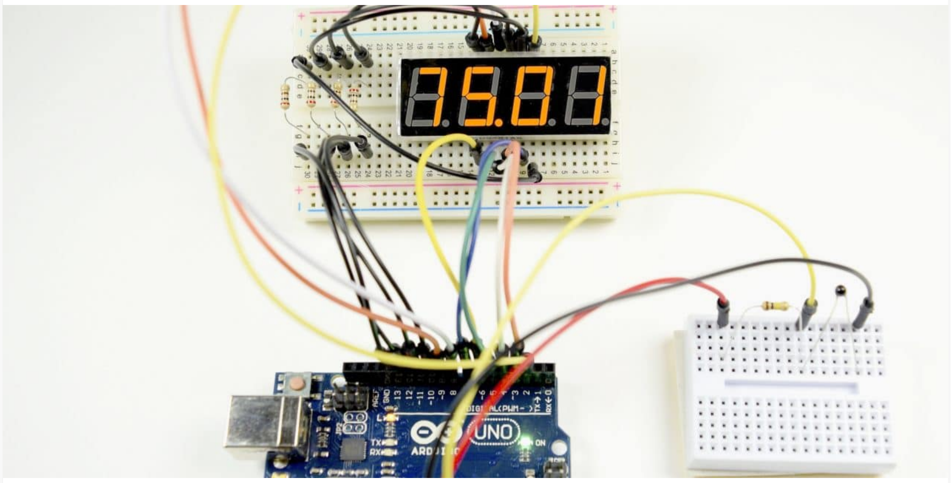
HOW TO SET UP 7-SEGMENT DISPLAYS ON THE ARDUINO

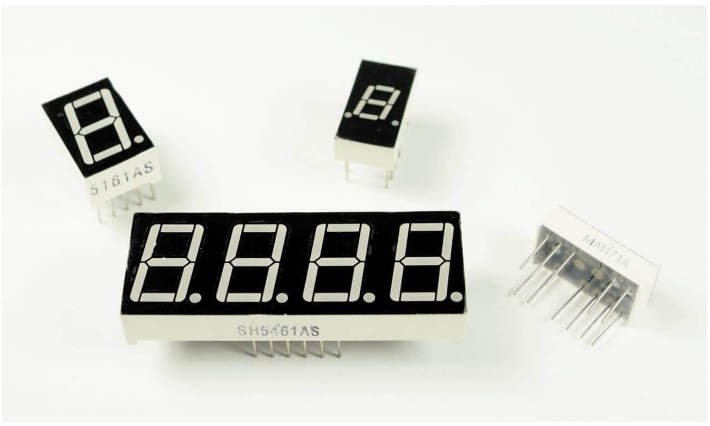
Posted by Krishna Pattabiraman | Arduino | 3



Seven segment displays are used in many day to day consumer devices like microwave ovens, washing machines, and air conditioners. They are a simple but effective way to display numerical data like time or quantity. Since they are made out of LEDs, they are a low cost option for displaying information.

In this tutorial I'm going to show you how to set up and program single digit and multi-digit seven segment displays on an Arduino.

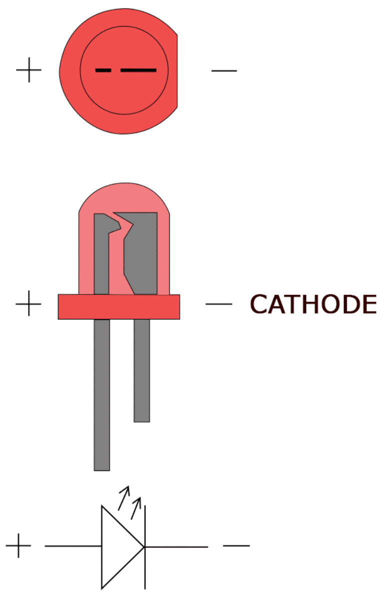
Seven segment displays come in a wide variety of sizes and colors. Red, blue, and green are the easiest colors to find. Sizes range from small 0.56 inch displays up to large 4 inch and even 6.5 inch displays. Some displays have a single digit, and others have two or four.



