

GUÍA DEL CURSO **ARDUINO KIT**

René Domínguez Escalona



Autor de la guía: René Domínguez Escalona

Proyectos, diagramas de circuitos y textos elaborados por: René Domínguez Escalona

Imágenes libres de la red pertenecientes a sus respectivos creadores

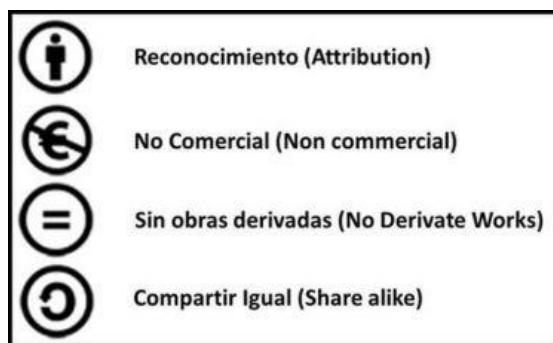
Obra basada en: Curso Arduino Kit en YouTube

El logotipo y nombre de Arduino son marcas de Arduino, registradas en U.S.A y el resto del mundo.

La mención de otros productos y nombre de compañías dentro del libro son marcas de sus respectivas compañías.

Licencia de Creative Commons

Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.



Este libro por ningún motivo debe venderse o modificarse sin permiso del autor.

ÍNDICE

INTRODUCCIÓN A ARDUINO	3
INTRODUCCIÓN AL LENGUAJE C DE ARDUINO.....	15
ARDUINO STARTER KIT	27
CONFIGURACIÓN INICIAL	30
PROTOBOARD Y CABLES	38
LED	41
ARDUINODROID.....	43
POTECIÓMETRO.....	45
PUSH BUTTON	48
PWM (Modulación por Ancho de PulsoS)	51
RGB	54
LDR (FOTORRESISTENCIA)	57
LCD 16X02.....	60
DHT11	64
LM35.....	68
RTC RELOJ DE TIEMPO REAL DS1302.....	71
SERVOMOTOR SG90.....	75
SENSOR FLAMA.....	79
INFRARROJO IR	82
WATER SENSOR	85
DISPLAY 7 SEGMENTOS	88
DISPLAY 7 X4.....	93
MATRIZ DE LEDS 8X8	97
MATRIZ DE BOTONES (TECLADO).....	105
BUZZERS ACTIVO/PASIVO.....	110
SENSOR ELECTRET	113
JOSTICK ANALÓGICO (2 EJES CON BOTÓN)	115
RELEVADOR	119
MOTOR A PASOS 28BYJ-48.....	122
REGISTRO DE DESPLAZAMIENTO 74HC595	126
RFID R522	130
ULTRASÓNICO	136
CONCLUSIÓN	139

INTRODUCCIÓN A ARDUINO

PROTOTIPOS CON ARDUINO

Arduino es una placa orientada a la creación de prototipos. No sólo porque todo lo que está en la propia placa y el software están pensados para crear prototipos de una forma muy fácil y rápida. Para empezar a crear Prototipo en Arduino es necesario tener conocimientos básicos de programación en C++ y electrónica básica.

Metodología para crear un prototipo con Arduino

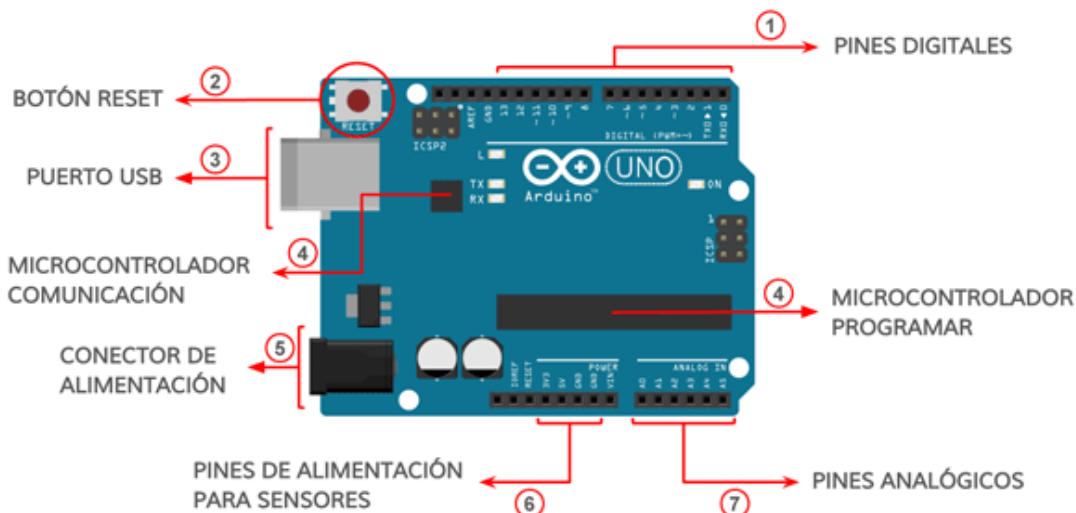
Cuando queremos llevar a cabo un **proyecto o prototipo con Arduino** o cualquier otro hardware, debemos seguir un proceso. Este proceso nos guiará y ayudará a través de diferentes fases que debemos ir afrontando a lo largo de la creación de un prototipo.

La metodología para crear prototipos se engloba en 3 pasos:

- La idea general
- El prototipo mínimo
- El diseño incremental

PARTES DE ARDUINO UNO

En la imagen siguiente se muestra los componentes centrales de una placa de Arduino UNO, dentro de los que destacan, los pines digitales, los pines analógicos, el microcontrolador, pines de alimentación y puerto USB.



Arduino cuenta con la tecnología plug and play, que permite conectar y usar el Arduino sin necesidad de hacer algo más.

ENTORNO DE DESARROLLO DE ARDUINO

Con el **IDE (Entorno de Desarrollo Integrado)** de Arduino podemos escribir código de una forma muy fácil, ya que cuenta con una guía de referencia en línea en <https://www.arduino.cc/reference/es/>

```
#include <DHT.h>

#define sensor A0
DHT dht;
void setup() {
    Serial.begin(9600);
}
void loop() {
    DHT.read11(sensor);
    Serial.print("TEMPERATURA:");
    Serial.println(dht.temperature);
    Serial.print("HUMEDAD:");
}
```

Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) en COM6

Funciones principales:

La palabra reservada nos la cambia de color.

Se puede contraer estructuras de control o funciones

Insertar y gestionar librerías a través del menú

Formatear el código.

Cargar el código a la placa.

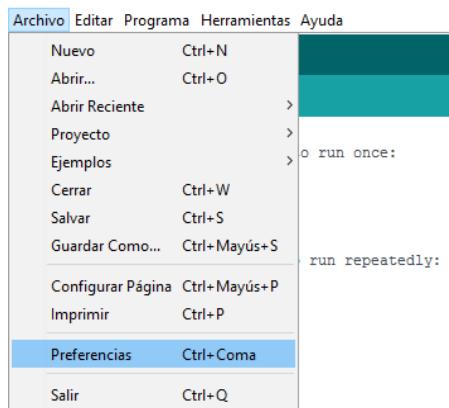
Compilar el código y subirlo a la placa.

IDE de Arduino es que es de código abierto. Esto significa que es un software de gratuito y que respeta las 4 libertades del software libre

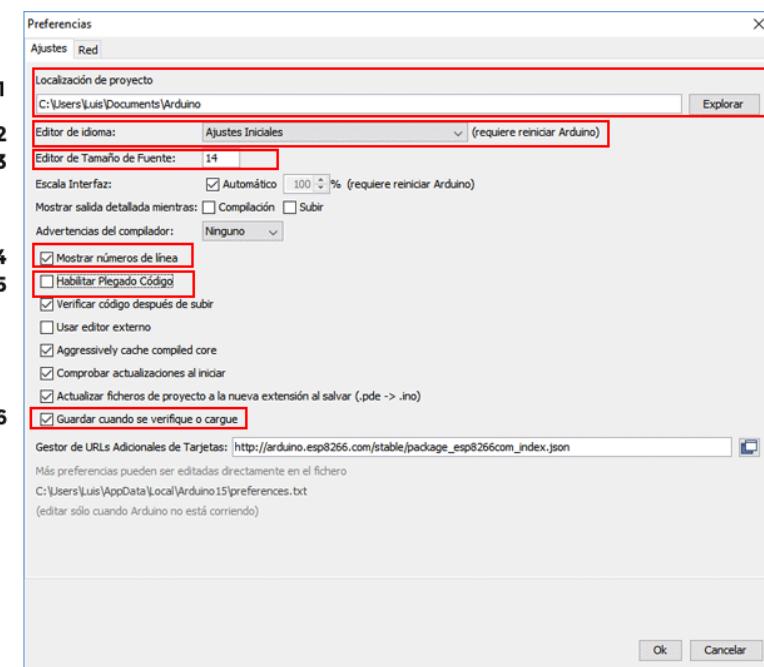
PREFERENCIAS DEL SISTEMA

Como en casi todos los programas que utilizamos, en el IDE de Arduino tenemos una opción para **configurar las preferencias del sistema**. Nos permite modificar el idioma, el tamaño de letra y algunas cosas más.

Para acceder a esta opción solo tenemos que ir al menú *Abrir>Preferencias*.



Vamos a ver las opciones más importantes que nos permiten modificar el aspecto y funcionamiento del IDE de Arduino.



Localización del proyecto: podemos seleccionar una carpeta donde iremos guardando los proyectos. Por defecto será la que ha creado el instalador en documentos/Arduino. Esta ruta varía según el sistema operativo.

Editor de idioma: con esta opción podemos cambiar el idioma del IDE.

Editor de Tamaño de Fuente: indica el tamaño de fuente del editor del IDE.

Mostrar número de línea: para que muestre los números de líneas en el editor.

Habilitar plegado el código: siempre que el código tenga una sentencia con {} nos permitirá contraer y expandir ese código. Muy útil cuando trabajamos con archivos muy grandes.

Guardar cuando se verifique o cargue: es importante que cuando verifiquemos el código o lo carguemos al microcontrolador haga un guardado automático. Déjalo marcado.

SISTEMA DE FICHEROS DE ARDUINO

La extensión de un archivo de Arduino es **.ino**.

Cuando creas un programa o sketch (esquema o bosquejo) verás que tiene una extensión. **.ino**.

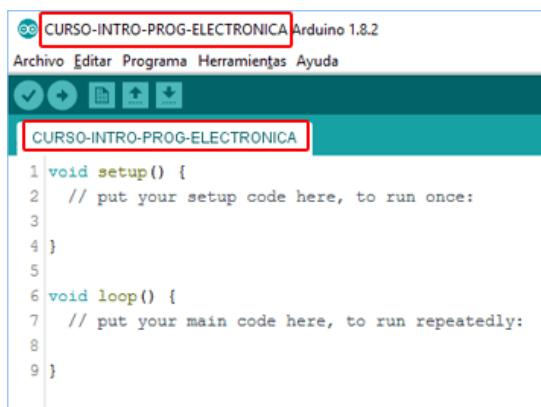
Cuando guardas un archivo en tu computadora, el propio IDE de Arduino ya lo organiza por ti y crea una carpeta con el mismo nombre que el archivo y dentro guarda el archivo.

Por ejemplo, si creas un nuevo programa y vas al menú *Archivo>Guardar*, te permitirá guardarlo con un nombre.

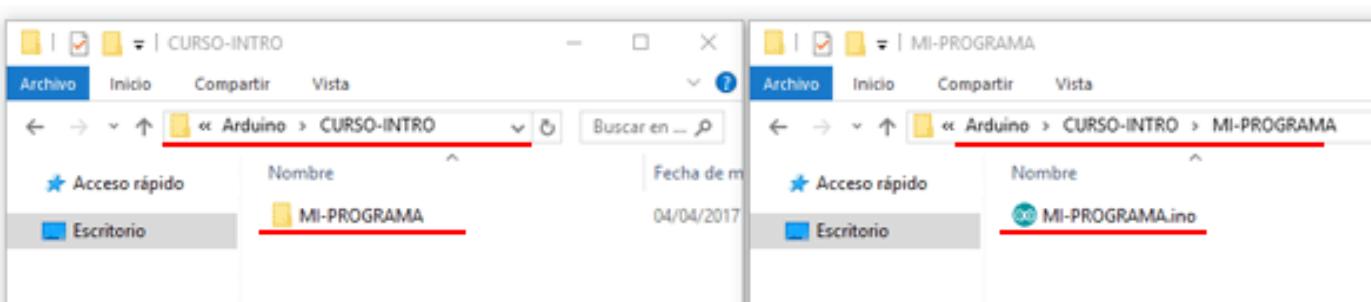


Puedes darle el nombre que quieras siempre y cuando cumplas las reglas de tu sistema operativo. Te recomiendo que sea un nombre descriptivo y que **no utilices caracteres especiales**.

Cuando haces esto suceden varias cosas. Por un lado, cambia el nombre en el IDE de Arduino. Así sabes en todo momento con qué programa estás trabajando.



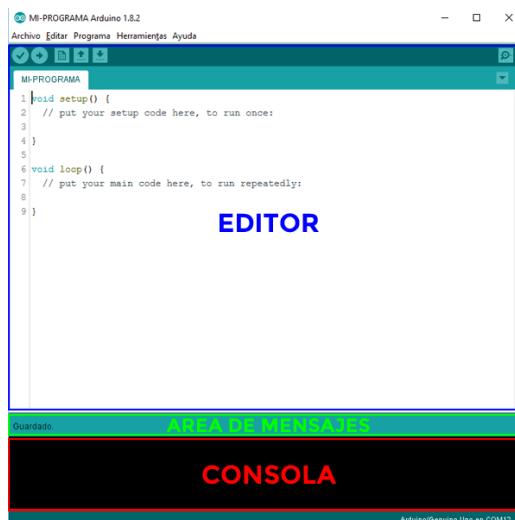
Y luego, en la ruta que hayas elegido habrá creado una carpeta con el mismo nombre y dentro el fichero.



La ruta por defecto donde se guarda es la que hemos configurado en preferencias.

PARTES FUNDAMENTALES DEL IDE DE ARDUINO

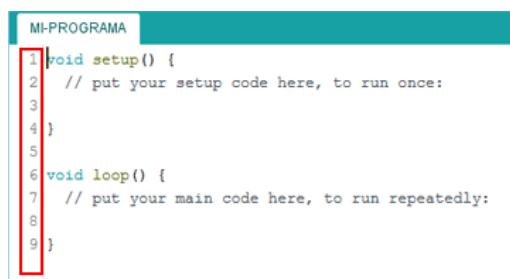
Dentro del IDE de Arduino podemos destacar 3 partes principales. El editor, el área de mensajes y la consola.



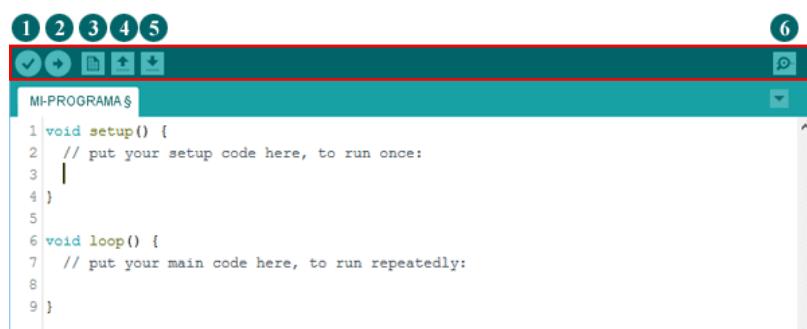
EL EDITOR

El editor es la parte del IDE donde puedes escribir el código fuente de Arduino. Pero no solo eso, también tenemos acceso a las funciones más utilizadas.

En la parte central encontramos el propio editor, incluye el número de línea útil, por ejemplo, para detectar errores.



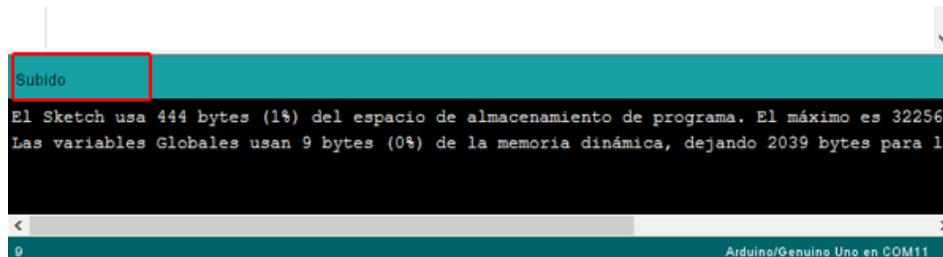
Justo arriba del editor tenemos los accesos directos a las funciones más utilizadas.



1. **Verificar/Compilar:** este botón verifica el código en busca de errores y lo compila. Cuando hablo de compilar me refiero a traducir el lenguaje de programación que entendemos los humanos en código máquina que entienden las máquinas.
2. **Subir:** el botón subir nos permite cargar o subir el código al microcontrolador a través del puerto serie USB.
3. **Nuevo:** sirve para crear un programa nuevo. Esto genera una nueva ventana donde escribir el código de ese nuevo programa.
4. **Abrir:** abre un programa que hayas guardado previamente en el disco duro.
5. **Salvar:** guarda el archivo en el disco duro. Es como la opción que hemos visto anteriormente.
6. **Monitor serie:** es una de las partes más importantes del IDE de Arduino. Sirve para mostrar información de la comunicación entre el ordenador y Arduino en las dos direcciones.

Todos estos accesos directos tienen su correspondencia en el menú de opciones y también tienen su atajo de teclado.

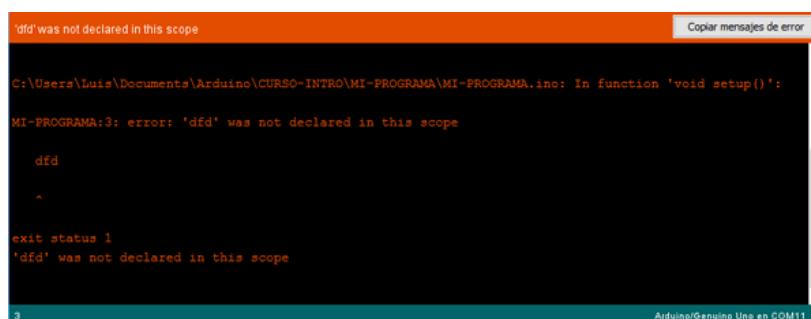
EL ÁREA DE MENSAJES



En esta área de mensajes se muestra la última acción que has realizado. También muestra mensajes cuando se está realizando alguna tarea como subiendo un programa a la placa.

LA CONSOLA

La consola nos va a dar información muy valiosa. Nos puede dar información sobre una acción concreta, por ejemplo, los datos tras subir un programa a la placa. Pero lo más importante, nos informa si hay algún error.



OTRAS PARTES IMPORTANTES DEL IDE DE ARDUINO

Una de las áreas donde podemos encontrar información muy valiosa es justo abajo del todo. Se pueden ver dos áreas de texto.

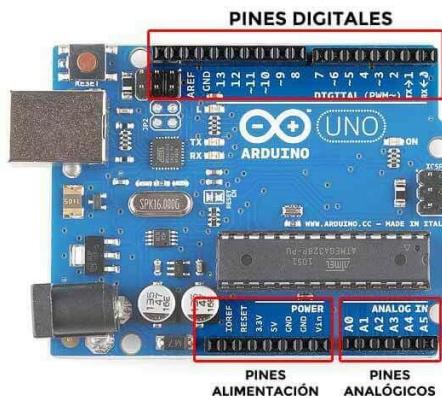
```
Subido  
El Sketch usa 444 bytes (1%) del espacio de almacenamiento de programa. El máximo es 32256  
Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 2039 bytes para la  
  
< >  
9 Arduino/Genuino Uno en COM11
```

En la parte izquierda nos informa del número de línea donde está situado el cursor. En la parte de la derecha tenemos un resumen de la placa que tenemos seleccionada y el puerto serie que estamos utilizando.

EJEMPLOS DE TIPOS DE PLACAS ARDUINO



Pines de Arduino UNO



PINES DIGITALES

Es el zócalo más grande. Tiene **14 pines numerados del 0 al 13**.

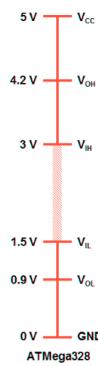


¿Qué quiere decir digital? Digital es algo abstracto así que mejor verlo con una analogía.

Imagínate que eres pintor. Cuando vas a pintar un cuadro solo te permiten utilizar dos colores: blanco y negro. Si quieres pintar con un gris claro, por ejemplo, no puedes, no existe ese color. Solo puedes elegir entre blanco o negro.

Si esto lo llevamos al mundo de la electrónica a través de un voltaje, solo podríamos tener dos voltajes. Esto es lo que ocurre en los pines digitales de Arduino donde **solo podemos tener dos estados HIGH o LOW que equivalen a 5V y 0V**.

En realidad, esto no es cierto totalmente. Podríamos tener un voltaje de 3V por ejemplo. Para estos casos hay una regla interna que determina si un voltaje es HIGH o LOW.



Estos son los niveles lógicos del microcontrolador ATMega328. Todo lo que esté entre 3V y 5V se considera nivel alto (HIGH) y todo lo que esté entre 0V y 1,5V es nivel bajo (LOW). El resto, entre 1,5V y 3V es una indeterminación.

Esto quiere decir que cualquier voltaje dentro de este rango, el microcontrolador no sabrá si es estado HIGH o LOW.

Además, los pines digitales pueden funcionar en 3 modos diferentes:

Modo entrada (INPUT): puede leer voltajes. Por ejemplo, ¿está pulsado un botón? si (HIGH) o no (LOW).

Modo salida (OUTPUT): puede suministrar un voltaje. Por ejemplo, encender/apagar un led on (HIGH) o off (LOW).

Excepción (PWM): algunos pines del microcontrolador pueden funcionar en modo salida suministrando un valor entre el rango 0V y 5V. Esto ya no sería un pin digital. Estos pines van marcados con el símbolo ~ y hay 6 dentro de la placa de Arduino (3, 5, 6, 9, 10, 11).

Por último, señalar que los **pines 0 y 1 son Rx (recibir) y Tx (transmitir)**. Se utilizan para la comunicación serie entre el ordenador y Arduino y están conectados a los LEDs de la placa donde pone RX y TX. Se recomienda no utilizar estos pines.

El pin 13 es el de la mala suerte dentro de Arduino UNO es un pin que está conectado a un LED integrado dentro de la placa.

Hay algún pin más dentro de este zócalo, pero como ya te he dicho al principio de este curso de Arduino, aquí vamos a ver lo esencial para ponernos en acción.

PINES ANALÓGICOS

Es el zócalo donde pone **ANALOG IN** y van numerados del **A0 al A5**, 6 pines.



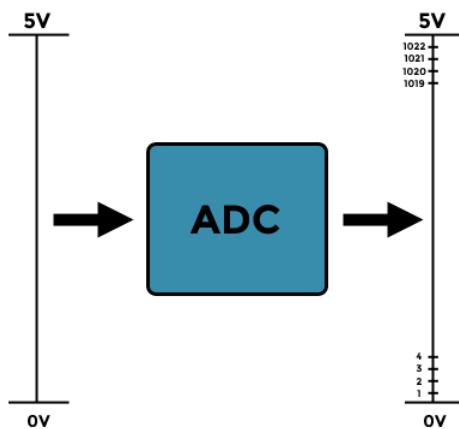
¿Recuerdas al pintor que solo podía pintar con dos colores, blanco o negro? Pues en el mundo analógico tenemos una amplia gama de colores, ahora podemos pintar con diferentes tonos de gris.

Si nos llevamos esto al mundo de la electrónica con Arduino, **con estos pines podemos medir diferentes voltajes entre 0V y 5V**. Es decir, podemos tener un voltaje de 3,5V en uno de estos pines y Arduino sería capaz de leerlo.

Sin embargo, existe un problema. **El microcontrolador no entiende de números decimales, sólo entiende datos digitales 1's y 0's**. Para resolver esto, la MCU incorpora un ADC (son las siglas de *Analog Digital Converter* o en español Conversor Analógico Digital).

Por otro lado, Arduino no es capaz de medir cualquier voltaje, me explico. ¿Cuántos números hay entre 0 y 5? realmente hay infinitos números. Puedes empezar con el 0 e ir aumentando de 0,000001 o en 0,0000000000000001.

La consecuencia de todo esto es que **Arduino solo entiende datos digitales y además estos deben estar acotados**. El ADC se encargará de convertir esos valores en datos digitales y además solo podrán ser un número concreto de valores. A esto último **se le llama resolución**.



El **ADC** que viene integrado dentro de la MCU que lleva **Arduino UNO** tiene una **resolución de 10-bit**. Esto equivale a que solo vamos a poder medir 1024 valores posibles que van del 0 al 1023. Básicamente lo que estamos haciendo es dividir el rango de 0V a 5V en 1024 partes.

PINES DE ALIMENTACIÓN

El zócalo de pines de alimentación nos sirve para alimentar los componentes, sensores y actuadores.



Hay que destacar 4 de todos los que hay:

3,3V: suministra ese voltaje por ese pin.

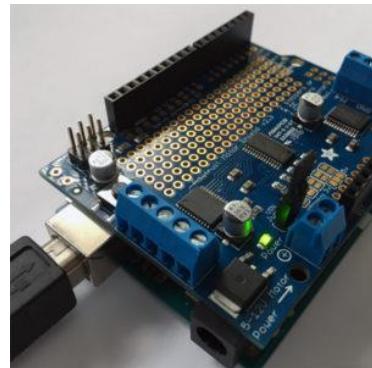
5V: suministra ese voltaje por ese pin.

GND: hay dos pines con esta función además del que está en el zócalo de los pines digitales. Es la toma de tierra y por donde debemos cerrar el circuito.

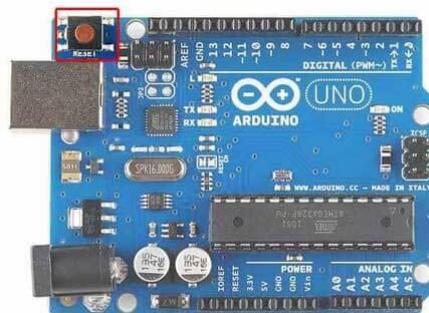
Otras características a destacar

Por último vamos a ver una serie de características secundarias que es importante destacar.

La huella que forman los pines se ha convertido en un estándar para conectar los shields.



El botón reset resetea la placa y hace que empiece a ejecutar el código desde el principio.



El LED de encendido nos informa si la placa está alimentada.



El pin Vin nos da otra alternativa a la hora de alimentar Arduino con un voltaje de entre 6V y 12V. De momento te recomiendo que lo alimentes a través del puerto USB.



Conector jack de alimentación. Es igual que el pin Vin pero a través de un conector jack. El voltaje de alimentación que soporta es de 6V a 12V.



INTRODUCCIÓN AL LENGUAJE C DE ARDUINO

CARACTERES

Los caracteres del lenguaje en C pueden agruparse en letras, dígitos, espacios en blanco, caracteres especiales, signos de puntuación y secuencias de escape.

LETRAS, DÍGITOS, ETC.

Estos caracteres son utilizados para formar las constantes, los identificadores y las palabras clave del lenguaje C. Son los siguientes:

- Las letras mayúsculas del alfabeto inglés:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- Las letras minúsculas del alfabeto inglés:
a b c d e f g h i j k l m n o p q r s t u v w x y z
- Dígitos enteros
0 1 2 3 4 5 6 7 8 9 ...
- Carácter de subrayado “_”

El compilador trata las letras mayúsculas y minúsculas como caracteres diferentes.

ESPACIOS EN BLANCO

Los espacios en blanco, el tabulador (todos los tipos), avance de página y nueva línea, son caracteres denominados en blanco, debido a que la labor que desempeñan es la misma que la del espacio en blanco: actuar como separadores, lo cual permite escribir un programa mucho más limpio y legible. Por ejemplo:

```
main() { printf("Hola Mundo.\n"); }
```

Puede escribirse de una forma más legible así:

```
main()
{
printf("Hola Mundo.\n");
}
```

Los espacios en blanco en exceso son ignorados.

CARACTERES ESPECIALES Y SIGNOS DE PUNTUACIÓN

Este grupo de caracteres se utilizan de diferentes formas, para indicar que un identificador es una función o una matriz; para especificar una determinada operación aritmética lógica o de relación, etc. Son los siguientes:

```
, ; : ¿ ‘ “ ( ) [ ] { } < ! | / \ ~ + # % & ^ * - =
```

TIPO DE DATOS

Veamos el siguiente programa que lleva a cabo una operación aritmética la cual realiza una suma de dos valores:

```
variable1 = 80;  
variable2 = 10;  
resultado = variable1 + variable2;
```

Para que el compilador reconozca esta operación es necesario especificar previamente el tipo de variable para cada uno de los operandos que intervienen en la misma, así como el tipo de variable del resultado.

Para ello escribiremos una línea como la siguiente:

```
int variable1, variable2, resultado;  
variable1 = 80;  
dato2 = 10;  
variable2 = variable1 + variable2;
```

La declaración anterior le dice al compilador que variable1, variable2 y resultado son del tipo entero (int). Observe que se puede declarar más de una variable del mismo tipo empleando una lista separada por comas.

Los tipos de datos se clasifican como: tipos primitivos y tipos derivados.

TIPOS PRIMITIVOS

Se les llama primitivos porque están definidos por el compilador. Hay siete tipos primitivos de datos que se clasifican en: variables tipo enteros y variables tipo reales.

Tipos enteros: char, short, int, long y enum.

Tipos reales: float y double.

Cada tipo primitivo abarca un rango diferente de los valores positivos y negativos.

El tipo de datos que se seleccione para declarar las variables de un determinado programa dependerá del

rango y del tipo de valores que vayan a almacenar cada una de ellas, así como si son enteros o fracciones.

CHAR

El tipo char declara datos enteros entre -128 y +127. Un tipo char se define como un conjunto de 8 bits,

de los cuales uno es para especificar el signo y el resto para el valor; dicho conjunto de bits recibe el

nombre de byte. El siguiente ejemplo declara la variable b tipo char y le asigna el valor inicial de 0. Es

recomendable iniciar todas las variables que se declaren.

```
char t = 0;
```

Los valores de 0 a 127 corresponden con los 128 primeros caracteres de los códigos internacionales ASCII,

ANSI o UNICODE empleados para la representación de caracteres.

El siguiente ejemplo declara la variable car del tipo char a la que se le asigna el carácter 'a' como valor

inicial. Las cuatro declaraciones siguientes son idénticas:

```
char caracter = 't';
char caracter = 116; // la 't' es el decimal 116 en ASCII
char caracter = 0x74; // la 't' es en hexadecimal 0074
char caracter = 0164; // la 't' en octal es 0164
```

SHORT

El tipo short, abreviatura de signed short int, permite declarar datos enteros comprendidos entre -32768

y +32767. Un valor short se define como un dato de 16 bits de longitud, independientemente de la

plataforma utilizada. El siguiente ejemplo declara x y y como variables enteras del tipo short:

```
short x = 0, y = 0;
```

INT

El tipo int, abreviatura de signed int permite declarar datos enteros comprendidos entre -2147483647 y

+2147483647. Un valor int se define como un dato de 32 bits de longitud. El siguiente ejemplo declara e

inicia tres variables a, b y c, de tipo int.

```
int a = 2000;
int n = -30;
int c = 0XF003 // valor en hexadecimal
```

En general, el uso de enteros de cualquier tipo produce un código compacto y rápido.

LONG

El tipo long se utiliza para declarar datos enteros comprendidos entre los valores -214748648 y +2147483647. Un valor long se define como un dato de 32 bits de longitud. El siguiente ejemplo declara e inicia las variables a, b y c, del tipo long.

```
long a = -1L /* L indica que la constante -1 es long */
long b = 125;
long c = 0x1F00230F; /* valor en hexadecimal */
```

FLOAT

El tipo float se utiliza para declarar un dato en coma flotante de 32 bits en el formato IEEE 754. Los datos de tipo float almacenan valores con una precisión aproximada de 7 dígitos. Para especificar que una constante es un tipo float, hay que añadir al final de su valor la letra 'f' o 'F'. En el siguiente ejemplo se declaran las variables a, b y c:

```
float a = 3.141592f;
float b = 2.2e-5F /* 2.2e-5 = 2.2 por 10 elevado a -5 */
float c = 2/3.0F; /* 0.666667 */
```

DOUBLE

El tipo double se utiliza para declarar un dato en coma flotante de 64 bits. Los datos de tipo double almacenan valores con una precisión aproximada de 16 dígitos. Por omisión, una constante es considerada del tipo double. En el siguiente ejemplo se declaran las variables x, y y z:

```
double x = 3.141592; // una constante es double por omisión
double y = 2.2e+8F // 2.2e+5 = 2.2 por 10 elevado a 8
double z = 5/4.0;
```

COMENTARIOS

Un comentario es un mensaje que se encuentra en el código fuente. Añadiendo comentarios se hace más fácil la comprensión de un programa para el programador, la finalidad de los comentarios es explicar el código fuente o incluir mensajes importantes. El compilador soporta dos tipos de comentarios:

- Comentario tradicional. Un comentario tradicional empieza con los caracteres /** y finaliza con */. Estos comentarios pueden ocupar más de una línea, pero no pueden anidarse, y pueden aparecer en cualquier lugar donde se permita aparecer un espacio en blanco. Por ejemplo:

```
/**  
 * Esto es un comentario tradicional,  
 */
```

- Comentario de una sola línea. Este tipo de comentario comienza con una doble barra (//) y se extiende hasta el final de la línea. Esto quiere decir que han incorporado algunas características de interés de C++; una de ellas es esta. La siguiente línea muestra un ejemplo:

```
// Agregar aquí el código de iniciación
//Este es otro comentario de una sola linea
```

DECLARACIÓN DE CONSTANTES SIMBÓLICAS

Declarar una constante simbólica significa decirle al compilador el nombre de la constante y su valor. Esto se hace generalmente antes de la función main utilizando la directriz #define, cuya sintaxis es así:

```
#define NOMBRE VALOR
```

El siguiente ejemplo declara la constante real PI con el valor de 3.14159, la constante de un solo carácter NL con el valor '\n' y la constante de caracteres MENSAJE con el valor "Pulse una tecla para continuar\n":

```
#define PI 3.14159
#define SALTO '\n'
#define MENSAJE "Pulse una tecla para continuar\n"
```

Observe que no hay un punto y coma después de la declaración. Esto es así, porque una directriz no es una sentencia, sino una orden para el preprocesador. El tipo de una constante es el tipo del valor asignado.

Suele ser habitual escribir el nombre de una constante en mayúsculas.

CONSTANTES

Otra de las características incorporadas por los compiladores es la palabra reservada const. Utilizándola disponemos de una forma adicional para declarar una constante; basta con anteponer el calificador const al nombre de la constante seguido del tipo de la misma; si el tipo se omite, se supone int. Por ejemplo, la siguiente línea declara la constante real Pi con el valor 3.14:

```
const double E = 2.71;
```

Una vez declarada e iniciada una constante, ya no se puede modificar su valor. Por ello, al declararla debes ser iniciada.

DECLARACIÓN DE UNA VARIABLE

Una variable representa un espacio de memoria para almacenar un valor de un determinado tipo. El valor de una variable, a diferencia de una constante, puede cambiar su valor durante la ejecución de un programa. Para utilizar una variable en un programa, primero hay que declararla. La declaración de una variable consiste en enunciar el nombre de la misma y asociarle un tipo:

```
tipo nombre, nombre,...
```

En el siguiente ejemplo se declaran e inician cuatro variables: una del tipo char, int, float y double:

```
char c = 't';
main() {
    int i = 0;
    float f = 0.0F;
    double d = 0.0;
//...
}
```

El tipo, primitivo o derivado, determina los valores que puede tomar la variable así como las operaciones que pueden realizarse con ella. Los operadores serán expuestos más adelante.

En el ejemplo anterior se puede observar que hay dos lugares en donde es posible realizar la declaración de una variable: fuera de todo bloque, entendido por bloque un conjunto de sentencias encerradas entre el carácter '{' y el carácter '}', y dentro de un bloque de sentencias.

En nuestro ejemplo, se ha declarado la variable c antes de la función main (fuera de todo bloque) y las variables i, f y d dentro de la función (dentro de un bloque). Una variable declarada fuera de todo bloque se dice que es global porque es accesible en cualquier parte del código desde su declaración hasta el final del fichero fuente. Por el contrario, una variable declarada dentro de un bloque, se dice que es local porque solo es accesible dentro de este.

Según lo expuesto, la variable c es global y las variables y, f y d son locales.

INICIACIÓN DE UNA VARIABLE

Las variables globales son iniciadas por omisión por el compilador: las variables numéricas con 0 y los caracteres con '0'. También pueden ser iniciadas explícitamente, como hemos hecho en el ejemplo anterior con la variable c. En cambio, las variables locales no son inicializadas por el compilador, por lo tanto, depende de nosotros iniciarlas o no.

EXPRESIONES NUMÉRICAS

Una expresión es un conjunto de operandos unidos mediante operadores para especificar una operación determinada. Cuando todas las expresiones se evalúan retornan un valor. Por ejemplo:

```
t + 1
sum + c
cantidad * precio
x = 7 * sqrt(t) - x / 2
```

OPERADORES

Los operadores son funciones a las cuales se les ha asignado un símbolo que indican cómo se manipulan los datos. Se pueden clasificar en los siguientes grupos: aritméticos, relacionales, lógicos, unitarios, a nivel de bits, de asignación, operador condicional, etc.

OPERADORES ARITMÉTICOS

Los operadores aritméticos los utilizamos para realizar operaciones matemáticas y son los siguientes:

Operador	Operación
+	Suma. Los operandos pueden ser enteros o reales.
-	Resta. Los operandos pueden ser enteros o reales.
*	Multiplicación. Los operandos pueden ser enteros o reales
/	División. Los operandos pueden ser enteros o reales. Si ambos operandos son enteros el resultado es entero, en el resto de los casos el resultado es real.
%	Módulo o resto de una división entera. Los operandos tienen que ser enteros.

El siguiente ejemplo muestra cómo utilizar estos operadores.

```
int t =10, b = 3, n;
float x = 2.0F, y;
y = x + t; // El resultado es 12.0 de tipo float
n = t / b; // El resultado es 3 del tipo int
n = t % b; // El resultado es 1 del tipo int
y = t / b; // El resultado es 3 de tipo int. Se convierte a float
```

```
para ser asignado a y
n = x / y; // El resultado es 0.666667 de tipo float. Se convierte
a int para asignarlo a n (n = 0)
```

Cuando en una operación aritmética los operandos son de diferentes tipos, ambos son convertidos al tipo del operando de precisión más alta.

OPERADORES DE RELACIÓN

Los operadores de relación o de comparación permiten evaluar la igualdad y la magnitud. El resultado de una operación de relación es un valor booleano verdadero o falso (1 o 0). Los operadores de relación son

los siguientes:

Operador	Operación
<	El primer operando <i>menor que</i> el segundo
>	El primer operando <i>mayor que</i> el segundo
<=	El primer operando <i>menor o igual que</i> el segundo
>=	El primer operando <i>mayor o igual que</i> el segundo
!=	El primer operando <i>distinto que</i> el segundo
==	El primer operando <i>igual que</i> el segundo

Los operandos tienen que ser de un tipo primitivo. Por ejemplo:

```
int r = 10, t = 0, y = 0;
y = r == t; // y = 0 (falso) porque r no es igual a t
y = r > t; // y = 1 (verdadero) porque r es mayor que t
y = r != t; // y = 1 (verdadero) porque r no es igual a t
```

Un operador de relación equivale a una pregunta relativa sobre cómo son dos operandos entre sí. Por ejemplo, la expresión `r == t` equivale a la pregunta ¿x es exactamente igual a y? Una respuesta si equivale a un valor verdadero (1) y una respuesta no equivale a un valor falso (0).

OPERADORES LÓGICOS

El resultado de una operación lógica (AND, OR, XOR y NOT) es un valor booleano verdadero o falso (1 o 0). Las expresiones que dan como resultado valores booleanos (véanse los operadores de relación) pueden combinarse para formar expresiones booleanas utilizando los operadores lógicos indicados a continuación.

Los operandos deben ser expresiones que den un resultado verdadero o falso.

Operador	Operación
&&	<i>AND.</i> Da como resultado verdadero si al evaluar cada uno de los operandos el resultado es verdadero. Si uno de ellos es falso, el resultado es falso. Si el primer operando es falso, el segundo operando no es evaluado.

	<i>OR.</i> El resultado es falso si al evaluar cada uno de los operandos el resultado es falso. Si uno de ellos es verdadero, el resultado es verdadero. Si el primer operando es verdadero, el segundo operando no es evaluado.
!	<i>NOT.</i> El resultado de aplicar este operador es falso si al evaluar su operando el resultado es verdadero, y verdadero en caso contrario.
^	<i>XOR.</i> Da como resultado verdadero si al evaluar cada uno de los operandos el resultado de uno es verdadero y el del otro falso; en otro caso el resultado es falso.

El resultado de una operación lógica es de tipo int o booleana. Por ejemplo:

```
int o = 10, p = 0, q = 0;
q = (o != 0) && (p != 0); // q = 0 (falso)
```

Los operandos del operador `&&` son: `o != 0` y `p != 0`. El resultado de la expresión `o != 0` es verdadero porque `o` vale 10 y `p != 0` es falso porque `p` es 0. Por lo tanto, el resultado de verdadero `&&` falso es falso.

OPERADORES UNARIOS

Los operadores unarios se aplican a un solo operando y son siguientes: `!`, `-`, `~`, `++` y `--`. El operador `!` ya lo hemos visto y los operadores `++` y `--` los veremos más adelante.

Operador	Operación
<code>~</code>	Complemento a 1 (cambia los ceros por unos y unos por ceros). El carácter <code>~</code> es el ASCII 126. El operando debe de ser de un tipo primitivo entero.
<code>-</code>	Cambia el signo al operando (esto es, se calcula el complemento a dos que es el complemento 1 más 1). El operando puede ser de un tipo primitivo entero o real.

El siguiente ejemplo muestra cómo utilizar estos operadores:

```
int a = 2, b = 0, c = 0;
c = -a; // resultado c = -2
c = ~b; // resultado c = -1
```

OPERADORES A NIVEL DE BITS

Estos operadores permiten realizar con sus operandos las operaciones AND, OR, XOR y desplazamientos, bit por bit. Los operandos tienen que ser enteros.

Operador	Operación
<code>&</code>	Operación AND a nivel de bits.
<code> </code>	Operación OR a nivel de bits.
<code>^</code>	Operación XOR a nivel de bits.
<code><<</code>	Desplazamiento a la izquierda rellenando con ceros por la derecha.

>>	Desplazamiento a la derecha rellenando con el bit de signo por la izquierda.
----	--

Los operandos tienen que ser de un tipo primitivo entero.

```
int a = 255, r = 0, m = 32;
r = a & 017; // r = 15. Pone a cero todos los bits de a excepto los
4 bits de menor peso.
r = r | m. | m // r = 47. Pone a 1 todos los bits de r que estén a 1 en
r = a & ~07; // r = 248. Pone a 0 los 3 bits de menor peso de a.
r = a >> 7; // r = 1. Desplazamiento de 7 bits a la derecha.
r = m << 1; // r = 64. Equivale a r = m * 2
r = a >> 7; // r = 16. Equivale a r = m / 2
```

OPERADORES DE ASIGNACIÓN

El resultado de una operación de asignación es el valor almacenado en el operando izquierdo, lógicamente después de que la asignación se ha realizado. El valor que se agrega es convertido implícitamente o explícitamente al tipo del operando de la izquierda.

