EduCake 使用 LED 矩陣

一、 LED 矩陣原理介紹

之前曾經介紹過利用 86Duino EduCake 的數位/類比輸出功能控制單顆或少數的 LED，但這種針腳一對一控制 LED 的方式，缺點是需要占用較多的控制器腳位，若專案裡其他功能也需要這些腳位就不方便了。讀者應該想知道，如果希望一次控制數十上百顆的 LED、燈泡等等，有甚麼方法可以用呢？本章節就來介紹利用86Duino EduCake控制多個LED矩陣的方法，讀者將可以學到使用 LED 矩陣做出圖樣顯、文字顯示等等功能的原理，就像車站、廣告看板常看到的 LED 跑馬燈那樣。

一般市面上可以買到的 LED 矩陣相當多樣化，但主要分為共陽極跟共陰極兩種，以 8X8 的 LED 矩陣為例，結構如下面圖 1、圖 2：

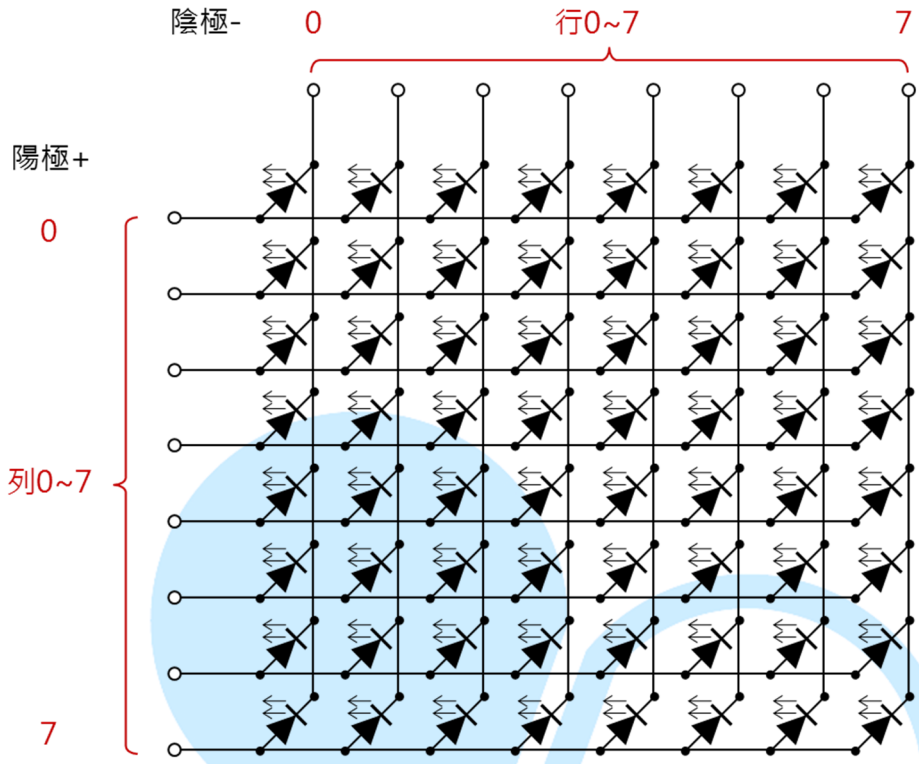


圖 1. 共陽極 LED 矩陣

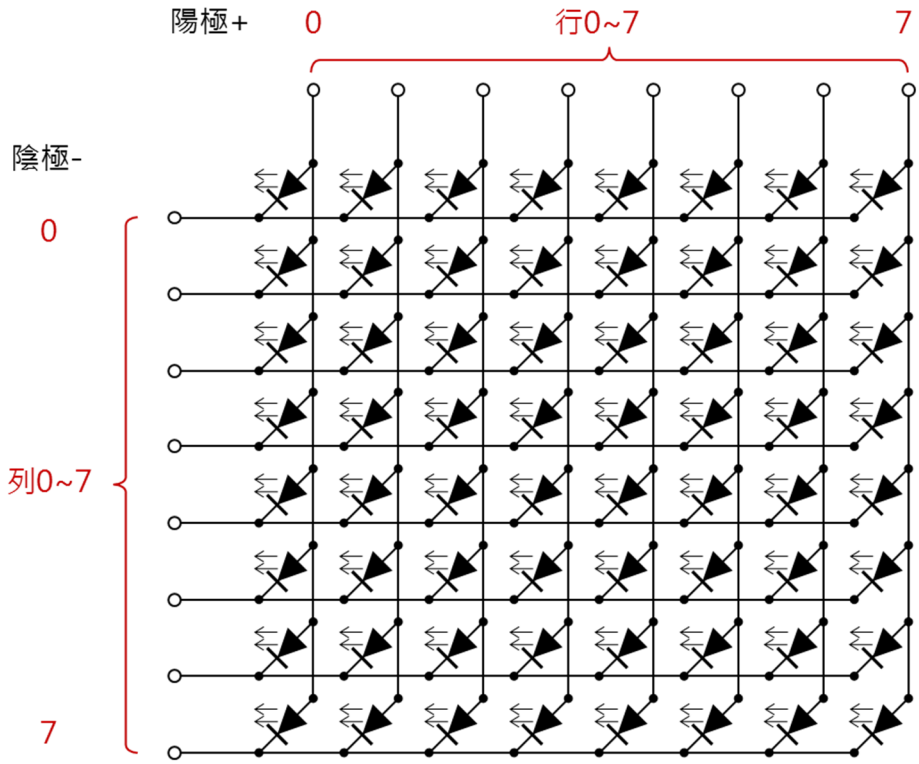


圖 2. 共陰極 LED 矩陣

從上面的圖片可以看到，所謂共陽極或共陰極表示有一整排的 LED 陰極或陽極是連接在同一個點上（事實上對於單色的 LED 矩陣來說，共陽極或共陰極實際上只是擺放方向的不同，多色 LED 就會有差異）；這種架構的 LED 矩陣便無法像以前一樣單顆進行控制，必須利用「掃描+視覺暫留」的方式來控制。如下圖 3 所示，一次點亮一排 LED 中的某幾顆，再依序換成下一排，依這樣的原理顯示整片畫面。

