Introduction to robotics

4th lab

Remember, when possible, choose the wire color accordingly:

- **BLACK** for **GND (dark colors if not available)**
- **RED** for **POWER (3.3V / 5V / VIN) (bright colors if not available)**
- **Bright Colored** for read and write signal (use **red** when none available and **black** only as a last option)
- We know it is not always possible to respect this due to lack of wires, but the first rule is **DO NOT feUSE BLACK FOR POWER OR RED FOR GND!**

Now, let's pick it up where we left off…
Pull out your Arduino and breadboard and connect them like in the schematic. This is to "power up" the breadboard so we can easily have access to **5V** and **GND**.

Attention! Remember how the breadboard works. Use correct wire colors.

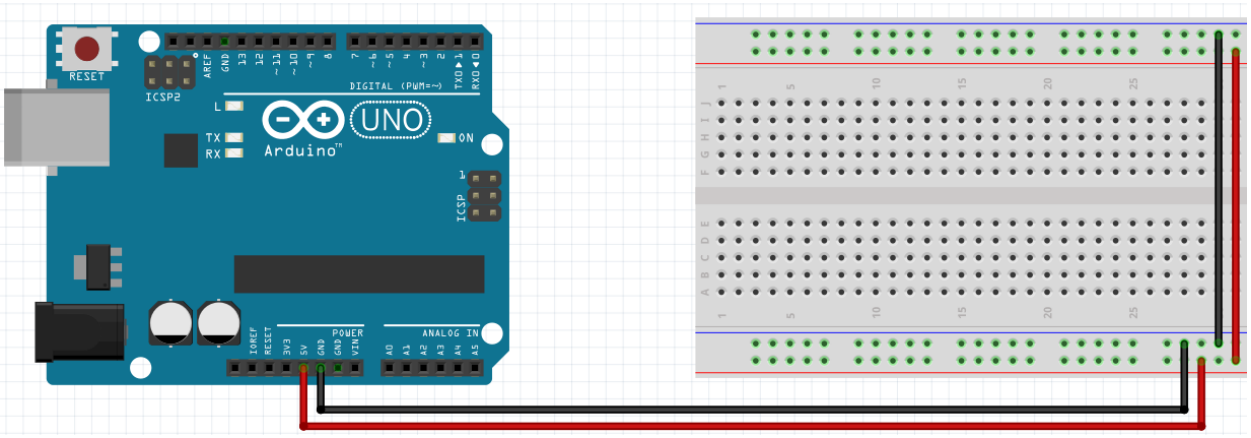


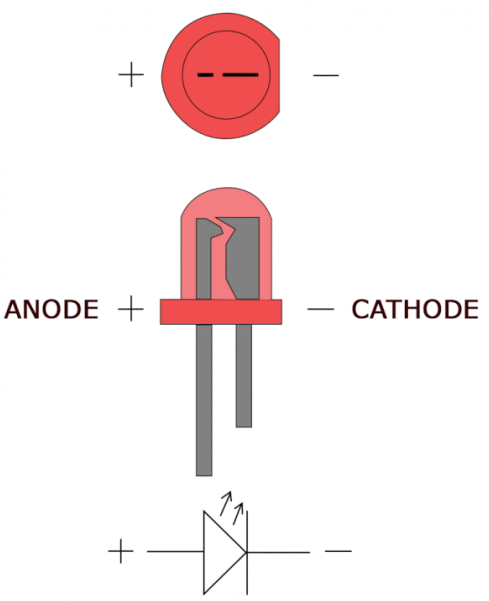
Fig. 1.1 - Default setup

0. Homework check

1. 7-segment display

1.1 Recap

Remember the LED basics:



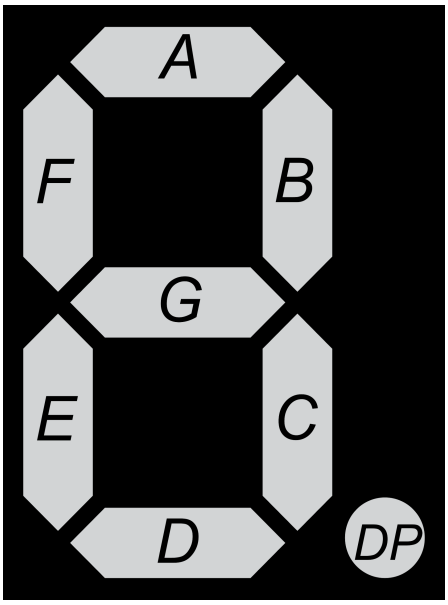
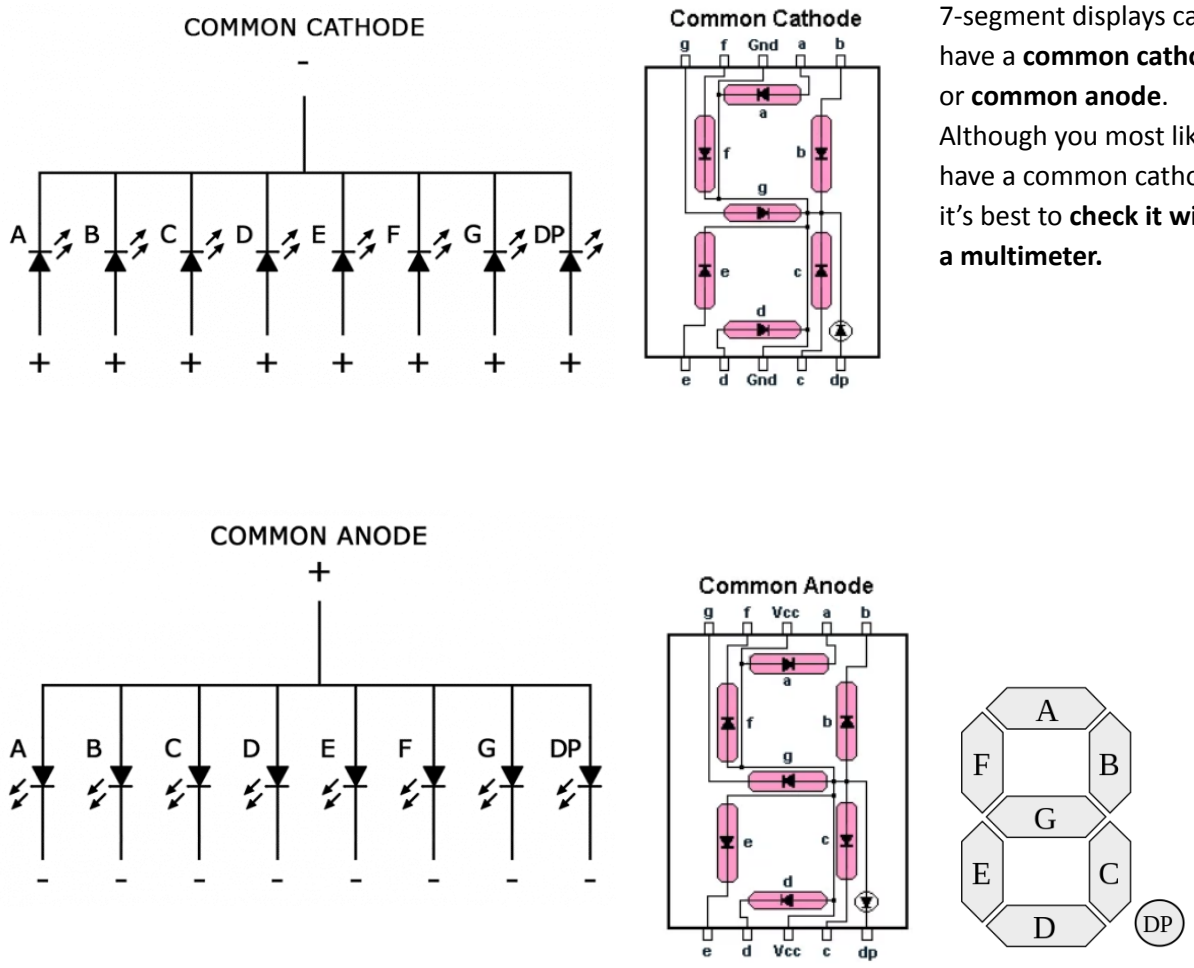


Fig. 1.2 - Light Emitting Diode (left), 7-segment-display LEDs encoding (right)
(source: https://en.wikipedia.org/wiki/Seven-segment_display)

1.2 Introduction

The 7-segment display, also written as “seven-segment display”, consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a numerical digit (both Decimal and Hex) to be displayed. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.



