SM39R 系列 IIC 使用說明

1 適用產品：

1.1 SM39R02A2/ SM39R04A2

1.2 SM39R2051G1/ SM39R4051G1

1.3 SM39R08A2/ SM39R12A2/ SM39R16A2

2 IIC 使用概述：

2.1 使用硬件 SCL (clock)及 SDA (data)管腳，須接上拉電阻，建議 2K~10KΩ。

2.2 速度：SCL 最高可達 400Kbps(由軟件設定 SFR IICBR[2:0])。

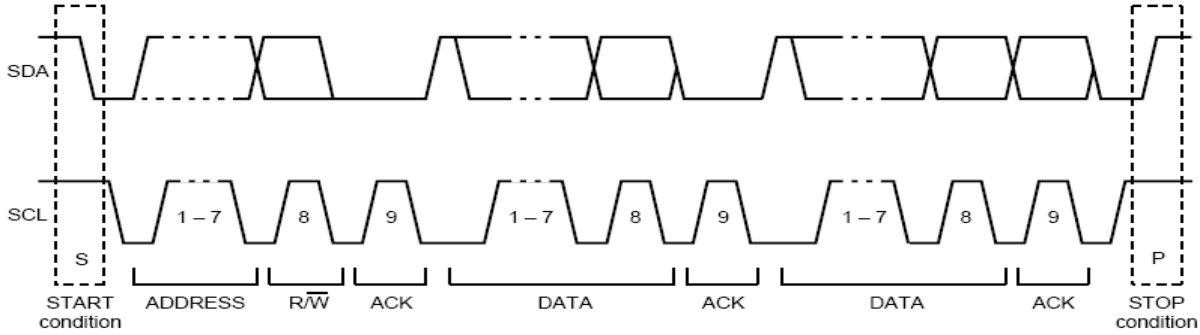
2.3 IIC 和 ICE 內部使用同一電路，所以使用 IIC 功能和 ICE 模擬不可同時使用。

2.4 IIC 可選擇 master 或 slave 兩種模式，提供中斷(RXIF, TXIF, MPIF)及兩組控制位址(IICA1, IICA2)使用。

2.5 Master 硬件可自動產生 IIC 的啓動(START)、重新啓動(Re-START)及停止(STOP)信號，自動偵測 bus busy(BB flag)及 arbitration(LAIF)。

2.6 Slave 硬件可偵測 IIC 的 START(Match)、Re-START(Match)及 STOP(MPIF)信號。

2.7 Slave 最多可接 127 devices，線材電容量最高限制爲 400pF。



3 IIC 相關的特殊暫存器 Special Function Register (SFR)

Mnemonic	Description	Direct	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	RESET
IIC function											
AUX	Auxiliary register	91h	BRGS	-	P2SPI	P2UR	P2IIC	-	-	DPS	00H
IICCTL	IIC control register	F9h	IICEN	MSS	MAS	AB_EN	BF_EN	IICBR[2:0]			04H
IICS	IIC status register	F8h	-	MPIF	LAIF	RXIF	TXIF	RXAK	TXAK	RW or BB	00H
IICA1	IIC Address 1 register	FAh	IICA1[7:1]							MATCH1 or RW1	A0H
IICA2	IIC Address 2 register	FBh	IICA2[7:1]							MATCH2 or RW2	60H
IICRWD	IIC Read/Write register	FCh	IICRWD[7:0]								00H
IICEBT	IIC Enaable Bus Transaction	FDh	FU_EN		-	-	-	-	-	-	00H

Mnemonic: AUX **Address: 91h**

7	6	5	4	3	2	1	0	Reset
BRGS	-	P2SPI	P2UR	P2IIC	-	-	DPS	00H

P2IIC: P2IIC = 0 – IIC function on P1.
P2IIC = 1 – IIC function on P2.

P2IIC	SCL	SDA
0	P1.2	P1.3
1	P2.6	P2.7

IIC 腳位直接由軟件指定至 P1 或 P2，可避免和其它特殊功能腳位重複，提高硬件規劃的相容性。有此功能的 MCU 如下表：

MCU device	P4IIC(IIC function pin assignment)
SM39R16A2/ SM39R12A2/ SM39R08A2	○
SM39R4051/ SM39R2051	X
SM39R04G1/ SM39R02G1	X

○：表示該型號可以使用 P2IIC 功能
X：表示該型號不可使用 P2IIC 功能

Mnemonic: IICCTL **Address: F9h**

7	6	5	4	3	2	1	0	Reset
IICEN	MSS	MAS	AB_EN	BF_EN	IICBR[2:0]			04h

IICEN: **IIC 致能位元**(Enable IIC module)

IICEN = 1，IIC 致能。

IICEN = 0，IIC 禁能。

MSS: **主從模式選擇位元 Master or slave mode select：**

MSS = 1，設定爲 master mode。

MSS = 0，設定爲 slave mode。

***當使用 IIC 設定其它 IIC SFR 時，程序必須先致能此位元。**

MAS: **控制位址位元 Master address select (master mode only)：**

MAS = 0 ，選擇控制位址(control byte)由 SFR IICA1 送出。

MAS = 1 ，選擇控制位址(control byte)由 SFR IICA2 送出。

AB_EN: **仲裁檢查致能位元 Arbitration lost enable bit. (Master mode only)**

AB_EN = 1，致能仲裁檢查。

AB_EN = 0，禁能仲裁檢查，master不理會主控權丟失情況。

仲裁檢查：在multi-master架構，master由硬件送出SDA爲"1"，卻偵測到爲"0"時，即發生仲裁丟失，失去對IIC bus的主控權(表示IIC bus上已有其它master正在通訊)，**LAIF**將被設置爲"1"。

當系統爲多主從(multi-master)應用時，必須致能AB_EN，並檢查LAIF位元。

當系統爲單主從(single-master)應用時，可禁能AB_EN。

BF_EN: **匯流排忙碌偵測致能位元 Bus busy enable bit. (Master mode only)**

BF_EN = 1，致能，若BF=1，master會無法送出啓動條件(Start condition)，直到BF=0。
BF_EN = 0，禁能，master硬件不理會BF=0或1。

