

LCD 模組之應用

本章內容豐富，主要包括兩部分：

硬體部分：

LCD 模組的結構，及其與 8x51 之介面。

LCD 模組的指令、編碼與其應用。

認識中文 LCD 模組與其應用。

程式與實作部分：

基本 LCD 模組的應用、自建字型等。

13-1

認識 LCD 模組

LCD 在應用

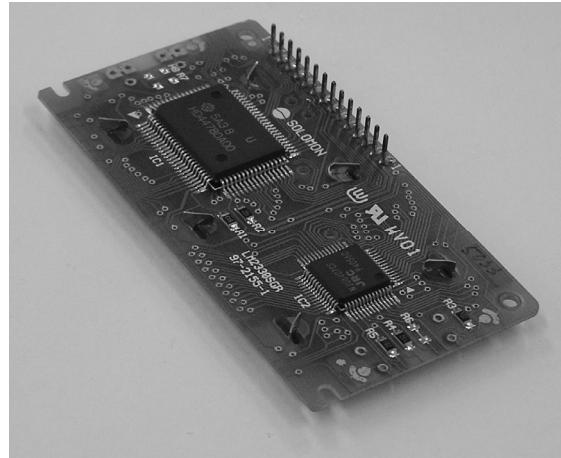
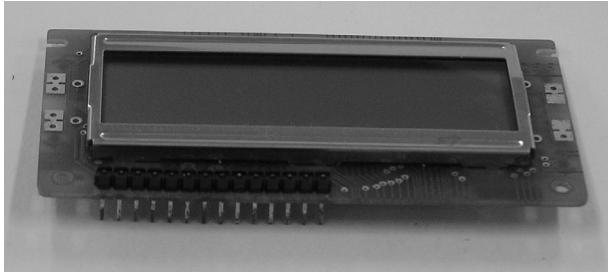


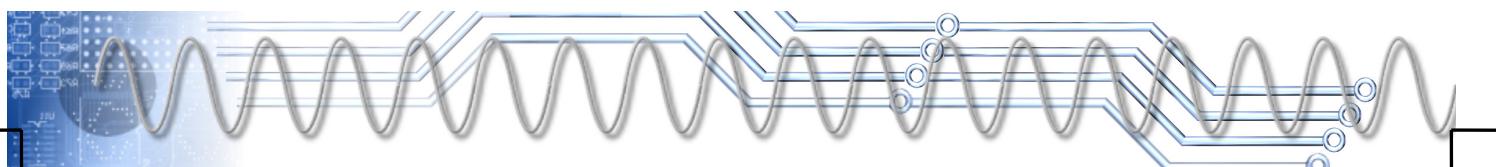
圖1 LCD 模組(左邊為正面圖、右邊為背面圖)

LCD(Liquid Crystal Display)為液晶顯示面板，由於 LCD 的控制須專用的驅動電路，且 LCD 面板的接線須特殊技巧，加上 LCD 面板結構較脆弱，通常不會單獨使用。而是將 LCD 面板、驅動與控制電路組合而成一個 **LCM**(Liquid Crystal Display Moulde，簡稱為 LCM)。LCM 是一種很省電的顯示裝置，常被應用在數位或微電腦控制的系統，做為簡易的人機介面，如圖 1 所示為常用的 LCD 模組。

LCM 基本資料

LCM 的種類頗多，而在學校與訓練單位所採用的 LCM，大都是以日商日立公司的控制器(HD44780)所組成的 LCM，其內部結構如圖 2 所示，而其特性如下：

- 內建 80bytes 資料顯示記憶體(Data Display RAM，簡稱 **DD RAM**)，可顯示 16 字×1 列、20 字×1 列、16 字×2 列、20 字×2 列、40 字×2 列等模式。
- 內建字型產生器(Character Generate ROM，簡稱 **CG ROM**)，可產生 160 個 5×7 字型，如表 1 所示。
- 自建字型產生器(Character Generate RAM，簡稱 **CG RAM**)，可



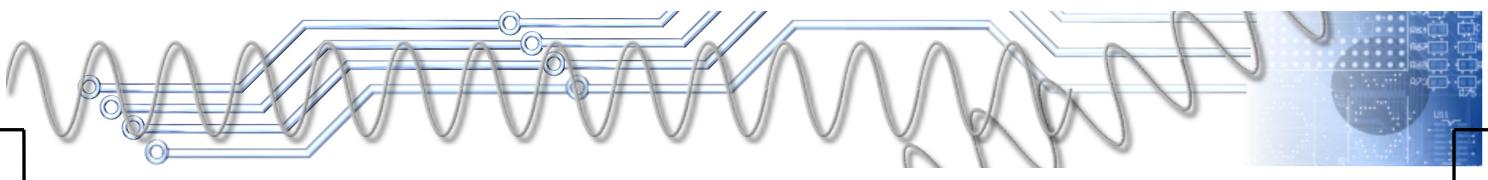
由使用者自建 8 個 5×7 字型。

		高四位元																
		0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111																
低四位元	CG RAM (1)	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
	CG RAM (2)	0001	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (3)	0010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (4)	0011	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (5)	0100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (6)	0101	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (7)	0110	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (8)	0111	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (1)	1000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (2)	1001	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (3)	1010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (4)	1011	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (5)	1100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (6)	1101	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (7)	1110	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
	CG RAM (8)	1111	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	

表 1 LCD 字型編碼表

LCM 內部結構

如圖 2 所示，LCM 內部結構，如下說明：



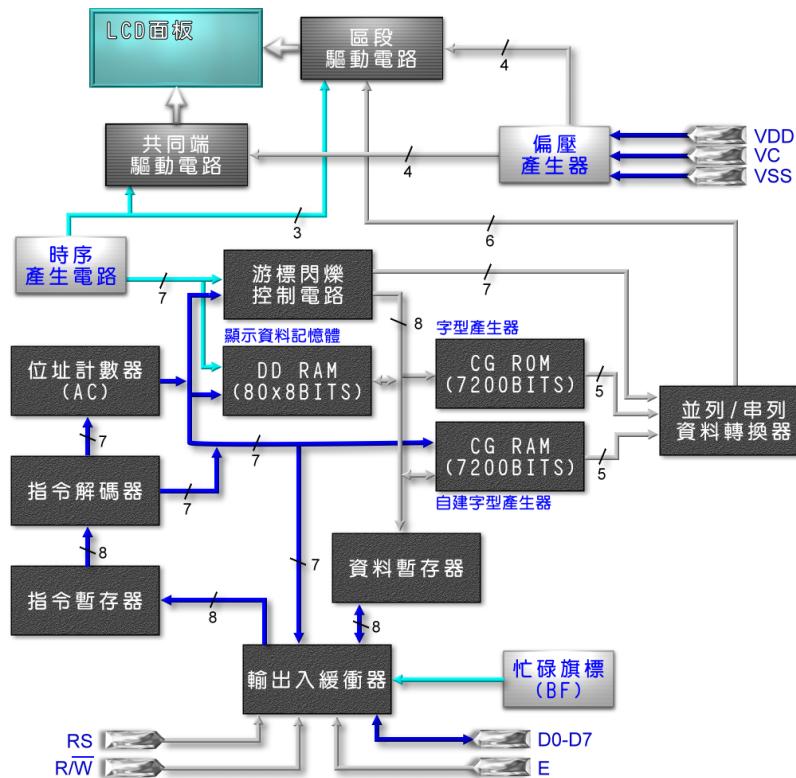
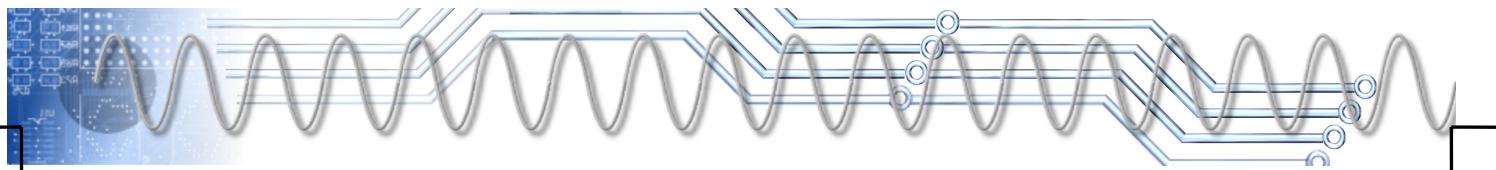


圖2 HD44780 LCM 內部結構

- **輸出入緩衝器**為 LCM 的大門，所有資料與控制信號都須透過本單元才得以進出 LCM。
- **指令暫存器** (Instruction Register, 簡稱 **IR**)為一個 8 位元暫存器，其功能是存放微處理器所送入之 LCM 指令、DD RAM 或 CG RAM 之位址。當我們要將資料輸入到 DD RAM 或 CG RAM 時，首先將資料放入資料暫存器，再把指令與 DD RAM 或 CG RAM 之位址放入本暫存器，即可將該資料輸入到 DD RAM 或 CG RAM。同樣地，若要讀取 DD RAM 或 CG RAM 的資料，則將指令與 DD RAM 或 CG RAM 之位址放入本暫存器，即可於資料暫存器中，取得該位址的資料。
- **指令解碼器**的功能是將指令暫存器裡的指令解碼，以獲得所要操作 DD RAM 或 CG RAM 的位址。
- **資料暫存器**(Data Register, 簡稱 **DR**)連接 LCM 內部資料匯流排，DD RAM 或 CG RAM 的資料存取，都需透過本暫存器。當 CPU 讀取 DR 內容後，DR 將自動載入下一個位址的內容。因此，若要連續讀取 DD



RAM 或 CG RAM 的資料，只要指定其起始的位址即可。

- **位址計數器**(Address Counter，簡稱 AC)連接 LCM 內部位址匯流排，DD RAM 或 CG RAM 的操作，都需透過本計數器所提供的位址來定址。當存取 DD RAM 或 CG RAM 時，AC 具有自動增加的功能，也就是自動指到下一個記憶位址。當 RS=0、R/W=1 時，進入讀取 AC 內容的狀態，AC 裡的資料將輸出到 D0 到 D7 資料匯流排上。
- **忙碌旗標**(Busy Flag，簡稱 BF)用以表示 LCM 當時的狀態，若 BF=1，則表示 LCM 處於忙碌狀態，無法接受外部指令或資料；若 BF=0，則可接受外部指令或資料。
- **顯示資料記憶體** (Display Data RAM，簡稱 DD RAM)映射所要顯示的資料，為 LCM 的主戰場。實際上，在本記憶體裡存放的是所要顯示資料的 ASCII 碼，再以該 ASCII 碼為地址，到 CG ROM 或 HCG ROM 裡找到該字型的顯示編碼。DD RAM 的記憶體位址，分為四列，第一列的位址為 0x80~0x8f、第二列的位址為 0x90~0x9f、第三列的位址為 0xa0~0xaf、第四列的位址為 0xb0~0xbf，其中第一列與第二列直接映對到 LCD 面板，如下：

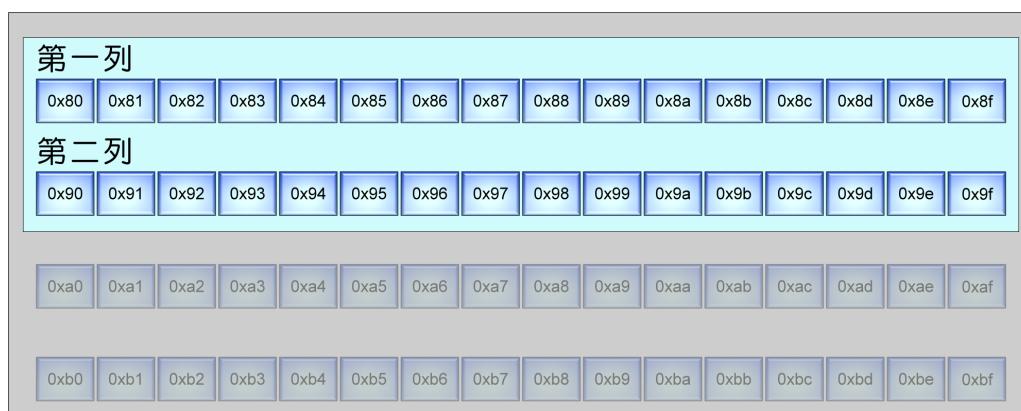
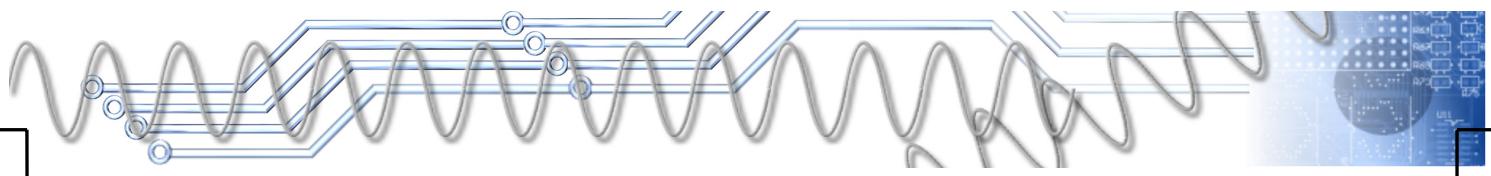


