

MEMORIA DE PRÁCTICAS LDH

LABORATORIO DE DESARROLLO HARDWARE

Rubén Luque Romero



ÍNDICE

1. Arduino	2
1.1 Introducción.....	2
1.2 Objetivos.....	3
1.3 Material empleado.....	3
1.4 Entorno de desarrollo.....	4
1.5 Prácticas realizadas.....	5
2. Papilio/ZPUino.....	19
2.1 Introducción	19
2.2 Objetivos.....	20
2.3 Material empleado.....	20
2.4 Entorno de desarrollo.....	20
2.5 Prácticas realizadas.....	21
3. Raspberry PI	31
2.1 Introducción	31
2.2 Objetivos.....	31
2.3 Material empleado.....	32
2.4 Entorno de desarrollo.....	32
2.5 Prácticas realizadas.....	35

1. Arduino

1.1 Introducción

- **Definición**

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Es una placa programable con entradas y salidas digitales y analógicas, cuyo bajo coste la hace ideal para iniciarse en automatización o realizar pequeños proyectos domésticos en electrónica y robótica. Basado inicialmente en microcontroladores de 8 bits AVR (Atmega 8, 128, 328, 1280) aunque con alguna versión basada en ARM de 32 bits (Arduino Due).

- **Ventajas**

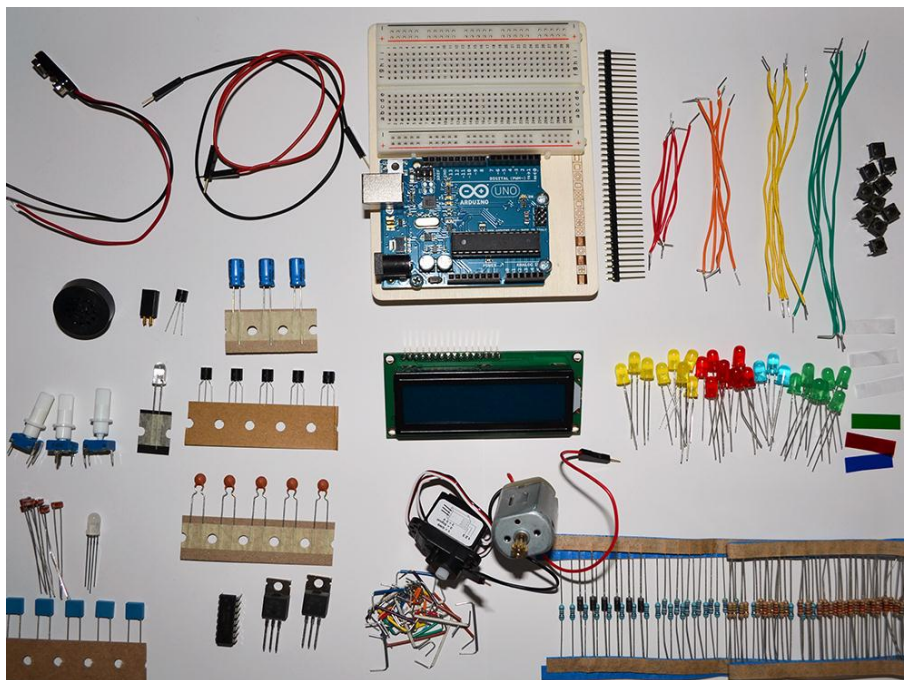
- **Barato:** Las placas Arduino son relativamente baratas comparadas con otras plataformas microcontroladoras.
- **Multiplataforma:** El software de Arduino se ejecuta en sistemas operativos Windows, Macintosh OSX y GNU/Linux. La mayoría de los sistemas microcontroladores están limitados a Windows.
- **Entorno de programación simple y claro:** El entorno de programación de Arduino es fácil de usar para principiantes, pero suficientemente flexible para que usuarios avanzados puedan aprovecharlo también.
- **Código abierto y software extensible:** El software Arduino está publicado como herramientas de código abierto, disponible para extensión por programadores experimentados.
- **Código abierto y hardware extensible:** El Arduino está basado en microcontroladores ATMEGA8 y ATMEGA168 de Atmel. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer su propia versión del módulo, extendiéndolo y mejorándolo.

1.2 Objetivos

El principal objetivo de Arduino es interactuar con el mundo físico. Además, conocer los mecanismos de programación, sus principales características, usos y aplicaciones, sus variantes y conocer una serie de componentes básicos de hardware típicos de aplicaciones de sistemas empuetrados.

1.3 Material empleado

Utilizaremos la placa ARDUINO UNO y diferentes componentes electrónicos que nos servirán para el desarrollo de las prácticas:



Características de ARDUINO UNO:

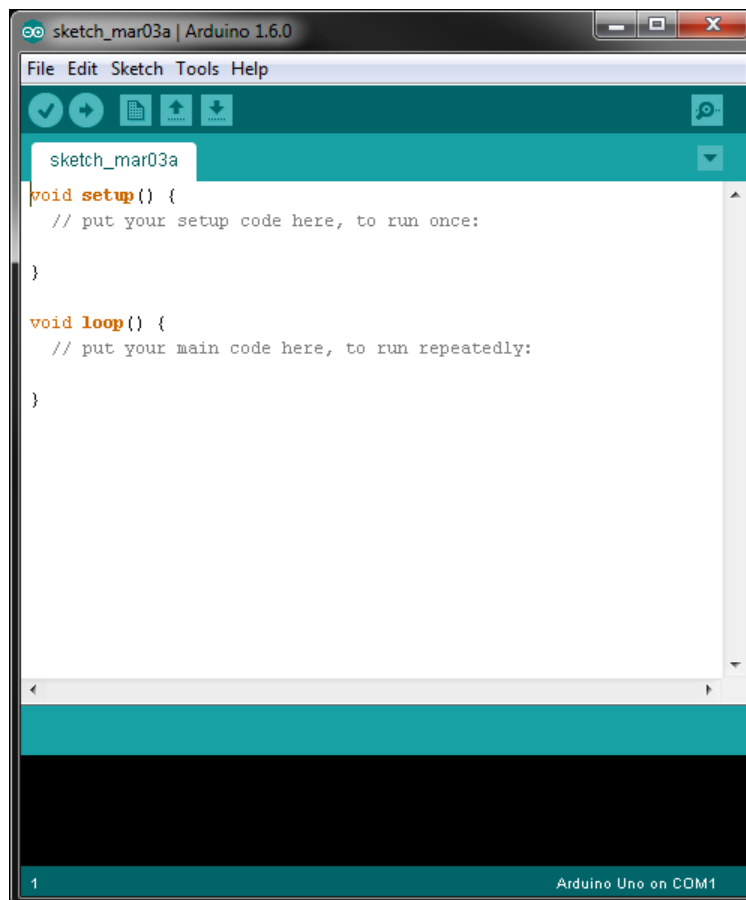
- Procesador ATMEGA 328p a 16 Mhz.
- 32k Flash memory.
- 14 GPIO (6 pueden ser PWM).
- 6 entradas analógicas.

1.4 Entorno de desarrollo

El entorno de desarrollo debemos descargarlo desde la página oficial de Arduino, y arrancarlo desde Linux. Seguiremos los siguientes pasos:

1. Descargaremos el software (IDE) desde la página oficial de Arduino.
2. Conectar la placa al PC mediante USB e instalar los drivers necesarios.
3. Ejecutamos la aplicación desde el terminal de Linux.
4. Seleccionamos el tipo de placa que tenemos, en nuestro caso Arduino Uno.
5. Seleccionamos el puerto serie de comunicación con la placa.

De esta forma, ya tenemos el entorno puesto en marcha, sólo falta crear el código, compilarlo y subirlo a la placa. La interfaz del entorno de desarrollo es la siguiente:



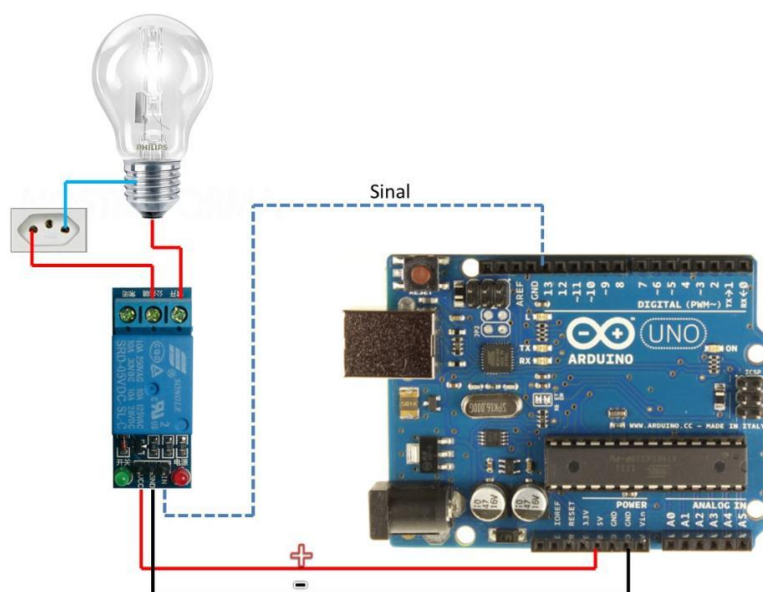
1.5 Prácticas realizadas

PRÁCTICA 1 (ARDUINO)

Objetivo: Es esta práctica controlaremos el encendido de una bombilla, para ello utilizaremos un relé

Introducción: En esta práctica, el relé funciona como un interruptor, que por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.

