ikerlan

PROTOTIPO. Sistema de seguridad para freno de vehículo.

Daniel Pillado y Mario Eguillor

ÍNDICE:

REQUISITOS FUNCIONALES:

1.- Pulsador para iniciar el sistema, a continuación el display LCD mostrará “SYSTEM ON”.

2.- El sistema debe leer continuamente la velocidad del vehículo (emulada con un potenciómetro conectado al módulo ADC). Se hace uso de un filtro de mediana para evitar valores outliers.

3.- Los valores de salida del filtro deben ser enviados a la RPi mediante UART. La RPi guardará los valores de lectura en un log.

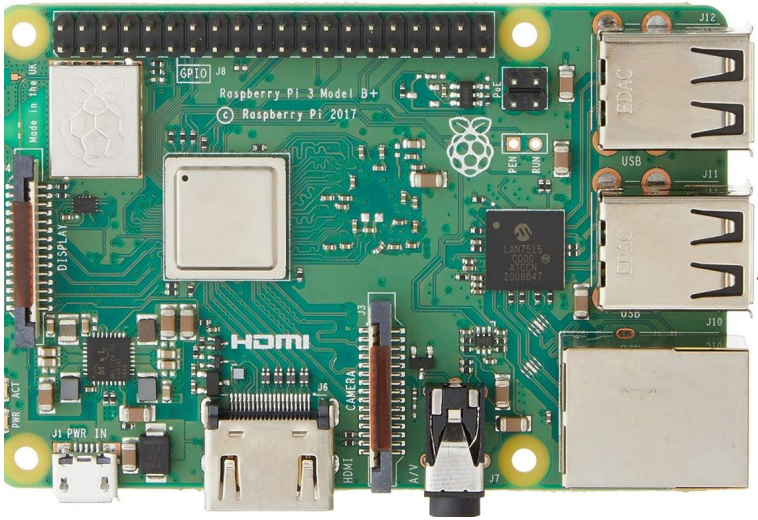
4, 5.- Cuando la velocidad del vehículo supere el umbral del 75% de la velocidad máxima, emitirá un comando de freno de emergencia. El comando consiste en: parpadear el LED, mostrar por la pantalla del display LCD (mediante I2C), la velocidad actual en la primera línea, y los caracteres “PELIGRO” en la segunda línea. Esto se mantiene mientras la velocidad no baje del umbral . Al activar el freno, la EK hará sonar un zumbador durante un segundo, una vez.

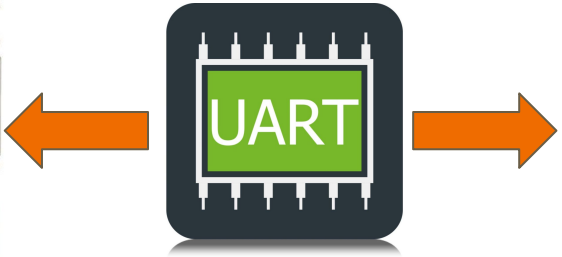
6.- Mensaje a través de MQTT cada vez que se activa el freno de emergencia.

7.- Cuando la velocidad leída baje del umbral de emergencia, el LED dejará de parpadear y en el display LCD ya no aparecerá el mensaje peligro, ni la velocidad actual, volverá a escribir el mensaje “SYSTEM ON”.

8.- Visualización en terminal del PC escuchando las publicaciones.

