30-Day FreeRTOS Course for ESP32 Using ESP-IDF

(Day 4)

“Creating and Deleting Tasks”

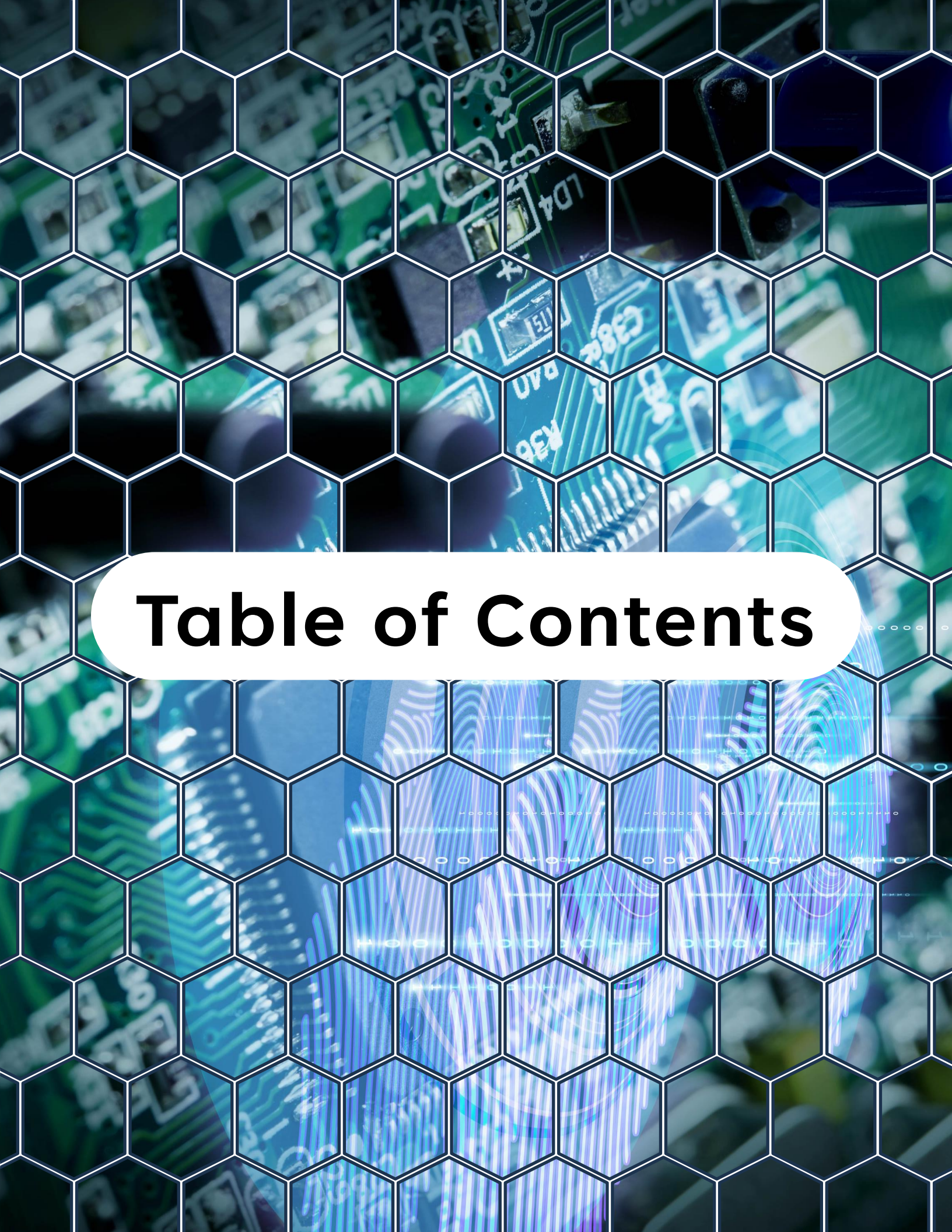
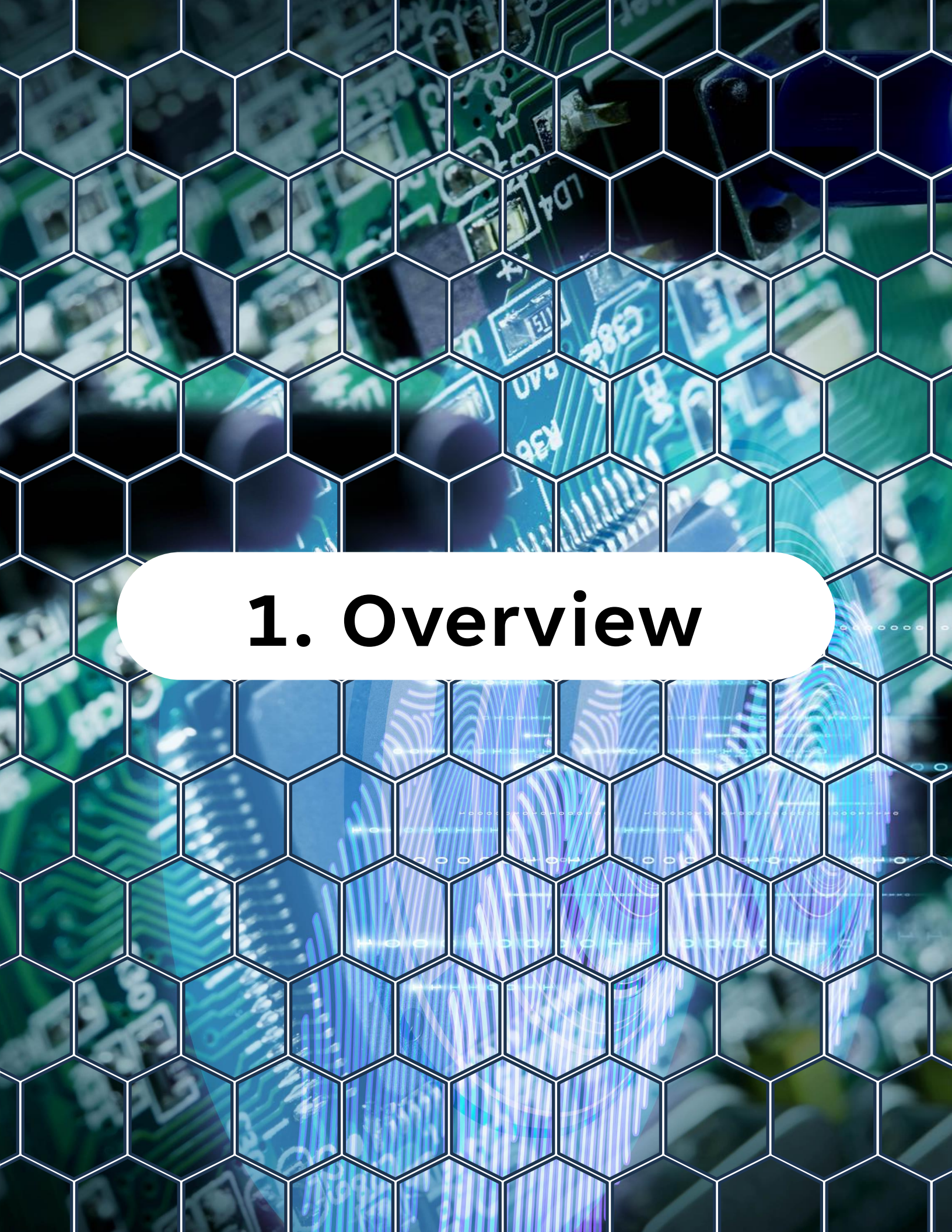


