

Memoria de Prácticas

Plataformas de desarrollo

Rafael Escudero Lirio
rafaeslir@gmail.com
Ingeniería Informática de Computadores

Índice

-Plataforma Arduino Página 3

-Plataforma Zpuino Página 26

-Plataforma Raspberry PI 40

Plataforma Arduino

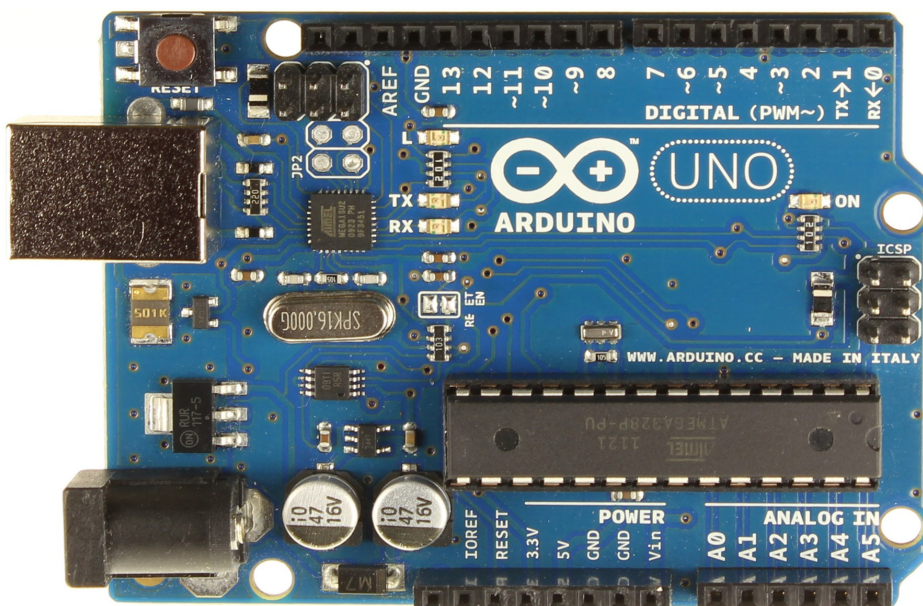
-Objetivos:

- ◆ Conocer la plataforma arduino, sus características, sus variantes, sus modos de programación.
- ◆ Conocer una serie de componentes básicos de hardware típicos de aplicaciones de sistemas empotrados
- ◆ Preparar el PC para que funcione el entorno de desarrollo de arduino
- ◆ Realizar ejemplos básicos de funcionamiento sobre arduino
- ◆ Desarrollar otros ejemplos de uso de arduino manejando diversos componentes hardware

-Introducción a 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).



-Especificaciones de la placa:

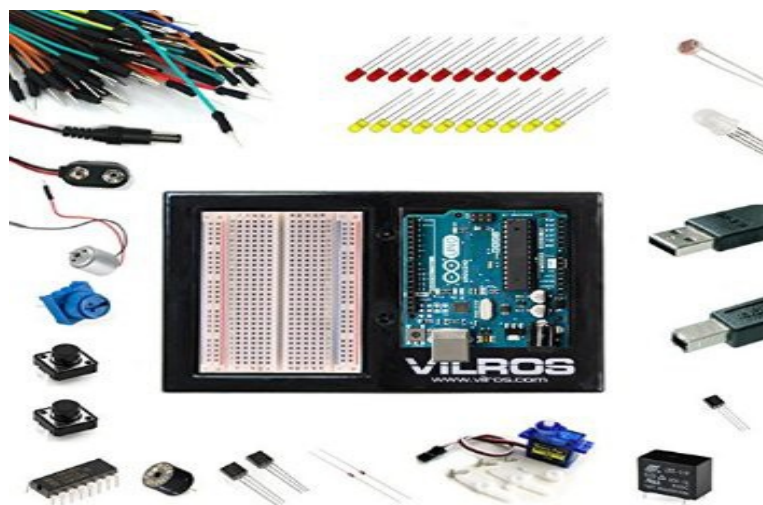
Cada uno de los 14 pines digitales se puede usar como entrada o como salida. Funcionan a 5V, cada pin puede suministrar hasta 40 mA. La intensidad máxima de entrada también es de 40 mA.

Cada uno de los pines digitales dispone de una resistencia de pull-up interna de entre 20K Ω y 50 K Ω que está desconectada, salvo que nosotros indiquemos lo contrario.

Arduino también dispone de 6 pines de entrada analógicos que trasladan las señales a un conversor analógico/digital de 10 bits.

Microcontrolador	Atmega328
Voltaje de operación	5V
Voltaje de entrada (Recomendado)	7 – 12V
Voltaje de entrada (Límite)	6 – 20V
Pines para entrada- salida digital.	14 (6 pueden usarse como salida de PWM)
Pines de entrada analógica.	6
Corriente continua por pin IO	40 mA
Corriente continua en el pin 3.3V	50 mA
Memoria Flash	32 KB (0,5 KB ocupados por el bootloader)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

En las prácticas usaremos el kit para principiante de Arduino con sus componentes además de un motor DC y relés.



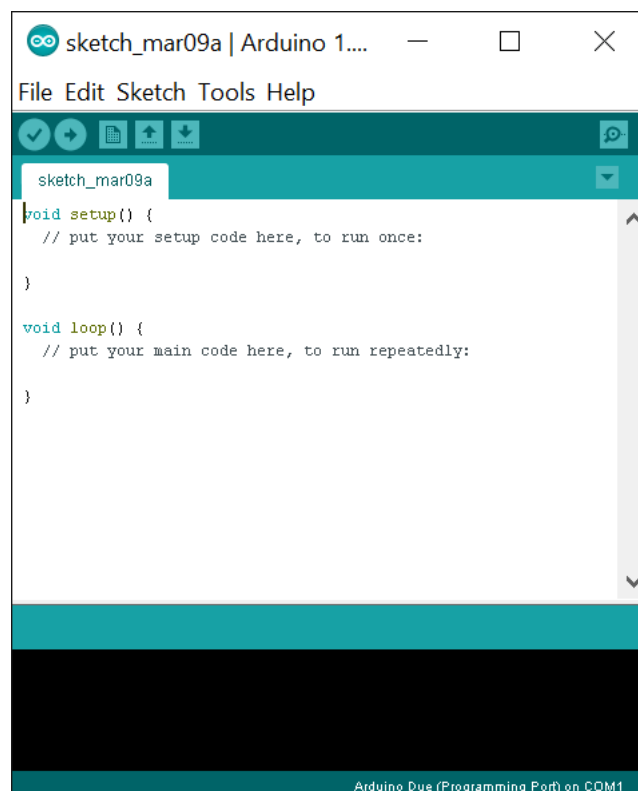
-Entorno de desarrollo:

El lenguaje de programación de Arduino es C++, aunque es posible programarlo en otros lenguajes. No es un C++ puro sino que es una adaptación que proviene de avr-libc que provee de una librería de C de alta calidad para usar con GCC (compilador de C y C++) en los microcontroladores AVR de Atmel, denominado avr-gcc, y otras muchas funciones específicas para las MCU AVR de Atmel.

El lenguaje de los AVR es un entorno C para programar los chips de Atmel, la mayor parte del lenguaje o core de Arduino está escrito con constantes y funciones AVR y hay bastantes cosas que no son fáciles de hacer con el lenguaje de Arduino sin el uso de código AVR de la avr-libc.

Como el entorno de Arduino usa GCC para compilar el código, todas las variables de los puertos y de los registros de una MCU de Atmel son soportadas por el IDE de Arduino.

Las herramientas necesarias para programar los microcontroladores AVR de Atmel son avr-binutils, avr-gcc y avr-libc y ya están incluidas en el IDE de Arduino, cuando compilamos y cargamos un sketch estamos usando estas herramientas.

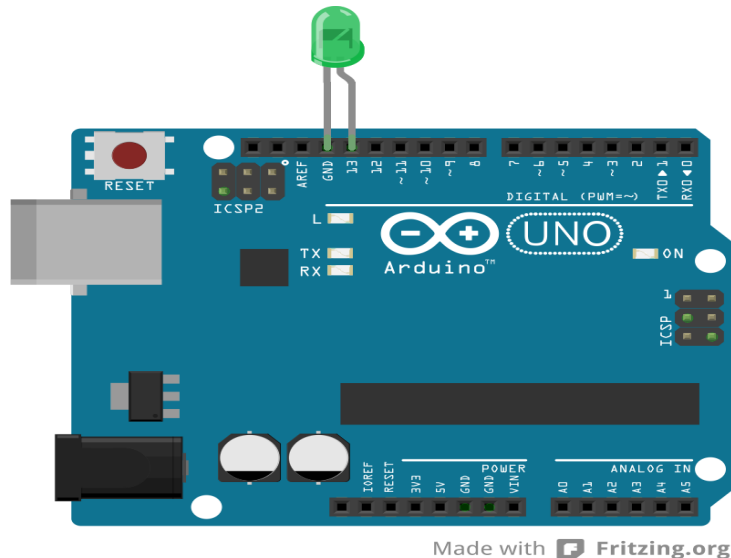


-Ejercicio 1:

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

.Circuito:

El circuito a realizar para este ejercicio es muy sencillo, en la imagen se muestra la disposición correcta si vamos a usar la resistencia interna de la placa, de otra manera bastaría con crear un circuito idéntico colocando una resistencia entre el LED y la entrada o la salida. Nunca se debe conectar un LED directamente a una fuente ya que no sabemos cuánto son capaces de resistir estos.



.Código:

En este código básico declaramos la variable led como el valor 13, que será la salida que vamos a utilizar. En el cuerpo del programa lo que hacemos es comprobar el puerto serie y los valores que nos llegan de él. Según estos valores hacemos que la salida 13 de un 0 o un 1 lógico, con lo que encendemos o apagamos el LED.

```
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);
    }
  }
}
```

Código en Python:

En este código definimos un puerto serie al que llamamos arduino y que nos permite enviar los comando 'H' y 'L' al puerto de la placa. De esta forma si escribimos en la consola: 'H' el código de arduino capta el mensaje y enciende el LED, lo mismo pasa de forma inversa para el comando 'L'.

```
import serial
```

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

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

```
while True:
```

```
    comando = raw_input('Introduce un comando: ') #Input
```

```
    arduino.write(comando) #Mandar un comando hacia Arduino
```

```
    if comando == 'H':
```

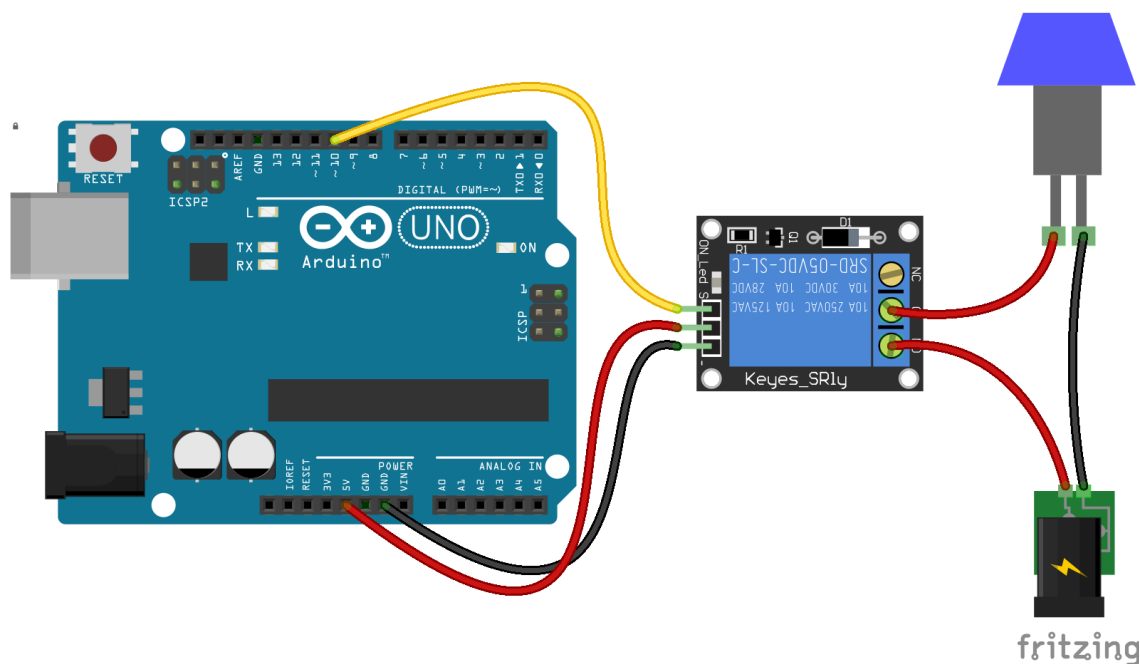
```
        print('LED ENCENDIDO')
```

```
    elif comando == 'L':
```

```
        print('LED APAGADO')
```

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

Para finalizar el ejercicio se pedía añadir al circuito un relé conectado a una bombilla de forma que el relé funcionase tal y como hasta ahora lo estaba haciendo el LED. El relé al conmutar dejaría pasar la corriente a la bombilla conectada a 220V.



Nota: Cambiar salida 10 de la imagen por la 13, funciona exactamente igual.

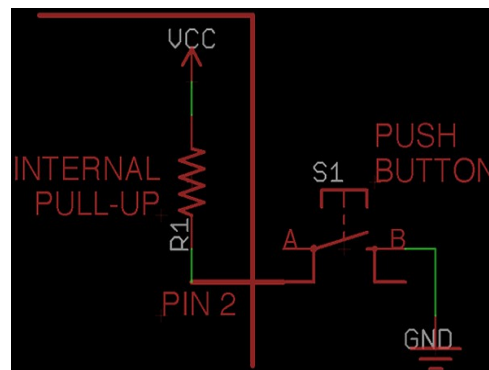
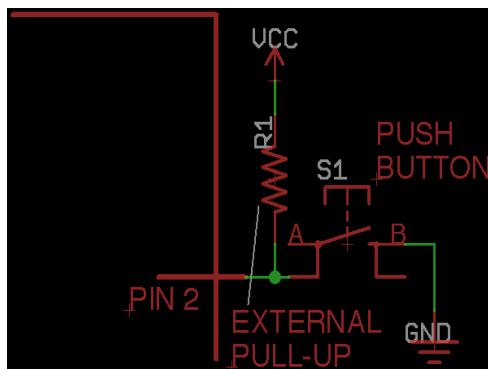
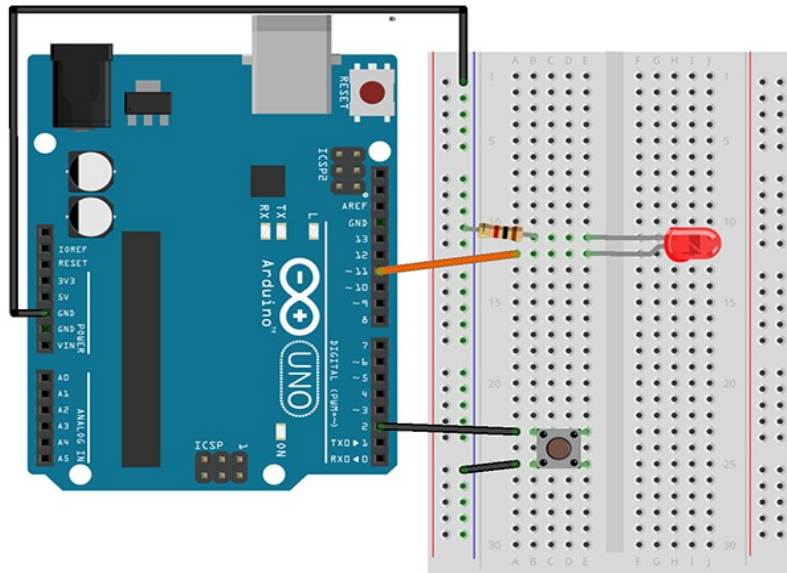
De esta forma concluimos la creación de nuestro primer sistema domótico.