El circuito que tenemos que montar para que esto funcione es el siguiente:



Controlaremos el encendido y apagado de la bombilla a través de un script en lenguaje python, donde si enviamos el carácter 'L' (LOW) se apagará y al contrario si enviamos 'H' (HIGH) se encenderá.

Código utilizado:

- Código arduino:

```
int led = 13;
void setup () {
    pinMode(led, OUTPUT); //LED 13 como salida
    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);
        }
    }
}
```

- Script python:

```
import serial
arduino = serial.Serial('/dev/ttyACM0', 9600)
print("Empezando!")

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')

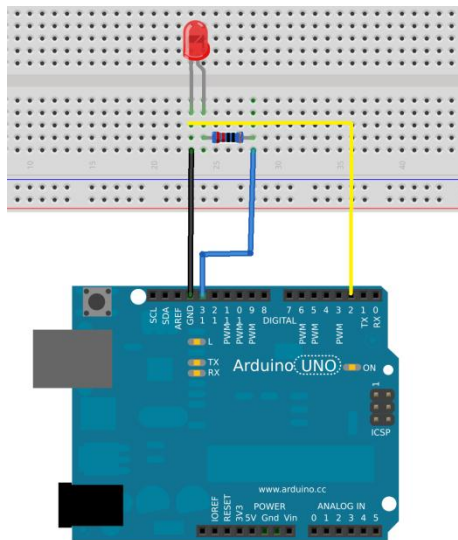
arduino.close() #Finalizamos la comunicacion
```

PRÁCTICA 2 (ARDUINO)

Objetivo: Controlar el encendido de un led utilizando el sistema de interrupciones de la placa de desarrollo.

Introducción: En esta práctica utilizaremos el sistema de interrupciones de la placa, concretamente utilizaremos la interrupción del pin 2, y cuando este pin intercepte una interrupción cambiaremos el estado del led, es decir, de encendido a apagado y viceversa.

Montaremos el siguiente circuito:



Código utilizado:

- Código arduino:

```
int led = 13;
int pinInte = 2;
volatile byte estado_actual = LOW;

void setup()
{
  pinMode(led, OUTPUT);
  pinMode(pinInte, INPUT_PULLUP);
  attachInterrupt(0, cambio_estado, FALLING);
}

void loop()
{
  digitalWrite(led, estado_actual);
}

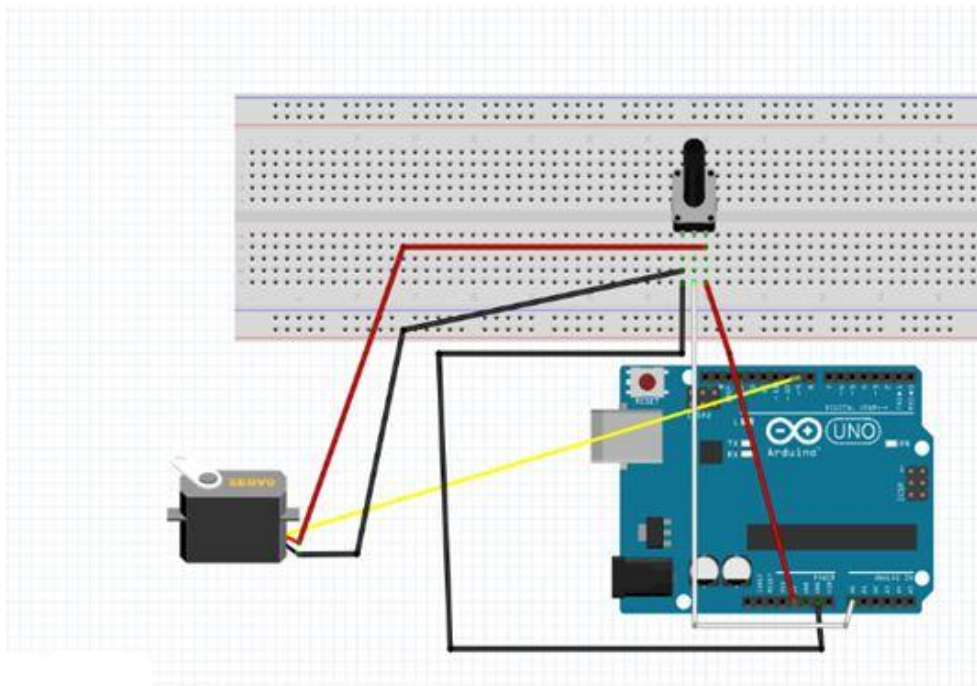
void cambio_estado()
{
  estado_actual = !estado_actual;
}
```


PRÁCTICA 3 (ARDUINO)

Objetivo: Controlar el ángulo de posicionamiento de un servo motor, enviando el ángulo a través del puerto serie.

Introducción: Para esta práctica tendremos que utilizar otro de los componentes electrónicos, un servo motor. Un servo motor es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición.

Montaremos el siguiente circuito, tal y como se muestra en la imagen:



Volveremos a utilizar un script en python para enviar por el puerto serie el ángulo que queremos desplazar del servo motor. En el código arduino haremos uso de la librería <Servo.h>.

Código utilizado:

- Código arduino:

```
#include <Servo.h>

int val = 0; //variable de entrada del serial
Servo servo; //Creamos un objeto Servo de nombre... servo

void setup()
{
  Serial.begin(9600); //Iniciamos el serial
  servo.attach(3); //Conectamos el servo al pin digital 3
}

void loop()
{
  if(Serial.available() > 0) //Detecta si hay alguna entrada por serial
  {
    val = Serial.parseInt();
    if(val != 0)
    {
      servo.write(val); //Mueve el servo a la posición entrada (excepto si es 0)
    }
  }
  delay(500);
}
```

- Script python:

```
import serial

arduino = serial.Serial('/dev/ttyACM0', 9600)
print("Starting!")

while True:
    comando = raw_input('Introduce un grado: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino

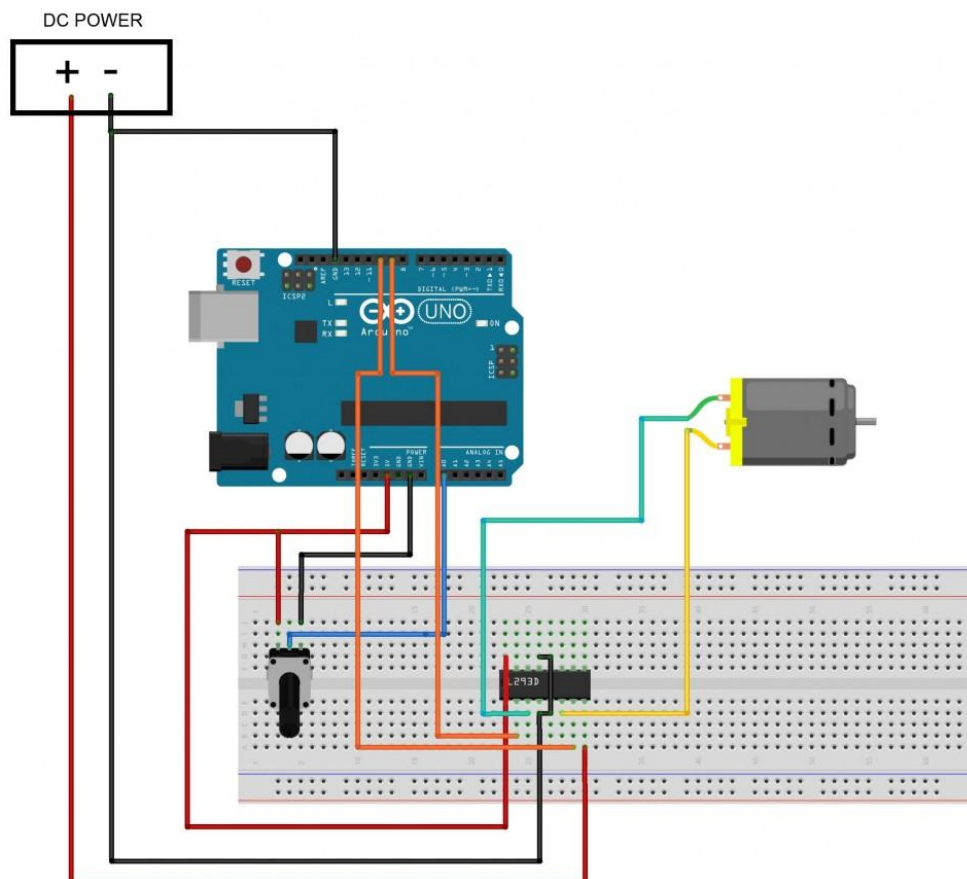
arduino.close() #Finalizamos la comunicacion
```

PRÁCTICA 4 (ARDUINO)

Objetivo: Controlar la velocidad y sentido de giro de un motor de corriente continua.

