



Escuela Técnica Superior

INGENIERÍA INFORMÁTICA

LABORATORIO DE DESARROLLO HARDWARE

MEMORIA DE PRÁCTICAS

Ricardo Reina Martín.

Curso 2016/2017

UNIVERSIDAD DE SEVILLA

Índice:

Contenido

Índice:.....	2
1. ARDUINO	3
1.1. Objetivos.	3
1.2. Introducción. ¿Qué es Arduino?	3
1.3. Puesta en marcha del entorno de desarrollo.....	4
1.4. Desarrollo de las prácticas.	5
Práctica 1. Encendido/Apagado de una bombilla mediante un relé.....	5
Práctica 2. Interrupciones Arduino.	7
Práctica 3. PC Servo.....	8
Práctica 4. Motor CC.....	10
Práctica 5. Display: Serie.	11
Práctica 6. Display: Termómetro.....	14
Práctica 7. Contador.....	16
2. PLATAFORMA PAPILIO/ZPUINO	20
2.1. Objetivos.	20
2.2. Introducción.	20
2.3. Puesta en marcha del entorno de desarrollo.....	21
2.4. Desarrollo de las prácticas.	21
2.4.1. Inversor.	21
2.4.2. Pc-led. Cargar SoC ZPUINO y desarrollar SKETCHES.....	24
2.4.3. Analizador lógico.	25
2.4.4. Rediseñando el SoC. Añadiendo periféricos.	27
3. RASPBERRY PI	28
3.1. Introducción.	28
3.2. Objetivos.	28
3.3. Puesta en marcha del entorno de desarrollo.....	28
3.4. Desarrollo de las prácticas.	30
3.4.1. Gpio-Python.	30
3.4.2. Servidor web GPIO.	32
3.4.3. Servidor web Temperatura.	36
3.4.4. Servidor web luz.	40

1. ARDUINO

1.1. Objetivos.

Conocer la plataforma Arduino, sus características, sus variantes, sus modos de programación y conocer una serie de componentes básicos de hardware típicos de aplicaciones de sistemas empotrados.

Para ello lo principal es aprender a preparar el PC para que funcione el entorno de desarrollo de Arduino para poder realizar ejemplos básicos de funcionamiento sobre Arduino y desarrollar otros ejemplos de uso de Arduino manejando diversos componentes hardware, estos componentes se irán conociendo a medida que se vaya avanzando.

1.2. Introducción. ¿Qué es Arduino?

Arduino es una plataforma de desarrollo de hardware abierta (open Hardware) 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 tipo de actuadores.

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 principales de Arduino.

- ➔ Un entorno de desarrollo muy fácil de manejar (basado en processing, <https://processing.org/>).
- ➔ Un amplio conjunto de librerías de manejo de periféricos.
- ➔ Una comunidad de desarrolladores muy grande.

La placa que se va a utilizar es la Arduino Uno y las características de este modelo son:

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

1.3. Puesta en marcha del entorno de desarrollo.

Vamos a desarrollar las prácticas en el sistema operativo Ubuntu de Linux y para poder poner en marcha el sistema hay que seguir los siguientes pasos:

1. Se descarga el software del entorno de desarrollo (IDE).
2. Se conecta la placa de desarrollo al PC mediante un cable USB.
3. Se instalan los drivers necesarios (descargados).
4. Se ejecuta la aplicación Arduino.
5. Se abre el ejemplo del parpadeo (blink example).
6. Se selecciona el tipo de placa que tenemos.
7. Se selecciona el puerto serie de comunicación con la misma.
8. Se transfiere el programa a la placa.

1.4. Desarrollo de las prácticas.

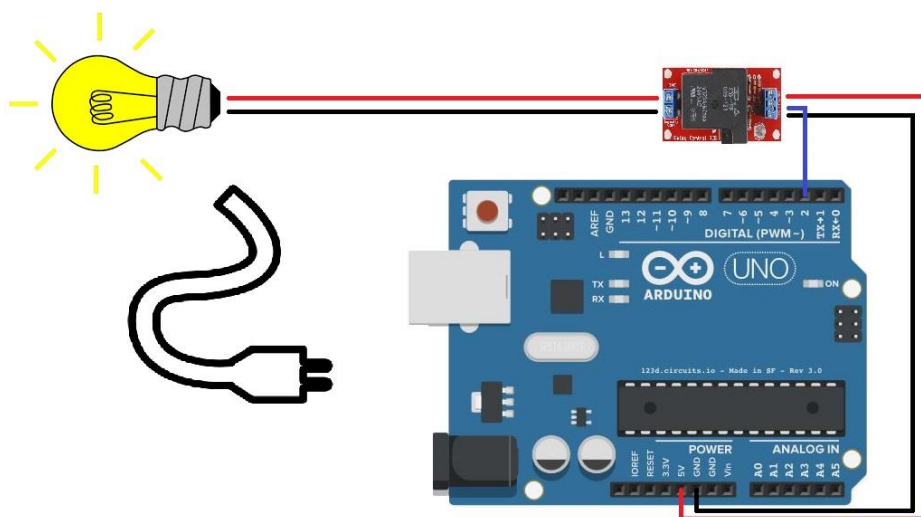
Práctica 1. Encendido/Apagado de una bombilla mediante un relé.

Objetivos.

Debemos ser capaces de encender y apagar una bombilla mediante un relé.

Introducción.

Se va a usar en esta práctica uno de los elementos que es el relé. Un relé es un electromagnético que, estimulado por una corriente eléctrica muy débil, abre o cierra un circuito en el cual se disipa una potencia mayor que en el circuito estimulador. El esquema del circuito se ha montado es el siguiente:



El código se probó para comprobar que el relé conmutaba. Posteriormente se conectó la bombilla, y una vez ejecutado el archivo Python escribiendo "H" y "L" se apagaba y se encendía, respectivamente.

Problemas detectados:

Tuve problemas con el archivo python ya que desconocía que es fundamental en dicho lenguaje de programación dejar sangría con el que se identifica el bloque al que pertenece ya que no se usan llaves como en otros lenguajes de programación. Solucionado esto me daba un error con el puerto serie ya que el Arduino estaba usando el puerto serie diferente al escrito en el código. Lo cambié y ya funcionó.

Código en Arduino.

```
int led = 2;
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/ttyACM1', 9600)
print("Epezamos!")
while True:
  comando = raw_input('Introduce un comando: ')
  arduino.write(comando) # Se manda un comando hacia Arduino
  if comando == 'H':
    print('LED ENCENDIDO')
  elif comando == 'L':
    print('LED APAGADO')
arduino.close() # Se finaliza la comunicacion
```

Práctica 2. Interrupciones Arduino.

Objetivos.

Debemos ser capaces de encender/apagar un led mediante un pulsador, empleando las interrupciones en arduino.

Desarrollo..

En el siguiente enlace <https://www.arduino.cc/en/Reference/AttachInterrupt> podemos encontrar información para trabajar con las interrupciones en arduino y el código usado en esta práctica el cual habrá que modificar para conseguir nuestros objetivos.

En esta práctica se hace uso de uno de los elementos auxiliares como es el pulsador. Se ha usado la configuración pull-up para activar la interrupción en el pin 2, el led de visualización será el 13, con el que no necesitaremos un led extra ya que arduino permite ver su estado con un pequeño led en la misma placa.

Problemas detectados:

Se detecta un problema con el pulsador ya que no funciona bien a veces ya que parece que a veces que “coge” el impulso y otras veces no, esto ocurre por el rebote. Hay formas de solucionar ese problema mediante por software.

