SPI 功能使用方法

1 適用產品：

1.1 SM59R16A2/ SM59R08A2

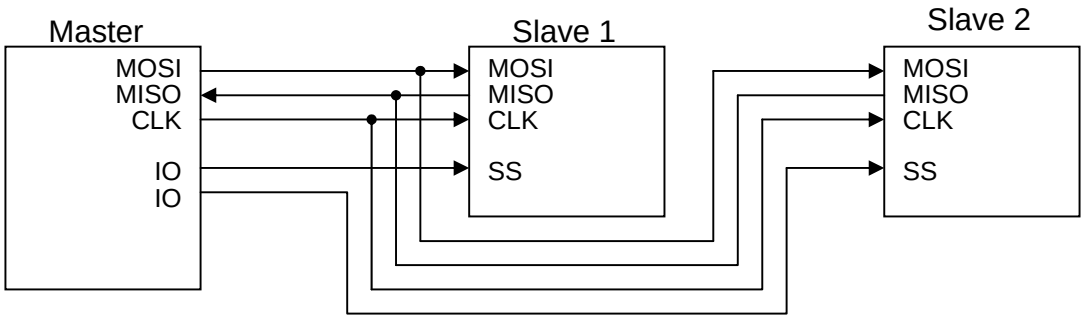
1.2 SM59R16A5/ SM59R09A5/SM59R05A5/SM59R16A3/ SM59R09A3 /SM59R05A3

1.3 SM59R04A2/ SM59R04A1/ SM59R03A1/ SM59R02A1

2 SPI 使用概述：

SPI 通信使用 4 個引腳，分別爲：

- SPI_MOSI: 做爲master時資料輸出；做爲slave時資料輸入
- SPI_MISO: 做爲master時資料輸入；做爲slave時資料輸出
- SPI_CLK: SPI的時脈信號由master主控產生；資料 (輸出及輸入) 和時脈同步
- SPI_SS: 此引腳唯有做爲slave mode時有做用；做爲master mode，此引腳當做GPIO使用
  = 0: master致能slave
  = 1: master禁能slave



3 P4SPI (SPI function pin assignment)說明：

	P4SPI (SPI function pin assignment)
SM59R16A5/SM59R16A3	○
SM59R09A5/SM59R09A3	○
SM59R05A5/SM59R05A3	○
SM59R16A2	×
SM59R08A2	×
SM59R04A2/SM59R04A1	○
SM59R03A1	○
SM59R02A1	○

○：表示該型號可以使用 P4SPI 功能

x：表示該型號不可使用 P4SPI 功能

Notice：所有系列的 Package type DIP-40 沒有 P4，所以沒有用此功能

P4SPI (SPI function pin assign)說明

Mnemonic	Description	Direct	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	RESET
Serial interface 0 and 1											
AUX	Auxiliary register	91h	BRS	P4CC	***P4SPI***	P4UR1	P4IIC	P0KBI	-	DPS	00H

內建SPI，可直接經由軟體設定指定其腳位至P1或P4，避免和其它特殊功能腳位重複，並提高硬體規劃的相容性。

P4SPI: P4SPI = 0 – SPI function on P1.
SS – P1.4
MOSI – P1.5
MISO – P1.6
CLK – P1.7
P4SPI = 1 – SPI function on P4.
SS – P4.0
MOSI – P4.1
MISO – P4.2
CLK – P4.3

4 SPI 相關的特殊暫存器 SPI Special Function Register (SFR)

SPI	Description	Direct	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	RESET
SPI function											
AUX	Auxiliary register	91h	BRS	P4CC	P4SPI	P4UR1	P4IIC	P0KBI	-	DPS	00H
SPIC1	SPI control register 1	F1h	SPIEN	SPIMSS	SPISSP	SPICKP	SPICKE	SPIBR[2:0]			08H
SPIC2	SPI control register 2	F2h	SPIFD	TBC[2:0]			-	RBC[2:0]			00H
SPIS	SPI status register	F5h	-	SPIMLS	SPIOV	SPITXIF	SPITDR	SPIRXIF	SPIRDR	SPIRS	40H
SPITXD	SPI transmit data buffer	F3h	SPITXD[7:0]								00H
SPIRXD	SPI receive data buffer	F4h	SPIRXD[7:0]								00H

Mnemonic: AUX **Address: 91h**

7	6	5	4	3	2	1	0	Reset
BRS	P4CC	P4SPI	P4UR1	P4IIC	P0KBI	-	DPS	00H

P4SPI: P4SPI = 0 – SPI function on P1.
P4SPI = 1 – SPI function on P4.

