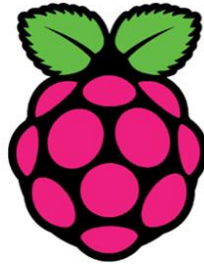


MEMORIA DE PRÁCTICA DE RASPBERRY PI



RaspberryPi

Nombre: Francisco Núñez Cid

Correo: francisco.nunez.cid@gmail.com

Curso: 2016 – 2017

ÍNDICE

• Introducción	3
• Objetivos	3
• Fundamentos teóricos	
○ ¿Qué es Raspberry Pi?	4
○ Arquitectura de la placa	4
○ Programación	5
○ Instalación del Sistema Operativo a la placa	5
○ Configuración del Sistema Operativo	6
○ Utilización de la placa	7
○ Hardware utilizado	7
• Ejercicios	8
• Conclusión	25
• Glosario	25
• Bibliografía	27

Introducción

Durante el desarrollo de todas las prácticas de laboratorio del primer bloque, de la parte de **Raspberry Pi**, he ido tomando una serie de notas de lo que he ido realizando, los problemas que me he ido encontrado a la hora de hacer los diferentes ejercicios, las soluciones de cada uno de ellos y los diferentes resultados que voy concluyendo.

Objetivos

Los principales propósitos de las prácticas de Raspberry Pi son los siguientes:

- Preparar la plataforma Raspberry Pi para que puedan cargarse diferentes versiones de Sistema Operativo.
- Arrancar y comprobar el funcionamiento de la placa Raspberry Pi.
- Desarrollar ejemplos de utilización de los pines de expansión GPIOs y utilizarla en comunicación con un PC.
- Instalar un Servidor WEB que pueda ejecutar código PYTHON.

Fundamentos teóricos

- **¿Qué es Raspberry Pi?**

Es un ordenador de placa reducida o de placa simple de bajo coste desarrollado en Reino Unido por la fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

- **Arquitectura de la placa**

La placa Raspberry Pi 3 Model B se muestra en la siguiente imagen marcando los elementos más usados.

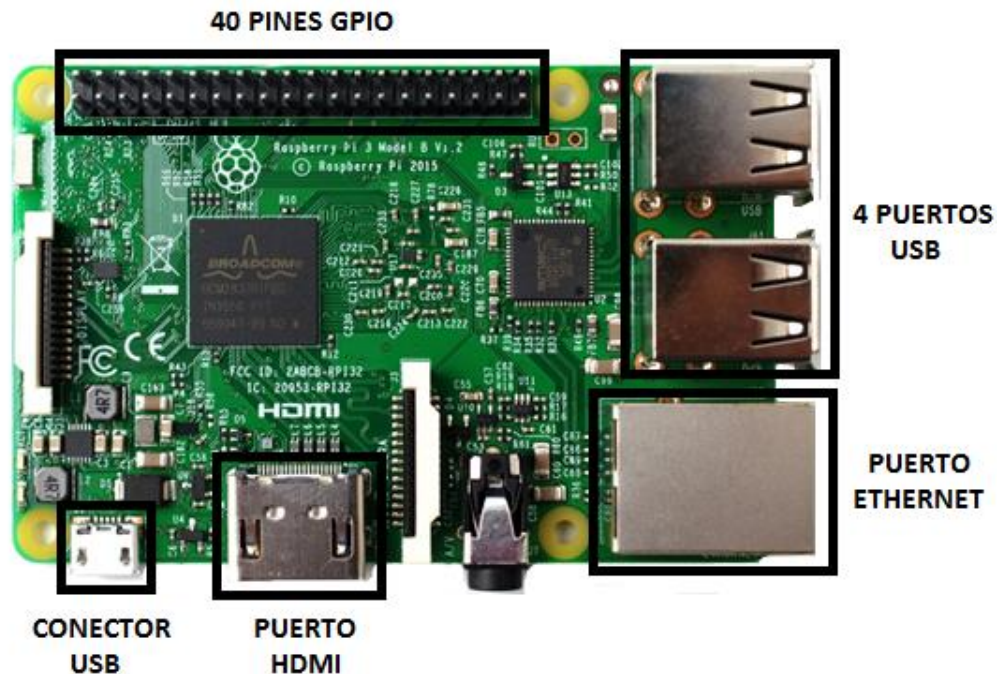


Imagen 1

Esta placa posee un SoC Broadcom BCM2387 con cuatro núcleos ARM Cortex-A53 a 1.2 GHz y un 1GB de RAM.

Dispone de 40 pines GPIO (entre los que hay de entrada/salida, 3.3V, 5V, GND), 1 puerto HDMI, 1 puerto de Ethernet, 4 puertos USB y 1 conector USB para conectar con el PC.

También dispone de una ranura de tarjeta de memoria empuje (Micro SD), cable Jack de 3,5 mm de salida de audio y otros componentes.

- **Programación**

Raspberry Pi no tiene un entorno de desarrollo de programación como Arduino y ZPUino. Para poder ejecutar cosas se va a realizar a través de una ventana de comandos y programas en Python, se verá en la parte de ejercicios algunos ejemplos.

