Pakistan Institute of Engineering and Applied Sciences

Lab Report

Lab-01: Digital Input and Output

Group Members:

- M. Hammad Tahir
- Ehmaan Shafqat
- Khadija Arif
- Ali Raza

Course Name: Practical IoT

Instructor: Dr. Naveed Akhtar

Submission Date: 20/04/2025

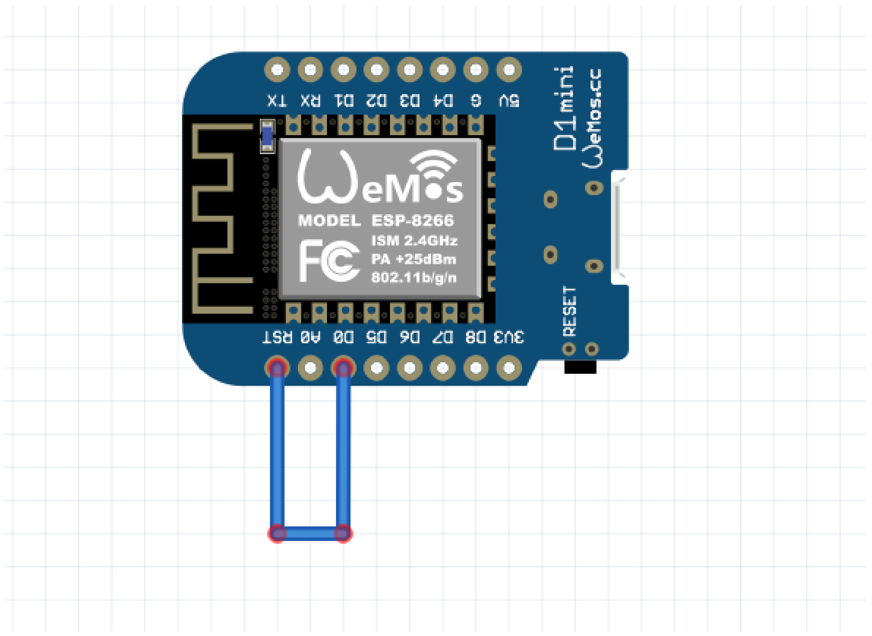
Required Components:

- WeMos D1 Mini (ESP8266)
- Esp32
- 10kΩ resistor
- Push Button (momentary type)
- Breadboard & Jumper Wires
- USB cable for programming

Task 1: ESP8266 Deep Sleep with Timer Wake Up

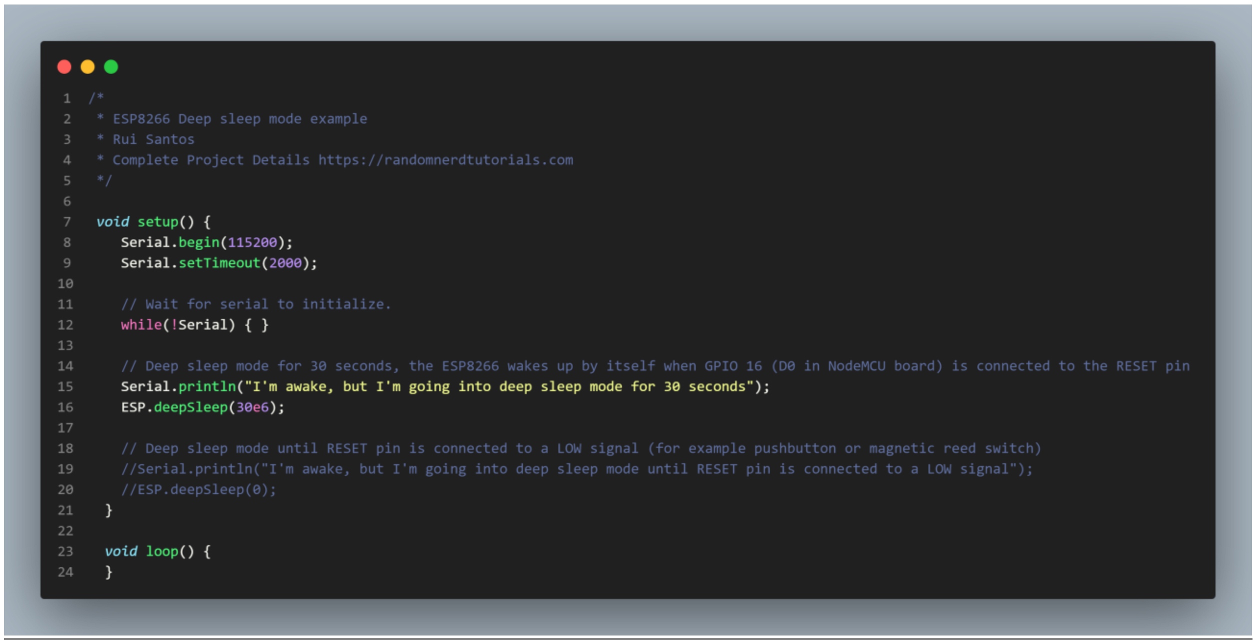
Circuit Diagram:

To use timer wake up with ESP8266, you need to connect the RST pin to GPIO 16 which is labeled as D0. As shown in figure:



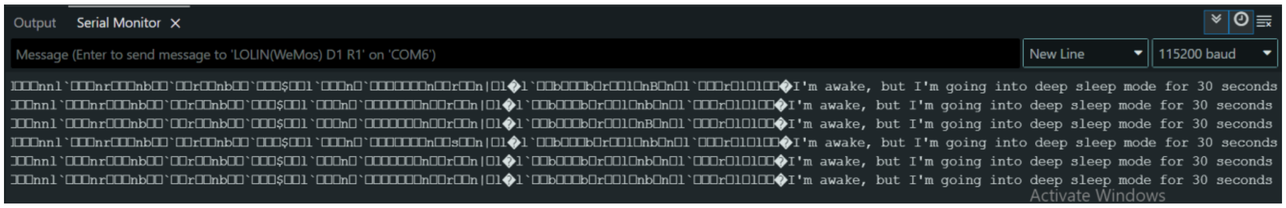
Code:

```
/*
 * ESP8266 Deep sleep mode example
 * Rui Santos
 * Complete Project Details https://randomnerdtutorials.com
 */

 void setup() {
    Serial.begin(115200);
    Serial.setTimeout(2000);

    // Wait for serial to initialize.
    while(!Serial) { }

    // Deep sleep mode for 30 seconds, the ESP8266 wakes up by itself
when GPIO 16 (D0 in NodeMCU board) is connected to the RESET pin
    Serial.println("I'm awake, but I'm going into deep sleep mode for
30 seconds");
    ESP.deepSleep(30e6);

    // Deep sleep mode until RESET pin is connected to a LOW signal
(for example pushbutton or magnetic reed switch)
    //Serial.println("I'm awake, but I'm going into deep sleep mode
until RESET pin is connected to a LOW signal");
    //ESP.deepSleep(0);
  }

  void loop() {
  }
```

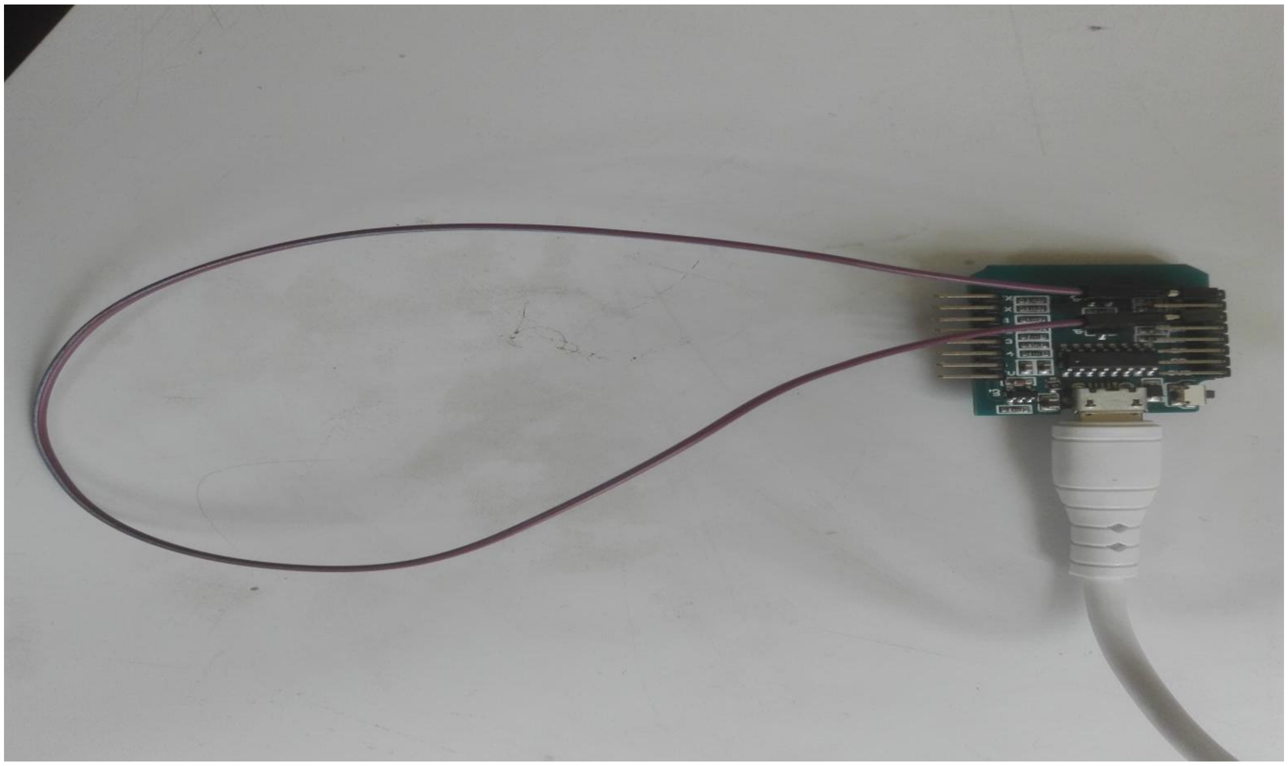


```cpp
/*
 * ESP8266 Deep sleep mode example
 * Rui Santos
 * Complete Project Details https://randomnerdtutorials.com
 */

void setup() {
   Serial.begin(115200);
   Serial.setTimeout(2000);

   // Wait for serial to initialize.
   while(!Serial) { }

   // Deep sleep mode for 30 seconds, the ESP8266 wakes up by itself when GPIO 16 (D0 in NodeMCU board) is connected to the RESET pin
   Serial.println("I'm awake, but I'm going into deep sleep mode for 30 seconds");
   ESP.deepSleep(30e6);

   // Deep sleep mode until RESET pin is connected to a LOW signal (for example pushbutton or magnetic reed switch)
   //Serial.println("I'm awake, but I'm going into deep sleep mode until RESET pin is connected to a LOW signal");
   //ESP.deepSleep(0);
 }

 void loop() {
 }
```

Output:


Output
Serial Monitor
Message (Enter to send message to 'LOLIN(WeMos) D1 R1' on 'COM6')
New Line
115200 baud
Activate Windows

```
]□□□nnl`□□□nr□□□nb□□`□□r□□nb□□`□□□$□□1`□□□n□`□□□□□□□n□□r□□n|□1�1`□□b□□□b□r□□1□nB□n□1`□□□r□1□1□□�I'm awake, but I'm going into deep sleep mode for 30 seconds
]□□nnl`□□□nr□□□nb□□`□□r□□nb□□`□□□$□□1`□□□n□`□□□□□□□n□□r□□n|□1�1`□□b□□□b□r□□1□nb□n□1`□□□r□1□1□□�I'm awake, but I'm going into deep sleep mode for 30 seconds
]□□nnl`□□□nr□□□nb□□`□□r□□nb□□`□□□$□□1`□□□n□`□□□□□□□n□□r□□n|□1�1`□□b□□□b□r□□1□nB□n□1`□□□r□1□1□□�I'm awake, but I'm going into deep sleep mode for 30 seconds
]□□□nnl`□□□nr□□□nb□□`□□r□□nb□□`□□□$□□1`□□□n□`□□□□□□□n□□s□□n|□1�1`□□b□□□b□r□□1□nb□n□1`□□□r□1□1□□�I'm awake, but I'm going into deep sleep mode for 30 seconds
]□□nnl`□□□nr□□□nb□□`□□r□□nb□□`□□□$□□1`□□□n□`□□□□□□□n□□r□□n|□1�1`□□b□□□b□r□□1□nb□n□1`□□□r□1□1□□�I'm awake, but I'm going into deep sleep mode for 30 seconds
]□□nnl`□□□nr□□□nb□□`□□r□□nb□□`□□□$□□1`□□□n□`□□□□□□□n□□r□□n|□1�1`□□b□□□b□r□□1□nb□n□1`□□□r□1□1□□�I'm awake, but I'm going into deep sleep mode for 30 seconds
```



Code explanation:

In this example, we print a message in the Serial Monitor:

```
Serial.println("I'm awake, but I'm going into deep sleep mode
until RESET pin is connected to a LOW signal");
```

After that, the ESP8266 goes to sleep for 30 seconds.

```
ESP.deepSleep(30e6);
```

To put the ESP8266 in deep sleep, you use ESP.deepsleep(uS) and pass as argument the sleep time in microseconds.

In this case, 30e6 corresponds to 30000000 microseconds which is equal to 30 seconds.

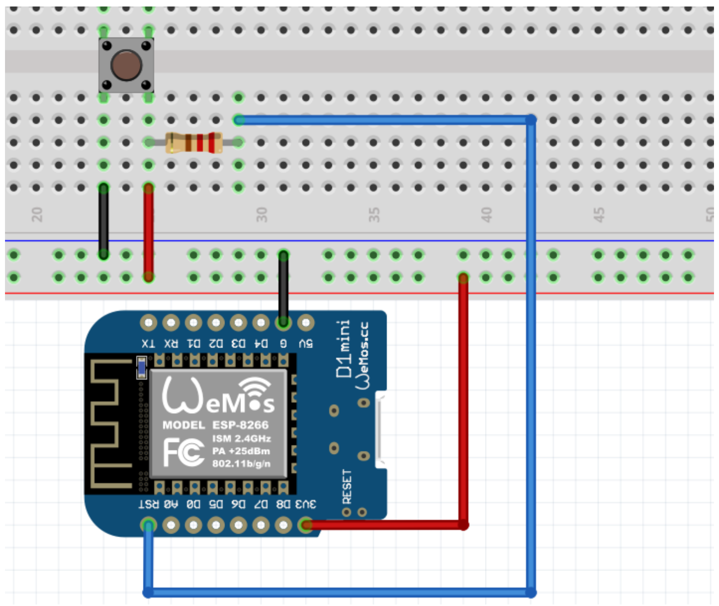
After uploading the code, press the RST button to start running the code, and then connect RST to GPIO 16.

Task 2: ESP8266 Deep Sleep with External Wake Up

The Problem:

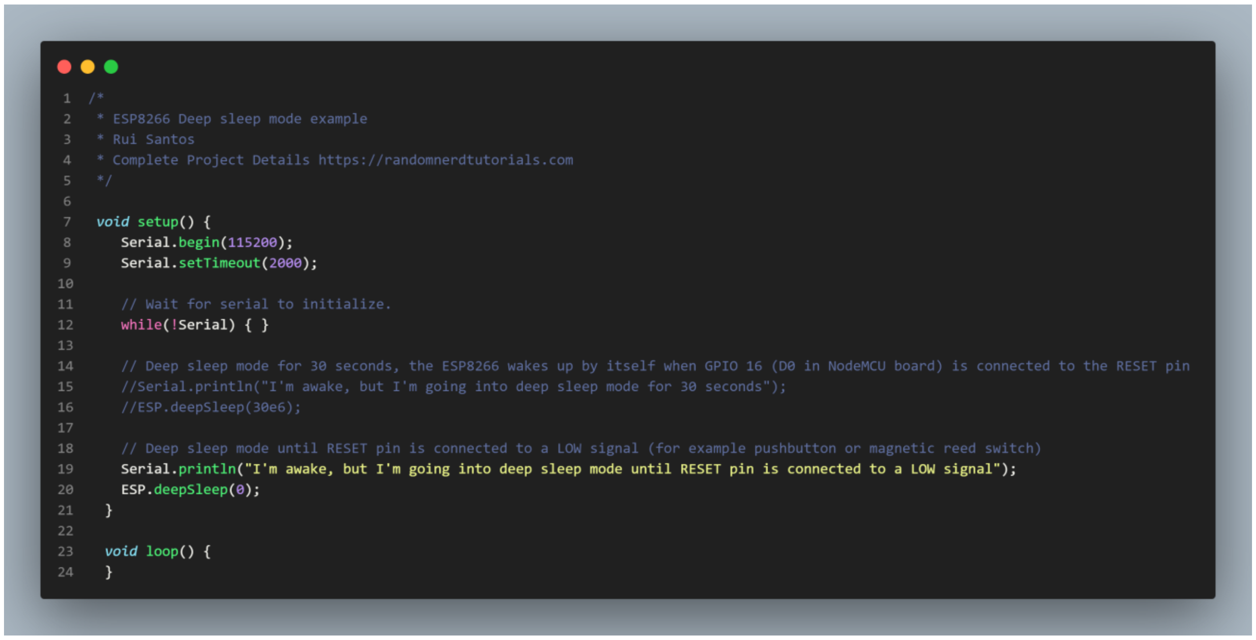
• Mechanical push buttons can produce spurious signals (bounces) which may result in multiple unwanted triggers.

Circuit diagram:



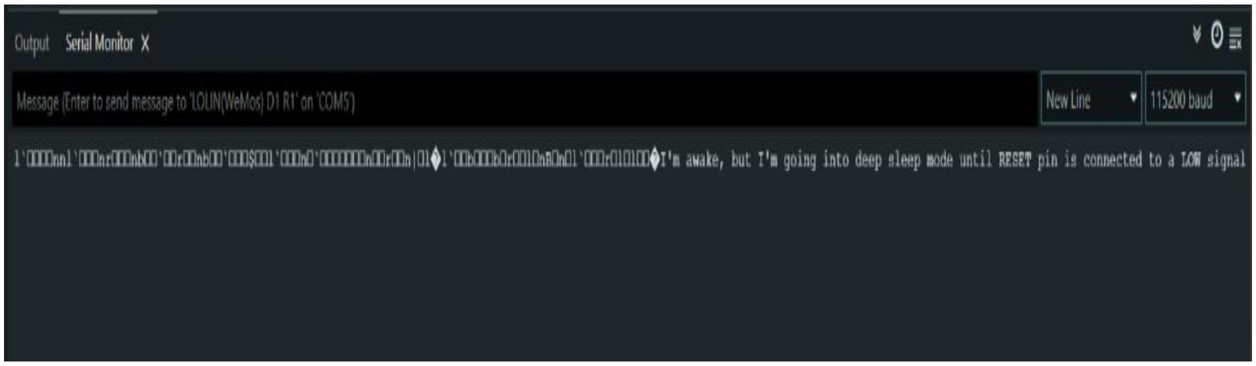
Code:

```
/*
 * ESP8266 Deep sleep mode example
 * Rui Santos
 * Complete Project Details https://randomnerdtutorials.com
 */

 void setup() {
    Serial.begin(115200);
    Serial.setTimeout(2000);

    // Wait for serial to initialize.
    while(!Serial) { }

    // Deep sleep mode for 30 seconds, the ESP8266 wakes up by itself
when GPIO 16 (D0 in NodeMCU board) is connected to the RESET pin
    //Serial.println("I'm awake, but I'm going into deep sleep mode
for 30 seconds");
    //ESP.deepSleep(30e6);

    // Deep sleep mode until RESET pin is connected to a LOW signal
(for example pushbutton or magnetic reed switch)
    Serial.println("I'm awake, but I'm going into deep sleep mode
until RESET pin is connected to a LOW signal");
    ESP.deepSleep(0);
  }

  void loop() {
  }
```

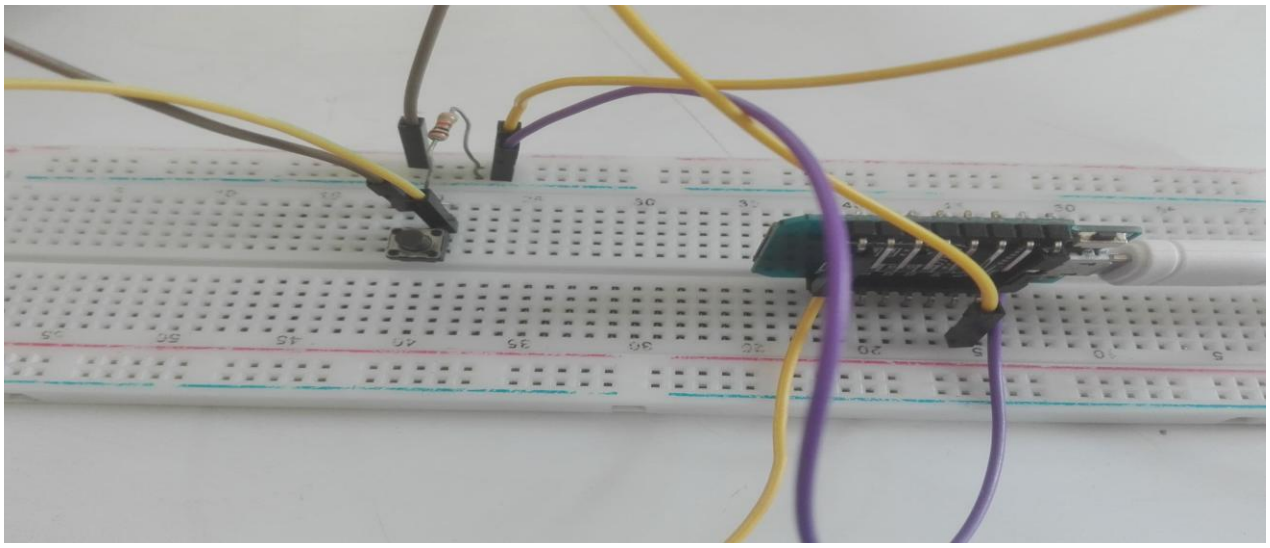


```
/*
 * ESP8266 Deep sleep mode example
 * Rui Santos
 * Complete Project Details https://randomnerdtutorials.com
 */

void setup() {
   Serial.begin(115200);
   Serial.setTimeout(2000);

   // Wait for serial to initialize.
   while(!Serial) { }

   // Deep sleep mode for 30 seconds, the ESP8266 wakes up by itself when GPIO 16 (D0 in NodeMCU board) is connected to the RESET pin
   //Serial.println("I'm awake, but I'm going into deep sleep mode for 30 seconds");
   //ESP.deepSleep(30e6);

   // Deep sleep mode until RESET pin is connected to a LOW signal (for example pushbutton or magnetic reed switch)
   Serial.println("I'm awake, but I'm going into deep sleep mode until RESET pin is connected to a LOW signal");
   ESP.deepSleep(0);
 }

 void loop() {
 }
```