圖3 DDRAM 記憶體位址

而第三列與第四列空有記憶體位置，在 LCD 面板上沒有實際映對的顯示區，就當它不存在。

- **字型產生器**(Character Generate ROM，簡稱 CG ROM)為一個唯讀記憶體，其中包括所有預置的顯示資料的編碼(如表 1 為其編碼表)，而這個編碼表就是 ASCII 編碼。



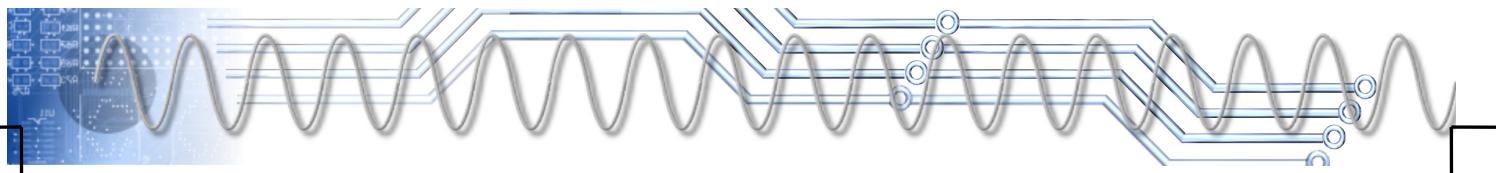
- **自建字型產生器**(Character Generate RAM，簡稱 **CG RAM**)為一個隨機存取記憶體，其功能是提供存放使用者所建立的字型樣板(pattern)，最多可自建 8 個字型(在中文 LCM 為 40 個 16×16 字型)。如圖 4 所示，分別為自建的「◎」及「±」：

CG RAM 位址	7	6	5	4	3	2	1	0	資料編碼
0 0 0 0 0 0	*	*	*						00000
0 0 0 0 0 1	*	*	*	■	■	■	■	■	01110
0 0 0 0 1 0	*	*	*	■	■	■	■	■	10101
0 0 0 0 1 1	*	*	*	■	■	■	■	■	11111
0 0 0 1 0 0	*	*	*	■	■	■	■	■	10101
0 0 0 1 0 1	*	*	*	■	■	■	■	■	11011
0 0 0 1 1 0	*	*	*	■	■	■	■	■	01110
0 0 0 1 1 1	*	*	*	■	■	■	■	■	游標位置
0 0 1 0 0 0	*	*	*						00100
0 0 1 0 0 1	*	*	*						00100
0 0 1 0 1 0	*	*	*	■	■	■	■	■	11111
0 0 1 0 1 1	*	*	*	■	■	■	■	■	00100
0 0 1 1 0 0	*	*	*	■	■	■	■	■	00100
0 0 1 1 0 1	*	*	*	■	■	■	■	■	00000
0 0 1 1 1 0	*	*	*	■	■	■	■	■	11111
0 0 1 1 1 1	*	*	*	■	■	■	■	■	游標位置

圖4 自建字型

每個字型由 8 組資料編碼所構成，每個資料編碼僅用到前 5 個位元(4-0)，若要顯示則於該位置標示「1」，不要顯示則於該位置標示「0」，最後一組編碼通常是空白，留給游標使用。而這八組資料編碼在 CG RAM 的位址是以其低三位元來編列，分別為 000 到 111。CG RAM 提供 8 個字型的位址，總共 8×8 個位元組記憶體空間，高三位元 000 代表第一個自建字型、001 代表第二個自建字型...，111 代表第八個自建字型。

- **串列/並列資料轉換器**(parallel-serial converter)的功能是將從 CG RAM 或 CG ROM 所取出之並列顯示資料，轉換成串列資料，以提供驅動電路推動 LCD 面板。
- **游標閃爍控制電路**(cursor/blink controller)的功能是用以控制游標，以及閃爍字型的產生。
- **時序產生電路**(timing generator)的功能是產生 LCM 所須之時鐘脈波。
- **偏壓產生電路**(bias voltage generator)的功能是提供推動 LCD 面板所須之偏壓。



- **共同端驅動電路**(common driver)的功能是提供 LCD 面板共同端之掃瞄信號。
- **區段驅動電路**(segment dirver)的功能是提供 LCD 面板之顯示信號。
- **LCD 面板**(LCD panel)為一點矩陣式液晶顯示面板。

● LCM 之接腳

如圖 1 所示，LCM 包括 14 隻接腳，如下說明：

- **電源接腳**：第 2 腳 VDD 為電源接腳，連接+5V 即可，而第 1 腳 VSS 為接地接腳。第 3 腳 Vo 為面板明亮度調整接腳，當此接腳的電壓越低，則面板明亮度越高。我們可以利用一個 $10k\Omega$ 可變電阻(或半固定電阻)，做為明亮度調整電路，如圖 5 所示。不同廠牌的 LCM，其明亮度調整方式不見得一樣。大部分英文 LCM，若將 Vo 電壓調低，面板將更明亮，甚至直接與 VSS 及 GND 連接。本章所採用的 WG14432B 中文 LCM，其明亮度調整方式剛好相反，若將 Vo 電壓調高，面板將更明亮，甚至直接與 VDD 及+5V 連接，但 Vo 不可空接。

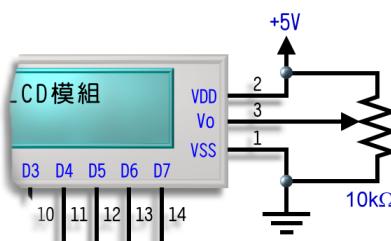
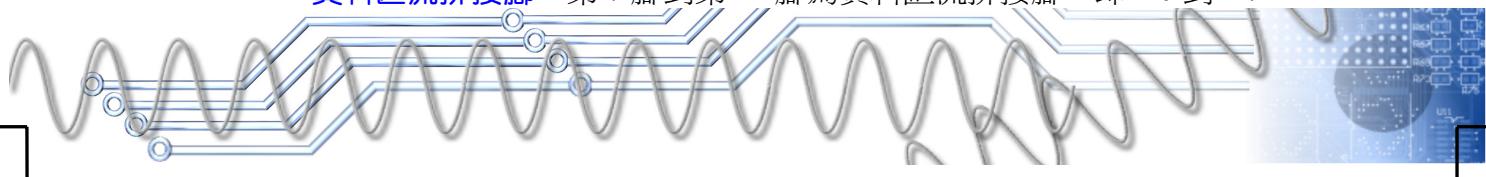


圖5 明亮度控制

- **暫存器選擇接腳**：RS 腳(第 4 腳)為 LCM 內部暫存器選擇接腳(即 Register Selector，簡稱為 RS)，當 RS=0 時，匯流排將連接到 LCM 內部暫存器的指令暫存器 IR(即 Instruction Register)；當 RS=1 時，匯流排將連接到 LCM 內部暫存器的資料暫存器 DR(即 Data Register)。
- **讀寫控制接腳**：R/W 腳(第 5 腳)為匯流排方向控制接腳，當 R/W=0 時，匯流排將由微處理器輸入到 LCM 內部，以進行資料/指令寫入 LCM；當 R/W=1 時，匯流排將由 LCM 內部讀取資料。
- **致能接腳**：E 腳(第 6 腳)為 LCM 的致能信號，此為負緣觸發式接腳。
- **資料匯流排接腳**：第 7 腳到第 14 腳為資料匯流排接腳，即 D0 到 D7。



● LCM 之包裝

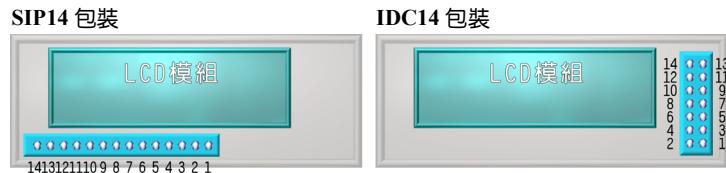


圖6 LCM 之包裝

常見的 LCM 接腳包裝有兩種，如上圖所示，第一種採單排接腳包裝(SIP14)，第二種採雙排接腳包裝(IDC14)。至於接腳的實際位置，不同的廠牌、型號各有不同，使用之前必須詳閱其 data sheet。

13-2

中文 LCD 模組

LCD 之應用

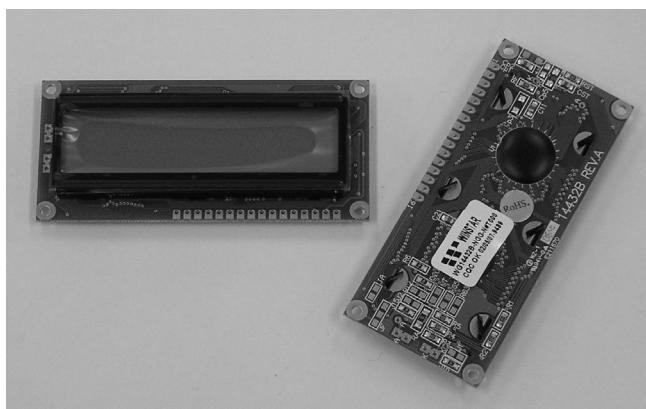
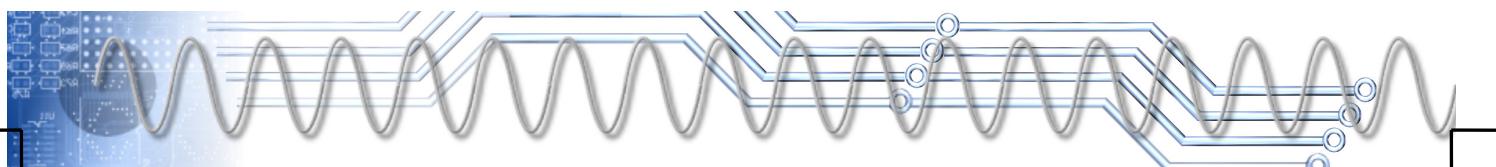


圖7 中文 LCM—WG14432B-YYH-N

一般的 LCM 都會預留 8×8 個位元組記憶體空間(CG RAM)，讓使用者自建圖形或字型， 8×8 個位元組可建立 8 個 5×8 的圖案，若要勉強製作中文字型，則須要兩個 5×8 的圖案才能構成一個中文字，也就是最多可自建四個中文字。顯然這不是個好主意！而中文市場日漸增長，LCM 廠商絕不會坐視不管，所以，近年來中文 LCM 大行其道。儘管如此，中文 LCM 與一般非中文 LCM 之差別不大，只是中文 LCM 多出了中文編碼(big-5 碼)的 ROM，外表上很難分辨其差異，而驅動的指令、外部接腳等，並無不同！以下將介紹國產華凌光電股份有限公司 (www.winstar.com.tw) 型號 WG14432B-NGG-N#T000 之中文 LCM 模組：

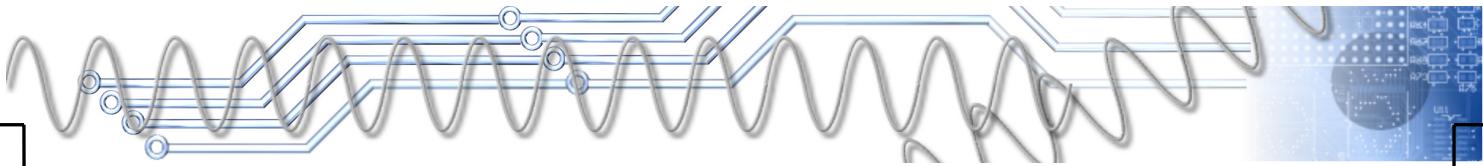


● 型號之編號方式

W G 1 4 4 3 2 B - N G G - N #T000

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

項目	功 能	說 明	
①	廠牌	W 代表 Winstar Display 公司所生產的產品。	
②	顯示種類	H 代表文字形式 G 代表圖形形式	
③	顯示字型	144×32 點陣	
④	模組序號	零件編號	
⑤	背光形式	N 代表沒有背光 B 代表 EL，藍灰色 A 代表 LED，琥珀色 D 代表 EL，綠色 R 代表 LED，紅色 W 代表 EL，白色 O 代表 LED，橙色 F 代表 CCFL，白色 G 代表 LED，綠色 Y 代表 LED，黃綠色	
⑥	LCD 模式	B 代表 TN 正，灰色 N 代表 TN 負 F 代表 FSTN 正 G 代表 STN 正，灰色 T 代表 FSTN 負 Y 代表 STN 正，黃綠色 M 代表 STN 負，藍色	
⑦	LCD 偏光鏡 溫度範圍 展示方向	A 代表反射式，一般溫度範圍，6 點鐘方向 D 代表反射式，一般溫度範圍，12 點鐘方向 G 代表反射式，寬溫度範圍，6 點鐘方向 J 代表反射式，寬溫度範圍，12 點鐘方向 B 代表透射式，一般溫度範圍，6 點鐘方向 E 代表透射式，一般溫度範圍，12 點鐘方向 H 代表透射式，寬溫度範圍，6 點鐘方向 K 代表透射式，寬溫度範圍，12 點鐘方向 C 代表透射式，一般溫度範圍，6 點鐘方向 F 代表透射式，一般溫度範圍，12 點鐘方向 I 代表透射式，寬溫度範圍，6 點鐘方向 L 代表透射式，寬溫度範圍，12 點鐘方向	
⑧	特殊碼	N 代表不需要負電壓	
⑨	地區碼	T000 代表台灣區的繁體文編碼。	



