How to Use the NVS (Non-Volatile Storage) on ESP32 Using ESP-IDF and FreeRTOS

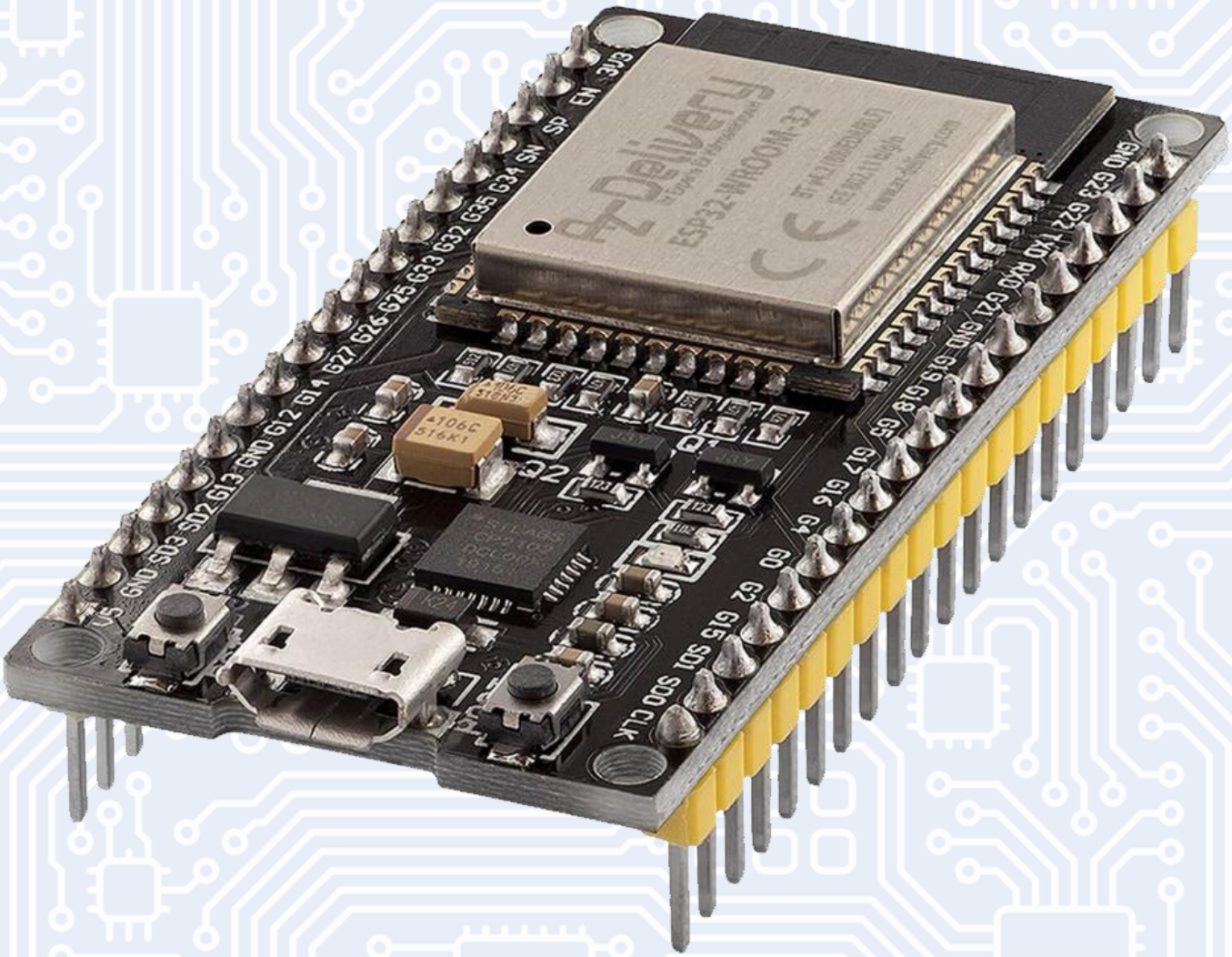

Table of Contents

1. Introduction

In IoT development using the **ESP32 MCU**, the **NVS (Non-Volatile Storage)** is a critical component provided by the **ESP-IDF framework**. It allows developers to store key-value pairs persistently in flash memory, surviving reboots and power cycles. This is ideal for saving configuration parameters, calibration values, user preferences, counters, credentials, and more.

1. Introduction

NVS (Non-Volatile Storage) is a key-value storage system that uses **flash memory**. It is designed for storing small data persistently in a compact, fault-tolerant way.

ESP-IDF offers nvs.h, which allows you to:

- Store strings, integers, blobs, etc.
- Read and write with minimal flash wear.
- Share namespaces across applications or tasks.