Output:



```
Message (Enter to send message to 'LOLIN(WeMos) D1 R1' on 'COM5')    New Line    115200 baud

l`□□□□nnl`□□□nr□□□nb□□`□□r□□nb□□`□□□$□□l`□□□n□`□□□□□□□n□□r□□n|□l�l`□□b□□□b□r□□l□nB□n□l`□□□r□l□l□□�I'm awake, but I'm going into deep sleep mode until RESET pin is connected to a LOW signal
```



Explanation:

This code puts the ESP8266 in deep sleep mode for an indefinite period of time. For that, you just need to pass 0 as an argument to the deepSleep() function:

ESP.deepSleep(0);

The ESP will only wake up when something resets the board. In this case, it's the press of a pushbutton that pulls the RST pin to GND.

When you press the pushbutton, the ESP8266 wakes up, does the programmed task and goes back to sleep until a new reset event is triggered.

Additional Tasks (Optional): ***For ESP32***

Timer Wake Up

Code:

```
/* Simple Deep Sleep with Timer Wake Up
   ESP32 offers a deep sleep mode for effective power saving as power
is an important factor for IoT applications. In this mode CPUs, most
of the RAM, and all the digital peripherals which are clocked
   from APB_CLK are powered off. The only parts of the chip which can
still be powered on are: RTC controller, RTC peripherals ,and RTC
memories This code displays the most basic deep sleep with a timer to
wake it up and how to store data in RTC memory to use it over reboots
This code is under Public Domain License.
   Author: Pranav Cherukupalli <cherukupallip@gmail.com> */

   #define uS_TO_S_FACTOR 1000000ULL  // Conversion factor for micro
seconds to seconds
   #define TIME_TO_SLEEP  5           // Time ESP32 will go to sleep
(in seconds)

   RTC_DATA_ATTR int bootCount = 0;

   // Method to print the reason by which ESP32 has been awaken from
sleep
   void print_wakeup_reason(){
     esp_sleep_wakeup_cause_t wakeup_reason;

     wakeup_reason = esp_sleep_get_wakeup_cause();
     switch(wakeup_reason)
     {
       case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by
external signal using RTC_IO"); break;
       case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by
external signal using RTC_CNTL"); break;
       case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by
timer"); break;
       case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused
by touchpad"); break;
       case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by
ULP program"); break;
       default : Serial.printf("Wakeup was not caused by deep sleep:
%d\n",wakeup_reason); break;
     }
   }

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

     // Increment boot number and print it every reboot
     ++bootCount;
     Serial.println("Boot number: " + String(bootCount));

     // Print the wakeup reason for ESP32
     print_wakeup_reason();

     // First we configure the wake up source We set our ESP32 to wake
up every 5 seconds
     esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
     Serial.println("Setup ESP32 to sleep for every " +
String(TIME_TO_SLEEP) +
     " Seconds");

     /*
     Next we decide what all peripherals to shut down/keep on
     By default, ESP32 will automatically power down the peripherals
     not needed by the wakeup source, but if you want to be a
poweruser
     this is for you. Read in detail at the API docs
     http://esp-idf.readthedocs.io/en/latest/api-
reference/system/deep_sleep.html
     Left the line commented as an example of how to configure
peripherals.
     The line below turns off all RTC peripherals in deep sleep.
     */
     //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH,
ESP_PD_OPTION_OFF);
     //Serial.println("Configured all RTC Peripherals to be powered
down in sleep");

     // Now that we have setup a wake cause and if needed setup the
peripherals state in deep sleep, we can now start going to deep sleep.
     // In the case that no wake up sources were provided but deep
sleep was started, it will sleep forever unless hardware reset occurs.
     Serial.println("Going to sleep now");
     delay(1000);
     Serial.flush();
     esp_deep_sleep_start();
     Serial.println("This will never be printed");
   }

   void loop(){
     // This is not going to be called
   }
```

Define the Sleep Time

These first two lines of code define the period of time the ESP32 will be sleeping.

```
#define uS_TO_S_FACTOR 1000000ULL /* Conversion factor for
micro seconds to seconds */
#define TIME_TO_SLEEP  5         /* Time ESP32 will go to sleep
(in seconds) */
```

This example uses a conversion factor from microseconds to seconds, so that you can set the sleep time in the TIME_TO_SLEEP variable in seconds. In this case, the example will put the ESP32 into deep sleep mode for 5 seconds.

Save Data on RTC Memories

With the ESP32, you can save data on the RTC memories. The ESP32 has 8kB SRAM on the RTC part, called RTC fast memory. The data saved here is not erased during deep sleep. However, it is erased when you press the reset button (the button labeled EN on the ESP32 board).

To save data on the RTC memory, you just have to add RTC_DATA_ATTR before a variable definition. The example saves the bootCount variable on the RTC memory. This variable will count how many times the ESP32 has woken up from deep sleep.

```
RTC_DATA_ATTR int bootCount = 0;
```

Wake Up Reason

Then, the code defines the print_wakeup_reason() function, that prints the source that caused the wake-up from deep sleep.

```
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Wakeup caused by ULP program"); break;
    default:                        Serial.printf("Wakeup was not caused by deep sleep: %d\n", wakeup_reason); break;
  }
}
```

The setup()

In the setup() is where you should put your code. You need to write all the instructions before calling the esp_deep_sleep_start() function.

This example starts by initializing the serial communication at a baud rate of 115200.

```
Serial.begin(115200);
```

Then, the bootCount variable is increased by one in every reboot, and that number is printed in the serial monitor.

```
++bootCount;
Serial.println("Boot number: " + String(bootCount));
```

Then, the code calls the print_wakeup_reason() function, but you can call any function you want to perform a desired task. For example, you may want to wake up your ESP32 once a day to read a value from a sensor.

Next, the code defines the wake-up source by using the following function:

```
esp_sleep_enable_timer_wakeup(time_in_us)
```

This function accepts as argument the time to sleep in microseconds as we've seen previously. In our case, we have the following:

```
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
```

Then, after all the tasks are performed, the ESP32 goes to sleep by calling the following function:

```
esp_deep_sleep_start()
```

As soon as you call the esp_deep_sleep_start() function, the ESP32 will go to sleep and will not run any code written after this function. When it wakes up from deep sleep, it will run the code from the beginning.

loop()

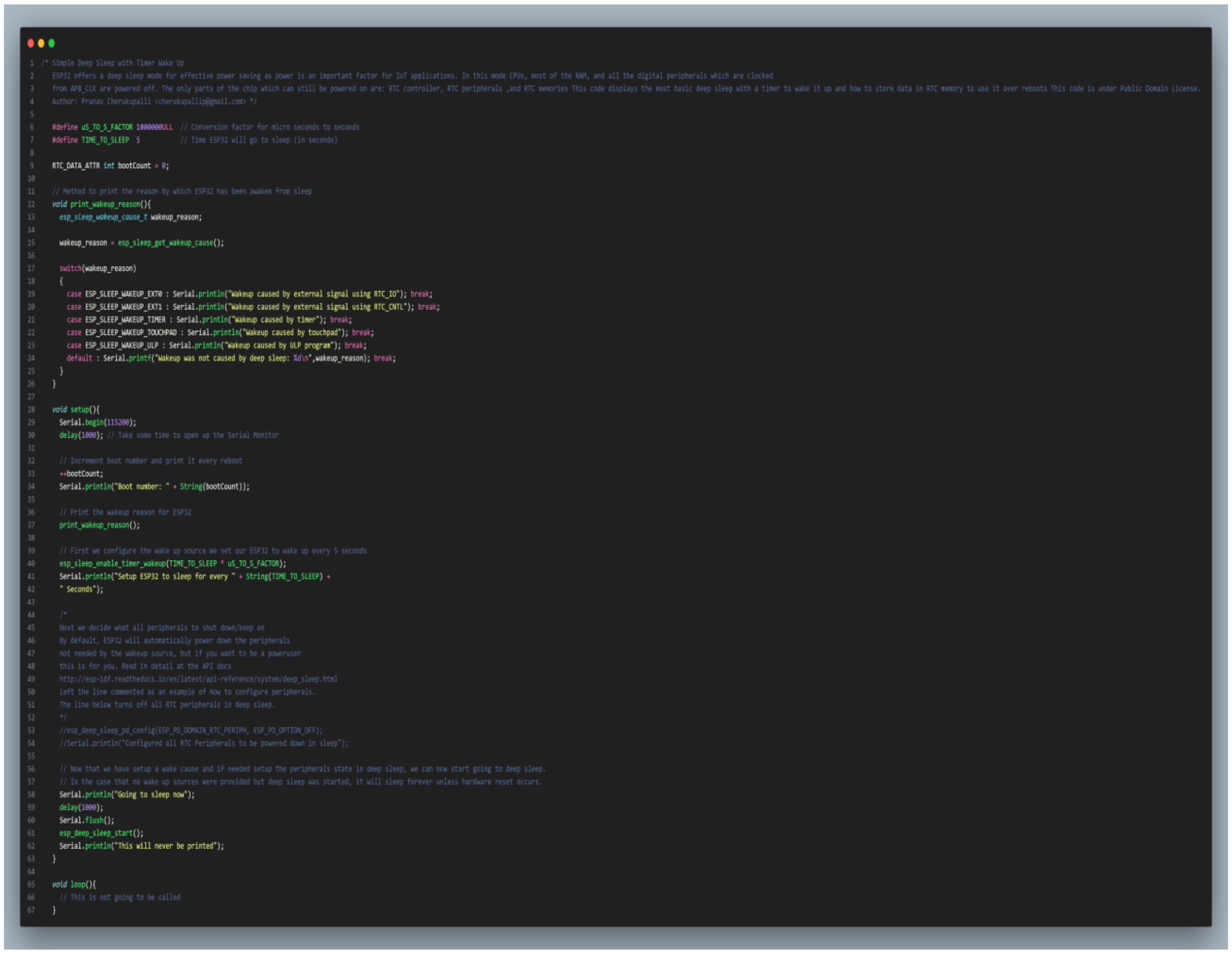
The loop() section is empty because the ESP32 will sleep before reaching this part of the code. So, you need to write all your tasks in the setup() before calling the esp_deep_sleep_start() function.

