

Universidad de Sevilla

Memorias Practicas

Laboratorio de Desarrollo de Hardware

Francisco Javier Solís Franco
30-11-2016

Índice

Memoria 3: Plataforma Raspberry Pi.	2
Objetivos.....	2
Introducción.....	2
Desarrollo de la practica.....	2
Control de led's con python.	3
Web Framework con led's.....	4
Web Framework con sensor de temperatura.	7
Web Framework con temperatura y relé.	10
Conclusión.....	14

Memoria 3: Plataforma Raspberry Pi.

Objetivos

Conocer la plataforma Raspberry Pi, así como sus usos y posibles aplicaciones superando estos objetivos mínimos:

- Preparar la plataforma la instalación de diferentes S.O.
- Comprobar el funcionamiento de la placa Raspberry Pi 3.
- Desarrollar ejemplos de utilización de los pines de expansión GPIO.
- Instalar un servidor web capaz de ejecutar código Python.

Introducción

¿Qué es Raspberry Pi?

Es una plataforma de desarrollo denominada "SBC" (Single Board Computer), computador de placa simple, de bajo coste desarrollado por la fundación Raspberry Pi, cuyo software que se usa con la placa es software libre (Raspbian, versión modificada de debian).



Esta plataforma cuenta con salida de video (HDMI o RCA, según versión), puertos USB, conector para display y para cámara, salida de audio Jack de 3,5mm, zócalo para microSD (microSD con el S.O), los pines de expansión GPIO (similares a los de la plataforma arduino), conector de ethernet RJ45 y como fuente de alimentación tenemos un conector micro USB.

En el último modelo (Rpi 3) ya contamos con chip bluetooth y wifi.

Con los SBC podemos controlar otros dispositivos hardware.

Desarrollo de la practica

Para comenzar las practicas tenemos que cargar el sistema operativo en la tarjeta microSD, con lo que nos basaremos en el siguiente tutorial:

<http://www.raspberrypi.org/documentation/installation/installing-images/linux.md>.(Tendremos que descargar la imagen ISO del sistema previamente)

Que básicamente consisten en ejecutar los siguientes comandos es la consola:

Lo primero es conectar la microSD al lector de tarjetas y ejecutar el comando “df -h” que nos mostrará todos los dispositivos montados, localizaremos la tarjeta y nos quedaremos con el nombre.

El segundo paso es desmontar la tarjeta que lo realizaremos con el comando: “umount /dev/<nombre sd>”.

Una vez desmontada la tarjeta procederemos a cargar en ella el sistema, para ello ejecutaremos: “dd bs=4M if=<nombre_imagen_SO>.img of=/dev/<nombre_sd>”.

De esta forma ya tenemos la tarjeta preparada, ahora solo nos queda conectar a la Rpi un teclado y ratón, un HDMI-DVI de la Rpi al monitor, un cable de Ethernet y un USB para la alimentación de la Rpi.

Cuando haya arranco el sistema operativo configuraremos algunas opciones como distribución de teclado, zona horario y fecha, etc... tras esto tendremos que configurar la red, la cual tendrá esta configuración:

-ip manual

-ip: 10.1.15.XX (XX corresponde la numero de PC al que hemos quitado teclado, ratón, etc...)

-Puerta de enlace: 10.1.15.78

-mascara: 255.255.254.0

-dns: 8.8.8.8

Control de led's con python.

El primer programa que haremos será controlar el parpadeo de unos leds conectados al GPIO de la Rpi.

Comenzaremos siguiente este tutorial: <https://geekytheory.com/tutorial-raspberry-pi-gpio-parte-2-control-de-leds-con-python/> .

Este tutorial consiste en un programa sencillo que nos permite encender y apagar 2 led's de forma alterna 15 veces cada uno. Por lo que conectaremos 2 led's al GPIO en los pines adecuados, tendremos que mirar que pines son GND, VCC y pin para salida o entrada para conectar los led's de forma adecuada.

Tras conectarlos de forma correcta realizaremos el programa que será el siguiente.

