IMP

Microcontroller power management

December 15, 2024

Daria Kinash (xkinas00)

Contents

1 Introduction

1.1 Topic and Motivation

Embedded systems are essential in modern technology, powering devices from smart home appliances to industrial machines. These systems are designed to perform specific tasks efficiently, reliably, and with minimal power consumption. Microcontrollers, the heart of embedded systems, can operate in different power modes such as Run, Light Sleep, and Deep Sleep, depending on the task requirements.

This project aims to explore and demonstrate how the ESP32 microcontroller (used in the Wemos D1 R32 development board) transitions between these power modes. It will also show how these transitions affect important resources like the CPU and GPIO. The topic is important because it helps optimize power usage while maintaining functionality, which is critical for battery-powered and energy-efficient devices.

1.2 Expected Goals and Benefits

The project's goal is to implement a simple embedded application that shows how the ESP32 manages power modes. The expected goals are:

- Understanding and demonstrating transitions between Run, Light Sleep, and Deep Sleep modes.
- Highlighting the impact of power mode transitions on system resources like CPU and GPIO.

The benefits of this project are twofold. First, it provides a basic understanding of low-power design principles, which can be applied to real-world embedded systems. Second, it shows how the ESP32 can be used to reduce power consumption in devices where saving energy is essential.

1.3 Expected Features

The solution includes the following key features:

1. **Power Mode Demonstration**:
    - Switching between Run, Light Sleep, and Deep Sleep modes.
    - Utilizing the built-in LED on the Wemos D1 R32 to visually indicate mode transitions.
2. **Resource Impact Analysis**:
    - Demonstrating CPU behavior and performance during each mode.
    - Analyzing GPIO state and functionality across the modes.
3. **Wake-Up Mechanisms**:
    - Timer-based wake-up from Light Sleep mode.
    - Timer-based wake-up from Deep Sleep mode.

1.4 Use Cases

This project's demonstration can be applied to these real-world scenarios:

1. **Battery-Powered Devices**:
    - Reducing power consumption in devices like sensors, wearables, or portable gadgets.
2. **Smart Home Systems**:
    - Efficiently managing resources in devices like smart thermostats, lighting, and security systems.
3. **IoT Applications**:
    - Extending the battery life of IoT nodes while keeping them responsive through wake-up mechanisms.

1.5 Verification and Significance

To verify the solution, the following steps will be taken:

- Checking LED behavior and console logs to ensure proper transitions between power modes.
- Confirming wake-up mechanisms.

2 Analysis and Proposal of the Solution

2.1 Operating Modes

The ESP32 microcontroller supports multiple power modes to optimize energy efficiency while maintaining required functionality. Below is a summary of the key operating modes and their properties:

Table 4-2. Power Consumption by Power Modes

Power mode	Description			Power Consumption
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 5-4 for details.
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered up.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	-			0.8 mA
Deep-sleep	The ULP coprocessor is powered up.			150 μA
	ULP sensor-monitored pattern			100 μA @1% duty
	RTC timer + RTC memory			10 μA
Hibernation	RTC timer only			5 μA
Power off	CHIP_PU is set to low level, the chip is powered down.			1 μA

Figure 1: Power Consumption by Operating Modes (Extracted from ESP32 Datasheet)

Table 5-4. Current Consumption Depending on RF Modes

Work Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	—	240	—	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	—	190	—	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	—	180	—	mA
Receive 802.11b/g/n	—	95 ~ 100	—	mA
Transmit BT/BLE, POUT = 0 dBm	—	130	—	mA
Receive BT/BLE	—	95 ~ 100	—	mA

Figure 2: Current Consumption Depending on RF Modes (Extracted from ESP32 Datasheet)

Run Mode

- **Description:** In Run Mode, the CPU operates at full performance, with all peripherals such as Wi-Fi, Bluetooth, GPIO, and ADC fully active. This mode is ideal for executing tasks requiring high computational power or active communication.
- **Power Consumption:** 20–240 mA (depending on activity, such as data transmission or reception).
- **Resource Behavior:**
  - **CPU:** Fully active and capable of executing tasks.
  - **GPIO:** Fully operational for input/output tasks.
  - **RTC:** Active for timekeeping and task scheduling.
  - **Memory:** Fully accessible for read/write operations.
  - **Wi-Fi/Bluetooth:** Fully operational for wireless communication.
- **Transitions:**
  - **To Light Sleep:** Quick transition (low latency, typically within milliseconds) when the system is idle or no longer requires full processing.
  - **To Deep Sleep:** Requires additional preparation, as most components will power down.
- **Transition Cost:** Minimal latency when transitioning to sleep modes.

