

GUIA DE LABORATORIO DE PROCESADORES Y CONTROLADORES DIGITALES

**ING. MARCIO CARVAJAL C.
2022**

CONTENIDO**Página**

INTRODUCCIÓN	1
---------------------------	----------

MICROCONTROLADOR ARDUINO

Laboratorio 1: funciones Básicas	2
Práctica para el alumno	4
Laboratorio 2: Variables, Operadores y tipos de entrada	5
Práctica para el alumno	11
Laboratorio 3: Pantalla LCD	12
Práctica para el alumno	15
Laboratorio 4: Control de velocidad y sentido de Motor DC	16
Práctica para el alumno	21
Laboratorio 5: Control de motor paso a paso	22
Práctica para el alumno	33
Laboratorio 6: Control de Servomotor	34
Práctica para el alumno	40
Laboratorio 7: Sensor de Ultrasonidos	41
Práctica para el alumno	45
Laboratorio 8: Sensor de Color	46
Práctica para el alumno	51
Laboratorio 9: Teclado Matricial	52
Práctica para el alumno	55
Laboratorio 10: Bluetooth	56
Práctica para el alumno	60

Introducción

La presente guía de laboratorio de la materia de “Procesadores y Controladores Digitales” está constituido en dos partes. La primera contiene programas realizados en el software IDE Arduino y sus respectivas simulaciones en proteus para Arduino, como ser rotación de leds, manejo de pantalla LCD, control de motores DC, paso a paso y sevomotres y sensores. Existiendo prácticas propuestas para el estudiante.

La segunda parte contiene ejercicios y programas para el microcontrolador arduino, con sus respectivos programas, diseñados con el software de diseño electrónico Fritzing. Las practicas a realizar contiene el uso de sensores, actuadores y visualización los mismo que serán implementados en el laboratorio.

“Un hombre como yo no es descriptible por las cosas que dice o los sentimientos que tiene, sino por lo que piensa y como lo piensa.”

Albert Einstein

LABORATORIO N° 1

FUNCIONES BASICAS ARDUINO

➤ **Objetivo.-**

Aprender la estructura de programación y las funciones basicas para la programacion en arduino.

➤ **Marco teórico.-**

Para realizar la programación se tiene que tomar en cuenta los siguientes puntos:

- Cada instrucción (a excepción de las funciones y librerías) deben terminar con un punto y coma “;” .
- Cada función debe tener un inicio y un final y estos se definen con llaves “{ }” .
- Todo programa de arduino debe llevar como mínimo las funciones:
Void setup() y void loop().
- Para hacer un comentario se utiliza “//” o “/*” y “*/” .

Funciones básicas

- void setup() Su principal función es la de configuración del Arduino para inicializar los modos de trabajos de los pines (entrada o salida) o el puerto serial.

Ejemplo:

```
void setup()
{
  pinMode(pin, OUTPUT); // configura el 'pin' como salida
}
```

- void loop() Función utilizada para que el programa se ejecute de forma cíclica.

Ejemplo:

```
void loop()
{
  digitalWrite(pin, HIGH); // pone en uno (on, 5v) el 'pin'
  delay(1000); // espera un segundo (1000 ms)
  digitalWrite(pin, LOW); // pone en cero (off, 0v.) el 'pin'
  delay(1000);
}
```

Instrucciones para salidas digitales

- **pinMode(pin, Modo);** //configura el pin especificado para entrada o salida. Esta instrucción trabaja dentro de la función void setup.

Ejemplo:

```
pinMode(13, OUTPUT);  
pinMode(9, INPUT);
```

- **digitalWrite(pin, Valor);** //Establece un valor alto (HIGH) o bajo (LOW), de un pin digital configurado anteriormente. Esta instrucción trabaja dentro de la función void loop.

Ejemplo:

```
digitalWrite(13, HIGH);  
digitalWrite(8, LOW);
```

- **delay(milisegundos);** //Retrasa el programa la cantidad de milisegundos especificados. Esta instrucción trabaja dentro de la función void loop.

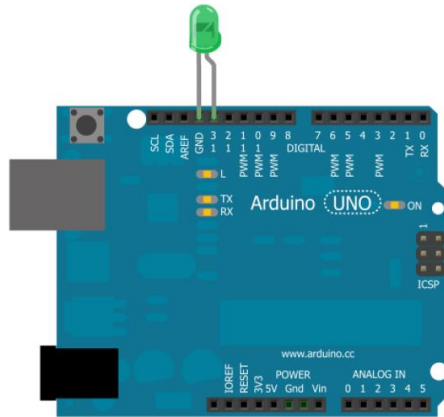
Ejemplo:

```
delay(1000);  
delay(2000);  
delay(100);
```

- **Programación y Circuito.-**

Enciende un LED por un segundo y lo apaga por el mismo tiempo

```
// Inicio del programa  
void setup() // Se ejecuta cada vez que el Arduino se inicia  
{  
  pinMode(13,OUTPUT); // Inicializa el pin 13 como una salida  
}  
void loop() // Esta función se mantiene ejecutando  
{ // cuando este energizado el Arduino  
  digitalWrite(13,HIGH); // Enciende el LED  
  delay(1000); // Temporiza un segundo (1s = 1000ms)  
  digitalWrite(13,LOW); // Apaga el LED  
  delay(1000); // Temporiza un segundo (1s = 1000ms)  
}  
// Fin del programa
```



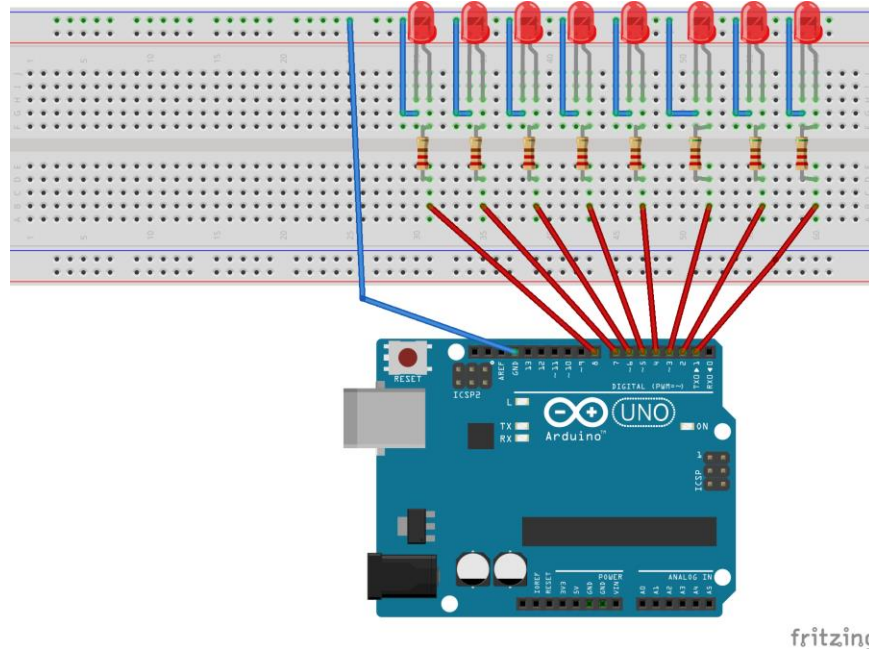
PRÁCTICA PARA EL ESTUDIANTE:

Leds con secuencia

Diseñar e implementar la rotación de 8 leds conectados a partir del pin 1 al 8. Utilizar una resistencias de protección de 220 ohms, la secuencia consiste en la siguiente, cuando enciende el arduino los leds realizarán un corrimiento de izquierda a derecha con una duración entre encendido y apagado de 1 segundo, cuando se apague el octavo led el corrimiento continua de derecha a izquierda con la misma duración de tiempo

Reglas:

- NO se pueden prender dos o más leds al mismo tiempo
- NO se pueden quedar todos los leds apagados
- Una vez terminada la secuencia modifique los tiempos para que practique.



fritzing

LABORATORIO N° 2

VARIABLES, OPERADORES Y TIPOS DE ENTRADAS EN ARDUINO

➤ **Objetivo.-**

Identificar las variables de programación, operadores y tipos de entradas de datos para la programación de Arduino.

➤ **Marco teórico.-**

Las variables se utilizan básicamente para almacenar información útil para nuestro programa como puede ser: valor de un sensor, el resultado de una operación matemática o cualquier dato que querremos conservar en el código.

¿Cómo y en donde se declaran las variables en Arduino?

Las variables generalmente se declaran antes de la función void setup, con esto la variable se vuelve global y puede ser utilizada en cualquier parte del código, si se declara dentro de una función en específico solo se puede utilizar en esa función debido a que se vuelve local.

Para declarar una variable primero especificamos el tipo de variable, seguido del nombre de la variable y podemos asignarle un valor de inicio o no, al final lleva punto y coma

Ejemplo:

```
int dato = 3;  
int x;
```

4 tipos de variables más comunes en Arduino

- **Boolean (Booleano)**, un booleano solo puede tomar dos valores: falso o verdadero
Declaración en Arduino: `boolean x = 0;`
- **Int (Entero)**, principal tipo de datos para almacenar números, es de 2 bytes y tiene un rango de -32768 a 32767. Declaración en Arduino: `int y = 4000;`
- **Float (Flotante)**, tipo de datos que utiliza puntos decimales, generalmente su uso es en operaciones matemáticas, Declaración en Arduino: `float z = 1.7;`
- **Char (Carácter)**, tipo de datos que almacena una variable tipo carácter de 1 byte, su uso principal es para texto: `char letra = 'a';` (el carácter debe ir entre comillas sencillas)

Operadores Aritméticos y Comparativos

- **Aritméticos:**

= Asignación, ejemplo: `int x = 8;`

+ Suma, ejemplo `x = y + z;` - Resta, ejemplo `x = y - z;`

* Multiplicación, ejemplo `x = x * y;`

/ División, ejemplo `x = x / y;`

- **Comparativos:**

== Igual a, ejemplo `x == 8;`

!= Distinto a, ejemplo `x != 10;`

< Menor que, ejemplo `x < 8;`

> Mayor que, ejemplo `x > 7;`

Entrada digital en Arduino

Arduino tiene la capacidad de leer entradas digitales por cualquiera de sus pines, incluso en los pines análogos, el arduino lee un cambio de voltaje de 0 a 5 volts, donde 0 = 0 o false, y 5v = 1 o true. Las entradas digitales generalmente se utilizan para la lectura de botones o cualquier señal que cambie de 0 a 5 volts.

La instrucción que se utiliza es: **`digitalRead(pin);`**

En donde pin representa el número del pin por donde vamos a conectar la entrada digital. Para leer una entrada digital generalmente se debe utilizar una variable tipo booleana.

Ejemplo:

```
boolean x; x = digitalRead(4);
```

Comunicación Serial

La comunicación serial consiste en el envío de un bit de información de manera secuencial, esto es un bit a la vez y a un ritmo acordado entre el emisor y el receptor. La comunicación serial en computadores ha seguido los estándares definidos en 1969 por El RS232 (Recommended Standard 232) que establece niveles de voltaje y la velocidad de transmisión de los datos. Puntos importantes con Arduino:

- Maneja una comunicación serial RS232 a nivel TTL.
- La velocidad típica de comunicación es 9600 baudios.
- Los pines de transmisión son TX (transmisor) y RX (receptor)

Instrucciones básicas en monitor serial de arduino

Serial.begin(rate); Establece la velocidad en bits por segundo en la que va a operar la comunicación serial, estas velocidades pueden ser: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200.

Ejemplo:

```
void setup()
{
  Serial.begin(9600); // abre el Puerto serie y configura la velocidad a 9600 bps
}
```

Serial.print(data); Imprime los datos al puerto serial como texto ASCII.

Serial.println(); Imprime los datos al puerto serial como texto ASCII y además agrega un salto de línea.

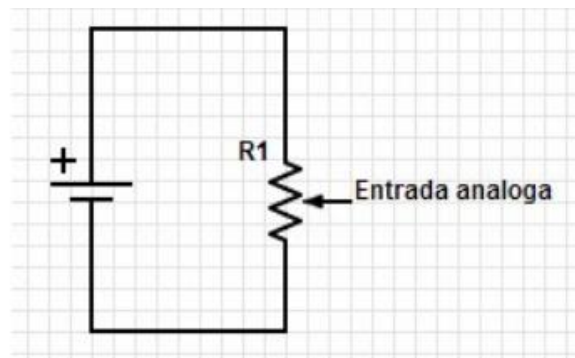
Entrada analógica

analogRead(); Lee el valor en voltaje del pin analógico especificado, los ADC de arduino tienen una resolución de 10 bits, esto quiere decir que proporciona una lectura de 5volts/1024 unidades.

Ejemplo:

```
void setup()
{
  Serial.begin(9600); // configura el puerto serie a 9600bps
}
void loop()
{
  Serial.println(analogRead(0)); // envía valor analógico
  delay(1000); // espera 1 segundo
}
```

ADC
0 --- 1023
0 ---- 5V



Condicional if

If() Estructura de control condicional utilizada para comparar.

Sintaxis

EJ:

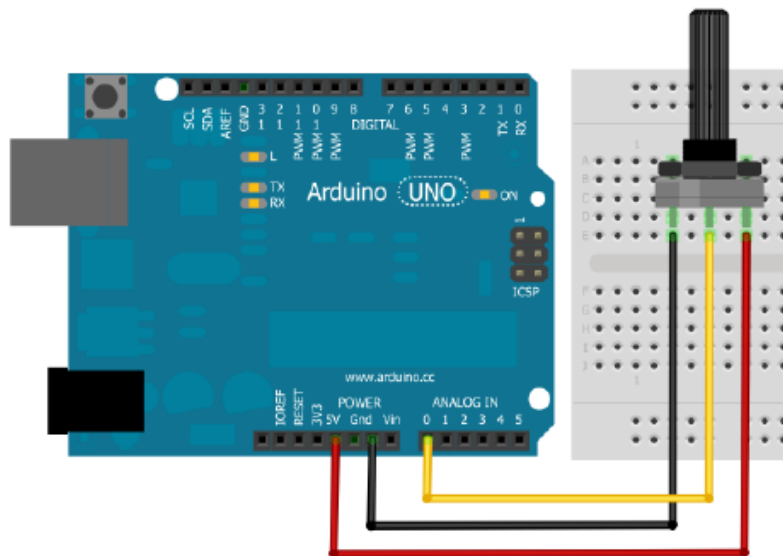
```
if (x == 8){ //hace lo que este dentro de las llaves
}
```

EJ:

```
if (x < 45){
//accion a
}
else{
//accion b
}
```

PROGRAMAS BASICOS PARA FAMILIARIZARSE CON ARDUINO➤ **Ejemplo 1.- Entrada Análoga**

Leer una entrada análoga y mostrar por la pantalla del computador (consola serial) el valor luego de girar el potenciómetro



```
void setup() // Se ejecuta cada vez que el Arduino se inicia
{
  Serial.begin(9600); //Inicia comunicación serial
}
void loop() // Esta función se mantiene ejecutando
```

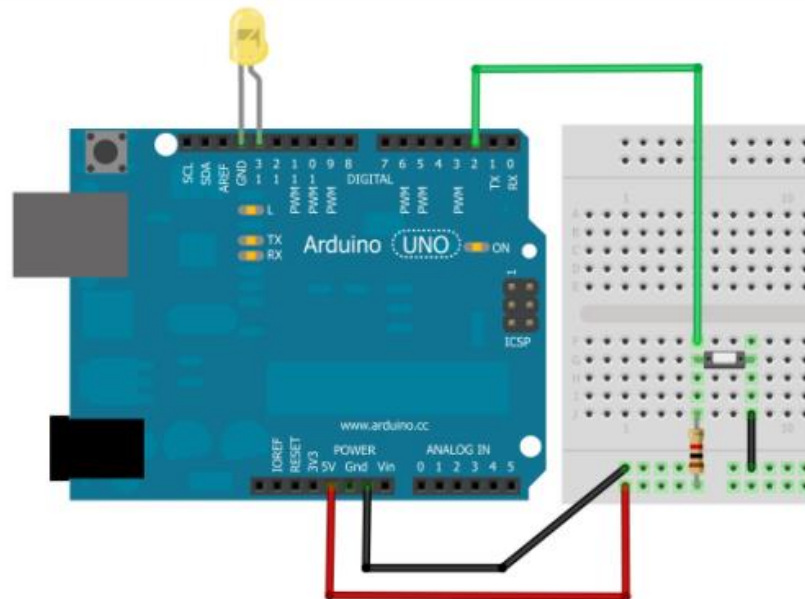
```

{ // cuando este energizado el Arduino
//Guardar en una variable entera el valor del potenciómetro 0 a 1024
int valor= analogRead(A0);
//Imprime en la consola serial el valor de la variable
Serial.println(valor);
//Retardo para la visualización de datos en la consola
delay(100);
}
//Fin programa

```

➤ **Ejemplo 2.- Encender un LED con un pulsador**

Oprimir un pulsador y mientras este se mantenga accionado un LED se enciende



```

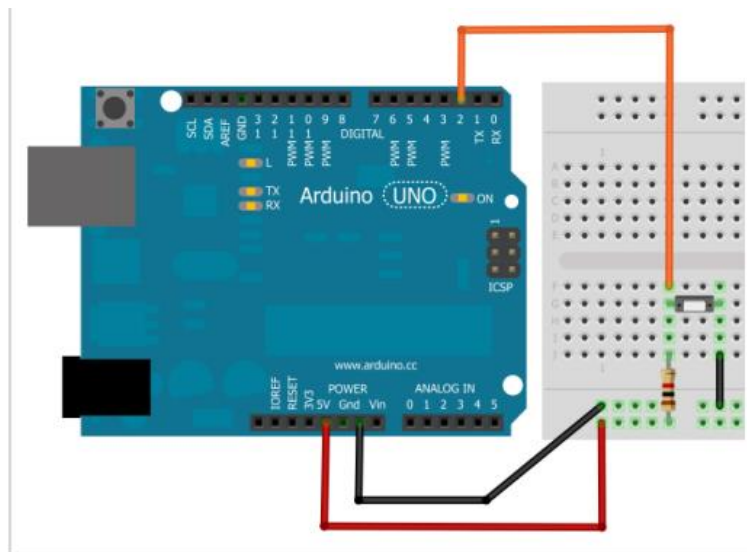
//-----
//Declara puertos de entradas y salidas
//-----
int pulsador=2; //Pin donde se encuentra el pulsador, entrada
int led=13; //Pin donde se encuentra el LED, salida
//-----
//Funcion principal
//-----
void setup() // Se ejecuta cada vez que el Arduino se inicia
{
pinMode(pulsador, INPUT); //Configurar el pulsador como una entrada
pinMode(led,OUTPUT); //Configurar el LED como una salida
}
//-----
//Funcion ciclica

```

```
//-----
void loop() // Esta funcion se mantiene ejecutando
{ // cuando este energizado el Arduino
  //Condicional para saber estado del pulsador
  if (digitalRead(pulsador)==HIGH)
  {
    //Pulsador oprimido
    digitalWrite(led,HIGH); //Enciende el LED
  }
  else
  {
    //Pulsador NO oprimido
    digitalWrite(led,LOW); //Apaga el LED
  }
}
//Fin programa
```

Ejemplo 3.-Lectura Serial de una Entrada Digital

Leer una entrada digital y mostrar por la pantalla del computador (consola serial) el estado del pulsador cuando es oprimido



```
//Declara puertos de entradas y salidas
//-----
int boton=2; //Pin donde se encuentra el pulsador, entrada
//-----
void setup() // Se ejecuta cada vez que el Arduino se inicia
{
  //Configuración
  pinMode(boton,INPUT); //Configurar el boton como una entrada
  Serial.begin(9600); //Inicia comunicación serial
}
void loop() // Esta funcion se mantiene ejecutando
```

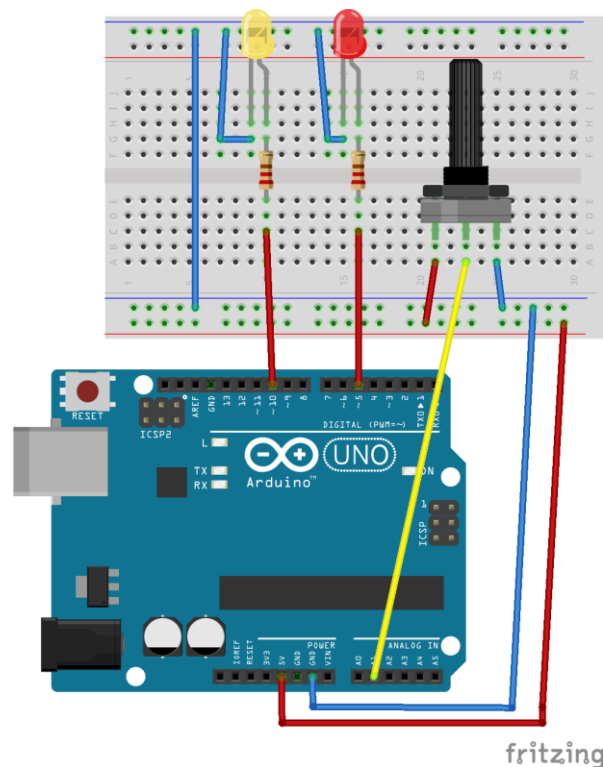
```

{ // cuando este energizado el Arduino
// Guardar en una variable entera el valor del boton 0 ó 1
int estado = digitalRead(boton);
// Condicional para saber estado del pulsador
if (estado==1)
{
// Pulsado
Serial.println("Pulsado"); //Imprime en la consola serial
} // "Pulsado"
else
{
// No esta pulsado
Serial.println("NO Pulsado"); //Imprime en la consola serial
} // "NO Pulsado"
delay(100); //Retardo para la visualización de datos en la consola
}
//Fin programa

```

PRÁCTICA PARA EL ESTUDIANTE:

Diseñar e implementar un circuito en base al microcontrolador Arduino que lea la entrada análoga por el pin 1 conectado a un potenciómetro. Si el valor del potenciómetro esta entre 800 y 1023 que encienda un led rojo. Si el valor del potenciómetro esta entre 0 y 400 que encienda un led amarillo. El estado del potenciómetro se verá en la pantalla del monitor serial.



LABORATORIO N° 3

PANTALLA LCD

➤ **Objetivo.-**

Aprender a programar una pantalla LCD para representar los datos que necesitemos visualizar.

➤ **Marco teórico.-**

Pantalla LCD

Dentro de la computación física, tenemos sensores y actuadores. Estos componentes son las interfaces hardware que ponen en contacto el mundo físico con el mundo virtual. Al igual que con los ordenadores tenemos teclado, ratón, altavoces, etc..., con las placas microcontroladoras como Arduino, tenemos hardware que harán la misma función. En este caso vamos a ver un actuador, una pantalla LCD con Arduino.

Este componente se encarga de convertir las señales eléctricas de la placa en información visual fácilmente entendible por los seres humanos. Debemos de dominar tanto las conexiones como la programación de la pantalla LCD con Arduino ya que es un componente muy útil en muchos proyectos. La gran ventaja es que gracias a la pantalla LCD, podremos mostrar información de datos como temperatura, humedad, presión o voltaje.