Before we start working with 7 segment displays, we need to understand some of the basics of LEDs and how to control them.

LED BASICS

A single LED consists of two terminals, an anode and a cathode. The anode is the positive terminal and the cathode is the negative terminal:

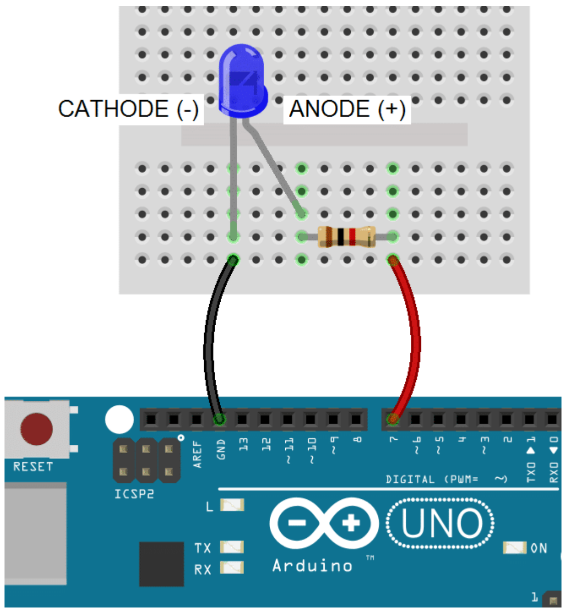


To power the LED, you connect the cathode to ground and the anode to the voltage supply. The LED can be turned on or off by switching power at the anode or the cathode.

JLCPCB is the largest PCB prototype enterprise in China with 290,000+ customers worldwide and 8000+ online orders per day.

ANODE TO GPIO

With the LED's anode connected to a digital pin, the cathode is connected to ground:



Note: All LEDs need a current limiting resistor placed on either the anode side or cathode side to prevent the LED from burning out. The resistor value will determine how bright the LED shines. 1K ohms is a good place to start, but you can calculate the ideal value with an LED resistor calculator.

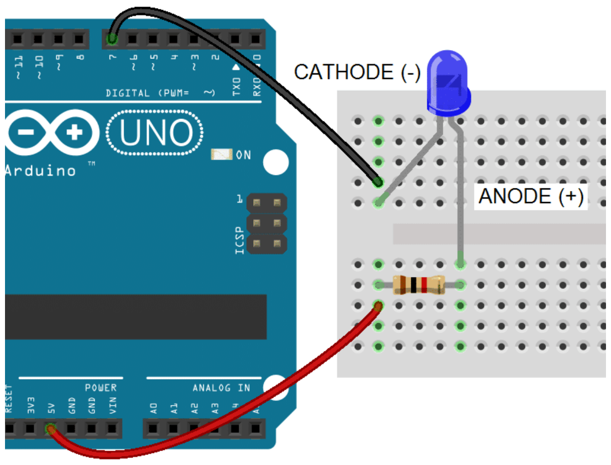
To light up an LED with the anode connected to a digital pin, you set the digital pin to HIGH:

```
void setup(){
    pinMode(7, OUTPUT);
    digitalWrite(7, HIGH);
}

void loop(){
}

```

In the `void setup()` block, we configure GPIO pin 7 as an output with `pinMode(7, OUTPUT);` and drive it high with `digitalWrite(7, HIGH);`.

