Introduction to robotics

7th 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 USE 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.

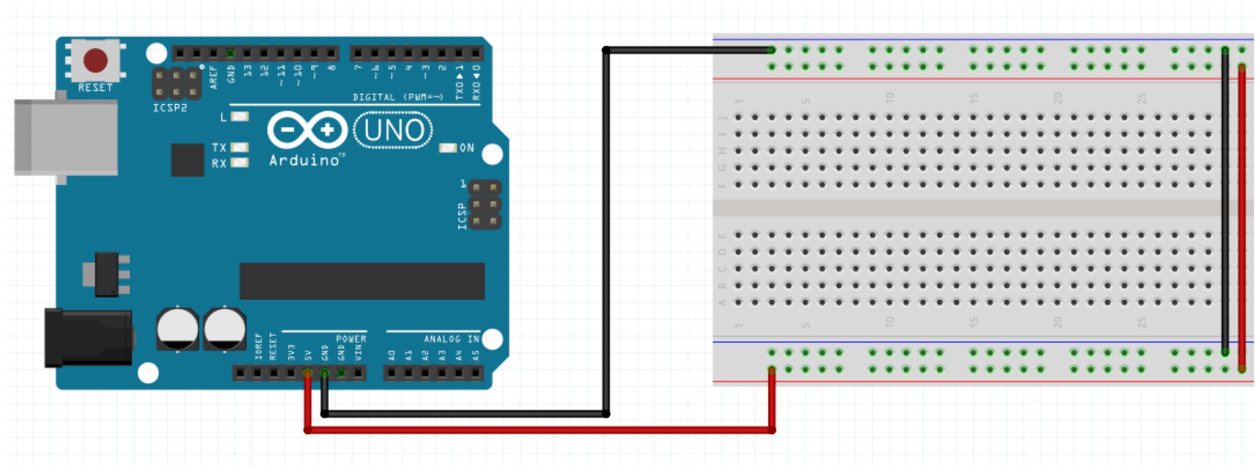


Fig. 0.1 - Default setup

0. Homework check

1. MAX7219 Driver

Datasheet: https://www.sparkfun.com/datasheets/Components/General/COM-09622-MAX7219-MAX7221.pdf

The MAX7219/MAX7221 are compact, serial input/out-put common-cathode display drivers that interface microprocessors (µPs) to 7-segment numeric LED displays of up to 8 digits, bar-graph displays, or 64 individual LEDs.

(source: Datasheet)

Basically, it's a simple and somewhat inexpensive method of controlling 64 LEDs in either matrix or numeric display form. Furthermore they can be chained together to control two or more units for even more LEDs. Overall – they're a lot of fun and can also be quite useful.

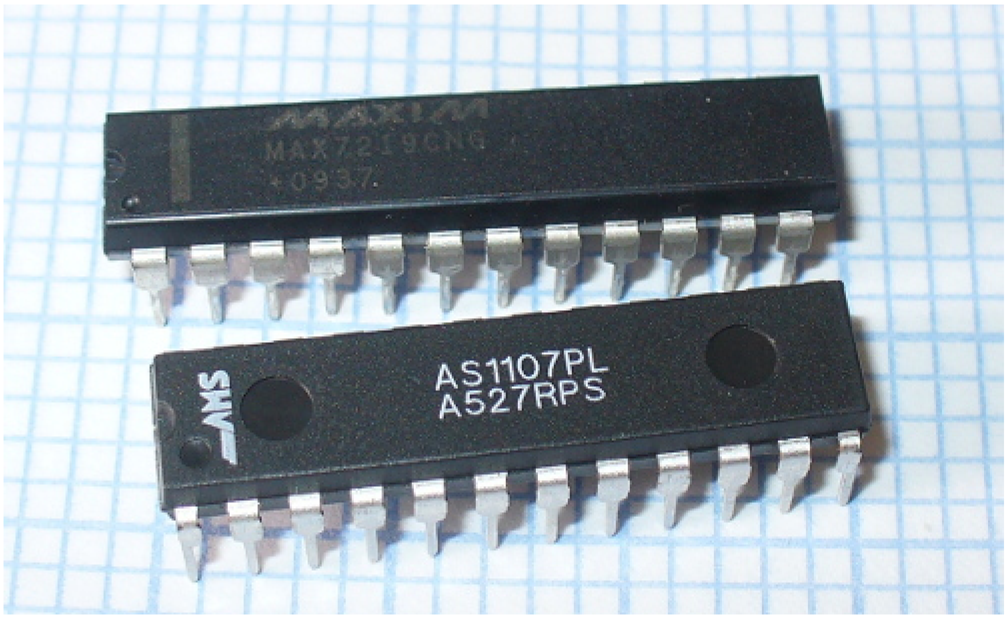


Fig. 1.1 MAX7219 LED Driver

As mentioned earlier, the MAX7219 can completely control 64 individual LEDs – including maintaining equal brightness, and allowing you to adjust the brightness of the LEDs either with hardware or software (or both). It can refresh the LEDs at around 800 Hz (min 500Hz - max 1300Hz), so no more flickering, uneven LED displays.

2. LED matrix display with MAX7219 driver

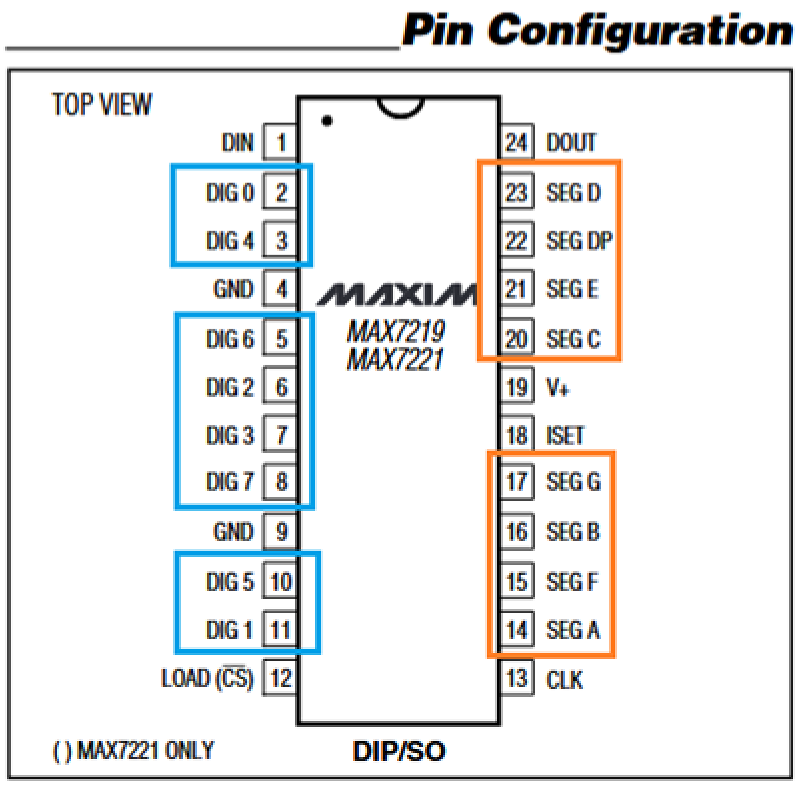
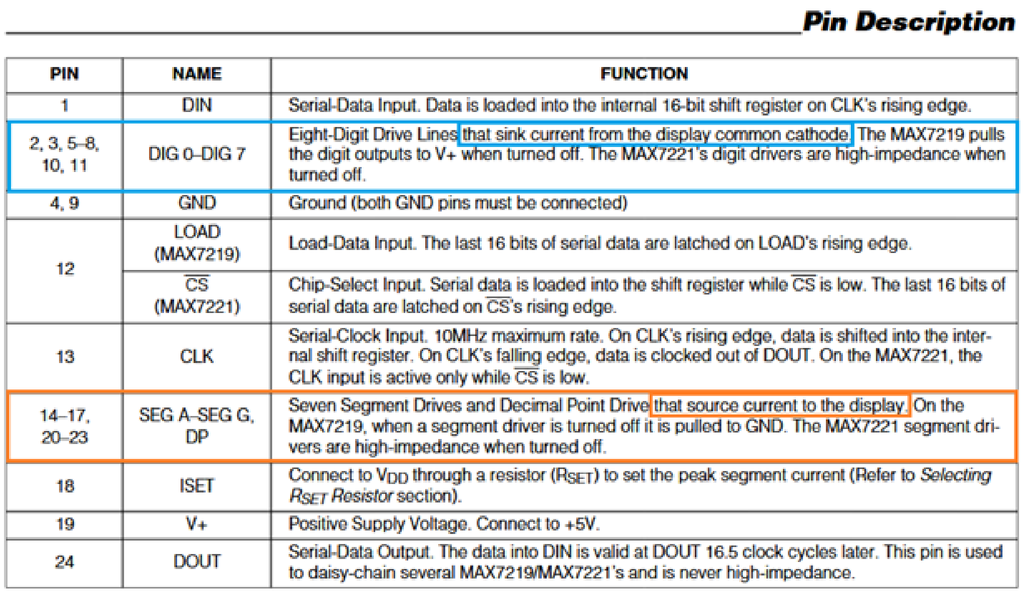


