

[BLOG](#)[SOY UN PARDILLO](#)[PODCAST](#)[CURSO ARDUINO](#)[CURSO DOMÓTICA](#)[ACCEDER](#)

Usted está aquí: [Inicio](#) / [Blog](#) / [Arduino](#) / Matriz LED con Arduino MAX7219 MAX7221

## Matriz LED con Arduino MAX7219 MAX7221

José Guerra Carmenate

Con este artículo vas a ver el control de una **matriz LED con Arduino** y el integrado **MAX7219 o MAX7221** (el que tengas a tu disposición) que permite utilizar unos pocos pines del microcontrolador para su funcionamiento.

Imagínate que quieres hacer una pancarta digital para mostrar un emoji o cualquier chorrada que se te ocurra. Utilizando una **matriz LED con Arduino** puede ser bastante sencillo.

Ojo, que también puedes utilizarlo para algo más serio como indicador de obras en una carretera o peligro nieve.

Lo que tengo claro es que si necesitas utilizar una **matriz de LED**, la forma más sencilla que conozco es utilizando un **Arduino**.

Y da lo mismo si es un [Arduino UNO](#), [Arduino MEGA](#) o un [Arduino Nano](#). Todas estas placas son **compatibles** a nivel de **hardware** y **software**.

En este tutorial aprenderás a:

- Qué es una **matriz de LEDs** y cuál es su estructura interna.
- El funcionamiento del circuito integrado **MAX7219 y MAX7221** como **driver** de **matrices de LED**.
- Los módulos de **matriz LED con Arduino** existentes.
- Cómo hacer un cartel de **matrices LEDs con arduino** a tu medida.



### Indice de contenidos

- 1 ¿Qué es una matriz de LEDs?
- 2 ¿Cómo manejar una matriz LED?
- 3 Circuito integrado MAX7219 y MAX7221
- 4 Pinout del MAX7219
- 5 Conexión MAX7219 y matriz LED con Arduino
- 6 Como conectar un módulo matriz LED con Arduino
- 7 Librería MAX7219 con Arduino
- 8 Como crear dibujos y animaciones en una matriz LED con Arduino
- 9 Scroll de mensajes con varias matrices de LED con Arduino
- 10 Conclusiones matriz LED con Arduino MAX7219

## ¿Qué es una matriz de LEDs?

Una **matriz LED** no es otra cosa que un **conjunto de LEDs** agrupados por filas y columnas.



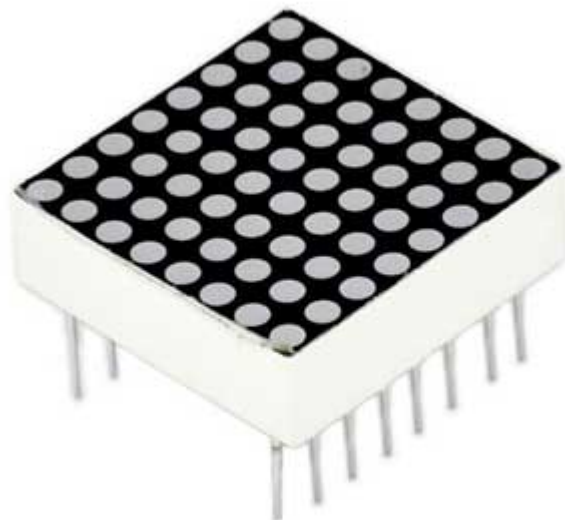
Encendiendo y apagando LEDs de la matriz podrás crear gráficos, figuras, textos y animaciones. Con estas cualidades, tu próximo letrero no pasará desapercibido.



Te las puedes encontrar en casi cualquier lugar, ya sean carteles publicitarios, señales de tráfico o en un ascensor.



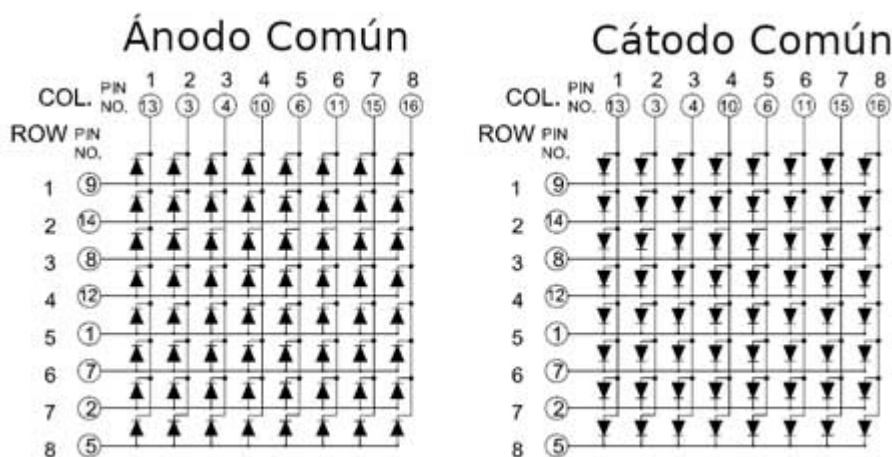
Las **matrices LEDs 8x8** son sumamente conocidas. Su nombre se debe a que están compuestas por 64 LEDs dispuestos en forma de cuadrado con 8 columnas de 8 LEDs cada una.



Su funcionamiento es muy parecido al [display de 7 segmentos](#).

## Matriz LED cátodo común y ánodo común

Las **matrices de LEDs** pueden ser de dos tipos: ánodo común o cátodo común.



**Matriz cátodo común:** los terminales negativos (cátodos) de todos los LEDs de cada fila están conectados juntos. Lo mismo ocurre con los pines positivos (ánodos) de cada columna.

**Matriz ánodo común:** las conexiones son contrarias, es decir, los ánodos se conectan a las filas y los cátodos a las columnas.

Puedes notar que solo se cuenta con 16 pines para controlar la matriz, 8 para las columnas y 8 para las filas.

## Características técnicas de una matriz LED

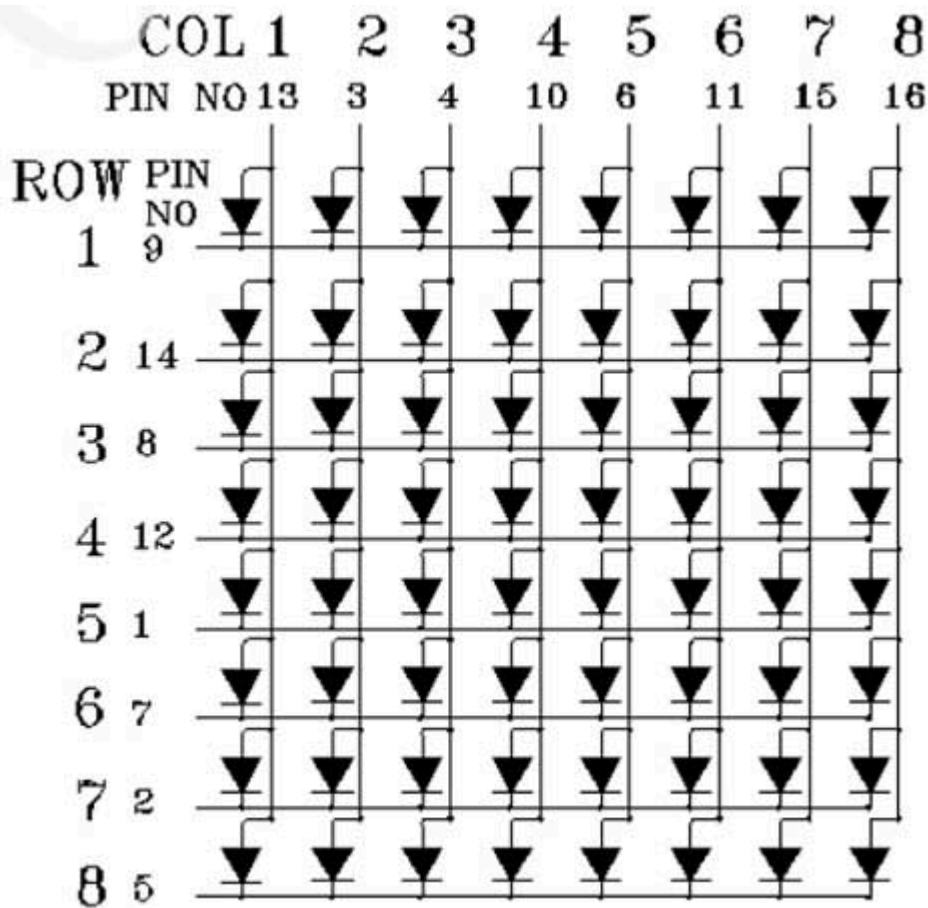
Teniendo en cuenta que una matriz sólo está compuesta por LEDs nos interesa conocer los siguientes parámetros:

- Tipo de matriz y distribución de pines
- Voltaje de operación de los LEDs
- Corriente de operación de los LEDs
- Color de los LEDs

Para analizar las características técnicas tomaré como ejemplo la matriz LED TOP-CC-1088AS-N4, muy común en los módulos de Arduino.

Para identificar el tipo de matriz debes observar la [hoja de datos](#) que ofrece el siguiente esquema con la estructura interna.





Se trata de una matriz tipo cátodo común. Puede ver la conexión de los LEDs y los pines en cada fila y columna.

En la tabla de “**ABSOLUTE MAXIMUM RATINGS**” puedes obtener la **corriente y el voltaje inverso máximos soportado por los LEDs**. En este caso son 5 voltios y 20 mA respectivamente.

#### 5-1. ABSOLUTE MAXIMUM RATINGS (Ta=25°C)

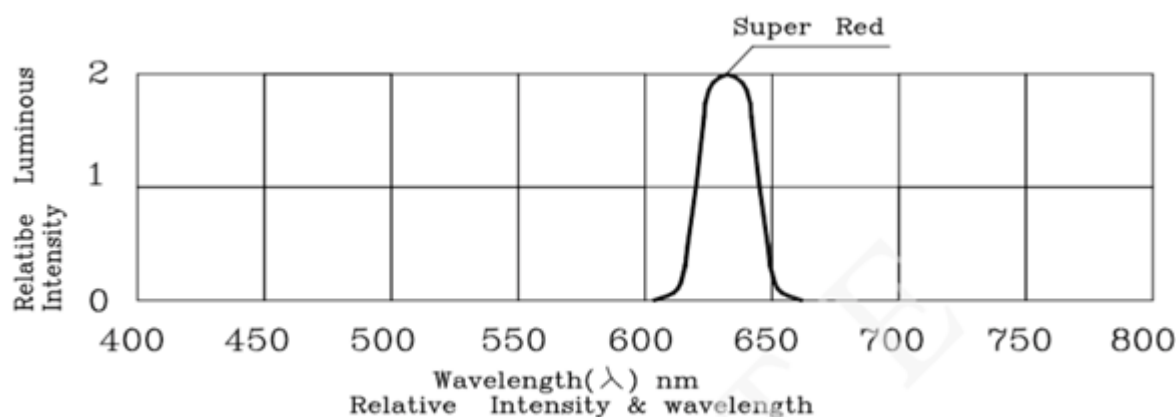
PARAMETER	SYMBOL	VALUE	UNIT
Reverse Voltage	$V_R$	5	V/dot*
Forward Current	$I_F$	20	mA/dot*

El **voltaje de operación de los LEDs** lo puedes encontrar en la tabla “**ELECTRICAL-OPTICAL CHARACTERISTICS**”.

PARAMETER	SYMBOL	MIN.	TYP.	MAX.	UNIT	TEST CONDITIONS
Luminous Intensity	I <sub>V</sub>	R	13500	15525	ucd	I <sub>F</sub> =10mA
		S	17550	21938		
		T	26326	32908		
Forward Voltage	V <sub>F</sub>	1.80	2.10	2.40	v/dot*	I <sub>F</sub> =20mA
Dominant wavelength	λ <sub>d</sub>	630	-	640	nm	I <sub>F</sub> =20mA
Spectral Line Half-Width	Δ λ	-	20	-	nm	I <sub>F</sub> =20mA
Reverse Current	I <sub>R</sub>	-	-	20	uA	V <sub>R</sub> =5v

El voltaje de operación de los LEDs puede variar de 1.8 a 2.4 voltios, es decir, cuando lo enciendas esa es la tensión que consume el LED.

Por último, el color de los LEDs es mostrado en forma de gráfica, indicando cuál es la longitud de onda predominante.

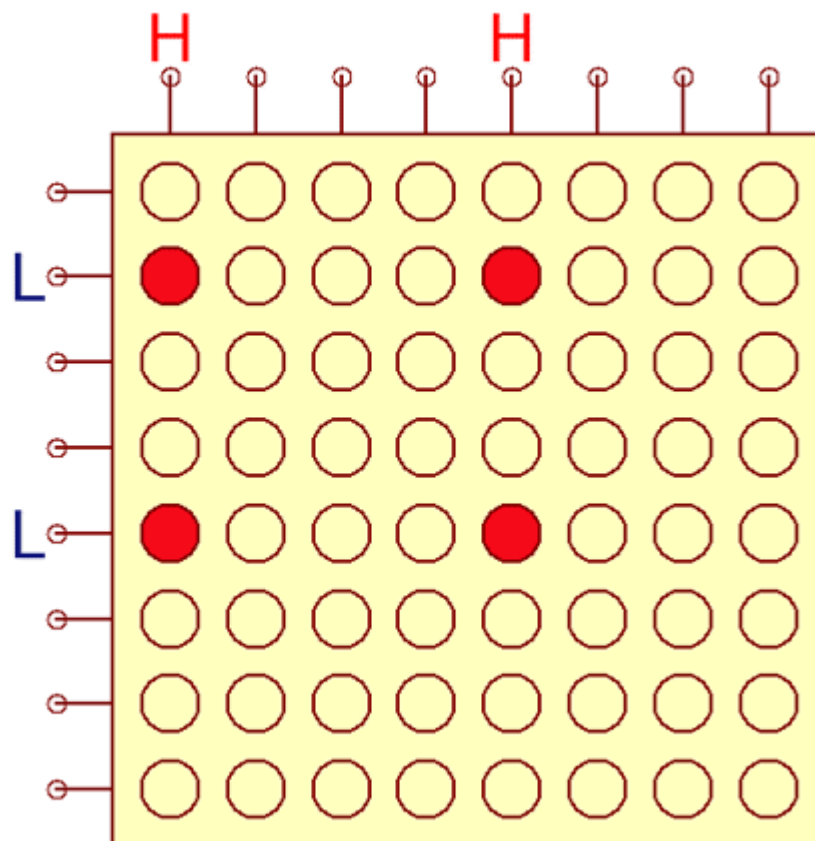


En este caso, se encuentra entre los 600 y 650 nm, correspondiente al color rojo (*Super Red*).

