Chap 8-3
串列傳輸 SPI

SPI
（串列週邊介面）

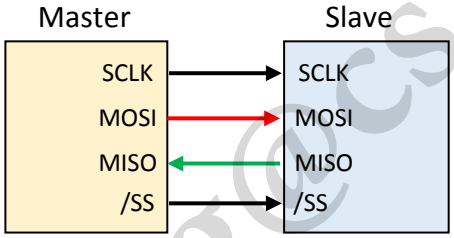
8-3 SPI串列通訊

串列週邊介面（Serial Peripheral Interface），縮寫為 SPI，是由 Motorola 公司所研發的另一種串列通訊技術，1985 年首次應用在 Motorola 自己的 8 位元微控器上，有別於 1982 年由 Philips 公司所提出的 I2C 只能進行半雙工的資料傳輸，SPI 是一種同步而且全雙工的串列通訊技術，所以嚴格來說，SPI 是 I2C 的進化版，除了繼承 I2C 的優點外，更針對 I2C 低速傳輸的缺點，改用全雙工大幅的提升資料傳輸的效率。

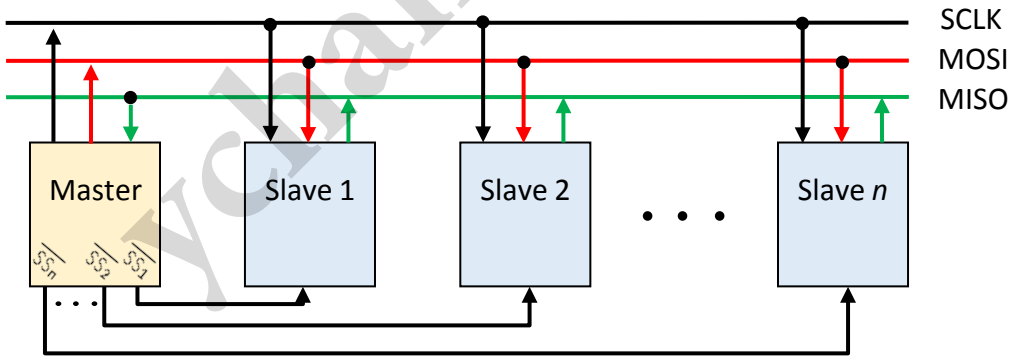
	同步	非同步
半雙工	I2C	
全雙工	SPI	UART

8-3-1 SPI匯流排

SPI 匯流排架構如圖 9.5.1 所示，與 I2C 一樣，SPI 也是屬於主從式的通訊架構，圖 9.5.1(a)是一對一最簡單的連接，而 9.5.1(b)則為一對多的連接方式，從圖中我們可以觀察到，SPI 需要利用四條傳輸線才能達成 master 與 slave 之間的資料傳輸，這四條傳輸線分別是 SCLK，MOSI，MISO 與 /SS，其意義與作用分別敘述在表 9.5.1 中。