P4SPI setting	SPI_SS	SPI_MOSI	SPI_MISO	SPI_CLK
0	P1.4	P1.5	P1.6	P1.7
1	P4.0	P4.1	P4.2	P4.3

Mnemonic: SPIC1 **Address: F1h**

7	6	5	4	3	2 1 0	Reset
SPIEN	SPIMSS	SPISSP	SPICKP	SPICKE	SPIBR[2:0]	08h

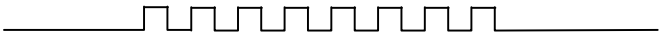
SPIEN: SPI 模組致能旗標：
"1" – 致能
"0" – 禁能

SPIMSS: 主從模式選擇旗標（Master or Slave mode Select）
"1" – MCU 做爲 Master mode.
"0" – MCU 做爲 Slave mode.

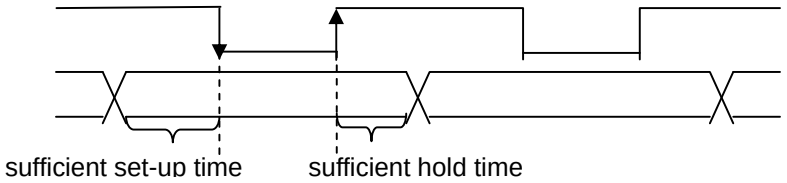
SPISSP: (SS)引腳致能狀態旗標；當 MCU 爲 slave 時，可由旗標設定 Slave Select (SS)引腳致能狀態 (slave mode used only)
"1" – 高準位致能 high active.
"0" – 低準位致能 low active.

SPICKP: 時脈閒置準位旗標(master mode used only)
"1" – 時脈信號閒置時爲高準位(SCK high during idle), Ex :

"0" – 時脈信號閒置時爲低準位(SCK high during idle), Ex :



SPICKE: 時脈取樣旗標 Clock sample edge select.
"1" – 正緣取樣 data latch in rising edge
"0" – 負緣取樣 data latch in falling edge.
* 爲確保資料取樣的正確性，無論使用正緣或負緣取樣，時脈及資料同步時皆需有足夠的準備時間 (set-up time)及保持時間(hold time)，時序產生如下圖：



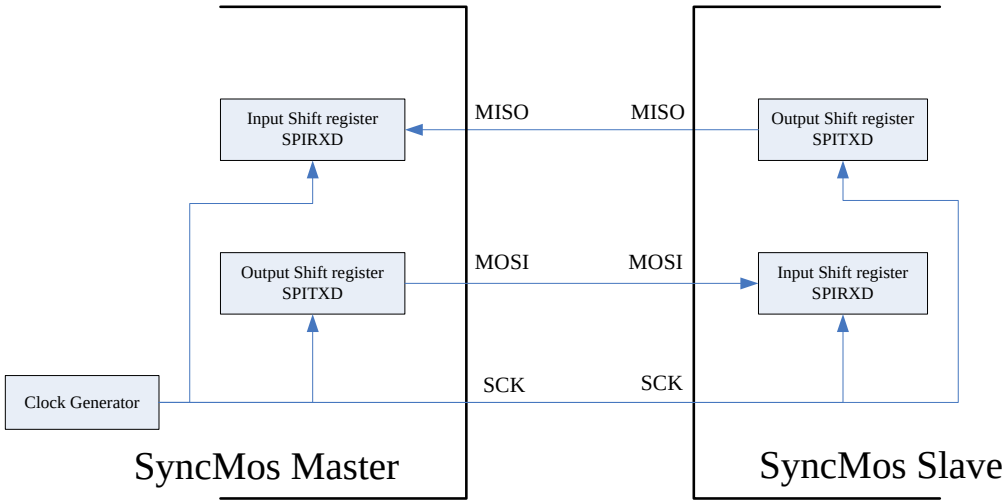
SPIBR[2:0]: SPI 鮑率選擇(master mode used only), Fosc 爲晶振頻率：

SPIBR[2:0]	Baud rate
0:0:0	Fosc/4
0:0:1	Fosc/8
0:1:0	Fosc/16
0:1:1	Fosc/32
1:0:0	Fosc/64
1:0:1	Fosc/128
1:1:0	Fosc/256
1:1:1	Fosc/512

Mnemonic: SPIC2 **Address: F2h**

7	6	5	4	3	2	1	0	Reset
SPIFD	TBC[2:0]			-	RBC[2:0]			00h

SPIFD: 全雙工模式致能旗標(Full-duplex mode enable)
"1"： 全雙工模式致能
"0"： 全雙工模式禁能
當該旗標致能時，TBC[2:0]和RBC[2:0]會被清除並保持爲零，SPI全雙工模式僅允許8位元通訊，Master透過MOSI引腳做資料輸出，slave時透過MISO回傳資料，SPI的時脈信號由master主控產生；所有資料 (輸出及輸入) 皆和時脈同步。



