Chap 8
串列傳輸

使用的 I/O 裝置

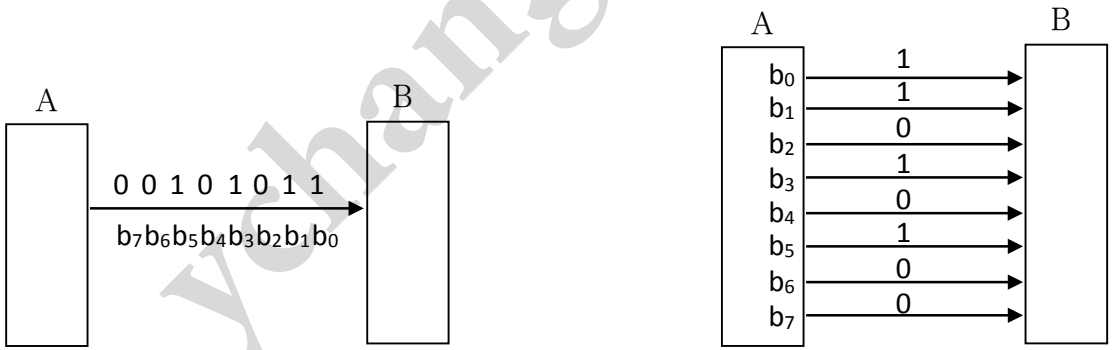
1602 I2C 液晶螢幕

8-1 通訊的分類 1/3

通訊的分類有多種方式，第一種是根據傳輸時資料的排列方式來區分，可以分成串列傳輸與並列（平行）傳輸二種。

(1) 串列傳輸：只需利用一條資料線，將資料按照順序以一位元接著一位元的方式傳送，如圖 9.1.1(a)所示，串列傳輸的特點是線路簡單，成本低廉，適合長距離的資料傳輸，但是傳輸速度較慢。

(2) 並列傳輸：利用多條資料線同時傳送多個位元，如圖 9.1.1(b)所示，特點是速度快，適合短距離的傳輸，但是資料線數量較多，會增加通訊介面的成本。



(a) 串列傳輸　　(b)並列傳輸

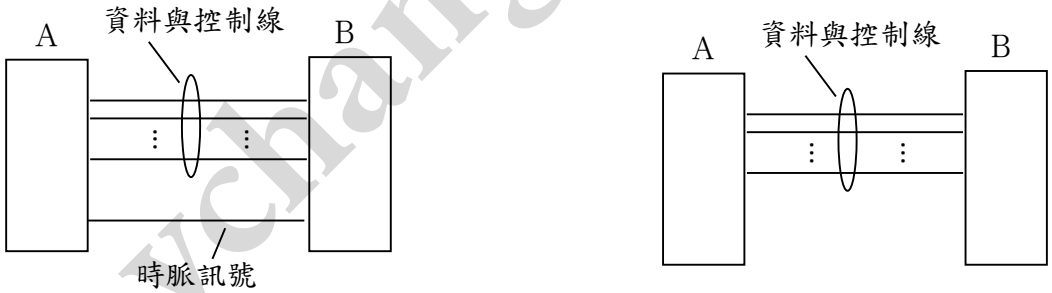
圖 9.1.1 傳輸資料的排列方式

8-1 通訊的分類 2/3

第二種分類方式，根據資料傳輸時是否有使用同步的時脈，可以分成同步傳輸與非同步傳輸。

(1) 同步傳輸：傳送端與接收端除了資料線之外，還必須提供一個額外的時脈訊號線，如圖 9.1.2(a)所示，藉此確保接收端能正確的讀取資料。特點是傳輸速度快，適用於高速短距離的傳輸。

(2) 非同步傳輸：如圖 9.1.2(b)所示，不需要額外的時脈訊號線，但必須在傳輸的資料中加入起始、結束等必要的同步資訊，以達到正確的資料傳輸。特點是線路較簡單，但是傳輸速度較慢，僅適用於低速傳輸。



(a) 同步傳輸　　(b)非同步傳輸

圖 9.1.2 資料傳輸是否有使用同步的時脈

8-1 通訊的分類 3/3

第三種分類方式，根據資料傳輸的方向性，可以分成單工，半雙工，與全雙工三種傳輸方式。

(1) 單工傳輸：如圖 9.1.3(a)所示，只允許單向的資料傳輸，且方向固定不能改變，例如廣播電台將節目訊號傳送到收音機。

(2) 半雙工傳輸：允許雙向的資料傳輸，但是不能同時進行，如圖 9.1.3(b)所示，通常是傳送與接收共用同一條資料線，特點是線路簡單，傳輸效率較差。

(3) 全雙工傳輸：允許同時雙向的資料傳輸，也就是傳送資料的同時也能接收資料，如圖 9.1.3(c)傳送與接收使用不同的資料線，特點是資料傳輸較有效率，但是線路較多。

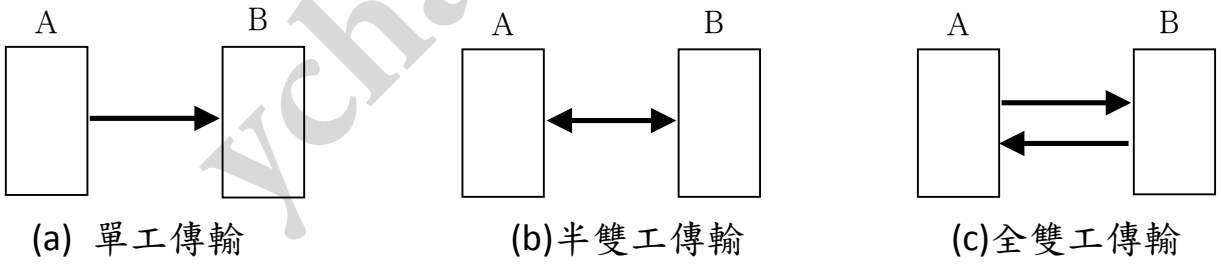


圖 9.1.3 資料傳輸的方向性

8-1 嵌入式系統常見的串列通訊

- 受限微控器有限的接腳數量，絕大多數的嵌入式系統所使用的通訊傳輸方式皆是串列通訊，常見的串列通訊有
- UART（Universal Asynchorous Receiver/Transmitter）：通用非同步收發器
- I^2C, I2C, IIC（Inter IC）：IC積體電路間通訊
- SPI（Serial Peripheral Interface）：串列週邊介面

