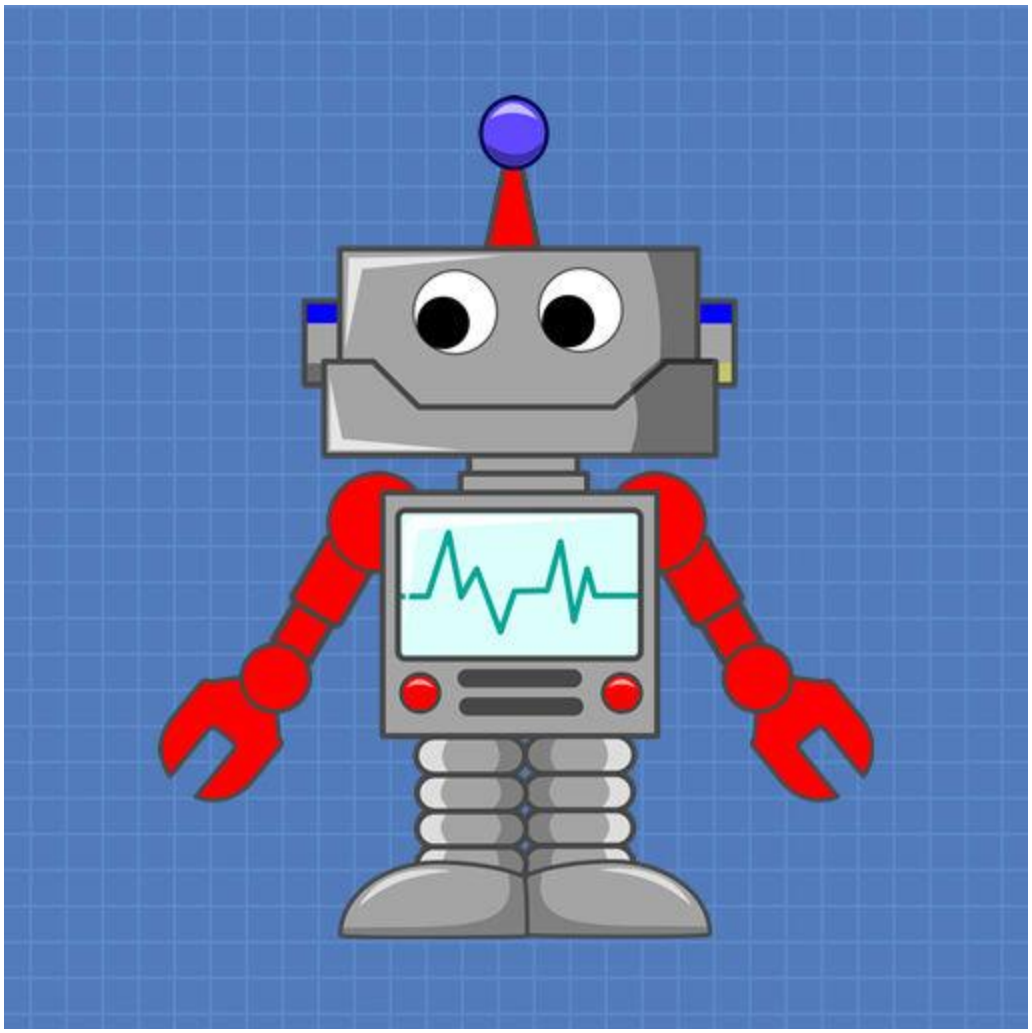
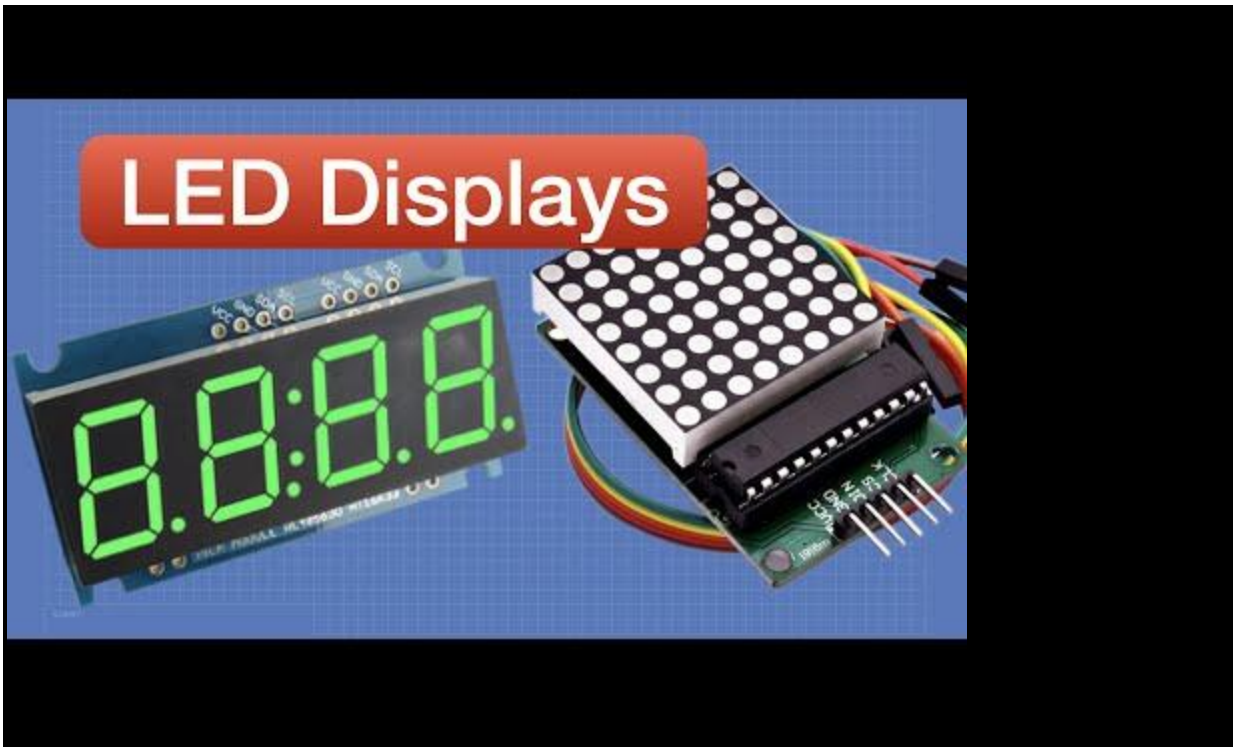
LED Displays with Arduino



DroneBot Workshop Tutorial

https://dronebotworkshop.com

Yes, they have been around forever, but LEDs still make great display devices. Today we will see how to work with both 7-segment and dot matrix LED displays and an Arduino Uno.



Introduction

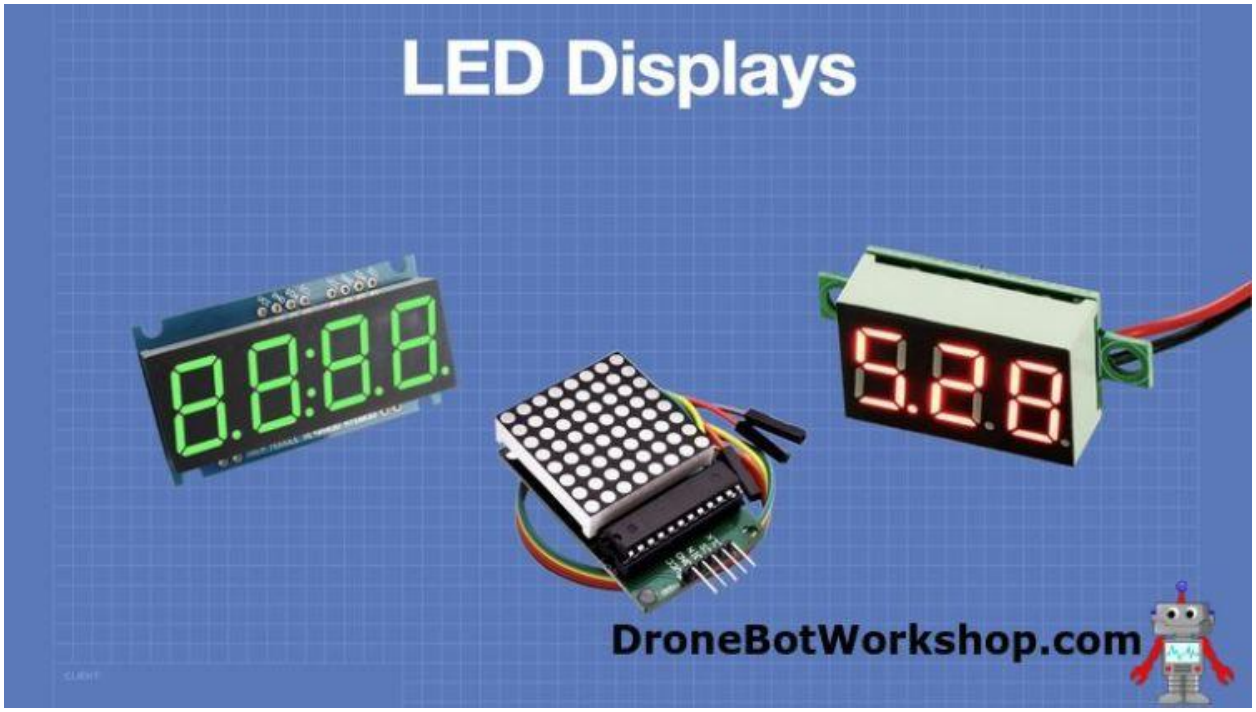
We have worked with several types of displays before, including Liquid Crystal Displays (LCDs) and Organic Light-Emitting Diodes (OLEDs). Each of them has advantages and disadvantages in terms of cost, performance, and ease of use.

Today we are getting back to basics and working with LED displays. Despite having been around for over half a century, LED displays are as popular as ever.

These make great displays for hobbyist projects, as they are inexpensive and pretty easy to work with. Today, I'll show you how you can add LED displays to your next Arduino project.

LED Displays

Light Emitting Diodes, or LEDs, have been around since 1962, when they were developed by Nick Holonyak at General Electric.



During the 1960s LEDs were mostly laboratory curiosities, but in 1969 Hewlett Packard developed an LED equivalent of a Nixie tube, which was a popular display type that used filament with patterns of characters.

Cost reductions and manufacturing improvements led to LEDs taking the market by storm in the mid-1970s, bringing consumers a new world of "digital" appliances like calculators and watches.

While the original LED displays were red and green, they are now available in a variety of colors. And they come in sizes ranging from almost microscopic to several inches tall.

