Using Arduinos in Vocational Training

UsingARDinVET

IPSIA "G.Giorgi" - Potenza (Italy)

Dot Matrix Display Module

RGB LEDs

A Light-Emitting Diode (LED) is a small component that illuminates when current flows through it. RGB LEDs (Figure 1) operate on the same principle, but they internally contain three LEDs (Red, Green, and Blue) capable of combining to produce nearly any color output.

Figure 1: RGB LEDs.

The RGB color model is a way to represent colors by mixing red, green, and blue light (Figure 2). Each color channel's intensity determines the overall color displayed. Combining these primary colors at different levels generates millions of colors visible to the human eye. For example, to create purely blue light, you have to adjust the blue LED to the highest intensity while setting the green and red LEDs to the lowest. But, for white light, all three LEDs have to be set to their highest intensity.

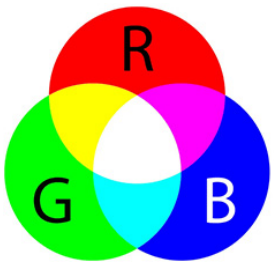

Figure 2: RGB color model.

RGB LEDs contain three LEDs inside, and usually, these three LEDs share a common anode or cathode. This categorizes RGB LEDs as either a common anode or a common cathode type (Figure 3).

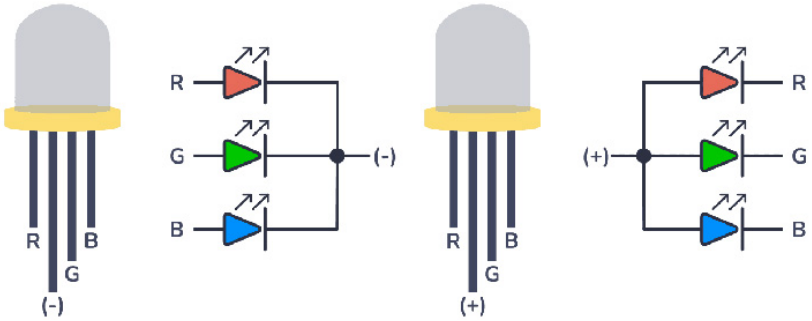

Figure 3: Types of RGB LEDs.

Circuit 1. Using a RGB LED

To achieve different colors with an RGB LED you need to control the brightness of each internal LED. This can be accomplished by using PWM signals with an Arduino.
In the circuit shown in the figure 4 the cathode is connected to GND, and the three anodes are connected to three digital pins on the Arduino Board through 220 Ohms resistors. It is important that the pins you use in your Arduino can output PWM signals.

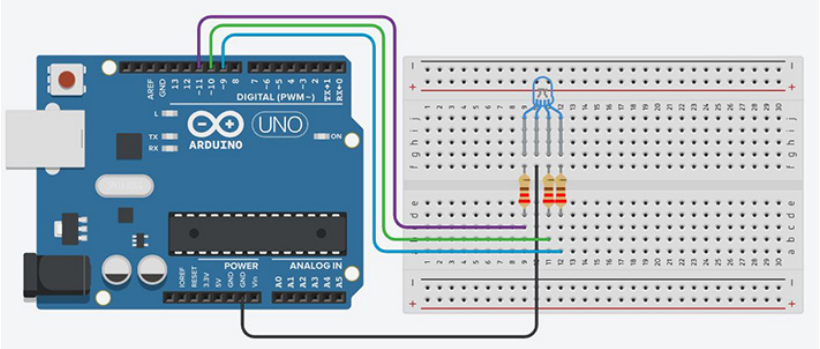

Figure 4: Connect an RGB LED to an Arduino.

The following code will make the RGB LED change a few colors:

```
// Declare the PWM LED pins
int redLED = 9;
int greenLED = 10;
int blueLED = 11;

void setup() {
  // Declare the pins for the LED as Output
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(blueLED, OUTPUT);
}
```

```
// A simple function to set the level for each color from 0 to 255
void setColor(int redValue, int greenValue, int blueValue) {
  analogWrite(redLED, redValue);
  analogWrite(greenLED, greenValue);
  analogWrite(blueLED, blueValue);
}

void loop() {
  // Change a few colors
  setColor(255, 0, 0);  // Red Color
  delay(1000);
  setColor(0, 255, 0);  // Green Color
  delay(1000);
  setColor(0, 0, 255);  // Blue Color
  delay(1000);
  setColor(255, 255, 0);  // Yellow
  delay(1000);
  setColor(0, 255, 255);  // Cyan
  delay(1000);
  setColor(255, 0, 255);  // Magenta
  delay(1000);
  setColor(255, 255, 255);  // White
  delay(1000);
}
```

In the setup function, pins 9, 10, and 11 are configured as outputs. The loop function repeatedly calls the `setColor` function to display different colors at one-second intervals. The `setColor` function takes three parameters (red, green, and blue values) which can range from 0 to 255. These values are used in the `analogWrite` function, which outputs PWM signals to control the intensity of each RGB LED channel color.

