Mastering Sleep Modes on the ESP32-C6

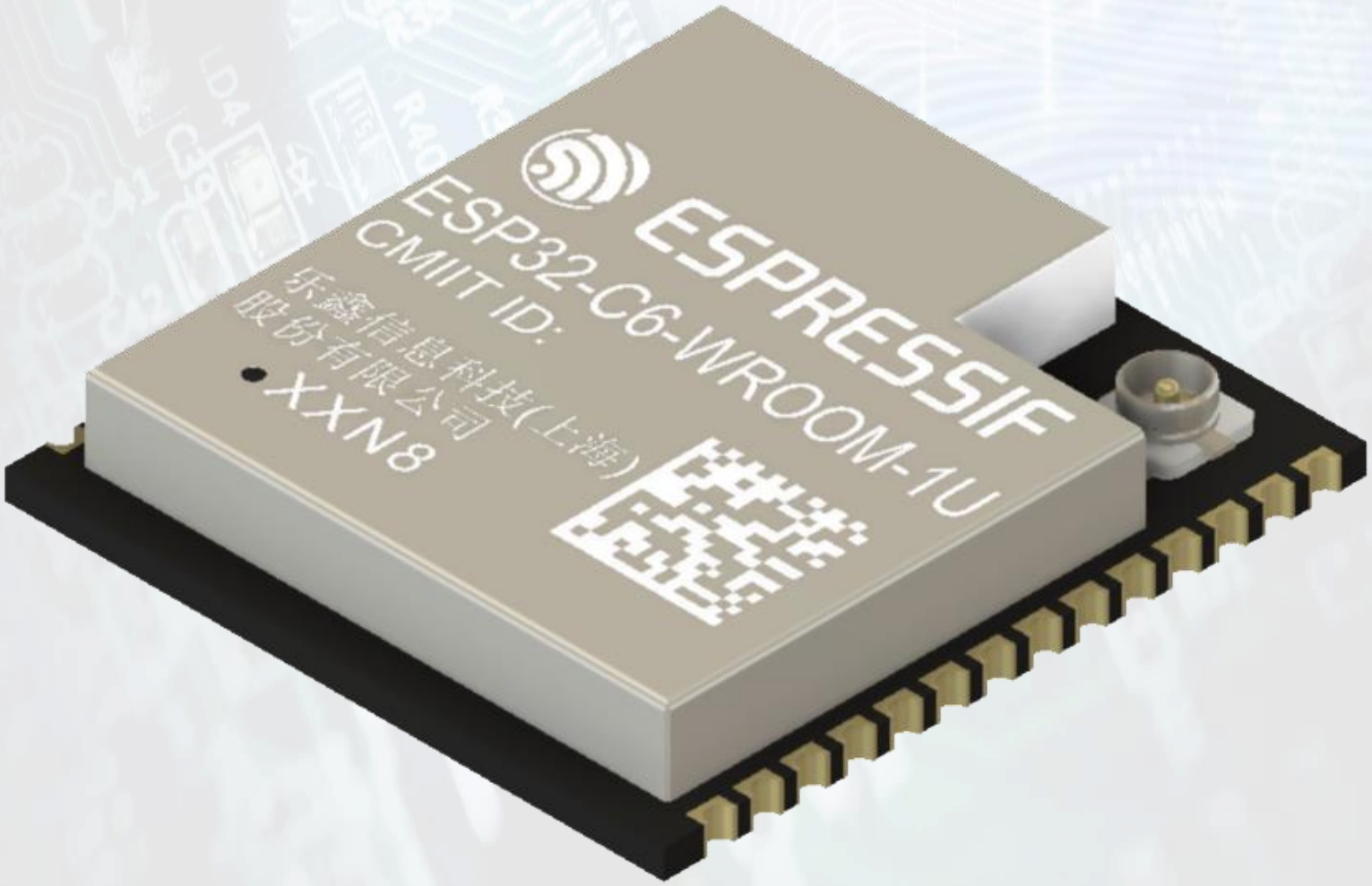

Table of Contents

1. Introduction

In IoT and battery-operated embedded systems, **power efficiency** is often just as critical as processing capability. The **ESP32-C6**, a RISC-V-based MCU from Espressif, is equipped with multiple sleep modes that enable designers to finely control power usage depending on the system's activity levels.

This guide dives into the effective use of **Modem Sleep**, **Light Sleep**, and **Deep Sleep** modes on the ESP32-C6, showcasing practical code examples and configuration tips using the ESP-IDF framework and FreeRTOS.

Whether you're building a wireless sensor node, a smart switch, or a portable monitoring

1. Introduction

C6, a RISC-V-based MCU from Espressif, is equipped with multiple sleep modes that enable designers to finely control power usage depending on the system's activity levels.