Like the RGB LED, 7-segment displays can have a **common cathode** or **common anode**. Although you most likely have a common cathode, it’s best to **check it with a multimeter.**

Fig. 1.3 - 7 segment display setup: Common Cathode (top) vs Common Anode (bottom)

1.3 Displaying characters

The display system is mainly used for numbers and not letters. That is why some letters are not compatible and are easily mistaken for a number. Short messages giving status information (e.g. "no dISC" on a CD player) are also commonly represented on 7-segment displays. In the case of such messages it is not necessary for every letter to be unambiguous, merely for the words as a whole to be readable

Punctuation encodings

Glyph	Display	Name(s)
-		Minus and Hyphen
=	*	Equals
_		Underscore
°		Degree
[		Left square bracket
]		Right square bracket
		Space

Hexadecimal encodings for displaying the digits 0 to F[11][12]

Digit	Display	gfedcba	abcdefg	a	b	c	d	e	f	g
0		0x3F	0x7E	**on**	**on**	**on**	**on**	**on**	**on**	off
1		0x06	0x30	off	**on**	**on**	off	off	off	off
2		0x5B	0x6D	**on**	**on**	off	**on**	**on**	off	**on**
3		0x4F	0x79	**on**	**on**	**on**	**on**	off	off	**on**
4		0x66	0x33	off	**on**	**on**	off	off	**on**	**on**
5		0x6D	0x5B	**on**	off	**on**	**on**	off	**on**	**on**
6		0x7D	0x5F	**on**	off	**on**	**on**	**on**	**on**	**on**
7		0x07	0x70	**on**	**on**	**on**	off	off	off	off
8		0x7F	0x7F	**on**	**on**	**on**	**on**	**on**	**on**	**on**
9		0x6F	0x7B	**on**	**on**	**on**	**on**	off	**on**	**on**
A		0x77	0x77	**on**	**on**	**on**	off	**on**	**on**	**on**
b		0x7C	0x1F	off	off	**on**	**on**	**on**	**on**	**on**
C		0x39	0x4E	**on**	off	off	**on**	**on**	**on**	off
d		0x5E	0x3D	off	**on**	**on**	**on**	**on**	off	**on**
E		0x79	0x4F	**on**	off	off	**on**	**on**	**on**	**on**
F		0x71	0x47	**on**	off	off	off	**on**	**on**	**on**

Fig. 1.4 - Encodings combinations for punctuations (left) and digits 0 to F (right) (source: https://en.wikipedia.org/wiki/Seven-segment_display).

Example of short messages:

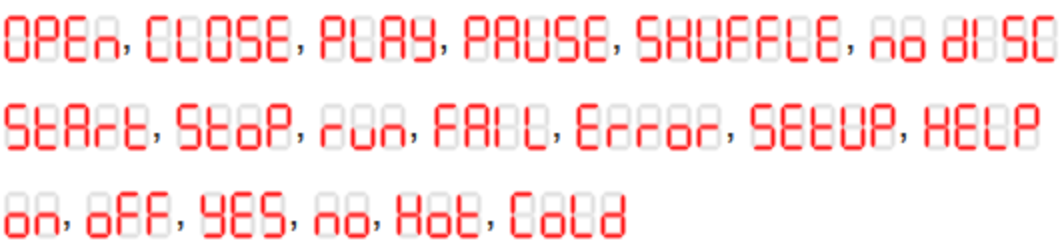


Fig. 1.5 - Different messages that can be written on 4 digit 7 segment display

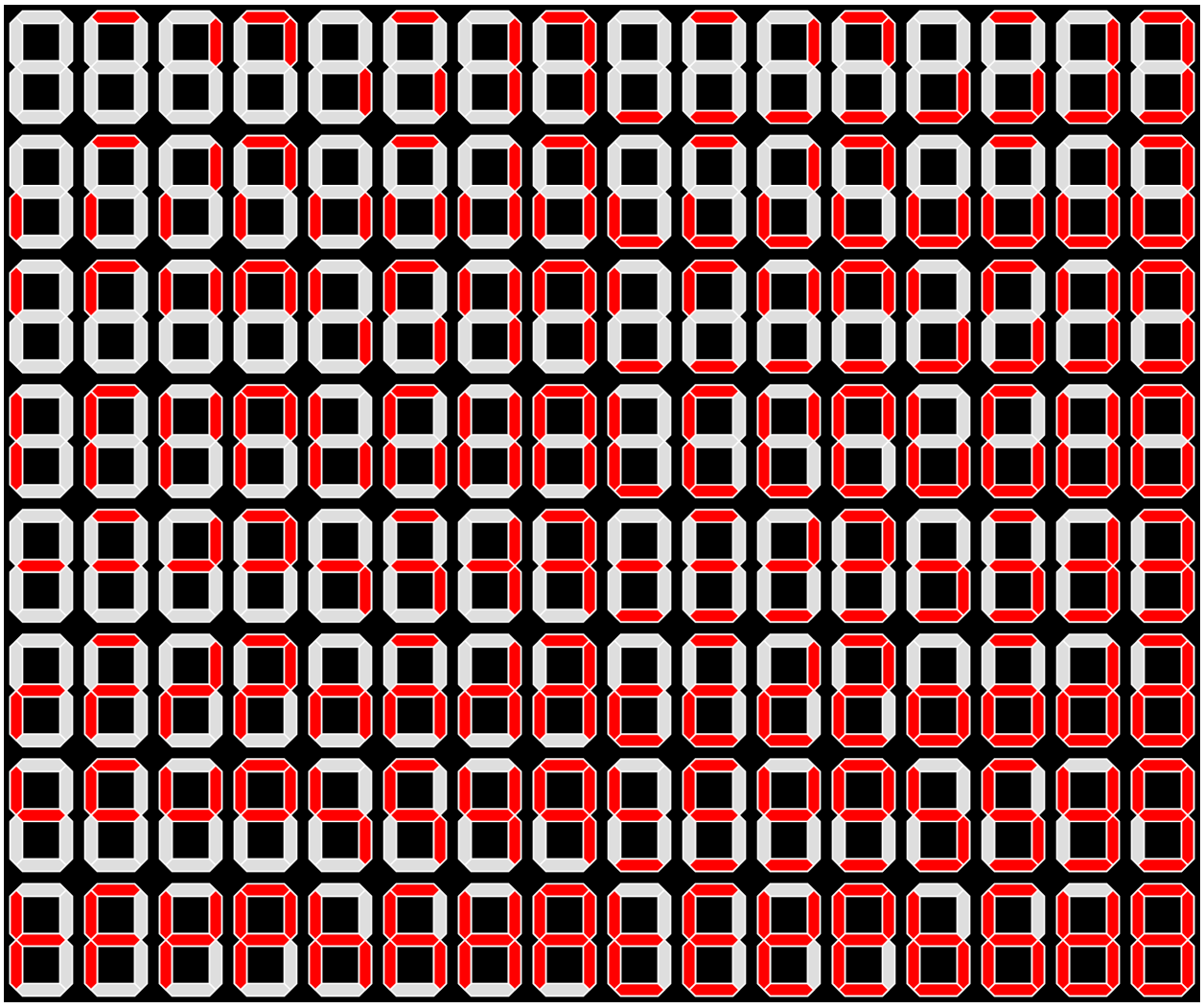


Fig. 1.6 - 16x8-grid showing the 128 states of a seven-segment display

To determine which type of display you have, probe the pins with a multimeter. Try connecting the middle pin to the **COM - black -** of the multimeter, and any other pin (except the middle one on the other side) to the **V+ - red - .** If any of the segments start lighting, it's a common cathode. If not, try reversing the wires; if any of the segments start lighting then it's a common anode.

