UNIVERSIDAD DE ALMERÍA

Proyecto 2: Control de Presencia

INSTALACIONES INDUSTRIALES AVANZADAS

Pedro José Fenoy Illacer, Carlos Nache Romo, Néstor Pastor Gutiérrez, Ángel Oller Oller, Javier López Milán, Gustavo Martín de Dios

DICIEMBRE 2017 | MÁSTER INGENIERÍA INDUSTRIAL

Índice de Contenido

Índice de Ilustraciones

1. Características del material utilizado para el desarrollo de la práctica

1.1. Placa Waspmote

- Características Generales:

  - Microcontrolador: ATmega 1281.
  - Frecuencia: 14.7456 Mhz.
  - SRAM: 8 KB.
  - EEPROM: 4 KB.
  - FLASH: 128 KB.
  - SD Card: 2 GB.
  - Peso: 20 gr.
  - Dimensiones: 73.5 x 51 x 13 mm.
  - Rango de Temperatura: [-10ºC, +65ºC].
  - Reloj: RTC (32 Khz).

- Consumo:

  - ON: 17 mA.
  - Sleep: 30 uA.
  - Deep Sleep: 33 uA.
  - Hibernate: 7 uA.
  - Funcionamiento sin recarga: 1año (En el modo Hibernate).

- Entradas y Salidas:

  - 7 entradas analógicas.
  - 8 entradas/salidas digitales.
  - 1 PWM (Pulse Width Modulation - Modulación por ancho de pulsos).
  - 2 UART (Universal Asynchronous Receiver-Transmitter).
  - 1 I2C (Inter-Integrated Circuit).
  - 1 USB.
  - 1 SPI (Scholarly Publishers Indicators).

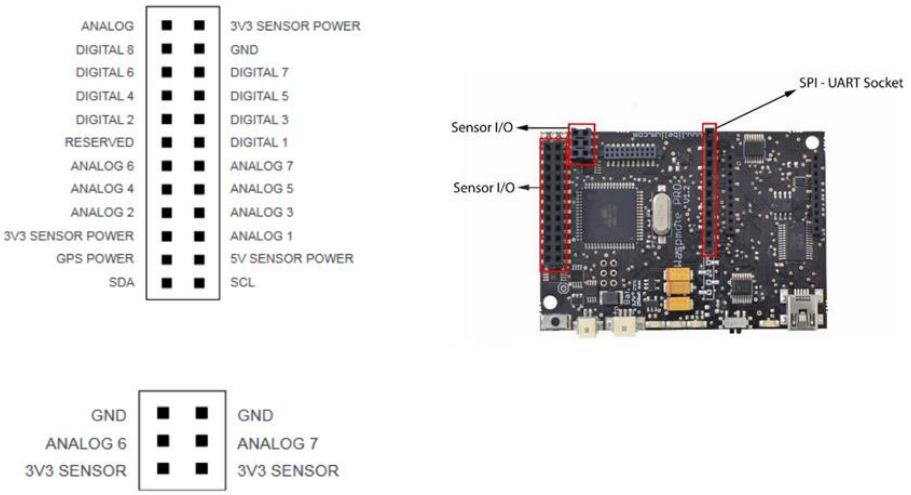

Ilustración 1. Puertos de entrada y salida de la Waspmote

- <u>Características Eléctricas:</u>

  - Tensión de batería: 3.3V - 4.2V.
  - Carga USB: 5V – 480 mA.
  - Carga placa solar: 6 - 12V - 330 mA.

- <u>Sensores integrados en la placa:</u>

  - Acelerómetro: ±2g ±4g ±8g.
  - Baja potencia: 0.5Hz/1Hz/2Hz/5Hz/10Hz.
  - Modo normal: 50Hz/100Hz/400Hz/1000Hz.

La placa base permite modificar su firmware de manera remota. Podemos conectar diversos módulos de comunicación (wifi, bluetooth, GSM/GPRS, RFID/NFC y GPS).

También dispone de multitud de módulos-sensores opcionales:

- Gases (CO, CO2, O2, CH4, H2, NH3, C4H10, CH3Ch2OH, C6H5CH3, H2S, NO2, O3, VOC), temperatura, humedad, presión atmosférica.
- Eventos (presión/peso, doblez/curvatura, vibración, impacto, efecto Hall, inclinación, presencia de líquido, nivel de líquido, luminosidad, presencia (PIR), estiramiento)
- Smart Cities (micrófono, detección de grietas, propagación de grietas, desplazamiento lineal, polvo, ultrasonidos, temperatura, humedad, luminosidad).
- Campo magnético.
- Sensores para agricultura.
- Radiación.

1.2. Mini USB Cable

Permite conectar mediante un cable USB a un PC para utilizarlo como fuente de alimentación. También permite la transmisión del programa a la placa.