- **Instalación del Sistema Operativo a la placa**

1. Descargar la imagen de Ubuntu Mate, desde: <http://coria.dte.us.es/~bellido/>.
2. La imagen a descargar es: **ubuntu-mate-16.04-desktop-armhf-raspberry-pi-resize.img**.
3. Abrir una ventana de símbolos y escribir: **"df -h"**, para ver qué dispositivos están montados actualmente.
4. Insertar la tarjeta micro SD al ordenador.
5. Ejecutar de nuevo **"df -h"**, debería de aparecer un nuevo dispositivo, esa es la tarjeta SD. Se mostrará algo como **"/dev/sdd1"**. Esta última parte, es decir, el 1, es el número de particiones pero se desea escribir en toda la tarjeta SD, por lo tanto, hay que borrar esa última parte y debería de salir algo así o parecido: **"/dev/sdd"** como el nombre del dispositivo para toda la tarjeta.
6. Una vez que se ha anotado el nombre del dispositivo hay que desmontarlo para que los archivos no se puedan leer ni escribir en la tarjeta SD mientras copia sobre la imagen SD.
7. Ejecutar **"umount/dev/sdd1"**, reemplazando **sdd1** con el nombre del dispositivo de su tarjeta SD (incluyendo el número de partición).
8. Si su tarjeta SD se muestra más de 1 vez en la salida de **"df"** es debido a tener varias particiones en la tarjeta SD, debe desmontarlas todas.
9. Escribir en la ventana de símbolos: **"dd bs=4M if=XXX.img of=/dev/sdd"**. Siendo XXX = argumento con la ruta al archivo .img de Ubuntu Mate.
10. Si no ha iniciado sesión como root, tendrá que prefijar el comando anterior con **"sudo"**, luego empieza a escribir en la tarjeta y el proceso de escritura puede tardar unos 5 minutos (hay que esperar).
11. Una vez finalizada la escritura, retirar la micro SD del ordenador y probarlo en la Raspberry Pi 3.

• Configuración del Sistema Operativo

1. Al arrancar configurar idioma, teclado, localización, fecha y hora.
2. Guardar información sobre usuario y password elegidos.

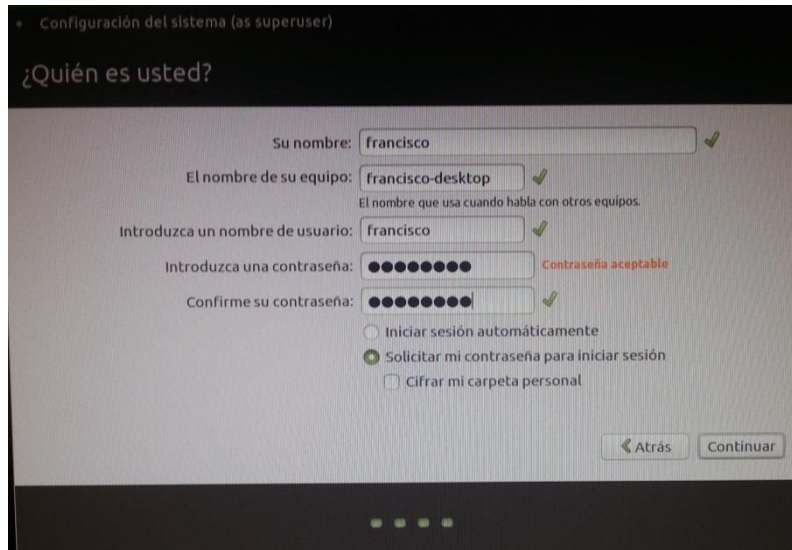


Imagen 2

3. Paso ahora a configurar la red manualmente.
4. Desactivar **network-manager**.
5. Editar **/etc/network/interfaces**.
 - Iface <nombre_InterfazDeRed> inet static (cambiar dhcp por static).
 - Address 10.1.15.xx (siendo xx-número del PC).
 - Netmask 255.255.252.0 ó 22.
 - Gateway 10.1.15.78
 - Dns-nameservers 8.8.8.8

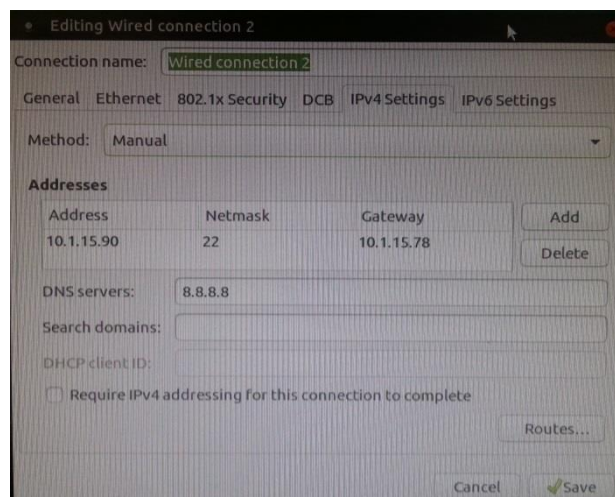


Imagen 3

6. Ejecutar sucesivamente: **\$ sudo ifdown <nombre_InterfazDeRed>** y **\$ sudo ifup <nombre_InterfazDeRed>**.
7. Probar conexión: **\$ ping www.google.es**.
8. Una vez configurada la red hay que modificar la hora para poder acceder sin problemas a Internet.

- **Utilización de la placa**

1. Conectar la SD a Raspberry Pi 3.
2. Desconectar el teclado, ratón y cable Ethernet del PC y conectarlo a la placa.
3. Conectar cable HDMI-DVI a Raspberry Pi y a la pantalla del ordenador.
4. Conectar cable USB-Micro usb a Raspberry Pi y PC para alimentar a la placa.

- **Hardware utilizado**

1. Placa Raspberry Pi 3 Modelo B.
2. Cable USB.
3. Resistencias.
4. Diodos led.
5. Cables para realizar las conexiones.
6. Cable HDMI.
7. Cable TTL.
8. Sensor de temperatura tmp36GZ.

Ejercicios

1. Manejar GPIOs desde la línea de comandos.

En este primer ejercicio se va a realizar el control de los GPIOs de Raspberry Pi (encender y apagar leds) a través de una ventana de comandos. Para realizar el circuito voy a añadirle una resistencia al led.

