EduCake 使用 LED 矩阵

一、 LED 矩阵原理介绍

之前曾经介绍过利用 86Duino EduCake 的数字/模拟输出功能控制单颗或少数的 LED，但这种针脚一对一控制 LED 的方式，缺点是需要占用较多的控制器脚位，若项目里其他功能也需要这些脚位就不方便了。读者应该想知道，如果希望一次控制数十上百颗的 LED、灯泡等等，有甚么方法可以用呢？本章节就来介绍利用 86Duino EduCake 控制多个 LED 矩阵的方法，读者将可以学到使用 LED 矩阵做出图样显、文字显示等等功能的原理，就像车站、广告广告牌常看到的 LED 跑马灯那样。

一般市面上可以买到的 LED 矩阵相当多样化，但主要分为共阳极跟共阴极两种，以 8X8 的 LED 矩阵为例，结构如下面图 1、图 2：

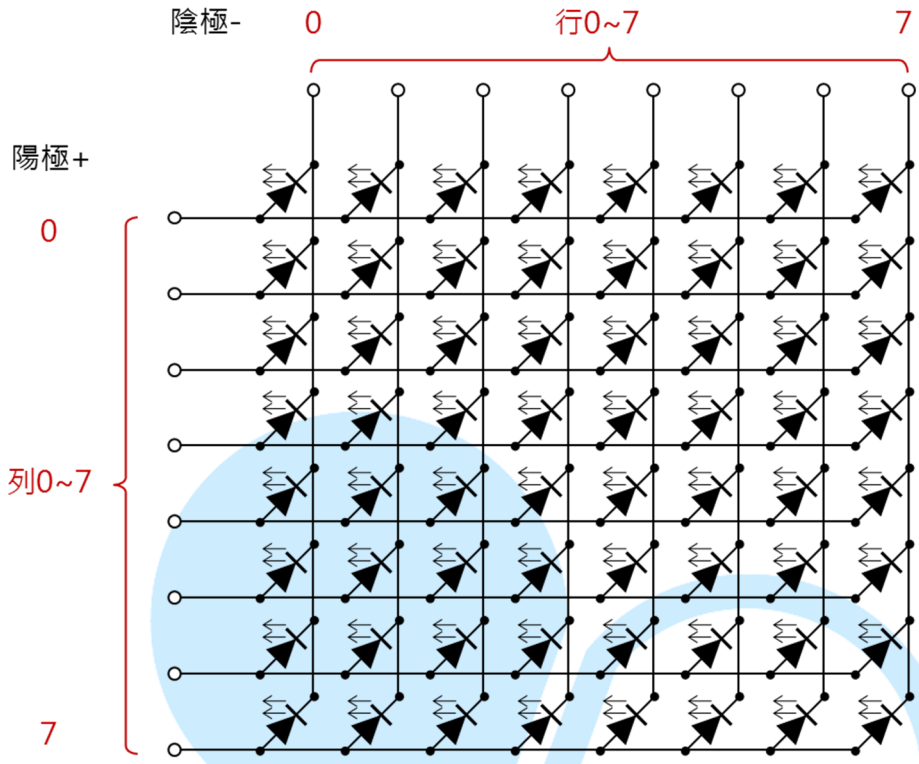


图 1. 共阳极 LED 矩阵

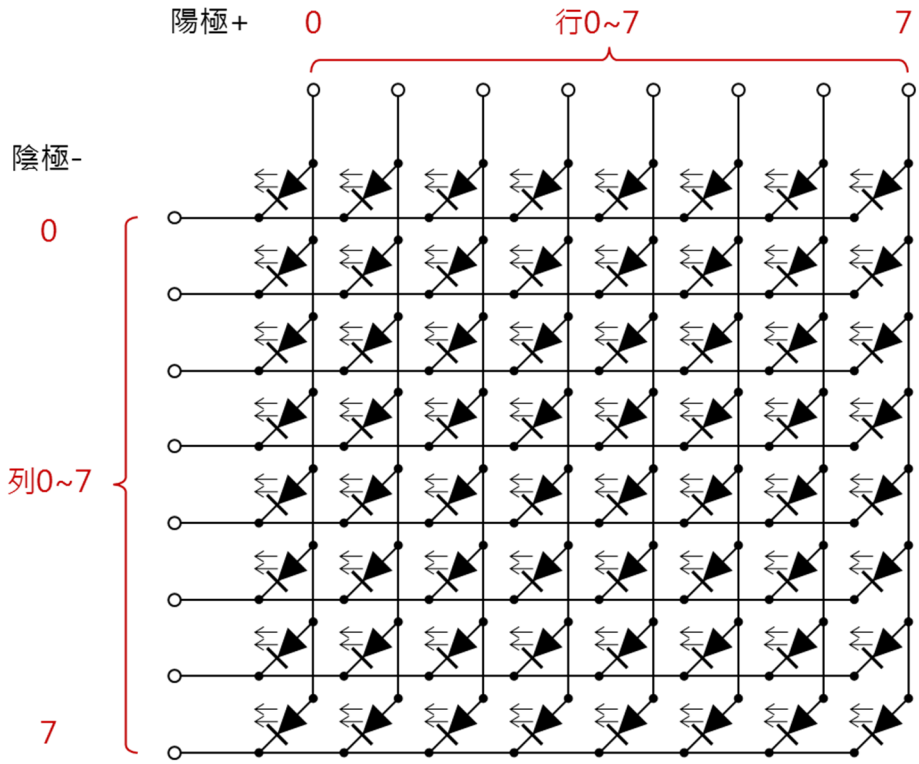


图 2. 共阴极 LED 矩阵

从上面的图片可以看到，所谓共阳极或共阴极表示有一整排的 LED 阴极或阳极是连接在同一个点上（事实上对于单色的 LED 矩阵来说，共阳极或共阴极实际上只是摆放方向的不同，多色 LED 就会有差异）；这种架构的 LED 矩阵便无法像以前一样单颗进行控制，必须利用「扫描+视觉暂留」的方式来控制。如下图 3 所示，一次点亮一排 LED 中的某几颗，再依序换成下一排，依这样的原理显示整片画面。