1.3. Led Rojo / Verde

Un led o diodo emisor de luz, es una fuente de luz constituida por un material semiconductor dotado de dos terminales. Se trata de un diodo de unión p-n, que emite luz cuando está activado. Si se aplica una tensión adecuada a los terminales, los electrones se recombinan con los huecos en la región de la unión p-n del dispositivo, liberando energía en forma de fotones**.**

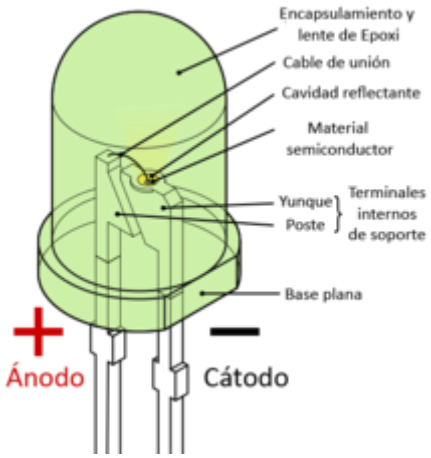

Ilustración 2. Diodo led

1.4. Waspmote Events Board

- Especificaciones:

  - Peso: 20 gr.
  - Dimensiones: 73.5 x 51 x 1.3 mm.
  - Rango de Temperatura: [-20ºC, +65ºC].

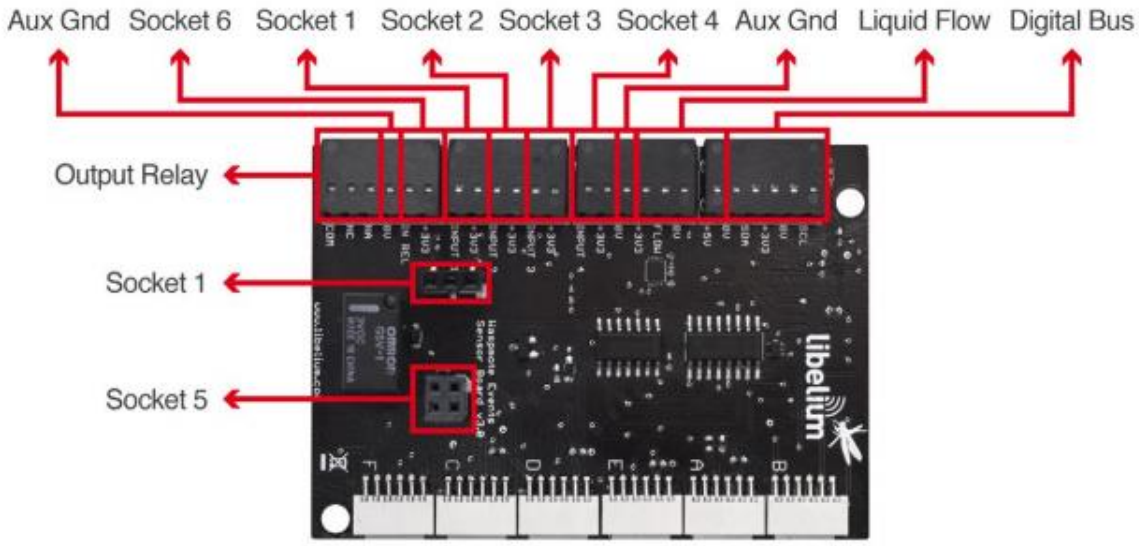

Ilustración 3. Waspmote Events Board

- <u>Características Eléctricas:</u>

  - Voltaje de la placa: 3.3 V.
  - Voltaje del sensor: 3.3 V.
  - Máxima corriente admitida (continua): 200 mA.
  - Máxima corriente admitida (pico): 400 mA.

- <u>Descripción general:</u>

Los sensores están activos en la “Events sensor Board Sensor” mientras Waspmote está en modo “Sleep” o modo “Deep Sleep”. Cuando un sensor cambia su señal de salida a un valor "alto", se genera una señal que despierta a la mota de su estado de bajo consumo y le dice qué sensor ha generado la señal. Si Waspmote estuviera en modo activo (activado), recibiría la interrupción y podría responder de la misma manera que en el caso anterior.

- <u>Funcionamiento:</u>

La placa permite la conexión simultánea con hasta 5 sensores cuyas salidas se combinan en una puerta lógica OR que implementa un cambio en el bit de interrupción que despierta la mota. También puede conectar diferentes dispositivos I2C, además de una salida de relé y una entrada de relé. El sensor que ha interrumpido la mota se identifica en un registro de desplazamiento que Waspmote puede leer una vez que está en funcionamiento normal.

- <u>Pir Sensor</u>

  - Altura: 25.4 mm.
  - Anchura: 24.3 mm.
  - Longitud: 28.0 mm.

