SPI 功能使用方法

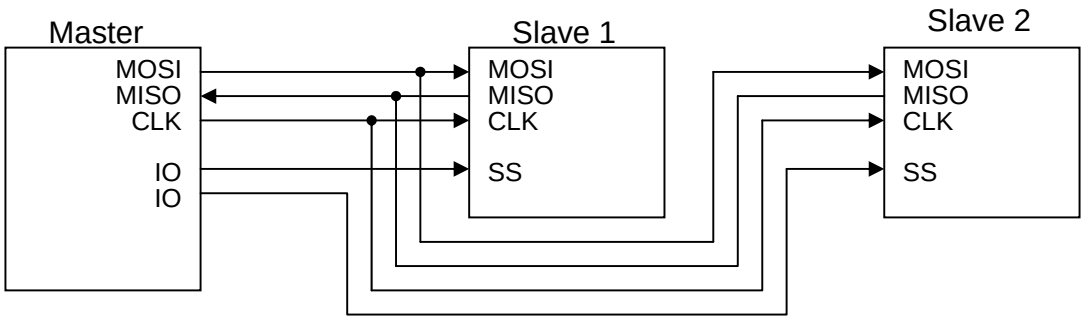
1 適用產品：

1.1 SM39R16A2/ SM39R12A2/ SM39R08A2

2 SPI 使用說明:

2.1 SPI 通信使用 4 個引腳，分別為：

- SPI_MOSI: 做為 master 時資料輸出；做為 slave 時資料輸入
- SPI_MISO: 做為 master 時資料輸入；做為 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: 此引腳唯有做為 slave mode 時有做用；做為 master mode，此引腳當做 GPIO 使用
  = 0: master致能slave
  = 1: master禁能slave



3 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	RST
SPI function											
AUX	Auxiliary register	91h	BRGS	-	***P2SPI***	P2UR	P2IIC	-	-	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]***			***SPIRST***	***RBC[2:0]***			00H
SPIS	SPI status register	F5h	***SPIRF***	***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
BRGS	-	**P2SPI**	P2UR	P2IIC	-	-	DPS	00H

P2SPI: P2SPI = 0 – SPI function on P1.
P2SPI = 1 – SPI function on P2.

P4SPI setting	SPI_SS	SPI_MOSI	SPI_MISO	SPI_CLK
0	P1.4	P1.7	P1.6	P0.0
1	P2.4	P2.2	P2.3	P2.5

Mnemonic: SPIC1 **Address: F1h**

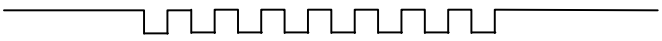
7	6	5	4	3	2	1	0	Reset
SPIEN	**SPIMSS**	**SPISSP**	**SPICKP**	**SPICKE**	**SPIBR[2:0]**			08h

SPIEN: SPI 模組致能旗標：
"1" – 致能
"0" – 禁能

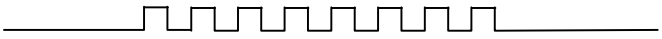
SPIMSS: 主從模式選擇旗標（Master or Slave mode Select）
"1" – MCU 做為 Master mode.
"0" – MCU 做為 Slave mode.

SPISSP: (SS)引腳致能狀態旗標；當 MCU 為 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.

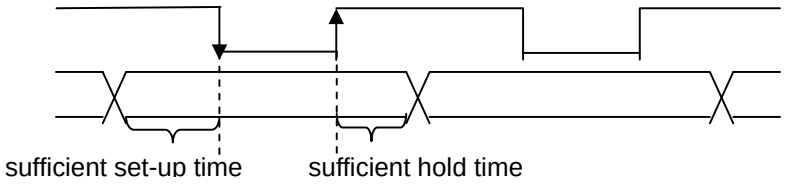
SPICKP: 時脈閒置準位旗標(master mode used only)
"1" – 時脈信號閒置時為高準位(SCK high during idle), Ex :



"0" – 時脈信號閒置時為低準位(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.
* 為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 為晶振頻率：

