Contenido

INTRODUCCIÓN

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo integrado (IDE), diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares, que pueden abarcar desde sencillas aplicaciones electrónicas domésticas hasta proyectos más elaborados para la industria. Su principal ventaja es su facilidad de programación, al alcance de cualquiera. Para encontrar toda la información que necesites sobre Arduino en la

EL HARDWARE DE LA PLACA ARDUINO

A modo de ejemplo mostramos la placa Arduino UNO que incorpora el chip ATmega328. Tienen 14 entradas/salidas digitales, 6 entradas analógicas, entradas de alimentación, conexión USB para conectar al ordenador, conector de alimentación y un botón de Reset.

Alimentación

Las placas pueden ser alimentadas por medio de la conexión USB o con una fuente de Alimentación externa de entre 6 a 20 V, aunque el rango recomendado es de 7 a 12 V.

Las fuentes de alimentación externas (no-USB) pueden ser tanto un transformador como una batería. El transformador se puede conectar usando un conector macho de 2.1 mm con centro positivo en el conector hembra de la placa. Los cables de la batería pueden conectarse a los pines **Gnd** y **Vin** en los conectores de alimentación (POWER)

Entradas y Salidas

Las entradas son señales que brindan información a un centro de procesamiento. Como por ejemplo un mouse o un teclado son entradas o inputs de un ordenador. Arduino utiliza como entradas a los sensores.

Los sensores toman del mundo físico señales como

- Temperatura
- Luz
- Distancia
- Presencia, entre otras.

Existen otros componentes que ingresan información al microcontrolador, pero que no son necesariamente sensores, por ejemplo

- Un botón que ingresa un valor binario
- Un potenciómetro que ingresa múltiples valores en un rango
- O un teclado numérico que ingresa valores personalizados

Las salidas son señales que provienen de un centro de procesamiento. Como por ejemplo un monitor o un parlante son salidas o outputs de un ordenador. Arduino utiliza como salidas a los actuadores.

Los actuadores convierten las señales de corriente o voltaje en señales físicamente útiles como:

- La Luz de un led
- Los mensajes en un display
- El giro y movimiento de un motor
- y los sonidos en un parlante.

Cada uno de los **14 pines digitales** (de 0 a 13) pueden utilizarse como entradas o como salidas usando las funciones pinMode(), digitalWrite() y digitalRead(). Las E/S operan a 5 V. Cada pin puede proporcionar o recibir una intensidad máxima de 40 mA.
Los pines **3, 5, 6, 9, 10,** y **11** proporcionan una salida PWM (modulación por anchura de pulsos) de 8 bits de resolución (valores de 0 a 255) mediante la función **analogWrite()**.
El pin digital 13 lleva conectado un LED integrado en la propia placa. Se encenderá cuando dicho pin se configura como salida y adopte un valor HIGH.; con valor LOW se apaga.
La placa tiene **6 entradas analógicas**, y cada una de ellas proporciona una resolución de 10 bits (1024 valores).

Comunicaciones

La placa Arduino Uno proporciona comunicación en modo serie a través de los pines digitales **0** (RX) y **1**(TX). Un chip (*CH340*) integrado en la placa canaliza esta comunicación serie a través del puerto USB. El software de Arduino incluye un **monitor de puerto serie** que *permite enviar y recibir información textual hacia y desde la placa Arduino*. Los leds RX y TX de la placa parpadearán cuando se detecte comunicación transmitida a través de la conexión USB.

Microcontrolador

Son circuitos electrónicos que sirven para manipular, interpretar y transformar las señales de voltaje y corriente provenientes de los sensores o entradas, procesar esta información para tomar las decisiones y generar acciones en los actuadores o salidas.

Arduino puede tomar información del entorno a través de sus entradas, para esto posee toda una gama de sensores, y puede afectar aquello que le rodea controlando luces, motores y otros actuadores.

El microcontrolador en la tarjeta Arduino es el Atmega328p. Cuando un proyecto ha sido prototipado, en la implementación se puede solo utilizar el microcontrolador sin necesidad de utilizar toda la tarjeta Arduino, permitiendo su reutilización.

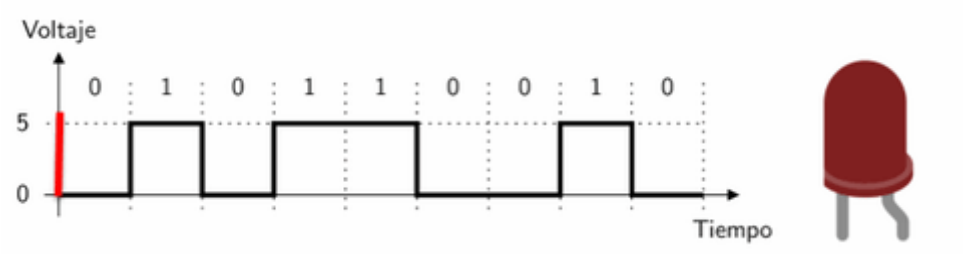
Señales electrónicas

En electrónica se trabaja con variables que se toman en forma de voltaje o corriente, éstas se pueden denominar señales electrónicas.

Las señales pueden ser de dos tipos:

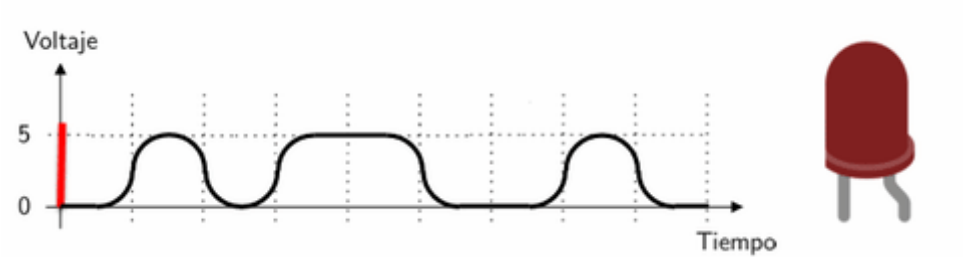
Variables digitales

Las variables digitales también llamadas variables discretas. Se caracterizan por tener dos estados diferenciados, es por esto que se suele llamarlas variables binarias. Un ejemplo son las transmisiones de datos a través de un cable de red.



Variables Analógicas

Son aquellas que pueden tomar un número infinito de valores comprendidos entre dos límites. La mayoría de los fenómenos de la vida real son señales de este tipo.(el sonido, la temperatura, la voz) Un ejemplo de una variable analógica es el información enviada a un altavoz.



PROGRAMACIÓN EN ARDUINO

La programación en Arduino, es muy similar a la de muchos otros lenguajes de programación esencialmente al lenguaje C, del cual hereda muchas funcionalidades.

Algoritmo

Un algoritmo es un conjunto de instrucciones detalladas paso a paso para resolver un problema o completar una tarea. Los algoritmos pueden ser cosas muy cotidianas como, una receta para cocinar, el método usado en resolver una suma, o el proceso de doblar una camisa, para todas estas actividades implícitamente realizamos pasos que hemos aprendido con el pasar de los años. En Arduino, escribiremos algoritmos que le indicarán al microcontrolador cómo realizar una tarea. Por ejemplo para hacer parpadear un led.

Esta tarea posee los pasos siguientes.

- Encender el led
- Esperar
- Apagar el led
- Esperar

Para que este conjunto de pasos se ejecutan continuamente, anidamos los pasos en un bucle o ciclo, un bucle realiza la repetición de un conjunto de instrucciones infinitamente, hasta cumplir una condición o definiendo previamente un número de veces.

En la programación de microcontroladores es normal que el programa principal se repita infinitamente hasta que se resetee el programa o se desconecte el microcontrolador.

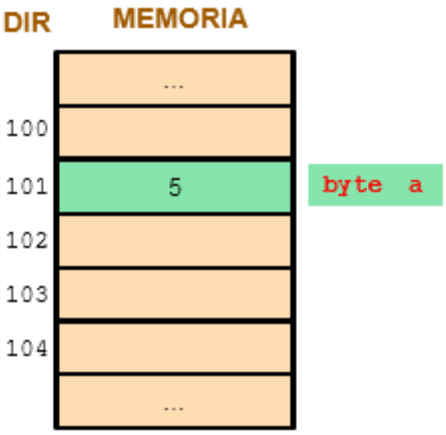
Variables

Una variable es la manera de codificar, representar y almacenar un valor dentro de un programa y así facilitar su manipulación. Las variables son almacenadas en la memoria del microcontrolador. Para que Arduino consiga realizar las tareas que le asignamos necesita el procesamiento e interpretación de estos datos.

```
byte  a = 5;
```

Hagamos una abstracción.

Consideremos a las variables como los cajones de un armario, cada uno tiene una etiqueta describiendo el contenido, acompañado de un tipo que indicará el número de cajones necesarios para almacenar un contenido y dentro se encuentra el valor de la variable como el contenido del cajón.



Tipos de dato en Arduino

Arduino considera los tipos de datos siguientes:

Entre los más importantes y usados tenemos:

- El tipo *boolean* que indica verdadero o falso.
- El tipo *int* para almacenar números enteros.
- El tipo *float* para números con decimales, no enteros
- El tipo *string* que es una secuencia y agrupación de datos de tipo char. Se utiliza para almacenar cadenas de texto.
- y los *array* que es una colección de datos de un mismo tipo.

Array

Un array es una colección de variables de un mismo tipo, que comparten la misma etiqueta o nombre.

Declaración

La sintaxis para declarar un array es simple solo agregamos corchetes al nombre de la variable, tener en cuenta que este array no posee un tamaño determinado.

Variable

```
int Pin;
```

```
int Pin = 5;
```

Array

```
int Pins[];
```

Si queremos que este array tenga un número determinado de elementos colocamos tal numero dentro de los corchetes.

```
int Pins[3];
```

Para agregar elementos en la declaración del array, los colocamos entre llaves separados por comas.

```
int Pins[3] = {1,2,3};
```

Para acceder a cada uno de los elementos del array. sólo hay que colocar el índice de este entre los corchetes, recordar que el primer índice del array es 0.

```
Pins[índice];
```

Funciones

Una función es un bloque de código que tiene un conjunto de instrucciones que realizan una tarea específica. Estas instrucciones son ejecutadas cuando la función es invocada dentro de un programa principal.

Poseen la siguiente estructura

```
type name (parámetros)
{
        Instrucción 1;
        Instrucción 2;
}
```

El tipo de dato es el valor que devolverá la función a un programa principal, por ejemplo 'int' se utilizará cuando la función devuelve un dato numérico de tipo entero.

Si la función no devuelve ningún valor entonces se colocará delante la palabra “void”, que significa “vacío”.

Después se escribe el nombre de la función o etiqueta con el que se identifica y con el cual será invocada dentro del programa principal.

Entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función. Un parámetro es información que se brinda a la función para que pueda ser procesada.

