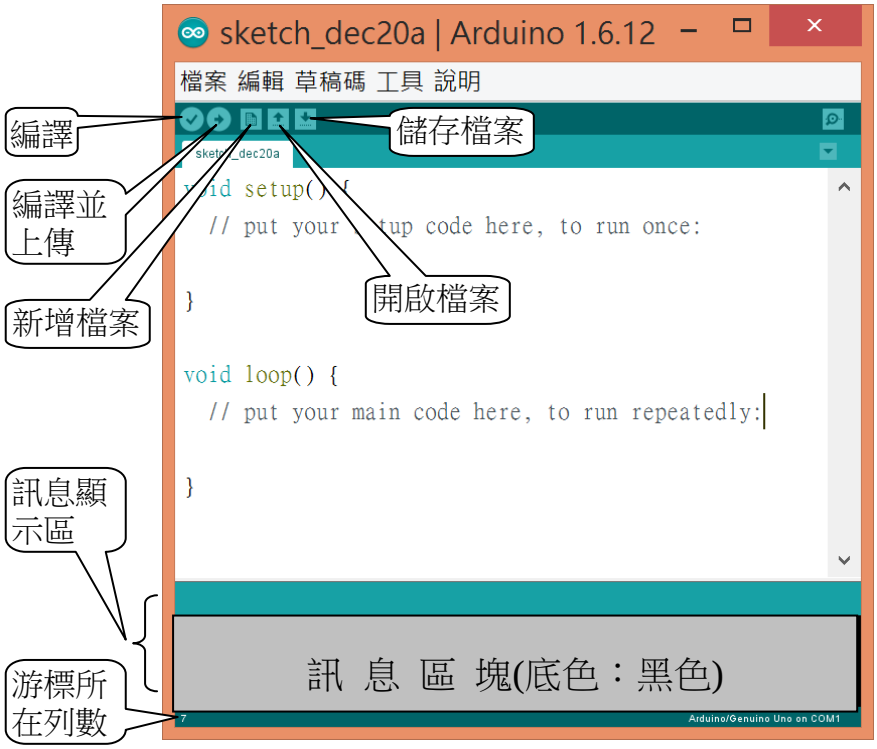
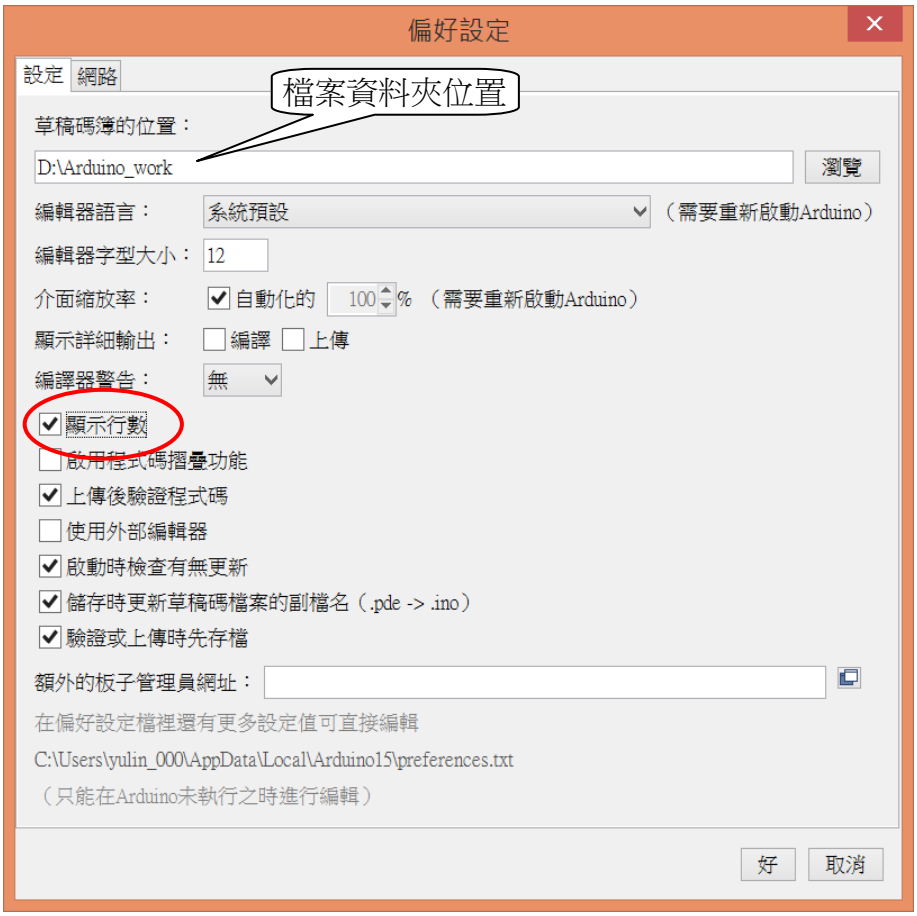
壹、Arduino C語言開發軟體使用