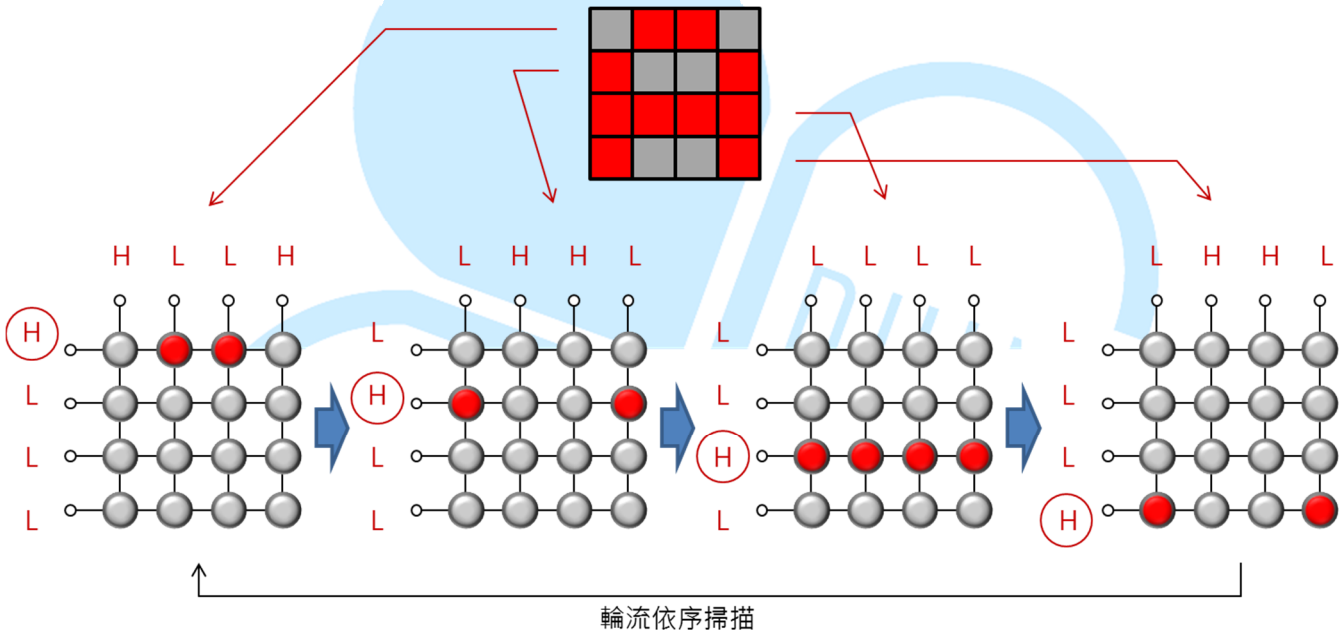


圖 3. LED 矩陣掃描控制原理圖

這種掃描式的 LED 控制可以大幅減少需要的控制針腳數，如果單純使用 86Duino EduCake 做上述的控制，則只需要 16 隻針腳跟一些電阻即可，是不是比利用 64 隻針腳控制來的有效率呢？但即使控制一個 8X8 LED 矩陣只占用 16 隻腳，實際應用上還是有諸多不便；除了造成程式變得複雜外，如果需要控制多個 LED 矩陣，那麼同樣的佔用腳位問題又會重複出現。

還好市面上已經有專用的 LED 控制 IC，可以自動幫使用者作上述的「掃描」工作，程式碼只需對 IC 做簡單的設定，便可簡易進行 LED 矩陣控制。本章節將會介紹如何使用 86Duino EduCake 與 LED 矩陣控制 IC 做搭配，來控制單個與多個 8X8 LED 矩陣；這裡使用的 LED 控制 IC 型號為 MAX7219，接線與架構圖如下圖 4：

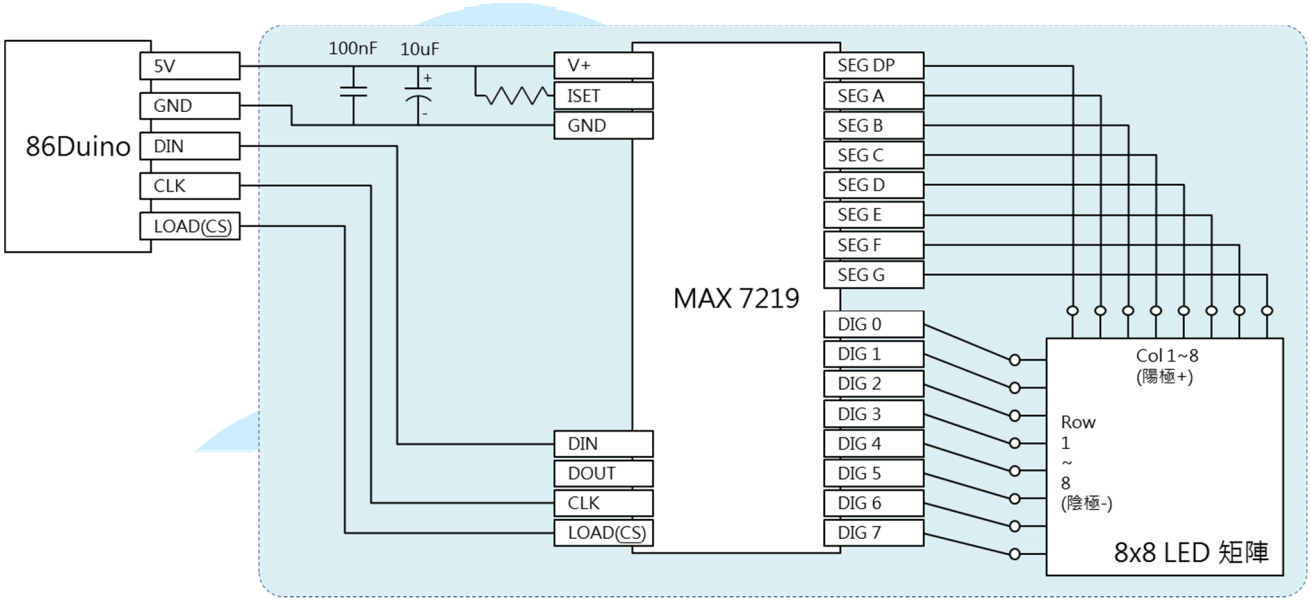


圖 4. 86Duino EduCake + MAX7219 架構圖

從上圖可以看到，MAX7219 與 86Duino EduCake 之間僅需 3 條控制訊號線，其餘兩條為電源線；讀者可以參考上圖做接線，或者也可購買市面上的整合模組（模組已包含圖中籃框內所有組件）。使用 MAX7219 除了控制訊號相當精簡外，也可以多個單元進行串聯，完全不用再消耗主控制器的寶貴針腳空間。圖 4 中可看到 MAX7219 針腳內有一支為「DOUT」，若使用多模組串聯時，此腳位需連接到下一個模組的「DIN」腳位，其餘腳位接法同第一個單元，以現成的 MAX7219+8X8 LED 模組為例，多個單元串聯可以參考圖 5：