Los parámetros son declarados con el tipo de dato al que pertenecen y son separados con comas.

Todas las instrucciones pertenecientes a la función son declaradas dentro de dos corchetes que delimitan su alcance.

Si la función retorna un valor al programa principal, se utiliza la palabra clave return.

```
int suma (int a, int b)
{
        return a+b;
}
```

Sintaxis básica de un programa en Arduino La estructura básica de un programa en Arduino es bastante simple y se compone de un mínimo de dos partes. La de configuraciones y el programa principal.

La primera función es void setup() es la parte encargada de las configuraciones. Es la primera función a ejecutar al correr el programa en el microcontrolador, se ejecuta sólo una vez, y se utiliza para configurar o inicializar los pines en un modo de trabajo específico, de entrada o salida, además de la configuración de la comunicación y otras.

```
void setup ()
{
        Configuración 1;
        Instrucción 1;
}
```

La segunda es void loop(), como mencionamos antes la repetición constante de un grupo de instrucciones es fundamental en la programación en microcontroladores. void loop() contiene las instrucciones que se ejecutaran continuamente como, lectura de entradas, señales de salida, etc. Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte de las tareas.

```
void loop ()
{
        Instrucción 2;
        Instrucción 3;
}
```

Sintaxis

Algunos elementos de sintaxis a tener en cuenta:

Llaves

Las llaves sirven para definir el inicio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), funciones de usuario, etc.

```
type function()
{
        instrucciones;
}
```

Punto y coma

El punto y coma ";" se utiliza para separar instrucciones y así poder ser interpretadas y compiladas. Muchas veces olvidamos de colocarlos y esto nos dará error al momento de compilar el programa.

```
int x = 13;
```

Las salidas son señales que provienen de un centro de procesamiento. Como por ejemplo un monitor o un parlante son salidas o outputs de un ordenador. Arduino utiliza como salidas a los actuadores.

Los actuadores convierten las señales de corriente o voltaje en señales físicamente útiles como:

- La Luz de un led
- Los mensajes en un display
- El giro y movimiento de un motor
- y los sonidos en un parlante.

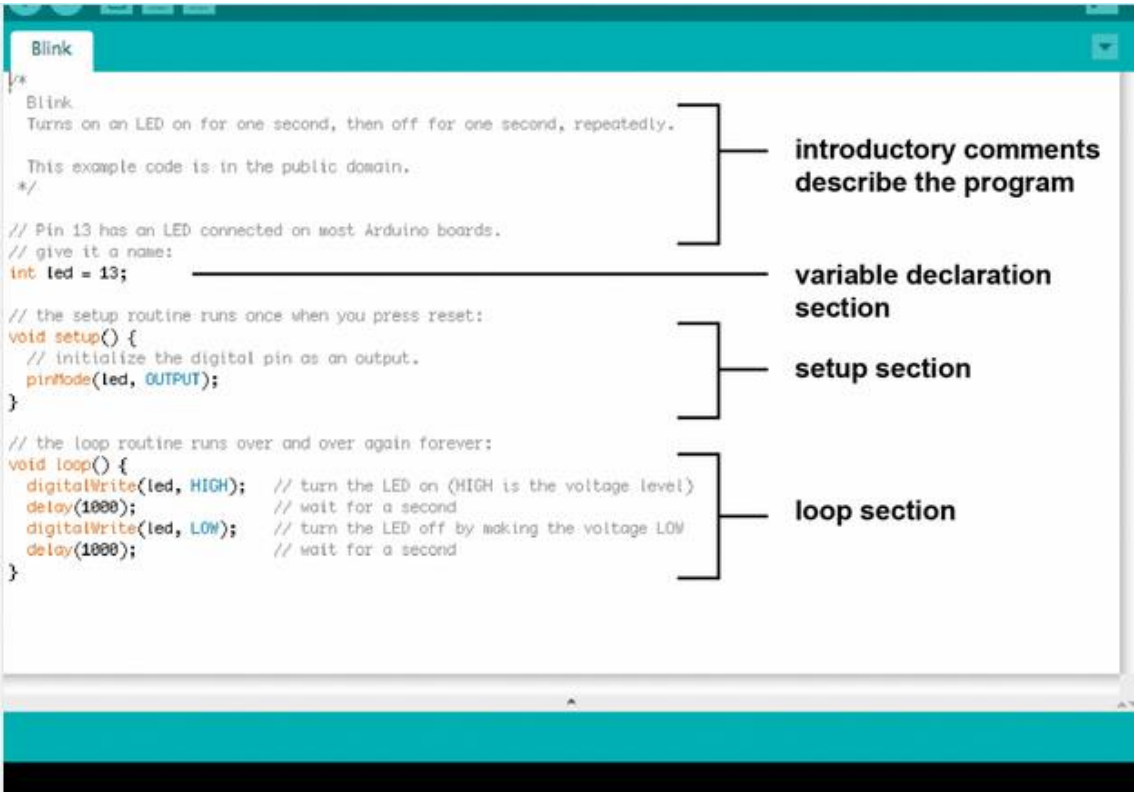
3Una línea de comentario empieza con // y terminan con la siguiente línea de código, las línea de comentarios son ignoradas por el programa y no ocupan espacio en la memoria.

```
// Esto es un comentario
```

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace ésta o para recordarla más adelante.

Con esto hemos finalizado el presente artículo, los espero en el siguiente para aprender más de Arduino.

Estructura de un sketch en Arduino, introducción a la programación

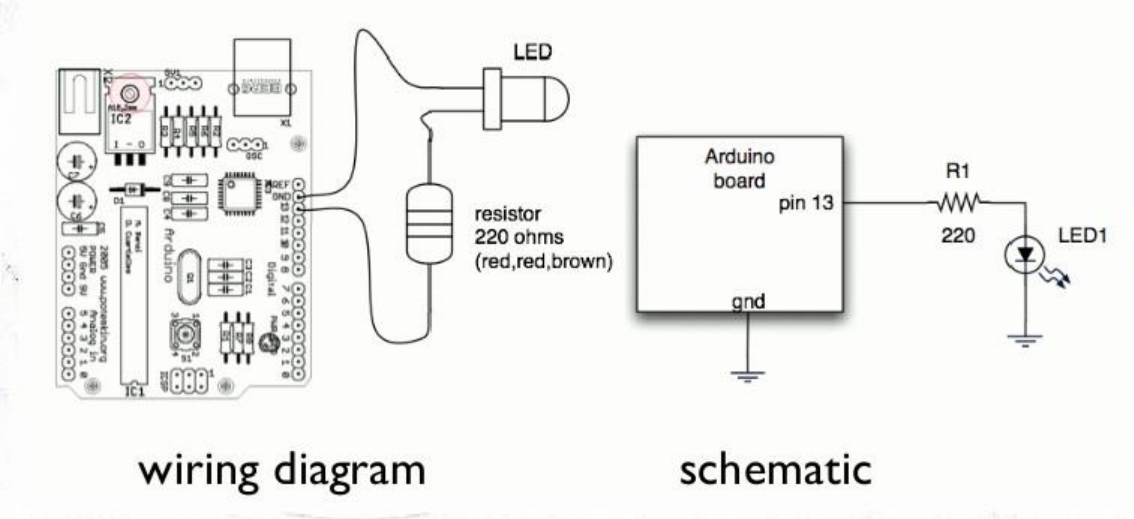


EJERCICIOS DE ARDUINO.

Led parpadeante

Se trata de conectar un led al pin13, haciendo que luzca durante 500 ms y que se apague durante 100 ms, este
proceso se repetirá cíclicamente.
Objetivos:

- Reconocer partes de la placa.
- Aprender a conexionar leds a la placa.
- Familiarizarse con el entorno de programación.
- Reconocer las partes de un programa de arduino.
- Conocer órdenes como: pinMode, digitalWrite y delay.




```
void setup() { //comienza la configuracion
pinMode(13, OUTPUT); //configura el pin 13 como de salida
} //termina la configuracion
void loop() { //comienza el bucle principal del programa
digitalWrite(13, HIGH); //envia 5V al pin (salida) 13
delay (500); //espera 500 ms pin 13 con 5V

digitalWrite(13, LOW); //envia 0V al pin (salida) 13
delay (100); //espera 100 ms pin 13 con 0V
}
```

Secuencia de leds.

Se trata de encender y apagar 4 leds secuencialmente. Los leds deben estar conectados a los pines 5,6,7 y 8.
Se deben encender y posteriormente apagar los leds desde el pin 5 al 8, con un tiempo de duración de
encendido y apagado de 200 milisegundos.
Nota: en una segunda solución la secuencia principal del programa debe estar reproducida en una función a la
que llamará el programa principal.
Objetivos:

- Familiarizarse con el entorno de programación.
- Aprender a declarar variables y variables tipo lista de valores.
- Aprender a declarar una función y llamarla cuando sea necesario.

Solución 1:

```
int tiempo=200; //declara una variable como entero y de
valor 200
void setup() { //comienza la configuracion
pinMode(5,OUTPUT);
pinMode(6,OUTPUT);
pinMode(7,OUTPUT);
pinMode(8,OUTPUT);
}
void loop() { //comienza el bucle principal del programa
digitalWrite(5,HIGH);
delay(tiempo);
digitalWrite(5,LOW);
delay(tiempo);
digitalWrite(6,HIGH);
delay(tiempo);
digitalWrite(6,LOW);
delay(tiempo);
digitalWrite(7,HIGH);
delay(tiempo);

digitalWrite(7,LOW);
delay(tiempo);
digitalWrite(8,HIGH);
delay(tiempo);
digitalWrite(8,LOW);
delay(tiempo);
}
```

Solución 2:

```
int tiempo=200;
int n;
void setup() { //comienza la configuracion
for (n=5;n<9;n++) {
pinMode (n, OUTPUT);
}
}
void secuencia() {
for (n=5;n<9;n++) {
digitalWrite (n, HIGH);
delay (tiempo);
digitalWrite (n, LOW);
delay (tiempo);
}
}
void loop() {
secuencia();
}
```

Solución 3:

```
int leds[]={5,6,7,8};// Declara variables tipo lista de valores
int tiempo=200;
int n=0;
void setup() { //comienza la configuracion
for (n=0;n<4;n++) {
pinMode (leds[n], OUTPUT);
}
}
void secuencia() {
for (n=0;n<4;n++) {
digitalWrite (leds[n], HIGH);
delay (tiempo);
digitalWrite (leds[n], LOW);
delay (tiempo);
}
}
void loop() {
secuencia();
}
```

Cruce de semáforos.

