



LABORATORIO DE DESARROLLO HARDWARE - MEMORIA DE RASPBERRY



José Manuel Jarana Expósito

Objetivos.....	2
-----------------------	----------

Introducción.....	2
--------------------------	----------

ZPUIno ¡Error! Marcador no definido.

Es un SoC implementado en VHDL (soft core) y adaptado para el uso de la IDE de Arduino basado en el microprocesador ZPU de Zylins. Es un procesador de 32 bits que trabaja a una frecuencia de 100MHz. Todo es controlado por Sketchs y por el sencillo uso de las bibliotecas de Arduino. ¡Error! Marcador no definido.

Desarrollo de la práctica	3
--	----------

1. Nuestro primer objetivo para empezar a manejarnos bien con el diseñador de circuitos, haremos uno que simplemente consiste en crear un inversor y conectarle un led, para que se encienda con una entrada de 0. ¡Error! Marcador no definido.

2. Rediseñando el SoC. Añadiendo periféricos (Display 7-seg)..... ¡Error! Marcador no definido.

3. Convirtiendo la placa papilio en un Analizador lógico ¡Error! Marcador no definido.

Conclusiones	19
---------------------------	-----------

Objetivos

Tras las dos placas anteriores, ahora veremos la Raspberry Pi 3 y seguiremos algunos tutoriales para, básicamente, tener un servidor web en nuestra placa Raspberry en la que instalaremos nuestro sistema operativo. En la Web podremos manejar cuando encender una luz por ejemplo o recibir información, como la temperatura.

1. Vamos a ver el manejo de GPIOs desde línea de comandos:
2. Vamos a ver ejemplos sencillos con PYTHON.
3. Vamos a conectarnos a la placa mediante SSH.
4. Instalar un Servidor WEB que pueda ejecutar código PYTHON desde el servidor, haciendo que se enciendan unos Leds en los GPIO de la placa.
5. Más tarde, conectaremos con Arduino para que nos dé la temperatura y añadiremos ese valor que recibimos a la web para que nos lo muestre.
6. Añadir al sensor de temperatura un Relé con bombilla y que podamos encender o apagar la bombilla.

Introducción

RaspBerry Pi 3

La Raspberry, también conocida como pequeño computador, es una placa que soporta varios componentes de un ordenador común y que puede ser utilizado para cosas como procesadores de texto, visualizador de video e imágenes o un servidor, por eso lo de “minipc”.

Esta placa se desarrolló en el Reino Unido por la fundación Raspberry Pi, la cual se comercializó por primera vez en 2012 y fue un éxito. Desde entonces la placa ha cambiado mucho, ahora el modelo con el que hemos trabajado nosotros, como la cantidad de USB que tiene ahora o la nueva característica del WI-FI, que se incluyó en la última versión además de un procesador más potente.

Como ya hemos dicho, la placa cuenta varios puertos como 4 USBs, una ranura MicroSD, 17 GPIO y un bus HAT ID y una salida de HDMI. Además, sus especificaciones Hardware también han aumentado bastante, teniendo un ARMv8 quad-core de 1.2GHz 64-bit y una SDRAM de 1 GB.

En la propia página de RaspBerry hay para instalarle los sistemas operativos que queramos a nuestra placa, pudiendo utilizar el propio creado por ellos, RaspBian, u otros de Ubuntu (que será el que utilicemos nosotros) o Windows incluso. Una vez instalemos este sistema operativo en nuestra tarjeta microSD, lo que hagamos con la placa ya dependerá de la funcionalidad que

queramos darle. Además, al igual que Arduino, esta placa tiene una comunidad muy grande detrás debido a su extensa comercialización y a su versatilidad (por no hablar de su precio), que nos pueden ayudar a la hora de manejarnos con la placa con tutoriales o dudas que nos puedan surgir.

Desarrollo de la práctica

Antes de todo esto, como dijimos antes debemos de instalar Ubuntu Mate y configurarlo correctamente para nuestra tarjeta SD y que tengamos conexión y demás.

Primero la imagen del sistema operativo en Linux en nuestra tarjeta. Para ello usaremos la herramienta de Linux “dd”.

