Tutorial 06 – Deep Sleep and Wakeup Sources

Make sure you complete **Setup** and
Test Your Adafruit Feather Huzzah32 Connection
document before starting this Tutorial 06.

Learning Objectives

1. Understand the use of the Pyroelectric (passive) infrared (PIR) sensor.
2. Understand the difference between serial and parallel communication.
3. Understand the basics of programming, i.e., different control structures (i.e., `switch`).
4. Implement function calls related to deep sleep, i.e., `esp_sleep_get_wakeup_cause()`, `esp_deep_sleep_start()`, on Adafruit Feather Huzzah32.
5. Implement function calls related to using timer as a wake up source during deep sleep, i.e., `esp_sleep_enable_timer_wakeup()`, on Adafruit Feather Huzzah32.
6. Implement function calls related to using a touch pin as a wake up source during deep sleep, i.e., `esp_sleep_get_touchpad_wakeup_status()`, `touchAttachInterrupt()`, `esp_sleep_enable_touchpad_wakeup()`, on Adafruit Feather Huzzah32.
7. Implement function calls related to using external peripherals as wake up sources during deep sleep, i.e., `esp_sleep_enable_ext0_wakeup()`, on Adafruit Feather Huzzah32.
8. Implement function calls related to serial communications, i.e., `Serial.flush()`, `Serial.availableForWrite()`, `if(Serial)`, on Adafruit Feather Huzzah32.

Theory

Communication protocols in embedded electronics can be grouped into 2 categories: parallel and serial. Parallel interfaces transfer multiple bits at a time and usually require buses of data (transmitting across 8 or more wires). Serial interfaces stream 1 single bit at a time on as little as one wire. Figure 1 and Figure 2 illustrate the differences between serial and parallel communication respectively.

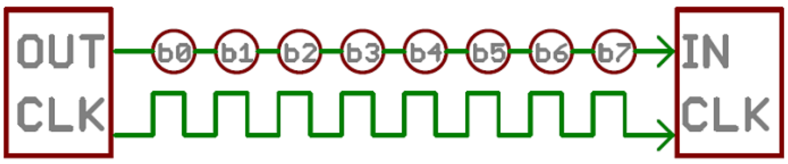


Figure 1: An example of a serial interface (Sparkfun, n.d.).

Parallel communication, like a mega-highway, is faster, but requires more input and output lines (e.g., copper). Serial communication, like a rural country road, serves its purpose but cost less. Universal Serial Bus (USB) is one of the more well-known computing serial interfaces.

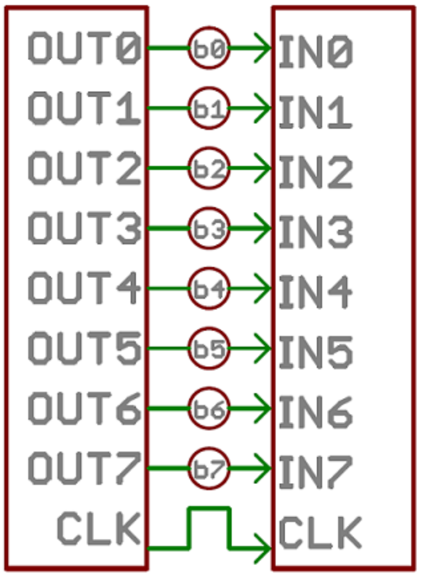


Figure 2: An example of a parallel interface (Sparkfun, n.d.).

Serial interfaces can be categorised into 2 groups, i.e., synchronous, and asynchronous. A synchronous serial interface pairs its data line(s) with a clock signal, therefore an extra line between communicating devices. An asynchronous serial interface transfer data without support from an internal clock signal.

To ensure robust data transfer in asynchronous serial communication, 4 mechanisms are used instead of the clock signal, i.e., data bits, synchronisation bits, parity bits and baud rate. Communicating devices on a serial bus must be configured to use exactly the same configurations. Baud rate (bits per seconds or bps) specifies how fast data is sent over a serial line. Common baud rate where speed isn't critical is 9600 bps. For most microcontrollers, 115200 bps is fast, and anything beyond that will begin to cause errors at the receiving end. For ESP32, the default baud rate is 115200 bps.

Each block of data (usually a byte) is sent in a frame as shown in Figure 3. The start and stop bit mark the beginning and end of a packet. There is always only 1 start bit, but the number of stop bits is configurable to either 1 or 2 (but 1 is more common). The parity bit performs a simple error checking and is optional. Depending on the chosen protocol, i.e., odd or even, the parity bit is set to 1 or 0, to make to total number of 1s of the data plus parity to odd or even. For example, a data byte like 01011101 has 5 1s, the parity bit would be set to 1 assuming parity is set to even.

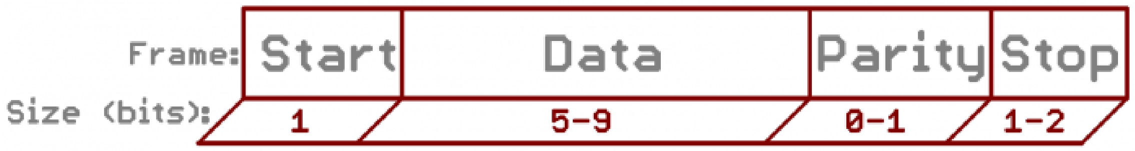


Figure 3: An example of a serial frame (Sparkfun, n.d.).

Figure 4 illustrates an example of sending "OK" on 9600 8N1, i.e., 9600 baud rate, 8 data bits, no parity, and 1 stop bit, which is one of the more commonly used serial protocols.

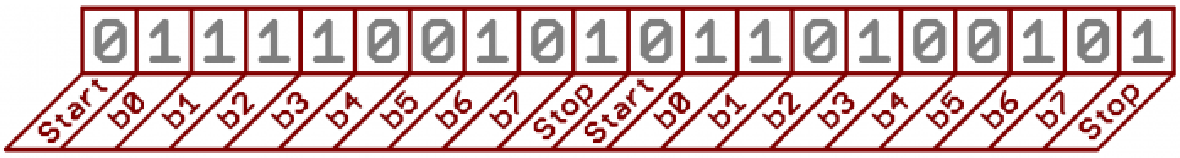

0 1 1 1 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1
Start b0 b1 b2 b3 b4 b5 b6 b7 Stop Start b0 b1 b2 b3 b4 b5 b6 b7 Stop

Figure 4: An example serial transmission (Sparkfun, n.d.).