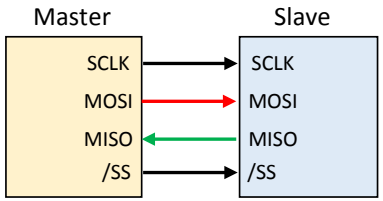
(a) 一對一，一個 master 連結一個 slave



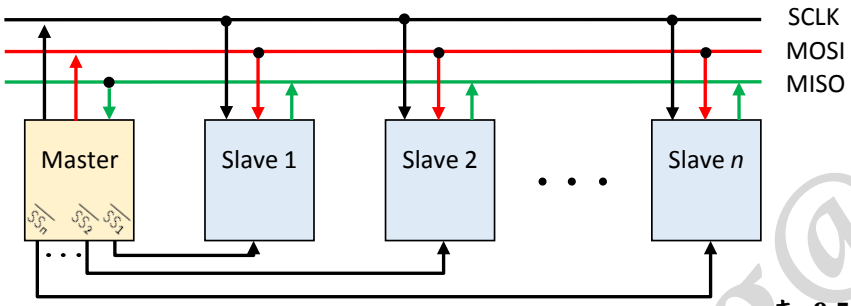
(b) 一對多，一個 master 連結多個 slave

圖 9.5.1 SPI 匯流排架構

8-3-1 SPI匯流排



(a) 一對一，一個 master 連結一個 slave



(b) 一對多，一個 master 連結多個 slave

表 9.5.1 SPI 串列傳輸的訊號線

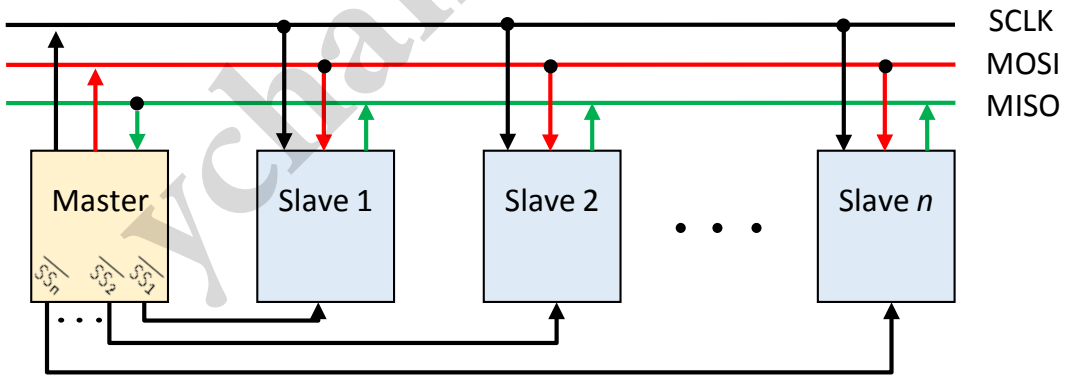
傳輸線	全名	描述
SCLK	Serial Clock	由 master 所產生的時脈訊號，其目的是為了要正確的控制資料同步傳輸的動作。
MOSI	Master Out Slave In	主出從入，表示 master 是輸出端，slave 是輸入端，所以在此傳輸線上的資料流向是從 master 輸出到 slave。
MISO	Master In Slave Out	主入從出，表示 master 是輸入端，而 slave 是輸出端，此傳輸線上的資料流向是從 slave 輸出到 master。
/SS	Slave Select	由 master 所產生的 slave 選擇訊號，slave 只有在其專屬的/SS 訊號線為 LOW 時才表示被致能，可以與 master 進行資料的傳輸。

8-3-1 SPI匯流排的特色

(1) 除/SS 外，SCLK，MOSI 與 MISO 均為共用的訊號線，slave 數量的增減不會影響這三條訊號線的數量。

(2) /SS 是專屬的致能訊號線，每個 slave 與 master 之間都有獨立的/SS，不能共用而且是訊號為 LOW 的時候才表示致能，與 I2C 比較起來，SPI 選擇 slave 致能的動作較為簡單直接，但是卻增加了 master 晶片接腳與連接導線的數量。

(3) 因為/SS 是專屬訊號線的原因，SPI 匯流排只允許一個 master 裝置的存在，而且也無法像 I2C 一樣可以熱插拔的特性，必須停止系統才能增減 slave 裝置。



(b) 一對多，一個 master 連結多個 slave

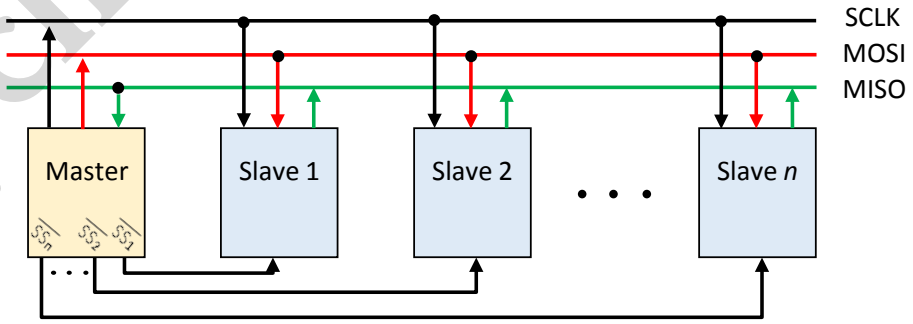
8-3-1 SPI匯流排的特色

(4) MOSI 與 MISO 為二條傳輸方向不同的資料線，在 SCLK 的觸發下同時會進行資料的傳輸，MOSI 是從 master 輸出到 slave，而 MISO 則是從 slave 輸出到 master，如此實現了 SPI 同步全雙工的資料傳輸，傳輸速率一般可達 5M/10M/20Mbps 或是更快，這也是 SPI 最吸引人的效能優勢。

(5) SPI 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，MOSI，MISO 與 /SS 都會維持在高電位的狀態，而 SCLK 的電壓準位則要看模式而定。

