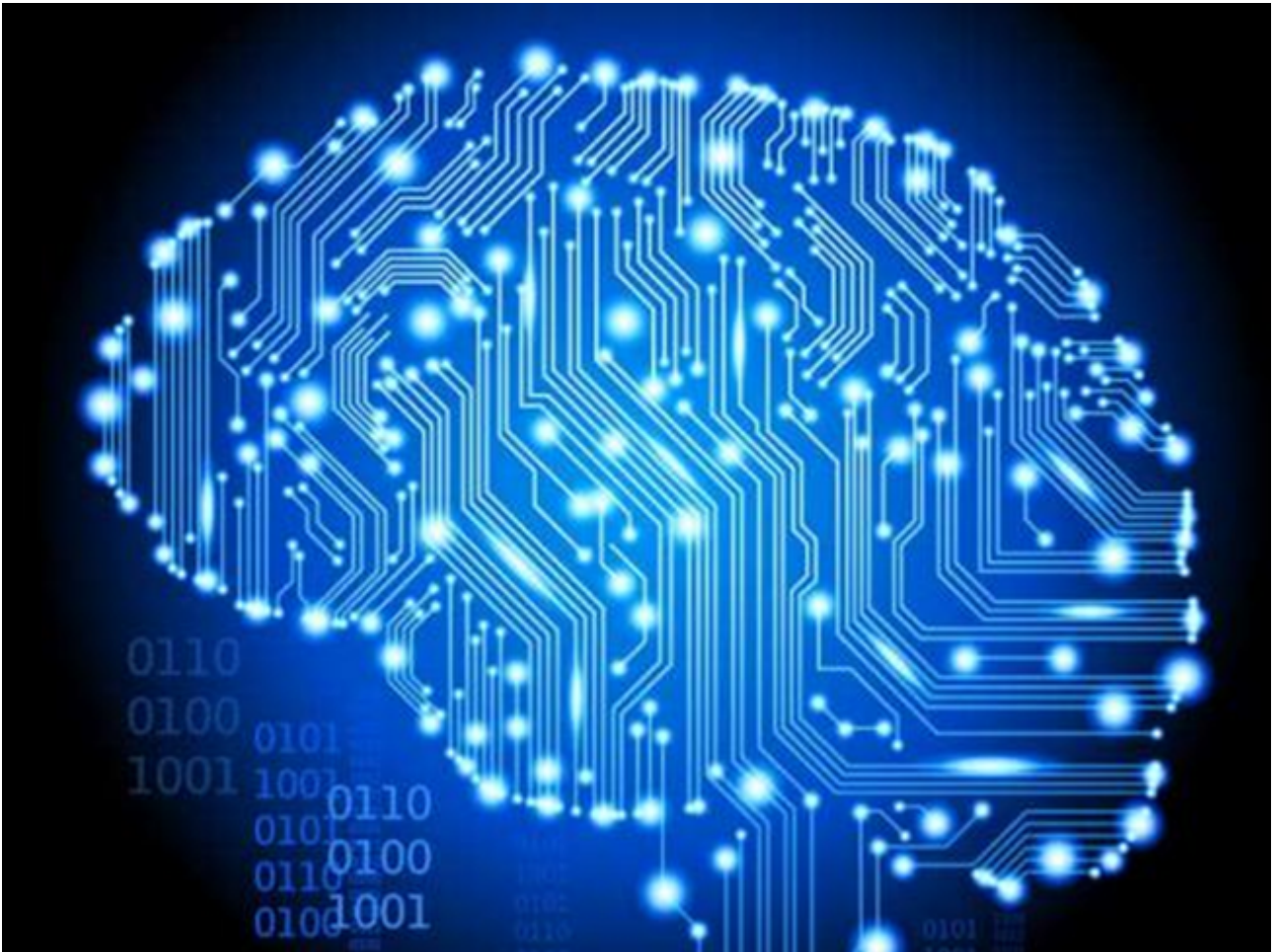


MEMORIA PRÁCTICAS LABORATORIO DE DESARROLLO DE HARDWARE 2016/2017



DANIEL CASTRO CHINCHO

PLATAFORMA ARDUINO:

INTRODUCCIÓN

Arduino es una plataforma de desarrollo de hardware abierta basada en software y hardware flexibles y fáciles de usar. Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros tipos de actuadores.

Está basado en microcontroladores de 8bits AVR aunque con alguna versión basada en ARM de 32 bits.

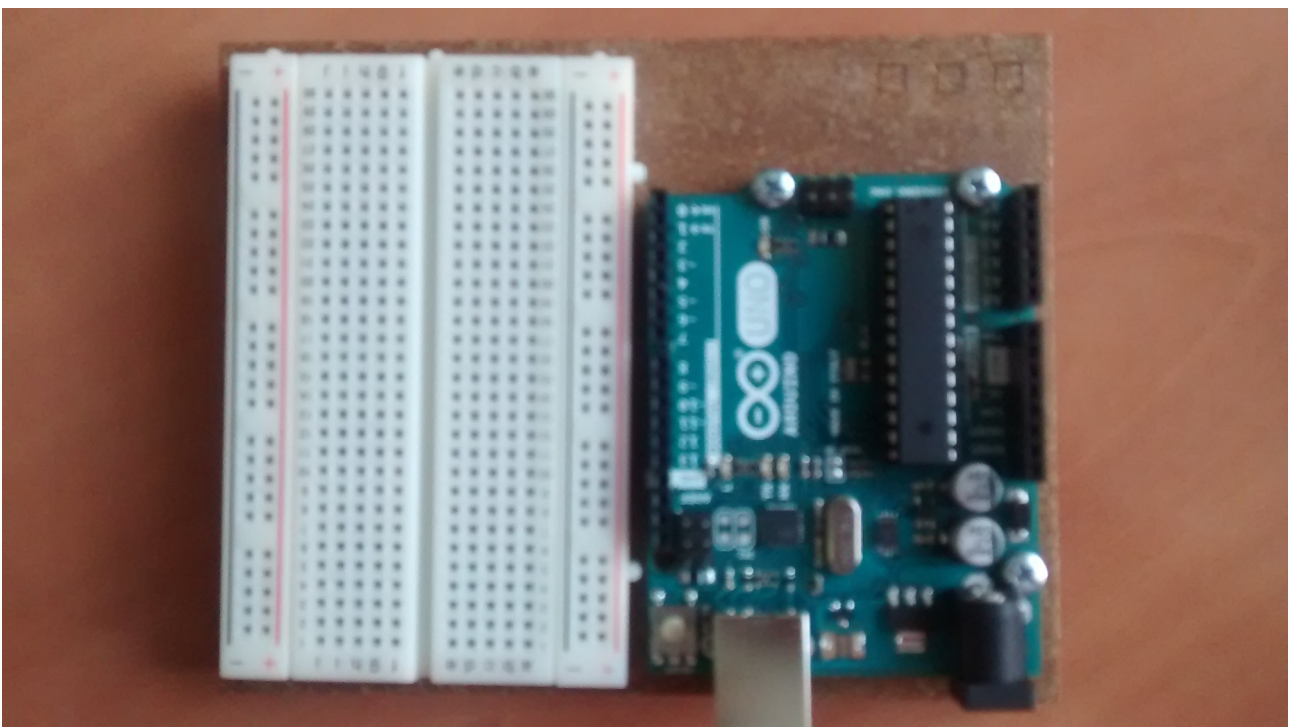
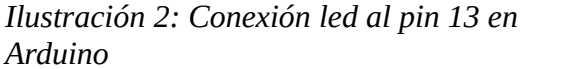


Ilustración 1: Arduino Uno +Protoboard

Incluye un entorno de desarrollo propio muy fácil de manejar, basado en lenguaje C.

Los componentes habituales junto con la placa de desarrollo Arduino, son:

- Shield o placa de expansión
- Resistencias, condensadores, leds, diodos, pulsadores, interruptores, display 7-segmentos, cables de interconexión, tiras de pines...



RaspDuino | Arduino 1.0.1

```
14 Arduino Uno on /dev/ttyACM0
```

Declaramos la variable ‘led’ como entero, a la que le asignaremos el valor 13, que es el pin de conexión en la Arduino.

En el ‘loop’ leeremos constantemente los valores que nos estén llegando desde Python.

El código Python es el siguiente:

```
import serial

arduino = serial.Serial('/dev/ttyACM0', 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
```

En este código, se importan las funciones necesarias para utilizar el puerto serie. Declaramos una variable ‘arduino’ con los datos necesarios para establecer la comunicación (puerto y baudios). Tras esto, en un while-true, pedimos al usuario que introduzca un comando (H y L). Si es H, se encenderá el led, y si es L, se apagará.