Circuit 2. Control RGB LED with potentiometer

In this example (figure 5), we are going to modify the color of the RGB LED when we turn the potentiometer knob.
Let's write the code for that.

```
#define RGB_RED_PIN 11
#define RGB_BLUE_PIN 10
#define RGB_GREEN_PIN 9
#define POTENTIOMETER_PIN A0

void setup()
```

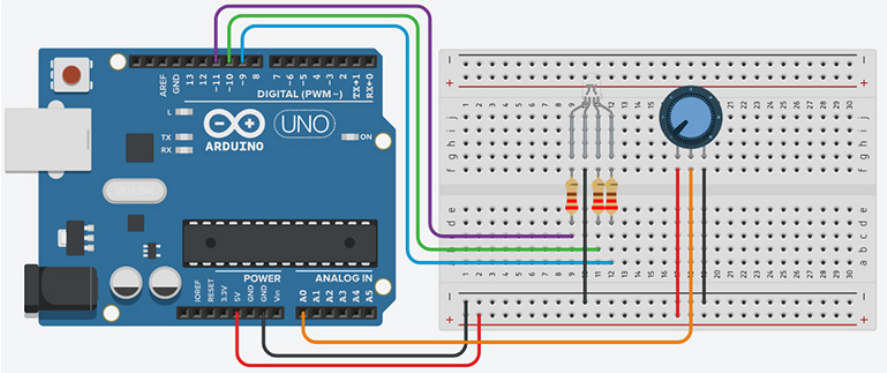

Figure 5: Arduino circuit with RGB LED and potentiometer.

```
{
  pinMode(RGB_RED_PIN, OUTPUT);
  pinMode(RGB_BLUE_PIN, OUTPUT);
  pinMode(RGB_GREEN_PIN, OUTPUT);
}

void loop()
{
  int potentiometerValue = analogRead(POTENTIOMETER_PIN);
  int rgbValue = map(potentiometerValue, 0, 1023, 0, 1535);
  int red;
  int blue;
  int green;

  if (rgbValue < 256) {
    red = 255;
    blue = rgbValue;
    green = 0;
  }
  else if (rgbValue < 512) {
    red = 511 - rgbValue;
    blue = 255;
    green = 0;
  }
  else if (rgbValue < 768) {
    red = 0;
    blue = 255;
    green = rgbValue - 512;
  }
  else if (rgbValue < 1024) {
    red = 0;
```

```
    blue = 1023 - rgbValue;
    green = 255;
  }
  else if (rgbValue < 1280) {
    red = rgbValue - 1024;
    blue = 0;
    green = 255;
  }
  else {
    red = 255;
    blue = 0;
    green = 1535 - rgbValue;
  }

  analogWrite(RGB_RED_PIN, red);
  analogWrite(RGB_BLUE_PIN, blue);
  analogWrite(RGB_GREEN_PIN, green);
}
```

At first, we create a define for each pin we are going to use. One for the potentiometer, and one for each color of the LED (we write the code as if we were controlling 3 different LEDs).
In the `void setup()`, we initialize all LEDs (in fact, the 3 legs of the RGB LED) to OUTPUT mode. Nothing to do for the potentiometer, as an analog pin is already in input mode by default.
In the `void loop()`, we first read the potentiometer's value with `analogRead()`. This gives us a value between 0 and 1023. Because we want to choose between 1536 different options, we use the `map()` function to transform this value from the range 0-1023 to the range 0-1535.
So, we have 6 different steps for changing the color. Also, you can note that the first color and the last color are the same (red).
For the 1st step: we set red to 255, and we increase the blue color from 0-255, according to the rgbValue we computed (in the range 0-1535).
If the rgbValue is more than 255, we go to step number 2. Now we have values from 256 to 511. We set blue to 255, and then decrease the red value. To do so, we need to subtract the rgbValue to the max value for this block, which is 511. As an example, if we enter the if structure with rgbValue = 400, then we have red = 511 - 400 = 111.
For step number 3, we keep blue to 255, and this time we increase green. The rgbValue is now between 512 and 767. So, to start from 0 and get to 255, we subtract 512 to each value we get.
Steps number 4, 5, and 6 are following the same logic as the previous steps.
Now, we have 3 values between 0-255, stored into 3 different variables. After the computation, we use `analogWrite()` on each leg of the RGB like if it were 3 different LEDs, with the corresponding values for red, blue, and green.

