Fingerprint Module Library

- Core Functionality: Fingerprint enrollment, matching, and template storage using an event-driven architecture

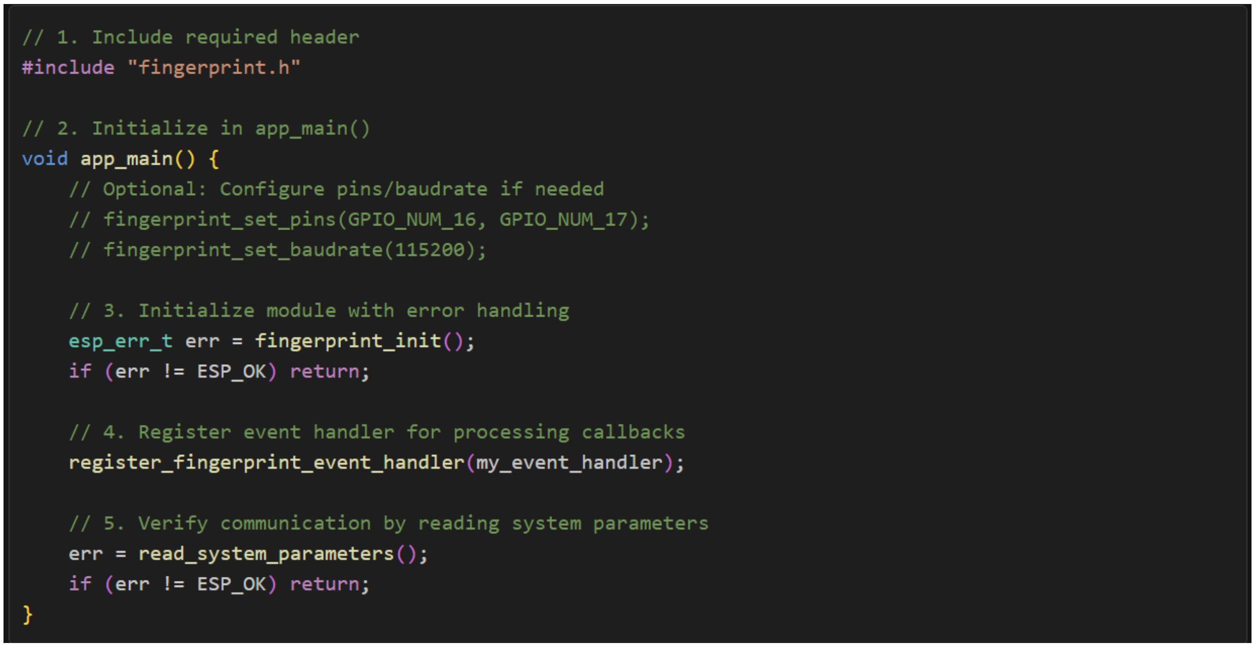
- Hardware Requirements:
  - UART communication at 57600 bps (adjustable 9600-115200 bps)
  - Pin connections: TX→GPIO6, RX→GPIO5, INT→GPIO15, VIN→GPIO9
  - 3.3V power supply

- Software Requirements:
  - ESP-IDF v4.x+ framework
  - FreeRTOS for asynchronous operations
  - Standard ESP32 toolchain