接腳表

接腳號碼	接腳名稱	準位	說明
1	VSS	0V	接地
2	VDD	5.0V	LCD 模組邏輯電路電源(+5V)
3	Vo	—	LCD 面板明亮度電源
4	RS	H/L	RS=1，處理資料 RS=0，處理指令
5	R/W	H/L	R/W=1，讀取 LCM(MPU←LCM) R/W=0，寫入 LCM(MPU→LCM)
6	E	H/L	啟能信號
7	DB0	H/L	匯流排
8	DB1	H/L	匯流排
9	DB2	H/L	匯流排
10	DB3	H/L	匯流排
11	DB4	H/L	匯流排
12	DB5	H/L	匯流排
13	DB6	H/L	匯流排
14	DB7	H/L	匯流排
15	A	—	背光 LED 之正端
16	K	—	背光 LED 之負端

內部架構

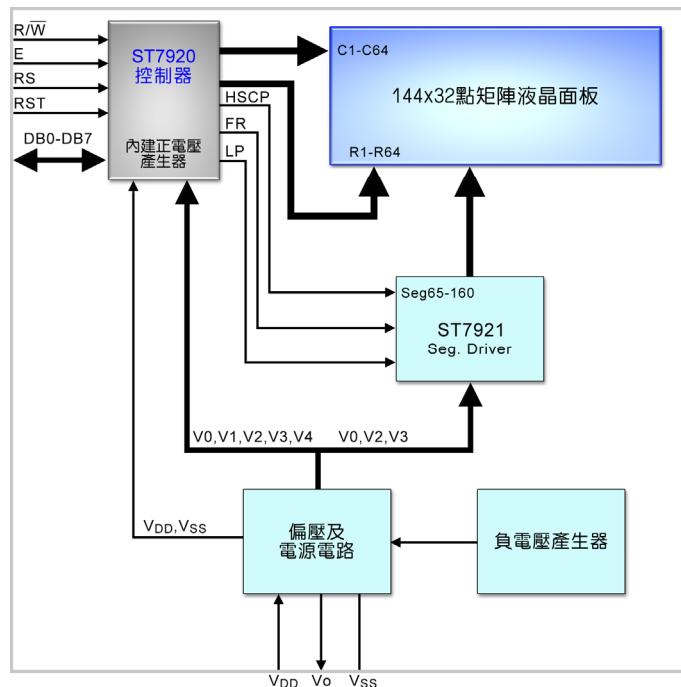
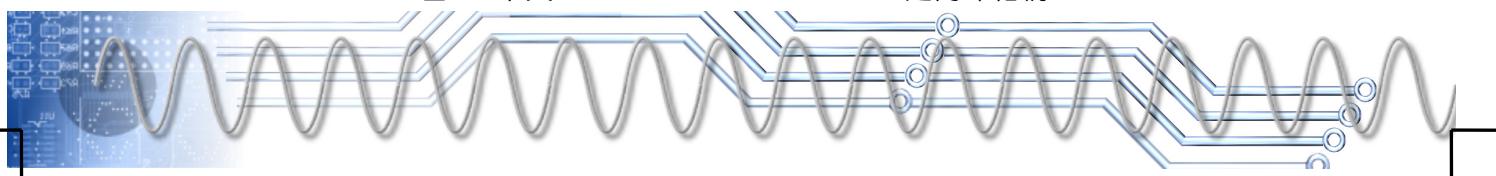


圖8 中文 LCM-WG14432B-YYH-N 之內部結構

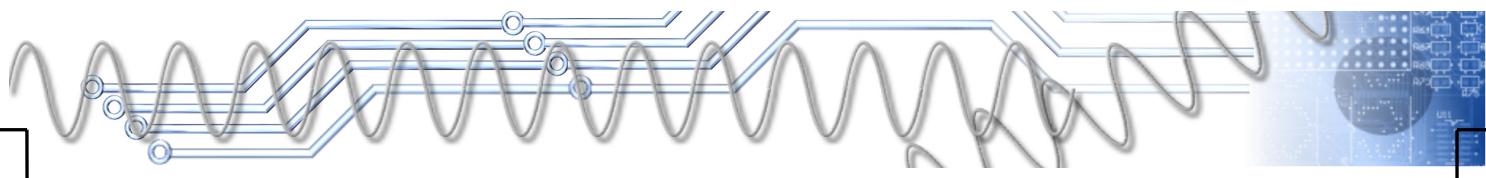


如圖 8 所示，**WG14432B-NGG-N#T00** 系列採用 LCD 驅動晶片大廠 **矽創電子股份有限公司**(www.sitronix.com.tw)的中文 LCD 驅動控制器 ST7920，不是前面所介紹的 HD44780，但其指令與操作方式幾乎完全一樣，關於 ST7920 控制器，可參閱隨書光碟中的 data sheet。

● 字型產生器

ST7920 控制器提供具有 8192 個 16×16 字型(可為中文字)及 126 個 8×16 字型(英文字母與數字)的記憶體(ROM)，如此將可支援多語文的應用，例如中文與英文並行。對於連續的兩個位元組，可用來指定一個 16×16 字型或兩個 8×16 字型，而 8×16 字型就是俗稱的**半高字型**(half-height characters)，所指定的字型碼將被寫入 DDRAM，同時從 CGROM 或 HCGROM 對應到其字型，即可顯示之。

- **自建字型產生器**：ST7920 控制器提供四組使用者定義字型(16×16 字型)的記憶體(即 **CGRAM**)，而其顯示方式與前述應對到 CGROM 或 HCGROM 的方式一樣。
- **自建圖示產生器**：ST7920 控制器提供 240 個圖示(Icon)，包括 15 組 IRAM(即 Icon RAM)的位址，每組 IRAM 位址是包含 16 位元資料，而其資料安排的順序是高 8 位元(D15~D8)先、低 8 位元 (D7~D0)後。
- **顯示資料記憶體**：DDRAM(Display Data RAM)是指顯示資料記憶體，ST7920 控制器裡的 DDRAM，可以儲存 16 個 16×16 字型(4 列)或 32 個 8×16 字型(4 列)，當然，在 **WG14432B-YYH-N** 的 LCD 面板裡，最多只能顯示兩列。文字資料碼儲存在 DDRAM 裡，而對應到 CGROM、HCGROM 及 CGRAM。ST7920 能夠顯示 HCGROM 的半高字型、CGRAM 裡使用者自行定義的字型，以及 CGROM 裡的 16×16 全高字型，如下說明：
 - 顯示 HCGROM 半高字型：寫入 2 bytes 資料到 DDRAM，則顯示 2 個 8×16 字型，每個 byte 代表一個字型；而其文字資料碼為 0x02~0x7f。
 - 顯示 CGRAM 字型：寫入 2 bytes 資料到 DDRAM，則顯示 1 個 16×16 字型；而其文字資料碼只能為 0x0000、0x0002、0x0004 及 0x0006。



- 顯示 CGROM 字型：寫入 2 bytes 資料到 DDRAM，則顯示 1 個 16×16 字型。若採 big-5 碼，其文字資料碼為 $0xa140 \sim 0xd75f$ ；若採大陸的 GB 碼，其文字資料碼為 $0xa1a0 \sim 0xf7ff$ 。
- 顯示圖案記憶體：GDRAM(即 Graphic Display RAM)提供 64×256 位元、採位元對應記憶體空間。GDRAM 的位址是由連續的兩 bytes 所組成，分為垂直位址與水平位址，兩個 bytes 資料將寫入 GDRAM 的同一個位址裡，其步驟如下：
 1. 設定 GDRAM 的垂直位址。
 2. 設定 GDRAM 的水平位址。
 3. 將 D15~D8 寫入 GDRAM(第一個位元組)。
 4. 將 D7~D0 寫入 GDRAM(第二個位元組)。

除了華凌光電股份有限公司外，國內還有很多家提供中文 LCM 的廠商，例如雄鐸科技股份有限公司(www.sdec.com.tw)等，其 S14B32 系列，也是採用 ST7920 控制晶片，與 **WG14432B-NGG-N#T00** 完全相容。

13-3 LCM 控制指令 @ LCD 在應用

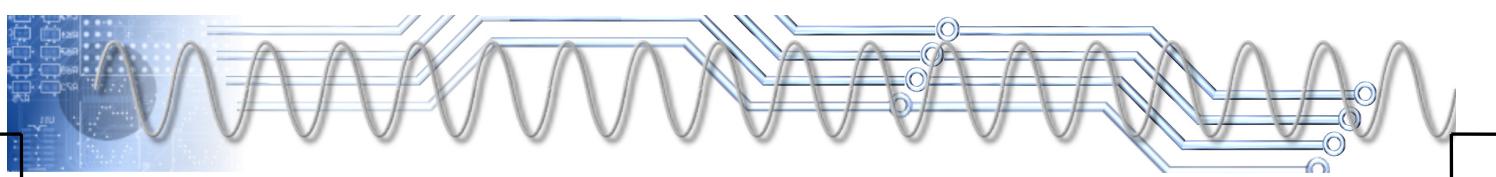
LCM 控制指令只有 11 個控制指令，如表 2 所示(依據 ST7920 之 data sheet)，在本單元裡，將依序介紹這 11 個指令：

● 清除顯示幕(CLEAN DISPLAY)

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 00000001(即 0x01)，其動作為：

1. 讓顯示幕變成空白，LCM 將會把 DDRAM 全部填入 20H(即空白)。
2. 將游標移至左上角(HOME)。
3. 使位址計數器(AC)歸零。
4. 整個執行時間需要 1.6 毫秒。



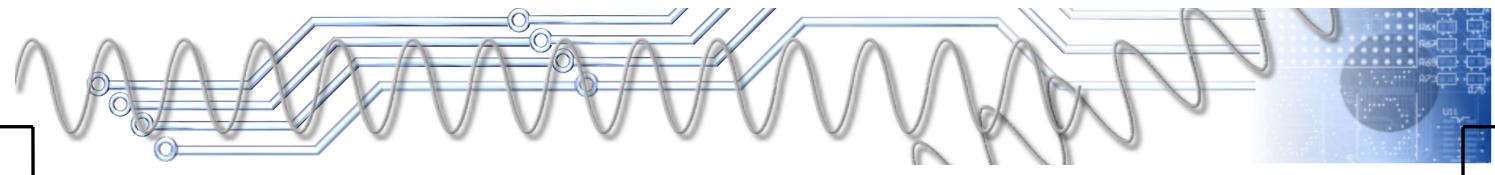
功 能	控制線		匯 流 排								執 行 時 間							
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0								
清除顯示幕	0	0	0	0	0	0	0	0	1		1.6ms							
清除顯示幕，並把游標移至左上角																		
游標歸位	0	0	0	0	0	0	0	0	1	x	72μs							
游標移至左上角，顯示內容不變																		
設定輸入模式	0	0	0	0	0	0	0	1	I/D	S	72μs							
I/D=1：位址遞增、I/D=0：位址遞減																		
S=1：顯示幕移位、S=0：顯示幕不移位																		
開關顯示幕	0	0	0	0	0	0	1	D	C	B	72μs							
D=1 開啓顯示幕、D=0 關閉顯示幕																		
C=1 開啓游標、C=0 關閉游標																		
B=1 游標所在位置之字元反白、B=0 游標所在位置之字元不反白																		
移位方式	0	0	0	0	0	1	S/C	R/L	x	x	72μs							
S/C=1 顯示幕移位、S/C=0 游標移位																		
R/L=1 向右移、R/L=0 向左移																		
功能設定	0	0	0	0	1	DL	x	RE	x	x	72μs							
DL=1 資料長度為 8 位元、DL=0 資料長度為 4 位元																		
RE=1 採用延伸指令、RE=0 採用一般指令，請參閱隨書光碟裡的 ST7920 之 data sheet，在此不探討延伸指令																		
CGRAM 定址	0	0	0	1	CG RAM 位址				72μs									
將所要操作之 CG RAM 位址放入位址計數器																		
DDRAM 定址	0	0	1	DD RAM 位址				72μs										
將所要操作之 DD RAM 位址放入位址計數器，第 1 行 80~8F，第 2 行 90~9F																		
讀取 BF 與 AC	0	1	BF	位址計數器內容				0μs										
讀取位址計數器，並查詢 LCM 是否忙碌																		
BF=1 表示 LCM 忙碌、BF=0 表示 LCM 可接受指令或資料																		
寫入資料	1	0	所要寫入之資料				72μs											
將資料寫入內部記憶體(GDRAM、IRAM、CG RAM、DD RAM)																		
讀取資料	1	1	所要讀取之資料				72μs											
讀取內部記憶體(GDRAM、IRAM、CG RAM、DD RAM)之資料																		

執行時間是依據 ST7920 在 540kHz 下的執行時間

表 2 LCM 指令速查表

游標歸位

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	*



RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 0000001*(即 0x02 或 0x03)，其中的「*」代表可為 0 或 1，而其動作為：