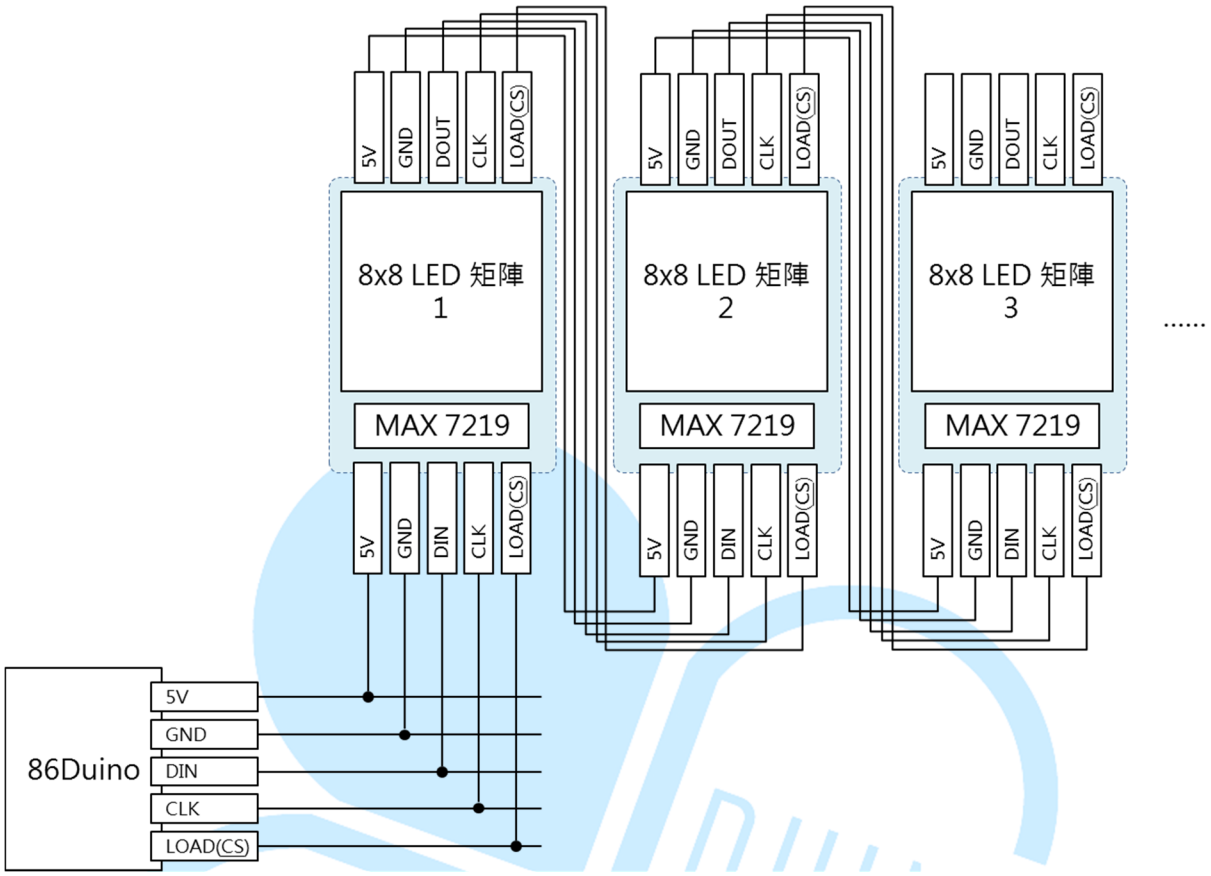


圖 5. 86Duino EduCake + MAX7219 多單元串聯架構圖

MAX7219 這個 LED 矩陣控制 IC，使用時需利用 DIN、CLK、LOAD 這三支針腳作控制；其實這是類似稱為「SPI」通訊介面的一部份，不過 86Duino EduCake 的麵包板上沒有預留 SPI 的硬體接腳插槽(實際 CPU 有支援，只是沒有拉出線路)，但我們依然可以透過軟體模擬的方式做 SPI 通訊。DIN 腳位負責送出序列的資料，CLK 腳位送出同步時脈，而 LOAD 腳位用來 disable/enable 線路上的 MAX7219 裝置。

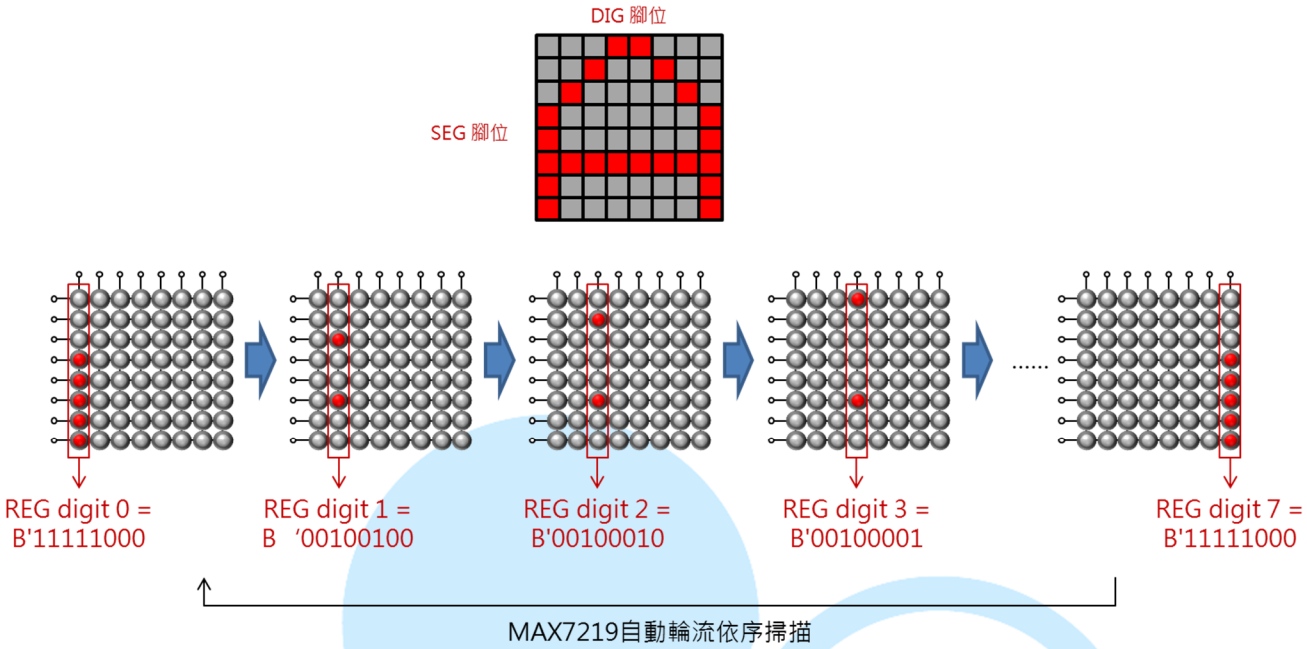
使用 MAX7219 時，主控制器需透過 DIN 針腳，對 IC 內部的幾個暫存器（Register）作寫入設定，MAX7219 的暫存器定義如下表：

暫存器名稱	位址	定義
No-Op	0xX0	無動作
Digit 0	0xX1	Digit 0 腳位對應的行/列資料值
Digit 1	0xX2	Digit 1 腳位對應的行/列資料值
Digit 2	0xX3	Digit 2 腳位對應的行/列資料值
Digit 3	0xX4	Digit 3 腳位對應的行/列資料值
Digit 4	0xX5	Digit 4 腳位對應的行/列資料值
Digit 5	0xX6	Digit 5 腳位對應的行/列資料值
Digit 6	0xX7	Digit 6 腳位對應的行/列資料值
Digit 7	0xX8	Digit 7 腳位對應的行/列資料值
Decode Mode	0xX9	設定使否啟用資料解碼模式
Intensity	0xXA	亮度控制
Scan Limit	0xXB	設定 Digit 0~Digit 7 掃描範圍
Shutdown	0xXC	設定是否關閉 LED 矩陣輸出
Display Test	0xXF	測試模式