- Consumo: 100 µA.
- Rango de detección: 6 ~ 7 m.
- Rango espectral: ~ 10 µm.

El sensor PIR (Passive Infra Red) es un sensor piroeléctrico que consiste principalmente en un receptor de infrarrojos y una lente de enfoque que basa su operación en el monitoreo de las variaciones en los niveles de recepción de los infrarrojos detectados, lo que refleja este movimiento estableciendo su señal de salida alta. Al ser un sensor digital, debe estar situado en el socket 1, socket 2, socket 3, socket 4 y socket 6. El espectro de 10 µm corresponde a la radiación de calor de la mayoría de los mamíferos, ya que emiten temperaturas de alrededor de 36 °C. La dirección de detección máxima es perpendicular a la placa del sensor de eventos, por eso se recomienda colocar Waspmote perpendicular al suelo cuando se usa el sensor PIR.

- Luminosity Sensor

  - Características Eléctricas:

    - Rango dinámico: 0.1 to 40000 lux.
    - Rango espectral: 300 ~ 1100 nm.
    - Rango de Voltajes: 2.7 ~ 3.6 V.
    - Corriente de alimentación: 0.24 mA.
    - Máxima corriente modo Sleep: 0.3 µA.
    - Rango de Temperatura: [-30ºC, +70ºC].

  - Proceso de medición:

Es un convertidor “light-to-digital” que transforma la intensidad de la luz en una salida de señal digital. Este dispositivo combina un fotodiodo de banda ancha (visible más infrarrojo) y un fotodiodo de respuesta infrarroja en un solo circuito integrado CMOS capaz de proporcionar una respuesta casi fotópica sobre un rango dinámico efectivo de 20 bits (resolución de 16 bits). Dos ADCs integradores convierten las corrientes de fotodiodo a una salida digital que representa la irradiancia medida en cada canal. Esta salida digital en lux se deriva usando una fórmula empírica para aproximar la respuesta del ojo humano.

- WIFI Pro Onchip:

El módulo WIFI Waspmote incorpora la radio 802.11 b/g, procesador de 32 bits, TCP/IP pila, reloj en tiempo real, el acelerador de cifrado, la unidad de administración de energía y la interfaz de sensor analógico.
A continuación se muestra en una tabla comparativa las diferencias entre el sistema WiFi y el sistema WiFi PRO, donde se puede observar el gran número de ventajas si se trabaja con WiFi PRO.

	[v12] WiFi	[v15] WiFi PRO
Simultaneous TCP/UDP sockets	1	10
HTTP GET	Yes	Yes
HTTPS POST	No	Yes
HTTPS GET	No	Yes
HTTPS POST	No	Yes
FTP	Yes	Yes
Multiple SSIDs	No	Yes
Roaming mode	No	Yes
Max Tx power	12 dBm	17 dBm
Max Power Consumption	120 mA	350 mA

Ilustración 4. WiFi vs WiFi PRO

2. Pasos llevados a cabo para el desarrollo del proyecto

2.1. Funcionamiento de la placa

1) El sensor recoge la información de la habitación y se estabiliza.
2) Tras esto, el sensor enciende una luz roja en caso de que no detecte presencia.
3) Cuando detecta algún movimiento, el sensor detecta presencia y enciende un led verde.
4) Por último, se ha establecido que es de noche cuando el nivel de iluminación cae por debajo de los 10 luxes. Cuando el sensor de luminosidad detecta una caída por debajo de este umbral, se detiene todo el proceso anterior y deja encendido el led verde.

2.2. Configuración de la placa

1) En el socket 1 de la Waspmote Events Sensor Board v3.0 se ha conectado el sensor de presencia. Esto se ha realizado conectado las patillas de la siguiente forma:

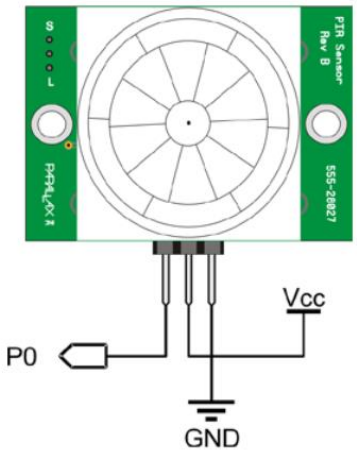

Ilustración 5. Waspmote Events Sensor Board v3.0

- La patilla GND se ha conectado en el pin 21 de la Waspmote.
- La patilla de VCC se ha conectado al pin 17 de la Waspmote.
- La salida del sensor se ha conectado al pin 18 de la Waspmote.

2) El socket 5 de esta placa se ha utilizado para conectar el sensor de luminosidad.
3) En la placa Waspmote PRO v1.5 se han utilizado los pines Analog 6 para encender el led verde y la Analog 7 para el led rojo.