1. 將游標移至左上角(HOME)，但 DD RAM 的內容不變。
2. 使位址計數器(AC)歸零。
3. 整個執行時間需要 72 微秒。

設定輸入模式

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	I/D	S

RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 000001 I/D S，其中的 I/D 與 S 位元如下：

I/D	S	功 能
0	0	顯示的字元不動，游標左移，AC-1
0	1	顯示的字元右移，游標不動，AC 不變
1	0	顯示的字元不動，游標右移，AC+1
1	1	顯示的字元左移，游標不動，AC 不變

設定顯示幕

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	D	C	B

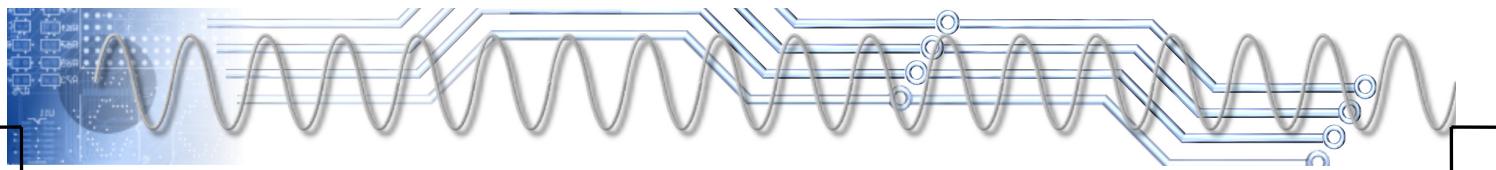
RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 00001 D C B，其中的 D、C 與 B 位元如下：

1. D 位元顯示幕控制開關，D=1 時可開啟顯示幕、D=0 時則關閉顯示幕。
2. C 位元游標控制開關，C=1 時可顯示游標、C=0 時則不顯示游標。
3. B 位元字元反白控制開關，B=1 時則游標所在之字元將反白、B=0 時則游標所在之字元將不反白。
4. 整個執行時間需要 72 微秒。

設定移位方式

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	S/C	R/L	*	*

RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 0001* S/C



R/L，其中的「*」代表可為 0 或 1，而 S/C 與 R/L 位元如下：

S/C	R/L	功 能
0	0	游標左移，AC-1
0	1	游標右移，AC+1
1	0	整個顯示幕左移
1	1	整個顯示幕右移

功能設定

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	*	RE	*	*

RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 001 DL

* RE **，其中的「*」代表可為 0 或 1，而 DL 與 RE 位元如下：

1. DL 位元為傳送的資料長度設定，DL =1 則採 8 位元方式的資料傳送，DL =0 則採 4 位元方式的資料傳送，其中高四位元先傳送，再傳送低四位元。
2. RE 位元為延伸指令設定位元，RE =1 採延伸指令，RE=0 則採一般指令。
3. 整個執行時間需要 72 微秒。

CG RAM 定址

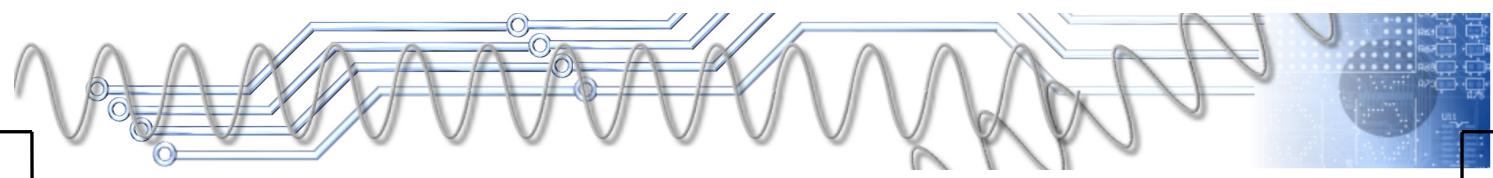
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	A5	A4	A3	A2	A1	A0

RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 01A5 A4 A3 A2 A1 A0，其中的 A5 A4 A3 A2 A1 A0 代表所要操作的 CG RAM 位址。緊接於本指令之後，即可將所要輸入的資料，輸入到這個位址。而整個執行時間需要 72 微秒。

DD RAM 定址

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	A6	A5	A4	A3	A2	A1	A0

RS=0、R/W=0 是執行指令寫入的操作，而資料匯流排上的指令為 1 A6 A5 A4 A3 A2 A1 A0，其中的 A6 A5 A4 A3 A2 A1 A0 代表所要操作的 DD RAM 位址。緊接於本指令之後，即可將所要輸入的資料，



輸入到這個位址。而整個執行時間需要 72 微秒。

● 讀取 BF 與 AC

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	1	BF	A6	A5	A4	A3	A2	A1	A0

RS=0、R/W=1 是執行讀取的操作，這時候，LCM 的忙碌旗標 BF 將放置在資料匯流排上的 D7 位元，而 LCM 的位址計數器內容也將放置在資料匯流排上的 D6-D0 位元，分別為 A6 A5 A4 A3 A2 A1 A0。整個執行時間需要 0 微秒(依據 ST7920 之 data sheet)。

● 資料寫入

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0	D7	D6	D5	D4	D3	D2	D1	D0

RS=1、R/W=0 是執行資料寫入的操作，這時候，在資料匯流排上的資料將寫入前一個指令所指定的 DD RAM 或 CG RAM 位址裡。整個執行時間需要 72 微秒。

● 讀取資料

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	1	D7	D6	D5	D4	D3	D2	D1	D0

RS=1、R/W=1 是執行讀取資料的操作，這時候，前一個指令所指定的 DD RAM 或 CG RAM 位址中的資料，將被放置在資料匯流排上。而讀取資料之後，位址計數器將自動加 1，指向下一個位址。整個執行時間需要 72 微秒。

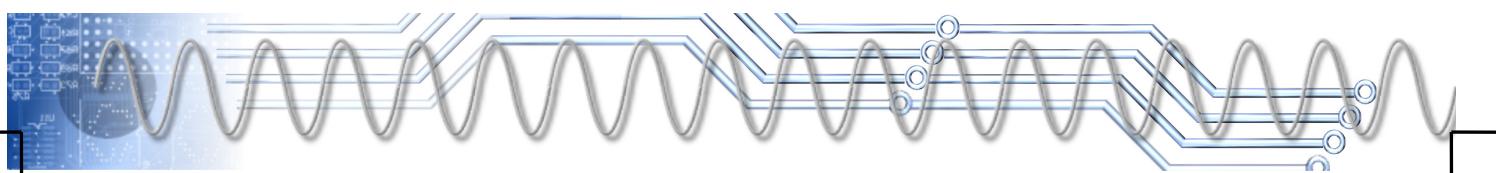
13-4

LCM 之初始設定與常用函數

LCD 之應用



基本上，LCM 是一個小系統，不管是 HD44780 或是 ST79202 都是微處理機。既然是微處理機，就有初始設定的問題，而 LCM 的資料模式有 8 位元與 4 位元(可利用 DL 位元來設定)，兩種模式的初始設定有些許差異，如下說明：



8 位元模式之初始設定

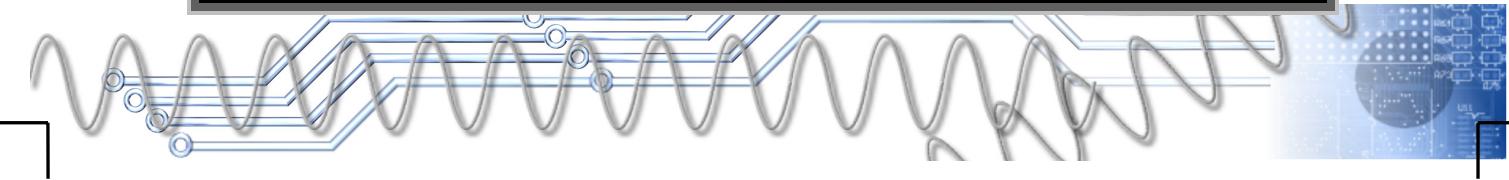


圖9 8 位元模式之初始設定

根據初始設定的流程，若採 8 位元的資料傳輸模式(LCM 與微處理機之間的資料匯流排為 8 位元)，而 8x51 的 Port 0 連接匯流排、P3.0 連接 E、P3.1 連接 RW、P3.2 連接 RS，則在送入電源後，我們可以下列指令進行初始設定：

```

sbit E = P3^0;           // 宣告 E 的位址
sbit RW = P3^1;          // 宣告 RW 的位址
sbit RS = P3^2;          // 宣告 RS 的位址
//=====
main()
{ RS = 0; RW=0;          // 寫入指令模式
  E = 1;                  // 致能
  P0 = 0x30;              // 設定功能
  
```



```
check_BF();           // 完成
//=====
RS = 0; RW=0;          // 寫入指令模式
E = 1;                // 致能
P0 = 0x30;             // 設定功能
check_BF();           // 完成
//=====
RS = 0; RW=0;          // 寫入指令模式
E = 1;                // 致能
P0 = 0x08;             // 關閉顯示功能
check_BF();           // 完成
//=====
RS = 0; RW=0;          // 寫入指令模式
E = 1;                // 致能
P0 = 0x01;             // 清除顯示幕
check_BF();           // 完成
//=====
RS = 0; RW=0;          // 寫入指令模式
E = 1;                // 致能
P0 = 0x06;             // 設定輸入模式
check_BF();           // 完成
//=====
```

4 位元模式之初始設定

根據初始設定的流程，若採 4 位元的資料傳輸模式(LCM 與微處理機之間的資料匯流排為 4 位元)，很明顯地，每個指令分兩次送入 LCM，在送入電源後，我們可以下列指令進行初始設定：

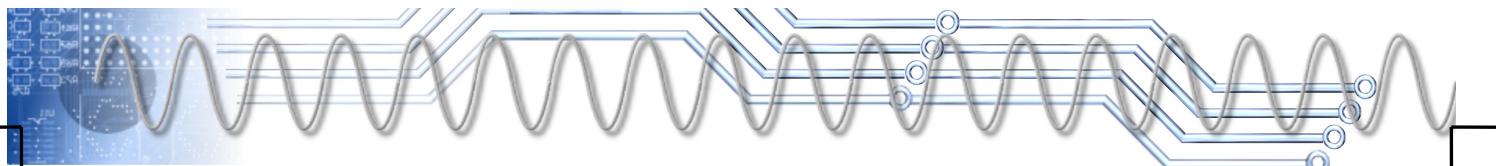




圖10 4位元模式之初始設定

```

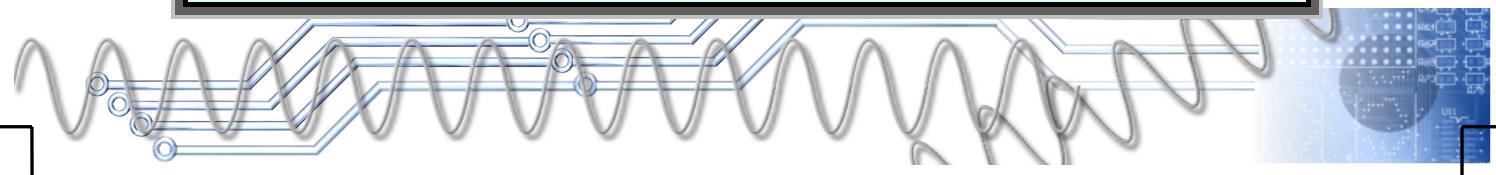
sbit E = P3^0;           // 宣告 E 的位址
sbit RW = P3^1;          // 宣告 RW 的位址
sbit RS = P3^2;          // 宣告 RS 的位址
=====

main()
{ RS = 0; RW=0;          // 寫入指令模式
  E = 1;                 // 致能
  P0 = 0x20;              // 設定功能
  check_BF();             // 完成
=====

  RS = 0; RW=0;          // 寫入指令模式
  E = 1;                 // 致能
  P0 = 0x00;              // 設定功能
  check_BF();             // 完成
=====

  RS = 0; RW=0;          // 寫入指令模式

```



```

E = 1;                                // 致能
P0 = 0x20;                            // 設定功能
check_BF();                           // 完成
//=====
RS = 0; RW=0;                          // 寫入指令模式
E = 1;                                // 致能
P0 = 0x00;                            // 關閉顯示功能
check_BF();                           // 完成
//=====
RS = 0; RW=0;                          // 寫入指令模式
E = 1;                                // 致能
P0 = 0x80;                            // 關閉顯示功能
check_BF();                           // 完成
//=====
RS = 0; RW=0;                          // 寫入指令模式
E = 1;                                // 致能
P0 = 0x00;                            // 清除顯示幕
check_BF();                           // 完成
//=====
RS = 0; RW=0;                          // 寫入指令模式
E = 1;                                // 致能
P0 = 0x10;                            // 清除顯示幕
check_BF();                           // 完成
//=====
RS = 0; RW=0;                          // 寫入指令模式
E = 1;                                // 致能
P0 = 0x00;                            // 設定輸入模式
check_BF();                           // 完成
//=====
RS = 0; RW=0;                          // 寫入指令模式
E = 1;                                // 致能
P0 = 0x06;                            // 設定輸入模式
check_BF();                           // 完成
//=====