Se trata de un cruce de semáforos controlado por arduino, para ello utilizaremos en el primer semáforo los pines 3 (led rojo), 4 (led ambar), 5 (led verde), en el segundo semáforo utilizaremos los pines 6 (led rojo), 7 (led ambar) y 8 (led verde). La secuencia de funcionamiento debe ser : rojo 1 – verde 2 durante 3 segundos, rojo 1 – ambar 2 durante 500 ms, verde 1 – rojo 2 durante 3 segundos, ambar 1 - , rojo 2 durante 500 ms.

Objetivos:

- Familiarizarse con el entorno de programación.
- Aprender a declarar variables tipo lista de valores.

Solución:

```
int leds[]={3,4,5,6,7,8);
int tiempo1=3000;
int tiempo2=500;
int n;

void setup() {
for (n=0;n<6;n++) {
pinMode (leds[n],OUTPUT);
}
}

void loop () {
digitalWrite (leds[0],HIGH);
digitalWrite (leds[5],HIGH);
delay (tiempo1);
digitalWrite (leds[5],LOW);
digitalWrite (leds[4],HIGH);
delay (tiempo2);
difitalWrite[leds[0],LOW);
digitalWrite (leds[2],HIGH);
digitalWrite (leds[4],LOW);
digitalWrite (leds[3],HIGH);
delay (tiempo1);
digitalWrite (leds[2],LOW);
digitalWrite(leds[1],HIGH);
delay (tiempo2);
}
```

SOS con zumbador.

Se trata de un zumbador que en código morse (pitidos largos/cortos) especifica una palabra, en nuestro caso SOS. Para el que no lo sepa, la S son tres señales acústicas de corta duración y la O tres señales acústica de
larga duración.
El zumbador debe estar conectado al pin 13, los pitidos cortos tendrán una duración de 100 ms y los largos
300 ms. Entre letra y letra debe pasar un tiempo de 300 ms y entre SOSs debe haber un tiempo de 1000 ms.
Nota: Debes usar variables para guardar los tiempos que vas a usar.
Objetivos:

- Reconocer partes de la placa.
- Aprender a conexionar un zumbador a la placa.
- Familiarizarse con el entorno de programación.
- Reconocer las partes de un programa de arduino.
- Aprender a como declarar variables.
- Conocer órdenes de control de programa como: for.

Solución1:

```
int corto=100; //Declara la variable de argumento entero "corto"
y la inicializa con el valor 100 (letra S)
int pausa=300;//tiempo entre letra y letra
int largo=300; //variable de argumento entero "largo" y la
inicializa con el valor 300 (letra O)
int espera=1000;//variable argumento entero "espera" y la
inicializa con el valor 1000 (tiempo entre SOS -
SOS)
int n=0;
int zumb=13; //PIN digital al que conectamos el zumbador
void setup(){ //comienza la configuracion
pinMode(zumb,OUTPUT);
}
void loop(){
for(n=0;n<3;n++){ //Iteracion en la que la variable n comienza
con el valor 0
digitalWrite(zumb, HIGH); // y va aumentando en 1 en cada ciclo
hasta que toma el valor 2,
delay(corto); // con lo que las instrucciones comprendidas entre
los corchetes
digitalWrite(zumb,LOW); // se repiten 3 veces
delay(corto);
}
delay(pausa); //Tiempo entre letras
for(n=0;n<3;n++){ //Aqui esta la O
digitalWrite(zumb, HIGH);
delay(largo);
digitalWrite(zumb,LOW);
delay(largo);
}
delay(pausa);
for(n=0;n<3;n++){
```

```
digitalWrite(zumb, HIGH);
delay(corto);
digitalWrite(zumb,LOW);
delay(corto);
}
delay(espera); //Tiempo hasta repetir SOS de nuevo
}
```

Solución 2:

```
int tcorto=100;
int tlargo=300;
int pausa=300;
int espera=1000;
int n=0;
void setup(){ //comienza la configuracion
pinMode(13,OUTPUT);
}
void s(){ //comienza el bucle para la letra S
  for(n=0;n<3;n++) {
    digitalWrite (13,HIGH);
    delay (tcorto);
    digitalWrite (13,LOW);
    delay (tcorto);
}
}
void o(){ //comienza el bucle para la letra O
  for(n=0;n<3;n++) {
    digitalWrite (13,HIGH);
    delay (tlargo);
    digitalWrite (13,LOW);
    delay (tlargo);
}
}
void loop(){ //se ejecuta el bucle principal en el orden
siguiente
  s();
  delay(pausa);
  o();
  delay(pausa);
  s();
  delay(espera);
}
```

Coche Fantástico.

Se trata de encender y apagar 7 leds secuencialmente. Los leds deben estar conectados a los pines 5,6,7,8,9,10 y 11.

Se deben encender y apagar los leds desde el pin 5 al 11, con un tiempo de encendido y apagado de 50 ms, más tarde se deben encender y apagar los leds desde el pin 11 al 5, con un tiempo de encendido y apagado de 50 ms. La secuencia se debe repetir indefinidamente. El efecto del programa es el de las luces delanteras de nuestro querido "Coche fantástico".
Objetivos:
• Familiarizarse con el entorno de programación.
• Repasar declaración de variables tipo lista de valores.
• Repasar órdenes de control de programa como: for.

Solución1:

```
int leds[]={5,6,7,8,9,10,11};
int n=0;
int tiempo=50;
void setup() { //comienza la configuración
  for (n=0;n<7;n++) {
    pinMode(leds[n],OUTPUT);
  }
}
void loop() {
    for (n=0;n<7;n++) {
    digitalWrite (leds[n],HIGH);
    delay(tiempo);
    digitalWrite (leds[n],LOW);
    delay(tiempo);
  }
  for (n=6;n>=0;n--) {
    digitalWrite (leds[n],HIGH);
    delay(tiempo);
    digitalWrite (leds[n],LOW);
    delay(tiempo);
  }
}
```

Solución2 (sin variable de listas de valores (array)):

```
int n=0;
int tiempo=50;
void setup() { //comienza la configuración
  for (n=5;n<12;n++) {
    pinMode(n,OUTPUT);
  }
}
```

```
void loop() {
  for (n=5;n<12;n++) {
    digitalWrite (n,HIGH);
    delay(tiempo);
    digitalWrite (n,LOW);
    delay(tiempo);
```

```
  }
  for (n=11;n>=5;n--) {
    digitalWrite (n,HIGH);
    delay(tiempo);
    digitalWrite (n,LOW);
    delay(tiempo);
  }
}
```

Solución 3 (Mejorando el efecto visual):

```
int leds[]={5,6,7,8,9,10,11};
int n=0;
int tiempo=30;
void setup() { //comienza la configuración
  for (n=0;n<7;n++) {
    pinMode(leds[n],OUTPUT);
  }
}
void loop() {
    for (n=0;n<7;n++) {
    digitalWrite (leds[n],HIGH);
    delay(tiempo);
    digitalWrite(leds[n+1],HIGH);
    delay(tiempo);
    digitalWrite (leds[n],LOW);
    delay(tiempo*2);
  }
  for (n=6;n>=0;n--) {
    digitalWrite (leds[n],HIGH);
    delay(tiempo);
    digitalWrite(leds[n-1],HIGH);
    delay(tiempo);
    digitalWrite (leds[n],LOW);
    delay(tiempo*2);
  }
}
```

Secuencia de leds con pulsador.

Se trata de encender y apagar 4 leds secuencialmente al accionar un pulsador. El pulsador debe estar conectado al pin 4, y los leds a los pines 5,6,7 y 8.
Se deben encender y posteriormente apagar los leds desde el pin 5 al 8, con un tiempo de duración de encendido y apagado de 200 milisegundos.
Nota: la secuencia principal del programa debe estar reproducida en una función a la que llamará el programa principal.

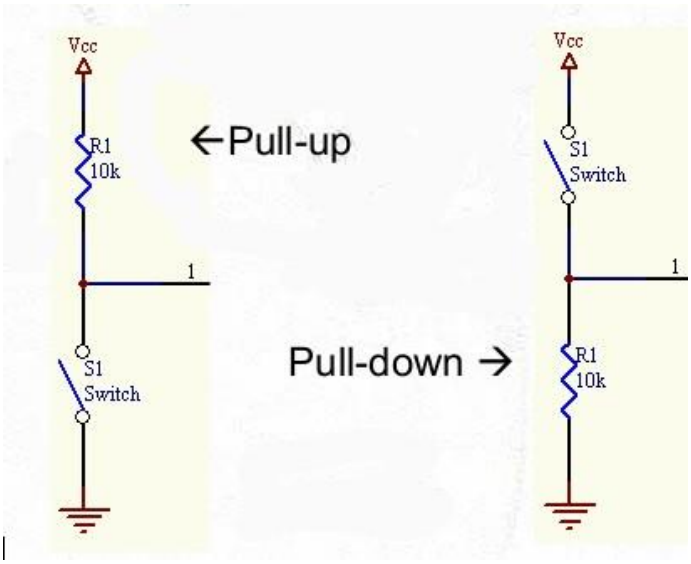
Objetivos:

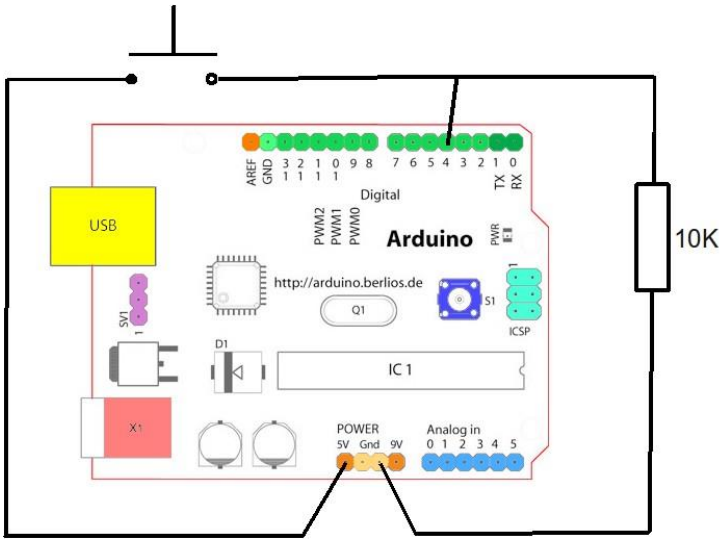
- Familiarizarse con el entorno de programación.
- Aprender a conectar una entrada digital a arduino (pulsador).
- Aprender a declarar variables tipo lista de valores.

- Aprender a declarar una función y llamarla cuando sea• Conocer órdenes como: digitalRead.

- Conocer órdenes de control de programa como: If.