-Ejercicio 2:

Encendido apagado de un led mediante pulsador

.Circuito:

Para este ejercicio necesitamos usar un botón además del LED y la resistencia. Vamos a hacer uso de las interrupciones para controlar el encendido de un LED, en concreto usaremos la resistencia PULL-UP de arduino.



.Código:

En este código declaramos el pin que estemos usando como `INPUT_PULLUP` con el propósito que se menciona anteriormente. Después leemos la entrada de la placa conectada al pulsador y en función del valor Apagamos o encendemos en LED. Pulsador en bajo = encendemos LED, en alto = apagamos LED.


```

pinMode(button_pin, INPUT_PULLUP);

button_state = digitalRead(button_pin);

if (button_state == LOW)
{
    // Turn LED on:
    digitalWrite(led_pin, HIGH);
}
// otherwise (i.e., if button_state is HIGH)
else
{
    // Turn LED off:
    digitalWrite(led_pin, LOW);
}

```

Ahora encontramos un último problema, y es que al pulsar el botón el LED puede encenderse y apagarse de forma errática, para solucionar esto debemos añadir esperas al código tras escribir en las salidas de la placa:

```

if (button_state == LOW)
{
    // Turn LED on:
    digitalWrite(led_pin, HIGH);
    delay(200);
}
// otherwise (i.e., if button_state is HIGH)
else
{
    // Turn LED off:
    digitalWrite(led_pin, LOW);
    delay(200);
}

```

-Ejercicio 3:

Control de la posición de un servo motor con un potenciómetro

.Circuito:

Para este ejercicio haremos uso de un servomotor, y un potenciómetro.

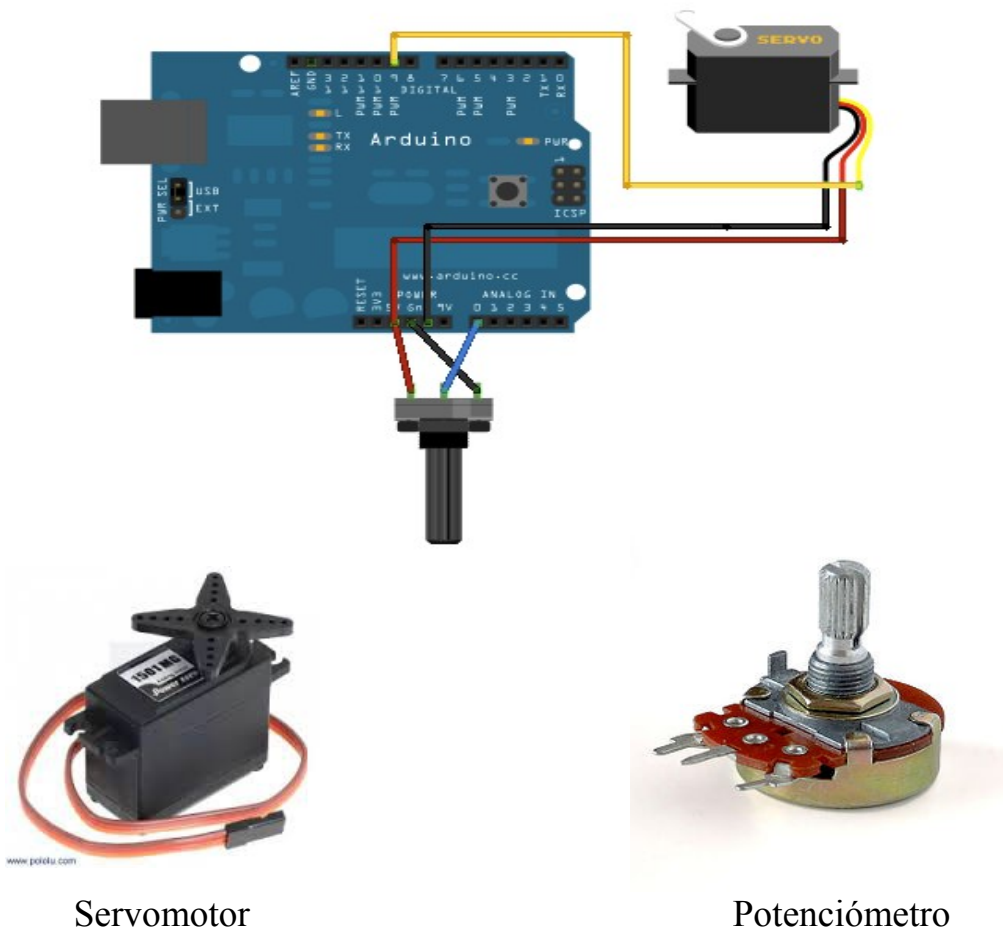
-Servomotor:

Un servomotor (también llamado 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.

-Potenciómetro:

Un potenciómetro es uno de los dos usos que posee la resistencia o resistor variable mecánica (con cursor y de al menos tres terminales). Conectando los terminales extremos a la diferencia de potencial a regular (control de tensión), se obtiene entre el terminal central (cursor) y uno de los extremos una fracción de la diferencia de potencial total, se comporta como un divisor de tensión o voltaje.

El ejercicio consiste en controlar el ángulo del servomotor mediante los valores recogidos con el potenciómetro.



.Código:

Podemos usar el sencillo ejemplo de la web de arduino.

En este código definimos un servomotor usando la librerías del lenguaje y le asociamos el pin que vamos a usar. En el cuerpo del programa hemos de recoger el valor registrado del potenciómetro y convertirlo a un valor legible por la función que mueve al servo.

```
Servo myservo; // create servo object to control a servo
int potpin = 0; // analog pin used to connect the potentiometer
int val; // variable to read the value from the analog pin

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

void loop() {
  val = analogRead(potpin); // reads the value of the potentiometer (value
  // between 0 and 1023)
  val = map(val, 0, 1023, 0, 180); // scale it to use it with the servo (value
  // between 0 and 180)
  myservo.write(val); // sets the servo position according to the scaled
  // value
  delay(15); // waits for the servo to get there
}
```

Para terminar el ejercicio necesitamos poder controlar el ángulo del servo mediante la consola con programa en python parecido al del ejercicio 1.

Para ello modificaremos el código de arduino de forma que el valor que pasemos a la función del servo será el que leamos por el puerto serie, también podemos modificar el código de python para que sea acorde con el programa pero ni siquiera es necesario para el correcto funcionamiento de éste.

```
Servo myservo; // create servo object to control a servo
int val; // variable to read the value from the analog pin

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

void loop() {
  // between 0 and 180)
  myservo.write(Serial.parseInt()); // sets the servo position according
  // to the scaled value
  delay(15); // waits for the servo to get there
}
```

Ya no es necesario realizar la conversión del valor dado que le pasaremos un valor directamente legible por la función. Podemos utilizar la función `parseInt()` de las librerías de arduino que cogerá el primer valor integer que se pasa por el puerto serie y lo devuelve como respuesta.

Podemos modificar el código de python para que vaya acorde con el programa actual aunque en cuestión de funcionalidad no es vital hacerlo.

```
import serial
```

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

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

```
while True:
```

```
    comando = raw_input('Introduce un ángulo: ') #Input
```

```
    arduino.write(comando) #Mandar un comando hacia Arduino
```

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

-Ejercicio 4:

Control de la velocidad de giro de un motor de continua

.Circuito:

Para este ejercicio necesitaremos además de un potenciómetro, un motor DC y el controlador L293D.

-Motor DC:

Un motor de corriente continua convierte la energía eléctrica en mecánica. Se compone de dos partes: el estator y el rotor.

El estator es la parte mecánica del motor donde están los polos del imán.

El rotor es la parte móvil del motor con devanado y un núcleo, al que llega la corriente a través de las escobillas.

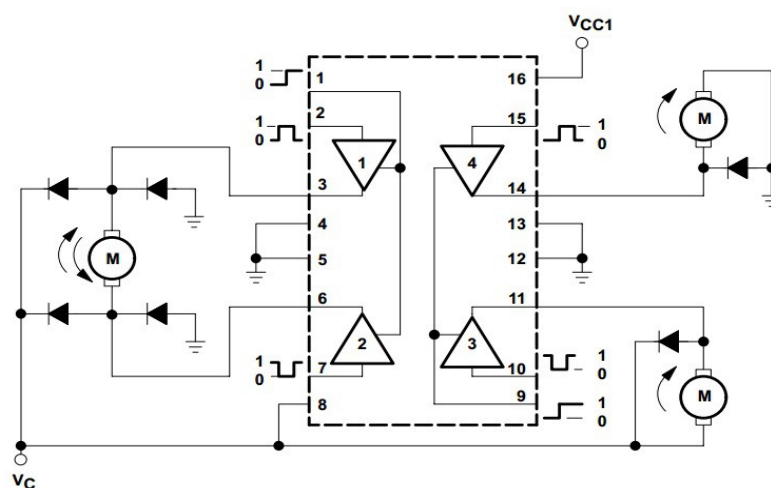
Cuando la corriente eléctrica circula por el devanado del rotor, se crea un campo electromagnético. Este interactúa con el campo magnético del imán del estator. Esto deriva en un rechazo entre los polos del imán del estator y del rotor creando un par de fuerza donde el rotor gira en un sentido de forma permanente.

Si queremos cambiar el sentido de giro del rotor, tenemos que cambiar el sentido de la corriente que le proporcionamos al rotor; basta con invertir la polaridad de la pila o batería.

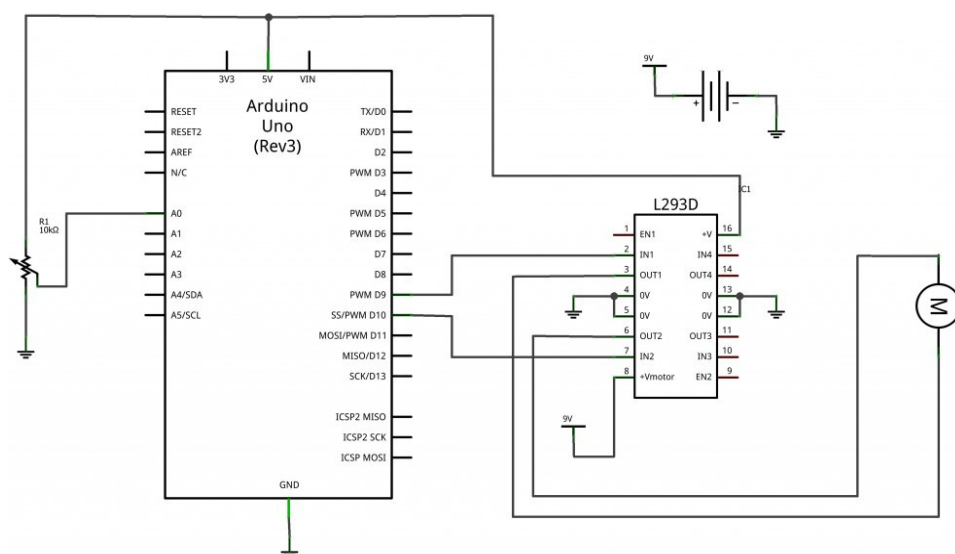
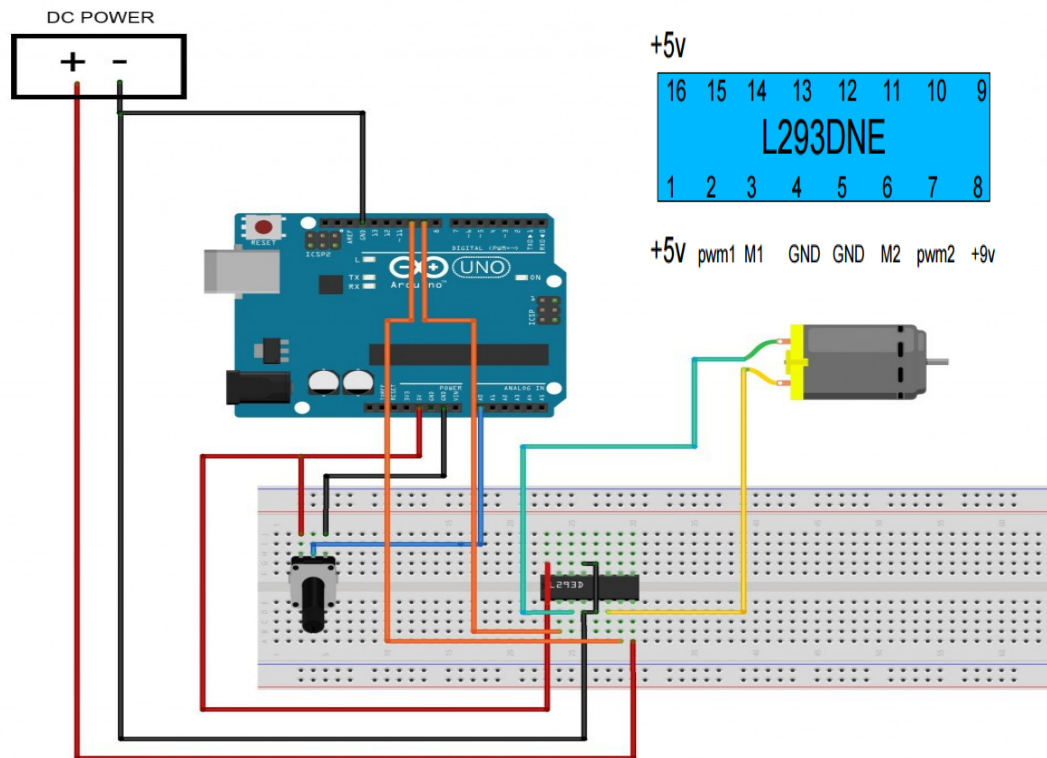
-Controlador L293D:

Para controlar un motor DC desde Arduino, tendremos que usar un driver para motores para proporcionarle más corriente al motor ya que las salidas del Arduino sólo dan 40mA. De esta manera, con el driver podemos alimentar el motor con una fuente de alimentación externa.