致能BF_EN時：

1. 當Master硬件偵測到**啓動條件(Start condition)**，或**BUS上的SCL=0**、或**SDA=0**時，即爲bus busy狀態，**BB會由硬件設爲"1"**。
2. 當Master硬件偵測到**停止條件(Stop condition)**後，**BB由硬件清除爲"0**"。此位元也可由軟件清除。

當系統爲多主從(multi-master)應用時，必須致能BF_EN，並檢查BB(bus busy)位元。

當系統爲單主從(single-master)應用時，可禁能BF_EN。

IICBR[2:0]: **IIC 鮑率選擇元位 Baud rate selection (master mode only)：**

系統頻率(Fosc)依據 MCU 外部晶振(或內部晶振)而定，預設值爲 Fosc/512 。

IICBR[2:0]	Baud rate
000	Fosc/32
001	Fosc/64
010	Fosc/128
011	Fosc/256
100	Fosc/512
101	Fosc/1024
110	Fosc/2048
111	Fosc/4096

Mnemonic: IICS **Address: F8H**

7	6	5	4	3	2	1	0	Reset
-	MPIF	LAIF	RXIF	TXIF	RXAK	TxAK	RW or BB	00H

MPIF: **停止中斷旗標 Stop Interrupt Flag：**

1. master: **送出**或**偵測(multi-master)**到stop condition時，MPIF會由HW設置爲1，會產生中斷，但與RxIF及TxIF無關。MPIF必須由軟件清除，或離開中斷時由HW清除。
2. slave: 收到stop condition時，MPIF會由HW設置爲1，但不產生中斷，MPIF必須由軟件清除。

LAIF: **仲裁檢查旗標 Lost Arbitration bit. (Master mode only)**

LAIF = 1，失去匯流排主控權 lost bus arbitration。
LAIF = 0，匯流排閒置 bus ready。

當系統爲多主從(multi-master)應用時，請致能該偵測位元(AB_EN)。

當系統爲單主從(single-master)應用時，可禁能該偵測位元(AB_EN)。

AB_EN = 1，致能仲裁檢查。
AB_EN = 0，禁能仲裁檢查，master不理會主控權丟失情況。
仲裁檢查：在multi-master架構，master由硬件送出SDA爲"1"，卻偵測到爲"0"時，即發生仲裁丟失，失去對IIC bus的主控權(表示IIC bus上已有其它master正在通訊)，**LAIF**將被設置爲"1"。

當系統爲多主從(multi-master)應用時，必須致能AB_EN，並檢查LAIF位元。

當系統爲單主從(single-master)應用時，可禁能AB_EN。

RxIF: **資料接收中斷旗標 Data receive interrupt flag**

以下事件發生時 RxIF 會由**硬件**設爲"1"，並發生中斷：

1. master mode: IICRWD 載入資料完成時。

2. slave mode: IICRWD 載入資料完成時。
3. slave mode: Match (即 salve 收到 start condition 及 device ID)時。

RxIF必須由軟件清除。

TxIF: **資料傳送中斷旗標 Data transmit interrupt flag**

以下事件發生時 TxIF 會由硬件設爲"1"，並發生中斷：

1. master mode: 送出 start condition 及 device ID 時。
2. master mode: 資料從 IICRWD 載至位移暫存器，並已由位移暫存器傳送時。
3. slave mode: 資料從 IICRWD 載至位移暫存器，並已由位移暫存器傳送時。

TxIF必須由軟件清除。

RxAK: **接收應答位元 Receive acknowledgement (read only)**

此爲唯讀位元，傳送端等待接收端回覆應答(ACK/NACK)，不可由程序編寫。

1. master mode：傳出資料(8-bit)後，由 slave 回覆應答位元(RxAK)。
2. slave mode：傳出資料(8-bit)後，由 master 回覆應答位元(RxAK)。

TxAK: **傳送應答位元 Transmit acknowledgement**

接收端接收前必須先設定好 TxAK(ACK or NACK)，接收資料完成後，即由硬件送出 TxAK 至傳送端。

RW or BB: **Master Mode:**

匯流排忙碌偵測位元 Bus busy bit

BB = 1，匯流排忙碌 bus busy。

BB = 0，匯流排閒置 bus ready。

致能BF_EN時：

1. 當Master硬件偵測到**啓動條件(Start condition)**，或**BUS上的SCL=0**、或**SDA=0**時，即爲bus busy狀態，**BB會由硬件設爲"1"**。
2. 當Master硬件偵測到**停止條件(Stop condition)**後，**BB由硬件清除爲"0"**。此位元也可由軟件清除。

當系統爲多主從(multi-master)應用時，必須致能BF_EN，並檢查BB(bus busy)位元。

當系統爲單主從(single-master)應用時，可禁能BF_EN。

Slave Mode:

讀寫狀態位元 Read or Write status bit(read only)