Two packets of data are needed, one for each character. The ASCII value of 'O' is 79, i.e., 01001111. The ASCII value of 'K' is 75, i.e., 01001011. Based on the figure, the data is assumed to have been transferred least-significant bit first (i.e., little endian), as the bits for each character are read right to left. This little endianness is the dominant ordering for processor architectures, whereas big endianness (i.e., most significant bit first) is the dominant ordering for networking protocols. In this example, because there 10 bits per each byte of data, the transmission rate is 960 bytes (characters) per second, i.e., 9600/10.

Commands such as `Serial.print()` and `Serial.println()` do not directly write their output to Serial Monitor. These serial communication functions are interrupt based. They store their output in a transmit buffer and will normally return before any characters are transmitted over serial line. This process limits the chance of serial communications blocking other function calls to proceed within the program, and therefore making your program more responsive. However, it should be noted that these function calls will block the program until there is enough space in the buffer. `Serial.availableForWrite()` can be used to find out the number of bytes (characters) available for writing in serial buffer, e.g., 128 bytes for Adafruit Feather Huzzah32.

If blocking is required, i.e., to wait until the **outgoing** transmission is completed, `Serial.flush()` can be used. This can be useful in avoiding output being left in the transmit buffer (e.g., incomplete transmissions) before putting Adafruit Feather Huzzah32 to sleep. To empty the **incoming** buffer, the program will have to read all of the buffer's content using statements such as this:

```
while(Serial.available())
{
  Serial.read();
}
```

or this:

```
while(Serial.read() >= 0);
```

Please revisit Tutorial 02 if you do not understand how the two snippets of code above work.

One common issue with Serial Monitor is sending a message, but nothing is displayed (especially at the start of the program for the first time). This can happen when the Serial Monitor is not quite ready or active yet. To ensure that the USB connect is ready before a message is sent from the program, this code snippet:

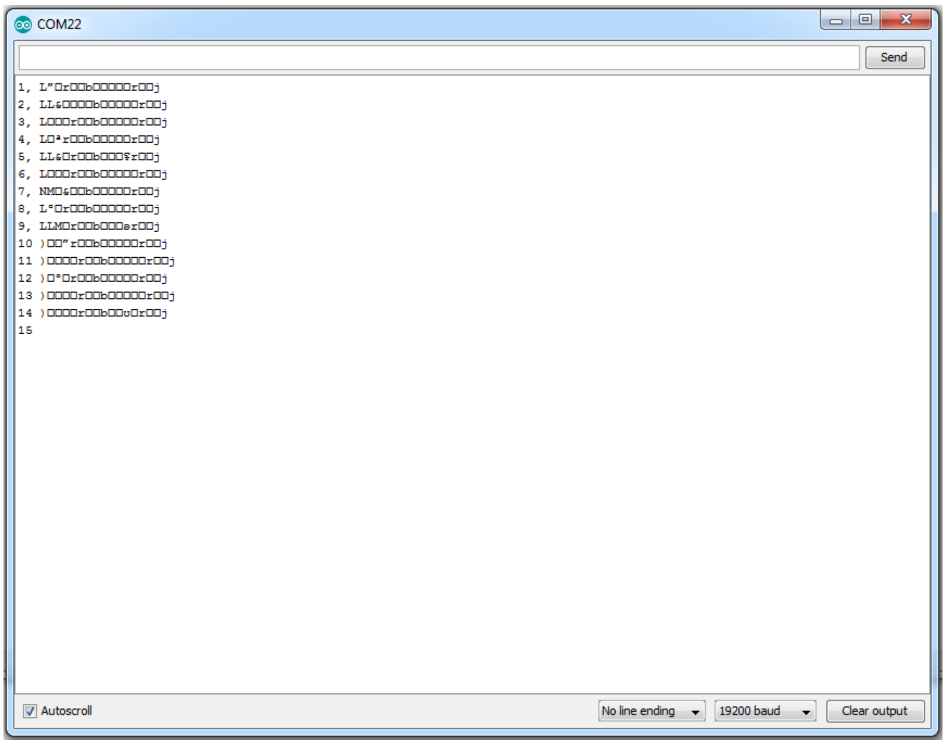
```
While(!Serial);
```

or this:

```
delay();
```

can be added to the program after you have specified a baud rate in your program, i.e., after `Serial.begin()`. The latter is less appropriate as it may be impossible for you to guess how long to wait for.

It is important to note the two common causes of garbage on Serial Monitor, as shown in Figure 5. These are the mismatched baud rate between 2 communicating devices, and the unemptied transmit buffer. Therefore, whenever you see garbage on your Serial Monitor, makes sure to call appropriate functions to address these two common causes.



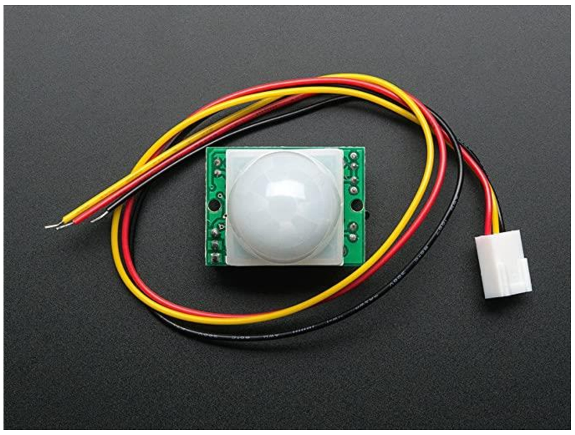
```
1, L"□r□□b□□□□□□r□□j
2, LL&□□□□□b□□□□□□r□□j
3, L□□□□r□□b□□□□□□r□□j
4, L□ªr□□b□□□□□□r□□j
5, LL&□r□□b□□□♀r□□j
6, L□□□□r□□b□□□□□□r□□j
7, NM□&□□b□□□□□□r□□j
8, L°□r□□b□□□□□□r□□j
9, LLM□r□□b□□□ɘr□□j
10 )□□"r□□b□□□□□□r□□j
11 )□□□□□r□□b□□□□□□r□□j
12 )□°□r□□b□□□□□□r□□j
13 )□□□□□r□□b□□□□□□r□□j
14 )□□□□□r□□b□□ʋ□r□□j
15
```

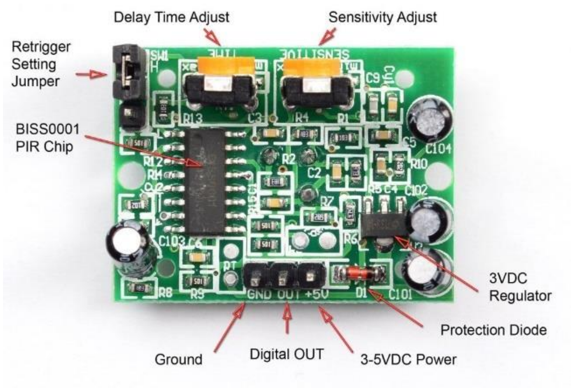
Figure 5: Serial monitor's garbage output (Arduino.cc.,2020).

