屏東縣第 62 屆國中小學科學展覽會
作品說明書

科　　別：生活與應用科學科(一)

組　　別：國中組

作品名稱：天啊！字體變大了

關 鍵 詞：點陣字、**Arduino**、**LED**

編號：**B6009**

摘要

本研究乃將升將資訊課程中 Arduino 的教學課程，作進一步探討，與當中內建的 LedControl、MaxMatrix 強大模組併用，製成日常可見的 LED 跑馬燈。

同時，本研究乃利用 8*8 共陰極矩陣 LED、配合 MAX7219 整合晶片模組，透過 Arduino 控制顯示繁體中文字，在模擬顯示點陣中文字體時，運用了 Pytthon 抓取 Excel 表格化產生位元陣列，提升寫程式的速度及方便性。再進一步，串聯 4 個 4 合一 Max7219 的 8*8Led 矩陣，共 16 個元件，以顯示 4 個 16*16 大小的點陣中文字。

壹、 研究動機

於當今的資訊課程中，LED 版面配控常為其實作課程之一主要單元。若能將此技能加以進階使用並融入平常生活中，可謂學以致用。

舉凡商店的霓虹招牌、掛在醫院和學校門口上的字幕版，抑或是位於十字路口的廣告大看板，皆多由這些微小的發光二極體，點點排列而成。

因此，經過多次編輯與改良，我們研究了平常可見的跑馬燈，並在老師的協助下，編製其內部程式和支援工具。

編寫的過程中，可以逐步地觀察資料的轉載及顯示時，能親眼觀察的排列順序、變化，將此一現象用以探討之潛在用途，加以落實相關物件上。

貳、 研究目的

國外已有相關能顯示英文字母及符號的 LED 函式庫，還能從函式庫中調整字體大小，跑馬燈上下左右移動的效果。而我們欲顯示指定繁體中文字在 LED 上，在國內並無相關或是公開的函式庫可以利用，因此欲顯示的繁體中文字，須利用相關的程式語言，協助二進位資料快速處理，從單一個 8*8LED 成功顯示基本的點陣中文字，我們更進一步做出 16*16 的點陣繁體中文字。

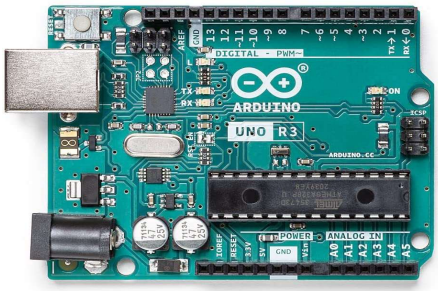
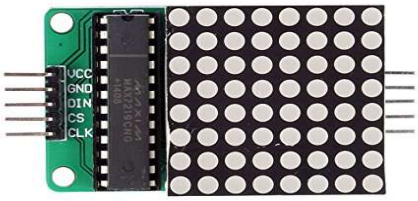
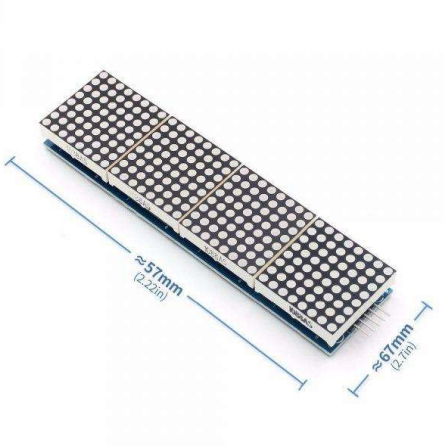
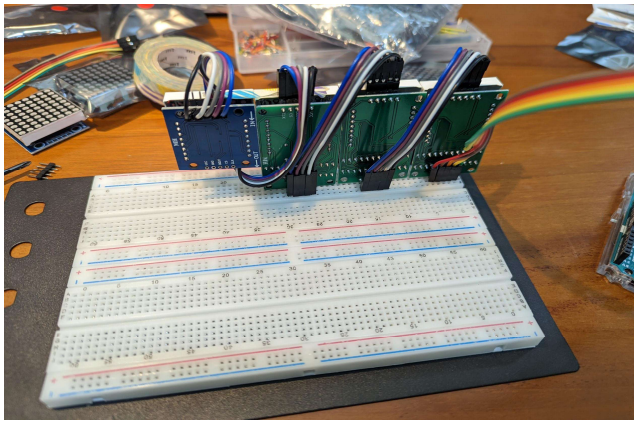
了解跑馬燈原理之後，做出了 8*8 的繁體中文字往左移位的跑馬燈；並串聯拼接 16 個 8*8 的 LED，做出 16*16 點陣字動態顯示在外掛的 LED 燈板上。

參、 研究設備及器材

一、軟體方面

程式語言	Python、C 語言(Arduino 基本語法)
應用軟體	PyCharm IDE、Excel

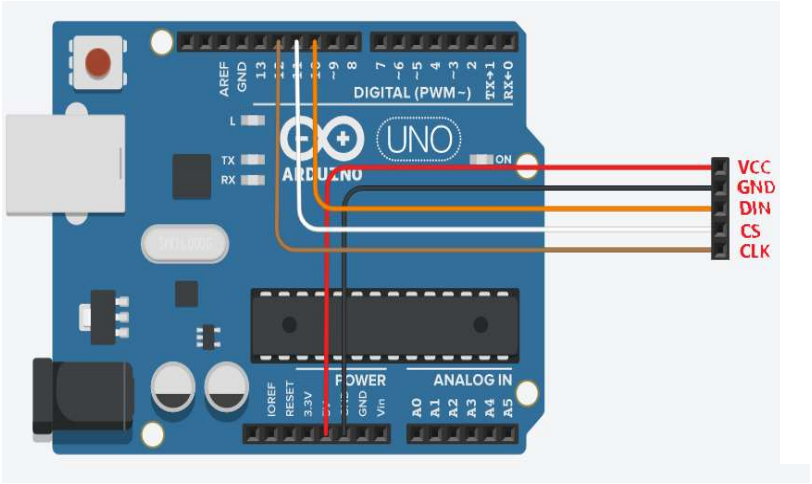
二、硬體方面

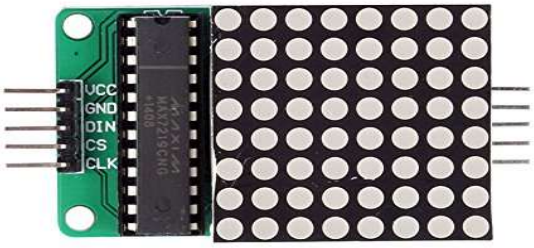
<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>Arduino MEGA2560 開發版</td><td>Arduino UNO 開發版</td></tr>
<tr><td colspan="2">為本次研究之主要操作核心，兩者由電腦輸入程式碼後，對其插件進行控制</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>Max7219 LED 模組</td><td>4 合 1 Max7219 LED 模組</td></tr>
<tr><td colspan="2">用以顯示成果，兩端為輸出和輸入端，可一次串聯多個 Max7219 擴增螢幕面積</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>麵包版</td><td>杜邦線</td></tr>
</table>