Fig. 2.1 MAX7219 Pin Configuration



Pin Description

PIN	NAME	FUNCTION
1	DIN	Serial-Data Input. Data is loaded into the internal 16-bit shift register on CLK's rising edge.
2, 3, 5–8, 10, 11	DIG 0–DIG 7	Eight-Digit Drive Lines that sink current from the display common cathode. The MAX7219 pulls the digit outputs to V+ when turned off. The MAX7221's digit drivers are high-impedance when turned off.
4, 9	GND	Ground (both GND pins must be connected)
12	LOAD (MAX7219)	Load-Data Input. The last 16 bits of serial data are latched on LOAD's rising edge.
	$\overline{CS}$ (MAX7221)	Chip-Select Input. Serial data is loaded into the shift register while $\overline{CS}$ is low. The last 16 bits of serial data are latched on $\overline{CS}$'s rising edge.
13	CLK	Serial-Clock Input. 10MHz maximum rate. On CLK's rising edge, data is shifted into the internal shift register. On CLK's falling edge, data is clocked out of DOUT. On the MAX7221, the CLK input is active only while $\overline{CS}$ is low.
14–17, 20–23	SEG A–SEG G, DP	Seven Segment Drives and Decimal Point Drive that source current to the display. On the MAX7219, when a segment driver is turned off it is pulled to GND. The MAX7221 segment drivers are high-impedance when turned off.
18	ISET	Connect to V_{DD} through a resistor (R_{SET}) to set the peak segment current (Refer to *Selecting R_{SET} Resistor* section).
19	V+	Positive Supply Voltage. Connect to +5V.
24	DOUT	Serial-Data Output. The data into DIN is valid at DOUT 16.5 clock cycles later. This pin is used to daisy-chain several MAX7219/MAX7221's and is never high-impedance.

Fig. 2.2 MAX7219 Pin Description

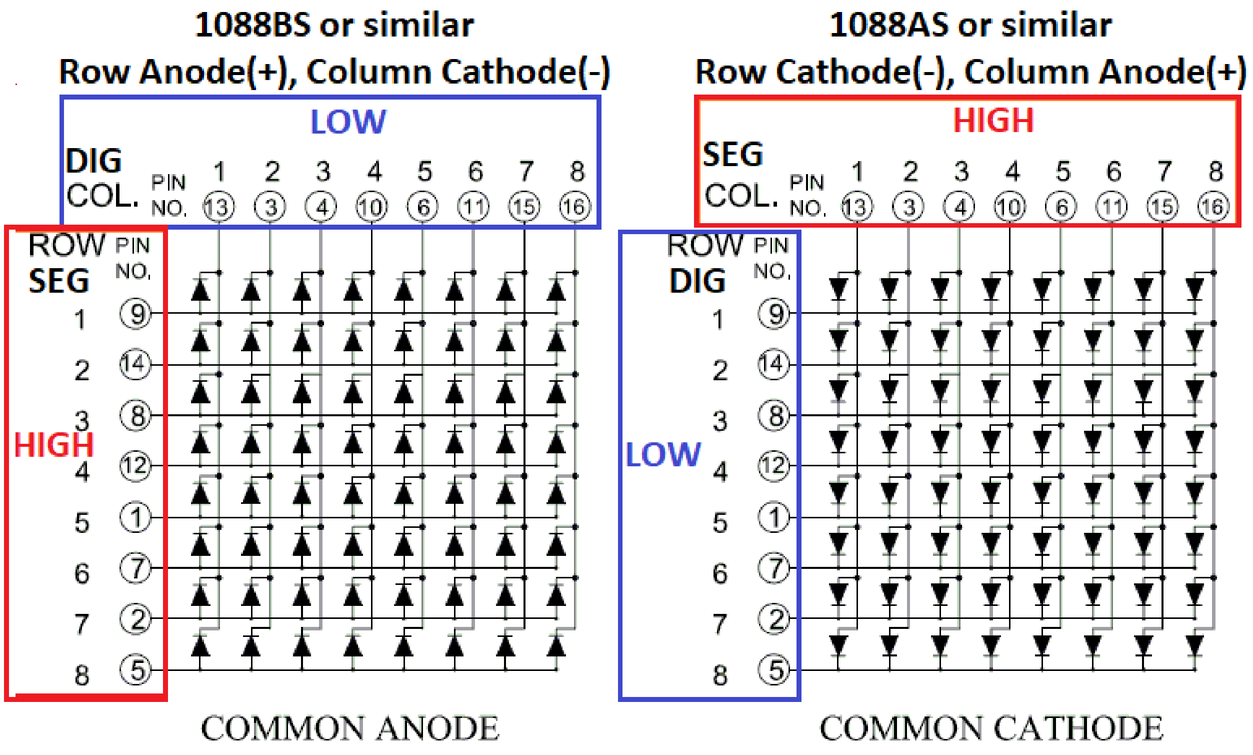
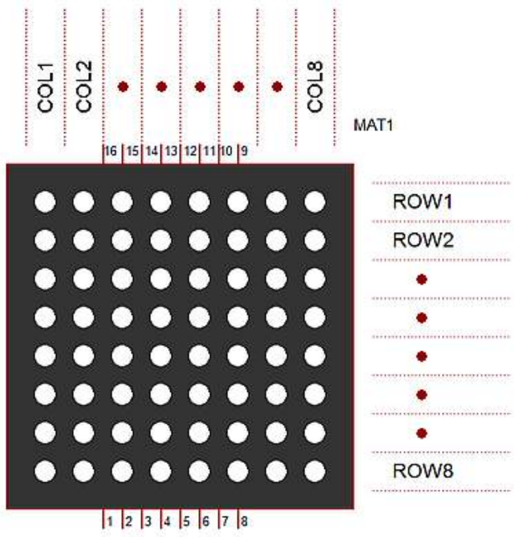


Fig. 2.3 8x8 LED Matrix configuration (common anode vs common cathode)

From the description it is clear that:

1. **DIG** pins are used for the cathodes of the LED matrix
2. **SEG** pins are used for the anodes of the LED matrix
3. Careful, first check which type of matrix you have



!important Check with a multimeter to see what matrix type you have. DO NOT RELY ON THE WRITTEN NUMBER.

