

CUADERNO DE PRÁCTICAS

JORGE ALMINGOL ESTRADA

NIA: 684039

REDES DE SENSORES

MASTER EN INGENIERÍA ELECTRÓNICA

CURSO 2020-2021

Índice

1. Practica 1	3
1.1. Configuración y control de módulos hardware.....	3
1.2. Lectura del valor analógico cada 10s mediante el uso de la interrupción de timer.....	3
1.3. Genera una salida pwm a 5kHz proporcional a la lectura del ADC.	5
1.4. Comanda por la UART los periféricos	5
1.5. Conecta un sensor inercial por I2C (o SPI), muestrea la aceleración cada 100 ms y manda los datos cada segundo vía UART.....	7
2. Practica 2: Implementar el modelo anterior mediante tareas con el sistema operativo FreeRTOS.....	10
3. Practica 3. Comunicación, gestión y representación de datos de sensores con Python. ...	12
3.1. Programa en Python un programa que acceda al puerto serie y muestre por pantalla los datos en tiempo real.....	12
3.2. Modifica el programa para que almacene los datos en fichero .txt.....	12
3.3. Modifica el programa para que acumule los datos durante 5 segundos, calcule el promedio y desviación estándar los represente en tiempo real.	14
4. Comunicaciones WIFI y stack IP.....	17
4.1. Conéctate a la red wifi del laboratorio o a una creada por el móvil como punto de acceso. Extrae tu IP.	17
4.2. Pon en hora el módulo mediante un servidor NTP (Network Time Protocol)	19
4.3. Monta un chat una aplicación software PC	20
4.4. Después sustituye uno de los extremos por el módulo hardware siendo cliente y envía cada segundo tu hora local.	20
4.5. Añadir una capa de control de tal modo que cuando se le mande “start” empiece a mandar la hora hasta que se le mande “stop”.	20
4.6. Basándote en el estándar SENML crea un fichero JSON.....	22
4.7. Sube datos a la nube, en concreto al servicio gratuito proporcionado por Adafruit. .	24
5. Conexiones BLE y Bluetooth	27
5.1. Extrae temperatura, humedad y presión de los mensajes recibidos utilizando Python del sensor “RuuviTagSensor”.....	27
5.2. Haz un advertising con tu módulo siguiendo la identificación iBeacon	27
5.3. Establece un “chat” Bluetooth con perfil SPP con una APP en el móvil.....	28
6. Comunicaciones Lora y LoraWAN	30
6.1. Establece comunicaciones ping-pong con otros compañeros basada en broadcasts.	30
6.2. Envía cada 30 segundos un número incremental de 2 bytes con protocolo LoraWAN a un servidor en la nube.....	31

1. Practica 1

1.1. Configuración y control de módulos hardware

Lectura analógica de una tensión entre 0 y 3.3V

Introducimos las constantes necesarias

```
const int pin3v3 = 3.3;
const int lectorADC = 2;
float valorVoltiosAdc;
float valor_voltios = 0;
```

El setup se inicia la configuración del puerto serie

```
void setup() {
    Serial.begin(9600);
```

Por último se lee el valor del sensor del ADC y se adapta a voltios, mostrándolo por pantalla ya dentro del loop

```
    contador=contador+1;
    valorVoltiosAdc = analogRead(lectorADC); /* (5/4095);
    valor_voltios = valorVoltiosAdc * 0.000805;
    Serial.print("Medida del sensor numero ");
    Serial.println(contador);
    Serial.print("sensor value en 0-4096 = ");
    Serial.println(valorVoltiosAdc);
    Serial.print("sensor value en Voltios = ");
    Serial.println(valor_voltios);
```

1.2. Lectura del valor analógico cada 10s mediante el uso de la interrupción de timer

En primer lugar declaramos el timer y su función correspondiente. Esta función pone un flag a 1 cada vez que se ejecuta

```
//declaracion timer
volatile int interruptCounter;
hw_timer_t* timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR onTimer(){
    portENTER_CRITICAL_ISR(&timerMux);
    interruptCounter++;
    portEXIT_CRITICAL_ISR(&timerMux);
}
```

Dentro del setup declaramos las características del timer

```

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

  //timer configuration
  timer = timerBegin (0, 80, true);
  //80MHz dividido entre 80, dejando 1MHZ de frecuencia y por tanto un timer de microsegundos
  timerAttachInterrupt(timer, &onTimer, true);
  //declara el timer de la interrupcion
  timerAlarmWrite(timer, 10000000, true);
  //indica momento alarma del timer, (vector de inicializacion, momento de alarma y true para reiniciar)
  timerAlarmEnable(timer);

```

Y por último en el loop, cuando detecte que la interrupción ha activado el flag, lo pondrá a 0 y realizara la lectura del sensor

```

void loop() {

  if(interruptCounter > 0){
    portENTER_CRITICAL_ISR(&timerMux);
    interruptCounter--;
    portEXIT_CRITICAL_ISR(&timerMux);

    contador=contador+1;
    valorVoltiosAdc = analogRead(lectorADC);/*(5/4095);
    valor_voltios = valorVoltiosAdc * 0.000805;
    Serial.print("Medida del sensor numero ");
    Serial.println(contador);
    Serial.print("sensor value en 0-4096 = ");
    Serial.println(valorVoltiosAdc);
    Serial.print("sensor value en Voltios = ");
    Serial.println(valor_voltios);
    Serial.println("");
    Serial.println("");

    //delay (5000);

  }

```

1.3. Genera una salida pwm a 5kHz proporcional a la lectura del ADC.

Declaramos las variables necesarias para el funcionamiento del programa

```
const int pin3v3 = 3.3;
const int lectorADC = 2;
int valorVoltiosAdc;
float valor_voltios = 0;
int contador = 0;

const int freq = 5000;
const int resolution = 12;
int signalVoltios;
int pinPWM = 18; // Poner pin correspondiente al PWM
int PWMchannel = 0;
```

Después ajustamos dentro del setup la frecuencia del PWM a los 5khz requeridos y la resolución, puesto que la del adc es de 12, le pondremos 12.

Una vez declarado le asignamos un pin de salida. PWMchannel sería el pwm configurado y el pin asignado seria pinPWM.

```
ledcSetup(PWMchannel, freq, resolution);

ledcAttachPin(pinPWM, PWMchannel);
```

Ya dentro del loop, asignamos el valor de entrada del PWM el de salida del ADC. Y para terminar se mostrará por pantalla como hemos hecho en programas anteriores

```
void loop() {

valorVoltiosAdc = analogRead(lectorADC);
ledcWrite(PWMchannel, valorVoltiosAdc);
```

1.4. Comanda por la UART los periféricos

- ADC: Envíe la lectura del ADC actual
- ADC(x): envíe la lectura del ADC cada x segundos. Si x=0, deja de mandar datos
- PWM(x): comanda el duty cycle del módulo PWM con números del 0 al 9.

La adquisición de los datos y el control de la salida del PWM se realizan igual que en el programa anterior. En este se desarrolla el control por UART.