Pantalla monocromática 16x2 caracteres, utilizada generalmente para el despliegue de mensajes de texto y algunos caracteres especiales. Se llama 16x2 porque tiene la capacidad de mostrar 16 caracteres en dos líneas

Funciones para una pantalla LCD

Para utilizar una pantalla LCD debemos incluir una librería que se llama LiquidCrystal:

```
#include LiquidCrystal lcd(12,11,5,4,3,2); //indicamos donde se conecto físicamente la pantalla al arduino
```

```
lcd.begin(16,2); Configura la pantalla donde indicamos la cantidad de caracteres y filas que tiene
```

```
lcd.print("text"); imprime en la pantalla LCD
```

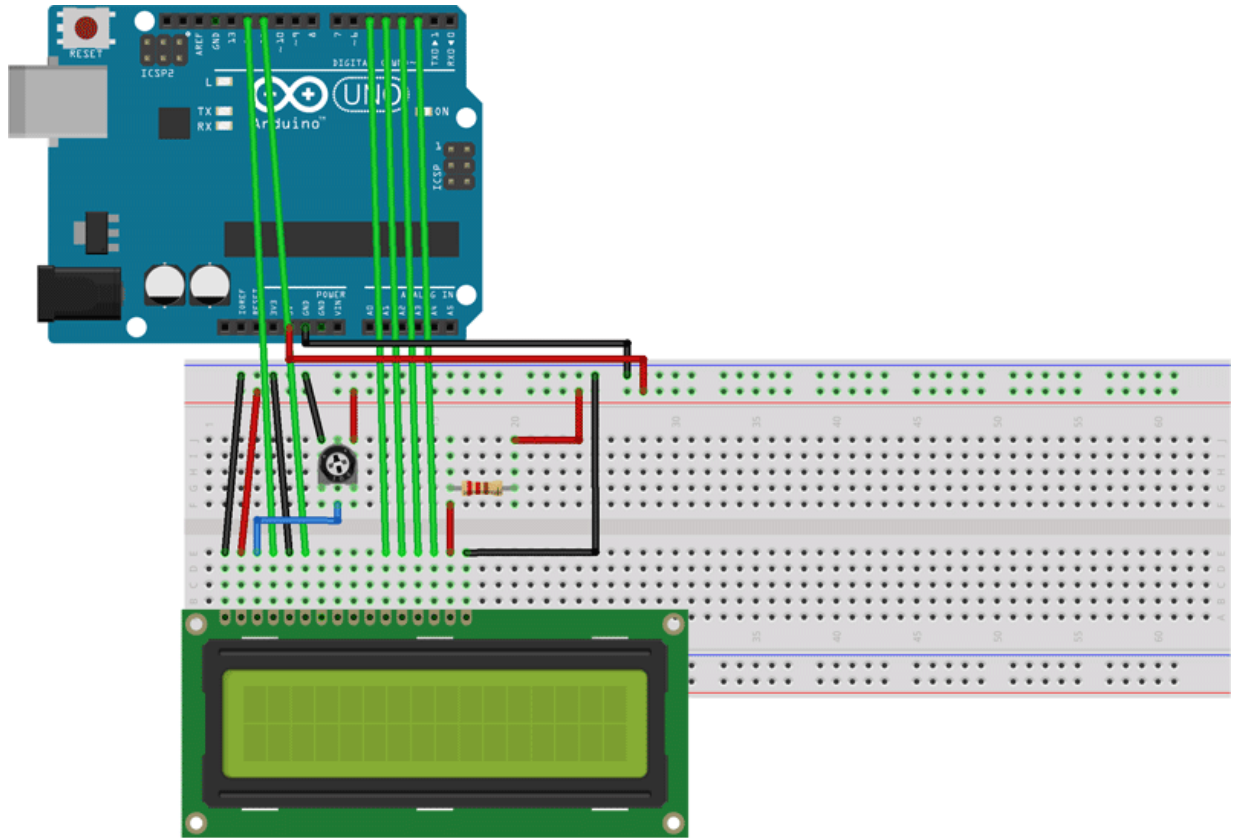
```
lcd.clear(); Limpia la pantalla lcd
```

```
lcd.setCursor(Posicion horizontal, Fila); ubica el cursor donde se va a escribir en la LCD.
```

➤ **Circuito y Programación.-**

El siguiente programa es un ejemplo básico para comenzar a utilizar nuestra pantalla y probar que las conexiones se encuentran correctas. Al cargarlo debemos ver la palabra

HOLA MUNDO en pantalla LCD y además una animación del texto moviéndose en la pantalla. Hemos comentado lo más posible el código de manera que sea fácil de entender.



fritzing

```

*/
#include <LiquidCrystal.h>
// CONSTRUCTOR PARA LA PANTALLA LCD 16X2
// AQUI SE CONFIGURAN LOS PINES PARA LA COMUNICACION CON LA PANTALLA
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup()
{
  // INDICAMOS QUE TENEMOS CONECTADA UNA PANTALLA DE 16X2
  lcd.begin(16, 2);
  // MOVER EL CURSOR A LA PRIMERA POSICION DE LA PANTALLA (0, 0)
  lcd.home();
  // IMPRIMIR "INGENIERIA" EN LA PRIMERA LINEA
  lcd.print("INGENIERIA");
  // MOVER EL CURSOR A LA SEGUNDA LINEA (1) PRIMERA COLUMNA (0)
  lcd.setCursor ( 0, 1 );
  // IMPRIMIR OTRA CADENA EN ESTA POSICION
  lcd.print("ELECTROMECANICA");
  // ESPERAR UN SEGUNDO
  delay(1000);
}

```

```

void loop()
{
  // EN EL CICLO PRINCIPAL SOLAMENTE RECORREMOS EL MENSAJE DE UN LADO A OTRO
  // VARIABLE PARA CONTROL DE CICLOS
  int i;
  // DESPLAZAR LA PANTALLA A LA DERECHA 2 VECES
  for ( int i = 0; i < 5; i++ ) {
    lcd.scrollDisplayRight();
    delay (1000);
  }
  // DESPLAZAR LA PANTALLA A LA IZQUIERDA 2 VECES
  for ( int i = 0; i < 5; i++ ) {
    lcd.scrollDisplayLeft();
    delay (1000);
  }
}

```

Mostrar el valor de un potenciómetro en la pantalla LCD 16X2 con Arduino

Otro de los usos comunes de la pantalla LCD de 16x2 es mostrar valores medidos por sensores analógicos. El siguiente programa está diseñado para ser un ejemplo de cómo desplegar el valor medido por un sensor en la pantalla, en este ejemplo estaremos utilizando un potenciómetro conectado a la entrada analógica A0 para simular la entrada del sensor. Se ha comentado el código para que se facilite la comprensión:

```

#include <LiquidCrystal.h>

// AQUI SE CONFIGURAN LOS PINES PARA LA COMUNICACION CON LA PANTALLA
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  // INDICAMOS QUE TENEMOS CONECTADA UNA PANTALLA DE 16X2
  lcd.begin(16, 2);
  // MOVER EL CURSOR A LA PRIMERA POSICION DE LA PANTALLA Y BORRAR (0, 0)
  lcd.clear();
  // IMPRIMIR CADENA EN LA PRIMERA POSICION
  lcd.print(" ELECTROMECHANICA ");
  // ESPERAR UN SEGUNDO
  delay(1000);
}

void loop()
{
  // BORRAMOS TODA LA PANTALLA PARA ACTUALIZARLA CADA SEGUNDO
  lcd.clear();
  // IMPRIMIR UN ENCABEZADO
  lcd.print(" POTENCIOMETRO ");
}

```



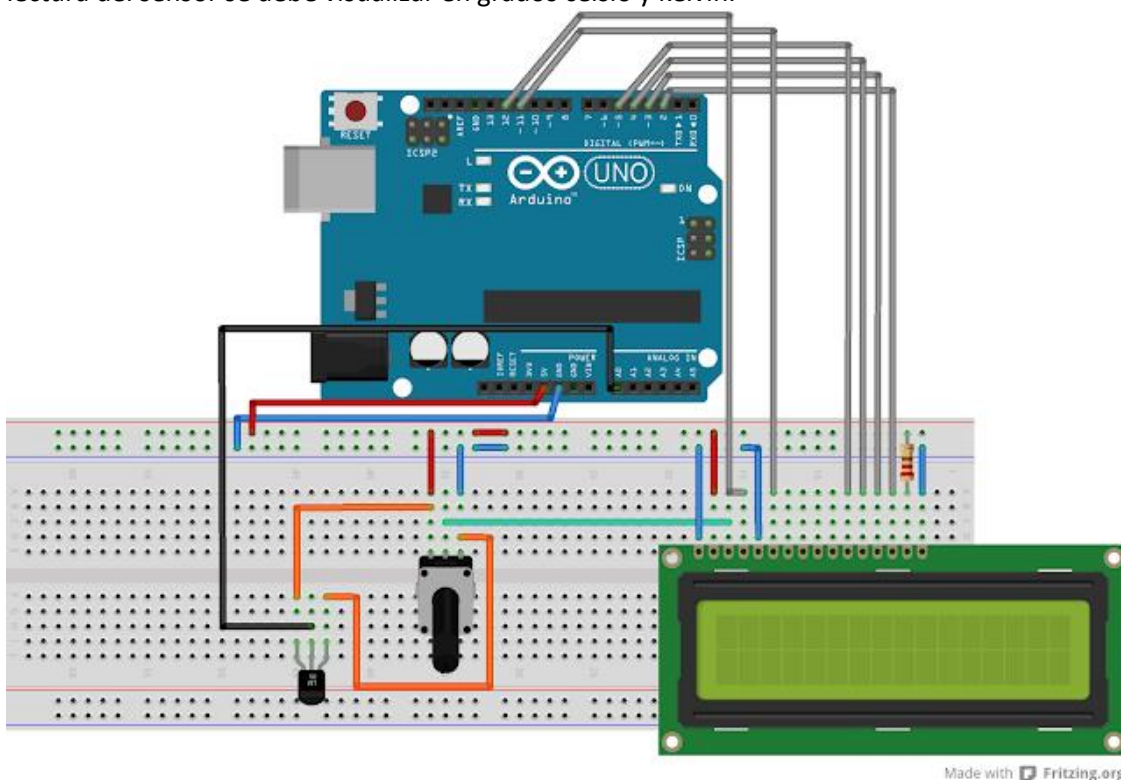
```
// REALIZAR LECTURA ANALOGICA EN PIN A0
unsigned int val = analogRead(A0);
// CONVERTIR ESE VALOR A VOLTAJE (ASUMIENDO QUE EL ARDUINO SE ALIMENTA A 5 VOLTS)
float volts = (val * 5.0) / 1024.0;

// IMPRIMIR VALORES EN LA SEGUNDA LINEA, COMENZANDO POR EL VALOR DIRECTO DEL ADC
lcd.setCursor(0, 1);
lcd.print(val);
// IMPRIMIR EL VALOR EN VOLTAJE, DESPUES DE LA LECTURA DEL ADC
lcd.setCursor(6, 1);
lcd.print(volts, 1);

// ESPERAR UN SEGUNDO ANTES DE CONTUNUAR
delay(1000);
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Utilizar el sensor LM35 para medir la temperatura y luego visualizar en una pantalla LCD 16x2, la lectura del sensor se debe visualizar en grados celsio y kelvin.



LABORATORIO N° 4

CONTROL DE VELOCIDAD Y SENTIDO DE GIRO DE MOTOR DC

➤ **Objetivo.-**

Controlar la velocidad y el sentido de un motor DC usando el microcontrolador Arduino.

➤ **Marco teórico.-**

Motor DC

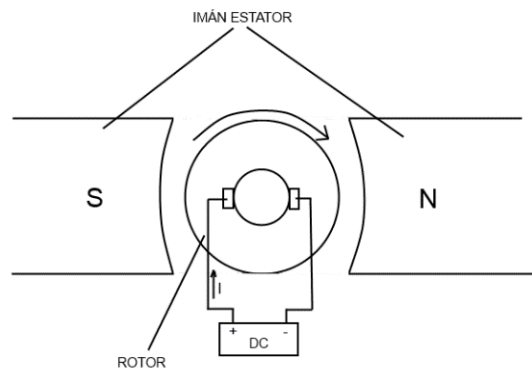
Un motor de corriente continua convierte la energía eléctrica en mecánica. Se compone de dos partes: el estator y el rotor.

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.

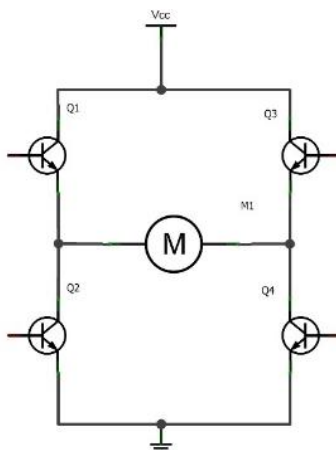
Si queremos cambiar el sentido de giro del rotor, tenemos que cambiar el sentido de la corriente que le proporcionamos al rotor; basta con invertir la polaridad de la pila o batería.



PUENTE H 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.



Modulador de ancho de pulso PWM

La modulación por ancho de pulsos (también conocida como PWM, siglas en inglés de pulse-width modulation) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica para controlar la cantidad de energía que se envía a una carga.

Instrucción analog Write

Instrucción utilizada en arduino para controlar un pin PWM, arduino ya cuenta con una librería interna la cual se utiliza en cuanto el programa localiza la instrucción y la misma librería nos simplifica el trabajo y solo nos pide la cantidad de porcentaje de PWM que deseamos entregar

Sintaxis: `analogWrite(Pin, Valor)` El valor va de 0 a 255 donde 0 es proporcional al 0% y 255 es proporcional al 100%

Instrucción Map

Con la función `map()`, podemos adaptar un valor de un rango determinado a otro con un rango diferente, es decir, podemos “escalar” una señal a nuestra conveniencia.

Sintaxis:

`map(value, fromLow, fromHigh, toLow, toHigh)`

Parámetros:

`value` : el valor a mapear.

`fromLow` : el límite inferior del rango actual del valor.

`fromHigh` : el límite superior del rango actual del valor.

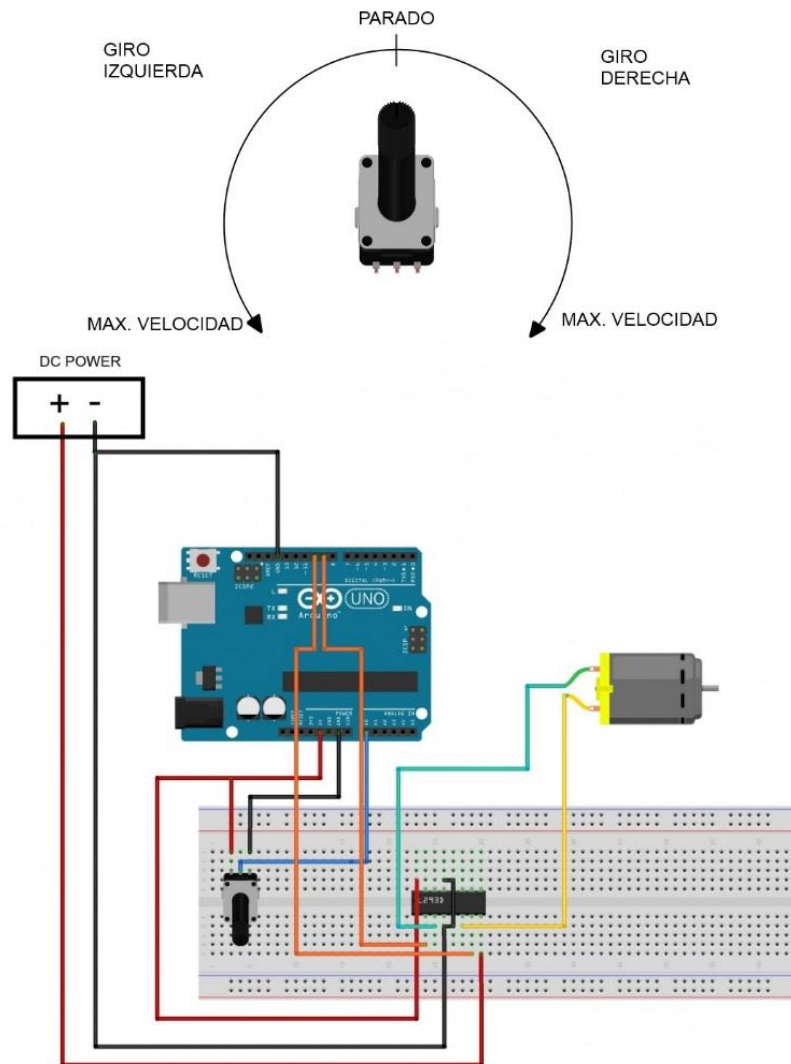
`toLow` : el límite inferior del rango objetivo del valor.

`toHigh` : el límite superior del rango objetivo del valor.

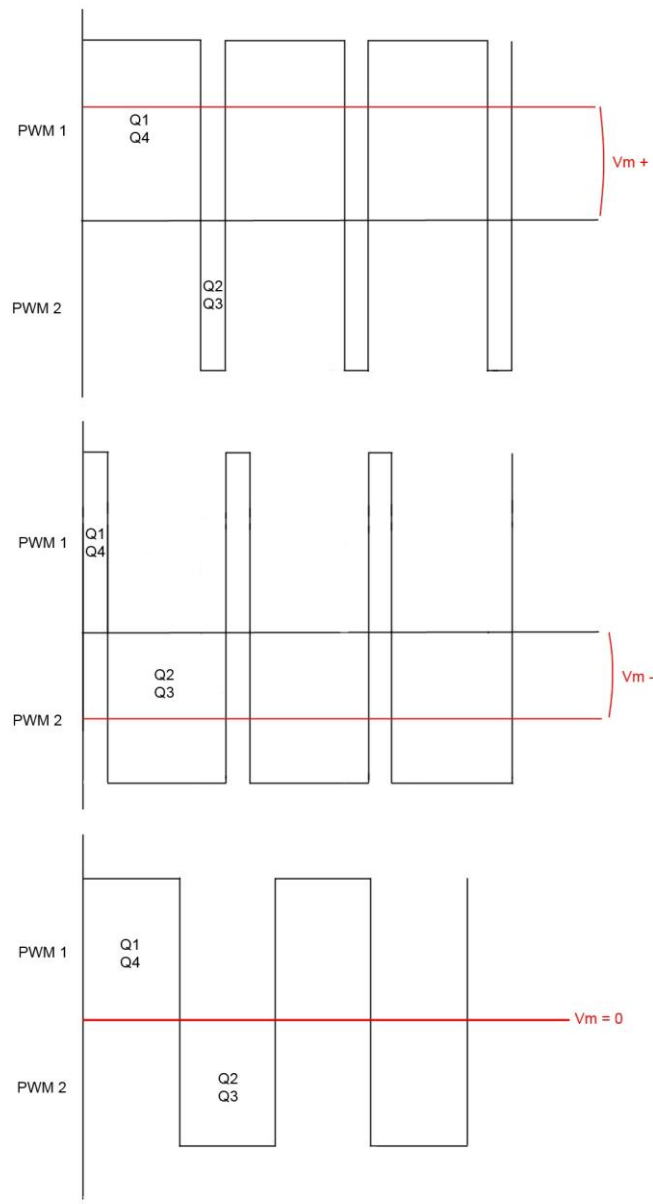
➤ Circuito y Programación.-

Controlar la velocidad y el sentido de un motor DC a través de un potenciómetro desde Arduino.

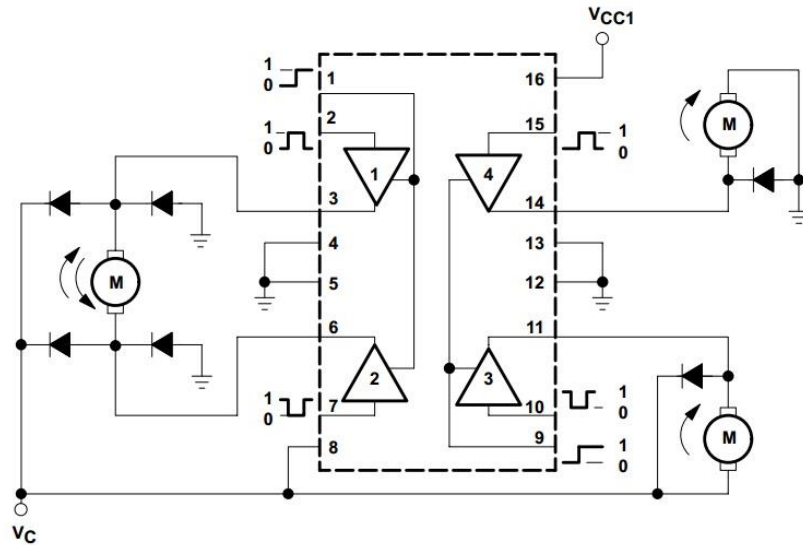
- Cuando el potenciómetro este entre el valor medio y el valor mínimo el motor debe girar en sentido anti horario (ver la gráfica para mayor entendimiento)
- Cuando el potenciómetro este entre el valor medio y el valor máximo el motor debe girar en sentido horario (ver la gráfica para mayor entendimiento)
- Cuando el potenciómetro esté en su valor medio el motor debe estar detenido



Sabemos que hay que atacar los pins 2 y 7 del L293D desde dos salidas del Arduino. En estas dos salidas habrá un PWM a cada una. Pero tenemos que invertir un PWM. ¿Qué quiere decir invertir? Pues que cuando en un PWM tengamos un pulso a un valor alto, en el otro PWM el mismo pulso sea valor bajo. En la imagen lo entenderemos de una manera más gráfica.



Nosotros usaremos la parte de la izquierda (los diodos externos en el L293D están dentro). Como se aprecia en la imagen, los pins 3 y 6 son las salidas y se conectan a los bornes del motor. Y los pins 2 y 7 son las entradas donde conectaremos las salidas del Arduino. Dependiendo que valor ponemos entre los pins 2 y 7 el motor girará en un sentido o en otro.



```

int pin2=9; //Entrada 2 del L293D
int pin7=10; //Entrada 7 del L293D
int pote=A0; //Potenci3metro

int valorpote; //Variable que recoge el valor del potenci3metro
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 potenci3metro en la variable
  valorpote=analogRead(pote);

