

Memoria

Domotica con Esp 32 y Raspberry pi

Intro:

Tener una casa domotizada era un lujo hasta hace poco o una fantasía de la ciencia ficción, pero hoy en día a 20 de Junio de 2019. Es posible tener nuestro hogar domotizado, por un puñado de dólares.

El entorno domótico:

La idea general de un sistema domótico es tener un centro de control que puede ser:

Una Raspberry pi.

Un dispositivo google home.

Un dispositivo Alexa.

Un dispositivo Apple:

Apple TV.

ipad.

Apple Home Pod.

Un dispositivo NAS.

Una Smart TV con el sistema Android(Normalmente tienen integrado el Google Assistant).

Además deberemos de tener una red de dispositivos Satélites como pueden ser:
sensores, luces, persianas, enchufes inteligentes, dispositivos conectados, etc...

Normalmente los dispositivos conectados suelen ser muy caros.

La idea de esta memoria es construir nuestros propios dispositivos, los cuales sean capaces de funcionar como si fueran dispositivos "Oficiales" de otras marcas, es decir, compatibles con:

Homekit de apple.

Home assistant de Google.

Alexa.

Phillips.

SonOff.

etc...

Memoria

Excusas

Podemos construir una serie de dispositivos “baratos” y controlarlos a través de la Raspberry Pi, mediante el protocolo de comunicaciones MQTT. Así mismo, la propia Rasp, nos hará de puente entre los dispositivos y la plataforma en nube que nosotros queramos. En mi caso Homekit de apple. Porque mi telefono es un iphone y necesito conectarlo con Homekit, para tener acceso a mis dispositivos desde fuera de casa.

Para ello uso el puente HAPNodeJS, que su función es simular que mis dispositivos son dispositivos compatibles con Homekit.

Pero si tuviera que hacerlo para un dispositivo **No** Apple, en vez de usar HAPNodeJS usaría [Home Assistant](#), también conocido como [HASSIO](#).

El software que normalmente instalo en mis dispositivos cliente(ESP8266, Wemos D1 mini), es el software [TASMOTA](#). Ya que es de código abierto, pesa muy poco, es compatible con multitud de dispositivos, soporta programación OTA(Over the Air). y nos da la posibilidad de conectarnos mediante MQTT.

Sin embargo en esta memoria el software que he usado ha sido micropython y un ESP32.

Normalmente no usaría un ESP32, ya que está sobre dimensionado para lo que pretendo hacer, pero lo he hecho porque tenía muchas ganas de probar micropython.

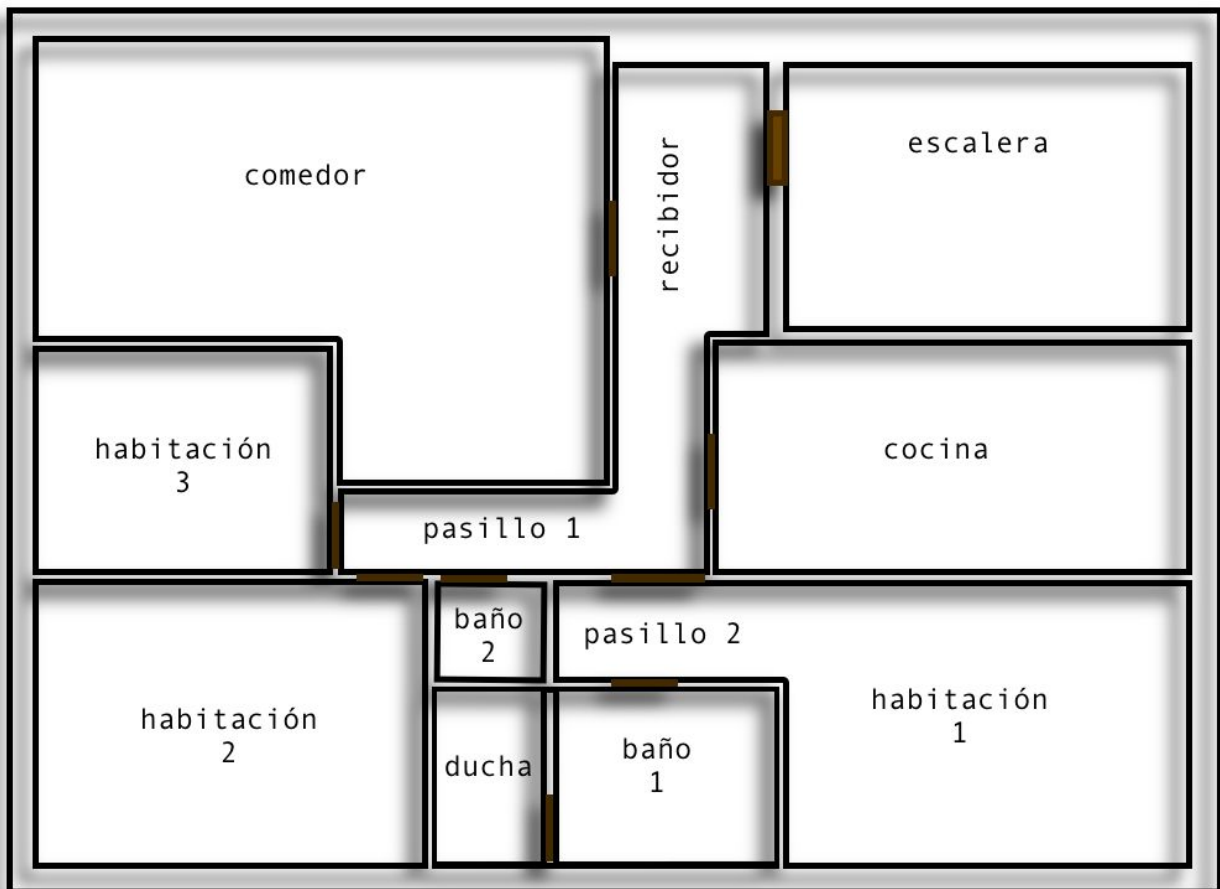
Como ya he dicho el ESP32 está sobredimensionado, ya que es el cadillac de los microcontroladores, el solito es capaz de controlar una micro cámara además de multitud de dispositivos. Yo soy de la opinión que en breve seremos capaces de usar este dispositivo directamente sin necesidad de usar la Rasp como puente.

Memoria

Descripción del proyecto:

Tenemos un piso de 90 metros cuadrados que queremos domotizar. Para ello utilizaremos una Raspberry pi (RPI a partir de ahora), como centro de control. También usaremos los dispositivos inalámbricos ESP8266, ESP32 y Wemos D1 mini.

En el Salón-comedor tenemos ya instalada nuestra RPi con un broker Mosquitto corriendo además de HAP-NodeJS, una librería que nos permite enlazar dispositivos ESPxxxx con HomeKit de Apple. Esto nos aumenta la conectividad desde fuera de casa, ya que el Apple TV nos proporciona acceso gracias a Apple iCloud.



Memoria

Las luces del comedor, ya están domotizadas. El comedor se divide en dos zonas, la del sofá y la de la mesa. Además del enchufe que se encuentra debajo de las llaves de la luz.

La parte que voy a domotizar en este proyecto es la del baño 1. Que a su vez está dividida en dos zonas de luz. La del aseo y la de la ducha. Además mi intención es activar o desactivar el extractor de humos cuando la humedad suba de cierto nivel.

Descripción de funcionamiento:

Nuestro Proyecto se basa en dos circuitos separados, por una parte tenemos la Raspberry Pi, que será nuestro centro de control. Por otra parte un microcontrolador ESP32 que se encarga de tomar las medidas de temperatura y Humedad y mandar los datos a las RASP(Raspberry pi) para que los muestre por pantalla. Además de controlar las luces del baño y enviar los datos a la Rasp o recibir órdenes de la misma para encender o apagar dichas luces.

