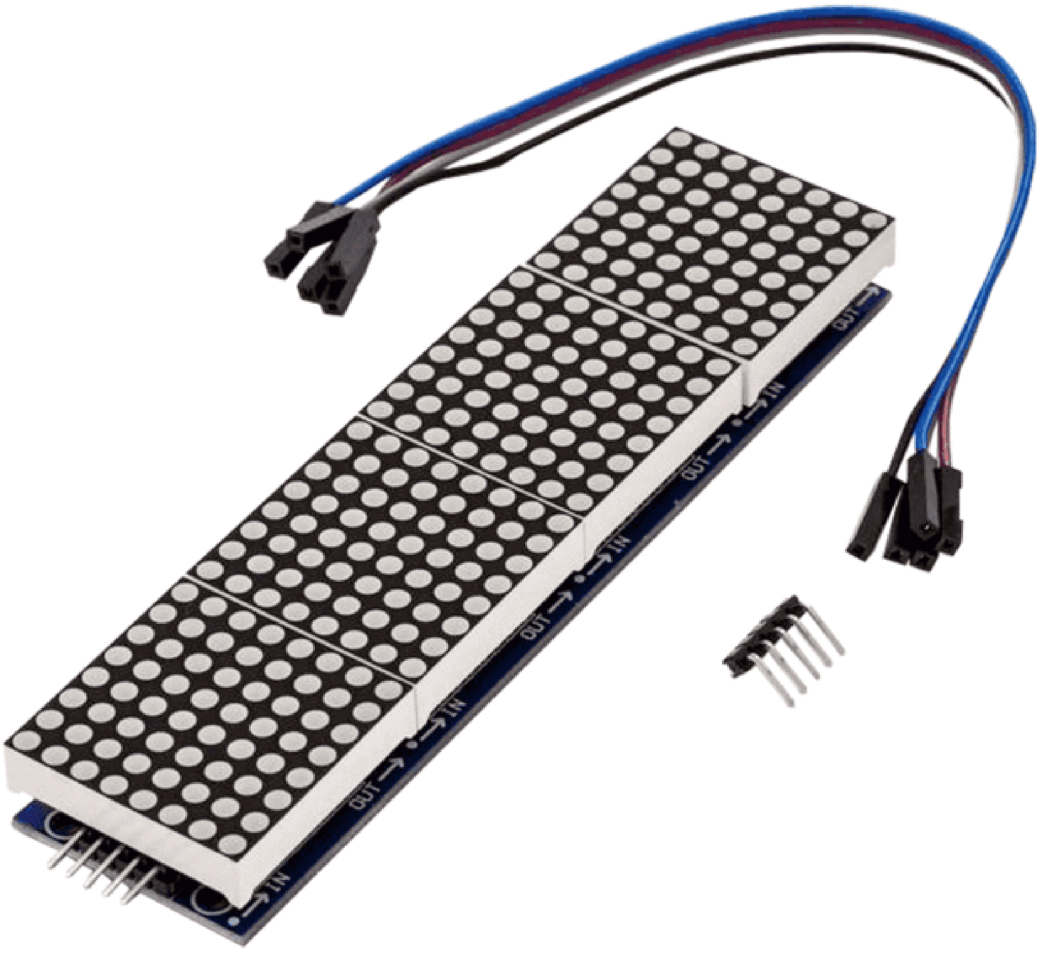
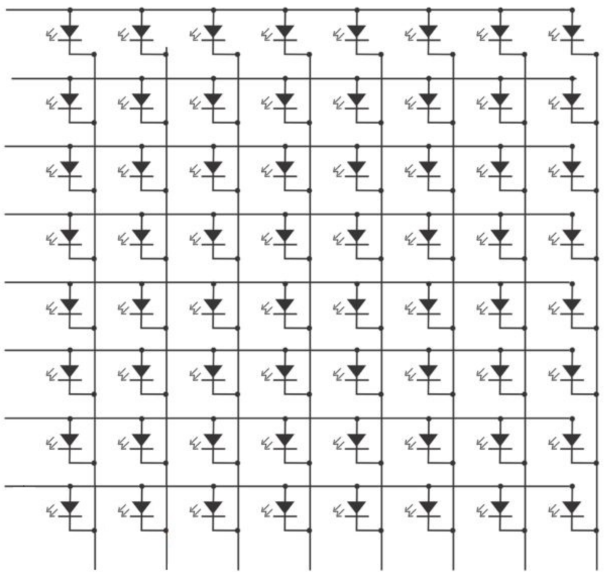
Welcome!

Thank you for purchasing our AZ-Delivery *4x64 LED Matrix Display Module*. On the following pages, we will introduce you to how to use and set-up this handy device.

Have fun!



The *4x64* LED matrix screen is a screen with arrays of LEDs stacked one by one in a *"matrix"*. The *4x64* LED display is made up of four smaller *8x8* screens which are called *"segments"*. They are called *8x8* because there are *8* LEDs in one row, and there are *8* rows in one segment which creates a matrix of *64* LEDs (*64* pixels). Each LED needs two wires to work (one for power and one for ground). To avoid having to wire all *64* LEDs one by one, they are connected into a LED matrix, as shown on the image below:



Putting the LEDs into a matrix reduces the number of output pins to *16*, which is still too much, that is why LED matrix drivers were invented. In our case, used LED matrix driver is called the "*MAX7219*", which uses the SPI interface. The *4x64* LED matrix screen has four segments, and one driver chip per segment. Driver chips are connected serially in the *"Daisy chain"*. It is possible to stack more, but you have to use external power supply.