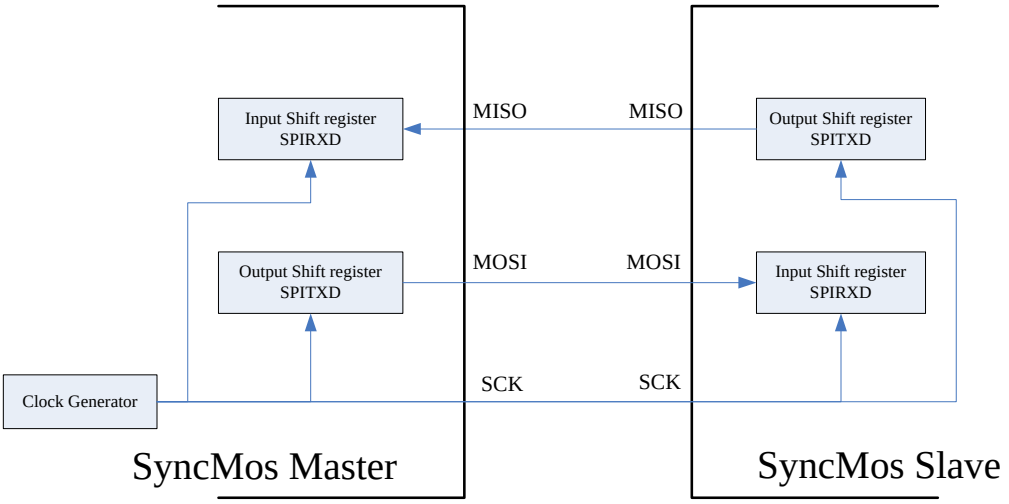
SPIBR[2:0]	Baud rate
0:0:0	Fosc/4
0:0:1	Fosc/8
0:1:0	Fosc/16
0:1:1	Fosc/32
1:0:0	Fosc/64
1:0:1	Fosc/128

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]**	**SPIRST**	**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]會被清除並保持為零，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。



SPIRST: SPI Re-start (Slave mode used only)
SPIRST=0: 重新啟動功能關閉，當SS動作時SPI模組傳送/接收數據。
SS動作期間先前在SPITXD/SPIRXD緩衝區的數據將不會被清除(Shift counter不會歸零，數據是有效的)
SPIRST=1: 重新啟動功能致能，當SS重新動作時SPI模組傳送/接收新的數據。
SS 動作期間先前在 SPITXD/SPIRXD 緩衝區的數據將會被清除(Shift counter 會歸零，數據是無效的)

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
SPIRF	SPIMLS	SPIOV	SPITXIF	SPITDR	SPIRXIF	SPIRDR	SPIRS	40h

SPIRF: SPI SS pin Release Flag.(SPI SS 釋放旗標)
當SS腳釋放(高電平)和SPIRST為"1"時，此位元被設定為1，可供Slave判斷是否已被釋放
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 將被設定為"1"，告知 SPIRXD 資料以有損毀。
"0"： 當 SPIRDR 清為零時，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，由硬體設定為"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": 當程序為傳送模式時，資料儲存至 SPITXD 後，由軟體設定此旗標為"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，由硬體設定為"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由硬體自動設定為"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"： 由軟件設為"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，由硬體清為"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)

4 SPI 中斷

4.1 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

*請參考Keil C用戶指南中的有關中斷功能使用說明

4.2 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	RST
Interrupt											
IEN1	Interrupt Enable 1 register	B8h	EXEN2	-	IEIIC	IELVI	IEKBI	IEADC	**IESPI**	IEPWM	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