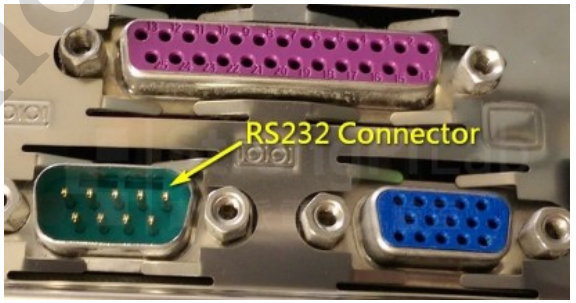
	同步	非同步
半雙工	I2C	
全雙工	SPI	UART

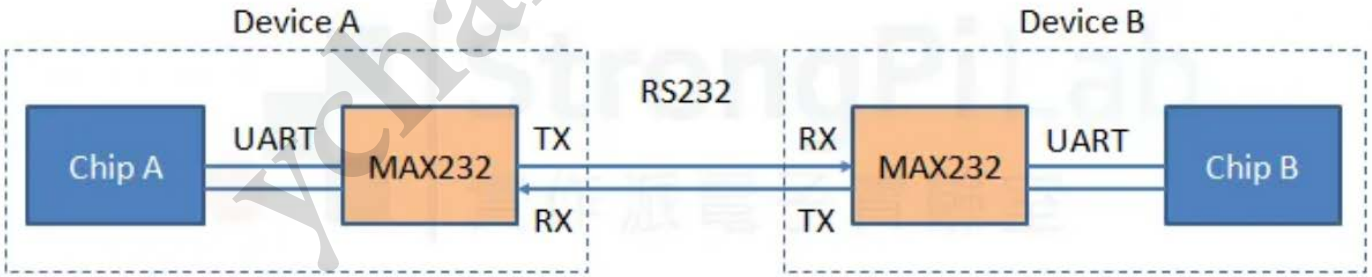
UART
（通用非同步收發器）

8-2 UART串列通訊

- RS232 與 UART 的差別

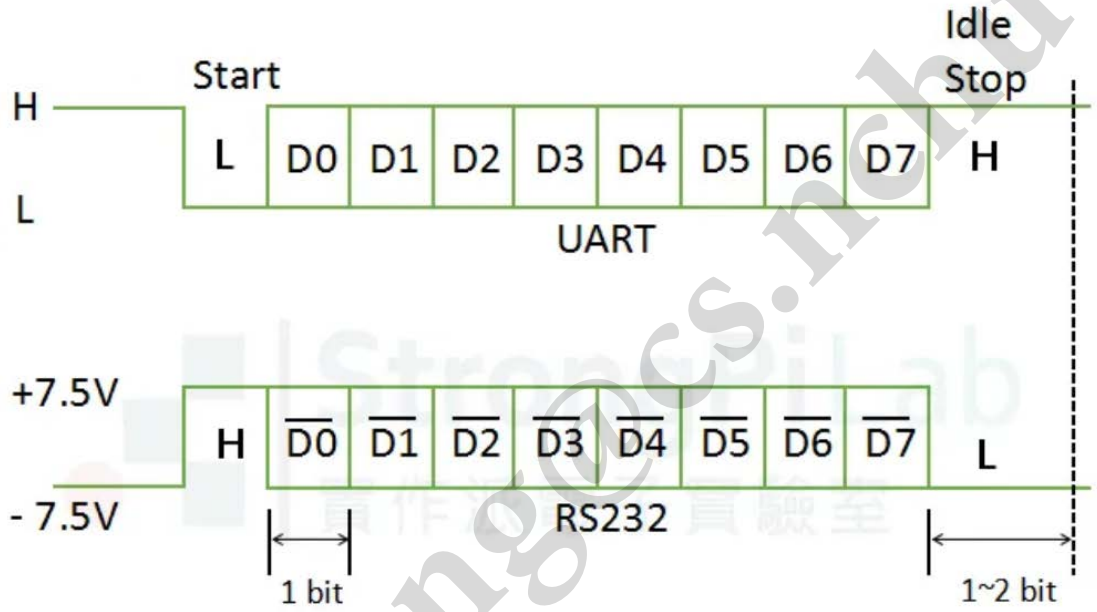
➢ UART是通訊協定裡的Layer2，也就是Data Link Layer，它只定義傳送一個byte時，頭尾應該要有那些0與1資料，因此沒有定義接頭形狀，接頭的形狀與訊號準位則是定義在RS232，它是Layer1或說實體層Physical Layer。因此UART底層的實體要接哪種介面，完全看個人需求而定，你可以直接導線對接也可以接RS232 (或RS-422 與 RS-485)。





