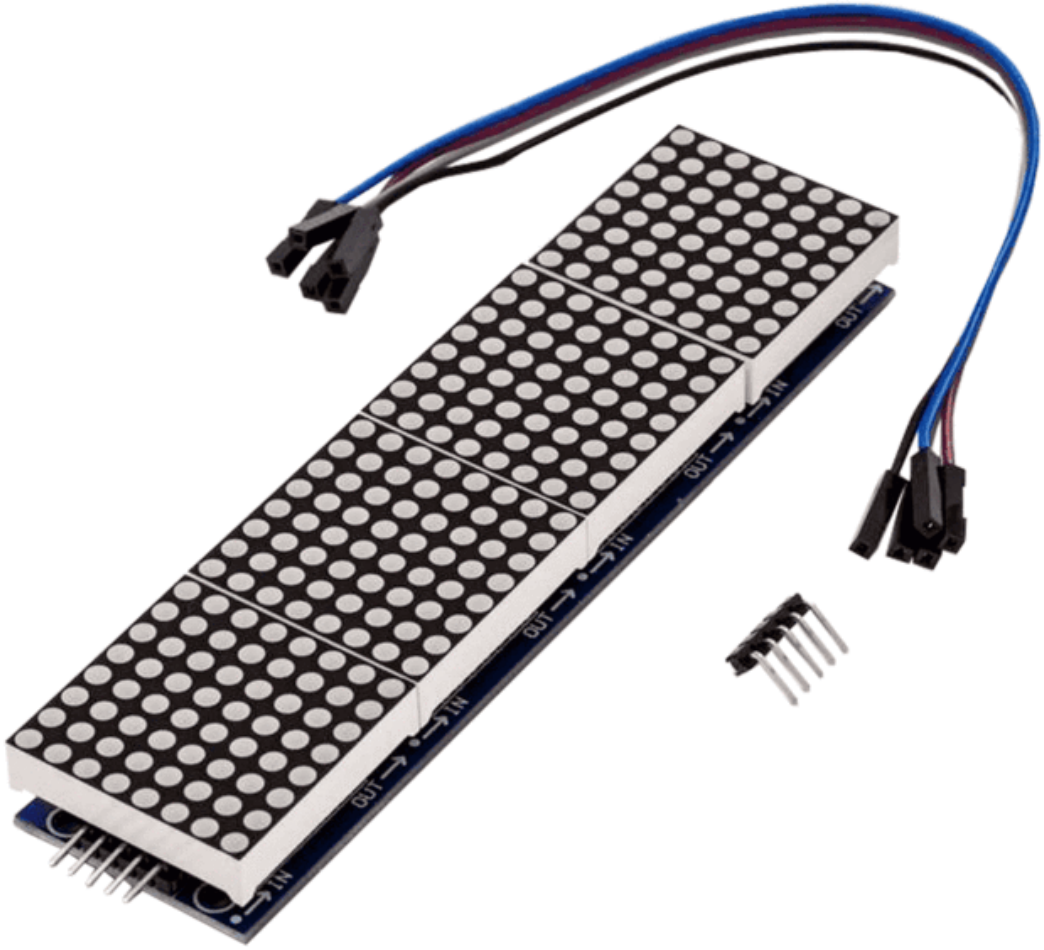
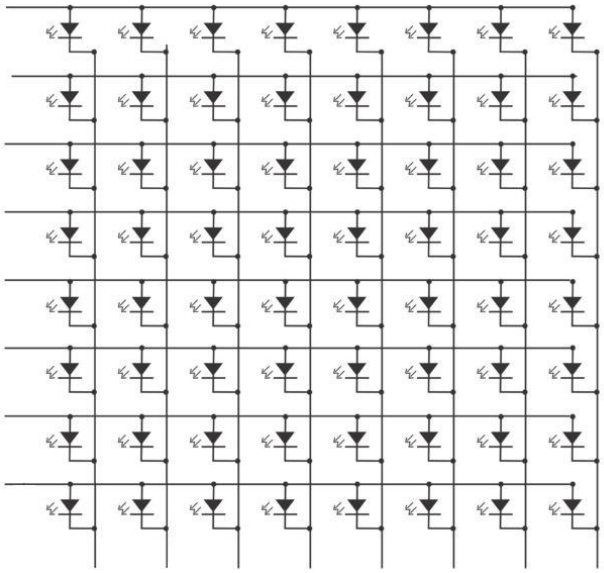
Bienvenue !