CATHODE TO GPIO

With an LED's cathode connected to a digital pin, the anode is connected to Vcc. To turn on the LED, the digital pin is switched LOW, which completes the circuit to ground:

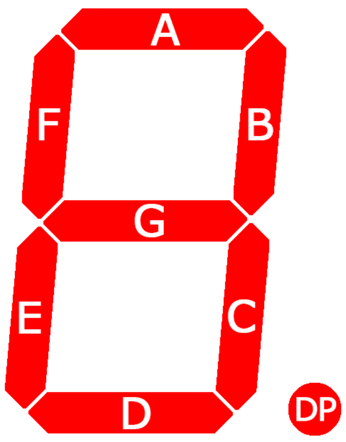


In this case we drive GPIO pin 7 LOW with `digitalWrite(7, LOW);`. This closes the circuit and allows current to flow from Vcc to ground:

```
void setup(){
    pinMode(7, OUTPUT);
    digitalWrite(7, LOW);
}

void loop(){
}

```

HOW 7-SEGMENT DISPLAYS WORK

Seven segment displays consist of 7 LEDs, called segments, arranged in the shape of an “8”. Most 7-segment displays actually have 8 segments, with a dot on the right side of the digit that serves as a decimal point. Each segment is named with a letter A to G, and DP for the decimal point:

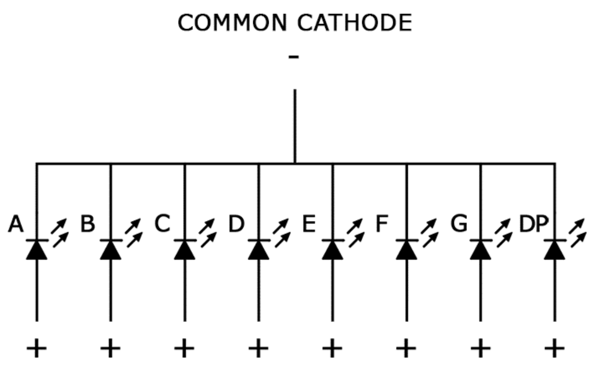


Each segment on the display can be controlled individually, just like a regular LED.

There are two types of 7-segment displays – *common cathode* and *common anode*.

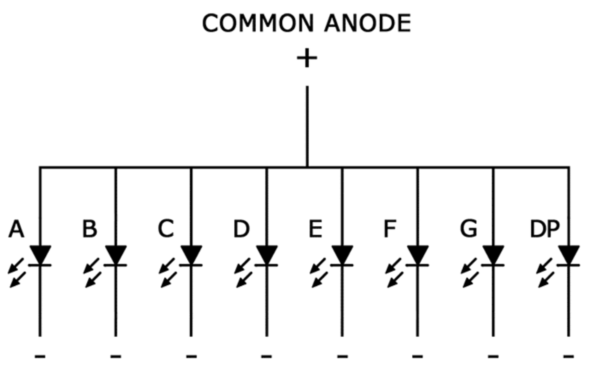
COMMON CATHODE DISPLAYS

In common cathode displays, all of the cathodes are connected to ground and individual segments are turned on and off by switching power to the anodes:



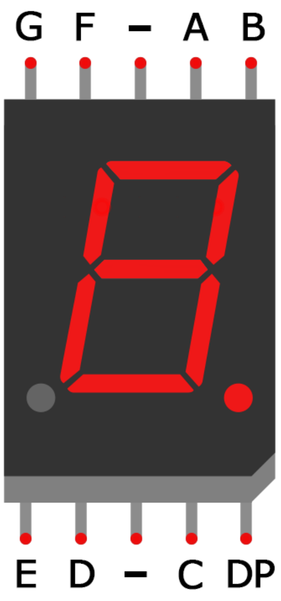
COMMON ANODE DISPLAYS

In common anode displays, all of the anodes are connected to Vcc, and individual segments are turned on and off by switching power to the cathodes:



