

# LDH - Memoria de prácticas

José Antonio Torrecilla Fuentes

<b>Plataforma Arduino</b>	<b>3</b>
Primeros pasos:	3
Encendido/apagado de una bombilla mediante un relé:	3
Encendido/apagado de un LED mediante pulsador empleando interrupciones:	4
Control de la posición de un servomotor a través del puerto serie:	5
Control de velocidad de giro de un motor de continua con un potenciómetro:	6
Control de velocidad de giro de un motor de continua en doble sentido con un potenciómetro:	6
Impresión en pantalla LCD de lectura del puerto serie:	6
Impresión en pantalla LCD de temperatura tomada desde sensor tmp36GZ:	7
Cronómetro en display 7 segmentos:	8
<b>Plataforma ZPUino:</b>	<b>8</b>
Diseñando circuitos básicos sobre FPGA:	8
Cargar SoC ZPUino y desarrollar sketches:	8
Convirtiendo la placa Papilio en un analizador lógico:	8
Rediseñando el SOC. Añadiendo periféricos:	9
<b>Plataforma Raspberry PI:</b>	<b>11</b>
Instalación de Ubuntu Mate:	11
Tarea 1:	12
Tarea 2:	12
Tarea 4:	13
Tarea 5:	15
Tarea 6:	18

# Plataforma Arduino

## Primeros pasos:

El objetivo ha sido recorrer brevemente algunos de los códigos más básicos que se pueden ejecutar en la plataforma Arduino:

1. Conmutar un LED integrado en la placa. El entorno de desarrollo de Arduino incluye una serie de códigos de ejemplo, entre los cuales, está incluido el código que necesitamos, conocido como "Blink".
2. Subir y bajar la intensidad luminosa de un LED. El código de este programa también está incluido en los ejemplos del entorno de Arduino, con el nombre "Fade".
3. Conmutar un LED con un pulsador. El código de este programa también está incluido en los ejemplos del entorno de Arduino, con el nombre "Button".
4. Controlar el encendido/apagado de un LED desde PC vía puerto serie. Este programa tiene dos partes claramente diferenciadas: Un programa para el PC que envíe datos por el puerto serie (USB) escrito en python y un programa para que el Arduino reciba esos datos y conmute el LED en consecuencia. El código de estos programas podemos encontrarlos en el siguiente link:

<https://geekytheory.com/arduino-raspberry-pi-raspduino/>

## Encendido/apagado de una bombilla mediante un relé:

El objetivo es controlar el encendido/apagado de una bombilla AC220V a través de un relé y el puerto serie (USB). Para escribir el código, reciclaremos los códigos anteriores; de hecho, lo único que cambiaremos serán las indicaciones en el caso del programa en python y usar la función "parseInt()" en lugar de "read()" en el código del Arduino.

```
// ej1.ino

int luz = 13;
void setup () {
  pinMode(luz, OUTPUT); // luz 13 como salida
  Serial.begin(9600); // Inicializo el puerto serial a 9600 baudios
}

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

```
#ej1.py

import serial
arduino = serial.Serial('/dev/ttyACM0', 9600)
print("Starting!")
while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino
    if comando == 'H':
        print('LUZ ENCENDIDA')
    elif comando == 'L':
        print('LUZ APAGADA')

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

En cuanto al esquemático, sencillamente, conectamos el relé a tierra en su pin GND, a 5V en su pin VCC y al puerto 13 de la placa Arduino en su pin de control. Por otro lado, conectamos en serie el cable de la bombilla a los bornes del relé, usando siempre el central.

## Encendido/apagado de un LED mediante pulsador empleando interrupciones:

El objetivo es controlar un diodo LED a través de un pulsador que active una interrupción hardware. Para ello, declararemos una variable tipo “volátil” que controle el estado del diodo LED y configuraremos el pin 2 como línea de interrupción mediante la función “attachInterrupt(...)”.

```
const byte led = 13;
const byte interruptPin = 2;
volatile byte estado = LOW;

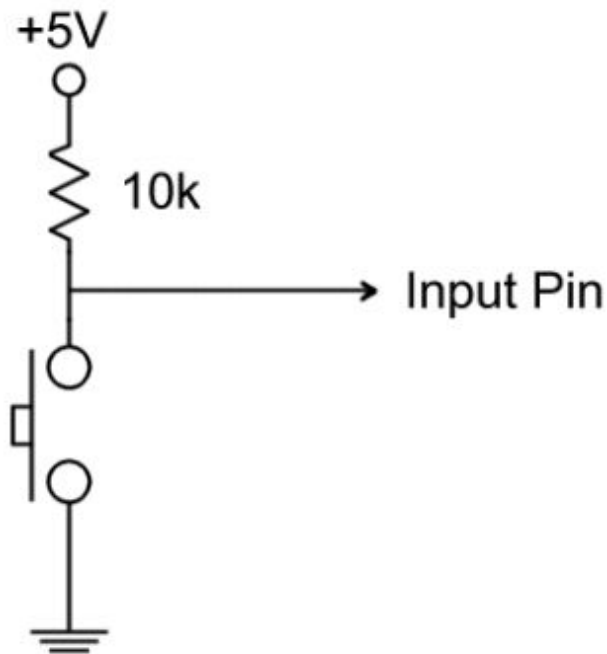
void setup() {
    pinMode(led, OUTPUT);
    pinMode(interruptPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

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

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

En cuanto al esquemático, el led se conectará al pin 13 y a tierra a través de una resistencia, por ejemplo, de 10kΩ. Por otro lado, el pulsador se conectará a la señal de 5V a

través de una resistencia junto con el pin 2 (configuración de pull-up) y a tierra por el otro extremo.



Configuración de Pull-up

## Control de la posición de un servomotor a través del puerto serie:

El objetivo es controlar la posición de un servomotor, pasándole a través del puerto serie como parámetro el ángulo (entre 0 y 180). Para ello usaremos la librería “Servo.h” y configuramos el pin de control del servomotor con la función “attach(pin)”.