Read the following Arduino reference pages to better understand serial communications.

1. `write()`: (see the Notes and Warnings in particular) https://www.arduino.cc/reference/en/language/functions/communication/serial/write/ .
2. `availableForWrite()`: https://www.arduino.cc/reference/en/language/functions/communication/serial/availableforwrite/ .
3. `flush()`: https://www.arduino.cc/reference/en/language/functions/communication/serial/flush/ .
4. `if(Serial)`: https://www.arduino.cc/reference/en/language/functions/communication/serial/ifserial/ .

Pyroelectricity is the ability of certain materials to generate a temporary voltage when they are heated or cooled. Everything emits some level of radiation. The hotter something (or someone) is, the more infrared (IR) radiation they emit. Pyroelectric sensors or passive infrared sensors or PIR sensors are inexpensive, low-power motion sensors. Figure 8 illustrates an example of PIR sensors and their pins.


(a)



(b)

Figure 8: (a) PIR sensor (Amazon, 2022), and (b) their pins (Adafruit, n.d. b).

A PIR sensor is made up of two halves to detect the change in IR radiation, i.e., when one half detects more (or less) IR radiation than the other, the output will be high (or low). Most PIR sensor will have a 3-pin connection. One pin will be Ground, another will be data and the final pin will be power. For 5COM2004, it is more reliable to **connect to the USB pin** (instead of 3V like most other sensors which have been used in 5COM2004) on Adafruit Feather Huzzah32 as this will supply 5V DC for the sensor. Please ensure that **the black wire is plugged on the Ground side**, this will ensure that the connector aligns correctly so that the PIR sensor and the female connector will not be damaged. Please do not adjust the delay time or sensitivity of the PIR sensor as small adjustments make a huge impact, e.g., if the delay time become too long, you may thought your sketch is not working correctly.

Reference

Sparkfun. (n.d.). Serial Communication. [Website] Available at: https://learn.sparkfun.com/tutorials/serial-communication/all

Arduino.cc. (2020). Problem with Serial Monitor and Sleep functions. [Website] Available at: https://forum.arduino.cc/t/problems-with-serial-monitor-and-sleep-functions/514264/2

Amazon. (2022). Adafruit PIR (motion) Sensor [ADA189]. [Website] Available at: https://www.amazon.co.uk/Adafruit-PIR-motion-Sensor-ADA189/dp/B00JOZTAC6

Adafruit. (n.d. b). PIR Motion Sensor. [Website] Available at: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor

Practice 6.1

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 references page (and the esp_sleep.h) below 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 .

```
#define sleep_microsecond  5000000 /* 5 seconds */

RTC_DATA_ATTR int bootCount = 0;

void wakeup_cause()
{
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  // TO DO: write a switch statement for ext0, ext1, timer, touchpad and default
}

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

  Serial.begin(115200);
  delay(1000); //Take some time to open up the Serial Monitor

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

  wakeup_cause();

  esp_sleep_enable_timer_wakeup(sleep_microsecond);

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

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

}
```

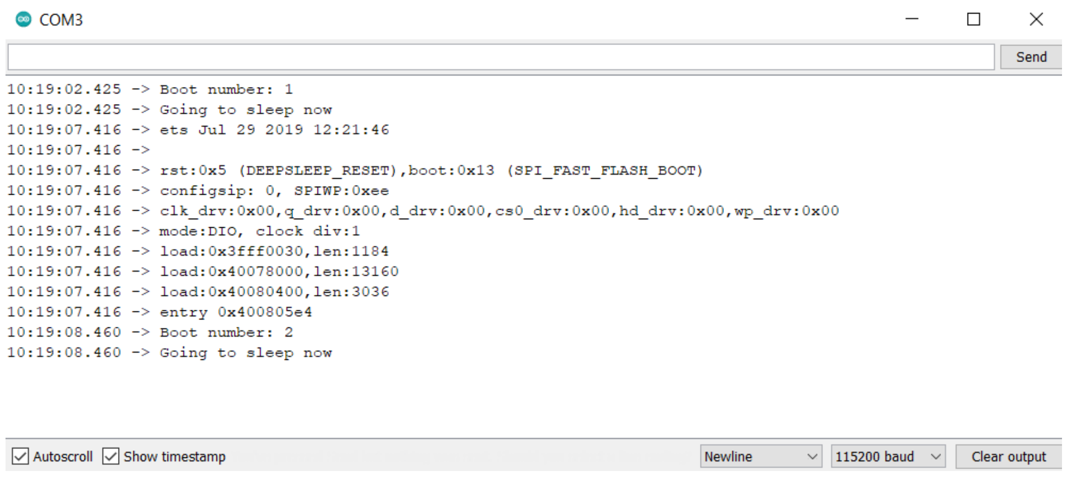
Note: Use default.json for WOKWI.

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_06_timer_wakeup. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

4. **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. Please note that the majority of the message shown here is automatically generated, except for the first two lines and last two lines.



```
10:19:02.425 -> Boot number: 1
10:19:02.425 -> Going to sleep now
10:19:07.416 -> ets Jul 29 2019 12:21:46
10:19:07.416 ->
10:19:07.416 -> rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
10:19:07.416 -> configsip: 0, SPIWP:0xee
10:19:07.416 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
10:19:07.416 -> mode:DIO, clock div:1
10:19:07.416 -> load:0x3fff0030,len:1184
10:19:07.416 -> load:0x40078000,len:13160
10:19:07.416 -> load:0x40080400,len:3036
10:19:07.416 -> entry 0x400805e4
10:19:08.460 -> Boot number: 2
10:19:08.460 -> Going to sleep now
```

Note: Please note that Sleep Mode doesn't get simulated correctly in WOKWI and you will not be able to see the increase in Boot number, i.e., it will always be 1.

TO DO

1. Read the following Arduino reference page on `switch … case` https://www.arduino.cc/reference/en/language/structure/control-structure/switchcase/ .

