1.3 Microcontroller Architecture

Besides the differences in the type of OS used, there are also differences in the microcontrollers used. In this IoT lab, the ESP-32 microcontroller is used, which differs from the Arduino Uno used in the Embedded Systems lab. Look at the table below for a comparison between the two microcontrollers. [3]

What-Are-the-Advantages-of-EPP32-Over-Arduino-UNO

In addition to the increase in RAM and wireless communication modules, a key difference between the ESP32 and the Arduino UNO is the number of cores.

This means the ESP32 can achieve true parallelism in executing its tasks.

Why is this important?

First, let's look at the program structure used on the Arduino Uno, which lacks parallelism and runs on bare metal, often using a "Superloop" architecture. In this architecture, a single setup process is performed to initialize components before entering a loop that executes all tasks. During the loop, interrupts can be processed based on external events. For many use cases, this architectural structure is sufficient to complete the required tasks.

a5ac711c-328b-48f6-9d8c-f207e3abb184

Tasks in this architecture are executed sequentially. Consequently, if there are many tasks to run, there is a possibility of missing deadlines.

For example, if you create a device to read sensor data (like temperature or heart rate) and simultaneously upload it in real-time to a server via a WiFi connection, the superloop architecture on an Arduino Uno would struggle. This is because the process of communicating with the server (e.g., an HTTP request) takes a relatively long time and will block the main loop. As a result, sensor readings could be delayed or even missed entirely. This means that if sensor data needs to be read with precision (e.g., every few microseconds or milliseconds), this architecture cannot guarantee that timing.

The Solution?

The ESP-32 can leverage parallelism to complete these tasks so they run concurrently.

GPOS systems are designed to run multiple processes at once, commonly supported by multi-tasking and multi-threading, so the user can execute several tasks simultaneously. In general, timing deadlines for tasks on a GPOS are not crucial, and delays can be tolerated as long as they are not visible to the user.

In this context, the RTOS acts as the operating system that manages resource allocation and scheduling for each task. [1]. 16adc64e-5857-4505-b27e-e66b375037a1

Based on the image above, an RTOS can divide program execution into several tasks. For instance, Task 1 is responsible for reading data from a sensor, Task 2 is responsible for uploading data to a server, while Task 3 can be used for a periodic process like logging.

Each task has its own priority, and through the RTOS API, we can configure it to run on a specific thread or schedule it as needed. In this way, the RTOS ensures that each task can be executed concurrently and on schedule, without interfering with each other.

This does not mean the number of tasks in an RTOS is limited to the two cores of the ESP-32. Rather, the RTOS can manage priorities and perform event scheduling and time-slicing to ensure the timeliness and deadlines of each task are met according to its priority.

Takeaway

Although an RTOS offers many advantages, it doesn't mean the bare-metal (super loop) architecture is always unsuitable. On 8-bit microcontrollers like the Arduino UNO (ATmega328p), the bare-metal approach is actually more efficient because the overhead of an RTOS scheduler is too large for the available resources. With bare metal, simple applications like reading a sensor, turning on an LED, or serial communication can run more lightly without additional overhead. Consequently, microcontrollers that utilize an RTOS tend to have higher minimum specifications.

cbf78b6c-3d84-4222-a636-2b619714ef66

When moving to more powerful microcontrollers like the ESP32, the use of an RTOS becomes increasingly relevant. Besides having dual-core capabilities, the ESP32 is designed for IoT applications, thus requiring multitasking capabilities so that the WiFi and Bluetooth stacks can run concurrently with the user's application, while also ensuring the deadlines of its operations are met.

Revision #2
Created 2025-08-29 13:26:43 UTC by NS
Updated 2025-08-29 13:33:35 UTC by NS