD debug server..."
191     )
192 endif()
```

What if CMake gets confused

- cmake --build build --target clean

CMakeCache.txt / CMakeFiles/ any cached options you set before are kept.

If you then still run into problems: Pristine build

- rm -rf build
- cmake -S . -B build
- cmake --build build

Alternative to Make: Ninja

Ninja is a build system (like Make) but designed to be fast.

CMake can generate Ninja build files instead of Makefiles.

Alternative to Make: Ninja

Ninja is a build system (like Make) but designed to be fast.

CMake can generate Ninja build files instead of Makefiles.

cmake -B build_make time
cmake --build build_make

Make: 2.3 seconds

cmake -B build_ninja -G Ninja time

cmake --build build_ninja

Ninja: 0.8 seconds

Alternative to Make: Ninja

Why Ninja is Faster:

No shell overhead - Make spawns shells, Ninja doesn't

Parallel by default - Ninja automatically uses all cores

Minimal features - Ninja only builds, Make has a lot of often not used features

Better dependency tracking - Rebuilds only what's needed

Embedded projects with 100+ files compile in seconds, not minutes

Faster iteration = more learning

Industry uses Ninja (Google Chrome, Android, LLVM)

Alternative to Make: Ninja and others

For your embedded systems:

- Ninja - Fast, simple, used by modern CMake
- Meson - Great cross-compilation support
- Bazel - If working with larger embedded projects
- SCons/Waf - Common in embedded open source

The trend is toward:

- Faster incremental builds (Ninja, tup)
- Better dependency tracking (Bazel, Buck2)
- Cross-platform support (Meson, CMake+generators)
- Reproducible builds (Bazel, Nix-based systems)

“

That’s all…

Any questions?