(6) 與 I2C 一樣，SPI 的資料傳輸順序也是從 MSB 開始先傳，依序往低位元的方向，LSB 是最後傳輸的位元。

(7) SPI 的傳輸協定沒有包含交握機制（handshaking），所以無法確認資料是否已正確的傳輸。



(b) 一對多，一個 master 連結多個 slave

8-3-1 SPI的接腳

表 9.5.2 不同 Arduino 開發版中 SPI 的接腳對應

開發版	MOSI	MISO	SCLK	SS (Slave)	SS (Master)	level
Uno Duemilanove	11 ICSP-4	12 ICSP-1	13 ICSP-3	10	-	5V
Mega1280 Mega2560	51 ICSP-4	50 ICSP-1	52 ICSP-3	53	-	5V
Leonardo	ICSP-4	ICSP-1	ICSP-3	-	-	5V
Due	ICSP-4	ICSP-1	ICSP-3	-	4,10,52	3.3V
Zero	ICSP-4	ICSP-1	ICSP-3	-	-	3.3V
101	11 ICSP-4	12 ICSP-1	13 ICSP-3	10	10	3.3V
MKR1000	8	10	9	-	-	3.3V

8-3-2 SPI的資料傳輸

在 SPI 匯流排中，slave 裝置一旦被選擇致能之後，就會與 master 之間形成一個資料傳輸的迴路，如圖 9.5.2 所示，master 內部的資料移位暫存器（shift register），會與 slave 內部的資料移位暫存器透過 MOSI 與 MISO 二條傳輸線頭尾相接，形成一個環狀佇列。這裡要特別注意資料的流向，SPI 也是最高位元 MSB 先傳，所以 master 暫存器的 MSB 移出後，會經由 MOSI 傳輸線，輸入成為 slave 暫存器的 LSB；同理，slave 暫存器的 MSB 移出後，也會經由 MISO 傳輸線成為 master 暫存器的 LSB。

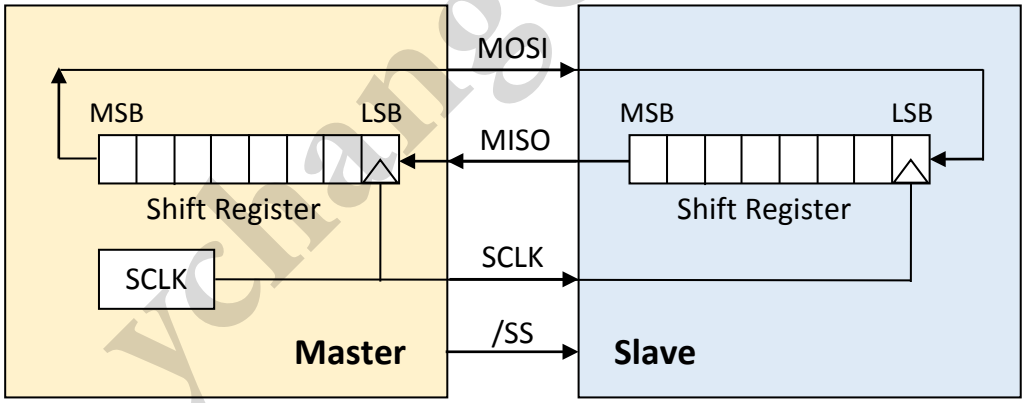


圖 9.5.2 SPI master 與 slave 之間的資料傳輸

8-3-2 SPI的資料傳輸

而另一個控制的重點，就是 master 所產生的時脈訊號，不僅會連接到 master 內部的移位暫存器，也會經由 SCLK 傳輸線傳送到 slave 的移位暫存器使用，其目的是為了要同步觸發二個暫存器的移位動作，達到資料接收與傳送可同時進行的全雙工。在 SPI 的傳輸協定下，每一次 SCLK 時脈的觸發，都會造成 master 與 slave 之間一個位元資料的交換，所以在 8 個時脈觸發之後，就可完成一個 byte 的資料傳輸，其速度可快可慢，也可暫停不動，換句話說 SPI 的傳輸速率完全取決於 master 所產生的 SCLK 時脈頻率。