Código Python:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT) ## GPIO 17 como salida
GPIO.setup(27, GPIO.OUT) ## GPIO 27 como salida

def blink():
    print "Ejecucion iniciada..."
    iteracion = 0
    while iteracion < 30: ## Segundos que durara la funcion
        GPIO.output(17, True) ## Enciende el 17
```

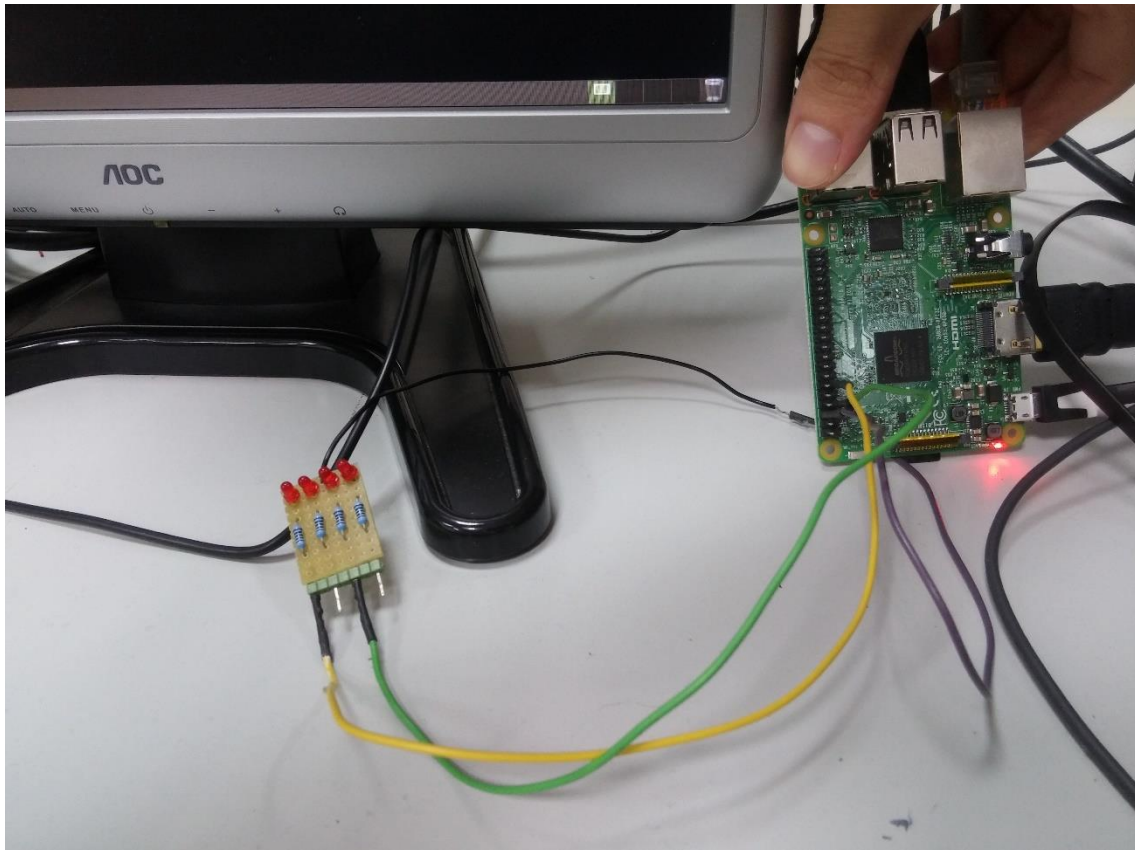
```

GPIO.output(27, False) ## Apago el 27
time.sleep(1) ## Esperamos 1 segundo
GPIO.output(17, False) ## Apago el 17
GPIO.output(27, True) ## Enciendo el 27
time.sleep(1) ## Esperamos 1 segundo
iteracion = iteracion + 2 ## Sumo 2 porque he hecho
dos parpadeos
print "Ejecucion finalizada"
GPIO.cleanup() ## Hago una limpieza de los GPIO