Digit 0~7 的資料部分，從 MSB~LSB 分別對應 SEG DP、SEG A、SEG B、SEG C、SEG D、SEG E、SEG F、SEG G 腳位接的 LED，要亮的 LED 位置位元為 HIGH，反之熄滅為 LOW；若讀者自行使用 MAX7219 IC 加上自製的周邊 LED 或燈泡，須注意各種腳位與資料的對應關係喔。

讀者若仔細觀察 MAX7219 的針腳名稱，會發現其實這顆 IC 除了用在控制單一個 8X8 LED 矩陣外，也可以用來控制 8 個 7 段顯示器（一個 7 段顯示器有 8 個 LED，一般稱為：DP、A、B、C、D、E、F、G），所以讀者學會使用控制 LED 矩陣後，也可以將相同原理用在控制整排的 7 段顯示器用途。

一般使用上，只會需要設定暫存器 Digit 0~Digit 7 的內容，便可以控制 LED 矩陣的顯示圖樣；以本文選用的 MAX7219+LED 矩陣模組來說，IC 朝下方擺設時，Digit 0 對應的是矩陣最左邊第一行，Digit 1 對應左邊第二行，依此類推。而 Digit 0 暫存器的內容有 8 位元，每個位元剛好對應某一行的亮度模式;這裡選用的模組內定接線方式,由下而上是 SEG DP、SEG A、SEG B、...、SEG G。原理如圖 5 流程：



前面提到「掃描」的工作就交給 MAX7219 自動去完成，因此只要將 Digit 0~7 的暫存器內容設定完成，LED 矩陣便可以顯示出某個圖案。當然，定期更改圖案內容，就可以出現動畫了。若讀者自行製作其他接線架構的 LED 矩陣，需要注意一下暫存器資料與對應 LED 的位置關係，才會出現如同預期的圖案喔。

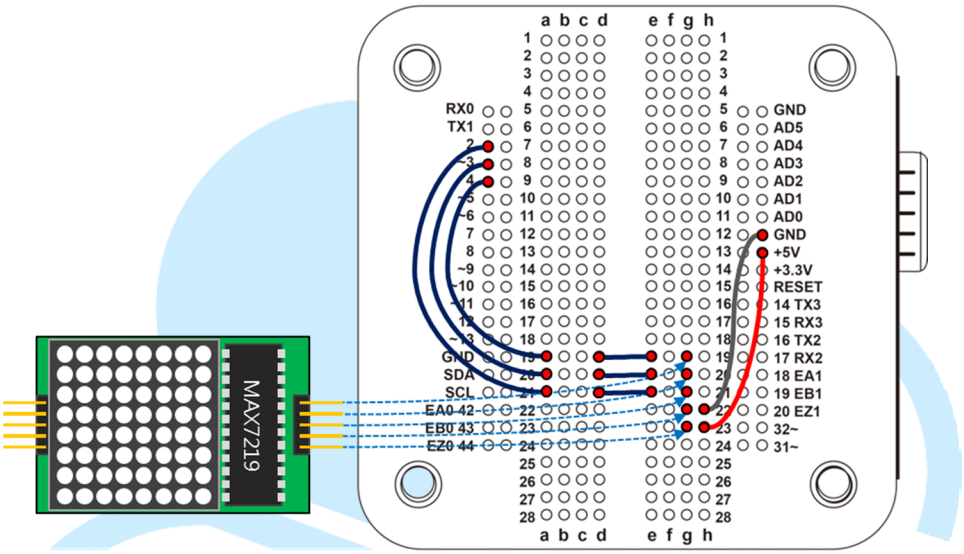
若需要更詳細的規格資訊，可參考 MAX7219 IC 說明文件：

http://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf

下面便讓我們利用程式，實際練習上面所講解的 MAX7219 控制原理，並利用 LED 矩陣做實際顯示圖樣的功能吧。

二、 第一個程式 – 單一 LED 矩陣練習 1

第一個範例程式先來練習如何使用 86Duino EduCake 與 MAX7219 模組的通訊，控制一組 8X8 LED 矩陣的圖案，讀者請先依下圖接線：



接著請打開 86Duino Coding IDE，輸入以下程式碼：

```
// 控制腳位定義
int DIN_pin = 2;
int LOAD_pin = 3;
int CLOCK_pin = 4;

// MAX7219 暫存器位址
byte max7219_REG_noop        = 0x00;
byte max7219_REG_digit0      = 0x01;
byte max7219_REG_digit1      = 0x02;
byte max7219_REG_digit2      = 0x03;
byte max7219_REG_digit3      = 0x04;
byte max7219_REG_digit4      = 0x05;
byte max7219_REG_digit5      = 0x06;
byte max7219_REG_digit6      = 0x07;
byte max7219_REG_digit7      = 0x08;
```

```
byte max7219_REG_decodeMode  = 0x09;
byte max7219_REG_intensity   = 0x0a;
byte max7219_REG_scanLimit   = 0x0b;
byte max7219_REG_shutdown    = 0x0c;
byte max7219_REG_displayTest = 0x0f;

void SPI_SendByte(byte data) {// 模擬 SPI 介面依序送出 byte 資料
  byte i = 8;
  byte mask;
  while(i > 0) {
    mask = 0x01 << (i - 1);// 製造位元遮罩，從最左邊位元開始
    digitalWrite( CLOCK_pin, LOW);// 時脈同步線 = LOW
    if (data & mask){// 判斷位元遮罩對應的位元是 0 或 1
      digitalWrite(DIN_pin, HIGH);// 如果對應位元為 1，DIN 送出 HIGH
    }
    else{
      digitalWrite(DIN_pin, LOW);// 如果對應位元為 1，DIN 送出 LOW
    }
    digitalWrite(CLOCK_pin, HIGH);// 時脈同步線 = HIGH
    --i;// 移到下一個位元
  }
}

void MAX7219_1Unit(byte reg_addr, byte reg_data) {// 控制單一個 MAX7219 模組
  digitalWrite(LOAD_pin, LOW);// 傳送前 LOAD 腳要先變成 LOW
  SPI_SendByte(reg_addr);// 先送出要設定的暫存器位址
  SPI_SendByte(reg_data);// 接著送出資料
  digitalWrite(LOAD_pin,HIGH);// 傳送完 LOAD 腳要變成 HIGH
}
```

```
    byte matrixData_8X8[8] = {//  圖樣資料矩陣
      B01010101,//  第一行由下而上
      B10000001,
      B10101010,
      B11111111,
      B00000000,
      B11110000,
      B00001111,
      B11001100
    };

    void Draw (byte *LED_matrix)//  靜態繪製整個畫面
    {
      MAX7219_1Unit(1, LED_matrix[0]);
      MAX7219_1Unit(2, LED_matrix[1]);
      MAX7219_1Unit(3, LED_matrix[2]);
      MAX7219_1Unit(4, LED_matrix[3]);
      MAX7219_1Unit(5, LED_matrix[4]);
      MAX7219_1Unit(6, LED_matrix[5]);
      MAX7219_1Unit(7, LED_matrix[6]);
      MAX7219_1Unit(8, LED_matrix[7]);
    }

    void setup ( ) {
      pinMode(DIN_pin, OUTPUT);
      pinMode(CLOCK_pin, OUTPUT);
      pinMode(LOAD_pin, OUTPUT);

      digitalWrite(CLOCK_pin, HIGH);

      //  初始化  MAX7219  的暫存器
      MAX7219_1Unit(max7219_REG_scanLimit,  0x07);//  設定為掃
描所有行
```