Nous vous remercions d'avoir acheté notre module d'affichage matriciel LED 4x64 AZ-Delivery. Dans les pages qui suivent, nous vous expliquerons comment utiliser et configurer cet appareil pratique.

Amusez-vous bien !



L'écran matriciel 4x64 LED est un écran avec des réseaux de LED empilés un par un dans une "matrice". L'écran 4x64 LED est composé de quatre écrans 8x8 plus petits, appelés "segments". Ils sont appelés 8x8 parce qu'il y a 8 LED dans une rangée, et il y a 8 rangées dans un segment, ce qui crée une matrice de 64 LED (64 pixels). Chaque LED a besoin de deux fils pour fonctionner (un pour l'alimentation et un pour la masse). Pour éviter de devoir câbler les 64 LED une par une, elles sont connectées dans une matrice de LED, comme le montre l'image ci-dessous :



Mettre les *LEDs* dans une matrice réduit le nombre de broches de sortie à 16, ce qui est encore trop, c'est pourquoi les drivers de matrice *LED* ont été inventés. Dans notre cas, le driver de matrice *LED* utilisé est appelé *"MAX7219"*, qui utilise l'interface *SPI*. L'écran matriciel *LED* 4x64 a quatre segments, et une puce de pilotage par segment. Les puces pilotes sont connectées en série dans la *"Daisy chain"*. Il est possible d'en empiler plus, mais vous devez utiliser une alimentation externe.

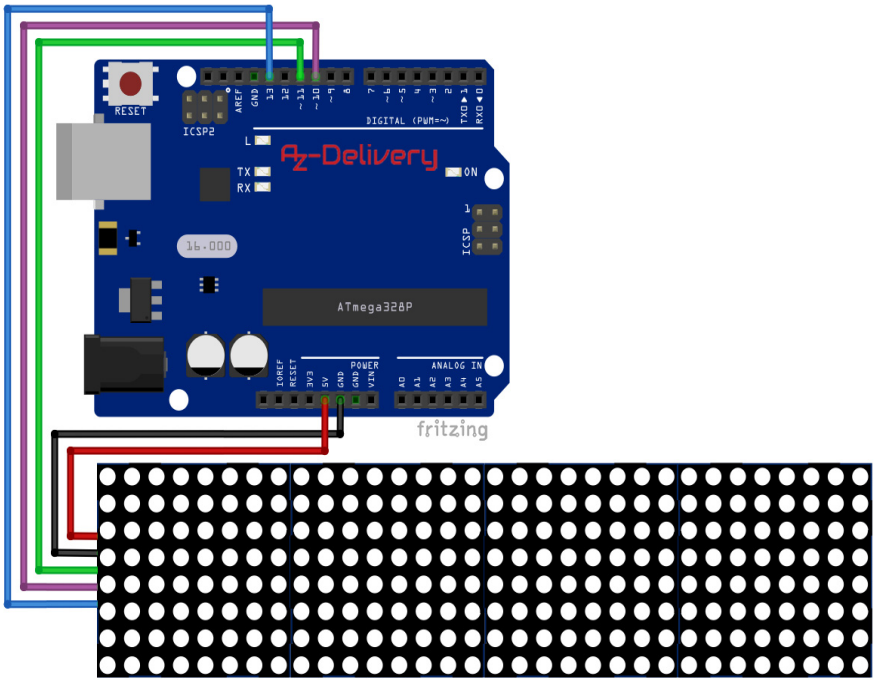
La puce *MAX7219* utilise l'interface *SPI* pour communiquer avec les microcontrôleurs. *SPI* est une interface de communication périphérique série *synchrone*, et elle est utilisée pour la communication à courte distance, principalement dans les systèmes embarqués. *Synchrone* signifie que les données sont envoyées et reçues dans un cycle d'horloge spécifique, et série signifie que les données sont envoyées en série, bit par bit.