此位元由 master 的位址 IICA1(或 IICA2) 的 8th-bit 所控制：

RW = 0，master 要求 slave 的 IIC module 爲**接收模式**(即 master write, slave read)。

RW = 1，master要求slave的IIC module爲**傳送模式**(即master read, slave write)。

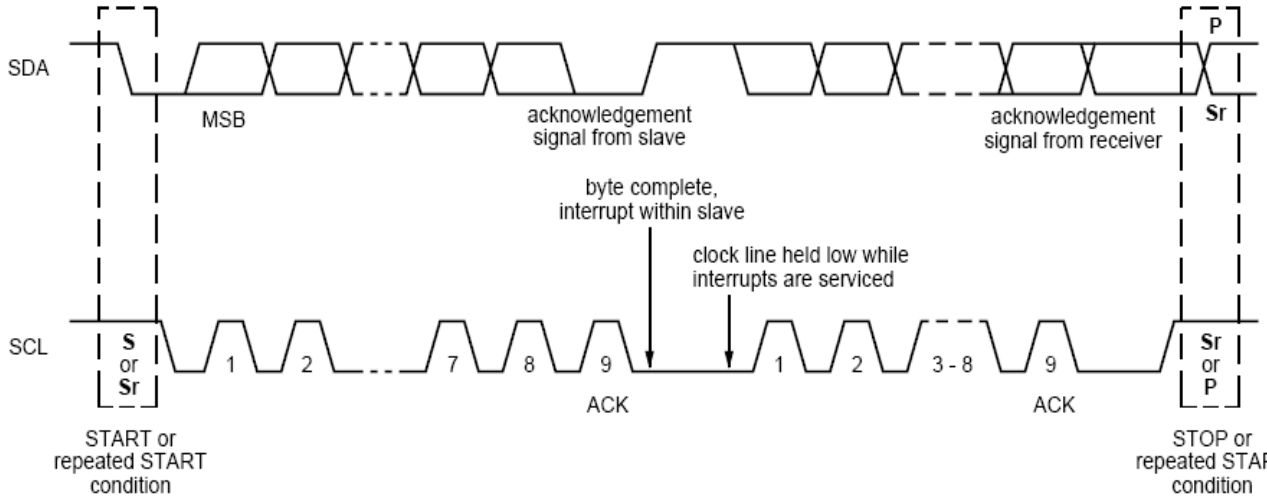


Fig. Acknowledgement bit in the 9th bit of a byte transmission

Mnemonic: IICA1 **Address: FAH**

7	6	5	4	3	2	1	0	Reset
IICA1[7:1]							Match1 or RW1	A0H
R/W							R or R/W	

Slave mode:

IICA1[7:1]: **IIC Address registers**

控制位址(第一組)IICA1 共 7-bit，須由軟件設定。

Match1: 當 slave 接收到 master 的**啓動條件(Start or Re-Start condition)**，及控制位址(IICA1)7-bit，由硬件比對：

= 1，當比對相同時，Match1 及 RxIF 皆由硬件設置爲"1"，並產生中斷。

= 0，Match1 由硬件清除爲"0"，如下情況：

1. RXIF 發生但未偵測到**啓動條件(Start or Re-Start condition)**。
2. TXIF 發生時。

Master mode:

IICA1[7:1]: **IIC Address registers**

1. 第一組控制位址 IICA1 共 7-bit，由軟件設定。
2. 當 MAS = 0，選擇控制位址第一組(IICA1)。
   當 MAS = 1，選擇控制位址第二組(IICA2)。
3. 當 IICEBT 由**軟件**設置爲"10"時，會送出**啓動條件(Start or Re-Start condition)**及**控制位址位元(IICA1)**。

RW1: master 送出的 8th-bit(R／W-bit)，告知 slave 讀寫的狀態

RW1= 1：爲 master IIC module 接收模式(即 master read, slave write)。

RW1= 0：爲 master IIC module 傳送模式(即 master write, slave read)。

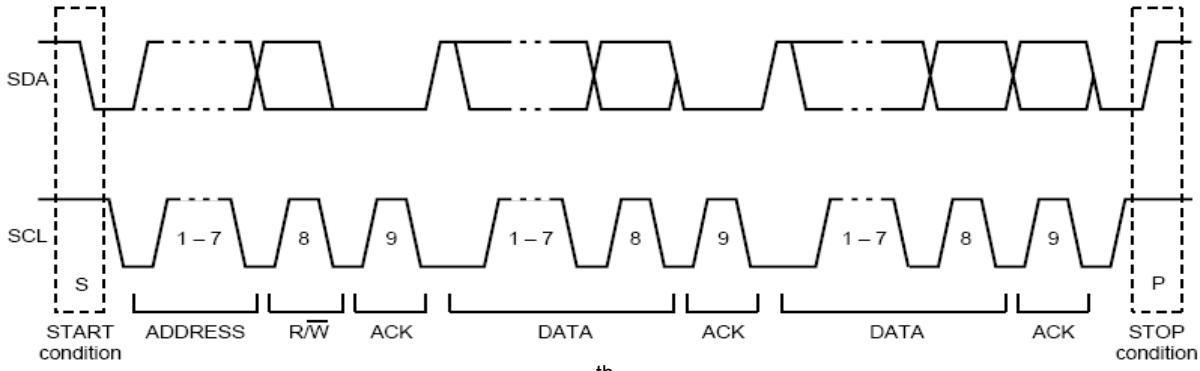


Fig. : RW bit in the 8th bit after IIC address

Mnemonic: IICA2 **Address: FBh**

7	6	5	4	3	2	1	0	Reset
IICA2[7:1]							Match2 or RW2	60h
R/W							R or R/W	

Slave mode:

IICA2[7:1]: IIC Address registers

控制位址(第二組)IICA2 共 7-bit，須由軟件設定。

Match2: 當 slave 接收到 master 的**啓動條件(Start or Re-Start condition)**，及控制位址(IICA2)7-bit，由硬件比對：

= 1，當比對相同時，Match2 及 RxIF 皆由硬件設置爲"1"，並產生中斷。

= 0，Match2 由硬件清除爲"0"，如下情況：
1. RXIF 發生但未偵測到**啓動條件(Start or Re-Start condition)**。
2. TXIF 發生時。

Master mode:

IICA2[7:1]: IIC Address registers
1. 控制位址(第二組)IICA2 共 7-bit，由軟件設定。
2. 當 MAS = 0，選擇控制位址第一組(IICA1)。
   當 MAS = 1，選擇控制位址第二組(IICA2)。
3. 當 IICEBT 由**軟件**設置爲"10"時，會送出**啓動條件(Start or Re-Start condition)**及**控制位址位元(IICA2)**。

RW2: master 送出的 8th-bit(R／W-bit)，告知 slave 讀寫的狀態
RW2= 1：爲 master IIC module 接收模式(即 master read, slave write)。
RW2= 0：爲 master IIC module 傳送模式(即 master write, slave read)。

Mnemonic: IICRWD **Address: FCh**

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

IICRWD[7:0]: 資料讀寫暫存器(8-bit) IIC read write data buffer：
1. IIC 接收時，IICRWD 爲接收資料的暫存區；
   當資料接收完成時 RxIF=1，再讀取 IICRWD，讀完後必須 release IIC bus(即 IICEBT 寫入"01")。
2. IIC 傳送時，IICRWD 爲傳送資料的暫存區；
   先將資料寫入 IICRWD，再寫入 IIC 傳送指令(即 IICEBT 寫入"01")即可，傳送完成時 TxIF=1。