Testing the Timer Wake Up

Upload the example sketch to your ESP32. Make sure you have the right board and COM port selected. Open the Serial Monitor at a baud rate of 115200.

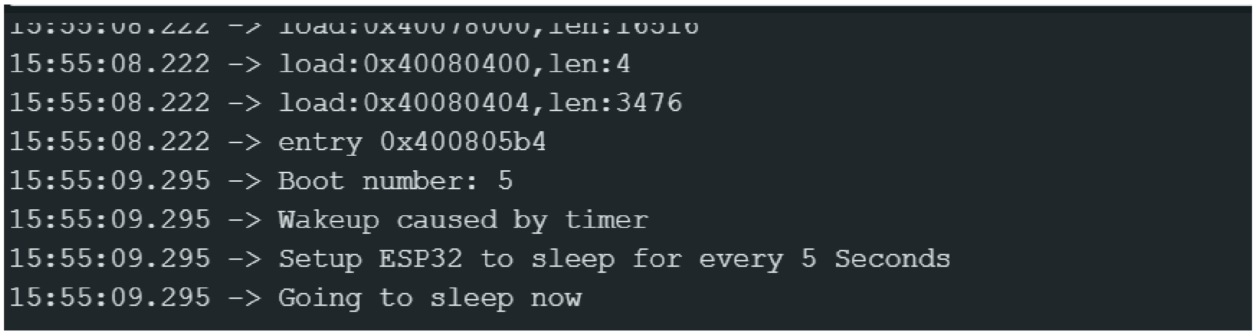
Every 5 seconds, the ESP32 wakes up, prints a message on the serial monitor, and goes to deep sleep again.

Every time the ESP32 wakes up, the bootCount variable increases. It also prints the wake-up reason as shown in the figure below.



```
/* Simple Deep Sleep with Timer Wake Up
  ESP32 offers a deep sleep mode for effective power saving as power is an important factor for IoT applications. In this mode CPUs, most of the RAM, and all the digital peripherals which are clocked
  from APB_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals ,and RTC memories This code displays the most basic deep sleep with a timer to wake it up and how to store data in RTC memory to use it over reboots This code is under Public Domain License.
  Author: Pranav Cherukupalli <cherukupallip@gmail.com> */

  #define uS_TO_S_FACTOR 1000000ULL  // Conversion factor for micro seconds to seconds
  #define TIME_TO_SLEEP  5          // Time ESP32 will go to sleep (in seconds)

  RTC_DATA_ATTR int bootCount = 0;

  // Method to print the reason by which ESP32 has been awaken from sleep
  void print_wakeup_reason(){
    esp_sleep_wakeup_cause_t wakeup_reason;

    wakeup_reason = esp_sleep_get_wakeup_cause();

    switch(wakeup_reason)
    {
      case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
      case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
      case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
      case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
      case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
      default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
    }
  }

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

    // Increment boot number and print it every reboot
    ++bootCount;
    Serial.println("Boot number: " + String(bootCount));

    // Print the wakeup reason for ESP32
    print_wakeup_reason();

    // First we configure the wake up source We set our ESP32 to wake up every 5 seconds
    esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
    Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
    " Seconds");

    /*
    Next we decide what all peripherals to shut down/keep on
    By default, ESP32 will automatically power down the peripherals
    not needed by the wakeup source, but if you want to be a poweruser
    this is for you. Read in detail at the API docs
    http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep_sleep.html
    Left the line commented as an example of how to configure peripherals.
    The line below turns off all RTC peripherals in deep sleep.
    */
    //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
    //Serial.println("Configured all RTC Peripherals to be powered down in sleep");

    // Now that we have setup a wake cause and if needed setup the peripherals state in deep sleep, we can now start going to deep sleep.
    // In the case that no wake up sources were provided but deep sleep was started, it will sleep forever unless hardware reset occurs.
    Serial.println("Going to sleep now");
    delay(1000);
    Serial.flush();
    esp_deep_sleep_start();
    Serial.println("This will never be printed");
  }

  void loop(){
    // This is not going to be called
  }
```

Output:



```
15:55:08.222 -> load:0x40078000,len:16516
15:55:08.222 -> load:0x40080400,len:4
15:55:08.222 -> load:0x40080404,len:3476
15:55:08.222 -> entry 0x400805b4
15:55:09.295 -> Boot number: 5
15:55:09.295 -> Wakeup caused by timer
15:55:09.295 -> Setup ESP32 to sleep for every 5 Seconds
15:55:09.295 -> Going to sleep now
```

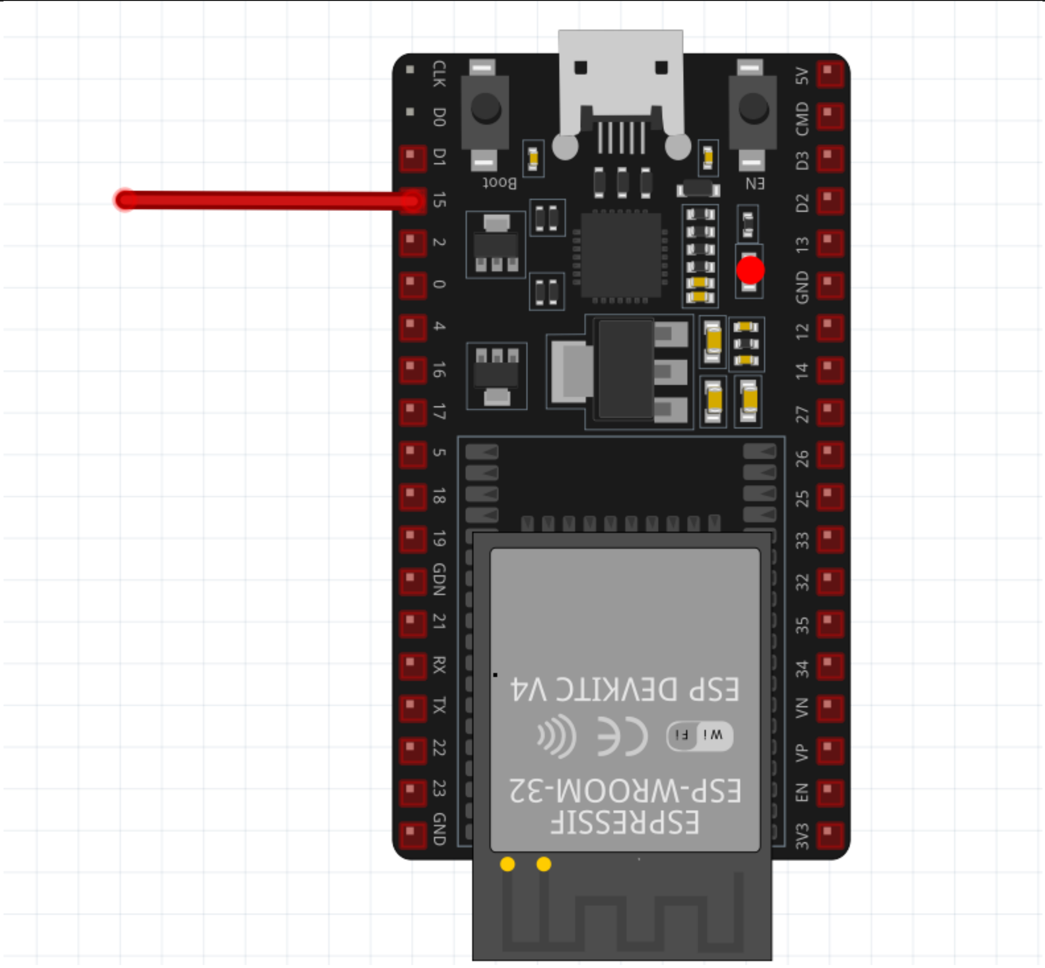
Touch Wake Up

Code:

```
/* Simple Deep Sleep with Timer Wake Up
   ESP32 offers a deep sleep mode for effective power saving as power
is an important factor for IoT applications. In this mode CPUs, most
of the RAM, and all the digital peripherals which are clocked
   from APB_CLK are powered off. The only parts of the chip which can
still be powered on are: RTC controller, RTC peripherals ,and RTC
memories This code displays the most basic deep sleep with a timer to
wake it up and how to store data in RTC memory to use it over reboots
This code is under Public Domain License.
   Author: Pranav Cherukupalli <cherukupallip@gmail.com> */

   #define uS_TO_S_FACTOR 1000000ULL  // Conversion factor for micro
seconds to seconds
   #define TIME_TO_SLEEP  5          // Time ESP32 will go to sleep
(in seconds)

   RTC_DATA_ATTR int bootCount = 0;

   // Method to print the reason by which ESP32 has been awaken from
sleep
   void print_wakeup_reason(){
     esp_sleep_wakeup_cause_t wakeup_reason;

     wakeup_reason = esp_sleep_get_wakeup_cause();

     switch(wakeup_reason)
     {
       case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by
external signal using RTC_IO"); break;
       case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by
external signal using RTC_CNTL"); break;
       case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by
timer"); break;
       case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused
by touchpad"); break;
```

```
       case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by 
ULP program"); break;
       default : Serial.printf("Wakeup was not caused by deep sleep: 
%d\n",wakeup_reason); break;
     }
   }

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

     // Increment boot number and print it every reboot
     ++bootCount;
     Serial.println("Boot number: " + String(bootCount));

     // Print the wakeup reason for ESP32
     print_wakeup_reason();

     // First we configure the wake up source We set our ESP32 to wake 
up every 5 seconds
     esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
     Serial.println("Setup ESP32 to sleep for every " + 
String(TIME_TO_SLEEP) +
     " Seconds");

     /*
     Next we decide what all peripherals to shut down/keep on
     By default, ESP32 will automatically power down the peripherals
     not needed by the wakeup source, but if you want to be a 
poweruser
     this is for you. Read in detail at the API docs
     http://esp-idf.readthedocs.io/en/latest/api-
reference/system/deep_sleep.html
     Left the line commented as an example of how to configure 
peripherals.
     The line below turns off all RTC peripherals in deep sleep.
     */
     //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, 
ESP_PD_OPTION_OFF);
```

```
    //Serial.println("Configured all RTC Peripherals to be powered
down in sleep");

    // Now that we have setup a wake cause and if needed setup the
peripherals state in deep sleep, we can now start going to deep sleep.
    // In the case that no wake up sources were provided but deep
sleep was started, it will sleep forever unless hardware reset occurs.
    Serial.println("Going to sleep now");
    delay(1000);
    Serial.flush();
    esp_deep_sleep_start();
    Serial.println("This will never be printed");
  }

  void loop(){
    // This is not going to be called
  }
```



Setting the Threshold

The first thing you need to do is setting a threshold value for the touch pins.

```
#define THRESHOLD 40 // Greater the value, more the sensitivity
```

The values read by a touch pin decrease when you touch it. The threshold value means that the ESP32 will wake up when the value read on the touch pin is below 40. You can adjust that value depending on the desired sensitivity.