Al pulsar el botón 1 se encenderá la luz 1, al pulsarlo de nuevo se apagará. Además se mandará a la Rasp el estado de la luz.

Al pulsar el botón 2 se encenderá la luz 2, al pulsarlo de nuevo se apagará. Además se mandará a la Rasp el estado de la luz.

El DHT11 mandará la información de temperatura y humedad a la rasp, para que esta las muestre por el LCD.

La Comunicación:

El Broker:

La comunicación se lleva a cabo mediante el protocolo MQTT de la siguiente manera:

La rasp Hace de centro de comunicaciones mediante MQTT, es decir, está montada como Broker mosquitto ella recibe los mensajes de todos los clientes mqtt y envía las respuestas a cada cliente.

El Cliente:

Cada microcontrolador que pongamos con un cliente mqtt instalado, es un cliente. Un cliente MQTT está suscrito a un tema(topic) en concreto y solo recibe los mensajes de su topic. Así mismo el puede mandar mensajes a un topic concreto para que quien lo reciba haga una acción previamente programada para tal fin.

Es importante que el Topic de Suscripción y el de publicación de un cliente, no sean el mismo, ya que se podrían producir bucles infinitos.

Ejemplos de topics de publicación y de Suscripción:

Publicación: /esp32/baño/L1/salida

Suscripción: /esp32/baño/L1/entrada

Memoria

En la RASP:

En la Raspberry PI, arrancaremos el programa que se encarga de mostrar los datos por el LCD con la orden:

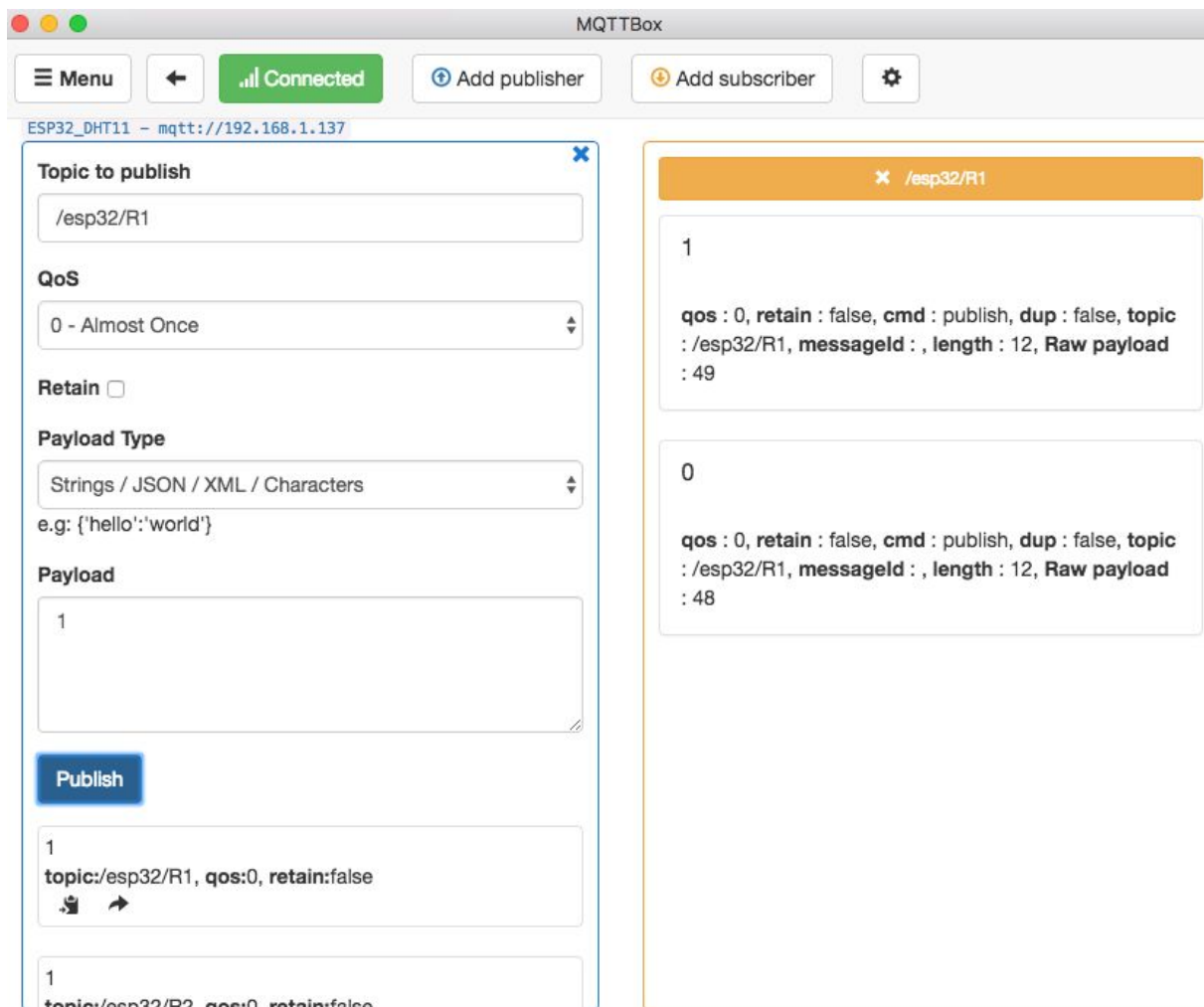
```
python3 funciones.py
```

La Raspberry PI se encargará de monitorear la temperatura y humedad recibidas del ESP32.

Para comprobar que mensajes se envían y se reciben y poder encender las luces a distancia desde la Rasp utilizaremos la herramienta:

[MQTTBOX](#)

En la cual crearemos un cliente con los datos de nuestro servidor mqtt. y después nos suscribimos a el topic que deseemos monitorizar. para enviar o recibir los datos que queramos.



Memoria

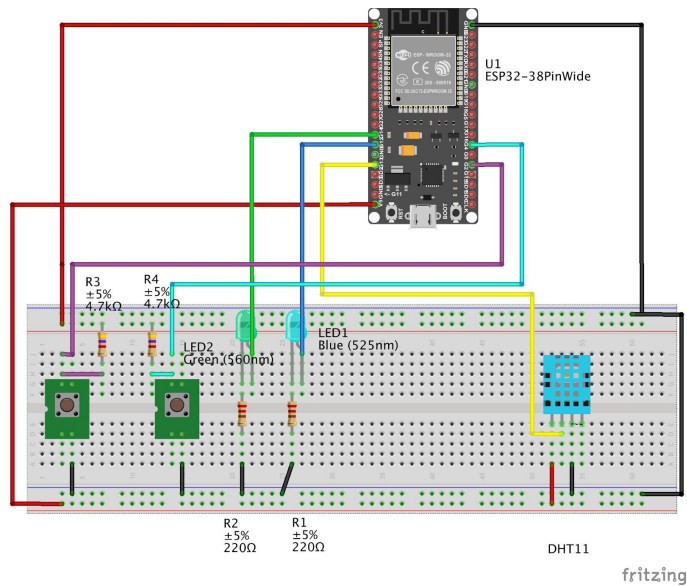
En el ESP32:

Simplemente cargamos los ficheros boot.py, main.py, umqttsimple.py, Globales.py.