UART 與 RS232 的接線方塊圖

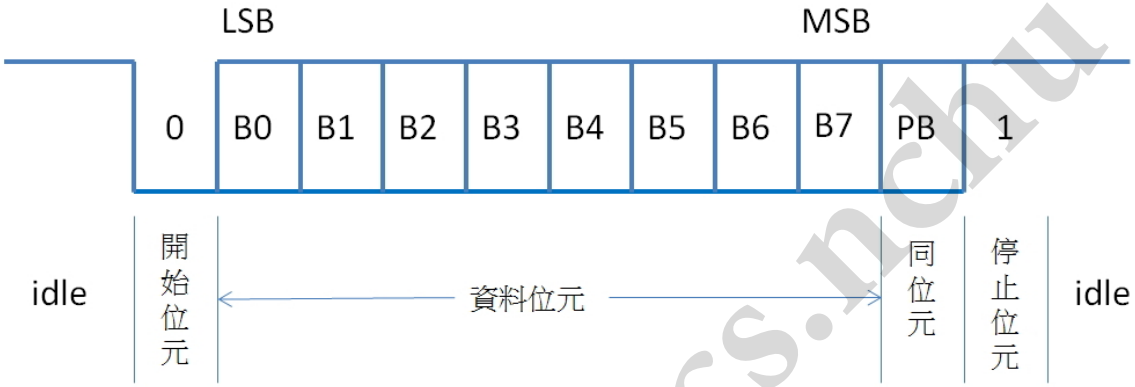
8-2 UART串列通訊



UART 與 RS232 的資料格式

- RS232 的Vpp電壓較高有6V~30V，UART則是較低的3.3V或5V。
- RS232 為負邏輯， UART 則為正邏輯，因此兩者波形是反相的。

8-2 UART串列通訊



Idle

表示Serial port資料送完了，目前沒事做，等待下一筆資料中，UART level固定在H。

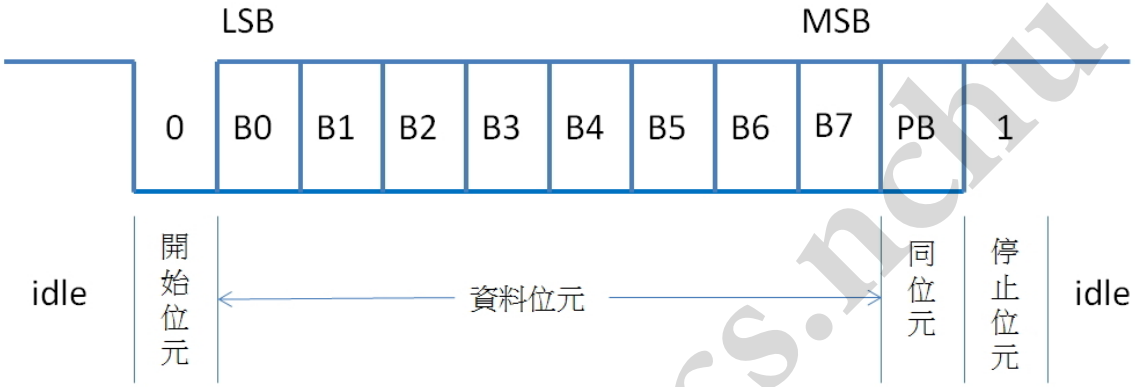
Start

將狀態反相，是送資料前的準備動作，這樣接收端才知道後面有資料要送，UART level固定為L。

D0~D7

送資料的順序是D0先送，D7最後送，所以在示波器顯示的波形是D0:D7，要判讀示波器上的資料前記得先在腦中反序變成D7:D0再來判讀，也別忘了RS232是負邏輯喔! 這個錯誤很多人都掉進去過。I2C就沒這個困擾，它是D7先送，所以波形從左到右可以很順的解讀。

8-2 UART串列通訊



Stop

資料送完後，要變成Stop狀態。Stop bit很妙，我遇過Stop bit太短導致接收端會誤判，最短一個bit，最長其實可以很長，看你高興，通常有1bit、1.5bit、2bit的選項，個人建議隔開一點比較安全，倒不是硬體來不及收，而是可能你程式會來不及處理，尤其是在慢速的MCU上。

實際上Stop bit本身就可視為下個bit的idle狀態，因為兩個狀態一樣。邏輯上來說，你可以把Stop state經過2 bit之後才視為Idle，但實際上D7送完，接收器馬上就會standby要收下個byte了，除非你跟我一樣遇到兩光接收器，非得每個byte都間格一段時間，不然stop bit設定為1bit應該就可以了。

8-2 UART串列通訊函式庫

表 3.1 Serial 類別的方法

Serial 函式	功能說明
Serial.begin(baud) Serial.begin(baud,cfg)	【說明】啟用串列埠。 【參數】baud：傳輸速率，可選擇的速度有 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200。 cfg：設定資料長度、同位元跟停止位元，預設為 SERIAL_8N1。 【回傳】無
end()	【說明】停用串列埠；D0 與 D1 可當成數位接腳正常使用 【參數】無 【回傳】無
available()	【說明】查詢串列埠緩衝區中是否有可以讀取的資料。 【參數】無 【回傳】傳回串列埠緩衝區中可以讀取的字元數。
availableForWrite()	【說明】查詢是否有可以寫入串列埠的資料。 【參數】無 【回傳】傳回可以寫入的字元數。
flush()	【說明】等待串列埠輸出資料傳送完畢 【參數】無 【回傳】無

8-2 UART串列通訊函式庫

4.1.1. Serial.begin( )

【描述】啟用串列埠

【語法】Serial.begin(baud)

Serial.begin(baud, cfg)

【參數】

baud：設定傳輸速率，單位為 bps (bits per second) ，意思是「每秒傳送多少個位元」，可選擇的速率有 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200。

cfg：設定傳輸格式，包括資料長度、同位元跟停止位元，預設為 SERIAL_8N1，可選擇的格式有 SERIAL_5N1，SERIAL_6N1，SERIAL_7N1，SERIAL_8N1，
SERIAL_5N2，SERIAL_6N2，SERIAL_7N2，SERIAL_8N2，
SERIAL_5E1，SERIAL_6E1，SERIAL_7E1，SERIAL_8E1，
SERIAL_5E2，SERIAL_6E2，SERIAL_7E2，SERIAL_8E2，
SERIAL_5O1，SERIAL_6O1，SERIAL_7O1，SERIAL_8O1，
SERIAL_5O2，SERIAL_6O2，SERIAL_7O2，SERIAL_8O2。

【傳回值】無

8-2 UART串列通訊函式庫

print(val) print(val, format)	【說明】將資料以 ASCII 文字型式輸出（印出）至串列埠。 【參數】val：任意資料類型，包含數字，字元，字串。 format：數值格式，包含 BIN，DEC，OCT，HEX，小數點位數。 【回傳】傳回輸出（印出）的字元數。
println(val) println(val, format)	與 print()一樣，只是多了一個換行的動作。
read()	【說明】讀取從串列埠輸入（incoming）的資料，一次一個位元組，讀取後就從緩衝區移除。 【參數】無 【回傳】有資料就傳回第一個位元組，一定≧0； 如果沒資料就傳回-1。
peek()	【說明】窺視從串列埠輸入（incoming）的資料，與 read()最大的差別是讀取後並不會從緩衝區移除，若連續執行，讀到的都是同一個也是第一個位元組。 【參數】無 【回傳】有資料就傳回第一個位元組，一定≧0； 如果沒資料就傳回-1。
readBytes(buf, len)	【說明】從串列埠讀取資料存放到指定的記憶體空間，直到等於指定長度，或時間終了（參考 setTimeout()）。 【參數】buf：存放讀取資料的記憶體空間，常為字元陣列。 len：指定讀取資料的長度。 【回傳】傳回讀取並存放成功的字元數。