```
#include <Servo.h>

Servo myservo;
int pos = 0;
void setup() {
  myservo.attach(9);
  Serial.begin(9600);
}

void loop() {
  if(Serial.available())
    pos = Serial.parseInt();
  if(pos > 0 && pos <= 180)
    myservo.write(pos);
  delay(15);
}
```

En cuanto al esquemático, conectamos el servomotor a 5V en su pin VCC, a tierra en su pin GND y al puerto 9 en su pin de control.

## Control de velocidad de giro de un motor de continua con un potenciómetro:

El objetivo es controlar un motor de corriente continua de 9V con un potenciómetro. Para ello vamos a usar un transistor de potencia IRF520 que controlaremos a través de una señal PWM.

No conservo el código.

## Control de velocidad de giro de un motor de continua en doble sentido con un potenciómetro:

El objetivo es controlar un motor de corriente continua de 9V con un potenciómetro en doble sentido, de manera que dividiremos la señal analógica generada por el potenciómetro en dos partes, una para el giro negativo del motor y otra para el giro positivo.

No conservo el código.

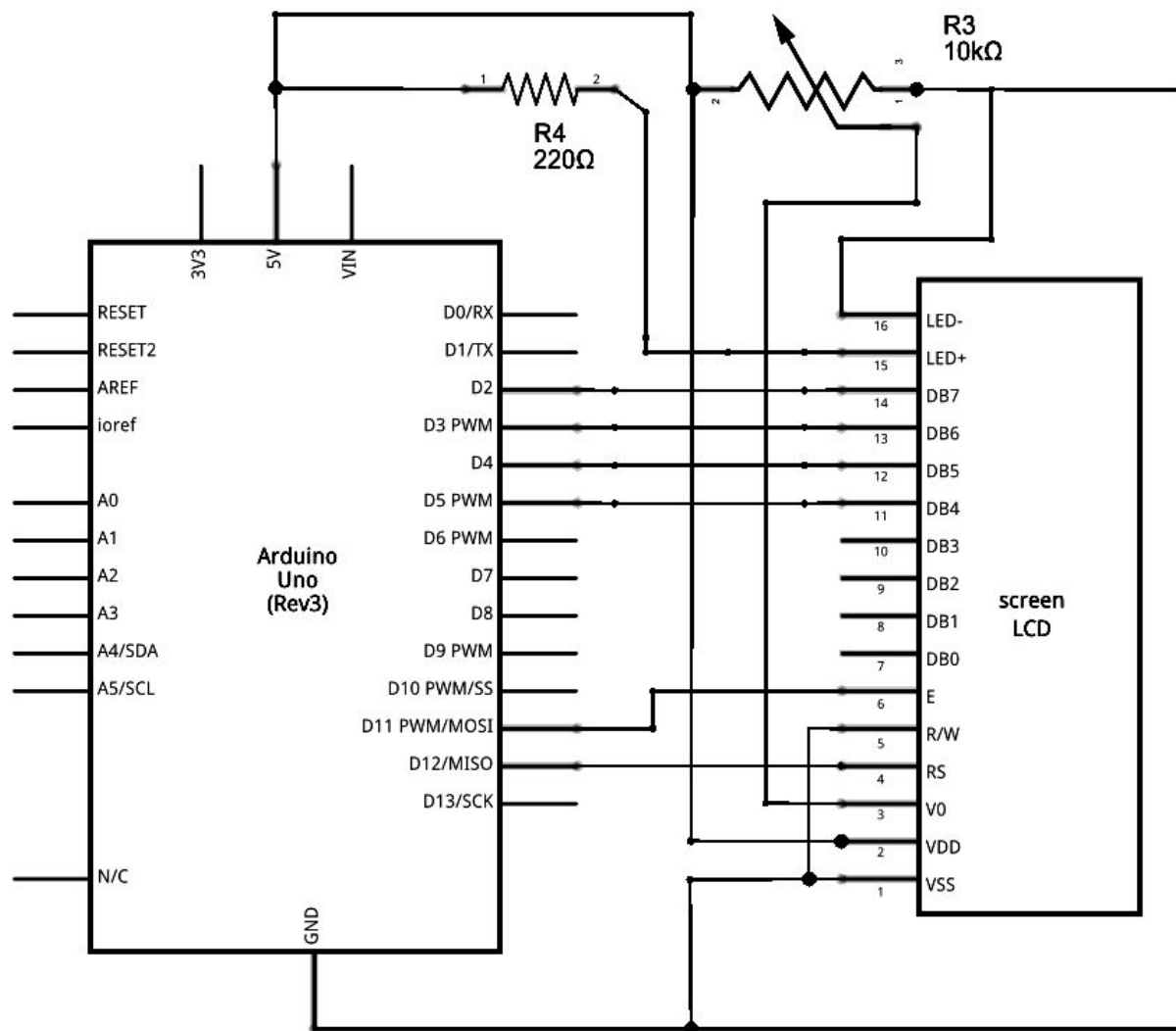
## Impresión en pantalla LCD de lectura del puerto serie:

El objetivo es visualizar en una pantalla LCD de 16x2, las cadenas enviadas desde el PC a través del puerto serie USB. Para ello usaremos la librería "LiquidCrystal.h", inicializamos la pantalla con los pines de conexión y la configuramos con la función "begin(*columns*, *lines*)".

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
char c;
void setup() {
  lcd.begin(16, 2);
}

void loop() {
  if(Serial.available())
    c = Serial.parseInt();
  lcd.setCursor(0, 1);
  lcd.clear();
  lcd.print(c);
}
```