Solo nos quedaría ejecutar el archivo y probarlo con el siguiente comando: Damos permisos de escritura al archivo con:

```
sudo chmod +x RaspDuino.py
```

Y ejecutamos con:

```
python RaspDuino.py
```


CONTROL ON/OFF LED MEDIANTE PULSADOR EMPLEANDO INTERRUPCIONES CON UN PULSADOR EN CONFIGURACIÓN DE PULL UP O PULL DOWN

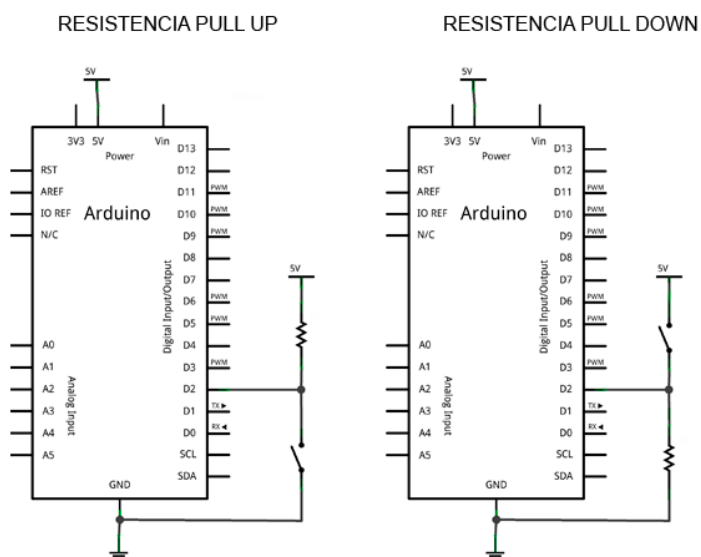


Ilustración 3: Esquema conexionado pulsador Pull Up y Pull Down

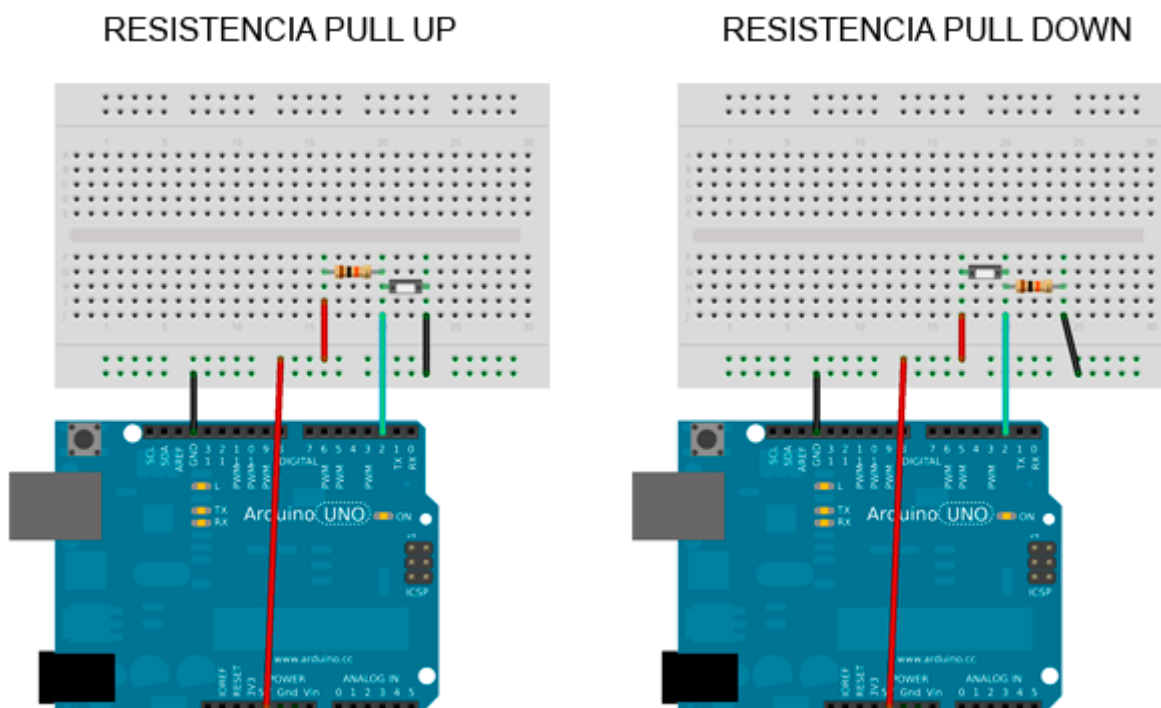


Ilustración 4: Cableado en Protoboard+Arduino Pulsador.

El led, por otro lado, estaría conectado al Pin 13 y a GND.

Las resistencias de Pull-Down y Pull-Up se conectan entre el PIN digital y una de las tensiones de referencia (0V o 5V) y “fuerzan” (de ahí su nombre) el valor de la tensión a LOW o HIGH, respectivamente.

- La resistencia de Pull-Up fuerza HIGH cuando el pulsador está abierto. Cuando está cerrado el PIN se pone a LOW, la intensidad que circula se ve limitada por esta resistencia
- La resistencia de Pull-Down fuerza LOW cuando el pulsador está abierto. Cuando está cerrado el PIN se pone a HIGH, y la intensidad que circula se ve limitada por esta resistencia

El código Arduino empleado,haciendo uso de la función `attachInterrupt()` en el control de la iluminación del led es:

```
const byte ledPin = 13;
const byte interruptPin=2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin,OUTPUT);
  pinMode(interruptPin,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin),blink,CHANGE);
}

void loop(){
  digitalWrite(ledPin, state);
}

void blink(){
  state=!state;
}
```

El comando **`attachInterrupt()`** asigna la función que se ejecutará siempre que una interrupción externa sea generada en el pin especificado. Este comando también asigna el modo en el cual la interrupción externa es generada dependiendo del estado del pin especificado, LOW, CHANGE cuando el estado del pin cambia, RISING cuando el pin va de LOW a HIGH o FALLING cuando el pin va de HIGH a LOW.

El error que se observa es debido a rebotes.A veces el pulsador apaga y otras enciende.

CONTROL DE LA POSICIÓN DE UN SERVO-MOTOR CON UN POTENCIÓMETRO

En ésta práctica, controlaremos el ángulo de rotación de un servomotor desde el PC a través del puerto serie (0 a 180 grados). Lo haremos a través de un código escrito en Python, haciendo uso de `parseInt()` de la librería `serial`. El esquema de las conexiones a realizar es el siguiente:

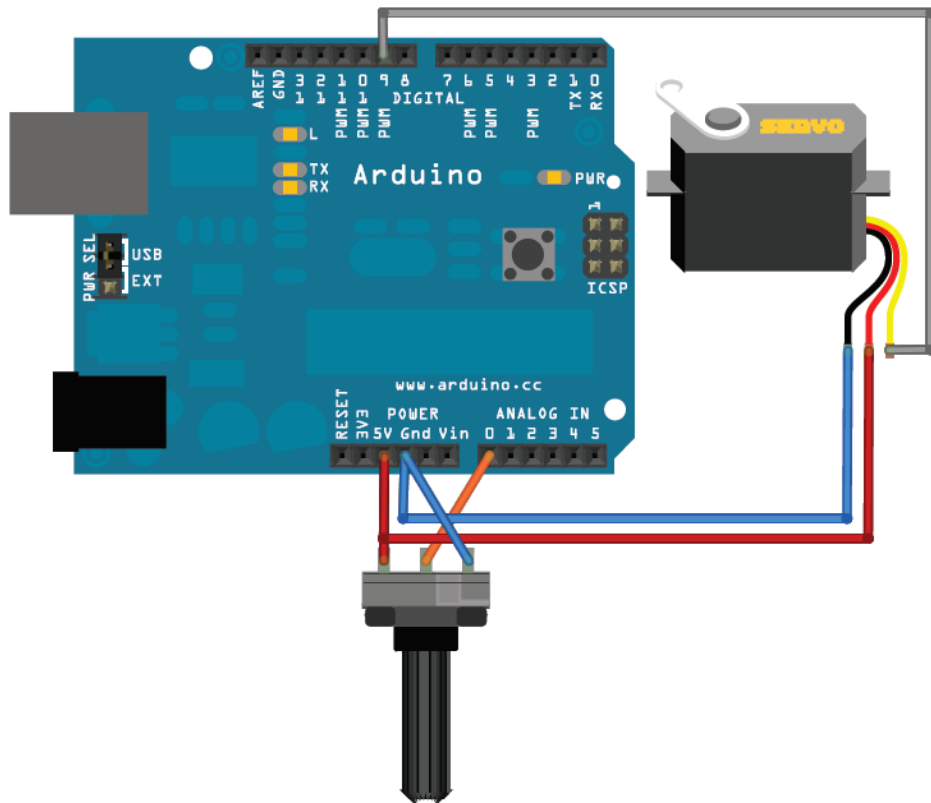


Ilustración 5: Potenciómetro y servomotor en Arduino

El código Arduino empleado:

```
#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(9600);

  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);

  }

}

```

Voy a detallar a continuación una breve explicación acerca de señales analógicas junto con el potenciómetro:

-Por definición, las señales analógicas generan lecturas eléctricas que varían constantemente en amplitud y/o periodo en función del tiempo. Esto hace que puedan ser erráticas y altamente variables, o bien mostrar un patrón estable de cambio.