2.3. Programación

Para la programación del sensor PIR, se han partido de los siguientes ejemplos: Ev_v30_02_PIR y Ev_v30_14_luxes. A continuación, se realiza una explicación del código incluyendo los diferentes cambios aplicados para lograr el objetivo planteado anteriormente.

En primer lugar, se definen las bibliotecas a utilizar y las variables que posteriormente utilizaremos. Además, se asocia el sensor PIR con su socket correspondiente.

```
#include <WaspSensorEvent_v30.h>

uint8_t value = 0;
uint32_t luxes = 0;

/*
 * Define object for sensor. Choose board socket.
 * Waspmote OEM. Possibilities for this sensor:
 *   - SOCKET_1
 *   - SOCKET_2
 *   - SOCKET_3
 *   - SOCKET_4
 *   - SOCKET_6
 * P&S! Possibilities for this sensor:
 *   - SOCKET_A
 *   - SOCKET_C
 *   - SOCKET_D
 *   - SOCKET_E
 */
pirSensorClass pir(SOCKET_1);
```

Ilustración 6. Inicialización del código

Tras esto se inicia el programa y se establecen los pines 19 y 20 como salidas. Además, se activa por primera vez el sensor PIR para su estabilización y se activan las interrupciones.

```
void setup()
{
  // Turn on the USB and print a start message
  USB.ON();
  USB.println(F("Start program"));

  // initialize digital pin LED_BUILTIN as an output.
  pinMode(19, OUTPUT);
  pinMode(20, OUTPUT);

  // Turn on the sensor board
  Events.ON();

  // Firstly, wait for PIR signal stabilization
  value = pir.readPirSensor();
  while (value == 1)
  {
    USB.println(F("...wait for PIR stabilization"));
    delay(1000);
    value = pir.readPirSensor();
  }

  // Enable interruptions from the board
  Events.attachInt();
}
```

Ilustración 7. Activación del sensor PIR

Una vez estabilizado el sensor, se comienza a leer la información del sensor y a su vez entra en funcionamiento el sensor de luminosidad. Si el nivel de luminosidad está por encima de 10 luxes (hemos tomado que por encima de 10 luxes es de día) existen dos posibilidades:

- El sensor PIR detecta movimiento (value==1) y, tras apagar el led rojo, enciende el led verde.
- El sensor PIR no detecta movimiento y se enciende el led rojo.

```
void loop()
{
  ///////////////////////////////////////
  // 1. Read the sensor level
  ///////////////////////////////////////
  // Read the PIR Sensor
  value = pir.readPirSensor();
  luxes = Events.getLuxes(INDOOR);

  if (luxes >= 10)
  {
    digitalWrite(19, LOW);    // turn the GREEN LED off by making the voltage LOW
    digitalWrite(20, LOW);    // turn the RED LED off

  // Print the info
  if (value == 1)
  {
    USB.println(F("Sensor output: Presence detected"));
    digitalWrite(20, LOW);    // turn the LED off
    digitalWrite(19, HIGH);   // turn the GREEN LED on

  }
  else
  {
    USB.println(F("Sensor output: Presence not detected"));
    digitalWrite(19, LOW);    // turn the LED off by making the voltage LOW
    digitalWrite(20, HIGH);   // turn the RED LED
  }
```

Ilustración 8. Funcionamiento de sensor de luminosidad

Realizada la primera comprobación, el sensor PIR entra en modo “sleep” durante 10 segundos o hasta que se produzca una interrupción.

```
///////////////////////////////////////
// 2. Go to deep sleep mode
///////////////////////////////////////
USB.println(F("enter deep sleep"));
PWR.deepSleep("00:00:00:10", RTC_OFFSET, RTC_ALM1_MODE1, SENSOR_ON);
USB.ON();
USB.println(F("wake up\n"));
```

Ilustración 9. PIR en modo “sleep”

En caso de que se produzca una interrupción, se activa una alarma que manda un mensaje por pantalla avisando de la misma.

```
////////////////////////////////////////
// 3. Check Interruption Flags
////////////////////////////////////////

// 3.1. Check interruption from RTC alarm
if (intFlag & RTC_INT)
{
  USB.println(F("-----------------------------"));
  USB.println(F("RTC INT captured"));
  USB.println(F("-----------------------------"));

  // clear flag
  intFlag &= ~(RTC_INT);
}

// 3.2. Check interruption from Sensor Board
if (intFlag & SENS_INT)
{
  // Disable interruptions from the board
  Events.detachInt();

  // Load the interruption flag
  Events.loadInt();

  // In case the interruption came from PIR
  if (pir.getInt())
  {
    USB.println(F("-----------------------------"));
    USB.println(F("Interruption from PIR"));
    USB.println(F("-----------------------------"));
  }
```