Introducción: En esta práctica haremos uso de otro de los componentes auxiliares, un puente H, en concreto el L293D. Un Puente en H es un circuito electrónico que permite a un motor eléctrico de corriente continua girar en ambos sentidos, *avance* y *retroceso*. Son ampliamente usados en robótica y como convertidores de potencia. Los puentes H están disponibles como circuitos integrados, pero también pueden construirse a partir de componentes discretos.

Montaremos el circuito de la siguiente imagen:



Cuando el potenciómetro este en el centro, el motor estará parado. Si giramos el potenciómetro a la derecha el motor girará en sentido horario y con una velocidad proporcional a la posición del potenciómetro, y cuando lo giramos a la izquierda el motor girará en sentido anti-horario con una velocidad proporcional a la posición del potenciómetro.

Código utilizado:

- Código arduino:

```
int pin2=9;    //Entrada 2 del L293D
int pin7=10;   //Entrada 7 del L293D
int pote=A0;   //Potenciómetro

int valorpote; //variable que recoge el valor del potenciómetro
int pwm1;      //Variable del PWM 1
int pwm2;      //Variable del PWM 2

void setup()
{
  pinMode(pin2,OUTPUT);
  pinMode(pin7, OUTPUT);
}

void loop()
{
  valorpote=analogRead(pote);

  pwm1 = map(valorpote, 0, 1023, 0, 255);
  pwm2 = map(valorpote, 0, 1023, 255, 0); //El PWM 2 esta invertido respecto al PWM 1

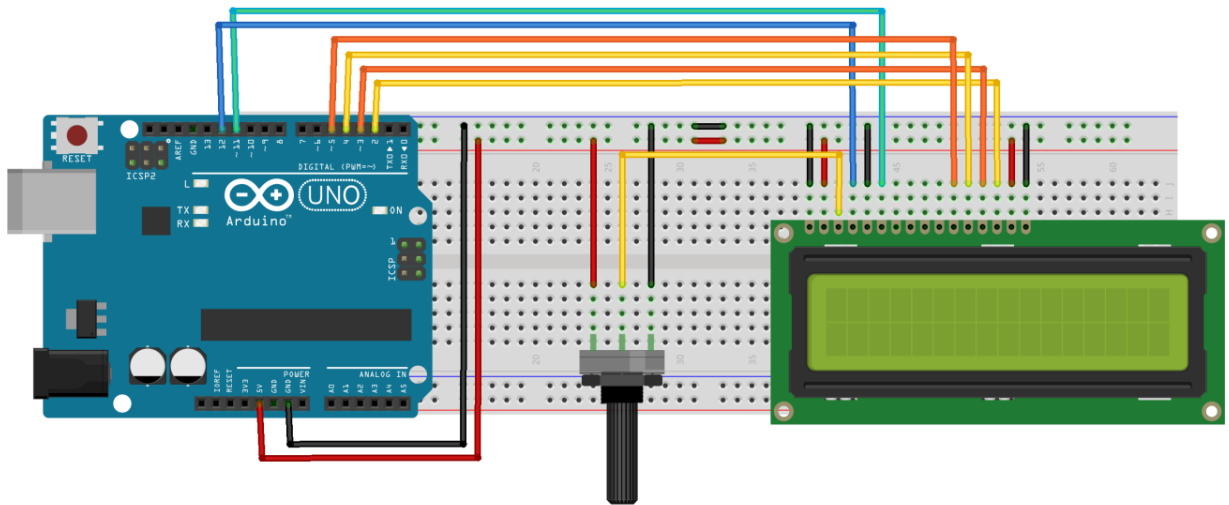
  analogwrite(pin2,pwm1);
  analogwrite(pin7,pwm2);
}
```

PRÁCTICA 5 (ARDUINO)

Objetivo: Visualizar en un LCD datos que enviamos a través del puerto serie.

Introducción: Para esta práctica utilizaremos la pantalla LCD LCM1602C, la cual es capaz de manejar LCD's de 2x20, 2x40, 4x20, 1x40 o 1x80. Esta pantalla LCD consta de 16 filas y 2 columnas, las cuales tendremos que ir desplazando, a través de un comando específico, para poder visualizar las distintas ventanas de visualización disponibles.

La pantalla LCD LCM1602C está basada en el controlador Hitachi HD44780 o compatible. Para montar el circuito correctamente, tenemos que hacerlo como en la siguiente imagen:



Código utilizado:

Importaremos la librería LiquidCrystal para el correcto funcionamiento.

- Código arduino:

```
#include <LiquidCrystal.h>
String muestra;
LiquidCrystal lcd(11, 10, 5, 4, 3, 2);

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

void loop()
{
  if(Serial.available()>0){
    muestra=Serial.readString();
    lcd.setCursor(0, 0);
    lcd.print(muestra);
  }
}
```

- Script python:

```
import serial

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

while True:
    comando = raw_input("Introduce lo que quiere mostrar :");
    arduino.write(comando);
arduino.close();
```

PRÁCTICA 6 (ARDUINO)

Objetivo: Utilizar un sensor de temperatura para diseñar un termómetro, el cual imprimirá en un LCD la temperatura.

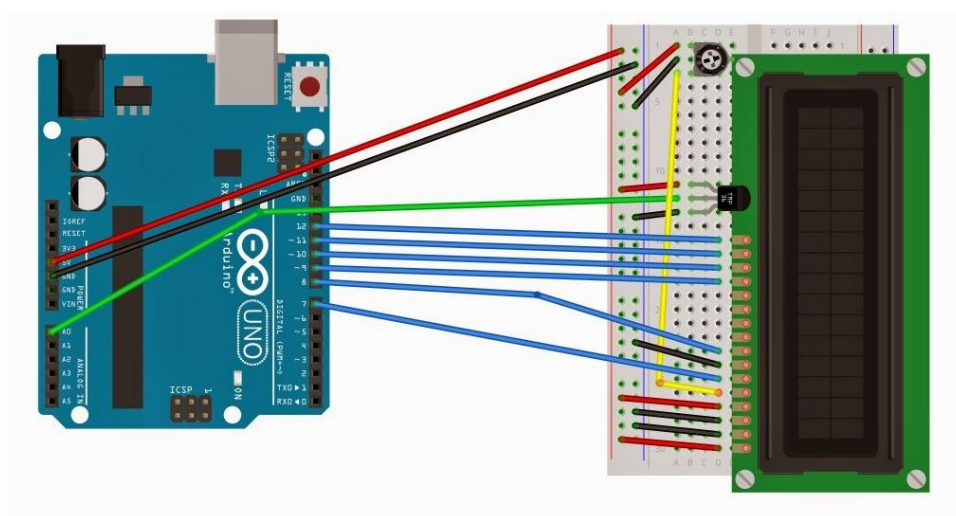
Introducción: El ADC interno de Arduino tiene un rango de entrada de 0 a 5V, y como tiene una resolución de 10 bits, convierte el valor de entrada en un rango de 0 a 1023 en digital. Para averiguar cuál es el valor de tensión de la entrada, se emplea la ecuación:

$$V_{int} = \frac{\text{Valor ADC} * 5}{1023} \text{ Voltios}$$

Para ello conectaremos el sensor de temperatura, que da una salida de tensión lineal a + 10.0mV/°C. Debemos convertir la tensión de entrada en temperatura, y lo haremos utilizando la ecuación:

$$T^{\circ} = V_{int} * 0.01$$

Para el funcionamiento de esta práctica, montaremos el siguiente circuito de la imagen:



Código utilizado:

- Código arduino:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
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);
}
```


PRÁCTICA 7 (ARDUINO)

Objetivo: Realizar un contador que cuente desde el 00 al 59.

Introducción: Para realizar esta práctica necesitaremos de dos dispositivos electrónicos llamados display 7 segmentos. El display 7 segmentos está compuesto de siete segmentos que se pueden encender o apagar individualmente. Cada segmento tiene la forma de una pequeña línea.

Utilizaremos una ampliación shield de arduino ya diseñada con los 2 displays, la cual podremos insertar directamente en la placa arduino. El display 7 segmentos es de la siguiente forma:

**Código utilizado:**

- Código arduino:

```
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() {
  for (int thisseg = 0; thisseg < 8; thisseg++) {
    pinMode(segPins[thisseg], OUTPUT);
  }
  pinMode(disp1, OUTPUT);
  pinMode(disp2, OUTPUT);
}
```

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

void refresh( int data1, int data0) {
  int tiempo_refresco = tiempototal/(2*j);
  int i;
  for (i = 0; i < j; i++){
    delay(tiempo_refresco);
    digitalWrite(displ1, 1);
    digitalWrite(displ2, 0);
    write_data(data1);
    delay(tiempo_refresco);
    digitalWrite(displ1, 0);
    digitalWrite(displ2, 1);
    write_data(data0);
  }
}

void write_data (int arg) {
  switch (arg) {
    case 0:
      write7seg(0x7e);
      break;
    case 1:
      write7seg(0x30);
      break;
    case 2:
      write7seg(0x6d);
      break;
    case 3:
      write7seg(0x79);
      break;
    case 4:
      write7seg(0x33);
      break;
    case 5:
      write7seg(0x5b);
      break;
  }
}
```

```
    case 6:
        write7seg(0x1f);
        break;
    case 7:
        write7seg(0x70);
        break;
    case 8:
        write7seg(0x7f);
        break;
    case 9:
        write7seg(0x73);
        break;
}
}

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; }
}
```