El L293D es un integrado para controlar motores DC que usa el sistema puente en H. ¿Qué es el puente en H? Es un sistema para controlar el sentido de giro de un motor DC usando cuatro transistores. En la imagen vemos que los transistores se comportan como interruptores y dependiendo que transistores conducen y cuáles no cambia la polarización del motor, y con esto el sentido de giro.



Vamos a realizar un proyecto de forma que el motor DC vaya a unos determinados velocidad y sentido que controlaremos mediante el potenciómetro, la complicación llega si queremos que el motor pueda cambiar su sentido, para ello necesitamos invertir los valores que llegan al motor (ver funcionamiento de un motor DC anteriormente explicado) por lo que necesitaremos la ayuda de un controlador L293D.



- Pins 4,5,12,13 del L293D a masa.
- Juntar las masas del Arduino y de la fuente de alimentación externa.
- Pin 8 del L293D a 9V de la fuente de alimentación externa. Es el voltaje que proporciona al motor.
- Pin 16 del L293D a 5V. Es la alimentación del L293D, puede alimentarse directamente desde la alimentación que proporciona el Arduino.
- El potenciómetro puede ser de cualquier valor.

.Código:

El código está explicado en los comentarios del mismo aunque su funcionamiento es muy sencillo, recogemos los valores PWM que recogemos del potenciómetro y los traducimos a valores legibles estos se pasan al controlador que finalmente hace funcionar el motor.

```
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()
{
    //Inicializamos los pins de salida
    pinMode(pin2,OUTPUT);
    pinMode(pin7, OUTPUT);
}

void loop()
{
    //Almacenamos el valor del potenciómetro en la variable
    valorpote=analogRead(pote);

    //Como la entrada analógica del Arduino es de 10 bits, el rango va de 0 a 1023.
    //En cambio, las salidas del Arduino son de 8 bits, quiere decir, rango entre 0 a
    255.
    //Por esta razón tenemos que mapear el número de un rango a otro usando este
    código.
    pwm1 = map(valorpote, 0, 1023, 0, 255);
    pwm2 = map(valorpote, 0, 1023, 255, 0); //El PWM 2 esta invertido respecto al
    PWM 1

    //Sacamos el PWM de las dos salidas usando analogWrite(pin,valor)
    analogWrite(pin2,pwm1);
    analogWrite(pin7,pwm2);
}
```

-Ejercicio 5:

Impresión en una pantalla LCD

.Circuito:

Para este ejercicio necesitaremos un potenciómetro y una Pantalla LCM1602C.

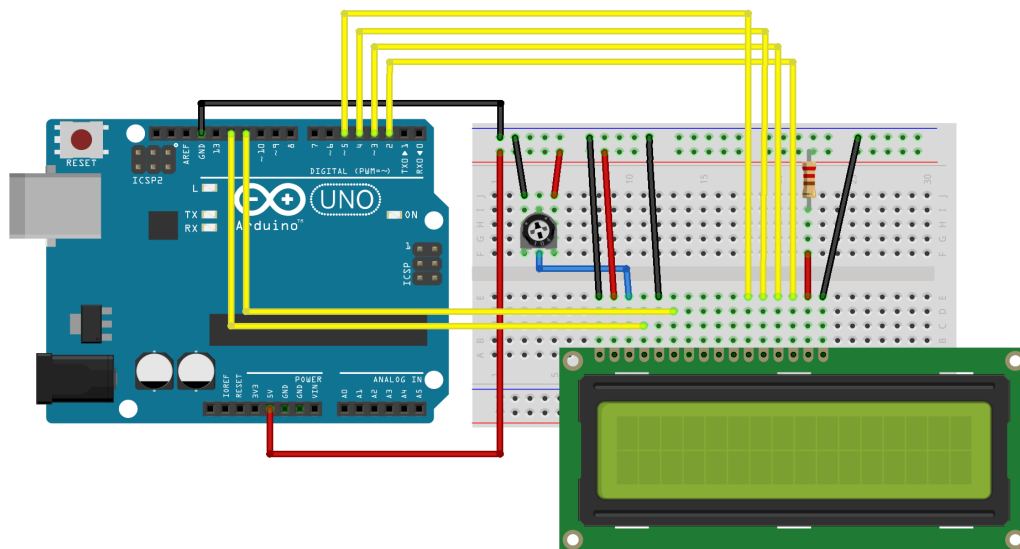
-Pantalla LCM1602C:

Una pantalla de cristal líquido o LCD es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de energía eléctrica.

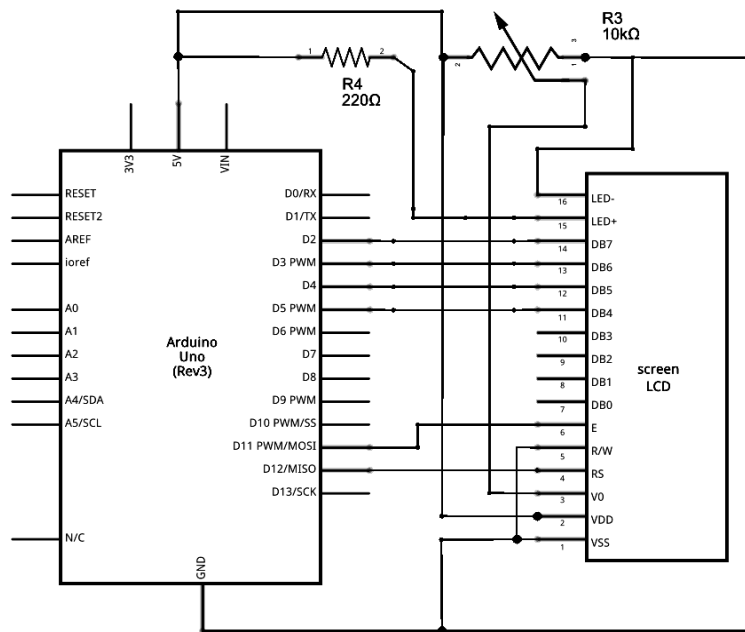


El ejercicio consiste en realizar un programa que haga uso de una pantalla LCD conectada a arduino de forma que, en última instancia seamos capaces de leer los mensajes que enviemos por consola en la pantalla LCD.

Podemos usar el ejemplo de la web de Arduino:



Como puede observarse, lo más complicado será conectar la pantalla en todos los pines sin equivocarse.



- LCD RS pin to digital pin 12
- LCD Enable pin to digital pin 11
- LCD D4 pin to digital pin 5
- LCD D5 pin to digital pin 4
- LCD D6 pin to digital pin 3
- LCD D7 pin to digital pin 2

El potenciómetro que se observa en el circuito tiene como función controlar la potencia lumínica de la pantalla, no es necesario controlar esto por software, vale con colocarlo en esa determinada posición.

.Código:

Podemos probar primero el LCD mediante un sencillo programa en el que definimos un lcd gracias a las librerías de arduino y le pasamos un mensaje con la función print() y para definir la posición de escritura setCursor(x,y).

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  lcd.begin(16, 2);
  Serial.begin(9600);
}

void loop() {
  lcd.setCursor(0, 0);
  lcd.print("Laboratorio de Desarrollo Hardware");
  lcd.setCursor(0, 1);
  lcd.print("Rafael Escudero Lirio");
}
```

Para leer mensajes del puerto serie también es sencillo, simplemente esperamos a que el puerto esté disponible, una vez pase esto leemos las líneas del puerto y las escribimos en el LCD. Esto se repite en bucle que refresca al LCD para que podamos enviar tantos mensajes como queramos.

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

```
void setup() {  
  // set up the LCD's number of columns and rows:  
  lcd.begin(16, 2);  
  // initialize the serial communications:  
  Serial.begin(9600);  
}  
  
void loop() {  
  // when characters arrive over the serial port...  
  if (Serial.available()) {  
    // wait a bit for the entire message to arrive  
    delay(100);  
    // clear the screen  
    lcd.clear();  
    // read all the available characters  
    while (Serial.available() > 0) {  
      // display each character to the LCD  
      lcd.write(Serial.read());  
    }  
  }  
}
```

-Ejercicio 6:

Empleando el sensor de temperatura tmp36GZ diseñar un termómetro

.Circuito:

Para este ejercicio vamos a necesitar la pantalla LCD al igual que en el anterior y vamos a añadir un sensor de temperatura TMP36.

-Sensor TMP36:

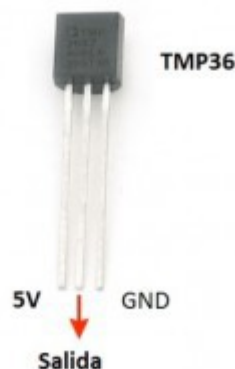
Mide la temperatura en grados centígrados.

Funciona entre -50° C y 125°C para el TMP36.

Funciona entre 0° C y 100°C para el LM35DZ .

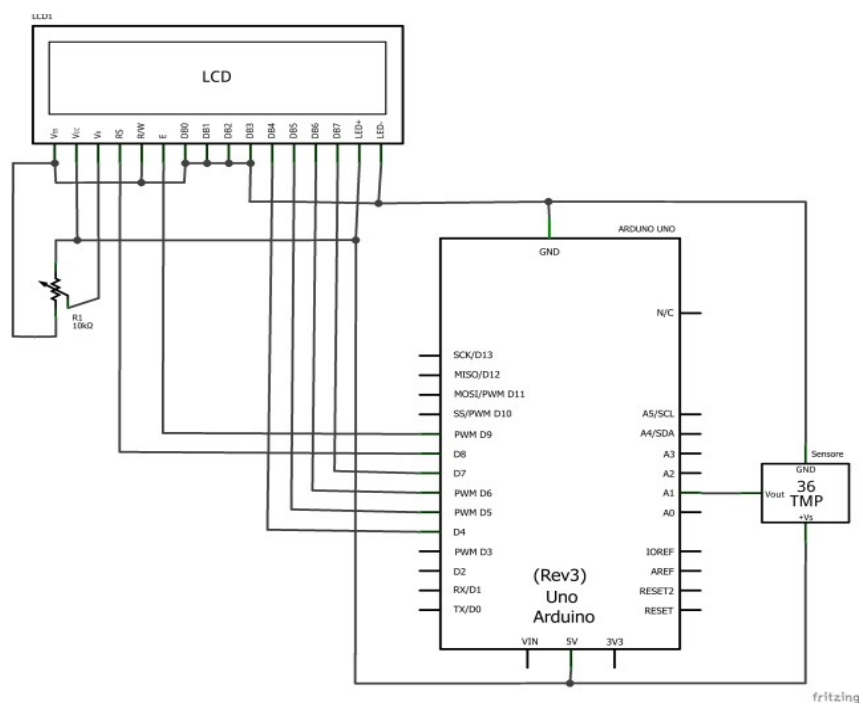
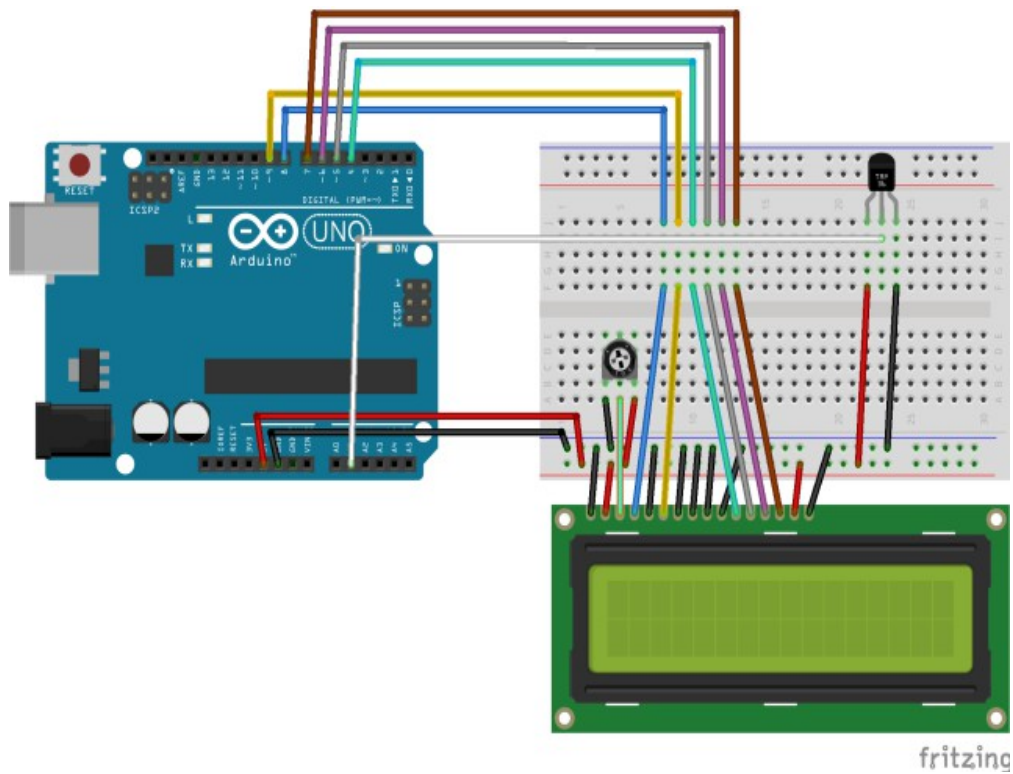
No es especialmente preciso, ya que tiene $\pm 1^{\circ}\text{C}$ de incertidumbre, pero normalmente nos sobra para proyectos sencillos y es muy barato.

EL encapsulado es similar al de un transistor y también tiene tres patas, así que mucho cuidado con confundirlos. Intentad leer las letras que lleva serigrafiadas (si podéis, porque suelen ser tan pequeñas que a veces no se leen ni con lupa).



El pin central es el de señal, pero para saber cuál es GND y 5V, el encapsulado tiene una cara plana y otra curva. Poniendo la cara plana mirando hacia vosotros con las patas hacia abajo (de modo que podáis leer el modelo), el pin de la izquierda es alimentación 5V y naturalmente el otro es GND.

Básicamente vamos a reutilizar lo aprendido en el ejercicio anterior y vamos a añadir al sistema un sensor de temperatura. Después mostramos en el LCD la temperatura registrada.



Para el circuito sencillamente vamos a reutilizar el anterior y vamos a añadir el sensor, dándole corriente y recogiendo su valor en el pin A1.

.Código:

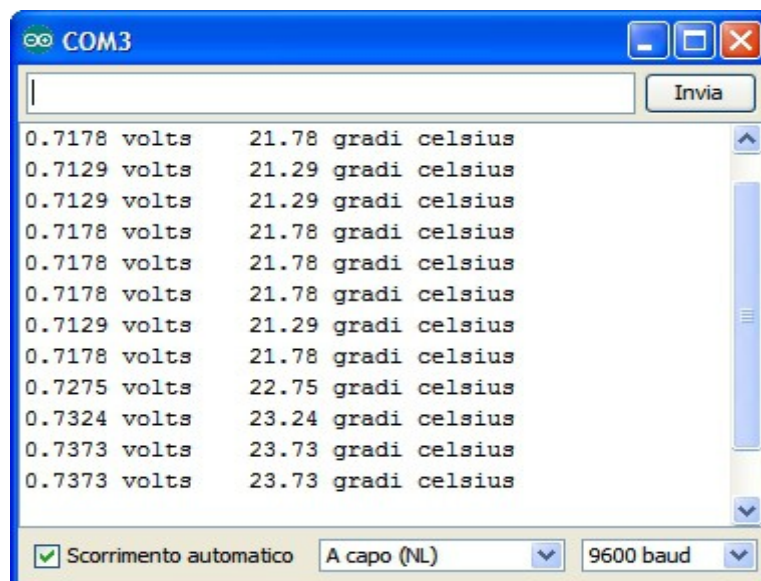
Básicamente leemos el pin A1, al que hemos conectado la salida del sensor, se trata de un valor analógico que debemos convertir, tras de esto simplemente guardamos los valores en variables y los mostramos en el LCD como hacíamos en ejercicios anteriores. También los mostramos por el puerto para comprobar que todo se ejecuta sin errores.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
int val_Adc = 0;
float temp = 0;
const int Pin_TMP36 = A1;

void setup()
{
  lcd.begin(16, 2);
  Serial.begin(9600);
  lcd.clear();
  lcd.setCursor( 0, 0 );
  lcd.print( "Temperatura: ");
}

void loop()
{
  int val_Adc = analogRead(Pin_TMP36);
  float voltage = (val_Adc / 1024.0) * 5.0;
  float temp = (voltage - .5) * 100;
  Serial.print(voltage,4);
  Serial.print(" voltios ");
  Serial.print(temp,2);
  Serial.println(" Grados");
  lcd.setCursor( 0, 1 );
  lcd.print( temp,1 );
  lcd.print( ' ' );
  lcd.print( (char) 223 );
  lcd.print( 'C' );
  delay( 500 );
}
```