Operador	Operación
++	Incremento.
--	Decremento.
=	Asignación simple.
*=	Multiplicación más asignación.
/=	División más asignación.
%=	Módulo más asignación.
+=	Suma más asignación.
-=	Resta más asignación.
<<=	Desplazamiento a izquierda más asignación.
>>=	Desplazamiento a derecha más asignación.
&=	Operación AND sobre bits más asignación.
 =	Operación OR sobre bits más asignación.
^=	Operación XOR sobre bits más asignación.

A continuación, se muestran algunos ejemplos con estos operandos.

```
int x = 10, y = 1;
x++; // Incrementa el valor de n en 1
x--; // Decrementa el valor de n en 1
y +=2 // Realiza la operación i = i + 2
```

SENTENCIAS DE CONTROL

En esta parte se aprenderá a escribir código para que un programa tome decisiones y sea capaz de ejecutar bloques de sentencias repetidas veces.

SENTENCIA IF

La sentencia if permite tomar una decisión para ejecutar una acción u otra, esta decisión es del tipo booleana ya sea verdadero o falso y la sintaxis es la siguiente.

```
if(condición)
sentencia 1;
[else
sentencia 2];
```

donde la condición es una expresión lógica o relacional y sentencia 1 y sentencia 2 representan el código que quieren que se ejecute.

Una sentencia if se ejecuta de la forma siguiente:

1. Se evalúa la expresión condición.
2. Si el resultado de la evaluación de la condición es verdadero se ejecutará la sentencia 1.
3. Si el resultado de la condición es falso se ejecutará la sentencia 2.
1. Si el resultado de la condición es falso y no se ha puesto la cláusula else, la sentencia 1 será ignorada.

A continuación, se exponen algunos ejemplos de cómo se utiliza la sentencia if.

```
int a = 5, b = 4;
if(a < b){
    printf("a es menor que b");
}else{
    printf("a no es menor que b");
}
```

En este ejemplo, la condición esta impuesta por una expresión de relación. Si al evaluar la condición se cumple que a es menor que b lo cual es falso, entonces imprimirá un mensaje el cual es “a es menor que b”, como sabemos que la condición es falsa se ejecuta la sentencia dos que imprime el mensaje “a no es menor que b”.

Con esto queda por visto la sentencia de control if.

ESTRUCTURA ELSE IF

Esta estructura else if es una estructura consecuente de la sentencia if, en la cual se evalúan diferentes casos, su forma general es:

```
if(condición 1)
sentencia 1;
else if(condición 2)
sentencia 2;
else if(condición 3)
sentencia 3;
...
else
sentencia n;
```

La estructura else if se evalúa así: Si se cumple el primer caso (condición 1), se ejecuta lo que se encuentra en la sentencia 1 y si no se cumple se examina secuencialmente los siguientes casos (condiciones) hasta llegar al último else if. Si ninguna condición es verdadera entonces se ejecutara la sentencia n que corresponde al último else. El siguiente ejemplo se muestra cómo funciona:

```

int a = 10, b = 5, c = 11;
if(a < b){
printf("a es menor que b");
}else if(a == b){
printf("a es igual que b");
}else if(a > c){
printf("a es mayor que c");
}else{
printf("Ningún caso es verdadero");
}

```

En este ejemplo podemos observar que las condiciones son falsas porque así lo hemos hecho, pero primero se evalúa la condición 1 que es `a < b` y si no se cumple sigue con la condición 2 que es `a == b` hasta llegar a una condición verdadera que es el último `else` ya que las anteriores han sido falsas.

SENTENCIA SWITCH

La sentencia switch permite ejecutar varias acciones en función de una expresión. Es una sentencia para decisiones múltiples dado un determinado valor el cual se le da a la expresión. Su sintaxis es:

```

switch(expresión) {
case [expresión-constante 1]:
sentencia 1;
break;
case [expresión-constante 2]:
sentencia 2;
break;
...
default:
sentencia n;
}

```

donde `expresión` es la variable que almacenará o recibirá los datos de cada caso (`case`). La sentencia Switch evalúa la expresión entre paréntesis y compara su valor con las constantes de cada `case`. La ejecución de las sentencias del bloque de la sentencia switch, comienza en el `case` cuya constante coincide con el valor de la expresión y continúa hasta el final del bloque, si no existe ninguna variable para `case` entra al `default`, un `default` sería como un `else` poniendo como ejemplo la sentencia `if`.

Existen también otras sentencias, como lo es la sentencia `while` que se seguía ejecutando dependiendo de su condición y la sentencia `do while` que es muy parecida a `while` pero estas se dejan al estudio del lector.

SENTENCIA FOR

La sentencia for nos permite ejecutar una o varias líneas de código repetitivamente a un número determinado de veces. Su sintaxis es:

```

for(val1 = val2/const; condición 1 ; condición 2){
sentencia;
}

```

La sentencia for se evalúa de la siguiente forma:

1. A val1 se le asigna un valor constante o el valor de alguna otra variable.
2. Se evalúa la condición:
 - a. Si la condición 1 es verdadera respecto a val1, se ejecuta la sentencia respecto a condición 2 y así sucesivamente dependiendo de esta.
 - b. Si la condición 1 es falsa respecto a val1, la sentencia for termina.

Ahora veremos algunos ejemplos, el primer ejemplo nos imprimirá los valores desde el 0 al 100, existen dos modos de hacer esto y depende de la condición 1.

```
for(int i = 0; i <= 100 ; i++) {
    printf("%d", i);
}
```

Como podemos ver en este ejemplo la variable i comienza en el valor constante 0, la primera condición declara que tiene que ser menor o igual a 100, o sea que llegará al valor a 100, ahora en la condición 2 que ya se veo anteriormente hace un incremento en 1. Ahora en el siguiente ejemplo la condición 1 cambia, pero hace lo mismo que el primer ejemplo.

```
for(int i = 0; i < 101 ; i++) {
    printf("%d", i);
}
```

La explicación es igual a la anterior solo que en la condición 1 se evalúa un < explícito, o sea que imprimirá

hasta el número 100, pero cuando llegue a 101 se detiene la sentencia for.

```
for(int i = 100; i >= 1 ; i--) {
    printf("%d", i);
}
```

Este ejemplo es similar a los anteriores, pero al revés, empieza en el número 100 y se hace un decremento en 1 cuando se evalúa la condición 1, como en las explicaciones anteriores

ARDUINO STARTER KIT

El Arduino starter Kit incluye todo el material necesario para seguir los tutoriales de inicio con el objetivo de aprender a programar en C++ e iniciarse en la electrónica básica y en los sensores, de un modo sencillo.

Arduino kit es ideal para educación y para introducir a los alumnos en el mundo del Arduino Uno, ya que cuenta con todo el material necesario en la caja.

MATERIAL INCLUIDO EN EL KIT ARDUINO

IMAGEN	NOMBRE
	Arduino UNO
	Cable USB, para conectarlo a tu PC directamente
	LCD 16x2 con interface I2C para simplificar la conexión.
	Un display 7 segmentos
	Un display 7 segmentos de 4 dígitos
	Una matriz de puntos LED de 8x8
	Joystick de 2 ejes más un botón
	Un teclado de 16 botones
	Un reloj RTC con interface I2C. <i>DS1302</i>
	Un sensor de agua / Lluvia



Un sensor de temperatura y humedad
DHT11



Un LED RGB con PCB



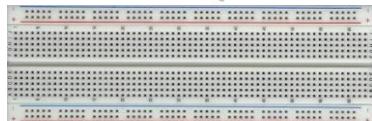
Motor a pasos 28BYJ-48 con controlador



Servomotor SG90



Relevador / Relay



Protopboard



Cables de colores Macho/Macho



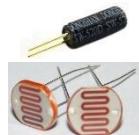
10 cables Macho / Hembra



1 x Detector de sonido



2 Buzzer Activo/Pasivo



Sensor de inclinación



2 Fotoresistencias o LDR

4 Push Button



5 LED rojo



5 LED amarillo



5 LED verde



Colección de resistencias de 220 ohm,
1kohm y 10 Kohm



Sensor de flama



Sensor de Temperatura LM35



Registro de Desplazamiento



74HC595 Shift register



Receptor IR Infrarrojo



Control IR Infrarrojo



Adaptador de pila de 9V



Modulo receptor de RFID Rc522



Tarjeta RFID



Identificador RFID



Potenciómetro 5Kohm



LED RGB

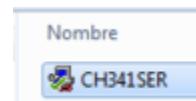
CONFIGURACIÓN INICIAL

INSTALAR DRIVER CH340

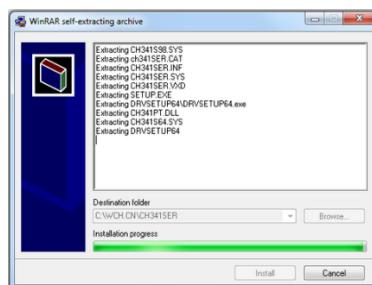
Lo primero que debemos hacer es descargar el software CH340 que se encuentra en el siguiente enlace, este controlador es esencial para el funcionamiento del Arduino genérico:

<https://cdn.shopify.com/s/files/1/0557/2945/files/CH341SER.EXE?9597575494380707938>

Una vez descargado, veremos el siguiente archivo:

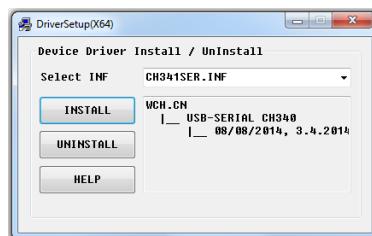


Ejecutamos el archivo y a continuación nos desplegará la siguiente pantalla:

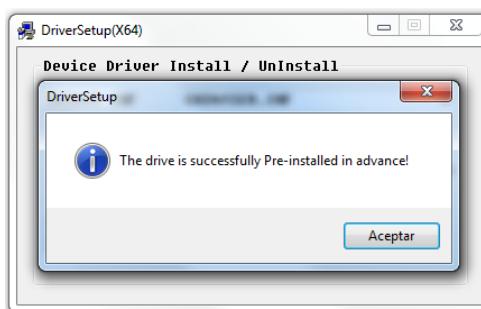


Durante la instalación, el ordenador nos preguntará si deseamos ejecutar el archivo, seleccionaremos "*Ejecutar*".

Posteriormente aparecerá la siguiente pantalla:

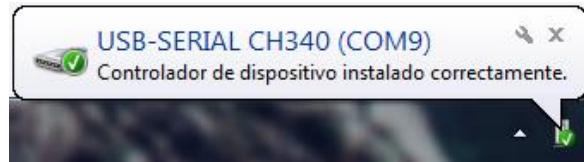


Seleccionaremos "*INSTALL*" y con ello finalizaremos el proceso de instalación, desplegando el siguiente mensaje:



Y seleccionaremos "*Aceptar*".

A continuación, conectaremos nuestro Arduino genérico (cualquier modelo) a el ordenador, y en la esquina inferior derecha podremos el siguiente mensaje, el cual nos indica que podemos comenzar a usar nuestro Arduino de forma normal:



Podemos también abrir el Administrador de Dispositivos, para comprobar que el driver fue instalado correctamente, en este caso nuestro ordenador lo detectó como "USB-SERIAL CH340 (COM9)".

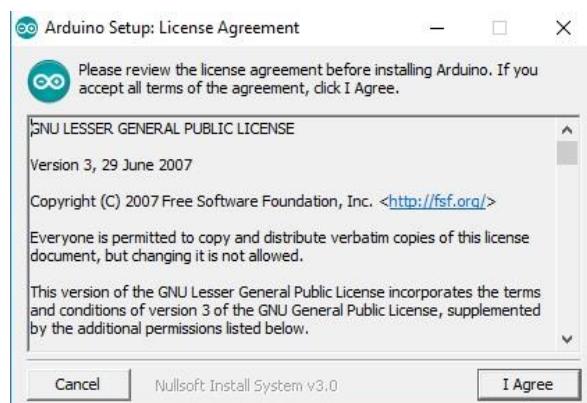


INSTALAR ARDUINO IDE

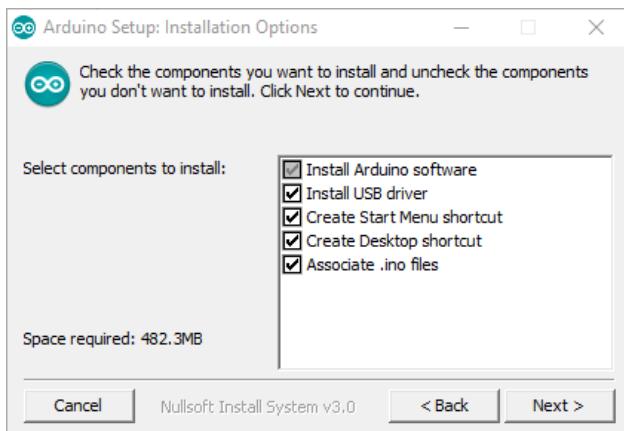
Ejecuta el instalador del programa y sigue los pasos de instalación.



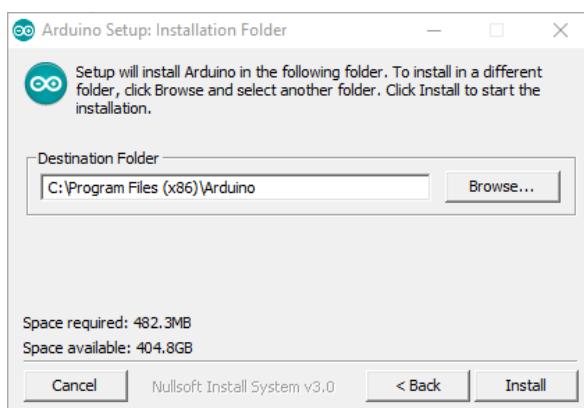
Acepta los términos y condiciones de la licencia.



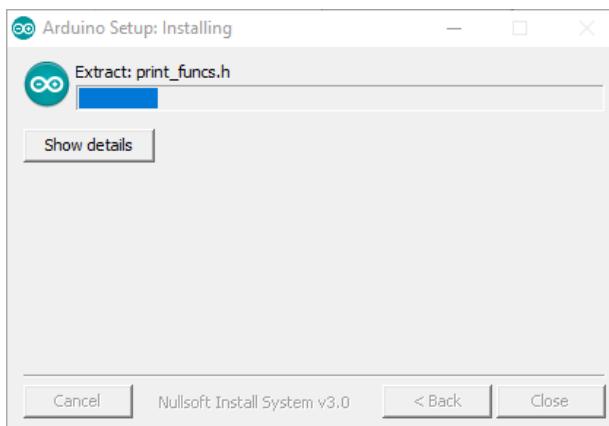
Selecciona todas las opciones para que instale todos los complementos y drivers necesarios.



Selecciona la ruta de instalación y presiona «install».



Espera un par de minutos que termine el proceso de instalación.

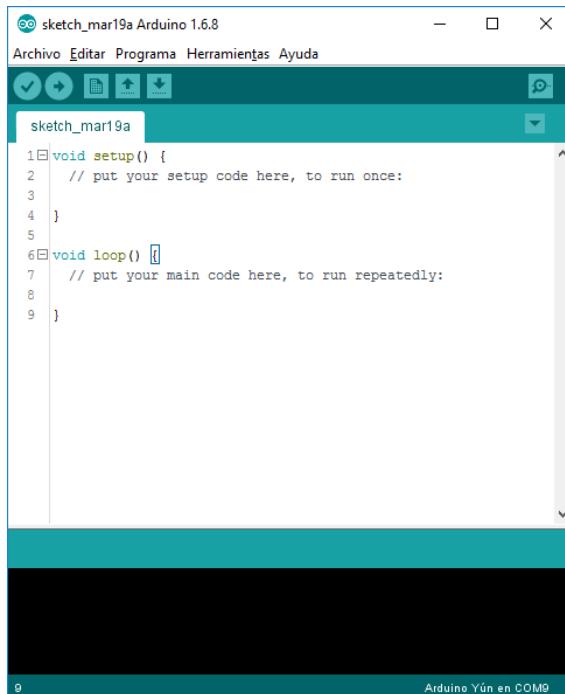


En este momento ya tenemos instalado el IDE en nuestro ordenador. Con las nuevas versiones del IDE de Arduino no es necesario instalar los drivers en Windows al venir integrados en el IDE y estos tienen las firmas correspondientes.

Ejecutar la aplicación:

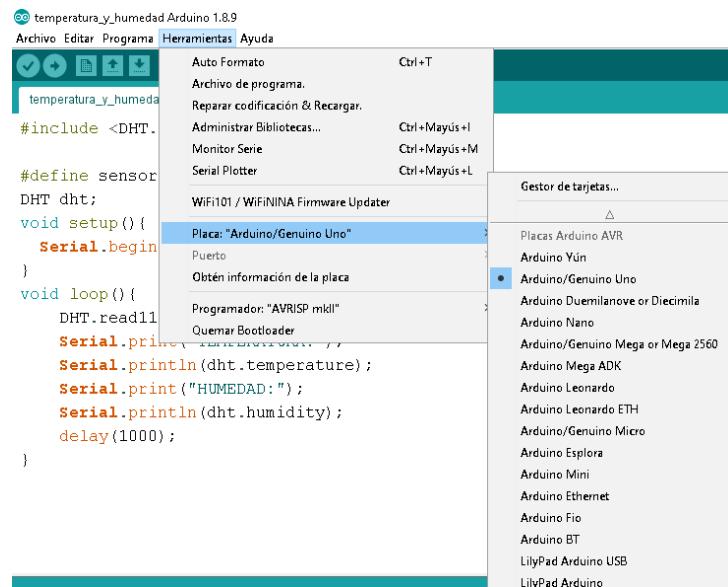


Y este es el aspecto del IDE:

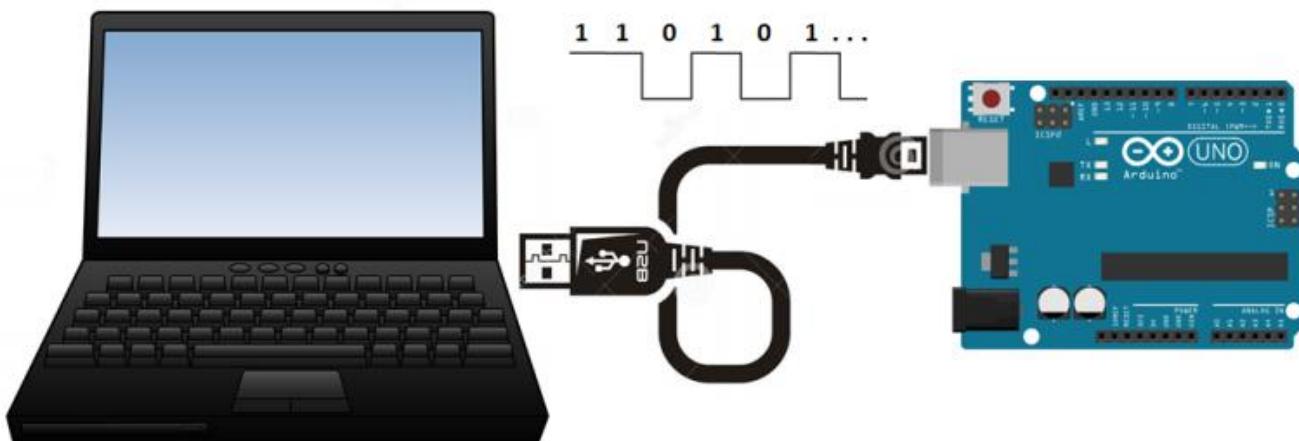


Seleccionar la placa correcta y el puerto serie

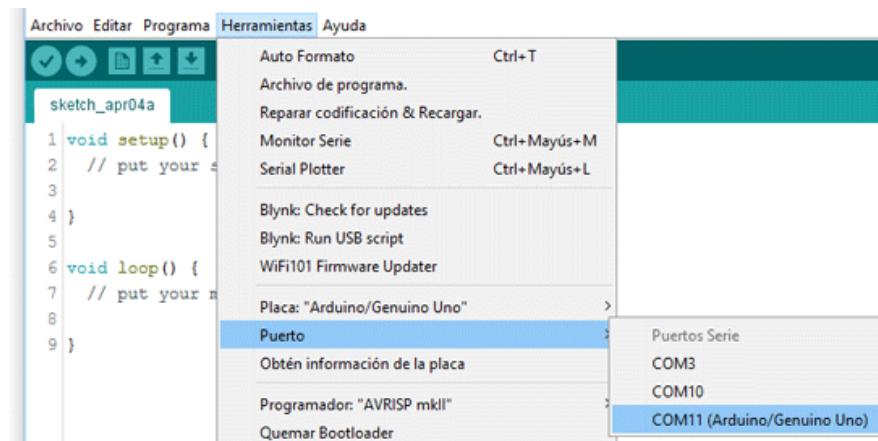
Antes de empezar a programar en la IDE tienes que seleccionar la placa a través del menú en **Herramientas>Placa>Arduino/Genuino UNO**. No hace falta que conectes la placa al ordenador para seleccionar un modelo.



El puerto serie es por donde se comunican Arduino y el ordenador. Es necesario que tengas conectado tu Arduino al ordenador.



Para seleccionar el puerto lo hacemos a través del menú *Herramientas>Puerto*. Puede que aparezca más de uno y además el nombre varía según el sistema operativo.



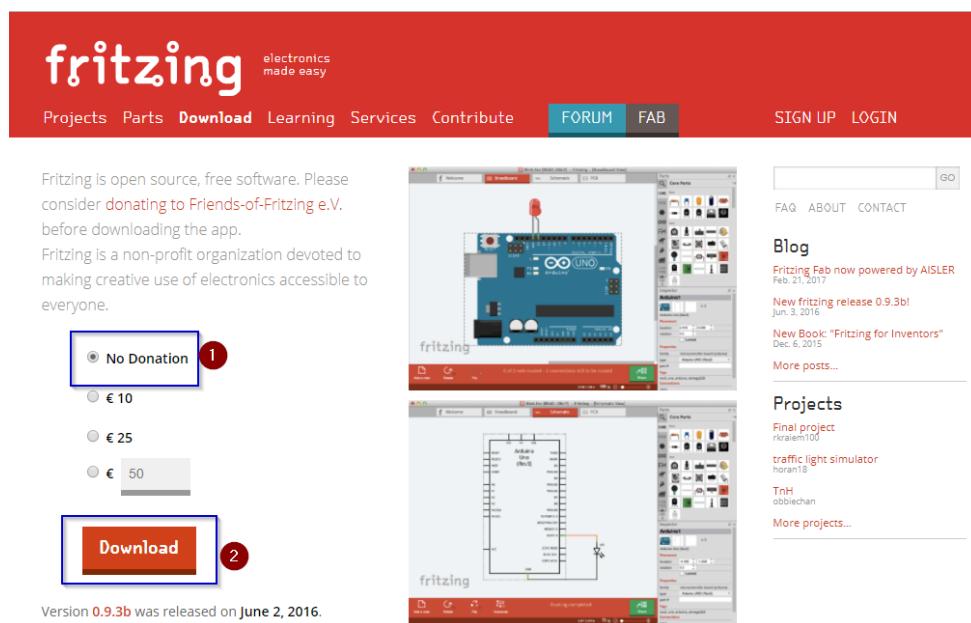
INSTALAR FRITZING

Fritzing es un programa libre de automatización de diseño electrónico que busca ayudar a diseñadores y artistas para que puedan pasar de prototipos (usando, por ejemplo, placas de pruebas) a productos finales.

Fritzing fue creado bajo los principios de Processing y Arduino, y permite a los diseñadores, artistas, investigadores y aficionados documentar sus prototipos basados en Arduino y crear esquemas de circuitos impresos para su posterior fabricación. Además, cuenta con un sitio web complementario que ayuda a compartir y discutir bosquejos y experiencias y a reducir los costos de fabricación. y su diseño de arte de artistas.

Personalmente lo ocupo para diseñar circuitos de Arduino, ya que tiene muchos componentes. Además, nosotros podemos crear los nuestros. Es gratuito y puede exportar a imagen, PDF, etcétera. Vamos a la página de descargas: <http://fritzing.org/download/>

En la siguiente página seleccionamos **No donation** y hacemos clic en **Download**:



Ahora nos llevará a otra página en donde vamos a elegir nuestro sistema operativo. Si tenemos Windows o Linux podemos elegir entre 32 y 64 bits. Si tenemos MacOS X sólo tenemos una opción.

Fritzing is open source, free software. Be aware that the development of it depends on the active support of the community.

Select the download for your platform below.

Version 0.9.3b was released on June 2, 2016.

[Windows 32 bit](#)

[Windows 64 bit](#)

[Mac OS X 10.7 and up](#)

[Linux 32 bit](#)

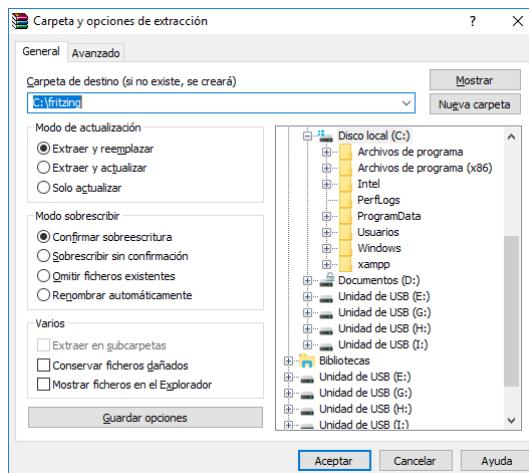
[Linux 64 bit](#)

[Source Github](#)

Downloaded 2537524 times.

En mi caso la descargaré para Windows de 64 bits. Es un archivo comprimido que podemos extraer con nuestro software de compresión favorito.

Recomiendo extraerlo en **C:\fritzing** para tenerlo como un programa normal



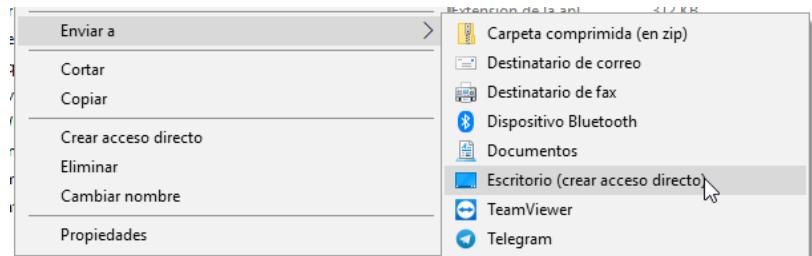
CREAR ACCESO DIRECTO

Cuando haya terminado podemos abrir la carpeta que elegimos para extraerlo y dentro habrá una carpeta llamada **fritzing.0.9.3b.64.pc** o algo parecido.

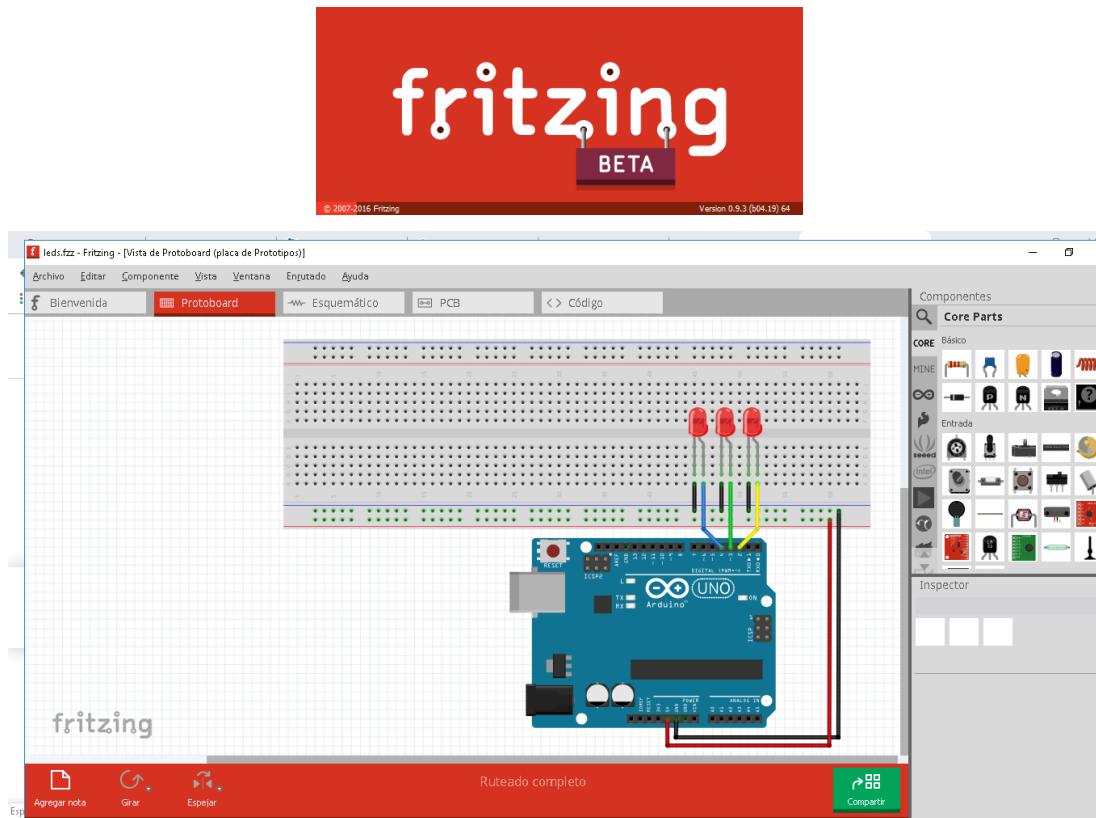
Entraremos a ella y dentro veremos muchos archivos, aunque sólo nos importa el ejecutable, ya que lo demás son componentes y librerías:

Nombre	Fecha de modifica...	Tipo	Tamaño
fritzing-parts	10/06/2016 11:53 a...	Carpeta de archivos	
help	10/06/2016 11:51 a...	Carpeta de archivos	
lib	10/06/2016 11:51 a...	Carpeta de archivos	
platforms	10/06/2016 11:51 a...	Carpeta de archivos	
sketches	10/06/2016 11:51 a...	Carpeta de archivos	
translations	10/06/2016 11:53 a...	Carpeta de archivos	
Fritzing	10/06/2016 11:22 a...	Aplicación	8,020 KB
git2.dll	10/06/2016 11:00 a...	Extensión de la apl...	3,121 KB

Con un recuadro azul marqué la aplicación. Ahora podemos crear un acceso directo a ella o anclarlo al inicio o a la barra de herramientas.



Y finalmente podemos abrirlo:



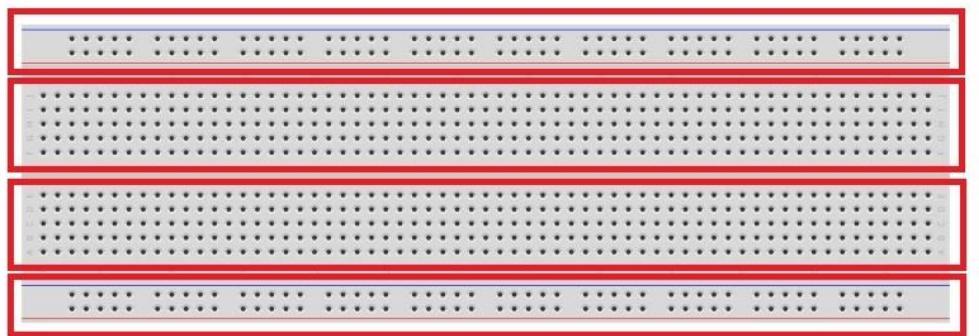
PROTOBOARD Y CABLES

La protoboard (breadboard en inglés) es una placa que posee unos orificios conectados eléctricamente entre sí siguiendo un patrón horizontal o vertical. Es empleada para realizar pruebas de circuitos electrónicos, insertando en ella componentes electrónicos y cables como puente. Es el boceto de un circuito electrónico donde se realizan las pruebas de funcionamiento necesarias antes de trasladarlo sobre un circuito impreso. Esta placa puede llamarse de varias formas, las más comunes son protoboard, breadboard, placa protoboard o incluso placa de pruebas.

Partes de una placa protoboard (breadboard)

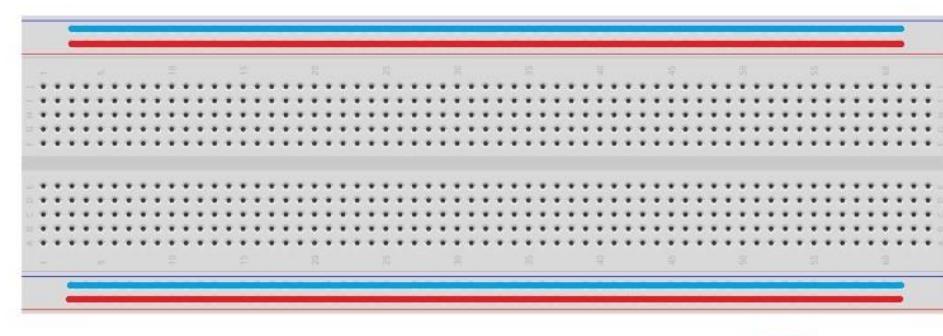
Existen muchos **modelos de placas protoboard**, se pueden diferenciar principalmente por la cantidad de orificios que poseen, pero por lo general en todos los **tipos de placas de pruebas** podemos diferenciar tres partes:

- En uno de los extremos o en los dos, podemos tener la **zona de alimentación**.
- Para conectar los componentes entre si se emplea la **zona de conexiones superior** o **zona de conexión inferior**.



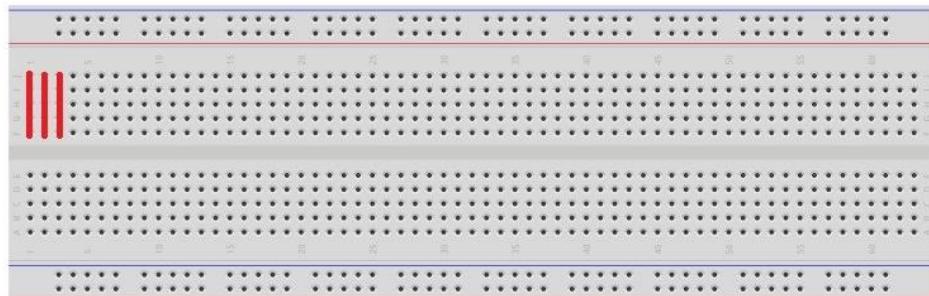
Zona de alimentación

La zona de alimentación está compuesta por orificios horizontales conectados entre sí eléctricamente a lo largo de toda la placa. Son dos líneas independientes; una para alimentación (+) y otra para masa (-). Normalmente las protoboard tienen dos zonas de alimentación situadas en lados opuestos para distribuir diferente alimentación.



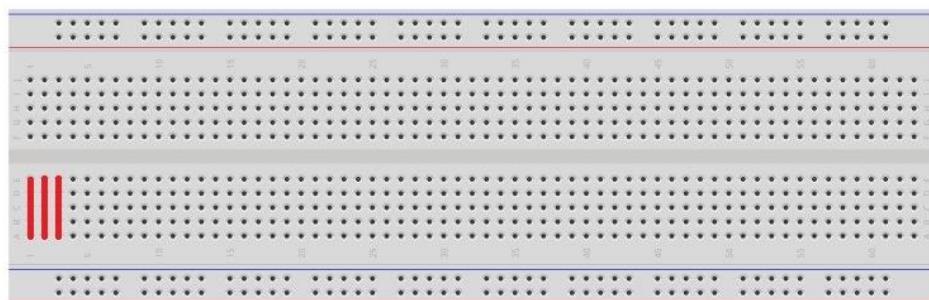
Zona de conexiones superior

La zona de conexiones superior está compuesta por columnas de orificios conectados eléctricamente entre sí. Cada columna es independiente eléctricamente con las demás, es decir, los orificios solo están conectados de forma vertical.

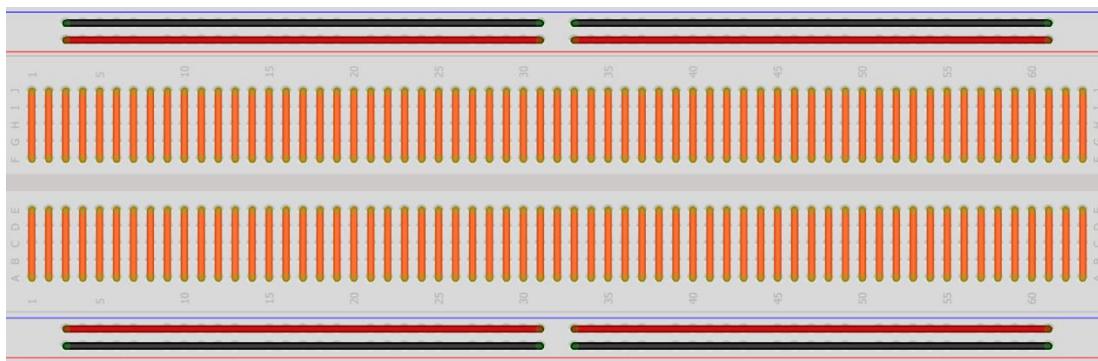


Zona de conexiones inferior

La zona de conexiones inferior es igual a la zona de conexiones superiores. Ambas zonas están separadas eléctricamente. Estas dos zonas son muy necesarias para la inserción de circuitos integrados con dos filas de pines



Vista interior de una placa protoboard, donde aprecia la conexión interna entre los puntos externos.



CABLES / JUMPERS / DUPONT

Los cables jumpers, que son cables que tienen en sus terminales los jumpers que contienen sockets o pines, de ahí que se conozcan como cables jumper hembra o cables jumper macho.

Cabe mencionar que estos cables también son conocidos como cables dupont, por lo que ahora ya sabes cómo pedirlos cuando vayas a la tienda a adquirirlos.

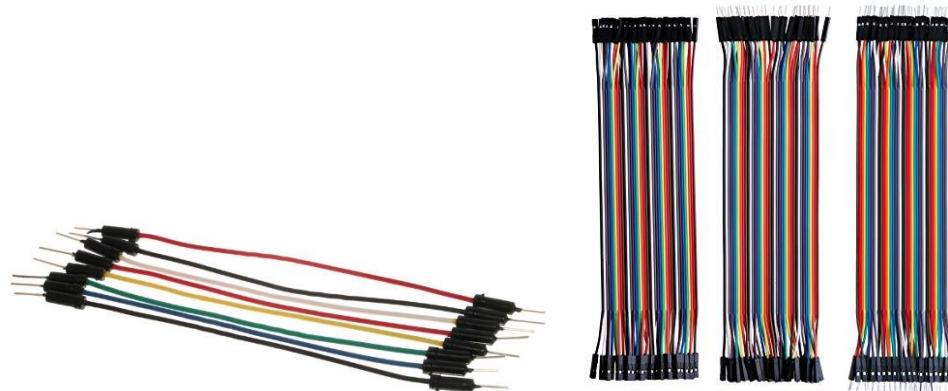
Un cable puente para prototipos (o simplemente puente para prototipos), es un cable con un conector en cada punta (o a veces sin ellos), que se usa normalmente para interconectar entre sí los componentes en una placa de pruebas. Por ejemplo, se utilizan de forma general para transferir señales eléctricas de cualquier parte de la placa de prototipos a los pines de entrada/salida de un microcontrolador.

Los cables puente se fijan mediante la inserción de sus extremos en los agujeros previstos a tal efecto en las ranuras de la placa de pruebas, la cual debajo de su superficie tiene unas planchas interiores paralelas que conectan las ranuras en grupos de filas o columnas según la zona. Los conectores se insertan en la placa de prototipos, sin necesidad de soldar, en los agujeros que convengan para el conexionado del diseño.

En el tipo con terminales aislados la disposición de los elementos y la facilidad de insertar los "conectores aislados" de los "cables puente" sobre la placa de pruebas permite el incremento de la densidad de montaje de ambos (componentes y puentes) sin temor a los cortocircuitos. Los cables puente varían en tamaño y color para distinguir las señales con las que se está trabajando.

Variación de cables puente con terminales esmaltados, según las combinaciones macho-hembra:

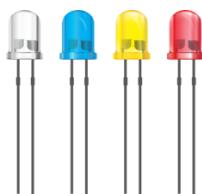
- Macho - macho
- Macho - hembra
- Hembra - hembra



Fuente de información:

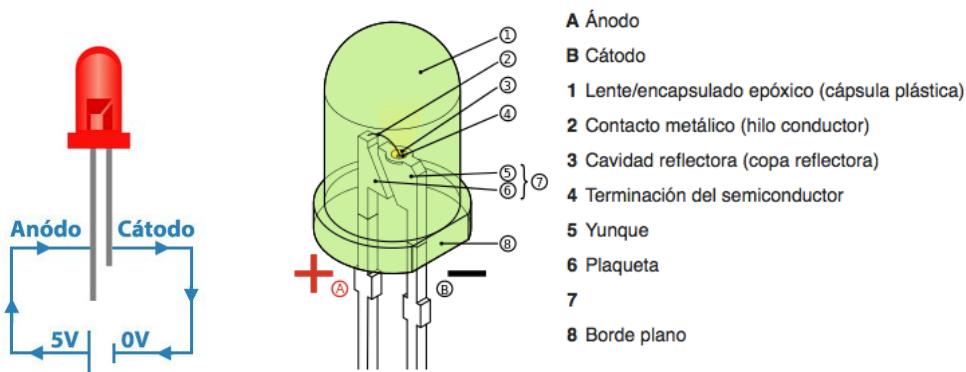
<https://tuelectronica.es/que-es-la-protoboard/>
<https://edutech.atlantistelecom.com/img2/2017/04/protoboard-post-255709.png>
https://es.wikipedia.org/wiki/Cable_puente
https://images-na.ssl-images-amazon.com/images/I/81yjq1pkiGL._SX425_.jpg

LED (DIODO EMISOR DE LUZ)



MARCO TEÓRICO

El LED, acrónimo de “Light Emitting Diode”, o diodo emisor de luz de estado sólido (solid state), constituye un tipo especial de semiconductor, cuya característica principal es convertir en luz la corriente eléctrica de bajo voltaje que atraviesa su chip. Desde el punto de vista físico un LED común se presenta como un bulbo miniaturizado, carente de filamento o de cualquier otro tipo de elemento o material peligroso, con la ventaja sobre otras tecnologías que no contamina el medio ambiente.