Código en Arduino.

```
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, RISING);
}

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

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

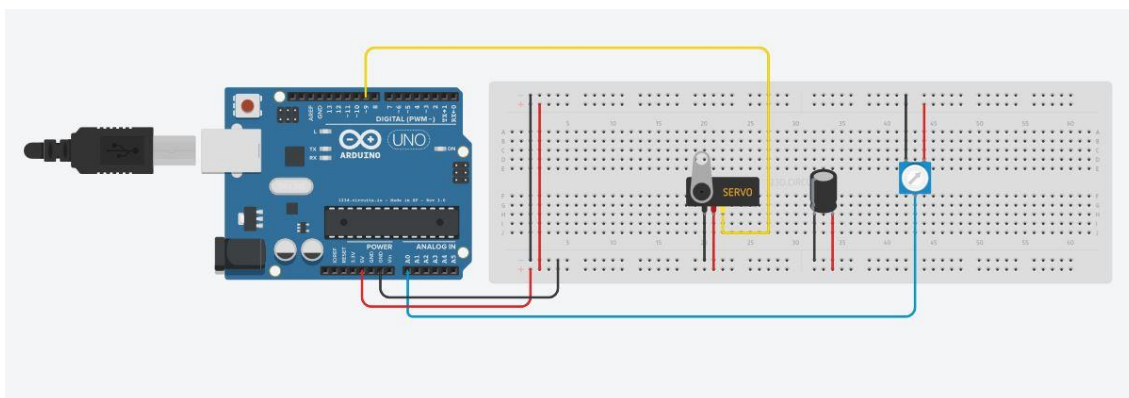
Práctica 3. PC Servo.

Objetivos.

Debemos ser capaces de controlar la posición de un servo motor con un potenciómetro. Una vez conseguido, hacerlo desde el PC, enviando el valor de ángulo (desde 0 a 180 grados) a través del puerto serie.

Introducción.

En esta práctica haremos uso de otro de los elementos auxiliares, el servo. El servo 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. En primer lugar, montamos el circuito tal como se muestra en la imagen:



Desarrollo:

Para poder trabajar con el servo hay que cargar las librerías del servo de Arduino. Entonces, ya se podrá controlar el servo de forma manual con un potenciómetro.

Código en Arduino para controlar el servo con el potenciómetro:

```
#include <Servo.h>
Servo myservo; // crear objeto servo para controlar un servo
int potpin = 0; // pin analógico usado para conectar el potenciómetro
int val; // variable para leer el valor del valor del pin analógico
void setup() {
  myservo.attach(9); // conecta el servo del pin9 al objeto servo
}
void loop() {
  val = analogRead(potpin); // lee el valor del potenciómetro(entre 0 y 1023)
  val = map(val, 0, 1023, 0, 180); // se escala para usarlo con el servo (entre 0 y 180)
  myservo.write(val); // establece el servo en la posición acorde al valor escalado
  delay(15); // espera
}
```


Código en arduino para controlar servo desde el PC:

```
#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/ttyACM2', 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. Motor CC

Objetivos.

Debemos ser capaces de controlar la velocidad de giro de un motor de corriente continua con un potenciómetro. Una vez controlado, controlar la velocidad desde el PC, enviando el valor de la velocidad a través del puerto serie.

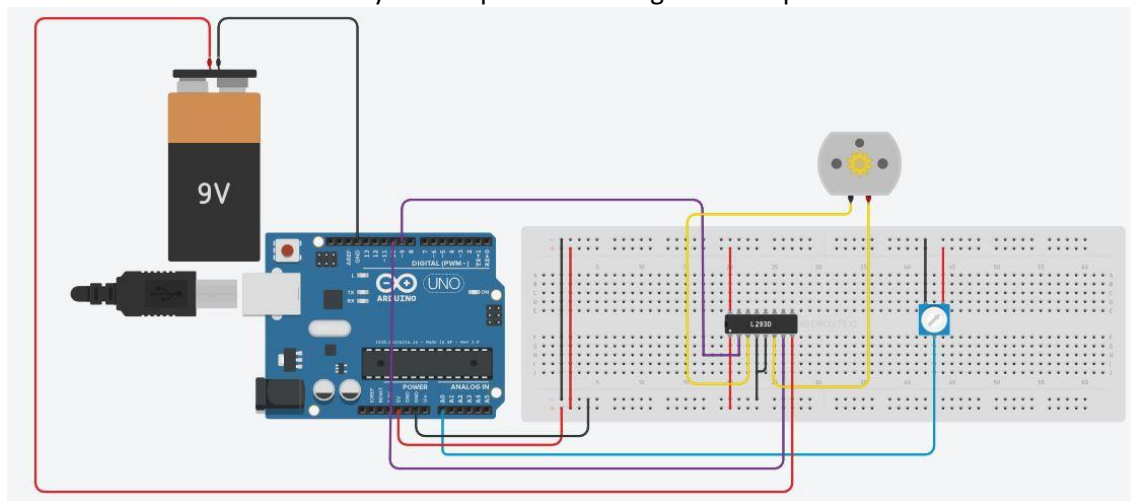
Introducción.

En esta práctica vamos a usar un componente llamado **punte H** (vamos a usar en concreto el L293DNE), que actúa entre Arduino y **el motor de corriente continua**.

El puente H es un circuito que permite intercambiar la polaridad en las señales de salida con señales de control de entrada.

Desarrollo:

Vamos a montar el circuito tal y como aparece en el siguiente esquema:



El funcionamiento es sencillo, cuando el potenciómetro esté en la posición central, el motor estará parado, cuando se gire a la izquierda, el motor girará en el sentido anti-horario, con una velocidad proporcional a la posición del potenciómetro y lo mismo cuando se gire a la derecha, el motor girará en sentido horario con una velocidad proporcional a la posición del potenciómetro.

Código en Arduino:

```
int pin2=9; //Entrada 2
int pin7=10; //Entrada 7
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()
{
  //Inicializamos los pins de salida
  pinMode(pin2,OUTPUT);
  pinMode(pin7, OUTPUT);
}
void loop()
```

Se deberá ser capaz de imprimir datos en una pantalla LCD.

Introducción.

En esta práctica haremos uso de una pantalla LCD, concretamente la pantalla LCM1602C, basada en controlador Hitachi HD44780 o compatible.

El display LCD que vamos a utilizar, es de 16 filas, y 2 columnas la visualización de caracteres, pero internamente, está fabricada con un chip capaz de manejar LCD's de 2x20, 2x40, 4x20, 1x40 o 1x80. Debido a esto, si tratamos de escribir un dato en la columna > 15, éste se escribirá pero no se podrá ver. Para poder visualizarlo, hay que utilizar un comando específico que desplaza la ventana de visualización.

Desarrollo:

Primero deberemos ser capaces de imprimir en dos filas con desplazamiento a la izquierda.

➔ Fila 1: Nombre de la asignatura.

➔ Fila 2: Nombre del alumno.

Para poder utilizar el display con Arduino, hay que cargar las librerías de control LCD (LiquidCrystal.h).

