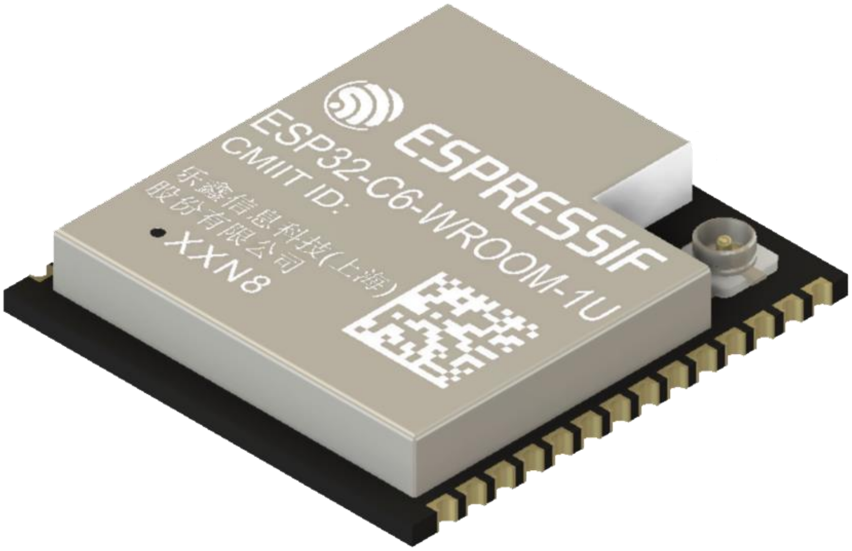
Day 1

Introduction to RTOS and FreeRTOS

freeRTOS



DISCOVER, LEARN, SUCCEED!

https://www.linkedin.com/in/yamil-garcia
https://www.youtube.com/@LearningByTutorials
https://github.com/god233012yamil

