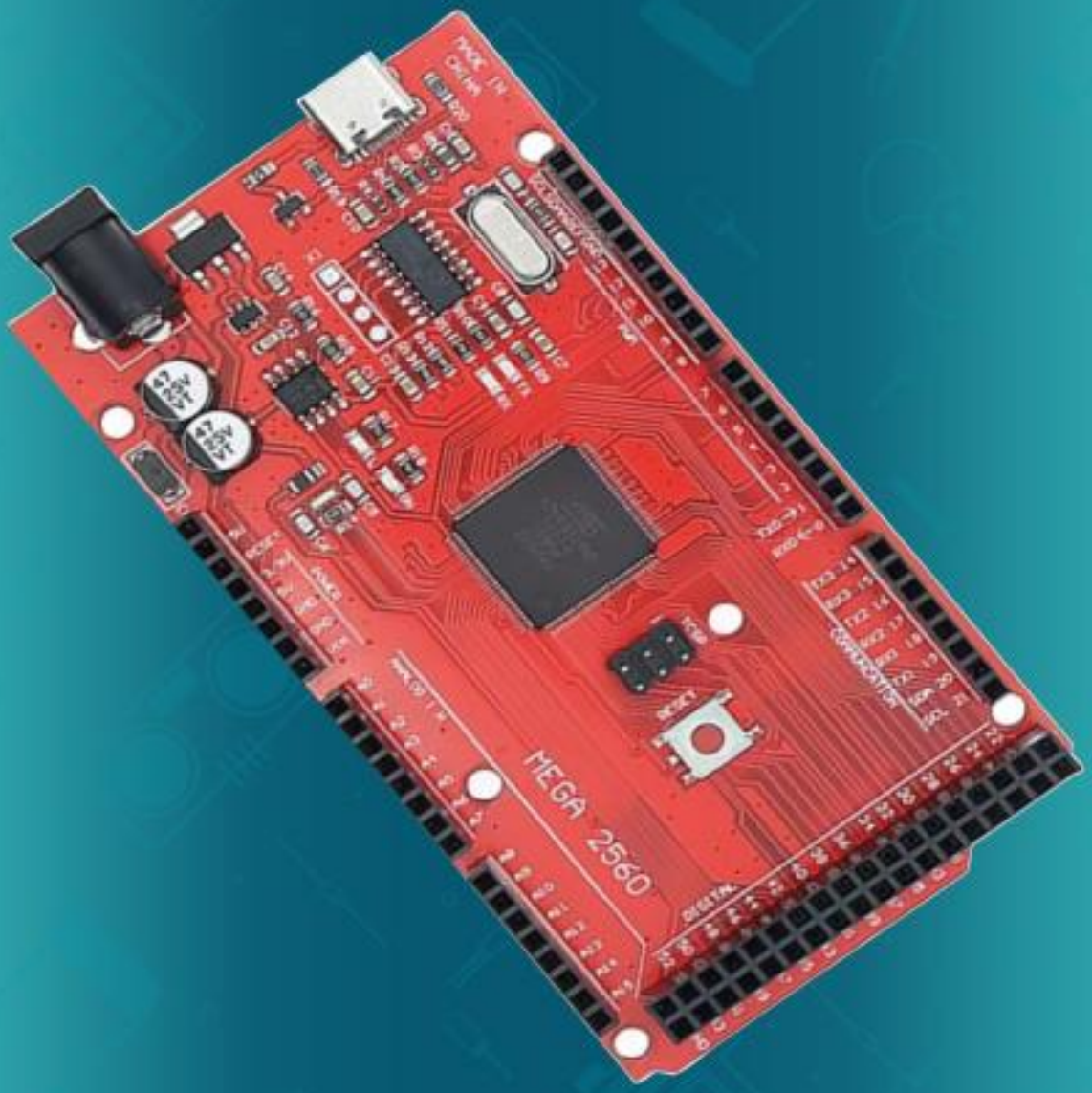
Starter KIT Mega 2560

Transforma tus ideas en acción.

Manual de Prácticas



UNIT ELECTRONICS

ÍNDICE

Primeros pasos con Mega2560

Este tutorial es parte del Starter Kit Mega2560. Incluye 31 prácticas súper variadas para que te diviertas y aprendas. Todas las prácticas cuentan con la siguiente estructura:

- Descripción de práctica
- Material
- Diagrama de conexión
- Requerimientos previos
- Código de funcionamiento

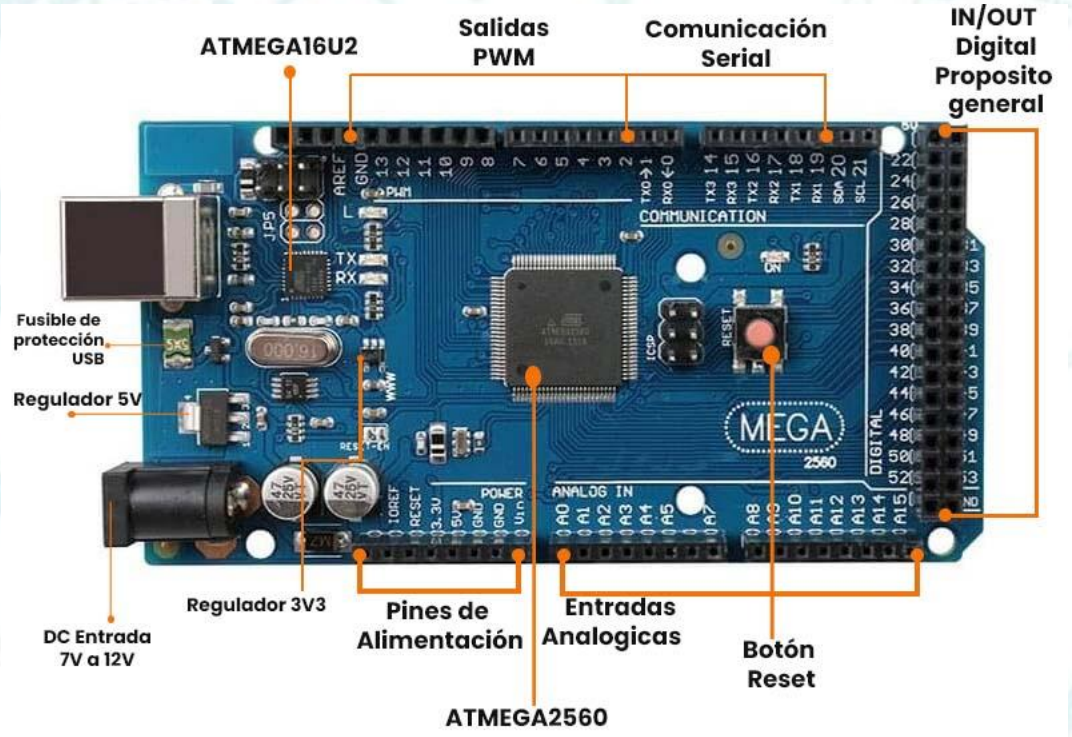
Vas a trabajar con la tarjeta de desarrollo Mega2560, aprenderás a usar sensores, actuadores, displays y componentes básicos como LEDs, resistencias y transistores. ¡Y lo mejor! Vas a programar todo en C++ usando el entorno de Arduino IDE.

Arquitectura del Mega2560

- Microcontrolador: ATmega2560 (8-bit AVR).
- Voltaje de operación: 5V.
- Entradas/Salidas digitales (I/O): 54 pines, de los cuales 15 pueden usarse como salidas PWM.
- Entradas analógicas: 16 pines (resolución de 10 bits).
- Memoria Flash: 256 KB (para almacenar el código).
- Memoria SRAM: 8 KB (para variables durante la ejecución).
- Memoria EEPROM: 4 KB (para almacenar datos persistentes).
- Velocidad de reloj: 16 MHz.
- Comunicación: UART (4 puertos), SPI, I2C.
- Conectividad USB: Permite la programación y comunicación con el ordenador.

Zonas de trabajo del Arduino Mega 2560

El Arduino Mega 2560 está organizado en varias zonas o bloques funcionales que facilitan su uso y comprensión. A continuación, se describen las principales:



Microcontrolador (ATmega2560): Es el cerebro de la placa, encargado de ejecutar el programa cargado por el usuario. Controla todas las operaciones de entrada y salida, así como la comunicación con otros dispositivos.

Pines de Entrada/Salida Digital (I/O): Los 54 pines digitales pueden configurarse como entradas o salidas. Algunos de estos pines tienen funciones especiales, como PWM (modulación por ancho de pulso) o comunicación serial.

Pines de Entrada Analógica: Los 16 pines analógicos permiten leer señales de sensores o dispositivos que generan voltajes variables.Tienen una resolución de 10 bits, lo que permite detectar 1024 niveles diferentes (0 a 1023).

Puertos de Comunicación

- **UART**: 4 puertos seriales para comunicación con otros dispositivos.
- **I2C**: Protocolo de comunicación para conectar sensores y módulos.
- **SPI**: Protocolo de comunicación rápida para dispositivos como pantallas o memorias.
- **Regulador de Voltaje**: Asegura que la placa reciba un voltaje estable de 5V, incluso si se alimenta con voltajes más altos (por ejemplo, 7-12V a través del jack de alimentación).
- **Conector USB**: Permite la conexión con un ordenador para programar la placa y alimentarla. También se utiliza para la comunicación serial entre el Arduino y el PC.
- **Cristal oscilador**: Proporciona una señal de reloj estable de 16 MHz, necesaria para sincronizar las operaciones del microcontrolador.

Memorias

- **Flash**: Almacena el programa cargado por el usuario.
- **SRAM**: Almacena temporalmente las variables y datos durante la ejecución del programa.
- **EEPROM**: Permite guardar datos de forma permanente, incluso cuando la placa se apaga

Pines de Alimentación

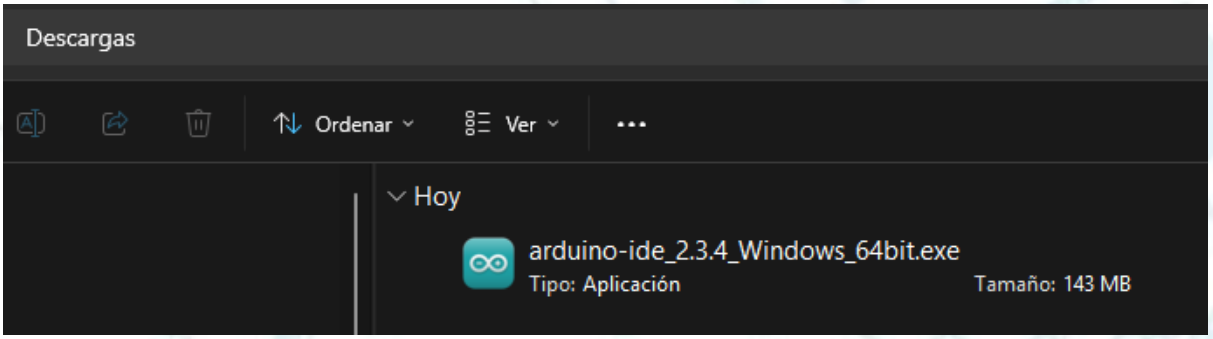
- **5V y 3.3V**: Proporcionan voltajes estables para alimentar sensores y módulos externos.
- **GND**: Pines de tierra para completar los circuitos.
- **Vin**: Permite alimentar la placa con un voltaje externo (7-12V recomendado).
- **Botón de Reset**: Reinicia el microcontrolador, permitiendo que el programa cargado comience desde el principio.

Instalación de Arduino IDE

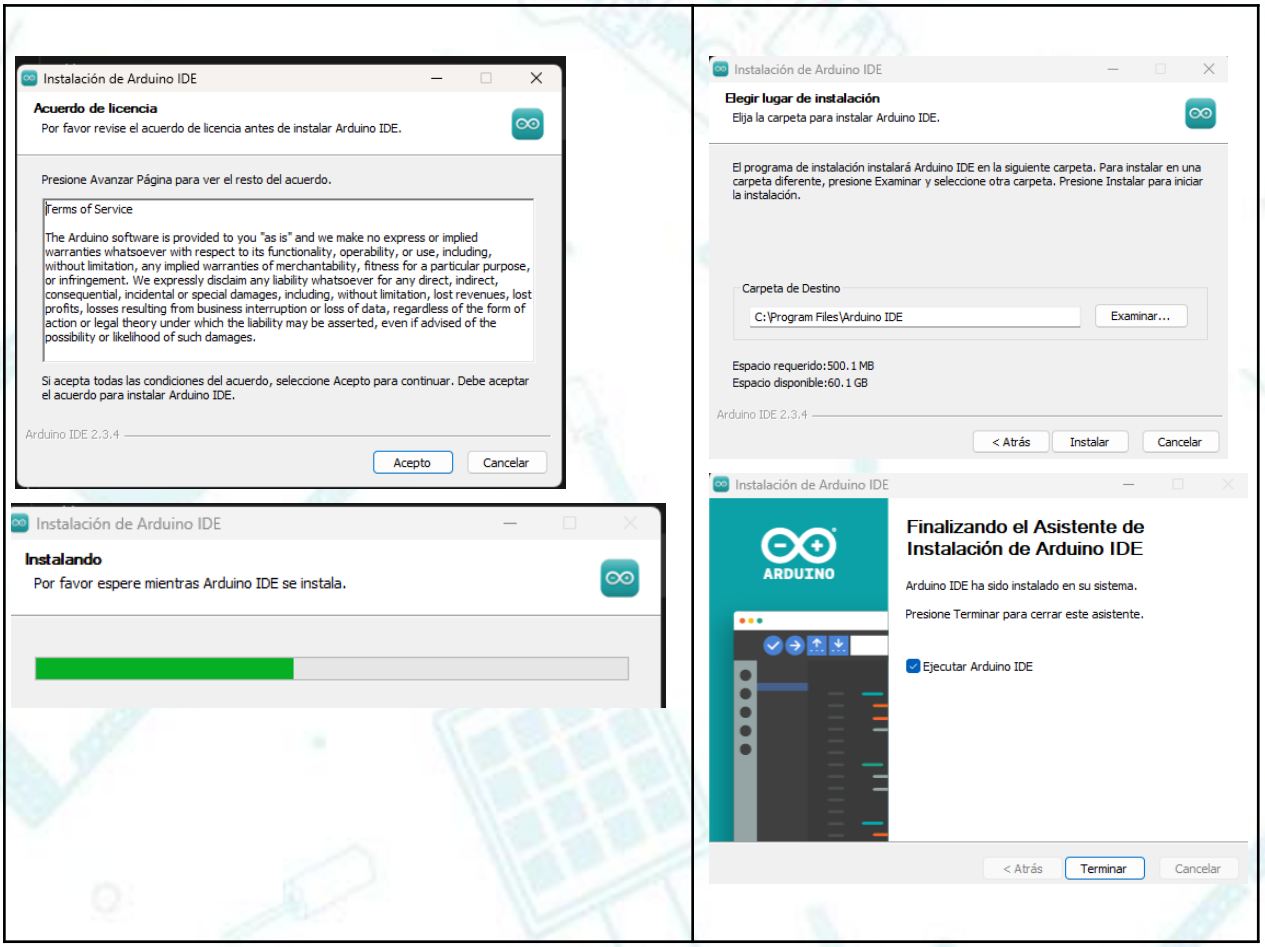
Para comenzar a utilizar el es necesario tener instalado el software de programación Arduino IDE. Sigue los siguientes pasos:

Descarga IDE Arduino, disponible para Windows, Linux o macOS desde el siguiente link: **https://www.arduino.cc/en/software**, (da click en JUST DOWNLOAD)

2. Dependiendo de tu sistema operativo, abre tu carpeta de **Descargas** y podrás ver el ejecutable , listo para comenzar la instalación.



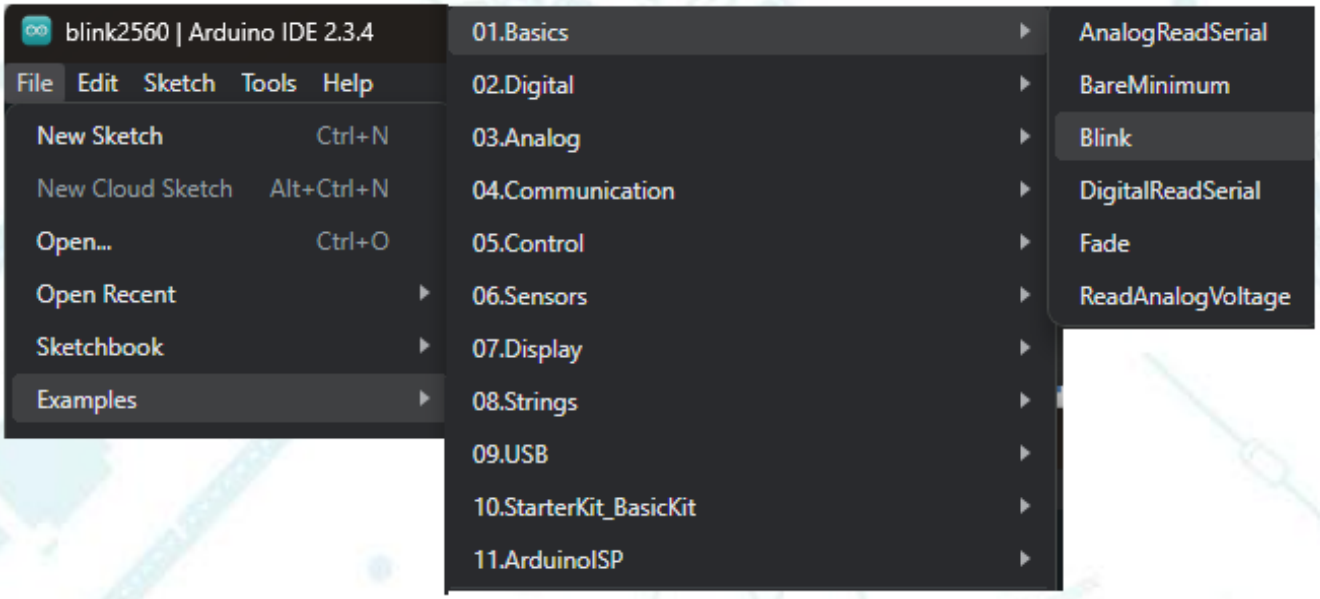
3. Acepta los acuerdos de licencia y elige donde guardarás el programa de instalación y comenzará la instalación.



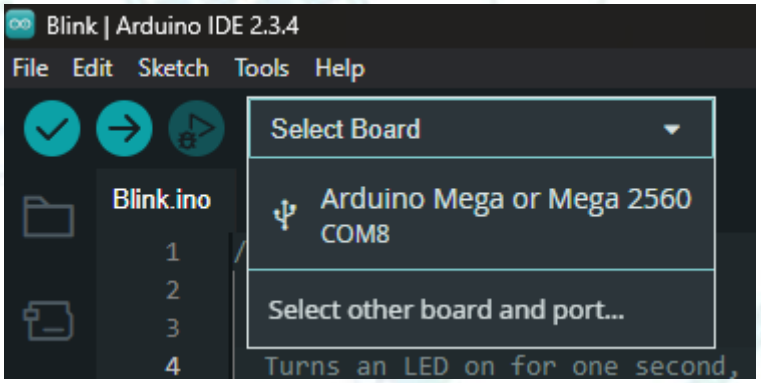
Compila y carga tu primer código en Mega2560

Para realizar este ejemplo compilaremos y cargaremos un ejemplo de código disponible desde el Arduino IDE, llamado BLINK. Para ello seguimos la siguiente instrucciones:

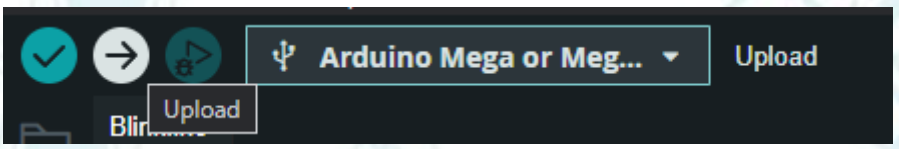
1. Conecta el Mega2560 al ordenador mediante el puerto USB.
2. Busca el código a cargar en : **File> Examples>01.Basic>Blink**



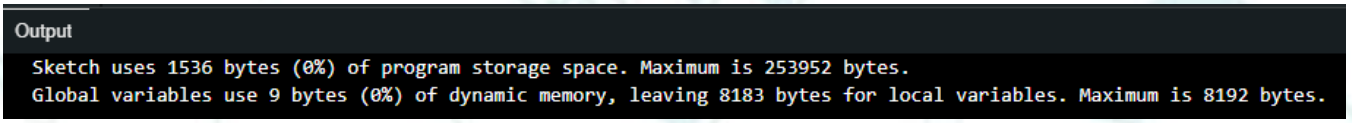
3. Selecciona la tarjeta Mega 2560 en : **Select Board**



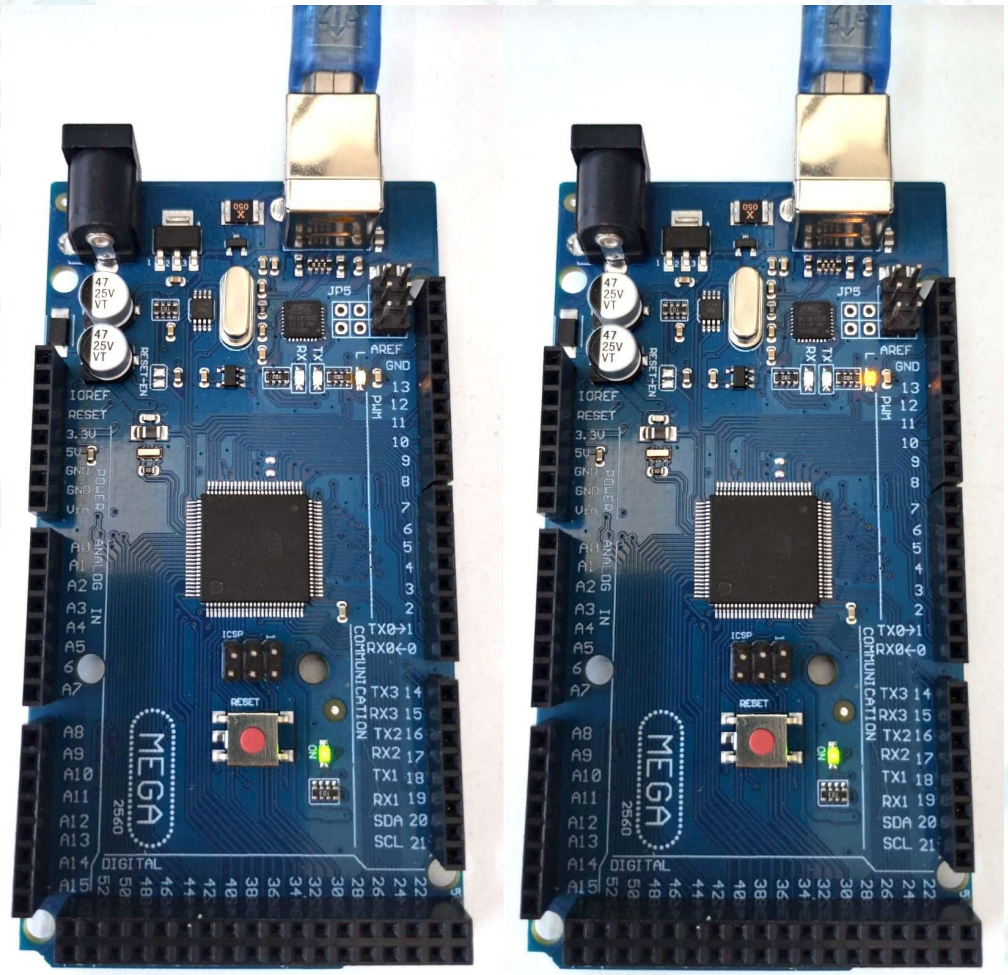
4. Carga el código a la tarjeta



5. Se confirmara la carga por el monitor serial y el led de la placa Mega2560 comenzará a parpadear a una velocidad de 1ms



¡Felicidades has cargado un código en el Mega2560, el primero de muchos!

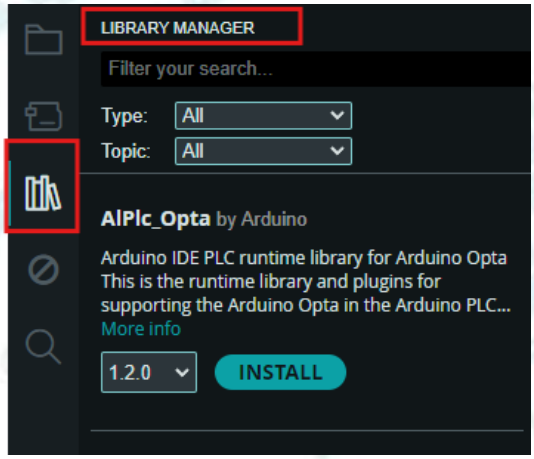


Herramientas del IDE Arduino

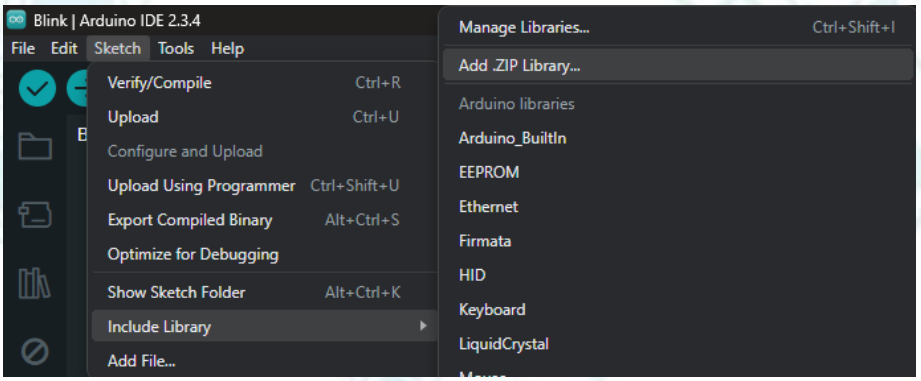
En el entorno de Arduino IDE existen herramientas que serán de utilidad en el desarrollo de las prácticas, ya que fácilmente podrás realizar programas en C + + . Pero en ocasiones nos apoyaremos de Bibliotecas o Librerías para facilitar el uso de sensores o actuadores con la Mega2560.

Instalación de Bibliotecas

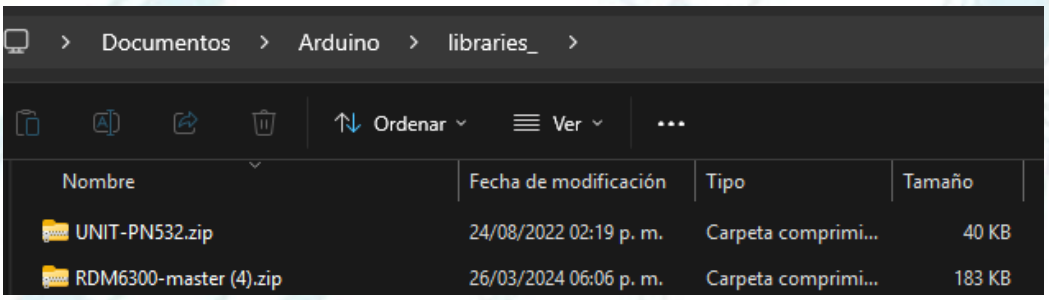
En la versión 2.3.4 del IDE Arduino, podrás tener fácil acceso a la instalación de Bibliotecas (Library) desde el costado superior izquierdo de tu pantalla.



Si la biblioteca no está incluida en la selección por default del IDE, podrás hacer la carga de la misma por la siguiente ruta (previamente haber descargado la librería en formato .zip)



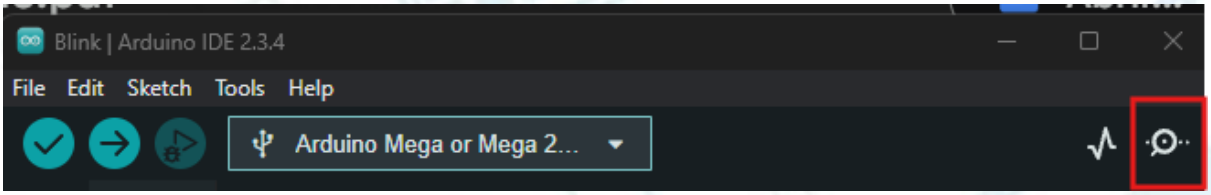
Selecciona el archivo .zip acorde a la librería que requieras cargar al IDE Arduino



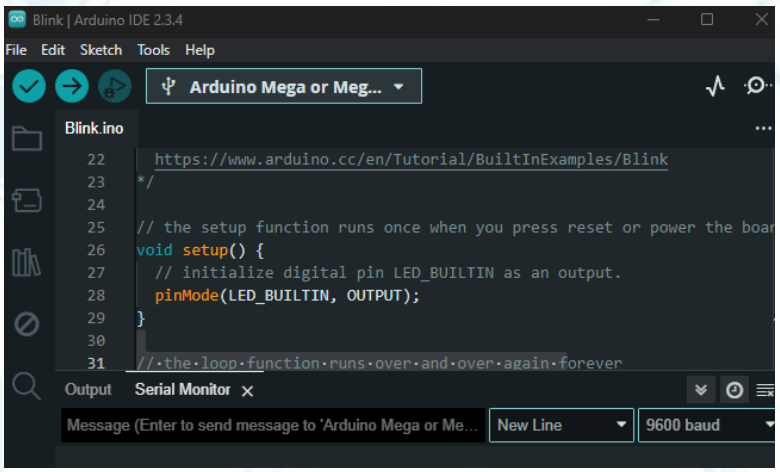
Uso del Monitor Serie

El Monitor Serie del IDE Arduino es una herramienta que permite la comunicación entre la placa Arduino y el ordenador.

Esta sección nos ayuda a monitorear el comportamiento del hardware e interactuar con el código en ejecución. En la versión 2.3.4 del IDE Arduino se podrá encontrar el acceso rápido en el extremo superior derecho.



El icono que se encuentra de lado izquierdo, es también un visualizador de los datos, pero en forma de gráfica. Por el momento solo basta con el Monitor Serial, el cual aparecerá en la parte inferior de la ventana



En esta sección se podrá enviar y recibir datos así como mensajes de depuración, valores de sensores, etc. Adicional se podrá configurar la velocidad de comunicación (baud rate) que nos ayudará a ajustar la velocidad de transmisión de datos para que coincida con la configurada en el código de Arduino. De momento se colocó 9600 baud.

Lección 1 LED Intermitente

En esta lección, aprenderás a utilizar la tarjeta Arduino Mega 2560 para que el LED parpadee en intervalos de tiempo random.

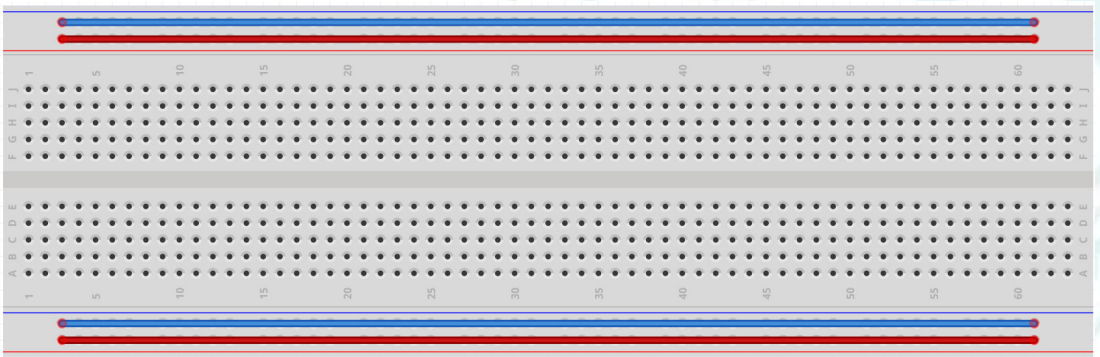
Materiales necesarios

- Arduino Mega 2560
- Led rojo 5mm
- Resistencia de 220 ohm
- Protoboard
- Cables Dupont Macho-Macho

Conocimientos previos

Protoboard

Una placa de pruebas o también llamada protoboard es un pequeño tablero lleno de agujeros unidos electrónicamente y sirve para interconectar diferentes componentes, cuenta buses de alimentación y una serie de agujeros acomodados en **filas (1 al 63)** y **columnas(A - J)** con un espacio estándar de 2.54mm.

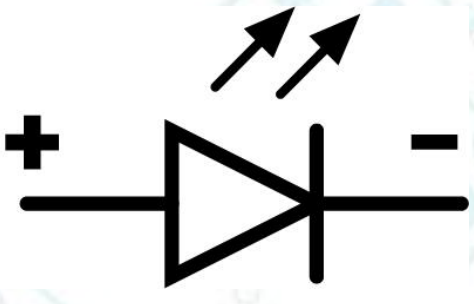


1. **Buses de alimentación:** Los buses se localizan en ambos extremos del protoboard, se representan por las líneas rojas (buses positivos o de voltaje) y azules (buses negativos o de tierra) no existe conexión física entre ellas. La fuente de alimentación generalmente se conecta aquí.
2. **Filas:** Los puntos que se encuentran consecutivos en la misma fila están interconectados entre sí, teniendo continuidad.
3. **Columnas:** Los puntos que se encuentran juntos de manera horizontal están aislados entre sí.

LED

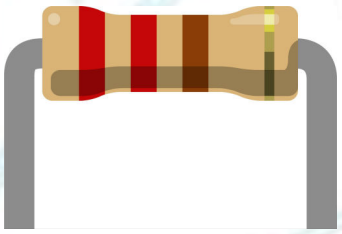
Led (Light Emisor Diode) es un diodo semiconductor capaz de emitir luz cuando se le aplica tensión, ,es decir, convierte energía eléctrica en energía luminosa. Se debe conectar de forma correcta donde, la terminal más larga es positiva (+) y la terminal más corta es negativa (-); otra forma de diferenciar la terminal negativa, el led tiene un lado plano para indicar que esa es la terminal negativa.

Normalmente la tensión utilizada en los leds es de 1V a 3V, si la tensión es superada podrías dañar el led. Para evitar eso es conveniente usar una resistencia.

Polaridad	Símbolo
	

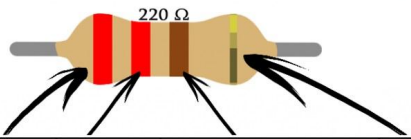
Resistencia

Una resistencia es un componente electrónico pasivo que sirve para limitar la corriente en un circuito eléctrico o ayudar al correcto funcionamiento de otros componentes electrónicos. La unidad de medida es Ohmio que se representa con la letra griega omega (Ω).

Componente	Símbolo
	

Las resistencias no tienen polaridad. Pueden ser conectadas de cualquier manera.

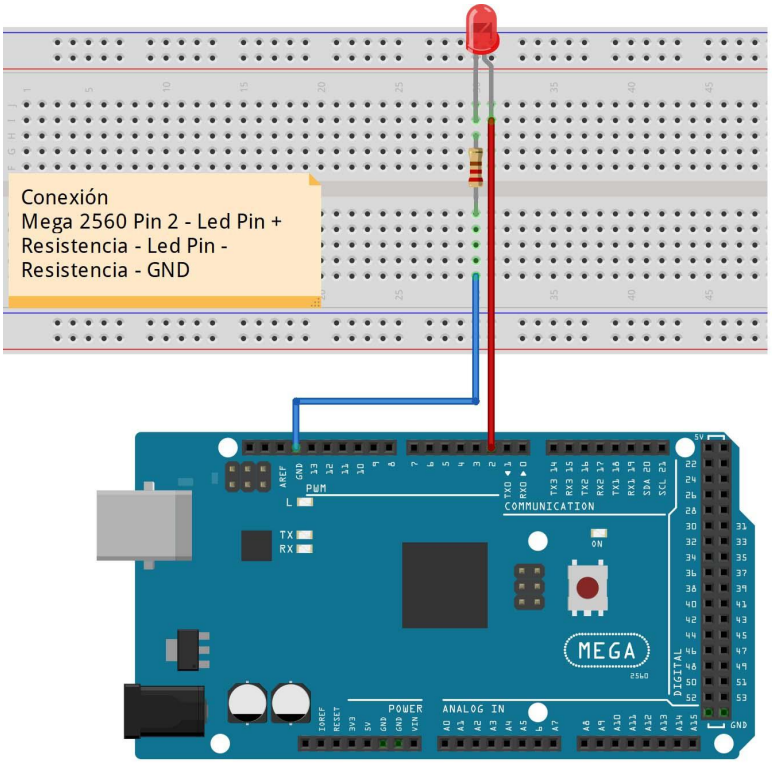
Las resistencias incorporan bandas de colores que indican cuál es su valor. Para saber el valor de una resistencia se utiliza el código de colores, que facilita la identificación y el valor de las resistencias de acuerdo al color que incorporan en sus bandas. En la siguiente imagen se muestra el código de colores:

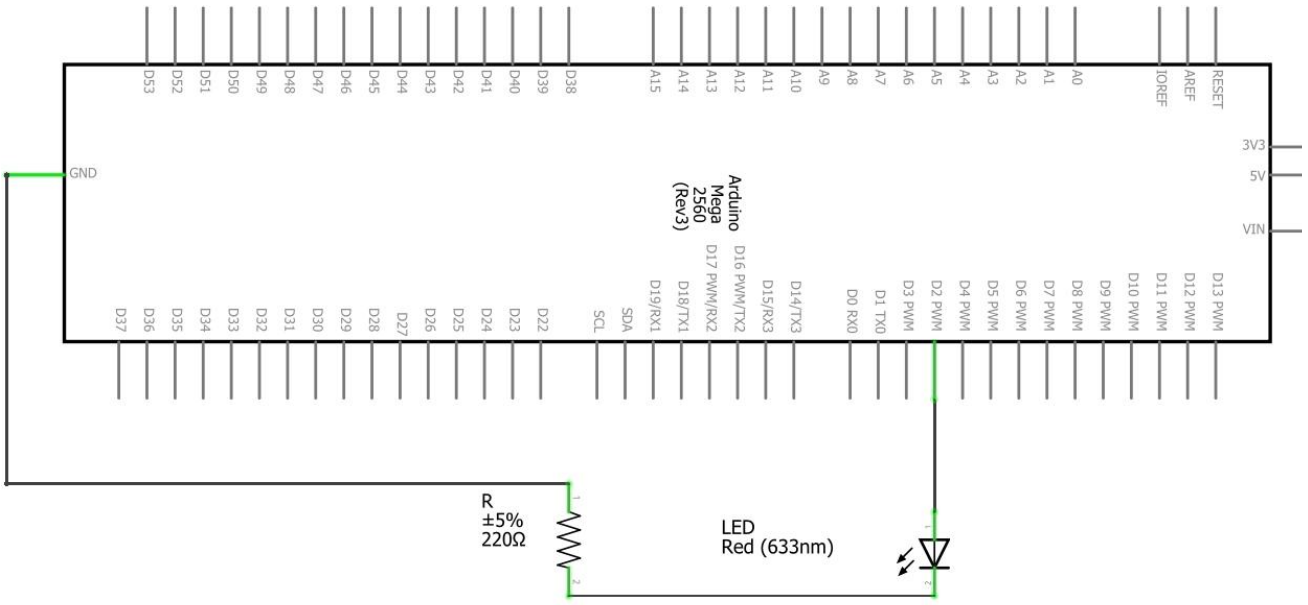


Color	Banda 1	Banda 2	Banda 3 (multiplicadora)	Tolerancia
Negro	0	0	x1	
Café	1	1	x10	1%
Rojo	2	2	x100	2%
Naranja	3	3	x1000	
Amarillo	4	4	x10000	
Verde	5	5	x100000	0.5%
Azul	6	6	x1000000	0.25%
Morado	7	7	x10000000	0.10%
Gris	8	8	x100000000	0.05%
Blanco	9	9	x1000000000	
				Dorado 5%
				Plata 10%

Diagrama de conexión

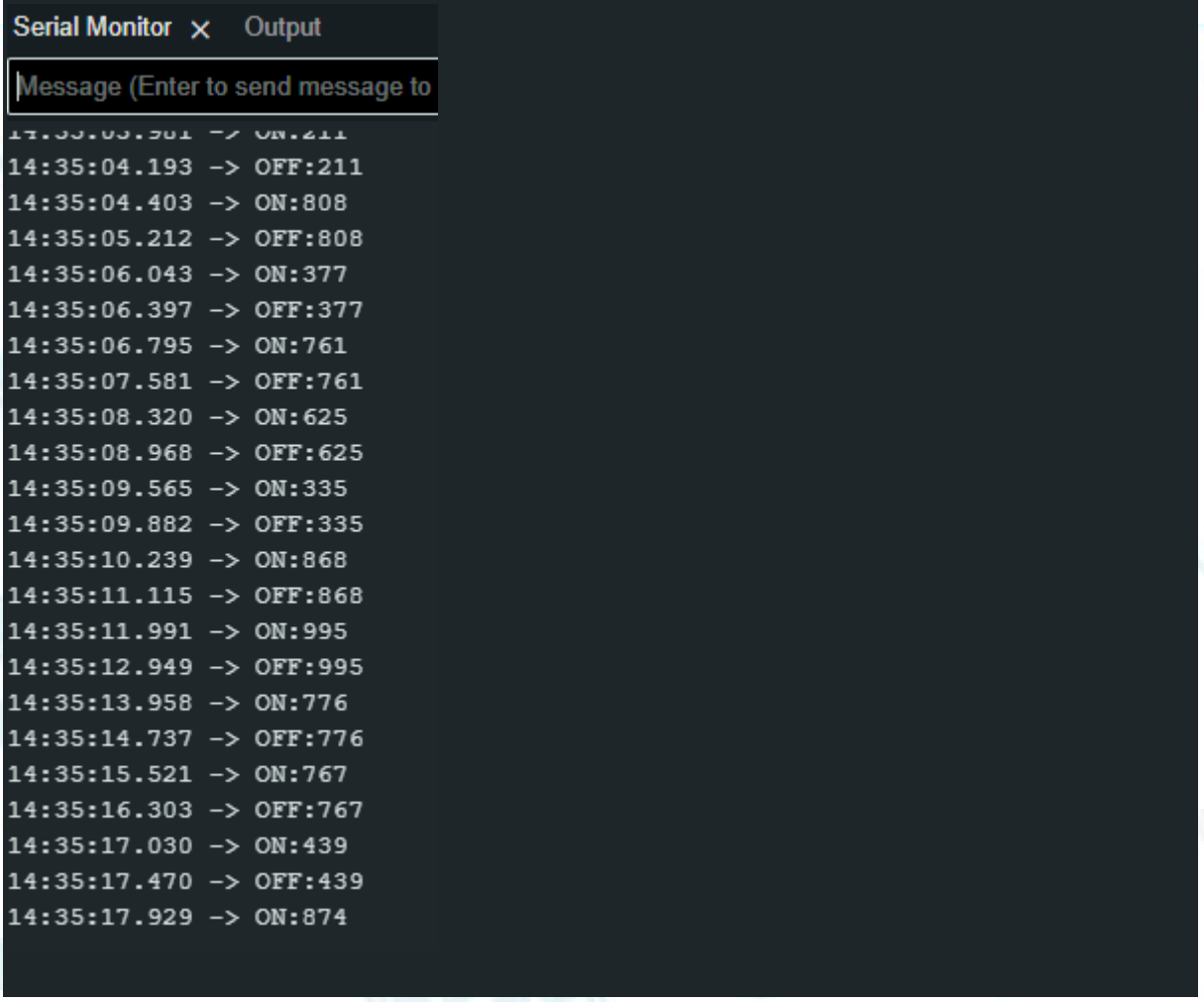
A continuación se muestran las conexiones entre la resistencia y led en la protoboard.





Código de funcionamiento

```
int led= 2;  //digital pin2  del arduino mega

int ranNum;  // se define la variable entero con nombre ranNum para almacenar el valor random

void setup() {

Serial.begin(9600);

pinMode(led,OUTPUT);    //definición de pin 2 como salida

}

void loop() {

  ranNum = random(0, 1000);  //uso de la función random colocando un rango de valores donde podrá variar de 0 a 1000

//impresión en el monitor serial del valor que será tomado para encender el led en un tiempo de ms

  Serial.print("ON:");

  Serial.println(ranNum);

  //El led encenderá en el intervalo de ms que el valor ranNum defina

  digitalWrite (led,HIGH);

  delay(ranNum);

     //impresion en el monitor serial del valor que será tomado para encender el led en un tiempo de ms

  Serial.print("OFF:");

  Serial.println(ranNum);

  digitalWrite(led,LOW);

  //El led apagará en el intervalo de ms que el valor ranNum defina

  delay(ranNum); }
```


Serial Monitor
Output
Message (Enter to send message to
14:35:04.193 -> OFF:211
14:35:04.403 -> ON:808
14:35:05.212 -> OFF:808
14:35:06.043 -> ON:377
14:35:06.397 -> OFF:377
14:35:06.795 -> ON:761
14:35:07.581 -> OFF:761
14:35:08.320 -> ON:625
14:35:08.968 -> OFF:625
14:35:09.565 -> ON:335
14:35:09.882 -> OFF:335
14:35:10.239 -> ON:868
14:35:11.115 -> OFF:868
14:35:11.991 -> ON:995
14:35:12.949 -> OFF:995
14:35:13.958 -> ON:776
14:35:14.737 -> OFF:776
14:35:15.521 -> ON:767
14:35:16.303 -> OFF:767
14:35:17.030 -> ON:439
14:35:17.470 -> OFF:439
14:35:17.929 -> ON:874

Lección 2 RGB LED Intermitente

En esta lección, aprenderás a utilizar LED RGB con la tarjeta Arduino Mega 2560

Materiales necesarios

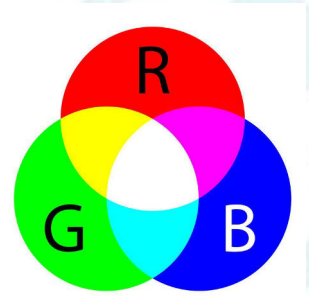
- Arduino Mega 2560
- Led RGB
- Resistencias de 220 ohm
- Protoboard
- Cables Dupont Macho-Macho

Conocimientos previos

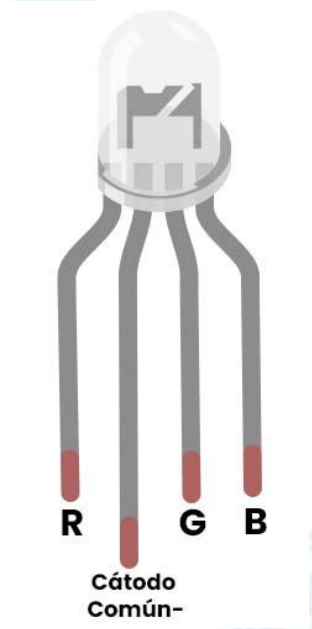
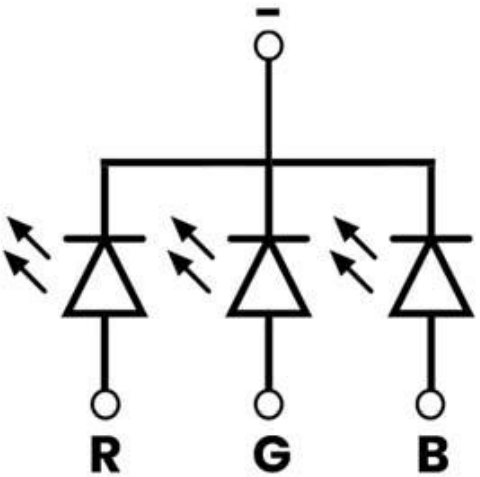
LED RGB

RGB es un sistema de color basado en tres colores primarios Rojo, Verde y Azul (por sus siglas en inglés (R) Red, (G) Green, (B) Blue), el cual en base a la combinación de estos 3 colores podemos representar otros colores como se muestra a continuación.

Por lo tanto, un Led RGB está formado internamente por tres diodos emisores de luz, uno rojo, uno verde y por último uno azul, con el propósito de poder crear, gran variedad de colores mediante la combinación de cada color con intensidades distintas.



Los Led´s RGB tiene 4 terminales y es común utilizar el encapsulado de 5mm y hay 2 tipos de cátodo o ánodo común, que define cómo está conformado internamente el led RGB, (como se muestra a continuación), aunque en esta lección vamos a trabajar con un LED RGB de cátodo común.

Componente	Símbolo
	

El control se recomienda que se haga por medio de PWM, de tal forma que, la resolución del PWM en cada color es de 8 bits, tendremos la posibilidad de generar más de 16 millones de colores distintos.

De esta manera el led RGB brillará con una intensidad de color determinada en base al voltaje que se le suministre a cada diodo led interno, pudiéndose representar cualquier color derivado de los primarios.