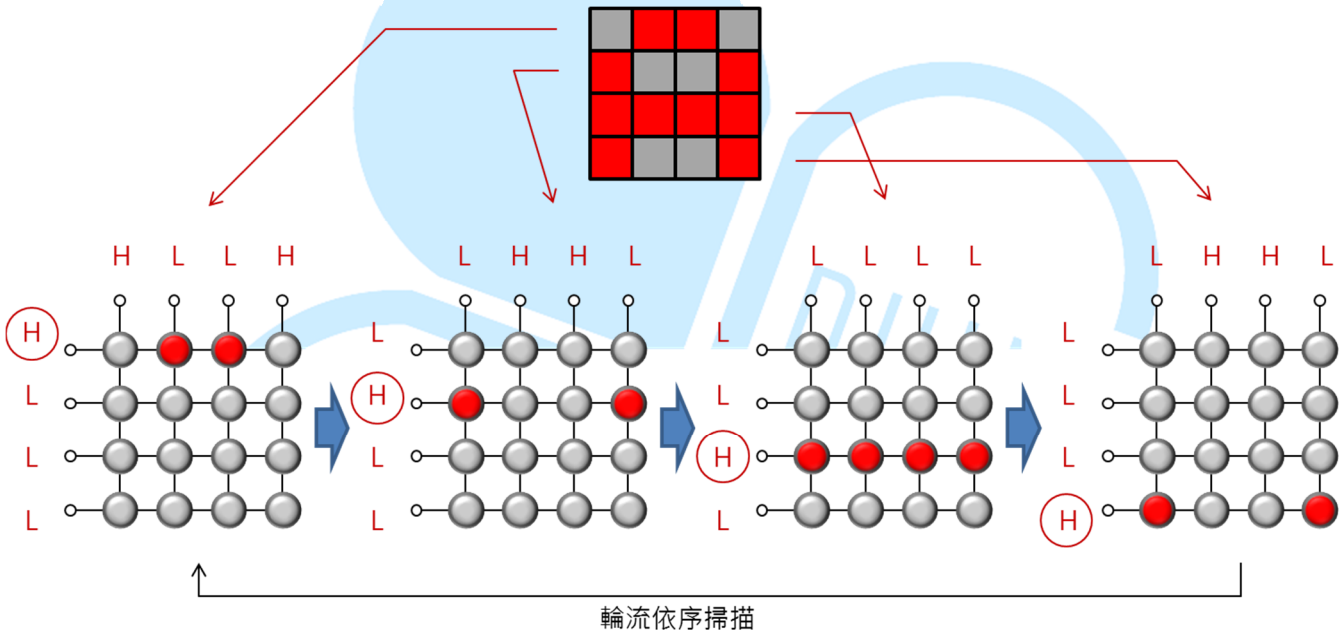


图 3. LED 矩阵扫描控制原理图

这种扫描式的 LED 控制可以大幅减少需要的控制针脚数，如果单纯使用 86Duino EduCake 做上述的控制，则只需要 16 只针脚跟一些电阻即可，是不是比利用 64 只针脚控制来的有效率呢？但即使控制一个 8X8 LED 矩阵只占用 16 只脚，实际应用上还是有诸多不便；除了造成程序变得复杂外，如果需要控制多个 LED 矩阵，那么同样的占用脚位问题又会重复出现。

还好市面上已经有专用的 LED 控制 IC，可以自动帮使用者作上述的「扫描」工作，程序代码只需对 IC 做简单的设定，便可简易进行 LED 矩阵控制。本章节将会介绍如何使用 86Duino EduCake 与 LED 矩阵控制 IC 做搭配，来控制单个与多个 8X8 LED 矩阵；这里使用的 LED 控制 IC 型号为 MAX7219，接线与架构图如下图 4：