Important: however, if you're using an ESP32-S2 or ESP32-S3 model, things work a little
differently. That's why there's a different section in the code defining a different threshold for those boards. In this case, the lower the value, the more the sensitivity.

```
#if CONFIG_IDF_TARGET_ESP32
#define THRESHOLD 40 // Greater the value, more the sensitivity
#else // ESP32-S2 and ESP32-S3 + default for other chips (to be adjusted)
#define THRESHOLD 5000 // Lower the value, more the sensitivity
#endif
```

Setting Touch Pins as a Wake-up Source

To set a touch pin as a wake-up source, you can use
the `touchSleepWakeUpEnable()` function that accepts as arguments the touch pin and the threshold value that will wake up the board.
In this example, it sets GPIO 15 (T3) and GPIO 27 (T7) as wake-up sources with the same threshold value.

```
#if CONFIG_IDF_TARGET_ESP32
  // Setup sleep wakeup on Touch Pad 3 + 7 (GPIO15 + GPIO 27)
  touchSleepWakeUpEnable(T3, THRESHOLD);
  touchSleepWakeUpEnable(T7, THRESHOLD);
```

If you're using an ESP32-S2 or S3 model, the example just defines wake-up on GPIO 3 (T3). Note that the touch pin numbering might be different depending on the board model you're using.

```
#else // ESP32-S2 + ESP32-S3
  // Setup sleep wakeup on Touch Pad 3 (GPIO3)
  touchSleepWakeUpEnable(T3, THRESHOLD);
```

Finally, call the `esp_deep_sleep_start()` to put the ESP32 in deep sleep mode.

```
esp_deep_sleep_start();
```

These were the general instructions to set up touch pins as a wake-up source. Now, let's take a look at the other sections of the code.

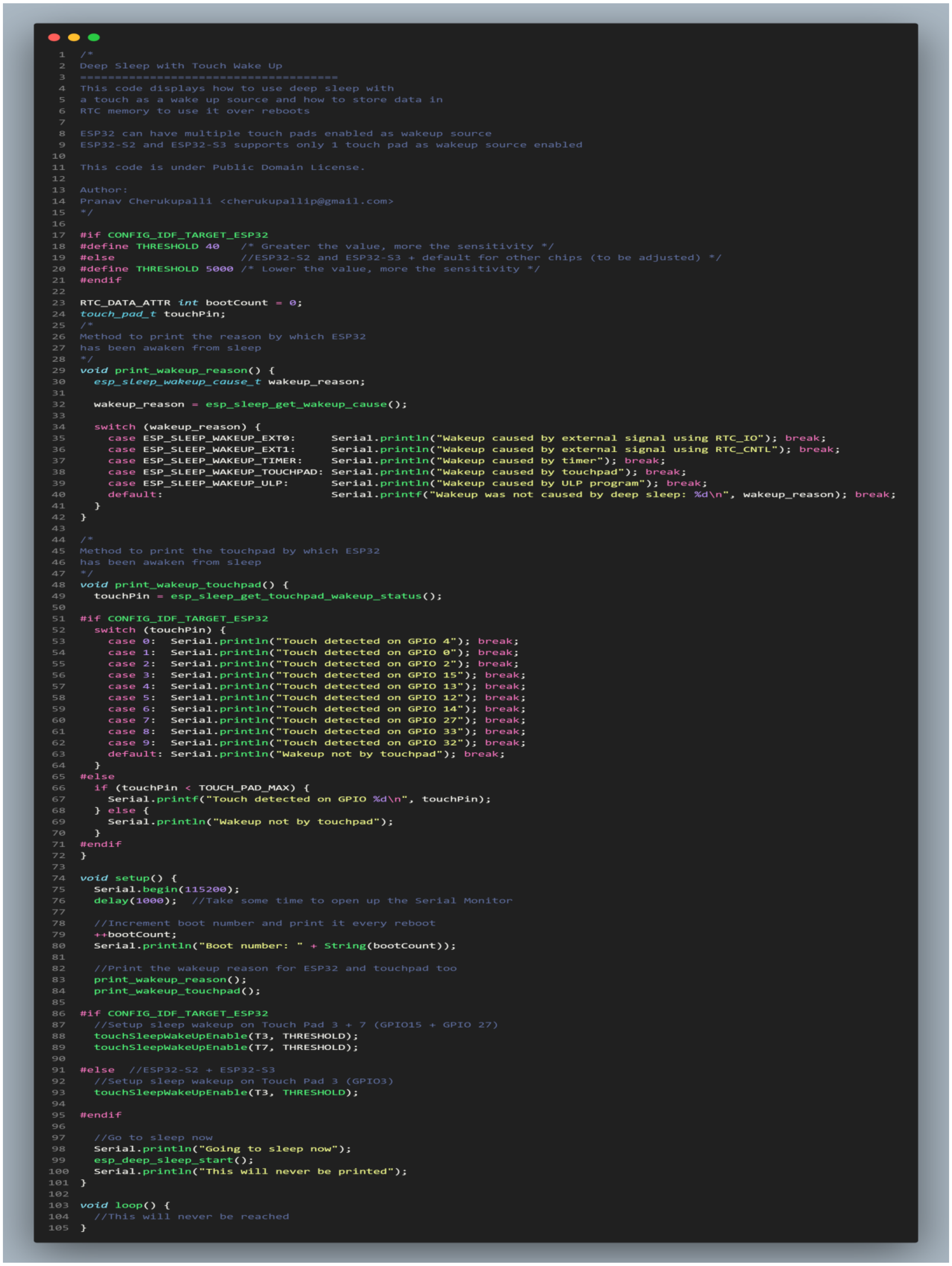
setup()

When the ESP32 wakes up from sleep, it will run the code from the start until it finds the `esp_deep_sleep_start()` function.
In this particular example, we have a control variable called `bootCount` that will be increased between each sleep cycle so that we have an idea of how many times the ESP32 woke up.

```
// Increment boot number and print it every reboot
++bootCount;
Serial.println("Boot number: " + String(bootCount));
```

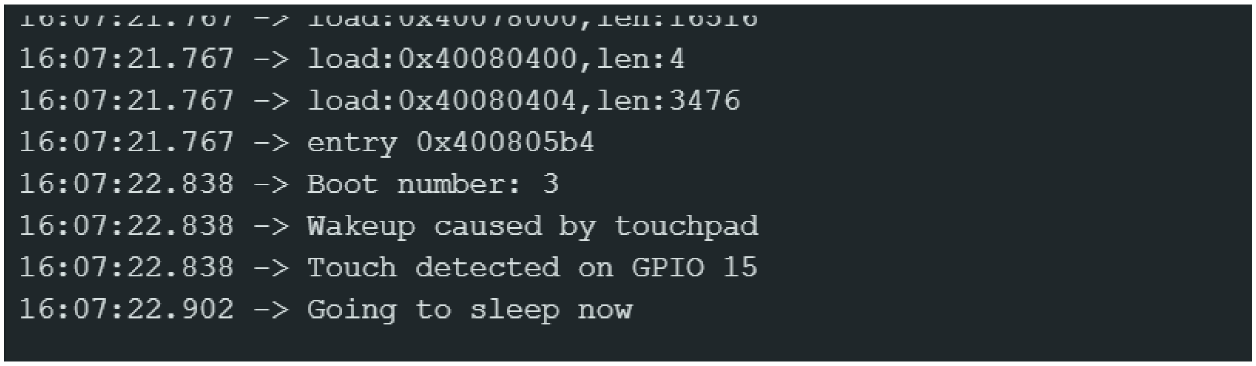
We also call the `print_wakeup_reason()` and `print_wakeup_touchpad()` functions defined earlier in the code to print the wake-up reason and the pin that caused the wake-up.

```
// Print the wakeup reason for ESP32 and touchpad too
print_wakeup_reason();
print_wakeup_touchpad();
```



```cpp
/*
Deep Sleep with Touch Wake Up
=====================================
This code displays how to use deep sleep with
a touch as a wake up source and how to store data in
RTC memory to use it over reboots

ESP32 can have multiple touch pads enabled as wakeup source
ESP32-S2 and ESP32-S3 supports only 1 touch pad as wakeup source enabled

This code is under Public Domain License.

Author:
Pranav Cherukupalli <cherukupallip@gmail.com>
*/

#if CONFIG_IDF_TARGET_ESP32
#define THRESHOLD 40   /* Greater the value, more the sensitivity */
#else                  //ESP32-S2 and ESP32-S3 + default for other chips (to be adjusted) */
#define THRESHOLD 5000 /* Lower the value, more the sensitivity */
#endif

RTC_DATA_ATTR int bootCount = 0;
touch_pad_t touchPin;
/*
Method to print the reason by which ESP32
has been awaken from sleep
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Wakeup caused by ULP program"); break;
    default:                        Serial.printf("Wakeup was not caused by deep sleep: %d\n", wakeup_reason); break;
  }
}

/*
Method to print the touchpad by which ESP32
has been awaken from sleep
*/
void print_wakeup_touchpad() {
  touchPin = esp_sleep_get_touchpad_wakeup_status();

#if CONFIG_IDF_TARGET_ESP32
  switch (touchPin) {
    case 0:  Serial.println("Touch detected on GPIO 4"); break;
    case 1:  Serial.println("Touch detected on GPIO 0"); break;
    case 2:  Serial.println("Touch detected on GPIO 2"); break;
    case 3:  Serial.println("Touch detected on GPIO 15"); break;
    case 4:  Serial.println("Touch detected on GPIO 13"); break;
    case 5:  Serial.println("Touch detected on GPIO 12"); break;
    case 6:  Serial.println("Touch detected on GPIO 14"); break;
    case 7:  Serial.println("Touch detected on GPIO 27"); break;
    case 8:  Serial.println("Touch detected on GPIO 33"); break;
    case 9:  Serial.println("Touch detected on GPIO 32"); break;
    default: Serial.println("Wakeup not by touchpad"); break;
  }
#else
  if (touchPin < TOUCH_PAD_MAX) {
    Serial.printf("Touch detected on GPIO %d\n", touchPin);
  } else {
    Serial.println("Wakeup not by touchpad");
  }
#endif
}

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

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32 and touchpad too
  print_wakeup_reason();
  print_wakeup_touchpad();

#if CONFIG_IDF_TARGET_ESP32
  //Setup sleep wakeup on Touch Pad 3 + 7 (GPIO15 + GPIO 27)
  touchSleepWakeUpEnable(T3, THRESHOLD);
  touchSleepWakeUpEnable(T7, THRESHOLD);

#else  //ESP32-S2 + ESP32-S3
  //Setup sleep wakeup on Touch Pad 3 (GPIO3)
  touchSleepWakeUpEnable(T3, THRESHOLD);

#endif

  //Go to sleep now
  Serial.println("Going to sleep now");
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

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

Output:



```
16:07:21.767 -> load:0x40078000,len:16516
16:07:21.767 -> load:0x40080400,len:4
16:07:21.767 -> load:0x40080404,len:3476
16:07:21.767 -> entry 0x400805b4
16:07:22.838 -> Boot number: 3
16:07:22.838 -> Wakeup caused by touchpad
16:07:22.838 -> Touch detected on GPIO 15
16:07:22.902 -> Going to sleep now
```

External Wake Up

Code:

```
/*
  Deep Sleep with External Wake Up
  =====================================
  This code displays how to use deep sleep with
  an external trigger as a wake up source and how
  to store data in RTC memory to use it over reboots

  This code is under Public Domain License.