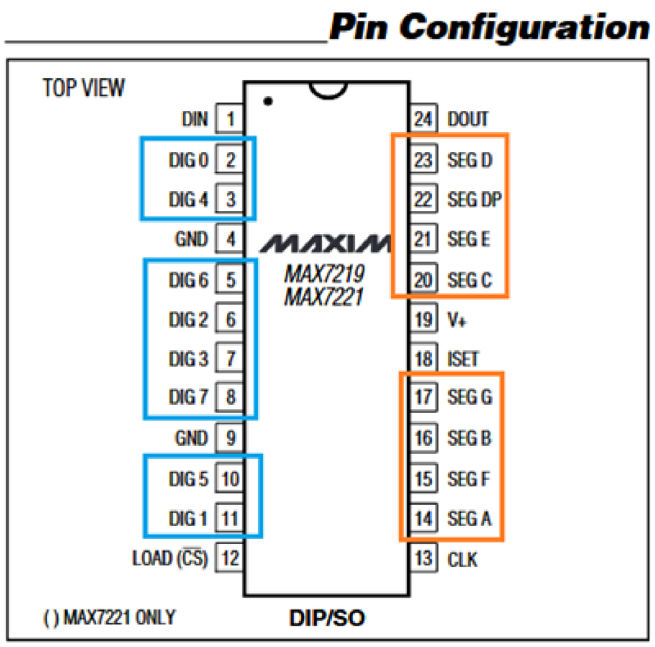
Fig 2.3 Matrix structure & Pin Numbering

3. Matrix to Driver Connections Table:

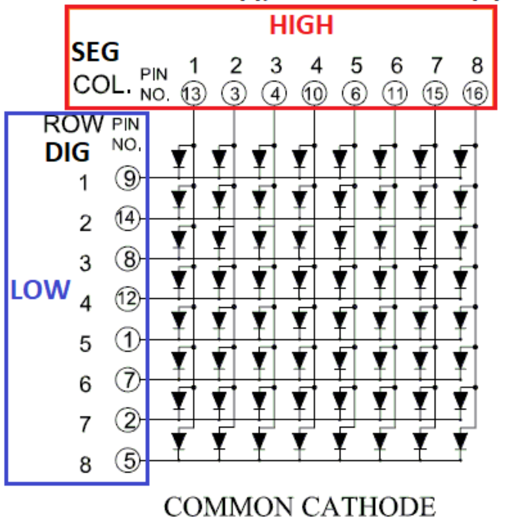
3.1 1088AS or similar

(Row Cathode-, Column Anode+)

1088AS Matrix (Row Cathode-, Column Anode+)		MAX7219 Driver	
Row / Column	Matrix Pin	DIG / SEG Number	Pin Number
Row 5	1	DIG4	3
Row 7	2	DIG6	5
Col 2	3	SEG A	14
Col 3	4	SEG B	16
Row 8	5	DIG7	8
Col 5	6	SEG D	23
Row 6	7	DIG5	10
Row 3	8	DIG2	6
Row 1	9	DIG0	2
Col 4	10	SEG C	20
Col 6	11	SEG E	21
Row 4	12	DIG3	7
Col 1	13	SEG DP	22
Row 2	14	DIG1	11
Col 7	15	SEG F	15
Col 8	16	SEG G	17



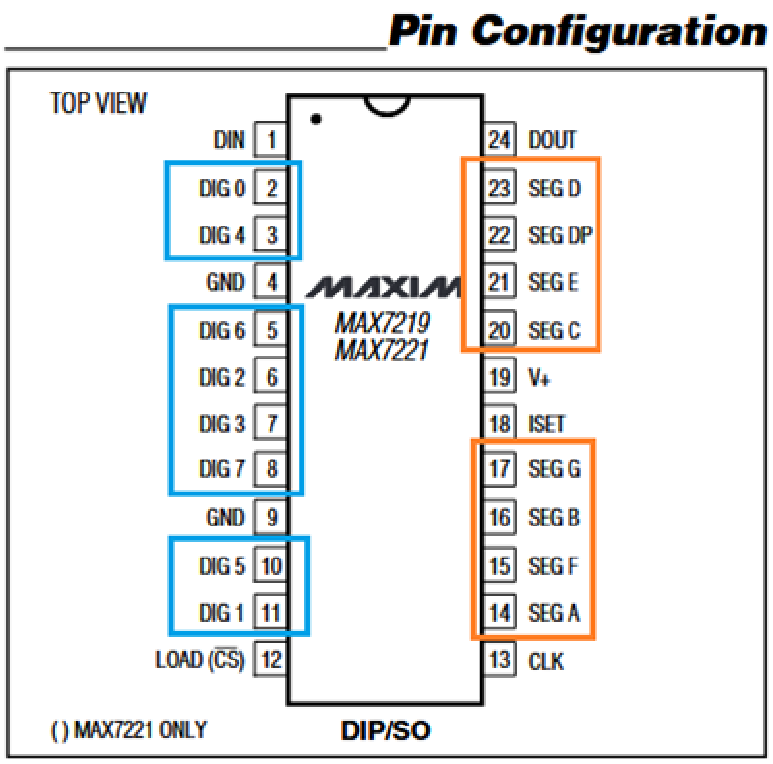
1088AS or similar
Row Cathode(-), Column Anode(+)

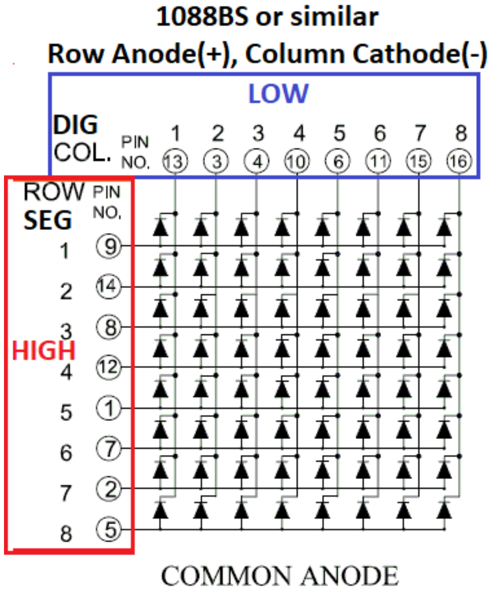


3.2 1088BS or similar

(Row Anode+, Column Cathode-)

1088BS Matrix (Row Anode+, Column Cathode-)		MAX7219 Driver	
Row / Column	Matrix Pin	DIG Number	Pin Number
Col 5	1	SEG D	23
Col 7	2	SEG F	15
Row 2	3	DIG1	11
Row 3	4	DIG2	6
Col 8	5	SEG G	17
Row 5	6	DIG4	3
Col 6	7	SEG E	21
Col 3	8	SEG B	16
Col 1	9	SEG DP	22
Row 4	10	DIG3	7
Row 6	11	DIG5	10
Col 4	12	SEG C	20
Row 1	13	DIG0	2
Col 2	14	SEG A	14
Row 7	15	DIG6	5
Row 8	16	DIG7	8





4. Connecting the driver to Arduino

Note: if the matrix doesn't work or turns off when half the leds are on, try a 100k resistor instead of 10k.