-Buscamos nuestra tarjeta SD con el siguiente comando.

```
df -h
```

-Tras esto, desmontamos la unidad en la que está nuestra tarjeta(suponemos “/dev/sdd1” nuestro slot).

```
umount /dev/sdd1
```

-Ahora utilizaremos “dd”. Poniendo nuestro nombre de imagen en “if=” y nuestro slot desmontado en “of=”.

```
dd bs=4M if=2016-09-23-raspbian-jessie.img of=/dev/sdd
```

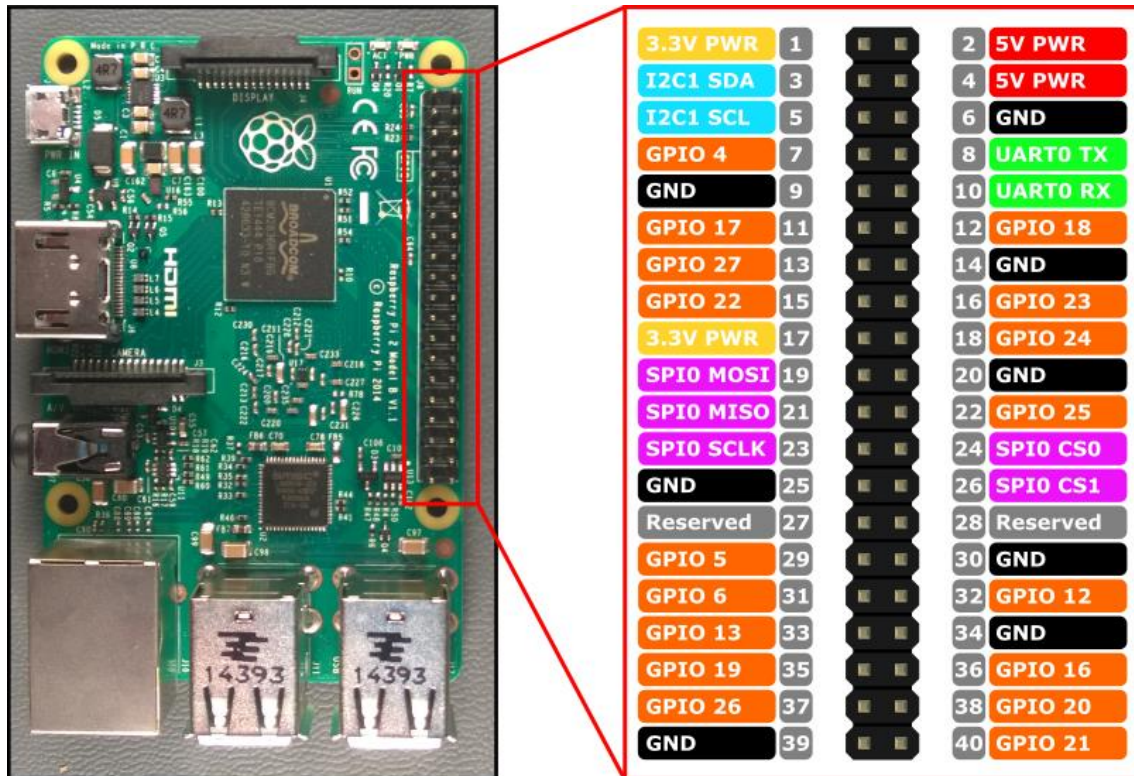
Tras esto esperaremos a se escriba la imagen, que tardará unos 10 minutos. Cuando el prompt aparezca de nuevo, será la señal de que ha terminado de escribir la imagen. Ahora procederemos a la instalación.

Empezaremos la instalación nada más meter la tarjeta en la placa, y veremos que es igual exactamente a la instalación de Linux. Deberemos elegir idioma, zona horaria, teclado...

Ahora configuramos la conexión, accediendo a las interfaces de nuestra placa y configurándola manualmente con la intención de que no tengamos que hacer esto de nuevo y la conectemos cambiando la sd de placa siga la conexión y no se deshaga. Sin embargo, como comprobamos en la siguiente práctica, esto no es así, ya que también depende de la dirección física de placa, y por tanto, si está no es la misma, aunque cambiemos la interfaz.

1. Vamos a ver el manejo de GPIOs desde línea de comandos

Vamos a encender un led (con su propia resistencia) que colocaremos en el GPIO 17 de nuestra placa. Este GPIO da una señal de 3.3V y la otra pata al GND. Para guiarnos, he mirado la siguiente imagen como esquema para saber cuales son los pines. y donde conectarlo todo.



Una vez hemos puesto correctamente el led, ahora tendremos que utilizar nuestra GPIO y la configuraremos así.