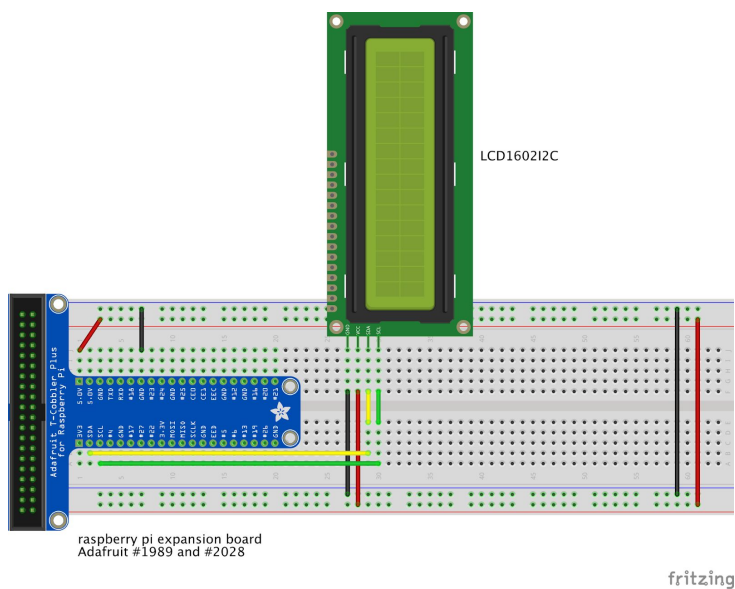
Y arrancamos pulsando el botón de reset de nuestra placa, el proceso será automático.

Esquema de conexión del Proyecto:

ESP32:



Raspberry Pi:



Memoria

EL ESP32

Pins y GPIO

Los pines disponibles son de los siguientes rangos (inclusive): 0-19, 21-23, 25-27, 32-39. Estos corresponden a los números de pin GPIO reales del chip ESP32. Tenga en cuenta que muchas tarjetas de usuario final utilizan su propia numeración de pines adhoc (marcada, por ejemplo, D0, D1, ...). Para la asignación entre las patillas lógicas de la placa y las patillas del chip físico, consulte la documentación de su placa.

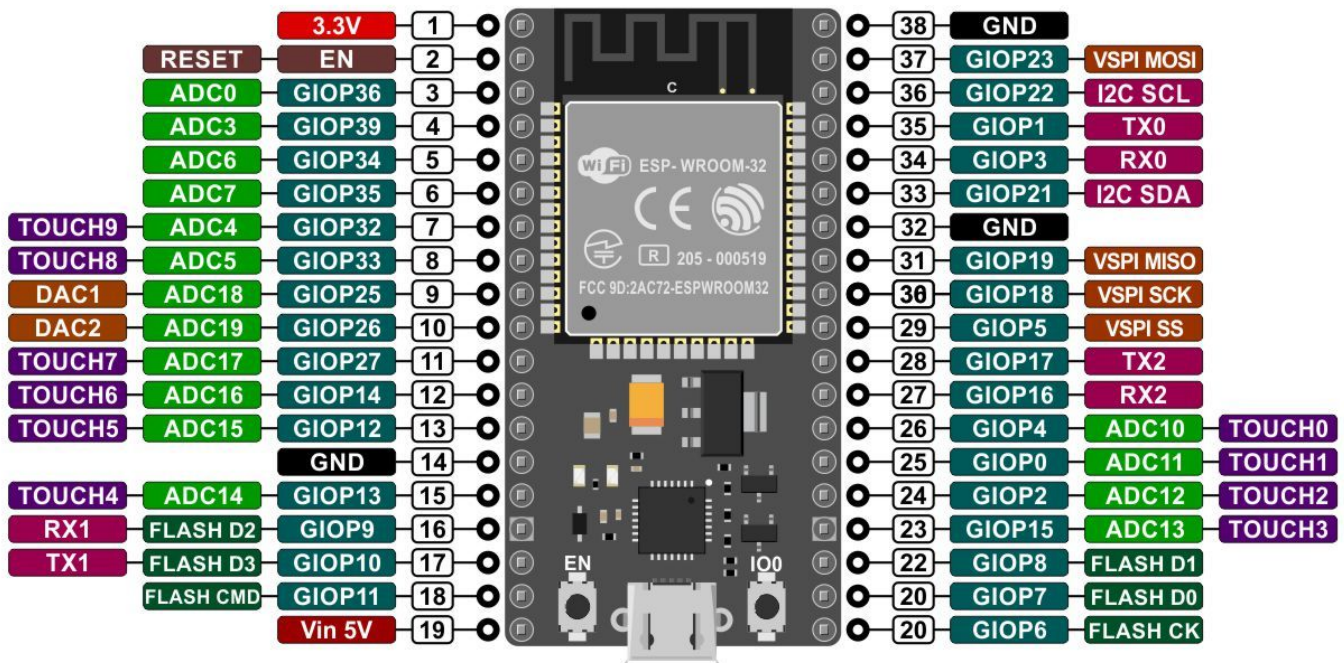
Notas:

Los pines 1 y 3 son REPL UART TX y RX respectivamente

Los pines 6, 7, 8, 11, 16 y 17 se usan para conectar el flash incorporado, y no se recomiendan para otros usos

Los pines 34-39 son solo de entrada y tampoco tienen resistencias internas de levantamiento

El valor de extracción de algunos pines se puede configurar Pin.PULL_HOLD para reducir el consumo de energía durante el sueño profundo.



Memoria

Ejemplo Pin GPIO:

```
from machine import Pin
```

```
p0 = Pin(0, Pin.OUT)    # create output pin on GPIO0
p0.on()                  # set pin to "on" (high) level
p0.off()                 # set pin to "off" (low) level
p0.value(1)              # set pin to on/high
```

```
p2 = Pin(2, Pin.IN)     # create input pin on GPIO2
print(p2.value())        # get value, 0 or 1
```

```
p4 = Pin(4, Pin.IN, Pin.PULL_UP) # enable internal pull-up resistor
p5 = Pin(5, Pin.OUT, value=1) # set pin high on creation
```


Memoria

Preparación del entorno:

Drivers del ESP32:

<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>

Instalar MicroPython

Instalar esptool:

```
sudo pip install esptool
```

Firmware para tarjetas ESP32

- Los siguientes archivos son firmware diario para tarjetas basadas en ESP32, con firmware separado para tarjetas con y sin SPIRAM externo. El firmware no SPIRAM funcionará en cualquier placa, mientras que el firmware habilitado para SPIRAM sólo funcionará en placas con 4MiB de pSRAM externo.
- Programe su placa usando el programa esptool.py, que se encuentra aquí. Si está poniendo MicroPython en su tarjeta por primera vez, primero debe borrar todo el flash utilizando:

```
esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash
```

A partir de ese momento, programe el firmware a partir de la dirección 0x1000:

```
esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 460800 write_flash  
-z 0x1000 esp32-20190125-v1.10.bin
```

Firmware estándar:

[esp32-20190612-v1.11-44-g8b18cfede.bin \(más reciente a fecha 12/06/2019\)](#)

Thonny IDE:

<https://randomnerdtutorials.com/getting-started-thonny-micropython-python-ide-esp32-esp8266/>

Memoria

MicroPython

Referencia:

<http://docs.micropython.org/en/latest/index.html>

OS

<http://docs.microPython.org/en/latest/esp8266/tutorial/filesystem.html?highlight=remove>

Descarga del firmware Micropython para esp 32:

<http://micropython.org/download#esp32>

Ejemplo de configuración ip fija en micropython

```
wlan.ifconfig(( '10.199.160.226', '255.255.255.0', '10.199.160.254',  
'212.0.97.81' ))
```

Ejemplos De Uso de MicroPython:

MicroPython – Getting Started with MQTT on ESP32/ESP8266

<https://randomnerdtutorials.com/micropython-mqtt-esp32-esp8266/>

MicroPython: ESP32 / ESP8266 con servidor web DHT11 / DHT22:

<https://randomnerdtutorials.com/micropython-esp32-esp8266-dht11-dht22-web-server/>

ESP32 MicroPython MQTT Tutorial with Raspberry Pi, DHT-22 & OLED:

https://www.youtube.com/watch?v=_vcQTyLU1WY

<https://www.rototron.info/raspberry-pi-esp32-micropython-mqtt-dht22-tutorial/>

MQTT & MicroPython

<https://www.hackster.io/bucknalla/mqtt-micropython-044e77>

micropython-lib

<https://github.com/micropython/micropython-lib/tree/master/umqtt.simple>

Memoria

ESP32

Requisitos MicroPython en ESP32:

Primero conectaremos el ESP32 a nuestra RPi, mediante el cable USB.

