

## Sistemas Embebidos - Memoria práctica 4.



Elvi Mihai Sabau Sabau<sup>[51254875L]</sup>

<sup>1</sup> Universidad de Alicante, Alicante, España.  
[emss5@alu.ua.es](mailto:emss5@alu.ua.es)

**Abstract.** En esta práctica usaremos un arduino nano 33 IoT con headers para conectarnos y detectar dispositivos Bluetooth además de analizar la distancia entre dispositivos mediante la intensidad de la señal.

**Keywords:** IoT, Arduino, LED, Wi-FiNINA, BLT.

<b>Descripción.</b>	<b>3</b>
<b>Instalación de la librería BLE.</b>	<b>3</b>
<b>Funcionamiento BLE.</b>	<b>4</b>
<b>Modo Emisor.</b>	<b>4</b>
<b>Modo monitor.</b>	<b>6</b>
<b>Librería RTCZero.</b>	<b>8</b>
<b>Cálculo de la distancia relativa al RSSI.</b>	<b>9</b>
<b>Funcionamiento combinado WiFi / BLE</b>	<b>10</b>
<b>Conclusiones.</b>	<b>13</b>

## 1 Descripción.

En esta práctica aprenderemos a interactuar con el módulo BLE del dispositivo, realizando varias tareas, entre ellas, el barrido de señales BLE, emparejamiento con dispositivos, y el uso de otras librerías de control del tiempo.

## 2 Instalación de la librería BLE.

Primero instalaremos la librería BLE en nuestro Arduino IDE, esto lo haremos desde herramientas > Administrador de Librerías > Arduino BLE.

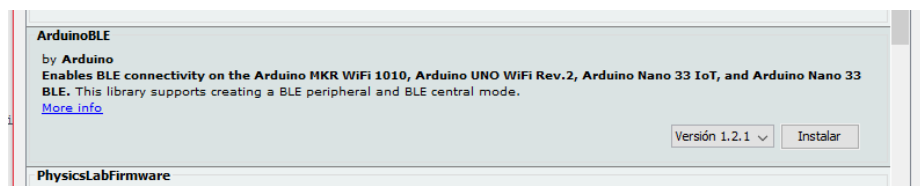


Fig. 1 Imágen de la librería a instalar, listada en el gestor de librerías.

### 3 Funcionamiento BLE.

En este apartado aprenderemos a usar el funcionamiento BLE ( Bluetooth Low Energy ) del Arduino.

#### 3.1 Modo Emisor.

Primero, vamos a conectar nuestro teléfono móvil al dispositivo mediante bluetooth, para ello cargaremos el ejemplo “Arduino BLE/Peripheral/BatteryMonitor”.