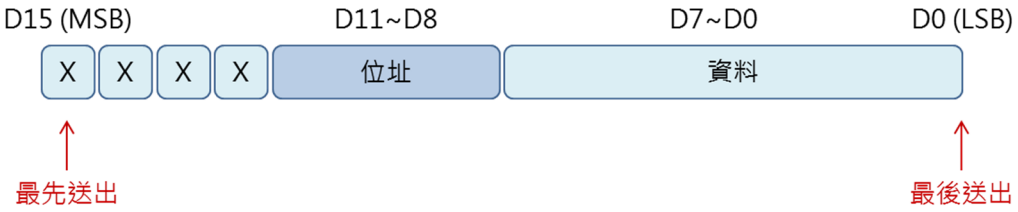
```
    MAX7219_1Unit(max7219_REG_decodeMode,  0x00);// 不使用解碼模式
    MAX7219_1Unit(max7219_REG_shutdown, 0x01);// 設定為不在關閉模式
    MAX7219_1Unit(max7219_REG_displayTest, 0x00); // 設定為不在測試模式

    for(int i=1; i<=8; i++) {// 先把所有 LED 矩陣變暗
      MAX7219_1Unit(i,0);
    }

    MAX7219_1Unit(max7219_REG_intensity, 0x0f);// 設定亮度範圍，0x00 ~ 0x0f

    delay(1000);
  }

  void loop ( ) {
    Draw(matrixData_8X8);
    delay(500);
  }
```

燒錄完成後，讀者將會看到如下圖的靜態圖樣：

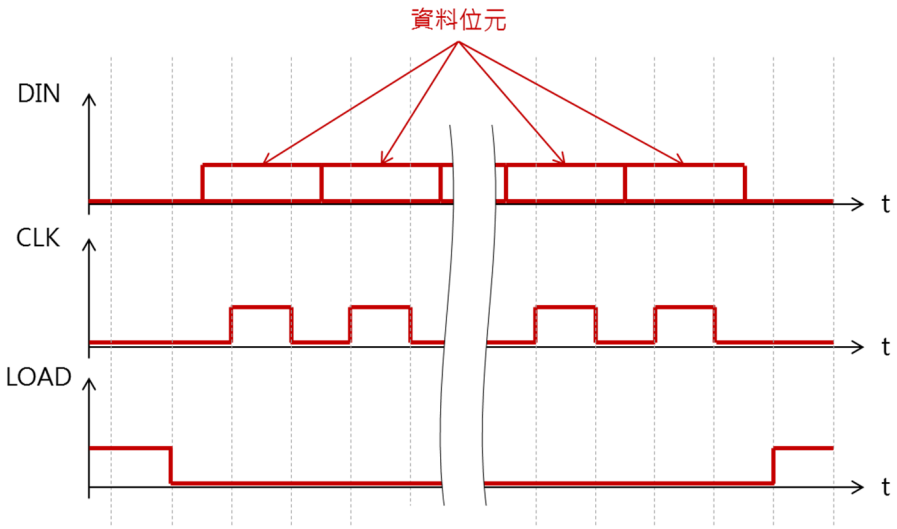
此範例程式功能為，設定一次 MAX7219 的 Digit 0~7 暫存器內容，讓 LED 矩陣顯示出單一圖樣；可使用這圖樣快速找出 LED 矩陣模組擺放方向與資料定義的關係。

程式一開始先設定各個控制腳位的編號，以及 MAX7219 所有暫存器的位址，接著 SPI_SendByte(byte data)這個功能用來模擬 SPI 傳輸一個 byte 的資料、MAX7219_1Unit(byte reg_addr, byte reg_data)則用在寫入一個 MAX7219 單元的特定設定資料。

MAX7219_1Unit( )函式每一次對暫存器寫入的資料單位為 16 位元，其中 D0~D7 為資料位元，D8~D11 為位址，D12~D15 沒有定義，序列資料以 D15(MSB)→D0(LSB)的順序送出，如下圖示意：



DIN 資料線與 CLK 時脈同步線的波形如下：



當 DIN 資料準備好時，CLK 需變為 HIGH，然後變為 LOW；而 LOAD 線則在傳送前設為 LOAD，傳送完變成 HIGH。matrixData_8X8[8]矩陣用來儲存某個圖樣的資料，0 為 LED 熄滅，1 為 LED 點亮。

Draw( )函式內，由於需繪製整個 8X8 矩陣的畫面，因此使用語法 MAX7219_1Unit( )，分別設定 8 行的數據即可完成畫面的圖案設定。

setup( )內除了設定使用腳位的 I/O 模式與初始值外，另外初始化 MAX7219 的許多暫存器，這些暫存器需要初始化才能夠正常運作。而 loop( ) 內則是定時 500ms 呼叫 Draw( )函式。此處用來當作顯示資料的 matrixData_8X8 矩陣內容一直保持不變，當然圖樣也就一直保持靜態囉。

三、 第二個程式 – 單一 LED 矩陣練習 2

了解 86Duino EduCake 控制 MAX7219 與 LED 矩陣的基本原理後，接著來做點小變化，讓圖案變成動畫效果吧。讀者請打開 86Duino Coding IDE，接著在上一個範例程式中加入以下程式碼：

```
int shift = 0;
void ShiftDraw(byte *LED_matrix)// 位移繪製整個畫面
{
  MAX7219_1Unit(1, LED_matrix[(shift) % 8]);// 繪製第 1 行的資料
  MAX7219_1Unit(2, LED_matrix[(shift+1) % 8]);// 繪製第 2 行的資料
  MAX7219_1Unit(3, LED_matrix[(shift+2) % 8]);// 繪製第 3 行的資料
  MAX7219_1Unit(4, LED_matrix[(shift+3) % 8]);// 繪製第 4 行的資料
  MAX7219_1Unit(5, LED_matrix[(shift+4) % 8]);// 繪製第 5 行的資料
  MAX7219_1Unit(6, LED_matrix[(shift+5) % 8]);// 繪製第 6 行的資料
  MAX7219_1Unit(7, LED_matrix[(shift+6) % 8]);// 繪製第 7 行的資料
  MAX7219_1Unit(8, LED_matrix[(shift+7) % 8]);// 繪製第 8 行的資料

  shift++;
  if(shift>=8){// 讓索引在 0~7 循環
    shift = 0;
  }
}
```