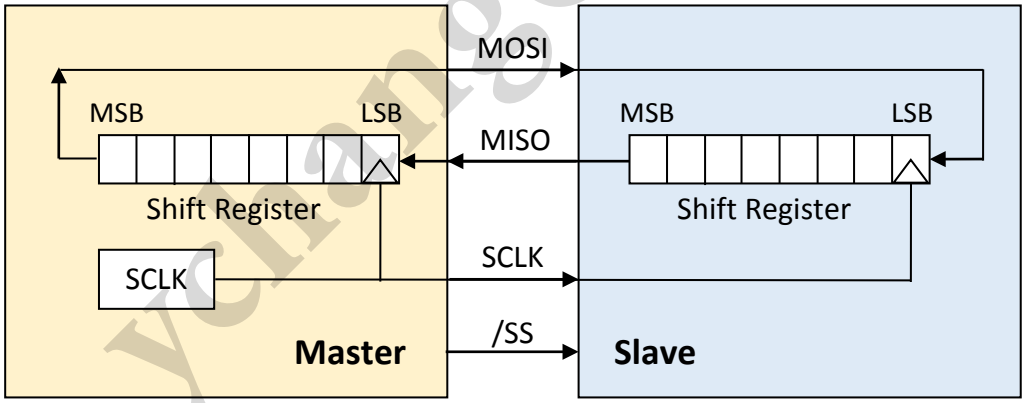


圖 9.5.2 SPI master 與 slave 之間的資料傳輸

8-3-3 SPI的傳輸模式

與 I2C 比較起來，SPI 的傳輸算是一個比較鬆散，沒有標準化的通訊協定，所以在使用 SPI 裝置之前必須仔細閱讀裝置的資料手冊與規格，唯有正確的設定才能完成 SPI 的資料傳輸。不同於 I2C 有明確而且一致的傳輸方式，SPI 有四種不同的傳輸模式，取決於二個重要的參數，時脈極性 CPOL 與時脈相位 CPHA。

時脈極性（clock polarity），縮寫為 CPOL，此參數是用來表示時脈閒置時的電壓準位究竟是 HIGH 還是 LOW？如表 9.5.2 所示，CPOL=0 表示時脈閒置時的電壓準位為 LOW；反之 CPOL=1 則表示時脈閒置時的電壓準位為 HIGH。

時脈相位（clock phase），縮寫為 CPHA，此參數是用來表示資料取樣（sample）的動作是發生在時脈的前邊緣（leading edge）還是後邊緣（trailing edge）？也有人說是第一邊緣跟第二邊緣，如表 9.5.2 所示，CPHA=0 表示資料的取樣是發生在時脈的前邊緣，而 CPHA=1 則表示資料的取樣是發生在時脈的後邊緣。

參數	值	描述
CPOL 時脈極性	=0	時脈閒置時的電壓準位為 LOW
	=1	時脈閒置時的電壓準位為 HIGH
CPHA 時脈相位	=0	資料的取樣發生在時脈的前邊緣或第一邊緣（leading edge），
	=1	資料的取樣發生在時脈的後邊緣或第二邊緣（trailing edge），

8-3-3 SPI的傳輸模式

(1) 模式 0：

設定 CPOL=0 且 CPHA=0，如圖 9.5.3 的圖示說明，CPHA=0 表示資料的取樣發生在時脈的第一邊緣（leading edge），因為 CPOL=0 代表時脈閒置時的電壓準位為 LOW，所以第一邊緣必為 L 到 H 的轉換，也就是說資料取樣是發生在時脈的上升邊緣，MISO 與 MOSI 上的資料在 SCLK=H 時必須保持穩定不變，在 SCLK=L 時才允許改變。