8-2 UART串列通訊函式庫

readBytesUntil(char, buf, len)	【說明】從串列埠讀取資料存放到指定的記憶體空間，直到等於指定長度，或遇到終止字元，或時間終了（參考 setTimeout()）。 【參數】char：終止字元。 buf：存放讀取資料的記憶體空間，常為字元陣列。 len：指定讀取資料的長度。 【回傳】傳回讀取並存放成功的字元數。
readString()	【說明】將串列埠緩衝區中的字元讀取到字串，直到時間終了。 【參數】無 【回傳】傳回存放資料的字串。
readStringUntil(char)	【說明】將串列埠緩衝區中的字元讀取到字串，直到遇到終止字元，或時間終了。 【參數】char：終止字元。 【回傳】傳回存放資料的字串。
parseInt()	【說明】從串列埠中尋找第一個有效的整數。 【參數】無 【回傳】傳回第一個有效的整數。
parseFloat()	【說明】從串列埠中尋找第一個有效的浮點數。 【參數】 【回傳】傳回第一個有效的浮點數。

8-2 UART串列通訊函式庫

find(target)	【說明】在串列埠緩衝區中搜尋目標字串。 【參數】target：目標字串。 【回傳】若找到傳回 true，否則傳回 false。
findUntil(target,terminal)	【說明】在串列埠緩衝區中搜尋目標字串直到終止字串出現。 【參數】target：目標字串。 terminal：終止字串。 【回傳】若找到傳回 true，否則傳回 false。
write(val) write(str) write(buf, len)	【說明】寫出資料到串列埠。 【參數】val：一個位元組數值。 str：字串。 buf：字元陣列。 len：字元陣列的長度。 【回傳】傳回寫出成功的字元數。
setTimeout(time)	【說明】設定從串列埠讀取資料最大的等待時間。 【參數】time：最大的等待時間，單位是毫秒 ms，預設為 1000。 【回傳】無

serialEvent() {
//statements
}
	【說明】當串列埠有可用的資料時會觸發此一函式。
	【參數】任何有效的敘述
	【回傳】無

範例 8-1

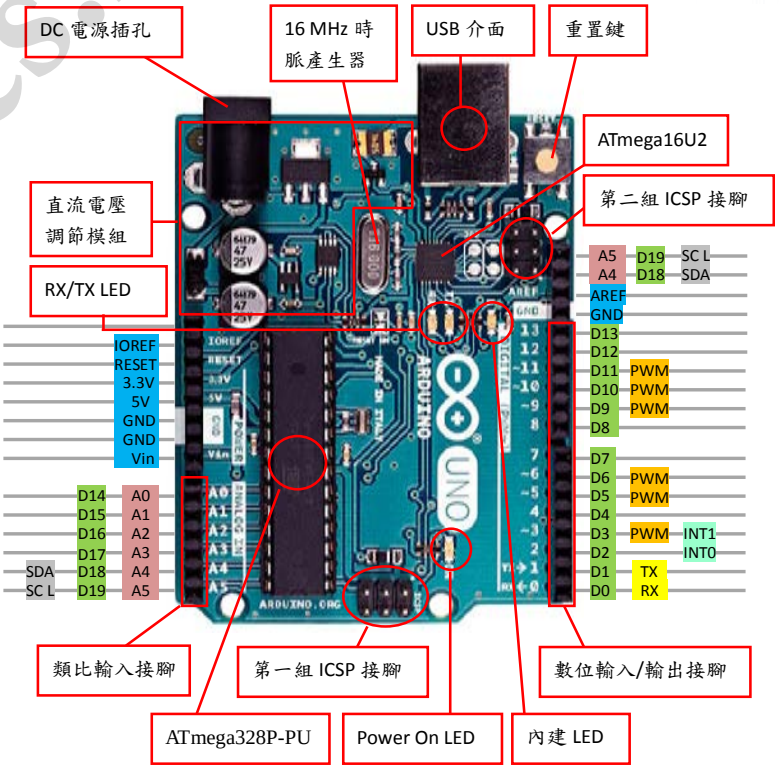
- 把左邊 arduino 的 RX 接到右邊的 TX
- 把左邊 arduino 的 TX 接到右邊的 RX
- 最後再把 GND 互連就可以了

```
1
2  int LED=13;
3
4  void setup() {
5    pinMode(LED, OUTPUT);
6    Serial.begin(9600);
7  }
8
9  void loop() {
10   Serial.write('Y'); //先對 TX 送出字元 Y
11   //檢查 RX 緩衝器, 直到有資料進來
12   while (!Serial.available()) {}
13   if (Serial.read()=='Y') {
14   //若收到 Y, LED 閃兩下
15     led_blink();
16     led_blink();
17   }
18 }
19
20 void led_blink() {
21   digitalWrite(LED, HIGH);
22   delay(1000);
23   digitalWrite(LED, LOW);
24   delay(500);
25 }
```

```
1
2  int LED=13;
3
4  void setup() {
5    pinMode(LED, OUTPUT);
6    Serial.begin(9600);
7  }
8
9  void loop() {
10   //檢查 RX 緩衝區, 直到有資料進來
11   while (!Serial.available()) {}
12   if (Serial.read()=='Y') {
13   //若收到字元 Y, LED 閃兩下
14     led_blink();
15     led_blink();
16   }
17   Serial.write('Y'); //在 TX 送出字元 Y 給對方
18 }
19
20 void led_blink() {
21   digitalWrite(LED, HIGH);
22   delay(1000);
23   digitalWrite(LED, LOW);
24   delay(500);
25 }
```

SoftwareSerial

- Arduino 板子一般常用的 UNO, Nano, Pro mini 都只有一組硬體 UART 串列埠, 即 DIO Pin0 (RX) 與 Pin1 (TX), 而 Mega 則有四組, 因為 UNO 與 Nano 的串列埠與 USB 並接, 因此如果 USB 接 PC 時就只能與 PC 通訊, 無法同時接其他設備, 於是 Mikal Hart 開發了 SoftwareSerial 函式庫, 可以用軟體模擬方式定義多組 DIO 腳當作 UART 埠, 速率可達 11500 bps, Arduino IDE 1.0 版之後已經納入此函式庫。