Código en Arduino:

```
#include <LiquidCrystal.h> //Importamos la librería LiquidCrystal
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //Creamos la variable y establecemos los pins del display
void setup()
{
  lcd.begin(16, 2); //Inicializamos el display configurando 16 columnas por 2 filas
  lcd.setCursor(0,0); //Ponemos el cursor en la primera fila a la izquierda
  lcd.print("Iniciando..."); //Imprimimos un mensaje inicial
  delay(2000); //Esperamos 2 segundos
  lcd.clear(); //Borramos lo que pone a la pantalla
}
void loop()
{
  //Primera fila
  lcd.setCursor(0, 0);
  lcd.print("LDH");
  delay(2000);
  //Segunda fila
  lcd.setCursor(0, 1);
  lcd.print("RICARDO");
  delay(2000);
  lcd.clear(); //Borramos lo que pone a la pantalla
}
```

Una vez que hemos conseguido visualizar el texto introducido. Vamos a imprimir información que envía el pc vía serie. Para ello haremos uso de un script en python que nos pida la información que queremos imprimir y la envíe a través del puerto serie.

Código en Arduino:

```
#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);
}
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);
    }
  }
}
```

```
}
```

Script python:

```
import serial
import time
s = serial.Serial('/dev/ttyACM0', 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)
```

Práctica 6. Display: Termómetro.

Objetivos.

Emplear un sensor de temperatura tmp36GZ para diseñar un termómetro que mostrará la temperatura en el display LCD.

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:

$$V_{\text{int}} = \frac{\text{Valor ADC} * 5}{1023} \text{ (V)}$$

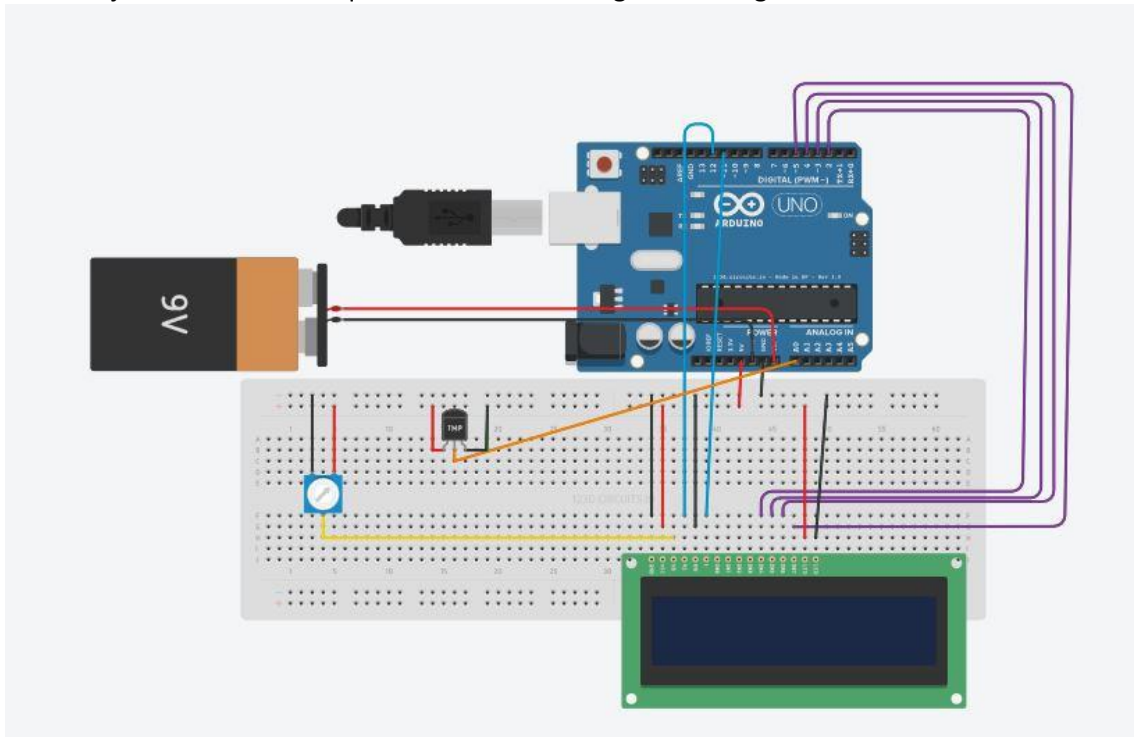
Para ello conectaremos el sensor de temperatura, que da una salida de tensión lineal a + 10.0mV/°C. Debemos convertir la tensión de entrada en temperatura, y lo haremos utilizando la ecuación:

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

Desarrollo.

Aprovechando todo el sistema montado en la práctica anterior, incorporamos el sensor de temperatura.

El montaje del circuito es el que se muestra en la siguiente imagen:



Código en 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: ");
  float temperature = (voltage - .5) * 100;
  Serial.println(temperature);
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.write("Temperatura: ");
  lcd.setCursor(0,1);
  lcd.print(temperature);
  lcd.setCursor(5,1);
  lcd.write((char)223);
  lcd.setCursor(6,1);
  lcd.write("C");
  delay(2000);
}
```

Problemas detectados o a evitar.

Perdí mucho tiempo en este apartado ya que el montaje era muy sencillo si disponía del montaje del circuito anterior, pero yo no me di cuenta y lo desmonté entero. Por lo que tuve que volverlo a montar y perder más tiempo del debido.

Práctica 7. Contador.

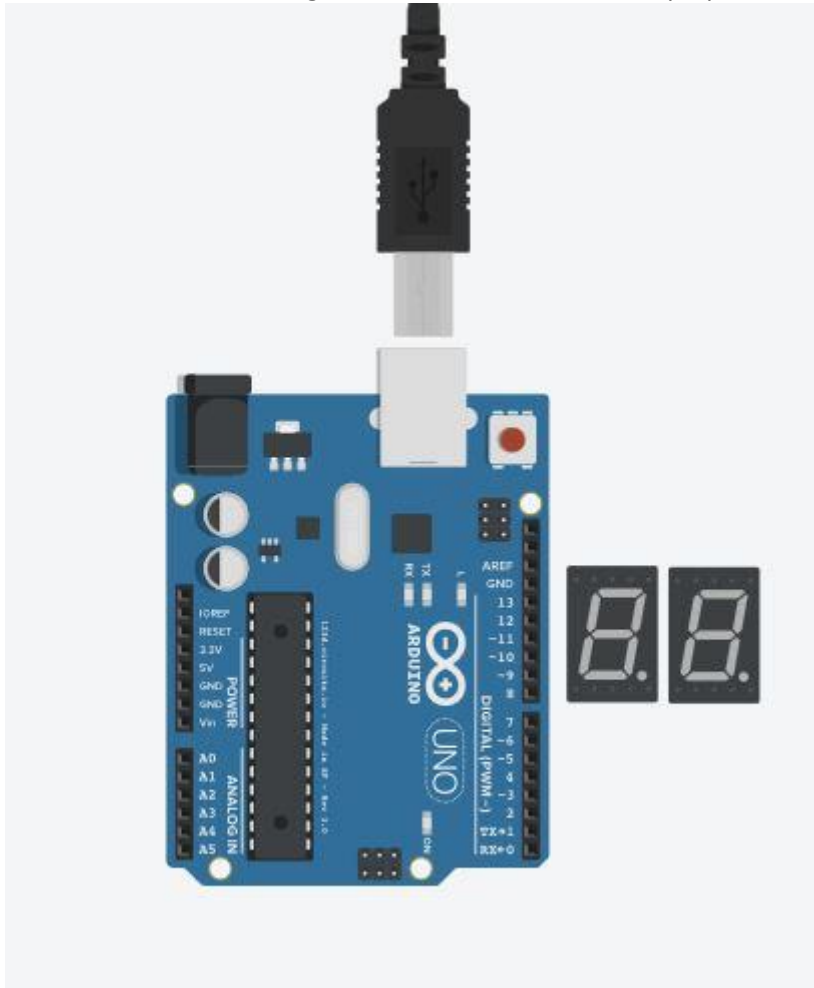
Objetivos.

Debemos ser capaces de realizar un contador que cuenta desde 00 a 59.

Introducción.

Para la realización de esta práctica, utilizaremos dos unidades de display de 7 segmentos, con cátodo común que ya están montados en una ampliación shield de Arduino y que solamente hay que conectar directamente al PCB de Arduino.

El visualizador de siete segmentos es una forma de representar números en equipos electrónicos. Está compuesto de siete segmentos que se pueden encender o apagar individualmente. Cada segmento tiene la forma de una pequeña línea.



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() {
```



```
// 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(dis1, OUTPUT);
pinMode(dis2, 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
  // -- aquí colocar código
  // Cálculo correcto de los datos dat1, dat0 --> desde 0 0 hasta 5 9
  int h;
  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 (dis1 con dat0 y dis2 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(dis1, 1);
    digitalWrite(dis2, 0);
    write_data(data1);
    delay(tiempo_refresco);
    digitalWrite(dis1, 0);
    digitalWrite(dis2, 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;
```

