

freeRTOS con ESP32 con Interrupciones

FreeRTOS es un sistema operativo en tiempo real que ofrece soporte para interrupciones de hardware. Las interrupciones permiten que el procesador responda a eventos de hardware que ocurran en cualquier momento, lo que permite una respuesta más rápida y eficiente a las señales de entrada.

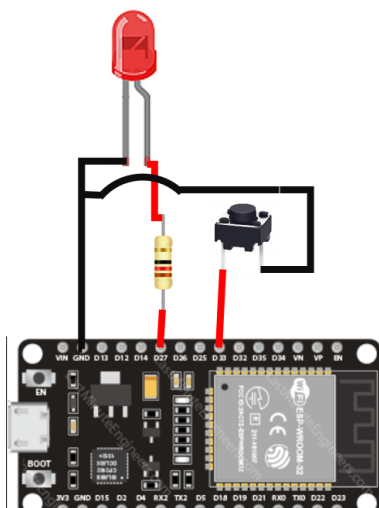
FreeRTOS ofrece un modelo de programación basado en tareas, que permite a los desarrolladores definir tareas independientes que se ejecutan en paralelo. Cada tarea puede tener un nivel de prioridad diferente, lo que permite al desarrollador definir qué tarea se debe ejecutar en primer lugar en caso de que varias tareas estén listas para ejecutarse al mismo tiempo.

Las interrupciones con FreeRTOS se manejan a través de una función de controlador de interrupción, que se configura para activarse cuando se produce una interrupción de hardware específica. Cuando se produce una interrupción, la función de controlador de interrupción se ejecuta y puede notificar a una tarea específica de FreeRTOS que debe reanudar la ejecución. Esto se logra a través de la función `vTaskResumeFromISR()`, que permite que una tarea suspendida se reanude desde una interrupción.

En la siguiente practica Incorporaremos la interrupción para alternar un led conectado con la placa ESP32 con un botón.

Materiales Necesarios


- ESP32
- Cable USB
- Computadora con el IDE de Arduino instalado
- Push button
- Led
- Resistencia
- Cable o jumpers



Este es el diagrama que ocuparemos conectaremos el boton a tierra y al puerto 33 y el led la parte negativa va a tierra y la positiva a una resistencia al puerto 27.


Desarrollo del programa

1. Para empezar incluimos las librerías que utilizaremos para esta práctica




```
1  #include <freertos/FreeRTOS.h>
2  #include <freertos/task.h>
```

2. Definimos dos variables enteras que almacenan los números de los pines del LED y el botón, respectivamente.




```
1  int LED_PIN = 27;
2  int PUSH_BUTTON_PIN = 33;
```

3. Se declara una variable de tipo TaskHandle_t llamada ISR y se inicializa con NULL. Esta variable se utilizará más adelante para hacer referencia a la tarea que se crea para controlar el LED.



```
1  TaskHandle_t ISR = NULL;
```

4. Se define una función de interrupción para manejar la interrupción generada por el botón. La función simplemente llama a la función xTaskResumeFromISR() para reanudar la tarea de control del LED.



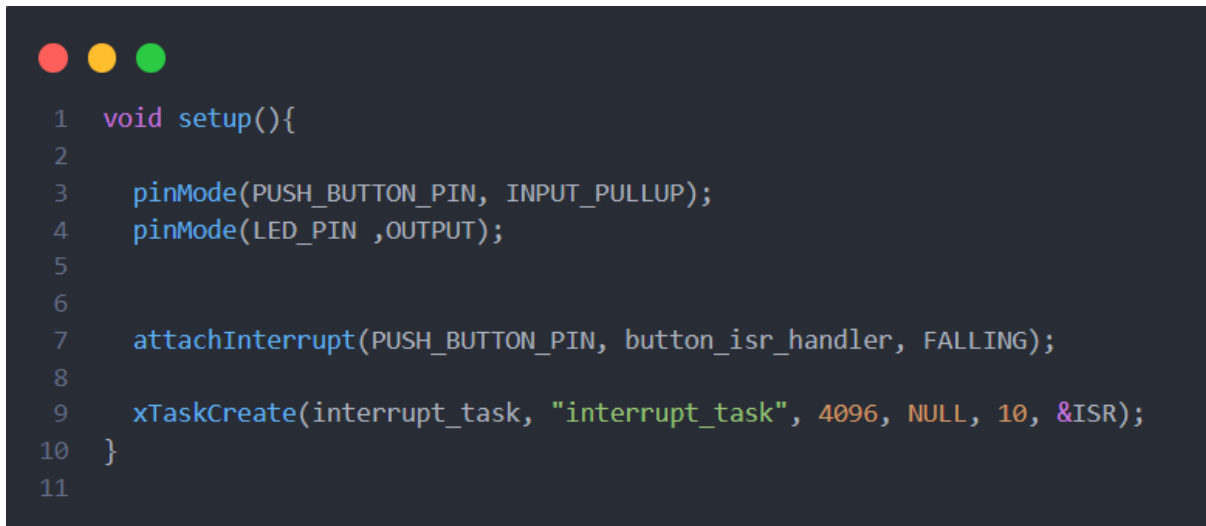
```
1  void IRAM_ATTR button_isr_handler(){
2      xTaskResumeFromISR(ISR);
3  }
```

5. Se define la tarea que controla el LED. La tarea está en un bucle infinito y se suspende a sí misma hasta que se produce una interrupción de botón. Cuando se reanuda, cambia el estado del LED y envía un mensaje por el puerto serie.



```
1 void interrupt_task(void *arg){
2     bool led_status = false;
3     while(1){
4         vTaskSuspend(NULL);
5         led_status = !led_status;
6         digitalWrite(LED_PIN, led_status);
7         printf("Button pressed!\n");
8     }
9 }
```

6. En esta función se ejecuta una sola vez al inicio del programa. Configura los pines del botón y el LED como entrada y salida, respectivamente. Luego, se llama a la función `attachInterrupt()` para configurar la interrupción del botón. Finalmente, se crea la tarea `interrupt_task` utilizando la función `xTaskCreate()`.



```
1 void setup(){
2
3     pinMode(PUSH_BUTTON_PIN, INPUT_PULLUP);
4     pinMode(LED_PIN ,OUTPUT);
5
6
7     attachInterrupt(PUSH_BUTTON_PIN, button_isr_handler, FALLING);
8
9     xTaskCreate(interrupt_task, "interrupt_task", 4096, NULL, 10, &ISR);
10 }
11
```

7. `void loop(){...}`: esta función se ejecuta en un bucle infinito. No contiene ninguna instrucción porque todo el trabajo se realiza en la tarea `interrupt_task`.

8. Subes el código a tu ESP32 y si todo está bien podrás ver en el monitor serial button pressed y el led encendido cada vez que presiones el push button.

Y ¡FELICIDADES creaste una interrupción utilizando freeRTOS!.

Puedes encontrar el código completo y los pdf en la sig. página de GitHub:

https://github.com/GabrielCorUs/freeRTOS_ESP32/tree/main/ISRfreeRTOS_ESP32