TBC[2:0]: SPI 傳送元位計數旗標(SPI transmitter bit counter)
可設定 1~8 位元通訊，但全雙工模式僅允許 8 位元通訊。

TBC[2:0]	Bit counter
0:0:0	8 bits output
0:0:1	1 bit output
0:1:0	2 bits output
0:1:1	3 bits output
1:0:0	4 bits output
1:0:1	5 bits output
1:1:0	6 bits output
1:1:1	7 bits output

RBC[2:0]: SPI 接收元位計數旗標(SPI receiver bit counter)
可設定 1~8 位元通訊，但全雙工模式僅允許 8 位元通訊。

RBC[2:0]	Bit counter
0:0:0	8 bits input
0:0:1	1 bit input
0:1:0	2 bits input
0:1:1	3 bits input
1:0:0	4 bits input
1:0:1	5 bits input
1:1:0	6 bits input
1:1:1	7 bits input

Mnemonic: SPIS **Address: F5h**

7	6	5	4	3	2	1	0	Reset
-	SPIMLS	SPIOV	SPITXIF	SPITDR	SPIRXIF	SPIRDR	SPIRS	40h

SPIMLS: MSB or LSB output /input first
"1"： 最高位元先傳送 (MSB output/input first)。
"0"： 最低位元先傳送 (LSB output/input first)。

SPIOV: 溢位旗標(Overflow flag)
"1"： 當 SPIRDR 已設定（SPIRXD 原有資料未被讀取）且下一筆資料正寫入 SPIRXD 時，SPIOV 將被設定爲"1"，告知 SPIRXD 資料以有損毀。
"0"： 當 SPIRDR 清爲零時，SPIOV 則由硬體清除。

SPITXIF: 傳送中斷旗標(Transmit Interrupt Flag)
"1"： 當 SPITXD 的資料已載入移位暫存器，由硬體設定爲"1" 。
"0"： 傳送資料完成後必須由軟體清除。

SPITDR: 資料傳送位元(Transmit Data Ready)
"1": 當程序爲傳送模式時，資料儲存至 SPITXD 後，由軟體設定此旗標爲"1" ，告知 SPI module 允許傳出資料。
"0": 當 SPI module 由 SPITXD 完成傳送時(或 SPITXD 被載至移位暫存器時)，此旗標則由硬體自動清除。

SPIRXIF: 接收中斷旗標(Receive Interrupt Flag)
"1"： 當 SPIRXD 被載入新一筆資料後，由硬體設定爲"1" 。
"0"： 接收資料完成後必須由軟體清除。

SPIRDR: 資料接收位元(Receive Data Ready)
"1" : SPI module接收資料時，SPIRDR由硬體自動設定爲"1"，以告知MCU完成接收並儲存至SPIRXD；當新的一筆資料寫入SPIRXD，而SPIRDR未清除時，SPIRXD原有的資料將被覆寫，產生overflow
"0"： 由 SPIRXD 讀取資料後，必須由軟體清除此旗標。

SPIRS: 接收開始位元(Receive Start)
"1"： 由軟件設爲"1"，告知 SPI 模組 SPIRXD 開始接收資料(即 SPI_CLK 開始送 clock)。
"0"： 當資料接收完成，由硬體清爲"0"

Mnemonic: SPITXD **Address: F3h**

7	6	5	4	3	2	1	0	Reset
SPITXD[7:0]								00h

SPITXD[7:0]: 傳送資料緩衝器(Transmit data buffer)

Mnemonic: SPIRXD **Address: F4h**

7	6	5	4	3	2	1	0	Reset
SPIRXD[7:0]								00h

SPIRXD[7:0]: 接收資料緩衝器(Receive data buffer)

5 SPI 中斷

5.1 向量表(Interrupt vectors table)

Interrupt Request Flags	Interrupt Vector Address	Interrupt Number *(use Keil C Tool)
IE0 – External interrupt 0	0003h	0
TF0 – Timer 0 interrupt	000Bh	1
IE1 – External interrupt 1	0013h	2
TF1 – Timer 1 interrupt	001Bh	3
RI0/TI0 – Serial channel 0 interrupt	0023h	4
TF2/EXF2 – Timer 2 interrupt	002Bh	5
PWMIF – PWM interrupt	0043h	8
SPIIF – SPI interrupt	***004Bh***	***9***
ADCIF – A/D converter interrupt	0053h	10
KBIIF – keyboard Interface interrupt	005Bh	11
LVIIF – Low Voltage Interrupt	0063h	12
IICIF – IIC interrupt	006Bh	13
RI1/TI1 – Serial channel 1 interrupt	0083h	16