Para instalar en micropython desde nuestra RPi haremos lo siguiente:

```
sudo apt update
```

```
dmesg | grep ttyUSB #Para que la RPi nos monte el ESP32 nos da una salida tipo:
```

```
[5851217.573888] usb 1-1.5: cp210x converter now attached to ttyUSB0
```

Instalar esptool

Lo siguiente que haremos será instalar la herramienta de comunicación con nuestro esp, el esptool:

```
sudo pip3 install esptool
```

Una vez instalado, podemos ver la info de nuestra placa con el comando:

```
esptool.py --port /dev/ttyUSB0 flash_id
```

```
esptool.py v2.6
```

```
Serial port /dev/ttyUSB0#Puerto
```

```
Connecting....._
```

```
Detecting chip type... ESP32 #Tipo de chip
```

```
Chip is ESP32D0WDQ6 (revision 1)
```

```
Features: WiFi, BT, Dual Core, Coding Scheme None
```

```
MAC: 30:ae:a4:8b:b2:08#MAC Adress
```

```
Uploading stub...
```

```
Running stub...
```

```
Stub running...
```

```
Manufacturer: c8
```

```
Device: 4016
```

```
Detected flash size: 4MB#Memoria
```

```
Hard resetting via RTS pin...
```

```
pi@raspberrypi:~$
```

Seguidamente descargamos la imagen del micropython en la RPi:

```
cd /Home/pi/Downloads
```

```
wget
```

```
http://micropython.org/resources/firmware/esp32-20190615-v1.11-45-g14cf91f70.bin
```

Memoria

Y lo cargamos en el ESP32 con:

```
esptool.py --port /dev/ttyUSB0 write_flash 0x010000  
Downloads/esp32-20190615-v1.11-45-g14cf91f70.bin
```

Instalar Remote Shell

A continuación instalaremos un shell remoto para poder comunicarnos con el ESP32 desde nuestra RPi con el comando:

```
sudo pip3 install rshell
```

Conectar por RSHELL

```
dmesg | grep ttyUSB
```

Y nos conectaremos al ESP32 con la instrucción:

```
rshell --buffer-size=30 -p /dev/ttyUSB0
```

el apuntador nos cambia de

```
pi@raspberrypi:~$
```

a:

```
/home/pi>
```

Con la instrucción:

```
boards
```

nos debería mostrar :

```
pyboard @ /dev/ttyUSB0 connected Epoch: 2000 Dirs: /boot.py  
/pyboard/boot.py
```

Este texto nos muestra los archivos cargados en nuestro ESP32.

Ahora vamos a interactuar directamente con el python instalado en vuestro ESP32, para ello haremos un reset con la instrucción:

```
repl
```

el apuntador nos cambia de

```
/home/pi>
```

a:

```
>>>>
```

para salir de python pulsamos CTRL+x

Si estando en...

```
/home/pi>
```

, hacemos...

```
ls
```

 nos mostrará el contenido del usuario pi, pero si hacemos

```
ls /pyboard
```

 nos mostrará el contenido del ESP32 en este caso:

```
boot.py
```

Ahora ya estamos preparados para programar nuestro ESP32.

Memoria

instalar cliente MQTT en ESP32:

Lo primero que debemos hacer es ir a la página:

[micropython-lib/umqtt.simple/umqtt/simple.py](https://micropython-lib.com/micropython/umqtt/simple.py)

Para copiar el código de simple.py.

En /home/pi/Downloads/ , abrimos un nano y pegamos el código que hemos copiado, lo guardamos como: simple.py

A continuación entramos en el modo de comunicación con el ESP32 con la orden:

```
dmesg | grep ttyUSB  
rshell --buffer-size=30 -p /dev/ttyUSB0
```

Creamos un directorio en la raíz del ESP32 al que llamamos umqtt de la siguiente manera:

```
mkdir /pyboard/umqtt
```

Para ver el directorio creado hacemos:

```
ls /pyboard/umqtt
```

Copiamos el fichero simple.py en la carpeta umqtt de esta forma:

```
cp simple.py /pyboard/umqtt
```

Si deseamos ver el código de un fichero instalado en el ESP32 hacemos:

```
cat /pyboard/umqtt/simple.py
```

Activar la red Wlan en el ESP32

Debemos entrar en el remote shell del ESP32 con los pasos de siempre:

```
dmesg | grep ttyUSB  
rshell --buffer-size=30 -p /dev/ttyUSB0  
/home/pi>
```

Ahora entramos en la consola python:

```
repl  
>>>>
```

A continuación hacemos los siguientes pasos:

```
import network  
wlan = network.WLAN(network.STA_IF)  
wlan.active(True)  
wlan.connect('Cisco03222', '')  
wlan.ifconfig(('10.199.160.226', '255.255.255.0', '10.199.160.254', '212.0.97.  
81'))  
if wlan.isconnected():  
    print('estas conectado')
```

Memoria

RaspberryPi

Instalar NODEJS:

visita :

<http://nodejs.org/dist/latest-v9.x/>

para la última versión disponible.... (y sustituye la dirección en el wget de debajo) y el número de versión en las siguientes líneas...

```
wget https://nodejs.org/dist/latest-v9.x/node-v9.11.2-linux-armv7l.tar.gz
```

```
tar -xvf node-v9.11.2-linux-armv7l.tar.gz
```

```
cd node-v9.11.2-linux-armv7l
```

```
sudo cp -R * /usr/local
```

```
sudo npm install -g npm
```

```
sudo npm install -g node-gyp
```

Memoria

MQTT

Instalación en Raspberry pi

requisitos previos: NodeJS

Versión 1 Instalación Automática:

Enlace:

<https://www.rototron.info/raspberry-pi-esp32-micropython-mqtt-dht22-tutorial/>

```
sudo apt-get install mosquitto mosquitto-clients
```

Versión 2 Instalación Manual:

Enlace:

<https://lisergio.wordpress.com/instalacion-homekit-server-y-accesorios/>

```
ssh pi@(ip de vuestra RaspberryPi)
```

```
sudo apt-get update
```

```
sudo apt-get install git-core libnss-mdns libavahi-compat-libdnssd-dev -y
```

Instalar MOSQUITTO (MQTT Broker):

```
cd /
```

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
```

```
sudo apt-key add mosquitto-repo.gpg.key
```

```
cd /etc/apt/sources.list.d/
```

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
```

```
apt-get update
```

```
sudo apt-get install mosquitto
```

```
apt-get install mosquitto-clients
```

Ahora sería el momento de instalar MotionEye para añadir el accesorio cámara ...
(si después de hacer todo el proceso el servidor da algún error y no arranca, prueba a repetir el proceso pero sin instalar esto, podría no ser compatible)

```
cd /
```

```
wget goo.gl/dEx8mY
```

```
sh dEx8mY
```

Memoria

HAP-NodeJS:

Instalación Automática:

Página de referencia:

<https://www.youtube.com/watch?v=3RmuXn8eS9s&feature=youtu.be>

Instrucciones:

en el usuario pi:

```
curl -sSL goo.gl/k8QMGM | bash
```

Instalación manual:

Página de referencia:

<https://lisergio.wordpress.com/instalacion-homekit-server-y-accesorios/>

```
sudo git clone https://github.com/KhaosT/HAP-NodeJS.git
```

```
cd HAP-NodeJS/
```

```
sudo su
```

```
npm rebuild
```

```
npm install node-persist --unsafe-perm
```

```
npm install debug
```

```
npm install ed25519 -unsafe-perm
```

```
npm install mqtt -unsafe-perm
```

```
npm install mdns -unsafe-perm
```

```
npm install srp -unsafe-perm
```

```
npm install curve25519-n -unsafe-perm
```

```
npm install ip -unsafe-perm
```

```
npm install fast-srp-hap
```

```
npm install buffer-shims sudo
```

```
sudo npm install curve25519-n2
```

```
sudo npm install decimal.js --save
```

```
npm install node-persist -unsafe-perm
```

```
npm install bonjour-hap
```

```
npm install ed25519-hap -unsafe-perm
```

Ahora que tenemos todos los paquetes instalados, es el momento de probar si funciona el servidor Core.js y CameraCore.js...

