Task 1: Firmware Foundations & Environment Setup

1. Firmware Library

A firmware library is a set of pre-developed, reusable software routines stored in non-volatile memory such as ROM or flash, tailored for microcontrollers and embedded systems. These libraries manage essential operations like peripheral control, interrupt handling, and communication interfaces (e.g., UART, I2C, SPI, GPIO), shielding developers from hardware details so they don't have to constantly implement low-level code from scratch.

By offering validated and optimized components, firmware libraries improve code portability across related hardware platforms and accelerate development, helping to minimize defects and shorten certification efforts in safety-critical environments.

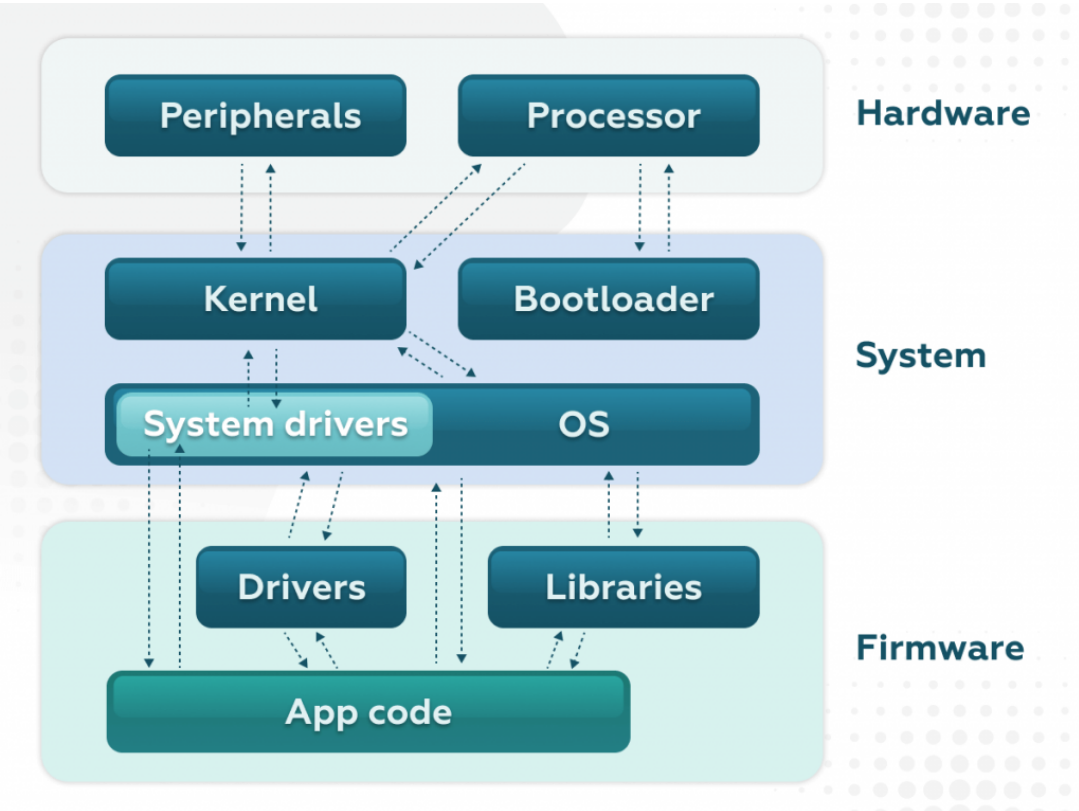

Figure 1: Layered Model showing the relationship between App code,Firmware Libraries and Syste, Hardware(GPIO,I2C,SPI)

2. Importance of APIs in Embedded systems

APIs, or Application Programming Interfaces, play a vital role in embedded systems because they define standardized function calls and data structures that let application code communicate with hardware, drivers, or operating system services without revealing low-level implementation details. This layer of abstraction streamlines development on resource-constrained microcontrollers, supporting modular design in which higher-level logic can invoke simple functions like sensor_read() rather than directly manipulating hardware registers.

If APIs were absent, embedded software would be fragile and tightly bound to specific hardware platforms. By contrast, relying on APIs improves maintainability, speeds up debugging, and promotes interoperability across ecosystems such as wireless sensor networks and wearable devices.