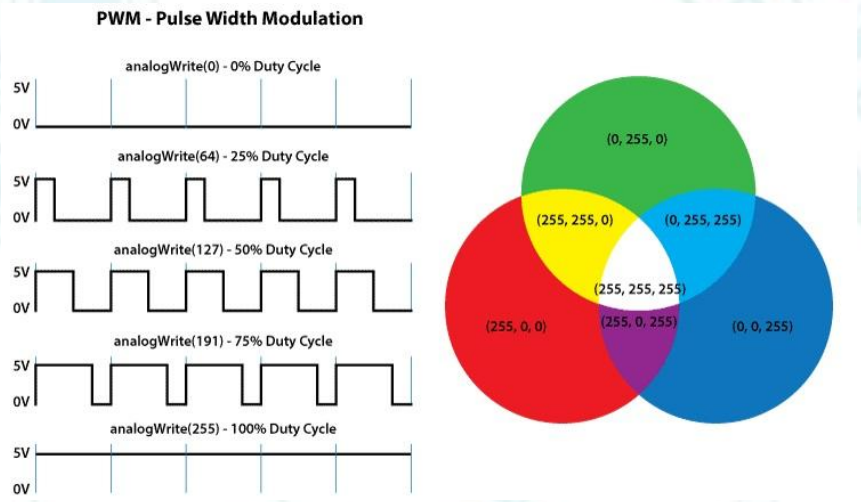
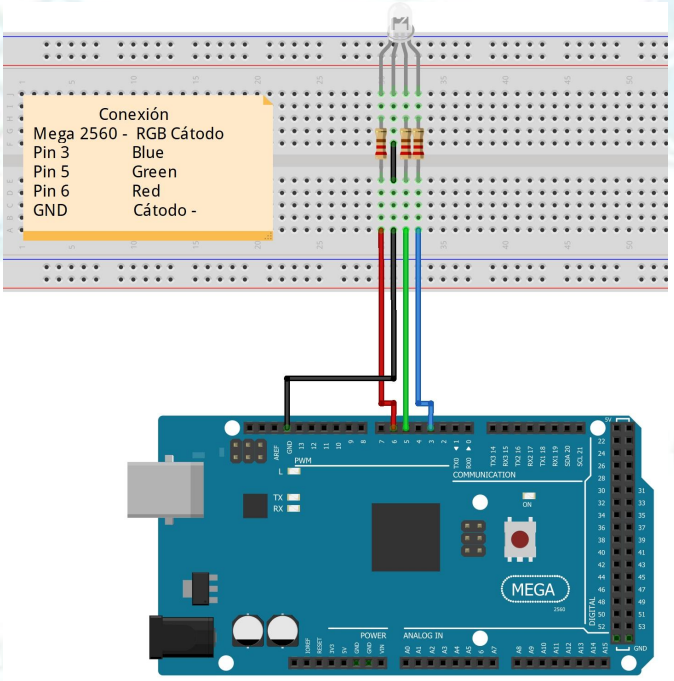
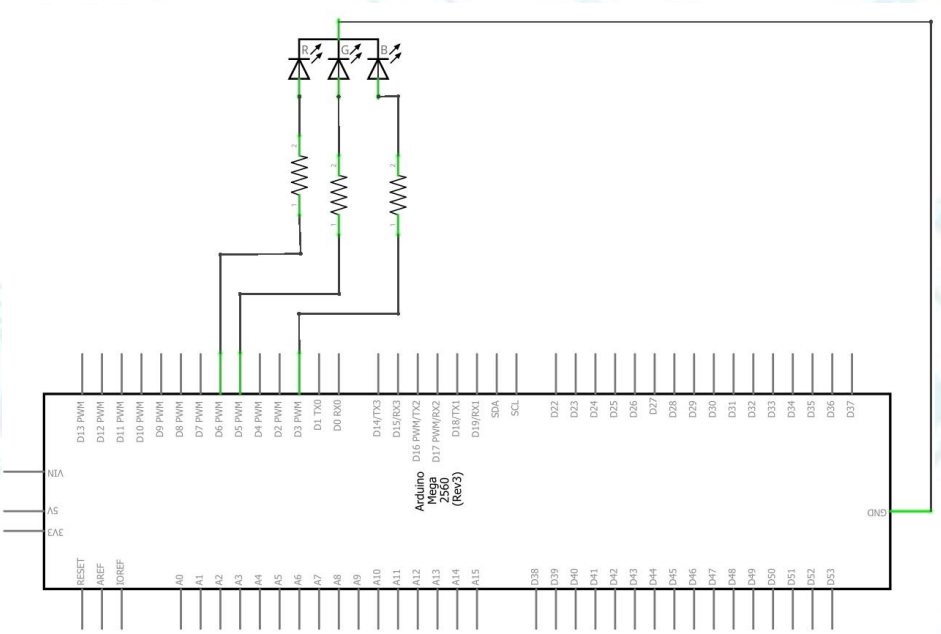


Diagrama de conexión

A continuación se muestran las conexiones entre la resistencia y led en la protoboard.





Código de funcionamiento

```
int rojoPin= 3;           // El control del color Rojo por el pin 3

int verdePin = 5;     //El control del color Verde por el pin 5

int azulPin = 6;        //El control del color Azul por el pin 6

int ranNum; // se define la variable entero con nombre ranNum para almacenar el valor random Se definen como salidas a los pines 3, 5, 6; correspondientes al color del RGB

void setup() {

 pinMode(rojoPin, OUTPUT);

 pinMode(verdePin, OUTPUT);

pinMode(azulPin, OUTPUT);

}

void loop() {

// Valor del ciclo de trabajo PWM (0 a 255), que nos ayudará a tener control de la intensidad de brillo por cada uno de los colores (rojo, verde, azul), con ayuda de la función "setColor"

 ranNum = random(0, 1000); //uso de la función random colocando un rango de valores donde podrá variar de 0 a 1000

setColor(255, 0, 0);//Brillará en color Rojo

delay(ranNum); //Instrucción que durará entre 0 a 1000ms, dependiendo el valor random

setColor(0, 255, 0); //Brillará en color Verde

delay(ranNum);//Instrucción que durará entre 0 a 1000ms, dependiendo el valor random

setColor(0, 0, 255); //Brillará en color Azul

delay(ranNum);//Instrucción que durará entre 0 a 1000ms, dependiendo el valor random si colocamos el PWM de todos los colores en máximo

setColor(255, 255, 255);  //Brillara en Blanco

delay(ranNum);//Instrucción que durará entre 0 a 1000ms, dependiendo el valor random, si realizamos combinación de los valores podremos obtener más gama de colores
```

```
setColor(255, 0, 255);    // Brillará en color Morado

delay(ranNum);//Instrucción que durará entre 0 a 1000ms, dependiendo el valor random

}

//Función "setColor()" tendremos acceso al control de los pines PWM, mostrando a la salida los colores que previamente indicamos, por medio del argumento (rojo, verde, azul); que serán valores de entra: rojoValue, verdeValue and azulValue.

void setColor(int rojoValue, int verdeValue, int azulValue) {

  analogWrite(rojoPin, rojoValue);

  analogWrite(verdePin, verdeValue);

  analogWrite(azulPin, azulValue);

}
```

Lección 3 Control de un LED por un botón

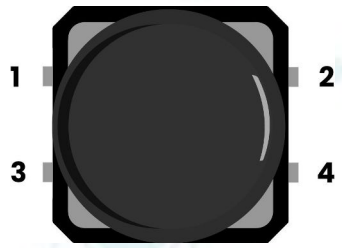
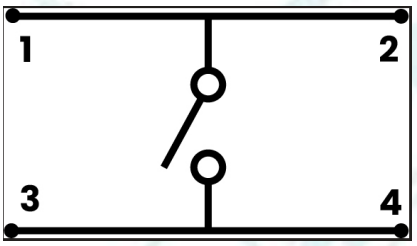
En esta lección, aprenderás cómo utilizar pulsadores con entradas digitales para tener control de encendido y apagado de un led**.**

Materiales necesarios

- Arduino Mega 2560
- Protoboard
- Led de 5mm rojo
- Resistencia de 220 Ohms
- Push Button 4 pines
- Cables Dupont Macho-Macho

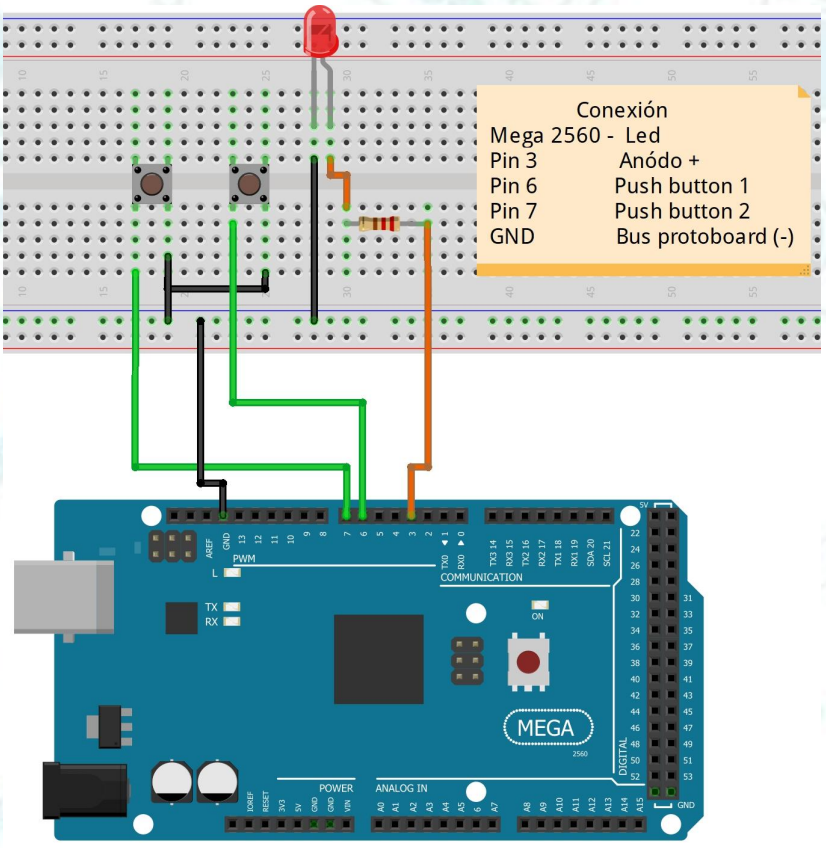
Conocimientos previos

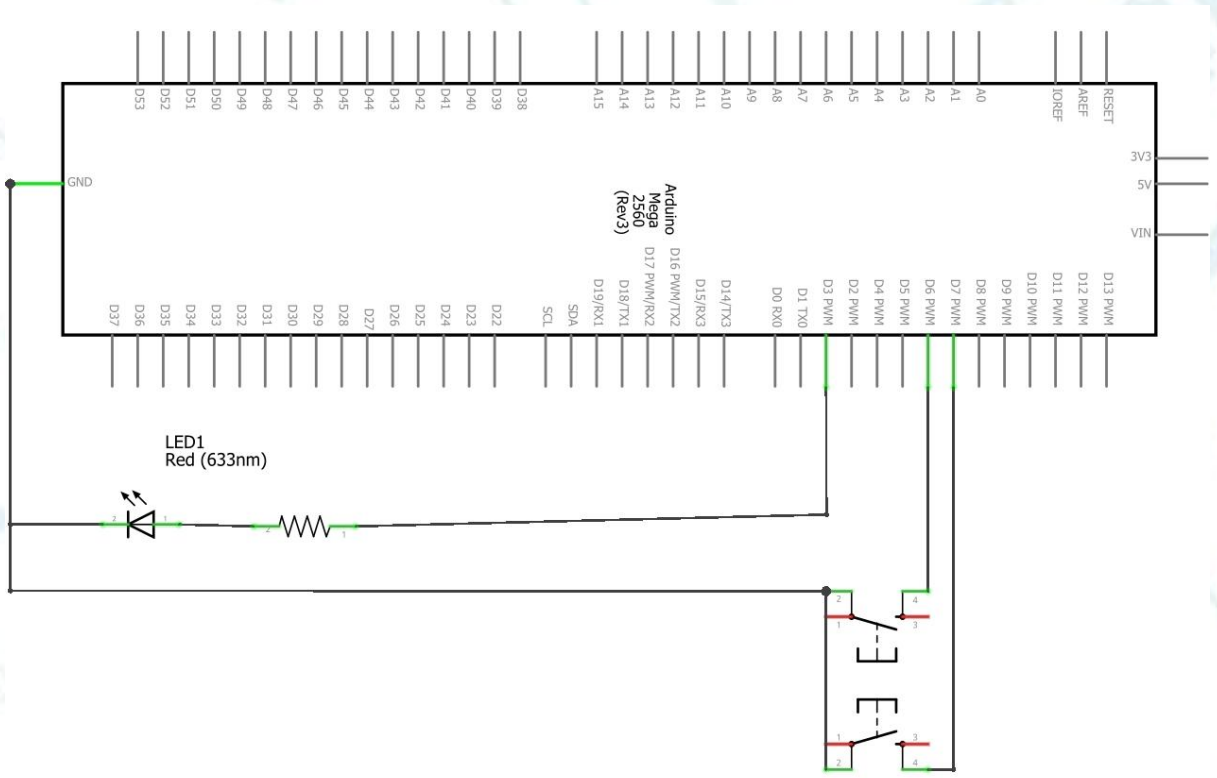
Push Botton 4 pines es un dispositivo táctil que sirve como interruptor ya que puede ser activado, al ser pulsado con el dedo y permiten el flujo de corriente mientras es accionado.

Componente	Símbolo
	

Para realizar el circuito en el protoboard se utilizará 2 botones, 1 para encender el led y otro para apagarlo, también se utilizará una resistencia de 220 Ohms para proteger al led.

Diagrama de Conexión





Código de funcionamiento

```
int ledPin = 3; // Asignación del Pin 10  para el Led

int onApin = 6; //Asignación del Pin 9 para el push button que nos ayudará a encender el Led

int offBpin = 7;//Asignación del Pin 8 para el push button que nos ayudará a apagar el Led

//Se declara una variable leds para controlar en estado, actual del led (on/off)Tipo byte (números que representan 8 bits)

byte leds = 0;

void setup()

{

  pinMode(ledPin, OUTPUT); //Declaración del pin de salida a ledPin / pin 10, declaración de los push button en modo PULL UP (),Podamos conectar directamente al Arduino sin apoyo de una resistencia física

  pinMode(onApin, INPUT_PULLUP);

  pinMode(offBpin, INPUT_PULLUP);

}

void loop()

{

//Realizaremos lectura de la entrada digital del pin 9 / pin 8 y realizando acciones dependiendo//el estado en que se encuentre

  if (digitalRead(onApin) == LOW) {

  //Si se presiona el botón del pin 9 que se encuentra en nivel //Alto pasará a nivel Bajo y por consiguiente...

 digitalWrite(ledPin, HIGH);  //El led se encenderá

 }

 if (digitalRead(offBpin) == LOW) //Si se presiona el botón del pin 8

 {

 digitalWrite(ledPin, LOW);  } //El led se apagará

}
```

Lección 4 Buzzer Activo

En esta lección aprenderemos a utilizar un buzzer activo para generar un tono mediante un pin digital de la tarjeta desarrollo Arduino Mega 2560.

Materiales necesarios

- Arduino Mega 2560
- Protoboard
- Zumbador activo
- Cables Dupont Macho-Macho

Conocimientos previos

Buzzer activo es un pequeño dispositivo que puede producir un sonido cuando se aplica una corriente directa DC. Por ejemplo, si se conecta directamente a los 5V de la tarjeta MEGA 2560, este genera un tono fijo con una frecuencia ya predeterminada debido a que incorpora un oscilador interno.

El buzzer activo tiene 2 terminales, negativo y positivo, comúnmente incorporan una etiqueta que nos indica cual es la terminal positiva mediante el símbolo (+), la terminal que se encuentre más cerca a esta referencia será la terminal positiva.

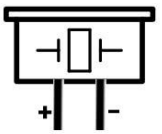
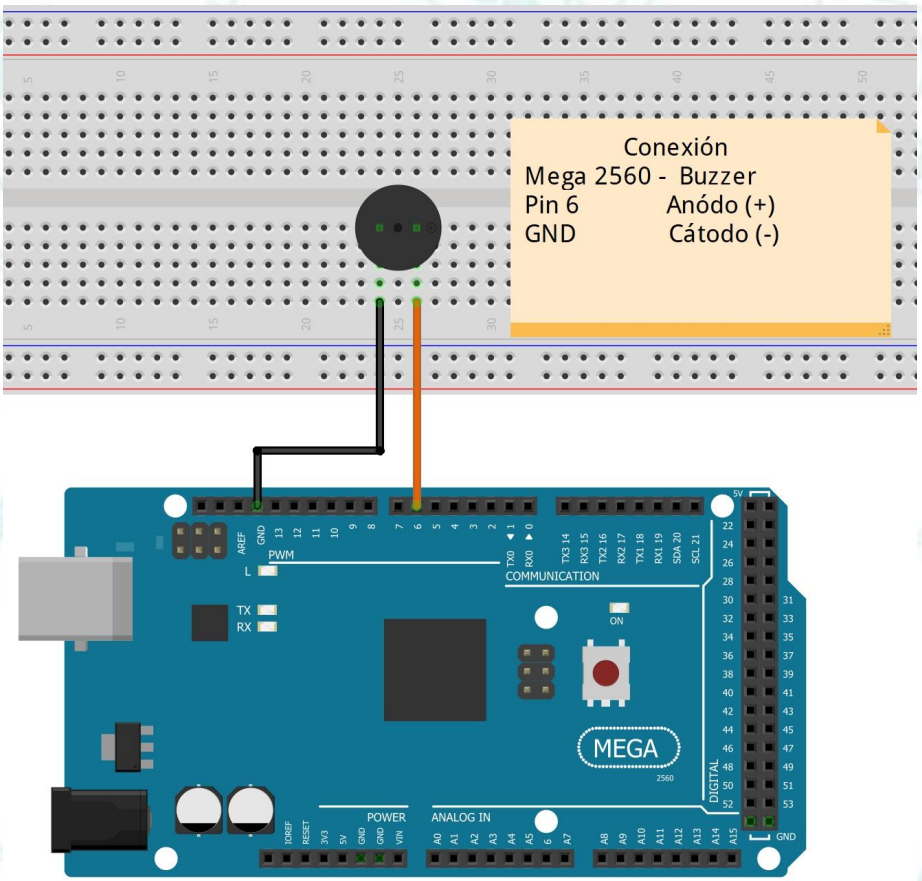
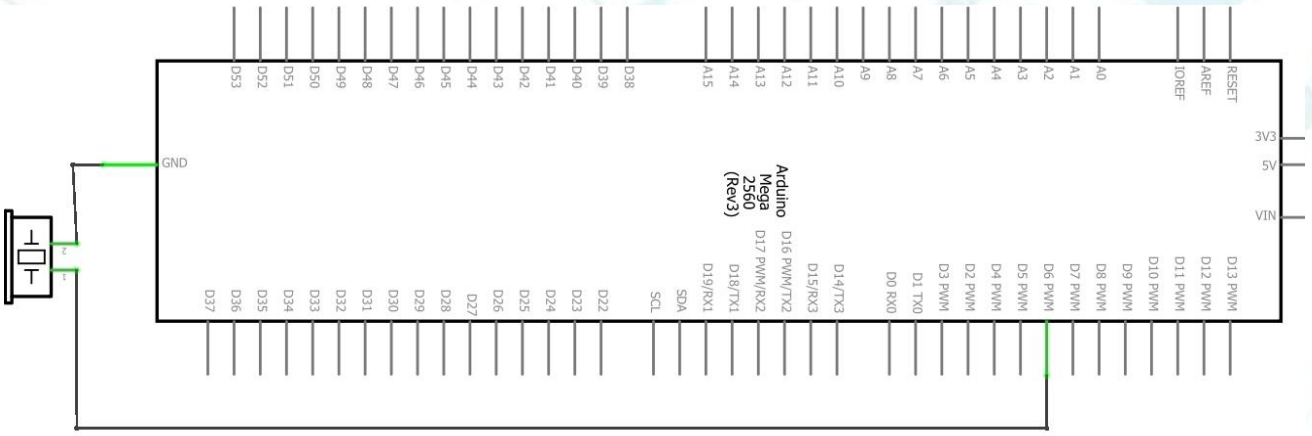
Componente	Símbolo
	

Diagrama de conexión





Código de funcionamiento

```
//Encendido y control de tono por medio de la tarjeta Uno R3

int buzzer = 6;    // Pin 12 para el buzzer activo

void setup()

{

 pinMode(buzzer,OUTPUT); //Declaración de pin de salida asignado al buzzer

 Serial.begin(9600);

}

void loop(){

unsigned char i;                          //Variable tipo carácter sin signo
llamada "i" para simular una  frecuencia

    for(i=0;i<80;i++){                //Generación de frecuencia de pitidos

        digitalWrite(buzzer,HIGH);    //El buzzer emitirá un pitido...

          delay(10);                        //...por 50ms, variando este tiempo
puedes cambiar la frecuencia del pitido

        digitalWrite(buzzer,LOW);     //El buzzer se apagará...

        delay(10);                    //...por 50ms

      }

  }
```

Lección 5 Buzzer Pasivo

En esta lección aprenderemos a utilizar el buzzer pasivo y la tarjeta MEGA 2560 para generar ocho sonidos diferentes, cada sonido durará 1 segundos desde Do (262Hz), Re (294Hz), Mi (330Hz), Fa (349Hz), Sol (392Hz), La (440Hz), Si (494Hz) hasta Do Agudo (524Hz).

Materiales necesarios

- Arduino Mega 2560
- Protoboard
- Zumbador pasivo
- Cables Dupont Macho-Macho

Conocimientos previos

Un buzzer pasivo es un dispositivo electromecánico que produce sonido cuando se le aplica una señal eléctrica alterna. A diferencia de un buzzer activo (como el que se utilizó en la práctica anterior), que tiene un oscilador interno y solo necesita una tensión continua para funcionar, el buzzer pasivo requiere una señal de frecuencia, la cual, generamos por medio del Mega 2560.

Es más versátil en términos de control de tono y frecuencia, pero necesita un circuito adicional para generar la señal adecuada. Se usa comúnmente en alarmas, timbres y sistemas de notificación.

Para cambiar la frecuencia del tono y generar diferentes melodías se recomienda aplicar una señal de onda cuadrada con una frecuencia específica, para este caso se recomienda utilizar en programación la función Tone () la cual permitirá definir 3 parámetros, el pin del buzzer la frecuencia y la duración de la nota.

El buzzer pasivo tiene 2 terminales, negativo y positivo, para identificar la terminal positiva el buzzer comúnmente incorpora en la parte superior una referencia con el símbolo (+), la terminal que se encuentre más cerca a esta referencia será la terminal positiva.

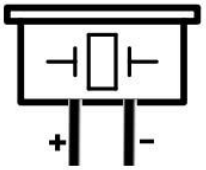
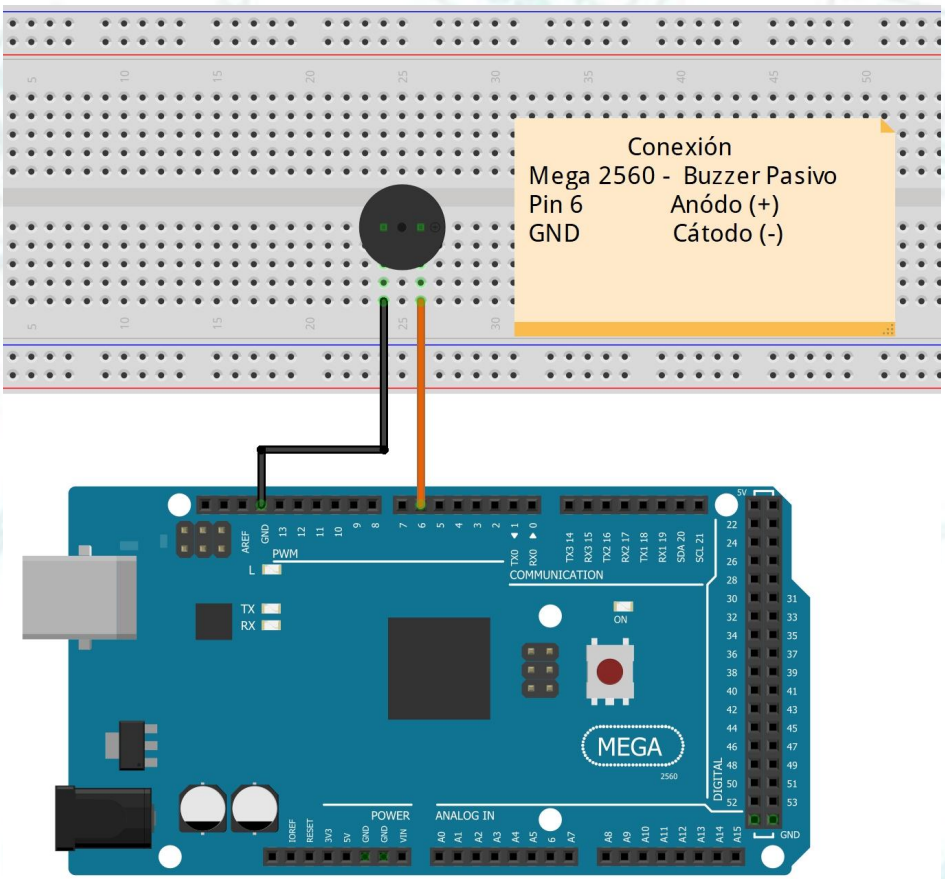
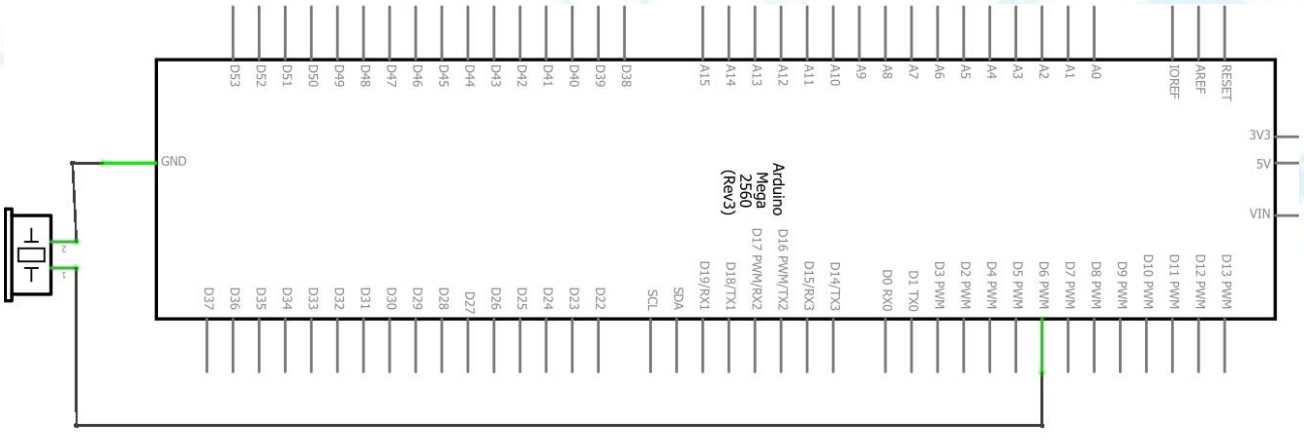
Componente	**Símbolo**
	

Diagrama de conexión





Código de funcionamiento

```
int Do = 262, Re = 294, Mi = 330, Fa = 349, Sol = 392, La = 440, Si = 494, Do2 = 524;  // valor de las frecuencia de cada una de las notas de la escala principal de Do

int buzz = 6;   //Pin 6 para control del buzzer pasivo

int inicio = 0; //Variable para control de tiempo inicializada con valor 0

void setup(){

  pinMode(buzz, OUTPUT); }  //Inicialización del pin 12 como salida

void loop(){

inicio = 500;  //Duración de cada una de las notas será de 500ms .5seg

tone(buzz, Do, inicio);//Generación de onda cuadrada en el pin 6, DO

delay(1000);   //Espera 1s

tone(buzz, Re, inicio);//Generación de onda cuadrada en el pin 6,  RE

delay(1000);  //Espera 1s

tone(buzz, Mi, inicio); //Generación de onda cuadrada en el pin 6,  MI

delay(1000);  //Espera 1s

tone(buzz, Fa, inicio); //Generación de onda cuadrada en el pin 6, FA

delay(1000);  //Espera 1s

tone(buzz, Sol, inicio);//Generación de onda cuadrada en el pin 6, SOL

delay(1000);   //Espera 1s

tone(buzz, La, inicio); //Generación de onda cuadrada en el pin 6, LA

delay(1000);    //Espera 1s

tone(buzz, Si, inicio);//Generación de onda cuadrada en el pin 6, SI

delay(1000);// Espera 1s

tone(buzz, Do2, inicio); //Generación de onda cuadrada en el pin 6,DO sostenido

delay(1000); //Espera 1s

noTone(buzz); } //Terminara la reproducción de sonido
```

Lección 6 Ocho Led Controlados con el 74HC595

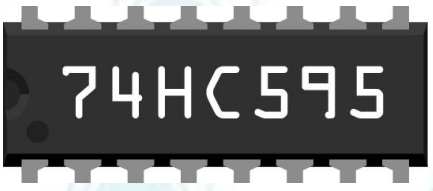
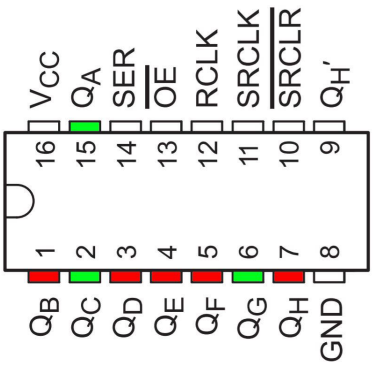
En esta lección aprenderemos a utilizar el circuito 74HC595 para encender 8 led mediante la utilización de 3 pines de la tarjeta MEGA 2560. Ya que el CI 74HC595 es un registro de desplazamiento que permite ampliar el número de pines de E/S de la tarjeta MEGA 2560.

Materiales necesarios

- Arduino Mega 2560
- Protoboard (Placa de pruebas)
- Led de 5mm
- Resistencia de 220 Ohms
- IC 74HC595
- Cables Dupont Macho-Macho

Conocimientos previos

El IC 74HC595 o SN74HC595N es un registro de desplazamiento de 8 bits con salida en serie y paralelo . Permite controlar múltiples salidas (como LEDs o displays) utilizando solo unos pocos pines de un microcontrolador.

Componente	Símbolo
	

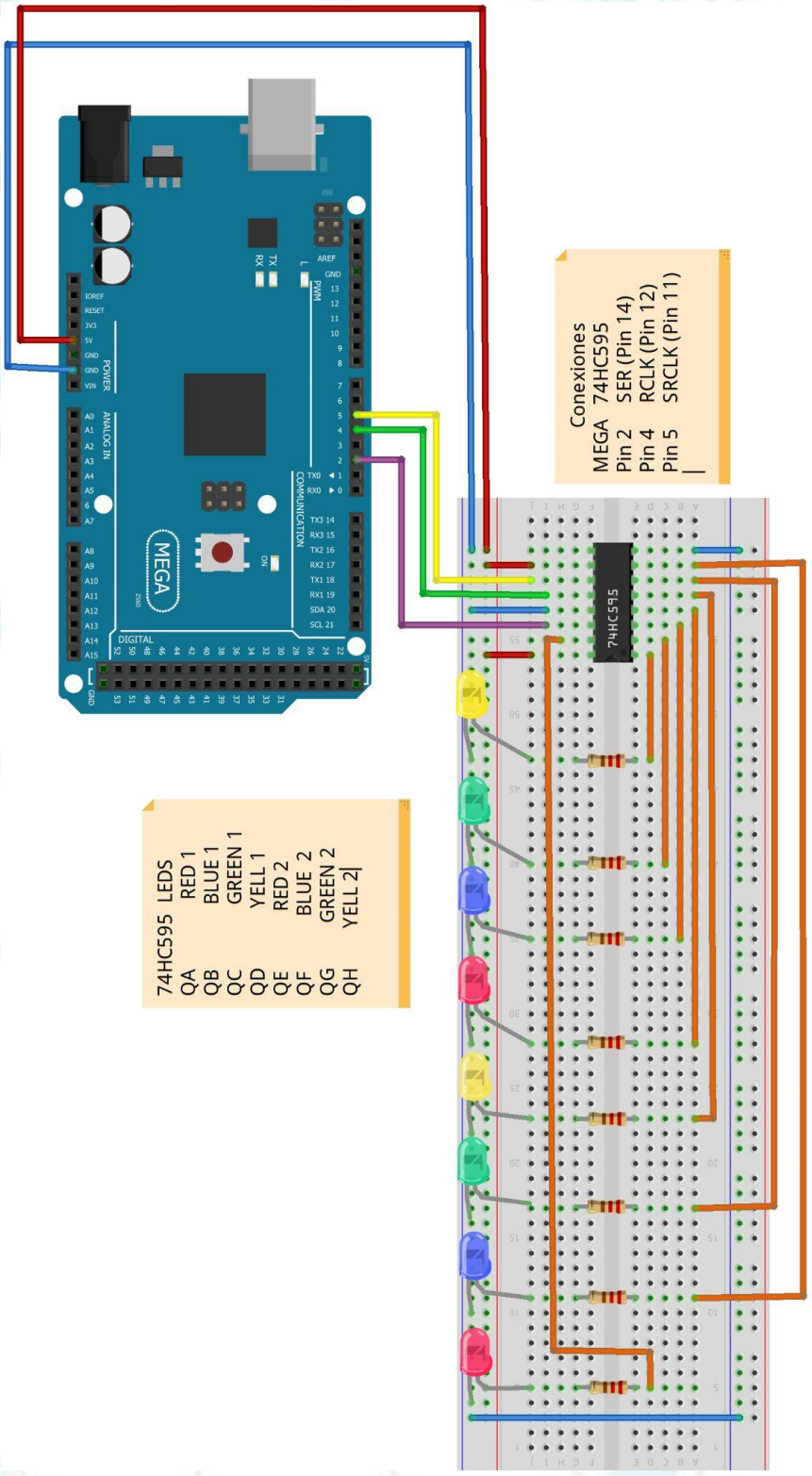
Funciona recibiendo datos en serie a través de un pin de entrada (DS) y desplazándose internamente mediante pulsos de reloj (SHCP). Una vez cargados los 8 bits, se transfieren a los pines de salida paralela mediante un pulso en el pin de almacenamiento (STCP). También tiene una salida en serie (Q7') para conectar varios ICs en cascada y ampliar el número de salidas. Es ampliamente usado en proyectos de electrónica para ahorrar pines de control.

El chip contiene ocho pines que podemos utilizar como salida, cada uno de ellos está asociado con un bit en el registro. En el caso del CI 74HC595, nos referimos a ellos desde QA hasta QH.

Para escribir estas salidas con la tarjeta UNO R3, tenemos que enviar un valor binario al registro de desplazamiento y con ese número el registro de desplazamiento puede comprender qué salidas utilizar.

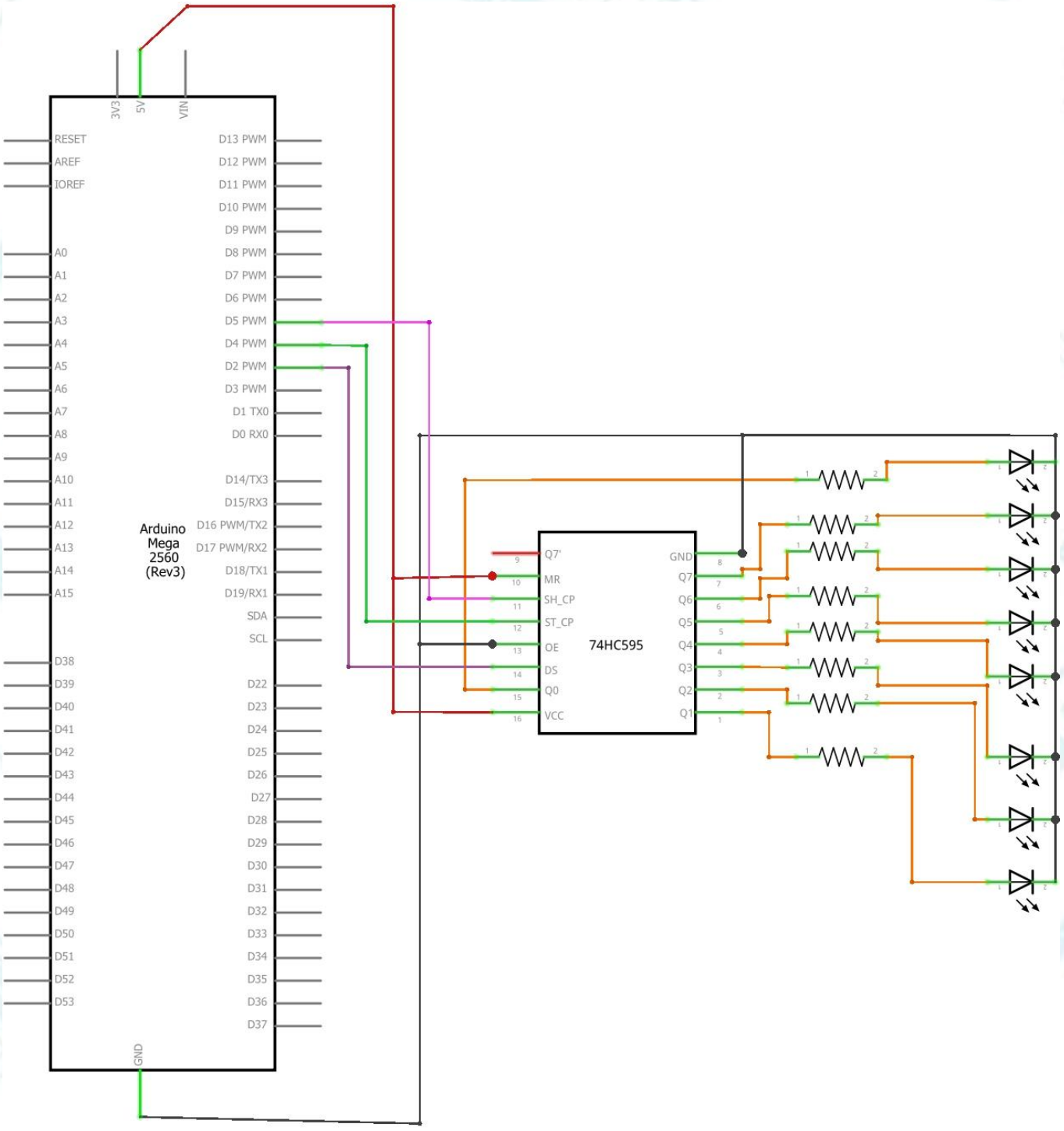
Por ejemplo, si enviamos el valor binario 10100010, los pines destacados en verde en la imagen siguiente estarán activos y los destacados en rojo permanecerán inactivos.

Diagrama de conexión



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

```
int RCLKPin = 4;   //Usaremos el Pin 4 de la tarjeta UNO R3 para el manejo del Reloj de Registro (Pin 12 RCLK del 74HCC595).

int SRCLKPin = 5;  //Usaremos el Pin 5 de la tarjeta UNO R3 para el manejo del Reloj de Registro de desplazamiento (Pin 11 SRCLK del 74HCC595).

int datoPin = 2;    //Usaremos el Pin 2 de la tarjeta UNO R3 para entrada Serial del Dato (Pin 14 SER del 74HCC595).

byte leds = 0; //Variable llamada leds con espacio de 8 bits, donde se guardará el patrón de encendido/apagado de leds

int nextLED = 0;   //Variable para detectar el byte en la variable leds

void setup() {  //Declaración de variables, como salidas digitales

  pinMode(RCLKPin, OUTPUT);

  pinMode(datoPin, OUTPUT);

  pinMode(SRCLKPin, OUTPUT);

  leds = 0; }  //Inicialización del programa y de la variable tipo byte en 0

void loop() {

leds = 0; //Limpieza del bucle de la variable byte en 0, para que el led encienda una vez

 if (nextLED == 7) { //Se comienza el corrimiento de los bits 0, si se llega a

    nextLED = 0; } //Reiniciamos el avance en 0

  else {

    nextLED++;  }//De lo contrario seguirá avanzando hasta llegar a 8 bits

   bitSet(leds, nextLED); //Usará la función bitSet para el control de c/led dependiendo el avance del bit

   digitalWrite(RCLKPin, LOW); // Pone en nivel bajo a latchPin durante la transmisión

  shiftOut(datoPin, SRCLKPin, MSBFIRST, leds);

  digitalWrite(RCLKPin, HIGH);// Devuelve el latchpin a nivel alto

  delay(1000);   //Un segundo de tiempo para esta rutina

}
```

Lección 7 Fotorresistor

En esta lección aprenderás a usar una fotorresistencia, se utilizará el circuito de la lección anterior, con la diferencia de que se modulará la intensidad de luz de los LED por medio de la fotorresistencia utilizando de una entrada analógica de la tarjeta MEGA 2560.

Materiales necesarios

- Arduino Mega 2560
- Protoboard
- Led de 5mm
- Resistencia de 220 Ohms
- IC 74HC595
- Fotorresistencia LDR
- Resistencia de 1k
- Cables Dupont Macho-Macho

Conocimientos previos

Una fotorresistencia o LDR (por sus siglas en inglés “light-dependent resistor”) es un componente electrónico cuya resistencia varía en función de la luz. Se trata de un sensor que actúa como una resistencia variable en función de la luz que capta. A mayor intensidad de luz, menor resistencia.

Para conocer la cantidad de luz que el sensor capta en cierto ambiente, necesitamos medir la tensión de salida del mismo. Para lograr esto se debe realizar un divisor de tensión colocando una resistencia en serie. Hay dos formas básicas para conectar una LDR:

1. **Mayor luz, mayor voltaje:** Al conectar la fotoresistencia al nodo positivo de nuestra fuente de voltaje tendremos que, al incidir una mayor cantidad de luz provocará una menor caída de voltaje o diferencia de potencial entre la fuente y el pin de referencia (Vout), por lo tanto, se tendrá una lectura mayor.
2. **Mayor luz, menor voltaje:** En pocas palabras la fotorresistencia se conecta al nodo de GND y provocará un comportamiento opuesto al punto 1.

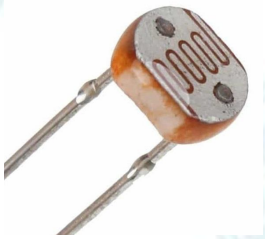
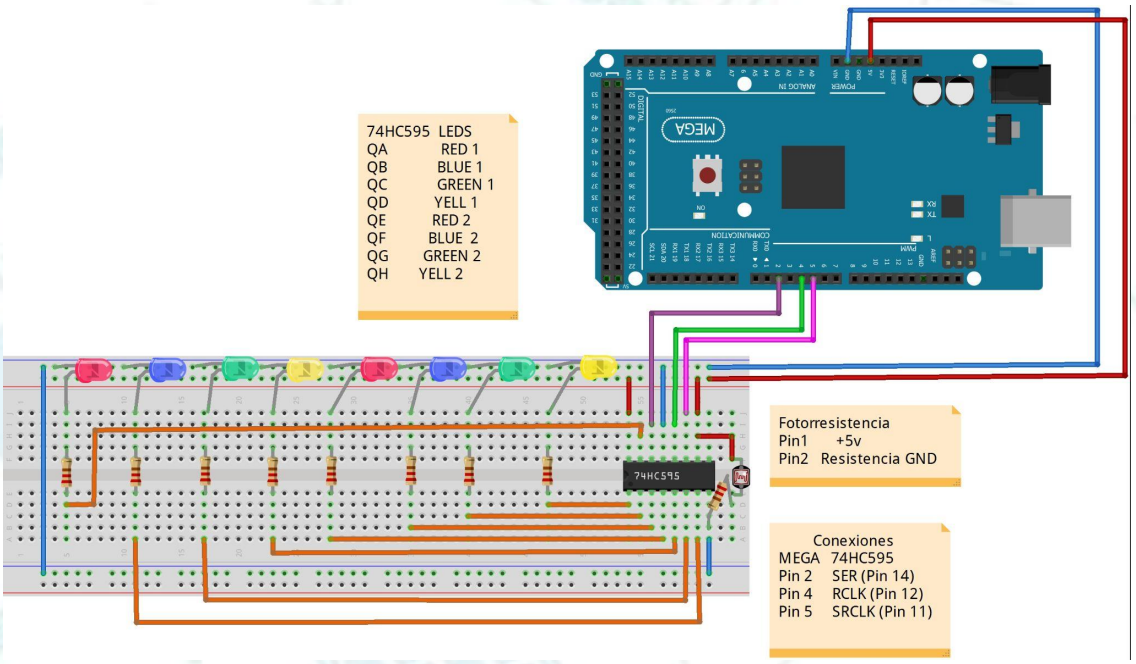
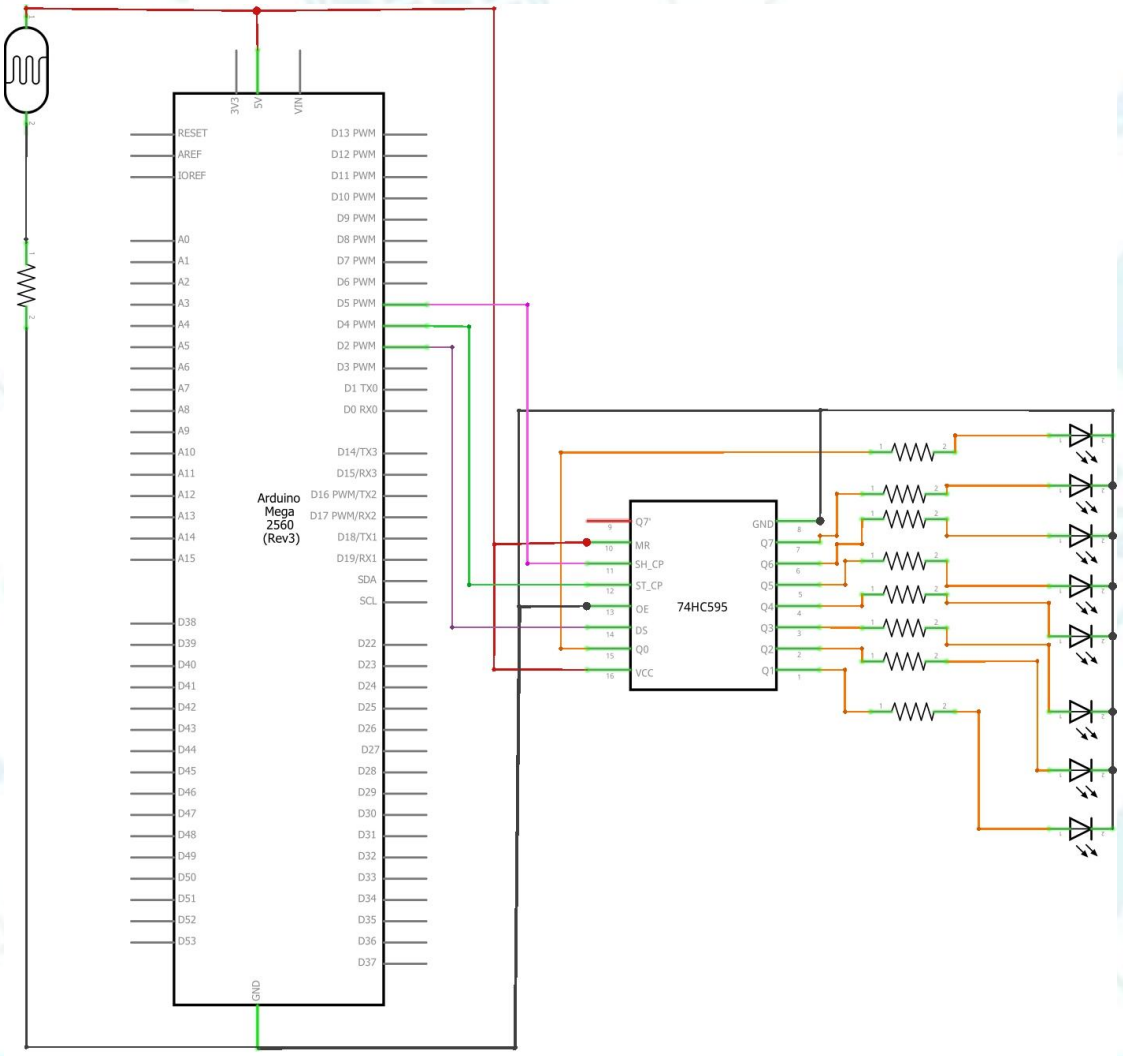
Componente	**Símbolo**
	

Diagrama de conexión





Código de funcionamiento

```
int lightPin = 0;  // Pin analógico A0 de la tarjeta UNO R3 para conectar la LDR

int latchPin = 4;  //Usaremos el Pin 11 para el manejo del Reloj de Registro (Pin 12 RCLK del 74HCC595).

int clockPin = 5;  //Usaremos el Pin 12 para el manejo del Reloj de Registro de Desplazamiento (Pin 11 SRCLK del 74HCC595)

int dataPin = 2; //Usaremos el Pin 9 para entrada Serial del Dato (Pin 14 SER del 74HCC595)

int leds = 0;  //Variable entera para el registro de leds

void setup(){ //Declaración de variables, como de salida digital

  pinMode(latchPin, OUTPUT);

  pinMode(dataPin, OUTPUT);

  pinMode(clockPin, OUTPUT);

   Serial.begin(9600); }//El valor que se obtiene la fotorresistencia es analógico entre 0 a 1023 0 (obscuro) y 1023(luz)

void loop(){

   int reading   = analogRead(lightPin); //Lectura del valor analógico del fotorresistor

  Serial.println(reading);

  int numLEDSLit = reading / 10; // 1023 entre 8 (8 bites) / 2 = ~63

   numLEDSLit  =  8  -  numLEDSLit;   //Para cambiar el sentido de cambio de encendido en los leds

   Serial.println(numLEDSLit);

  if (numLEDSLit > 8)

  numLEDSLit = 8;//Si el valor del led es mayor que 8, el valor será igual a 8

  leds = 0;  //...no habrá iluminación en los leds

   for (int i = 0; i < numLEDSLit; i++)   //tendremos un incremento de la variable i, hasta que sea menor a numLEDSLit

  {
```

```
    leds = leds + (1 << i);  // Establece el i-décimo bit

  }

  actualizaRegistro(); //Se ejecuta la función

}

void actualizaRegistro() //Función para realizar el encendido de los leds a
partir del Reloj de Registro

{

  digitalWrite(latchPin, LOW);//Pone en nivel bajo a latchPin , led apagado

    shiftOut(dataPin, clockPin, MSBFIRST, leds); //función shiftOut para
desplazar el bit,

    digitalWrite(latchPin, HIGH); //pone en nivel bajo a latchPin , led
encendido

}
```

Lección 8 74HC595 y Display de 7 segmentos

En esta lección aprenderás a utilizar el display de 7 segmentos de ánodo común con el registro de desplazamiento 74HC595.