2. Complete the TO DO in tutorial_06_timer_wakeup.ino to print informative messages for different `esp_sleep_wakeup_cause_t`, i.e.,
"Wakeup caused by external signal using RTC_IO" for `ESP_SLEEP_WAKEUP_EXT0` case,
"Wakeup caused by external signal using RTC_CNTL" for `ESP_SLEEP_WAKEUP_EXT1` case,

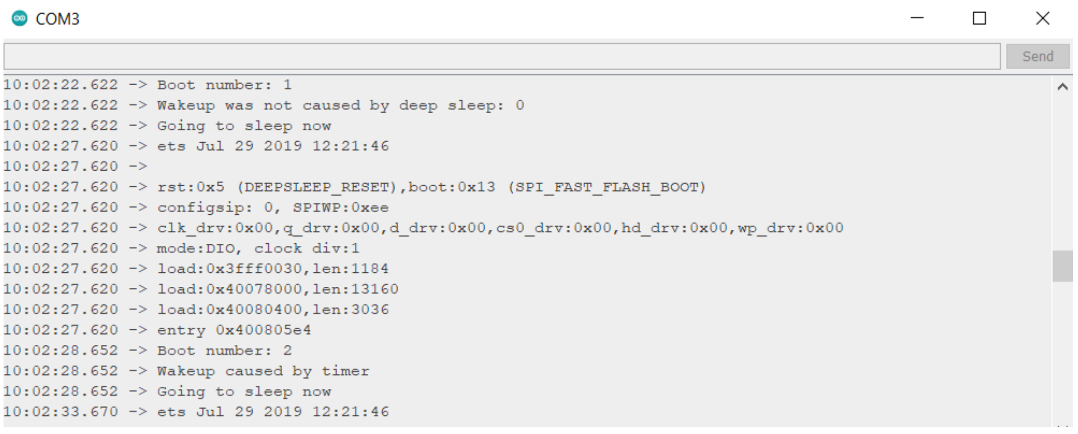
"Wakeup caused by timer" for `ESP_SLEEP_WAKEUP_TIMER` case,
"Wakeup caused by touchpad" for `ESP_SLEEP_WAKEUP_TOUCHPAD` case, and
"Wakeup was not caused by deep sleep: %d\n" for `default` case where %d displays the integer value of `esp_sleep_wakeup_cause_t`.

Note: Please revisit Tutorial 05, if you do not remember how to use format specifier.

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

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.



```
10:02:22.622 -> Boot number: 1
10:02:22.622 -> Wakeup was not caused by deep sleep: 0
10:02:22.622 -> Going to sleep now
10:02:27.620 -> ets Jul 29 2019 12:21:46
10:02:27.620 ->
10:02:27.620 -> rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
10:02:27.620 -> configsip: 0, SPIWP:0xee
10:02:27.620 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
10:02:27.620 -> mode:DIO, clock div:1
10:02:27.620 -> load:0x3fff0030,len:1184
10:02:27.620 -> load:0x40078000,len:13160
10:02:27.620 -> load:0x40080400,len:3036
10:02:27.620 -> entry 0x400805e4
10:02:28.652 -> Boot number: 2
10:02:28.652 -> Wakeup caused by timer
10:02:28.652 -> Going to sleep now
10:02:33.670 -> ets Jul 29 2019 12:21:46
```

Practice 6.2

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 .

```
#define threshold 40

RTC_DATA_ATTR int bootCount = 0;
RTC_DATA_ATTR int touchCount = 0;

touch_pad_t touchPin;

int touched = 0;

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 wakeup_touch()
{
  touchPin = esp_sleep_get_touchpad_wakeup_status();

  // TO DO: write the switch statement for all touch pins
}

void callback()
{
  touched = 1;
}

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

```
  Serial.begin(115200);
  delay(1000); //Take some time to open up the Serial Monitor

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

  wakeup_cause();
  wakeup_touch();

  //Setup interrupt on Touch Pad 3 (GPIO15)
  touchAttachInterrupt(T3, callback, threshold);
  delay(1000); // due to the fact that the board has to be awake for call back to be executed

  if(touched == 1)
  {
    ++touchCount;
    Serial.print("Touch number: ");
    Serial.println(touchCount);
    touched = 0;
  }

  //Configure Touchpad as wakeup source
  esp_sleep_enable_touchpad_wakeup();

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

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

Note: Use default.json for WOKWI.

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_06_touch_wakeup. 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. Check `enum touch_pad_t` from Espressif's reference page, which indicates the values for 10 possible touch pins on ESP32 boards. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/touch_pad.html#_CPPv411touch_pad_t .

2. Reuse `wakeup_cause()` from tutorial_06_timer_wakeup.ino to print informative messages for different `esp_sleep_wakeup_cause_t` cases.

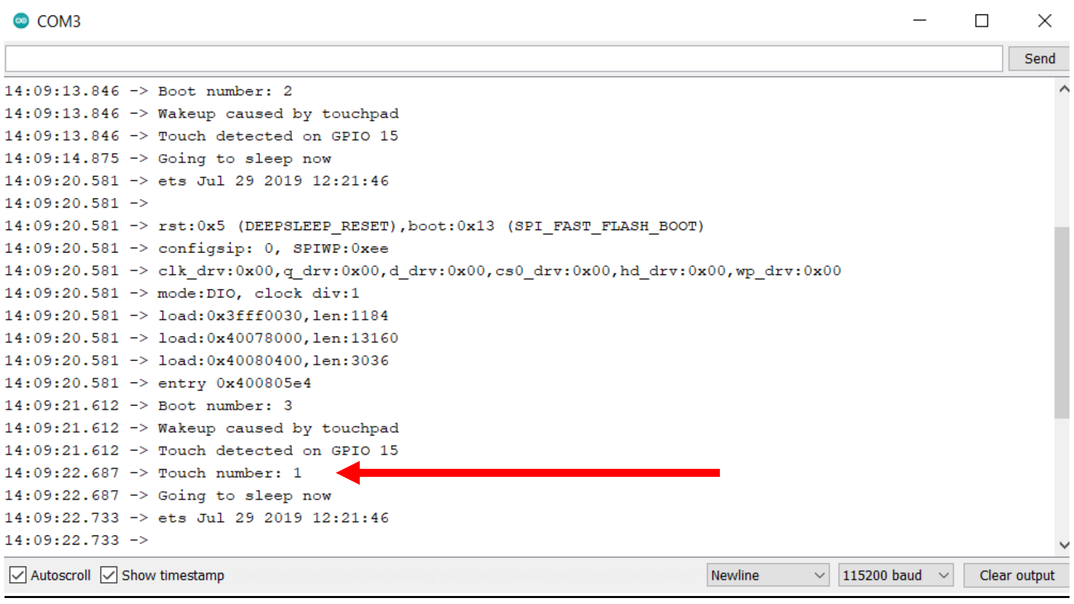
3. Complete the TO DO in tutorial_06_touch_wakeup.ino to print informative messages for when different touch pins on ESP32 are being touched, e.g., "Touch detected on GPIO X" where X represents the GPIO number of that touch pin. You should also include the default case to provide this message "Wakeup not by touchpad" as well.