Ilustración 10. Activación de alarma

Inmediatamente después de la interrupción se realiza una nueva lectura del sensor, repitiéndose el proceso explicado anteriormente para encender un led u otro. Por otro lado, una vez terminado este proceso se para la interrupción.

```
  value = pir.readPirSensor();


if (value == 1)
{
        USB.println(F("Sensor output: Presence detected"));
        digitalWrite(20, LOW);    // turn the LED off by making the voltage LOW
        digitalWrite(19, HIGH);   // turn the GREEN LED on

}
else
  {

        USB.println(F("Sensor output: Presence not detected"));
        digitalWrite(19, LOW);    // turn the LED off by making the voltage LOW
        digitalWrite(20, HIGH);   // turn the RED LED
  }



  while (value == 1)
  {
    USB.println(F("...wait for PIR stabilization"));
    delay(1000);
    value = pir.readPirSensor();
    luxes = Events.getLuxes(INDOOR);
  }

  // Clean the interruption flag
  intFlag &= ~(SENS_INT);

  // Enable interruptions from the board
  Events.attachInt();
}
```

Ilustración 11. Repetición del proceso y parada de la interrupción

Por último, en caso de que los luxes estén por debajo de 10 (es de noche) no se realiza el proceso anterior, sino que directamente se mantiene el led verde encendido hasta que el nivel de luxes supere ese umbral. Un detalle para tener en cuenta es que se ha utilizado la interrupción RCP cada 10 segundos, por lo que la lectura de los luxes no es continua, sino que se realiza una lectura de este sensor en intervalos de este mismo tiempo.

```
else
{
        digitalWrite(20, LOW);    // turn the LED off by making the voltage LOW
        digitalWrite(19, HIGH);   // turn the GREEN LED
}
```

Ilustración 12. Led verde con luxes bajos

3. Mostrar distintas configuraciones de salida de los sensores que demuestren que se ha realizado el proyecto

En los videos adjuntados junto con este documento se pueden observar el correcto funcionamiento de la Waspmote, verificando así la realización del proyecto. Se han adjuntado dos videos, uno del problema original propuesto y otro con la ampliación propuesta usando el WiFi Pro Onchip.

4. Ampliación: ¿Cómo podría enviarse la hora de la última detección de presencia a otra placa Waspmote utilizando WIFI Pro Onchip?

4.1. Inicialización fecha y hora

En primer lugar, es necesario introducir la fecha y hora en la placa. En función de los componentes que se disponen no es posible leer los datos a partir del sistema al que esta se encuentra conectada, para que la placa realizase dicha operación de forma autónoma los datos de día y hora se deberían leer a partir de una tarjeta con conexión GPRS o 3G.

A continuación, se describe el código que realiza dicha operación de almacenamiento de los datos de hora y fecha.

Se requiere por pantalla que es necesario almacenar los datos correspondientes.

```
/*****************************************
*
*  Set time
*
*****************************************/

void decideTime()
{

  // Powers RTC up, init I2C bus and read initial values
  RTC.ON();


  USB.println(F("---------------------------------------"));
  USB.println(F("Set RTC Date and Time via USB port"));
  USB.println(F("---------------------------------------"));
```

Ilustración 13. Requerimiento de almacenamiento de datos

A partir de este punto se requiere que se introduzcan por pantalla los datos actuales correspondientes a: año, mes, día, horas, minutos y segundos. Estas acciones están apoyadas por la función GetData() que se mostrará posteriormente.

- Introducción del año:

```
////////////////////////////////
//  YEAR
////////////////////////////////
do
{
  USB.print("Insert year [yy]:");
}
while( getData(2) != true );

year=atoi(input);
USB.println(year);
```

Ilustración 14. Introducción de año

- Introducción del mes:

```
////////////////////////////////
//  MONTH
////////////////////////////////
do
{
  USB.print("Insert month [mm]:");
}
while( getData(2) != true );

month=atoi(input);
USB.println(month);
```

Ilustración 15. Introducción de mes

- Introducción del día:

```
////////////////////////////////
//  DAY
////////////////////////////////
do
{
  USB.print("Insert day [dd]:");
}
while( getData(2) != true );

day=atoi(input);
USB.println(day);
```

Ilustración 16. Introducción del día

- Introducción de la hora:

```
////////////////////////////////
//  HOUR
////////////////////////////////
do
{
  USB.print("Insert Hour [HH]:");
}
while( getData(2) != true );

hour=atoi(input);
USB.println(hour);
```

Ilustración 17. Introducción de la hora

- Introducción de los minutos:

```
////////////////////////////////
//  MINUTE
////////////////////////////////
do
{
  USB.print("Insert minute [MM]:");
}
while( getData(2) != true );

minute=atoi(input);
USB.println(minute);
```