1.4 Active HIGH vs Active LOW LEDS configurations

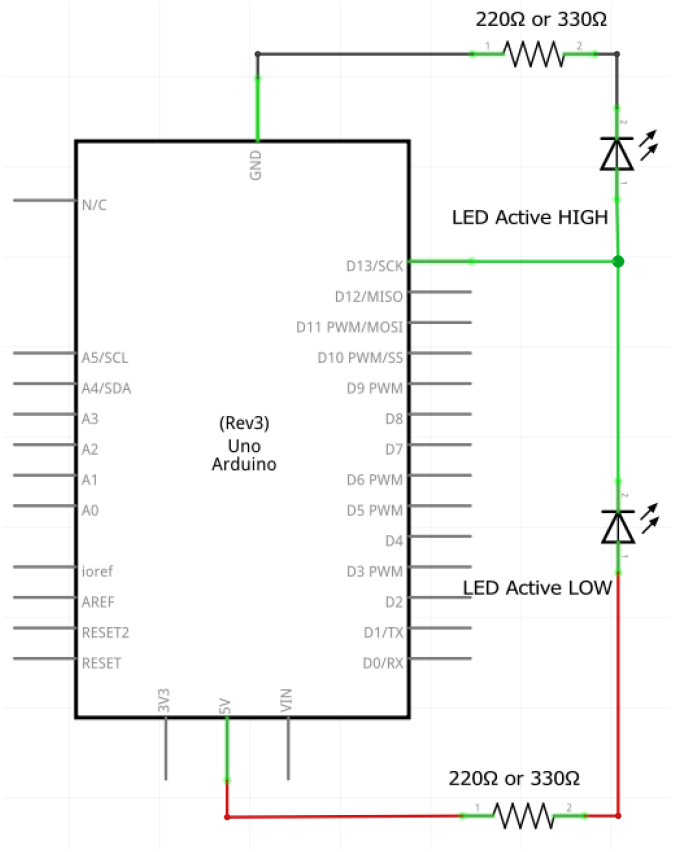
LEDs, like many electronic components, can be operated in two primary configurations: **active high** and **active low**. In an active high setup, an LED is lit when a positive voltage is applied to its anode (positive terminal), with the cathode (negative terminal) grounded. Conversely, in an active low configuration, the LED's anode is constantly connected to a positive voltage, and it illuminates when its cathode is grounded.

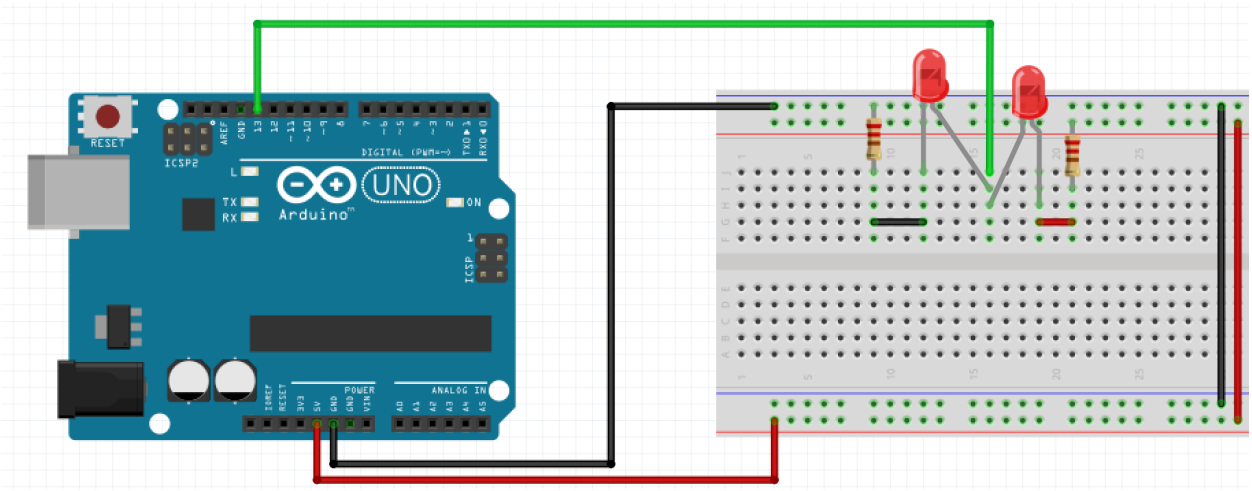
This concept extends to RGB LEDs, which essentially contain three separate LEDs (Red, Green, and Blue) within a single package. An RGB LED can be Common Anode (CA) or Common Cathode (CC). In a CA RGB LED, all three LEDs share a common anode and light up when their individual cathodes are grounded (active low). In a CC RGB LED, the three LEDs share a common cathode and illuminate when a positive voltage is applied to their individual anodes (active high).

Building on this foundation, let's consider the 7-segment display, which is essentially seven individual LEDs arranged to form numbers and certain letters. These displays come in two variants: Common Anode (CA) and Common Cathode (CC). In a CA 7-segment display, all seven segments share a common anode. They light up when their specific cathodes are grounded, making it an active low device. On the other hand, a CC 7-segment display has a common cathode for all segments, and they illuminate when a positive voltage is applied to their respective anodes, classifying it as an active high device.

Understanding the active high vs. active low principle, and its application in devices like RGB LEDs and 7-segment displays, is fundamental for effective circuit design and troubleshooting.

1.5 Active HIGH vs Active LOW LEDS connections





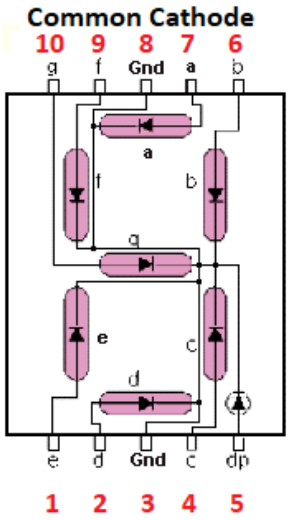
1.6 Connecting the 7-segment display

The two primary types of 7-segment displays based on internal connections are the Common Cathode and Common Anode displays.

Understanding the distinction between these two types is essential as they require different wiring and programming strategies. The key difference is in the common pin: in Common Cathode displays, the negative side (cathode) of all the LEDs is connected together, while in Common Anode displays, the positive side (anode) is connected in common.

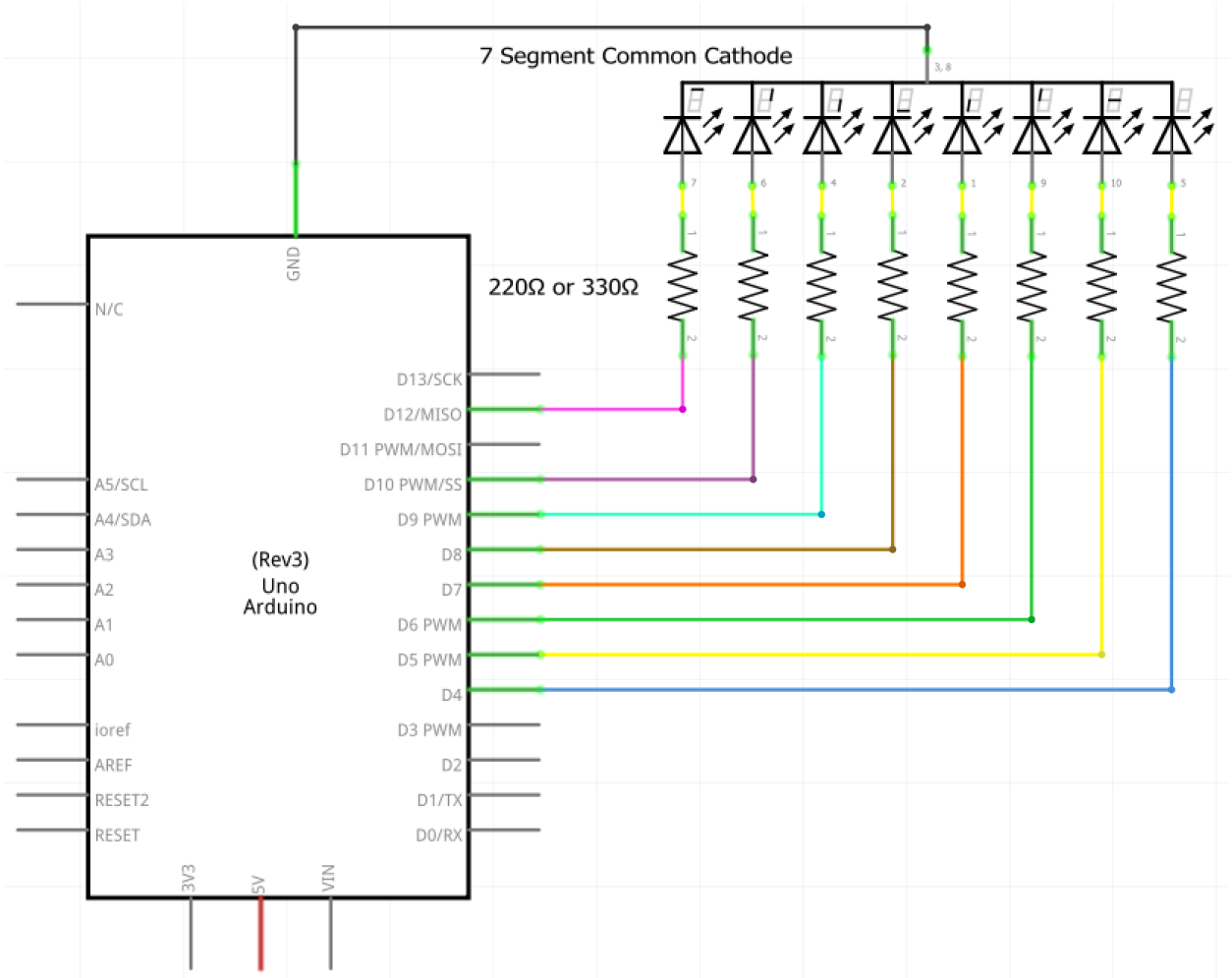
First of all, check which type of display you have. Common cathode must have Ax at the end. (x can have any value). If you don’t have a common cathode, ask for another one.
Common cathode: (5161Ax)
Connecting the 7 segment display to the Arduino board:
Attention! The pins have a corresponding letter and a number based on their circular order, starting with 1 from the bottom left. The red numbers are **NOT** the corresponding Arduino pins.