Impresión en pantalla LCD de temperatura tomada desde sensor tmp36GZ:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int sensorPin = 0;
void setup() {
  lcd.begin(16, 2);
  lcd.print("Temperatura:");
}

void loop() {
  lcd.setCursor(0, 1);
  int reading = analogRead(sensorPin);
  float voltage = reading * 5.0;
  voltage /= 1024.0;
  float temperatureC = (voltage - 0.5) * 100;
  lcd.print(temperatureC);
}
```

```
lcd.print("");  
delay(1000);  
}
```

## Cronómetro en display 7 segmentos:

El objetivo es crear un cronómetro mediante la función `millis()` y visualizarlo en dos displays 7 segmentos incluidos en un shield para Arduino.

No conservo el código.

## Plataforma ZPUino:

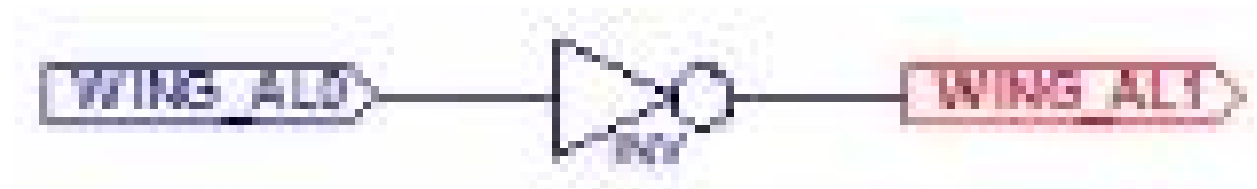
Vamos a trabajar sobre una nueva plataforma, una placa de desarrollo hardware abierta que monta un FPGA de Xilinx; en concreto vamos a usar la Papilio One 500K que monta una Spartan-3E. Esta placa está diseñada para funcionar como una placa Arduino, ya que respeta las dimensiones del pinout de esta placa y es posible cargar cualquier SOC “System On Chip”, por ejemplo el ZPUino. Para trabajar sobre esta placa usaremos el entorno de desarrollo para Papilio, Design Lab, cuya instalación es rápida y sencilla.