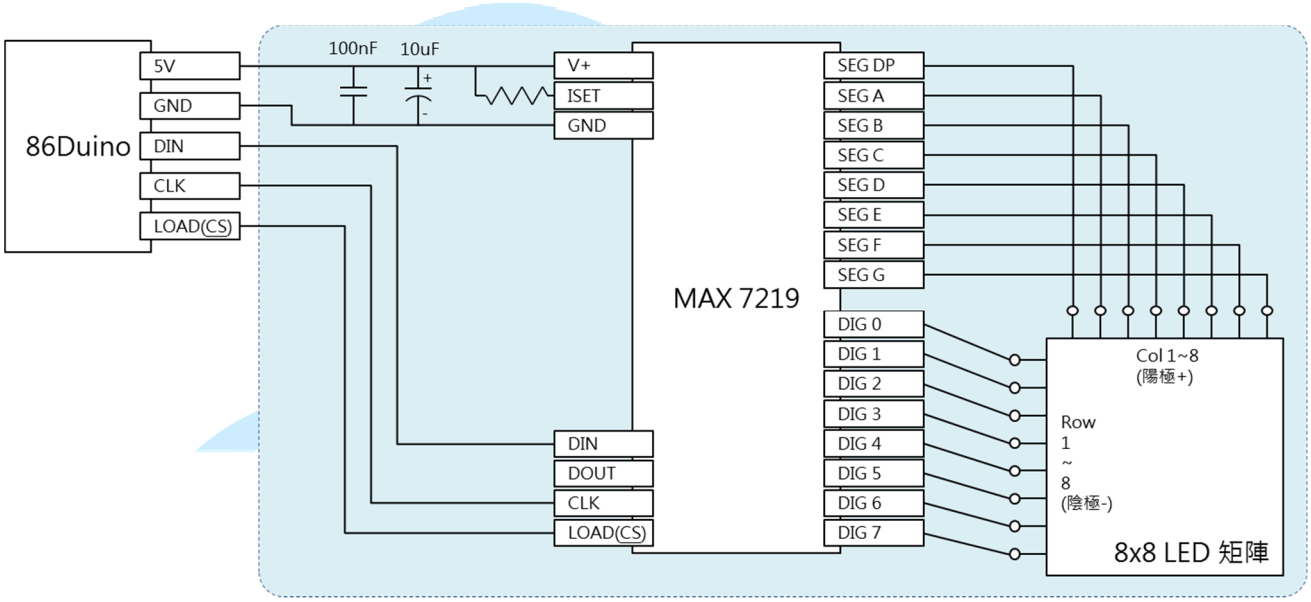


图 4. 86Duino EduCake + MAX7219 架构图

从上图可以看到，MAX7219 与 86Duino EduCake 之间仅需 3 条控制讯号线，其余两条为电源线；读者可以参考上图做接线，或者也可购买市面上的整合模块（模块已包含图中篮框内所有组件）。使用 MAX7219 除了控制讯号相当精简外，也可以多个单元进行串联，完全不用再消耗主控制器的宝贵针脚空间。图 4 中可看到 MAX7219 针脚内有一支为「DOUT」，若使用多模块串联时，此脚位需连接到下一个模块的「DIN」脚位，其余脚位接法同第一个单元，以现成的 MAX7219+8X8 LED 模块为例，多个单元串联可以参考图 5：

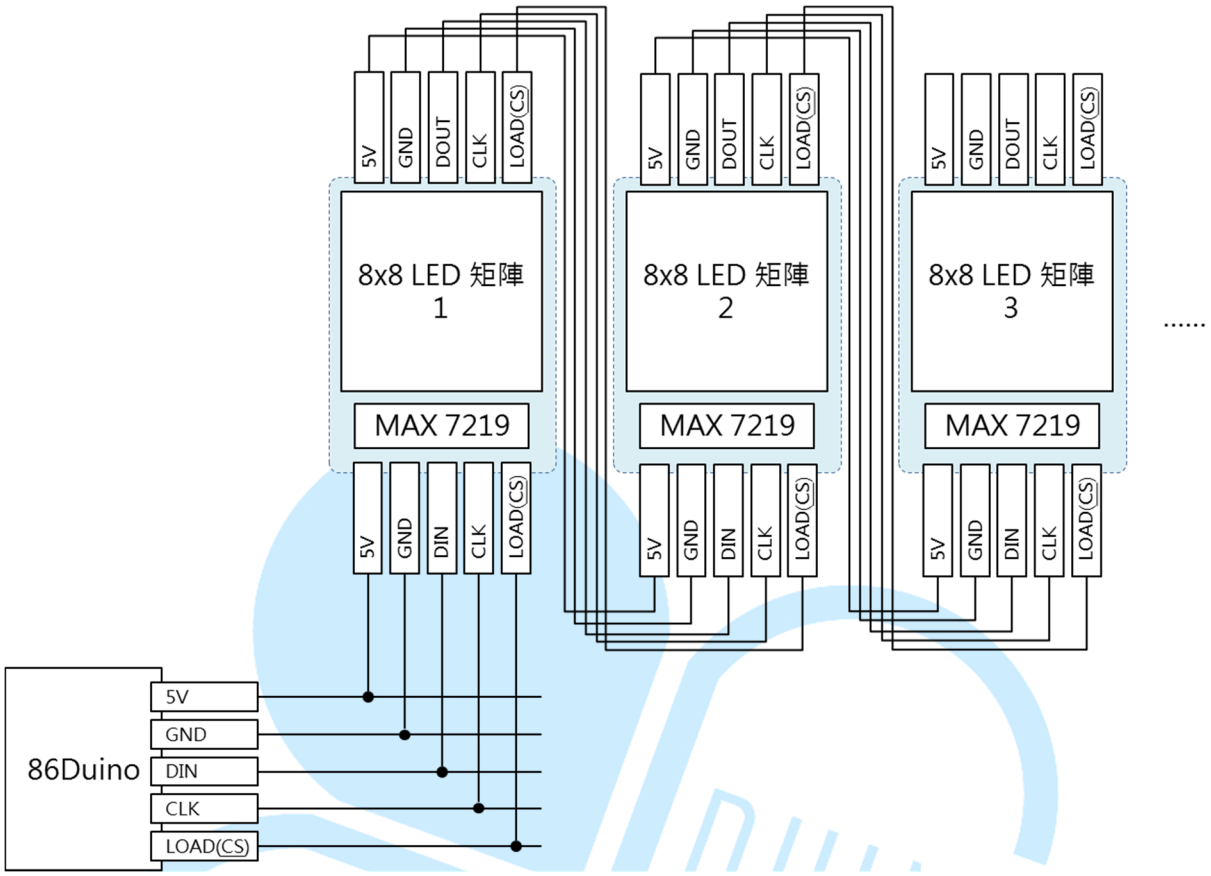


图 5. 86Duino EduCake + MAX7219 多单元串联架构图

MAX7219 这个 LED 矩阵控制 IC，使用时需利用 DIN、CLK、LOAD 这三支针脚作控制；其实这是类似称为「SPI」通讯接口的一部份，不过 86Duino EduCake 的面包板上没有预留 SPI 的硬件接脚插槽(实际 CPU 有支持，只是没有拉出线路），但我们依然可以透过软件仿真的方式做 SPI 通讯。DIN 脚位负责送出序列的资料，CLK 脚位送出同步频率，而 LOAD 脚位用来 disable/enable 线路上的 MAX7219 装置。