blink() ## Hago la llamada a la funcion blink

```

Obteniendo este resultado:



Vídeo del funcionamiento: <https://youtu.be/G-aR6jXcd5o>

Web Framework con led's.

Lo siguiente que haremos será comprobar un aspecto interesante de estas placas tipo SBC, que es el control de otros dispositivos hardware, que podemos hacerlo a través de internet median un servidor web.

Comenzaremos por realizar la parte del servidor web siguiendo un tutorial:

<http://mattrichardson.com/Raspberry-Pi-Flask/index.html>

Pero nos limitaremos a controlar los led's conectados al GPIO como en el apartado anterior.

Para ello tendremos que instalar flask, un framework que nos permite ejecutar código Python y lo visualizaremos y controlaremos todo median un web. Una vez instalado escribiremos el código Python y crearemos un subdirectorio llamado "templates" donde estará el html que usaremos como interfaz de control.

Código Python:

```
import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

# Create a dictionary called pins to store the pin number, name, and
pin state:
pins = {
    24 : {'name' : 'LED1', 'state' : GPIO.LOW},
    25 : {'name' : 'LED2', 'state' : GPIO.LOW}
}

# Set each pin as an output and make it low:
for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
    # For each pin, read the pin state and store it in the pins
    dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    # Put the pin dictionary into the template data dictionary:
    templateData = {
        'pins' : pins
    }
    # Pass the template data into the template main.html and return it
    to the user
    return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the
pin number and action in it:
@app.route("/<changePin>/<action>")
def action(changePin, action):
    # Convert the pin from the URL into an integer:
    changePin = int(changePin)
    # Get the device name for the pin being changed:
    deviceName = pins[changePin]['name']
    # If the action part of the URL is "on," execute the code indented
    below:
    if action == "on":
        # Set the pin high:
        GPIO.output(changePin, GPIO.HIGH)
        # Save the status message to be passed into the template:
        message = "Turned " + deviceName + " on."
    if action == "off":
        GPIO.output(changePin, GPIO.LOW)
        message = "Turned " + deviceName + " off."
    if action == "toggle":
```

```

        # Read the pin and set it to whatever it isn't (that is, toggle
it):
        GPIO.output(changePin, not GPIO.input(changePin))
        message = "Toggled " + deviceName + "."

    # For each pin, read the pin state and store it in the pins
dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    # Along with the pin dictionary, put the message into the template
data dictionary:
    templateData = {
        'message' : message,
        'pins' : pins
    }

    return render_template('main.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)

```

Código HTML:

```

<!DOCTYPE html>
<head>
    <title>Current Status</title>
</head>

<body>
    <h1>Device Listing and Status</h1>

    {% for pin in pins %}
    <p>The {{ pins[pin].name }}
    {% if pins[pin].state == true %}
        is currently on (<a href="/{{pin}}/off">turn off</a>)
    {% else %}
        is currently off (<a href="/{{pin}}/on">turn on</a>)
    {% endif %}
    </p>
    {% endfor %}