```

● 寫入指令函數

不管是 8 位元還是 4 位元的資料傳輸模式，都是一堆重複的動作，從上面的程式看起來，有點笨！我們可利用函數來執行寫入指令的動作，以簡化初始設定的程式，如下：



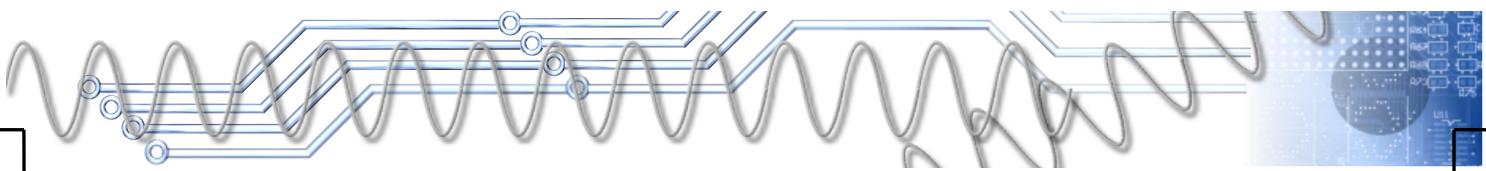
```
=====寫入指令函數=====
void write_inst(char inst)
{ RS = 0; RW=0;           // 寫入指令模式
  E = 1;                  // 致能
  P0 = inst;               // 寫入指令
  check_BF();              // 完成
}
}                         // 函數結束
```

利用這個函數，則 8 位元資料傳輸模式的初始設定簡化，並寫成一個初始設定的函數，而在初始設定的最後，將開啓顯示功能，如下：

```
=====初始設定函數(8 位元傳輸模式)=====
void init_LCM(void)
{ write_inst(0x30);        // 設定功能
  write_inst(0x30);        // 設定功能
  write_inst(0x08);        // 關閉顯示功能
  write_inst(0x01);        // 清除顯示幕
  write_inst(0x06);        // 設定輸入模式
  write_inst(0x0c);        // 開啟顯示功能
}
}                         // 函數結束
```

利用這個函數，則 4 位元資料傳輸模式的初始設定函數，如下：

```
=====初始設定函數(4 位元傳輸模式)=====
void init_LCM(void)
{ write_inst(0x20);        // 設定功能
  write_inst(0x00);        // 設定功能
  write_inst(0x20);        // 設定功能
  write_inst(0x00);        // 設定功能
  write_inst(0x00);        // 關閉顯示功能
  write_inst(0x08);        // 關閉顯示功能
  write_inst(0x00);        // 清除顯示幕
  write_inst(0x01);        // 清除顯示幕
  write_inst(0x00);        // 設定輸入模式
  write_inst(0x06);        // 設定輸入模式
  write_inst(0x00);        // 開啟顯示功能
  write_inst(0x0c);        // 開啟顯示功能
}
}                         // 函數結束
```



● 相容的初始設定

上述的初始設定是依據 ST7920 控制器的 data sheet，主要是針對中文 LCM。我們可將它改成與 HD44780 控制還器相容的初始設定，讓它可同時用於 ST7920 控制器或 HD44780 控制器，如下：

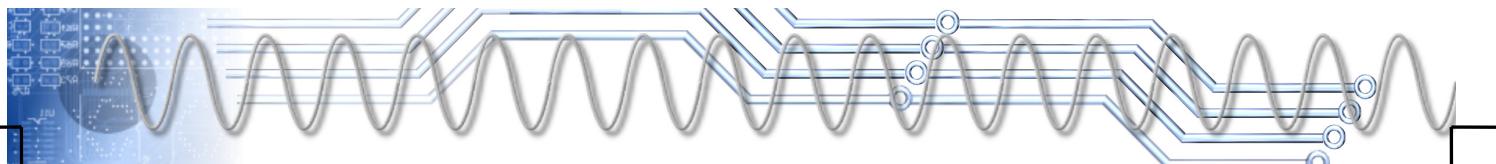
```
//====初始設定函數(8位元傳輸模式)=====
void init_LCM(void)
{ write_inst(0x30);           // 設定功能
  write_inst(0x30);           // 設定功能
  write_inst(0x30);           // 設定功能
  write_inst(0x38);           // 設定兩列、5x7字型(HD44780)
  write_inst(0x08);           // 關閉顯示功能
  write_inst(0x01);           // 清除顯示幕
  write_inst(0x06);           // 設定輸入模式
  write_inst(0x0c);           // 開啟顯示功能
}
// 函數結束
```

4 位元資料傳輸模式的初始設定函數，如下：

```
//====初始設定函數(4位元傳輸模式)=====
void init_LCM(void)
{ write_inst(0x20);           // 設定功能
  write_inst(0x00);           // 設定功能
  write_inst(0x20);           // 設定功能
  write_inst(0x00);           // 設定功能
  write_inst(0x20);           // 設定功能
  write_inst(0x00);           // 設定功能
  write_inst(0x20);           // 設定功能
  write_inst(0x80);           // 設定功能
  write_inst(0x00);           // 關閉顯示功能
  write_inst(0x08);           // 關閉顯示功能
  write_inst(0x00);           // 清除顯示幕
  write_inst(0x01);           // 清除顯示幕
  write_inst(0x00);           // 設定輸入模式
  write_inst(0x06);           // 設定輸入模式
  write_inst(0x00);           // 開啟顯示功能
  write_inst(0x0c);           // 開啟顯示功能
}
// 函數結束
```

● 檢查忙碌函數

溝通是雙方面的事，當 8x51 對 LCM 下指令或丟資料，LCM 不見得



有空處理，所以在前面的 write_inst 函數或 write_data 函數的最後，都會利用「check_BF();」指令，敲敲 LCM 的門，問它忙不忙？只要讀取 DB7，也就 BF 旗標，若 BF=1 表示 LCM 還沒處理完成，不要再繼續丟資料或指令給它。因此，在此就以一個簡單的檢查忙碌函數，以確認可繼續下一步的操作，如下：

```
=====檢查忙碌函數=====
void check_BF(void)
{ E=0;                                // 禁止讀寫動作
  do{ BF = 1;                            // 設定 BF 為輸入
      RS = 0; RW = 1;                    // 讀取指令
      E = 1;                            // 致能(讀取 BF 及 AC)
      } while(BF==1)                     // 忙碌繼續等
}
// 結束
```

● 寫入資料函數

8x51 對 LCM 之溝通，除了寫入指令外，寫入資料也是很常用的動作，將這項操作，寫成一個函數，其應用就能使程式更簡潔，如下：

```
=====寫入資料函數=====
void write_char(char character)
{ check_BF();                         // 檢查忙碌
  RS = 1; RW=0; E = 1;                 // 寫入資料模式
  P0 = character;                     // 寫入字元
  check_BF();                         // 完成
}
// 函數結束
```

● 寫入指定位置函數

以 2 列每列 16 個字的 LCM 而言，我們可將寫入指令定義的更完善，讓所要顯示的字元能「對號入座」，如下所示，在函數的引數裡，增加 line 與 location 兩個引數，line 就是第幾列，0 代表第一列、1 代表第二列；location 則為第幾個位置，有效的位置為 0 到 15，整個函數如下：

```
=====寫入指定位置函數=====
void display_char(char line,location,character)
{ check_BF();                         // 檢查忙碌
  RS = 0; RW= 0; E = 1;                // 寫入指令模式
```



```

if (line=0)
    P0 = 0x80+location;           // 寫入第一列
if (line=1)
    P0 = 0x90+location;           // 寫入第二列
    check_BF();                  // 完成
    write_char(character);        // 寫入字元
}
}                                     // 函數結束

```

● 整列空白函數

若要將整列填入空白，可使用下列函數，其中的 line 引數為所要填入空白的列，0 代表第一列、1 代表第二列，整個函數如下：

```

===== 整列空白函數 =====
void blank_line(char line)
{ check_BF();                      // 檢查忙碌
  RS = 0; RW= 0; E = 1;             // 寫入指令模式
  if (line=0)
    P0 = 0x80;                     // 寫入第一列
  If (line=1)
    P0 = 0x90;                     // 寫入第二列
    check_BF();                    // 完成
    for (char i=0;i<16;i++)
      write_char(' ');
}
}                                     // 函數結束

```

13-5

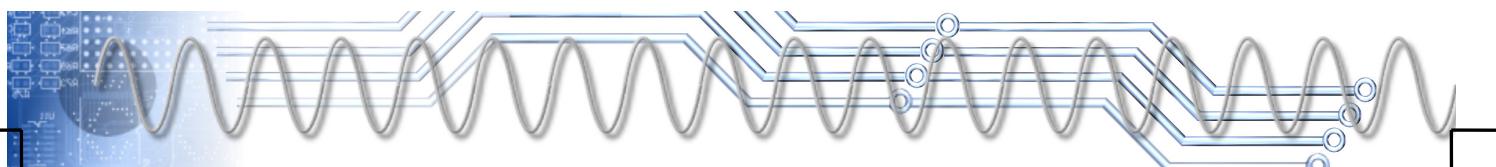
LCM 與 8x51 之連接

LCD 之應用



如圖 11 所示，LCM 的資料匯流排 D0-D7，可與 8x51 的 Port 1、Port 2 或 Port 3 連接。若要與 8x51 之 Port 0 連接，其中每一條線必須各連接一個提升電阻器。而 LCM 的三條控制線 E、R/W 與 RS，則可連接到其它沒被用到的輸出入埠。在 89S51 燒錄實驗板(USB 版)裡，已從 Port 0 連接到 LCM 插座(JP2)的資料匯流排(包含提升電阻器)，且由 P3.0、P3.1 及 P3.2 連接至 LCM 插座的 E、R/W 與 RS，我們只要將 LCM 插到此插座即可。

另外，LCM 的電源接腳 VCC 連接 +5V、VSS 接腳接地(LCM 插座上已連接)；明亮度控制接腳 Vo 可藉由跳線插座(JP4)，選擇連接 VCC 或 GND，若使用中文 LCM，則跳接至 VCC、英文 LCM，則跳接至 GND。



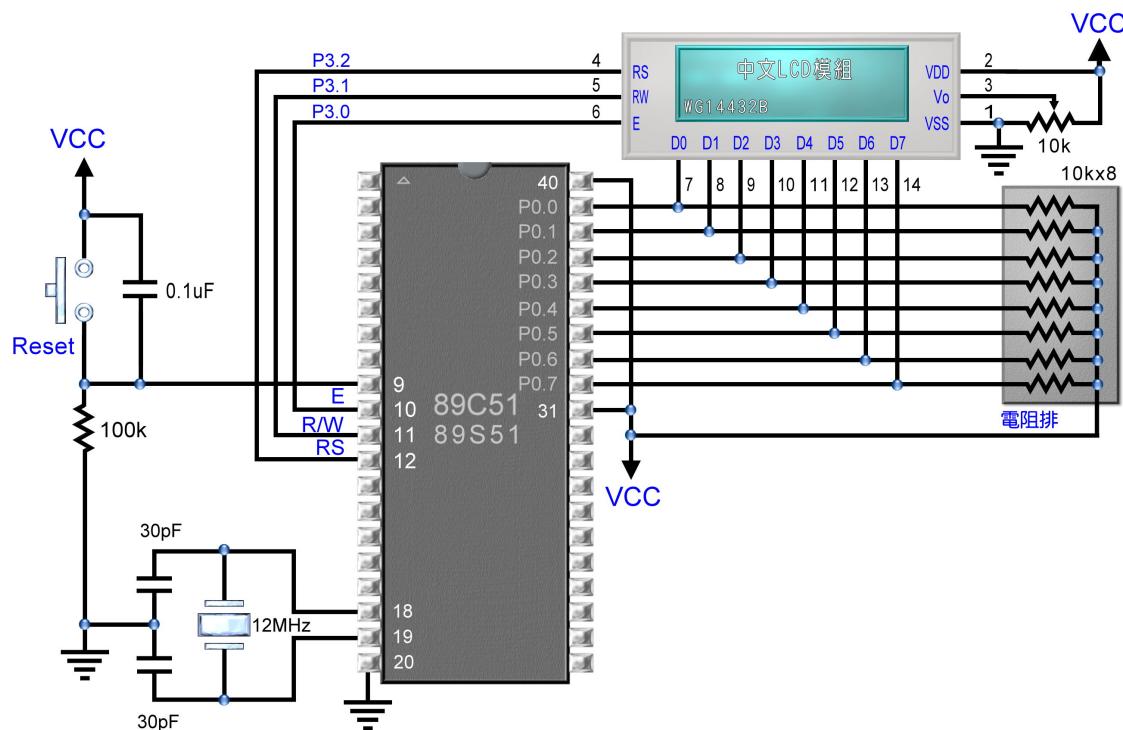


圖11 LCM 與 8x51 之連接

13-6

實例演練

LCD 之應用



在本單元裡提供兩個範例，如下所示：

13-6-1

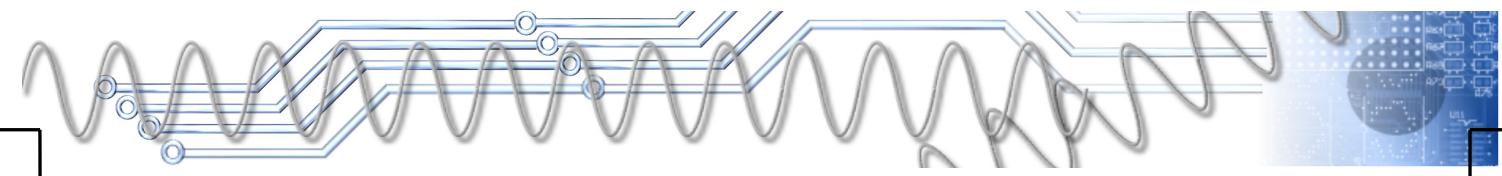
LCD 文字顯示實例演練

實驗要點

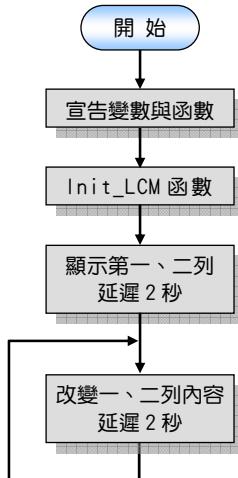
如圖 11 所示，在本單元裡將進行簡單的 LCD 文字顯示的實驗，首先在第一列裡顯示「**LCM test program**」，2 秒後在第二列裡顯示「**Everything is OK**」；再經 2 秒後在第一列裡改為「**中文 LCM 測試程式**」，第二列裡改為「**一切正常歡迎使用**」...，如圖 12 所示。



