# ESP32 Firmware Developer Guide

**Understanding the PAROL6 ESP32 Communication Firmware**

This guide helps you understand the ESP32 firmware code and how to modify it for motor control.

---

## 📚 Table of Contents

1. ESP-IDF Project Structure
2. Code Walkthrough
3. Key Concepts
4. Adapting for Motor Control
5. ESP-IDF Patterns & Best Practices
6. Debugging Tips

---

## 📁 ESP-IDF Project Structure

```
esp32_benchmark_idf/
├── CMakeLists.txt              ← Root project config
├── main/
│   ├── CMakeLists.txt          ← Main component config
│   └── benchmark_main.c        ← Our firmware code
├── build/                      ← Build output (auto-generated)
├── sdkconfig                   ← ESP-IDF configuration
└── flash.sh                    ← Build & flash script
```

### Understanding CMakeLists.txt

**Root `CMakeLists.txt`:**

```
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(esp32_benchmark)
```

- Sets minimum CMake version
- Includes ESP-IDF build system
- Names the project

**`main/CMakeLists.txt`:**

```
idf_component_register(SRCS "benchmark_main.c"
                       INCLUDE_DIRS ".")
```

- Registers our C file as a component
- Tells ESP-IDF what to compile

---

## 🔍 Code Walkthrough

Main File: `benchmark_main.c`

#### 1. Includes & Defines

```c
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/uart.h"
#include "esp_log.h"
#include "esp_timer.h"

#define UART_NUM UART_NUM_0         // Use USB serial port
#define BUF_SIZE 1024               // UART buffer size
#define TAG "BENCHMARK"             // Log tag for filtering
```

**Why these?**

- `freertos/*`: ESP32 runs FreeRTOS (real-time OS)
- `driver/uart.h`: Serial communication
- `esp_log.h`: Logging to serial monitor
- `esp_timer.h`: Microsecond-precision timing

#### 2. Global Variables

```c
static int received_count = 0;
static int lost_packets = 0;
static int expected_seq = 0;
```

**Purpose:**

- `received_count`: Total commands received
- `lost_packets`: Detected packet losses
- `expected_seq`: Next sequence number we expect

**Thread Safety Note:** These are simple counters accessed from one task, OK for now. For multi-threaded access, use mutexes.

#### 3. UART Configuration

```
void configure_uart(void) {
    uart_config_t uart_config = {
        .baud_rate = 115200,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    };
```

**Key Settings:**

- **115200 baud**: Industry standard for robotics
- **8N1**: 8 data bits, No parity, 1 stop bit
- **No flow control**: Simplicity over hardware handshaking

```
    ESP_ERROR_CHECK(uart_param_config(UART_NUM, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(UART_NUM, UART_PIN_NO_CHANGE,
                                 UART_PIN_NO_CHANGE,
                                 UART_PIN_NO_CHANGE,
                                 UART_PIN_NO_CHANGE));
```

`ESP_ERROR_CHECK` **macro:**

- Crashes the program if UART setup fails
- Good for catching config errors early

```
    ESP_ERROR_CHECK(uart_driver_install(UART_NUM, BUF_SIZE * 2,
                                        BUF_SIZE * 2, 0, NULL, 0));
```

**Driver Install:**

- RX buffer: 2048 bytes (can hold ~30 commands)
- TX buffer: 2048 bytes (for ACKs)
- No event queue (we poll instead)

#### 4. Message Parsing

```
int parse_message(const char* buffer, int* seq, float joints[6]) {
    // Expected format: <SEQ,J1,J2,J3,J4,J5,J6>

    if (buffer[0] != '<') {
        return 0;  // Invalid start
    }
```

**Why check first character?**

- Messages might arrive mid-stream
- Noise on serial line
- Partial messages

```
    // Find the closing '>'
    const char* end = strchr(buffer, '>');
    if (!end) {
        return 0;  // Incomplete message
    }
```

`strchr()` **function:**

- Searches for '>' character
- Returns NULL if not found
- Prevents parsing truncated messages

```
    int parsed = sscanf(buffer, "<%d,%f,%f,%f,%f,%f,%f>",
                        seq,
                        &joints[0], &joints[1], &joints[2],
                        &joints[3], &joints[4], &joints[5]);

    return (parsed == 7) ? 1 : 0;  // 1 seq + 6 joints = 7 values
}
```