Probar la Instalación:

```
cd /home/pi/HAP-NodeJS
```

```
ls
```

```
sudo node Core.js
```


Memoria

El Código:

Para El ESP32

Programas:

boot.py

```
from umqtt.simple import MQTTClient
import time
import ubinascii
import machine
import micropython
import network
import esp
esp.osdebug(None)
import gc
gc.collect()

ssid = 'TU SIDD'# THEJAXCO
password = 'TU PASSWORD'
mqtt_server = 'IP DE LA RASPBERRY'
estIP='IP FIJA '
estMasc='MASCARA'#'255.255.255.0'
estRou='IP DE LA PUERTA DE ENLACE'#'10.199.160.254'
estDns='DNS'#'212.0.97.81'

client_id = ubinascii.hexlify(machine.unique_id())
topic_sub = b'notification'
topic_pub = b'hello'

last_message = 0
message_interval = 5
counter = 0

def conexion():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid,password)
```

Memoria

```
    if estIP != '':
        #print('estas Aquí')
        wlan.ifconfig((estIP, estMasc, estRou, estDns))
        while wlan.isconnected() == False:
            pass

        print('Connection successful')
        print(wlan.ifconfig())
        print('IDCliente:', client_id)

conexion()
```

Memoria

main.py

```
from umqtt.simple import MQTTClient
from machine import Pin
import dht
import time
import Globales

SERVER = 'IP DEL BROKER' # MQTT Server Address ('192.168.1.22')
CLIENT_ID = 'ESP32_DHT11_Sensor'
TOPIC = b'/lababo/dht11'
TTOPIC = b'/esp32/temperature'
HTOPIC = b'/esp32/humidity'
LTOPIC = b'/esp32/'
L1TOPIC = b'/esp32/R1'
L2TOPIC = b'/esp32/R2'
#print('LTOPIC:', LTOPIC)
#print('L2TOPIC:', L2TOPIC)
P1 = Pin(2, Pin.IN, Pin.PULL_UP)
P2 = Pin(4, Pin.IN, Pin.PULL_UP)
L1 = Pin(12, Pin.OUT)
L2 = Pin(14, Pin.OUT)
Est = Pin(27, Pin.OUT)
Globales.L1Estado = False
Globales.L2Estado = False
Globales.bolo1=False
Globales.bolo2=False

def sub_cb(topic, msg):
    text = str(topic)
    text = text[10:11]
    msg = str(msg)
    msg = msg[2:3]

    if text == '1':
        if msg == '1':
            print('Recibido', msg, text, Globales.bolo1)
            Globales.bolo1 = False
        else:
            print('Recibido', msg, text, Globales.bolo1)
```

Memoria

```
Globales.bolo1 = True
Pin(12).value(int(msg))
Globales.L1Estado = Globales.bolo1
print('enviado',Globales.L1Estado)
time.sleep(1)
else:
    if msg == '1':
        print('Recibido',msg,text,Globales.bolo2)
        Globales.bolo2 = False
    else:
        print('Recibido',msg,text,Globales.bolo2)
        Globales.bolo = True
        Pin(14).value(int(msg))
        Globales.L2Estado = Globales.bolo2
        print('enviado',Globales.L2Estado)
        time.sleep(1)

client = MQTTClient(CLIENT_ID, SERVER)#MQTTClient(CLIENT_ID,
SERVER,1883,USER,PASSWORD)
client.set_callback(sub_cb)
client.connect()
client.subscribe(L2TOPIC)
client.subscribe(L1TOPIC)
# Conexión a MQTT broker
sensor = dht.DHT11(Pin(13))

def lecturas():
    sensor.measure() # medición sensor
    t = sensor.temperature()
    h = sensor.humidity()
    if isinstance(t,int) and isinstance(h,int): # confirmacion de
resultados enteros
        msg = ('{0:3.1f},{1:3.1f}'.format(t, h))
        tm = (b'{}'.format(t))
        hm = (b'{}'.format(h))
        client.publish(TOPIC,msg)
        client.publish(TTOPIC,tm) # publicamos en MQTT el topic, mensaje
        client.publish(HTOPIC,hm) # publicamos en MQTT el topic, mensaje
        print(TOPIC,msg)
```

Memoria

```
    else:
        print('Lectura Invalida del sensor.')
        #client.disconnect()
        time.sleep(0.7)

def cambiaestado(estado,luz):
    txt=''
    try:
        if luz == Pin(12):
            topic =LTOPIC+'L1'
        else:
            topic =LTOPIC+'L2'
        if estado:
            luz.value(1)
            txt='On'
        else:
            luz.value(0)
            txt='Off'
        client.publish(topic,txt)
        #print(topic,txt)
        return not estado
    except OSError:
        pass

def leebotones():
    uno=[1,1]
    try:
        uno[0]=P1.value()
        uno[1]=P2.value()
        time.sleep(0.15)

        if (uno[0]==0):
            Globales.L1Estado = cambiaestado(Globales.L1Estado,L1)
            print('Globales.L1Estado',Globales.L1Estado)

        if (uno[1]==0):
            Globales.L2Estado = cambiaestado(Globales.L2Estado,L2)
            print(P2.value(),'Globales.L2Estado',Globales.L2Estado)
```

Memoria

```
except OSError:
    pass

while True:
    for i in range(25):
        #print(i)
        if i ==1:
            lecturas()

    else:
        leebotones()
        client.check_msg()
```

Memoria

Librerías:

umqttsimple.py

```
try:
    import usocket as socket
except:
    import socket
import ustruct as struct
from ubinascii import hexlify

class MQTTException(Exception):
    pass

class MQTTClient:

    def __init__(self, client_id, server, port=0, user=None, password=None,
        keepalive=0,
        ssl=False, ssl_params={}):
        if port == 0:
            port = 8883 if ssl else 1883
        self.client_id = client_id
        self.sock = None
        self.server = server
        self.port = port
        self.ssl = ssl
        self.ssl_params = ssl_params
        self.pid = 0
        self.cb = None
        self.user = user
        self.pswd = password
        self.keepalive = keepalive
        self.lw_topic = None
        self.lw_msg = None
        self.lw_qos = 0
        self.lw_retain = False

    def _send_str(self, s):
```

Memoria

```
self.sock.write(struct.pack("!H", len(s)))
self.sock.write(s)
```

```
def _recv_len(self):
    n = 0
    sh = 0
    while 1:
        b = self.sock.read(1)[0]
        n |= (b & 0x7f) << sh
        if not b & 0x80:
            return n
        sh += 7
```

```
def set_callback(self, f):
    self.cb = f
```

```
def set_last_will(self, topic, msg, retain=False, qos=0):
    assert 0 <= qos <= 2
    assert topic
    self.lw_topic = topic
    self.lw_msg = msg
    self.lw_qos = qos
    self.lw_retain = retain
```

```
def connect(self, clean_session=True):
    self.sock = socket.socket()
    addr = socket.getaddrinfo(self.server, self.port)[0][-1]
    self.sock.connect(addr)
    if self.ssl:
        import ssl
        self.sock = ssl.wrap_socket(self.sock, **self.ssl_params)
    premsg = bytearray(b"\x10\0\0\0\0\0")
    msg = bytearray(b"\x04MQTT\x04\0\0\0\0")
```

```
sz = 10 + 2 + len(self.client_id)
msg[6] = clean_session << 1
if self.user is not None:
    sz += 2 + len(self.user) + 2 + len(self.pswd)
    msg[6] |= 0xC0
if self.keepalive:
```


