



2016/17

MEMORIAS DE LABORATORIO

Arduino, Papilo (ZPUino), Raspberry Pi

Rafael Torres Malpartida

Contenido

Arduino Contenido	1
Software	2
Hardware	2
Arduino UNO + Protoboard (regleta)	3
Instalando el entorno de desarrollo y primeras pruebas	3
Controlar el encendido-apagado del LED desde el PC vía puerto serie.....	4
Encendido apagado de un led mediante pulsador empleando las interrupciones en arduino	5
Controlar el ángulo de posicionamiento del servo desde el PC.	7
Control de la velocidad de giro de un motor de continua en doble sentido con un potenciómetro.	9
Impresión en una pantalla LCD: imprimir información que envía el pc vía serie	10
Empleando el sensor de temperatura tmp36GZ diseñar un termómetro con la temperatura en el display LCD	11
Plataformas de desarrollo Papilio	13
Diseñar circuitos para implementarlos en la FPGA: Inversor	15
Controlar el encendido apagado de un Led desde el PC vía puerto serie	18
Convirtiendo la placa papilio en un Analizador Lógico	20
Raspberry Pi.....	23
1ª Sesión	24
Manejo de GPIOs desde Python.....	26
weblamp.py	30
main.html	31
Servidor web temperatura: Conectar Arduino	33

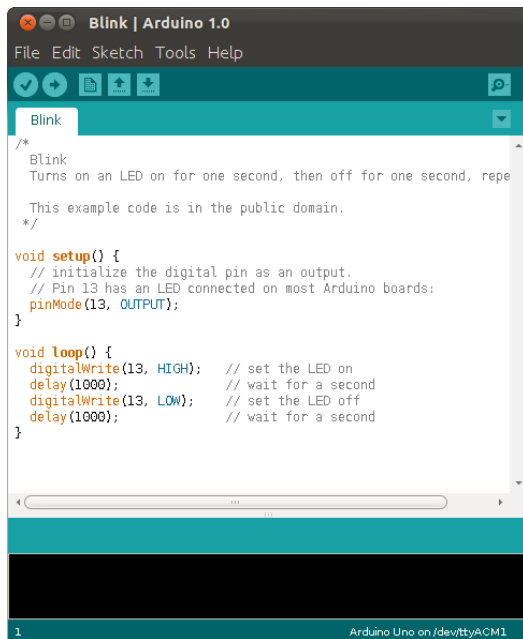
Plataforma Arduino

Arduino es una plataforma de desarrollo de hardware abierta (open Hardware) relativamente barata (10 – 20 €). Existen multitud de variantes de esta placa (Uno, Leonardo, Yún, Ethernet, Mega, Nano...).

Actualmente hay 2 compañías principales entorno es este producto. Arduino y Genuino fruto de la separación de uno de los fundadores. En la práctica, Arduino y Genuino son casi iguales, aunque hay nuevas versiones diferentes en hardware y software.

Esta plataforma está basada en:

Software

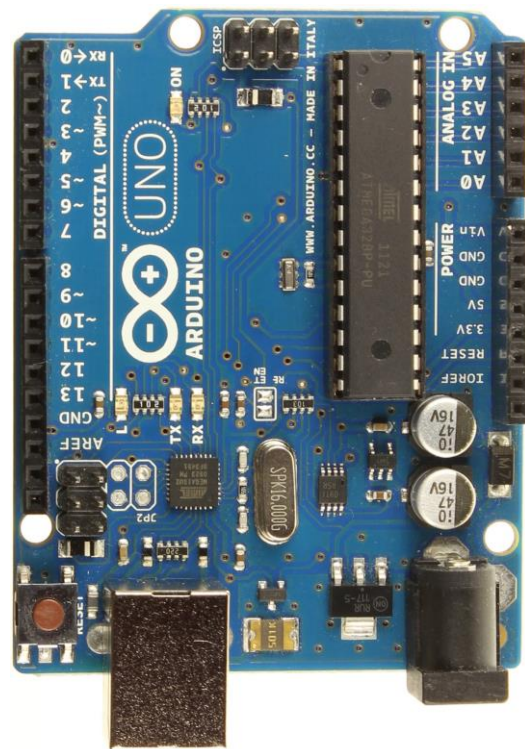


Entorno de desarrollo basado en *processing* (**sketches**) con una sintaxis similar a **c**, muy sencillo de usar y gratuito. Dispone de muchos ejemplos de toda clase resueltos.

La estructura que se observa en la foto presenta una función de inicialización **setup** y el clásico bucle de este tipo de sistemas empotrados.

La comunidad de Arduino es muy grande y hay a disposición del desarrollador una gran cantidad de librerías y ejemplos para el manejo de sus periféricos

Hardware



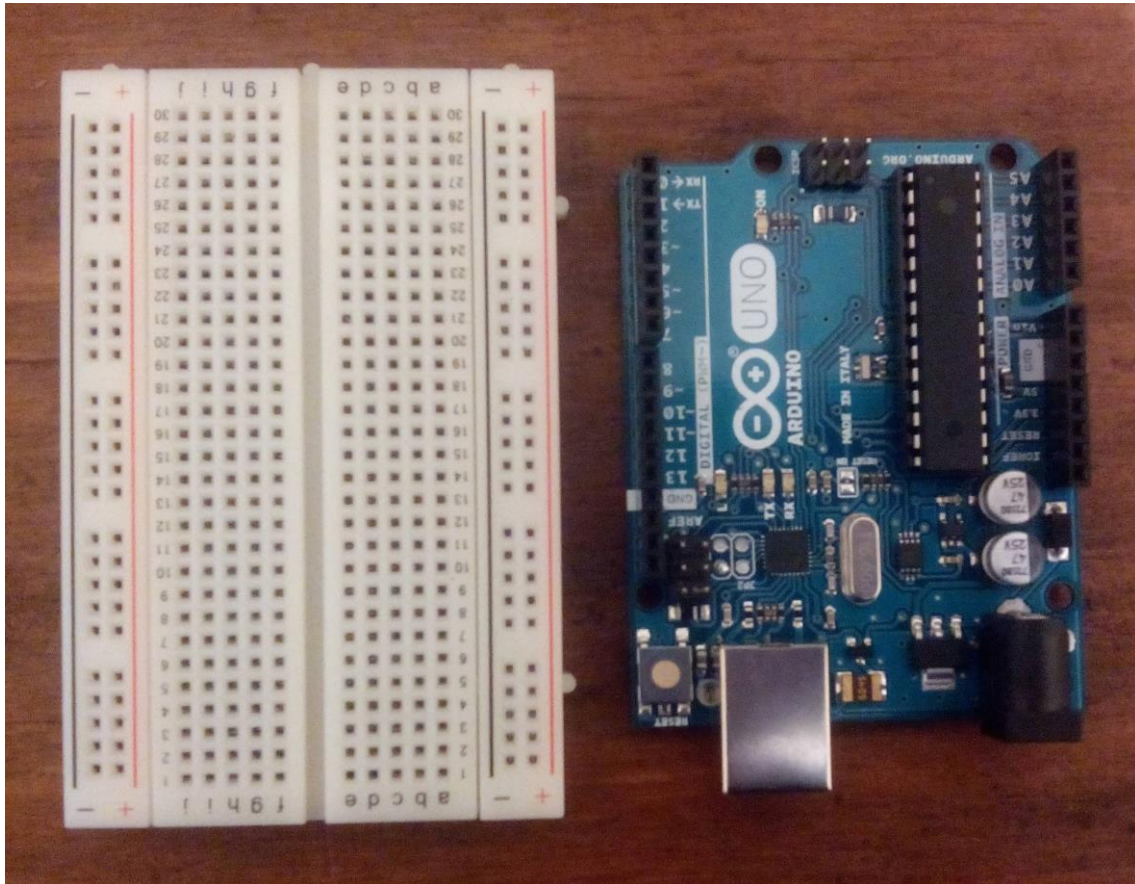
Inicialmente basada en AVR 8 Bits, hay versiones de hasta 32 bits

No necesita depurador

Pines de entrada/salida analógicos y digitales que pueden también conectarse a placas de expansión, llamados *shields*, ethernet, wifi, motor... Reloj a 16 MHz Gran gama de sensores y dispositivos, hicimos uso de algunos de ellos disponibles en Arduino Starter Kit:

Potenciómetros, Fotorresistencias, sensores de inclinación o temperatura, transistores, Puente en H, Relés, Leds, Motores...

Arduino UNO + Protoboard (regleta)



Concretamente nuestro dispositivo es un Arduino Uno con ATMEGA 328P, 32K de memoria Flash y 14 GPIO (6 pueden ser PWM), 6 entradas analógicas y reloj a 16Mhz

Instalando el entorno de desarrollo y primeras pruebas

Nos dirigimos a <https://www.arduino.cc/en/Main/Software>, Descargamos la versión para nuestro sistema operativo. En prácticas hemos usado Ubuntu, 64 bits. Descargamos, descomprimos en el directorio home y ejecutamos *ubuntu-setup.sh*, esto creará el *path* necesario en el sistema y a partir de ahora podremos ejecutarlo desde el lanzador: tecla Windows y empezamos a escribir Arduino.

Conectamos la placa con el cable USB que viene en el kit y a partir de este momento podemos empezar a desarrollar con nuestra placa. Haremos unas primeras pruebas con sketches básicos.

Blink: Conceptos: estructura general del programa; escritura en pines digitales; retrasos

Fade: Conceptos: escritura analógica;

AnalogInOutserial: Conceptos: escritura en puerto serie, hipertextual; lectura analógica; adecuación valor de lectura a valor de escritura.

Puede ocurrir el problema de que al querer subir a la placa no tengamos permisos para usar el puerto en el que la tenemos conectada.