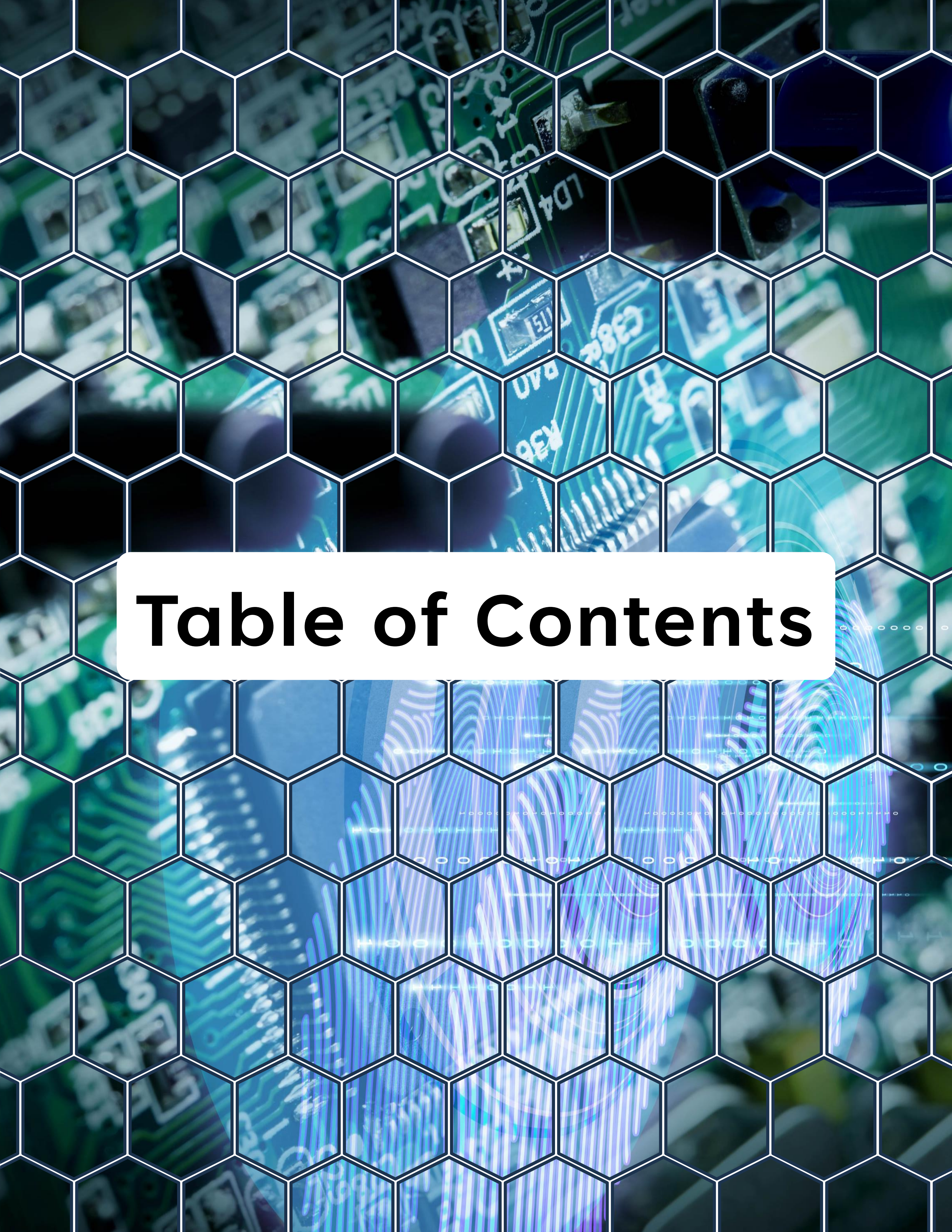
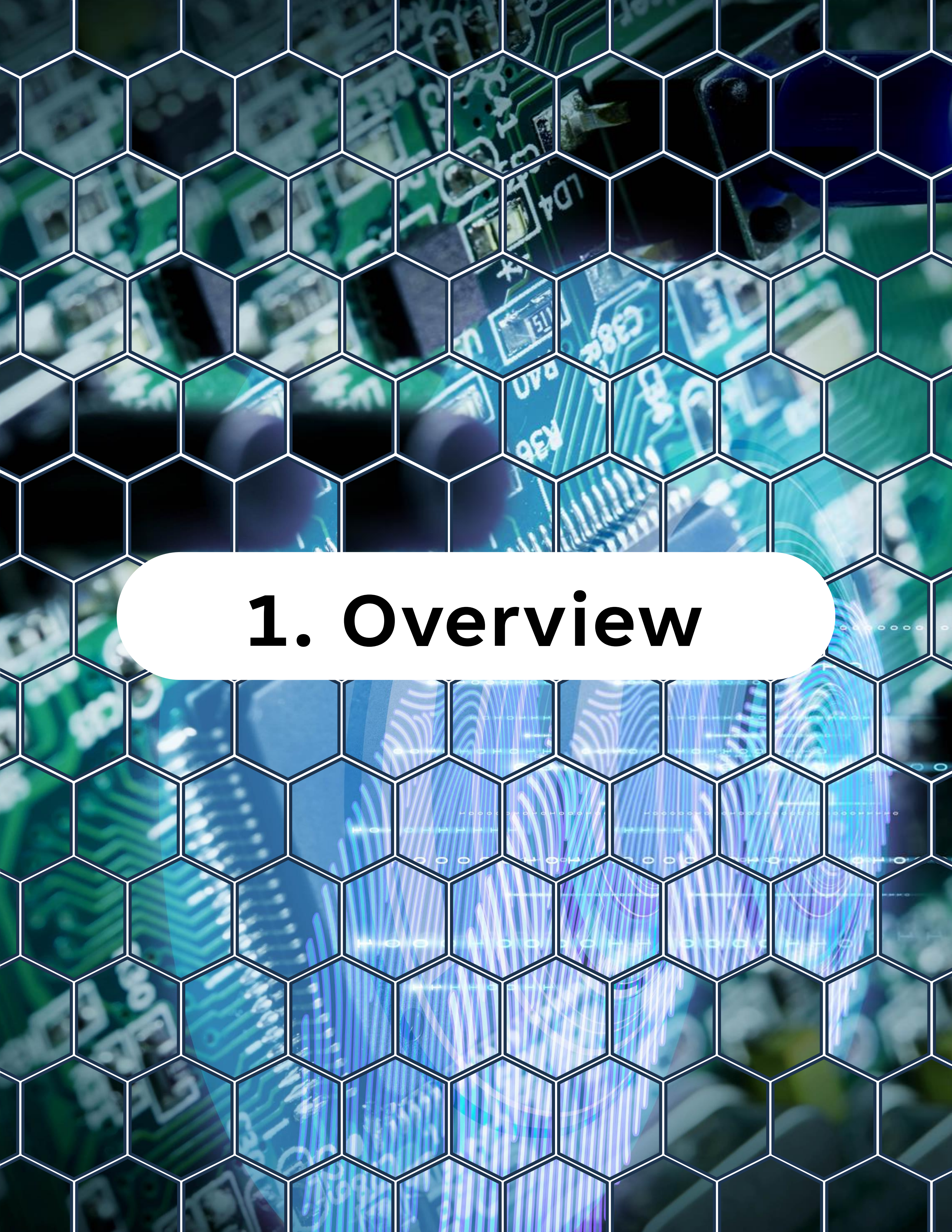