肆、 研究過程與結果

一、 8*8 字體恆亮

1. 線路

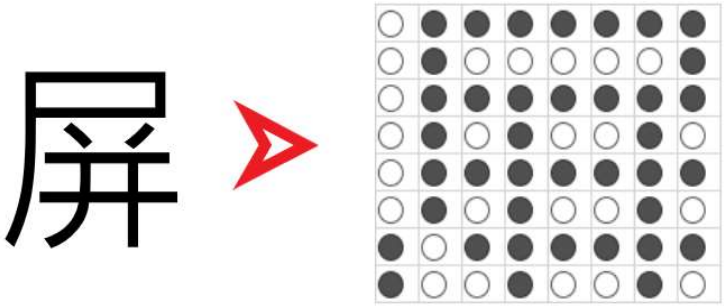




(**註**：彩虹色杜邦線為 Arduino-LED 輸入端，即前一圖所示；暗色系杜邦線為燈板的連接線，上方為輸出端，下方為輸入端，深藍色燈板為其擴充)

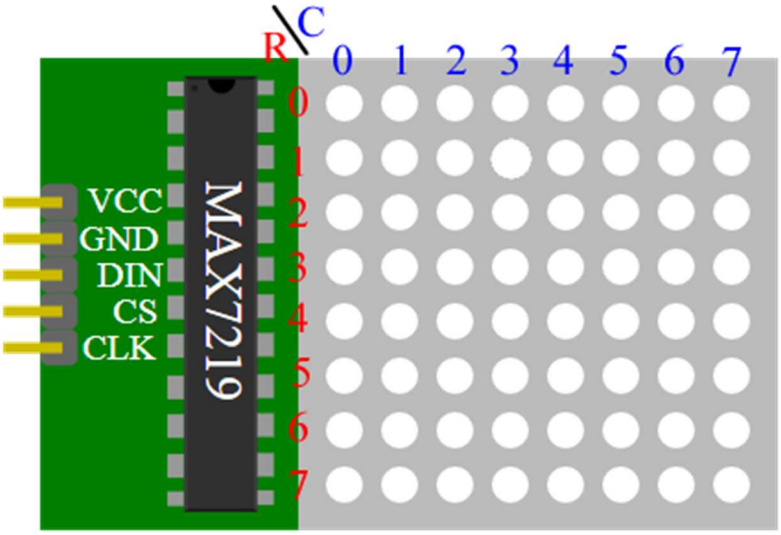
2. 字樣轉換

在漢字輸入 Arduino 主程式前，必須經過一串處理程序，即將其轉換為點陣字型(在此使用 Excel 做模擬)，如圖：



如此一來，就可將此資料傳換為一格格的可用資料了。但還不可止步於此。還得將其

轉換為二進位制，才可進行實際應用。



以上為 Max7219 LED，其座標的圖示範例。可見整個 8*8 的燈板是被分成 R(row)0~7 和 C(column)0~7 的座標平面。這時，我們須借助一個在 Arduino IDE 裡，稱作 LedControl 的函式庫，將前一張圖中的點陣字「屏」，能夠成功地輸入開發版中。

```
{B10010010,
 B10111111,
 B01010010,
 B01111111,
 B01010010,
 B01111111,
 B01000001,
 B01111111}
```

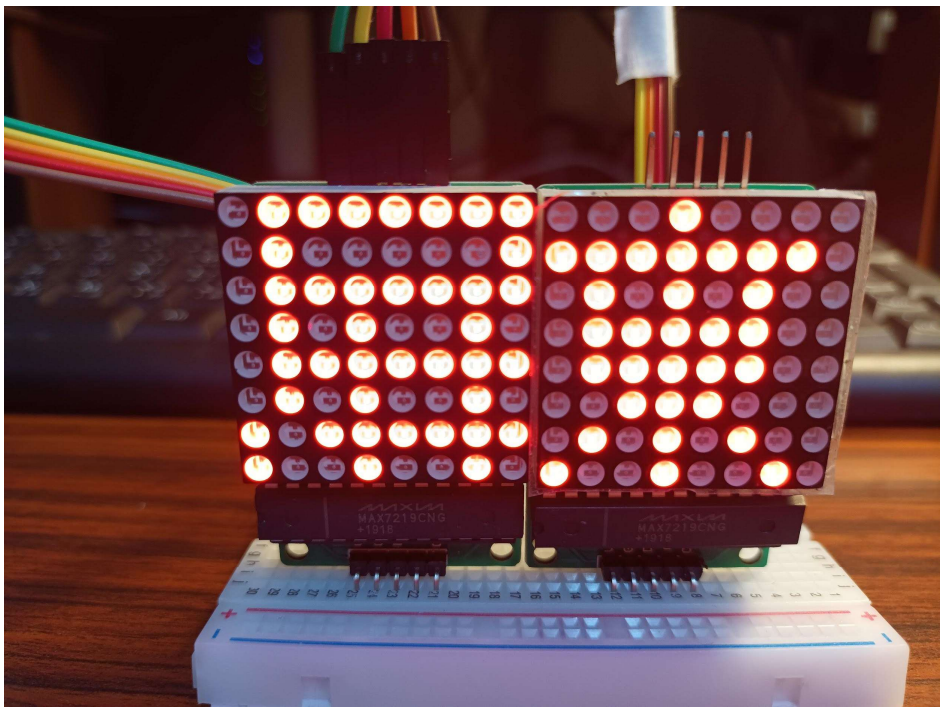
(預覽：由於為由下到上，由左至右，從陣列第一串數印出，因此在檢視中如同上下顛倒般)

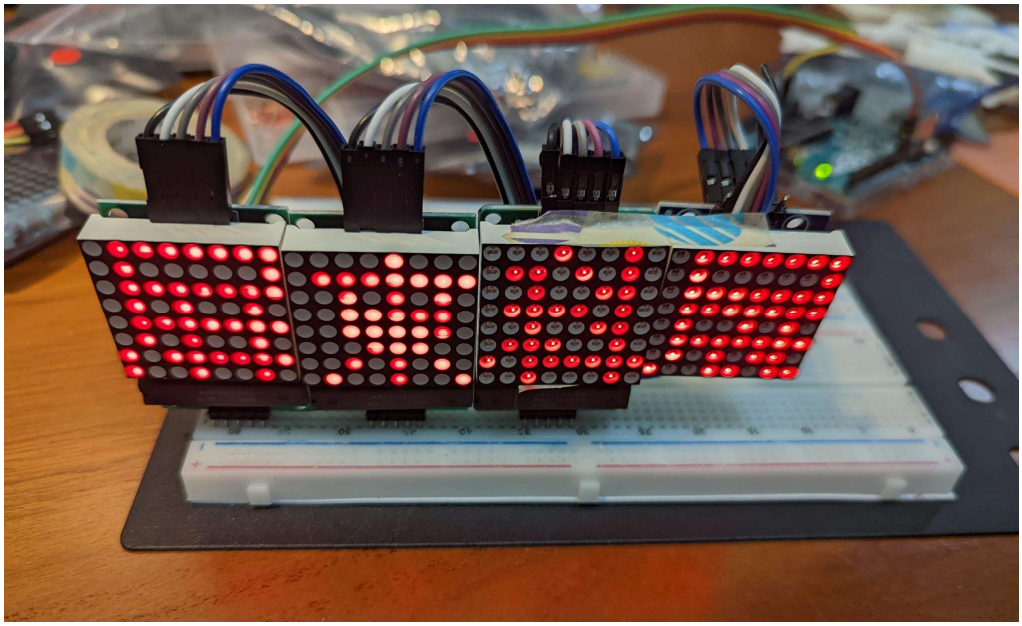
3. 程式

終於，我們能開始編譯主要的程式碼了。(DIN→12、CS→10、CLK→11)

```
#include <LedControl.h>  //引用 LedControl 程式庫
LedControl lc=LedControl(12,11,10,2);  //宣告 LedControl 物件

void setup() {
  for(int i=0;i<2;i++){
    lc.shutdown(i,false);/* 將亮度設定為正常模式 */
    lc.setIntensity(i,1);/* 調整顯示器亮度 */
    lc.clearDisplay(i);/* 所有LED設定成不顯示 */
  }
}

void loop() {
  int row;
  byte a[8]={B10010010,B10111111,B01010010,B01111111,B01010010,B01111111,B01000001,B01111111};/* 屏 */
  byte b[8]={B10010010,B01010100,B00111000,B01111100,B01111100,B01010100,B11111110,B00010000};/* 東 */
  for(int row=0;row<8;row++){
    lc.setColumn(0,row,a[row]);/* 設定第一個裝置，第row行的8個LED為亮(1)或滅(0)狀態 */
    lc.setColumn(1,row,b[row]);/* 設定第二個裝置，設定第row行的8個LED為亮(1)或滅(0)狀態 */
    delay(100);
    }
}
```