-Ejercicio 7:

Contador 00 a 59 en display 7-seg cada segundo

.Circuito:

Para este ejercicio vamos a necesitar una placa de expansión con los displays 7 segmentos.

-Placa de expansión:

Esta placa cuenta con 2 displays 7 segmentos y está especialmente diseñada para usarse con arduino, de hecho no es necesario más que conectarla directamente a la placa puesto que todos sus pins coinciden.

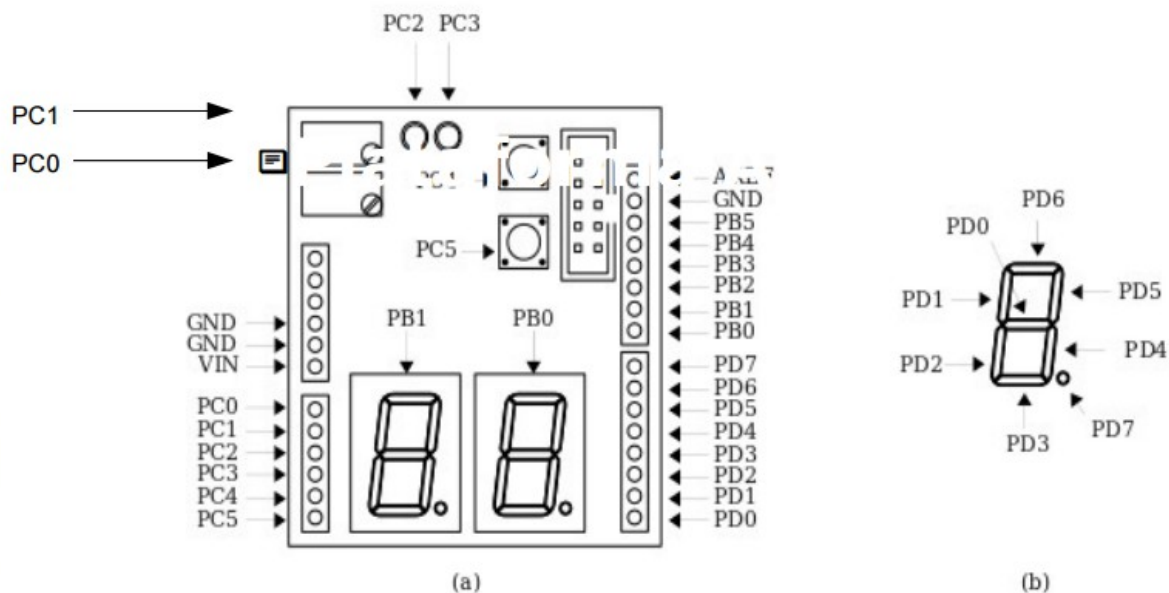
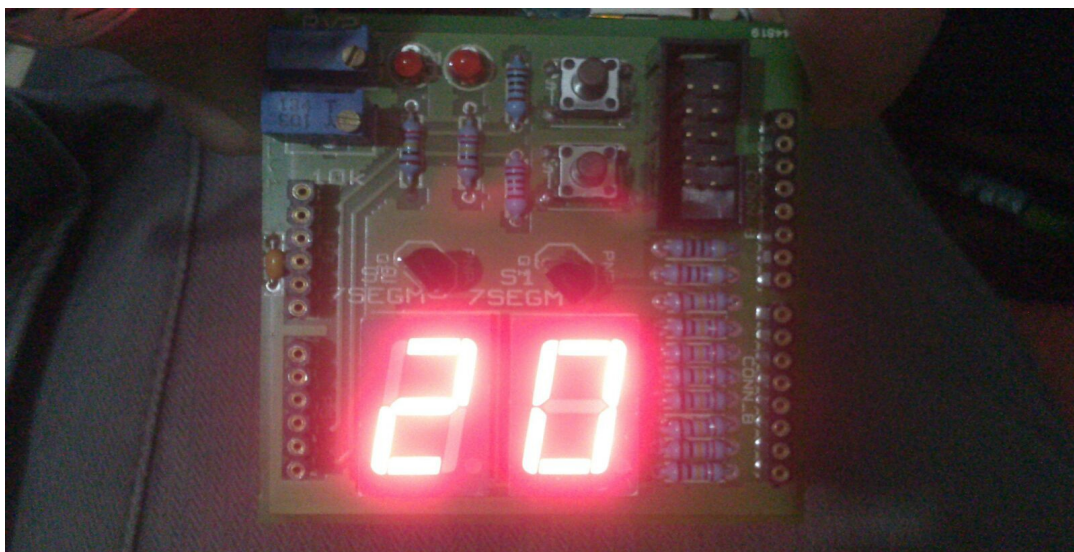


Figura 8. a) Placa de expansión E/S para Arduino. b) Detalle de conexión de los segmentos a los pines.



Sólo hay una manera correcta de conectar la placa de expansión, a continuación, para hacerla funcionar correctamente debemos controlar sus entradas mediante software.

.Código:

Cada 7 segmentos tiene 8 señales que determinan que número será visible, aún siendo 2 display sólo hay 8 entradas y una señal de selección, esto se debe a que debemos usar. En el código se define cada dato con su serie de entradas y salidas como alto o bajo según el valor. Lo importante es el loop final del código en el que se un un doble bucle for, uno para controlar las decenas y el otro para las unidades, se deben dar muchas iteraciones de forma que el refresco sea tal que el ojo humano no pueda percibirlo, por último añadimos los delays al código y hacemos el cálculo de forma que una iteración dure 1 segundo.

```
void setup() {  
  pinMode(0, OUTPUT);  
  pinMode(1, OUTPUT);  
  pinMode(2, OUTPUT);  
  pinMode(3, OUTPUT);  
  pinMode(4, OUTPUT);  
  pinMode(5, OUTPUT);  
  pinMode(6, OUTPUT);  
  pinMode(7, OUTPUT);  
  pinMode(8, OUTPUT);  
  pinMode(9, OUTPUT);  
  pinMode(10, OUTPUT);  
  pinMode(11, OUTPUT);  
  pinMode(12, OUTPUT);  
  pinMode(13, OUTPUT);  
  pinMode(A5, OUTPUT);  
  pinMode(A4, OUTPUT);  
  pinMode(A3, OUTPUT);  
  pinMode(A2, OUTPUT);  
  pinMode(A1, OUTPUT);  
  pinMode(A0, OUTPUT);  
}  
  
void write_data(int dato){  
  if(dato == 0){ //define el 0  
    digitalWrite(0, LOW);  
    digitalWrite(1, HIGH);  
    digitalWrite(2, HIGH);  
    digitalWrite(3, HIGH);  
    digitalWrite(4, HIGH);  
    digitalWrite(5, HIGH);  
    digitalWrite(6, HIGH);  
    digitalWrite(7, LOW);  
  }  
  if(dato == 1){ //define el 1  
    digitalWrite(0, LOW);  
    digitalWrite(1, LOW);  
    digitalWrite(2, LOW);  
    digitalWrite(3, LOW);  
    digitalWrite(4, HIGH);  
    digitalWrite(5, HIGH);  
    digitalWrite(6, LOW);  
    digitalWrite(7, LOW);  
  }  
}
```

```

}
if(dato == 2){
    digitalWrite(0, HIGH);
    digitalWrite(1, LOW);
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
}
if(dato == 3){
    digitalWrite(0, HIGH);
    digitalWrite(1, LOW);
    digitalWrite(2, LOW);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
}
if(dato == 4){
    digitalWrite(0, HIGH);
    digitalWrite(1, HIGH);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(6, LOW);
    digitalWrite(7, LOW);
}
if(dato == 5){
    digitalWrite(0, HIGH);
    digitalWrite(1, HIGH);
    digitalWrite(2, LOW);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
}
if(dato == 6){
    digitalWrite(0, HIGH);
    digitalWrite(1, HIGH);
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(6, LOW);
    digitalWrite(7, LOW);
}
if(dato == 7){
    digitalWrite(0, LOW);
    digitalWrite(1, LOW);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
}

```



```

}
if(dato == 8){
    digitalWrite(0, HIGH);
    digitalWrite(1, HIGH);
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
}
if(dato == 9){
    digitalWrite(0, HIGH);
    digitalWrite(1, HIGH);
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
}
}

void loop() {
    for(int i = 3; i < 8; i++){
        for(int e = 0; e < 100; e++){
            digitalWrite(8, LOW);
            digitalWrite(9, HIGH);
            write_data(e/10);
            delay(50);
            digitalWrite(8, HIGH);
            digitalWrite(9, LOW);
            write_data(i);
            delay(50);
        }
    }
}

```

Plataforma ZPUino

-Objetivos:

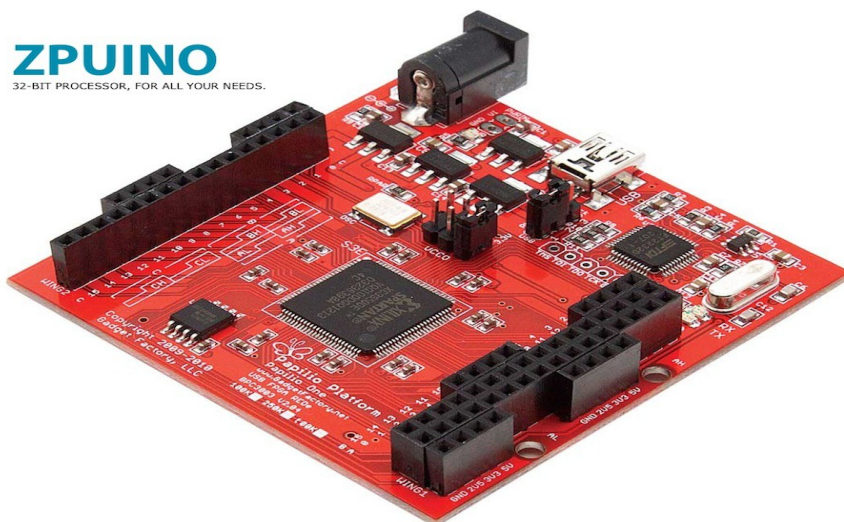
- ◆ Conocer la plataforma PAPILIO
- ◆ Instalar y configurar el entorno de trabajo de papilio: DESIGN LAB
- ◆ Conocer que se puede hacer desde DESIGN LAB sobre las placas PAPILIOS:
 - Diseñando circuitos a nivel de captura de esquemáticos e implementarlos en la FPGA
 - Cargar el SoC ZPUINO → Desarrollar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

-Introducción a ZPUino:

Son placas de desarrollo hardware abiertas basadas en FPGAs XILINX:

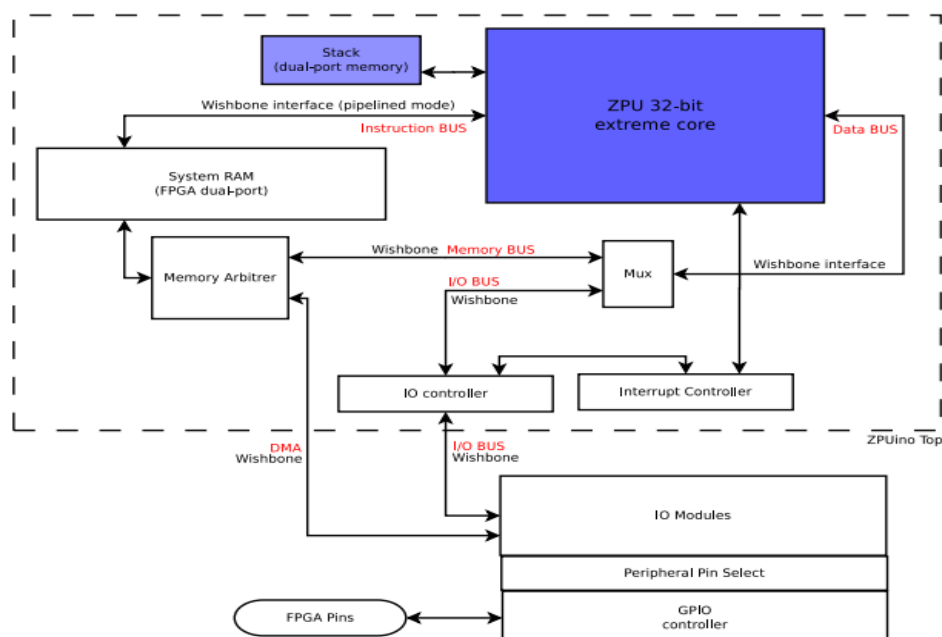
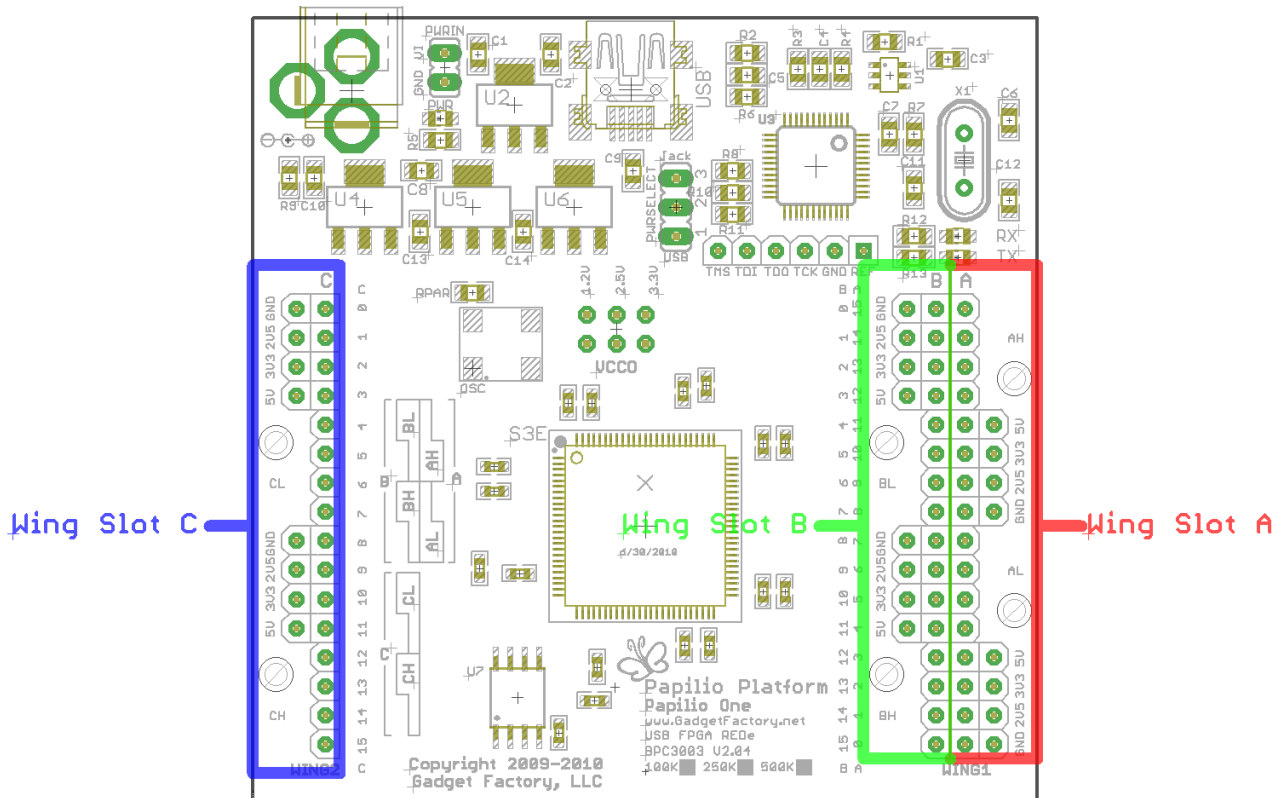
- Papilio one – SPARTAN3E (250 y 500)
- Papilio Pro – SPARTAN6
- Papilio duo – SPARTAN6 y atmega32U4

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



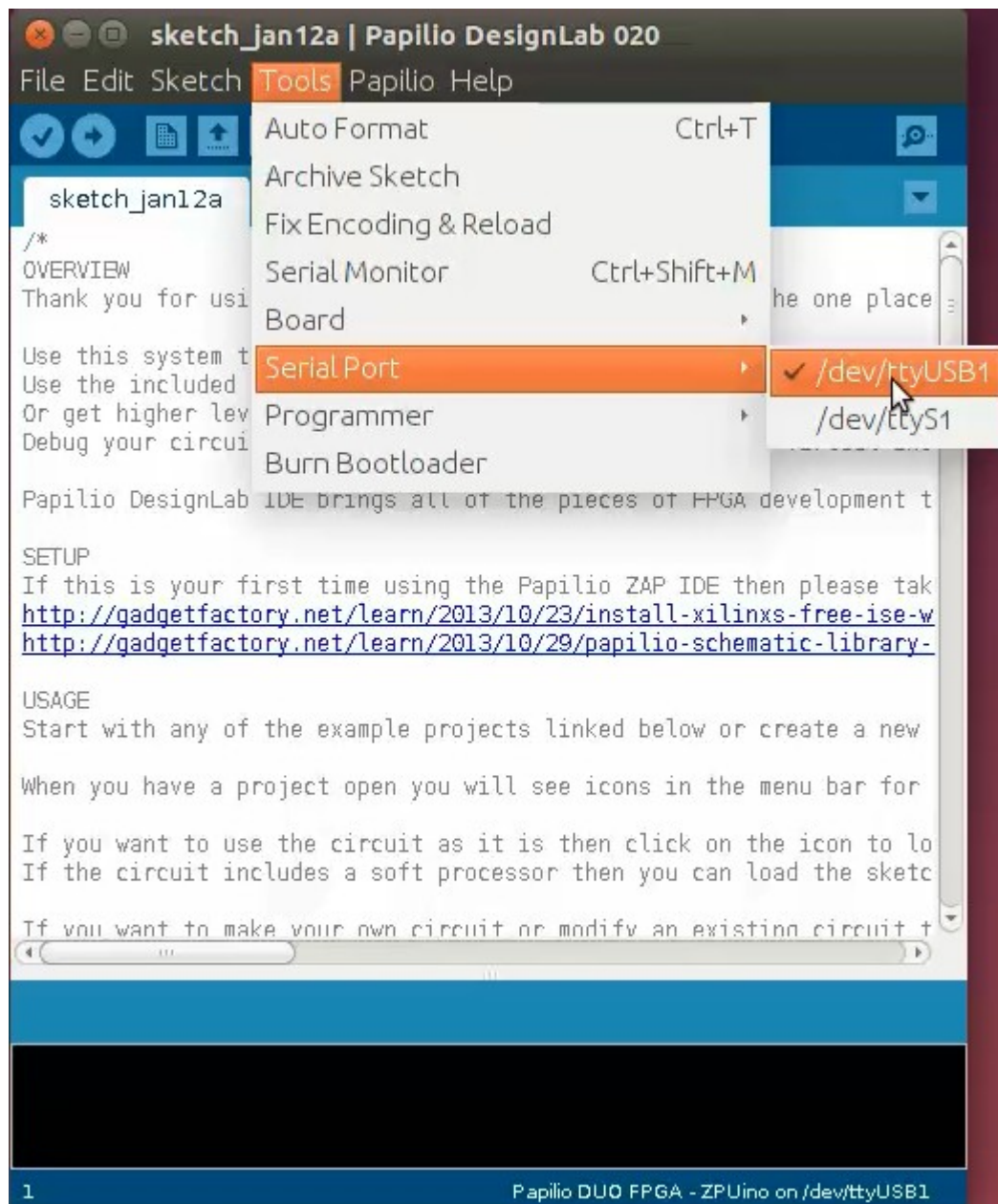
Tanto las especificaciones del microprocesador como el diseño de ZPUino son abiertas

- Al ser diseño soft-core, se pueden añadir nuevos periféricos al SoC según las necesidades (diseño hardware en HDLs)
- Se puede trabajar con ZPUino de modo equivalente a Arduino ya que se ha adaptado el IDE (entorno de desarrollo software de arduino) para ser compatible con ZPUino. Como veremos, el entorno también permitira modificar el SoC a nivel hardware para añadir nuevos periféricos al mismo.



-Entorno de desarrollo:

El entorno DesignLab está basado en Arduino, esto puede verse a simple vista con la interfaz de usuario del entorno.



Aunque añade más opciones y podemos usarlo como veremos a continuación para realizar circuitos más complejos.

-Ejercicio 1:

Usar la placa como inversor

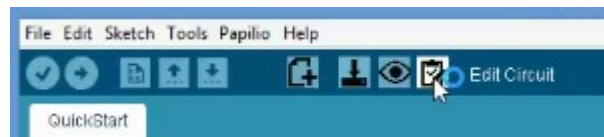
Para realizar este ejercicio seguimos el tutorial:

<http://gadgetfactory.net/learn/2015/05/03/designlab-make-a-simple-fpga-circuit-2/>

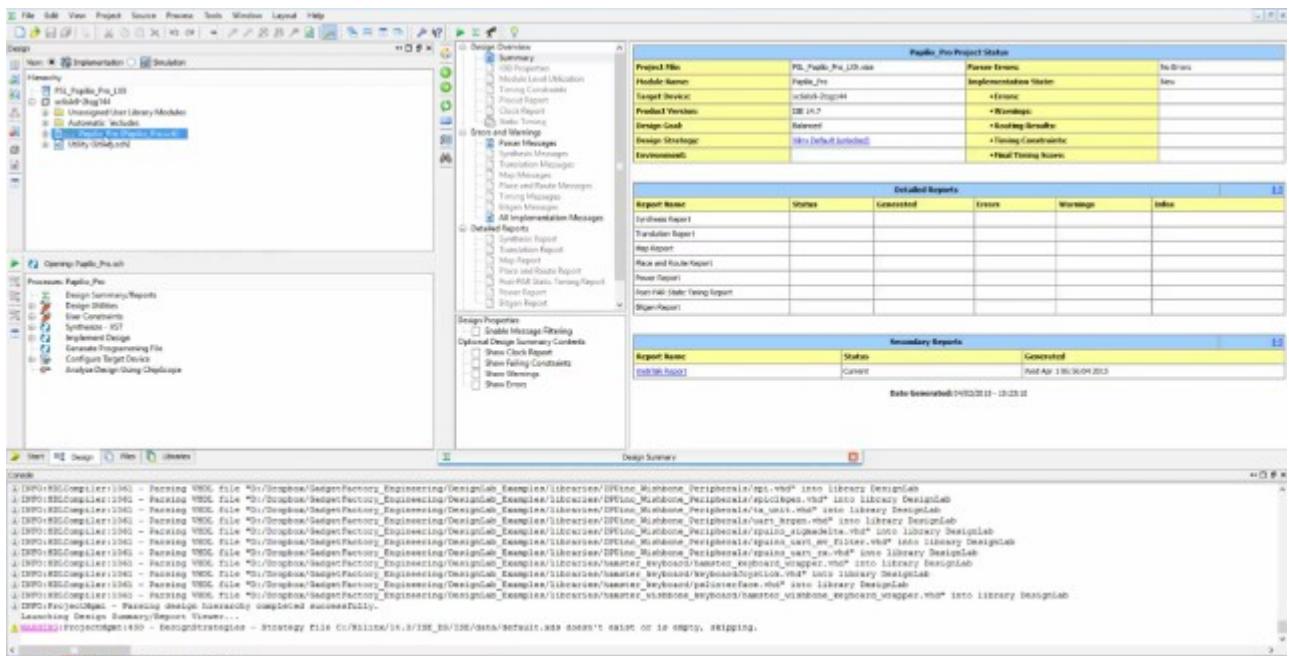
-En primer lugar seleccionamos un nuevo circuito FPGA



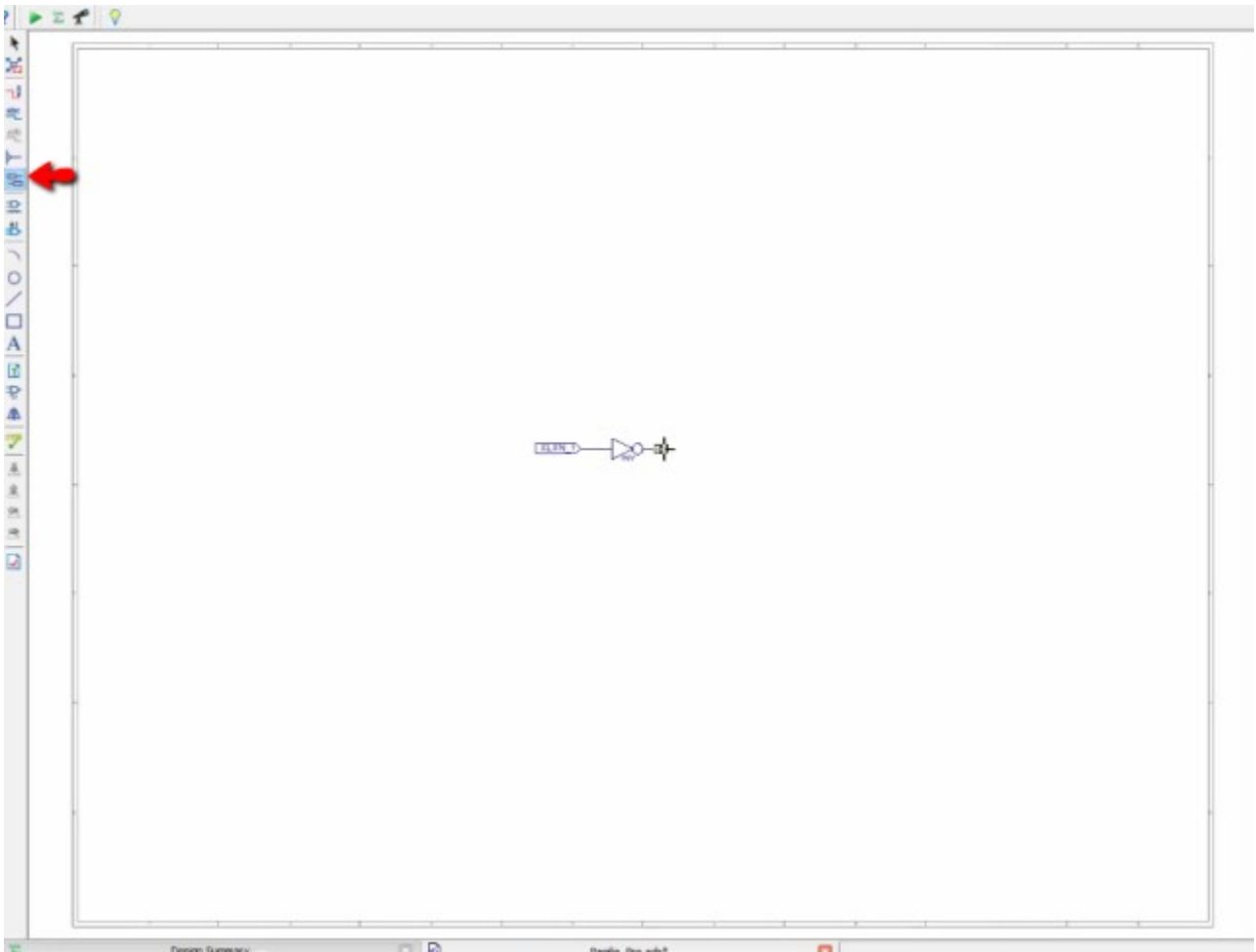
-Cuando el proyecto esté montado clicamos en editar:



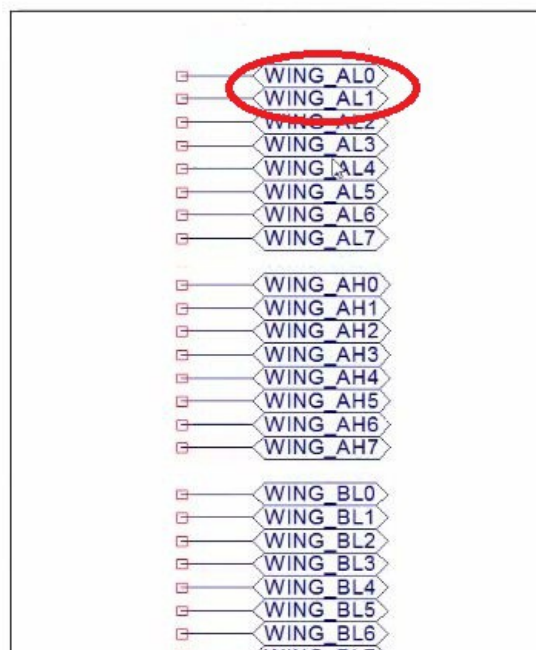
-Esto nos lleva al esquemático donde haremos las modificaciones:



-Buscamos el símbolo de inversor en la lista de componentes y colocamos uno, clicamos donde señala la flecha para conectar las E/S.