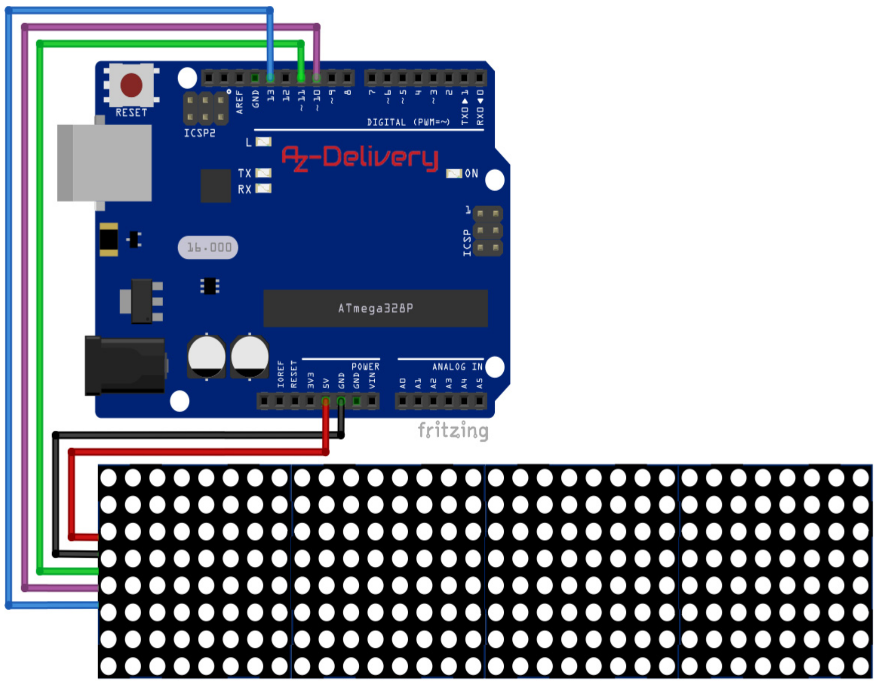
The MAX7219 chip uses the SPI interface for communication with microcontrollers. SPI stands of a synchronous serial peripheral communication interface, and it is used for short-distance communication, primarily in embedded systems. Synchronous means that the data is being sent and recieved within a specific clock cycle, and serial means that the data is sent serially, bit by bit.

Specifications:

»	Power supply and logic voltage range:	from +3.3V to +5V DC
»	Color of LEDs:	red
»	Brightness control:	digital and analog
»	Dimensions:	32 x 129mm [1.26 x 5.1in]
»	10MHz SPI interface	
»	Individual LED segment control	
»	Display Blanked on Power-Up	
»	Drive Common-Cathode LED Screens	

Connecting the display with Atmega328P Board

Connect the module with the microcontroller board as shown on the following connection diagram:

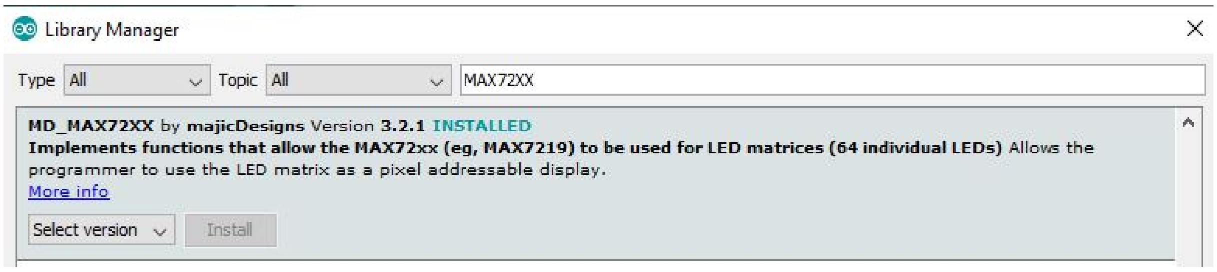


Module pin		**>**	**Board pin**	
VCC		>	5V	**Red wire**
GND		>	GND	**Black wire**
CS	(SS)	>	D10 pin	**Purple wire**
DATA	(MOSI)	>	D11 pin	**Green wire**
CLK	(SCK)	>	D13 pin	**Blue wire**

Make sure to read the pin designations before you start connecting them, as there are different versions of the module. We have included some alternative pin names inside of the parentheses.

Arduino IDE library

To use the module with the Atmega328P Board, we recommend to download the external library for it. First, go to *Tools* > *Manage Libraries*, and when a new pop-up window appears, type *"MAX72XX"* into the search box. The library that we are going to use is called *"MD_MAX72XX"* made by *"majicDesigns"*, as shown on the image below:



MD_MAX72XX by majicDesigns Version 3.2.1 INSTALLED
Implements functions that allow the MAX72xx (eg, MAX7219) to be used for LED matrices (64 individual LEDs) Allows the programmer to use the LED matrix as a pixel addressable display.

Installing is as simple as clicking the *"Install"* button which appears when you hover with your mouse over the search result.