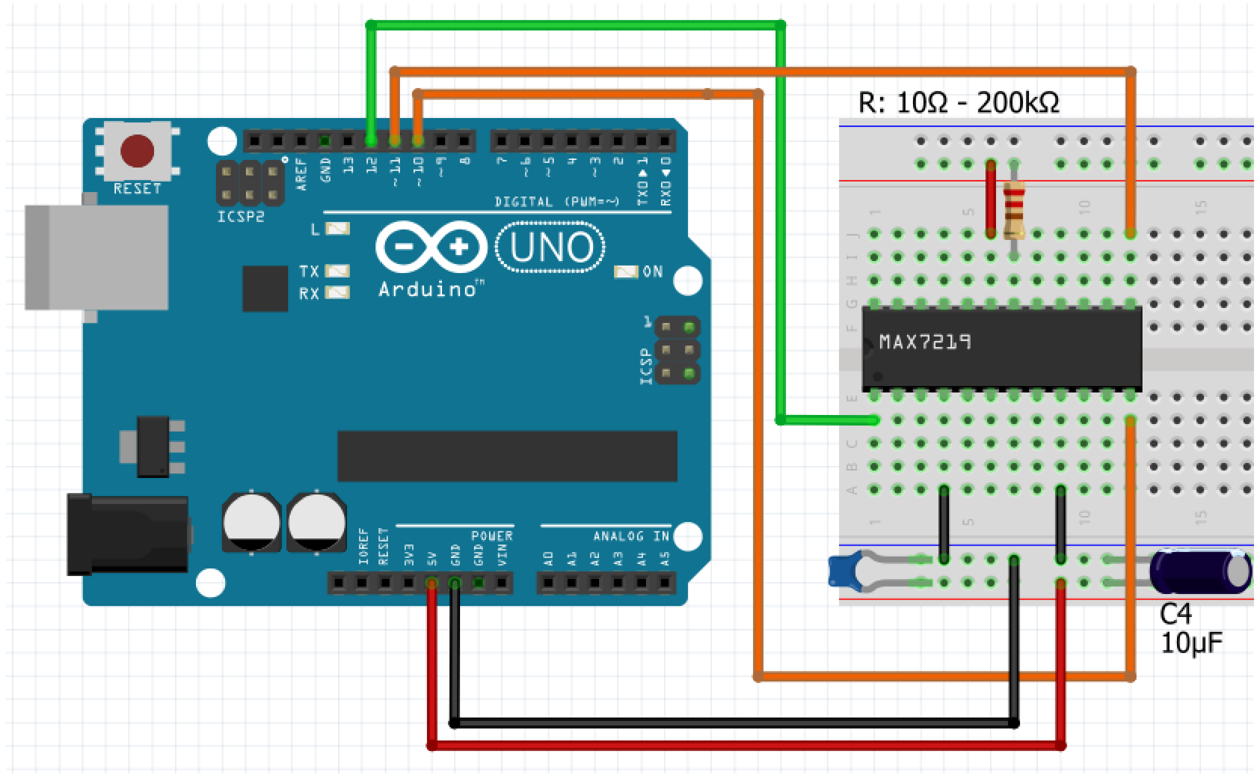


Fig 4.1 MAX7219 Arduino Connections (breadboard)

To minimize power-supply ripple due to the peak digit driver currents, connect a 10µF electrolytic and a 0.1µF ceramic capacitor between V+ and GND as close to the device as possible. The MAX7219/MAX7221 should be placed in close proximity to the LED display, and connections should be kept as short as possible to minimize the effects of wiring inductance and electromagnetic interference. Also, both GND pins must be connected to ground.

Source: Datasheet

!important If connected wrong, the electrolytic capacitor can blow up (remember the first lab, that's what we blew up).

Max7219 Driver Pins	Arduino Pins
4 (GND)	GND
9 (GND)	GND
18 (ISET)	5V, **through a 10k or 100k+ resistor**
19 (V+)	5V
1 (DIN)	12
12 (LOAD/CS)	10
13 (CLK)	11

As you can see in the schematic, there are also 2 capacitors that are connected in parallel to the + and - of our circuit. **If you change the pinout order, remember to change it in the code as well.**

- 1 electrolytic capacitor of 10 µF
- 1 ceramic capacitor of 104 pF

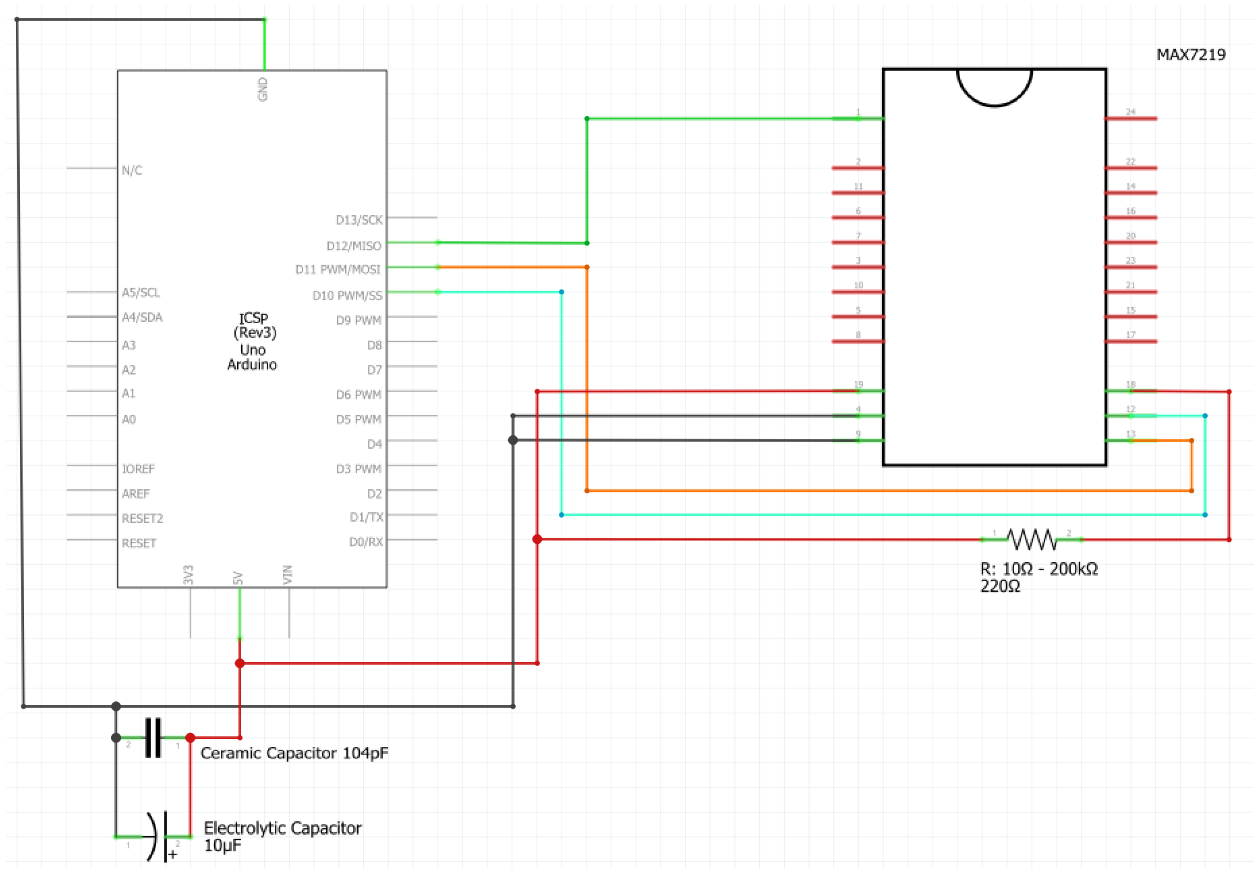
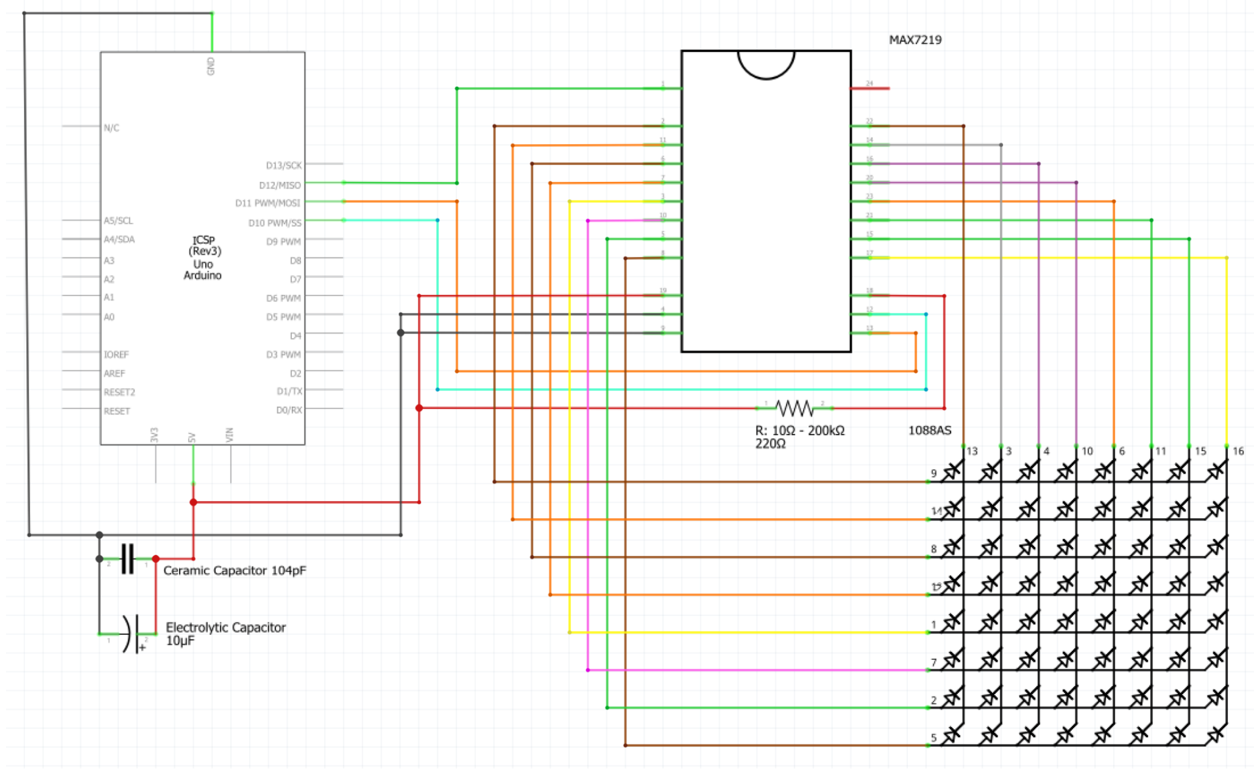