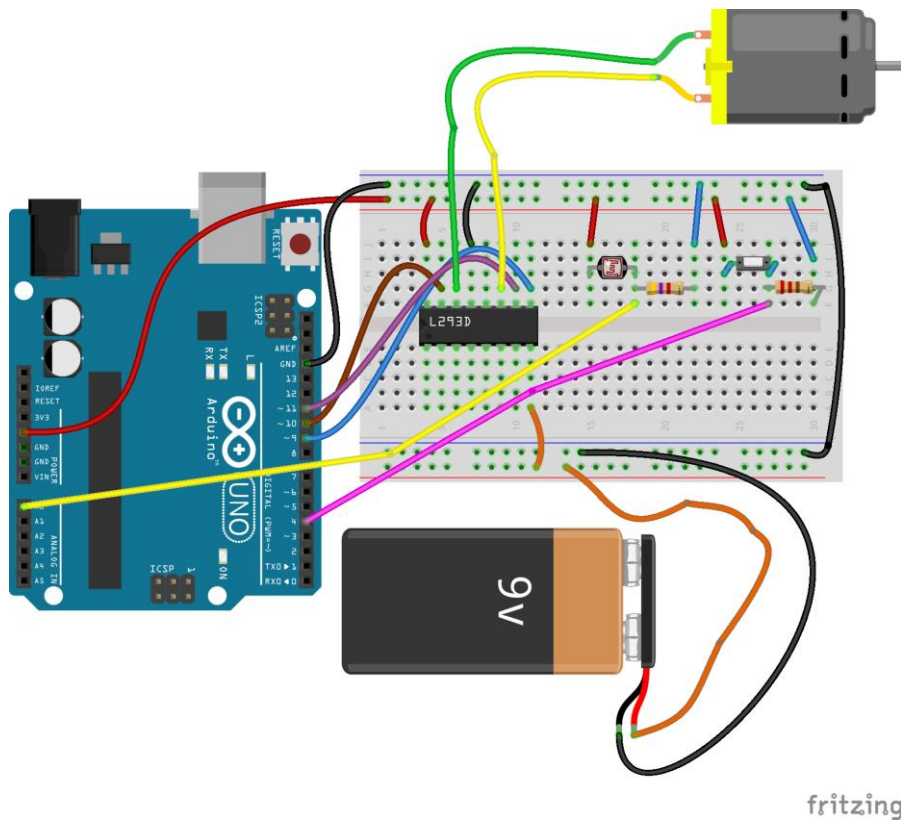
  //Como la entrada anal3gica del Arduino es de 10 bits, el rango va de 0 a 1023.
  //En cambio, la salidas del Arduino son de 8 bits, quiere decir, rango entre 0 a 255.
  //Por esta raz3n tenemos que mapear el n3mero de un rango a otro usando este c3digo.
  pwm1 = map(valorpote, 0, 1023, 0, 255);
  pwm2 = 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);
}

```

PRÁCTICA PARA EL ESTUDIANTE:

Regular la velocidad de un motor por el efecto de luminosidad aplicada a una foto resistencia

- Cuando la fotoresistencia no detecte luz el motor debe estar detenido
- Cuando la fotoresistencia detecte luz el motor debe funcionar (la velocidad del motor dependera de que tanta luz detecte la fotoresistencia)
- Inverir el giro del motor con un pulsador



LABORATORIO N° 5

CONTROL DE MOTOR PASO A PASO

➤ **Objetivo.-**

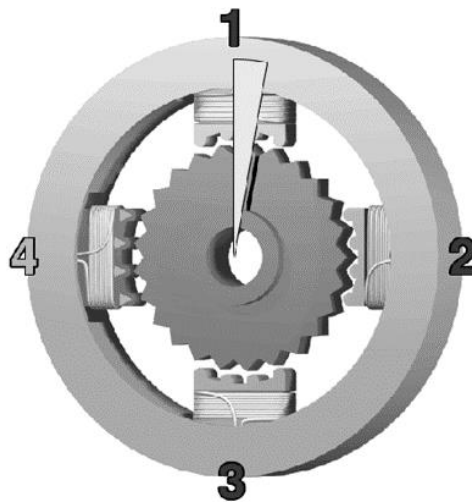
Diseñar e implementar en base al microcontrolador arduino un circuito que me permita el control de giro de un motor PAP.

➤ **Marco teórico.-**

El motor de paso a paso es un motor muy distinto al de corriente continua, ya que como indica su nombre no realiza giros continuos, sino con un paso de un ángulo determinado por el motor.

El ángulo de paso del motor depende del número de bobinas por el que está compuesto, por lo tanto el giro completo depende de la magnitud de este ángulo de paso; por ejemplo si es un motor con 90° por paso, realizará un giro completo en 4 pasos.

Este motor es muy utilizado en los campos de la robótica y automatización ya que permite un control más exacto del movimiento angular.



Paso 1; la bobina 1 esta activada, atrayendo los cuatro dientes superiores imantados del rotor.

Paso 2; la bobina 1 se apaga, y la bobina 2 (derecha) se activa, moviendo los dientes cercanos a la derecha. Resulta una rotación de 3.6° .

Paso 3; De nuevo la bobina 2 se apaga, y la bobina 3 se activa. Resulta otra rotación de 3.6° .

Paso 4; La activación de la bobina 4 permite de nuevo la rotación de 3.6° . Cuando

la bobina 1 se cargue de nuevo, un diente habrá permutado su posición a la derecha; como hay 25 dientes, se necesitarán 100 pasos para un giro completo.

Existen dos tipos de motores paso a paso de imán permanente:

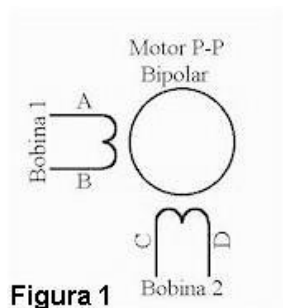


Figura 1

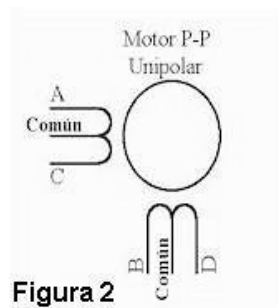
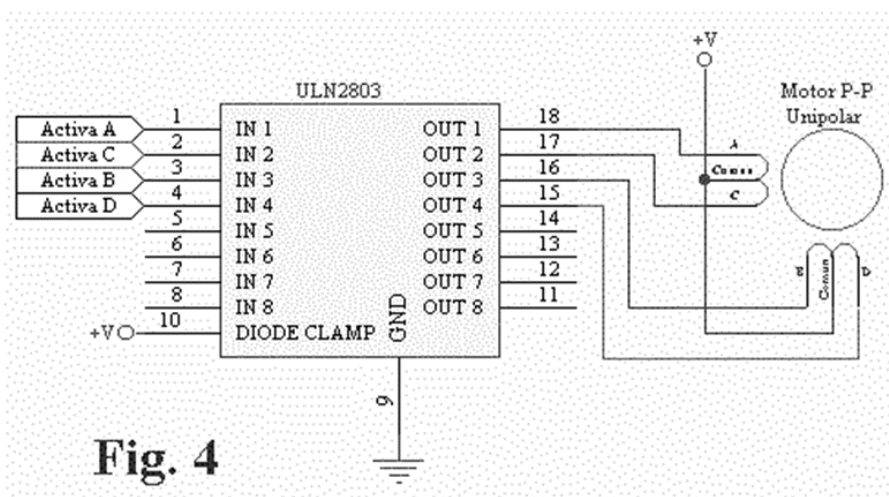


Figura 2

- **Unipolar:** Estos motores suelen tener 6 o 5 cables de salida, dependiendo de su conexionado interno (ver figura 2). Este tipo se caracteriza por ser más simple de controlar. En la figura 4 podemos apreciar un ejemplo de conexionado para controlar un motor paso a paso unipolar mediante el uso de un ULN2803, el cual es una array de 8 transistores tipo Darlington capaces de manejar cargas de hasta 500mA. Las entradas de activación (Activa A, B, C y D) pueden ser directamente activadas por un microcontrolador.

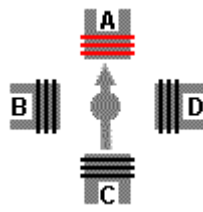


Motor paso a paso 28BYJ-48

- Motor paso a paso con 5 cables (unipolar 4 bobinas)
- Voltaje de funcionamiento 5V o 12V
- Cada paso avanza 5,625°
- Giro completo del rotor requiere de 8 ciclos (1 ciclo = 4 pasos)
- Caja reductora mediante engranajes 1/64
- Se consigue un paso de $5,625/64 = 0,088^\circ$
- Resistencia del bobinado de 50 Ω
- Torque de 34 Newton Metro más o menos 35 gramos por cm
- Frecuencia máxima 100Hz que equivale a un delay de 10 ms

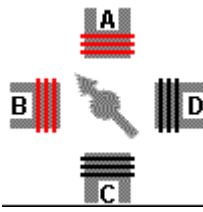
Movimiento de paso completo (menor torque)

Consiste en excitar una bobina cada vez. El consumo se reduce pero también el par, por lo tanto es un consumo y par moderados.

PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	HIGH	LOW	LOW	LOW	
2	LOW	HIGH	LOW	LOW	
3	LOW	LOW	HIGH	LOW	
4	LOW	LOW	LOW	HIGH	

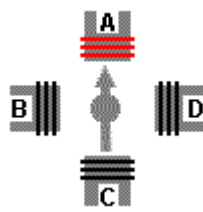
Movimiento paso completo 2 Bobinas (máximo torque)

Consiste en excitar, administrar corriente, a dos bobinas a la vez en cada paso. Se consigue el máximo par pero también es el máximo consumo.

PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	HIGH	HIGH	LOW	LOW	
2	LOW	HIGH	HIGH	LOW	
3	LOW	LOW	HIGH	HIGH	
4	HIGH	LOW	LOW	HIGH	

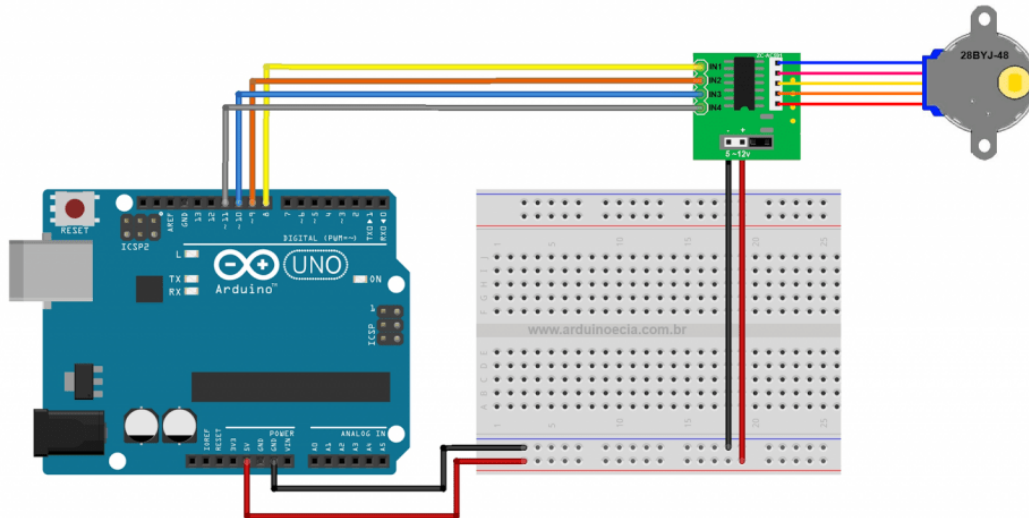
Movimiento de medio paso

Se consigue un movimiento lento y suave con un par y consumo entre medias de los otros dos movimientos.

PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	HIGH	LOW	LOW	LOW	
2	HIGH	HIGH	LOW	LOW	
3	LOW	HIGH	LOW	LOW	
4	LOW	HIGH	HIGH	LOW	
5	LOW	LOW	HIGH	LOW	
6	LOW	LOW	HIGH	HIGH	
7	LOW	LOW	LOW	HIGH	
8	HIGH	LOW	LOW	HIGH	

➤ **Circuito y Programación.-**

Para los siguientes ejemplos de programación se utilizara el siguiente circuito, ya que se explican diferentes formas de realizar la programación.



A) PROGRAMACION MANUAL

EJEMPLO 1- PROGRAMACION MANUAL PASO COMPLETO, ENERGIZANDO DE UNA BOBINA

Programar el microcontrolador arduino para que el motor paso a paso gire una vuelta completa a través del movimiento de paso completo (menor torque) y luego se detenga durante 5 segundos.

- Un ciclo requiere de 4 pasos.
- Un giro completo del rotor requiere 8 ciclos.
- Un giro completo del eje exterior requiere de 64 vueltas (revoluciones) del rotor.

$$4 * 8 * 64 = 2048 \text{ pasos para una revolución}$$

/*

Programa que realiza un giro completo del motor 28BYJ-48 en conjunto con el controlador basado en ULN2003, detiene 5 segundos y luego comienza nuevamente. La secuencia es la de paso completo simple (wave drive) energizando de a una bobina por vez.

*/

```
int IN1 = 8; // pin digital 8 de Arduino a IN1 de modulo controlador
int IN2 = 9; // pin digital 9 de Arduino a IN2 de modulo controlador
int IN3 = 10; // pin digital 10 de Arduino a IN3 de modulo controlador
int IN4 = 11; // pin digital 11 de Arduino a IN4 de modulo controlador
int demora = 20; // demora entre pasos, no debe ser menor a 10 ms.
```

```
void setup() {
  pinMode(IN1, OUTPUT); // todos los pines como salida
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}

void loop() {

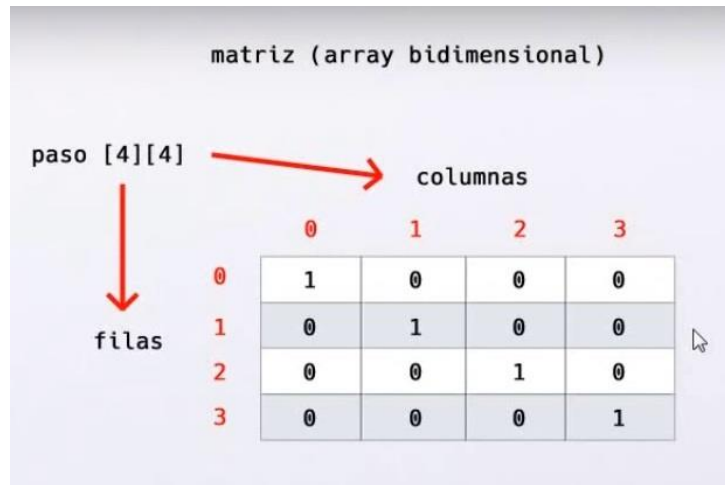
  for (int i = 0; i < 512; i++) // 512*4 = 2048 pasos
  {
    digitalWrite(IN1, HIGH); // paso 1
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    delay(demora);

    digitalWrite(IN1, LOW); // paso 2
    digitalWrite(IN2, HIGH);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    delay(demora);

    digitalWrite(IN1, LOW); // paso 3
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH);
    digitalWrite(IN4, LOW);
    delay(demora);

    digitalWrite(IN1, LOW); // paso 4
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, HIGH);
    delay(demora);
  }

  digitalWrite(IN1, LOW); // detiene por 5 seg.
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  delay(5000);
}
```

EJEMPLO 2 PROGRAMACION MANUAL A TRAVES DE MATRIZ PASO COMPLETO

/*
Programa que realiza un giro completo del motor 28BYJ-48 en conjunto con el controlador basado en ULN2003, detiene 5 segundos y luego comienza nuevamente. La secuencia es la de paso completo simple (wave drive) energizando de a una bobina por vez utilizando una matriz para su definicion.

```
*/
int IN1 = 8;           // pin digital 8 de Arduino a IN1 de modulo controlador
int IN2 = 9;           // pin digital 9 de Arduino a IN2 de modulo controlador
int IN3 = 10;          // pin digital 10 de Arduino a IN3 de modulo controlador
int IN4 = 11;          // pin digital 11 de Arduino a IN4 de modulo controlador
int demora = 20;       // demora entre pasos, no debe ser menor a 10 ms.
```

```
// paso completo simple
int paso [4][4] =     // matriz (array bidimensional) con la secuencia de pasos
```

```
{
  {1, 0, 0, 0},
  {0, 1, 0, 0},
  {0, 0, 1, 0},
  {0, 0, 0, 1}
};
```

```
void setup() {
  pinMode(IN1, OUTPUT);           // todos los pines como salida
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}
```

```
void loop() {

  for (int i = 0; i < 512; i++)    // 512*4 = 2048 pasos
  {
    for (int i = 0; i < 4; i++)    // bucle recorre la matriz de a una fila por vez
    {                              // para obtener los valores logicos a aplicar
      digitalWrite(IN1, paso[i][0]); // a IN1, IN2, IN3 e IN4
    }
  }
}
```

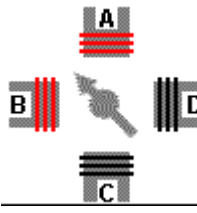
```

    digitalWrite(IN2, paso[i][1]);
    digitalWrite(IN3, paso[i][2]);
    digitalWrite(IN4, paso[i][3]);
    delay(demora);
  }
}

digitalWrite(IN1, LOW);      // detiene por 5 seg.
digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);
digitalWrite(IN4, LOW);
delay(5000);
}

```

EJEMPLO 3 PROGRAMACION MANUAL A TRAVES DE MATRIZ PASO COMPLETO ENERGIZANDO DOS BOBINAS (MAXIMO TORQUE)

PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	HIGH	HIGH	LOW	LOW	
2	LOW	HIGH	HIGH	LOW	
3	LOW	LOW	HIGH	HIGH	
4	HIGH	LOW	LOW	HIGH	

/*
 Programa que realiza un giro completo del motor 28BYJ-48 en conjunto con el controlador basado en ULN2003, detiene 5 segundos y luego comienza nuevamente. La secuencia es la de paso completo con dos bobinas para una maximo torque utilizando una matriz para su definicion.

*/

```

int IN1 = 8;          // pin digital 8 de Arduino a IN1 de modulo controlador
int IN2 = 9;          // pin digital 9 de Arduino a IN2 de modulo controlador
int IN3 = 10;         // pin digital 10 de Arduino a IN3 de modulo controlador
int IN4 = 11;         // pin digital 11 de Arduino a IN4 de modulo controlador
int demora = 20;      // demora entre pasos, no debe ser menor a 10 ms.
// paso completo con maximo torque
int paso [4][4] =     // matriz (array bidimensional) con la secuencia de pasos
{

```

```

    {1, 1, 0, 0},
    {0, 1, 1, 0},
    {0, 0, 1, 1},
    {1, 0, 0, 1}
};

void setup() {
    pinMode(IN1, OUTPUT);          // todos los pines como salida
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
}

void loop() {

    for (int i = 0; i < 512; i++)    // 512*4 = 2048 pasos
    {
        for (int i = 0; i < 4; i++)    // bucle recorre la matriz de a una fila por vez
        {                             // para obtener los valores logicos a aplicar
            digitalWrite(IN1, paso[i][0]); // a IN1, IN2, IN3 e IN4
            digitalWrite(IN2, paso[i][1]);
            digitalWrite(IN3, paso[i][2]);
            digitalWrite(IN4, paso[i][3]);
            delay(demora);
        }
    }

    digitalWrite(IN1, LOW);          // detiene por 5 seg.
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    delay(5000);
}

```

EJEMPLO 4 PROGRAMACION MANUAL A TRAVES DE MATRIZ DE MEDIO PASO

Un ciclo requiere de 8 pasos

Un giro completo del rotor requiere de 8 ciclos

Un giro completo del eje exterior requiere de 64 vueltas (revoluciones del rotor)

$$8 * 8 * 64 = 4096 \text{ paso para una revolucion}$$

/*

Programa que realiza un giro completo del motor 28BYJ-48 en conjunto con el controlador basado en ULN2003, detiene 5 segundos y luego comienza nuevamente. La secuencia es la de medio paso para maxima precision y torque medio utilizando una matriz para su definicion.

*/

```
int IN1 = 8;          // pin digital 8 de Arduino a IN1 de modulo controlador
```

```

int IN2 = 9;           // pin digital 9 de Arduino a IN2 de modulo controlador
int IN3 = 10;          // pin digital 10 de Arduino a IN3 de modulo controlador
int IN4 = 11;          // pin digital 11 de Arduino a IN4 de modulo controlador
int demora = 20;       // demora entre pasos, no debe ser menor a 10 ms.
// medio paso
int paso [8][4] =      // matriz (array bidimensional) con la secuencia de pasos
{
  {1, 0, 0, 0},
  {1, 1, 0, 0},
  {0, 1, 0, 0},
  {0, 1, 1, 0},
  {0, 0, 1, 0},
  {0, 0, 1, 1},
  {0, 0, 0, 1},
  {1, 0, 0, 1}
};

void setup() {
  pinMode(IN1, OUTPUT);           // todos los pines como salida
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}

void loop() {
  for (int i = 0; i < 512; i++)    // 512*8 = 4096 pasos
  {
    for (int i = 0; i < 8; i++)    // bucle recorre la matriz de a una fila por vez
    {                             // para obtener los valores logicos a aplicar
      digitalWrite(IN1, paso[i][0]); // a IN1, IN2, IN3 e IN4
      digitalWrite(IN2, paso[i][1]);
      digitalWrite(IN3, paso[i][2]);
      digitalWrite(IN4, paso[i][3]);
      delay(demora);
    }
  }

  digitalWrite(IN1, LOW);         // detiene por 5 seg.
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  delay(5000);
}

```


B) PROGRAMACION A TRAVES DE LIBRERÍA**Stepper(steps, pin1, pin2, pin3, pin4)**

Con esta función podemos crear una instancia u objeto de la clase Stepper, que representa un motor paso a paso particular conectado a nuestra placa Arduino. Los parámetros que recibe esta función son: el número de pasos que tiene que dar nuestro motor para completar una revolución (steps) y los pines (pin1, pin2, pin3 y pin4) que emplearemos para controlarlo. Para el motor unipolar 28BYJ-48, debemos crear un objeto de la siguiente forma:

```
Stepper motor1(2048, 8, 9, 10, 11);
```

SetSpeed(rpm)

Esta función nos sirve únicamente para ajustar la velocidad de giro del eje de nuestro motor. El parámetro que recibe dicha función es un número entero, el cual representa la velocidad de rotación (en RPM) que deseamos darle a nuestro motor. Por lo tanto no es una función que retorne ningún valor.

Aunque esta función acepta que se le ingrese cualquier número entero, debemos respetar la frecuencia máxima admisible entre pulsos dada por el fabricante del motor. Para el caso en concreto del 28BYJ-48 este valor es de 100 Hz, es decir un delay o demora entre pulsos de $1/100=0.01$ segundos o 10 milisegundos. Con la finalidad de asegurarnos de que el valor que ingresemos a la función setSpeed cumpla con esta especificación podemos realizar el siguiente cálculo:

1 RPM: 1 vuelta/min = 1 vuelta cada 60 seg, pero para dar una vuelta necesita dar 2048 pasos

$$\begin{aligned} 60/2048 &= 0,02929 \text{ seg} * 1000 \\ &= 29,3 \text{ milisegundos} \end{aligned}$$

2 RPM: 2 vuelta/min = 1 vuelta cada 30 seg, pero para dar una vuelta necesita dar 2048 pasos

$$\begin{aligned} 30/2048 &= 0,01464 \text{ seg} * 1000 \\ &= 14,6 \text{ milisegundos} \end{aligned}$$

3 RPM: 3 vuelta/min = 1 vuelta cada 20 seg, pero para dar una vuelta necesita dar 2048 pasos

$$\begin{aligned} 20/2048 &= 0,00976 \text{ seg} * 1000 \\ &= 9,8 \text{ milisegundos} = 10 \text{ milisegundos} \end{aligned}$$

Step(steps)

Esta función manda a girar el motor una cantidad específica de pasos igual al valor del parámetro steps ingresado en la misma. La velocidad de rotación en cada paso estará determinada por la llamada más reciente de la función setSpeed(). Si queremos invertir el

sentido de giro de nuestro motor basta con colocar la cantidad de pasos precedida por el signo menos. Por ejemplo, si quisiéramos que nuestro 28BYJ-48 diera un cuarto de vuelta en sentido contrario al original debemos hacer la siguiente llamada

```
setSpeed(-512);
```

EJEMPLO 5 PROGRAMACION A TRAVES DE LIBRERIA

```
/*
Programa que utiliza la Libreria Stepper para el control de un motor unipolar
de 5 hilos modelo 28BYJ-48 y su correspondiente driver ULN2003. Se establece una
velocidad de 2 RPM y un total de 512 pasos equivalentes a un cuarto de vuelta.
*/

#include <Stepper.h>    // incluye libreria stepper
Stepper motor1(2048, 8, 10, 9, 11); // pasos completos definimos nuestro motor

void setup() {
  motor1.setSpeed(2);    // en RPM (valores de 1, 2 o 3 para 28BYJ-48)
}

void loop() {
  motor1.step(512);      // cantidad de pasos
  delay(2000);           // demora de 2 seg. por cuestiones practicas
}
```

EJEMPLO 6 PROGRAMACION A TRAVES DE LIBRERIA E INVERSIÓN DE GIRO

```
/*
Programa que utiliza la Libreria Stepper para el control de un motor unipolar
de 5 hilos modelo 28BYJ-48 y su correspondiente driver ULN2003. Se establece una
velocidad de 2 RPM y un total de 512 pasos equivalentes a un cuarto de vuelta.
Luego con una cantidad negativa de pasos se produce el giro tambien de un cuarto
de vuelta pero en sentido opuesto.
*/

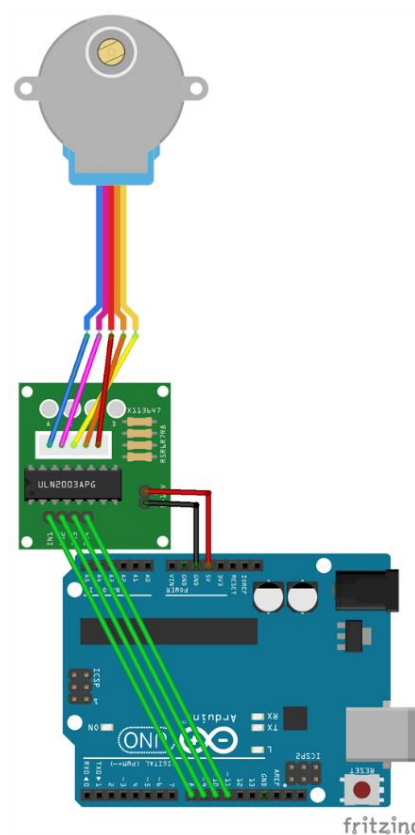
#include <Stepper.h>    // incluye libreria stepper
Stepper motor1(2048, 8, 10, 9, 11); // pasos completos

void setup() {
  motor1.setSpeed(2);    // en RPM (valores de 1, 2 o 3 para 28BYJ-48)
}

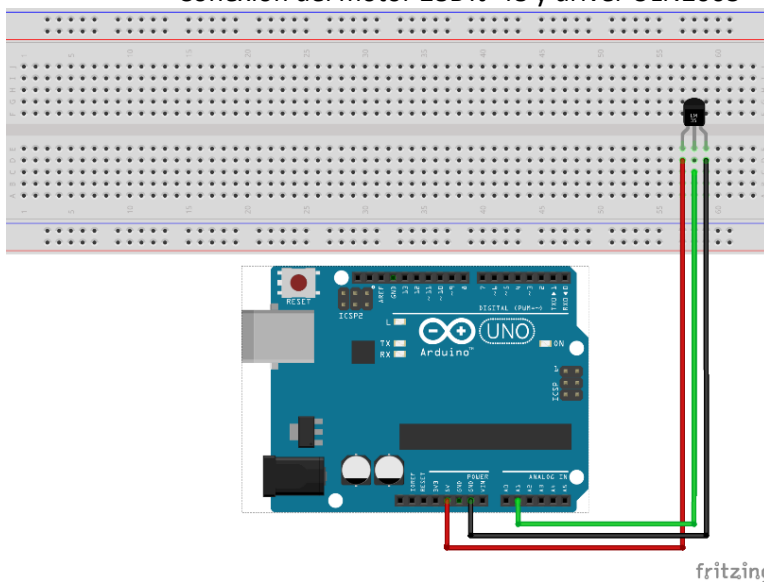
void loop() {
  motor1.step(512);      // cantidad de pasos
  delay(2000);           // demora de 2 seg. por cuestiones practicas
  motor1.step(-512);     // signo menos indica giro en sentido opuesto
  delay(2000);           // demora de 2 seg. por cuestiones practicas
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Medir la temperatura con un sensor LM35 y controlar un motor paso a paso, si la temperatura es mayor a 30º gire en sentido horario media vuelta y se detenga 2 segundos y si es menor a 30º gire en sentido anti horario media vuelta y se detenga 2 segundos, visualizar la temperatura en el monitor serial.



Conexión del motor 28BYJ-48 y driver ULN2003



Sensor LM35

LABORATORIO N° 6

SERVOMOTOR

➤ **Objetivo.-**

Conocer el funcionamiento y el manejo de un servomotor.

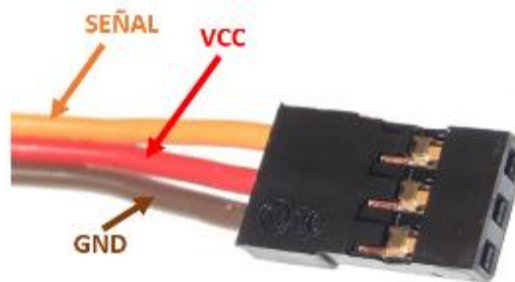
➤ **Marco teórico.-**

Servomotor

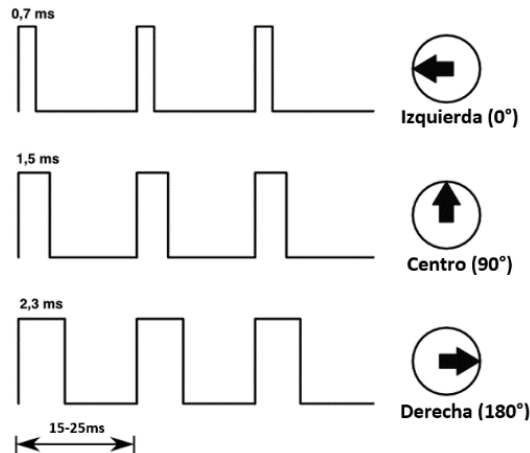
Un servomotor o comúnmente llamado servo, es un motor DC con la capacidad de ubicar su eje en una posición o ángulo determinado, internamente tiene una caja reductora la cual le aumenta el torque y reduce la velocidad, un potenciómetro encargado de sensar la posición del eje y una pequeña tarjeta electrónica que junto al potenciómetro forman un control de lazo cerrado.

En el mercado existen diferentes modelos de servomotores, la principal diferencia entre ellos es el torque. Este punto es bueno tenerlo claro para elegir el servomotor adecuado. Es mejor elegir un servo con torque superior al que requerimos, pues el consumo de corriente es proporcional a la carga. En cambio sí sometemos un servomotor a cargas superiores a su torque, corremos el riesgo de malograr tanto la parte mecánica (engranes) y eléctrica del servo, incluso podemos generar ruido en la fuente.