CONNECTING 7-SEGMENT DISPLAYS TO THE ARDUINO

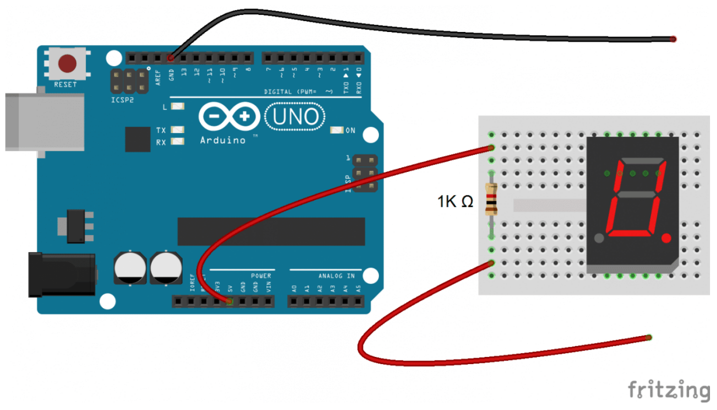
Single digit seven segment displays typically have 10 pins. Two pins connect to ground, and the other 8 connect to each of the segments. Here is a pin diagram of the popular 5161AS common cathode display:



Before you can connect your display to the Arduino, you need to know if it's common anode or common cathode, and which pins connect to each segment. This information should be in the datasheet, but if you can't find the datasheet or you don't know your display's part number, I'll show you how to figure this out below…

HOW TO TELL IF YOU HAVE A COMMON ANODE OR COMMON CATHODE DISPLAY

To determine if a display is common anode or common cathode, you can probe the pins with a test circuit constructed like this:



Connect the ground (black) wire to any pin of the display. Then insert the positive (red) wire into each one of the other pins. If no segments light up, move the ground wire over to another pin and repeat the process. Do this until at least one segment lights up.

When the first segment lights up, leave the ground wire where it is, and connect the positive wire to each one of the other pins again. If a different segment lights up with each different pin, you have a common cathode display. The pin that's connected to the ground wire is one of the common pins. There should be two of these.

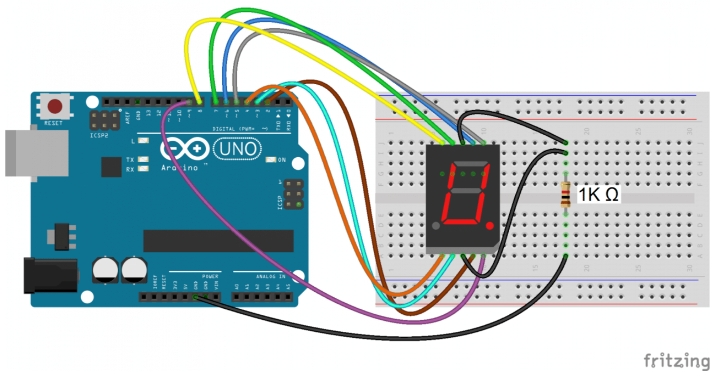
If two different pins light up the same segment, you have a common anode display. The pin that's connected to the positive wire is one of the common pins. Now if you connect the ground wire to each one of the other pins, you should see that a different segment lights up with each different pin.

HOW TO DETERMINE THE PINOUT FOR YOUR DISPLAY

Now draw a diagram showing the pins on your display. With the common pin connected to the ground wire (common cathode) or positive wire (common anode), probe each pin with the other wire. When a segment lights up, write down the segment name (A-G, or DP) next to the corresponding pin on your diagram.

CONNECTING SINGLE DIGIT DISPLAYS TO THE ARDUINO

Once you have the pin layout figured out, connecting the display to an Arduino is pretty easy. This diagram shows how to connect a single digit 5161AS display (notice the 1K ohm current limiting resistor connected in series with the common pins):



In the example programs below, the segment pins connect to the Arduino according to this table:

Segment Pin	Arduino Pin
A	6
B	5
C	2
D	3
E	4
F	7
G	8
DP	9

PROGRAMMING SINGLE DIGIT DISPLAYS

INSTALL THE LIBRARY

We'll use a library called SevSeg to control the display. The SevSeg library works with single digit and multi-digit seven segment displays. You can download the library's ZIP file from GitHub or download it here:

zip SevSeg.zip

To install it, open the Arduino IDE, go to Sketch > Include Library > Add .ZIP Library, then select the SevSeg ZIP file that you downloaded.