4. 結果：屏東二字恆亮





(**註**：在程式碼裡再加以修改後，即可擴增愈廣的螢幕空間，如上圖)

二、 16*16 字體恆亮

1. 線路

由於 LedControl 函式庫內建函式中，相關指令只能最多控制 8 個 MAX7219 LED 元件，因次我們焊接串聯 2 個 4 合一 LED 元件後(如圖 2-2)，再拉出 VCC 和 GND 的線串接第二排的 LED(如圖 2-1)，而第二排 8 顆 MAX7219 LED 的 DIN、CS、CLK 則接到開發版的 9、7、8，第一排的接線方式則和上述一顆 8*8LED 相同。

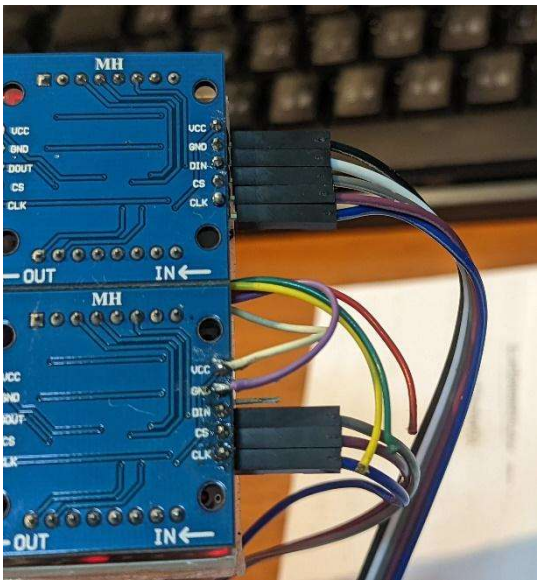


圖 2-1

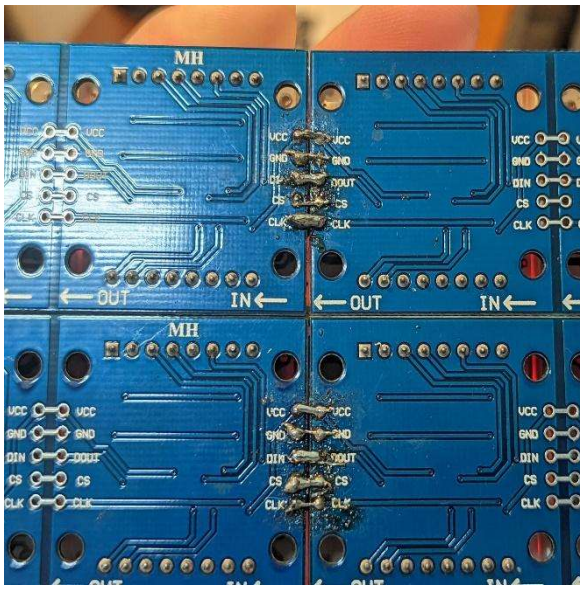
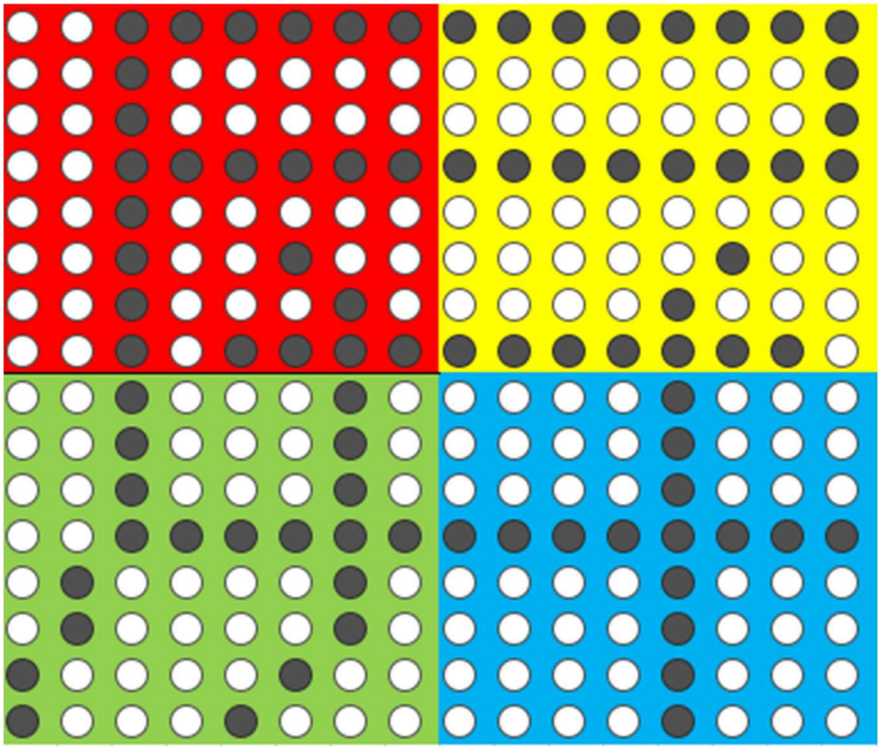


圖 2-2

2. 程式

接下來，我們測試的就是 16*16 的字體顯示了。它較 8*8 矩陣不同的是，當顯示 16*16 的漢字時，每個字是以四個燈板所組成的；換句話說，每個字得賦予它共四個陣列，各自輸入進相對的裝置，才可完整地印出單一個字。如圖：