## ¿Cómo manejar una matriz LED?

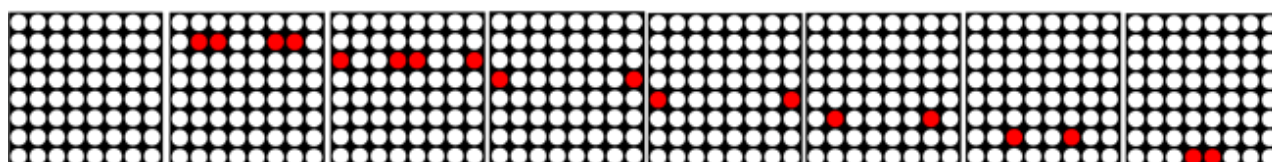
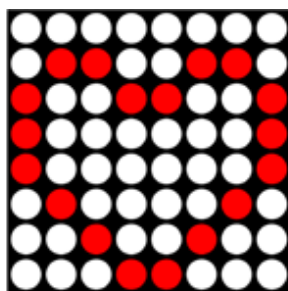
En una matriz no es posible controlar todos los LEDs como si fueran independientes. Esto pasa porque solo se dispone de los pines correspondientes a filas y columnas.

Si se aplican valores de alto (HIGH) y bajo (LOW) a varias columnas y filas, respectivamente, se encenderán todos los LEDs de las intersecciones.



Por ello, es difícil generar **gráficos complejos**.

Para poder mostrar gráficos correctamente es necesario realizar un barrido por filas o columnas. Se iluminará sólo una fila a la vez.



Por ejemplo, si quieres mostrar un corazón son necesarios 8 pasos, uno por cada fila.

¿Pero esto no muestra solo una fila a la vez?

Pues no.



Las actualizaciones se realizan suficientemente rápido y con ayuda del efecto de **persistencia visual** el ojo humano no es capaz de notar el instante donde los **LEDs** se apagan.

Así es como funcionaban los antiguos televisores CRT.



Por lo tanto, tu ves **LEDs** funcionando pero realmente solo se activa una fila a la vez.

Si utilizas la matriz de esta forma es necesario colocar una resistencia en cada fila para limitar la corriente por los LEDs.

## Circuito integrado MAX7219 y MAX7221

Controlar una **matriz de 8x8** implica utilizar **16 señales digitales** y refrescar la imagen de una forma constante. Es por eso que en lugar de utilizar un Arduino directamente, emplearemos el circuito integrado **MAX7219** o **MAX7221** para esta tarea.

Los circuitos integrados **MAX7219** y **MAX7221** son casi iguales. Eso significa que son fácilmente intercambiables uno por el otro. De ahora en adelante me referiré al **MAX7219**, aunque todo es igual de válido para el **MAX7221**.

El **MAX7219** es un circuito integrado que facilita el control de LEDs. Es usado principalmente en pantallas de 7 segmentos, **paneles de LEDs industriales** y como controlador de **matriz de LED con Arduino**.

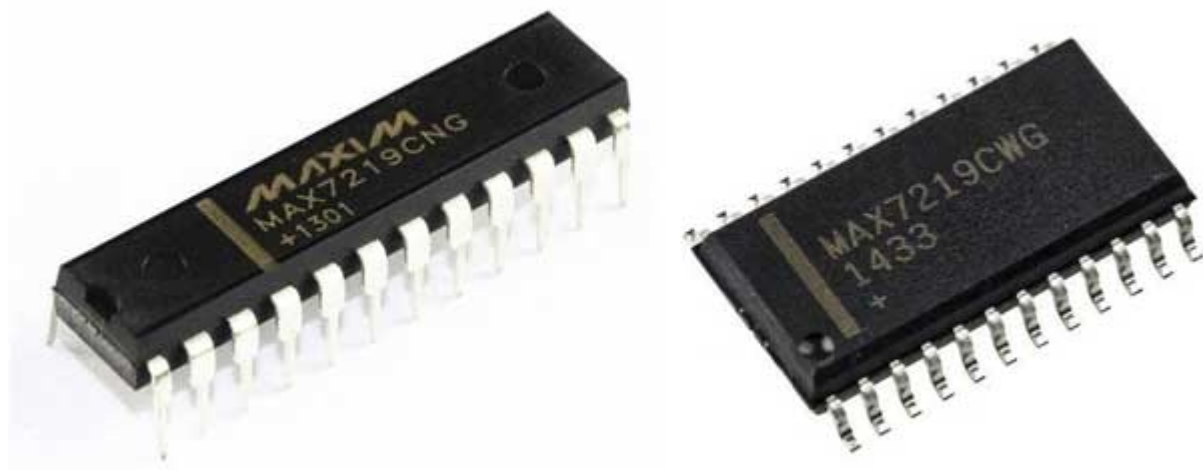
Lo puedes encontrar en cualquier tienda de electrónica e incluso en grandes portales como Amazon donde ya viene todo incluido: [matriz de LEDs y circuito integrado](#).

Entre sus ventajas:

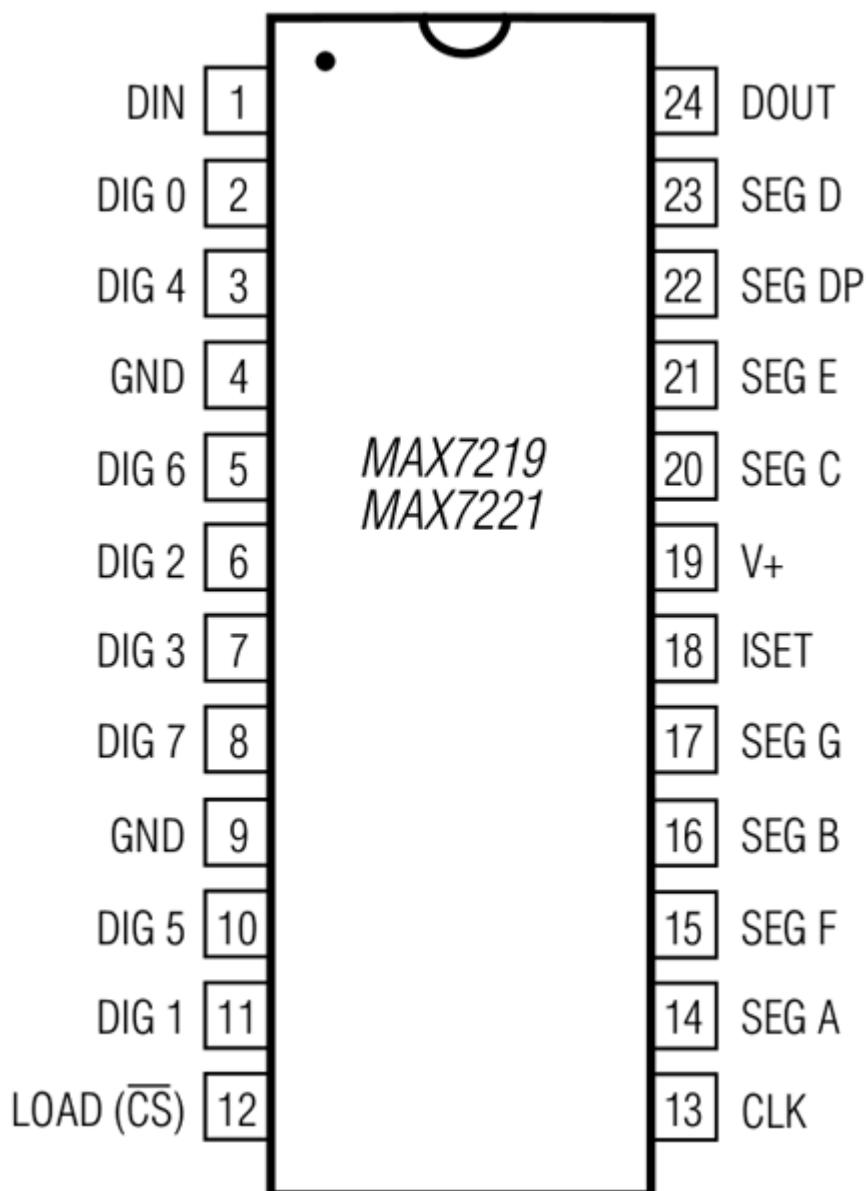
- **Interfaz de control serie:** con tan solo 3 pines podemos controlar toda una matriz de LEDs.
- **Circuito externo simple:** requiere pocos componentes externos.
- **Conexión en cascada:** se pueden conectar varios MAX7219 en cascada. De esta forma se puede controlar varias matrices LEDs utilizando solo 3 pines de la **placa Arduino**.
- **Modo de bajo consumo:** puede consumir solo 120 uA.

## Pinout del MAX7219

En la distribución de pines del chip MAX7219 puedes ver que cuenta con 24 pines y lo puedes encontrar, en encapsulado DIP o SO.



En la siguiente imagen extraída de su [hoja de datos](#) puedes observar su **distribución de pines**.



- El pin **V+** es el pin de alimentación y según la tabla “**ABSOLUTE MAXIMUM RATINGS**” de la hoja de datos puede soportar hasta 6 voltios.

## ABSOLUTE MAXIMUM RATINGS

Voltage (with respect to GND)

V+ .....	-0.3V to 6V
DIN, CLK, LOAD, CS .....	-0.3V to 6V
All Other Pins .....	-0.3V to (V+ + 0.3V)

Current

DIG 0–DIG 7 Sink Current.....	500mA
SEG A–G, DP Source Current.....	100mA

Continuous Power Dissipation ( $T_A = +85^\circ\text{C}$ )

Narrow Plastic DIP (derate 13.3mW/°C above +70°C).....	1066mW
Wide SO (derate 11.8mW/°C above +70°C).....	941mW
Narrow Cerdip (derate 12.5mW/°C above +70°C) ..	1000mW

- Los pines **GND** se conectan al terminal negativo de la alimentación.

- Los pines **DIG0-DIG7** se usan para controlar las filas de la matriz.
- Los pines **SEGA-SEGG**, **DP** son empleados para controlar las columnas de la matriz.
- Los pines **DIN**, **SCK** y **CS** conforman la interfaz de comunicación, es decir, que mediante estos pines la **placa Arduino** le envía comandos al chip.
- El pin **DOUT** es utilizado para conectar varios MAX7219 en cascada.
- Por último el pin **ISSET** permite configurar la corriente utilizada para cada LED. Esto influye en la intensidad de los LEDs.

Aunque es posible alimentarlo con 6 voltios lo recomendable es no aplicar un voltaje superior a los 5.5 voltios en ninguno de sus pines.

En la tabla “**ELECTRICAL CHARACTERISTICS**” de la hoja de datos puedes encontrar, voltajes de niveles lógicos para la comunicación, o sea, que voltajes son necesarios para comunicarse con el chip.

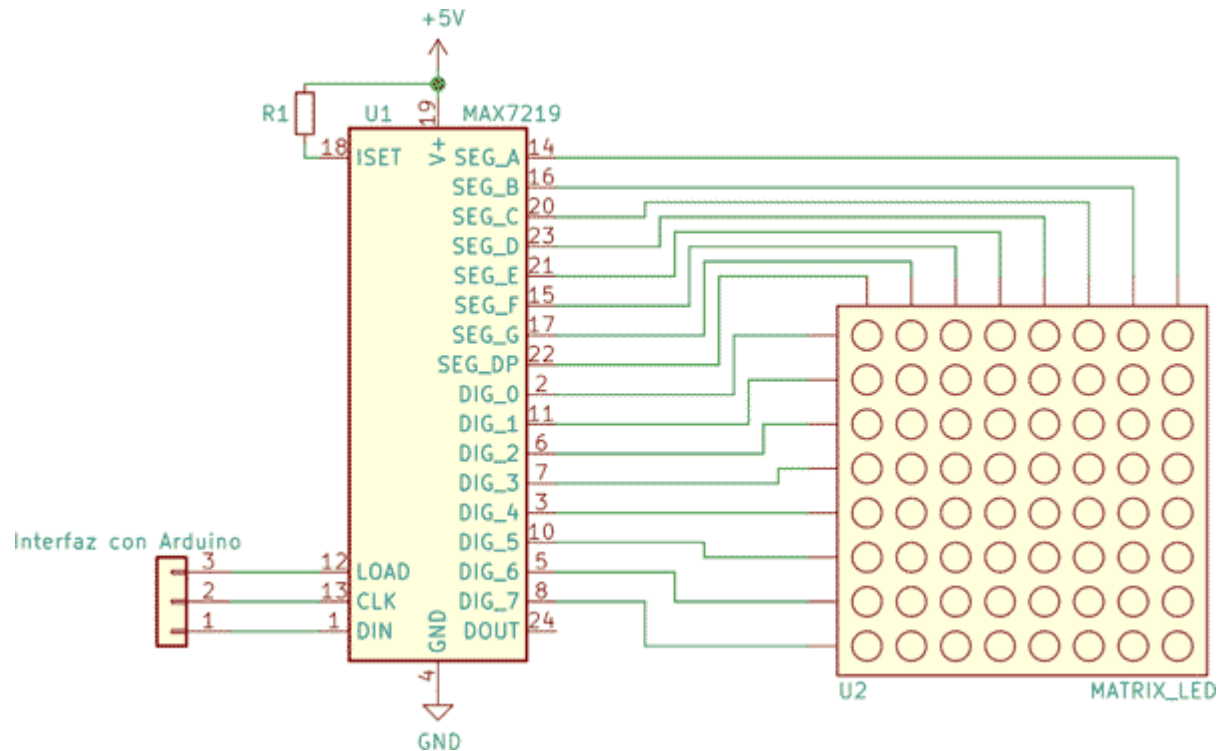
PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
<b>LOGIC INPUTS</b>						
Input Current DIN, CLK, LOAD, $\overline{CS}$	$I_{IH}, I_{IL}$	$V_{IN} = 0V \text{ or } V_{+}$	-1		1	$\mu A$
Logic High Input Voltage	$V_{IH}$		3.5			V
Logic Low Input Voltage	$V_{IL}$				0.8	V
Output High Voltage	$V_{OH}$	DOUT, $I_{SOURCE} = -1mA$	$V_{+} - 1$			V
Output Low Voltage	$V_{OL}$	DOUT, $I_{SINK} = 1.6mA$			0.4	V
Hysteresis Voltage	$\Delta V_I$	DIN, CLK, LOAD, $\overline{CS}$		1		V

En la figura anterior he señalado el valor “**Logic High Input Voltage**”, que es el voltaje de entrada de estado alto. Es necesario aplicar un voltaje superior a los 3.5 voltios en los pines de comunicación para que el MAX7219 lo interprete como un estado alto (HIGH).