Mnemonic: IICEBT Address: FDH

7	6	5	4	3	2	1	0	Reset
FU_EN		-	-	-	-	-	-	00H

Master Mode

FU_EN[7:6]:

FU_EN[7:6]	
00	Reserved
01	**Master transmit data 傳資料操作命令及流程：** 1. Write data to **IICRWD** (by software) **2. *FU_EN* write 01 (by software)** 3. Transmit data finish and receive **RxAK** when **FU_EN** auto-clear (by HW) 4. **TxIF** will set (by HW) then interrupt occur, **TxIF** must clear (by software) 5. Check **RxAK** status *EX:* *IICRWD=TxData; // load data* *IICEBT=0x40; // trans. data*

	while(IICEBT != 0x00);// waiting data trans. Finish *while(TxIF=0);* *TxIF=0;* *if(RxAK)* *{* *// by user…* *}* *else* *{* *// by user…* *}* **Master receive data 收資料操作命令及流程：** 1. **TxAK** set ACK/NACK before receive data (by software) 2. ***FU_EN write 01(by software)*** for release bus 3. Receive data finish and transmit **TxAK** when **FU_EN** auto-clear (by HW) 4. **RxIF** will set (by HW) then interrupt occur 5. User can read data from **IICRWD** (by software) 6. **RxIF** must clear (by software) *Ex:* *TXAK=ACKStatus; // Feedback ACK/d_NACK to slave* *IICEBT=0x40; // Ready for receive* *while(IICEBT != 0x00);// waiting data recive finish* *while(RxIF=0);* *RxIF=0;* *temp=IICRWD;*
10	送出**啓動條件(Start or Re-Start condition)**及**控制位址位元(IICA1/IICA2)** *Ex:* *IICEBT = 0x80; // generate a start condition* *while(IICEBT != 0x00);* *return RXAK;*
11	送出**停止條件(Stop condition)** IIC bus module generate a stop condition on the SDA/SCL.

	EX: *IICEBT = 0xC0; // generate a stop condition* *while(IICEBT != 0x00);// waiting data recive finish*

Notice:

在此僅作簡易的程序示範，完整程序可參考”完整範例程序”的master及slave，或使用SyncMOS Codzard產生。

Slave mode:

Slave transmit data 傳資料操作命令及流程(同 master)：

1. Write data to **IICRWD** (by software)
2. ***FU_EN write 01 (by software)***
3. Transmit data finish and receive **RxAK** when **FU_EN** auto-clear (by HW)
4. **TxIF** will set (by HW) then interrupt occur, **TxIF** must clear (by software)

Slave receive data 收資料操作命令及流程(不同於 master)：

1. **TxAK** set **ACK** before receive data (by software)
2. Receive data finish and transmit **TxAK** when **IICEBT** auto-clear (by HW)
3. **RxIF** will set (by HW) then interrupt occur
4. User can read data from **IICRWD** (by software)
5. ***FU_EN write 01(by software)*** for release bus
6. **RxIF** must clear (by software)

Slave receive Start or Re-Start condition 操作命令及流程：

1. **TxAK** set **ACK** before match (by software)
2. Transmit **TxAK** when device ID match
3. **RxIF** will set (by HW) then interrupt occur
4. User can read data from **IICRWD** (by software)
5. ***FU_EN write 01(by software)*** for release bus
6. **RxIF** must clear (by software)

```
void IIC_interrupt() interrupt d_IIC_Vector
{
    unsigned char buf;

    //slave rx=============================================================
    if(RXIF)
    {
        //=====================================================
        if ((IICA1 & 0x01) || (IICA2 & 0x01))   // match
        {
```

```
            if (n_RW == d_Write)            // don't release if read phase
                IICEBT  = d_CMD_RW;         //IIC bus release if write phase
        }

        //=========================================================
        else
        {
            buf         = IICRWD;           // save addr offset
            IICEBT      = 0x40;         // IIC bus release
        }
        //=========================================================
        RXIF = 0;
    }

    //slave tx=================================================================
    if (TXIF)
    {
        TXIF = 0;                   // Clear interrupt flag
    }
    //=========================================================================
}
```

Notice:

1. Slave只須設定**FU_EN**[7:6] =[0 1] (**當作Release BUS的功能**)，其它值不可設定。
2. 如果沒release，IIC SCL會被lock住(=0)。
3. 在此僅作簡易的程序示範，完整程序可參考"完整範例程序"的master及slave，或使用SyncMOS Codzard產生。

4 IIC 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 (SM39R16A2/ SM39R12A2/ SM39R08A2 only)	002Bh	5
PWMIF – PWM interrupt (SM39R16A2/ SM39R12A2/ SM39R08A2 only)	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
Comparator interrupt (SM39R16A2/ SM39R12A2/ SM39R08A2 only)	0093h	18

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

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	RESET
Interrupt											
IEN0	Interrupt Enable 0 register	A8H	***EA***	-	ET2	ES	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	-	-	-	-	-	ECmpI	-	-	00H
IRCON	Interrupt request register	C0H	EXF2	TF2	***IICIF***	LVIIF	KBIIF	ADCIF	SPIIF	PWMIF	00H
IRCON2	Interrupt request register 2	97H	-	-	-	-	-	CmpIF	-	-	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

IIC 中斷應用可參考以下設定：

(1) IIC 中斷致能設定：

```
IEN0 |= 0x80;                    //Enable interrupt All
IEN1 |= 0x20;                    //Enable interrupt IIC
```