Ya dentro del loop, en primer lugar comprobamos si hemos recibido un mensaje

```
//comprobamos si se ha recibido un mensaje
if (Serial.available() > 0) {
  //leemos
  option = Serial.readStringUntil('\n');
  Serial.println(option);
}
```

Si el mensaje es ADC envía el valor del ADC

```
if (option == "ADC") {
  Serial.println("ValorVoltiosAdc = ");
  Serial.print(ValorVoltiosAdc);
}
```

Si es ADC(x) envía el valor cada x segundos

```
if (option == "ADC(0)") {
  lectorADC = 0;
}
if (option == "ADC(1)") {
  lectorADC = 1;
  tiempoADC = 1;
}
if (option == "ADC(2)") {
  lectorADC = 1;
  tiempoADC = 2;
}
if (option == "ADC(3)") {
  lectorADC = 1;
  tiempoADC = 3;
}
if (option == "ADC(4)") {
  lectorADC = 1;
  tiempoADC = 4;
}
if (option == "ADC(5)") {
  lectorADC = 1;
  tiempoADC = 5;
}
if (option == "ADC(6)") {
  lectorADC = 1;
  tiempoADC = 6;
}
if (option == "ADC(7)") {
  lectorADC = 1;
  tiempoADC = 7;
}
if (option == "ADC(8)") {
  lectorADC = 1;
  tiempoADC = 8;
}
if (option == "ADC(9)") {
  lectorADC = 1;
  tiempoADC = 9;
}
```

Contamos los ciclos correspondientes y enviamos el mensaje (Si ADC(0) dejamos de enviar)

```
if (lectorADC == 1) {  
    if (contadorADC == tiempoADC){  
        Serial.print("ValorVoltiosAdc = ");  
        Serial.println(ValorVoltiosAdc);  
        contadorADC = 0;  
    }  
}  
}
```

Para el PWM hacemos algo similar

```
if (option == "PWM(0)") {  
    Dutty = 0;  
}  
if (option == "PWM(1)") {  
    Dutty = 0.1;  
}  
if (option == "PWM(2)") {  
    Dutty = 0.2;  
}  
if (option == "PWM(3)") {  
    Dutty = 0.3;  
}  
if (option == "PWM(4)") {  
    Dutty = 0.4;  
}  
if (option == "PWM(5)") {  
    Dutty = 0.5;  
}  
if (option == "PWM(6)") {  
    Dutty = 0.6;  
}  
if (option == "PWM(7)") {  
    Dutty = 0.7;  
}  
if (option == "PWM(8)") {  
    Dutty = 0.8;  
}  
if (option == "PWM(9)") {  
    Dutty = 0.9;  
}  
  
}  
salidaPWM = Dutty * 255;  
ledcWrite (pinPWM, salidaPWM);
```

1.5. Conecta un sensor inercial por I2C (o SPI), muestrea la aceleración cada 100 ms y manda los datos cada segundo vía UART

Definimos los datos de configuración

```

#define MPU9250_ADDRESS      0x68
#define MAG_ADDRESS          0x0C

#define GYRO_FULL_SCALE_250_DPS  0x00
#define GYRO_FULL_SCALE_500_DPS  0x08
#define GYRO_FULL_SCALE_1000_DPS 0x10
#define GYRO_FULL_SCALE_2000_DPS 0x18

#define ACC_FULL_SCALE_2_G  0x00
#define ACC_FULL_SCALE_4_G  0x08
#define ACC_FULL_SCALE_8_G  0x10
#define ACC_FULL_SCALE_16_G 0x18

```

Las funciones de lectura y escritura de I2C

```

//Funcion auxiliar lectura
void I2Cread(uint8_t Address, uint8_t Register, uint8_t Nbytes, uint8_t* Data)
{
    Wire.beginTransmission(Address);
    Wire.write(Register);
    Wire.endTransmission();

    Wire.requestFrom(Address, Nbytes);
    uint8_t index = 0;
    while (Wire.available())
        Data[index++] = Wire.read();
}

// Funcion auxiliar de escritura
void I2CwriteByte(uint8_t Address, uint8_t Register, uint8_t Data)
{
    Wire.beginTransmission(Address);
    Wire.write(Register);
    Wire.write(Data);
    Wire.endTransmission();
}

```

Dentro del setup configuramos el sensor de aceleración y de giro y sus rangos de trabajo

```

// Configurar acelerometro
I2CwriteByte(MPU9250_ADDRESS, 28, ACC_FULL_SCALE_16_G);

// Configurar giroscopio
I2CwriteByte(MPU9250_ADDRESS, 27, GYRO_FULL_SCALE_2000_DPS);

```

Y ya dentro del loop, controlado por la interrupción de timer leemos los datos


```

// --- Lectura acelerometro y giroscopio ---
uint8_t Buf[14];
I2Cread(MPU9250_ADDRESS, 0x3B, 14, Buf);

// Convertir registros acelerometro
int16_t ax = -(Buf[0] << 8 | Buf[1]);
int16_t ay = -(Buf[2] << 8 | Buf[3]);
int16_t az = Buf[4] << 8 | Buf[5];

// Convertir registros giroscopio
int16_t gx = -(Buf[8] << 8 | Buf[9]);
int16_t gy = -(Buf[10] << 8 | Buf[11]);
int16_t gz = Buf[12] << 8 | Buf[13];

```

Para capturar en tiempo tenemos la interrupción temporal configurada en 100ms. Captura los datos y los guarda en un vector de tamaño 10, una vez lleno lo muestra. Los dos primeros ciclos de llenado encienden un led.

```

if (contadorEnvio == 10) {
    LED = 0;
    contador = contador + 1;
    //Serial.print("Medida del sensor numero ");
    //Serial.println(contador);
    //Serial.println("sensor value = ");
    Serial.println(datos[0], DEC);
    Serial.println(datos[1], DEC);
    Serial.println(datos[2], DEC);
    Serial.println(datos[3], DEC);
    Serial.println(datos[4], DEC);
    Serial.println(datos[5], DEC);
    Serial.println(datos[6], DEC);
    Serial.println(datos[7], DEC);
    Serial.println(datos[8], DEC);
    Serial.println(datos[9], DEC);
    //    Serial.println("");
    //    Serial.println("");
    contadorEnvio = 0;
}

if (LED < 2) {
    digitalWrite (LEDpin, LOW);
} else {
    digitalWrite (LEDpin, HIGH);
}

```

2. Practica 2: Implementar el modelo anterior mediante tareas con el sistema operativo FreeRTOS.

Dentro del Setup creamos las tareas, en este caso se asigna a ambas la misma prioridad, pero en caso contrario la que tenga el número más alto será la más prioritaria, (la prioridad es el 5 elemento de la declaración, en ambas es 1)

```
xTaskCreate(Tareal, "Tareal", 10000, NULL, 1, NULL);
xTaskCreate(Tarea2, "Tarea2", 10000, NULL, 1, NULL);
```