然後將原本 loop( ) 中的「 Draw(matrixData_8X8); 」改成「ShiftDraw(matrixData_8X8);」便能夠讓上個範例中的圖案依行不停往左位移循環囉。ShiftDraw( )這裡使用的原理是，使用一個索引變數 shift，用 shift 作為設定 LED 各行亮度資料的索引值；shift 每次加 1，因此會依序對應到 matrixData_8X8 矩陣中不同的位置，就可以達成位移圖案的效果。這裡須注意 matrixData_8X8 矩陣只有索引 0~7 的範圍是圖案資料，所以實

際計算索引值時須使用(shift+N) % 8 語法，將數值限制在 0~7 的範圍以免出錯喔。同樣 shift 變數在每次+1 之後，會檢查是否超出範圍，若超出範圍也需將變數歸 0，如此 shift 便會在 0~7 的數值內不停循環。讀者也可以嘗試將 matrixData_8X8 矩陣設定成不同的圖案、或是使用不同的索引變化來試試其他特效喔。

四、 第三個程式 – 單一 LED 矩陣練習 3

這個範例電路不用更改，我們繼續來加入其他的功能，讀者請打開 86Duino Coding IDE，在原本的「matrixData_8X8」下方輸入以下程式碼：

```
// 0
byte matrixData_num_0[8] = {// 圖樣資料矩陣
  B00000000,// 第一行由下而上
  B01111110,
  B10010001,
  B10001001,
  B10001001,
  B10000101,
  B01111110,
  B00000000
};
// 1
byte matrixData_num_1[8] = {// 圖樣 1 資料矩陣
  B00000000,// 第一行由下而上
  B00000000,
  B10000000,
  B10000010,
  B11111111,
  B10000000,
  B00000000,
```

```
    B00000000
    };
    // 2
    byte matrixData_num_2[8] = {//  圖樣  1  資料矩陣
      B00000000,//  第一行由下而上
      B10000110,
      B10000001,
      B11000001,
      B10100001,
      B10010001,
      B10001110,
      B00000000
    };
    // 3
    byte matrixData_num_3[8] = {//  圖樣  1  資料矩陣
      B00000000,//  第一行由下而上
      B01000010,
      B10001001,
      B10001001,
      B10001001,
      B10001001,
      B01110110,
      B00000000
    };
    // 4
    byte matrixData_num_4[8] = {//  圖樣  1  資料矩陣
      B00000000,//  第一行由下而上
      B00110000,
      B00101000,
      B00100100,
      B00100010,
      B11111111,
      B00100000,
```

```
    B00000000
};
// 5
byte matrixData_num_5[8] = {// 圖樣 1 資料矩陣
    B00000000,// 第一行由下而上
    B01001111,
    B10001001,
    B10001001,
    B10001001,
    B10001001,
    B01110011,
    B00000000
};
// 6
byte matrixData_num_6[8] = {// 圖樣 1 資料矩陣
    B00000000,// 第一行由下而上
    B01111110,
    B10001001,
    B10001001,
    B10001001,
    B10001001,
    B01110010,
    B00000000
};
// 7
byte matrixData_num_7[8] = {// 圖樣 1 資料矩陣
    B00000000,// 第一行由下而上
    B00000011,
    B00000001,
    B00000001,
    B11110001,
    B00001001,
    B00000111,
```

```
    B00000000
    };
    // 8
    byte matrixData_num_8[8] = {//  圖樣  1  資料矩陣
      B00000000,//  第一行由下而上
      B01110110,
      B10001001,
      B10001001,
      B10001001,
      B10001001,
      B01110110,
      B00000000
    };
    // 9
    byte matrixData_num_9[8] = {//  圖樣  1  資料矩陣
      B00000000,//  第一行由下而上
      B01001110,
      B10010001,
      B10010001,
      B10010001,
      B10010001,
      B01111110,
      B00000000
    };
```

然後在 setup( )裡面加入：

```
Serial.begin(115200);
```

接著將 loop( )內改為：

```
void loop ( ) {
```

```
if(Serial.available())// 檢查 Serial port 是否收到資料
{
  byte num = Serial.read();
  switch(num)// 針對不同數值顯示不同數字
  {
    case '0':
      Draw(matrixData_num_0);
      break;

    case '1':
      Draw(matrixData_num_1);
      break;

    case '2':
      Draw(matrixData_num_2);
      break;

    case '3':
      Draw(matrixData_num_3);
      break;

    case '4':
      Draw(matrixData_num_4);
      break;

    case '5':
      Draw(matrixData_num_5);
      break;

    case '6':
      Draw(matrixData_num_6);
      break;
```

```
            case '7':
                Draw(matrixData_num_7);
                break;

             case '8':
                Draw(matrixData_num_8);
                break;

             case '9':
                Draw(matrixData_num_9);
                break;

             default:
                break;
          }
        }
        delay(100);
    }
```

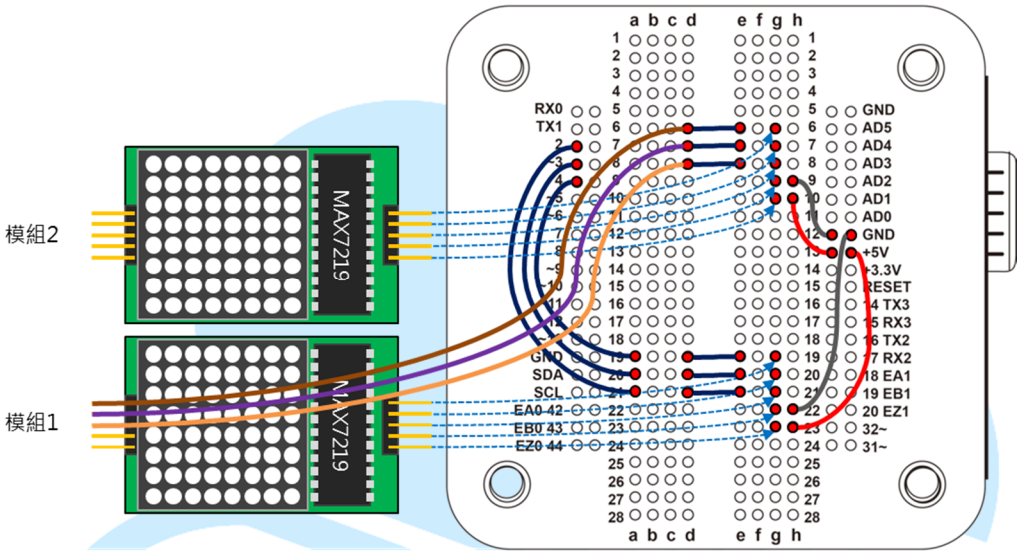
編譯並上傳程式後，請打開 Serial Monitor，注意 baudrate 要設定成跟程式內一樣；接著可以輸入 0~9 任一數字，86Duino EduCake 接的 LED 矩陣便會顯示對應的數字圖案囉。