Modem Sleep Mode

- **Description:** The CPU remains powered, but the wireless RF module (Wi-Fi/Bluetooth) is turned off to save power. Suitable for background processing tasks without active communication.
- **Power Consumption:** 20–68 mA depending on the CPU clock speed (80–240 MHz).
- **Resource Behavior:**
  - **CPU:** Fully operational for task execution.
  - **GPIO:** Fully functional for input/output operations.
  - **RTC:** Active for timekeeping and event scheduling.
  - **Wi-Fi/Bluetooth:** Disabled to reduce power consumption.
- **Transitions:** Quick and seamless transitions between Run and Modem Sleep Mode.
- **Transition Cost:** Very low latency, suitable for real-time applications.

Light Sleep Mode

- **Description:** The CPU is halted to save power, while peripherals such as RTC and GPIO remain active. This mode is used for periodic wake-ups or event-driven tasks.
- **Power Consumption:** 0.8 mA.
- **Resource Behavior:**
  - **CPU:** Halted, resumes operation upon wake-up.
  - **GPIO:** Active, can trigger wake-up events.
  - **RTC:** Active, supports periodic wake-ups or time-based tasks.
  - **Memory:** Retained, no data loss during this mode.
- **Transitions:**
  - **To Run Mode:** Triggered by a timer or external event.
- **Transition Cost:** Low latency, suitable for scenarios requiring periodic operation.

Deep Sleep Mode

- **Description:** The CPU and most peripherals are powered down, leaving only the RTC and ULP (Ultra Low Power) coprocessor active. This mode is ideal for applications that require long idle periods with infrequent wake-ups.
- **Power Consumption:** 10 µA (extremely low).
- **Resource Behavior:**
  - **CPU:** Powered down and inactive.
  - **GPIO:** Configurable as wake-up sources.
  - **RTC:** Active for timer-based wake-ups.
  - **Memory:** Limited retention; data in RAM is lost unless explicitly stored in RTC memory.
- **Transitions:**
  - **To Run Mode:** Triggered by external events or timers.
- **Transition Cost:** Higher latency due to the need to reinitialize peripherals and restart the CPU.

Hibernation Mode

- **Description:** Only the RTC remains active, consuming the least amount of power. Suitable for long-term inactivity with very rare wake-ups.
- **Power Consumption:** 5 µA.
- **Resource Behavior:**
  - **CPU:** Powered down and inactive.
  - **RTC:** Active for timekeeping and wake-up events.
  - **Memory:** No retention; all data in RAM is lost.
- **Transitions:**
  - **To Run Mode:** Triggered by an RTC alarm or external wake-up source.
- **Transition Cost:** High latency due to the need to fully restart the system.

2.2 Application Components and Interactions

Key Equipment:

- **Internal Peripherals:**
  - **GPIO:** Used for controlling the built-in LED to indicate system activity and transitions.
  - **RTC Timer:** Configured to enable periodic wake-ups from Light Sleep and Deep Sleep modes, ensuring power-efficient functionality.
- **External Peripherals:**
  - None required, as the application use the built-in features of the ESP32 microcontroller.

Interconnections and Control:

- The **GPIO** is used to control the built-in LED. For example, different blinking patterns are programmed to represent Run, Light Sleep, and Deep Sleep modes.

- The **RTC Timer** is configured as the primary wake-up source. It wakes the system from Light Sleep after 10 seconds and from Deep Sleep after 5 seconds.
- Power mode transitions and wake-up configurations are managed dynamically using the ESP-IDF framework.
- All interactions are handled through software using the ESP32's internal peripherals and ESP-IDF APIs.