1.6.1 Connecting the 7-segment display - **Common Cathode**

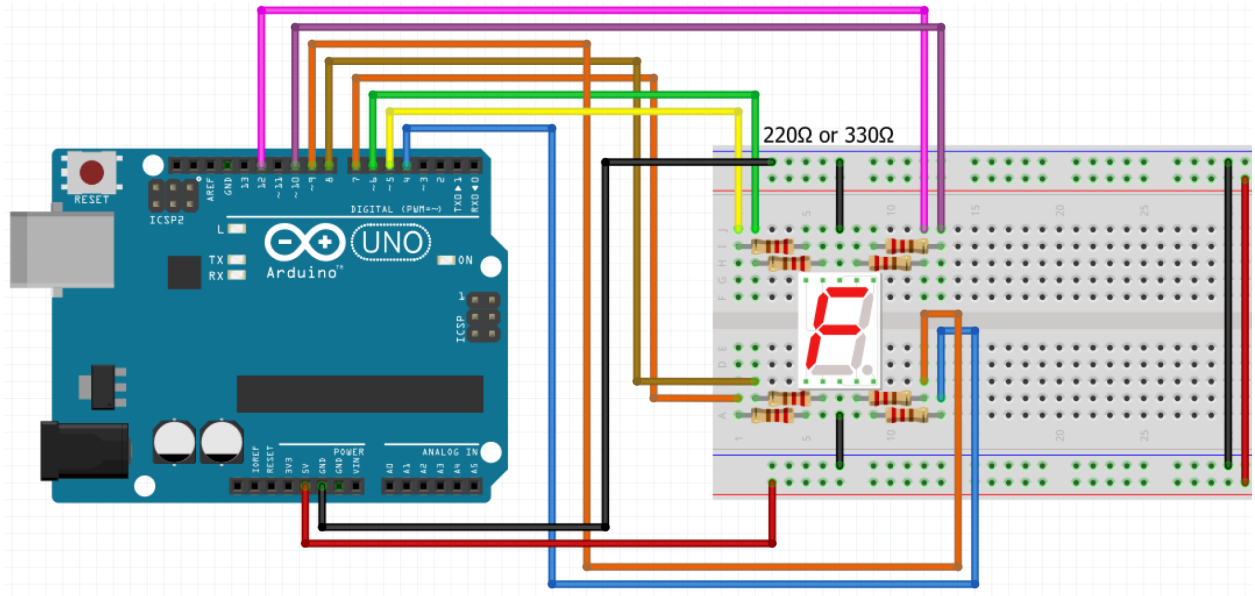
Arduino PIN	Display PIN	Schematic (Common Cathode)
12	7 (A)	
10	6 (B)	
9	4 (C)	
8	2 (D)	
7	1 (E)	
6	9 (F)	
5	10 (G)	
4	5 (DP)	
GND	3, 8 (GND)	

Arduino connections to 7 segment display **common cathode**

In a Common Cathode (CC) 7-segment display, all the cathodes (negative terminals) of the seven LEDs are interconnected and brought out to a common pin, typically marked as 'COM' or 'GND'. This common pin must be connected to the ground (0V). To illuminate a particular segment, a positive voltage should be applied to its respective anode, while the common cathode remains grounded. By selectively applying voltage to specific segments, you can represent numbers and certain letters. The advantage of a CC display is that it's often considered simpler to interface with systems that provide positive logic output.

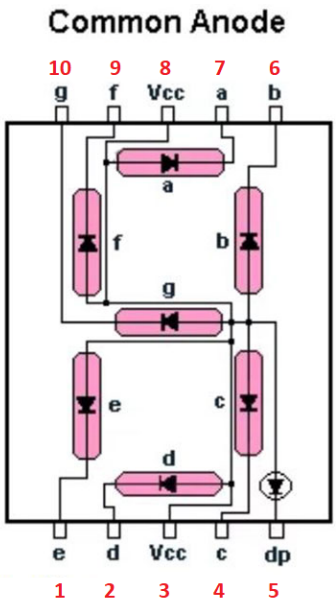


Check out the each LED, it shows its segment on the top right



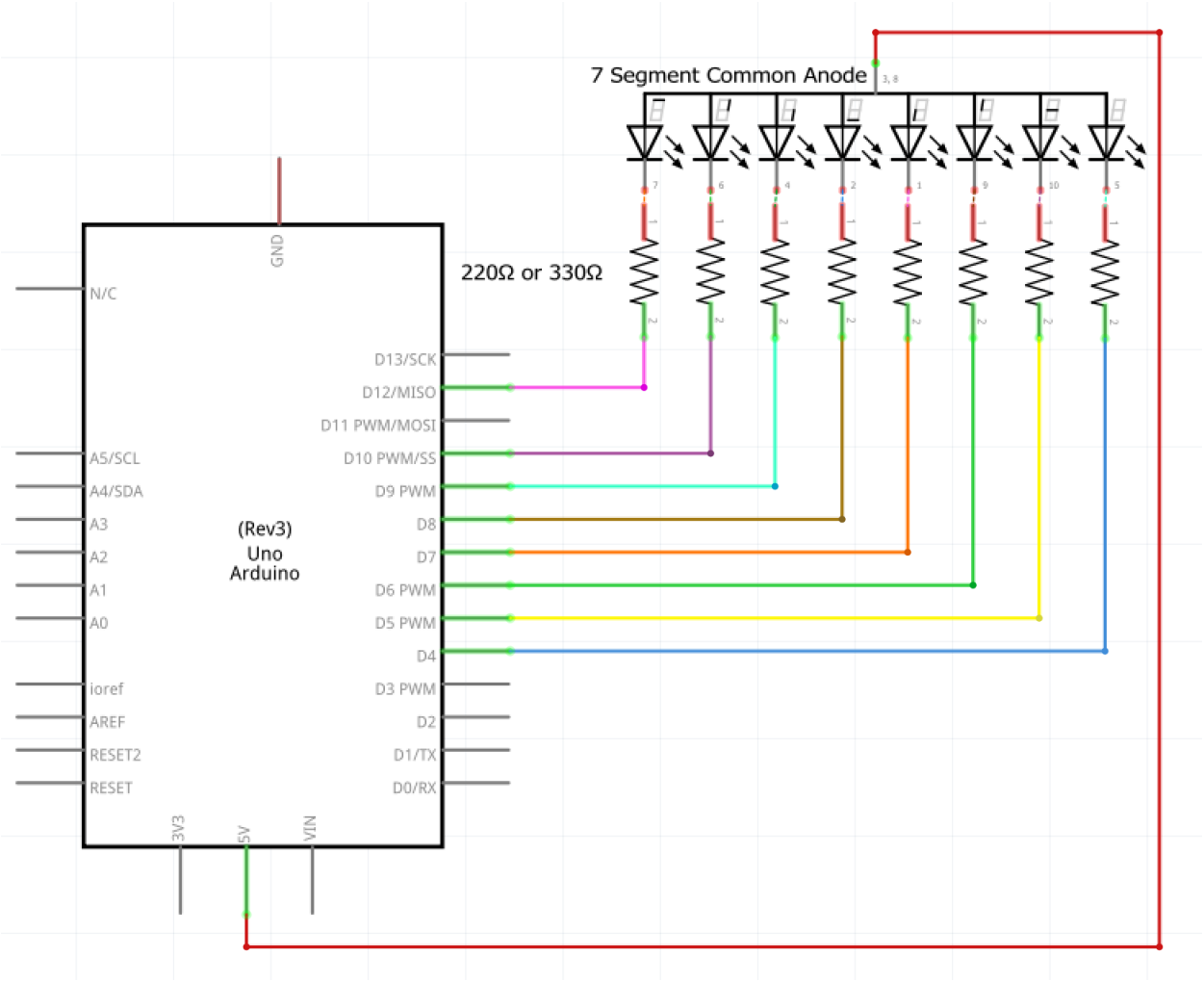
I really don't advise following the breadboard connections. Use the table.