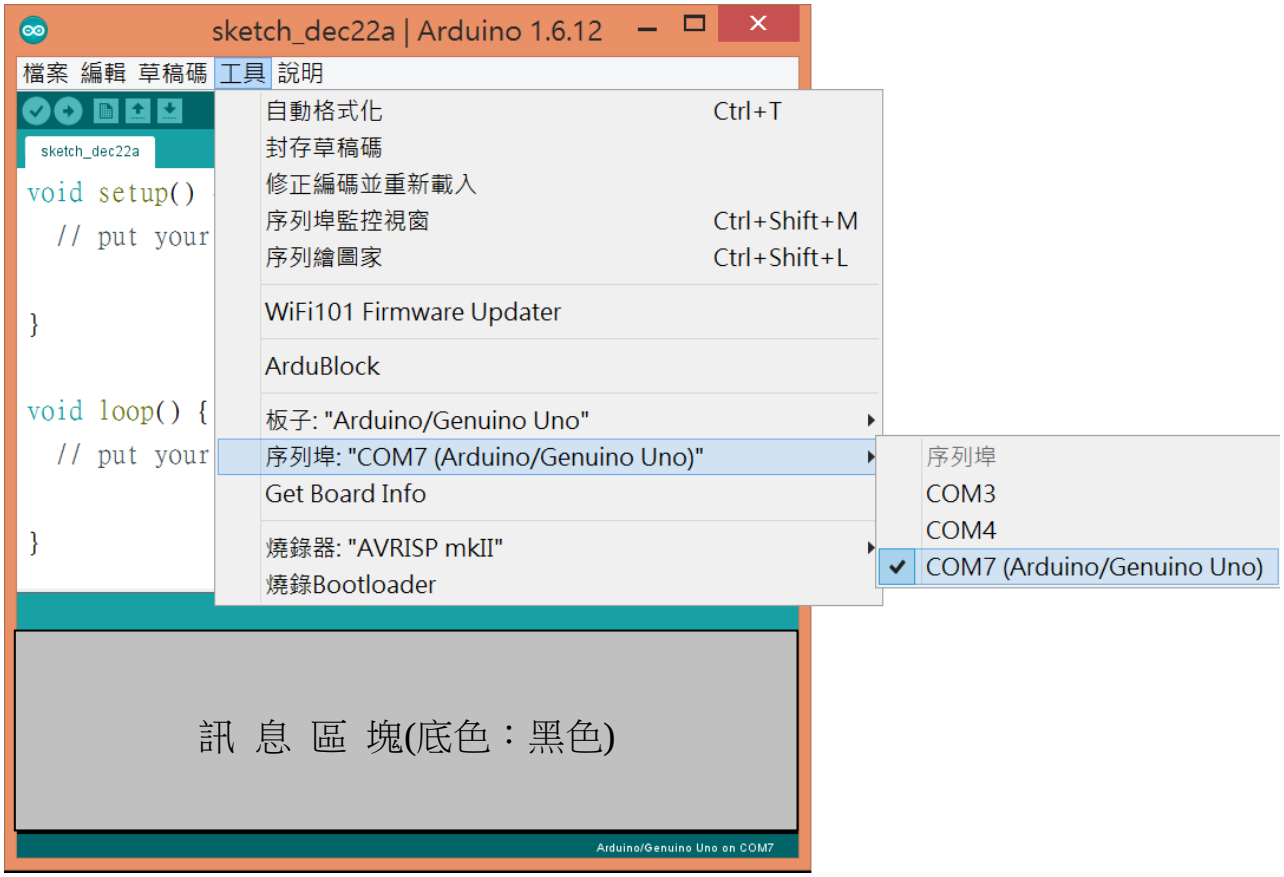
一、程式編輯環境：



二、偏好設定：執行【檔案】→【偏好設定】



三、USB連接埠設定：利用【工具】功能表選項設定正確的Arduino裝置連接埠，編譯上傳檔案才能成功。



四、程式結構：Arduino 程式檔案(*.ino)其檔案內容包含：

1. 標頭檔案(*.H)：預先定義的變數或函數庫
2. I/O 腳位定義或參數初始化設定程式段【setup()】僅執行一次
3. 主程式段【loop()】
4. 自定函式或副程式段
5. 註解有(a)區段註解"/*...*/"，(b)單列註解"//..."

貳、Arduino C程式語法

資料類型:

Arduino 的資料型態與 C 語言一樣, 但資料長度可能因板子而異, 下表適用於 UNO, Nano, Pro Mini 等以 ATmega328 為處理器的板子, 只有 Due 板子有所不同 :

資料型態	說明	記憶體長度	數值範圍
boolean	布林	8 bits	true (1, HIGH), false(0, LOW)
byte	位元組	8 bits	0~255
char	字元	8 bits	-128~127
short	短整數	16 bits	-32768~32767
int	整數	16 bits	-32768~32767
word	字	16 bits	0~65535
long	長整數	32 bits	-2147483648~2147483647
float	浮點數	32 bits	+-3.4028235e+38
double	倍精度浮點數	32 bits	+-3.4028235e+38

Arduino 還有兩個非數值的資料類型 :

資料型態	說明
void	用來表示函數無傳回值時之資料類型
String	用來表示字串 (Arduino 0019 Alpha 版以後)

要注意的是 : 一般 Arduino 板子的double 跟 float 是完全一樣的, 都是 32 位元, 而 Due 板則跟一般 C 語言一樣是 64 位元. 其次,整數的 short 與 int 也是一樣 16 位元的(但在 Due 板子, int 是 32 位元). 文件中整數類的 char, int, 與 long 這三種型態有分 signed (有號) 與 unsigned (無號的), 沒有提到 unsigned short, 但我測試是有的.

資料型態	說明	記憶體長度	數值範圍
unsigned char	字元	8 bits	0~255
unsigned short	短整數	16 bits	0~65535
unsigned int	整數	16 bits	0~65535
unsigned long	長整數	32 bits	0~4294967295

有號與無號差別在於最高位元是否拿來當正負符號 (2 的補數), signed 使用最高位元來表示正負數, 1 為負數, 0 為正數; 而 unsigned 則全部都是正數, 因此其可表示的正數範圍是 signed 的兩倍, 例如儲存字元用的 char 也可以用來儲存較小的整數 (8 位元), 若宣告為 unsigned char, 其範圍便從 -128~127 變成 0~255.

Char, short, int, 與 long預設是 signed, 亦即只有全部用做正數的變數才需要宣告unsigned.正負值都有的變數宣告為 signed 是多此一舉. 整數類中最常用的是 int, 因其範圍可以滿足大部分應用所需. Char 是其實是 8 位元整數, 用來表示 ASCII 字元, 例如 'A' 實際上是以其 ASCII 碼 65 儲存的, 好處是可以透過運算處理字元, 例如 'A' + 2 就得到 'C'. 所以字元可以用下面兩種方式表示 :

```
char c='A';   //字元必須用單引號括起來, 不能用雙引號
char c=65;
```

Byte 與 word 類型只有正數, byte 與 char 一樣是 8 位元, 等於 unsigned char; 而 word 與 int 一樣是 16 位元, 等於 unsigned int.

布林值可用 true/false (必須小寫), 0/1, 或者 HIGH/LOW 表示, 依據使用場合何者較有意義而定. 其中 HIGH/LOW 是 Arduino 定義的常數, 適合用在 digitalWrite() 函數中表示 LED 輸出位準, 而 true/false 適合用在程式邏輯的分岐判斷.

指定資料類型時須注意不能超過範圍, 超過時將歸零或變成負數等非預期結果, 成為 roll-over (反折), 例如 byte 最大 255, 若存入 256, 仍可通過編譯, 但真正存入的值會變成 0; 而整數最大為 32767, 若存入 32768, 結果變成 -32768

內建常數:

Arduino 定義了八個內建常數 (注意, true 與 false 須小寫) :

常數	說明
HIGH	輸出高電位
LOW	輸出低電位
INPUT	輸入腳
OUTPUT	輸出腳
INPUT_PULLUP	啟動上拉電阻之輸入腳
LED_BUILTIN	內建 LED (Pin 13)
true	真 (=1)
false	假 (=0)
PI	圓周率 3.14159
DEG_TO_RAD	角度轉弧度常數=PI/180=0.0174533
RAD_TO_DEG	弧度轉角度常數=180/PI=57.29578

例如 LED 閃爍程式 :

```
void setup() {
      pinMode(LED_BUILTIN, OUTPUT);
 }

void loop() {
      digitalWrite(LED_BUILTIN, HIGH);
      delay(1000);
      digitalWrite(LED_BUILTIN, LOW);
      delay(1000);
 }
```

也可以用 const 自行定義常數 (但不可重複定義內建常數), 例如可定義常數 LED 以取代上內建的 LED_BUILTIN, 效果一樣 :

```
const int LED=13;
```

常數賦值後無法更改, 否則編譯時即會失敗. 此外常數名稱通常使用大寫, 但這只是習慣, 並非語法規則. 另外,Arduino 的 I/O 針腳預設是 INPUT, 因此只需要宣告哪些腳要當 OUTPUT 即可, pinMode(2, INPUT) 其實是多此一舉的.