PRINTING NUMBERS TO THE DISPLAY

This program will print the number "4" to a single digit 7-segment display:

```
    byte hardwareConfig = COMMON_CATHODE;
    sevseg.begin(hardwareConfig, numDigits, digi
    sevseg.setBrightness(90);
}

void loop(){
        sevseg.setNumber(4);
        sevseg.refreshDisplay();
}

```

In this program, we create a `sevseg` object on line 2. To use additional displays, you can create another object and call the relevant functions for that object. The display is initialized with the `sevseg.begin()` function on line 11. The other functions are explained below:

`hardwareConfig = COMMON_CATHODE;` This sets the type of display. I'm using a common cathode, but if you're using a common anode then use `COMMON_ANODE` instead.

`byte numDigits = 1;` This sets the number of digits on your display. I'm using a single digit display, so I set it to 1. If you're using a 4 digit display, set this to 4.

`byte digitPins[] = {};` Creates an array that defines the ground pins when using a 4 digit or multi-digit display. Leave it empty if you have a single digit display. For example, if you have a 4 digit display and want to use Arduino pins 10, 11, 12, and 13 as the digit ground pins, you would use this: `byte digitPins[] = {10, 11, 12, 13};`. See the 4 digit display example below for more info.

`byte segmentPins[] = {6, 5, 2, 3, 4, 7, 8, 9};` This declares an array that defines which Arduino pins are connected to each segment of the display. The order is alphabetical (A, B, C, D, E, F, G, DP where DP is the decimal point). So in this case, Arduino pin 6 connects to segment A, pin 5 connects to segment B, pin 2 connects to segment C, and so on.

`resistorsOnSegments = true;` This needs to be set to true if your current limiting resistors are in series with the segment pins. If the resistors are in series with the digit pins, set this to false. Set this to true when using multi-digit displays.

`sevseg.setBrightness(90);` This function sets the brightness of the display. It can be adjusted from 0 to 100.

`sevseg.setNumber();` This function prints the number to the display. For example, `sevseg.setNumber(4);` will print the number "4" to the display. You can also print numbers with decimal points. For example, to print the number "4.999", you would use `sevseg.setNumber(4999, 3);`. The second parameter (the 3) defines where the decimal point is located. In this case it's 3 digits from the right most digit. On a single digit display, setting the second parameter to "0" turns on the decimal point, while setting it to "1" turns it off.

`sevseg.refreshDisplay();` This function is required at the end of the loop section to continue displaying the number.

COUNT UP TIMER

This simple program will count up from zero to 9 and then loop back to the start:

```
 1  #include "SevSeg.h"
 2  SevSeg sevseg;
 3
 4  void setup(){
 5      byte numDigits = 1;
 6      byte digitPins[] = {};
 7      byte segmentPins[] = {6, 5, 2, 3, 4, 7, 8,
 8      bool resistorsOnSegments = true;
 9
10      byte hardwareConfig = COMMON_CATHODE;
```

The code is similar to the previous sketch. The only difference is that we create a count variable "i" in the for statement on line 16 and increment it one number at a time.

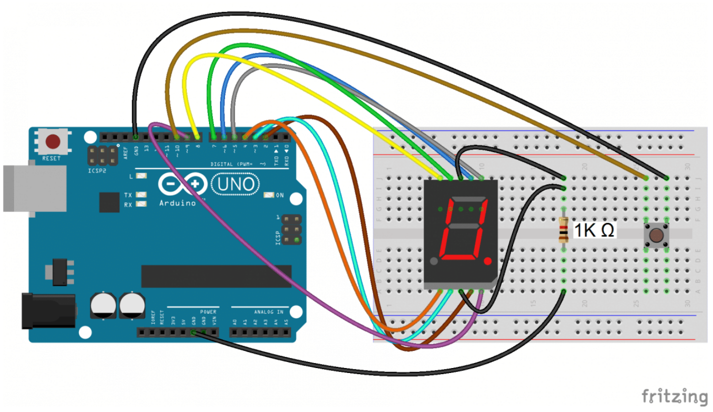
The `sevseg.setNumber(i, i%2);` function prints the value of i. The i%2 argument divides i by 2 and returns the remainder, which causes the decimal point to turn on every other number.

