Tutorial 07 – Interrupts and External Wakeup Sources

Make sure you complete **Setup** and
Test Your Adafruit Feather Huzzah32 Connection
document before starting this Tutorial 07.

Learning Objectives

1. Understand the use of the Single Pole Double Throw (SPDT) slide switch, the micro servo and the RGB LED.
2. Understand the basics of programming, i.e., generating random numbers (i.e., `random()`), and compound operators (i.e., `-=`, `+=`).
3. Implement function calls related to hardware interrupt, i.e., `attachInterrupt()`, on Adafruit Feather Huzzah32.
4. Implement function calls related to using software interrupt, i.e., `timerBegin()`, `timerAttachInterrupt()`, `timerAlarmWrite()`, `timerAlarmEnable()`, on Adafruit Feather Huzzah32.
5. Implement built-in functions from `ESP32Servo.h` library, i.e., `servo.attach()`, `servo.write()`, in the code in order to control the micro servo using Adafruit Feather Huzzah32.

Theory

One common problem with interrupts is that they often get triggered multiple times for the same event, e.g., due to the mechanical delay setting on PIR sensors (see Figure 8 (b) in Tutorial 06). Figure 1 also illustrates another example of a possible mechanical cause of multiple interrupts from a tactile switch button.

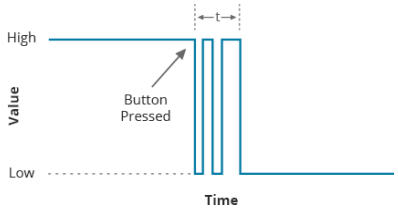


Figure 1: A possible voltage signal when a tactile switch button is pressed.

In Figure 1, a variable voltage signal when a tactile switch button is pressed can be caused from the mechanical parts within a tactile switch button coming into contact several times before they settle into a particular state. This is purely a mechanical phenomenon which is known as a switch bounce, i.e., similar to a ball bounces several times, when it is dropped before finally landing on the ground.

Debouncing or the process to eliminate switch bounce can be achieved through hardware or software. Hardware debouncing is outside the scope of 5COM2004. However, as shown in Figure 2, software debouncing can be achieved by temporarily ignoring further interrupts for a short period of time after the first interrupt is triggered.

```
void IRAM_ATTR isr() {
    button_time = millis();
    if (button_time - last_button_time > 250)
    {
        button1.numberKeyPresses++;
        button1.pressed = true;
        last_button_time = button_time;
    }
}
```

Figure 2: A software debouncing example (Last Minute Engineers, 2022).

From Figure 2, the 3 variables, i.e., `numberKeyPresses`, `pressed` and `last_button_time` are updated only once every 250 milliseconds after the last time the button is pressed, or the interrupt being triggered.

RGB LED is a combination of 3 LEDs in one and can be used to produce almost any colours through configuring the intensity of red (R), green (G), and blue (B) light. The light intensity ranges between 0-255 using `analogWrite()`. For example, you can produce purely red light, by setting blue and green at the lowest intensity and red at the highest intensity. Figure 3 illustrates an example of colours when mixing red, green, and blue together.

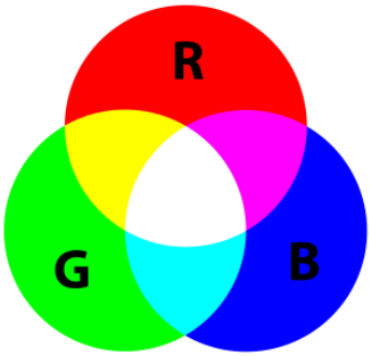
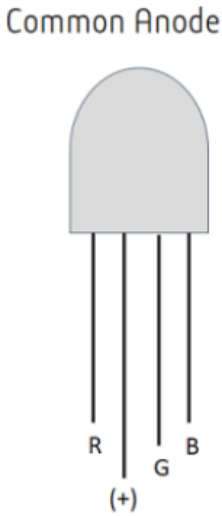
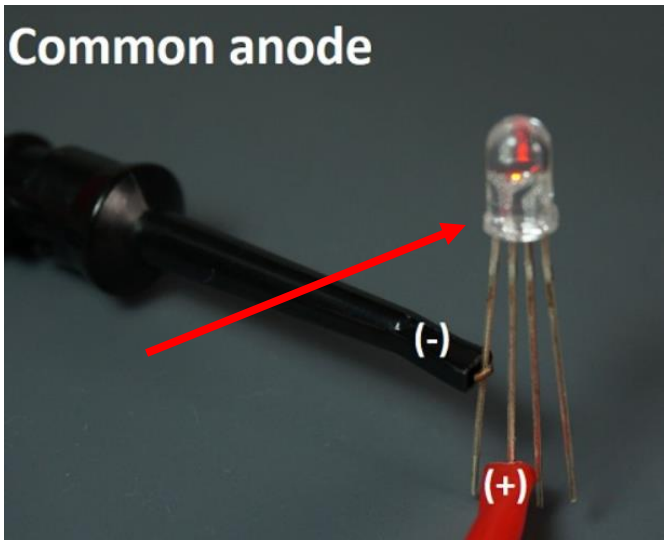