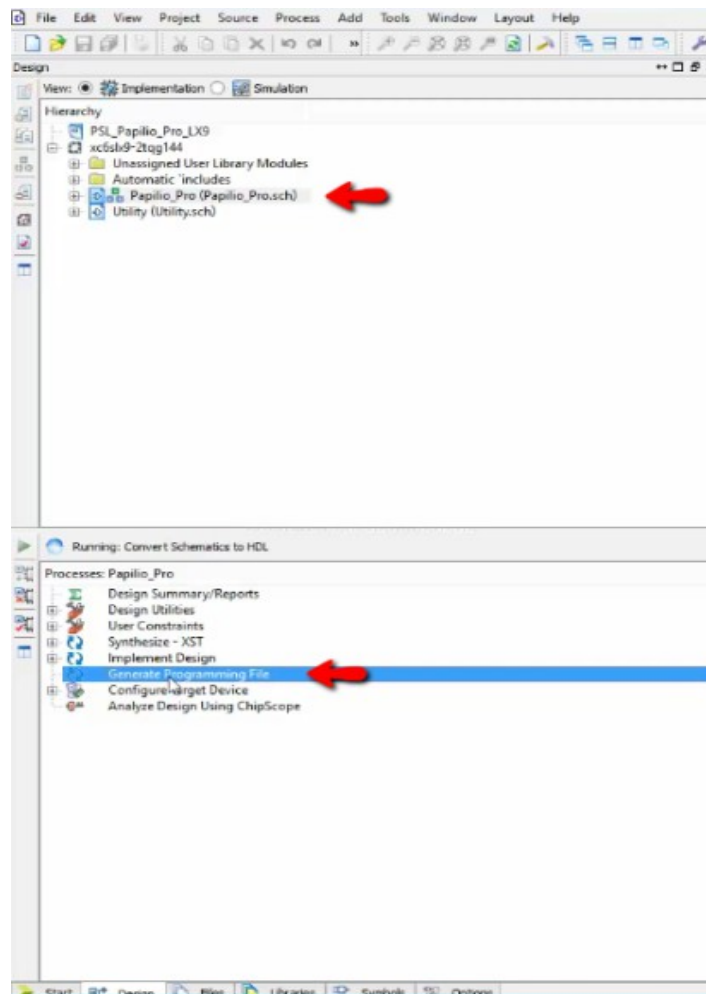


-Asignamos WING_AL0 y WING_AL1.

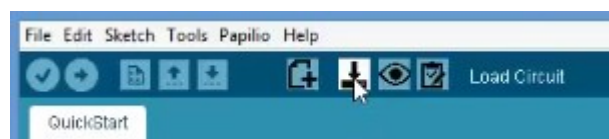


-Click derecho y renombramos los puertos, uno de ellos será el LED.

-Sintetizamos código.



-Cargamos en la placa



Y con esto concluye el tutorial, para comprobar el funcionamiento simplemente debemos conectar un LED con resistencia al pin elegido y un switch, con esto comprobamos el funcionamiento del inversor.

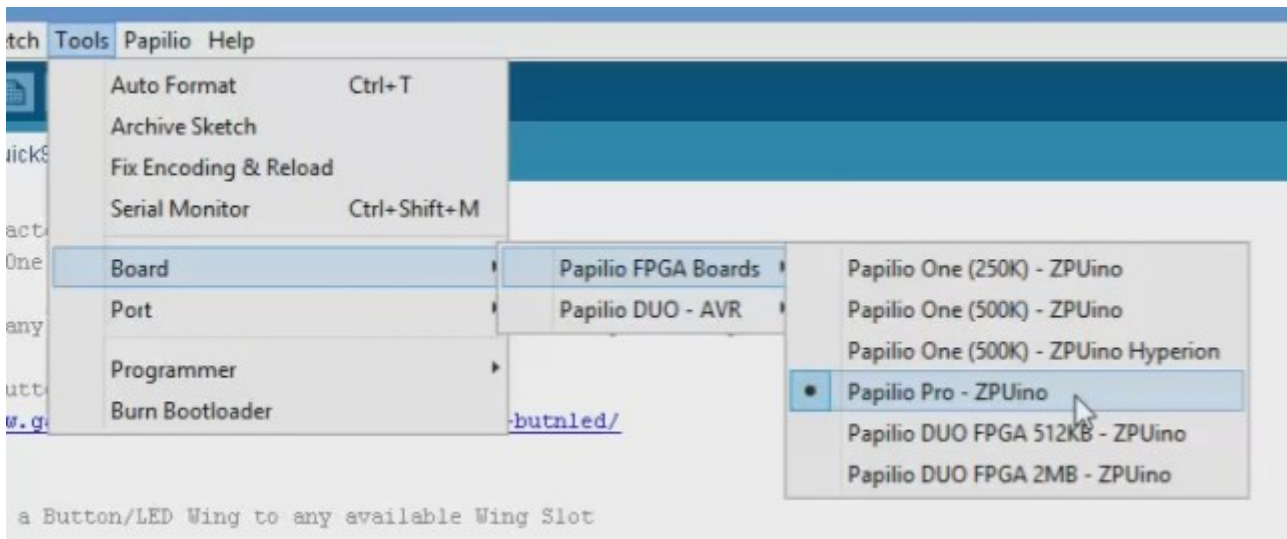
-Ejercicio 2:

Cargar SoC ZPUINO y desarrollar SKETCHES

De nuevo vamos a realizar un tutorial:

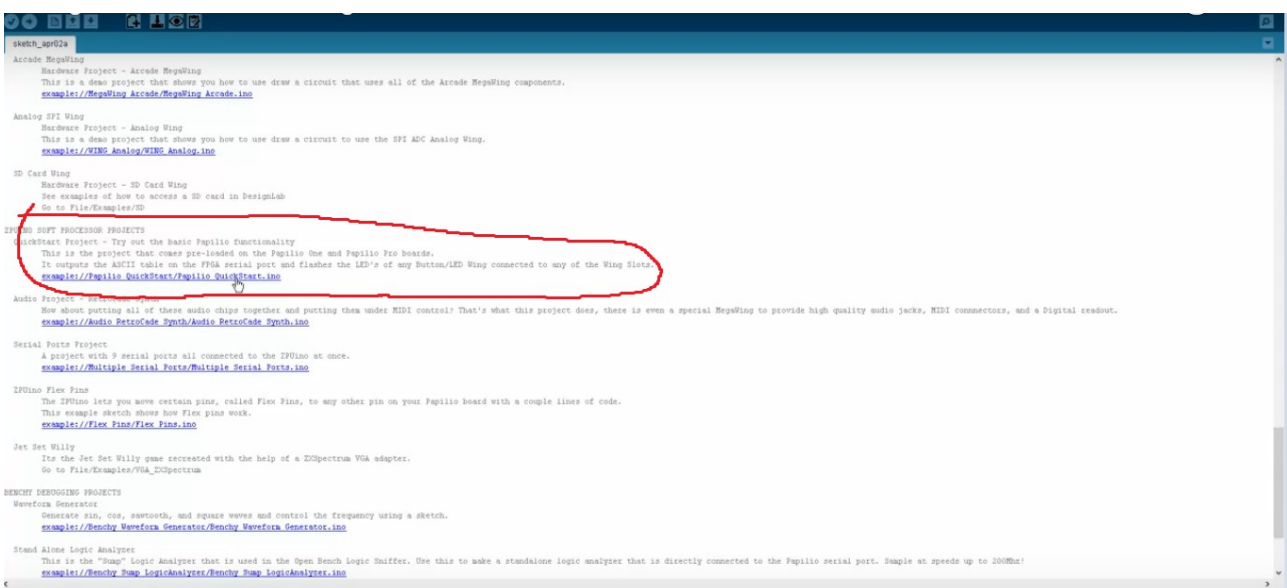
<http://gadgetfactory.net/learn/2015/04/03/designlab-using-the-ide-for-the-first-time/>

-En primer lugar seleccionamos nuestra placa, en nuestro caso la 550k aseca.



-Seleccionamos el puerto también

-Vamos a usar la placa como FPGA por lo que nos vamos a la pestaña correspondiente.



-Código que vamos a ejecutar:

```
Gadget Factory
Papilio One QuickStart Example

Pressing any of the 4 pushbuttons on the Button/LED Wing will light the corresponding LED.

BPM5007 Button/LED Wing Reference:
http://www.gadgetfactory.net/gf/project/bpm5007-buttonled/

Hardware:
* Connect a Button/LED Wing to any available Wing Slot

created 2010
by Jack Gaseport from existing Arduino code snippets
http://www.gadgetfactory.net

This example code is in the public domain.
*/

#define circuit ZPUino_Vanilla

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

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

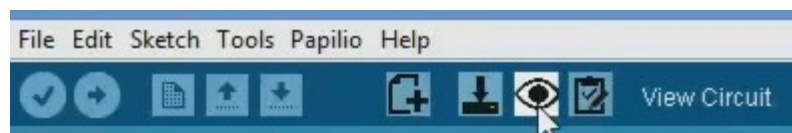
// variables will change:
int buttonState = 0; // variable for reading the pushbutton status
int thisPin;
int ledState = LOW;

// first variable ASCII character '1' is number 33;
int thisByte = 33;
// you can also write ASCII characters in single quotes.
// for example, '1' is the same as 33, so you could also use this:
//int thisByte = '1';

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

-Cargamos el circuito

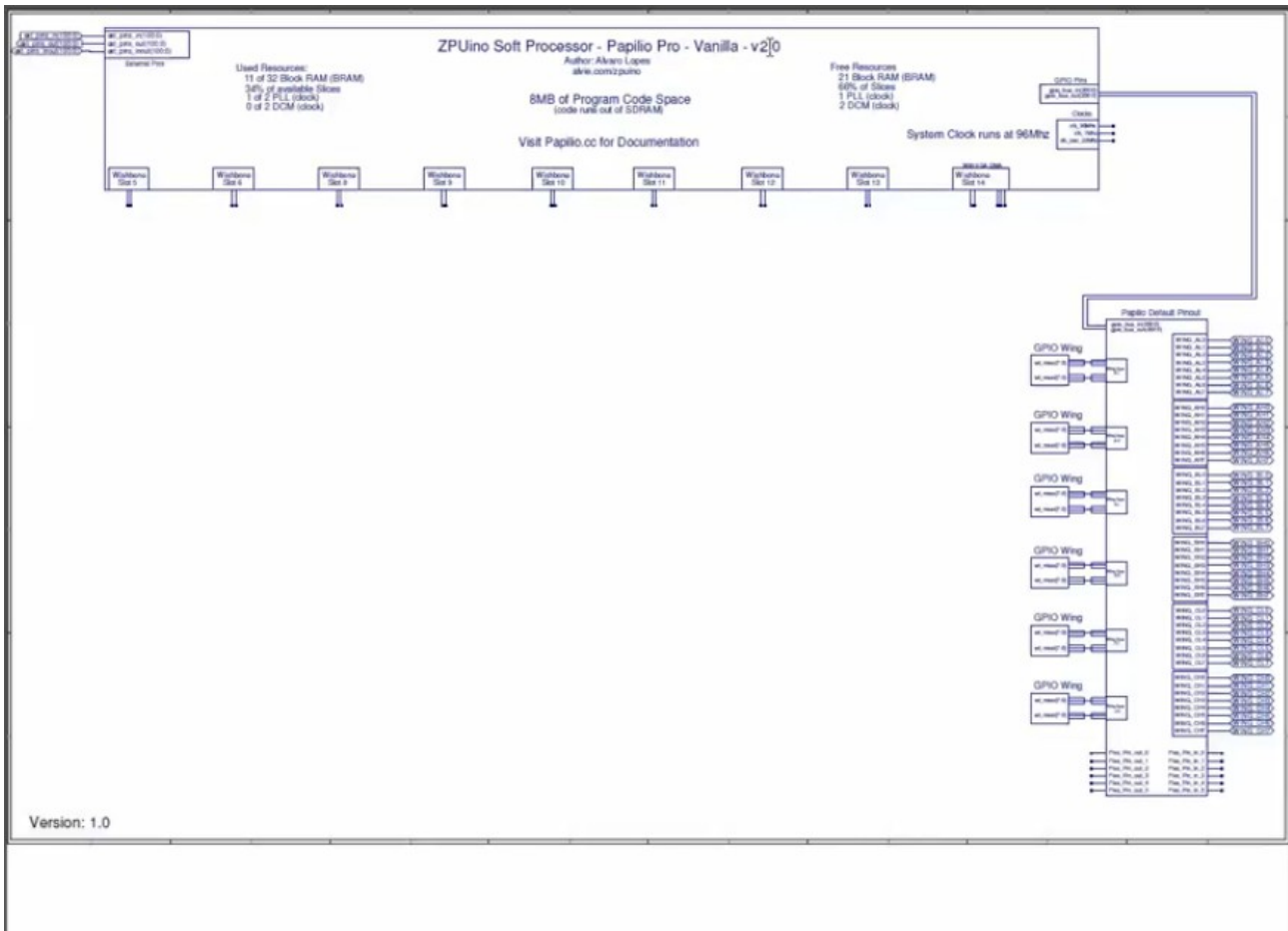
-Y clicamos en ver circuito



-Cargamos el circuito en la placa como en el ejercicio anterior



-Y aparecerá algo como esto:



-Al ejecutar vemos los resultados por pantalla:
Si queremos modificar el código para enviar los datos por consola podemos usar el programa de python que usábamos en arduino:

dec 69,	hex: 45,	oct: 105,	bin: 1000101
dec 70,	hex: 46,	oct: 106,	bin: 1000110
dec 71,	hex: 47,	oct: 107,	bin: 1000111
dec 72,	hex: 48,	oct: 110,	bin: 1001000
dec 73,	hex: 49,	oct: 111,	bin: 1001001
dec 74,	hex: 4A,	oct: 112,	bin: 1001010
dec 75,	hex: 4B,	oct: 113,	bin: 1001011
dec 76,	hex: 4C,	oct: 114,	bin: 1001100
dec 77,	hex: 4D,	oct: 115,	bin: 1001101
dec 78,	hex: 4E,	oct: 116,	bin: 1001111
dec 79,	hex: 4F,	oct: 117,	bin: 1001111
dec 80,	hex: 50,	oct: 120,	bin: 1010000
dec 81,	hex: 51,	oct: 121,	bin: 1010001
dec 82,	hex: 52,	oct: 122,	bin: 1010010
dec 83,	hex: 53,	oct: 123,	bin: 1010011
dec 84,	hex: 54,	oct: 124,	bin: 1010100
dec 85,	hex: 55,	oct: 125,	bin: 1010101
dec 86,	hex: 56,	oct: 126,	bin: 1010110
dec 87,	hex: 57,	oct: 127,	bin: 1010111
dec 88,	hex: 58,	oct: 130,	bin: 1011000
dec 89,	hex: 59,	oct: 131,	bin: 1011001
dec 90,	hex: 5A,	oct: 132,	bin: 1011010
dec 91,	hex: 5B,	oct: 133,	bin: 1011011
dec 92,	hex: 5C,	oct: 134,	bin: 1011100
dec 93,	hex: 5D,	oct: 135,	bin: 1011101
dec 94,	hex: 5E,	oct: 136,	bin: 1011110
dec 95,	hex: 5F,	oct: 137,	bin: 1011111
dec 96,	hex: 60,	oct: 140,	bin: 1100000
dec 97,	hex: 61,	oct: 141,	bin: 1100001
dec 98,	hex: 62,	oct: 142,	bin: 1100010
dec 99,	hex: 63,	oct: 143,	bin: 1100011
dec 100,	hex: 64,	oct: 144,	bin: 1100100
dec 101,	hex: 65,	oct: 145,	bin: 1100101
dec 102,	hex: 66,	oct: 146,	bin: 1100110
dec 103,	hex: 67,	oct: 147,	bin: 1100111
dec 104,	hex: 68,	oct: 150,	bin: 1101000
dec 105,	hex: 69,	oct: 151,	bin: 1101001
dec 106,	hex: 6A,	oct: 152,	bin: 1101010
dec 107,	hex: 6B,	oct: 153,	bin: 1101011
dec 108,	hex: 6C,	oct: 154,	bin: 1101100
dec 109,	hex: 6D,	oct: 155,	bin: 1101101
dec 110,	hex: 6E,	oct: 156,	bin: 1101110
dec 111,	hex: 6F,	oct: 157,	bin: 1101111
dec 112,	hex: 70,	oct: 160,	bin: 1110000
dec 113,	hex: 71,	oct: 161,	bin: 1110001
dec 114,	hex: 72,	oct: 162,	bin: 1110010
dec 115,	hex: 73,	oct: 163,	bin: 1110011
dec 116,	hex: 74,	oct: 164,	bin: 1110100
dec 117,	hex: 75,	oct: 165,	bin: 1110101
dec 118,	hex: 76,	oct: 166,	bin: 1110110
dec 119,	hex: 77,	oct: 167,	bin: 1110111
dec 120,	hex: 78,	oct: 170,	bin: 1111000
dec 121,	hex: 79,	oct: 171,	bin: 1111001
dec 122,	hex: 7A,	oct: 172,	bin: 1111010