一開始在程式內加入的幾個 byte 矩陣 matrixData_num_0~matrixData_num_9 便是數字 0~9 的文字圖樣；此程式功能為，利用 Serial Monitor 通訊功能，當使用者輸入數字時，86Duino EduCake 便會判斷接收數字字元為何；接著使用 switch-case 語法，針對不同的字元資料，控制 MAX7219 在 LED 矩陣上顯示出對應的文字圖樣。讀者也可以自行增加通訊可用的字元，並修改自己喜歡的圖樣做顯示喔。

五、 第四個程式 – 控制多個 LED 矩陣 1

練習過控制單個 MAX7219+LED 模組後，繼續來練習多個單元的串聯，

讀者請加入下列接線：



接著請打開 86Duino Coding IDE，在 int CLOCK_pin = 4;之後加入：

```
int MAX7219_units = 2;// 設定使用的 MAX7219 串聯單元數量
```

在 void MAX7219_1Unit( )函式下方加入：

```
    void MAX7219_AllUnit( byte reg_addr, byte reg_data ) {// 控制所有串聯的 MAX7219 模組，寫入同樣資料
      digitalWrite( LOAD_pin, LOW );// 傳送前 LOAD 腳要先變成 LOW
      for (int c =1; c<= MAX7219_units; c++) {
        SPI_SendByte(reg_addr);// 先送出要設定的暫存器位址
        SPI_SendByte(reg_data);// 接著送出資料
      }
      digitalWrite(LOAD_pin,HIGH);// 傳送完 LOAD 腳要變成 HIGH
    }
    void MAX7219_indexUnit( byte unit_index, byte reg_addr, byte reg_data ) {// 控制所有串聯中的某一個 MAX7219 模組，其餘模組圖樣不變
      int c = 0;
      digitalWrite(LOAD_pin, LOW);// 傳送前 LOAD 腳要先變成 LOW
      // 從串聯最尾端的模組開始控制
      for ( c = MAX7219_units; c > unit_index; c--) {
        SPI_SendByte(0);// NO-OP 暫存器
        SPI_SendByte(0);// 資料 = 0
      }

      SPI_SendByte(reg_addr);// 先送出要設定的暫存器位址
      SPI_SendByte(reg_data);// 接著送出資料

      for ( c = unit_index-1; c >= 1; c--) {
        SPI_SendByte(0);// NO-OP 暫存器
        SPI_SendByte(0);// 資料 = 0
      }
      digitalWrite(LOAD_pin,HIGH);// 傳送完 LOAD 腳要變成 HIGH
    }
```

在 Draw( )之後加入：

```
void Draw_Unit( byte index, byte *LED_matrix )// 靜態繪製特定單元的圖樣
{
  MAX7219_indexUnit(index, 1, LED_matrix[0]);
  MAX7219_indexUnit(index, 2, LED_matrix[1]);
  MAX7219_indexUnit(index, 3, LED_matrix[2]);
  MAX7219_indexUnit(index, 4, LED_matrix[3]);
  MAX7219_indexUnit(index, 5, LED_matrix[4]);
  MAX7219_indexUnit(index, 6, LED_matrix[5]);
  MAX7219_indexUnit(index, 7, LED_matrix[6]);
  MAX7219_indexUnit(index, 8, LED_matrix[7]);
}
```

```
void setup ( ) {
  pinMode(DIN_pin, OUTPUT);
  pinMode(CLOCK_pin, OUTPUT);
  pinMode(LOAD_pin, OUTPUT);

  digitalWrite(CLOCK_pin, HIGH);

  // 初始化所有 MAX7219 的暫存器
  MAX7219_AllUnit( max7219_REG_scanLimit, 0x07 );// 設定為掃描所有行
  MAX7219_AllUnit( max7219_REG_decodeMode, 0x00 );// 不使用解碼模式
  MAX7219_AllUnit( max7219_REG_shutdown, 0x01 );// 設定為不在關閉模式
  MAX7219_AllUnit( max7219_REG_displayTest, 0x00 ); // 設定為不在測試模式
```

修改 setup( )內容為：

```
        for( int i=1; i<=8; i++ ) {// 先把所有 LED 矩陣變暗
          MAX7219_AllUnit(i,0);
        }

        MAX7219_AllUnit( max7219_REG_intensity, 0x0f );// 設定亮
度範圍，0x00 ~ 0x0f

        delay(1000);
      }
```

以及修改 loop( )內容為：

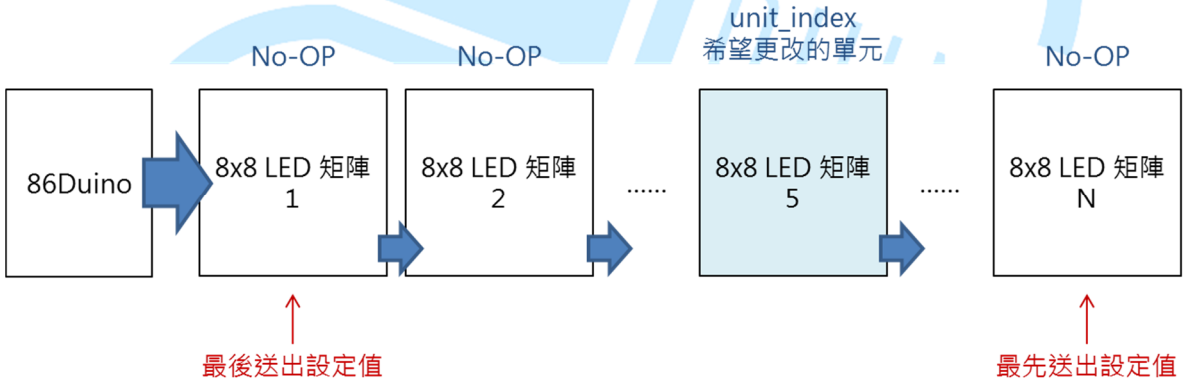
```
void loop () {
  Draw_Unit(1, matrixData_num_0);
  Draw_Unit(2, matrixData_num_1);
  delay(500);
}
```

此範例程式從前一個進行修改，加入支援多個 MAX7219 模組的函式「 MAX7219_AllUnit(byte reg_addr, byte reg_data) 」、「 MAX7219_indexUnit(byte unit_index, byte reg_addr, byte reg_data)」，以及「Draw_Unit(byte index, byte *LED_matrix)」。

如前面提過的，由於 MAX7219 可以多個模組串聯，且前一個模組的 DIN 腳位資料會被 DOUT 傳遞到下一個串聯模組去，所以有多個 MAX7219 模組時，需要針對每個模組作暫存器設定才行。所以前面新宣告一個 int MAX7219_units 變數，用在設定使用的 MAX7219 串聯模組數量，使用者須視實際使用的模組數量狀況，設定這個數值；這裡因為使用兩個模組所以設定為 2。