Solución:





Solución 1:

```
int cadenaleds[]={5,6,7,8};
int pulsador=4;
```

```
int tiempo=200;
int n=0;
void setup() {
for(n=0;n<4;n++) {
pinMode (cadenaleds[n],OUTPUT);
}
pinMode (pulsador,INPUT);
}
void flash() {
for (n=0;n<4;n++) {
digitalWrite (cadenaleds[n],HIGH);
delay (tiempo);
digitalWrite (cadenaleds[n],LOW);
delay (tiempo);
}
}
void loop() {
if (digitalRead(pulsador)==HIGH) {
flash ();
}
}
```

Solución 2:

```
int leds[]={5,6,7,8};
int tiempo=200;
int pulsador=4;
int n=0;
int valorpulsador=0;

void setup(){
  for(n=0;n<4;n++){
    pinMode(leds[n],OUTPUT);
  }
  pinMode(pulsador,INPUT);
  Serial.begin(9600);
}
void monitoriza(){
  Serial.print("El valor del pulsador es ...");
  Serial.println(valorpulsador);
  delay(1000);
}
void secuencia(){
  for(n=0;n<4;n++){
    digitalWrite(leds[n],HIGH);
    delay(tiempo);
```

```
    digitalWrite(leds[n],LOW);
    delay(tiempo);
  }
}
void loop(){
  valorpulsador=digitalRead(pulsador);
  monitoriza();
  if (valorpulsador==1){
    secuencia();
  }
}
```

Ruleta de la fortuna.

Se trata de cinco leds que se van encendiendo y apagando formando una secuencia, el jugador debe dar al pulsador cuando el led intermedio se enciende, si acierta funciona un zumbador y la velocidad de la secuencia aumenta.
Los leds deben estar conectados de los pines 5 a 9 (inclusives), el zumbador al pin 10, el pulsador al pin 11.
El tiempo inicial entre encendido y encendido de leds debe ser 200 ms, si se acierta se decrementa el tiempo en 20 ms, si el tiempo entre encendidos llegase a 10 ms, se devuelve el tiempo a 200 ms.
Objetivos:

- Repaso de conexión de entrada digital a arduino (pulsador).
- Repaso de variables tipo lista de valores.
- Repaso de declarar una función y llamarla cuando sea necesario.
- Repaso de órdenes como: digitalRead.
- Repaso de órdenes de control de programa como: For, If.

Solución:

```
int leds[]={5,6,7,8,9};
int n=0;
int tiempo=200;
int zumbador=10;
int pulsador=11;
void setup (){
```

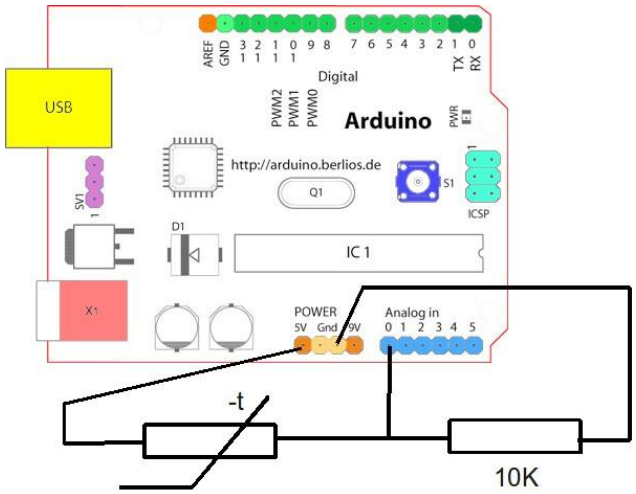
```
for(n=0;n<5;n++) {
pinMode(leds[n],OUTPUT);
}
pinMode(zumbador,OUTPUT);
pinMode(pulsador,INPUT);
}
void compruebaacierto(){
if(digitalRead(pulsador)==HIGH && n==2) {
digitalWrite(zumbador,HIGH);
delay (1000);
digitalWrite(zumbador,LOW);
tiempo=tiempo-20;
if(tiempo<10){
tiempo=200;}
}}
void loop () {
for(n=0;n<5;n++) {
digitalWrite(leds[n],HIGH);
delay(tiempo);
compruebaacierto();
digitalWrite(leds[n],LOW);
delay(tiempo);
}
```

Termostato.

Se trata de un dispositivo que haga funcionar un motor y un led cuando la temperatura supera cierto umbral.
Para ello conectaremos una ntc a la entrada analógica 0, un led al pin 5 y un motor de corriente continua al pin 10. Cuando la temperatura llegue a cierto umbral de voltaje (entre 0 y 1024) que nosotros decidamos, se conectarán a la vez el diodo led y el motor que puede tener unas aspas de ventilador en su eje para enfriar la ntc. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en una consola en el PC.

Objetivos:
• Conexión de entrada analógica a arduino (ntc).
• Órdenes como: analogRead.
• Visualizar datos en consola de puerto serie, con órdenes como: Serial.begin, Serial.print.
• Repaso de órdenes de control de programa como: If else.

Solución:




```
int led=5;
int ntc=0;
int motor=10;
int medida=0;
int nivel=700; //variable que guarda el límite de
temperatura al que se activa el ventilador
void setup(){
pinMode(led,OUTPUT);
pinMode(motor,OUTPUT);
Serial.begin(9600);
}
void monitoriza(){ //procedimiento que envía al puerto
serie, para ser leído en el monitor,
Serial.print("La medida es ...");
Serial.println(medida);
Serial.print();
delay(1000); //para evitar saturar el puerto serie
}
void loop(){
medida=analogRead(ntc);
monitoriza();
if(medida>nivel){ //si la señal del sensor supera el nivel
marcado:
digitalWrite(led,HIGH); //se enciende un aviso luminoso
digitalWrite(motor,HIGH); //arranca el motor
}
else{ // si la señal está por debajo del nivel marcado
digitalWrite(led,LOW);
digitalWrite(motor,LOW); // el motor se para
```

Aumentar y disminuir intensidad luminosa de led (fading).

Se trata aumentar y disminuir la luminosidad de un led usando la capacidad de ofrecer una tensión variable que da una salida analógica. Para ello se conecta un led al pin 11 y se provoca que su luminosidad pase de mínima a máxima, para luego ir de máxima a mínima. Los valores de salidas analógicas van del mínimo 0 al máximo 255.

Objetivos:
• Conexionado de salidas analógicas (power with module pwm).
• Conocer órdenes como analogWrite.

Solución:

```
int luminosidad = 0;  // variable para asignar la
luminosidad al led
int led = 11; // pin del led
void setup()
{
  // en el setup no hay que configurar nada
}
void loop()
{
  for (luminosidad = 0 ; luminosidad <= 255;
luminosidad=luminosidad+3) // fade in (from min to max)
  {
    analogWrite(led, luminosidad);  // ilumina el led con
el valor asignado a luminosidad (entre 0 y 255)
    delay(30); // espera 30 ms para que se vea el efecto
«Ejercicios de Arduino resueltos», Grupo Sabika 16
  }
  for (luminosidad = 255; luminosidad >=0;
luminosidad=luminosidad-3)    // fade out (from max to min)
  {
    analogWrite(led, luminosidad);
    delay(30);
  }
}
```

Luz de led en función de la luz.

Se trata de un dispositivo que haga lucir un led más o menos en función de la luz externa. Para ello conectaremos una ldr a la entrada analógica 0 y un led al pin 9. Cuando la luz se encuentre entre 0 y 512 el led debe colocarse en el nivel de potencia máxima (255), si la luz se encuentra entre valores 512 y 1024 el debe lucir al nivel de potencia 64. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en una consola en el PC.

Objetivos:
• Repaso conexión de entrada analógica a arduino (ldr).
• Conexionado de salidas analógicas.
• Órdenes como: analogWrite.
• Repaso de visualizar datos en consola de puerto serie, con órdenes como: Serial.begin, Serial.print.
• Repaso de órdenes de control de programa como: If else.

```
int led=9;
int ldr=0;
int luz=0;
void setup(){
pinMode(9,OUTPUT)Serial.begin(9600);
}

void monitoriza(){
Serial.print("El valor de luz es ...");
Serial.println(luz);
delay(1000);
}
void loop(){
luz=analogRead(ldr);
monitoriza();
if(luz<512 && luz>=0){
analogWrite(led,255);
}
if(luz>=512 && luz<=1024) {
analogWrite(led,64);
}
}
```

Luz de leds en función de la luz. Versión 2.

Se trata de un dispositivo que haga lucir tres leds más o menos en función de la luz externa. Para ello conectaremos una ldr a la entrada analógica 0 y los leds a los pines 9,10 y 11. Cuando la luz se encuentre entre 768 y 1023 los leds debe colocarse en el nivel de potencia 64, si la luz se encuentra entre valores 512 y 767 los leds deben lucir al nivel de potencia 127, si la luz se encuentra entre valores 256 y 511 los leds deben lucir al nivel de potencia 191, si la luz se encuentra entre valores 0 y 255 los leds deben lucir al nivel de potencia 255. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en una consola en el PC.

Objetivos:
• Repaso conexión de entrada analógica a arduino (ldr).
• Repaso conexionado de salidas analógicas.
• Repaso órdenes como: analogWrite.
• Repaso de visualizar datos en consola de puerto serie, con órdenes como: Serial.begin, Serial.print.
• Repaso de órdenes de control de programa como: If else.

Solución:

```
int leds[]={9,10,11};
int tiempo=300;
int ldr=0;
int n=0;
int luz=0;

void setup(){
for(n=0;n=2;n++) {
pinMode(leds[n],OUTPUT);
}
Serial.begin(9600);
}
void monitoriza() {
Serial.print("El valor de la luz es ...");
Serial.println(luz);
delay(1000);
}
void loop(){
luz=analogRead(ldr);
monitoriza();
if (luz<=1023 && luz>=768) {
for (n=0;n=2;n++) {
analogWrite(leds[n],64);
delay(tiempo);
}
}
if (luz<=767 && luz>=512) {
for (n=0;n=2;n++) {
analogWrite(leds[n],127);
delay(tiempo);
}
}
if (luz<=511 && luz>=256) {
for (n=0;n=2;n++) {
analogWrite(leds[n],191);
delay(tiempo);
}
}
if (luz<=255 && luz>=0) {
for (n=0;n=2;n++) {
analogWrite(leds[n],255);
delay(tiempo);
}
}
}
```

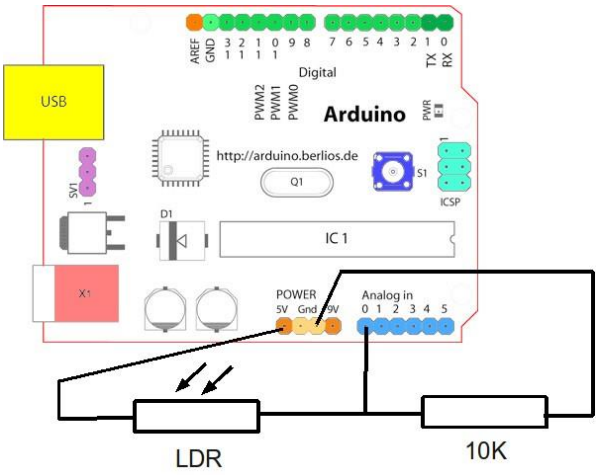
Luz de leds en función de la luz. Versión 3.