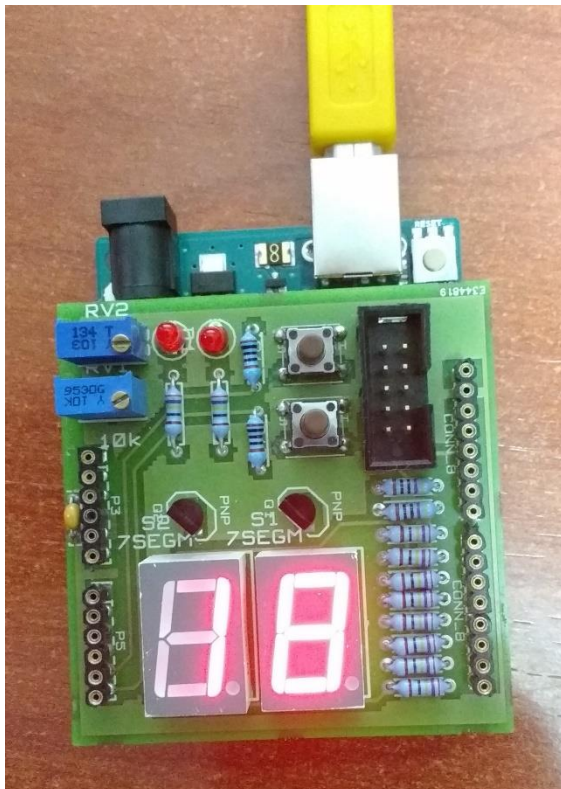
```
case 1:
//do something when var equals 2
write7seg(0x30);
break;
case 2:
//do something when var equals 1
write7seg(0x6d);
break;
case 3:
//do something when var equals 2
write7seg(0x79);
break;
case 4:
//do something when var equals 1
write7seg(0x33);
break;
case 5:
//do something when var equals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

Se comprueba que el contador funciona correctamente, contando desde 00 a 59.

Al final, el profesor pide que se cambie el contador para que inicialmente empiece a contar desde 50 y termine en 60, y después cuente normalmente desde 00 a 60. Eso se consigue sustituyendo el bloque de while (1) por el código siguiente:

```
for (int h=50; h < 60; h++){  
    dat1 = h / 10;  
    dat0 = h % 10;  
    refresh(dat1, dat0);  
}  
  