範例 8-2: SoftwareSerial

- 把左邊 arduino 的 D11 接到右邊的 D10
- 把左邊 arduino 的 D10 接到右邊的 D11
- 最後再把 GND 互連就可以了

```
1
2  #include <SoftwareSerial.h>
3
4  //建立軟體串列埠腳位 (RX, TX)
5  SoftwareSerial mySerial(11,10);
6
7  void setup() {
8    mySerial.begin(9600);
9  }
10
11 void loop() {
12   while (!mySerial.available()) {}
13   Serial.write(mySerial.read());
14   while(mySerial.available() > 0) {
15     mySerial.read();
16   }
17   mySerial.write("1");
18 }
```

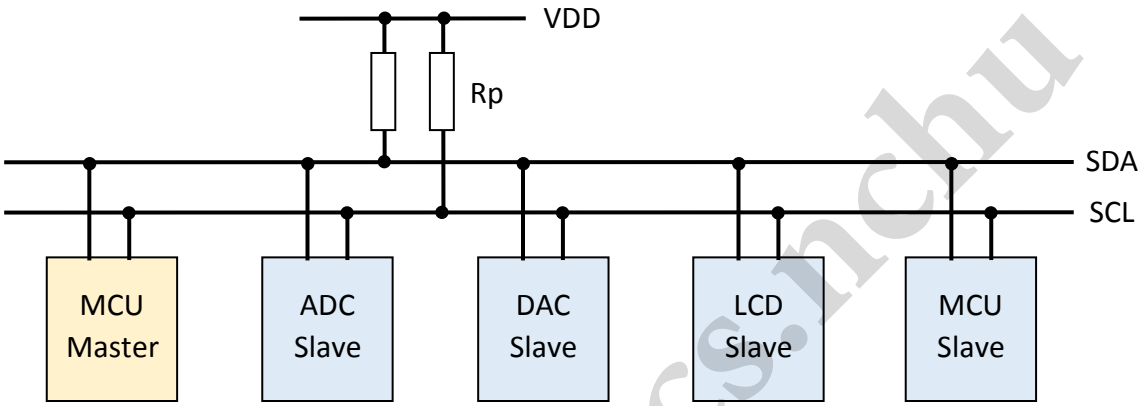
```
1
2  #include <SoftwareSerial.h>
3
4  //建立軟體串列埠腳位 (RX, TX)
5  SoftwareSerial mySerial(11,10);
6
7  void setup() {
8    Serial.begin(9600);
9    mySerial.begin(9600);
10 }
11
12 void loop() {
13   while (!mySerial.available()) {}
14   Serial.write(mySerial.read());
15   while (mySerial.available()>0) {
16     mySerial.read();
17   }
18 }
```

I2C
（IC積體電路間通訊）

8-3 I2C

IIC，I^{2}C，I2C 指的都是同一項技術的縮寫，其正確的唸法應該是 “I-Square-C”，可能是不好唸的原因，後來雖然是錯誤卻廣為流行的唸成 “I-Two-C”。I2C 是 1982 年由荷蘭飛利浦半導體公司所開發的串列通訊技術，其英文全名為 Inter Integrated Circuit (Inter IC, IIC)，意思即為 IC 積體電路之間的通訊，其發展目的是為了讓微控制器（MCU）以較少的接腳連接眾多的低速裝置之用，由於 I2C 簡單易用而且具有高度的可擴充性，所以很快的就廣為流行使用，後來飛利浦公司在 1987 年申請了 I2C 的商標及專利，並陸續控告過許多半導體公司侵權，為了規避 I2C 的專利，其他公司就將其產品設計成和 I2C 相容，但是名稱卻改成了 Two Wire Interface (TWI) ，所以 I2C 與 TWI 實際上是相同的技術內容，只是使用的名稱不同。一直到 2004 年 I2C 的專利到期，從 2006 年 10 月 1 日開始，在晶片上加入 I2C 功能已經不需要再支付專利費，所以現在各家廠商都可以自由的使用 I2C 名稱，但周邊裝置的製造商如果想取得專屬的 I2C slave（從屬）裝置地址則還是需要付費給 NXP 恩智浦半導體公司（此為飛利浦在 2006 年分拆出來的半導體公司）。

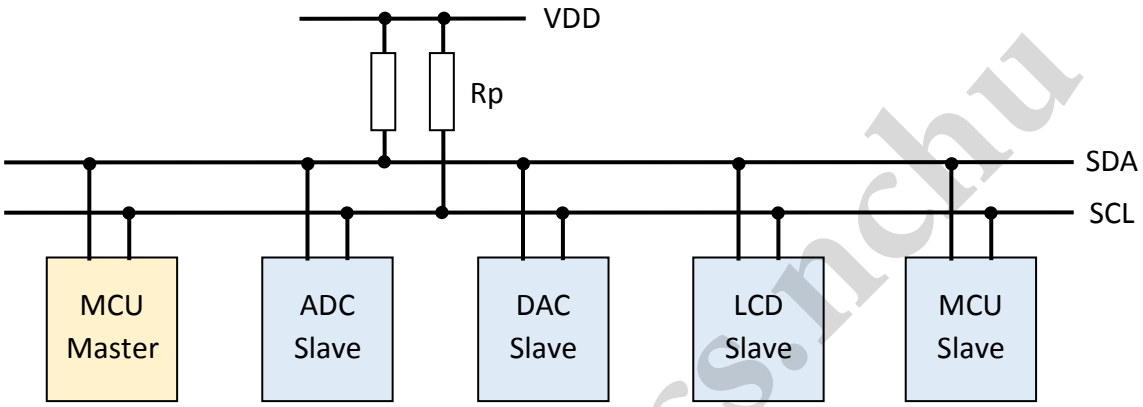
8-3-1 I2C 匯流排



I2C 是以匯流排型式（bus）連接所有的裝置，整個 I2C 匯流排架構如圖 9.4.1 所示，從中我們可以觀察到，I2C 最大的特色就是只需要利用二條傳輸導線就能連接多個裝置，並且完成裝置間的資料傳輸，這二條傳輸線分別是 SDA 與 SCL，SDA（Serial Data）代表串列資料，而 SCL（Serial Clock）則是串列時脈。整理 I2C 匯流排的特色條列如下：