COMPONENTES/HARDWARE

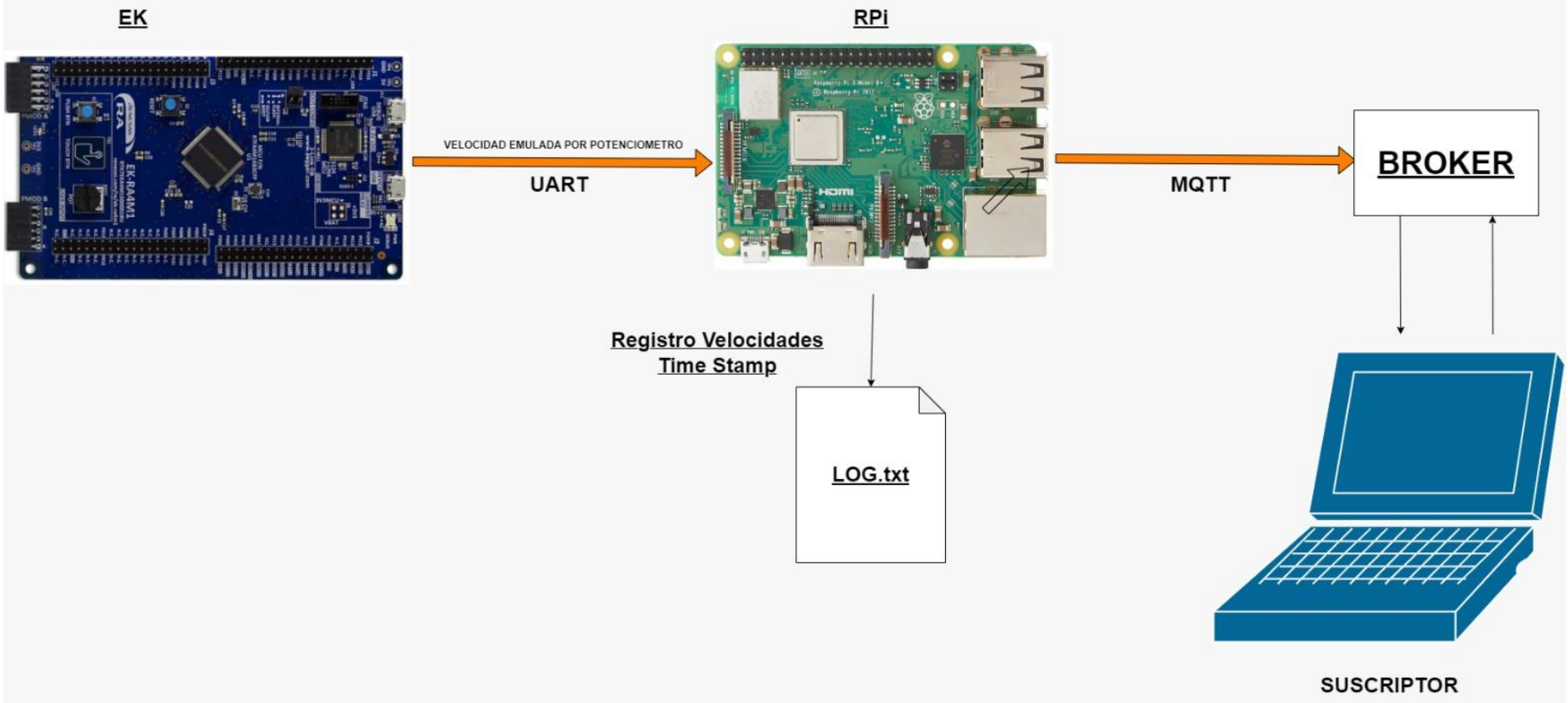




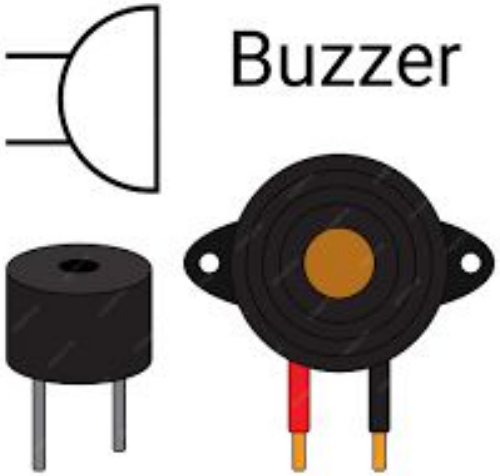
Raspberry Pi 3 Model B Rev 1.2

Renesas RA Family RA4 Series EK-RA4M1

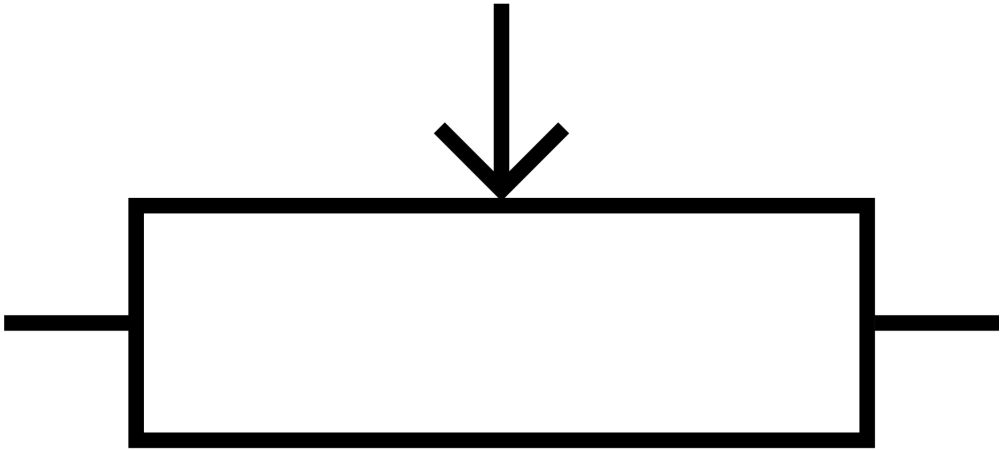
ESQUEMA DE CONEXIONES:



COMPONENTES/HARDWARE



Bocina



Potenciómetro

SOFTWARE/CODIGO

Ejercicio 1:

Pulsador para iniciar el sistema: el sistema de seguridad sólo se iniciará tras activar el pulsador, y a continuación el display LCD mostrará: “SYSTEM ON”. Una vez activado el sistema, estará activo de manera indefinida mientras la EK esté alimentada.

SOFTWARE/CODIGO

Ejercicio 1:

```
//PULSADOR
void init_pulsador(void) {
    fsp_err_t err;
    // Inicializar la interrupción externa (pulsador)
    err = R_ICU_ExternalIrqOpen(&g_external_irq0_ctrl, &g_external_irq0_cfg);
    assert(FSP_SUCCESS == err);

    // Establecer la función de callback para la interrupción externa
    err = R_ICU_ExternalIrqCallbackSet(&g_external_irq0_ctrl, btn_callback, NULL, NULL);
    assert(FSP_SUCCESS == err);

}

//FUNCIONES PRINCIPALES

//BOTON