Table of Contents

Table of Contents

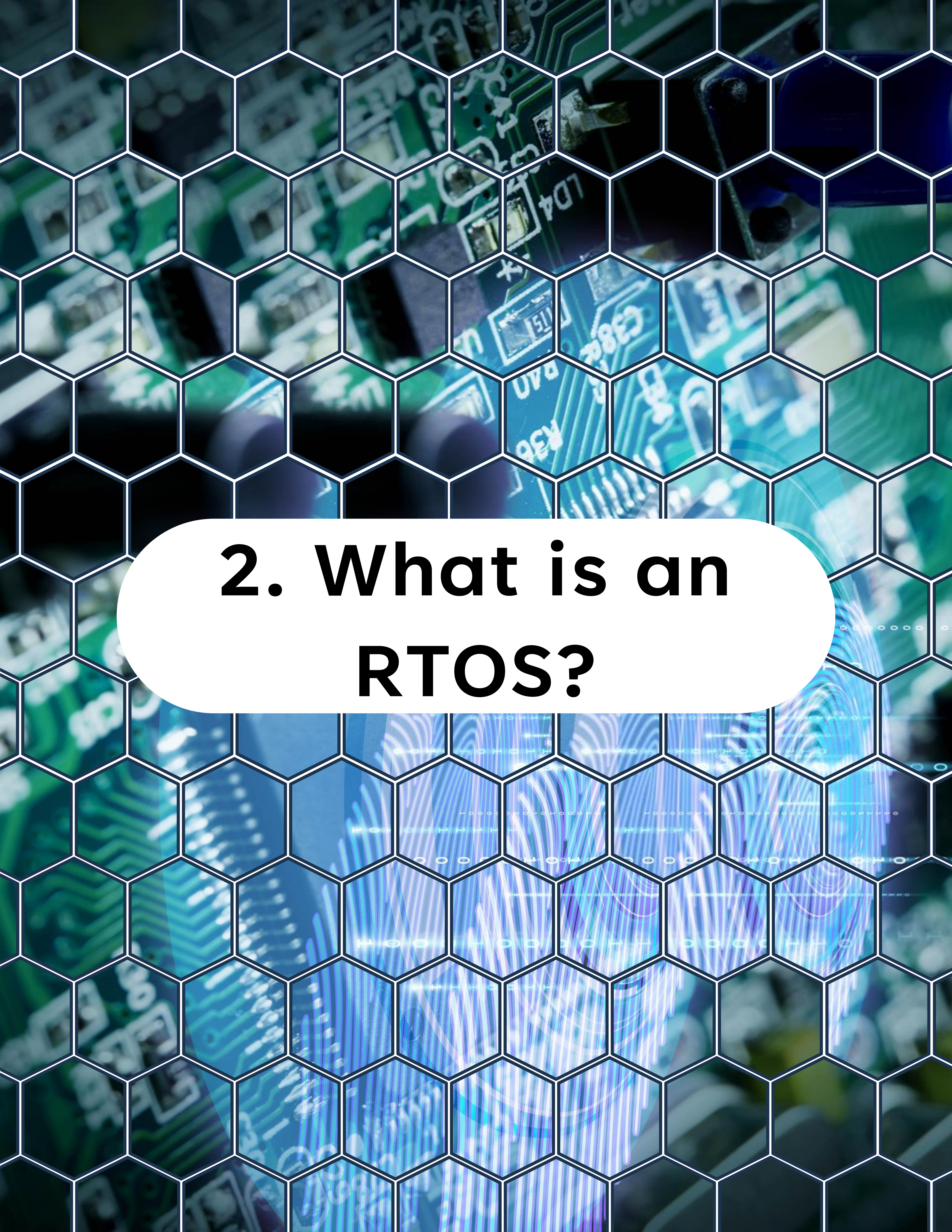
1. Overview



1. Overview

On Day 1, you will learn:

- What an RTOS is and why it's useful.
- The basics of FreeRTOS and its relevance to embedded systems.
- Why ESP32 and ESP-IDF are a perfect match for FreeRTOS development.
- What to expect from the upcoming days.



