Practical Sections

Setting Up FreeRTOS on ESP-32

Two cores in ESP-32 let this low-power microcontroller operate:

- `CPU0`: Handles BLE, Bluetooth, and Wi-Fi wireless protocols.
- `CPU1`: Executes code for user apps.

Installing and configuring the ESP-32 Arduino Core:
1. Obtain the most recent Arduino IDE and install it.
2. Launch Arduino IDE then go to File / Preferences. Enter in the field **Additional Boards Manager URLs**:

https://dl.espressif.com/dl/package_esp32_index.json

3. Go to Tools / Board / Boards Manager, look for **esp32**, and install the most recent release from Espressif Systems.
4. Go to Tools / Board / ESP32 Arduino and pick the right board (like **ESP32 Dev Module** or **ESP32 Wrover Module**).

All about FreeRTOS APIs

1. xTaskCreate()

- Purpose: Creates a new task and dynamically allocates the required memory. Returns a handle to the created task, or NULL if creation fails. Syntax:

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

Parameters

1. **pvTaskCode**: Pointer to the function that implements the task. The function must have a prototype of void vTaskCode(void * pvParameters).
2. **pcName**: Descriptive name of the task (helps with debugging).
3. **usStackDepth**: Stack size in words (not bytes) for the task.
4. **pvParameters**: Pointer to the arguments passed to the task function.
5. **uxPriority**: Priority of the task execution (higher number = higher priority).
6. **pvCreatedTask**: Pointer to the variable that will receive the created task handle.

Return Value

1. **pdPASS**: The task was successfully created and added to the ready list.
2. **errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY**: The task could not be created because there is not enough heap memory available.

---

2. xTaskCreatePinnedToCore()

- Purpose: Similar to xTaskCreate(), except that you can specify the core number on which the task will run. This is useful for performance reasons. Here is the syntax of the function:

```
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
);
```

Parameters

Same as xTaskCreate(), except for:

1. **xCoreID**: The core number on which the task should run. Can be 0 or 1 for a dual-core target ESP, or any other valid number of cores for a multi-core target ESP, i.e. 2 cores, 3 cores, etc.

Return Value

Same as xTaskCreate().

---

3. vTaskDelete()

- Purpose: Delete a task and free the memory allocated by it; delete other tasks. The syntax of this function is as follows:

```
void vTaskDelete(TaskHandle_t xTask);
```

Parameters

1. **xTask**: The handler of the task to be deleted. Passing NULL will delete the calling task.

Return Value

None.

---

4. vTaskDelay()

- Purpose: Causes the calling task to block for the specified number of ticks (ms). The syntax of this function is as follows:

```
void vTaskDelay(const TickType_t xTicksToDelay);
```

Parameters

1. **xTicksToDelay**: The number of ticks to delay. One tick = the unit of time specified by the configTICK_RATE_HZ configuration constant in FreeRTOSConfig.h.

Return Value

None.

---

5. vTaskDelayUntil()

- Purpose: This function blocks the calling task for a specified period of time, relative to the time the function was last called. In other words, it can be used when you want a task to run with a fixed frequency. The syntax of this function is as follows:

```
void vTaskDelayUntil(TickType_t * const pxPreviousWakeTime, const TickType_t xTimeIncrement);
```

Parameters

1. **pxPreviousWakeTime**: Pointer to a TickType_t variable that stores the time when the task was last unblocked. This variable must be initialized with the current time before the first call to the vTaskDelayUntil() function. The function will update the variable with the current time after each call.
2. **xTimeIncrement**: The time period between executions (cycle time) in ticks. The task will be unblocked at times (pxPreviousWakeTime + xTimeIncrement), (pxPreviousWakeTime +

xTimeIncrement2), and so on.

Return Value

None.

---

6. vTaskSuspend()

- Purpose: This function temporarily suspends a task, preventing it from being scheduled until it is reactivated by another task. The syntax of this function is as follows:

```
void vTaskSuspend(TaskHandle_t xTaskToSuspend);
```

Parameters

1. **xTaskToSuspend**: The handle of the task to be suspended. Passing a NULL value will suspend the calling task.

Return Value

None.

---

7. vTaskResume()

- Purpose: This function reactivates a task that has been paused by vTaskSuspend(). The syntax of this function is as follows:

```
void vTaskResume(TaskHandle_t xTaskToResume);
```

Parameters

1. **xTaskToResume**: The handle of the task to be reactivated.

Return Value

None.

---

8. vTaskPrioritySet()

- Purpose: Change the priority of a task. The syntax of this function is as follows:

```
void vTaskPrioritySet(TaskHandle_t xTask, UBaseType_t uxNewPriority);
```

Parameters

1. **xTask**: The handle of the task whose priority will be changed. Passing a NULL value will change the priority of the calling task.
2. **uxNewPriority**: The new priority for the task.

Return Value

None.

---

9. uxTaskPriorityGet()

- Purpose: Returns the priority of a task. The syntax of this function is as follows:

```
UBaseType_t uxTaskPriorityGet(TaskHandle_t xTask);
```

Parameters

1. **xTask**: The handle of the task whose priority is to be obtained. Passing a NULL value will return the priority of the calling task.

Return Value

- The priority of the task.

---

10. eTaskGetState()

- Purpose: Returns the status of a task. The syntax of this function is as follows:

```
eTaskState eTaskGetState(TaskHandle_t xTask);
```

Parameters

1. **xTask**: The handle of the task whose status is to be obtained.

Return Value

1. **xTask**: The handle of the task whose status is to be returned.

The following are the possible task states:

Status	Description
eRunning	The task is running.
eReady	The task is ready to run.

eBlocked	The task is blocked, waiting for an event.
eSuspended	The task is temporarily suspended.
eDeleted	The task has been deleted.
eInvalid	The task marker is invalid.

Revision #11
Created 2025-09-06 06:40:52 UTC by PI
Updated 2025-09-06 18:18:28 UTC by PI