Materiales necesarios

- Arduino Mega 2560
- Protoboard
- Resistencia de 220 Ohms
- IC 74HC595
- Display de 7 Segmentos cátodo común
- Cables Dupont Macho-Macho

Conocimientos previos

El display de 7 Segmentos es un dispositivo opto-electrónico que permite visualizar números del 0 al 9. Se utiliza para representar visualmente números y algunos caracteres. El display está compuesto por 4 dígitos, cada dígito tiene 7 segmentos y un punto decimal que pueden encender o apagar individualmente. En la siguiente imagen se muestra los pines del display de 7 Segmentos:

PINOUT

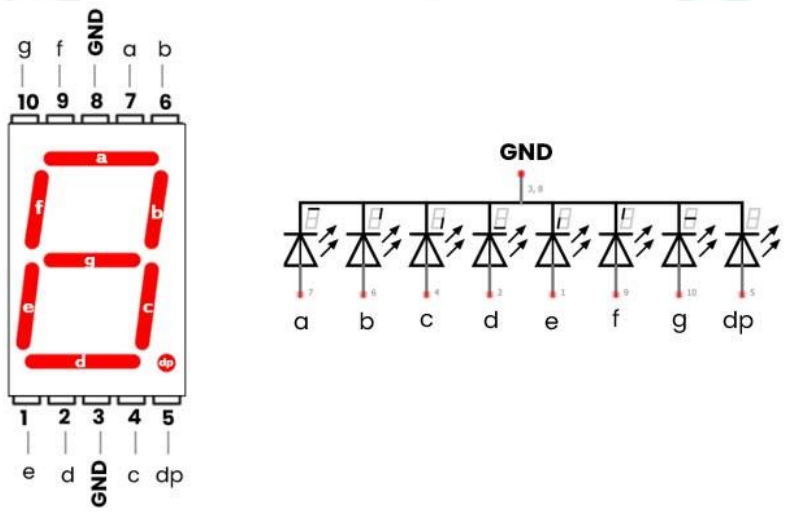
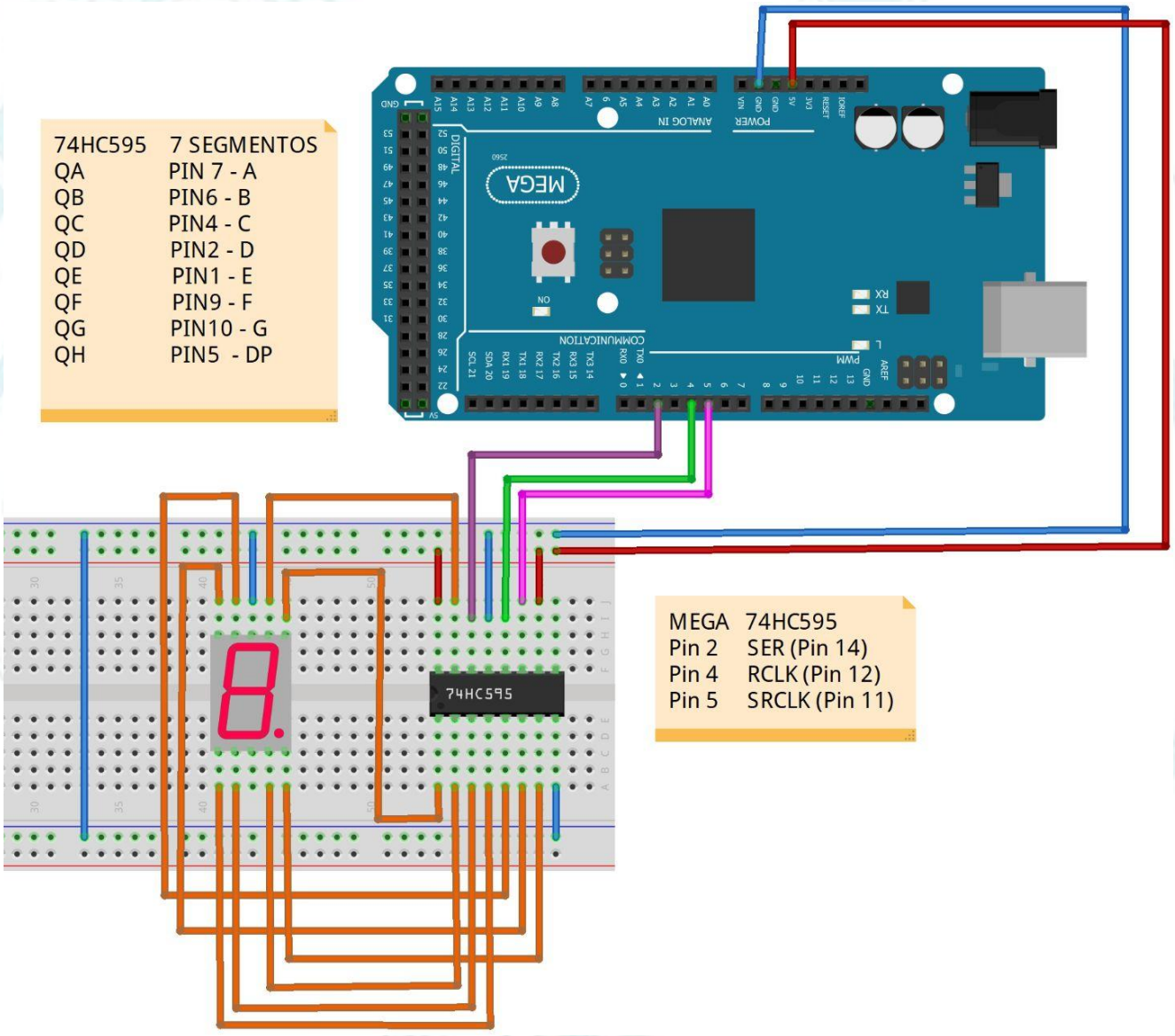
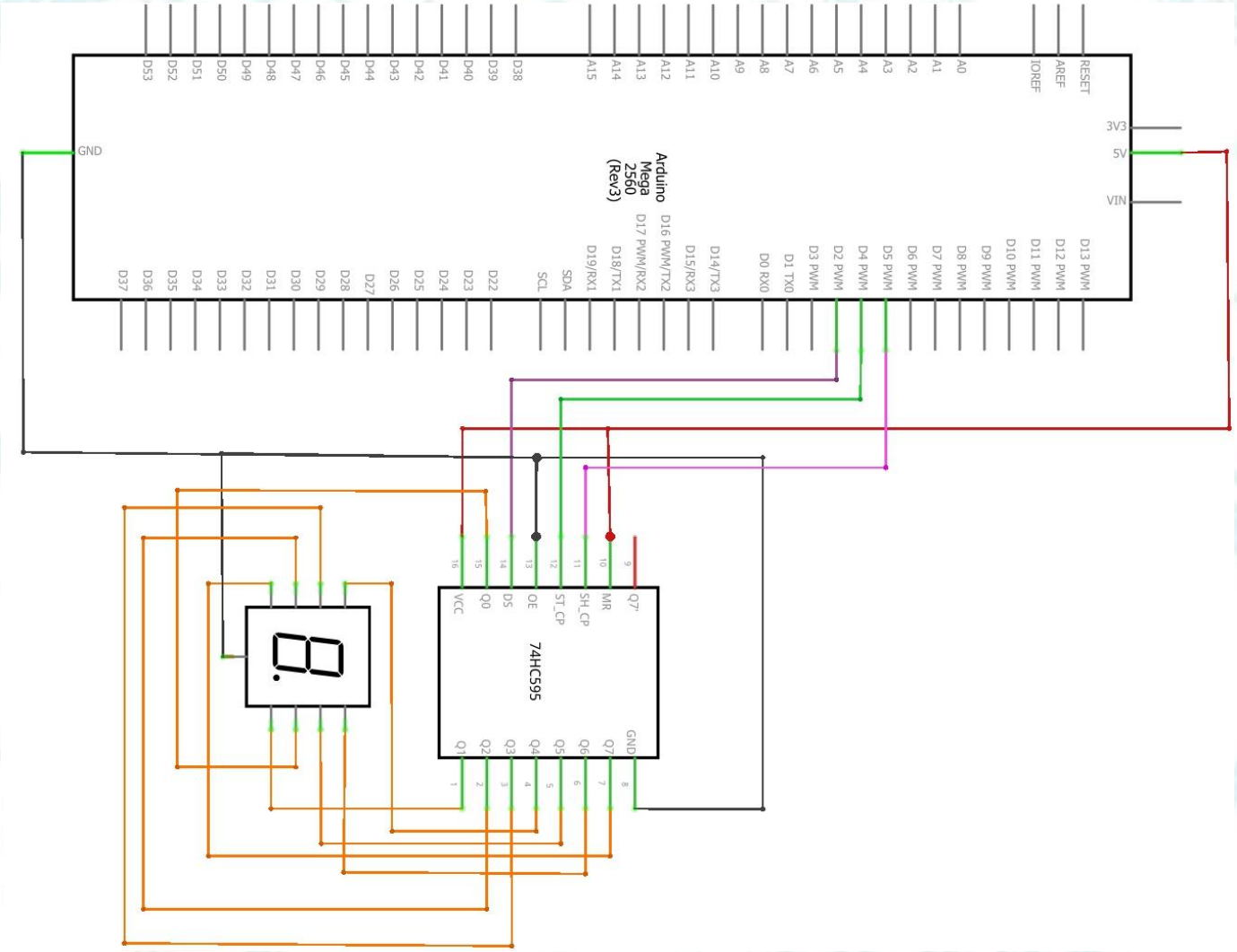


Diagrama de conexión



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

```
int latchPin =4 ;  // Pin conectado al Pin 12 del 74HC595 (Latch)

int clockPin = 5;  // Pin conectado al Pin 11 del 74HC595 (Clock)

int dataPin = 2;// Pin conectado al Pin 14 del 74HC595 (Data)

int i = 0;//Variable para realizar conteo

//Combinación de bit por led del display de 7 segmentos realizando una
secuencia desde 0 a la letra C: A,B,C,D,E,F,G,PUNTO DECIMAL

const byte numeros[14] = {

  B11111101,          // = 0.

  B01100000,          // = 1

  B11011010,         // = 2

  B11110010,         // = 3

  B01100110,         // = 4

  B10110110,         // = 5

  B10111110,         // = 6

  B11100000,         // = 7

  B11111110,         // = 8

  B11100110,         // = 9

  B11101110,         // = A

  B00111110,         // = B

  B10011101,         // = C

};

void setup() {

  Serial.begin(9600); //Declaración de variables, como de salida digital

  pinMode(latchPin, OUTPUT);

  pinMode(clockPin, OUTPUT);

  pinMode(dataPin, OUTPUT);}
```

```
void loop() {

 for (i = 0; i < 14; i++) {   //repetición de la secuencia de encendido y apagado de byte del 1 al 14 anteriormente declarada en la función números

digitalWrite(latchPin, LOW); //desactiva a Reloj de Registro de Desplazamiento función shifOut que nos ayudará a controlar al 74HC para desplazar el bit,indicando porque pin sale el bit (dataPin),cambio del pin una vez que dataPin tenga el valor adecuado(clockPin), dirección de desplazamiento (LSBFIRST o MSBFIRST) los datos que se desplazaran(leds)byte

shiftOut(dataPin, clockPin, LSBFIRST, numeros[i]);

   digitalWrite(latchPin,   HIGH);//activado el Reloj de Registro de Desplazamiento

    delay(1000); //espera de 1 segundo entre cada secuencia

  }

}
```

Lección 9 74HC595 y Display 4 Dígitos 7 segmentos

En esta lección aprenderás a utilizar el display 4 dígitos 7 segmentos de ánodo común con el registro de desplazamiento 74HC595.

Materiales necesarios

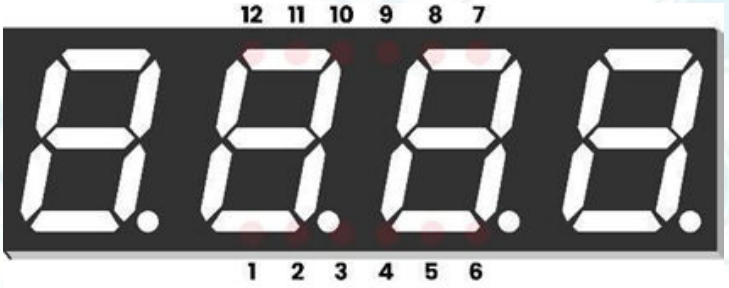
- Arduino Mega 2560
- Protoboard
- Resistencia de 220 Ohms
- IC 74HC595
- Display 4 Dígitos 7 Segmentos cátodo común
- Cables Dupont Macho-Macho

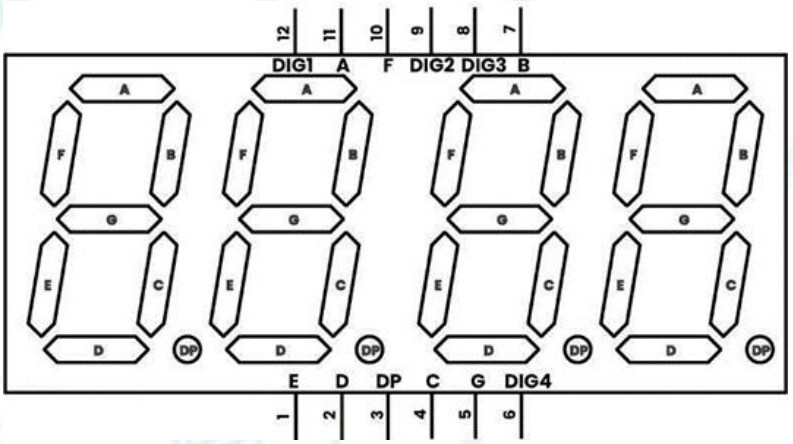
Conocimientos previos

El display 4 dígitos 7 Segmentos es un dispositivo opto-electrónico que permite visualizar números del 0 al 9. Se utiliza para representar visualmente números y algunos caracteres. El display está compuesto por 4 dígitos, cada dígito tiene 7 segmentos y un punto decimal que pueden encender o apagar individualmente.

En la siguiente imagen se muestra los pines del display 4 dígitos 7 Segmentos:

PINOUT


12 11 10 9 8 7
1 2 3 4 5 6


12 11 10 9 8 7
DIG1 A F DIG2 DIG3 B
E D DP C G DIG4
1 2 3 4 5 6

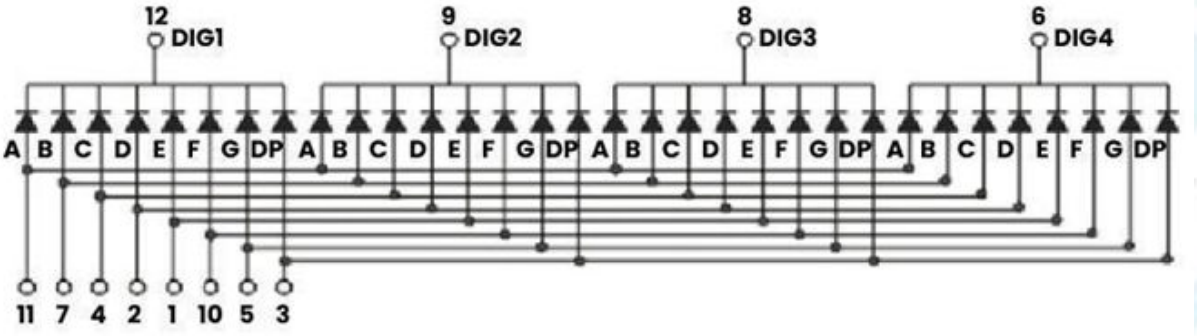
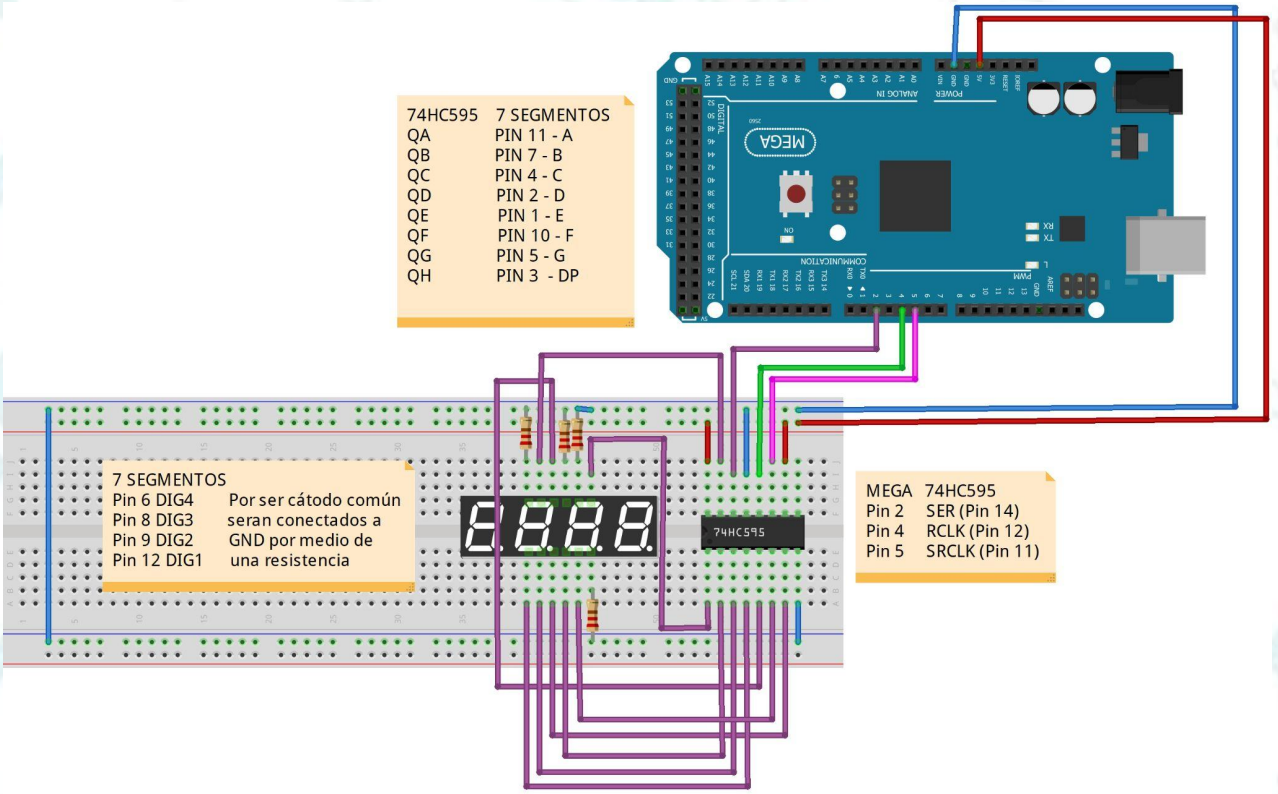
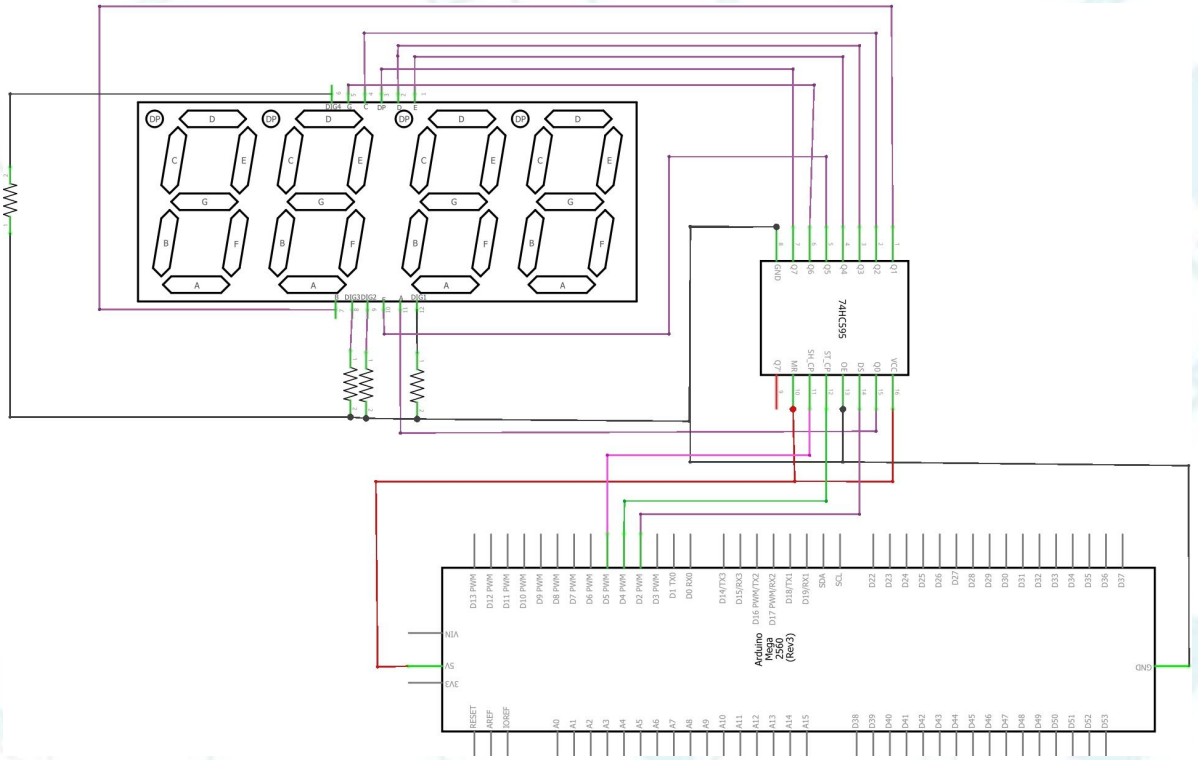

12 DIG1
9 DIG2
8 DIG3
6 DIG4
A B C D E F G DP A B C D E F G DP A B C D E F G DP A B C D E F G DP
11 7 4 2 1 10 5 3

Diagrama de conexión





Código de funcionamiento

```
int latch=9;                          //Pin 9 al 12 (STCP) del 74HC595
int clockPin=10;                    //Pin 10 al 11 (SHCP) del 74HC595
int data=8;                           //Pin 8 al 14 (DS) del 74HC595
int i = 0;
//Matriz de codificación del display de 7 segmentos - hexadecimal
const byte numeros[14] = {
  B11111101,          // = 0.
  B01100000,          // = 1
  B11011010,         // = 2
  B11110010,         // = 3
  B01100110,         // = 4
  B10110110,         // = 5
  B10111110,         // = 6
  B11100000,         // = 7
  B11111110,         // = 8
  B11100110,         // = 9
  B11101110,         // = A
  B00111110,         // = B
  B10011101,};         // = C
void setup() {                              //inicializa pines de salida
  pinMode(latch, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(data, OUTPUT);}
//Función Display en donde controlaremos el Latch del dato
void Display(unsigned char num){
  digitalWrite(latch,LOW);
```

```
  shiftOut(data, clockPin, MSBFIRST,numeros[num]);

  digitalWrite(latch, HIGH);

}

//impresión de los valores de la matriz dependiendo del orden que fueron agregados los datos

void loop() {

 for (i = 0; i < 56; i++) {   //repetición de la secuencia de encendido y apagado de byte del 1 al 14 anteriormente declarada en la función números

digitalWrite(latch, LOW); //desactiva a Reloj de Registro de Desplazamiento función shifOut que nos ayudará a controlar al 74HC para desplazar el bit,indicando por que pin sale el bit (dataPin),cambio del pin una vez que dataPin tenga el valor adecuado(clockPin), dirección de desplazamiento (LSBFIRST o MSBFIRST) los datos que se desplazaran(leds)byte



shiftOut(data, clockPin, LSBFIRST, numeros[i]);

         digitalWrite(latch,  HIGH);//activado el Reloj de Registro de Desplazamiento

    delay(1000);                //espera de 1 segundo entre cada secuencia

  }

}
```

Lección 10 Sensor de Inclinación

En esta lección, aprenderás a utilizar el sensor de inclinación SW-520D, un dispositivo sencillo pero muy útil que detecta cambios en la inclinación cuando se supera su umbral.

Materiales necesarios

- Tarjeta Mega 2560
- Protoboard
- Resistencias de 220 ohms
- Sensor de inclinación SW-520D
- Cables Dupont Macho-Macho
- Led rojo

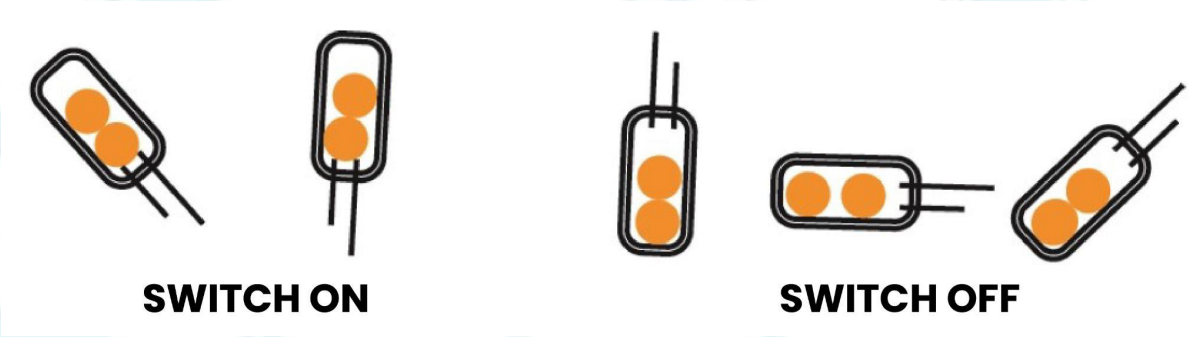
Conocimientos previos

El sensor de inclinación SW-520D es un dispositivo que se utiliza para detectar cambios en la inclinación o la posición de un objeto.

Antes de utilizar este sensor, es útil entender los siguientes conceptos básicos.

1. **Funcionamiento interno**: El sensor SW-520D contiene una pequeña esfera metálica en su interior que actúa como un interruptor mecánico. Dependiendo de su inclinación, la esfera se mueve y permite cerrar o abrir su circuito eléctrico.
2. **Señal de salida**: La señal de salida del sensor es digital, lo que significa que solo tiene dos estados: Alto (ON) o Bajo (OFF), dependiendo de su inclinación.

En la siguiente imagen se muestran los estados dependiendo de su inclinación:

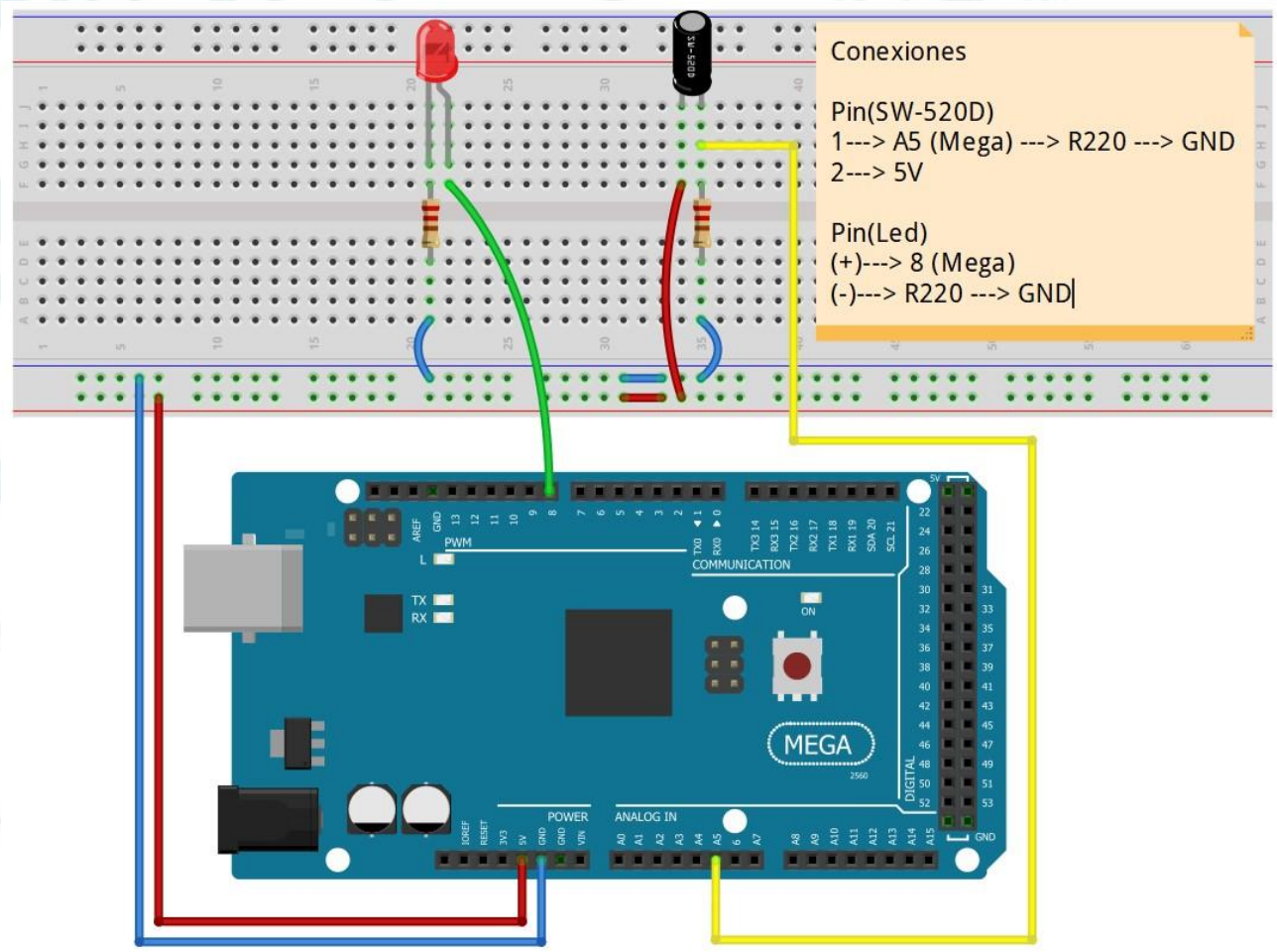


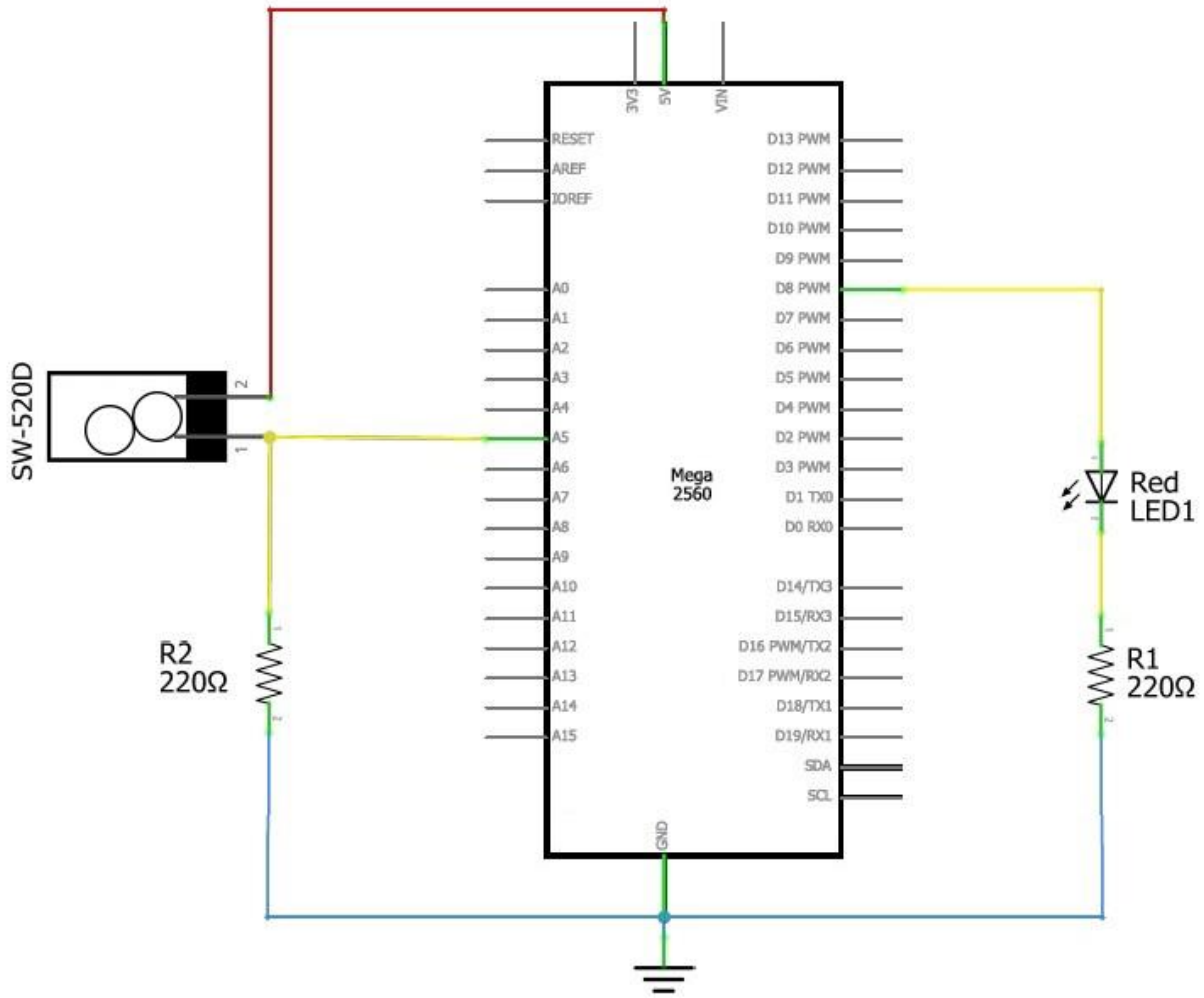
Este comportamiento lo hace ideal para detectar cambios simples, si un objeto está inclinado o no.

3. **Precisión y sensibilidad:** El SW-520D no detecta ángulos específicos, sino cambios de posición. Para mediciones más precisas de inclinación, se requieren otros sensores como acelerómetros.

Diagrama de conexión

A continuación, se detallan las conexiones entre el sensor de inclinación, la tarjeta Mega 2560 y el LED, el cual se encenderá o apagará en función de la respuesta del sensor de inclinación.




SW-520D
2
1
R2
220Ω
R1
220Ω
Red
LED1
Mega
2560
3V3
5V
VIN
GND
RESET
AREF
IOREF
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D13 PWM
D12 PWM
D11 PWM
D10 PWM
D9 PWM
D8 PWM
D7 PWM
D6 PWM
D5 PWM
D4 PWM
D2 PWM
D3 PWM
D1 TX0
D0 RX0
D14/TX3
D15/RX3
D16 PWM/TX2
D17 PWM/RX2
D18/TX1
D19/RX1
SDA
SCL

Código de funcionamiento

El siguiente código permite leer el estado del sensor de inclinación para encender o apagar un led dependiendo de la inclinación del sensor.

```
const int LedPin = 8;  //Variable para el Pin 8 que se conectará con el Led

void setup() {

  pinMode(LedPin, OUTPUT);

}

void loop()  //Función Loop

{

  int i;

  while (1)  //While para tener una instrucción bucle sin fin

  {

      i = analogRead(5);   //La variable i tendrá lectura de los valores provenientes del pin 5 analógico

    if (i > 512)          //rango ajustable de o a 1023, 512 tendremos una sensibilidad media en el sensor

    {

      digitalWrite(LedPin, LOW);  //El led apagará, en posición vertical del sensor

    } else {

        digitalWrite(LedPin, HIGH);   //El led se encenderá, en posición horizontal del sensor

    }

  }

  }
```

Lección 11 Sensor de Ultrasónico

En esta lección, aprenderás a utilizar el sensor ultrasónico HC-SR04 para medir distancias con precisión. Este sensor es ideal para proyectos que requieren detección de obstáculos o medir distancias.

Materiales necesarios

- Tarjeta Mega 2560
- Sensor Ultrasónico
- Cables Dupont Macho-Macho

Conocimientos previos

El sensor ultrasónico HC-SR04 es un dispositivo que mide distancias utilizando ondas de sonido de alta frecuencia (ultrasonido). Se usa en proyectos de robótica, automatización y detección de obstáculos.

Antes de utilizar este sensor, es útil entender los siguientes conceptos básicos.

1. **Funcionamiento**

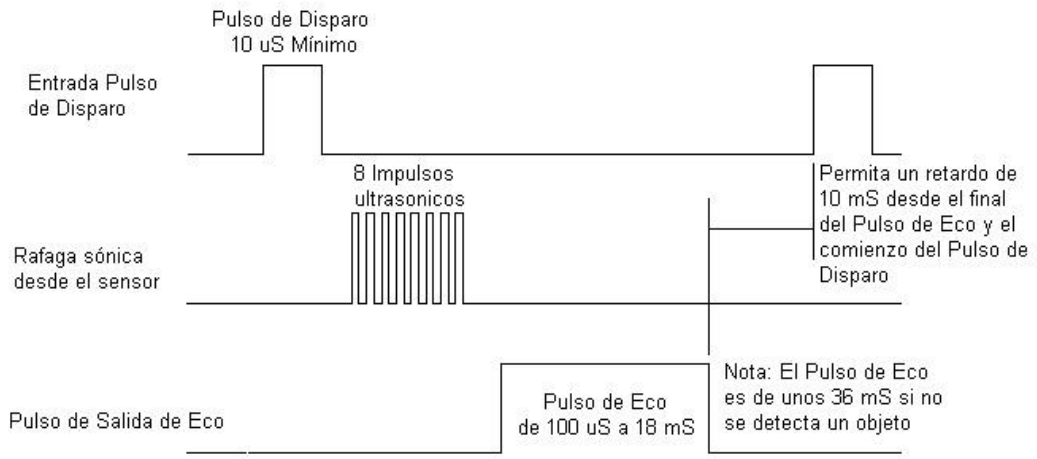
El **HC-SR04** tiene un **rango de medición de 2 cm a 400 cm** y funciona midiendo el tiempo que tarda una onda de ultrasonido en viajar hasta un objeto y regresar.

Este sensor cuenta con **dos componentes principales**:

- **Emisor (Trigger):** Envía un pulso ultrasónico.
- **Receptor (Echo):** Detecta el eco reflejado por el objeto y mide el tiempo que tardó en regresar.

El proceso de medición ocurre de la siguiente manera:

1. Se activa el **Trigger** enviando un pulso de al menos **10 µs** en nivel alto.
2. El sensor emite automáticamente **8 pulsos ultrasónicos** a **40 kHz**.
3. Si el sonido rebota en un objeto, el **Echo** recibe la señal reflejada y mantiene un pulso alto por un tiempo proporcional a la distancia.
4. Con el tiempo medido y la velocidad del sonido (**343 m/s**), se calcula la distancia al objeto. En la siguiente imagen se ilustra su funcionamiento.



2. Cálculo de distancia

Para determinar la distancia al objeto, se utiliza la siguiente fórmula:

$$\text{Distancia} = \frac{\text{Tiempo} \times \text{Velocidad del sonido}}{2}$$

Donde:

- **Tiempo** es el que tarda el sonido en ir y regresar (medido por el sensor).
- **Velocidad del sonido** es aproximadamente **343 m/s** o **0.0343 cm/µs**.
- Se divide entre **2** porque el sonido viaja hasta el objeto y regresa.

En unidades prácticas para el HC-SR04:

$$\text{Distancia (cm)} = \frac{\text{Tiempo (µs)} \times 0.0343}{2}$$

Ejemplo de cálculo:

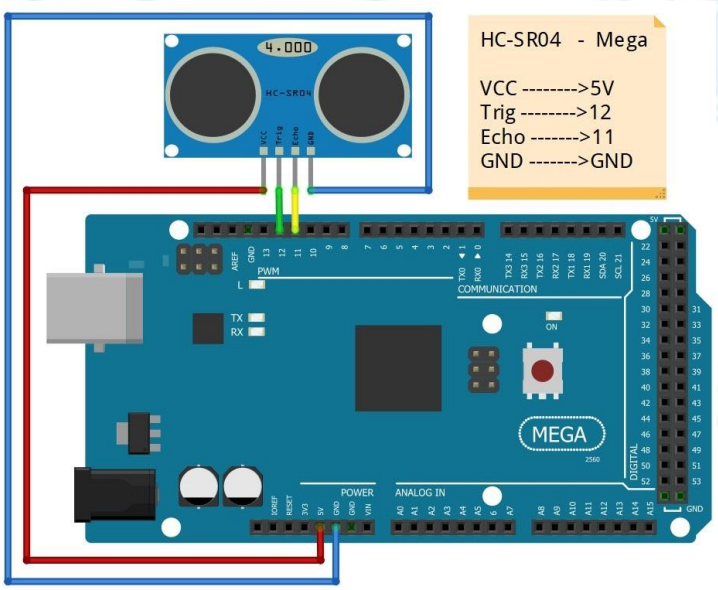
Si el sensor mide un tiempo de 1000 µs (1 ms), la distancia se calcula así:

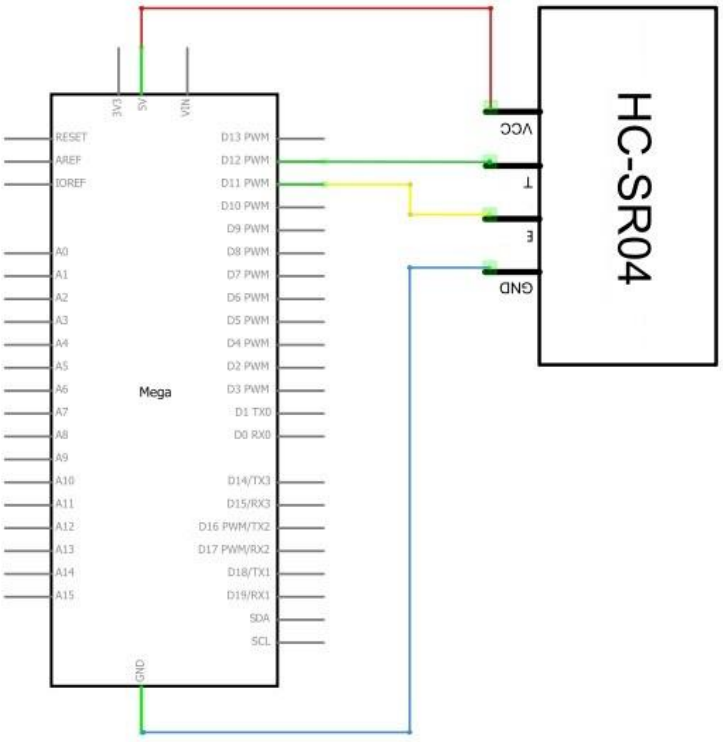
$$\text{Distancia} = \frac{1000 \times 0.0343}{2} = \frac{34.3}{2} = 17.15 \text{ cm}$$

Esto significa que el objeto está ubicado aproximadamente a 17.15 cm del sensor. Así que para utilizar este sensor con la tarjeta Mega 2560 a continuación se detallan sus conexiones.

Diagrama de conexión

A continuación, se muestra cómo conectar el sensor ultrasónico HC-SR04 a la tarjeta Mega 2560 para medir distancias. Además, utilizaremos el Monitor Serie para visualizar los datos en tiempo real.





Código de funcionamiento

El siguiente código mide la distancia utilizando el sensor ultrasónico HC-SR04 y la muestra en el Monitor Serie, sin necesidad de librerías adicionales.

```
// Definición de pines del sensor ultrasónico
const int triggerPin = 12;  // Pin de activación del sensor
const int echoPin = 11;     // Pin de recepción de la señal
// Función para medir la distancia en cm
float medirDistancia() {
    long duration;  // Variable para almacenar el tiempo del eco
    float distance; // Variable para almacenar la distancia calculada
    // Generar un pulso en el Trigger para iniciar la medición
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    // Leer el tiempo que tarda en regresar la señal de ultrasonido
    duration = pulseIn(echoPin, HIGH);
     // Calcular la distancia en centímetros (velocidad del sonido = 34300 cm/s)
    distance = (duration * 0.0343) / 2;
    return distance; // Retorna la distancia calculada
}
void setup() {
     Serial.begin(9600);             // Inicia la comunicación serie a 9600 baudios
    pinMode(triggerPin, OUTPUT); // Configura el pin Trigger como salida
```

```
    pinMode(echoPin, INPUT);     // Configura el pin Echo como entrada
}

void loop() {

    float distancia = medirDistancia(); // Llama a la función para obtener la
distancia

    // Imprimir la distancia en el Monitor Serie

    Serial.print("Distancia: ");

    Serial.print(distancia);

    Serial.println(" cm");

    delay(1000); // Pequeña pausa antes de la siguiente medición

}
```

Lección 12 Módulo Receptor IR

En esta lección, aprenderás a utilizar el sensor infrarrojo KY-022 junto con un control remoto para manipular un LED RGB, permitiéndote encender, apagar y ajustar sus colores de forma remota.

Materiales necesarios

- Tarjeta Mega 2560
- Sensor IR KY-022 con Control
- Protoboard
- Cables Dupont Macho-Macho

Conocimientos previos

El sensor IR KY-022 es un receptor infrarrojo que permite la detección y decodificación de señales enviadas por un control remoto. Se utiliza en múltiples aplicaciones, como el control de dispositivos electrónicos, automatización y robótica.

Antes de utilizar este sensor, es útil entender los siguientes conceptos básicos.

1. **Funcionamiento**

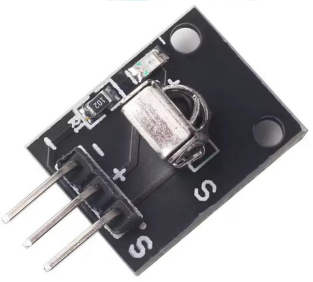
El sensor IR KY-022 detecta pulsos de luz infrarroja emitidos por un LED IR en un control remoto. Estos pulsos están modulados en una frecuencia de 38 kHz para evitar interferencias con otras fuentes de luz ambiental. El proceso de funcionamiento se divide en los siguientes pasos:

1. **El control remoto emite un haz de luz infrarroja** cuando se presiona un botón.
2. **El sensor IR KY-022 recibe esta señal** y la filtra, eliminando cualquier interferencia de luz visible.
3. **El sensor convierte la señal infrarroja en pulsos eléctricos**, los cuales representan un código único para cada botón.
4. **El microcontrolador o tarjeta de desarrollo interpreta el código** y ejecuta una acción programada.

2. Pines

El sensor IR KY-022 tiene tres pines que deben conectarse correctamente para su funcionamiento:

Pin	Descripción
(-) o GND	Tierra
(+) o VCC	Alimentación (+5V)
S o OUT	Salida de datos digital



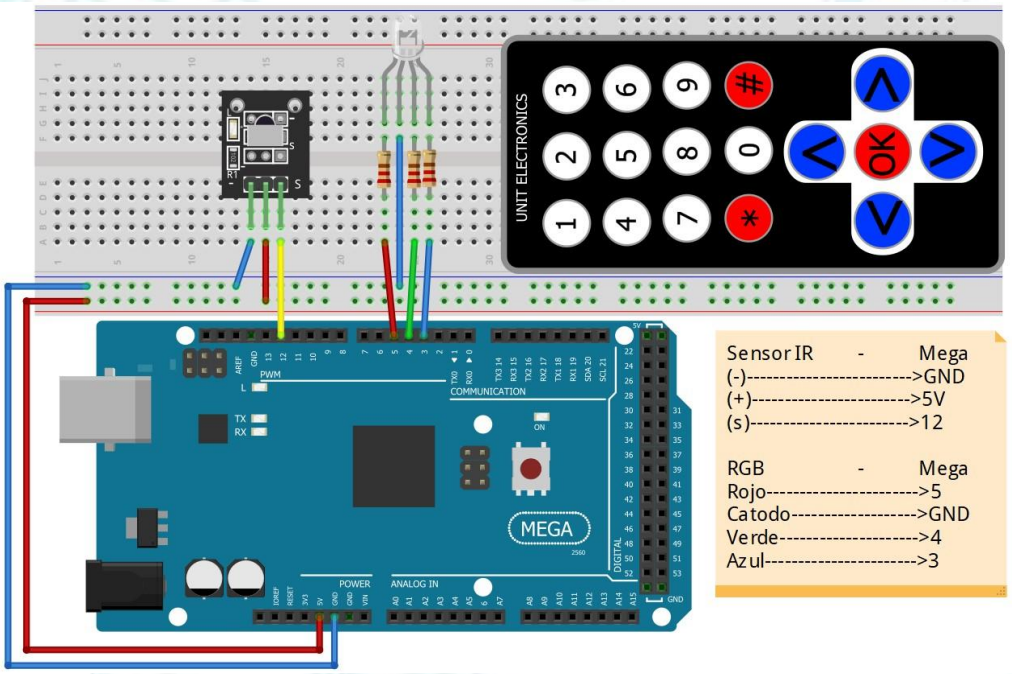
3. Control Remoto

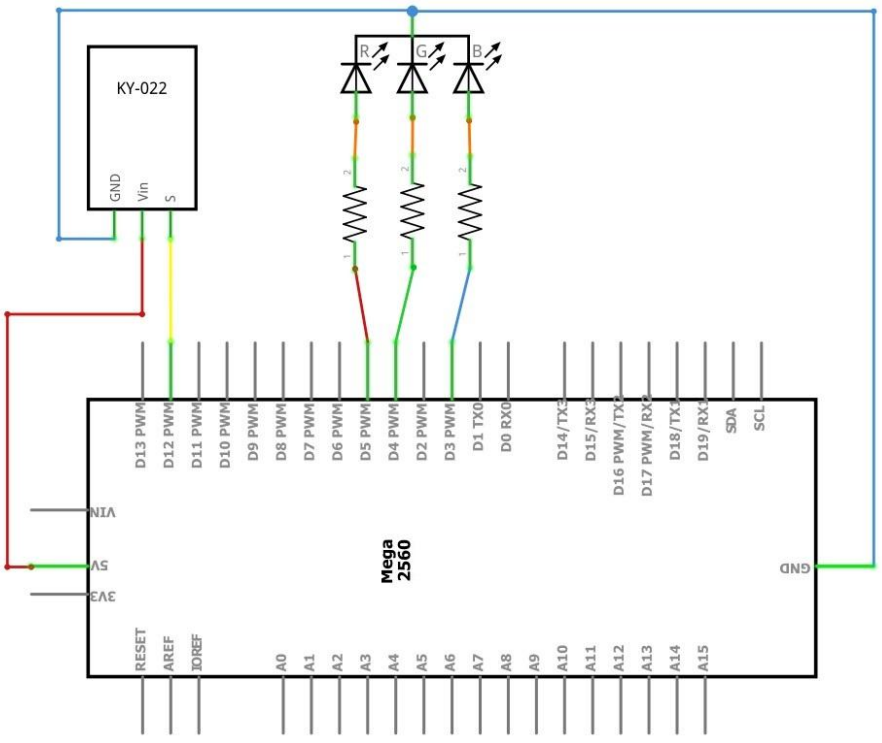
El sensor infrarrojo KY-022 viene acompañado de un control remoto, el cual no incluye una pila instalada. Para su funcionamiento, es necesario utilizar la batería tipo CR2032 que incluye el kit, la cual es compatible con el control.