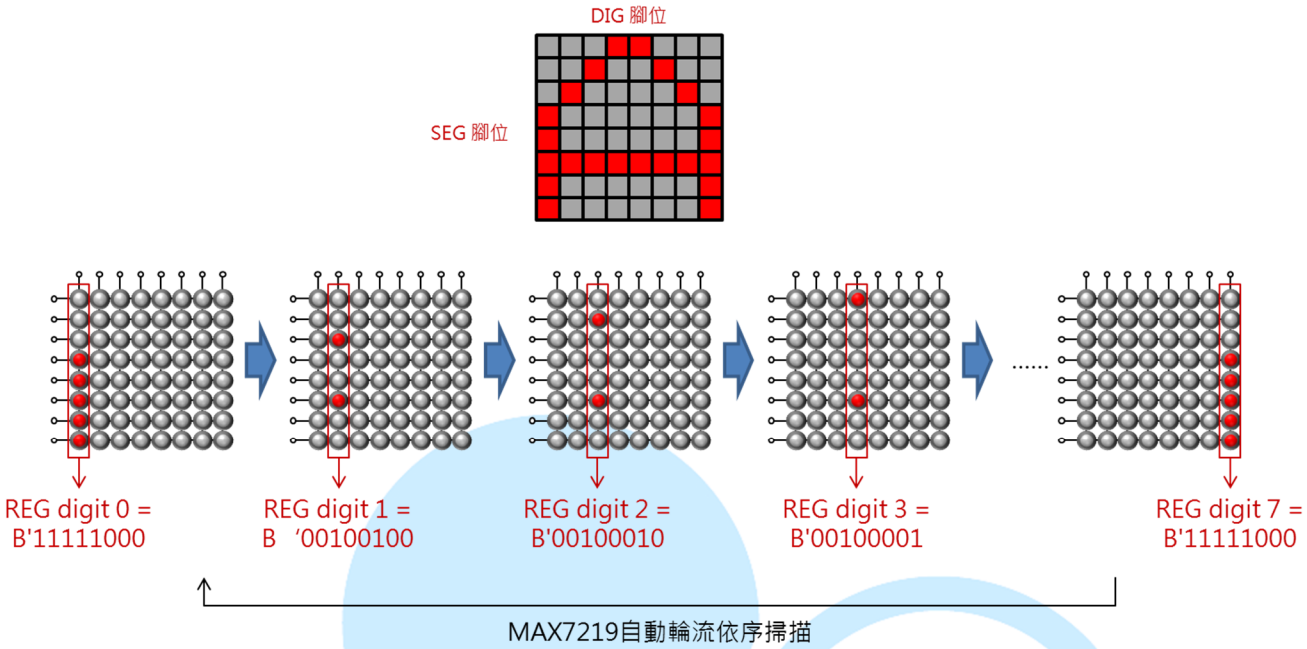
使用 MAX7219 时，主控制器需透过 DIN 针脚，对 IC 内部的几个缓存器（Register）作写入设定，MAX7219 的缓存器定义如下表：

缓存器名称	地址	定义
No-Op	0xX0	无动作
Digit 0	0xX1	Digit 0 脚位对应的行/列数据值
Digit 1	0xX2	Digit 1 脚位对应的行/列数据值
Digit 2	0xX3	Digit 2 脚位对应的行/列数据值
Digit 3	0xX4	Digit 3 脚位对应的行/列数据值
Digit 4	0xX5	Digit 4 脚位对应的行/列数据值
Digit 5	0xX6	Digit 5 脚位对应的行/列数据值
Digit 6	0xX7	Digit 6 脚位对应的行/列数据值
Digit 7	0xX8	Digit 7 脚位对应的行/列数据值
Decode Mode	0xX9	设定使否启用数据译码模式
Intensity	0xXA	亮度控制
Scan Limit	0xXB	设定 Digit 0~Digit 7 扫描范围
Shutdown	0xXC	设定是否关闭 LED 矩阵输出
Display Test	0xXF	测试模式

Digit 0~7 的数据部分，从 MSB~LSB 分别对应 SEG DP、SEG A、SEG B、SEG C、SEG D、SEG E、SEG F、SEG G 脚位接的 LED，要亮的 LED 位置位为 HIGH，反之熄灭为 LOW；若读者自行使用 MAX7219 IC 加上自制的周边 LED 或灯泡，须注意各种脚位与资料的对应关系喔。

读者若仔细观察 MAX7219 的针脚名称，会发现其实这颗 IC 除了用在控制单一个 8X8 LED 矩阵外，也可以用来控制 8 个 7 段显示器（一个 7 段显示器有 8 个 LED，一般称为：DP、A、B、C、D、E、F、G），所以读者学会使用控制 LED 矩阵后，也可以将相同原理用在控制整排的 7 段显示器用途。

一般使用上，只会需要设定缓存器 Digit 0~Digit 7 的内容，便可以控制 LED 矩阵的显示图样；以本文选用的 MAX7219+LED 矩阵模块来说，IC 朝下方摆设时，Digit 0 对应的是矩阵最左边第一行，Digit 1 对应左边第二行，依此类推。而 Digit 0 缓存器的内容有 8 位，每个位刚好对应某一行的亮度模式；这里选用的模块内定接线方式，由下而上是 SEG DP、SEG A、SEG B、…、SEG G。原理如图 5 流程：



前面提到「扫描」的工作就交给 MAX7219 自动去完成，因此只要将 Digit 0~7 的缓存器内容设定完成，LED 矩阵便可以显示出某个图案。当然，定期更改图案内容，就可以出现动画了。若读者自行制作其他接线架构的 LED 矩阵，需要注意一下缓存器数据与对应 LED 的位置关系，才会出现如同预期的图案喔。