(1) I2C 是一種主從式（Master-Slave）的通訊架構，在同一組匯流排上允許有多個主控裝置（master）和多個從屬裝置（slave）。如果 master 裝置只有一個，稱為單主控架構，大部分系統都採用此架構；若 master 裝置不只一個，則為多主控架構，此架構較為複雜少用，雖然有多個主控裝置，但同一時間也只允許一個主控有動作。

8-3-1 I2C 匯流排

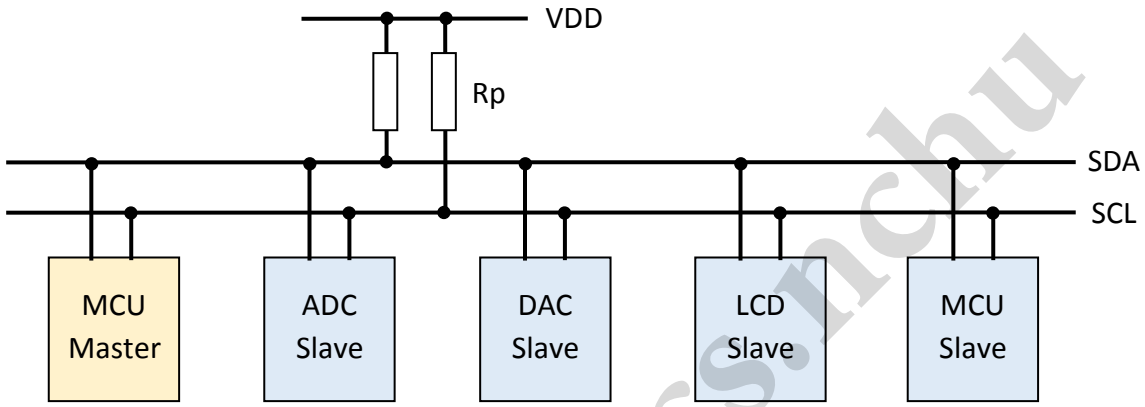


(1) SDA 與 SCL 均為雙向的訊號線，而且都經過上拉電阻 Rp 連接到電壓源，所以 I2C 匯流排在閒置狀態時，也就是沒有任何裝置在進行資料傳輸，SDA 與 SCL 都會保持在高電位的狀態。

(2) I2C 上所有的裝置都以 wired-AND 的方式並接在匯流排上，在閒置狀態下，若有裝置有傳輸的需求，該裝置只要將訊號線拉降到 LOW 就可取得匯流排的控制權進行傳輸，這裡要特別強調，在一個時間點，只能允許一個裝置進行拉降的動作。

(3) SDA 訊號線是用來傳輸資料的內容，相較之下功能較為單純，而 SCL 則是傳輸時脈訊號，用來控制資料讀寫的動作，所以 SCL 上的時脈頻率可決定 I2C 資料傳輸的速率。通常 I2C 在標準模式下的傳輸速率為 100 Kbit/s、低速模式可到 10 Kbit/s，但時脈頻率也可下降至 0，這代表暫停通訊；隨著硬體技術的演進與應用的需求，在後續的改版修訂中，I2C 也提供了更快速的傳輸速率，在快速模式下可達到 400 Kbit/s，高速模式下可達 3.4 Mbit/s，甚至是 5Mbit/s 的超快速模式。

8-3-1 I2C 匯流排

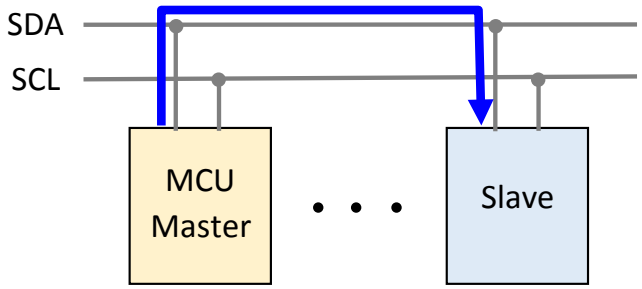


(1) I2C 使用 7-bit 的長度來定義連接裝置的位址，所以理論上一組 I2C 匯流排最多可以連接到 128 個裝置，不過 I2C 的通訊協定保留了 16 個位址作為系統擴充使用，所以實際上最多只能接到 112 個裝置，表 9.4.1 列出了這 16 個保留位址，分別是位址 0~7 跟位址 120~127。

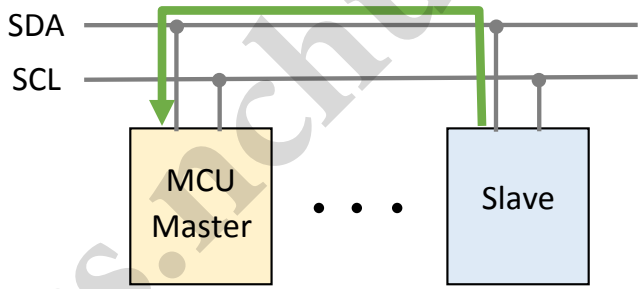
(2) 後續新版的 I2C 更擴充了可連接裝置的位址空間，將 7-bit 的位址空間增加到 10-bit，如此可大幅的增加 slave 的數量，使用者可根據晶片的規格，選用標準的 7-bit 或是擴充版的 10-bit 的位址空間。

(3) 任何 slave 裝置都可在系統仍然在運作的時候加入或移出 I2C 匯流排，這表示對於有熱插拔需求的裝置而言，I2C 是個理想的匯流排。

8-3-2 I2C 讀寫動作



(a)寫入資料



(b)讀取資料

因為 I2C 匯流排是屬於主從式的架構，所有資料傳輸的動作一定都是由 master 所發起，slave 只會針對 master 的命令被動的做出反應，分析 I2C 的資料傳輸只會有二種動作，如圖 9.4.3 所示，第一種是 master 寫資料到 slave，第二種是 master 從 slave 讀取資料。首先解釋第一種寫入資料的流程，參考圖 9.4.3 的圖例(a)說明，master 會將資料寫到匯流排，再由指定的 slave 從匯流排將資料讀取下來，在此過程中很明顯的 master 是傳送端，而 slave 是接收端，完整的傳輸流程可搭配圖 9.4.4 的圖例詳細說明如下：

8-3-3 I2C的函式庫

1. Wire.begin( )

【描述】初始化 I2C 並將裝置加入 I2C 匯流排。

【語法】Wire.begin()

Wire.begin(address)