Yo he realizado este ejercicio de la siguiente manera:

- Primero, hay que tener conectada la placa al ordenador (ver **Utilización de la placa**).
- Ahora, hago el montaje del circuito en la Raspberry. He conectado el led al GPIO 17 que es el pin 11 y la salida (GND) al pin 6, ver imagen 4 para ver las distintas conexiones de los GPIOs.

3.3 V	1	2	5V
SDA0	3	4	5V
SCL0	5	6	GROUND
GPIO4	7	8	UART TX
GROUND	9	10	UART RX
GPIO17	11	12	GPIO 18 PWM
GPIO21	13	14	GROUND
GPIO 22	15	16	GPIO23
3.3 V	17	18	GPIO24
MOSI	19	20	GROUND
MISO	21	22	GPIO 25
SCLK	23	24	CE0
GROUND	25	26	CE1

Imagen 4

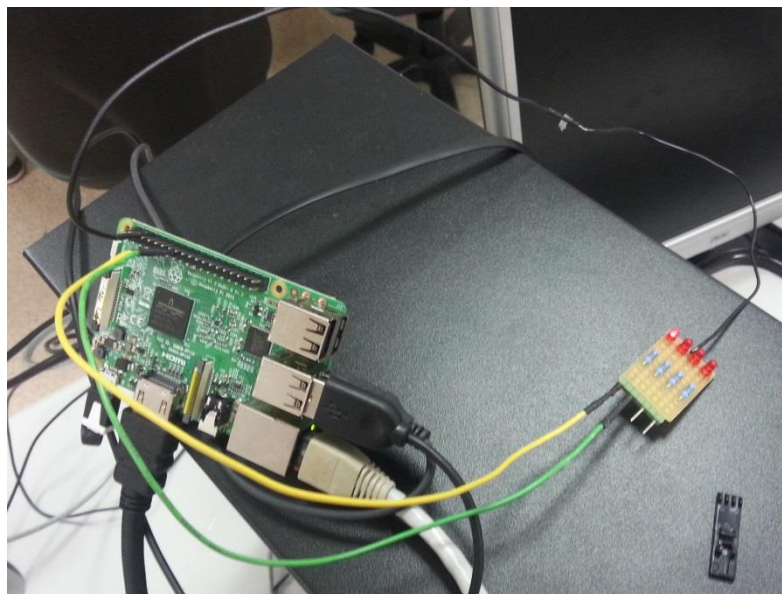


Imagen 5

- A continuación, me paso con la parte de software. Abro una ventana de comandos y como no hay ningún pin accesible (ni de entrada ni de salida) hay que crearlo. Para conectarme al GPIO 17, hay que introducir el siguiente comando:

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

Esto hace que el sistema cree un archivo con una estructura GPIO que corresponde al número 17.

- Ahora, informo a la Raspberry Pi de si el pin va a ser de entrada o de salida, como lo que quiero hacer es encender el LED, el GPIO 17 será de salida. Le introduzco el siguiente comando:

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

- Después, para encender el LED, escribo el siguiente comando:

```
echo 1 > /sys/class/gpio/gpio17/value
```

O si quiero apagar el LED, escribo lo siguiente:

```
echo 0 > /sys/class/gpio/gpio17/value
```

- Por último, hay que eliminar la entrada GPIO creada, es decir, el GPIO utilizado, que ha sido el GPIO 17. Para ello se introduce el siguiente comando en la ventana de símbolos:

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

Nota: En este primer ejercicio, en la parte de hardware no he tenido problemas, solo ha sido conectar los leds a los pines correctos de la placa y conectar los distintos cables (USB a PC, Ratón, Teclado, HDMI y Ethernet) para poder visualizar el Sistema Operativo que tiene la placa en el ordenador. La parte de software tampoco ha sido ningún problema porque he seguido un tutorial que venía muy guiado y en el que se explica paso a paso todas las cosas que hay que realizar.

2. Manejar GPIOs con un programa python.

En este segundo ejercicio hay que hacer algo parecido al ejercicio anterior. Realizar el control de los GPIOs de Raspberry Pi (encender y apagar 2 leds pero alternativamente y esperar 1 segundo), pero en este caso es a través de un programa en python. Hay que añadirle una resistencia a los leds.

Yo he realizado este ejercicio de la siguiente manera:

- Primero, hay que tener conectada la placa al ordenador (ver **Utilización de la placa**).
- Ahora, hago la parte hardware. He conectado 2 leds, el primero al GPIO 17 que es el pin 11, el segundo al GPIO 27 que es el pin 13 y la salida (GND) al pin 6 , ver imagen 4 para ver las distintas conexiones de los GPIOs.