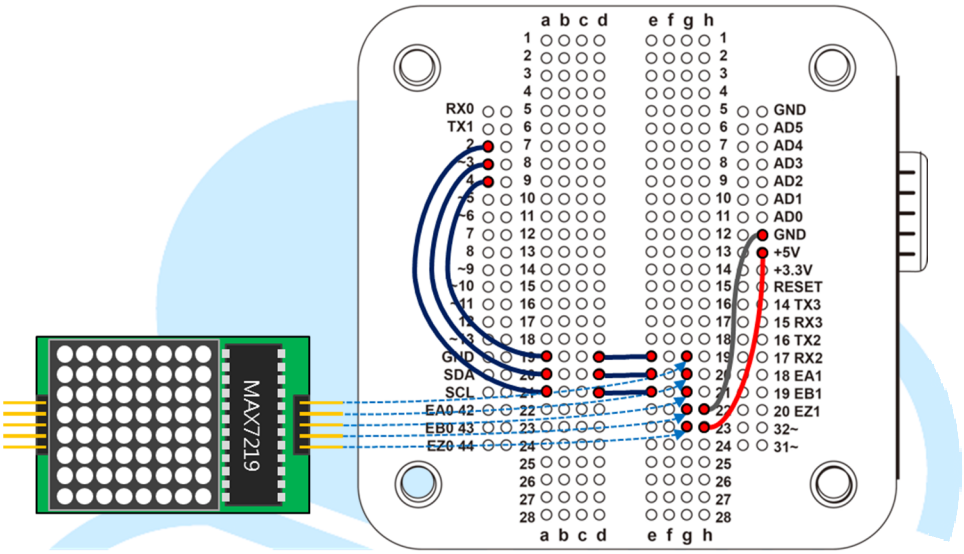
若需要更详细的规格信息，可参考 MAX7219 IC 说明文件：

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

下面便让我们利用程序，实际练习上面所讲解的 MAX7219 控制原理，并利用 LED 矩阵做实际显示图样的功能吧。

二、 第一个程序 – 单一 LED 矩阵练习 1

第一个范例程序先来练习如何使用 86Duino EduCake 与 MAX7219 模块的通讯，控制一组 8X8 LED 矩阵的图案，读者请先依下图接线：



接着请打开 86Duino Coding IDE，输入以下程序代码：

```
// 控制脚位定义
int DIN_pin = 2;
int LOAD_pin = 3;
int CLOCK_pin = 4;

// MAX7219 缓存器地址
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) {// 仿真 SPI 接口依序送出 byte 数据
      byte i = 8;
      byte mask;
      while(i > 0) {
        mask = 0x01 << (i - 1);// 制造位掩码，从最左边位开始
        digitalWrite( CLOCK_pin, LOW);// 频率同步线 = LOW
        if (data & mask){// 判断位掩码对应的位是 0 或 1
          digitalWrite(DIN_pin, HIGH);// 如果对应位为 1，DIN 送出 HIGH
        }
        else{
          digitalWrite(DIN_pin, LOW);// 如果对应位为 1，DIN 送出 LOW
        }
        digitalWrite(CLOCK_pin, HIGH);// 频率同步线 = HIGH
        --i;// 移到下一个位
      }
    }

    void MAX7219_1Unit(byte reg_addr, byte reg_data) {// 控制单一个 MAX7219 模块
      digitalWrite(LOAD_pin, LOW);// 传送前 LOAD 脚要先变成 LOW
      SPI_SendByte(reg_addr);// 先送出要设定的缓存器地址
      SPI_SendByte(reg_data);// 接着送出资料
      digitalWrite(LOAD_pin,HIGH);// 传送完 LOAD 脚要变成 HIGH
    }
```

```
byte matrixData_8X8[8] = {// 图样资料矩阵
  B01010101,// 第一行由下而上
  B10000001,
  B10101010,
  B11111111,
  B00000000,
  B11110000,
  B00001111,
  B11001100
};

void Draw (byte *LED_matrix)// 静态绘制整个画面
{
  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 的缓存器
  MAX7219_1Unit(max7219_REG_scanLimit, 0x07);// 设定为扫描所有行
```

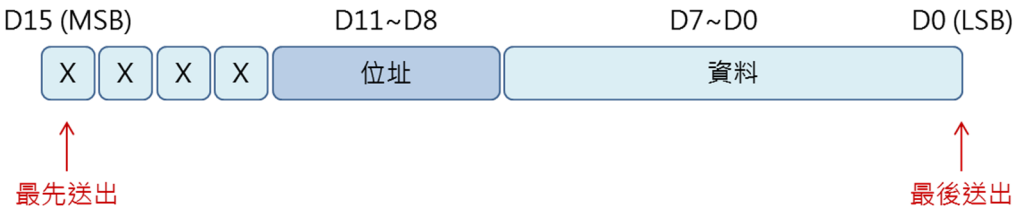
```
    MAX7219_1Unit(max7219_REG_decodeMode,  0x00);//  不使用译码模式
    MAX7219_1Unit(max7219_REG_shutdown, 0x01);//  设定为不在关闭模式
    MAX7219_1Unit(max7219_REG_displayTest, 0x00); //  设定为不在测试模式

    for(int i=1; i<=8; i++) {//  先把所有 LED 矩阵变暗
      MAX7219_1Unit(i,0);
    }

    MAX7219_1Unit(max7219_REG_intensity, 0x0f);//  设定亮度范围，0x00 ~ 0x0f

    delay(1000);
  }

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

刻录完成后，读者将会看到如下图的静态图样：

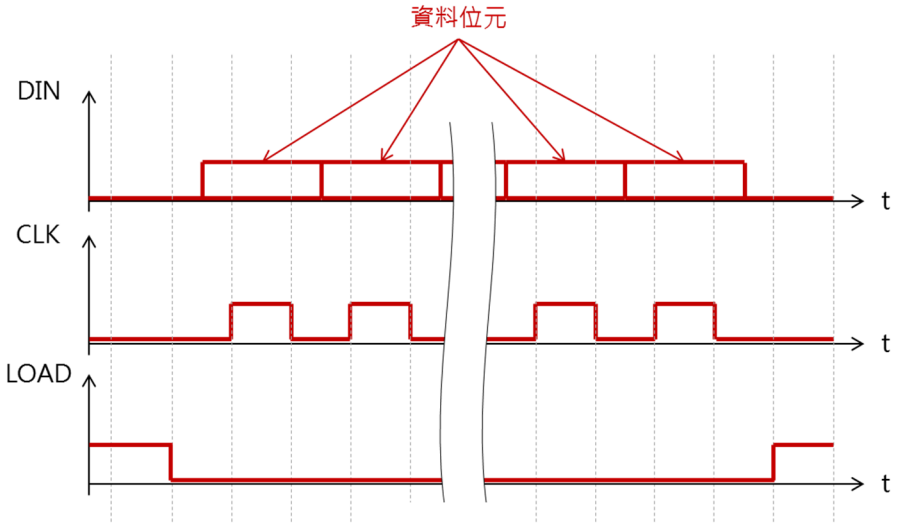
此范例程序功能为，设定一次 MAX7219 的 Digit 0~7 缓存器内容，让 LED 矩阵显示出单一图样；可使用这图样快速找出 LED 矩阵模块摆放方向与数据定义的关系。

程序一开始先设定各个控制脚位的编号，以及 MAX7219 所有缓存器的地址，接着 SPI_SendByte(byte data)这个功能用来仿真 SPI 传输一个 byte 的数据、MAX7219_1Unit(byte reg_addr, byte reg_data)则用在写入一个 MAX7219 单元的特定设定数据。

MAX7219_1Unit( )函式每一次对缓存器写入的数据单位为 16 位，其中 D0~D7 为数据位，D8~D11 为地址，D12~D15 没有定义，序列数据以 D15(MSB)→D0(LSB)的顺序送出，如下图示意：



DIN 数据线与 CLK 频率同步线的波形如下：



当 DIN 资料准备好时，CLK 需变为 HIGH，然后变为 LOW；而 LOAD 线则在传送前设为 LOAD，传送完变成 HIGH。matrixData_8X8[8]矩阵用来储存某个图样的数据，0 为 LED 熄灭，1 为 LED 点亮。

Draw( )函式内，由于需绘制整个 8X8 矩阵的画面，因此使用语法 MAX7219_1Unit( )，分别设定 8 行的数据即可完成画面的图案设定。

setup( )内除了设定使用脚位的 I/O 模式与初始值外，另外初始化 MAX7219 的许多缓存器，这些缓存器需要初始化才能够正常运作。而 loop( ) 内则是定时 500ms 呼叫 Draw( )函式。此处用来当作显示数据的 matrixData_8X8 矩阵内容一直保持不变，当然图样也就一直保持静态啰。

三、 第二个程序 – 单一 LED 矩阵练习 2

了解 86Duino EduCake 控制 MAX7219 与 LED 矩阵的基本原理后，接着来做点小变化，让图案变成动画效果吧。读者请打开 86Duino Coding IDE，接着在上一个范例程序中加入以下程序代码：