*See Keil C about C51 User's Guide about Interrupt Function description

5.2 中斷相關暫存器(Interrupt SFR)

Mnemonic	Description	Direct	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	RESET
Interrupt											
IEN0	Interrupt Enable 0 register	A8h	EA	-	ET2	ES0	ET1	EX1	ET0	EX0	00h
IEN1	Interrupt Enable 1 register	B8h	EXEN2	-	IEIIC	IELVI	IEKBI	IEADC	IESPI	IEPWM	00h
IEN2	Interrupt Enable 2 register	9Ah	-	-	-	-	-	-	-	ES1	00h
IRCON	Interrupt request register	C0H	EXF2	TF2	IICIF	LVIIF	KBIIF	ADCIF	SPIIF	PWMIF	00H
IP0	Interrupt priority level 0	A9h	-	-	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0	00h
IP1	Interrupt priority level 1	B9h	-	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0	00h

***如果需要使用中斷程序，可參考以下設置：**

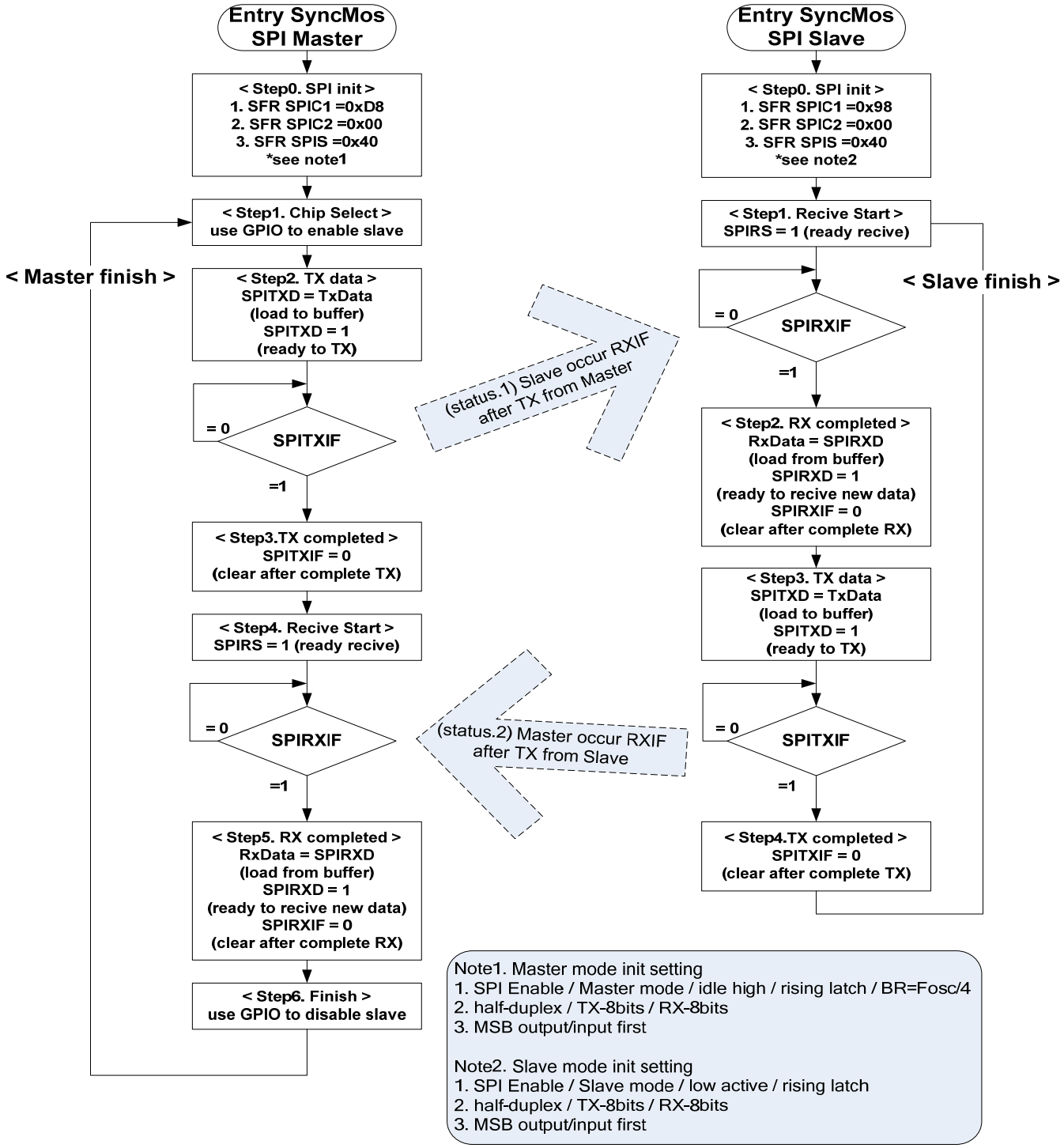
(1) SPI 中斷致能設置：

```
IEN0  |= 0x80;                    //Enable interrupt All
IEN1 |= 0x02;                   //Enable interrupt SPI
```