Ya luego introducimos el código que queremos que realice cada tarea. La tarea 1 se encargara de recoger datos del sensor y la tarea 2 los mostrara por pantalla y encenderá el led

```
void Tareal( void * parameter ) {
    while(1){
        if (interruptCounter > 0) {
            portENTER_CRITICAL_ISR(&timerMux);
            interruptCounter--;
            portEXIT_CRITICAL_ISR(&timerMux);

            contadorEnvio++;
            LED++;

            // --- Lectura acelerometro y giroscopio ---
            uint8_t Buf[14];
            I2Cread(MPU9250_ADDRESS, 0x3B, 14, Buf);

            // Convertir registros acelerometro
            int16_t ax = -(Buf[0] << 8 | Buf[1]);
            int16_t ay = -(Buf[2] << 8 | Buf[3]);
            int16_t az = Buf[4] << 8 | Buf[5];

            // Convertir registros giroscopio
            int16_t gx = -(Buf[8] << 8 | Buf[9]);
            int16_t gy = -(Buf[10] << 8 | Buf[11]);
            int16_t gz = Buf[12] << 8 | Buf[13];

            //Serial.println(ax, DEC);
            datos[contadorEnvio] = ax ;
            contadorExtra = 0;

            delay(1);
        }
    }
}
```

Esta tarea usa la interrupción aunque realmente no es del todo necesario. En FreeRTOS lo interesante es utilizar los delay y confiar en la gestión temporal del propio sistema operativo. La tarea dos encargada del envío de los datos:

```

void Tarea2( void * parameter) {
    while (1){
        if (contadorEnvio == 10) {
            LED = 0;
            contador = contador + 1;
            //Serial.print("Medida del sensor numero ");
            //Serial.println(contador);
            //Serial.println("sensor value = ");
            Serial.println(datos[0], DEC);
            Serial.println(datos[1], DEC);
            Serial.println(datos[2], DEC);
            Serial.println(datos[3], DEC);
            Serial.println(datos[4], DEC);
            Serial.println(datos[5], DEC);
            Serial.println(datos[6], DEC);
            Serial.println(datos[7], DEC);
            Serial.println(datos[8], DEC);
            Serial.println(datos[9], DEC);
            //    Serial.println("");
            //    Serial.println("");
            contadorEnvio = 0;
            contadorExtra = 0;
        }

        if (LED < 2) {
            digitalWrite (LEDpin, LOW);

        } else {
            digitalWrite (LEDpin, HIGH);
        }
        delay(1);
    }

    Serial.println("Finalizando tarea 2");
    vTaskDelete( NULL );
}

```

3. Practica 3. Comunicación, gestión y representación de datos de sensores con Python.

3.1. Programa en Python un programa que acceda al puerto serie y muestre por pantalla los datos en tiempo real

3.2. Modifica el programa para que almacene los datos en fichero .txt

Se van a realizar ambas a la vez

Como indica el enunciado instalamos pyserial y procedemos a desarrollar el código. Para ello hacemos del entorno anaconda y el programa spyder 3, que trabaja con python 3.5. El código es el siguiente:

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6
7 https://www.luisllamas.es/controlar-arduino-con-python-y-la-libreria-pyserial/
8 """
9 import serial
10 import time
11 |
12 i=1
13
14 #Serial takes two parameters: serial device and baudrate
15 arduino = serial.Serial('COM3', 9600) #abrimos comunicación con el arduino
16 time.sleep(2)
17 while i <= 3:
18     data = arduino.readline().strip()
19     # data_decode = data.decode("base64")
20     print(str(data))
21     archivo_1 = open("datos_prueba_arduino.txt", "a") #ponemos a en lugar de w para añadir, no para escribir encima
22     archivo_1.write(str(data))
23     archivo_1.write(";\n")
24     archivo_1.close()
25 arduino.close() #cerramos comunicaciones con el arduino
26
```

Creamos una comunicación serie llamada arduino con las características de comunicación del programa de arduino.

Una vez ya en el bucle leemos los datos del puerto serie, abrimos un documento de texto los guardamos incluyendo las terminaciones de punto y coma y retorno de carro para leerlas desde Excel.

Además la consola muestra los datos por pantalla.

```
IPython console
Console 1/A
b'-407'
b'-400'
b'-403'
b'-406'
b'-400'
b'-403'
b'-399'
b'-404'
b'-403'
b'-405'
b'-402'
b'-412'
b'-403'
b'-396'
b'-403'
b'-406'
b'-404'
b'-395'
b'-399'
b'-396'
b'-393'
b'-408'
b'-404'
b'-407'
b'-391'
b'-407'
b'-405'
b'-403'
b'-401'
b'-404'
b'-399'
b'-395'
b'-405'
b'-399'
b'-399'
b'-401'
b'-401'
b'-394'
b'-399'
b'-402'
b'-402'
b'-405'
b'-399'
```

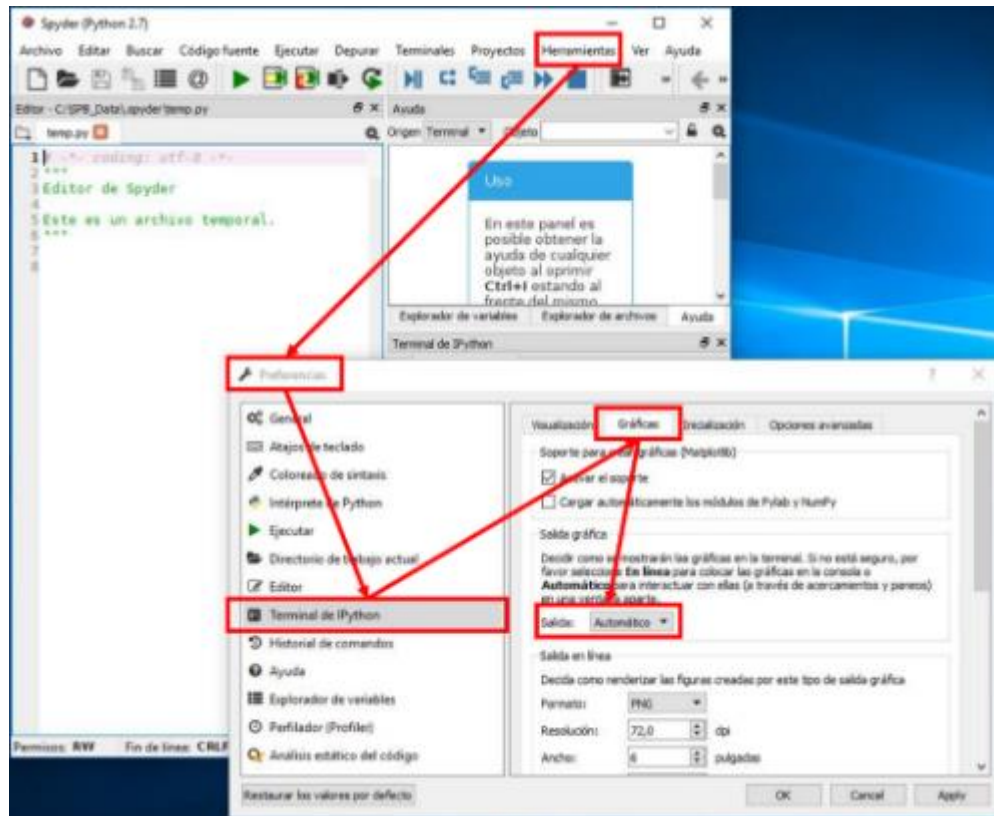
El txt. queda también bien realizado con los datos correctamente guardados.