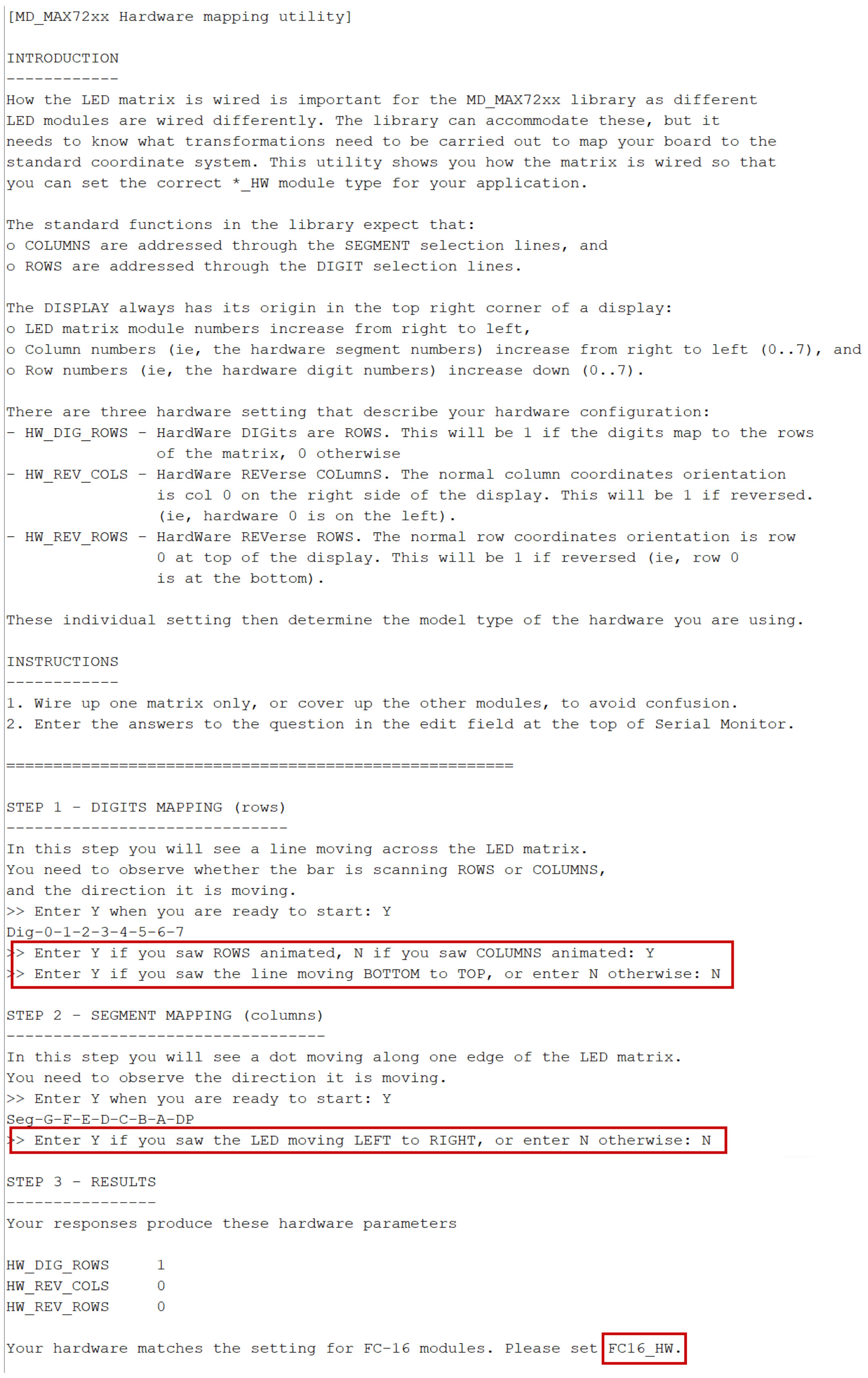
There are many versions of these displays. In order to detect which one you are using, you can run Hardware Mapper sketch that comes with the library. To open the sketch, go to:
File > *Examples* > *MD_MAX72XX* > *MD_MAX72xx_HW_Mapper,*
and change the following line of the code:

```
#define SERIAL_SPEED 57600
```

into:

```
#define SERIAL_SPEED 9600
```

Upload the sketch to the microcontroller board and open the Serial Monitor (*Tools* > *Serial Monitor*). After that, follow the three-step instructions that microcontroller board sends to the Serial Monitor, as shown on the image on the next page:


```
[MD_MAX72xx Hardware mapping utility]

INTRODUCTION
------------
How the LED matrix is wired is important for the MD_MAX72xx library as different
LED modules are wired differently. The library can accommodate these, but it
needs to know what transformations need to be carried out to map your board to the
standard coordinate system. This utility shows you how the matrix is wired so that
you can set the correct *_HW module type for your application.

The standard functions in the library expect that:
o COLUMNS are addressed through the SEGMENT selection lines, and
o ROWS are addressed through the DIGIT selection lines.

The DISPLAY always has its origin in the top right corner of a display:
o LED matrix module numbers increase from right to left,
o Column numbers (ie, the hardware segment numbers) increase from right to left (0..7), and
o Row numbers (ie, the hardware digit numbers) increase down (0..7).

There are three hardware setting that describe your hardware configuration:
- HW_DIG_ROWS - HardWare DIGits are ROWS. This will be 1 if the digits map to the rows
                of the matrix, 0 otherwise
- HW_REV_COLS - HardWare REVerse COLumnS. The normal column coordinates orientation
                is col 0 on the right side of the display. This will be 1 if reversed.
                (ie, hardware 0 is on the left).
- HW_REV_ROWS - HardWare REVerse ROWS. The normal row coordinates orientation is row
                0 at top of the display. This will be 1 if reversed (ie, row 0
                is at the bottom).

These individual setting then determine the model type of the hardware you are using.

INSTRUCTIONS
------------
1. Wire up one matrix only, or cover up the other modules, to avoid confusion.
2. Enter the answers to the question in the edit field at the top of Serial Monitor.

=======================================================

STEP 1 - DIGITS MAPPING (rows)
------------------------------
In this step you will see a line moving across the LED matrix.
You need to observe whether the bar is scanning ROWS or COLUMNS,
and the direction it is moving.
>> Enter Y when you are ready to start: Y
Dig-0-1-2-3-4-5-6-7
>> Enter Y if you saw ROWS animated, N if you saw COLUMNS animated: Y
>> Enter Y if you saw the line moving BOTTOM to TOP, or enter N otherwise: N

STEP 2 - SEGMENT MAPPING (columns)
----------------------------------
In this step you will see a dot moving along one edge of the LED matrix.
You need to observe the direction it is moving.
>> Enter Y when you are ready to start: Y
Seg-G-F-E-D-C-B-A-DP
>> Enter Y if you saw the LED moving LEFT to RIGHT, or enter N otherwise: N

STEP 3 - RESULTS
----------------
Your responses produce these hardware parameters

HW_DIG_ROWS     1
HW_REV_COLS     0
HW_REV_ROWS     0

Your hardware matches the setting for FC-16 modules. Please set FC16_HW.
```