(2) IIC 中斷程序表示：

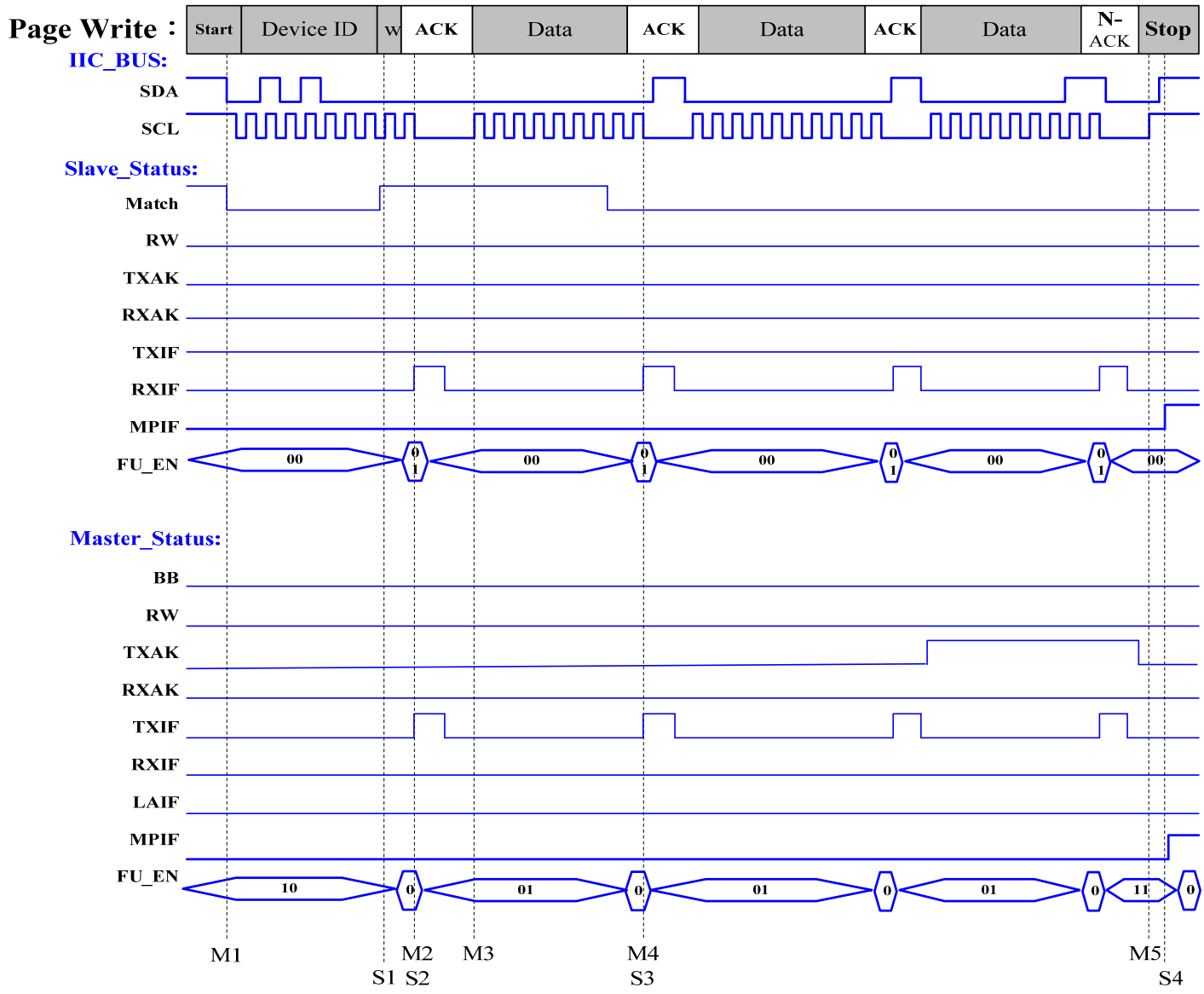
```
void ISR_IIC (void) interrupt 13
{
    if(TXIF)
    {
        TXIF = 0;//Clear interrupt flag
    }
    if(RXIF)
    {
        RXIF = 0;//Clear interrupt flag
    }
    if(MPIF)
    {
        MPIF = 0;//Clear interrupt flag
    }
}
```

Ps. IIC 中斷發生，必須清除 IICS 的 TXIF 或 RXIF 或 MPIF，而非 IRCON 的 IICIF！

5 以下使用兩顆 MCU，分別做爲 master 及 slave，以最普遍使用的 EEPROM 24Cxx 做爲通訊 protocal 來說明 IIC 應用：

5.1 MCU IIC flag 狀態圖(IIC Page Write)：

Master:	Slave:
• M1. **FU_EN**寫入10，Master送出Start、Device ID及RW訊號。 • M2. 送出完成時，**FU_EN由硬件清爲零**，**TxIF**由硬體設置爲1，並產生中斷，**TxIF**由軟體清除。時也需判斷**RxAK**的狀態。 • M3. Data寫入**IICRWD**後， **FU_EN**寫入01，即會將Data送出。 • M4. Data送出後，**FU_EN由硬件清爲零**，**TxIF**由軟件設置爲1，並產生中斷。 • M5. **FU_EN寫入11**，送出Stop訊號，後**MPIF**由硬件設爲1，並產生中斷。	• S1. S1. 當slave硬件偵測到Start condition及device ID，**Match** bit由硬件設爲1。 • S2. 收到Device ID(及RW)時，硬件會傳出**TxAK**；**RxIF**會由硬件設爲1，並觸發中斷，軟件必須寫**FU_EN**=01，用來release bus。 • S3. 收到Data時，硬件會傳出**TxAK**；**RxIF**會由硬件設爲1，並觸發中斷，軟件必須寫**FU_EN**=01，用來release bus。 • S4. 在收到Stop condition後，**MPIF**由硬件設爲1，**MPIF**須由軟體清除爲0。