Fuente: http://www.asifunciona.com/fisica/ke_led/ke_led_2.htm

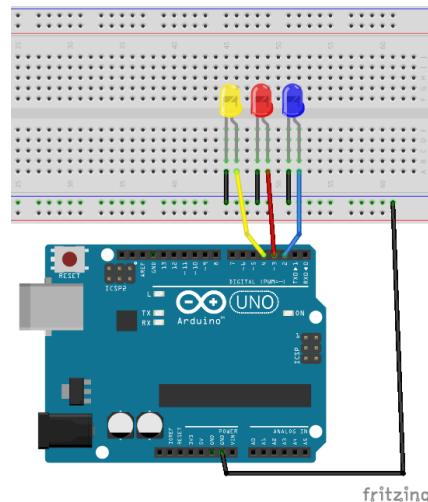
DESCRIPCIÓN DEL EJEMPLO

Para utilizar un PIN digital o analógico de Arduino se tiene que indicar mediante el comando **pinMode (PIN, OUTPUT)**, el pin a utilizar (numero) y si lo vas a utilizar con salida (OUTPUT) o entrada (INPUT). Se conectan 3 LED's de diferentes colores y se prueba el encendido y apagado, mediante la instrucción **digitalWrite (PIN, HIGH)**, indicando como primer parámetro el PIN de salida digital, y como segundo parámetro existe dos opciones LOW para salida 0V o apagado y HIGH para salida 5V o encendido.

MATERIALES A UTILIZAR

- Arduino UNO
- Cables
- Protoboard
- 3 LED

DIAGRAMA DE CONEXIÓN

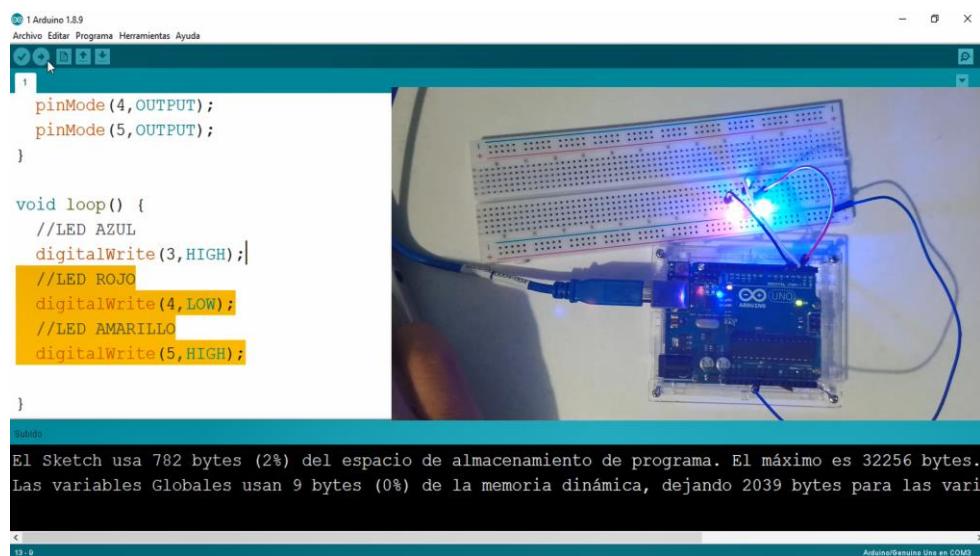


CÓDIGO FUENTE

```
//SE CONFIGURA COMO SALIDA EL PIN DIGITAL 3, 4 Y 5
void setup() {
    pinMode(3,OUTPUT);
    pinMode(4,OUTPUT);
    pinMode(5,OUTPUT);
}

void loop() {
    //LED AZUL
    digitalWrite(3,HIGH); //SE ESCRIBE HIGH SOBRE EL PIN 3
    //LED ROJO
    digitalWrite(4,LOW); //SE APAGA LED DEL PIN 4
    //LED AMARILLO
    digitalWrite(5,HIGH);
    delay(1000); // PAUSA DE 1 SEGUNDO
    digitalWrite(3,LOW);
    digitalWrite(4,HIGH);
    digitalWrite(5,LOW);
    delay(1000);
}
```

RESULTADO



ARDUINODROID (ARDUINO IDE DESDE EL CELULAR)



ArduinoDroid es una aplicación que se puede descargar de forma gratuita desde el Play Store de los dispositivos Android, con la cual se pueden escribir los programas e incluso programar algunas referencias de tarjetas Arduino desde un celular o una tablet que cuente con sistema operativo Android.



En cuanto a las tarjetas que soporta la aplicación se puede escoger entre una gran variedad de referencias desde la más utilizada la Arduino uno, incluyendo las nano, mega, pro, Leonardo entre otras, en lo personal he probado la aplicación con una tarjeta Arduino MEGA 2560 y ha funcionado muy bien tanto en la compilación del código como en la programación de la tarjeta desde la Tablet.



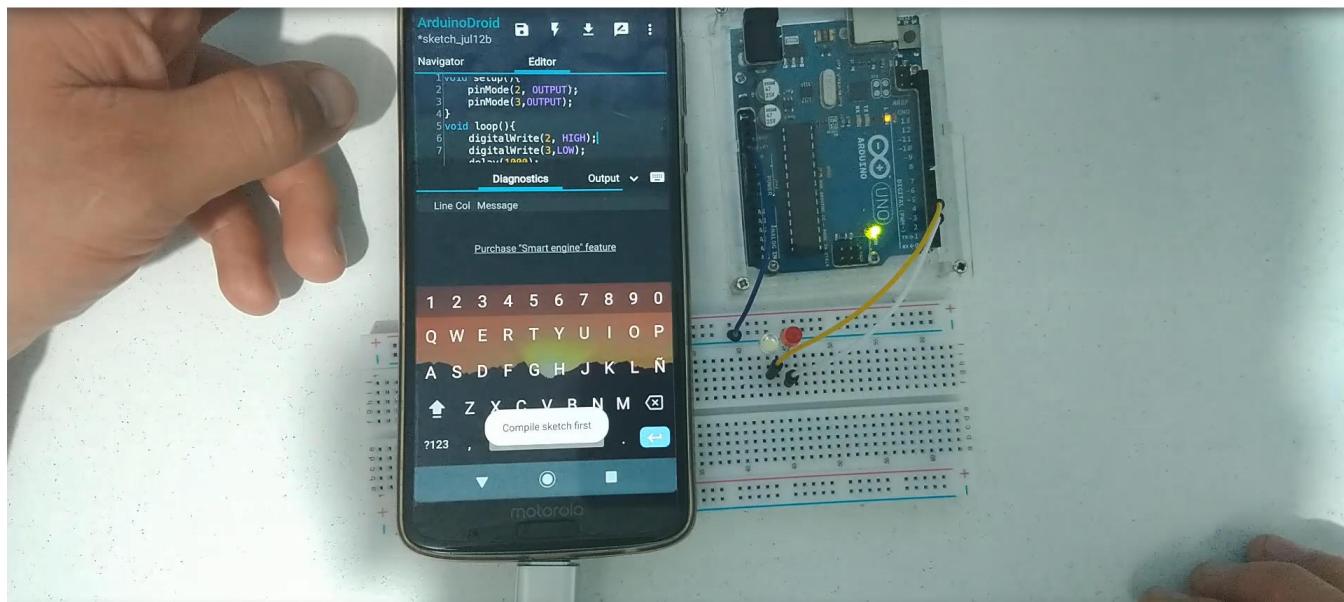
El lenguaje de programación utilizado por la aplicación es el mismo utilizado por el IDE oficial de Arduino, por lo tanto, no hay necesidad de aprender un nuevo lenguaje o nuevas instrucciones, es mas esta herramienta también nos permite abrir los Sketch escritos en el IDE de Arduino, compilarlos y programarlos sin ningún problema.

Fuente de información:

<https://geekelectronica.com/arduinodroid-una-aplicacion-para-programar-tarjetas-arduino-desde-dispositivos-android/>

RESULTADO

CURSO ARDUINO



ARDUINO DROID

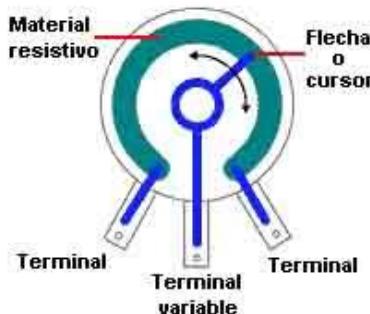
POTENCIÓMETRO



MARCO TEÓRICO

Un potenciómetro es un resistor eléctrico con un valor de resistencia variable y generalmente ajustable manualmente. Los potenciómetros utilizan tres terminales y se suelen utilizar en circuitos de poca corriente, para circuitos de mayor corriente se utilizan los reóstatos. En muchos dispositivos eléctricos los potenciómetros son los que establecen el nivel de salida. Por ejemplo, en un altavoz el potenciómetro ajusta el volumen; en un televisor o un monitor de ordenador se puede utilizar para controlar el brillo.

El valor de un potenciómetro viene expresado en ohmios (símbolo Ω) como las resistencias, y el valor del potenciómetro siempre es la resistencia máxima que puede llegar a tener. El mínimo lógicamente es cero. Por ejemplo, un potenciómetro de $10K\Omega$ puede tener una resistencia variable con valores entre 0Ω y 10.000Ω .



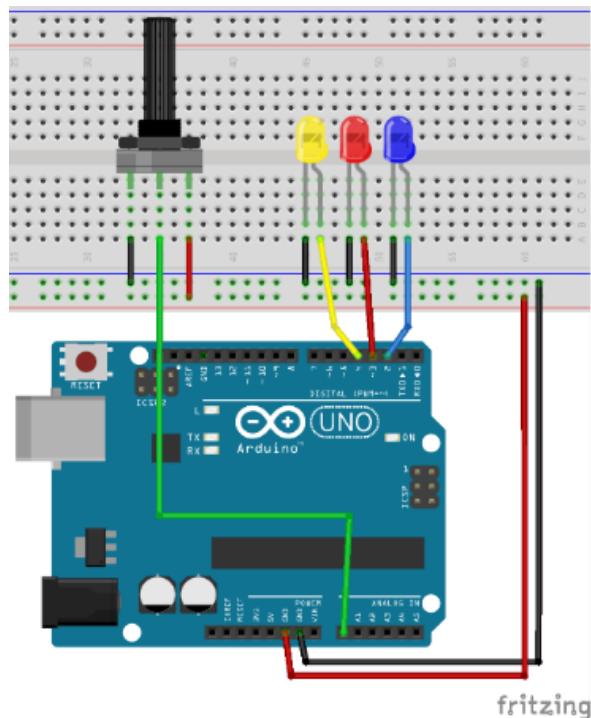
DESCRIPCIÓN DE EJEMPLO

Se conecta un potenciómetro al Arduino UNO para leer su valor, como una entrada analógica en el PIN A0, mediante la función `analogRead(A0)`, el valor de lectura a diferencia de una lectura digital que solo es 0 (LOW) o 1 (HIGH), en la lectura analógica de entrada se obtiene un rango de valores numéricos entre 0 y 1023. El funcionamiento requerido es que de acuerdo al valor numérico de la lectura analógica, se enciende 1, 2 o 3 LED's.

MATERIAL A UTILIZAR

- Potenciómetro 5 Kohm
- Cables
- Arduino UNO
- 3 LED

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

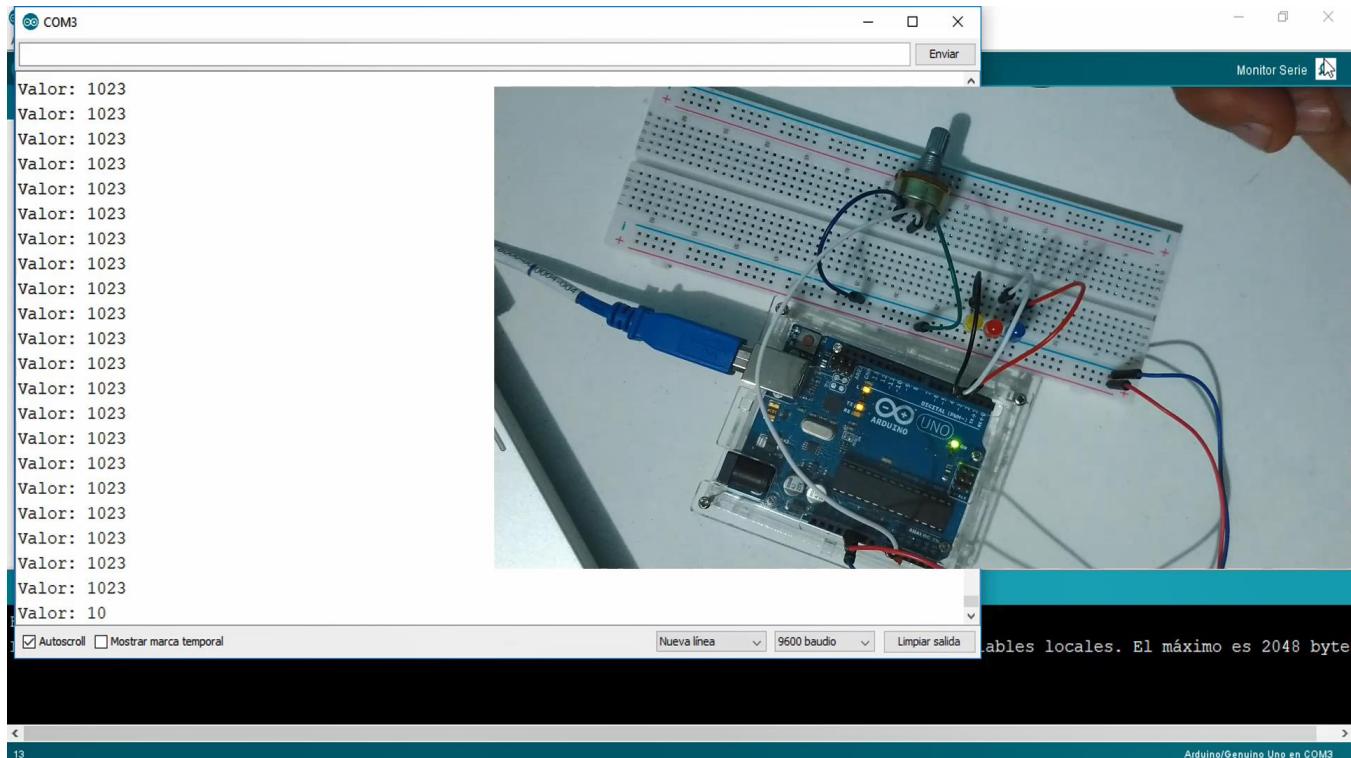
```

void setup() { //SE DECLARA PINES DE ENTRADA/SALIDA
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(A0, INPUT);
  Serial.begin(9600); //SE INICIALIZA MONITOR SERIE
}

void loop() {
  int x = analogRead(A0); //SE REALIZA LECTURA ANALÓGICA
  Serial.print("Valor: ");
  Serial.println(x); //SE IMPRIME EL VALOR
  delay(1);
  if (x < 300) { //SE COMPARA PARA PRENDER LEDS
    digitalWrite(3, HIGH);
  } else {
    digitalWrite(3, LOW);
  }
  if (x >= 300 && x < 700) {
    digitalWrite(4, HIGH);
  } else {
    digitalWrite(4, LOW);
  }
  if (x > 700) {
    digitalWrite(5, HIGH);
  } else {
    digitalWrite(5, LOW);
  }
}

```

RESULTADO



PUSH BUTTON



MARCO TEÓRICO

El Push Button es un interruptor de presión, botones de encendido/apagado, variaciones de “1” y “0” lógico de alguna señal electrónica. Si tu proyecto es muy importante y requieres máxima precisión en el voltaje emitido de alguna señal recomendamos utilizar otro tipo de interruptor debido a que los Push Button tienden a generar pequeños rebotes en dicha señal.

El interruptor (Switch) de presión (Push), de 12 Vcc, 50 mA, 4 terminales, normalmente abierto (NA). Se fabrica en plástico color negro, su vida útil es de 200,000 operaciones eléctricas y 100,000 mecánicas.

Este es un interruptor momentáneo de 4 terminales. Es más grande que el botón estándar para prácticas de electrónica y además es posible agregarle una capucha en la parte superior para personalizarlo. Se adapta perfectamente a las protoboard estándar. Este producto es compatible con Capuchón para Push Button Grande de 4 terminales.

- Tamaño 12 x 12 x 7.3 mm (L * W * T)
- Tamaño del cuadrado 4 x 4 mm
- Material: Plástico negro y metal.
- Corriente máxima: 50mA.

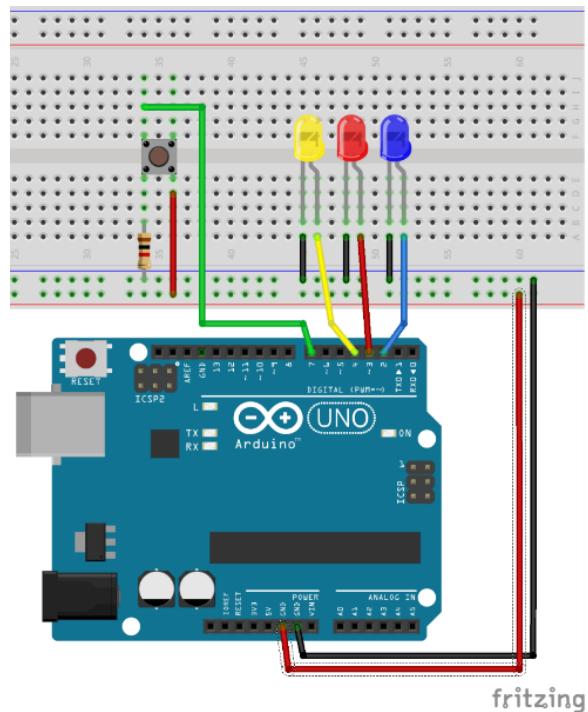
DESCRIPCIÓN DE EJEMPLO

Se realiza una práctica para leer el valor digital emitido por un Push Button, para indicar si se presionó o no, el funcionamiento de la práctica se inicia con la lectura del valor entrante del Push en el PIN digital 7, mediante la función digitalRead(7), donde devuelve 1, si se presionó el Push y 0 si no está presionado, de esta forma cual el Push se presiona activa una secuencia de encendido y apagado de 3 LED de diferentes colores.

MATERIAL A UTILIZAR

- 1 Push Button
- 3 LED
- Arduino UNO
- Protoboard
- 1 resistencia 220 ohm
- Cables

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```

void setup() // SE INCIALIZA PINES DE ENTRADA/SALIDA
{
    pinMode(3, OUTPUT);
    pinMode(4, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(7, INPUT);
    Serial.begin(9600);
}

void loop() {
    int valor = digitalRead(7); //SE REALIZAR LECTURA DIGITAL PIN 7
    Serial.print("Push: ");
    Serial.println(valor); //SE IMPRIME VALOR DE LECTURA
    delay(1);
    if (valor == 1) { //SI EL VALOR ES 1 REALIZA SECUENCIA CON LEDS
        for (int i = 0; i < 2; i++) {
            digitalWrite(3, HIGH);
            digitalWrite(4, LOW);
            digitalWrite(5, HIGH);
            delay(1000);
            digitalWrite(3, LOW);
            digitalWrite(4, HIGH);
            digitalWrite(5, LOW);
            delay(1000);
            digitalWrite(3, LOW);
            digitalWrite(4, LOW);
            digitalWrite(5, LOW);
            delay(1000);
        }
    }
}

```

RESULTADO



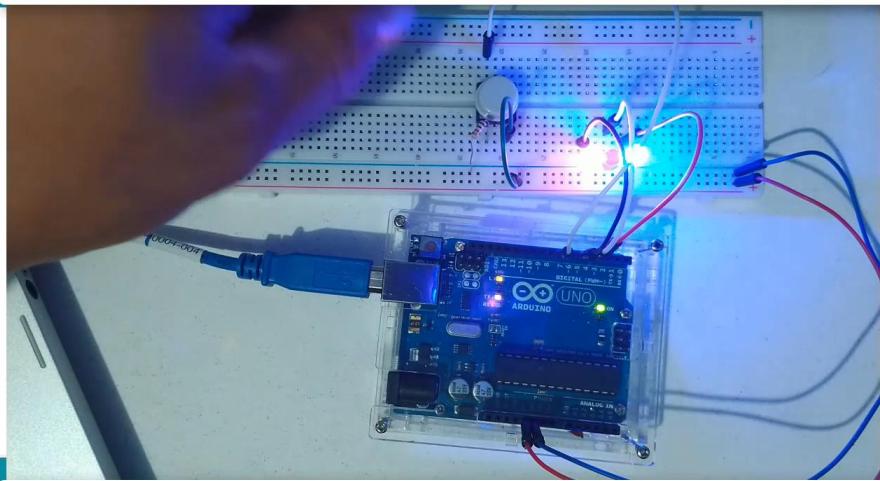
The screenshot shows the Arduino IDE interface. At the top, it says "1 Arduino 1.8.9" and has menu options: Archivo, Editar, Programa, Herramientas, Ayuda. Below the menu is a toolbar with icons for upload, save, and other functions. The main area contains the following sketch code:

```
1
delay(1);
if(valor==1){
  for(int i=0;i<2;i++){
    digitalWrite(3,HIGH);
    digitalWrite(4,LOW);
    digitalWrite(5,HIGH);
    delay(500);
    digitalWrite(3,LOW);
    digitalWrite(4,HIGH);
    digitalWrite(5,LOW);
    delay(500);
    digitalWrite(3,LOW);
    digitalWrite(4,LOW);
    digitalWrite(5,LOW);
  }
}

Subido
El Sketch usa 2476 bytes (7%) del espacio de al
Las variables Globales usan 194 bytes (9%) de l
28 - 16
```

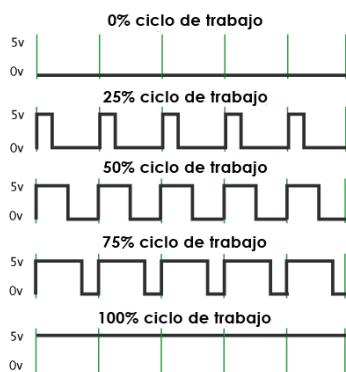
The code uses digital pins 3, 4, and 5 to control three LEDs connected to a breadboard. It toggles the state of these pins every 500ms. The status bar at the bottom indicates the sketch uses 2476 bytes (7%) of memory and 194 bytes (9%) of global variables.

CURSO ARDUINO



DIGITALREAD / PUSH BUTTON

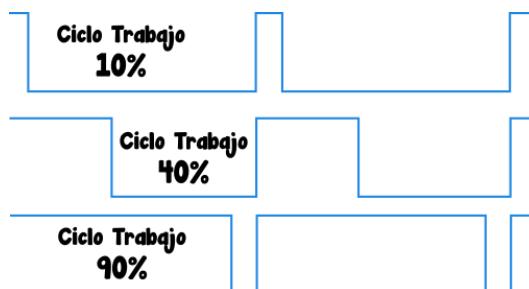
PWM (MODULACIÓN POR ANCHO DE PULSOS)



MARCO TEÓRICO

La modulación por ancho o de pulso (o en inglés Pulse Width Modulation PWM) es un tipo de señal de voltaje utilizada para enviar información o para modificar la cantidad de energía que se envía a una carga. Este tipo de señal es muy utilizada en circuitos digitales que necesitan emular una señal analógica.

Este tipo de señales son de tipo cuadrada o sinusoidales en las cuales se le cambia el ancho relativo respecto al período de la misma, el resultado de este cambio es llamado ciclo de trabajo y sus unidades están representadas en términos de porcentaje.



Para emular una señal analógica se cambia el ciclo de trabajo de tal manera que el valor promedio de la señal sea el voltaje aproximado que se desea obtener, pudiendo entonces enviar voltajes entre 0[V] y el máximo que soporte el dispositivo PWM utilizado, en el caso de Arduino es 5[V].

En Arduino este tipo de señales sólo puede ser realizado con los pines que tienen el símbolo ~ en sus números. En Arduino UNO son los pines 3, 5, 6, 9, 10 y 11.

La señal en Arduino tiene valores de 0[V] a 5[V] y una frecuencia de aproximadamente 500[Hz]. En los pines 5 y 6 esta frecuencia es aproximadamente el doble.

Las aplicaciones típicas para este tipo de señales son: Controlar intensidad de luz de un LED, mover servomotores, controlar LED RGB, controlar velocidad de motores de corriente continua y controlar motores eléctricos de inducción o asincrónicos.

Fuentes de información
<http://www.arduino.utfsm.cl/modulacion-por-ancho-de-pulso-pwm/>

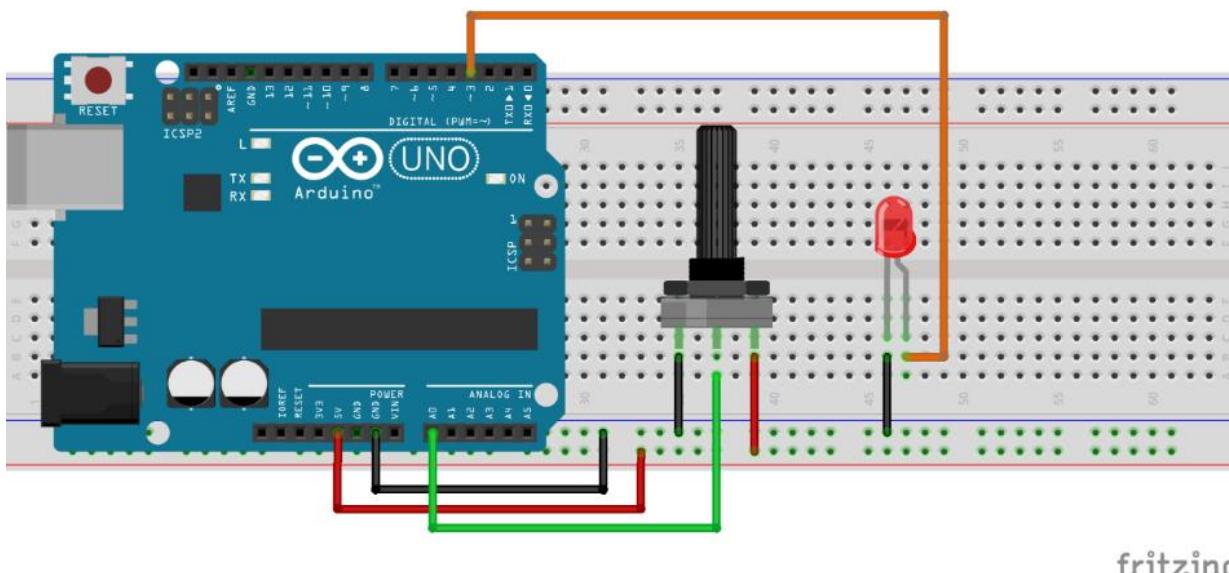
DESCRIPCIÓN DE EJEMPLO

En este ejemplo se realiza la escritura o salida analógica mediante un pin digital utilizando la función **analogWrite (PIN, VALOR)**, en donde tiene como parámetros el número de PIN **PWM** (con símbolo ~) del Arduino, y como segundo parámetro el valor que esta Mapeado (función **map**) a valores entre 0 a 255, a referencia de los valores de lectura analógica del potenciómetro que son de 0 a 1023. La idea se centra en que a medida que el potenciómetro cambie el valor de su resistencia, esto modifica la intensidad del brillo del LED.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Potenciómetro
- Un LED

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

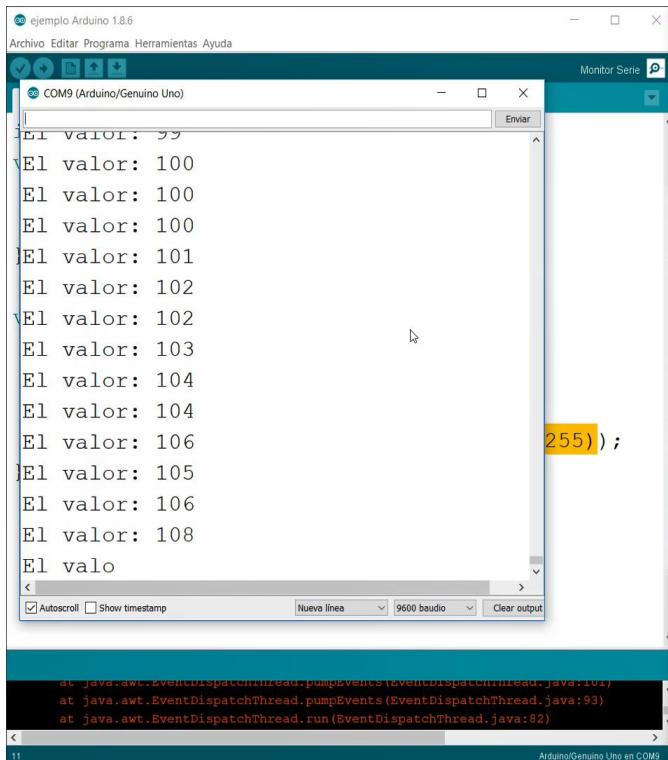
```

int valor;//DE DECLARA VARIABLE
void setup(){//SE INICIALIZA PINES EN MODO SALIDA
  pinMode(3, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  valor = analogRead(A0);//SE REALIZA LECTURA ANALÓGICA
  Serial.print("El valor: ");
  Serial.println(valor);// SE IMPRIME VALOR EN MONITOR SERIE
  //SE MAPEA VALOR 0-1023 A 0-255 SOBRE PIN 3 PWM
  analogWrite(3, map(valor, 0, 1023, 0, 255));
}

```

RESULTADO

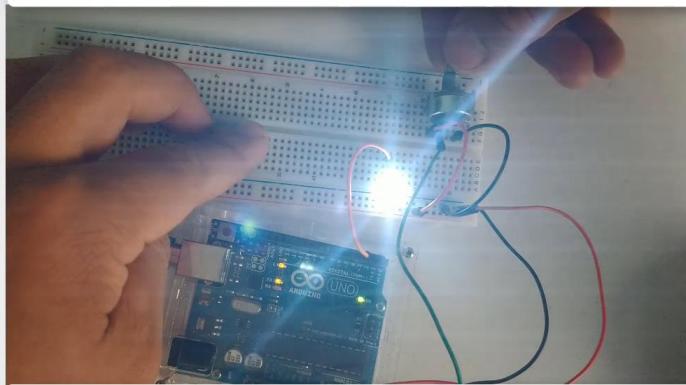


The screenshot shows the Arduino Serial Monitor window. The title bar says "ejemplo Arduino 1.8.6". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". The main area displays the following text:
El valor: 22
El valor: 100
El valor: 100
El valor: 100
El valor: 101
El valor: 102
El valor: 102
El valor: 103
El valor: 104
El valor: 104
El valor: 105
El valor: 106
El valor: 105
El valor: 106
El valor: 108
El valor:
The bottom status bar shows "Autoscroll" checked, "Show timestamp" unchecked, "Nueva linea" dropdown set to "Nueva linea", "9600 baudio" dropdown set to "9600", and "Clear output" button.

```
at java.awt.EventQueue.dispatchEvent(EventDispatchThread.java:101)
at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:93)
at java.awt.EventDispatchThread.run(EventDispatchThread.java:82)
```

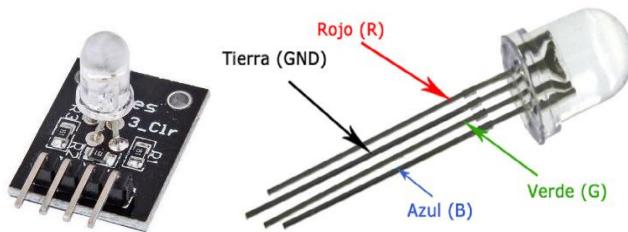
11 Arduino/Genuino Uno en COM9

CURSO ARDUINO



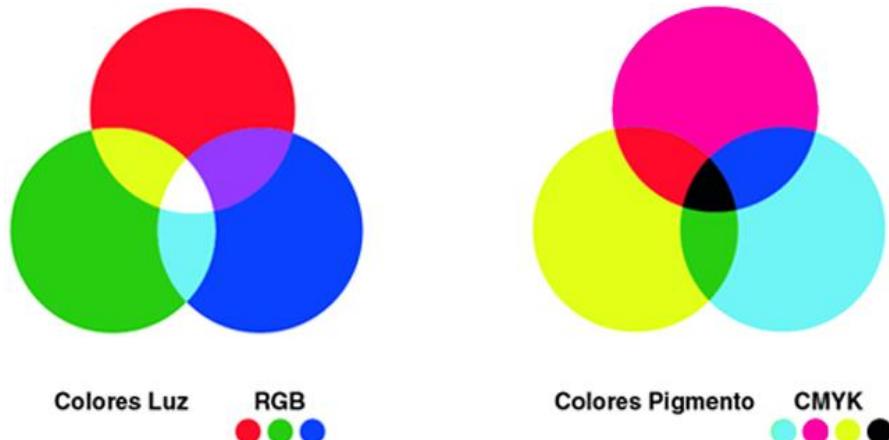
PWM
ANALOGWRITE

LED RGB (RED-GREEN-BLUE)

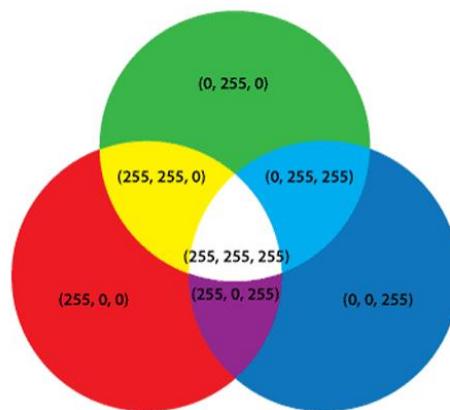


MARCO TEÓRICO

LED RGB (Red, Green, Blue) significa LED rojo, azul y verde. Este tipo LED RGB combinan estos tres colores para producir más de 16 millones de tonos de luz. Pero no todos los colores son posibles. Algunos se encuentran "fuera" del triángulo formado por los LED RGB. Además, los colores pigmento, como el marrón o el rosa, son difíciles o imposibles de lograr.



Los colores del Led RGB vienen representados con números comprendidos entre el valor 0 y el valor 255. De esta forma, para componer el color rojo pondríamos el valor máximo del rojo y el valor mínimo de los otros colores, es decir, el rojo equivale a "R=255; G=0; B=0". Y así sucesivamente con el resto de colores.



Fuente de información

<http://www.lighting.philips.com.mx/soporte/soporte/preguntas-frecuentes/white-light-and-colour/what-does-rgb-led-mean>
<https://www.programoergosum.com/cursos-online/robotica-educativa/251-led-rgb-del-robot-mbot/que-es-un-led-rgb>

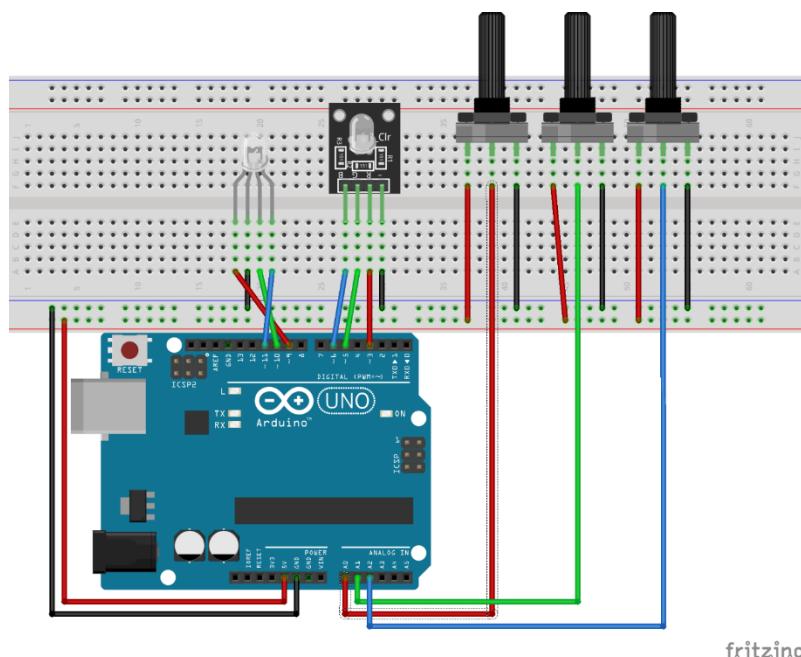
DESCRIPCIÓN DE EJEMPLO

Para este ejemplo se conectan 3 potenciómetros para el control de la intensidad de brillo del color Rojo, Verde y Azul respectivamente y se realiza la lectura del valor mediante pin analógico y la función **analogRead**, que devuelve valores comprendidos entre 0 a 1023, después de obtener el valor se mapea el valor entre 0 a 255 para escribir mediante la función **analogWrite**, sobre pines PWM 3, 5, 6, 9, 10 y 11 para controlar la intensidad de brillo de cada LED, con el objetivo de poder apreciar la combinación de colores entre rojo, verde y azul, la teoría nos dice que hay 16 millones de colores.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- LED RGB sin PCB
- LED RGB con PCB
- 3 potenciómetro de 5 Kohm

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```
void setup() {
  //LED RGB PCB
  pinMode(3,OUTPUT);//rojo
  pinMode(5,OUTPUT);//verde
  pinMode(6,OUTPUT);//azul
  //LED RGB
  pinMode(9,OUTPUT);//rojo
  pinMode(10,OUTPUT);//verde
  pinMode(11,OUTPUT);//azul
  //POTENCIOMETROS
  pinMode(A0,INPUT);//rojo
  pinMode(A1,INPUT);//verde
```

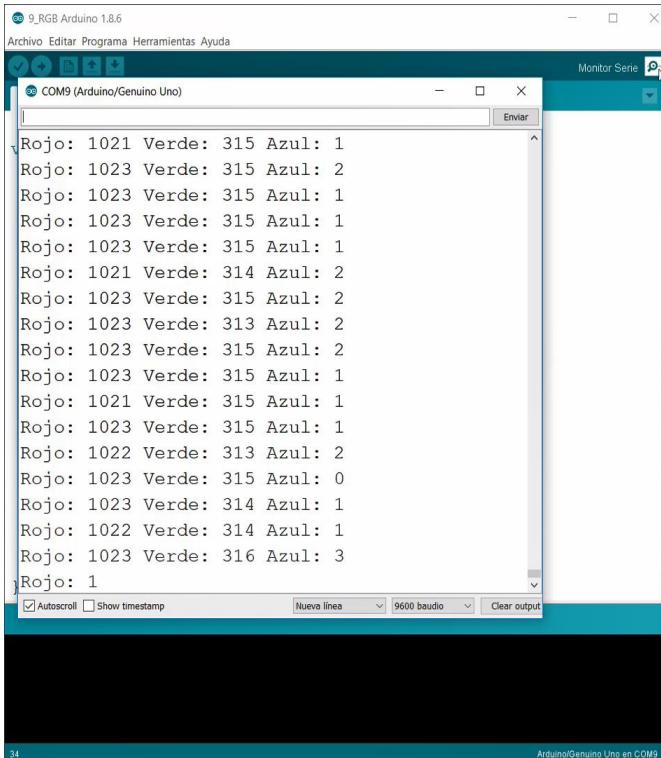
```

pinMode(A2, INPUT); //azul
Serial.begin(9600);
}

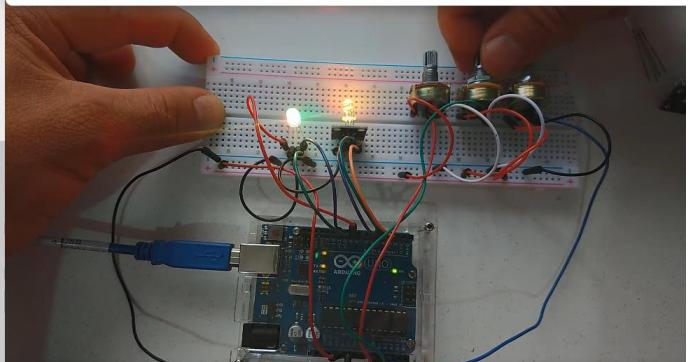
void loop() {
  int r=analogRead(A0); //LECTURA POTENCIOMETRO1
  int v=analogRead(A1); //LECTURA POTENCIOMETRO2
  int a=analogRead(A2); //LECTURA POTENCIOMETRO3
  Serial.print("Rojo: "); //IMPRIME VALORES
  Serial.print(r);
  Serial.print(" Verde: ");
  Serial.print(v);
  Serial.print(" Azul: ");
  Serial.println(a);
  //MAPEA VALORES SOBRE LOS PINES DIGITALES PWM
  analogWrite(3, map(r, 0, 1023, 0, 255));
  analogWrite(5, map(v, 0, 1023, 0, 255));
  analogWrite(6, map(a, 0, 1023, 0, 255));
  analogWrite(9, map(r, 0, 1023, 0, 255));
  analogWrite(10, map(v, 0, 1023, 0, 255));
  analogWrite(11, map(a, 0, 1023, 0, 255));
  delay(10);
}

```

RESULTADO

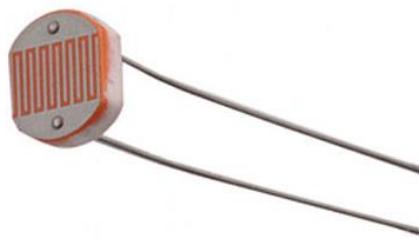


CURSO ARDUINO



LED RGB

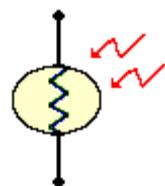
LDR (FOTORRESISTENCIA)



MARCO TEÓRICO

LDR (Light-Dependent Resistor, resistor dependiente de la luz)

Un LDR es un resistor que varía su valor de resistencia eléctrica dependiendo de la cantidad de luz que incide sobre él. Se le llama, también, fotorresistor o fotorresistencia. El valor de resistencia eléctrica de un LDR es bajo cuando hay luz incidiendo en él (en algunos casos puede descender a tan bajo como 50 ohm) y muy alto cuando está a oscuras (puede ser de varios Mohm).



El LDR es fabricado con materiales de estructura cristalina, y utiliza sus propiedades fotoconductoras. Los cristales utilizados más comunes son: sulfuro de cadmio y seleniuro de cadmio. El valor de la fotorresistencia (en Ohmios) no varía de forma instantánea cuando se pasa de luz a oscuridad o, al contrario, y el tiempo que se dura en este proceso no siempre es igual si se pasa de oscuro a iluminado o si se pasa de iluminado a oscuro.



Esto hace que el LDR no se pueda utilizar en muchas aplicaciones, especialmente aquellas que necesitan de mucha exactitud en cuanto a tiempo para cambiar de estado (oscuridad a iluminación o iluminación a oscuridad) y a exactitud de los valores de la fotorresistencia al estar en los mismos estados anteriores. Su tiempo de respuesta típico es de aproximadamente 0.1 segundos.

Pero hay muchas aplicaciones en las que una fotorresistencia es muy útil. En casos en que la exactitud de los cambios no es importante como en los circuitos:

- Luz nocturna de encendido automático con 555 y relé, que utiliza una fotorresistencia para activar una o más luces al llegar la noche.
 - Relé controlado por luz, donde el estado de iluminación de la fotorresistencia, activa o desactiva un Relay (relé), que puede tener un gran número de aplicaciones

El LDR o fotoresistencia es un elemento muy útil para aplicaciones en circuitos donde se necesita detectar la ausencia de luz de día.