Alternativamente, también puedes utilizar la pila **CR2025**, aunque esta no viene incluida en el kit. La diferencia principal entre ambas es que la CR2032 tiene mayor tamaño.

Diagrama de conexión

A continuación, se muestra cómo conectar el sensor IR KY-022 con la tarjeta Mega 2560 y para controlar un led RGB cátodo común.





Código de funcionamiento

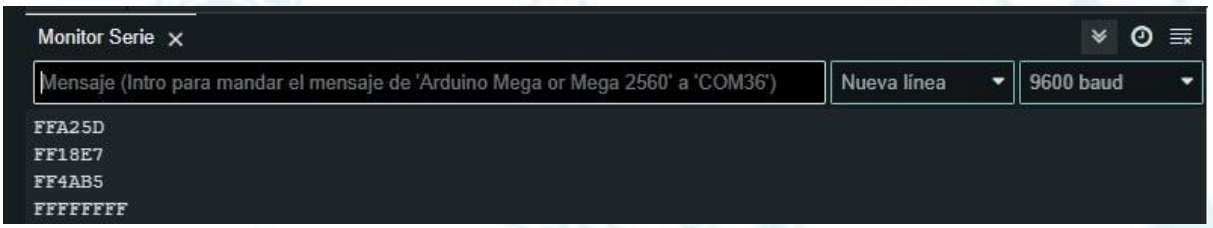
Al programar el sensor IR KY-022, lo primero que debes hacer es instalar la librería IRremote, la cual puedes encontrar en el Gestor de Bibliotecas del Arduino IDE. Es recomendable instalar la versión 2.2.3 para tener compatibilidad y funcionamiento óptimo.

En la siguiente imagen se muestra una referencia de la librería y la versión que debes instalar.

Una vez instalada la librería, utilizaremos el siguiente código para leer los valores de cada botón a través del Monitor Serie. Esto nos permitirá identificar los códigos únicos que emite el control remoto y asignarles una función en específico.

```
#include <IRremote.h> // Importa la biblioteca IRremote para manejar señales infrarrojas.

int SENSOR = 12;          // Define el pin donde está conectado el receptor infrarrojo.

IRrecv irrecv(SENSOR); // Crea un objeto IRrecv para recibir señales IR en el pin definido.

decode_results codigo; // Declara una variable para almacenar los datos decodificados de la señal IR.

void setup() {

   Serial.begin(9600);   // Inicia la comunicación serie a 9600 baudios para visualizar los datos en el monitor serie.

  irrecv.enableIRIn(); // Habilita la recepción de señales infrarrojas.

}

void loop() {

 if (irrecv.decode(&codigo)) {   // Verifica si se ha recibido y decodificado una señal IR.

Serial.println(codigo.value, HEX); // Muestra el valor codificado en formato hexadecimal en el monitor serie.

 irrecv.resume();  // Reinicia la recepción para capturar una nueva señal IR.

delay(100); // Pausa de 100 ms para evitar lecturas excesivamente rápidas.

 }
```

Por ejemplo, en la siguiente imagen, el Monitor Serie muestra los códigos que corresponden a cada botón que oprimas.



En este caso, utilizaremos los siguientes botones del control remoto para manipular el LED RGB:

Botón	**Código**	**Función**
Botón 1	FFA25D	Encender/Apagar color **Rojo**
Botón 2	FF629D	Encender/Apagar color **Verde**
Botón 3	FFE21D	Encender/Apagar color **Azul**
Botón Arriba	FF18E7	**Aumentar** el brillo del LED
Botón Abajo	FF4AB5	**Disminuir** el brillo del LED
Botón OK	FF38C7	**Apagar** el LED RGB

El siguiente código permite controlar un LED RGB de cátodo común con el sensor infrarrojo KY-022 y el control remoto IR.

El código permite encender y apagar cada color (rojo, verde y azul) y ajustar el brillo con los botones de arriba y abajo y apagar el led con el botón OK.

```
#include <IRremote.h>

// Definición de pines
#define RECV_PIN 12  // Pin del receptor IR KY-022
#define RED_PIN 3    // Pin del LED Rojo
#define GREEN_PIN 4  // Pin del LED Verde
#define BLUE_PIN 5   // Pin del LED Azul

// Códigos de los botones del control remoto
#define BTN_RED 0xFFA25D       // Botón para encender/apagar el LED rojo
#define BTN_GREEN 0xFF629D     // Botón para encender/apagar el LED verde
#define BTN_BLUE 0xFFE21D      // Botón para encender/apagar el LED azul
#define BTN_BRIGHT_UP 0xFF18E7 // Botón para aumentar el brillo
#define BTN_BRIGHT_DOWN 0xFF4AB5 // Botón para disminuir el brillo
#define BTN_OK 0xFF38C7        // Botón OK para reiniciar el color y brillo

// Instancia del receptor IR
IRrecv irrecv(RECV_PIN);
decode_results results;

// Variables de estado de los LEDs
bool redState = false;
bool greenState = false;
bool blueState = false;
int brightness = 255;  // Brillo inicial al máximo

void setup() {
    Serial.begin(9600);
    irrecv.enableIRIn(); // Inicia el receptor IR
```

```
    pinMode(RED_PIN, OUTPUT);
    pinMode(GREEN_PIN, OUTPUT);
    pinMode(BLUE_PIN, OUTPUT);
    resetColor(); // Apaga el LED al inicio
}
void loop() {
    // Verifica si se ha recibido una señal IR
    if (irrecv.decode(&results)) {
        Serial.println(results.value, HEX); // Imprime el código recibido
        handleIRCommand(results.value); // Maneja la señal recibida
        irrecv.resume(); // Recibir el siguiente código
    }
}
// Maneja los comandos IR recibidos
void handleIRCommand(unsigned long command) {
    switch (command) {
        case BTN_RED:
            redState = !redState; // Cambia el estado del LED rojo
            greenState = false;
            blueState = false;
            break;
        case BTN_GREEN:
            greenState = !greenState; // Cambia el estado del LED verde
            redState = false;
            blueState = false;
            break;
        case BTN_BLUE:
            blueState = !blueState; // Cambia el estado del LED azul
            redState = false;
            greenState = false;
            break;
        case BTN_BRIGHT_UP:
```

```
   brightness = min(255, brightness + 25);// Aumenta el brillo sin exceder 255
            break;
        case BTN_BRIGHT_DOWN:
   brightness = max(0, brightness - 25); // Disminuye el brillo sin bajar de 0
            break;
        case BTN_OK:
            resetColor(); // Reinicia el color y brillo
            return; // Evita ejecutar updateLED después de reiniciar
    }
    updateLED(); // Actualiza el color según los estados actuales
}
// Actualiza el color del LED según los estados actuales
void updateLED() {
    if (redState) {
        setColor(brightness, 0, 0); // Enciende rojo con el brillo actual
    } else if (greenState) {
        setColor(0, brightness, 0); // Enciende verde con el brillo actual
    } else if (blueState) {
        setColor(0, 0, brightness); // Enciende azul con el brillo actual
    } else {
        setColor(0, 0, 0); // Apaga el LED si no hay color activo
    }
}
// Reinicia los valores a su estado inicial (apagado y brillo al máximo)
void resetColor() {
    redState = false;
    greenState = false;
    blueState = false;
    brightness = 255; // Restablece brillo al máximo
    setColor(0, 0, 0); // Apaga LED
}
// Ajusta el color del LED RGB mediante PWM
```

```
void setColor(int red, int green, int blue) {
    analogWrite(RED_PIN, red);
    analogWrite(GREEN_PIN, green);
    analogWrite(BLUE_PIN, blue);
}
```

Lección 13 Sensor Joystick

En esta lección, aprenderás a utilizar el Sensor Joystick KY-023 con la tarjeta Mega 2560. Este sensor permite detectar el movimiento en dos ejes (X e Y) y cuenta con un botón integrado. Lo usaremos para controlar cuatro LEDs, encendiendo uno diferente según la dirección del joystick y utilizando el botón para encender o apagar todos a la vez.

Materiales necesarios

- Tarjeta Mega 2560
- Protoboard
- Resistencias de 220 ohms
- Sensor Joystick KY-023
- Cables para protoboard MH y MM
- Leds rojo, verde, amarillo y azul

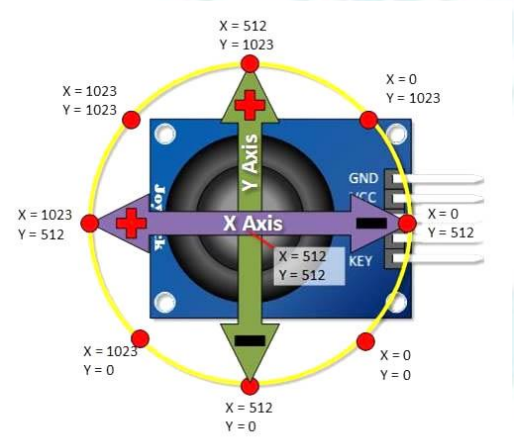
Conocimientos previos

El sensor joystick KY-023 es un módulo analógico que permite medir la inclinación en dos ejes (X e Y) y cuenta con un botón integrado al presionarlo. Funciona de manera similar a los joysticks de los controles de videojuegos.

Antes de utilizar este sensor, es útil entender los siguientes conceptos básicos.

1. Funcionamiento

El sensor **JoyStick KY-023** cuenta con dos potenciómetros internos que detectan los movimientos en los ejes **X** y **Y**, generando señales analógicas que pueden ser leídas por Arduino. En la siguiente imagen se muestran los valores analógicos correspondientes a cada posición del joystick.



A continuación se muestran los valores analógicos de los ejes **X** y **Y**, así como el estado del botón en cada posición del **JoyStick KY-023**.

Movimiento	**Eje X**	**Eje Y**	**Botón**
Centro (reposo)	~512	~512	1 (sin presionar)
Derecha	~1023	~512	1
Izquierda	~0	~512	1
Arriba	~512	~1023	1
Abajo	~512	~0	1
Botón presionado	-	-	0 (presionado)

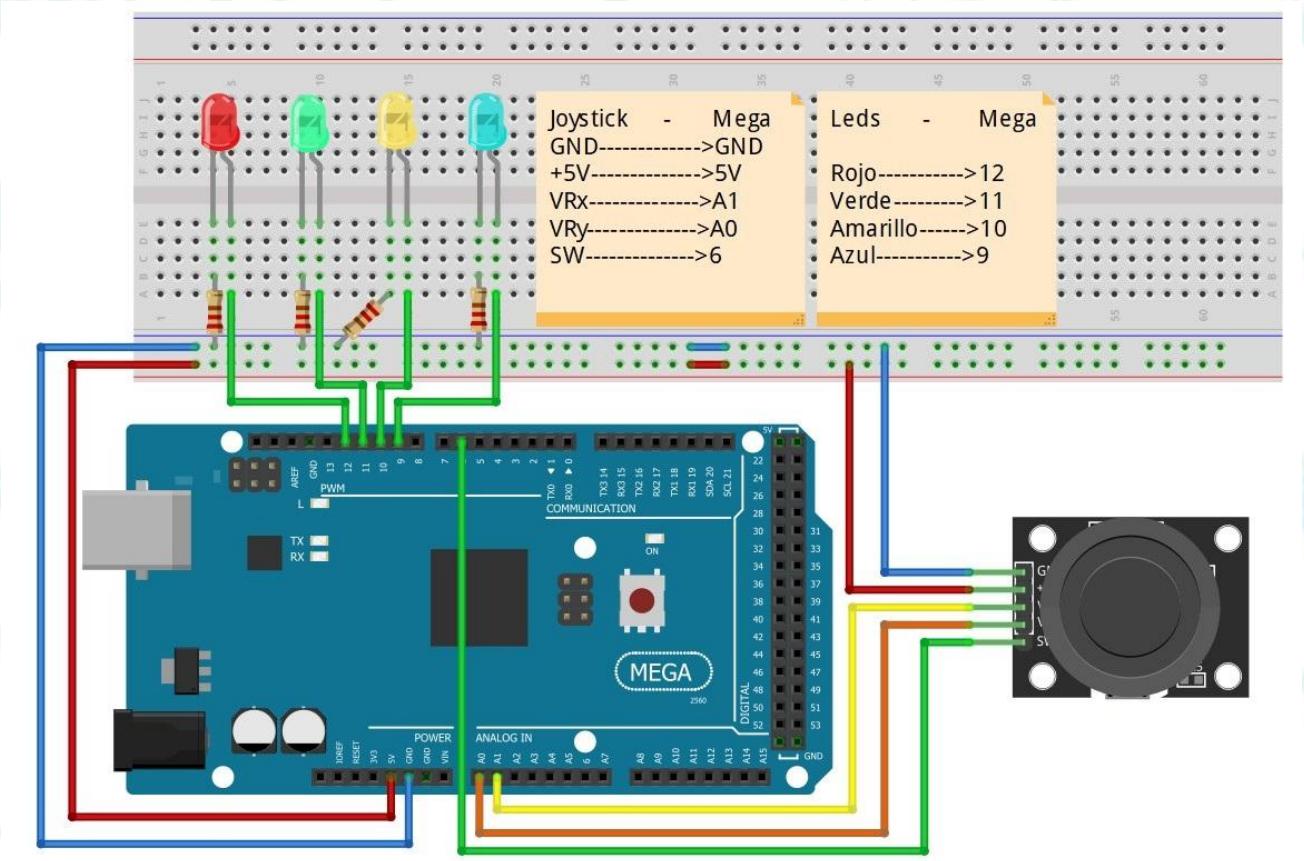
2. Pines del sensor JoyStick

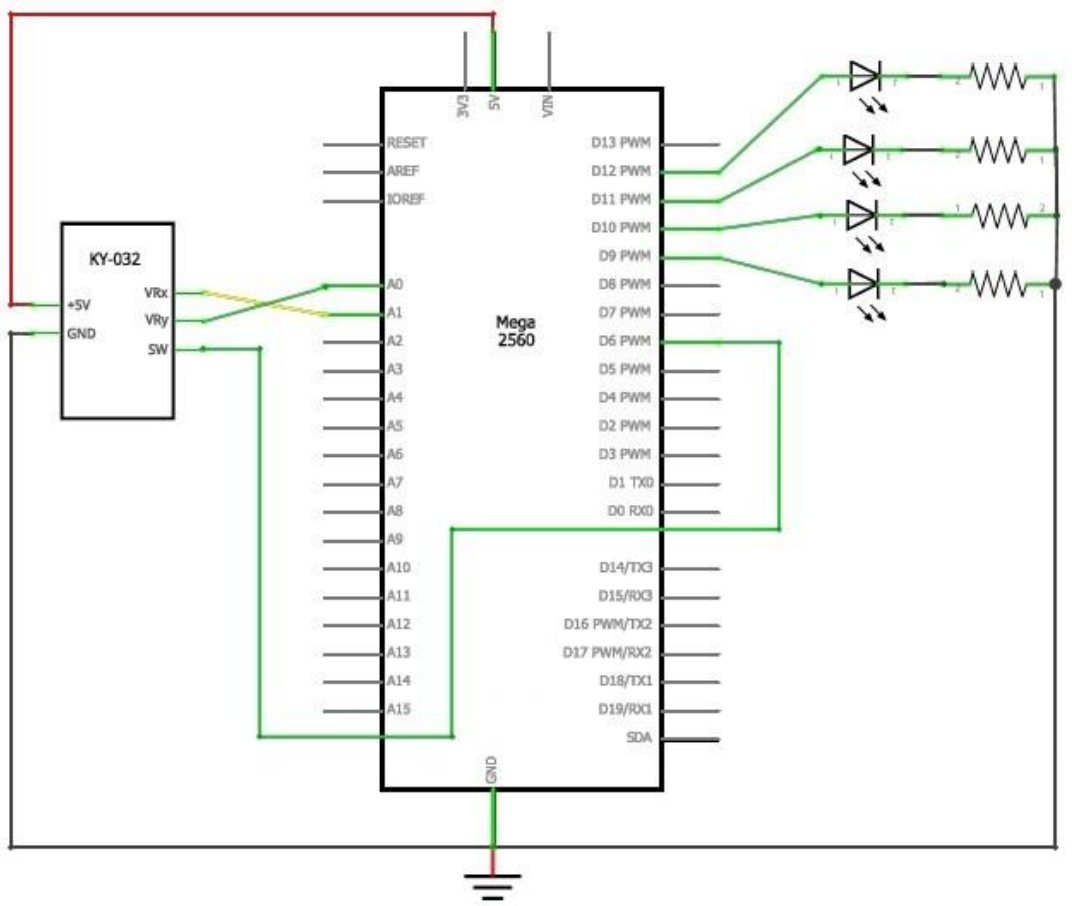
Este sensor cuenta con 5 pines:

- **GND** → Tierra
- **VCC** → 5V
- **VRX** → Salida analógica del eje X
- **VRY** → Salida analógica del eje Y
- **SW** → Botón digital

Diagrama de conexión

A continuación se muestra cómo conectar el sensor JoyStick KY-023, con la tarjeta Mega para controlar cuatro LEDs, encendiendo según la dirección del joystick o encender y apagar todos a la vez.




KY-032
+5V
GND
VRx
VRy
SW
Mega
2560
3V3
5V
VIN
RESET
AREF
IOREF
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D13 PWM
D12 PWM
D11 PWM
D10 PWM
D9 PWM
D8 PWM
D7 PWM
D6 PWM
D5 PWM
D4 PWM
D2 PWM
D3 PWM
D1 TX0
D0 RX0
D14/TX3
D15/RX3
D16 PWM/TX2
D17 PWM/RX2
D18/TX1
D19/RX1
SDA
GND

Código de funcionamiento

El siguiente código lee los valores del joystick (movimiento en los ejes X y Y) y enciende un LED correspondiente a la dirección de movimiento. Además, mediante el botón del joystick, es posible alternar el estado de todos los LEDs (encenderlos o apagarlos). Todo esto se monitorea a través del monitor serie, que proporciona información sobre el estado de los valores leídos por el joystick.

```
// Definición de pines del joystick
const int pinX = A1;  // Eje X conectado a A0
const int pinY = A0;  // Eje Y conectado a A1
const int pinSW = 6;  // Botón del joystick
// Definición de pines de los LEDs
const int ledRojo = 12;
const int ledVerde = 11;
const int ledAmarillo = 10;
const int ledAzul = 9;
bool estadoLeds = false;  // Estado de los LEDs (encendidos/apagados)
bool botonPresionado = false; // Variable para evitar rebotes del botón
void setup() {
    pinMode(pinSW, INPUT_PULLUP); // Botón con resistencia interna
    pinMode(ledRojo, OUTPUT);
    pinMode(ledVerde, OUTPUT);
    pinMode(ledAmarillo, OUTPUT);
    pinMode(ledAzul, OUTPUT);
    Serial.begin(9600); // Monitor Serie para depuración
}
void loop() {
    int valorX = analogRead(pinX);  // Leer eje X
```

```
    int valorY = analogRead(pinY);  // Leer eje Y
    int boton = digitalRead(pinSW); // Leer botón
    // Si el botón se presiona, alterna el estado de los LEDs
    if (boton == LOW && !botonPresionado) {
        estadoLeds = !estadoLeds; // Cambia el estado
        botonPresionado = true; // Evita rebotes
        encenderTodos(estadoLeds);
    } else if (boton == HIGH) {
        botonPresionado = false; // Permite detectar una nueva pulsación
    }
    // Solo controlar LEDs individualmente si están apagados por el botón
    if (!estadoLeds) {
        controlarLedsPorMovimiento(valorX, valorY);
    }
    // Monitoreo en la terminal serie
    Serial.print("X: ");
    Serial.print(valorX);
    Serial.print(" | Y: ");
    Serial.print(valorY);
    Serial.print(" | Botón: ");
    Serial.println(boton);
    delay(100);
}
// Función para controlar los LEDs según el movimiento del joystick
void controlarLedsPorMovimiento(int valorX, int valorY) {
        apagarLeds();  // Apagar todos los LEDs antes de encender el
correspondiente
    if (valorX > 800) {
        digitalWrite(ledAzul, HIGH);  // Derecha
    } else if (valorX < 200) {
        digitalWrite(ledAmarillo, HIGH);  // Izquierda
    } else if (valorY > 800) {
```

```
        digitalWrite(ledVerde, HIGH);  // Arriba
    } else if (valorY < 200) {
        digitalWrite(ledRojo, HIGH);  // Abajo
    }
}
// Función para apagar todos los LEDs
void apagarLeds() {
    digitalWrite(ledRojo, LOW);
    digitalWrite(ledVerde, LOW);
    digitalWrite(ledAmarillo, LOW);
    digitalWrite(ledAzul, LOW);
}
// Función para encender o apagar todos los LEDs según el estado
void encenderTodos(bool estado) {
    digitalWrite(ledRojo, estado);
    digitalWrite(ledVerde, estado);
    digitalWrite(ledAmarillo, estado);
    digitalWrite(ledAzul, estado);
}
```

Lección 14 Pantalla LCD

En esta lección, aprenderás a utilizar el display LCD 16x2 con interfaz I2C con la tarjeta Mega 2560 para mostrar mensajes.

Materiales necesarios

- Tarjeta Mega 2560
- Display LCD 16x2 con I2C
- Cables para protoboard HM

Conocimientos previos

El display LCD 16x2 con interfaz I2C es una pantalla de visualización que se utiliza para mostrar información ya sea texto, números, símbolos y valores sensores.

Antes de utilizar este display, es útil entender los siguientes conceptos básicos.

1. **Funcionamiento**

El LCD 16x2 es un dispositivo de visualización basado en cristal líquido que puede mostrar hasta 32 caracteres simultáneamente (16 en cada una de sus 2 líneas). Para simplificar su conexión, tiene integrado un adaptador I2C, que convierte la comunicación en serie.

El protocolo I2C (Inter-Integrated Circuit) permite que múltiples dispositivos compartan el mismo bus de comunicación mediante dos líneas:

- **SDA (Serial Data - Datos)**: Transporta la información entre el maestro (tarjeta Mega) y el esclavo (LCD).
- **SCL (Serial Clock - Reloj)**: Sincroniza la transmisión de datos con una señal de reloj.

El LCD 16x2 con I2C se comunica con los microcontroladores o tarjetas de desarrollo mediante una dirección específica en el bus I2C. Esta dirección suele ser **0x27** o **0x3F**, dependiendo de la LCD

2. Pines

El LCD 16x2 con I2C tiene cuatro pines que deben conectarse correctamente para su funcionamiento.

Pin	Nombre	Función
1	GND	Tierra
2	VCC	Alimentación de 5V para el módulo I2C y el LCD
3	SDA	Línea de datos serie del bus I2C
4	SCL	Línea de reloj del bus I2C

3. Ajustar el contraste

Los LCD con I2C, incorporan un potenciómetro para ajustar el contraste. Gira el potenciómetro en sentido horario o antihorario para mejorar la visualización de los caracteres en el display. En la siguiente imagen se muestra el potenciómetro que debes girar.

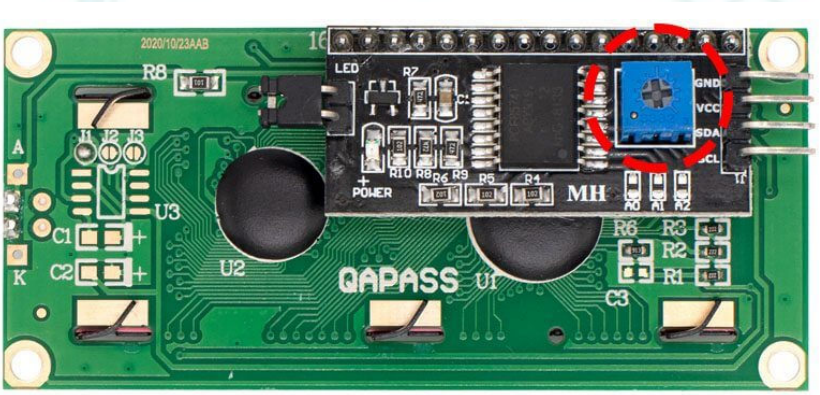
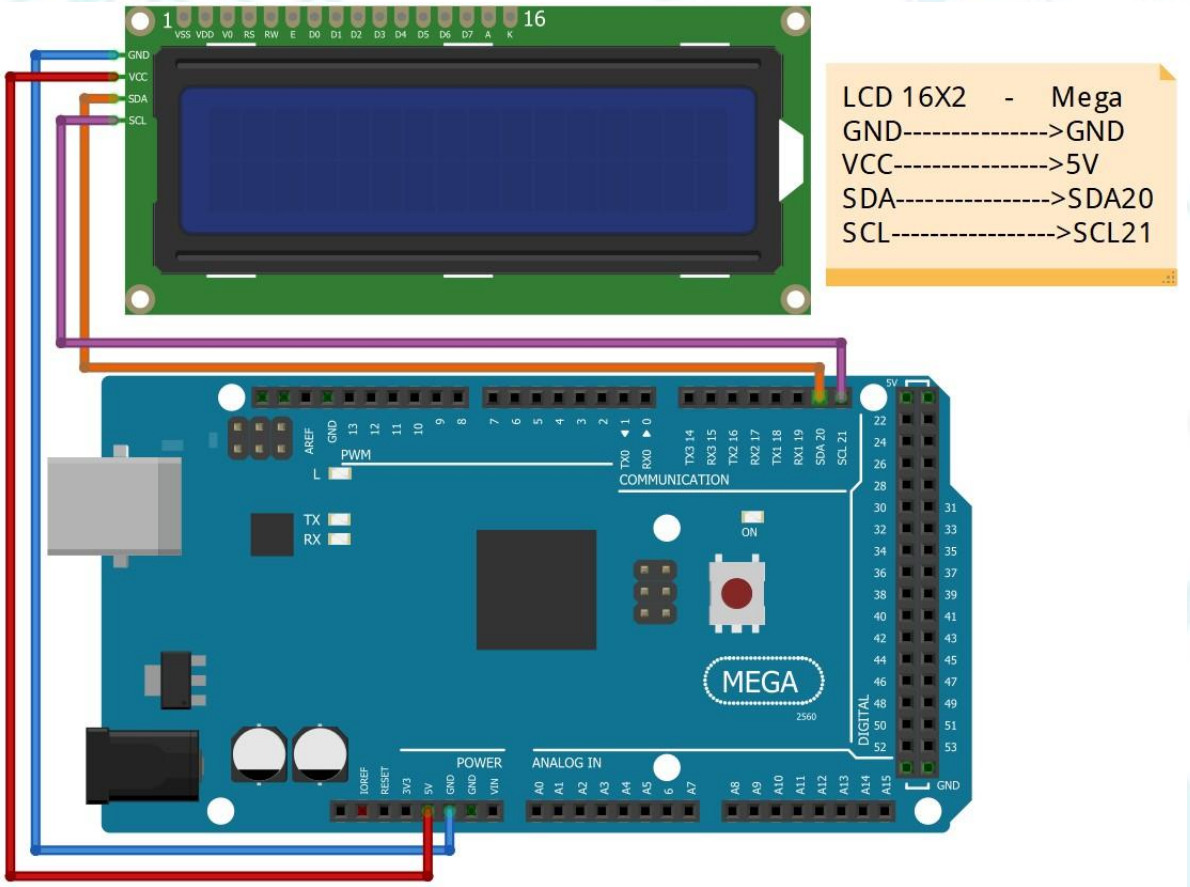
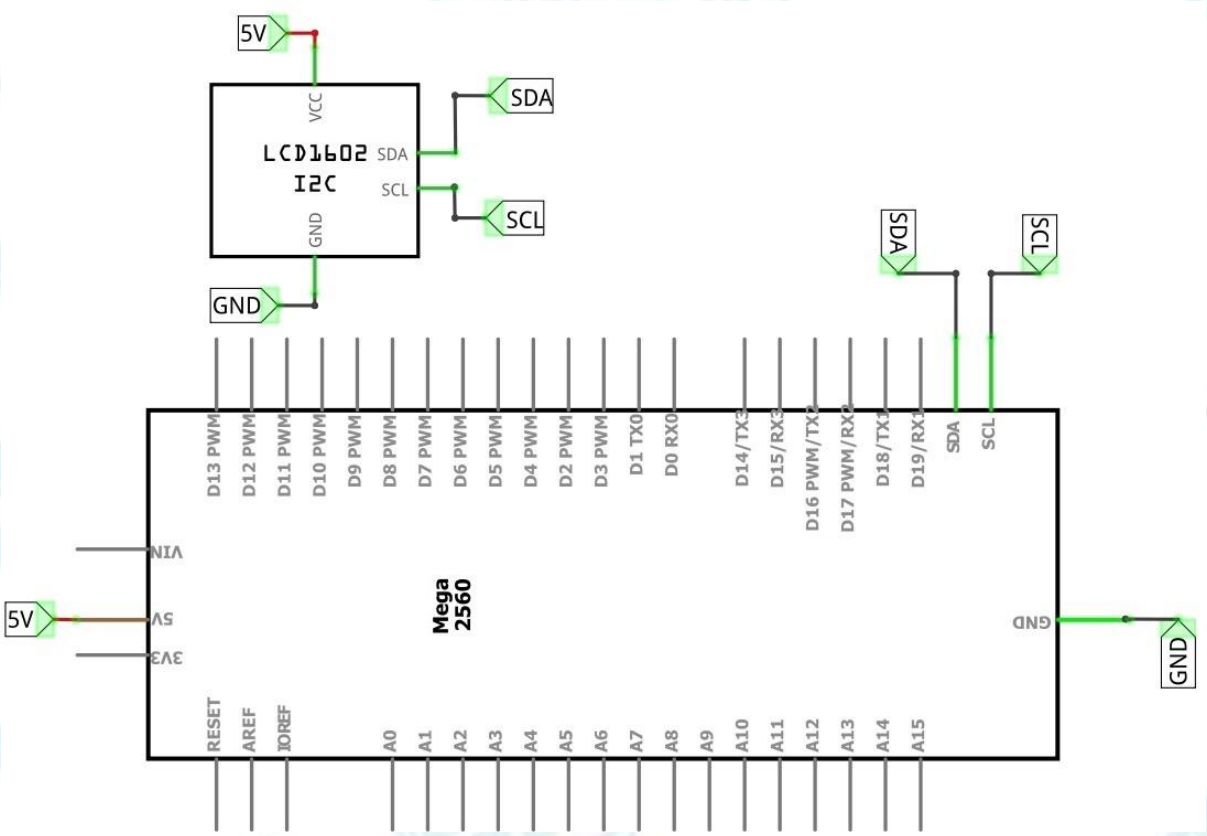


Diagrama de conexión

A continuación, se detallan las conexiones entre la tarjeta Mega 2560 y el LCD 16x2 con I2C.





Código de funcionamiento

Antes de programar el display LCD de 16x2 con I2C debes instalar la librería **LiquidCrystal_I2C** desde el **Gestor de Librerías** en el **Arduino IDE**.

En la siguiente imagen se muestra una referencia de la librería y la versión que debes instalar.

Este código muestra cómo inicializar y controlar la LCD 16x2 con interfaz I2C, utilizando la librería **LiquidCrystal_I2C**. En la función setup(), se configura el LCD y se activa la retroiluminación, mientras que en loop() se define la posición del texto y se muestran los mensajes en pantalla.

```
//Se incluye la librería LiquidCrystal contienen todo lo necesario para hacer funcionar la LCD

#include <LiquidCrystal_I2C.h>

//Se inicia la librería y se define la dirección y el tamaño del display

LiquidCrystal_I2C lcd(0x27, 16, 2);

//Primera función en ejecutarse, solo se ejecuta una vez cada que se energiza el Arduino UNO o se realiza un RESET.

void setup() {

//Se inicializa el LCD

lcd.init(); //Si marca error intenta con lcd.begin();

lcd.backlight();

}

//Lógica principal. Se ejecuta una y otra vez. Se define la función que ejecutará la LCD en este caso

//mostrará un mensaje.

void loop() {

//Se define la posición que se mostrará el mensaje con setCursor

//Se muestra un referencia de las columnas y filas

/*Columnas:    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

             ------------------------------

Filas:    0  # # # # # # # # # # # # # # # #

          1  # # # # # # # # # # # # # # # #

             ------------------------------ */

lcd.setCursor(6, 0);

//Mensaje a mostrar
```

```
lcd.print("UNIT");
lcd.setCursor(2, 1);
lcd.print("ELECTRONICS");
}
/* Como se verá el mensaje en el display
Columnas:     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
              ------------------------------
Filas:     0  # # # # # U N I T # # # # # # #
           1  # # E L E C T R O N I C S # # #
              ------------------------------ */
```

Lección 15 DHT11 Sensor de temperatura y humedad

En esta lección, aprenderás a utilizar el sensor DHT11 para medir temperatura y humedad, mostrando los valores en un display LCD 16x2 con interfaz I2C. Esta combinación es útil en proyectos de monitoreo ambiental, estaciones meteorológicas y control de temperatura en interiores.

Materiales necesarios

- Tarjeta Mega 2560
- Sensor DHT11
- Display LCD 16x2 I2C
- Protoboard
- Cables para protoboard MH -MM

Conocimientos previos

El DHT11 es un sensor digital que mide temperatura y humedad. Es utilizado en proyectos de domótica, estaciones meteorológicas y sistemas de control ambiental.

Antes de utilizar este sensor, es útil entender los siguientes conceptos básicos.

1. **Características del DHT11**

- **Temperatura**:
  - Rango de medición: **0°C a 50°C**
  - Precisión: **±2°C**
- **Humedad:**
  - Rango de medición: **20% a 90%**
  - Precisión: **±5%**

2. **Funcionamiento**

El sensor combina dos tecnologías de medición:

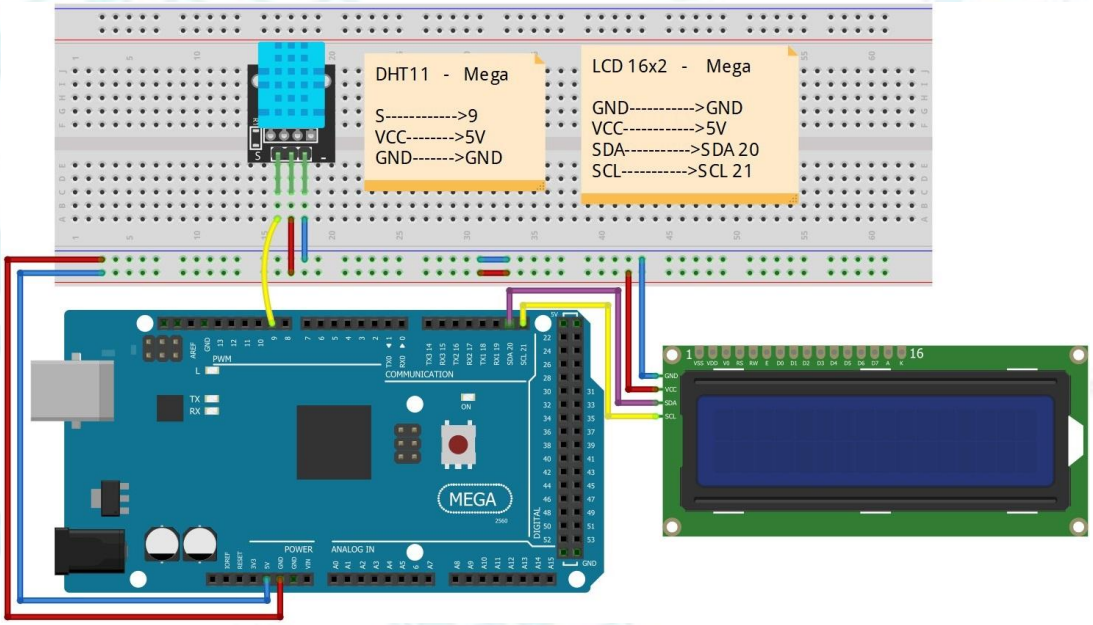
1. Un termistor para detectar la temperatura ambiente.
2. Un sensor capacitivo para medir la humedad relativa del aire.

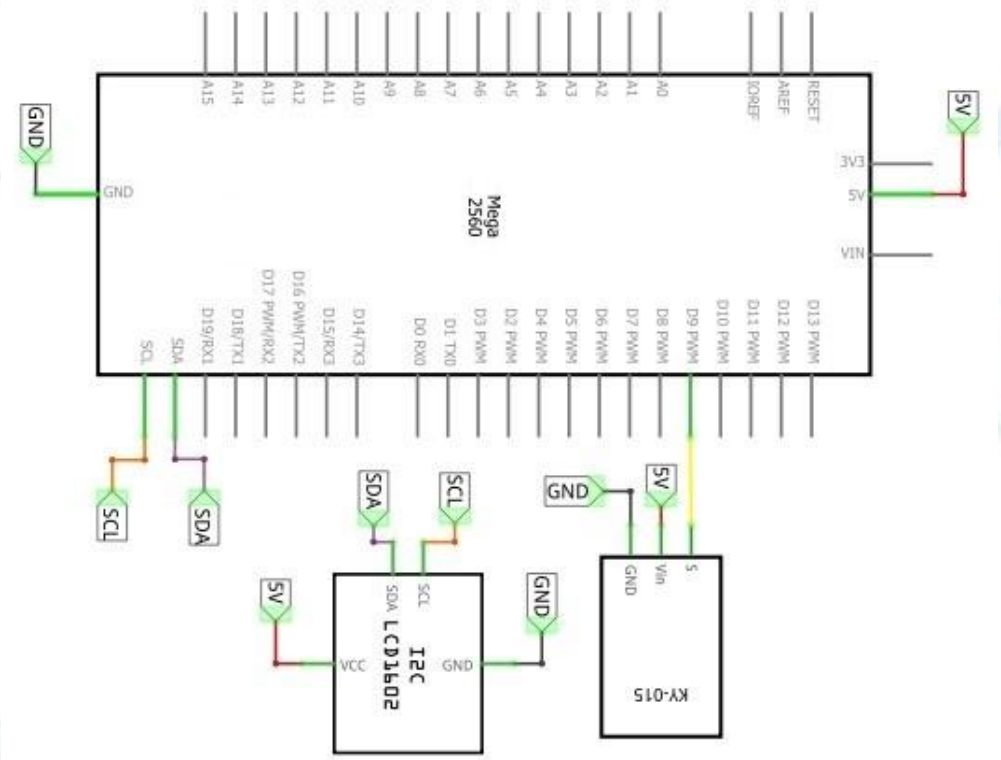
3. Principio de Operación

- La comunicación es digital, eliminando la necesidad de conversión analógica.
- Para obtener una medición, el microcontrolador envía una señal de inicio al sensor.
- El DHT11 responde enviando un conjunto de bits digitales, donde se codifican los valores de temperatura y humedad.
- Solo requiere un único pin de datos, lo que simplifica su conexión con placas Arduino, ESP32, Raspberry Pi y otras tarjetas.

Diagrama de conexión

A continuación se muestra cómo conectar el sensor DHT11 y la pantalla LCD 16x2 a la tarjeta Mega.





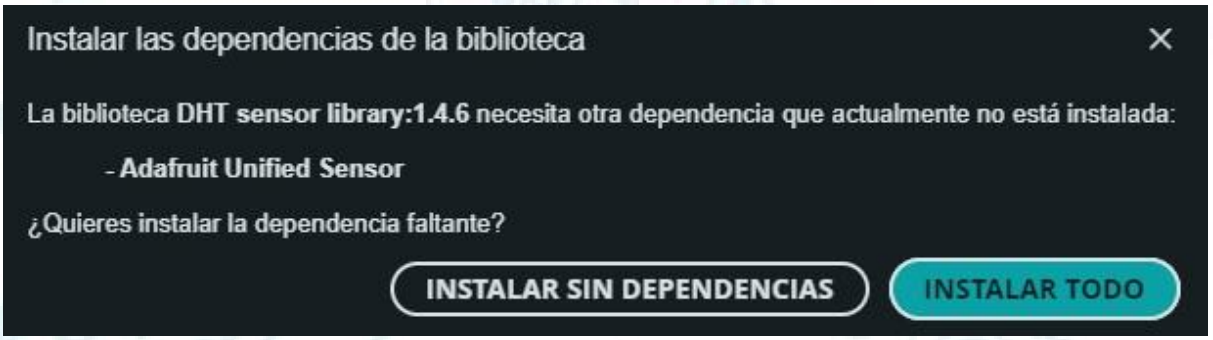
Código de funcionamiento

El siguiente código permite leer el estado del sensor DHT11 y visualizar la temperatura y humedad con el display LCD 16x2.

Antes de cargar el código, asegúrate de instalar las siguientes librerías en el Arduino IDE para su correcto funcionamiento.

LiquidCrystal_I2C.h	**DHT.h**
	

Para utilizar la librería DHT.h correctamente, es fundamental instalar todos sus complementos y dependencias. Se recomienda instalar la librería desde el **Gestor de Librerías** de Arduino IDE e instalar todas sus dependencias para contar con la versión más reciente, compatible y evitar errores en la lectura de datos.




```
#include <Wire.h>       // Librería para comunicación I2C

#include <LiquidCrystal_I2C.h>  // Librería para el LCD con I2C

#include <DHT.h>        // Librería para el sensor DHT

// Definimos el pin donde está conectado el sensor DHT11

#define DHTPIN 9

// Definimos el tipo de sensor DHT (puede ser DHT11, DHT22, etc.)

#define DHTTYPE DHT11

// Creamos un objeto DHT con el pin y tipo de sensor especificados

DHT dht(DHTPIN, DHTTYPE);

// Creamos un objeto para el LCD I2C con la dirección 0x27 y dimensiones 16x2
(16 columnas, 2 filas)

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {

    Serial.begin(9600);  // Iniciamos la comunicación serie a 9600 baudios

    dht.begin();  // Inicializamos el sensor DHT11

    lcd.init();  // Inicializa el LCD con la configuración establecida

    lcd.backlight();  // Enciende la luz de fondo del LCD

    // Mostramos un mensaje de inicio en el LCD

     lcd.setCursor(0, 0); // Posicionamos el cursor en la primera columna y
primera fila

    lcd.print("Iniciando...");

    delay(2000);  // Esperamos 2 segundos antes de comenzar las lecturas

}

void loop() {

    // Leemos la temperatura y la humedad desde el sensor

      float temperatura = dht.readTemperature();   // Obtener temperatura en
grados Celsius
```

```
    float humedad = dht.readHumidity();  // Obtener humedad relativa en %

       // Verificamos si la lectura es válida (el sensor puede fallar ocasionalmente)

    if (isnan(temperatura) || isnan(humedad)) {

          Serial.println("Error al leer DHT11");  // Mensaje de error en el Monitor Serie

          return;  // Salimos de la función loop() y esperamos la siguiente iteración

    }

    // Mostrar los valores en el Monitor Serie

    Serial.print("Temp: ");

    Serial.print(temperatura);

    Serial.print(" °C | Hum: ");

    Serial.print(humedad);

    Serial.println(" %");

    // Mostrar los valores en la pantalla LCD

    lcd.clear();  // Limpiamos la pantalla antes de escribir nuevos valores


     lcd.setCursor(0, 0);  // Posicionamos el cursor en la primera columna y primera fila

    lcd.print("Temp: ");

    lcd.print(temperatura);  // Mostramos la temperatura en la pantalla

    lcd.print(" C");  // Agregamos la unidad de medida (°C)

     lcd.setCursor(0, 1);  // Posicionamos el cursor en la primera columna y segunda fila

    lcd.print("Hum: ");

    lcd.print(humedad);  // Mostramos la humedad en la pantalla

    lcd.print(" %");  // Agregamos la unidad de medida (%)

    delay(2000);  // Esperamos 2 segundos antes de realizar una nueva lectura

}
```

Lección 16 Termometro

En esta lección, aprenderás a utilizar el Termistor NTC MF52 10K para medir la temperatura y visualizar los valores en un display LCD 16x2 con interfaz I2C, utilizando una tarjeta Mega 2560.

Materiales necesarios

- Tarjeta Mega 2560
- Termistor NTC MF52 10K
- Protoboard
- Display LCD 16x2 con I2C
- Cables para protoboard HM y MM
- Resistencia de 10K

Conocimientos previos

El Termistor NTC MF52 10K es un sensor cuya resistencia disminuye al aumentar la temperatura. Se usa en monitoreo térmico, control de temperatura y protección contra sobrecalentamiento.

Antes de utilizar este sensor, es útil entender los siguientes conceptos básicos.

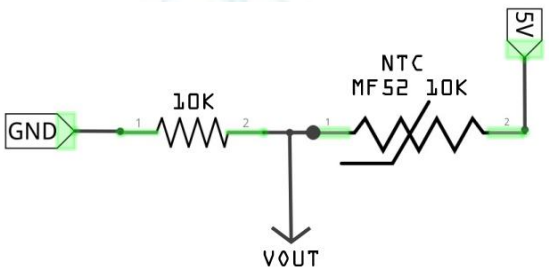
1. Propiedades clave

Para usar este sensor correctamente, es importante conocer sus características y valores esenciales.

Característica	**Valor**
Tipo	NTC (Coeficiente de temperatura negativo)
Resistencia nominal (R25)	10KΩ a 25°C
Coeficiente Beta (β)	3950K
Tolerancia	±1%, ±3%, ±5% (según fabricante)
Rango de temperatura	-40°C a 125°C

2. Funcionamiento del termistor NTC

Su resistencia disminuye cuando la temperatura aumenta, lo que permite medir la temperatura a través de un divisor de voltaje. Para mayor precisión, se usa la ecuación de Steinhart-Hart o la ecuación Beta. A continuación, se muestra el esquemático del divisor de voltaje.



3. Implementación del Termistor NTC

Para medir la temperatura con la tarjeta Mega 2560 o otra tarjeta de desarrollo, el termistor se conecta en un divisor de voltaje junto con una resistencia de 10KΩ. La tarjeta Mega mide el voltaje en el nodo central y calcula la resistencia del termistor con la fórmula:

$$R_{NTC} = \left(\frac{V_{cc} \times R_{fija}}{V_{medido}} \right) - R_{fija}$$

Donde:

- **Vcc** = 5V (Alimentación)
- **Rfija** = 10KΩ (resistencia en serie)
- **Vmedido** = Voltaje leído por la tarjeta Mega en el pin analógico

Luego, la temperatura se obtiene con la ecuación Beta:

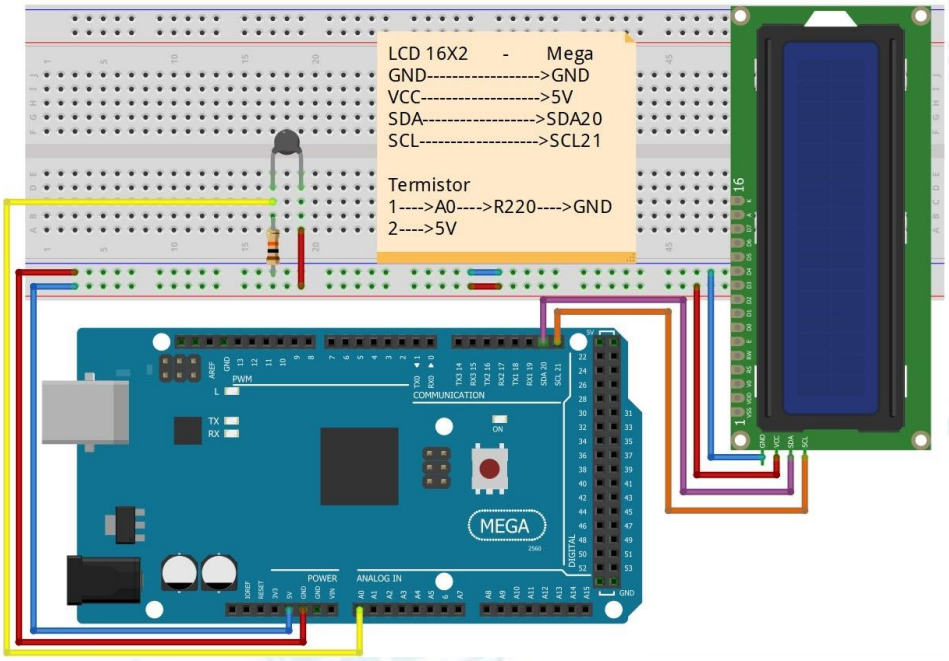
$$T = \frac{B \times T_0}{B + (T_0 \times \ln(R_{NTC}/R_0))}$$

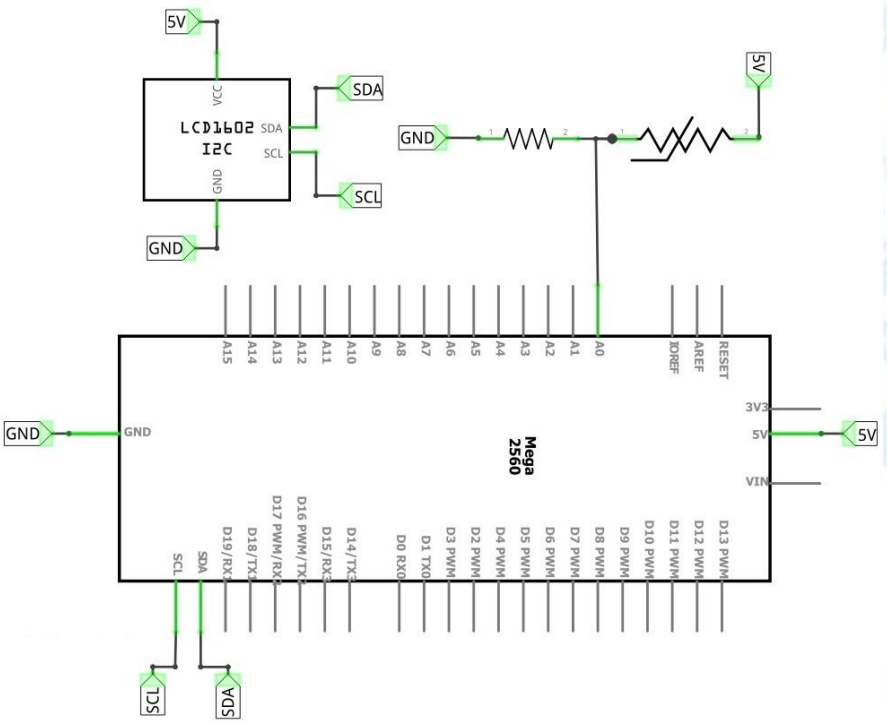
Donde:

- **B** = 3950K (coeficiente Beta)
- **T_0** = 298.15K (temperatura nominal en Kelvin, equivalente a 25°C)
- **R_0** = 10KΩ (resistencia a 25°C)
- **ln** = Logaritmo natural

Diagrama de conexión

A continuación se muestra cómo conectar el sensor NTC MF52 10K y la pantalla LCD 16x2 a la tarjeta Mega.