Caractéristiques :

» Gamme de tension d'alimentation et de tension logique : de +3,3V à +5V DC
» Couleur des LEDs : Rouge
» Contrôle de la luminosité : numérique et analogique
» Dimensions : 32 x 129mm [1.26 x 5.1in]
» Interface SPI 10MHz
» Contrôle individuel des segments de LED
» L'écran s'assombrit lorsqu'il est allumé
» Écrans LED à cathode commune

Connecter l'écran avec l'Atmega328P Board

Connectez le module avec l'Atmega328P Board comme indiqué sur le schéma de connexion suivant :

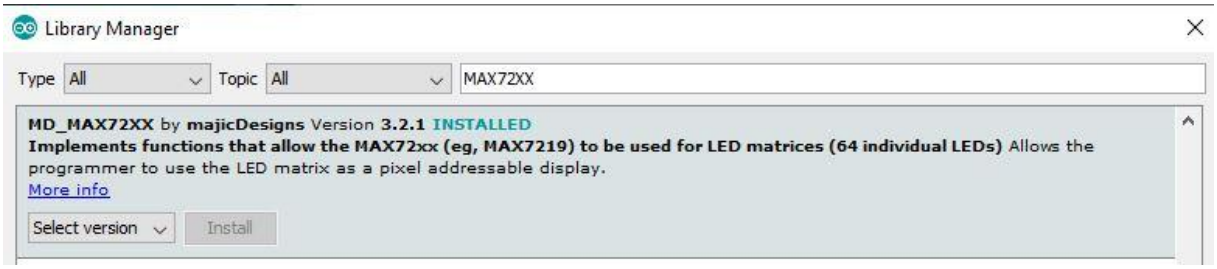


Pin du module		> Board pin		
VCC		>	5V	**Câble Rouge**
GND		>	GND	**Câble Noir**
CS	(SS)	>	D10 pin	**Câble Violet**
DATA	(MOSI)	>	D11 pin	**Câble Vert**
CLK	(SCK)	>	D13 pin	**Câble Bleu**

Assurez-vous de lire la désignation des broches avant de commencer à les connecter, car il existe différentes versions du module. Nous avons inclus des noms de broches alternatifs à l'intérieur des parenthèses.

Bibliothèque de l'IDE Arduino

Pour utiliser le module avec l'Atmega328P Board, nous vous recommandons de télécharger la bibliothèque externe correspondante. Tout d'abord, allez dans Tools > Manage Libraries, et quand une nouvelle fenêtre pop-up apparaît, tapez *"MAX72XX"* dans la boîte de recherche. La bibliothèque que nous allons utiliser s'appelle *"MD_MAX72XX"* et a été créée par *"majicDesigns"*, comme le montre l'image ci-dessous :



L'installation est aussi simple que de cliquer sur le bouton *"Install"* qui apparaît lorsque vous passez votre souris sur le résultat de la recherche.

Il existe de nombreuses versions de ces écrans. Afin de détecter celle que vous utilisez, vous pouvez exécuter le *sketch Hardware Mapper* qui est fourni avec la bibliothèque. Pour ouvrir le sketch, allez sur :
File > Examples > MD_MAX72XX > MD_MAX72xx_HW_Mapper,
et modifiez la ligne suivante du code:

```
#define SERIAL_SPEED 57600
```

en :

```
#define SERIAL_SPEED 9600
```

Téléchargez le sketch sur l'Atmega328P Board et ouvrez le Serial Monitor (Tools > Serial Monitor). Ensuite, suivez les instructions en trois étapes que l'Atmega328P Board envoie au Serial Monitor, comme le montre l'image de la page suivante :

```
[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.
```

Pour obtenir le bon résultat, vous devez orienter l'écran comme sur le schéma de connexion. Le côté droit de l'écran est le côté où se trouvent les broches d'entrée de l'écran (où nous avons connecté les fils).

La carte matérielle résultante est *"FC16_HW"*. Par conséquent, lorsque vous utilisez le module d'affichage matriciel *AZ-Delivery 64 LED*, vous devez définir la macro *"HARDWARE_TYPE"* sur cette valeur de résultat.