Display Types

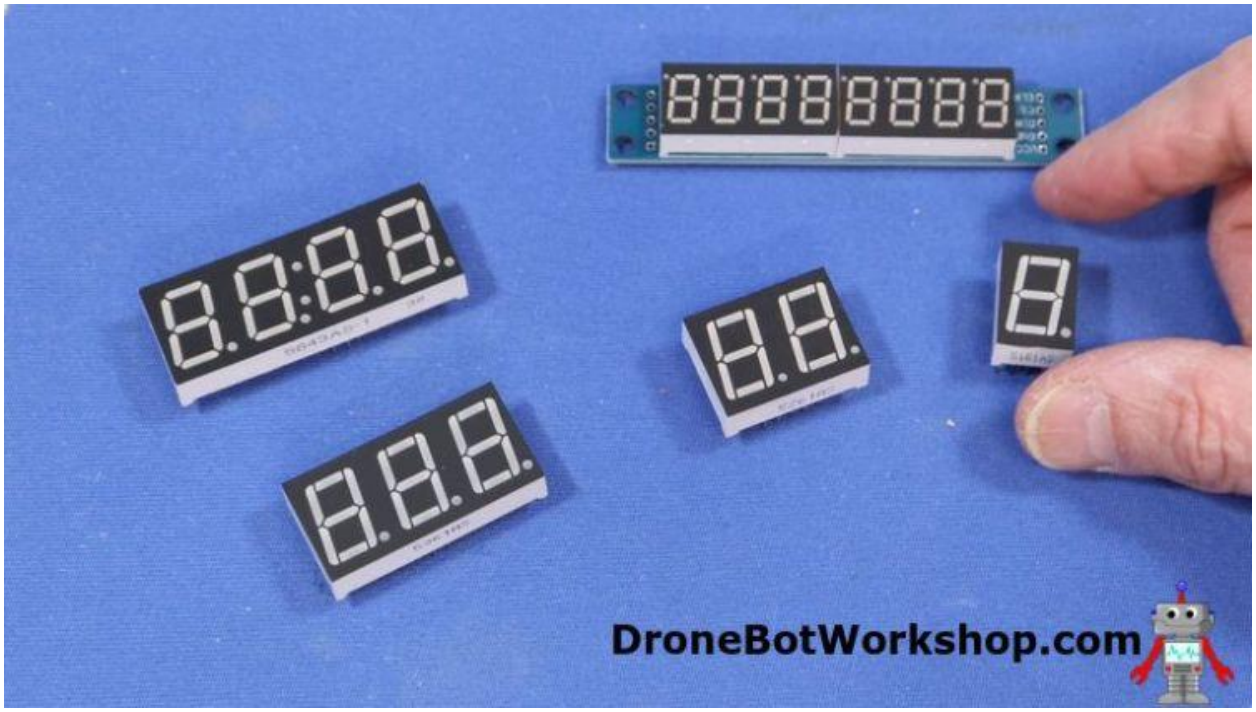
We can break down our LED displays into two basic types:

- 7-Segment Displays – These use seven segments to form a number (and a few letters). The name is a bit of a misnomer, as most of these actually have an eighth segment for a decimal point.
- Dot-Matrix Displays – These displays consist of a grid of LEDs arranged in a matrix. They can display numbers, text (upper and lowercase), and custom characters.

We will be working with both types today.

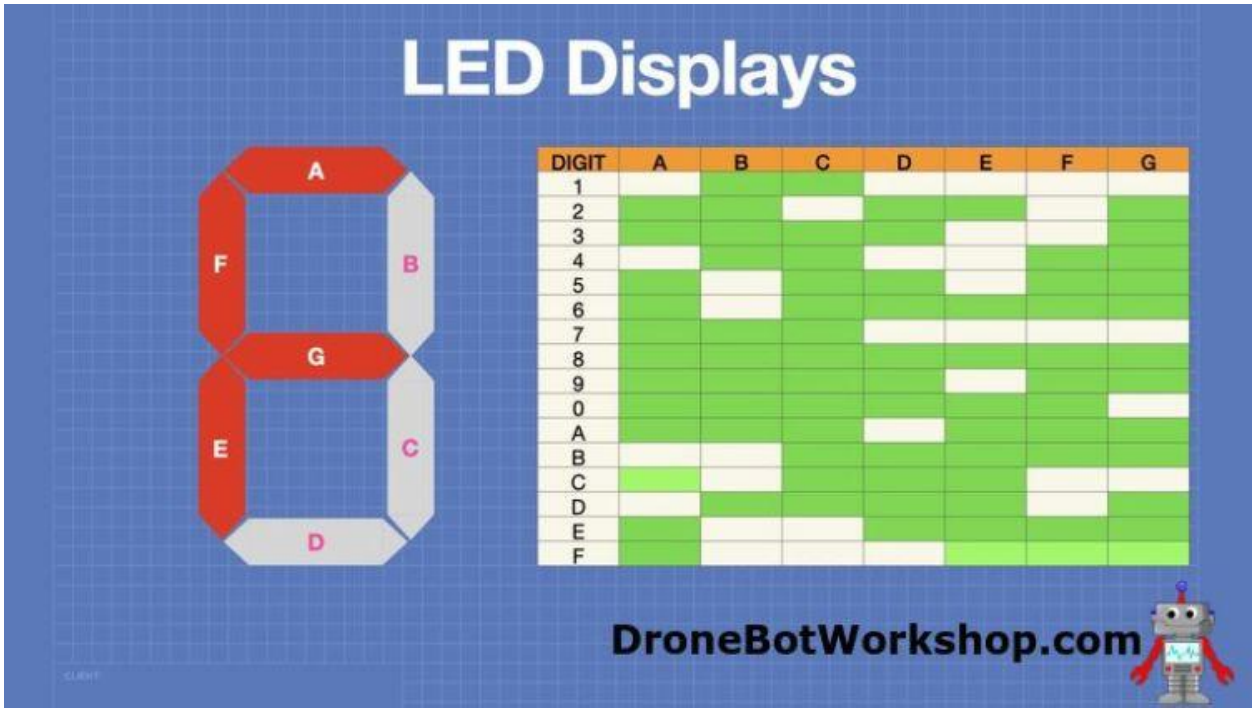
7-Segment Displays

The 7-Segment Display is a very common way of displaying numerical information. You see them everywhere, from digital clocks to ovens.



Although they might seem to be a product of the semiconductor age, 7-segment displays can be traced back to the early 1900s, when mechanical representations were used. In 1904 a patent was granted for sending a 7-segment code via telegraph, and in 1910 an incandescent version of the 7-segment display was used in telephone operator centers.

These displays consist of seven elements, labels a to g, with each element being an LED. By lighting up specific patterns of the display, you can make any numerical character, as well as a few crude representations of letters.



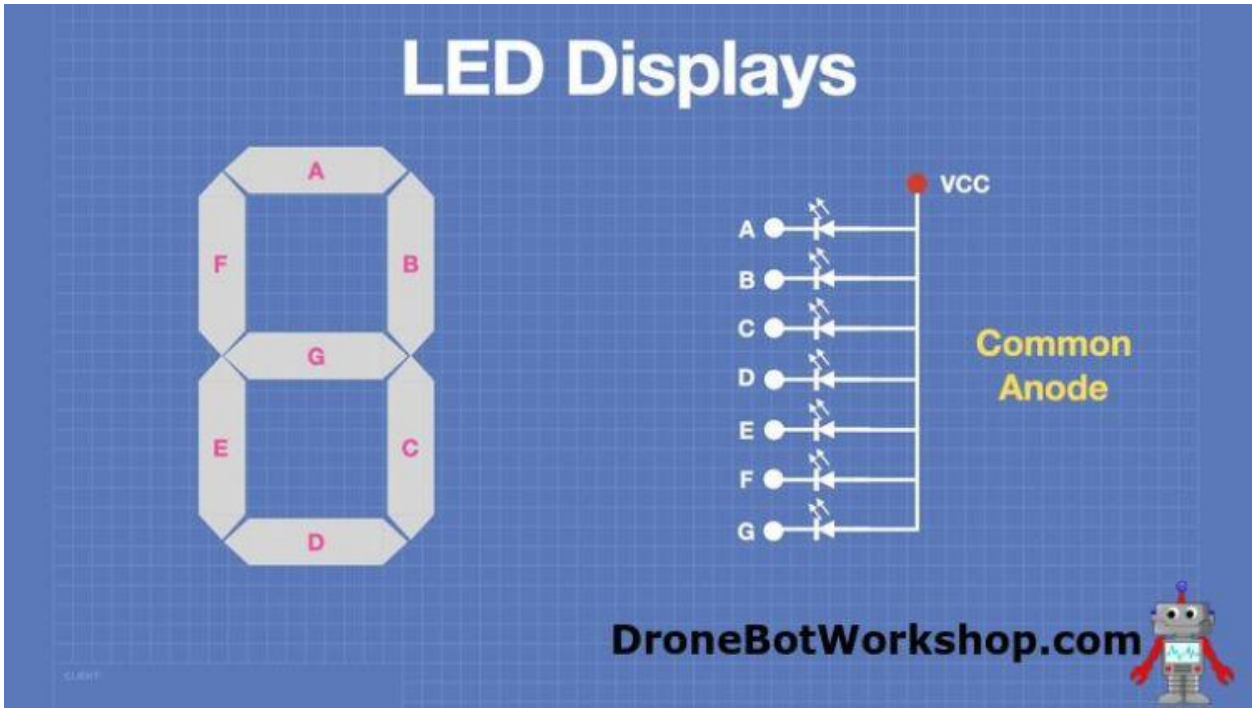
Other variations of this display exist, including ones with more segments to allow alpha characters as well. But the 7-segment variety is by far the most common, and it's what we will be using today.

Common Cathode vs Common Anode

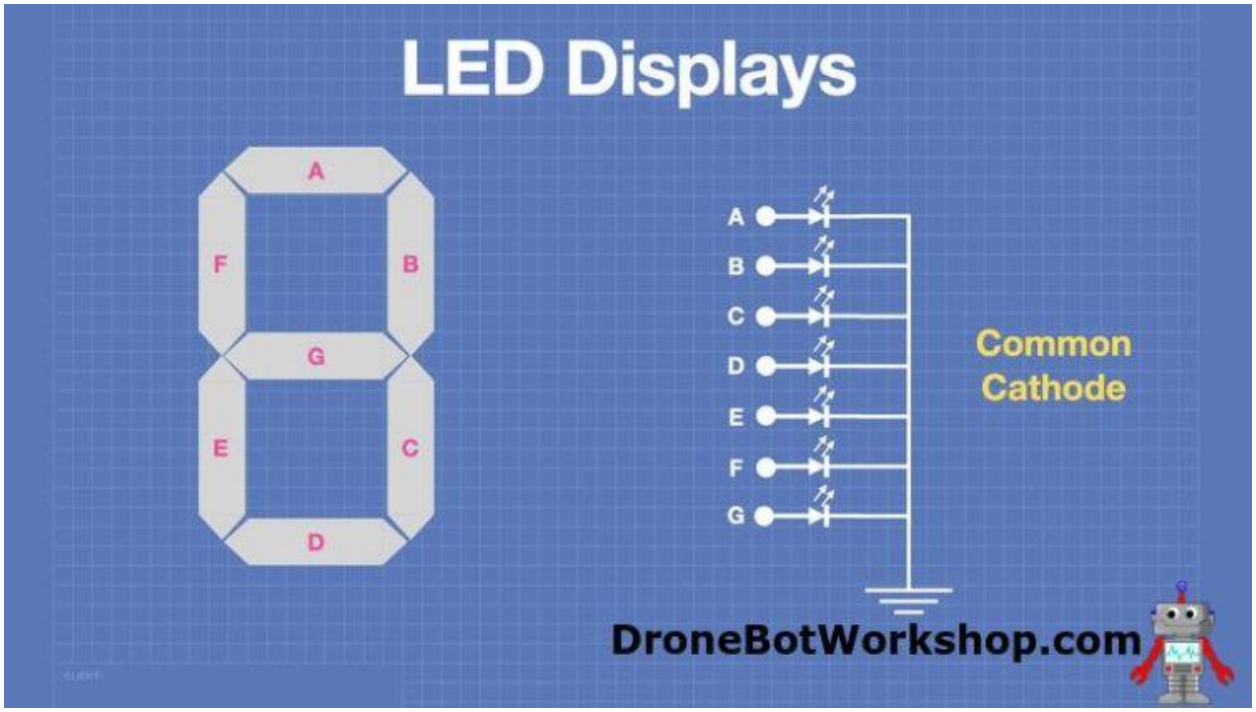
The eight LEDs (seven segments plus a decimal point) in the display would require a total of 16 wires for connections. In order to reduce this to a more manageable number, one side of each LED is tied to a common bus.