This guide dives into the effective use of **Modem Sleep**, **Light Sleep**, and **Deep Sleep** modes on the ESP32-C6, showcasing practical code examples and configuration tips using the ESP-IDF framework and FreeRTOS.

Whether you're building a wireless sensor node, a smart switch, or a portable monitoring system, this article will help you implement sleep modes intelligently to extend battery life without sacrificing responsiveness.

2. Why Sleep Modes Matter in Embedded Systems

Sleep modes allow embedded systems to **reduce current consumption dramatically** by powering down unused subsystems.

Proper use of sleep states ensures:

- Extended battery runtime
- Reduced thermal footprint
- Compliance with energy-efficiency standards
- Better responsiveness with real-time wake-up triggers

On the **ESP32-C6**, selecting the right sleep mode can reduce power draw from **tens of milliamps to microamps**, especially when Wi-Fi isn’t needed continuously.

3. Overview of Sleep Modes on ESP32-C6

Mode	CPU	Memory	Peripherals	Wi-Fi
Modem Sleep	On	On	On	Off
Light Sleep	Off	On	On (limited)	Off
Deep Sleep	Off	Off (except RTC)	Off	Off

Mode	Wake-up Time	Power
Modem Sleep	Instant	~10–20 mA
Light Sleep	Fast (<1 ms)	~1 mA
Deep Sleep	Slow (~200 ms)	~20–50 µA

4. Modem Sleep: Efficient Networking Idle

Use Case:

Wi-Fi connected, but no data transmission (e.g., MQTT keep-alive idle periods)

How It Works:

Only the Wi-Fi modem is powered down, while CPU and RAM stay active. It’s managed automatically by the Wi-Fi driver in `WIFI_PS_MIN_MODEM` or `WIFI_PS_MAX_MODEM`.

4. Modem Sleep: Efficient Networking Idle

Implementation:

```c
#include "esp_wifi.h"

void enable_modem_sleep() {
    // Minimal modem sleep
    esp_wifi_set_ps(WIFI_PS_MIN_MODEM);
}
```

Notes:

- The system is still running; **GPIO interrupts and tasks remain active**.
- No special wake-up setup is needed; **the system is always awake**.

5. Light Sleep: Balancing Power and Responsiveness

Use Case:

Idle device that must wake quickly (e.g., wake on UART or GPIO edge)

How It Works:

The CPU halts; RAM and RTC remain active. Peripherals can stay partially active (UART, timers, GPIO).

5. Light Sleep: Balancing Power and Responsiveness

Implementation (Timer-Based Wake-Up):

```c
#include "esp_sleep.h"
#include "esp_log.h"

void app_main(void) {
    esp_sleep_enable_timer_wakeup(5000000); // 5 seconds
    esp_light_sleep_start();
    ESP_LOGI("SLEEP", "Woke up from light sleep");
}
```

Implementation (GPIO-Based Wake-Up):

```c
#include "driver/gpio.h"

#define WAKE_GPIO GPIO_NUM_0

void app_main(void) {
    gpio_pullup_en(WAKE_GPIO);
    esp_sleep_enable_ext0_wakeup(WAKE_GPIO, 0); // Wake on LOW
    esp_light_sleep_start();
}
```

6. Deep Sleep: Ultra-Low Power Operation

Use Case:

Battery-powered sensor that wakes up periodically to sample and transmit data

How It Works:

Most components (CPU, RAM, peripherals) are powered off. Only RTC memory and wake-up sources are retained.

6. Deep Sleep: Ultra-Low Power Operation

Implementation (Timer + GPIO Wake-Up Example):

```c
#include "esp_sleep.h"
#include "driver/gpio.h"
#include "esp_log.h"

#define WAKE_GPIO GPIO_NUM_0

void app_main(void) {
    esp_sleep_enable_ext0_wakeup(WAKE_GPIO, 0);  // GPIO wake-up
    esp_sleep_enable_timer_wakeup(10000000);     // 10 seconds

    gpio_pullup_en(WAKE_GPIO);
    gpio_set_direction(WAKE_GPIO, GPIO_MODE_INPUT);

    ESP_LOGI("DEEP", "Entering deep sleep...");
    esp_deep_sleep_start();
}
```

⚠ **Important:**

- Upon waking from deep sleep, the chip **resets**, so `app_main()` runs fresh.
- Store data in **RTC memory** if needed

6. Deep Sleep: Ultra-Low Power Operation

Example):

```c
#include "esp_sleep.h"
#include "driver/gpio.h"
#include "esp_log.h"

#define WAKE_GPIO GPIO_NUM_0

void app_main(void) {
    esp_sleep_enable_ext0_wakeup(WAKE_GPIO, 0);  // GPIO wake-up
    esp_sleep_enable_timer_wakeup(10000000);     // 10 seconds

    gpio_pullup_en(WAKE_GPIO);
    gpio_set_direction(WAKE_GPIO, GPIO_MODE_INPUT);

    ESP_LOGI("DEEP", "Entering deep sleep...");
    esp_deep_sleep_start();
}
```

⚠ **Important:**

- Upon waking from deep sleep, the chip **resets**, so `app_main()` runs fresh.
- Store data in **RTC memory** if needed across deep sleep cycles.