(2) SPI 中斷程序表示：

```
  void SPI_interrupt(void) interrupt 9
  {
    //IRCON_SPIIF = 0;              //Clear interrupt flag
    If(SPIS&=0x04)
    {
        SPIS &= (~0x04);    //Clear TXIF bit
    }
    {
        SPIS &= (~0x20;     //Clear RXIF bit
    }
```

6 以下爲兩顆 MCU 分別當做 SPI master 及 slave 通訊的流程圖



7 SPI master 及 slave 通訊的範例程式

7.1 Master 傳資料(0xFF~0x00)至 Slave

7.2 Slave 接受後，以相同的資料回傳給 Master

7.3 Master 接受 Slave 回傳的資料後，做資料比較，相同則表示通訊正確，不相同則反之

Description	Master:
Main program	

```
//====================================================================
//
//              S Y N C M O S   T E C H N O L O G Y
//
//====================================================================

#include  "..\h\SM59R16A2.h"
#include  "..\h\SM59R16A2_extradef.h"
#include  "..\LCD\LCD16x2.h"
#include "..\MISC\delay.h"

#define Control_Byte    0xA0
#define SPI_SS          P1_0
#define SPITDR=1        SPIS  |= 0x08

/* void SPI_interrupt(void) interrupt 9
{
    IRCON_SPIIF = 0;                //Clear interrupt flag
}*/

void SPI_Init(void)
{
    SPIC1 = 0xD8;   //SPI Enable/ master/ NC/ idle hi/ rising latch
    //SPIC1 |= 7;   //BR=(Fosc/512)
    SPIC2 = 0x00;   //SPI full-duplex disable /8-bit communicate
    SPIS  = 0x40;   //MSB send first
}

void SPI_TX( unsigned char DATA)
{
    SPITXD  = DATA;                 //Load to Buffer
    SPIS    |= 0x08;                //SPITDR
    while( !(SPIS&SPI_TXIF) );      //TX completed
    SPIS        &= (~SPI_TXIF);     //Clear TXIF bit
}

unsigned char SPI_RX(void)
{
    unsigned char Temp;
    SPIS    |= 0x01;                //Receive Start
    while( !(SPIS&SPI_RXIF) );      //RX completed
    Temp    = SPIRXD;               //Load from Buffer
    SPIS    |= 0x02;                //Receive Data Ready
    SPIS    &= (~SPI_RXIF);         //Clear RXIF bit
    return Temp;
}

void main()
{
    unsigned long Err =0, counter =1;
    unsigned char TxData, RxData,j;

    LCD_Init();     Delay10mSec(1); //must wait for LCD stable
    SPI_Init();
```

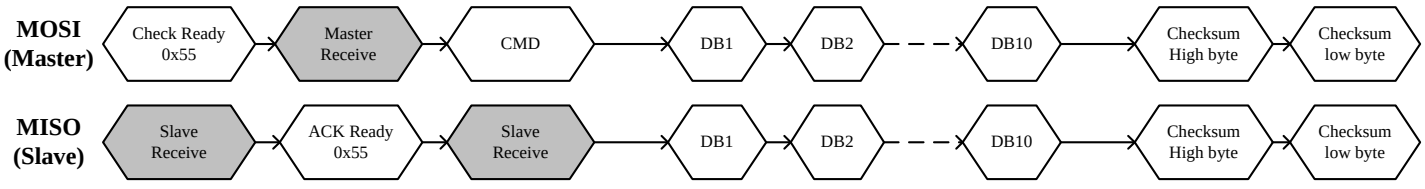
```
    while(1)
    {
        for(TxData=0xFF; TxData>0; TxData--)
        {
            PrintLcdStrLX( 1, 0, "M_TX:          ");
            PrintLcdStrLX( 2, 0, "M_RX:          ");

            SPI_SS = 0;              //CS
            SPI_TX( TxData );            //TX
            //for(j=0; j<0; j++);           //Dealy time
            RxData = SPI_RX();           //RX
            SPI_SS = 1;              //CS

            SetCursorAddr(1, 6);     PrintLcdHex( TxData );
            SetCursorAddr(2, 6);     PrintLcdHex( RxData );

            counter++;
            SetCursorAddr(1, 12);    PrintLcdDec( counter );
            if(TxData!=RxData)       Err++;
            SetCursorAddr(2, 12);    PrintLcdDec( Err );
            P2=TxData;           //result
        }
    }
}
```