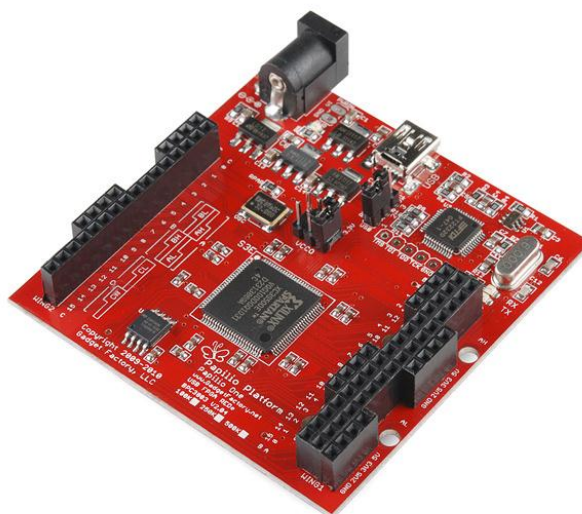
2. Papilio/ZPUino

2.1 Introducción

La placa **Papilio** es una placa de código abierto de gran alcance, y una placa de desarrollo ampliable perfecta para el diseño y creación de prototipos con ideas únicas. Con esta placa podemos personalizar las capacidades con alas integrables en expansiones modulares (similar a la shield de Arduino), que proporcionan una mayor funcionalidad a la placa, y al mismo tiempo poder ampliar las posibilidades creativas.

Para el desarrollo de las prácticas con la placa Papilio utilizaremos la Papilio One de 500k. En el fondo, la Papilio One de 500K tiene un chip FPGA Xilinx Spartan 3E, proporcionando una cantidad abundante de lógica digital para obtener rápidamente el diseño de prototipos. Además, se puede programar la FPGA utilizando el IDE de Arduino para escribir fácilmente código de Arduino y subirlo al procesador AVR de 8 bits. También podremos utilizarla como un procesador Soft Core ZPUINO de 32 bits corriendo a 96Mhz, simplemente cargando un nuevo archivo de configuración. Esta placa es ideal para el aprendizaje de diseño de circuitos digitales, VHDL, soft processors, y para el desarrollo de prototipos, soft modems, etc.

La placa Papilio One 500k que vamos a utilizar tiene el siguiente aspecto:



2.2 Objetivos

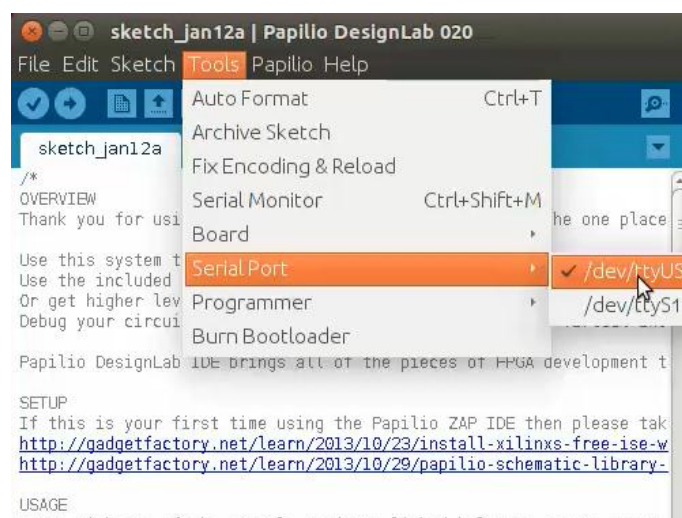
- Conocer la plataforma PAPILIO.
- Instalar y configurar el entorno de trabajo de papilio: DESIGN LAB .
- Conocer que se puede hacer desde DESIGN LAB sobre las placas PAPILIOS:
 - Diseñando circuitos a nivel de captura de esquemáticos e implementarlos en la FPGA.
 - Cargar el SoC ZPUINO.
 - Desarrollar 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.

2.3 Material empleado

Como hemos dicho antes, para el desarrollo de las prácticas utilizaremos la placa Papilio One 500k, y una serie de componentes electrónicos como switch y led o la propia placa de desarrollo arduino.

2.4 Entorno de desarrollo

El entorno de desarrollo que vamos a utilizar para programar sobre la placa Papilio es el Design Lab, basado en el entorno de desarrollo arduino con la misma interfaz gráfica, como se muestra en la imagen:



Para instalarlo debemos descárgalo desde:

<http://forum.gadgetfactory.net/index.php?/files/file/236-papilio-designlab-ide/>

Una vez descargado:

- Descomprimir el archivo y ejecutar el script ubuntu-setup.sh con permisos de superusuario.
- Debemos tener instalado Java, si no es el caso ejecutar el siguiente comando desde el terminal:

```
sudo apt-get install default-jre
```

- Tras esto ya podemos arrancar el entorno ejecutando desde la consola el siguiente comando:

```
./Designlab
```

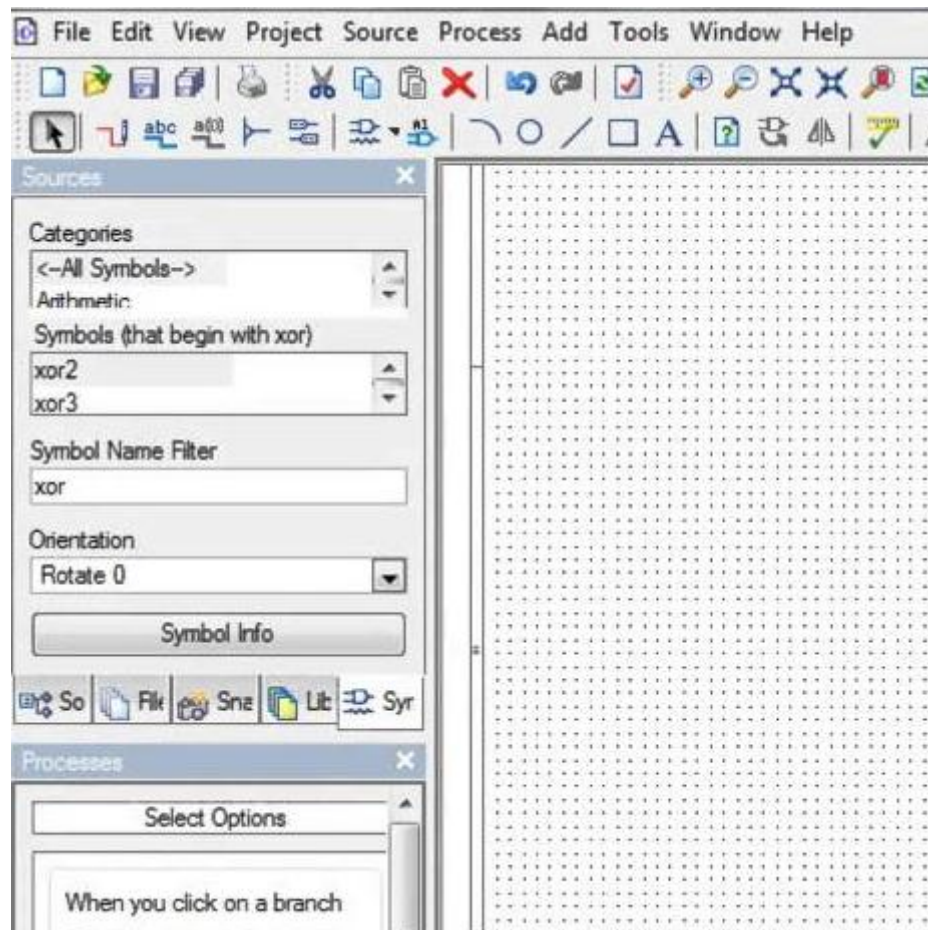
2.5 Prácticas realizadas

PRÁCTICA 1 (PAPILIO)

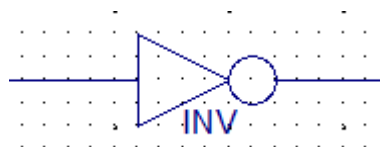
Objetivo: Controlar el encendido de un led con un switch haciendo uso del programa Xilinx.

Introducción: Comenzaremos la práctica creando un nuevo proyecto vacío, para ello buscamos el botón “Nuevo Circuito FPGA”. Guardaremos el proyecto para poder editarlo, a través del botón de edición. De esta forma se nos abrirá el programa Xilinx, donde tendremos que buscar nuestra placa.