為加速我們獲取每個 8*8 的 2 進位，我們寫了一個以 Python 為語言的分割工具，以協助我們編寫主程式。

但在實作的過程當中，我們發現到這裡行，對應到 MAX72192 的顯示程式碼中，是 Row(列)，因此還要在 Aduino 進行一次轉換。

以下為我們利用 Python 內建的 openpyxl 的函式庫寫了一個抓取 Excel 二進位資料的程式碼，加快了我們抓取位元值的速度很多。

程式接下頁：

```python
from openpyxl import Workbook, load_workbook  # 引用openpyxl 程式庫
from openpyxl.utils import get_column_letter  # 引用openpyxl 擴充函式

BitWord = load_workbook('點陣圖式1110225.xlsx')  # 匯入檔案
title = BitWord.sheetnames
save_list = []

for name in range(len(title)):  # 資料存入陣列(尚未分割，橫列儲存)
    sheet = BitWord[title[name]]
    save_list.append([])
    for row in range(1, 17):  # 分16行
        save_list[name].append([])
        for col in range(1, 17):  # 輸入數值
            char = get_column_letter(col)
            save_list[name][row - 1].append(sheet[char + str(row)].value)

"""..."""

save_char = ""  # 四部分分割
t_list = []
o_list = []

for name in range(4):  # 四字版
    t_list.append([[], [], [], []])  # 1/4
    o_list.append([[], [], [], []])
    for row in range(16):
        for sep in range(2):  # 判斷上下部分
            for num in range(8):  # 判斷左右部分
                if sep == 0:
                    save_char += str(save_list[name][row][num])
                else:
                    save_char += str(save_list[name][row][num + 8])
            if row < 8:
                t_list[name][sep].append(save_char)
            else:
                t_list[name][sep + 2].append(save_char)
            save_char = ""
```

```
save_char = ""  # 四部分分割
t_list = []
o_list = []

for name in range(4):  # 四字版
    t_list.append([[], [], [], []])  # 1/4
    o_list.append([[], [], [], []])
    for row in range(16):
        for sep in range(2):  # 判断上下部分
            for num in range(8):  # 判断左右部分
                if sep == 0:
                    save_char += str(save_list[name][row][num])
                else:
                    save_char += str(save_list[name][row][num + 8])
            if row < 8:
                t_list[name][sep].append(save_char)
            else:
                t_list[name][sep + 2].append(save_char)
            save_char = ""

save_char = "B"  # +B&縦置

for word in range(4):
    for part in range(4):
        for char in range(8):
            for line in range(7, -1, -1):
                save_char += t_list[word][part][line][char]
            o_list[word][part].append(save_char)
            save_char = "B"

for word in range(4):  # 印出程序&整理
    for part in range(4):  # 分字
        print("{", end="")
        for row in range(8):
            print(o_list[word][part][row], end="")
            if row < 7:
                print(", ", end="")
        print("}")
    print("")
```

```
{B00000000, B00000000, B11111111, B00001001, B10001001, B10101001, B11001001, B10001001}
{B10001001, B10001001, B10001001, B10001001, B11001001, B10101001, B10001001, B00001111}
{B11000000, B00110000, B00001111, B00001000, B10001000, B01001000, B00111111, B00001000}
{B00001000, B00001000, B00001000, B00001000, B11111111, B00001000, B00001000, B00001000}

{B00000000, B00000010, B00000010, B11111010, B01001010, B01001010, B01001010, B01001010}
{B11111111, B01001010, B01001010, B01001010, B01001010, B11111010, B00000010, B00000010}
{B00000000, B00000000, B00000000, B01000011, B00100010, B00010010, B00001010, B01000110}
{B11111111, B00000110, B00001010, B00010010, B00100010, B01000011, B01000000, B00000000}

{B00010000, B00010010, B10010010, B11111110, B01010001, B10010001, B00000000, B00100010}
{B01000100, B10011000, B00010000, B00000000, B11111111, B00000000, B00000000, B00000000}
{B00000010, B00000001, B00000000, B01111111, B00000000, B00000001, B00001000, B00001000}
{B00001000, B00001001, B00000101, B00000100, B01111111, B00000100, B00000100, B00000000}

{B00000000, B00000000, B11111111, B01001001, B01001001, B01001001, B11101001, B01001001}
{B01001001, B01001001, B01001001, B01001001, B11101001, B01001001, B01001001, B00001111}
{B11000000, B00110000, B00001111, B00000010, B00000010, B11111110, B10000011, B01000010}
{B01000110, B00001010, B00010010, B00110010, B01101011, B11001010, B10000010, B10000010}
```

有了這些資料後，就能進行下一步的編譯了。

在 16*16 的矩陣中，每個燈板的排列方式入下：(橫置)

其中數字就是在 LedControl 程式中每個 LED 元件的位址

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

因此，把以上陣列置入程式碼後，還得將其個別配對，以正確印出。

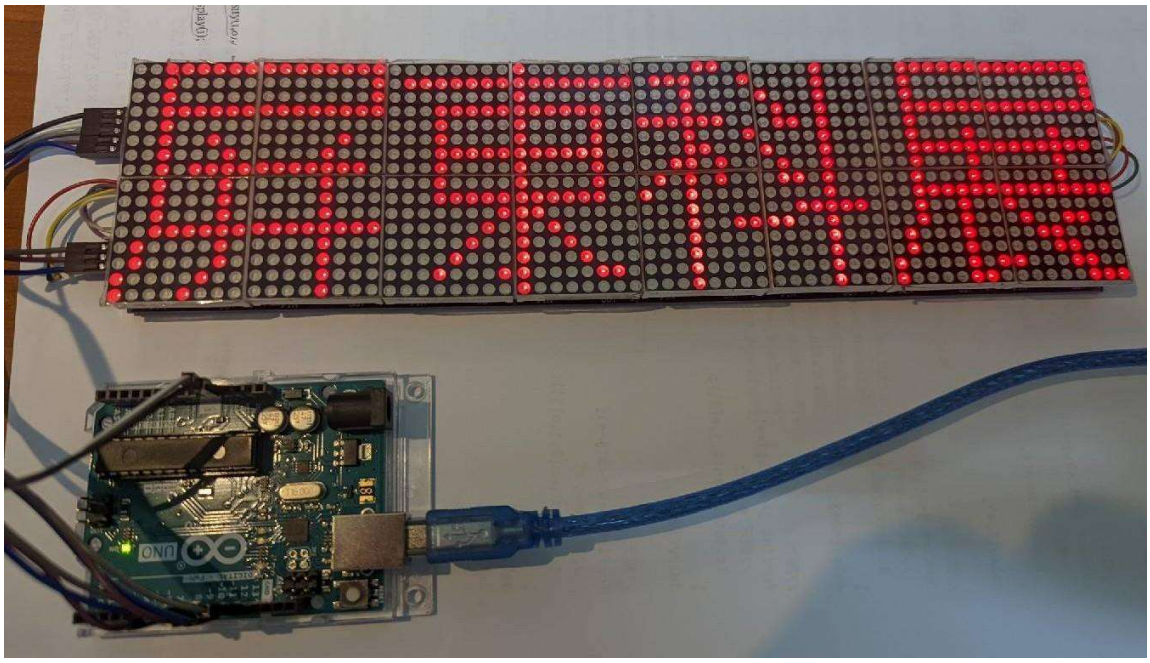
Arduino 程式碼接下頁：

```
#include "LedControl.h"  //引用 LedControl 程式庫
LedControl lc1=LedControl(12,11,10,8);
LedControl lc2=LedControl(9,8,7,8);//宣告 LedControl 物件
byte a[8]={B00000000,B00000000,B11111111,B00001001,B10001001,B10101001,B11001001,B10001001};
byte b[8]={B10001001,B10001001,B10001001,B10001001,B11001001,B10101001,B10001001,B00001111};
byte c[8]={B00000000,B00000010,B00000010,B11111010,B01001010,B01001010,B01001010,B01001010};
byte d[8]={B11111111,B01001010,B01001010,B01001010,B01001010,B11111010,B00000010,B00000010};
byte e[8]={B00010000,B00010010,B10010010,B11111110,B01010001,B10010001,B00000000,B00100010};
byte f[8]={B01000100,B10011000,B00010000,B00000000,B11111111,B00000000,B00000000,B00000000};
byte g[8]={B00000000,B00000000,B11111111,B01001001,B01001001,B01001001,B11101001,B01001001};
byte h[8]={B01001001,B01001001,B01001001,B01001001,B11101001,B01001001,B01001001,B00001111};
byte i[8]={B11000000,B00110000,B00001111,B00001000,B10001000,B01001000,B00111111,B00001000};
byte j[8]={B00001000,B00001000,B00001000,B00001000,B11111111,B00001000,B00001000,B00001000};
byte k[8]={B00000000,B00000000,B00000000,B01000011,B00100010,B00010010,B00001010,B01000110};
byte l[8]={B11111111,B00000110,B00001010,B00010010,B00100010,B01000011,B01000000,B00000000};
byte m[8]={B00000010,B00000001,B00000000,B11111111,B00000000,B00000001,B00001000,B00001000};
byte n[8]={B00001000,B00001001,B00000101,B00000100,B11111111,B00000100,B00000100,B00000000};
byte o[8]={B11000000,B00110000,B00001111,B00000010,B00000010,B11111110,B10000011,B01000010};
byte p[8]={B01000110,B00001010,B00010010,B00110010,B01101011,B11001010,B10000010,B10000010};
void setup() {
  for(int i=0;i<8;i++){
    lc1.shutdown(i,false);/* Set the brightness to a medium values */
    lc1.setIntensity(i,0);/* and clear the display */
    lc1.clearDisplay(i);
  }
  for(int i=0;i<8;i++){
    lc2.shutdown(i,false);/* Set the brightness to a medium values */
    lc2.setIntensity(i,0);/* and clear the display */
    lc2.clearDisplay(i);
  }
```

```
void loop() {
  for (int s=0;s<8;s++){
        lc1.setColumn(0,7-s,a[s]);
        lc1.setColumn(1,7-s,b[s]);
        lc1.setColumn(2,7-s,c[s]);
        lc1.setColumn(3,7-s,d[s]);
        lc1.setColumn(4,7-s,e[s]);
        lc1.setColumn(5,7-s,f[s]);
        lc1.setColumn(6,7-s,g[s]);
        lc1.setColumn(7,7-s,h[s]);
        lc2.setColumn(0,7-s,i[s]);
        lc2.setColumn(1,7-s,j[s]);
        lc2.setColumn(2,7-s,k[s]);
        lc2.setColumn(3,7-s,l[s]);
        lc2.setColumn(4,7-s,m[s]);
        lc2.setColumn(5,7-s,n[s]);
        lc2.setColumn(6,7-s,o[s]);
        lc2.setColumn(7,7-s,p[s]);

  }
}  //置入
```