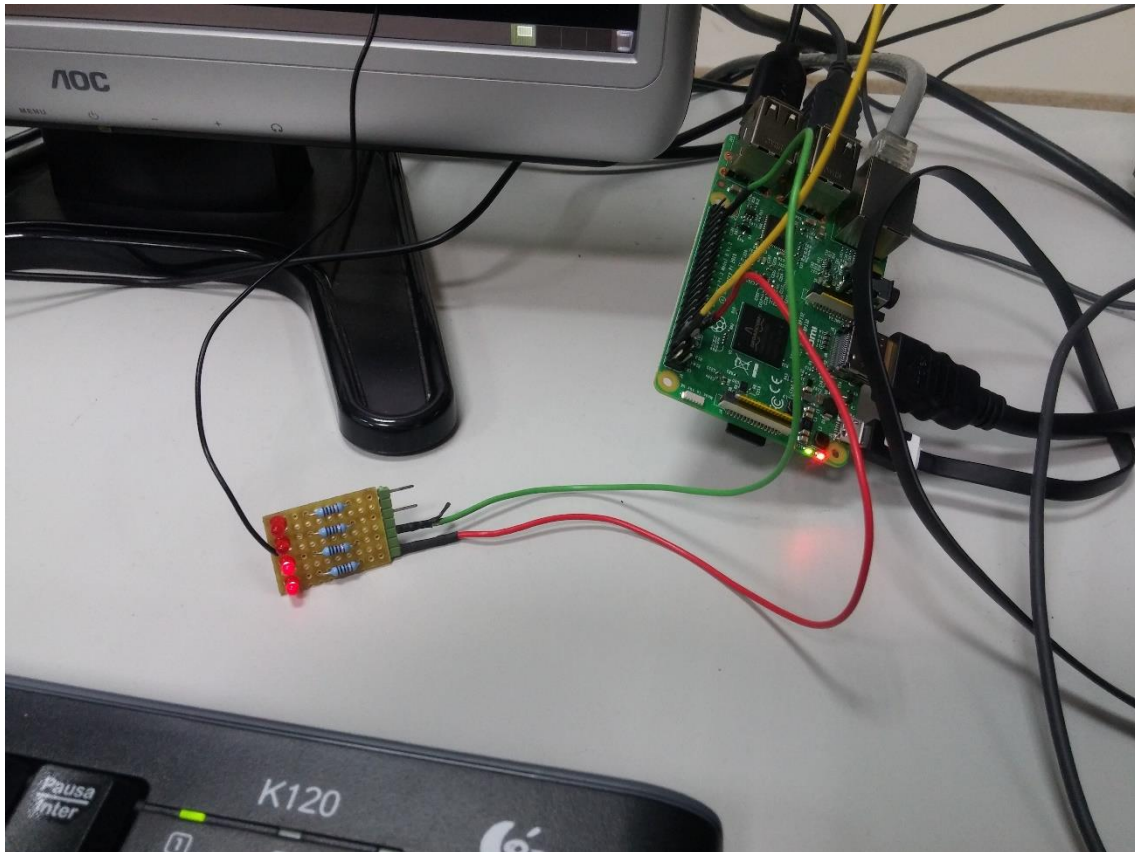
    {% if message %}
    <h2>{{ message }}</h2>
    {% endif %}

</body>
</html>

```

Para comprobar el funcionamiento desde otro equipo abriremos el navegador y pondremos la ip de la Rpi una vez hayamos iniciado el programa Python en la Rpi.

Obteniendo el siguiente resultado:



Vídeo del funcionamiento: <https://youtu.be/e0t6lha5vgE>

Web Framework con sensor de temperatura.

Ahora comprobaremos como podemos controlar otro dispositivo hardware desde la Rpi, en este caso, conectaremos una placa Arduino a la Rpi por el puerto serie(USB) y tendremos que leer la temperatura de un sensor TMP36GZ conectado a arduino.

Todo esto controlando la lectura y visualizándola desde la interfaz web(servidor). Por lo que tendremos que modificar el programa Python que realizamos anteriormente para permitir la lectura y también tendremos que modificar el html para mostrar el resultado de la lectura, además de hacer el código arduino para la lectura y transmisión de la temperatura por el puerto serie.

Código Python:

```
import RPi.GPIO as GPIO
import serial
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

# Create a dictionary called pins to store the pin number, name, and
pin state:
```



```

pins = {
    24 : {'name' : 'LED1', 'state' : GPIO.LOW},
    25 : {'name' : 'LED2', 'state' : GPIO.LOW}
}

ser = serial.Serial('/dev/ttyACM0', 9600)

# Set each pin as an output and make it low:
for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
    temps = ser.readline()
    # For each pin, read the pin state and store it in the pins
    dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    # Put the pin dictionary into the template data dictionary:
    templateData = {
        'pins' : pins,
        'temps' : temps
    }
    # Pass the template data into the template main.html and return it
    to the user
    return render_template('main.html', **templateData)

@app.route("/temperature")
def temp():
    temps = ser.readline()
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    message = "La temperatura se ha refrescado"
    templateData = {
        'pins' : pins,
        'temps' : temps,
        'message' : message
    }
    return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the
pin number and action in it:
@app.route("/<changePin>/<action>")
def action(changePin, action):
    # Convert the pin from the URL into an integer:
    changePin = int(changePin)
    # Get the device name for the pin being changed:
    deviceName = pins[changePin]['name']
    # If the action part of the URL is "on," execute the code indented
    below:
    if action == "on":
        # Set the pin high:
        GPIO.output(changePin, GPIO.HIGH)
        # Save the status message to be passed into the template:
        message = "Turned " + deviceName + " on."
    if action == "off":
        GPIO.output(changePin, GPIO.LOW)
        message = "Turned " + deviceName + " off."
    if action == "toggle":

```

```

        # Read the pin and set it to whatever it isn't (that is, toggle
it):
        GPIO.output(changePin, not GPIO.input(changePin))
        message = "Toggled " + deviceName + "."

