

# **MEMORIA PRACTICAS DE LABORATORIO DE DESARROLLO HARDWARE**

**Escuela Técnica Superior de  
Ingeniería Informática  
Grado en Ingeniería de Computadores**



**Alumno:**

**Manuel Jesús Gómez Rodríguez**

**Tutorado por:**

**Manuel Jesús Bellido Díaz**

## **INTRODUCCIÓN**

Esta memoria describe parte de las prácticas realizadas en el bloque 1 de la asignatura LDH (Laboratorio de desarrollo hardware) del cuarto curso del grado en Ingeniería Informática de Computadores.

Se citan algunas descripciones técnicas de componentes utilizados así como los montajes realizados y el código creado o importado para el correcto funcionamiento de los circuitos. Constan de tres partes correspondientes a las tres plataformas desarrolladas: plataforma Arduino, Zpuino y Raspberry.

## **CONTENIDO DE LA MEMORIA**

### 1. PLATAFORMA ARDUINO

#### 1.1 OBJETIVOS

#### 1.2 INTRODUCCION A ARDUINO

##### 1.2.1 HISTORIA

##### 1.2.2 OPEN HARDWARE

##### 1.2.3 ANALISIS PLACA ARDUINO

#### 1.3 PRACTICAS DE PLATAFORMA ARDUINO

##### 1.3.1 PRACTICA 1

##### 1.3.2 PRACTICA 2

##### 1.3.3 PRACTICA 3

##### 1.3.4 PRACTICA 4

##### 1.3.5 PRACTICA 5

##### 1.3.6 PRACTICA 6

##### 1.3.7 PRACTICA 7

### 2. PLATAFORMA ZPUINO

#### 2.1 OBJETIVOS

#### 2.2 INTRODUCCION A PAPILIO

#### 2.3 PRACTICAS DE PLATAFORMA ZPUINO

##### 2.3.1 PRACTICA 1

##### 2.3.2 PRACTICA 2

##### 2.3.3 PRACTICA 3

##### 2.3.4 PRACTICA 4

## 3. PLATAFORMA RASPBERRY PI

### 3.1 OBJETIVOS

### 3.2 INTRODUCCION A RASPEBERRY PI

### 3.3 PRACTICAS PLATAFORMA RASPBERRY PI

## **PARTE 1: PLATAFORMA ARDUINO**

### **1.1 OBJETIVOS**

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

El objetivo práctico consistirá en saber reutilizar el software que han desarrollado otros y aprovecharlo de forma idónea para nuestro proyecto.

Este código ajeno podremos encontrarlo de diversas formas y provenir de distintas fuentes que irán desde la página del fabricante a alguna otra que adjunte código de otros desarrolladores.

Al software le acompañará documentación que será de mucha ayuda y que, según de donde provenga, poseerá una claridad, calidad y complejidad que será de suma importancia.

Utilizaremos pues código ajeno como base y lo adaptaremos a nuestros propósitos y necesidades de forma que realizaremos los ejemplos que se nos proponen.

Por tanto va a ser más una labor de análisis y adaptación del software existente que no el escribir muchas líneas en una pantalla en blanco.

### **1.2 INTRODUCCION A ARDUINO**

Arduino es una placa de circuito impreso con la que, junto con unos componentes electrónicos, un microcontrolador y una serie de pines de entrada y salida, podemos crear proyectos basados en sistemas electrónicos.

Mediante un lenguaje de alto nivel programaremos el microcontrolador para que ejecute las acciones que deseamos llevar a cabo.

Por otro lado Arduino es un proyecto *OPEN-HARDWARE*. Todos los esquemas de la placa, diseños y componentes son públicos. Esto implica que podremos desarrollar una placa propia basada en Arduino. El hecho de que sea de dominio público no implica que no tenga ningún tipo de licencia. La licencia que emplea Arduino es del tipo GPL.

En cuanto a su tecnología, Arduino está basado inicialmente en microcontroladores de 8 bits AVR aunque con alguna versión basada en ARM de 32 bits ( Arduino DUE).

Los AVR son una familia de microcontroladores RISC del fabricante estadounidense ATMEL y en el caso concreto de ARDUINO UNO R3 el modelo es el ATMEL328P de que es sencillo y de bajo coste.

Podemos encontrar en el mercado versiones más potentes de Arduino con diversas configuraciones.

## 1.2.1 HISTORIA

Arduino fue inventado en el año 2005 por el entonces estudiante del instituto IVRAE Massimo Banzi, en un principio, por una necesidad de aprendizaje para los estudiantes de computación y electrónica del mismo instituto.

El primer prototipo de Arduino fue fabricado en el instituto IVRAE. Inicialmente estaba basado en una simple placa de circuitos eléctricos, donde estaban conectados un microcontrolador simple junto con resistencias de voltaje, además de que únicamente podían conectarse sensores simples como leds u otras resistencias, y es más, aún no contaba con el soporte de algún lenguaje de programación para manipularla.

Años más tarde, se desarrolló un entorno para la programación del procesador de esta placa y posteriormente se agregaron puertos USB para poder conectarla a un ordenador.

Un breve tiempo más tarde, al ver los grandes resultados que tuvo Arduino y las grandes aceptaciones que tuvo por parte del público, comenzó a distribuirse en Italia, después en España, hasta colocarse en el número uno de herramientas de aprendizaje para el desarrollo de sistemas autómatas, siendo además muy económica (300-500 pesos) en comparación con otras placas de microcontroladores (800 pesos en adelante).

En 2015 ha habido una división en el equipo de Arduino :

Web inicial: [www.arduino.cc](http://www.arduino.cc) ----> Cuatro de los cinco fundadores

Nueva web: [www.arduino.org](http://www.arduino.org) ----> Un fundador

Cada rama tiene IDEs diferentes y placas nuevas distintas

## 1.2.2 OPEN HARDWARE

Arduino es una placa Open Hardware. Esto quiere decir que todos los esquemas y diseños de dominio público.

De esta forma cualquiera puede comprar sus componentes, descargar esquemas y construir su propia placa compatible con Arduino.

Otro aspecto que se debe tener en cuenta es que el software que se requiere para programar la placa Arduino mediante interfaz USB también es público y gratuito, pudiéndose descargar de la web de Arduino.

## 1.2.3 ANALISIS DE LA PLACA ARDUINO

Analicemos la placa ARDUINO UNO R3. En la placa se pueden distinguir:

**\*BOTON DE RESET** Permite realizar el reinicio de la placa. Una vez reiniciada vuelve a ejecutar el programa que tiene grabado.

**\*CONECTOR USB** Se emplea para comunicar la placa con el PC. También se usa para conectarnos con Arduino a través del monitor serie.

**\*CONEXION 7v-12v** Mediante un Jack de 2,1 mm alimentamos a la placa con un rango de tensión comprendido entre 7 y 12 voltios. Una tensión que suele ir bien es la que proporciona una pila de petaca de 9v.

**\*PINES DE ALIMENTACIÓN.** Hay 6 pines que se detallan:

**IOREF** Es un pin de referencia del voltaje al que tendrá que trabajar el microcontrolador. También se usa para cuando conectamos un shield a Arduino regulando la tensión para un correcto funcionamiento.

**RESET** Este pin tiene la misma función que el botón RESET. Aquí lo encontramos para poder resetear la placa desde un pulsador externo.

**3,3 V** Este pin proporciona 3,3 voltios.

**5 V** Este pin proporciona 5 voltios para dispositivos, sensores, etc...

**GND** Aquí se conectarán los terminales de masa de los componentes electrónicos

**Vin** Este terminal permite alimentar Arduino de la misma forma que se hace con el Jack de 2,1 mm.

### **\*PINES ANALOGICOS**

Terminales que se emplean para conectar la placa con el exterior, conectando sensores que proporcionan señales analógicas.

Se podrán configurar como entrada o salida.

Arduino dispone de seis pines de este tipo.

**\*MICROCONTROLADOR Atmega328P** Este circuito integrado es el cerebro de la placa. Es el encargado de ejecutar las instrucciones de los programas.

**\*INDICADOR TX-RX** Indica que Arduino se está comunicando vía serie con el PC. Cuando esto ocurre los indicadores parpadean alertando

de la transmisión y recepción de información.