## Diseñando circuitos básicos sobre FPGA:

Vamos a ver como diseñar un circuito electrónico digital básico como es un inversor, lo vamos a cargar en nuestra placa y vamos a comprobar su funcionamiento conectando un diodo LED.

Para ello abriremos Design Lab, pincharemos en “File > New FPGA Circuit Project”, crearemos un nuevo Sketch guardándolo con el nombre que queramos y pincharemos en el botón “Edit Circuit” que nos llevará al entorno ISE de Xilinx en el que podremos diseñar de forma gráfica el esquemático del circuito digital que queremos crear.

Pinchamos sobre el archivo “.sch” para acceder al editor esquemático, pinchamos en el botón “Symbols” y luego en “All Symbols”, buscaremos el inversor y lo colocamos. A continuación debemos configurar los pines clicando en el botón “Connector” y conectaremos dos marcadores de entrada salida. Por último los renombraremos: A la entrada la llamaremos “WING\_AL0” y a la salida “WING\_AL1” de manera que quedará de la siguiente forma:



Ya solo falta sintetizar el circuito, seleccionamos el archivo “.sch” y pinchamos sobre “Generate Proramming File”. Cuando se genere el fichero, volvemos al Design Lab y cargamos el circuito pinchando en el botón “Load Circuit”. Comprobamos que cumple su función con un par de diodos LED y un switch.



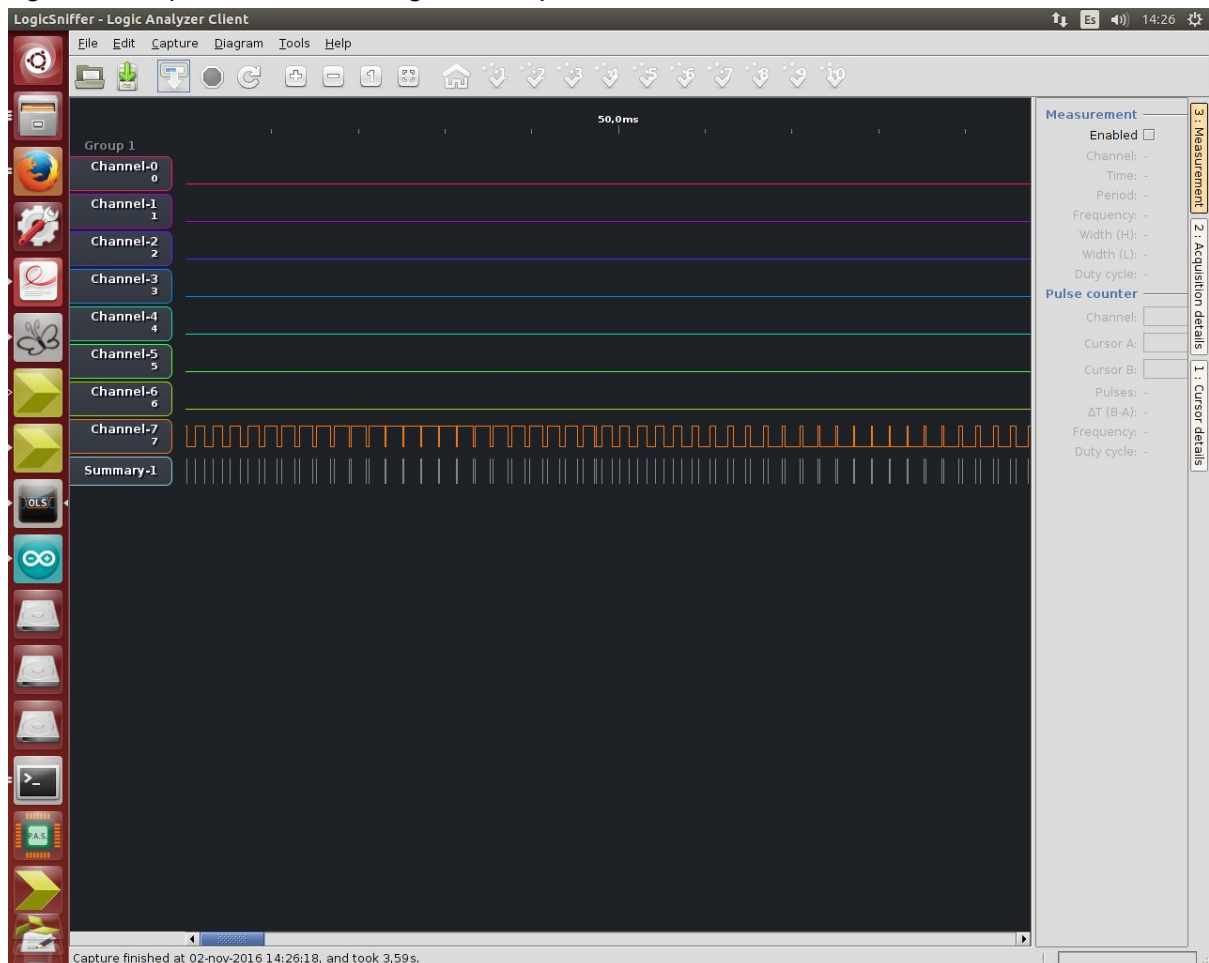
## Cargar SoC ZPUino y desarrollar sketches:

Vamos a ver como cargar el SoC ZPUino para utilizar nuestra Papilio One como una placa Arduino. Abrimos Design Lab y abrimos el Sketch “QuickStart”, seleccionamos el modelo de nuestra placa y el puerto al que está conectada y cargamos pinchando en el botón “Load Circuit”. Ahora nuestra Papilio One es, a efectos prácticos, una placa Arduino. Por último vamos crear un sencillo sketch de Arduino para comprobar el funcionamiento de ZPUino.

No conservo el código.

## Convirtiendo la placa Papilio en un analizador lógico:

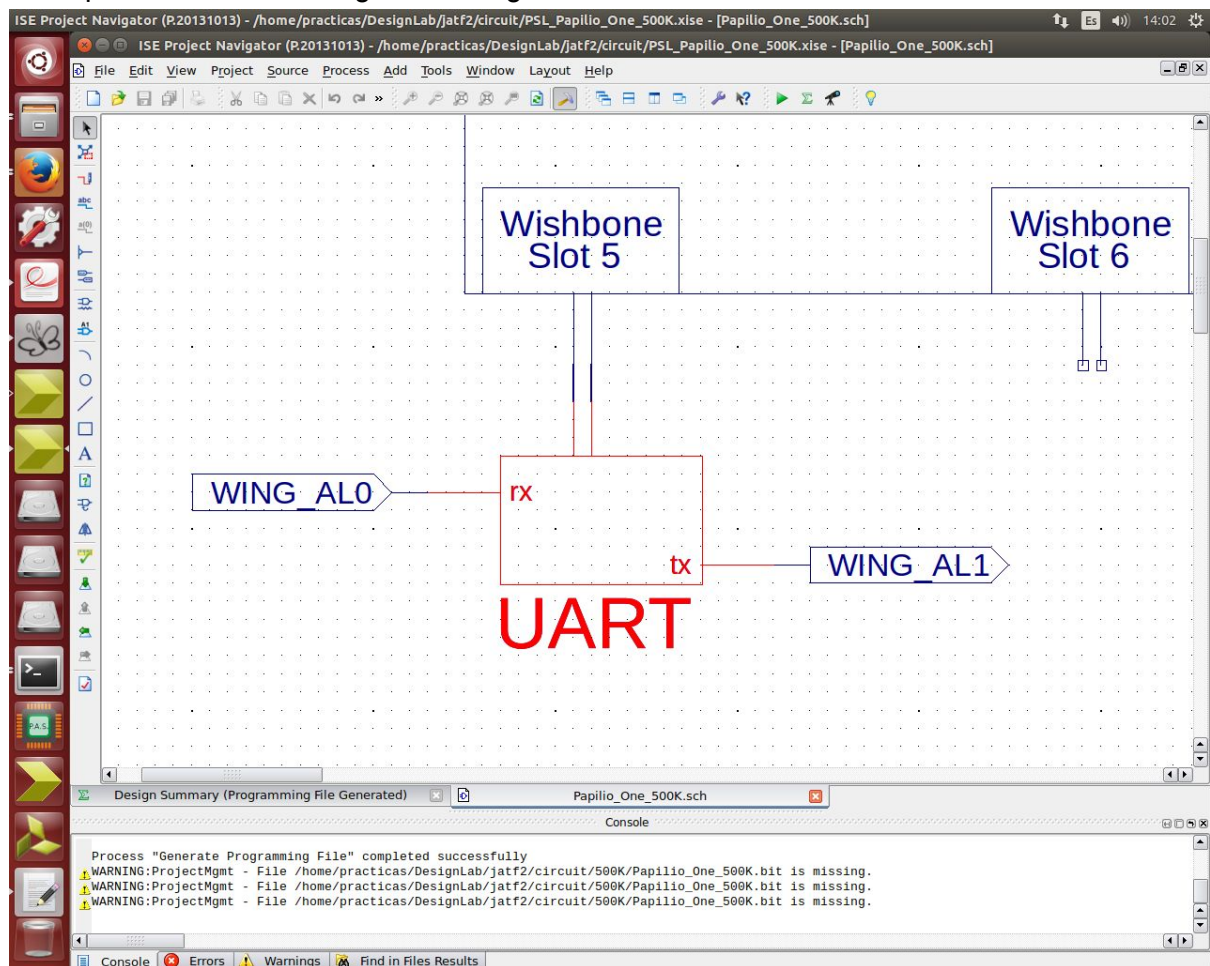
Vamos a usar nuestra placa Papilio One como un analizador lógico. En la última versión de Design Lab (1.07) o posterior, tenemos un botón cuya función es la de cargar el fichero “.bit” de un analizador lógico, pinchamos en este botón, tras comprobar la correcta configuración de puerto y placa. Una vez cargado se nos abrirá un entorno para configurar las entradas del analizador. Una vez configurado, comenzamos a capturar. En el ejemplo siguiente, capturamos la señal generada por el sketch de Arduino “Fade”.

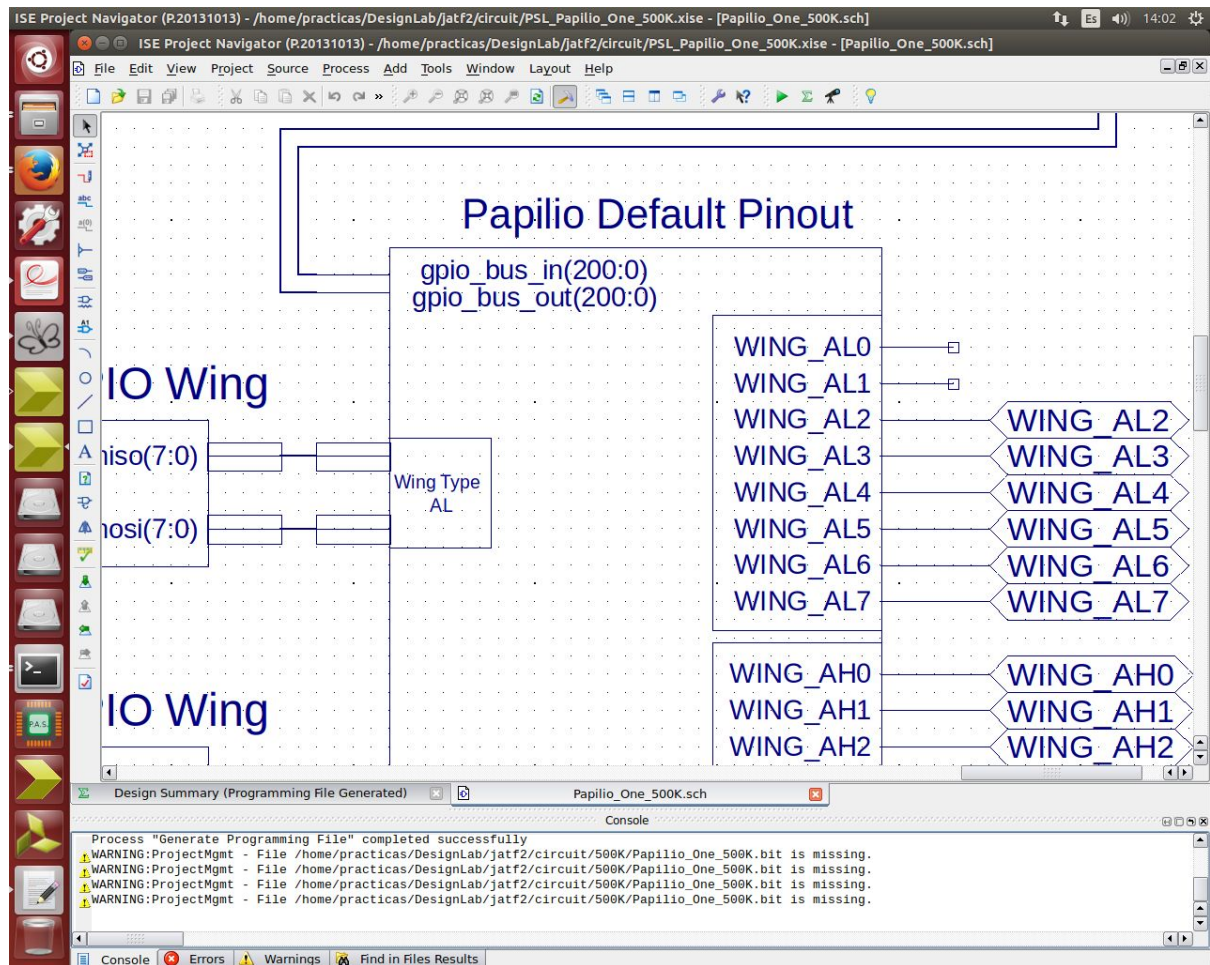


## Rediseñando el SoC. Añadiendo periféricos:

Vamos a rediseñar el SoC ZPUino, para ello crearemos un nuevo proyecto, con la tecla “Ctrl” pulsada, pinchamos en “New ZPUino SOC project”. Primero vamos a guardar el proyecto con un nuevo nombre y procedemos editando el circuito con el botón “Edit Circuit”.

En el entorno Xilinx ISE, vamos a modificar el esquemático pinchando sobre el archivo “.sch”. Podremos ver el esquemático de ZPUino, al que añadiremos una UART. Buscamos la UART entre los símbolos, la colocamos y la conectamos a los puertos tal y como podemos ver en la siguiente imagen.





Finalmente sintetizamos el circuito, generamos el fichero, lo cargamos en nuestra placa Papilio y cargamos el siguiente sketch para probar la UART.