To get the right result, you have to make the orientation of the screen like on the connection diagram. Right side of the screen is the side where the input pins of the screen are located (where we connected wires).

The result hardware map is "*FC16_HW*", thus when using AZ-Delivery 64 LED matrix display module you need to set the "*HARDWARE_TYPE"* macro to this result value.

Sketch code:

```
#include <MD_MAX72xx.h >

#define PRINT(s, x) { Serial.print(F(s)); Serial.print(x); }
#define PRINTS(x) Serial.print(F(x))
#define PRINTD(x) Serial.println(x, DEC)
#define PRINT(s, x)
#define PRINTS(x)
#define PRINTD(x)

#define HARDWARE_TYPE MD_MAX72XX::FC16_HW

#define MAX_DEVICES  4

#define CLK_PIN   13  // or SCK
#define DATA_PIN  11  // or MOSI
#define CS_PIN    10  // or SS

MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

#define  DELAYTIME  100  // in milliseconds
```

```
void scrollText(char *p) {
  uint8_t charWidth;
  uint8_t cBuf[8];  // this should be ok for all built-in fonts
  PRINTS("\nScrolling text");
  mx.clear();
  while (*p != '\0') {
    charWidth = mx.getChar(*p++, sizeof(cBuf) / sizeof(cBuf[0]), cBuf);
    // allow space between characters
    for (uint8_t i=0; i<=charWidth; i++) {
      mx.transform(MD_MAX72XX::TSL);
      if (i < charWidth) {
        mx.setColumn(0, cBuf[i]);
      }
      delay(DELAYTIME);
    }
  }
}

void rows() {
  mx.clear();
  for (uint8_t row=0; row<ROW_SIZE; row++) {
    mx.setRow(row, 0xff); delay(2*DELAYTIME);
    mx.setRow(row, 0x00);
  }
}

void columns() {
  mx.clear();
  for (uint8_t col=0; col<mx.getColumnCount(); col++) {
    mx.setColumn(col, 0xff); delay(DELAYTIME/MAX_DEVICES);
    mx.setColumn(col, 0x00);
  }
}
```

```
void stripe() {
  const uint16_t maxCol = MAX_DEVICES*ROW_SIZE;
  const uint8_t stripeWidth = 10;
  mx.clear();
  for (uint16_t col=0; col<maxCol + ROW_SIZE + stripeWidth; col++) {
    for(uint8_t row=0; row < ROW_SIZE; row++) {
      mx.setPoint(row, col-row, true);
      mx.setPoint(row, col-row - stripeWidth, false);
    }
    delay(DELAYTIME);
  }
}

void spiral() {
  int  rmin = 0, rmax = ROW_SIZE-1;
  int  cmin = 0, cmax = (COL_SIZE*MAX_DEVICES)-1;
  mx.clear();
  while ((rmax > rmin) && (cmax > cmin)) {
    for (int i=cmin; i<=cmax; i++) { // do row
      mx.setPoint(rmin, i, true); delay(DELAYTIME/MAX_DEVICES);
    }
    rmin++;
    for (uint8_t i=rmin; i<=rmax; i++) { // do column
      mx.setPoint(i, cmax, true); delay(DELAYTIME/MAX_DEVICES);
    }
    cmax--;
    for (int i=cmax; i>=cmin; i--) { // do row
      mx.setPoint(rmax, i, true); delay(DELAYTIME/MAX_DEVICES);
    }
    rmax--;
    for (uint8_t i=rmax; i>=rmin; i--) { // do column
      mx.setPoint(i, cmin, true); delay(DELAYTIME/MAX_DEVICES);
    }
    cmin++;
  }
}
```

```
void bounce() {
  const int minC = 0;
  const int maxC = mx.getColumnCount()-1;
  const int minR = 0;
  const int maxR = ROW_SIZE-1;
  int  nCounter = 0;
  int  r = 0, c = 2;
  int8_t dR = 1, dC = 1;  // delta row and column
  mx.clear();
  while (nCounter++ < 200) {
    mx.setPoint(r, c, false);
    r += dR; c += dC;
    mx.setPoint(r, c, true);
    delay(DELAYTIME/2);
    if ((r == minR) || (r == maxR)) {
      dR = -dR;
    }
    if ((c == minC) || (c == maxC)) {
      dC = -dC;
    }
  }
}

void intensity() {
  uint8_t row;
  mx.clear();
  for (int8_t i=0; i<=MAX_INTENSITY; i++) {
    mx.control(MD_MAX72XX::INTENSITY, i);
    mx.setRow(0, 0xff); delay(DELAYTIME*10);
  }
  mx.control(MD_MAX72XX::INTENSITY, 8);
}
```

```
void transformation() {
  uint8_t arrow[COL_SIZE] = {
    0b00001000,
    0b00011100,
    0b00111110,
    0b01111111,
    0b00011100,
    0b00011100,
    0b00111110,
    0b00000000 };
  MD_MAX72XX::transformType_t  t[] = {
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TFLR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TRC,
    MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD,
    MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD,
    MD_MAX72XX::TFUD,
    MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU,
    MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU,
    MD_MAX72XX::TINV,
    MD_MAX72XX::TRC, MD_MAX72XX::TRC, MD_MAX72XX::TRC, MD_MAX72XX::TRC,
    MD_MAX72XX::TINV };
  mx.clear();
  mx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::OFF);
  for (uint8_t j=0; j<mx.getDeviceCount(); j++) {
    mx.setBuffer(((j+1)*COL_SIZE)-1, COL_SIZE, arrow);
  }
  mx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::ON);
  delay(DELAYTIME);
  mx.control(MD_MAX72XX::WRAPAROUND, MD_MAX72XX::ON);
// one tab
  for (uint8_t i=0; i<(sizeof(t)/sizeof(t[0])); i++) {
    mx.transform(t[i]); delay(DELAYTIME*4);
  }
  mx.control(MD_MAX72XX::WRAPAROUND, MD_MAX72XX::OFF);
}

void showCharset() {
  mx.clear();
  mx.update(MD_MAX72XX::OFF);
  for (uint16_t i=0; i<256; i++) {
    mx.clear(0);
    mx.setChar(COL_SIZE-1, i);
    if (MAX_DEVICES >= 3) {
      char hex[3];
      sprintf(hex, "%02X", i);
      mx.clear(1); mx.setChar((2*COL_SIZE)-1, hex[1]);
      mx.clear(2); mx.setChar((3*COL_SIZE)-1, hex[0]);
    }
    mx.update();
    delay(DELAYTIME*2);
  }
  mx.update(MD_MAX72XX::ON);
}
void setup() { mx.begin(); }
void loop() {
  scrollText("Graphics");
  rows();
  columns();
  stripe();
  bounce();
  spiral();
  intensity();
  transformation();
  showCharset();
  delay(3000);
}
```