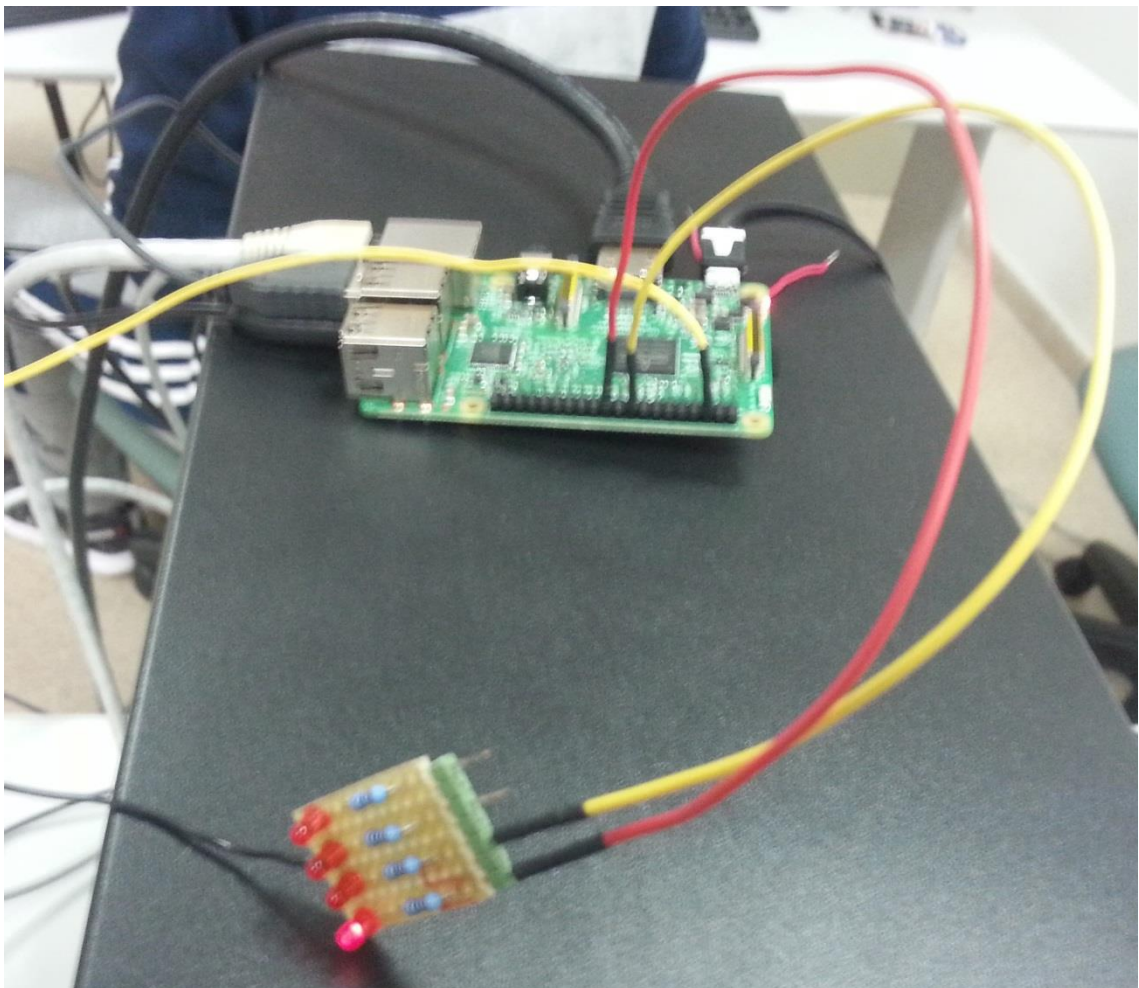


Imagen 6

- A continuación, me paso con la parte de software, en la que voy a escribir un pequeño programa en Python que haga que se enciendan y se apaguen los LEDs de forma intermitente. Para ello, abro una ventana de símbolos y ejecuto:

sudo nano blink.py

Esto hace que se me abra un fichero nuevo de tipo Python, que lo he llamado Blink.

- Dentro del archivo creado "Blink.py" importo la librería GPIO y declaro los pines. Los pines serán de salida, porque hay que encender los leds. Voy a utilizar los GPIOs 17 y 27.

```
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, me paso a crear una función para ejecutar el bucle que encienda y apague los LEDs.

```
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 parpadeos
    print "Ejecucion finalizada"
    GPIO.cleanup() ## Hago una limpieza de los GPIO
```

- Por último, una vez que se ha hecho el código en Python lo único que hay que hacer es ejecutarlo en la ventana de símbolos, con el siguiente comando:

```
sudo python blink.py
```

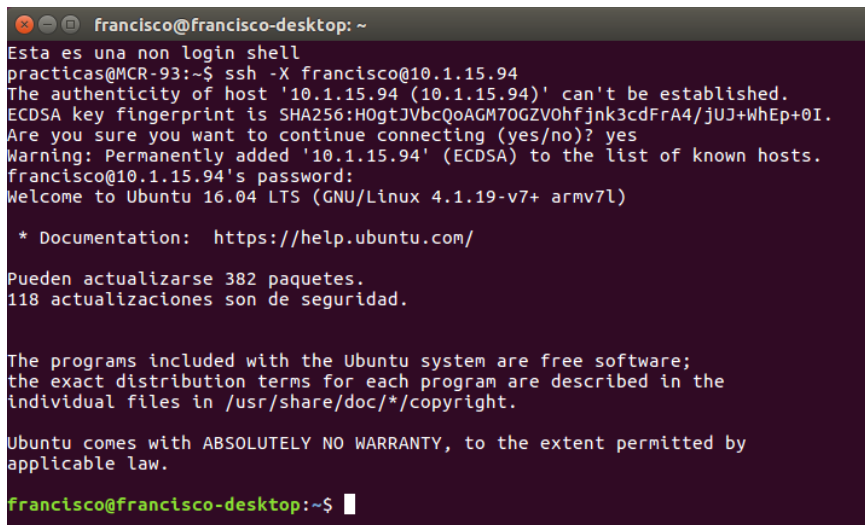
Nota: En este segundo ejercicio, en la parte de hardware no he tenido problemas, porque he reutilizado el circuito que ya tenía montado para el primer ejercicio. Simplemente, ha sido conectar los leds a los pines correctos de la placa (17 y 27) y conectar los distintos cables (USB a PC, Ratón, Teclado, HDMI y Ethernet) para poder visualizar el Sistema Operativo que tiene la placa en el ordenador. La parte de software tampoco ha sido ningún problema porque he seguido un tutorial que venía muy guiado y en el que se explica paso a paso todas las cosas que hay que realizar, la única diferencia con el ejercicio 1 es que esta vez se han encendido los leds a través de una función en Python que enciende el Led 1 y el Led 2 está pagado y un segundo después se apaga el Led 1 y se enciende el Led 2 y así sucesivamente.

3. Acceso a una consola de la Raspberry Pi a través de conexión vía SSH.

Yo he realizado este ejercicio de la siguiente manera:

- Apagar Raspberry Pi, desconectar el teclado, el monitor y el ratón.
- Conectar solo el cable de Ethernet de red.
- Encender Raspberry Pi.
- Conexión por SSH a Raspberry Pi desde el PC, con el comando:

"\$ ssh -X <usuario>@<IPadressRasp>"