Memoria

```
        assert self.keepalive < 65536
        msg[7] |= self.keepalive >> 8
        msg[8] |= self.keepalive & 0x00FF
        if self.lw_topic:
            sz += 2 + len(self.lw_topic) + 2 + len(self.lw_msg)
            msg[6] |= 0x4 | (self.lw_qos & 0x1) << 3 | (self.lw_qos & 0x2)
<< 3
            msg[6] |= self.lw_retain << 5

        i = 1
        while sz > 0x7f:
            premsg[i] = (sz & 0x7f) | 0x80
            sz >>= 7
            i += 1
            premsg[i] = sz

        self.sock.write(premsg, i + 2)
        self.sock.write(msg)
        #print(hex(len(msg)), hexlify(msg, ":"))
        self._send_str(self.client_id)
        if self.lw_topic:
            self._send_str(self.lw_topic)
            self._send_str(self.lw_msg)
        if self.user is not None:
            self._send_str(self.user)
            self._send_str(self.pswd)
        resp = self.sock.read(4)
        assert resp[0] == 0x20 and resp[1] == 0x02
        if resp[3] != 0:
            raise MQTTException(resp[3])
        return resp[2] & 1

    def disconnect(self):
        self.sock.write(b"\xe0\0")
        self.sock.close()

    def ping(self):
        self.sock.write(b"\xc0\0")

    def publish(self, topic, msg, retain=False, qos=0):
```

Memoria

```
pkt = bytearray(b"\x30\0\0\0")
pkt[0] |= qos << 1 | retain
sz = 2 + len(topic) + len(msg)
if qos > 0:
    sz += 2
assert sz < 2097152
i = 1
while sz > 0x7f:
    pkt[i] = (sz & 0x7f) | 0x80
    sz >>= 7
    i += 1
pkt[i] = sz
#print(hex(len(pkt)), hexlify(pkt, ":"))
self.sock.write(pkt, i + 1)
self._send_str(topic)
if qos > 0:
    self.pid += 1
    pid = self.pid
    struct.pack_into("!H", pkt, 0, pid)
    self.sock.write(pkt, 2)
    self.sock.write(msg)
    if qos == 1:
        while 1:
            op = self.wait_msg()
            if op == 0x40:
                sz = self.sock.read(1)
                assert sz == b"\x02"
                rcv_pid = self.sock.read(2)
                rcv_pid = rcv_pid[0] << 8 | rcv_pid[1]
                if pid == rcv_pid:
                    return
            elif qos == 2:
                assert 0

def subscribe(self, topic, qos=0):
    assert self.cb is not None, "Subscribe callback is not set"
    pkt = bytearray(b"\x82\0\0\0")
    self.pid += 1
    struct.pack_into("!BH", pkt, 1, 2 + 2 + len(topic) + 1, self.pid)
    #print(hex(len(pkt)), hexlify(pkt, ":"))
```

Memoria

```
self.sock.write(pkt)
self._send_str(topic)
self.sock.write(qos.to_bytes(1, "little"))
while 1:
    op = self.wait_msg()
    if op == 0x90:
        resp = self.sock.read(4)
        #print(resp)
        assert resp[1] == pkt[2] and resp[2] == pkt[3]
        if resp[3] == 0x80:
            raise MQTTException(resp[3])
        return

# Wait for a single incoming MQTT message and process it.
# Subscribed messages are delivered to a callback previously
# set by .set_callback() method. Other (internal) MQTT
# messages processed internally.
def wait_msg(self):
    res = self.sock.read(1)
    self.sock.setblocking(True)
    if res is None:
        return None
    if res == b"":
        raise OSError(-1)
    if res == b"\xd0": # PINGRESP
        sz = self.sock.read(1)[0]
        assert sz == 0
        return None
    op = res[0]
    if op & 0xf0 != 0x30:
        return op
    sz = self._recv_len()
    topic_len = self.sock.read(2)
    topic_len = (topic_len[0] << 8) | topic_len[1]
    topic = self.sock.read(topic_len)
    sz -= topic_len + 2
    if op & 6:
        pid = self.sock.read(2)
        pid = pid[0] << 8 | pid[1]
        sz -= 2
```

Memoria

```
msg = self.sock.read(sz)
self.cb(topic, msg)
if op & 6 == 2:
    pkt = bytearray(b"\x40\x02\0\0")
    struct.pack_into("!H", pkt, 2, pid)
    self.sock.write(pkt)
elif op & 6 == 4:
    assert 0
```

```
# Checks whether a pending message from server is available.
# If not, returns immediately with None. Otherwise, does
# the same processing as wait_msg.
def check_msg(self):
    self.sock.setblocking(False)
    return self.wait_msg()
```

Memoria

Utilidades:

Globales

```
'''Esta es la manera de tener  
variables globales en micropython en el esp32  
'''  
L1Estado = False  
L2Estado = False  
bolo1=False  
bolo2=False
```

Memoria

Para La Raspberry Pi

Programa funciones.py

```
import smbus
import time
import paho.mqtt.client as mqtt

# Define some device parameters
I2C_ADDR = 0x27 # I2C device address, if any error, change this address to 0x27
LCD_WIDTH = 16 # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line
LIN = 0
LCD_BACKLIGHT = 0x08 # On
#LCD_BACKLIGHT = 0x00 # Off

ENABLE = 0b00000100 # Enable bit

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

TXT1=TX2=DAT01=DAT02=''

#Open I2C interface
#bus = smbus.SMBus(0) # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd_init():
    # Initialise display
    lcd_byte(0x33,LCD_CMD) # 110011 Initialise
```

Memoria

```
lcd_byte(0x32,LCD_CMD) # 110010 Initialise
lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
time.sleep(E_DELAY)

def lcd_byte(bits, mode):
    # Send byte to data pins
    # bits = the data
    # mode = 1 for data
    #       0 for command

    bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
    bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

    # High bits
    bus.write_byte(I2C_ADDR, bits_high)
    lcd_toggle_enable(bits_high)

    # Low bits
    bus.write_byte(I2C_ADDR, bits_low)
    lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
    # Toggle enable
    time.sleep(E_DELAY)
    bus.write_byte(I2C_ADDR, (bits | ENABLE))
    time.sleep(E_PULSE)
    bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
    time.sleep(E_DELAY)

def lcd_string(message,line):
    message = message.ljust(LCD_WIDTH, " ")
    lcd_byte(line, LCD_CMD)
    for i in range(LCD_WIDTH):
        lcd_byte(ord(message[i]),LCD_CHR)

def main():
    client = mqtt.Client()
```

Memoria

```
client.on_connect = on_connect
client.on_message = on_message # Specify on_message callback
client.connect('localhost', 1883, 60) # Connect to MQTT broker (also
running on Pi).
lcd_init()
client.loop_forever()

def on_connect(client, userdata, flags, rc):
    #print('Cliente MQTT conectado Errores:{0}'.format(rc))
    client.subscribe("/lababo/dht11")

def on_message(client, userdata, message):
    message = str(message.payload)
    DAT01 = message[2:6]
    DAT02 = message[7:11]
    TXT1='Temperatura:'+ DAT01
    TXT2='Humedad      :'+ DAT02
    lcd_string(TXT1,LCD_LINE_1)
    lcd_string(TXT2,LCD_LINE_2)
    topic = message.topic
    qos = str(message.qos)
    #print(message[2:11])
import smbus
import time
import paho.mqtt.client as mqtt

# Define some device parameters
I2C_ADDR = 0x27 # I2C device address, if any error, change this address to
0x27
LCD_WIDTH = 16 # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line
```