Sketch starts with including the “*MD_MAX72XX.h*” library, then continues with several definitions of macros. A special example of using one of the macros is `Serial.print(F(x))`. `F(x)` is a macro that stores “*x*” string in flash memory, not SRAM. Flash memory is a memory in the microcontrollers where programs are stored, and SRAM memory is a type of RAM memory. We decrease limit of memory for our programs when using `F(x)` macro, which increases SRAM memory (very important for running programs on microcontrollers). If you do not use `F(x)` macro, most likely that your program will not work as designed. That is because, in the sketch, there are too many strings, which in the *C* programming language are char arrays and that eats up SRAM memory easily.

Than we specify which display we are using, by setting *`HARDWARE_TYPE`* macro to *`FC16_HW`* (the result value we got in the previous chapter).

After this, we specify how many segments LED matrix screen has by setting *`MAX_DEVICES`* macro to *4*. If you connect multiple LED matrix screens in the Daisy chain, you have to change this value accordingly.

There are three more macros which define the SPI interface pins.

After that, we create “*mx*” object, the screen object that is used throughout the whole sketch.

After all of this, many functions are created.

First function is called `scrollingText()`, which is used for scrolling text on display. The `scrollingText()` function use the function from the "*MD_MAX72XX.h*" library called `setColumn()`. This function turns *ON* or *OFF* one column array of LEDs on the screen. The rest of the function is algorithm for the scrolling effect. We take string "*p*" from flash memory, and convert it into an array of unsigned integers which represents the LEDs in the column array. Then we display that array on the screen. At the end of the function we specify the delay time, which is the speed of the scrolling text.

Second function is called `rows()`, which uses function `setRow()` from the "*MD_MAX72XX.h*" library. `setRow()` function accepts two arguments. First which row will be used, where *0* means first row starting from the top. And second argument is number in hexadecimal, that represents *8* bit value for row array of pixels (*8* pixels in one row). In this case we use *0xFF* which turns *ON* all pixels of a specific row.

Third function is called `columns()`, which is the same as `rows()` but in this case we turn *ON* column arrays of pixels. The function uses function `setColumn()` from the "*MD_MAX72XX.h*" library. `setColumn()` function is the same as `setRow()` but just for columns.

Fourth function is called *`stripe()`*, which displays scrolling diagonal stripe (*4* pixels wide), on the whole screen. The function uses the *`setPoint()`* function from the "*MD_MAX72XX.h*" library. *`setPoint()`* function is used to turn *ON* specific LED on the screen, which accepts three arguments. First and second arguments are the *X* and *Y* position of the LED (top left corner of the screen is *X=0* and *Y=0*, bottom right corner of the screen is *X=31* and *Y=7*). Third argument is a boolean value representing state of LED, turned *ON* = *true*, and turned *OFF* = *false*. At the end of the function we specify the delay time, which is used for changing the speed of scrolling stripe.

Fifth function is called *`spiral()`*, which displays snake like line that turns all pixels on the screen in spiral moving effect. The function uses *`setPoint()`* function from the "*MD_MAX72XX.h*" library. The rest of the function is the algorithm of turning *ON* LED by LED until all LEDs are turned *ON*.

Sixth function is called *`bounce()`* which displays moving point that bounces up of edges of the LED matrix screen. The functions uses few functions that we already explained, and an additional function *`getColumnCont()`* which returns the number of columns on LED matrix display. The rest of the function is the algorithm for moving pixel.