```
francisco@francisco-desktop: ~  
Esta es una non login shell  
practicass@MCR-93:~$ ssh -X francisco@10.1.15.94  
The authenticity of host '10.1.15.94 (10.1.15.94)' can't be established.  
ECDSA key fingerprint is SHA256:H0gtJVbcQoAGM70GZV0hfjnk3cdFrA4/jUJ+WhEp+0I.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '10.1.15.94' (ECDSA) to the list of known hosts.  
francisco@10.1.15.94's password:  
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.1.19-v7+ armv7l)  
  
* Documentation:  https://help.ubuntu.com/  
  
Pueden actualizarse 382 paquetes.  
118 actualizaciones son de seguridad.  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
francisco@francisco-desktop:~$
```

Imagen 7

Nota: En este tercer ejercicio no ha habido que hacer parte de hardware, sino solo software. Simplemente escribir el comando anterior para poder hacer uso de la conexión y así no tener que utilizar siempre el teclado, monitor y ratón, pero esto nos ha dado siempre un problema en las prácticas. Cuando configuramos la placa había que configurarle una MAC, que cada placa tenía una distinta y nosotros cada vez que teníamos prácticas siempre cogíamos una placa diferente (se debería de ver puesto una etiqueta a la placa para identificar de quien es o quien hace uso de ella) por lo que daba fallos si se cogía otra diferente.

4. Montar un “**web framework**” para Python (se podrá ejecutar código python desde el servidor web) y en el que no se empleará Relés, sino solo Leds.

En este cuarto ejercicio se va a hacer un servidor web en el que se probará ejemplos realizados en Python. En este caso, se hará que pulsando en un mensaje de un led (LED 1 o LED 2) se encienda o se apague el led (ver abajo imagen 9).

Yo he realizado este ejercicio de la siguiente manera:

- Primero, he realizado el montaje del circuito, reutilización el montaje de los ejercicios 1 y 2, pero en este caso he utilizado los pines 24 y 25 para los leds (ver explicación del montaje en [ejercicio 2](#)).

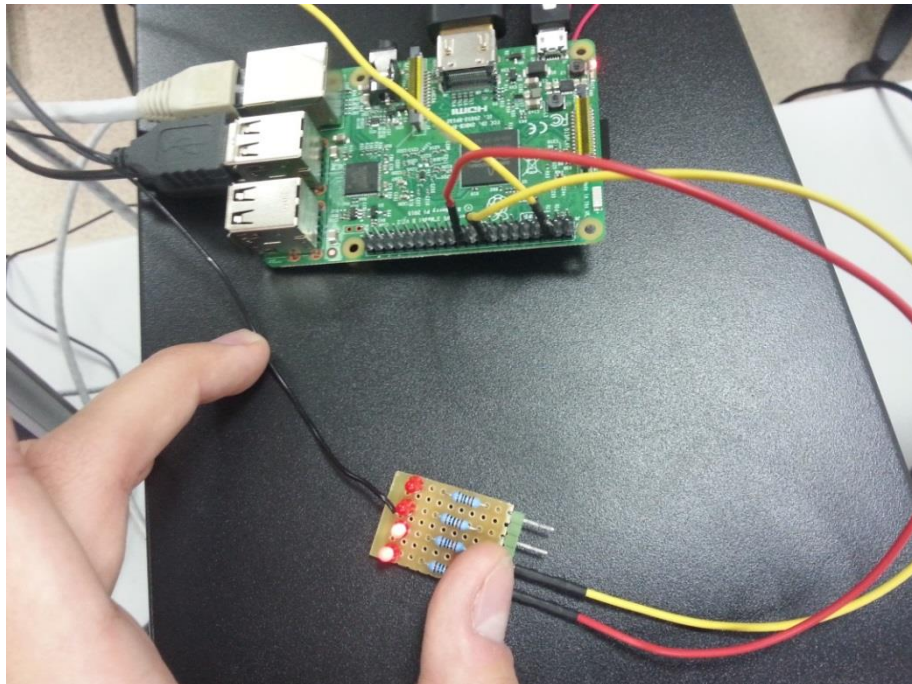


Imagen 8

- Ahora, me paso con la parte de software. Primero, instalo el *Flask*, con el siguiente comando: “\$ **sudo apt-get install python-pip**” y luego “\$ **sudo pip install flask**”.
- Me creo una carpeta que la he llamado “**WebLamp**” en la que meto dentro un archivo **.py**, que lo he llamado “**weblamp.py**” y dentro de la carpeta “**WebLamp**” también me creo otra carpeta, llamada “**templates**”, en la que meto el “**main.html**”.

El código de “**weblamp.py**” es el siguiente:

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

GPIO.setmode(GPIO.BCM)

# Se crea un diccionario llamado Pins para almacenar
el número, el nombre y el estado del pin:
pins = {
    24 : {'name' : 'LED1', 'state' : GPIO.LOW},
    25 : {'name' : 'LED2', 'state' : GPIO.LOW}
}

# Establezco cada pin como una salida y lo pongo a
LOW:
for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
    # Para cada pin, leer el estado y guardarlo en el
    diccionario Pins creado antes:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    # Colocar el diccionario de Pins en el diccionario
    templateData:
    templateData = {
        'pins' : pins
    }
    # Pasar los datos de la plantilla a main.html y
    devolverse al usuario:
    return render_template('main.html',
        **templateData)

# Esta función se ejecuta cuando alguien solicita una
URL con el número de pin y acción en él:
@app.route("/<changePin>/<action>")
def action(changePin, action):
    # Convierte el pin de la URL en un entero:
    changePin = int(changePin)
    # Obtener el nombre del dispositivo para el pin
    que se está cambiando:
    deviceName = pins[changePin]['name']
    # Si está activada la parte de acción de la URL,
    ejecuta el código señalado:
```



```
if action == "on":
    # Poner el pin en alto:
    GPIO.output(changePin, GPIO.HIGH)
    # Guardar el mensaje del estado que se pasará a
    la plantilla:
    message = "Turned " + deviceName + " on."
if action == "off":
    GPIO.output(changePin, GPIO.LOW)
    message = "Turned " + deviceName + " off."
if action == "toggle":
    # Leer el pin y póngalo a lo que cambie:
    GPIO.output(changePin, not
GPIO.input(changePin))
    message = "Toggled " + deviceName + "."

    # Para cada pin, leer el estado del pin y
    guardarlo en el diccionario Pins:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    # Junto con el diccionario de Pins, colocar el
    mensaje en el diccionario TemplateData:
    templateData = {
        'message' : message,
        'pins' : pins
    }

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

if __name__ == "__main__":
    app.run(host='10.1.15.94', port=80, debug=True)
```