2. What is an RTOS?

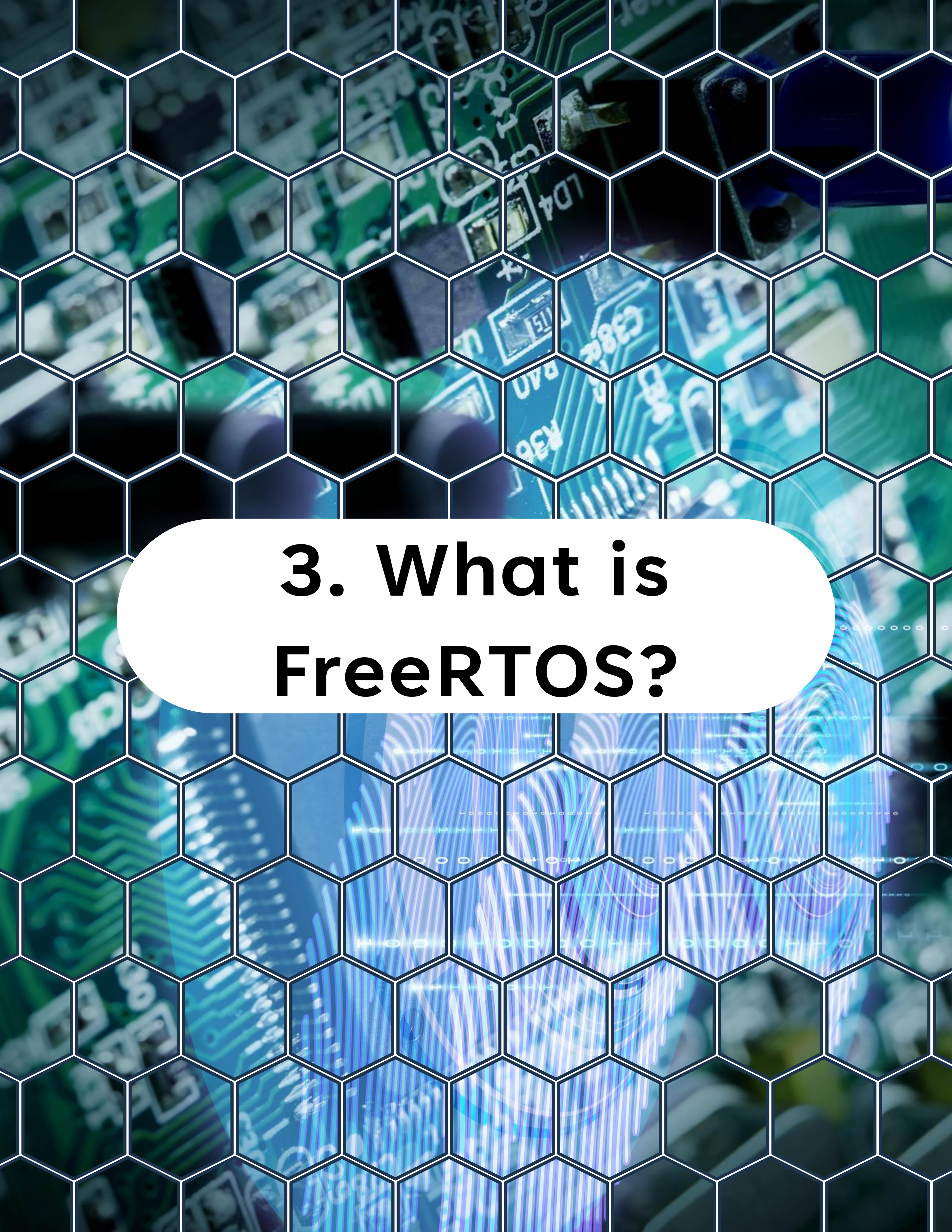
2. What is an RTOS?

A **Real-Time Operating System (RTOS)** is a specialized OS that enables deterministic task scheduling with guaranteed response times. Unlike general-purpose OSs, it focuses on predictability and timing constraints, critical for embedded applications such as motor control, data acquisition, or communication systems.

Key Characteristics:

- **Multitasking** with preemptive or cooperative scheduling.
- **Real-time constraints** (e.g., deadlines).
- **Minimal latency**.
- Support for **inter-task communication** and **resource synchronization**.