-Al ocupar un potenciómetro para variar la intensidad de corriente, obtenemos una lectura analógica de cambio que depende mucho de la fabricación de dicho componente, ya que existen potenciómetros lineales, logarítmicos, etc.

-En nuestro caso, usaremos un potenciómetro rotativo lineal.

El código Python empleado para controlar el servomotor via puerto serial desde el PC es:

```

import serial

ser = serial.Serial('/dev/ttyACM0', 115200)

print("Starting!")

while True:

    angle = raw_input('Introduce un comando: ')

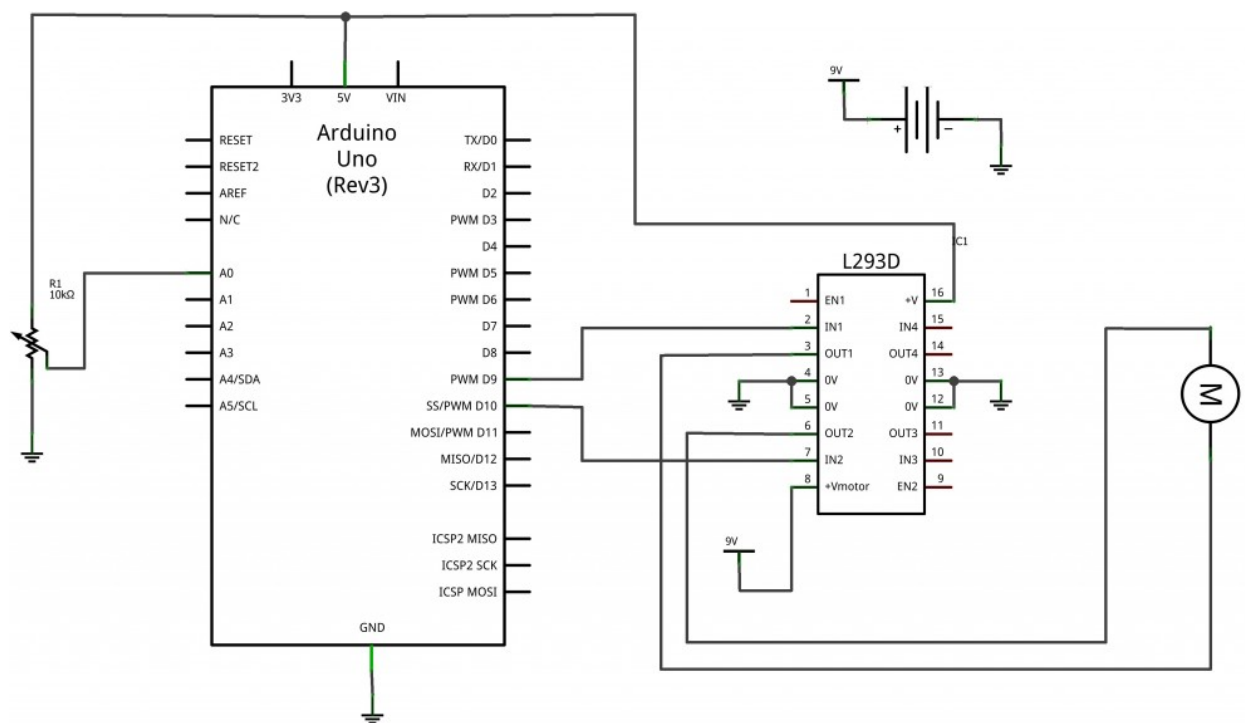
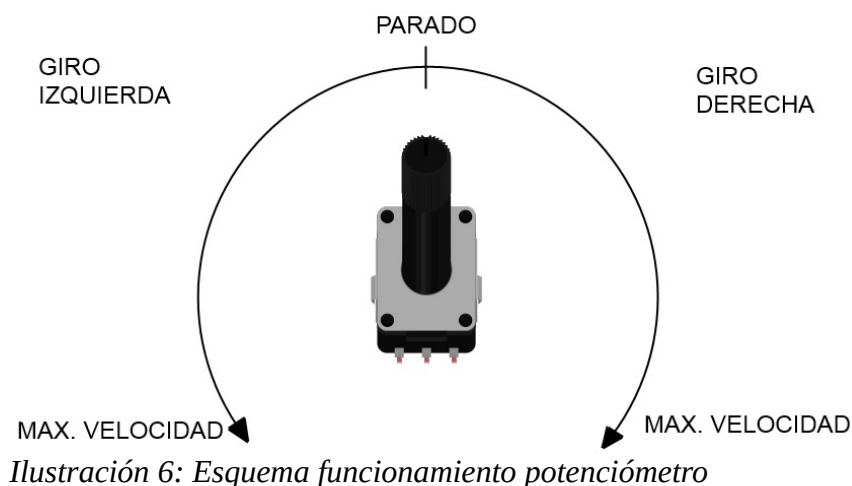
    ser.write(angle)

ser.close() #Finalizamos la comunicacion

```

La función `raw_input` hace que cualquier cosa que le escribamos, la pasa a tipo String. El dígito que le pasemos será pasado a través del serial y comunicará con el servomotor.

CONTROL DE LA VELOCIDAD Y SENTIDO DE GIRO DE UN MOTOR DC CON UN POTENCIÓMETRO



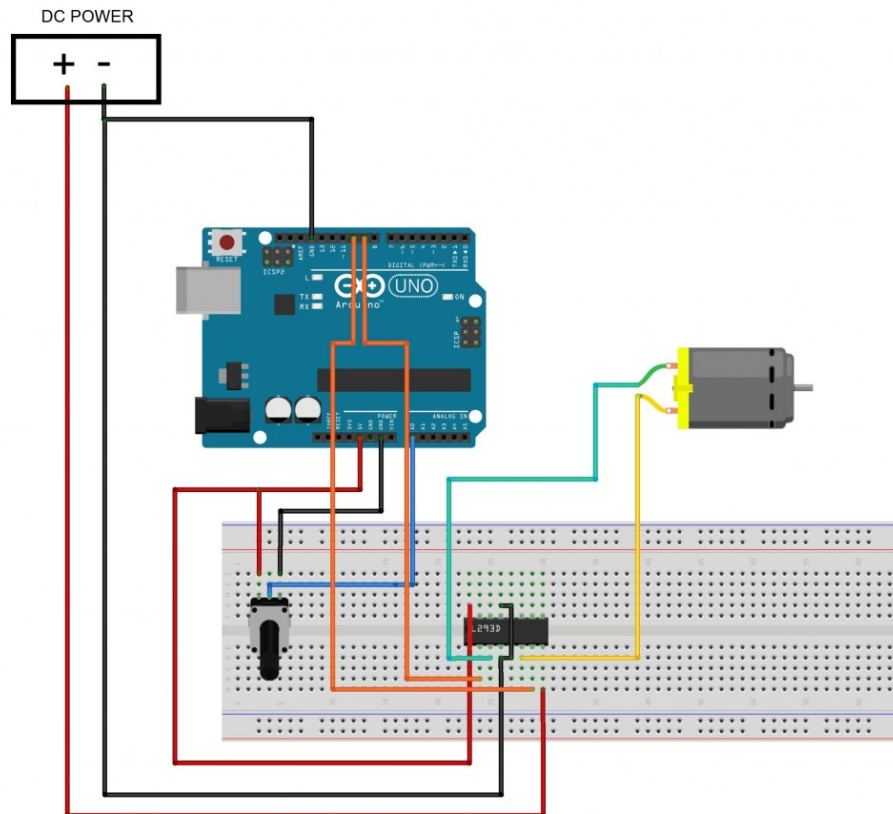


Ilustración 8: Representación montaje completo

- Pins 4,5,12,13 del L293D a masa.
- Juntar las masas del Arduino y de la pila de corriente continua.
- Pin 8 del L293D a 9V de la fuente de alimentación externa. Es el voltaje que proporciona al motor.
- Pin 16 del L293D a 5V. Es la alimentación del L293D. Se alimenta directamente desde la alimentación que proporciona Arduino.