```
import serial

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

print("Starting!")

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

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

y en el programa de Papilio simplemente modificar la línea que determina el valor del pin por:

```
if(Serial.read() == 'L'){
    pinMode(ledUsado, LOW);
}

if(Serial.read() == 'H'){
    pinMode(ledUsado, HIGH);
}
```

Y con esto concluye el segundo ejercicio.

-Ejercicio 4:

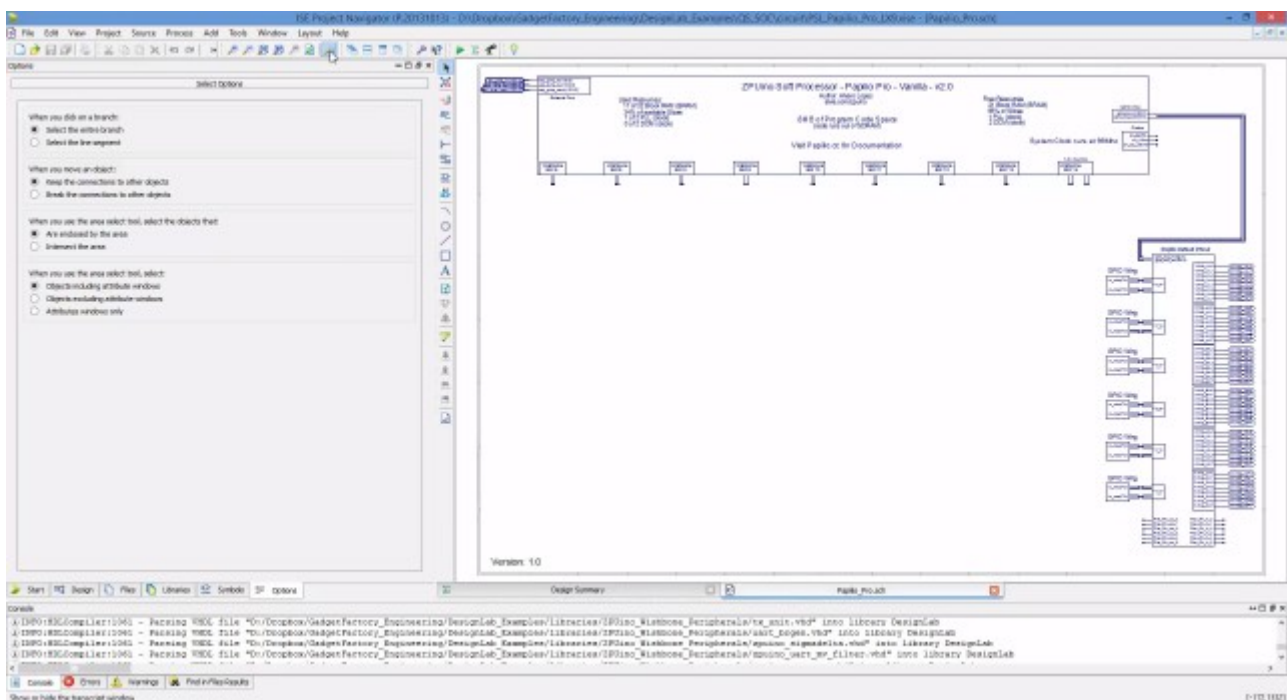
Rediseñando el SoC. Añadiendo periféricos

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

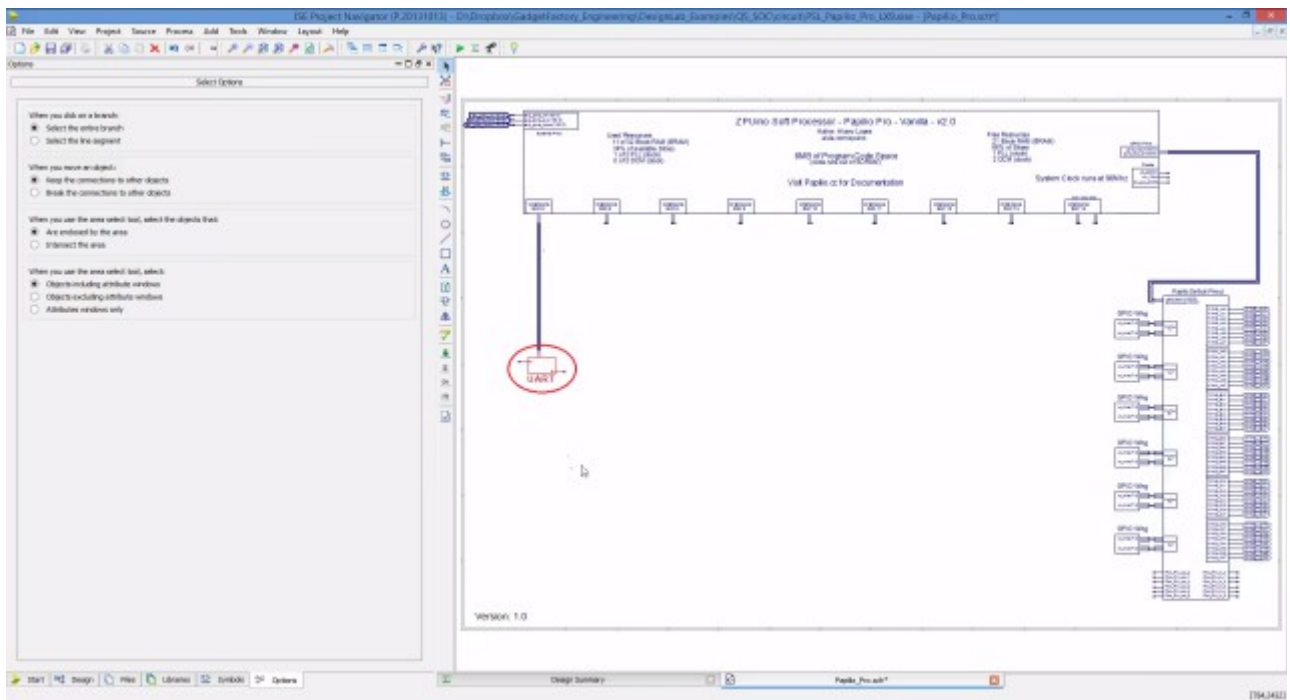
Desarrollamos el tutorial:

<http://gadgetfactory.net/learn/2015/05/15/designlab-make-a-custom-zpuino-system-on-chip-2/>

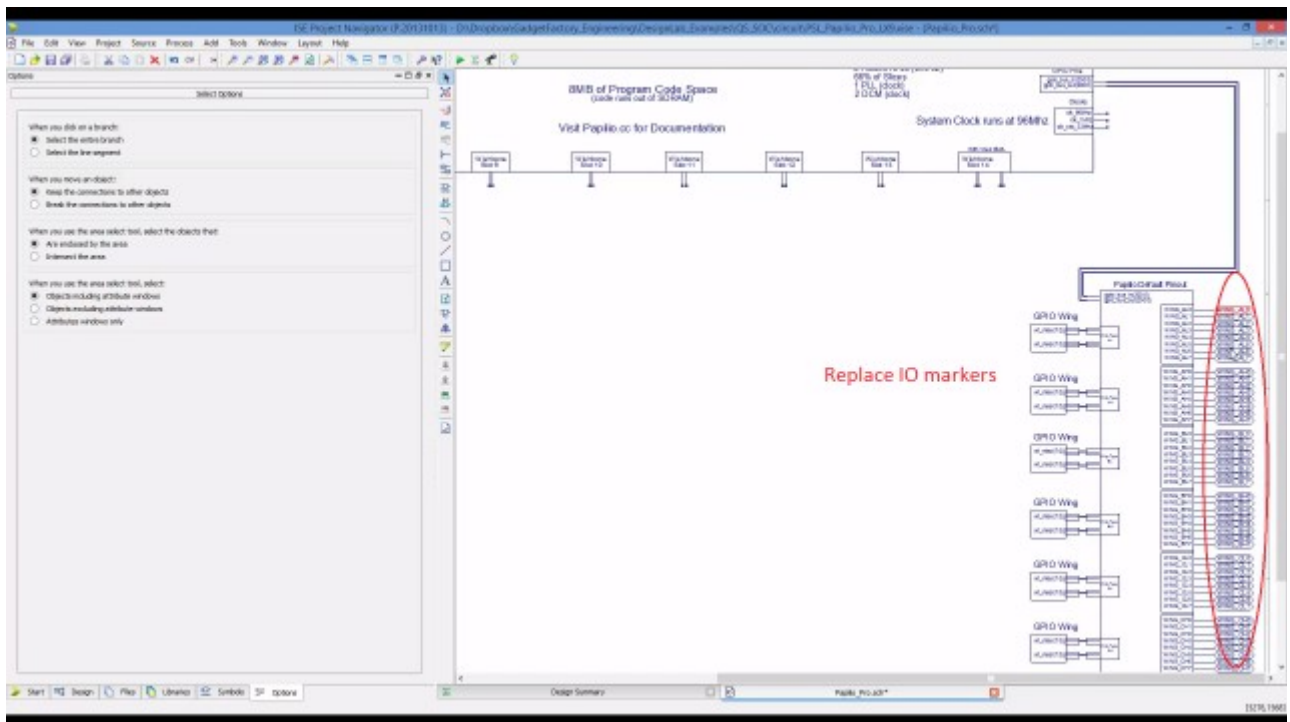
-Al igual que en los tutoriales anteriores creamos un proyecto FPGA y vamos al editor del circuito:



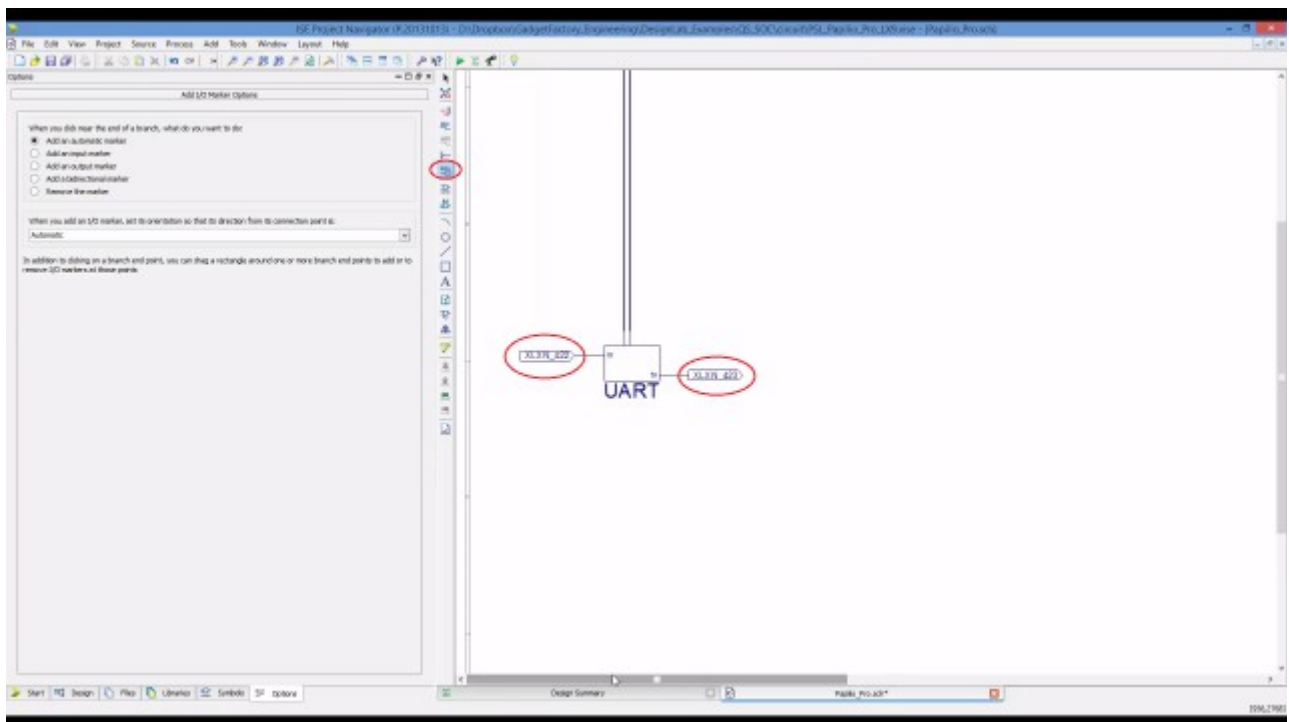
-Añadimos un UART



-Borramos los 2 primeros E/S

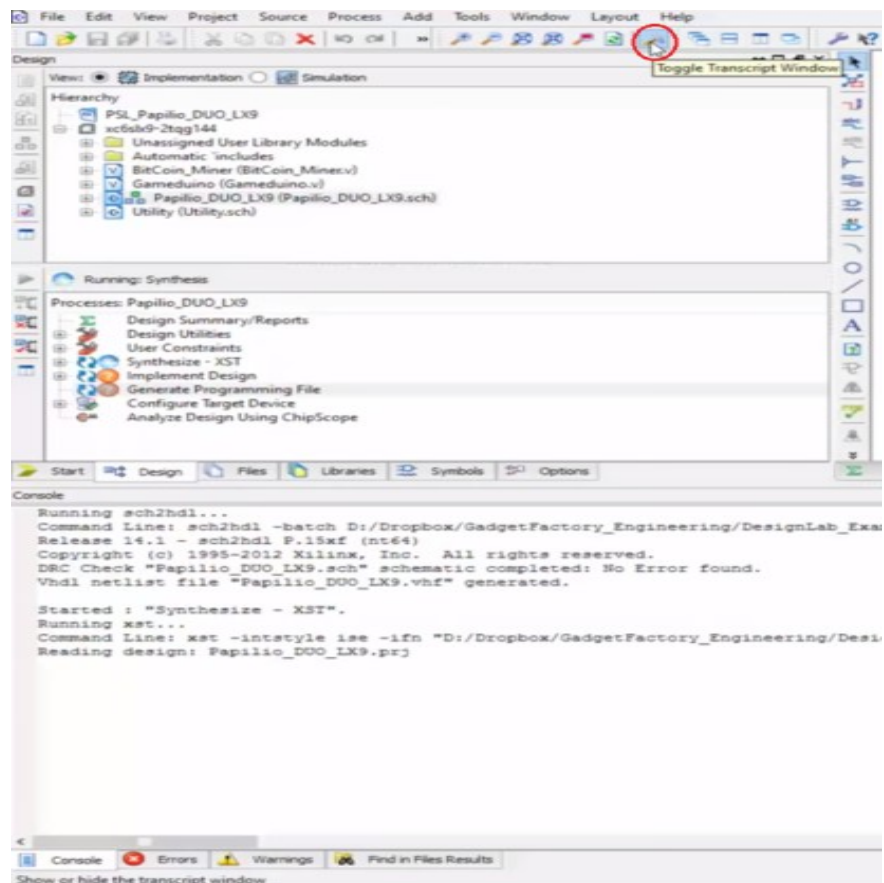


-Las I/O que hemos borrado las ponemos en el UART



-Las renombramos

-Sintetizamos el circuito



-Añadimos estas líneas al código

[illegible]

-Cargamos el circuito, lo compilamos y cargamos en la placa

Con esto concluye el tutorial, ahora para comprobar que funciona debemos hacer uso del clable TTL-RS232-USB



Lo conectamos al pc y hacemos uso del circuito de control de LED y de nuevo del código python del puerto serie para comprobar que efectivamente funciona.

Plataforma Raspberry Pi

-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

-Introducción a Raspberry PI :

Raspberry Pi es un computador de placa reducida, computador de placa única o computador de placa simple (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.



	Raspberry Pi 1 Modelo A	Raspberry Pi 1 Modelo B	Raspberry Pi 1 Modelo B+	Raspberry Pi 2 Modelo B	Raspberry Pi 3 Modelo B
SoC: ⁵	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB) ³			Broadcom BCM2836 (CPU + GPU + DSP + SDRAM + Puerto USB)	Broadcom BCM2837 (CPU + GPU + DSP + SDRAM + Puerto USB)
CPU:	ARM 1176JZF-S a 700 MHz (familia ARM11) ³			900 MHz quad-core ARM Cortex A7	1.2GHz 64-bit quad-core ARMv8
Juego de instrucciones:	RISC de 32 bits				
GPU:	Broadcom VideoCore IV, ⁶¹ OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), ⁵⁹ 1080p30 H.264/MPEG-4 AVC ³				
Memoria (SDRAM):	256 MiB (compartidos con la GPU)	512 MiB (compartidos con la GPU) ⁴ desde el 15 de octubre de 2012		1 GB (compartidos con la GPU)	
Puertos USB 2.0: ⁵⁵	1	2 (vía hub USB integrado) ⁵⁴	4		
Entradas de vídeo: ⁶²	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF				
Salidas de vídeo: ⁵	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), ⁶³ Interfaz DSI para panel LCD ^{64 65}				
Salidas de audio: ⁵	Conector de 3.5 mm, HDMI				
Almacenamiento integrado:	SD / MMC / ranura para SDIO		MicroSD		
Conectividad de red: ⁵	Ninguna	10/100 Ethernet (RJ-45) vía hub USB ⁵⁴			10/100 Ethernet (RJ-45) vía hub USB ⁶⁶ , Wifi 802.11n, Bluetooth 4.1
Periféricos de bajo nivel:	8 x GPIO, SPI, I2C, UART ⁶¹			17 x GPIO y un bus HAT ID	
Reloj en tiempo real: ⁵	Ninguno				
Consumo energético:	500 mA, (2.5 W) ⁵	700 mA, (3.5 W)	600 mA, (3.0 W)	800 mA, (4.0 W)	
Fuente de alimentación: ⁵	5 V vía Micro USB o GPIO header				
Dimensiones:	85.60mm x 53.98mm ⁶⁷ (3.370 x 2.125 inch)				
Sistemas operativos soportados:	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux, SUSE ⁶⁸ Enterprise for ARM System . RISC OS ²				

-Entorno:

Para desarrollar los ejercicios vamos a instalar Ubuntu-MATE en una tarjeta de memoria que irá a la raspberry pi, que trabajará como pc totalmente funcional.

Ubuntu MATE es una distribución Linux basada en Ubuntu. Está mantenida por la comunidad y es un derivado de Ubuntu oficialmente reconocido por Canonical, usando el entorno de escritorio MATE.



Requisitos [\[editar \]](#)

Ubuntu MATE actualmente soporta la arquitectura ARM, x86 y x64. Para activar los efectos de escritorio se necesita una GPU compatible.

Requisitos de hardware	Mínimos	Recomendados
Microprocesador	Pentium III 750 MHz	Core 2 Duo 1,6 GHz
Memoria RAM	512 MB	2 GB
Disco duro (espacio libre)	8 GB	16 GB
Resolución	1024 × 768 o superior	1366 × 768 o superior

Configurar ubuntu-mate:

- ◆ Seguir los pasos en el arranque configurando idioma, teclado, localización, fecha y hora
- ◆ IMPORTANTE: guardar información sobre usuario y passwd elegidos.
- ◆ Configuraremos la red manualmente:

- Desactivar network-manager
- Editar /etc/network/interfaces:
iface <nombre_InterfazDeRed> 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
- Ejecutar sucesivamente:
\$ sudo ifdown <nombre_InterfazDeRed>
- \$ sudo ifup <nombre_InterfazDeRed>
- Probar conexión: \$ ping www.google.es

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

También se puede realizar el ajuste del tamaño del sistema de ficheros al tamaño.

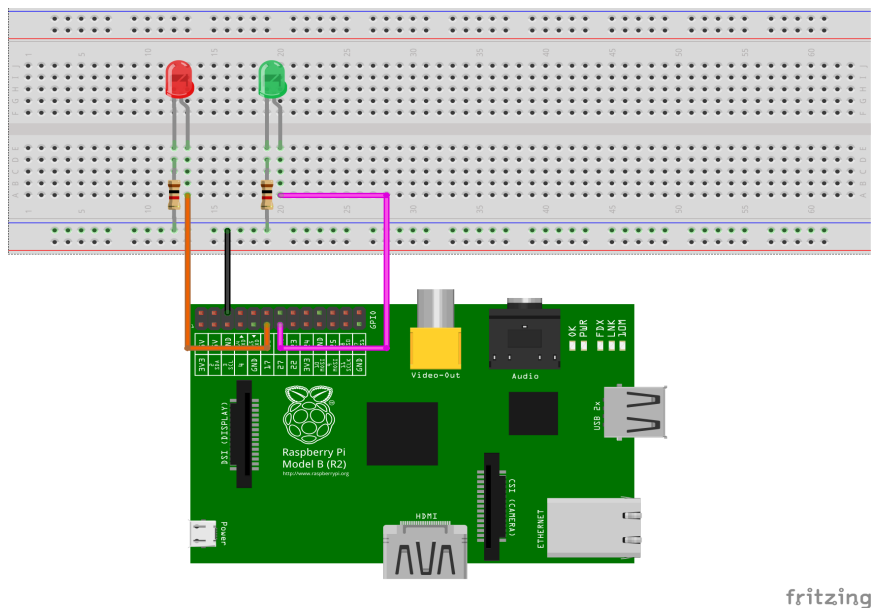
-Ejercicio 1:

Manejo de GPIOs desde línea de comandos

.Circuito:

El ejercicio consistirá en manejar LEDS conectados a la placa mediante GPIO con un sencillo programa en python.

Montamos el siguiente circuito, con la raspberry, es muy sencillo, el único detalle a tener en cuenta es que la placa no especifica los pines, hay que irse a un esquemático para saber con que pin estamos trabajando.



A continuación vamos a crear un programa en python:

```
sudo nano blink.py
```

Importamos la librería GPIO y declaramos pines:

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

Creamos una función que ejecute el bucle de blinkeo

```
def blink():
    print "Ejecucion 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 "Ejecucion finalizada"
    GPIO.cleanup() ## Hago una limpieza de los GPIO
```

Llamamos a la función:

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

Y ejecutamos el código Python:

```
sudo python blink.py
```

Y así terminaría el ejercicio, observaríamos como los dos LED conectados a la Rpi parpadearían repetidamente.

-Ejercicio 2 y 3:

Servidor Web GPIO + Temperatura

.Circuito:

Ahora nuestra intención es usar la Rpi como servidor Web de forma que podamos encender LED o apagarlos mediante una conexión red.

Para ello vamos a usar FLASK, el web framework de Python para convertir la Rpi en un servidor dinámico.

Para instalar FLASK:

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

Instalar las dependencias del FLASK:

```
pi@raspberrypi ~ $ sudo pip install flask
```

Y ya está, podemos probarlo con un código como este:

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

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

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

Este código carga el script de FLASK, crea un objeto FLASK y ejecuta el código que continúa.

Para ejecutar el server:

```
pi@raspberrypi ~ $ sudo python hello-flask.py
```

Para mandar información HTML con formato debemos usar los templates como haremos a continuación:

Haremos el programa weblamp.py mostrado a continuación:

```

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) //capturamos el serial de ARDUINO
# Create a dictionary called pins to store the pin number, name, and pin state:
pins = {
    24 : {'name' : 'LED1', 'state' : GPIO.LOW},
    25 : {'name' : 'LED2', 'state' : GPIO.LOW}
}

temperatura = arduino.readline() //Leemos el valor del serial

# 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:

    temperatura = arduino.readline()

    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    # Put the pin dictionary into the template data dictionary:
    templateData = {
        'pins' : pins,
        'temperatura' : temperatura
    }
    # 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):
    temperatura = arduino.readline()

    # 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)

```

```

#pone el mensaje en el diccionario del template
templateData = {
    'message': message,
    'pins': pins,
    'temperatura': temperatura #dato de la temperatura
}

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

@app.route("/temp")
def temperatura():
    # Para cada pin lee el valor y lo guarda
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    temperatura = arduino.readline()

    # pone los PINS en el diccionario
    templateData = {
        'pins': pins,
        'temperatura': temperatura
    }
    # Le pasa el diccionario al HTML
    return render_template('main.html', **templateData)

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

```

En este código creamos rutas y podremos ver la información enviada mediante código 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 %}

    <h1>Temperatura</h1>
    <p>{{ temperatura }}</p>

    <h1>Luz</h1>

    <form method="post">
    <input type="button" value="Actualizar Página" onclick="window.location.reload()" />
    </form>

</body>
</html>

```

En este código HTML Usamos los datos del diccionario de templates que tenemos

para hacer uso de la información. Al ejecutar podremos ver en la dirección de nuestra Rpi algo como lo que sigue:

Device Listing and Status

```
{% for pin in pins %}
The {{ pins[pin].name }} {% if pins[pin].state == true %} is currently on (turn off) {% else %} is currently off (turn on) {% endif %}
{% endfor %} {% if message %}
{{ message }}
{% endif %}
```

Temperatura

```
{{ temperatura }}
```

Luz

[Actualizar PÁgina](#)

Nota: Al no estar usando la Rpi para hacer la captura, el HTML no puede coger los datos del diccionario

Para hacer uso del sensor de temperatura podemos reutilizar el código arduino de usamos en las prácticas de la placa:

```
int val_Adc = 0;
float temp = 0;
const int Pin_TMP36 = A1;

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

void loop()
{
    int val_Adc = analogRead(Pin_TMP36);
    float voltage = (val_Adc / 1024.0) * 5.0;
    float temp = (voltage - .5) * 100;
    Serial.print(voltage, 4);
    Serial.print(" voltios    ");
    Serial.print(temp, 2);
    Serial.println(" Grados");
    delay( 500 );
}
```

De esta forma los datos del sensor se muestran en el serial y podemos captarlos como se muestra en el código Python.