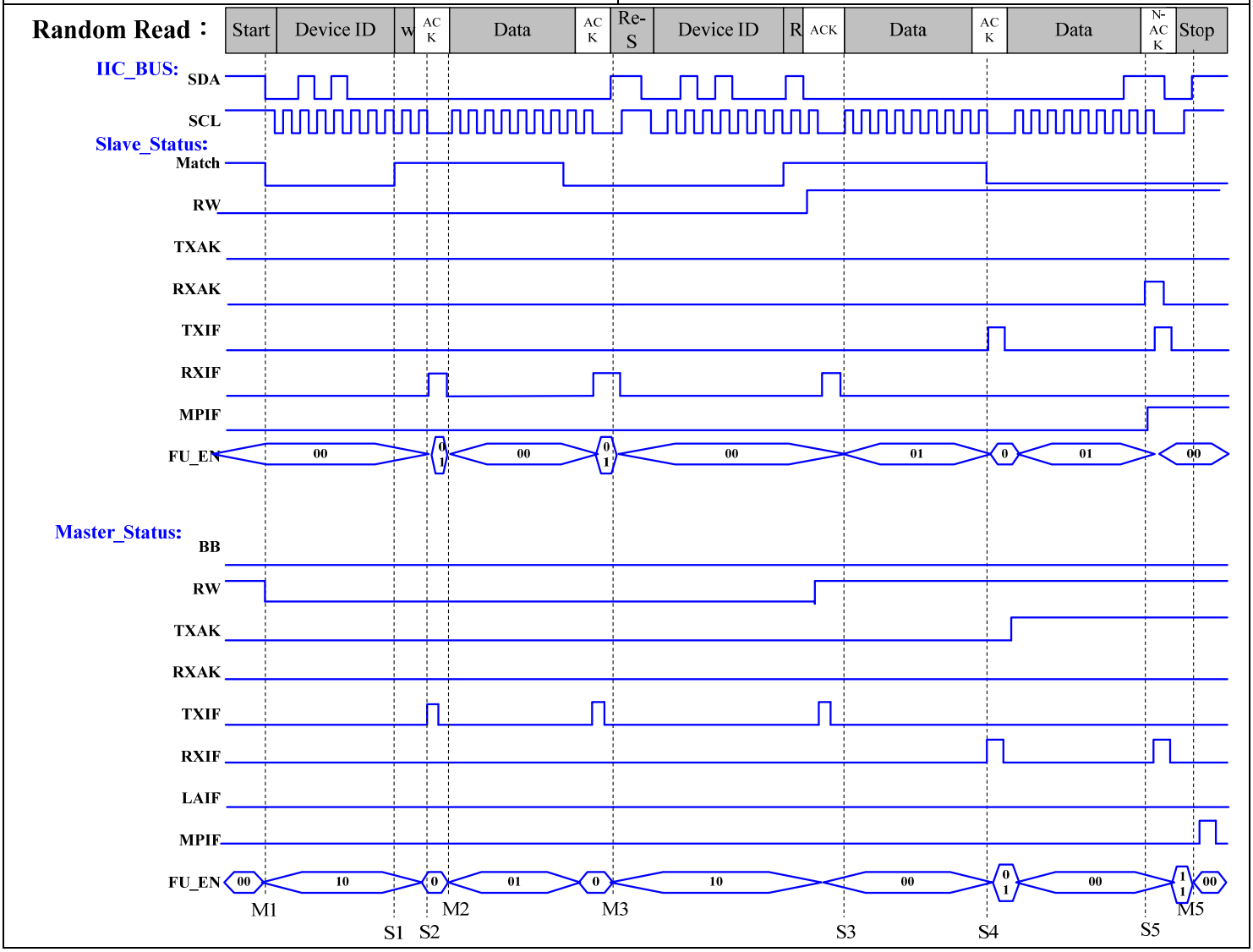


5.2 MCU IIC flag 狀態圖(IIC Random Read)：

Master:	Slave:

- M1&M3. **FU_EN**寫入10h，Master送出Start(or Re-Start)、Device ID及RW訊號。收到ACK後**FU_EN**由硬件清除，並觸發**TxIF**。
- M2. TX：將Data寫入**IICRWD**後，**FU_EN**寫入01h，即送出Data。收到ACK後**FU_EN**由硬件清除，並觸發**TxIF**。
- M4. RX：**FU_EN**寫入01h後，SCL開始送出CLK，並將Data收進**IICRWD**，收到ACK後**FU_EN**由硬件清除，並觸發**RxIF**及中斷。
- M5. Stop：**FU_EN**寫入11h後，送出Stop，收到ACK後**FU_EN**由硬件清除，並觸發**MSIF**及中斷。

- S1. 當slave硬件偵測到Start condition及device ID，**Match** bit由硬件設爲1。
- S2. 收到Device ID(及RW)或Data，硬件會接著傳出**TxAK**；**RxIF**會由硬件設爲1，並觸發中斷，軟件必須寫**FU_EN**=01，用來release bus。
- S3. 每次Data寫入**IICRWD**後，**FU_EN**寫入01，即會將Data送出。
- S4. 每次Data完成送出後，**FU_EN**由硬件清除爲0，**TxIF**由硬體設置爲1，並觸發中斷，**TxIF**須由軟體清除爲0。
- S5. 當收到N-ACK時，Slave即停止Data回傳。在收到Stop condition，**MPIF**由硬件設爲1，**MPIF**須由軟體清除爲0。



5.3 MCU IIC 完整範例程序(master)：

Description	MCU master w/r EEPROM(24cxx):
Main program	

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

//==============================
//IIC DEFINITIONs
//==============================
#define  d_DEVICE_ID1      0x60              // user modify
#define  d_DEVICE_ID2      0xA0              // user modify

#define d_IIC_ADR_CA1      0x00
#define d_IIC_ADR_CA2      0x20
#define  d_IIC_ADR_SEL     d_IIC_ADR_CA1     // user modify, Device ID select

#define d_BR32                 0x00
#define d_BR64                 0x01
#define d_BR128                0x02
#define d_BR256                0x03
#define d_BR512                0x04
#define d_BR1024           0x05
#define d_BR2048           0x06
#define d_BR4096           0x07
#define d_IIC_BR           d_BR32                // user modify    //691.2->588

#define d_MASTER           0x40
#define d_SLAVE                0x00
#define d_MODE_SEL             d_MASTER          // use master mode
#define d_IIC_EN           0x80
#define  d_NACK                1
#define  d_ACK                 0
#define  d_WRITE               0
#define  d_READ                1

#define d_BB_DIS           0x00
#define d_BB_EN                0x08
#define d_BUS_BUSY             d_BB_EN
#define d_AR_DIS           0x00
#define d_AR_EN                0x10
#define d_ARBITRATION      d_AR_EN

