ESP-IDF FreeRTOS

FreeRTOS Overview

ESP-IDF FreeRTOS Overview

- Lesson content
  - FreeRTOS documentation and resources
  - ESP-IDF RTOS differences from Vanilla FreeRTOS and ESP-IDF application startup
  - FreeRTOS task states
  - ESP-IDF FreeRTOS task creation and xTaskCreatePinnedToCore API
  - Use of vTaskDelay

FreeRTOS Resources

- FreeRTOS is a real-time operating system kernel for embedded devices.
  - What is an RTOS? → https://www.freertos.org/about-RTOS.html
  - FreeRTOS Books → https://www.freertos.org/Documentation/RTOS_book.html
  - RTOS Fundamentals → https://www.freertos.org/implementation/a00002.html
  - ESP-IDF FreeRTOS → https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html

ESP-IDF FreeRTOS

- Differences from Vanilla FreeRTOS
  - The ESP-IDF FreeRTOS is adapted for multi-core support.
  - Unlike Vanilla FreeRTOS, we don't have to call vTaskStartScheduler.
  - FreeRTOS task stack size is specified in Bytes in ESP-IDF FreeRTOS not Words.
- Application Startup Flow
  - app_main and application startup → https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/startup.html

FreeRTOS Task Creation

- Create tasks using the FreeRTOS xTaskCreate APIs
  - We must include freertos/task.h.
  - xTaskCreate → Lets the ESP-IDF FreeRTOS choose which core the task runs on.
  - xTaskCreatePinnedToCore → Allows for specifying which core the task runs on.

xTaskCreatePinnedToCore Parameters

```
BaseType_t xTaskCreatePinnedToCore(TaskFunction_t pvTaskCode, const char *const pcName, const uint32_t usStackDepth,
void *const pvParameters, UBaseType_t uxPriority, TaskHandle_t *const pvCreatedTask, const BaseType_t xCoreID)
```

- pvTaskCode → This is the custom C function (task) that runs in an infinite loop.
- pcName → Descriptive name for the task. Only used as a debugging aid.
- usStackDepth → Memory in bytes that should be allocated by the kernel to the task.
- pvParameters → Optional parameter. Pointer that can be used by the task.
- uxPriority → The priority at which the task should run on. Higher priority number takes precedence.
- pvCreatedTask → Optional task handle by which the created task can be referenced, e.g., if you need to use various FreeRTOS APIs, like, vTaskDelete.
- xCoreID → The core of the ESP32 the task is assigned to (Core 0 or Core 1).

FreeRTOS Tasks

- FreeRTOS task states → https://www.freertos.org/RTOS-task-states.html
  - Running → Executing and utilizing the processor.
  - Ready → Able to execute (not in the Blocked or Suspended states).
  - Blocked → Because of temporal event, e.g., call to vTaskDelay or external event, e.g., waiting for a queue, semaphore, event group, notification or semaphore event.
  - Suspended → There is no timeout and tasks only enter and exit the Suspended state when explicitly commanded to do so using vTaskSuspend or xTaskResume API calls.

vTaskDelay

- vTaskDelay is used to send a task into Blocked state for a set number of Ticks.
  - The actual time that the task remains blocked depends on the tick rate.
  - The constant portTICK_PERIOD_MS can be used to calculate real time from the tick rate – with the resolution of one tick period.

```
void vTaskFunction( void * pvParameters )
{
// Block for 500ms.
const TickType_t xDelay = 500 / portTICK_PERIOD_MS;

  for( ;; )
  {
      // Simply toggle the LED every 500ms, blocking between each toggle.
      vToggleLED();
      vTaskDelay( xDelay );
  }
}
```

Next lesson