La explicación que hace dicho código está integrada en el código.

El código de “**main.html**” es el siguiente:

```
<!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 %}

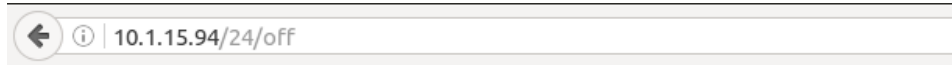
  {% if message %}
  <h2>{{ message }}</h2>
  {% endif %}

</body>
</html>
```

La explicación que hace dicho código está integrada en el código.

- Una vez que se tengan ya los códigos, se ejecuta: “**\$ sudo python weblamp.py**”
- Abro un navegador de internet y escribo la dirección que tiene mi Pc, en este caso es el **10.1.15.94**.

Se debería de mostrar algo como en la imagen 9.



Device Listing and Status

The LED1 is currently off ([turn on](#))

The LED2 is currently off ([turn on](#))

Turned LED1 off.

Imagen 9

Y ya falta pinchar en LED 1 o LED 2 o los dos y se ve que encienden los leds o se apagan y también muestra un mensaje del estado del led debajo.

Nota: En este cuarto ejercicio, en la parte de hardware no he tenido ningún problema en realizarla, porque ha sido reutilización de los ejercicios 1 y 2. La parte de software tampoco ha sido ningún problema porque he seguido un tutorial que venía muy guiado y en el que se explica paso a paso todas las cosas que hay que realizar para hacer los códigos, donde me lié un poco fue a la hora de crearme los archivos y las carpeta porque no lo hice como el tutorial, pero al final si lo hice y lo arreglé.

5. Montar un “**web framework**” para Python (se podrá ejecutar código python desde el servidor web), se empleará leds, un sensor de temperatura tmp36GZ y la placa Arduino para mostrar la temperatura.

En este quinto ejercicio se va a hacer un servidor web en el que se probará ejemplos realizados en Python. En este caso se hará que a través de la placa Arduino y un sensor de temperatura podamos ver en la web la temperatura que hace (es parecido al ejercicio anterior).

Yo he realizado este ejercicio de la siguiente manera:

- Voy a comenzar este ejercicio por la parte de software. Primero, instalo el Flask, con el siguiente comando: “\$ **sudo apt-get install python-pip**” y luego “\$ **sudo pip install flask**”.
- Me creo una carpeta que la he llamado “**Temperatura**” en la que meto dentro un archivo **.py**, que lo he llamado “**weblamp.py**” y dentro de la carpeta “**Temperatura**” también me creo otra carpeta, llamada “**templates**”, en la que meto el “**main.html**”.

El código de “**weblamp.py**” es el siguiente:

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

GPIO.setmode(GPIO.BCM)
arduino=serial.Serial('/dev/ttyACM0',baudrate=9600,ti
meout=1.0)

# Se crea un diccionario llamado Pins para almacenar
el número, el nombre y el estado del pin:
pins = {
    24 : {'name' : 'LED1', 'state' : GPIO.LOW}, 25 :
        {'name' : 'LED2', 'state' : GPIO.LOW}
}

# Establezco cada pin como una salida y lo pongo a
LOW:
for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
```

```
def main():
    # Para cada pin, leer el estado y guardarlo en el
    # diccionario Pins creado antes:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    # Colocar el diccionario de Pins en el diccionario
    # templateData:

    templateData = {'pins' : pins}

    # Pasar los datos de la plantilla a main.html y
    # devolverlo al usuario:
    return render_template('main.html', **templateData)

# Esta función se ejecuta cuando alguien solicita una
# URL con el número de pin y acción en él:
@app.route("/<changePin>/<action>")
def action(changePin, action):
    # Convierte el pin de la URL en un entero:
    changePin = int(changePin)
    # Obtener el nombre del dispositivo para el pin que
    # se está cambiando:
    deviceName = pins[changePin]['name']
    # Si está activada la parte de acción de la URL,
    # ejecuta el código señalado:
    if action == "on":
        # Poner el pin en alto:
        GPIO.output(changePin, GPIO.HIGH)
        # Guardar el mensaje del estado que se pasará a
        # la plantilla:

        message = "Turned " + deviceName + " on."
    if action == "off":
        GPIO.output(changePin, GPIO.LOW)
        message = "Turned " + deviceName + " off."
    if action == "toggle":
        # Leer el pin y póngalo a lo que cambie:
        GPIO.output(changePin, not GPIO.input(changePin))
    message = "Toggled " + deviceName + "."

