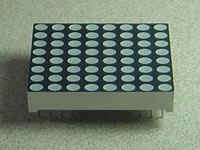
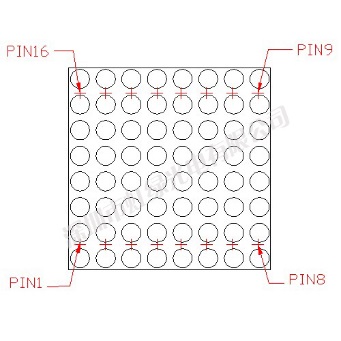
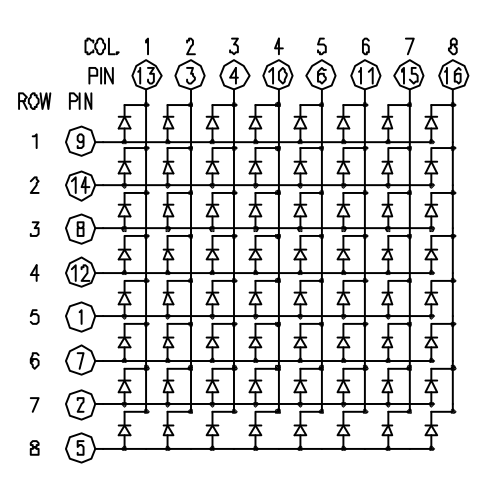
Row-columm Scanning to control an 8x8 LED Matrix

LED displays are often packaged as matrixes of LEDs arranged in rows of common anodes and columns of common cathodes, or the reverse. Here's a [typical example](http://sigma.octopart.com/140413/datasheet/Lumex-LDM-24488NI.pdf), and its schematic:



These can be very useful displays. To control a matrix, you connect both its rows and columns to your microcontroller. The columns are connected to the LEDs cathodes, so a column needs to be LOW for any of the LEDs in that column to turn on. The rows are connected to the LEDs anodes, so the row needs to be HIGH for an individual LED to turn on. If the row and the column are both high or both low, no voltage flows through the LED and it doesn’t turn on.

To control an individual LED, you set its column LOW and its row HIGH. To control multiple LEDs in a row, you set the row HIGH, then take the column high, then set the columns LOW or HIGH as appropriate; a LOW column will turn the corresponding LED ON, and a HIGH column will turn it off.

Tip - Pins set to OUTPUT by use of the PinMode command are set to LOW if not otherwise stated

Although there are pre-made LED matrices, you can also make your own matrix from 64 LEDs, using the schematic as shown above.

It doesn’t matter which pins of the microcontroller you connect the rows and columns to, because you can assign things in software. Connected the pins in a way that makes wiring easiest. A typical layout is shown below.

Here's a matrix of the pin connections, based on the diagram above:

|  |  |  |  |
| --- | --- | --- | --- |
| Matrix pin no. | Row | Column | Arduino pin number |
| 1 | 5 | - | 13 |
| 2 | 7 | - | 12 |
| 3 | - | 2 | 11 |
| 4 | - | 3 | 10 |
| 5 | 8 | - | 16 (analog pin 2) |
| 6 | - | 5 | 17 (analog pin 3) |
| 7 | 6 | - | 18 (analog pin 4) |
| 8 | 3 | - | 19 (analog pin 5) |
| 9 | 1 | - | 2 |
| 10 | - | 4 | 3 |
| 11 | - | 6 | 4 |
| 12 | 4 | - | 5 |
| 13 | - | 1 | 6 |
| 14 | 2 | - | 7 |
| 15 | - | 7 | 8 |
| 16 | - | 8 | 9 |