LED Matrices

An LED matrix is useful for a wide range of applications. An 8x8 matrix, like the one shown in Figure 6, can be used to display letters or numbers. If you have several of these modules placed side by side, you can create a display with scrolling text.

Figure 6: An 8x8 dot matrix LED.

Note that the module is designed so that there is very little space between the LEDs and the edges of the module. When these types of matrix modules are mounted side by side, the distance between the last column or row on one module and the adjacent column or row on the next module is equal to the distance between the LEDs located at the center of the module. This maintains a consistent spacing when using multiple modules to create large displays.

Circuit 3. Controlling an LED Matrix

This sketch uses an LED matrix of 64 LEDs, with anodes connected in rows and cathodes in columns (as in the Jameco 2132349). Figure 7 shows the connections (Dual-color LED displays may be easier to obtain, and you can drive just one of the colors if that is all you need).

```
const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[] = {10, 11, 12, A1, A2, A3, A4, A5};
int pixel = 0;          // 0 to 63 LEDs in the matrix
int columnLevel = 0;    // pixel value converted into LED column
int rowLevel = 0;       // pixel value converted into LED row

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(columnPins[i], OUTPUT);
    pinMode(rowPins[i], OUTPUT);
  }
}

void loop() {
  pixel = pixel + 1;
  if (pixel > 63) pixel = 0;
```

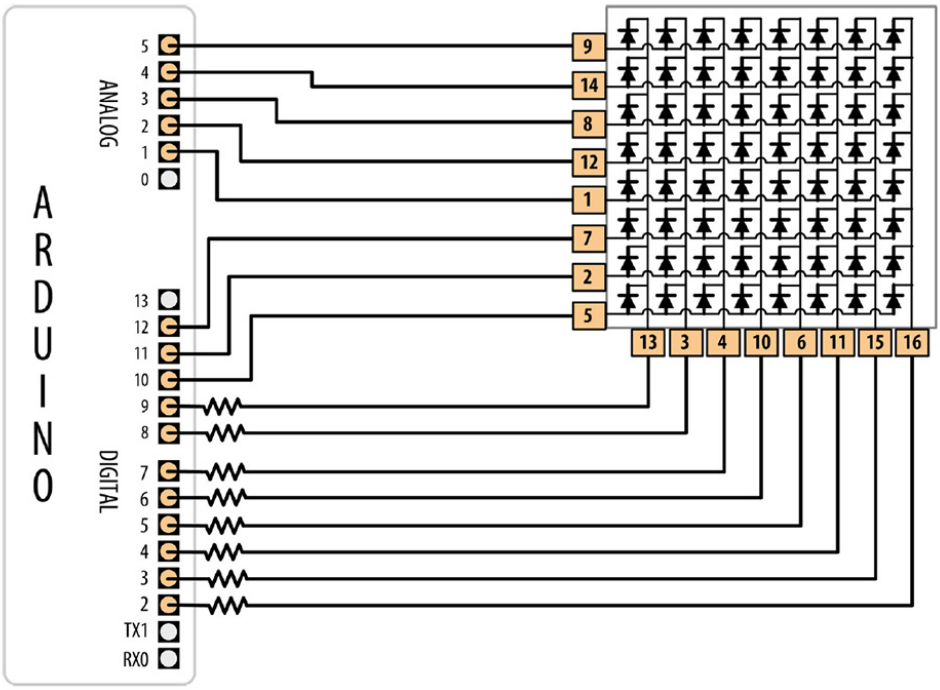

Figure 7: An LED matrix connected to 16 digital pins.

```
  columnLevel = pixel / 8;  // map to the number of columns
  rowLevel = pixel % 8;     // get the fractional value
  for (int column = 0; column < 8; column++) {
    digitalWrite(columnPins[column], LOW);
    for (int row = 0; row < 8; row++) {
      if (columnLevel > column) {
        digitalWrite(rowPins[row], HIGH);
      } else if (columnLevel == column && rowLevel >= row) {
        digitalWrite(rowPins[row], HIGH);
      } else {
        // turn off all LEDs in this row
        digitalWrite(columnPins[column], LOW);
      }
      delayMicroseconds(300);
      digitalWrite(rowPins[row], LOW);  // turn off LED
    }
    // disconnect this column from Ground
    digitalWrite(columnPins[column], HIGH);
  }
}
```