Figure 3: An example of colours when red, green and blue is mixed (Random Nerd Tutorial, 2022).

There are two kinds of RGB LEDs, i.e., common anode and common cathode. The RGB LEDs used in 5COM2004 is a common anode RGB LED, where all three LEDs share a positive connection (i.e., anode). The other 3 pins are for each of the LEDs. Figure 4 (a) illustrates the pins and (b) illustrates the flat side of the RGB LEDs to help identifying the order of the pins. If you lift up the LED and look at it through some light, you will also see a flag like shape (similar to the LED you have been using in previous tutorials) on the same side as the flat side.



(a)



(b)

Figure 4: (a) the RGB LED pins, (b) flat side of the RGB LEDs (Random Nerd Tutorial, 2022).

A Servo motor or Servomotor is one of the most important actuators in robotics with a wide range of applications from radio control aeroplanes to automatic door locks. Servo motors use in 5COM2004 are limited in rotation to 180 degrees. Pulse Width Modulation (PWM) is used to control the motor shaft position. Figure 5 illustrates the pins on the Servo motor.

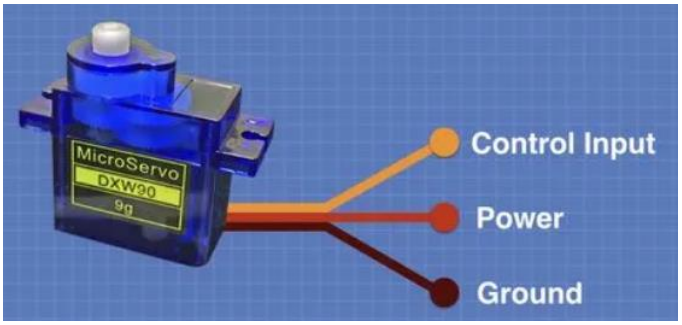


Figure 5: Servo motor's pins (DroneBot Workshop, 2022).

Servo motors use servomechanism, i.e., uses feedback to control its motor and final position. This error-sensing negative feedback to correct the action of a mechanism can control mechanical position, speed, attitude or any other measurable variable.

Read the following ESP32 Servo reference pages to better understand functions to control Servo motors, specifically `attach()` and `write()`.

1. ESP32Servo.h: https://github.com/madhephaestus/ESP32Servo/blob/master/src/ESP32Servo.h

A Single Pole Double Throw (SPDT) switch is a three-terminal (pin) switch which has a single input that can switch between two outputs. The number of poles signifies the number of circuits the switch can control, whereas the number of throws signifies the number of positions each pole of the switch can connect to. Figure 6 illustrates an example of SPDT swich.

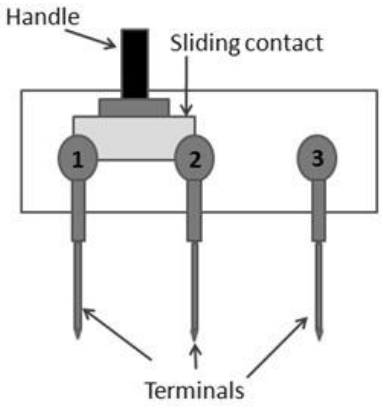


Figure 6: A Single Pole Double Throw (SPDT) switch (Elprocus, 2022).

The SPDT switches used in 5COM2004 have 1 common pin (i.e., the middle pin to be connected to 3V) and 2 output pins on either side which compete for connection toward the common. The “on” state is when the switch closes the circuit, and therefore allowing the current to flow through the rest of the system.