    # Para cada pin, leer el estado del pin y guardarlo
    # en el diccionario Pins:
    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)
```

```
@app.route("/<temper>")
def leerTemp(temper):
    # Junto con el diccionario de Pins, colocar el
    mensaje en el diccionario TemplateData:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    if temper == "L":
        temperatura=''
        arduino.write("N")
        time.sleep(0.8)
        while arduino.inWaiting() != 0:
            temperatura += arduino.read(1)

    templateData = {
        'temperatura' : temperatura,
        'pins' : pins
    }

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

if __name__ == "__main__":
    app.run(host=10.1.15.94', port=80, debug=True)
```

La explicación que hace dicho código está integrada en el código.

El código de “**main.html**” es el siguiente:

```
<!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 %}

    <h2>Actualiza temperatura: (<a
    href="/L">Leer</a></h2>
    El valor es: {{temperatura}}

</body>

```

La explicación que hace dicho código está integrada en el código.

- Una vez hecha la parte de software me paso al montaje del circuito. Los dos leds los he conectado a los pines 24 y 25 y la salida que va al GND al pin 6 (esto es en Raspberry). Además, le he conectado un cable TTL a la placa Raspberry Pi que va a la placa Arduino, porque no me dejaba instalar Arduino en el Ubuntu Mate, por lo que lo he hecho en el otro ordenador donde tenía conexión vía SSH (ver ejercicio 3 como lo he realizado).
- Los 3 cables que tiene el TTL los he conectado: el Amarillo, que es RX -> Pin 1 (TX) de Arduino (lo tiene configurado por defecto), el Naranja, que es TX -> Pin 0 (RX) de Arduino (lo tiene configurado por defecto) y el Negro, que es GND -> Pin GND de Arduino.

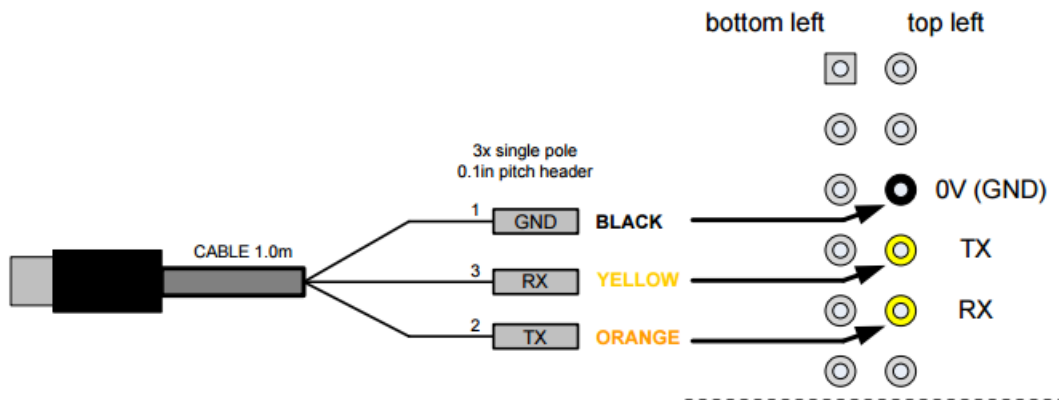


Imagen 10

- También, en la placa de Arduino he conectado el sensor de temperatura tmp36GZ. La pata de la izquierda va a positivo, la pata de la derecha va a negativo (GND) y la pata central va al ping analógico A0.

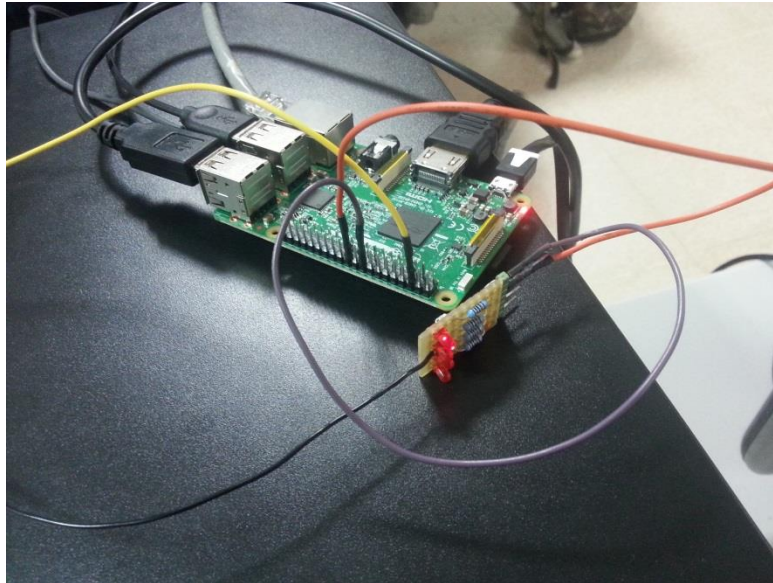


Imagen 11

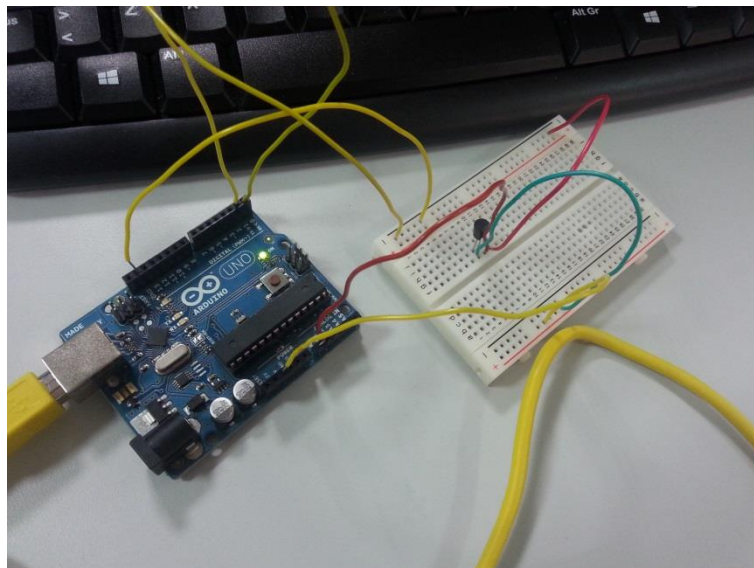


Imagen 12

- Una vez montado el circuito, me paso a escribir el código en Arduino para que me calcule y lea la temperatura.

El código es el siguiente:

```
int LDRPin = 0;

void setup () {
  Serial.begin(9600);
}

void loop () {
  if(Serial.available()) {

    char c = Serial.read();
```

```
int value = analogRead(LDRPin);  
float millivolts = (value / 1023.0) * 5000;  
float celsius = millivolts / 10;  
Serial.print(celsius);  
Serial.println(" C");  
}  
}
```

- Se carga el código en la placa de Arduino y una vez cargado, se ejecuta en la ventana de símbolos: “**\$ sudo python weblamp.py**”.
- Abro un navegador de internet y escribo la dirección que tiene mi Pc, en este caso es el **10.1.15.94**.

Me saldrá un mensaje parecido al ejercicio anterior para encender o apagar los led (LED 1 y LED 2) y abajo uno para la temperatura que cuando se activa muestra el valor de temperatura.

Nota: En este quinto ejercicio, en la parte de hardware no he tenido ningún problema en realizarla, aunque ha sido un poco más laboriosa, porque ha visto que utilizar la placa Raspberry Pi y la placa Arduino para poder mostrar la temperatura en la web framework (los montajes están arriba explicados). En la parte de software tampoco he tenido problemas, porque ha sido los códigos parecidos como los del ejercicio anterior (**weblamp.py** y **main.html**) lo único que ha visto que añadirle ha sido lo de la temperatura para que la leyera, y el programa en Arduino que ha visto que hacer también.

Conclusión

Con la realización de estas dos prácticas de Raspberry Pi, he aprendido a usar la placa, a instalarle un Sistema Operativo y a realizar algunos ejercicios.

Algunas cosas que me han parecido interesantes en las prácticas son: el encendido y apagado de los leds a través de la ventana de comandos y la comunicación vía puerto serie, que ha habido que hacerlo a través de un programa en Python.

También, conectarse desde otro PC a la Raspberry gracias a la conexión vía SSH y montar servidores web a través del cual se puede enviar datos a los actuadores, por ejemplo, encendido y apagado de leds y cálculo de la temperatura con Arduino, etc.

Todos estos ejercicios se ejecutan desde la ventana de símbolos, no tiene entorno de programación como Arduino o ZPUino.

Glosario

- **ARM:** es una arquitectura RISC (conjunto de instrucciones simples) de 32 bits y recientemente con la llegada de su versión V8-A también de 64 bits desarrollada por ARM Holdings.
- **Ethernet:** es un estándar de redes de área local para computadores con acceso al medio por detección de la onda portadora y con detección de colisiones. Define las características de cableado y señalización de nivel físico y los formatos de tramas de datos del nivel de enlace de datos del modelo OSI.
- **Flask:** es un microframework para python basado en Werkzeug (librería python).
- **GND (toma de tierra):** se emplea en las instalaciones eléctricas para llevar a tierra cualquier derivación indebida de la corriente eléctrica a los elementos que puedan estar en contacto, ya sea directa o indirectamente, con los usuarios de aparatos de uso normal, por un fallo del aislamiento de los conductores activos, evitando el paso de corriente al posible usuario.

- **GPIO:** es un pin genérico en un chip, cuyo comportamiento se puede programar por el usuario en tiempo de ejecución.
- **HDMI (Interfaz multimedia de alta definición):** es una norma de audio y video digital cifrado sin compresión apoyada por la industria para que sea el sustituto del euroconector.
- **Jack:** es un conector de audio analógico que se utiliza para conectar micrófonos, auriculares y otros sistemas de señal analógica a dispositivos electrónicos aunque sobre todo audio.
- **Led:** es un diodo que emite luz.
- **Micro SD:** es una tarjeta de memoria flash. Se suele utilizar con el adaptador SD para cámaras digitales, teléfonos móviles, etc.
- **Resistencia:** oposición que tienen los electrones al moverse a través de un conductor.
- **Sensor temperatura:** 2 terminales de alimentación y un tercero de señal.
- **SoC:** circuito integrado que incorpora gran parte de los componentes de un ordenador o cualquier otro sistema informático o electrónico.
- **SSH:** es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos.
- **TTL:** es una familia de cables convertidores serie USB a TTL que incorporan USB a UART serie, que gestiona todas las señales y protocolos USB.
- **Web Framework:** es un framework (conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve para enfrentar y resolver nuevos problemas de índole similar) diseñado para apoyar el desarrollo de sitios web dinámicos, aplicaciones web y servicios web. Este tipo de frameworks intenta aliviar el exceso de carga asociado con actividades comunes usadas en desarrollos web.

Bibliografía

- **Información de Raspberry Pi**

- <https://www.raspberrypi.org/>
- https://es.wikipedia.org/wiki/Raspberry_Pi
- <https://ubuntu-mate.org/raspberry-pi/>

- **Características de la placa**

- <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>
- <https://www.pccomponentes.com/raspberry-pi-3-modelo-b>

- **Instalación del Sistema Operativo a la placa**

- <https://coria.dte.us.es/~bellido/>
- <https://www.raspberrypi.org/documentation/installation/installing-images/linux.md>

- **Ejercicio 1**

- <https://geekytheory.com/tutorial-raspberry-pi-gpio-parte-1-control-de-un-led/>

- **Ejercicio 2**

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

- **Ejercicio 4**

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