while(1){  
  
    for (int h=0; h < 60; h++){  
        dat1 = h / 10;  
        dat0 = h % 10;  
        refresh(dat1, dat0);  
    }  
}
```

Como última modificación el profesor pide que el refresco se note, eso se consigue modificando la línea 4 del código, donde antes ponía "int j = 40", escribir "int j = 4".



2. PLATAFORMA PAPILIO/ZPUINO

2.1. 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.

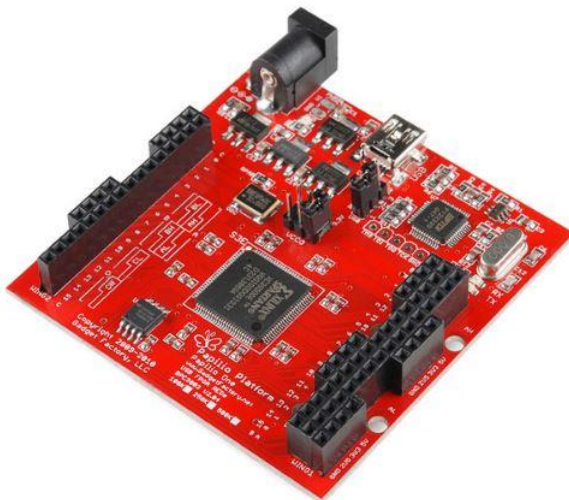
2.2. Introducción.

La placa PAPILIO abre de una manera sencilla las puertas al mundo del diseño FPGA. Es una placa open source, con multitud de recursos y una comunidad muy activa desarrollando proyectos muy interesantes. Papilio nació para emular maquinas arcade como PACMAN, SPACE INVADERS, etc, pero por la propia naturaleza de hardware reconfigurable de las FPGAs, las aplicaciones son infinitas. La misma placa puede comportarse como una placa Arduino con procesador AVR de 8 bits, o pasar a un procesador Soft Core ZPUINO de 32 bits corriendo a 96Mhz, simplemente cargando un nuevo archivo de configuración.

Las FPGA son HARDWARE RECONFIGURABLE, esto quiere decir, que la emulación de procesadores y circuitos no se hace por software, lo cual es lento, sino que al cargar un archivo de configuración a la FPGA le estamos indicando como se deben interconectar las compuertas y los bloques internos para armar físicamente el circuito que nos interesa. Así, la velocidad de funcionamiento es la del hardware, y permite diseñar placas de captura de datos de alta velocidad, procesamiento de protocolos, etc. Esta placa es ideal para el aprendizaje de diseño de circuitos digitales, VHDL, soft processors, y para el desarrollo de prototipos, soft modems, etc.

Durante las prácticas de laboratorio se usó la placa Papilio One Spartan3E 500.

ZPUino es un SoC implementado en VHDL (soft core) basado en el microprocesador ZPU de Zylins. Tanto las especificaciones del microprocesador como el diseño de ZPUino son abiertas.



2.3. Puesta en marcha del entorno de desarrollo.

El entorno de desarrollo para papilio/ZPUino es: Design Lab basado en el entorno de desarrollo de 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.

Para instalar el Design lab debemos de descargarlo desde:

<http://coria.dte.us.es/~bellido/>

Para instalarlo debemos seguir los pasos de la siguiente guía de instalación

<http://gadgetfactory.net/learn/2015/01/13/designlab-installation-guide-linux/>, pero debemos tener especial cuidado y hacer algunos cambios con respecto a la guía de instalación, los cambios son los siguientes:

1. En la versión actual el script “ftdi_user.sh” ha sido cambiado por “**ubuntu-setup.sh**”
2. Será necesario instalar el JRE
3. La placa que debemos seleccionar para verificar que funciona el sistema es aquella que tengamos.
4. Comprobar que en archivo --> preferencias, la “Linux Ise Location” es:
/opt/Xilinx/14.7/ISE_DS/ISE/bin/linux64/

2.4. Desarrollo de las prácticas.

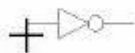
2.4.1. Inversor.

Para realizar esta práctica vamos a desarrollar el tutorial

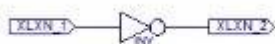
<http://gadgetfactory.net/learn/2015/05/03/designlab-make-a-simple-fpga-circuit-2/> donde se va a mostrar como realizar un nuevo proyecto usando un circuito FPGA. Se creará un circuito simple que consiste en un inversor y se conectará la entrada a un botón y la salida a un LED. Cuando presionamos el botón, el LED mostrará la salida invertida.

Empezamos pulsando “New FPGA Circuit Project” del menú File. Nos aparecerá un nuevo proyecto vacío, lo guardamos para poder editarlo y pulsamos el botón de edición, se nos abre el XILINX, buscamos nuestra placa y hacemos click sobre ella.

A continuación, abrimos el editor de esquemáticos para crear un circuito simple, para ello buscamos el símbolo “inv”, que tiene el siguiente aspecto:



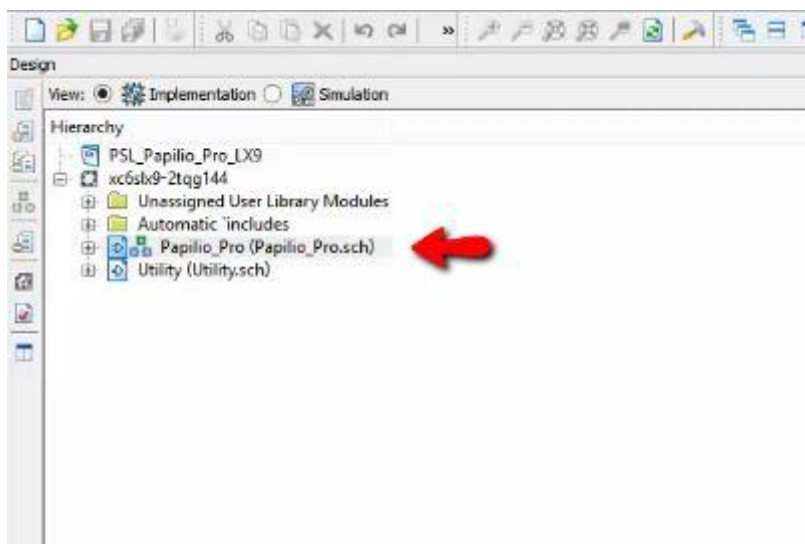
Después le conectamos dos conectores I/O, como se ilustra:



Abrimos el diagrama de conexiones que nos mostrara los pines disponibles. Los marcadores de E/S, sirven para informarle al software, qué pines hay que conectar a la FPGA. Para ver que contactos están disponibles en la placa Papilio, clicar en la pestaña de DISEÑO-> UTILIDAD, y aparecerán los terminales que deberemos utilizar WING_ALO Y WING_ALO1.



Volvemos al esquemático y editamos las conexiones para asignarle los nombres correspondientes y sintetizamos el circuito.



(Nota: la anterior imagen fue sacada del tutorial y no corresponde con el modelo usado en laboratorio que es Papilio One 500)

Por último es muy importante para cargar correctamente el bitfile hay que hacer la siguiente modificación en el nombre del fichero que genera ISE:

En la carpeta del proyecto /circuit/500K/

Debemos borrar `papilio_one_500k.bit` y renombrar `Papilio_One_500K.bit` a `papilio_one_500k.bit`

Por último, se carga el modelo en la FPGA y se monta el circuito para usar un switch y un led.

Problemas detectados.

No me funcionaba el sistema y no encontraba el fallo, estuve un rato revisando. Lo que ocurrió fue que en los últimos pasos de renombrado de ficheros al poner en minúsculas el archivo `papilio_one_500k.bit`, la `k` estaba en mayúscula y por eso no funcionaba. Lo corregí y todo al final fue correcto.

2.4.2. Pc-led. Cargar SoC ZPUINO y desarrollar SKETCHES.

Desarrollaremos el tutorial "Open QuickStart Sketch":

<http://gadgetfactory.net/learn/2015/04/03/designlab-using-the-ide-for-the-first-t>

El sketch que debemos cargar esta en la sección ejemplos-- > Papilo_QuickStart

1. Se debe seleccionar el tipo de placa Papilio One 500.
2. Se selecciona el puerto y cargamos el circuito en el icono Load Circuit.
3. Para finalizar se carga el esquemático.

Se ha revisado el código desde el tutorial y se ha modificado para que en vez de parpadear el led quede apagado y se añade a la línea del monitor serie la información correcta:

➔ LED: Encendido

➔ LED: Apagado

El código que se ha usado es el siguiente:

```
HardwareSerial mySerial1(WishboneSlot(5));
int led = 13;

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);
    }
  }
}
```

Código Python:

```
import serial

Zpuino = serial.Serial('/dev/ttyUSB0', 9600);

print ("Starting o Inicializando")