1.6.2 Connecting the 7-segment display - **Common Anode**

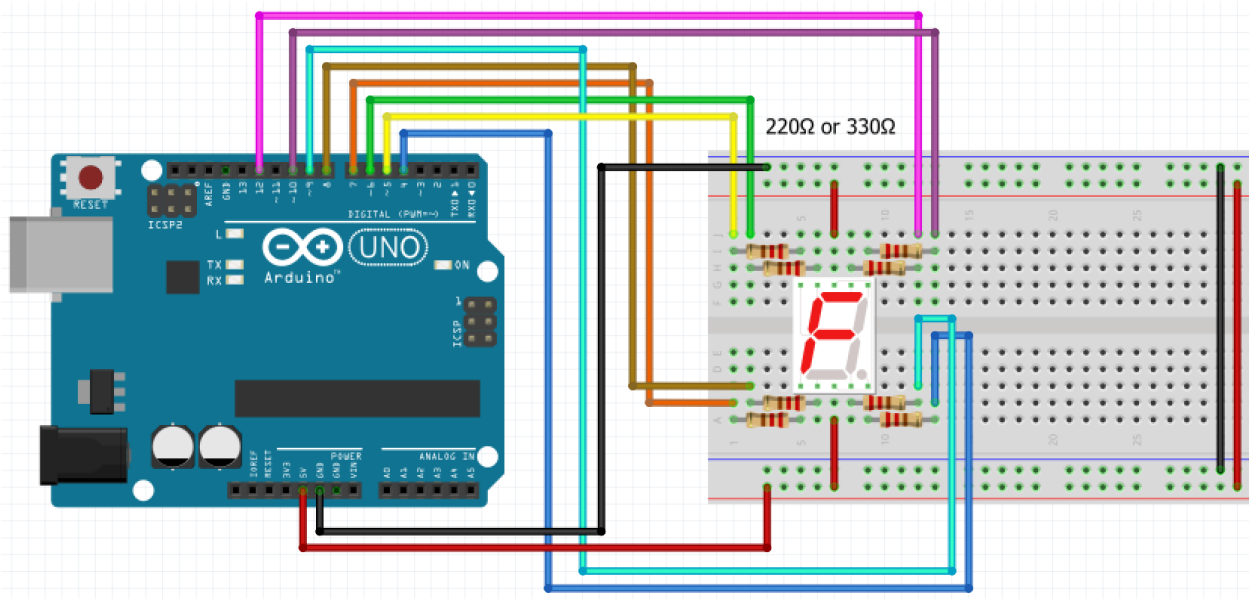
Arduino PIN	Display PIN	Schematic (Common Anode)
12	7 (A)	
10	6 (B)	
9	4 (C)	
8	2 (D)	
7	1 (E)	
6	9 (F)	
5	10 (G)	
4	5 (DP)	
5V	3, 8 (VCC)	

Arduino connections to 7 segment display **common anode**

Contrary to the Common Cathode display, in a Common Anode (CA) 7-segment display, all the anodes (positive terminals) of the seven LEDs are interconnected and routed to a common pin, often marked as 'COM' or 'VCC'. This common pin should be connected to the supply voltage. To light up a particular segment, its respective cathode should be grounded or provided a low logic level. In essence, you're switching on or off the ground connection for each segment. CA displays are particularly suitable for systems with negative logic outputs.



Check out the each LED, it shows its segment on the top right



1.7 Checking each segment

1.7.1 Initial code.

```
// Declare all the segments pins
const int pinA = 12;
const int pinB = 10;
const int pinC = 9;
const int pinD = 8;
const int pinE = 7;
const int pinF = 6;
const int pinG = 5;
const int pinDP = 4;

const int segSize = 8;
int index = 0;

bool commonAnode = false; // Modify if you have common anode
byte state = HIGH;
int segments[segSize] = {
  pinA, pinB, pinC, pinD, pinE, pinF, pinG, pinDP
};

void setup() {
  for (int i = 0; i < segSize; i++) {
    // TODO: Set the pin mode for each segment
  }
  if (commonAnode == true) {
    state = !state;
  }
}

void loop() {
  // TODO: Control the segments to create a pattern

  // TODO: Update the in index to move to the next segment

  // TODO: Set the delay time between segment changes

  // TODO: Reset the index and change state if necessary
}
```

1.7.2 Complete code.

```
// declare all the segments pins
const int pinA = 12;
const int pinB = 10;
const int pinC = 9;
const int pinD = 8;
const int pinE = 7;
const int pinF = 6;
const int pinG = 5;
const int pinDP = 4;

const int segSize = 8;
int index = 0;

bool commonAnode = false; // modify if you have common anode
byte state = HIGH;
int segments[segSize] = {
  pinA, pinB, pinC, pinD, pinE, pinF, pinG, pinDP
};

void setup() {
  for (int i = 0; i < segSize; i++) {
    // TODO: Set the pin mode for each segment
    pinMode(segments[i], OUTPUT);
  }
  if (commonAnode == true) {
    state = !state;
  }
}

void loop() {
  // TODO: Control the segments to create a pattern
  digitalWrite(segments[index], state);
  // TODO: Update the index to move to the next segment
  index++;
  // TODO: Set the delay time between segment changes
  delay(500);
  // TODO: Reset the index and change state if necessary
```

```
  if (index >= segSize) {
    index = 0;
    state = !state;
  }
}
```

Turning all the segments on, and then all the segments off, in order

1.8 Counting from 0 to 9

1.8.1 Initial code.

```
// Declare all the segments pins
const int pinA = 12;
const int pinB = 10;
const int pinC = 9;
const int pinD = 8;
const int pinE = 7;
const int pinF = 6;
const int pinG = 5;
const int pinDP = 4;

const int segSize = 8;
int index = 0;

bool commonAnode = false; // Modify if you have common anode
const int noOfDigits = 10;
byte state = HIGH;
byte dpState = LOW;
int segments[segSize] = {
  pinA, pinB, pinC, pinD, pinE, pinF, pinG, pinDP
};

byte digitMatrix[noOfDigits][segSize - 1] = {
// a  b  c  d  e  f  g
  {1, 1, 1, 1, 1, 1, 0}, // 0
  {0, 1, 1, 0, 0, 0, 0}, // 1
  {1, 1, 0, 1, 1, 0, 1}, // 2
  {1, 1, 1, 1, 0, 0, 1}, // 3
  {0, 1, 1, 0, 0, 1, 1}, // 4
  {1, 0, 1, 1, 0, 1, 1}, // 5
  {1, 0, 1, 1, 1, 1, 1}, // 6
  {1, 1, 1, 0, 0, 0, 0}, // 7
  {1, 1, 1, 1, 1, 1, 1}, // 8
  {1, 1, 1, 1, 0, 1, 1}  // 9
};

void setup() {
  // Initialize all the pins
```

```
  for (int i = 0; i < segSize; i++) {
    pinMode(segments[i], OUTPUT);
  }
  // TODO: Check if commonAnode should be modified here
}

void loop() {
  // TODO: Implement the logic to display numbers on the 7-segment display

  // TODO: Control the timing for switching between numbers
}

void displayNumber(byte digit, byte decimalPoint) {
  // TODO: Write the code to display the given digit on the 7-segment
display
}
```