語法參考：

結構

setup()
loop()
控制結構
if
if…else
for
switch case
while
do… while
break
continue
return
goto
更多程式結構語法
; (分號)
{} (大括號)
// (單行註解)
/* */ (多行註解)
#define
#include

算術運算子
= (指派運算子)
+ (加法運算子)
- (減法運算子)
* (乘法運算子)
/ (除法運算子)
% (餘數運算子)
比較運算子
== (等於)
!= (不等於)
< (小於)
> (大於)
<= (小於等於)
>= (大於等於)
布林運算子
&& (and)
|| (or)
! (not)

指標運算子
* 取值運算子
& 取址運算子

位元運算子
& (位元 and)
| (位元 or)
^ (位元 xor)
~ (位元 not)
<< (位元左移運算)
>> (位元右移運算)

複合運算子
++ (遞增運算子)
- - (遞減運算子)
+= (複合加法運算)
-= (複合減法運算)
*= (複合乘法運算)
/= (複合除法運算)
&= (複合位元 and)
|= (複合位元 or)

位元及位元組操作
bitRead()
bitWrite()
bitSet()
bitClear()
bit()

函式

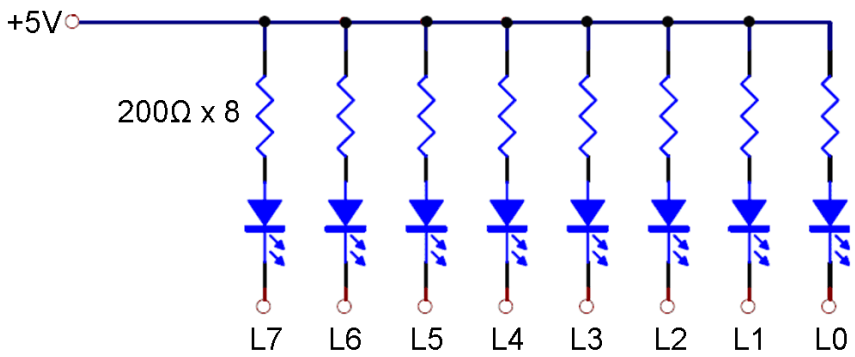
數位 I/O 功能
pinMode()
digitalWrite()
digitalRead()
類比 I/O 功能
analogReference()
analogRead()
analogWrite() – PWM
analogReadResolution()
analogWriteResolution()
進階 I/O 功能
tone()
noTone()
shiftOut()
shiftIn()
pulseIn()

時間
millis()
micros()
delay()
delayMicroseconds()
數學
min()
max()
abs()
constrain()
map()
pow()
sqrt()
三角函式
sin()
cos()
tan()

隨機數
randomSeed()
random()