while 1 :
  comando = raw_input('Introduce un comando H o L: ')
```



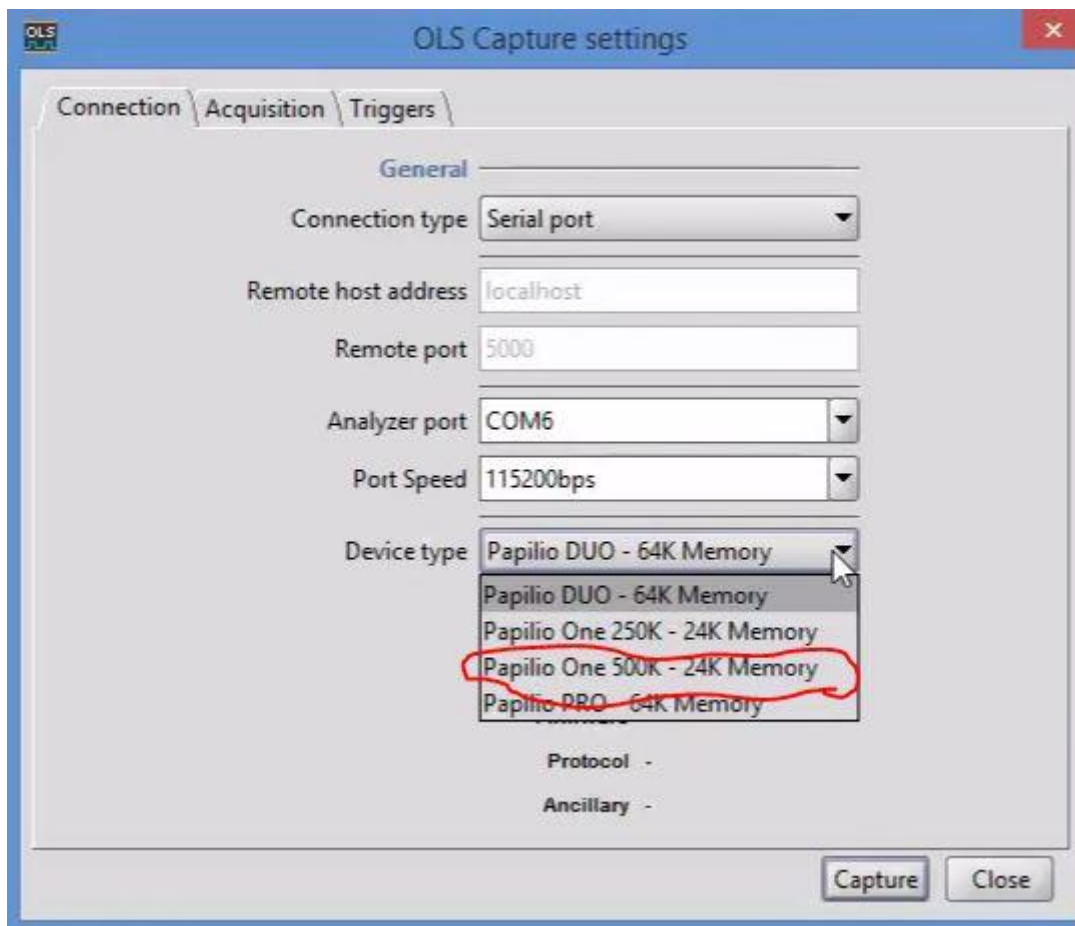
```
Zpuino.write (comando)
if comando == 'H':
    print('LED ENCENDIDO')
elif comando == 'L':
    print('LED APAGADO')
Zpuino.close()
```

2.4.3. Analizador lógico.

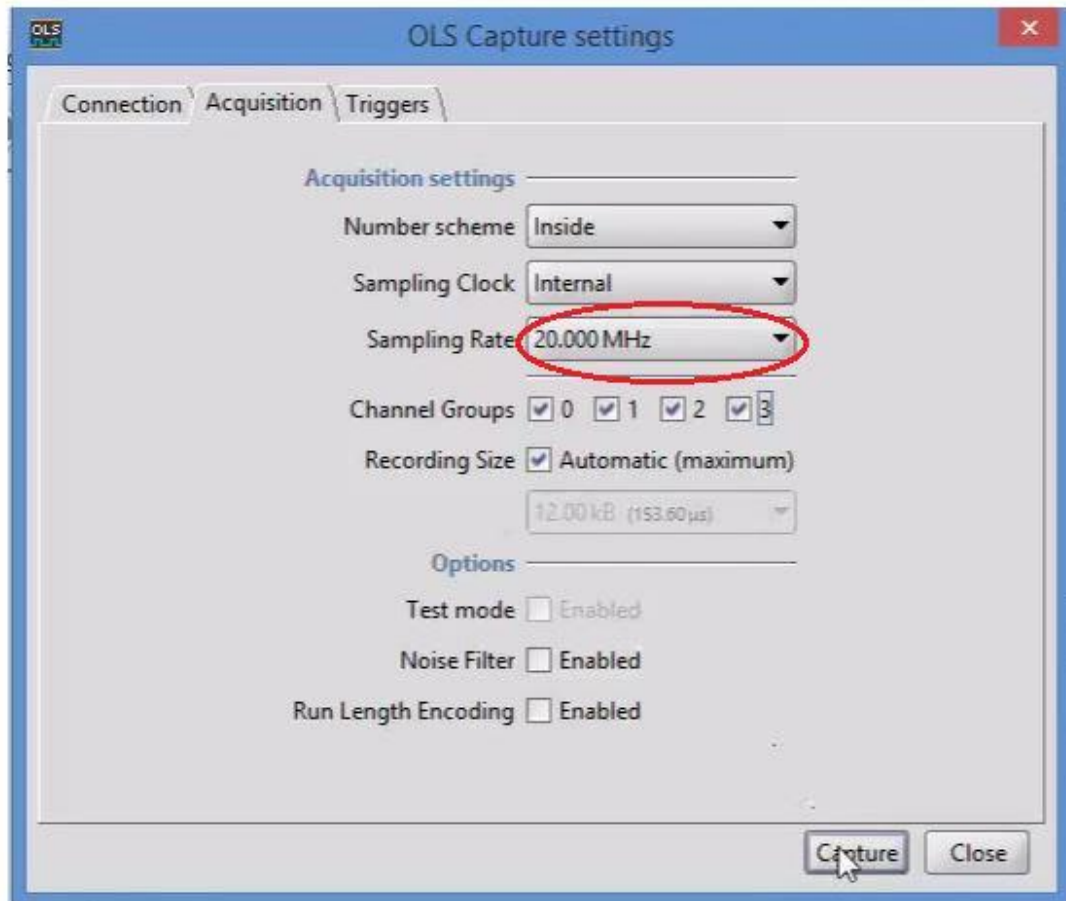
Abrimos el analizador lógico de Design Labs.



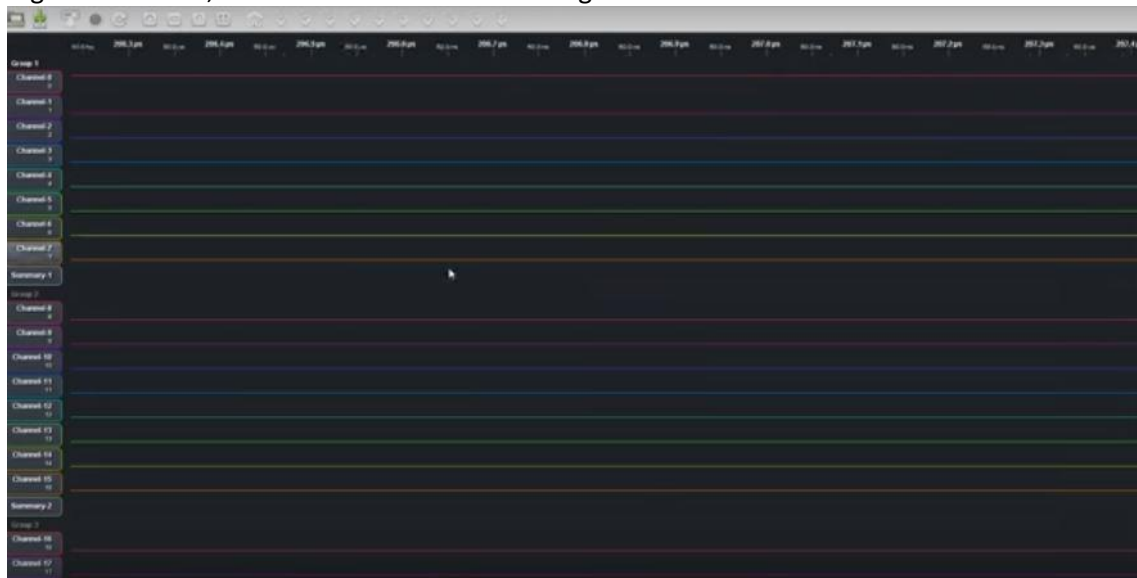
Al pulsar “Iniciar Captura de datos” podremos modificar la configuración. En la pestaña de Connection ajustamos los siguientes valores:



En la pestaña Acquisition, ajustamos los siguientes valores:



Finalmente, por el canal 0 debemos ver la señal. Si metemos 5v en el canal 0, tendremos un 1 lógico en el visor, tal como se muestra en la imagen.



Una vez finalizado todo lo anterior, el profesor me pide que le meta una señal PWM desde una placa Arduino. Una vez montado el sistema y metida la señal en el canal 0, le damos a capturar nuevamente pero no se visualiza la señal PWM. Esto es debido a la diferencia de frecuencia en la que trabaja y como anteriormente teníamos 200.000 MHz, no se consigue visualizar. Finalmente cambiamos la frecuencia y vemos correctamente la señal PWM.

2.4.4. Rediseñando el SoC. Añadiendo periféricos.

Se va a desarrollar un 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. Se va a seguir el tutorial siguiente: <http://gadgetfactory.net/learn/2015/05/15/designlab-make-a-custom-zpuino-system-on-chip-2/> titulado "Creating a new ZPUINO SOC Project"

Una vez que se tiene la UART disponible en los Pines WA0 y WA1 se debe conectar al PC con el cable TTL-RS232-USB.

La imagen del cable es la siguiente:



A la hora de la conexión debemos asegurarnos que el cable negro va a tierra, el orange a RX y el amarillo a TX. En la siguiente imagen se puede ver el montaje final:



Para controlar el encendido/apagado de un LED desde el PC a través de la nueva UART, debemos tener el código del script de Python usado en las prácticas de Arduino que se usaban para el mismo propósito.

3. RASPBERRY PI

3.1. 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 del modelo de Raspberry Pi 3 Modelo B ensambla en su circuito un chipset Broadcom BCM2387 con cuatro núcleos ARM Cortex-A53 a 1.2 GHz. Dicho procesador es capaz de mover con soltura videojuegos y aplicaciones, además de disponer de una gran potencia de procesamiento para otros tipos de tareas. 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.

3.2. 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.

3.3. Puesta en marcha del entorno de desarrollo.

La preparación consiste en instalar un sistema operativo en una tarjeta SD que posteriormente conectaremos a la Raspberry Pi, que nos permitirá arrancarlo desde esta.

Para ello seguiremos los siguientes pasos:

1. Formatear la tarjeta SD.
2. Instalar la imagen del S.O. Ubuntu Mate. Se debe descargar desde <http://coria.dte.us.es/~bellido/> (En el laboratorio el profesor dio las indicaciones necesarias para instalar la imagen desde un equipo con Ubuntu, pero yo lo hice desde Windows ya que lo había hecho más veces y me pareció más sencillo con el uso de Win32DiskImager).
3. Realizar el conexionado de la placa de desarrollo siguiendo los pasos:
 - a. Conectar la SD a RaspberryPi (RPI)
 - b. Desconectar teclado, ratón y cable Ethernet del PC y conectarlo a RPI.
 - c. Conectar cable HDMI-DVI a RPI y monitor.
 - d. Conectar cable USB-Micro usb a RPI (microusb) y PC para alimentar la RPI

Después procederemos a configurar Ubuntu-mate para ello se deben seguir los pasos en el arranque configurando idioma, teclado, localización, fecha y hora.

Es importante guardar el usuario y contraseñas elegidas.

En el primer arranque de Ubuntu Mate es cuando se realiza una configuración del sistema y se completa la instalación del sistema de ficheros.

Algo que se debe hacer es ajustar el tamaño del sistema de ficheros al tamaño de la imagen tal y como se cuenta en la página web de información de Ubuntu Mate para Raspberry Pi con "resize".

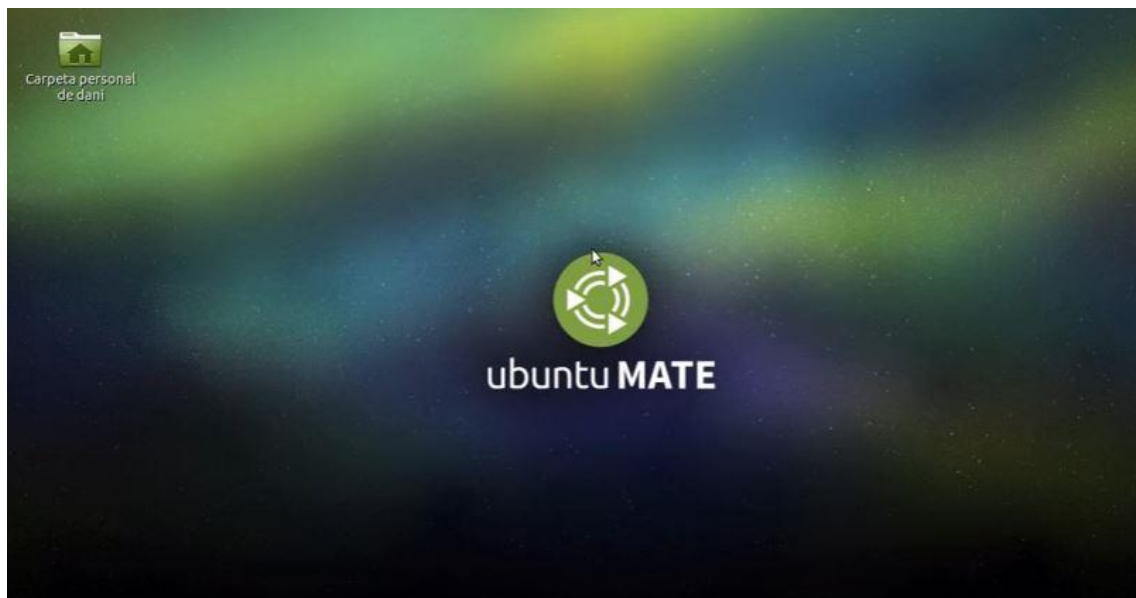
Se ha procedido a configurar la red manualmente. Para ello debemos hacer lo siguiente:

1. Desactivar network-manager.
2. Editar /etc/network/interfaces. En mi caso particular lo que muestra el archivo interfaces es lo siguiente:

```
auto lo
iface lo inet loopback
auto enxb827eb393174
iface enxb827eb393174 inet static
address 10.1.15.94
netmask 255.255.252.0
gateway 10.1.15.78
dns-nameservers 8.8.8.8
```

Donde “enxb827eb393174” es el nombre de la interfaz de mi red , y “10.1.15.94” es la IP de la red local del pc del laboratorio.

3. Se ejecuta sucesivamente `sudo ifdown <nombre_interfazDeRed>`, `$ sudo ifup <nombre_interfazDeRed>`
4. Se prueba que efectivamente hay conexión haciendo ping a www.google.es.



3.4. Desarrollo de las prácticas.

3.4.1. Gpio-Python.

En esta parte debemos realizar el encendido y apagado de un led a través del terminal. La conexión del LED, se realizará en el puerto 21.

Desde RPI, abrimos el terminal y ejecutamos:

- Configurar el puerto:

```
echo 21 > /sys/class/gpio/export
```

El sistema habrá creado un archivo con una estructura GPIO que corresponde al número 21.

- Para que el pin va a ser de salida, ejecutaremos lo siguiente:

```
echo out > /sys/class/gpio/gpio21/direction
```

Una vez realizado lo anterior, para encender o apagar el LED escribiremos, lo siguiente:

- Para encender el LED:

```
echo 1 > /sys/class/gpio/gpio21/value
```

- Para apagar el LED:

```
echo 0 > /sys/class/gpio/gpio21/value
```

- Para liberar el puerto una vez que hemos terminado:

```
echo 21 > /sys/class/gpio/unexport
```

Ahora, realizaremos el encendido y apagado de un LED, desde un script en Python.

El montaje será el mismo, lo único que cambiará es que se automatizará el ON-OFF del led (Blink), de manera que quedará parpadeando.

Primero hay que instalar la librería para poder controlar los GPIO con Python. Está alojada en SourceForge, pero podemos descargarla en la Raspberry Pi con el siguiente comando:

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

Una vez descargada, descomprimos el fichero:

```
tar zxvf RPi.GPIO-0.5.4.tar.gz
```

Accedemos en el directorio que acabamos de descomprimir:

```
cd RPi.GPIO-0.5.4/
```

Instalamos la librería. Si no tenemos instalado python, introducimos el siguiente comando:

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

Cuando acabe la instalación del paquete anterior, procedemos a instalar la librería:

```
sudo python setup.py install
```

Y creamos el script en Python:

```
sudo nano blink.py
```

Script Python:

```
## Programa Blink.py -> Pone intermitente un LED en el puerto 21 de una RaspBerry Pi
##Importamos la librería
import RPi.GPIO as GPIO
import time
##Programamos los puertos
GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT) ## GPIO 21 como salida
##Definimos la funcion blink
def blink():
    print "Ejecucion iniciada..."
    iteracion = 0
    while iteracion < 30: ## Segundos que durara la funcion
        GPIO.output(21, True) ## Enciendo el 21
        time.sleep(1) ## Esperamos 1 segundo
        GPIO.output(21, False) ## Apago el 21
        time.sleep(1) ## Esperamos 1 segundo
        iteracion = iteracion + 1
    print "Ejecucion finalizada"
    GPIO.cleanup() ## Hago una limpieza de los GPIO
```

3.4.2. Servidor web GPIO.

3.4.2.1. Objetivos.

En esta práctica se va a montar un “web framework” para Python. Es decir, seremos capaces de ejecutar código Python desde el servidor web.

3.4.2.2. Desarrollo.

A partir de ahora, nos conectaremos a la consola de la Raspberry Pi por red, vía SSH. En el equipo desde el que se quiera acceder a la RPI debemos tener instalado ssh y ejecutar lo siguiente:

```
$ ssh -X usuario@IP_de_usuario.
```

Donde usuario es el nombre del usuario de la RPI y IP_de_usuario es la IP de la RPI.

Se va a desarrollar el siguiente tutorial: • <http://mattrichardson.com/Raspberry-Pi-Flask/index.html>

Para instalar el servidor web, ejecutaremos en el terminal lo siguiente:

- Instalar Python-pip:

```
$sudo apt-get install python-pip
```
- Instalar las dependencias:

```
$sudo pip install flask
```

Se creará un fichero llamado hello-flsk.py con el siguiente código:

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
    return "Hola mundo!"
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Para probarlo, ejecutamos en el terminal el comando:

```
$ sudo Python hello-flask.py
```

Una vez arrancado creamos el fichero hello-template.py con 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)
```

Como se observa en el script, se va a necesitar una interfaz intermedia entre el script y el usuario, para ello nos creamos una web simple en un fichero llamado main.html con el siguiente código:

```
/-- 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>
```

Finalmente ejecutamos nuestro script desde la consola de comandos, ejecutando lo siguiente y estando en el directorio donde se encuentra el script:

```
$ sudo python hello-template.py
```

Ahora si abrimos el navegador y tecleamos la dirección IP de la RPI, tendremos la página con el título “HOLA” y la fecha y hora actual de la RPI, esto podremos hacerlo o bien desde la propia Raspberry Pi en local o desde cualquier otro PC que tenga acceso a la Raspberry Pi tecleando la IP de esta.

Ya podemos acceder a controlar el LED desde un navegador web. Para ello debemos montar la página web, que lo haremos en un nuevo directorio, llamado weblamp, una vez situado en el directorio personal, ejecutar el siguiente comando:

```
$ sudo mkdir weblamp
```

Y en él crearemos los siguientes ficheros:

Fichero 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)
```

Fichero main.html:

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

<body>
  <h1>Lista de dispositivos y estado</h1>

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

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

</body>
</html>
```

Finalmente tenemos la web y comprobamos que al hacer click en encender o apagar los leds responden adecuadamente.

Imagen de la página web:

Lista de dispositivos:

El LED1 esta encendido ([Apagar](#))

El LED2 esta apagado ([Encender](#))

3.4.3. Servidor web Temperatura.

3.4.3.1. Objetivos.

Se debe ser capaz de leer la temperatura desde el servidor.

3.4.3.2. Desarrollo.

Para desarrollar esta práctica se deben realizar varias fases:

1. Desde Arduino leer la temperatura y enviarla al puerto serie.
2. Desde RPI leer la temperatura desde el puerto serie para poder mostrarla en el servidor web.

Debemos montar parte del esquema ya visto en la práctica 6 de arduino pero en vez de mostrarla en el display, debemos enviarla por el puerto serie. Para ello, montamos el esquema y cargamos en Arduino en siguiente código:

```
const int pinAnalogico =A0;

void setup () {
  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(2000);
}
```

Con esto ya estamos enviando la temperatura, se puede comprobar mirando el puerto serie desde el mismo Arduino para ver que todo está correcto hasta este punto.

Desde la RPI debemos leer la temperatura para mostrarla en el servidor web. Teniendo como base los archivos python y html, los modificamos para que se incluya la lectura de la temperatura desde el puerto serie y su visualización en la página web.

[Fichero weblampTemperatura.py](#)

```
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("/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)

```

Fichero main.html

```

<!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 %}
    esta encendido (<a href="{{pin}}/off">Apagar</a>)
  {% else %}
    esta apagado (<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>  
  
</body>  
</html>
```

Imagen de la página web:

Lista de dispositivos:

El LED1 esta encendido ([Apagar](#))

El LED2 esta apagado ([Encender](#))

Lectura de temperatura:

Temperatura: 19.34

Actualizar temperatura: ([Actualizar](#))

3.4.3.3. Problemas encontrados.

La mayor dificultad de esta práctica ha sido entender bien lo que hacía el código html y como se le pasaban los datos desde el archivo Python. Una vez entendido, se pueden añadir variables y mostrarlas sin dificultad.

3.4.4. Servidor web luz.

3.4.4.1. Objetivos.

Desde el servidor web se debe ser capaz de encender o apagar la luz (manteniendo la lectura de la temperatura).

3.4.4.2. Desarrollo.

Siguiendo el esquema del circuito del apartado anterior, se sigue con esta práctica. Lo único que hay que añadir es a la placa Arduino el relé con bombilla. También hay que modificar el Arduino y añadirle el código para que a la vez que envía la temperatura, lea del puerto serie cada vez que le llegue una orden de encender o apagar la bombilla con el relé.

Arduino:

```
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);
}
```

[sistemaCompleto.py](#)

```
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)

```

main.html

```

<!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>Encender(<a href="/off">ON</a>)</p>

<p>Apagar(<a href="/on">OFF</a>)</p>

</body>
</html>
```

Imagen de la página web, con todo el sistema final:

Lista de dispositivos:

El LED1 esta encendido ([Apagar](#))

El LED2 esta apagado ([Encender](#))

Lectura de temperatura:

Temperatura: 19.34

Actualizar temperatura: ([Actualizar](#))

Bombilla:

Quiero encenderla ([Encender](#))

Quiero apagarla ([Apagar](#))

3.4.4.3. Problemas encontrados.

La mayor dificultad ha estado en editar el archivo pyhon para que mantuviese la temperatura a la vez que se pulsaban las acciones de encender y apagar la bombilla del relé. Pero finalmente, se consiguió.

Tuve un pequeño problema a la hora de plantear la práctica y empezar a montar el sistema, tenía dudas acerca de como se podían mandar y recibir datos a la vez desde el puerto serie, pero el profesor me aclaró las dudas. Así que finalmente pude realizar la práctica satisfactoriamente.