Code du schéma :

```
#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);
}
```

L'esquisse commence par l'inclusion de la bibliothèque *"MD_MAX72XX.h"*, puis se poursuit avec plusieurs définitions de macros. Un exemple particulier d'utilisation d'une des macros est *Serial.print(F(x))*. *F(x)* est une macro qui stocke la chaîne *"x"* dans la mémoire flash, et non dans la *SRAM*. La mémoire flash est une mémoire dans les microcontrôleurs où les programmes sont stockés, et la mémoire *SRAM* est un type de mémoire *RAM*. Nous diminuons la limite de la mémoire pour nos programmes en utilisant la macro *F(x)*, ce qui augmente la mémoire *SRAM* (très importante pour l'exécution des programmes sur les microcontrôleurs). Si vous n'utilisez pas la macro *F(x)*, il est fort probable que votre programme ne fonctionnera pas comme prévu. En effet, dans le sketch, il y a trop de chaînes de caractères, qui sont des tableaux de caractères dans le langage de programmation *C* et qui consomment facilement la mémoire *SRAM*.

Ensuite, on précise l'écran que l'on utilise, en donnant à la macro *HARDWARE_TYPE* la valeur *FC16_HW* (la valeur du résultat que nous avons obtenu dans le chapitre précédent).

Ensuite, on indique le nombre de segments de l'écran matriciel *LED* en fixant la macro *MAX_DEVICES* à *4*. Si vous connectez plusieurs écrans matriciels *LED* dans la chaîne *Daisy*, vous devez modifier cette valeur en conséquence.
Il y a trois autres macros qui définissent les broches de l'interface *SPI*.

Après cela, nous créons l'objet *"mx",* l'objet *écran* qui est utilisé dans tout le sketch.

Après tout cela, de nombreuses fonctions sont créées.

La première fonction est appelée *scrollingText()*, qui est utilisée pour faire défiler le texte à l'écran. La fonction *scrollingText()* utilise la fonction de la bibliothèque *"MD_MAX72XX.h"* appelée *setColumn()*. Cette fonction permet d'activer ou de désactiver une colonne de *LED* à l'écran. Le reste de la fonction est un algorithme pour l'effet de défilement. On prend la chaîne *"p"* de la mémoire flash, et on la convertit en un tableau d'entiers non signés qui représente les *LEDs* dans le tableau de colonnes. Ensuite, nous affichons ce tableau sur l'écran. A la fin de la fonction, on spécifie le temps de retard, qui est la vitesse de défilement du texte.

La deuxième fonction est appelée *rows()*, qui utilise la fonction *setRow()* de la bibliothèque *"MD_MAX72XX.h"*. La fonction *setRow()* accepte deux arguments. Le premier est la ligne qui sera utilisée, où *0* signifie la première ligne en commençant par le haut. Le deuxième argument est un nombre en hexadécimal, qui représente la valeur *8 bits* pour la rangée de pixels (8 pixels dans une rangée). Dans ce cas, nous utilisons *0xFF* qui active tous les pixels d'une rangée spécifique.

La troisième fonction s'appelle *columns()*, c'est la même chose que *rows()* mais dans ce cas, nous activons les tableaux de colonnes de pixels. La fonction utilise la fonction *setColumn()* de la bibliothèque *"MD_MAX72XX.h".* La fonction *setColumn()* est la même que *setRow()* mais uniquement pour les colonnes.

La quatrième fonction est appelée *stripe()*, qui affiche une bande diagonale défilante (de 4 pixels de large) sur l'ensemble de l'écran. Cette fonction utilise la commande *setPoint()* de la bibliothèque *"MD_MAX72XX.h"*. La fonction *setPoint()* est utilisée pour activer une *LED* spécifique sur l'écran et accepte trois arguments. Les premier et deuxième arguments sont les positions *X* et *Y* de la *LED* (le coin supérieur gauche de l'écran est X=0 et Y=0, le coin inférieur droit de l'écran est X=31 et Y=7). Le troisième argument est une valeur booléenne représentant l'état de la *LED*, allumée = vrai, et éteinte = faux. À la fin de la fonction, nous spécifions le temps de retard, qui est utilisé pour changer la vitesse de la bande de défilement.

