



Leveraging AI to Accelerate Embedded Software Development

# DAY 3 : Optimizing your Build System with AI

Sponsored by



## 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.

## THE SPEAKER

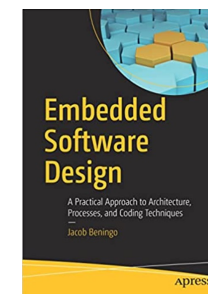
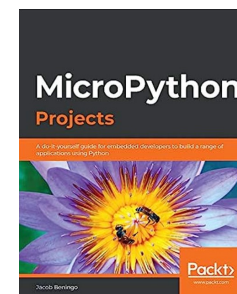
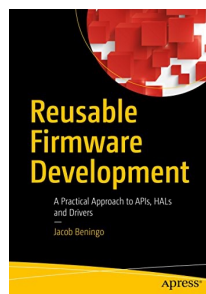


## Jacob Beningo

Jacob@beningo.com

## Beningo Embedded Group - President

Focus: Embedded Software Consulting and Training

Specializes 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](http://www.beningo.com) to learn more



01

# Improving your Makefiles

# Improving your makefiles

They're fine right? . . .

JA You

The following makefile is being used to compile my stm32 project. Can you review and provide any recommendations for improvements?

```
export
```

```
# Define what we are building  
TARGET = controller
```

ChatGPT

Your Makefile for the STM32 project looks well-structured and covers various aspects necessary for building firmware, like application code, drivers, libraries, and tests. Here are some recommendations and observations for potential improvements:

## 1. Consistency in Wildcard Usage:

- For `MCU_HEADERS`, you've used `$(shell find $(MCU_DIR) -type f -name '*.h*')`, which is a bit different from the other sections where `$(wildcard ...)` is used. While `find` is more powerful, consider consistency. If `find` is necessary due to directory depth, it's okay, but ensure it's consistently applied where needed.

## 2. Separation of Source and Header Files:

- You've effectively separated headers and sources, which is good practice. Ensure that this separation is maintained and clear in your documentation.

## 3. Use of Recursive Make:

- You're calling `$(MAKE)` recursively for `release`, `debug`, and `unit_tests`. This is a common pattern, but be aware that it can lead to complexities, especially with dependency tracking. Consider whether a non-recursive approach could be beneficial, although it may require a more complex initial setup.

## 4. Dependency Management:

- You have a `-include $(wildcard $(BUILD_DIR)/*.d)` at the end of dependencies. Make sure that your individual build rules generate these `.d` files appropriately for accurate dependency tracking.

# Improving your makefiles

Make an improvement

JA

You

Can you provide me with an update for the makefile to fix #1?

## Original Code:

```
# List of the microcontroller headers and sources. The MCU code should be
# hardware dependent and should be able to be used on-target.
# NOTE: The following looks messy because of how STM32CubeMx generates code
MCU_HEADERS := $(shell find $(MCU_DIR) -type f -name '*.h*')$(wildcard $(MCU_DIR)/*.h*)
MCU_SOURCES := $(wildcard $(MCU_DIR)/*.cpp)
MCU_C_SOURCES := $(wildcard $(MCU_DIR)/Core/Src/*.c)
MCU_C_SOURCES += $(filter-out %template.c, $(wildcard $(MCU_DIR)/Drivers/STM32L4xx_HAL_Driver/Src/*.c))
MCU_C_SOURCES += $(wildcard $(MCU_DIR)/Middlewares/Third_Party/FreeRTOS/Source/*.c)
MCU_C_SOURCES += $(wildcard $(MCU_DIR)/Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS_V2/*.c)
MCU_C_SOURCES += $(wildcard $(MCU_DIR)/Middlewares/Third_Party/FreeRTOS/Source/portable/MemMang/heap_4.c)
MCU_C_SOURCES += $(wildcard $(MCU_DIR)/Middlewares/Third_Party/FreeRTOS/Source/portable/GCC/ARM_CM4F/*.c)
MCU_ASSEMBLY := $(wildcard $(MCU_DIR)/*.[sS])
MCU_INCLUDES := $(patsubst %, -I%, $(sort $(dir $(MCU_HEADERS))))
```

## ChatGPT

Certainly! To address point #1, we will update the `MCU\_HEADERS` variable in your Makefile to use the `\$(wildcard ...)` pattern for consistency with the rest of your file. This change assumes that the directory structure of your MCU code is not too deep, allowing `\$(wildcard ...)` to capture all the necessary header files. If your directory structure is deeply nested, `\$(wildcard ...)` might not be sufficient, and you may need to revert to using `find` or another method.