El código Arduino es el siguiente:

```
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 TEXTO EN UNA PANTALLA LCD

El objetivo es imprimir en la pantalla LCD en dos filas con desplazamiento a la izquierda el nombre de la asignatura y el nombre del alumno. También se realizará un código Python para enviarla a través del PC vía serial.

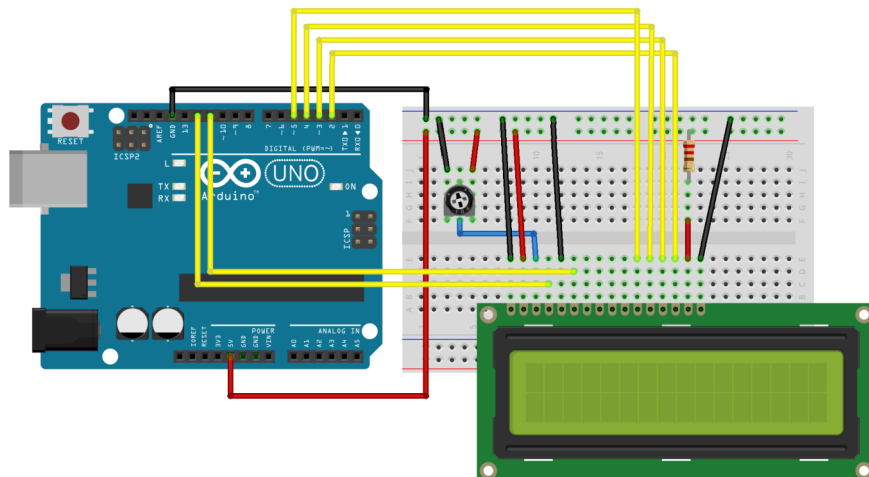


Ilustración 9: Montaje completo del LCD en la placa Arduino, con potenciómetro incluido.

El código Arduino sería el siguiente:

```
// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  lcd.print("LDH");
}

void loop() {
  if (Serial.available()) {
    delay(100); //wait some time for the data to fully be read
    lcd.clear();
    while (Serial.available() > 0) {
      char c = Serial.read();
      lcd.write(c);
    }
  }
}
```

En el anterior loop, el código lee a través del puerto serial lo que escribamos y lo imprime en el LCD.

Si quisiéramos detallar nuestro nombre en la segunda fila, simplemente con el código `lcd.setCursor(0,1);` (0 detalla la columna y el 1 la fila que es la segunda) y a continuación `lcd.print("Daniel Castro");`

El código Python para escribir a través del puerto serie es:

```
import serial
import time

s = serial.Serial(11, 9600) #port is 11 (for COM12), and baud rate is 9600
time.sleep(2) #wait for the Serial to initialize
s.write('Ready...')
while True:
    str = raw_input('Enter text: ')
    str = str.strip()
```

```

if str == 'exit' :
    break
s.write(str)

```

Con esto, especificándole el puerto, podemos pasarle un string con `raw_input()` y eliminamos los espacios en blanco con `strip()`. Si le pasamos la cadena `exit`, finalizaremos la comunicación.

IMPRESIÓN TEMPERATURA EN UNA PANTALLA LCD

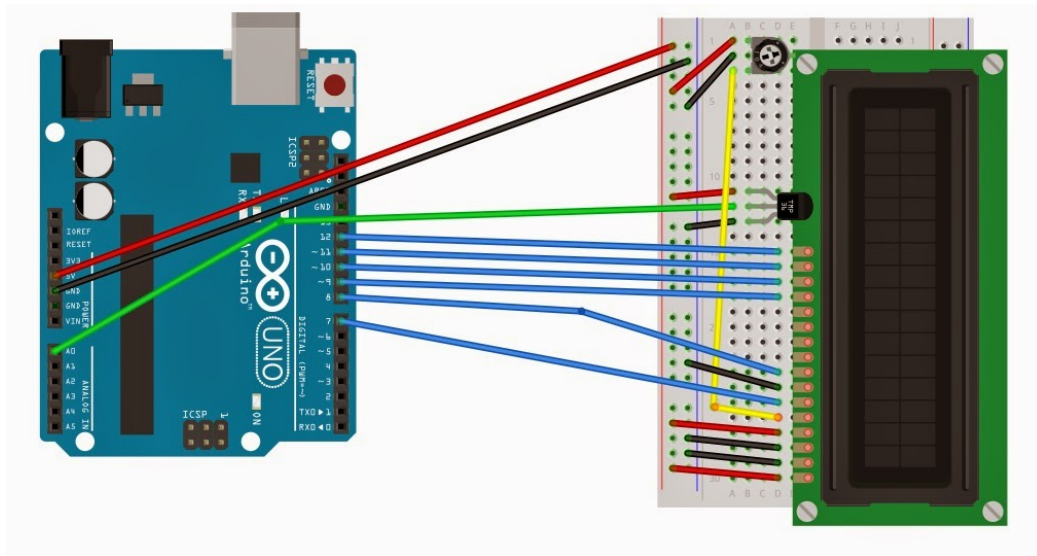


Ilustración 10: Montaje completo Arduino+sensor temperatura+LCD

Al sensor, el pin central se conecta a la entrada analógica A0 del Arduino, uno de los pines extremos se conecta a la alimentación y el otro a GND. Respecto a la conexión del LCD es similar a la práctica anterior. El sensor usado es el TMP36, que es un sensor analógico cuyo rango es de -50°C a 125°C y cuya salida saca 10mV por cada $^{\circ}\text{C}$ de temperatura. Debido a esto, habrá que realizar en el código de Arduino una ecuación que obtenga el voltaje, y a través de él, se obtenga la temperatura, que se imprimirá por pantalla. El código es el siguiente:

```

#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
const int sensorPin = A0;

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

void loop(){

```

```

int sensorVal =analogRead(sensorPin);

Serial.print("Sensor Value: ");
Serial.print(sensorVal);

float voltage = (sensorVal/1024.0) * 5.0;

Serial.print(", Voltios: ");
Serial.print(voltage);

Serial.print("; Temp. Grados: ");
int temperature = (voltage - .5) * 100;
Serial.println(temperature);

lcd.begin(16, 2);
lcd.setCursor(0,0);
lcd.write("Temperatura: ");
lcd.setCursor(12,0);
lcd.print(temperature);
lcd.setCursor(14,0);
lcd.write((char)223);
lcd.setCursor(15,0);
lcd.write("C");
delay(2000);
}

```

El código lo que hace es definir los pines usados para el Display. También definimos una constante sensorPin, para el sensor TMP36, asignándole el pin A0.

En el setup usamos Serial.begin(9600) para comunicarnos con el PC. A través de analogRead, vamos leyendo los valores que va recogiendo el sensor.

El TMP36 emite un voltaje en función de la temperatura (10mV por grado) el Arduino no nos da un voltaje como valor de la entrada analógica, para dicha lectura utiliza un protocolo llamado ADC (Analog to Digital Converter) que nos da un valor comprendido entre 0 y 1023 proporcional al voltaje recibido, por lo que si queremos operar con valores de voltajes, es necesario realizar una pequeña operación matemática, dividir el valor de la lectura entre 1024 (valores posibles) y multiplicar por 5 (las entradas analógicas Arduino solo pueden medir valores entre 0 y 5V), de esta forma:

```
float voltage = (sensorVal/1024.0) * 5.0;
```


Con esos voltios, para obtener una temperatura, de nuevo hay que formular, al voltaje obtenido antes hay que restarle 0,5, para registrar las temperaturas negativas dado que el TMP36 puede mediar hasta -50°C y multiplicar por 100. Ambas variables, voltaje y temperatura, las mostramos en el Monitor Serial de la misma forma que hicimos con el sensorVal:

Por último usamos las funciones del lcd para representarlas e imprimirlas por pantalla del LCD. El tamaño del display usado es de 16 caracteres y 2 líneas.

CONTADOR 00 A 59 EN DISPLAY 7-SEG CADA SEGUNDO

En ésta práctica se le conectará a la placa Arduino una placa de expansión de un display 7-seg . Sobre ella realizaremos un programa que nos haga un contador de 00 a 59 segundos sobre el 7-seg.

Para ello haremos uso de las siguientes funciones:

- **Función principal (loop()):** Bucle infinito que se ejecuta 1 vez por segundo. Calcula el valor correcto de dat1=0, dat0=0. Llama a refresh (dat1,dat0)
- **Función refresh(int dat1, int dat0):** Función que va refrescando los displays 7-seg alternativamente cada Xms y que en total tiene que durar un tiempo total=1seg=1000ms. Para activar display1 y display0 se emplean los bits PB0 y PB1 de la placa de expansión (uno activo y el otro desactivo y así alternativamente cada Xms). El refresco lo puede hacer con llamada a la función write_data (int dato).
- **Función write_data(int dato):** Función que dado un dato entero (dígito de 0 a 9) calcula el valor correcto de representación en los displays 7-seg y lo escribe en la función mediante write7seg(unsigned char arg)
- **Función write7seg(unsigned char):** Se verá a continuación cuando se detalle el código completo.

El código completo es el siguiente:

```

int segPins[] = {
  0, 1, 2, 3, 4, 5, 6, 7};

int tiempototal= 1000;

int j = 40;

int disp1 =8;
int disp2= 9;

int dat1 = 4;
int dat0 = 9;
void setup() {
  // put your setup code here, to run once:
  // loop over the pin array and set them all to output:
  for (int thisseg = 0; thisseg < 8; thisseg++) {
    pinMode(segPins[thisseg], OUTPUT);
  }
  pinMode(disp1, OUTPUT);
  pinMode(disp2, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  dat1 = 0;
  dat0 = 0;
  // while (1) {
  //LLamada a función refresh pasando los datos correctos
  //-- aqui colocar código

  // Cálculo correcto de los datos dat1, dat0 --> desde 0 0 hasta 5 9

  int h=52;

  while(1){
    for (h = 0; h < 60; h++){
      dat1 = h / 10;
      dat0 = h % 10;
      refresh(dat1, dat0);
    }

  }

}

// Función refresh: Duración total de ejecución de refresh: tiempototal.

```

//Se van intercambiando los displays (disp1 con dat0 y disp2 con dat1) a una frecuencia que evite el parpadeo (j--> numero de veces que se activan ambos displays)

```
void refresh( int data1, int data0) {  
    int tiempo_refresco = tiempototal/(2*j);  
    int i;
```

// Bucle de activación de los displays. El bucle se ejecuta j veces. Para escribir en los displays se llama

//a la función write_data (dato)

```
    for (i = 0; i < j; i++){  
  
        delay(tiempo_refresco);  
        digitalWrite(disp1, 1);  
        digitalWrite(disp2, 0);  
        write_data(data1);  
        delay(tiempo_refresco);  
        digitalWrite(disp1, 0);  
        digitalWrite(disp2, 1);  
        write_data(data0);  
    }  
}
```

// Función write_data(dato): Transforma dato en su código siete segmentos para escribirlo en el display

```
void write_data (int arg) {
```

```
    switch (arg) {  
        case 0:  
            //do something when var equals 1  
            write7seg(0x7e);  
            break;  
        case 1:  
            //do something when var equals 2  
            write7seg(0x30);  
            break;  
        case 2:  
            //do something when var equals 1  
            write7seg(0x6d);  
            break;  
        case 3:  
            //do something when var equals 2  
            write7seg(0x79);  
            break;  
        case 4:  
            //do something when var equals 1  
            write7seg(0x33);  
            break;  
        case 5:  
            //do something when var equals 2
```

```

    write7seg(0x5b);
    break;
case 6:
    //do something when var equals 1
    write7seg(0x1f);
    break;
case 7:
    //do something when var equals 2
    write7seg(0x70);
    break;
case 8:
    //do something when var equals 1
    write7seg(0x7f);
    break;
case 9:
    //do something when var equals 1
    write7seg(0x73);
    break;

}

}

// Función write7seg(dato_7seg): escribe el valor dato_7seg en el display

void write7seg (unsigned char arg) {
    unsigned char segmen = 0x01;
    unsigned char display1;
    display1 = arg;

    for (int i = 0; i < 8; i++) {
        if ((display1 & segmen) == 0x00)
            digitalWrite(i, LOW);
        else
            digitalWrite(i, HIGH);

        segmen <<= 1; }

}

```

PLATAFORMA PAPILIO:

INTRODUCCIÓN

Conocer la plataforma PAPILIO

Instalar y configurar el entorno de trabajo de papilio: DESIGNLAB

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

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

Tanto las especificaciones del microprocesador como el diseño de ZPUino son abiertas.

Al ser diseño soft-core, se pueden añadir nuevos periféricos al SoC según las necesidades (diseño hardware en HDLs).

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 permitira modificar el SoC a nivel hardware para añadir nuevos periféricos al mismo.

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
- Comprobar que funciona con un Switch y un LED

INVERSOR PAPILIO

En esta práctica vamos a desarrollar un nuevo proyecto usando un circuito FPGA. Consistirá en un inversor, que irá conectada la entrada a un botón y la salida a un LED . Cuando pulsemos el botón, mostrará la salida inversa.

Para ello crearemos un nuevo proyecto circuito FPGA y a continuación editaremos el circuito con ISE Schematic Editor.

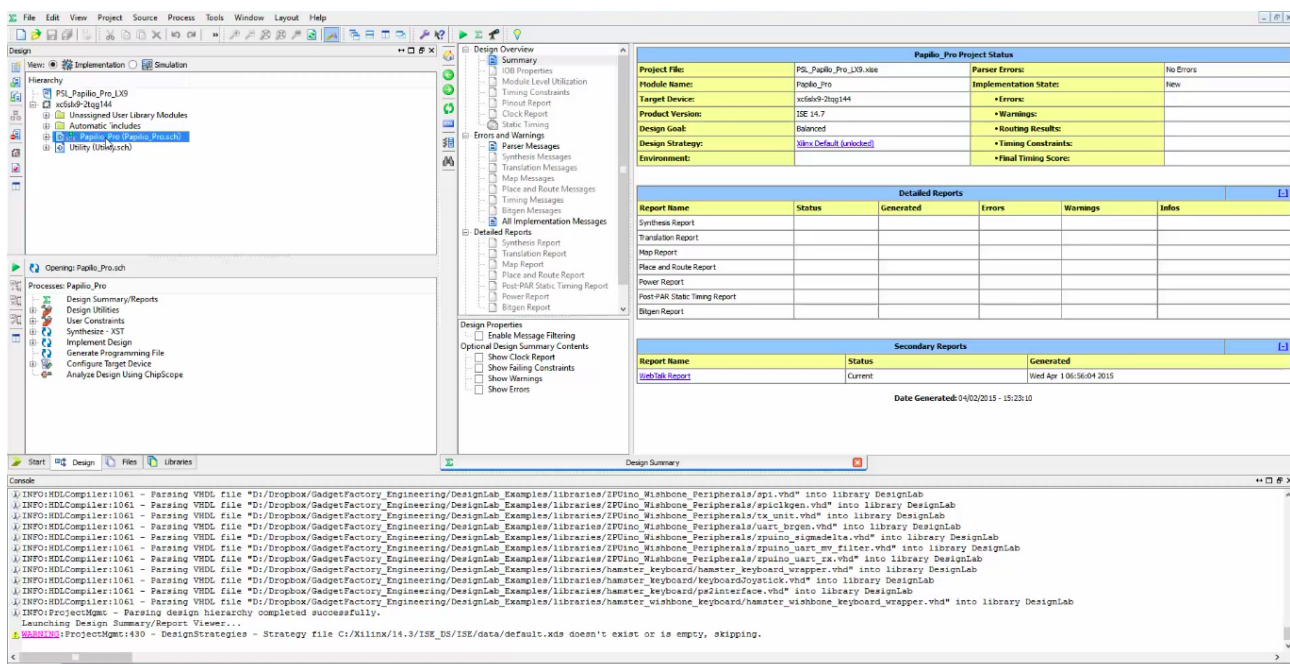
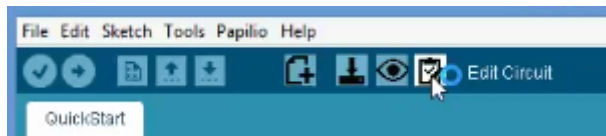


Illustration 11: Apertura ISE Schematic Editor

A continuación, se nos abrirá el editor y crearemos el inversor con sus respectivas salidas.

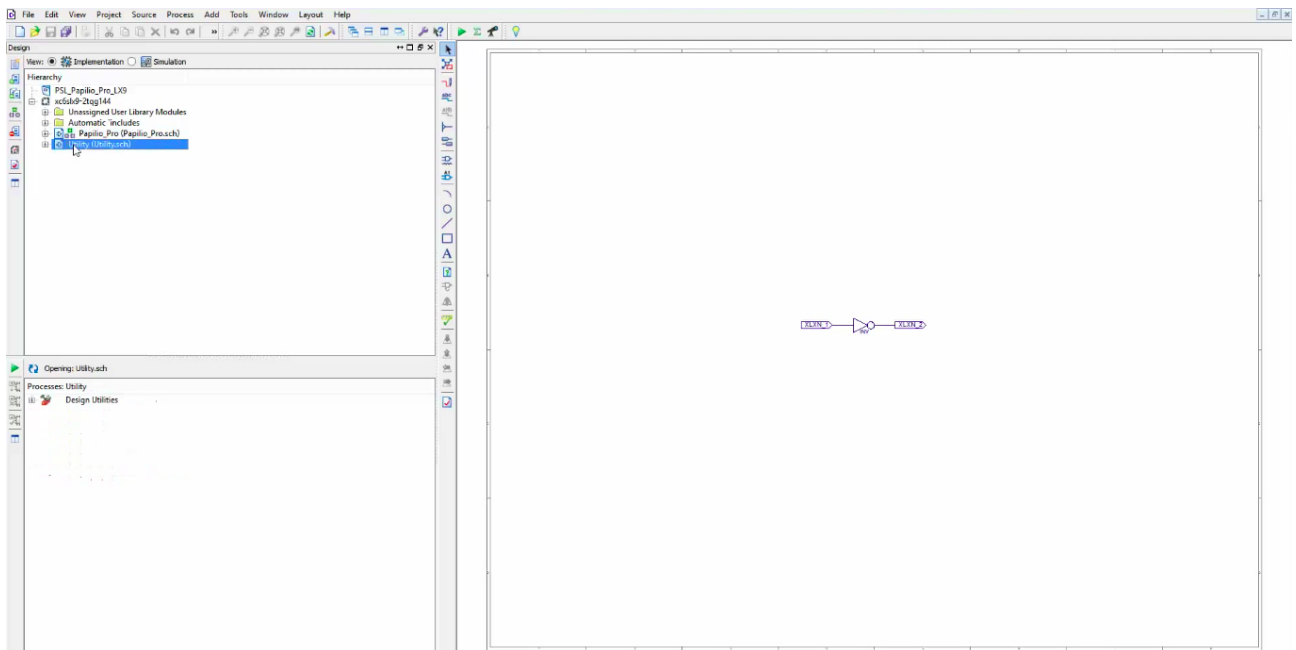


Illustration 12: Creación inversor

Las marcas de I/O nos indican los pines externos a los que nos queremos conectar de nuestra FPGA. Clicando en Utility podemos verlos.

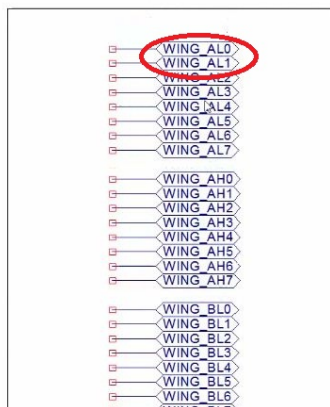


Illustration 13: Pines Papilio

Una vez que tenemos acceso a los pines, los podemos renombrar. Tomamos los nombres WING_AL0 y WING_AL1 y se los asignamos a las I/O del inversor que hemos creado. Una vez acabado, procederemos a generar el Programming File.

Una vez que se ha originado satisfactoriamente, cargaremos el circuito a la placa y se procederá a probar el montaje final.

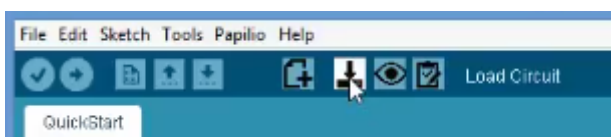


Illustration 15: Carga del circuito a la placa

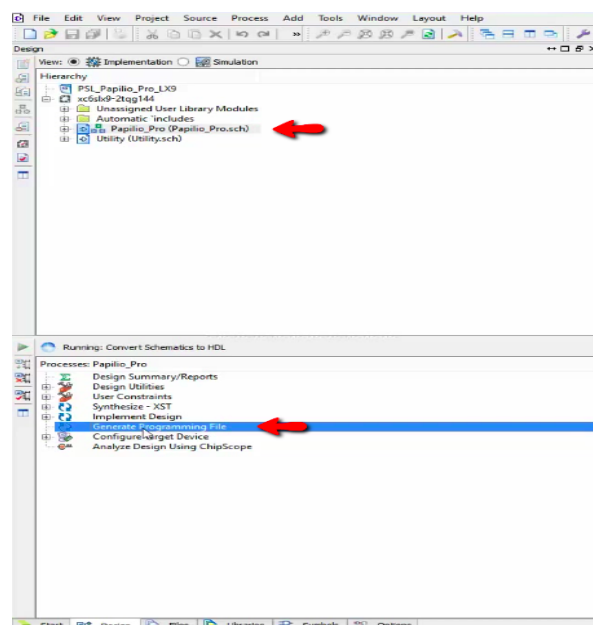


Illustration 14: Generate Programming File

QUICKSTART LED

Para ésta práctica se procederá a abrir el QuickStart Sketch.