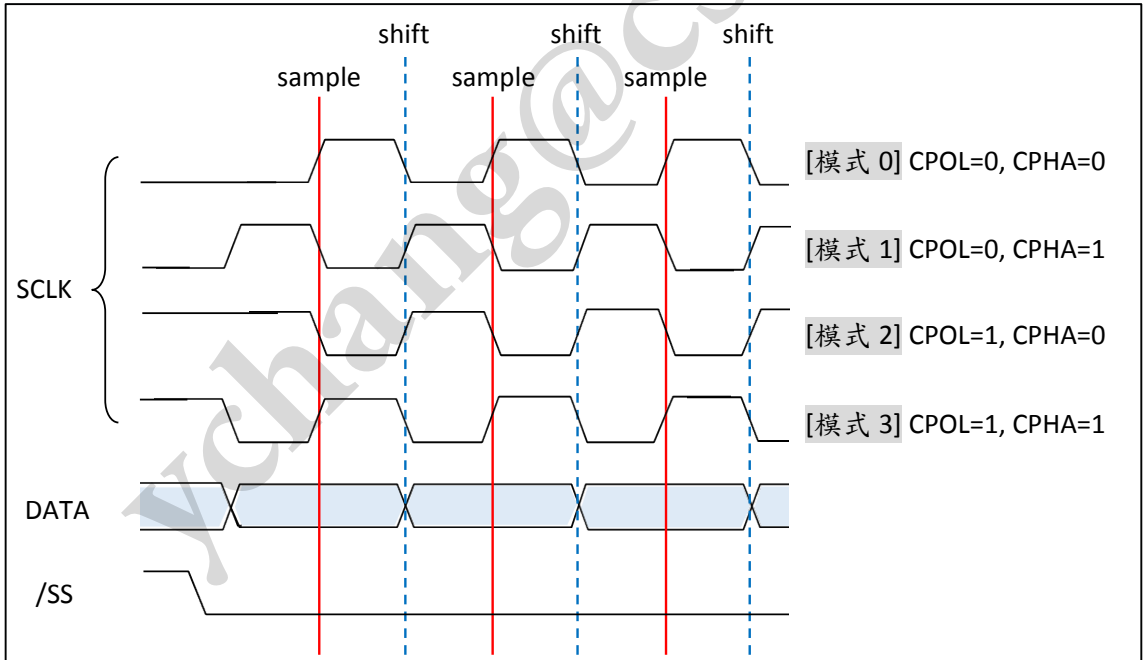


圖 9.5.3 SPI 的四種傳輸模式

8-3-3 SPI的傳輸模式

(2) 模式 1：

設定 CPOL=0 且 CPHA=1，與模式 0 相比，CPHA=1 表示資料的取樣發生在時脈的第二邊緣（trailing edge），所以在模式 1 下，資料取樣是發生在時脈的下降邊緣，MISO 與 MOSI 上的資料在 SCLK=L 時必須保持穩定不變，在 SCLK=H 時才允許改變。

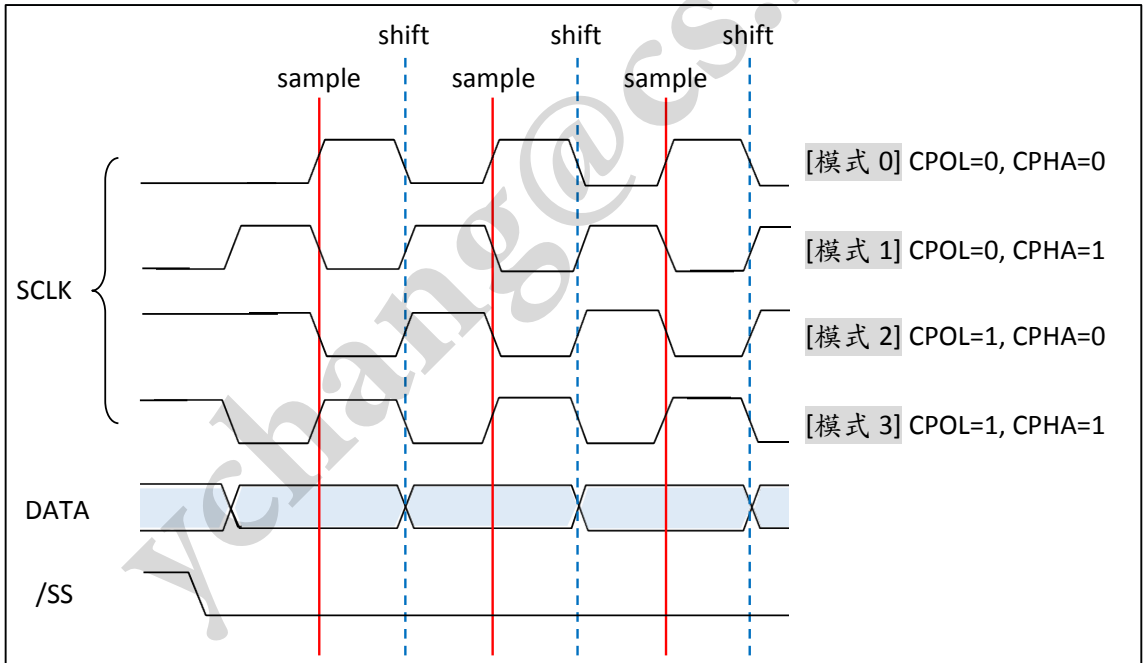


圖 9.5.3 SPI 的四種傳輸模式

8-3-3 SPI的傳輸模式

(3) 模式 2：

設定 CPOL=1 且 CPHA=0，如圖 9.5.3 的圖示說明，CPHA=0 表示資料的取樣發生在時脈的第一邊緣（leading edge），因為在模式 2 下 CPOL=1 代表時脈閒置時的電壓準位為 HIGH，所以第一邊緣為 H 到 L 的轉換，也就是說資料取樣會發生在時脈的下升邊緣，MISO 與 MOSI 上的資料在 SCLK=L 時必須保持穩定不變，在 SCLK=H 時才允許改變。