Here's the updated section for `MCU\_HEADERS`:

makefile

[Copy code](#)

```
# List of the microcontroller headers. The MCU code should be
# hardware dependent and should be able to be used on-target.
MCU_HEADERS := $(wildcard $(MCU_DIR)/*.h)
MCU_HEADERS += $(wildcard $(MCU_DIR)/**/*.h)
MCU_INCLUDES := $(patsubst %, -I%, $(sort $(dir $(MCU_HEADERS))))
```

In this update, I've used `\$(wildcard \$(MCU\_DIR)/\*.h)` to include headers in the root of the `MCU\_DIR` directory, and `\$(wildcard \$(MCU\_DIR)/\*\*/\*.h)` to include headers in all subdirectories. This approach should work well if your headers are not nested too deeply. If your directory structure is more complex, you might need a more robust solution.

## Audience POLL Question

Have you asked an AI to review and improve makefiles before?

- a) Yes, it was very helpful
- b) Yes, it was not helpful
- c) No, but I may try it
- d) No, and no thanks



02

# AI-Generated Docker Container

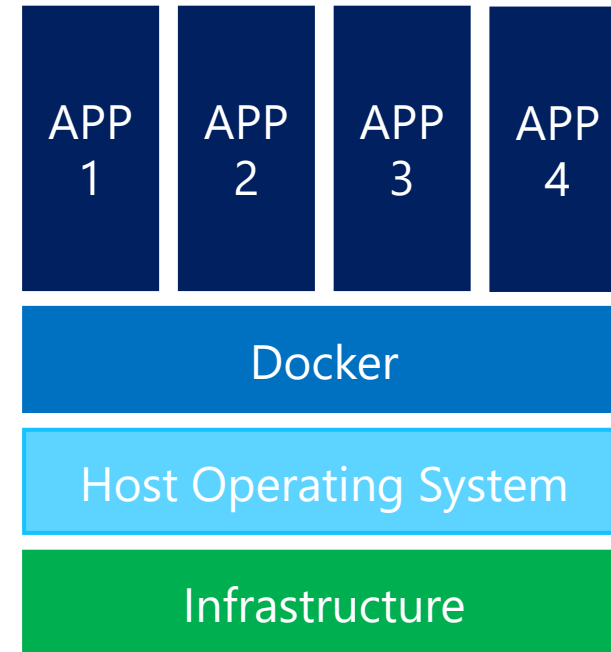


# AI-Generated Docker Files

## Dockers Container Definitions

A **container** is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.

A **Docker container** image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.



# AI-Generated Docker Files

## Docker Benefits

- 1. Consistent Environment:** Containers provide a consistent development environment across all stages of the embedded software lifecycle, from development to testing to deployment. This reduces the "it works on my machine" problem.
- 2. Isolation:** Containers isolate the build environment from the host system, ensuring that dependencies and configurations do not clash with the developer's environment or with other projects.
- 3. Dependency Control:** By using containers, you can precisely control and version the dependencies and tools required for your build, ensuring that every build uses the exact same set of tools and libraries.
- 4. Portability:** Since containers package an application with all of its dependencies, the containerized build system can easily be moved between different machines, different CI/CD systems, or even different cloud providers without needing any changes.
- 5. CI/CD Integration:** Containers integrate well with continuous integration and continuous deployment (CI/CD) pipelines, allowing you to create reproducible builds and automate the testing and deployment of embedded software.

# AI-Generated Docker Files

## Generate the default dockerfile

JA You

I want to create a dockerfile that can be used for embedded software development for an Arm Cortex®-M4 processor. The dockerfile will have the following requirements:

- Uses the latest version of ubuntu
- Installs basic linux support tools like update, clean, autoconf, curl, etc
- Sets up a tool directory in /home/dev
- Installs the gnu arm toolchain with C++ support for newlib
- Installs clang tools for formatting and static analysis