（註：已經過處理）

3. 結果



三、 8*8 字體左移跑馬燈

1. 原理

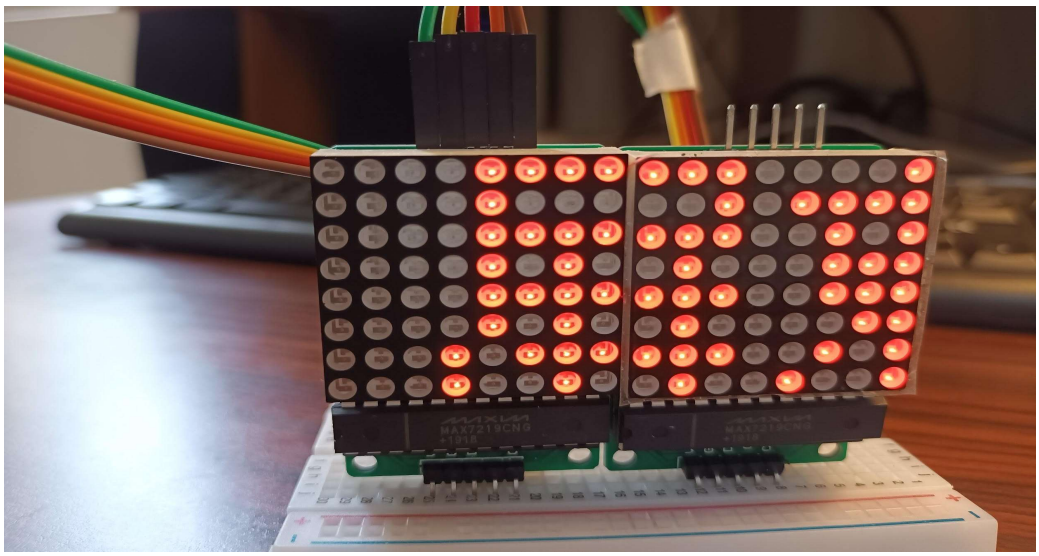
做完了靜態的字幕顯示後，下一步就要來製出動態的跑馬燈了。同樣地，我們先從 8*8 矩陣開始。

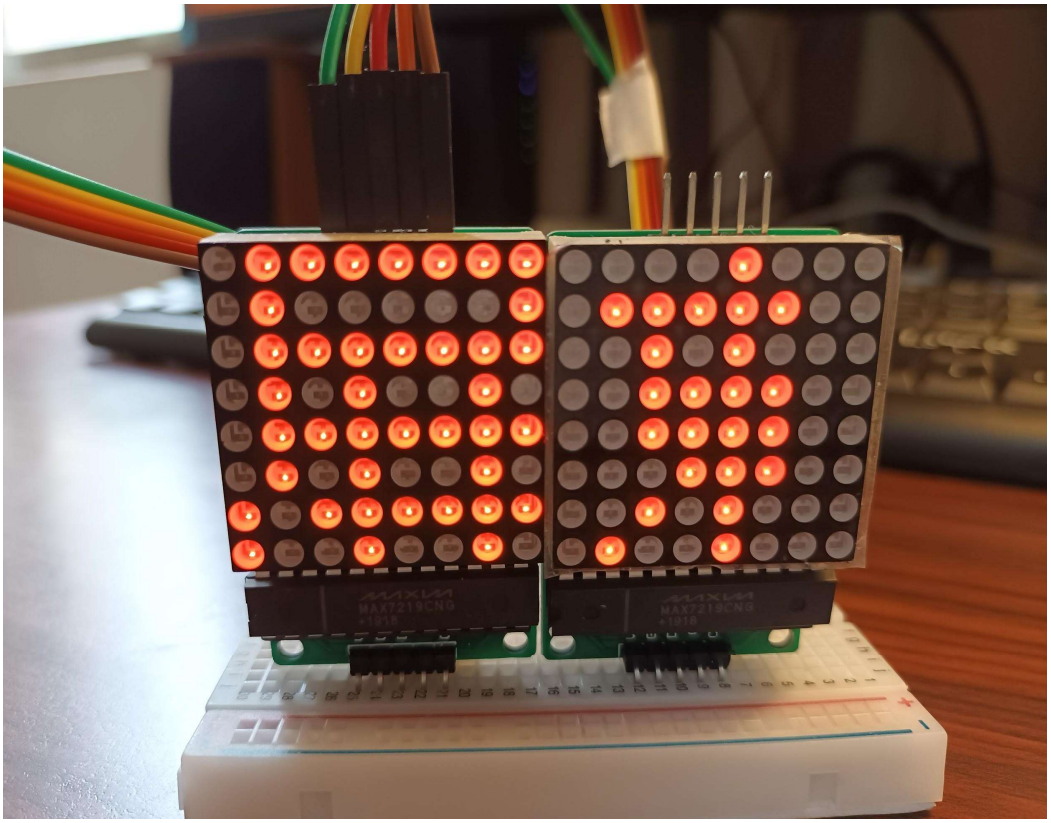
如何讓字體逐時移動位置呢？我們想到了一套辦法—將資料本身與燈板陣列各自獨立後，將資料在每次迴圈啟動時向左移動，並複製前一列的數串，同時最右端的串列再更新成新的一列，