3. What is FreeRTOS?

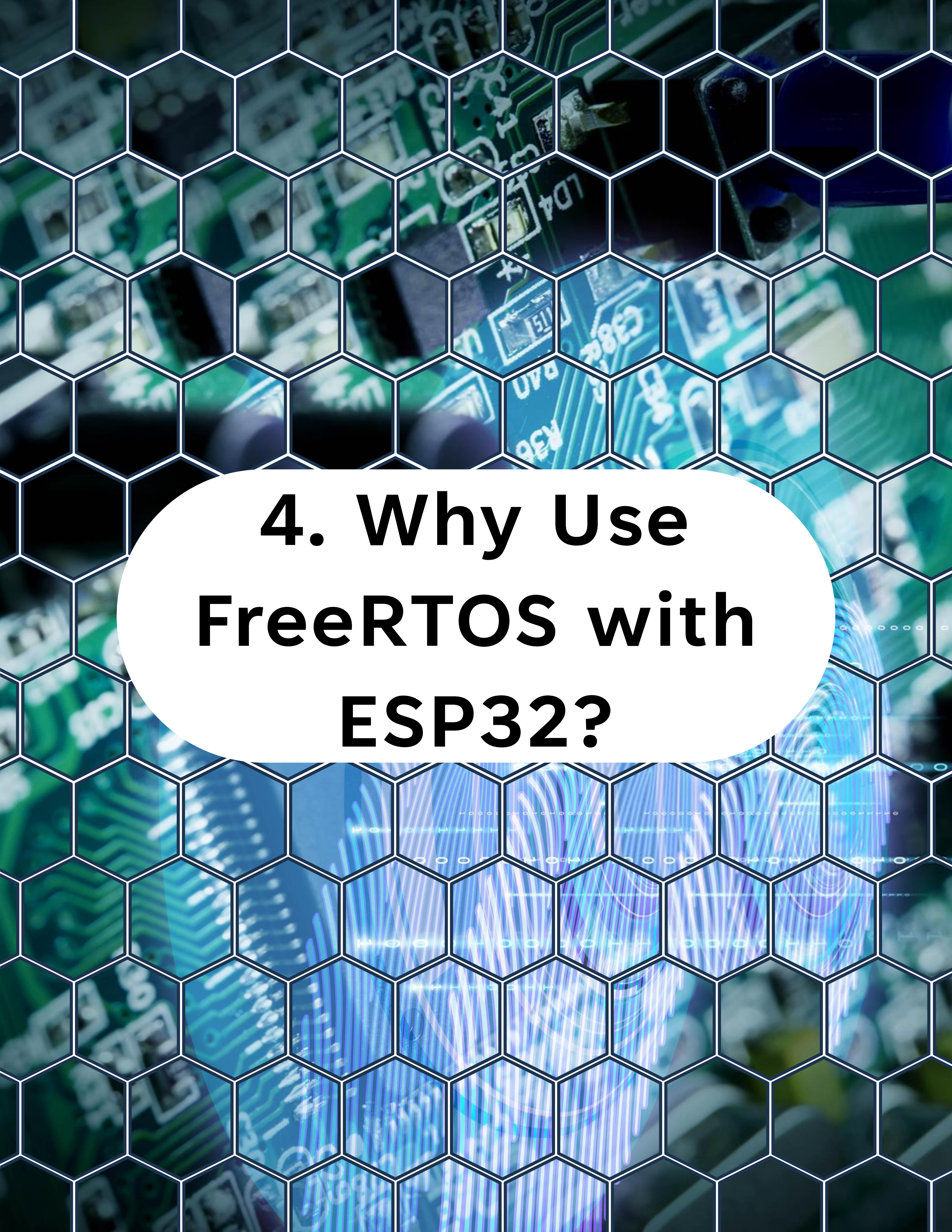


3. What is FreeRTOS?

FreeRTOS is an open-source real-time operating system kernel for embedded devices. It provides lightweight but powerful APIs for multitasking, synchronization, and timing.

Why Use FreeRTOS?

- Small footprint (few KBs of ROM/RAM).
- Portable (runs on many MCU architectures).
- Rich feature set: tasks, queues, semaphores, timers, etc.
- Large community and official support from Espressif for ESP32.



4. Why Use FreeRTOS with ESP32?

4. Why Use FreeRTOS with ESP32?

ESP32 is a dual-core Xtensa microcontroller with built-in Wi-Fi and Bluetooth. FreeRTOS runs natively on ESP32 using Symmetric Multiprocessing (SMP), enabling efficient multitasking on both cores.

ESP-IDF, Espressif’s official development framework, integrates FreeRTOS by default. That means no extra installation is needed, you're ready to write multitasking code out of the box.