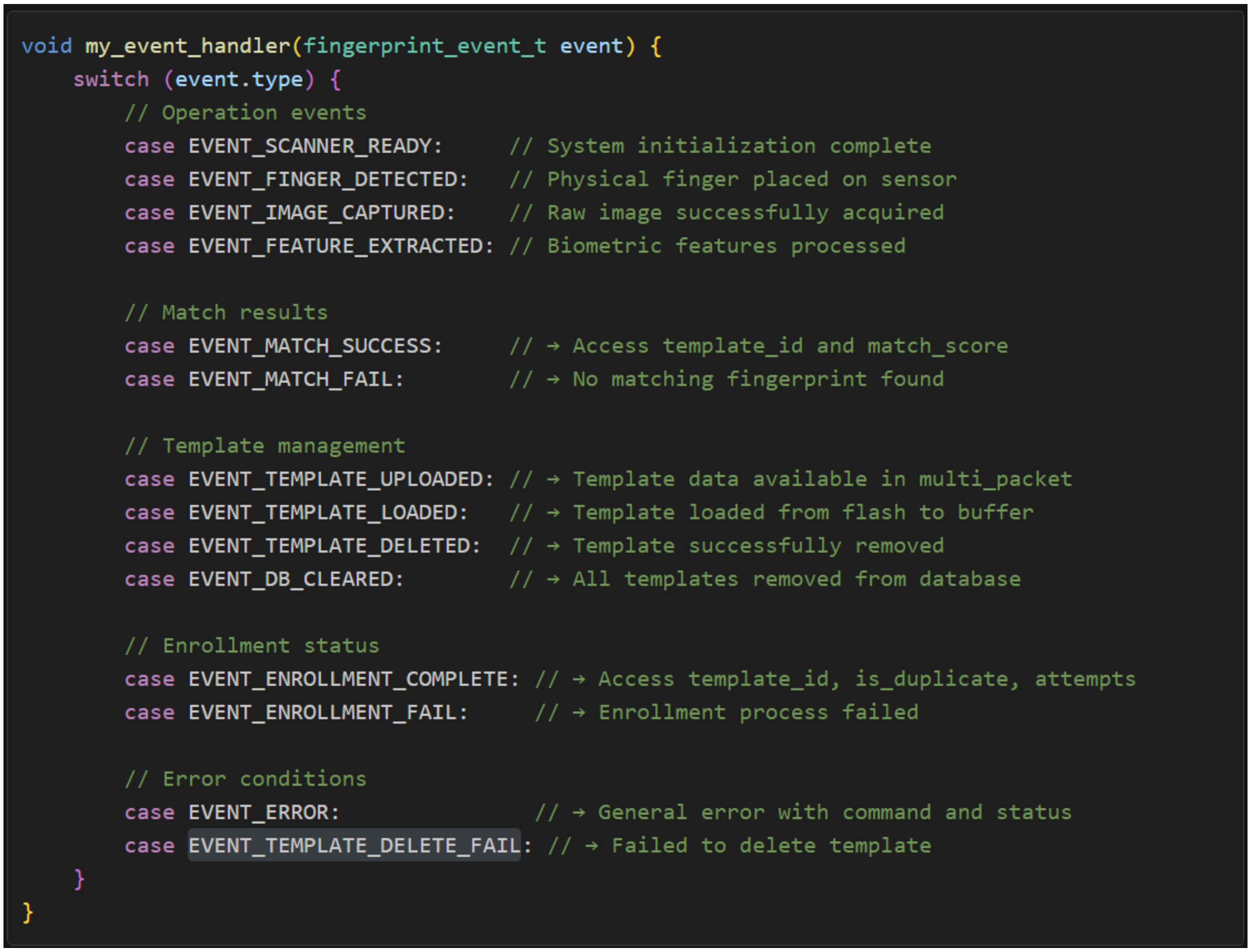
- Memory Requirements:
  - Buffer sizes (RX_BUF_SIZE = 1024)
  - Template buffer size (4096 bytes)
  - Task stack sizes (8192 bytes for read task, 4096 bytes for others)

- Core Operations:
  - Enroll: Use enroll_fingerprint(location) to add new fingerprints to a specific storage slot
  - Verify: Call verify_fingerprint() to match a scanned fingerprint against the database
  - get_enrolled_count() - Check how many templates are stored
  - delete_fingerprint(id) - Remove a specific template
  - clear_database() - Delete all templates
  - backup_template(id) - Export a template to ESP32 memory. Data is stored in event.multipacket->template_data and event.multipacket->packets
  - restore_template_from_multipacket(id, data) - Import a previously backed up template. Use event.multipacket as an argument in parameter data because this function will access the packets inside the multipacket.

- Software Initialization:

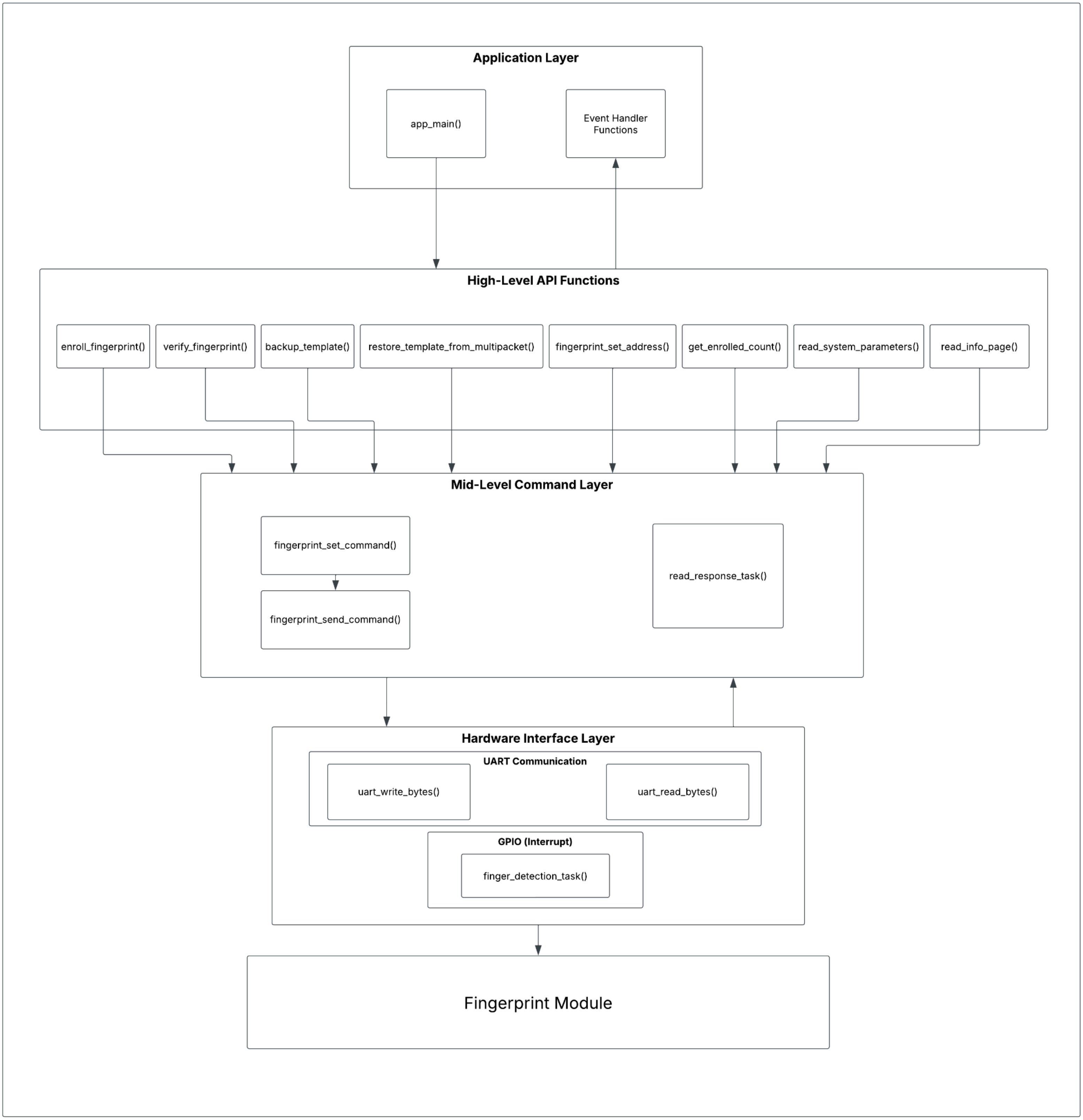


```
// 1. Include required header
#include "fingerprint.h"

// 2. Initialize in app_main()
void app_main() {
    // Optional: Configure pins/baudrate if needed
    // fingerprint_set_pins(GPIO_NUM_16, GPIO_NUM_17);
    // fingerprint_set_baudrate(115200);

    // 3. Initialize module with error handling
    esp_err_t err = fingerprint_init();
    if (err != ESP_OK) return;

    // 4. Register event handler for processing callbacks
    register_fingerprint_event_handler(my_event_handler);

    // 5. Verify communication by reading system parameters
    err = read_system_parameters();
    if (err != ESP_OK) return;
}
```



```
void my_event_handler(fingerprint_event_t event) {
    switch (event.type) {
        // Operation events
        case EVENT_SCANNER_READY:      // System initialization complete
        case EVENT_FINGER_DETECTED:    // Physical finger placed on sensor
        case EVENT_IMAGE_CAPTURED:     // Raw image successfully acquired
        case EVENT_FEATURE_EXTRACTED: // Biometric features processed

        // Match results
        case EVENT_MATCH_SUCCESS:      // → Access template_id and match_score
        case EVENT_MATCH_FAIL:         // → No matching fingerprint found

        // Template management
        case EVENT_TEMPLATE_UPLOADED: // → Template data available in multi_packet
        case EVENT_TEMPLATE_LOADED:    // → Template loaded from flash to buffer
        case EVENT_TEMPLATE_DELETED:   // → Template successfully removed
        case EVENT_DB_CLEARED:         // → All templates removed from database

        // Enrollment status
        case EVENT_ENROLLMENT_COMPLETE: // → Access template_id, is_duplicate, attempts
        case EVENT_ENROLLMENT_FAIL:     // → Enrollment process failed

        // Error conditions
        case EVENT_ERROR:               // → General error with command and status
        case EVENT_TEMPLATE_DELETE_FAIL: // → Failed to delete template
    }
}
```