**No puedes utilizar este circuito integrado con placas que operen a 3.3 voltios** como la [NodeMCU](#) o [WeMos](#) para este proyecto.

## Conexión MAX7219 y matriz LED con Arduino

En la siguiente figura puedes ver el esquema necesario para conectar un **MAX7219** a una matriz de 8x8.



Solo es necesario usar una resistencia externa entre los pines **ISET** y **V+**. Esta resistencia permite configurar la intensidad máxima con que se iluminan los LEDs de la matriz. En la hoja de datos del MAX7219 se ofrece una tabla que nos permite estimar qué resistencia debemos utilizar.

**Table 11. RSET vs. Segment Current and LED Forward Voltage**

ISEG (mA)	VLED (V)				
	1.5	2.0	2.5	3.0	3.5
40	12.2	11.8	11.0	10.6	9.69
30	17.8	17.1	15.8	15.0	14.0
20	29.8	28.0	25.9	24.5	22.6
10	66.7	63.7	59.3	55.4	51.2

Por ejemplo, si utilizas una **matriz con LEDs** de 3.5 voltios y quieres que cada LED se encienda con 20 mA es necesario que la resistencia tenga un valor de 22.6 kΩ.

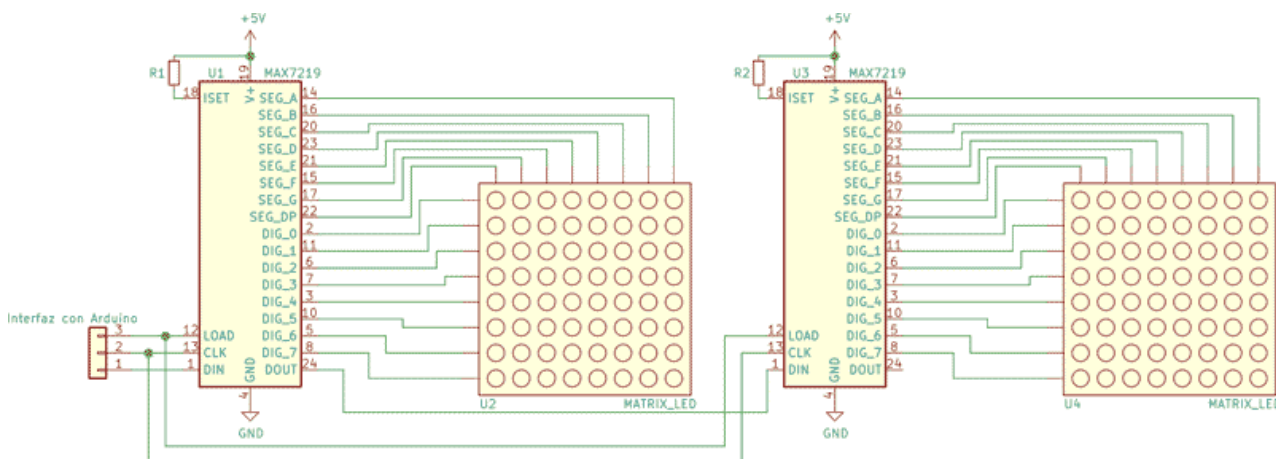
Los pines **LOAD** (o **CS**), **CLK** y **DIN** están conectados al **Arduino** para controlar el circuito.

## Conectar matrices LED en cascada



Conectando varios **MAX7219** en cascada te permite controlar las matrices con solo tres pines del Arduino.

En la siguiente figura puedes ver como quedaría la conexión para dos matrices LEDs con Arduino.



Los pines **CLK** y **LOAD** de ambos chips se conectan juntos. Por otra parte, el pin **DOUT** del primer **MAX7219** se ha conectado al pin **DIN** del segundo.

## Protocolo de comunicación del MAX7219

El **MAX7219** es compatible con protocolos **SPI**, **QSPI** y **MICROWIRE**. Eso significa puedes usar cualquiera de ellos para enviar comandos.

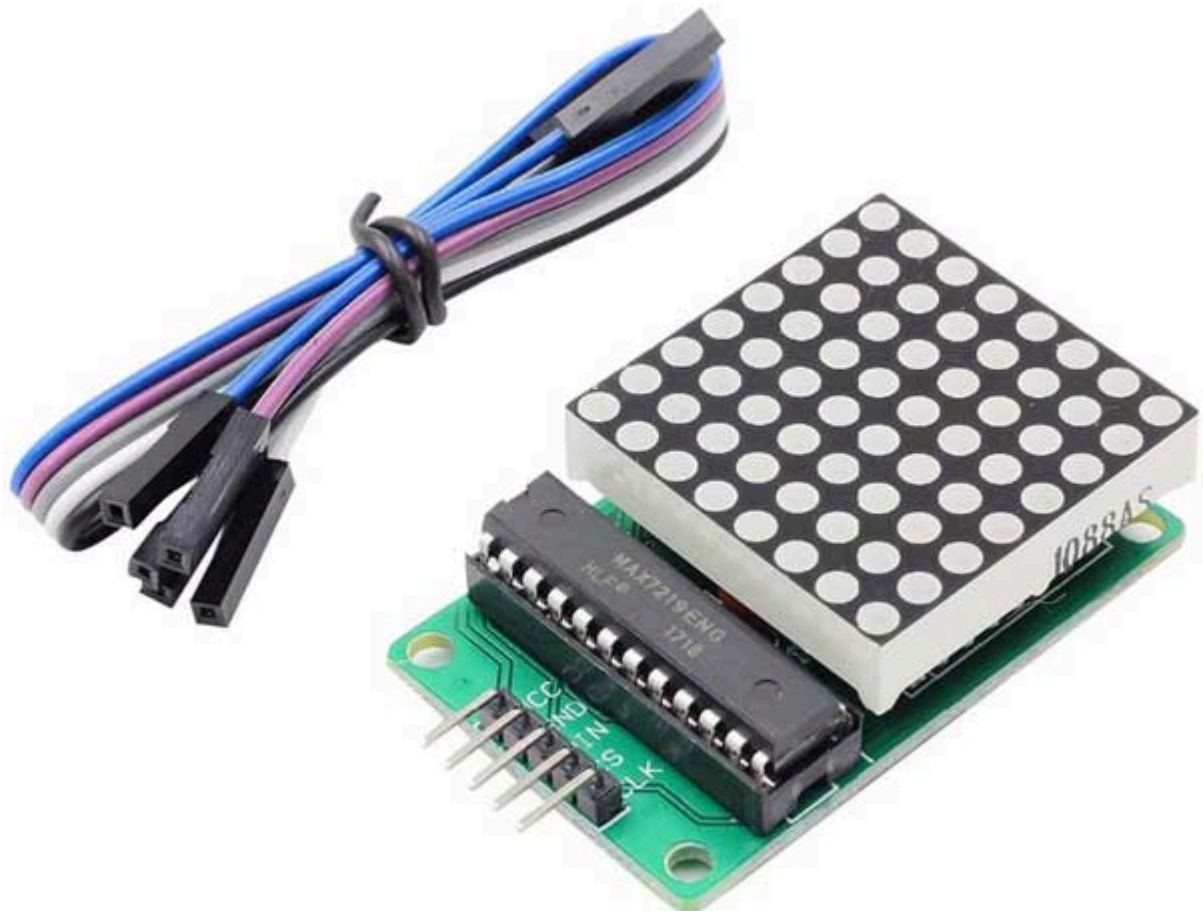
Los comandos que admite son de 16 bits (2 bytes). El primer byte se refiere al registro a modificar y el segundo al valor que se le asigna. Por ejemplo, el registro puede ser el asociado a una columna de la matriz y el valor, indicar que LEDs encender y cuáles apagar.

Esta dificultad se soluciona con una librería que se encarga de realizar todo el trabajo de comunicación a bajo nivel.

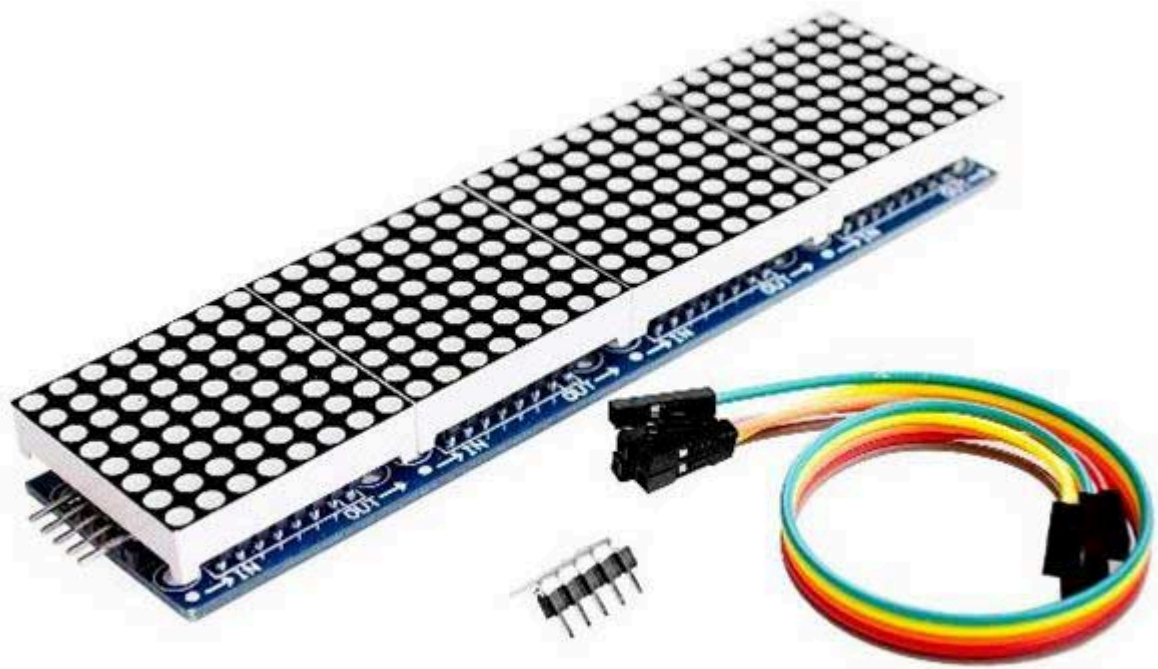
## Módulos de matriz LED con Arduino basados en MAX7219

Ya sabes cómo conectar un **MAX7219** para controlar una **matriz LED con Arduino**. Sin embargo, es tedioso activar 6 **matrices de LED con Arduino** por la cantidad de conexiones necesarias.

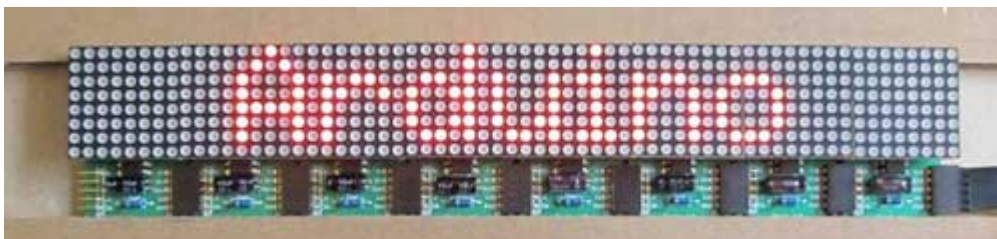
Por suerte, existen módulos de **matrices LEDs para Arduino** con el circuito montado. Existen de una única matriz o de varias.



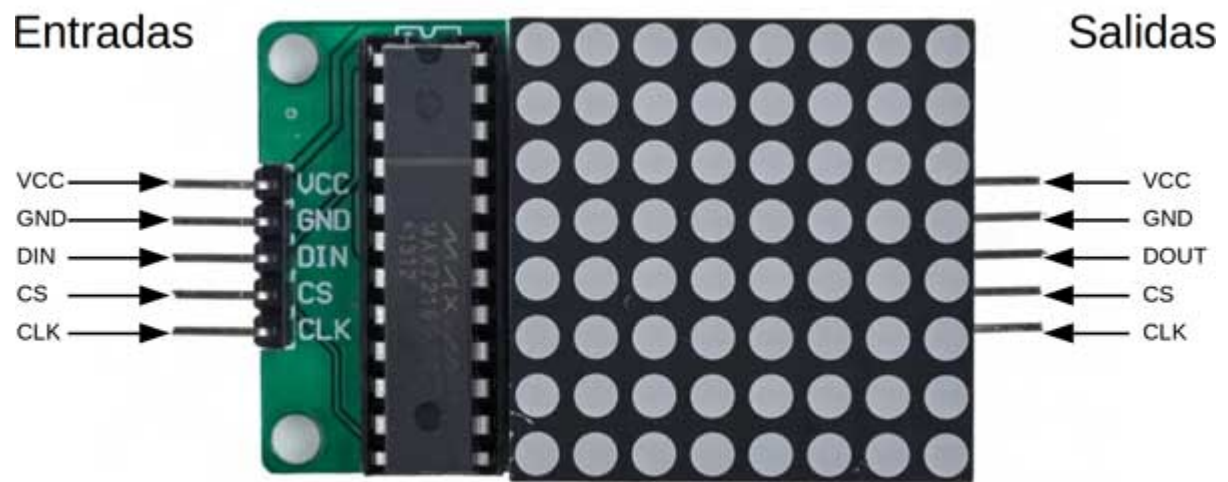
Los que están compuestos por varias matrices utilizan un **MAX7219** por cada una, y estos están conectados tal y como vimos en el esquema anterior. Es decir, que **se conectan todos los MAX7219 en cascada**.



Algunos módulos como los de *Parola for Arduino* están diseñados para unir varios como si fuese una única pantalla.



Los módulos simples, no vienen diseñados para este tipo de acople, pero cuentan con dos conectores: uno de entrada y uno de salida. Esto nos permite conectar varios **módulos en cascada**.



El conector de entrada cuenta con los pines **VCC** y **GND** que son utilizados para alimentar el módulo. Además, están los pines **DIN**, **CS** y **CLK** que son utilizados para la comunicación.