The count up timer is a nice way to demonstrate the basics of how to program the display, but now let's try to make something more interesting.

ROLLING DICE

This example consists of a push button and a single 7 segment display. Every time the push button is pressed and held, the display loops through numbers 0-9 rapidly. Once the button is released, the display continues to loop for a period of time almost equal to the time the button was pressed, and then displays a number along with the decimal point to indicate the new number.

To build the circuit (with the 5161AS display), connect it like this:



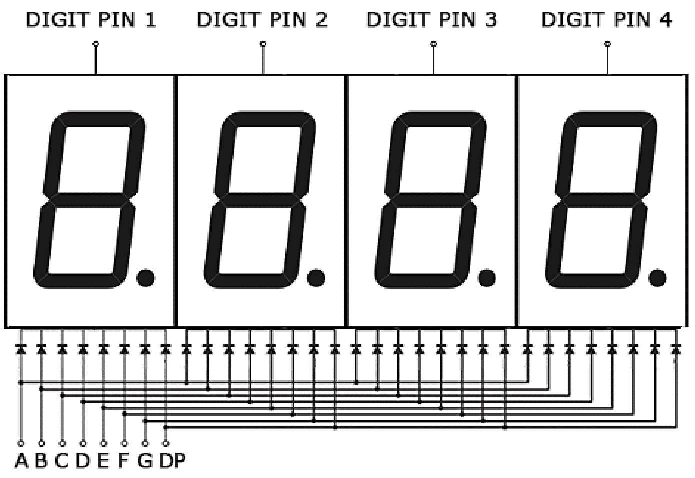
Then upload this program to the Arduino:

```
#include "SevSeg.h"
SevSeg sevseg;

const int  buttonPin = 10;    // the pin that th
int buttonState = 0;          // current state o
int lastButtonState = LOW;    // previous state
int buttonPushCounter = 0;    // counter for the
long counter = 0;
long max_long_val = 2147483647L;

```

4 DIGIT 7-SEGMENT DISPLAYS

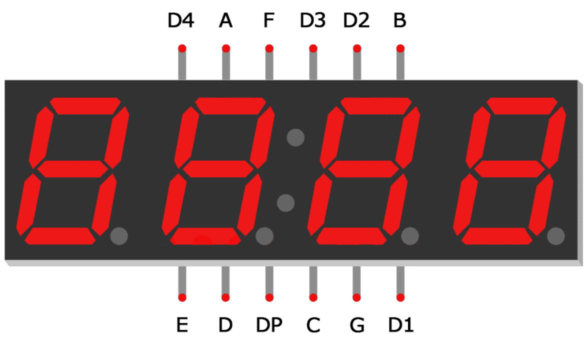
So far we have only worked with single digit 7-segment displays. To display information such as the time or temperature, you will want to use a 2 or 4 digit display, or connect multiple single digit displays side by side.

In multi-digit displays, one segment pin (A, B, C, D, E, F, G, and DP) controls the same segment on all of the digits. Multi-digit displays also have separate common pins for each digit. These are the digit pins. You can turn a digit on or off by switching the digit pin.