```
datos_prueba_...
Archivo Edición Formato Ver Ayuda
-264;
-134;
-7;
114;
180;
128;
12;
-219;
-387;
-427;
-406;
-461;
-416;
-258;
-85;
76;
203;
152;
-63;
-433;
-1085;
-309;
-210;
-62;
133;
218;
70 Windows (CRLF) UTF-8
```

Solo se realizó con un dato, pero en caso de usar más la solución sería añadir solo punto y coma y cuando se reciban los 6 datos incluir el final de línea

3.3. Modifica el programa para que acumule los datos durante 5 segundos, calcule el promedio y desviación estándar los represente en tiempo real.

Realizamos la configuración recomendada en el enunciado



Y modificamos el código anterior para que genere una gráfica.

Declaramos las gráficas

```
24
25
26 plt.figure()
27 plot1 = plt.subplot(2,2,1)
28 plot2 = plt.subplot(2,2,2)
29 plot3 = plt.subplot(2,2,3)
30
```

Iniciamos comunicación con el micro

```
32
33 try:
34     #Serial takes two parameters: serial device and baudrate
35     arduino = serial.Serial('COM3', 9600) #abrimos comunicación con el arduino
36 except serial.SerialException:
37     print("No Conectado")
38     Conectado = False
39
40
```

Y posteriormente, comunicamos con él, leemos el dato, ponemos un modo de saltarnos un posible error de decodificación y por ultimo modificamos los datos para que cumplan los requerimientos necesarios.

```
4
3 while Conectado:
4     try:
5         try:
6             data = arduino.readline().decode("ascii")
7         except UnicodeDecodeError:
8             print("")
9
10        # data_decode = data.decode("base64")
11        data = data.replace("\n", "")
12        data = data.replace("\r", "") |
13        data = data.replace("\\", "")
14        data = data.rstrip('\n')
15        temp = data.split(',')
16
17        if r == 1:
18
19            datos[i] = int(data)
20            i=i+1
21        else:
22            r=1
23
24        print(data)
25        plot1.plot(contador1 / 10 , int(data) , 'o', c='r')
26
27
28        archivo_1 = open("datos_prueba_arduino.txt", "a") #ponemos a en lugar de w para añadir, no para escribir encima
29        archivo_1.write(data)
30        archivo_1.write(";\n")
31        archivo_1.close()
32
33        contador1=contador1+1
34
```

Una vez hecho esto guarda los datos en un array de tamaño 50 y, individualmente los guarda en un fichero de texto y los muestra en plot1.

Después calcula la media y la varianza de los datos recogidos en el array y los muestra en otras 2 gráficas

```
if i == 50:
    i=0

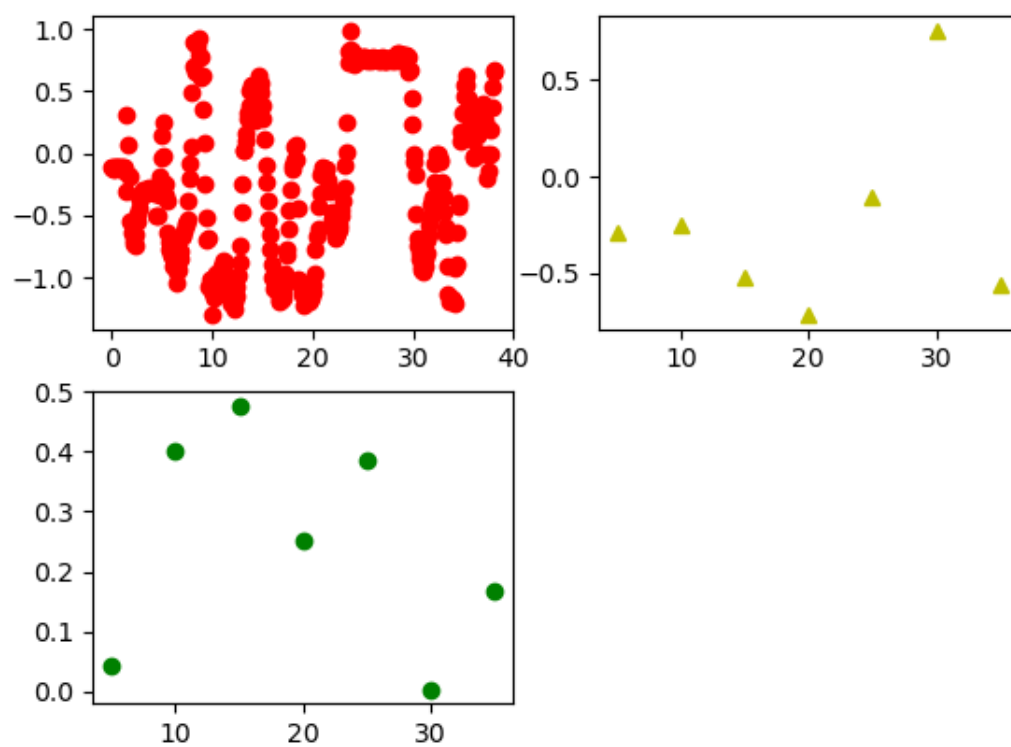
    media = datos.mean()
    print ("esta es la media =")
    print (media)
    plot2.plot(contador2, media, '^-', c='y')

    varianza = datos.var()
    plot3.plot(contador2, varianza, 'o-', c='g')

    desviacionTipica = datos.std()
    #plot3.plot(contador2, desviacionTipica, marker = 'x', c='b')

    contador2=contador2+5
    plt.show()
```

LA grafica queda de la siguiente manera mostrando las aceleraciones del eje x medidas cada 100ms y la media y varianza tomadas cada 50 datos.



4. Comunicaciones WIFI y stack IP

4.1. Conéctate a la red WIFI del laboratorio o a una creada por el móvil como punto de acceso. Extrae tu IP.

Incluimos las librerías correspondientes y añadimos la SSID y la PASS de la red WiFi a la que nos queremos conectar

```
#include <Wire.h>
#include <WiFi.h>

const char* ssid = "Jorge_Almin";
const char* password = "iotdejorge";
```

Dentro del Setup indicamos los datos de la red a la que nos queremos conectar y esperamos a que se conecte

```
void setup() {
  Serial.begin(9600);
  delay(10);