**\*CONECTORES ICSP** Se utilizan cuando se desea programar Arduino desde un entorno diferente del IDE y de la conexión típica por USB.

Para realizar esta operación se requiere un programador externo que ira conectado a los conectores mencionados. Si se desea programar Arduino de este modo se ha de hacer en lenguaje ensamblador o en C.

**\*INDICADOR DE ENCENDIDO** Mediante una luz verde indicara que Arduino está alimentado correctamente y listo para funcionar.

**\*INDICADOR DE CARGA** Parpadea cuando se carga un programa la placa.

**\*PINES DIGITALES** Terminales que comunican con el exterior a sensores digitales. Como los analógicos se podrán configurar como entrada o salida. Arduino tiene 14 pines de este tipo. Dentro de este conjunto existe uno llamado AREF que proporciona el voltaje de referencia para los pines analógicos.

**\*MICROCONTROLADOR ATMEGA 328P.CARACTERISTICAS:**

Modelo : 328.Microcontrolador de 8 bits

Voltaje de funcionamiento : 5 voltios

Memoria flash 32 KB

Memoria SRAM : 2 KB

Memoria EEPROM : 1 KB

Velocidad de proceso : 16 Mhz

La comunicación del microcontrolador con el PC se realiza mediante comunicación serie, es decir que los 8 bits llegan a Arduino de uno en uno, ya que el microcontrolador trabaja con grupos de 8 bits, por lo que dispone de otro circuito integrado soldado a la placa llamado UART para adecuar la llegada de los bits al microcontrolador.

La velocidad es de 16Mhz, es decir, que puede procesar 16.000.000 de instrucciones en un segundo o ciclo.

Podemos encontrar el 328P en dos formatos: formato DIP que viene introducido en un zócalo o formato SMD que viene soldado a la placa.

De la memoria flash se puede decir que es donde se almacenan los programas del usuario. Esta memoria esta compartida con u gestor de arranque, que incorpora las instrucciones necesarias para que Arduino esté listo para usar .La cantidad compartida es de 0.5 KB.

La memoria SDRAM es la encargada, entre otros, de almacenar los datos resultantes de la ejecución de instrucciones del programa.

Por último la EEPROM es solo lectura. En ella van grabadas las librerías necesarias para interpretar los programas de Arduino.

## **\*COMUNICACION ARDUINO-PC**

El problema que podemos advertir cuando vemos que la transmisión de datos es en serie (un bit tras otro) y el microcontrolador necesitar grupos de 8 bits para procesar. Para ello la placa posee como ya se ha indicado un circuito integrado soldado en placa que une el conector hembra USB con el microcontrolador. Este circuito llamado UART (Transmisor-Receptor Asíncrono Universal) es el encargado de adecuar y gestionar los bits según se necesiten en serie o en grupos de 8.

## **\*CONEXION PC-ARDUINO Y CONFIGURACION DEL IDE**

La conexión es bastante sencilla .Solo se necesita un cable tipo USB tipo A- tipo B. El conector tipo B irá conectad a la placa y el tipo A al PC.

Con el programa abierto procedemos a interconectar las dos partes con el cable mencionado. Una vez conectados el sistema operativo detecta el USB .Normalmente en los IDE actuales los drivers se instalan en el momento en que se está instalando el IDE. Una vez detectado el dispositivo se han de seguir estos pasos:

Primero nos cercioraremos de que la placa que tiene configurado el IDE es la que poseemos. Para ello vamos a Herramientas->Placa y desplegaremos el menú para seleccionarla. Segundo desde el mismo menú Herramientas iremos a Puerto Serie y seleccionamos el COM asignado.

Una vez hecho esto la placa está perfectamente reconocida por el IDE.

El IDE tiene como funciones más importantes:

**Botón VERIFICAR** Después de escribir u programa es aconsejable revisar los posibles errores. En caso de que los tuviera el IDE de Arduino lo muestra en la ventana inferior.

**Botón CARGAR** Permite cargar el programa ya escrito y corregido sintácticamente, al microcontrolador.

**Botón NUEVO** Genera un nuevo Sketch.

**Botón Abrir** Abre una ventana de dialogo mostrando la carpeta por defecto donde el programa guarda los sketches. Permite recuperar los proyectos anteriores.

**Botón GUARDAR** Como su nombre indica permite guardar los sketches en el directorio indicado.

**Botón MONITOR SERIE** Abre una ventana en la que podremos observar el valor que van adquiriendo las variables o para interaccionar con Arduino.

Para realizar cualquiera de estas dos acciones se deben introducir las órdenes necesarias.

## **\*LIBRERIAS**

Son un conjunto de funciones e instrucciones que hacen que un dispositivo se pueda vincular a Arduino de forma más sencilla.

Al instalar Arduino también se instalan unas librerías que vienen por defecto.

Tenemos dos tipos de librerías:

Las que incluye el IDE y las que son de contribución. Estas últimas son desarrolladas y distribuidas por usuarios para facilitar la programación. Normalmente tienen licencia GPL.

Para instalar las librerías debemos:

Descargar la librería. Normalmente en ZIP.

Descomprimir la librería en la carpeta Libraries

Abrir el IDE DE Arduino y acceder al menu Sketch

Clicar Add Library y seleccionar la librería

## **1.3 PRACTICAS DE PLATAFORMA ARDUINO**

- 1.-Controlar el encendido/apagado de un LED desde el PC vía puerto serie. Usar un programa en Python que envíe comandos de encendido y apagado a través del puerto serie.
- 2.- Encendido y apagado de un LED mediante pulsador utilizando interrupciones.
- 3.- Control de la posición de un servomotor con un potenciómetro.
- 4.- Control de la velocidad de giro de un motor de continua con un potenciómetro.
- 5.-Impresión en una pantalla LCD.
- 6.- Empleando el sensor de temperatura tmp36GZ diseñar un termómetro en una LCD.
- 7.-Contador 00 a 59 en un display 7 segmentos cada segundo.

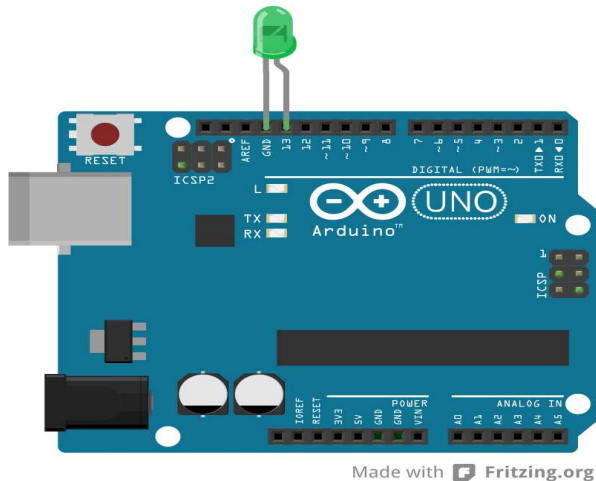
### **1.3.1 Práctica 1**

Usaremos la placa para encender y apagar un LED conectado al pin 13.

Para ello procederemos como indica el tutorial propuesto.

Se coloca un LED con el ánodo en el terminal 13 y el cátodo en el pin GND.





A continuación se arranca el IDE en el PC y se conecta la placa. Tras comprobar que se detecta el hardware, seleccionamos el puerto que corresponda. En el momento que no existan errores sintácticos se procede a cargar el programa a la placa.

Tras ello y ya que estamos usando un sistema operativo Ubuntu tecleamos en el terminal:

```
sudo apt-get install python-serial
```

para instalar la librería serial .

Previamente se habrá guardado en un archivo llamado RaspDuino.py (por ejemplo) el código que se adjunta más tarde y se procede:

```
python RapsDuino.py
```

para ejecutarlo .

Aparecerá en pantalla la petición de introducción de comando y se podrá cambiar el estado del Led con la pulsación de “H” o “L”.