`sscanf()` **pattern:**

- Parses formatted string into variables
- Returns number of successfully parsed values
- `%d` for integer (seq), `%f` for float (joints)

**5. Packet Loss Detection**

```
void handle_packet_loss(int actual_seq) {
    if (actual_seq != expected_seq) {
        int gap = actual_seq - expected_seq;
        if (gap > 0) {
            lost_packets += gap;
            ESP_LOGW(TAG, "PACKET LOSS: Expected %d, got %d (lost %d)",
                     expected_seq, actual_seq, gap);
        }
        expected_seq = actual_seq + 1;
    } else {
        expected_seq++;
    }
}
```

**How it works:**

1. Compare received seq with expected seq
2. If gap > 0: packets were lost
3. Update expected to actual + 1
4. Continue tracking

**Example:**

```
Received: 100, 101, 105  (102, 103, 104 lost!)
Expected: 100, 101, 102
Gap at 105: 105 - 102 = 3 packets lost
```

**6. ACK Response**

```c
void send_ack(int seq) {
    int64_t timestamp_us = esp_timer_get_time();
    char ack[64];
    int len = snprintf(ack, sizeof(ack), "<ACK,%d,%lld>\n",
                       seq, timestamp_us);
    uart_write_bytes(UART_NUM, ack, len);
}
```

**Components:**

- `esp_timer_get_time()`: Microseconds since boot
- `snprintf()`: Safe string formatting (prevents buffer overflow)
- `uart_write_bytes()`: Send to PC

**ACK format:** `<ACK,SEQ,TIMESTAMP_US>`

**Why ACK?**

- PC knows message was received
- PC can measure round-trip latency
- Useful for debugging timing issues

**7. Main Application Task**

```c
void app_main(void) {
    configure_uart();

    printf("\n\n\nESP32 Benchmark Firmware (ESP-IDF)\n");
    printf("=====================================\n");
    printf("Ready to receive commands...\n");
```

```
    printf("Send 'STATS' to view statistics\n\n");
    printf("READY: ESP32_BENCHMARK_V2\n");
```

**Why `app_main()`?**

- ESP-IDF entry point (like `main()` in normal C)
- Runs as a FreeRTOS task
- Don't return from this function!

```
    uint8_t data[BUF_SIZE];

    while (1) {
        int len = uart_read_bytes(UART_NUM, data, BUF_SIZE - 1,
                                  20 / portTICK_PERIOD_MS);
```

**Infinite Loop:**

- Embedded systems run forever
- `uart_read_bytes()` blocks for up to 20ms
- `portTICK_PERIOD_MS`: Converts ms to FreeRTOS ticks

```
        if (len > 0) {
            data[len] = '\0';  // Null-terminate for string functions

            // Check for STATS command
            if (strstr((char*)data, "STATS") != NULL) {
                float loss_rate = (received_count > 0)
                    ? (100.0 * lost_packets / (received_count +
lost_packets))
                    : 0.0;

                printf("\n=== Communication Statistics ===\n");
                printf("Packets Received: %d\n", received_count);
                printf("Packets Lost:     %d\n", lost_packets);
                printf("Loss Rate:        %.2f%%\n", loss_rate);
                printf("================================\n\n");
                continue;
            }
```

**STATS Command:**

- Useful for debugging
- Send "STATS" from PC to see counters
- `strstr()`: Searches for substring

```
            // Parse incoming command
            int seq;
```

```
            float joints[6];

            if (parse_message((char*)data, &seq, joints)) {
                received_count++;
                handle_packet_loss(seq);

                // Log received command
                ESP_LOGI(TAG, "SEQ:%d J:[%.3f,%.3f,%.3f,%.3f,%.3f,%.3f]",
                        seq, joints[0], joints[1], joints[2],
                        joints[3], joints[4], joints[5]);

                // Send ACK back to PC
                send_ack(seq);

                // *** THIS IS WHERE YOU'D CONTROL MOTORS ***
                // move_motors(joints);
            }
        }
    }
}
```

**Main Processing:**

1. Parse message
2. If valid: log, detect losses, send ACK
3. **Motor control would go here** (see next section)

---

## 🎯 Key Concepts

### 1. FreeRTOS Tasks

ESP32 runs FreeRTOS, a real-time operating system.