    # For each pin, read the pin state and store it in the pins
dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    # Along with the pin dictionary, put the message into the template
data dictionary:
    templateData = {
        'message' : message,
        'pins' : pins
    }

    return render_template('main.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)

```

Código HTML:

```

<!DOCTYPE html>
<head>
    <title>Current Status</title>
</head>

<body>
    <h1>Device Listing and Status</h1>

    {% for pin in pins %}
    <p>The {{ pins[pin].name }}
    {% if pins[pin].state == true %}
        is currently on (<a href="/{{pin}}/off">turn off</a>)
    {% else %}
        is currently off (<a href="/{{pin}}/on">turn on</a>)
    {% endif %}
    </p>
    {% endfor %}
    <p>La temperatura es: {{ temps }}(<a
href="/temperature>Refrescar</a>)</p>
    {% if message %}
    <h2>{{ message }}</h2>
    {% endif %}
</body>
</html>

```

Código Arduino:

```

int sensorPin = A0;
char val;

void setup() {
    pinMode(sensorPin, INPUT);
    Serial.begin(9600);
}

void loop() {

```

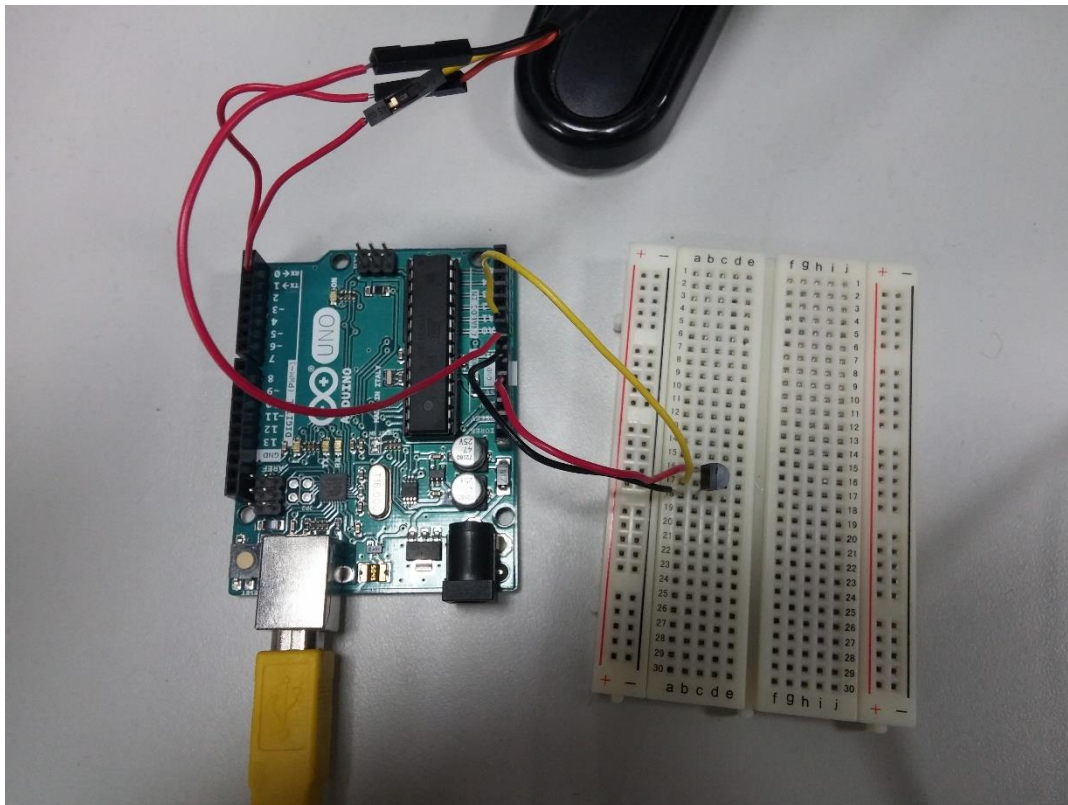
```

float sensorVal;
float temperatura;