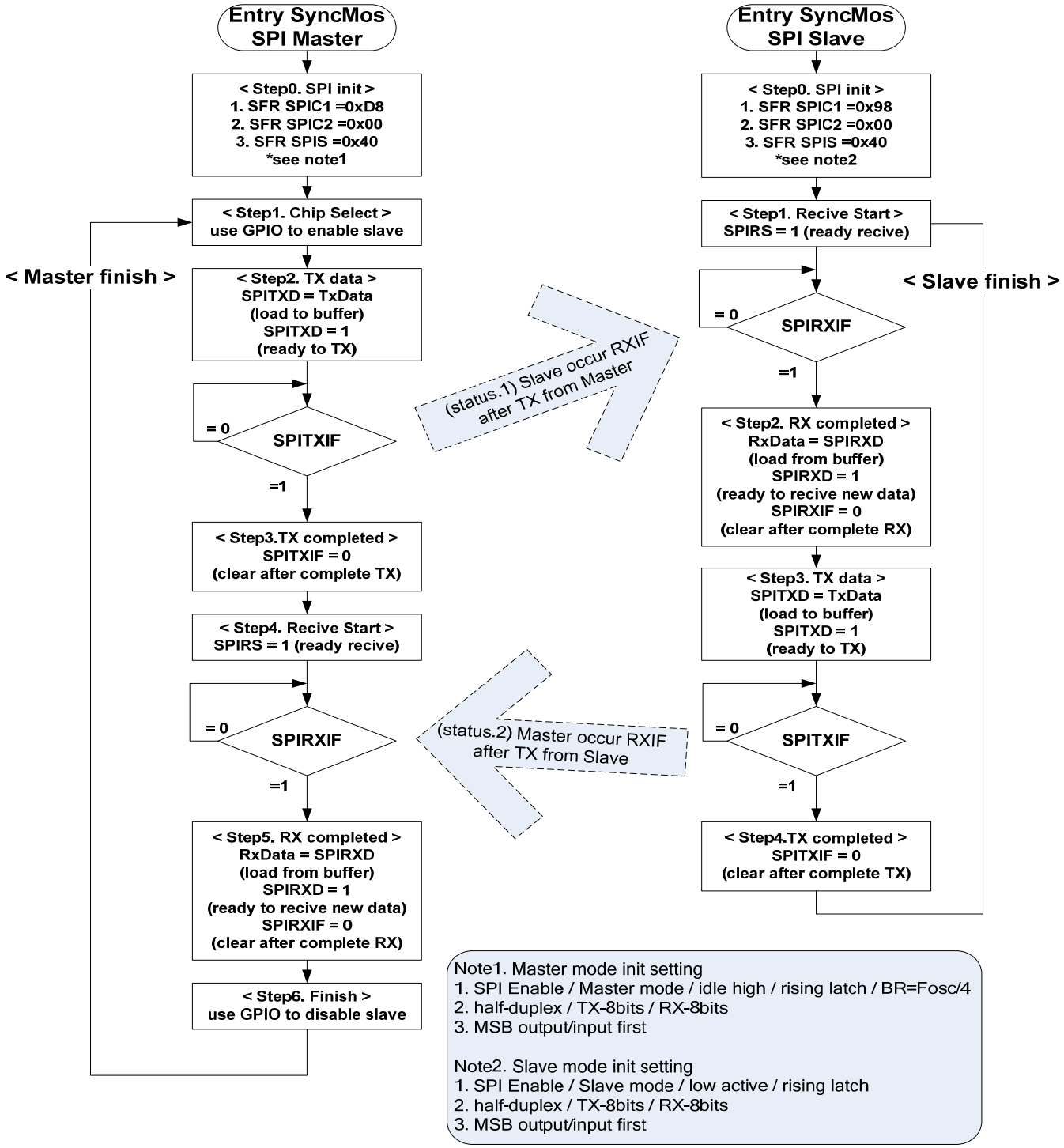
Table: Priority levels

IP1.x	IP0.x	Priority Level
0	0	Level0 (lowest)
0	1	Level1
1	0	Level2
1	1	Level3 (highest)

Table: Groups of priority

Bit	Group		
IP1.0, IP0.0	External interrupt 0	-	PWM interrupt
IP1.1, IP0.1	Timer 0 interrupt	-	***SPI interrupt***
IP1.2, IP0.2	External interrupt 1	Comparator interrupt	ADC interrupt
IP1.3, IP0.3	Timer 1 interrupt	-	KBI interrupt
IP1.4, IP0.4	Serial channel 0 interrupt	-	LVI interrupt
IP1.5, IP0.5	Timer 2 interrupt	-	IIC interrupt