Se trata de un dispositivo que haga lucir tres leds más o menos en función de la luz externa. Para ello conectaremos una ldr a la entrada analógica 0 y los leds a los pines 9,10 y 11. El valor de la entrada analógica 0 está comprendido entre 0 y 1024, y el valor de la luminosidad de los leds entre 0 y 255. Los ledsdeben lucir entre 0 y 255 en función del valor de la entrada analógica 0, siendo su valor inversamente

proporcional al valor de la entrada analógica 0 (de 0 a 1024), o sea a más luz menor intensidad luminosa de los leds.

Objetivos:

- Repaso conexión de entrada analógica a arduino (ldr).
- Repaso conexionado de salidas analógicas.
- Repaso órdenes como: analogWrite.
- Repaso de visualizar datos en consola de puerto serie, con órdenes como:Serial.begin, Serial.print.
- Repaso de órdenes de control de programa como: If else.




```
int ldr=0;
int leds[]={9,10,11};
int n=0;
int medida=0;
int luzled=0;
void setup(){
  for (n=0;n<3;n++) {
    pinMode(leds[n],OUTPUT);
  }
 Serial.begin(9600);
}

void monitoriza(){
  Serial.print("La medida de luz es ...");
  Serial.println(medida);
  Serial.print("La luz a dar en los leds es ...");
  Serial.println(luzled);
  delay(1000);
}
void loop(){
medida=analogRead(ldr);
luzled=255-(medida/4);
monitoriza();
for (n=0;n<3;n++){
    analogWrite(leds[n],luzled);
```

```
    delay(200);
  }
}
```

Termostato con velocidad de motor variable.

Se trata de diseñar un dispositivo que haga lucir un led y funcionar el motor de un ventilador cuando la temperatura llegue a cierto valor umbral (entre 0 y 1024). Para ello conectaremos una ntc a la entrada analógica 0, el led al pin 13 y el motor al pin 9. El motor debe funcionar a cierto nivel de potencia a elegir entre 0 y 255. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en una consola en el PC.

Objetivos:

- Repaso conexión de entrada analógica a arduino (ntc).
- Repaso conexionado de salidas analógicas.
- Repaso órdenes como: analogWrite.
- Repaso de visualizar datos en consola de puerto serie, con órdenes como: Serial.begin, Serial.print.
- Repaso de órdenes de control de programa como: If else.

Solución:

```
int motor=9;
int led=13;
int ntc=0;
int temperatura=0;
void setup(){
pinMode(led,OUTPUT);
pinMode(motor,OUTPUT);
Serial.begin(9600);
}
void monitoriza(){
Serial.print("El valor de temperatura es
..."Serial.println(temperatura);
delay(1000);
}
void loop(){
temperatura=analogRead(ntc);
monitoriza();
if(temperatura>530){
digitalWrite(led,HIGH);

analogWrite(motor,200);
}
else {
digitalWrite(led,LOW);
digitalWrite(motor,LOW);
}
```

```
}
```

Termostato con velocidad de motor variable (Versión 2).

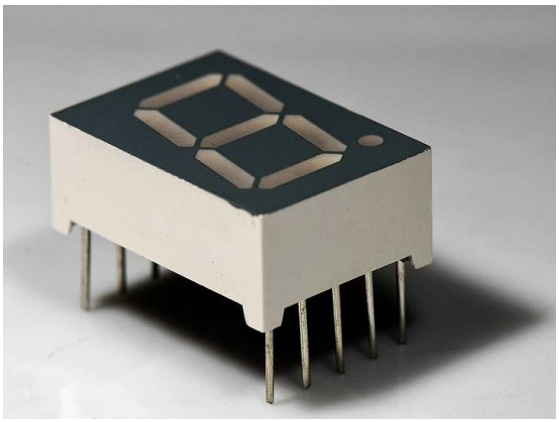
Se trata de un circuito que haga girar un motor más o menos rápido en función de la temperatura. Para ello conectaremos una NTC a la entrada analógica 0 y un led al pin 9 y el motor al pin 10. El valor de la entrada analógica 0 está comprendido entre 0 y 1024, y el valor del la tensión del pin 10 entre 0 y 5 voltios
(entre 0 y 255). El motor debe girar a una velocidad entre 0 y 255 en función del valor de la entrada analógica 0, siendo su valor directamente proporcional al valor de la entrada analógica 0 (de 0 a 1024), o sea a más temperatura más velocidad del motor. Además el led del pin 9 debe encenderse.

Objetivos:
• Repaso conexión de entrada analógica a arduino (ntc).
• Repaso conexionado de salidas analógicas.
• Repaso órdenes como: analogWrite.
• Repaso de visualizar datos en consola de puerto serie, con órdenes como:
Serial.begin, Serial.print.
• Repaso de órdenes de control de programa como: If else.
Solución:

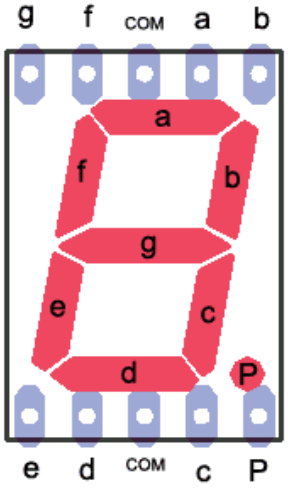
Display de 7 segmentos (1 dígito)

Introducción teórica

Un display de segmentos (o visualizador) es un componente electrónico que se utiliza para representar números. Como su propio nombre indica y, como se puede observar en la imagen siguiente, el display está compuesto por 7 segmentos, los cuales se encenderán y/o apagarán en función del número a representar. De forma interna, se asemeja a siete LEDs conectados estratégicamente formando el número 8, aunque externamente dicha semejanza no se observa, de ahí su simplicidad.



Cada uno de los segmentos que componen este display se denominan **a, b, c, d, e, f** y **g,** tal y como se muestra a continuación.

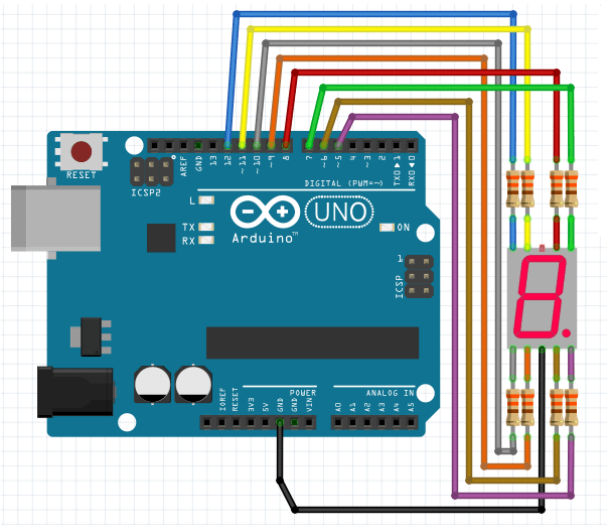


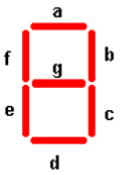
Para mostrar el número 0, tendremos que encender a, b, c, d, e y f.
Para el número 2, tendríamos a, b, g, e y d.
Y de la misma forma para cualquier otro número.
El **P** simboliza el punto decimal.
En cuanto a la clasificación, existen de dos tipos:
1. Display de segmentos de **cátodo común**, en la que todos los cátodos de los LEDs estan internamente unidos a una patilla común, la cual está conectada al potencial negativo de la placa.
2. Display de segmentos de **ánodo común**, en la que todos los ánodos se encuentran al positivo.

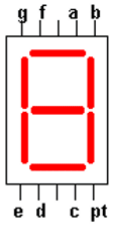
Para nuestro caso, disponemos de uno de tipo cátodo común, el **LMS5161AS** (luz roja, tamaño de dígito 0.56'').

Conexiones

En lo que se refiere a las conexiones, tenemos que tener en cuenta cada segmento a qué pin lo vamos a conectar, para poder efectuar una llamada a los pines correcta. En nuestro caso, hemos hecho las siguientes conexiones (puede variar la designación según convenga):







Led	Arduino
a	8
b	7
c	6
d	9
e	10
f	11
g	12
h(pt)	5

Ejemplo

Vamos a realizar un ejemplo en el cual nuestro display irá mostrando de forma ascendente todos los números con un intervalo de separación de 1 segundo. Aquí tenemos el sketch:

```
int pausa=1000; // Variable que define el intervalo
                // de tiempo entre cada digito

void setup()
{
  pinMode(6, OUTPUT);  // Asignación de las salidas digitales
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
}
```

```
void display (int a, int b, int c, int d, int e, int f, int g)
// Funcion del display
{
  digitalWrite (7,a);   //Se reciben 7 variables y se asignan
  digitalWrite (8,b);   //a cada una de las salidas
  digitalWrite (9,c);
  digitalWrite (10,d);
  digitalWrite (11,e);
  digitalWrite (12,f);
  digitalWrite (13,g);
}

void loop() //Funcion principal
// Dependiendo de cada dígito, se envía a la función display
// los estados (0 y 1) a cada uno de los segmentos
{
  display (1,1,1,1,1,1,0); //escribe 0
  delay(pausa);
  display (0,1,1,0,0,0,0); //escribe 1
  delay(pausa);
```

LCD 16X2 (PARALELO)

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

LCD Liquid Crystal Display

LCD es el acrónimo de Liquid Crystal Display (en español Pantalla de Cristal Líquido). No podemos considerar que se trate de una tecnología novedosa. El LCD lleva con nosotros mucho tiempo, solo tenemos que echar la mirada hacia atrás y recordar esos relojes Casio o las calculadoras que llevamos a clase de matemáticas.

Estamos acostumbrados a que la materia pueda estar en estado **sólido, líquido o gaseoso**, los llamados **estados de la materia**. Pero ya en el siglo XIX se descubrió que había más estados en los que podía encontrarse la materia. El cristal líquido se encuentra en algún lugar entre el sólido y líquido.

Hay una **amplia gama de pantallas LCDs** que podemos utilizar con Arduino. Aparte de las funcionalidades extra que nos puedan dar cada una de ellas, las podemos diferenciar por el número de filas y columnas, su tamaño.

Por ejemplo, una pantalla LCD de 16×1 tendrá una fila de 16 caracteres es decir, solo podremos mostrar 16 caracteres simultáneamente, al igual que un LCD de 20×4 tendrá 4 filas de 20 caracteres cada una.