Primero crearemos una estructura GPIO para nuestro GPIO 17.

```
echo 17 > /sys/class/gpio/export
```

Después diremos que esta GPIO será de salida modificando el archivo que creamos.

```
echo out > /sys/class/gpio/gpio17/direction
```

Tras esto, encenderemos el led mandado '1' o '0'

```
echo 1 > /sys/class/gpio/gpio17/value //Para encender el led
```

ó

```
echo 0 > /sys/class/gpio/gpio17/value //Para apagar el led
```

Algo importante que debemos hacer es eliminar la GPIO que hemos creado previamente, para que si después de darle nuestro uso queremos usarla en algún momento ese mismo GPIO con otra intención, no haya conflicto de intereses.

```
echo 17 > /sys/class/gpio/unexport
```

2. Vamos a ver ejemplos sencillos con PYTHON.

Lo primero será instalar la librería de Python, que utilizaremos estos comandos en el terminal para instalarla:

- Para descargarla

```
wget 'http://downloads.sourceforge.net/project/raspberry-gpio-python/RPi.GPIO-0.5.4.tar.gz'
```

- Descomprimos e instalamos desde la carpeta que nos hemos creado.

```
tar zxvf RPi.GPIO-0.5.4.tar.gz
cd RPi.GPIO-0.5.4/
sudo apt-get install python-dev //Es opcional por si no tenemos este paquete
sudo python setup.py install
```

- Importamos ahora la librería que hemos instalado y declararemos los pines como salida

```
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
```

- Ahora definiremos el programa en Python, uno sencillo en el que ambos leds se encienden y apagan de forma intermitente.

```
def blink():
    print "Ejecucion iniciada..."
    iteracion = 0
    while iteracion < 30: ## Segundos que durara la funcion
```

```

GPIO.output(17, True) ## Enciendo 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 blink
print "Ejecucion finalizada"
GPIO.cleanup() ## Hago una limpieza de los GPIO

```

Tras guardar esto en un archivo con terminación .py y lo ejecutaremos como administrador para comprobar como funciona.

3. Conexión SSH y web framework para Python.

Lo primero que haremos será la conexión ssh, por lo que podremos quitar todos los periféricos menos el cable de Ethernet. Después ejecutamos en otro PC

```
$ ssh -X <usuario>@<IPadressRasp>
```

Donde usuario será el usuario de nuestra placa que pusimos al instalar Ubuntu y la IPadressRasp será la que configuramos anteriormente en la interfaces.

Una vez hecha la conexión, lo que haremos será la Web Framework.

Vamos a escribir un código en Python muy simple, en el que sólo mostraremos un “Hola mundo” y la hora que es actualmente. Lo llamaremos hello-template.py

```

from flask import Flask, render_template
import datetime
app = Flask(__name__)

@app.route("/")
def hello():
    now = datetime.datetime.now()
    timeString = now.strftime("%Y-%m-%d %H:%M")
    templateData = {
        'title' : 'HELLO!',

```

```

        'time': timeString
    }
    return render_template('main.html', **templateData)

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

```

Crearemos un código html simple para mostrar la cadena y el la hora actual:

```

<!DOCTYPE html>
<head>
    <title>{{ title }}</title>
</head>

<body>
    <h1>Hello, World!</h1>
    <h2>The date and time on the server is: {{ time }}</h2>
</body>
</html>

```

Ejecutamos el código Python por ssh de forma remota en la Raspberry.

Comprobamos que la página funciona escribiendo en nuestro navegador la dirección IP de nuestra placa.

4. *Instalar un Servidor WEB que pueda ejecutar código PYTHON desde el servidor, haciendo que se enciendan unos Leds en los GPIO de la placa.*

Tras crear el código anterior, vamos a hacer otro en el que maneje los GPIOs de la placa a través de la página para que encienda dos leds.

El programa en Python que utilizaremos será:

```

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' : 'coffee maker', 'state' : GPIO.LOW},
    25 : {'name' : 'lamp', '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)</html>

```

Cambiaremos obviamente el HTML para que se adapte a este nuevo programa, y podamos modificar el estado de los leds. Este será nuestro código.

```

<!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>

```

De nuevo comprobamos que funciona ejecutando el código en Python y después poniendo la dirección Ip en nuestro navegador.

5. Más tarde, conectaremos con Arduino para que nos dé la temperatura y añadiremos ese valor que recibimos a la web para que nos lo muestre.