5. What’s Under the Hood?

5. What’s Under the Hood?

When you run a FreeRTOS-based ESP32 project via ESP-IDF:

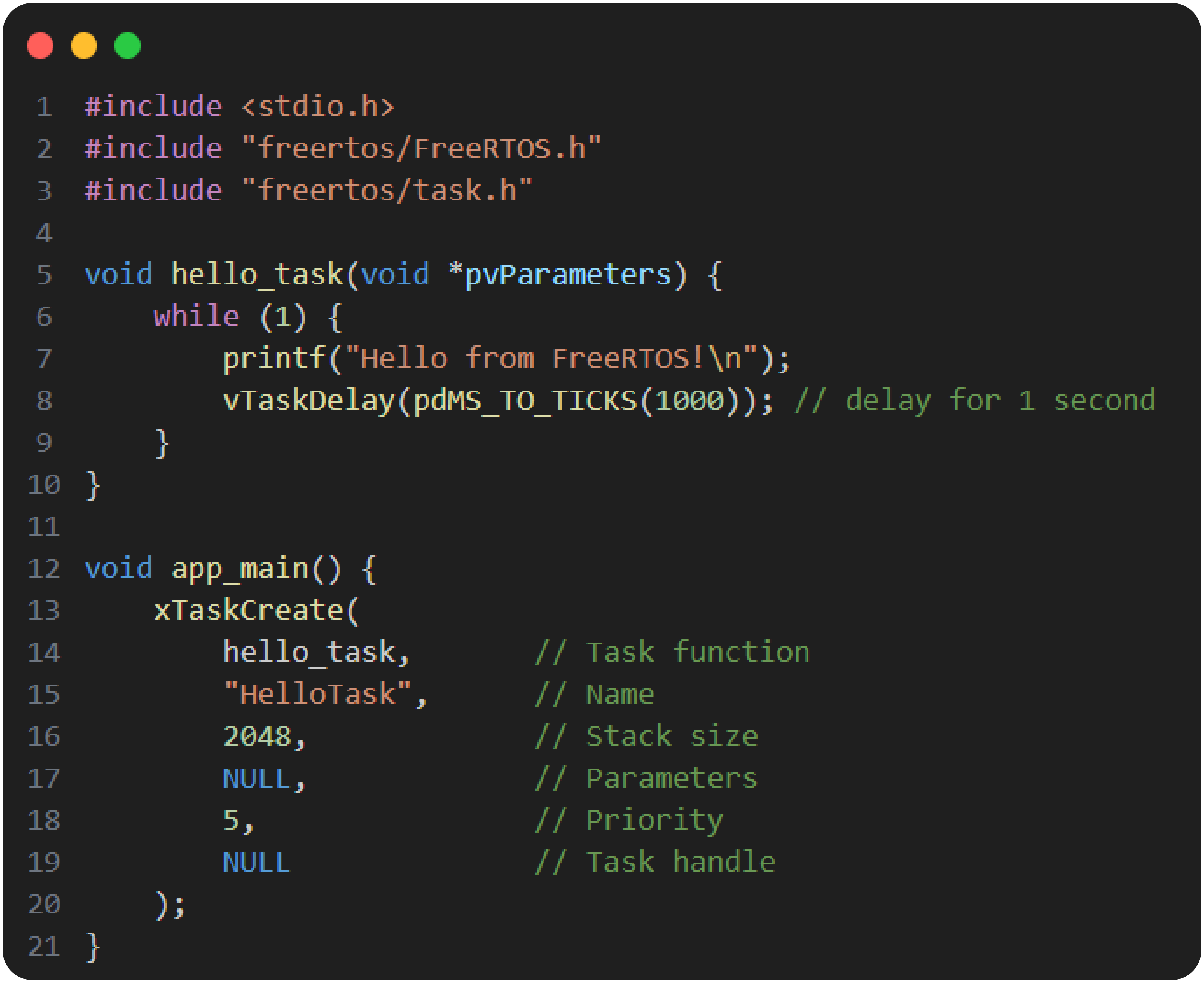
- **Tasks** are distributed across **Core 0** and **Core 1**.
- The **main application runs as a FreeRTOS task.**
- ESP-IDF sets up **system tasks** for Wi-Fi, Bluetooth, and other components.

6. First Look at a FreeRTOS Task

6. First Look at a FreeRTOS Task

Let’s look at what a simple FreeRTOS task looks like in ESP-IDF.