void btn_callback(external_irq_callback_args_t *p_args) {
    if (p_args->channel == g_external_irq0_cfg.channel) {
        sistema = true;  // Activar el sistema
    }
}
```

```
mostrarSystemOn();
```

```
//SYSTEM ON
void mostrarSystemOn() {

        mensaje1[0x00] = 0x40;
        mensaje1[0x01] = 0x53;//S
        mensaje1[0x02] = 0x59;//Y
        mensaje1[0x03] = 0x53;//S
        mensaje1[0x04] = 0x54;//T
        mensaje1[0x05] = 0x45;//E
        mensaje1[0x06] = 0x4D;//M
        mensaje1[0x07] = 0x80;
        mensaje2[0x00] = 0x40;
        mensaje2[0x01] = 0x80;
        mensaje2[0x02] = 0x80;
        mensaje2[0x03] = 0x80;
        mensaje2[0x04] = 0x4F;//O
        mensaje2[0x05] = 0x4E;//N
        mensaje2[0x06] = 0x80;
        mensaje2[0x07] = 0x80;
    write_LCD(fila1, mensaje1);
    write_LCD(fila2, mensaje2);

}
```

SOFTWARE/CODIGO

Ejercicio 2:

El sistema deberá leer continuamente la velocidad del vehículo (emulada con la lectura del potenciómetro conectado al módulo ADC) de manera frecuente, usando un filtro de mediana móvil para evitar valores outliers

