Embedded System Project Guideline — Extended Edition

# Embedded System Project Guideline — Comprehensive Reference

This document is a \*\*deeply structured, modular, and highly detailed guideline\*\* for embedded C projects targeting MCU platforms. It combines real code, callback logic, documentation tools, and best practices across hardware abstraction, driver design, application logic, build systems, and future RTOS expansion.

## 1. Introduction & Objectives

\*\*Purpose:\*\* To provide a generic, scalable, and reusable firmware structure tailored for embedded microcontrollers (MCUs).

\*\*Objectives:\*\*

* - Structure firmware into hardware abstraction (HAL), device-level drivers, and application layers.
* - Use consistent naming, MISRA-C/Barr-C-compliant patterns, and modular design.
* - Facilitate integration of RTOS, unit testing, CI/CD, and static analysis.
* - Enable clean documentation through Doxygen and UML tooling.

## 2. Project Structure Overview

embedded\_system\_project/

├── CMakeLists.txt

├── main.c # Central app logic

├── include/ # Global typedefs, constants

│ └── global\_defs.h

├── hal/

│ ├── include/hal\_gpio.h # GPIO abstraction

│ └── src/hal\_gpio.c

├── device/button/

│ ├── include/button.h # Button API and types

│ └── src/button.c # Button logic

├── docs/ # UML, Doxygen

├── tests/ # Unit tests with mocks

└── build/ # Generated output

## 3. Layered Architecture

### 3.1 HAL — Hardware Abstraction Layer

* - hal\_gpio\_config\_input(uint8\_t pin)
* - hal\_gpio\_read(uint8\_t pin)

### 3.2 Device Layer — Button Driver

* - \*\*button.h\*\* defines:

typedef enum {

BUTTON\_EVENT\_NONE,

BUTTON\_EVENT\_PRESSED,

BUTTON\_EVENT\_RELEASED

} button\_event\_e;

typedef void (\*button\_callback\_t)(button\_event\_e event);

* - \*\*button.c\*\*:

bool button\_init(button\_handle\_t \*handle, const button\_config\_t \*cfg);

void button\_task(button\_handle\_t \*handle);

### 3.3 Application Layer — main.c

button\_config\_t cfg = {

.gpio\_pin = 0,

.active\_low = true,

.callback = button\_event\_handler

};

Simulates button polling and invokes:

void button\_event\_handler(button\_event\_e event);

## 4. Callback Mechanism Explained

A \*\*callback\*\* is a user-provided function pointer passed to the driver. It is triggered \*\*only\*\* when a relevant state transition is detected by polling logic.

### Example:

void button\_event\_handler(button\_event\_e event) {

switch (event) {

case BUTTON\_EVENT\_PRESSED:

printf("Button Pressed\n");

break;

case BUTTON\_EVENT\_RELEASED:

printf("Button Released\n");

break;

default:

break;

}

}

This pattern promotes \*\*decoupling\*\* and allows reuse of the driver in any context.

## 5. Source Code Analysis

### main.c

* - Simulates GPIO using a variable (`fake\_button\_pressed`)
* - Calls `button\_task()` periodically.

### button.c

if (curr\_state != h->last\_state) {

h->last\_state = curr\_state;

if (h->config.callback != NULL) {

h->config.callback(event);

}

}

### hal\_gpio.c

* - Stub for hardware abstraction. Later replaced with MCU registers.

## 6. Build System — CMake

include\_directories(

include

hal/include

device/button/include

)

file(GLOB BUTTON\_SRC device/button/src/\*.c)

add\_executable(embedded\_app main.c ${BUTTON\_SRC})

### Outputs

* - ELF → embedded\_app
* - HEX → embedded\_app.hex
* - BIN → embedded\_app.bin

## 7. Documentation Strategy

### Doxygen

EXTRACT\_ALL = YES

INPUT = include hal/include device/button/include

FILE\_PATTERNS = \*.h

GENERATE\_LATEX = YES

### UML

* - Sequence: main() → button\_task() → callback
* - Class: button\_handle\_t and dependencies

## 8. Testing & CI/CD

### Testing

* - Use Unity + CMock
* - Mock hal\_gpio\_read() for button logic

### GitHub Actions

* - build.yml: builds via CMake
* - lint.yml: runs cppcheck or clang-tidy
* - doc.yml: generates Doxygen site

## 9. RTOS & HAL Extension (Planned)

### RTOS Integration

* - button\_task() → button\_thread()
* - Add semaphore or queue to push events
* - ISR-safe event registration

### HAL Abstractions

* - GPIO, EXTI, I2C
* - Board-level hal\_board\_init()

### Future APIs

void button\_register\_callback(button\_handle\_t \*h, button\_callback\_t cb);

### MISRA Support

* - Enable clang-tidy checks
* - Maintain MISRA\_DEVIATIONS.md

## END OF GUIDELINE — Extended Edition