sensorVal = analogRead(sensorPin);
temperatura = ((sensorVal/1024)*5 - .5)*100;
Serial.println(temperatura);
delay(2000);
}

```

Imagen de la conexión del sensor de temperatura:



Web Framework con temperatura y relé.

Ahora incorporaremos un relé y tendremos que mantener la lectura de la temperatura, por lo que tendremos que añadir al código Python la lectura del estado como en los led's y ofrecer en la web la posibilidad de cambiarlo, de esta forma escribiremos en el puerto serie un comando que indicará a la arduino si encender o apagar el relé.

Mantendremos el mismo esquema de conexión que con el sensor de temperatura y añadiremos un relé conectado al pin 8 de arduino (con sus respectivas conexiones de VCC y GND). Quedando como resultado lo siguiente:

Código Python:

```

import RPi.GPIO as GPIO
import serial
from flask import Flask, render_template, request
app = Flask(__name__)

```

```

GPIO.setmode(GPIO.BCM)

# Create a dictionary called pins to store the pin number, name, and
pin state:
pins = {
    17 : {'name' : 'Relay', 'state' : GPIO.LOW},
    24 : {'name' : 'LED1', 'state' : GPIO.LOW},
    25 : {'name' : 'LED2', 'state' : GPIO.LOW}
}

ser = serial.Serial('/dev/ttyACM0', 9600)

# Set each pin as an output and make it low:
for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
    temps = ser.readline()
    # For each pin, read the pin state and store it in the pins
    dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    # Put the pin dictionary into the template data dictionary:
    templateData = {
        'pins' : pins,
        'temps' : temps
    }
    # Pass the template data into the template main.html and return it
    to the user
    return render_template('main.html', **templateData)

@app.route("/temperature")
def temp():
    temps = ser.readline()
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    message = "La temperatura se ha refrescado"
    templateData = {
        'pins' : pins,
        'temps' : temps,
        'message' : message
    }
    return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the
pin number and action in it:
@app.route("/<changePin>/<action>")
def action(changePin, action):
    # Convert the pin from the URL into an integer:
    changePin = int(changePin)
    # Get the device name for the pin being changed:
    deviceName = pins[changePin]['name']
    # If the action part of the URL is "on," execute the code indented
    below:
    if action == "on":
        if changePin == 17:
            ser.write('L')
        # Set the pin high:

```

```

GPIO.output(changePin, GPIO.HIGH)
# Save the status message to be passed into the template:
message = "Cambiado " + deviceName + " a encendido."
if action == "off":
    if changePin == 17:
        ser.write('H')
    # Set the pin high:
    GPIO.output(changePin, GPIO.LOW)
    message = "Cambiado " + deviceName + " a apagado."
if action == "toggle":
    # Read the pin and set it to whatever it isn't (that is, toggle
it):
    GPIO.output(changePin, not GPIO.input(changePin))
    message = "Toggled " + deviceName + "."

# For each pin, read the pin state and store it in the pins
dictionary:
for pin in pins:
    pins[pin]['state'] = GPIO.input(pin)

# Along with the pin dictionary, put the message into the template
data dictionary:
templateData = {
    #'message' : message,
    'pins' : pins
}

return render_template('main.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)

```

Código HTML:

```

<!DOCTYPE html>
<head>
    <title>Current Status</title>
</head>

<body>
    <h1>Device Listing and Status</h1>

    {% for pin in pins %}
    <p>The {{ pins[pin].name }}
    {% if pins[pin].state == true %}
        is currently on (<a href="/{{pin}}/off">turn off</a>)
    {% else %}
        is currently off (<a href="/{{pin}}/on">turn on</a>)
    {% endif %}
    </p>
    {% endfor %}
    <p>La temperatura es: {{ temps }}(<a
href=/temperature>Refrescar</a>)</p>
    {% if message %}
    <h2>{{ message }}</h2>
    {% endif %}
</body>
</html>

```

Código Arduino:

```

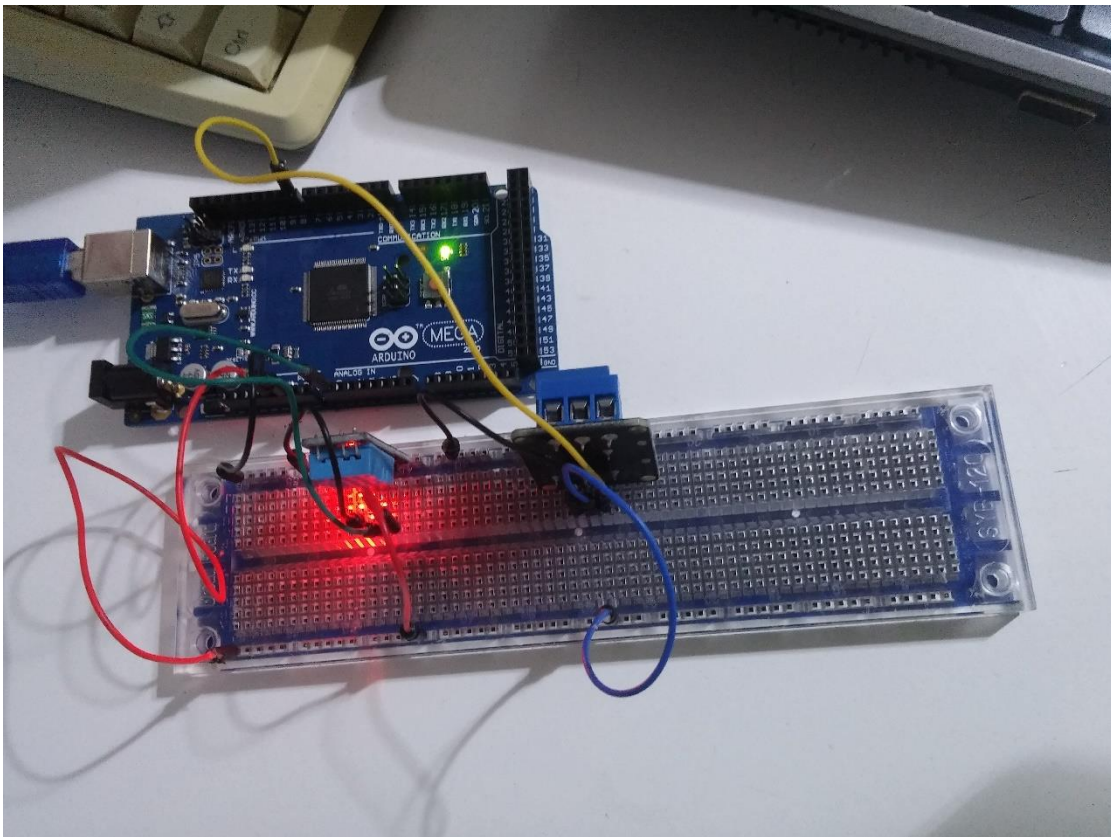
int pinRele = 8;
int sensorPin = 0;
char val;