La cinquième fonction, appelée *spiral()*, affiche une ligne en forme de serpent qui transforme tous les pixels de l'écran en un effet de mouvement en spirale. Cette fonction utilise la fonction *setPoint()* de la bibliothèque *"MD_MAX72XX.h"*. Le reste de la fonction est l'algorithme qui consiste à allumer les *LED* les unes après les autres jusqu'à ce que toutes les *LED* soient allumées.

La sixième fonction s'appelle *bounce()* et affiche un point mobile qui rebondit sur les bords de l'écran matriciel *LED*. Cette fonction utilise quelques fonctions que nous avons déjà expliquées, et une fonction supplémentaire *getColumnCont()* qui renvoie le nombre de colonnes sur l'écran matriciel *LED*. Le reste de la fonction est l'algorithme de déplacement du pixel.

La septième fonction, appelée *intensity()*, est utilisée pour allumer uniquement la première rangée de *LED* de l'écran. La fonction utilise la fonction *control()* de la bibliothèque *"MD_MAX72XX.h"*. La fonction *control()* accepte deux arguments, le premier définit la propriété de l'objet *"mx"* que nous allons modifier, où nous stockons la valeur *"MD_MAX72XX::INTENSITY"*. Nous allons donc modifier l'intensité de la luminosité. Le deuxième argument est un nombre qui représente le niveau d'intensité et sa valeur est comprise entre *0* et *15*, soit *16* niveaux différents (la valeur par défaut est 8).

La huitième fonction est appelée *transformation()*, qui est utilisée pour animer l'image bitmap sur l'écran. Les données de l'image bitmap sont stockées dans un tableau d'entiers non signés. Ce tableau contient 8 éléments, qui contiennent des nombres binaires. Ces nombres binaires représentent les rangées d'un segment de l'écran, où la valeur binaire 0 représente une *LED* éteinte, et la valeur *1* représente une *LED* allumée. On a défini un autre tableau appelé *"t"*, qui contient des énumérations utilisées pour l'animation. Le format de ces énumérations est *"MD_MAX72XX::{enumération}"*. Les énumérations utilisées sont :

TSL - Déplacement vers la gauche d'un pixel ; TSR - Déplacement vers la droite pour un pixel

TSU - Déplacement vers le haut pour un pixel ; TSD - Déplacement vers le bas pour un pixel

TFLR - Tourner de gauche à droite ; TFUD - Basculer de haut en bas

TRC - Rotation de 90 degrés dans le sens horaire.

TINV - Inverser les pixels (tous les pixels sont inversés, ceux qui étaient allumés sont éteints, et vice versa).

Ensuite, nous utilisons la fonction *setBuffer()* pour envoyer les données à afficher à la puce *MAX7219*. Cette fonction accepte trois arguments, le premier et le deuxième sont les positions *X* et *Y* de l'image, et le troisième sont les données de l'image bitmap.

Pour afficher les données, on utilise la ligne de code suivante :
```
mx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::ON);
```

Ensuite, on anime les données affichées. Nous le faisons avec la propriété *WRAPAROUND* de la fonction *control()* et de la fonction *transform()*. On utilise la *for loop* pour parcourir toutes les énumérations du tableau *"t"*.

La dernière fonction est appelée *showCharsset()* qui est utilisée pour afficher tous les caractères *UNICODE* qui peuvent être utilisés sur l'écran de la matrice *LED*. Ici, on utilise la fonction intégrée *update()* qui agit comme la fonction *control()* avec la propriété *UPDATE*. La fonction *update()* accepte un argument, dont la valeur peut être l'une des deux : *MD_MAX72XX::OFF et MD_MAX72XX::ON.*
Tout d'abord, nous définissons la valeur de l'argument à *MD_MAX72XX::OFF,* puis nous effectuons des modifications, changeons le caractère qui sera affiché, et enfin changeons la valeur de l'argument à *MD_MAX72XX::ON.* On utilise la for loop pour parcourir tous les caractères *UNICODE.*