Ilustración 18. Introducción de los minutos

- Introducción de los segundos:

```
////////////////////////////////
// SECOND
////////////////////////////////
do
{
 USB.print("Insert second [SS]:");
}
while( getData(2) != true );

second=atoi(input);
USB.println(second);
```

Ilustración 19. Introducción de los segundos

Se crea un buffer en el cual se almacenan los datos que se han ido requiriendo por pantalla, para tenerlos disponibles cuando en los siguientes pasos sea necesario su uso.

```
  ////////////////////////////////
  //  create buffer
  ////////////////////////////////
  sprintf(buffer, "%02u:%02u:%02u:%02u:%02u:%02u:%02u",
                                                    year,
                                                    month,
                                                    day,
                                                    RTC.dow(year, month,day),
                                                    hour,
                                                    minute,
                                                    second );
  USB.println(buffer);

  // Setting time [yy:mm:dd:dow:hh:mm:ss]
  RTC.setTime(buffer);
}
```

Ilustración 20. Creación del buffer

Finalmente, con la función RTC.setTime se actualiza la información de fecha y hora que utilizará la placa.

Con la función siguiente se verifica que los datos que se van introduciendo en los puntos anteriores son de dos enteros como máximo, cabe mencionar, que es necesario introducir los datos de forma correcta, ya que la siguiente función no diferencia entre datos válidos o no.

```
/*********************************
*
* get numBytes from USB port
*
*********************************/
boolean getData(int numBytes)
{
  memset(input, 0x00, sizeof(input) );
  int i=0;
  USB.flush();
  int nRead=0;

  while(!USB.available());

  while(USB.available()>0)
  {
    input[i]=USB.read();

    if( (input[i]=='\r') && (input[i]=='\n') )
    {
      input[i]='\0';
    }
    else
    {
      i++;
    }
  }

  nRead=i;

      if(nRead != numBytes)
      {
        USB.print(F("must write "));
        USB.print(numBytes, DEC);
        USB.print(F(" characters. Read "));
        USB.print(nRead, DEC);
        USB.println(F(" bytes"));
        return false;
      }
      else
      {
        input[i]='\0';
        return true;
      }

   }
```

Ilustración 21. Comprobación de datos

4.2. Envío de aviso

Para la comunicación de la placa WASPMOTE, dada la imposibilidad de realizar la simulación de comunicación con otra placa, por la falta de material, se ha decidido realizar la comunicación mediante el Protocolo de Control de Transmisión con un teléfono móvil.

El Protocolo de control de transmisión (en inglés Transmission Control Protocol o TCP), es uno de los protocolos fundamentales en Internet. Fue creado entre los años 1973 y 1974 por Vint Cerf y Robert Kahn.

Muchos programas dentro de una red de datos compuesta por redes de computadoras pueden usar TCP para crear "conexiones" entre sí a través de las cuales puede enviarse un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. También proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto de puerto.

TCP da soporte a muchas de las aplicaciones más populares de Internet (navegadores, intercambio de ficheros, clientes FTP, etc.) y protocolos de aplicación HTTP, SMTP, SSH y FTP.

- Funcionamiento del protocolo en detalle:

Las conexiones TCP se componen de tres etapas:

1. Establecimiento de conexión.
2. Transferencia de datos.
3. Fin de la conexión.

Para establecer la conexión se usa el procedimiento llamado "negociación en tres pasos" (3-way handshake). Para la desconexión se usa una "negociación en cuatro pasos" (4-way handshake). Durante el establecimiento de la conexión, se configuran algunos parámetros tales como el número de secuencia con el fin de asegurar la entrega ordenada de los datos y la robustez de la comunicación.

A partir de este punto se describe el código que realiza la operación anterior:

Inicialización de la función:

```
#include <WaspWIFI_PRO.h>

// choose socket (SELECT USER'S SOCKET)
uint8_t socket = SOCKET0;

// WiFi AP settings (CHANGE TO USER'S AP)
////////////////////////////////////////
char ESSID[] = "NET";
char PASSW[] = "PASSWORD";

// choose TCP server settings
////////////////////////////////////////
char HOST[]        = "192.168.1.217";
char REMOTE_PORT[] = "12345";
char LOCAL_PORT[]  = "3000";
```

Ilustración 22. Inicialización de la función

La tarjeta WIFI PRO onChip está conectada al SOCKET0, se inicializa este socket y se define el nombre de la red junto la contraseña.

Para definir el servidor, que estará conectado a la misma red se necesita definir el HOST y el PUERTO.