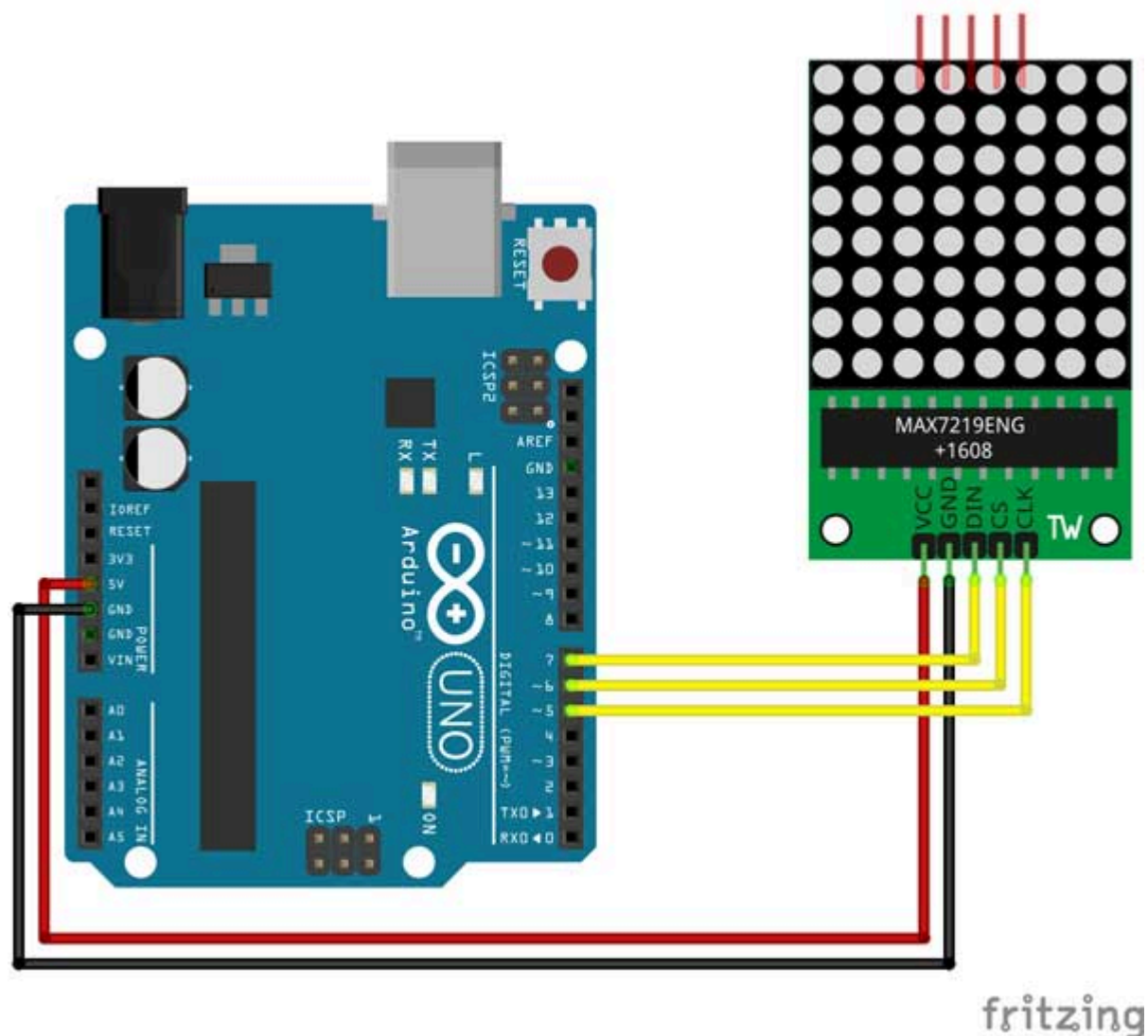
En algunos módulos el pin **CS** es nombrado como **LOAD**.

El conector de salida es idéntico al de entrada. La única diferencia es que se ha sustituido el pin **DIN** por el pin **DOUT**. Este segundo conector es el que permite **conectar varios módulos en cascada**.

## Como conectar un módulo matriz LED con Arduino

Para usar un **módulo de matriz de LED para Arduino basado en MAX7219** es necesario conectar el pin **VCC**, al pin **+5V** de la placa y unir los pines **GND** del módulo y el Arduino. Los pines **DIN**, **CLK** y **CS** se conectan a tres **pinos digitales del Arduino**.

Aquí puedes ver cómo queda la conexión de un **Arduino UNO** con un módulo simple.

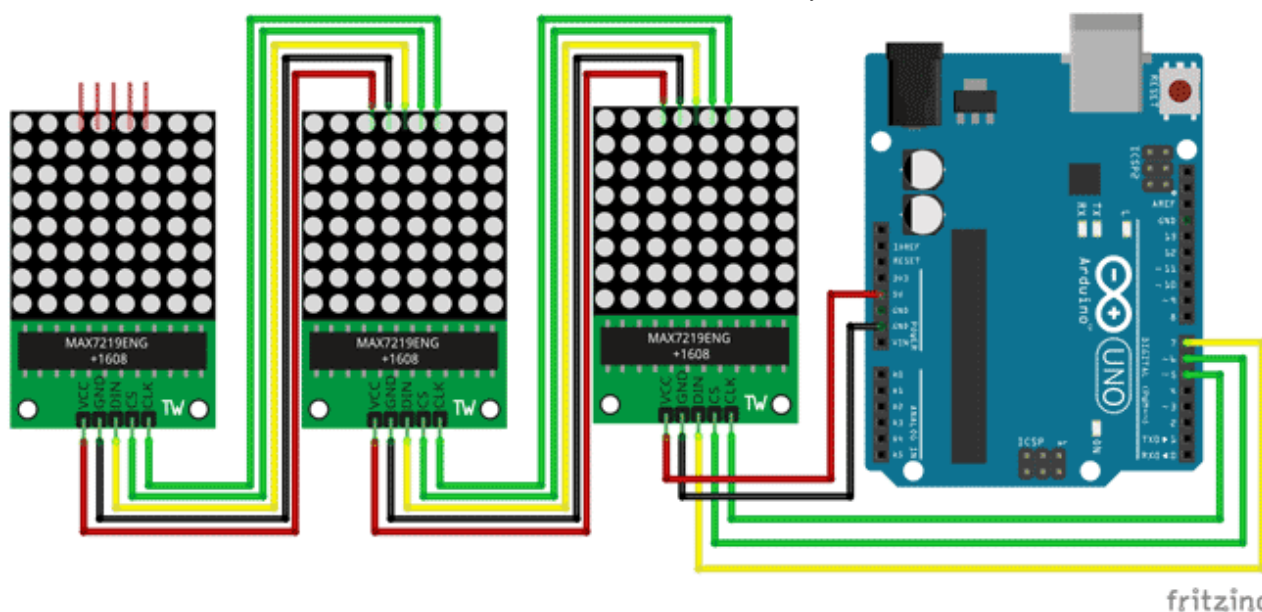


El orden de los pines puede cambiar de un módulo a otro, por lo tanto, es importante que revises la nomenclatura antes de realizar la conexión.

## Conexión de varios módulos de matrices LEDs con Arduino

Para utilizar varios **módulos MAX7219** es necesario conectar el primero al Arduino como el caso anterior. Luego, se une el terminal de salida de ese módulo con el de entrada del siguiente y así sucesivamente.





Aquí se muestra el esquema de conexiones para tres módulos de matriz LED con Arduino basados en **MAX7219**.

## Librería MAX7219 con Arduino

Existen muchas de librerías para **módulos de matriz LED con Arduino**. Aquí vas a ver la biblioteca [MD\\_MAX72xx](#) que te permite hacer maravillas con tu cartel digital.

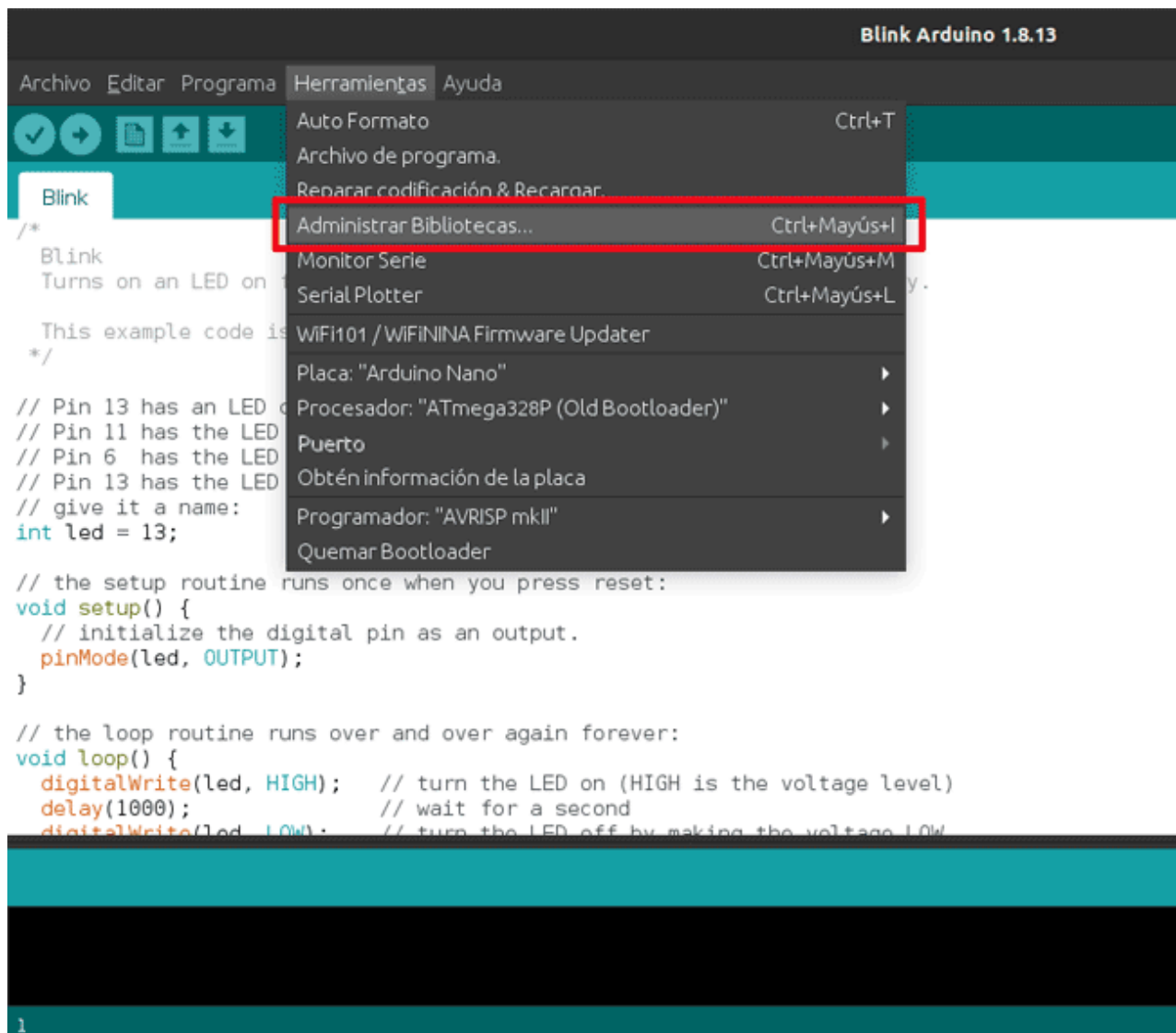
Ya sabes como conectar un modulo de **matriz de LED con Arduino basado en MAX7219**, ahora solo queda aprender a programar su código.

Una de las ventajas de esta librería es que **si conectas cuatro módulos en cascada los puedes usar como si fueran una matriz de 8 pixeles de altura y 32 de ancho**.

## Instalar librería MAX7219 con Arduino

Comencemos instalando la librería.

Para esto tienes que abrir el **Gestor de Librerías** que se encuentra en *Herramientas/Administrar Bibliotecas...*



Una vez abierto el Gestor de Librerías busca **MD\_MAX72** e instala la librería.



Listo ya tienes la librería instalada.

## Funciones principales de la librería MD\_MAX72xx

Como todas las **librerías para Arduino**, la MDMAX72xx tiene un montón de funcionalidades que pueden hacer muchas cosas.

De momento, las funciones y método más interesantes para nosotros son las siguientes.

## Función MD\_MAX72XX()

Esta sentencia es llamada constructor. Se usa para crear un objeto **MD\_MAX72XX**. Simboliza los módulos de matrices LEDs que conectas al Arduino. La función cuenta con dos **sobrecargas**.

Una función que puede ser llamada de diferentes maneras dependiendo el número de argumentos, se dice que es una **función sobrecargada**.

**Primer sobrecarga:** sólo requiere tres argumentos y devuelve un objeto que **utiliza el puerto SPI del Arduino para comunicarse** con el módulo de matrices LEDs.

```
1 MD_MAX72XX mxObj = MD_MAX72XX(hardware, pinCS, matCount);  
2
```

Donde:

- **mxObj:** es el objeto retornado por la función. Para ejecutar el resto de las funciones es necesario escribir su nombre y luego llamar la función.
- **hardware:** este parámetro permite establecer la posición de la matriz, es decir, te permite indicar a la librería cómo has colocado tu matriz de LEDs.
- **pinCS:** especifica el pin utilizado como *chip select*, es decir, el pin del Arduino que se conecta al pin LOAD(CS) del módulo.
- **matCount:** la cantidad de matrices que trae nuestro módulo o la cantidad de módulos simples que hemos conectado en cascada.

Si utilizas esta sobrecarga la librería usa el puerto SPI del Arduino para comunicarse con el módulo. Por lo tanto, los **pines DIN y CLK del módulo MAX7219** se tienen que conectar a los **pines MOSI y SCK de la placa Arduino**, respectivamente.

```
1 //Modulo de matriz LED con Arduino basado en MAX7219 o MAX7221  
2 // Utilizar:  
3 // - el tipo de hardware DR0CR0RR1_HW,
```

```

4 // - la interfaz SPI para comunicación,
5 // - el pin 10 como CS,
6 // - se conectan 5 matrices
7 MD_MAX72XX mx = MD_MAX72XX(MD_MAX72XX::DR0CR0RR1_HW, 10, 5);

```

**Segunda sobrecarga:** permite controlar los módulos con cualquier pin digital.

```

1 MD_MAX72XX mxObj = MD_MAX72XX(hardware, pinDIN, pinCLK, pinCS, matCount);

```

Donde:

- **pinDIN:** es el pin digital del Arduino que se conecta al pin DIN del módulo.
- **pinCLK:** es el pin digital del Arduino que se conecta al pin CLK del módulo.
- El resto de las variables cumplen la misma función que en la sobrecarga anterior.

```

1 //Modulo de matriz LED con Arduino basado en MAX7219 o MAX7221
2 // Utilizar:
3 // - PAROLA_HW como tipo de hardware
4 // - pin digital 2 conectado al pin DIN
5 // - pin digital 4 conectado al pin CLK
6 // - pin digital 3 conectado al pin CS
7 // - se conectan dos matrices
8
9 MD_MAX72XX mx = MD_MAX72XX(MD_MAX72XX::PAROLA_HW, 2, 4, 3, 2);

```

En ambas sobrecargas el argumento **hardware** puede tomar cualquiera de los siguientes valores:

- **MD\_MAX72XX::GENERIC\_HW:** hardware genérico.
- **MD\_MAX72XX::FC16\_HW:** para el uso de módulos tipo **FC-16**.
- **MD\_MAX72XX::PAROLA\_HW:** para el uso de módulos tipo *Parola*.
- **MD\_MAX72XX::ICSTATION\_HW:** para el uso de módulos tipo **ICStation**.
- **MD\_MAX72XX::DR0CR0RR0\_HW \***
- **MD\_MAX72XX::DR0CR0RR1\_HW \***
- **MD\_MAX72XX::DR0CR1RR0\_HW:** equivalente a *GENERIC\_HW*.
- **MD\_MAX72XX::DR0CR1RR1\_HW \***
- **MD\_MAX72XX::DR1CR0RR0\_HW:** equivalente a *FC16\_HW*.
- **MD\_MAX72XX::DR1CR0RR1\_HW \***
- **MD\_MAX72XX::DR1CR1RR0\_HW:** equivalente a *PAROLA\_HW*.
- **MD\_MAX72XX::DR1CR1RR1\_HW:** equivalente a *ICSTATION\_HW*.