Todos los servos usados para robótica, tiene un conector de 3 cables. VCC (rojo), GND (Marrón) y Señal (Naranja):



La señal o dato que hay que enviarle al servo es una señal de PWM donde el tiempo en alto es equivalente al ángulo o posición del servo. Estos valores pueden variar y van desde 0.5 a 1 milisegundo para la posición 0° y 2 a 2.4 milisegundos para la posición de 180°, el periodo de la señal debe ser cercano a 20 milisegundos.



Librería servo de Arduino

El IDE Arduino trae una librería completa para el control de servomotores, la documentación completa lo encontramos en su página oficial: Servo Library

A continuación se muestran las funciones de la librería Servo:

attach(Pin)

Establece el pin indicado en la variable servo. Ej: `servo.attach(2);`

attach(Pin,min,max)

Establece el pin indicado en la variable servo, considerando min el ancho de pulso para la posición 0° y max el ancho de pulso para 180°. Ej: `servo.attach(2,900,2100);`

write(angulo)

Envía la señal correspondiente al servo para que se ubique en el ángulo indicado, ángulo es un valor entre 0 y 180°. Ej: `servo.write(45);`

writeMicroseconds(tiempo)

Envía al servo el ancho de pulso=tiempo en microsegundos. Ej:
`servo.writeMicroseconds(1500);`

read ()

Lee la posición actual del servo, devuelve un valor entre 0 y 180. Ej: `angulo=read () ;`

attached(Pin)

Verifica si la variable servo está unido al pin indicado, devuelve true o false. Ej:
`if(attached(2))`

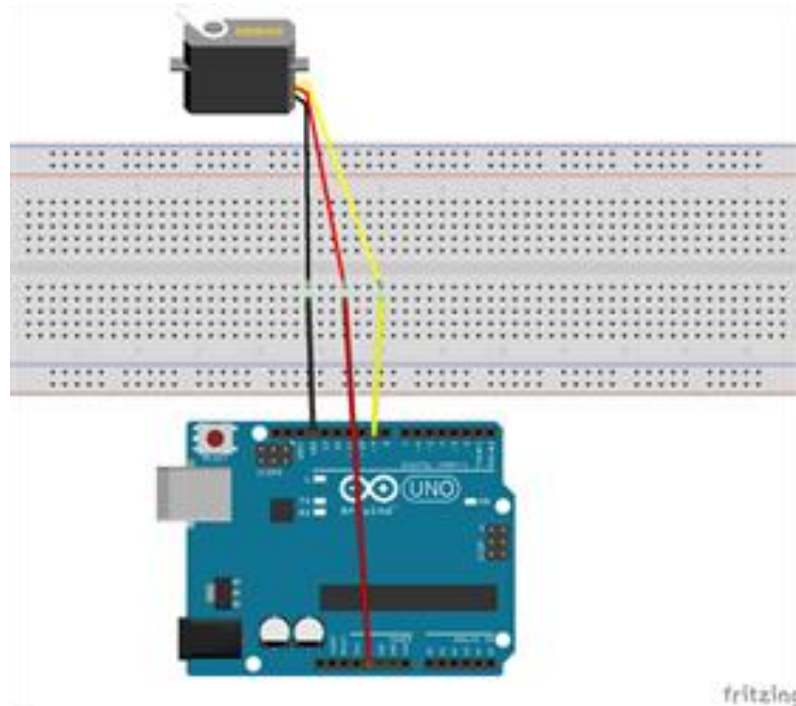
detach(pin)

Separa la variable Servo del pin indicado. Ej: servo.detach(2);

➤ **Circuito y Programación.-**

Ejemplo 1

Programar el arduino para que el servomotor se establezca en 3 posiciones en los ángulos 0º, 90º y 180º, en cada posición se debe detener 1 segundo.



```
// Incluimos la librería para poder controlar el servo
#include <Servo.h>
// Declaramos la variable para controlar el servo
Servo servoMotor;
void setup() {
  // Iniciamos el monitor serie para mostrar el resultado
  Serial.begin(9600);
  // Iniciamos el servo para que empiece a trabajar con el pin 9
  servoMotor.attach(9);
}

void loop() {
  // Desplazamos a la posición 0º
  servoMotor.write(0);
  // Esperamos 1 segundo
  delay(1000);
```

```
// Desplazamos a la posición 90º
servoMotor.write(90);
// Esperamos 1 segundo
delay(1000);

// Desplazamos a la posición 180º
servoMotor.write(180);
// Esperamos 1 segundo
delay(1000);
}
```

Ejemplo 2 Girando grado a grado el servomotor

Realizar un barrido con el servomotor de 0º a 180 º y luego de 180º a 0º.

```
// Incluimos la librería para poder controlar el servo
#include <Servo.h>
// Declaramos la variable para controlar el servo
Servo servoMotor;

void setup() {
  // Iniciamos el monitor serie para mostrar el resultado
  Serial.begin(9600);

  // Iniciamos el servo para que empiece a trabajar con el pin 9
  servoMotor.attach(9);
  // Inicializamos al ángulo 0 el servomotor
  servoMotor.write(0);
}

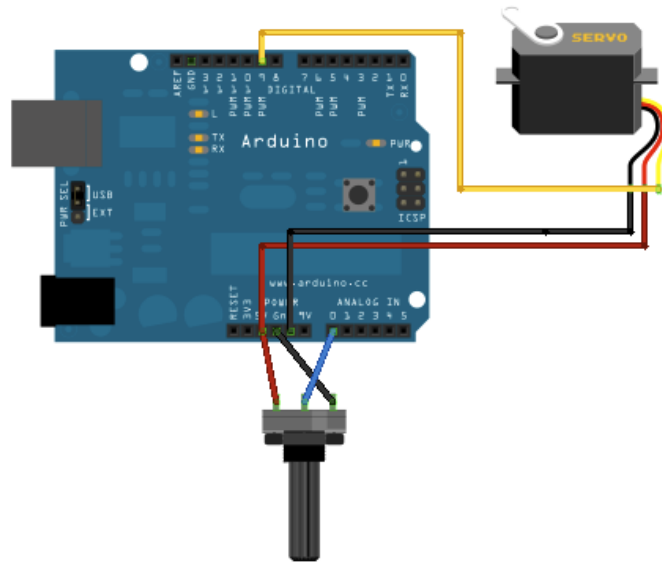
void loop() {
  // Vamos a tener dos bucles uno para mover en sentido positivo y otro en sentido
  negativo
  // Para el sentido positivo
  for (int i = 0; i <= 180; i++)
  {
    // Desplazamos al ángulo correspondiente
    servoMotor.write(i);
    // Hacemos una pausa de 25ms
    delay(25);
  }
  // Para el sentido negativo
  for (int i = 179; i > 0; i--)
```

```

{
  // Desplazamos al ángulo correspondiente
  servoMotor.write(i);
  // Hacemos una pausa de 25ms
  delay(25);
}
}

```

Ejemplo 3 Control de un servomotor a través de un potenciómetro:



```

#include <Servo.h>
Servo myservo; //creamos un objeto servo
void setup()
{
  myservo.attach(9); // asignamos el pin 9 al servo.
  Serial.begin(9600);
}

void loop()
{
  int adc = analogRead(A0); // realizamos la lectura del potenciómetro
  int angulo = map(adc, 0, 1023, 0, 180); // escalamos la lectura a un valor entre 0 y 180
  myservo.write(angulo); // enviamos el valor escalado al servo.
  Serial.print("Angulo: ");
  Serial.println(angulo);
  delay(10);
}

```


Ejemplo 4 Control de un servomotor a través del puerto serial

```
#include <Servo.h>
int posicion = 0;
Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

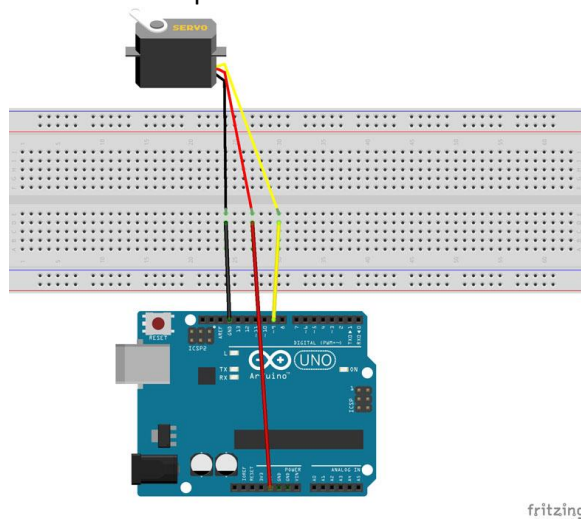
void setup() {
  Serial.begin(9600);
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
  Serial.println("Escribe la posición donde mover el servo (0 - 180)");
}

void loop() {

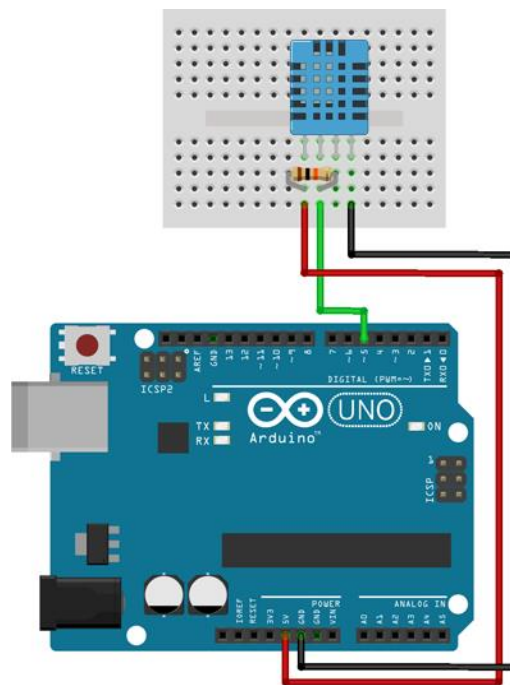
  if (Serial.available() > 0) {
    String grados = "";
    do {
      grados = grados + (char)Serial.read();
      //Serial.println(grados);
      delay(5);
    }
    while (Serial.available() > 0);
    Serial.println("Movido a la posición: " + (String)grados);
    posicion = grados.toInt();
    myservo.write(posicion);
    Serial.println("Escribe la posición donde mover el servo (0 - 180)");
  }
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Medir la temperatura con un sensor DHT 11/DHT22 y controlar un servomotor, estando en posición inicial ángulo "0" si la temperatura es mayor a 30° el servomotor gire en la posición de 180° y si es menor a 30° gire 90°, visualizar la temperatura en el monitor serial.



Conexion del Sermotor



Conexión del sensor

LABORATORIO N° 7

SENSOR DE ULTRASONIDOS

➤ **Objetivo.-**

Conocer el sensor de ultrasonidos, su funcionamiento y programación en arduino.

➤ **Marco teórico.-**

Sensor de ultrasonidos

Un sensor de ultrasonidos es un dispositivo para medir distancias. Su funcionamiento se basa en el envío de un pulso de alta frecuencia, no audible por el ser humano. Este pulso rebota en los objetos cercanos y es reflejado hacia el sensor, que dispone de un micrófono adecuado para esa frecuencia.

Midiendo el tiempo entre pulsos, conociendo la velocidad del sonido, podemos estimar la distancia del objeto contra cuya superficie impacta el impulso de ultrasonidos

El rango de medición teórico del sensor HC-SR04 es de 2cm a 400 cm, con una resolución de 0.3cm. En la práctica, sin embargo, el rango de medición real es mucho más limitado, en torno a 20cm a 2 metros.

¿Cómo funciona un sensor de ultrasonidos?

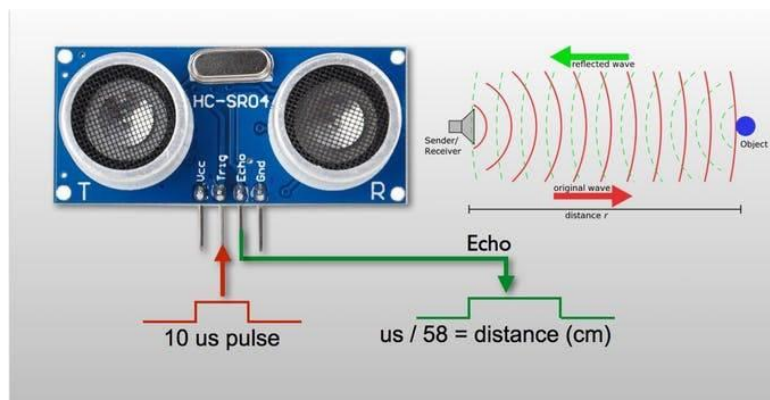
El sensor se basa simplemente en medir el tiempo entre el envío y la recepción de un pulso sonoro. Sabemos que la velocidad del sonido es 343 m/s en condiciones de temperatura 20 °C, 50% de humedad, presión atmosférica a nivel del mar. Transformando unidades resulta

$$343 \frac{m}{s} * 100 \frac{cm}{m} * \frac{1 s}{1000000 us} = \frac{1cm}{29,2us}$$

Es decir, el sonido tarda 29,2 microsegundos en recorrer un centímetro. Por tanto, podemos obtener la distancia a partir del tiempo entre la emisión y recepción del pulso mediante la siguiente ecuación.

$$Distancia (cm) = \frac{Tiempo (us)}{29,2 * 2}$$

El motivo de dividir por dos el tiempo (además de la velocidad del sonido en las unidades apropiadas, que hemos calculado antes) es porque hemos medido el tiempo que tarda el pulso en ir y volver, por lo que la distancia recorrida por el pulso es el doble de la que queremos medir.

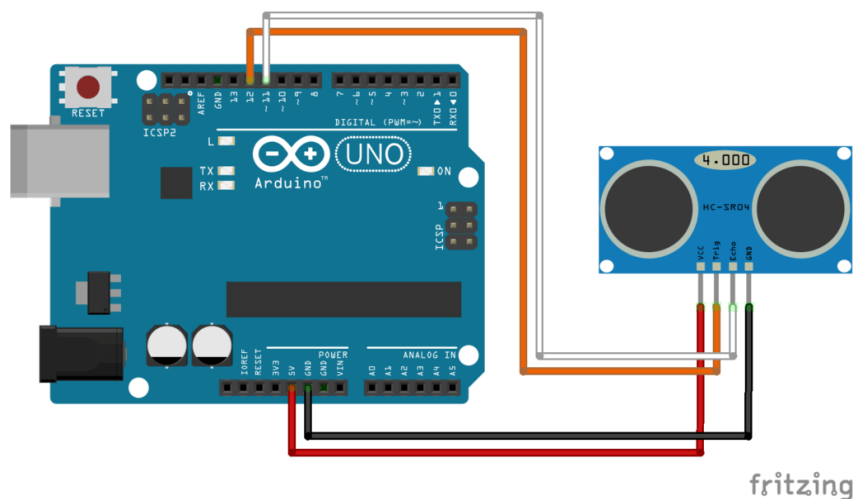


¿Cómo se comunica con Arduino el sensor HC-SR04?

La interfaz del sensor HC-SR04 y arduino se logra mediante 2 pines digitales: el pin de disparo (trigger) y eco (echo). La función de cada uno de estos pines es la siguiente:

El pin trigger recibe un pulso de habilitación de parte del microcontrolador, mediante el cual se le indica al módulo que comience a realizar la medición de distancia.

En el pin echo el sensor devuelve al microcontrolador un pulso cuyo ancho es proporcional al tiempo que tarda el sonido en viajar del transductor al obstáculo y luego de vuelta al módulo.



Funcion pulsein()

Lee un pulso (HIGH o LOW) en un pin. Por ejemplo, si el valor es alto, pulsein () espera a que el pin pase a nivel HIGH, se inicia el tiempo, espera a que el pin pase a nivel LOW y para el cronómetro. Devuelve la longitud del impulso en microsegundos o 0 si no se recibe un pulso completo dentro del tiempo de espera.

Funciona en pulsos de 10 microsegundos a 3 minutos de duración. Por otra parte la resolución más alta se obtiene con intervalos cortos.

Sintaxis

`pulseIn (pin, value)`

`pulseIn (pin, value, timeout)`

Parámetros

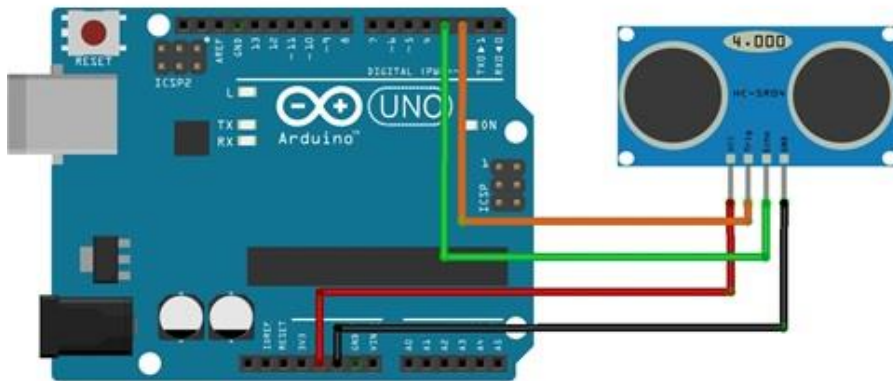
- `pin`: el número de pin en el que desea leer el pulso. (int)
- `value`: tipo de pulso a leer: HIGH o LOW. (int)
- `timeout` (opcional): el número de microsegundos que espera a que el pulso se complete: la función devuelve 0 si el pulso completo no se recibe dentro del tiempo de espera. Por defecto es de un segundo (unsigned long).

`timeout`, si se usa, es recomendable que sea, al menos, 1,3 veces superior a la duración del pulso a medir. Por ejemplo, si se mide un pulso de duración 0,01 segundo, `timeout` deber ser al menos 13000.

➤ Circuito y Programación.-

Realiza la medición de distancia con un sensor ultrasónico HC-SR04 conectado al arduino

a. Programación sin librería



```
const int Trigger = 2; //Pin digital 2 para el Trigger del sensor
const int Echo = 3; //Pin digital 3 para el Echo del sensor
```

```
void setup() {
  Serial.begin(9600); //iniciailzamos la comunicación
  pinMode(Trigger, OUTPUT); //pin como salida
  pinMode(Echo, INPUT); //pin como entrada
  digitalWrite(Trigger, LOW); //Inicializamos el pin con 0
}
void loop()
```

```

{
  long t; //timepo que demora en llegar el eco
  long d; //distancia en centimetros

  digitalWrite(Triquer, HIGH);
  delayMicroseconds(10);    //Enviamos un pulso de 10us
  digitalWrite(Triquer, LOW);

  t = pulseIn(Echo, HIGH); //obtenemos el ancho del pulso
  d = t/59;                //escalamos el tiempo a una distancia en cm

  Serial.print("Distancia: ");
  Serial.print(d);    //Enviamos serialmente el valor de la distancia
  Serial.print("cm");
  Serial.println();
  delay(100);        //Hacemos una pausa de 100ms
}

```

b. Programación con librería

A continuación se detallan algunos comandos más usados de la librería

- **Sonar NewPing (trigger_pin, echo_pin [, max_cm_distance]);**

Ejemplo:

```
Sonar NewPing (12, 11, 200);
```

Esto inicializa NewPing para usar el pin 12 para la salida del disparador, el pin 11 para la entrada de eco, con una distancia de ping máxima de 200 cm.

- **sonar.ping ();** - Enviar un ping, devuelve el tiempo de eco en microsegundos o 0 (cero) si no hay eco de ping dentro del límite de distancia establecido
- **sonar.ping_in ();** - Enviar un ping, devuelve la distancia en pulgadas o 0 (cero) si no hay eco de ping dentro del límite de distancia establecido
- **sonar.ping_cm ();** - Enviar un ping, devuelve la distancia en centímetros o 0 (cero) si no hay eco de ping dentro del límite de distancia establecido

```

// -----
// Medir distancia aproximadamente 20 veces por segundo.
// -----

```

```
#include <NewPing.h>
```

```

#define TRIGGER_PIN 2 // pin del arduino conectado al pin trigger del sensor ultrasonico.
#define ECHO_PIN 3 // pin del arduino conectado al pin echo del sensor ultrasonico.
#define MAX_DISTANCE 200 // Distancia máxima que queremos hacer ping (en centímetros). La
distancia máxima del sensor es de 400-500 cm.

```

```
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // configuración del nombre del sensor,
pines y distancia máxima.
```

```

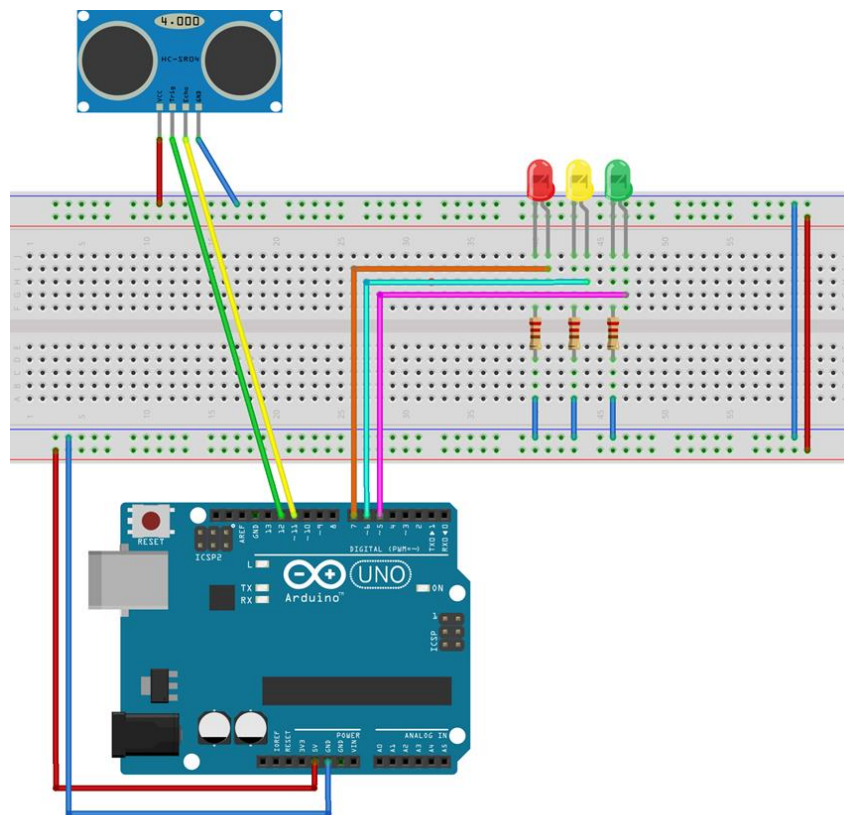
void setup() {
  Serial.begin(115200); //
}

void loop() {
  delay(50);           // Esperar 50 ms entre pings (unos 20 pings / seg). 29ms debe ser el menor
retraso entre pings.
  Serial.print("Ping: ");
  Serial.print(sonar.ping_cm()); // Enviar ping, obtener la distancia en cm e imprimir el resultado (0 =
fuera del rango de distancia establecido)
  Serial.println("cm");
}

```

PRÁCTICA PARA EL ESTUDIANTE:

Medir distancia de forma cualitativa (con los colores de tres leds) mediante el sensor de ultrasonidos. el LED rojo debe encender si algun objeto está a menos de 20 centímetros, amarillo si está entre 20 y 40 centímetros, y verde si está a más de 40 centímetros, además de mostrar la distancia en una pantalla LCD.



fritzing

LABORATORIO N° 8

SENSOR DE COLOR

➤ **Objetivo.-**

Diseñar e implementar en base al microcontrolador arduino la detección de color mediante el sensor TCS230

➤ **Marco teórico.-**

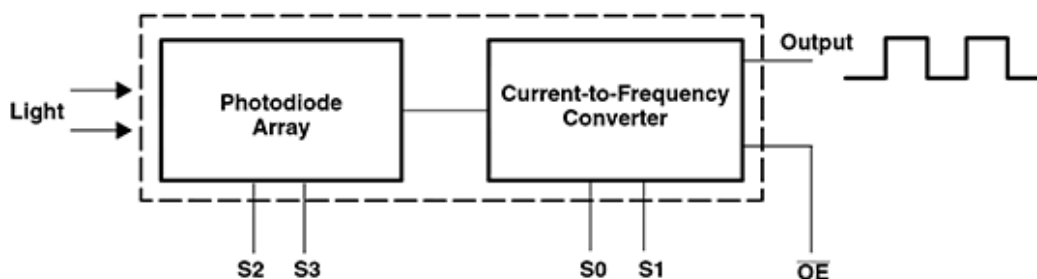
Sensor De Color TCS230

EL TCS230 es un sensor óptico que permite detectar el color de un objeto ubicado en frente de él. Podemos conectarlo este sensor con facilidad a un autómata o procesador como Arduino.

Internamente, el TCS230 está formado por una matriz de fotodiodos de silicón junto con un conversor de frecuencia, en un único integrado CMOS.

La matriz dispone de 8 x 8 fotodiodos de 110 μm , de los cuales 16 tienen filtros azules, 16 verdes, 6 rojos, y 16 no tienen filtro. Los fotodiodos están distribuidos de forma que minimizan el efecto la incidencia no uniforme de la luz.

La salida del TCS3200 es una onda cuadrada del 50% duty, cuya frecuencia es proporcional a la intensidad luminosa. La tensión de alimentación del sensor es de 2.7V a 5.5V.



Frecuentemente el TCS230 se distribuye en módulos que incorporan dos o cuatro LED de luz blanca y un protector de plástico. El objetivo de ambas medidas es minimizar los efectos de la iluminación ambiente en la medición.

Pese a sus especificaciones y elementos para eliminar la luz ambiente, el TCS230 no es capaz de medir de forma precisa el color RGB de un objeto, o la temperatura de color de una fuente luminosa.

Sin embargo, podemos emplearlo para distinguir entre colores básicos. Por ejemplo, podemos emplearlo para reconocer el color de una tarjeta o un objeto, y guiar a un robot en un recorrido

El TCS230 tiene cuatro entradas digitales S0, S1, S2, y S3, y una salida digital Out. Para conectarlo a Arduino necesitaremos emplear al menos 3 pines digitales.

En primer lugar debemos alimentar el módulo conectando los pines Gnd y Vcc del TCS230, respectivamente, a Gnd y Vcc de Arduino.

Los pines S0 y S1 controlan la frecuencia de la salida y la desactivación del módulo. Los conectamos a dos salidas digitales, o podemos conectarlas a 5V si no queremos poder apagar el módulo.

	Power Down	2%	20%	100%
S0	Low	Low	High	High
S1	Low	High	Low	High

Por otra parte, los pines S2 y S3 seleccionan el color a medir. Deberemos conectarlos a dos Salidas digitales de Arduino.

	Red	Blue	Clear	Green
S2	Low	Low	High	High
S3	Low	High	Low	High

Finalmente, conectamos la salida del sensor Out a una entrada digital de Arduino.

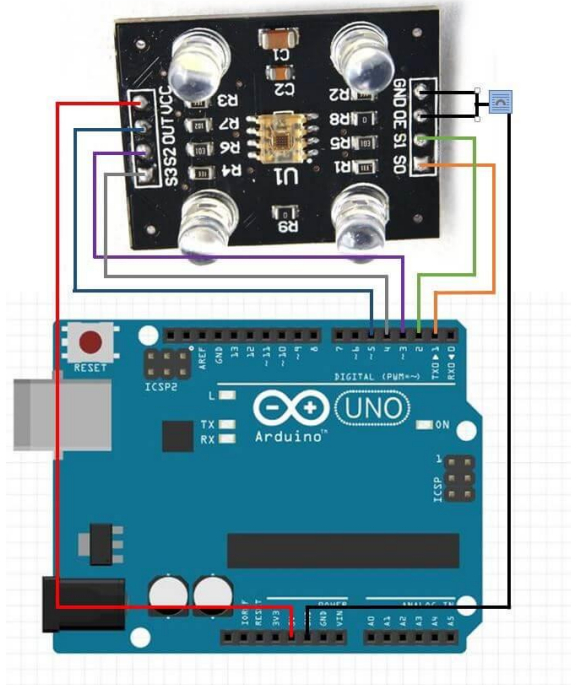
➤ **Circuito y Programación.-**

Ejemplo 1 Lectura de la frecuencia de cada color

Lectura y visualización de la frecuencia de salida en el monitor serie. En esta parte, anotaremos los valores de frecuencia al colocar diferentes colores frente al sensor.

Primeramente se coloca un color en frente del sensor (aproximadamente 3 cm) y veremos que la frecuencia que muestre en el monitor serial será la menor de los tres colores RGB, por ejemplo nosotros ponemos el color rojo vemos que su lectura es menor en comparación de los 3 colores (36). Este valor nos servirá para usar como condicionante en la siguiente programación para ya detectar el color

Siempre se debe hacer lectura de las frecuencias ya que existen varias tonalidades de cada color, y los valores usados por cada estudiante siempre serán distintos