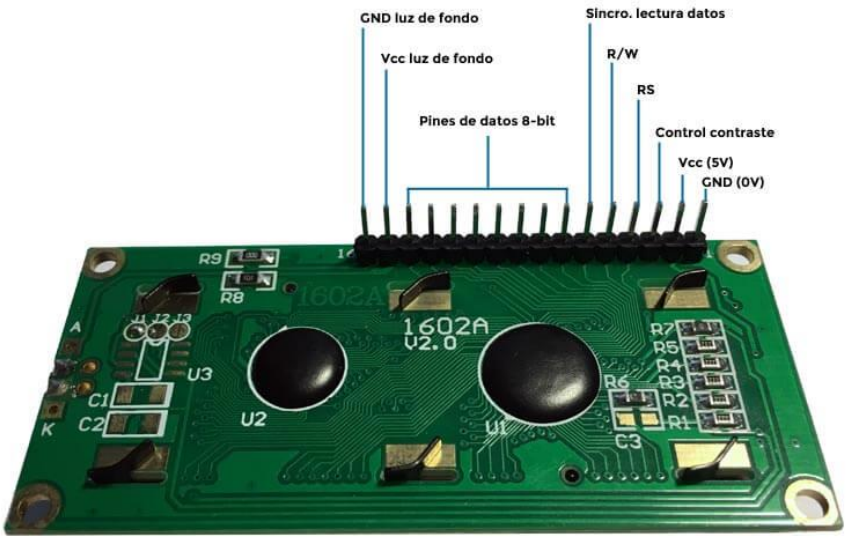
En este caso vamos a trabajar con un **LCD típico, de 16×2**. Esto significa que vamos a poder mostrar **16 caracteres en la primera fila y los mismos en la segunda fila**

Conexiones de un LCD con Arduino

La mayoría de las pantallas LCD que se están haciendo ahora, vienen con una fila de dieciséis pines. Los primeros catorce pines se utilizan para controlar la visualización. Los dos últimos son para la iluminación de fondo.

PIN	FUNCIÓN
1	GND (Tierra)
2	5 Voltios
3	Control de contraste pantalla
4	RS – Selector entre comandos y datos
5	RW – Escritura y lectura de comandos y datos
6	Sincronización de lectura de datos
7-14	Pines de datos de 8-bit
15	Alimentación luz de fondo (5V)
16	GND (Tierra) luz de fondo (0V)

En la siguiente imagen te muestro la correspondencia con los pines físicos del LCD.

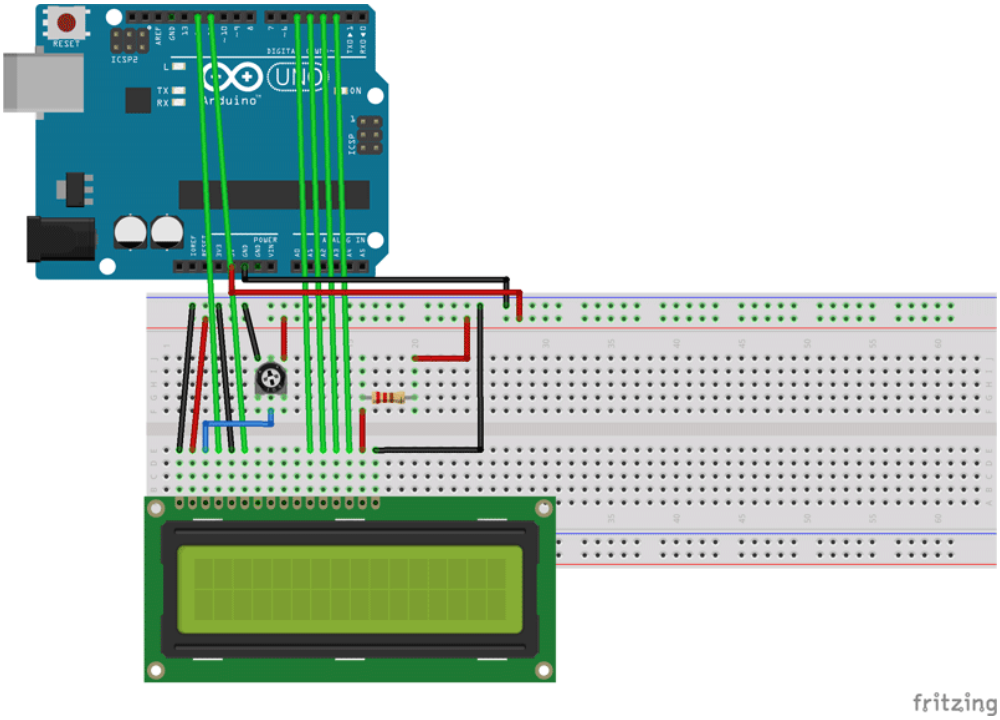


Montando el circuito

Una vez conocemos los conectores, ya podemos montar el circuito básico. El material que vamos a utilizar es el siguiente

- Arduino UNO o similiar
- Protoboard
- Cables
- Pantalla LCD 16X2
- Potenciómetro de 10 kΩ
- Resistencia 200 Ω

En la siguiente imagen te muestro cómo hacer el cableado.



Como puedes comprobar, el circuito es muy sencillo. La **resistencia de 220 Ω** permite **bajar el voltaje a la entrada al led** de la pantalla LCD. El **potenciómetro** se utiliza para **regular el contraste de la pantalla**.

Ejemplo básico LCD con Arduino

Para programar con el código nativo de Arduino, vamos a utilizar una librería que nos hará más fácil el desarrollo. Esta librería es *LiquidCrystal.h*. No hace falta instalarla en el entorno de desarrollo oficial ya que viene por defecto. Lo único que tenemos que hacer es añadirla como un *include* en nuestro programa o *sketch*. Vamos a ver un ejemplo muy simple donde escribamos un texto en el LCD.

En este ejemplo vamos a escribir la frase «Hola Mundo!!!!!!» en la primera fila y «Probando el LCD» en la segunda fila. Vamos a ir viendo el código parte por parte así quedará más claro.

Importar librería, declaraciones de constantes y variables

```
Importamos la librería LiquidCrystal.h


   // Incluímos la libreria externa para poder utilizarla

   #include <LiquidCrystal.h> // Entre los símbolos <> buscará en la

   carpeta de librerías configurada
```

La librería *LyquidCrystal.h* no es exclusiva para un LCD de 16×2. También puede ser utilizada por otro tipo de LCDs con diferente número de filas y columnas. En las siguientes líneas de código vamos a definir dos constantes que nos indiquen las filas y columnas de nuestro LCD.

```
   // Definimos las constantes

   #define COLS 16 // Columnas del LCD

   #define ROWS 2 // Filas del LCD
```

Esto lo hacemos a través del constructor *LyquidCrystal(…)*. Esta función permite diferente número de parámetros que dependerá de la configuración que estemos

utilizando. En la llamada más básica, la que vamos a utilizar, debemos pasar como parámetros el RS, la sincronización, y los 4 pines para enviar datos. La función quedaría de la siguiente manera *LyquidCrystal(RW, Enable, d4, d5, d6, d7)*. Si sustituimos estos datos con los pines de Arduino tenemos lo siguiente.

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

Donde 12 es el pin de Arduino donde hemos conectado el pin RW, el pin 11 es el Enable y el 5, 4, 3 y 2 son los pines correspondientes a los datos.

La palabra lcd es el nombre que le vamos a dar a esta variable (clase). Este nombre puede ser modificado siguiendo las reglas del lenguaje y no utilizando palabras reservadas.

Función setup()

Con esto ya podemos pasar a la función *setup()*. Esta función nos sirve para hacer una configuración inicial en nuestros programas. Lo primero que hacemos es configurar el monitor serie y luego inicializamos la interfaz del LCD con Arduino y especificamos las dimensiones (16 columnas y 2 filas).

```
void setup() {
  // Configuración monitor serie
  Serial.begin(9600);
  // Configuramos las filas y las columnas del LCD en este caso 16
columnas y 2 filas
  lcd.begin(COLS, ROWS);
}
```

Función loop()

Ahora ya podemos empezar a escribir en la pantalla del LCD. Para ello vamos a utilizar tres métodos:

- *clear()*: limpia la pantalla.
- *setCursor(columna, fila)*: sitúa el cursor en una columna y una fila para después escribir a partir de esa posición.
- *print(datos)*: escribe en la posición definida por setCursor(columna, fila) los

datos. Normalmente los datos son una cadena de caracteres (*String*).

setCursor comienza desde 0 es decir, si queremos escribir en la columna 1 fila 1 debemos poner *setCursor(0,0)*. Esta característica de un lenguaje se suele llamar índice cero (del inglés zero index).

Comenzamos la función *loop(…)* donde lo primero que hacemos es limpiar el LCD, situar el cursor en la columna 0 (sería la primera columna) y la fila 0 (sería la primera fila). Por último escribimos el primer texto «Hola Mundo!!!!!!».

```
void loop() {
  // Limpiamos la pantalla
  lcd.clear();

  // Situamos el cursor en la columna 0 fila 0
  lcd.setCursor(0,0);

  // Escribimos Hola Mundo!!!!!!
  lcd.print("Hola Mundo!!!!!!");}
```

Para escribir el segundo texto, tenemos que mover el cursor a la nueva posición. Queremos que se sitúe justo debajo así que movemos a la siguiente fila (fila 1 que corresponde con la segunda fila) y la columna será la 0 (primera columna). Después escribimos el texto. Por último ponemos un delay() (esperar un tiempo) más que nada para que no esté constantemente mandando datos. Esto último no sería necesario.

```
// Situamos el cursor en la columna 0 fila 1
  lcd.setCursor(0,1);

  // Escribimos Probando el LCD.
  lcd.print("Probando el LCD.");

  // Esperamos 2 segundos igual a 2000 milisegundos
  delay(2000);
}
```

Moviendo el texto horizontalmente en un LCD con Arduino

En el siguiente ejemplo vamos a ver cómo podemos mover el texto de forma horizontal. Para realizar este ejemplo es importante tener ciertos conocimientos de programación ya que vamos a utilizar bucles de repetición y funciones de caracteres. El fin es poder desplazar un texto en horizontal de izquierda a derecha en la fila 1 (índice 0) y de derecha a izquierda en la fila 2 (índice 1).

El algoritmo

Antes de empezar a programar, es conveniente plantear el algoritmo. Esto nos ahorrará tiempo a la hora de codificar con el **lenguaje nativo de Arduino**. Desarrollaremos la idea en sucesivos pasos bien estructurados y ordenados. En este punto, no hay que profundizar en cuántas variables o constantes necesito. Lo único que debemos plasmar son la secuencia de pasos.