Description	Slave:
Main program	

```
//====================================================================
//
//               S Y N C M O S   T E C H N O L O G Y
//
//====================================================================

#include  "..\h\SM59R16A2.h"
#include  "..\h\SM59R16A2_extradef.h"
#include  "..\LCD\LCD16x2.h"
#include "..\MISC\delay.h"

void SPI_Init(void)
{
    SPIC1 =0x98;    //SPI Enable/ slave/ Low active / NC / rising latch/
NC NC NC
    SPIC2 =0x00;    //SPI full-duplex disable /8-bit communicate
    SPIS  =0x40;//MSB send first
}

void SPI_TX( unsigned char DATA)
{
    SPITXD  = DATA;              //Load to Buffer
    SPIS    |= 0x08;                 //SPITDR
    while( !(SPIS&SPI_TXIF) );       //TX completed
    SPIS        &= (~SPI_TXIF);      //Clear TXIF bit
```

```
}

unsigned char SPI_RX(void)
{
    unsigned char Temp;
    SPIS    |= 0x01;                //Receive Start
    while( !(SPIS&SPI_RXIF) );      //RX completed
    Temp    = SPIRXD;           //Load from Buffer
    SPIS    |= 0x02;                //SPIRDR
    SPIS    &= (~SPI_RXIF);         //Clear RXIF bit
    return Temp;
}

void main()
{
    unsigned long Err =0, counter =1;
    unsigned char TxData=0, RxData=0,j;

    LCD_Init();         Delay10mSec(1); //must wait for LCD stable
    SPI_Init();

    while(1)
    {
            RxData = SPI_RX();  //RX
            SPI_TX( RxData );   //TX
            P2 = RxData;        //result
    }
}
```

8 SPI 全雙工說明，使用兩顆 SM59R04A2 分別當做 SPI master 及 slave 做通訊，以下是範例程序的通訊協定：



9 SPI master 及 slave 全雙工通訊範例程式

9.1 Master 不斷送 Ready Byte(0x55)，直到 Slave 同樣回傳 Ready Byte (0x55)做確認

9.2 待兩方確認後，Master 會送 command byte(0x0A 或 0x0B)