Can you generate the dockerfile that meets the above requirements?

```
# Use the latest Ubuntu image as the base
FROM ubuntu:latest

# Set the working directory
WORKDIR /home/dev

# Install basic Linux support tools
RUN apt-get update && apt-get install -y \
  {
    autoconf \
    curl \
    clang \
    clang-format \
    clang-tidy \
    && rm -rf /var/lib/apt/lists/*
  }

# Create a tools directory
RUN mkdir -p /home/dev/tools

# Set the path for the tools
ENV PATH="/home/dev/tools:${PATH}"

# Install GNU Arm toolchain with C++ support for newlib
RUN cd /home/dev/tools && \
  curl -LO https://developer.arm.com/-/media/Files/downloads/gnu-rm/1
  tar xjf gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2 &&
  rm gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2

# Add the GNU Arm toolchain to the PATH
ENV PATH="/home/dev/tools/gcc-arm-none-eabi-10-2020-q4-major/bin:${PATH}"

# Set the entry point to a shell
ENTRYPOINT ["/bin/bash"]
```

# AI-Generated Docker Files

Test the result

```
docker build -t beningo/cec-dev .
```

```
-----  
dockerfile:23
```

```
22 | # Install GNU Arm toolchain with C++ support for newlib  
23 | >>> RUN cd /home/dev/tools && \  
24 | >>> curl -LO https://developer.arm.com/-/media/Files/downloads/gnu-rm/10-2020-q4-major/gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2 && \  
25 | >>> tar xjf gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2 && \  
26 | >>> rm gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2  
27 |
```

```
-----  
ERROR: failed to solve: process "/bin/sh -c cd /home/dev/tools && curl -LO https://developer.arm.com/-/media/Files/downloads/gnu-rm/10-2020-q4-major/gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2 && tar xjf gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2 && rm gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2" did not complete successfully: exit code: 2
```

# AI-Generated Docker Files

## Give ChatGPT Feedback

JA

You

i get the following error:

dockerfile:23

```

-----
22 | # Install GNU Arm toolchain (Version 13.2.Rel1)
23 | >>> RUN cd /home/dev/tools && \
24 | >>> curl -LO https://developer.arm.com/-/media/Files/downloads/gnu-rm/13-
2023-q1-update/gcc-arm-none-eabi-13-2023-q1-update-x86_64-linux.tar.bz2 && \
25 | >>> tar xjf gcc-arm-none-eabi-13-2023-q1-update-x86_64-linux.tar.bz2 && \
26 | >>> rm gcc-arm-none-eabi-13-2023-q1-update-x86_64-linux.tar.bz2
27 |
-----

```

```

ERROR: failed to solve: process "/bin/sh -c cd /home/dev/tools && curl -LO
https://developer.arm.com/-/media/Files/downloads/gnu-rm/13-2023-q1-update/gcc-
arm-none-eabi-13-2023-q1-update-x86_64-linux.tar.bz2 && tar xjf gcc-arm-none-
eabi-13-2023-q1-update-x86_64-linux.tar.bz2 && rm gcc-arm-none-eabi-13-2023-
q1-update-x86_64-linux.tar.bz2" did not complete successfully: exit code: 2

```

```

# Use the latest Ubuntu image as the base
FROM ubuntu:latest

# Set the working directory
WORKDIR /home/dev

# Install basic Linux support tools and xz-utils for .xz file support
RUN apt-get update && apt-get install -y \
    autoconf \
    curl \
    clang \
    clang-format \
    clang-tidy \
    xz-utils \
    && rm -rf /var/lib/apt/lists/*

# Create a tools directory
RUN mkdir -p /home/dev/tools

# Install GNU Arm toolchain (Version 13.2.Rel1)
RUN cd /home/dev/tools && \
    curl -LO https://developer.arm.com/-/media/Files/downloads/gnu/13.2
    tar xf arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-eabi.tar.xz && \
    rm arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-eabi.tar.xz

# Add the GNU Arm toolchain to the PATH
# Update the PATH line below according to the actual structure of the e
ENV PATH="/home/dev/tools/arm-gnu-toolchain-13.2.rel1/bin:${PATH}"

# Set the entry point to a shell
ENTRYPOINT ["/bin/bash"]

```

## Audience POLL Question

Do you currently use docker to package your build environment?