1. Obtenemos el tamaño del texto
2. Entrada del texto por la derecha primera fila
    1. Bucle recorriendo los caracteres de mayor a menor índice
        1. Obtenemos un trozo de la subcadena desde el índice hasta el final
        2. Limpiamos LCD
        3. Situamos el cursor en la columna 0 fila 0
        4. Mostramos trozo de texto en la pantalla del LCD
        5. Esperamos tiempo
3. Desplazamiento de texto hacia la derecha primera fila
    1. Bucle recorriendo todas las columnas desde la segunda hasta la última más uno
        1. Limpiamos LCD
        2. Situamos el cursor en la columna correspondiente fila 0
        3. Mostramos texto completo en la pantalla del LCD
        4. Esperamos tiempo
4. Desplazamiento de texto hacia la izquierda segunda fila
    1. Bucle recorriendo las columnas desde la última más uno hasta la columna 1
        1. Limpiamos LCD
        2. Situamos el cursor en la columna correspondiente fila 0
        3. Mostramos texto completo en la pantalla del LCD
        4. Esperamos tiempo
5. Salida de texto por la izquierda segunda fila
    1. Bucle recorriendo los caracteres de menor a mayor índice
        1. Obtenemos un trozo de la subcadena desde el índice hasta el final
        2. Limpiamos LCD
        3. Situamos el cursor en la columna 0 fila 0
        4. Mostramos trozo de texto en la pantalla del LCD
        5. Esperamos tiempo

Es un algoritmo un poco complejo que se divide en 4 bloques. Los dos primeros nos

servirán para desplazar el texto hacia la derecha y los dos siguientes para desplazarlo hacia la izquierda.

Código del texto en movimiento con LCD y Arduino

Una vez tenemos claro lo que queremos hacer, pasamos a traducirlo al lenguaje de programación con Arduino. Es importante hacer el paso anterior y, en este punto, te recomiendo que intentes hacerlo tu mismo. Seguramente te equivocarás y no sabrás cómo hacerlo, pero a base de practicar es como se adquiere la experiencia para poder desarrollar tus propios proyectos.

Declaración de constantes y variables

Lo primero es declarar las constantes y variables necesarias. Algunas ya las hemos visto en el ejercicio básico. Para este ejemplo he añadido una constante, VELOCIDAD, que nos permitirá controlar la velocidad de desplazamiento del texto. Además he creado una variable del tipo String que contendrá el texto que queremos desplazar.

```
// Incluímos la libreria externa para poder utilizarla

#include <LiquidCrystal.h> // Entre los símbolos <> buscará en la

carpeta de librerías configurada

// Definimos las constantes

#define COLS 16 // Columnas del LCD

#define ROWS 2 // Filas del LCD

#define VELOCIDAD 300 // Velocidad a la que se mueve el texto


// Lo primero is inicializar la librería indicando los pins de la

interfaz

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


// Textos

String texto_fila = "EET 1 Parana";
```

```
void setup() {

  // Configuración monitor serie

  Serial.begin(9600);


  // Configuramos las filas y las columnas del LCD en este caso 16

columnas y 2 filas

  lcd.begin(COLS, ROWS);

}
```

Lo primero que hacemos es obtener el tamaño del texto es decir, el número de caracteres que tiene nuestro texto. Para ello utilizamos la función *length()* que nos devuelve un número entero indicando su tamaño.

```
// Obtenemos el tamaño del texto
  int tam_texto=texto_fila.length();
```

La función *length()* nos devuelve el número de caracteres de un texto. Si queremos recorrer todos los caracteres empezaremos por 0 y acabaremos en número de caracteres – 1. Característica de los lenguajes índice cero.

En la siguiente parte vamos a mostrar cómo entra el texto por la parte derecha. En esta parte no vamos a trabajar con el LCD ya que siempre se encontrará en la misma posición, columna 0 fila 0. El truco está en ir cogiendo trozos de texto, de más pequeño a más grande, y mostrarlos en el LCD. Esto nos dará la sensación de que se está moviendo hacia la derecha aunque en realidad lo que hacemos es mostrar cada vez un texto diferente.

```
for(int i=tam_texto; i>0 ; i--)
  {
    String texto = texto_fila.substring(i-1);

    // Limpiamos pantalla
    lcd.clear();

    //Situamos el cursor
    lcd.setCursor(0, 0);

    // Escribimos el texto
    lcd.print(texto);

    // Esperamos
    delay(VELOCIDAD);
  }
```

La función *substring(numero)* nos devuelve un trozo del texto. El número indica a partir de qué letra vamos a coger por ejemplo si hacemos «Hola Mundo».substring(2) nos devolverá «la Mundo». Recuerda que es un lenguaje de índice cero, el primer carácter del texto empieza por cero.

Ahora si que toca desplazar el texto. Para ello hacemos un simple bucle que recorra todas las columnas del LCD. Además tenemos que forzar para que la última visualización sea en la columna 17 y así simularemos que el texto ha desaparecido por la derecha.

```
// Desplazamos el texto hacia la derecha

  for(int i=1; i<=16;i++)

  {

    lcd.clear();// Limpiamos pantalla

    lcd.setCursor(i, 0); //Situamos el cursor

    lcd.print(texto_fila); // Escribimos el texto

    delay(VELOCIDAD); // Esperamos

  }
```

Una vez que ha desaparecido por la derecha, lo vamos a mover de derecha a izquierda. Si te fijas en el código, es exactamente igual que en los pasos anteriores. Lo único que cambia es como se recorren los bucles, ahora en sentido contrario. Lo primero es desplazar el texto a través de la función *setCursor()* e indicando en que posición mostrar el texto.

```
Desplazamos el texto hacia la izquierda en la segunda fila
  for(int i=16;i>=1;i--)
  {
    lcd.clear();// Limpiamos pantalla
    lcd.setCursor(i, 1); //Situamos el cursor
    lcd.print(texto_fila);     // Escribimos el texto
    delay(VELOCIDAD); // Esperamos
  }
```

```
  // Mostramos salida del texto por la izquierda
  for(int i=1; i<=tam_texto ; i++)
  {
    String texto = texto_fila.substring(i-1);
    lcd.clear();  // Limpiamos pantalla
    lcd.setCursor(0, 1);   //Situamos el cursor
    lcd.print(texto); // Escribimos el texto
    delay(VELOCIDAD); // Esperamos
  }
```

Código completo

```
// Incluímos la libreria externa para poder utilizarla
#include <LiquidCrystal.h>

//Definimos las constantes
#define COLS 16 // Columnas del LCD
#define ROWS 2 // Filas del LCD
#define VELOCIDAD 300 // Velocidad a la que se mueve el texto

// Inicializar la librería indicando los pins de la interfaz
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

String texto_fila = "EET 1  Parana"; // Textos

void setup() {
    Serial.begin(9600); // Configuración monitor serie

  // Configuramos las filas y las columnas del LCD en este caso 16
columnas y 2 filas
  lcd.begin(COLS, ROWS);
}
void loop() {
   int tam_texto=texto_fila.length();// Obtenemos el tamaño del texto
  // Mostramos entrada texto por la izquierda
  for(int i=tam_texto; i>0 ; i--)
  {
    String texto = texto_fila.substring(i-1);
    lcd.clear();    // Limpiamos pantalla
    lcd.setCursor(0, 0); //Situamos el cursor
    lcd.print(texto); // Escribimos el texto
    delay(VELOCIDAD); // Esperamos
  }

  // Desplazamos el texto hacia la derecha
  for(int i=1; i<=16;i++)
  {
    lcd.clear();// Limpiamos pantalla
    lcd.setCursor(i, 0); //Situamos el cursor
    lcd.print(texto_fila); // Escribimos el texto
    delay(VELOCIDAD); // Esperamos
  }
  // Desplazamos el texto hacia la izquierda en la segunda fila
  for(int i=16;i>=1;i--)
  {
    lcd.clear();// Limpiamos pantalla
    lcd.setCursor(i, 1); //Situamos el cursor
    lcd.print(texto_fila); // Escribimos el texto
    delay(VELOCIDAD); // Esperamos
  }

   for(int i=1; i<=tam_texto ; i++)  // Mostramos texto por izquierda
  {
    String texto = texto_fila.substring(i-1);

    lcd.clear();// Limpiamos pantalla
    lcd.setCursor(0, 1); //Situamos el cursor
    lcd.print(texto);     // Escribimos el texto
    delay(VELOCIDAD);     // Esperamos
  }
}
```

LCD 16x2 (Serie I2C)

EL BUS I2C

A medida que la capacidad integración en un único chip aumentaba, el número de componentes comerciales disponibles, aumentaba exponencialmente. Cada vez era, y es, más sencillo fabricar bloques de construcción electrónicos integrados en un único chip, y pronto el grosor de los catálogos de los fabricantes, engordó peligrosamente.

Era relativamente fácil encontrar esos bloques de construcción pero cuando tu diseño requería usar una docena de esos bloques, ponerlos de acuerdo y conseguir que se comunicaran eficazmente, se convirtió en un problema.

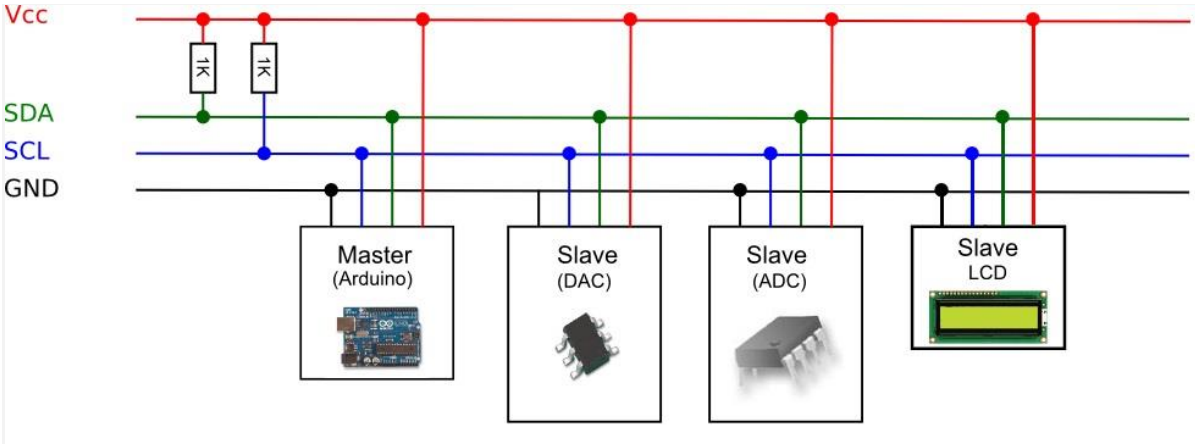
Por eso, en los primeros 80, uno de los grandes fabricantes de electrónica (Phillips), propuso una norma de comunicación digital, entre los diferentes componentes de una sistema electrónico.

Una norma que especificaba la velocidad, niveles de tensión, y el protocolo a seguir para conseguir esa comunicación y la hizo abierta.