4. Plug in a jumper wire to the appropriate GPIO, according to the value specified in tutorial_06_touch_wakeup.ino.

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

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

7. Go to **Tools** and then select **Serial Monitor**. You should now see the output of similar to what is shown below. Please note that the majority of the message shown here is automatically generated, except for those in `Serial.print()` or `Serial.println()` in tutorial_06_touch_wakeup.ino.



```
14:09:13.846 -> Boot number: 2
14:09:13.846 -> Wakeup caused by touchpad
14:09:13.846 -> Touch detected on GPIO 15
14:09:14.875 -> Going to sleep now
14:09:20.581 -> ets Jul 29 2019 12:21:46
14:09:20.581 ->
14:09:20.581 -> rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
14:09:20.581 -> configsip: 0, SPIWP:0xee
14:09:20.581 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
14:09:20.581 -> mode:DIO, clock div:1
14:09:20.581 -> load:0x3fff0030,len:1184
14:09:20.581 -> load:0x40078000,len:13160
14:09:20.581 -> load:0x40080400,len:3036
14:09:20.581 -> entry 0x400805e4
14:09:21.612 -> Boot number: 3
14:09:21.612 -> Wakeup caused by touchpad
14:09:21.612 -> Touch detected on GPIO 15
14:09:22.687 -> Touch number: 1
14:09:22.687 -> Going to sleep now
14:09:22.733 -> ets Jul 29 2019 12:21:46
14:09:22.733 ->
```

Note: Please note that Sleep Mode doesn't get simulated correctly in WOKWI and you will not be able to see the increase in Boot number, i.e., it will always be 1. There is no simple touch input on WOKWI, so you won't be able to test that either.

8. Note the fact that there is a message about the Touch number in Boot number 3 (as pointed out using the red arrow), but not in Boot number 2. This example was generated by touching and holding the jumper wire for Boot number 3, but touching and releasing instantly for Boot number 2. Please revisit Lecture 06, if you do not understand why the output appears as shown above.

Practice 6.3

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.

   ```
   #define led 21
   #define pir 26

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

     Serial.begin(9600);

     pinMode(pir, INPUT);
     pinMode(led, OUTPUT);
   }

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

     // TO DO: read motion sensor input

     // TO DO: display the input from motion sensor

     // TO DO: turn on the LED if there is any motion,
     //       turn off the LED if there isn't any motion

     delay(1000);
   }
   ```

   Note: Use practice6_3.json for WOKWI.

2. Connect PIR motion sensor, LED diode and resistor based on the information given in the sketch above.

   Note: Please make sure to **unplug your USB cable before connecting your PIR motion sensor**. Please ensure that you connect the 3 wires correctly before you reconnect the USB cable as incorrectly connecting the PIR sensor's 3 wires could damage the sensor. Revisit Figure 8 of this practical sheet if you are unsure, **the black wire must be plugged on the Ground side.**

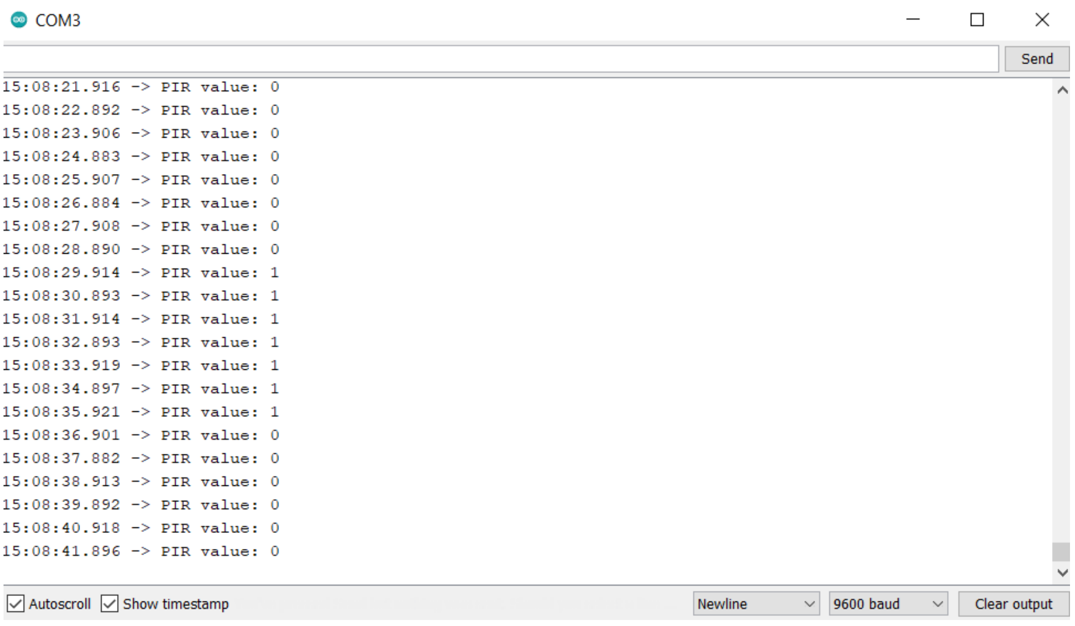
3. If you have a Serial Monitor open already, please close it.

4. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_06_pir. 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 TO DOs and modify tutorial_06_pir.ino to produce the output on the Serial Monitor similar to what is shown below, i.e., produce the message “PIR value: X”, where X represent the value read from the specified pin. The sketch should also light up the LED diode, and when the PIR motion sensor detects a motion, and turn off the LED diode otherwise.



```
15:08:21.916 -> PIR value: 0
15:08:22.892 -> PIR value: 0
15:08:23.906 -> PIR value: 0
15:08:24.883 -> PIR value: 0
15:08:25.907 -> PIR value: 0
15:08:26.884 -> PIR value: 0
15:08:27.908 -> PIR value: 0
15:08:28.890 -> PIR value: 0
15:08:29.914 -> PIR value: 1
15:08:30.893 -> PIR value: 1
15:08:31.914 -> PIR value: 1
15:08:32.893 -> PIR value: 1
15:08:33.919 -> PIR value: 1
15:08:34.897 -> PIR value: 1
15:08:35.921 -> PIR value: 1
15:08:36.901 -> PIR value: 0
15:08:37.882 -> PIR value: 0
15:08:38.913 -> PIR value: 0
15:08:39.892 -> PIR value: 0
15:08:40.918 -> PIR value: 0
15:08:41.896 -> PIR value: 0
```

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** to see if it still works correctly.