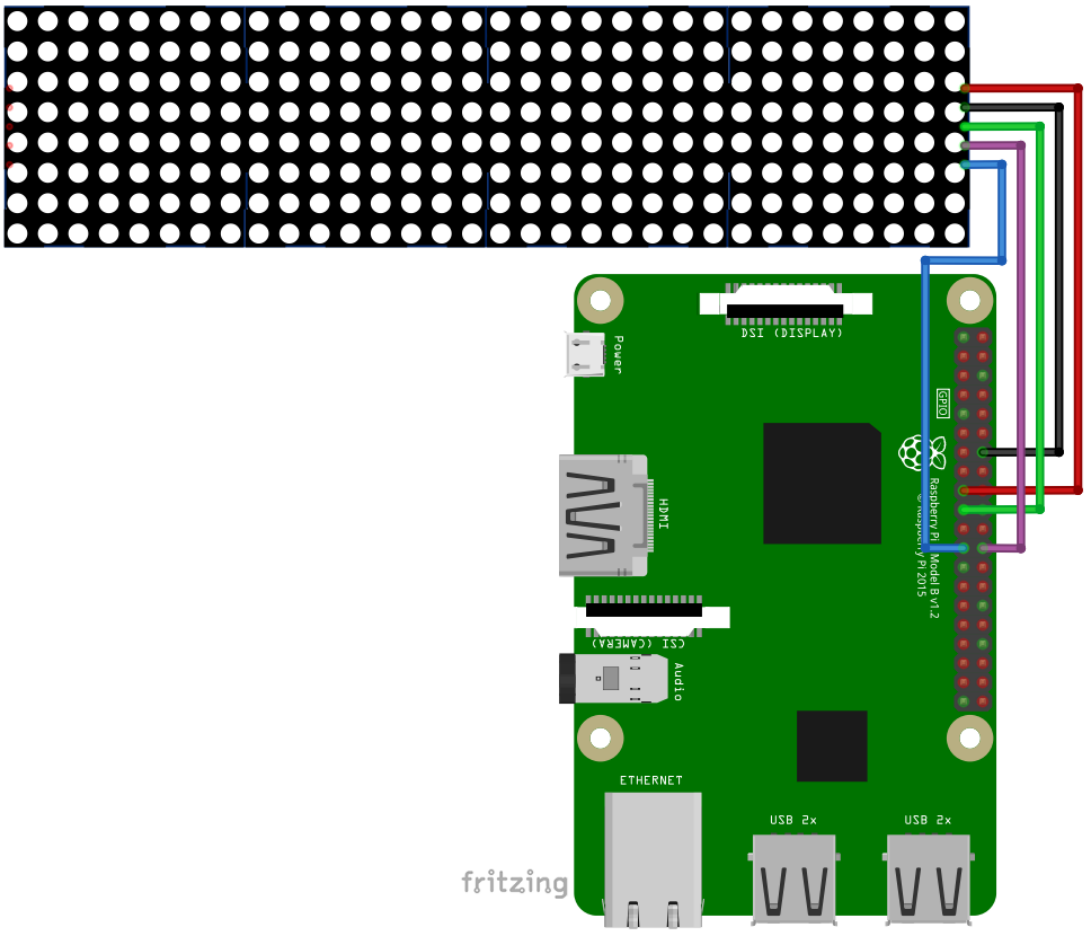
Dans la fonction *setup()* on initialise l'objet *écran* avec la ligne de code suivante : *mx.begin()*

Dans la fonction *loop()*, nous appelons toutes les fonctions que nous avons créées, une par une.

Si vous souhaitez en savoir plus sur ces fonctions de la bibliothèque *"MD_MAX72XX.h"*, vous pouvez consulter le site de documentation officiel de la bibliothèque :
https://majicdesigns.github.io/MD_MAX72XX/class_m_d___m_a_x72_x_x.html

Connexion de l'écran avec Raspberry Pi

Connectez le module avec le *Raspberry Pi* comme indiqué sur le schéma de connexion suivant :