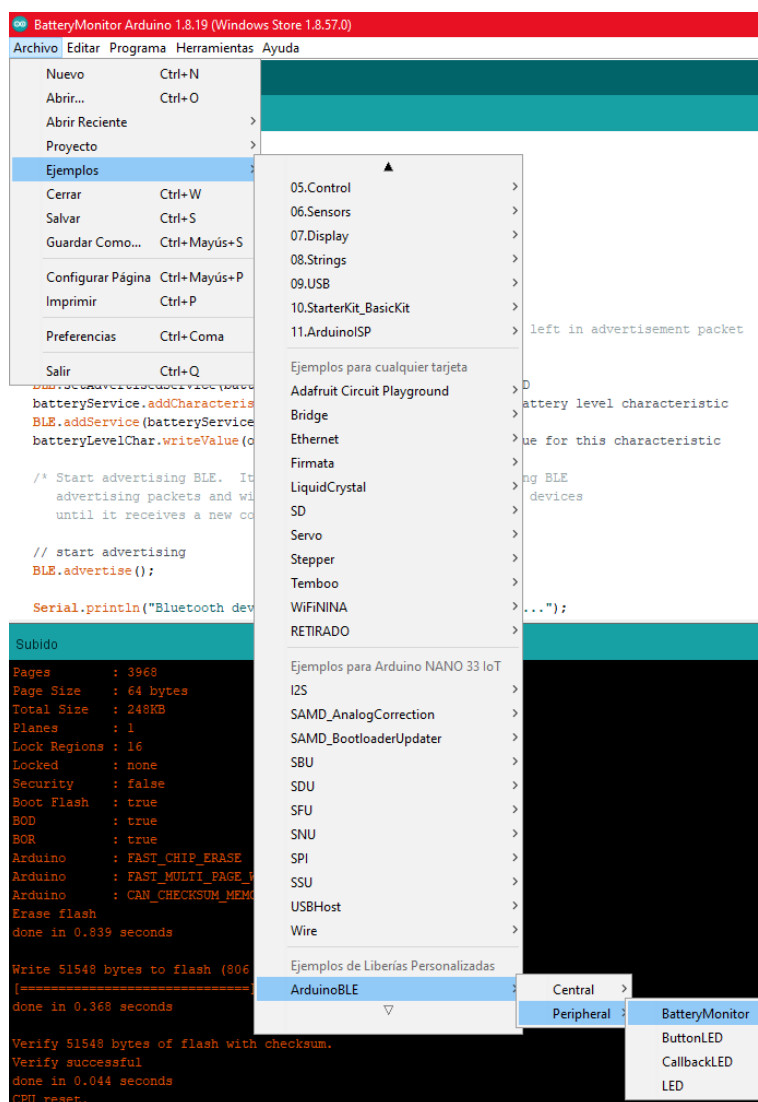
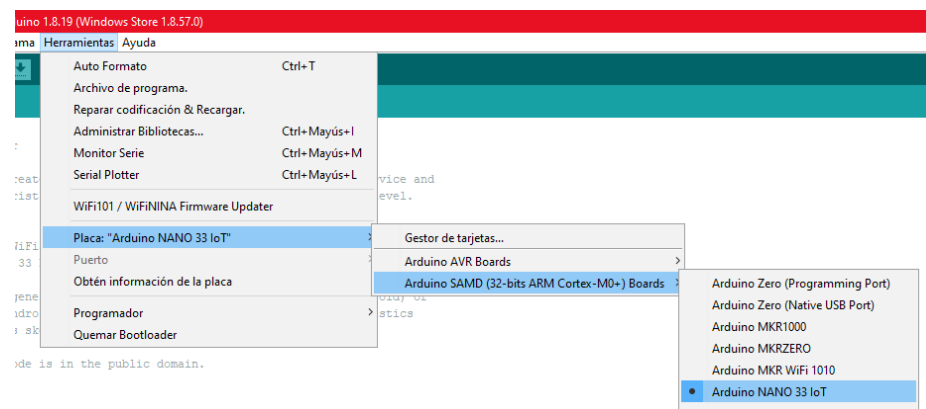


Fig. 2 Imagen que muestra la ruta a tomar para cargar el ejemplo.

También, antes de todo, verificamos que la arquitectura seleccionada es SAMD, ya que la librería Arduino BLE no es compatible con arquitecturas AVR.



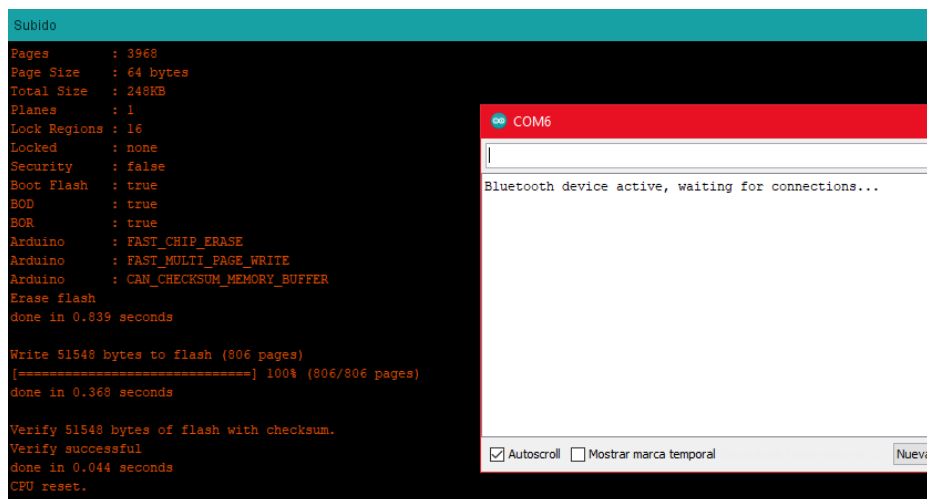
**Fig. 3** Imágen que muestra la arquitectura a seleccionar.