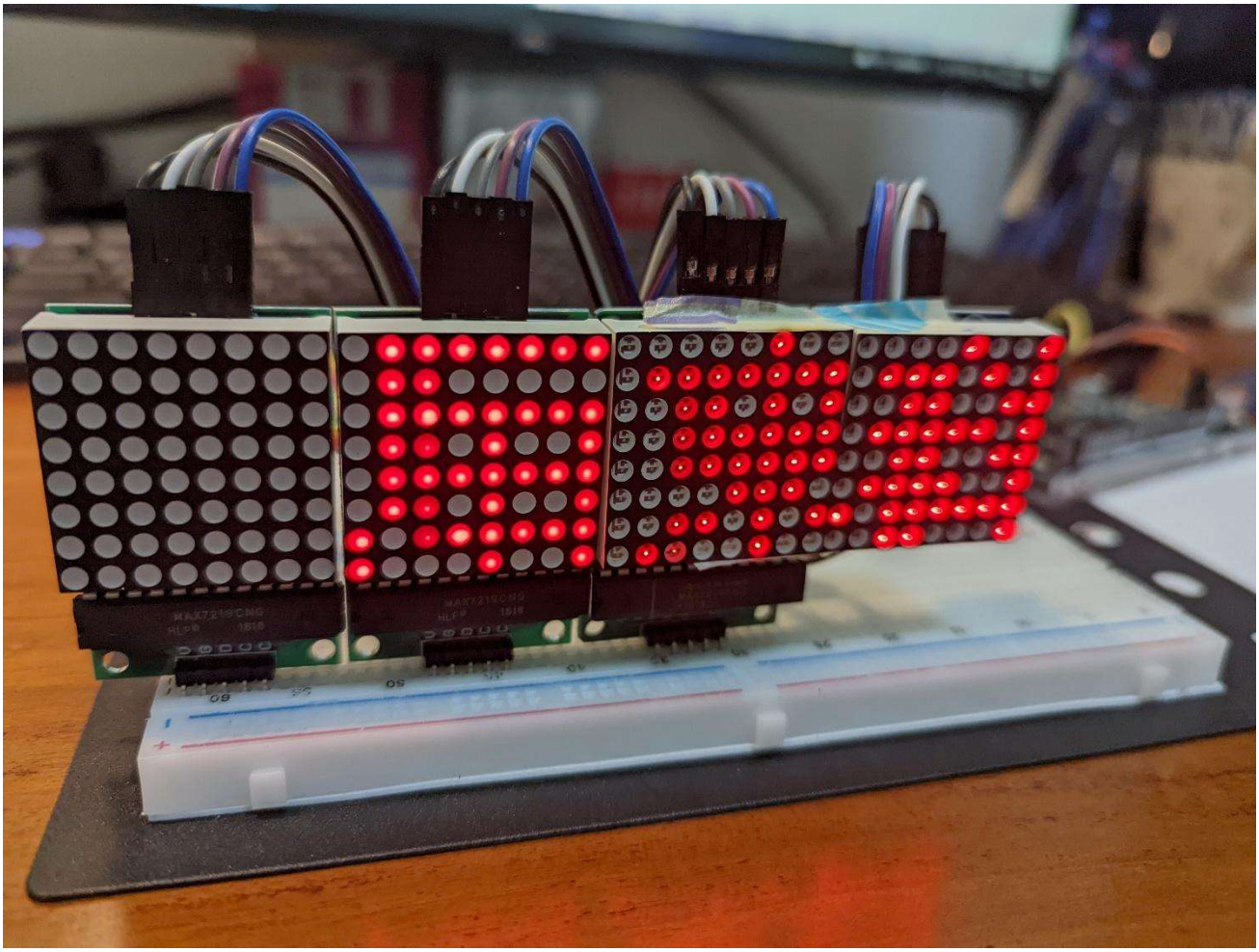
```
#include "LedControl.h"  //引用 LedControl 程式庫
LedControl lc=LedControl(12,11,10,8);  //宣告 LedControl 物件

byte a[8]={B11000000,B00111111,B01010101,B11111101,B01010101,B01010101,B11111101,B01010111};
byte b[8]={B00000000,B10000010,B01011110,B00111010,B11111111,B00111010,B01011110,B10000010};
byte temp1[8]={0,0,0,0,0,0,0,0};//第1顆LED暫存陣列
byte temp2[8]={0,0,0,0,0,0,0,0};//第0顆LED暫存陣列

void setup() {
  for(int j=0;j<2;j++){
    lc.shutdown(j,false);  // 關閉省電模式
    lc.setIntensity(j,0);  // 設定亮度為 0 (介於0~15之間)
    lc.clearDisplay(j);    // 清除螢幕
  }

}
```

```
void loop() {
  for(int j=0;j<8;j++){
    for(int i=0;i<7;i++){
      temp1[i]=temp1[i+1];
      lc.setRow(1,i,temp1[i]);/* 前次讀入資料移左 */
    }
    temp1[7]=a[j];/* 逐右讀取 */
    lc.setRow(1,7,temp1[7]);/* 最末端(右)印入「屏」的第j列 */
    delay(200);
  }//第1顆先跑屏字進來
  for(int j=0;j<8;j++){
    for(int i=0;i<7;i++){/* 資料移左 */
      temp1[i]=temp1[i+1];
      temp2[i]=temp2[i+1];
      lc.setRow(1,i,temp1[i]);
      lc.setRow(0,i,temp2[i]);
    }
    temp1[7]=b[j];/* 第一顆讀入「東」 */
    temp2[7]=a[j];/* 第零顆讀入「屏」 */
    lc.setRow(1,7,temp1[7]);
    lc.setRow(0,7,temp2[7]);
    delay(200);
  }//第1顆繼續跑東字，同時第0顆跑屏字進來
  for (int i=0;i<8;i++){
      temp1[i]=0;
      temp2[i]=0;
  }//消除2個暫存器陣列
  for(int j=0;j<8;j++){
    lc.setRow(1,7-j,0);
    temp1[j]=0;
    delay(200);
  }
  for(int j=0;j<8;j++){
    lc.setRow(0,7-j,0);
    temp1[j]=0;
    delay(200);
  }
}
```