- a) Yes, I love it
- b) Yes, I hate it
- c) No, but it's an interesting idea
- d) No, not really for me

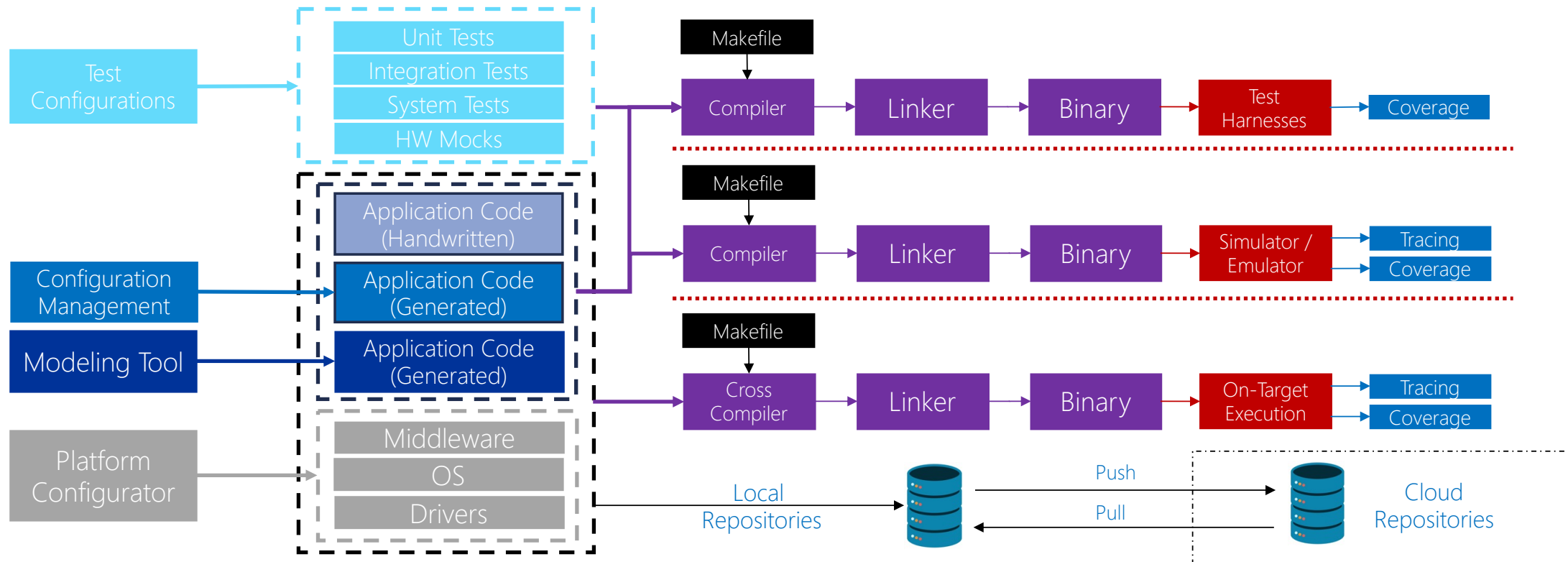


03

# Generating a Modern Build Environment

# Generating a Modern Build Environment

Define your ideal build system





# Generating a Modern Build Environment

## Define Your Project Structure

For a build system, you need your code structure:

- Ask the tool to generate an example
- Modify the example with your details
- Feed the structure into the tool

Several additional suggestions:

- Ask for feedback
- Make any modifications
- Continue with build system generation

```
YourEmbeddedProjectName
|-- /docs
|   |-- /standards # Project documentation, datasheets, and notes.
|   |   |-- /standards # Coding style and industry standards followed in this project.
|   |   |   |-- cstyle.md # Example coding style document in markdown
|   |-- /datasheets # Microcontroller and peripheral datasheets.
|   |-- /design_notes # Design decisions, rationale, etc.
|   |-- /doxygen # Doxygen generated documentation.
|-- /firmware # Firmware code directory.
|   |-- /app # Application-specific source and header files.
|   |   |-- main.c
|   |   |-- /tasks # Application tasks or threads.
|   |   |-- /config # Configuration files (e.g., system_config.h).
|   |   # Other potential application-specific folders could be added here.
|   |-- /boot # bootloader project. Application-specific source and header files.
|   |   |-- main.c
|   |   |-- /config # Configuration files
|   |   # Other potential bootloader specific folders could be added here.
|   |-- /bsp # Board Support Package - low-level drivers.
|   |   |-- /cfg # Config files for the bsp devices
|   |   |-- /devices # Other potential bsp specific folders could be added here.
|   |-- /hal # Hardware Abstraction Layer.
|   |   |-- /inc # Header files for HAL.
|   |   |-- /src # Source files for HAL.
|   |   |-- /cfg # Config files for HAL.
|   |-- /drivers # Device drivers for peripherals (e.g., SPI, UART).
|   |   |-- /devices # Header and source files for drivers.
|   |   |-- /cfg # Config files for drivers.
|   |-- /lib # Libraries and middleware (e.g., FreeRTOS, communication protocols).
|   |   |-- /stm32 # Example mcu device folder
|   |   |-- /cmsis # Example Arm CMSIS support
|   |   |-- /trace # Example Percepio trace recorder library
|   |   |-- /freertos # Example FreeRTOS folder for device target
|   |   |-- /linux # Example FreeRTOS folder for linux
|   |   |-- /win32 # Example FreeRTOS folder for Win32
|   |-- /utils # Utilities, helpers, and service functions.
|   |-- /test # Unit tests, mocks, and testing scripts.
|   |-- /ld # Linker scripts.
|   |   |-- linker.ld
|-- /hw # Hardware-related files (like PCB design).
|   |-- /schematics # Schematic design files.
|   |-- /layouts # PCB layout files.
|-- /tools # Build tools, scripts, and utilities.
|-- /output # Compiled binaries, hex files, etc.
|-- Makefile # Or CMakeLists.txt, depending on the build system.
```