Vemos que en el código, se especifica el nombre del dispositivo, vamos a cambiarlo por “Chipitin”.

```
/* Set a local name for the BLE device
   This name will appear in advertising packets
   and can be used by remote devices to identify this BLE device
   The name can be changed but maybe be truncated based on space left in advertisement packet
*/
BLE.setLocalName("Chipitin");
```

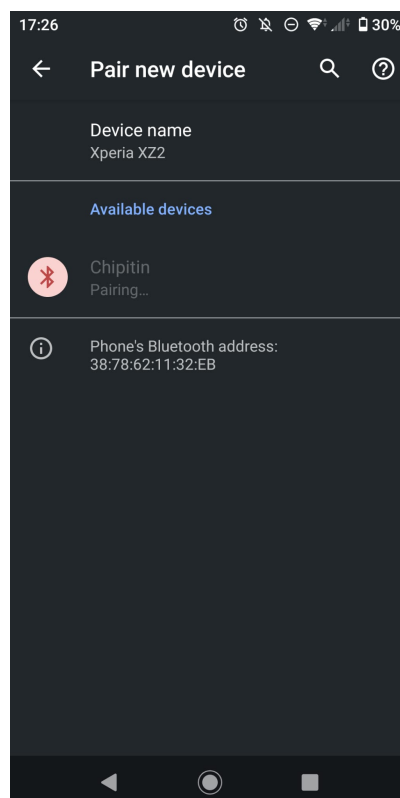
**Fig. 4** Captura de la porción del código de ejemplo que configura el nombre del dispositivo.

Compilamos, cargamos, y revisamos la terminal y la salida sería.



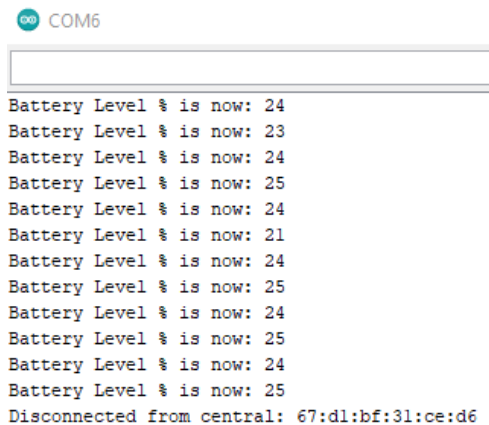
**Fig. 5** Captura que muestra la carga exitosa del ejemplo, y la salida seria, mostrando un mensaje que nos notifica que el dispositivo está listo para recibir conexiones.

Ahora, desde nuestro movil, podremos ver el dispositivo, y conectarnos vía bluetooth.



**Fig. 6** Captura que muestra la lista de dispositivos bluetooth en rango del teléfono.

Al conectarnos al dispositivo, desde la monitor serial, podremos ver que se nos suministra información del teléfono conectado, en este caso la información es el % de batería restante del teléfono.



```

COM6
Battery Level % is now: 24
Battery Level % is now: 23
Battery Level % is now: 24
Battery Level % is now: 25
Battery Level % is now: 24
Battery Level % is now: 21
Battery Level % is now: 24
Battery Level % is now: 25
Battery Level % is now: 24
Battery Level % is now: 25
Battery Level % is now: 24
Battery Level % is now: 25
Disconnected from central: 67:d1:bf:31:ce:d6