Reference

Last Minute Engineers. (2022). Configuring and Handling ESP32 GPIO Interrupts in Arduino IDE. [Website] Available at: https://lastminuteengineers.com/handling-esp32-gpio-interrupts-tutorial/

Random Nerd Tutorials. (2022). How do RGB LEDs work?. [Website] Available at: https://randomnerdtutorials.com/electronics-basics-how-do-rgb-leds-work/

DroneBot Workshop. (2022). Using Servo Motors with ESP32. [Website] Available at: https://dronebotworkshop.com/esp32-servo/

Elprocus. (2022). What is a Slide Switch: Working and Its Application. [Website] Available at: https://www.elprocus.com/slide-switch/

Practice 7.1

1. Read the following Arduino reference page on generating pseudo-random numbers and C reference page on escape sequence.
   a. `random()`: https://www.arduino.cc/reference/en/language/functions/random-numbers/random/ .
   b. Escape Sequence: https://www.ibm.com/docs/en/rdfi/9.6.0?topic=set-escape-sequences .

2. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
const int r = A0;                    – change A0 to 4 for WOKWI
const int g = A1;                    – change A1 to 5 for WOKWI
const int b = A5;                    – change A5 to 15 for WOKWI

int r_shade = 256;
int g_shade = 256;
int b_shade = 256;

void led_off()
{
 // TO DO: specify the RGB values to turn off the LED

}

void led_random()
{
 // TO DO: randomly select the RGB values

}

void led_change()
{
 // TO DO: send the RGB values to their corresponding pins

}

void setup() {
 // put your setup code here, to run once:
 pinMode (r, OUTPUT);
 pinMode (g, OUTPUT);
 pinMode (b, OUTPUT);

 // TO DO: call the relevant functions to pick an "off" RGB colour and
 //       send the output to their corresponding pins
```

```
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:

  Serial.print("R: ");
  Serial.print(r_shade);
  Serial.print("\tG: ");
  Serial.print(g_shade);
  Serial.print("\tB: ");
  Serial.println(b_shade);

  // TO DO: call the relevant functions to pick a random RGB colour and
  //       send the output to their corresponding pins