Seventh function is called `intensity()` which is used to turn *ON* only the first row of the LEDs on the screen. The function uses `control()` function from the "*MD_MAX72XX.h*" library. `control()` function accepts two arguments, the first is defines which property of "*mx*" object we will change, where we store value "*MD_MAX72XX::INTENSITY*". So we will change the intensity of brightness. The second argument is a number that represents the level of intensity and its value is in the range from *0* to *15*, or *16* different levels (default value is *8*).

Eighth function is called `transformation()`, which is used to animate bitmap image on the screen. The bitmap image data is stored in the array of unsigned integers. This array contains *8* elements, which contain binary numbers. These binary numbers represent the rows of the one segment of the screen, where binary value *0* represents turned *OFF* LED, and value *1* represents turned *ON* LED. We defined one more array called "*t*", which contains enumerations used for the animation. The format of these enumerations is "*MD_MAX72XX::{enumeration}*". Used enumerations are:

TSL - Shift Left for one pixel;
TSR - Shift Right for one pixel
TSU - Shift Up for one pixel;
TSD - Shift Down for one pixel
TFLR - Flip Left to Right;
TFUD - Flip Up to Down
TRC - Rotate Clockwise 90 degrees
TINV - Invert pixels (all pixels inverted, those who were turned ON, are turned OFF, and vice versa).

After this, we use `setBuffer()` function to send the data for displaying to the MAX7219 chip. This function accepts three arguments, the first and the second arguments are the *X* and *Y* positions of the image, and the third is bitmap image data.

To display data we use the following line of the code:

```
mx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::ON);
```

Then we animate displayed data. We do that with *`WRAPAROUND`* property of the *`control()`* function and *`transform()`* function. We use *`for`* loop to loop through all of enumerations in the "*t*" array.

The last function is called *`showCharsset()`* which is used to show all UNICODE characters that can be used on LED matrix screen. Here we use built in function *`update()`* which acts like *`control()`* function with property *`UPDATE`*. *`update()`* function accepts one argument, which value can be on of the two: *`MD_MAX72XX::OFF`* and *`MD_MAX72XX::ON`* First we set the argument value to *`MD_MAX72XX::OFF`*, than make changes, change the character that will be displayed, and then change the argument value to *`MD_MAX72XX::ON`*. We use *`for`* loop to loop through all UNICODE characters.

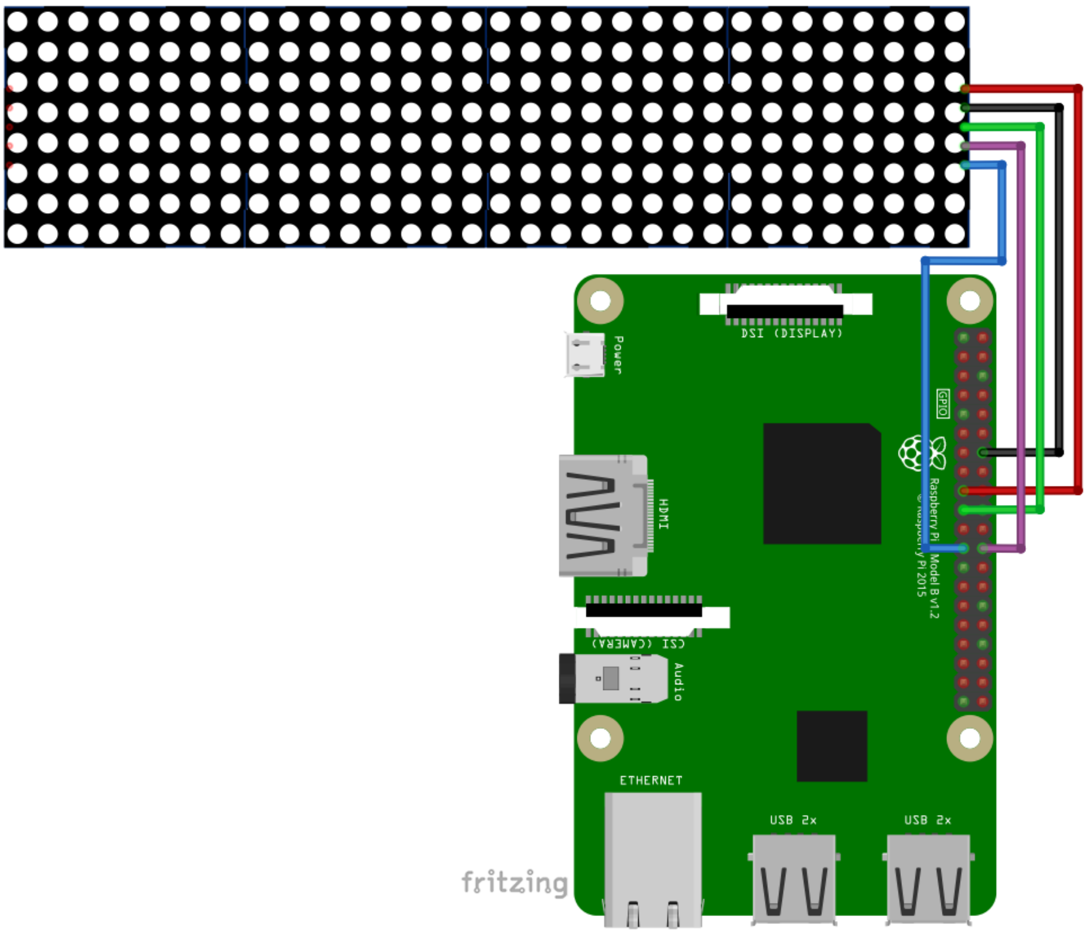
In the *`setup()`* function we initialize screen object with the following line of the code: `mx.begin()`

In the *`loop()`* function we call all functions that we created, one by one.