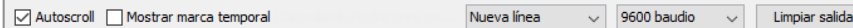
  //conecta a la red wify
  Serial.println();
  Serial.print("Conectando con ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    Serial.print(WiFi.status());
  }
  Serial.println("");
  Serial.println("Conectado con wify.");
  //Serial.println(WiFi.status());
```

Posteriormente mostramos por pantalla los datos requeridos mediante las siguientes peticiones entre las que se incluye mostrar la conexión IP, la máscara de red, la puerta de enlace, los datos de la red, la mac de la tarjeta e incluso os datos de la potencia de la señal.

El resultado:



4.2. Pon en hora el módulo mediante un servidor NTP (Network Time Protocol)

Para la conexión a la red WIFI usamos lo visto en el modelo anterior. En este caso añadimos la conexión a un servidor que proporciona la hora y lo configuramos a la hora de nuestro país teniendo en cuenta el cambio horario

```
const char* ntpServer = "pool.ntp.org";
const long  gmtoffset_sec = 3600;    //si es gtm+1 3600, si es +2 7200 etc...
const int   daylightOffset_sec = 0; //3600; //verano 3600, invierno 0.
```

También añadimos una función encargada de mostrar correctamente la hora:

```
void printLocalTime()
{
    struct tm timeinfo;
    if(!getLocalTime(&timeinfo)){
        Serial.println("Failed to obtain time");
        return;
    }
    Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
}
```

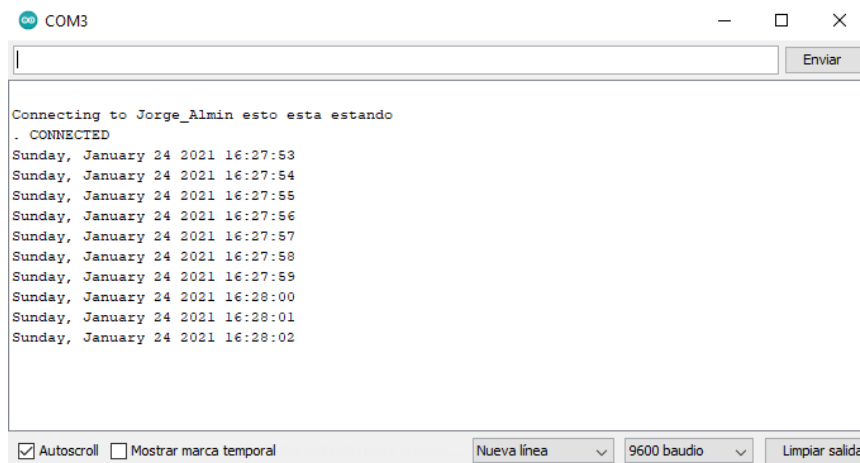
Dentro del setup, además de iniciar la comunicación con la red WIFI iniciamos la comunicación con el servidor ntp.

```
//init and get the time
configTime(gmtoffset_sec, daylightOffset_sec, ntpServer);
printLocalTime();
```

Y ya dentro del loop demandamos la hora cada segundo

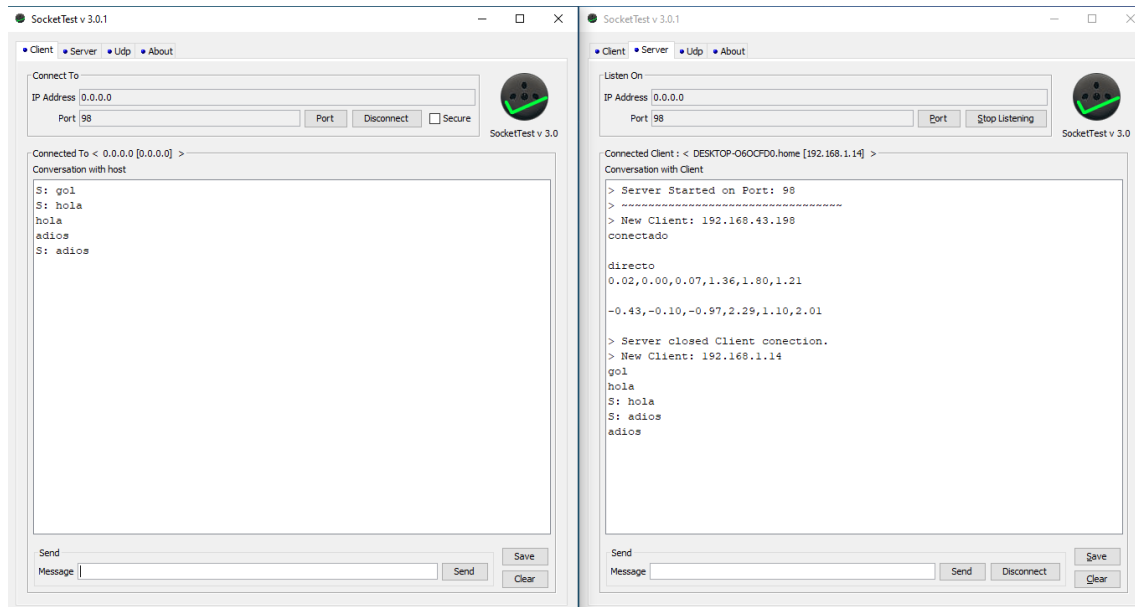
```
void loop()
{
    delay(1000);
    printLocalTime();
}
```

Con el siguiente resultado:



4.3. Monta un chat una aplicación software PC

Mediante la aplicación SocketTest creamos un chat, para ello ejecutamos 2 ventanas, una será cliente y el otro servidor. Para ello, puesto que están dentro del pc solo es necesario configurar los dos con la misma dirección IP y en el mismo puerto.



4.4. Después sustituye uno de los extremos por el módulo hardware siendo cliente y envía cada segundo tu hora local.

4.5. Añadir una capa de control de tal modo que cuando se le mande “start” empiece a mandar la hora hasta que se le mande “stop”.

Realizamos estas dos funciones en un mismo código. La configuración de la puesta en hora y la conexión WIFI la omitimos debido a que ya se ha explicado anteriormente. En primer lugar para conectarnos a un servidor de SocketTest introducimos la ip estática v4 del ordenador en el programa y creamos un cliente.