A continuación, se explican los pasos llevados a cabo para la comunicación de WASPMOTE con el servidor

```
void sendWarning()
{
```

Activación de la placa WIFI PRO ON CHIP:

```
////////////////////////////////////////////////
// 1. Switch ON
////////////////////////////////////////////////
error = WIFI_PRO.ON(socket);

if (error == 0)
{
  USB.println(F("1. WiFi switched ON"));
}
else
{
  USB.println(F("1. WiFi did not initialize correctly"));
}
```

Ilustración 23. Activación de la placa WiFi

Se restablecen los valores predeterminados por defecto:

```
////////////////////////////////////////////////
// 2. Reset to default values
////////////////////////////////////////////////
error = WIFI_PRO.resetValues();

if (error == 0)
{
  USB.println(F("2. WiFi reset to default"));
}
else
{
  USB.println(F("2. WiFi reset to default ERROR"));
}
```

Ilustración 24. Restablecimiento de valores

WIFI PRO ON CHIP busca el nombre de la red proporcionado y verifica que coincide:

```
////////////////////////////////////////////////
// 3. Set ESSID
////////////////////////////////////////////////
error = WIFI_PRO.setESSID(ESSID);

if (error == 0)
{
  USB.println(F("3. WiFi set ESSID OK"));
}
else
{
  USB.println(F("3. WiFi set ESSID ERROR"));
}
```

Ilustración 25. Verificación de que la red es correcta

Al igual que en el código anterior verifica que coincide la contraseña:

```
////////////////////////////////////////////////
// 4. Set password key (It takes a while to generate the key)
// Authentication modes:
//    OPEN: no security
//    WEP64: WEP 64
//    WEP128: WEP 128
//    WPA: WPA-PSK with TKIP encryption
//    WPA2: WPA2-PSK with TKIP or AES encryption
////////////////////////////////////////////////
error = WIFI_PRO.setPassword(WPA2, PASSW);

if (error == 0)
{
  USB.println(F("4. WiFi set AUTHKEY OK"));
}
else
{
  USB.println(F("4. WiFi set AUTHKEY ERROR"));
}
```

Ilustración 26. Verificación de que la contraseña es correcta

Una vez que el módulo se ha establecido en la configuración correcta, se mantienen en la memoria no volátil del módulo. Además, es obligatorio reiniciar el módulo para obligar al módulo a usar la nueva configuración. Por lo que la función softReset () se usa para realizar un restablecimiento de software al módulo. Después de llamar a esta función, la nueva configuración tendrá efecto.

```
////////////////////////////////////////////////
// 5. Software Reset
// Parameters take effect following either a
// hardware or software reset
////////////////////////////////////////////////
error = WIFI_PRO.softReset();

if (error == 0)
{
  USB.println(F("5. WiFi softReset OK"));
  WIFI_PRO.setTimeFromWIFI();// set time from WIFI
  USB.println(RTC.getTime());
}
else
{
  USB.println(F("5. WiFi softReset ERROR"));
}

// get current time
previous = millis();
```

Ilustración 27. Restablecimiento de software al módulo

Una vez que el módulo tiene una configuración válida en la memoria no volátil, automáticamente comienza a buscar unirse al punto de acceso. La función isConnected () permite saber si el módulo WiFi PRO ya está conectado al punto de acceso. Esta función devuelve valores verdaderos o falsos para proporcionar la información de estado.

```
////////////////////////////////////////////////
// 6. Join AP
////////////////////////////////////////////////

// check connectivity
status =  WIFI_PRO.isConnected();

// Check if module is connected
if (status == true)
{
  USB.print(F("6. WiFi is connected OK."));
  USB.print(F(" Time(ms):"));
  USB.println(millis()-previous);

  error = WIFI_PRO.ping("www.google.com");

  if (error == 0)
  {
    USB.print(F("7. PING OK. Round Trip Time(ms)="));
    USB.println( WIFI_PRO._rtt, DEC );
  }
  else
  {
    USB.println(F("7. Error calling 'ping' function"));
  }
}
else
{
```

Ilustración 28. Comprobación de si está conectado al punto de acceso

La función ping () envía un paquete de solicitud ICMP PING de dos bytes al host remoto definido como argumento de entrada. La entrada de la función puede ser un nombre lógico del host de destino o una dirección IP de host. Al recibir con éxito una respuesta ICMP PING del host, el tiempo de ida y vuelta en milisegundos se devuelve (RTT).

```
    USB.print(F("6. WiFi is connected ERROR."));
    USB.print(F(" Time(ms):"));
    USB.println(millis()-previous);
  }
```

Ilustración 29. Respuesta del host

La función setTCPclient () abre un socket de cliente de Protocolo de control de transmisión (TCP) e intenta conectarse al puerto especificado en el servidor definido como entrada. Por lo tanto, esta función necesita tres entradas diferentes:

- Host: el nombre del servidor puede ser cualquier nombre legal de servidor de Internet que pueda resolverse mediante el DNS del módulo (Dominio Configuración del Servidor de nombres). El nombre del servidor también se

puede especificar como una dirección IP absoluta dada en punto decimal notación.