#define  d_CMD_RW          0x40
#define  d_CMD_Start           0x80
#define  d_CMD_Stop            0xC0

//=================================================================
//OWNED SUBROUTINES
//=================================================================

void IIC_interrupt(void) interrupt d_IIC_Vector
{
    if(TXIF)
    {
        TXIF = 0;                      // Clear interrupt flag
    }
    if(RXIF)
    {
        RXIF = 0;                      // Clear interrupt flag
```

```
    }
    if(MPIF)
    {
        MPIF=0;                         // Clear interrupt flag
    }

}

void IIC_Init_master(void)
{
    IICA1     = d_DEVICE_ID1;
    IICA2     = d_DEVICE_ID2;
    IICS  = 0x00;               // Clear IIC all status
    IEN0 |= 0x80;               // Enable interrupt All
    IEN1 |= 0x20;               // Enable interrupt IIC
    IICCTL    = d_IIC_EN | d_MODE_SEL | d_ARBITRATION | d_BUS_BUSY | d_IIC_ADR_SEL |
d_IIC_BR;
}

void IIC_Disable(void)
{
    IICCTL    = 0x00;               // Disable IIC all function
    IEN1 &= 0xDF;               // Disable interrupt IIC
    IICS  = 0x00;               // Clear IIC all status
}

//=================================================================
//This function will send out the Start or Re-Start pulse and CONTROL byte.
//=================================================================
bit IIC_SendStart(unsigned char ControlByte)
{
    IICA2    = ControlByte;         // MAS = 1
    IICA1    = ControlByte;         // MAS = 0

    IICEBT = d_CMD_Start;       // generate a start condition
    while(IICEBT != 0x00)
    {
        ;
    }
    return RXAK;
}

//=================================================================
//This function will send out the Stop pulse
//=================================================================
void IIC_SendStop(void)
{
    IICEBT = d_CMD_Stop;            // generate a stop condition
    while(IICEBT != 0x00)           // waiting data recive finish
    {
        ;
    }
}
//=================================================================
//This function will wait and receive one byte, then feedback ACK or NACK.
//=================================================================
unsigned char IIC_ReceiveByte(bit ACKStatus)
{
    unsigned char temp;
    TXAK = ACKStatus;           // Feedback ACK/d_NACK to slave
    IICEBT   = d_CMD_RW;                // Ready for receive

    while(IICEBT != 0x00)           // waiting data recive finish
```

```
    {
        ;
    }
    temp = IICRWD;
    //Delay(100);                  // finetune by user
    return(temp);
}

void IIC_TransmitByte(unsigned char TxData)
{
    IICRWD   = TxData;                  // load data
    IICEBT   = d_CMD_RW;                // trans. data
    while(IICEBT != 0x00)          // waiting data trans. finish
    {
        ;
    }
    //Delay(100);                  // finetune by user
}

void IIC_Page_Write( char Crtl_byte, unsigned char Addr, char LEN)
{
    unsigned char counter=0;

    IIC_SendStart(Crtl_byte | d_WRITE);             // -- Send CONTROL byte --

    if(RXAK==d_ACK)
    {
        IIC_TransmitByte( Addr );                   // -- Send ADDRESS --

        for (counter=0; counter<LEN; counter++)     // -- Write N bytes --
        {
            if(RXAK==d_ACK)
            {
                IIC_TransmitByte(Addr + counter);   // -- Transmit one byte --
            }
            else
            {
                ;                                       // error process
            }

        }
    }
    else
    {
        ;                                               // error process
    }
    IIC_SendStop();                                     // -- Send STOP -- MStart= 0
}

void IIC_Random_Read( char Crtl_byte, unsigned char Addr, char LEN)
{
    unsigned char Temp[16];
    unsigned char counter=0;

    IIC_SendStart(Crtl_byte | d_WRITE);             // -- Send CONTROL byte --
    if(RXAK==d_ACK)
    {
        IIC_TransmitByte(Addr);                         // -- Send ADDRESS --

        if(RXAK==d_ACK)
        {
            IIC_SendStart(Crtl_byte | d_READ);      // -- Send CONTROL byte --
```

```
                for(counter=0; counter<LEN; counter++)  // -- Read N byte --
                {
                     if(RXAK==d_ACK)
                     {
                          if(counter<(LEN-1))
                          {
                               Temp[counter] = IIC_ReceiveByte(d_ACK); // Receive one byte
                          }
                          else
                          {
                               Temp[counter] = IIC_ReceiveByte(d_NACK);// Receive last byte
                          }
                     }
                     else
                     {
                          ;                                        // error process
                     }
                }
           }
           else
           {
                ;                                                  // error process
           }
     }
     else
     {
          ;                                                        // error process
     }
     IIC_SendStop();                                               // -- Send STOP -- MStart= 0

}

void Check_Arbitration(void)        // multi-master use
{
     // arbitration lost, set by H/W
     // clear by S/W
     while(LAIF)
     {
          LAIF = 0;
     }
}

void Check_Busy(void)               // multi-master use
{
     // Bus busy, BB set by H/W
     // Bus ready, BB clear by H/W or S/W
     while(BB)
     {
          ;
     }
}

void Delay(unsigned int NTime)
{
     unsigned int      i, j;
     for(i=0; i<NTime; i++)
     {
          for(j=0; j<100; j++)
          {
               ;
          }
     }
}
```

```
#define waitslave 10
void main()
{
     IIC_Init_master();