### CÓDIGO USADO

```
int led = 13;

void setup (){
    pinMode(led,OUTPUT); //led 13 como salida
    Serial.begin(9600); //Inicializo puerto serie
}

void loop (){
    if (Serial.available()) {
        char c = Serial.read();
        if (c == 'H') {
            digitalWrite(led,HIGH);
        }else if (c == 'L'){
            digitalWrite(led,LOW);
        }
    }
}
```

## Memorias prácticas LDH

---

En este primer ejemplo se pueden observar algunos de los principales comandos fundamentales. Estos son básicamente:

- pinMode(pin, INPUT/OUTPUT): Configura los pines de la placa.
- digitalWrite(pin, valor): Cuando el pin sea digital y esté configurado OUTPUT para transferirle un valor.
- digitalRead: El valor que lee esta función de un sensor se suele almacenar en una variable definida por el usuario.
- analogWrite: Cuando el pin sea analógico y esté configurado como OUTPUT se usará esta función.
- analogRead: Para almacenar en una variable un valor desde un pin analógico configurado como INPUT.
- delay(): Usada para retrasar la ejecución de la siguiente instrucción.

Otro aspecto a tener en cuenta es la importación de la librería Serial que se usa en la comunicación por puerto serie de Arduino con el PC o los componentes.

Por otro lado el código en Python seleccionado aparece a continuación y posibilita al PC comunicarse con Arduino.

### ARCHIVO PYTHON

```
import serial

arduino = serial.Serial('/dev/ttyACM0', 9600) # Poner el puerto correcto en nuestro caso

print("Comienzo")

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
```

### **CONCLUSIONES:**

La primera práctica ha servido como toma de contacto con las placas de desarrollo a través de Arduino Uno. Hemos podido comprobar también como se maneja su IDE. Mediante el uso de éste hemos conocido algunas estructuras básicas y propias de los sistemas empuotrados que se ejecutan en modo stand-alone (no dependen del usuario) y hemos importado librerías preexistentes. Además se han usado los puertos I/O para manejar componentes (en este caso un LED) y aprendido a nombrarlos y controlarlos. En definitiva un primer repaso del conjunto de conceptos asociados a los microcontroladores.

### 1.3.2 Práctica 2

De nuevo trataremos de controlar el encendido o apagado de un LED pero esta vez usando un pulsador que active una interrupción en un pin.

Una interrupción la podemos definir como una llamada al microcontrolador, el cual, deja lo que está ejecutando y atiende dicha llamada. Esa llamada lleva al microcontrolador a otra parte del código que debe ejecutarse con mayor prioridad.

Una vez ejecutado este trozo de código, el microcontrolador vuelve al punto anterior, es decir, a la línea de la instrucción donde lo había dejado antes de la interrupción.

Arduino UNO tiene dos pines para crear interrupciones (el 2 y el 3).

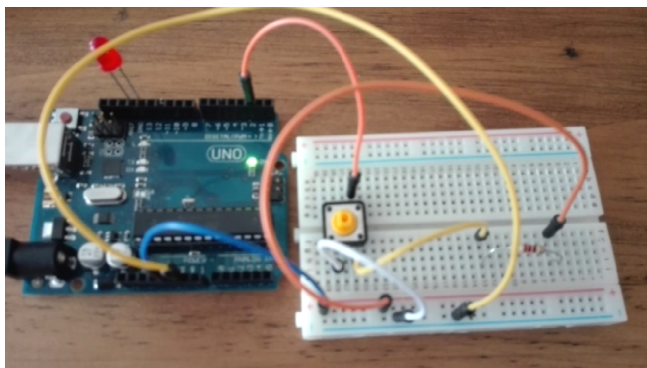
Hay tres tipos de interrupciones: un timer, una interrupción hardware y una llamada software.

En este ejemplo usaremos un evento de hardware para hacer saltar la interrupción, como es la pulsación del botón.

Disponemos para ello los siguientes elementos:

- Placa Arduino
- Protoboard
- Cable USB
- 1 LED.
- 1 pulsador
- 1 resistencia (220 OHM o 330 OHM)
- Cables para conexión

El montaje realizado sobre la protoboard es el que se muestra a continuación:



## Memorias prácticas LDH

---

Una vez montada la práctica procederemos a conectar la placa al PC y seleccionar el puerto correcto.

Al ejecutar el código en el IDE el LED reacciona al pulsar el botón pero se puede apreciar un funcionamiento errático. Esto es debido a que hay un rebote en la señal.

### CÓDIGO PARA ARDUINO

```
int boton = 2;
int led = 13;
int valor = LOW;

void setup (){
  pinMode (boton,INPUT);
  pinMode (led,OUTPUT);
  Serial.begin(9600);
  attachInterrupt(0,cambio,FALLING);
}

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

void cambio (){
  valor=!valor;
  delay(500);
}
```

Puede observarse que hemos usado una función que genera la interrupción deseada.

La función `attachInterrupt(nº int,nombre_función,modo)` usa los parámetros:

- nº int: numero de la interrupción.
- nombre\_función: nombre que le hemos dado a la función interrupción que queremos llamar.
- modo: define cuando ha de ser activada. Puede ser CHANGE, LOW, RISING o FALLING segun se quiera activar en un cambio de nivel, cuando el valor del pin es LOW, cuando el valor sube o cuando desciende.

### **CONCLUSIONES:**

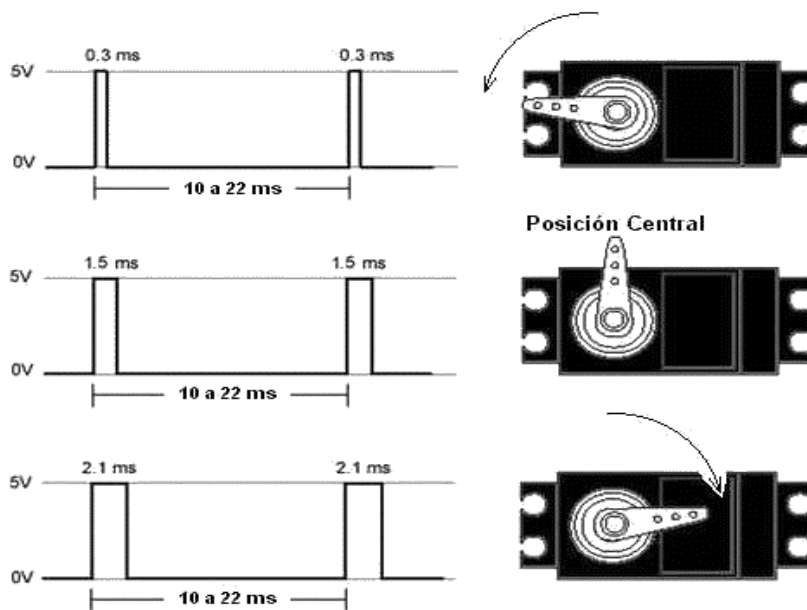
Si durante la ejecución de un `delay()`, por ejemplo, un sensor detecta un valor nuevo este se perderá y una forma de evitarlo sería con el uso de una interrupción. La segunda práctica añade el uso de interrupciones como medio de evitar una pérdida de información por parte del microcontrolador mientras está ocupado ejecutando algo.

## 1.3.3 Práctica 3

La práctica 3 consiste en ejercer el control de la posición sobre un servomotor. Lo haremos indicando el ángulo de posicionamiento en el PC. Enviaremos un valor entre 0 y 180 ° por el puerto serie usando un script escrito en python a tal efecto.

### SERVOS

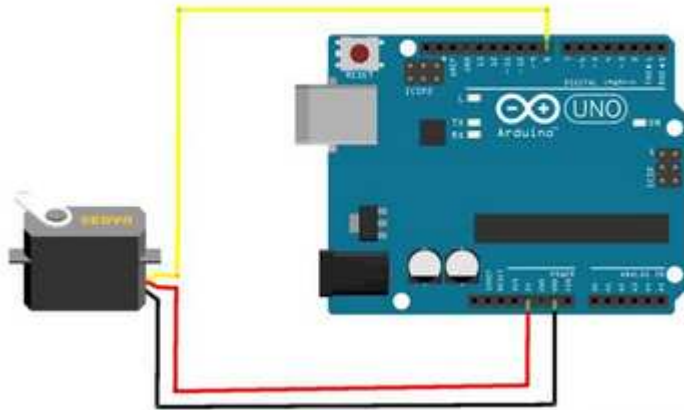
Los servos de modelismo están compuestos por un pequeño motor de DC y un controlador de posición integrado, al que sólo hay que especificarle en qué posición (ángulo) ha de situarse. Estos dispositivos son muy fáciles de usar, ya que el controlador interno se encarga de posicionar el servo con alta precisión, y además contiene la circuitería necesaria para proporcionar la potencia necesaria al motor de DC en su interior. Nosotros sólo hemos de transferirle la posición en que queramos situar el servo, para ello usan señales de PWM con periodos de entre 50/60Hz, codificándose la posición del servo en el tiempo en alto de la señal PWM. Nótese que la información no reside estrictamente en el duty-cycle de la señal PWM, sino en el tiempo en alto de esta.