1.8.2 Complete code.

```
// declare all the segments pins
const int pinA = 12;
const int pinB = 10;
const int pinC = 9;
const int pinD = 8;
const int pinE = 7;
const int pinF = 6;
const int pinG = 5;
const int pinDP = 4;

const int segSize = 8;
int index = 0;

// modify if you have common anode
bool commonAnode = false;

const int noOfDigits = 10;
byte state = HIGH;
byte dpState = LOW;

int segments[segSize] = {
  pinA, pinB, pinC, pinD, pinE, pinF, pinG, pinDP
};


byte digitMatrix[noOfDigits][segSize - 1] = {
// a  b  c  d  e  f  g
  {1, 1, 1, 1, 1, 1, 0}, // 0
  {0, 1, 1, 0, 0, 0, 0}, // 1
  {1, 1, 0, 1, 1, 0, 1}, // 2
  {1, 1, 1, 1, 0, 0, 1}, // 3
  {0, 1, 1, 0, 0, 1, 1}, // 4
  {1, 0, 1, 1, 0, 1, 1}, // 5
  {1, 0, 1, 1, 1, 1, 1}, // 6
  {1, 1, 1, 0, 0, 0, 0}, // 7
  {1, 1, 1, 1, 1, 1, 1}, // 8
  {1, 1, 1, 1, 0, 1, 1}  // 9
};
void setup() {
```

```
  // initialize all the pins
  for (int i = 0; i < segSize; i++) {
    pinMode(segments[i], OUTPUT);
  }
  if (commonAnode == true) {
    state = !state;
  }
}

void loop() {
  // turn all the pins on in order. It is a good exercise to see if you connected the wires properly
  displayNumber(index, dpState);
  index++;
  delay(500);
  if (index == noOfDigits) {
    index = 0;
    dpState = !dpState;
  }
}

void displayNumber(byte digit, byte decimalPoint) {
  for (int i = 0; i < segSize - 1; i++) {
    digitalWrite(segments[i], digitMatrix[digit][i]);
  }
  // write the decimalPoint to DP pin
  digitalWrite(segments[segSize - 1], decimalPoint);
}
```

Displaying, in order, digit 0 to 9 without decimal point and then with decimal point

1.9 Lab Exercise 1: Displaying "HELLO" on a 7-Segment Display

In this exercise, your goal is to modify the previous code in order to create a looping message that displays "hello!" on the 7-segment display. You should print each letter in the word "HELLO" with approximately 1 second between each letter and introduce a pause of about 2 seconds after the word ends before it starts displaying again. During the 2 seconds pause, there should be nothing displayed.

To accomplish this task, you'll need to write an Arduino program that controls the 7-segment display to show each letter in sequence, introducing delays to achieve the desired timing. Pay attention to the specific segments you need to light up to form each letter.

1.9.1 Solution

```
// Declare the number of letters in the message
const int messageLength = 5;

// Declare all the segments pins
const int pinA = 12;
const int pinB = 10;
const int pinC = 9;
const int pinD = 8;
const int pinE = 7;
const int pinF = 6;
const int pinG = 5;
const int pinDP = 4;

const int segSize = 8;
int index = 0;

// Modify if you have common anode
bool commonAnode = false;

const int noOfDigits = 10;
byte state = HIGH;
byte dpState = LOW;

int segments[segSize] = {
  pinA, pinB, pinC, pinD, pinE, pinF, pinG, pinDP
};

// Define custom digits for "hello"
byte helloDigits[messageLength][segSize - 1] = {
  // a  b  c  d  e  f  g
  {0, 1, 1, 0, 1, 1, 1}, // "H"
  {1, 0, 0, 1, 1, 1, 1}, // "E"
  {0, 0, 0, 1, 1, 1, 0}, // "L"
  {0, 0, 0, 1, 1, 1, 0}, // "L"
  {1, 1, 1, 1, 1, 1, 0}  // "O"
};

int helloDelay = 1000; // 1 second between each letter
```

```
int wordDelay = 2000; // 2 seconds after the word ends

void setup() {
  // Initialize all the pins
  for (int i = 0; i < segSize; i++) {
    pinMode(segments[i], OUTPUT);
  }
  if (commonAnode == true) {
    state = !state;
  }
}

void loop() {
  // Display the "hello!" message
  for (int i = 0; i < messageLength; i++) {
    displayCustomDigit(helloDigits[i], dpState);
    delay(helloDelay);
  }

  // Delay before starting the word again
  delay(wordDelay);
}

void displayCustomDigit(byte digit[], byte decimalPoint) {
  for (int i = 0; i < segSize - 1; i++) {
    digitalWrite(segments[i], digit[i]);
  }
  // Write the decimalPoint to DP pin
  digitalWrite(segments[segSize - 1], decimalPoint);
}
```

2. Joystick

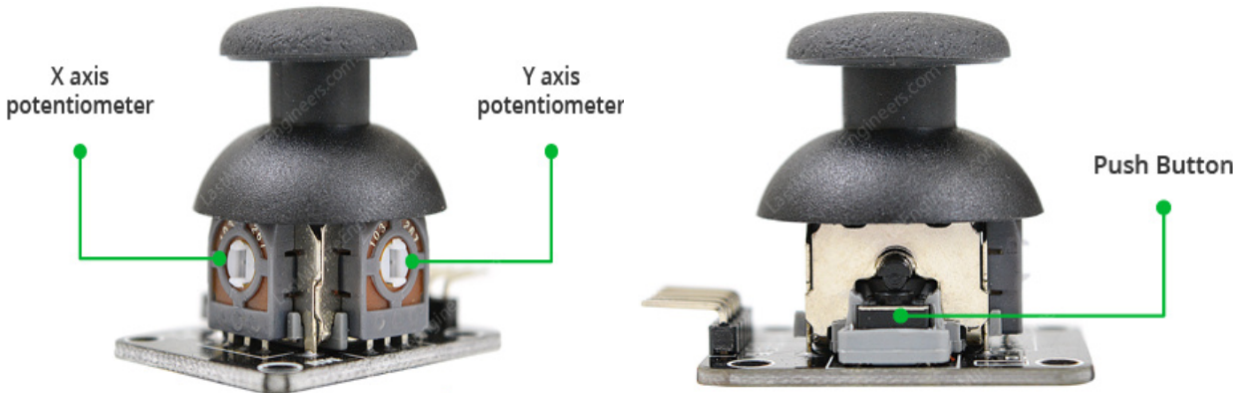
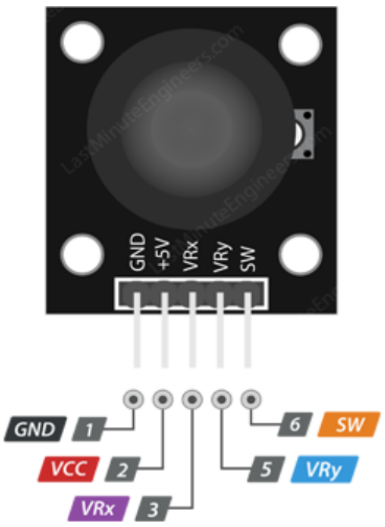


Fig. 2.1 - Joystick axis and button

The joystick is similar to two potentiometers connected together, one for the vertical movement (Y-axis) and the other for the horizontal movement (X-axis). It also contains a switch that activates when you push down on the cap.
What we're gonna use in this lab is a self-centering spring-loaded joystick, meaning when you release the joystick it will center itself. It also contains a comfortable cup-type knob/cap which gives the feel of a thumb-stick.