**What this means:**

- Multiple tasks can run "simultaneously" (time-sliced)
- Tasks have priorities
- Use `vTaskDelay()` instead of busy-waiting

**Example - Creating a task:**

```
void motor_control_task(void *pvParameters) {
    while(1) {
        // Control motors
        vTaskDelay(10 / portTICK_PERIOD_MS);  // 10ms delay
    }
}

// In app_main():
xTaskCreate(motor_control_task, "motor_ctrl", 4096, NULL, 5, NULL);
```

### 2. ESP-IDF Logging

```
ESP_LOGE(TAG, "Error message");    // Error (red)
ESP_LOGW(TAG, "Warning message");  // Warning (yellow)
ESP_LOGI(TAG, "Info message");     // Info (white)
ESP_LOGD(TAG, "Debug message");    // Debug (gray, disabled by default)
ESP_LOGV(TAG, "Verbose message");  // Verbose (disabled by default)
```

**Filtering logs:**

```
# In menuconfig:
idf.py menuconfig
# Component config → Log output → Default log verbosity
```

### 3. UART vs USB-Serial

**UART_NUM_0:**

- ESP32's built-in USB-serial converter
- Same port used for programming and logging
- `/dev/ttyUSB0` on Linux

**For real projects:**

- Use UART_NUM_1 or UART_NUM_2 for robot communication
- Keep UART_NUM_0 for debugging logs

### 4. Timing Functions

```
esp_timer_get_time();      // Microseconds since boot
xTaskGetTickCount();       // FreeRTOS ticks since boot
vTaskDelay(ms);            // Non-blocking delay
```

---

## 🤖 Adapting for Motor Control

### Step 1: Add Motor Control Function

```
// At top of file:
#include "driver/mcpwm.h"  // For PWM control
#include "driver/gpio.h"   // For GPIO pins

// Motor control pins (example)
#define MOTOR1_PWM_PIN  25
```

```
#define MOTOR1_DIR_PIN  26
// ... define for all 6 motors

void move_motors(float joints[6]) {
    // Convert radians to motor positions/pulses
    for(int i = 0; i < 6; i++) {
        // Example for stepper motor:
        int steps = (int)(joints[i] * STEPS_PER_RADIAN);

        // Send pulses to motor driver
        // stepper_move_to(i, steps);

        // OR for servo:
        // servo_set_angle(i, joints[i]);

        ESP_LOGD(TAG, "Motor %d: %.3f rad = %d steps",
                 i, joints[i], steps);
    }
}
```

### Step 2: Call in Main Loop

```
if (parse_message((char*)data, &seq, joints)) {
    received_count++;
    handle_packet_loss(seq);

    // *** ADD THIS ***
    move_motors(joints);

    send_ack(seq);
}
```

### Step 3: Add Velocity Calculation (Optional)

```
static float prev_joints[6] = {0};
static int64_t prev_time_us = 0;

void move_motors_with_velocity(float joints[6]) {
    int64_t current_time_us = esp_timer_get_time();
    float dt = (current_time_us - prev_time_us) / 1000000.0;  // seconds

    for(int i = 0; i < 6; i++) {
        float velocity = (joints[i] - prev_joints[i]) / dt;  // rad/s

        // Use velocity for motion planning
        // smooth_move_motor(i, joints[i], velocity);

        prev_joints[i] = joints[i];
    }
```

```
    prev_time_us = current_time_us;
}
```

### Step 4: Add Trajectory Interpolation (Advanced)

For smoother motion with 50ms update rate:

```
void interpolate_trajectory(float start[6], float end[6], int steps) {
    for(int step = 0; step <= steps; step++) {
        float t = (float)step / steps;  // 0.0 to 1.0

        for(int i = 0; i < 6; i++) {
            float pos = start[i] + t * (end[i] - start[i]);
            // set_motor_position(i, pos);
        }

        vTaskDelay(5 / portTICK_PERIOD_MS);  // 5ms per step
    }
}
```