Los componentes necesarios son:

- Placa Arduino
- Protoboard
- Cable USB
- Un servomotor

Montaje:



### CÓDIGO PARA ARDUINO

```
#include <Servo.h>

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

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

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

Se puede apreciar que hemos usado la librería servo.h. Esta biblioteca permite que Arduino controle los servomotores RC. La biblioteca Servo admite hasta 12 motores en la mayoría de las placas Arduino y 48 en el Arduino Mega. En otras tarjetas que no sea el Mega, el uso de la biblioteca deshabilita la funcionalidad de analogWrite () (PWM) en los pines 9 y 10. En el Mega, pueden usarse hasta 12 servos sin interferir con la funcionalidad de PWM. El uso de 12 a 23 motores deshabilitará PWM en los pines 11 y 12.

## CÓDIGO PYTHON

```
import serial
serie=serial.Serial('/dev/ttyACM0',115200) //indicar el puerto
print("Comienzo")
while True:
    angle = raw_input('Introduce ángulo:')
    serie.write(angle)
serie.close()    #Finalizamos la comunicacion
```

## CONCLUSIONES:

Esta vez el dispositivo que hemos controlado vía puerto serie ha sido un servo de modelismo. Se escribe el dato al puerto del mismo modo pero esta vez el dato lo recoge la las funciones de la librería servo para así desde el pin adecuado recoger un valor que se tomará como ángulo por el dispositivo que girará en consecuencia.

### 1.3.4 Práctica 4

Vamos a mover un motor de CC con la ayuda de Arduino.

El motor de continua es un dispositivo que convierte la energía eléctrica en energía electromotriz. Esta energía se destina, mediante engranajes externos, a producir movimiento. El motor CC funciona sobre la base de unos imanes que crean campos magnéticos opuestos y esto hace que su eje interno gire, produciendo movimiento.

Los motores CC no tienen polaridad, a no ser que deseemos cambiar el sentido del giro del rotor o eje. Esto quiere que cambiando la polaridad de la tensión de entrada, cambiamos el sentido de la rotación.

Estos motores necesitan de un regulador de velocidad con Arduino para poder ser utilizados en proyectos. Esta electrónica adicional suele venir en el shield o mediante algún controlador. Componentes que usamos:

- Placa Arduino
- Protoboard
- Cable USB
- Un servomotor
- Potenciometro
- Motor DC
- L293D
- Fuente alimentación externa

## 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.

### 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.

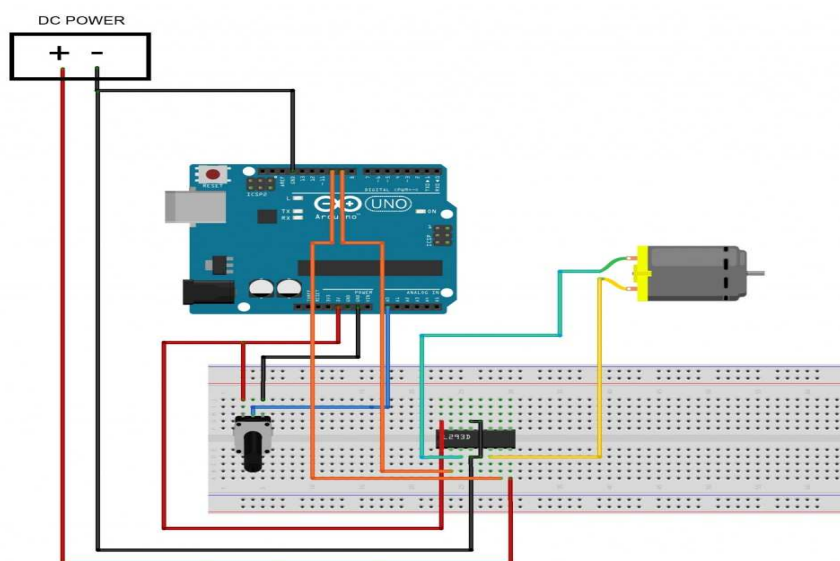
El L293D es un integrado para controlar motores DC que usa el sistema puente en H. ¿Qué es el puente en H? Es un sistema para controlar el sentido de giro de un motor DC usando cuatro transistores. En la imagen vemos que los transistores se comportan como interruptores y dependiendo que transistores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

El L293D tiene dos puentes H y proporciona 600mA al motor y soporta un voltaje entre 4,5V y 36V tal y cómo pone en el datasheet.

### CONTROL DE LA VELOCIDAD A TRAVÉS DEL PWM

Hasta este punto sabemos cómo controlar el sentido de giro del motor DC a través del L293D. Pero ¿y la velocidad de giro? En este proyecto lo que haremos es controlar la velocidad y el giro del motor con un solo potenciómetro.

Para hacerlo utilizaremos el PWM. Sabemos que hay que atacar los pins 2 y 7 del L293D desde dos salidas a cada una. Pero tenemos que invertir un PWM.





## CÓDIGO PARA ARDUINO

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

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

void setup()
{
    //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, la 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);
}
```

## **CONCLUSIONES:**

La lectura analógica del potenciómetro nos ha servido como entrada para manejar el sentido de giro y la intensidad de un motor DC. Nos ha servido esta práctica como introducción del concepto de PWM (modulación por ancho de pulso) como técnica para regular el ciclo de trabajo de una señal periódica.

## 1.3.5 Práctica 5

En esta práctica dotamos a nuestro sistema de la capacidad de representar texto y números mediante una pantalla LCD. Usaremos para ello el modelo LCM1602C, basada en el controlador Hitachi HD44780 y mediante el uso de la librería LiquidCrystal podremos enviar cadenas de texto desde nuestro PC vía puerto serie.

### DISPLAYS DE CRISTAL LÍQUIDO (LCD).

Los displays LCD están compuestos por un cristal líquido retenido entre dos placas transparentes conductoras. Dicho cristal cambia su estado de visualización dependiendo del potencial a que se somete dichas placas, estas deben estar divididas en pequeñas celdas controlables independientemente para poder activar cada elemento que compone el visualizador.

Según se puede deducir de esta descripción básica, la primera característica que destaca de los LCDs es su bajo consumo, las placas conductoras se comportan como condensadores que se cargan o descargan. La segunda característica a resaltar es la posibilidad de miniaturización; en la actualidad hay televisores portátiles del tamaño de un reloj de pulsera con pantallas LCDs en color.

El "cristal líquido" del que están compuestos los LCDs puede actuar de dos formas diferentes: cambiando la reflectividad o la transparencia. En el primer caso es la luz que incide y se refleja sobre la pantalla la que permite ver la información, en el segundo es la luz que pasa a través de la pantalla la que descubre el mensaje. Todos los LCD usan ambos métodos de visualización, pero dependiendo de cuál es el principal se puede distinguir dos tipos de LCD: de iluminación frontal y retroiluminados.

La tensión que se debe aplicar a cada una de las placas conductoras no puede ser continua (posibilidad de rotura), debe ser una onda cuadrada, si las dos placas tienen dicha señal en fase el "cristal líquido" es transparente, si no están en fase es opaco. Por tanto, en principio, para gobernar un display LCD se necesitarían dos señales para cada elemento activo (celdilla). Lo normal es que el control de cada celdilla y de las señales necesarias se lleve a cabo por un controlador específico que independice completamente la visualización de la aplicación que se pretenda desarrollar. En la mayoría de los casos se encuentran el controlador y la pantalla LCD integrado en una sola pieza, de forma que basta una "interface hard" simple y unos pocos comandos para gobernar completamente un visualizador, por muy compleja que sea la pantalla LCD.

A nivel académico los LCD se dividen en :

#### -Segmentado

Proviene de los de 7 segmentos a los que se añadieron más para visualizar algunos símbolos. Llevan unas tablas que describen como representar cada símbolo. Cada patilla controla un segmento.