  Hardware Connections
  ======================
  Push Button to GPIO 33 pulled down with a 10K Ohm
  resistor

  NOTE:
  ======
  Only RTC IO can be used as a source for external wake
  source. They are pins: 0,2,4,12-15,25-27,32-39.

  Author:
  Pranav Cherukupalli <cherukupallip@gmail.com>
*/
#include "driver/rtc_io.h"
```

```
#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)  // 2 ^ GPIO_NUMBER in
hex
#define USE_EXT0_WAKEUP          1               // 1 = EXT0 wakeup, 0
= EXT1 wakeup
#define WAKEUP_GPIO              GPIO_NUM_33     // Only RTC IO are
allowed - ESP32 Pin example
RTC_DATA_ATTR int bootCount = 0;

/*
  Method to print the reason by which ESP32
  has been awaken from sleep
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Wakeup caused by
external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Wakeup caused by
external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Wakeup caused by
timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Wakeup caused by
touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Wakeup caused by
ULP program"); break;
    default:                        Serial.printf("Wakeup was not
caused by deep sleep: %d\n", wakeup_reason); break;
  }
}

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

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32
  print_wakeup_reason();

  /*
    First we configure the wake up source
    We set our ESP32 to wake up for an external trigger.
    There are two types for ESP32, ext0 and ext1 .
    ext0 uses RTC_IO to wakeup thus requires RTC peripherals
    to be on while ext1 uses RTC Controller so does not need
    peripherals to be powered on.
    Note that using internal pullups/pulldowns also requires
    RTC peripherals to be turned on.
  */
#if USE_EXT0_WAKEUP
  esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1);  //1 = High, 0 = Low
  // Configure pullup/downs via RTCIO to tie wakeup pins to inactive
level during deepsleep.
  // EXT0 resides in the same power domain (RTC_PERIPH) as the RTC IO
pullup/downs.
  // No need to keep that power domain explicitly, unlike EXT1.
  rtc_gpio_pullup_dis(WAKEUP_GPIO);
  rtc_gpio_pulldown_en(WAKEUP_GPIO);

#else  // EXT1 WAKEUP
  //If you were to use ext1, you would use it like
  esp_sleep_enable_ext1_wakeup_io(BUTTON_PIN_BITMASK(WAKEUP_GPIO),
ESP_EXT1_WAKEUP_ANY_HIGH);
  /*
    If there are no external pull-up/downs, tie wakeup pins to
inactive level with internal pull-up/downs via RTC IO
         during deepsleep. However, RTC IO relies on the RTC_PERIPH
power domain. Keeping this power domain on will
         increase some power comsumption. However, if we turn off the
RTC_PERIPH domain or if certain chips lack the RTC_PERIPH
         domain, we will use the HOLD feature to maintain the pull-up
and pull-down on the pins during sleep.
  */
  rtc_gpio_pulldown_en(WAKEUP_GPIO);  // GPIO33 is tie to GND in order
to wake up in HIGH
  rtc_gpio_pullup_dis(WAKEUP_GPIO);   // Disable PULL_UP in order to
allow it to wakeup on HIGH
#endif
  //Go to sleep now
  Serial.println("Going to sleep now");
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop() {
  //This is not going to be called
}
```

Save Data on RTC Memories

With the ESP32, you can save data on the RTC memories. The ESP32 has 8kB SRAM on the RTC part, called RTC fast memory. The data saved here is not erased during deep sleep. However, it is erased when you press the reset button (the button labeled EN on the ESP32 board).

To save data on the RTC memory, you just have to add `RTC_DATA_ATTR` before a variable definition. The example saves the `bootCount`variable on the RTC memory. This variable will count how many times the ESP32 has woken up from deep sleep.

```
                  RTC_DATA_ATTR int bootCount = 0;
```

Wake Up Reason

Then, the code defines the `print_wakeup_reason()` function, that prints the reason by which the ESP32 has been awakened from sleep.

```
                  void print_wakeup_reason() {
            esp_sleep_wakeup_cause_t wakeup_reason;

         wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Wakeup
caused by external signal using RTC_IO"); break;
```

```
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Wakeup
caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Wakeup
caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Wakeup
caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Wakeup
caused by ULP program"); break;
    default:                        Serial.printf("Wakeup was
not caused by deep sleep: %d\n", wakeup_reason); break;
  }
}
```

setup()

In the `setup()`, you start by initializing the serial communication:

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

Then, you increment one to the `bootCount` variable, and print that variable in the Serial Monitor.

```
++bootCount;
Serial.println("Boot number: " + String(bootCount));
```

Next, you print the wake-up reason using the `print_wakeup_reason()` function defined earlier.

```
//Print the wakeup reason for ESP32
print_wakeup_reason();
```

After this, you need to enable the wake-up sources. We'll test each of the wake-up sources, ext0 and ext1, separately.

ext0

In this example, the ESP32 wakes up when the GPIO 33 is triggered to high:

```
esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1);  // 1 = High, 0 =
Low
```

Instead of GPIO 33, you can use any other RTC GPIO pin. Just define it on the `WAKEUP_GPIO` variable at the beginning of the code.

```
#define WAKEUP_GPIO GPIO_NUM_33     // Only RTC IO are allowed
```

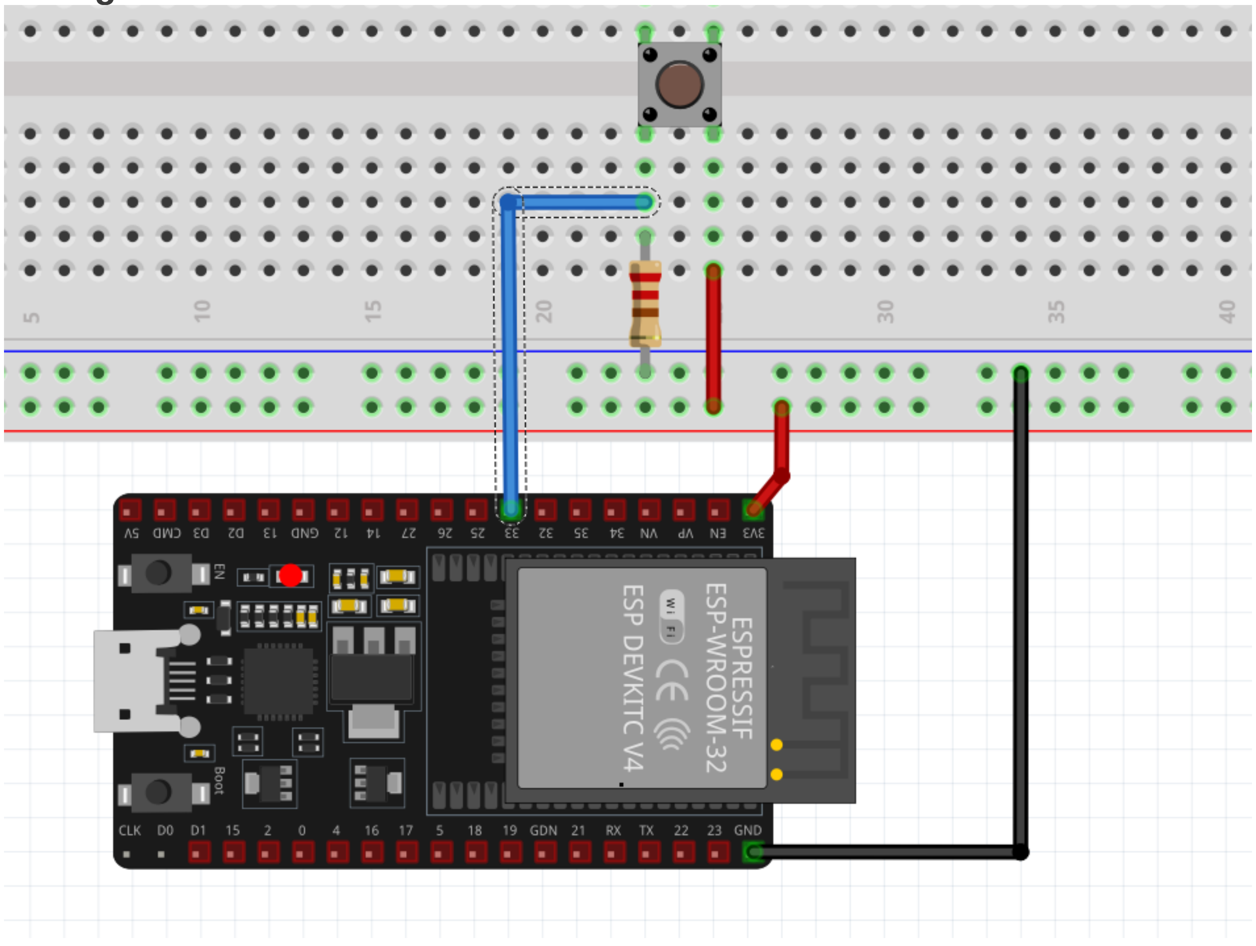
Internal Pull-Up and Pull-Down Resistors

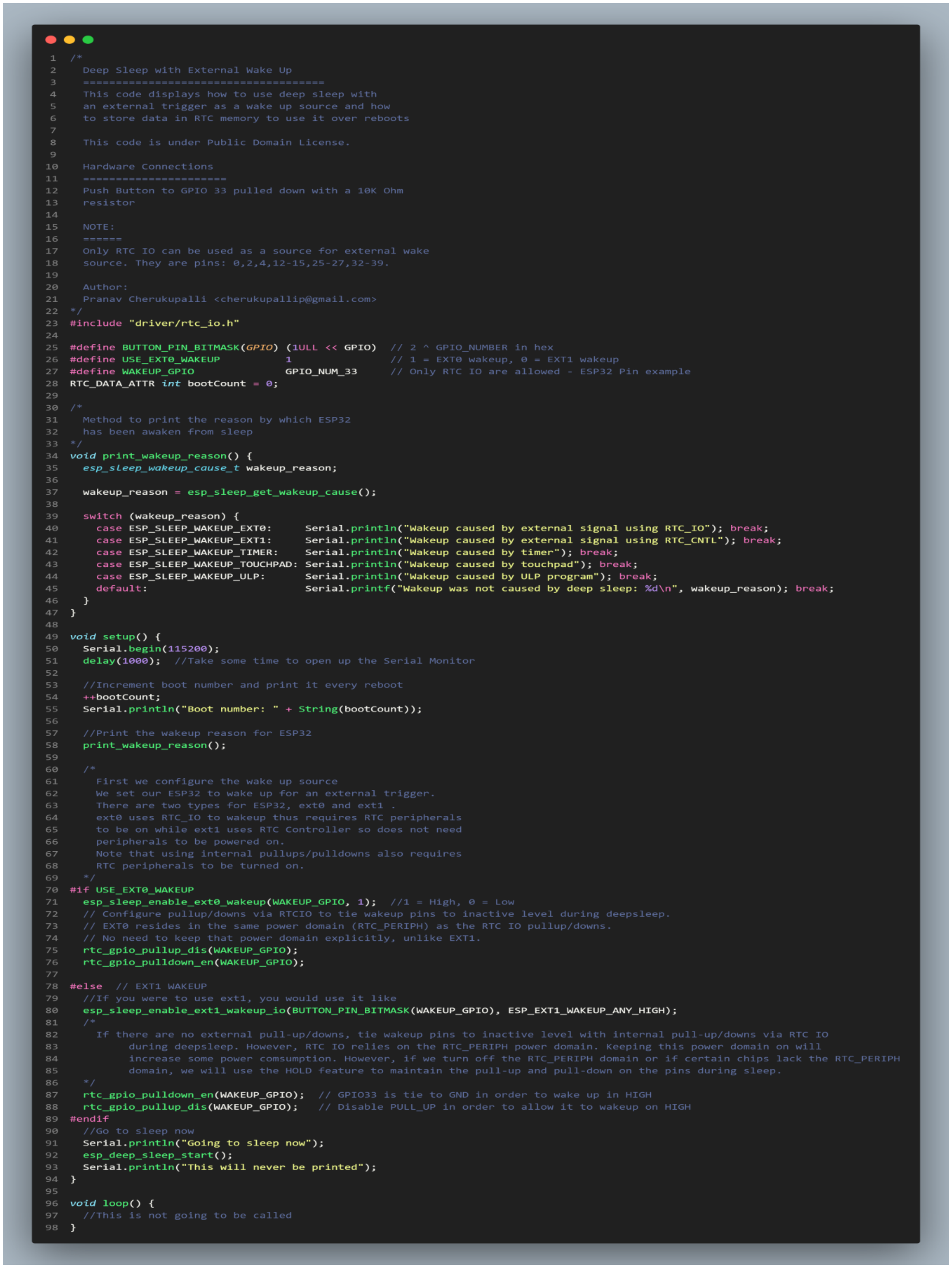
We also call the following two lines:

```
rtc_gpio_pullup_dis(WAKEUP_GPIO);
rtc_gpio_pulldown_en(WAKEUP_GPIO);
```

The `rtc_gpio_pullup_dis()` function disables any internal pull-up resistors on the wake-up GPIO—it ensures that the pin is not unintentionally held high. This is important because a pull-up resistor would keep the pin high, potentially causing unintended wakeups.
The `rtc_gpio_pulldown_en()` function enables the internal pull-down resistor on the wake-up GPIO—it ensures the pin is held low until a valid wakeup signal (HIGH) is received. By configuring the pin with a pull-down resistor, we guarantee that it remains in a stable low state during deep sleep. This stability ensures that the ESP32 wakes up only when the specified GPIO pin receives an external high signal, matching the wakeup condition set by the `esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1)`.

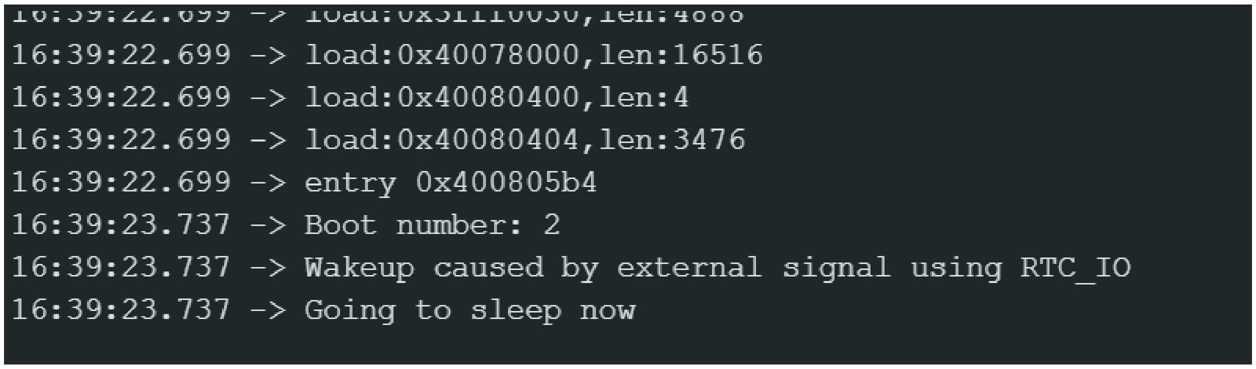
Wiring the Circuit:





```
/*
  Deep Sleep with External Wake Up
  =====================================
  This code displays how to use deep sleep with
  an external trigger as a wake up source and how
  to store data in RTC memory to use it over reboots

  This code is under Public Domain License.

  Hardware Connections
  ======================
  Push Button to GPIO 33 pulled down with a 10K Ohm
  resistor

  NOTE:
  ======
  Only RTC IO can be used as a source for external wake
  source. They are pins: 0,2,4,12-15,25-27,32-39.

  Author:
  Pranav Cherukupalli <cherukupallip@gmail.com>
*/
#include "driver/rtc_io.h"

#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)  // 2 ^ GPIO_NUMBER in hex
#define USE_EXT0_WAKEUP          1               // 1 = EXT0 wakeup, 0 = EXT1 wakeup
#define WAKEUP_GPIO              GPIO_NUM_33     // Only RTC IO are allowed - ESP32 Pin example
RTC_DATA_ATTR int bootCount = 0;

/*
  Method to print the reason by which ESP32
  has been awaken from sleep
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Wakeup caused by ULP program"); break;
    default:                        Serial.printf("Wakeup was not caused by deep sleep: %d\n", wakeup_reason); break;
  }
}

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

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32
  print_wakeup_reason();

  /*
    First we configure the wake up source
    We set our ESP32 to wake up for an external trigger.
    There are two types for ESP32, ext0 and ext1 .
    ext0 uses RTC_IO to wakeup thus requires RTC peripherals
    to be on while ext1 uses RTC Controller so does not need
    peripherals to be powered on.
    Note that using internal pullups/pulldowns also requires
    RTC peripherals to be turned on.
  */
#if USE_EXT0_WAKEUP
  esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1);  //1 = High, 0 = Low
  // Configure pullup/downs via RTCIO to tie wakeup pins to inactive level during deepsleep.
  // EXT0 resides in the same power domain (RTC_PERIPH) as the RTC IO pullup/downs.
  // No need to keep that power domain explicitly, unlike EXT1.
  rtc_gpio_pullup_dis(WAKEUP_GPIO);
  rtc_gpio_pulldown_en(WAKEUP_GPIO);

#else  // EXT1 WAKEUP
  //If you were to use ext1, you would use it like
  esp_sleep_enable_ext1_wakeup_io(BUTTON_PIN_BITMASK(WAKEUP_GPIO), ESP_EXT1_WAKEUP_ANY_HIGH);
  /*
    If there are no external pull-up/downs, tie wakeup pins to inactive level with internal pull-up/downs via RTC IO
         during deepsleep. However, RTC IO relies on the RTC_PERIPH power domain. Keeping this power domain on will
         increase some power comsumption. However, if we turn off the RTC_PERIPH domain or if certain chips lack the RTC_PERIPH
         domain, we will use the HOLD feature to maintain the pull-up and pull-down on the pins during sleep.
  */
  rtc_gpio_pulldown_en(WAKEUP_GPIO);  // GPIO33 is tie to GND in order to wake up in HIGH
  rtc_gpio_pullup_dis(WAKEUP_GPIO);   // Disable PULL_UP in order to allow it to wakeup on HIGH
#endif
  //Go to sleep now
  Serial.println("Going to sleep now");
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop() {
  //This is not going to be called
}
```

Output:



```
16:39:22.699 -> load:0x3fff0030,len:4888
16:39:22.699 -> load:0x40078000,len:16516
16:39:22.699 -> load:0x40080400,len:4
16:39:22.699 -> load:0x40080404,len:3476
16:39:22.699 -> entry 0x400805b4
16:39:23.737 -> Boot number: 2
16:39:23.737 -> Wakeup caused by external signal using RTC_IO
16:39:23.737 -> Going to sleep now
```

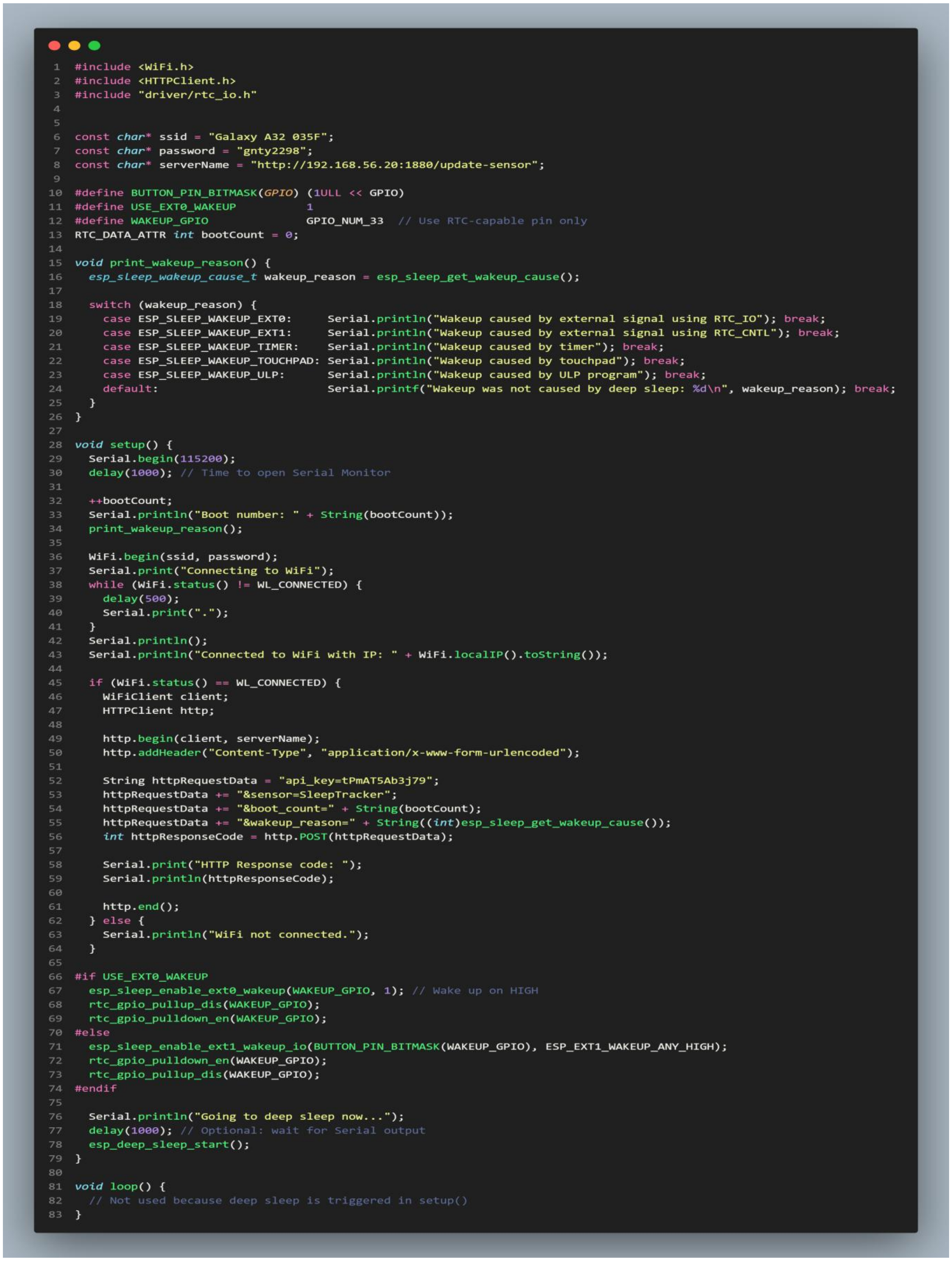
Task 3: ESP8266 Deep Sleep with External Wake Up and sending data on NODE-RED Server

Code:

```
#include <WiFi.h>
#include <HTTPClient.h>
#include "driver/rtc_io.h"


const char* ssid = "Galaxy A32 035F";
const char* password = "gnty2298";
const char* serverName = "http://192.168.56.20:1880/update-sensor";

#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)
#define USE_EXT0_WAKEUP          1
#define WAKEUP_GPIO              GPIO_NUM_33  // Use RTC-capable pin
only
RTC_DATA_ATTR int bootCount = 0;

void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason =
esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Wakeup caused by
external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Wakeup caused by
external signal using RTC_CNTL"); break;
```

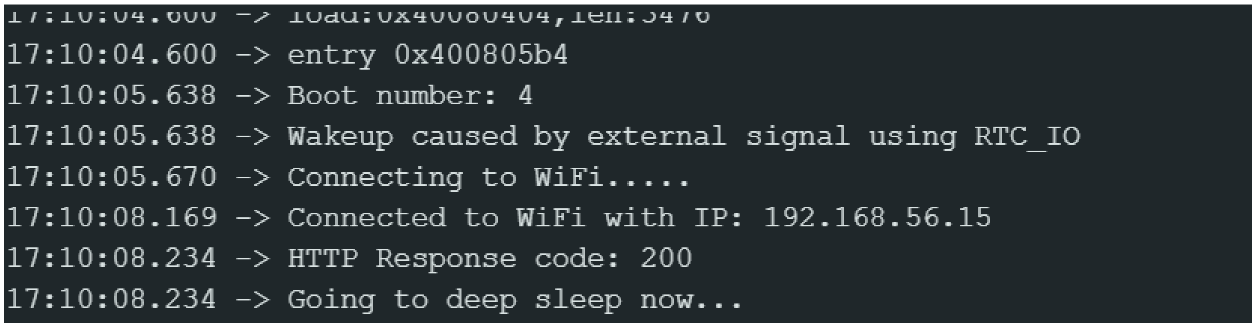
```
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Wakeup caused by
timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Wakeup caused by
touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Wakeup caused by
ULP program"); break;
    default:                        Serial.printf("Wakeup was not
caused by deep sleep: %d\n", wakeup_reason); break;
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000); // Time to open Serial Monitor

  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));
  print_wakeup_reason();

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("Connected to WiFi with IP: " +
WiFi.localIP().toString());

  if (WiFi.status() == WL_CONNECTED) {
    WiFiClient client;
    HTTPClient http;

    http.begin(client, serverName);
    http.addHeader("Content-Type", "application/x-www-form-
urlencoded");

    String httpRequestData = "api_key=tPmAT5Ab3j79";
    httpRequestData += "&sensor=SleepTracker";
    httpRequestData += "&boot_count=" + String(bootCount);
    httpRequestData += "&wakeup_reason=" +
String((int)esp_sleep_get_wakeup_cause());
    int httpResponseCode = http.POST(httpRequestData);

    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);

    http.end();
  } else {
    Serial.println("WiFi not connected.");
  }

#if USE_EXT0_WAKEUP
  esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1); // Wake up on HIGH
  rtc_gpio_pullup_dis(WAKEUP_GPIO);
  rtc_gpio_pulldown_en(WAKEUP_GPIO);
#else
  esp_sleep_enable_ext1_wakeup_io(BUTTON_PIN_BITMASK(WAKEUP_GPIO),
ESP_EXT1_WAKEUP_ANY_HIGH);
  rtc_gpio_pulldown_en(WAKEUP_GPIO);
  rtc_gpio_pullup_dis(WAKEUP_GPIO);
#endif

  Serial.println("Going to deep sleep now...");
  delay(1000); // Optional: wait for Serial output
  esp_deep_sleep_start();
}

void loop() {
  // Not used because deep sleep is triggered in setup()
}
```

Same circuit as we use in previous task (wake-up through external button) but in that code we add code for sending data through HTTP post method that we prepare in last lab. The only new thing is that here I modify the post string so that it can send deep sleep wake up data on the server as shown in below pictures.



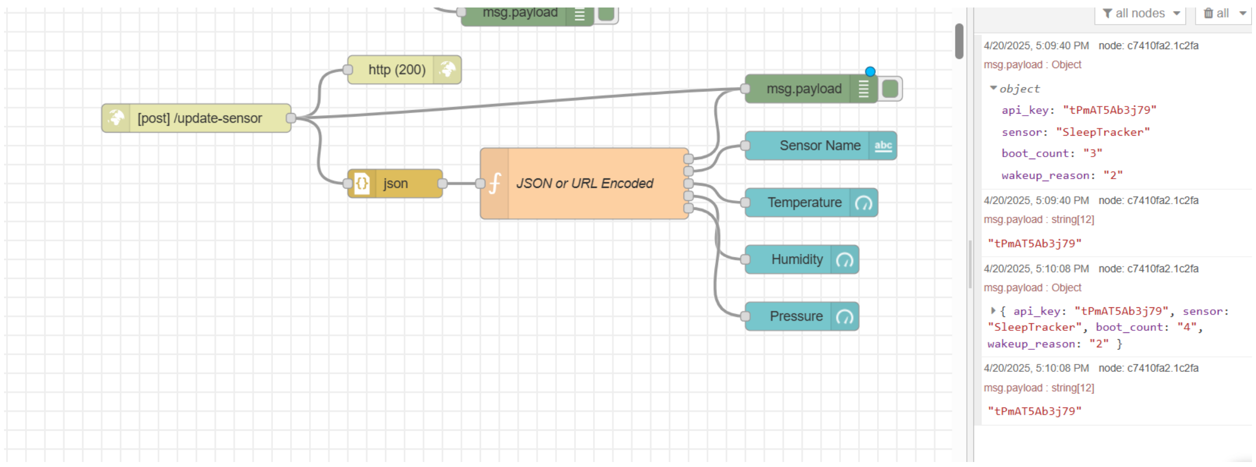
```
#include <WiFi.h>
#include <HTTPClient.h>
#include "driver/rtc_io.h"


const char* ssid = "Galaxy A32 035F";
const char* password = "gnty2298";
const char* serverName = "http://192.168.56.20:1880/update-sensor";

#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)
#define USE_EXT0_WAKEUP          1
#define WAKEUP_GPIO              GPIO_NUM_33  // Use RTC-capable pin only
RTC_DATA_ATTR int bootCount = 0;

void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Wakeup caused by ULP program"); break;
    default:                        Serial.printf("Wakeup was not caused by deep sleep: %d\n", wakeup_reason); break;
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000); // Time to open Serial Monitor

  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));
  print_wakeup_reason();

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("Connected to WiFi with IP: " + WiFi.localIP().toString());

  if (WiFi.status() == WL_CONNECTED) {
    WiFiClient client;
    HTTPClient http;

    http.begin(client, serverName);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    String httpRequestData = "api_key=tPmAT5Ab3j79";
    httpRequestData += "&sensor=SleepTracker";
    httpRequestData += "&boot_count=" + String(bootCount);
    httpRequestData += "&wakeup_reason=" + String((int)esp_sleep_get_wakeup_cause());
    int httpResponseCode = http.POST(httpRequestData);

    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);

    http.end();
  } else {
    Serial.println("WiFi not connected.");
  }

#if USE_EXT0_WAKEUP
  esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1); // Wake up on HIGH
  rtc_gpio_pullup_dis(WAKEUP_GPIO);
  rtc_gpio_pulldown_en(WAKEUP_GPIO);
#else
  esp_sleep_enable_ext1_wakeup_io(BUTTON_PIN_BITMASK(WAKEUP_GPIO), ESP_EXT1_WAKEUP_ANY_HIGH);
  rtc_gpio_pulldown_en(WAKEUP_GPIO);
  rtc_gpio_pullup_dis(WAKEUP_GPIO);
#endif

  Serial.println("Going to deep sleep now...");
  delay(1000); // Optional: wait for Serial output
  esp_deep_sleep_start();
}

void loop() {
  // Not used because deep sleep is triggered in setup()
}
```

Output:



```
17:10:04.600 -> load:0x40080404,len:3476
17:10:04.600 -> entry 0x400805b4
17:10:05.638 -> Boot number: 4
17:10:05.638 -> Wakeup caused by external signal using RTC_IO
17:10:05.670 -> Connecting to WiFi.....
17:10:08.169 -> Connected to WiFi with IP: 192.168.56.15
17:10:08.234 -> HTTP Response code: 200
17:10:08.234 -> Going to deep sleep now...
```

Node-red screen shot:



```
4/20/2025, 5:09:40 PM  node: c7410fa2.1c2fa
msg.payload : Object
▾object
  api_key: "tPmAT5Ab3j79"
  sensor: "SleepTracker"
  boot_count: "3"
  wakeup_reason: "2"
4/20/2025, 5:09:40 PM  node: c7410fa2.1c2fa
msg.payload : string[12]
"tPmAT5Ab3j79"
4/20/2025, 5:10:08 PM  node: c7410fa2.1c2fa
msg.payload : Object
▸{ api_key: "tPmAT5Ab3j79", sensor:
"SleepTracker", boot_count: "4",
wakeup_reason: "2" }
4/20/2025, 5:10:08 PM  node: c7410fa2.1c2fa
msg.payload : string[12]
"tPmAT5Ab3j79"
```