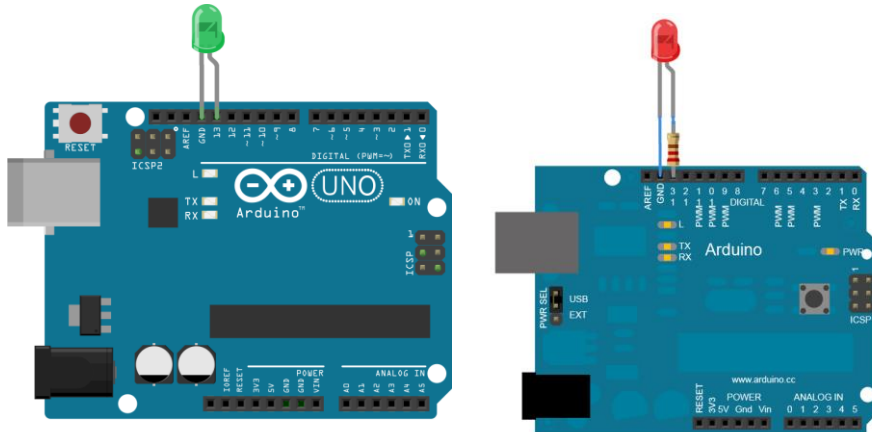
Arduino IDE error - avrdude: ser_open(): can't open device "/dev/ttyACM0": Permission denied

Este error lo solucionamos o bien cerrando el IDE y volviéndolo a abrir con el comando **sudo** antes o bien dando permisos al usuario sobre este puerto con:

```
$ sudo usermod -a -G dialout <username>
```

```
$ sudo chmod a+rw /dev/ttyACM0
```

Controlar el encendido-apagado del LED desde el PC vía puerto serie



Montamos un led en el pin digital 13 de nuestro Arduino.

Conectado al USB, arduino funciona como un puerto Serial a través de este con el nombre en mi caso: **tttyACM0**

Por tanto, nos hará falta desarrollar un sketch para la placa con el comportamiento que tendrá ante los estímulos en el puerto serie y código Python en el PC que se comunique con la placa a través de este puerto serie emulado:

```
const int ledPin = 13;
int incomingByte;

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

void loop() {
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    if (incomingByte == 'H') {
      digitalWrite(ledPin, HIGH);
    }
    if (incomingByte == 'L') {
      digitalWrite(ledPin, LOW);
    }
  }
}
```

Luego, en nuestro entorno Linux, haciendo uso de la consola de comandos (Ctrl+Alt+T) podemos desarrollar el código en Python. Usando por ejemplo nano o gedit editamos un archivo con extensión .py que luego ejecutaremos desde la consola.

```
nano serial.py
```

Editamos el archivo con el código:

```
import serial

arduino = serial.Serial('/dev/ttyACM1', 9600)
```



```

print("Starting!")

while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino
    if comando == 'H':
        print('LED ENCENDIDO')
    elif comando == 'L':
        print('LED APAGADO')

arduino.close() #Finalizamos la comunicacion

```

Ejecutar con:

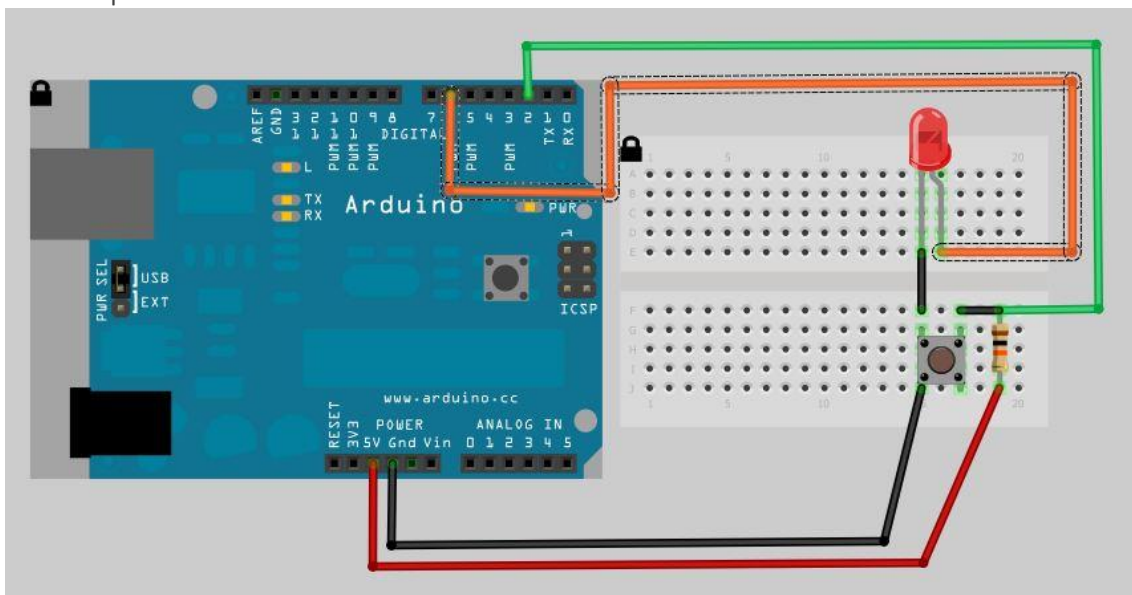
```
python serial.py
```

Posibles problemas:

No hemos elegido nuestra placa correctamente: Arduino/Genuino Uno

No hemos elegido correctamente el puerto: Los puertos de serie emulados comienzan por USBtty...

Encendido apagado de un led mediante pulsador empleando las interrupciones en arduino



```

int button = 2;
int led = 6;
int value = LOW;

void setup (){

    pinMode (button,INPUT);
    pinMode (led,OUTPUT);
    Serial.begin(9600);
    attachInterrupt(0,intFunction,FALLING);
}

```

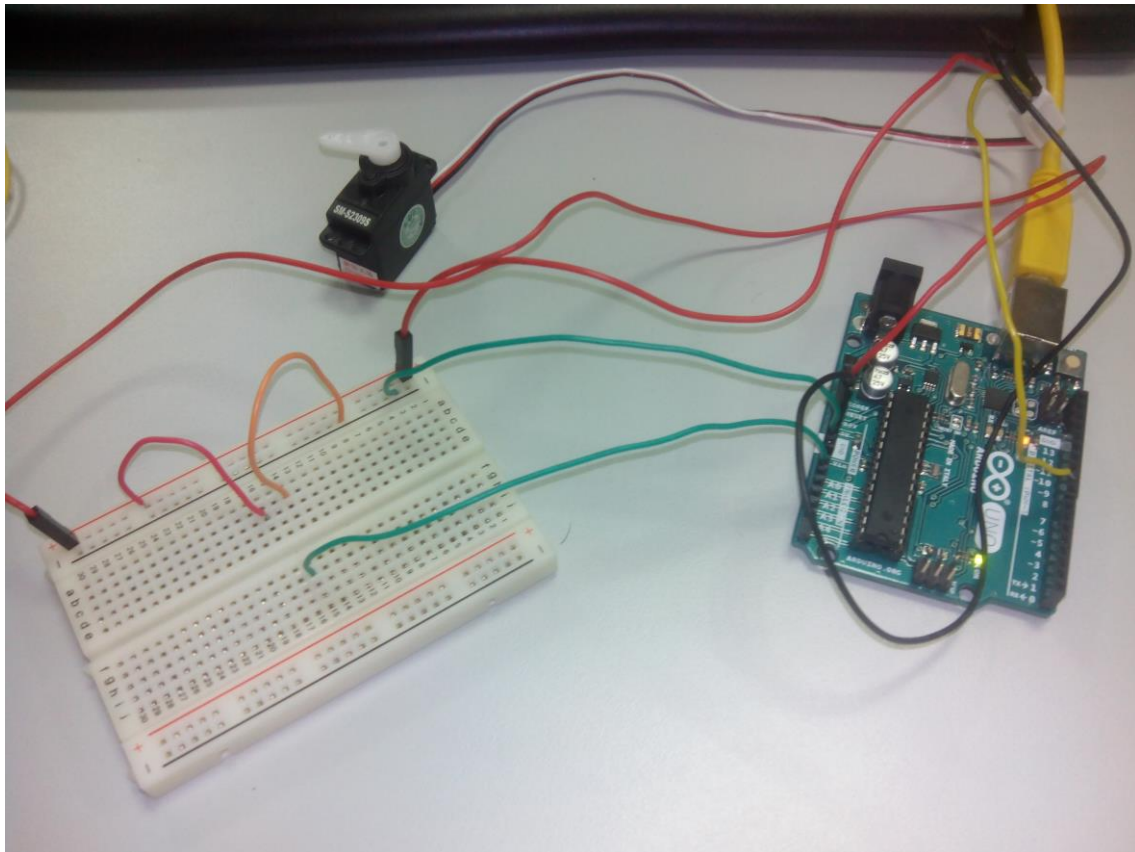
```
}  
  
void loop() {  
    digitalWrite(led, value);  
}  
  
void intFunction () {  
    value=!value;  
    delay(500);  
}
```

Controlar el ángulo de posicionamiento del servo desde el PC.

Enviando el valor de ángulo (desde 0 a 180 grados) a través del puerto serie.

Programa en Python en PC que envíe el valor del ángulo a través del puerto serie.