Table of Contents

Table of Contents



1. Overview

1. Overview

On Day 4, you'll learn:

- How to **create tasks** in FreeRTOS using ESP-IDF
- Understanding task creation parameters
- How to get and use **task handles**
- How to **delete tasks** safely
- Best practices for task lifecycle management
- Practical examples

By the end of this lesson, you'll be able to start and stop tasks cleanly in your ESP32 applications.

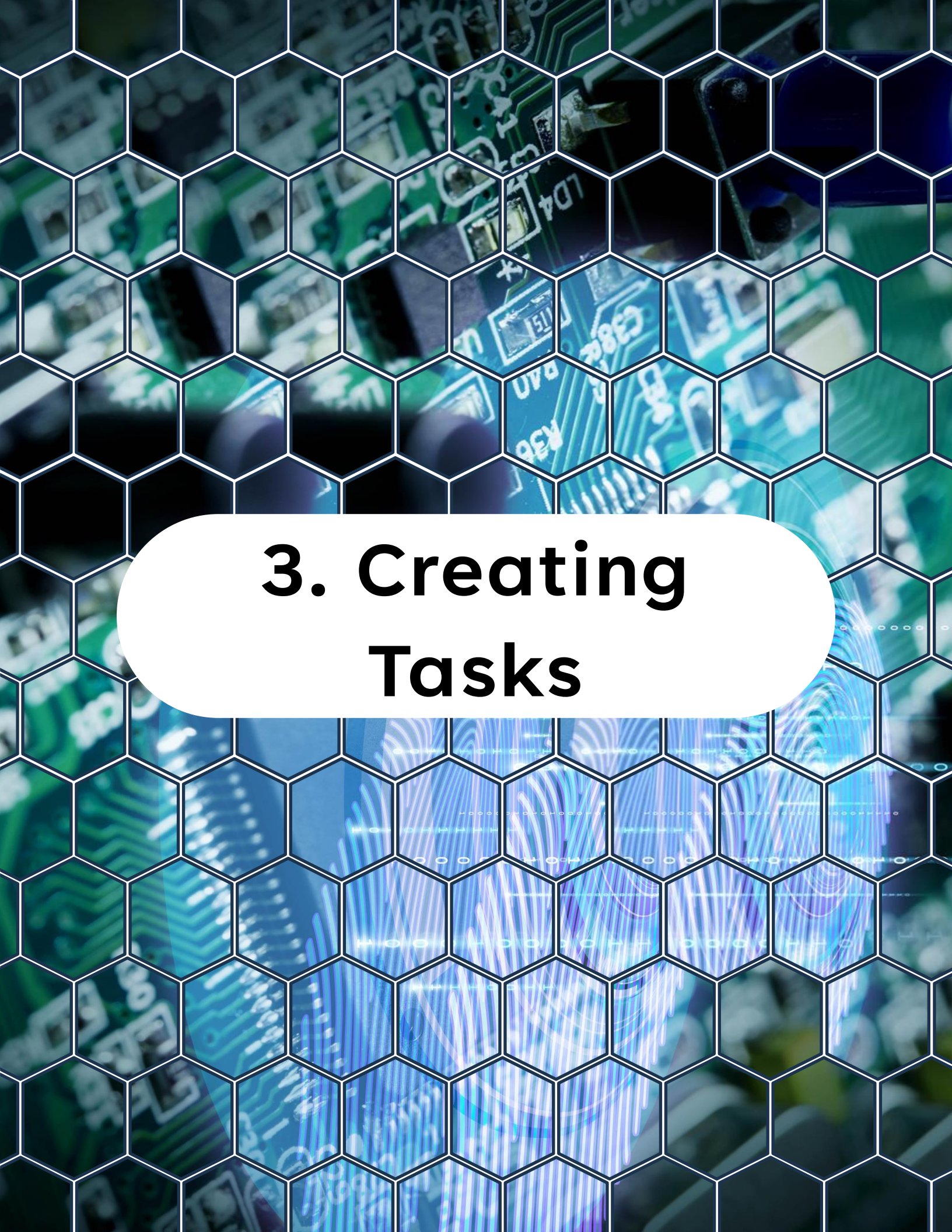
2. Tasks in FreeRTOS

2. Tasks in FreeRTOS

A task in FreeRTOS is a lightweight thread of execution that runs independently. Each task has:

- Its own **stack**
- Its own **priority**
- A unique **function** that defines what it does
- An optional **handle** to control it

FreeRTOS multitasks using preemptive scheduling, meaning multiple tasks share CPU time according to their priority.