Fuente de información

http://robots-argentina.com.ar/Sensores_LDR.htm
<https://unicrom.com/ldr-fotorresistencia-fotorresistor/>

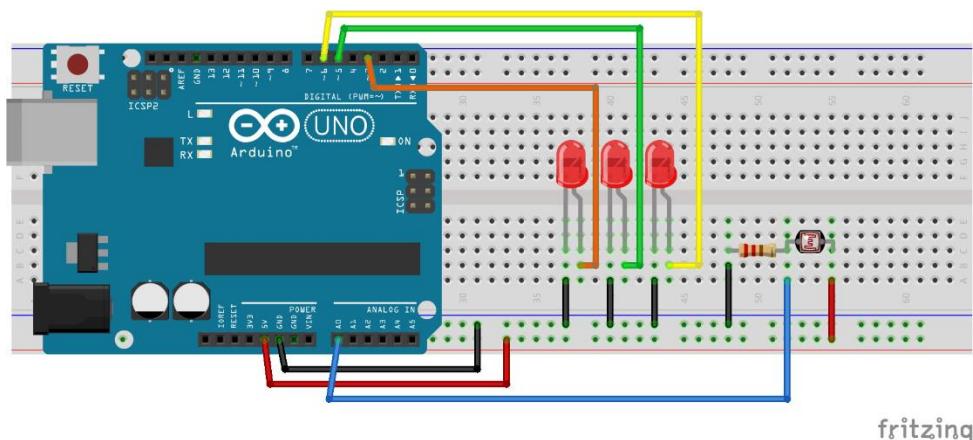
DESCRIPCIÓN DE EJEMPLO

Se conectan 3 LED a pines **PWM** para que al momento de leer el valor del sensor **LDR** mediante pin analógico A0, obtenemos valores entre 0 y 1023, estos valores se mapean para que al momento de detectar luminosidad aumente o disminuya la intensidad de brillo de los 3 LED de forma paralela, mediante la función **analogWrite** de **PWM**.

MATERIAL A UTILIZAR

- Arduino UNO
 - Protoboard
 - Cables
 - 3 LED
 - 1 resistencia 220 ohm
 - 1 LDR (Fotorresistencia)

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

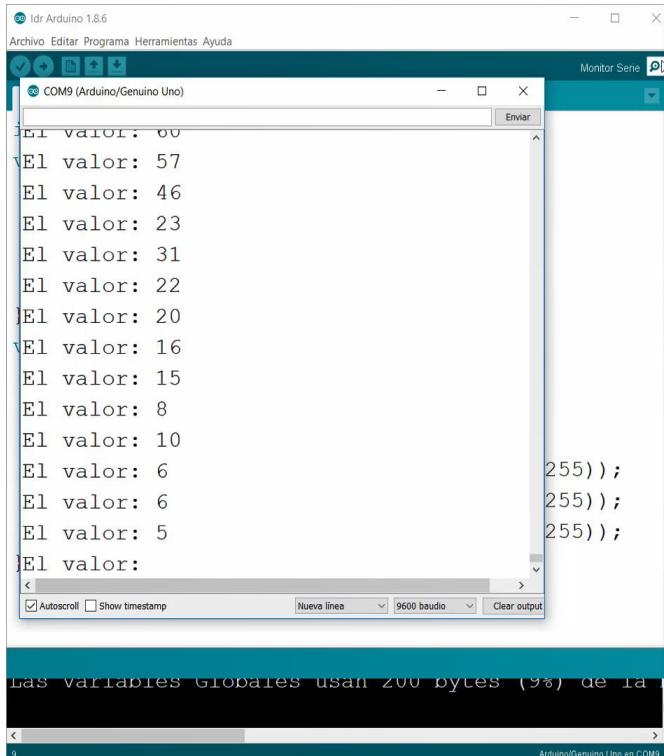
```
int valor;//SE DECLARA VARIABLE  
//SE INICIALIZA PINES COMO SALIDA  
void setup(){  
    pinMode(3, OUTPUT);  
    pinMode(5, OUTPUT);
```

```

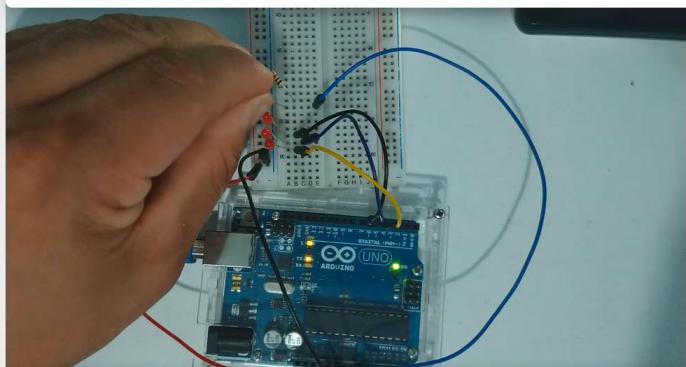
pinMode(6, OUTPUT);
Serial.begin(9600);
}
void loop(){
    valor = analogRead(A0); //SE LEE VALOR DE LDR VIA PIN ANALÓGICO
    Serial.print("El valor: "); //SE IMPRIME VALOR DE LDR EN MONITOR SERIE
    Serial.println(valor);
    //SE MAPAEA EL VALOR SOBRE 3 LEDS CONECTADOS A PINES PWM
    analogWrite(3, map(valor, 0, 1023, 0, 255));
    analogWrite(5, map(valor, 0, 1023, 0, 255));
    analogWrite(6, map(valor, 0, 1023, 0, 255));
}

```

RESULTADO



CURSO ARDUINO



LDR
FOTORESISTENCIA

LCD 16X02 CON I2C (DISPLAY DE CRISTAL LIQUIDO)

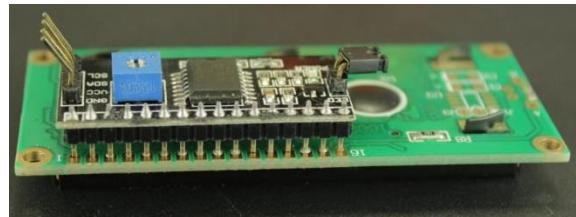


MARCO TEÓRICO

LCD viene del inglés Liquid Crystal Display, o sea Pantalla de cristal líquido.

La pantalla LCD de 16x2 basada en el controlador HD44780 de Hitachi es un periférico muy común, que se utiliza ampliamente en proyectos con Arduino, sin embargo, es bien sabido por todo aquel entusiasta que ha incluido una en sus proyectos, que este tipo de pantalla requiere muchos pines del microcontrolador para ser controlada, debido principalmente a que utiliza un bus paralelo para comunicarse. Afortunadamente existe una solución muy fácil y económica para este problema: un adaptador basado en el PCF8574 que permite conectar la pantalla al Arduino usando solamente dos líneas digitales a través del bus **I2C**. Dichos pines, pueden además ser compartidos por otros periféricos como el RTC.

A continuación, se muestra el LCD con el circuito I2C para comunicar la pantalla con el Arduino mediante dos pines SDA y SCL.



El módulo PCF8574A permite expandir las entradas o salidas digitales de Arduino utilizando solo 2 líneas del bus I2C (SDA y SCL). El chip PCF8574 logra convertir datos en paralelo (8 E/S) a I2C y viceversa, por lo que es ideal para el manejo de dispositivos como: displays LCD alfanuméricos, teclados matriciales, LEDs, relays y más. Puede compartir el bus I2C con otros dispositivos y así ahorrar pines, por ejemplo: RTC, memoria, sensores. Se pueden manejar hasta 8 expansores PCF8574 en un mismo bus I2C y de esa forma manejar un total de 64 E/S utilizando tan solo 2 pines.

El PCF8574 es un expander de E/S compatible con la mayoría de microcontroladores, permite una comunicación bidireccional utilizando solo dos líneas a través del bus I2C.

Partes de Controlador I2C



Fuente de Información:

<https://www.prometec.net/displays-lcd/>

<https://www.geekfactory.mx/tutoriales/tutoriales-arduino/lcd-16x2-por-i2c-con-arduino/>

DESCRIPCIÓN DE EJEMPLO

Este es un ejemplo básico sobre el funcionamiento del **LCD** con controlador **I2C**, donde solo se conectan 2 pines **SDA** y **SCL** al Arduino y todo el control se realiza mediante la librería **LiquidCrystal_I2C.h** y la librería **Wire.h**, donde se crear un objeto tipo **LiquidCrystal_I2C** que tiene como parámetros las filas y columnas, Se utilizar **setCursor(Fila , Columna)** y la función **print** para imprimir el mensaje.

Librería

Es importante agregar la librería correcta y la versión para el control por I2C, aquí hay 3 opciones para descargar:

https://github.com/pkourany/LiquidCrystal_V1.2.1

<https://playground.arduino.cc/Code/LCDI2c/>

<https://www.arduinolibraries.info/libraries/liquid-crystal-i2-c>

Librería LiquidCrystal_I2C para Arduino

Existen diferentes tipos y versiones de librerías para trabajar con el módulo Adaptador LCD a I2C.

Las funciones que utiliza esta librería son similares a la librería Liquid Crystal de Arduino, revisaremos las funciones principales:

- **LiquidCrystal_I2C (lcd_Addr, lcd_cols, lcd_rows)** Función constructora, crea un objeto de la clase LiquidCrystal_I2C, con dirección, columnas y filas indicadas.
- **init()** Inicializa el módulo adaptador LCD a I2C, esta función internamente configura e inicializa el I2C y el LCD.
- **clear()** Borra la pantalla LCD y posiciona el cursor en la esquina superior izquierda (posición (0,0)).
- **setCursor(col, row)** Posiciona el cursor del LCD en la posición indicada por col y row(x,y); es decir, establecer la ubicación en la que se mostrará posteriormente texto escrito para la pantalla LCD.
- **print()** Escribe un texto o mensaje en el LCD, su uso es similar a un Serial.print

- **scrollDisplayLeft()** Se desplaza el contenido de la pantalla (texto y el cursor) un espacio hacia la izquierda.
 - **scrollDisplayRight()** Se desplaza el contenido de la pantalla (texto y el cursor) un espacio a la derecha.
 - **backlight()** Enciende la Luz del Fondo del LCD
 - **noBacklight()** Apaga la Luz del Fondo del LCD
 - **createChar (num, datos)** Crea un carácter personalizado para su uso en la pantalla LCD. Se admiten hasta ocho caracteres de 5x8 píxeles (numeradas del 0 al 7). Dónde: num es el número de carácter y datos es una matriz que contienen los píxeles del carácter. Se verá un ejemplo de esto más adelante.

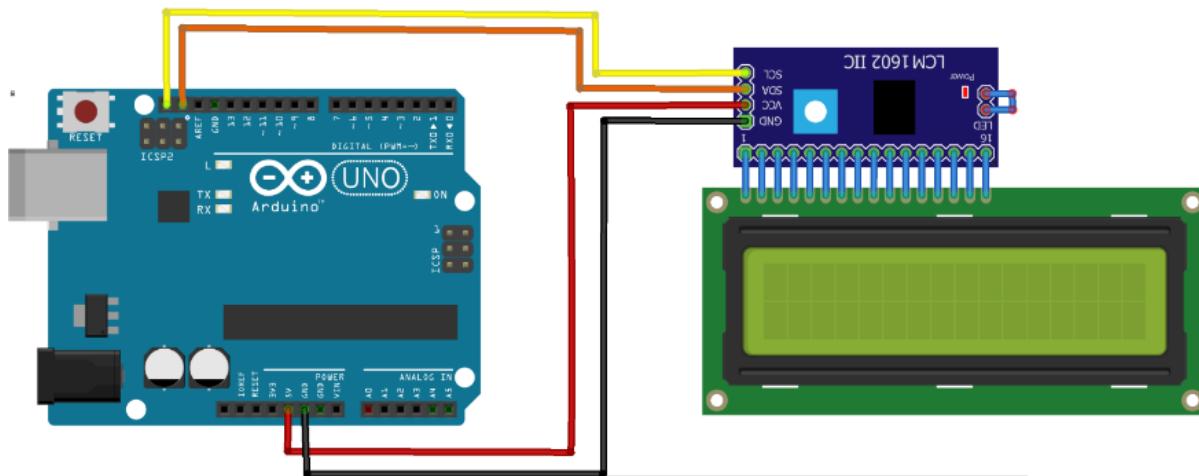
Referencia de información:

<https://www.geekfactory.mx/tutoriales/tutoriales-arduino/lcd-16x2-por-i2c-con-arduino/>

MATERIAL A UTILIZAR

- Arduino UNO
 - Cables
 - Pantalla LCD 16x02
 - Circuito Integrado I2C

DIAGRAMA DE CONEXIÓN



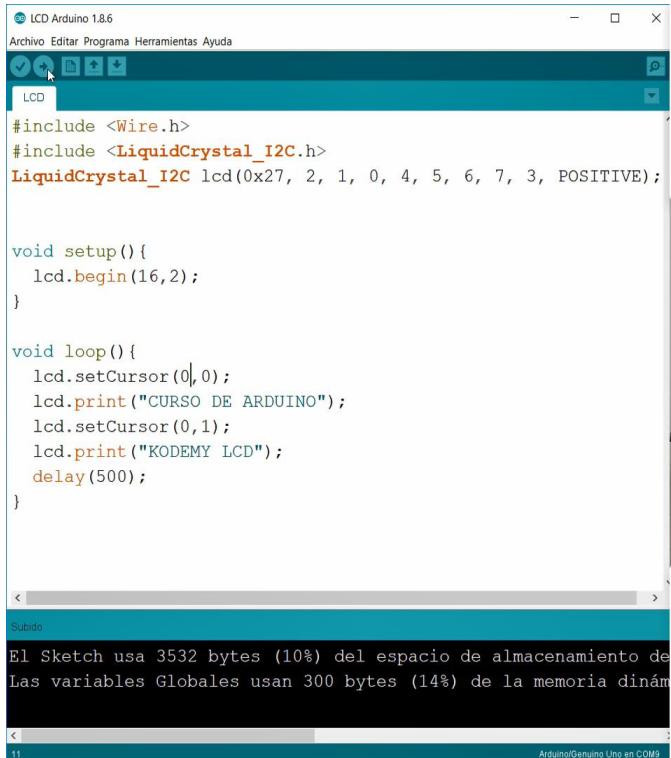
CÓDIGO FUENTE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

void setup() {
    lcd.begin(16,2);
}

void loop() {
    lcd.setCursor(0,0);
    lcd.print("CURSO DE ARDUINO");
    lcd.setCursor(0,1);
    lcd.print("KODEMY LCD");
    delay(500);
}
```

RESULTADO



```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

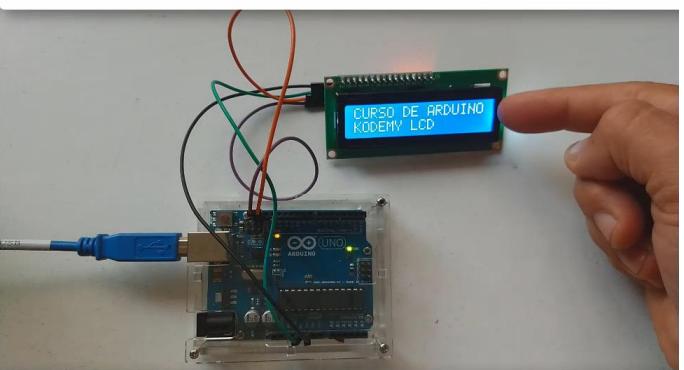
void setup(){
  lcd.begin(16,2);
}

void loop(){
  lcd.setCursor(0,0);
  lcd.print("CURSO DE ARDUINO");
  lcd.setCursor(0,1);
  lcd.print("KODEMY LCD");
  delay(500);
}

```

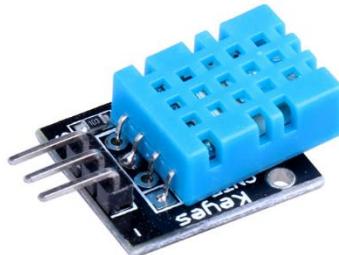
Subido
El Sketch usa 3532 bytes (10%) del espacio de almacenamiento de
Las variables Globales usan 300 bytes (14%) de la memoria dinámica

CURSO ARDUINO



LCD 16X02 I2C

SENSOR DHT11



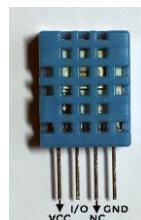
MARCO TEÓRICO

DHT11 un único sensor para la temperatura y humedad

El DHT11 presume de ser un sensor con una alta fiabilidad y estabilidad debido a su señal digital calibrada. Lo podemos comprar de dos maneras, de forma individual donde solo tenemos el sensor DHT11, o insertado en una PCB.

La diferencia en precio no es excesiva y la versión con PCB aporta una resistencia pull-up de 5 kΩ y un LED que nos avisa de su funcionamiento. Otra diferencia entre estas dos versiones del DHT11 son los pines.

En la versión sin PCB tenemos 4 pines y en la versión con PCB tenemos 3 pines.



Los pines de la versión sin PCB del DHT11 son:

- VCC: alimentación
- I/O: transmisión de datos
- NC: no conecta, pin al aire
- GND: conexión a tierra



Los pines de la versión con PCB del DHT11 son:

- GND: conexión con tierra
- DATA: transmisión de datos
- VCC: alimentación

¿Cómo transmite los datos el DHT11?

No tenemos que confundirnos entre analógico y digital. Aunque lo conectemos a un pin digital, se trata de un dispositivo analógico. Dentro del propio dispositivo se hace la conversión entre analógico y digital.

Por lo tanto, partimos de una señal analógica que luego es convertida en formato digital y se enviará al microcontrolador. La trama de datos es de 40 bits correspondiente a la información de humedad y temperatura del DHT11.

<u>0011 0101</u>	<u>0000 0000</u>	<u>0001 1000</u>	<u>0000 0000</u>	<u>0100 1001</u>
8 bits humedad	8 bits humedad	8 bits temperatura	8 bits temperatura	bits de paridad

El primer grupo de 8-bit es la parte entera de la humedad y el segundo grupo la parte decimal. Lo mismo ocurre con el tercer y cuarto grupo, la parte entera de la temperatura y la parte decimal. Por último, los bits de paridad para confirmar que no hay datos corruptos.

Estos bits de paridad lo único que hacen es asegurarnos de que la información es correcta, sumando los 4 primeros grupos de 8-bit. Esta suma debe ser igual a los bits de paridad. Si nos centramos en la imagen anterior y sumamos los bits, comprobamos que todo está correcto.

$$\textbf{0011 0101 + 0000 0000 + 0001 1000 + 0000 0000 = 0100 1101}$$

Fuente de información: <https://programarfácil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/>

DESCRIPCIÓN DE EJEMPLO

Se conecta el sensor DHT11 al pin digital 2, para enviar su trama de datos y visualizar los datos de temperatura y humedad en la pantalla LCD, es muy importante agregar la librería correspondiente **DHT.h**, que nos permite crear un objeto tipo **DHT_Unified**, que tiene como parámetros el PIN 2 del Arduino al que se conecta y el tipo de sensor que es DHT11.

La temperatura se obtiene mediante **dht.temperature().getEvent(&event)**; y se imprime mediante **event.temperature**, de esta misma forma se obtiene la humedad **dht.humidity().getEvent(&event)**; y se imprime en monitor serie y en LCD con **event.relative_humidity**.

LIBRERÍA

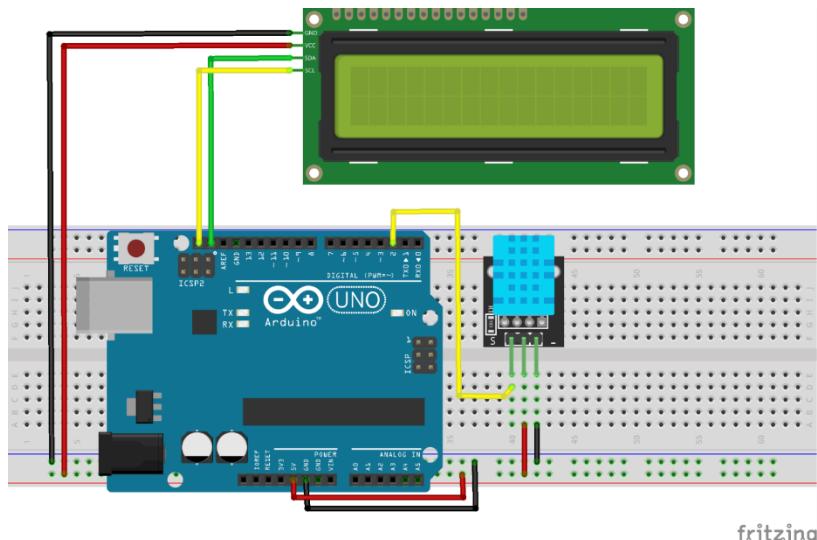
Adafruit Unified Sensor by Adafruit, que se encuentra dentro del gestor de librerías

Las variables, funciones u objetos cambian de acuerdo a la librería que se agrega

MATERIAL A UTILIZAR

- Arduino UNO
 - Protoboard
 - Cables
 - Sensor DHT11
 - Pantalla LCD 16x02 con I2C

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

```
//SE AGREGA LIBRERIAS
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
//SE DEFINE VARIABLES U OBJETOS A UTILIZAR
#define DHTPIN 2
#define DHTTYPE DHT11
DHT_Unified dht(DHTPIN, DHTTYPE);
uint32_t delayMS;
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

void setup(){ //SE INICIALIZAR PANTALLA LCD Y SENSOR SHT11
lcd.begin(16,2);
dht.begin();
Serial.begin(9600);
}

void loop(){
delay(delayMS);
sensors_event_t event;
//SE REALIZAR LECTURA DE LA TEMPERATURA Y SE COMPARA
dht.temperature().getEvent(&event);
if (isnan(event.temperature)) {
  Serial.println(F("Error al leer Temperatura"));
}
else {
  Serial.print(F("Temperatura: "));
  Serial.print(event.temperature);
  Serial.println(F(" C"));
  lcd.setCursor(0,0);
  lcd.print("TEMPERATURA ");
}
```

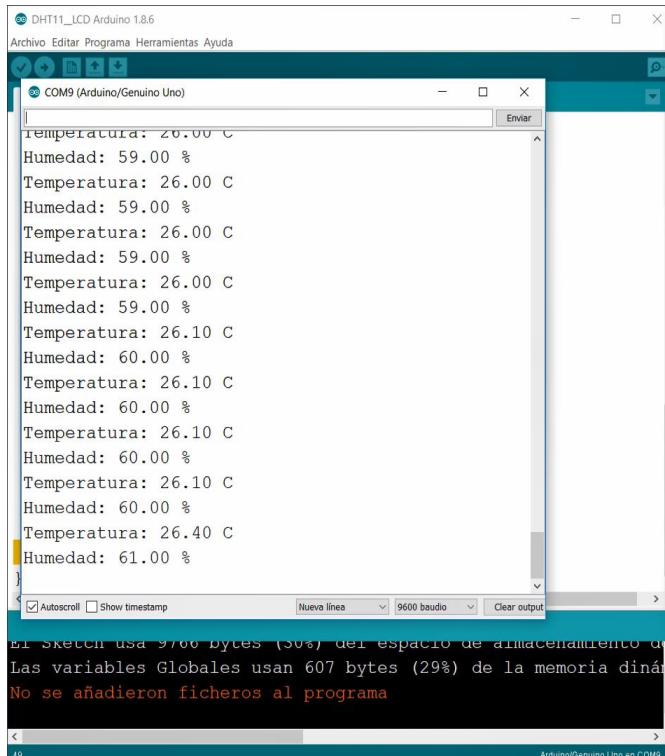
```

    lcd.print(event.temperature);
    lcd.print("C");
}

//SE REALIZAR LECTURA DE LA HUMEDAD Y SE COMPARA
dht.humidity().getEvent(&event);
if (isnan(event.relative_humidity)) {
    Serial.println(F("Error al leer Humedad"));
}
else {
    Serial.print(F("Humedad: "));
    Serial.print(event.relative_humidity);
    Serial.println(F(" %"));
    lcd.setCursor(0,1);
    lcd.print("HUMEDAD ");
    lcd.print(event.relative_humidity);
    lcd.print("%");
}
delay(500);
}

```

RESULTADO



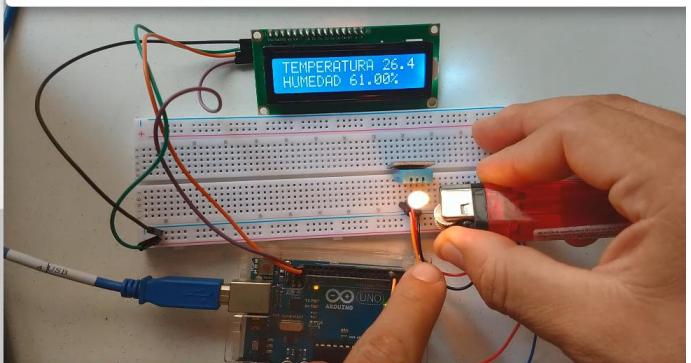
```

DHT11_LCD Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
COM9 (Arduino/Genuino Uno)
Temperatura: 26.00 °C
Humedad: 59.00 %
Temperatura: 26.10 °C
Humedad: 60.00 %
Temperatura: 26.40 °C
Humedad: 61.00 %
}

```

El Sketch usa 9700 bytes (50%) del espacio de almacenamiento disponible. Las variables Globales usan 607 bytes (29%) de la memoria dinámica. No se añadieron ficheros al programa.

CURSO ARDUINO



**DHT11
TEMPERATURA/HUMEDAD**

SENSOR DE TEMPERATURA LM35



MARCO TEÓRICO

El sensor de temperatura LM35 detecta temperaturas desde -55°C a 150°C, 1°C equivale a 10mV y soporta voltajes de entre 4V y 30V. Cuando se realiza la lectura de este sensor analógico con Arduino se hace a través de la función **analogRead** que nos da un valor entre 0 y 1023, 1024 valores posibles. Si tenemos 0V a la entrada nos devolverá 0 y si tenemos 5V nos devolverá 1023.

A partir de esta información podemos obtener una fórmula matemática que nos calcule la temperatura en función del voltaje que nos facilita el LM35.

$$\text{Temperatura} = \text{Valor} * 5 * 100 / 1024$$

Características

- Calibrado directamente en Celsius
- Escala de factor lineal
- Exactitud garantizada 0.5 °C (a +25 °C)
- Rango entre -55° a +150°C
- Conveniente para aplicaciones remotas
- Bajo costo debido al ajuste del wafer-level
- Opera entre 4 y 30 volts de alimentación
- Bajo autocalentamiento

Fuente de Información:

<http://blog.utp.edu.co/jnsanchez/files/2011/03/LM351.pdf>

<https://programarfácil.com/tutoriales/fragmentos/leer-el-sensor-de-temperatura-lm35-en-arduino/>

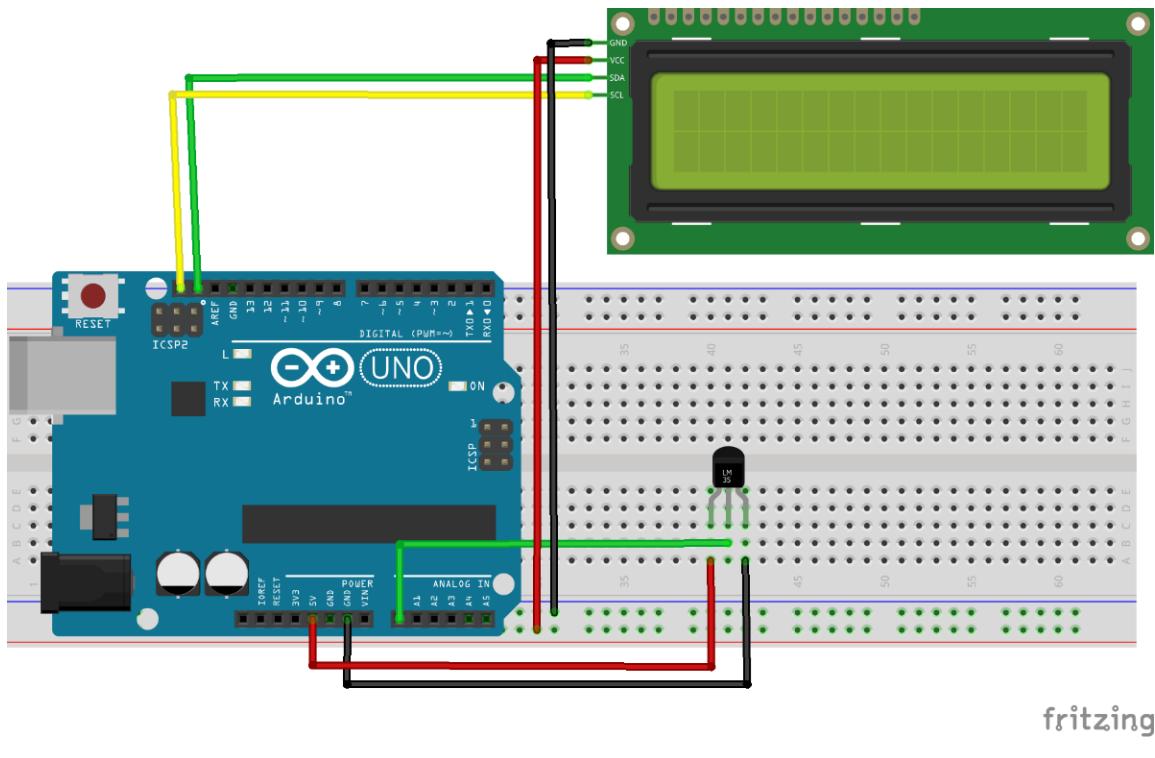
DESCRIPCIÓN DE EJEMPLO

Para este ejemplo se realiza la lectura del sensor mediante el pin analógico **A0** del Arduino, utilizando la función **analogRead**, después a este valor que oscila entre 0 y 1023 se le aplica la formula anterior para obtener la temperatura en grados **Celsius** y se imprime tanto en **Monitor Serie** como en pantalla **LCD**.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Sensor LM35
- Pantalla LCD 16x02 con I2C

DIAGRAMA DE CONEXIÓN



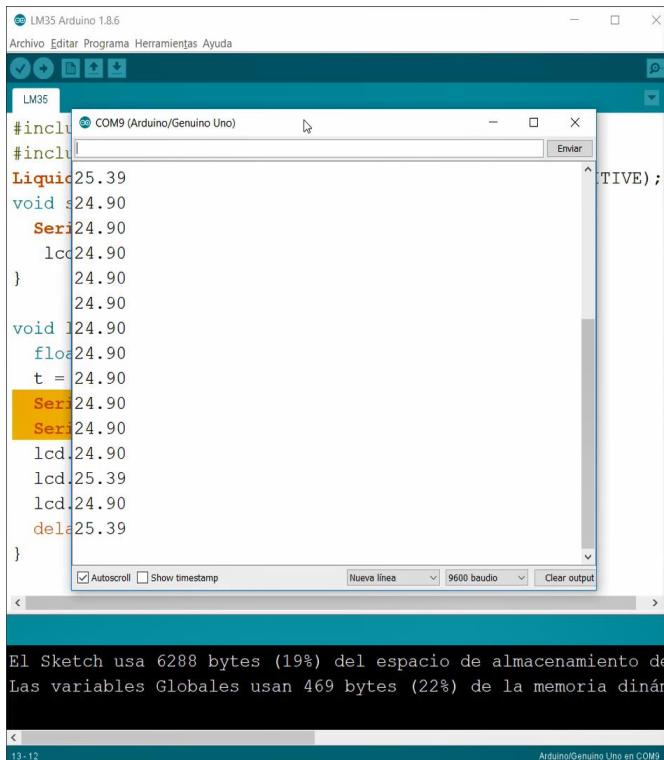
fritzing

CÓDIGO FUENTE

```
//SE INCLUYEN LIBRERÍAS PARA LCD
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
//SE CREA OBJETO PARA CONTROLAR LCD
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
// SE INICIALIZA MONITOR SERIE Y PANTALLA LCD
void setup() {
  Serial.begin(9600);
  lcd.begin(16,2);
}

void loop() {
  float t = analogRead(A0); //SE REALIZAR LECTURA DE SENSO LM35
  t = (5.0 * t * 100.0)/1024.0; //SE APLICA FÓRMULA PARA CALCULAR TEMPERATURA
  Serial.println(t); // SE IMPRIME VALOR EN MONITOR SERIE
  lcd.setCursor(0,0); //SE UBICA CURSOR EN PANTALLA LCD
  lcd.print("TEMP: "); // SE IMPRIME TEXTO Y VALOR DE TEMPERATURA EN LCD
  lcd.print(t);
  delay(1000); // PASUA DE 1 SEGUNDO
}
```

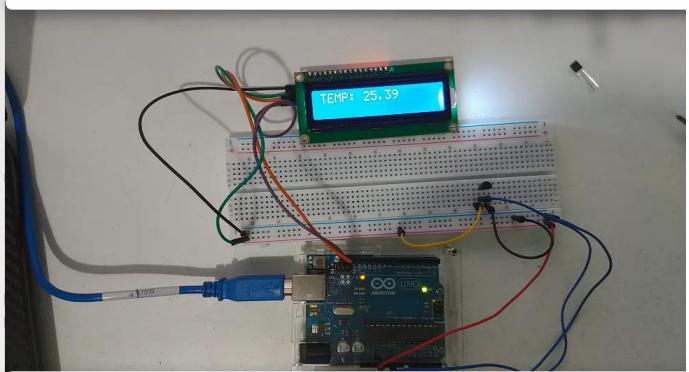
RESULTADO



```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  // initialize the LCD
  lcd.begin(16, 2);
  lcd.print("TEMP: 25.39");
}
void loop() {
  float t = analogRead(A0);
  t = t * 5.0 / 1024.0;
  t = t - 0.5;
  lcd.setCursor(0, 1);
  lcd.print("TEMP: ");
  lcd.print(t);
}
```

El Sketch usa 6288 bytes (19%) del espacio de almacenamiento de Flash. Las variables Globales usan 469 bytes (22%) de la memoria dinámica.

CURSO ARDUINO



SENSOR TEMPERATURA LM35

RTC RELOJ DE TIEMPO REAL DS1302



MARCO TEÓRICO

Un módulo RTC (Real Time Clock) o Reloj de tiempo real consiste en un circuito integrado alimentado por una batería 2032 el cual, en todo momento, registra la fecha, día de la semana y hora al igual que un reloj digital convencional. Sólo que estos datos únicamente podrán ser consultados mediante comunicación I2C.

DESCRIPCIÓN

El chip DS1302 contiene un reloj/calendario de tiempo real y 31 bytes de RAM estática. Se comunica con un microprocesador mediante una interfaz serial simple.

Solamente se requieren 3 alambres para comunicarse con el reloj y la RAM: CE (chip enable), I/O (data line) y SCLK (serial clock). Los datos pueden ser transferidos desde y hacia el reloj/RAM 1byte a la vez o 31 bytes de una sola vez. El DS1302 está diseñado para operar con muy baja potencia y retiene los datos y la información del reloj con menos de 1 μ W.

Para su funcionamiento se necesita un cristal de 32.768 kHz con capacitancia de carga de 6 pF (vendido por separado). Recuerde que hablamos del Chip, el módulo si tiene el cristal.

Características:

- Cuenta segundos, minutos, horas, mes, día de la semana y año
- Año bisiesto válido hasta el 2100
- Soporta modo de 24 o 12 horas con indicador AM/PM
- RAM estática de 31 bytes
- Empaque: 8-pin DIP

Especificaciones Clave:

- Requerimiento de potencia: 2 a 5 VDC; menos de ~300 nA
- Comunicación: TTL compatible serial sincrónico de 3 líneas
- Temperatura de Operación: -32° a +185° F (-0° a + 70° C)
- Dimensiones: 0.4 x .025 in (10 x 6.4 mm)

Fuente de información:

<http://librearduino.blogspot.com/2014/01/rtc-arduino-modulo-reloj-tiempo-real-tutorial.html>
[Http://www.bolanosdj.com.ar/movil/arduino2/rtcds1302.pdf](http://www.bolanosdj.com.ar/movil/arduino2/rtcds1302.pdf)

DESCRIPCIÓN DE EJEMPLO

Se conectar el **RTC DS1302**, especialmente 3 pines **CLK** al pin 5, **DAT** al pin 4 y **RST** al pin 2, básicamente el funcionamiento refiere a configurar una fecha y hora de inicio y posteriormente arranca el reloj con su ciclo normal para marca fecha y hora sin necesidad de que el Arduino esté conectado al equipo de cómputo, debido a que cuenta con la batería **2032** para su alimentación independiente.

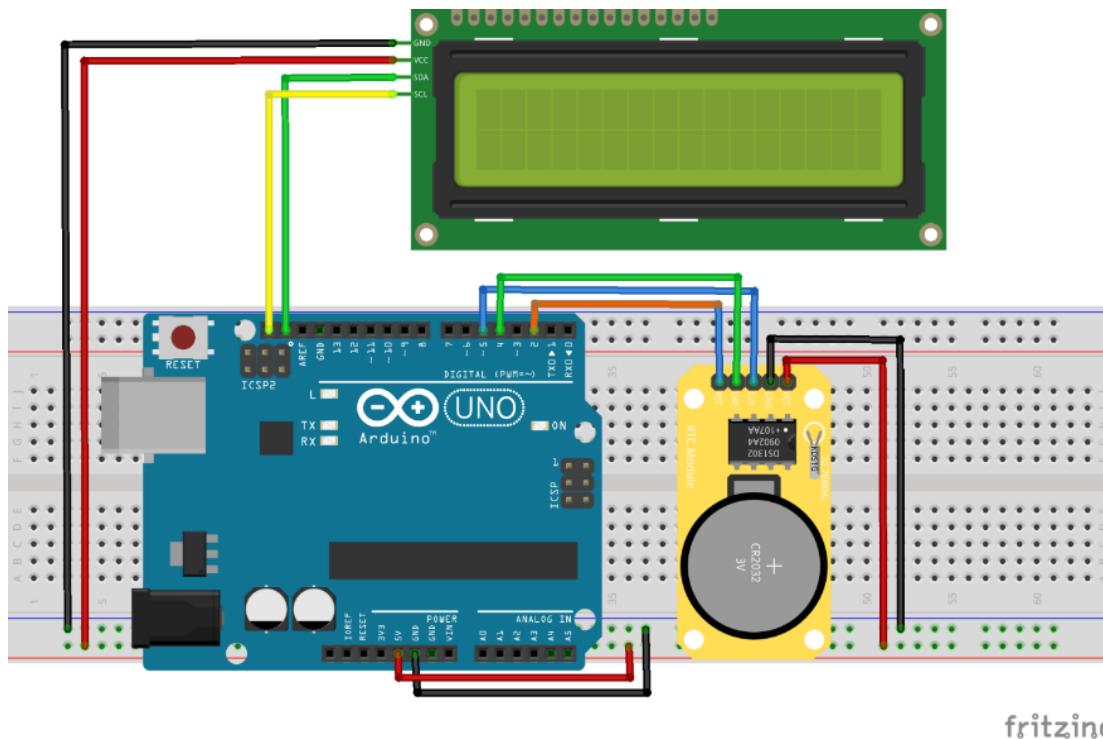
Para este ejemplo se requiere la librería **ThreeWire.h** y **RtcDS1302.h** que se puede agregar descargando de los siguientes enlaces:

<https://github.com/Makuna/Rtc>

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- RTC DS1302
- Pantalla LCD 16x02 con I2C
- Batería 2032 3v

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```
//SE AGREGA LIBRERIAS DE LCD
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
```

```

//SE AGREGA LIBRERÍAS PARA RTC

#include <ThreeWire.h>
#include <RtcDS1302.h>
//SE CREA Y CONFIGURA OBJETO PARA MANEJAR RTC
ThreeWire myWire(4,5,2); //PINES IO, SCLK, CE -> DE RTC DS1302
RtcDS1302<ThreeWire> Rtc(myWire);

void setup () {
    // SE INICAR MONITOR SERIE E IMPRIME FECHA Y HORA
    Serial.begin(57600);
    Serial.print("CONFIGURACIÓN INICIAL ");
    Serial.print(__DATE__);
    Serial.println(__TIME__);
    lcd.begin(16,2); //SE INICIARLIZA LCD
    Rtc.Begin(); // SE INICIALIZA RTC
    RtcDateTime compiled = Rtc.GetDateTime(__DATE__, __TIME__); //SE CONFIGURA FECHA Y HORA ACTUAL
    printDateTime(compiled); //SE IMPRIME FECHA Y HORA
    Serial.println();
}

void loop () {
    //SE OBTENIE FECHA Y HORA ACTUAL Y SE ACTUALIZA
    RtcDateTime now = Rtc.GetDateTime();
    printDateTime(now);
    Serial.println();
    if (!now.IsValid()){
        Serial.println("ERROR EN FECHA Y HORA");
    }
    delay(1000);RETARDO DE 1 SEGUNDO
}

#define countof(a) (sizeof(a) / sizeof(a[0]))

//FUNCION PARA IMPRIMIR FECHA Y HORA CON FORMATO ADECUADO

void printDateTime(const RtcDateTime& dt){
    char datestring[20];
    snprintf_P(datestring,
               countof(datestring),
               PSTR("%02u/%02u/%04u %02u:%02u:%02u"), //SE CONFURA FORMATO DE FECHA Y HORA
               dt.Month(),
               dt.Day(),
               dt.Year(),
               dt.Hour(),
               dt.Minute(),
               dt.Second() );
    //SE IMPRIME LA FECHA Y HORA EN LCD
    lcd.setCursor(0,0);
    lcd.print("DATE ");
    lcd.print(dt.Month());
    lcd.print("/");
    lcd.print(dt.Day());
    lcd.print("/");
    lcd.print(dt.Year());
    lcd.setCursor(0,14);
    lcd.print(" ");
    lcd.setCursor(0,1);
    lcd.print("TIME ");
    lcd.print(dt.Hour());
    lcd.print(":");
    lcd.print(dt.Minute());
    lcd.print(":");
    lcd.print(dt.Second());
    Serial.print(datestring); //SE IMPRIME EN MONITOR SERIE
}

```

RESULTADO

The screenshot shows the Arduino IDE interface. On the left, the Serial Monitor window displays a series of timestamp entries from "07/09/2019 16:51:25" to "07/09/2019 16:51:40". Below the monitor is the code for the sketch, which includes a loop that prints the current second and date/time string via the Serial port. At the bottom, a status message indicates the sketch uses 7730 bytes (23%) of storage and global variables use 540 bytes (26% of dynamic memory). On the right, the Monitor Serie window shows the same timestamp data.

```

07/09/2019 16:51:25
07/09/2019 16:51:26
07/09/2019 16:51:27
07/09/2019 16:51:28
07/09/2019 16:51:29
07/09/2019 16:51:30
07/09/2019 16:51:31
07/09/2019 16:51:32
07/09/2019 16:51:33
07/09/2019 16:51:34
07/09/2019 16:51:35
07/09/2019 16:51:36
07/09/2019 16:51:38
07/09/2019 16:51:39
07/09/2019 16:51:40

lcd.print(":");
lcd.print(dt.Second());
Serial.print(datestring);

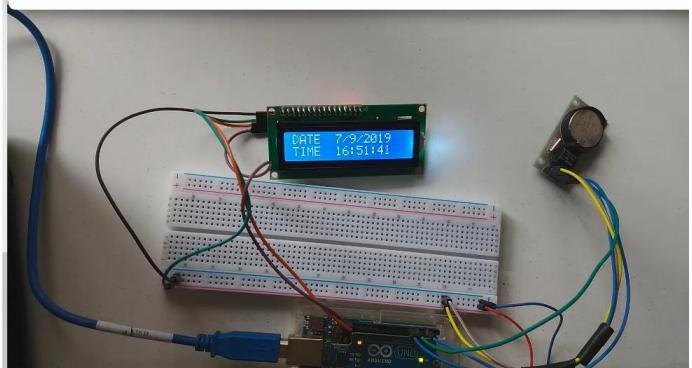
}

El Sketch usa 7730 bytes (23%) del espacio de almacenamiento de
Las variables Globales usan 540 bytes (26%) de la memoria dinámica

37
Arduino/Genuino Uno en COM9

```

CURSO ARDUINO



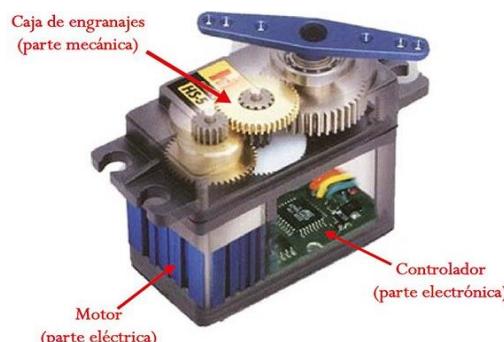
**RTC DS1302
RELOJ DE TIEMPO REAL**

SERVOMOTOR SG90

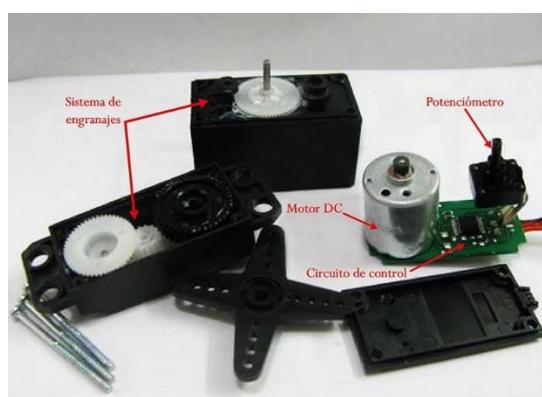


MARCO TEÓRICO

Un servomotor (o servo) es un tipo especial de motor con características especiales de control de posición. Al hablar de un servomotor se hace referencia a un sistema compuesto por componentes electromecánicos y electrónicos.