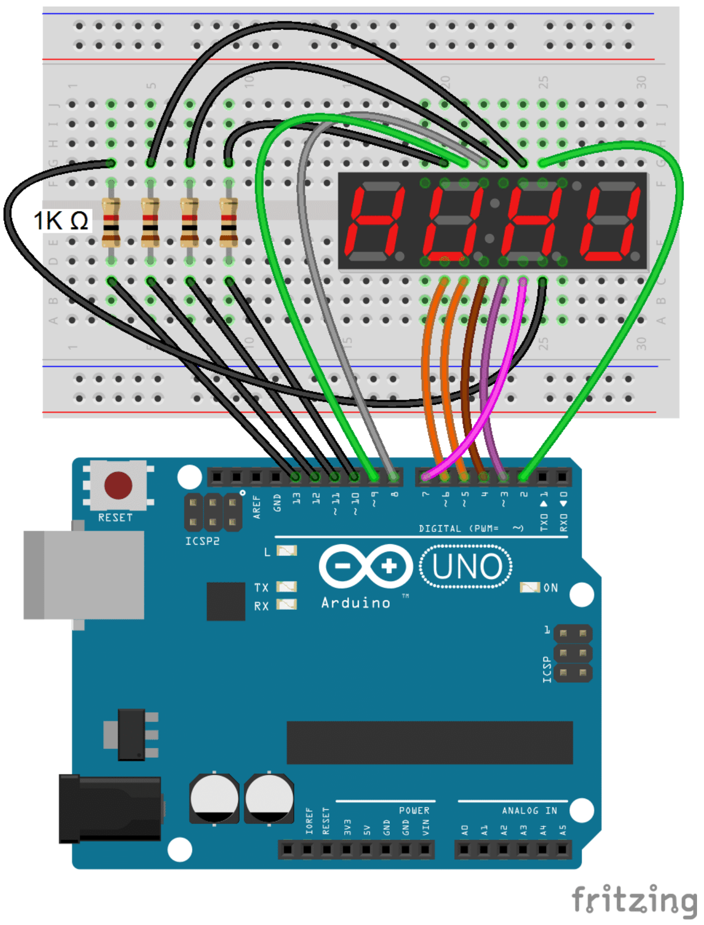
I'm using a 4 digit 7-segment display with the model number 5641AH, but the wiring diagrams below will also work with the 5461AS.

Here is a diagram showing the pinout of these displays:



The digit pins D1, D2, D3 and D4 need to be connected to current limiting resistors, since they

are the common terminals of the digits.
The connections are shown below:



This simple program will print the number “4.999” to the display:

```
11    byte hardwareConfig = COMMON_CATHODE;
12    sevseg.begin(hardwareConfig, numDigits, digitP
13    sevseg.setBrightness(90);
14  }
15
16  void loop(){
17      sevseg.setNumber(4999, 3);
18      sevseg.refreshDisplay();
19  }
20
```

In the code above, we set the number of digits in line 5 with `byte numDigits = 4;`.

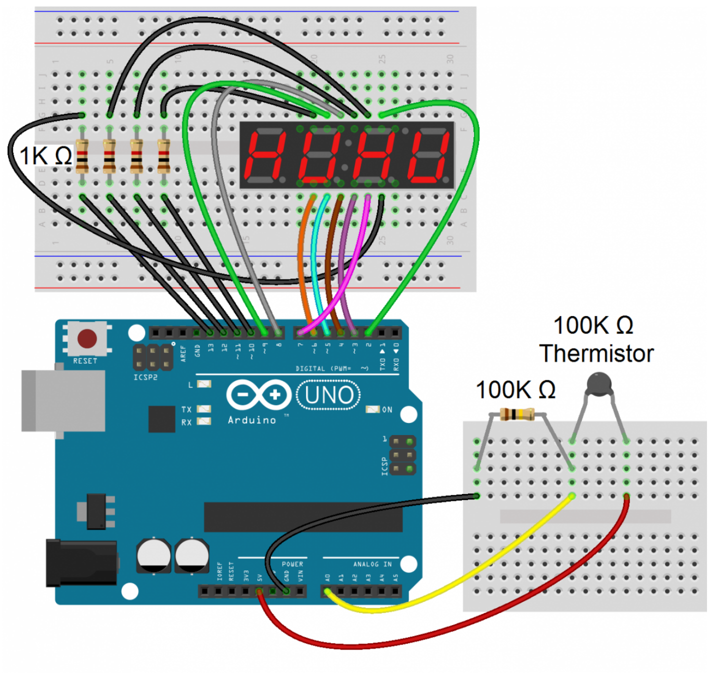
Since multi-digit displays use digit pins, we also need to define which Arduino pins will connect to the digit pins. Using `byte digitPins[] = {10, 11, 12, 13};` on line 6 sets Arduino pin 10 as the first digit pin, Arduino pin 11 to the second digit pin, and so on.