2.1 Pinout



GND is the Ground Pin
VCC supplies power for the module
VRx gives readout of the joystick in the horizontal direction (X-coordinate) i.e. how far left or right the joystick is pushed.
VRy gives readout of the joystick in the vertical direction (Y-coordinate) i.e. how far up and down the joystick is pushed.
SW is the output from the pushbutton. It's normally open, meaning the digital readout from the SW pin will be HIGH. When the button is pushed, it will connect to GND, giving output LOW.

Fig 2.2 - Joystick pinout configuration

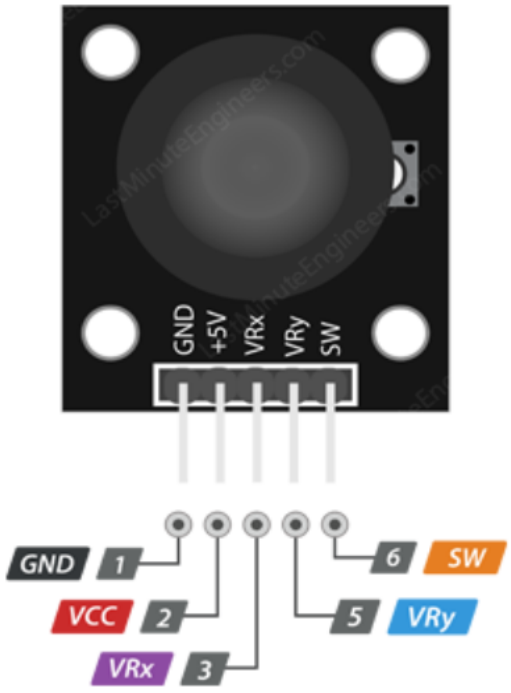
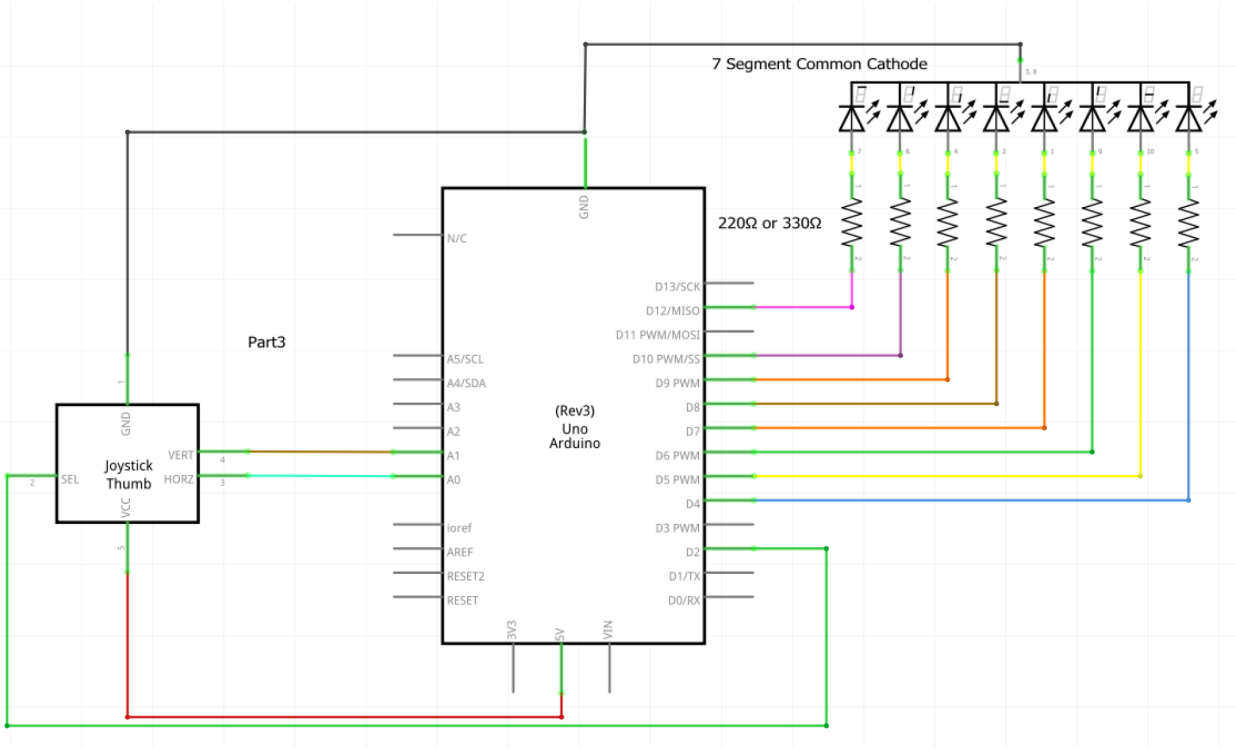
Arduino PIN	Joystick PIN	Schematic
GND	1 (GND)	
5V	2 (5V)	
A0	3 (VRx)	
A1	4 (VRy)	
2	5 (SW)	

Table 2.1 - Arduino connections to joystick

2.2 Reading values from Joystick

In order to read the joystick's physical position, we need to measure the change in resistance of a potentiometer.

We'll use the joystick with the small breadboard so that we can add the other elements on the medium breadboard without removing the joystick. Make sure you connect your joystick in such a way that you can use it (don't put the wires in front of it).



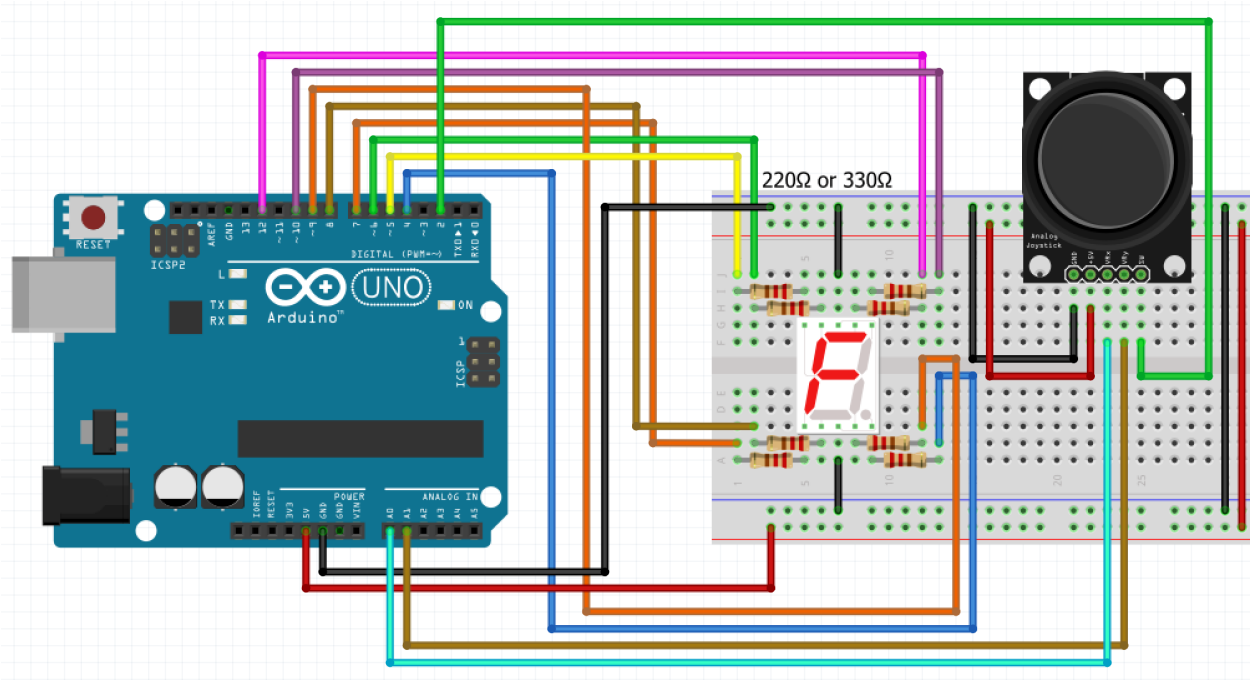


Fig. 2.3 - Joystick 7 segment display (common cathode) connections

```
// declare all the pins
const int pinSW = 2; // digital pin connected to switch output
const int pinX = A0; // A0 - analog pin connected to X output
const int pinY = A1; // A1 - analog pin connected to Y output
byte swState = LOW;
int xValue = 0;
int yValue = 0;
void setup() {
  pinMode(pinSW, INPUT_PULLUP); // activate pull-up resistor on the                // push-button pin
  // Start the serial communication.
  Serial.begin(9600);
}

void loop() {
  swState = digitalRead(pinSW);
  xValue = analogRead(pinX);
  yValue = analogRead(pinY);

  Serial.print("Switch:  ");
  Serial.print(swState);
  Serial.print("  |  ");
  Serial.print("X-axis: ");
  Serial.print(xValue);
```

```
  Serial.print("  |  ");
  Serial.print("Y-axis: ");
  Serial.print(yValue);
  Serial.println("  |  ");
  delay(200);
}
```

Code snippet 2.1 - Joystick interaction values

Based on the values displayed on the serial monitor, try and decipher which direction is X and which direction is Y. If needed, change the delay value to a bigger one.