5 以下為兩顆 MCU 分別當做 SPI master 及 slave 通訊的流程圖



6 SPI master 及 slave 通訊的半雙工範例程式

6.1 Master 傳資料(CMD 0x80)至 Slave

6.2 Slave 接受 CMD 後，判斷是否為正確 CMD，回傳資料給 Master

6.3 Master 接受 Slave 回傳的資料後，做資料比較，正確在 P0 輸出 0xaa，反之輸出 0x55

Description	Master:
C 語言	

```
//===================================================================
//                      SYNCMOS   TECHNOLOGY
//===================================================================
#include "SM39R16A2.h"
#include "SPI.h"

#define SPI_VECTOR 9    //SPI Interrupt Vevtor
#define SPIBR 7             //SPIBR[2:0] (SPI Baud Rate select 0~7)
#define SPIEN   1       //SPI Enable
#define SPIMSS 1        //Master/Slave Select
//#define SPISSP 0      //Slave active polarity Select(Slave used only)
#define SPICKP 1        //Clock idle polarity(Master used only)
#define SPICKE 0        //Clock sample edge select
#define SPIFD 0             //Full-duplex mode enable
#define SPIMLS 1        //MSB/LSB O/I First
#define SPITXIF 0x10    //Transmit Interrupt Flag
#define SPIRXIF 0x04    //Receive Interrupt Flag
#define SPITDR 0x08             //Transmit Data Ready
#define SPIRDR 0x02             //Receive Data Ready
#define SPIRS 0x01      //Receive Start
#define SPI_SS P1_4             //SPI device select pin
#define P2SPI 0x00      //SPI Function on P1(P2SPI=0) or P2(P2SPI=1)
#define TBC 0           //TBC[2:0] (SPI transmitter bit counter 0~7)
#define RBC 0           //RBC[2:0] (SPI receiver bit counter 0~7)

unsigned char nRData;
//-------------------------------------------------------------------------//
void SPI_Master_initialize(void)        //Initialize SPI
{
    EA = 0;                                 //Disable All Interrupt Function
    AUX = AUX | (P2SPI<<5);         //Set SPI Function on P1 or P2
    SPI_SS = 0;
    IESPI = 1;                          //Enable SPI Interrupt Function
    SPIC1 = (SPIEN<<7) | (SPIMSS<<6) | (SPICKP<<4) | (SPICKE<<3) | SPIBR;
    SPIC2 = (SPIFD<<7) | (TBC<<4) | RBC;
    SPIS = (SPIMLS<<6);
    SPITXD = 0xFF;
    SPIRXD = 0xFF;
    EA = 1;                                 //Enable All Interrupt
}
//-------------------------------------------------------------------------//

void SPI_MTransmit(unsigned char nTData)
{
    if(SPIMSS)                                      //SPIMSS=1,Master mode
    {
        SPITXD = nTData;                    //Load to TX Buffer
        SPIS = SPIS | SPITDR;               //set SPITDR, TX proceed
```

```
                while((SPIS&SPITDR) == SPITDR)    //SPITDR=(SPIS&0x08),TDR clear by H/W
                {
                        ;
                }
                while((SPIS&SPITXIF) == SPITXIF)
                {
                        ;
                }
        }
}
//---------------------------------------------------------------------//
unsigned char SPI_MReceive()
{
        if(SPIMSS)                                      //SPIMSS=1,Master mode
        {
                unsigned char nRData;
                SPI_SS = 0;
                SPIRXD = 0xFF;                          // clear SPITXD, useless