Utilizar `parseInt()` de la librería serial



```
#include <Servo.h>

Servo myservo; // create servo object to control a servo

int potpin = A0; // analog pin used to connect the potentiometer
int val; // variable to read the value from the analog pin
int incomingByte;

void setup() {
  Serial.begin(115200);
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
  //val = analogRead(potpin);
  if (Serial.available() > 0) {
    val = Serial.parseInt();
    //val = map(val, 0, 1023, 0, 180);
    myservo.write(val);
  }
}
```



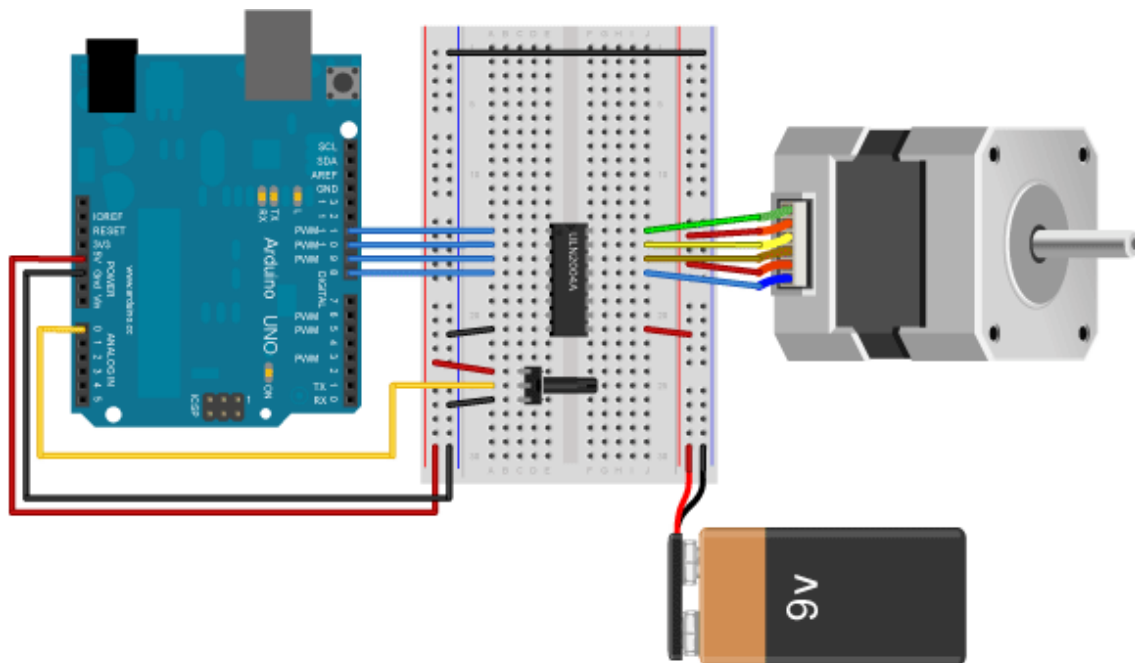
```
}
```

Python

```
import serial
ser = serial.Serial('/dev/ttyACM0', 115200)
print("Starting!")
while True:
    angle = raw_input('Introduce ángulo: ')
    ser.write(angle)
ser.close() #Finalizamos la comunicacion
```

Control de la velocidad de giro de un motor de continua en doble sentido con un potenciómetro.

Alimentación del motor con 9v. Driver circuito driver en puente H (aporta potencia y cambia polaridad), mitad señal pot, giro pos, otra mitad señal pot., giro neg



```
int pin2=9; //entrada 2 del L293D
int pin7=10; //entrada 7 del L293D
int pote=A0; // potenciómetro
int valopote; //variable que recoge el valor del potenciómetro
int pwm1; //variable del PWM1
int pwm2; //variable del PWM2

void setup(){ //inicializamos los pines de salida
    pinMode(pin2,OUTPUT);
    pinMode(pin7,OUTPUT);
}
void loop(){
    //almacenamos el valor del potenciómetro en la variable
    Valorpote = analogRead(pote);
    //Como la entrada analógica del Arduino es de 10 bits, el rango va
    de 0 a 1023.
    //En cambio, la salidas del Arduino son de 8 bits, quiere decir,
    rango entre 0 a 255.
    pwm1=map(valorpote,0,1023,0,255);
    pwm1=map(valorpote,0,1023,255,0);
    analogWrite(pin2,pwm1);
    analogWrite(pin7,pwm2);
}
```

Impresión en una pantalla LCD: imprimir información que envía el pc vía serie

Pantalla LCM1602C, basada en controlador Hitachi HD44780 o compatible.

Librería arduino: LiquidCrystal

lcd_serial.py

```
import serial

arduino=serial.Serial('/dev/ttyACM0',baudrate=9600)
cadena=""

while True:

    aux = raw_input("Mensaje: ")
    arduino.write(aux)

    while arduino.inWaiting() > 0:
        cadena += arduino.readline()
        print cadena.rstrip('\n')
        cadena = ""

arduino.close()
```

lcd_serial.ino

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

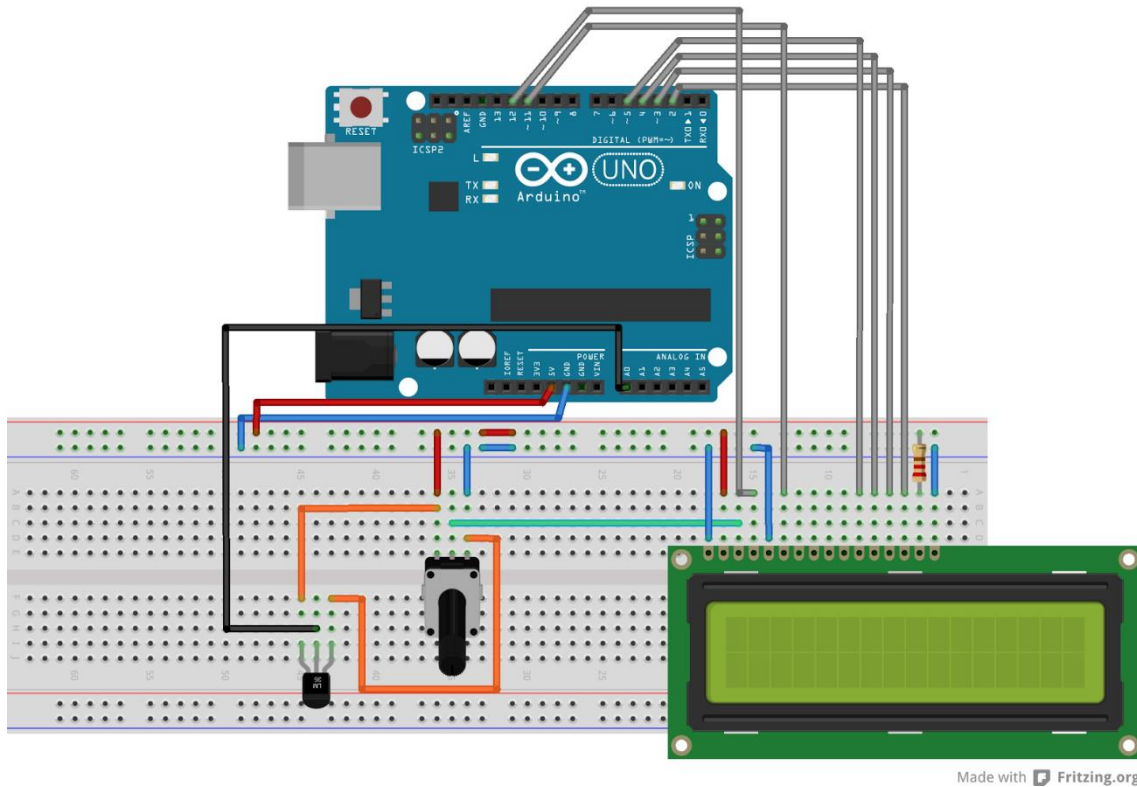
void setup() {
    Serial.begin(9600);
    lcd.begin(16, 2);
    lcd.clear();
}

void loop() {

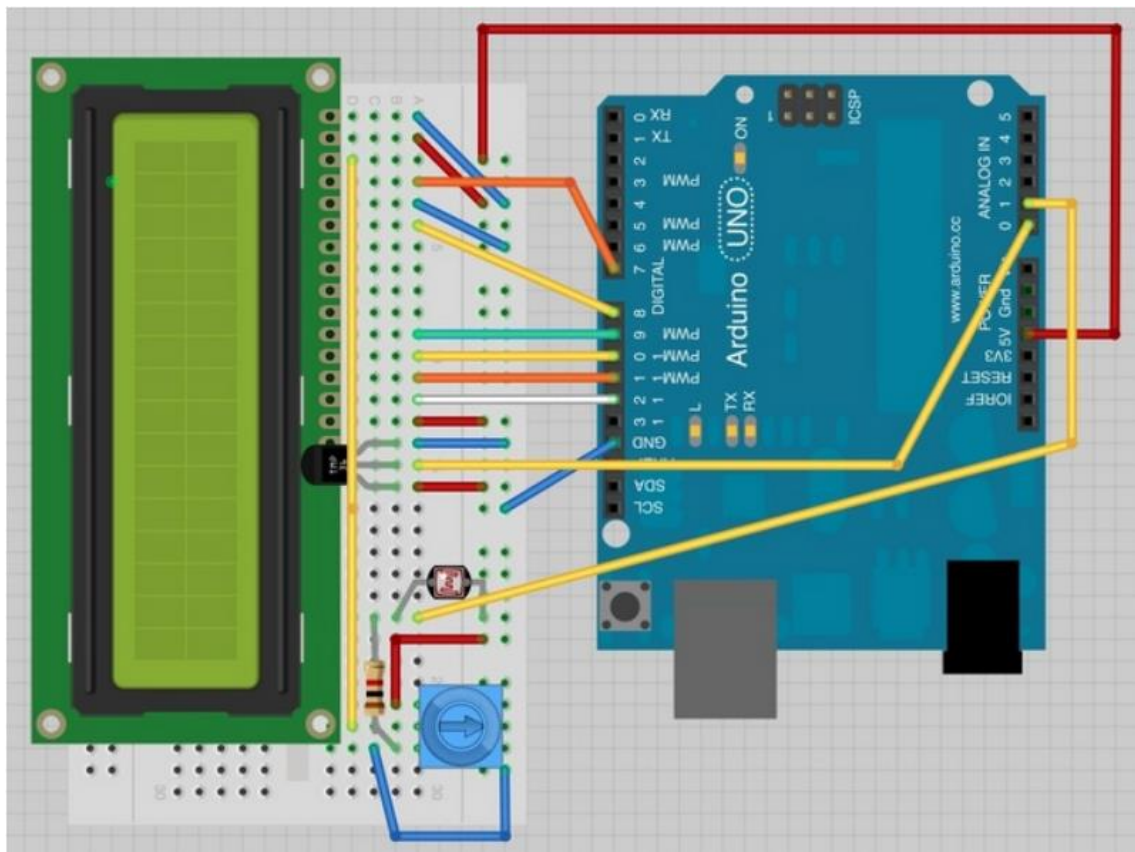
    if (Serial.available()) {

        while (Serial.available() > 0) {
            char c = Serial.read();
            lcd.write(c);
        }
    }
}
```

Empleando el sensor de temperatura tmp36GZ diseñar un termómetro con la temperatura en el display LCD



Fallo



Correcto