  delay(100);
}
```

Note: Use practice7_1.json for WOKWI.

3. If you have a Serial Monitor open already, please close it.

4. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_07_rgb. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

5. **Upload** the code.

TO DO

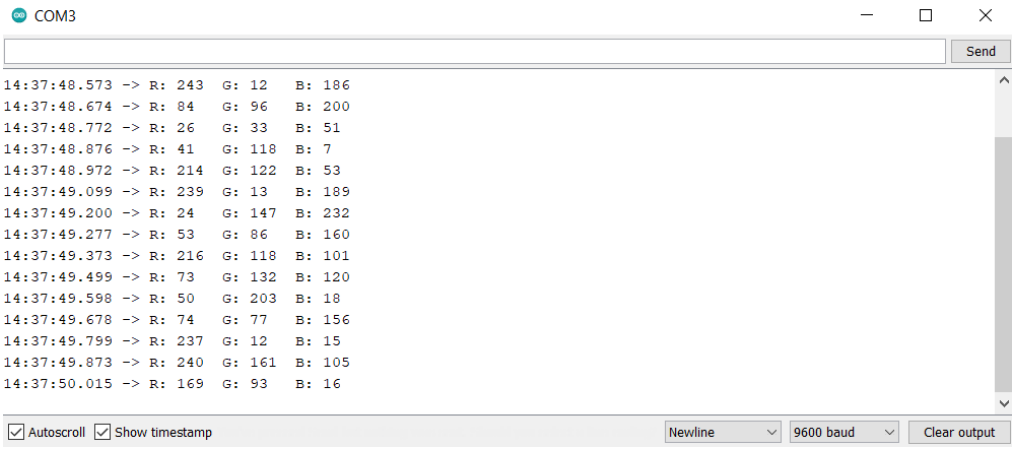
1. Complete the TO DO in tutorial_07_rgb.ino to select random RGB intensities and change these intensities every 100 milliseconds.

2. If you have a Serial Monitor open already, please close it.

3. Connect the RGB LED according to the values provided in tutorial_07_rgb.ino. The second pin should be connected to resistor to 3V.

4. **Verify** your modification and **Upload** the code.

5. Go to **Tools** and then select **Serial Monitor**. You should now see the output of similar to what is shown below as well as the RGB LED blinking in different colours.


COM3
Send
14:37:48.573 -> R: 243 G: 12 B: 186
14:37:48.674 -> R: 84 G: 96 B: 200
14:37:48.772 -> R: 26 G: 33 B: 51
14:37:48.876 -> R: 41 G: 118 B: 7
14:37:48.972 -> R: 214 G: 122 B: 53
14:37:49.099 -> R: 239 G: 13 B: 189
14:37:49.200 -> R: 24 G: 147 B: 232
14:37:49.277 -> R: 53 G: 86 B: 160
14:37:49.373 -> R: 216 G: 118 B: 101
14:37:49.499 -> R: 73 G: 132 B: 120
14:37:49.598 -> R: 50 G: 203 B: 18
14:37:49.678 -> R: 74 G: 77 B: 156
14:37:49.799 -> R: 237 G: 12 B: 15
14:37:49.873 -> R: 240 G: 161 B: 105
14:37:50.015 -> R: 169 G: 93 B: 16
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

Practice 7.2

1. Read the following Arduino reference page on `attachInterrupt()` and `millis()` which are used in this tutorial to achieve hardware interrupts:
   a. `attachInterrupt()`: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/ .
   b. `millis()`: https://www.arduino.cc/reference/en/language/functions/time/millis/ .

2. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
const int r = A0;                    – change A0 to 4 for WOKWI
const int g = A1;                    – change A1 to 5 for WOKWI
const int b = A5;                    – change A5 to 15 for WOKWI

int r_shade = 256;
int g_shade = 256;
int b_shade = 256;

#define timeSeconds 10
const int motionSensor = 27;

unsigned long now = millis();
unsigned long lastTrigger = 0;
boolean startTimer = false;

void led_off()
{
 // TO DO: specify the RGB values to turn off the LED

}

void led_random()
{
 // TO DO: randomly select the RGB values

}

void led_change()
{
 // TO DO: send the RGB values to their corresponding pins

}

// Checks if motion was detected, and starts a timer
```

```
void IRAM_ATTR detectsMovement()
{
 startTimer = true;
 lastTrigger = millis();
}

void setup() {
 // put your setup code here, to run once:
 pinMode (r, OUTPUT);
 pinMode (g, OUTPUT);
 pinMode (b, OUTPUT);

 // TO DO: call the relevant functions to pick an "off" RGB colour and
 //       send the output to their corresponding pins


 // PIR Motion Sensor mode INPUT_PULLUP
 pinMode(motionSensor, INPUT_PULLUP);
 attachInterrupt(motionSensor, detectsMovement, HIGH);

 Serial.begin(9600);
}

void loop() {
 // put your main code here, to run repeatedly:

 // Current time
 now = millis();
 // Turn off the LED after the number of seconds defined in the timeSeconds variable
 if(startTimer && (now - lastTrigger > (timeSeconds*1000)))
 {
  Serial.println("Motion stopped...");
  startTimer = false;

  // TO DO: call relevant functions to turn off the LED

 }
 // TO DO: turn on the LED with random RGB values for the number of seconds defined
 //       in the timeSeconds variable

 Serial.print("R: ");
 Serial.print(r_shade);
 Serial.print("\tG: ");
 Serial.print(g_shade);
 Serial.print("\tB: ");
 Serial.println(b_shade);

 delay(100);
```

}

Note: Use practice7_2.json for WOKWI.

3. If you have a Serial Monitor open already, please close it.

4. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_07_pir_interrupt. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

5. **Upload** the code.

TO DO

1. Reuse `led_off()`, `led_random()` and `led_change()` from tutorial_07_rgb.ino to to select random RGB intensities and change these intensities every 100 milliseconds.

2. Complete the TO DO in tutorial_07_pir_interupt.ino to turn on the RGB LED using a randomly selected colours, and to turn off the RGB LED 10 seconds after the PIR motion sensor detects any movements.