El motor en el interior de un servomotor es un motor DC común y corriente. El eje del motor se acopla a una caja de engranajes similar a una transmisión. Esto se hace para potenciar el torque del motor y permitir mantener una posición fija cuando se requiera. De forma similar a un automóvil, a menor velocidad, menor torque. El circuito electrónico es el encargado de manejar el movimiento y la posición del motor.

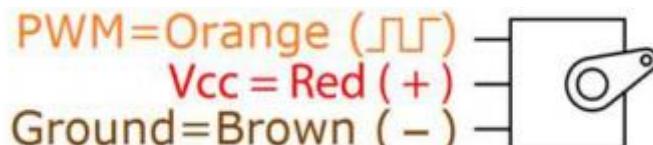


Tipos de servomotores

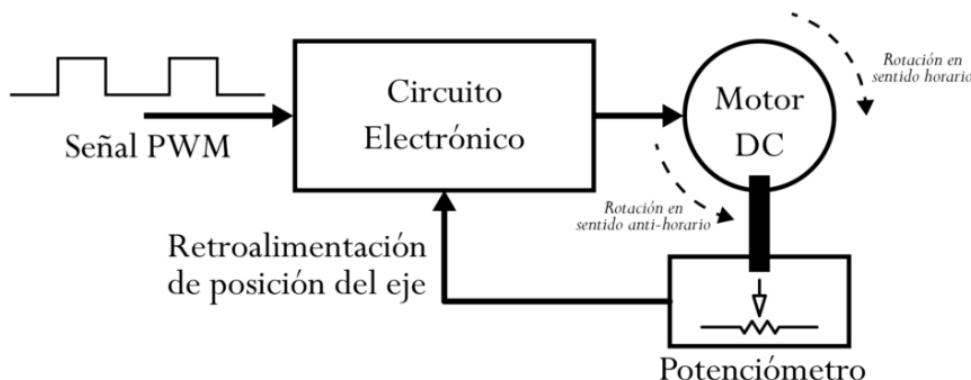
- **Servomotores de rango de giro limitado 0 a 180 grados**
- **Servomotores de rotación continua 360 grados**

Funcionamiento de un servomotor

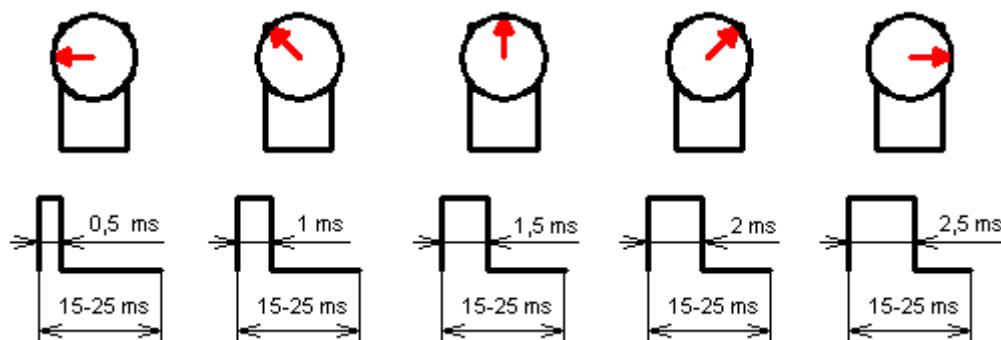
El servo SG90 tiene un conector universal tipo "S" y los cables en el conector están distribuidos de la siguiente forma: Rojo = Alimentación (+), Marrón= Alimentación (-) o tierra, Naranja= Señal PWM.



El diagrama de bloque del servomotor representa de forma visual el servomotor como un sistema. El circuito electrónico es el encargado de recibir la señal PWM y traducirla en movimiento del Motor DC. El eje del motor DC está acoplado a un potenciómetro, el cual permite formar un divisor de voltaje. El voltaje en la salida del divisor varía en función de la posición del eje del motor DC.



Las señales de PWM requeridas para que el circuito de control electrónico es similar para la mayoría de los modelos de servo. Esta señal tiene la forma de una onda cuadrada. Dependiendo del ancho del pulso, el motor adoptará una posición fija.



Ancho de pulsos para lograr diferentes posiciones en un servomotor (180°, 135°, 90°, 45° y 0°)

Las señales que vemos en la imagen son las que permiten que el eje del motor adquiera determinada posición. Éstas señales deben repetirse en el tiempo para que el motor mantenga una posición fija.

Fuente de información:

http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
<https://www.iberobotics.com/producto/micro-servo-towerpro-sg90-1-8kg9g0-12seg/>
<http://panamahitek.com/que-es-y-como-funciona-un-servomotor/>

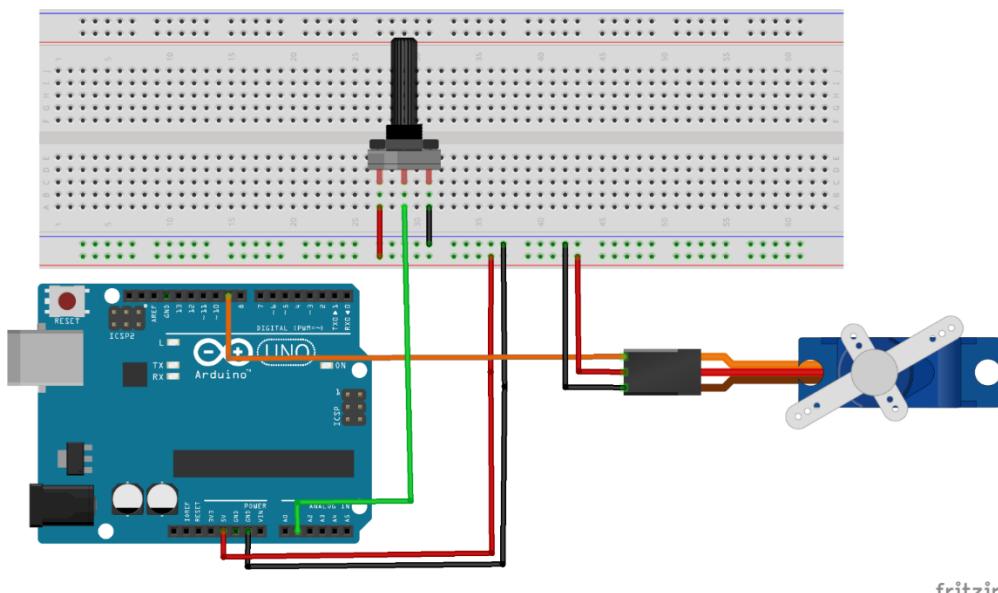
DESCRIPCIÓN DE EJEMPLO

El ejemplo consiste en conectar un servomotor y un potenciómetro al Arduino y controlar el giro del Servo mediante el valor analógico del potenciómetro que es de 0 a 1023, mapeado a valor de en grados de 0 a 180, utilizando la librería **Servo.h**, que viene integrada en la **IDE de Arduino** para poder manipular, se crea un objeto tipo **Servo**, se le asigna el puerto **PWM** con el que será controlador con la función **servo.attach(PIN PWM)** y se escribe mediante la función **write** el valor en grados en que se desea mover el servo (0 a 180).

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Un potenciómetro 5 Kohm
- Un servomotor SG90

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```
#include <Servo.h> //LIBRERÍA DE SERVOMOTOR
Servo myservo; // SE CREA OBJEETO TIPO SERVO
int val; //SE DECLARA VARIABLE DE VALOR DE SERVO

void setup() {
  myservo.attach(9); //SE ASIGNA PIN 9 PARA CONTROL DEL SERVO
  Serial.begin(9600); // SE INICIA MONITOR SERIE
}

void loop() {
  val = analogRead(A1); //SE LEE VALOR DEL POTENCIOMETRO
```

```

val = map(val, 0, 1023, 0, 180); //SE MAPEA VALOR DE 0-1023 A 0-180
Serial.print("Angulo:");
Serial.println(val);//SE IMPRIME VALOR EN MONITOR SERIE
myservo.write(val);//SE ESCRIBE VALOR SOBRE EL SEROMOTOR
delay(15);//RETARDO
}

```

RESULTADO

```

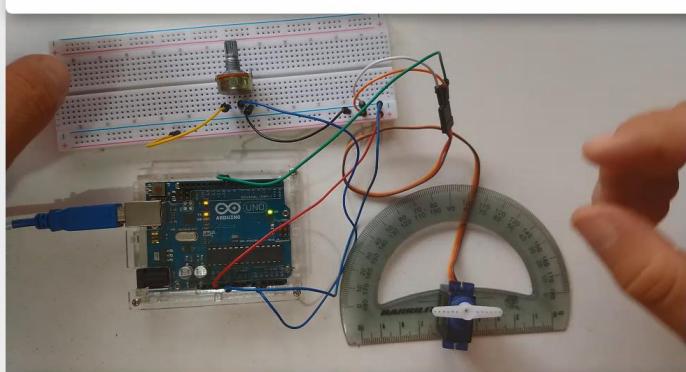
servomotor Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
servomotor
#inclu COM9 (Arduino/Genuino Uno)
Servo
int val=Angulo:180
void sAngulo:180
myseAngulo:180
SeriAngulo:180
} Angulo:180
Angulo:180
void lAngulo:180
val Angulo:180
val Angulo:180
SeriAngulo:180
SeriAngulo:180
myseAngulo:180
del Angulo:180
} Angulo:180

Autoscroll Show timestamp Nuevo linea 9600 baudio Clear output

```

El Sketch usa 3312 bytes (10%) del espacio de almacenamiento de Flash. Las variables Globales usan 239 bytes (11%) de la memoria dinámica.

CURSO ARDUINO



SERVOMOTOR SG90

SENSOR FLAMA



MARCO TEÓRICO

SENSOR DE LLAMA INFRAROJO

El sensor de llama óptico, este dispositivo permite detectar la existencia de calor por la luz emitida de la llama. Esta luz es capturada por las entradas analógicas y digitales de Arduino.

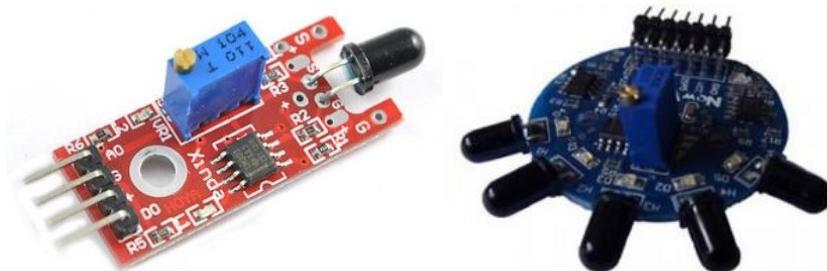
¿QUE ES UNA LLAMA?

La llama es un fenómeno de emisión de luz asociado a los procesos de combustión. Al proceso que desprende grandes cantidades de energía en forma de calor se lo conoce como combustión. Mientras la reacción comienza a realizarse se generan compuestos que liberan parte de su energía mediante la emisión de luz.

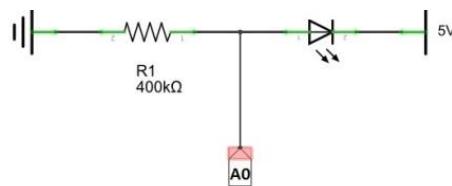
Por este motivo mencionado anteriormente, se incorporan los sensores de llama infrarrojo como dispositivos de seguridad. Al activarse los sensores se puede detener el proceso impidiendo el comienzo de una combustión. Normalmente estos dispositivos combinan las señales ultravioletas y de infrarrojo.

El sensor de flama sencillo y fácil de manejar es un sensor infrarrojo de llamas. Funcionan detectando una longitud de onda específica (de unos 760 nm) que son características de las llamas, aunque son relativamente fáciles de engañar y pueden dar falsos positivos con ciertas luces.

Este sensor se encuentra muy frecuentemente en diferentes versiones, uno encapsulado con un soporte y un potenciómetro para ajustar la sensibilidad, y otro en formato múltiple, con varias cabezas apuntando en distintas direcciones.



Para este caso el ejemplo se realiza con un sensor de flama simple mediante el siguiente circuito



Fuente de información:

<https://www.thermalcombustion.com/2018/que-son-y-como-funcionan-los-sensores-de-flama/>

<https://www.electrontools.com/Home/WP/2018/04/23/que-es-un-sensor-de-llama-infrarojo/>

<https://www.prometec.net/detector-llama/>

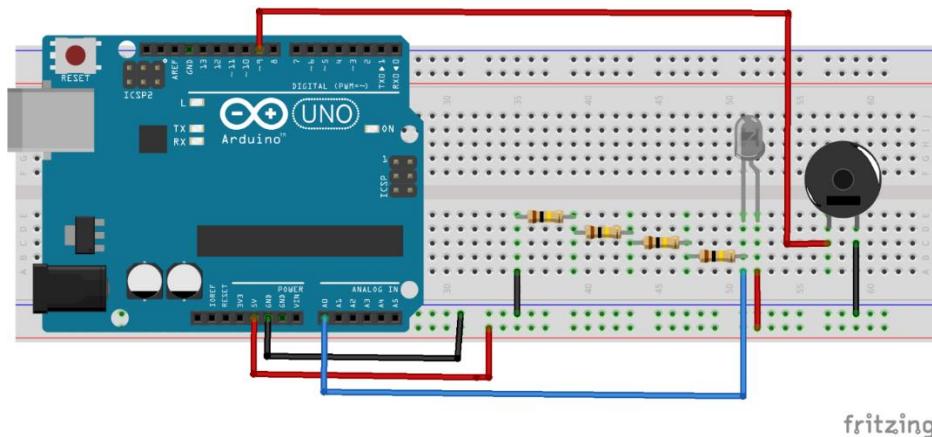
DESCRIPCIÓN DE EJEMPLO

Para este ejemplo se monta el sensor de **Flama** a un arreglo en serie de resistencias de 100 Kohm y se realiza una lectura analógica al pin Negativo (-) del sensor de flama que va al pin **A0** del Arduino, como salida de datos se tiene un **Buzzer** que emite pitidos cuando el sensor detecta una flama o llama.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Sensor de Flama sencillo
- Un Buzzer Activo
- 4 resistencias de 100 Kohm

DIAGRAMA DE CONEXIÓN



fritzing

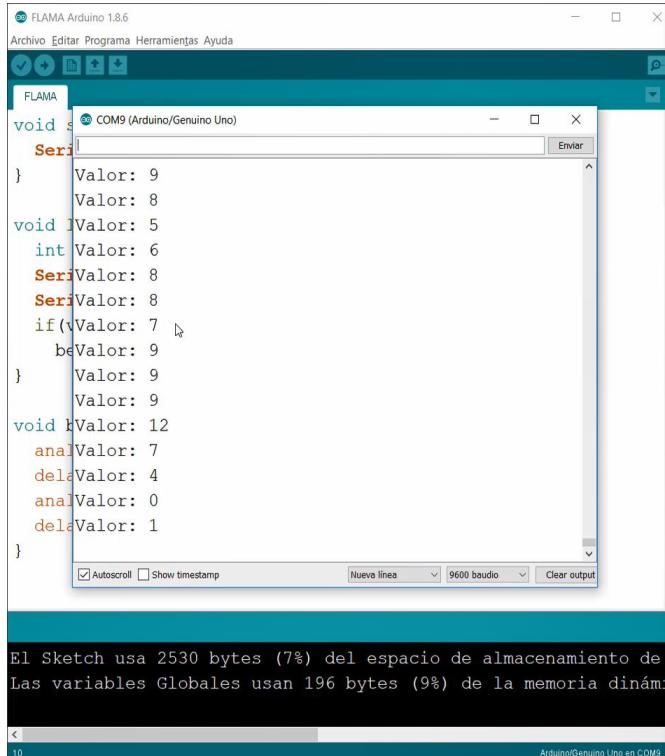
CÓDIGO FUENTE

```
void setup() {
    Serial.begin(9600); // SE INICIALIZA MONITOR SERIE
}

void loop() {
    int v = analogRead(A1); // SE LEE VALOR ANALÓGICO POR PIN A1
    Serial.print("Valor: "); // SE IMPRIME VALOR EN MONITOR SERIE
    Serial.println(v);
    if(v>0)// SI ES MAYOR A 0 LLAMA A LA FUNCION BEEP
        beep(200/v);
}
```

```
//FUNCIÓN BEEP PARA EMITIR UN PITIDO CON BUZZER
void beep(int pausa) {
    analogWrite(9, 40);
    delay(pausa);
    analogWrite(9, 0);
    delay(pausa);
}
```

RESULTADO



```
FLAMA Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
FLAMA
void setup() {
  Serial.begin(9600);
}
void loop() {
  Valor: 9
  Valor: 8
  Valor: 5
  int Valor: 6
  Serial.print("Valor: ");
  Serial.println(Valor);
  Valor: 8
  Valor: 8
  if(Valor == 7) {
    bebe();
  }
  Valor: 9
  Valor: 9
  Valor: 12
  analisis();
  Valor: 7
  delay(4);
  analisis();
  Valor: 0
  delay(1);
}

void bebe() {
  Valor: 12
  analisis();
  Valor: 7
  delay(4);
  analisis();
  Valor: 0
  delay(1);
}

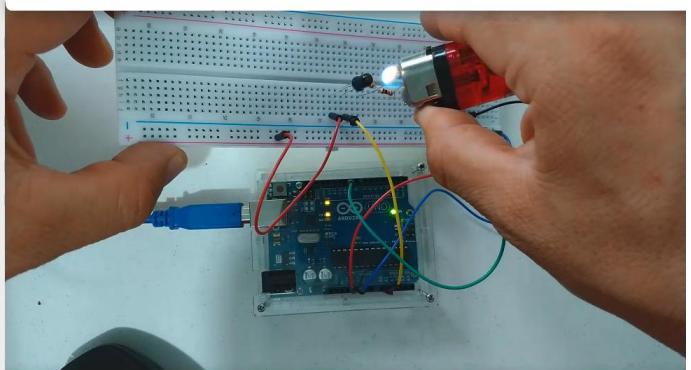
void analisis() {
  Valor: 12
  analisis();
  Valor: 7
  delay(4);
  analisis();
  Valor: 0
  delay(1);
}

Serial.println("Nuevo linea");
Serial.println("9600 báudio");
Serial.println("Clear output");

El Sketch usa 2530 bytes (7%) del espacio de almacenamiento de
Las variables Globales usan 196 bytes (9%) de la memoria dinámica
```

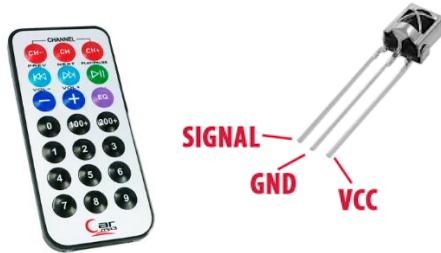
Arduino/Genuino Uno en COM9

CURSO ARDUINO



SENSOR FLAMA

RECEPTOR INFRARROJO (IR)



MARCO TEÓRICO

El receptor de infrarrojos 1838 es un componente que incorpora un diodo IR, amplificador, demodulador y comparador, siendo capaz de recibir señales de luz infrarroja con datos digitales.

Es comúnmente usado como receptor en sistema de control remoto, como los que existen en televisores o equipos de música, y puede ser fácilmente implementado junto a microcontroladores como Arduino.

Especificaciones:

- Tensión de Alimentación: 2.7 a 5.5 Volts DC
- Frecuencia de Operación Central: 38 kHz
- Longitud de Onda de Recepción: 940 nm
- Ángulo de Detección Máximo: 45°
- Temperatura de Operación: -20 a 65 °C
- Alcance: 18 metros en posición frontal (8 metros en ángulo de 45°)
- Número de pines: 3

Fuente de información

<http://www.geekbotelectronics.com/producto/receptor-infrarrojo-hx1838/>

<https://electrocrea.com/products/led-infrarrojo-receptor-vs1838b>

DESCRIPCIÓN DE EJEMPLO

Se conectan 5 LED a la placa Arduino en los pines digitales 2, 3, 4, 5 y 6, a un lado en el pin digital 7 se conecta la salida de señal del receptor IR, es de suma importancia agregar la librería **IRremote.h**, se crea el objeto **IRrecv irrecv(PIN)**, se habilita el receptor mediante la función **enableIRIn()**, se obtiene la lectura del receptor con la función **irrecv.decode(&results)**, posteriormente se imprime el valor numérico con **results.value** y se libera el receptor con **resume()**.

De acuerdo a los valores numéricos de los botones del control IR, decodificados en valores decimales se compara mediante una estructura de control condicional para encender y apagar el LED correspondiente con la asignación de código numérico.

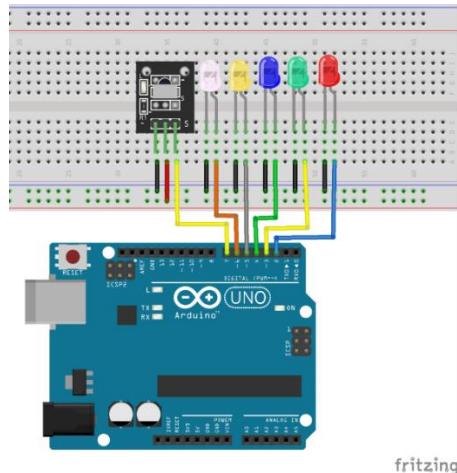
LIBRERÍA

<https://www.arduinolibraries.info/libraries/i-rremote>

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Receptor IR 1838
- Control IR
- 5 LED de colores

DIAGRAMA DE CONEXIÓN



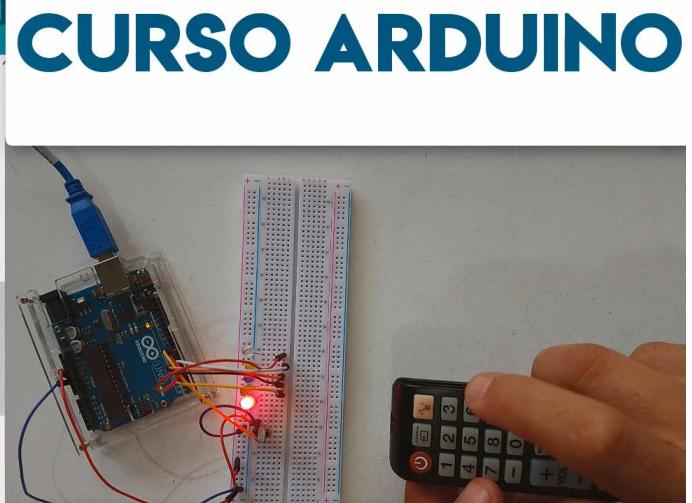
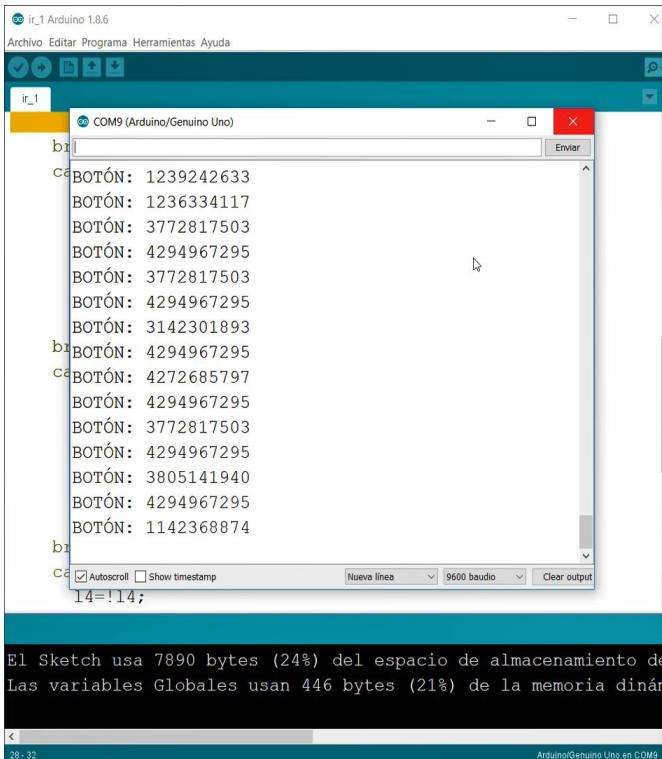
CÓDIGO FUENTE

```
#include "IRremote.h" //LIBRERÍA IR
int receiver = 7; // PIN RECEPTOR IR
IRrecv irrecv(receiver); //SE CREA OBJETO IR
decode_results results; //SE CREA VARIABLE PARA GUARDAR VALOR
//SE CREA VARIABLES BOOLEANAS PARA LEDS
boolean l1=false,l2=false,l3=false,l4=false,l5=false;
int v;
void setup(){
  Serial.begin(9600);
  Serial.println("IR FUNCIONANDO");
  irrecv.enableIRIn(); //SE HABILITA RECEPTOR
  //SE INICIALIZA PINES COMO SALIDA
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
}

void loop() {
  if (irrecv.decode(&results)){//SE LEE VALOR DEL RECEPTOR
    v=results.value;// SE OBTIENE VALOR NUMERICO DE LA LECTURA
    Serial.print("BOTÓN: ");
    Serial.println(results.value);//SE IMPRIME VALOR EN MONITOR SERIE
    irrecv.resume(); //SE LIBERA RECEPTOR
  }
  // DE ACUERDO AL VALOR RECIBIDO SE PRENDE O SE APAGA LED SEGUN EL VALOR
  switch(v){
    case 3772784863:
      l1=!l1;
      if(l1){
        digitalWrite(2,HIGH);
      }else{
        digitalWrite(2,LOW);
      }
    }
}
```

```
        }
    break;
case 3772817503:
    12=!12;
    if(12){
        digitalWrite(3,HIGH);
    }else{
        digitalWrite(3,LOW);
    }
break;
case 3772801183:
    13=!13;
    if(13){
        digitalWrite(4,HIGH);
    }else{
        digitalWrite(4,LOW);
    }
break;
case 3772780783:
    14=!14;
    if(14){
        digitalWrite(5,HIGH);
    }else{
        digitalWrite(5,LOW);
    }
break;
case 3772813423:
    15=!15;
    if(15){
        digitalWrite(6,HIGH);
    }else{
        digitalWrite(6,LOW);
    }
}
break;
}
```

RESULTADO



RECEPTOR IR

WATER SENSOR



MARCO TEÓRICO

Con este sensor se puede detectar la profundidad del agua, el componente principal es un circuito amplificador que está formado principalmente por un transistor y unas líneas metálicas en el PCB. Cuando está puesto en el agua, el elemento sensor presentará una resistencia que puede cambiar junto con el cambio de profundidad del agua. La señal de la profundidad del agua es convertida en señal eléctrica, y podemos conocer el cambio de profundidad del agua.

Esta placa de sensor de agua puede ser ampliamente utilizado en la detección de la precipitación, nivel del agua, incluso fugas. Las huellas del sensor tienen una resistencia de pull-up débil de $1\text{ M}\Omega$. La resistencia tiene el valor de seguimiento alta hasta que una gota de agua corta la huella del sensor con conexión a tierra.

Este sensor es muy simple y lo único que hay que hacer es conectar a tensión y GND y el tercer pin es una señal analógica, proporcional a la cantidad de agua que detecta.

Fuente de información

<https://aprendiendoarduino.wordpress.com/2018/10/17/sensor-deteccion-de-agua-para-arduino/>

<https://www.prometec.net/sensor-agua/>

DESCRIPCIÓN DE EJEMPLO

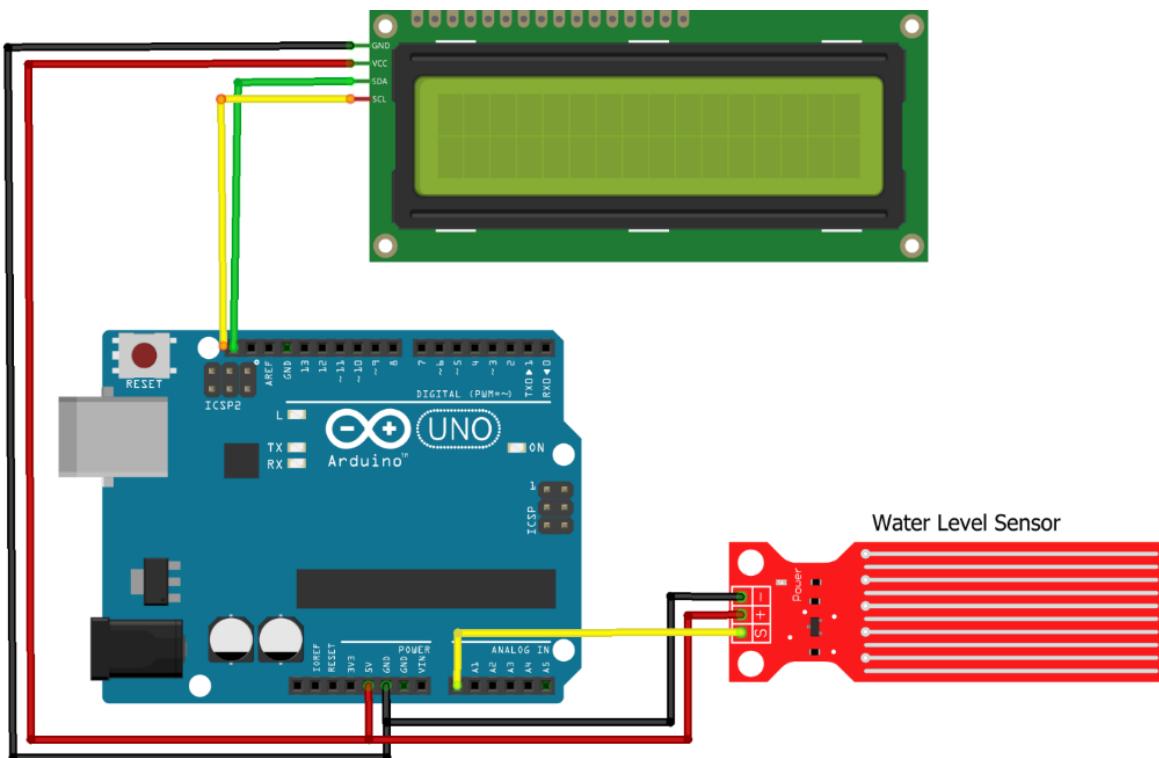
Se conecta el sensor de agua al Arduino mediante el pin analógico A0, se obtiene un valor que oscila entre 0 y 1023 y se imprime en la pantalla LCD, ejemplo bastante sencillo conforme a lo que se viene trabajando.

No se requiere alguna librería para manejar este sensor.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Water sensor
- Pantalla LCD 16x02 con I2C
- Un Vaso con agua

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```
//SE INICIALIZA LIBRERÍAS Y OBJETOS DE LCD
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

//SE CONFIGURA LCD Y MONITOR SERIE

void setup(){
  lcd.begin(16,2);
  Serial.begin(9600);
}

void loop(){
  //SE IMPRIME MENSAJE EN PRIMERA FILA
  lcd.setCursor(0,0);
  lcd.print("CURSO ARDUINO");
  int x=analogRead(A0);//SE REALIZAR LECTURA DEL SENSOR DE AGUA

  //SE IMPRIME VALOR DEL SENSOR EN LA SEGUNDA FILA DEL LCD
  lcd.setCursor(0,1);
  lcd.print("Valor: ");
  lcd.print(x);
  lcd.print(" ");
  delay(1000);// PAUSA DE 1 SEGUNDO
}
```

RESULTADO

```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

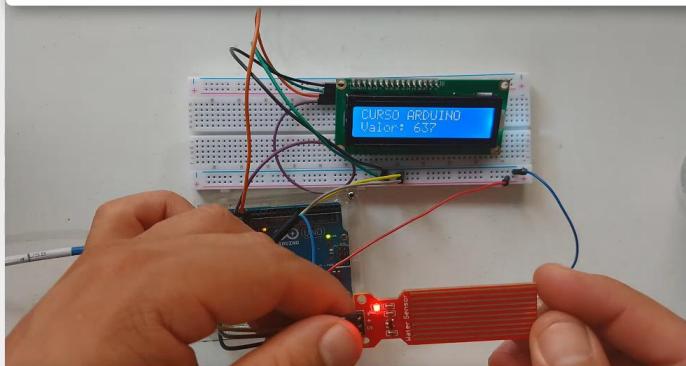
void setup(){
  lcd.begin(16,2);
  Serial.begin(9600);
}

void loop(){
  lcd.setCursor(0,0);
  lcd.print("CURSO ARDUINO");
  int x=analogRead(A0);
  lcd.setCursor(0,1);
  lcd.print("Valor: ");
  lcd.print(x);
  lcd.print("  ");
  delay(1000);
}

```

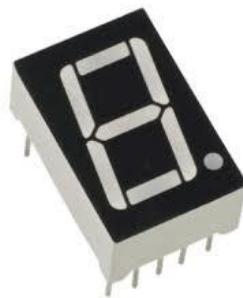
Subido
El Sketch usa 4644 bytes (14%) del espacio de almacenamiento de Las variables Globales usan 473 bytes (23%) de la memoria dinámica

CURSO ARDUINO



WATER SENSOR

DISPLAY 7 SEGMENTOS

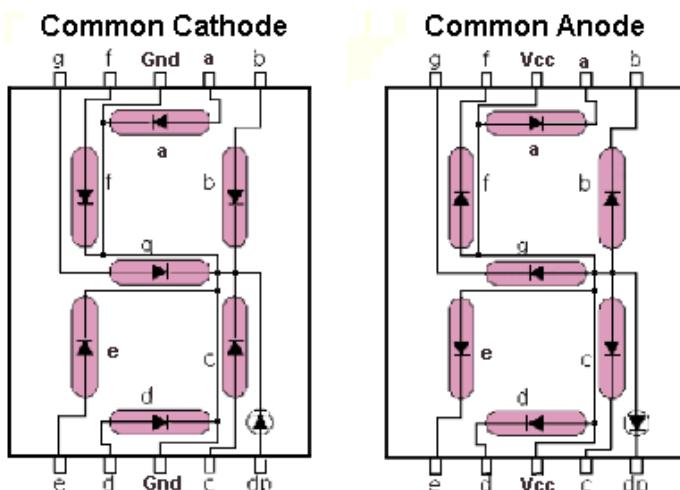


MARCO TEÓRICO

El display 7 segmentos es un componente electrónico muy utilizado para representar visualmente números y letras, es de gran utilidad dado su simpleza para implementar en cualquier proyecto electrónico.

Está compuesto por 7 dispositivos lumínicos (Led) que forman un “8”, de esta forma controlando el encendido y apagado de cada led, podremos representar el numero o letra que necesitamos.

DISPLAY 7 SEGMENTOS DE ANODO COMÚN Y CATODO COMÚN



Existen dos tipos de display de 7 segmentos, su principal diferencia es la conexión que debemos implementar para encenderlos, estos dos tipos se conocen como Ánodo común y Cátodo común. En los 7 segmentos de Cátodo Común, el punto en común para todos los Led es el Cátodo (GND), cero volts, Mientras que el Ánodo común el punto de referencia es Vcc (5 volt).

¿COMO CONTROLO QUE NUMERO QUIERO DIBUJAR?

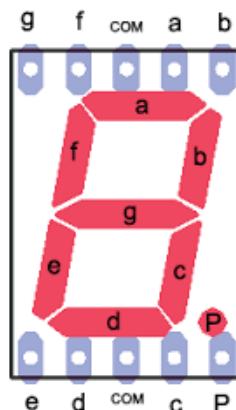
Fácil, Lo primero que tenemos que identificar es con qué tipo de display estamos trabajando (Cátodo o Ánodo común), una vez identificado nos basamos en la siguiente tabla de verdad dado el caso que corresponda.

Catodo Común							
Numero	A	B	C	D	E	F	G
Enable	0	1	1	1	1	1	0
0	1	0	1	1	0	0	0
0	2	1	1	0	1	1	0
0	3	1	1	1	1	0	1
0	4	0	1	1	0	0	1
0	5	1	0	1	1	0	1
0	6	1	0	1	1	1	1
0	7	1	1	1	0	0	0
0	8	1	1	1	1	1	1
0	9	1	1	1	1	0	1

Anodo Común							
Numero	A	B	C	D	E	F	G
Enable	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
1	2	0	0	1	0	1	0
1	3	0	0	0	0	1	0
1	4	1	0	0	1	1	0
1	5	0	1	0	0	1	0
1	6	0	1	0	0	0	0
1	7	0	0	0	1	1	1
1	8	0	0	0	0	0	0
1	9	0	0	0	1	0	0

El Pin de Enabled representa al pin (VCC – GND) de la imagen superior, según sea el tipo de display utilizado. cómo podemos ver el cátodo Común se enciende con un 0 lógico (0 Volt) mientras que el Ánodo Común lo hace con un 1 lógico (5 volt).

Los siguientes pines (A-B-C-D-E-F-G) representan cada led interno de los 7 segmentos, en el caso del Cátodo Común se encenderán con un 1 lógico mientras que en Ánodo Común se encenderá con un 0 Lógico.



Fuente de información

<https://www.electrontools.com/Home/WP/2016/03/09/display-7-segmentos/>

DESCRIPCIÓN DE EJEMPLO

Se conecta un display de 7 segmentos Catado común al Arduino como salida y como entrada un Push Button, que al presionar cambiara el digito que se muestra en el display, iniciando desde 0 al 9, después las letras A, B, C, D y F, de esta ultima se regresa al digito 0.