To print numbers with a decimal point, we set the second parameter in `sevseg.setNumber(4999,` 3); to three, which puts it three decimal places from the right most digit.

TEMPERATURE DISPLAY

This example reads the temperature from a thermistor and displays it on a 4 digit display.

Connect the circuit like this:



If you have questions about using a thermistor, or just want to learn more about them, check out our other tutorial on using a thermistor with an Arduino.

Once everything is connected, upload this code to the Arduino:

```
28   static unsigned long timer = millis();
29
30   if (millis() >= timer) {
31     timer += 300;
32     sevseg.setNumber(T, 2);
33   }
34
35   sevseg.refreshDisplay();
36 }
37
```

This will output the temperature in Fahrenheit. To display the temperature in Celsius, comment out line 28.

By itself, the display will update every time the temperature changes even slightly. This creates an annoying flickering. In order to deal with this, we introduce a timer mechanism, where we only read the value from the thermistor every 300 milliseconds (lines 30 to 34).

The temperature variable "T" is printed to the display on line 35 with `sevseg.setNumber(T, 2, false);`.

Hopefully this article should be enough to get you started using seven segment displays. If you want to display readings from other sensors, the example program above can easily be modified to do that. If you have any questions or trouble setting up these circuits, feel free to leave a comment below.

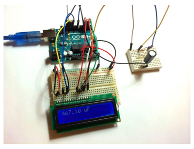
SHARE:

ABOUT THE AUTHOR

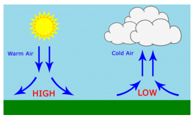
Krishna Pattabiraman

Krishna is a frequent guest writer on Circuit Basics and the founder of www.codelectron.com.

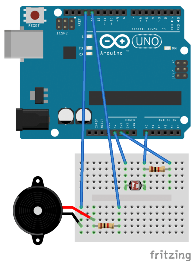
RELATED POSTS



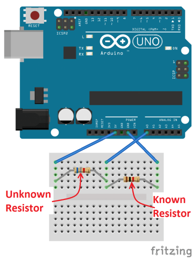
How to Make an Arduino Capacitance Meter



How to Set Up the BMP180 Barometric Pressure Sensor on an Arduino



Getting Started with the Arduino – Controlling the LED (Part 2)



How to Make an Arduino Ohm Meter

3 COMMENTS

B Mc Carthy on December 23, 2017 at 7:14 pm

Thanks for these tutorials about using the NTC probe with the Arduino.
What I would also like is using setpoint buttons to control a relais.
Many thanks.

REPLY

Spike on February 4, 2018 at 10:53 pm

Thank you so much! FINALLY got it working, now I can tinker all I want ☺
This is by far the clearest explanation out there.

REPLY

Robo IoT on May 5, 2018 at 2:54 pm

kid wow https://t.co/DiuUrfMbzF

REPLY

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

COMMENT

NAME * EMAIL * WEBSITE

☐ Save my name, email, and website in this browser for the next time I comment.

☑ Send new posts to my inbox.

☐ Notify me of follow-up comments by email. POST COMMENT

Raspberry Pi Arduino DIY Electronics Programming Videos Resources About Contact Us Privacy Policy