\* Estos valores se utilizan cuando se conectan matrices en cascada y no corresponden ninguno de los estándares. Para la conexión propuesta anteriormente (la de tres módulos simples con Arduino UNO) es necesario utilizar **MD\_MAX72XX::DR0CR0RR1\_HW**.

## Función control()

Esta función permite configurar algunos parámetros de control en los módulos de matrices LED basados en MAX7219. Puede ser utilizada para establecer la intensidad del brillo de los LEDs o poner el módulo en modo de bajo consumo.

**Primera sobrecarga:** admite dos argumentos y permite configurar un parámetro en todos los módulos:

```
1 mxObj.control(param, value);
```

Donde:

- **param:** es el parámetro a modificar.
- **value:** es el valor a aplicar al parámetro indicado.

```
1 // Sacar todas las matrices del modo de bajo consumo
2 mx.control(MD_MAX72XX::SHUTDOWN, false);
3
4 // Establecer a 1 la intensidad de los LEDs
5 mx.control(MD_MAX72XX::INTENSITY, 1);
```

**Segunda sobrecarga:** admite tres argumentos y permite configurar el parámetro de un solo módulo.

```
1 mxObj.control(mat, param, value);
```

Donde:

- **mat:** es un entero entre 0 y la cantidad de módulos conectados. Indica el módulo a configurar.
- **param:** es el parámetro a modificar.
- **value:** es el valor a aplicar al parámetro indicado.

```
1 // Poner el segundo módulo en modo de bajo consumo
2 mxObj.control(1, MD_MAX72XX::SHUTDOWN, true);
```

**Tercera sobrecarga:** permite aplicar la configuración a un intervalo de módulos.

```
1 mxObj.control(matIni, matFin, param, value);
```

Donde:

- **matIni:** inicio del intervalo de módulos.
- **matFin:** fin del intervalo de módulos.



- ***param***: parámetro a modificar.
- ***value***: valor a aplicar al parámetro indicado.

```
1 // Poner las matrices 0, 1, 2 y 3 en modo de prueba
2 mx.control(0, 3, MD_MAX72XX::TEST, true);
```

En todas las sobrecargas el argumento *param* puede tener varios valores. Los más importantes son:

- **MD\_MAX72XX::TEST**: activa o desactiva el modo de prueba. Cuando se activa (*value* = *true*) se encienden todos los **LEDs** para comprobar que ninguno esté dañado.
- **MD\_MAX72XX::SHUTDOWN**: activar o desactivar el modo de bajo consumo.
- **MD\_MAX72XX::INTENSITY**: permite especificar el brillo de los leds. El valor de *value* tiene que estar entre 0-15, donde 0 apaga los LEDs y 15 es el brillo máximo.
- **MD\_MAX72XX::UPDATE**: Permite habilitar o deshabilitar la actualización automática.

Cuando la **auto-actualización** está deshabilitada es necesario ejecutar la **función *update()*** para que los cambios sean mostrados en la matriz.

## Función getColumnCount()

Esta función permite conocer el número total de columnas, es decir, retorna la cantidad de matrices multiplicada por 8.

```
1 // obtener la última columna de todo el conjunto de matrices
2 int last_col = mx.getColumnCount() - 1;
```

## Función clear()

Esta función permite “limpiar” la información de las matrices, en otras palabras, apaga todos los LEDs de las matrices.

**Primera sobrecarga**: no requiere argumentos y **limpia todas las matrices**.

```
1 // Limpiar todas las matrices
2 mx.clear();
```

**Segunda sobrecarga**: permite limpiar un intervalo de matrices.

```
1 mxObj.clear(matIni, matFin);
```

Donde:

- **matIni**: inicio del intervalo de matrices a limpiar.
- **matFin**: fin del intervalo de matrices a limpiar.

```
1 // Limpiar las matrices 0, 1 y 2.  
2 mx.clear(0, 2);
```

## Función setColumnn()

Esta función permite establecer el estado de una columna de LEDs.

**Primera sobrecarga:** recibe tres parámetros y permite establecer el estado de una columna indicando su matriz:

```
1 mxObj.setColumn(mat, col, estado);
```

Donde:

- **mat**: es un entero que indica la matriz a modificar.
- **col**: es un entero entre 0 y 7 que indica la columna a modificar.
- **estado**: es un entero de 8 bits que indica los LEDs a encender. Cada bit puesto a 1 indica que se encenderá su LED correspondiente.

```
1 // Establecer un nuevo estado en la cuarta columna de la primera matriz  
2 // Se encienden los dos LEDs inferiores y el LED superior de la columna  
3 mx.setColumn(0, 3, B11000001);  
4  
5 // Establecer un nuevo estado en la octava columna de la tercera matriz  
6 // Se encienden el LED superior y el inferior  
7 mx.setColumn(2, 7, B10000001);
```

**Segunda sobrecarga:** recibe dos parámetros y también permite establecer un nuevo estado en una columna. La diferencia con la sobrecarga anterior es que en este caso **se asume que las matrices forman una matriz más extensa**.

Eso significa que si se emplean dos matrices es como si se utilizara una matriz de 8 filas y 16 columnas. Donde las columnas 0-7 se corresponden con la primera matriz y las columnas 8-15 con la segunda.

```
1 mxObj.setColumn(col, value);
```

Donde:

- **col:** es un entero que indica la columna a modificar. El valor de *col* debe estar entre 0 y la cantidad de columnas entre todos los módulos, es decir, que si utilizas tres matrices *col* puede tener valores entre 0 y 23.
- **value:** es un entero de 8 bits que indica los LEDs a encender. Cada bit puesto a 1 indica que se encenderá su LED correspondiente.

```

1 // Configurar la tercera columna de la segunda matriz con todos los LEDs encendidos
2 // - Las columnas 8-15 pertenecen a la segunda matriz
3 // - 0xFF = B11111111
4 mx.setColumn(10, 0xFF);
5
6 // Configurar la última columna de la segunda matriz con todos los LEDs apagados
7 // - Las columnas 8-15 pertenecen a la segunda matriz
8 // - 0x00 = B00000000
9 mx.setColumn(15, 0x00);
10

```

## Función setPoint()

Esta función permite encender o apagar un LED indicando su fila y columna.

```

1 mxObj.setPoint(fila, col, estado);

```

Donde:

- **fila:** indica la fila en que se ubica el LED. Es un valor entre 0 y 7.
- **col:** indica la columna del LED. Es un valor entre 0 y la cantidad de columnas total.
- **estado:** indica si el LED se debe encender o apagar.

```

1 // Encender el LED de la segunda fila y la 11na columna (tercera columna de la segunda
2 // - Las columnas 8-15 pertenecen a la segunda matriz
3 // - true -> encender
4 mx.setPoint(1, 10, true);
5
6 // Apagar el LED de la primera fila y la 7ma columna (7ma columna de la primera matriz)
7 // - Las columnas 0-7 pertenecen a la primera matriz
8 // - false -> apagar
9 mx.setPoint(0, 7, false);

```

## Función setRow()

Esta función permite establecer el estado de una fila de LEDs.

**Primera sobrecarga:** necesita tres argumentos y permite establecer el estado de una fila correspondiente a una matriz en particular.

```

1 mxObj.setRow(mat, fil, value);

```

Donde:

- **mat:** es un entero que indica sobre qué matriz actuar.
- **fil:** es un entero entre 0 y 7 que indica la fila a modificar
- **estado:** es un entero de 8 bits con el nuevo estado de la fila. Los bits puestos a uno indican los LEDs que se encenderán.

```
1 // Encender todos los LEDs de la 5ta fila de la primera matriz
2 // - 0xFF = B11111111
3 mx.setRow(0, 4, 0xFF);
4
5 // Encender la mitad de los LEDs de la 6ta fila de la tercera matriz
6 // - 0xF0 = B11110000
7 mx.setRow(2, 5, 0xF0);
```

La segunda sobrecarga solo necesita dos argumentos y establece un nuevo estado en la fila indicada para todas las matrices.

```
1 mxObj.setRow(fil, estado);
```

Donde:

- **fil:** es un entero entre 0 y 7 que indica la fila a modificar
- **estado:** es un entero de 8 bits con el nuevo estado de la fila. Los bits puestos a uno indican los LEDs que se encenderán.

```
1 // Encender todos los LEDs de la 5ta fila de todas las matrices
2 // - 0xFF = B11111111
3 mx.setRow(4, 0xFF);
4
5 // Encender la mitad de los LEDs de la 6ta fila de todas las matrices
6 // - 0xF0 = B11110000
7 mx.setRow(5, 0xF0);
```

**Tercera sobrecarga:** admite cuatro argumentos y permite establecer el estado de una fila en un intervalo de matrices.

```
1 mxObj.setRow(matIni, matFin, fil, estado);
```

Donde:

- **matIni:** es un entero que indica la primera matriz del intervalo.
- **matFin:** es un entero que indica la última matriz del intervalo.
- **fil:** es un entero entre 0 y 7 que indica la fila a modificar.
- **estado:** es un entero de 8 bits con el nuevo estado de la fila. Los bits puestos a uno indican los LEDs que se encenderán.

```

1 // Encender todos los LEDs de la 5ta fila de las matrices 2, 3 y 4
2 // - 0xFF = B11111111
3 mx.setRow(1, 3, 4, 0xFF);
4
5 // Encender la mitad de los LEDs de la 6ta fila de las matrices 1, 2 y 3
6 // - 0xF0 = B11110000
7 mx.setRow(0, 2, 5, 0xF0);

```

## Función transform()

Esta función permite aplicar transformaciones a la información mostrada en las matrices, es decir, que permite rotar, desplazar o invertir los LEDs en la matriz.

**Primera sobrecarga:** admite dos argumentos y permite aplicar una transformación al contenido de una sola matriz.

```
1 mxObj.transform(mat, trans);
```

Donde:

- **mat:** matriz a la que se le aplicará la transformación.
- **trans:** tipo de transformación a aplicar.

```

1 // Invertir todos los LEDs de la segunda matriz
2 // - MD_MAX72XX::TINV -> invertir LEDs
3 mx.transform(1, MD_MAX72XX::TINV );
4
5 // Desplazar el contenido de la primera matriz una posición a la izquierda
6 // - MD_MAX72XX::TSL -> Desplazar a la izquierda
7 mx.transform(0, MD_MAX72XX::TSL );

```

**Segunda sobrecarga:** admite tres argumentos y permite realizar una transformación a un intervalo de matrices.

```
1 mxObj.transform(matIni, matFin, trans);
```

Donde:

- **matIni:** es un entero que indica la primera matriz del intervalo.
- **matFin:** es un entero que indica la última matriz del intervalo.
- **trans:** tipo de transformación a aplicar.

```

1 // Desplazar el contenido de las matrices 1 y 2 una posición a la derecha
2 // - MD_MAX72XX::TSR -> desplazar a la derecha
3 mx.transform(1, 2, MD_MAX72XX::TSR );

```

**Tercera sobrecarga:** admite un solo parámetro y permite aplicarle una transformación a todas las matrices.



```
1 mxObj.transform(trans);
```

Donde:

- **trans**: tipo de transformación a aplicar.

```
1 // Desplazar el contenido de todas las matrices una posicion hacia arriba
2 mx.transform( MD_MAX72XX::TSU );
```

En la siguiente tabla tienes el conjunto de transformaciones que puedes realizar con la función *transform()*.

Valor	Transformación
MD_MAX72XX::TSL	Desplazar un píxel a la izquierda
MD_MAX72XX::TSR	Desplazar un píxel a la derecha
MD_MAX72XX::TSU	Desplazar un píxel hacia arriba
MD_MAX72XX::TSD	Desplazar un píxel hacia abajo
MD_MAX72XX::TFLR	Voltear de derecha a izquierda
MD_MAX72XX::TFUD	Voltear de arriba hacia abajo
MD_MAX72XX::TRC	Rotar 90 grados en sentido horario
MD_MAX72XX::TINV	Invertir píxeles

## Función setChar()

Esta función permite dibujar un carácter empezando en una columna específica.

```
1 byte w = mxObj.setChar(col, c);
```

Donde:

- **col**: columna donde comienza el carácter.
- **c**: carácter a mostrar.

- **w:** valor retornado por `setChar()`. Indica cuantas columnas ocupa el carácter dibujado.

```
1 // Dibujar el caracter 'A' comenzando en la columna 9 (segunda columna de la segunda
2 int ancho = mx.setChar(9, 'A');
```

## Función `update()`

La función `update()` actualiza el contenido de la matriz cuando la auto-actualización ha sido deshabilitada.

**Primera sobrecarga:** no recibe parámetros y actualiza el estado de todas las matrices conectadas.

```
1 mxObj.update();
```

**Segunda sobrecarga:** admite un parámetro y actualiza solo el estado de la matriz indicada.

```
1 mxObj.update(mat);
```

Donde:

- **mat:** es el número de la matriz a actualizar.

```
1 // actualizar la información de la matriz número 3
2 mx.update(2);
```

Ya conoces cómo conectar un módulo de **matriz de LED** con **Arduino** basados en **MAX7219** y cómo utilizar las funciones de la librería **MD\_MAX72xx** para controlarla, por lo tanto, ya estas en condiciones de empezar a trabajar.

# Como crear dibujos y animaciones en una matriz LED con Arduino

Primero vas a ver un caso simple donde solo se controla una **matriz de LEDs** utilizando un **Arduino Nano**. Con este ejercicio aprenderás a hacer gráficos simples y letras, así como a aplicarle transformaciones a la información de la matriz.