7. Wake-Up Sources and Configuration

Wake Source	Light Sleep	Deep Sleep	Notes
Timer	✅	✅	Microsecond precision
RTC GPIO (EXT0)	✅	✅	One pin, level triggered
RTC GPIO (EXT1)	❌	✅	Multiple pins, any HIGH or all LOW
UART	✅	❌	Common for modems
Touch / ULP	❌	✅	Not yet fully supported on all C6 modules

Use `esp_sleep_enable_*()` APIs to configure the desired source.

8. Best Practices for Power-Optimized Design

- **Disable unused peripherals** (`esp_pm_configure()` for dynamic frequency scaling).
- Use **GPIO pullups/pulldowns** to avoid floating inputs during sleep.
- For **network-connected devices**, combine **modem sleep** with **event-driven architecture**.
- Use **deep sleep** for sensors with long idle periods.
- Store minimal state in **RTC memory** when resuming from deep sleep.

9. Example: Modem Sleep + GPIO Interrupt Wake

Let's write a working example that:

- Uses **Wi-Fi in modem sleep mode**,
- Goes into idle state,
- Wakes up on **GPIO interrupt** (e.g., button press).

```c
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_wifi.h"
#include "esp_log.h"
#include "driver/gpio.h"
#include "esp_event.h"
#include "nvs_flash.h"

#define WAKE_GPIO GPIO_NUM_0

static const char *TAG = "MODEM_SLEEP";

void IRAM_ATTR gpio_isr_handler(void* arg) {
    // Simple ISR to notify or resume task
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    vTaskNotifyGiveFromISR((TaskHandle_t)arg, &xHigherPriorityTaskWoken);
    if (xHigherPriorityTaskWoken) {
        portYIELD_FROM_ISR();
    }
}

void gpio_init(TaskHandle_t task_to_notify) {
    gpio_config_t io_conf = {
```

9. Example: Modem Sleep + GPIO Interrupt Wake

```
}

void gpio_init(TaskHandle_t task_to_notify) {
    gpio_config_t io_conf = {
        .intr_type = GPIO_INTR_NEGEDGE,     // Falling edge interrupt
        .mode = GPIO_MODE_INPUT,
        .pin_bit_mask = 1ULL << WAKE_GPIO,
        .pull_up_en = GPIO_PULLUP_ENABLE,
    };
    gpio_config(&io_conf);
    gpio_install_isr_service(0);
    gpio_isr_handler_add(WAKE_GPIO, gpio_isr_handler, (void*)task_to_notify);
}

void wifi_init_modem_sleep() {
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&cfg);
    esp_wifi_set_mode(WIFI_MODE_STA);
    esp_wifi_start();

    // Enable modem sleep
    esp_wifi_set_ps(WIFI_PS_MIN_MODEM);
    ESP_LOGI(TAG, "Wi-Fi in Modem Sleep mode (WIFI_PS_MIN_MODEM)");
}

void app_main(void) {
    ESP_ERROR_CHECK(nvs_flash_init());
    wifi_init_modem_sleep();

    TaskHandle_t task_handle = xTaskGetCurrentTaskHandle();
    gpio_init(task_handle);

    while (1) {
        ESP_LOGI(TAG, "Going idle... Press GPIO0 to wake.");
        ulTaskNotifyTake(pdTRUE, portMAX_DELAY); // Block until ISR notifies

        ESP_LOGI(TAG, "GPIO interrupt received! System is awake.");
        vTaskDelay(pdMS_TO_TICKS(1000)); // Simulate some work
    }
}
```

9. Conclusion

The **ESP32-C6** offers a powerful set of sleep modes that let engineers optimize power consumption based on real-world usage patterns. Whether you're building always-connected smart devices or energy-harvesting IoT sensors, knowing when and how to use **modem**, **light**, or **deep sleep** can make or break your product’s power efficiency.

By applying the techniques and examples outlined in this article, you'll be able to unlock the full low-power potential of the ESP32-C6 — making your devices smarter, leaner, and more responsive to the needs of modern embedded design.