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