```

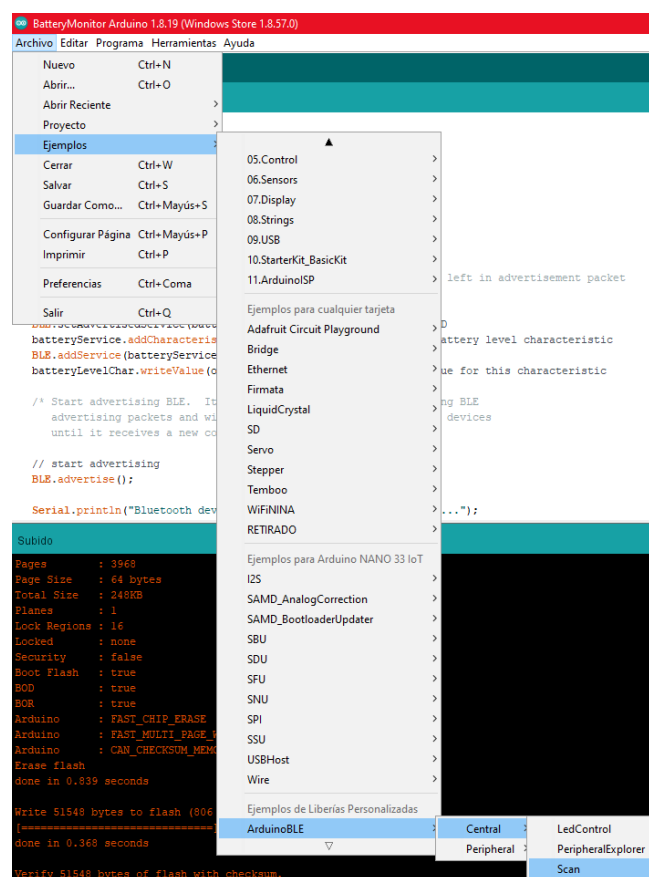
**Fig. 7** Captura que muestra la salida por el monitor serial.

Como podemos observar entre la captura del teléfono y la de la monitor serial, la estimación no es exacta al valor mostrado por el teléfono. Esto se debe a varios factores, como por ejemplo el modo stamina del propio teléfono entre otros.

### 3.2 Modo monitor.

En este apartado, en vez de conectar el dispositivo con el teléfono, lo que haremos será usar el dispositivo como rastreador de señales Bluetooth.

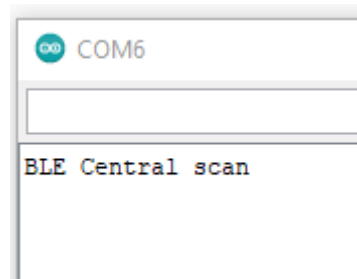
Para ello, primero cargaremos el ejemplo “Arduino BLE/Central/Scan”.



**Fig. 8** Captura que muestra la ruta para cargar el ejemplo “Arduino BLE/Central/Scan”.

Una vez abierto el ejemplo, lo cargamos y revisamos el monitor serial.

Vemos que está escaneando dispositivos Bluetooth Low Energy.

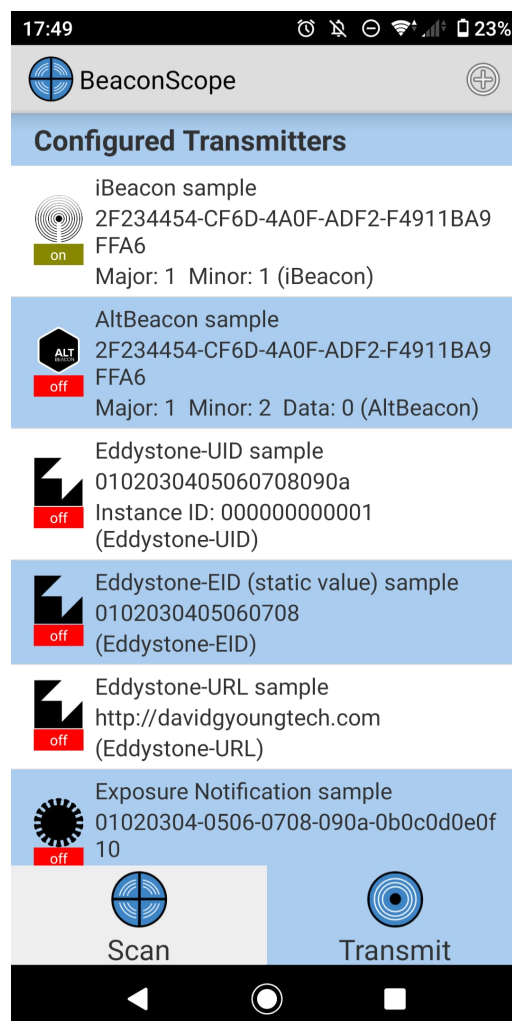


**Fig. 9** Captura que muestra la salida del monitor serial.

Pero no vemos ningún dispositivo, para ello, en nuestro teléfono deberemos aplicar una señal BLT, y para ello descargamos e instalamos la aplicación “Beacon Scope”.