2.3 API Usage

- `driver/gpio.h`: For configuring and controlling the built-in LED.
- `esp_sleep.h`: For configuring and managing power mode transitions and RTC wake-up sources.
- `freertos/FreeRTOS.h`, `freertos/task.h`: For creating delays, managing task execution, and logging CPU utilization statistics.
- `esp_timer.h`: Used to simulate CPU workload and measure execution time, highlighting the CPU's activity in Run Mode.

3 Own solution

3.1 Behavior and Implementation

The system demonstrates transitions between three power modes (Run, Light Sleep, and Deep Sleep) and the impact on resources like CPU and GPIO. Below is a detailed description of each mode's behavior and implementation:

1. **Run Mode:** The system starts in Run Mode, where the CPU is fully active. During this mode:
   - The LED blinks once to visually indicate the active state.
   - The system performs active tasks, such as simulating a CPU workload and logging runtime statistics for CPU utilization.
   - This mode demonstrates how the CPU and GPIO are fully operational. After 2 seconds, the system transitions to Light Sleep Mode.
2. **Light Sleep Mode:** After completing tasks in Run Mode, the system enters Light Sleep Mode. In this mode:
   - The CPU halts, but peripherals like RTC and GPIO remain active for event-driven tasks.
   - The LED blinks five times to indicate the transition into Light Sleep.
   - The RTC timer is configured to wake the system after 10 seconds.
   - Upon waking up, the LED blinks twice to signal the return to an active state. This step demonstrates the use of the RTC as a low-power wake-up source and the CPU resuming from a halted state.
3. **Deep Sleep Mode:** Following the Light Sleep wake-up, the system transitions to Deep Sleep Mode. Here:
   - The LED blinks three times to indicate the preparation for Deep Sleep.
   - The CPU and most peripherals are powered down, leaving only the RTC active.
   - The RTC timer is configured to wake the system after 5 seconds. This demonstrates the lowest power consumption scenario, where the CPU is entirely powered off and resumes operation only upon a timer-based wake-up.

3.2 System Output

To validate the implementation, the system behavior was monitored using the ESP-IDF serial monitor. Below is an example output captured during a test run:

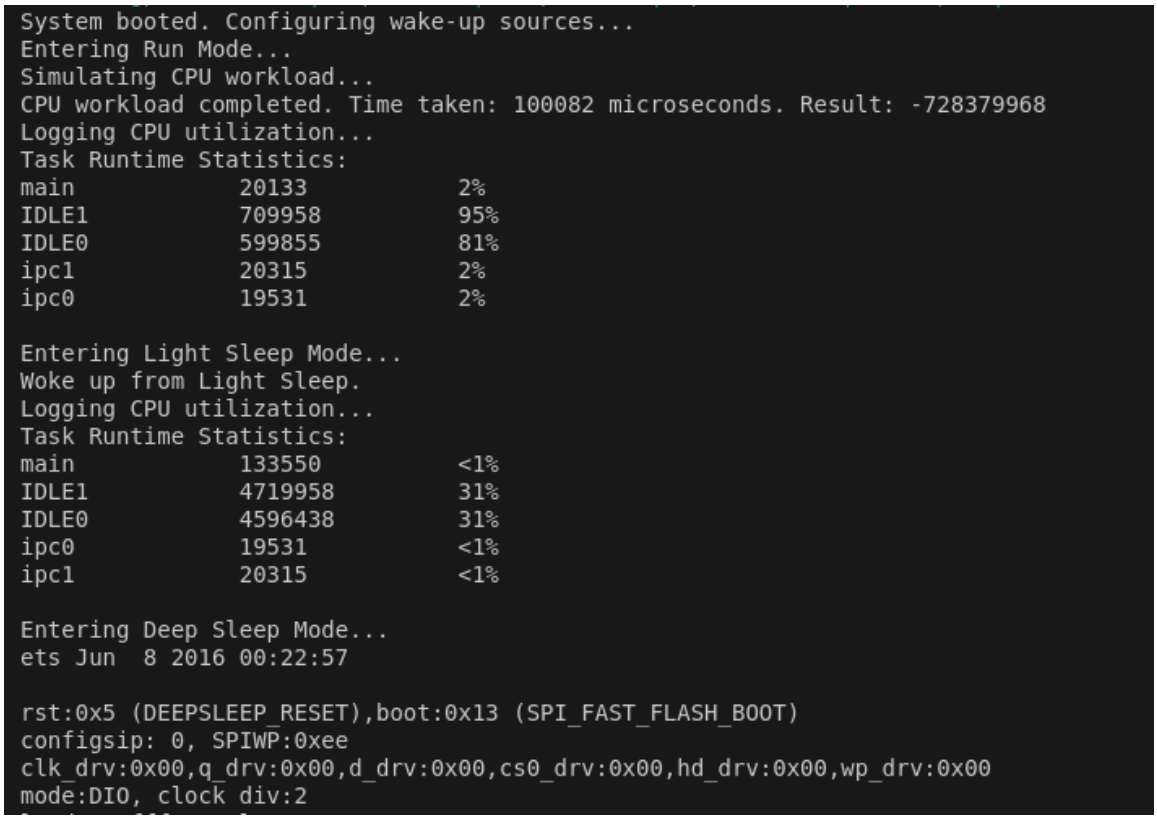

Figure 3: System Output Captured from ESP-IDF Monitor

- The system starts in **Run Mode**, where it performs a simulated CPU workload. The total time taken by the workload and the result of the computation are displayed. CPU utilization statistics are also logged, showing the distribution of time spent by various tasks.
- The system then transitions to **Light Sleep Mode**. In this mode the CPU is halted, as reflected in the significantly lower activity of the main task. CPU utilization statistics confirm that the system remains idle, with minimal power usage. Upon waking up from Light Sleep after 10 seconds, the system logs the wake-up event and transitions to Deep Sleep.
- In **Deep Sleep Mode**, the CPU is entirely powered down. The system resets upon waking, as shown by the `DEEPSLEEP_RESET` in the monitor output.

3.3 Problems Encountered and Their Resolution

Problem: Maintaining Wake-Up Sources After Sleep Transitions Initially, the system encountered issues where wake-up sources were cleared upon waking from Light Sleep or Deep Sleep. This behavior caused failures during subsequent attempts to re-enter sleep modes.

Solution: Reconfiguring Wake-Up Sources The issue was resolved by dynamically reconfiguring wake-up sources before each sleep mode transition. Specifically `configure_wakeup_sources_light_sleep()` and `configure_wakeup_sources_deep_sleep()` were introduced to ensure RTC wake-up sources were correctly set for Light Sleep and Deep Sleep, respectively.

Final Evaluation

The solution successfully demonstrates the following features:

- Smooth transitions between Run, Light Sleep, and Deep Sleep modes.
- Use of the built-in LED to indicate power mode transitions and wake-up states.
- Reliable wake-up mechanisms using timers for both Light Sleep and Deep Sleep modes.

Fulfillment of Objectives: The project fulfills its primary objective of demonstrating the ESP32's power-saving capabilities and resource utilization. By implementing and testing transitions between power modes, it provides a practical understanding of low-power design principles. The modular nature of the solution allows for future enhancements, such as integrating additional wake-up sources or external peripherals for expanded functionality.

Self-Evaluation: The project meets the objectives of showing ESP32 power modes and transitions. I feel the project helped me develop a solid understanding of power management in embedded systems. However, I recognize there is room to explore additional advanced features like the ULP coprocessor or integrating more peripherals.

Overall Evaluation: The solution fulfills the assignment's requirements by effectively demonstrating energy-efficient design principles in embedded systems. Overall, this project was a valuable learning experience and a good demonstration of low-power embedded design with the ESP32.

Use of Information Sources

The following resources were critical in designing and implementing the solution:

- Espressif Systems. *ESP32 Technical Reference Manual*. Available at: `https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf`.
- Random Nerd Tutorials. *ESP32 Deep Sleep with Arduino IDE*. Available at: `https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/`.
- ESP-IDF Documentation. Available at: `https://docs.espressif.com/projects/esp-idf/en/latest/`.