9.3 兩方資料長度定爲 10-Byte

9.4 Master 送出資料固定爲 0x00~0x09

9.5 Slave 送出的資料則依據收到 Master 的 command byte 有所不同

9.5.1 當command byte=0x0A，slave送出資料爲0x00~0x09

9.5.2 當command byte=0x0B，slave送出資料爲0x10~0x19

9.6 資料送出完成後，master 及 slave 會互傳該次所接收到的資料的 checksum

Description	Master:
Main program	

```
//=====================================================================
//
//               S Y N C M O S   T E C H N O L O G Y
//
//=====================================================================
// SM59R04A2 SPI Master full-duplex ,8-bit communication
// System clock = 22.1184 MHz
//=====================================================================
#include  "SM59R04A2.h"

#define SPI_RxStart     0x01
#define SPI_RDR         0x02
#define SPI_RXIF        0x04
#define SPI_TDR         0x08
#define SPI_TXIF        0x10
#define SPI_Overflow    0x20
#define SPI_MSB_LSB     0x40
#define CMD_0A          0x0A
#define CMD_0B          0x0B
#define CMD_Ready       0x55
#define finetune        15                  // for timing finetune
#define SPI_SS          P1_0                // slave select

bit SPI_Intrrupt=1;
unsigned char temx;

void Delay1mSec(unsigned int NTime) ;
void Delay_tune(unsigned char tune);
void SPI_Init(void);
unsigned char SPI_RX(void);
void SPI_TX( unsigned char DATA);
unsigned char SPI_full_duplex(unsigned char DATA);
void SPI_communication(unsigned char command);

void SPI_interrupt(void) interrupt 9
{
    while(!((SPIS & SPI_RDR)==SPI_RDR));    // RDR set 1 by H/W
    while((SPIS & SPI_TDR)==SPI_TDR);       // TDR clr 0 by H/W
    SPIS &= 0xE9;                           // Clear SPITXIF & SPIRXIF & SPIRDR

    SPI_Intrrupt = 0;
}

//=====================================================================
//Freq = 22.1184 MHz,  @ 64-pin demo board
//=====================================================================
void Delay1mSec(unsigned int NTime)
{
```

```
    unsigned int i, j;

    for(i=0; i<NTime; i++)
        for(j=0; j<1300; j++);
}

void Delay_tune(unsigned char tune)
{
    unsigned char i;
    for(i=0; i<tune; i++);
}

void SPI_Init(void)
{
    SPIC1 = 0xD8;       //SPI Enable/ master/ NC/ idle hi/ rising latch
    SPIC2 = 0x80;       //SPI full-duplex ,8-bit communication only
    //SPIC2 = 0x00;     //SPI half-duplex ,8-bit communication
    SPIS  = 0x40;       //MSB send first
    SPIS |= 0x01;       // RX Start
    //SPIS  &= 0xFE;// Receive Stop

    //SPIC1 |= 0;       //BR=(Fosc/4)
    //SPIC1 |= 1;       //BR=(Fosc/8)
    //SPIC1 |= 2;       //BR=(Fosc/16)
    SPIC1 |= 3;     //BR=(Fosc/32)
    //SPIC1 |= 4;       //BR=(Fosc/64)
    //SPIC1 |= 5;       //BR=(Fosc/128)
    //SPIC1 |= 6;       //BR=(Fosc/256)
    //SPIC1 |= 7;       //BR=(Fosc/512)

//  IEN0    |= 0x80;    // EA=1
//  IEN1    |= 0x02;    // IESPI=1
}

unsigned char SPI_RX(void)
{
    SPI_SS = 0;
    SPITXD = 0xFF;                      // clear SPITXD, useless
    SPIS |= 0x01;                       // RX Start // send SPI CLK
//  SPI_Intrrupt = 1;
//  while(SPI_Intrrupt);
    while(!((SPIS & SPI_RDR)==SPI_RDR));// RDR set 1 by H/W
    SPIS &= 0xE9;                       // Clear SPITXIF & SPIRXIF & SPIRDR

    temx = SPIRXD;
    SPI_SS = 1;
    Delay_tune(finetune);               // CS, must wait slave
    return temx;
}

void SPI_TX( unsigned char DATA)
{
    SPI_SS = 0;
    SPITXD = DATA;                      // Load to TX Buffer
    SPIS  |= SPI_TDR;                   // set SPITDR, TX proceed
//  SPI_Intrrupt = 1;
//  while(SPI_Intrrupt);
    while((SPIS & SPI_TDR)==SPI_TDR);   // TDR clr 0 by H/W
    SPIS &= 0xE9;                       // Clear SPITXIF & SPIRXIF & SPIRDR

    SPI_SS = 1;
    Delay_tune(finetune);               // CS, must wait slave
}
```

```
unsigned char SPI_full_duplex(unsigned char DATA)
{
    SPI_SS = 0;
    SPITXD = DATA;                      // Load to TX Buffer
    SPIS  |= SPI_TDR;                   // set SPITDR, TX proceed
//  SPI_Intrrupt = 1;
//  while(SPI_Intrrupt);
    while(!((SPIS & SPI_RDR)==SPI_RDR));// RDR set 1 by H/W
    while((SPIS & SPI_TDR)==SPI_TDR);   // TDR clr 0 by H/W
    SPIS &= 0xE9;                       // Clear SPITXIF & SPIRXIF & SPIRDR

    temx = SPIRXD;
    SPI_SS = 1;
    Delay_tune(finetune);       // CS, must wait slave
    return temx;
}

void SPI_communication(unsigned char command)
{
    unsigned char RxData_H=0, RxData_L=0, TxData=0, loop=0, temp=0, ready=0;
    unsigned int checksum=0;

    SPI_TX(CMD_Ready);          // check  slaver ready, send 0x55
    ready = SPI_RX();           // receive ready_byte,  wait 0x55

    if(ready == CMD_Ready)
    {
        SPI_TX(command);    checksum=0;     // send command 0x0A or 0x0B
        for(loop=0x00; loop<0x10; loop++)
        {
            checksum += SPI_full_duplex(loop);
        }
        RxData_H = SPI_full_duplex((unsigned char)(checksum>>8));   // send
checksum high byte
        RxData_L = SPI_full_duplex((unsigned char)checksum);        // send
checksum low byte
        ready=0;
    }
    Delay_tune(50);             // useless
}

void main()
{
    Delay1mSec(100);            // wait system stable
    //SPI_Intrrupt = 1;         //
    IFCON |= 0x80;              // set 1T
    SPI_Init();                 //
    Delay1mSec(1);              // must wait SPI init

    while(1)
    {
        SPI_communication(CMD_0A);
        SPI_communication(CMD_0B);
    }
}
```