圖12 功能示意圖



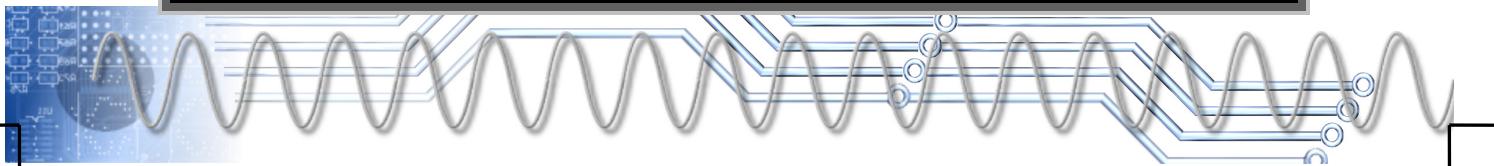
流程圖與程式設計



```

/*LCD 文字顯示實驗(ch13-6-1.c)適用於 89S51 線上燒錄實驗板(USB 版)*/
#include <reg51.h>
#define LCDP P0           // 定義 LCM 資料匯流排接至 P0
sbit RS = P3^2;          // 暫存器選擇位元(0:指令,1:資料)
sbit RW= P3^1;           // 設定讀寫位元 (0:寫入,1:讀取)
sbit E = P3^0;           // 致能位元 (0:禁能,1:致能)
sbit BF = P0^7;          // 忙碌檢查位元(0:不忙,1:忙碌)
char line1[]="LCM test program";// 第 1 次顯示字串(第 1 行)
char line2[]="Everything is OK";// 第 1 次顯示字串(第 2 行)
char line3[]="中文 LCM 測試程式";// 第 2 次顯示字串(第 1 行)
char line4[]="一切正常歡迎使用";// 第 2 次顯示字串(第 2 行)
void init_LCM(void);      // 初始設定函數
void write_inst(char);    // 寫入指令函數
void write_char(char);    // 寫入字元資料函數
void check_BF(void);     // 檢查忙碌函數
void delay1ms(int);      // 延遲函數
// ====== 主程式 ======
main()
{
    char i;                // 告變數
    init_LCM();             // 初始設定
    while(1)                // 無盡迴圈
    //=====LCM test program=====
    {
        write_inst(0x80);    // 指定第一列位置
        for (i=0;i<16;i++)    // 迴圈
            write_char(line1[i]); // 顯示 16 個字
    //=====Everything is OK=====
        write_inst(0x90);    // 指定第二列位置
        for (i=0;i<16;i++)    // 迴圈
            write_char(line2[i]); // 顯示 16 個字
}

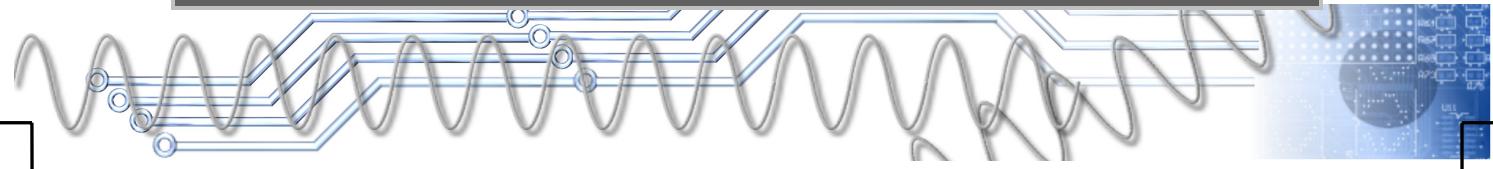
```



```

delay1ms(2000);           // 延遲 2 秒
//===== 中文 LCM 測試程式 ======
write_inst(0x80);         // 指定第一列位置
for (i=0;i<16;i++)        // 迴圈
    write_char(line3[i]); // 顯示 16 個字
//===== 一切正常歡迎使用 ======
write_inst(0x90);         // 指定第二列位置
for (i=0;i<16;i++)        // 迴圈
    write_char(line4[i]); // 顯示 16 個字
delay1ms(2000);           // 延遲 2 秒
}
}
// while 結束
// 主程式 main()結束
//=====初始設定函數(8 位元傳輸模式)=====
void init_LCM(void)
{
    write_inst(0x30);          // 設定功能-8 位元-基本指令
    write_inst(0x30);          // 設定功能-8 位元-基本指令
    write_inst(0x30);          // 英文 LCM 相容設定，中交 LCM 可忽略
    write_inst(0x38);          // 英文 LCM 設定兩列，中交 LCM 可忽略
    write_inst(0x08);          // 顯示功能-關顯示幕-無游標-游標不閃
    write_inst(0x01);          // 清除顯示幕(填 0x20,I/D=1)
    write_inst(0x06);          // 輸入模式-位址遞增-關顯示幕
    write_inst(0x0c);          // 顯示功能-開顯示幕-無游標-游標不閃
} // init_LCM()函數結束
//===== 寫入指令函數 ======
void write_inst(char inst)
{
    check_BF();               // 檢查是否忙碌
    LCDP = inst;              // LCM 讀入 MPU 指令
    RS = 0; RW = 0; E = 1;      // 寫入指令至 LCM
    check_BF();               // 檢查是否忙碌
} // write_inst()函數結束
//===== 寫入字元資料函數 ======
void write_char(char chardata)
{
    check_BF();               // 檢查是否忙碌
    LCDP = chardata;          // LCM 讀入字元
    RS = 1; RW = 0 ;E = 1;     // 寫入資料至 LCM
    check_BF();               // 檢查是否忙碌
} // write_char()函數結束
//=====檢查忙碌函數=====
void check_BF(void)
{
    E=0;                      // 禁止讀寫動作
    do                        // do-while 迴圈開始
    {
        BF=1;                  // 設定 BF 為輸入
        RS = 0; RW = 1;E = 1;   // 讀取 BF 及 AC
    }while(BF == 1);          // 忙碌繼續等
}

```



```

}
// check_BF()函數結束
//===== 延遲函數 =====
void delay1ms(int x)
{
    int i,j;           //宣告變數
    for (i=1;i<x;i++) //執行 x 次,延遲 X*1ms
        for (j=1;j<120;j++); //執行 120 次,延遲 1ms
}
// delay1ms()函數結束

```

LCD 文字顯示實驗(ch13-6-1.c)

操作

- 依功能需求與電路結構，在 Keil C 裡撰寫程式，並進行建構(按  鈕)，以產生*.HEX 檔。然後進行軟體除錯/模擬，看看其功能是否正常？若有錯誤或非預期的狀況，則檢視原始程式，看看哪裡出問題？修改之，並將它記錄在實驗報告裡。
- 若軟體除錯/模擬功能正常，可按圖 11 連接線路，並使用實體模擬器，載入新的程式(*.HEX)，以模擬該電路的動作。若有非預期的狀況，則檢視線路的連接狀況，看看哪裡出問題？並將它記錄在實驗報告裡。
- 若實體模擬功能正常，請將程式燒錄到 89S51(可使用 89S51 線上燒錄實驗板)，再把該 89S51 放入實體電路，以取代剛才的實體模擬器，然後直接送電，看看是否正常？
- 撰寫實驗報告。

13-6-2

自編字型圖案實例演練

實驗要點

如圖 11 所示為本實驗所要採用的電路圖，在此我們將自建兩個字型，分別是「」及「」，其中的「」代表上午、「」代表下午，在 LCM 裡，第一列將顯示以「時:分:秒」的格式顯示時間，每秒鐘改變一次顯示，在其右邊將以自建字型「」、「」區別上下午，如圖 13 所示：



圖13 功能示意圖



流程圖與程式設計

依功能需求，在此分成三部分來說明，第一部分是自建字型，第二部分是時間的產生，第三部分是將計時數轉換成顯示資料。

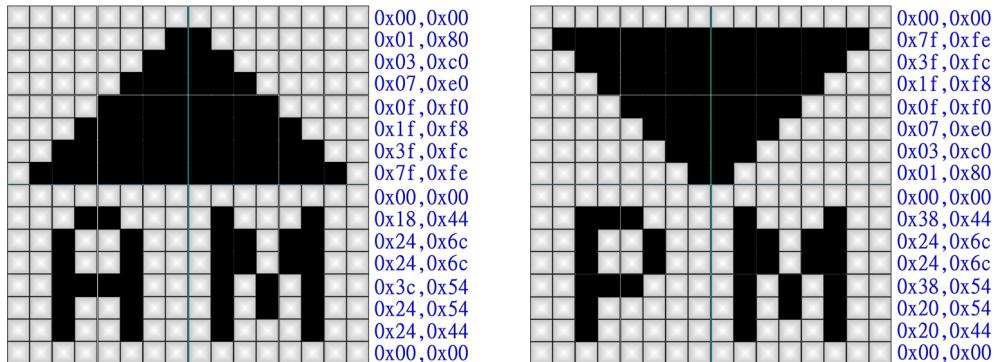


圖14 自建字型

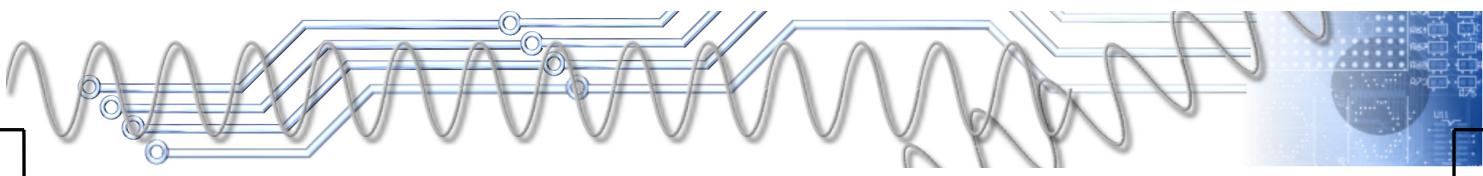
- 在自建字型方面，如圖 14 所示為本次所要建的字型，其中的「**AM**」編碼為「0x00, 0x00, 0x01, 0x80, 0x03, 0xc0, 0x07, 0xe0, 0x0f, 0xf0, 0x1f, 0xf8, 0x3f, 0xfc, 0x7f, 0xfe, 0x00, 0x00, 0x18, 0x44, 0x24, 0x6c, 0x24, 0x6c, 0x3c, 0x54, 0x24, 0x54, 0x24, 0x44, 0x00, 0x00」；而「**PM**」編碼為「0x00, 0x00, 0x7f, 0xfe, 0x3f, 0xfc, 0x1f, 0xf8, 0x0f, 0xf0, 0x07, 0xe0, 0x03, 0xc0, 0x01, 0x80, 0x00, 0x00, 0x38, 0x44, 0x24, 0x6c, 0x24, 0x6c, 0x38, 0x54, 0x20, 0x54, 0x20, 0x44, 0x00, 0x00」，在此將它們存入 pat[16]陣列，如下所示：

```
char code am[32] = {           // 顯示上三角及 AM
    0x00, 0x00, 0x01, 0x80, 0x03, 0xc0, 0x07, 0xe0,
    0x0f, 0xf0, 0x1f, 0xf8, 0x3f, 0xfc, 0x7f, 0xfe,
    0x00, 0x00, 0x18, 0x44, 0x24, 0x6c, 0x24, 0x6c,
    0x3c, 0x54, 0x24, 0x54, 0x24, 0x44, 0x00, 0x00};

char code pm[32] = {           // 顯示下三角及 PM
    0x00, 0x00, 0x7f, 0xfe, 0x3f, 0xfc, 0x1f, 0xf8,
    0x0f, 0xf0, 0x07, 0xe0, 0x03, 0xc0, 0x01, 0x80,
    0x00, 0x00, 0x38, 0x44, 0x24, 0x6c, 0x24, 0x6c,
    0x38, 0x54, 0x20, 0x54, 0x20, 0x44, 0x00, 0x00};
```