【參數】

address：指定裝置 I2C 的位址，長度為 7 個位元，此參數可有可無。（1）若沒指定位址，在加入 I2C 匯流排之後就會是主控（Master）裝置。（2）若有指定位址，在加入 I2C 匯流排之後就是從屬（Slave）裝置，其位址就是 address 參數。根據 Arduino 官方網頁說明，位址 0~7 是系統保留不能使用，所以第一個可以使用的 I2C 位址是 8。

【傳回值】無

8-3-3 I2C的函式庫

1. Wire.requestFrom( )

【描述】由 master 裝置發起，向 slave 裝置請求資料，後續再使用 available() 和 read() 函式讀取資料。

【語法】Wire.requestFrom(address, quantity)

Wire.requestFrom(address, quantity, stop)

【參數】

address：指定 slave 裝置的位址，長度為 7 個位元。

quantity：請求的 byte 數

stop：資料型別為 boolean，此參數可有可無，若沒指定則預設為 true。這個參數可以改變 requestFrom() 執行後的 I2C 狀態，（1）如果 stop 為 true，則 requestFrom() 在傳送完請求後會送出停止的訊息，釋放 I2C 介面。（2）如果 stop 為 false，則 requestFrom() 在傳送完請求後會送出 restart 的訊息，繼續佔用 I2C 介面，使得其他的 master 裝置無法發送請求，這樣可以允許一個 master 向 slave 裝置連續發送多次請求。

【傳回值】從 slave 裝置傳回的 byte 數

8-3-3 I2C的函式庫

1. Wire.beginTransmission( )

【描述】開始傳送資料到指定的 slave 裝置的動作，接下來會使用 write() 函式把資料寫入佇列（queue），最後再呼叫 endTransmission() 把資料傳送到 slave 裝置。

【語法】Wire.beginTransmission(address)

【參數】

address：指定 slave 裝置的位址，長度為 7 個位元。

【傳回值】無

8-3-3 I2C的函式庫

1. Wire.endTransmission( )

【描述】結束傳送資料到指定的 slave 裝置的動作，此一傳輸必須先由 beginTransmission() 開始，再將 write() 寫入到佇列的資料傳送到 slave 裝置。

【語法】Wire.endTransmission( )

Wire.endTransmission(stop)

【參數】

stop：資料型別為 boolean，此參數可有可無，若沒指定則預設為 true。這個參數可以改變 endTransmission() 執行後的 I2C 狀態，（1）如果 stop 為 true，則 endTransmission() 在傳送完資料後會送出停止的訊息，釋放 I2C 介面。（2）如果 stop 為 false，則 endTransmission() 在傳送完資料後會送出 restart 的訊息，繼續佔用 I2C 介面，使得其他的 master 裝置無法傳送資料，這樣可以允許一個 master 向 slave 裝置連續傳送多筆資料。

【傳回值】傳回資料傳送的狀態，其資料型態為 byte，有下列 5 種傳回值。

0：表示傳送成功

1：資料長度超過傳輸緩衝區

2：在傳送位址時收到 NACK（Negative acknowledge）否認訊號

3：在傳送資料時收到 NACK（Negative acknowledge）否認訊號

4：其他錯誤

8-3-3 I2C的函式庫

1. Wire.write( )

【描述】將資料從 slave 裝置寫入（傳送）到發出請求的 master 裝置，或是將 master 要傳送到 slave 的資料寫入到佇列(queue)，此函式會使用在 beginTransmission() 和 endTransmission() 之間。

【語法】Wire.write(value)

Wire.write(string)

Wire.write(data, length)

【參數】

value：要傳送的數值，資料型別為 byte。

string：要傳送的字串資料。

data：要傳送的資料陣列，其單元的資料型別為 byte。

length：要傳送的 byte 數。

【傳回值】傳回傳送成功的 byte 數。

8-3-3 I2C的函式庫

1. Wire.available( )

【描述】傳回可用 read() 讀取的 byte 數，此函式可在 master 裝置向 slave 裝置請求資料 requestFrom() 之後使用，或是在 slave 裝置使用 onReceive() 註冊的函式裡呼叫使用。

【語法】Wire.available( )

【參數】無

【傳回值】傳回可讀取的 byte 數。

2. Wire.read( )

【描述】讀取接收到的資料，一次一個 byte，在 master 裝置向 slave 裝置請求資料 requestFrom() 之後使用，或是從 master 裝置傳送資料到 slave 裝置使用。

【語法】Wire.read( )

【參數】無

【傳回值】傳回一個 byte 的資料，如果沒有資料可以讀取，則傳回-1。

8-3-3 I2C的函式庫

1. Wire.setClock( )

【描述】設定 I2C 傳輸的時脈頻率，對 slave 裝置而言沒有最低的工作時脈頻率，但一般都是以 100KHz 為下限。

【語法】Wire.setClock(freq)

【參數】

freq：欲設定的頻率值，以赫茲（Hertz）為單位，可接受的頻率值有標準模式的 100000 (100KHz)和快速模式的 400000 (400KHz)。有些處理器也支援更低速的 10KHz，或更高速的 1000KHz，甚至是超高速的 3400K，這些細節必須要參考特定處理器的規格書才能確定有沒有支援。

【傳回值】無

8-3-3 I2C的函式庫

1. Wire.onReceive( )

【描述】指定 slave 裝置在接收到從 master 裝置傳送過來的資料時要執行的函式。

【語法】Wire.onReceive(handler)

【參數】

handler：函式的名稱，此函式需要一個資料型別為 int 的參數，而且沒有回傳值，例如：void myHandler(int numBytes)，其中參數 numBytes 表示從 master 裝置傳送過來的 byte 數。

【傳回值】無

2. Wire.onRequest( )

【描述】指定 slave 裝置在接收到從 master 裝置傳送過來的請求時要執行的函式。

【語法】Wire.onRequest(handler)

【參數】

handler：函式的名稱，此函式不需要參數也沒有回傳值，例如：void myHandler( )。

【傳回值】無

範例 8-3

```
1  #include <Wire.h>
2  //*****************************************************
3  void setup()
4  {
5    Serial.begin(9600);
6    Serial.println("I2C Slave Scanning ...");
7    Wire.begin();                          //將本裝置初始成 master
8    for(byte addr=8; addr<120; addr++) {   //從位址 8 開始掃瞄到位址 119
9                                           //原因詳見表 9.4.1
10     Wire.beginTransmission(addr);        //開始傳送資料到指定的 slave 位址
11     if(Wire.endTransmission()==0) {      //假如有回傳 ACK 就顯示其位址
12       Serial.print("Find slave at: 0X");
         Serial.println(addr, HEX);
       }
     }
     Serial.println("End ...");
   }

   //*****************************************************
   void loop()
   { }
```