This technique reduces the number of connections to nine, the eight segments plus a common connection.

You can get 7-segment LED displays with either a Common Anode or Common Cathode configuration.



In a Common Anode LED display, the common line is connected to VCC, usually 3.3 or 5 volts. Grounding the other side (through a dropping resistor) will light the display element.



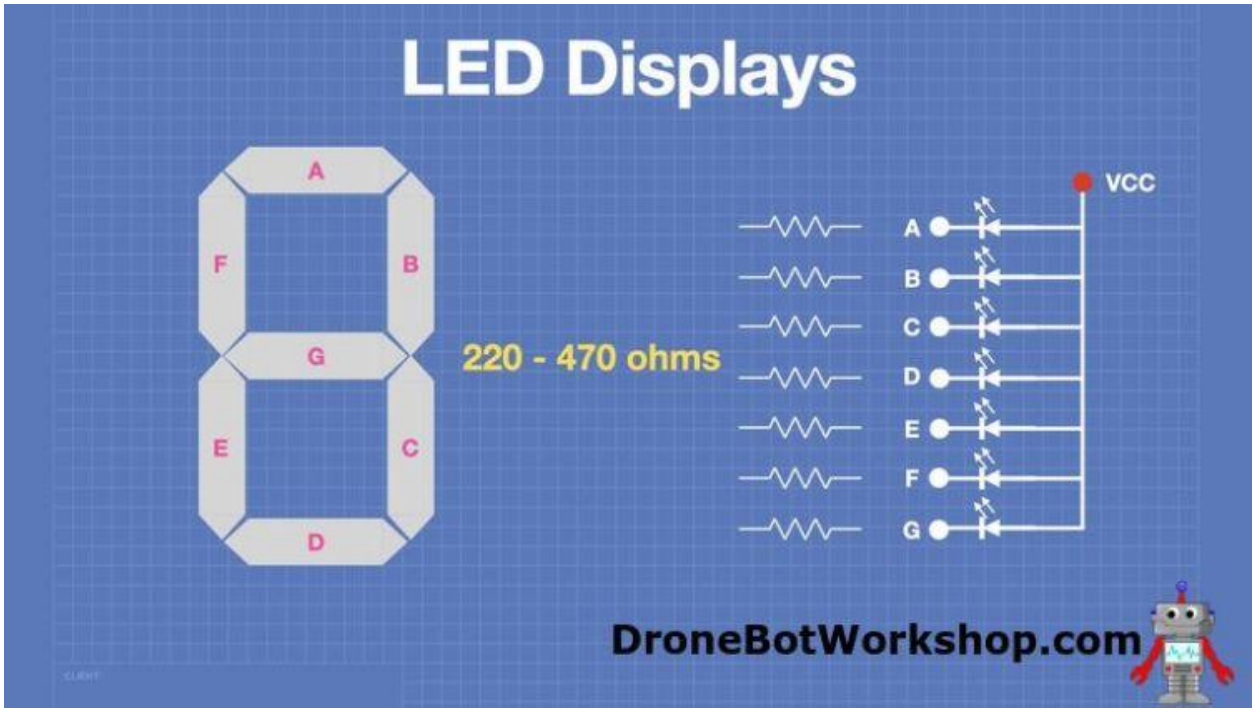
For a Common Cathode device, we hook the common side to ground and apply a positive voltage to each element to light it. This is usually an easier arrangement.

Dropping Resistors

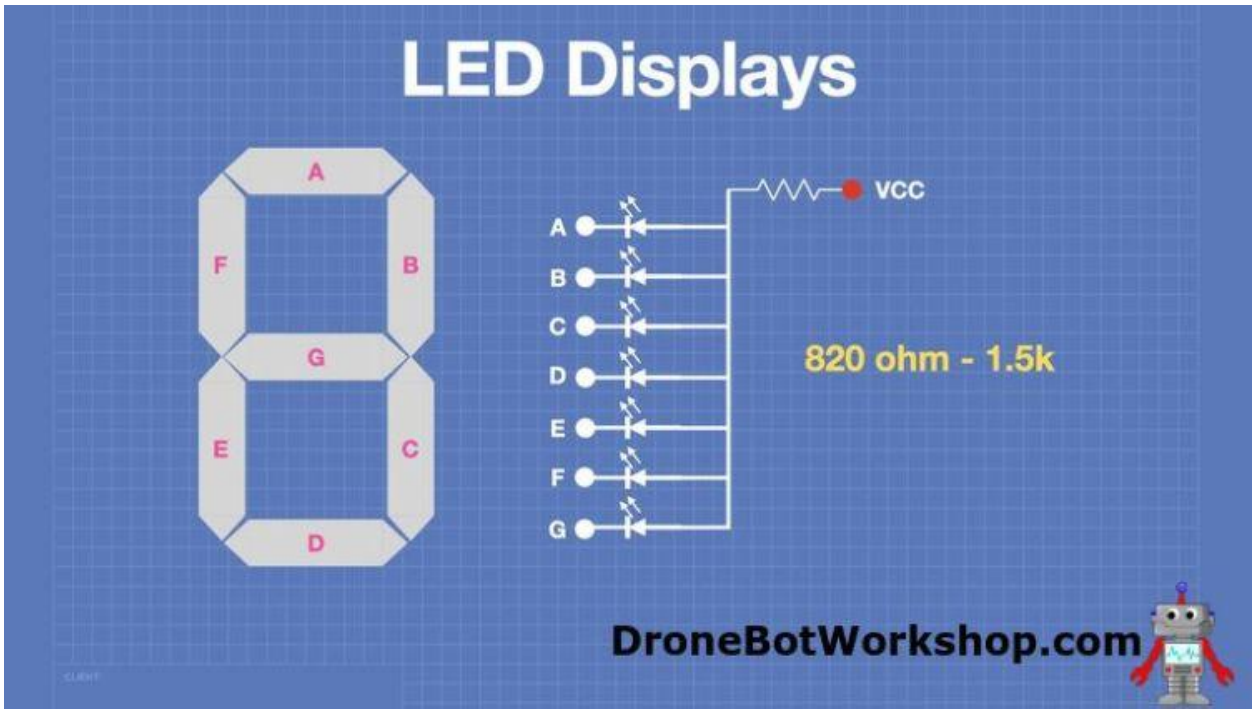
Each element of an LED display is, of course, an LED, so it will usually require a dropping resistor.

LED devices with multiple displays and internal controllers (we'll get to those in a bit) already have this figured out. But for discrete and multiplexed LED displays, you'll need to provide current limiting with dropping resistors.

There are two ways to hook up a dropping resistor to a 7-segment display.



In the more traditional method, a separate dropping resistor is used for each display element. This is actually the preferable method, but it does use up 8 resistors.

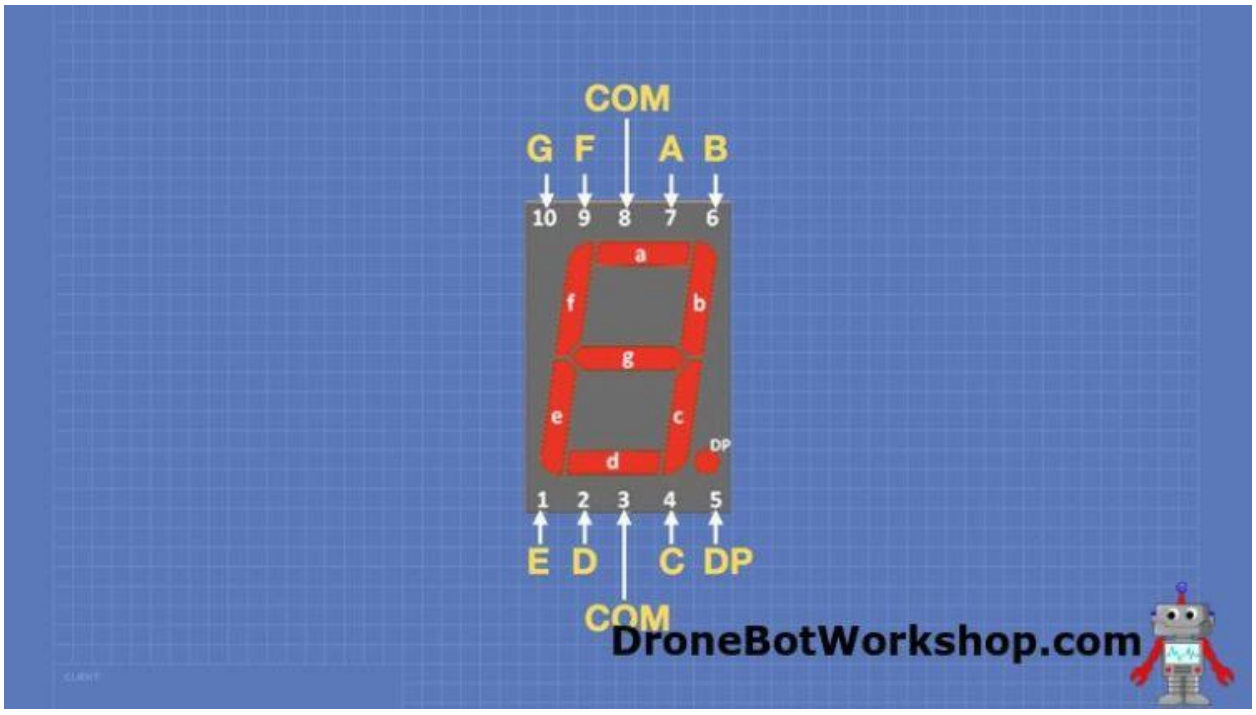


Another method uses a single dropping resistor on the common connection, this method can be used with both common anode and common cathode devices. This one resistor provides current limiting for all the segments.