```

#include <LiquidCrystal.h>
int tempPin = 0;
int lightPin = 1;

// BS E D4 D5 D6 D7
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
void setup() {
  lcd.begin(16, 2);
}

void loop() {
  // Temperatura en C
  int tempReading = analogRead(tempPin);
  float tempVolts = tempReading * 5.0 / 1024.0;
  float tempC = (tempVolts - 0.5) * 100.0;
  float tempF = tempC * 9.0 / 5.0 + 32.0;
  //Limpiar
  lcd.clear();

  lcd.print("Temp C ");
  lcd.setCursor(8, 0);
  lcd.print(tempC);
  // mostrar la luz
  int lightReading = analogRead(lightPin);
  lcd.setCursor(0, 1);
  // -----
  lcd.print("Luz ");
  lcd.setCursor(6, 1);
  lcd.print(lightReading);
  delay(500);
}

```

El primer montaje calentaba el sensor de temperatura

PLATAFORMA ZPUINO

Objetivos mínimos del trabajo:

Conocer la plataforma PAPILIO

Instalar y configurar el entorno de trabajo de papilio: DESIGN LAB

Conocer que se puede hacer desde DESIGN LAB sobre las placas PAPILIOS:

- Diseñando circuitos a nivel de captura de esquemáticos e implementarlos en la FPGA
- Cargar el SoC ZPUINO
- Desarrollar Sketches para ZPUINO
- Configurar la Placa Papilio para que funcione como un analizador lógico
- Modificar el SoC ZPUINO añadiendo algún nuevo periférico y desarrollando algún sketch que lo utilice

Plataformas de desarrollo Papilio

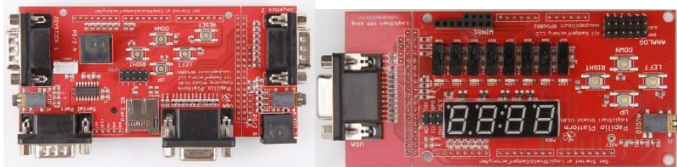
Son placas de desarrollo hardware abiertas basadas en FPGAs de XILINX:

Papilio one – SPARTAN3E (250 y 500)

Papilio Pro – SPARTAN6

Papilio duo – SPARTAN6 y atmega32U4

WINGS: son placas de expansión adaptadas al conector de expansión de papilio

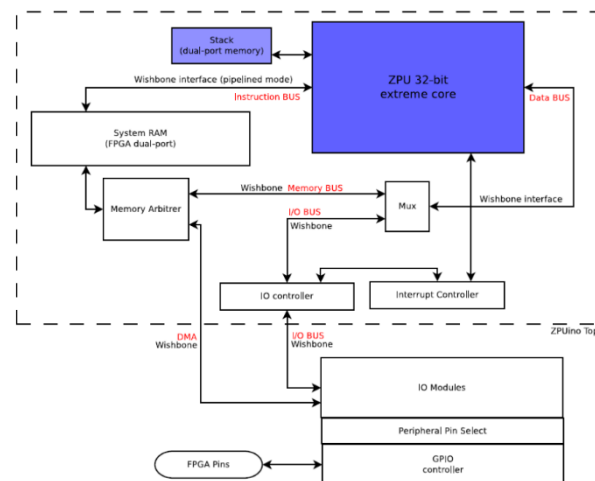


Computing Shield: hardware de un PC típico: VGA, Stereo jacks, ...

LogicStart Shield: Display 7 segmentos x 4, VGA, botones...

Existen muchas más aplicaciones muy diferentes

ZPUino: Es un SoC implementado en VHDL (soft core) basado en el microprocesador ZPU de Zylins.



Se puede trabajar con ZPUino de modo equivalente a Arduino ya que se ha adaptado el IDE (entorno de desarrollo software de arduino) para ser compatible con ZPUino. Como veremos, el entorno también permitirá modificar el SoC a nivel hardware para añadir nuevos periféricos al mismo.



Lo descargamos del servidor personal: <http://10.1.15.78/~bellido>, también podemos descargarlo de <http://forum.gadgetfactory.net/index.php?files/file/236-papilio-designlab-ide/>

Descomprimos por ejemplo en la carpeta home

Entramos en el directorio y encontramos un fichero llamado Ubuntu-setup.sh, una modificación del fichero de instalación original para Ubuntu

Es necesario instalar JRE para Ubuntu

```
sudo apt-get install default-jre  
es conveniente realizar antes: sudo apt-get update  
Ahora ya podemos ejecutar el IDE:
```

Si no estamos aún en el directorio donde descomprimos volvemos a entrar y ejecutamos:

```
sudo ./Designlab
```

Se nos abrirá el entorno con el aspecto de la figura de arriba.

Ahora es el momento de configurar nuestra placa y puerto: Papilio One 500kb - ZPUino



En el sistema es necesario una instalación de ISE Design Suite, sin ella no podremos realizar algunas de las funciones que se mencionaron como sería modificar el SoC y añadirle periféricos.

EN la página de Xilinx, podemos descargar la versión de Linux 14.7 actualmente. Cuidado ocupa casi 7GB, se recomienda un gestor de descarga.

Una vez hecha esta comprobación, en el entorno de Designlab, Archivo, preferencias y comprobamos que la ruta de ISE Design Suite es /opt/Xilinx/14.7/ISE_DS/ISE/bin/linux64/

Diseñar circuitos para implementarlos en la FPGA: Inversor

La entrada del inversor la vamos a conectar a un botón y la salida a un LED, de esta forma veremos como se muestra la salida invertida.

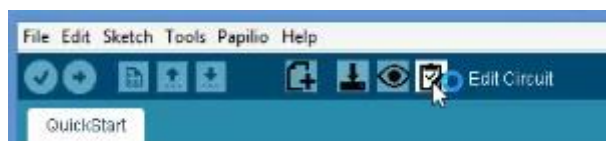
Desarrollaremos el esquema:



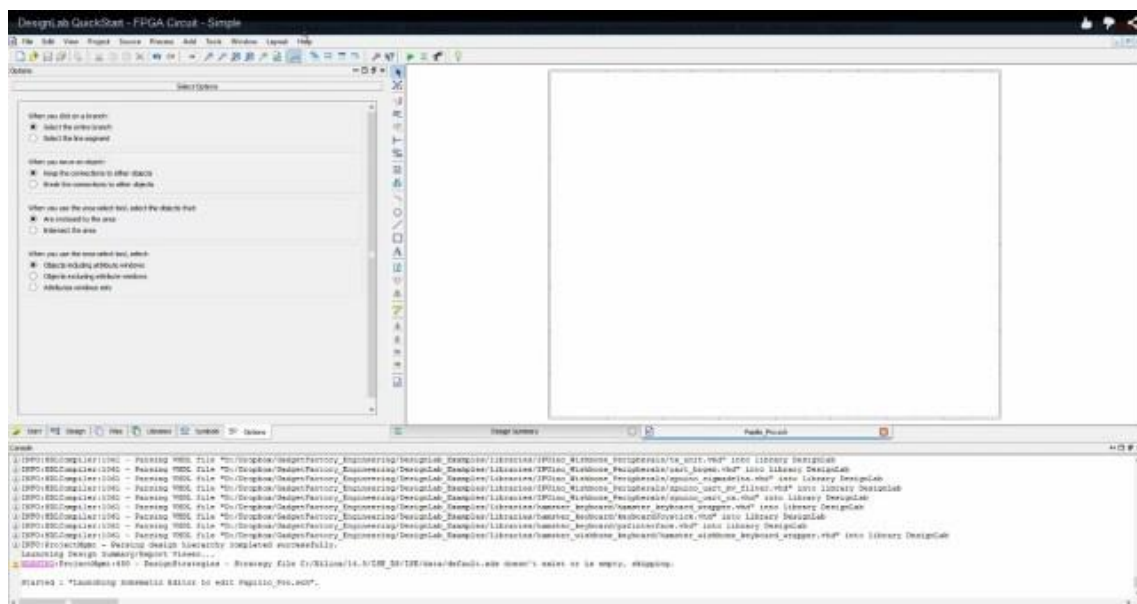
Nuevo circuito FPGA

Se nos abrirá de nuevo la ventana con el template. Estos son de solo lectura y debemos guardarlo en un nuevo directorio para poder editarlo. EL sketch se guardará con el directorio.

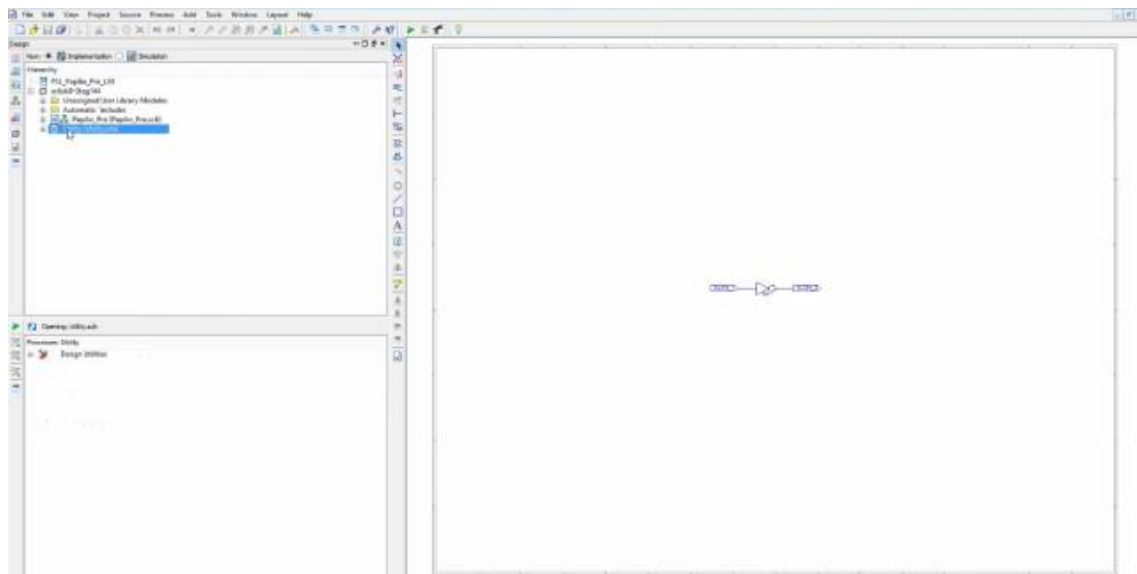
Ahora podemos editar el circuito:



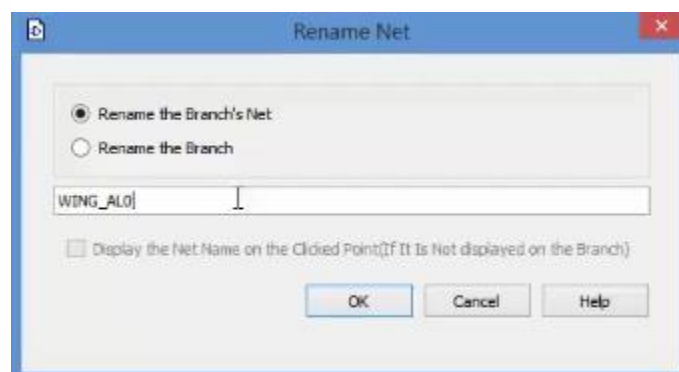
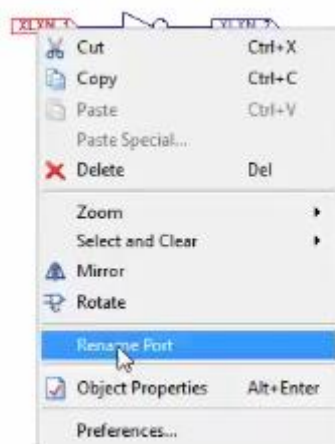
Esto nos abrirá el editor de esquemas de ISE Design Suite, buscamos el esquema de nuestro Papilo, un fichero .sch y hacemos doble clic en el:



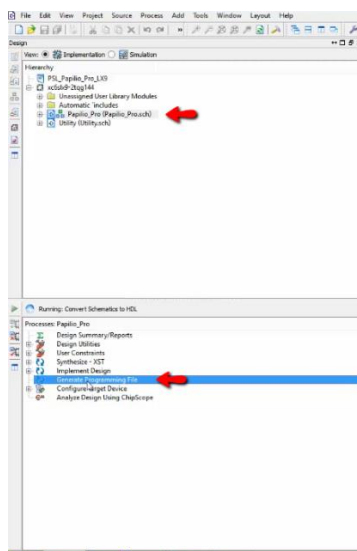
Hacemos clic en la pestaña de Symbolos y con el buscador vamos a localizar el inversor, escribiendo el principio de la palabra "inv" nos aparecerá. Hacemos doble clic y ya lo tenemos añadido:



Vamos a conectar 2 marcadores a los terminales. Vamos a Design lab-> Utility para ver el diagrama de conexiones y editar los wings, vamos a usar AL0 y AL1



Sintetizamos el circuito:



Y vemos en la ventana de consola la compilación: “Process ‘Generate Programming File’ completed successfully

Ahora solo tenemos q cargar el circuito en la placa y tendremos nuestro nuevo SoC



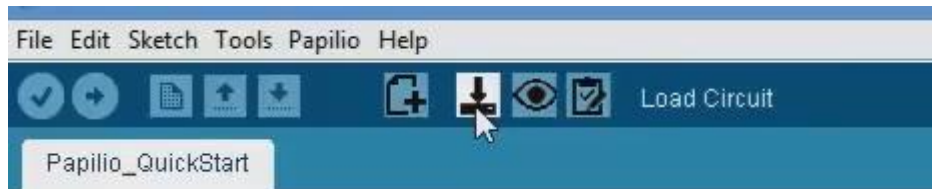
PARA CARGAR CORRECTAMENTE EL BITFILE HAY QUE HACER UNA MODIFICACIÓN EN EL NOMBRE DEL FICHERO QUE GENERA ISE:

- Carpeta: <proyecto>/circuit/500K/
- 1.- Borrar papilio_one_500k.bit
- 2.- Renombrar Papilio_One_500K.bit a papilio_one_500k.bit

Controlar el encendido apagado de un Led desde el PC vía puerto serie

Para este ejemplo vamos a cargar el SoC ZPUino y a partir de ahí podremos generar sketches para este procesador. Como siempre abrimos nuestro Design lab y elegimos la configuración adecuada de placa y puerto: Papilio One 500K ZPUino y puerto USB com emulado.

Cargamos el SoC



```
Done burning bitfile.
Found Macronix Flash (Pages=32768, Page Size=256 bytes, 67108864 bits).
Erasing :
Doing Partial Erase
.....Ok
Verifying :
.....Pass
Programming :
.....Ok
Verifying :
.....Pass
Using devlist.txt
Done.
SPI execution time 19327.6 ms
USB transactions: Write 6856 read 6687 retries 0
JTAG chainpos: 0 Device IDCODE = 0x24001093 Desc: XC6SLX9

Using devlist.txt
ISC_Done      = 0
ISC_Enabled   = 0
House Cleaning = 1
DONE          = 0
```

```
HardwareSerial mySerial1(WishboneSlot(5));
int led = 32;

void setup() {
  // put your setup code here, to run once:

  pinMode(led, OUTPUT);
  mySerial1.begin(9600);
}

void loop() {

  if(mySerial1.available()){
    char c = mySerial1.read();
```

```
        if(c == 'H'){
            digitalWrite(led, HIGH);
        }else if(c == 'L'){
            digitalWrite(led, LOW);
        }
    }
}
```

Python

```
import serial

arduino = serial.Serial('/dev/ttyACM1', 9600)

print("Starting!")

while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino
    if comando == 'H':
        print('LED ENCENDIDO')
    elif comando == 'L':
        print('LED APAGADO')