```

/*
Gadget Factory - Project Name
Use this as a template for DesignLab ZPUino System on Chip Projects
To learn more about using DesignLab please visit http://learn.gadgetfactory.net

Tutorials:

Related library documentation:

Hardware:

Special Notes:

created 2014
by Jack Gassett
http://www.gadgetfactory.net

This example code is in the public domain.
*/

```

```

HardwareSerial mySerial(WishboneSlot(5));

int led = 13;
void setup () {
  pinMode(led, OUTPUT); //LED 13 como salida
  mySerial.begin(9600); //Inicializo el puerto serial a 9600 baudios
}

void loop () {
  if (mySerial.available()) { //Si está disponible
    char c = mySerial.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);
    }
  }
}
}

```

Enviaremos comandos para encender y apagar el diodo LED como hemos hecho anteriormente, mediante una sencilla aplicación en python.

```

import serial

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

print("Starting!")

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

arduino.close() #Finalizamos la comunicación

```

## Plataforma Raspberry PI:

### Instalación de Ubuntu Mate:

Descargamos la imagen de Ubuntu Mate y la grabamos en la tarjeta microSD que luego insertaremos en la Raspberry PI.

```

1124 df
1125 umount /dev/sdb1
1126 cd Descargas/

```

```
1127 ls
1128 dd bs=4M if=ubuntu-mate-16.04-desktop-armhf-raspberry-pi-resize.img
of=/dev/sdb1
1129 sudo dd bs=4M
if=ubuntu-mate-16.04-desktop-armhf-raspberry-pi-resize.img of=/dev/sdb
```

Una vez funcionando Ubuntu Mate en la Raspberry PI, conectada al monitor, ratón y teclado, hacemos ifconfig desde un terminal para copiar el nombre de la interfaz ethernet, desactivamos network manager y configuramos la interfaz desde terminal para poder conectarnos a la red de la escuela o a cualquier otra red:

```
2 sudo pluma /etc/network/interfaces
3 sudo ifup enxb827ebfbeada
```

```
auto lo
iface lo inet loopback

iface enxb827ebfbeada inet static
address 10.1.15.96
netmask 255.255.252.0
gateway 10.1.15.78
dns-nameservers 8.8.8.8
```

Una vez tengamos conexión, comprobada haciendo ping al servidor DNS de Google, nos conectamos mediante protocolo ssh (**tarea 3**) desde un ordenador conectado a la misma red, por comodidad, para trabajar de nuevo desde el terminal y empezar a usar los GPIO's.

```
4 ping 8.8.8.8
5 ssh jose@10.1.15.96
```

## Tarea 1:

Vamos a probar a activar y desactivar la salida del puerto 17 al que conectaremos un diodo LED.

```
19 echo 17 > /sys/class/gpio/export
20 echo out > /sys/class/gpio/gpio17/direction
21 echo 1 > /sys/class/gpio/gpio17/value
22 echo 0 > /sys/class/gpio/gpio17/value
23 echo 17 > /sys/class/gpio/unexport
```

## Tarea 2:

Vamos a crear un pequeño programa en python que realizará una función de parpadeo sobre diodos LED en los puertos 17 y 27 y lo vamos a ejecutar para comprobar su funcionamiento.

```
25 sudo nano blink.py
26 sudo python blink.py
```

El código en python "blink.py" que se ha usado es:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT) ## GPIO 17 como salida
GPIO.setup(27, GPIO.OUT) ## GPIO 27 como salida

def blink():
    print "Ejecución iniciada..."
    iteracion = 0
    while iteracion < 30: ## Segundos que durara la funcion
        GPIO.output(17, True) ## Enciendo el 17
        GPIO.output(27, False) ## Apago el 27
        time.sleep(1) ## Esperamos 1 segundo
        GPIO.output(17, False) ## Apago el 17
        GPIO.output(27, True) ## Enciendo el 27
        time.sleep(1) ## Esperamos 1 segundo
        iteracion = iteracion + 2 ## Sumo 2 porque he hecho dos parpadeos
    print "Ejecución finalizada"
    GPIO.cleanup() ## Hago una limpieza de los GPIO

blink() ## Hago la llamada a la función blink
```

## Tarea 4:

Vamos a utilizar la Raspberry PI como servidor web, para ello instalaremos flask, un pequeño framework muy ligero que nos permite crear aplicaciones web de una forma muy sencilla. Escribiremos una aplicación servidor web en python en el archivo weblamp.py y una página web en el archivo main.html que incluiremos en el directorio "/templates". Finalmente ejecutamos la aplicación servidor.

```
1 sudo apt-get install python-pip
2 sudo pip install flask
3 nano weblamp.py
4 mkdir templates
5 cd templates
6 nano main.html
7 sudo python weblamp.py
```

# weblamp.py

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

```
GPIO.setmode(GPIO.BCM)
```

```
# Create a dictionary called pins to store the pin number, name, and pin state:
```

```
pins = {  
    24 : {'name' : 'coffee maker', 'state' : GPIO.LOW},  
    25 : {'name' : 'lamp', '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)