Código de funcionamiento

El siguiente código permite leer el estado del sensor NTC MF52 10K y visualizar la temperatura con el display LCD 16x2.

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

// Definiciones de pines y constantes

#define TERMISTOR_PIN A0 // Pin analógico donde está conectado el termistor

#define RESISTENCIA 10000 // Valor de la resistencia en ohmios

// Configuración del LCD con dirección I2C 0x27 y dimensiones 16x2

LiquidCrystal_I2C lcd(0x27, 16, 2);

// Función para inicializar el LCD y la comunicación serie

void iniciarSistema() {

    lcd.init(); // Inicializa la pantalla LCD

    lcd.backlight(); // Enciende la retroiluminación del LCD

    Serial.begin(9600); // Inicializa la comunicación serie

    lcd.setCursor(0, 0);

    lcd.print("Temperatura:"); // Muestra el título en la pantalla

}

// Función para obtener la temperatura en grados Celsius

float obtenerTemperatura() {

int lectura = analogRead(TERMISTOR_PIN); // Lee el valor analógico del termistor

float voltaje = lectura * (5.0 / 1023.0); // Convierte la lectura a voltaje

float resistencia = (5.0 * RESISTENCIA / voltaje) - RESISTENCIA; // Calcula la resistencia del termistor

//Cálculo de temperatura en grados Celsius usando la ecuación de Steinhart-Hart

float temperatura = 1.0 / (0.001129148 + (0.000234125 * log(resistencia)) + (0.0000000876741 * pow(log(resistencia), 3))) - 273.15;
```

```
return temperatura;

}

// Función para mostrar la temperatura en la pantalla LCD y en el monitor serie

void mostrarTemperatura(float temperatura) {

Serial.print("Temperatura: ");

Serial.print(temperatura);

Serial.println(" C");

   lcd.setCursor(0, 1);

    lcd.print(temperatura);

    lcd.print(" C   "); // Espacios extra para sobrescribir datos previos

}

void setup() {

    iniciarSistema(); // Llama a la función para inicializar el sistema

}


void loop() {

    float temperatura = obtenerTemperatura(); // Obtiene la temperatura actual

    mostrarTemperatura(temperatura); // Muestra la temperatura en el LCD y en el monitor serie

    delay(1000); // Espera 1 segundo antes de la siguiente lectura

}
```

Lección 17 Servomotor

En esta lección, aprenderás a utilizar el servomotor SG90 y a controlarlo con la tarjeta Mega 2560 mediante un potenciómetro. Este proyecto te permitirá comprender cómo manipular el ángulo del servo en función de la posición del potenciómetro.

Materiales necesarios

- Tarjeta Mega 2560
- Servomotor SG90
- Potenciómetro de 10K
- Protoboard
- Cables para protoboard MM

Conocimientos previos

El Servomotor SG90 es un motor compacto de precisión que gira hasta 180° mediante señales PWM. Es ideal para robótica, modelismo y automatización de pequeños mecanismos. Antes de utilizar el servomotor SG90, es útil entender los siguientes conceptos básicos.

1. **Funcionamiento**

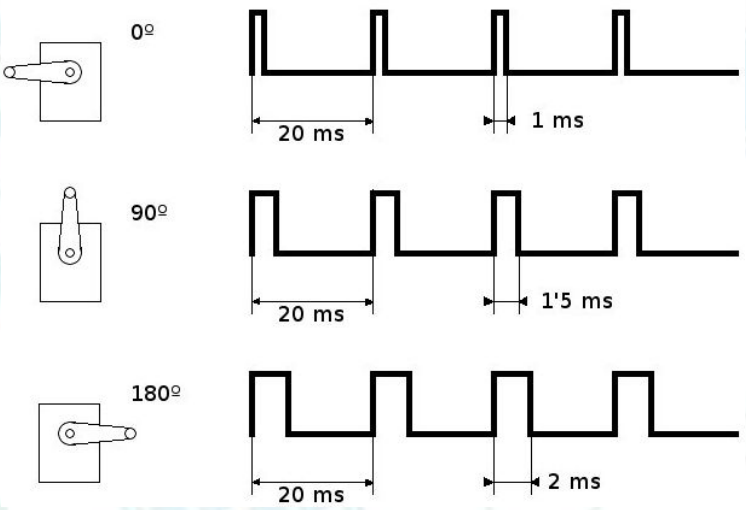
El **Servomotor SG90** es un motor de corriente continua con un sistema de engranajes y un circuito de control interno que permite mover su eje a un ángulo específico dentro de un rango de **0° a 180°**.

Su funcionamiento se basa en señales **PWM (Modulación por Ancho de Pulso)** para determinar la posición del eje.

2. **Señal de Control (PWM)**

- El SG90 recibe una señal de control en forma de pulsos eléctricos a través de su **pin de señal**.
- La duración de estos pulsos determina el ángulo al que se moverá el eje.
- Generalmente, una señal de **0.5 ms** lo posiciona en **0°**, **1.5 ms** en **90°** y **2.5 ms** en **180°** (aunque esto puede variar ligeramente según el modelo).

La siguiente imagen muestra cómo la señal PWM controla la posición del Servomotor SG90. Cada pulso tiene un período de 20 ms, mientras que la duración del pulso alto determina el ángulo del servo: 1 ms para 0°, 1.5 ms para 90° y 2 ms para 180°.



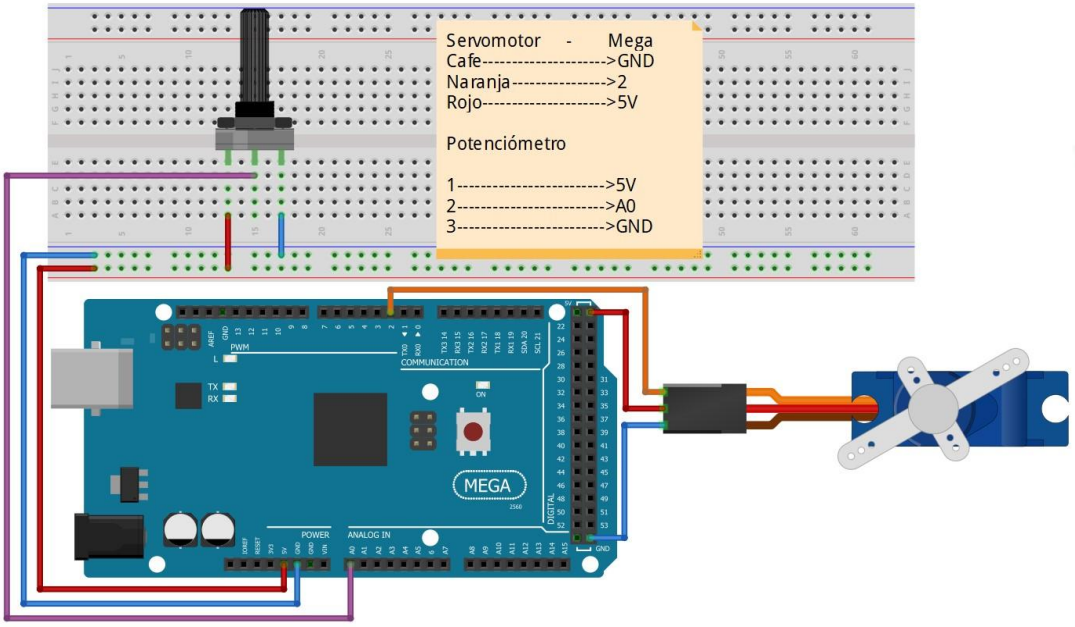
3. Pines

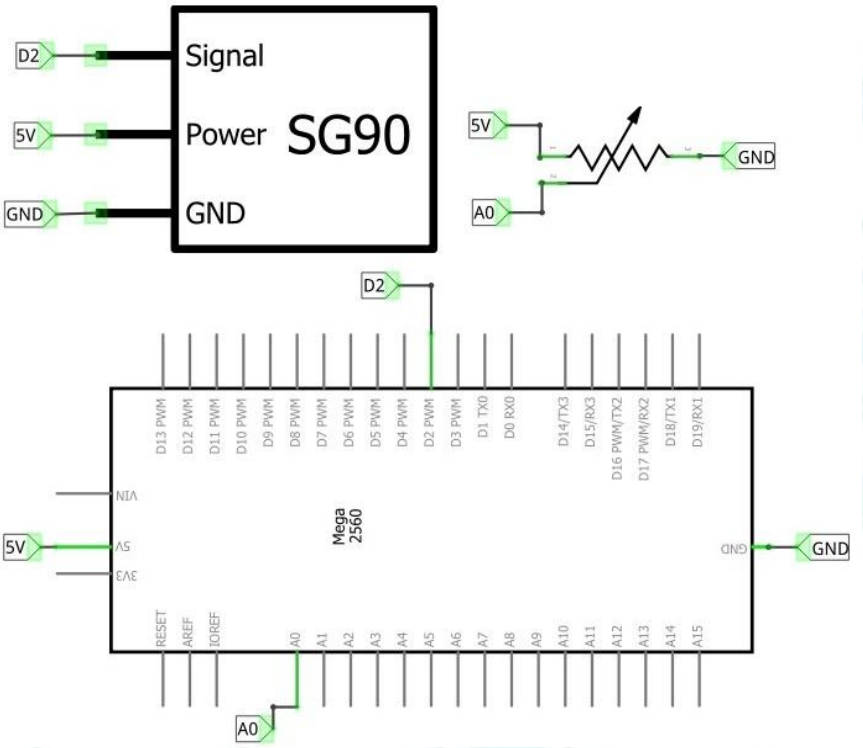
El **Servomotor SG90** tiene 3 pines que deben conectarse correctamente para su funcionamiento.

Pin	Color del Cable	Descripción
VCC	Rojo	Alimentación (4.8V - 6V)
GND	Cafe	Tierra (GND)
Señal	Naranja	Entrada PWM para el control del ángulo

Diagrama de conexión

A continuación, se muestra cómo conectar el Servomotor SG90 y el potenciómetro a la tarjeta Mega 2560.





Código de funcionamiento

El siguiente código permite leer los valores del potenciómetro para tener control del servomotor para moverlo de 0° a 180°.

```
#include <Servo.h>  // Librería para controlar servos

Servo servoMotor;  // Crear objeto servo

int pinPot = A0;   // Pin analógico del potenciómetro

int pinServo = 2;  // Pin PWM para el servomotor

void setup() {

    servoMotor.attach(pinServo);  // Asignar el pin del servo

}

void loop() {

    int valorPot = analogRead(pinPot);  // Leer el valor del potenciómetro

     int angulo = map(valorPot, 0, 1023, 0, 180);  // Convertir valor a grados
(0-180)

    servoMotor.write(angulo);  // Mover el servo a la posición correspondiente

    delay(15);  // Pequeña pausa para estabilidad

}
```

Lección 18 Control de motor a pasos

En esta lección, aprenderás a utilizar el motor a pasos 28BYJ-48 con su driver ULN2003, controlándolo con la tarjeta Mega 2560 y el sensor IR KY-022 y su control remoto. Al finalizar, podrás encender y apagar el motor, así como cambiar su dirección de giro con el control remoto.

Materiales necesarios

- Tarjeta Mega 2560
- Protoboard
- Motor a pasos 28BYJ-48
- Driver ULN2003
- Fuente de alimentación para protoboard
- Sensor IR KY-022 con Control
- Cables para protoboard MM y HM

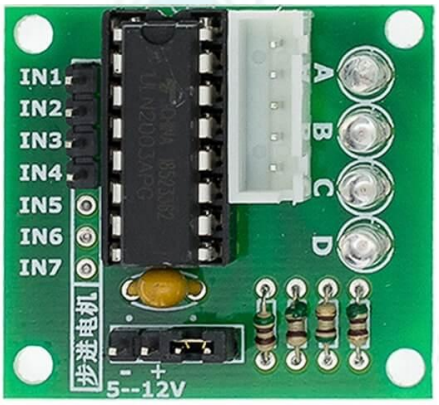
Conocimientos previos

Antes de utilizar el motor a pasos 28BYJ-48, es útil entender los siguientes conceptos básicos.

El **28BYJ-48** es un motor a pasos unipolar de 5 fases que se utiliza comúnmente en proyectos de electrónica, automatización y robótica. A diferencia de los motores DC, que giran de forma continua cuando reciben energía, los motores a pasos giran en pequeños incrementos llamados pasos, lo que les permite un control preciso de la posición y velocidad.

Este motor a pasos tiene un módulo de control o **Driver ULN2003** que funciona como interfaz entre el motor y el microcontrolador o tarjeta de desarrollo. Su función es amplificar la señal de control y proporcionar la corriente necesaria para activar el motor, además de tener las siguientes características.

- **Protección** → Evita sobrecargas en el microcontrolador o tarjeta de desarrollo.
- **Facilidad de conexión** → Se conecta directamente al motor y al microcontrolador o tarjeta de desarrollo.
- **Distribuye la corriente** → Permite controlar las 4 bobinas del motor de manera eficiente.



¿Cómo utilizar el Motor 28BYJ-48 con su driver ULN2003?

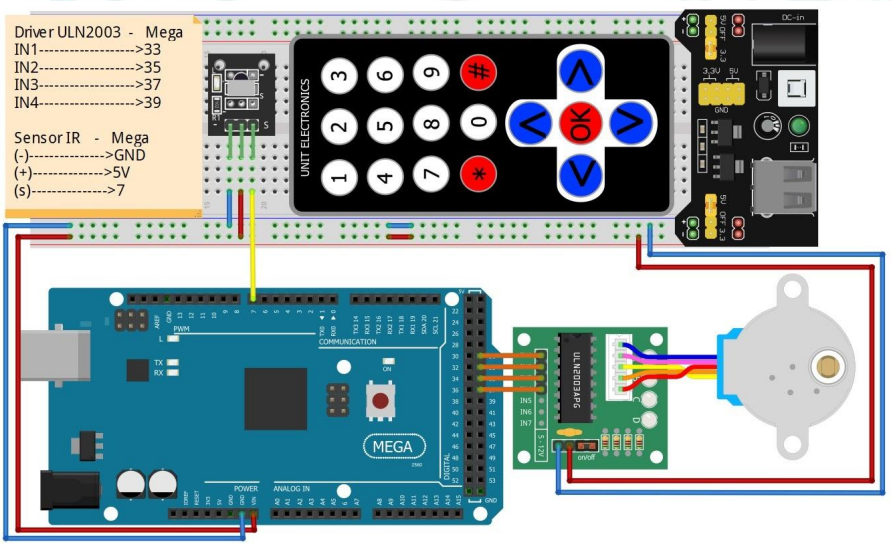
Para usar este motor con la tarjeta Mega 2560, se deben seguir los siguientes pasos:

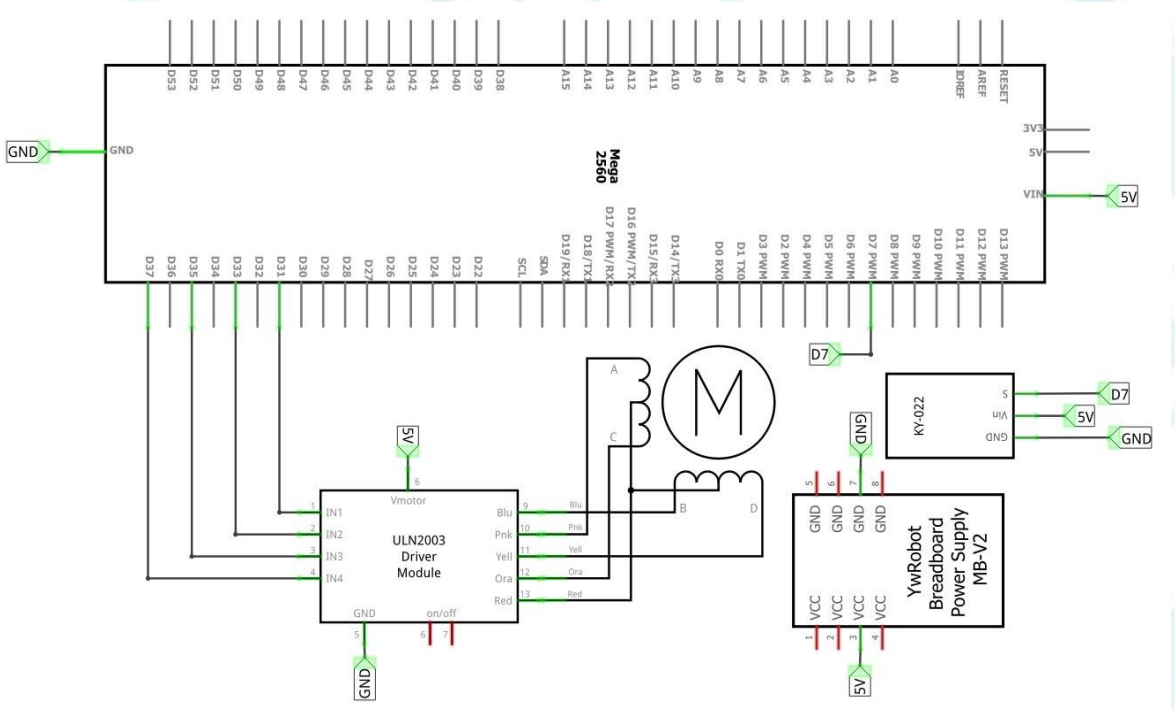
1. Conectar el motor al driver ULN2003 utilizando su conector de 5 pines.
2. Alimentar el driver con 5V mínimo o máximo 12V de una fuente externa.
3. Conectar las señales de control desde la tarjeta Mega 2560 a las entradas del driver (IN1, IN2, IN3, IN4).
4. Programar la tarjeta Mega 2560 para enviar pulsos al driver, indicando la dirección y velocidad del motor.

Con estas conexiones y el código, el motor podrá girar en ambos sentidos y con control de velocidad y posición.

Diagrama de conexión

A continuación, se muestra como conectar el motor a pasos 28BYJ-48 con su driver ULN2003, el sensor IR KY-022 y la tarjeta Mega 2560, junto con la fuente de alimentación para la protoboard





Código de funcionamiento

Este código permite **encender y apagar el motor**, así como **cambiar la dirección de giro** presionando los botones del control.

Librerías utilizadas

- **IRremote.h** → Permite recibir señales del control remoto (Ya instalada en prácticas anteriores).
- **Stepper.h** → Controla el motor a pasos (Incluida por defecto en el Arduino IDE, no requiere instalación).

Botones del control remoto y su función

Botón	**Código HEX**	**Función**
Izquierda	`0xFF10EF`	Gira el motor en una dirección
Derecha	`0xFF5AA5`	Invierte la dirección de giro
OK	`0xFF38C7`	Detiene el motor

Si deseas utilizar otros botones del control remoto, puedes **abrir el Monitor Serie del IDE de Arduino** y observar los códigos que se reciben al presionarlos. Luego, solo reemplázalo en el código.

```
#include <IRremote.h>
#include <Stepper.h>
// Definir el pin donde está conectado el sensor IR
#define IR_RECEIVER_PIN 7
// Definir el número de pasos por vuelta del motor 28BYJ-48
#define STEPS 2048
// Definir los pines de conexión del ULN2003 a la tarjeta mega
#define IN1 33
#define IN2 35
#define IN3 37
#define IN4 39
// Crear objeto Stepper para controlar el motor
Stepper stepper(STEPS, IN1, IN3, IN2, IN4);
// Crear objeto IRrecv para recibir señales del control remoto
IRrecv irrecv(IR_RECEIVER_PIN);
decode_results results;
// Variables de control del motor
bool motorEncendido = false; // Estado del motor (apagado por defecto)
int sentido = 1; // Dirección del giro: 1 = horario, -1 = antihorario
void setup() {
    Serial.begin(9600); // Iniciar comunicación serie para depuración
    irrecv.enableIRIn();  // Iniciar receptor IR
    stepper.setSpeed(10); // Establecer velocidad del motor en RPM
}
void loop() {
    // Verificar si se ha recibido una señal del control remoto
    if (irrecv.decode(&results)) {
        Serial.print("Código recibido: ");
          Serial.println(results.value, HEX); // Mostrar el código en formato
hexadecimal
        // Evaluar qué botón se presionó en el control remoto
```

```
        switch (results.value) {
            case 0xFF5AA5:  // Botón de la derecha
                motorEncendido = true; // Encender el motor
                sentido = 1;  // Configurar giro en sentido horario
                break;
            case 0xFF10EF:  // Botón de la izquierda
                motorEncendido = true; // Encender el motor
                sentido = -1; // Configurar giro en sentido antihorario
                break;
            case 0xFF38C7:  // Botón OK
                motorEncendido = false; // Apagar el motor
                break;
        }
        irrecv.resume(); // Habilitar recepción de la siguiente señal IR
    }
    // Si el motor está encendido, girar en la dirección establecida
    if (motorEncendido) {
        stepper.step(sentido * 10); // Mover el motor 10 pasos en la dirección indicada
    }
}
```

Lección 19 Teclado Matricial, LCD y RGB

En esta lección, aprenderás a utilizar el teclado matricial 4x4 para implementar un sistema de control de acceso. Utilizaremos una pantalla LCD para ingresar y visualizar contraseñas, y un LED RGB para indicar el estado del sistema: cerrado o con acceso permitido.

Materiales necesarios

- Tarjeta Mega 2560
- Protoboard
- Resistencias de 220 ohms
- Cables para protoboard MM y HM
- Leds RGB Cátodo Común
- Teclado Matricial 4×4
- Display LCD 16x2

Conocimientos previos

El teclado matricial 4x4 es un componente que permite la entrada de datos mediante 16 teclas organizadas en una matriz de 4 filas y 4 columnas. Antes de utilizar este sensor, es útil entender los siguientes conceptos básicos.

1.- Funcionamiento

Cada tecla del teclado está conectada a una fila y una columna. Cuando se presiona una tecla, se cierra el circuito entre una fila y una columna específica, permitiendo detectar la tecla presionada mediante el escaneo de filas y columnas.

2.- Pines del teclado 4x4

- **4 pines de filas**: Se utilizan como salidas para enviar señales.
- **4 pines de columnas**: Se utilizan como entradas para detectar las teclas presionadas.

En la siguiente imagen se muestran sus pines correspondientes a filas (F) y columnas (C).

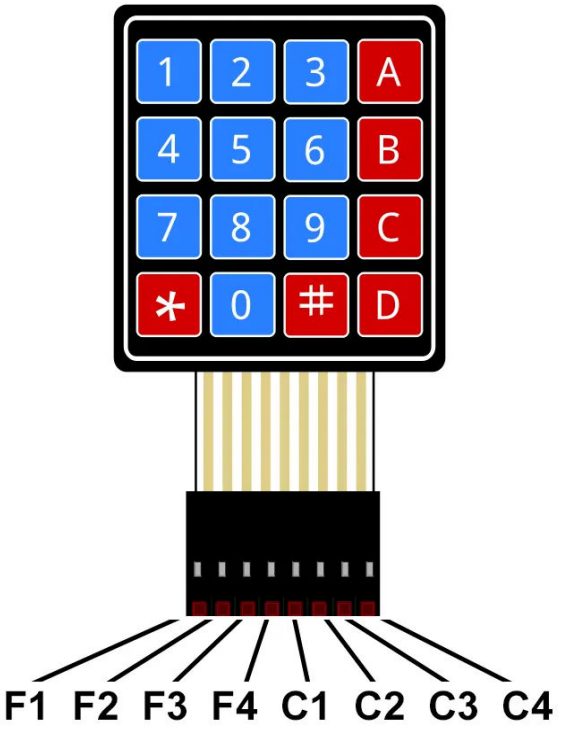
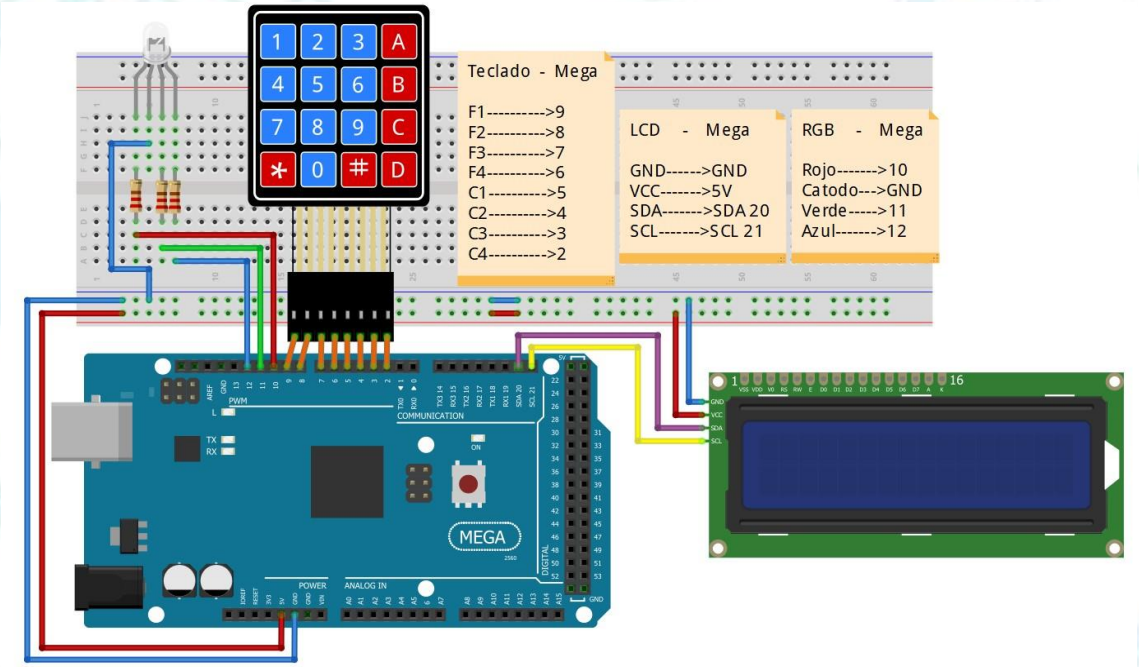
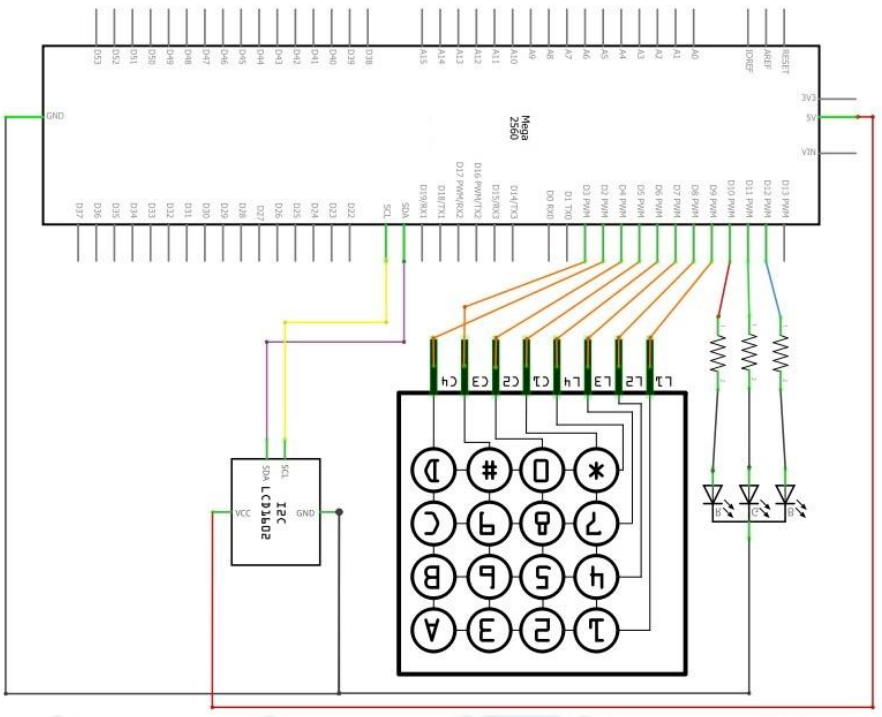


Diagrama de conexión

A continuación se muestra cómo conectar el teclado matricial 4x4, la pantalla LCD 16x2 I2C y el LED RGB.





Código de funcionamiento

Este código implementa un sistema de control de acceso con el teclado matricial 4x4, la pantalla LCD 16x2 I2C y el LED RGB. El usuario ingresa una contraseña en el teclado, que se muestra en la LCD y se valida al presionar #. Si es correcta, el LED cambia a verde por 2 segundos y luego vuelve a rojo para indicar que el acceso está restringido. En caso de error, se muestra un mensaje en la LCD y se reinicia el proceso. Además, el usuario puede reiniciar la entrada de la contraseña *, haciendo el sistema dinámico y fácil de usar.

Antes de cargar el código, asegúrate de instalar las siguientes librerías en el Arduino IDE para su correcto funcionamiento.

LiquidCrystal_I2C.h	**Keypad.h**
	

Puedes instalar ambas librerías desde el **Gestor de Librerías** en el **Arduino IDE**. Una vez instaladas, compila y carga el código en la **tarjeta Mega** para comenzar a utilizar el sistema.

```
// Librerías necesarias

#include <Wire.h>                 // Comunicación I2C

#include <LiquidCrystal_I2C.h>     // Manejo del LCD 16x2 con I2C

#include <Keypad.h>               // Manejo del teclado matricial



// Inicialización del LCD 16x2 con dirección I2C (puede ser 0x27 o 0x3F según el módulo)

LiquidCrystal_I2C lcd(0x27, 16, 2);



// Definición de la matriz del teclado 4x4

const byte FILAS = 4;       // Número de filas

const byte COLUMNAS = 4;    // Número de columnas

// Asignación de teclas al teclado matricial 4x4

char teclas[FILAS][COLUMNAS] = {

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

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

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

  {'*', '0', '#', 'D'}  // '*' para borrar, '#' para validar

};

// Definición de los pines de conexión del teclado a la placa

byte pinesFilas[FILAS] = {9, 8, 7, 6};         // Pines de las filas

byte pinesColumnas[COLUMNAS] = {5, 4, 3, 2};   // Pines de las columnas

// Creación del objeto teclado

Keypad teclado = Keypad(makeKeymap(teclas), pinesFilas, pinesColumnas, FILAS, COLUMNAS);
```

```
// Definición de los pines del LED RGB (cátodo común)
const int pinRojo = 10;
const int pinVerde = 11;
const int pinAzul = 12;

// Contraseña de acceso
String password = "1234";    // Se puede modificar según necesidad
String inputPassword = "";   // Variable para almacenar la clave ingresada por el usuario

void setup() {
  Serial.begin(9600);  // Iniciar comunicación serial
  // Configurar pines del LED como salida
  pinMode(pinRojo, OUTPUT);
  pinMode(pinVerde, OUTPUT);
  pinMode(pinAzul, OUTPUT);
  // Iniciar LCD
  lcd.init();         // Inicializar el LCD
  lcd.backlight();    // Encender la retroiluminación del LCD
  // Mensaje de inicio en el LCD
  lcd.setCursor(0, 0);
  lcd.print("  Ingresa la clave  ");
  // Encender el LED en rojo al inicio (indica bloqueo)
  encenderLEDRojo();
}
void loop() {
  char tecla = teclado.getKey();  // Leer la tecla presionada
  if (tecla) {  // Si se presiona una tecla
    Serial.print("Tecla presionada: ");
    Serial.println(tecla);
    // Mostrar la clave ingresada en el LCD
    lcd.setCursor(0, 1);
    lcd.print("Clave: ");
    lcd.print(inputPassword + tecla);
```

```
// Si se presiona '#', se valida la contraseña
 if (tecla == '#') {
   if (inputPassword == password) {  // Comparar con la clave correcta
     Serial.println("Contraseña correcta");
     // Mostrar mensaje de acceso concedido en el LCD
     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print("Acceso correcto!");
     lcd.setCursor(0, 1);
     lcd.print("Bienvenido");
     // Encender LED verde (indica acceso concedido)
     encenderLEDVerde();
     delay(2000);  // Esperar 2 segundos
     encenderLEDRojo();  // Volver a rojo después de la espera
     // Restaurar mensaje de inicio
     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print("  Ingresa la clave  ");
   } else {
     Serial.println("Contraseña incorrecta");
     // Mostrar mensaje de error en el LCD
     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print("Clave incorrecta");
     lcd.setCursor(0, 1);
     lcd.print("Reintente...");
     delay(2000);
     // Restaurar mensaje de inicio
     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print("  Ingresa la clave ");
   }
```

```
      inputPassword = "";  // Borrar la clave ingresada tras validar
    }
      else if (tecla == '*') {   // Si se presiona '*', se borra la clave ingresada
      inputPassword = "";
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Clave reiniciada");
      delay(2000);
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("  Ingresa la clave  ");
    }
    else {
      inputPassword += tecla;  // Agregar la tecla ingresada a la contraseña
    }
  }
}
// Función para encender el LED en verde (acceso permitido)
void encenderLEDVerde() {
  digitalWrite(pinRojo, LOW);
  digitalWrite(pinVerde, HIGH);
  digitalWrite(pinAzul, LOW);
}
// Función para encender el LED en rojo (acceso denegado)
void encenderLEDRojo() {
  digitalWrite(pinRojo, HIGH);
  digitalWrite(pinVerde, LOW);
  digitalWrite(pinAzul, LOW);
}
```

Lección 20 Control de motor DC

En esta lección, aprenderás a controlar la velocidad y el sentido de giro un motor DC pequeño usando el Mega 2560 y el circuito integrado L293D a través de señales PWM por medio de la sentencia analogWrite();

Materiales necesarios

- Mega 2560
- Portoboard
- Módulo de fuente de alimentación para protoboard
- Circuito integrado L293D
- M-M Cables jumper
- Eliminador 9V 1A
- Protoboard
- Motorreductor 48:1

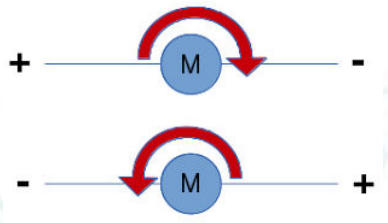
Conocimientos previos

Control del giro de un motor DC

Para cambiar el sentido de giro de un motor de corriente continua (DC), se logra invirtiendo la polaridad de la corriente que llega al motor. Hay dos formas principales de hacerlo:

1. Invertir las conexiones de los cables de alimentación

El motor de corriente continua tiene dos terminales de entrada: uno positivo (+) y uno negativo (-). Si inviertes las conexiones, es decir, conectas el terminal positivo a lo que originalmente estaba conectado al terminal negativo, y viceversa, el motor cambiará el sentido de su giro.

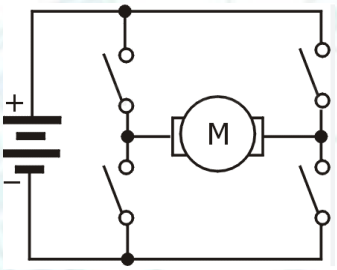


2. Uso de un puente H

Un puente H es un circuito electrónico cuya función principal es permitir que la corriente fluya en ambas direcciones a través del motor que a través de un microcontrolador o circuito lógico digital se lleva a cabo sin la incómoda necesidad de intercambiar la polaridad hacia las conexiones físicas.

¿Cómo funciona un puente H?

Un puente H está formado por cuatro interruptores (transistores o relés) dispuestos de tal manera que se forma una figura de "H". Los interruptores están conectados de la siguiente manera:



Cuando los interruptores se abren o se cierran en diferentes combinaciones, puedes hacer que la corriente fluya de manera que el motor gire en una dirección u otra.

Circuito integrado L293D

El **L293D** es un circuito integrado (IC) que se utiliza como un controlador de motores en proyectos electrónicos. Es específicamente un **puente H** integrado que permite:

1. Control de dirección y velocidad (hacia adelante o hacia atrás) mediante señales de control y la velocidad mediante un pulso de modulación (PWM).
2. Puede manejar voltajes de 4.5V a 36V, lo que lo hace adecuado para motores pequeños y medianos que no consuman más de 600mA
3. Controlar dos motores de forma independiente, lo que lo hace ideal para proyectos como robots.

Pin 1 (EN 1, 2): Este pin habilita los canales 1 y 2 (que controlan el primer motor). Si está en alto (5V), los canales estarán activos, permitiendo el control del motor. Si está en bajo (0V), los canales estarán deshabilitados.

Pin 9 (EN 3, 4): Similar al pin 1, este pin habilita los canales 3 y 4 (que controlan el motor 2). Si está en alto (5V), los canales estarán activos.Se conecta a Vcc (5V) para habilitar el control de los motores.

Pines de entradas 2, 7 10 y 4 (1A, 2A, 3A y 4A): Estos pines reciben las señales de control que determina la dirección de los motores. Se conectan a un pin de salida de un microcontrolador o dispositivo de control

Pines de salidas 3, 6, 11 y 14 (1Y, 2Y, 3Y, 4Y): Estos pines son las salidas que conectan a los motores. El voltaje aquí depende de las señales de control en los pines de entradas. Se conectan a las terminales de los motores.

Pin 8 (Vcc2): Este es el pin de alimentación del motor (fuente de voltaje para los motores). Generalmente se conecta a un voltaje adecuado para el motor, que puede ser diferente del voltaje de control (Vcc1). Se conecta a la fuente de alimentación para los motores de 4.5V a 36V.

Pin 16 (Vcc1): Es el pin de alimentación del circuito integrado L293D que requiere de 5v .

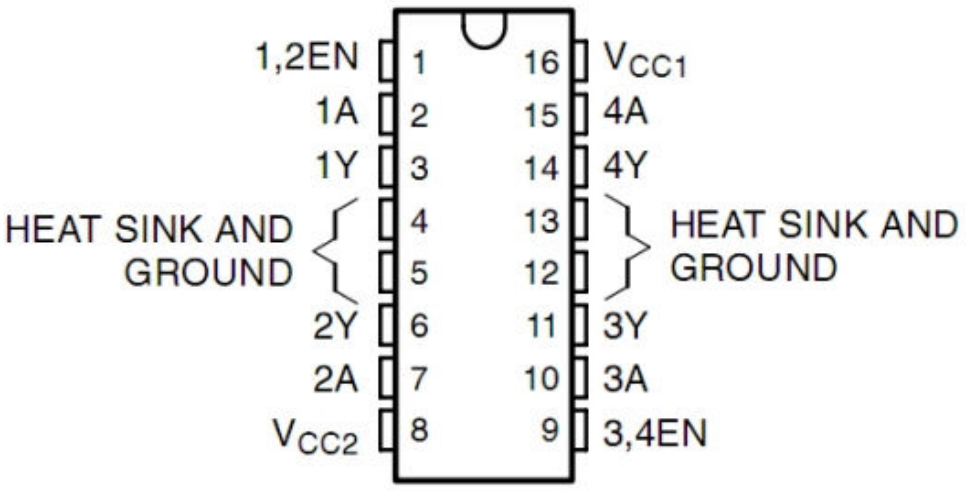
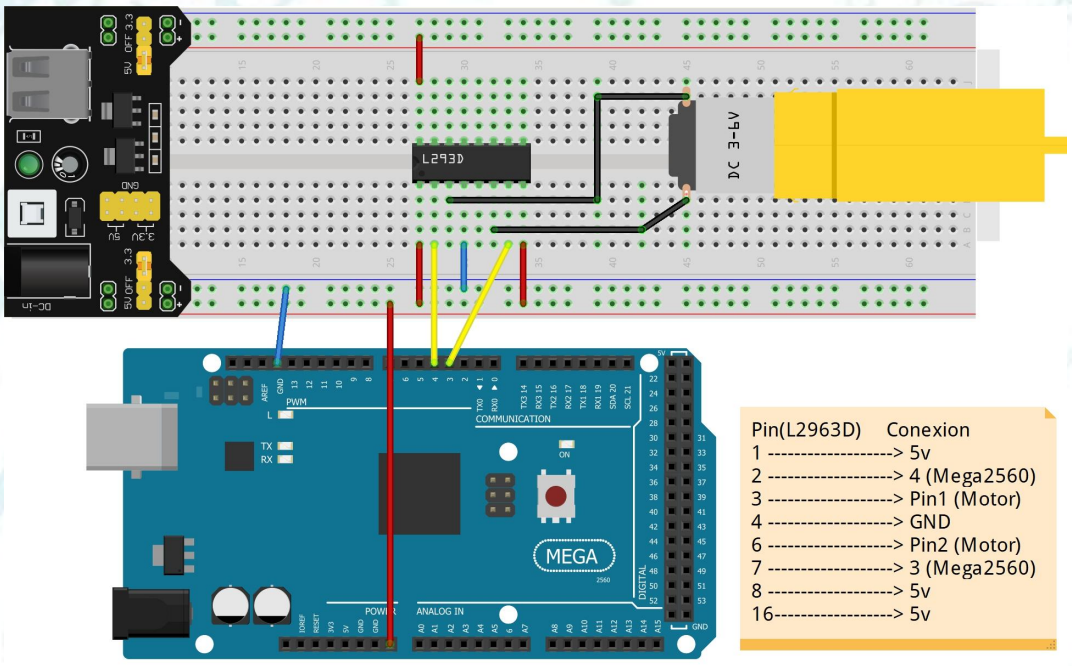
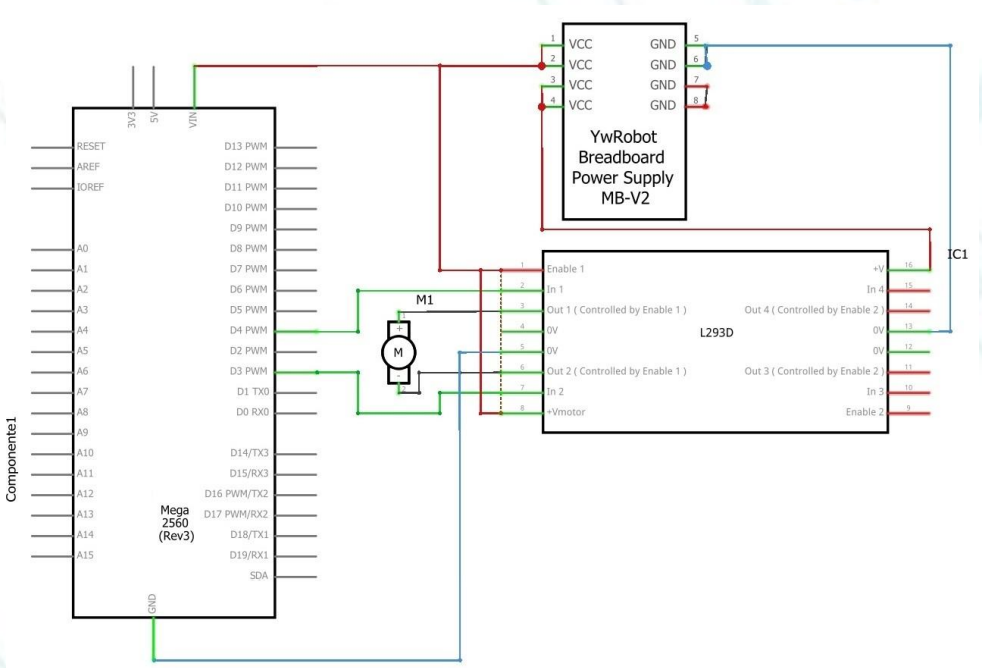


Diagrama de conexión

A continuación se muestran las conexiones entre el circuito integrado L293D, el módulo de alimentación para protoboard al Mega 2560 y el motor a controlar.





Código de funcionamiento

```
#define I1 3// PIN 2 del L293D

#define I2 4 // PIN 7 del L293D

void setup() {

 pinMode(I1, OUTPUT);

 pinMode(I2, OUTPUT); // Declaramos ambos pines como salidas

}

void loop() {

 analogWrite(I1, 123);// Gira en un sentido a la mitad de la velocidad

 digitalWrite(I2, LOW);

 delay(5000);

 digitalWrite(I1, LOW);

 digitalWrite(I2, LOW); // Realiza una parada

 delay(5000);

 digitalWrite(I1, LOW);

 analogWrite(I2, 255);   // Gira en un sentido opuesto al del inicio a
la máxima velocidad

 delay(5000);