Practice 6.4

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 to make sure that you thoroughly understand the meaning of every line of the code below.

```
#define led 21
#define pir 26

unsigned long interval = 10000; // 10 seconds
unsigned long current = 0;
unsigned long previous = 0;
int ledState = 0;

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

  Serial.begin(9600);

  pinMode(pir, INPUT);
  pinMode(led, OUTPUT);
}

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

  // TO DO: read motion sensor input

  // TO DO: display the input from motion sensor

  // TO DO: turn on the LED if there is any motion,
  //       turn off the LED if there isn't any motion after 10 seconds.
  //       ONLY when the LED is on, display elasped time since the LED was on.

}
```

   Note: Use practice6_3.json for WOKWI.

2. Connect PIR motion sensor, LED diode and resistor based on the information given in the sketch above.

   Note: Please make sure to **unplug your USB cable before connecting your PIR motion sensor**. Please ensure that you connect the 3 wires correctly before you reconnect the USB cable as incorrectly connecting the PIR sensor's 3 wires could damage the sensor. Revisit Figure 8 of this practical sheet if you are unsure, **the black wire must be plugged on the Ground side.**

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

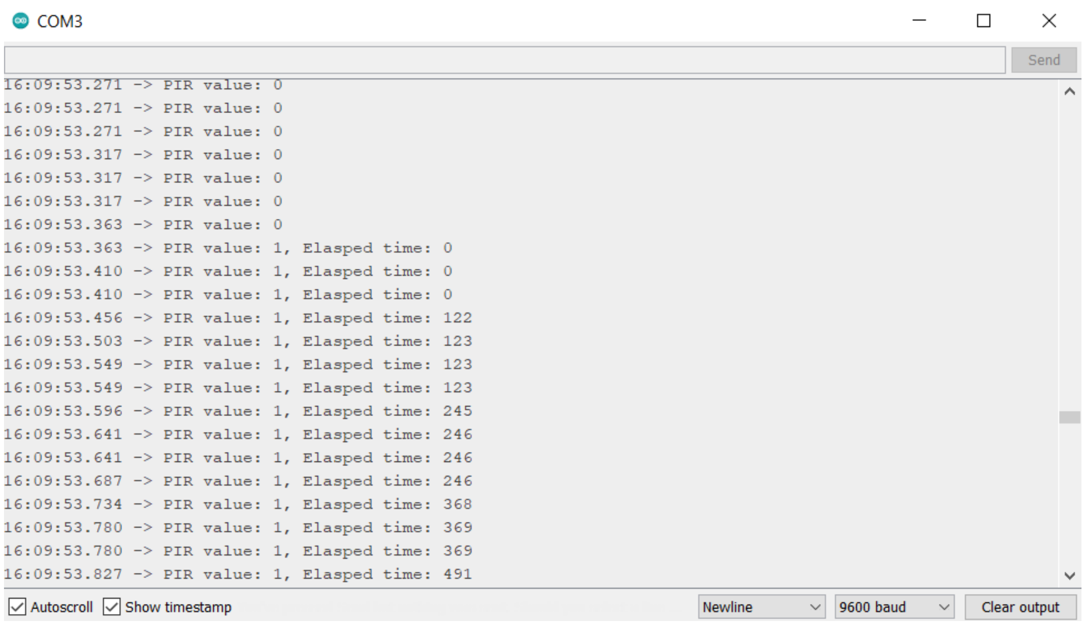
4. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_06_pir_millis. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

5. **Upload** the code.

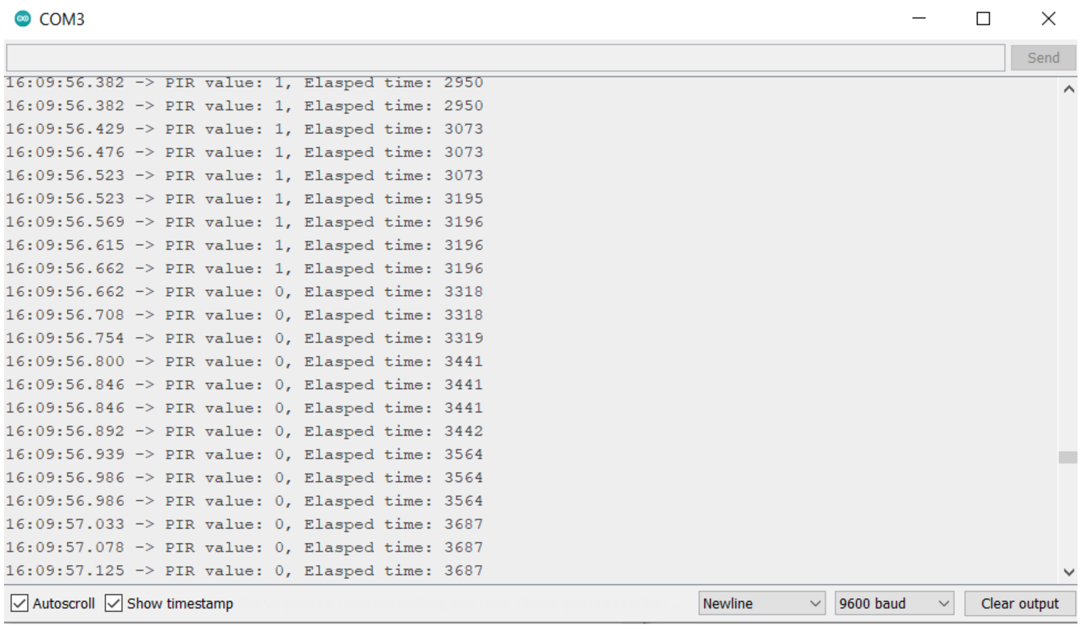
TO DO

1. Complete all TO DOs and modify tutorial_06_pir_millis.ino to produce the output on the Serial Monitor similar to what is shown below, i.e., produce the message "PIR value: X, Elapsed Time: YYYY", where X represent the value read from the specified pin and YYYY represents the number of milliseconds since the LED was turned on. It should be noted that this Elapsed Time: YYYY should **only** be produced when the LED is on. You will note from the screenshot below that the sketch continues to produce the Elapsed Time: YYYY even when there is no longer any motion. The sketch should also light up the LED diode, and when the PIR motion sensor detects a motion for the specified interval, i.e., 10 seconds.

   Hint: Use `millis()`.