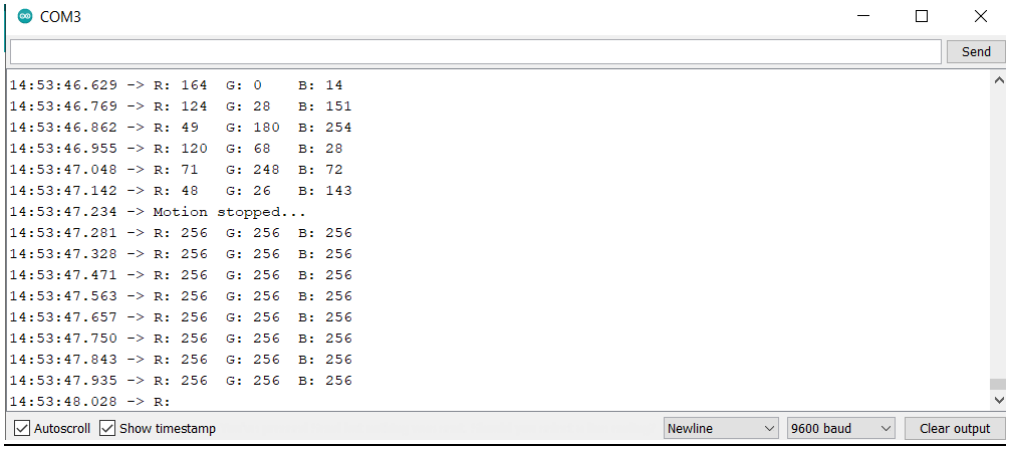
3. Connect the RGB LED and the PIR motion sensor, according to the value specified in tutorial_07_pir interrupt.ino.

   Note: Please ensure that the black wire is plugged on the Ground side (the red wire is therefore on the 5V side), this will ensure that the connector aligns correctly so that the PIR sensor and the female connector will not be damaged.

4. If you have a Serial Monitor open already, please close it.

5. **Verify** your modification and **Upload** the code.

6. Go to **Tools** and then select **Serial Monitor**. You should now see the output of similar to what is shown below.


COM3
Send
14:53:46.629 -> R: 164 G: 0 B: 14
14:53:46.769 -> R: 124 G: 28 B: 151
14:53:46.862 -> R: 49 G: 180 B: 254
14:53:46.955 -> R: 120 G: 68 B: 28
14:53:47.048 -> R: 71 G: 248 B: 72
14:53:47.142 -> R: 48 G: 26 B: 143
14:53:47.234 -> Motion stopped...
14:53:47.281 -> R: 256 G: 256 B: 256
14:53:47.328 -> R: 256 G: 256 B: 256
14:53:47.471 -> R: 256 G: 256 B: 256
14:53:47.563 -> R: 256 G: 256 B: 256
14:53:47.657 -> R: 256 G: 256 B: 256
14:53:47.750 -> R: 256 G: 256 B: 256
14:53:47.843 -> R: 256 G: 256 B: 256
14:53:47.935 -> R: 256 G: 256 B: 256
14:53:48.028 -> R:
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

Practice 7.3

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
#include <ESP32Servo.h>

#define servo_pin 27

Servo servo;
int pos = 0;

void setup() {
  // put your setup code here, to run once:
  servo.attach(servo_pin);
}

void loop() {
  // put your main code here, to run repeatedly:

  // TO DO: move the servo blade between 0 - 180 degrees
  //       creating a sweep from left to right
  //       e.g., servo.write(x)
  //       Note: add a small delay for the servo to move the blade into position

  // TO DO: move the servo blade between 180 - 0 degrees
  //       creating a sweep back from right to left
  //       e.g., servo.write(x)
  //       Note: add a small delay for the servo to move the blade into position

}
```

Note: Use practice7_3.json for WOKWI. You will also need to add ESP32Servo library on WOKWI using the Library Manager tab.

2. If you have a Serial Monitor open already, please close it.

3. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_07_servo. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

4. **Upload** the code.

TO DO

1. Complete all TO DOs and modify tutorial_07_servo.ino to turn the Servo motor's blade smoothly from left to right and vice versa.

   Hint: Use `for` loop to adjust the blade's angles.

2. Revisit the following Arduino reference pages on compound operators which may be useful for adjusting the blade's angles if you have forgotten how to use them.

   a. `-=`: https://www.arduino.cc/reference/en/language/structure/compound-operators/compoundsubtraction/ .
   b. `--`: https://www.arduino.cc/reference/en/language/structure/compound-operators/decrement/ .
   c. `+=`: https://www.arduino.cc/reference/en/language/structure/compound-operators/compoundaddition/ .
   d. `++`: https://www.arduino.cc/reference/en/language/structure/compound-operators/increment/ .

3. Connect jumper wires to the Servo motor according to the values specified in tutorial_07_servo.ino.

4. If you have a Serial Monitor open already, please close it. **Verify** your modification.

5. **Upload** the code to see if it still works correctly.

Practice 7.4

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial as well as the link to Espressif's references page below to make sure that you thoroughly understand the meaning of every line of the code below.

   Hint: Espressif's Timer - https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/api/timer.html .