}
```

Lección 21 Rele

En esta leccion aprenderas a controlar un relevador con un pulsador desde el Mega 2560 utilizando un transistor BJT tipo NPN

Materiales necesarios

- Resistencias 10k, 22k, 100 todas a 1/4w
- Transistor BC547
- Diodo 1N4007
- LED
- Relevador SPDT 5V
- Módulo de fuente de alimentacion para protoboard
- Eliminador 9V 1A
- Tarjeta Mega 2560
- Protoboard
- Push button

Conocimientos previos

Relevador

Un relevador (también conocido como relé) es un dispositivo electromecánico que se utiliza para abrir o cerrar circuitos eléctricos. Funciona como un interruptor controlado por una señal eléctrica. Cuando una corriente pasa por la bobina del relevador, esta crea un campo magnético que mueve una palanca o un conjunto de contactos, lo que a su vez cambia el estado del circuito (encender o apagar). Los relevadores se usan comúnmente en aplicaciones como:

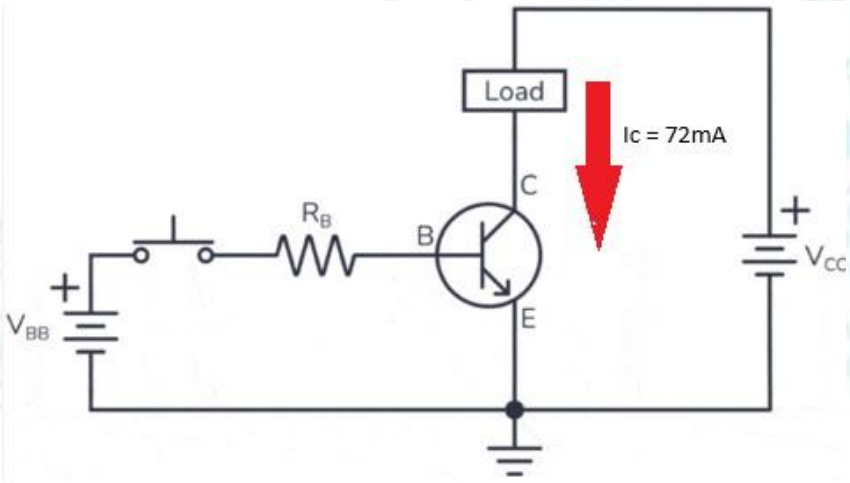
- Automatización de sistemas: para controlar dispositivos de alta potencia con señales de baja potencia.
- Protección de circuitos: como parte de sistemas de seguridad en equipos eléctricos.
- Controles de maquinaria y sistemas de control.

BJT como switch electrónico

Para lograr la activación del relevador tenemos que hacer fluir una corriente a través de la bobina del mismo. Dicha corriente es de 72mA como se observa en la imagen

Esto resulta problemático debido a que no debemos intentar conectar directamente el relevador a la tarjeta Mega 2560 ni a ningún microcontrolador o circuito integrado que no disponga de la suficiente corriente a la salida para manejar al relé. Es por esto que utilizaremos un transistor BC547 el cual es del tipo NPN y puede manejar corrientes de 100mA como máximo.

Por lo tanto el problema del manejo de la corriente se reduce al cálculo de una resistencia de base (Rb) que le permita al transistor lidiar con los 72mA (Icq) que necesita el relevador (Load).



Procedimiento

1.- Identificación de datos

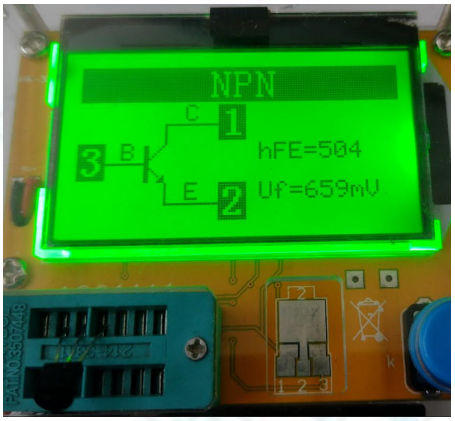
Del circuito mostrado en la figura superior sabemos que la fuente de tensión VBB refiere al Mega 2560 y los 5V que esta puede dar en cada uno de sus salidas digitales, por lo tanto $V_{BB} = 5\,\mathrm{V}$

Load hace referencia a la carga que se va a manejar, en este caso el relevador con los 72mA de corriente que tienen que transitar por el colector del transistor, por consiguiente $I_{CQ} = 72\,\mathrm{mA}$

También necesitamos el dato de la tensión entre la base y el emisor que se produce cuando el transistor entra en conducción, que típicamente es de 0.7v, por ello: $V_{BE} = 0.7\,\mathrm{V}$

Por último necesitamos el dato de la ganancia de corriente del transistor BC547 en CD, identificada con los símbolos “hFe” o “β” que puedes obtener de la hoja de datos o midiendo directamente el transistor con un probador de componentes o un multímetro con la función hFe. El hFe que se obtuvo para el presente ejemplo fue de 504, por lo tanto $h_{FE} = 504$

Tal y como se observa en la imagen donde se obtuvo la medición



2.- Cálculo de la corriente de base

Reemplazando con los datos antes reunidos, calculamos la corriente de base por medio de la siguiente fórmula:

$$I_{BQ} = \frac{I_{CQ}}{h_{FE}}$$

por lo que tenemos:

$$I_{BQ} = \frac{72\,\mathrm{mA}}{504} = 142\,\mu\mathrm{A}$$

Esta corriente sera la que el Mega 2560 tendrá que aportar para que el circuito funcione y como ves, se trata de un valor 500 veces menor que los 72mA que el tramnsistor va a manejar, asi evitamos dañar al microcontrolador

3.- Cálculo de la resistencia de base

Finalmente todos los datos hasta ahora recopilados y calculados los reemplazamos en la siguiente fórmula para obtener el valor de resistencia que nos garantice un correcto funcionamiento además de seguro ya que no existirán sobrecorrientes que puedan provocar daños al equipo.

$$R_B = \frac{V_{BB} - V_{BE}}{I_{BQ}}$$

Sustituyendo datos tendremos:

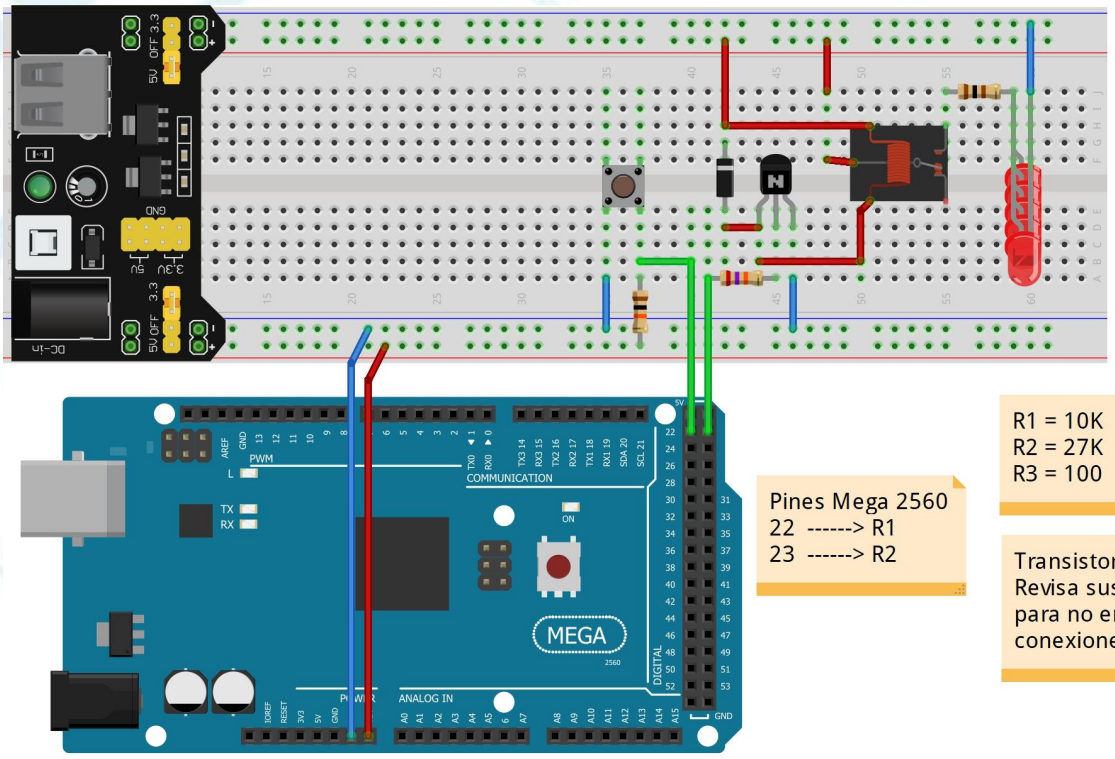
$$R_b = \frac{V_{BB} - V_{BE}}{I_{BQ}} = \frac{5V - 0.7V}{142\mu A} = 29,577\,\Omega$$

Dado que este valor de resistencia no es comercial, siempre utilizaremos el valor inferior más próximo al calculado que en este caso será una resistencia de base de $R_b = 27\,k\Omega$

Diagrama de conexión

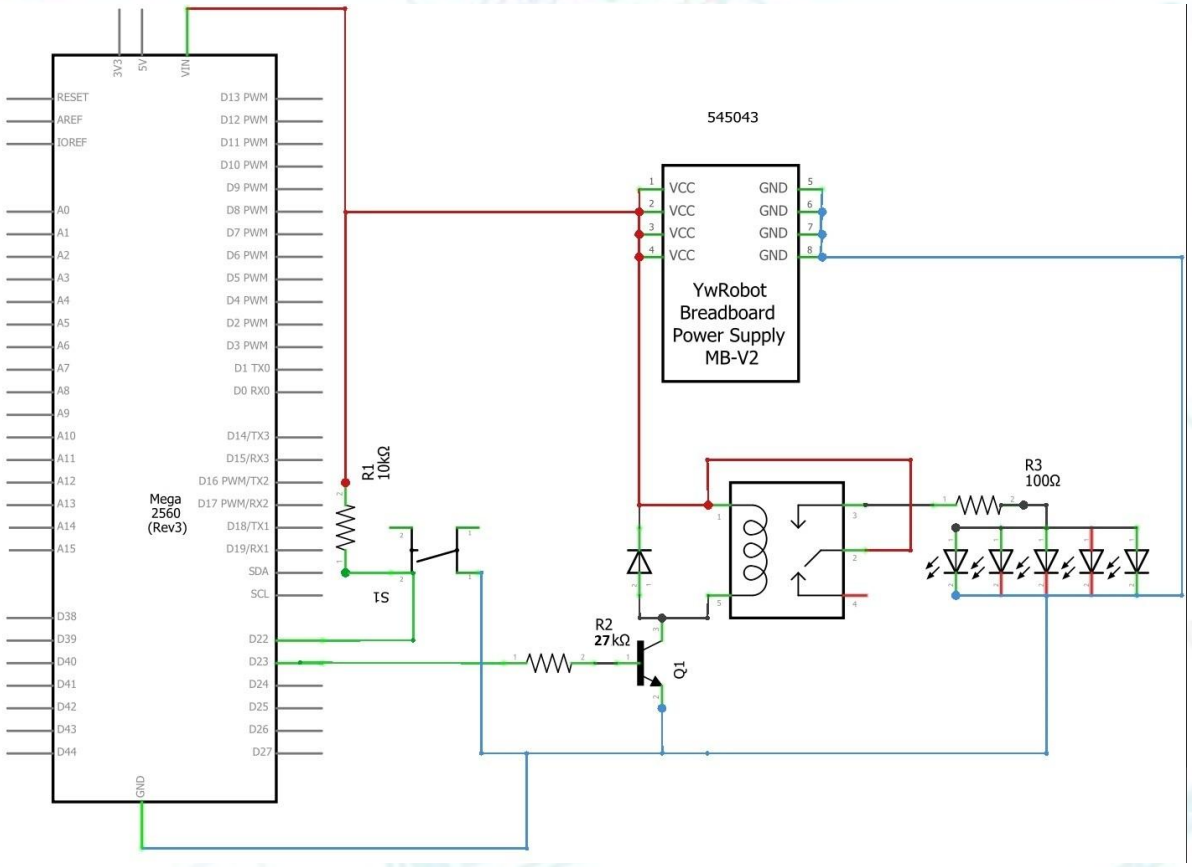
En el diagrama poner especial atención a las resistencias ya que R1 va al botón, R2 al transistor y R3 a los leds

Es muy importante que conecte correctamente el diodo 1N4007 ya que este va a impedir que la bobina destruya al transistor debido a que esta genera picos de tensión muy elevados al conmutar, dichos picos son redirigidos por el diodo brindando protección.



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

El siguiente código permite la activación de un transistor que a través de un relevador enciende 5 leds tras presionar un botón conectado en el pin 22 del Mega 2560. El uso de las sentencias while dentro del void loop permite la enclavación de estado del transistor para que únicamente con un pulso del botón las luces permanezcan encendidas

```
#define boton 22      // Conectamos el boton al pin digital 22 del mega

#define transistor 23 // Conectamos la resistencia de base al pin digital 23 del mega 2560

int estado = LOW;

void setup() {

pinMode(boton, INPUT);           // declaramos al botón como entrada

pinMode(transistor, OUTPUT);    // declaramos al transistor como salida

digitalWrite(transistor, HIGH); // El iniciar el programa el transistor estará en estado alto

}

void loop() {

   while(digitalRead(boton) == LOW){}    // Mientras que el botón esté presionado no hace nada

      estado = digitalRead(transistor);          // Estado guarda el comportamiento del transistor

    digitalWrite(transistor, !estado);  // Se escribe en el transistor el estado opuesto que previamente tenía

  while(digitalRead(boton) == HIGH){}  // Mientras que el botón no esté presionado no hace nada

}
```

Lección 22 Matriz LED MAX7219

En esta lección aprenderás a mostrar caracteres en un módulo de Matriz LED MAX7219 utilizando el Mega 2560.

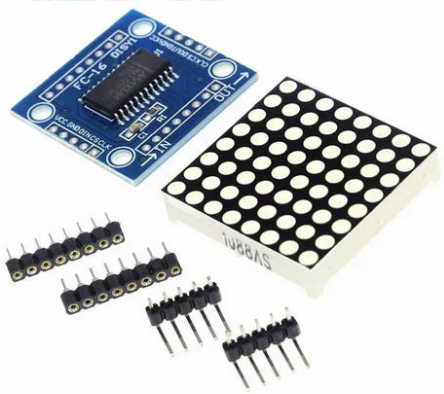
Materiales necesarios

- Tarjeta Mega 2560 x1
- Módulo MAX7219 x1
- Matriz LED de puntos 8X8 x1
- Cables jumper Macho-Hembra

Conocimientos previos

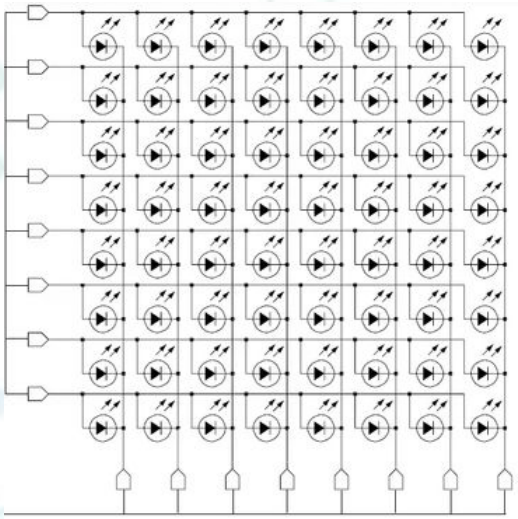
Módulo MAX7219

El módulo LED 8x8 MAX7219 es un controlador de matriz de LEDs que facilita la conexión y control de matrices de 8x8 LEDs (64 LEDs en total) a través de un bus de datos, usando una interfaz SPI



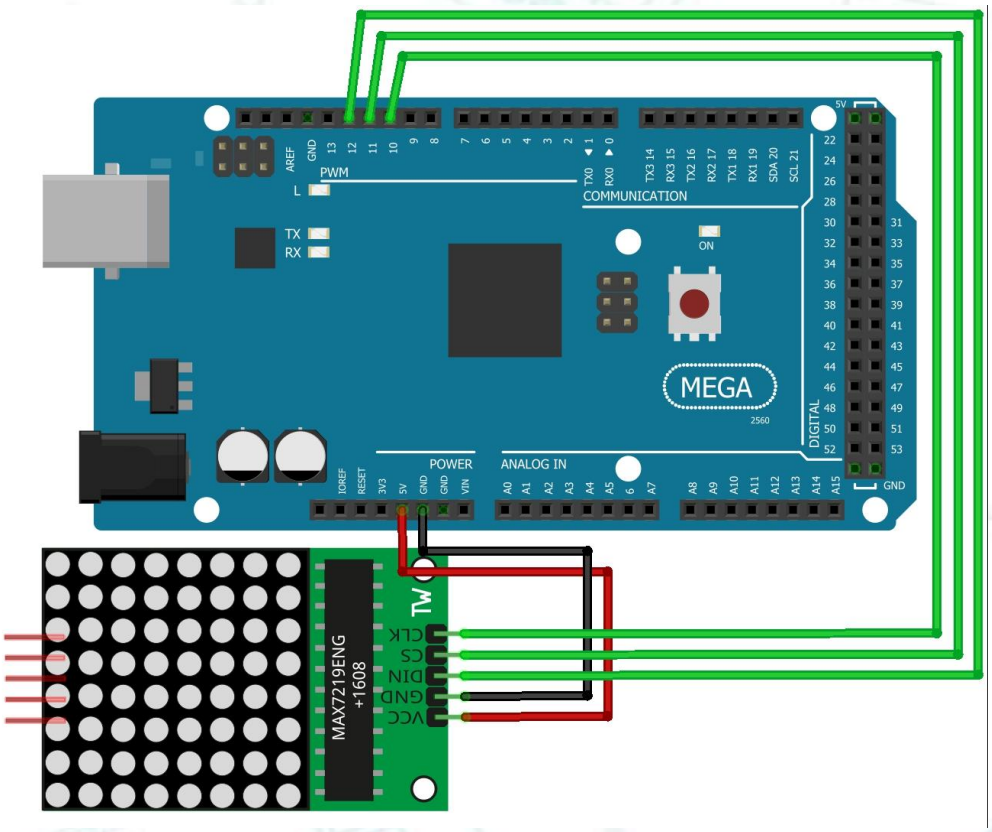
1. El módulo tiene una matriz de 64 LEDs organizados en 8 filas y 8 columnas. Cada LED puede ser encendido o apagado individualmente, y el MAX7219 se encarga de manejar la

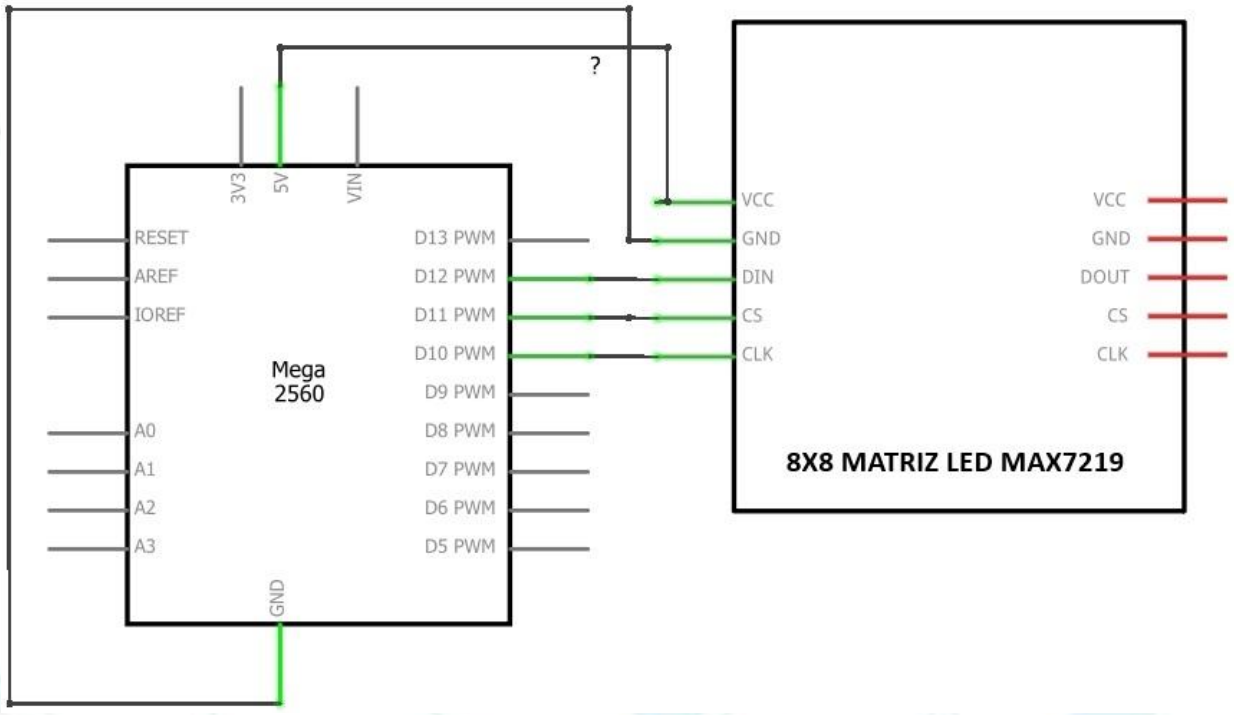
multiplexación (es decir, encender y apagar los LEDs rápidamente para que la matriz parezca completamente iluminada a simple vista debido a la persistencia de la visión humana).



2. El MAX7219 es un integrado que utiliza un protocolo de comunicación SPI (Serial Peripheral Interface), lo que significa que los datos se transmiten de manera secuencial (bit por bit) a través de 3 pines: Data In (DIN), Clock (CLK) y Load (CS).
3. Los datos que se envían indican qué LEDs deben encenderse en cada fila o columna de la matriz.
4. Comunicaciones SPI:
    - DIN: Pin de entrada de datos. A través de este pin se envían los datos para encender o apagar los LEDs.
    - CLK: Pin de reloj, utilizado para sincronizar el envío de los datos.
    - CS: Pin de selección de chip, que indica al MAX7219 cuándo debe aceptar los datos enviados o en caso de tener mas módulos conectados entre si, CS indica cual de todos los módulos deberá a obedecer a los datos recibidos.

Diagrama de conexiones



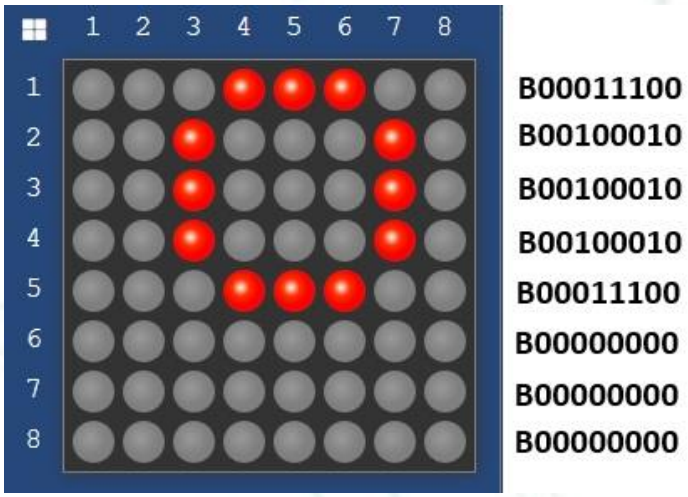


Código de funcionamiento

Tras subir el siguiente programa obtendrás las letras que forman las palabras “unit Electronics” mismas que se encuentran dibujadas a modo de bits en arreglos de tipo byte, por ejemplo la letra o se encuentra en el programa como:

```
byte o[5]={B00011100,B00100010,B00100010,B00100010,B00011100};
```

Donde cada 1 corresponde a un punto de la matriz encendido y cada 0 a un punto apagado de la matriz. De manera que con el arreglo tipo byte para la “o” tendremos el siguiente resultado para la matriz led:



Debido a que las últimas tres filas consisten únicamente en leds apagados, es decir, todos a 0 , en la programación no fueron colocados, por ello el arreglo tiene máximo 5 espacios.

Para que tu puedas programar diferentes caracteres, letras o símbolos sin mucho esfuerzo, te proporcionamos el siguiente link donde podrás obtener los arreglos de bits dibujando las figuras que desees: https://xantorohara.github.io/led-matrix-editor/#

```
//Tenemos que incluir la libreria LedControl

#include <LedControl.h>

#include "LedControl.h"

/*pin 12 está conectado a DataIn ,pin 11 está conectado a LOAD(CS),pin
10 está conectado a CLK */

LedControl lc=LedControl(12,10,11,1);

/*siempre esperamos un tiempo entre las actualizaciones de la pantalla
*/

unsigned long delaytime2=5;

unsigned long delayt = 400;

void setup() {

 /*La MAX72XX se encuentre en modo de ahorro de energía al iniciarla,

 tenemos que hacer una llamada de atención */

 lc.shutdown(0,false);  /* Ajustar el brillo a un valor medio */

 lc.setIntensity(0,8); /* y borrar la pantalla */

 lc.clearDisplay(0);

}

/*Este método mostrará los caracteres de la palabra "unit Electronics",
uno después del otro en la matriz. (necesitas al menos 5x7 leds para
ver los caracteres completos)*/

void escribirenmatriz() { /* estos son los datos para los caracteres */

 byte r[5]={B00010000,B00100000,B00100000,B00010000,B00111110};

 byte u[5]={B00111110,B00000100,B00000010,B00000010,B00111100};

 byte i[5]={B00000000,B00000010,B10111110,B00100010,B00000000};

 byte n[5]={B00011110,B00100000,B00100000,B00010000,B00111110};

 byte o[5]={B00011100,B00100010,B00100010,B00100010,B00011100};

 byte t[5]={B00000010,B00010010,B00010010,B11111100,B00010000};

 byte E[5]={B10000010,B10000010,B10010010,B10010010,B11111110};

 byte l[5]={B00000000,B00000010,B11111110,B00000010,B00000000};

 byte e[5]={B00000000,B00011010,B00101010,B00101010,B00011100};

 byte c[5]={B00000000,B00100010,B00100010,B00100010,B00011100};

 byte s[5]={B00000000,B00101100,B01010010,B01010010,B00100100};

 /*ahora los mostrará uno por uno con un pequeño retraso*/
```

```
lc.setRow(0,0,u[0]);
lc.setRow(0,1,u[1]);
lc.setRow(0,2,u[2]);
lc.setRow(0,3,u[3]);
lc.setRow(0,4,u[4]);
delay(delayt);
lc.setRow(0,0,n[0]);
lc.setRow(0,1,n[1]);
lc.setRow(0,2,n[2]);
lc.setRow(0,3,n[3]);
lc.setRow(0,4,n[4]);
delay(delayt);
lc.setRow(0,0,i[0]);
lc.setRow(0,1,i[1]);
lc.setRow(0,2,i[2]);
lc.setRow(0,3,i[3]);
lc.setRow(0,4,i[4]);
delay(delayt);
lc.setRow(0,0,t[0]);
lc.setRow(0,1,t[1]);
lc.setRow(0,2,t[2]);
lc.setRow(0,3,t[3]);
lc.setRow(0,4,t[4]);
delay(delayt);
lc.setRow(0,0,E[0]);
lc.setRow(0,1,E[1]);
lc.setRow(0,2,E[2]);
lc.setRow(0,3,E[3]);
lc.setRow(0,4,E[4]);
delay(delayt);
lc.setRow(0,0,l[0]);
lc.setRow(0,1,l[1]);
```

```
lc.setRow(0,2,l[2]);
lc.setRow(0,3,l[3]);
lc.setRow(0,4,l[4]);
delay(delayt);
lc.setRow(0,0,e[0]);
lc.setRow(0,1,e[1]);
lc.setRow(0,2,e[2]);
lc.setRow(0,3,e[3]);
lc.setRow(0,4,e[4]);
delay(delayt);
lc.setRow(0,0,c[0]);
lc.setRow(0,1,c[1]);
lc.setRow(0,2,c[2]);
lc.setRow(0,3,c[3]);
lc.setRow(0,4,c[4]);
delay(delayt);
lc.setRow(0,0,t[0]);
lc.setRow(0,1,t[1]);
lc.setRow(0,2,t[2]);
lc.setRow(0,3,t[3]);
lc.setRow(0,4,t[4]);
delay(delayt);
lc.setRow(0,0,r[0]);
lc.setRow(0,1,r[1]);
lc.setRow(0,2,r[2]);
lc.setRow(0,3,r[3]);
lc.setRow(0,4,r[4]);
delay(delayt);
lc.setRow(0,0,o[0]);
lc.setRow(0,1,o[1]);
lc.setRow(0,2,o[2]);
lc.setRow(0,3,o[3]);
```

```
lc.setRow(0,4,o[4]);
delay(delayt);
 lc.setRow(0,0,n[0]);
lc.setRow(0,1,n[1]);
lc.setRow(0,2,n[2]);
lc.setRow(0,3,n[3]);
lc.setRow(0,4,n[4]);
delay(delayt);
lc.setRow(0,0,i[0]);
lc.setRow(0,1,i[1]);
lc.setRow(0,2,i[2]);
lc.setRow(0,3,i[3]);
lc.setRow(0,4,i[4]);
delay(delayt);
lc.setRow(0,0,c[0]);
lc.setRow(0,1,c[1]);
lc.setRow(0,2,c[2]);
lc.setRow(0,3,c[3]);
lc.setRow(0,4,c[4]);
delay(delayt);
lc.setRow(0,0,s[0]);
lc.setRow(0,1,s[1]);
lc.setRow(0,2,s[2]);
lc.setRow(0,3,s[3]);
lc.setRow(0,4,s[4]);
delay(delayt);
lc.setRow(0,0,0);
lc.setRow(0,1,0);
lc.setRow(0,2,0);
lc.setRow(0,3,0);
lc.setRow(0,4,0);
}
```

```
/*Esta función enciende algunos Leds en una fila.El patrón se repetirá en cada fila.El patrón parpadeará junto al número de fila.el número de fila 4 (índice==3) parpadeará 4 veces, etc.*/

void filas() {

 for(int row=0;row<8;row++) {

 delay(delaytime2);

 lc.setRow(0,row,B10100000);

 delay(delaytime2);

 lc.setRow(0,row,(byte)0);

 for(int i=0;i<row;i++) {

 delay(delaytime2);

 lc.setRow(0,row,B10100000);

 delay(delaytime2);

 lc.setRow(0,row,(byte)0);

 }

 }

}

/* Esta función enciende algunos Leds en una columna.El patrón se repetirá en cada columna. El patrón parpadeará junto al número de columna.el número de columna 4 (índice==3) parpadeará 4 veces, etc.*/

void columnas() {

 for(int col=0;col<8;col++) {

 delay(delaytime2);

 lc.setColumn(0,col,B10100000);

 delay(delaytime2);

 lc.setColumn(0,col,(byte)0);

 for(int i=0;i<col;i++) {

 delay(delaytime2);

 lc.setColumn(0,col,B10100000);

 delay(delaytime2);

 lc.setColumn(0,col,(byte)0);

 }

 }

}
```

```
/*Esta función encenderá todos los Led en la matriz.El led parpadeará junto al número de fila.el número de fila 4 (índice==3) parpadeará 4 veces, etc.*/

void single() {

 for(int row=0;row<8;row++) {

 for(int col=0;col<8;col++) {

 delay(delaytime2);

 lc.setLed(0,row,col,true);

 delay(delaytime2);

 for(int i=0;i<col;i++) {

 lc.setLed(0,row,col,false);

 delay(delaytime2);

 lc.setLed(0,row,col,true);

 delay(delaytime2);

 }

 }

 }

}

void loop() {

 escribirenmatriz();

 filas();

 columnas();

 single();

}
```

Lección 23 MPU-5060

En esta lección aprenderás a poner en marcha el sensor MPU-6050 desplegando sus lecturas en el monitor serial del IDE

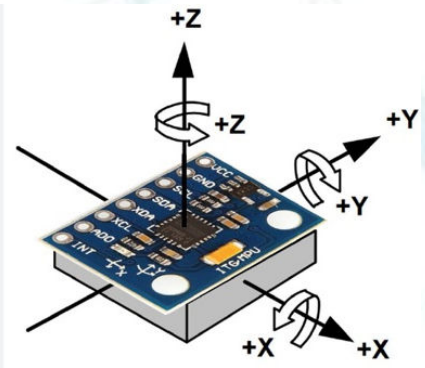
Material Necesario

- Tarjeta Mega 2560 x1
- Módulo MPU-6050
- Cables jumper M-H

Conocimientos previos

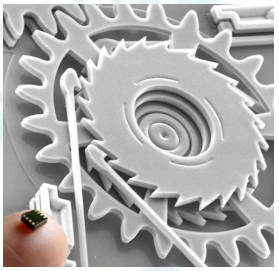
Módulo MPU-6050

El MPU-6050 es un módulo que combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes en un solo chip, lo que lo hace muy útil para proyectos de detección de movimiento, control de estabilización, navegación y más. Este módulo se comunica a través de la interfaz I2C y se usa comúnmente en aplicaciones como drones, robots y sistemas de estabilización.



El principio de funcionamiento interno del giroscopio y acelerómetro del módulo MPU-6050 se basa en la tecnología de fabricación tipo MEMS.

MEMS



MEMS significa Microelectromechanical Systems (Sistemas Microelectromecánicos). Es una tecnología que combina componentes mecánicos, eléctricos y electrónicos a una escala microscópica, generalmente en la escala de micrómetros (millones de metros). Los MEMS son dispositivos miniaturizados que integran estructuras mecánicas y componentes electrónicos en un solo chip, lo que les permite realizar tareas complejas como la detección, el control y la actuación, todo en un espacio muy pequeño.

1.- Acelerómetro MEMS

El acelerómetro del MPU-6050 mide la aceleración en tres ejes (X, Y, Z). Este acelerómetro se basa en principios MEMS, en los que pequeños componentes mecánicos interactúan con señales eléctricas.

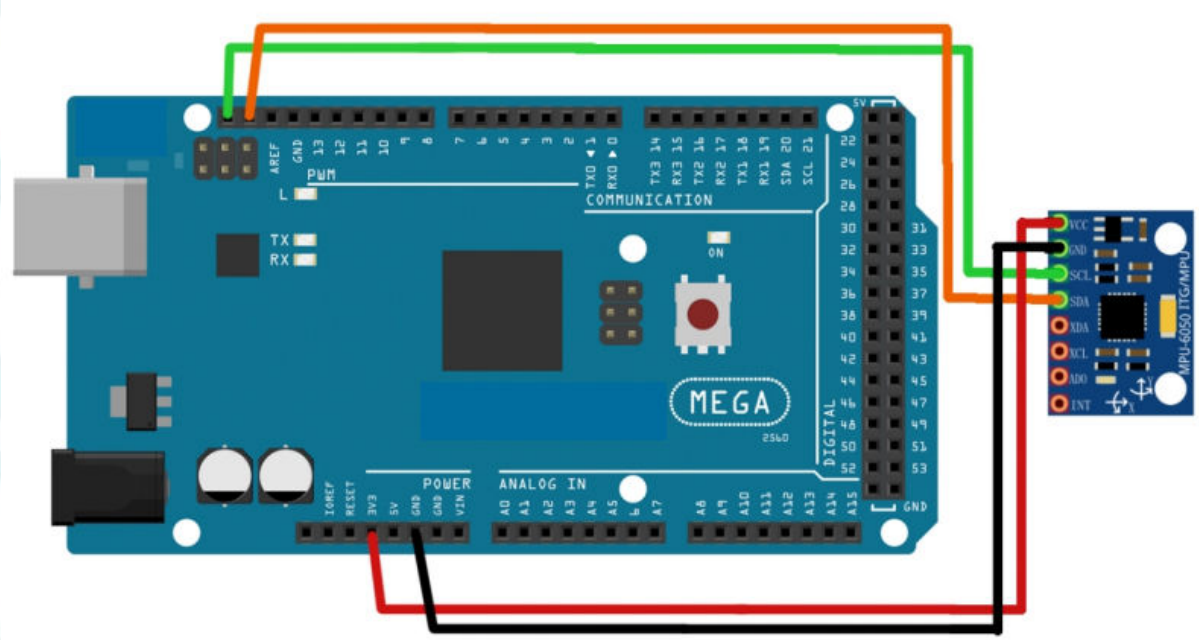
- Dentro del acelerómetro, hay una masa suspendida por un resorte (todo a escala microscópica). Esta masa está conectada a sensores piezoeléctricos que miden las variaciones en la distancia o el desplazamiento de la masa cuando el dispositivo experimenta aceleración.
- Cuando el MPU-6050 experimenta un cambio en su aceleración (por ejemplo, si se mueve o se inclina), la masa suspendida se desplaza, lo que genera una señal eléctrica. Esta señal es procesada y convertida en datos que pueden ser leídos por un microcontrolador.
- La salida del acelerómetro está en función de la aceleración lineal (en unidades de g, que es la aceleración debido a la gravedad, 1g ≈ 9.81 m/s^2). Según la dirección y magnitud del desplazamiento de la masa, el sensor mide cómo cambia la aceleración en cada eje.

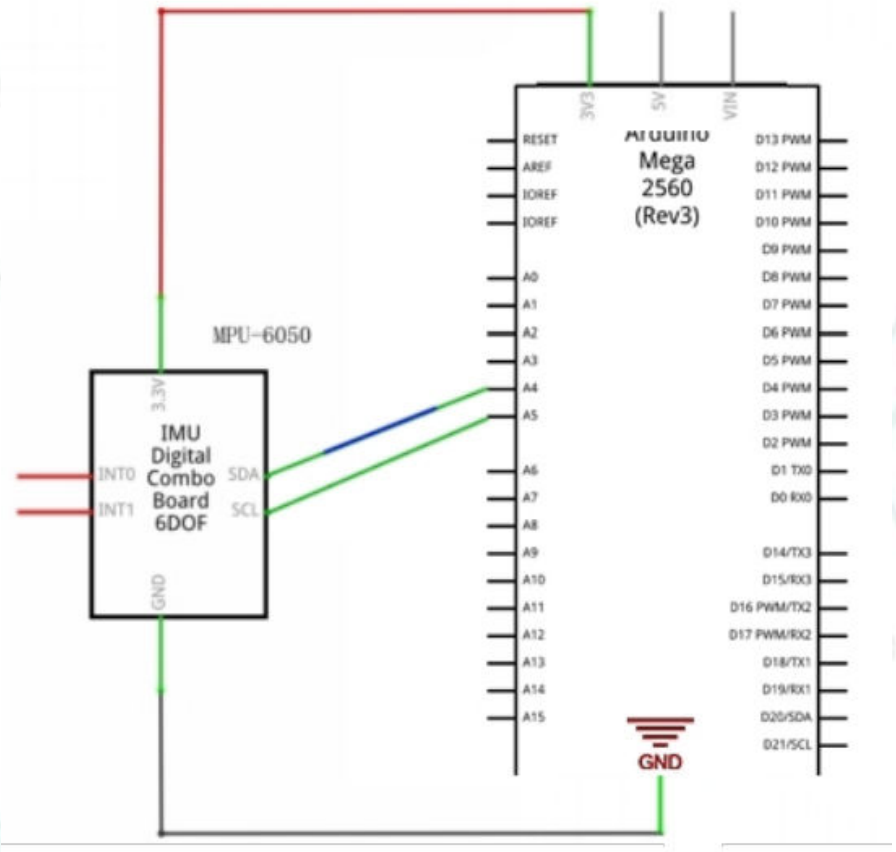
2. Giroscopio MEMS

El giroscopio del MPU-6050 mide la velocidad angular o rotación en tres ejes. Los giroscopios MEMS son también sensores miniaturizados que funcionan utilizando principios de física similares a los del acelerómetro.

- El giroscopio MEMS del MPU-6050 usa un dispositivo de vibración en su interior. Cuando el dispositivo rota, las fuerzas centrífugas afectan la vibración del resonador. Esta alteración es medida por sensores capacitores que detectan los cambios en la vibración.
- Cuando el giroscopio experimenta rotación (como girar un objeto alrededor de su eje), esta vibración cambia, y la señal eléctrica generada se traduce en la velocidad angular (en grados por segundo, °/s).
- La salida del giroscopio indica cuánto se está rotando el dispositivo en cada uno de los tres ejes: X, Y y Z.

Diagrama de conexiones





Código de funcionamiento

Con el siguiente código obtendremos los valores leídos por el MPU-6050 ante diferentes inclinaciones y temperaturas a las que esté expuesto, cabe mencionar que previamente debes instalar la librería llamada "MPU-6050"

Por otra parte observaras en el programa que toda la comunicación se realiza mediante el protocolo I2C, controlando dicho intercambio de datos con la librería Wire.h, misma que no será necesario instalar ya que está cargada por defecto en el IDE.

```
#include <MPU6050.h>
#include <Wire.h>
// MPU-6050 Esquema de ejemplo corto
const int MPU_addr=0x68; // Dirección I2C del MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
 Wire.begin();
 Wire.beginTransmission(MPU_addr);
 Wire.write(0x6B); // Registro PWR_MGMT_1
 Wire.write(0); // Poner a cero (enciende el MPU-6050)
 Wire.endTransmission(true);
 Serial.begin(9600);
}
void loop(){
 Wire.beginTransmission(MPU_addr);
 Wire.write(0x3B); // Empezar con el registro 0x3B
//(ACCEL_XOUT_H)
 Wire.endTransmission(false);
 Wire.requestFrom(MPU_addr,14,true); // Solicita un total de 14
//registros
```

```
 AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) y
//0x3C (ACCEL_XOUT_L)
 AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) y
//0x3E (ACCEL_YOUT_L)
 AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) y
//0x40 (ACCEL_ZOUT_L)
 Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) y 0x42
//(TEMP_OUT_L)
 GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) y
//0x44 (GYRO_XOUT_L)
 GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) y
//0x46 (GYRO_YOUT_L)
 GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) y
//0x48 (GYRO_ZOUT_L)
 Serial.print("AcX = "); Serial.print(AcX);
 Serial.print(" | AcY = "); Serial.print(AcY);
 Serial.print(" | AcZ = "); Serial.print(AcZ);
 Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53); // Ecuación
//para la temperatura en grados C de la hoja de datos
 Serial.print(" | GyX = "); Serial.print(GyX);
 Serial.print(" | GyY = "); Serial.print(GyY);
 Serial.print(" | GyZ = "); Serial.println(GyZ);
 delay(1000);

}
```

Lección 24 Sensor PIR

En esta lección, aprenderás a utilizar un detector de movimiento PIR a través del Mega 2560.

Material necesario

- Resistencias 10k, 22k, 100 todas a 1/4w
- Transistor BC547
- Diodo 1N4007
- LED
- 5.- Relevador SPDT 5V
- 6.- Módulo de fuente de alimentacion para protoboard
- 7.- Eliminador 9V 1A
- 8.- Tarjeta Mega 2560
- 9.- Sensor de movimiento PIR HC-SR501
- 10.- Cables jumper Hembra-Macho
- 11.- Protoboard

Conocimientos previos

Sensor PIR

Un sensor PIR (Passive InfraRed) es un dispositivo electrónico que detecta la radiación infrarroja emitida por objetos que tienen temperatura superior a la del entorno, como seres humanos o animales.

La "radiación infrarroja" es una forma de energía que se irradia en longitudes de onda que no son visibles para el ojo humano, pero que los sensores PIR pueden detectar.

El sensor PIR funciona bajo el efecto termoeléctrico. Este efecto se refiere a la capacidad de un material para generar una señal eléctrica cuando hay una diferencia de temperatura en sus extremos. Los sensores PIR generalmente utilizan un material semiconductor que responde a las variaciones en la radiación infrarroja en su entorno.

Cuando una persona o animal se mueve frente al sensor, la radiación infrarroja que emite el cuerpo es detectada por el sensor, que puede enviar una señal de activación a otro sistema, como una alarma o una luz.

Son ampliamente utilizados en sistemas de seguridad, iluminación automática y automatización del hogar.

El sensor PIR tiene la siguiente distribución de pines:

Podemos configurar al sensor PIR con la ayuda del par de potenciómetros y el jumper selector cuyas funciones se enlistan a continuación:

1.- Selector de modo: nos permite alternar entre el modo de funcionamiento continuo y el modo de repetición. En el modo continuo, si el sensor detecta movimiento de forma continua, mantendrá una señal constante. En el modo de repetición, el sensor se activará al detectar movimiento y luego regresará a su estado normal; si detecta nuevamente movimiento, se activará otra vez y completará un nuevo ciclo, pero no funcionará de manera continua aunque el movimiento se repita. El primer modo es ideal para utilizarlo como detector de movimiento en zonas donde se necesite iluminación constante mientras haya presencia.

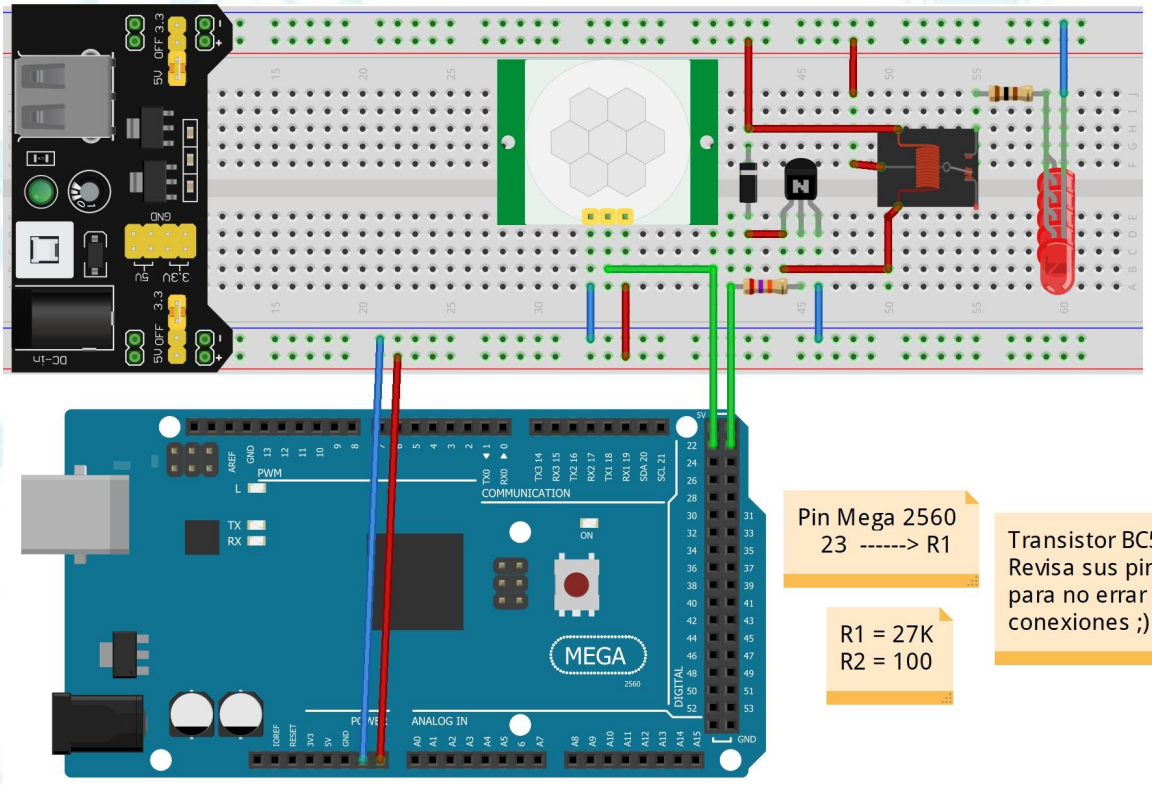
2.- Potenciómetro de sensibilidad: permite incrementar o reducir la distancia a la que se activará y/o la cantidad de movimiento requerida para que el sensor se active, por ejemplo, para diferenciar a una persona de una mascota.

3.- Potenciómetro del temporizador: permite ajustar el tiempo durante el cual el sensor permanecerá activado después de detectar presencia, con un rango que varía aproximadamente entre 3 segundos y 5 minutos.

Si el sensor está en modo continuo, el tiempo de activación será como mínimo el ajustado, sin límite máximo, siempre y cuando el sensor siga detectando presencia mientras esté activado.

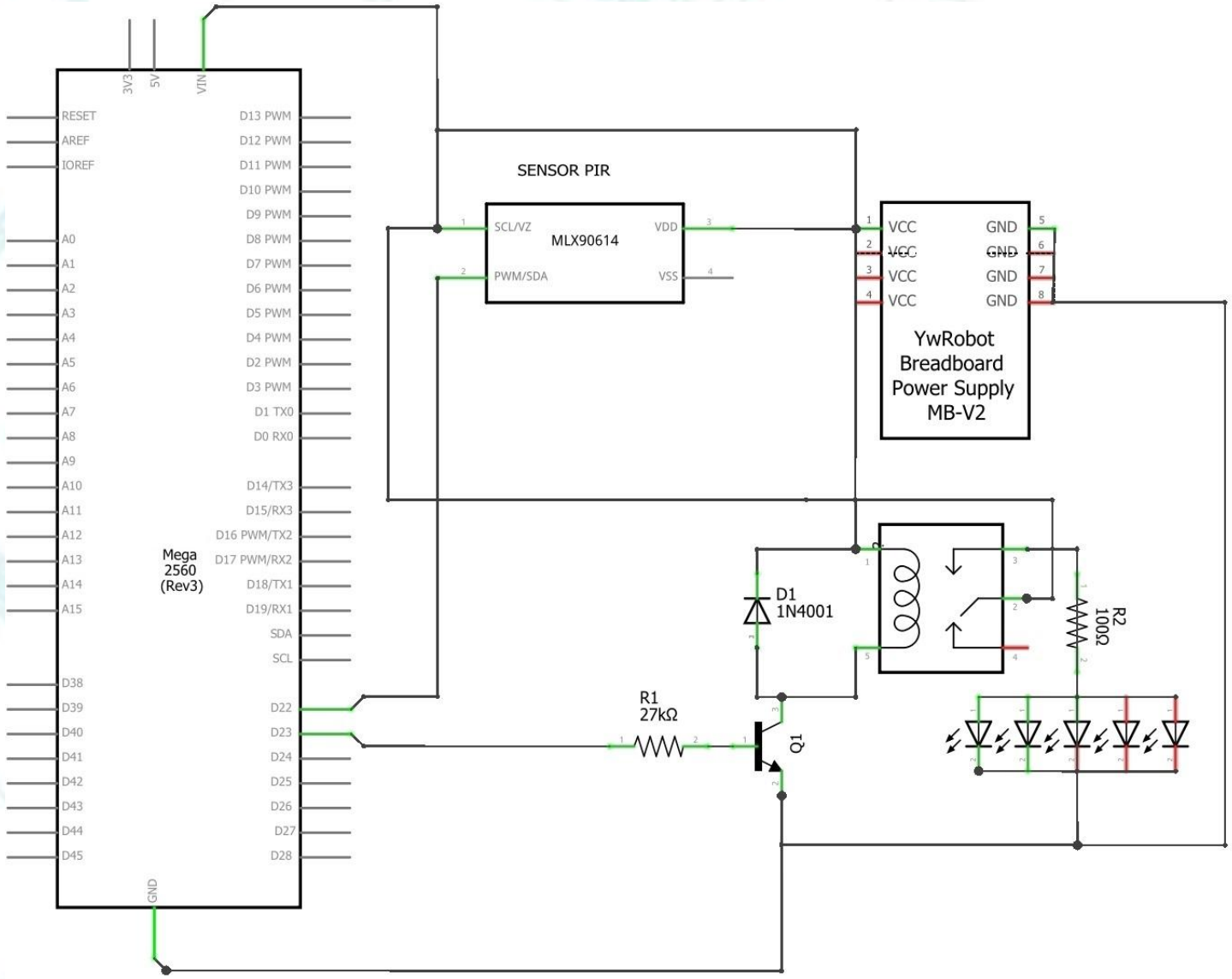
Diagrama de conexiones

Es muy importante que conecte correctamente el diodo 1N4007 ya que este va a impedir que la bobina destruya al transistor debido a que esta genera picos de tensión muy elevados al conmutar, dichos picos son redirigidos por el diodo brindando protección.



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

Como ya te habrás dado cuenta el diagrama y en este caso el código es exactamente el que abordamos en la lección 21 con la única diferencia de que en el botón fue reemplazado por la salida del sensor PIR, esto con la finalidad de mostrar las diferentes entrada de las que puede provenir una señal, ya sea de forma manual pulsando el botón o mediante la presencia que es detectada por un sensor PIR, por lo que te invitamos a conectar otro tipo de sensores de salida digital presentes en estas lecciones a modo de que experimentes y combines recursos que puedan responder a diferentes aplicaciones y necesidades.