                SPIS = SPIS | SPIRS;                    // RX Start and send SPI CLK
                //while((SPIS&SPIRDR)==SPIRDR);  // SPIRDR=(SPIS&0x08),RDR set 1 by H/W
                while((SPIS&SPIRXIF)!=SPIRXIF)
                {
                        ;
                }
                nRData = SPIRXD;
                return nRData;
        }
}
//---------------------------------------------------------------------//
//*****************Variable bit counter*****************************//
/*void SPI_MVTransmit(unsigned char nTBC,unsigned char nTData)          //Slave Transmit Subroutine
variable bit counter
{
        if(SPIMSS)                              //SPIMSS=0,Slave mode
        {
                SPIC2=SPIC2|(nTBC<<4);
                SPITXD=(nTData);            //Load to TX Buffer
                SPIS=SPIS|SPITDR;           //set SPITDR, TX proceed
                while((SPIS&SPITDR)==SPITDR);           //SPITDR=(SPIS&0x08),TDR clear by H/W
                while((SPIS&SPITXIF)==SPITXIF);
        }
} */
//---------------------------------------------------------------------//
/*unsigned char SPI_MVReceive(unsigned char nRBC)                       //Slave Receive Subroutine
variable bit counter
{
        if(SPIMSS)                              //SPIMSS=0,Slave mode
        {
                unsigned char nRData;
                SPIC2=SPIC2|nRBC;
                SPIRXD=0xFF;
                SPIS=SPIS|SPIRS;            // RX Start and send SPI CLK
                //while((SPIS&SPIRDR)==SPIRDR);  // SPIRDR=(SPIS&0x08),RDR set 1 by
                                                        H/W
                while((SPIS&SPIRXIF) != SPIRXIF);
```

```
            nRData=SPIRXD;
            return nRData;
        }
}*/
//*********************************************************************//

void SPI_Disable(void)
{
    SPI_SS=1;
    SPIC1=0;            //Disable SPI Function
}
//---------------------------------------------------------------------//
void SPI_ISR(void) interrupt SPI_VECTOR //SPI Interrupt Subroutine
{
        if((SPIS & SPIRXIF) == SPIRXIF)     //SPIRDR=(SPIS&0x02) Receive Data Ready
        {
            SPIS = SPIS & 0xE1;             //Clear SPITXIF & SPITDR & SPIRXIF & SPIRDR
        }
        else if((SPIS & SPITXIF) == SPITXIF)
        {
            SPIS = SPIS & 0xE1;             //Clear SPITXIF & SPITDR & SPIRXIF & SPIRDR
        }
}
//---------------------------------------------------------------------//
void main(void)             //Main Function Start
{
    unsigned char SACK,CMD=0x80,loop=0,CMD_PASS=0xaa,CMD_FAIL=0x55;

   SPI_Master_initialize();    //Call SPI Initial Subroutine
    while(1)
    {
        SPI_MTransmit(CMD);         //Master transmit CMD to Slave
        SACK = SPI_MReceive();  //Call SPI Receive Subroutine
        if(SACK == CMD_PASS)    //If Receive CMD_PASS then Master P0 show CMD_PASS
        {                           //Else Master P0 show CMD_FAIL
            P0 = CMD_PASS;
        }
        else
        {
            P0 = CMD_FAIL;
        }
        SPI_Disable();
        while(1);
    }
}
```