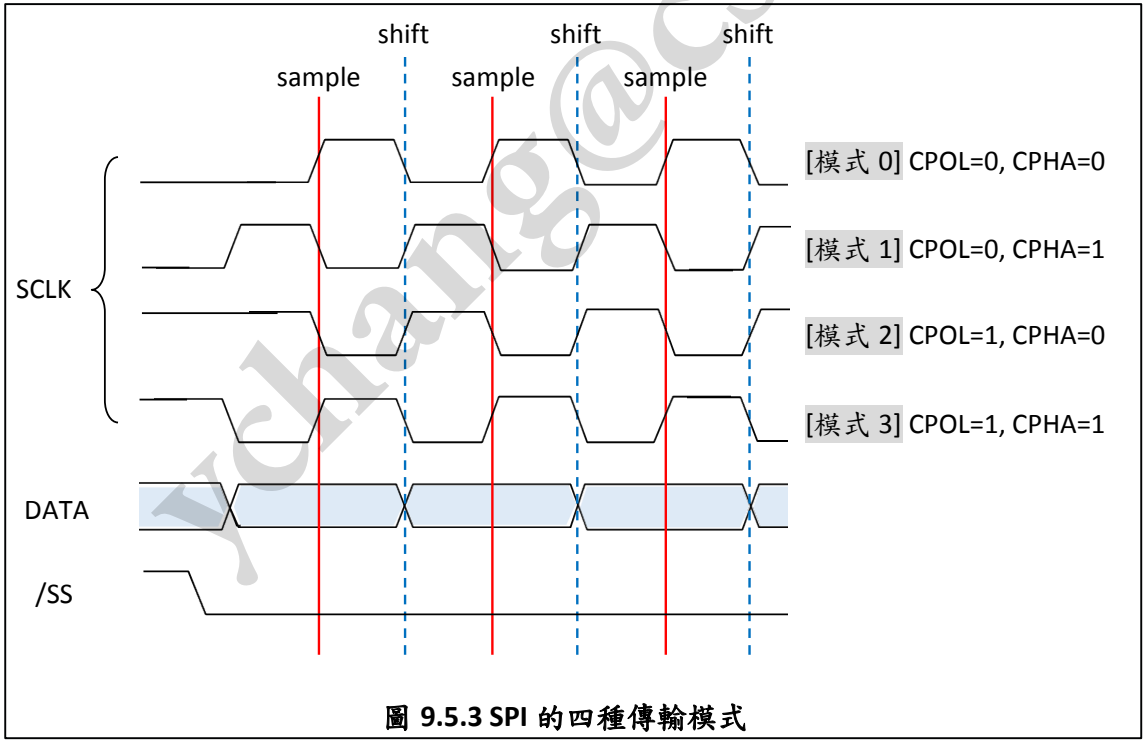


圖 9.5.3 SPI 的四種傳輸模式

8-3-3 SPI的傳輸模式

(4) 模式 3：

設定 CPOL=1 且 CPHA=1，與模式 2 相比，CPHA=1 表示資料的取樣發生在時脈的第二邊緣（trailing edge），所以在模式 3 下，資料取樣是發生在時脈的上升邊緣，MISO 與 MOSI 上的資料在 SCLK=H 時必須保持穩定不變，在 SCLK=L 時才允許改變。