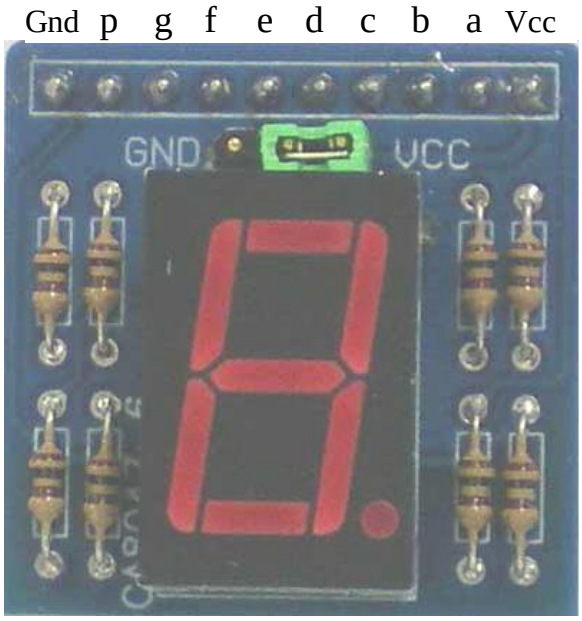
叁、實驗電路及套件使用說明

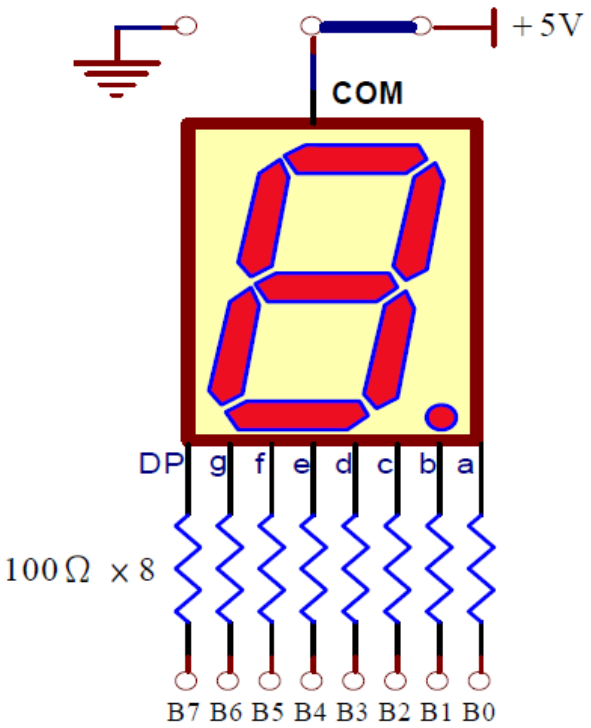
一、LED電路



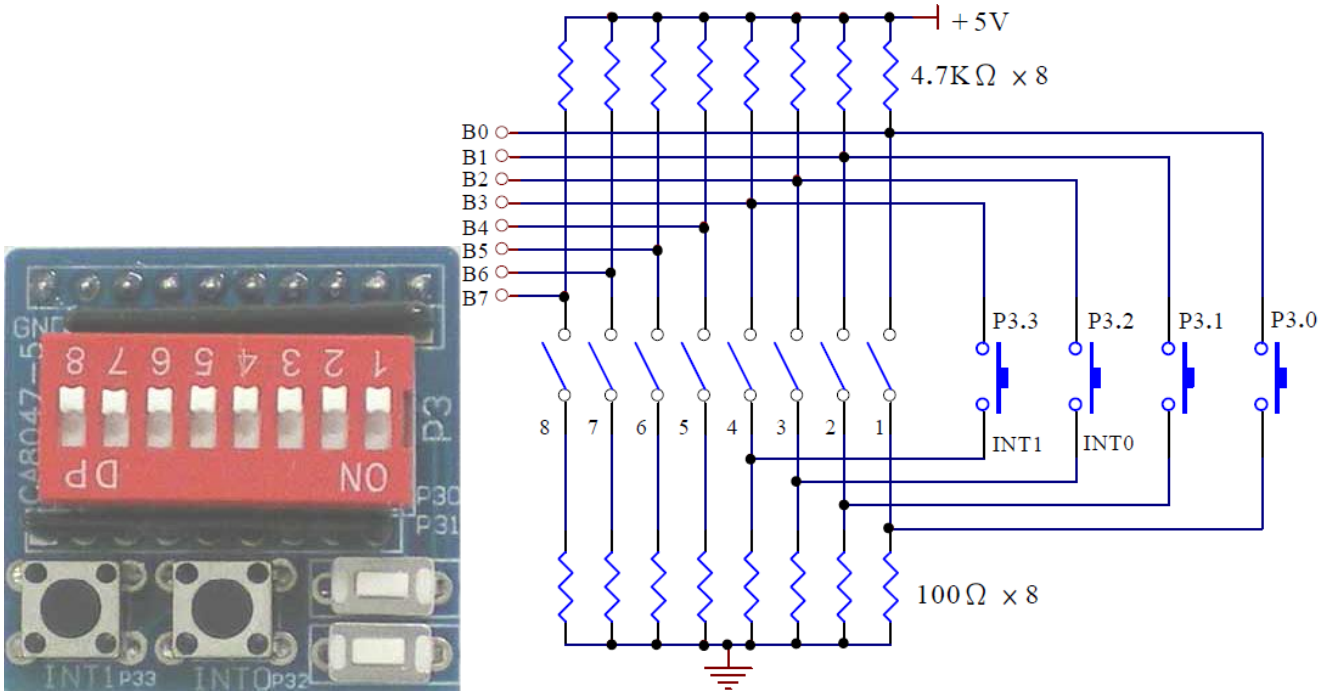
二、七節顯示器實驗板

使用共陰極七段顯示器時，將短路帽移至左側

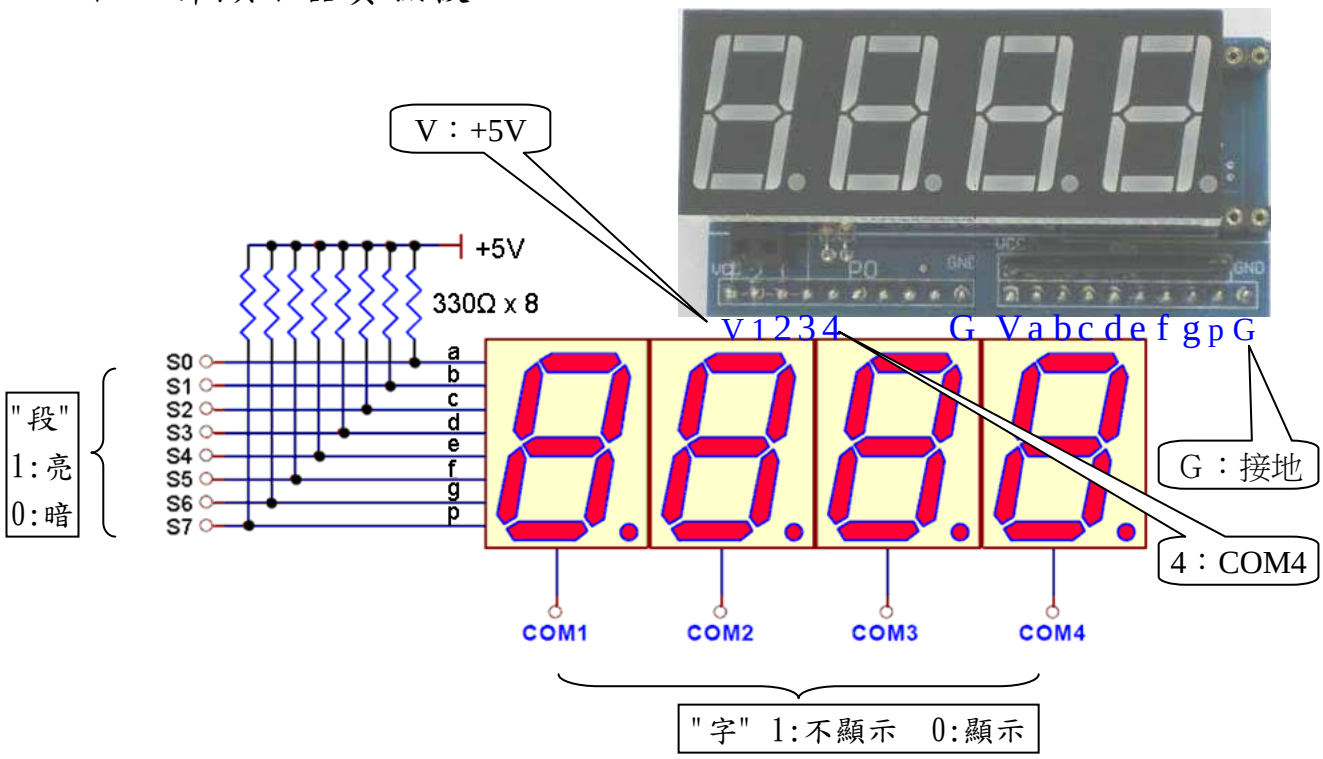




三、指撥&按鈕實驗板



四、四位七節顯示器實驗板



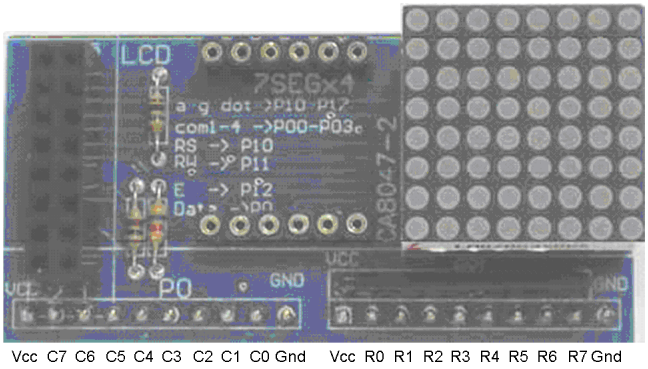
五、8 × 8共陽極矩陣LED實驗板

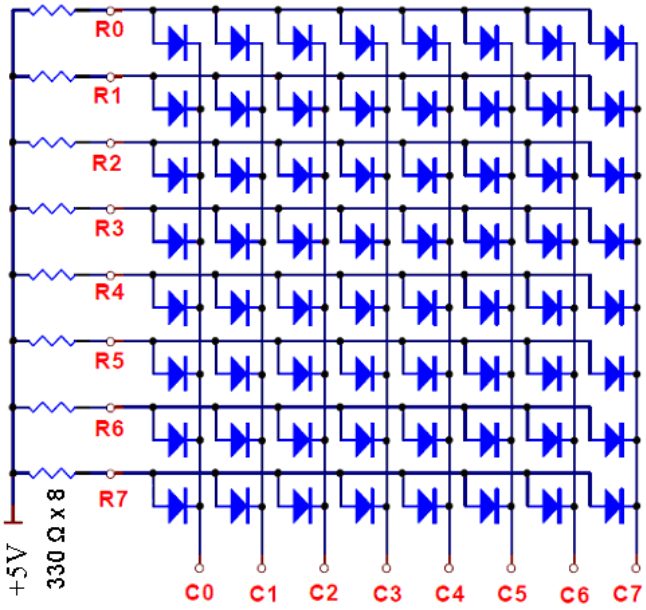
因電路設計結構，為使亮度均勻，安排如下：

Rn(橫列)：字型資料 1:亮，0:不亮

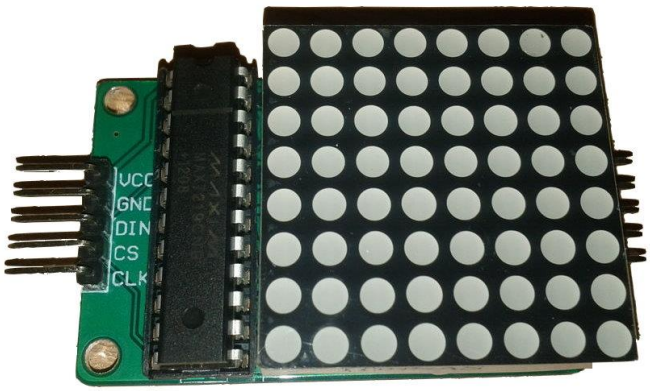
Cn(直行)：顯示掃描 1:不顯示，0:顯示

(n：0..7)