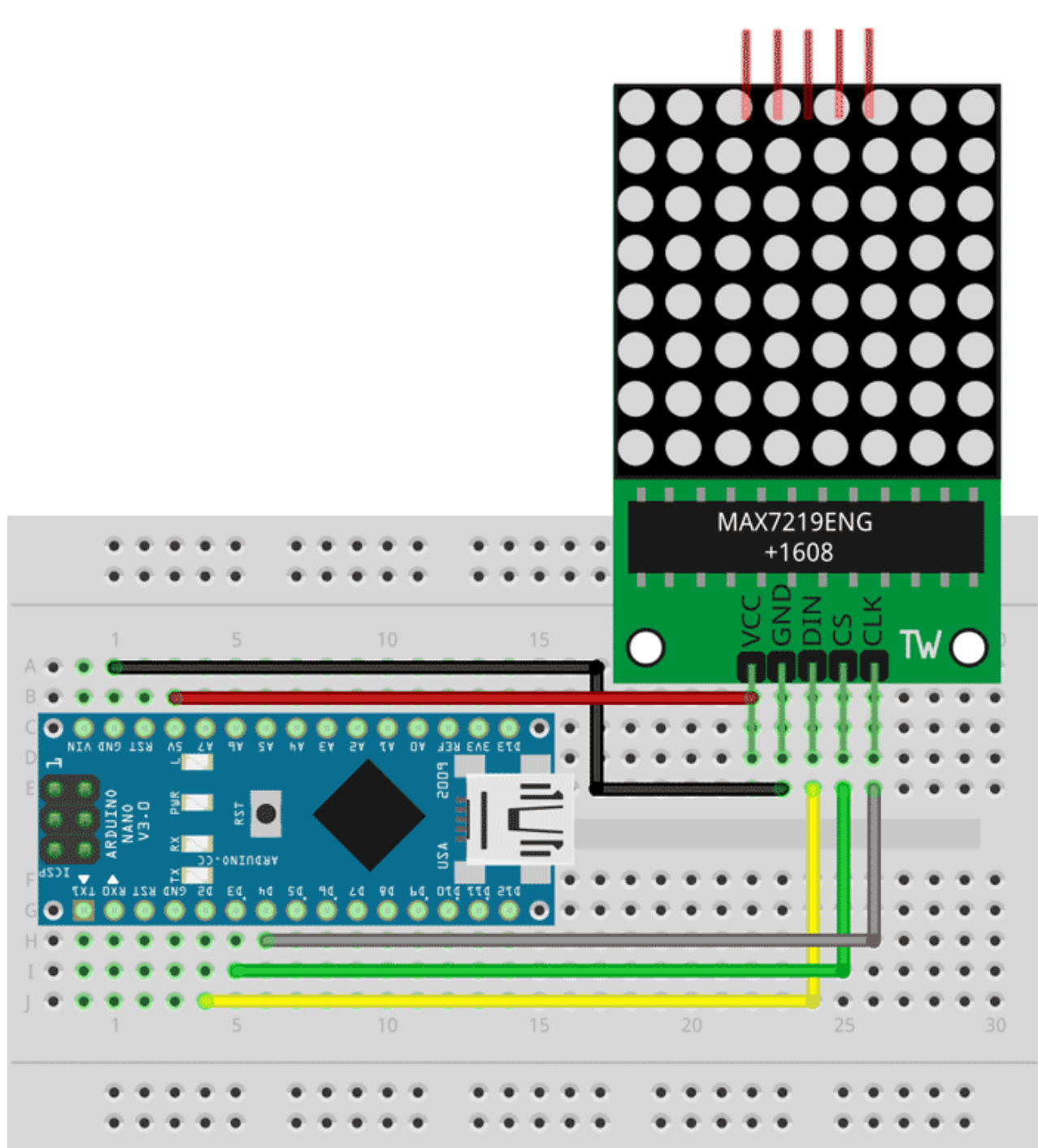
Para este ejemplo necesitarás:

- 1x **Arduino Nano** \*

- 1x **Módulo simple de matriz de LED.**
- 1x Placa de prototipos
- Cables para conexiones

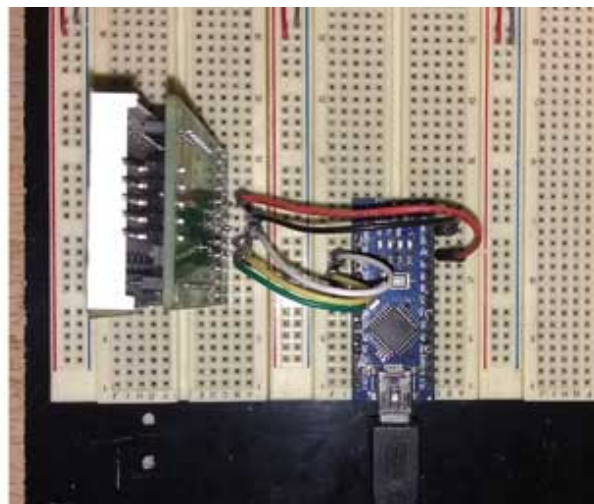
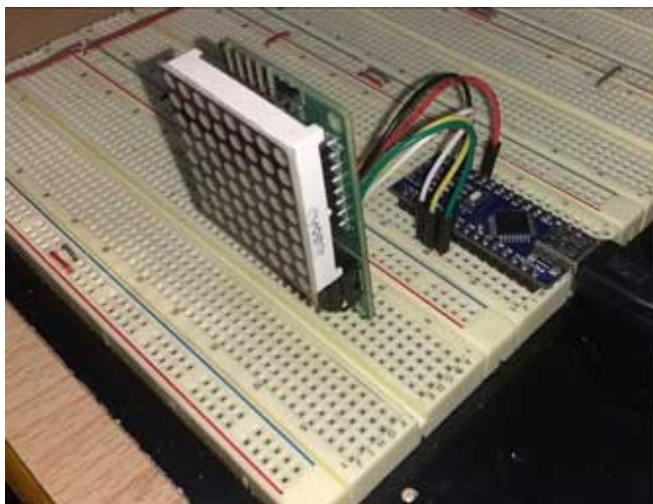
\* Te sirve cualquier placa Arduino de 5 voltios: **Arduino UNO** o **Arduino MEGA** son buenas opciones.

## Circuito eléctrico matriz LED con Arduino



fritzing

Montado en una placa de prototipos quedaría de la siguiente forma.



## Código matriz LED con Arduino

Teniendo en cuenta que se trata de un código relativamente extenso lo veras por partes.

### Librerías y constantes

El código comienza incluyendo la biblioteca *MD\_MAX72xx*.

```
1 #include <MD_MAX72xx.h>
```

Luego se definen un grupo de constantes para la creación de un objeto *MD\_MAX72XX* que permite controlar la matriz.

La constante *HARDWARE\_TYPE* es utilizada para indicar el tipo de matriz. Como yo he utilizado un módulo simple y lo he colocado como se muestra en la figura he tenido que darle el valor **MD\_MAX72XX::DR0CR0RR1\_HW**.

Si al ejecutar el código notas que los gráficos se muestran de lado o de cabeza solo tienes que cambiar este valor por otro de la lista que está en la descripción de la **función MD\_MAX72XX()**.

La constante **NUM\_OF\_MATRIX** es para indicar la cantidad de matrices. Una sola en este caso.

Las constantes **CLK\_PIN**, **DATA\_PIN** y **CS\_PIN** son para indicar los pines utilizados en la comunicación.

```
1 #define HARDWARE_TYPE MD_MAX72XX::DR0CR0RR1_HW
2 #define NUM_OF_MATRIX 1
3
4 #define CLK_PIN 4
```

```
5 #define DATA_PIN 2
6 #define CS_PIN 3
```

También se ha definido una constante **DELAY\_ANIMATION** para generar los retardos en las animaciones del ejemplo.

En este caso le he dado un valor de 200 ms pero lo puedes modificar para acelerar o ralentizar las animaciones.

```
1 #define DELAY_ANIMATION 200
```

Se han definido varios arreglos constantes. Cada uno de estos arreglos representa una imagen a dibujar en la matriz. Son utilizados por la **función *animateGhost()*** para generar algunas animaciones.

```
1 const byte ghost1[] = {0x18,0x7e,0xff,0xbd,0xff,0xff,0xff,0xa5};
2 const byte ghost2[] = {0x18,0x7e,0xbd,0xff,0xff,0xff,0xff,0xa5};
3 const byte ghost3[] = {0x18,0x7e,0xdb,0xff,0xff,0xff,0xff,0xa5};
4 const byte ghost4[] = {0x18,0x7e,0xff,0xdb,0xff,0xff,0xff,0xa5};
5 const byte ghost5[] = {0x18,0x7e,0x99,0x99,0xff,0xff,0xff,0xa5};
6 const byte ghost6[] = {0x3c,0x7e,0xd7,0xff,0xc3,0xff,0xff,0xdb};
7
```

## Variables

Se han declarado dos variables globales. La primera es un objeto **MD\_MAX72XX** llamado *mx* que utilizaremos para controlar la **matriz LED con Arduino**.

```
1 MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, NUM_OF_MATRIX);
```

El objeto creado utiliza los pines definidos por las constantes **CLK\_PIN**, **DATA\_PIN** y **CS\_PIN** para controlar la matriz.

Una segunda variable llamada *letter* se ha declarado. Esta es utilizada por la función ***animateChar()***.

```
1 char letter = 'a';
```

## Funciones

En la función ***setup()*** se inicializa el objeto *mx*.

```
1 mx.begin();
```

Una vez inicializado el objeto la matriz se pone en modo de prueba durante 2 segundos. Esto enciende todos los LEDs, permitiendo comprobar que ninguno esté dañado.

```
1 mx.control(MD_MAX72XX::TEST, true);
2 delay(2000);
3 // desactivando el modo de prueba
4 mx.control(MD_MAX72XX::TEST, false);
```

Luego se establece la intensidad de los LEDs a un valor de 5.

```
1 mx.control(MD_MAX72XX::INTENSITY, 5);
```

La función **loop()** ejecuta tres funciones que muestran el funcionamiento de la matriz:

- **misc()**: muestra cómo utilizar instrucciones sencillas
- **animateChar()**: muestra cómo imprimir un carácter y desplazarlo por la matriz.
- **animateGhost()**: muestra como realizar una animación simple a partir de varios gráficos.

```
1 void loop() {
2   misc();
3   animateChar();
4   animateGhost();
5 }
```

La función **misc()** comienza limpiando la matriz.

```
1 void misc(){
2   // limpiar la matriz
3   mx.clear();
```

Una vez la matriz está limpia se encienden los cuatro **LEDs** de las esquinas y se espera un tiempo.

```
1 // encender los LEDs de las esquinas
2 mx.setPoint( 0, 0, 1 );
3 mx.setPoint( 7, 7, 1 );
4 mx.setPoint( 0, 7, 1 );
5 mx.setPoint( 7, 0, 1 );
6 delay(DELAY_ANIMATION);
```

Luego se encienden los cuatro **LEDs** del centro y nuevamente se espera un tiempo.

```
1 mx.setPoint( 3, 3, 1 );
2 mx.setPoint( 3, 4, 1 );
3 mx.setPoint( 4, 3, 1 );
4 mx.setPoint( 4, 4, 1 );
5 delay(2*DELAY_ANIMATION);
```

Después se utiliza un ciclo para rellenar las columnas de izquierda a derecha encendiendo los **LEDs** alternadamente.



```
1  for( int i = 0; i < 8; i += 2 ){
2
3      // activar los LEDs alternos
4      mx.setColumn( i, 0xAA );
5
6      // esperar un tiempo
7      delay(DELAY_ANIMATION);
8
9      // estado invertido de la columna anterior
10     mx.setColumn( i+1, ~0xAA );
11
12     // esperar un tiempo
13     delay(DELAY_ANIMATION);
14 }
```

Antes de terminar, se utiliza la función ***mx.transform()*** con un ciclo para invertir los **LEDs** 12 veces.

```
1  delay(2*DELAY_ANIMATION);
2  // invertir varias veces
3  for( int i = 0; i < 12; i ++ ){
4      mx.transform(MD_MAX72XX::TINV);
5      delay(DELAY_ANIMATION);
6  }
7 } // misc end
```

En el siguiente video puedes ver el efecto que se obtiene si en la función ***loop()*** solo se ejecuta la función ***misc()***.

La función ***animateChar()*** muestra cómo desplazar un carácter por la matriz.

Lo primero que se realiza en la función es incrementar la variable ***letter***, de esta forma en cada llamada a la función se utilizará una letra diferente. También se utiliza una

instrucción *if* para garantizar que la variable siempre esté entre 'a' y 'z'.

```
1 void animateChar( ){
2   letter++;
3   if( letter > 'z' ){
4     letter = 'a';
5   }
```

Para obtener el efecto de desplazamiento se utiliza un ciclo *for*, de forma tal que en cada iteración, la letra se desplaza una posición a la izquierda. La posición de la letra es determinada por la variable local *i* (empleada en el control del ciclo).

Por lo tanto, en cada iteración se limpia la matriz y se dibuja la letra comenzando en la *i*-ésima columna.

```
1   for( int i = 0; i < 12; i++ ){
2     // limpiar la matriz
3     mx.clear();
4     // dibujar el caracter c empezando en la columna i
5     mx.setChar( i, letter );
6     // esperar un tiempo
7     delay(DELAY_ANIMATION);
8   }
9 } // animateChar end
```

En el siguiente video puedes ver el efecto conseguido si en la función ***loop()*** solo se ejecuta la función ***animateChar()***.

La función ***animateGhost()*** se utiliza para obtener una animación simple: uno de los fantasmas de PACMAN moviendo los ojos y dando vueltas.

Antes de comenzar la animación se apagan todos los bits de la **matriz**.

```
1 void animateGhost(){
2   // limpiar la pantalla
3   mx.clear();
```

La animación se realiza dibujando cada uno de los gráficos y generando un pequeño retardo entre gráficos. Para dibujar los gráficos se utilizó la función *drawRows()*, después veremos su implementación.

```
1 // dibujar gráfico 1
2 drawRows(ghost1);
3 delay(2*DELAY_ANIMATION);
4 // dibujar gráfico 2
5 drawRows(ghost2);
6 delay(2*DELAY_ANIMATION);
7 // dibujar gráfico 3
8 drawRows(ghost3);
9 delay(2*DELAY_ANIMATION);
10 // dibujar gráfico 4
11 drawRows(ghost4);
12 delay(2*DELAY_ANIMATION);
13 // dibujar gráfico 5
14 drawRows(ghost5);
15 delay(2*DELAY_ANIMATION);
16 // dibujar gráfico 6
17 drawRows(ghost6);
18 delay(2*DELAY_ANIMATION);
```

Al final de la función se utiliza la **función *transform()*** para rotar la figura varias veces.

```
1 // rotar varias veces la matriz.
2 for( int i = 0; i < 8; i++ ){
3   mx.transform( MD_MAX72XX::TRC );
4   delay(DELAY_ANIMATION*3);
5 }
6 }
```

La **función *drawRows()*** recibe un arreglo de 8 bytes como parámetro y dibuja la matriz a partir de ese arreglo. Para eso cada posición del arreglo se corresponde con una fila de la matriz donde los bits puestos a 1 indican cuáles LEDs deben encenderse.

```
1 void drawRows( const byte fig[] ){
2   for( int i = 0; i < 8; i++ ){
3     mx.setRow(0, i, fig[i]);
4   }
5 }
```

En el siguiente video puedes ver el efecto obtenido al ejecutar la **función *animateGhost()***.