```

```

<!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>

```

## Tarea 5:

Vamos a modificar los ficheros de la anterior tarea para mostrar la temperatura en la página web, recibida a través del puerto serie. Este dato será enviado por la placa Arduino que deberá por lo tanto, tomar este dato a través del sensor tmp36GZ.

```

int sensorPin = 0;
void setup(){
    Serial.begin(9600);
}
void loop(){
    int reading = analogRead(sensorPin);

```



```

        float voltage = reading * 5.0;
        voltage /= 1024.0;
        float temp = (voltage - 0.5) * 100 ;
        byte * b = (byte *) &temp;
        Serial.println(temp);
        delay(1000);
    }

```

```

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

GPIO.setmode(GPIO.BCM)

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

# Create a dictionary called pins to store the pin number, name, and pin state:
pins = {
    24 : {'name' : 'coffee maker', 'state' : GPIO.LOW},
    25 : {'name' : 'lamp', '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)

    temp = arduino.readline(5)

    # Put the pin dictionary into the template data dictionary:
    templateData = {
        'pins' : pins,
        'temp' : temp
    }

    # 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)

arduino.close()

```

```

<!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 %}

{% if temp %}
<h2>The current temperature is: {{ temp }}°</h2>
{% endif %}
</body>
</html>
```

## Tarea 6:

Vamos a volver a modificar estos ficheros para añadir el control de un relé a través de la placa Arduino. Para ello, el Arduino deberá recibir comandos a través del puerto serie para activar y desactivar el relé, a la misma vez que continúa enviando datos del sensor de temperatura por el puerto serie. El servidor igualmente deberá crear y enviar estos comandos para activar y desactivar el relé, usando los mismos botones creados en la página web “main.html” de la tarea 4.

```
//Codigo arduino
int sensorPin = 0;
int relePin = 13;

void setup(){
  Serial.begin(9600);
  pinMode(relePin, OUTPUT);
  digitalWrite(relePin, LOW);
}

void loop(){
  int reading = analogRead(sensorPin);
  char comando;

  if(Serial.available() > 0)
    comando = Serial.read();
    if(comando == 'H')
      digitalWrite(relePin, HIGH);
    else if(comando == 'L')
      digitalWrite(relePin, LOW);

  float voltage = reading * 5.0;
  voltage /= 1024.0;
  float temp = (voltage - 0.5) * 100;
  byte * b = (byte *) &temp;
  if(Serial.availableForWrite())
    Serial.println(temp);
    delay(1000);
}
```

```

# Codigo servidor web en python

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

GPIO.setmode(GPIO.BCM)

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

# Create a dictionary called pins to store the pin number, name, and pin state:
pins = {
    24 : {'name' : 'coffee maker', 'state' : GPIO.LOW},
    25 : {'name' : 'lamp', 'state' : GPIO.LOW}
}

estado = False

temp = 0.0

# 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)

    global temp
    temp = arduino.readline(5)

    # Put the pin dictionary into the template data dictionary:
    templateData = {
        'pins' : pins,
        'temp' : temp
    }

    # 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)
#    arduino.write("H")
    # Save the status message to be passed into the template:
    message = "Turned " + deviceName + " on."
if action == "off":
    GPIO.output(changePin, GPIO.LOW)
#    arduino.write("L")
    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)

global temp
# Along with the pin dictionary, put the message into the template data dictionary:
templateData = {
    'message' : message,
    'pins' : pins,
    'temp' : temp
}

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

@app.route("/<action>")
def rele(action):
    # If the action part of the URL is "on," execute the code indented below:
    if action == "on":
        # Set the pin high:
        arduino.write("H")
        # Save the status message to be passed into the template:
        message = "rele encendido."
        estado = True
    if action == "off":
        arduino.write("L")
        message = "rele apagado."
        estado = False

global temp

```

# Along with the pin dictionary, put the message into the template data dictionary:

```
templateData = {  
    'message' : message,  
    'pins' : pins,  
    'estado' : estado,  
    'temp' : temp  
}
```

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

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

```
arduino.close()
```

```
<!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 temp %}
```

```
    <h2>The current temperature is: {{ temp }}°</h2>
```

```
    <a href="/">Actualizar</a>
```

```
    {% endif %}
```

```
    <p>El relé
```

```
    {% if estado == True %}
```

```
        is currently on (<a href="/off">turn off</a>)
```

```
    {% else %}
```

```
        is currently off (<a href="/on">turn on</a>)
```

```
    {% endif %}
```

```
    </p>
```

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

```
</body>  
</html>
```

```
<!--Codigo pagina web pin.html-->
```

```
<!DOCTYPE html>  
<head>  
    <title>{{ title }}</title>  
</head>  
  
<body>  
    <h1>Pin Status</h1>  
    <h2>{{ response }}</h2>  
</body>  
</html>
```