```
#define PIR 22// Conectamos la salida del sensor PIR al pin digital 22 del mega 2560

#define transistor 23 // Conectamos la resistencia de base al pin digital 23 del mega 2560

int estado = LOW;

void setup() {

pinMode(PIR, INPUT);            // declaramos al PIR como entrada

pinMode(transistor, OUTPUT);    // declaramos al transistor como salida

digitalWrite(transistor, HIGH); // El iniciar el programa el transistor estará en estado alto

}

void loop() {

while(digitalRead(PIR) == LOW){}//Mientras que el PIR detecte no hace nada

estado = digitalRead(transistor); //Estado guarda el comportamiento del transistor

digitalWrite(transistor, !estado);//Se escribe en el transistor el estado opuesto que tenía

while(digitalRead(PIR) == HIGH){}// Mientras que el PIR detecte presencia no hace nada  }
```

Lección 25 Detector de Lluvia

En esta lección aprenderás a construir y programar un sencillo control a través de un motor a pasos de una ventana que se cierra cuando se detecta lluvia sobre el sensor de agua. Estas acciones serán desplegadas en forma de mensajes en una LCD 16X2.

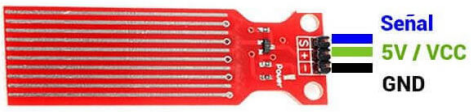
Materiales necesarios

- Tarjeta Mega 2560
- Módulo fuente de alimentación para protoboard
- Sensor de Humedad del Suelo FC-28 Higrómetro
- Motor a pasos 28BYJ-48
- Driver para motor a pasos ULN2003
- LCD 16X2
- I2C Interfaz LCD1602 PCF8574
- Cables jumper Macho-Macho
- Cables jumper Macho-Hembra
- Protoboard

Conocimientos previos

Sensor de nivel de líquidos

El sensor de nivel de líquidos es un dispositivo que entrega una señal analogica de acuerdo con el cambio en la conductividad eléctrica de las pistas impresas en su placa cuando entran en contacto con el agua común, que, según las sales disueltas, tendrá más o menos conductividad, dicho cambio de la resistencia eléctrica provocará una corriente de base que para un transistor BJT tipo NPN significa una corriente proporcional en el colector, lo que a su vez implica una tensión capaz de ser leída por el ADC de un microcontrolador.



Valor Analógico	Nivel de Líquido
0	0%
256	25%
512	50%
1023	100%

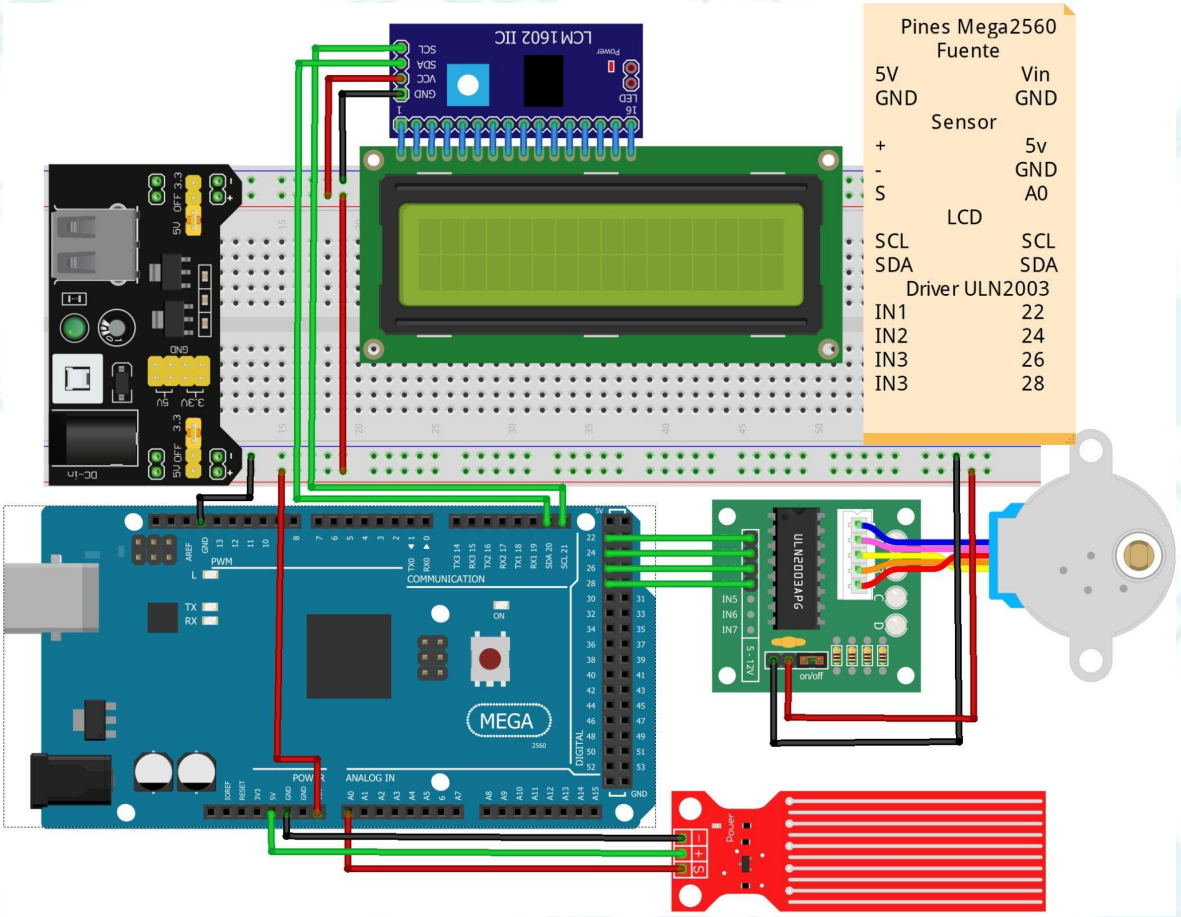
Resolución
10 bit
(0 a 1023)

Podemos caracterizar los diferentes niveles de agua que pueda detectar el sensor para un ADC con una resolución de 10 bits, es decir, de 1023 pasos, tal como a continuación se observa en la imagen.

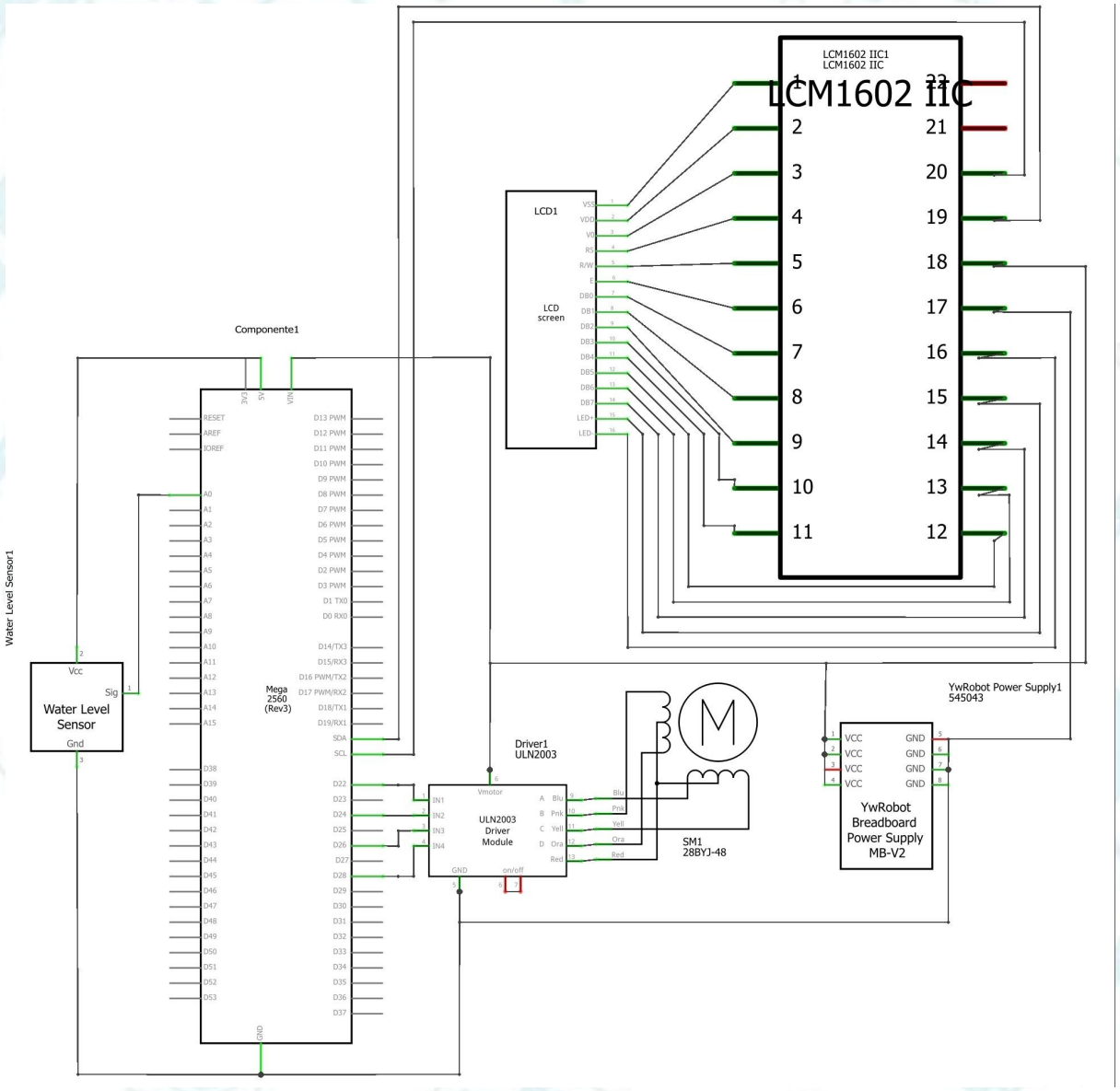
Diagrama de conexión

Sigue exactamente el diagrama como se muestra en la imagen, sobre todo teniendo en cuenta que jamás deberías conectar la alimentación del driver del motor a pasos directamente a la tarjeta Mega 2560 ya que consume demasiada corriente y corres el riesgo de quemar tu placa.

Presta especial atención a que el módulo de interfaz I2C con la LCD nos evita tener que conectar los 16 pines de la pantalla al Mega, ocupando sólo dos pines de comunicación (SCL y SDA) además de los pines de alimentación.



En caso de no tener imagen en la pantalla LCD, prueba presionando el botón del reset del Mega 2560 o moviendo el potenciómetro del módulo interfaz I2C para variar el contraste hasta que aparezcan los caractere



Código de funcionamiento

Una vez que cargues el código te retamos a modificarlo añadiendo un buzzer que se active cuando la venta se cierra o se abre ;)

```
//Incluimos las librerías para la comunicación I2C y para el control de la LCD

#include <Wire.h>

#include <LiquidCrystal_I2C.h>

// Definimos los pines que van a usar el driver ULN2003 y la salida del sensor de agua

#define sensoragua A0

#define motorPin1  22 // IN1

#define motorPin2  24 // IN2

#define motorPin3  26 // IN3

#define motorPin4  28 // IN4

// Definir el arreglo de secuencia de pasos para el motor 28BYJ-48

int pasos[8][4] = {

  {1, 0, 0, 1},  // Paso 1

  {1, 0, 0, 0},  // Paso 2

  {1, 1, 0, 0},  // Paso 3

  {0, 1, 0, 0},  // Paso 4

  {0, 1, 1, 0},  // Paso 5

  {0, 0, 1, 0},  // Paso 6

  {0, 0, 1, 1},  // Paso 7

  {1, 0, 1, 0}   // Paso 8

};

// Declaramos las variables que nos ayudarán en la lógica de programación

int  nivelagua = 0, op = 0, og = 0;
```

```
// Esta variable determinará la sensibilidad del sensor de agua si incrementas su valor, entonces se necesitará de más agua para activar el sistema

int umbral = 100;

// Colocamos la dirección I2C de la interfaz que es 0x27 además de declara que es un LCD DE 16X2

LiquidCrystal_I2C lcd(0x27,16,2);

void setup()

{

  //Inicializamos la LCD y encendemos la luz de fondo de la pantalla

  lcd.init();

  lcd.init();

  lcd.backlight();

// Declaramos salidas y entradas

  pinMode(sensoragua, INPUT);

   pinMode(motorPin1, OUTPUT);

  pinMode(motorPin2, OUTPUT);

  pinMode(motorPin3, OUTPUT);

  pinMode(motorPin4, OUTPUT);

}

void loop()

{

  //Realizamos un promedio de 150 lecturas del sensor de agua

  for(int i = 0; i <150; i++){

   nivelagua += analogRead(sensoragua);

   delay(10);

   }

int lectura = (nivelagua / 150);

  // Comparamos la lectura del sensor con el valor umbral,Si hay agua, es decir, si la lectura es mayor al umbral entonces, se muestra en la LCD el mensaje "LLUVIA DETECTADA" y el motor a pasos gira

  if(lectura>umbral)

  {

  lcd.clear();

  lcd.setCursor(0,0);
```

```
  lcd.print("LLUVIA DETECTADA ");

  lcd.setCursor(0,1);

  lcd.print("Ventana cerrada");

//Con este bucle while logramos que el motor gire una vuelta y se detenga

while(op == 1){

  for (int i = 0; i < 512; i++) {// 512 pasos para una vuelta completa

    for (int j = 7; j >= 0; j--) {

      digitalWrite(motorPin1, pasos[j][0]);

      digitalWrite(motorPin2, pasos[j][1]);

      digitalWrite(motorPin3, pasos[j][2]);

      digitalWrite(motorPin4, pasos[j][3]);

      delayMicroseconds(1000);//Determina la velocidad de giro

    }

// Aquí apagamos el motor porque sin poner a LOW los pines el motor podría quemar a la fuente

     if(i ==511) {

      op = 0;

      digitalWrite(motorPin1, LOW);

      digitalWrite(motorPin2, LOW);

      digitalWrite(motorPin3, LOW);

      digitalWrite(motorPin4, LOW);

      }

  }

   }

  //Leemos el sensor para que en el momento que se detecte que ya no hay agua salimos del if

for(int i = 0; i <150; i++){

   nivelagua += analogRead(sensoragua);

   delay(10);

   }

int lectura = (nivelagua / 150);

op = 0;

og =1;
```

```
  }
// SI el sensor está seco entonces se abre la ventana y se manda el mensaje en la LCD
  else{
     lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Ventana abierta");
   nivelagua = analogRead(sensoragua);
lectura = constrain( nivelagua, 0,1023);
//Giramos el motor en sentido inverso y cuando da la vuelta se apaga
while(og == 1){
    for (int i = 0; i < 512; i++) {   // 512 pasos para una vuelta completa
    for (int j = 0; j < 8; j++) {
      digitalWrite(motorPin1, pasos[j][0]);
      digitalWrite(motorPin2, pasos[j][1]);
      digitalWrite(motorPin3, pasos[j][2]);
      digitalWrite(motorPin4, pasos[j][3]);
      delayMicroseconds(1000);  // Retardo de 2000 microsegundos (2ms)
    }
     if(i ==511) {
      og = 0;
      digitalWrite(motorPin1, LOW);
      digitalWrite(motorPin2, LOW);
      digitalWrite(motorPin3, LOW);
      digitalWrite(motorPin4, LOW);
      }
  }
  }
op = 1; // Esta variable junto con og nos ayudan a detener al motor
  }
}
```

Lección 26 Checador basado en el módulo RTC DS3231

En esta lección aprenderás a implementar un checador básico usando el módulo RTC DS3132 para un conteo fiable de la hora y la fecha, además de un acceso por RFID similar al visto en la lección 26, todo ello empleando el Mega 2560.

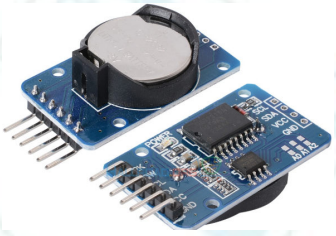
Materiales necesarios

- Tarjeta Mega 2560
- Módulo RTC DS3231
- Buzzer activo
- Modulo RFID MFRC522
- Servomotor SG-90
- LCD 16X2
- I2C Interfaz LCD1602 PCF8574
- Módulo fuente de alimentación para protoboard
- Eliminador 9v 1A
- Protoboard
- Cables jumper Macho-Macho
- Cables jumper Macho-Hembra
- Batería de botón CR2032

Conocimientos previos

Módulo RTC DS3231

El módulo RTC (Real Time Clock) DS3231 es un dispositivo de registro de tiempo que cuenta con una batería integrada, para que el reloj continúe calculando la hora incluso cuando está desenchufado.

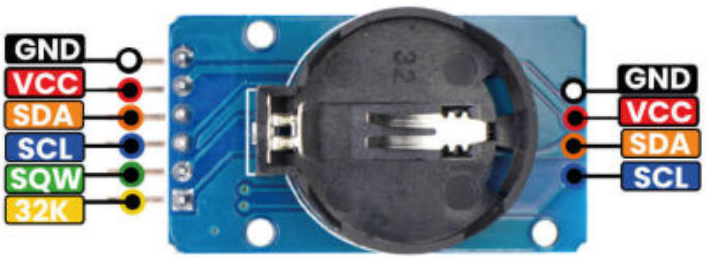


El DS3231 es un reloj en Tiempo Real de alta precisión que puede contar horas, minutos y segundos, así como también información sobre el día, mes y año. Además, cuenta con una compensación automática para años bisiestos y para meses con menos de 31 días.

El DS3231 es conocido por su **alta precisión** en comparación con otros módulos RTC. Tiene un **oscilador de temperatura compensada (TCXO)**, lo que significa que tiene un rendimiento más estable a lo largo de diferentes rangos de temperatura. Su precisión es de aproximadamente **±1 minuto por año**, lo que lo convierte en uno de los RTC más precisos disponibles.

El módulo puede funcionar con 3.3 o 5 V, lo que lo hace adecuado para muchas plataformas de desarrollo o microcontroladores. La entrada de la batería es de 3V y una típica batería CR2032 de 3V puede alimentar el módulo y mantener la información durante más de un año.

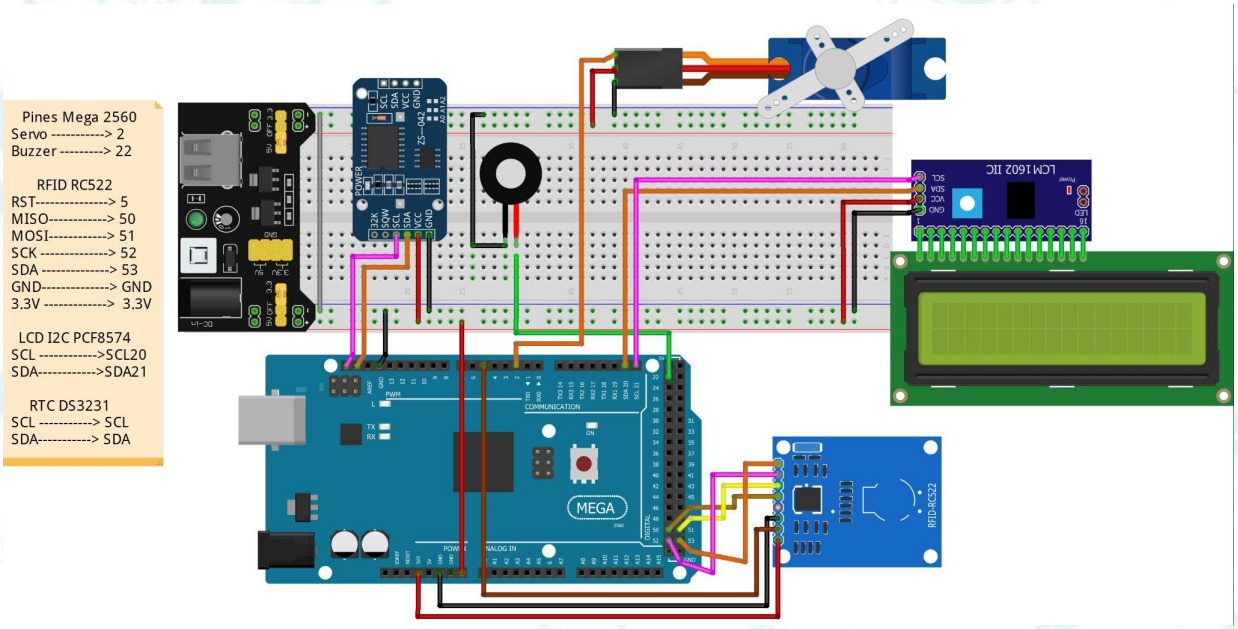
El módulo utiliza el Protocolo de Comunicación I2C, el cual facilita la conexión al Mega 2560.



GND:Tierra
VCC: 3.3V a 5V
SDA: Datos
SCL: Reloj
SQW: 1 Hz, 1 Khz, 4 Khz, 8 Khz
32K: Oscilador 32.768 Khz

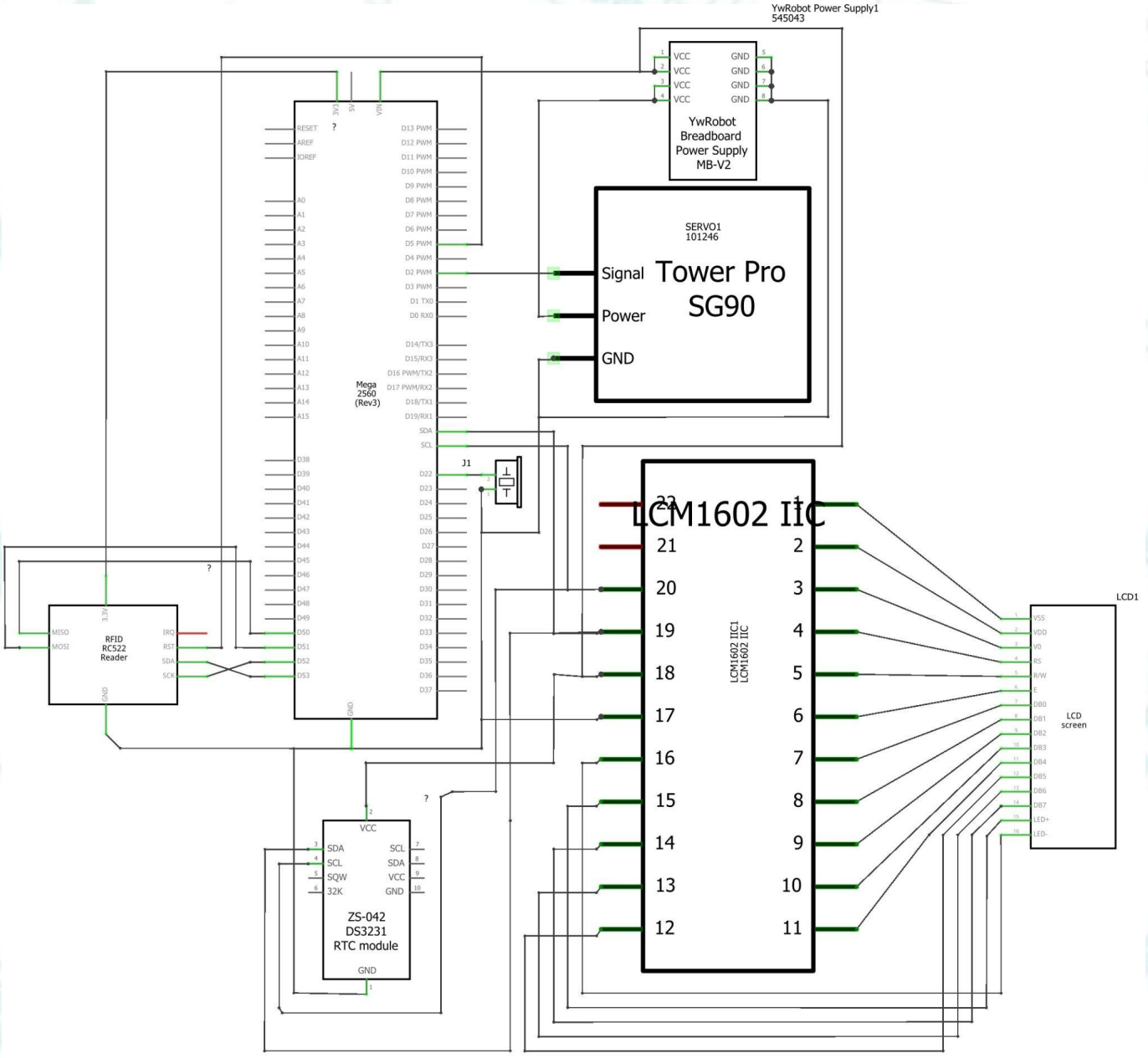
Diagrama de conexiones

En el siguiente diagrama la tarjeta Mega 2560 es alimentado desde el módulo de fuente para protoboard hacia su pin Vin, pero si desconectas el cable perfectamente puedes alimentar al mega 2560 desde tu PC mediante el cable USB y visualizar la información en el monitor serial si lo deseas



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

Es muy importante que la primera vez que cargues el código lo hagas con la siguiente línea sin comentar

```
rtc.adjust(DateTime(__DATE__, __TIME__));
```

Ya que esta se encarga de obtener la fecha y hora desde tu PC para no tener que configurarla manualmente.

Después, vuelve a cargar el código con dicha línea comentada para que el RTC pueda funcionar correctamente

```
//rtc.adjust(DateTime(__DATE__, __TIME__));
```

Esto para que ahora el RTC funcione independientemente con la fecha y hora que ya almacenó en su memoria y que si tiene la batería conectada no perderá aunque se desconecte la alimentación por los pines VCC y GND.

Tras esa segunda carga del programa el sistema deberá funcionar correctamente.

```
#include <Wire.h>   // incluye libreria para interfaz I2C

#include <RTClib.h>   // incluye libreria para el manejo del modulo RTC

#include <LiquidCrystal_I2C.h>

#include <SPI.h>

#include <MFRC522.h>

#include <Servo.h>

#define RST_PIN 5 // Pin del RFID-RC522 a Pin 5

#define SS_PIN 53 // También llamado SDA en el MFRC522

#define buzzer 22

Servo sg90;

MFRC522 mfrc522(SS_PIN, RST_PIN); // Creando instancia para RFID

byte i; //variable , elemento de matriz de lectura de datos por el
lector
```

```
byte datoUID[4]; //matriz para guardar valores arrojados por el detector Valor hexadecimal al usar el llave de la persona1 <previamente comprobado>, puede variar

byte persona1[4]={0x53,0x8F,0x32,0xBD};

//Valor hexadecimal al usar la tarjeta <previamente comprobado>, puede variar

byte tag[4]={0x33,0xF0,0x20,0x0D};



RTC_DS3231 rtc;     // crea objeto del tipo RTC_DS3231

LiquidCrystal_I2C lcd(0x27,20,4);

void setup () {

 Serial.begin(9600);    // inicializa comunicacion serie a 9600 bps

  lcd.init();

  lcd.init();

  lcd.backlight();

 if (! rtc.begin()) {       // si falla la inicialización del módulo

Serial.println("Modulo RTC no encontrado !");   // muestra mensaje de error

while (1); // bucle infinito que detiene ejecución del programa

 }

// función que adquiere la fecha y hora desde la computadora a la que el Mega este conectado. Carga por primera vez el código con esta línea sin comentar para que el RTC adquiera la hora y fecha desde tu PC. Después vuelve a cargar el código con esta línea comentada para que el RTC funcione normalmente rtc.adjust(DateTime(__DATE__, __TIME__));

Serial.begin(9600); // Inicializa comunicación serial

SPI.begin(); // Iniciación de comunicación por bus SPI

sg90.attach(2); //Pin para la señal al servo-Pin 3

sg90.write(0); //Siempre que se inicie el programa el servo tendrá un valor de 0°

mfrc522.PCD_Init(); // inicialización de MFRC522

Serial.println(F("Coloque tarjeta para realizar escaner")); //mensaje para usuario

pinMode(buzzer, OUTPUT);

}

void loop () {

   delay(1000);        // demora de 1 segundo
```

```
DateTime fecha = rtc.now();// funcion que devuelve fecha y horario en formato, DateTime y asigna a variable fecha

  lcd.clear();

  lcd.setCursor(0,0);

  lcd.print(fecha.day());

 Serial.print(fecha.day()); //función que obtiene el dia de la fecha completa

lcd.setCursor(2,0);

lcd.print("/");

Serial.print("/");               // caracter barra como separador

lcd.setCursor(3,0);

lcd.print(fecha.month());

Serial.print(fecha.month());// función que obtiene el mes de la fecha completa

lcd.setCursor(5,0);

lcd.print("/");

Serial.print("/");

lcd.setCursor(6,0);

lcd.print(fecha.year());

Serial.print(fecha.year());// función que obtiene el año de la fecha completa

Serial.print(" ");

lcd.setCursor(0,1);

lcd.print(fecha.hour());

Serial.print(fecha.hour());       // función que obtiene la hora de la fecha completa

lcd.setCursor(2,1);

lcd.print(":");

Serial.print(":");

lcd.setCursor(3,1);

lcd.print(fecha.minute());

Serial.print(fecha.minute()); // función que obtiene los minutos de la fecha completa

lcd.setCursor(5,1);

lcd.print(":");
```

```
Serial.print(":");

lcd.setCursor(6,1);

lcd.print(fecha.second());

Serial.println(fecha.second()); // función que obtiene los segundos de
la fecha completa

if ( ! mfrc522.PICC_IsNewCardPresent())

return;

// Lee el valor del ID presentado

if ( ! mfrc522.PICC_ReadCardSerial())

return;

Serial.println("UID:");

//Se guarda los datos del ID

for (byte i = 0; i < mfrc522.uid.size; i++) {

//limitando la lectura de los valores en la matriz hexadecimal a 10
dígitos

if (mfrc522.uid.uidByte[i] < 0x10) {

Serial.print("0");

} else {

Serial.print("");

}

//imprime en Monitor Serial, el valor leído por el detector

Serial.print(mfrc522.uid.uidByte[i], HEX);

datoUID[i]= mfrc522.uid.uidByte[i]; //Se guardan los valores en la
variable dato UID

}

mfrc522.PICC_HaltA();

if (datoUID[i] == persona1[i]){ //Si la lectura corresponde a los
valores de la matriz persona1

lcd.setCursor(0,0);

lcd.print("Persona #1");

lcd.setCursor(8,1);

lcd.print("<--CHECK");

digitalWrite(buzzer, HIGH);

delay(50);
```

```
digitalWrite(buzzer, LOW);
delay(50);
digitalWrite(buzzer, HIGH);
delay(50);
digitalWrite(buzzer, LOW);
sg90.write(180); //se abre la puerta
delay(5000); // La puerta permanece abierta durante 5 segundos
sg90.write(0);
}
else{ // Si no se reconoce la tarjeta sonará una alerta en el buzzer
digitalWrite(buzzer, HIGH);
delay(350);
digitalWrite(buzzer, LOW);
delay(50);
digitalWrite(buzzer, HIGH);
delay(350);
digitalWrite(buzzer, LOW);
}
}
```

Lección 27 Sensor de sonido

En esta lección aprenderás a construir un sencillo sonómetro cuyas lecturas podrás visualizar en el serial plotter del IDE además de un interruptor por aplausos con el que podrás controlar un relevador mediante el sensor de sonido KY-037 y el Mega 2560.

Materiales necesarios

- Resistencias 10k, 22k, 100 todas a 1/4w
- Transistor BC547
- Diodo 1N4007
- LED
- Relevador SPDT 5V
- Módulo de fuente de alimentacion para protoboard
- Eliminador 9V 1A
- Tarjeta Mega 2560
- Protoboard
- Sensor de sonido KY-037

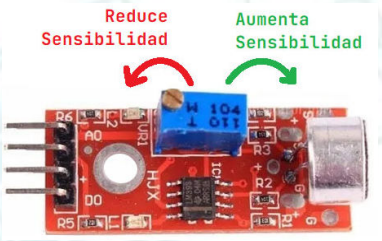
Conocimientos previos

El KY-037 es un sensor de sonido que puede detectar sonidos o niveles de ruido en su entorno. Es un dispositivo comúnmente utilizado con microcontroladores como el Mega 2560, el cual tiene dos partes principales.

1.- **Micrófono incorporado:** El sensor tiene un micrófono que capta las ondas sonoras que se generan en su entorno. Estas ondas sonoras son transformadas en señales eléctricas por el micrófono.

2.- **Amplificador de señal:** La señal del micrófono es muy débil, por lo que pasa por un amplificador operacional LM393 para aumentar su intensidad, lo que facilita que el sensor detecte el sonido de manera más eficiente.

Dicha amplificación estará relacionada con el potenciómetro de sensibilidad que viene integrado en el sensor, para lo cual, según la aplicación, primero deberás girar la perilla hasta que obtengas la activación del sensor, que notarás cuando se encienda el led a la intensidad sonora que desees.



Salida digital y analógica

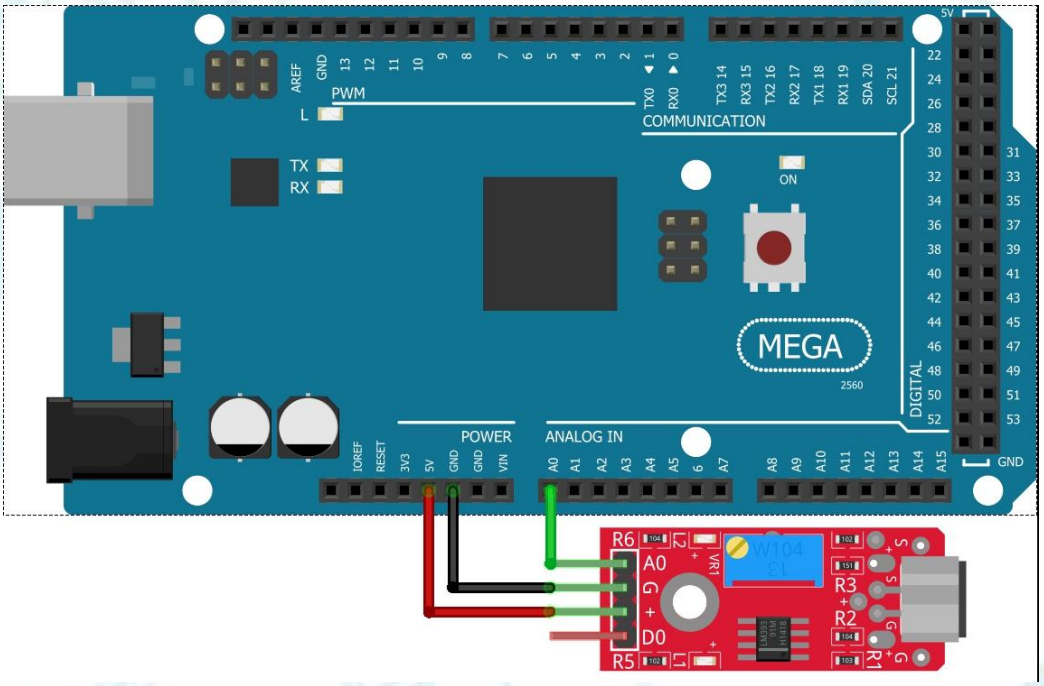
Salida digital: El sensor tiene una salida digital (pin DO) que activa un "alto" (1) o "bajo" (0) dependiendo de si el sonido detectado supera un umbral determinado. Este umbral se ajusta a través de un potenciómetro en el módulo, lo que te permite determinar la sensibilidad del sensor. Si el sonido detectado es lo suficientemente fuerte como para superar ese umbral, la salida digital cambiará.

Salida analógica: También tiene una salida analógica (pin AO) que da una señal proporcional a la intensidad del sonido. Esto permite obtener una lectura más precisa de la amplitud del sonido, en lugar de solo detectar su presencia.

Puedes utilizar este sensor en proyectos de detección de sonido, control de volumen, alarmas de sonido, y otros sistemas que necesiten medir o reaccionar al sonido.

Diagrama de conexión

Para este pequeño prototipo haremos uso de la salida analogica del sensor de sonido (A0)



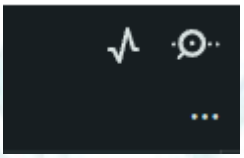
Código de funcionamiento

Como podrás observar, en este programa el sonómetro consiste en el despliegue de las lecturas en el serial plotter a traves de la sentencia Serial.println(sensor);, por ello para poder tener un instrumento de medición sonora tendrás que interpretar los valores que te entrega el ADC de la tarjeta Mega 2560 respecto a una escala

```
#define sonido A0

float sensor  = 0.0;

void setup(){

  pinMode(sonido, INPUT);

  Serial.begin(9600);

}
```

```
void loop(){

sensor = analogRead(sonido);

Serial.println(sensor);

delay(300);

}
```

Para abrir el serial plotter deberás dar click en el siguiente icono ubicado en la parte superior derecha del IDE mientras la tarjeta Mega 2560 permanece conectado a tu PC

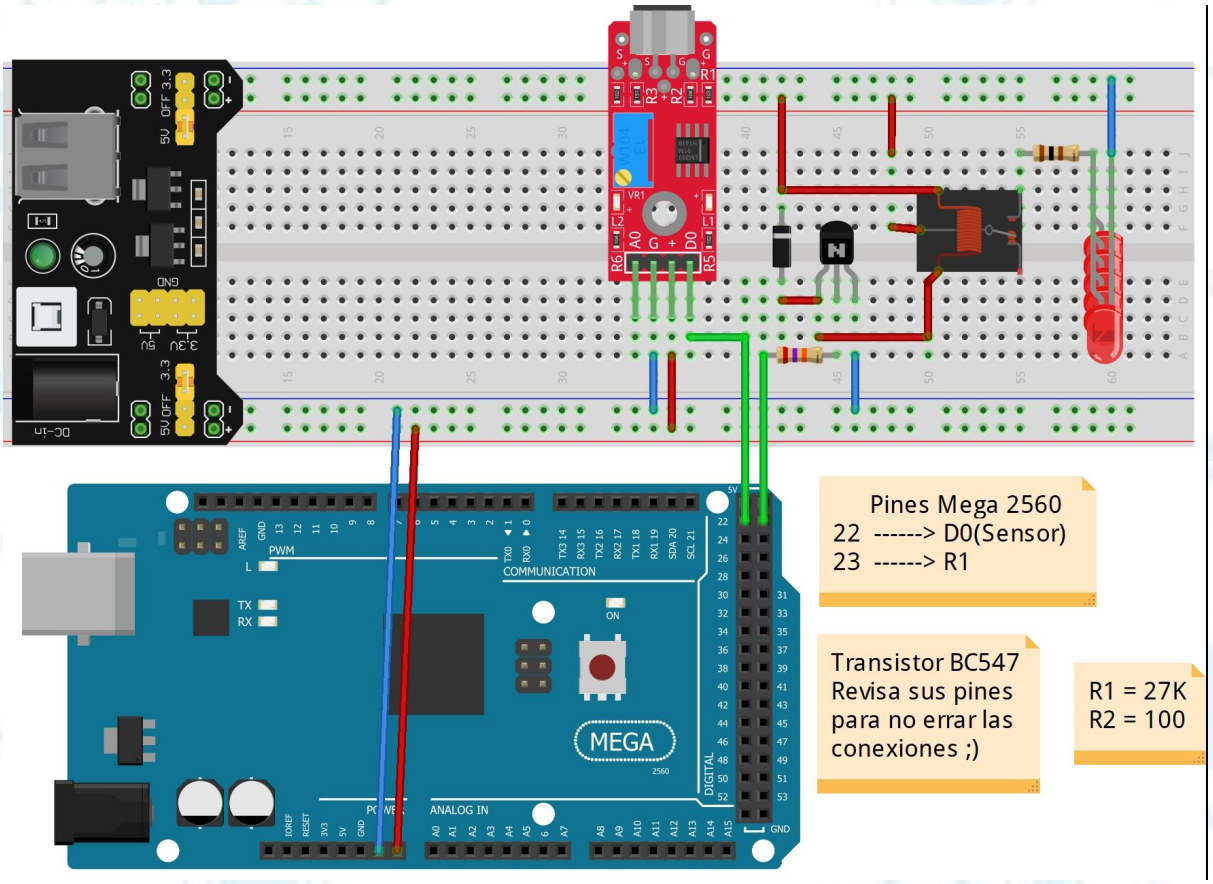


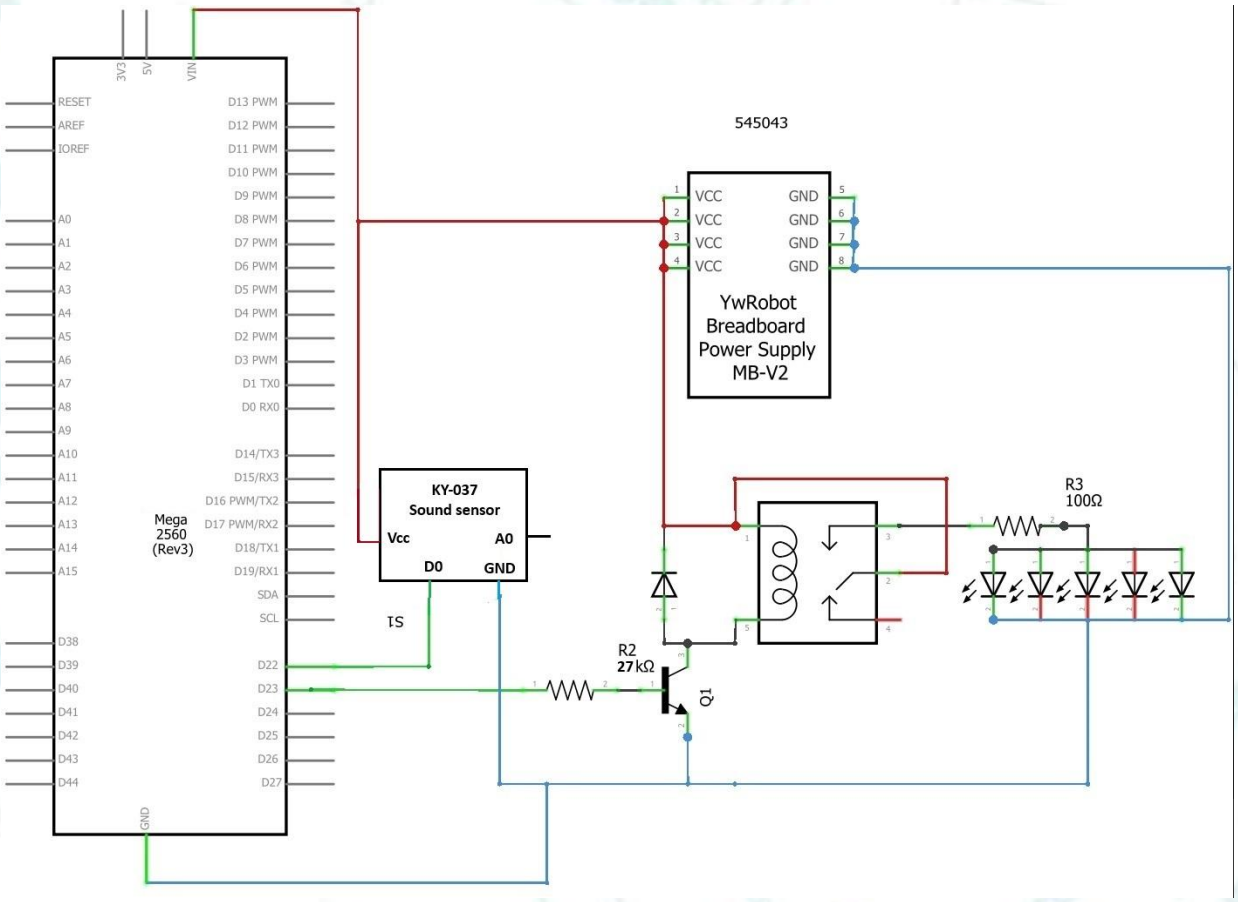
Y obtendrás una ventana como la siguiente:

Si tienes problemas para obtener el serial plotter, revisa que puerto COM pertenece al Mega.

Diagrama de conexión del interruptor por aplausos

En este segundo diagrama hacemos uso de pin digital del sensor de sonido (D0) y del ya conocido diagrama de transistor y relevador para activar cargas elevadas que ya vimos en la lección 21 y 24, solo que esta vez la señal de activación vendrá del sensor que genera un pulso alto cuando detecte un aplauso.




3V3
5V
VIN
RESET
AREF
IOREF
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D38
D39
D40
D41
D42
D43
D44
GND
D13 PWM
D12 PWM
D11 PWM
D10 PWM
D9 PWM
D8 PWM
D7 PWM
D6 PWM
D5 PWM
D4 PWM
D2 PWM
D3 PWM
D1 TX0
D0 RX0
D14/TX3
D15/RX3
D16 PWM/TX2
D17 PWM/RX2
D18/TX1
D19/RX1
SDA
SCL
D22
D23
D24
D25
D26
D27
Mega
2560
(Rev3)
545043
VCC
VCC
VCC
VCC
GND
GND
GND
GND
YwRobot
Breadboard
Power Supply
MB-V2
KY-037
Sound sensor
Vcc
A0
D0
GND
R2
27 kΩ
Q1
R3
100Ω

Código de funcionamiento del interruptor por aplausos

De igual manera que en las lecciones 21 y 24 el código consiste en el cambio de estado del transistor cuando recibe pulsos de activación, que naturalmente en esta lección proviene del sensor de sonido.

```
#define sensorsonido 22        // Conectamos el pin digital del sensor
(D0) al pin digital 22 del mega

#define transistor 23 // Conectamos la resistencia de base al pin
digital 23 de la tarjeta Mega 2560

int estado = LOW;

void setup() {

pinMode(sensorsonido, INPUT);       // declaramos al PIN digital del
sensor como entrada

pinMode(transistor, OUTPUT);    // declaramos al transistor como salida

digitalWrite(transistor, HIGH); // El iniciar el programa el transistor
estara en estado alto

}

void loop() {

  while(digitalRead(sensorsonido) == LOW){}   // Mientras que el sensor
no detecte sonido no hace nada

        estado = digitalRead(transistor);         // Estado guarda el
comportamiento del transistor

   digitalWrite(transistor, !estado);  // Se escribe en el transitor el
estado opuesto que previamente tenia

  while(digitalRead(sensorsonido) == HIGH){}  // Mientras que el sensor
detecte sonido no hace nada

}
```

Lección 28 Control de motor a pasos con un encoder KY-040

En esta lección aprenderás a implementar un control del giro de un motor a pasos usando un encoder incremental KY-040 utilizando un Mega 2560

Materiales necesarios

- Tarjeta Mega 2560
- Módulo fuente de alimentación para protoboard
- Motor a pasos 28BYJ-48
- Driver ULN2003
- Encoder KY-040
- Cables jumper Macho-Hembra
- Cables jumper Macho-Macho
- Eliminador 9v 1A

Conocimientos previos

Encoder

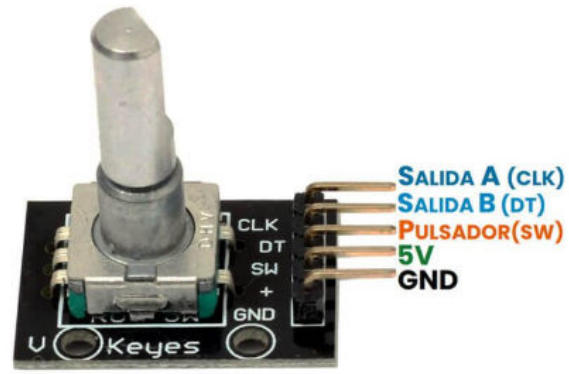
Un codificador rotatorio es un tipo de sensor de posición que se utiliza para determinar la posición angular de un eje giratorio. Genera una señal eléctrica, ya sea analógica o digital, de acuerdo con el movimiento de rotación.

Hay muchos tipos diferentes de codificadores giratorios que se clasifican por señal de salida o tecnología de detección. El codificador rotativo particular que usaremos en este tutorial es un codificador rotatorio incremental y es el sensor de posición más simple para medir la rotación.

Este codificador rotatorio también se conoce como codificador en cuadratura o codificador rotatorio relativo y su salida es una serie de pulsos de onda cuadrada.

KY-040

El **KY-040** es un **encoder incremental** que genera pulsos conforme se gira el eje, lo que permite detectar la rotación en una dirección específica.



El KY-040 tiene un eje que puede girar (normalmente 360 grados) y está conectado a dos interruptores de **tipo mecánico** llamados **A** y **B**.

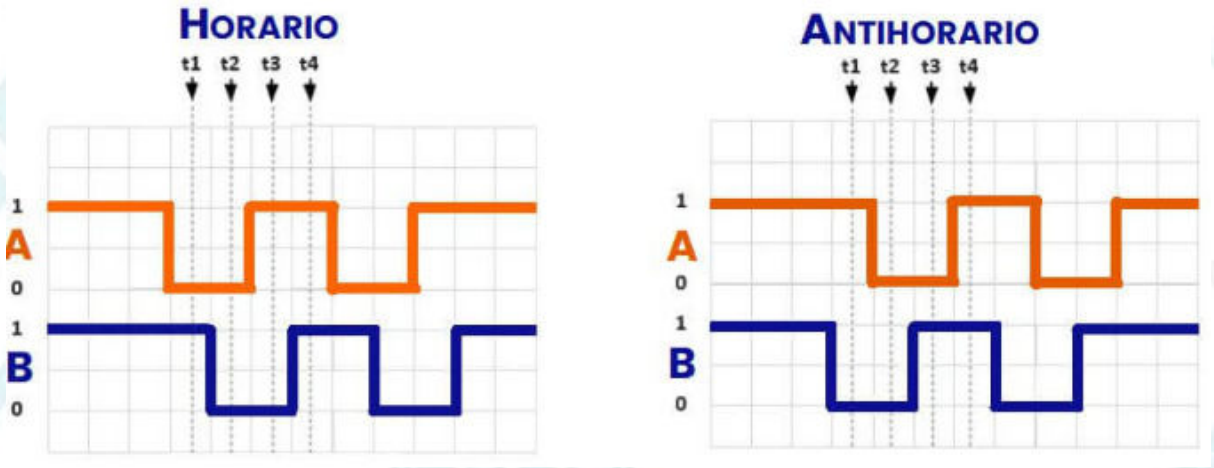
Además, cuenta con un pin de **presión** o **pulsador**, que se puede presionar para enviar una señal (comúnmente se usa como un botón para hacer una acción, como confirmar una selección en un menú).

Los pines **A** y **B** generan señales cuadradas (pulsos) a medida que el eje del encoder gira.

Las señales **A** y **B** están desfasadas entre sí 90 grados. Esto significa que el orden en que se activan los pulsos de A y B nos indica la dirección de la rotación:

- **Si A lidera a B**, el eje está girando en una dirección (por ejemplo, en sentido horario).
- **Si B lidera a A**, el eje está girando en la dirección opuesta (por ejemplo, en sentido antihorario)

GIRO	Tiempo	PINES	
		A(CLK)	B(DT)
Horario	t1	0	1
	t2	0	0
	t3	1	0
	t4	1	1
Antihorario	t1	1	0
	t2	0	0
	t3	0	1
	t4	1	1



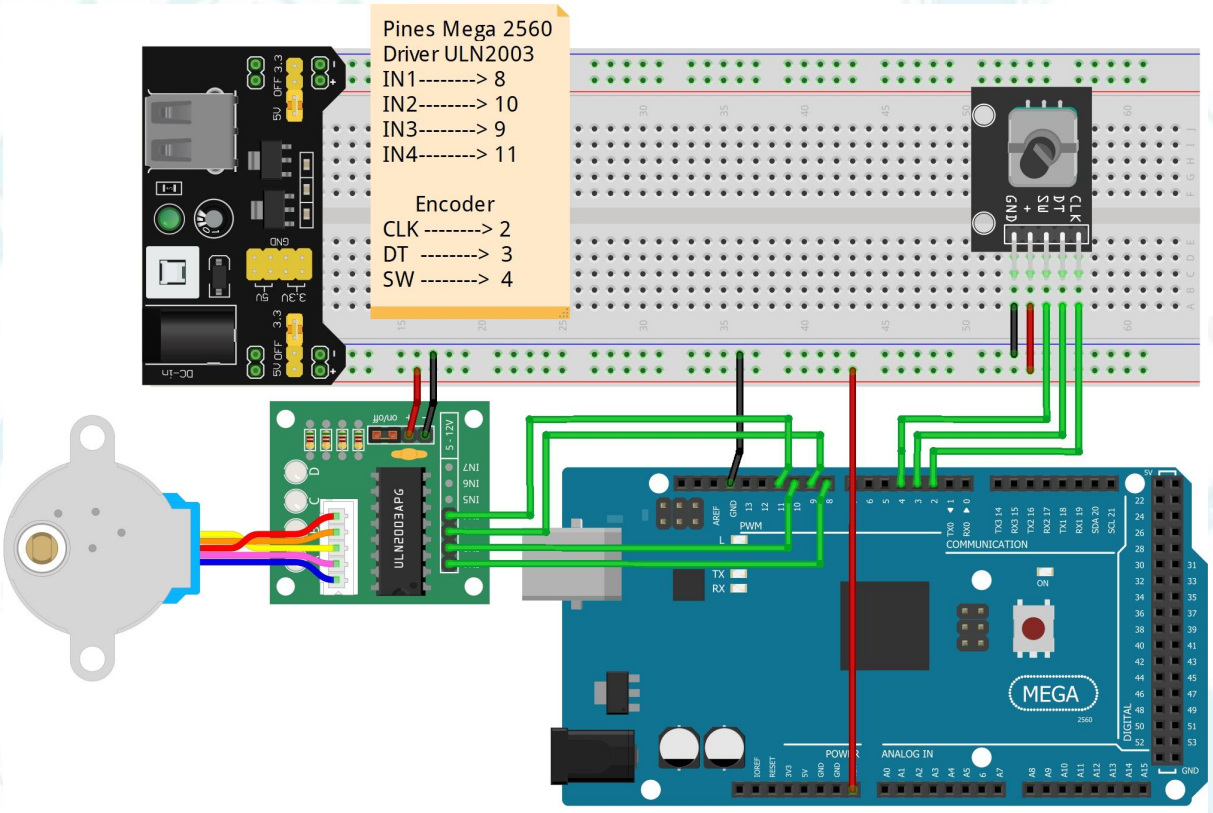
Cada vez que el eje gira, las señales A y B generan pulsos que pueden ser leídos por un microcontrolador.

Al contar los pulsos y observar el desfase entre A y B, el microcontrolador puede determinar con precisión la **dirección** de rotación y **cuánto ha girado** el encoder.

El KY-040 también incluye un **pulsador** (conectado al pin de **"SW"**), que se activa cuando presionas el eje del encoder hacia abajo. Esto puede ser usado como un botón para realizar una acción o seleccionar algo.

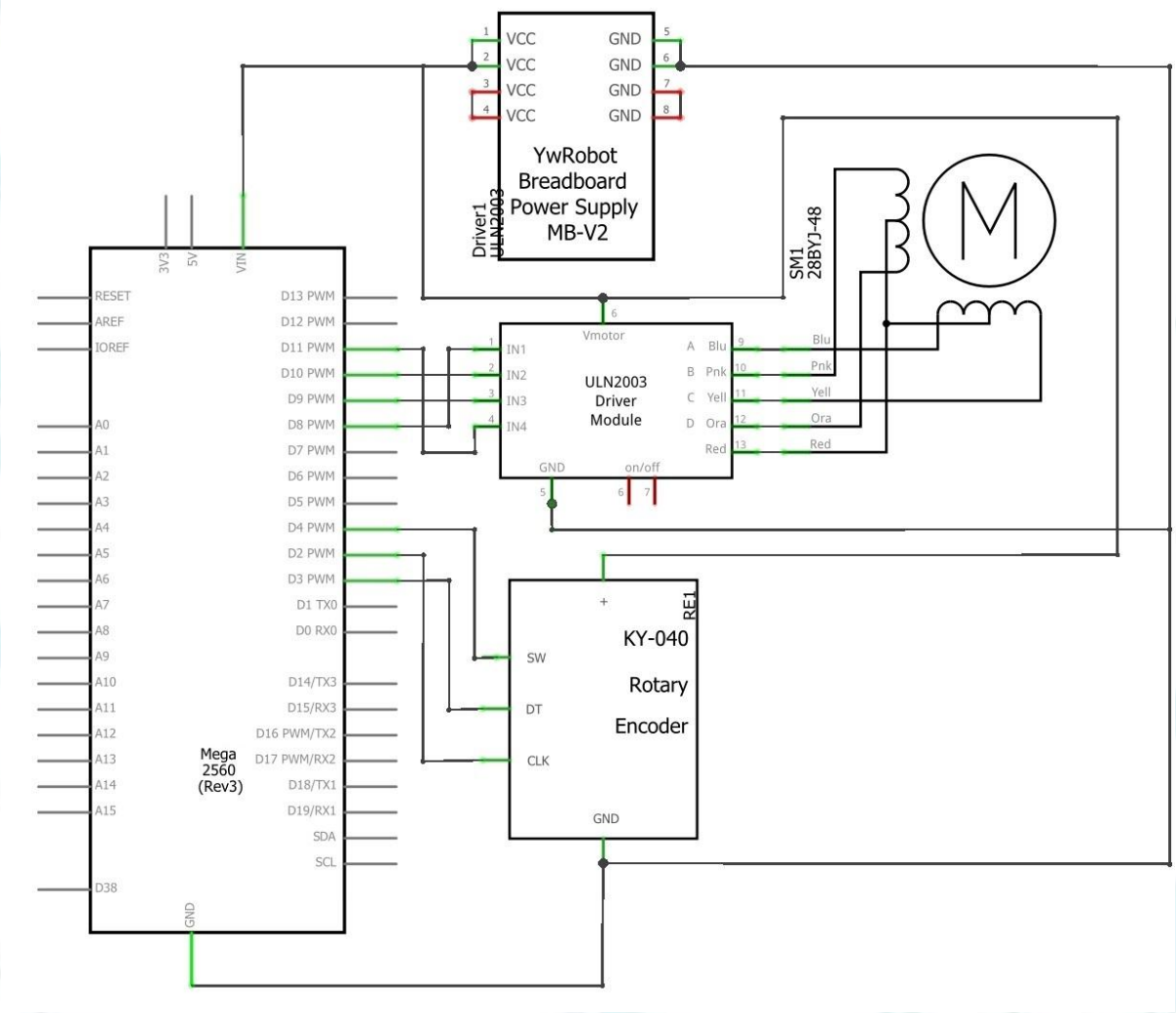
Diagrama de conexiones

Como ya se ha visto, el motor al ser un elemento que consume bastante potencia por que de ninguna manera se conecta su alimentación a la salida de 5v de la tarjeta Mega 2560, sino, más bien del módulo fuente de alimentación para protoboard.



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

El siguiente programa permite que el motor a pasos gire en la misma dirección que el encoder rotativo, en una única dirección. Dicho movimiento se graba y al pulsar el interruptor del encoder el motor gira con el movimiento que guardó

Es muy importante que no te excedas con el tiempo que el motor a pasos funciona ya que bastan 20 segundos para que se caliente uno de los reguladores de la fuente de alimentación para protoboard.