CG RAM 的起始位址為 01000000B，即 0x40，每 8 個位址為一個自建字型。在程式裡，可以下列指令將這個陣列的內容填入 CG RAM：

```
char i;
write_inst(0x40);           // 設定 CGRAM 的位置
for (i=0;i<32;i++)
```



```

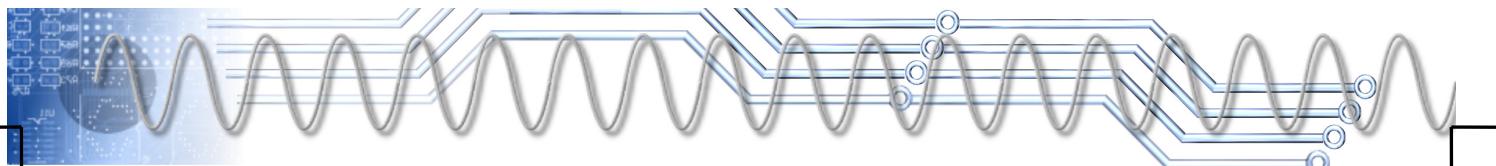
        write_char(am[i]); // 寫入上午之自建字型
for (i=0;i<32;i++)
        write_char(pm[i]); // 寫入下午之自建字型
    
```

- 時間的產生是利用 TIMER0 中斷，每次中斷為 50ms，每 20 次中斷就是 1 秒鐘，因此要調整時間的輸出。在此，hour 變數為「時」數、minute 變數為「分」數、second 變數為「秒」數。若秒數超過 60 秒，則秒數歸零，進而調整分數(即分+1)；若分數超過 60 分，則分數歸零，進而調整時數(即時+1)；若時數超過 12 時，則時數調整為 1，進而改變上/下午狀態，如下：

```

void clock(void) interrupt 1 // T0 中斷副程式
{
    TH0=(65636-50000)/256; // 填入計時量
    TL0=(65636-50000)%256; // 填入計時量
    if (--count==0) // 中斷次數是否達到 20 次
    {
        count=20; // 重新計次
        second++; // 秒數加 1
        if (second>=60) // 是否達到 60 秒
        {
            second=0; // 秒數歸零
            minute++; // 分數加 1
            if (minute>=60) // 是否達到 60 分
            {
                minute=0; // 分數歸零
                hour++; // 時數加 1
                if (hour == 13)// 是否達到 13 小時
                    hour=1; // 時數改為 1
                if (hour == 12)// 是否達到 12 小時
                    ampm=~ampm;// 切換上下午
            }
        }
    }
} } // 結束
    
```

- 緊接著是將 hour、minute、second、ampm 轉換成在 LCM 顯示的 ASCII 碼及自建的圖案。hour、minute、second 變數是以 16 進位數字儲存時數、分數與秒數，我們可利用「/10」萃取其十位數、「%10」萃取其個位數；再加上 0x30，則數字就變成 ASCII，然後把它存入顯示區陣列 d[]，如下：



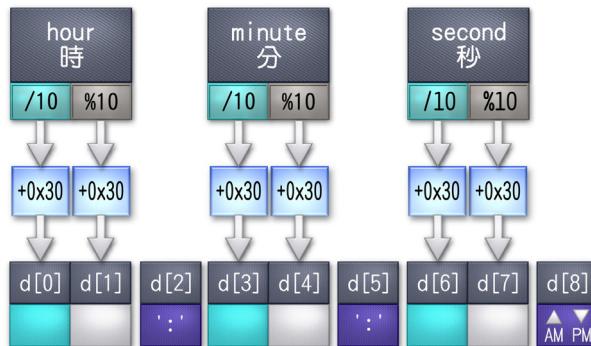


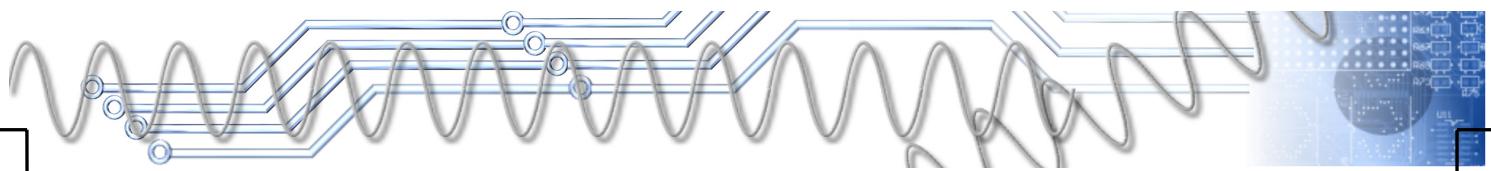
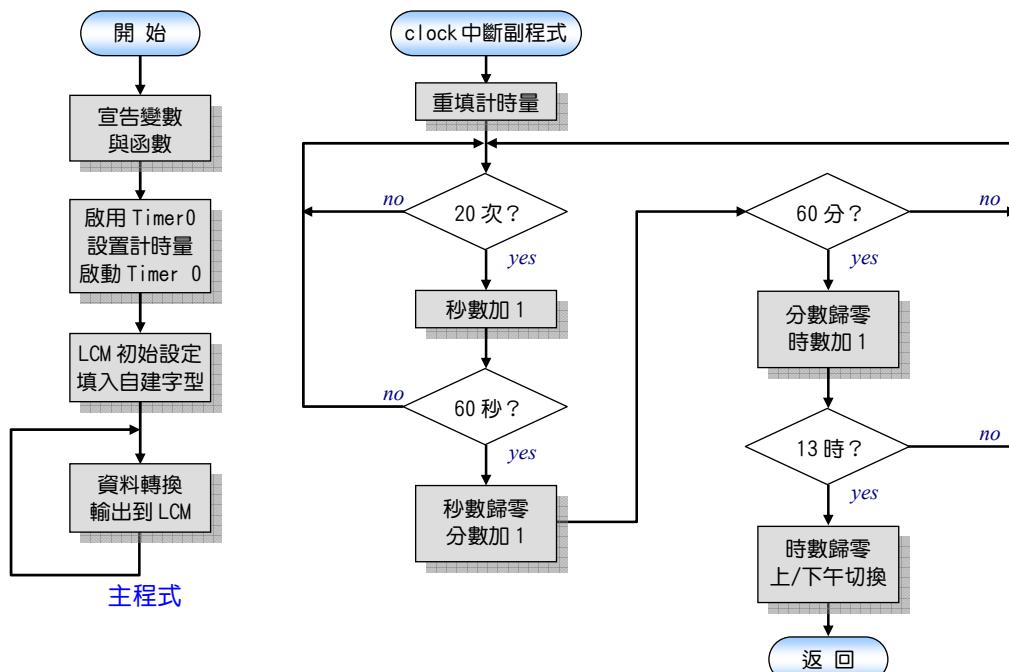
圖 15 數字之時分秒轉換成 ASCII 碼

```

d[0]=hour/10+0x30;           // 時數之十位數顯示資料
d[1]=hour%10+0x30;          // 時數之個位數顯示資料
d[2]=':';
d[3]=minute/10+0x30;        // 分數之十位數顯示資料
d[4]=minute%10+0x30;        // 分數之個位數顯示資料
d[5]=':';
d[6]=second/10+0x30;        // 秒數之十位數顯示資料
d[7]=second%10+0x30;        // 秒數之個位數顯示資料
if (ampm==0)    d[9]=0x00;   // 上午
else d[9]=0x02;             // 下午

```

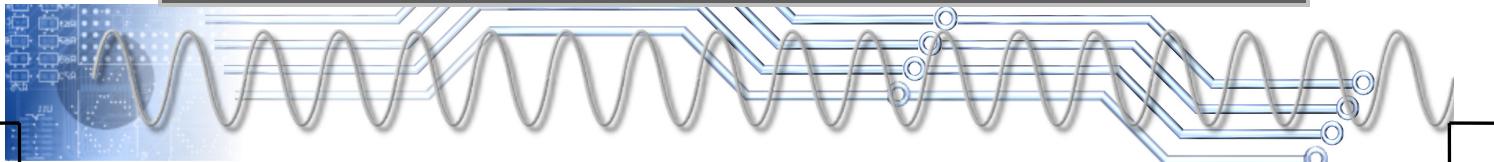
流程圖與整個程式如下所示：



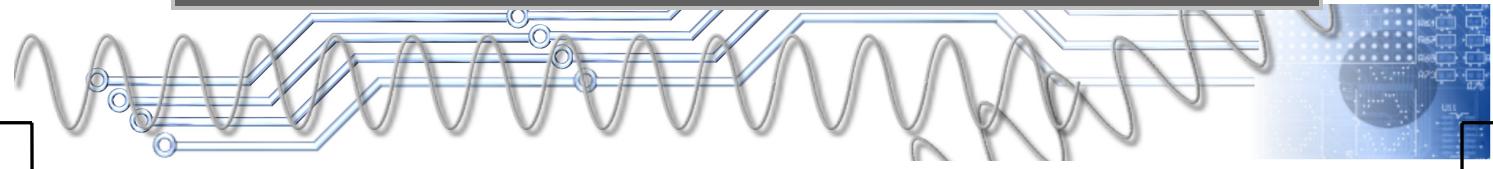
```

/*自編字型圖案實驗(ch13-6-2.c)適用於 89S51 線上燒錄實驗板(USB 版)*/
#include <reg51.h>
#define LCDP P0           // 定義 LCM 資料匯流排接至 P0
sbit RS = P3^2;          // 暫存器選擇位元(0:指令,1:資料)
sbit RW = P3^1;          // 設定讀寫位元 (0:寫入,1:讀取)
sbit E = P3^0;           // 致能位元 (0:禁能,1:致能)
sbit BF = P0^7;          // 忙碌檢查位元(0:不忙,1:忙碌)
char count=20;           // 中斷次數計數，20 次*50ms=1 秒
char time[10];           // 顯示時間陣列(第 1 行)
/* 宣告自建字型陣列變數 */
char code am[32] = {      // 顯示上三角及 AM
0x00, 0x00, 0x01, 0x80, 0x03, 0xC0, 0x07, 0xE0,
0xF, 0xF0, 0x1F, 0xF8, 0x3F, 0xFC, 0x7F, 0xFE,
0x00, 0x00, 0x18, 0x44, 0x24, 0x6C, 0x24, 0x6C,
0x3C, 0x54, 0x24, 0x54, 0x24, 0x44, 0x00, 0x00};
char code pm[32] = {      // 顯示下三角及 PM
0x00, 0x00, 0x7F, 0xFE, 0x3F, 0xFC, 0x1F, 0xF8,
0xF, 0xF0, 0x07, 0xE0, 0x03, 0xC0, 0x01, 0x80,
0x00, 0x00, 0x38, 0x44, 0x24, 0x6C, 0x24, 0x6C,
0x38, 0x54, 0x20, 0x54, 0x20, 0x44, 0x00, 0x00};
bit ampm=1;               // 0:上午(am),1:下午(pm),初值下午
char hour=11;              // 宣告時,初值為 11 點
char minute=59;             // 宣告分,初值為 59 分
char second=50;             // 宣告秒,初值為 50 秒
void transfer(void);        // 轉換時分秒至 time 陣列中
void write_inst(char);       // 寫入指令函數
void write_char(char);       // 寫入字元函數
void write_pat(void);        // 寫入自建字型函數
void check_BF(void);         // 檢查忙碌函數
void init_LCM(void);         // 宣告 LCM 初始設定函數
=====主程式=====
main()
{ char i;
  init_LCM();                // 初始設定
  write_pat();                // 寫入自建字型
  IE=0X82;                   // Timer 0 中斷致能
  TMOD=0x01;                  // T0 設為 MODE1
  TH0=(65636-50000) / 256;    // 填入計時量之高位元組
  TL0=(65636-50000) % 256;    // 填入計時量之低位元組
  TR0=1;                      // 啟動 Timer 0
  while(1)                    // 無窮迴圈
  {
    transfer();                // 轉換時分秒至 time 陣列中
    write_inst(0x80);           // 指定第 1 列位置
    for (i=0;i<10;i++)          // 迴圈