Code Snippet: Basic Task

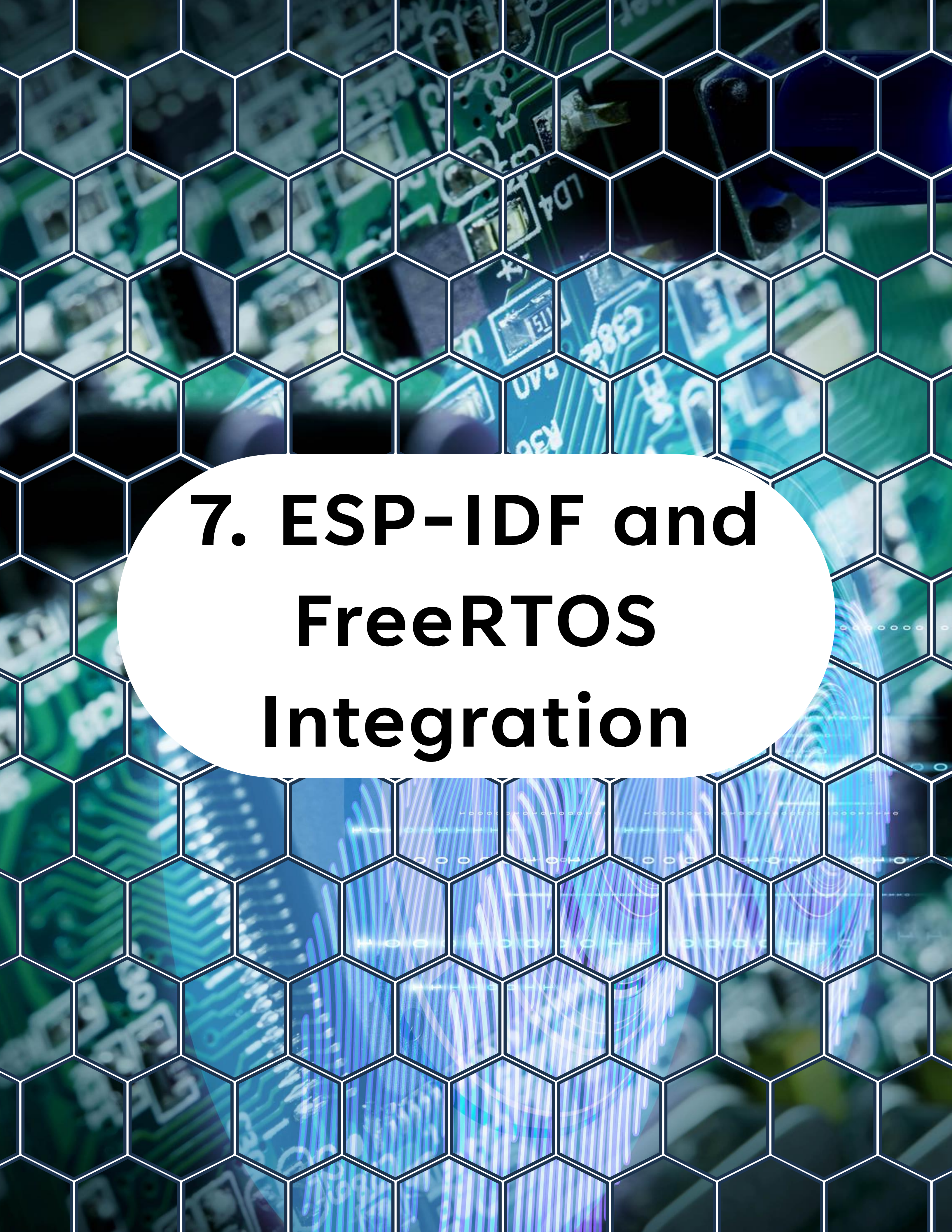


```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

void hello_task(void *pvParameters) {
    while (1) {
        printf("Hello from FreeRTOS!\n");
        vTaskDelay(pdMS_TO_TICKS(1000)); // delay for 1 second
    }
}

void app_main() {
    xTaskCreate(
        hello_task,       // Task function
        "HelloTask",      // Name
        2048,             // Stack size
        NULL,             // Parameters
        5,                // Priority
        NULL              // Task handle
    );
}
```

6. First Look at a FreeRTOS Task

Explanation:

- xTaskCreate() creates a task.
- The task runs in a loop and prints a message.
- vTaskDelay() pauses the task without blocking others.
- ESP32 handles the scheduling.