The resistor's value must be chosen to ensure that the maximum current through a pin does not exceed 40 mA on the Arduino Uno. Because the current for up to eight LEDs can flow through

each column pin, the maximum current for each LED must be one-eighth of 40 mA, or 5 mA. Each LED in a typical small red matrix has a forward voltage of around 1.8 volts. Calculating the resistor that results in 5 mA with a forward voltage of 1.8 volts gives a value of 680Ω. Check your datasheet to find the forward voltage of the matrix you want to use. Each column of the matrix is connected through the series resistor to a digital pin. When the column pin goes low and a row pin goes high, the corresponding LED will light. For all LEDs where the column pin is high or its row pin is low, no current will flow through the LED and it will not light. The for loop scans through each row and column and turns on sequential LEDs until all LEDs are lit. The loop starts with the first column and row and increments the row counter until all LEDs in that row are lit; it then moves to the next column, and so on, lighting another LED with each pass through the loop until all the LEDs are lit.

You don't have to light an entire row at once. The following sketch will light one LED at a time as it goes through the sequence:

```
const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[] = {10, 11, 12, A1, A2, A3, A4, A5};
int pixel = 0;  // 0 to 63 LEDs in the matrix

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(columnPins[i], OUTPUT);  // make all the LED pins outputs
    pinMode(rowPins[i], OUTPUT);
    digitalWrite(columnPins[i], HIGH);
  }
}

void loop() {
  pixel = pixel + 1;
  if (pixel > 63) pixel = 0;
  int column = pixel / 8;                  // map to the number of columns
  int row = pixel % 8;                     // get the fractional value
  digitalWrite(columnPins[column], LOW);   // Connect this column to GND
  digitalWrite(rowPins[row], HIGH);        // Take this row HIGH
  delay(125);                              // pause briefly
  digitalWrite(rowPins[row], LOW);         // Take the row low
  digitalWrite(columnPins[column], HIGH);  // Disconnect the column from GND
}
```

Circuit 4. Displaying Images on an LED Matrix

You want to display one or more images on an LED matrix, perhaps creating an animation effect by quickly alternating multiple images. This solution can use the same wiring as in figure

7. The sketch creates the effect of a heart beating by briefly lighting LEDs arranged in the shape of a heart. A small heart followed by a larger heart is flashed for each heartbeat (the images look like figure 8):

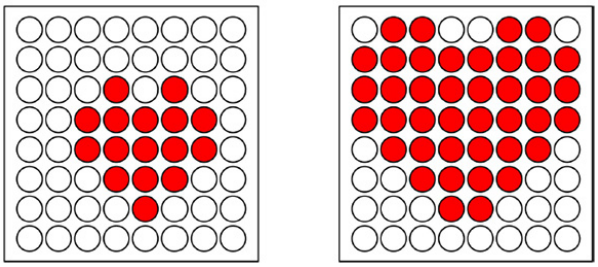

Figure 8: The two heart images displayed on each beats.

```
byte bigHeart[] = {
    B01100110,
    B11111111,
    B11111111,
    B11111111,
    B01111110,
    B00111100,
    B00011000,
    B00000000};
byte smallHeart[] = {
    B00000000,
    B00000000,
    B00010100,
    B00111110,
    B00111110,
    B00011100,
    B00001000,
    B00000000};
const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[] = {10, 11, 12, A1, A2, A3, A4, A5};

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(rowPins[i], OUTPUT);   // make all the LED pins outputs
    pinMode(columnPins[i], OUTPUT);
    digitalWrite(columnPins[i], HIGH);   // disconnect column pins from Ground
  }
}

void loop() {
```

```
  int pulseDelay = 800;   // milliseconds to wait between beats
  show(smallHeart, 80);   // show the small heart image for 80 ms
  show(bigHeart, 160);    // followed by the big heart for 160 ms
  delay(pulseDelay);      // show nothing between beats
}

// Show a frame of an image stored in the array pointed to by the image
// parameter. The frame is repeated for the given duration in milliseconds.
void show(byte* image, unsigned long duration) {
  unsigned long start = millis();       // begin timing the animation
  while (start + duration > millis())   // loop until the duration has passed
  {
    for (int row = 0; row < 8; row++) {
      digitalWrite(rowPins[row], HIGH);  // connect row to +5 volts
      for (int column = 0; column < 8; column++) {
        bool pixel = bitRead(image[row], column);
        if (pixel == 1) {
          digitalWrite(columnPins[column], LOW);  // connect column to Gnd
        }
        delayMicroseconds(300);                   // a small delay for each LED
        digitalWrite(columnPins[column], HIGH);  // disconnect column from Gnd
      }
      digitalWrite(rowPins[row], LOW);  // disconnect LEDs
    }
  }
}
```