Lo que haremos en este punto será conectar nuestra placa Arduino a la Raspberry Pi para que nos envíe la temperatura que detectará con un sensor previamente puesto en nuestro Arduino. Después ese dato de la temperatura lo que haremos será mostrarlo en nuestra página Web.

Instalamos el IDE de Arduino en nuestra Raspberry, para que podamos programar y manejar la placa desde ahí. Nos dará un error de conexión el IDE, el cual podemos solucionar fácil ejecutando en la consola de comandos:

```
Sudo adduser <username> dialout
```

Para que surja efecto la nueva configuración debemos reiniciar la placa, o simplemente cerrar sesión. Tras esto, y poner adecuadamente el puerto serie en el que está el modelo de la placa, ya tenemos nuestro IDE configurado correctamente.

Una vez hecho esto, lo que haremos será crear nuestro programa en Arduino para que lea del sensor de temperatura que le hemos conectado. Nuestro código será idéntico al que teníamos en la práctica de Arduino.

```
int sensorPin=0;

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

void loop() {
  // Espera el carácter le llegue
  delay(1000);

  //Lee el dato que nos envía
  int reading = analogRead(sensorPin);

  // Convertimos la lectura en voltaje
  float voltage = reading * 5.0;
  voltage /= 1024.0;

  // Hacemos la conversión de voltaje a grados
  float temperatureC = (voltage - 0.5) * 100 ;

  // Lo imprimimos por pantalla
  Serial.println(temperatureC);
}
```

El código HTML de la página es lo modificamos un poco.

```
<!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>Temperatura: {{tmp}} (<a href=/temperature> Actualizar</a>)</p>

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

</body>
</html>

```

El código de Python que se ejecutará para manejar los leds y enviar la temperatura es:

```

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():
    tmp = 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
        'tmp' : tmp
    }

    # 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 = "Se actualizó la temp"
    templateData = {
        'pins' : pins,
        'tmp' : tmp,
        'message' : message
    }
    return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the pin num
ber 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)

```

6. Añadir al sensor de temperatura un Relé con bombilla y que podamos encender o apagar la bombilla.

Ahora añadiremos al sensor de temperatura un con relé con bombilla para encender o apagar la luz mientras vemos la temperatura. Ahora mostraremos los tres códigos necesarios para esto, como en el punto anterior.

Código de Arduino:

```
int sensorPin=0;
int rele = 8;
int sensorPin = 0;
char val;
void setup(){
    pinMode(pinRele, OUTPUT);
    Serial.begin(9600);
}
void loop(){
    delay(1000);
    float temp;
    if(Serial.available() > 0){
        char c = Serial.read();
        if(c == 'H'){
            digitalWrite(rele, HIGH);
        }else if(c == 'L'){
            digitalWrite(rele, LOW);
        }
    }
    int reading = analogRead(sensorPin);
    temp = ((sensorVal/1024)*5 - .5)*100;
    Serial.println(temp);
}
```


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>Temperatura: {{tmp}} (<a href=/temperature> Actualizar</a>)</p>

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

</body>
</html>
```

Código Python para manejar.

```
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 = {
    17 : {'name' : 'rel', 'state' : GPIO.LOW},
    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():
    tmp = 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
        'tmp' : tmp
    }

    # 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():
    tmp = ser.readline()
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    message = "Se actualizó la temp"
    templateData = {
        'pins' : pins,
        'tmp' : tmp,

```

```

        'message' : message
    }

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

@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 = "Turned " + deviceName + " on."
    if action == "off":
        if changePin == 17:
            ser.write('H')
        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)
```

Conclusiones

Por las prácticas hemos podido ver la gran versatilidad de esta placa, teniendo en cuenta que es capaz de soportar un Sistema Operativo, por lo tanto cumplir con la mayoría de las funciones capaz de hacer un Pc. Esto unido a sus diferentes pines y sus variedades en los tipos de conexión como los GPIOs que dispone, nos permite, no solo hacerlo utilizarla la placa como un PC ordinario, sino también conectarle varios sensores de entrada o de salida según la necesidad que tengamos. Uniendo ambas características podemos comprobar el gran potencial que tiene Raspberry.

Además de estas características se suma la posibilidad de usarla como servidor controlado a distancia de forma inalámbrica si queremos, y manejarlo desde un Pc para mayor comodidad, en caso de que por ejemplo, queramos modificar algo del entorno de nuestra placa y nos encontremos lejos.