If you want to know more about these functions from the "*MD_MAX72XX.h*" library, you can go to the library official documentation site:
https://majicdesigns.github.io/MD_MAX72XX/class_m_d___m_a_x72_x_x.html

Connecting the display with Raspberry Pi

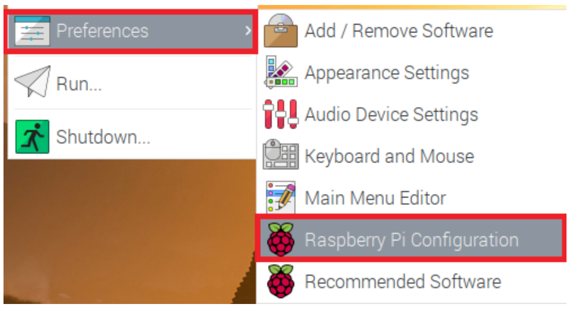
Connect the module with the Raspberry Pi as shown on the following connection diagram:



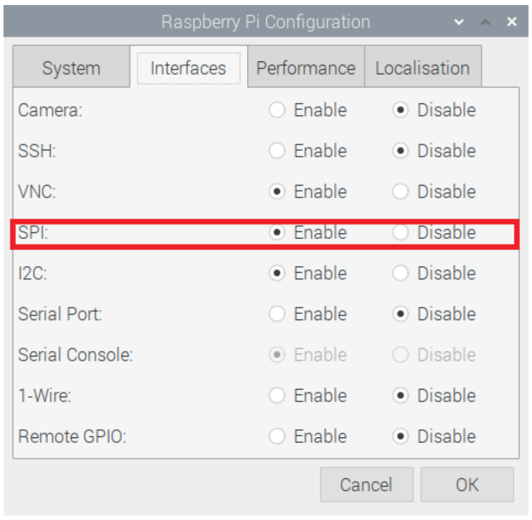
Module pin		**>**	**Raspberry Pi pin**		
VCC		>	5V		Red wire
GND		>	GND		**Black Wire**
DATA	(MOSI)	>	GPIO10	[pin 19]	Blue Wire
CLK	(SCK)	>	GPIO11	[pin 23]	Brown Wire
CS	(SS)	>	GPIO8	[pin 24]	Green Wire

Enabling SPI Interface

In order to use the module with Raspberry Pi, first we have to enable SPI interface in the Raspbian. To do so, go to:
Preferences > Raspberry Pi Configuration
as shown on the image below:



Next, open the *"Interfaces"* tab, locate the *"SPI"* radio button and enable it, as shown on the following image:



Camera:	Enable	[✓ Disable]
SSH:	Enable	[✓ Disable]
VNC:	[✓ Enable]	Disable
SPI:	[✓ Enable]	Disable
I2C:	[✓ Enable]	Disable
Serial Port:	Enable	[✓ Disable]
Serial Console:	[✓ Enable]	Disable
1-Wire:	Enable	[✓ Disable]
Remote GPIO:	Enable	[✓ Disable]

You will be prompted to reboot the system. Reboot it.

Libraries and tools for Python

To use the module with the Raspberry Pi it is recomended to download external library for it. The library we are going to use is called "*luma.led_matrix*". For the library it is necessary to install the dependencies for first. To do so open the terminal and the following commands:

```
sudo apt install build-essential python-dev
sudo apt install python-pip libfreetype6-dev
sudo apt install libjpeg-dev libopenjp2-7 libtiff5
```

We have to make *"pip"* and *"setuptools"* up to date, and to do so, run the following command:

```
sudo -H pip install --upgrade --ignore-installed pip setuptools
```

Next, install the latest version of the "*luma.led_matrix*" library by running the following command:

```
sudo -H pip install --upgrade luma.led_matrix
```

Python Script:

```
import re
import time
import argparse
from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.core.legacy import text, show_message
from luma.core.legacy.font import proportional, CP437_FONT, TINY_FONT,
SINCLAIR_FONT, LCD_FONT

def demo(n, block_orientation, rotate, inreverse):

    serial = spi(port=0, device=0, gpio=noop())
    device = max7219(serial, cascaded=n or 1,
                  block_orientation=block_orientation, rotate=rotate or 0,
                  blocks_arranged_in_reverse_order=inreverse)
    print('Created device')
    msg = 'MAX7219 LED Matrix Demo'
    print(msg)
    show_message(device, msg, fill='white',font=proportional(CP437_FONT),
                                             scroll_delay=0)
    time.sleep(1)
    print('Vertical scrolling')
    words = ['Victor', 'Echo', 'Romeo', 'Tango', 'India', 'Charlie',
              'Alpha', 'Lima', ' ', 'Sierra', 'Charlie', 'Romeo', 'Oscar',
              'Lima', 'Lima', 'India', 'November', 'Golf', ' ']

    virtual = viewport(device, width=device.width, height=len(words) * 8)
    with canvas(virtual) as draw:
        for i, word in enumerate(words):
            text(draw, (0, i * 8), word,     fill='white',
                                             font=proportional(CP437_FONT))
```

```
# one tab
    for i in range(virtual.height - device.height):
        virtual.set_position((0, i))
        time.sleep(0.05)

    msg = 'Brightness'
    print(msg)
    show_message(device, msg, fill='white')
    time.sleep(1)

    for _ in range(5):
        for intensity in range(16):
            device.contrast(intensity * 16)
            time.sleep(1)

    device.contrast(0x80)
    time.sleep(1)
    msg = 'Alternative font!'
    print(msg)
    show_message(device, msg, fill='white', font=SINCLAIR_FONT)
    time.sleep(1)
    msg = 'Proportional font - characters are squeezed together!'
    print(msg)
    show_message(device,msg,fill='white',font=proportional(SINCLAIR_FONT))
    time.sleep(1)
    msg = 'Tiny is, I believe, the smallest possible font \
    (in pixel size). It stands at a lofty four pixels \
    tall (five if you count descenders), yet it still \
    contains all the printable ASCII characters.'
    msg = re.sub(' +', ' ', msg)
    print(msg)
    show_message(device, msg, fill='white', font=proportional(TINY_FONT))
    time.sleep(1)
```

```
# one tab
    msg = 'CP437 Characters'
    print(msg)
    show_message(device, msg)
    time.sleep(1)
    for x in range(256):
        with canvas(device) as draw:
            text(draw, (0, 0), chr(x), fill='white')
            time.sleep(0.3)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='matrix_demo arguments',
                    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--cascaded', '-n', type=int, default=1,
                    help='Number of cascaded MAX7219 LED matrices')
    parser.add_argument('--block-orientation', type=int, default=0,
                    choices=[0, 90, -90],
                    help='Corrects block orientation when wired vertically')
    parser.add_argument('--rotate',type=int,default=0,choices=[0, 1, 2, 3],
                    help='Rotate display 0=0°, 1=90°, 2=180°, 3=270°')
    parser.add_argument('--reverse-order', type=bool, default=False,
                    help='Set to true if blocks are in reverse order')
    args = parser.parse_args()
    try:
        while True:
        demo(args.cascaded,args.block_orientation,
                                   args.rotate,args.reverse_order)
        time.sleep(1)

    except KeyboardInterrupt:
        print('Script end!')
```