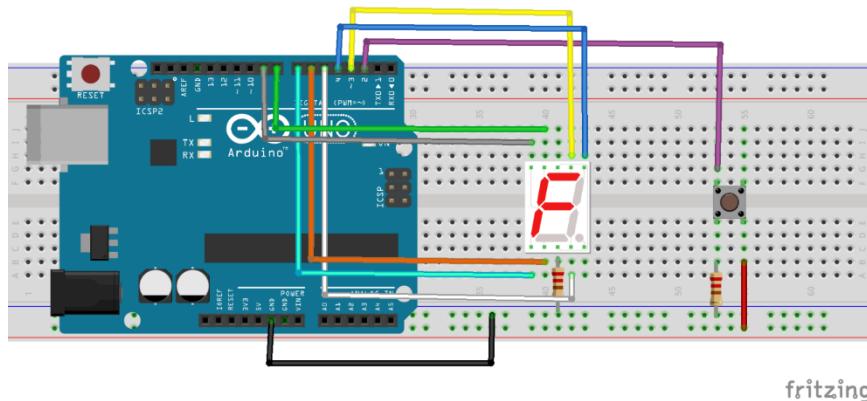
Para desarrollar este ejemplo solo se desarrollan funciones que permiten indicar el segmento del display a encender mediante la letra de la posición,

Para este ejemplo no se requiere librerías.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- 1 display de 7 segmentos Catodo Común
- 1 Push Button
- 2 resistencias de 220 ohm

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```

//SE CREAN VARIABLES DE ACUERDO AL DISPLAY, PUSH
Y CONTADOR
int a=3;
int b=4;
int c=5;
int d=6;
int e=7;
int f=8;
int g=9;
int push=2;
int contador=0;
//SE CONFIGURA PINES DE DISPLAY Y PUSH BUTTON
void setup(){
  pinMode(a,OUTPUT);
  pinMode(b,OUTPUT);
  pinMode(c,OUTPUT);
  pinMode(d,OUTPUT);
  pinMode(e,OUTPUT);
  pinMode(f,OUTPUT);
  pinMode(g,OUTPUT);
  pinMode(push,INPUT);
  Serial.begin(9600);
  cero();
}
void loop(){
  int x=digitalRead(push); //SE LEE EL VALOR DEL
PUSH
  Serial.println(x); //SE IMPRIME EN MONITOR
SERIE
  if(x==1){ //SE SUMA AL CONTADOR
    contador++;
  }
  delay(10);
  //SE IMPRIME EL VALOR DEL CONTADOR EN MONITOR
SERIE
  Serial.print("CONTADOR: ");
  Serial.println(contador);
  //DE ACUERDO AL VALOR DEL CONTADOR
  //LLAMA A LA FUNCION CORRESPONDIENTE
  //PARA IMPRIMIR EL VALOR EN EL DISPLAY
  switch(contador){
    case 0: cero();break;
    case 1: uno();break;
    case 2: dos();break;
    case 3: tres();break;
    case 4: cuatro();break;
    case 5: cinco();break;
    case 6: seis();break;
    case 7: siete();break;
    case 8: ocho();break;
    case 9: nueve();break;
    case 10: aa();break;
    case 11: bb();break;
    case 12: cc();break;
    case 13: dd();break;
    case 14: ee();break;
    case 15: ff();break;
    default: cero();break;
  }
  delay(100);
}
//FUNCIONES DE VALOR DE DISPLAY
void cero(){
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,LOW);
}

```

```

void uno() {
  digitalWrite(a,LOW);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,LOW);
  digitalWrite(e,LOW);
  digitalWrite(f,LOW);
  digitalWrite(g,LOW);
}
void dos() {
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,LOW);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,LOW);
  digitalWrite(g,HIGH);
}
void tres() {
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,LOW);
  digitalWrite(f,LOW);
  digitalWrite(g,HIGH);
}
void cuatro() {
  digitalWrite(a,LOW);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,LOW);
  digitalWrite(e,LOW);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void cinco() {
  digitalWrite(a,HIGH);
  digitalWrite(b,LOW);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,LOW);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void seis() {
  digitalWrite(a,HIGH);
  digitalWrite(b,LOW);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void siete() {
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,LOW);
  digitalWrite(e,LOW);
  digitalWrite(f,LOW);
  digitalWrite(g,HIGH);
}
void ocho() {
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
}

  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void nueve() {
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,LOW);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void aa() {
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,LOW);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void bb() {
  digitalWrite(a,LOW);
  digitalWrite(b,LOW);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void cc() {
  digitalWrite(a,HIGH);
  digitalWrite(b,LOW);
  digitalWrite(c,LOW);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,LOW);
}
void dd() {
  digitalWrite(a,LOW);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,LOW);
  digitalWrite(g,HIGH);
}
void ee() {
  digitalWrite(a,HIGH);
  digitalWrite(b,LOW);
  digitalWrite(c,LOW);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void ff() {
  digitalWrite(a,HIGH);
  digitalWrite(b,LOW);
  digitalWrite(c,LOW);
  digitalWrite(d,LOW);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
  contador=0;
}

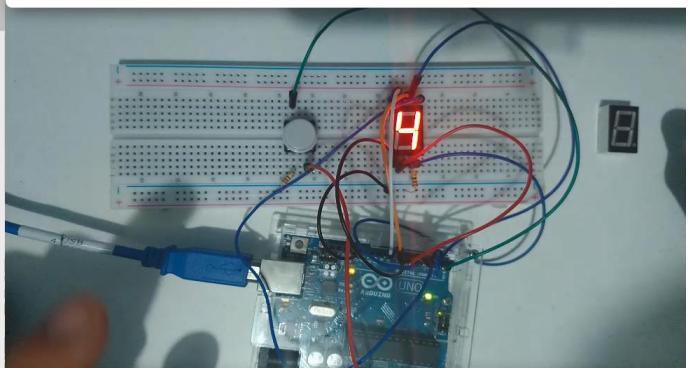
```

RESULTADO

The screenshot shows the Arduino IDE interface. The title bar reads "display_7 Arduino 1.8.6". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". The toolbar has icons for file operations like Open, Save, and Print. The code editor contains the following sketch:

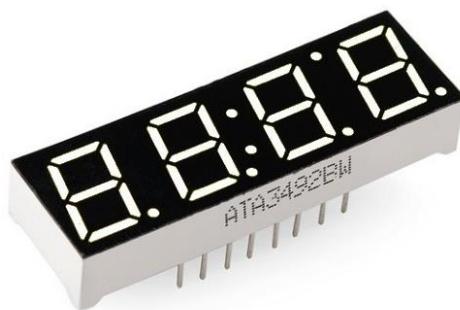
```
void setup() {  
    COM9 (Arduino/Genuino Uno)  
    int  
    Serial.println("CONTADOR: 4");  
    if(0  
    {  
        COM9 (Arduino/Genuino Uno)  
        Serial.println("CONTADOR: 4");  
    }  
    delay(0);  
    Serial.println("CONTADOR: 4");  
    Serial.println("CONTADOR: 4");  
    switch(0  
    {  
        case 0:  
            COM9 (Arduino/Genuino Uno)  
            case 0:  
                COM9 (Arduino/Genuino Uno)  
                case 9: nueve();break;  
    }  
}
```

The serial monitor window is open, showing the output of the sketch. The text "CONTADOR: 4" is repeated multiple times, followed by a series of "COM9 (Arduino/Genuino Uno)" messages. The serial port dropdown shows "COM9 (Arduino/Genuino Uno)". The bottom status bar indicates "197" and "Arduino/Genuino Uno en COM9".



DISPLAY 7 SEGMENTOS

DISPLAY 7 X 4 DÍGITOS

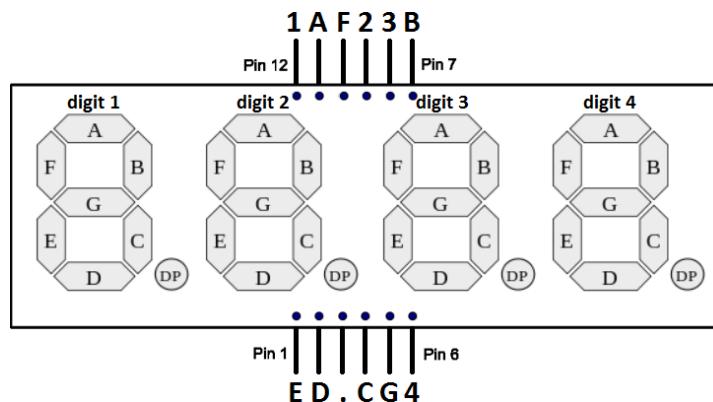


MARCO TEÓRICO

Display 7 segmentos de 4 dígitos Cátodo Pines Horizontales.

El visualizador de siete segmentos (llamado también display) es una forma de representar números en equipos electrónicos. Está compuesto de siete segmentos que se pueden encender o apagar individualmente. Cada segmento tiene la forma de una pequeña línea. Aunque externamente su forma difiere considerablemente de un led típico, internamente están constituidos por una serie de ledes con unas determinadas conexiones internas, estratégicamente ubicados de tal forma que forme un número '8'.

Si se utiliza 4 display de 7 segmentos individuales para representar 4 dígitos, se requiere de 30 a 34 pines digitales de un Arduino, sin embargo, existe una solución a este problema, que utiliza los 8 pines digitales para las letras a, b, c, d, e, f, g y el punto, y se incluye 4 pines más para habilitar el digito a mostrar, en total son 12 pines digitales en lugar de 30 o más para controlar un display de 7 segmentos con 4 dígitos.



En el diagrama se muestra la posición de los 12 símbolos de los pines del display común con el agregado de 4 pines numéricos que indican el número de dígito a habilitar.

DESCRIPCIÓN DE EJEMPLO

Fuente de información:

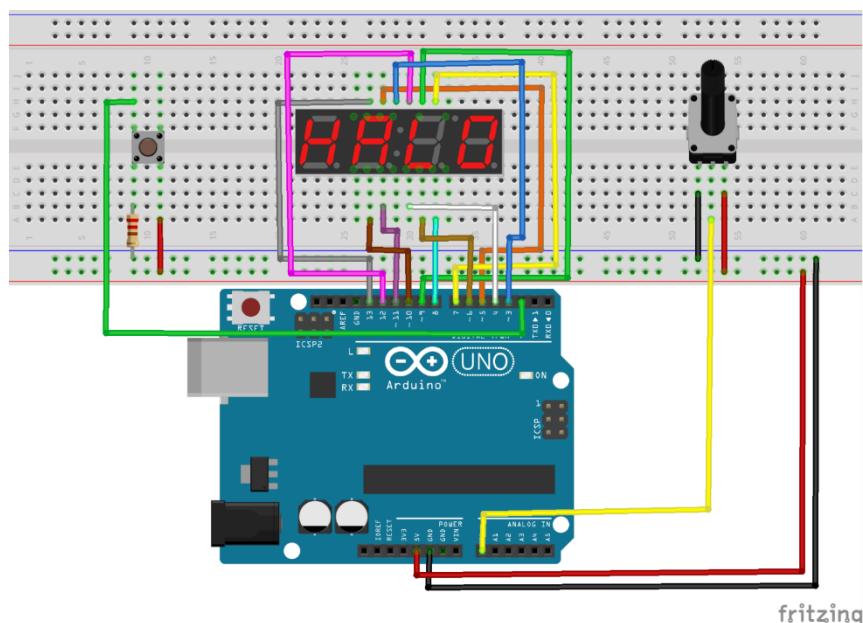
<https://www.prometecl.net/display-4digitos/>

<http://www.geekbotelectronics.com/producto/display-7-segmentos-de-4-digitos/>

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Un potenciómetro 5 Kohm
- Un display 7 segmentos x 4 dígitos
- Un Push Button
- Una resistencia 220 ohm

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

```

long n = 0;
int x = 100;
int del = 1000;
int a=5;
int b=7;
int c=4;
int d=11;
int e=10;
int f=3;
int g=6;
//SE CONFIGURA PINES COMO SALIDA Y ENTRADA
void setup(){
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(2, INPUT); //PUSH BUTTON
  pinMode(A0, INPUT); //POTENCIOMETRO
}

void loop(){
  int v=digitalRead(2); //LEER VALOR DEL PUSH
  //SI SE OPRIME EL PUSH SE REINICIA CONTADOR
  if(v==1){
    limpiar();
    n=0;
    delayMicroseconds(500);
  }
  //LEER VALOR DEL POTENCIOMETRO
  int t=analogRead(A0);
  //DE ACUERDO AL VALOR DE t ES EL RETARDO
  limpiar(); //LIMPIAR DISPLAY'S
  digito(1); //HABILITAR DIGITO 1
  numero((n/x/1000)%10); //OBTENER DIGITO 1
  delayMicroseconds(t); // RETARDO
  limpiar();
  digito(2); //HABILITAR DIGITO 2
  numero((n/x/100)%10); //OBTENER DIGITO 2
  delayMicroseconds(t);
  limpiar();
  digito(3); //HABILITAR DIGITO 3
  numero((n/x/10)%10); //OBTENER DIGITO 2
  delayMicroseconds(t);
  limpiar();
}

```

```

digito(4); //HABILITAR DIGITO 4
numero(n/x % 10); //OBTENER DIGITO 4
delayMicroseconds(t);
if(n/x > 9999){ //SI LLEGA A 9999 SE REINICIAR
    n = 0;
    delay(5000);
}
else n++;
}
// FUNCION PARA HABILITAR DIGITO A IMPRIMIR
void digito(int x){
    digitalWrite(13, HIGH); // dig 1
    digitalWrite(12, HIGH); // dig 2
    digitalWrite( 9, HIGH); // dig 3
    digitalWrite( 8, HIGH); // dig 4
//DE ACUERDO AL VALOR SE HABILITA DIGITO
switch(x){
    case 1: digitalWrite(13, LOW); break;
    case 2: digitalWrite(12, LOW); break;
    case 3: digitalWrite(9, LOW); break;
    case 4: digitalWrite(8, LOW); break;
}
}
//DE ACUERDO AL VALOR SE IMPRIME DIGITO EN
DISPLAY
void numero(int x){
    switch(x){
        case 1: uno(); break;
        case 2: dos(); break;
        case 3: tres(); break;
        case 4: cuatro(); break;
        case 5: cinco(); break;
        case 6: seis(); break;
        case 7: siete(); break;
        case 8: ocho(); break;
        case 9: nueve(); break;
        default: cero(); break;
    }
}
//FUNCION LIMPIAR DIGITO, TODOS LOS LEDS
APAGADOS
void limpiar(){
    digitalWrite( 5, LOW); // A
    digitalWrite( 7, LOW); // B
    digitalWrite( 4, LOW); // C
    digitalWrite(11, LOW); // D
    digitalWrite(10, LOW); // E
    digitalWrite( 3, LOW); // F
    digitalWrite( 6, LOW); // G
}

//FUNCIONES DE NUMEROS PARA DISPLAY
void cero(){
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,HIGH);
    digitalWrite(f,HIGH);
    digitalWrite(g,LOW);
}
void uno(){
    digitalWrite(a,LOW);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,LOW);
    digitalWrite(e,LOW);
    digitalWrite(f,LOW);
    digitalWrite(g,LOW);
}
void dos(){
    digitalWrite(a,HIGH);
    digitalWrite(b,LOW);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,LOW);
    digitalWrite(f,HIGH);
    digitalWrite(g,HIGH);
}
void tres(){
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,LOW);
    digitalWrite(f,LOW);
    digitalWrite(g,HIGH);
}
void cuatro(){
    digitalWrite(a,LOW);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,LOW);
    digitalWrite(f,HIGH);
    digitalWrite(g,HIGH);
}
void cinco(){
    digitalWrite(a,HIGH);
    digitalWrite(b,LOW);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,LOW);
    digitalWrite(f,HIGH);
    digitalWrite(g,HIGH);
}
void seis(){
    digitalWrite(a,HIGH);
    digitalWrite(b,LOW);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,HIGH);
    digitalWrite(f,HIGH);
    digitalWrite(g,HIGH);
}
void siete(){
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,LOW);
    digitalWrite(e,LOW);
    digitalWrite(f,LOW);
    digitalWrite(g,HIGH);
}
void ocho(){
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,HIGH);
    digitalWrite(f,HIGH);
    digitalWrite(g,HIGH);
}
void nueve(){
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,LOW);
    digitalWrite(f,HIGH);
    digitalWrite(g,HIGH);
}
void aa(){
    digitalWrite(a,HIGH);
}

```

```

digitalWrite(b,HIGH);
digitalWrite(c,HIGH);
digitalWrite(d,LOW);
digitalWrite(e,HIGH);
digitalWrite(f,HIGH);
digitalWrite(g,HIGH);
}
void bb() {
  digitalWrite(a,LOW);
  digitalWrite(b,LOW);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
}
void cc() {
  digitalWrite(a,HIGH);
  digitalWrite(b,LOW);
  digitalWrite(c,LOW);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,LOW);
}
void dd() {
}
  
```

RESULTADO

```

displex4_2 Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
displex4_2
else n++;
}

void digito(int x){
  digitalWrite(13, HIGH); // dig 1
  digitalWrite(12, HIGH); // dig 2
  digitalWrite( 9, HIGH); // dig 3
  digitalWrite( 8, HIGH); // dig 4

  switch(x) {
    case 1: digitalWrite(13, LOW); break;
    case 2: digitalWrite(12, LOW); break;
    case 3: digitalWrite(9, LOW); break;
    case 4: digitalWrite(8, LOW); break;
  }
}

void numero(int x){
  switch(x) {
    
```

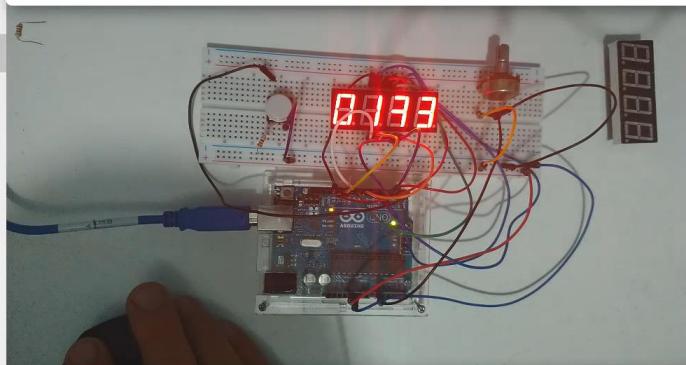
Subido

El Sketch usa 2212 bytes (6%) del espacio de almacenamiento de Las variables Globales usan 13 bytes (0%) de la memoria dinámica

52 - 55

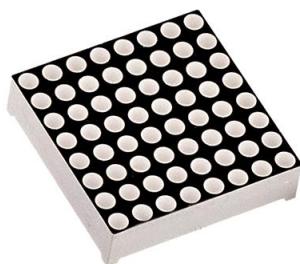
Arduino/Genuino Uno en COM9

CURSO ARDUINO



DISPLAY 7 SEGMENTOS DE 4 DÍGITOS

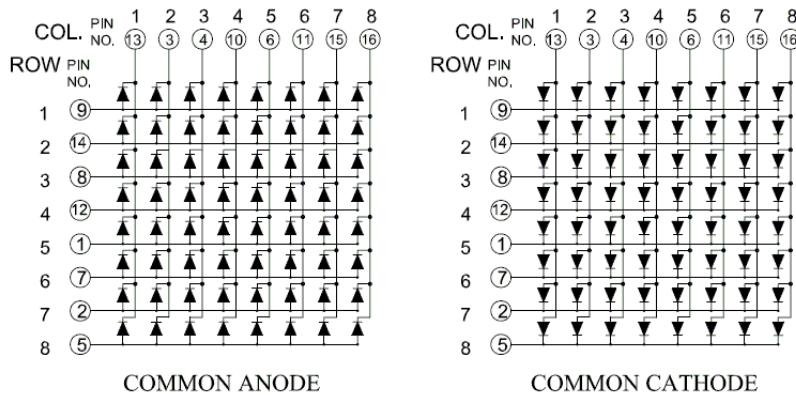
MATRIZ DE LEDS 8X8



MARCO TEÓRICO

La matriz de LEDs (o LED arrays) son, como su nombre indica, una matriz de diodos LED normales que se comercializa en multitud de formatos y colores. Desde las de un solo color, a las que tienen varios colores posibles, e incluso las hay de una matriz RGB.

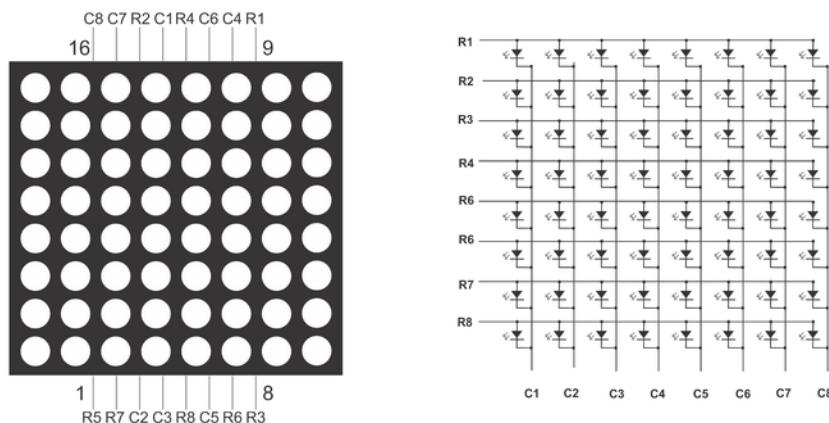
INTERNAL CIRCUIT DIAGRAM



Si los diodos se unen por el positivo, se dice que la matriz es Ánodo común y se une por el negativo la matriz es Cátodo común.

Si ponemos HIGH en una columna, digamos la 2, no se iluminará nada aún. Pero cuando hagamos LOW en, digamos la fila 4, se cerrará el circuito a GND el pin col 2 x fila 4, se encenderá.

A continuación, se muestra la distribución de los pines de Filas y Columnas de la Matriz de LED 8x8 modelo 1588BS



Fuente de información:

<https://www.prometec.net/matriz-led-8x8/>

<https://www.microproyectos.net/blog/wp-content/uploads/2017/03/Matrix-8X8-Pinout.png>

DESCRIPCIÓN DE EJEMPLO

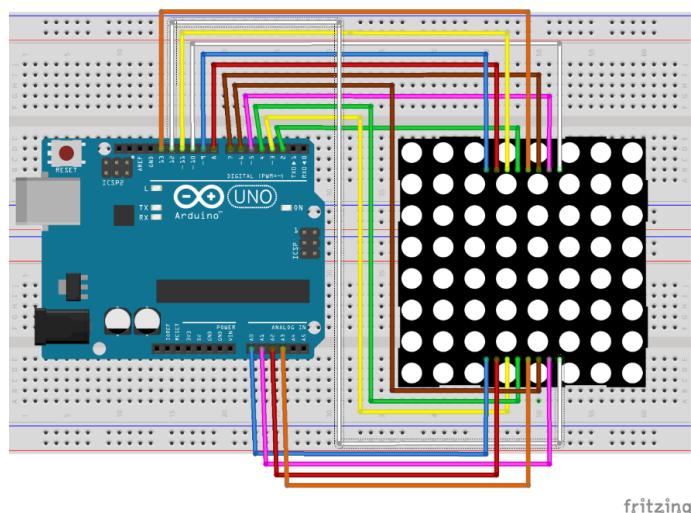
Se conecta la matriz de 8x8 LED's mediante 2 protoboard debido a que no se ajusta a uno solo, después se conecta conforme al diagrama, Columnas van del Pin 2 al 9 y la Filas van del Pin 10,11, 12, 13, A0, A1, A2 y A3.

Se carga dos códigos diferentes, uno para testear en encendido y apagado de cada LED, y un segundo código que permite mostrar un mensaje en la Matriz (KODEMY), donde se define el texto mediante un array de decimales que posteriormente se convierten a binario, donde 1 indica LED prendido y 0 LED apagado, en este sentido se forma las letras con los LED encendidos.

MATERIAL A UTILIZAR

- Arduino UNO
- 2 Protoboard
- Cables
- Matriz de LEDs 8x8

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

Test de encendido de leds

```

void setup(){ //SE INICIALIZAR LOS PINES DEL 2 AL 18 COMO SALIDA
    for (int j=2; j<19; j++)
        pinMode(j, OUTPUT);
}
void loop(){
    // SE LIMPIA LA MATRIZ DE LEDS
    for (int j=2; j<10; j++){
        digitalWrite(j, HIGH);
        for (int k= 10 ; k<18 ; k++) {
            digitalWrite(k, LOW);
        }
    }
    //SE ENCIENDE LED POR LEDS
    //RECOGE CADA FILA DE CADA COLUMNA
    for (int j=2; j<10; j++) {
        digitalWrite(j, LOW);
        for (int k= 10 ; k<18 ; k++) {
            digitalWrite(k, HIGH);
            delay(250);
            digitalWrite(k, LOW);
        }
        digitalWrite(j, HIGH);
    }
}

```

TEXTO

```

//SE DECLARA ARRAY DE BYTES CON LOS VALORES NECESARIOS
//PARA FORMAR CADA LETRA 0-> APAGADO, 1->ENCENDIDO
//SON 8 VALORES, UNO POR CADA FILA
// PARA FORMAR LA LETRA "K" SE CALCULA->
//EL VALOR 63 EQUIVALE A 01100011
//EL VALOR 66 EQUIVALE A 01100110
//EL VALOR 6C EQUIVALE A 01101100
//EL VALOR 78 EQUIVALE A 01111000
//EL VALOR 78 EQUIVALE A 01111000
//EL VALOR 6C EQUIVALE A 01101100
//EL VALOR 66 EQUIVALE A 01100110
//EL VALOR 63 EQUIVALE A 01100011
byte K[] = { 0x63, 0x66, 0x6C, 0x78, 0x6C, 0x66, 0x63 };
byte O[] = { 0x3C, 0x42, 0x42, 0x42, 0x42, 0x42, 0x3C };
byte D[] = { 0x78, 0x7C, 0x6E, 0x66, 0x66, 0x6E, 0x7C, 0x78 };
byte E[] = { 0x7C, 0x40, 0x40, 0x78, 0x78, 0x40, 0x40, 0x7C };
byte M[] = { 0x66, 0x7E, 0x5A, 0x42, 0x42, 0x42, 0x42, 0x42 };
byte Y[] = { 0xC3, 0xC3, 0x66, 0x3C, 0x18, 0x18, 0x18, 0x18 };
//ESTE ARRAY ES PARA LIMPIAR MATRIZ
byte sp[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
//SE INICIALIZA LOS PINES COMO SALIDA
void setup() {
    for (int j=2; j<19; j++)
        pinMode(j, OUTPUT);
    Serial.begin(9600);
}
//FUNCION PARA IMPRIMIR LETRA EN MATRIZ DE LEDS
void SetChar(char p) {
    Clear(); //FUNCION PARA LIMPIAR MATRIZ
    for (int fil = 0; fil <8 ; fil++) {
        digitalWrite( fil + 10 , HIGH );
        byte F = Selecciona( p, fil );
        for (int col =7; col >= 0 ; col--) {
            digitalWrite(8-col, HIGH);
            bool b = GetBit(F, col);
            if (b)
                digitalWrite( 9 - col ,LOW );
            else
                digitalWrite( 9 - col ,HIGH );
        }
        digitalWrite( fil + 10 , LOW );
    }
}

```

```

}

//FUNCIÓN PARA CONVETIR HEXADECIMAL A BINARIO
bool GetBit( byte N, int pos) {
    int b = N >> pos ;
    b = b & 1 ;
    return b ;
}

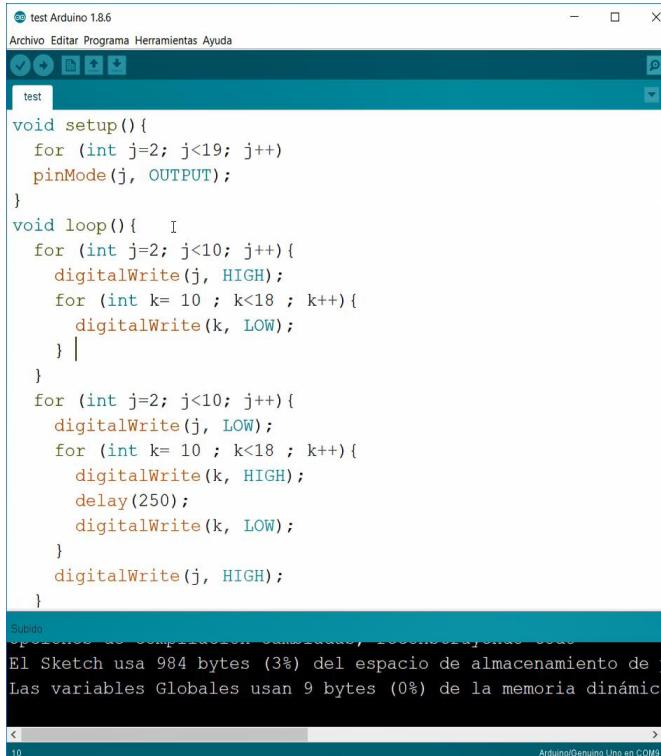
//FUNCIÓN PARA LIMPIAR MATRIZ-TODOS LOS LEDS APAGADOS
void Clear() {
    for (int j=2; j<10; j++)
        digitalWrite(j, HIGH);
    for (int k= 10 ; k<18 ; k++)
        digitalWrite(k, LOW);
}

//SELECCIONA FILA DE ARRAY DE LETRA A IMPIRMIR
byte Selecciona( char c, byte fil) {
    if ( c == 'K') return(K[fil]) ;
    if ( c == 'O') return( O[fil]) ;
    if ( c == 'D') return( D[fil]);
    if (c == 'E') return( E[fil]);
    if (c == 'M') return( M[fil]);
    if (c == 'Y') return( Y[fil]);
    if (c == ' ') return( sp[fil]);
}

void loop(){
    String s = "KODEMY " ; //PALABRA A IMPRIMIR
    int l = s.length(); // CALCULO DE LONGITUD DE PALABRA
    for ( int n = 0; n< l; n++ ) { //RECORRE PALABRA E IMPRIME EN MATRIZ
        long t = millis();
        char c = s[n]; //RECORRE PALABRA Y OBTIENE LETRA POR LETRA
        while ( millis()< t + 800)
            SetChar(c); //LLAMADA A FUNCIÓN QUE IMPRIME EN MATRIZ
    }
}

```

RESULTADO



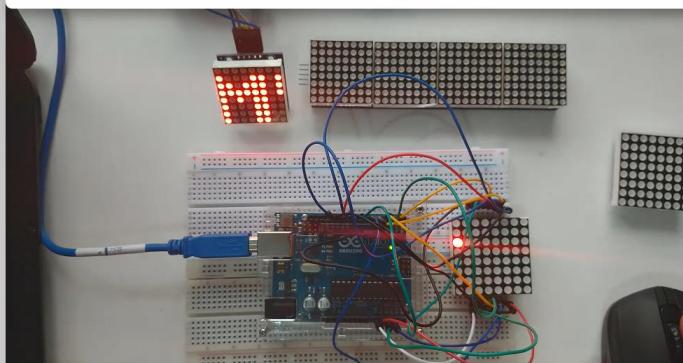
```

test: Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
test
void setup(){
    for (int j=2; j<19; j++)
        pinMode(j, OUTPUT);
}
void loop(){ I
    for (int j=2; j<10; j++){
        digitalWrite(j, HIGH);
        for (int k= 10 ; k<18 ; k++){
            digitalWrite(k, LOW);
        }
    }
    for (int j=2; j<10; j++){
        digitalWrite(j, LOW);
        for (int k= 10 ; k<18 ; k++){
            digitalWrite(k, HIGH);
            delay(250);
            digitalWrite(k, LOW);
        }
        digitalWrite(j, HIGH);
    }
}

```

Subido
Procesos de compilación completados, 1000ms por cada uno.
El Sketch usa 984 bytes (3%) del espacio de almacenamiento de memoria.
Las variables Globales usan 9 bytes (0%) de la memoria dinámica.

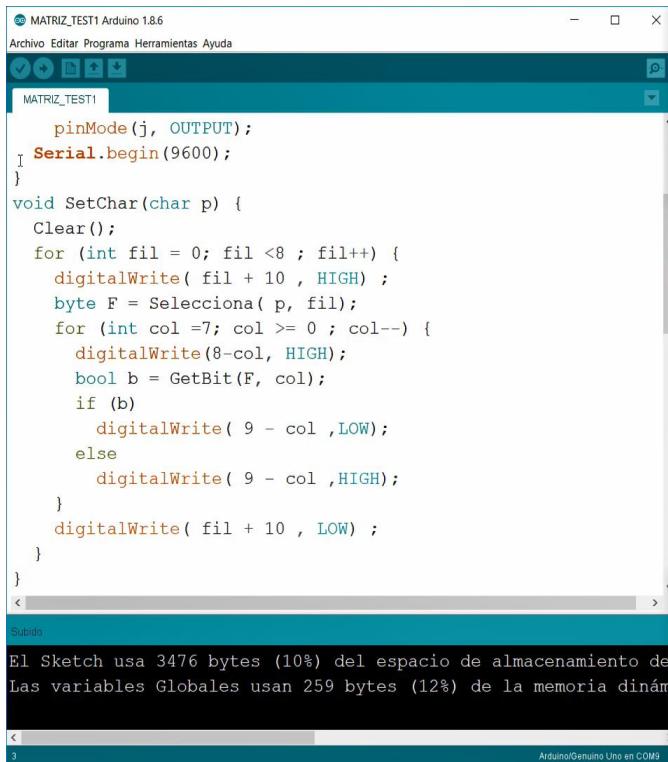
CURSO ARDUINO



MATRIZ DE LEDS

8X8

Test de LEDs



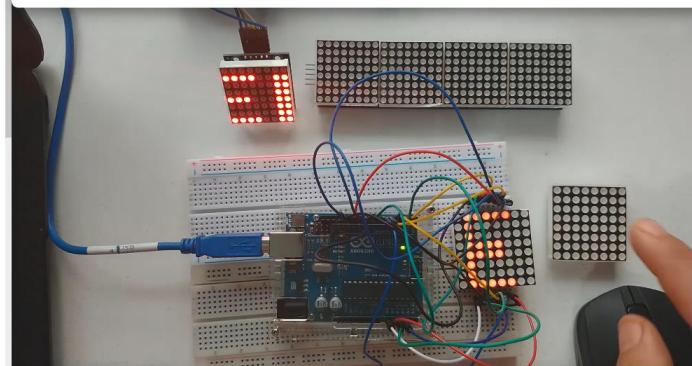
```

MATRIZ_TEST1 Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
MATRIZ_TEST1
pinMode(j, OUTPUT);
Serial.begin(9600);
}
void SetChar(char p) {
Clear();
for (int fil = 0; fil <8 ; fil++) {
digitalWrite( fil + 10 , HIGH) ;
byte F = Selecciona( p, fil);
for (int col =7; col >= 0 ; col--) {
digitalWrite(8-col, HIGH);
bool b = GetBit(F, col);
if (b)
digitalWrite( 9 - col ,LOW);
else
digitalWrite( 9 - col ,HIGH);
}
digitalWrite( fil + 10 , LOW) ;
}
}
Subido
El Sketch usa 3476 bytes (10%) del espacio de almacenamiento de
Las variables Globales usan 259 bytes (12%) de la memoria dinámica
Arduino/Genuino Uno en COM3

```

Se muestra la letra E

CURSO ARDUINO



MATRIZ DE LEDS 8X8

SENSOR INCLINACIÓN (TILT SWITCH)

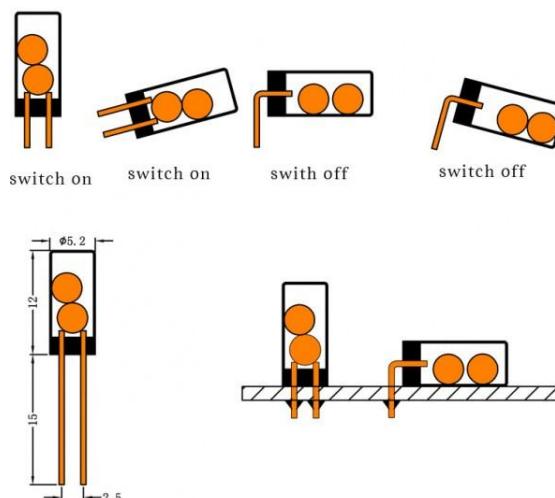


MARCO TEÓRICO

Sensor de inclinación o también conocido como Tilt Switch, en inglés Tilt es inclinar o ladear y se refiere a inclinar el plano de referencia sobre el que se asienta nuestro sistema de coordenadas.

Son sensores de unos pocos milímetros de longitud, que llevan en su interior una o dos pequeñas bolas conductoras, capaces de cerrar el circuito con los pinos metálicos inferiores del cilindro.

Cuando hacen contacto permiten el paso de la corriente y cierran el contacto exactamente igual que si fueran un interruptor (Y de hecho se manejan igual) pero que a partir de un cierto Ángulo de inclinación dejan de hacer contacto y abren el contacto.



No es una tecnología reciente, pues llevan en uso mucho tiempo, pero antes se hacían con una gotita de mercurio líquido, que es conductor y que al desplazarse por el interior del cilindro acababa por cerrar el circuito entre un par de contactos en el fondo.

También existe el sensor de inclinación de mercurio que tiene exactamente el mismo funcionamiento.



Fuente de información:

<https://www.prometec.net/tilt-switch/>

https://http2.mlstatic.com/sensor-de-mercurio-para-alarmas-D_NQ_NP_156115-MLA25155659893_112016-Q.jpg

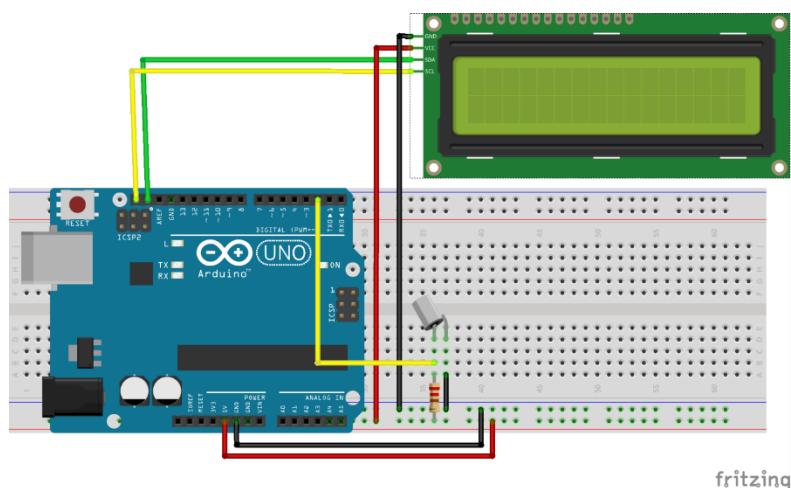
DESCRIPCIÓN DE EJEMPLO

El sensor de inclinación se conecta mediante una resistencia a VCC y al pin digital 2 del Arduino, activado en modo **INPUT_PULLUP** (Entrada con acoplamiento positivo) para realizar una lectura del sensor, y se imprime en pantalla LCD si el resultado del sensor está en horizontal o en vertical.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Tilt Switch o sensor de inclinación
- Pantalla LCD 16x02 con I2C
- Una resistencia 220 ohm

DIAGRAMA DE CONEXIÓN

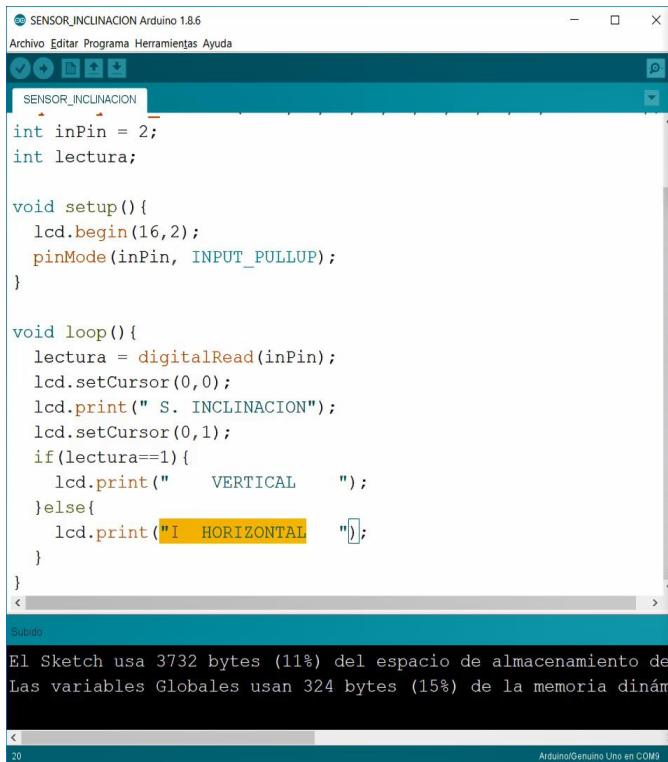


CÓDIGO FUENTE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
int inPin = 2;
int lectura;
//SE CONFIGURA E INICIALIZA LCD
void setup(){
  lcd.begin(16,2);
  pinMode(inPin, INPUT_PULLUP);
}

void loop(){
  lectura = digitalRead(inPin); //REALIZA LECTURA DIGITAL DEL SENSOR
  lcd.setCursor(0,0);
  lcd.print(" S. INCLINACION");
  lcd.setCursor(0,1);
  //DE ACUERDO AL VALOR SE IMPRIME MSJ EN LCD
  if(lectura==1){
    lcd.print("      VERTICAL      ");
  }else{
    lcd.print("      HORIZONTAL      ");
  }
}
```

RESULTADO



```

// SENSOR_INCLINACION Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
SENSOR_INCLINACION
int inPin = 2;
int lectura;

void setup(){
  lcd.begin(16,2);
  pinMode(inPin, INPUT_PULLUP);
}

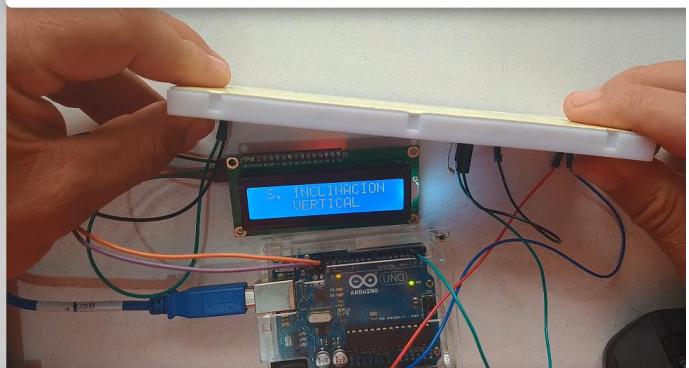
void loop(){
  lectura = digitalRead(inPin);
  lcd.setCursor(0,0);
  lcd.print(" S. INCLINACION");
  lcd.setCursor(0,1);
  if(lectura==1){
    lcd.print("      VERTICAL      ");
  }else{
    lcd.print("I      HORIZONTAL      ");
  }
}

```

Subido
El Sketch usa 3732 bytes (11%) del espacio de almacenamiento de
Las variables Globales usan 324 bytes (15%) de la memoria dinámica

20 Arduino/Genuino Uno en COM9

CURSO ARDUINO



TILT SHIFT SENSOR INCLINACIÓN

MATRIZ DE BOTONES 4X4 (TECLADO)



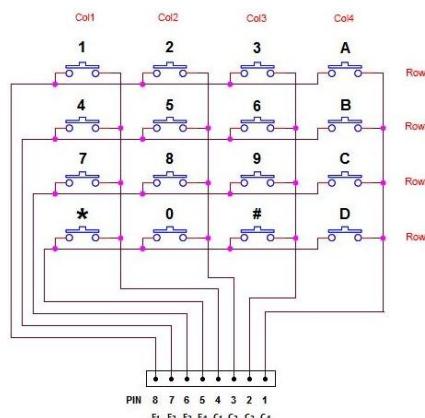
MARCO TEÓRICO

Un teclado no es más que una colección de botones, a cada uno de los cuales le asignamos un símbolo o una función determinada.

Para que nuestro Arduino pueda saber que tecla se pulsa, basta con poner tensión en las filas de forma secuencial y luego leer las columnas para ver cuál de ellas tiene HIGH. Los teclados matriciales usan una combinación de filas y columnas para conocer el estado de los botones. Cada tecla es un pulsador conectado a una fila y a una columna. Cuando se pulsa una de las teclas, se cierra una conexión única entre una fila y una columna.

Por ejemplo, ponemos HIGH en la primera fila y después leemos sucesivamente los hilos correspondientes a las columnas. Si ninguno está en HIGH es que no se ha pulsado ninguna tecla de la primera fila.

De este modo, para leer un teclado matricial de 4x4 necesitamos 8 pines.



Fuente de información:

<https://www.prometecc.net/teclados-matriciales/>

<https://www.electronicoscaldas.com/switches-teclados/129-teclado-4x4-matricial-membrana-27899.html>

DESCRIPCIÓN DE EJEMPLO

Para este ejemplo se conecta las columnas a los pines 2, 3, 4 y 5 y la filas a los pines digitales 6, 7, 8 y 9. Se utilizar la librería **Keypad.h** para poder controlar de una forma sencilla el teclado, permite definir el array de símbolos, configura los pines de filas y columnas, y mediante la función **keypad.getKey()**; se obtiene el valor del símbolo correspondiente al botón que se oprime y lo muestra en la pantalla LCD.

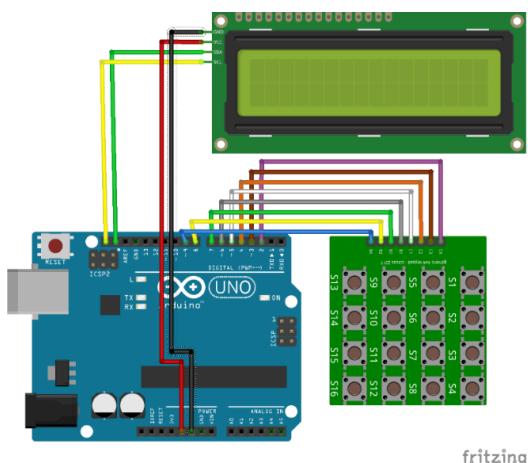
En un segundo ejemplo se realiza mediante la misma conexión del teclado, la simulación de una calculadora con las funciones básicas de aritmética como son la suma, resta, multiplicación y división. Todo el proceso de la operación se muestra en la Pantalla LCD.