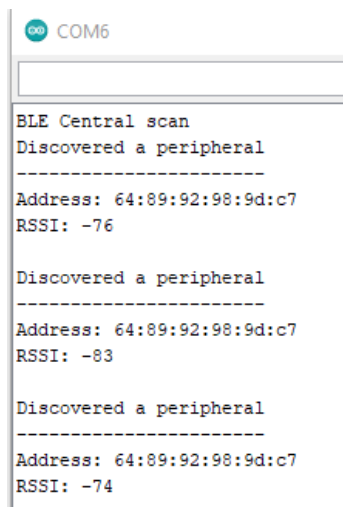
Una vez arrancada, vamos a “Transmit” y empezaremos a emitir una señal a 10 Hz a -60 de energía.

Para ello arrancaremos el ejemplo de “iBeacon Sample”, que es el primero de la lista.



**Fig. 10** Captura de la aplicación, mostrando la lista de señales BLE para emitir.

Ahora, si nos volvemos a fijar en el monitor serial, podremos ver la señal BLE detectada.



```
COM6

BLE Central scan
Discovered a peripheral
-----
Address: 64:89:92:98:9d:c7
RSSI: -76

Discovered a peripheral
-----
Address: 64:89:92:98:9d:c7
RSSI: -83

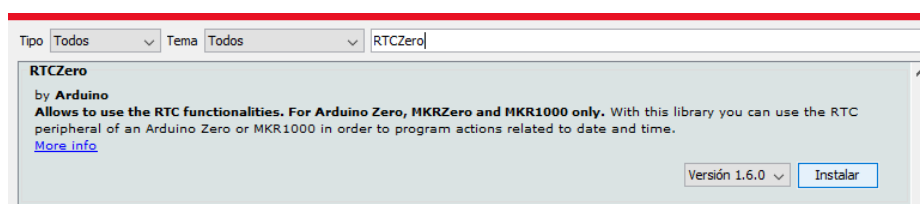
Discovered a peripheral
-----
Address: 64:89:92:98:9d:c7
RSSI: -74
```

**Fig. 11** Captura del monitor serial, mostrando la señal capturada.

#### 4 Librería RTCZero.

Podemos observar que la detección es casi instantánea, ya que por cada iteración se hace un barrido de la señales capturadas, para evitar que el escaneamiento sea instantáneo, usaremos la librería “RTC Zero” para añadir un contador de 5 segundos, y cada 5 segundos hacer un barrido.

Para ello, añadiremos la biblioteca RTCZero de la misma manera que añadiremos Arduino BLE, desde el gestor de librerías.



**Fig. 12** Captura del gestor de librerías, mostrando la librería a instalar.

Y una vez instalada, la importamos en el ejemplo de “Arduino BLE/Scan” actual.

```
// Import RTCZero and declare rtc objetc.
#include <RTCZero.h>
RTCZero rtc;

void setup() {
  Serial.begin(9600);
  while (!Serial);

  // Initialize rtc object.
  rtc.begin();
```



**Fig. 13** Captura del código, importando la librería RTC Zero y inicializando el objeto rtc en el setup().

Y en la función loop(), configuraremos el programa para que capture la señal cada 5 segundos.

```
void loop() {
  // Define BLE Device outside.
  BLEDevice peripheral;

  // Every 5 seconds, check BLE Devices...
  if (rtc.getSeconds() % 5 == 0) {
    // check if a peripheral has been discovered
    peripheral = BLE.available();
  }
}
```

**Fig. 14** Captura del código, configurando el barrido cada 5 segundos.

## 5 Cálculo de la distancia relativa al RSSI.

En este apartado, realizaremos una tabla en la cual por cada barrido nos alejaremos 25cm a lo largo de 2,50 metros, y compararemos las distancias con la intensidad de la señal, además de realizar un gráfico basado en los resultados.