MAX7219_AllUnit( ) 這個函式基本上流程跟原本的「MAX7219_1Unit( )」差不多，但因為需要對每個 MAX7219 單元寫入相同資料，所以 LOAD_pin 設定為 LOW 之後，中間使用 for 迴圈，對每個 MAX7219 的同個暫存器寫入相同數值，再將 LOAD_pin 設為 HIGH。

MAX7219_indexUnit( )跟上一個功能剛好相反，用在分別設定「不同資料」給特定的 MAX7219 單元；函式的引數多了一個 byte unit_index，用在指定希望更改的模組；由於送出設定資料會被傳遞到下一個模組，所以送出資料順序是從「串聯的最尾端模組」開始，如果有 N 個模組則須送出 N 次，如下圖所示：



只有 unit_index 指定的模組需要更改暫存器資料，其餘都不要更動，這要怎麼辦呢？前面 MAX7219 暫存器表格裡的「No-Op」暫存器這時就派上用場了。如上圖說明，當需要在 N 個串聯模組內更改第 5 個模組時，第 1~4 以及第 6~N 的模組都須在 No-Op 寫入數值，但寫入 No-Op 不會影響此單元的動作。MAX7219_indexUnit( )函式便是在做這個過程；注意跟 MAX7219_AllUnit( )相同，LOAD_pin 都是寫入前為 LOW，全部資料寫完後變為 HIGH，中間是沒有變化的。

Draw_Unit(byte index, byte *LED_matrix)這個函式則是為了在特定模組畫出圖案，但不更改其他模組的圖樣；所以裡面使用MAX7219_indexUnit( )函數針對 index 對應的 MAX7219 做設定便可達到功能。

setup( )裡面初始化的部分，由於有 N 個模組皆須做相同的初始化，所以直接把原本的 MAX7219_1Unit( )改用 MAX7219_AllUnit( )即可；而 loop( )裡面使用 Draw_Unit(1, matrixData_num_0)、Draw_Unit(2, matrixData_num_1)，在串聯的第 1 個模組繪出圖樣數字 0，第 2 個模組繪出圖樣數字 1。讀者也可以自行更改圖樣內容、繪出的單元位置等等試試其他效果喔。

六、 第五個程式 – 控制多個 LED 矩陣 2

最後一個練習程式，不更改接線，直接由第 4 個程式做點小變化，加入動態跑馬燈效果；讀者請在原程式內加入以下程式碼：

```
// 86Duino EduCake
const unsigned int string_len = 70;
byte  matrixData_86Duino_EduCake[string_len]  =  {//  圖樣資料矩陣
    B01110110,// 8
    B10001001,
    B10001001,
    B01110110,
    B00000000,

    B01111110,// 6
    B10001001,
```

```
B10001001,
B01110010,
B00000000,

B11111111,// D
B10000001,
B10000001,
B01111110,
B00000000,

B01111000,// u
B10000000,
B01000000,
B11111000,
B00000000,

B11111010,// i
B00000000,

B11111000,// n
B00010000,
B00001000,
B11110000,
B00000000,

B01110000,// o
B10001000,
B10001000,
B01110000,
B00000000,

B00000000,
```

```
B11111111,// E
 B10001001,
 B10001001,
 B10000001,
 B00000000,

 B01110000,// d
 B10001000,
 B10001000,
 B11111111,
 B00000000,

 B01111000,// u
 B10000000,
 B01000000,
 B11111000,
 B00000000,

 B01111110,// C
 B10000001,
 B10000001,
 B01100110,
 B00000000,

 B01100100,// a
 B10010100,
 B10010100,
 B11111000,
 B00000000,

 B11111111,// k
 B00100000,
 B01010000,
```

```
    B10001000,
      B00000000,

      B01110000,// e
      B10101000,
      B10101000,
      B10110000,

      B00000000,
      B00000000,
      B00000000
    };

    int shift = 0;
    void ShiftDraw_2Unit(byte *LED_matrix)// 位移繪製整個畫面
    {
      // Unit 1
      MAX7219_indexUnit(1, 1, LED_matrix[(shift) % string_len]);// 繪製第 1 行的資料
      MAX7219_indexUnit(1, 2, LED_matrix[(shift+1) % string_len]);// 繪製第 2 行的資料
      MAX7219_indexUnit(1, 3, LED_matrix[(shift+2) % string_len]);// 繪製第 3 行的資料
      MAX7219_indexUnit(1, 4, LED_matrix[(shift+3) % string_len]);// 繪製第 4 行的資料
      MAX7219_indexUnit(1, 5, LED_matrix[(shift+4) % string_len]);// 繪製第 5 行的資料
      MAX7219_indexUnit(1, 6, LED_matrix[(shift+5) % string_len]);// 繪製第 6 行的資料
      MAX7219_indexUnit(1, 7, LED_matrix[(shift+6) % string_len]);// 繪製第 7 行的資料
      MAX7219_indexUnit(1, 8, LED_matrix[(shift+7) % string_len]);// 繪製第 8 行的資料
```

```
        // Unit 2
          MAX7219_indexUnit(2,      1,      LED_matrix[(shift+8)      %
string_len]);//  繪製第 1 行的資料
          MAX7219_indexUnit(2,      2,      LED_matrix[(shift+9)      %
string_len]);//  繪製第 2 行的資料
          MAX7219_indexUnit(2,      3,     LED_matrix[(shift+10)      %
string_len]);//  繪製第 3 行的資料
          MAX7219_indexUnit(2,      4,     LED_matrix[(shift+11)      %
string_len]);//  繪製第 4 行的資料
          MAX7219_indexUnit(2,      5,     LED_matrix[(shift+12)      %
string_len]);//  繪製第 5 行的資料
          MAX7219_indexUnit(2,      6,     LED_matrix[(shift+13)      %
string_len]);//  繪製第 6 行的資料
          MAX7219_indexUnit(2,      7,     LED_matrix[(shift+14)      %
string_len]);//  繪製第 7 行的資料
          MAX7219_indexUnit(2,      8,     LED_matrix[(shift+15)      %
string_len]);//  繪製第 8 行的資料

          shift++;
          if(shift>=string_len){//  讓索引在 0~61 循環
            shift = 0;
          }
        }
```

然後將 loop ( )改成：

```
        void loop ( )
          ShiftDraw_2Unit(matrixData_86Duino_EduCake);
          delay(100);
        }
```

上傳程式完成後，讀者便可以看到「86Duino EduCake」的字樣不停循環顯示，就像一般廣告跑馬燈效果一樣。ShiftDraw_2Unit( )這個

函式基本上跟 ShiftDraw( )差不多，只是用來當作圖樣的 byte 矩陣變長了而已；範例這個圖樣矩陣共有 70 列的長度，所以函式內的索引變數只能在 0~69 內循環，這裡使用 const unsigned int string_len 作為圖樣矩陣列數長度，方便程式撰寫與修改。當讀者擁有更多個 MAX7219 模組時，也可以用相同的概念去做延伸囉。