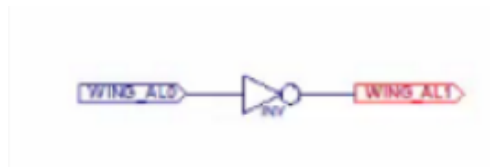
Al pulsar sobre ella se nos abrirá el editor donde podemos modificar el esquemático:



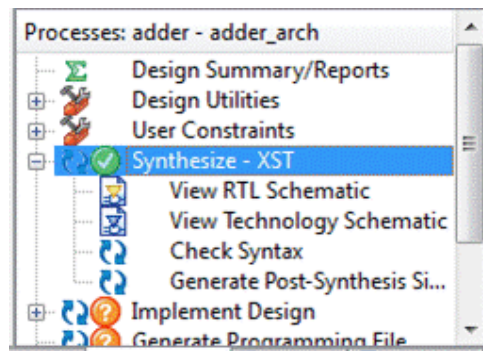
En el editor tendremos que buscar el símbolo inversor, para ello buscamos “inv” y añadimos el inversor al esquemático:



Buscamos dos conectores I/O y se lo conectamos al inversor en la entrada y la salida. Estos conectores los utilizaremos para conectarnos con la FPGA, para ello abriremos el diagrama de conexiones para ver los pines disponibles, en nuestro caso utilizaremos el WING_AL0 y WING_AL1. Seguidamente vamos al esquemático y editamos el nombre, como se muestra en la siguiente imagen:



Ya tenemos todo para poder sintetizar el circuito y generar el archivo de programación:



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

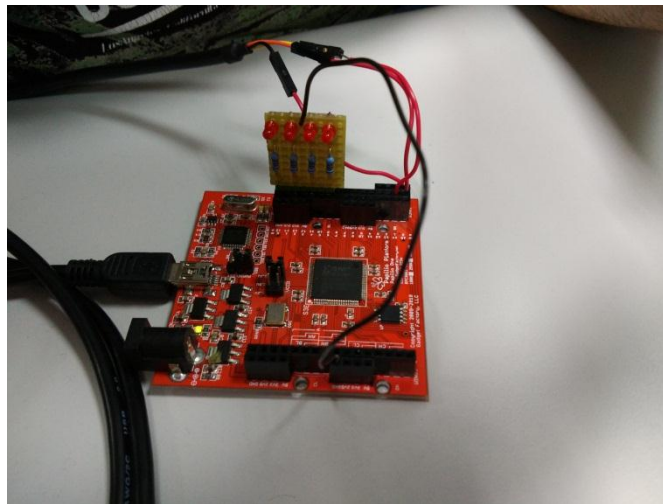
- Carpeta: <proyecto>/circuit/500K/
 - 1.- Borrar papilio_one_500k.bit
 - 2.- Renombrar Papilio_One_500K.bit a papilio_one_500k.bit

Por último cargamos el bitfile en la FPGA integrada en la placa papilio, montamos un circuito con un led conectado a un switch y comprobamos que funciona.

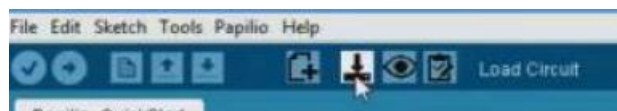
PRÁCTICA 2 (PAPILIO)

Objetivo: Controlar el encendido de un led con un switch a través del envío del comando adecuado a través del puerto serie.

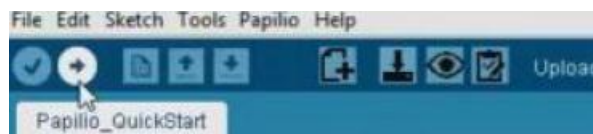
Introducción: Desarrollaremos un sketch para realizar esta práctica. Para ello, volvemos a montar el mismo circuito de la PRÁCTICA 1, desconectando el switch del circuito, y realizamos un script en python que nos permita encender y apagar el led a través de comandos enviados por el puerto serie.



Para subir el sketch a la placa, tenemos que primeramente seleccionar nuestro tipo de placa, y cuando tengamos realizado el sketch lo cargamos a la placa, a través del siguiente botón:



Por último debemos cargar el esquemático:



Código utilizado:

- Script python:

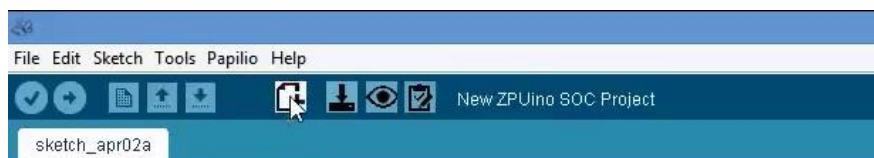
```
import serial
papilo = serial.Serial('/dev/ttyUSB1', 9600)
print("Starting!")

while True:
    comando = raw_input('Introduce un comando: ') #Input
    papilo.write(comando) #Mandar un comando hacia zpuino
    if comando == 'H':
        print('LED ENCENDIDO')
    elif comando == 'L':
        print('LED APAGADO')
    papilo.close() #Finalizamos la comunicación
```

PRÁCTICA 3 (PAPILIO)

Objetivo: Crear un nuevo circuito FPGA a través del editor de Xilinx y cargarlo en la placa.

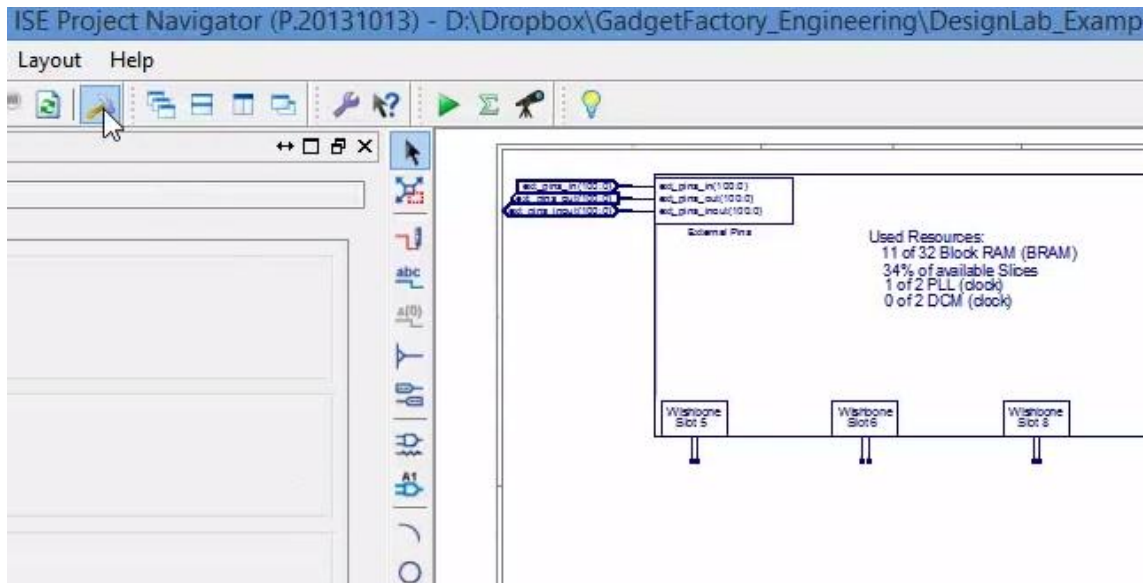
Introducción: Utilizaremos el editor de Xilinx para el desarrollo de esta práctica, para ello cuando estemos en la ventana de desarrollo arduino, creamos un nuevo proyecto ZPuino SOC:



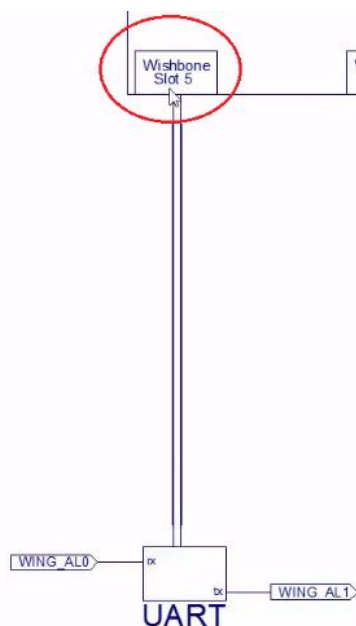
Llegados a este punto tenemos que modificar el circuito, pulsando en el botón de edición:



Buscamos nuestra placa y le damos doble click, con lo que se nos tenía que abrir el editor de esquemáticos, como se muestra en la siguiente imagen:



Seguidamente buscaremos el icono de la UART y lo añadimos al editor, después tenemos que unirlo con un Wishbone, que en nuestro caso hemos utilizado el Wishbone 5. Observamos que la UART tiene una entrada y una salida, Rx y Tx, las cuales son las encargadas de recibir y transmitir los datos. Configuraremos la entrada y la salida con los pines WING_AL0 Y WING_AL1 respectivamente:



Ya que hemos utilizado estos pines, tenemos que eliminarlos de la lista que tiene Papilio por defecto:



Cuando tengamos todo preparado sintetizamos el circuito y creamos el archivo de programación BitFile, como hemos hecho en prácticas anteriores.

NOTA: Igual que en las prácticas anteriores tendremos que borrar el archivo papilio_one_500k.bit y renombrar el archivo Papilio_One_500K.bit con el nombre del anterior.