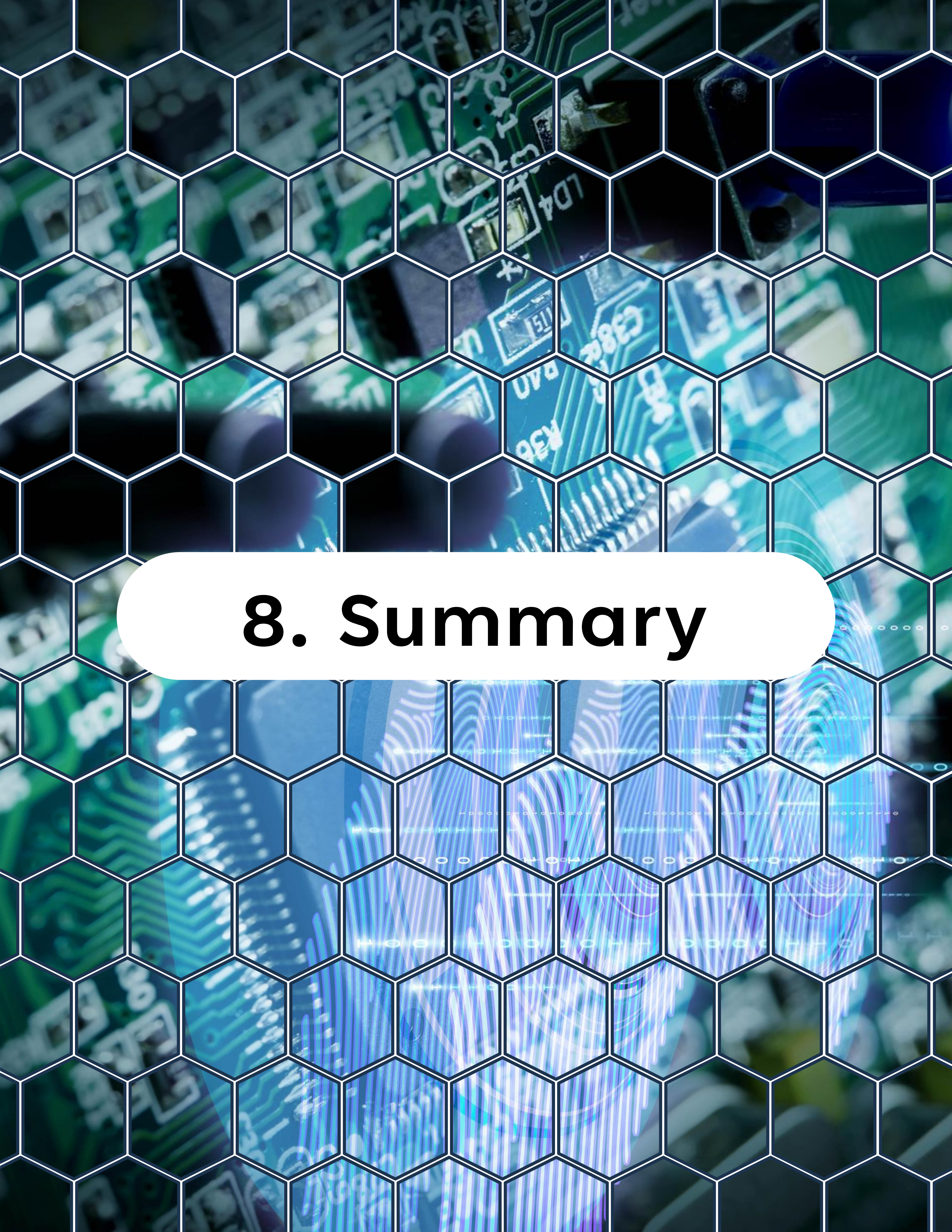
7. ESP-IDF and FreeRTOS Integration

7. ESP-IDF and FreeRTOS Integration

You don’t need to install FreeRTOS separately when using ESP-IDF. It's tightly integrated:

- **Headers available**: freertos/FreeRTOS.h, freertos/task.h, freertos/queue.h, etc.
- **Managed by CMake** in CMakeLists.txt.
- **Configurable via** menuconfig → Component config → FreeRTOS.

8. Summary



8. Summary

Today, you learned:

- What an RTOS is and why FreeRTOS is ideal for embedded systems.
- The basics of FreeRTOS architecture.
- How FreeRTOS is integrated with ESP-IDF.
- What a simple FreeRTOS task looks like.

9. Challenge for Today

9. Challenge for Today

Modify the example task to:

- Blink the onboard LED (GPIO2) every 500ms.
- Use gpio_set_level() to toggle it.

Use the GPIO and FreeRTOS delay functions together.

This will be a warm-up for next day!