



# WiFi Reset Button Implementation

---

Using the BOOT button to disconnect ESP32 and clear the credentials

The background features a complex network of thin, light-colored lines and small circles, forming a web-like structure. Overlaid on this are several translucent triangles of various sizes and orientations. The overall aesthetic is technical and minimalist.

# **Our Implementation**

---

# Disconnecting WiFi and Clearing Credentials Using the BOOT Button on the DevKit

- About the Implementation

- The BOOT button will be configured to generate an interrupt on I00.
- When the interrupt occurs, a message will be sent to the WiFi Application about the user request to disconnect/clear credentials.
- Upon receiving the message, the WiFi Application will check if there really is an active connection prior to disconnecting and clearing credentials.

The background features a complex network of thin, light-colored lines and small circles, forming a web-like structure. Overlaid on this are several triangles of various sizes and orientations, some of which are filled with a light yellow color. The overall aesthetic is technical and modern.

# **Additional Information About the Implementation**

GPIO Interrupt on the BOOT Button and Binary Semaphores

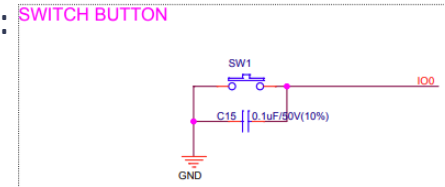
---

# GPIO Interrupt for WiFi Reset Button

- About

- Background info: an interrupt is a signal that indicates the occurrence of a specific event that requires immediate attention and blocks the normal program execution to run an interrupt service routine (ISR) – which reacts to the event that occurred.
  - Note: ISRs must have a short execution.*

- The GPIO interrupt in our case, we are configuring the interrupt to trigger on a falling edge signal, because when you press the BOOT button, the pin connects to ground as shown here in the ESP32 DevKit-C schematic:



- Due to the short execution time of ISRs; a **binary semaphore** will be used to notify a FreeRTOS task, which will handle the actions performed when the button is pressed.

# About Binary Semaphore APIs from ESP-IDF

- Binary Semaphore (from ISR)
  - Binary semaphores are semaphores which can assume the values of 0 and 1 only → Hence, they can be used as a signaling mechanism.
  - The ESP-IDF API for Semaphores can be found here → <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html#semaphore-api>
  - Within the ISR, the API we need is → [xSemaphoreGiveFromISR](#).
  - Within the FreeRTOS task, the API we need is → [xSemaphoreTake](#).



# **ESP-IDF APIs Used for the WiFi Reset Button GPIO Configuration**

---

---

# Utilizing ESP-IDF for GPIO Interrupt Configuration

- Suggested Reading & About

- About GPIO ESP-IDF → <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html>
  - Set the direction of the GPIO we are working with as an input → [gpio\\_set\\_direction](#).
  - Set the interrupt type of the GPIO e.g., Falling Edge ([GPIO\\_INTR\\_NEGEDGE](#)) → [gpio\\_set\\_intr\\_type](#).
  - Install the ISR service → [gpio\\_install\\_isr\\_service](#).
  - Specify the ISR that will run when the interrupt for that pin is triggered → [gpio\\_isr\\_handler\\_add](#).
  - Define the ISR and place it into the IRAM region → Using the [IRAM\\_ATTR](#) (attribute).
-



The background is a light cream color with a complex, abstract geometric pattern. It features a network of thin, light brown lines connecting small circular nodes. These nodes and lines form various triangular shapes of different sizes and orientations. Some triangles are filled with a very light, pale yellow color, while others are just outlines. The pattern is denser on the left side and becomes sparser towards the right. In the upper right corner, there are several small, isolated clusters of nodes without connecting lines.

**Let's get started!**

---