```
int shift = 0;
void ShiftDraw(byte *LED_matrix)// 位移绘制整个画面
{
  MAX7219_1Unit(1, LED_matrix[(shift) % 8]);// 绘制第 1 行的资料
  MAX7219_1Unit(2, LED_matrix[(shift+1) % 8]);// 绘制第 2 行的资料
  MAX7219_1Unit(3, LED_matrix[(shift+2) % 8]);// 绘制第 3 行的资料
  MAX7219_1Unit(4, LED_matrix[(shift+3) % 8]);// 绘制第 4 行的资料
  MAX7219_1Unit(5, LED_matrix[(shift+4) % 8]);// 绘制第 5 行的资料
  MAX7219_1Unit(6, LED_matrix[(shift+5) % 8]);// 绘制第 6 行的资料
  MAX7219_1Unit(7, LED_matrix[(shift+6) % 8]);// 绘制第 7 行的资料
  MAX7219_1Unit(8, LED_matrix[(shift+7) % 8]);// 绘制第 8 行的资料

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

然后将原本 loop( ) 中的「Draw(matrixData_8X8);」改成「ShiftDraw(matrixData_8X8);」便能够让上个范例中的图案依行不停往左位移循环啰。ShiftDraw( )这里使用的原理是，使用一个索引变量 shift，用 shift 作为设定 LED 各行亮度数据的索引值；shift 每次加 1，因此会依序对应到 matrixData_8X8 矩阵中不同的位置，就可以达成位移图案的效果。这里须注意 matrixData_8X8 矩阵只有索引 0~7 的范围是图案数据，所以实

际计算索引值时须使用(shift+N) % 8 语法，将数值限制在 0~7 的范围以免出错喔。同样 shift 变量在每次+1 之后，会检查是否超出范围，若超出范围也需将变量归 0，如此 shift 便会在 0~7 的数值内不停循环。读者也可以尝试将 matrixData_8X8 矩阵设定成不同的图案、或是使用不同的索引变化来试试其他特效喔。

四、 第三个程序 – 单一 LED 矩阵练习 3

这个范例电路不用更改，我们继续来加入其他的功能，读者请打开 86Duino Coding IDE，在原本的「matrixData_8X8」下方输入以下程序代

```
// 0
byte matrixData_num_0[8] = {// 图样资料矩阵
  B00000000,// 第一行由下而上
  B01111110,
  B10010001,
  B10001001,
  B10001001,
  B10000101,
  B01111110,
  B00000000
};
// 1
byte matrixData_num_1[8] = {// 图样 1 资料矩阵
  B00000000,// 第一行由下而上
  B00000000,
  B10000000,
  B10000010,
  B11111111,
  B10000000,
  B00000000,
```

码：

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

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

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

然后在 setup( )里面加入：

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

接着将 loop( )内改为：

```
void loop ( ) {
```

```
if(Serial.available())// 检查 Serial port 是否收到资料
{
  byte num = Serial.read();
  switch(num)// 针对不同数值显示不同数字
  {
    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;
```