Memoria

```
LIN = 0
LCD_BACKLIGHT = 0x08 # On
#LCD_BACKLIGHT = 0x00 # Off

ENABLE = 0b00000100 # Enable bit

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

TXT1=TXT2=DAT01=DAT02=''

#Open I2C interface
#bus = smbus.SMBus(0) # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd_init():
    # Initialise display
    lcd_byte(0x33,LCD_CMD) # 110011 Initialise
    lcd_byte(0x32,LCD_CMD) # 110010 Initialise
    lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
    lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
    lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
    lcd_byte(0x01,LCD_CMD) # 000001 Clear display
    time.sleep(E_DELAY)

def lcd_byte(bits, mode):
    # Send byte to data pins
    # bits = the data
    # mode = 1 for data
    #      0 for command

    bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
    bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

    # High bits
    bus.write_byte(I2C_ADDR, bits_high)
    lcd_toggle_enable(bits_high)
```

Memoria

```
# Low bits
bus.write_byte(I2C_ADDR, bits_low)
lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
    # Toggle enable
    time.sleep(E_DELAY)
    bus.write_byte(I2C_ADDR, (bits | ENABLE))
    time.sleep(E_PULSE)
    bus.write_byte(I2C_ADDR, (bits & ~ENABLE))
    time.sleep(E_DELAY)

def lcd_string(message, line):
    message = message.ljust(LCD_WIDTH, " ")
    lcd_byte(line, LCD_CMD)
    for i in range(LCD_WIDTH):
        lcd_byte(ord(message[i]), LCD_CHR)

def main():
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message # Specify on_message callback
    client.connect('localhost', 1883, 60) # Connect to MQTT broker (also
running on Pi).
    lcd_init()
    client.loop_forever()

def on_connect(client, userdata, flags, rc):
    client.subscribe("/lababo/dht11")
    print('Cliente MQTT conectado Errores:{0}'.format(rc))

def on_message(client, userdata, message):
    message = str(message.payload)
    DAT01 = message[2:6]
    DAT02 = message[7:11]
    TXT1='Temperatura:'+ DAT01
    TXT2='Humedad:'+ DAT02
    lcd_string(TXT1,LCD_LINE_1)
    lcd_string(TXT2,LCD_LINE_2)
```

Memoria

```
    topic = message.topic
    qos = str(message.qos)
    #print(mensaje[2:11])

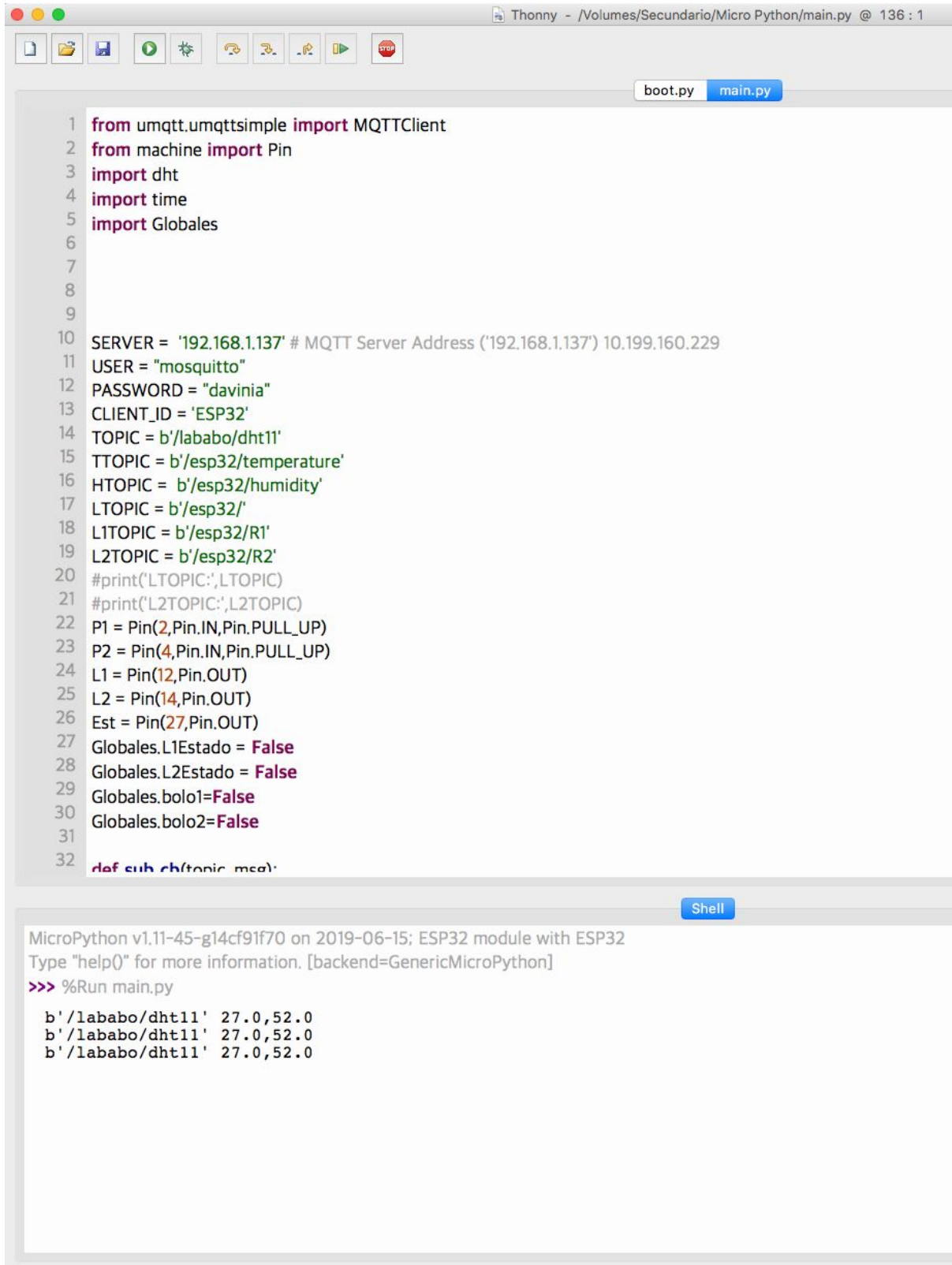
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
    finally:
        lcd_byte(0x01, LCD_CMD)
```

Memoria

SoftWare de Control

[Thonny IDE:](#)

IDE usado para programar el microcontrolador.



```
Thonny - /Volumes/Secundario/Micro Python/main.py @ 136 : 1

1 from umqtt.umqttsimple import MQTTClient
2 from machine import Pin
3 import dht
4 import time
5 import Globales
6
7
8
9
10 SERVER = '192.168.1.137' # MQTT Server Address ('192.168.1.137') 10.199.160.229
11 USER = "mosquitto"
12 PASSWORD = "davinia"
13 CLIENT_ID = 'ESP32'
14 TOPIC = b'/lababo/dht11'
15 TTOPIC = b'/esp32/temperature'
16 HTOPIC = b'/esp32/humidity'
17 LTOPIC = b'/esp32/'
18 L1TOPIC = b'/esp32/R1'
19 L2TOPIC = b'/esp32/R2'
20 #print('LTOPIC:', LTOPIC)
21 #print('L2TOPIC:', L2TOPIC)
22 P1 = Pin(2, Pin.IN, Pin.PULL_UP)
23 P2 = Pin(4, Pin.IN, Pin.PULL_UP)
24 L1 = Pin(12, Pin.OUT)
25 L2 = Pin(14, Pin.OUT)
26 Est = Pin(27, Pin.OUT)
27 Globales.L1Estado = False
28 Globales.L2Estado = False
29 Globales.bolo1=False
30 Globales.bolo2=False
31
32 def sub_cb(topic, msg):

MicroPython v1.11-45-g14cf91f70 on 2019-06-15; ESP32 module with ESP32
Type "help()" for more information. [backend=GenericMicroPython]
>>> %Run main.py
b'/lababo/dht11' 27.0,52.0
b'/lababo/dht11' 27.0,52.0
b'/lababo/dht11' 27.0,52.0
```

Memoria

[MQTTBOX](#)

Software Usado para comprobar el funcionamiento de la red MQTT.