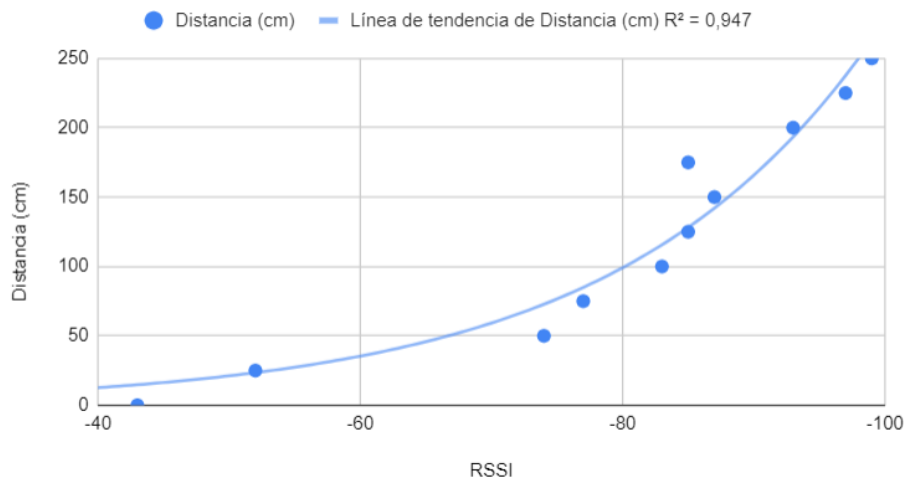
Como los resultados pueden variar debido a varios factores, tomaremos 11 muestras por cada distancia, y seleccionaremos la mediana de los valores.

Los resultados obtenidos son los siguientes:

Distancia (cm)	RSSI
0	-43
25	-52
50	-74
75	-77
100	-83
125	-85
150	-87
175	-85
200	-93
225	-97
250	-99

Los resultados representados en forma de grafo:

RSSI frente a Distancia (cm)



**Fig. 15** Captura del gráfico incluida la interpolación.

## 6 Funcionamiento combinado Wifi/BLE

En este apartado intentaremos combinar la funcionalidad del ejemplo WiFi NINA/WiFi Ping con la funcionalidad del ejemplo “Arduino BLE/Scan”, y ver el resultado.

Primero cargaremos el ejemplo de “WiFi NINA/WiFi Ping” desde el apartado de ejemplos como hemos hecho con el resto de ejemplos, y le añadiremos la funcionalidad de los LED para comprobar la conexión a Internet:

- LED on: inicio.
- LED off: conexión correcta.
- LED blink: conexión incorrecta.

Para ello, usaremos las funciones que definimos en la práctica 2 de parpadeo del LED.

Modificamos el código:

```
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // Setup leds
  pinMode(LED_BUILTIN, OUTPUT);

  // Turn led on on start.
  led_on();

  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }
}
```

Fig. 16 Captura del código, configurando los leds.

```
// attempt to connect to WiFi network:
while ( status != WL_CONNECTED) {
  Serial.print("Attempting to connect to WPA SSID: ");
  Serial.println(ssid);
  // Connect to WPA/WPA2 network:
  status = WiFi.begin(ssid, pass);

  // wait 5 seconds for connection:
  delay(5000);

  led_off();
  led_blink(100, 700);
}

// you're connected now, so print out the data:
Serial.println("You're connected to the network");
led_off();
printCurrentNet();
printWiFiData();
}
```