範例 8-3

【範例】I2C slave 位址掃描程式

【說明】

在嵌入式系統應用中，我們常常會使用到 I2C 的 I/O 裝置或感測器，可是這些裝置的技術文件轉來轉去，可能下載錯了，也可能是型號改版了，導致我們明明按照手冊上的 I2C 位址撰寫，但就是無法驅動，所以底下的範例就是提供一個 I2C 位址掃描的程式，只要接上 I2C 的裝置，執行程式後就可以抓取到正確的位址，再也不怕搞錯了。

【結果】

在此範例中，我們使用 I2C 介面的 LCD 1602 當做 slave 裝置，如圖 9.4.6(a)與 UNO 連接正確後，執行結果即可顯示出 LCD 1602 的位址為 0X27。

範例 8-4

【範例】使用 I2C 完成二塊 UNO 開發板之間的通訊

【說明】

如圖 9.4.7 將二塊 UNO 開發板的 A4 (SDA) 與 A5 (SCL) 各自對接，即可建立 I2C 匯流排的連線，其中要特別注意，雖然二塊板子採用獨立供電的方式，但是 I2C 通訊協定是靠拉降訊號線的動作以取得匯流排的控制權，所以二塊板子的 GND 一定要接在一起共用，如此 I2C 才能正常的動作。在此範例中，我們設定 UNO_A 為 master，而 UNO_B 規劃成 slave，其位址指定為 0x08，二個微控器的程式碼完全不同，要各自上傳執行。

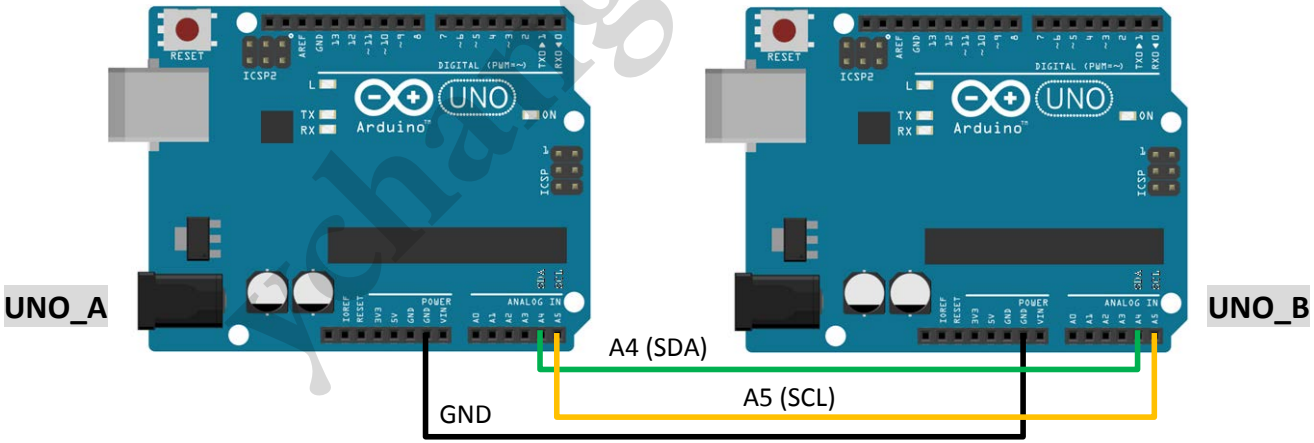


圖 9.4.7 二塊 UNO 開發板之間的 I2C 通訊

範例 8-4

UNO_A: Master

```
1  #include <Wire.h>
2  int cnt=0;
3  //*****************************************************
4  void setup() {
5    pinMode(13,OUTPUT);         //設定內建 LED(D13)接腳為輸出模式
6    Wire.begin();               //將本裝置初始成 master
7  }
8
9  //*****************************************************
10 void loop() {
11   if(cnt==10) cnt=0;
12   digitalWrite(13,HIGH); delay(1000-cnt*100);  //點亮 LED 一段時間後熄滅
13   digitalWrite(13,LOW);
14   Wire.beginTransmission(8);      //向 slave #8 傳送資料
15   Wire.write(cnt);                //傳送 cnt 的值
16   if(Wire.endTransmission()==0) { //傳送成功
17     while(1) {
18       Wire.requestFrom(8,1);      //向 slave #8 請求 1 byte 資料
19       while(!Wire.available());
20       if(Wire.read()==(cnt+1))    //如果 slave 回傳的值等於 cnt+1
21       { cnt++; break; }           //cnt=cnt+1，並跳出等待迴圈
22     }
23   }
24 }
```

範例 8-4

UNO_B: Slave

```
1  #include <Wire.h>
2  int flag=0, cnt=0;
3  //*****************************************************
4  void setup() {
5    pinMode(13,OUTPUT);             //設定內建 LED(D13)接腳為輸出模式
6    Wire.begin(8);                  //將本裝置初始成 slave，且位址為 8
7    Wire.onReceive(receiveEvent);   //註冊 onReceive 要執行的函式
8    Wire.onRequest(requestEvent);   //註冊 onRequest 要執行的函式
9  }
10
11 //*****************************************************
12 void loop() {
13   if(flag==1) {                   //flag 為 1 表示從 master 接收到資料
14     digitalWrite(13,HIGH); delay(1000-cnt*100);
15     digitalWrite(13,LOW);
16     cnt++;  flag=0;
17   }
18 }
19
20 //*****************************************************
21 void receiveEvent(int bytes) {
22   while(!Wire.available());
23   cnt=Wire.read();
24   flag=1;
25 }
26
27 //*****************************************************
28 void requestEvent( ) {
29   Wire.write(cnt);
30 }
```

範例 8-4

【結果】

觀察執行結果，二塊開發板上的 LED 燈會交互閃爍，master 先亮，熄滅後換 slave 亮，熄滅後又換回 master，而且閃爍的時間會從 1 秒遞減到 0.1 秒，之後再回到 1 秒的時間，如此無限循環。