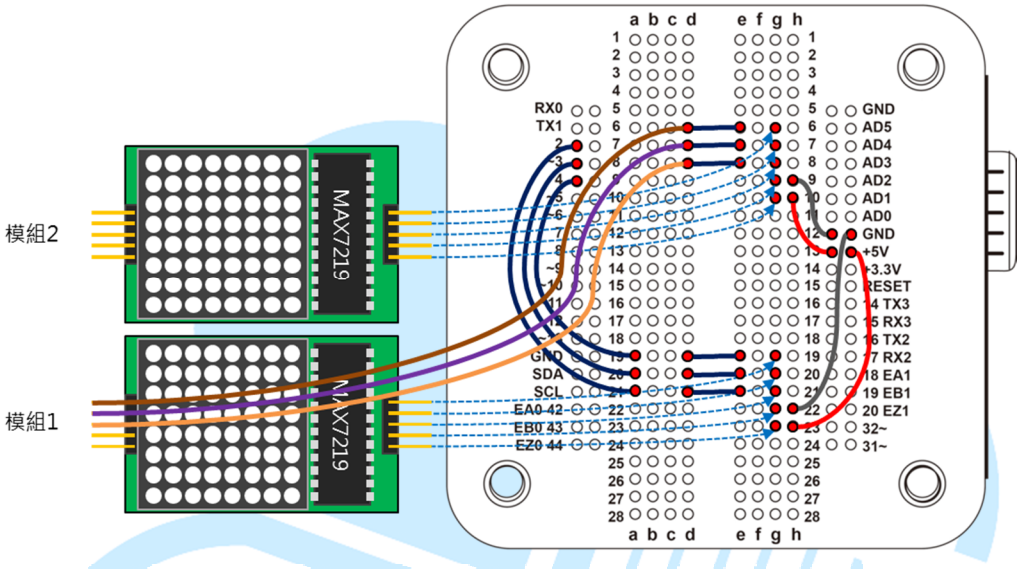
```
            case '7':
                Draw(matrixData_num_7);
                break;

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

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

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

编译并上传程序后，请打开 Serial Monitor，注意 baudrate 要设定成跟程序内一样；接着可以输入 0~9 任一数字，86Duino EduCake 接的 LED 矩阵便会显示对应的数字图案啰。

一开始在程序内加入的几个 byte 矩阵 matrixData_num_0~matrixData_num_9 便是数字 0~9 的文字图样；此程序功能为，利用 Serial Monitor 通讯功能，当用户输入数字时，86Duino EduCake 便会判断接收数字字符为何；接着使用 switch-case 语法，针对不同的字符数据，控制 MAX7219 在 LED 矩阵上显示出对应的文字图样。读者也可以自行增加通讯可用的字符，并修改自己喜欢的图样做显示喔。

五、 第四个程序 – 控制多个 LED 矩阵 1

练习过控制单个 MAX7219+LED 模块后，继续来练习多个单元的串联，读者请加入下列接线：




```
int MAX7219_units = 2;// 设定使用的 MAX7219 串联单元数量
```

接着请打开 86Duino Coding IDE，在 int CLOCK_pin = 4;之后加入：

在 void MAX7219_1Unit( )函式下方加入：

```
    void MAX7219_AllUnit( byte reg_addr, byte reg_data ) {// 控制所有串联的 MAX7219 模块，写入同样资料
      digitalWrite( LOAD_pin, LOW );// 传送前 LOAD 脚要先变成 LOW
      for (int c =1; c<= MAX7219_units; c++) {
        SPI_SendByte(reg_addr);// 先送出要设定的缓存器地址
        SPI_SendByte(reg_data);// 接着送出资料
      }
      digitalWrite(LOAD_pin,HIGH);// 传送完 LOAD 脚要变成 HIGH
    }
    void MAX7219_indexUnit( byte unit_index, byte reg_addr, byte reg_data ) {// 控制所有串联中的某一个 MAX7219 模块，其余模块图样不变
      int c = 0;
      digitalWrite(LOAD_pin, LOW);// 传送前 LOAD 脚要先变成 LOW
      // 从串联最尾端的模块开始控制
      for ( c = MAX7219_units; c > unit_index; c--) {
        SPI_SendByte(0);// NO-OP 缓存器
        SPI_SendByte(0);// 资料 = 0
      }

      SPI_SendByte(reg_addr);// 先送出要设定的缓存器地址
      SPI_SendByte(reg_data);// 接着送出资料

      for ( c = unit_index-1; c >= 1; c--) {
        SPI_SendByte(0);// NO-OP 缓存器
        SPI_SendByte(0);// 资料 = 0
      }
      digitalWrite(LOAD_pin,HIGH);// 传送完 LOAD 脚要变成 HIGH
    }
```

在 Draw( )之后加入：

```
void Draw_Unit( byte index, byte *LED_matrix )// 静态绘制特定单元的图样
{
  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 的缓存器
  MAX7219_AllUnit( max7219_REG_scanLimit, 0x07 );// 设定为扫描所有行
  MAX7219_AllUnit( max7219_REG_decodeMode, 0x00 );// 不使用译码模式
  MAX7219_AllUnit( max7219_REG_shutdown, 0x01 );// 设定为不在关闭模式
  MAX7219_AllUnit( max7219_REG_displayTest, 0x00 ); // 设定为不在测试模式
```

修改 setup( )内容为：

```
        for( int i=1; i<=8; i++ ) {// 先把所有 LED 矩阵变暗
          MAX7219_AllUnit(i,0);
        }