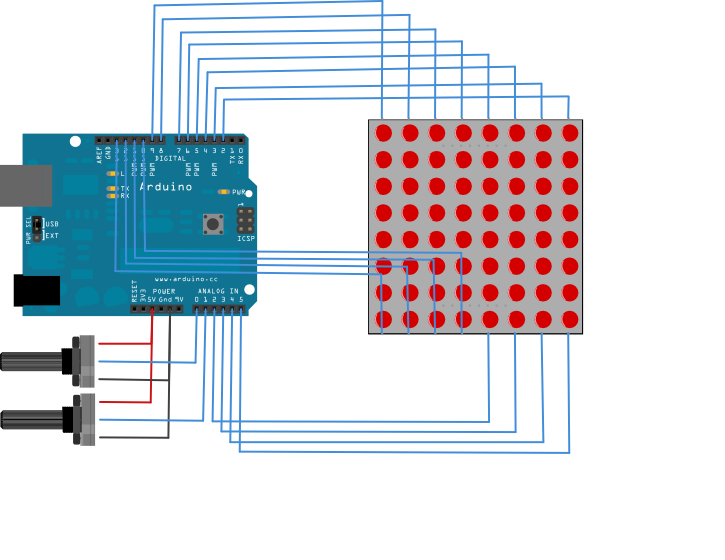
Hardware Required

1. Arduino or Genuino Board
2. 8 x 8 LED Matrix
3. 2 10k ohm potentiometers
4. hook-up wires
5. breadboard

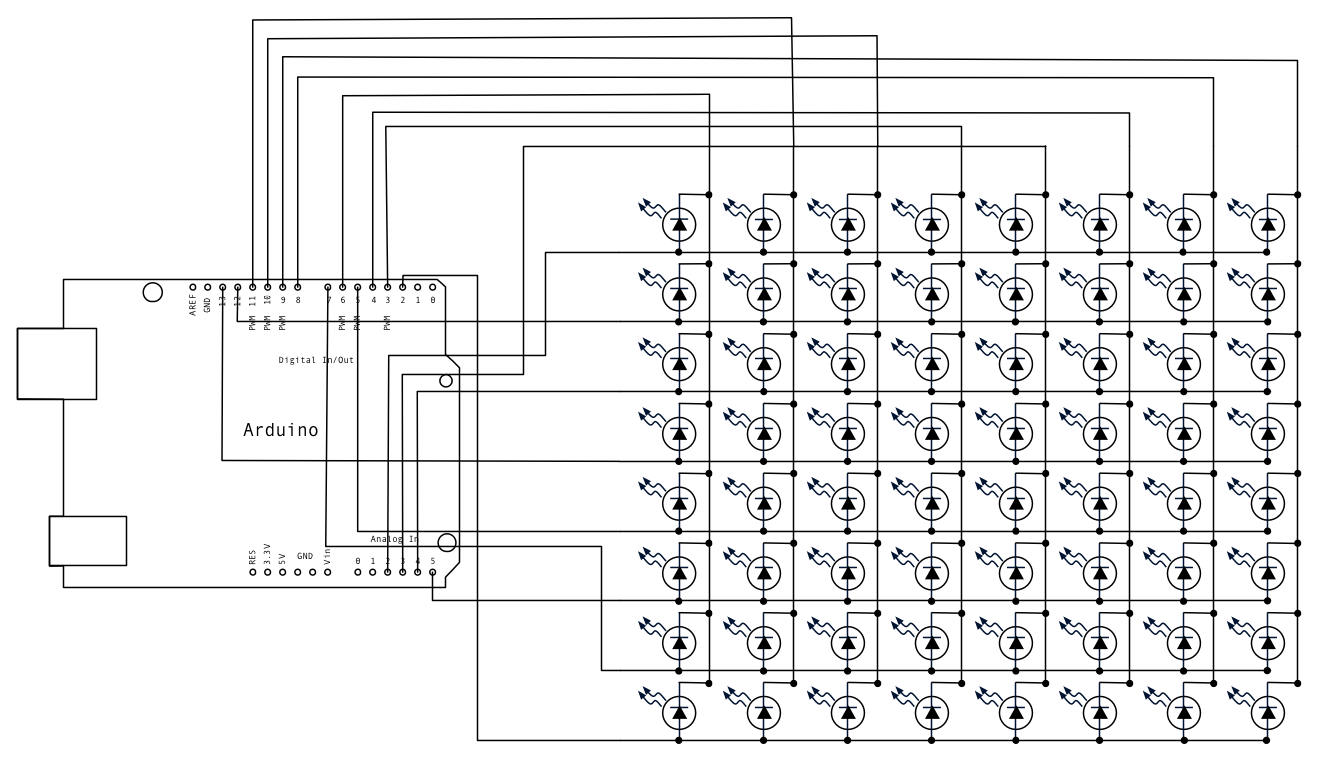
Circuit

The 16 pins of the matrix are hooked up to 16 pins of the Arduino or Genuino board. Four of the analog pins are used as digital inputs 16 through 19. The order of the pins is assigned in two arrays in the code.

Two potentiometers, connected to analog pins 0 and 1, control the movement of a lit LED in the matrix.

[](https://www.arduino.cc/en/uploads/Tutorial/rowColScan_bb.png)

Schematic

[](https://www.arduino.cc/en/uploads/Tutorial/rowColScan_schem.png)

Code

#define ROW\_1 2

#define ROW\_2 7

#define ROW\_3 19

#define ROW\_4 5

#define ROW\_5 13

#define ROW\_6 18

#define ROW\_7 12

#define ROW\_8 16

#define COL\_1 9

#define COL\_2 8

#define COL\_3 4

#define COL\_4 17

#define COL\_5 3

#define COL\_6 10

#define COL\_7 11

#define COL\_8 6

const byte rows[] = {

   ROW\_1, ROW\_2, ROW\_3, ROW\_4, ROW\_5, ROW\_6, ROW\_7, ROW\_8

};

const byte cols[] = {

 COL\_1,COL\_2, COL\_3, COL\_4, COL\_5, COL\_6, COL\_7, COL\_8

};

// The display buffer

// It's prefilled with a smiling face (1 = ON, 0 = OFF)

// https://xantorohara.github.io/led-matrix-editor/

byte smileFace[] = {B11000011,B10111101,B01011010,B01111110,B01011010,B01100110,B10111101,B11000011};

byte sadFace[] = {B11000011,B10111101,B01011010,B01111110,B01100110,B01011010,B10111101,B11000011};

float timeCount = 0;

const int speedChange = 100;

void setup()