While the second method does save on a number of resistors, it doesn't perform as well. Consider how much current is required for a number "1", which only uses two segments, and a number "8", which uses all seven. The "1"' will be giving more current per segment than the "8" will, so the brightness can vary between characters,

Single Digit Demonstration

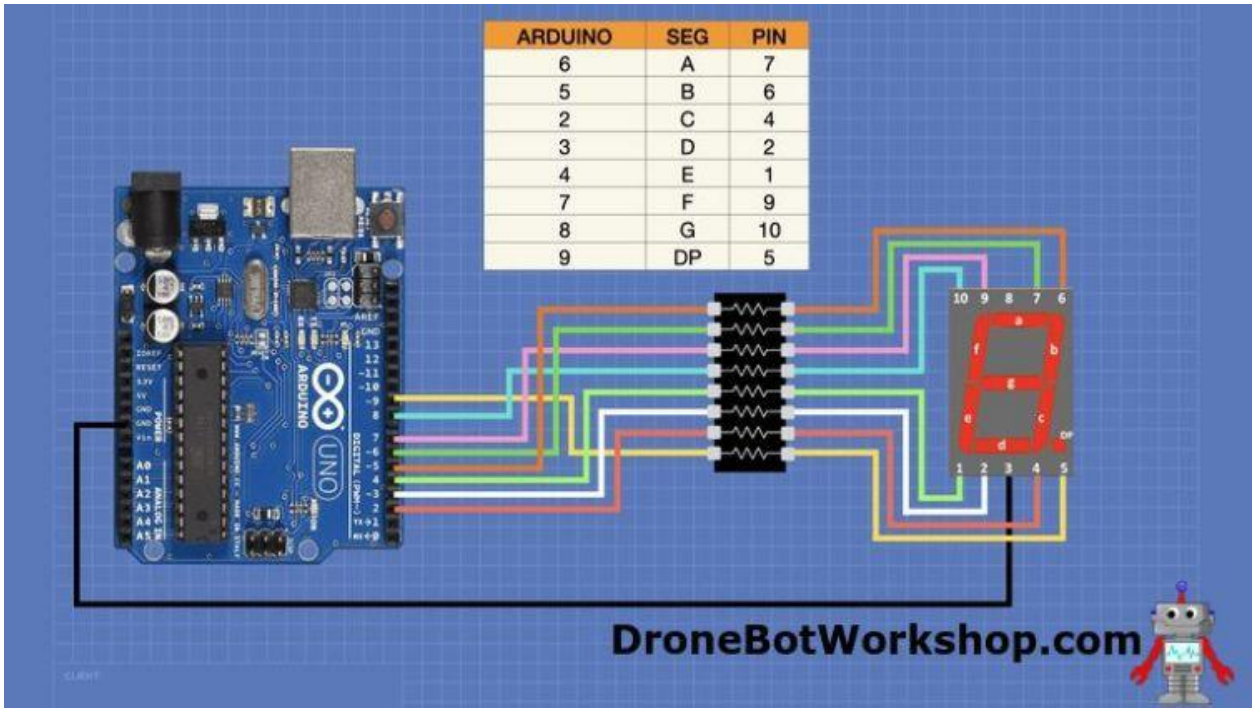
Let's begin by hooking up a single 7-segment LED to an Arduino Uno. Then we'll see how we can display some numbers on it.



Here is the pinout of a standard 7-segment LED display. In this experiment, I will be using a common cathode device, but the pinout for a common anode device is identical.

Display Hookup

As we are only driving one display device, we can safely power our display with the Arduino Uno's built-in power supply.



We will also need eight dropping resistors, any value from 220 to 470 ohms will be just fine. In order to save breadboard space, I used an 8-resistor 220-ohm resistor array, which packs eight 220-ohm resistors into a 16-pin DIP pack. Of course, you can also just use regular resistors.

If you want, you can write some simple code to check your work, just do a *digitalWrite* to each LED segment pin and observe the display.

Once you are satisfied that it is hooked up, we can look at some code we'll use to make it work.

The SevSeg Library

You'll find many examples on the inter4net for working with a directly connected 7-segment display and an Arduino/. In many of these examples, you simply create an array, using the segment states as values. There are at least 10 (perhaps 16) entries in the array, one for each digit you want to display.

I prefer to use a library when I can, and a good library for working with 7-segment displays is SevSeg.

You can install this library from your Arduino IDE Library Manager. Just filter by "sevseg" and you will find it.

Once you have it we can run our first sketch, which will simply print a number to our solitary LED display.

Single Display Code – Single Digit

Here is our first sketch:

```
/*
  7-Segment Display - Single Digit, directly wired
  7-seg-single-digit.ino
  Send a number to a single 7-segment display

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include library
#include "SevSeg.h"

// Create object
SevSeg sevseg;

// Number of digits in display
byte numDigits = 1;
// Display select pins (not used with single display)
byte digitPins[] = {};
// Display segment pins A,B,C,D,E,F,G,DP
byte segmentPins[] = {6, 5, 2, 3, 4, 7, 8, 9};
// Dropping resistors used
bool resistorsOnSegments = true;
// Display type
byte hardwareConfig = COMMON_CATHODE;

void setup() {

  // Start display object
```

```
  sevseg.begin(hardwareConfig, numDigits, digitPins, segmentPins, resistorsOnSegments);
  // Set brightness
  sevseg.setBrightness(90);

}

void loop() {

  // Set number to display
  sevseg.setNumber(3);
  // Refresh the display
  sevseg.refreshDisplay();
}
```

We start by including the SevSeg library and then defining an object to represent our display.

We then define a number of display parameters.

- **numDigits** – the number of digits in our display. In our case, this is 1, but we'll use the library with a multi-digit display a bit later.
- **digitPins** – This is an array with the digit selector pins in a multi-digit display. As we are only using one display it is left empty.
- **segmentPins** – This is an array of the I/O pins wired to the LED segments, in the order A, B, C, D, E, F, G, DP.
- **resistorsOnSegments** – Are we using dropping resistors?
- **hardwareConfig** – Valid responses are COMMON_ANODE and COMMON_CATHODE. Ours is a common cathode display.

In the Setup, we start the display object with the parameters we just defined. We also set the display brightness, any value from 0 to 150 will work here.

In the loop we just use the *setNumber* function of the library to print a digit to the display, it's that easy! We then refresh it, and go back and do it again.

Load the sketch to your Arduino and observe the display. It should read the number "3". Try changing the number and see if it works.

You'll also get interesting results using "non-numbers", give it a try!

Single Display Code – Counter

Here is an expansion of the first sketch, this time we will build a counter with our single-digit display.

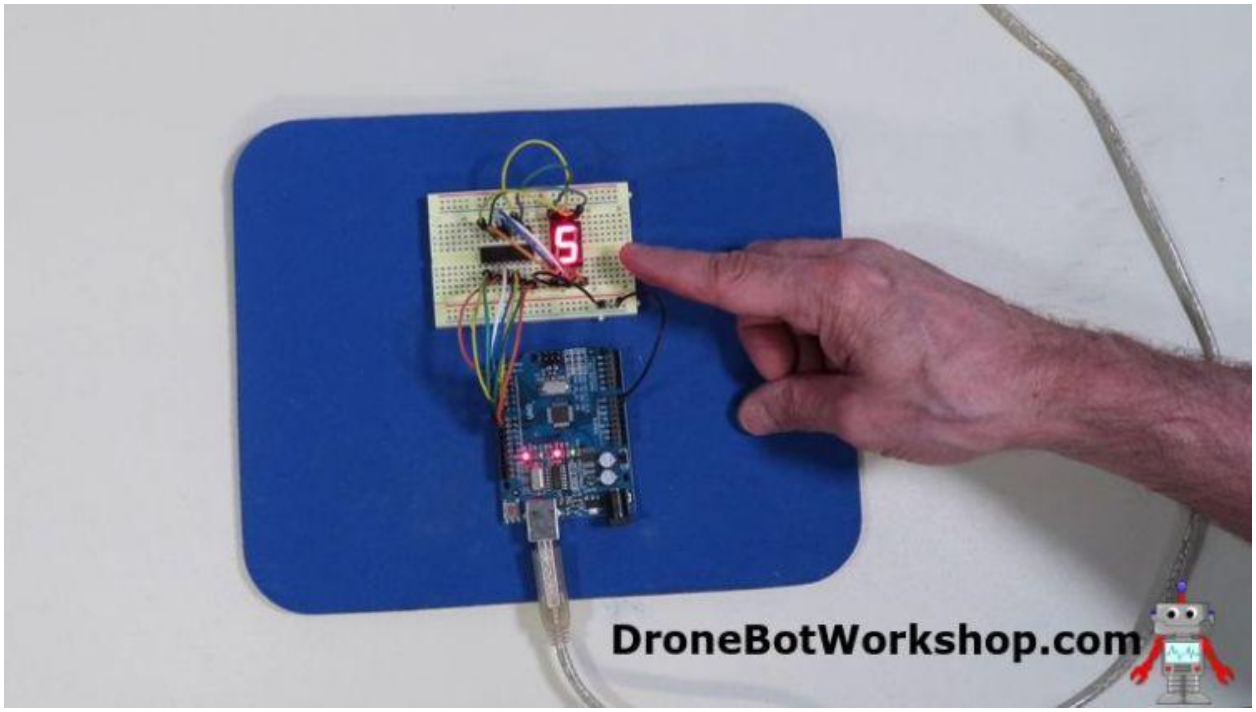
```
/*
  7-Segment Counter - Single Digit, directly wired
  7-seg-single-count.ino
  Count to a single 7-segment display

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include library
#include "SevSeg.h"

// Create object
SevSeg sevseg;

// Number of digits in display
byte numDigits = 1;
// Display select pins (not used with single display)
byte digitPins[] = {};
// Display segment pins A,B,C,D,E,F,G,DP
byte segmentPins[] = {6, 5, 2, 3, 4, 7, 8, 9};
// Dropping resistors used
bool resistorsOnSegments = true;
// Display type
byte hardwareConfig = COMMON_CATHODE;

void setup() {

  // Start display object
```

```
  sevseg.begin(hardwareConfig, numDigits, digitPins, segmentPins, resistorsOnSegments);
  // Set brightness
  sevseg.setBrightness(90);

}

void loop() {

  // Count from 0 to 9
  for (int i = 0; i < 10; i++) {
    // Set number for display
    sevseg.setNumber(i);
    // Wait one second
    delay(1000);
    // Refresh the display
    sevseg.refreshDisplay();
  }
}
```

The beginning of the sketch is identical to the previous one, using the SevSeg library to set up a display object.

In the Loop, we build a counter and then print its value to the display using the *setNumber* function. We then hold it for a second before refreshing the display.

Load it up to your display and watch it in action, you should observe it counting from 0 to 9 and then recycling.



The SevSeg library makes it very easy to write to a 7-segment LED display.

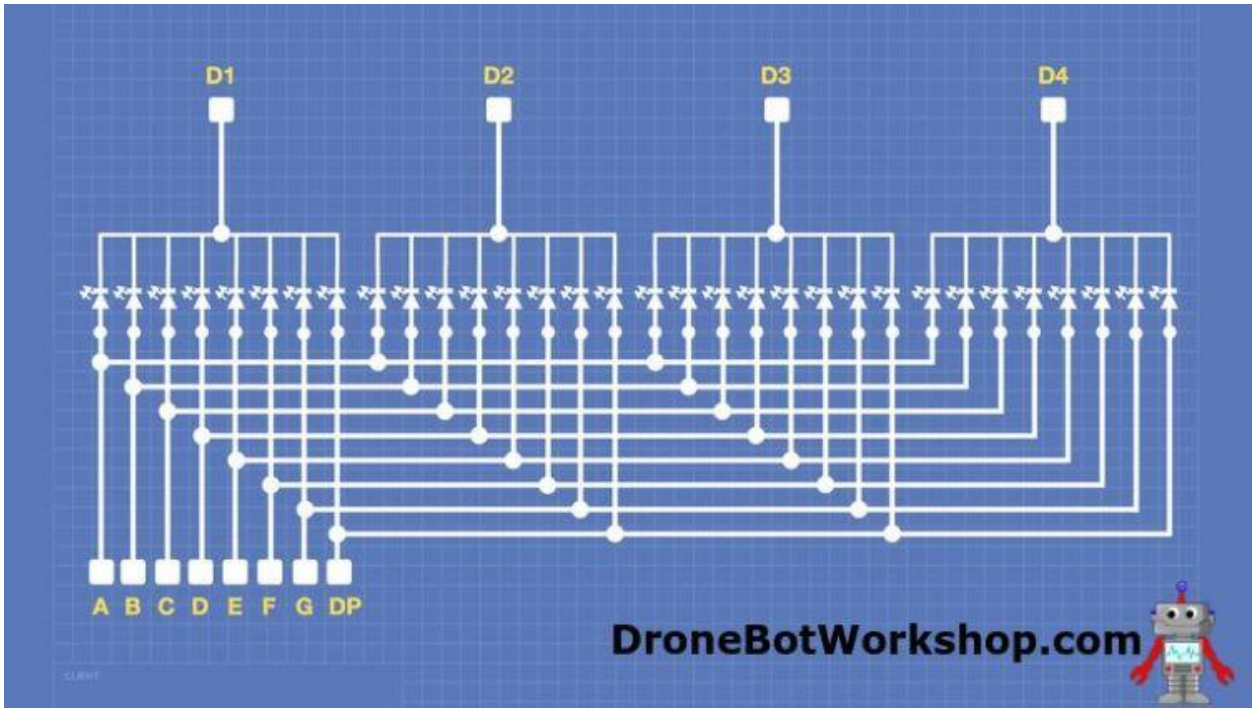
Using Multiple Displays

We often require more than one digit on our display, in fact, it's more common to need two, three, or four digits to display a value.