        MAX7219_AllUnit( max7219_REG_intensity, 0x0f );// 设定亮度范围，0x00 ~ 0x0f

        delay(1000);
      }
```

以及修改 loop( )内容为：

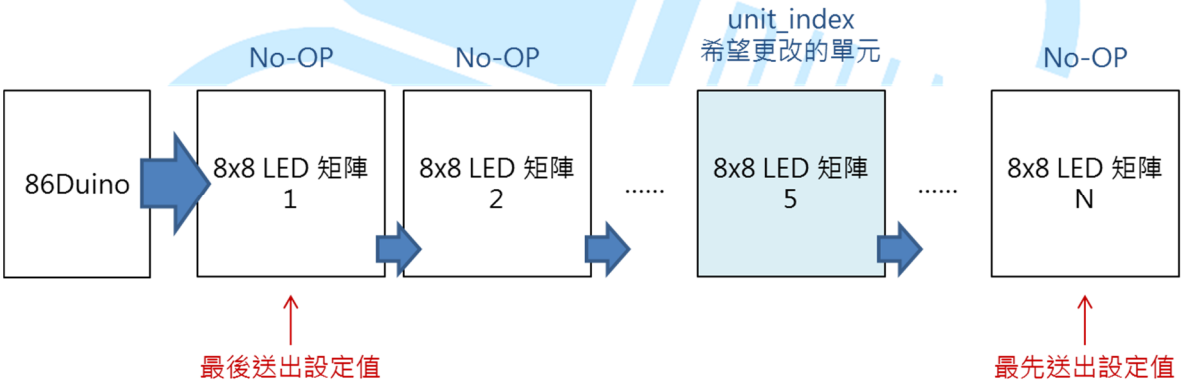
```
void loop () {
  Draw_Unit(1, matrixData_num_0);
  Draw_Unit(2, matrixData_num_1);
  delay(500);
}
```

此范例程序从前一个进行修改，加入支持多个 MAX7219 模块的函式「 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)」。

如前面提过的，由于 MAX7219 可以多个模块串联，且前一个模块的 DIN 脚位数据会被 DOUT 传递到下一个串联模块去，所以有多个 MAX7219 模块时，需要针对每个模块作缓存器设定才行。所以前面新宣告一个 int MAX7219_units 变量，用在设定使用的 MAX7219 串联模块数量，用户须视实际使用的模块数量状况，设定这个数值；这里因为使用两个模块所以设定为 2。

MAX7219_AllUnit( ) 这个函式基本上流程跟原本的「MAX7219_1Unit( )」差不多，但因为需要对每个 MAX7219 单元写入相同数据，所以 LOAD_pin 设定为 LOW 之后，中间使用 for 循环，对每个 MAX7219 的同个缓存器写入相同数值，再将 LOAD_pin 设为 HIGH。

MAX7219_indexUnit( )跟上一个功能刚好相反，用在分别设定「不同数据」给特定的 MAX7219 单元；函式的自变量多了一个 byte unit_index，用在指定希望更改的模块；由于送出设定数据会被传递到下一个模块，所以送出数据顺序是从「串联的最尾端模块」开始，如果有 N 个模块则须送出 N 次，如下图所示：



只有 unit_index 指定的模块需要更改缓存器数据，其余都不要更动，这要怎么办呢？前面 MAX7219 缓存器表格里的「No-Op」缓存器这时就派上用场了。如上图说明，当需要在 N 个串联模块内更改第 5 个模块时，第 1~4 以及第 6~N 的模块都须在 No-Op 写入数值，但写入 No-Op 不会影响此单元的动作。MAX7219_indexUnit( )函式便是在做这个过程；注意跟 MAX7219_AllUnit( )相同，LOAD_pin 都是写入前为 LOW，全部资料写完后变为 HIGH，中间是没有变化的。

Draw_Unit(byte index, byte *LED_matrix)这个函式则是为了在特定模块画出图案，但不更改其他模块的图样；所以里面使用 MAX7219_indexUnit( )函数针对 index 对应的 MAX7219 做设定便可达到功能。

setup( )里面初始化的部分，由于有 N 个模块皆须做相同的初始化，所以直接把原本的 MAX7219_1Unit( )改用 MAX7219_AllUnit( )即可；而 loop( )里面使用 Draw_Unit(1, matrixData_num_0)、Draw_Unit(2, matrixData_num_1)，在串联的第 1 个模块绘出图样数字 0，第 2 个模块绘出图样数字 1。读者也可以自行更改图样内容、绘出的单元位置等等试试其他效果喔。

六、 第五个程序 – 控制多个 LED 矩阵 2

最后一个练习程序，不更改接线，直接由第 4 个程序做点小变化，加入动态跑马灯效果；读者请在原程序内加入以下程序代码：

```
// 86Duino EduCake
const unsigned int string_len = 70;
byte matrixData_86Duino_EduCake[string_len] = {// 图样资料矩阵
  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)// 位移绘制整个画面
    {
      // Unit 1
      MAX7219_indexUnit(1, 1, LED_matrix[(shift) % string_len]);// 绘制第 1 行的资料
      MAX7219_indexUnit(1, 2, LED_matrix[(shift+1) % string_len]);// 绘制第 2 行的资料
      MAX7219_indexUnit(1, 3, LED_matrix[(shift+2) % string_len]);// 绘制第 3 行的资料
      MAX7219_indexUnit(1, 4, LED_matrix[(shift+3) % string_len]);// 绘制第 4 行的资料
      MAX7219_indexUnit(1, 5, LED_matrix[(shift+4) % string_len]);// 绘制第 5 行的资料
      MAX7219_indexUnit(1, 6, LED_matrix[(shift+5) % string_len]);// 绘制第 6 行的资料
      MAX7219_indexUnit(1, 7, LED_matrix[(shift+6) % string_len]);// 绘制第 7 行的资料
      MAX7219_indexUnit(1, 8, LED_matrix[(shift+7) % string_len]);// 绘制第 8 行的资料
```

```
    // Unit 2
      MAX7219_indexUnit(2,    1,    LED_matrix[(shift+8)    %
string_len]);// 绘制第 1 行的资料
      MAX7219_indexUnit(2,    2,    LED_matrix[(shift+9)    %
string_len]);// 绘制第 2 行的资料
      MAX7219_indexUnit(2,    3,    LED_matrix[(shift+10)   %
string_len]);// 绘制第 3 行的资料
      MAX7219_indexUnit(2,    4,    LED_matrix[(shift+11)   %
string_len]);// 绘制第 4 行的资料
      MAX7219_indexUnit(2,    5,    LED_matrix[(shift+12)   %
string_len]);// 绘制第 5 行的资料
      MAX7219_indexUnit(2,    6,    LED_matrix[(shift+13)   %
string_len]);// 绘制第 6 行的资料
      MAX7219_indexUnit(2,    7,    LED_matrix[(shift+14)   %
string_len]);// 绘制第 7 行的资料
      MAX7219_indexUnit(2,    8,    LED_matrix[(shift+15)   %
string_len]);// 绘制第 8 行的资料

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

然后将 loop ( )改成：

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

上传程序完成后，读者便可以看到「86Duino EduCake」的字样不停循环显示，就像一般广告跑马灯效果一样。ShiftDraw_2Unit( )这个

函式基本上跟 ShiftDraw( )差不多，只是用来当作图样的 byte 矩阵变长了而已；范例这个图样矩阵共有 70 列的长度，所以函式内的索引变量只能在 0~69 内循环，这里使用 const unsigned int string_len 作为图样矩阵列数长度，方便程序撰写与修改。当读者拥有更多个 MAX7219 模块时，也可以用相同的概念去做延伸啰。