```
int status = WL_IDLE_STATUS;
IPAddress server(192, 168, 1, 14);
// Initialize the client library
WiFiClient client;
```

Una vez declarados estos datos nos conectamos con el servidor añadiendo la ip del servidor y el puerto de conexión dentro del setup:

```
if (client.connect(server, 98)) {
  Serial.println("connected");
  // Make a HTTP request:
  client.println("conectado");
  client.println();
}
```

De esta forma ya podríamos enviar los datos al servidor pero además vamos a añadir el código completo con las peticiones de start stop.

Ya en el loop mediante una interrupción gestionamos el tiempo. Además en cada iteración evaluamos si hay algún mensaje pendiente que provenga del servidor. Se realiza igual que con las comunicaciones de puerto Serial.

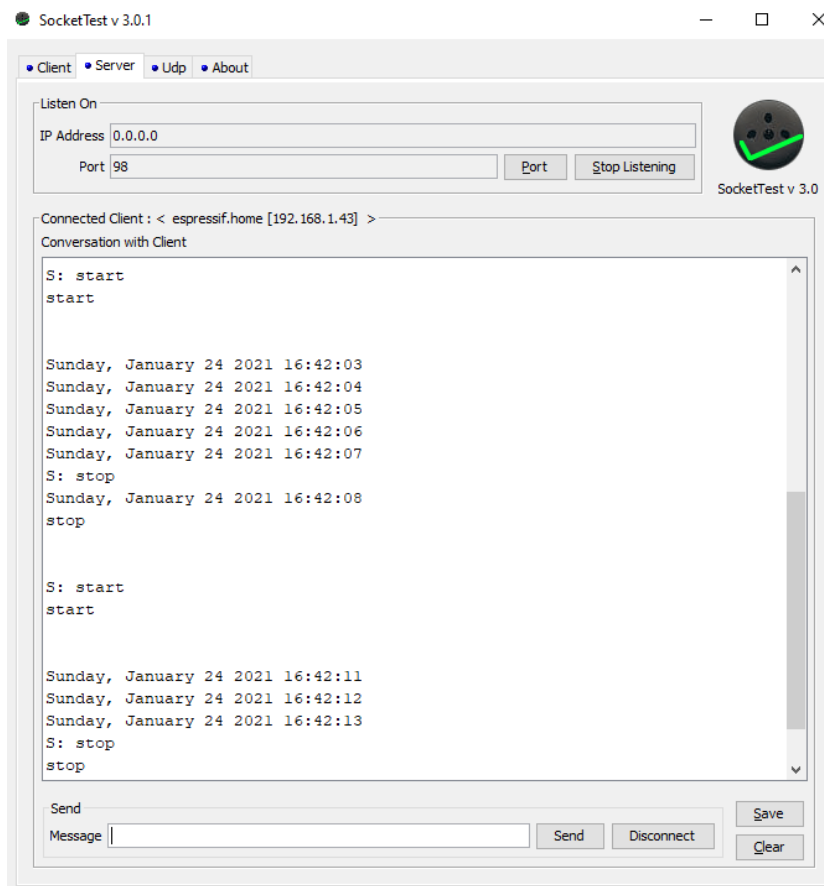
```
if (client.available() > 0) {  
    //leemos  
    option = String(client.readStringUntil('\r'));  
    client.println(option);  
    Serial.println(option);  
}
```

En caso de haber mensaje se comprueba si coincide con los comandos start o stop.

```
if (option == String("start")) {  
    startCount = 1;  
    Serial.println("entro al primer bucle");  
}  
if (option == String("stop")) {  
    startCount = 0;  
}
```

En caso de que el flag valga uno el programa mostrara la hora y, en caso de valer 0 no lo hará

```
if (startCount == 1) {  
    printLocalTime();  
}
```



4.6. Basándote en el estándar SENML crea un fichero JSON

Incluimos la biblioteca de JSON y creamos un documento y un array asignando memoria al documento. En este caso 500.

```
#include <ArduinoJson.h>

DynamicJsonDocument doc(500);
JsonArray data = doc.createNestedArray("data");
```

JSON permite añadir datos etiquetados. En este caso haremos uso de 3 etiquetas: Sensor, time y data (el array)

En el setup inicializamos estos grupos y añadimos los primeros datos

```
doc["sensor"] = "gps";
doc["time"] = 1351824120;

// Add an array.
//
//JsonArray data = doc.createNestedArray("data");
data.add(48.756080);
data.add(2.302038);
data.add(1.345);
```

Ya en el loop añadimos más datos hasta completar la memoria RAM dedicada al archivo.

```
void loop() {

  JsonArray data = doc.createNestedArray("data");
  data.add(random(10));
  data.add(random(10));
  data.add(random(10));
  data.add(random(10));
  data.add(random(10));
  data.add(random(10));
  data.add(random(10));
  data.add(random(10));

  Serial.println();
  serializeJson(doc, Serial);
  doc.remove(data);
  delay(1000);
}
```

Aquí se comprueba cómo una vez lleno el documento deja de añadir datos

```
COM3

{"data": [
  48.75608,
  2.302038,
  1.345
],
"sensor": "gps",
"time": 1351824120
}
{"data": [3,9,8,9,8,3,4],"sensor": "gps", "time": 1351824120}
{"data": [2,5,6,6,1,7,8],"sensor": "gps", "time": 1351824120}
{"data": [3,7,9,5,3,1,7],"sensor": "gps", "time": 1351824120}
{"data": [2,3,1,3],"sensor": "gps", "time": 1351824120}
{"data": [], "sensor": "gps", "time": 1351824120}
{"data": [], "sensor": "gps", "time": 1351824120}
```

Continuamos añadiendo la comunicación ftp para enviar el documento al servidor.

Nos conectamos a la red WIFI como en anteriores programas e incluimos esta vez la ip del servidor ftp más los datos de usuario y contraseña

```
char ftp_server[] = "155.210.150.77";
char ftp_user[]   = "rsense";
char ftp_pass[]   = "rsense";

ESP32_FTPClient ftp (ftp_server, ftp_user, ftp_pass);
```

En el setup abrimos la comunicación con el servidor, abrimos el directorio de trabajo, creamos el documento y escribimos en el los datos.

```
ftp.OpenConnection();

//Change directory
ftp.ChangeWorkDir("/rsense/JAlmingol");

// Create a new file to use as the download example below:
ftp.InitFile("Type A");
ftp.NewFile("JAlmDatta.txt");
ftp.Write(Datos);
ftp.CloseFile();
```

También podemos leer los datos del servidor

```
//Download the text file or read it
String response = "";
ftp.InitFile("Type A");
ftp.DownloadString("JAlmDatta.txt", response);
Serial.println("The file content is: " + response);
ftp.CloseConnection();
```

4.7. Sube datos a la nube, en concreto al servicio gratuito proporcionado por Adafruit.

Creamos una cuenta en Adafruit y generamos una aplicación obteniendo un "AIO Key". Y Usando MQTT enviamos datos al servidor.

Para ello necesitamos adjuntar un archivo de configuración config.h con nuestras credenciales y el resto de datos necesarios como la red WIFI.