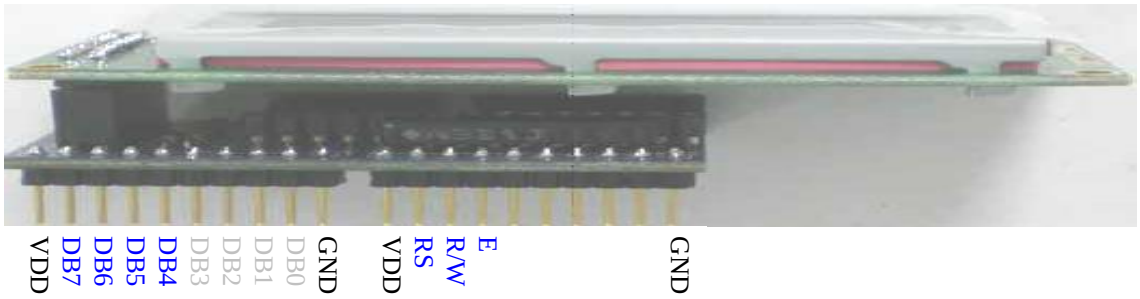


MAX7219 8×8 矩陣 LED 模組

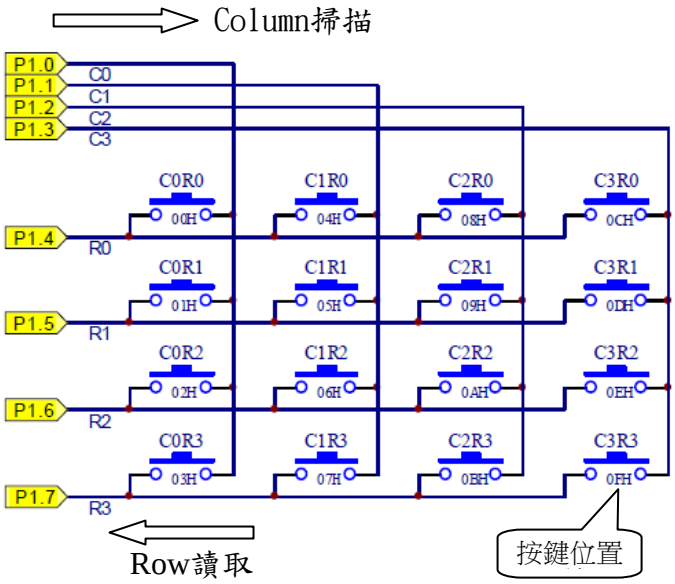


六、16 × 2 LCM實驗板

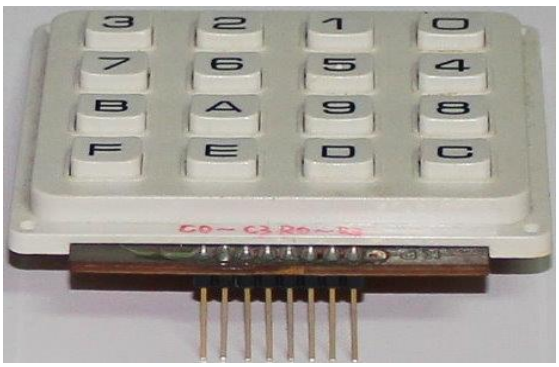
LCM接腳與Arduino實驗板之連線														
LCM接腳	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Vss	VDD	Vc	RS	R /W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
Arduino接腳				pin 3	GND	pin 2					pin 4	pin 5	pin 6	pin 7



七、4×4矩陣鍵盤



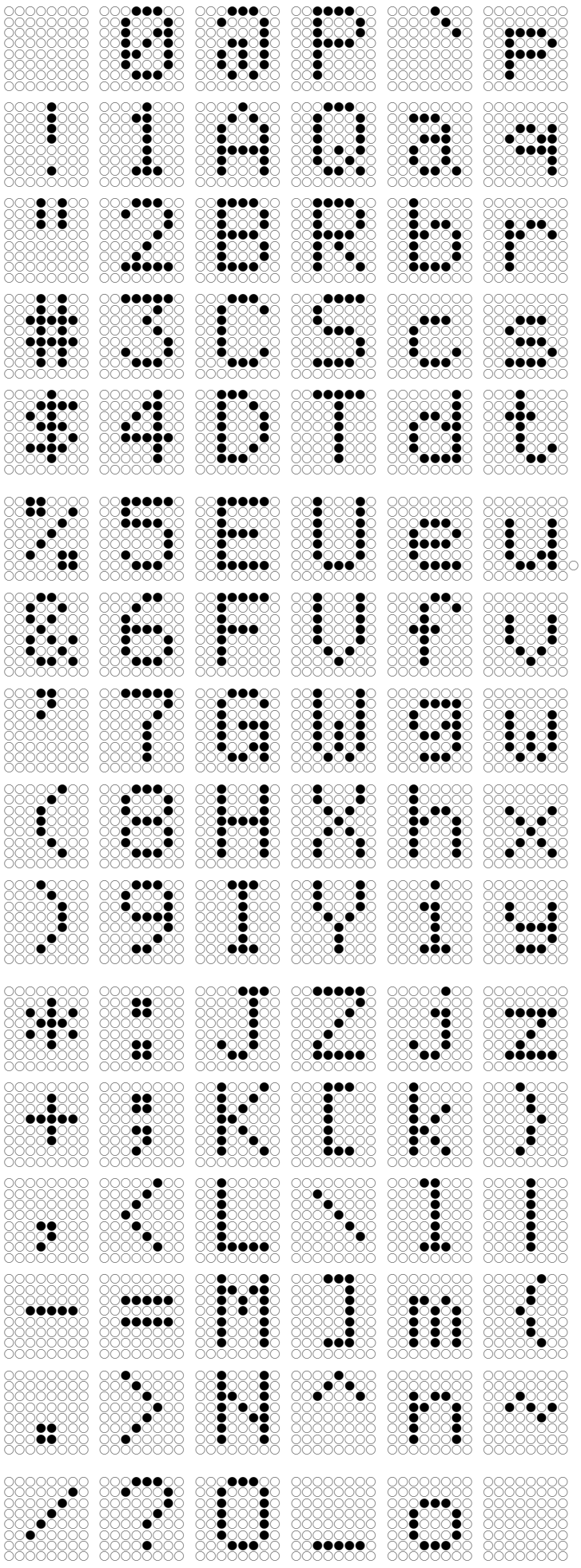
	Column輸出 C3 C2 C1 C0	按鍵	讀取Row R3 R2 R1 R0	按鍵位置碼 Cn ×4＋Rn
掃	1 1 1 0	沒有	1 1 1 1	×
描	1 1 1 0	第0列	1 1 1 0	00H
第	1 1 1 0	第1列	1 1 0 1	01H
0	1 1 1 0	第2列	1 0 1 1	02H
行	1 1 1 0	第3列	0 1 1 1	03H
掃	1 1 0 1	沒有	1 1 1 1	×
描	1 1 0 1	第0列	1 1 1 0	04H
第	1 1 0 1	第1列	1 1 0 1	05H
1	1 1 0 1	第2列	1 0 1 1	06H
行	1 1 0 1	第3列	0 1 1 1	07H
掃	1 0 1 1	沒有	1 1 1 1	×
描	1 0 1 1	第0列	1 1 1 0	08H
第	1 0 1 1	第1列	1 1 0 1	09H
2	1 0 1 1	第2列	1 0 1 1	0AH
行	1 0 1 1	第3列	0 1 1 1	0BH
掃	0 1 1 1	沒有	1 1 1 1	×
描	0 1 1 1	第0列	1 1 1 0	0CH
第	0 1 1 1	第1列	1 1 0 1	0DH
3	0 1 1 1	第2列	1 0 1 1	0EH
行	0 1 1 1	第3列	0 1 1 1	0FH