Fig. 17 Captura del código, configurando los leds.

```
void led_on() {
  digitalWrite(LED_BUILTIN, HIGH);
}

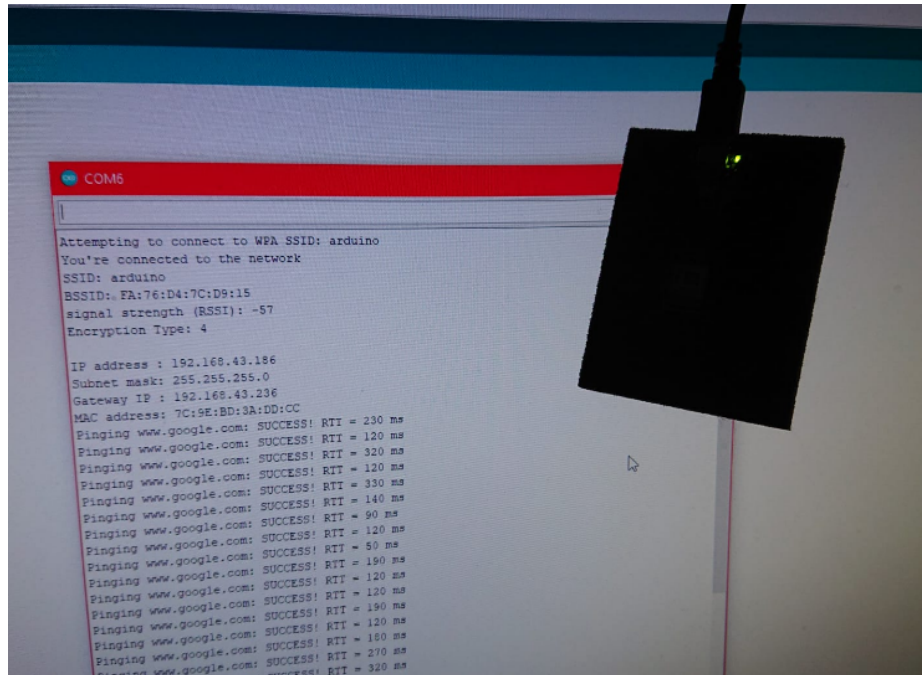
void led_off() {
  digitalWrite(LED_BUILTIN, LOW);
}

void led_blink(int del_time, int seq_time) {
  for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
      digitalWrite(LED_BUILTIN, HIGH);
      delay(del_time);
      digitalWrite(LED_BUILTIN, LOW);
      delay(del_time);
    }

    // Time delay between blink sequences.
    delay(seq_time);
  }
}
```

Fig. 18 Captura del código, configurando los leds.

Y lo probamos, podemos ver en la siguiente captura la salida del monitor serial, además de observar que el LED naranja está apagado, ambas salidas notifican que la conexión se ha establecido correctamente con la red.



**Fig. 19** Foto de la pantalla del monitor serial y el arduino, cuyo LED naranja está apagado, el único LED encendido es el de la corriente.

Ahora, reduciremos el código tanto del ejemplo de “Scan” como de “Ping” y lo combinaremos de tal manera que una vez que el dispositivo se ha conectado a internet, realice un barrido “scan” con BLE.

Para ello, lo que hemos hecho es primero arrancar WiFi NINA, y establecemos una conexión a la red, una vez que se ha conectado a la red, arrancaremos BLE y verificamos el estado de la conexión a internet.

```
void setup() {
  // The WiFi radio's status.
  int status = WL_IDLE_STATUS;

  // setup serial output
  Serial.begin(9600);
  while (!Serial) {
    // wait for serial port to connect. Needed for native USB port only
  }

  // attempt to connect to WiFi network:
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(SSID);

    // Connect to WPA/WPA2 network:
    status = WiFi.begin(SSID, PASSWORD);

    // wait 5 seconds for connection:
    delay(5000);
  }

  // Once connected, notify via serial monitor, and try BLE scanning.
  Serial.println("connected to the network");
  Serial.println("Starting BLE.");

  // begin initialization
  if (!BLE.begin()) {
    Serial.println("Starting BLE failed!");

    while (1);
  }

  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Disconnected from network!!");
  }

  // start scanning for peripheral
  Serial.println("Starting BLE Scanning...");
  BLE.scan();
}
```

Se queda  
atascado



**Fig. 20** Captura del código, mostrando en qué línea se queda atascado.

Si probamos el código, veremos que una vez que BLE toma control de la antena, nuestro arduino se quedará bloqueado si intentamos llamar a alguna otra función de WiFiNINA, esto es debido a que el dispositivo solo posee una antena, y en cuanto arrancamos BLE, este toma control de la antena, aunque el módulo WiFi haya tenido control sobre la antena de antes, pero al arrancar el módulo BLE, el módulo WiFi se queda sin control sobre la antena, y si intentamos llamar a alguna función relacionada con la librería WiFi NINA ( como en este caso “WiFi.status()” ) podremos observar que dichas funciones se han vuelto inutilizables.

## 7 Conclusiones.

En esta práctica hemos aprendido a usar los módulos BLE y WiFi en profundidad, además de aprender a como hacer un ping, y de darnos cuenta mediante pruebas que el dispositivo al solo tener una antena, solo un módulo puede acceder a ella a la vez.