```
int ledPins[] = {  
    0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,  
    42, 44, 46 };    // an array of pin numbers to which LEDs are attached  
int ledCount = 24;    // the number of pins (i.e. the length of the array)
```

```
int buttonPins[] = {  
    1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,  
    43, 45, 47 };    // an array of pin numbers to which Buttons are attached  
int buttonCount = 24;    // the number of pins (i.e. the length of the  
array)
```

```
// variables will change:  
int buttonState = 0;    // variable for reading the pushbutton status  
int thisPin;  
int ledState = LOW;
```

```
// first visible ASCII character '!' is number 33:  
int thisByte = 33;  
// you can also write ASCII characters in single quotes.  
// for example. '!' is the same as 33, so you could also use this:  
//int thisByte = '!';
```

```
void setup() {  
    // initialize the LED pins as an output:  
    for (int thisPin = 0; thisPin < ledCount; thisPin++) {  
        pinMode(ledPins[thisPin], OUTPUT);  
    }
```

```
    // initialize the pushbutton pin as an input:  
    for (int thisPin = 0; thisPin < buttonCount; thisPin++) {  
        pinMode(buttonPins[thisPin], INPUT);  
    }
```

```
    //Setup Serial port and send out Title
```

```
Serial.begin(9600);
```

```
// prints title with ending line break
```

```
Serial.println("ASCII Table ~ Character Map");
```

```
}
```

```
void loop(){
```

```
//This sends the ASCII table to the serial port.
```

```
// prints value unaltered, i.e. the raw binary version of the
```

```
// byte. The serial monitor interprets all bytes as
```

```
// ASCII, so 33, the first number, will show up as '!'.
```

```
//Serial.print(thisByte, BYTE);
```

```
Serial.print(", dec: ");
```

```
// prints value as string as an ASCII-encoded decimal (base 10).
```

```
// Decimal is the default format for Serial.print() and Serial.println(),
```

```
// so no modifier is needed:
```

```
Serial.print(thisByte);
```

```
// But you can declare the modifier for decimal if you want to.
```

```
//this also works if you uncomment it:
```

```
// Serial.print(thisByte, DEC);
```

```
Serial.print(", hex: ");
```

```
// prints value as string in hexadecimal (base 16):
```

```
Serial.print(thisByte, HEX);
```

```
Serial.print(", oct: ");
```

```
// prints value as string in octal (base 8);
```

```
Serial.print(thisByte, OCT);
```

```
Serial.print(", bin: ");
```

```
// prints value as string in binary (base 2)
```

```
// also prints ending line break:
```

```
Serial.println(thisByte, BIN);
```

```
// if printed last visible character '~' or 126, stop:
```

```
if(thisByte == 126) {    // you could also use if (thisByte == '~') {
    thisByte = 33;
}
// go on to the next character
thisByte++;
```

```
//This section blinks the LED's and keeps them solid if a button is
pressed.
```

```
delay(200);           // wait for a second
ledState = !ledState;
for (int thisPin = 0; thisPin < buttonCount; thisPin++) {
    // read the state of the pushbutton value:
    buttonState = digitalRead(buttonPins[thisPin]);
```

```
    // check if the pushbutton is pressed.
    // if it is, the buttonState is HIGH:
    if (buttonState == HIGH) {
        // turn LED on:
        digitalWrite(ledPins[thisPin], HIGH);
    }
    else {
        // toggle LED:
        digitalWrite(ledPins[thisPin], ledState);
    }
}
}
```

Se carga el circuito en la placa igual que en la práctica anterior y se prueba con el montaje final.

Un pequeño programa en Python para encender o apagar a través del puerto serie:

```
import serial
```

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

```
print("Starting!")
```

```
while True:
```



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

```
papi.close() #Finalizamos la comunicación
```

ANALIZADOR LÓGICO

Una aplicación que tienen las placas Papilios es poder funcionar como analizadores lógicos.



Illustration 16: Analizador Lógico en DesignLab

Una vez clicado, se generará un bit file y se nos abrirá la ventana del analizador.

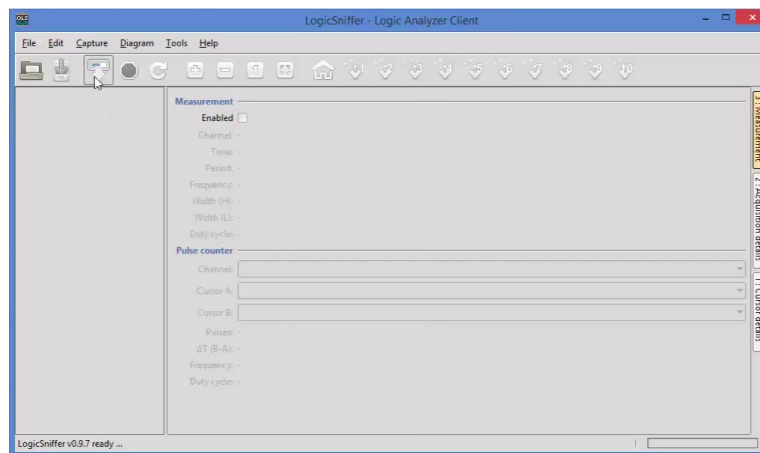


Illustration 17: Ventana Analizador Lógico

Una vez abierto, lo configuraremos de forma apropiada, donde seleccionaremos el serial port, el COM port apropiado, la velocidad del puerto y la placa que usaremos.

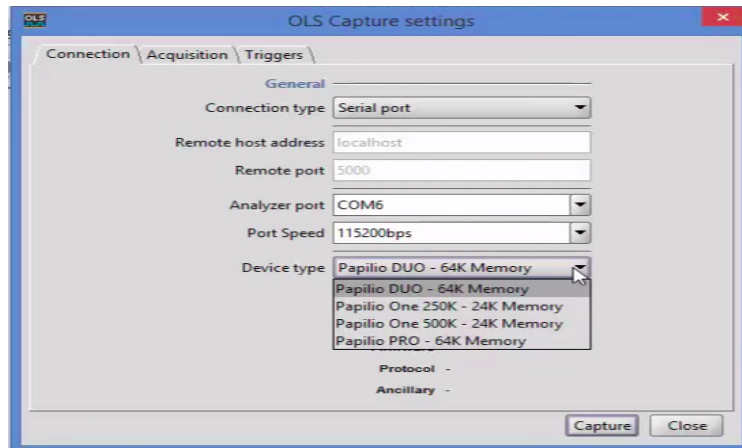


Illustration 18: Ventana configuración en el Analizador Lógico.

En la ventana de Acquisition, usaremos un Sampling Rate de 20000MHz.

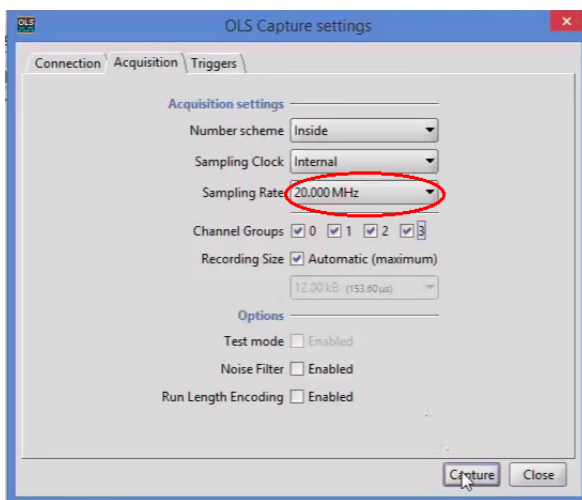


Illustration 19: Ventana Acquisition

Una vez configurado, se procederá a la captura, donde vislumbraremos los diferentes canales que pueden capturar información.



Illustration 20: Ventana de canales en el analizador lógico.

Tras esto, capturaremos la señal que nos produce la placa Arduino a través de un código de ejemplo. El código de ejemplo en este caso sería Fade

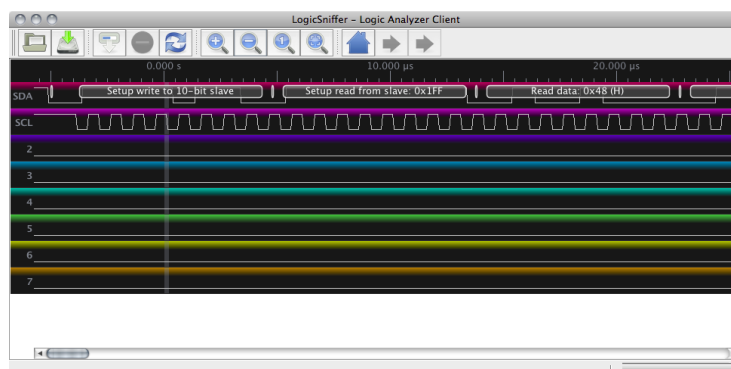
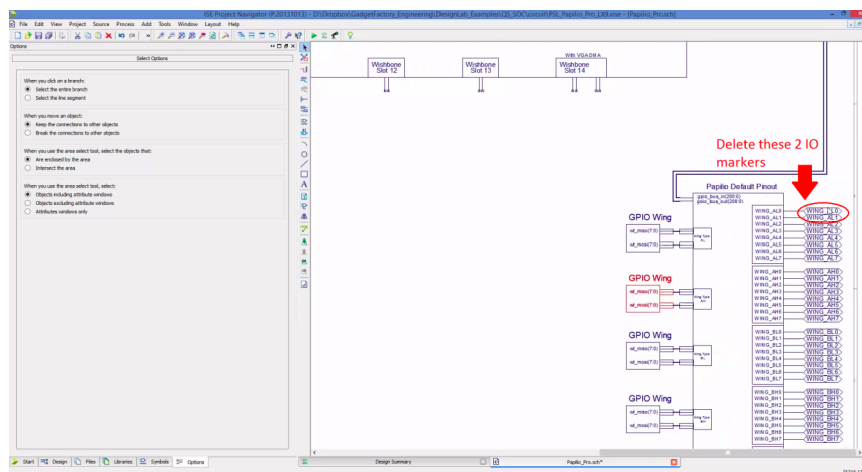


Illustration 21: Ejemplo toma señal analizador logico

REDISEÑANDO EL SOC. AÑADIENDO PERIFÉRICOS

Vamos a desarrollar algún ejemplo básico de como añadir un periférico al SOC de ZPUINO, sintetizarlo, generar el bit file, programarlo en la FPGA y utilizarlo desde un sketch.

El procedimiento será igual que las prácticas anteriores: crearemos un NEW ZPUINO SOC PROJECT, y a continuación editaremos el circuito. Una vez tengamos el esquemático, procederemos a su modificación.



En primer lugar, borraremos del esquema dos etiquetas, que serán las que usaremos en la UART que añadiremos

Illustration 22: Marcas I/O que se van a borrar

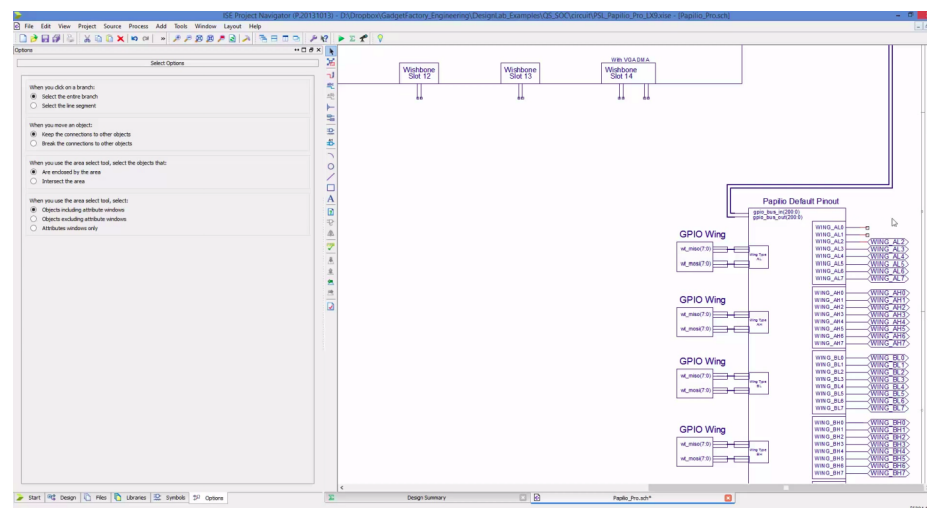
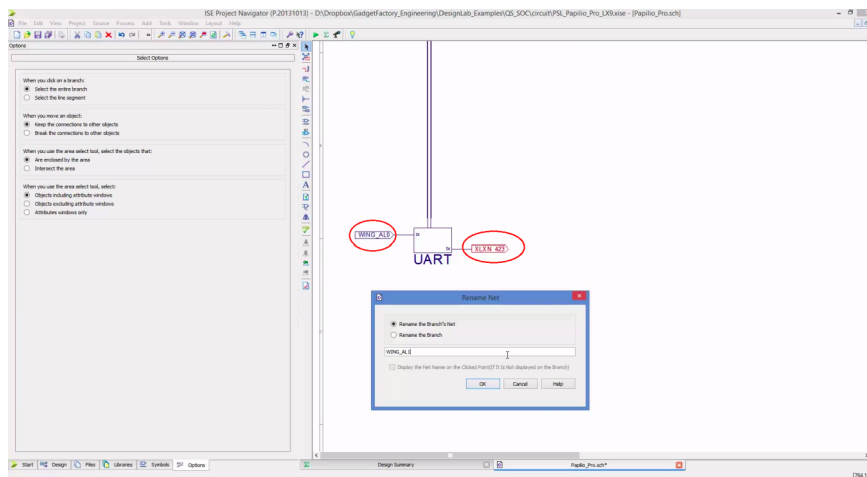


Illustration 23: Borrado de las etiquetas



Renombramos los puertos con WING_AL0 Y WING_AL1. Tras esto, ya tenemos nuestro sistema personalizado con un UART extra.

Illustration 24: Etiquetas añadidas a la UART

Tras todo esto sintetizamos y generamos el Programming File. Debemos de recordar el WishBone usado, que en este caso será el 5.

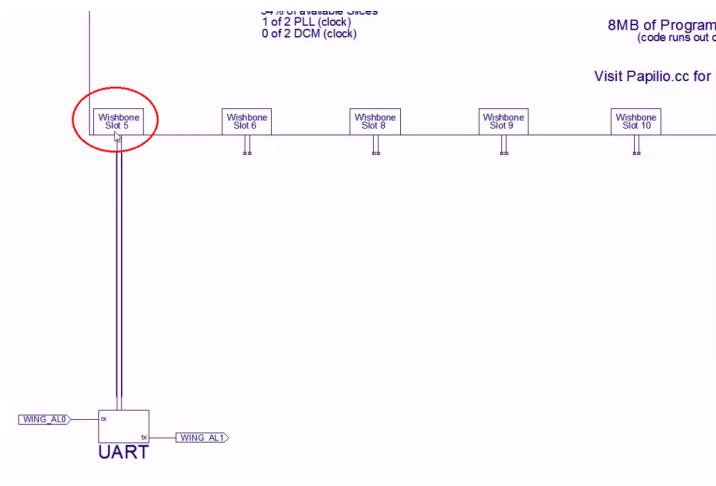


Illustration 25: UART conectada al WishBone 5

Tras todo esto, se procederá a cargar el circuito. Se probará con un led conectado a la placa junto con un conector para el puerto serie.

PLATAFORMA RASPBERRY PI:

INTRODUCCIÓN

Prepararemos la plataforma Raspberry Pi para cargar diferentes versiones de SO, comprobar el funcionamiento de la placa, así como usar ejemplos que usen los pines de expansión GPIOs, y por último, instalar un servidor web que ejecute código PYTHON.

Usaremos como SO ubuntu-mate, donde realizaremos las configuraciones de red pertinentes, sobre todo para poder conectarnos vía SSH posteriormente.

GPIOs PYTHON

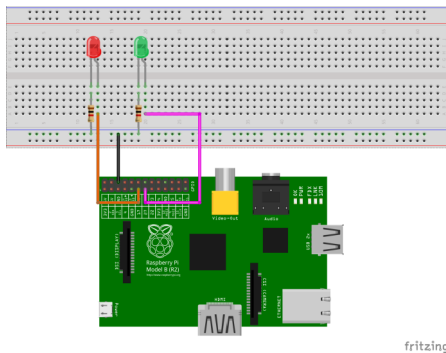


Illustration 26: Montaje GPIOs Raspberry Pi

Para comenzar, se realizará el montaje de la figura y se usarán los GPIO 17 Y 27. Tras esto, se instalarán las librerías GPIO para Python. Una vez instalada, se procederá a hacer un encendido intermitente de los leds.

Para ello crearemos un nuevo archivo Python `sudo nano blink.py`.

Importamos la librería que acabamos de instalar y declaramos los pines:

```
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
```

Creamos una función para ejecutar el bucle que enciende y apaga los LEDs.

```
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) ## Enciende 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
```

Llamamos a la función `blink()` ## Hago la llamada a la funcion blink
Ejecutamos el código `sudo python blink.py`
Tras esto, los leds estarán intermitentes. Todo esto se ha realizado a través de una conexión SSH hacia la raspberry pi.

SERVIDOR WEB GPIOs

Uno de los aspectos mas interesantes de los SBC es que permiten controlar otros dispositivos hardwares (con sensores y/o actuadores)

Además este control puede hacerse a distancia por ejemplo a través de internet.

Una forma de hacerlo es montando un servidor WEB a través del cual podamos bien enviar datos a los actuadores (por ejemplo para encender/apagar luces), bien para leer datos de sensores y dar información (por ejemplo, leer la temperatura de algún sensor).

En ésta práctica usaremos el framework FLASK para convertir a la Raspberry en un servidor web dinámico. Para instalar FLASK se procederá de la siguiente forma:

```
pi@raspberrypi ~ $ sudo apt-get install python-pip
pi@raspberrypi ~ $ sudo pip install flask
```

Para probar la instalación se probará con un programa llamado `hello-flask.py`

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

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

Donde definimos un objeto FLASK llamado app. La función nos imprimirá por pantalla la función definida hello.

Todo ésto puede ser representado en un documento HTML.

Crearemos un nuevo archivo llamado `hello-template.py` con el siguiente código:

```
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)

```

Éste programa nos imprimirá la fecha actual con un formato personalizado, y un texto HELLO, donde será representada en el html que crearemos a continuación a través de un documento llamado `main.html`.

```

<!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>

```

SERVIDOR WEB WEBLAMP

En ésta práctica controlaremos el encendido de 2 leds a través de la web. Crearemos un archivo `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 = {
    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)

```

Y un archivo main.html donde se podrá controlar los leds:

```

<!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 %}

```

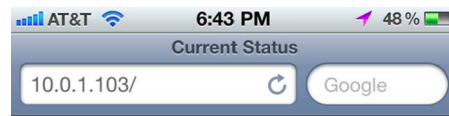


```
{% if message %}
<h2>{{ message }}</h2>
{% endif %}

</body>
</html>
```

El resultado sería el siguiente:

Donde el montaje sería equivalente al de la práctica de GPIOs PYTHON, con los leds conectados a los pines que hemos declarado en weblamp.py



Device Listing and Status

The coffee maker is currently on ([turn off](#))

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



Illustration 27: Web de control de los leds

SERVIDOR WEB TEMPERATURA Y RELÉ

La diferencia respecto a la anterior práctica es que se le ha incluido una función llamada leerTemperatura, la cual mostraremos en la web. También hemos declarado 2 variables al inicio del código: PuertoSerie, que es donde leerá, y sArduino, que tomará las lecturas desde la Arduino las tomas de temperatura.

El código es el siguiente:

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

GPIO.setmode(GPIO.BCM)

PuertoSerie = serial.Serial('/dev/ttyACM0', 9600)
sArduino = PuertoSerie.readline().rstrip()

variable = 23
pins = {
    17 : {'name' : 'LED1', 'state' : GPIO.LOW},
    27 : {'name' : 'LED2', 'state' : GPIO.LOW}
}

for pin in pins:
```

```

GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, GPIO.LOW)
@app.route("/")
def main():

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

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

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

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

```

También crearemos un archivo html llamado main.html, en el que aparte de controlar los leds, también leeremos la temperatura, pudiendo actualizarla cada vez que queramos.

```

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

<body>
<h1>Lista de dispositivos:</h1>

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

<h1>Bombilla: </h1>

<p>Quiero encenderla (<a href="/off">Encender</a>)</p>
<p>Quiero apagarla (<a href="/on">Apagar</a>)</p>

</body>
</html>

```

El código arduino empleado para tomar la temperatura es, donde led=7 es el pin donde conectaremos el relé:

```

const int pinAnalogico =A0;
int led = 7;

void setup () {
  pinMode(led, OUTPUT);
  Serial.begin(9600); //Inicializo el puerto serial a 9600 baudios
}

void loop () {
  if (Serial.available()) { //Si está disponible
    char c = Serial.read(); //Guardamos la lectura en una variable char
    if (c == 'H') { //Si es una 'H', enciendo el LED
      digitalWrite(led, HIGH);
    } else if (c == 'L') { //Si es una 'L', apago el LED
      digitalWrite(led, LOW);
    }
  }

  int lectura = analogRead(pinAnalogico);
  float voltaje = 5.0 /1024 * lectura ;
  float temp = voltaje * 100 -50 ;
  Serial.println(temp) ;
  delay(1000);
}

```