Aquí te dejo el código de la aplicación completa. Siente libre de experimentar como tu quieras.

```

1 //ejemplo para crear animaciones en módulo matriz LED con Arduino basado en MAX7219
2 #include <MD_MAX72xx.h>
3
4 #define HARDWARE_TYPE MD_MAX72XX::DR0CR0RR1_HW
5 #define NUM_OF_MATRIX 1
6
7 #define CLK_PIN 4
8 #define DATA_PIN 2
9 #define CS_PIN 3
10
11 #define DELAY_ANIMATION 200
12
13 const byte ghost1[] = {0x18,0x7e,0xff,0xbd,0xff,0xff,0xff,0xa5};
14 const byte ghost2[] = {0x18,0x7e,0xbd,0xff,0xff,0xff,0xff,0xa5};
15 const byte ghost3[] = {0x18,0x7e,0xdb,0xff,0xff,0xff,0xff,0xa5};
16 const byte ghost4[] = {0x18,0x7e,0xff,0xdb,0xff,0xff,0xff,0xa5};
17 const byte ghost5[] = {0x18,0x7e,0x99,0x99,0xff,0xff,0xff,0xa5};
18 const byte ghost6[] = {0x3c,0x7e,0xd7,0xff,0xc3,0xff,0xff,0xdb};
19
20 MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, NUM_OF_MATRIX)
21
22 char letter = 'a';
23
24 void setup() {
25     // inicializar el objeto mx
26     mx.begin();
27
28     // poniendo la matriz en modo de prueba
29     mx.control(MD_MAX72XX::TEST, true);
30     delay(2000);
31
32     // desactivando el modo de prueba
33     mx.control(MD_MAX72XX::TEST, false);
34
35     // Establecer intensidad a un valor de 5
36     mx.control(MD_MAX72XX::INTENSITY, 5);
37

```

```

38 }
39
40 /**
41  * Recibe un arreglo con la configuración de
42  * las filas y las dibuja en la matriz
43  */
44 void drawRows( const byte fig[] ){
45     for( int i = 0; i < 8; i++ ){
46         mx.setRow(0, i, fig[i]);
47     }
48 }
49
50 /**
51  * Muestra animación con fantasma de PACMAN
52  */
53 void animateGhost(){
54     // limpiar la pantalla
55     mx.clear();
56
57     // dibujar gráfico 1
58     drawRows(ghost1);
59     delay(2*DELAY_ANIMATION);
60     // dibujar gráfico 2
61     drawRows(ghost2);
62     delay(2*DELAY_ANIMATION);
63     // dibujar gráfico 3
64     drawRows(ghost3);
65     delay(2*DELAY_ANIMATION);
66     // dibujar gráfico 4
67     drawRows(ghost4);
68     delay(2*DELAY_ANIMATION);
69     // dibujar gráfico 5
70     drawRows(ghost5);
71     delay(2*DELAY_ANIMATION);
72     // dibujar gráfico 6
73     drawRows(ghost6);
74     delay(2*DELAY_ANIMATION);
75
76     // rotar varias veces la matriz.
77     for( int i = 0; i < 8; i++ ){
78         mx.transform( MD_MAX72XX::TRC );
79         delay(DELAY_ANIMATION*3);
80     }
81 }
82 }
83
84 /*
85  * Desplazar un caracter de derecha a izquierda
86  */
87 void animateChar( ){
88
89     letter++;
90     if( letter > 'z' ){
91         letter = 'a';
92     }
93
94     for( int i = 0; i < 12; i++ ){
95         // limpiar la matriz
96         mx.clear();
97         // dibujar el caracter c empezando en la columna i
98         mx.setChar( i, letter );
99         // esperar un tiempo
100        delay(DELAY_ANIMATION);
101    }
102 }
103 }
104
105 void misc(){
106     // limpiar la matriz

```

```

107  mx.clear();
108
109  // encender los LEDs de las esquinas
110  mx.setPoint( 0, 0, 1 );
111  mx.setPoint( 7, 7, 1 );
112  mx.setPoint( 0, 7, 1 );
113  mx.setPoint( 7, 0, 1 );
114  delay(DELAY_ANIMATION);
115
116  // encender los LEDs del centro
117  mx.setPoint( 3, 3, 1 );
118  mx.setPoint( 3, 4, 1 );
119  mx.setPoint( 4, 3, 1 );
120  mx.setPoint( 4, 4, 1 );
121  delay(2*DELAY_ANIMATION);
122
123  // repetir 4 veces
124  for( int i = 0; i < 8; i += 2 ){
125
126      // activar los LEDs alternos
127      mx.setColumn( i, 0xAA );
128
129      // esperar un tiempo
130      delay(DELAY_ANIMATION);
131
132      // estado invertido de la columna anterior
133      mx.setColumn( i+1, ~0xAA );
134
135      // esperar un tiempo
136      delay(DELAY_ANIMATION);
137  }
138
139  // esperar el doble del tiempo de animacion
140  delay(2*DELAY_ANIMATION);
141
142  // invertir varias veces
143  for( int i = 0; i < 12; i ++ ){
144      mx.transform(MD_MAX72XX::TINV);
145      delay(DELAY_ANIMATION);
146  }
147 }
148
149 void loop() {
150     misc();
151
152     animateChar();
153
154     animateGhost();
155 }
156

```

## Scroll de mensajes con varias matrices de LED con Arduino

Aquí veras como utilizar un conjunto de matrices para formar un pequeño **cartel LED**. El cartel permite visualizar un mensaje que se desliza de derecha a izquierda.

El mensaje puede ser modificado mediante el puerto serie sin necesidad de volver a cargar el código al **Arduino**.

Para este proyecto necesitarás:



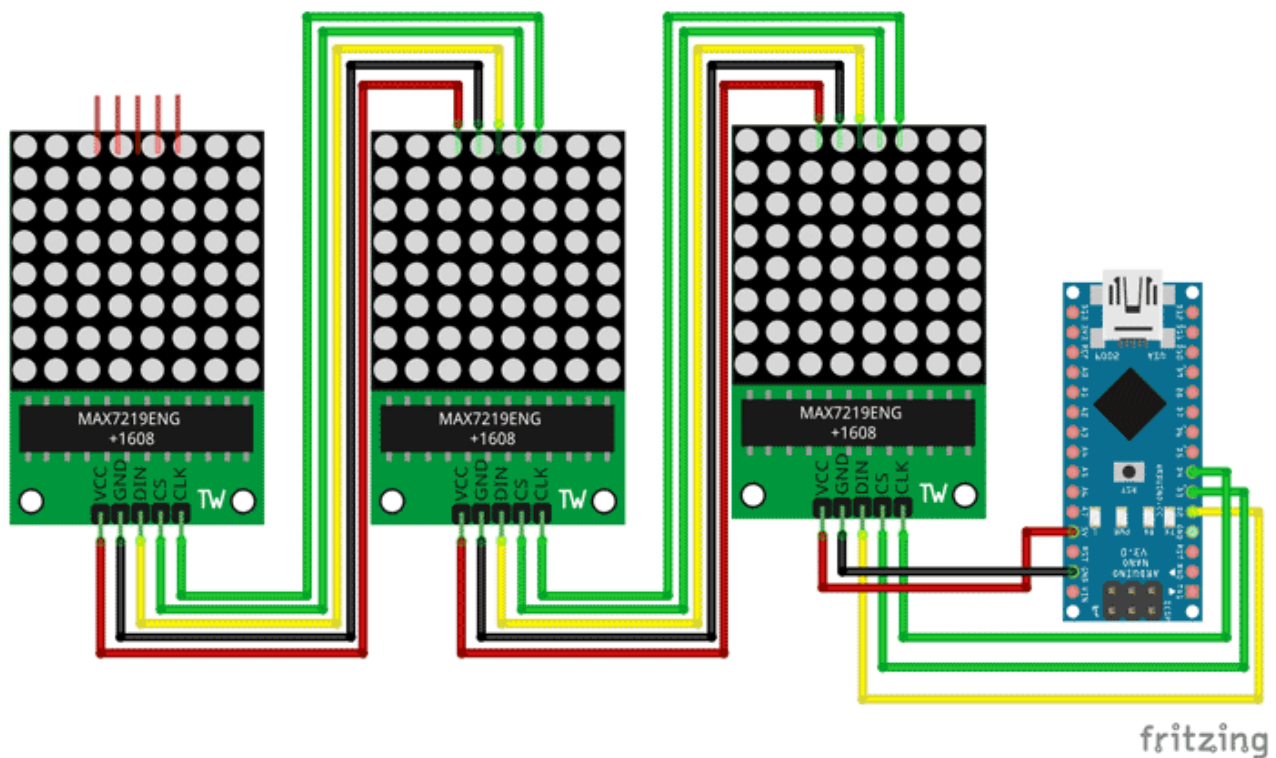
- 1x **Arduino Nano** \*
- 3x **Matrices de LEDs** \*\*
- 1x Placa de prototipos
- Cables para las conexiones

\* Puedes utilizar cualquier otra placa Arduino de 5 voltios.

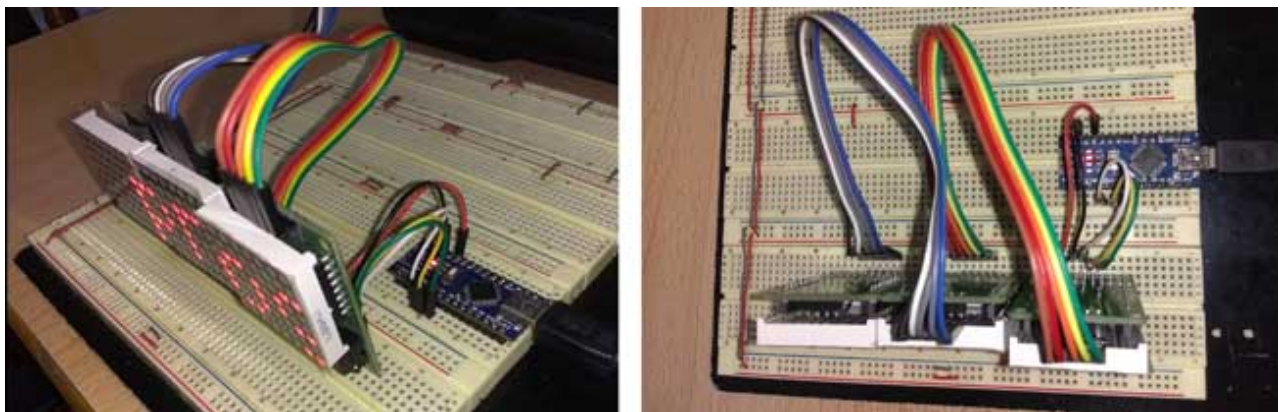
\*\* Yo solo he utilizado tres matrices porque es lo que tengo a mano, pero tú puedes utilizar más si quieres.

## Circuito eléctrico matrices LEDs con Arduino

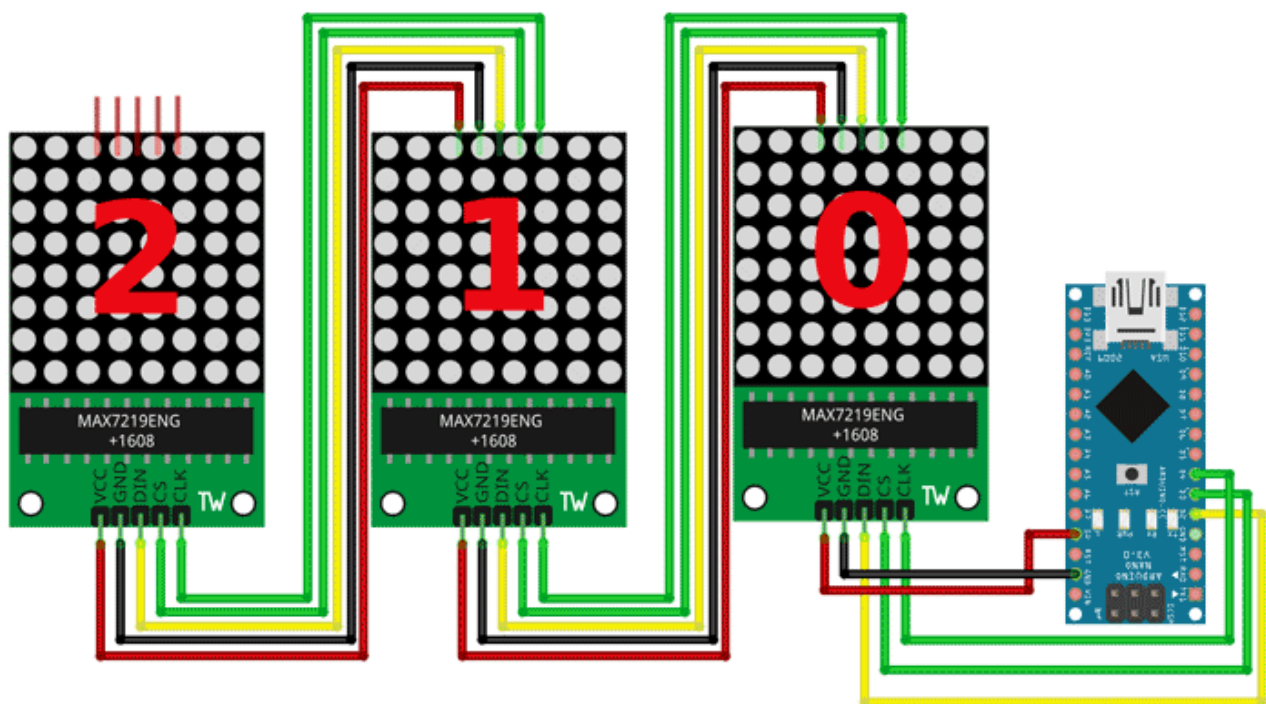
En la imagen puedes ver el esquema necesario para montar este proyecto.



El montaje en placa de prototipo quedaría como se muestra en la siguiente imagen.