```



```
        write_char(time[i]);      // 顯示時間
    } // while 結束
} // main() 結束
//====轉換函數=====
void transfer(void)
{ time[0]= hour/10 + 0x30;          // 時數之十位數顯示資料
  time[1]= hour%10 + 0x30;          // 時數之個位數顯示資料
  time[2]= ':';                   // 顯示冒號
  time[3]= minute/10 + 0x30;       // 分數之十位數顯示資料
  time[4]= minute%10 + 0x30;       // 分數之個位數顯示資料
  time[5]= ':';                   // 顯示冒號
  time[6]= second/10 + 0x30;       // 秒數之十位數顯示資料
  time[7]= second%10 + 0x30;       // 秒數之個位數顯示資料
  time[8]=0x00;                   // 自鍵字型之高位元組
  if (ampm==0)                    // 判定是否為上午
    time[9]=0x00;                 // 表示上午之自鍵字型
  else time[9]=0x02;              // 表示下午之自鍵字型
}
//====寫入自建字型函數=====
void write_pat(void)
{ char i;
  write_inst(0x40);               // 設定 CGRAM 的位置
  for (i=0;i<32;i++)
    write_char(am[i]);           // 寫入上午之自鍵字型
  for (i=0;i<32;i++)
    write_char(pm[i]);           // 寫入下午之自鍵字型
}
//==== Timer 0 中斷副程式 =====
void clock(void) interrupt 1      // T0 中斷副程式
{ TH0=(65636-50000)/256;         // 填入計時量
  TL0=(65636-50000)%256;         // 填入計時量
  if (--count==0)                 // 中斷次數是否達到 20 次
  { count=20;                     // 重新計次
    second++;                     // 秒數加 1
    if (second>=60)              // 是否達到 60 秒
    { second=0;                   // 秒數歸零
      minute++;                  // 分數加 1
      if (minute>=60)            // 是否達到 60 分
      { minute=0;                // 分數歸零
        hour++;                  // 時數加 1
        if (hour == 13)           // 是否達到 13 小時
          hour=1;                 // 時數改為 1
        if (hour == 12)           // 是否達到 12 小時
          ampm=~ampm;             // 切換上下午
    }
  }
}
```



```

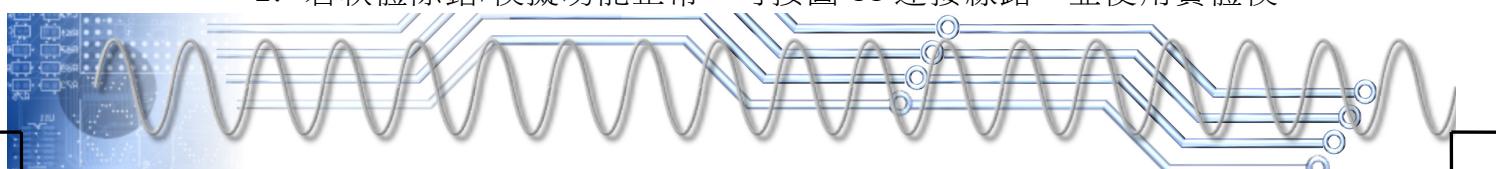
    } } } } // 結束
//=====初始設定函數(8 位元傳輸模式)=====
void init_LCM(void)
{ write_inst(0x30); // 設定功能-8 位元-基本指令
  write_inst(0x30); // 設定功能-8 位元-基本指令
  write_inst(0x30); // 英文 LCM 相容設定，中交 LCM 可忽略
  write_inst(0x38); // 英文 LCM 設定兩列，中交 LCM 可忽略
  write_inst(0x08); // 顯示功能-關顯示幕-無游標-游標不閃
  write_inst(0x01); // 清除顯示幕(填 0x20,I/D=1)
  write_inst(0x06); // 輸入模式-位址遞增-關顯示幕
  write_inst(0x0c); // 顯示功能-開顯示幕-無游標-游標不閃
}
} // init_LCM()函數結束
//===== 寫入指令函數 =====
void write_inst(char inst)
{ check_BF(); // 檢查是否忙碌
  LCDP = inst; // LCM 讀入 MPU 指令
  RS = 0; RW = 0; E = 1; // 寫入指令至 LCM
  check_BF(); // 檢查是否忙碌
}
} // write_inst()函數結束
//===== 寫入字元資料函數 =====
void write_char(char chardata)
{ check_BF(); // 檢查是否忙碌
  LCDP = chardata; // LCM 讀入字元
  RS = 1; RW = 0 ;E = 1; // 寫入資料至 LCM
  check_BF(); // 檢查是否忙碌
}
} // write_char()函數結束
//=====檢查忙碌函數=====
void check_BF(void)
{ E=0; // 禁止讀寫動作
  do // do-while 迴圈開始
  {
    BF=1; // 設定 BF 為輸入
    RS = 0; RW = 1;E = 1; // 讀取 BF 及 AC
  }while(BF == 1); // 忙碌繼續等
}
} // check_BF()函數結束

```

自編字型圖案實驗(ch13-6-2.c)

操作

- 依功能需求與電路結構，在 Keil C 裡撰寫程式，並進行建構(按  鈕)，以產生*.HEX 檔。然後進行軟體除錯/模擬，看看其功能是否正常？若有錯誤或非預期的狀況，則檢視原始程式，看看哪裡出問題？修改之，並將它記錄在實驗報告裡。
- 若軟體除錯/模擬功能正常，可按圖 11 連接線路，並使用實體模



擬器，載入新的程式(*.HEX)，以模擬該電路的動作。若有非預期的狀況，則檢視線路的連接狀況，看看哪裡出問題？並將它記錄在實驗報告裡。若實體模擬功能正常，即可將程式燒錄到89S51，再把該89S51放入實際電路，以取代剛才的實體模擬器，然後直接送電，看看是否正常？

3. 若實體模擬功能正常，請將程式燒錄到89S51(可使用89S51線上燒錄實驗板)，再把該89S51放入實體電路，以取代剛才的實體模擬器，然後直接送電，看看是否正常？
4. 撰寫實驗報告。

思考一下

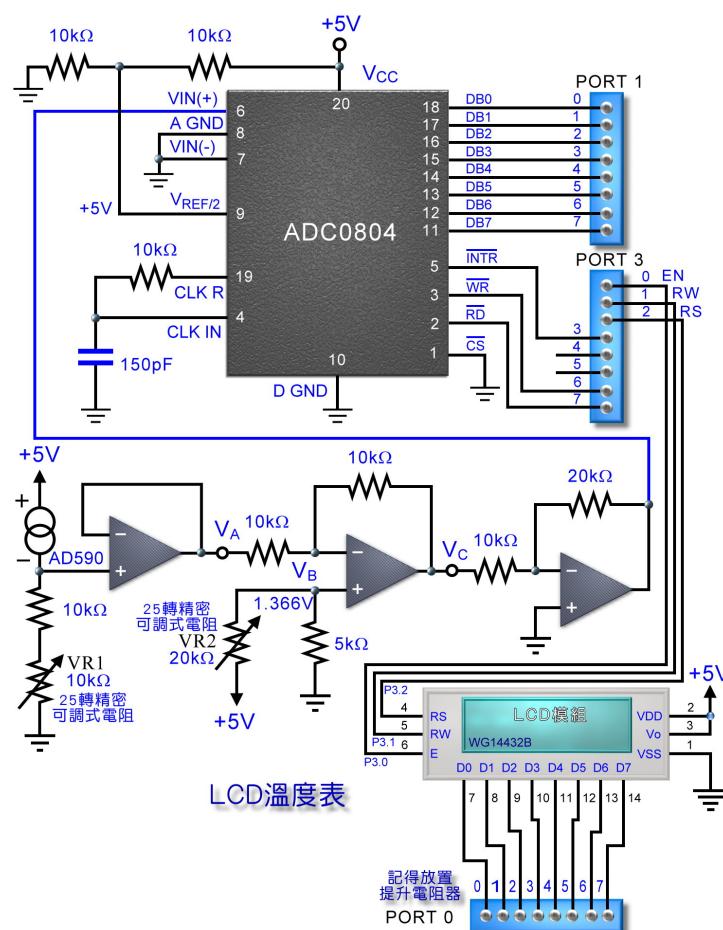
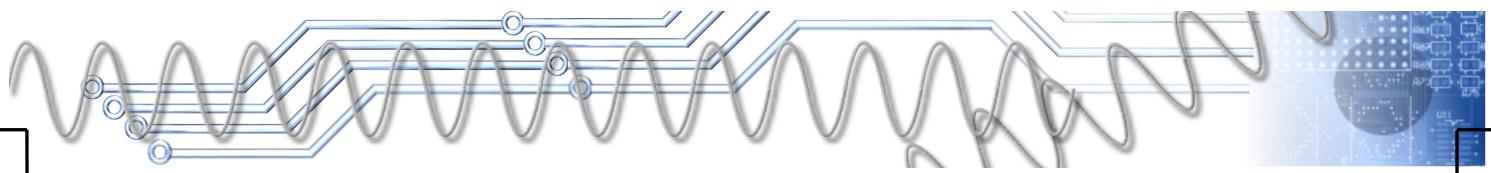


圖16 LCD 數位溫度表

1. 如何結合本實驗裡的LCM及第11章所介紹的ADC，製作一個LCD顯示的數位溫度表，電路圖可參考圖16。其中在LCD顯示



幕裡的「°C」字型，請自行編製，如下圖所示：

13-7

即時練習



LCD 在應用

在本章裡探討 LCM 的結構、指令與應用方法。在此請試著回答下列問題，以確認對於此部分的認識程度。



- () 1. 若要 LCM 顯示某些字元，則需把所要顯示的字元，放入何處？
(A) CG RAM (B) DDRAM (C) IRAM (D) GDRAM 。

() 2. 若要讀取 LCM 的狀態，則應如何設定？ (A) RS=0、R/ \overline{W} =0
(B) RS=1、R/ \overline{W} =0 (C) RS=1、R/ \overline{W} =1 (D) RS=0、R/ \overline{W} =1。

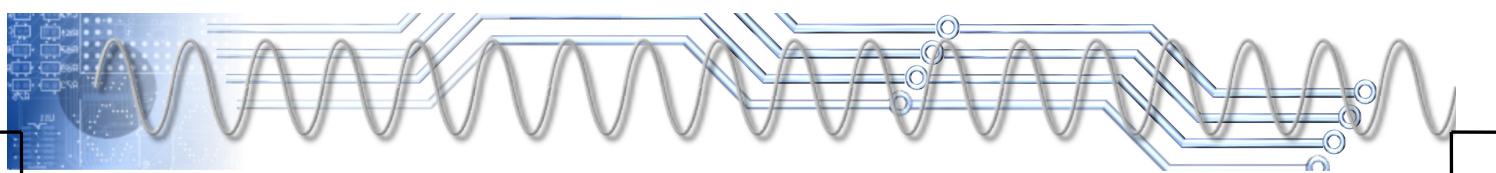
() 3. 若要對 LCM 下指令，則應如何設定？ (A) RS=0、R/ \overline{W} =0
(B) RS=1、R/ \overline{W} =0 (C) RS=1、R/ \overline{W} =1 (D) RS=0、R/ \overline{W} =1。

() 4. 若要將資料寫入 LCM，則應如何設定？ (A) RS=0、R/ \overline{W} =0
(B) RS=1、R/ \overline{W} =0 (C) RS=1、R/ \overline{W} =1 (D) RS=0、R/ \overline{W} =1。

() 5. 若要檢查 LCM 是否忙碌，則應如何設定？ (A) RS=0、R/ \overline{W} =0
(B) RS=1、R/ \overline{W} =0 (C) RS=1、R/ \overline{W} =1 (D) RS=0、R/ \overline{W} =1。

() 6. 若 LCM 更明亮，應如何處理？ (A) Vo 接腳調往高電壓 (B) Vo
接腳調往低電壓 (C) 加大電源電壓 (D) 降低電源電壓 。

() 7. 若對 LCM 操作，應對 EN 接腳做何操作？ (A) 送入一個正脈波



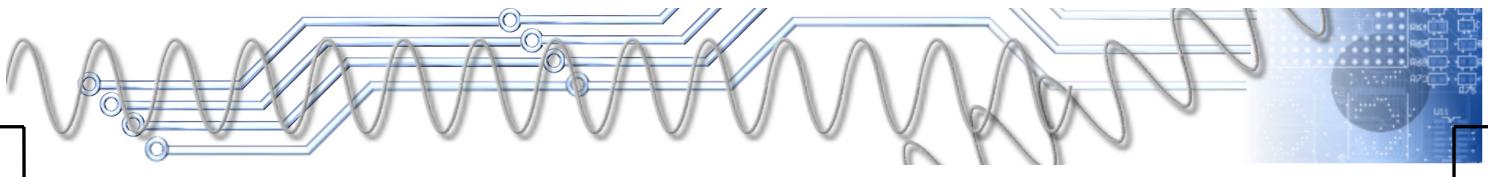
- (B) 送入一個負脈波 (C) EN 接腳接地即可 (D) EN 接腳不影響。
- ()8. 中文 LCM 的中文字型放置在哪裡？ (A) CG ROM (B) HCG ROM (C) DDRAM (D) GD RAM。
- ()9. 中文 LCM-WG14432J-NGG-N 的面板為？ (A) 彩色 LCD 面板 (B) 144×32 LCD 面板 (C) 128×64 LCD 面板 (D) 144×64 LCD 面板。
- ()10. 中文 LCM-WG14432J-NGG-N 採用哪個控制器？ (A) HD44780 (B) ST7920 (C) WG12864 (D) 以上皆非。



問答題

1. 常用的以 HD44780 控制器所組成的 LCM，有哪幾種顯示模式？
2. 請寫出常用 LCM 的接腳？其中調整明亮度的是哪一隻接腳？
3. 常用 LCM 共有 14 隻接腳，有哪兩種包裝？
4. 試述 LCM 初始化的步驟？
5. 若要對 LCM 下指令，必須等它有空，如何偵測 LCM 是否有空？
6. 常用的 LCM 擁有多少 CG RAM？多少 CG ROM？可自建多少個字型？
7. 當我們建好字型後，如何送入 LCM？
8. 常用的中文 LCM-WG14432J-YYH-N，使用哪個控制器？
9. 試簡述中文 LCM 裡，DD RAM、CG RAM、CG ROM、HCG ROM、IRAM、GD RAM 之功能？

加油



輕鬆學習