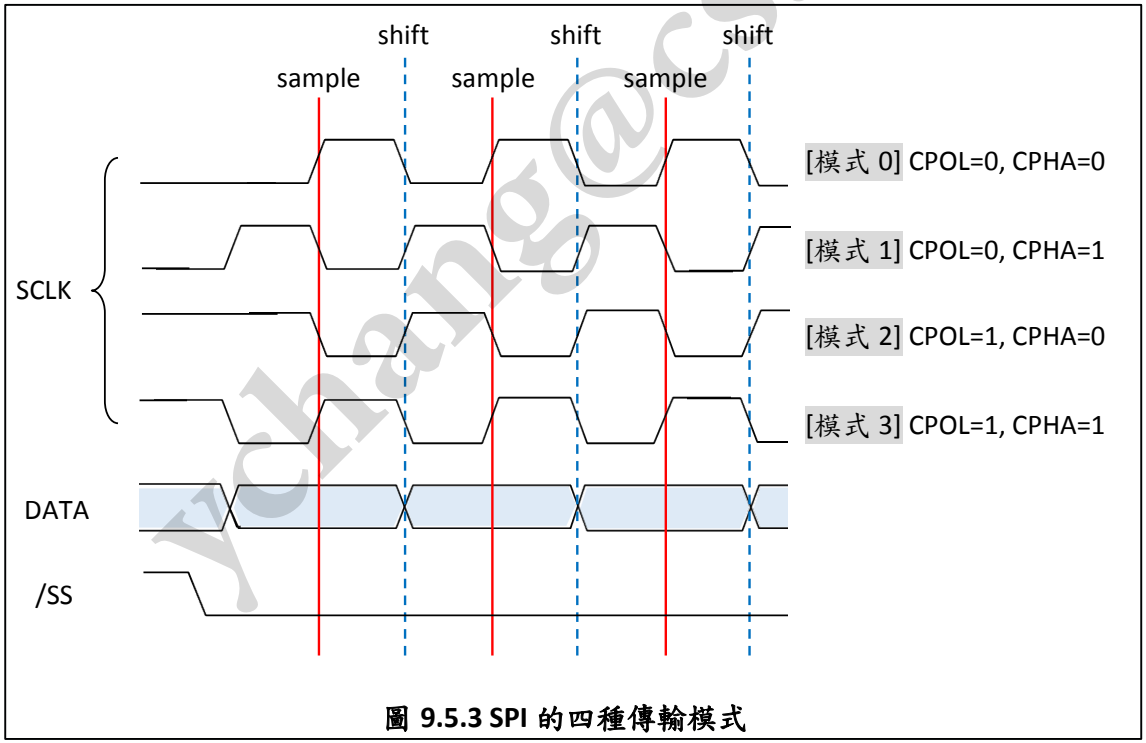


圖 9.5.3 SPI 的四種傳輸模式

8-3-4 SPI串列通訊函式庫

為簡化 SPI 串列傳輸的使用，Arduino 提供了 SPI 函式庫，讓使用者可以很容易的利用 SPI 介面達到資料傳輸的目的，這裡要特別注意，由於 SPI 的傳輸協定較為寬鬆，造成每個設備的通訊方式都有不同的差異，所以在撰寫 SPI 程式碼之前必須要仔細查閱通訊設備的資料手冊，確定下列重要的傳輸參數：(1) 資料傳輸的速率？(2)資料傳輸的順序是先送高位元還是低位元？(3)使用哪一種傳輸模式？也就是時脈閒置時為高或低電位？資料取樣是在時脈的前緣或後緣？

1. SPI.begin( )

【描述】初始化 SPI 的硬體設定並設成 master 模式，其動作是先將 SCLK，MOSI，SS 接腳模式設定成 OUTPUT，然後將 SCLK 與 MOSI 的訊號拉降為 LOW，SS 拉升到 HIGH。

【語法】SPI.begin()

【參數】無

【傳回值】無

8-3-4 SPI串列通訊函式庫

1. SPI.end( )

【描述】停止使用 SPI 並取消硬體設定，但其接腳模式維持不變。

【語法】SPI.end()

【參數】無

【傳回值】無

1. SPI.endTransaction( )

【描述】結束使用 SPI 匯流排，可改用其他的傳輸參數重新設定繼續使用。

【語法】SPI.endTransaction( )

【參數】無

【傳回值】無

8-3-4 SPI串列通訊函式庫

1. SPI.beginTransaction( )

【描述】使用定義的參數來初始化 SPI 匯流排。

【語法】SPI.beginTransaction(SPISettings(speedMaximum, dataOrder, dataMode))

【參數】SPISettings 物件的宣告是用來設定 SPI 在資料傳輸時的相關參數，其中包含了 speedMaximum，dataOrder，和 dataMode 三個重要參數，分別說明如下：

speedMaximum：表示 SPI 最大傳輸速度。例如裝置上的 SPI 晶片，若其時脈頻率最高可達 20MHz，此參數就可設為 20000000。

dataOrder：資料傳輸順序，有 MSBFIRST 與 LSBFIRST 二種，MSBFIRST 表示 MSB 先傳，而 LSBFIRST 則表示 LSB 先傳。一般而言 SPI 都是採用 MSBFIRST 的傳輸方式，但為求正確的設定，還是要以裝置的資料手冊為準。

dataMode：即為 9.5.3 所介紹的 SPI 傳輸模式，共有四種，分別為 SPI_MODE0，SPI_MODE1 SPI_MODE2，SPI_MODE3。

在早期的版本，speedMaximum，dataOrder，和 dataMode 這三個參數都可以個別的使用 setClockDivider()，setBitOrder()，和 setDataMode()來設定，但是現在已經一律改成 SPISettings 搭配 SPI.beginTransaction() 函式的設定方式。

【傳回值】無

8-3-4 SPI串列通訊函式庫

1. SPI.transfer( ) / SPI.transfer16( )

【描述】執行 SPI 的資料傳輸，要特別注意 SPI 的資料傳輸是傳送與接收雙向的動作同時執行，所以不管是傳送或接收的動作都是使用此函式。

【語法】RxVal = SPI.transfer(TxVal)