We can use multiplexed multi-digit displays to add more digits to our display, without adding too many more connections.

Multiple Display Theory

Multiple displays are like groups of common cathode or common anode displays. They are wired up with all the segment pins connected together.



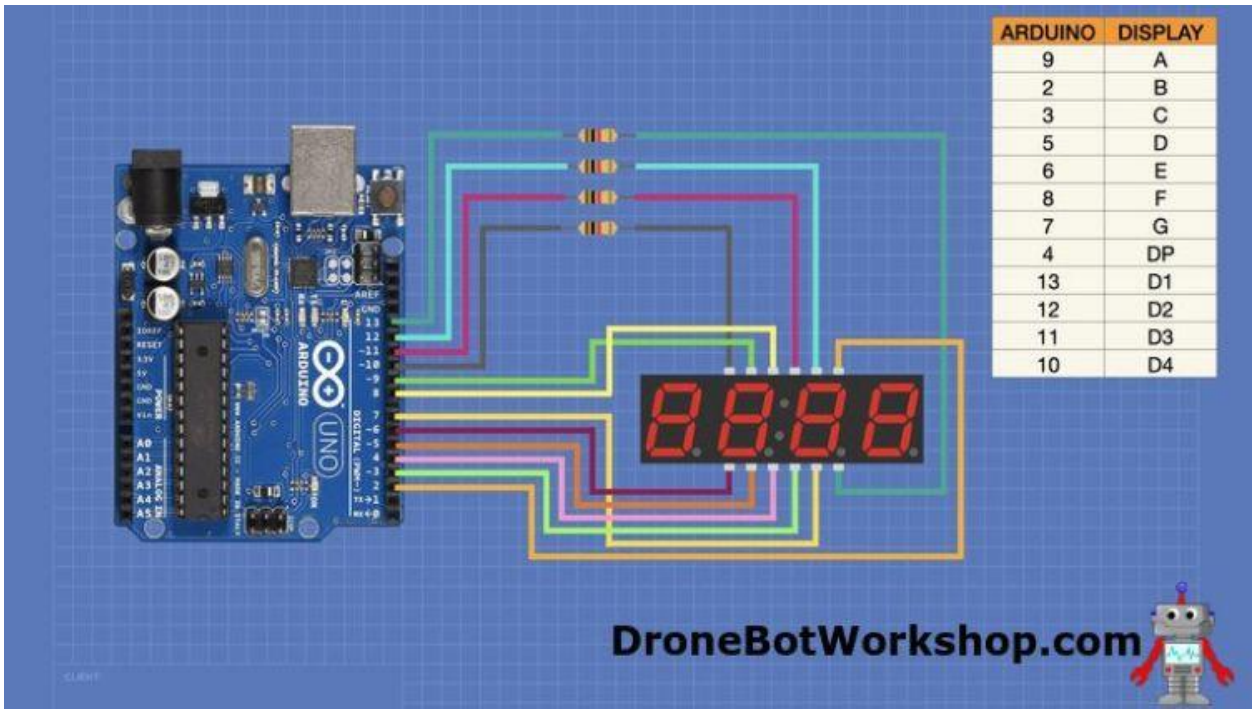
The common connection on each display is brought out to its own connection, a digit select pin. By using a combination of the digit select pin and the segment connections, you can write a pattern to a specific display.

Multiple Display Hookup

We will be using a 4-digit common cathode multiplexed display for this experiment. You could use two or three-digit devices as well.

Here is the pinout of the display I used, it's a pretty standard device, but you'll want to verify the pinout on your display to be sure before you hook it up.

We will also be using only four dropping resistors, hooked up to the digit select pins as shown in the following hookup diagram:



It's a lot of wires, so use the chart on the wiring diagram to double-check your wiring. After that, we can move on to the code.

Multiple Display Code

Once again we will be using the SevSeg library, as it also supports multiplexed displays.

```
/*
  7-Segment Display - Four Digit, directly wired
  7-seg-multi.ino
  Print a number to a 4-digit 7-segment display

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include library
#include "SevSeg.h"

// Create object
SevSeg sevseg;

// Number of digits in display
byte numDigits = 4;
// Display select pins
byte digitPins[] = {10, 11, 12, 13};
// Display segment pins A,B,C,D,E,F,G,DP
byte segmentPins[] = {9, 2, 3, 5, 6, 8, 7, 4};
// Dropping resistors used
```

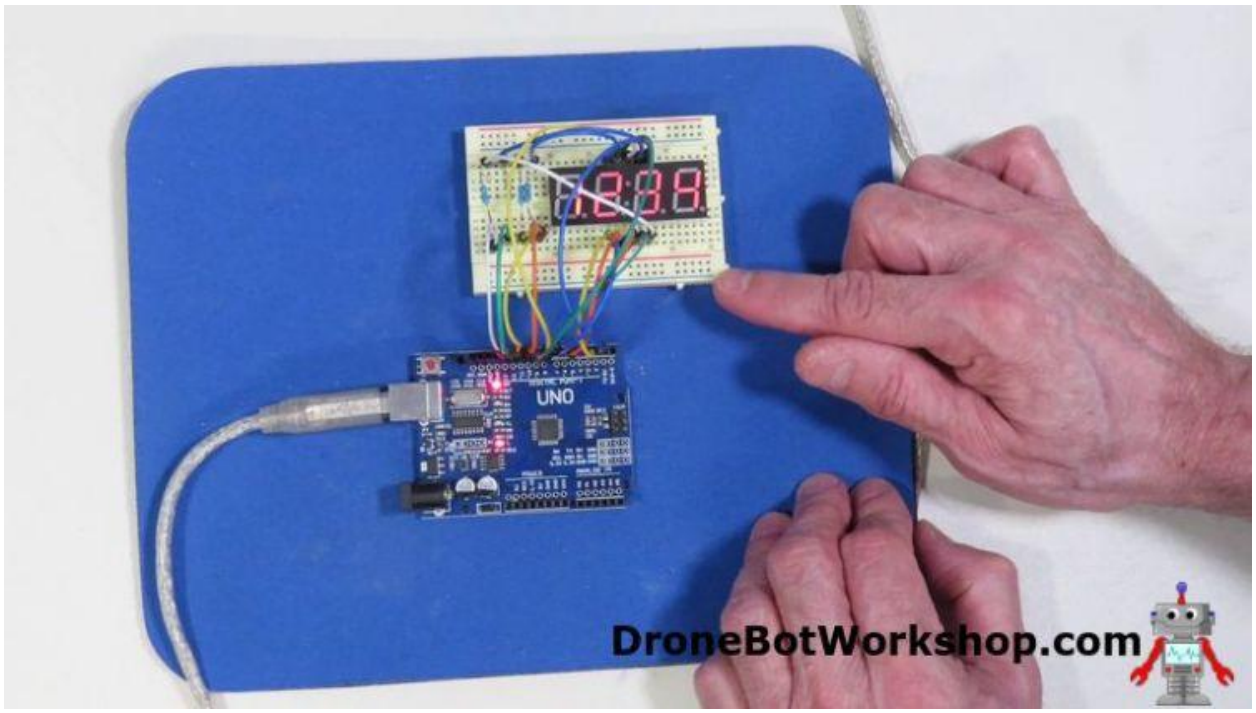
```
23 bool resistorsOnSegments = true;
24 // Display type
25 byte hardwareConfig = COMMON_CATHODE;
26
27 void setup() {
28
29   // Start display object
30   sevseg.begin(hardwareConfig, numDigits, digitPins, segmentPins,
   resistorsOnSegments);
31   // Set brightness
32   sevseg.setBrightness(90);
33 }
34
35 void loop() {
36
37   // Set the number to display
38   sevseg.setNumber(1234);
39   // Refresh the display
40   sevseg.refreshDisplay();
41 }
```

The code is very similar to our single-digit example, except you'll notice some key parameter value changes:

- **numDigits** – The number of digits is now four.
- **digitPins** – We now have values for our digit select pins. Note they are in order from right to left.

The Setup is identical to the previous code examples, and the Loop is almost identical to the first example. In fact, the only difference is that we are using *setNumber* with a four-digit value, instated of a single number.

Load the example and check out the display. Try changing the number to verify the operation.



You can check out the example in the SevSeg library for additional ideas.

MAX7219 8-Digit Display

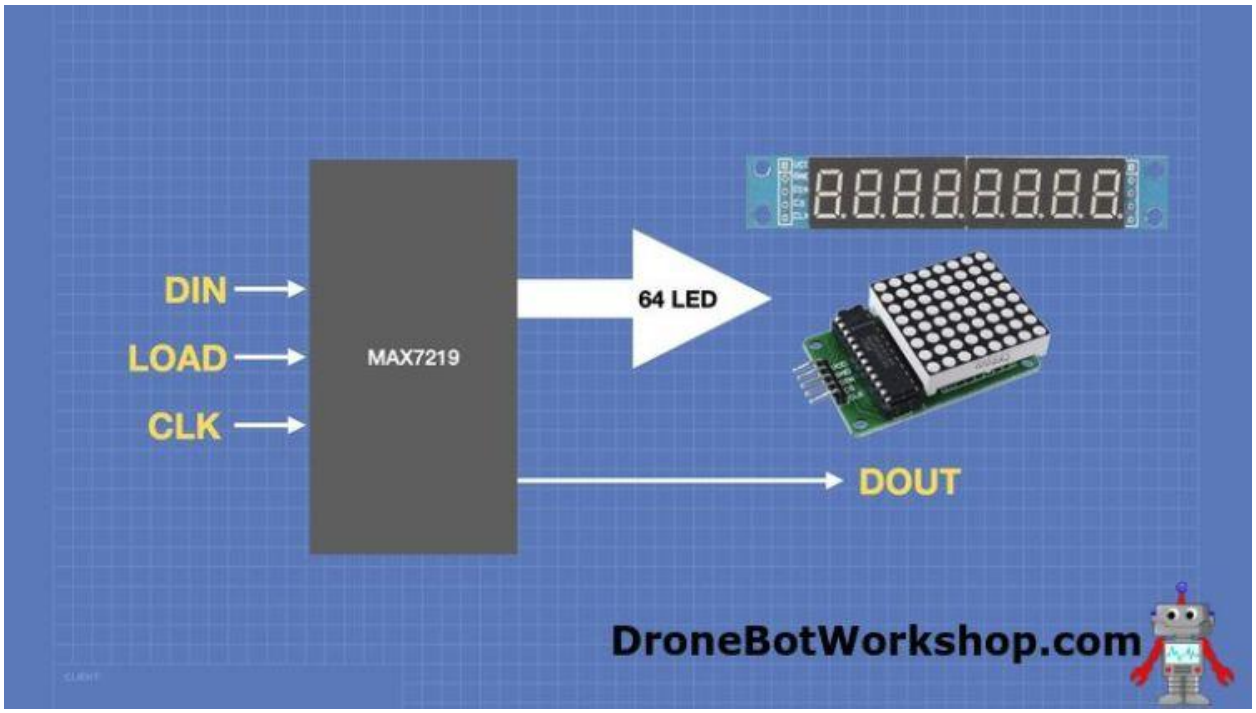
Multiplexing the wires in the display did save us a few connections, but nonetheless, we used up a dozen I/O pins on our Arduino Uno. That isn't going to leave room for many other peripherals, although you can use the analog pots as digital I/O ports if you wish.