arduino.close() #Finalizamos la comunicacion
```


Convirtiendo la placa papilio en un Analizador Lógico

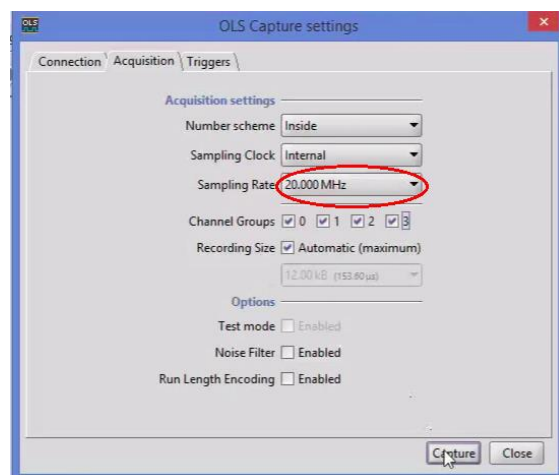
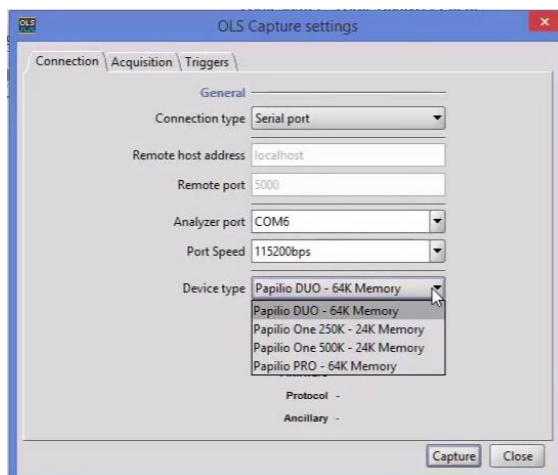
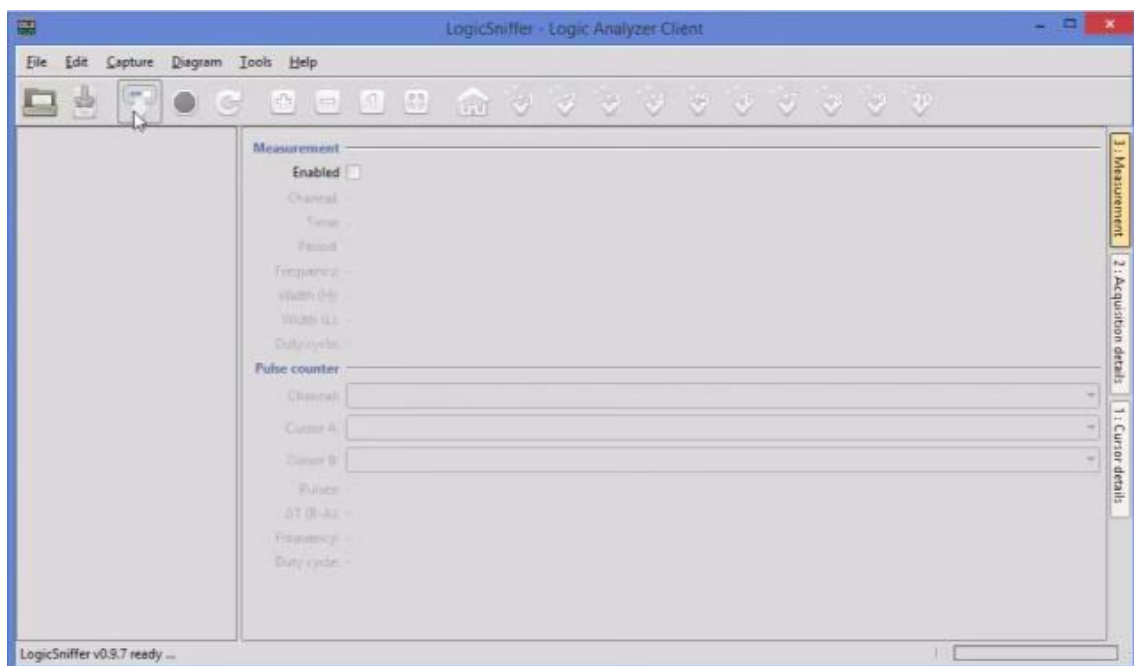
Vamos a realizar el siguiente tutorial;

<http://gadgetfactory.net/learn/2015/07/30/designlab-using-papilio-as-stand-alone-logic-analyzer/>

Designlab ya trae esta funcionalidad por defecto, para utilizarla, basta con conectar la placa y especificarla como en las practicas anteriores:

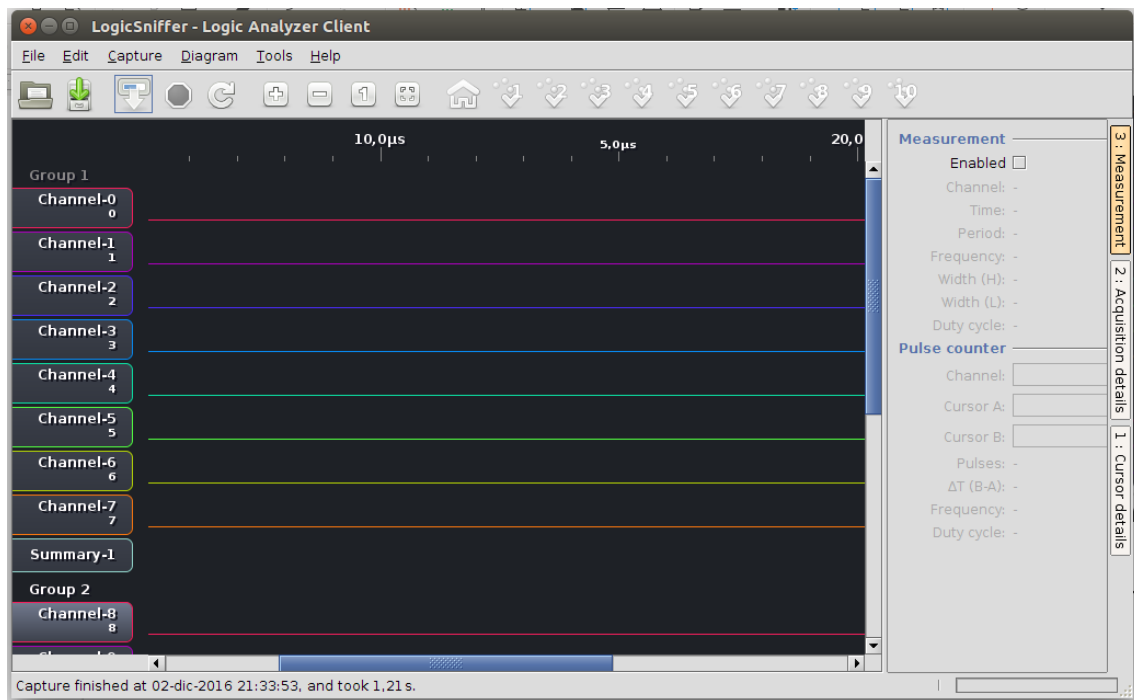


Una vez realizado nos saldrá un mensaje de confirmación y otro con información de los pines. Seguidamente se nos abrirá el software para el analizador que debemos configurar:

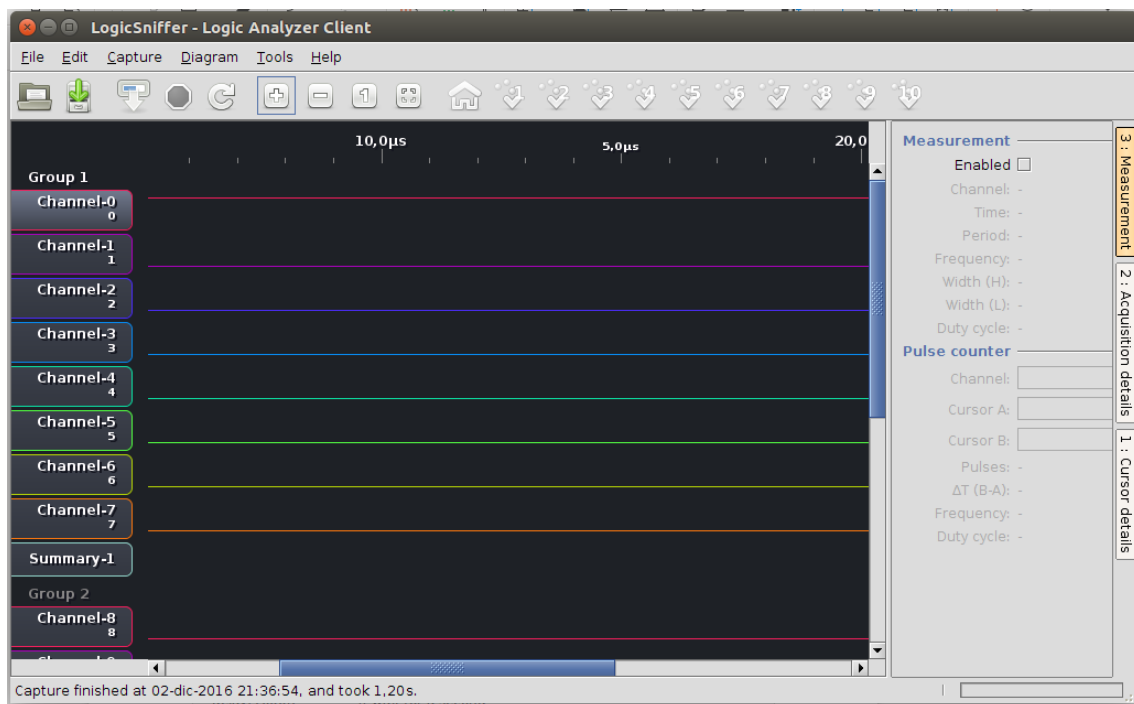


Papilio One 500kb,

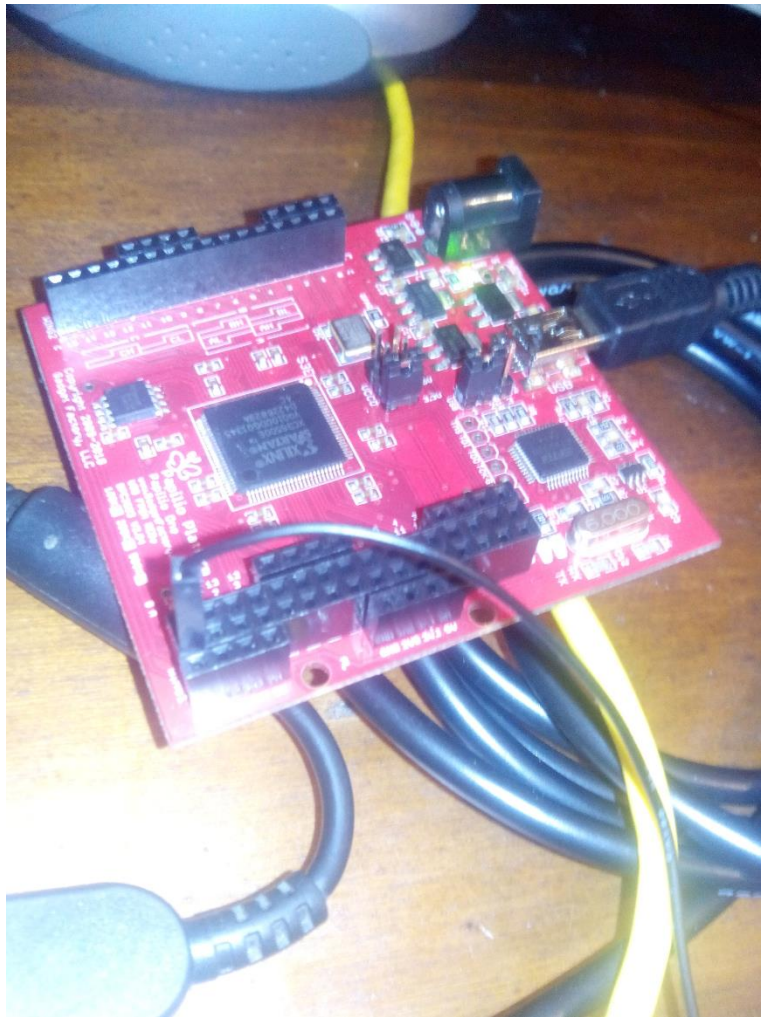
Clicamos en capture y comenzará el analizador:



Ahora es momento de conectar alguna señal y comprobar su funcionamiento:



Conectando la señal de 3 voltios de arduino vemos como ha capturado la señal en alto para el canal 0



```
HardwareSerial mySerial(WishboneSlot(5));

int led = 13;

void setup() {
  // put your setup code here, to run once:

  pinMode(led, OUTPUT);

  mySerial.begin(9600);
}

void loop() {

  mySerial.write(1);