The screenshot displays the MQTTBox application interface. At the top, there's a status bar with a 'Menu' button, a 'Connected' indicator, and buttons for 'Add publisher' and 'Add subscriber'. Below this, the connection details are shown: 'ESP32_DHT11 - mqtt://192.168.1.137'.

The main interface is divided into two main sections. On the left, the 'Topic to publish' section is active, showing a text input field with '/esp32/R1'. Below this, the 'QoS' is set to '0 - Almost Once', and the 'Retain' checkbox is unchecked. The 'Payload Type' is set to 'Strings / JSON / XML / Characters', with an example payload 'e.g: {'hello':'world'}' shown. The 'Payload' text area contains the number '1'. A 'Publish' button is located below the payload area. At the bottom of this section, there's a preview of the published message: '1' followed by 'topic:/esp32/R1, qos:0, retain:false'.

On the right, there are two subscriber panels. The top panel is for the topic '/esp32/R1' and shows a message with '1' in a box. Below the message box, the details are listed: 'qos : 0, retain : false, cmd : publish, dup : false, topic : /esp32/R1, messageld : , length : 12, Raw payload : 49'. The bottom panel is for the topic '/esp32/R2' and shows a message with '1' in a box. Below the message box, the details are listed: 'qos : 0, retain : false, cmd : publish, dup : false, topic : /esp32/R2, messageld : , length : 12, Raw payload : 49'.

Memoria

OTROS SISTEMAS:

La Familia ESP8266:

Existen otros dispositivos aparte del esp32 igualmente programables, normalmente con el IDE Arduino, aunque también pueden programarse con Atom y platformIO, además de otros IDEs.

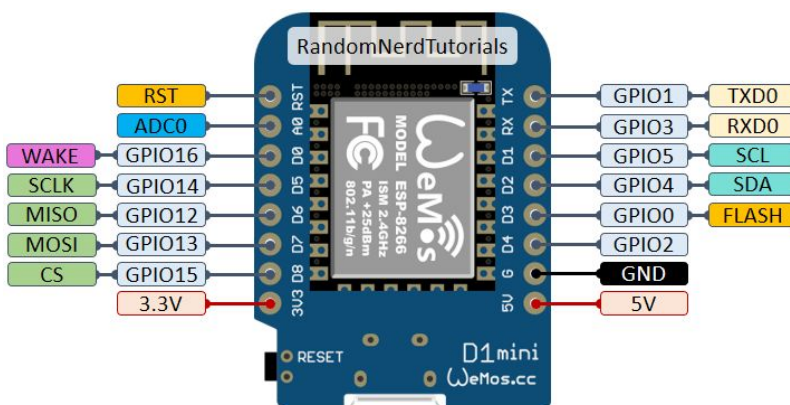
ESP01: el más pequeño de la familia, se puede usar como interruptor básico para dar órdenes a MQTT.

Normalmente lo que yo suelo hacer es instalarles el software de código abierto [TASMOTA\(WIKI\)](#). Que nos da un entorno de uso web, en el cual podemos programar las entradas y salidas como deseemos, es muy completo y compatible con una amplia gama de dispositivos.

Familia ESP8266



Node MCU Wemos D1 mini



El Wemos D1 mini, es una placa ideal para domotizar nuestra casa. Es barata, pequeña, con bastantes pines I/O con interface i2c, serie, una entrada analógica y wifi. su precio ronda de los 2,50€ a los 7€. y podemos instalarle [Tasmota](#).

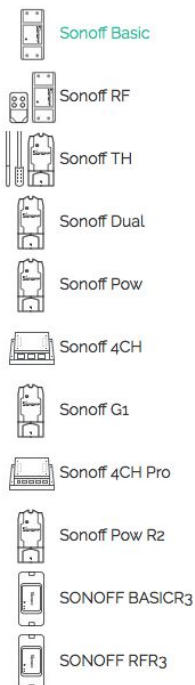
Memoria

Familia SonOFF

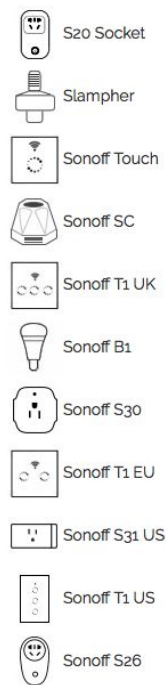
Los dispositivos SonOff se caracterizan porque pese a ser dispositivos ya fabricados como domóticos, se les puede instalar el software tasmota fácilmente. Además son bastante baratos y muy fáciles de conseguir. Normalmente vienen con un ESP8266 en su interior y ya tienen la fuente de alimentación incluida, lo que los hace más atractivos, ya que por el mismo precio no tendríamos que hacer un gasto doble en esp8266 + fuente.

Además hay de todo tipo no solo wifi tambien RF e infrarrojos.

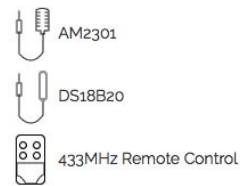
SONOFF



RESIDENTIAL



ACCESSORIES



APPLIANCE



Memoria

INFO

Datos de la clase:

Ip: 10.199.160.x x=229

mascara: 255.255.255.0

P.E.: 10.199.160.254

DNS: 212.0.97.81

212.0.97.82

Libros

enlaces de interes:

<https://www.flexbot.es/monitorizar-temperatura-humedad-movil-blynk/>

<https://randomnerdtutorials.com/micropython-esp32-esp8266-dht11-dht22-web-server/>

<https://towardsdatascience.com/python-webserver-with-flask-and-raspberry-pi-398423cc6f5d>

<http://fpaez.com/sensor-dht11-de-temperatura-y-humedad/>

<https://randomnerdtutorials.com/micropython-esp32-esp8266-dht11-dht22-web-server/>

Memoria

AGRADECIMIENTOS

Luis del Valle:
programarfcil.com

Rui Santos:
randomnerdtutorials.com

Fermín Ortega
www.flexbot.es

<https://www.home-assistant.io>

QuickPi
<https://www.youtube.com/channel/UC3AGxC2YOkov8plchTHRqQw>

<http://micropython.org>

www.silabs.com

www.espressif.com

rdagger68
www.rototron.info
<https://www.youtube.com/channel/UCp2rS5TxRt6W8fieAk74blw>

micropython-lib

University of Tartu
[Thonny IDE](http://Thonny.IDE)

lisergio
lisergio.wordpress.com

MQTTBOX

Theo Arends
Sonoff-Tasmota

SonOFF

Home Assistant
www.home-assistant.io
[DrZzs](https://www.youtube.com/watch?v=sVml02kP3DU&list=WL&index=31&t=577s)
<https://www.youtube.com/watch?v=sVml02kP3DU&list=WL&index=31&t=577s>

bitluni's lab
https://www.youtube.com/channel/UCp_5PO66faM4dBFbFFBdPSQ

Luis llamas
www.luisllamas.es