Description	Slave:
C 語言	#include "SM39R16A2.h" #include "SPI.h" #define SPI_VECTOR 9 //SPI Interrupt Vevtor #define SPIBR 7 //SPIBR[2:0] (SPI Baud Rate select 0~7) #define SPIEN 1 //SPI Enable #define SPIMSS 0 //Master/Slave Select #define SPISSP 1 //Slave active polarity Select(Slave used only)

```
//#define SPICKP 0          //Clock idle polarity(Master used only)
#define SPICKE 0            //Clock sample edge select
#define SPIFD 0                   //Full-duplex mode enable
#define SPIMLS 1            //MSB/LSB O/I First
#define SPITXIF 0x10        //Transmit Interrupt Flag
#define SPIRXIF 0x04        //Receive Interrupt Flag
#define SPITDR 0x08               //Transmit Data Ready
#define SPIRDR 0x02               //Receive Data Ready
#define SPIRS 0x01          //Receive Start
#define P2SPI 0x00          //SPI Function on P1(P2SPI=0) or P2(P2SPI=1)
#define TBC 0               //TBC[2:0] (SPI transmitter bit counter 0~7)
#define RBC 0               //RBC[2:0] (SPI receiver bit counter 0~7)

unsigned char nRData;
//---------------------------------------------------------------------//

void SPI_Slave_initialize(void)          //Initialize SPI
{
      EA = 0;                                  //Disable All Interrupt Function
      AUX = AUX | (P2SPI<<5);          //Set SPI Function on P1 or P2
      IESPI = 1;                               //Enable SPI Interrupt Function
      SPIC1 = (SPIEN<<7) | (SPIMSS<<6) | (SPISSP<<5) | (SPICKE<<3) | SPIBR;
      SPIC2 = (SPIFD<<7) | (TBC<<4) | RBC;
      SPIS = (SPIMLS<<6);
      SPITXD = 0xFF;
      SPIRXD = 0xFF;
     EA = 1;                                   //Enable All Interrupt
}
//---------------------------------------------------------------------//

void SPI_STransmit(unsigned char nTData)       //Slave Transmit Subroutine fixed bit counter
{
      if(~SPIMSS)                                      //SPIMSS=0,Slave mode
      {
            SPITXD = nTData;                   //Load to TX Buffer
            SPIS = SPIS | SPITDR;              //set SPITDR, TX proceed
            while((SPIS&SPITDR) == SPITDR)     //SPITDR=(SPIS&0x08),TDR clear by H/W
            {
                  ;
            }
            while((SPIS&SPITXIF) == SPITXIF)
            {
                  ;
            }
      }
}
//---------------------------------------------------------------------//
unsigned char SPI_SReceive()               //Slave Receive Subroutine fixed bit counter
{
      if(~SPIMSS)                                      //SPIMSS=0,Slave mode
      {
            unsigned char nRData;
            SPIRXD = 0xFF;
            SPIS = SPIS | SPIRS;                     // RX Start and send SPI CLK
            //while((SPIS&SPIRDR)==SPIRDR);  // SPIRDR=(SPIS&0x08),RDR set 1 by H/W
            while((SPIS&SPIRXIF) != SPIRXIF)
            {
```

```
                ;
            }
            nRData = SPIRXD;
            return nRData;
        }
}

//**********************Variable bit counter*************************************//
/*void SPI_SVTransmit(unsigned char nTBC,unsigned char nTData)          //Slave Transmit Subroutine
variable bit counter
{
        if(~SPIMSS)                             //SPIMSS=0,Slave mode
        {
            SPIC2=SPIC2|(nTBC<<4);
            SPITXD=(nTData);          //Load to TX Buffer
            SPIS=SPIS|SPITDR;         //set SPITDR, TX proceed
            while((SPIS&SPITDR)==SPITDR);           //SPITDR=(SPIS&0x08),TDR clear by H/W
            while((SPIS&SPITXIF)==SPITXIF);
        }
} */
//-----------------------------------------------------------------------//
/*unsigned char SPI_SVReceive(unsigned char nRBC)                       //Slave Receive Subroutine
variable bit counter
{
        if(~SPIMSS)                             //SPIMSS=0,Slave mode
        {
            unsigned char nRData;
            SPIC2=SPIC2|nRBC;
            SPIRXD=0xFF;
            SPIS=SPIS|SPIRS;          // RX Start and send SPI CLK
            //while((SPIS&SPIRDR)==SPIRDR);  // SPIRDR=(SPIS&0x08),RDR set 1 by H/W
            while((SPIS&SPIRXIF) != SPIRXIF);
            nRData=SPIRXD;
            return nRData;
        }
}*/
//*************************************************************************//
void SPI_ISR(void) interrupt SPI_VECTOR         //SPI Interrupt Subroutine
{
            if((SPIS&SPIRXIF) == SPIRXIF)//SPIRDR=(SPIS&0x02) Receive Data Ready
            {
                SPIS = SPIS & 0xE1;             //Clear SPITXIF & SPITDR & SPIRXIF & SPIRDR
            }
            else if((SPIS&SPITXIF) == SPITXIF)
            {
                SPIS = SPIS & 0xE1;             //Clear SPITXIF & SPITDR & SPIRXIF & SPIRDR
            }
}
//-----------------------------------------------------------------------//
void main(void)             //Main Function Start
{
        unsigned char MACK,CMD=0x80,CMD_PASS=0xaa,CMD_FAIL=0x5a;

       SPI_Slave_initialize();              //Call SPI Initial Subroutine
        while(1)
        {
```

```
        MACK = SPI_SReceive();          //Call SPI Receive Subroutine
        if(MACK == CMD)                 //If Receive CMD then Slave transmit CMD_PASS to
Master
        {                               //Else transmit CMD_FAIL to Master
            SPI_STransmit(CMD_PASS);
            P0 = CMD_PASS;
        }
        else
        {
            SPI_STransmit(CMD_FAIL);
            P0 = SPIRXD;
        }
        while(1);
    }
}
```