A better way of doing this would be to add some intelligence to our display and send it serial data. Just as we would with the Arduino SPI port.

A great way to construct such an intelligent display would be to use a MAX7219 chip.

MAX7219

The MAX7219 is an LED display driver chip from Maxim Semiconductors. It can control a total of 64 LED segments, in eight groups of eight.



This means it can control:

- Eight 7-segment LEDs.
- One 8×8 Dot Matrix display.

The MAX7219 has three input pins:

- DIN – Data Input, this takes serial data in.
- LOAD – Also called CS, this pin tells the chip when to display its buffered data.
- CLK – The Clock input.

The output of the chip is as follows:

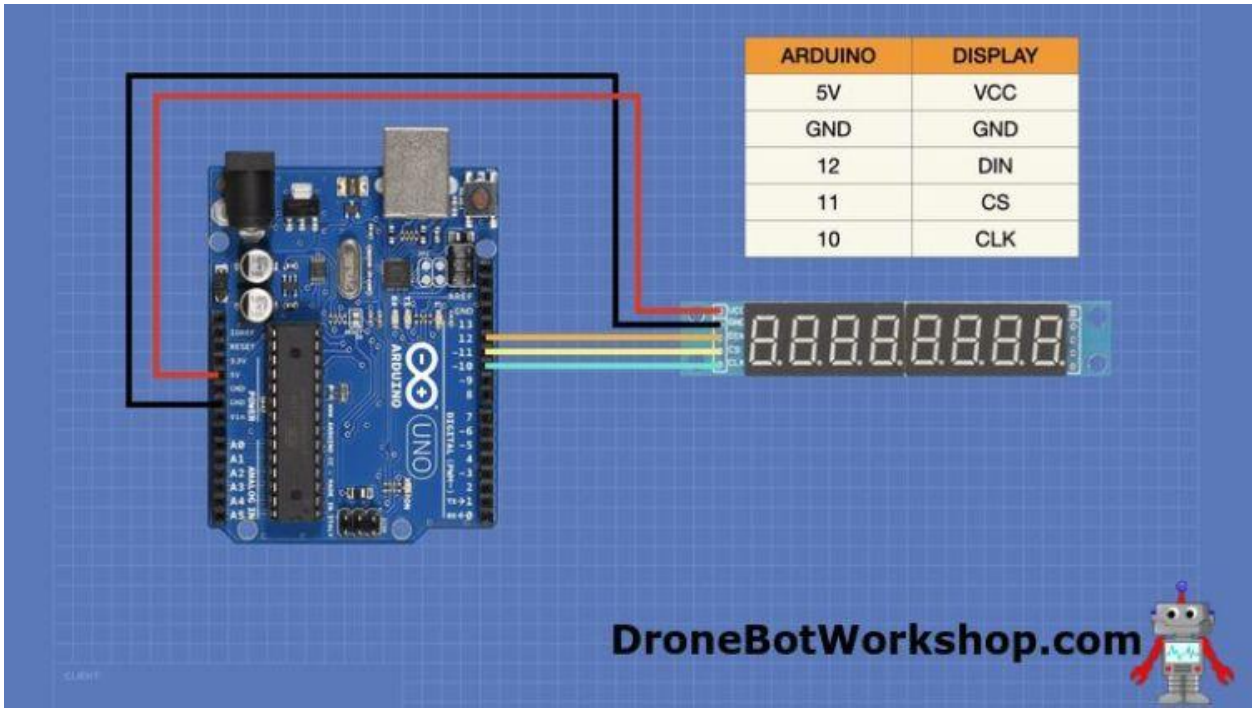
- DOUT – Data Out, for cascading devices.
- LEDs – the chip can drive 64 LEDs.

This is a great way to control a number of displays, or one dot matrix display, with a minimal number of connections to our Arduino.

8-Digit Display Hookup

In our first experiment with the MAX7219 we will hook up an 8-character 7-segment LED display. These are very common devices and have the MAX7219 hooked up, with only the three data connections plus power pins required to make it work.

These modules also have connections on both sides, the second set is the output. You can use these to cascade modules.



Here is how we will be hooking everything up. Again, we are powering the display from our Arduino, but you could also use a separate 5-volt power supply. If you do, make certain to connect the external power supplies ground to the Arduino ground.

8-Digit Display Code – One Display Unit

There are a number of libraries written for the MAX7219 chip, just typing “max7219” in the Library Manager filter window will reveal a good selection of them.

The first library I’m going to show you is a very simple one, that makes it easy to work with an 8-character LED display.

The library is called max7219 and it is by Jonathan Evans.

Install the library, then open the Demo sketch from its examples.

```
#include <max7219.h>


MAX7219 max7219;

void setup() {
  Serial.begin(9600);
  max7219.Begin();
}

void loop() {
  String temp;
  char temp2[8];
  int y;

  //DisplayText Demo
  max7219.DisplayText("95.67F", 1); //Right justified
  delay(3000);
  max7219.Clear();
  max7219.DisplayText("95.67F", 0); //Left justified
  delay(3000);
  max7219.Clear();

  //Counter with decimals
  //slow counter
  for (float x = 0; x < 1; x = x + 0.1) {
    temp = String(x);
    temp.toCharArray(temp2, temp.length());
    max7219.DisplayText(temp2, 1); //0=left justify 1=right justify
    Serial.println(x);
```

```
    delay(500);
  }
  //fast counter
  for (float x = 0; x < 500; x++) {
    temp = String(x);
    temp.toCharArray(temp2, temp.length());
    max7219.DisplayText(temp2, 1); //0=left justify 1=right justify
    Serial.println(x);
  }
  delay(500);

  //Display Char Demo
  max7219.Clear();
  max7219.DisplayChar(7, 'H', 0); //Position 7 is on the left of the display
  delay(500);
  max7219.DisplayChar(6, 'E', 0);
  delay(500);
  max7219.DisplayChar(5, 'L', 0);
  delay(500);
  max7219.DisplayChar(4, 'L', 0);
  delay(500);
  max7219.DisplayChar(3, 'O', 0);
  delay(500);
  max7219.DisplayChar(2, '1', 0);
  delay(500);
  max7219.DisplayChar(1, '2', 0);
  delay(500);
  max7219.DisplayChar(0, '3', 0);
  delay(500);
```

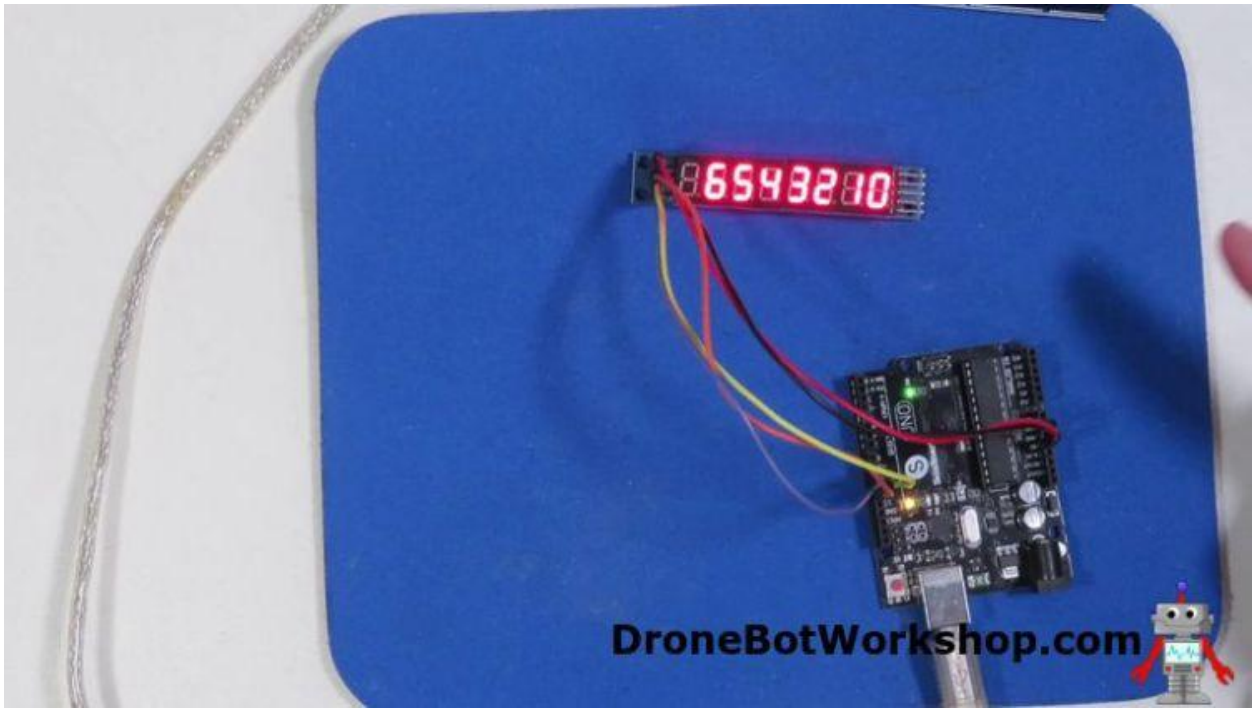
```
  max7219.Clear();
  //Count front the right
  for (int x = 0; x < 8; x++) {
    max7219.DisplayChar(x, 48 + x, 0); //48 is ASCII value for 0
    delay(500);
  }
  max7219.Clear();
  delay(500);
  //Count from the left
  for (int x = 7; x >= 0; x--) {
    max7219.DisplayChar(x, 48 + (7 - x), 0); //48 is ASCII value for 0
    delay(500);
  }
  max7219.Clear();
  //Count from the right
  for (int x = 0; x < 8; x++) {
    max7219.DisplayChar(x, 48 + x, 0); //48 is ASCII value for 0
    delay(500);
    max7219.Clear();
  }
  delay(500);
  //Count front the left
  for (int x = 7; x >= 0; x--) {
    max7219.DisplayChar(x, 48 + (7 - x), 0); //48 is ASCII value for 0
    delay(500);
    max7219.Clear();
  }
}
```

The sketch is pretty self-explanatory, it doesn't require any parameters for the display object and you then take it through a number of functions in the Loop.

Note how the DisplayChar function is used to write to the display. You pass it three parameters:

- The position of the text.
- The text itself (a single digit or letter).
- The text justification, 0 for left-justify and 1 for right-justify.

It is interesting that this takes alpha characters, try it out with a few of your own examples. If the letter is unprintable then the display will just stay blank.



Using Multiple Units

We can cascade these units, once again you will want to consider power supply requirements as we're getting to the limit of the Arduino Uno's internal voltage regulator.

Hooking this up is as simple as connecting the output connector (5-[pins) to the input of the second module.