```
// Cableado de TCS3200 a Arduino
//
#define S0 1
#define S1 2
#define S2 3
#define S3 4
#define salidaSensor 5

// Para guardar las frecuencias de los fotodiodos
int frecuenciaRojo = 0;
int frecuenciaVerde = 0;
int frecuenciaAzul = 0;

void setup() {
  // Definiendo las Salidas
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);

  // Definiendo salidaSensor como entrada
  pinMode(salidaSensor, INPUT);

  // Definiendo la escala de frecuencia a 20%
  digitalWrite(S0,HIGH);
  digitalWrite(S1,LOW);
}
```

```
// Iniciar la comunicacion serie
Serial.begin(9600);
}
void loop() {
  // Definiendo la lectura de los fotodiodos con filtro rojo
  digitalWrite(S2,LOW);
  digitalWrite(S3,LOW);

  // Leyendo la frecuencia de salida del sensor
  frecuenciaRojo = pulseIn(salidaSensor, LOW);

  // Mostrando por serie el valor para el rojo (R = Red)
  Serial.print("R = ");
  Serial.print(frecuenciaRojo);
  delay(100);

  // Definiendo la lectura de los fotodiodos con filtro verde
  digitalWrite(S2,HIGH);
  digitalWrite(S3,HIGH);

  // Leyendo la frecuencia de salida del sensor
  frecuenciaVerde = pulseIn(salidaSensor, LOW);

  // Mostrando por serie el valor para el verde (G = Green)
  Serial.print(" G = ");
  Serial.print(frecuenciaVerde);
  delay(100);

  // Definiendo la lectura de los fotodiodos con filtro azul
  digitalWrite(S2,LOW);
  digitalWrite(S3,HIGH);

  // Leyendo la frecuencia de salida del sensor
  frecuenciaAzul = pulseIn(salidaSensor, LOW);

  // Mostrando por serie el valor para el azul (B = Blue)
  Serial.print(" B = ");
  Serial.println(frecuenciaAzul);
  delay(100);
}
```

Ejemplo 2 Detección de Colores

Detectar los colores rojo, verde y azul y mostrar en el monitor serial. En esta sección, insertaremos los valores de frecuencia seleccionados en el código previo, para que el sensor pueda distinguir entre diferentes colores.

```
const int s0 = 1;
const int s1 = 2;
const int s2 = 3;
const int s3 = 4;
const int out = 5;
int rojo = 0;
int verde = 0;
int azul = 0;

void setup(){
  Serial.begin(9600);
  pinMode(s0,OUTPUT);
  pinMode(s1,OUTPUT);
  pinMode(s2,OUTPUT);
  pinMode(s3,OUTPUT);
  pinMode(out,INPUT);
  digitalWrite(s0,HIGH);
  digitalWrite(s1,HIGH);
}

void loop(){
  color();
  Serial.print(" ");
  Serial.print(rojo, DEC);
  Serial.print(" ");
  Serial.print(verde, DEC);
  Serial.print(" ");
  Serial.print(azul, DEC);

  if (rojo < azul && verde > azul && rojo < 40)
  {
    Serial.println(" Rojo");
  }
  else if (azul < rojo && azul < verde && verde < rojo)
  {
    Serial.println(" Azul");
  }

  else if (rojo > verde && azul > verde )
  {
    Serial.println(" Verde");
  }
  else{
    Serial.println(" ");
  }
  delay(900);
}
```

```

    }

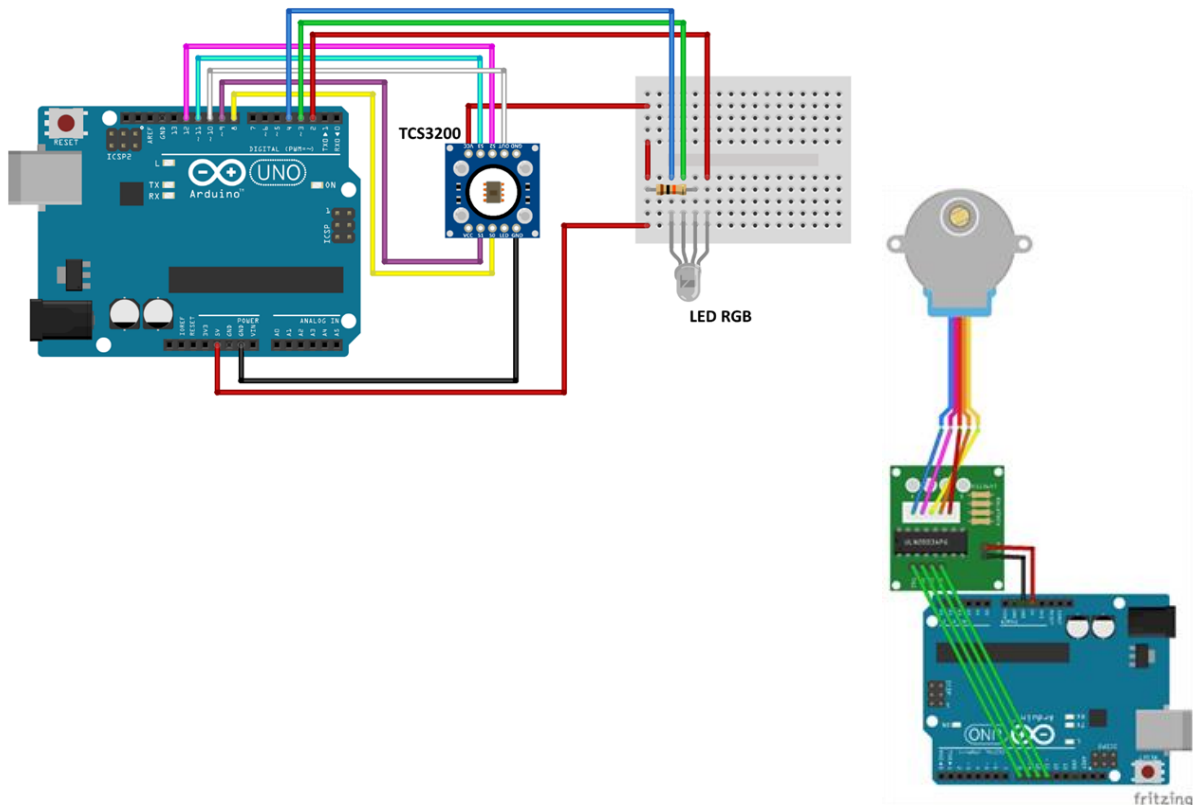
    void color()
    {
        digitalWrite(s2, LOW);
        digitalWrite(s3, LOW);
        rojo = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
        digitalWrite(s3, HIGH);
        azul = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
        digitalWrite(s2, HIGH);
        verde = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
    }

```

PRÁCTICA PARA EL ESTUDIANTE:

Utilizando el sensor de color TCS3200 si detecta el color rojo que gire un motor paso a paso en sentido horario, si detecta el color verde gire en sentido antihorario, visualizar cada color en un LED RGB.

NOTA: Para detectar los colores se deben hacer tres cubos cada uno de los colores mencionados



LABORATORIO N° 9

TECLADO MATRICIAL

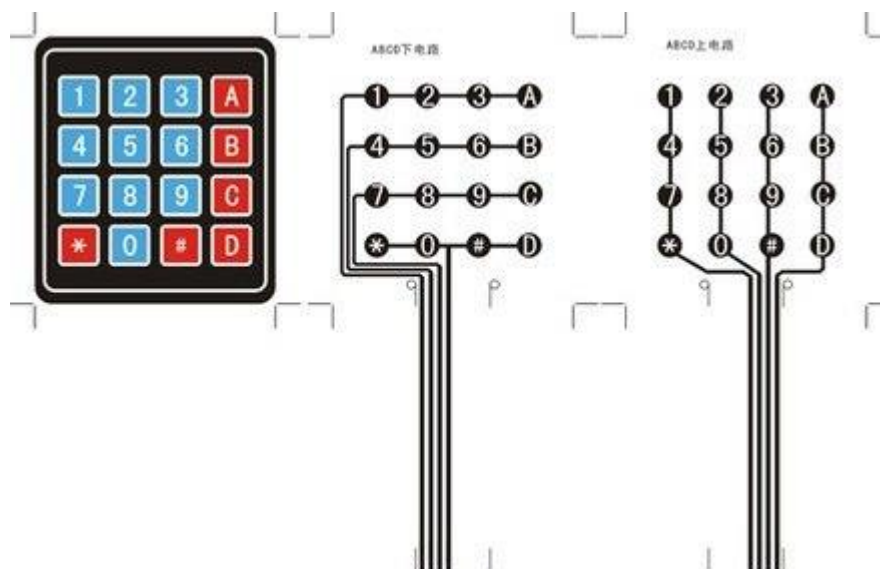
➤ **Objetivo.-**

Aprender que es un teclado matricial 4×4, como conectarlo a la tarjeta Arduino y por ultimo como realizar la programación para la visualización de los caracteres del teclado en el monitor serial del Software Arduino.

➤ **Marco teórico.-**

Teclado Matricial

Teclado matricial 4×4: Un teclado matricial no es más que una matriz de botones, los teclados comunes poseen los valores 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #, A, B, C, D, estos caracteres ocupan las 16 teclas de un teclado 4×4, para que estos teclados realicen la función de una matriz, cada casilla es asignada a una fila o a una columna, esto para que el software Arduino reconozca las teclas presionadas gracias a la librería Keypad.



Como se puede ver en la imagen superior de lado derecho se aprecian las filas y columnas así como la asignación de cada tecla.

El teclado matricial cuenta con 8 puntos de conexión, debido a la librería de Arduino esta conexión resulta muy fácil al no requerir resistencias u otros elementos aparte claro de cables o jumpers. Tal y como se ve en la siguiente imagen la conexión del primer pin del teclado se conecta a la entrada digital número 2 de Arduino, después de eso todo se conecta en una fila seguida

`#include<Keypad.h>` Incluir la librería

Ahora deberemos declarar las filas y columnas que posee nuestro teclado

```
const byte filas = 4; //Con esto definimos que utilizaremos las 4 filas de nuestro teclado
```

```
const byte columnas = 4; //Con esto definimos que utilizaremos las 4 columnas de nuestro teclado
```

```
byte pinesF[filas] = {9,8,7,6}; //Ahora definiremos los pines de Arduino a los que estarán conectadas las filas de nuestra matriz (para referirnos a estas conexiones en la programación usaremos "pinesF"
```

```
byte pinesC[columnas] = {5,4,3,2}; //Ahora definiremos los pines de Arduino a los que estarán conectadas las columnas de nuestra matriz (para referirnos a estas conexiones en la programación usaremos "pinesC"
```

```
char teclas[filas][columnas] = { //Ahora declaramos las teclas que conforman nuestra matriz, es muy importante ponerlas en el siguiente orden
```

```
{'1','2','3','A'},
```

```
{'4','5','6','B'},
```

```
{'7','8','9','C'},
```

```
{' ','0','#','D'}
```

```
};
```

```
Keypad teclado = Keypad(makeKeymap(teclas), pinesF, pinesC, filas, columnas); //Esta instrucción sirve para realizar el mapeo de nuestro teclado matricial
```

```
char tecla; // ahora deberemos definir la variable tecla, como su nombre lo indica es la tecla que se mostrara al presionarla en el teclado
```

```
void setup() { // abrimos el ciclo void setup
```

```
Serial.begin(9600); // el valor de 9600 es predefinido cuando se utiliza la comunicación serial
```

```
}
```

```
void loop() {
```

```
tecla = teclado.getKey(); // con esta instrucción le decimos a nuestro progrma que el caracter de tecla que se registre sera el caracter presionado de nuestro teclado
```

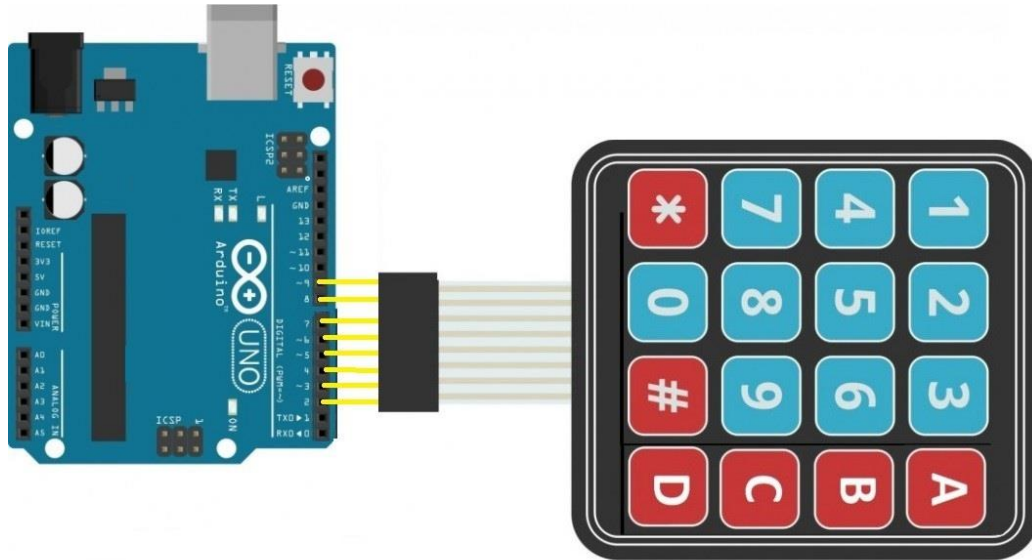
```
if (tecla != 0) // casi para terminar abrimos un ciclo if para preguntar si tecla es diferente que 0
```

```
Serial.print(tecla); // y ahora si por ultimo le pedimos al programa que imprima el valor de tecla en el monitor serial del software arduino
```

```
}
```

➤ **Circuito y Programación.-**

Tras la carga de la programación a Arduino al abrir el monitor serial de Arduino y presionar una tecla en el teclado matricial, el carácter se imprime correctamente en el monitor.



```
#include<Keypad.h>
const byte filas = 4;
const byte columnas = 4;
byte pinesF[filas] = {9,8,7,6};
byte pinesC[columnas] = {5,4,3,2};
char teclas[filas][columnas] = {

    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};

Keypad teclado = Keypad(makeKeymap(teclas), pinesF, pinesC, filas, columnas);
char tecla;

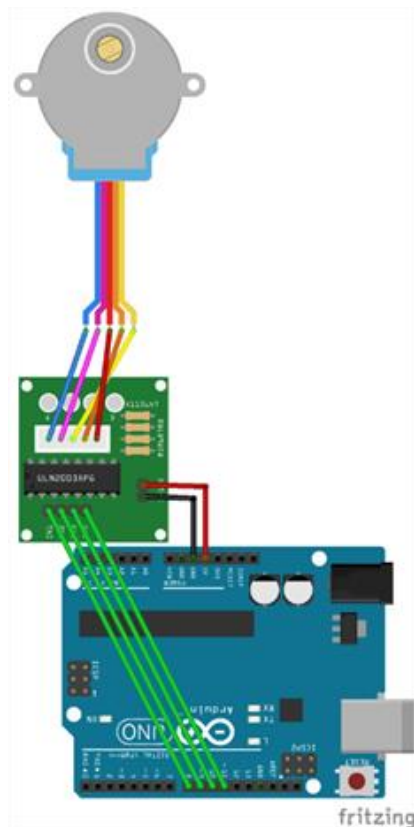
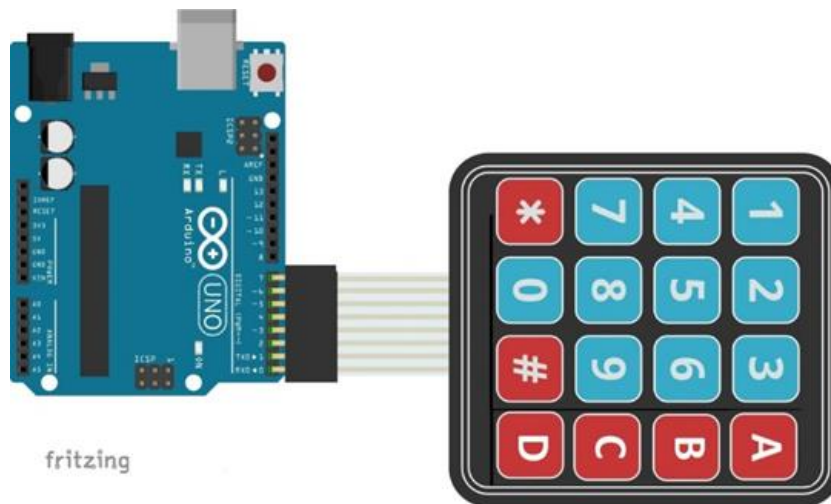
void setup() {
    Serial.begin(9600);
}

void loop() {
    tecla = teclado.getKey();
    if (tecla != 0)
        Serial.print(tecla);
}
```


PRÁCTICA PARA EL ESTUDIANTE:

Utilizar el teclado matricial para introducir los parámetros para el control de un motor paso a paso:

- A-Los grados que girara el motor paso a paso
- B-El tiempo de funcionamiento
- C-El sentido de giro



LABORATORIO N° 10

BLUETOOTH

➤ **Objetivo.-**

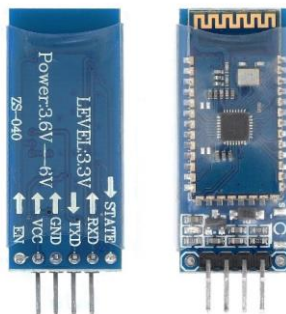
Diseñar e implementar un circuito digital en base al microcontrolador Arduino que permita la comunicación Bluetooth, veremos cómo configurarlo y cómo enviar y recibir datos desde un dispositivo Android. Se controlarán Leds y algunos actuadores desde una aplicación creada para nuestro smatphone Android, usando MIT App Inventor.

➤ **Marco teórico.-**

Módulo Bluetooth

El Bluetooth es un estándar de comunicación inalámbrica que permite la transmisión de datos a través de radiofrecuencia en la banda de 2,4 GHz. Existen muchos módulos Bluetooth para usarlos en nuestros proyectos de electrónica, pero los más utilizados son los módulos de JY-MCU, ya que son muy económicos y fáciles de encontrar en el mercado. Son módulos pequeños y con un consumo muy bajo que nos permitirán agregar funcionalidades Bluetooth a nuestro Arduino. Estos módulos contienen el chip con una placa de desarrollo con los pins necesarios para la comunicación serie.

Existen dos modelos de módulos Bluetooth: el HC-05 que puede ser maestro/esclavo (master/slave), y el HC-06 que solo puede actuar como esclavo (slave). Los módulos Bluetooth pueden configurarse como Maestro o Esclavo. Esto quiere decir que un módulo configurado como Maestro es el que inicia el emparejamiento o conexión, mientras que un módulo configurado como Esclavo espera a que otros se conecten a él.



Físicamente, los dos módulos son muy parecidos, solo varían algunas conexiones. Los pins que encontraremos son los siguientes:

- **Vcc:** Alimentación del módulo entre 3,6V y 6V.
- **GND:** La masa del módulo.

- **TXD:** Este es el pin de transmisión de datos. A través de este pin, el módulo transmite hacia el Arduino los datos que le llegan desde un dispositivo conectado, como un móvil. Se conecta al pin RX del Arduino.
- **RXD:** Este es el pin de recepción de datos. A través de este pin, el módulo recibe datos desde el Arduino para enviarlos hacia un dispositivo conectado, como un móvil. Se conecta al pin TX del Arduino.
- **KEY:** Poner a nivel alto para entrar en modo configuración del módulo (solo el modelo HC-05)
- **STATE:** Para conectar un led de salida para visualizar cuando se comuniquen datos.

Cuando realizamos las conexiones, el módulo se encenderá y un LED empezará a parpadear. Esto significa que el módulo no está emparejado. El LED dejará de parpadear cuando se establezca una conexión entre el HC-06 y otro dispositivo.

En cuanto a la programación, no es necesario utilizar ninguna librería. Sólo basta con poner en el `setup()` la inicialización del puerto serial a una determinada velocidad de baudios:

```
Serial.begin(9600);
```

Se suele utilizar una velocidad de 9600 baudios porque esa es la velocidad a la que se comunica el HC-06 por defecto. En cuanto al `setup()` declaramos los pines como entradas o salidas con la función `pinMode()`, dependiendo de lo que tengamos conectado al Arduino (por ejemplo, un LED, un pulsador, un servo...).

En este laboratorio se realizan varios ejemplos para demostrar las posibilidades de controlar a distancia diversos dispositivos en nuestros proyectos con Arduino. Todo se hará a través de aplicaciones desarrolladas para Android utilizando la plataforma MIT App Inventor.

Importante: Antes de subir un código al Arduino, se debe desconectar el módulo Bluetooth de los pines TX y RX de la tarjeta, ya que éstos son usados para la comunicación con el PC a la hora de subirle algún código. Luego de cargar el código, se vuelve a conectar el módulo al Arduino.

A la hora de conectar el HC-06 con nuestro teléfono Android, pedirá una contraseña para establecer la conexión. Por defecto es: 1234

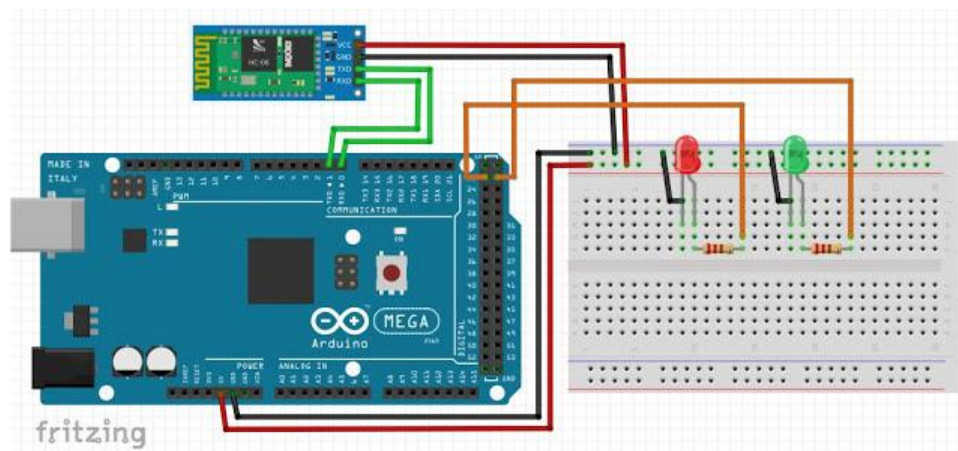
➤ **Circuito y Programación.-Ejemplo 1**

Controlar dos LEDs, los cuales se pueden encender y apagar desde el smartphone. A través del Bluetooth llegan datos de tipo "char" por parte del móvil. El Arduino decide qué acción ejecutar de acuerdo al dato que recibe, por ejemplo, si recibe la letra 'A' entonces

enciende un LED y si recibe la letra 'B' lo apaga. (En el circuito se usa un arduino mega pero también se puede utilizar el arduino uno)

Pasos:

- Cargar el código a nuestro arduino con modulo bluetooth desconectado
- Descargar e instalar la aplicación en nuestro Smartphone (CD)
- Activar el Bluetooth de nuestro Smartphone y entrar a la aplicación
- Conectar el modulo Bluetooth y encender nuestro arduino con el circuito también conectado y realizar la alimentación.
- Desde la aplicación emparejar el bluetooth del arduino con el del Smartphone
- Encender y apagar los led desde nuestro smartphone



//Variables asociadas a los dos LEDs que se van a controlar

```
int led_1 = 22;
```

```
int led_2 = 23;
```

```
char valor; //Variable para indicar que llega una orden
```

```
void setup() {
  pinMode(led_1, OUTPUT);
  pinMode(led_2, OUTPUT);
  Serial.begin(9600);
}
```

```
void loop() {
  if (Serial.available()) //Si el puerto serie (Bluetooth) está disponible
  {
    valor = Serial.read(); //Lee el dato entrante via Bluetooth

    if (valor == 'A') //Si el dato que llega es una A
    {
```

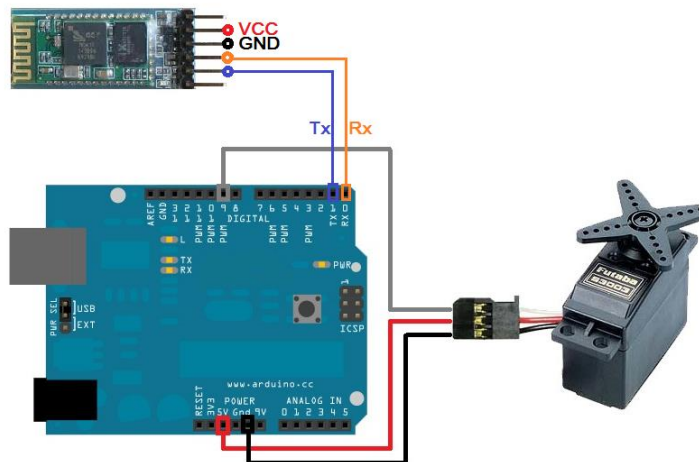
```

    digitalWrite(led_1, HIGH); //Enciende el LED 1
  }
  if (valor == 'B') //Si el dato que llega es una B
  {
    digitalWrite(led_1, LOW); //Apaga el LED 1
  }
  if (valor == 'C') //Si el dato que llega es una C
  {
    digitalWrite(led_2, HIGH); //Enciende el LED 2
  }
  if (valor == 'D') //Si el dato que llega es una D
  {
    digitalWrite(led_2, LOW); //Apaga el LED 2
  }
}
}
}

```

➤ **Circuito y Programación.-Ejemplo 2**

Controlar el PWM en una salida de nuestro arduino, para variar el ángulo de un servomotor. En este caso en particular, utilizamos deslizadores para enviar datos variables los cuales modificarán el ángulo del servo además también podemos variar el ángulo del servomotor con 5 pulsadores (0°, 45°, 90°, 135°, 180°) desde el smartphone.