#### -Alfanumérico

Formado por grupos de puntos en forma de tejas. Lleva un elemento inteligente para no tener que usar muchas patillas del microcontrolador.

## -Gráficos o matriciales

Su pantalla es como una matriz de puntos. Los hay con controlador interno o sin él . Si no lo tiene el propio controlador lo sustituye. Si no hay memoria interna usa la externa del controlador .Serán más fácil de usar las interfaces sin llevan controlador.

Esta gran diversidad de visualizadores LCDs podría hacer pensar que también hay una gran variedad de controladores. Sin embargo, en lo que se refiere a displays alfanuméricos de uso general, se ha impuesto un sistema controlador desde hace varios años, de forma que en la actualidad es prácticamente un "estándar": el Hitachi HD44780 (forma sistema con el HD44100). Este circuito puede gobernar una gran diversidad de LCDs, en principio desde pantallas de 1 línea por 16 caracteres hasta de 2 líneas por 40. En esta práctica se usará un display alfanumérico retroiluminado (el LM091LN, de 2 líneas por 16 caracteres de 5x7 "dots"), controlado por el HD44780.

## **EL LCD LCM1602C Y EL CONTROLADOR HD44780**

El HD44780 dispone de 192 caracteres en ROM, más la posibilidad de definir otros 8 por parte del diseñador (en RAM). Dispone de dos tipos de RAM: una donde se almacena el carácter que se visualiza en cada posición del LCD (DD RAM) y otra que sirve para definir nuevos símbolos (CG RAM). La cantidad de memoria para la visualización de caracteres es de 80 bytes, de forma que cada posición del display se asocia a una posición de memoria en la DD RAM, por tanto un display como el usado en la práctica de 2x16, sólo usa una parte de la DD RAM, siendo la restante memoria inútil para la visualización (a no ser que se produzcan desplazamientos que pasen los datos a la parte visible). De forma general siempre la primera línea empieza en la posición \$80 (realmente 00, el bit 7 no forma parte de la dirección de celda) y la segunda, si la hubiese, en la \$C0 (realmente \$40). Si fuese un display de 4 líneas por 20, la tercera línea empezaría después de la primera y la cuarta después de la segunda.

La transferencia entre el sistema procesador y el HD44780 puede ser mediante un bus de 8 o 4 bits.

El controlador de LCD dispone de dos registros accesibles desde el exterior, el registro de instrucciones (IR) y el de datos (DR), la señal RS indica cuando se accede a uno u otro. Estos se pueden leer o escribir dependiendo de la señal R/W.

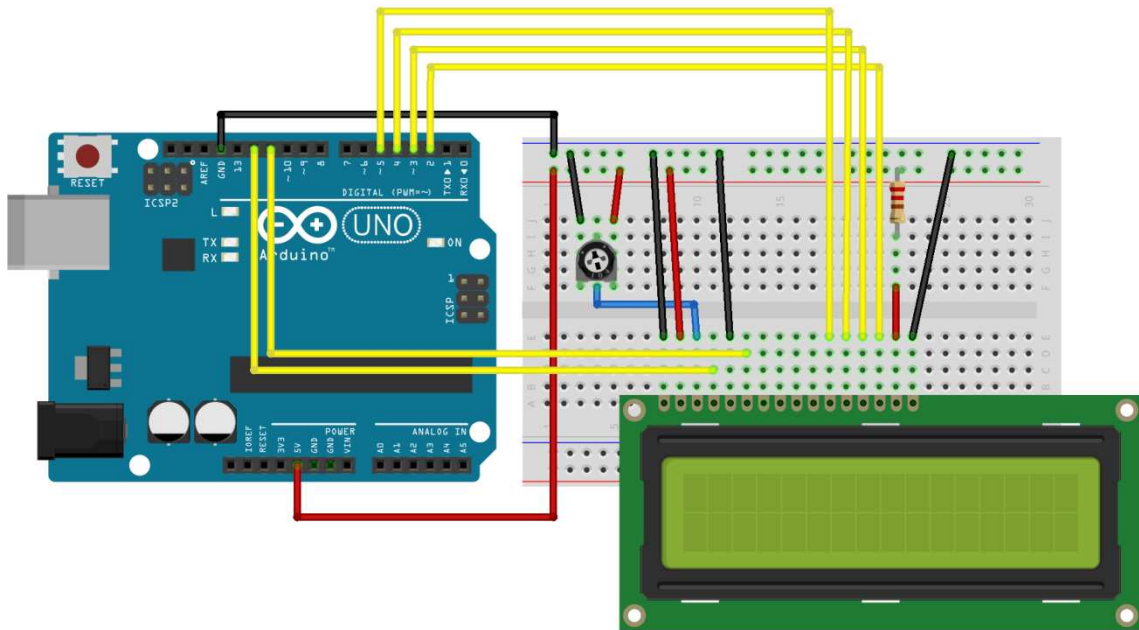
Cuando arranca el LCD el puntero en pantalla se pone en la primera posición y se va autoincrementando. Con un comando esta situación se podrá cambiar.

La memoria continua más allá de lo que abarca las 16 posiciones de la pantalla.

Una vez descrito el componente principal de esta práctica procedo a detallar todos los componentes que serán usados:

- Placa Arduino
- Protoboard
- Cable USB
- Display LCD LCM1602C
- Un potenciómetro de 4,7 a 10 KOhm
- Cable de conexión

El montaje quedará así:



Los dos terminales más a la derecha en el esquema superior son el ánodo y el cátodo. Estos son los responsables de iluminar el display. Si conectamos el primero de ellos a 5 voltios y el otro a masa, iluminaremos la pantalla.

El potenciómetro utilizado para obtener un buen contraste ha de ser de 4,7 a 10 KOhm. Mediante el potenciómetro podremos variar el contraste de la pantalla para una mejor visualización del texto.

Ello se consigue variando su resistencia mediante el giro de su rueda. Moveremos a un lado y a otro en busca del mejor contraste posible.

Tras ello introducimos en el IDE el código que usaremos.

### CÓDIGO PARA ARDUINO

```
#include <LiquidCrystal.h>
```

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

```
void setup() {  
  Serial.begin(9600);  
  lcd.begin(16, 2);  
  lcd.clear();  
  delay(1000);  
  lcd.print("preparado");  
  delay(1000);  
  lcd.clear();  
}
```

```
void loop() {  
  if (Serial.available()) {  
    delay(50);  
    lcd.clear();  
  }
```

```
while (Serial.available() > 0) {  
  char c = Serial.read();  
  lcd.write(c);  
  
}  
}}
```

Para que reconozca un display es necesaria la utilización de la librería LiquidCrystal, que proporciona los comandos esenciales para controlarlo. Esta librería viene por defecto al instalar el IDE de Arduino.

Para nuestro ejemplo usamos las instrucciones:

- lcd.begin( nº col,nº filas) donde indicamos el tamaño del LCD.
- lcd.clear() que borra todo lo que contenga la pantalla.
- lcd.print(texto) escribe el texto.
- lcd.write(carácter) que escribe un carácter en pantalla.

Por último y cuando tengamos la placa grabada procedo a escribir un script en lenguaje Python necesario para enviar caracteres vía puerto serie.

### ARCHIVO PYTHON

```
import serial  
  
arduino=serial.Serial('/dev/ttyACM0',baudrate=9600)  
cadena=''   
  
while True:  
    var = raw_input("Introduzca mensaje: ")  
    arduino.write(var)  
  
    while arduino.inWaiting() > 0:  
        cadena += arduino.readline()  
        print cadena.rstrip('\n')  
        cadena = ''  
arduino.close()
```

Al ejecutar el archivo donde se guardó este código aparecerá en el terminal “Introduzca mensaje: ” y tras ello estaremos en disposición de enviar nuestro texto a la pantalla.

### **CONCLUSIONES:**

A partir de este momento podremos dotar a los proyectos de una pantalla donde mostrar información al usuario usando todas las letras posibles y símbolos gráficos. Las pantallas LCD consumen menos corriente que los LED, son más flexibles y tienen menor coste por lo que es una buena opción incorporarlas.

## 1.3.6 Práctica 6

Enlazando con la práctica anterior usaremos la pantalla LCD para mostrar la temperatura registrada por un sensor.