```
#define led 21
hw_timer_t *my_timer = NULL;

void IRAM_ATTR onTimer(){
 // TO DO: turn the LED on or off
}

void setup() {
 // put your setup code here, to run once:
 pinMode(led, OUTPUT);
 my_timer = timerBegin(0, 80, true);
 timerAttachInterrupt(my_timer, &onTimer, true);
 timerAlarmWrite(my_timer, 1000000, true);
 timerAlarmEnable(my_timer);

 // TO DO: attach servo to a pin
}

void loop() {
 // put your main code here, to run repeatedly:

 // TO DO: reuse the sketch to sweep the servo's blade
}
```

   Note: Use practice7_4.json for WOKWI. You will also need to add ESP32Servo library on WOKWI using the Library Manager tab.

2. If you have a Serial Monitor open already, please close it.

3. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_07_timer_interrupt. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

4. **Upload** the code.

TO DO

1. Complete all TO DOs and modify tutorial_07_timer_interrupt.ino to turn the LED on/off every 10 seconds, and to turn the Servo motor's blade smoothly from left to right and vice versa (you can reuse tutorial_07_servo.ino for this part).

2. **Verify** your modification and **Upload** the code to see if it works correctly.

Challenge 7.1

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial as well as the link to Espressif's reference page (and the esp_sleep.h) to make sure that you thoroughly understand the meaning of every line of the code below.

   Hint:
   Espressif's Sleep Modes - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html ,
   Espressif's esp_sleep.h - https://github.com/espressif/esp-idf/blob/7869f4e151/components/esp_hw_support/include/esp_sleep.h .

```
// TO DO: calculate pin bitmask for pin 14 and pin 15 (RTC 16 and RCT 13 respectively)
#define BUTTON_PIN_BITMASK 0x

RTC_DATA_ATTR int bootCount = 0;

int slide_pin = 21;

void wakeup_cause()
{
 esp_sleep_wakeup_cause_t wakeup_reason;

 wakeup_reason = esp_sleep_get_wakeup_cause();

 // TO DO: reuse the switch statement from tutorial_06_timer_wakeup.ino

}

void GPIO_wake_up(){
 int GPIO_reason = esp_sleep_get_ext1_wakeup_status();

 Serial.print(GPIO_reason);
 Serial.print(" ");
 Serial.print("GPIO that triggered the wake up: GPIO ");
 Serial.println((log(GPIO_reason))/log(2), 0);

 // TO DO: output to the slide switch

}

void setup()
{
 // put your setup code here, to run once:

 Serial.begin(115200);
```

```
  delay(1000); //Take some time to open up the Serial Monitor

  pinMode(slide_pin, OUTPUT);

  ++bootCount; //Increment boot number
  Serial.print("Boot number: ");
  Serial.println(bootCount);

  wakeup_cause();
  GPIO_wake_up();

  // TO DO: Enable ext1 for RTC 13 and RTC 16 pin (using pin bit mask) as the wakeup source


  Serial.println("Going to sleep now");
  Serial.flush();
  esp_deep_sleep_start();
}

void loop()
{
  //This will never be reached
}
```

Note: Use Challenge7_1.json. Please note that Sleep Mode doesn't get simulated correctly in WOKWI. So, you will not be able to see the increase in Boot number, i.e., it will always be 1, and pressing the buttons won't actually wake up the ESP32.

2. Connect 2 tactile switch button, 2 LEDs (e.g., red and green), 4 resistors, 1 SPDT switch based on the information given in the sketch above. One leg of each button should be connected to 3V and the other leg should be connected to the relevant data pin, to resistor, and to Ground. The purpose of the SPDT switch is to allow the user to change between red or green LED.

3. If you have a Serial Monitor open already, please close it.

4. **Verify** the code. You will then be asked to save the sketch. Enter a meaningful name for your file, e.g., tutorial_07_ext1_challenge. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

5. **Upload** the code.

TO DO

1. Complete all of the TO DO tasks as shown in the comments in tutorial_07_ext1_challenge.ino.

2. If you have a Serial Monitor open already, please close it. **Verify** your modification.

3. **Upload** the code.

4. Go to **Tools** and then select **Serial Monitor**. You should now see the output indicating when a different button is pressed corresponding the RTC pin it is connected to.