2. 結果







四、 16*16 字體左移跑馬燈

第一次嘗試

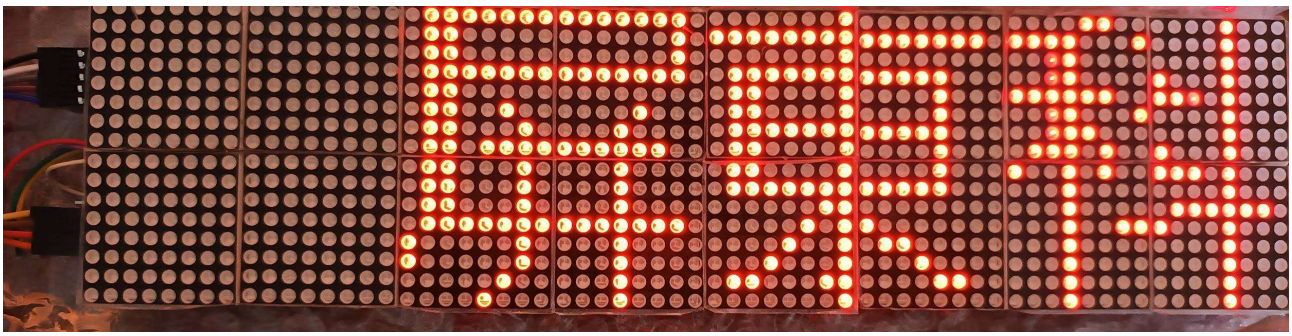
研究的最後，終於可以來製作最初的目標了。

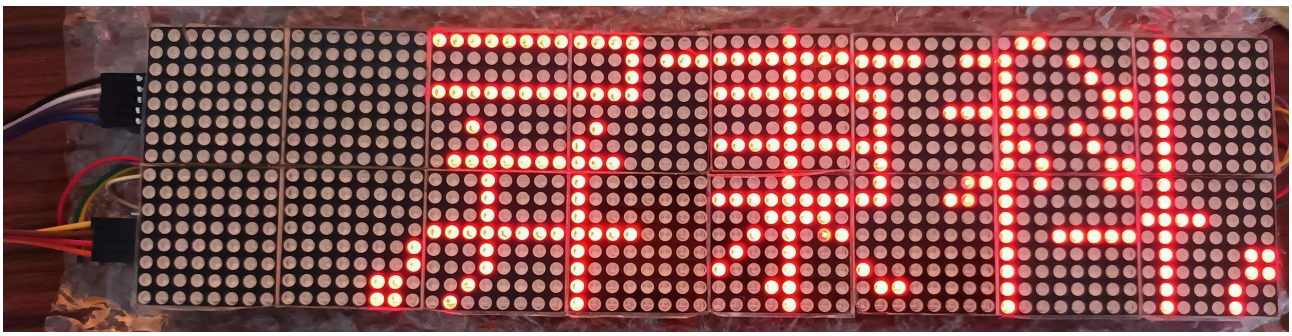
一開始，我們沿續上回製作 8*8 跑馬燈時的原理，套用在此次的程式碼中(以下程式碼部分省略，從 Void loop 開始顯示)；

```
void loop() {
  for (int s=0;s<8;s++){
    for(int t=0;t<7;t++){
      temp1[7-t]=temp1[6-t];
      Temp1[7-t]=Temp1[6-t];
      lc1.setColumn(7,7-t,temp1[7-t]);
      lc2.setColumn(7,7-t,Temp1[7-t]);
    }
    temp1[0]=a[s];
    Temp1[0]=i[s];
    lc1.setColumn(7,0,temp1[0]);
    lc2.setColumn(7,0,Temp1[0]);
    delay(100);
  }
  for(int s=0;s<8;s++){
    for(int t=0;t<7;t++){
        temp1[7-t]=temp1[6-t];
        temp2[7-t]=temp2[6-t];
        Temp1[7-t]=Temp1[6-t];
        Temp2[7-t]=Temp2[6-t];
        lc1.setColumn(7,7-t,temp1[7-t]);
        lc1.setColumn(6,7-t,temp2[7-t]);
        lc2.setColumn(7,7-t,Temp1[7-t]);
        lc2.setColumn(6,7-t,Temp2[7-t]);
      }
      temp1[0]=b[s];
      temp2[0]=a[s];
      Temp1[0]=j[s];
      Temp2[0]=i[s];
      lc1.setColumn(7,0,temp1[0]);
      lc1.setColumn(6,0,temp2[0]);
      lc2.setColumn(7,0,Temp1[0]);
      lc2.setColumn(6,0,Temp2[0]);
      delay(100);
    }
    for(int s=0;s<8;s++){
      for(int t=0;t<7;t++){
        temp1[7-t]=temp1[6-t];
        temp2[7-t]=temp2[6-t];
        temp3[7-t]=temp3[6-t];
        Temp1[7-t]=Temp1[6-t];
        Temp2[7-t]=Temp2[6-t];
        Temp3[7-t]=Temp3[6-t];
        lc1.setColumn(7,7-t,temp1[7-t]);
        lc2.setColumn(5,7-t,Temp3[7-t]);
```

(註：程式碼實在太長了，接下來的過程都是處理字幕跨版)

然而，在顯示時出了問題：





如上面兩圖所示，發現在每個 LED 元件與元件之間，發生訊號延遲出現的狀況，而且隨著時間越久，每行燈亮的時間也越久，已經快要沒有跑馬燈的效果出現。我們嘗試修改了程式，認為是否越到後面的迴圈裡面的程式碼太多導致。但我們回頭看了 4 個字的 8*8 跑馬燈並沒有此現象。

因此去觀察 LedControl 函式庫的原始碼，發現了 SetRow 的函式內容和 SetColumn 函式的寫法不盡相同。(如下圖)

```
141 void LedControl::setColumn(int
     addr, int col, byte value) {
142     byte val;
143
144     if(addr<0 || addr>=maxDevices)
145         return;
146     if(col<0 || col>7)
147         return;
148     for(int row=0;row<8;row++) {
149         val=value >> (7-row);
150         val=val & 0x01;
151         setLed(addr,row,col,val);
152     }
153 }
```

由 LedControl 原始的函式庫程式碼可知，我們所使用的 setColumn 其實是由 LedControl 其中的 setLed 衍伸而來的功能，這使得 Arduino 在計算時變得吃力，此次測試並未達到我們所要求的期望，在和老師討論之後，我們換了新的顯示方法。

第二次嘗試

setColumn 雖然有運算時間過長的問題，可能力與之相近的 setRow 並無此煩惱。

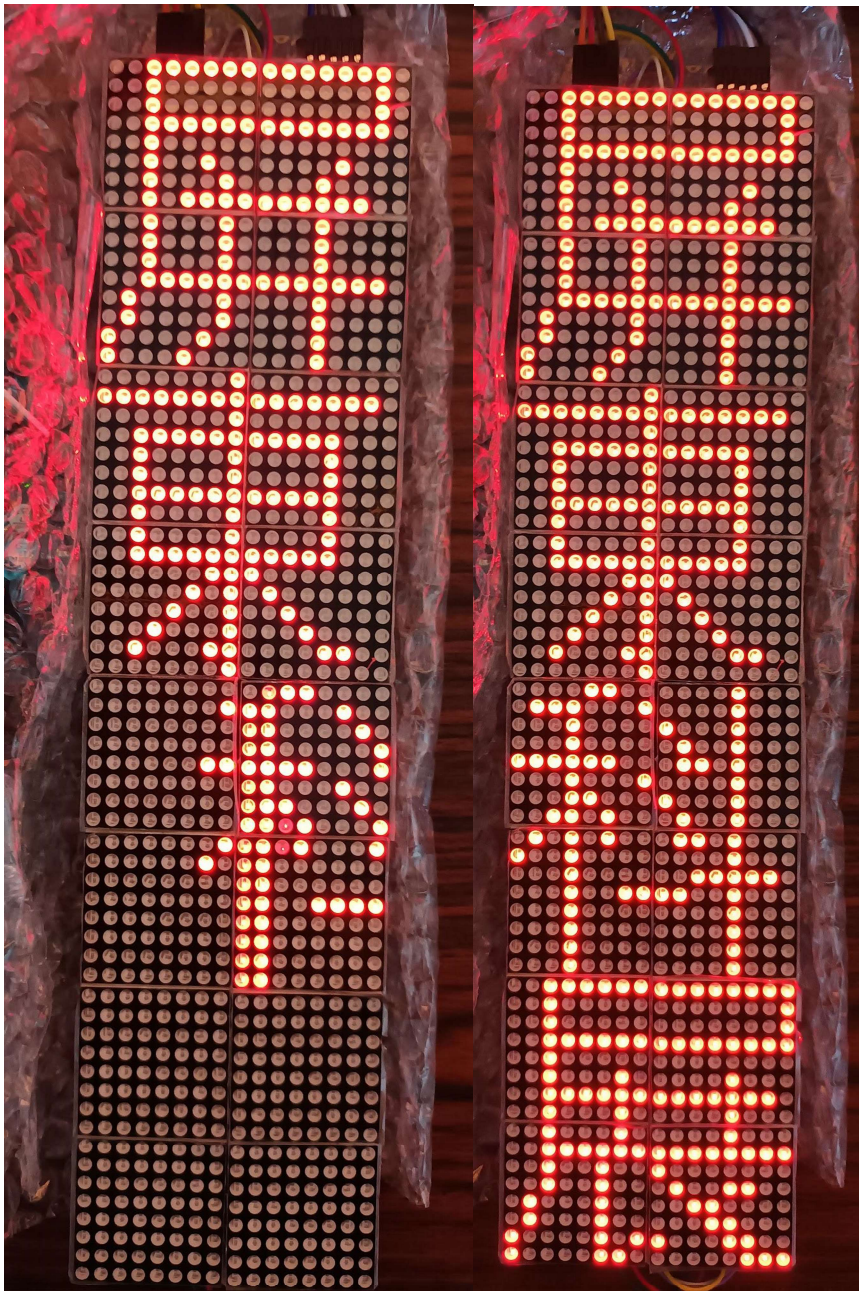
```
130 void LedControl::setRow(int addr,
    int row, byte value) {
131     int offset;
132     if(addr<0 || addr>=maxDevices)
133         return;
134     if(row<0 || row>7)
135         return;
136     offset=addr*8;
137     status[offset+row]=value;
138     spiTransfer(addr, row+1
    ,status[offset+row]);
139 }
```