  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}
```

Plataforma Raspberry Pi

Objetivos mínimos del trabajo

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

Instalar un Servidor WEB que pueda ejecutar código PYTHON

Raspberry Pi es un computador de placa simple (SBC) de bajo coste desarrollado en [Reino Unido](#) por la [Fundación Raspberry Pi](#). Aunque no se indica expresamente si es hardware libre o con derechos de marca, lo que se entiende es que es un producto con propiedad registrada, pero de uso libre.

En cambio, el software sí es open source, siendo su sistema operativo oficial una versión adaptada de Debian, denominada Raspbian, aunque permite otros sistemas operativos, incluido una versión de Windows 10. Nosotros usaremos una distribución de Ubuntu para Raspberry pi.

El Pi 3, ensambla en su circuito un chipset *Broadcom BCM2387* con cuatro núcleos *ARM Cortex-A53* a 1.2 GHz, dispone de 1GB de RAM. La GPU encargada de los gráficos es la Broadcom VideoCore IV, una solución Dual Core compatible con Open GL ES 2.0 y OpenVG que permite llegar a resoluciones Full HD con soltura. Además, 4 USB, HDMI, Ethernet, Wifi, Bluetooth...



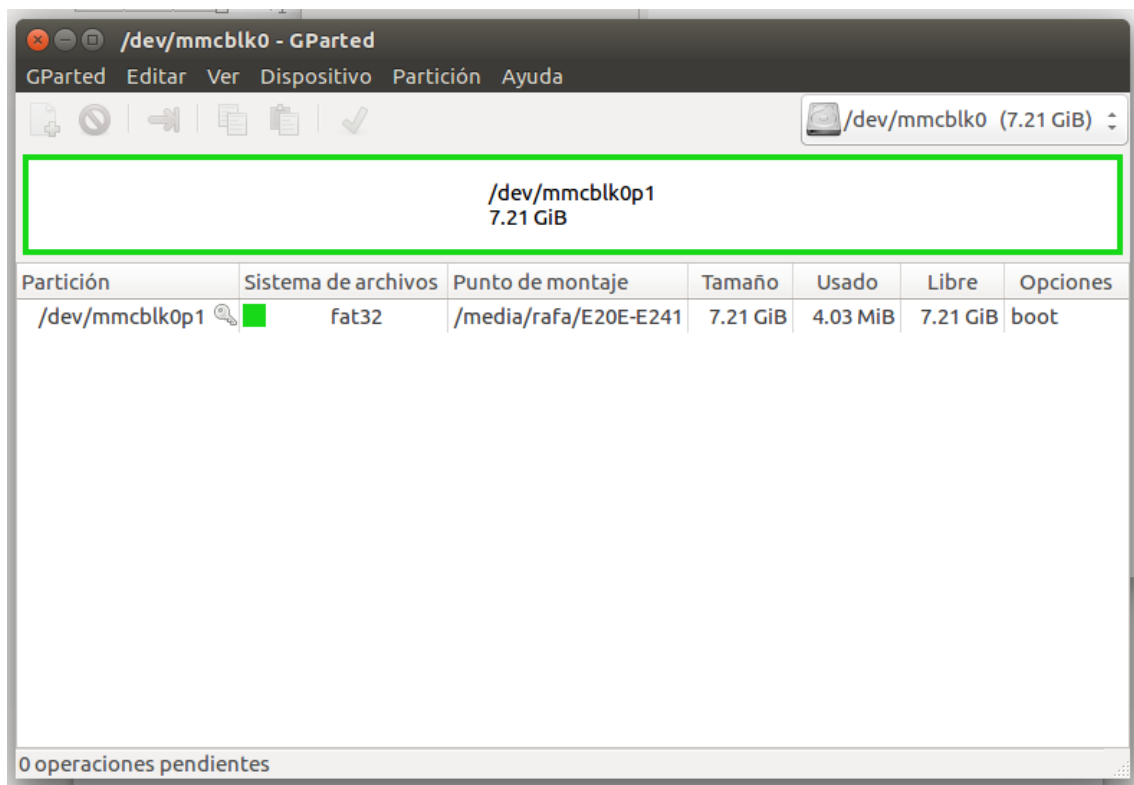
1ª Sesión

Para poder comenzar a realizar ejercicios, vamos a dedicar una sesión a crear la tarjeta SD con lo necesario para nuestra Raspberry Pi.

En Linux, pulsamos la tecla Windows para abrir el lanzador, comenzamos a escribir Gparted, si está en el sistema hacemos clic y ejecutamos si no debemos instalarlo desde el repositorio con:

```
sudo apt-get install Gparted
```

Al abrir, si tenemos introducida la tarjeta en nuestro pc, lo veremos de esta forma, si no será necesario seleccionar la unidad arriba a la derecha.



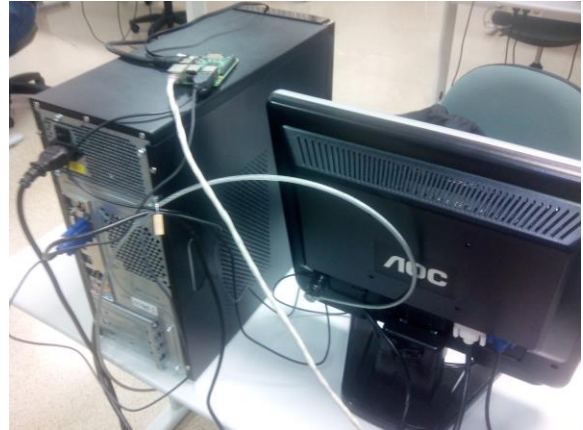
Tiene que quedar una partición como esa: FAT32, formateamos.

Seguimos estas instrucciones para instalar la imagen del sistema operativo que vamos a instalar:

<http://www.raspberrypi.org/documentation/installation/installing-images/linux.md>

Una vez terminado sacamos la tarjeta SD del ordenador y la introducimos en la Raspberry pi con cuidado.

Para este primer arranque vamos a conectar teclado, ratón (ambos a USBs), pantalla (mediante HDMI a DVI, conector del q dispone nuestro monitor en el laboratorio), la red y por último el USB que dará alimentación a la placa. Encendemos el monitor y veremos cómo arranca nuestro sistema. Al ser el primer arranque debemos indicarle un nombre de usuario y una contraseña. Al terminar de arrancar comprobamos que tenemos un sistema operativo al estilo de Ubuntu que parece completamente un PC.



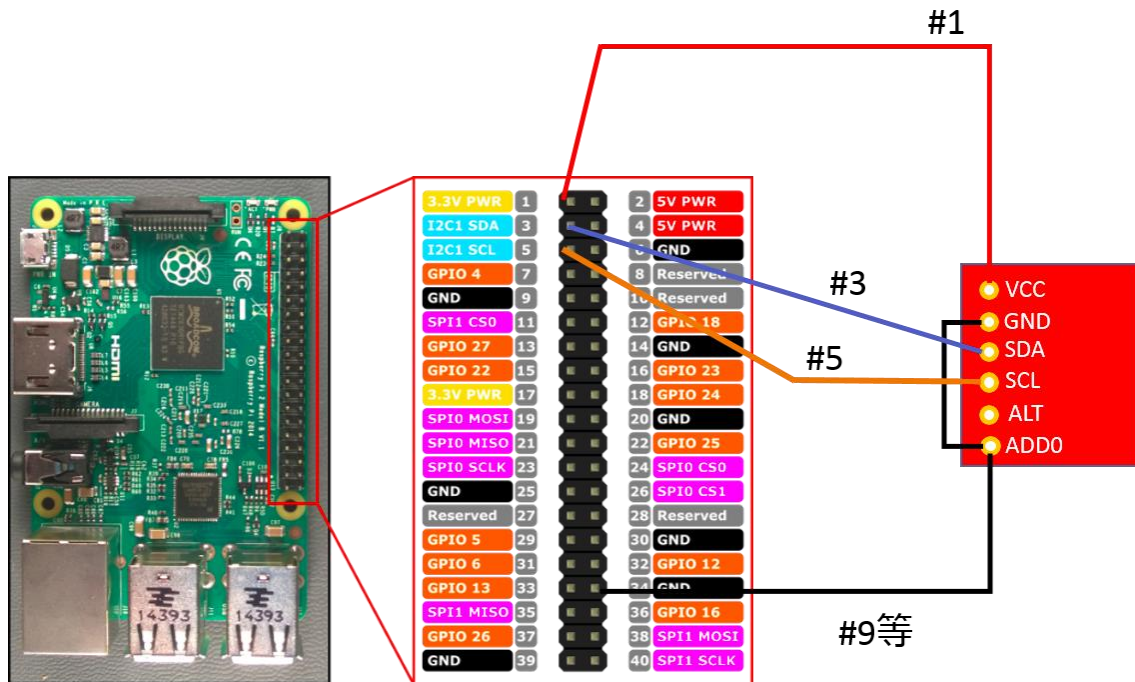
Para configurar la red la hacemos como con cualquier otro sistema Linux, arriba a la derecha clicamos en el icono de redes, editar las conexiones y creamos una nueva con los parámetros que nos indica en el laboratorio, en casa podemos utilizar DHCP para una configuración automática si no hay inconvenientes.

A partir de este momento, no necesitamos la pantalla, teclado ni ratón porque vamos a trabajar por medio de SSH desde otro ordenador. También podríamos acceder a su consola mediante cable serie-USB.

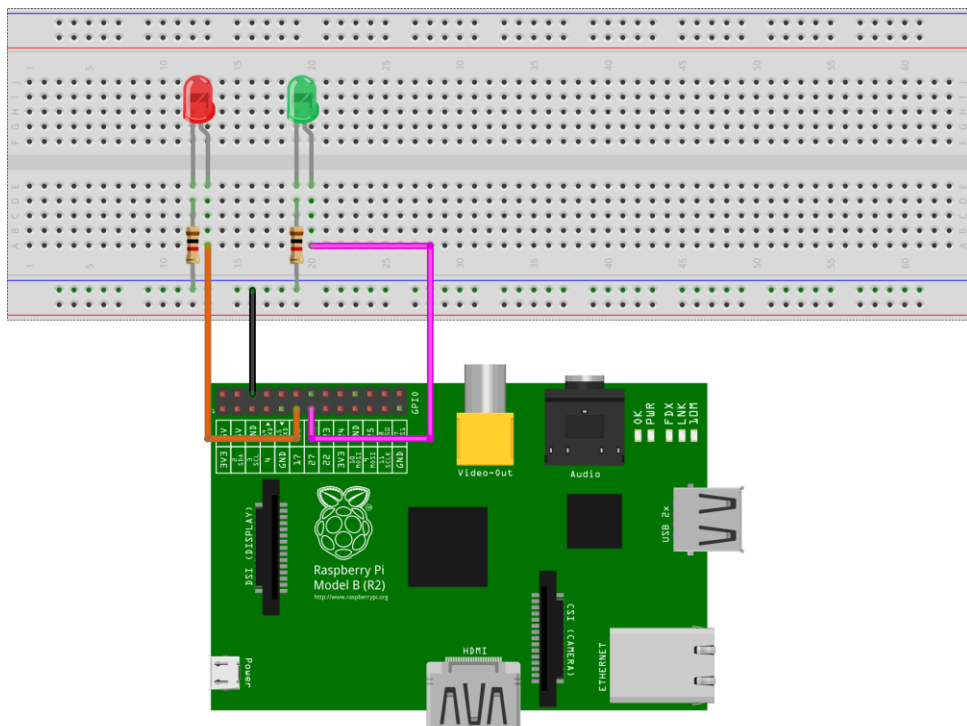
Manejo de GPIOs desde Python

Este ejercicio lo haremos usando el cable USB serie, la idea es encender y apagar un LED desde un código Python dentro de nuestra SD. El sistema operativo instalado ya incluye las librerías para el manejo de los GPIOs y el entorno debería incluir Python, si no, instalamos con:

```
sudo apt-get install python-dev
```



Preparamos el siguiente montaje con 2 leds y dos resistencias en pull-up o pull-down, entradas 17 y 27:



fritzing

weblamp.py

```
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' : 'PIN 1', 'state' : GPIO.LOW},
    27 : {'name' : 'PIN 2', '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)

```

main.html

```

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

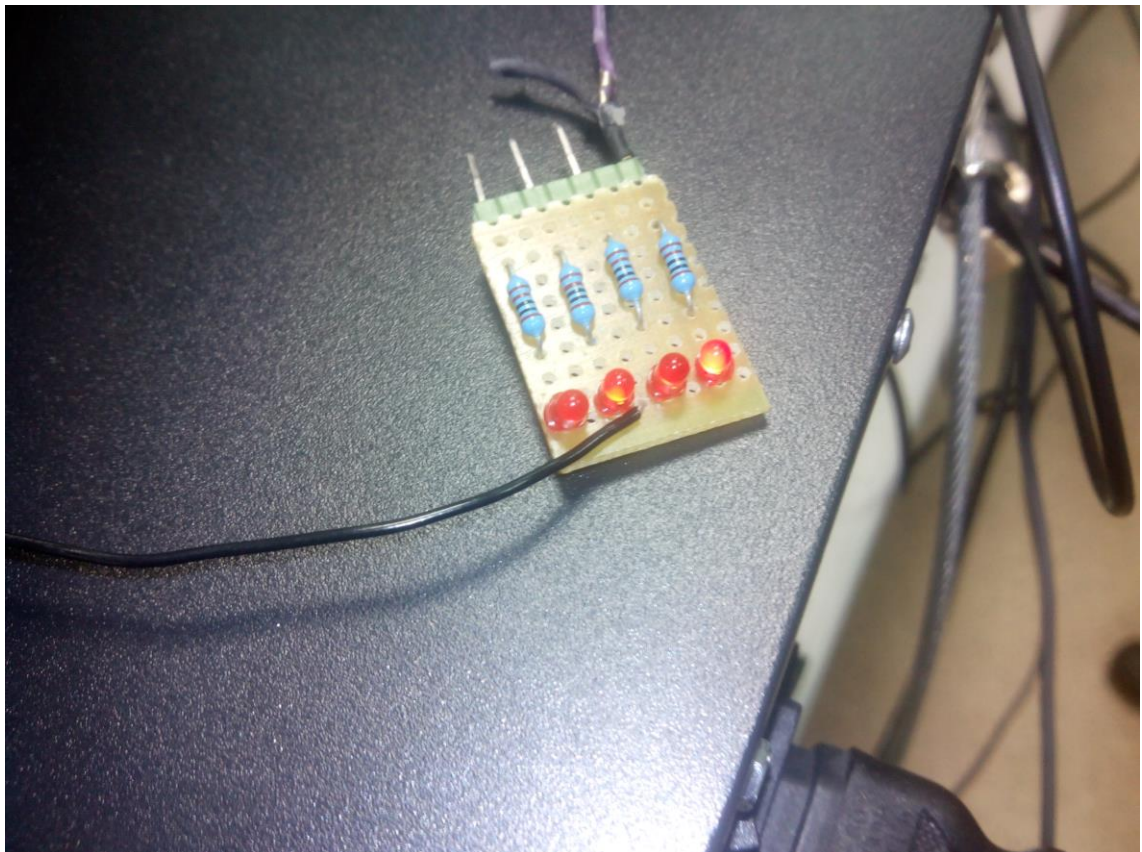
<body>
    <h1>IoT - Led ambience</h1>

    {% for pin in pins %}
    <p>Los {{ pins[pin].name }}
    {% if pins[pin].state == true %}
        está encendido (<a href="/{{pin}}/off">off</a>)
    {% else %}
        está apagado (<a href="/{{pin}}/on">on</a>)
    {% endif %}
    </p>
    {% endfor %}

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

</body></html>

```



Manejo remoto de GPIOs: Servidor Web

Dada la capacidad de estas placas, es posible ejecutar en ellas un servidor web. Si fuésemos capaces de crear un servidor web en Python, podríamos ejecutar códigos como el anterior de forma remota, por ejemplo, una aplicación móvil o una web.

Lo primero será conectarnos a nuestra placa mediante ssh:

```
ssh ip.en.clase.o.casa
```

introducimos usuario y contraseña (los mismos de la instalación u otro que hayamos creado, recordad permisos)

FLASK es un paquete servidor para Linux:

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

necesitamos pip para la instalación:

```
sudo apt-get install python-pip
```

ahora podemos instalar flask

```
sudo pip install flask
```

ahora ya podemos crear nuestro fichero HTML y .py básicos para visualizar una web (servidor y cliente). El contenido es dinámico, es decir, la información de los actuadores son datos que se pasarán al HTML estático para mostrarse.

`weblamp.py`

```
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' : 'LED1', 'state' : GPIO.LOW},
    27 : {'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)

```

main.html

```

<!DOCTYPE html>
<head>
    <title>WEB LAMP</title>
</head>
<body>
    <h1>Lista de Leds y status</h1>

    {% for pin in pins %}
    <p>The {{ pins[pin].name }}
    {% if pins[pin].state == true %}

```

```
    está ON (<a href="/{{pin}}/off">Apagar</a>)
  {% else %}
    está OFF (<a href="/{{pin}}/on">Encender</a>)
  {% endif %}
</p>
{% endfor %}

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

</body>
</html>
```


Servidor web temperatura: Conectar Arduino

Para este ejercicio haremos algunas modificaciones del ejercicio anterior y conectaremos arduino con Raspberry pi para transmitirla la información de este dato por medio del puerto de serie.

weblamp-2.py

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

print '1'
GPIO.setmode(GPIO.BCM)

PuertoSerie = serial.Serial('/dev/ttyACM0', 9600)
print 'serial'
sArduino = PuertoSerie.readline().rstrip()
print '2'
variable = 23
pins = {
    17 : {'name' : 'LED1', 'state' : GPIO.LOW},
    27 : {'name' : 'LED2', 'state' : GPIO.LOW}
}

print '3'

for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

print '4'

@app.route("/")
def main():

    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

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

    return render_template('main2.html', **templateData)

@app.route("/<action>")
def bombilla(action):

    if action == "on":
        PuertoSerie.write("H")
```

```

        if action == "off":
            PuertoSerie.write("L")

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

        return render_template('main2.html', **templateData)

@app.route("/actualiza")
def leerTemperatura():
    sArduino = PuertoSerie.readline().rstrip()
    message2 = "Temperatura:" + sArduino
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    templateData = {
        'message2' : message2,
        'pins' : pins,
        'sArduino' : sArduino
    }
    return render_template('main2.html', **templateData)

@app.route("/<changePin>/<action>")
def action(changePin, action):

    changePin = int(changePin)

    deviceName = pins[changePin]['name']

    if action == "on":

        GPIO.output(changePin, GPIO.HIGH)

        message = "Turned " + deviceName + " on."

    if action == "off":

        GPIO.output(changePin, GPIO.LOW)
        message = "Turned " + deviceName + " off."

    if action == "toggle":

        GPIO.output(changePin, not GPIO.input(changePin))
        message = "Toggled " + deviceName + "."

    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    templateData = {
        'message' : message,
        'pins' : pins,
        'sArduino' : sArduino

```

```
}

return render_template('main2.html', **templateData)

if __name__ == "__main__":
    app.run(host='192.168.0.172', port=80, debug=True)
```

main2.html

```
<!DOCTYPE html>
<head>
  <title></title>
</head>

<body>
  <h1>IoT - Led ambiente</h1>

  {% for pin in pins %}
  <p>Los {{ pins[pin].name }}
  {% if pins[pin].state == true %}
    está encendido (<a href="/{{pin}}/off">off</a>)
  {% else %}
    está apagado (<a href="/{{pin}}/on">on</a>)
  {% endif %}
</p>
{% endfor %}

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

  <h1>Lectura de temperatura: </h1>

  {% if message2 %}
  <h4>{{ message2 }}</h4>
  {% endif %}

  <p>Actualizar temperatura: (<a href="/actualiza">Actualizar</a>)</p>

</body>
</html>
```