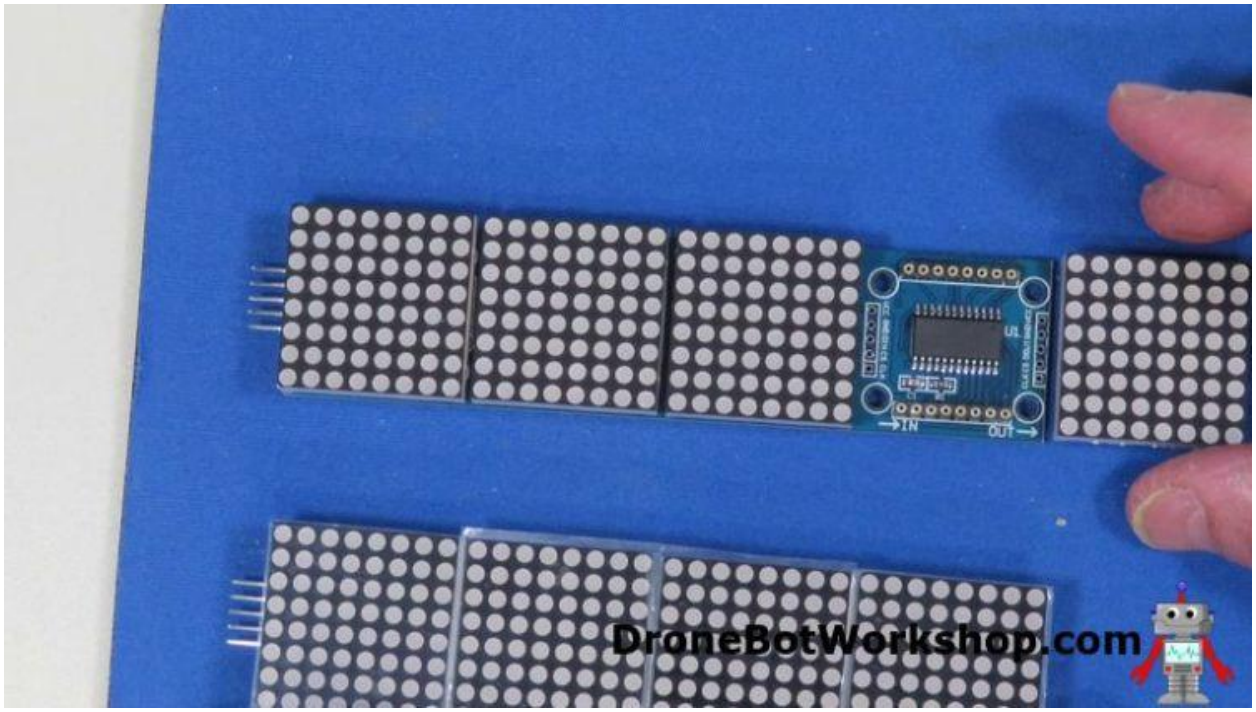
Unfortunately, the library we used for the single module cannot support multiple modules. So we'll need a different one.

The LedControl library is a common library for working with the MAX7219. You can install it using your Arduino IDE Library Manager.

The library comes with a number of examples, note that you'll probably have to change the object definition in the examples to reflect the wiring of our display, as the example files use different wiring. You can see an example in the video accompanying this article.

MAX7219 Dot Matrix Display

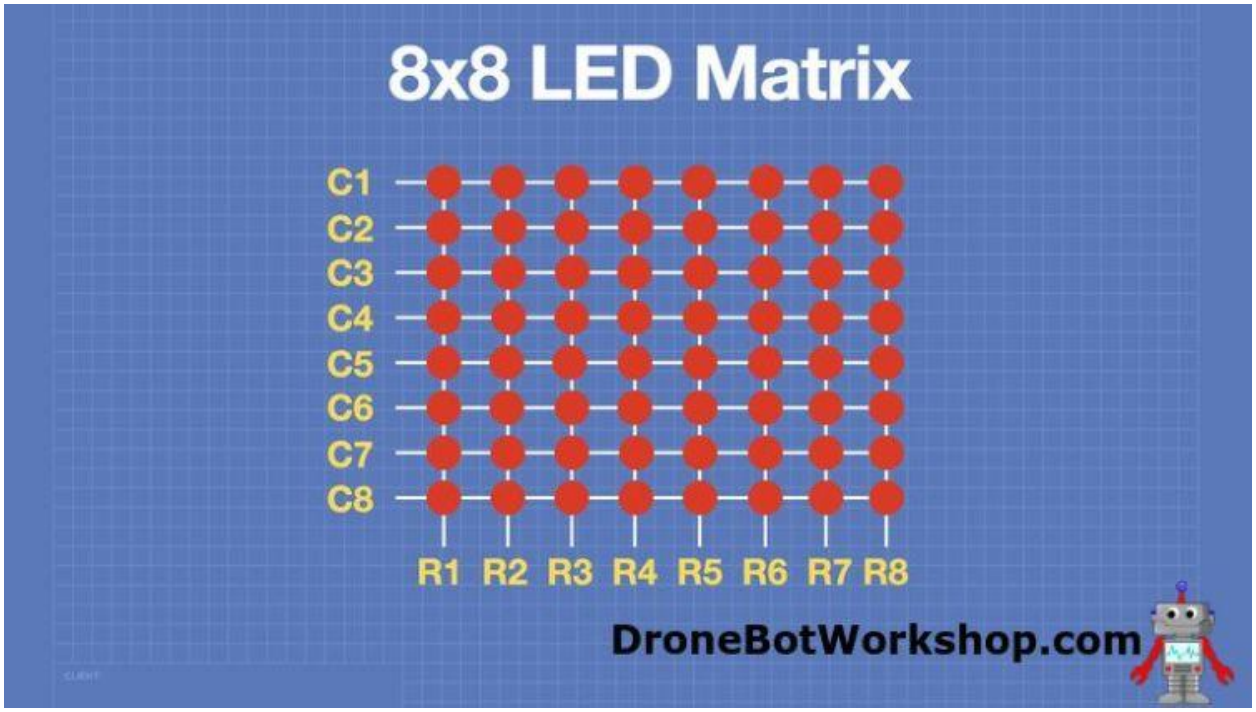
We can also use the MAX7219 to power an 8×8 LED matrix.



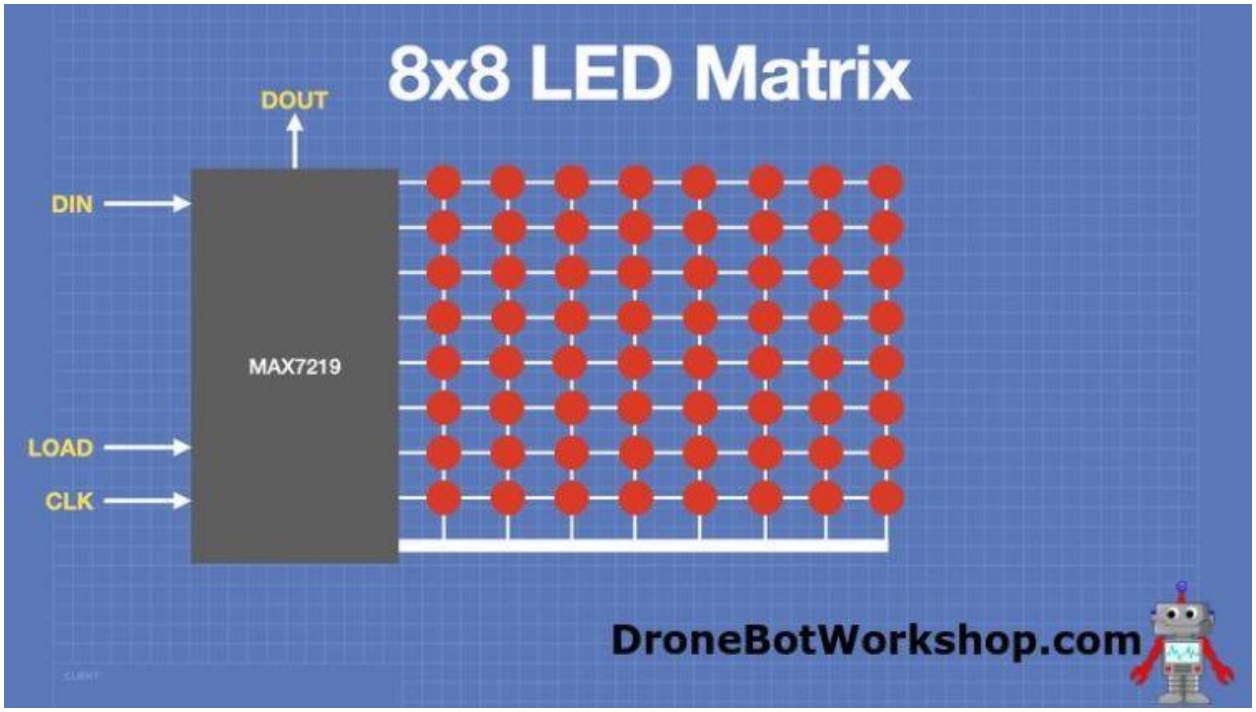
In this experiment, I am using a common module that actually contains four 8×8 LED dot matrix displays. Each display has its own MAX7219, and they are daisy-chained. The module has an output, so you can connect a second one.

Dot Matrix Displays

Dot Matrix displays have a number of advantages over standard 7-segment displays. Because it's a similar technique to what we use to print or to display characters on a video screen, these displays can be used for graphics as well as text and numbers.

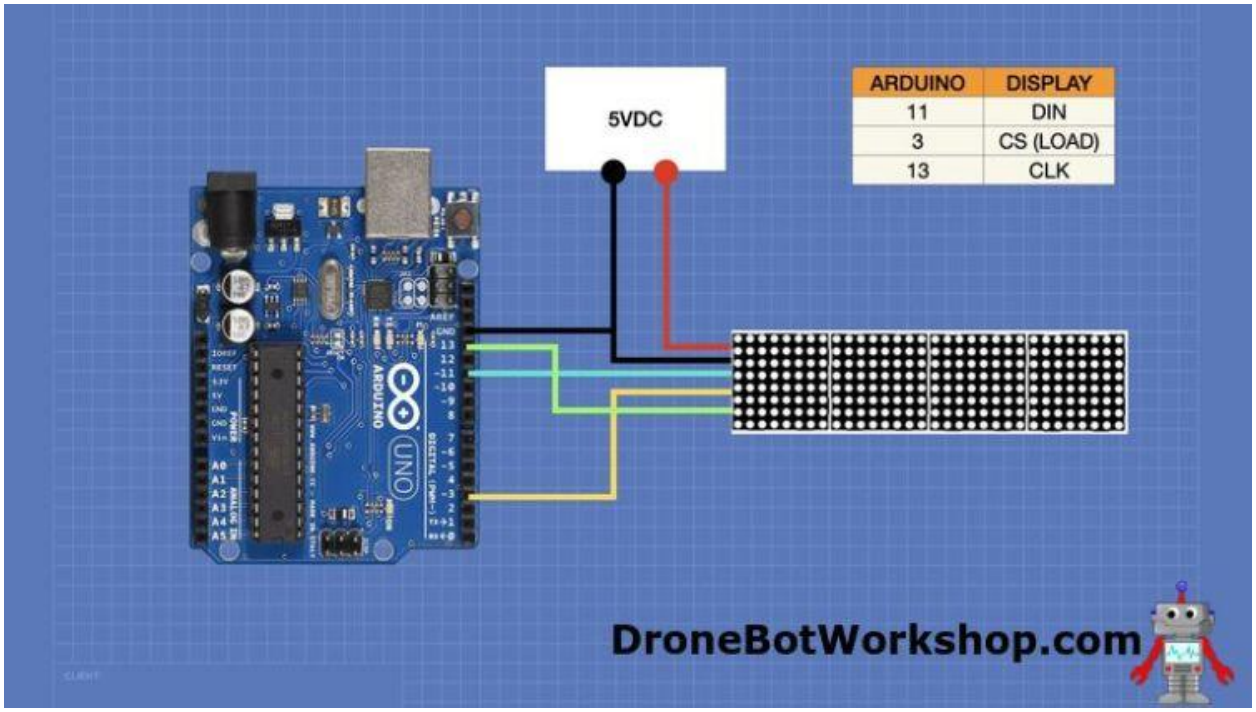


The 64 LED elements are arranged in an 8×8 grid and are addressed by specifying the grid junction point. As a MAX7219 is made for working with eight groups of eight LEDs, it's a perfect match.