Regresaremos a la ventana de sketches, y en esta ventana tendremos que añadir código al existente para configurar la UART que hemos integrado:

```

HardwareSerial mySerial1( WishboneSlot(5) );
// #define circuit ZPUino_Vanilla

int led = 13;

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

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

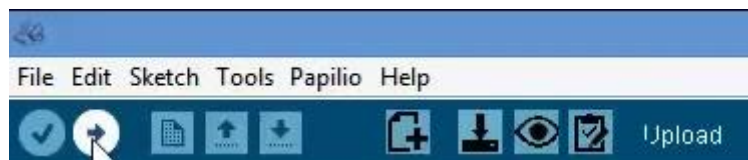
void loop() {
  mySerial1.write(1);
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the volt
  delay(1000);           // wait for a second
  digitalWrite(led, LOW); // turn the LED off by making the vc

```

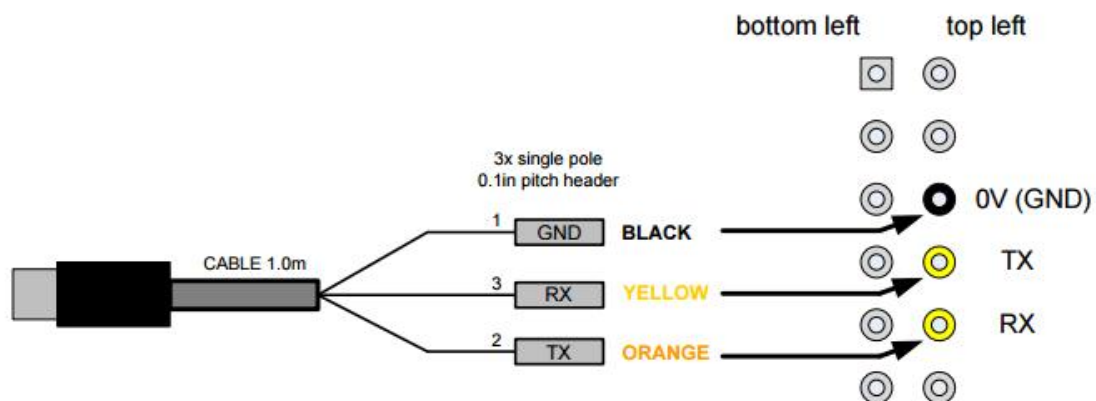
Finalmente escogemos el puerto utilizado por la placa Papilio y cargamos el circuito:



Esperamos hasta que el proceso termine, de esta forma ya podemos verificar que funciona, y por último subimos el código a la placa a través del siguiente botón:



Para probar el funcionamiento del circuito, conectaremos un led a la placa arduino, la cual también estará conectada a la placa Papilio y al pc. Controlaremos el encendido y apagado a través de la UART creada. La comunicación de la placa arduino con el pc será través de un cable TTL-RS232-USB. En la siguiente imagen podemos ver de qué forma conectamos el cable a la placa:

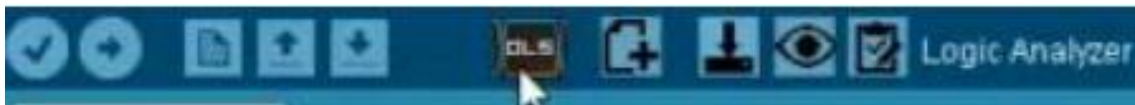


PRÁCTICA 4 (PAPILIO)

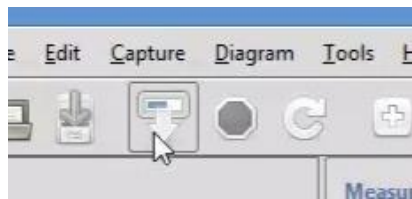
Objetivo: Utilizar la placa Papilio como un analizador lógico.

Introducción: En esta práctica haremos uso del analizador lógico de DesignLab para utilizar la placa como un analizador lógico que tendremos que configurar.

Abrimos el analizador lógico de DesignLab:

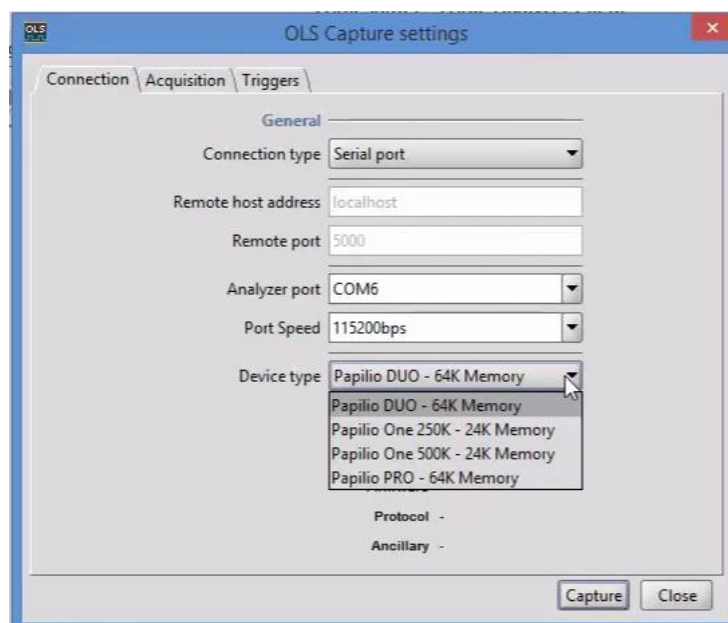


Abriremos la ventana de “Iniciar capturas de datos”:

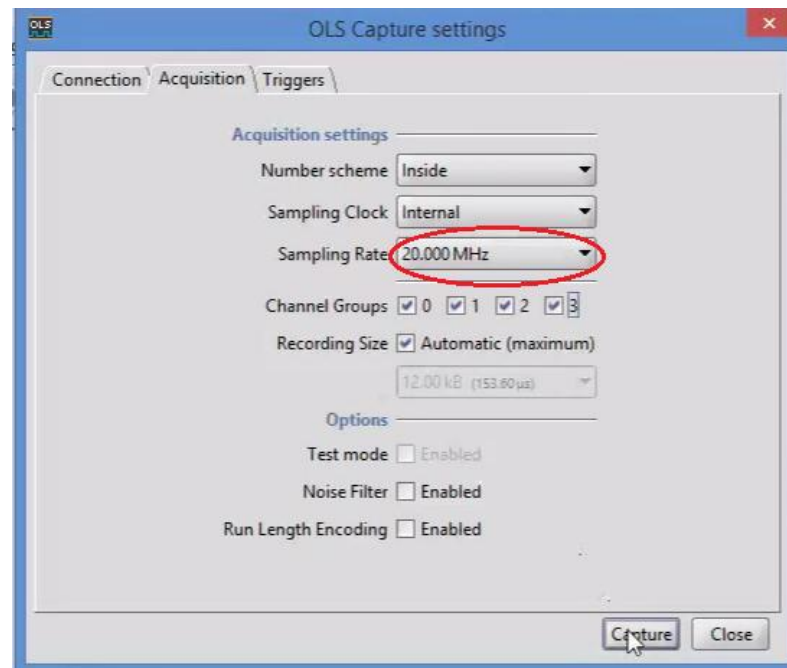


Cuando se nos abra la ventana podemos modificar la configuración del analizador. Para la práctica hemos utilizado la siguiente configuración:

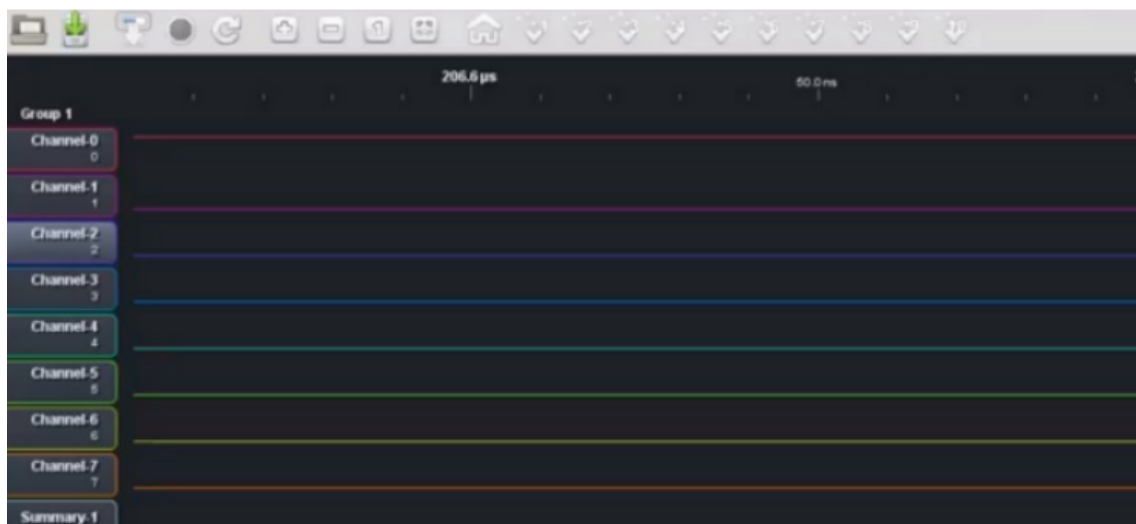
- En la pestaña Connection:



- En la pestaña Acquisition:



Por último pulsamos en el botón “Capture”, de esta forma empezaremos a capturar las entradas. Para ver que funciona correctamente, conectaremos 5V al canal 0, con lo que nos tendría que salir un 1 lógico en el canal 0 del visor, como se muestra en la siguiente imagen:



3. Raspberry PI

2.1 Introducción

Raspberry Pi es un ordenador de placa reducida o placa única (SBC) 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.

El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos “Turbo” para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía), un procesador gráfico (GPU) VideoCore IV, y 512 MB de memoria RAM (aunque originalmente al ser lanzado eran 256 MB).