```
#include <Stepper.h>

#define STEPS 32

// Número de pasos para una revolución de eje interno 2048 pasos para una revolución de eje externo

volatile boolean TurnDetected;  //Necesita volatilidad para las interrupciones

volatile boolean rotationdirection; // rotación CW o CCW

const int PinCLK=2; // Generar interrupciones usando señal CLK

const int PinDT=3; // Leyendo señal DT

const int PinSW=4; // Leyendo Interruptor Push

int RotaryPosition=0; // Para almacenar la posición del motor paso a paso

int PrevPosition; // Valor de Posición de rotación anterior para verificar la precisión

int StepsToTake; // Qué tanto mover el motor paso a paso

// Configuración de la secuencia correcta para los pines del motor

// In1, In2, In3, In4 en la secuencia 1-3-2-4

Stepper small_stepper(STEPS, 8, 10, 9, 11);

// Interrumpir ejecuta la rutina si CLK pasa de ALTO a BAJO

void isr () {

 delay(6); // delay for Debouncing

 if (digitalRead(PinCLK))
```

```
 rotationdirection= digitalRead(PinDT);

 else

 rotationdirection= !digitalRead(PinDT);

 TurnDetected = true;

}

void setup () {

pinMode(PinCLK,INPUT);

pinMode(PinDT,INPUT);

pinMode(PinSW,INPUT);

digitalWrite(PinSW, HIGH); // Levantar botón de Resistor

attachInterrupt (0,isr,FALLING); // Interrupción 0 siempre conectada al
pin 2 en el Mega

}

void loop () {

 small_stepper.setSpeed(700); //El Máx parece ser 700

 if (!(digitalRead(PinSW))) { // revisar si el botón está presionado

 if (RotaryPosition == 0) { // revisar si el botón ya estaba presionado

 } else {

 small_stepper.step(-(RotaryPosition*50));

 RotaryPosition=0; // Reestablecer posición a CERO

 }

 }

 // Se ejecuta si se detectó rotación

 if (TurnDetected) {

  PrevPosition = RotaryPosition; // Guardar posición anterior en
variable

 if (rotationdirection) {

 RotaryPosition=RotaryPosition-1;} // disminuir la posición en 1 más

 {

 RotaryPosition=RotaryPosition+1;} // incrementar la posición por 1

 TurnDetected = false; // NO repita el bucle HASTA que se detecte una
nueva rotación

 // En qué dirección mover el motor paso a paso

 if ((PrevPosition + 1) == RotaryPosition) { // Mover motor CW
```

```
 StepsToTake=50;

 small_stepper.step(StepsToTake);

 }

 if ((RotaryPosition + 1) == PrevPosition) { // Mover motor CCW

 StepsToTake=-50;

 small_stepper.step(StepsToTake);

 }

 }

// Poner a bajo las entradas del driver es indispensable si no queremos
quemar la fuente

 digitalWrite(8, LOW);

 digitalWrite(9, LOW);

 digitalWrite(10, LOW);

 digitalWrite(11, LOW);

}
```

Lección 29 RFID MFRC522

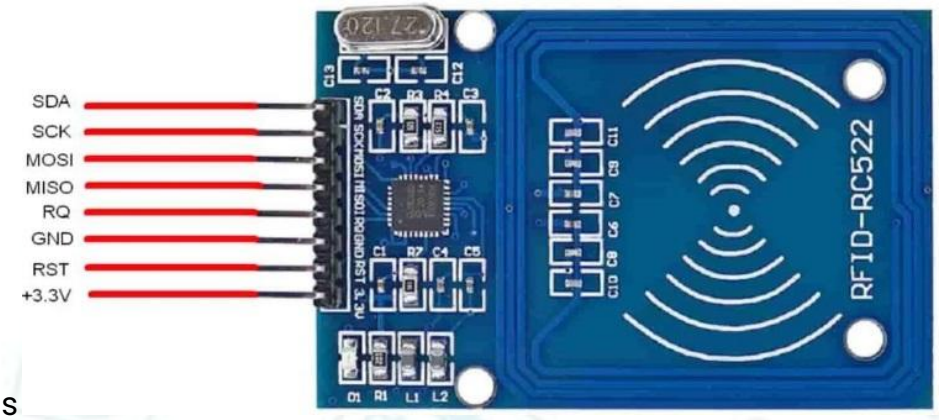
En esta lección aprenderás a implementar un control de acceso mediante el módulo RFID RC522, un servomotor sg90 y un buzzer a través del Mega 2560.

Materiales necesarios

- Tarjeta Mega 2560
- Modulo RFID RC522
- Servomotor sg-90
- Buzzer activo
- Resistencia de 220 Ω 1/4w
- Modulo fuente de alimentacion para protoboard
- Cables jumper Macho-Macho
- Cables jumper Macho-Hembra

Conocimientos previos

El módulo RFID MFRC522 es un lector de tarjetas RFID (identificación por radiofrecuencia) que permite interactuar con tarjetas o etiquetas RFID para leer información de ella



s

El MFRC522 se comunica con una tarjeta RFID utilizando ondas de radio a una frecuencia de 13.56 MHz. La tarjeta contiene un chip que responde a la señal del lector con un identificador único, lo que permite al lector leer la información. Cuando se acerca una tarjeta RFID al lector, el MFRC522 genera un campo electromagnético que es captado por la tarjeta.

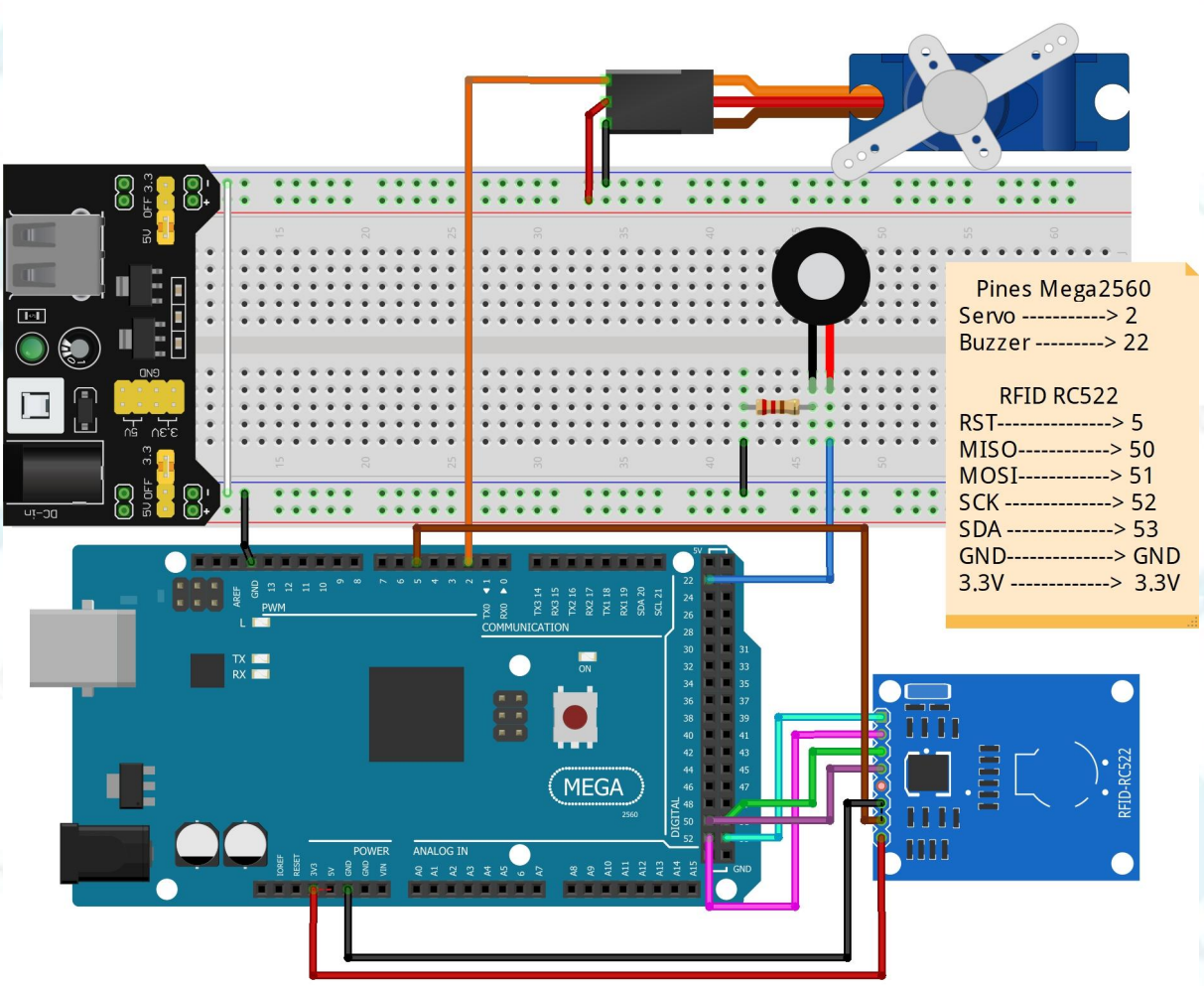
La tarjeta entonces responde enviando su número de serie (ID) o la información almacenada dentro de ella. Este ID es lo que el MFRC522 captura y envía al microcontrolador para que se pueda procesar.

El MFRC522 se comunica con un microcontrolador a través de un protocolo de comunicación SPI (Serial Peripheral Interface), lo que permite que se envíen y reciban datos entre ambos. A través de este protocolo, el MFRC522 puede enviar el identificador de la tarjeta o el estado de su lectura al microcontrolador.

Para usar el MFRC522, se necesita una librería compatible en el microcontrolador (por ejemplo, para el Mega, se utiliza la librería MFRC522). Con ella, puedes configurar el lector, leer los identificadores de las tarjetas y realizar diversas acciones según tu aplicación (encender una luz, desbloquear una puerta, etc.).

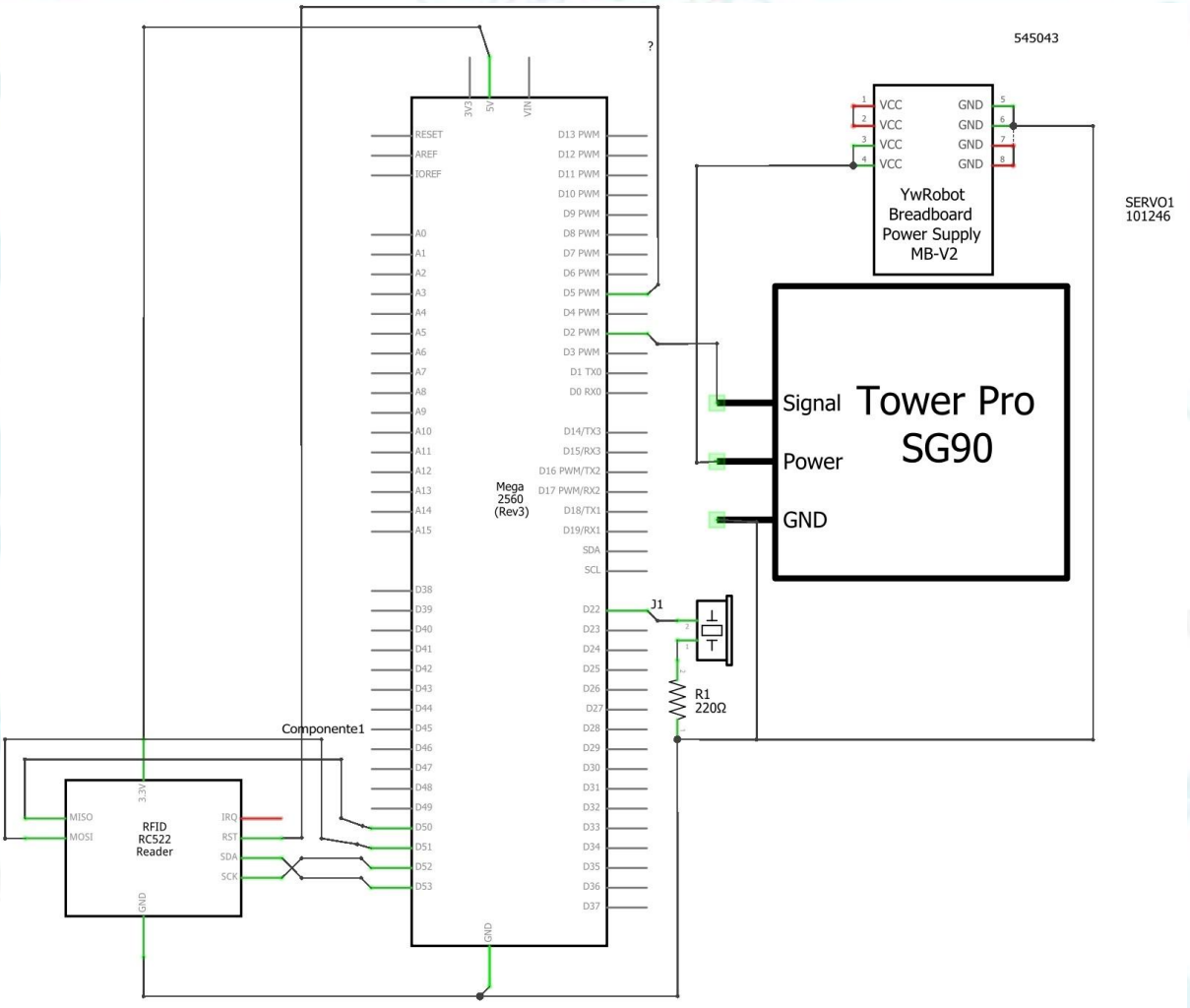
Diagrama de conexión

Ten presente que para esta lección haremos uso del monitor serie por lo que en el siguiente diagrama se obvia la conexión via USB a la PC ;)



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

Una vez que subas el siguiente código recuerda que cada TAG, ya sea en forma de llavero o tarjeta tiene un UID único, mismo que primero deberás leer en el monitor serial y después ingresarlos en forma hexadecimal en el código para que este lo reconozca y le conceda el acceso

```
//Librerías para poder utilizar el RFID-RC522, ServoMotor SG90 y protocolo de comunicación SPI

#include <SPI.h>

#include <MFRC522.h>

#include <Servo.h>

#define RST_PIN 9 // Pin del RDFI-MFRC522 a Pin 5

#define SS_PIN 53 // También llamado SCL O SCK

#define buzzer 22

Servo sg90;

MFRC522 mfrc522(SS_PIN, RST_PIN); // Creando instancia para RFID

byte i; //variable , elemento de matriz de lectura de datos por el lector

byte datoUID[4]; //matriz para guardar valores arrojados por el detector Valor hexadecimal al usar el llavero <previamente comprobado>, puede variar

byte llavero[4]={0x53,0x8F,0x32,0xBD};

//Valor hexadecimal al usar la tarjeta <previamente comprobado>, puede variar

byte tag[4]={0x33,0xF0,0x20,0x0D};

void setup() {

Serial.begin(9600); // Inicializa comunicación serial

SPI.begin(); // Iniciación de comunicación por bus SPI

sg90.attach(2); //Pin para la señal al servo-Pin 3

sg90.write(0); //Siempre que se inicie el programa el servo tendrá un valor de 0°
```

```
mfrc522.PCD_Init(); // inicialización de MFRC522

Serial.println(F("Coloque tarjeta para realizar escaner")); //mensaje para usuario

pinMode(buzzer, OUTPUT);

}



void loop() {

//Reinicia el ciclo si no hay detección de tarjeta

if ( ! mfrc522.PICC_IsNewCardPresent())

return;

// Lee el valor del ID presentado

if ( ! mfrc522.PICC_ReadCardSerial())

return;

Serial.println("UID:");

//Se guarda los datos del ID

for (byte i = 0; i < mfrc522.uid.size; i++) {

//limitando la lectura de los valores en la matriz hexadecimal a 10 dígitos

if (mfrc522.uid.uidByte[i] < 0x10) {

Serial.print("0");

} else {

Serial.print("");

}

//imprime en Monitor Serial, el valor leído por el detector

Serial.print(mfrc522.uid.uidByte[i], HEX);

datoUID[i]= mfrc522.uid.uidByte[i]; //Se guardan los valores en la variable dato UID

}

mfrc522.PICC_HaltA();

if (datoUID[i] == llavero[i]){ //Si la lectura corresponde a los valores de la matriz llavero

digitalWrite(buzzer, HIGH);

delay(50);

digitalWrite(buzzer, LOW);
```

```
delay(50);
digitalWrite(buzzer, HIGH);
delay(50);
digitalWrite(buzzer, LOW);
sg90.write(180); //se abre la puerta
delay(5000); // La puerta permanece abierta durante 5 segundos
sg90.write(0);
}
else{ // Si no se reconoce la tarjeta sonará una alerta en el buzzer
  digitalWrite(buzzer, HIGH);
delay(350);
digitalWrite(buzzer, LOW);
delay(50);
digitalWrite(buzzer, HIGH);
delay(350);
digitalWrite(buzzer, LOW);
}
}
```

Como ejemplo tengo el siguiente UID leído en el monitor serie tras colocar un TAG sobre el RFID:

```
Coloque tarjeta para realizar escaner
UID:
538F32BD
```

Deseo que dicho TAG sea admitido y que se accione el servomotor por lo que introduzco su UID en forma hexadecimal en la siguiente linea de codigo como a continuación se muestra

```
13    byte llavero[4]={0x53,0x8F,0x32,0xBD};
```

Mi UID está almacenado en el arreglo llamado “llavero” mismo que se admite para accionar el servomotor que abriría una pequeña puerta

```
47    if (datoUID[i] == llavero[i]){ //Si la lectura corresponde a los valores de la matriz llavero
```

Lección 30 Sensor de gas MQ-2

En esta lección aprenderás a construir una sencilla alarma que además de advertir sobre concentraciones peligrosas de gas mediante un buzzer, activa un servo motor simulando la apertura de una compuerta para ventilar el área.

Dicha alerta sonora solo se puede desactivar al presionar un botón a la vez que el sensor deja de detectar concentraciones peligrosas de gas.

Materiales necesarios

- Módulo de fuente de alimentacion para protoboard
- Eliminador 9V 1A
- Tarjeta Mega 2560
- Cables jumper Hembra-Macho
- Cables jumper Macho-Macho
- Protoboard
- Buzzer activo
- Resistencia 220Ω 1/4w
- Botón pulsador
- Sensor MQ-2

Conocimientos previos

Sensor MQ-2

El sensor MQ-2 es usado para detectar gases como el metano, propano, monóxido de carbono y humo. Funciona utilizando un material semiconductor en su interior que cambia de resistencia cuando detecta estos gases.

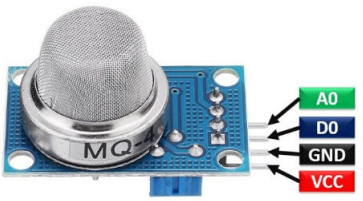
Cuando el gas entra en contacto con el sensor, reacciona con el material semiconductor, lo que provoca un cambio en la resistencia eléctrica. Este cambio es interpretado por un circuito de medición que genera una señal proporcional a la concentración de gas presente.

El sensor tiene dos partes clave:

1. Calentador (heater): Se encarga de calentar el material semiconductor.
2. Material sensible (semiconductor): Cambia de resistencia según la concentración del gas.

El sensor propiamente se encuentra encerrado en dos capas de malla de acero inoxidable que asegura que el elemento calentador interno no cause una explosión dado que su ambiente de sensado son gases inflamables, además filtra las partículas suspendidas para que solo gases accedan a la cámara. Dentro, se encuentra una bobina de níquel-cromo para formar el sistema de calefacción y un revestimiento de dióxido de estaño (que es sensible a gases combustibles) forma el sistema de detección.

El sensor MQ-2 generalmente tiene un pin de salida analógica (A0))y uno digital (D0). La salida analógica te da una señal continua que puedes medir con un microcontrolador, y la salida digital se usa para activar o desactivar un dispositivo según si el gas detectado excede un umbral preestablecido.



Como se observa en el siguiente gráfico los módulos MQ son sensibles a más de un gas y en diferente proporción por lo que no son recomendables para identificar la presencia de un gas específico.

El voltaje que el sensor emite cambia de acuerdo con el nivel del humo/gas que existe en la atmósfera. El sensor emite una tensión proporcional a la concentración del humo/gas.

En otras palabras, la relación entre el voltaje y la concentración del gas es la siguiente: A mayo concentración del gas, mayor la tensión de salida. A menor concentración del gas, menor tensión de salida.

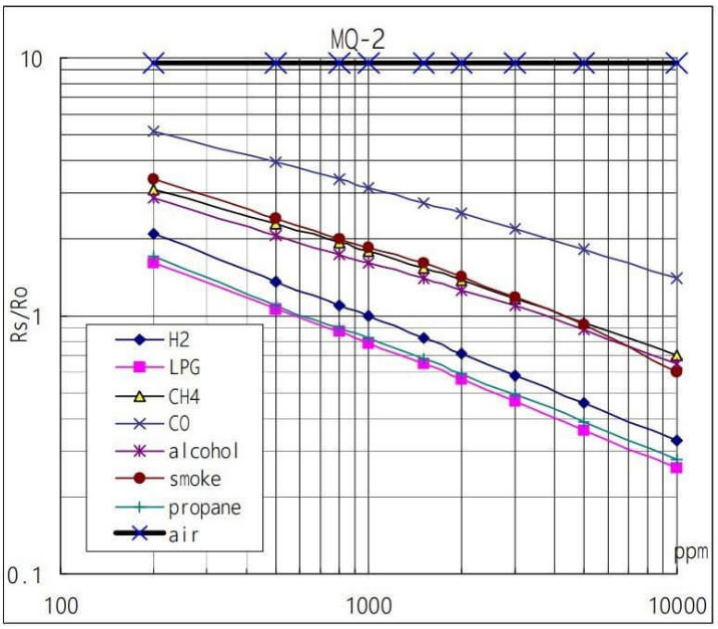
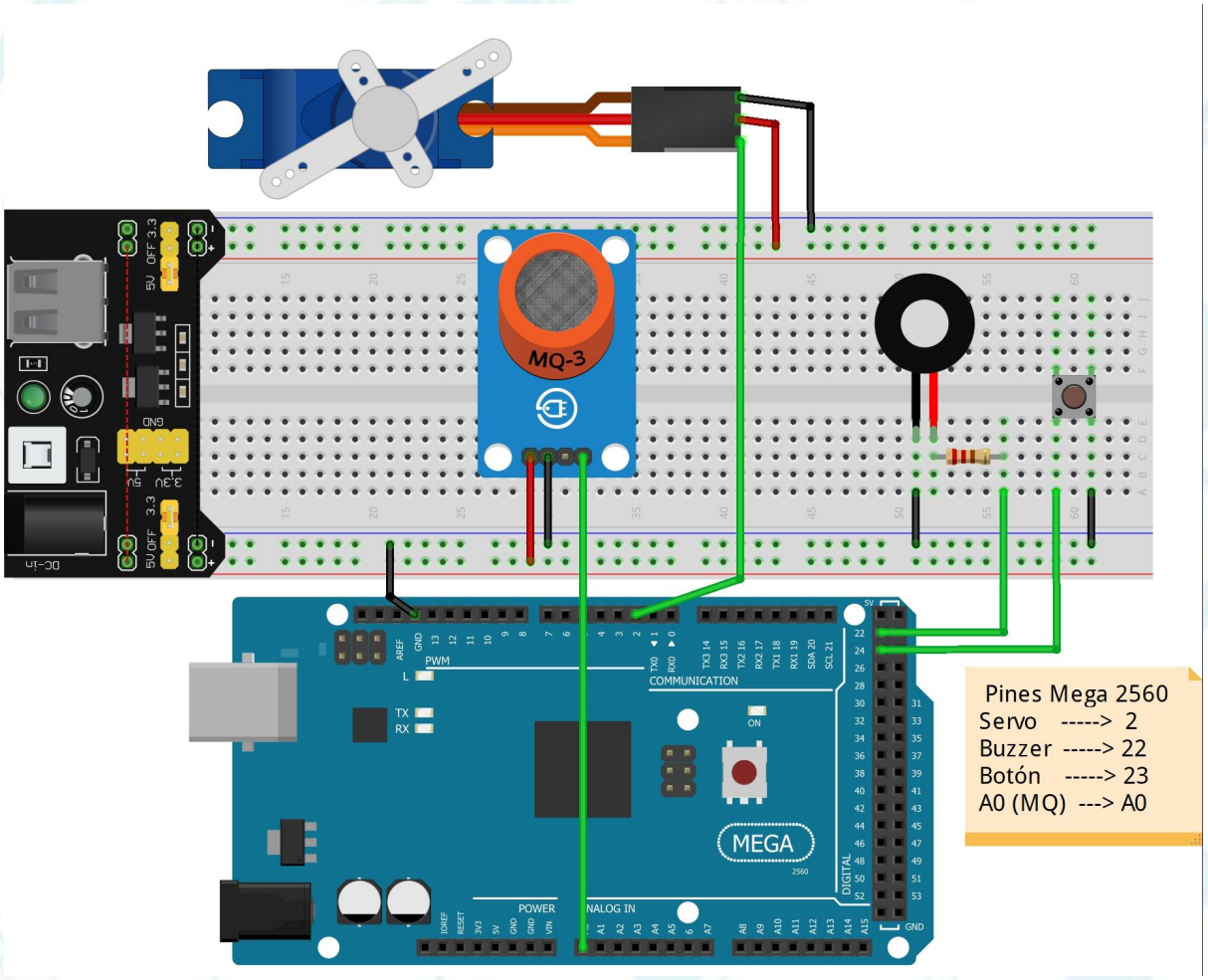


Diagrama de conexión

Debido a que el sensor de gas tiene un elemento calefactor el cual puede llegar a consumir 1W no se conecta a los 5v del Mega 2560 sino más bien al módulo de fuente de protoboard al igual que el servomotor

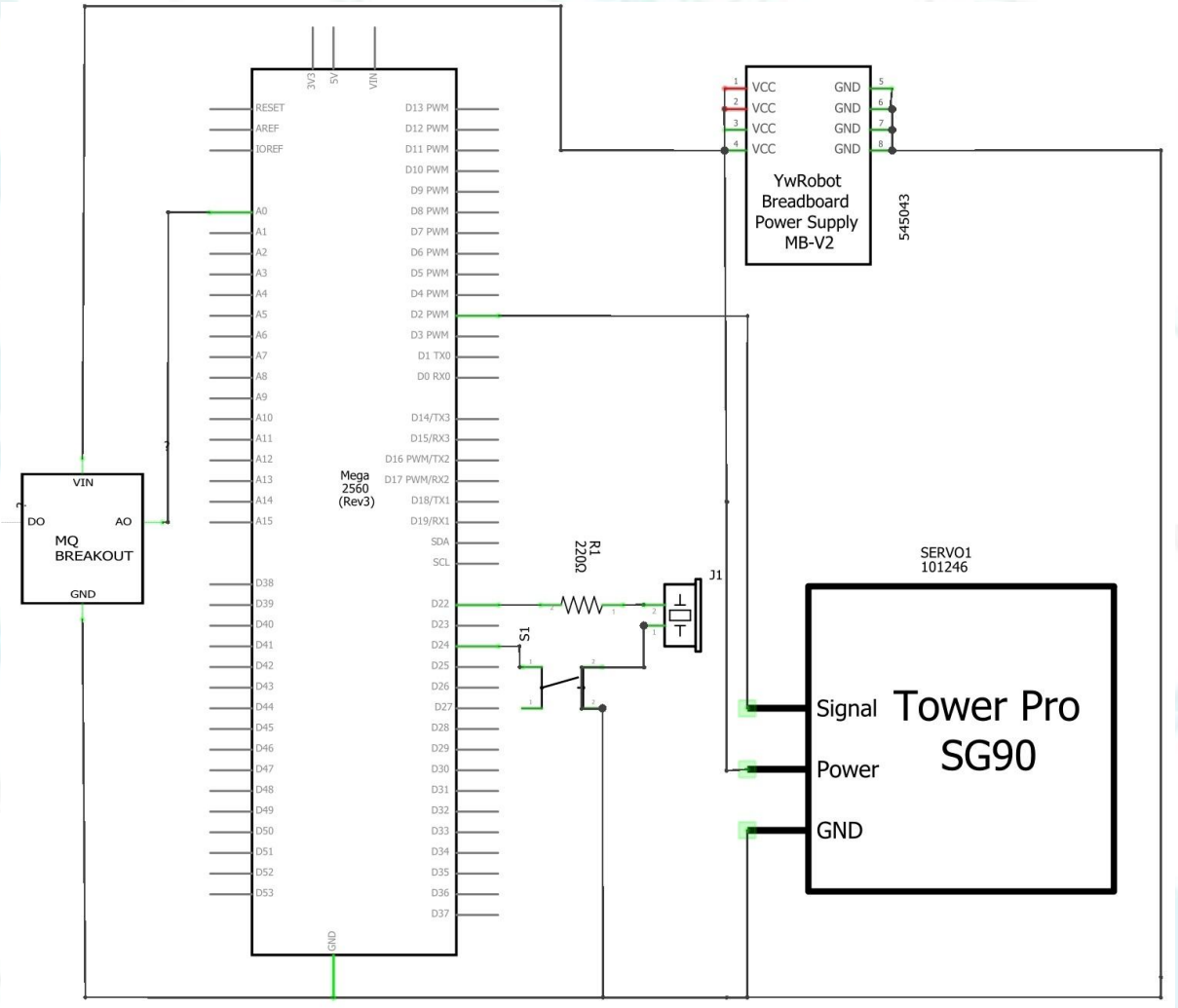
Siguiendo el diagrama que a continuación se propone estarás cuidando de tu equipo, no solo de tu Mega sino también de tu PC.

La única conexión que existe entre el Mega y el módulo de fuente de alimentación es hacia GND, por lo que podrás conectar USB a la tarjeta Mega 2560 con tu PC y ver en el monitor serial las lecturas del sensor de gas.



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

El programa que a continuación se muestra te permitirá tener una confiable alarma contra concentraciones peligrosas de gas ya que se activará un buzzer cuando el sensor MQ detecta alguna fuga. Dicha alarma solo es posible de desactivar al presionar un botón siempre y cuando la concentración de gas haya disminuido; es decir, la alarma se enclava

```
// Incluimos la libreria para manejar al servo motor

#include <Servo.h>

// Definimos variables que usaremos como pines del Mega

 #define buzzer 22  // Buzzer en el pin digital 22

 #define mq A0      // Salida analogica (A0) del sensor de gas MQ en el pin A0

 #define boton 23 //  Botón en el pin digital 23

// Tu valor de Umbral para comparar con la lectura del sensor

int umbral = 640;

int estado = HIGH; // Variable auxiliar que nos ayudará al envlavamiento de la alarma

Servo miservo;  // Creamos el objeto tipo Servo llamado "miservo"



void setup() {

pinMode(buzzer, OUTPUT); // Configuramos al buzzer como salida y al pin mq del sensor como entrada

pinMode(mq, INPUT);

pinMode(boton, INPUT_PULLUP); // Botón como entrada y habilitamos una resistencia de PULL UP  interna

miservo.attach(2);   // Establecemos que el pin 2 del Mega será donde manejaremos el servo motor

miservo.write(0);     // Iniciamos con el servo en la posición de 0 grados o ventila cerrada
```

```
// Este tiempo de doce segundos antes de iniciar el programa es necesario ya

// que le va a permitir al sensor MQ calentarse y brindar lecturas confiables

delay(12000);

Serial.begin(9600); // Iniciamos el monitor serial a 9600 Baudios

}

void loop() {

int sensorgas = analogRead(mq); // Leemos al sensor MQ y le asignamos el valor a la variable sensorgas

Serial.print("Sensor de gas: ");    // Imprime el valor de la lectura antes realizada

Serial.println(sensorgas);

if(sensorgas> umbral) estado = HIGH; // Checa si ha superado el valor de umbral

else estado = LOW;

// Mientras que la variable estado está en alto y la lectura del sensor sea mayor que el umbral

// se va a activar el buzzer y se abre la ventila mediante el servomotor

while(estado == HIGH && sensorgas > umbral ){

miservo.write(180);

digitalWrite(buzzer,HIGH);

delay(150);

digitalWrite(buzzer,LOW);

delay(50);

// Si se presiona el botón y el sensor detecta menos gas se sale de este bucle while

if(digitalRead(boton)==LOW && analogRead(mq) < umbral) estado = LOW;

else estado = HIGH;

}

// Estando fuera del bucle la alarma permanece apagada y la ventila cerrada

miservo.write(0);

digitalWrite(buzzer,LOW);

delay(800);

}
```

Lección 31 Sensor de humedad del suelo

En esta lección, aprenderá cómo usar el sensor de humedad del suelo

en sus proyectos con el Mega 2560

En este ejemplo, se leerán los valores de salida del sensor encendiendo un color diferente en un led RGB además de una bocina como alarma para los diferentes niveles de humedad detectados.

Materiales necesarios

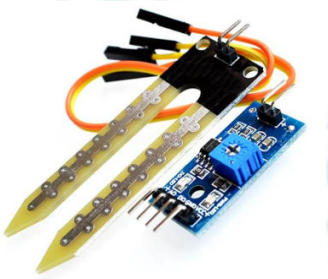
- Tarjeta Mega 2560
- Led RGB catodo comun
- Resistencias de 220Ω a 1/4w
- Buzzer activo
- Módulo sensor de humedad del suelo FC-28
- Protoboard

Conocimientos previos

Sensor FC-28

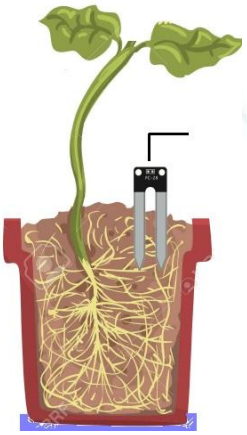
Este sensor está compuesto por dos partes principales:

Sonda de Humedad: Es la parte que se inserta en el suelo y se encuentra formada por dos electrodos que actúan como sensores para medir la resistencia eléctrica del suelo. La resistencia de los electrodos varía dependiendo de la cantidad de agua en el suelo. Cuanta más agua haya en el suelo, menor será la resistencia, ya que el agua conduce la electricidad mejor que el suelo seco.



Circuito de Medición: El circuito toma la señal de los electrodos y la convierte en un valor analógico que puede ser leído por un microcontrolador. Este valor puede ser interpretado para determinar el nivel de humedad del suelo.

¿Cómo se mide la humedad?



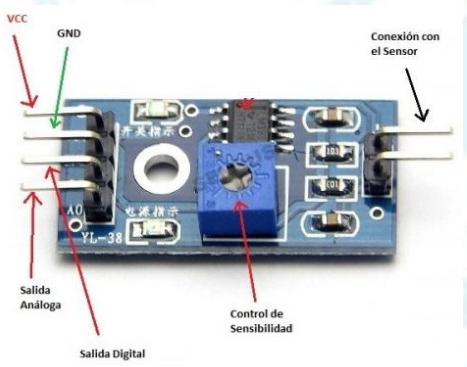
Suelo seco: La resistencia entre los electrodos será alta, ya que el agua es un buen conductor de electricidad y en su ausencia el suelo no conduce bien la corriente. Esto indica un bajo nivel de humedad.

Suelo mojado: La resistencia será baja, ya que el agua presente en el suelo facilita la conducción de electricidad, lo que indica un alto nivel de humedad.

Salida del sensor

Salida analógica(A0): Esta salida proporciona un valor proporcional al nivel de humedad. A medida que la humedad aumenta, el valor de la señal analógica disminuirá (menos resistencia = más humedad).

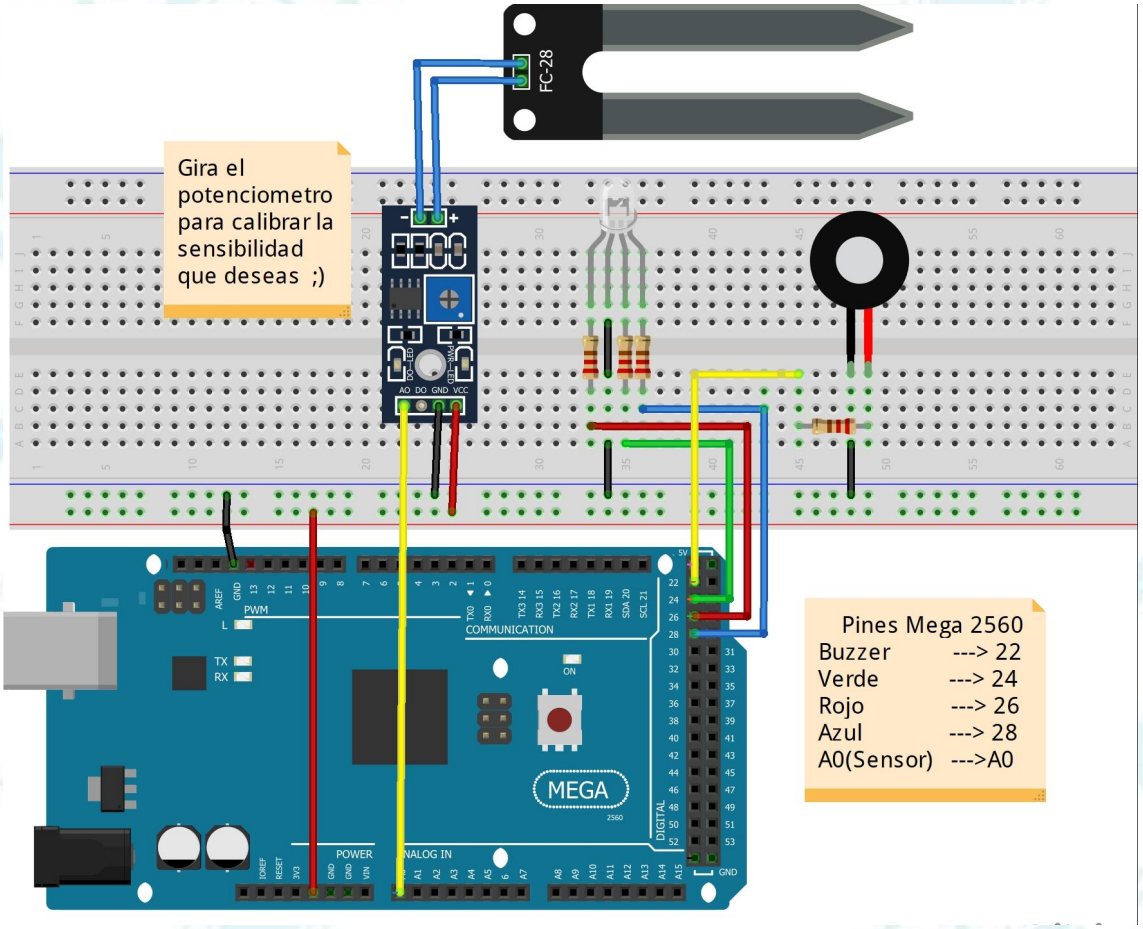
Salida digital(D0): Este pin genera una señal de encendido (HIGH) o apagado (LOW) basada en un umbral preestablecido. Por ejemplo, cuando la humedad baja por debajo de un nivel determinado, el pin digital se activa para encender una bomba de riego o enviar una alarma.



La precisión del sensor puede verse afectada por el tipo de suelo, ya que su conductividad varía según la cantidad de sal y minerales que tenga.

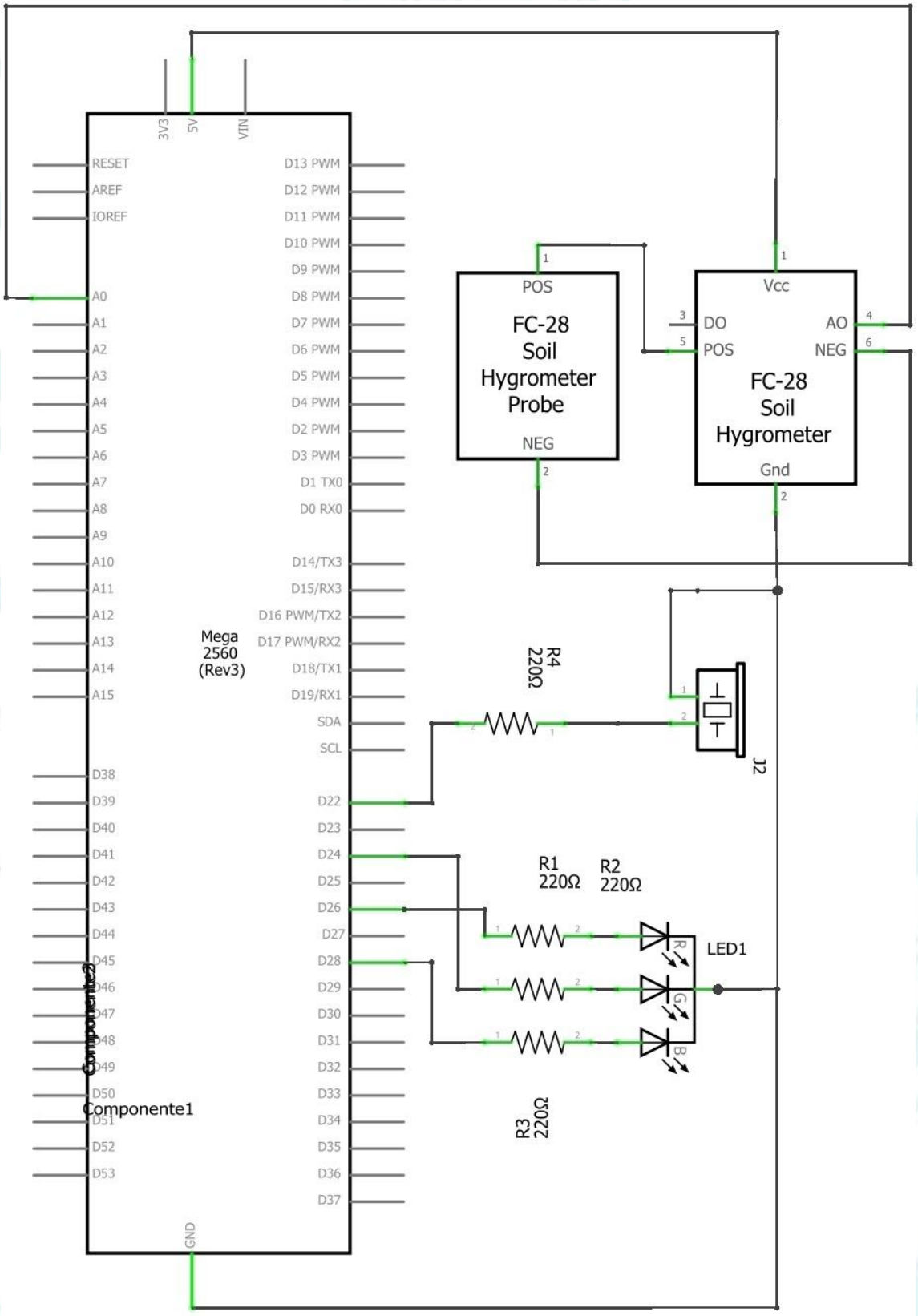
Diagrama de conexión

A diferencia de las prácticas donde se utilizaba el módulo de fuente de alimentación, en este caso alimentamos a todos los componentes desde el Mega 2560 y este a su vez está conectado por medio del cable USB a la PC.



MEGA 2560

Transforma tus ideas en acción.



Código de funcionamiento

Este codigo realizar la lectura analogica del sensor de humedad que adquiere valores de 0 a 1023 interpretando el 0 como máxima humedad detectada y el 1023 como el ambiente más seco al que la sonda puede estar expuesta y evaluando tres rangos de dichas lecturas para encender cada uno de los leds internos del led RGB y una alarma sonora mediante un buzzer.

```
// Declaramos variables con el número de Pin en el Mega

int sensor = A0;

int verde = 24;

int rojo = 26;

int azul = 28;

int alarma = 22;

void setup(){

  // Configuramos el pin del sensor como entrada

 pinMode(sensor, INPUT);

// Configuramos salidas

 pinMode(verde, OUTPUT);

 pinMode(rojo, OUTPUT);

 pinMode(azul, OUTPUT);

pinMode(alarma, OUTPUT);

 digitalWrite(alarma, LOW);

}

void loop() {

 // Leemos en el pin analógico 0 conectado a la salida del sensor de
humedad

 int humedad = analogRead(sensor);

  // Si el sensor entrega valores entre 0 y 341 se enciende el led azul
```

```
 if(341 >= humedad && humedad>0){

digitalWrite(azul, HIGH);

digitalWrite(rojo, LOW);

digitalWrite(verde, LOW);

digitalWrite(alarma, LOW);

 }

// Si el sensor entrega valores de 341 a 682 se enciende el led verde

 else if(682 >= humedad && humedad>341){

digitalWrite(azul, LOW);

digitalWrite(rojo, LOW);

digitalWrite(verde, HIGH);

digitalWrite(alarma, LOW);

 }

// Si el sensor entrega valores de 682 a 1023 se enciende el led rojo a
la par que una alarma

else if(1023 >= humedad && humedad>682){



digitalWrite(azul, LOW);

digitalWrite(rojo, HIGH);

digitalWrite(verde, LOW);

digitalWrite(alarma, LOW);

digitalWrite(alarma, HIGH);

delay(50);

digitalWrite(alarma, LOW);

delay(50);

 }

 delay(500);

}
```

Gracias por su preferencia

Manual elaborado por el Equipo UNIT Electronics
Marzo 2025

uelectronics.com