Dot Matrix Hookup

Here is how we will be hooking up our dot matrix board. If you wish, you could use four individual displays instead of the four-display module that I used, just daisy-chain them together.

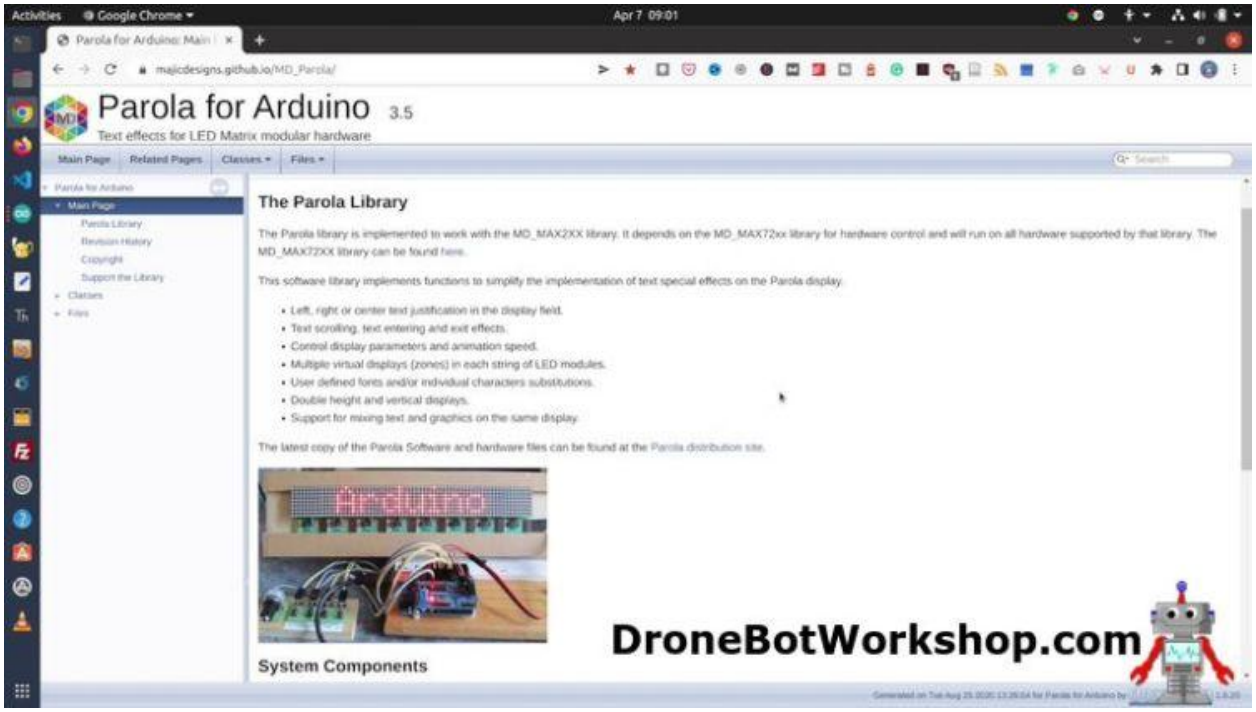


Note that I am specifying a separate power supply for this experiment, as it will exceed the capabilities of the Arduino's built-in voltage regulator.

Parola Library

Obviously, we will require a library in order to manipulate so many LED elements, and an excellent one is the Parola Library.

That library has functions and example code that will allow you to do pretty much everything with these dot matrix displays. From text to graphics to games, this library is absolutely amazing.



You can install the Parola library from the Library Manager in the Arduino Uno.

Dot Matrix Code – Demo Text

Our first example will display a few of the text positioning functions of the Parola library.

```
/*
  Matrix Display Demo
  matrix-display-demo.ino
  Demonstrates features of 4-element 8x8 LED matrix display

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include the required libraries
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

// Specify display hardware type
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
//#define HARDWARE_TYPE MD_MAX72XX::GENERIC_HW

// Display Size and CS Pin
#define MAX_DEVICES 4
#define CS_PIN 10

// Create a display object
MD_Parola myDisplay = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

void setup() {

  // Intialize the display object
  myDisplay.begin();
```

```
30
31    // Set the intensity (brightness) of the display (0-15)
32    myDisplay.setIntensity(5);
33
34    // Clear the display
35    myDisplay.displayClear();
36  }
37
38  void loop() {
39
40    // Align left
41    myDisplay.setTextAlignment(PA_LEFT);
42    myDisplay.print("Left");
43    delay(2000);
44
45    // Align center
46    myDisplay.setTextAlignment(PA_CENTER);
47    myDisplay.print("Center");
48    delay(2000);
49
50    // Align right
51    myDisplay.setTextAlignment(PA_RIGHT);
52    myDisplay.print("Right");
53    delay(2000);
54
55    // Invert display
56    myDisplay.setTextAlignment(PA_CENTER);
57    myDisplay.setInvert(true);
58    myDisplay.print("Invert");
```

```
  delay(2000);

  // Normal text
  myDisplay.setInvert(false);
  myDisplay.print(1234);
  delay(2000);
}
```

We include both the MD_Parola and MD_MAX72xx libraries, as well as the SPI library which we'll use to communicate to the display with.

We then set the display type, if you aren't using an identical display to mine you might have to change this.

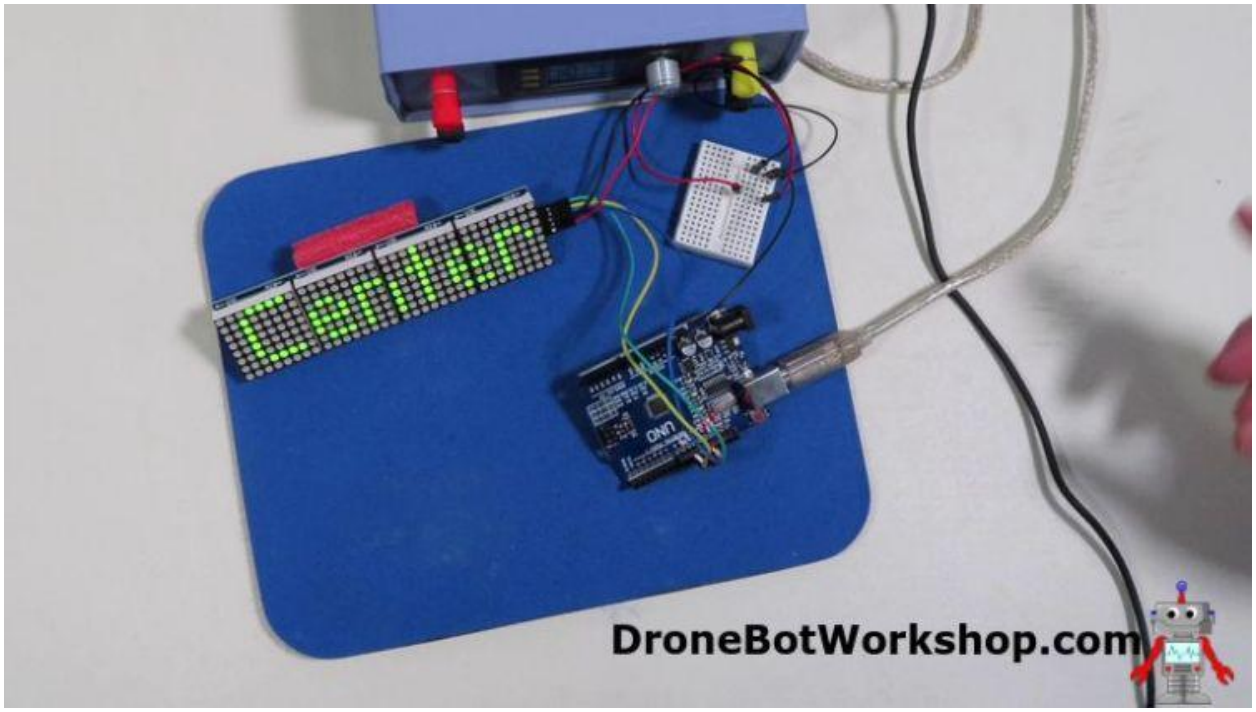
The MAX_DEVICES constant defines how many MAX7219 modules we are driving. My board has four modules, so I have that set to 4.

We also define the Chip Select pin for the SPI bus.

We then create a display object to represent our LED dot-matrix display module.

In the Setup, we initialize the display, and we also set its intensity and then clear it.

Then we come to the Loop, where we run the display through a number of different tests, using different parameter alignments.



Load the sketch to your Arduino and watch the display (you may need to Reset the Arduino). You should see a display of the library's text functions.

Dot Matrix Code – Scrolling Text

We can also use the Parola library to scroll text on the display, which is a very common application for dot-matrix LED displays.

```
/*
  Matrix Display Scrolling Demo
  matrix-scroll-demo.ino
  Scrolls text on a 4-element 8x8 LED matrix display

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include the required libraries
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

// Specify display hardware type
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
//#define HARDWARE_TYPE MD_MAX72XX::GENERIC_HW

// Display Size and CS Pin
#define MAX_DEVICES 4
#define CS_PIN 10

// Create a display object
MD_Parola myDisplay = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

void setup() {

  // Intialize the display object
  myDisplay.begin();
```

```
30
31    // Set the intensity (brightness) of the display (0-15)
32    myDisplay.setIntensity(5);
33
34    // Clear the display
35    myDisplay.displayClear();
36
37    // Display our text in a scroll
38    myDisplay.displayScroll("Welcome to the Workshop!", PA_CENTER, PA_SCROLL_LEFT,
100);
39  }
40
41
void loop() {
42
43    // Ensure display stays animated
44    if (myDisplay.displayAnimate()) {
45      myDisplay.displayReset();
46    }
47  }
```

In this sketch, we begin as we did in the previous sketch.

The main difference is in Setup, where we simply use the *displayScroll* function to scroll some text. The function has some parameters:

- The text you wish to scroll.
- The alignment of that text.
- The direction you wish to scroll.
- The speed you wish to scroll at.

The speed is a delay, so the smaller the number the faster the text will scroll.

The Loop just contains some text to keep everything going.

Load it up and watch your display. You can, of course, change the text if you don't want to welcome anyone to your own workshop!

Conclusion

LED displays are by no means out of fashion. They are used in new devices every day, and in some situations, they are the perfect display choice.

So the next time you're looking for an easy-to-use and easy-to-read display for your Arduino project consider using an LED.

It could be a very bright decision!