SOFTWARE/CODIGO

Ejercicio 2:

```
// Temporizador
void timer_callback(timer_callback_args_t *p_args) {
    if (TIMER_EVENT_CYCLE_END == p_args->event) {

        ADCStartScan();
        ADCWaitConversion();


        uint16_t resultado = ReadADC(ADC_CHANNEL_4);

        actualizarFiltro(resultado);
        uint16_t mediana = calcularMediana();
        int nivel = nivel_velocidad(resultado);
            if (nivel == 4) {
                DisplayLCD(mediana, mensaje1);
                DisplayLCDPeligro(nivel, mensaje2);
            } else {
                mostrarSystemOn();
            }
    }
}
```

```
//MEDIA DEL FILTRO
uint16_t calcularMediana() {
    uint16_t valoresOrdenados[Filtrogeneral];
    memcpy(valoresOrdenados, filtro, sizeof(filtro));
    for (int i = 0; i < Filtrogeneral - 1; i++) {
        for (int j = i + 1; j < Filtrogeneral; j++) {
            if (valoresOrdenados[j] < valoresOrdenados[i]) {
                uint16_t temp = valoresOrdenados[i];
                valoresOrdenados[i] = valoresOrdenados[j];
                valoresOrdenados[j] = temp;
            }
        }
    }
       return valoresOrdenados[Filtrogeneral / 2];
}
```

```
//Actualizar el LCD(potenciometro)
void DisplayLCD(uint16_t value, uint8_t men1[]) {
    char str[100];
    char numero_char;
    unsigned char ValorAsci1, ValorAsci2, ValorAsci3, ValorAsci4, ValorAsci5;

    sprintf(str, "%i", value);

    uart_write(str);


    numero_char = str[0];
    ValorAsci1 = (unsigned char)numero_char;

    numero_char = str[1];
    ValorAsci2 = (unsigned char)numero_char;

    numero_char = str[2];
    ValorAsci3 = (unsigned char)numero_char;

    numero_char = str[3];
    ValorAsci4 = (unsigned char)numero_char;

    numero_char = str[4];
    ValorAsci5 = (unsigned char)numero_char;


    men1[0x00] = 0x40;
    men1[0x01] = ValorAsci1;
    men1[0x02] = ValorAsci2;
    men1[0x03] = ValorAsci3;
    men1[0x04] = ValorAsci4;
    men1[0x05] = ValorAsci5;
    men1[0x06] = 0x80;
    men1[0x07] = 0x80;
}
```

SOFTWARE/CODIGO

Ejercicio 3:

Todos los valores a la salida del filtro serán enviados a la Raspberry Pi por UART. La RPi guardará las lectura en un log (documento csv o txt), junto con un timestamp de la fecha y hora en que se ha recibido el dato.

SOFTWARE/CODIGO

Ejercicio 3:

```python
# uscar en device manager el numero del puerto COM
uart_serial = serial.Serial('/dev/ttyUSB0',115200, timeout=10,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS)
def write_to_csv(data):
    with open('datosrecibidos.txt', 'a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([data, get_current_timestamp()])
def get_current_timestamp():
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")
if uart_serial.is_open:
 while True:
    size = uart_serial.inWaiting()
    if size:
        data = uart_serial.read(size)
 # El dato recibido está en formato de bytes
        received_data_string = data.decode('utf-8')
        print (received_data_string)
        write_to_csv(received_data_string)
        valor_pot = received_data_string
        topic = "pot"
```

```c
void DisplayLCD(uint16_t value, uint8_t men1[]) {
    char str[100];
    char numero_char;
    unsigned char ValorAsci1, ValorAsci2, ValorAsci3, ValorAsci4, ValorAsci5;

    sprintf(str, "%i", value);

    uart_write(str);


    numero_char = str[0];
    ValorAsci1 = (unsigned char)numero_char;

    numero_char = str[1];
    ValorAsci2 = (unsigned char)numero_char;

    numero_char = str[2];
    ValorAsci3 = (unsigned char)numero_char;

    numero_char = str[3];
    ValorAsci4 = (unsigned char)numero_char;

    numero_char = str[4];
    ValorAsci5 = (unsigned char)numero_char;


    men1[0x00] = 0x40;
```

SOFTWARE/CODIGO

Ejercicio 4 y 5:

4. Cuando la velocidad del vehículo sea superior a un umbral de seguridad (que supere el 75% de la velocidad máxima), la EK deberá emitir el comando de freno de emergencia, explicado a continuación.