```
/****** Configuration *****/  
  
// edit the config.h tab and enter your Adafruit IO credentials  
// and any additional configuration needed for WiFi, cellular,  
// or ethernet clients.  
#include "config.h"
```

Se va a realizar un contador que se incremente constantemente y mande el dato a Adafruit

```
// this int will hold the current count for our sketch  
int count = 0;  
  
// set up the 'counter' feed  
AdafruitIO_Feed *counter = io.feed("Datos_esp");
```

Dentro del Setup nos conectamos a Adafruit

```
void setup() {  
  
  // start the serial connection  
  Serial.begin(9600);  
  
  // wait for serial monitor to open  
  while(! Serial);  
  
  Serial.print("Connecting to Adafruit IO");  
  
  // connect to io.adafruit.com  
  io.connect();  
  
  // wait for a connection  
  while(io.status() < AIO_CONNECTED) {  
    Serial.print(".");  
    delay(500);  
  }  
  
  // we are connected  
  Serial.println();  
  Serial.println(io.statusText());  
  
}
```

Y ya en el loop enviamos los datos incrementales cada 3 segundos. Para ello iniciamos la comunicación mediante run y guardamos el dato en cada iteración


```

void loop() {

    // io.run(); is required for all sketches.
    // it should always be present at the top of your loop
    // function. it keeps the client connected to
    // io.adafruit.com, and processes any incoming data.
    io.run();

    // save count to the 'counter' feed on Adafruit IO
    Serial.print("sending -> ");
    Serial.println(count);
    counter->save(count);

    // increment the count by 1
    count++;

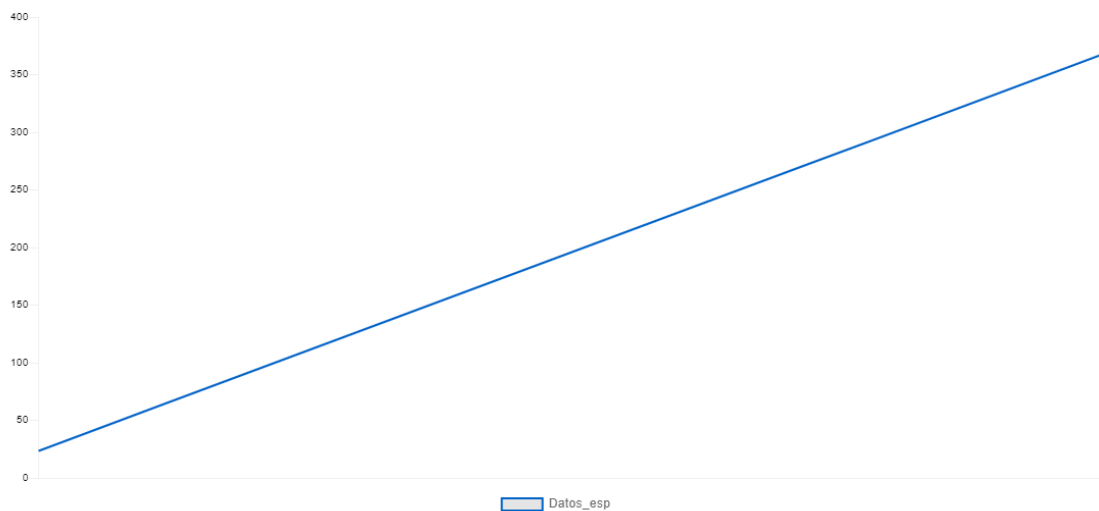
    // Adafruit IO is rate limited for publishing, so a delay is required in
    // between feed->save events. In this example, we will wait three seconds
    // (1000 milliseconds == 1 second) during each loop.
    delay(3000);

}

```

Los datos en la web de AdaFruit

JAlmin > Feeds > Datos_esp



Para subir los códigos el mecanismo es el mismo cambiando en el setup el código para que lea datos

```

// Because Adafruit IO doesn't support the MQTT retain flag, we can use the
// get() function to ask IO to resend the last value for this feed to just
// this MQTT client after the io client is connected.
counter->get();

// we are connected
Serial.println();
Serial.println(io.statusText());

```

Y leyendo mediante el loop cualquier dato que se actualice en el servidor

```
void loop() {  
  
  // io.run(); is required for all sketches.  
  // it should always be present at the top of your loop  
  // function. it keeps the client connected to  
  // io.adafruit.com, and processes any incoming data.  
  io.run();  
  
  // Because this sketch isn't publishing, we don't need  
  // a delay() in the main program loop.  
  
}  
  
// this function is called whenever a 'counter' message  
// is received from Adafruit IO. it was attached to  
// the counter feed in the setup() function above.  
void handleMessage(AdafruitIO_Data *data) {  
  
  Serial.print("received <- ");  
  Serial.println(data->value());  
  
}
```

5. Conexiones BLE y Bluetooth

5.1. Extrae temperatura, humedad y presión de los mensajes recibidos utilizando Python del sensor “RuuviTagSensor”

```
7 from ruuvitag_sensor.ruuvitag import RuuviTag
8
9 #sensor = RuuviTag('AA:2C:6A:1E:59:3D')
10 sensor = RuuviTag('C4:4F:33:65:D2:4B')
11
12 # update state from the device
13 state = sensor.update()
14
15 # get latest state (does not get it from the device)
16 state = sensor.state
17
18 print(state)
```

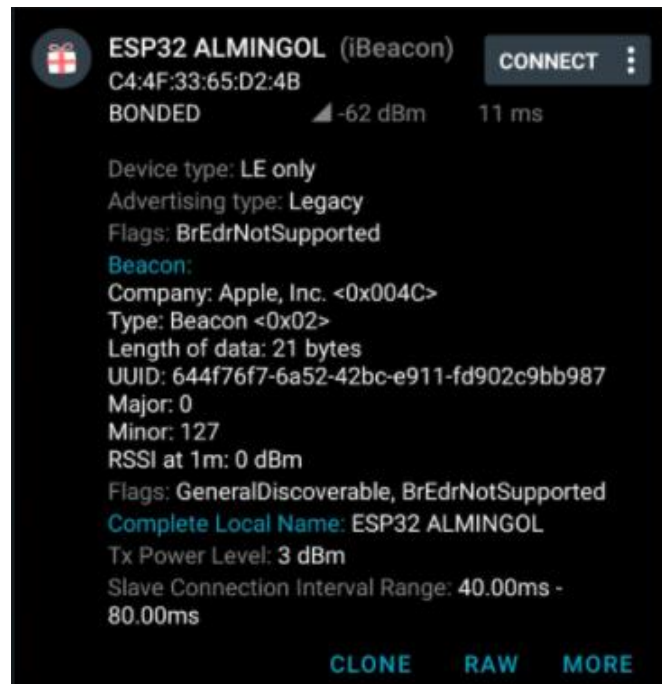
Obtenemos como resultado lo siguiente:

```
In [2]: runfile('J:/Master/RDSensores/
repositorio_practicas/Practica_4/prueba_BLE.py', wdir='J:/
Master/RDSensores/repositorio_practicas/Practica_4')
DATA TYPE 2 IS OBSOLETE. UPDATE YOUR TAG
{'pressure': 995.0, 'temperature': 24.0, 'humidity': 30.0,
'data_format': 2, 'identifier': None}
```

5.2. Haz un advertising con tu módulo siguiendo la identificación iBeacon

Mediante los ejemplos de esp32BLE obtenemos el código que nos permite generar un anuncio. Con el siguiente resultado.