- Puerto remoto: se supone que el sistema del servidor está escuchando en el puerto especificado.
- Puerto local: este es el puerto local cuando se abre el socket TCP.

```
////////////////////////////////////////////////
// 8. TCP    CLIENT!!!!
////////////////////////////////////////////////

// Check if module is connected
if (status == true)
{

////////////////////////////////////////////////
// 8.1. Open TCP socket
////////////////////////////////////////////////

error = WIFI_PRO.setTCPclient(HOST, REMOTE_PORT, LOCAL_PORT);

// check response
if (error == 0)
{
  // get socket handle (from 0 to 9)
  socket_handle = WIFI_PRO._socket_handle;

  USB.print(F("8.1. Open TCP socket OK in handle: "));
  USB.println(socket_handle, DEC);
}
else
{
  USB.println(F("8.1. Error calling 'setTCPclient' function"));
  WIFI_PRO.printErrorCode();
  status = false;
}

if (status == true)
{
```

Ilustración 30. Apertura del socket de TCP

La función send () envía una secuencia de bytes al socket especificado por la entrada del manejador de socket. Esta función necesita dos entradas diferentes:

- Socket handle: Un manejador de socket TCP / UDP de un socket previamente abierto.
- Data: esta es la secuencia de datos para enviar al socket TCP / UDP. Este flujo de datos se puede definir como un simple mensaje de cadena o una matriz de bytes, especificando una tercera entrada para la longitud de la matriz de bytes a enviar.

```
////////////////////////////////////////////////
// 8.2. send data
////////////////////////////////////////////////

//USB.println(RTC.getTime());

strcpy(mensaje," \nAn intruder has been detected at ");
strcat(mensaje,RTC.getTime());

error = WIFI_PRO.send( socket_handle, mensaje);//"This is a message from Waspmote  !!\n");


// check response
if (error == 0)
{
  USB.println(F("8.2. Send data OK"));
}
else
{
  USB.println(F("8.2. Error calling 'send' function"));
  WIFI_PRO.printErrorCode();
}
```

Ilustración 31. Envío de bytes al socket

La función receive () recibe una secuencia de bytes del socket TCP / UDP especificado por el socket handle. Los datos recibidos sólo son válidos si ya se encuentran en el búfer de entrada del socket del módulo en el momento en que se emite este comando.

```
////////////////////////////////////////////////
// 8.3. Wait for answer from server
////////////////////////////////////////////////
USB.println(F("Listen to TCP socket:"));
error = WIFI_PRO.receive(socket_handle, 300);//30000);

// check answer
if (error == 0)
{
  USB.println(F("\n========================================="));
  USB.print(F("Data: "));
  USB.println( WIFI_PRO._buffer, WIFI_PRO._length);

  USB.print(F("Length: "));
  USB.println( WIFI_PRO._length,DEC);
  USB.println(F("========================================="));
}
```

Ilustración 32. Recepción de bytes del socket

La función closeSocket () permite al usuario cerrar un cliente TCP / UDP previamente abierto. La función necesita un parámetro de entrada para el identificador de socket:

Socket handle: el identificador de socket utilizado para abrir la conexión.

```
        ////////////////////////////////////////////////
        // 8.4. close socket
        ////////////////////////////////////////////////
        error = WIFI_PRO.closeSocket(socket_handle);

        // check response
        if (error == 0)
        {
          USB.println(F("8.3. Close socket OK"));
        }
        else
        {
          USB.println(F("8.3. Error calling 'closeSocket' function"));
          WIFI_PRO.printErrorCode();
        }
      }
    }

    delay(10000);

}
```

Ilustración 33. Clausura de la comunicación

A continuación, se muestra cómo aparece en el receptor de la información a través del WiFi.

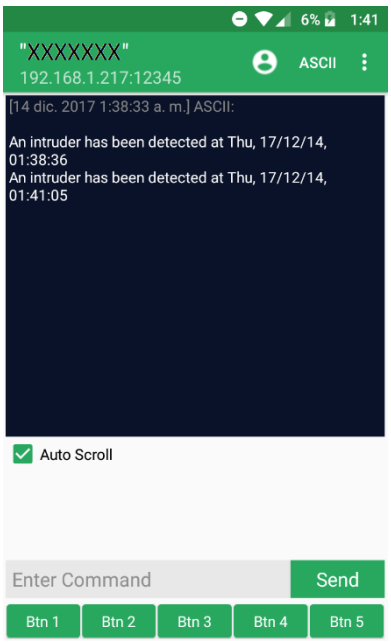

Ilustración 34. Recepción de la información