El diseño no incluye un disco duro ni unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente. Tampoco incluye fuente de alimentación ni carcasa.

Dentro del hardware disponemos de 1 controlador Ethernet, 4 puertos USB , 1 salida HDMI para video y un lector de tarjetas SD. No tiene reloj en tiempo real, por lo que el S.O. debe conectar con un servidor de hora en red, o ser inicializado al arrancar.

2.2 Objetivos

- 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

2.3 Material empleado

Para el desarrollo de estas prácticas utilizaremos en concreto la placa Raspberry PI. Utilizaremos también para alguna práctica la placa arduino y haremos uso de componentes electrónicos como LED's y switches.

La Raspberry PI 3 tiene el siguiente aspecto:



2.4 Entorno de desarrollo

Para trabajar sobre la Raspberry PI necesitaremos de instalarle un sistema operativo en una tarjeta SD que podremos conectar a la Raspberry PI a través del lector. En concreto instalaremos el Ubuntu Mate 16.04, con lo que tendremos que descargarnos la imagen. Después seguimos los siguientes pasos:

- Primeramente tendremos que preparar la SD para poder cargarle el sistema operativo. Para ello la formateamos con Gparted, sino lo tenemos instalado introducimos el siguiente comando `"$ sudo apt-get install gparted"`.

- Conectamos la tarjeta SD a un PC a través de un lector de tarjetas e introducimos el siguiente comando para guardar el sistema operativo:

```
$ dd bs=4M if=<nombre_imagen>.img of=/dev/sdd
```

- Una vez preparada la tarjeta conectamos la SD a la RaspberryPi (RPI), desconectamos el teclado, el ratón, el cable Ethernet del PC y los conectamos a la RPI. Conectamos el cable HDMI-DVI a la RPI y al monitor de un pc. Por último conectamos el cable USB-Micro usb a la RPI (microusb) y PC para alimentar la RPI.
- Instalamos el sistema operativo (puede tardar de 10 a 14 minutos).
- Al arrancar por primera vez el sistema operativo tenemos que cambiar la configuración, para ello entramos en:
 - Chage passwd: poner un passwd conocido.
 - Enable desktop : para arrancar directamente en entorno gráfico.
 - Internazionalization: Change Locale: seleccionar es_ES; es_ES.UTF-8;es_ES@Euro; change keyboard: generic 105; Spanish.
 - Finalmente reiniciamos.
- Una vez arrancado el SO hay que configurar la conexión de Internet. Editar /etc/network/interfaces:
 - iface eth0 inet static (cambiar dhcp por static)
 - address 10.1.15.xx (xx-numero del pc: RD-xx)
 - netmask 255.255.252.0
 - gateway 10.1.15.78
 - dns-nameservers 8.8.8.8

- Por último introducimos los siguientes comandos como superusuario:
 - \$ sudo ifdown eth0
 - \$ sudo ifup eth0

Si todo ha ido bien, ya tendremos la placa Raspberry PI 3 lista para su uso. La interfaz del escritorio es la siguiente:



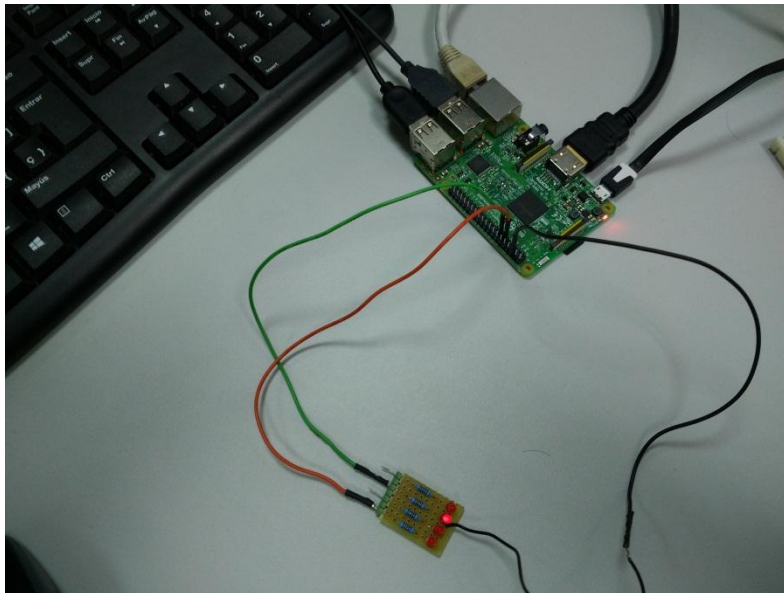
2.5 Prácticas realizadas

PRÁCTICA 1 (RASPBERRY PI)

Objetivo: Controlar el encendido y apagado de dos LED's mediante un script de python.

Introducción: Para la realización de esta práctica tenemos que usar los pines de la Raspberry PI 3, los llamados GPIO's. Para poder usar estos pines tendremos que configurarlos, en nuestro caso utilizaremos los pines GPIO 17 y 21. Para ello seguimos los siguientes pasos:

- Montamos el circuito de la siguiente imagen:



- Configuramos los puertos:
 - `echo 17 > /sys/class/gpio/export`
 - `echo 21 > /sys/class/gpio/export`
- Tendremos que informar a la Raspberry PI de que los pines 17 y 21 son de salida, ya que lo que queremos es encender un LED.
 - `echo out > /sys/class/gpio/gpio17/direction`
 - `echo out > /sys/class/gpio/gpio21/direction`

- Para probar que funciona introducimos los siguientes comandos:
 - Si queremos encender los LED's:
 - `echo 1 > /sys/class/gpio/gpio17/value`
 - `echo 1 > /sys/class/gpio/gpio21/value`
 - Si queremos apagar los LED's:
 - `echo 0 > /sys/class/gpio/gpio17/value`
 - `echo 0 > /sys/class/gpio/gpio21/value`
- Por último, tenemos que liberar el puerto, con el siguiente comando:
 - `echo 17 > /sys/class/gpio/unexport`
 - `echo 21 > /sys/class/gpio/unexport`

Cuando hayamos comprobado que todo ha funcionado realizaremos un script en python a través del cual podemos encender y apagar un LED. El montaje del circuito es el mismo que hemos hecho antes, pero esta vez mediante el script el python haremos que parpadeen.

Para poder controlar los GPIO's por python tenemos que instalar la librería que los controla, para ello ejecutamos el siguiente comando en la Raspberry PI:

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

Cuando se haya descargado descomprimos el archivo “.tar.gz” y accedemos a él mediante el comando “\$ cd RPi.GPIO-0.5.4/”.

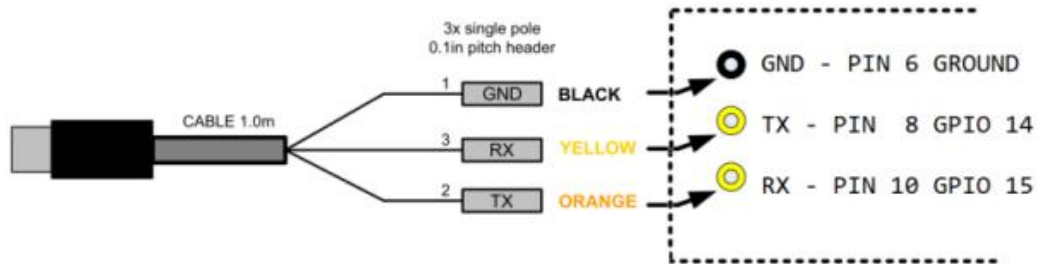
NOTA: Antes de instalar la librería tenemos que tener instalado python, si no lo tenemos ejecutamos el siguiente comando “\$ sudo apt-get install python-dev”.

Procedemos a la instalación de la librería, ejecutando el siguiente comando:

- `sudo python setup.py install`

Seguidamente creamos el script de python con el nombre “blink.py” (código más abajo) mediante el comando “\$ sudo nano blink.py”.

Llegados a este punto utilizaremos un cable TTL-232R-RPi para conectar la placa con el PC a través del puerto serie, conectando los pines de la siguiente manera:



Posteriormente configuramos el puerto serie con las siguientes características:

- * Speed: 115200 baud
- * Bits: 8
- * Parity: None
- * Stop Bits: 1
- * Flow Control: None

Finalmente, lanzamos el código de python y comprobamos que los LED's conectados a la Raspberry PI parpadean como hemos programado en el script de python.