void setup() {
  pinMode(pinRele, OUTPUT);
  Serial.begin(9600);
}

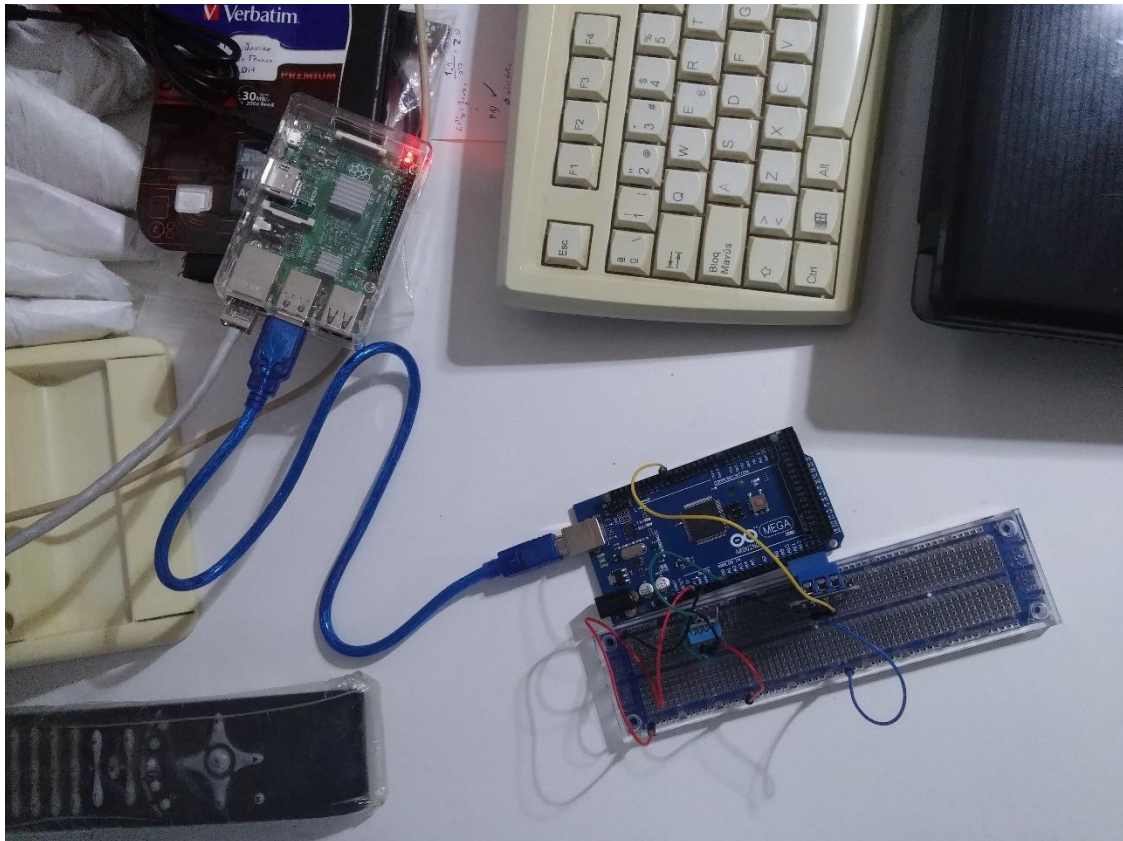
void loop() {
  float sensorVal;
  float temperatura;
  if(Serial.available() > 0){
    char c = Serial.read();
    if(c == 'H'){
      digitalWrite(pinRele, HIGH);
    }else if(c == 'L'){
      digitalWrite(pinRele, LOW);
    }
  }
  sensorVal = analogRead(sensorPin);
  temperatura = ((sensorVal/1024)*5 - .5)*100;
  Serial.println(temperatura);
  delay(2000);
}

```

Obteniendo los siguientes resultados:



(imagen del circuito arduino)



(imagen del circuito completo)

Video del funcionamiento: <https://youtu.be/TKtgZljtlzl>

NOTA:(En el vídeo en vez del relé enciendo el led del pin 13 de la placa arduino mega por un mal funcionamiento del relé).

Conclusión

Hemos podido comprobar como con Python podemos controlar la GPIO de esta placa SBC, además de comprobar las múltiples utilidades, tanto de control directo de hardware conectado a su GPIO como el control de otros dispositivos hardware a través del puerto serie.

Además, vemos como esta plataforma nos da la posibilidad de controlar la placa y el hardware asociado a través de internet mediante una interfaz web, instalando en la plataforma un framework que nos permite ejecutar código Python desde la interfaz web, dándonos control sobre los dispositivos.

De esta forma vemos las grandes capacidades de esta plataforma y la multitud de aplicaciones en la que se puede usar, como diseñar prototipos de sistemas más complejos como puede ser el caso de la domótica.