```
Advertisizing started...
enter deep sleep
KR$H"$D6$zbQRDa$Sr$??@start ESP32 131
deep sleep (1449s since last reset, 11s since last boot)
Advertisizing started...
enter deep sleep
KR$D"$9$G$RDT$Sb$!start ESP32 132
deep sleep (1460s since last reset, 11s since last boot)
Advertisizing started...
enter deep sleep
KR$D$wD$B$))$E$D$UQ$C$start ESP32 133
deep sleep (1471s since last reset, 11s since last boot)
Advertisizing started...
enter deep sleep
KR$D$D$y!vSOSDp$er#B$start ESP32 134
deep sleep (1482s since last reset, 11s since last boot)
Advertisizing started...
enter deep sleep
$
D$9EsL$BUR$D$g$@bS$start ESP32 135
deep sleep (1493s since last reset, 11s since last boot)
Advertisizing started...
enter deep sleep
KR$DHD$Sb$cf1$gQV$D$start ESP32 136
deep sleep (1504s since last reset, 11s since last boot)
Advertisizing started...
enter deep sleep
```



5.3. Establece un “chat” Bluetooth con perfil SPP con una APP en el móvil
Conexión serie mediante bluetooth.

Creamos un servidor bluetooth en el esp32 que muestra por pantalla los mensajes que recibe y que envíe los mensajes que escribamos por pantalla a los clientes

```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to enable it
#endif

char a;
BluetoothSerial SerialBT;

void setup() {
  Serial.begin(9600);
  SerialBT.begin("ESP32 de Jorge"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    a = Serial.read();
    SerialBT.write(a);
    Serial.print(a);
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

Con el siguiente resultado:

COM3

The device started, now you can pair it with bluetooth!
Prueba envio a movil
recibido
mensaje para pc



6. Comunicaciones Lora y LoraWAN

Para estas prácticas es necesario trabajar con micropython por tanto instalamos el entorno Visual Studio Code y en este instalamos el paquete Pymakr en la última versión 1.1.7. Además es necesario instalar NodeJS para que nos detecte la tarjeta y nos permita trabajar con ella. Utilizaremos la franja de 868 MHz para la comunicación. El modulo hardware queda de la siguiente forma:



6.1. Establece comunicaciones ping-pong con otros compañeros basada en broadcasts.

Para esta conexión utilizamos el protocolo Lora ya que enviamos mensajes a cualquier destinatario, es decir, no están cifrados por tanto son legibles por cualquier receptor.

El código es el siguiente, encargado de crear un socket y enviar por el los mensajes escritos por línea de comandos, además muestra por pantalla los mensajes que pueda recibir de otros dispositivos.

```
#!/usr/bin/env python
#
# Copyright (c) 2019, Pycom Limited.
#
# This software is licensed under the GNU GPL version 3 or any
# later version, with permitted additional terms. For more information
# see the Pycom Licence v1.0 document supplied with this file, or
# available at https://www.pycom.io/opensource/licensing
#
from network import LoRa
import socket

# Initialize LoRa in LORA mode.

# More params can be given, like frequency, tx power and spreading factor
# .
# Please pick the region that matches where you are using the device:
# Asia = LoRa.AS923
# Australia = LoRa.AU915
# Europe = LoRa.EU868
# United States = LoRa.US915
lora = LoRa(mode=LoRa.LORA, region=LoRa.EU868)

# create a raw LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
```

```

while True:

    # send some data
    s.setblocking(True)
    mensaje = input()
    s.send(mensaje)

    # get any data received...
    s.setblocking(False)
    data = s.recv(64)
    print(data)
    time.sleep(1)

```

6.2. Envía cada 30 segundos un número incremental de 2 bytes con protocolo LoRawan a un servidor en la nube.

Para este apartado nos registramos en la web oficial de The Things Network (TTN) y creamos una aplicación. Posteriormente registramos la dirección de nuestro dispositivo y ya tenemos la aplicación asignada a nuestro dispositivo y las credenciales correspondientes necesarias para hacer la comunicación punto a punto.

Código de identificación del dispositivo

```

1  from network import LoRa
2  import binascii
3  print(binascii.hexlify(LoRa().mac()).upper())

```

Nos remite la siguiente dirección

```

>>> Running j:\Master\RDSensores\repositorio_practicas\Practica_5\mac.py
>>>
>>>
b'70B3D549945815AA'

```

Y con ella identificamos el dispositivo, obteniendo las credenciales en la web de TTN:

Device EUI	<>	↔	70 B3 D5 49 94 58 15 AA	📋
Application EUI	<>	↔	70 B3 D5 7E D0 03 B1 41	📋
App Key	<>	↔	🔑 9E 04 98 37 FD 91 2F 1A 02 53 B5 4A 08 A9 9A 7A	📋

Una vez tenemos estos datos podemos crear la aplicación que se encarga del envío de información al servidor mediante el protocolo OTAA:

```

from network import LoRa

```

```
import socket
import time
import ubinascii

lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)

# create an OTAA authentication parameters, change them to the provide
d credentials
app_eui = ubinascii.unhexlify('70B3D57ED003B141')
app_key = ubinascii.unhexlify('9E049837FD912F1A0253B54A08A99A7A')
# uncomment to use LoRaWAN application provided dev_eui
#dev_eui = ubinascii.unhexlify('70B3D549938EA1EE')

# join a network using OTAA (Over the Air Activation)
#uncomment below to use LoRaWAN application provided dev_eui
lora.join(activation=LoRa.OTAA, auth=(app_eui, app_key), timeout=0)
#lora.join(activation=LoRa.OTAA, auth=(dev_eui, app_eui, app_key), tim
eout=0)

# wait until the module has joined the network
while not lora.has_joined():
    time.sleep(2.5)
    print('Not yet joined...')

print('Joined')
# create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# set the LoRaWAN data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)

# make the socket blocking
# (waits for the data to be sent and for the 2 receive windows to expi
re)
s.setblocking(True)

# send some data
s.send(bytes([0x01, 0x02, 0x03]))

# make the socket non-blocking
# (because if there's no data received it will block forever...)
s.setblocking(False)

# get any data received (if any...)
data = s.recv(64)
print(data)
```


Ejecutamos el programa

```
>>> Running j:\Master\RDSensores\repositorio_practicas\Practica_5\lorawan.py
>>>
>>>
Not yet joined...
Not yet joined...
Not yet joined...
Joined
b''
```

Y se comprueba en la web TTN que se ha recibido la información:

APPLICATION DATA

Filters

- uplink
- downlink
- activation
- ack
- error

	time	counter	port		
▲	12:04:45	0	2	retry	payload: 01 02 03

Uplink
Payload

01 02 03

En caso de enviar un comando de texto:

```
# send some data
#s.send(bytes([0x01, 0x02, 0x03]))
s.send("hola")
```

El servidor lo recibe en hexadecimal, no en ASCII:

APPLICATION DATA

Filters

- uplink
- downlink
- activation
- ack
- error

	time	counter	port		
▲	13:38:33	0	2	retry	payload: 68 6F 6C 61

Uplink
Payload


68 6F 6C 61


Paste hex numbers or drop file


686F6C61

Character encoding

ASCII

 Convert

 Reset

 Swap

hola