Código utilizado:

- Script de python:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT) ## GPIO 17 como salida
GPIO.setup(21, GPIO.OUT) ## GPIO 21 como salida

def blink():
    print "Ejecucion iniciada..."
    iteracion = 0
    while iteracion < 30:
        GPIO.output(17, True) ## Enciende el 17
        GPIO.output(21, False) ## Apaga el 21
        time.sleep(1)
        GPIO.output(17, False) ## Apaga el 17
        GPIO.output(21, True) ## Enciende el 21
        time.sleep(1)
        iteracion = iteracion + 1
    print "Ejecucion finalizada"
    GPIO.cleanup() ## Hago una limpieza de los GPIO

blink() ##Llamamos a la función
```

PRÁCTICA 2 (RASPBERRY PI)

Objetivo: Controlar el encendido y apagado de dos LED's a través de una interfaz web alojada en la Raspberry PI a la que accederemos remotamente desde otro PC.

Introducción: En esta práctica haremos que se enciendan y apaguen dos LED's a través de un PC conectado remotamente a la Raspberry PI, mediante la creación de un servidor web. Para instalar el servidor web ejecutaremos dos comandos para instalar el python-pip y sus dependencias:

- sudo apt-get install python-pip
- sudo pip install flask

Para probar que funciona tenemos que conectarnos a la Raspberry PI desde otro PC remotamente, ejecutando el siguiente comando:

- `ssh -X <nomUsuRPI>@<IP_RPI>`

Creamos un “HOLA MUNDO” en la Raspberry PI con el nombre “hello-template.py” y añadimos el siguiente código:

```
##Fichero hello-template.py

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

@app.route("/")
def hello():
    now = datetime.datetime.now()
    timeString = now.strftime("%Y-%m-%d %H:%M")
    templateData = {
        'title' : 'HOLA!',
        'time': timeString
    }
    return render_template('main.html', **templateData)

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

Necesitaremos una interfaz intermedia como podemos observar en el script para intermediar entre el script y el usuario. Para ello, nos creamos un archivo “main.html” donde crearemos una web simple. Añadimos el siguiente código al archivo:

```
/-- Fichero 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>
```


Por último para ver si funciona accedemos al directorio que hemos creado este archivo a través del terminal y ejecutamos el siguiente comando, el cual nos lanza el script de python:

- `sudo python hello-template.py`

A través del navegador del PC con el que estamos conectados remotamente a la RPI introducimos la IP de la RPI, y nos mostrará el mensaje que hemos programado en el código.

Hecho todo esto ya estamos preparados para acceder a controlar dos LED's desde el navegador web, para ello creamos un nuevo script de python llamado "weblamp.py" y una nueva carpeta con el nombre "templates" y creamos el fichero "main.html" dentro de esta carpeta (códigos más abajo).

Llegamos al mismo punto anterior, ya creados estos dos ficheros tenemos que volver a lanzar el script de python con el siguiente comando:

- `sudo python weblamp.py`

Finalmente volvemos a conectarnos en el navegador a la IP de la RPI y nos debería aparecer la opción de apagar y encender los LED's utilizados. El circuito que hay que montar es el mismo que el que montamos en la PRÁCTICA 1 de Raspberry Pi.

Código utilizado:

- Script en python weblam.py:

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

pins = {
    17 : {'name' : 'LED', 'state' : GPIO.LOW},
    21 : {'name' : 'LED', '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 = {
        'pins' : pins
    }
    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
    }
    return render_template('main.html', **templateData)

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

- Archivo web main.html:

```
/--main.html --/  
<!DOCTYPE html>  
<head>  
  <title>Current Status</title>  
</head>  
  
<body>  
  <h1>Device Listing and Status</h1>  
  {% for pin in pins %}  
  <p>The {{ pins[pin].name }}  
  {% if pins[pin].state == true %}  
    is currently on (<a href="/{{pin}}/off">turn off</a>)  
  {% else %}  
    is currently off (<a href="/{{pin}}/on">turn on</a>)  
  {% endif %}  
</p>  
  {% endfor %}  
  {% if message %}  
  <h2>{{ message }}</h2>  
  {% endif %}  
</body>  
</html>
```

PRÁCTICA 3 (RASPBERRY PI)

Objetivo: Calcularemos la temperatura con un sensor de temperatura que añadiremos a lo realizado en la práctica anterior.

Introducción: Para esta práctica lo que haremos será añadir a la práctica anterior un nuevo circuito en el que obtendremos la temperatura actualizada en tiempo real.

La realización de esta práctica también será remotamente a través de otro ordenador, conectándonos a la Raspberry PI remotamente mediante el siguiente comando:

- `ssh -X <nomUsuRPI>@<IP_RPI>`

Después de estar conectados remotamente modificaremos los archivos necesarios a través del PC en el que estamos conectado remotamente con la RPI. Tendremos que modificar los dos archivos creados en la práctica anterior, los archivos “weblamp.py”, que es donde tenemos montado el servidor Python Flask, y “main.html” donde se encuentra el código html.

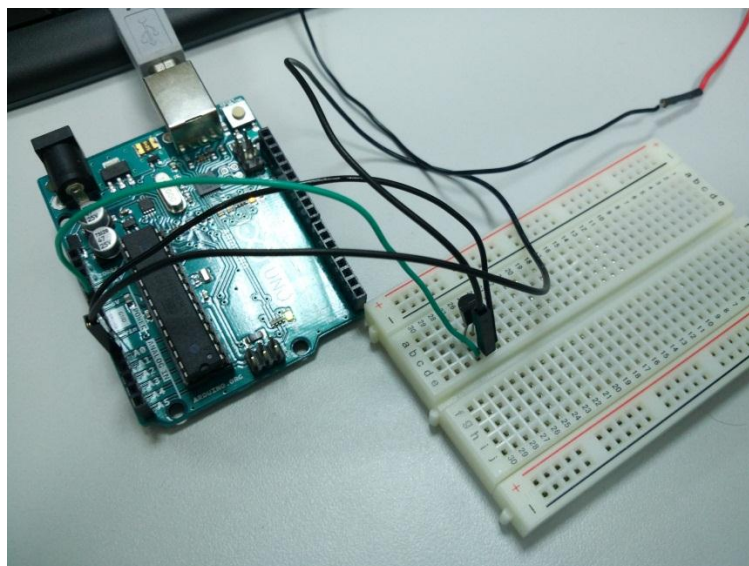
También tenemos que cargar un nuevo programa en Arduino donde añadiremos el siguiente código, que consiste en que Arduino escriba en el puerto serie el valor de la temperatura:

```
const int pinAnalogico =A0;

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

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

Montamos el circuito de la temperatura junto con el de los LED's en la placa Arduino:



Código utilizado:

- Modificación de weblamp.py:
 - Añadimos el código para configurar el puerto serie para leer el valor de temperatura:

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

- Añadimos una nueva ruta para la temperatura:

```
@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)
```

- Modificación de main.html:
 - Añadimos al main.html el siguiente código:

```
<h1>Lectura de temperatura: </h1>

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

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

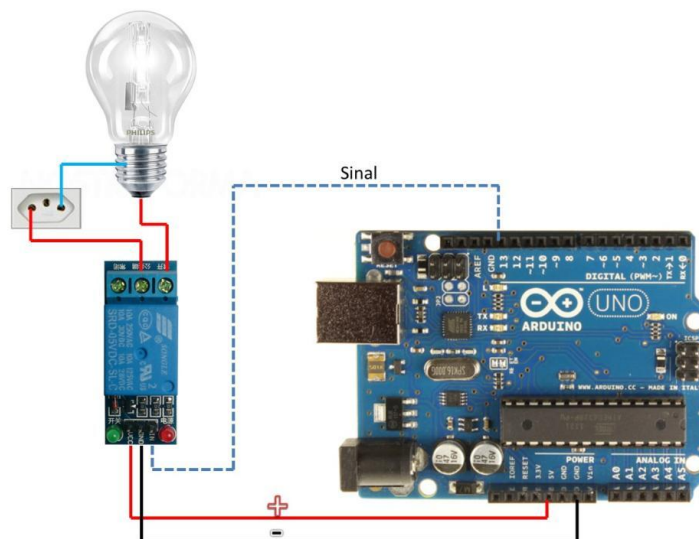
<h1>Bombilla: </h1>
```

PRÁCTICA 4 (RASPBERRY PI)

Objetivo: Controlaremos el encendido y apagado de una bombilla que añadiremos a la práctica anterior. En total tendremos el control de la bombilla, los LED's y el cálculo de la temperatura.

Introducción: Como hemos dicho en esta práctica añadiremos código a “weblamp.py” y “main.html” de forma que podamos controlar el encendido y apagado de una bombilla a través del servidor web formado en la Raspberry PI haciendo uso de un relé.

También haremos la realización de esta práctica a través de un ordenador conectado a la Raspberry PI remotamente a través del comando ssh, mencionado en las dos últimas prácticas. Volveremos a utilizar la placa de Arduino donde añadiremos un nuevo circuito conectando un relé el cual está conectado a una bombilla. Añadiremos el mismo circuito de la práctica 1 de Arduino, pero con la diferencia de que el Arduino esta vez está conectado a la Raspberry PI:



Tras la modificación de los archivos “weblamp.py” y “main.html” (código a añadir más abajo en “Código utilizado”) podemos ver en el servidor web como tendremos la opción de apagar y encender los LED’s, el valor de la temperatura tomada por el sensor y la opción de apagar y encender la bombilla. De nuevo tendremos que crear en “weblamp.py” una nueva ruta donde está vez es la Raspberry PI la que escribe en el puerto serie y el Arduino escucha, con lo que tenemos que modificar también el sketch de arduino y volverlo a cargar en la placa.

Cuando hayamos modificado los archivos lanzamos el comando “\$ sudo python weblamp.py” y podremos observar lo dicho antes.

Código utilizado:

- Modificación sketch arduino, así tiene que quedar en total:

```
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);
}
```

- Modificación weblamp.py:

```
@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)
```

- Modificación main.html:

```
<h1>Bombilla: </h1>

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

<p>Quiero apagarla (<a href="/on">Apagar</a>)</p>
```