接腳排列：C0 C1 C2 C3 R0 R1 R2 R3

鍵盤 Keypad.h 常用函式功能

- void begin（makeKeymap（userKeymap））
  將內部鍵映射初始化為等於 userKeymap
- char waitForKey（）
  此功能將永遠等待，直到有人按下一個鍵。警告：它阻止所有其他代碼，直到按下一個鍵。這意味著沒有閃爍的 LED，沒有 LCD 屏幕更新，沒有任何異常的中斷程序。
- char getKey（）
  返回按下的鍵（如果有）。此功能是非阻塞的。
- KeyState getState（）
  返回任何鍵的當前狀態。這有四個狀態是空閒，按壓，釋放和保持。
- boolean keyStateChanged（）
  新版本 2.0：讓我們知道鍵何時從一個狀態改變到另一個狀態。例如，不是僅僅測試有效的鍵，而是可以測試按下鍵的時間。
- setHoldTime（unsigned int time）
  設置用戶必須保持按鈕的毫秒數，直到 HOLD 狀態被觸發。
- setDebounceTime（unsigned int time）
  設置鍵盤等待直到接受新的 keypress / keyEvent 為止的毫秒數。這是“時間延遲”去抖動方法。



8×8
點
矩
陣
字
形
圖

【 8x8 C程式點矩陣字形定義表 】

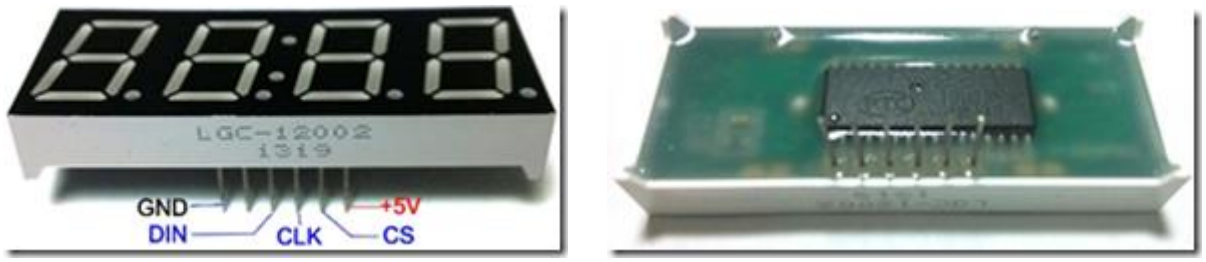
```
const byte patten[][8]={
  {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},   //' ' 20H
  {0x00,0x00,0x00,0x00,0x4f,0x00,0x00,0x00},   //'!' 21H
  {0x00,0x00,0x00,0x07,0x00,0x07,0x00,0x00},   //'"' 22H
  {0x00,0x00,0x14,0x7f,0x14,0x7f,0x14,0x00},   //'#' 23H
  {0x00,0x00,0x24,0x2a,0x7f,0x2a,0x12,0x00},   //'$' 24H
  {0x00,0x00,0x23,0x13,0x08,0x64,0x62,0x00},   //'%' 25H
  {0x00,0x00,0x36,0x49,0x55,0x22,0x50,0x00},   //'&' 26H
  {0x00,0x00,0x00,0x05,0x03,0x00,0x00,0x00},   //''' 27H
  {0x00,0x00,0x00,0x1c,0x22,0x41,0x00,0x00},   //'(' 28H
  {0x00,0x00,0x00,0x41,0x22,0x1c,0x00,0x00},   //')' 29H
  {0x00,0x00,0x14,0x08,0x3e,0x08,0x14,0x00},   //'*' 2AH
  {0x00,0x00,0x08,0x08,0x3e,0x08,0x08,0x00},   //'+' 2BH
  {0x00,0x00,0x00,0x50,0x30,0x00,0x00,0x00},   //',' 2CH
  {0x00,0x00,0x08,0x08,0x08,0x08,0x08,0x00},   //'-' 2DH
  {0x00,0x00,0x00,0x60,0x60,0x00,0x00,0x00},   //'.' 2EH
  {0x00,0x00,0x20,0x10,0x08,0x04,0x02,0x00},   //'/' 2FH
  {0x00,0x00,0x3e,0x51,0x49,0x45,0x3e,0x00},   //'0' 30H
  {0x00,0x00,0x00,0x42,0x7f,0x40,0x00,0x00},   //'1' 31H
  {0x00,0x00,0x42,0x61,0x51,0x49,0x46,0x00},   //'2' 32H
  {0x00,0x00,0x21,0x41,0x45,0x4b,0x31,0x00},   //'3' 33H
  {0x00,0x00,0x18,0x14,0x12,0x7f,0x10,0x00},   //'4' 34H
  {0x00,0x00,0x27,0x45,0x45,0x45,0x39,0x00},   //'5' 35H
  {0x00,0x00,0x3c,0x4a,0x49,0x49,0x30,0x00},   //'6' 36H
  {0x00,0x00,0x01,0x01,0x79,0x05,0x03,0x00},   //'7' 37H
  {0x00,0x00,0x36,0x49,0x49,0x49,0x36,0x00},   //'8' 38H
  {0x00,0x00,0x06,0x49,0x49,0x29,0x1e,0x00},   //'9' 39H
  {0x00,0x00,0x00,0x66,0x66,0x00,0x00,0x00},   //':' 3AH
  {0x00,0x00,0x00,0x56,0x36,0x00,0x00,0x00},   //';' 3BH
  {0x00,0x00,0x08,0x14,0x22,0x41,0x00,0x00},   //'<' 3CH
  {0x00,0x00,0x14,0x14,0x14,0x14,0x14,0x00},   //'=' 3DH
  {0x00,0x00,0x41,0x22,0x14,0x08,0x00,0x00},   //'>' 3EH
  {0x00,0x00,0x02,0x01,0x51,0x09,0x06,0x00},   //'?' 3FH
  {0x00,0x00,0x32,0x49,0x39,0x41,0x3e,0x00},   //'@' 40H
  {0x00,0x00,0x7c,0x12,0x11,0x12,0x7c,0x00},   //'A' 41H
  {0x00,0x00,0x7f,0x49,0x49,0x49,0x36,0x00},   //'B' 42H
  {0x00,0x00,0x3e,0x41,0x41,0x41,0x22,0x00},   //'C' 43H
  {0x00,0x00,0x7f,0x41,0x41,0x22,0x1c,0x00},   //'D' 44H
  {0x00,0x00,0x7f,0x49,0x49,0x49,0x41,0x00},   //'E' 45H
  {0x00,0x00,0x7f,0x09,0x09,0x09,0x01,0x00},   //'F' 46H
  {0x00,0x00,0x3e,0x41,0x49,0x29,0x7a,0x00},   //'G' 47H
  {0x00,0x00,0x7f,0x08,0x08,0x08,0x7f,0x00},   //'H' 48H
  {0x00,0x00,0x00,0x41,0x7f,0x41,0x00,0x00},   //'I' 49H
  {0x00,0x00,0x20,0x40,0x41,0x3f,0x01,0x00},   //'J' 4AH
  {0x00,0x00,0x7f,0x08,0x14,0x22,0x41,0x00},   //'K' 4BH
  {0x00,0x00,0x7f,0x40,0x40,0x40,0x40,0x00},   //'L' 4CH
  {0x00,0x00,0x7f,0x02,0x0c,0x02,0x7f,0x00},   //'M' 4DH
  {0x00,0x00,0x7f,0x04,0x08,0x10,0x7f,0x00},   //'N' 4EH
  {0x00,0x00,0x3e,0x41,0x41,0x41,0x3e,0x00},   //'O' 4FH
  {0x00,0x00,0x7f,0x09,0x09,0x09,0x06,0x00},   //'P' 50H
  {0x00,0x00,0x3e,0x41,0x51,0x21,0x5e,0x00},   //'Q' 51H
  {0x00,0x00,0x7f,0x09,0x19,0x29,0x46,0x00},   //'R' 52H
  {0x00,0x00,0x46,0x49,0x49,0x49,0x31,0x00},   //'S' 53H
  {0x00,0x00,0x01,0x01,0x7f,0x01,0x01,0x00},   //'T' 54H
  {0x00,0x00,0x3f,0x40,0x40,0x40,0x3f,0x00},   //'U' 55H
  {0x00,0x00,0x1f,0x20,0x40,0x20,0x1f,0x00},   //'V' 56H
  {0x00,0x00,0x3f,0x40,0x38,0x40,0x3f,0x00},   //'W' 57H
  {0x00,0x00,0x63,0x14,0x08,0x14,0x63,0x00},   //'X' 58H
  {0x00,0x00,0x07,0x08,0x70,0x08,0x07,0x00},   //'Y' 59H
  {0x00,0x00,0x61,0x51,0x49,0x45,0x43,0x00},   //'Z' 5AH
  {0x00,0x00,0x00,0x7f,0x41,0x41,0x00,0x00},   //'[' 5BH
  {0x00,0x00,0x02,0x04,0x08,0x10,0x20,0x00},   //'\' 5CH
  {0x00,0x00,0x00,0x41,0x41,0x7f,0x00,0x00},   //']' 5DH
  {0x00,0x00,0x04,0x02,0x01,0x02,0x04,0x00},   //'^' 5EH
  {0x00,0x00,0x40,0x40,0x40,0x40,0x40,0x00},   //'_' 5FH
  {0x00,0x00,0x00,0x00,0x01,0x02,0x04,0x00},   //'`' 60H
  {0x00,0x00,0x32,0x4A,0x4A,0x3C,0x40,0x00},   //'a' 61H
  {0x00,0x00,0x7F,0x48,0x44,0x44,0x38,0x00},   //'b' 62H
  {0x00,0x00,0x38,0x44,0x44,0x44,0x20,0x00},   //'c' 63H
  {0x00,0x00,0x38,0x44,0x44,0x48,0x7F,0x00},   //'d' 64H
  {0x00,0x00,0x38,0x54,0x54,0x54,0x48,0x00},   //'e' 65H
  {0x00,0x00,0x08,0x7E,0x09,0x01,0x02,0x00},   //'f' 66H
  {0x00,0x00,0x0C,0x52,0x52,0x4A,0x3E,0x00},   //'g' 67H
  {0x00,0x00,0x7f,0x08,0x04,0x04,0x78,0x00},   //'h' 68H
  {0x00,0x00,0x00,0x44,0x7D,0x40,0x00,0x00},   //'i' 69H
  {0x00,0x00,0x20,0x40,0x44,0x3D,0x00,0x00},   //'j' 6AH
  {0x00,0x00,0x7f,0x10,0x28,0x44,0x00,0x00},   //'k' 6BH
  {0x00,0x00,0x00,0x41,0x7F,0x40,0x00,0x00},   //'l' 6CH
  {0x00,0x00,0x7C,0x04,0x78,0x04,0x78,0x00},   //'m' 6DH
  {0x00,0x00,0x7C,0x08,0x04,0x04,0x78,0x00},   //'n' 6EH
  {0x00,0x00,0x38,0x44,0x44,0x44,0x38,0x00},   //'o' 6FH
  {0x00,0x00,0x7c,0x14,0x14,0x14,0x08,0x00},   //'p' 70H
  {0x00,0x00,0x08,0x14,0x14,0x18,0x7c,0x00},   //'q' 71H
  {0x00,0x00,0x7c,0x08,0x04,0x04,0x08,0x00},   //'r' 72H
  {0x00,0x00,0x48,0x54,0x54,0x54,0x20,0x00},   //'s' 73H
  {0x00,0x00,0x04,0x3f,0x44,0x40,0x20,0x00},   //'t' 74H
  {0x00,0x00,0x3c,0x40,0x40,0x20,0x7c,0x00},   //'u' 75H
  {0x00,0x00,0x1c,0x20,0x40,0x20,0x1c,0x00},   //'v' 76H
  {0x00,0x00,0x3c,0x40,0x30,0x40,0x3c,0x00},   //'w' 77H
  {0x00,0x00,0x44,0x28,0x10,0x28,0x44,0x00},   //'x' 78H
  {0x00,0x00,0x0c,0x50,0x50,0x50,0x3c,0x00},   //'y' 79H
  {0x00,0x00,0x44,0x64,0x54,0x4c,0x44,0x00},   //'z' 7AH
  {0x00,0x00,0x00,0x41,0x36,0x08,0x00,0x00},   //'>' 7BH
  {0x00,0x00,0x00,0x00,0x7f,0x00,0x00,0x00},   //'|' 7CH
  {0x00,0x00,0x00,0x08,0x36,0x41,0x00,0x00},   //'<' 7DH
  {0x00,0x00,0x08,0x04,0x08,0x10,0x08,0x00},   //'~' 7EH
  {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}};  //' ' 7FH
```