```

#include <SoftwareSerial.h> // TX RX software library for bluetooth
#include <Servo.h> // servo library
Servo myservo; // servo name

```

```

int bluetoothTx = 10; // bluetooth tx to 10 pin
int bluetoothRx = 11; // bluetooth rx to 11 pin
SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);

```

```

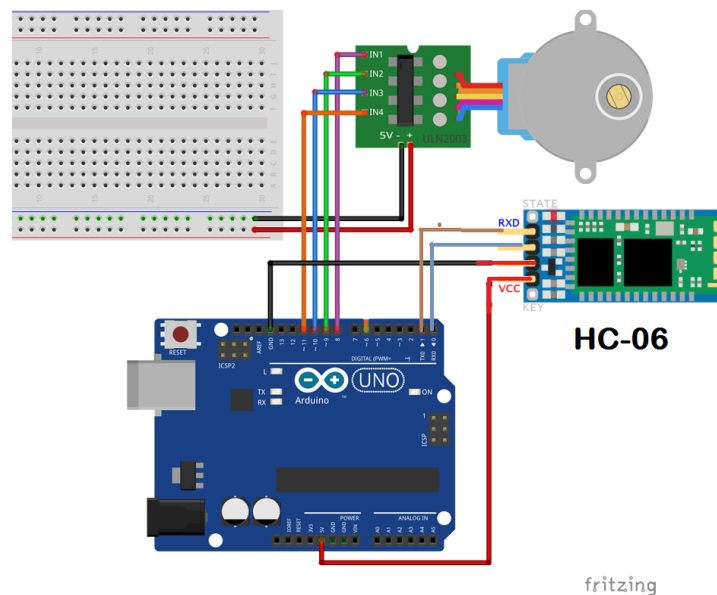
void setup()
{
  myservo.attach(9); // attach servo signal wire to pin 9
  //Setup usb serial connection to computer
  Serial.begin(9600);
  //Setup Bluetooth serial connection to android
  bluetooth.begin(9600);
}

void loop()
{
  //Read from bluetooth and write to usb serial
  if(bluetooth.available() > 0) // receive number from bluetooth
  {
    int servopos = bluetooth.read(); // save the received number to servopos
    Serial.println(servopos); // serial print servopos current number received from
    bluetooth
    myservo.write(servopos); // rotate the servo the angle received from the android app
  }
}

```

PRÁCTICA PARA EL ESTUDIANTE:

Controlar un motor paso a paso desde nuestro Smartphone a través del Módulo Bluetooth, desde el Smartphone podemos controlar el sentido de giro del motor (sentido horario y sentido antihorario con botones) y la velocidad de giro de nuestro motor (con un deslizador).



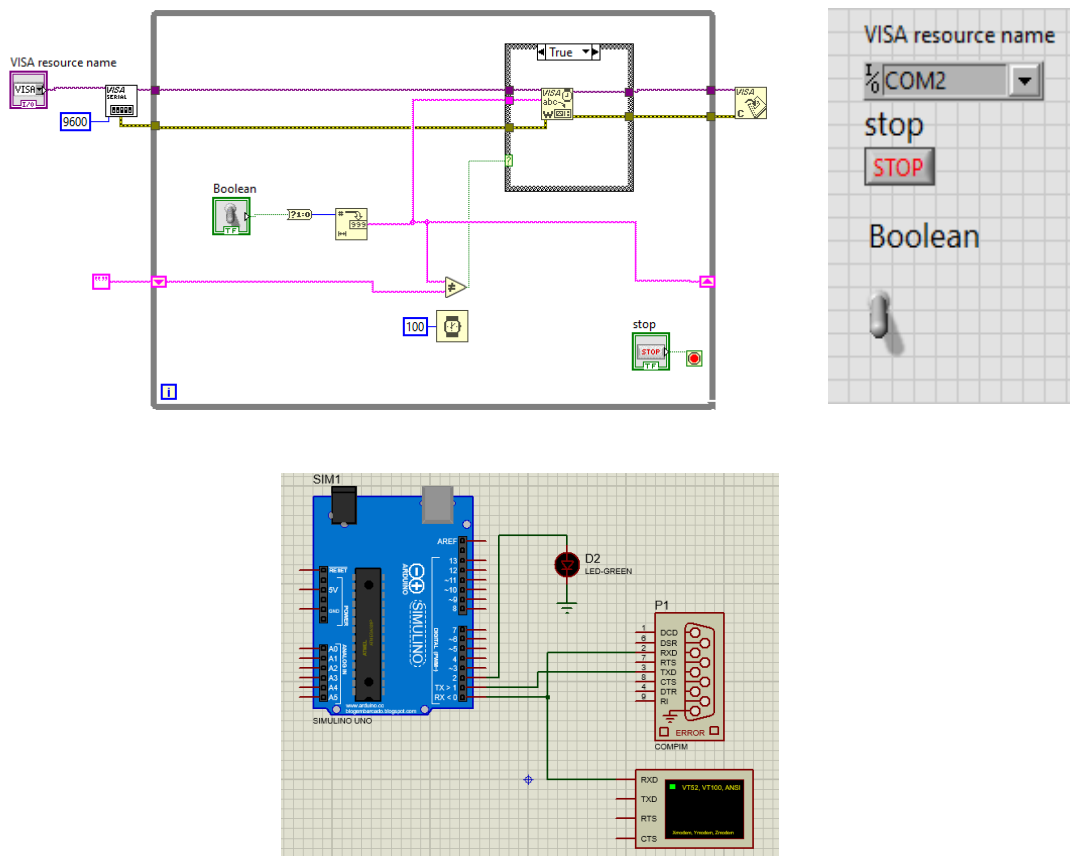
LABORATORIO N° 11

ARDUINO LABVIEW SALIDA DIGITAL

➤ Objetivo

Diseñar e implementar un circuito digital en base al microcontrolador Arduino y labview que permita encender ledS desde labview.

Ejemplo 1. Enciende un led externo desde Labview



```
int pinled=12;
```

```
char var;
```

```
void setup() {
  serial.begin(9600);
  pinmode(pinled,output);
}
```

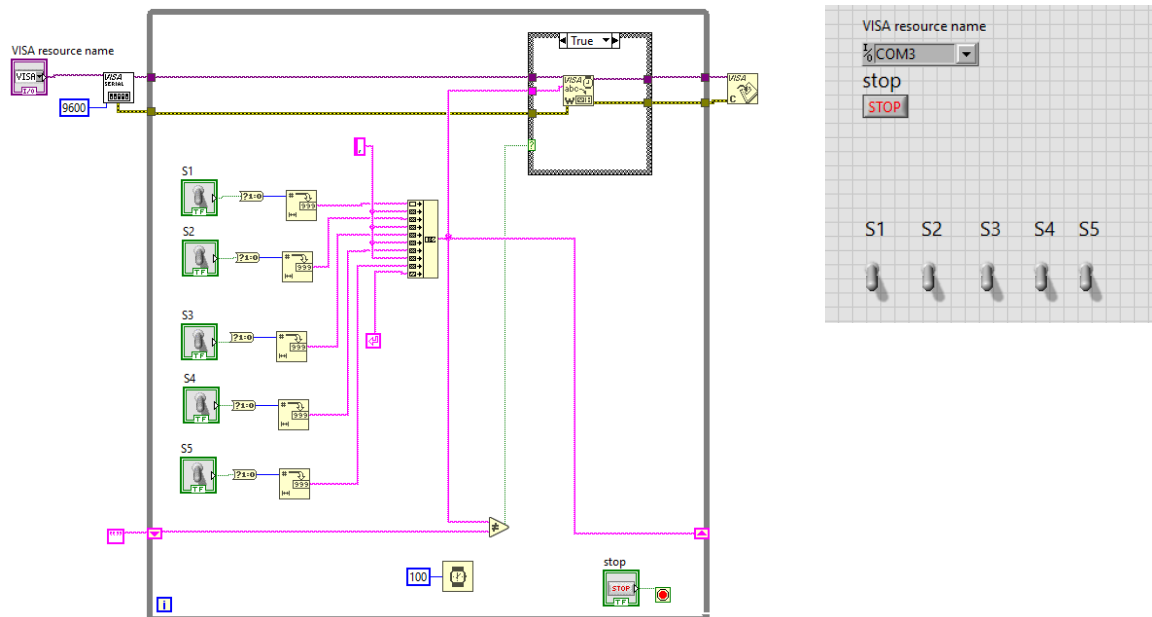
```
void loop()
{
}
```

```

void serialEvent()
{
    if (serial.available())
    {
        var=serial.read();
        if(var=='1')
        {digitalwrite(pinled,high);}
        else {digitalwrite(pinled,low);}
    }
}

```

Ejemplo 2. Enciende cinco leds externo desde labview



```

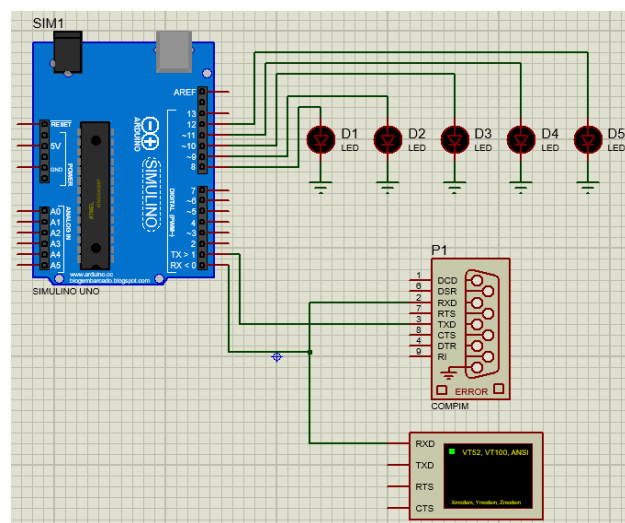
int pinled1=8;
int pinled2=9;
int pinled3=10;
int pinled4=11;
int pinled5=12;

```

```

int var1;
int var2;
int var3;
int var4;
int var5;

```




```
void setup()
{
  serial.begin(9600);
  pinMode(pinled1,output);
  pinMode(pinled2,output);
  pinMode(pinled3,output);
  pinMode(pinled4,output);
  pinMode(pinled5,output);
}

void loop()
{
}

void serialevent()
{
  if(serial.available())
  {
    var1=serial.parseInt();
    var2=serial.parseInt();
    var3=serial.parseInt();
    var4=serial.parseInt();
    var5=serial.parseInt();
    if(serial.read()==char(13))
    {
      digitalWrite(pinled1,var1);
      digitalWrite(pinled2,var2);
      digitalWrite(pinled3,var3);
      digitalWrite(pinled4,var4);
      digitalWrite(pinled5,var5);
    }
  }
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Realizar la rotación de ocho leds externos a la derecha si los swichs de labview SW1 esta en alto y SW2 esta en bajo. Si SW1 esta en bajo y SW2 está en alto rotar a la izquierda. En caso de ambos SW1 y SW2 estén en alto o bajo no realizar ninguna rotación.

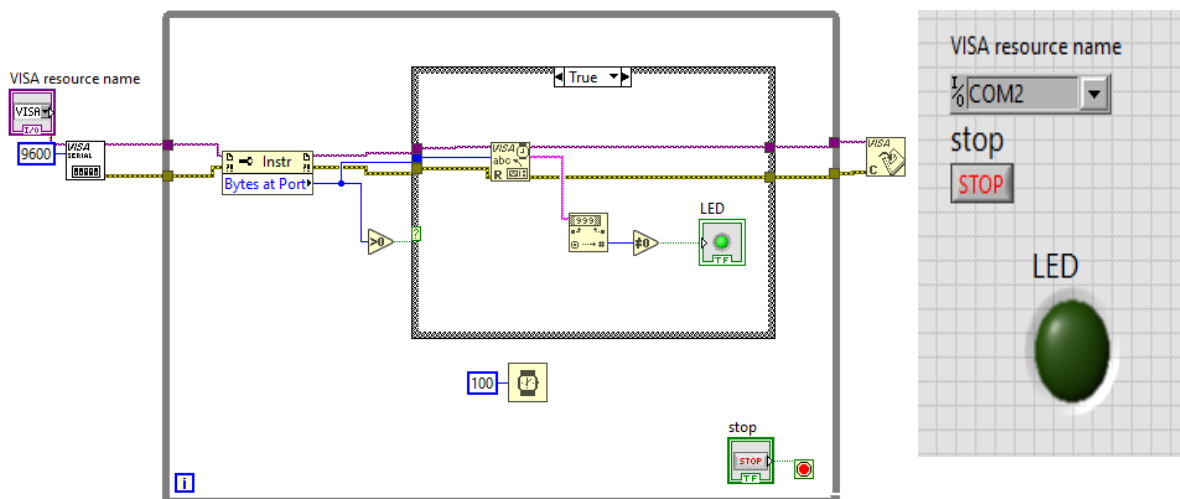
LABORATORIO Nº 12

ARDUINO LABVIEW ENTRADA DIGITAL

➤ Objetivo

Diseñar e implementar un circuito digital en base al microcontrolador Arduino y labview que permita encender leds de forma externa y sea mostrado en labview.

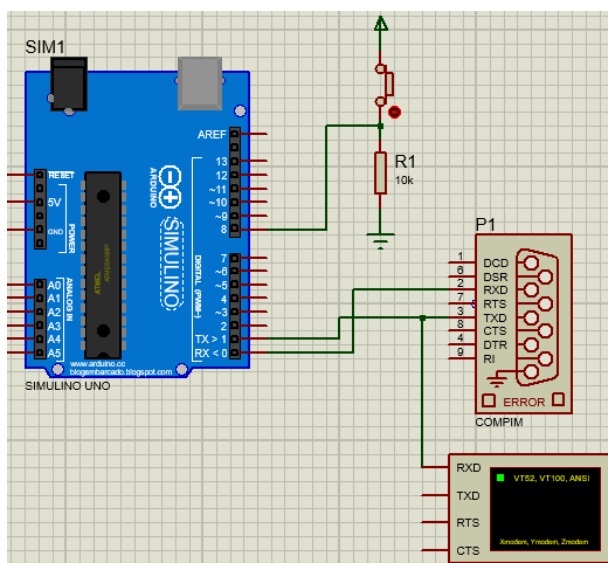
Ejemplo1. Enciende un led en la pantalla de labview con un swich externo



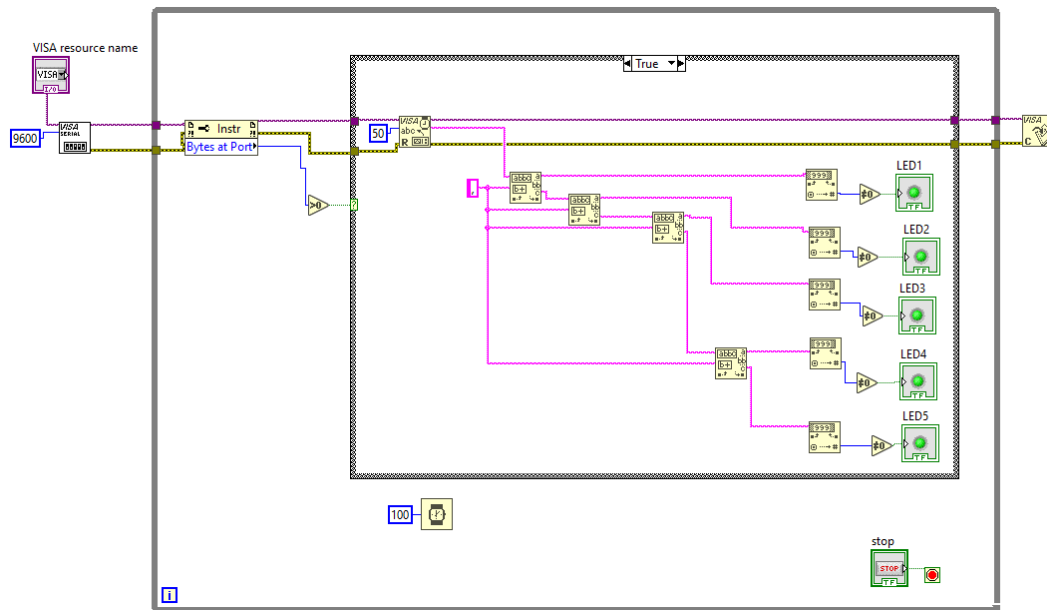
```
int pinbutton=8;
int var;
unsigned long msegi=0;

void setup()
{
  serial.begin(9600);
  pinmode(pinbutton, input);
}

void loop()
{
  var=digitalread(pinbutton);
  unsigned long msegf=millis();
  if (msegf-msegi>=100)
  {
    msegi=msegf;
    serial.println(var);
  }
}
```



Ejemplo 2. Enciende cinco leds en la pantalla de labview con swchs externos

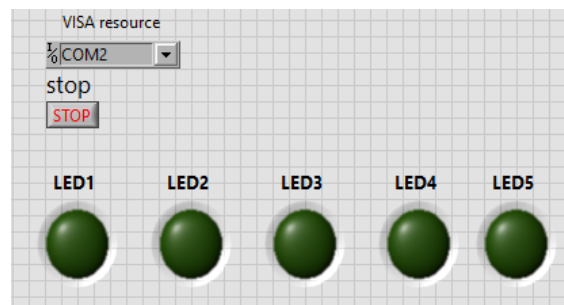


```
int pinbutton1=8;  
int pinbutton2=9;  
int pinbutton3=10;  
int pinbutton4=11;  
int pinbutton5=12;
```

```
int var1;  
int var2;  
int var3;  
int var4;  
int var5;
```

```
unsigned long mseg1=0;
```

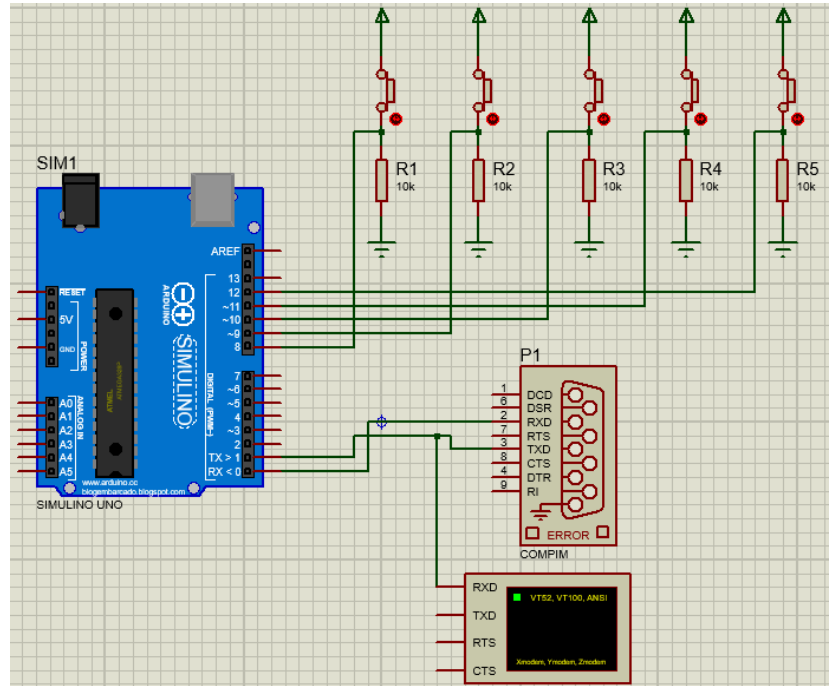
```
void setup()
{
  serial.begin(9600);
  pinMode (pinbutton1,input);
  pinMode (pinbutton2,input);
  pinMode (pinbutton3,input);
  pinMode (pinbutton4,input);
  pinMode (pinbutton5,input);
}
```



```

void loop()
{
    var1=digitalread(pinbutton1);
    var2=digitalread(pinbutton2);
    var3=digitalread(pinbutton3);
    var4=digitalread(pinbutton4);
    var5=digitalread(pinbutton5);
    unsigned long msegf=millis();
    if(msegf-msegi>=100)
    {
        msegi=msegf;
        serial.print(var1);
        serial.print(",");
        serial.print(var2);
        serial.print(",");
        serial.print(var3);
        serial.print(",");
        serial.print(var4);
        serial.print(",");
        serial.print(var5);
        serial.println(var5);
    }
}

```



PRÁCTICA PARA EL ESTUDIANTE:

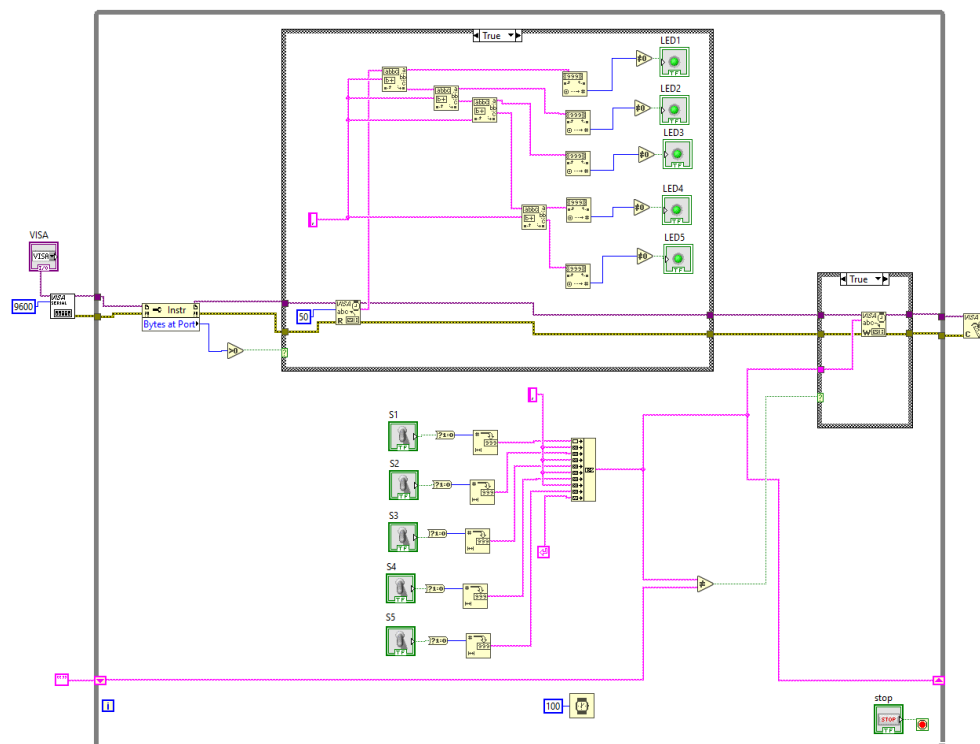
Realizar la rotación de ocho leds internos (labview) a la derecha si los swichs externos SW1 esta en alto y SW2 esta en bajo. Si SW1 esta en bajo y SW2 está en alto rotar a la izquierda. En caso de ambos SW1 y SW2 estén en alto o bajo no realizar ninguna rotación.