{

   // Open serial port

**Serial**.begin(9600);

   // Set all used pins to OUTPUT

   // This is very important! If the pins are set to input

   // the display will be very dim.

   for (byte i = 2; i <= 13; i++)

       pinMode(i, OUTPUT);

   pinMode(A2, OUTPUT);

   pinMode(A3, OUTPUT);

   pinMode(A4, OUTPUT);

   pinMode(A5, OUTPUT);

}

void loop() {

 delay(5);

 timeCount += 1;

 if(timeCount <  speedChange) {

   drawMatrix(smileFace);

 }

 else if(timeCount <  2\*speedChange) {

   drawMatrix(sadFace);

 }

 else {

   // back to the start

   timeCount = 0;

 }

}

void  drawMatrix(byte data[]){

 // Turn on each row in series

 for (byte i = 0; i < 8; i++) {

   digitalWrite(rows[i], HIGH);    //initiate whole row

   // Turn on each point in row

   for (byte a = 0; a < 8; a++){

     // if You set (~data[i] >> a) then You will have positive

     digitalWrite(cols[a], (data[i] >> a) & 0x01); // initiate whole column

     delayMicroseconds(100);       // uncoment deley for diferent speed of display

     digitalWrite(cols[a], HIGH);      // reset whole column

   }

   digitalWrite(rows[i], LOW);     // reset whole row

   // otherwise last row will intersect with next row

 }

}

Symbol and characters design

The symbol and characters can be design at the following link:

<https://xantorohara.github.io/led-matrix-editor/>

Here are some characters are designed.

byte CLEAR[] =  {B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111};

byte EX[] =   {B11111111,B11101111,B11101111,B11101111,B11101111,B11111111,B11101111,B11111111};

byte A[] = {B11100111,B11000011,B10011001,B10000001,B10011001,B10011001,B10011001,B10011001};

byte B[] = {B10000111,B10000011,B10011011,B10000111,B10011011,B10011011,B10000011,B10000111};

byte C[] = {B11000011,B10000001,B10011101,B10011111,B10011111,B10011101,B10000001,B11000011};

byte D[] = {B10000111,B10000011,B10011001,B10011001,B10011001,B10011001,B10000011,B10000111};

byte E[] = {B10000011,B10000011,B10011111,B10000011,B10011111,B10011111,B10000011,B10000011};

byte F[] = {B10000011,B10000011,B10011111,B10000011,B10011111,B10011111,B10011111,B10011111};

byte G[] = {B11000011,B10000001,B10011111,B10010000,B10011001,B10011001,B10000001,B11000011};

byte H[] = {B10011001,B10011001,B10011001,B10000001,B10011001,B10011001,B10011001,B10011001};

byte I[] = {B10000001,B10000001,B11100111,B11100111,B11100111,B11100111,B10000001,B10000001};

byte J[] = {B10000001,B10000001,B11111001,B11111001,B11111001,B10011001,B10000001,B11000011};

byte K[] = {B10011101,B10011011,B10010111,B10001111,B10000111,B10010011,B10011001,B10011100};

byte L[] = {B10011111,B10011111,B10011111,B10011111,B10011111,B10011001,B10000001,B10000001};

byte M[] = {B10011100,B10001000,B10000000,B10010100,B10011100,B10011100,B10011100,B10011100};

byte N[] = {B10011100,B10001100,B10000100,B10010000,B10011000,B10011100,B10011100,B10011100};

byte O[] = {B11000011,B10000001,B10011001,B10011001,B10011001,B10011001,B10000001,B11000011};

byte P[] = {B10000011,B10000001,B10011001,B10011001,B10000011,B10011111,B10011111,B10011111};

byte Q[] = {B11000011,B10000001,B10011001,B10011001,B10011001,B10010001,B10011001,B11000010};

byte R[] = {B10000011,B10000001,B10011001,B10000011,B10000111,B10010011,B10011001,B10011100};

byte S[] = {B11000111,B10011011,B10011111,B11000111,B11100011,B11111011,B10010011,B11000111};

byte T[] = {B10000001,B10000001,B11100111,B11100111,B11100111,B11100111,B11100111,B11100111};

byte U[] = {B10011001,B10011001,B10011001,B10011001,B10011001,B10011001,B10011001,B11000011};

byte V[] = {B10011100,B10011100,B10011100,B10011100,B10011100,B11011101,B11101011,B11110111};

byte W[] = {B10011100,B10011100,B10011100,B10011100,B10011100,B10010100,B10001000,B10011100};

byte X[] = {B10011100,B10011100,B11001001,B11110111,B11101011,B11011001,B10011100,B10011100};

byte Y[] = {B10011001,B10011001,B10011001,B11011011,B11100111,B11100111,B11100111,B11100111};

byte Z[] = {B10000001,B10011001,B11110001,B11100011,B11001111,B10011101,B10000001,B10000001};