---

## 🛠️ ESP-IDF Patterns & Best Practices

### 1. Error Handling

```
// Good: Check return values
esp_err_t ret = uart_read_bytes(...);
if (ret < 0) {
    ESP_LOGE(TAG, "UART read failed: %s", esp_err_to_name(ret));
}

// Or use macro for critical operations:
ESP_ERROR_CHECK(gpio_set_direction(GPIO_NUM_25, GPIO_MODE_OUTPUT));
```

### 2. Resource Management

```
// Allocate heap memory:
float* trajectory = malloc(100 * sizeof(float));
if (trajectory == NULL) {
    ESP_LOGE(TAG, "Failed to allocate memory!");
    return;
}

// Always free when done:
free(trajectory);
```

### 3. Configuration

Use `menuconfig` for compile-time settings:

```
// In Kconfig file:
menu "Motor Configuration"
    config MOTOR_COUNT
        int "Number of Motors"
        default 6

    config STEPS_PER_REV
        int "Steps per Revolution"
        default 200
endmenu

// In C code:
#include "sdkconfig.h"
#define NUM_MOTORS CONFIG_MOTOR_COUNT
```

### 4. Non-Blocking Operations

```
// Bad: Blocking
while(motor_is_moving()) {
    // Wastes CPU
}

// Good: Non-blocking
if (motor_is_moving()) {
    vTaskDelay(1);  // Yield to other tasks
}
```

---

## 🐛 Debugging Tips

### 1. Enable Verbose Logging

```
idf.py menuconfig
# Set "Default log verbosity" to "Verbose"
```

### 2. Use GDB Debugging

```
idf.py openocd  # Terminal 1
idf.py gdb      # Terminal 2
```

### 3. Monitor Task Stack Usage

```
// Check stack high water mark (free space remaining)
UBaseType_t stack_left = uxTaskGetStackHighWaterMark(NULL);
ESP_LOGI(TAG, "Stack remaining: %d bytes", stack_left * 4);
```

### 4. Assert Statements

```
#include <assert.h>

void process_joints(float joints[6]) {
    assert(joints != NULL);
    assert(joints[0] >= -3.14 && joints[0] <= 3.14);
    // ... will crash with error message if false
}
```

### 5. Serial Monitor Tips

```
# Normal monitoring
idf.py -p /dev/ttyUSB0 monitor

# With timestamps
idf.py -p /dev/ttyUSB0 monitor | ts

# Filter logs
idf.py -p /dev/ttyUSB0 monitor | grep "BENCHMARK"
```

---

## 📖 Next Steps

For Motor Integration:

1. **Choose motor driver library:**
   - Stepper: `esp32-stepper-motor-driver`
   - Servo: ESP-IDF PWM examples
   - Closed-loop: Custom PID controller
2. **Add hardware abstraction:**
   - Create `motor_driver.h/c` component
   - Abstract different motor types
   - Easy to swap implementations
3. **Implement safety:**

    - Position limits
    - Velocity limits
    - Emergency stop
    - Timeout detection

4. **Test incrementally:**

    - One motor at a time
    - Check at low speeds first
    - Verify with oscilloscope/logic analyzer

Learning Resources:

- **ESP-IDF Programming Guide:** https://docs.espressif.com/projects/esp-idf/
- **FreeRTOS Documentation:** https://www.freertos.org/
- **ESP32 Technical Reference:** Detailed hardware specs

---

## ❓ Common Questions

**Q: Why not use Arduino instead of IDF?** A: ESP-IDF gives you:

- Lower latency
- Better real-time performance
- Access to all

ESP32 features

- Professional development tools

**Q: Can I use C++ instead of C?** A: Yes! Rename `benchmark_main.c` to `benchmark_main.cpp` and update CMakeLists.txt.

**Q: How much flash/RAM does this use?** A: Current firmware: ~150KB flash, ~20KB RAM. ESP32 has 4MB flash, 520KB RAM.

**Q: Can ESP32 handle 6 motors at 20Hz?** A: Easily! ESP32 runs at 240MHz dual-core. Plenty of headroom.

---

**Questions or issues?** Check existing ESP-IDF examples or ask the team!