13-3-4 串列式四位元七段顯示器

IRA 中級認證的動作要求中，除七段顯示器外，其餘循跡感測、馬達控制、障礙物偵測、蜂鳴器等週邊均在前面章節中介紹過，要完成任務路徑需求沒有問題，故本節先介紹這一顆內建 PT6761 串列控制晶片的七段顯示器 TOFD-5465GGH-B。

一、 簡介

TOFD-5465GGH-B 是一個內嵌 PT6961 LED 驅動 IC 的四位元七段顯示器，如圖 13-17 所示，此元件具有 SPI 介面，iPOE-A1 可透過圖 13-17(a)的 DIN、CLK、CS 腳位控制顯示內容，使用上非常方便。



(a)正面及腳位圖 (b)背面圖

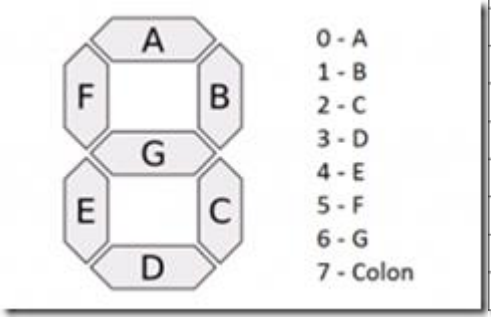
圖 13-17 內嵌 PT6961 的四位元七段顯示器

二、 顯示

1. PT6961 控制器使用顯示記憶體（RAM）來儲存微控制器傳送過來的數據，只要將想顯示的 LED 段儲存在 RAM 中，PT6961 會處理其多工顯示的部分，其中由左至右四位數對應的 RAM 位址分別為 0xC0（千位）、0xC2（百位）、0xC4（拾位）和 0xC6（個位）。

2. 七段顯示器 LED 的各段名稱如圖 13-18 所示，欲顯示一般的數字碼所對應的顯示控制碼表 13-6 所示。

3. TOFD-5465GGH-B 的小數點似乎沒有被連接到 LED 上，故無法顯示，能顯示的只有正中央的冒號部分，此部分由 0xC0 和 0xC2 的第 7 位元控制。



數字	顯示控制碼	
	d_p g f e d c b a	16 進制
0	0 0 1 1 1 1 1 1	0x3F
1	0 0 0 0 0 1 1 0	0x06
2	0 1 0 1 1 0 1 1	0x5B
3	0 1 0 0 1 1 1 1	0x4F
4	0 1 1 0 0 1 1 0	0x66
5	0 1 1 0 1 1 0 1	0x6D
6	0 1 1 1 1 1 0 1	0x7D
7	0 0 1 0 0 1 1 1	0x27
8	0 1 1 1 1 1 1 1	0x7F
9	0 1 1 0 0 1 1 1	0x67
	0 0 0 0 0 0 0 0	0x00

三、PT6961 函式庫

1. 下載與安裝：PT6961 控制器有許多控制命令，為方便使用，一般會使用 PT6961 函式庫，首先請進入底下網站 http://gtbtech.com/?attachment_id=864，將檔案下載後解壓縮至 Arduino 安裝資料夾的 libraries 資料夾中，例如 C:\arduino-1.0.6\libraries。或者也可從書後光碟的程式→libraries 中找到 PT6961 函式庫。

2. 函式庫指令簡介

(1) 建立 PT6961 物件：使用函式庫的第一個動作是指定腳位與物件實體化，可透過底下的命令完成。

```
PT6961 SEG7(DIN 腳位, CLK 腳位, CS 腳位);
```

(2) 初始化 PT6961：使用 PT6961 控制器前，可以底下的指令對 PT6961 進行一些設定，包含設定顯示記憶體為自動遞增、顯示的亮度最亮，同時清除顯示記憶體為 0（不顯示狀態）。

```
SEG7.initDisplay();
```

(3) 初始化顯示記憶體：執行底下指令後可清除顯示記憶體為 0。

```
SEG7.initRAM();
```

(4) 送控制命令給 PT6961，其中 cmd 為控制命令

```
SEG7.sendCmd(char cmd);
```

常用的控制命令如下。

```
char  _DISPLAY_6X12 = 0x02; //  使用 6 個數字 12 個 LED 段模式
char  _DISPLAY_7X11 = 0x03;   //  使用 7 個數字 11 個 LED 段模式
char  _AUTO_INCREMENT = 0x40; //  顯示記憶體位址自動遞增
char  _FIXED_ADDRESS = 0x44;   //  顯示記憶體位址固定
char  _DISPLAY_OFF = 0x80;   //  顯示器關閉(不顯示)
char  _DISPLAY_1_16 = 0x88;    //  顯示亮度為 1/16(最暗)
char  _DISPLAY_2_16 = 0x89; //  顯示亮度為 2/16
char  _DISPLAY_4_16 = 0x8A;     //  顯示亮度為 4/16
char  _DISPLAY_10_16 = 0x8B;    //  顯示亮度為 10/16
char  _DISPLAY_12_16 = 0x8D;    //  顯示亮度為 12/16
char  _DISPLAY_14_16 = 0x8F;    //  顯示亮度為 14/16(最亮)
```

(5) 顯示指定的數字 num，以及是否顯示中間冒號的 LED 段，colon 為 0 不顯示，為 1 顯示「:」。

```
SEG7.sendNum(int num, char colon);
```

(6) 顯示指定的個別數字，以及是否顯示中間冒號的 LED 段。

SEG7.sendDigits(char digit1, char digit2, char digit3, char digit4, char colon);

(7) 顯示指定位數的 LED 段控制碼，其中 digit 為顯示記憶體位址，由左至右四位數對應的 RAM 位址分別為 0xC0（千位）、0xC2（百位）、0xC4（拾位）和 0xC6（個位）；val 為顯示控制碼，可參考表 13-6 的轉碼。

SEG7.sendSegment(char digit, char val);

(8) 顯示指定位數的數字，其中 digit 為顯示記憶體位址，val 為顯示的數字，colon 指定是否顯示中間冒號的 LED 段（註：只有千位及百位數的 colon 有作用）。

SEG7.sendDigit(char digit, char val, char colon);