7 SPI master 及 slave 全雙工通訊範例程式

7.1 Master 送 CMD(0x80)，同時接收 Slave 回傳(0x66)做確認

7.2 待兩方確認，Master 若確認收到 SACK(0x66)會回傳 CMD (0x80)，反之回傳 CMD_FAIL(0xff)

7.3 Slave 若確認收到 CMD(0x80)會回傳 CMD_PASS(0xaa)，反之回傳 CMD_FAIL(0x5a)，並將接收值顯示於 P0

7.4 Master 若確認收到 CMD_PASS(0xaa)會將 CMD_PASS 顯示於 P0，反之將 CMD_FAIL(0x55)顯示於 P0

Description	Master:
C 語言	

```
//===================================================================
//                    S Y N C M O S    T E C H N O L O G Y
//===================================================================
#include "SM39R16A2.h"
#include "SPI.h"

#define SPI_VECTOR 9    //SPI Interrupt Vevtor
#define SPIBR 7             //SPIBR[2:0] (SPI Baud Rate select 0~7)
#define SPIEN   1       //SPI Enable
#define SPIMSS 1        //Master/Slave Select
//#define SPISSP 0      //Slave active polarity Select(Slave used only)
#define SPICKP 1        //Clock idle polarity(Master used only)
#define SPICKE 0        //Clock sample edge select
#define SPIFD 1             //Full-duplex mode enable
#define SPIMLS 1        //MSB/LSB O/I First
#define SPITXIF 0x10    //Transmit Interrupt Flag
#define SPIRXIF 0x04    //Receive Interrupt Flag
#define SPITDR 0x08         //Transmit Data Ready
#define SPIRDR 0x02         //Receive Data Ready
#define SPIRS 0x01      //Receive Start
#define SPI_SS P1_4         //SPI device select pin
#define P2SPI 0x00      //SPI Function on P1(P2SPI=0) or P2(P2SPI=1)

unsigned char nRData;
```

```
void SPI_Master_initialize(void)  //Initialize SPI
{
      EA = 0;                                       //Disable All Interrupt Function
      AUX = AUX | (P2SPI<<5);               //Set SPI Function on P1 or P2
      SPI_SS = 0;
      IESPI = 1;                                    //Enable SPI Interrupt Function
      SPIC1 = (SPIEN<<7) | (SPIMSS<<6) | (SPICKP<<4) | (SPICKE<<3) | SPIBR;
      SPIC2 = (SPIFD<<7);
      SPIS = (SPIMLS<<6) | SPIRS;
      SPITXD = 0xFF;
      SPIRXD = 0xFF;
     EA = 1;                                        //Enable All Interrupt
}
//---------------------------------------------------------------------//
void SPI_MFullTransmit(unsigned char nTData)
{
      if (SPIMSS)                                       //SPIMSS=1,Master mode
      {
            SPIRXD = 0xFF;                              // clear SPITXD, useless
            SPITXD = (nTData);                    //Load to TX Buffer
            SPIS = SPIS | SPITDR;                 //set SPITDR, TX proceed
            while((SPIS&SPITDR) == SPITDR)    //SPITDR=(SPIS&0x08),TDR clear by H/W
            {
                  ;
            }

            while((SPIS&SPITXIF) == SPITXIF)
            {
                  ;
            }
      }
}

//**********************Variable bit counter**************************************//
/*void SPI_MVFullTransmit(unsigned char nTData)           //Slave Transmit Subroutine variable bit
counter
{
      if(SPIMSS)                                    //SPIMSS=0,Slave mode
      {
            SPIRXD=0xFF;                  // clear SPITXD, useless
            SPITXD=(nTData);              //Load to TX Buffer
            SPIS=SPIS|SPITDR;             //set SPITDR, TX proceed
            while((SPIS&SPITDR)==SPITDR);           //SPITDR=(SPIS&0x08),TDR clear by H/W
            while((SPIS&SPITXIF)==SPITXIF);
      }
} */
//*************************************************************************************//

void SPI_Disable(void)
{
      SPI_SS = 1;
      SPIC1 = 0;                    //Disable SPI Function
}
//---------------------------------------------------------------------//
void SPI_ISR(void) interrupt SPI_VECTOR //SPI Interrupt Subroutine
```

```
{
        if(((SPIS&SPITXIF) == SPITXIF) & ((SPIS&SPIRXIF) == SPIRXIF))
        {
            nRData = SPIRXD;
            SPIS = SPIS & 0xE1;             //Clear SPITXIF & SPITDR & SPIRXIF & SPIRDR
        }
}
//-------------------------------------------------------------------//
void main(void)             //Main Function Start
{
    unsigned char SACK,CMD=0x80,loop=0,CMD_PASS=0xaa,CMD_FAIL=0x55;

   SPI_Master_initialize();    //Call SPI Initial Subroutine
    while(1)
    {
        SPI_MFullTransmit(CMD);
        SACK = nRData;
        if(SACK == 0x66)
        {
            SPI_MFullTransmit(CMD);
        }
        else
        {
            SPI_MFullTransmit(0xff);
        }
        SACK = nRData;
        if(SACK == CMD_PASS)        //If Receive CMD_PASS then Master P0 show CMD_PASS
        {                           //Else Master P0 show CMD_FAIL
            P0 = CMD_PASS;
        }
        else
        {
            P0 = CMD_FAIL;
        }
        SPI_Disable();
        while(1)
        {
            ;
        }
    }
}
```