相較於 setColumn 需二次引用，setRow 可直接下達指令；因此，我們將原定的橫向移動，改為縱向印入。(程式接下頁)

```
void loop() {
  for(int j=0;j<8;j++){
    for(int i=0;i<7;i++){
      temp1[i]=temp1[i+1];
      temp2[i]=temp2[i+1];
      lc1.setRow(0,i,temp1[i])
      lc1.setRow(1,i,temp2[i])
    }
    temp1[7]=a[j];
    temp2[7]=b[j];
    lc1.setRow(0,7,temp1[7]);
    lc1.setRow(1,7,temp2[7]);
    delay(200);
  }
  for(int j=0;j<8;j++){
    for(int i=0;i<7;i++){
      temp1[i]=temp1[i+1];
      temp2[i]=temp2[i+1];
      temp3[i]=temp3[i+1];
      temp4[i]=temp4[i+1];
      lc1.setRow(0,i,temp1[i]);
      lc1.setRow(1,i,temp2[i]);
      lc2.setRow(0,i,temp3[i]);
      lc2.setRow(1,i,temp4[i]);
    }
    temp1[7]=c[j];
    temp2[7]=d[j];
    temp3[7]=a[j];
    temp4[7]=b[j];
    lc1.setRow(0,7,temp1[7]);
    lc1.setRow(1,7,temp2[7]);
    lc2.setRow(0,7,temp3[7]);
    lc2.setRow(1,7,temp4[7]);
    delay(200);
  }//先跑屏字
    for (int i=0;i<8;i++){
      temp1[i]=0;
      temp2[i]=0;
      temp3[i]=0;
      temp4[i]=0;
    }//洗陣列
    for(int j=0;j<8;j++){
      for(int i=0;i<7;i++){
        temp1[i]=temp1[i+1];
        temp2[i]=temp2[i+1];
        lc1.setRow(2,i,temp1[i])
        lc1.setRow(3,i,temp2[i])
      }
      temp1[7]=e[j];
      temp2[7]=f[j];
      lc1.setRow(2,7,temp1[7]);
      lc1.setRow(3,7,temp2[7]);
      delay(200);
    }
    for(int j=0;j<8;j++){
      for(int i=0;i<7;i++){
        temp1[i]=temp1[i+1];
        temp2[i]=temp2[i+1];
        temp3[i]=temp3[i+1];
        temp4[i]=temp4[i+1];
        lc1.setRow(2,i,temp1[i]);
        lc1.setRow(3,i,temp2[i]);
        lc2.setRow(2,i,temp3[i]);
        lc2.setRow(3,i,temp4[i]);
      }
      for(int t=0;t<8;t++){
        for(int s=0;s<7;s++){
          temp1[s]=temp1[s+1];
          temp2[s]=temp2[s+1];
          lc1.setRow(4,s,temp1[s]);
          lc1.setRow(5,s,temp2[s]);
        }
        temp1[7]=i[t];
        temp2[7]=j[t];
        lc1.setRow(4,7,temp1[7]);
        lc1.setRow(5,7,temp2[7]);
        delay(200);
      }
      for(int t=0;t<8;t++){
        for(int s=0;s<7;s++){
          temp1[s]=temp1[s+1];
          temp2[s]=temp2[s+1];
          temp3[s]=temp3[s+1];
          temp4[s]=temp4[s+1];
          lc1.setRow(4,s,temp1[s]);
          lc1.setRow(5,s,temp2[s]);
          lc2.setRow(4,s,temp3[s]);
          lc2.setRow(5,s,temp4[s]);
        }
        temp1[7]=k[t];
        temp2[7]=l[t];
        temp3[7]=i[t];
        temp4[7]=j[t];
        lc1.setRow(4,7,temp1[7]);
        lc1.setRow(5,7,temp2[7]);
        lc2.setRow(4,7,temp3[7]);
        lc2.setRow(5,7,temp4[7]);
        delay(200);
      }//跑科字
```

(註：同樣地，程式碼太長，也是處理每個字逐行依序印出的暫存替換)

顯示結果(接下頁)



整個跑馬燈的效果相當流暢，沒有延遲的狀況發生。

伍、 討論

Arduino 公開的函式庫互相分享對於自造者的幫助真的很大，我們在此研究最依賴的 LedControl 幫我們大大的縮短程式碼，也讓我們能利用 8*8 共陰矩陣及 MAX7219 模組，顯示出任何圖案，包含本研究的繁體中文字。

若欲製造跑馬燈效果，在 LedControl 函式庫中，利用 SetRow 函數成效最佳，在每個 8*8Led 元件轉換之間沒有延遲顯示的狀況。因此本研究若不採用 4 合一 Led 燈板，而是利用 16 個單一 8*8Led 元件焊接拼接起來，轉換方向，並利用 SetRow 來修改程式，還是可以達成橫幅的跑馬燈中文字效果。

陸、 結論

未來想朝著讓使用者直接輸入欲顯示的中文字，可以直接一鍵執行程式就顯示出來，不用再透過人工用 Excel 軟體的一點一點轉換。但是，國內大部分繁體中文字體都有版權，尤其是早期 DOS 系統的倚天中文字體，是最適合 16*16 點陣字的字體。

此外，我們可以利用 OpenCV 強大的影像處理功能結合 Python 寫出程式，將字體轉成影像後，分析影像的 1、0 值，也可以幫助我們任意的轉換想顯示在 Led 燈板上的圖形了。還有很長的路可以走，希望我們在高中之後還可以繼續研究這個題目。

柒、 參考資料

1. 洪國勝、蔡懷文（2020）。Aduino 字幕機自造與程式設計。高雄市：泉勝出版有限公司。
2. 洪國勝、張孟庭（2019）。全民自造與程式設計—使用 Arduino。高雄市：泉勝出版有限公司。
3. 趙英傑（2020）。超圖解 Arduino 互動設計入門。臺北市：旗標科技股份有限公司。
4. Arduino 8*8 LED Matrix MAX7219 實習 – BLOCK 網誌
   https://www.block.tw/blog/arduino-88-led-matrix-max7219/