5. Comando de freno de emergencia: parpadear el LED, y mostrar en la pantalla del display LCD (conectada por I2C) la velocidad actual en la primera línea, y los caracteres “PELIGRO” en la segunda línea. Este comportamiento se mantendrá mientras no baje la velocidad del umbral. a. Al activar el freno, la EK hará sonar un zumbador durante 1 segundo, 1 única vez.

SOFTWARE/CODIGO

Ejercicio 4 y 5:

```
//UMBRALES DE SEGURIDAD
int nivel_velocidad(uint16_t mediana) {
    const uint16_t menor1 = 1000;
    const uint16_t menor2 = 2600;
    const uint16_t menor3 = 8000;

    if (mediana < menor1) {
        return 1; // Nivel de velocidad men
    } else if (mediana < menor2) {
        return 2; // Nivel de velocidad men
    } else if (mediana < menor3) {
        return 3; // Nivel de velocidad men
    } else {
        return 4; // Nivel de velocidad 4 m
    }
}
```

```
// Temporizador
void timer_callback(timer_callback_args_t *p_args) {
    if (TIMER_EVENT_CYCLE_END == p_args->event) {

        ADCStartScan();
        ADCWaitConversion();


        uint16_t resultado = ReadADC(ADC_CHANNEL_4);

        actualizarFiltro(resultado);
        uint16_t mediana = calcularMediana();
        int nivel = nivel_velocidad(resultado);
            if (nivel == 4) {
                DisplayLCD(mediana, mensaje1);
                DisplayLCDPeligro(nivel, mensaje2);
            } else {
                mostrarSystemOn();
            }
    }
}
```

```
//PELIGRO
void DisplayLCDPeligro(int nivel,uint8_t men1[]) {
    if (nivel == 4) {
        blink_led();
        if (bocinaon==0){
        bocina();
        }

        mensaje2[0x01] = 0x50; // P
        mensaje2[0x02] = 0x45; // E
        mensaje2[0x03] = 0x4C; // L
        mensaje2[0x04] = 0x49; // I
        mensaje2[0x05] = 0x47; // G
        mensaje2[0x06] = 0x52; // R
        mensaje2[0x07] = 0x4F; // O

    } else {
        men1[0x00] = 0x20;
        men1[0x01] = 0x20;
        men1[0x02] = 0x20;
        men1[0x03] = 0x20;
        men1[0x04] = 0x20;
        men1[0x05] = 0x20;
        men1[0x06] = 0x80;
        men1[0x07] = 0x80;
    }
}
```

SOFTWARE/CODIGO

Ejercicio 4 y 5:

```
//LED
void blink_led(void) {

        fsp_err_t err;
        //LED ON
        err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_01_PIN_06, BSP_IO_LEVEL_HIGH);
        assert(FSP_SUCCESS == err);

        // Delay para mantener el LED encendido
        R_BSP_SoftwareDelay(500, BSP_DELAY_UNITS_MILLISECONDS);

        //LED OFF
        err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_01_PIN_06, BSP_IO_LEVEL_LOW);
        assert(FSP_SUCCESS == err);

        // Delay antes de volver a llamar a blink_led
        R_BSP_SoftwareDelay(500, BSP_DELAY_UNITS_MILLISECONDS);

}

//BOCINA
void bocina(void){

    R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_01_PIN_13, BSP_IO_LEVEL_HIGH);
    R_BSP_SoftwareDelay(1000, BSP_DELAY_UNITS_MILLISECONDS);
    R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_01_PIN_13, BSP_IO_LEVEL_LOW);
    bocinaon = true;

}
```

SOFTWARE/CODIGO

Ejercicio 6:

Raspberry Pi: Publicar a un broker MQTT un mensaje cada vez que se active el freno de emergencia. Dicho mensaje contenderá a la velocidad actual y un timestamp. El mensaje se enviará una única vez cada vez que se active el freno de emergencia. Podeis usar el Broker de mosquitto que habéis instalado en la RPi en la última práctica.