Description	Slave:
Main program	//========== // // S Y N C M O S T E C H N O L O G Y // //========== // SM59R04A2 SPI slave full-duplex ,8-bit communication

```
// System clock = 22.1184 MHz
//================================================================================
#include  "SM59R04A2.h"

#define SPI_RxStart     0x01
#define SPI_RDR         0x02
#define SPI_RXIF        0x04
#define SPI_TDR         0x08
#define SPI_TXIF        0x10
#define SPI_Overflow    0x20
#define SPI_MSB_LSB     0x40
#define CMD_0A          0x0A
#define CMD_0B          0x0B
#define CMD_Ready       0x55

unsigned char TxData=0x10, RxData=0, RxBuf[5];
unsigned int loop=0, checksum=0;
unsigned char RxData_H=0, RxData_L=0;
unsigned char cmd=0, ready=0;

void Delay1mSec(unsigned int NTime);
void Delay_tune(unsigned char tune);
void Init_SPI(void);
void SPI_TX( unsigned char DATA);
unsigned char SPI_RX(void);

void Delay1mSec(unsigned int NTime)
{
  unsigned int i, j;

  for(i=0; i<NTime; i++)
     for(j=0; j<1300; j++);
}

void Init_SPI(void)
{
    SPIC1   =  0x98; //SPI Enable/ slave/ Low active / NC / rising latch
    SPIC2   =  0x80; //SPI full-duplex ,8-bit communication only
    //SPIC2 =  0x00; //SPI half-duplex ,8-bit communication
    SPIS    =  0x40; //MSB send first
    SPIS    |= 0x01;    //Receive Start
    SPITXD  =  0xFF;

    IEN0    |= 0x80;    // EA=1
    IEN1    |= 0x02;    // IESPI=1
}

void SPI_TX( unsigned char DATA)
{
    SPITXD = DATA;              // Load to TX Buffer
    SPIS  |= SPI_TDR;           // set SPITDR, TX proceed
    while((SPIS & SPI_TDR)==SPI_TDR);       // 1:Loading ; 0: finish
    while(!((SPIS & SPI_TXIF)==SPI_TXIF));  // TX completed
    SPIS    &= 0xE9;                        // Clear SPITXIF & SPIRXIF & SPIRDR
    //SPIS &= (~SPI_TXIF);      // Clear TXIF bit
}

unsigned char SPI_RX(void)
{
    unsigned char Temp;
    SPITXD = 0xFF;              // clear SPITXD, useless
//  SPIS    |= 0x01;            // Receive Start
    while(!((SPIS & SPI_RDR)==SPI_RDR));    // RX receive
    while(!((SPIS & SPI_RXIF)==SPI_RXIF));  // RX completed
```

```
    Temp    =  SPIRXD;          // Load from Buffer
    SPIS    &= 0xE9;            // Clear SPITXIF & SPIRXIF & SPIRDR
//  SPIS    &= (~SPI_RDR);      // RX Data Ready
//  SPIS    &= (~SPI_RXIF);     // Clear RXIF bit
//  SPIS    &= 0xFE;            // Receive Stop
    return Temp;
}

unsigned char SPI_full_duplex(unsigned char DATA)
{

    SPITXD = DATA;                      // Load to TX Buffer
    SPIS  |= SPI_TDR;                   // set SPITDR, TX proceed
    while((SPIS & SPI_TDR)==SPI_TDR);   // TDR clr 0 by H/W
    while(!((SPIS & SPI_RDR)==SPI_RDR));// RDR set 1 by H/W
    SPIS    &= (~SPI_RDR);              // RX Data Ready
    return SPIRXD;
}

void SPI_ISR(void) interrupt 9
{
    while(!((SPIS & SPI_RDR)==SPI_RDR));// RDR set 1 by H/W
    ready  = SPIRXD;                // SPI cmd receive ready
    SPIS &= 0xE9;                       // Clear SPITXIF & SPIRXIF & SPIRDR

    if(ready==CMD_Ready)
    {
        SPI_TX(CMD_Ready);  checksum = 0;
        cmd = SPI_RX();

        switch (cmd)
        {
            case CMD_0A:                // When cmd = 0x0A
                for(loop=0x00; loop<0x10; loop++)
                {
                    checksum+= SPI_full_duplex(loop);
                }

                break;
            case CMD_0B:                // When cmd = 0x0B
                for(loop=0x10; loop<0x20; loop++)
                {
                    checksum+= SPI_full_duplex(loop);
                }
                break;
            default:                    // When cmd error occur
                for(loop=0; loop<3; loop++)
                {
                    checksum+= SPI_full_duplex(0xAA);
                }
                break;
        }
        RxData_H = SPI_full_duplex((unsigned char)(checksum>>8));
        RxData_L = SPI_full_duplex((unsigned char)(checksum));
    }

    SPIS    &= 0xE9;        // Clear SPITXIF & SPIRXIF & SPIRDR
    SPITXD  =  0xFF;        // clear SPITXD, useless
    //SPIIF=0;
}

void main()
{
    Delay1mSec(100);        // wait system stable
```

```
    IFCON   |= 0x80;         // set 1T
    Init_SPI();

    while(1)
    {}
}
```