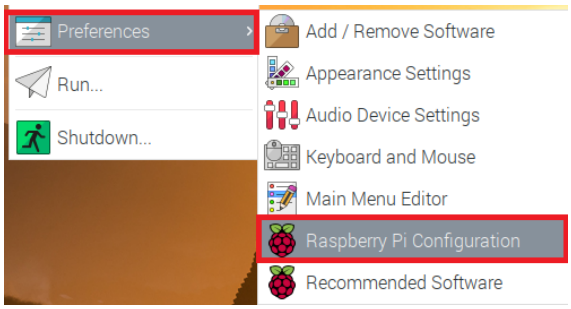


Pin du module > Pin du Raspberry Pi

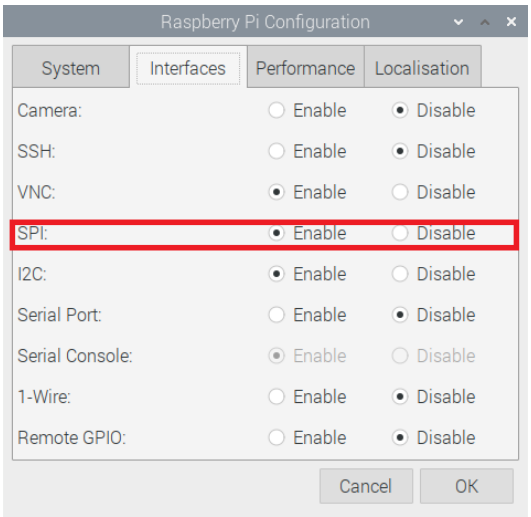
VCC		>	5V		**Câble Rouge**
GND		>	GND		**Câble Noir**
DATA	(MOSI)	>	GPIO10	[pin 19]	**Câble Bleu**
CLK	(SCK)	>	GPIO11	[pin 23]	**Câble Marron**
CS	(SS)	>	GPIO8	[pin 24]	**Câble Vert**

Activation de l'interface SPI

Afin d'utiliser le module avec Raspberry Pi, nous devons d'abord activer l'interface SPI dans Raspbian. Pour ce faire, allez à : *Preferences* > *Raspberry Pi Configuration*
comme le montre l'image ci-dessous :



Ensuite, ouvrez l'onglet *"Interfaces"*, localisez le bouton radio *"SPI"* et activez-le, comme indiqué sur l'image suivante :



Il vous sera demandé de redémarrer le système. Redémarrez-le.

Bibliothèques et outils pour Python

Pour utiliser le module avec le *Raspberry Pi,* il est recommandé de télécharger une bibliothèque externe. La bibliothèque que nous allons utiliser s'appelle *"luma.led_matrix".* Pour la bibliothèque, il faut d'abord installer les dépendances. Pour ce faire, ouvrez le terminal et tapez les commandes suivantes :

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

Nous devons mettre à jour *"pip"* et *"setuptools"*, et pour ce faire, exécuter la commande suivante :

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

Ensuite, installez la dernière version de la bibliothèque "luma.led_matrix" en exécutant la commande suivante :

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

Script Python :

```
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!')
```

Le code est basé sur le code du lien suivant :
https://github.com/rm-hull/luma.led_matrix/blob/master/examples/matrix_demo.py

Le script commence par l'importation des bibliothèques nécessaires. Ensuite, on crée la fonction *demo()* qui est utilisée pour exécuter toutes les fonctions qui viennent avec la bibliothèque *"luma.led_matrix".* La fonction accepte quatre arguments et ne renvoie aucune valeur. Les arguments sont les suivants :

» `n` - qui représente le nombre de segments de matrice LED en série.

» `block_orientation` - qui est utilisé pour corriger l'orientation du contenu sur l'écran ; ses valeurs sont : 0, 90 et -90

» `rotate` - qui fait pivoter le contenu sur l'écran ; ses valeurs sont : 0, 1, 2 et 3, qui représentent les angles 0°, 90°, 180° et 270°,

» `reverse` - est une valeur booléenne et lorsqu'elle est définie comme vraie, l'ordre des segments est inversé.

A l'intérieur de la fonction, un objet de communication *SPI* et un objet de dispositif sont créés.

Pour afficher un message défilant (horizontalement) sur l'écran, nous avons utilisé la fonction *show_message()* fournie avec la bibliothèque *"luma.led_matrix".* Cette fonction accepte cinq arguments et ne renvoie aucune valeur. Les trois derniers arguments sont facultatifs. Le premier argument est un objet de périphérique, le deuxième est le message, le troisième est l'argument *fill*, le quatrième est l'argument *font* et le cinquième est l'argument *scroll_delay.* La valeur de l'argument *fill* est le

nom de la couleur : *"blanc", "bleu", "rouge"*, etc. Si vous utilisez une autre couleur que *"noir"*, le message sera affiché en rouge, car la couleur rouge est la couleur des LEDs de l'écran.