Esa norma se llamó Inter Integrated Circuits bus, o IIC, y pronto se convirtió en un estándar de facto en la industria. Las especificaciones han ido mejorando con los años, pero la idea básica sigue siendo la misma:

- Protocolo de dos hilos de control, uno para transmitir los datos, SDA y otro, el reloj asíncrono que indica cuando leer los datos SCL. Mas GND y 5V (cuando se requiera).
- Cada dispositivo conectado al **bus I2C** y cada uno tiene su dirección exclusiva, de 7 bits, (Asi que, en teoría, podemos conectar 2^7 = 128, dispositivos).
- Uno de estos componentes, debe actuar como master, es decir controla el reloj.
- No se requiere una velocidad de reloj estricta, ya que es el master quien controla el Clock.
- Es multi master, el master puede cambiar, pero solo uno puede estar activo a la vez, y proporciona un protocolo de arbitraje y detección de colisiones. (Si no has entendido esto, no te preocupes, todavía es pronto).

Puedes encontrar que a este bus se le llama I2C, IIC o I^2C, y también, como TWI (Two Wire Interface, o interface de 2 hilos), pero siempre es lo mismo.



La idea es que todos los componentes se conecten en paralelo a las dos líneas del Bus, SDA y SCL. En cada momento solo puede haber un master, en este caso, nuestro Duino y los demás se configuran como esclavos.

- Puede haber más de un master. La norma propone un sistema de arbitraje, para transferir el control de uno a otro, pero en un instante dado, solo uno puede ser el master.
- Fijaros también que hay unas resistencias de Pullup conectadas a SDA y SCL. Son imperativas, ya que el bus es activo bajo (Esto es, la señal activa es un 0, no un 1. Pero tranquilo, que esto no te afecta)
- Cuando vayas a conectar algo al busI2C, es imprescindible que leas el manual para saber si los pullups los tienes que poner tú, o vienen puestos en el componente.
- En el caso del display I2C que vamos a usar, normalmente incluyen los pullups.

Y la buena noticia es que nuestro Arduino lo soporta de fábrica con una librería estándar, que utiliza dos de los pines analógicos para las funciones SDA (Datos) y SCL (Clock).

- En el Arduino UNO, los pines I2C están en los pines analógicos A4 (SDA) y A5 (SCL).
- En el Arduino Mega y DUE, son el 20 (SDA) y en el 21(SCL).

La librería I2C, en Arduino se llama Wire (para que no os confiéis), y gestiona el protocolo de comunicaciones completo, lo que es un detalle, pues nos ahorra la parte aburrida de estudiar el protocolo y escribir programas para ello.

- Esto no es vagancia, sino construir sobre el trabajo de terceros. Es una de las muy grandes virtudes de la comunidad Arduino. Muchas librerías para incorporar a nuestros proyectos, sin necesidad de mancharte las manos de grasa.

En esta sesión vamos a conectar un display LCD de 16×2 con interface I2C, y así podréis comprobar porque en la última sesión os recomendé usarlo, en lugar del que se conecta directamente con 16 pines.

Pero antes, tenemos que asegúranos de otro asunto.

SCANNER DE I2C

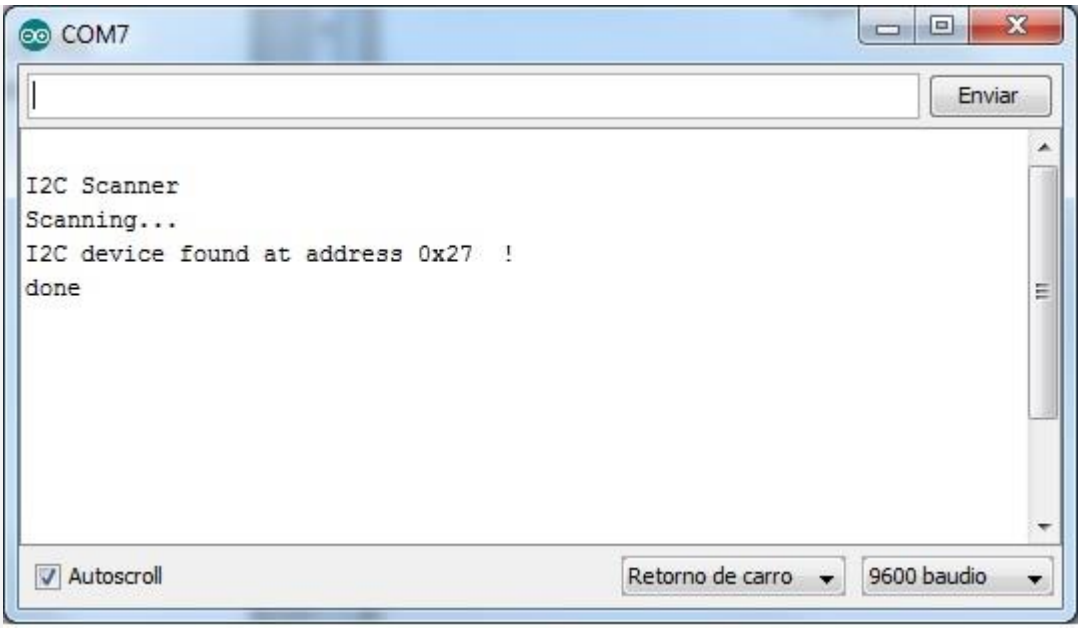
Cada componente que conectamos al **bus I2C** tiene una dirección única, y cada mensaje y orden que transmitimos al bus, lleva anexa esta dirección, indicando cuál de los muchos posibles, es el receptor del mensaje.

Pero, claro, esto implica que sabemos la dirección del componente. Lo normal es comprobar la información técnica del fabricante del componente, y ahí suele decirnos cuál es la dirección por defecto.

Pero como ya sabemos, que en la vida las cosas rara vez son como en los cuentos, algún alma caritativa (y con mucho mundo a sus espaldas), hizo un programita para Arduino, que nos informa, de lo que hay en nuestro bus y con qué dirección. I2C scanner

Naturalmente, este programa, no tiene ni idea de quien responde y lo que hace, pero bastante es que nos informe de que hay alguien en la dirección xx.

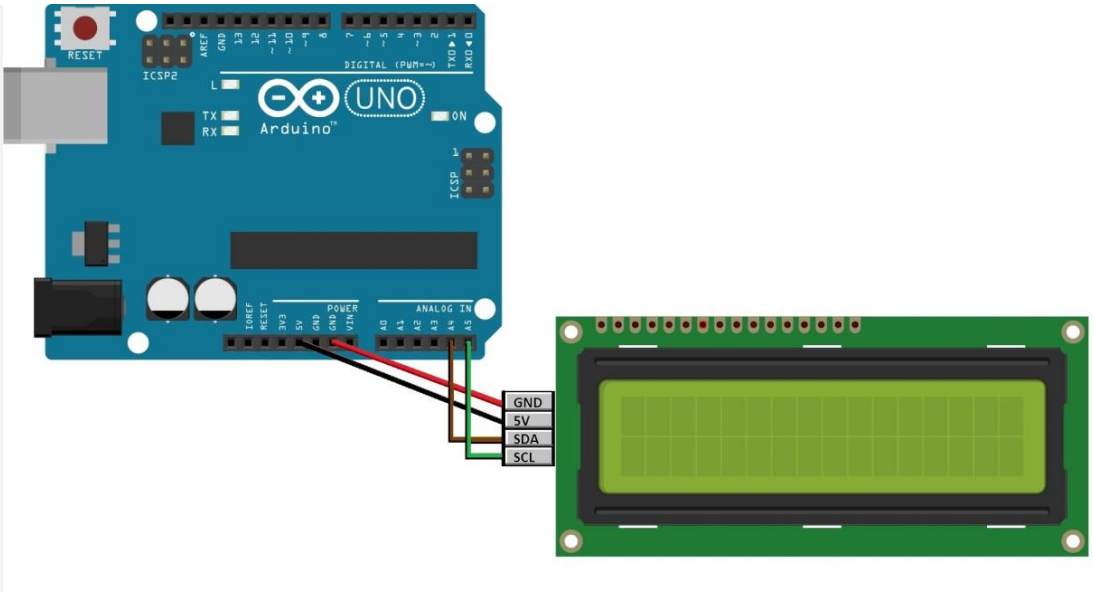
Si no sabemos en qué dirección está un componente dado, basta con colocarlo solo en el bus, y ver qué dirección nos reporta el I2C scanner. EL resultado para el LCD que tengo es 0x27 Hexadecimal.



Así que ya podemos pasar a hablar de como programar el uso del display.

DIAGRAMA DE CONEXIÓN

La conexión es, nuevamente, trivial:



Simplemente Arduino A4 a SDA (Datos) y A5 a SCL (Clock), más GND y alimentación. Vamos con el programa.

PROGRAMA DE CONTROL

Lo primero, es que descarguéis una nueva librería para maneja el display con I2C, Se llama LiquidCrystal_I2C y es un añadido para la librería estándar que viene con tu Duino (tranquilo no duele y debería ser automático).

No he encontrado información del autor o autores, y solo he visto que hacen referencia a que el nombre de la librería es Malpartida (A mí no me miréis).

Descarga la librería **LiquidCrystal_I2C** , y vamos a instalarla lo primero \\Programa\ImportarLibreria\Añadir \Librería

Busca en tu directorio de descargas, la librería LiquidCrystal_I2C .zip y haz doble click, para que se instale en nuestro IDE.

Ahora vamos a incluir la librería I2c que en Arduino se llama Wire, y como es estándar la incluyes con:

\\Programa\ImportarLibreria\Wire

Ahora hay que incluir la librería LiquidCrystal normal y después la de LiquidCrystal_I2C.Deberiamos tener 3lineas como estas:

```
#include <Wire.h>

#include <LCD.h>

#include <LiquidCrystal_I2C.h>
```

Vamos ahora a definir una variable con la dirección de nuestro display. En nuestro caso la 0x27.

```
byte dir = 0x27   // Ese 0x significa que es hexadecimal, no decimal
```

Y por último creamos una instancia del objeto LiquidCrystal_I2C:

```
LiquidCrystal_I2C

lcd( dir, 2, 1, 0, 4, 5, 6, 7);
```

Al que le pasamos la dirección dir del display y varios números que indican que pines debe usar la librería para el display, no de Arduino. Ignóralo y lo copias así para los displays. El resto queda así:

```
#include <Wire.h>

#include <LCD.h>

#include <LiquidCrystal_I2C.h>



#define I2C_ADDR    0x27



LiquidCrystal_I2C             lcd(I2C_ADDR,2, 1, 0, 4, 5, 6, 7);

void setup()

   {

       lcd.begin (16,2);    // Inicializar el display con 16 caracer
es 2 lineas

       lcd.setBacklightPin(3,POSITIVE);

       lcd.setBacklight(HIGH);



       lcd.home ();                   // go home

       lcd.print("Prometec.net");

       lcd.setCursor ( 0, 1 );        // go to the 2nd line

       lcd.print("Malpartida lib");

   }

void loop()

   {}
```