LABORATORIO Nº 13
ARDUINO LABVIEW ENTRADA Y SALIDA DIGITAL

➤ Objetivo

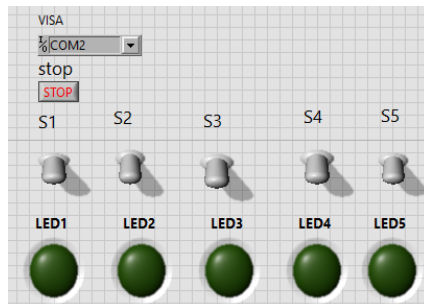
Diseñar e implementar un circuito digital en base al microcontrolador Arduino y labview que permita encender y apagar leds desde labview.

EJEMPLO 1. Enciende cinco leds en la pantalla de labview con swchs internos



```
INT PINLED1=3;
INT PINLED2=4;
INT PINLED3=5;
INT PINLED4=6;
INT PINLED5=7;
```

```
INT PINBUTTON1=8;  
INT PINBUTTON2=9;  
INT PINBUTTON3=10;  
INT PINBUTTON4=11;  
INT PINBUTTON5=12;
```



```

INT VAR1;
INT VAR2;
INT VAR3;
INT VAR4;
INT VAR5;
INT VAR6;
INT VAR7;
INT VAR8;
INT VAR9;
INT VAR10;

```

```

UNSIGNED LONG MSEG1=0;

```

```

VOID SETUP()

```

```

{
    SERIAL.BEGIN(9600);
    PINMODE (PINLED1,OUTPUT);
    PINMODE (PINLED2,OUTPUT);
    PINMODE (PINLED3,OUTPUT);
    PINMODE (PINLED4,OUTPUT);
    PINMODE (PINLED5,OUTPUT);

```

```

    PINMODE (PINBUTTON1,INPUT);
    PINMODE (PINBUTTON2,INPUT);
    PINMODE (PINBUTTON3,INPUT);
    PINMODE (PINBUTTON4,INPUT);
    PINMODE (PINBUTTON5,INPUT);
}

```

```

VOID LOOP()

```

```

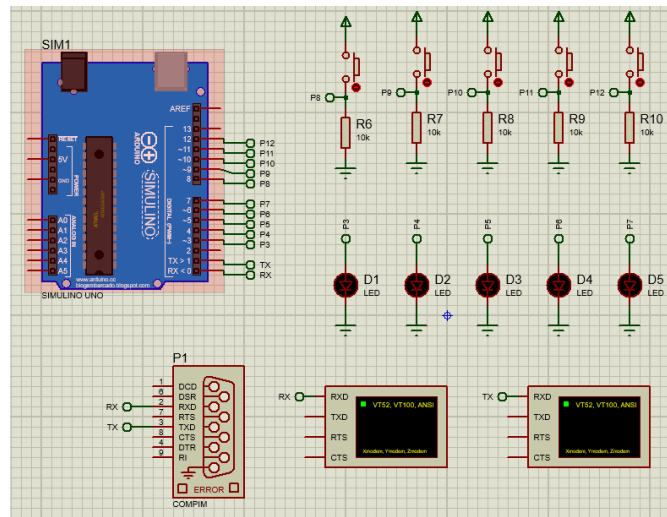
{
    VAR1=DIGITALREAD(PINBUTTON1);
    VAR2=DIGITALREAD(PINBUTTON2);
    VAR3=DIGITALREAD(PINBUTTON3);
    VAR4=DIGITALREAD(PINBUTTON4);
    VAR5=DIGITALREAD(PINBUTTON5);
    UNSIGNED LONG MSEG1=MILLIS();
    IF(MSEG1-MSEG0>=100)
    {

```

```

        MSEG1=MSEG0;
        SERIAL.PRINT(VAR1);
        SERIAL.PRINT(",");
        SERIAL.PRINT(VAR2);
        SERIAL.PRINT(",");
        SERIAL.PRINT(VAR3);
        SERIAL.PRINT(",");
        SERIAL.PRINT(VAR4);
    }
}

```



```
SERIAL.PRINT(",");
SERIAL.PRINTLN(VAR5);
}
}
VOID SERIALEVENT()
{
IF(SERIAL.AVAILABLE())
{
    VAR6=SERIAL.PARSEINT();
    VAR7=SERIAL.PARSEINT();
    VAR8=SERIAL.PARSEINT();
    VAR9=SERIAL.PARSEINT();
    VAR10=SERIAL.PARSEINT();
    IF(SERIAL.READ()==CHAR(13))
    {
        DIGITALWRITE(PINLED1,VAR6);
        DIGITALWRITE(PINLED2,VAR7);
        DIGITALWRITE(PINLED3,VAR8);
        DIGITALWRITE(PINLED4,VAR9);
        DIGITALWRITE(PINLED5,VAR10);
    }
}
}
```

PRÁCTICA PARA EL ESTUDIANTE:

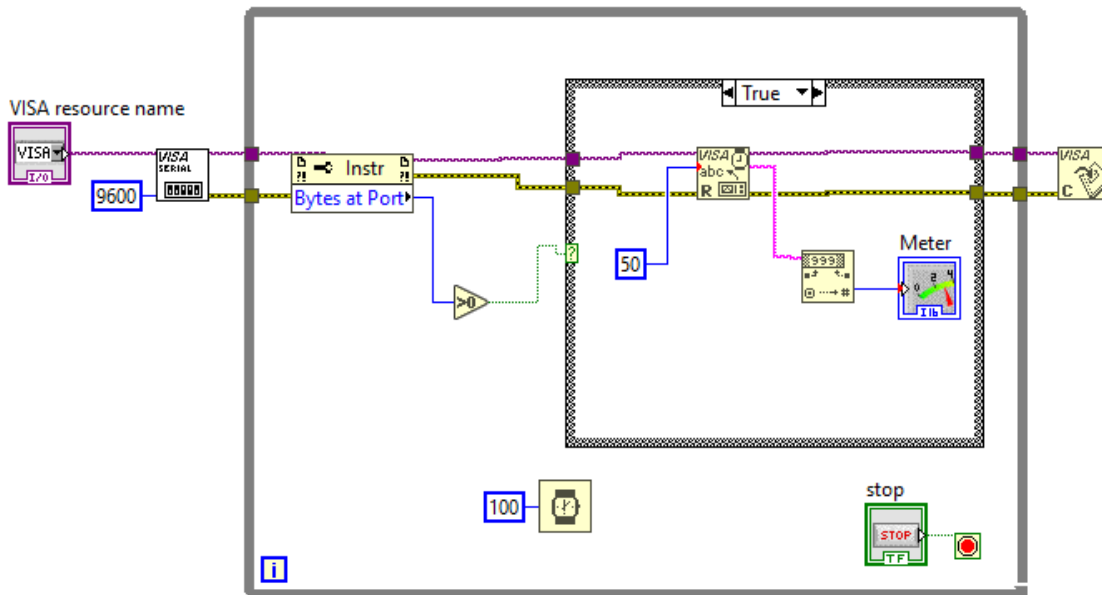
Realizar el encendido y pagado de un motor, DC, 6 leds y un motor AC.

LABORATORIO N° 14

ARDUINO LABVIEW ENTRADA ANALOGICA

➤ Objetivo

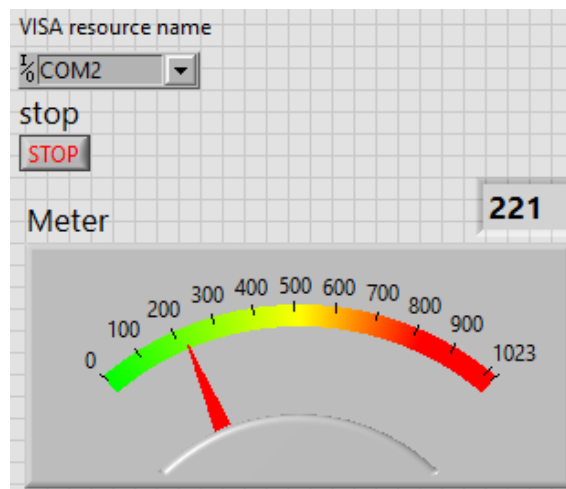
Diseñar e implementar un circuito digital en base al microcontrolador Arduino y labview que permita visualizar el estado de 6 potenciómetros desde labview.

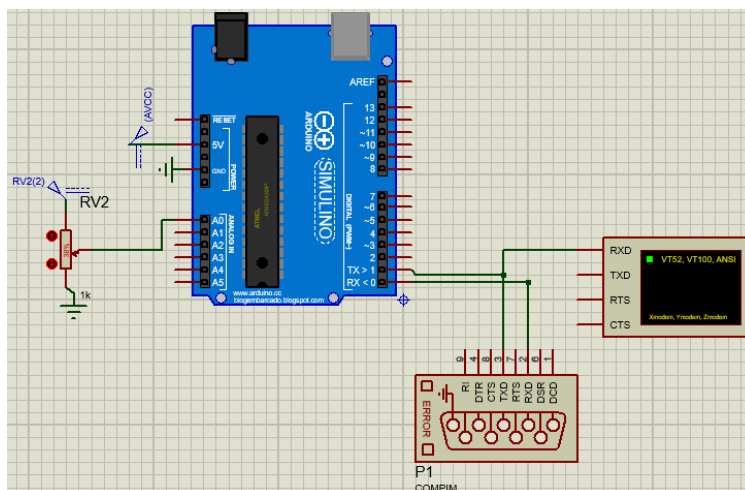


```
int vara0;

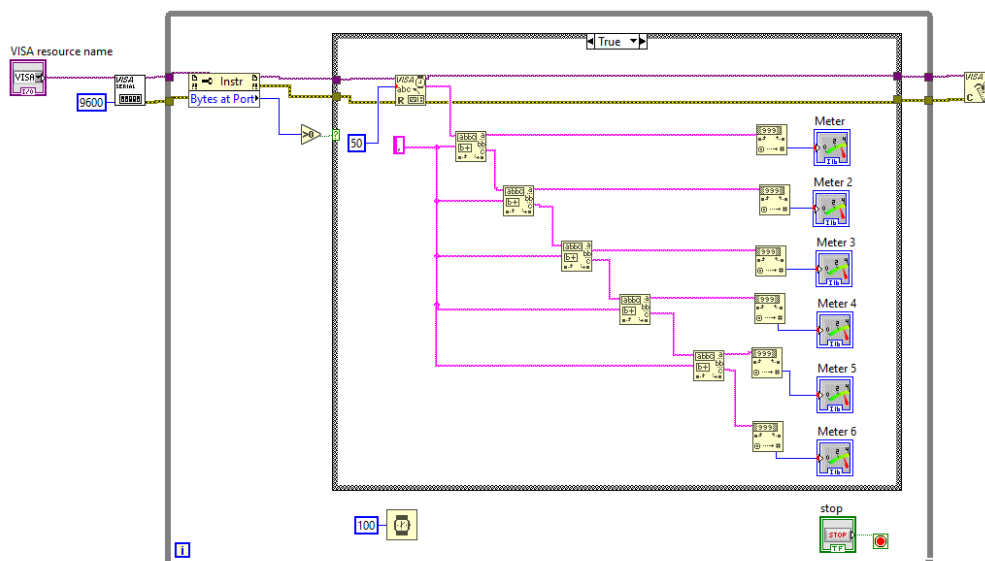
unsigned long msegf=0;
void setup()
{
  serial.begin(9600);
  pinmode(a0,input);
}

void loop()
{
  vara0=analogread(a0);
  unsigned long msegf=millis();
  if(msegf-msegf>=100)
  {
    msegf=msegf;
    serial.println(vara0);
  }
}
```





EJEMPLO 2. VISUALIZA EL ESTADO DE CINCO POTENCIOMETROS POTENCIOMETRO



```
int vara0;
int vara1;
int vara2;
int vara3;
int vara4;
int vara5;
```

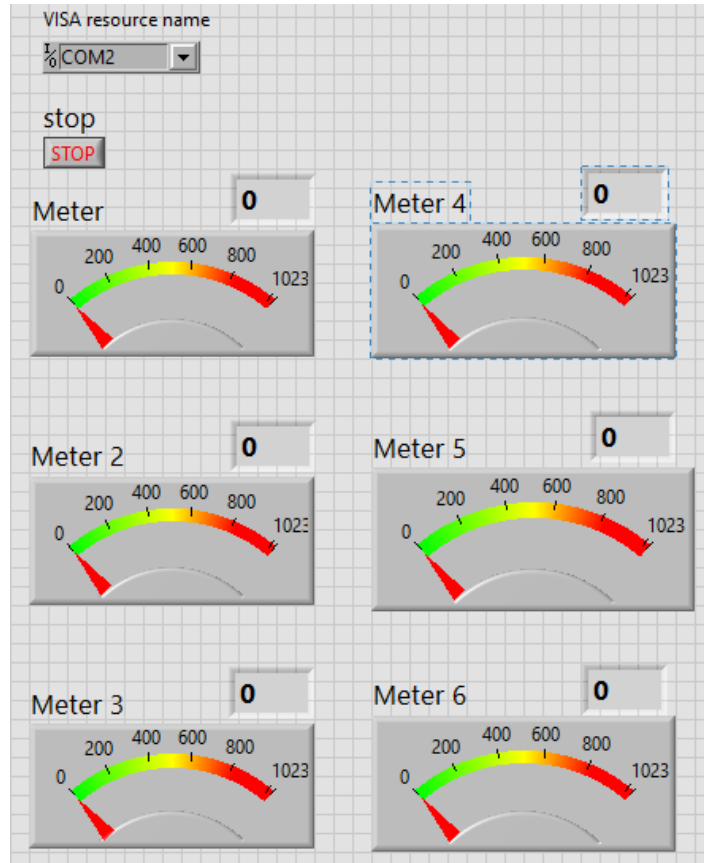
```
unsigned long mseg1=0;
void setup()
{
  serial.begin(9600);
  pinmode(a0,input);
  pinmode(a1,input);
```

```

pinmode(a2,input);
pinmode(a3,input);
pinmode(a4,input);
pinmode(a5,input);
}

void loop()
{
  vara0=analogread(a0);
  vara1=analogread(a1);
  vara2=analogread(a2);
  vara3=analogread(a3);
  vara4=analogread(a4);
  vara5=analogread(a5);
  unsigned long msegf=millis();
  if(msegf-msegi>=100)
  {
    msegi=msegf;
    serial.print(vara0);
    serial.print(",");
    serial.print(vara1);
    serial.print(",");
    serial.print(vara2);
    serial.print(",");
    serial.print(vara3);
    serial.print(",");
    serial.print(vara4);
    serial.print(",");
    serial.println(vara5);
  }
}

```



PRÁCTICA PARA EL ESTUDIANTE:

Realizar la medición del estado de 8 potenciómetros.

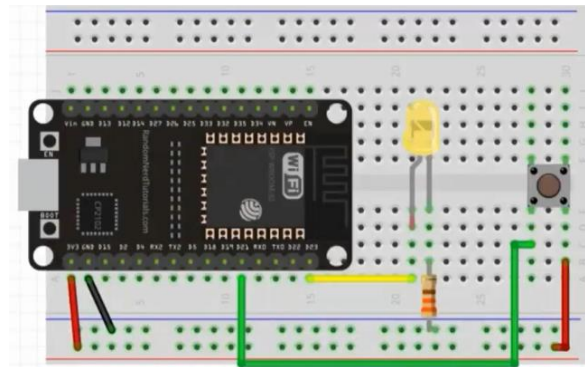
LABORATORIO N° 15

ESP32 ENTRADA Y SALIDA DIGITAL

➤ **Objetivo**

Diseñar e implementar un circuito digital en base al microcontrolador ESP32 encender leds permita con pulsador

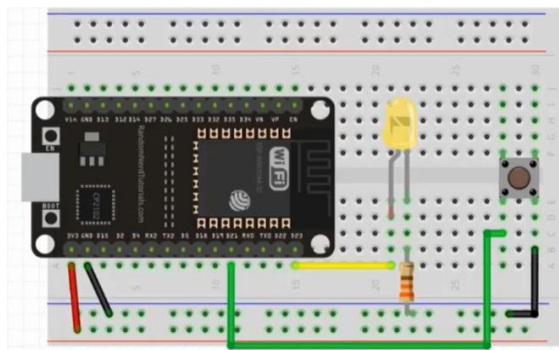
Ejemplo 1: Encender apagar un led con la resistencia integrada, emplear la función INPUT_PULLDOWN



```
int led = 23; //constante
int pulsador = 21; //constante
int lectura = 0; //variable
void setup() {
  pinMode(pulsador, INPUT_PULLDOWN);
  pinMode(led, OUTPUT);
}
```

```
void loop() {
  lectura = digitalRead(pulsador); //0 1
  if(lectura == 1){
    digitalWrite(led,1);
  }
  else{
    digitalWrite(led,0);
  }
}
```

Ejemplo 2: Encender apagar un led con la resistencia integrada, emplear la función INPUT_PULLUP



```
int led = 23; //constante
int pulsador = 21; //constante
int lectura = 0; //variable
void setup() {
  pinMode(pulsador, INPUT_PULLUP);
  pinMode(led, OUTPUT);
}

void loop() {
  lectura = digitalRead(pulsador); //0 1
  if(lectura == 0){
    digitalWrite(led, 1);
  }
  else{
    digitalWrite(led, 0);
  }
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Realizar la rotación de ocho leds externos a la derecha si los swichs de labview SW1 esta en alto y SW2 esta en bajo. Si SW1 esta en bajo y SW2 está en alto rotar a la izquierda. En caso de ambos SW1 y SW2 estén en alto o bajo no realizar ninguna rotación.

LABORATORIO N° 16

ESP32 SALIDA ANALOGICA

➤ **Objetivo**

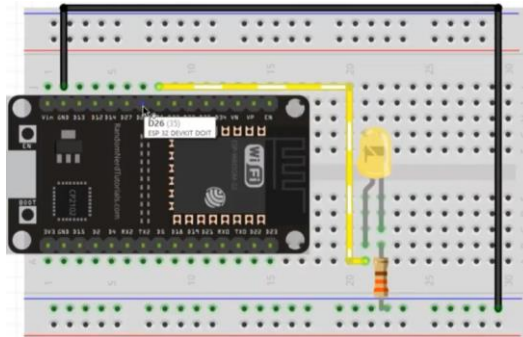
Diseñar e implementar un circuito digital en base al microcontrolador ESP32 que permita generar datos analogicos con la funcion DAC y PWM.

Una salida analogica es aquella que nos permite generar una señal continua, la cual tiene mas de 2 valores y que se encuentra en un rango determinando. En el caso del ESP32 0-255

La placa ESP32 permite genrar una salida analógica de dos formas diferentes:

- DAC (Conversor digital analógico)
- PWM (Modulación por ancho de pulso)

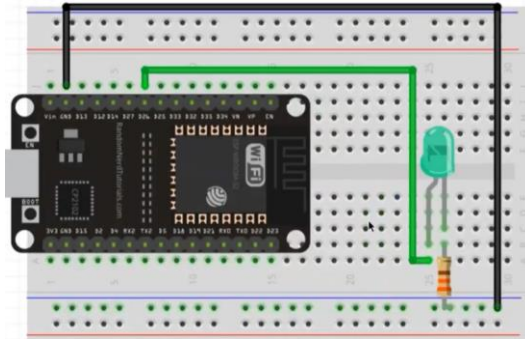
Ejemplo 1. DAC Encender un led al 100% 50% y 0% a razón de un 1 segundo empleando un DAC.



```
int amarillo = 25;
void setup() {
}

void loop() {
  dacWrite(amarillo,255);
  delay(1000);
  dacWrite(amarillo,128);
  delay(1000);
  dacWrite(amarillo,0);
  delay(1000);
}
```

EJEMPLO 2. DAC Que encienda el led que se encuentra en el pin 26 cada 10 milisegundos en el rango de 0 a 255 la intensidad de luz

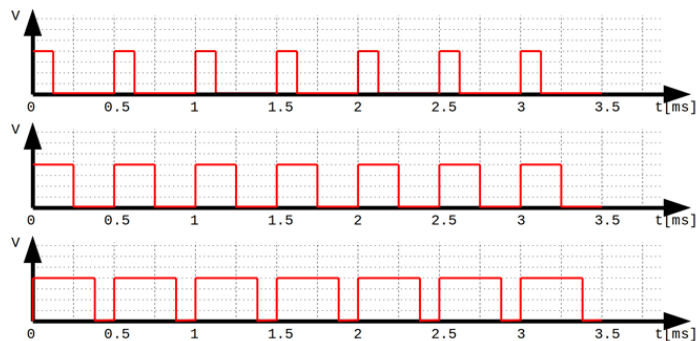


```
int verde = 26;
void setup() {
}

void loop() {
  for(int valor=0;valor<=255;valor++){
    dacWrite(verde,valor);
    delay(10);
  }
  for(int valor=254;valor>0;valor--){
    dacWrite(verde,valor);
    delay(10);
  }
}
```

SALIDA POR PWM

Su comportamiento de la salida PWM se genera a nivel de pulsos tomando como base la grafica de la señal digital en el que el periodo y la frecuencia son constantes y lo que varia es el tiempo que está en estado al alto (*HIGH*) y bajo (*LOW*).



Todos los pines que pueden actuar como salidas pueden ser utilizados como pines PWM (los pines GPIO 34 a 39 de sólo entrada no pueden generar PWM).

Para programa una señal PWM se necesita definir cuatro parámetros

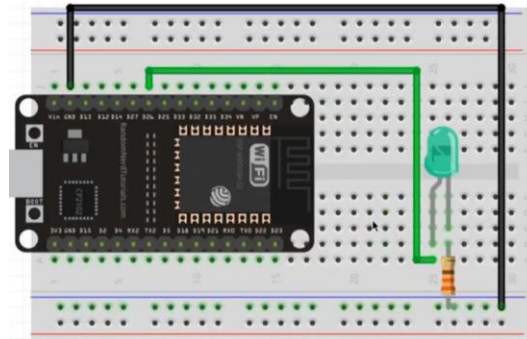
Frecuencia: Es la cantidad de ciclos por segundo

PWM: Canal de trabajo (0-15)

Ciclo de trabajo: Es la relación de tiempo de E/A

GPIO: Pin de trabajo por donde se emite la señal

EJEMPLO 3. PWM. Realizar un programa que haga variar la intensidad de brillo de un led utilizan PWM, con un retardo de 10 mseg.



```
int verde = 26;
int frecuencia = 5000; //kHz ->1-40000 kHz
int canal = 0; //0-15
int resolucion = 8; //0-255 -> 1 - 16 bits

void setup() {
  ledcSetup(canal,frecuencia,resolucion); //configura el modo de trabajo del PWM
  ledcAttachPin(verde,canal); //Agregar el canal al GPIO
}

void loop() {
  for(int valor=0;valor<=255;valor++){
    ledcWrite(canal,valor);
    delay(10);
  }
  for(int valor=254;valor>0;valor--){
    ledcWrite(canal,valor);
    delay(10);
  }
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Encender y apagar 2 leds de manera escalar, utilizar `for(int valor=0;valor<=255;valor++){`

Realizar un programa que encienda dos leds empleando **PMW**

LABORATORIO N° 17

SENSOR DE HUMEDAD DE SUELO

➤ **Objetivo**

Diseñar e implementar un circuito digital en base al microcontrolador ESP32 que permita leer datos de humedad de suelo.

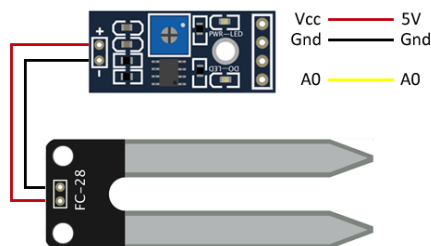
SENSOR DE HUMEDAD DE SUELO YL69/FC28

El sensor de humedad de suelo YL69 al igual que el FC28, nos permite medir la conductividad del suelo a través de electrodos.

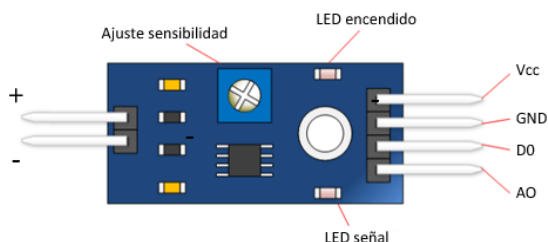
Características:

- Voltaje de entrada: 3.3 - 5 VCD
- Voltaje de salida: 0 ~ 4.2 V
- Corriente: 35 mA

Para la conexión alimentamos el módulo conectando GND y 5V a los pines correspondientes del Arduino o ESP32.