Fig 4.2 MAX7219 Arduino Connections (schematic)

The schematic is or 1088AS, pay attention to what type of matrix you have



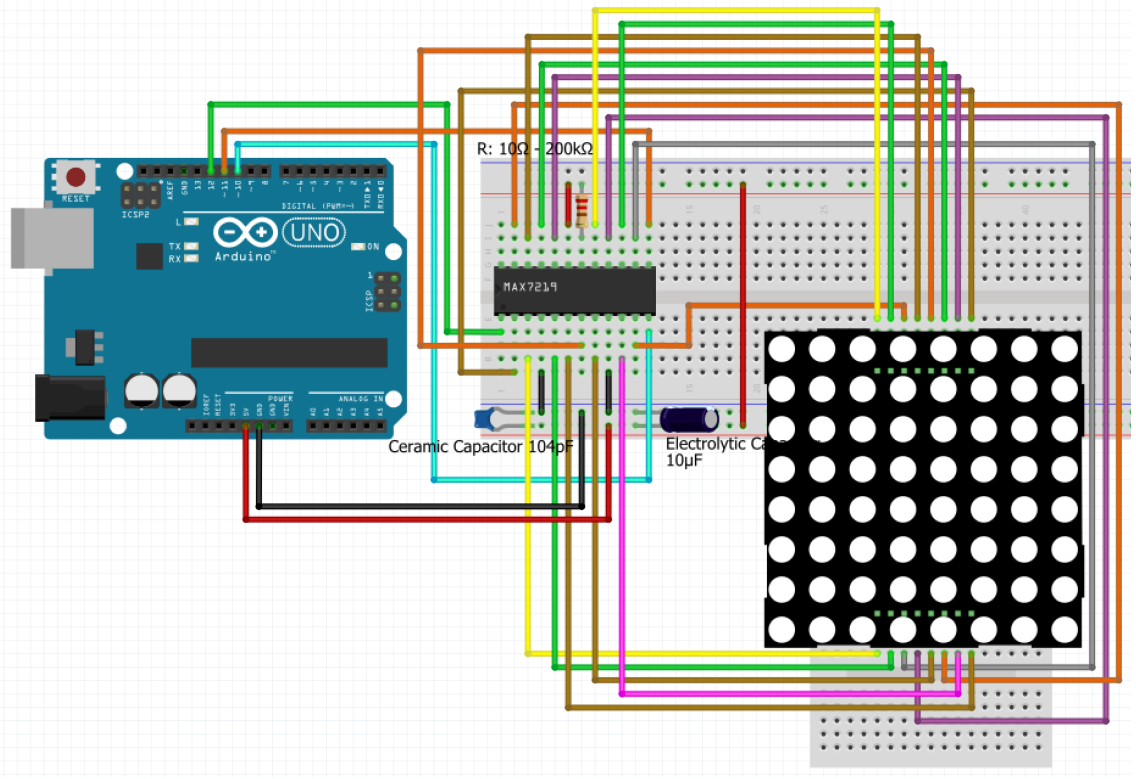
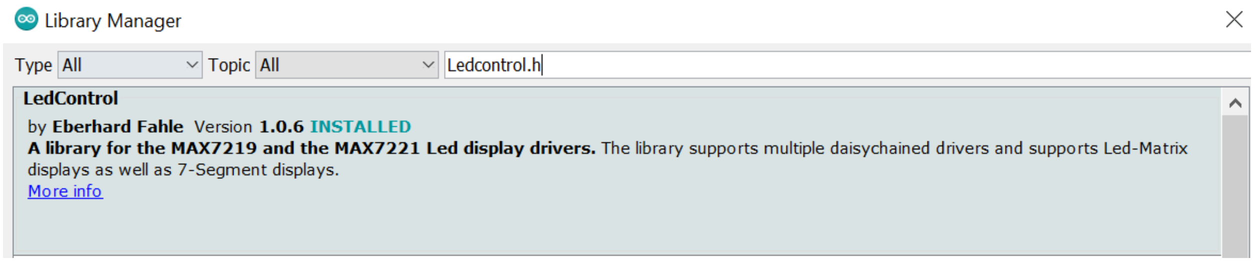


Fig 4.3 1088 AS Matrix Connections, schematic and breadboard

5. Code Examples

First of all, you need to install the LedControl.h library. Go to Tools -> Manage Libraries and search for Ledcontrol.h


Library Manager

Type All Topic All Ledcontrol.h

LedControl

by **Eberhard Fahle** Version **1.0.6** INSTALLED

A library for the MAX7219 and the MAX7221 Led display drivers. The library supports multiple daisychained drivers and supports Led-Matrix displays as well as 7-Segment displays.

More info

5.1 Turning the each led on and off

We will use the **LedControl library** in our code.

```
#include "LedControl.h" //  need the library
const byte dinPin = 12; // pin 12 is connected to the MAX7219 pin 1
const byte clockPin = 11; // pin 11 is connected to the CLK pin 13
const byte loadPin = 10; // pin 10 is connected to LOAD pin 12
const byte matrixSize = 8; // 1 as we are only using 1 MAX7219
LedControl lc = LedControl(dinPin, clockPin, loadPin, 1); //DIN, CLK, LOAD, No. DRIVER
byte matrixBrightness = 2;

void setup() {
 // the zero refers to the MAX7219 number, it is zero for 1 chip
 lc.shutdown(0, false); // turn off power saving, enables display
 lc.setIntensity(0, matrixBrightness); // sets brightness (0~15 possible values)
 lc.clearDisplay(0);// clear screen
}

void loop() {
 for (int row = 0; row < matrixSize; row++) {
  for (int col = 0; col < matrixSize; col++) {
   lc.setLed(0, row, col, true); // turns on LED at col, row
   delay(25);
  }
 }
 for (int row = 0; row < matrixSize; row++) {
  for (int col = 0; col < matrixSize; col++) {
   lc.setLed(0, row, col, false); // turns off LED at col, row
   delay(25);
  }
 }
}
```

5.2 Matrix representation

There are 2 ways to represent the matrix.