3. Creating Tasks

3. Creating Tasks

The standard API to create a task is:

```
BaseType_t xTaskCreate(
    TaskFunction_t pvTaskCode,    // Task function
    const char * const pcName,    // Task name (for debugging)
    const uint32_t usStackDepth, // Stack size in words
    void *pvParameters,           // Parameters passed to the task
    UBaseType_t uxPriority,       // Task priority
    TaskHandle_t *pxCreatedTask  // Pointer to task handle
);
```

Parameters Explained:

Parameter	Description
pvTaskCode	Pointer to the function that implements the task.
pcName	A human-readable name (appears in debugging tools).
usStackDepth	Stack size in **words**, not bytes (e.g., 2048 means 2048 × 4 bytes on ESP32).
pvParameters	Pointer to data passed to the task (can be NULL).
uxPriority	Task priority (0 = lowest).
pxCreatedTask	Returns a task handle (optional).

3. Creating Tasks

Returns:

- pdPASS if task creation succeeded
- errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY if not enough memory

4. Deleting Tasks

4. Deleting Tasks

Tasks can delete themselves or be deleted by another task.

API:

```
void vTaskDelete(TaskHandle_t xTaskToDelete);
```

- Pass NULL to delete the **calling task** (self-delete).
- Pass a valid task handle to delete another task.

⚠️ **Important:**

Deleting a task **does not free heap memory automatically**. The memory is reclaimed by FreeRTOS when the task is deleted.

5. Practical Example – Creating and Deleting Tasks

5. Practical Example – Creating and Deleting Tasks

Code Example

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

TaskHandle_t task_handle_hello = NULL;

void hello_task(void *pvParameters) {
    int counter = 0;
    while (1) {
        printf("Hello Task running, counter = %d\n", counter++);
        vTaskDelay(pdMS_TO_TICKS(1000));
        if (counter >= 5) {
            printf("Hello Task deleting itself...\n");
            vTaskDelete(NULL); // Delete self
        }
    }
}

void control_task(void *pvParameters) {
    printf("Control Task running...\n");
    vTaskDelay(pdMS_TO_TICKS(3000));
    if (task_handle_hello != NULL) {
        printf("Control Task deleting Hello Task...\n");
        vTaskDelete(task_handle_hello); // Delete other task
    }
    vTaskDelete(NULL);
}

void app_main() {
    // Create Hello Task
    xTaskCreate(
        hello_task,         // Task function
        "Hello Task",       // Name
        2048,               // Stack size (words)
        NULL,               // Parameters
```

5. Practical Example – Creating and Deleting Tasks

```
    xTaskCreate(
        hello_task,          // Task function
        "Hello Task",        // Name
        2048,                // Stack size (words)
        NULL,                // Parameters
        5,                   // Priority
        &task_handle_hello   // Handle
    );