The value written to the LED is based on images stored in the `bigHeart` and `smallHeart` arrays. Each element in the array represents a pixel (a single LED) and each array row represents a row in the matrix. A row consists of eight bits represented using binary format (as designated by the capital B at the start of each row). A bit with a value of 1 indicates that the corresponding LED should be on; a 0 means off. The animation effect is created by rapidly switching between the arrays. The `loop` function waits a short time (800 ms) between beats and then calls the `show` function, first with the `smallHeart` array and then followed by the `bigHeart` array. The show function steps through each element in all the rows and columns, lighting the LED if the corresponding bit is 1. The `bitRead` function is used to determine the value of each bit. A short delay of 300 microseconds between each pixel allows the eye enough time to perceive the LED. The timing is chosen to allow each image to repeat quickly enough (50 times per second) so that blinking is not perceptible.

Here is a variation that changes the rate at which the heart beats, based on the value from a sensor. You can test this using a variable resistor connected to analog input pin 0. Use the

wiring and code shown earlier, except replace the loop function with this code:

```
void loop() {
  int sensorValue = analogRead(A0);  // read the analog in value
  int pulseRate =
      map(sensorValue, 0, 1023, 40, 240);  // convert to beats / minute
  int pulseDelay = (60000 / pulseRate);     // milliseconds to wait between beats
  show(smallHeart, 80);  // show the small heart image for 100 ms
  show(bigHeart, 160);   // followed by the big heart for 200 ms
  delay(pulseDelay);     // show nothing between beats
}
```

This version calculates the delay between pulses using the map function to convert the sensor value into beats per minute.

Circuit 5. Controlling an Array of LEDs by using MAX72xx

You have an 8x8 array of LEDs to control, and you want to minimize the number of required Arduino pins. You can use a shift register to reduce the number of pins needed to control an LED matrix. This solution uses the MAX7219 or MAX7221 LED driver chip to provide this capability. Connect your Arduino, matrix, and MAX72xx as shown in figure 9).

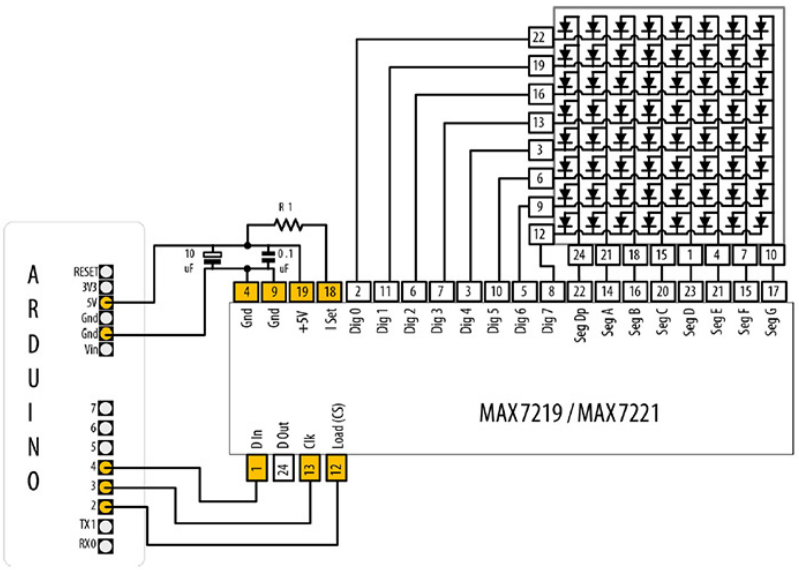

Figure 9: MAX72xx driving an 8x8 LED array.

This sketch is based on the MD_MAX72XX library, which can display text, draw objects on the display, and perform various transformations on the display. You can find the library in the Arduino Library Manager.

```
#include <MD_MAX72xx.h>
// Pins to control 7219
```

```
#define LOAD_PIN 2
#define CLK_PIN 3
#define DATA_PIN 4
// Configure the hardware
#define MAX_DEVICES 1
#define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW
MD_MAX72XX mx =
    MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, LOAD_PIN, MAX_DEVICES);

void setup() { mx.begin(); }

void loop() {
  mx.clear();  // Clear the display
  // Draw rows and columns
  for (int r = 0; r < 8; r++) {
    for (int c = 0; c < 8; c++) {
      mx.setPoint(r, c, true);  // Light each LED
      delay(50);
    }
    // Cycle through available brightness levels
    for (int k = 0; k <= MAX_INTENSITY; k++) {
      mx.control(MD_MAX72XX::INTENSITY, k);
      delay(100);
    }
  }
}
```

A matrix is created by passing the hardware type, pin numbers for the data, load, and clock pins, and also the maximum number of devices (in case you are chaining modules). `loop` clears the display, then uses the `setPoint` method to turn pixels on. After the sketch draws a row, it cycles through the available brightness intensities and moves on to the next row.