Los sensores de temperatura son unos dispositivos muy útiles que se utilizan en sistemas diseñados para medir temperaturas en lugares accesibles o no accesibles por el ser humano. Un ejemplo podrían ser las estaciones meteorológicas portátiles que también integran reloj, humedad relativa, día, mes, etc.

### SENSOR DE TEMPERATURA TMP36GZ

Un sensor de temperatura es simplemente un chip que nos devuelve un valor de tensión proporcional a la temperatura a la que está sometido. En esta práctica vamos a utilizar el modelo TMP36GZ.

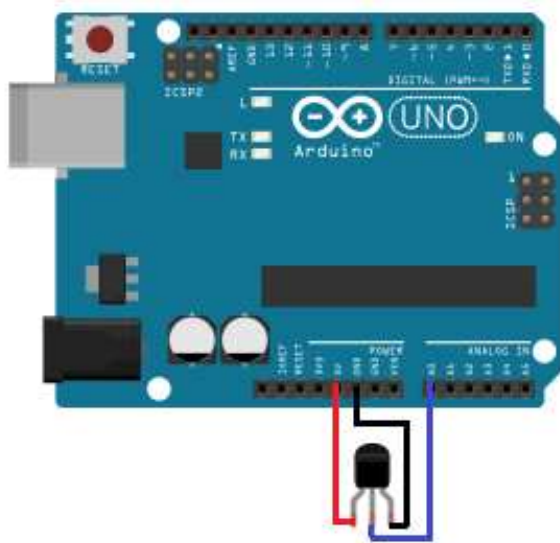
Una vez consultado el datasheet podemos resaltar como características principales:

- Mide la temperatura en grados centígrados.
- Funciona entre -50° C y 125°C para el TMP36.
- No es especialmente preciso, ya que tiene  $\pm 1^\circ\text{C}$  de incertidumbre.
- EL encapsulado es similar al de un transistor y también tiene tres patas.
- El pin central es el de señal, pero para saber cuál es GND y 5V, el encapsulado tiene una cara plana y otra curva.
- Si se conecta la tensión al revés comenzará a calentarse y será necesario desconectarlo y dejar que se enfríe.

Usaremos los componentes que se describen:

- Placa Arduino
- Protoboard
- Cable USB
- Display LCD
- Sensor tmp36GZ
- Cable de conexión

El montaje en este caso queda:



Añadiendo esta configuración al montaje de la práctica anterior conseguiremos mostrar la temperatura obtenida a través del LCD.

Después de conectar el sensor en la protoboard ya estamos preparados para tomar medidas y representarlas.

### CODIGO DE ARDUINO

El sketch que introducimos en el IDE es el que sigue,

```
#include <LiquidCrystal.h>
```

```
float medida;  
int tempPin = 0;
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);           // Inicializa librería indicando pines
```

```
void setup()  
{  
  Serial.begin(9600);                          // Abre Puerto a 9600 bps  
  lcd.begin(16, 2);                             // Inicializa tipo LCD  
  lcd.clear();                                  // limpia pantalla  
  delay(2000);                                  // Espera de 2 seg  
}
```

```
void loop()  
{  
  medida = ( 4.4 * analogRead(tempPin) * 26) / 1024.0;    // conversion para tmp36GZ .  
  lcd.setCursor(0, 0);                                     // Situa cursor  
  lcd.print("Temperatura: ");                             // Escribe texto  
}
```

```
lcd.setCursor(1, 1);           // Situa cursor
lcd.print(medida);             // Escribe la medida al LCD
lcd.setCursor(7, 1);           // Situa cursor
lcd.print("grados C");
delay(5000);                   // espera 5 seg
}
```

Puede observarse que hemos realizado la conversión de la lectura del sensor analógico a grados centígrados. También se añade una función no usada de la librería LiquidCrystal llamada `lcd.setCursor (fila, columna)` que sitúa el cursor como paso previo a una impresión de texto.

### CONCLUSIONES:

Con la realización de este ejemplo hemos conseguido tomar lecturas de un dispositivo analógico para su posterior uso. Los modelos de sensores para medir temperaturas son muy variados pero su uso no difiere mucho del que hemos llevado a cabo.

### 1.3.7 Práctica 7

La última práctica de esta parte consistirá en un contador de un minuto en un shield de la placa. Basta con añadirlo sobre Arduino aprovechando las conexiones que coinciden exactamente y sólo habrá que conectarla al PC.

#### SHIELD

Arduino es modular. La placa cuenta con conectores a los que conectar cables o componentes. Estos conectores presentan unos seis u ocho orificios, también llamados pines. Es posible acoplar a estos conectores placas de expansión que respeten la misma disposición de contactos.

Los shields son placas especiales para realizar funciones específicas. Para utilizar una de ellas basta con acoplarla encima de Arduino. Las shields están desarrolladas producidas por Arduino, aunque también otros conocidos productores de electrónica. Lo importante es respetar la disposición de los pines. Tenemos shields con pantalla LCD, con Ethernet para la WIFI, para Bluetooth, con GSM e incluso con tarjeta SD.

Para componer los circuitos simplemente se debe conectar a la placa madre.

En este caso los componentes serán sólo:

- Placa Arduino
- Cable de conexión
- Shield Arduino con display 7 segmentos

#### CODIGO DE 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 = 0;
int dat0 = 0;
int tiempoMax = 60;
void write_data(int arg) {
    switch (arg) {
        case 0:
            write7seg(0x7e);
            break;
        case 1:
            write7seg(0x30);
            break;
        case 2:
            write7seg(0x6d);
            break;
        case 3:
            write7seg(0x79);
            break;
        case 4:
            write7seg(0x33);
            break;
        case 5:
            write7seg(0x5b);
            break;
        case 6:
            write7seg(0x1f);
            break;
        case 7:
            write7seg(0x70);
            break;
        case 8:
            write7seg(0x7f);
            break;
        case 9:
            write7seg(0x73);
            break;
    }
}

void write7seg(unsigned char arg) {
    unsigned char segmen = 0x01;
    unsigned char display1;
    display1 = arg;
    for (int i = 0; i < 8; i++) {
        if ((display1 & segmen) == 0x00)
            digitalWrite(i, LOW);
        else
            digitalWrite(i, HIGH);
        segmen <<= 1;
    }
}
```

```
}  
void refresh (int data1, int data0) {  
    int j = 40;  
    int tiempo_refresco = tiempototal / (2 * j);  
    int i;  
    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);  
    }  
}  
void setup() {  
    for (int thisseg = 0; thisseg < 8; thisseg++) {  
        pinMode(segPins[thisseg], OUTPUT);  
    }  
    pinMode(dis1, OUTPUT);  
    pinMode(dis2, OUTPUT);  
}  
  