Librería: <https://playground.arduino.cc/uploads/Code/Keypad/index.zip>

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Matriz de Botones 4x4 / teclado / Keyboard / teclado Matricial
- Pantalla LCD 16x02 con I2C

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

TEST DE BOTONES

```
// SE INCLUYE LIBRERÍAS DE LCD
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
//SE INCLUYE LIBRERÍA DE TECLADO
#include <Keypad.h>
//SE INICIA OBJETO LCD
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
//SE DECLARA VARIABLE PARA TAMAÑO DE TECLADO
const byte ROWS = 4;// Filas
const byte COLS = 4;//Columnas
//SE DECLARA ARRAY CON SIMBOLOS CORRESPONDIENTES
char keys[ROWS][COLS] = {//Simbolos
    {'1','2','3','+'},
    {'4','5','6','-'},
    {'7','8','9','*'},
    {'=','0','.','/'}
};
// SE DECLARA PINES DIGITALES A UTILIZAR
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};
//SE CREA Y CONFIGURA OBJETO KEYPAD
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
//SE INICIALIZAR LCD Y MONITOR SERIE
void setup() {
  lcd.begin(16,2);
  Serial.begin(9600);
```

```

}
void loop() {
    //SE IMPRIME TITULO EN LCD
    lcd.setCursor(0,0);
    lcd.print("KEYBOARD KODEMY");
    // SE OBTIENE EL VALOR DE LA TECLA OPRIMIDA
    char key = keypad.getKey();
    if (key != NO_KEY){//SI ES DIFERENTE DE NULO
        //IMPRIME EN MONITOR SERIE Y EN LCD EL VALOR DE LA TECLA
        Serial.println(key);
        lcd.setCursor(0,1);
        lcd.print(key);
    }
}

```

CALCULADORA

```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
const byte ROWS = 4;// Filas
const byte COLS = 4;//Columnas
char keys[ROWS][COLS] = {//Simbolos
    {'1','2','3','+'},
    {'4','5','6','-'},
    {'7','8','9','*'},
    {'X','0','=','/'}
};
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
boolean valorActual = false;
boolean siguiente = false;
boolean final = false;
String num1, num2,num3;
int total;
int movimiento;
char op;
void setup(){
    lcd.begin(16,2);
    lcd.setCursor(0,0);
    lcd.print(" CALCULADORA");
    lcd.setCursor(0,1);
    lcd.print(" KODEMY");
    delay(2500);
    lcd.clear();
}
void loop(){
    char key = keypad.getKey();
    if (key != NO_KEY &&
    (key=='1'||key=='2'||key=='3'||key=='4'||key=='5'||key=='6'||key=='7'||key=='8'||key=='9'||key=='0')) {
        if (valorActual != true){
            num1 = num1 + key;
            int numLength = num1.length();
            movimiento = numLength;
            lcd.setCursor(0, 0);
            lcd.print(num1);
        }else {
            num2 = num2 + key;
            int numLength = num2.length();
            lcd.setCursor(movimiento+1, 0);
            lcd.print(num2);
            final = true;
        }
    }else if (valorActual == false && key != NO_KEY && (key == '/' || key == '*' || key == '-' || key ==
    '+')){
        if (valorActual == false){
            valorActual = true;
            op = key;
            lcd.setCursor(movimiento,0);
            lcd.print(op);
        }
    }
}

```

```

}else if (final == true && key != NO_KEY && key == '=') {
    if (op == '+') {
        total = num1.toInt() + num2.toInt();
    }else if (op == '-') {
        total = num1.toInt() - num2.toInt();
    }else if (op == '*') {
        total = num1.toInt() * num2.toInt();
    }else if (op == '/') {
        if(num2.toInt()==0) {
            total = ' ';
        }else{
            total = num1.toInt() / num2.toInt();
        }
    }
    num3=String(total);
    lcd.clear();
    lcd.setCursor(15,0);
    lcd.autoscroll();
    if(total==' '){
        lcd.print("Syntax Error");
    }else if(num3.length()>4){
        lcd.print("No hay memoria");
    }else{
        lcd.print(total);
    }
    lcd.noAutoscroll();
}else if (key != NO_KEY && key == 'X') {
    lcd.clear();
    valorActual = false;
    final = false;
    num1 = "";
    num2 = "";
    total = 0;
    op = ' ';
}
}

```

RESULTADO

```
keyboard Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda

keyboards
char keys[ROWS][COLS] = { //Simbolos
    {'1','2','3','+'},
    {'4','5','6','-'},
    {'7','8','9','*'},
    {'=','0','.','/'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup() {
    lcd.begin(16,2);
    Serial.begin(9600);
}

void loop() {
    lcd.setCursor(0,0);
    lcd.print("KEYBOARD KODEM!");
    char key = keypad.getKey();
    if^(key != NO_KEY){
        Serial.println(key);
        lcd.setCursor(0,1);
        lcd.print(key);
    }
}

Subido

El Sketch usa 5816 bytes (18%) del espacio de almacenamiento de programa. El máximo es
Las variables Globales usan 612 bytes (29%) de la memoria dinámica, dejando 1436 bytes
```

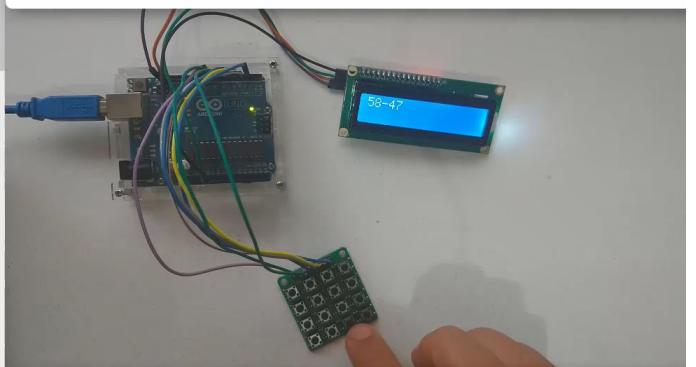


TECLADO/KEYBOARD MATRIZ DE BOTONES

Test de Botones

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** calculadora Arduino 1.8.6
- Menu Bar:** Archivo Editar Programa Herramientas Ayuda
- Search Bar:** calculadora
- Code Area:** The main code area contains C++ code for a keypad-based calculator using an LCD display. The code includes definitions for the LCD pins (ROWS=4, COLS=4), key symbols, and row/column pins. It also defines variables for keypad, row/col pins, and various integers (total, movimiento, op). The setup function initializes the LCD and prints " CALCULADORA".
- Status Bar:** Shows memory usage: El Sketch usa 7936 bytes (24%) del espacio de almacenamiento de programa. El máximo es 32256 bytes. Las variables Globales usan 506 bytes (24%) de la memoria dinámica, dejando 1542 bytes para la pila.
- Bottom Status:** Arduino/Genuino Uno en COM9



TECLADO/KEYBOARD MATRIZ DE BOTONES

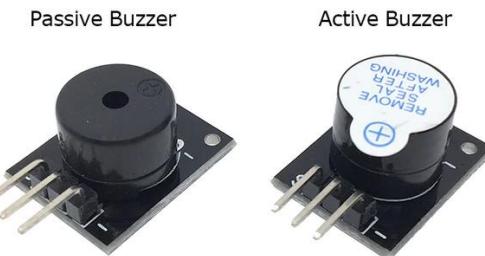
Calculadora con el Teclado

BUZZERS ACTIVO/PASIVO

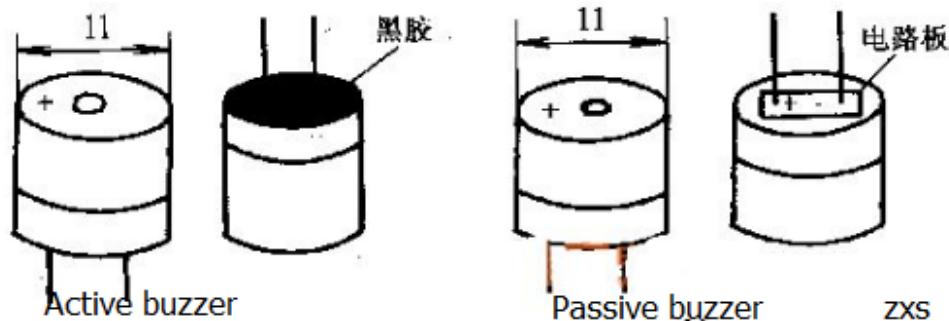


MARCO TEÓRICO

Los Buzzer se dividen en dos tipos de activo y pasivo, activo se refiere a su multivibrador interno y otras estructuras, el externo solo necesita proporcionar voltaje, puede enviar un sonido de frecuencia fija. Pasivo significa que no hay una fuente de oscilador interno, la necesidad de circuitos externos para proporcionar una cierta frecuencia de la señal de accionamiento.



Buzzer activo y pasivo, a simple vista parecen ser el mismo componente, sin embargo, el Buzzer activo tiene una calcomanía color blanco y por debajo de los pines se ve de color negro la base, el Buzzer pasivo no tiene la calcomanía y por debajo se ven unas líneas de circuito interno.



La diferencia entre el buzzer activo y pasivo. El buzzer activo generará un tono usando un oscilador interno, por lo que todo lo que se necesita es un voltaje de CC. Un buzzer pasivo requiere una señal de CA para hacer un sonido. Es como un altavoz electromagnético, donde una señal de entrada cambiante produce el sonido, en lugar de producir un tono automáticamente.

En cuanto a los comandos, si desea controlar el tono, necesitará un buzzer pasivo. PWM en el Arduino se puede usar para controlar el tono y el volumen al mismo tiempo.

Fuente de información:
<http://www.zxs1688.com/info/difference-between-active-buzzer-passive-buzzer-22548813.html>

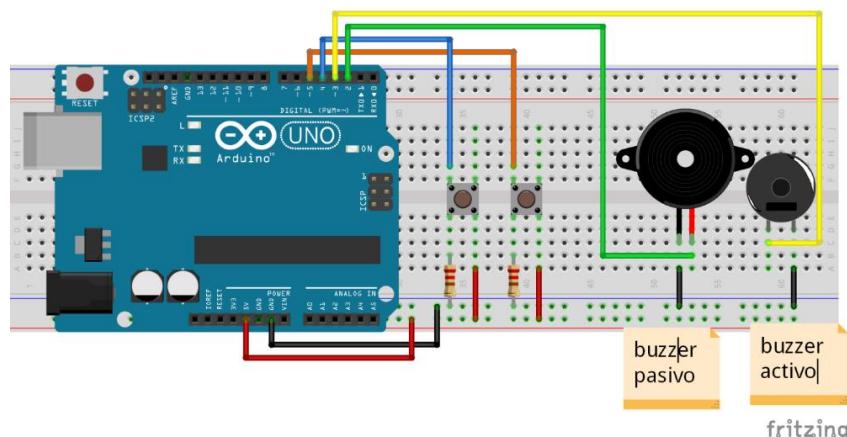
DESCRIPCIÓN DE EJEMPLO

Para este ejemplo se conectarán 2 buzzers, uno activo y otro pasivo, que serán activados mediante 2 Push Button. El buzzer activo solo emitirá un pitido a la misma frecuencia sin variación alguna por algunos segundos, en el segundo caso el buzzer pasivo emitirá tonos a diferentes frecuencias que obtendrá como resultado una melodía.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Un Buzzer Activo
- Un Buzzer Pasivo
- Dos Push Button
- Dos resistencias de 220 ohm

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

```

int speakerPin = 2; //PIN DE BUZZER
int length = 15; // LONGITUD
//NOTAS DE CANCIÓN
char notes[] = "ccggaagffeeddc ";
int beats[] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4 };
int tempo = 300;
//EJEMPLO DE ACUERDO A LA PAGINA DE ARDUINO
//EJECUTAR TONO
void playTone(int tone, int duration) {
    for (long i = 0; i < duration * 1000L; i += tone * 2) {
        digitalWrite(speakerPin, HIGH);
        delayMicroseconds(tone);
        digitalWrite(speakerPin, LOW);
        delayMicroseconds(tone);
    }
}
//EJECUTAR NOTA
void playNote(char note, int duration) {
    char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'c' };
    int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };
    for (int i = 0; i < 8; i++) {
        if (names[i] == note) {
            playTone(tones[i], duration);
        }
    }
}

```

```

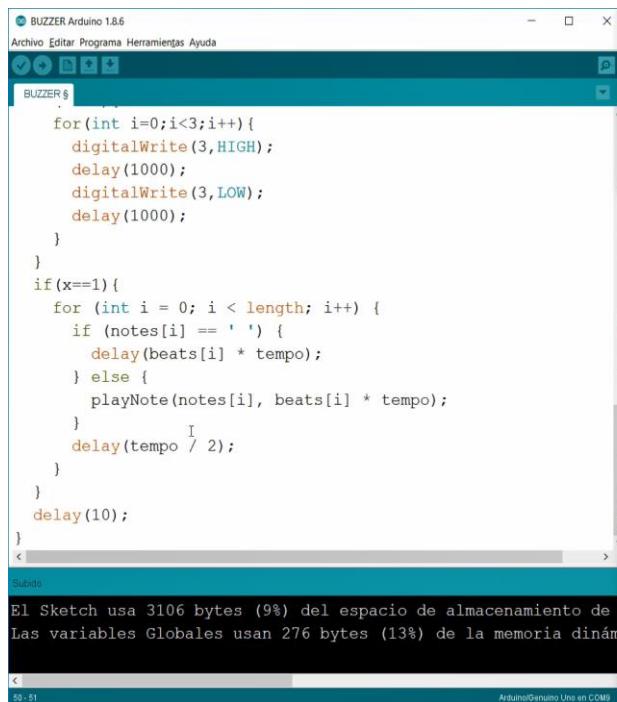
        }
    }

void setup() {
    pinMode(2, OUTPUT); //BUZZER PASIVO
    pinMode(3, OUTPUT); //BUZZER ACTIVO
    pinMode(5, INPUT); //BOTON 1
    pinMode(4, INPUT); //BOTON 2
    Serial.begin(9600);
}

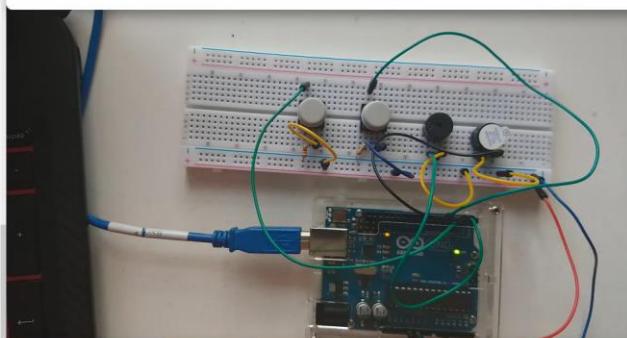
void loop(){
    int v=digitalRead(5); //LEER BOTON 1
    int x=digitalRead(4); // LEER BOTON 2
    //SE IMPRIME VALORES EN MONITOR SERIE
    Serial.print("Activo: ");
    Serial.print(v);
    Serial.print(" Pasivo: ");
    Serial.println(x);
    //SI V==1 ENTONCES REPITE 3 VECES PITIDOS BUZZER ACTIVO
    if(v==1){
        for(int i=0;i<3;i++){
            digitalWrite(3,HIGH);
            delay(1000);
            digitalWrite(3,LOW);
            delay(1000);
        }
    }
    //SECUENCIA DE INSTRUCCIONES PARA BUZZER PASIVO
    //MELODIA
    if(x==1){
        for (int i = 0; i < length; i++) {
            if (notes[i] == ' ') {
                delay(beats[i] * tempo);
            } else {
                playNote(notes[i], beats[i] * tempo);
            }
            delay(tempo / 2);
        }
    }
    delay(10);
}
}

```

RESULTADO

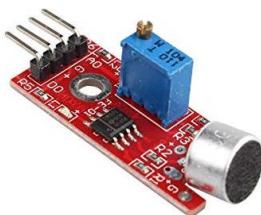


CURSO ARDUINO



**BUZZERS
ACTIVO/PASIVO**

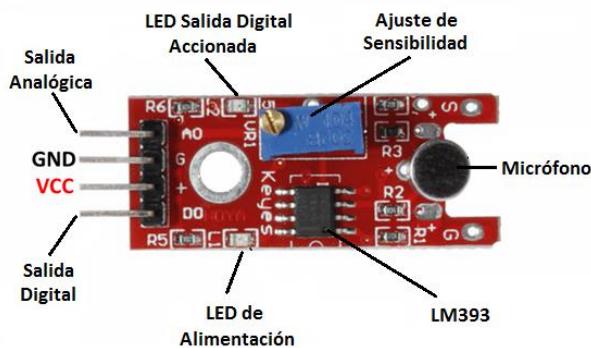
SENSOR ELECTRET



MARCO TEÓRICO

Este módulo permite la conexión de un micrófono Electret a los pines digitales de un microcontrolador. El nivel de detección acústico puede ser ajustado mediante un potenciómetro. Toda la electrónica para la detección de sonidos ya se encuentra incluida dentro de este práctico módulo. El sensor de sonido no es apto para la grabación de voz o audio, pues la señal de salida no es analógica, la señal del micrófono es pasada a un comparador analógico que envía un pulso digital cuando la amplitud de la señal recogida por el micrófono supera el valor seleccionado mediante el potenciómetro.

Partes del sensor



Fuente de información:

<https://www.geekfactory.mx/tienda/sensores/sensor-de-sonido-microfono-electret/>
<https://www.prometec.net/wp-content/uploads/2016/04/partes-ky-038.png>

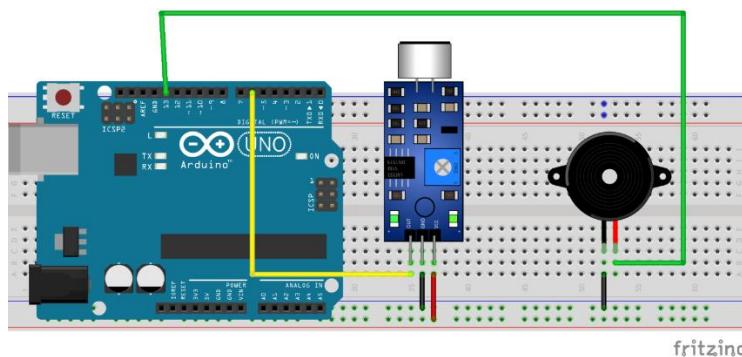
DESCRIPCIÓN DE EJEMPLO

Se conectar el sensor Electret al pin digital 6 del Arduino en modo **INPUT_PULLUP**, se ajusta la sensibilidad del sensor mediante el potenciómetro, se realizar la lectura con la función **digitalRead**, si detecta algún ruido fuera del rango, cualquiera que este sea, activara el buzzer por 1 segundo, si se vuelve a detectar un ruido se desactivara el buzzer. No requiere ninguna librería

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Sensor Electret / Audio
- Un Buzzer Activo

DIAGRAMA DE CONEXIÓN

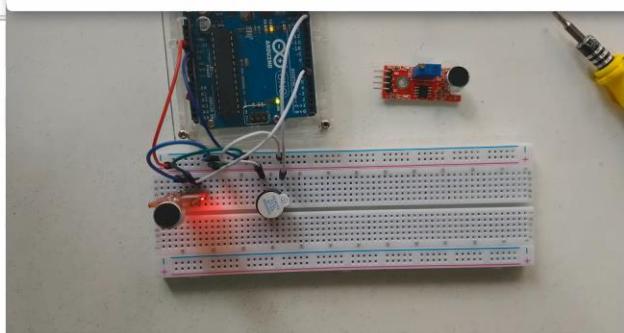


CÓDIGO FUENTE

```
//SE CREAN VARIABLES DE BUZZER Y SENSOR ELECTRET
int buzzer = 13 ;
int sensor = 6 ;
bool estado = false ;
//SE CONFIGURA PIN COMO SALIDA Y SENSOR ELECTRET COMO INPUT_PULLUP
void setup(){
  pinMode( buzzer, OUTPUT) ;
  pinMode( sensor , INPUT_PULLUP) ;
  digitalWrite(buzzer , LOW) ; //BUZZER INICIA APAGADO
  Serial.begin(9600);
}
void loop(){
  bool valor = digitalRead(sensor) ; //SE LEE VALOR DEL SENSOR ELECTRET
  Serial.println(valor); //SE IMPRIME VALOR DE ELECTRET EN MONITOR SERIE
  //COMO ES UNA LECTURA DIGITAL, SI ES TRUE O 1 SE ACTIVA EL BUZZER O DESACTIVA
  if ( valor == true ) {
    estado = ! estado ;
    digitalWrite(buzzer, estado) ;
    delay (1000);
  }
}
```

RESULTADO

CURSO ARDUINO



SENSOR ELECTRET/ MICRÓFONO KY-038

JOSTICK ANALÓGICO (2 EJES CON BOTÓN)



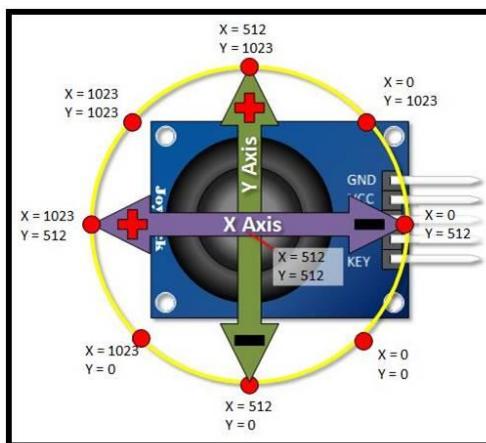
MARCO TEÓRICO

Un joystick es un elemento de entrada para programas digitales. Esté en particular está construido mediante un conjunto de resistencias variables. También es una herramienta muy útil, que nos ayuda a dar dirección a lo que queremos, por ejemplo, a un carro por control remoto, también están presentes en los controles de videojuegos. El funcionamiento está basado en el movimiento en dos dimensiones de una palanca, este movimiento es capturado por dos potenciómetros (uno para cada movimiento), de este modo se entiende que para cada movimiento en cada dirección será regulado un potenciómetro.

Con este módulo joystick se puede utilizar el cambio de valor resistivo para tomar lectura con dos entradas analógicas en el Arduino. Este módulo de joystick cuenta con cinco pines:

- GND: Pin conectado a tierra.
- +5V: pin de alimentación(5v).
- VRx: pin de lectura de potenciómetro para el eje X.
- VRy: pin de lectura de potenciómetro para el eje Y.
- SW: es un pin adicional que se utiliza para un Push Button en la parte inferior.

Los valores que oscila tanto en el eje x y en el eje y van de 0 a 1023, como un potenciómetro, en el siguiente diagrama se muestra el valor de ambos potenciómetros de acuerdo a la posición del joystick.



Fuente de información:

<https://hetpro-store.com/TUTORIALES/joystick-analogico-programado-con-arduino/>

<https://www.luisllamas.es/arduino-joystick/>

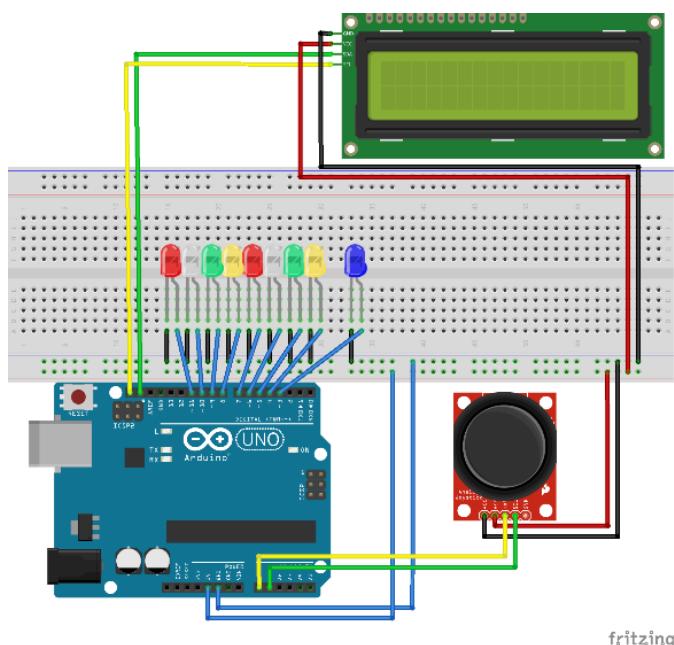
DESCRIPCIÓN DE EJEMPLO

El ejemplo consiste en conectar al Arduino el Joystick, VRx y VRy a los pines analógicos correspondientes, realizar una lectura con la función analogRead y de acuerdo a los 2 valores, tanto del potenciómetro del eje x, como el potenciómetro del y, se imprime en pantalla LCD la posición CENTRO, ABAJO, ABAJO DERECHA, ABAJO IZQUIERDA, DERECHA, IZQUIERDA, ARRIBA, ARRIBA IZQUIERDA Y ARRIBA DERECHA. De acuerdo a la posición además de enciende un LED que representa que el valor de los potenciómetros está dentro del rango de valores de la posición teórica del joystick.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Joystick analógico de 2 ejes y un botón
- Pantalla LCD 16x02 con I2C
- 9 LED de colores

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

```
//VARIABLES X,Y Y BOTON
int x1;
int y1;
int q1;
//LIBRERIAS PARA LCD
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

void setup(){
  lcd.begin(16,2); // SE INICIALIZA PANTALLA LCD
```

```

//SE CONFIGURA PINES DE LEDS
pinMode(3, OUTPUT);
pinMode(4, OUTPUT);
pinMode(5, OUTPUT);
pinMode(6, OUTPUT);
pinMode(7, OUTPUT);
pinMode(8, OUTPUT);
pinMode(9, OUTPUT);
pinMode(10, OUTPUT);
pinMode(11, OUTPUT);
Serial.begin(9600);

}

void loop(){
x1 = analogRead(A0); //SE LEE VALOR DEL EJE X
y1 = analogRead(A1); //SE LEE VALOR DEL EJE Y
//SE IMPRIME VALORES EN MONITOR SERIE
Serial.print("X1: ");
Serial.print(x1);
Serial.print(" Y1: ");
Serial.print(y1);
//SE IMPRIME TITULO EN LCD
lcd.setCursor(0,0);
lcd.print(" JOYSTICK");
lcd.setCursor(0,1);
//DE ACUERDO A LA POSICIÓN DEL JOSTICK DENTRO DEL X O Y
//SE IMPRIME LA UBICACIÓN Y SE PRENDE EL LED CORRESPONDIENTE
if (x1>450 && x1 < 550 && y1>450 && y1<550){
    lcd.print("CENTRO ");
    digitalWrite(3,HIGH);
} else if ( x1< 600 &&x1>500&& y1<50){
    lcd.print("ARRIBA ");
    digitalWrite(4,HIGH);
} else if (x1>800 && y1<100){
    lcd.print("DERECHA ARRIBA ");
    digitalWrite(5,HIGH);
} else if ( x1>800 && y1<600){
    lcd.print("DERECHA ");
    digitalWrite(6,HIGH);
} else if (x1>800 && y1>800){
    lcd.print("DERECHA ABAJO ");
    digitalWrite(7,HIGH);
} else if (x1>450 && y1>800){
    lcd.print("ABAJO ");
    digitalWrite(8,HIGH);
} else if (x1< 100 && y1 >800) {
    lcd.print("IZQUIERDA ABAJO ");
    digitalWrite(9,HIGH);
} else if (x1<100 && y1> 450){
    lcd.print("IZQUIERDA ");
    digitalWrite(10,HIGH);
} else if (x1 <100 && y1 <100) {
    lcd.print("IZQUIERDA ARRIBA ");
    digitalWrite(11,HIGH);
}
delay(1000);
//SE APAGA TODOS LOS LEDS, PARA LIMPIAR POSICIÓN
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,LOW);
digitalWrite(6,LOW);
digitalWrite(7,LOW);
digitalWrite(8,LOW);
digitalWrite(9,LOW);
digitalWrite(10,LOW);
digitalWrite(11,LOW);
}
}

```

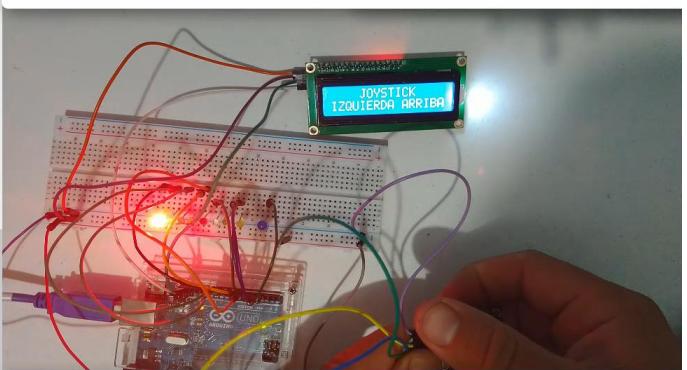
RESULTADO

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** JOYSTICK Arduino 1.8.6
- Menu Bar:** Archivo Editar Programa Herramientas Ayuda
- Toolbar:** Includes icons for Open, Save, Print, and others.
- Sketch Area:** Displays the following C++ code for a joystick control system:

```
JOYSTICK $
  if(y1<100 && y1>0) {
    lcd.print("ARRIBA ");
    digitalWrite(3,HIGH);
  }
  else if (x1>800 && y1>800) {
    lcd.print("DERECHA ABAJO ");
    digitalWrite(7,HIGH);
  }
  else if (x1>450 && y1>800) {
    lcd.print("ABAJO ");
    digitalWrite(8,HIGH);
  }
  else if (x1< 100 && y1 >800) {
    lcd.print("IZQUIERDA ABAJO ");
    digitalWrite(9,HIGH);
  } else if (x1<100 && y1> 450) {
    lcd.print("IZQUIERDA ");
    digitalWrite(10,HIGH);
  }
  else if (x1 <100 && y1 <100) {
    lcd.print("IZQUIERDA ARRIBA ");
    digitalWrite(11,HIGH);
  }
  delay(1000);
  digitalWrite(3,LOW);
  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
  digitalWrite(8,LOW);
  digitalWrite(9,LOW);
  digitalWrite(10,LOW);
  digitalWrite(11,LOW);
}

Subido
```
- Status Bar:** El Sketch usa 6078 bytes (18%) del espacio de almacenamiento de programa. El máximo es 32256 bytes. Las variables Globales usan 677 bytes (33%) de la memoria dinámica, dejando 1371 bytes libres.
- Bottom Navigation:** Includes '<' and '>' navigation arrows.



JOYSTICK ANALÓGICO 2 EJES

RELEVADOR / RELAY / RELÉ



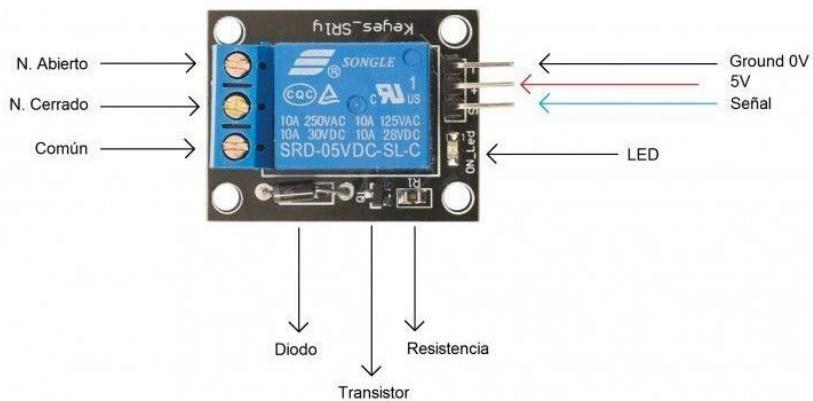
MARCO TEÓRICO

Módulo de relevadores (relés) para conmutación de cargas de potencia, ideal para cargas externas tales como bombillas, motores etc. Los contactos de los relevadores están diseñados para conmutar hasta 10 A y 250 VAC (5 VCD), aunque recomendamos dejar un margen hacia abajo de estos límites. Las entradas de control se encuentran protegida con un diodo para minimizar el ruido percibido por el circuito de control mientras se realiza la conmutación de la carga. La señal de control puede provenir de cualquier circuito de control TTL o CMOS como un microcontrolador.

- Fácil de instalar
- LED indicador de funcionamiento
- El voltaje de la bobina del relé es de 5 VDC
- Led indicador para cada canal (enciende cuando la bobina del relé esta activa)
- Activado mediante corriente el circuito de control debe proveer de 15 a 20 mA
- Puede controlado directamente por circuito lógicos
- Terminales de conexión de tornillo (clemas)

Un opto acoplador, también llamado optoisolador o aislador acoplado ópticamente, es un dispositivo de emisión y recepción que funciona como un interruptor activado mediante la luz emitida por un diodo LED que satura un componente optoelectrónico, normalmente en forma de fototransistor o fototriac.

Partes de Modulo



Fuente de información:

<https://www.carrod.mx/products/modulo-de-reles-1-canales-5-v-con-optoacoplador-generico>
<https://www.geekfactory.mx/tienda/relevadores-y-switches/modulo-de-4-relevadores-con-optoacoplador/>

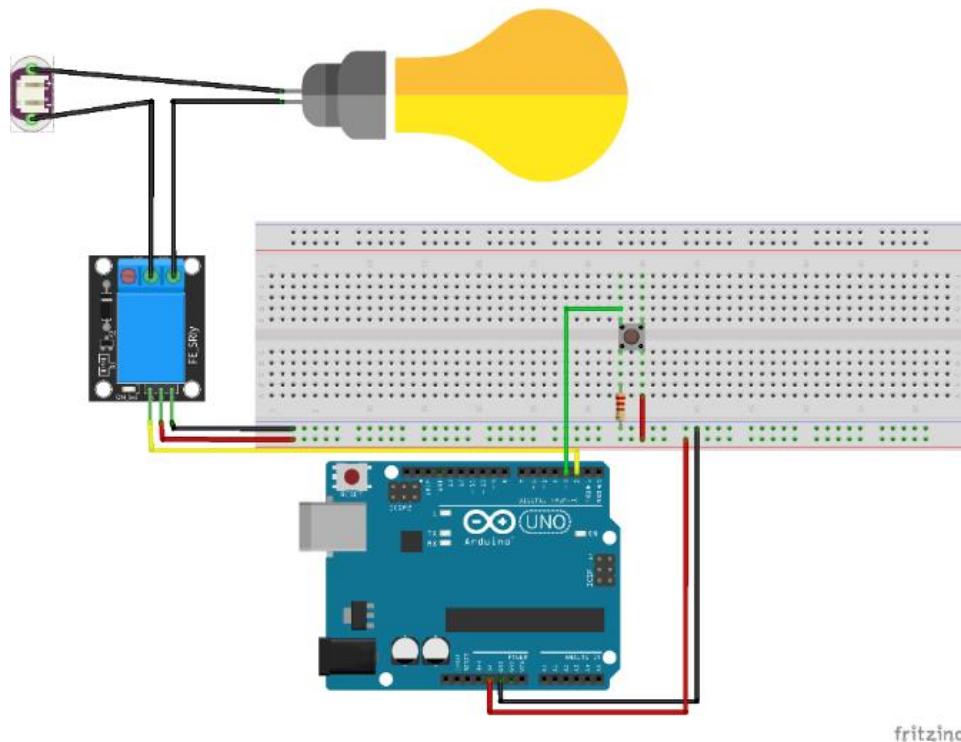
DESCRIPCIÓN DE EJEMPLO

Se conecta el relevador a un circuito abierto compuesto por una clavija con una extensión para encender un foco con voltaje 127v, se activa mediante un Push Button, conectado al pin digital 3 como entrada de datos y se activa el relevador con el pin digital 2 como salida.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Relevador / Relay
- Push Button
- resistencia 220 ohm
- Una extensión, clavija, socket y foco

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```

boolean x=false;
//SE CONFIGURA SALIDA RELE Y ENTRADA PUSH
void setup() {
  pinMode(2,OUTPUT);
  pinMode(3,INPUT);
  Serial.begin(9600);
}

void loop() {
  int valor=digitalRead(3); //SE LEE VALOR DE PUSH 0/1
  Serial.print("Push: "); //SE IMPRIME VALOR DE PUSH
  Serial.println(valor);
  delay(1);
}

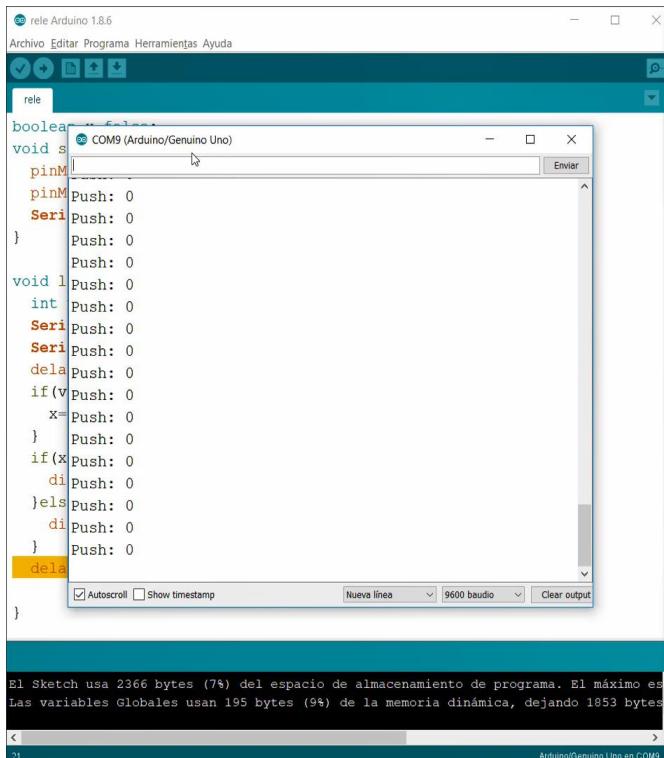
```

```

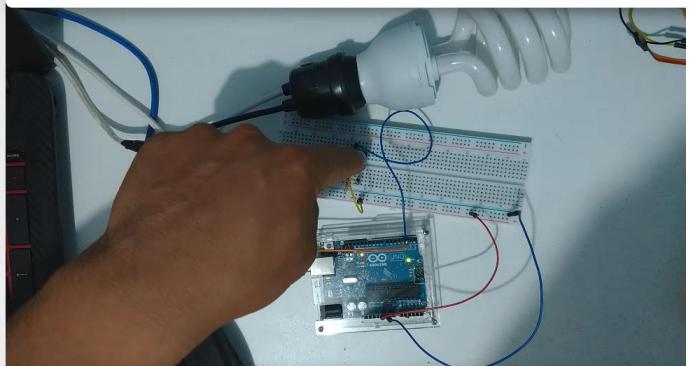
//SI EL VALOR ES 1 SE CAMBIA VARIABLE BOOLEANA
if(valor==1){
    x=!x;
}
//SI ES TRUE SE PRENDE FOCO
//SI ES FALSE SE APAGA FOCO
if(x){
    digitalWrite(2,HIGH);
}else{
    digitalWrite(2,LOW);
}
delay(200); //RETARDO
}

```

RESULTADO

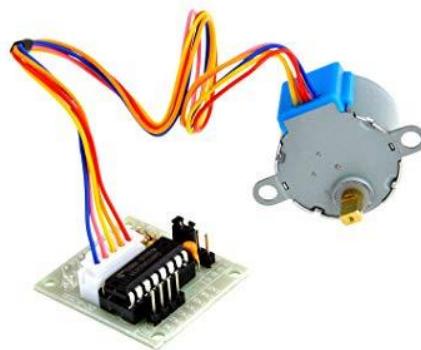


CURSO ARDUINO



RELEVADOR/ RELAY

MOTOR A PASOS 28BYJ-48



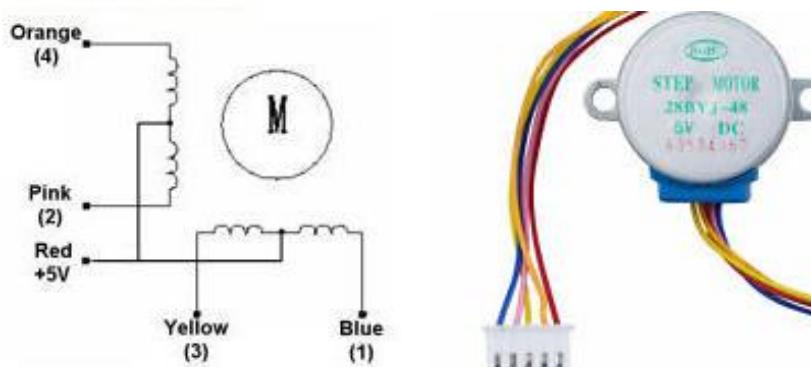
MARCO TEÓRICO

El motor paso a paso unipolar, muy común en el mundo Arduino por su pequeño tamaño y bajo coste, el 28BYJ-48 y el adaptador que suele venir con él, basado en el chip ULN2003A.

Es un motor unipolar con las siguientes características:

- Tensión nominal de entre 5V y 12 V.
- 4 fases.
- Resistencia 50 Ω.
- Par motor de 34 Newton / metro más o menos 0,34 Kg por cm.
- Consumo de unos 55 mA.
- 8 pasos por vuelta.
- Reductora de 1 / 64.

Es decir, que como es de 4 pasos (Steps), u 8 medios pasos (O half Steps) por vuelta y usa una reductora de 1 / 64, necesitamos dar $8 * 64 = 512$ impulsos para completar un giro completo a medios pasos.



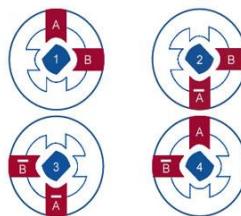
Este es su diagrama de conexión de bobinas, y además marca los colores del cable en función de su conexión interna.

Para controlar el Motor a pasos se utiliza un integrado del tipo ULN2003A que es un array de transistores Darlington, que soporta hasta 500 mA y que ya dispone de un conector para el motor y de unos pines (IN1 – IN4) para conectar a nuestro Arduino. (Reductor 1 a 64).

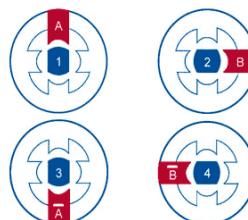


El motor a pasos 28BYJ-48 tiene 3 formas de controlarse mediante la activación del 4 fases o bobina.

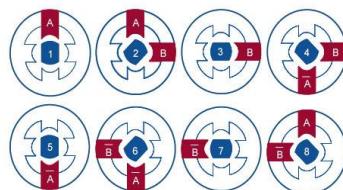
La primera forma, se activa 2 fases al mismo tiempo para dar pasos completos y completar un giro con 4 pasos.



La segunda forma, se activa una fase a la vez para dar cuatro pasos para completar un giro completo.



La tercera forma se activa 1 fase y 2 fases de manera intercalada para dar 8 medios pasos para completar un giro.



Fuente de información:
<http://www.iescamp.es/miarduino/2018/01/29/motor-paso-a-paso-28byj-48/>
<https://www.prometec.net/motor-28byj-48/>

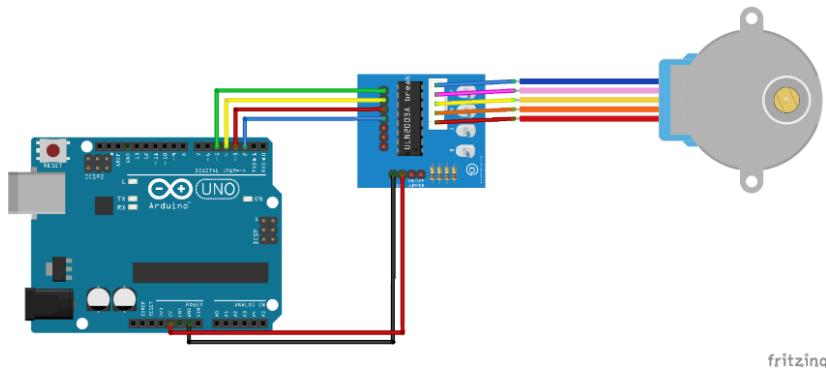
DESCRIPCIÓN DE EJEMPLO

Para este ejemplo se conecta el driver ULN2003A (Pines IN1 al IN4) a los pines digitales del Arduino 2, 3, 4 y 5. El control se realiza sin librería, pero básicamente el funcionamiento trata sobre hacer girar el motor a pasos en sentido del reloj y en sentido inverso, mediante los 8 medios pasos para completar un giro, en total para hacer girar el motor a pasos un giro completo, se requiere aproximadamente 4096 medios pasos.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Motor a pasos 28BYJ 48 con circuito controlador ULN2003A

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```

const int motorPin1 = 5;      // 28BYJ48 In1
const int motorPin2 = 4;      // 28BYJ48 In2
const int motorPin3 = 3;      // 28BYJ48 In3
const int motorPin4 = 2;      // 28BYJ48 In4
int motorSpeed = 1200;        //variable para fijar la velocidad
int stepCounter = 0;          // contador para los pasos
int stepsPerRev = 4096;        // pasos para una vuelta completa
const int numSteps = 8; // 8 MEDIOS PASOS
//SE CONFIGURA LOS MEDIOS PASOS, PARA LAS BOBINAS
const int stepsLookup[8] = { B1000, B1100, B0100, B0110, B0010, B0011, B0001, B1001 };

void setup() {
    //SE CONFIGURA PINES DE CONTROL DE MOTOR
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(motorPin3, OUTPUT);
    pinMode(motorPin4, OUTPUT);
}

void loop(){
    //DA UNA VUELTA COMPLETA
    for (int i = 0; i < stepsPerRev; i++) {
        clockwise();
        delayMicroseconds(motorSpeed); //RETARDO
    }
    delay(1000);
    //DA UNA VUELTA COMPLETA INVERSA
    for (int i = 0; i < stepsPerRev ; i++) {
        anticlockwise();
        delayMicroseconds(motorSpeed);
    }
}