Si vous mettez *"noir"* dans l'argument de *fill*, rien ne sera affiché.

L'argument *font* permet de définir la police spécifique et l'argument *scroll_delay* permet de modifier la vitesse de défilement du texte.

Pour afficher un message défilant verticalement, vous devez utiliser la fonction de bibliothèque viewport. Ces fonctions de bibliothèque sont utilisées pour créer un objet, ou un tableau, avec le contenu du message, puis le contenu de ce tableau est déplacé sur l'écran via une for loop, ce qui donne un effet de défilement vertical.

Pour modifier le niveau de luminosité, nous utilisons la fonction *contrast(),* comme dans la ligne de code suivante : *device.contrast(number)*
où le nombre est le niveau de luminosité. Il existe *16* valeurs de niveau de luminosité différentes, mais cette valeur doit être un nombre qui, lorsqu'il est divisé par *16,* renvoie un nombre entier, car les registres internes de la puce pilote *MAX7219* sont définis de cette manière.

Il y a deux façons d'utiliser la police. La première est de mettre le nom dans l'argument font de la fonction *show_message().* La seconde est de mettre la fonction *proportional()* dans le même argument. Cette fonction accepte un argument, dont la valeur est le nom de la police. La

différence est qu'avec la fonction *proportional()* la police est affichée en proportion de la taille de l'écran.

Pour afficher un caractère spécifique sur un segment de l'écran, nous utilisons la fonction *text()*. Cette fonction accepte quatre arguments et ne renvoie aucune valeur. Le premier argument est un objet de dessin. Cet objet contient une partie du tableau, ou un morceau de données de l'objet *tableau*. Nous avons expliqué l'objet *tableau* pour faire défiler le message verticalement. Le deuxième argument est une liste contenant deux éléments, la position *X* et *Y* du coin supérieur gauche du caractère. Le troisième argument est le caractère qui sera affiché. Le quatrième argument est l'argument de remplissage.

Sauvegardez le script sous le nom de *"ledMatrix.py"*. Pour l'exécuter, ouvrez un terminal dans le répertoire où vous avez enregistré le script, et exécutez la commande suivante :

```
python3 ledMatrix.py
```

Il est possible d'envoyer quatre arguments au script lorsque nous l'exécutons. Ces arguments sont facultatifs. Ceci est possible grâce à la bibliothèque que nous avons incluse, appelée argparse. Il y a quatre arguments optionnels que vous pouvez envoyer au script :

» `cascaded - représente le nombre de segments de la matrice de LED en cascade`

» `block-orientation` - utilisé pour corriger l'orientation des segments

» `rotate` - utilisé pour faire tourner le contenu sur l'écran

» *reverse-order* - utilisé pour changer la direction de défilement du contenu

Ces arguments sont utilisés lorsque nous appelons la fonction *demo().* La fonction *demo()* est appelée dans le bloc *infinity loop (while True :).*

Le bloc *infinity loop* est dans le bloc *try-except*. Nous utilisons le bloc *try-except* pour attendre l'interruption du clavier. L'interruption du clavier se produit lorsque nous appuyons sur *CTRL* + *C* sur le clavier, ce qui arrête le script.

Exemple d'utilisation de tous les arguments lorsque nous exécutons le script : **`python3 ledMatrix.py --cascaded 4 --block-orientation 90 --rotate 2 --reverse-order false`**

Ça y est, vous avez réussi !
Vous pouvez maintenant utiliser votre module pour différents projets.

Il est maintenant temps d'apprendre et de réaliser les projets par vous-même. Vous pouvez le faire à l'aide de nombreux exemples de scripts et d'autres didacticiels, que vous trouverez sur Internet.

Si vous recherchez microélectronique et accessoires de haute qualité, AZ-Delivery Vertriebs GmbH est l'entreprise idéale pour vous les procurer. Vous recevrez de nombreux exemples d'application, des guides d'installation complets, des livres électroniques, des bibliothèques et l'assistance de nos experts techniques.

https://az-delivery.de

Amusez-vous !

Mentions légales

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