The pin numbers shown here are for the green LEDs in the dual-color 8x8 matrix, available from Adafruit (part number 458). This sketch will work with a single-color matrix as well, since it only uses one of the two colors. If you find that your matrix is displaying text backward or not in the orientation you expect, you can try changing the hardware type in the line `#define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW` from `PAROLA_HW` to one of `GENERIC_HW`, `ICSTATION_HW`, or `FC16_HW`.

The resistor (marked R1 in figure 9) is used to control the maximum current that will be used to drive an LED. The MAX72xx datasheet has a table that shows a range of values. The green LED in the LED matrix shown in figure 9 has a forward voltage of 2 volts and a forward current of 20 mA. Table of resistor values (from MAX72xx datasheet) indicates $28k\Omega$, but to add a little

safety margin, a resistor of $30k\Omega$ or $33k\Omega$ would be a suitable choice. The capacitors ($0.1\ \mu F$ and $10\ \mu F$) are required to prevent noise spikes from being generated when the LEDs are switched on and off.

NeoPixel LED

NeoPixels are intelligent RGB LED strips whose elements can be controlled individually. They use WS2812, WS2811 or WS6812 drivers and use a single wire protocol to control the colour of the embedded LED. The LEDs are integral with the controller body and they are sold assembled in various formats: flexible, matrix, ring and as individual elements.
Each cell has five pins (figure 10):

- V_{CC}: 5V power supply for the control circuit;
- V_{DD}: 5V power supply for the LED;
- V_{SS}: ground;
- D_{IN}: data input;
- D_{OUT}: data output to be connected to the next LED in the chain.

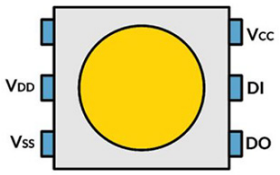

Figure 10: Pinout of a single WS2812 module.

In NeoPixel products, connections are simplified and only three pins are needed:

- $5V$: for power supply;
- GND: for ground, to be shared with the Arduino;
- D_{IN}: for data transmission.

Powering many LEDs requires a lot of power, so a power supply unit or battery with 5V and capable of supplying all the current required by the LEDs must be used. Between $5V$ and GND, it is advisable to put an electrolytic capacitor of a thousand microfarads to provide the inrush required to switch on the various LEDs. The data line must be connected to the Arduino via a 470 Ω (figure 11).
There is no limit to the number of LEDs a NeoPixel element can contain. The only limitations are:

- the power consumed by the strip increases for each LED added (each LED requires a maximum of 60 mA);

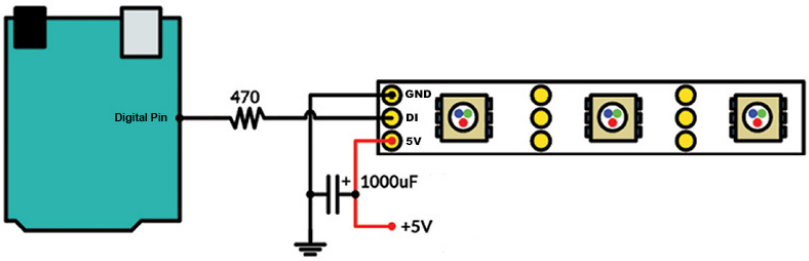

Figure 11: NeoPixel and Arduino interfacing.

- the response time increases as the number of LEDs increases;
- the memory required by the microcontroller increases as the number of LEDs increases.

Libraries exist to manage communication with the individual LEDs. The management library we will see is called *"Adafruit NeoPixel by Adafruit"* and can be installed via the Arduino Library Manager. To use the library, its definition must be included at the start of the sketch:

```
#include <Adafruit_NeoPixel.h>
```

The initialisation of the `Adafruit_NeoPixel` object involves a number of parameters such as the number of LEDs to be controlled, the pin used for communication and the driver model:

```
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN,
NEO_RGB + NEO_KHZ800);
```

The driver type is specified by combining various flag with the following signicate:

- `NEO_KHZ800`: uses an 800 kHz transmission ratez;
- `NEO_RGB`: pixels connected in RGB mode.