The code is based on the code from the following link:

https://github.com/rm-hull/luma.led_matrix/blob/master/examples/matrix_demo.py

The script starts with the importing necessary libraries. Next, we create the function *demo()* which is used to run all functions that come with "*luma.led_matrix*" library. The function accepts four arguments and returns no value. The arguments are:

» *n* - which represents the number of cascaded LED matrix segments

» *block_orientation* - which is used to correct orientation of content on the screen; its values are: *0*, *90* and *-90*

» *rotate* - which rotates the content on the screen; its values are: *0*, *1*, *2* and *3*, which represent angles *0°*, *90°*, *180°* and *270°*, respectively

» *reverse* - is a boolean value and when it is set to *true*, the segments order is in reverse.

Inside the function, an SPI communication object and a device object are created.

To display a scrolling message (horizontally) on the screen we used *show_message()* function that comes with the "*luma.led_matrix*" library. This function accepts five arguments and returns no value. The last three arguments are optional. The first argument is a device object, the second is the message, the third is the *fill* argument, the fourth is the *font* argument and the fifth is the *scroll_delay* argument. The value of the *fill* argument is the name of the color: "*white*", "*blue*", "*red*", etc. If you use any other color other than "*black*" the message will be displayed in red, because the color red is the color of LEDs onboard the screen.

If you put "*black*" to the fill argument, nothing would be displayed.

The *`font`* argument is for setting the specific font and the *`scroll_delay`* argument is for changing the speed of scrolling text.

In order to display scrolling message vertically you have to use *`viewport`* library function. These library functions are used to create an object, or a canvas, with message content, then the content of this canvas is moved over the screen via *`for`* loop giving vertical scrolling effect.

To change the brightness level, we use the *`contrast()`* function, like in the following line of the code: `device.contrast(number)`
where the *`number`* is the level of brightness. There are *`16`* different brightness level values, but this value has to be a number which when divided by *`16`* returns integer number, because MAX7219 driver chip internal registers are set in this manner.

There are two ways of using the fonts. First is to put the font name to font argument of the *`show_message()`* function. The second is to put the *`proportional()`* function to the same argument. This function accepts one argument, which value is the font name. The difference is that with *`proportional()`* function font is displayed in proportion to the screen size.

To display one specific character on the one segment of the screen we use *text()* function. This function accepts four arguments and returns no value. First argument is a draw object. This object contains one part of canvas, or piece of data from canvas object. We explained canvas object for scrolling message vertically. The second argument is a list containing two elements, *X* and *Y* position of the top left corner of the character. The third argument is the character that will be displayed. The fourth argument is the *fill* argument.

Save the script by the name "*ledMatrix.py*". To run it, open terminal into directory where you saved the script, and run the next command:

```
python3 ledMatrix.py
```

There is the option to send four arguments to the script when we run it. These arguments are optional. This is possible because of used library we included, called argparse. There are four optional arguments that you can send to the script:

» *cascaded* - represents the number of cascaded LED matrix segments

» *block-orientation* - used to correct orientation of the segments

» *rotate* - used to rotate the content on the screen

» *reverse-order* - used to change the scrolling direction of the content

These arguments are used when we call the *demo()* function. The *demo()* function is called in the infinity loop block (*while True:*).

The infinity loop block is in the `try-except` block. We use `try-except` block to wait for keyboard interrupt. The keyboard interrupt happens when we press *CTRL + C* on the keyboard, and this stops the script.

Example of using all the arguments when we run the script:

```
python3 ledMatrix.py --cascaded 4 --block-orientation 90
                     --rotate 2   --reverse-order false
```

You've done it!

Now you can use your module for various projects.

Now it is time to learn and make the Projects on your own. You can do that with the help of many example scripts and other tutorials, which you can find on the internet.

If you are looking for the high quality microelectronics and accessories, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.

https://az-delivery.de

Have Fun!

Impressum

https://az-delivery.de/pages/about-us