```

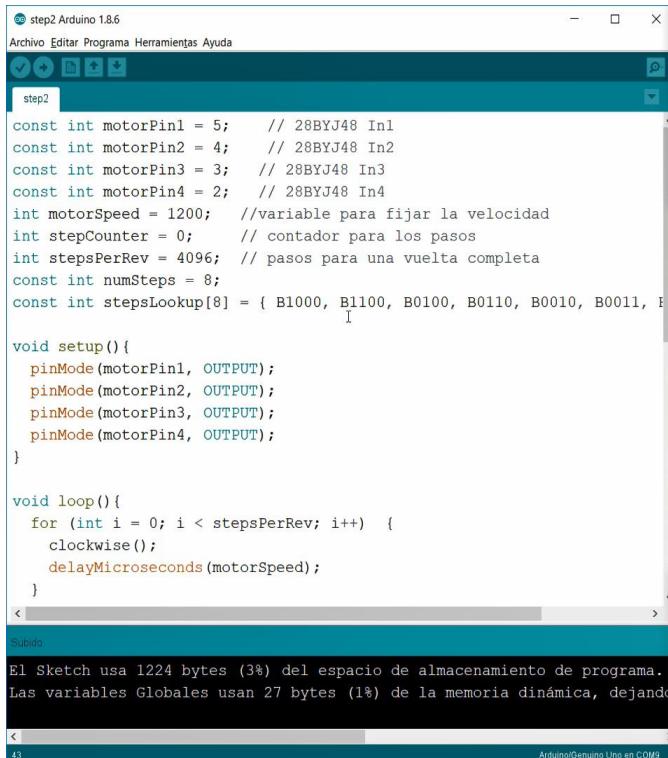
```

    }
    delay(1000);
}
//VUELTA EN SENTIDO DEL RELOJ
void clockwise(){
    stepCounter++;
    if (stepCounter >= numSteps) stepCounter = 0;
    setOutput(stepCounter);
}
//VUELTA EN SENTIDO DEL ANTI RELOJ
void anticlockwise(){
    stepCounter--;
    if (stepCounter < 0) stepCounter = numSteps - 1;
    setOutput(stepCounter);
}

//HABILITAR CONTROL PARA AVANZAR MEDIOS PASOS
void setOutput(int step){
    digitalWrite(motorPin1, bitRead(stepsLookup[step], 0));
    digitalWrite(motorPin2, bitRead(stepsLookup[step], 1));
    digitalWrite(motorPin3, bitRead(stepsLookup[step], 2));
    digitalWrite(motorPin4, bitRead(stepsLookup[step], 3));
}

```

RESULTADO



```

step2 Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
step2
const int motorPin1 = 5; // 28BYJ48 In1
const int motorPin2 = 4; // 28BYJ48 In2
const int motorPin3 = 3; // 28BYJ48 In3
const int motorPin4 = 2; // 28BYJ48 In4
int motorSpeed = 1200; //variable para fijar la velocidad
int stepCounter = 0; // contador para los pasos
int stepsPerRev = 4096; // pasos para una vuelta completa
const int numSteps = 8;
const int stepsLookup[8] = { B1000, B1100, B0100, B0110, B0010, B0011, B1010, B1110 };

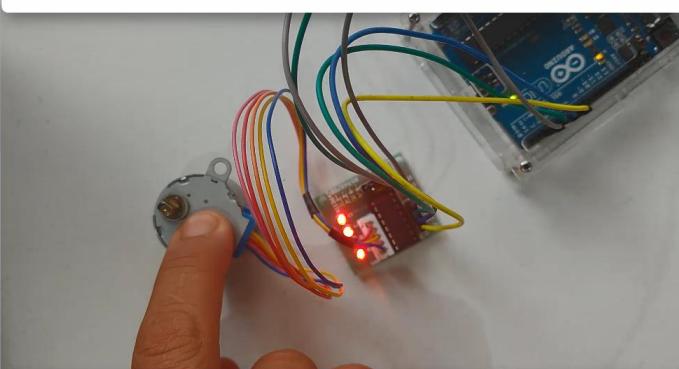
void setup(){
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(motorPin3, OUTPUT);
    pinMode(motorPin4, OUTPUT);
}

void loop(){
    for (int i = 0; i < stepsPerRev; i++) {
        clockwise();
        delayMicroseconds(motorSpeed);
    }
}

```

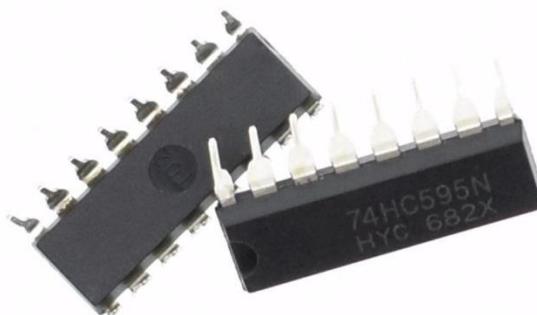
Subido
El Sketch usa 1224 bytes (3%) del espacio de almacenamiento de programa.
Las variables Globales usan 27 bytes (1%) de la memoria dinámica, dejando

CURSO ARDUINO



MOTOR A PASOS 28BYJ-48

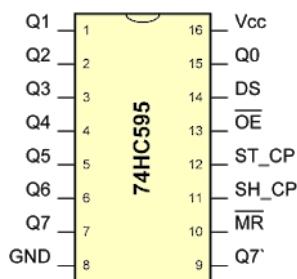
REGISTRO DE DESPLAZAMIENTO 74HC595 (SHIFT REGISTER)



MARCO TEÓRICO

El 74HC595 es un registro de desplazamiento de 8 bit con una entrada serie y salida paralelo, la gran utilidad de esto es poder utilizar y controlar ocho salidas con tan solo 3 pines de nuestro microcontrolador, mediante las entradas (Latch/Clock/Data), se puede controlar hasta ocho salidas, de manera que es de gran utilidad para proyectos en los que contamos con poca cantidad de pines en nuestro microcontrolador.

74HC595 8-Bit Shift Register Pinouts



Pin	Symbol	Description
1	Q1	Parallel data output (bit-1)
2	Q2	Parallel data output (bit-2)
3	Q3	Parallel data output (bit-3)
4	Q4	Parallel data output (bit-4)
5	Q5	Parallel data output (bit-5)
6	Q6	Parallel data output (bit-6)
7	Q7	Parallel data output (bit-7)
8	GND	Ground (0 V)
9	Q7'	Serial Data Output
10	MR	Master Reset (Active Low)
11	SH_CP	Shift Register Clock Input
12	ST_CP	Storage Register Clock Input
13	OE	Output Enable (Active Low)
14	DS	Serial Data Input
15	Q0	Parallel data output (bit-8)
16	Vcc	Positive Supply Voltage

Como se ve en la imagen, Latch es el pin 12, Clock el pin 11, y el bit de datos es el pin número 14. Nuestro chip se encuentra a la espera de una nueva secuencia de datos

Al cambiar de LOW a HIGH el bit de Data y generar un nuevo pulso de reloj pasando el bit de clock de HIGH a LOW, grabamos en la posición actual donde se encuentre el desplazamiento el valor ingresado en el pin de Data, esto lo repetimos 8 veces de manera de generar un Byte a la salida (Q0 – Q7), de esta manera podemos controlar un Byte de salida con solo tres pines de nuestro microcontrolador.

Fuente de información:

<https://www.electrontools.com/Home/WP/2016/03/09/registro-de-desplazamiento-74hc595/>

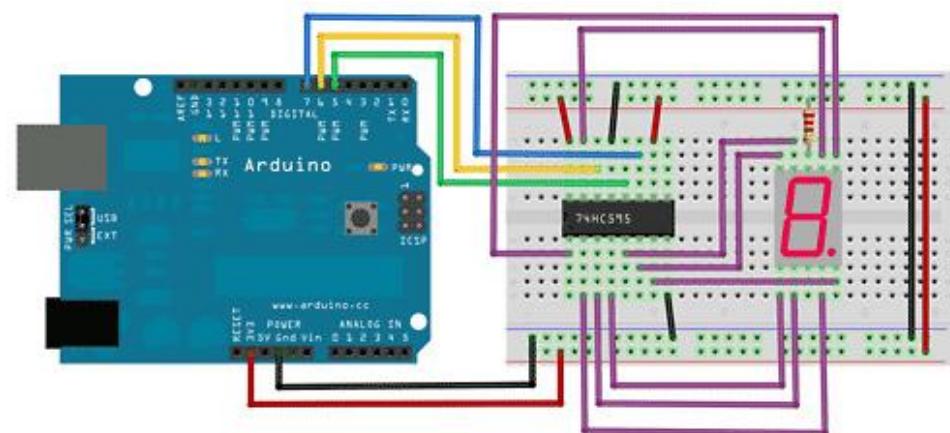
DESCRIPCIÓN DE EJEMPLO

Para el ejemplo se busca conectar un display al registro de desplazamiento **74HC595**, para simular la impresión de números de 0 a 9 y una secuencia de encendido y apagado de cada segmento del display para simular una figura en movimiento. Se utiliza la función **shiftOut(data_pin, clock_pin, MSBFIRST, valor)**, que tiene como entrada el pin del Data y el Clock, después puede tener el valor **MSBFIRST** (valor más a la izquierda) o **LSBFIRST** (valor más a la derecha), que indica por que lado debe ser leída la cadena de bits y en el último parámetro se envía el numero en binario que se pasara en serie al registro. Muy importante recalcar que antes de utilizar la función **shiftOut**, es necesario activar y desactivar el puerto **Latch**.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Un Shift Register / registro de desplazamiento 74HC595

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

```
# define latch_pin 5
#define data_pin 6
#define clock_pin 7
//SE HABILITAN PINES COMO SALIDA
void setup () {
    for (int n=5; n<8; n++) pinMode(n,OUTPUT);
}

void loop () {
    //SECUENCIA 0 A 8 PARA ENVIAR DATOS POR PIN 6
    for(int sec=0; sec<8;sec++) {
        digitalWrite(latch_pin,HIGH);
        //FUNCION PARA ENVIO DE DATOS AL SHIFT REGISTER
        shiftOut(data_pin,clock_pin, MSBFIRST, animacion(sec) );
        delay(50);
        digitalWrite(latch_pin,LOW);
    }
}
//ANIMACIÓN PARA PRENDER LEDS DE DISPLAY EN FORMA DE INFINITO
int animacion (int sec_){
```

```

int frame =0;
//SE PRENDE UN LED A LA VEZ
switch(sec_){
    case 0: frame = B00000001;
    break;
    case 1: frame = B00000010;
    break;
    case 2: frame = B01000000;
    break;
    case 3: frame = B00010000;
    break;
    case 4: frame = B00001000;
    break;
    case 5: frame = B00000100;
    break;
    case 6: frame = B01000000;
    break;
    case 7: frame = B00100000;
    break;
}
return frame;
}

```

NUMEROS 0-9

```

const int latchPin = 8; // Pin conectado al Pin 12 del 74HC595 (Latch)
const int dataPin = 9; // Pin conectado al Pin 14 del 74HC595 (Data)
const int clockPin = 10; // Pin conectado al Pin 11 del 74HC595 (Clock)
int i =0;
//SE CONFIGURA PINES DE SHIFT REGISTER
void setup() {
  Serial.begin(9600);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  delay(1000); //PAUSA
  Serial.println("CERO"); //IMPRIME EL VALOR EN MONITOR SERIE
  digitalWrite(latchPin, LOW); //HABILITA LATCH
  //ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 0 EN DISPLAY
  shiftOut(dataPin, clockPin, MSBFIRST, 252);
  digitalWrite(latchPin, HIGH); //DESHABILITA LATCH
  delay(1000);
  Serial.println("UNO");
  digitalWrite(latchPin, LOW);
  //ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 1 EN DISPLAY
  shiftOut(dataPin, clockPin, MSBFIRST, 96);
  digitalWrite(latchPin, HIGH);
  delay(1000);
  Serial.println("DOS");
  digitalWrite(latchPin, LOW);
  //ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 2 EN DISPLAY
  shiftOut(dataPin, clockPin, MSBFIRST, 218);
  digitalWrite(latchPin, HIGH);
  delay(1000);
  Serial.println("TRES");
  digitalWrite(latchPin, LOW);
  //ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 3 EN DISPLAY
  shiftOut(dataPin, clockPin, MSBFIRST, 242);
  digitalWrite(latchPin, HIGH);
  delay(1000);
  Serial.println("CUATRO");
  digitalWrite(latchPin, LOW);
  //ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 4 EN DISPLAY
  shiftOut(dataPin, clockPin, MSBFIRST, 102);
  digitalWrite(latchPin, HIGH);
  delay(1000);
  Serial.println("CINCO");
  digitalWrite(latchPin, LOW);
}

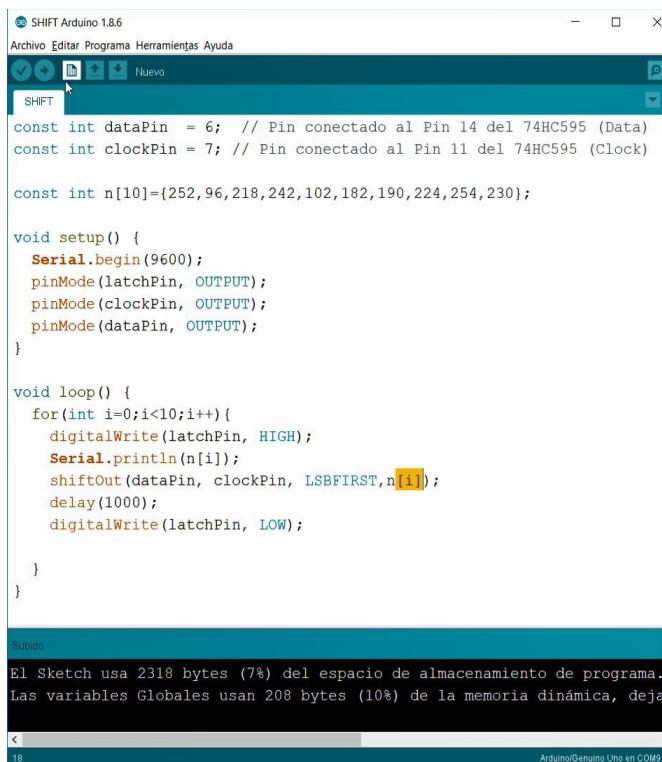
```

```

//ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 5 EN DISPLAY
shiftOut(dataPin, clockPin, MSBFIRST, 109);
digitalWrite(latchPin, HIGH);
delay(1000);
Serial.println("SEIS");
digitalWrite(latchPin, LOW);
//ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 6 EN DISPLAY
shiftOut(dataPin, clockPin, MSBFIRST, 125);
digitalWrite(latchPin, HIGH);
delay(1000);
Serial.println("SIETE");
digitalWrite(latchPin, LOW);
//ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 7 EN DISPLAY
shiftOut(dataPin, clockPin, MSBFIRST, 7);
digitalWrite(latchPin, HIGH);
delay(1000);
Serial.println("OCHO");
digitalWrite(latchPin, LOW);
//ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 8 EN DISPLAY
shiftOut(dataPin, clockPin, MSBFIRST, 127);
digitalWrite(latchPin, HIGH);
delay(1000);
Serial.println("NUEVE");
digitalWrite(latchPin, LOW);
//ENVIA BITS AL SHIFT REGISTER PARA IMPRIMIR 9 EN DISPLAY
shiftOut(dataPin, clockPin, MSBFIRST, 103);
digitalWrite(latchPin, HIGH);
}
}

```

RESULTADO



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** SHIFT Arduino 1.8.6
- Menu Bar:** Archivo Editar Programa Herramientas Ayuda
- Toolbars:** Standard toolbar with icons for Open, Save, Run, Stop, and Reload.
- Code Area:**

```

const int dataPin = 6; // Pin conectado al Pin 14 del 74HC595 (Data)
const int clockPin = 7; // Pin conectado al Pin 11 del 74HC595 (Clock)

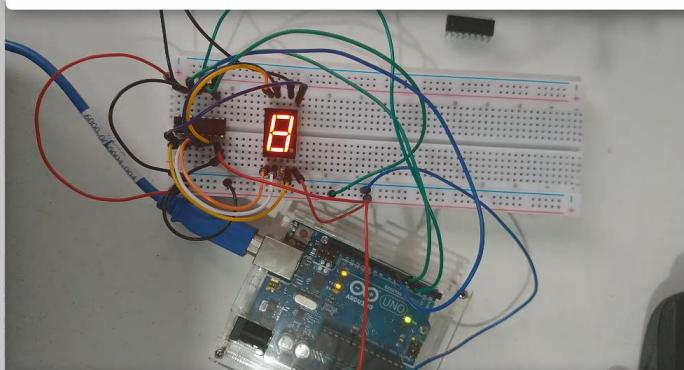
const int n[10]={252,96,218,242,102,182,190,224,254,230};

void setup() {
  Serial.begin(9600);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  for(int i=0;i<10;i++){
    digitalWrite(latchPin, HIGH);
    Serial.println(n[i]);
    shiftOut(dataPin, clockPin, LSBFIRST,n[i]);
    delay(1000);
    digitalWrite(latchPin, LOW);
  }
}

```
- Status Bar:**
 - Subido
 - El Sketch usa 2318 bytes (7%) del espacio de almacenamiento de programa.
 - Las variables Globales usan 208 bytes (10%) de la memoria dinámica, dejar
 - 18
 - Arduino/Genuino Uno en COM9

CURSO ARDUINO



SHIFT REGISTER 74HC595

RFID R522 (IDENTIFICADOR POR RADIOFRECUENCIA)



MARCO TEÓRICO

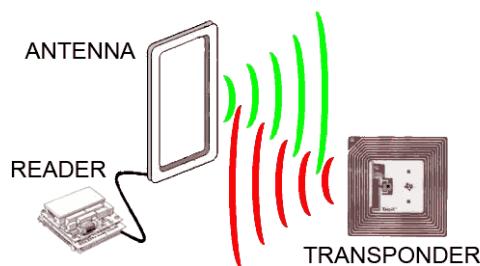
El RFID (Identificador por radiofrecuencia) es un conjunto de tecnologías diseñadas para leer etiquetas (tags) a distancia de forma inalámbrica. Los lectores RFID pueden ser conectados a un autómata o procesador como Arduino.

Las etiquetas RFID están disponibles en una gran variedad de formatos, tales como pegatinas adhesibles, tarjetas, llaveros, pueden integrarse en un determinado producto o, incluso, insertarse bajo la piel en un animal o humano.

Los RFID son ampliamente empleados, por ejemplo, en sistemas de alarma, aplicaciones comerciales en sustitución de códigos de barras, cerraduras electrónicas, sistemas de pago, tarjetas personales, control de accesos recintos como gimnasios o piscinas, fichaje en empresas, entre otras muchas aplicaciones.

En nuestros proyectos de electrónica con Arduino podemos usar el RFID, por ejemplo, en aplicaciones comerciales para mostrar en una pantalla los datos al acercar un producto, cambiar la luz o iluminación, abrir una puerta con una tarjeta personal, o hacer que un robot se comporte de forma distinta pasándole una tarjeta.

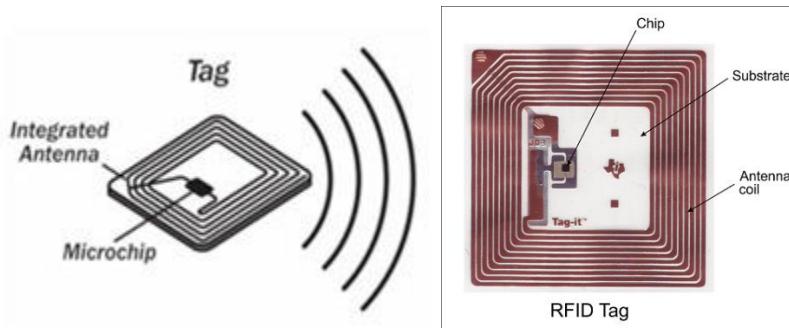
El RFID es un conjunto de tecnologías inalámbricas diseñadas para obtener una información almacenada en un dispositivo denominado etiqueta (tag). El lector (transceptor) es en realidad un emisor-receptor que, en primer lugar, emite una señal para iniciar la comunicación con las etiquetas (transpondedores). Esta señal es captada por las etiquetas dentro del alcance, las cuál responden transmitiendo la información que almacenada que, finalmente, es captada y decodificada por el lector RFID.



El RFID puede operar en cuatro bandas de frecuencia, siendo la más frecuente 13.56 Mhz.

- Baja frecuencia 125-134.2 kHz. Control de animales, llaves de automóviles...
 - Alta frecuencia 13.56 MHz. Control de accesos, control de artículos en tiendas...
 - Ultra alta frecuencia (UHF) 868 – 956 GHz
 - Microondas, 2,45 GHz

Existen etiquetas RFID de sólo lectura, es decir, en las que la información que contienen es grabada durante su fabricación y no puede modificarse, y etiquetas de lectura y escritura, en las que podemos sobreescibir la información de la etiqueta.

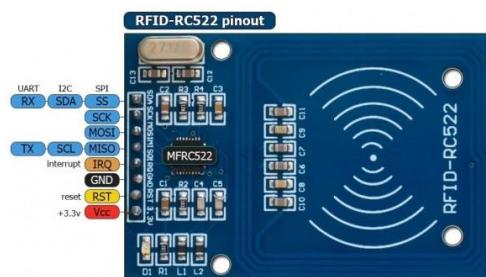


Respecto a la alimentación, existen etiquetas RFID activas que disponen de su propia fuente de energía (por ejemplo, una batería). El rango de lectura puede ser de 10m a 100m.

El NFC (Near Field Communication) es un subconjunto de la tecnología RFID que por diseño establece métodos para limitar la distancia de transmisión a menos de 10 cm. El NFC está experimentando un gran desarrollo debido a la inclusión en los smartphones, y posibles formas de pago.

Sin embargo, aunque íntimamente relacionados, no hay que confundir RFID y NFC. En particular, no todos los sistemas RFID y NFC serán compatibles, es decir, que no siempre vamos a poder leer una tarjeta RFID con el lector NFC de un smartphone.

MIFARE es una tecnología de tarjetas inalámbricas propiedad de NXP Semiconductores. Es uno de los estándares más implantados como tarjetas inteligentes sin contacto (TSIC). El Mifare MFRC522 es un lector de tarjetas RFID que incorpora comunicación por bus SPI, bus I2C y UART, por lo que es sencillo de conectar con Arduino.



El lector MFRC522 opera en la frecuencia de 13.56Mhz y tiene una distancia de lectura de 0 a 60. El MFRC522 tiene un consumo de 13-26 mA durante la escritura, 10-13mA en stanby e inferior a 80uA en modo sleep. La tensión de alimentación es de 3.3V.

Fuente de información:

<https://www.luisllamas.es/arduino-rfid-mifare-rc522/>

<https://www.prometec.net/arduino-rfid/>

DESCRIPCIÓN DE EJEMPLO

Para este ejemplo se conecta el lector RFID a los pines digitales del Arduino, mediante las librerías **SPI.h** y **MFRC522.h**, se crea el objeto **MFRC522 mfrc522(SS_PIN, RST_PIN)**, que como parámetros recibe el pin **Reset** y el pin **SS (SDA)**. Se utiliza una función que recibe el array de bytes, y que imprime el valor del **TAG** en Hexadecimal. En un segundo ejemplo se muestra como validar o comparar el valor del **TAG** para saber si es válida o no.

Librería:

<https://www.prometec.net/wp-content/uploads/2016/03/rfid-master.zip>

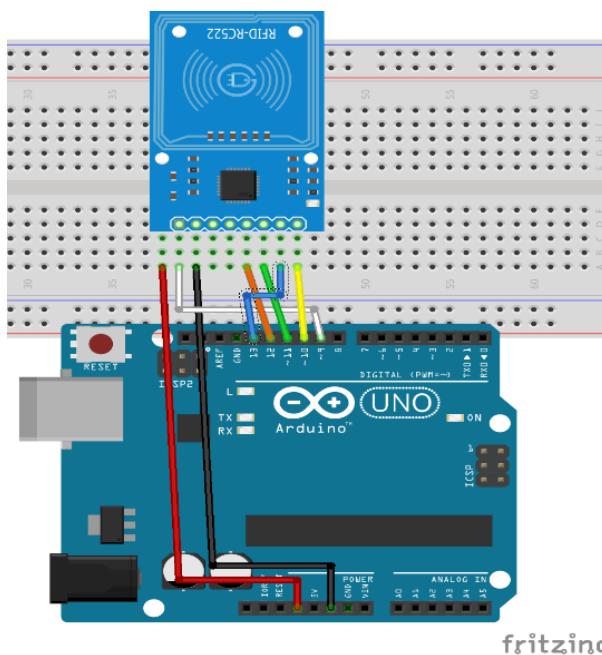
<https://github.com/miguelbalboa/rfid/tree/master/src>

<https://www.arduinolibraries.info/libraries/mfrc522>

MATERIAL A UTILIZAR

- Arduino UNO
 - Protoboard
 - Cables
 - Un lector RFID RC522
 - TAGs diferentes y llavero

DIAGRAMA DE CONEXIÓN



CÓDIGO FUENTE

MOSTRAR ID DEL TAG O LLAVERO

```
//SE AGREGA LIBRERÍAS DE RFID
#include <SPI.h>
#include <MFRC522.h>

const int RST_PIN = 9;           // Pin 9 para el reset del RC522
const int SS_PIN = 10;          // Pin 10 para el SS (SDA) del RC522
MFRC522 mfrc522(SS_PIN, RST_PIN); // Crear instancia del MFRC522

//FUNCION PARA IMPIRMIR EL VALOR DEL ID EN HEXADECIMAL
void printArray(byte *buffer, byte bufferSize) {
    for (byte i = 0; i < bufferSize; i++) {
        Serial.print(buffer[i] < 0x10 ? " 0" : " ");
        Serial.print(buffer[i], HEX);
    }
}

//SE CONFIGURA MONITO SERIE, SPI Y OBJETO PARA CONTROL RFID
void setup(){
    Serial.begin(9600);          //Inicializa la velocidad de Serial
    SPI.begin();                 //Función que inicializa SPI
    mfrc522.PCD_Init();         //Función que inicializa RFID
}

void loop(){
    //SE PREPARA EL RFID PARA UNA NUEVA LECTURA
    if (mfrc522.PICC_IsNewCardPresent()==1){
        if (mfrc522.PICC_ReadCardSerial()==1){
            //SE IMPRIME EL VALOR DEL ID DEL TAG O LLAVERO
            Serial.print(F("Card UID:"));
            printArray(mfrc522.uid.uidByte, mfrc522.uid.size); //SE IMPRIME ID
            Serial.println();
            mfrc522.PICC_HaltA(); //SE LIBERA LECTOR
        }
    }
    delay(250);
}
```

VALIDAR ID DE TAG

```
#include <SPI.h>
#include <MFRC522.h>

const int RST_PIN = 9;           // Pin 9 para el reset del RC522
const int SS_PIN = 10;          // Pin 10 para el SS (SDA) del RC522
MFRC522 mfrc522(SS_PIN, RST_PIN); // Crear instancia del MFRC522

byte validKey1[4] = { 0xC3, 0x33, 0x27, 0x19 }; // Ejemplo de clave valida

//Función para comparar dos vectores
bool isEqualArray(byte* arrayA, byte* arrayB, int length){
    for (int index = 0; index < length; index++) {
        if (arrayA[index] != arrayB[index]) return false;
    }
    return true;
}

void setup() {
    Serial.begin(9600); // Iniciar serial
    SPI.begin();        // Iniciar SPI
    mfrc522.PCD_Init(); // Iniciar MFRC522
}

void loop() {
    if (mfrc522.PICC_IsNewCardPresent()) {
        if (mfrc522.PICC_ReadCardSerial()) {
            //SE VALIDA SI EL ID DEL TAG CORRESPONDE
```

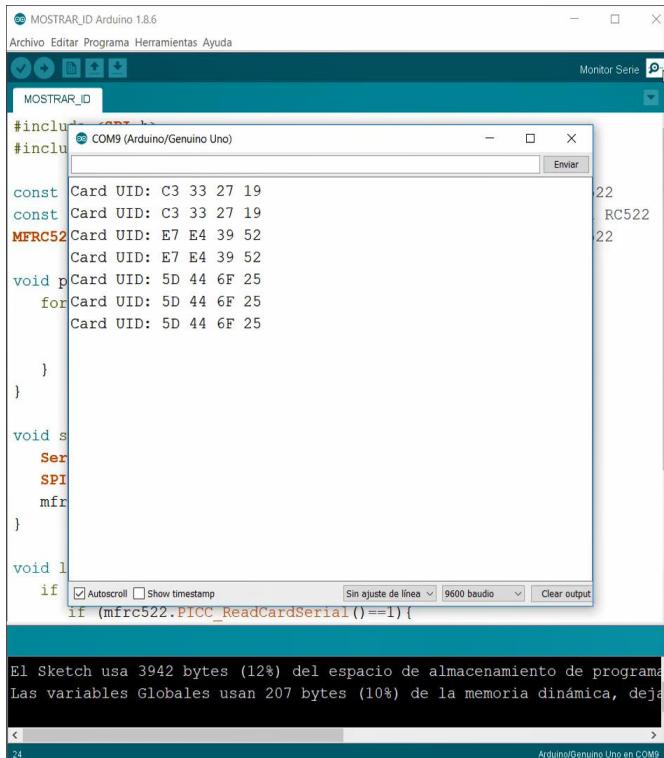
```

if (isEqualArray(mfrc522.uid.uidByte, validKey1, 4))
    Serial.println("Tarjeta valida");
else
    Serial.println("Tarjeta invalida");

// Finalizar lectura actual
mfrc522.PICC_HaltA();
}
}
delay(250);
}

```

RESULTADO



The screenshot shows the Arduino IDE interface. On the left, the code for reading an RFID card's ID is displayed. On the right, the Serial Monitor window shows the output of the program, which repeatedly prints the card's UID. The card being used has a UID of C3 33 27 19.

```

const Card UID: C3 33 27 19
const Card UID: C3 33 27 19
MFRC522 Card UID: E7 E4 39 52
Card UID: E7 E4 39 52
void pCard UID: 5D 44 6F 25
for Card UID: 5D 44 6F 25
Card UID: 5D 44 6F 25

}
}

void s
Serial
SPI
mfrc
}

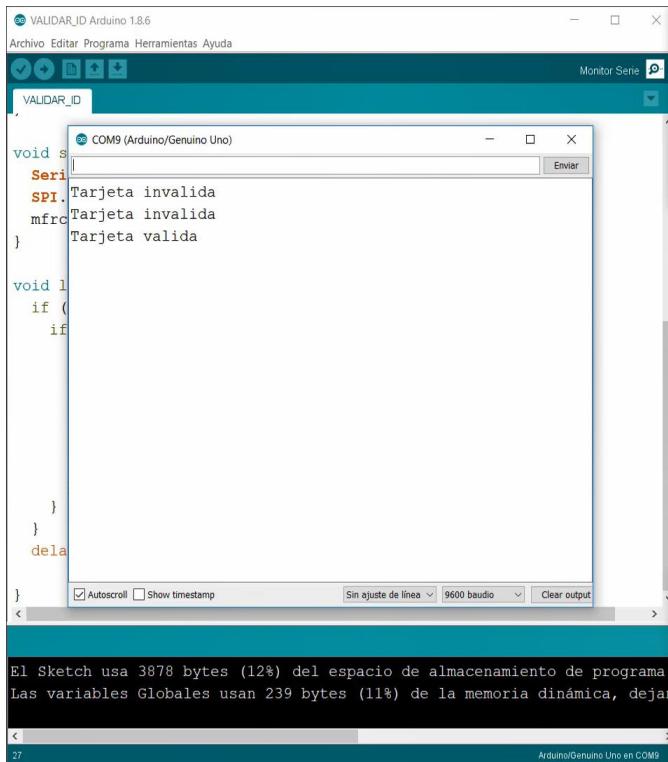
void l
if
if (mfrc522.PICC_ReadCardSerial() == 1) {

```

El Sketch usa 3942 bytes (12%) del espacio de almacenamiento de programa.
Las variables Globales usan 207 bytes (10%) de la memoria dinámica, deje



RFID RC522

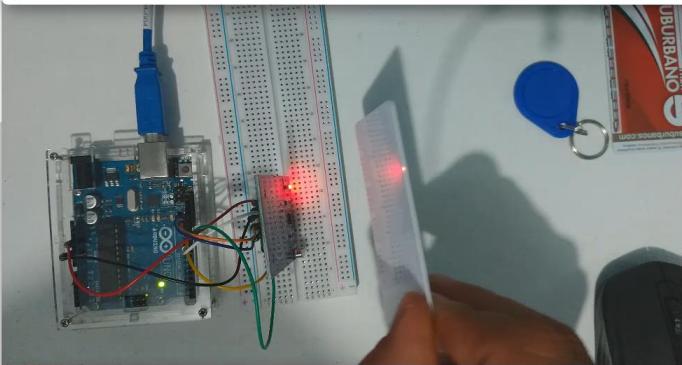


```
VALIDAR_ID Arduino 1.8.6
Archivo Editar Programa Herramientas Ayuda
VALIDAR_ID
void setup() {
  Serial.begin(9600);
  SPI.begin();
  mfrc522.begin();
}

void loop() {
  if (mfrc522.MFRC522_IsCardPresent()) {
    if (mfrc522.MFRC522_Read(&uid)) {
      delay(1000);
      Serial.println("Tarjeta valida");
    } else {
      Serial.println("Tarjeta invalida");
    }
  }
  delay(1000);
}
```

El Sketch usa 3878 bytes (12%) del espacio de almacenamiento de programa.
Las variables Globales usan 239 bytes (11%) de la memoria dinámica, dejan

CURSO ARDUINO



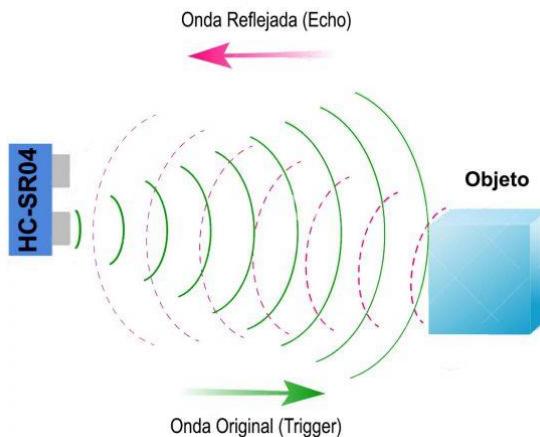
RFID RC522

ULTRASÓNICO (SENSOR DE DISTANCIA)



MARCO TEÓRICO

El sensor HC-SR04 es un sensor de distancia de bajo costo que utiliza ultrasonido para determinar la distancia de un objeto en un rango de 2 a 450 cm. Destaca por su pequeño tamaño, bajo consumo energético, buena precisión y excelente precio. El sensor HC-SR04 es el más utilizado dentro de los sensores de tipo ultrasonido, principalmente por la cantidad de información y proyectos disponibles en la web. De igual forma es el más empleado en proyectos de robótica como robots laberinto o sumo, y en proyectos de automatización como sistemas de medición de nivel o distancia.



El sensor HC-SR04 posee dos transductores: un emisor y un receptor piezoelectréticos, además de la electrónica necesaria para su operación. El funcionamiento del sensor es el siguiente: el emisor piezoelectrético emite 8 pulsos de ultrasonido(40KHz) luego de recibir la orden en el pin TRIGGER, las ondas de sonido viajan en el aire y rebotan al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoelectrético, luego el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto. El funcionamiento del sensor no se ve afectado por la luz solar o material de color negro.

La distancia se puede calcular utilizando la siguiente formula:

$$\text{Distancia(m)} = \{(Tiempo\ del\ pulso\ ECHO)\ * (Velocidad\ del\ sonido=340m/s)\}/2$$

Fuente de información:
<https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html>

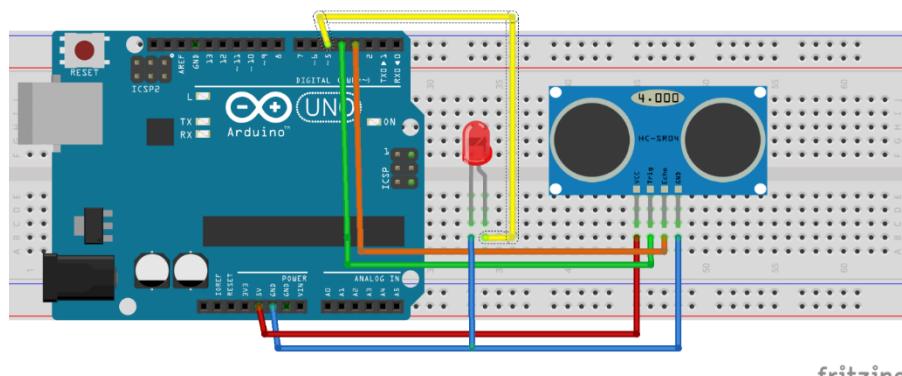
DESCRIPCIÓN DE EJEMPLO

Se conectar el sensor ultrasónico al Arduino, especialmente los pines importantes son el Trigger y Echo, no se requiere alguna librería específica, simplemente se activa el pin del Trigger o disparador para enviar un pulso, que dependiendo donde rebota, se recibe por el pin del ECHO mediante la función **pulseIn**, que nos entrega el valor del tiempo, se aplica la fórmula que calcula la distancia proporcional, que hay entre el objeto y el sensor (No es tan exacto). El LED se activa cuando la lectura del sensor ultrasónico sea igual o menor a 5 centímetros.

MATERIAL A UTILIZAR

- Arduino UNO
- Protoboard
- Cables
- Un LED
- Un Ultrasónico HC_SR04

DIAGRAMA DE CONEXIÓN



fritzing

CÓDIGO FUENTE

```
//SE DECLARAN VARIABLES A UTILIZAR
const int echo = 3;
const int trigger = 4;
const int led = 5;
float tiempo, distancia;
//SE CONFIGURA ECHO, TRIGGER Y LED
void setup() {
    Serial.begin(9600);
    pinMode(echo, INPUT);
    pinMode(trigger, OUTPUT);
    pinMode(led, OUTPUT);
}

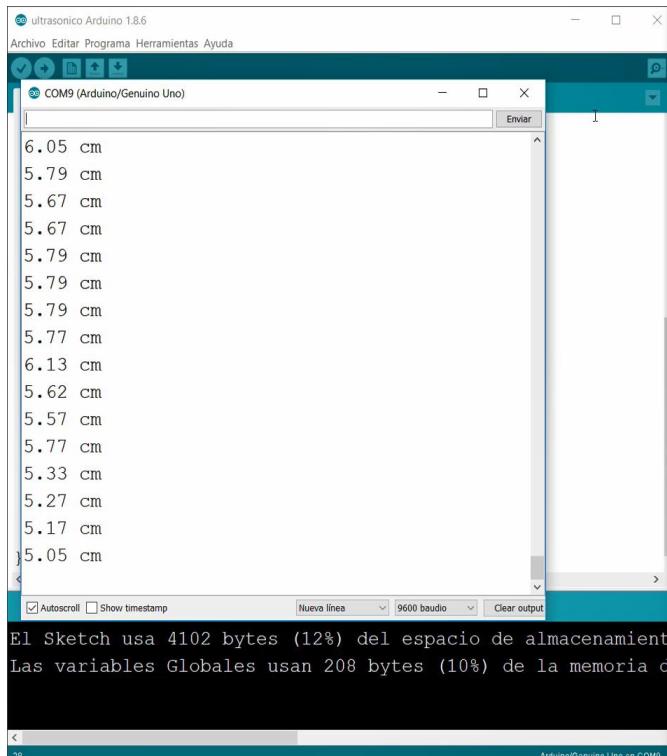
void loop() {
    digitalWrite(trigger, LOW); //DISPARO APAGADO
    delay(2);
    digitalWrite(trigger, HIGH); //SE HABILITA PULSO
    delay(10);
    digitalWrite(trigger, LOW); //SE APAGA
    tiempo = pulseIn(echo, HIGH); //SE LEE TIEMPO DE ECHO DEL PULSO RECIBIDO
    distancia = tiempo / 58.2; //SE CALCULA DISTANCIA
    //SE IMPRIME DISTANCIA
    Serial.print(distancia);
```

```

Serial.println(" cm");
delay(200);
//SI LA DISTANCIA ES MENOR O IGUAL A 5 SE ENCIENDE LED
if (distancia <= 5) {
  digitalWrite(led, HIGH);
  delay(500);
} else {
  digitalWrite(led, LOW);
}
}

```

RESULTADO



**CURSO
ARDUINO**
**SENSOR
ULTRASÓNICO**



CONCLUSIÓN

Esta es una guía de aprendizaje de Arduino, especialmente elaborada para utilizar el Arduino Starter Kit, todos los componentes que aquí se tratan vienen incluidos dentro de este Kit.

Esta guía no se pretende sustituir de ninguna forma el conocimiento que existe dentro de Internet y en medios oficiales.

Es importante siempre consultar la referencia oficial de Arduino en:

<https://www.arduino.cc/en/Guide/HomePage>

Atte. René Domínguez Escalona

Cualquier comentario o sugerencia

Medios de Contacto

Correo Electrónico: leo_skriom2k@hotmail.com

Celular: 55 64 21 22 21

Facebook: <https://www.facebook.com/skriom>