     Check_Arbitration();                    // multi-master use
     Check_Busy();                           // multi-master use

     while(1)
     {
          IIC_Page_Write(d_DEVICE_ID1,  0x00, 3);
          Delay(waitslave);                       // finetune by user
          IIC_Random_Read(d_DEVICE_ID1, 0x00, 3);
          Delay(waitslave);                       // finetune by user
          IIC_Page_Write(d_DEVICE_ID1,  0x03, 3);
          Delay(waitslave);                       // finetune by user
          IIC_Random_Read(d_DEVICE_ID1, 0x03, 3);
          Delay(waitslave);                       // finetune by user
          IIC_Random_Read(d_DEVICE_ID1, 0x00, 6);
          Delay(waitslave);
     }
}
```

5.4 MCU IIC 完整範例程序(slave)：

Description	MCU be slave as IIC-EEPROM:

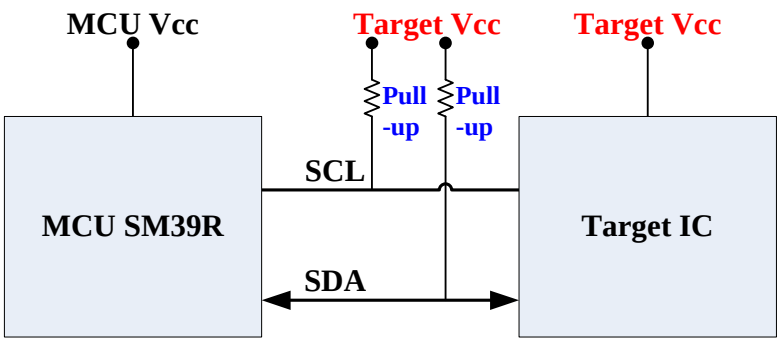
```
//=====================================================================
//
//              S Y N C M O S  T E C H N O L O G Y
//
//=====================================================================
// Test condition:
// Slave MCU Crystal 4~25MHz, pull-up 2.2KR
// Master SCL=100KHz and 400KHz test ok
//=====================================================================
#include "SM39R16A2.h"
#include "IIC.h"

//=====================================================================
#define  d_DEVICE_ID1      0xA0          // user modify
#define  d_DEVICE_ID2      0x60          // user modify

#define  d_ACK                  0
#define  d_NACK                 1
#define  d_Write                0
#define  d_Read                 1
#define  d_null                 0
#define  d_CMD_RW          0x40
#define d_SAVE_ADDR             1
#define d_SAVE_DATA             2

//=====================================================================
bit bACK_last = d_NACK;
unsigned char n_Next_Step  = d_null;
unsigned char n_RW              = d_null;
unsigned char n_Addr       = d_null;
unsigned char n_DAT[16];

//=====================================================================
void IIC_interrupt() interrupt d_IIC_Vector
{
```

```
    //slave rx=============================================================
    if(RXIF)
    {
        //=====================================================
        if ((IICA1 & 0x01) || (IICA2 & 0x01))   // match
        {
            n_Next_Step   = d_SAVE_ADDR;            // set next step
            n_RW          = RW;                     // save RW status
            if (n_RW == d_Write)                // don't release if read phase
                IICEBT    = d_CMD_RW;               // IIC bus release if write phase
        }

        //=====================================================
        else if (n_Next_Step == d_SAVE_ADDR)
        {
            n_Next_Step   = d_SAVE_DATA;            // set next step
            n_Addr        = IICRWD;                 // save addr offset
            IICEBT        = d_CMD_RW;               // IIC bus release
        }

        //=====================================================
        else if (n_Next_Step == d_SAVE_DATA)
        {
            n_Next_Step       = d_SAVE_DATA;        // set next step
            n_DAT[n_Addr++] = IICRWD;           // save data
            IICEBT            = d_CMD_RW;           // IIC bus release
        }
        //=====================================================
        RXIF = 0;
    }

    //slave tx=============================================================
    if (TXIF)
    {
        TXIF = 0;                       // Clear interrupt flag
    }
    //=====================================================================
}

void IIC_init_slave()
{
    IFCON    |= 0x80;              // MCU 1T
    IICS = 0x00;                   // init IICS
    IRCON    = 0x00;                   // init IRCON
    IICEBT   = d_CMD_RW;               // IIC bus ready
    IICA1    = d_DEVICE_ID1;           // Control Byte 1
    IICA2    = d_DEVICE_ID2;           // Control Byte 2
    IEN1     |= 0x20;              // Enable interrupt IIC
    IEN0 |= 0x80;              // Enable interrupt All
    IICCTL    = 0x80;              // Enable IIC module, slave mode, use IICA1
}

void IIC(void)
{

    //slave tx=============================================================
    if(n_RW==d_Read)
    {
        //slave tx get stop=================================
        if(MPIF)                            // get Stop condition
        {
            n_RW = d_null;                          // break while(n_RW==d_Read)
```

```
                MPIF = 0;
            }

            //=========================================================
            else
            {
                //slave tx finish=================================
                if((RXAK == d_NACK)&&(bACK_last==d_ACK))// End if RXAK recive NACK
                {
                    bACK_last    = d_NACK;
                }

                //slave tx on going===============================
                else if(MPIF==0)
                {
                    bACK_last    = d_ACK;
                    IICRWD   = n_DAT[n_Addr++];
                    IICEBT   = d_CMD_RW;               // IIC bus ready

                    while(IICEBT==d_CMD_RW)            //IICEBT will auto clear by HW
                    {
                        ;
                    }
                }
                //================================================
            }
        }
        //slave rx=============================================================
        else if(n_RW==d_Write)
        {
            //slave rx finish=====================================
            if(MPIF)                                  // get Stop condition
            {
                //slave rx get stop===============================
                //slave finish recive data, here user process data, example n_DAT[].....
                TXAK = d_NACK;                            //TX NACK if slave busy
                MPIF = 0;                                 // clear MPIF if finish process
                TXAK = d_ACK;
            }
        }
        //=====================================================================
}
void main(void)
{
    IIC_init_slave();

    while(1)
    {
        IIC();
    }//while(1)
}//end of main
```

6 不同工作電壓的 IIC 通訊應用：

IIC 使用說明

圖示	說明
	**HW:** 1. Bus 必須接上拉(pull-up)電阻，電阻值須視實際應用調整。 2. 必要時調降 MCU Vcc 可降低 MCU VIH/VIL，但仍須符合 MCU 工作電壓 2.7~5.5V 規範。 3. 建議 **Target Vcc> MCU Vcc*60%**。 (Spec. ideal MCU Vcc=3.3V：VIH=1.9V，VIL=0.7V) (Spec. ideal MCU Vcc=5V：VIH=2.7V，VIL=1V) **SW:** 1. 先設置 SM39R 的 GPIO 設置爲 open drain。 2. enable IIC function。