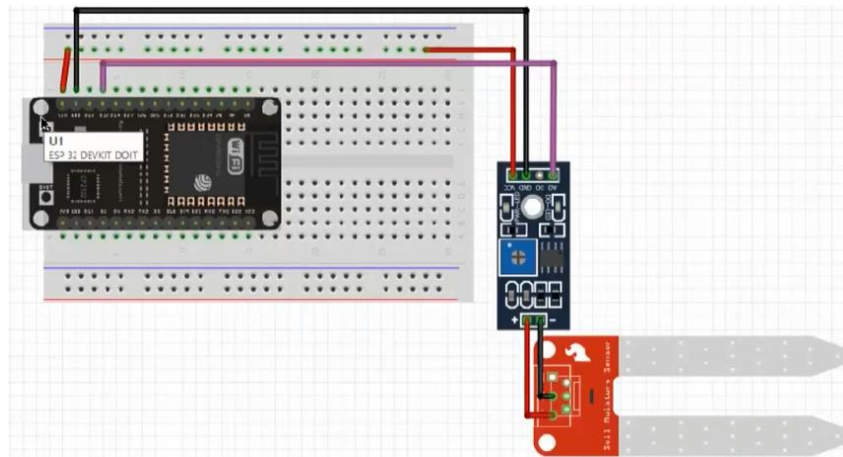


Cuenta con un integrado LM393 que se encarga de convertir la señal captada por los electrodos en una señal digital o analógica



Si queremos usar la lectura analógica, conectamos la salida A0 a una de las entradas analógicas de Arduino o ESP32.

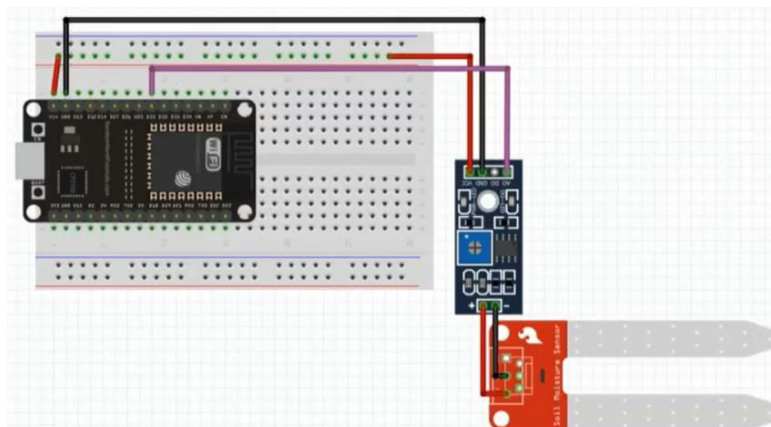
EJEMPLO 1: Mostrar los datos captados por el sensor de humedad de suelo conectado al GPIO 12



```
int suelos = 12;
void setup() {
  Serial.begin(115200);
}

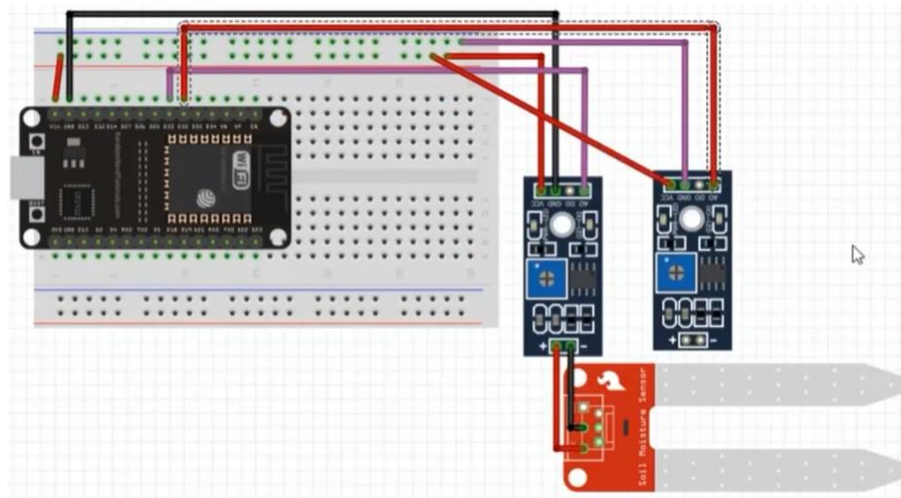
void loop() {
  int lectura = analogRead(suelos);
  Serial.println(lectura);
  delay(500);
}
```

ANEXO. Mostrar datos captados por el sensor de humedad por el GIPO 33 para que no se deshabiliten algunos pines.



```
int suelos = 33; //porque el begin esta habilitado, solo se usa los ADC1
void setup() {
  Serial.begin(115200);
}

void loop() {
  int lectura = analogRead(suelos);
  Serial.println(lectura);
}
```

Ejemplo 2: Lectura de dos sensores analogicos**Anexo1: Grafica 2 sensores al mismo tiempo**

```
int suelos = 33; //porque el begin esta habilitado, solo se usa los ADC1
int suelos2 = 32;
void setup() {
  Serial.begin(115200);
}
```

```
void loop() {
  int lectura = analogRead(suelos);
  int lectura2 = analogRead(suelos2);
  Serial.print(lectura);
  Serial.print(",");
  Serial.println(lectura2);
}
```

ANEXO2: Imprime dos sensores analógicos con frase de referencia

```
int suelos = 33; //porque el begin esta habilitado, solo se usa los ADC1
int suelos2 = 32;
void setup() {
  Serial.begin(115200);
}
void loop() {
  int lectura = analogRead(suelos);
  int lectura2 = analogRead(suelos2);
  Serial.print("Sensor YL-69:");
  Serial.println(lectura);
  Serial.print("Sensor H2O:");
  Serial.println(lectura2);
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Si el sensor de humedad es menor a 2000 significa que esta mojado, imprimir por el monitor mojado. Si el sensor esta entre 2000 y 3000 esta húmedo y si no esta ninguno de estos esta seco.

LABORATORIO N° 18

SENSOR BAROMETRICO BMP

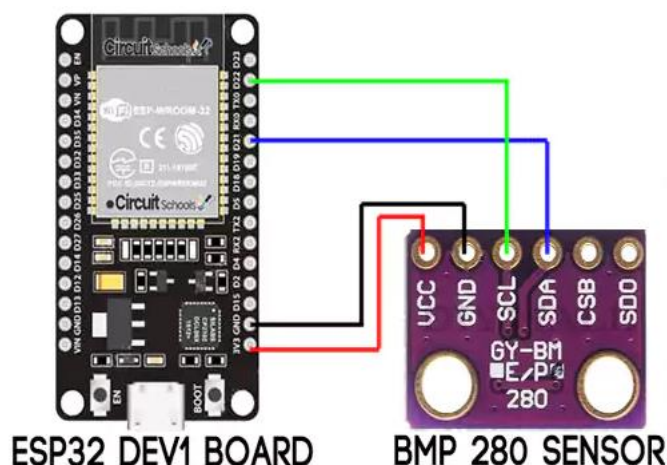
➤ **Objetivo**

Diseñar e implementar un circuito digital en base al microcontrolador ESP32 que permita datos de un sensor barométrico BMP

SENSOR BMP 280

El sensor barométrico BMP 280 Mide la presión entre 300 y 1100hPa, la altura 0 A 9000 metros y temperatura -40 °C a 85 °C

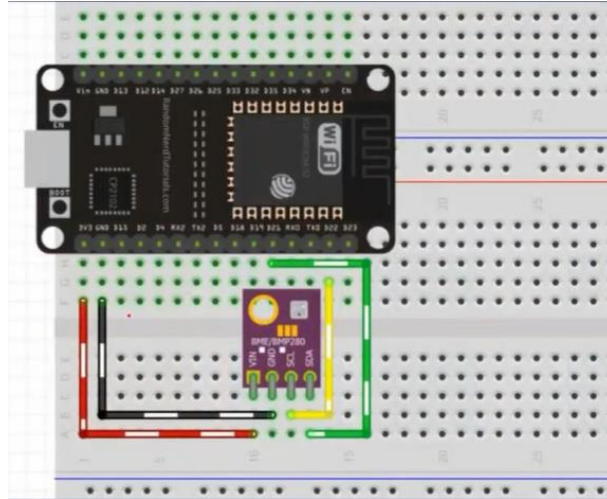
Comunicación I2C y SPI. Trabaja con la librería adafruit BMP 280



PROTOCOLOS DE COMUNICACIÓN

- **UART (TX/RX)** se utiliza BT wifi, reconocimiento de voz GPS GPRS Y MP3
- **I2C (SDA, SCL)** señal de datos y señal del reloj puedo conectar varios dispositivos. Pantalla lcd, pn532, rtc y bmp280 bmp180
- **SPI** utiliza 4 señales (MISO, MOSI, SS y SCLK) los dispositivos que maneja son Ethernet, sd, bmp280 y bmp180
- **PWM**, señales analógicas

EJEMPLO1 : Imprimir por el monitor serial los valores obtenidos por el sensor BMP280 a razón de 2 segundos



Descargar las librerías Adafruit BMP280 y adafruit sensor

https://drive.google.com/drive/u/1/folders/1FqijP-bc_TNmj7QfTnZCBh3QBb6HlcDT

Ejemplo 1.a

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_Sensor.h>
Adafruit_BMP280 bmp;
float t = 0, p = 0, a = 0;
void setup() {
  Serial.begin(115200);
  if (!bmp.begin()) {
    Serial.println("Error en el sensor");
    while (1); //bucle infinito
  }
}
void loop() {
  t = bmp.readTemperature();
  p = bmp.readpressure();
  a = bmp.readAltitude(1153.1); //1153.1 cambia segun a la localidad
  //Serial.print("Temperatura:");
  //Serial.println(t);
  Serial.println("Temperatura: " + String(t) + "*C"); //String convierte t a cadena
  Serial.println("Presion: " + String(p) + "Pa");
  Serial.println("Altitud: " + String(a) + "m");
  delay(2000);
}
```

Ejemplo 1.b

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
```

```
#include <Adafruit_Sensor.h>
Adafruit_BMP280 bmp;
float t = 0, p = 0, a = 0, p0=0;
void setup() {
  Serial.begin(115200);
  if (!bmp.begin()) {
    Serial.println("Error en el sensor");
    while (1); //bucle infinito
  }
  //p0 = bmp.readAltitude()/100; //p -> Hp
}

void loop() {
  t = bmp.readTemperature();
  p = bmp.readPressure();
  a = bmp.readAltitude(647); //1153.1 cambia segun a la localidad
  //Serial.print("Temperatura:");
  //Serial.println(t);
  Serial.println("Temperatura: " + String(t) + "*C"); //String convierte t a cadena
  Serial.println("Presion: " + String(p) + "Pa");
  Serial.println("Altitud: " + String(a) + "m");
  delay(2000);
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Imprimir por el monitor serial y pantalla LCD los valores obtenidos por el sensor BMP280 a razón de 2 segundos

LABORATORIO N° 19

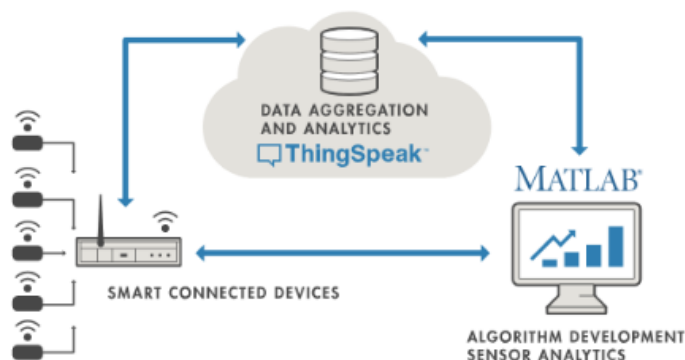
ESP32 THINGSPEAK IOT

➤ **Objetivo**

Diseñar e implementar un circuito digital en base al microcontrolador ESP32 que permita datos de un sensor barométrico BMP mediante thingspeak.

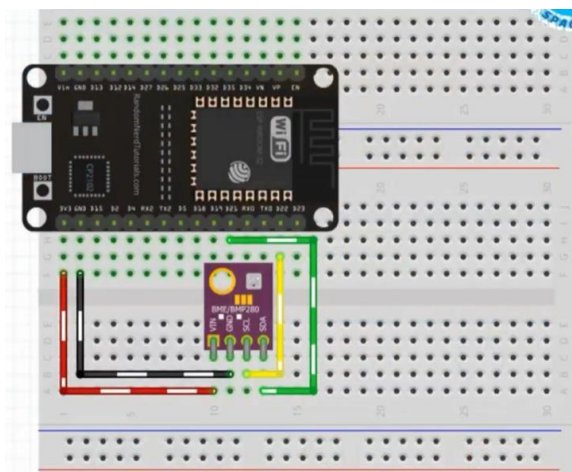
THINGSPEAK IOT

Es un plataforma de Internet of Things (IoT) que permite recoger y almacenar datos de sensores en la nube en tiempo real y desarrollar aplicaciones IoT. Thingspeak también ofrece aplicaciones que permiten analizar y visualizar tus datos en MATLAB y actuar sobre los datos. Los datos de los sensores pueden ser enviados desde Arduino, Raspberry Pi, BeagleBone Black y otro HW.



EJEMPLO1:

Graficar en Thingspeak los valores de presión atmosférica temperatura y altitud cada 15 segundos



Paso 1: Crear una cuenta en thingspeak

<https://thingspeak.com/>

Crear cuenta

Create MathWorks Account

Email Address

marciocarvajal@gmail.com

i To access your organization's MATLAB license, use your school or work email.

Location

Bolivia

First Name

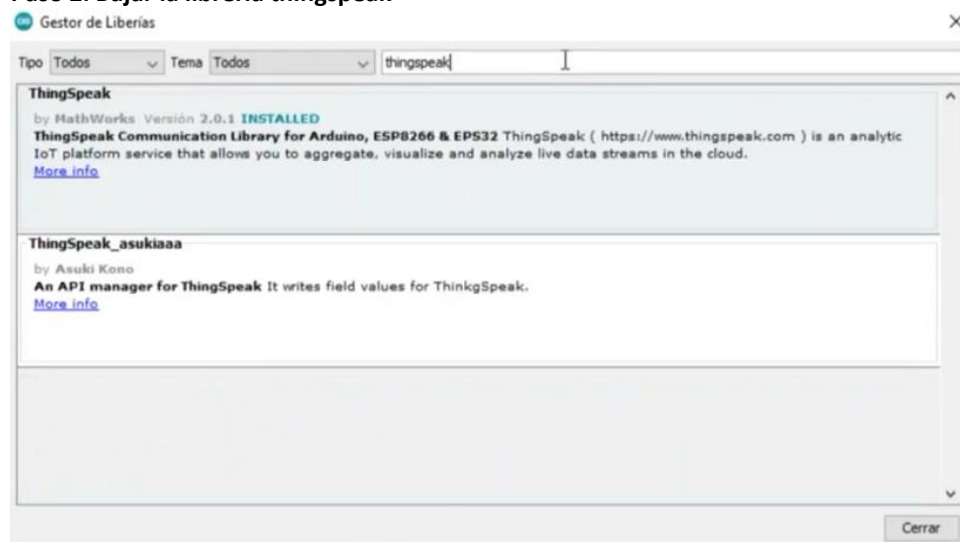
MARCIO

Last Name

CARVAJAL

Continue

Cancel

Paso 2. Bajar la librería thingspeak

```
#include <Wire.h>
#include <WiFi.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_Sensor.h>
#include "ThingSpeak.h"
#include <DHT.h>
const char* ssid = "Familia CB";
const char* password = "K6V8Q8Gb";
unsigned long ID_canal = 1800908; //id del canal en thingspeak
const char* KEY = "RTFQ2FW9EKXBTQBP"; //Key del canal
```

```
WiFiClient cliente;
Adafruit_BMP280 bmp;
DHT sensor(12, DHT11);
float t = 0, p = 0, a = 0, p0 = 0, h = 0;
int suelos = 34, lectura = 0;
void setup() {
```



```
Serial.begin(115200);
if (!bmp.begin()) {
  Serial.println("Error en el sensor");
  while (1); //bucle infinito
}
p0 = bmp.readAltitude() / 100; //p -> Hp
WiFi.begin(ssid, password); //agregamos los credenciales de nuestra red
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println();
Serial.println("Conectado al WiFi");
ThingSpeak.begin(cliente);
sensor.begin();
delay(5000);
}

void loop() {
  t = bmp.readTemperature();
  p = bmp.readPressure();
  a = bmp.readAltitude(p0); //1153.1 cambia segun a la localidad
  h = sensor.readHumidity();
  lectura=analogRead(suelos);
  ThingSpeak.setField(1, t);
  ThingSpeak.setField(2, p);
  ThingSpeak.setField(3, a);
  ThingSpeak.setField(4, h);
  ThingSpeak.setField(5, lectura);
  ThingSpeak.writeFields(ID_canal, KEY);
  Serial.println("Datos enviados correctamente!");
  delay(15000); //es el tiempo de retardo para comunicarse con thingspeak
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Imprimir por el monitor serial los valores obtenidos por el sensor BMP280 a razón de 2 segundos

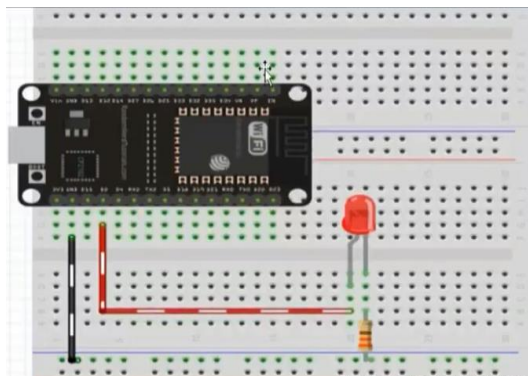
LABORATORIO N° 20

ESP32 BLYNK IOT

➤ **Objetivo**

Diseñar e implementar un circuito digital en base al microcontrolador ESP32 que permita leer datos de una estacion meteorologica mediante Blynk IOT.

EJEMPLO1: Encender y apagar un led desde Blynk IOT



Paso 1. Entrar a la pagina <https://blynk.io/> e ir a **LOG IN**
blynk entrar con su correo y contraseña

Paso 2. Ir al correo y precionar crear Password

Paso 3. Create new template

Start by creating your first template

Template is a digital model of a physical object. It is used in Blynk platform as a template to be assigned to devices.

[+ New Template](#)

Create New Template

NAME
LED IOT

HARDWARE
ESP32

CONNECTION TYPE
WiFi

DESCRIPTION
Proyectos empleando el ESP32

28 / 128

Cancel

Done

LED IOT

Info Metadata Datastreams Events Automations Web Dashboard Mobile Dashboard

TEMPLATE NAME
LED IOT

HARDWARE
ESP32

CONNECTION TYPE
WiFi

DESCRIPTION
Proyectos empleando el ESP32

28 / 128

TEMPLATE ID
TMLqLqGICjWl

MANUFACTURER
My organization 2309PU

OFFLINE IGNORE PERIOD
00 hrs 00 mins 00 secs

HOTSPOT PREFIX
Hotspot Prefix

TEMPLATE IMAGE (OPTIONAL)

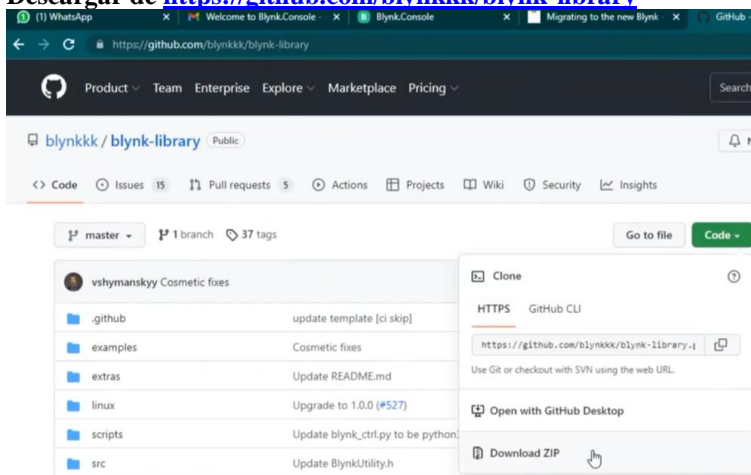
Add image
Upload from computer or drag-n-drop
.png or .jpg, minimum width 500px

FIRMWARE CONFIGURATION

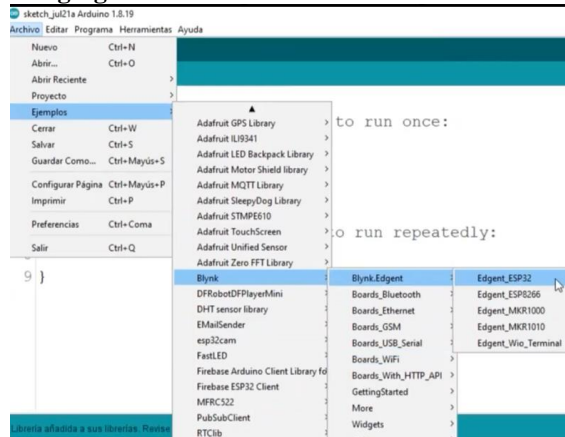
```
#define BLYNK_TEMPLATE_ID "TMLqLqGICjWl"
#define BLYNK_DEVICE_NAME "LED IOT"
```

Template ID and Device Name should be included at the top of your main firmware

Descargar de <https://github.com/blynkkk/blynk-library>



Se agrega en el IDE de Arduino la librería descargada



```
#define BLYNK_TEMPLATE_ID "TMPLZFkNscRf"
#define BLYNK_DEVICE_NAME "LED IOT"

#define BLYNK_FIRMWARE_VERSION    "0.1.0"

#define BLYNK_PRINT Serial
// #define BLYNK_DEBUG

#define APP_DEBUG

// Uncomment your board, or configure a custom board in Settings.h
// #define USE_WROVER_BOARD
// #define USE_TTGO_T7
// #define USE_TTGO_T_OI
// #define USE_ESP32C3_DEV_MODULE
// #define USE_ESP32S2_DEV_KIT

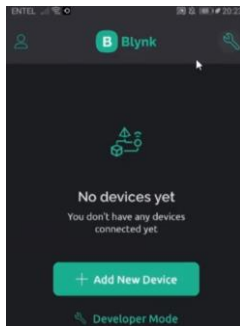
#include "BlynkEdgent.h"

void setup()
{
  Serial.begin(115200);
  delay(100);

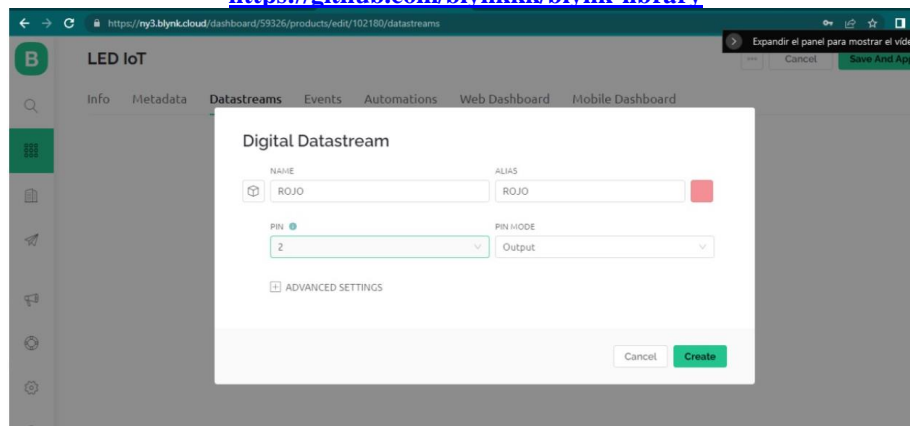
  BlynkEdgent.begin();
}

void loop() {
  BlynkEdgent.run();
}
```

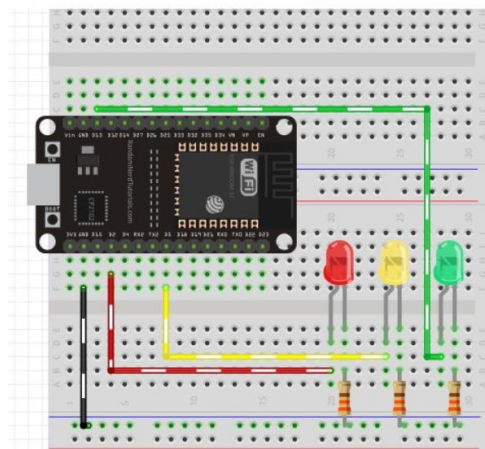
En el APP Blynk (celular) configurar



<https://github.com/blynkkk/blynk-library>



Ejemplo 2



```
#define BLYNK_TEMPLATE_ID "TMPLZFkNscrf"
#define BLYNK_DEVICE_NAME "LED IOT"
```

```
#define BLYNK_FIRMWARE_VERSION "0.1.0"
#define BLYNK_PRINT Serial
// #define BLYNK_DEBUG
#define APP_DEBUG
```

```
// Uncomment your board, or configure a custom board in Settings.h
// #define USE_WROVER_BOARD
// #define USE_TTGO_T7
// #define USE_TTGO_T_OI
// #define USE_ESP32C3_DEV_MODULE
// #define USE_ESP32S2_DEV_KIT

#include "BlynkEdgent.h"
const byte verde=13;

void setup()
{
  Serial.begin(115200);
  delay(100);
  pinMode(verde, OUTPUT);

  BlynkEdgent.begin();
}

void loop() {
  BlynkEdgent.run();
}

BLYNK_WRITE(V0){
  int estado=param.asInt();
  digitalWrite(verde, estado);
}
```

PRÁCTICA PARA EL ESTUDIANTE:

Crear una estación meteorológica y enviarlos datos a thingspeak cada 15 segundos con Blynk