Description	Slave:

```
C 語言
//=====================================================================
//                    S Y N C M O S    T E C H N O L O G Y
//=====================================================================
#include "SM39R16A2.h"
#include "SPI.h"

#define SPI_VECTOR 9    //SPI Interrupt Vevtor
#define SPIBR 7             //SPIBR[2:0] (SPI Baud Rate select 0~7)
#define SPIEN   1       //SPI Enable
#define SPIMSS 0        //Master/Slave Select
#define SPISSP 1        //Slave active polarity Select(Slave used only)
//#define SPICKP 0      //Clock idle polarity(Master used only)
```

```
#define SPICKE 0          //Clock sample edge select
#define SPIFD 1                 //Full-duplex mode enable
#define SPIMLS 1          //MSB/LSB O/I First
#define SPITXIF 0x10      //Transmit Interrupt Flag
#define SPIRXIF 0x04      //Receive Interrupt Flag
#define SPITDR 0x08             //Transmit Data Ready
#define SPIRDR 0x02             //Receive Data Ready
#define SPIRS 0x01        //Receive Start
#define P2SPI 0x00        //SPI Function on P1(P2SPI=0) or P2(P2SPI=1)

unsigned char nRData;

void SPI_Slave_initialize(void)          //Initialize SPI
{
      EA = 0;                                   //Disable All Interrupt Function
      AUX = AUX | (P2SPI<<5);         //Set SPI Function on P1 or P2
      IESPI = 1;                                //Enable SPI Interrupt Function
      SPIC1 = (SPIEN<<7) | (SPIMSS<<6) | (SPISSP<<5) | (SPICKE<<3) | SPIBR;
      SPIC2 = (SPIFD<<7);
      SPIS = (SPIMLS<<6) | SPIRS;
      SPITXD = 0xFF;
      SPIRXD = 0xFF;
     EA = 1;                                    //Enable All Interrupt
}

void SPI_SFullTransmit(unsigned char nTData)
{
      if(~SPIMSS)                                       //SPIMSS=1,Master mode
      {
            SPIRXD = 0xFF;                              // clear SPITXD, useless
            SPITXD = (nTData);                  //Load to TX Buffer
            SPIS = SPIS | SPITDR;               //set SPITDR, TX proceed
            while((SPIS&SPITDR)==SPITDR)      //SPITDR=(SPIS&0x08),TDR clear by H/W
            {
                  ;
            }

            while((SPIS&SPITXIF)==SPITXIF)
            {
                  ;
            }
      }
}

//**********************Variable bit counter**************************************//
/*void SPI_SVFullTransmit(unsigned char nTData)           //Slave Transmit Subroutine variable bit
counter
{
      if(~SPIMSS)                               //SPIMSS=0,Slave mode
      {
            SPIRXD=0xFF;                // clear SPITXD, useless
            SPITXD=(nTData);            //Load to TX Buffer
            SPIS=SPIS|SPITDR;           //set SPITDR, TX proceed
            while((SPIS&SPITDR)==SPITDR);           //SPITDR=(SPIS&0x08),TDR clear by H/W
            while((SPIS&SPITXIF)==SPITXIF);
      }
```

```
} */
//*****************************************************************************//

void SPI_ISR(void) interrupt SPI_VECTOR        //SPI Interrupt Subroutine
{
        if(((SPIS&SPITXIF) == SPITXIF) & ((SPIS&SPIRXIF) == SPIRXIF))
        {
            nRData = SPIRXD;
            SPIS = SPIS & 0xE1;             //Clear SPITXIF & SPITDR & SPIRXIF & SPIRDR
        }
}
void main(void)             //Main Function Start
{
    unsigned char MACK,CMD=0x80,CMD_PASS=0xaa,CMD_FAIL=0x5a;

   SPI_Slave_initialize();          //Call SPI Initial Subroutine
    while(1)
    {
        SPI_SFullTransmit(0x66);
        MACK = nRData;
        if(MACK == CMD)          //If Receive CMD then Slave transmit CMD_PASS to Master
        {                                   //Else transmit CMD_FAIL to Master
            SPI_SFullTransmit(CMD_PASS);
            P0 = CMD_PASS;
        }
        else
        {
            SPI_SFullTransmit(CMD_FAIL);
            P0 = SPIRXD;
        }
        while(1)
        {
            ;
        }
    }
}
```

新茂國際科技希望能爲客戶減少開發的時間及辛勞，故提供 “Codzard 範例程式產生器”
可於 新茂網站首頁>下載專區> 軟體下載 內下載此軟體，如有任何建議，請來信告知，謝謝!

銷售客服
電子信箱： sales@mail.syncmos.com.tw
技術支援
電子信箱： support@mail.syncmos.com.tw