void loop() {  
    int valor;  
    while (1) {  
        for (valor = 0; valor < 59 ; valor++) {  
            dat1 = valor / 10;  
            dat0 = valor % 10;  
            refresh(dat1, dat0);  
        }  
    }  
}
```

La Función refresh va refrescando los displays 7 segmentos alternativamente cada Xms y en total tiene que durar 1seg.

Para activar display1 o display0 se emplean los bits PB0 y PB1 de la placa de expansión.

El refresco lo puede hacer con llamada a la función write\_data( int dato).Es la función que dado un dato entero (dígito de 0 a 9) calcula el valor correcto de representación en los displays 7-seg y lo escribe en la función mediante write7seg(unsigned char arg).

## CONCLUSIONES:

Quedaba por usar un shield de Arduino que es cómodo e inmediato de usar. Hemos añadido el sketch a la IDE y tras compilarlo se podrá cargar a la placa. Sería posible desarrollar nuestro propio shield para una tarea específica que pretendamos haciendo coincidir los pines de forma adecuada.

## PARTE 2: PLATAFORMA 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 INTRODUCCION A PAPILIO

Papilio es una placa de desarrollo de código abierto basado en FPGA en el Xilinx Spartan 3E FPGA. Cuenta con 48 líneas de E/S, USB de doble canal, programador JTAG integrado, 4 fuentes de alimentación y un conector de alimentación. Nosotros utilizaremos Papilio one 500.



# Memorias prácticas LDH

---

## Características:

- Cuatro brazos de alimentación independientes a 5V, 3,3V, 2,5V y 1,2V.
- Alimentación suministrada por un conector de alimentación o USB.
- Entrada de CC.
- Voltaje de entrada (recomendado): 6.5-10V
- Dos conexiones USB para JTAG y comunicaciones serie implementados con FT2232D.
- La memoria EEPROM para almacenar ajustes de configuración para FT2232 de chips USB.
- Spartan 3E FPGA
  - Oscilador 32 MHz que puede ser utilizada por DCM de Xilinx para generar cualquier velocidad de reloj requerida.
  - VTQFP-100 es compatible con la huella que Xilinx XC3S100E, XC3S250E, y XC3S500E partes.
  - E / S se puede configurar para soportar 1.2V, 2.5V o 3.3V.

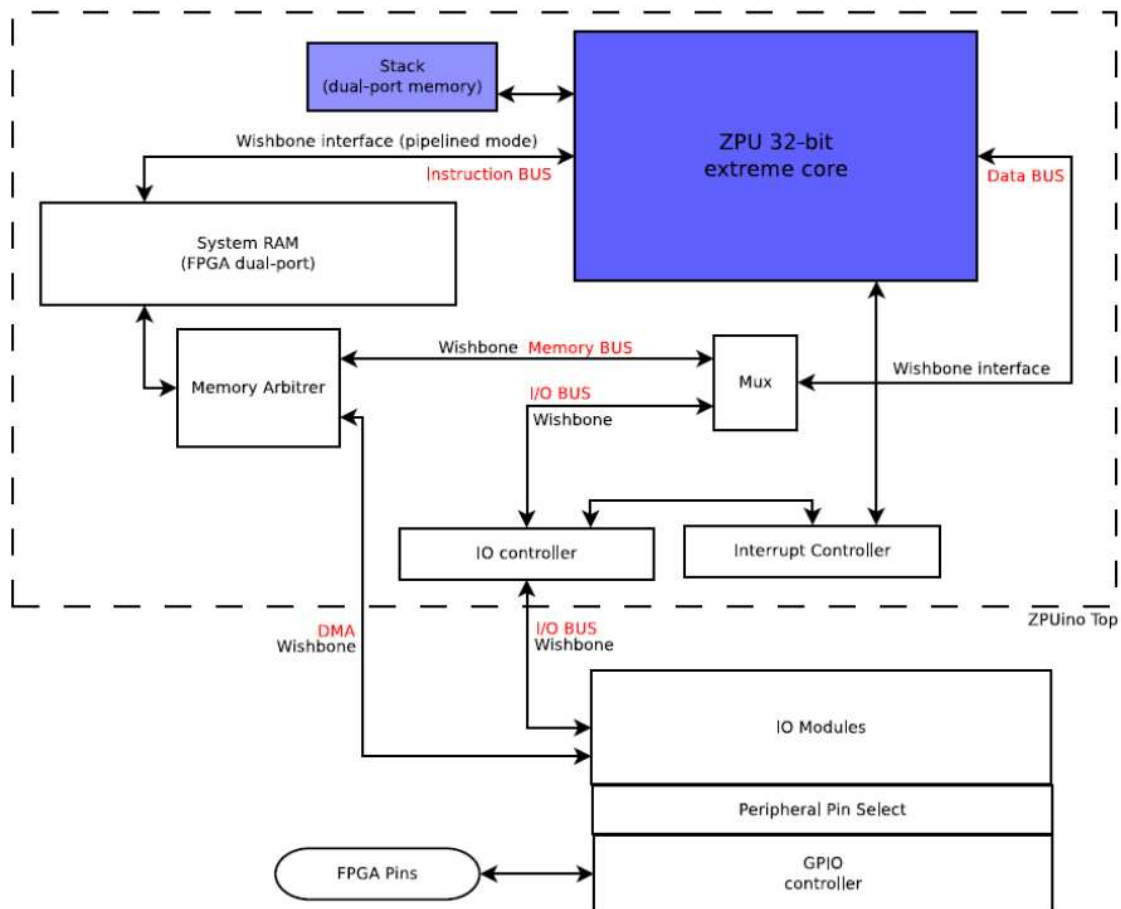


Papilio consta de un conjunto de plataformas de desarrollo Hardware basadas en FPGAs mas un entorno de desarrollo Software que permite tanto diseñar el hardware de la FPGA como desarrollar Sketches (programas) que se ejecutaran en el microcontrolador empleado (ZPUino/Arduino).

ZPUino es un SoC (System-on-a-Chip) basado en el núcleo ZPU de 32 bits de Zylín.

- Procesador de 32 bits funcionando a 96Mhz.
- Utiliza una versión modificada del IDE de Arduino para cargar bocetos.
- Wishbone bus para conectar periféricos
- Conmutador 64 SID Audio Chip
- YM2149 Audio Chip
- POKEY Audio Chip
- Mezclador de audio
- ZX Spectrum VGA
- HQVGA con color de 8 bits
- UART
- GPIO
- SPI
- I2C

- SigmaDelta DAC



ZPUino incluye una característica que se llama Peripheral Pin Select ( PPS). PPS le permite asignar cada pin de entrada o salida de dispositivo (como reloj SPI y datalines SPI) a cada pin individual (GPIO), por lo que no requiere que realice síntesis y P & R cada vez que desee utilizar un dispositivo en un IO diferente alfiler.

## 2.3 PRACTICAS DE PLATAFORMA ZPUINO

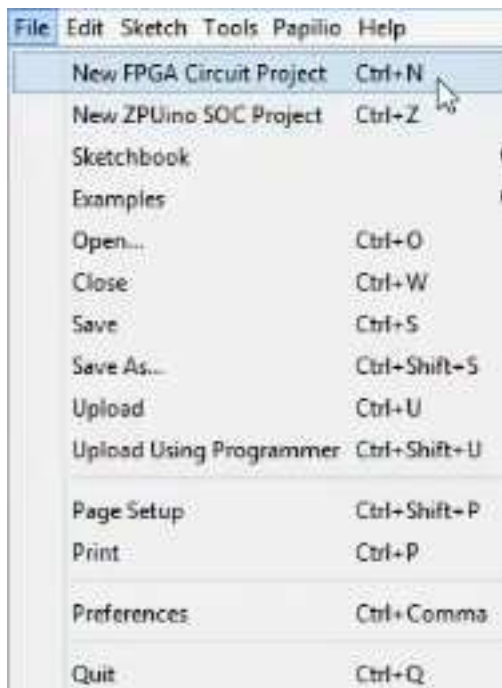
- 1.- Crear un inversor con un circuito FPGA
- 2.-Controlar el encendido apagado desde el PC vía puerto serie
- 3.- Convertir la placa Papilio en un Analizador lógico
- 4.- Controlar encendido/apagado de un led desde el PC a través de una UART

## 2.3.1 Práctica 1

La primera práctica consiste en la creación de un circuito en una FPGA.

Será un circuito simple que consiste en un inversor, conectando la entrada a un botón y la salida a un LED. Cuando presionamos el botón, el LED mostrará la salida invertida.

Primer paso pulsar nuevo circuito

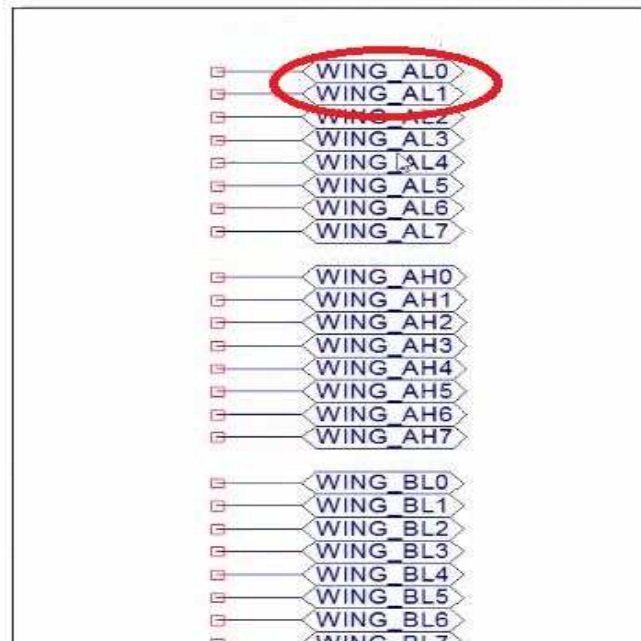


Salvando la plantilla con el nombre adecuado ya podremos modificarla.

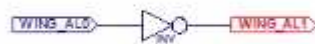
En la ventana de diseño con la opción implementación marcada seleccionamos Papilio\_Pro se abrirá el esquemático.

Buscaremos los símbolos y los iremos añadiendo. Los marcadores de E / S le indican al software qué pines externos desea conectar en la placa FPGA. Para ver qué pines están disponibles en la tarjeta Papilio, se usa la ficha Diseño y se hace clic en Utilidad. Esto abrirá el diagrama de conexiones Papilio One y Papilio Pro, mostrando todos los pines disponibles.

Vamos a usar WING\_AL0 y WING\_AL1.



Cambio el nombre del marcador de E / S de salida WING\_AL1. Este será un LED.



Por lo tanto, al conectar un botón y un LED a la posición AL en la placa FPGA, la entrada del botón se invertirá y se conectará al LED.

Sólo falta sintetizar el circuito y cargarlo a la placa. Conectando un switch y un led a la placa verificamos que se genera la señal invertida.

### CONCLUSIONES:

Con Design Lab se pueden diseñar circuitos para implementarlos en la FPGA. Para nuestro caso resulta simple implementar un inversor diseñando un circuito que se cargará a la FPGA para darle a esta la funcionalidad que necesitemos.

### 2.3.2 Práctica 2

Vamos a cargar el SoC ZPUINO y a desarrollar un sketch para controlar, tal como lo hicimos con Arduino, el encendido/apagado de un led vía puerto serie.

Usamos los siguientes componentes:

- Placa PAPILIO ONE
- Cable de conexión
- Protoboard
- 1 Resistencia
- 1 Led

Conectamos la placa de forma que a la salida del pin 13 colocamos la resistencia en serie con el led en la protoboard.

Siguiendo ahora los pasos para cargar el SoC de ZPUINO:

Ejecutar en el directorio donde instalamos DesignLab la aplicación.

A continuación cargamos el sketch QuickStart para iniciar. Seleccionamos la placa que estemos usando (para nosotros PAPILIO ONE (500 k)-ZPUINO. También el puerto adecuado y presionamos Load Circuit.

Ya estamos en disposición de cargar el siguiente sketch usando el IDE de Arduino