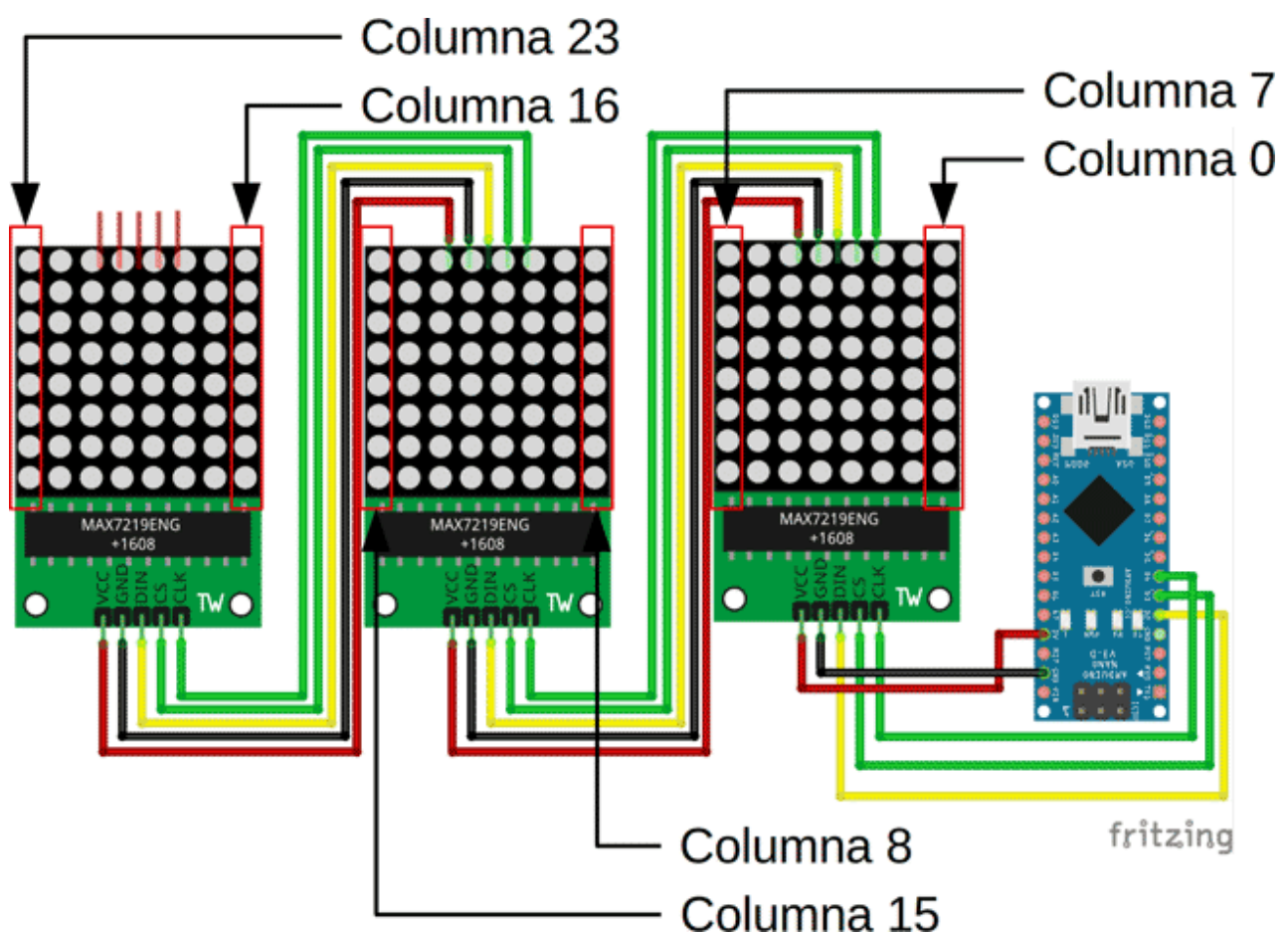


Cuando conectamos varias matrices en cascada como lo estamos haciendo en este proyecto las matrices se enumeran de derecha a izquierda.



fritzing

De igual manera las columnas son enumeradas de derecha a izquierda: de forma tal que la primera matriz ocupa las columnas 0-7, la segunda ocupa las columnas 8-15 y así sucesivamente.



fritzing

## Código matrices LED con Arduino

El código implementado es un tanto largo así que lo veras por partes.

## Librerías y constantes

Comenzamos igual que siempre, incluyendo las librerías. En este caso solo es necesario incluir la **librería MD\_MAX72xx**.

```
1 //Ejemplo matrices LEDs con Arduino y MAX7219
2 #include <MD_MAX72xx.h>
```

Después se han declarado las constantes para la definición del objeto **MD\_MAX72XX**.

```
1 #define HARDWARE_TYPE MD_MAX72XX::DR0CR0RR1_HW
2 #define NUM_OF_MATRIX 3
3
4 #define CLK_PIN 4
5 #define DATA_PIN 2
6 #define CS_PIN 3
```

Como puedes ver las constantes son similares al ejemplo anterior. Solo se ha modificado **NUM\_OF\_MATRIX**, ya que en este caso se utilizan 3 matrices.

Si piensas utilizar una cantidad de matrices diferentes debes cambiar el valor de esta constante por el de la cantidad de matrices que utilices.

## Variables globales

Se declara un objeto de tipo **MD\_MAX72XX** utilizando los valores de las constantes previamente definidas.

```
1 MD_MAX72XX cartel = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, NUM_OF_MATRI
```

Se utilizan además dos variables tipo **String**:

- **mensaje**: donde se almacena el mensaje a mostrar.
- **proximo\_mensaje**: utilizada como almacenamiento temporal cuando se actualiza el mensaje desde el puerto serie.

```
1 String mensaje = "programarfacil.com - Arduino y ESP";
2 String proximo_mensaje;
```

La variable **mensaje** se inicializa con un texto para que al energizar el Arduino comience mostrándolo.

## Funciones

La función **setup()** se encarga de inicializar las matrices y el puerto serie. En este caso la intensidad de los **LEDs** la he configurado a un valor de 10 unidades (recuerda que 15 es el máximo) y también se ha desactivado la actualización automática.

```
1 void setup() {  
2   // inicializar el objeto mx  
3   cartel.begin();  
4  
5   // Establecer intensidad a un valor de 5  
6   cartel.control(MD_MAX72XX::INTENSITY, 10);  
7  
8   // Desactivar auto-actualizacion  
9   cartel.control( MD_MAX72XX::UPDATE, false );  
10 }
```

Finalmente el puerto serie está configurado a 9600 baudios.

```
1 Serial.begin(9600);  
2 }  
3
```

La función **loop()** ejecuta las funciones **slide\_text()** para mostrar el texto y **actualizar\_mensaje()** para recibir un nuevo mensaje.

```
1 void loop() {  
2   slide_text( 150 );  
3   actualizar_mensaje();  
4 }
```

A parte de las funciones **setup()** y **loop()** se han implementado tres funciones para obtener un código más organizado:

- **printText()**: imprime un texto en las matrices comenzando en la columna indicada.
- **slide\_text()**: muestra el valor de la variable global *mensaje* desplazándose por la matriz de derecha a izquierda.
- **actualizar\_mensaje()**: comprueba el puerto serie y si existe algún mensaje nuevo actualiza el valor de la variable *mensaje*.

Veamos ahora los detalles de implementación de cada una.

La función **printText()** recibe dos parámetros:

- **pos**: a partir de qué columna se mostrará el texto.
- **text**: el texto a mostrar.

Para imprimir el mensaje se realiza un ciclo que pasa por cada letra del texto.

En cada iteración se realiza una llamada a la función **cartel.setChar( pos, text[i] )** para imprimir ese carácter. El valor retornado por la función **setChar()** es restado a **pos** para determinar la posición de la próxima letra.

```
1 int printText(int pos, const String text){
2   int w;
3
4   for( int i = 0; i < text.length(); i++ ){
5     // imprimir letra
6
7     w = cartel.setChar( pos, text[i] );
8     // la proxima letra empieza donde termina esta
9     pos = pos - w;
```

Adicionalmente se deja una columna en blanco para separar las letras.

```
1   cartel.setColumn(pos, B00000000);
2   pos = pos - 1;
```

Se comprueba si **pos** es negativo. En caso de que lo sea se termina el ciclo, ya que no tiene sentido intentar escribir en columnas que no existen :).

```
1   if( pos < 0 )
2     break;
3 }
```

Finalmente, se ejecuta la instrucción **cartel.update()** para actualizar el contenido de la pantalla y se retorna el valor de **pos**, es decir, que se retorna la columna donde se terminó el texto.

```
1   cartel.update();
2   return pos;
3 }
```

Si no se ejecuta la instrucción **cartel.update()** el texto no es mostrado en las matrices, ya que en la función **setup()** se deshabilitó la actualización automática.

La función **slide\_text()** es la encargada de hacer que el texto se desplace por las matrices. Esta función acepta como parámetro entero que indica cuán rápido se desplazará el texto.

Además, utiliza tres variables locales en su funcionamiento.



```

1 void slide_text(int ms_delay){
2   int col = 0;
3   int last_pos;
4   bool completo = false;

```

Lo primero que hace la función es borrar todo el contenido de las matrices.

```

1 cartel.clear();

```

Luego se ejecuta un ciclo mientras la variable **completo** es *false*. En cada iteración del ciclo ocurre lo siguiente:

- Se ejecuta la función **printText()** y el valor retornado por esta se almacena en **last\_pos**.
- Se espera un tiempo.
- Incrementa variable **col**.
- Se comprueba que la última columna utilizada por la función **printText()** sea mayor que la cantidad total de columnas:
  - En caso afirmativo se asigna *true* a la variable **completo** para interrumpir el ciclo.

```

1 while( !completo ){
2   last_pos = printText(col, mensaje);
3   delay(ms_delay);
4   col++;
5   if( last_pos > (int)cartel.getColumnCount() )
6     completo = true;
7 }
8 }

```

La función **actualizar\_mensaje()** lee caracteres del puerto serie y los almacenan en **proximo\_mensaje** hasta encontrar un carácter de fin de línea ('\n').

Cuando se alcanza el carácter de fin de línea el valor de **proximo\_mensaje** es asignado a la variable **mensaje**.

```

1 void actualizar_mensaje(){
2   while( Serial.available() ){
3     char c = Serial.read();
4
5     if( c == '\n' ){
6       mensaje = proximo_mensaje;
7       proximo_mensaje = "";
8       break;
9     }
10    else
11      proximo_mensaje += c;
12  }
13 }

```



Bueno eso es todo. Aquí te dejo el código completo para que lo puedas cargar al **Arduino**.

```

1  //Ejemplo para enviar mensajes deslizantes en módulos de matrices LEDs con Arduino b
2  #include <MD_MAX72xx.h>
3  #define HARDWARE_TYPE MD_MAX72XX::DR0CR0RR1_HW
4  #define NUM_OF_MATRIX 3
5  #define CLK_PIN 4
6  #define DATA_PIN 2
7  #define CS_PIN 3
8
9  MD_MAX72XX cartel = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, NUM_OF_MATR
10
11  String mensaje = "programarfacil.com - Arduino y ESP";
12
13  String proximo_mensaje;
14
15  void setup() {
16      // inicializar el objeto mx
17      cartel.begin();
18
19      // Establecer intensidad a un valor de 5
20      cartel.control(MD_MAX72XX::INTENSITY, 10);
21
22      // Desactivar auto-actualizacion
23      cartel.control( MD_MAX72XX::UPDATE, false );
24
25      // inicializar puerto Serie a 9600 baudios
26      Serial.begin(9600);
27  }
28  void loop() {
29      slide_text( 150 );
30      actualizar_mensaje();
31  }
32
33  void actualizar_mensaje(){
34
35      while( Serial.available() ){
36
37          char c = Serial.read();
38
39          if( c == '\n' ){
40              mensaje = proximo_mensaje;
41              proximo_mensaje = "";
42              break;
43          }
44          else
45              proximo_mensaje += c;
46      }
47  }
48  }
49
50  void slide_text(int ms_delay){
51      int col = 0;
52      int last_pos;
53      bool completo = false;
54
55      cartel.clear();
56
57      while( completo == false ){
58          last_pos = printText(col, mensaje);
59          delay(ms_delay);
60          col++;
61          if( last_pos > (int)cartel.getColumnCount() )
62              completo = true;
63      }

```

```
64 }  
65  
66 int printText(int pos, const String text){  
67     int w;  
68  
69     for( int i = 0; i < text.length(); i++ ){  
70         // imprimir letra  
71  
72         w = cartel.setChar( pos, text[i] );  
73         // la proxima letra empieza donde termina esta  
74         pos = pos + w;  
75         // Se deja una columna entre letras.  
76         cartel.setColumn(pos, B00000000);  
77  
78         pos = pos + 1;  
79  
80         if( pos > 7 )  
81             break;  
82     }  
83     cartel.update();  
84  
85     return pos;  
86 }  
87  
88
```

En el siguiente video puedes ver el resultado obtenido al cargar el código al **Arduino**.

## Conclusiones matriz LED con Arduino MAX7219

Solo queda que interactues con el contenido. Implementa un cartel, un reloj digital o dibuja animaciones con movimiento en tus **matriz de LED con Arduino basados en el integrado MAX7219**.

Solo un consejo final. Si planeas utilizar muchas matrices, por ejemplo, más de cuatro, es necesario que utilices una fuente de voltaje externa. De lo contrario el **regulador de**

**voltaje del Arduino** se calentará mucho, corriendo el riesgo de dañar la placa irreversiblemente.

Aquí te dejo una lista de proyectos que puedes hacer utilizando **matrices LED con Arduino y MAX7219**:

- [Reloj Digital con matrices](#)
- [Juego retro de conducción de coches](#)
- [Dado electrónico con Arduino](#) y MAX7219
- [Snake](#)

Para realizar este tipo de juegos vas a necesitar también un [joystick con Arduino](#).

Si te animas a realizar alguno de estos proyectos, déjame un comentario aquí abajo.  
Gracias :)

Gracias a [Shutterstock](#) por la cesión de las imágenes.

---

#### ALSO ON PROGRAMAR FÁCIL CON ARDUINO

##### Zigbee y Zigbee2MQTT

hace 2 años · 4 comentarios

Zigbee es un protocolo privativo usado para vincular dispositivos inteligentes. ...

##### LCD con Arduino texto en movimiento paso a ...

hace 3 años · 2 comentarios

En este artículo te enseñé como utilizar un LCD con Arduino paso a paso. ...


##### Servidor web con ESP32

hace 2 años · 7 comentarios

No tendrá la potencia de servidores de la mismísima Alphabet, pero sin duda ...

##### Kit de herramientas

hace 3 años

 Incursión maravillosa es imposible