# Generating a Modern Build Environment

## Create a Makefile

JA

You

Can you create a makefile that has a debug target to build an embedded application with the following directory structure?

```
/YourEmbeddedProjectName
|-- /docs          # Project documentation, datasheets, and notes.
| |-- /standards  # Coding style and industry standards followed in this project.
|   |--cstyle.md  # Example coding style document in markdown
| |-- /datasheets # Microcontroller and peripheral datasheets.
| |-- /design_notes # Design decisions, rationale, etc.
| '-- /doxygen    # Doxygen generated documentation.
|
|-- /firmware     # Firmware code directory.
| |-- /app        # Application-specific source and header files.
| | |-- main.c
| | |-- /tasks    # Application tasks or threads.
| | `-- /config   # Configuration files (e.g., system_config.h).
| | # Other potential application-specific folders could be added here.
```

```
# Project name
PROJ_NAME = YourEmbeddedProjectName

# Directories
FIRMWARE_DIR = firmware
APP_DIR = $(FIRMWARE_DIR)/app
BSP_DIR = $(FIRMWARE_DIR)/bsp
HAL_DIR = $(FIRMWARE_DIR)/hal
DRIVERS_DIR = $(FIRMWARE_DIR)/drivers
LIB_DIR = $(FIRMWARE_DIR)/lib
UTILS_DIR = $(FIRMWARE_DIR)/utils
OUTPUT_DIR = output
LD_DIR = $(FIRMWARE_DIR)/ld

# Compiler and flags
CC = arm-none-eabi-gcc
DEBUG_FLAGS = -g -O0
CFLAGS = $(DEBUG_FLAGS) -I$(APP_DIR) -I$(BSP_DIR) -I$(HAL_DIR)/inc -I$(
LDFLAGS = -T$(LD_DIR)/linker.ld

# Source files
APP_SRC = $(wildcard $(APP_DIR)/*.c)
BSP_SRC = $(wildcard $(BSP_DIR)/*.c)
HAL_SRC = $(wildcard $(HAL_DIR)/src/*.c)
DRIVERS_SRC = $(wildcard $(DRIVERS_DIR)/devices/*.c)
UTILS_SRC = $(wildcard $(UTILS_DIR)/*.c)

# Object files
OBJS = $(patsubst %.c,%.o,$(APP_SRC) $(BSP_SRC) $(HAL_SRC) $(DRIVERS_SRC)

# Debug target
debug: $(OUTPUT_DIR)/$(PROJ_NAME).elf
$(OUTPUT_DIR)/$(PROJ_NAME).elf: $(OBJS)
```

## Audience POLL Question

Are you going to use AI to improve how you build your code?

- a) Yes, I already am
- b) Yes, in the near future
- c) Maybe
- d) No



# Next Steps

04

## Additional Resources

Please consider the resources below:

- [Jacob's AI Blogs](#)
- [Jacob's CEC courses](#)
- [Jacob's ML Blogs](#)
- Embedded Bytes Newsletter
  - <http://bit.ly/1BAHYXm>

[www.beningo.com](http://www.beningo.com)



Consulting

Coaching

Training

## Next Steps

- ✓ The Rise of AI in Embedded Software
- ✓ Writing Better Prompts for Code Generation
- ✓ Optimizing your Build System with AI
- Abstracting your Hardware with an AI-Generated HAL
- Managing AI and ML Code



Thank You

Sponsored by