3. Joystick & 7-segment display

Now we know how to control a joystick and a 7-segment display individually. Let's use those components in the same system as follows:

- 7-segment display: used for displaying numbers given as input though the variable named **digit**
- Joystick: used for changing **digit** variable

Details:

- Digit value increases by 1 if the joystick thumbstick is moved on the right side of the Ox axis (values bigger than 512) or decreases by 1 unit if the joystick thumbstick is moved on the left side of the Ox axis (values lower than 512)
- It should only move once with each iteration meaning that it doesn't continuously decrease if you keep the joystick moved to the right; you have to bring it to the middle and move it again if you want to decrease it further
- The joystick should use the joystick's button to toggle the decimal point on and off

3.1 Lab exercise 2 (10 min).

Write code to cycle through numbers (increment and decrement) using the joystick. Display these on the 7 segment display. Implement a "state" mechanism for the joystick, allowing it to change the number once per movement and requiring it to return to its default position before changing its state again.

3.1.1 Initial code.

```
// Declare all the joystick pins
const int pinSW = 2; // Digital pin connected to switch output
const int pinX = A0; // A0 - Analog pin connected to X output
const int pinY = A1; // A1 - Analog pin connected to Y output

// Declare all the segments pins
const int pinA = 12;
const int pinB = 10;
const int pinC = 9;
const int pinD = 8;
const int pinE = 7;
const int pinF = 6;
const int pinG = 5;
const int pinDP = 4;

const int segSize = 8;
int index = 0;

bool commonAnode = false; // Modify if you have common anode
const int noOfDigits = 10;
byte state = HIGH;
byte dpState = LOW;
byte swState = LOW;
byte lastSwState = LOW;
int xValue = 0;
int yValue = 0;

bool joyMoved = false;
int digit = 0;
int minThreshold = 400;
int maxThreshold = 600;

int segments[segSize] = {
```

```
  pinA, pinB, pinC, pinD, pinE, pinF, pinG, pinDP
};

byte digitMatrix[noOfDigits][segSize - 1] = {
// a  b  c  d  e  f  g
  {1, 1, 1, 1, 1, 1, 0}, // 0
  {0, 1, 1, 0, 0, 0, 0}, // 1
  {1, 1, 0, 1, 1, 0, 1}, // 2
  {1, 1, 1, 1, 0, 0, 1}, // 3
  {0, 1, 1, 0, 0, 1, 1}, // 4
  {1, 0, 1, 1, 0, 1, 1}, // 5
  {1, 0, 1, 1, 1, 1, 1}, // 6
  {1, 1, 1, 0, 0, 0, 0}, // 7
  {1, 1, 1, 1, 1, 1, 1}, // 8
  {1, 1, 1, 1, 0, 1, 1}  // 9
};

void setup() {
  // TODO: Initialize joystick pins and commonAnode

  // Initialize all the pins
  for (int i = 0; i < segSize; i++) {
    pinMode(segments[i], OUTPUT);
  }
  // TODO: Check if commonAnode should be modified here
}

void loop() {
  // TODO: Implement the logic for joystick input and number display

  // TODO: Control the timing for switching between numbers
}

void displayNumber(byte digit, byte decimalPoint) {
  // TODO: Write the current number
}
```

3.2.2 Solution.

```
// declare all the joystick pins
const int pinSW = 2; // digital pin connected to switch output
const int pinX = A0; // A0 - analog pin connected to X output
const int pinY = A1; // A1 - analog pin connected to Y output

// declare all the segments pins
const int pinA = 12;
const int pinB = 10;
const int pinC = 9;
const int pinD = 8;
const int pinE = 7;
const int pinF = 6;
const int pinG = 5;
const int pinDP = 4;

const int segSize = 8;
int index = 0;

// modify if you have common anode
bool commonAnode = false;
const int noOfDigits = 10;
byte state = HIGH;
byte dpState = LOW;
byte swState = LOW;
byte lastSwState = LOW;
int xValue = 0;
int yValue = 0;

bool joyMoved = false;
int digit = 0;
int minThreshold = 400;
int maxThreshold = 600;

int segments[segSize] = {
  pinA, pinB, pinC, pinD, pinE, pinF, pinG, pinDP
};

byte digitMatrix[noOfDigits][segSize - 1] = {
// a  b  c  d  e  f  g
```

```
  {1, 1, 1, 1, 1, 1, 0}, // 0
  {0, 1, 1, 0, 0, 0, 0}, // 1
  {1, 1, 0, 1, 1, 0, 1}, // 2
  {1, 1, 1, 1, 0, 0, 1}, // 3
  {0, 1, 1, 0, 0, 1, 1}, // 4
  {1, 0, 1, 1, 0, 1, 1}, // 5
  {1, 0, 1, 1, 1, 1, 1}, // 6
  {1, 1, 1, 0, 0, 0, 0}, // 7
  {1, 1, 1, 1, 1, 1, 1}, // 8
  {1, 1, 1, 1, 0, 1, 1}  // 9
};

void setup() {
  // initialize all the pins
  for (int i = 0; i < segSize; i++) {
    pinMode(segments[i], OUTPUT);
  }
  pinMode(pinSW, INPUT_PULLUP);
  if (commonAnode == true) {
    state = !state;
  }
}

void loop() {
  // turn all the pins on in order. It is a good exercise to see if you connected the wires properly
  xValue = analogRead(pinX);

  if (xValue < minThreshold && joyMoved == false) {
    if (digit > 0) {
      digit--;
    }
    else {
      digit = 9;
    }
    joyMoved = true;
  }
  if (xValue > maxThreshold && joyMoved == false) {
    if (digit < 9) {
      digit++;
    }
    else {
      digit = 0;
```

```
    }
    joyMoved = true;
  }
  if (xValue >= minThreshold && xValue <= maxThreshold) {
    joyMoved = false;
  }

  swState = digitalRead(pinSW);
  if (swState != lastSwState) {
    if (swState == LOW) {
      dpState = !dpState;
    }
  }
  lastSwState = swState;
  displayNumber(digit, dpState);
  delay(1);
}

void displayNumber(byte digit, byte decimalPoint) {
  for (int i = 0; i < segSize - 1; i++) {
    digitalWrite(segments[i], digitMatrix[digit][i]);
  }
  // write the decimalPoint to DP pin
  digitalWrite(segments[segSize - 1], decimalPoint);
}
```

4. Exercises

4.1 Exercise 1: Dice Simulator

Objective: Create a virtual dice rolling simulator using the 7-segment display and a pushbutton. When the button is pressed, the display should show a random number between 1 and 6, just like rolling a physical die.

Instructions:

1. Write a program that generates a random number between 1 and 6 when the button is pressed.
2. Display the random number on the 7-segment display.
3. Test your program by pressing the button to simulate rolling a dice.

4.2 Exercise 2: LED Animation

Objective: Create a simple animation using the 7-segment display. The animation could be a bouncing ball, a scrolling pattern, or any other creative idea you have in mind.

Instructions:

1. Plan and design your LED animation. Decide on the pattern or movement you want to display. A bouncing ball, a scrolling text etc.
2. Write a program that controls the 7-segment display to create your chosen animation.
3. Upload your program to the Arduino and observe the animation on the 7-segment display.