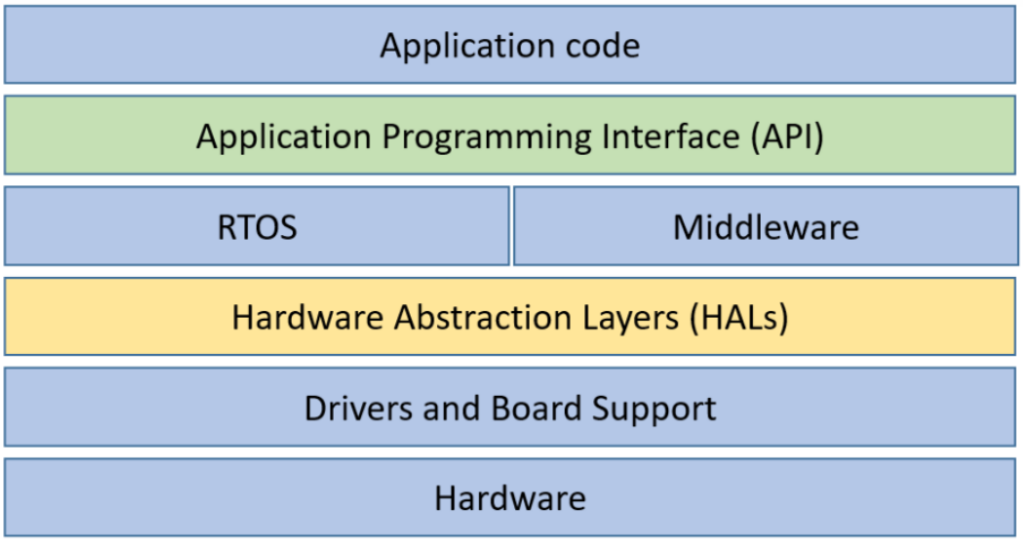

Figure 2: The role of APIs in connecting application logic to physical devices

3. Lab Code Analysis

By working through this task, you can observe how embedded software interfaces with hardware through a GPIO library while adhering to a professional Abstraction Layer model. The codebase is structured into three distinct files: 1. Header file (gpio.h), 2. Source file (gpio.c), 3. Main application file (main.c)

This program uses a standardized API (gpio_init(),gpio_write(),gpio_read()) to interact with hardware in a simple and structured way. The application code (main.c) works at a high level without directly accessing hardware registers, improving abstraction and readability.

By separating hardware-dependent code into gpio.c, the program becomes portable—only the GPIO layer needs modification if the processor changes.

The software follows a layered architecture:

1. Application layer (main.c)
2. Firmware/API layer (gpio.h, gpio.c)
3. Hardware layer (simulated GPIO)

During execution, the program configures the pins, turns the LED ON, reads the button state, then turns the LED OFF and terminates.

4. Build and Output Screenshots

The firmware code was compiled and executed using GCC in the MSYS2 MINGW64 environment.

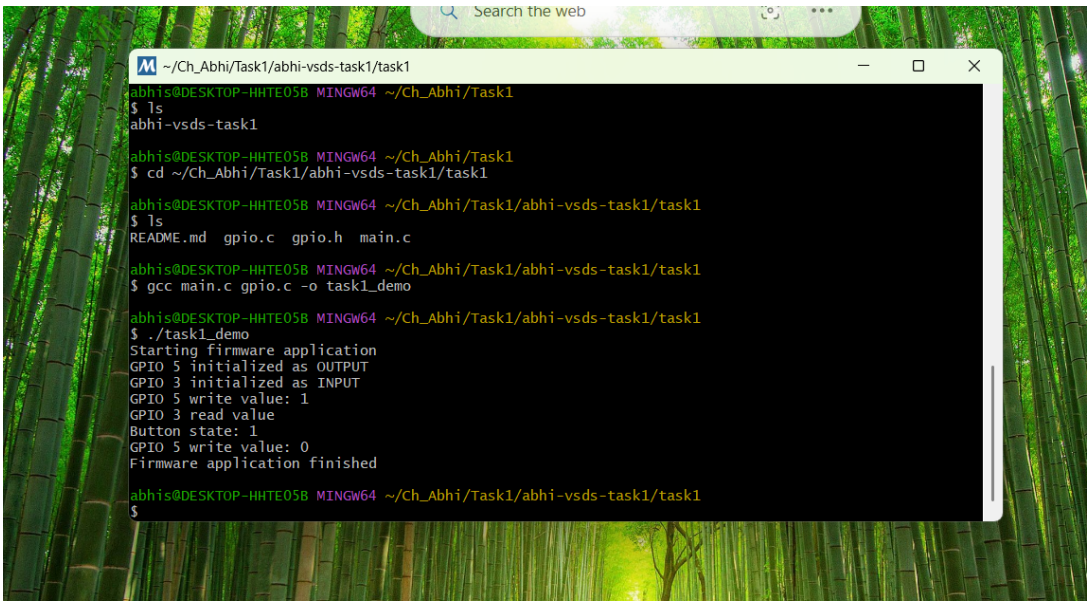

Figure 3: Execution output of the firmware program