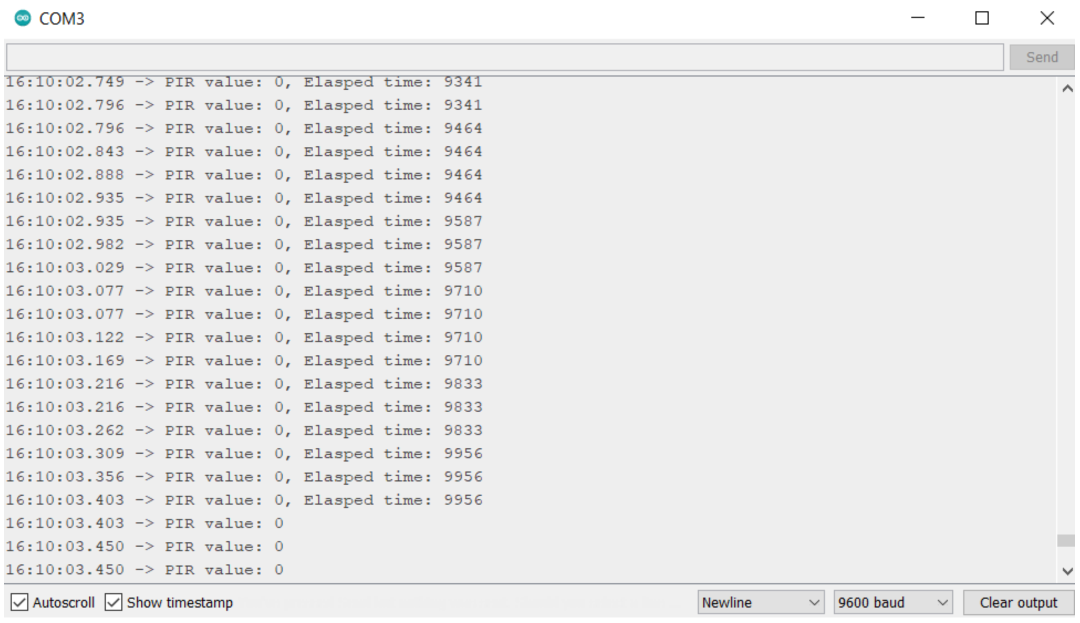


```
16:09:53.271 -> PIR value: 0
16:09:53.271 -> PIR value: 0
16:09:53.271 -> PIR value: 0
16:09:53.317 -> PIR value: 0
16:09:53.317 -> PIR value: 0
16:09:53.317 -> PIR value: 0
16:09:53.363 -> PIR value: 0
16:09:53.363 -> PIR value: 1, Elasped time: 0
16:09:53.410 -> PIR value: 1, Elasped time: 0
16:09:53.410 -> PIR value: 1, Elasped time: 0
16:09:53.456 -> PIR value: 1, Elasped time: 122
16:09:53.503 -> PIR value: 1, Elasped time: 123
16:09:53.549 -> PIR value: 1, Elasped time: 123
16:09:53.549 -> PIR value: 1, Elasped time: 123
16:09:53.596 -> PIR value: 1, Elasped time: 245
16:09:53.641 -> PIR value: 1, Elasped time: 246
16:09:53.641 -> PIR value: 1, Elasped time: 246
16:09:53.687 -> PIR value: 1, Elasped time: 246
16:09:53.734 -> PIR value: 1, Elasped time: 368
16:09:53.780 -> PIR value: 1, Elasped time: 369
16:09:53.780 -> PIR value: 1, Elasped time: 369
16:09:53.827 -> PIR value: 1, Elasped time: 491
```



```
16:09:56.382 -> PIR value: 1, Elasped time: 2950
16:09:56.382 -> PIR value: 1, Elasped time: 2950
16:09:56.429 -> PIR value: 1, Elasped time: 3073
16:09:56.476 -> PIR value: 1, Elasped time: 3073
16:09:56.523 -> PIR value: 1, Elasped time: 3073
16:09:56.523 -> PIR value: 1, Elasped time: 3195
16:09:56.569 -> PIR value: 1, Elasped time: 3196
16:09:56.615 -> PIR value: 1, Elasped time: 3196
16:09:56.662 -> PIR value: 1, Elasped time: 3196
16:09:56.662 -> PIR value: 0, Elasped time: 3318
16:09:56.708 -> PIR value: 0, Elasped time: 3318
16:09:56.754 -> PIR value: 0, Elasped time: 3319
16:09:56.800 -> PIR value: 0, Elasped time: 3441
16:09:56.846 -> PIR value: 0, Elasped time: 3441
16:09:56.846 -> PIR value: 0, Elasped time: 3441
16:09:56.892 -> PIR value: 0, Elasped time: 3442
16:09:56.939 -> PIR value: 0, Elasped time: 3564
16:09:56.986 -> PIR value: 0, Elasped time: 3564
16:09:56.986 -> PIR value: 0, Elasped time: 3564
16:09:57.033 -> PIR value: 0, Elasped time: 3687
16:09:57.078 -> PIR value: 0, Elasped time: 3687
16:09:57.125 -> PIR value: 0, Elasped time: 3687
```

☑ Autoscroll ☑ Show timestamp | Newline | 9600 baud | Clear output



```
16:10:02.749 -> PIR value: 0, Elasped time: 9341
16:10:02.796 -> PIR value: 0, Elasped time: 9341
16:10:02.796 -> PIR value: 0, Elasped time: 9464
16:10:02.843 -> PIR value: 0, Elasped time: 9464
16:10:02.888 -> PIR value: 0, Elasped time: 9464
16:10:02.935 -> PIR value: 0, Elasped time: 9464
16:10:02.935 -> PIR value: 0, Elasped time: 9587
16:10:02.982 -> PIR value: 0, Elasped time: 9587
16:10:03.029 -> PIR value: 0, Elasped time: 9587
16:10:03.077 -> PIR value: 0, Elasped time: 9710
16:10:03.077 -> PIR value: 0, Elasped time: 9710
16:10:03.122 -> PIR value: 0, Elasped time: 9710
16:10:03.169 -> PIR value: 0, Elasped time: 9710
16:10:03.216 -> PIR value: 0, Elasped time: 9833
16:10:03.216 -> PIR value: 0, Elasped time: 9833
16:10:03.262 -> PIR value: 0, Elasped time: 9833
16:10:03.309 -> PIR value: 0, Elasped time: 9956
16:10:03.356 -> PIR value: 0, Elasped time: 9956
16:10:03.403 -> PIR value: 0, Elasped time: 9956
16:10:03.403 -> PIR value: 0
16:10:03.450 -> PIR value: 0
16:10:03.450 -> PIR value: 0
```