    // Create Control Task
    xTaskCreate(
        control_task,
        "Control Task",
        2048,
        NULL,
        4,
        NULL
    );
}
```

Expected Output:

Hello Task running, counter = 0

Control Task running...

Hello Task running, counter = 1

Hello Task running, counter = 2

Control Task deleting Hello Task...

Hello Task running, counter = 3

6. Best Practices for Task Creation & Deletion

6. Best Practices for Task Creation & Deletion

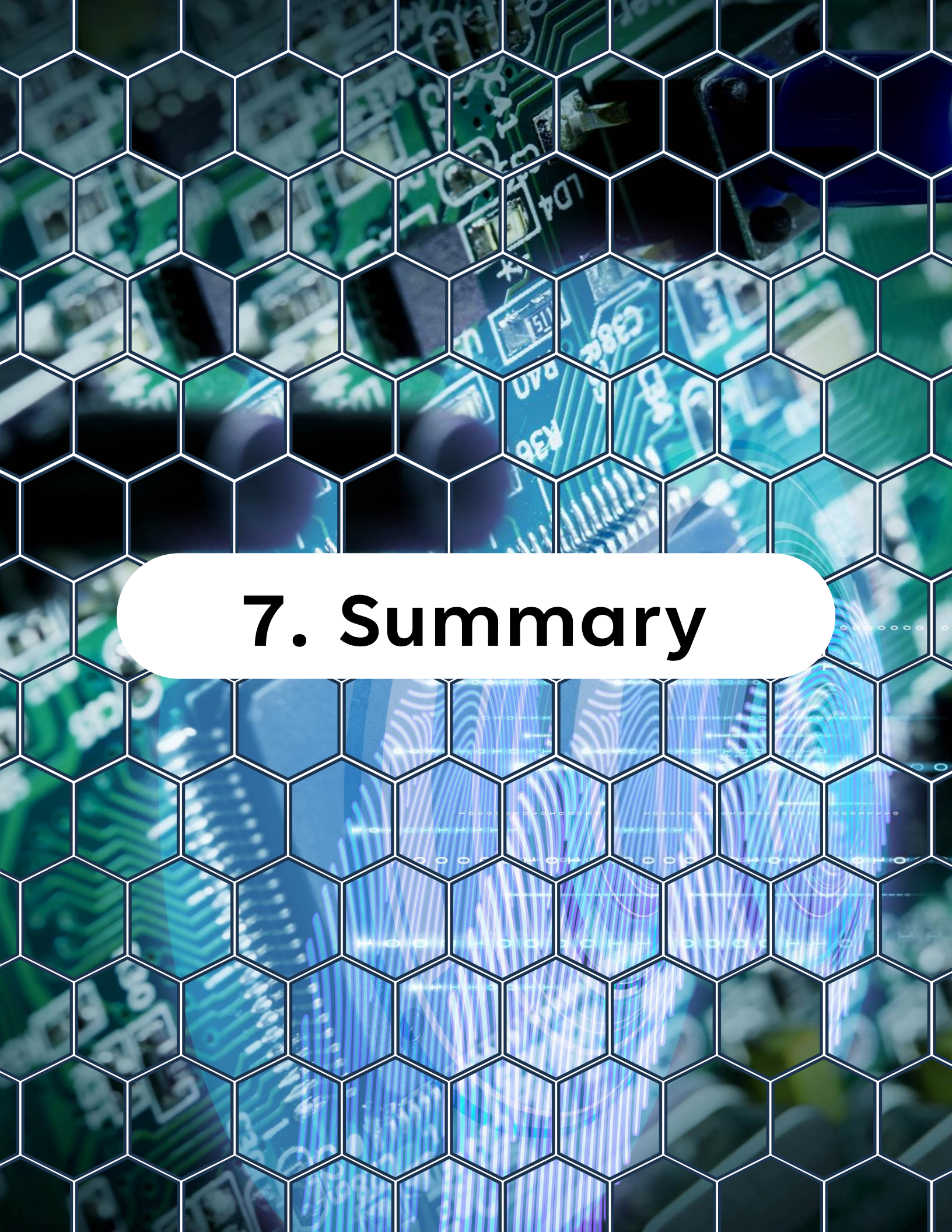
✅ **Always check return values** from `xTaskCreate()`

✅ **Keep stack sizes minimal but sufficient** (use `uxTaskGetStackHighWaterMark()` to measure)

✅ **Avoid frequent creation/deletion** – it can fragment memory

✅ **Prefer reusing tasks** by suspending/resuming rather than deleting when possible

✅ **Avoid deleting system tasks** like the idle task or timer service task



7. Summary

7. Summary

- **Task creation** uses xTaskCreate() or xTaskCreatePinnedToCore()
- **Task deletion** uses vTaskDelete() with a handle or NULL for self-deletion
- **Task handles** let you control tasks after creation
- Clean lifecycle management prevents memory leaks and undefined behavior

8. Challenge for Today

8. Challenge for Today

Create two tasks:

- **Blink Task** – blinks an LED every 500 ms, self-deletes after 10 blinks
- **Manager Task** – deletes the Blink Task after 5 seconds if still running

Observe how task deletion behaves in the monitor output.