5.2.1 8x8 byte matrix

8*8 array which you can cycle through, setting the value on each LED

```
byte matrix[matrixSize][matrixSize] = {
  {1, 0, 0, 0, 0, 0, 0, 1},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1}
};

for (int row = 0; row < matrixSize; row++) {
  for (int col = 0; col < matrixSize; col++) {
    lc.setLed(0, row, col, matrix[row][col]);
  }
}
```

5.2.2 8-byte array

8-byte array of bytes, which you can cycle through, setting the value on each ROW

```
const byte matrixByte[matrixSize] = {
  B10000001,
  B00000000,
  B00000000,
  B00000000,
  B00000000,
  B00000000,
  B00000000,
  B10000001
};
for (int row = 0; row < matrixSize; row++) {
  lc.setRow(0, row, matrixByte[row]);
}
```

5.2.3 Both representations in practice

```
#include "LedControl.h" //  need the library
const byte dinPin = 12;
const byte clockPin = 11;
const byte loadPin = 10;
const byte matrixSize = 8;
// pin 12 is connected to the MAX7219 pin 1
// pin 11 is connected to the CLK pin 13
// pin 10 is connected to LOAD pin 12
// 1 as we are only using 1 MAX7219
LedControl lc = LedControl(dinPin, clockPin, loadPin, 1); //DIN, CLK, LOAD, No. DRIVER

byte matrixBrightness = 2;

byte matrix[matrixSize][matrixSize] = {
  {1, 0, 0, 0, 0, 0, 0, 1},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {1, 0, 0, 0, 0, 0, 0, 1}
};

byte matrixByte[matrixSize] =
{
  B10000001,
  B00000000,
  B00000000,
  B00000000,
  B00000000,
  B00000000,
  B00000000,
  B10000001
};
```

```
void setup() {
  // the zero refers to the MAX7219 number, it is zero for 1 chip
  lc.shutdown(0, false); // turn off power saving, enables display
  lc.setIntensity(0, matrixBrightness); // sets brightness (0~15 possible values)
  lc.clearDisplay(0);// clear screen
}


void loop() {
  for (int row = 0; row < matrixSize; row++) {
    for (int col = 0; col < matrixSize; col++) {
      lc.setLed(0, row, col, matrix[row][col]);
    }
  }
//  for (int row = 0; row < matrixSize; row++)
//  {
//    lc.setRow(0, row, matrixByte[row]);
//  }
}
```

5.3 Some animations

(BADLY WRITTEN CODE, DON'T IMITATE IT)
Forgot the source, somewhere on the internet. If you find it, please linkit.

```
#include "LedControl.h" //  need the library
const int dinPin = 12;
const int clockPin = 11;
const int loadPin = 10;
const int rows = 8;
const int cols = 8;
LedControl lc = LedControl(dinPin, clockPin, loadPin, 1); //DIN, CLK, LOAD, No. DRIVER

const int n = 8;

void setup() {
  // the zero refers to the MAX7219 number, it is zero for 1 chip
  lc.shutdown(0, false); // turn off power saving, enables display
  lc.setIntensity(0, 2); // sets brightness (0~15 possible values)
  lc.clearDisplay(0);// clear screen
}
void loop() {
  //some animations
  road();
  delay(500);
  lc.clearDisplay(0);
  bread(); //or pizza
  delay(2000);
  lc.clearDisplay(0);
  flower();
  delay(2000);
  lc.clearDisplay(0);
  heart();
  delay(400);
  lc.clearDisplay(0);
  delay(500);
  heart();
  delay(400);
  lc.clearDisplay(0);
  delay(500);
  heart();
  delay(400);
  lc.clearDisplay(0);
  delay(500);
  heart();
  delay(400);
  lc.clearDisplay(0);
  delay(500);
}

//ROAD
void road() {
  int i = 0, j = 0;
  for (j = 0; j < n; j++) {
    lc.setLed(0, 0, j, true);
    lc.setLed(0, 1, j, true);
    lc.setLed(0, 6, j, true);
    lc.setLed(0, 7, j, true);
  }

  for (i = 0; i <= n; i++)
    for (j = 0; j < n + 3; j++) {
      lc.setLed(0, 3, j - 3, false);
      lc.setLed(0, 4, j - 3, false);
      lc.setLed(0, 3, j - 2, true);
      lc.setLed(0, 4, j - 2, true);
      lc.setLed(0, 3, j - 1, true);
      lc.setLed(0, 4, j - 1, true);
      lc.setLed(0, 3, j, true);
      lc.setLed(0, 4, j, true);
      delay(70);
    }
}

//BREAD
void bread() {
  int i = 0, j = 0;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if ((i + j + 1) < 2 * (n - 1) && (i + j) > 1 && !(j >= (n - 2) && i <= 1 && (j - i) >= n - 2) && !(i >= (n - 2)
&& j <= 1 && (i - j) >= n - 2))
        lc.setLed(0, i, j, true);
  delay(500);
  for (i = 0; i < n / 2; i++) {
    for (j = 0; j < n / 2; j++) {
      lc.setLed(0, i, j, false);
    }
  }
  delay(500);

  for (i = 0; i < n / 2; i++) {
    for (j = 0; j < n / 2 + 1; j++) {
      lc.setLed(0, i, n - j, false);
    }
  }
  delay(500);

  for (i = 0; i < n / 2 + 1; i++) {
    for (j = 0; j < n / 2 + 1; j++) {
      lc.setLed(0, n - i, n - j, false);
    }
  }
  delay(500);

  for (i = 0; i < n / 2 + 1; i++) {
    for (j = 0; j < n / 2 + 1; j++) {
      lc.setLed(0, n - i, j, false);
    }
  }
}

//FLOWER
void flower() {
  for (int i = 3; i >= 1; i--) {
    for (int j = 3; j >= 1; j--) {
      if ((i + j) != 4) {
        lc.setLed(0, i, j, true);
        delay(150);
        lc.setLed(0, n - i - 1, n - j - 1, true);
        delay(150);
        lc.setLed(0, n - i - 1, j, true);
        delay(150);
        lc.setLed(0, i, n - j - 1, true);
        delay(150);
      }
    }
  }
}

//HEART
void heart() {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n / 2; j++) {
      if (
        (i == 0 && (j == 0 || j == 3)) ||
        (i == 5 &&  j == 0) ||
        (i == 6 && (j == 0 || j == 1)) ||
        (i == 7 && (j == 0 || j == 1 || j == 2))
      ) {
        lc.setLed(0, i, j, false);
        lc.setLed(0, i, n - j - 1, false);
      }

      else {
        lc.setLed(0, i, j, true);
        lc.setLed(0, i, n - j - 1, true);
      }
    }
  }
}
```

6. Lab exercise: LED Matrix Control with Joystick

Objective: After connecting an 8x8 LED matrix to an Arduino using a MAX7219 driver, write code to control the position of a lit LED on the matrix using a joystick.

Requirements:

Hardware Setup: Connect an 8x8 LED matrix to the Arduino using the MAX7219 driver. Additionally, connect a joystick to two analog input pins of the Arduino.

Code Implementation: Write Arduino code to achieve the following functionalities:

- Initialize the LED matrix in the setup() function.
- Continuously read the joystick input in the loop() function and determine the direction of movement.
- Update the position of the lit LED on the matrix based on joystick input.
- Ensure the LED wraps around the edges of the matrix when it reaches the boundaries.
- The led should move continuously based on a set move interval when the joystick is oriented in a direction (use millis(), not delay()).