☑ Autoscroll ☑ Show timestamp | Newline | 9600 baud | Clear output

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

Challenge 6.1

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 .

```
RTC_DATA_ATTR int bootCount = 0;

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

  Serial.begin(115200);
  delay(1000); //Take some time to open up the Serial Monitor

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

  wakeup_cause();

  // TO DO: Enable ext0 for RTC17 pin 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 default.json for WOKWI.

2. Connect PIR motion sensor based on the information given in the sketch above.

   Note: Please make sure to **unplug your USB cable before connecting your PIR motion sensor**. Please ensure that you connect the 3 wires correctly before you reconnect the USB cable as incorrectly connecting the PIR sensor's 3 wires could damage the sensor. Revisit Figure 8 of this practical sheet if you are unsure, **the black wire must be plugged on the Ground side.**

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_06_ext0_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.

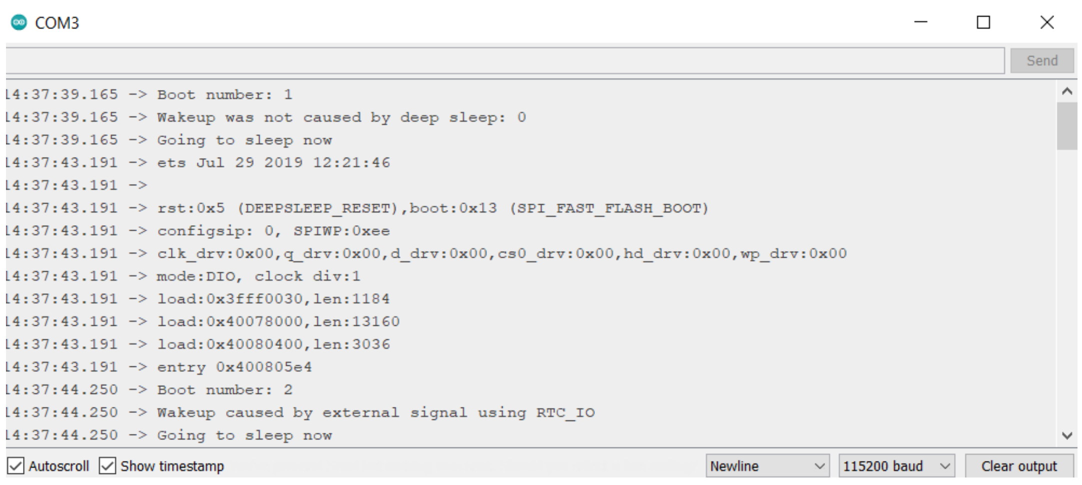
TO DO

1. Complete all of the TO DO tasks as shown in the comments in tutorial_06_ext0_challenge.ino.

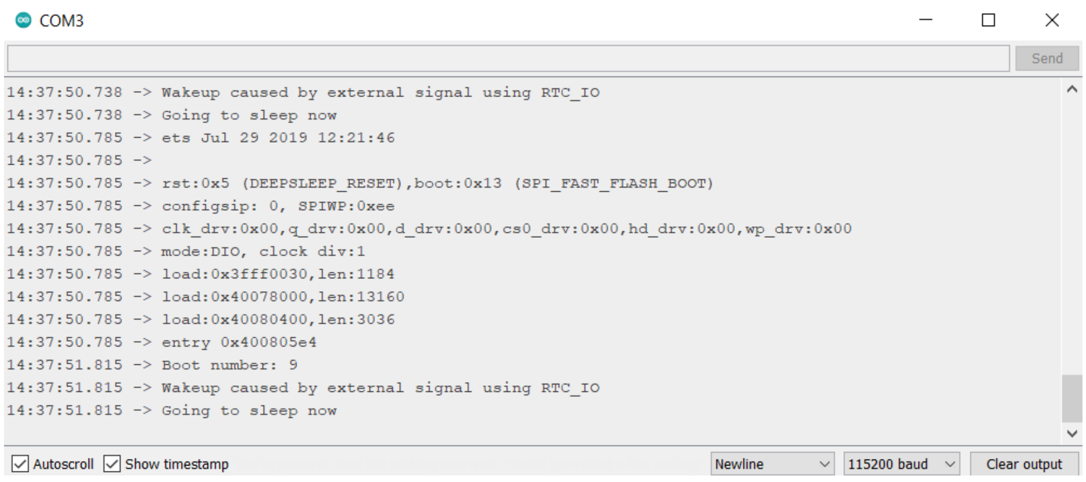
2. If you have a Serial Monitor open already, please close it. **Verify** your modification.

3. **Upload** the code.

4. Go to **Tools** and then select **Serial Monitor**. You should now see the output of similar to what is shown below. Please note that the majority of the message shown here is automatically generated, except for those in `Serial.print()` or `Serial.println()` in tutorial_06_ext0_challenge.ino.



```
14:37:39.165 -> Boot number: 1
14:37:39.165 -> Wakeup was not caused by deep sleep: 0
14:37:39.165 -> Going to sleep now
14:37:43.191 -> ets Jul 29 2019 12:21:46
14:37:43.191 -> 
14:37:43.191 -> rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
14:37:43.191 -> configsip: 0, SPIWP:0xee
14:37:43.191 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
14:37:43.191 -> mode:DIO, clock div:1
14:37:43.191 -> load:0x3fff0030,len:1184
14:37:43.191 -> load:0x40078000,len:13160
14:37:43.191 -> load:0x40080400,len:3036
14:37:43.191 -> entry 0x400805e4
14:37:44.250 -> Boot number: 2
14:37:44.250 -> Wakeup caused by external signal using RTC_IO
14:37:44.250 -> Going to sleep now
```



```
14:37:50.738 -> Wakeup caused by external signal using RTC_IO
14:37:50.738 -> Going to sleep now
14:37:50.785 -> ets Jul 29 2019 12:21:46
14:37:50.785 ->
14:37:50.785 -> rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
14:37:50.785 -> configsip: 0, SPIWP:0xee
14:37:50.785 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
14:37:50.785 -> mode:DIO, clock div:1
14:37:50.785 -> load:0x3fff0030,len:1184
14:37:50.785 -> load:0x40078000,len:13160
14:37:50.785 -> load:0x40080400,len:3036
14:37:50.785 -> entry 0x400805e4
14:37:51.815 -> Boot number: 9
14:37:51.815 -> Wakeup caused by external signal using RTC_IO
14:37:51.815 -> Going to sleep now
```

Note: Please note that Sleep Mode doesn't get simulated correctly in WOKWI and you will not be able to see the increase in Boot number, i.e., it will always be 1. Therefore it cannot be put to sleep and woken up by PIR sensor.

5. Note the message after Boot number 2, which indicates the RTC pin as the wakeup source. It should be noted that because of the delay in PIR sensor, the output remains `HIGH` for some time, as shown by a number of rebooting (i.e., up to 9) after Boot number 2.