SOFTWARE/CODIGO

Ejercicio 6:

```
uart_baudrate = 115200 #debe ser = que el configurado en el tx, este caso la EK
valor_pot= 0
# WINDOWS
# uscar en device manager el numero del puerto COM
uart_serial = serial.Serial('COM6',115200, timeout=10,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.SEVENBITS)
def write_to_csv(data):
    with open('C:\\Users\\Hezitzaile\\Desktop\\LOG.txt', 'a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([data, get_current_timestamp()])
def get_current_timestamp():
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
    mensaje= {
        "timestamp": get_current_timestamp(),
        "value": valor_pot,
        }
    mensaje_json= json.dumps(mensaje)
    hostname = "grupo8" #hostname de vuestra RPi
    publish.single(topic=topic, payload=mensaje_json, qos=1, hostname=hostname,keepalive=60)
    print("Pub done")
else:
 print ('no data')
 time.sleep(1)
```

SOFTWARE/CODIGO

Ejercicio 7:

Cuando la velocidad leída baje del umbral de emergencia, el LED dejará de parpadear, y en el display LCD ya no aparecerá el mensaje de PELIGRO ni la velocidad actual. En su lugar, volverá a estar el mensaje “SYSTEM ON”.

SOFTWARE/CODIGO

Ejercicio 7:

```
// Temporizador
void timer_callback(timer_callback_args_t *p_args) {
    if (TIMER_EVENT_CYCLE_END == p_args->event) {

        ADCStartScan();
        ADCWaitConversion();


        uint16_t resultado = ReadADC(ADC_CHANNEL_4);

        actualizarFiltro(resultado);
        uint16_t mediana = calcularMediana();
        int nivel = nivel_velocidad(resultado);
            if (nivel == 4) {
                DisplayLCD(mediana, mensaje1);
                DisplayLCDPeligro(nivel, mensaje2);
            } else {
                mostrarSystemOn();
            }
    }
}
```

SOFTWARE/CODIGO

Ejercicio 8:

Desde un ordenador portátil, se estará ejecutando un cliente MQTT (suscriptor) que estará suscrito al topic anterior y estará continuamente escuchando a las publicaciones. Cuando se reciba un nuevo mensaje, dicho mensaje se visualizará en la terminal.

SOFTWARE/CODIGO

Ejercicio 8:

```python
import paho.mqtt.client as mqtt
import json
topic = "pot"
# Callback que se llama cuando el cliente recibe el CONNACK del servidor
#Restult code 0 significa conexion sin errores
def on_connect(client, userdata, flags, rc):
 print("Connected with result code "+str(rc))
# Nos subscribirmos al topic
 client.subscribe(topic)
#--------------------------------------------
# Callback que se llama "automaticamente" cuando se recibe un mensaje del Publiser.
def on_message(client, userdata, msg):
 msg.payload = msg.payload.decode("utf-8")
 mensaje_recibido = msg.payload
 print("Mensaje recibido "+mensaje_recibido)
 #Deserializamos ej JSON
 mensaje_recibido_json =json.loads(msg.payload )
 timestamp =mensaje_recibido_json["timestamp"]
 print(timestamp)
 var_pot=mensaje_recibido_json["value"]
 print(var_pot)
#--------------------------------------------
# Creamos un cliente MQTT
client = mqtt.Client()
#Definimos los callbacks para conectarnos y subscribirnos al topic
client.on_connect = on_connect
client.on_message = on_message
hostname = "grupo8" #direccion del BROKER. Hostname de mi RPi
client.connect(hostname, 1883, 60) #1883 es el puerto por defecto
client.loop_forever() #Conecion inenterrumpida > el cliente está continuamente escuchando
```