RxVal16 = SPI.transfer16(TxVal16)

SPI.transfer(buffer, size)

【參數】

TxVal：要傳送的數值，資料型別為 byte (8-bit)。

TxVal16：要傳送的 16-bit 的數值。

buffer：要傳送的資料陣列，其單元的資料型別為 byte。

size：要傳送的 byte 數。

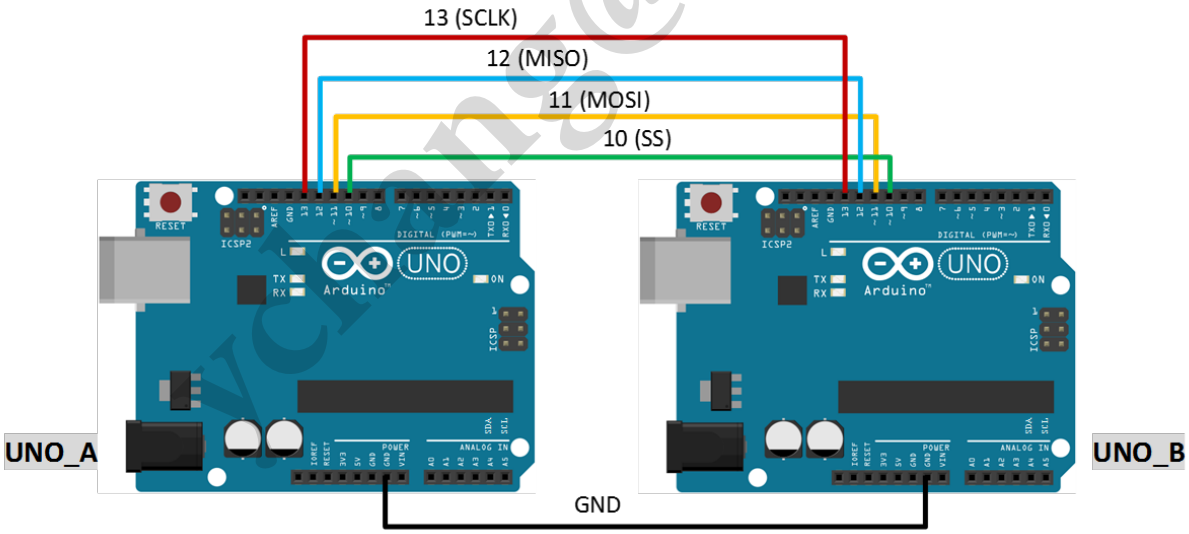
【傳回值】傳回接收到的數值，長度為 8-bit 或 16-bit。

範例 8-5

【範例】使用 SPI 完成二塊 UNO 開發板之間的通訊

【說明】

如圖 9.5.4 將二塊 UNO 開發板的 10 (SS)，11 (MOSI)，12 (MISO)與 13 (SCLK) 各自對接，即可建立 SPI 匯流排的連線，其中要特別注意二塊板子的 GND 一定要接在一起共用，如此 SPI 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，二個微控器的程式碼完全不同，要各自上傳執行。



範例 8-5

Master

```
#include <SPI.h>

void setup (void) {
    Serial.begin(115200); //set baud rate to 115200 for usart
    digitalWrite(SS, HIGH); // disable Slave Select
    SPI.begin ();
    SPI.setClockDivider(SPI_CLOCK_DIV8);//divide the clock by 8
}

void loop (void) {
    char c;
    digitalWrite(SS, LOW); // enable Slave Select
    // send test string
    for (const char * p = "Hello, world!\r" ; c = *p; p++) {
        SPI.transfer (c);
        Serial.print(c);
    }
    digitalWrite(SS, HIGH); // disable Slave Select
    delay(2000);
}
```

Slave

```
#include <SPI.h>
char buff [50];
volatile byte indx;
volatile boolean process;

void setup (void) {
    Serial.begin (115200);
    pinMode(MISO, OUTPUT); // have to send on master in so it set as output
    SPCR |= _BV(SPE); // turn on SPI in slave mode
    indx = 0; // buffer empty
    process = false;
    SPI.attachInterrupt(); // turn on interrupt
}
ISR (SPI_STC_vect) // SPI interrupt routine {
    byte c = SPDR; // read byte from SPI Data Register
    if (indx < sizeof buff) {
        buff [indx++] = c; // save data in the next index in the array buff
        if (c == '\r') //check for the end of the word
        process = true;
    }
}

void loop (void) {
    if (process) {
        process = false; //reset the process
        Serial.println (buff); //print the array on serial monitor
        indx= 0; //reset button to zero
    }
}
```