```
int led = 13;
void setup() {
  pinMode(led, OUTPUT);
  mySerial.begin(9600);
}
void loop() {
  if(Serial.available()){
    char c = Serial.read();
    if(c=='H'){
      digitalWrite(led, HIGH);
    }else{
      digitalWrite(led, LOW);
    }
  }
}
```

Tras compilar y cargar pasamos al terminal para ejecutar el script que ya habíamos usado en otras ocasiones.

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

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

Al presionar la tecla “L” se encenderá el led. Al presionar “L” este se apagará.



### CONCLUSIONES:

Hemos usado la placa PAPILIO con el IDE de Arduino tras haber programado la FPGA con el SoC de ZPUINO. Esto nos vale como ejemplo de uso ya que son muchas las configuraciones posibles que puede adquirir una placa de este tipo.

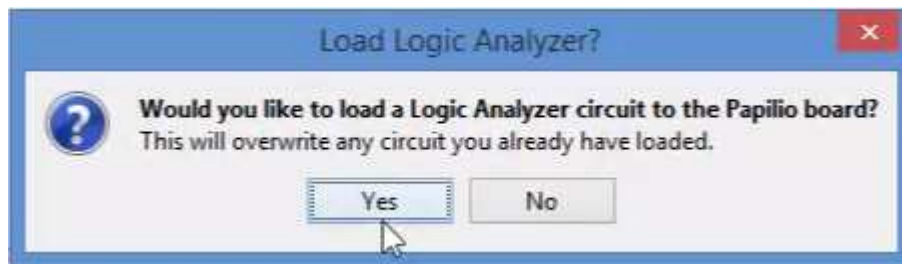
### 2.3.3 Práctica 3

La PAPILIO se comportará ahora como un analizador lógico. Podemos hacerlo sin hacer uso de sketch alguno. Seguimos los pasos del tutorial que acompaña a la práctica.

Utilizando la última versión de DesignLab pulsamos el icono de Logic Analyzer. Esta función cargará un archivo de bit a la Papilio y la transforma en un analizador lógico.

Se selecciona la placa como siempre.

A continuación clic en el icono de Logic Analyzer. Un mensaje pregunta si se desea sobrescribir el circuito existente en la tarjeta Papilio.



Después de pulsar en Yes se grabará un archivo y nos indicará como se conectan los canales.



Para capturar los datos del analizador pulsamos en el menú siguiente en comenzar a capturar datos.

En este punto, se mostrarán los ajustes de captura. Seleccionamos "Puerto serie", seleccionamos el puerto COM adecuado, establecemos la velocidad del puerto a 115200 bps y luego elegimos el tipo de dispositivo apropiado.

Se establece una Tasa de muestreo de acuerdo a las necesidades de 20.000 Mhz. También podemos ajustar la cantidad de memoria disponible. Cuantos más canales hayamos seleccionado, menos memoria tendremos. Por último se pulsa la captura.

Una vez que tengamos funcionando el analizador lógico en la placa Papilio One 500, vamos a coger un Arduino, cargaremos el ejemplo fade y debemos ver la señal PWM con el analizador lógico.

## CONCLUSIONES:

Con la ayuda de una placa Arduino hemos conseguido usar una FPGA para analizar una señal. Es una de las opciones que brinda la Papilio One sin necesidad de sketches, sólo con el uso de sus puertos para tomar la señal.

### 2.3.4 Práctica 4

Como última tarea a realizar con la placa Papilio vamos a añadirle un periférico al Soc de ZPUINO, sintetizarlo y usarlo desde un sketch.

Seguimos para ello el tutorial suministrado.

Una vez que tenemos el archivo que se nos indica editaremos el circuito y lo modificaremos para añadir una UART y usarla como alternativa a la disponible por defecto.

Sintetizando el circuito y cargándolo podremos disponernos a seleccionar el puerto en el IDE y cargar nuestro código.

```
HardwareSerial mySerial(WishboneSlot(5));
```

```
int led = 13;
void setup() {
  pinMode(led, OUTPUT);
  mySerial.begin(9600);
}
void loop() {
  if(mySerial.available()){
    char c = mySerial.read();
    if(c=='H'){
      digitalWrite(led, HIGH);
    }else{
      digitalWrite(led, LOW);
    }
  }
}
```

La primera línea hace referencia al slot del bus al que estamos conectados. Volvemos a nuestro script ya usados varias veces de Phyton :

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

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

## CONCLUSIONES:

Es posible manejar más de una UART configurándolas y nombrándolas de esta forma para manejar distintos dispositivos. Esta opción nos abre muchas vías de manejo hardware sin carga ninguna al procesador ya que este se limitará a recoger el dato procedente de la acción realizada por el periférico.

## PARTE 3: PLATAFORMA RASPBERRY PI 3

### 3.1 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.

### 3.2 INTRODUCCIÓN A RASPBERRY PI 3



Especificaciones técnicas:

- Procesador a 1,2 GHz de 64 bits con cuatro núcleos ARMv8.
- 802.11n Wireless LAN.
- Bluetooth 4.1.
- Bluetooth Low Energy (BLE).

- 4 puertos USB.
- 40 pines GPIO.
- Puerto Full HDMI.
- Puerto Ethernet.
- Conector combo compuesto de audio y vídeo de 3,5 mm.
- Interfaz de la cámara (CSI).
- Interfaz de pantalla (DSI).
- Ranura para tarjetas microSD (ahora push-pull en lugar de push-push).
- Núcleo de gráficos VideoCore IV 3D.
- Dimensiones de placa de 8.5 por 5.3 cm.

### **3.3 PRACTICAS DE PLATAFORMA RASPBERRY**

- 1.-Manejar GPIOs desde línea de comandos. Ejecutar código python desde el servidor web.
- 2.- Manejar GPIOs desde un servidor web.
- 3.-Conectar ARDUINO a la Raspberry PI .Desde un servidor ser capaz de leer una temperatura.
- 3.-Añadir al sensor de temperatura un relé y una bombilla. Desde el servidor web encender o apagar luz (manteniendo la lectura de la temperatura)