The pixels are initialised with:

```
pixels.begin();
```

It is possible to control the colour of each individual pixel using its index to set RGB values with:

```
pixels.setPixelColor(num_pixel, 0,150,0);
```

The colours are then transmitted with:

```
pixels.show();
```

The library functions also include a function to set the brightness of all LEDs:

```
strip.setBrightness(100);
```

Circuit 6. Using the NeoPixel Strip

In this example, we will learn how to use Arduino to control the NeoPixel RGB LED strip and how to use the Adafruit NeoPixel library to set up the NeoPixels. Figure 12 shows a very simple example of connecting the NeoPixel LED strip to the Arduino board. To connect a strip of NeoPixel LEDs to an Arduino board, hook up three wires:

- Power supply (+5V) goes to the plus of a power source.
- Ground (GND) goes to power source ground and should be additionally connected to the board ground if powered from a separate power source.
- Data input (DIN) goes to any digital pin of the board.

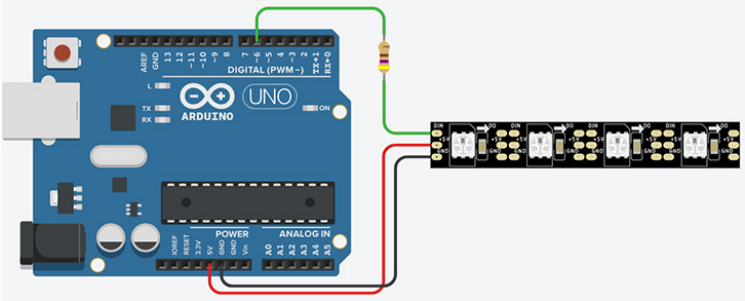

Figure 12: Connecting a NeoPixel strip to the Arduino.

The Adafruit_NeoPixel library allows you to easily turn on a specific LED with a certain intensity and color, or to turn it off. Each LED can be controlled individually. Here we have four LEDs, with the first one numbered 0. For example, to make it light up red, you would use the command: `strip.setPixelColor(0, 255, 0, 0);` However, the LED will only respond to this command if it is followed by `strip.show()`.

```
#include <Adafruit_NeoPixel.h>

#define LED_PIN    6
#define LED_COUNT 4

Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);

void setup() {
        strip.begin();
        strip.clear();
        strip.show();
}

void loop() {
        strip.setBrightness(50);
```

```
        strip.setPixelColor(0, 255, 0, 0);
        strip.show();
        delay(1000);
    strip.setPixelColor(0, 0, 0, 0);
    strip.setPixelColor(1, 0, 255, 0);
        strip.show();
        delay(1000);
    strip.setPixelColor(1, 0, 0, 0);
    strip.setPixelColor(2, 0, 0, 255);
        strip.show();
        delay(1000);
    strip.setPixelColor(2, 0, 0, 0);
    strip.setPixelColor(3, 255, 255, 255);
        strip.show();
        delay(1000);
    strip.setPixelColor(3, 0, 0, 0);
}
```

Circuit 7. Lighting a NeoPixel Ring

The NeoPixel ring is a circular arrangement of individually addressable RGB LEDs, allowing for a wide range of color and brightness combinations.
In the circuit shown in Figure 13, the power supply is connected to the 5V pin, the GND pin is connected to the ground of the circuit, and the DIN pin is connected to a digital pin on the Arduino board.

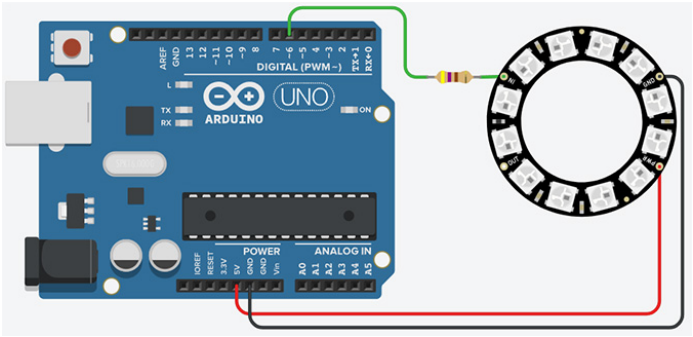

Figure 13: Neopixel ring interfacing with Arduino.

As shown in the code below, define the pin to which the NeoPixel data input is connected and how many LEDs there are in our ring. The next step is to actually declare our NeoPixel object, which we'll call `ring`. In our `setup()` function, we call the `begin()` function on that `ring`. Then, we call `show()` to clear all the LEDs, and finally, we set the brightness with `setBrightness()`, which we call once at the beginning to tell our NeoPixel what the maximum brightness will be.