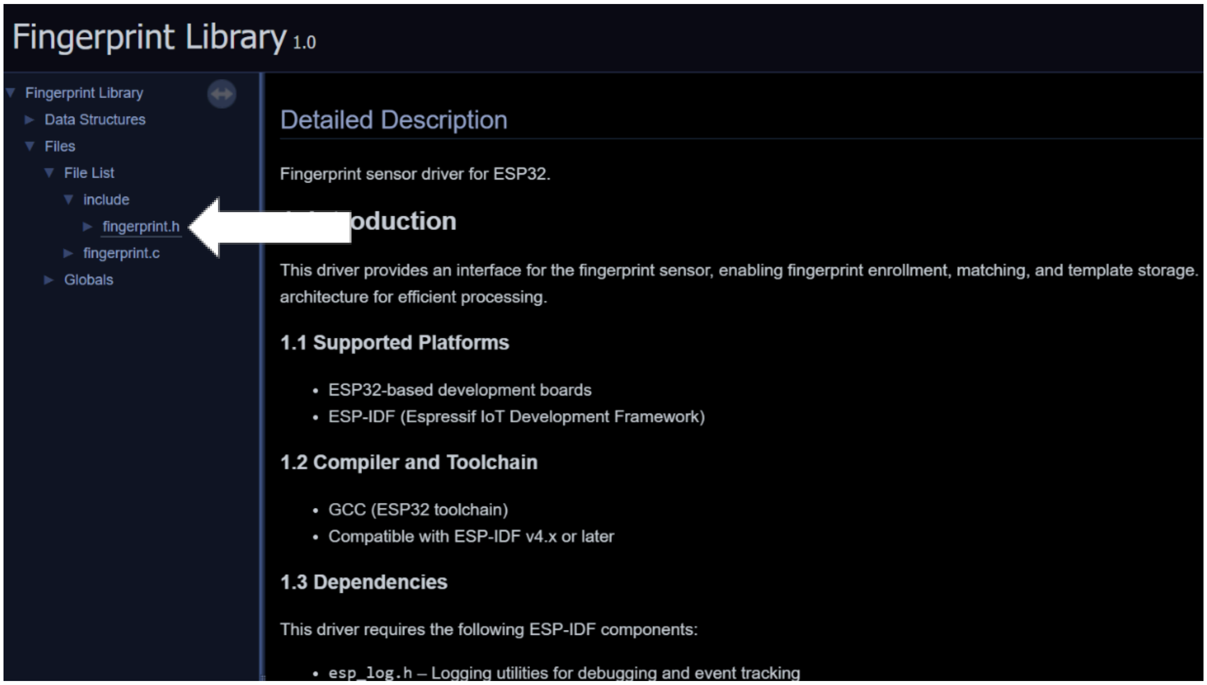
- Diagram:


Application Layer
app_main()
Event Handler Functions
High-Level API Functions
enroll_fingerprint()
verify_fingerprint()
backup_template()
restore_template_from_multipacket()
fingerprint_set_address()
get_enrolled_count()
read_system_parameters()
read_info_page()
Mid-Level Command Layer
fingerprint_set_command()
fingerprint_send_command()
read_response_task()
Hardware Interface Layer
UART Communication
uart_write_bytes()
uart_read_bytes()
GPIO (Interrupt)
finger_detection_task()
Fingerprint Module

Detailed documentation is provided called “fingerprint_documentation.html” in the same directory.

After clicking the .html file find the fingerprint.h:

Fingerprint Library→Files→File List→include→fingerprint.h



Detailed Description

Fingerprint sensor driver for ESP32.

oduction

This driver provides an interface for the fingerprint sensor, enabling fingerprint enrollment, matching, and template storage. architecture for efficient processing.

1.1 Supported Platforms

- ESP32-based development boards
- ESP-IDF (Espressif IoT Development Framework)

1.2 Compiler and Toolchain

- GCC (ESP32 toolchain)
- Compatible with ESP-IDF v4.x or later

1.3 Dependencies

This driver requires the following ESP-IDF components:

- `esp_log.h` – Logging utilities for debugging and event tracking