2. When to Use NVS in IoT Projects

Use NVS to store:

- Wi-Fi credentials
- Device-specific configuration (e.g., thresholds, calibration)
- Usage counters (e.g., number of resets)
- Last known states (e.g., switch state, brightness level)

3. Steps to Use NVS

1. Initialize the NVS flash

```c
esp_err_t ret = nvs_flash_init();
if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
    ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
    ESP_ERROR_CHECK(nvs_flash_erase());
    ret = nvs_flash_init();
}
ESP_ERROR_CHECK(ret);
```

2. Open a handle to a namespace

```c
nvs_handle_t my_handle;
ESP_ERROR_CHECK(nvs_open("storage", NVS_READWRITE,
                         &my_handle));
```

3. Write key-value data

```c
ESP_ERROR_CHECK(nvs_set_i32(my_handle, "boot_count", 42));
ESP_ERROR_CHECK(nvs_commit(my_handle));
```

3. Steps to Use NVS

4. Read data

```c
int32_t boot_count = 0;
esp_err_t err = nvs_get_i32(my_handle, "boot_count", &boot_count);
if (err == ESP_OK) {
    printf("Boot count = %d\n", boot_count);
} else if (err == ESP_ERR_NVS_NOT_FOUND) {
    printf("Key not found\n");
} else {
    printf("Error (%s) reading!\n", esp_err_to_name(err));
}
```

5. Close the handle

```c
nvs_close(my_handle);
```

4. Supported Data Types

Function	Data Type
nvs_set_i8	int8_t
nvs_set_u8	uint8_t
nvs_set_i16	int16_t
nvs_set_u16	uint16_t
nvs_set_i32	int32_t
nvs_set_u32	uint32_t
nvs_set_i64	int64_t
nvs_set_u64	uint64_t
nvs_set_str	null-terminated string
nvs_set_blob	binary blob

Each has a corresponding `nvs_get_...` variant.

5. Real-Life Code Example

Save and read a boot counter across restarts

```c
#include <stdio.h>
#include "nvs_flash.h"
#include "nvs.h"

void app_main(void) {
    // Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
        ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    // Open storage
    nvs_handle_t nvs_handle;
    ESP_ERROR_CHECK(nvs_open("boot", NVS_READWRITE, &nvs_handle));

    // Read boot counter
    int32_t boot_count = 0;
    ret = nvs_get_i32(nvs_handle, "boot_count", &boot_count);
    if (ret == ESP_OK) {
        printf("Previous boot count: %d\n", boot_count);
    } else {
        printf("Boot count not initialized yet.\n");
    }

    // Increment and write it back
    boot_count++;
    ESP_ERROR_CHECK(nvs_set_i32(nvs_handle, "boot_count", boot_count));
    ESP_ERROR_CHECK(nvs_commit(nvs_handle));
    printf("New boot count saved: %d\n", boot_count);

    // Close
```

5. Real-Life Code Example

```c
#include <stdio.h>
#include "nvs_flash.h"
#include "nvs.h"

void app_main(void) {
    // Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
        ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    // Open storage
    nvs_handle_t nvs_handle;
    ESP_ERROR_CHECK(nvs_open("boot", NVS_READWRITE, &nvs_handle));

    // Read boot counter
    int32_t boot_count = 0;
    ret = nvs_get_i32(nvs_handle, "boot_count", &boot_count);
    if (ret == ESP_OK) {
        printf("Previous boot count: %d\n", boot_count);
    } else {
        printf("Boot count not initialized yet.\n");
    }

    // Increment and write it back
    boot_count++;
    ESP_ERROR_CHECK(nvs_set_i32(nvs_handle, "boot_count", boot_count));
    ESP_ERROR_CHECK(nvs_commit(nvs_handle));
    printf("New boot count saved: %d\n", boot_count);

    // Close
    nvs_close(nvs_handle);
}
```

6. Error Handling and Best Practices

- Always **check return values** from `nvs_get_*()` and `nvs_set_*()`.
- Don’t forget to call `nvs_commit()` after writes.
- For heavy writes (like counters), consider caching and writing less frequently to reduce flash wear.
- Use **namespaces** to organize data (`nvs_open("wifi", ...)`, `nvs_open("user_cfg", ...)`).
- Use **custom partitions** if the default NVS is not enough or conflicts with Wi-Fi.

7. Conclusion

NVS is a powerful and easy-to-use key-value storage system that is essential in many ESP32-based IoT applications. It enables **persistent configuration and state management** in a lightweight way. Whether you're storing Wi-Fi settings, sensor calibration, or user preferences, the NVS API gives you a safe and robust foundation to work from.