Next, let's start with a for loop that runs from 0 to the number of LEDs in our ring. Then, we set the color of the individual LED in our ring. The function `setPixelColor()` takes the arguments, in order: numLED, red, green, and blue. We'll enter i as the numLED so that the for loop runs through each one. For the color, we use random values for red, green, and blue. Call `ring.show()` afterwards to actually update the color in the ring. Finally, we add a delay of 50 milliseconds after applying each color to create a loading animation effect.
Next, we take the same loop and reverse it. To do this, we start from the last LED and count backwards to 0. Then, we set the pixel color to 0, 0, 0, which means no color, and add those same two lines..

```
#include <Adafruit_NeoPixel.h>

#define LED_PIN    6
#define LED_COUNT 16

Adafruit_NeoPixel ring(LED_COUNT, LED_PIN, NEO_RGB + NEO_KHZ800);

void setup() {
  ring.begin();
  ring.show();
  ring.setBrightness(50);
}
void loop() {
  for(int i = 0; i < ring.numPixels(); i++){
    ring.setPixelColor(i, random(255), random(255), random(255));
    ring.show();
    delay(50);
  }
  for(int i = ring.numPixels()-1; i >= 0; i--){
    ring.setPixelColor(i, 0, 0, 0);
    ring.show();
    delay(50);
  }
}
```

Circuit 8. Controlling a NeoPixel Ring

This sketch uses the Adafruit Neopixels library (installed using the Arduino Library Manager) to change LED colors based on readings from an analog pin. Figure 14 shows the connection for a NeoPixel ring and a potentiometer to control the color:

```
#include <Adafruit_NeoPixel.h>
```

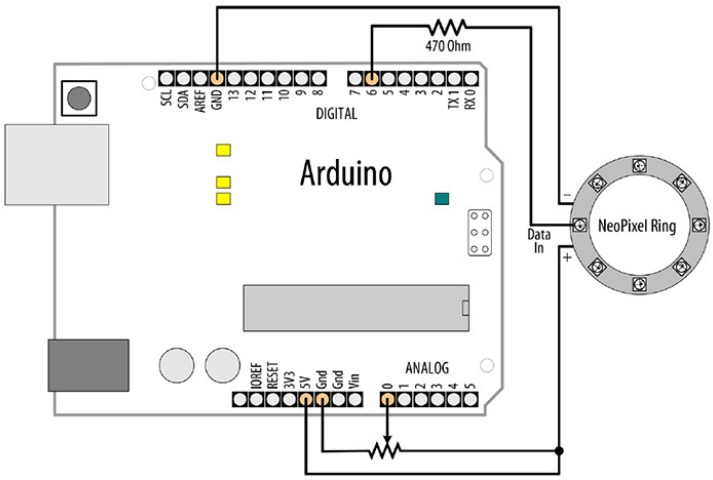

Figure 14: Connecting a NeoPixel ring.

```
const int sensorPin = A0;   // analog pin for sensor
const int ledPin = 6;       // the pin the LED strip is connected to
const int count = 8;        // how many LEDs in the strip
// declare LED strip
Adafruit_NeoPixel leds = Adafruit_NeoPixel(count, ledPin, NEO_GRB + NEO_KHZ800);

void setup() {
  leds.begin();  // initialize LED strip
  for (int i = 0; i < count; i++) {
    leds.setPixelColor(i, leds.Color(0, 0, 0));  // turn each LED off
  }
  leds.show();  // refresh the strip with the new pixel values (all off)
}

void loop() {
  static unsigned int last_reading = -1;
  int reading = analogRead(sensorPin);
  if (reading != last_reading) {  // If the value has changed
    // Map the analog reading to the color range of the NeoPixel
    unsigned int mappedSensorReading = map(reading, 0, 1023, 0, 65535);
    // Update the pixels with a slight delay to create a sweeping effect
    for (int i = 0; i < count; i++) {
      leds.setPixelColor(
          i, leds.gamma32(leds.ColorHSV(mappedSensorReading, 255, 128)));
      leds.show();
      delay(25);
    }
    last_reading = reading;
```

```
  }
}
```

You specify the number of LEDs in the strip `count`, the Arduino pin the data line is connected to `ledPin`, and the type of LED strip you are using (in this case: `NEO_GRB+NEO_KHZ800`). To set the color of an individual LED you use the `led.setPixelColor` method. You need to specify the number of the LED (starting at 0 for the first one) and the desired color. To transfer data to the LEDs you need to call `led.show`. You can alter multiple LED's values before calling `led.show` to make them change together. Values not altered will remain at their previous settings. When you create the `Adafruit_NeoPixel` object, all the values are initialized to 0.
The NeoPixel library includes its own function for converting a hue to an RGB value: `ColorHSV`. The first argument is the hue, the second is the color saturation, and the third is brightness. The `gamma32` function performs a conversion on the output of `ColorHSV` to compensate between the way that computers represent colors and the way that humans perceive them.