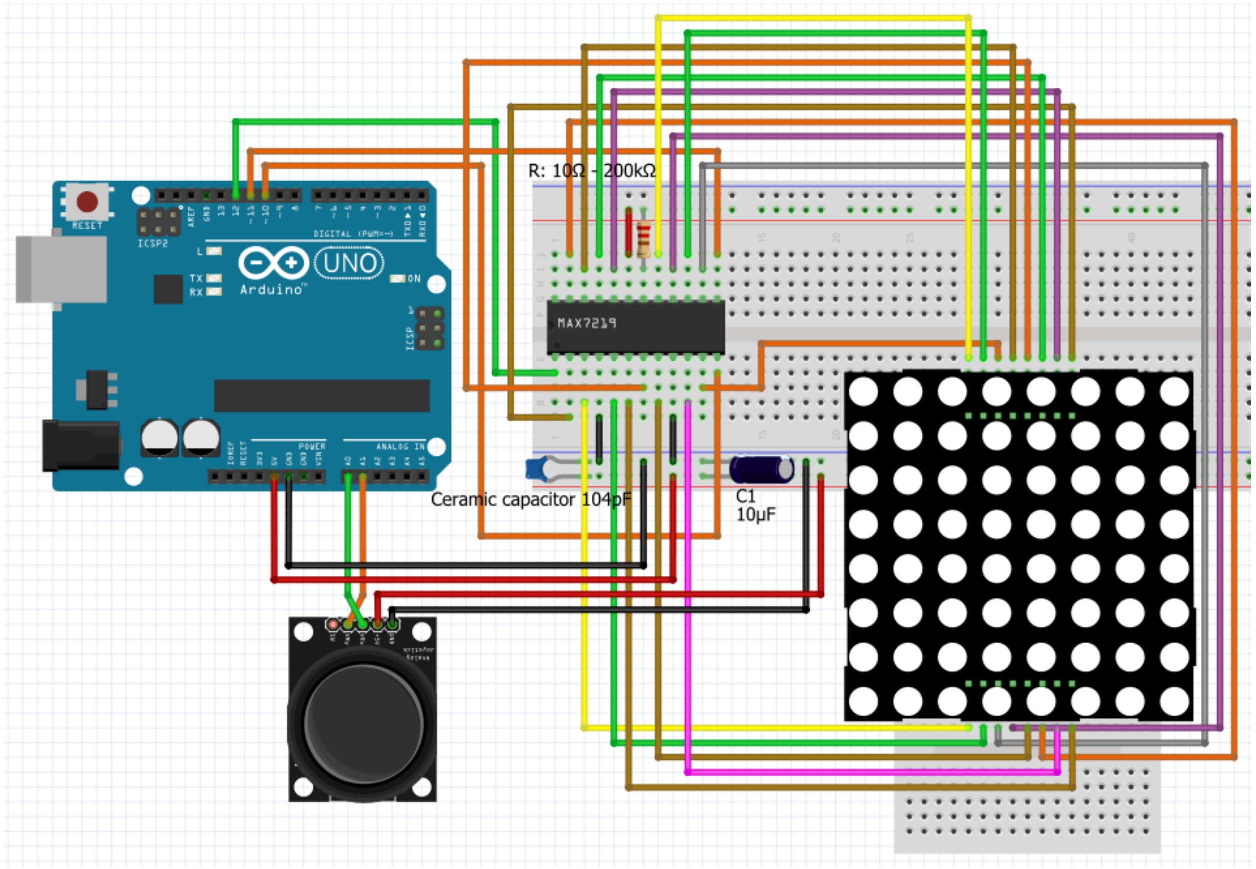


Fig 6.1 Arduino UNO, MAX7219, 1088AS Matrix and Joystick connections

Initial code:

```
#include "LedControl.h" // Include LedControl library for controlling the LED matrix
const int dinPin = 12;
const int clockPin = 11;
const int loadPin = 10;

const int xPin = A0;
const int yPin = A1;
// Create an LedControl object to manage the LED matrix
LedControl lc = LedControl(dinPin, clockPin, loadPin, 1); // DIN, CLK, LOAD, No. DRIVER
// Variable to set the brightness level of the matrix
byte matrixBrightness = 2;
// Variables to track the current and previous positions of the joystick-controlled LED
byte xPos = 0;
byte yPos = 0;
byte xLastPos = 0;
byte yLastPos = 0;
// Thresholds for detecting joystick movement
const int minThreshold = 200;
const int maxThreshold = 600;

const byte moveInterval = 100; // Timing variable to control the speed of LED movement
unsigned long long lastMoved = 0; // Tracks the last time the LED moved

const byte matrixSize = 8 ;// Size of the LED matrix
bool matrixChanged = true; // Flag to track if the matrix display needs updating
// 2D array representing the state of each LED (on/off) in the matrix
byte matrix[matrixSize][matrixSize] = {
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0}
};
// Array representing each row of the LED matrix as a byte
byte matrixByte[matrixSize] = {
  B00000000,
  B01000100,
  B00101000,
  B00010000,
  B00010000,
  B00010000,
  B00000000,
  B00000000
};
void setup() {
  Serial.begin(9600);
  // the zero refers to the MAX7219 number, it is zero for 1 chip
  lc.shutdown(0, false); // turn off power saving, enables display
  lc.setIntensity(0, matrixBrightness); // sets brightness (0~15 possible values)
  lc.clearDisplay(0); // Clear the matrix display
  matrix[xPos][yPos] = 1; // Initialize the starting position of the LED
}
void loop() {
 // TODO: Implement timing logic to control the update rate of the LED movement

  // TODO: Check if the matrix display needs to be updated based on the LED position change

  // TODO: Implement any additional functionality or features as required
}
void updateByteMatrix() {
  for (int row = 0; row < matrixSize; row++) {
    lc.setRow(0, row, matrixByte[row]);  // set each ROW (or COL) at the same time
  }
}

void updateMatrix() {
  for (int row = 0; row < matrixSize; row++) {
    for (int col = 0; col < matrixSize; col++) {
      lc.setLed(0, row, col, matrix[row][col]);  // set each led individually
    }
  }
}
// Function to read joystick input and update the position of the LED
void updatePositions() {
   // TODO: Read analog values from the joystick (X and Y axes)
   // TODO: Implement logic to update the LED position based on joystick input
   //      Make sure to include wrapping logic for the LED position when it reaches the matrix edges

  // TODO: Update the matrixChanged flag and the matrix array to reflect the new LED position
}
```

Solution:

Version 1: explicit conditional statements

```
#include "LedControl.h" // Include LedControl library for controlling the LED matrix
const int dinPin = 12;
const int clockPin = 11;
const int loadPin = 10;

const int xPin = A0;
const int yPin = A1;
// Create an LedControl object to manage the LED matrix
LedControl lc = LedControl(dinPin, clockPin, loadPin, 1); // DIN, CLK, LOAD, No. DRIVER
// Variable to set the brightness level of the matrix
byte matrixBrightness = 2;
// Variables to track the current and previous positions of the joystick-controlled LED
byte xPos = 0;
byte yPos = 0;
byte xLastPos = 0;
byte yLastPos = 0;
// Thresholds for detecting joystick movement
const int minThreshold = 200;
const int maxThreshold = 600;

const byte moveInterval = 100; // Timing variable to control the speed of LED movement
unsigned long long lastMoved = 0; // Tracks the last time the LED moved

const byte matrixSize = 8 ;// Size of the LED matrix
bool matrixChanged = true; // Flag to track if the matrix display needs updating
// 2D array representing the state of each LED (on/off) in the matrix
byte matrix[matrixSize][matrixSize] = {
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0}
};
// Array representing each row of the LED matrix as a byte
```

```
byte matrixByte[matrixSize] = {
  B00000000,
  B01000100,
  B00101000,
  B00010000,
  B00010000,
  B00010000,
  B00000000,
  B00000000
};

void setup() {
  Serial.begin(9600);
  // the zero refers to the MAX7219 number, it is zero for 1 chip
  lc.shutdown(0, false); // turn off power saving, enables display
  lc.setIntensity(0, matrixBrightness); // sets brightness (0~15 possible values)
  lc.clearDisplay(0); // Clear the matrix display
  matrix[xPos][yPos] = 1; // Initialize the starting position of the LED
}
void loop() {
//  updateByteMatrix();
  if (millis() - lastMoved > moveInterval) { // Check if it's time to move the LED
  // game logic
    updatePositions(); // Update the position of the LED based on joystick input
    lastMoved = millis(); // Update the time of the last move
  }
  if (matrixChanged == true) { // Check if the matrix display needs updating
    updateMatrix(); // Update the LED matrix display
    matrixChanged = false; // Reset the update flag
  }
}
// theoretical example
void generateFood() {
  // lastFoodPos = currentPos;
  // newFoodPos = random(ceva);
  // matrix[lastFoodPos] = 0;
  // matrix[newFoodPos] = 1;
  matrixChanged = true;
}



void updateByteMatrix() {
  for (int row = 0; row < matrixSize; row++) {
    lc.setRow(0, row, matrixByte[row]);  // set each ROW (or COL) at the same time
  }
}

void updateMatrix() {
  for (int row = 0; row < matrixSize; row++) {
    for (int col = 0; col < matrixSize; col++) {
      lc.setLed(0, row, col, matrix[row][col]);  // set each led individually
    }
  }
}
// Function to read joystick input and update the position of the LED
void updatePositions() {
  int xValue = analogRead(xPin);
  int yValue = analogRead(yPin);
  // Store the last positions of the LED
  xLastPos = xPos;
  yLastPos = yPos;
  // Update xPos based on joystick movement (X-axis)
  if (xValue < minThreshold) {
    if (xPos < matrixSize - 1) {
      xPos++;
    }
    else {
      xPos = 0;
    }
  }
  if (xValue > maxThreshold) {
    if (xPos > 0) {
      xPos--;
    }
    else {
      xPos = matrixSize - 1;
    }
  }

  if (yValue > maxThreshold) {
    if (yPos < matrixSize - 1) {
      yPos++;
    }
    else {
      yPos = 0;
    }
  }
  // Update xPos based on joystick movement (Y-axis)
  if (yValue < minThreshold) {
    if (yPos > 0) {
      yPos--;
    }
    else {
      yPos = matrixSize - 1;
    }
  }
 // Check if the position has changed and update the matrix if necessary
  if (xPos != xLastPos || yPos != yLastPos) {
    matrixChanged = true;
    matrix[xLastPos][yLastPos] = 0;
    matrix[xPos][yPos] = 1;
  }
}
```

Version 2: More compact conditional statements

This contains a more compact version of the updateMatrix function.
Note: I haven't tested the code, but I believe it should work fine.

```
#include "LedControl.h" // Include the LedControl library for controlling LED matrices

// Pin definitions for the LED matrix
const int dinPin = 12;
const int clockPin = 11;
const int loadPin = 10;

// Pin definitions for the joystick
const int xPin = A0;
const int yPin = A1;

// Create an LedControl object to interface with the LED matrix
LedControl lc = LedControl(dinPin, clockPin, loadPin, 1); // DIN, CLK, LOAD, number of devices

// Variables to control LED matrix brightness and position
byte matrixBrightness = 2;
byte xPos = 0;
byte yPos = 0;
byte xLastPos = 0;
byte yLastPos = 0;

// Thresholds for joystick movement detection
const int minThreshold = 200;
const int maxThreshold = 600;

// Timing variables for movement updates
const byte moveInterval = 100;
unsigned long lastMoved = 0;

// Size of the LED matrix (8x8)
const byte matrixSize = 8;
bool matrixChanged = true;

// 2D array representing the state (on/off) of each LED in the matrix
byte matrix[matrixSize][matrixSize] = { /* Initial state with all LEDs off */ };

// Setup function runs once when the sketch starts
```

```
void setup() {
  Serial.begin(9600); // Start serial communication for debugging

  // Initialize the LED matrix
  lc.shutdown(0, false); // Disable power saving, turn on the display
  lc.setIntensity(0, matrixBrightness); // Set the brightness level
  lc.clearDisplay(0); // Clear the display initially
  matrix[xPos][yPos] = 1; // Turn on the initial LED position
}

// Main loop, runs continuously
void loop() {
  // Check if it's time to move the LED
  if (millis() - lastMoved > moveInterval) {
    updatePositions(); // Update the LED position based on joystick input
    lastMoved = millis(); // Reset the movement timer
  }

  // Update the LED matrix display if there's been a change
  if (matrixChanged) {
    updateMatrix();
    matrixChanged = false;
  }
}

// Function to update the LED matrix display
void updateMatrix() {
  for (int row = 0; row < matrixSize; row++) {
    for (int col = 0; col < matrixSize; col++) {
      lc.setLed(0, row, col, matrix[row][col]); // Update each LED state
    }
  }
}

// Function to read joystick input and update LED positions accordingly
void updatePositions() {
  int xValue = analogRead(xPin); // Read the X-axis value
  int yValue = analogRead(yPin); // Read the Y-axis value

  // Store the last positions
  xLastPos = xPos;
  yLastPos = yPos;
  // Update xPos based on joystick movement
  if (xValue < minThreshold) {
    xPos = (xPos + 1) % matrixSize;
  } else if (xValue > maxThreshold) {
    xPos = (xPos > 0) ? xPos - 1 : matrixSize - 1;
  }

  // Update yPos based on joystick movement
  if (yValue < minThreshold) {
    yPos = (yPos > 0) ? yPos - 1 : matrixSize - 1;
  } else if (yValue > maxThreshold) {
    yPos = (yPos + 1) % matrixSize;
  }

  // Check if the position has changed and update the matrix accordingly
  if (xPos != xLastPos || yPos != yLastPos) {
    matrixChanged = true;
    matrix[xLastPos][yLastPos] = 0; // Turn off the LED at the last position
    matrix[xPos][yPos] = 1; // Turn on the LED at the new position
  }
}
```

7. Extra exercises:

1. Pattern Drawing: Allow the joystick to draw permanent patterns on the matrix. Moving the joystick lights up LEDs without turning them off when moving away.
2. Blinking LED: Modify the code so that the LED on the current position blinks at a regular interval. This involves toggling the LED state between on and off within the loop() function using a timing mechanism similar to the movement control.
3. Adjustable Speed: Implement a feature that allows the speed of the LED movement to be controlled by the joystick's vertical position. The higher the joystick, the faster the LED moves, and vice versa.
4. LED Trail Effect: Create a trail effect where the LED leaves a fading trail as it moves across the matrix. This could involve gradually dimming the LEDs in the trail over time.
5. Random LED Flash: Add functionality where a random LED on the matrix flashes briefly at random intervals, independent of the joystick-controlled LED.
6. Boundary Collision Alert: Make the edges of the matrix flash or change color when the joystick-controlled LED moves to the edge of the matrix, indicating a boundary collision.
7. Two-LED Control: Introduce a second LED that can be controlled independently using specific joystick movements (like pressing the joystick down).
8. Invert Control: Implement a mode where the joystick controls are inverted (up moves down, left moves right, etc.).
9. Dynamic Brightness Control: Use the joystick button (if available) to cycle through different brightness levels of the LED.
10. Matrix Reset: Implement a double joystick press action that resets the matrix, turning all LEDs off except the controlled one.

Resources

1. Matrix and scanning explanation https://www.youtube.com/watch?v=G4lIo-MRSiY
2. Led matrix editor: https://xantorohara.github.io/led-matrix-editor/
3. https://www.youtube.com/watch?v=X9tsfOeYnAU
4. https://www.youtube.com/watch?v=3FJuhMNPibQ