IO EXPANDER PCA9555

1. 簡介

我想要介紹一下用 MCU 來控制外部 IO 的方式，很多的產品在設計上可以會碰到像是 IO 不夠用的問題，那麼透過兩條線的 IIC 就可以連接許多顆這樣的 IO EXPANDER Ic 來達成 IO 不足的缺陷，如果用 4094 的方式也可以做到，但是透過串接多顆的話就會造成時序 CLK 會稍微長一點，因為 IIC 的連接方式可以帶有 ADDRESS 控制，因此我可以控制第 1 顆或是第 5 顆是可以任意選擇的，而且時序上的反應以這程式當初我在驗證上，是不會超過 200mSec 的，因此我會比較推荐這一顆，但是價格會比較貴一點點，以目前我找到的報價 1K 約是在 1.5 – 2 美元左右，一顆是 16 PIN，I/O 可以做選擇，並且可以搭配 INT 連接到 MCU 輸入·

2. 運作原理

GPIO 的控制由 IIC 來傳輸 COMMAND，一顆裡面有兩組 8 BIT 的 IO，因此分為以下控制命令：

```
#define PCA9555_CM_IN_P0                           0x00
#define PCA9555_CM_IN_P1                           0x01
#define PCA9555_CM_OUT_P0                          0x02
#define PCA9555_CM_OUT_P1                          0x03
#define PCA9555_CM_Polarity_Inversion_P0           0x04
#define PCA9555_CM_Polarity_Inversion_P1           0x05
#define PCA9555_CM_Config_P0                       0x06
#define PCA9555_CM_Config_P1                       0x07
```

共有 7 個 BYTE 來當設定暫存器就是 SFR，主要的設定是設為 IN 與 OUT，0x06 與 0x07 位址在設定高阻輸入與輸出，控制時要先設定 Config 的 PIN 做為高阻或輸出，再給 IN 或是 OUT 的命令，而後接著傳 PORT DATA·

透過設定 `PCA9555_CM_IN_Px` 或是設定 `PCA9555_CM_OUT_Px` 來控制 `pin` 腳為 `HI` 或是 `LO`，設定方式在以下程式部份會描述，

2. 1 運作原理 - Spec 部份參考：

REGISTERS

Command Byte

Command	Register
0	Input port 0
1	Input port 1
2	Output port 0
3	Output port 1
4	Polarity inversion port 0
5	Polarity inversion port 1
6	Configuration port 0
7	Configuration port 1

The command byte is the first byte to follow the address byte during a write transmission. It is used as a pointer to determine which of the following registers will be written or read.

Registers 0 and 1 — Input Port Registers

bit	I0.7	I0.6	I0.5	I0.4	I0.3	I0.2	I0.1	IO.0
default	X	X	X	X	X	X	X	X
bit	I1.7	I1.6	I1.5	I1.4	I1.3	I1.2	I1.1	I1.0
default	X	X	X	X	X	X	X	X

This register is an input-only port. It reflects the incoming logic levels of the pins, regardless of whether the pin is defined as an input or an output by Register 3. Writes to this register have no effect.

The default value 'X' is determined by the externally applied logic level.

Registers 2 and 3 — Output Port Registers

bit	O0.7	O0.6	O0.5	O0.4	O0.3	O0.2	O0.1	O0.0
default	1	1	1	1	1	1	1	1
bit	O1.7	O1.6	O1.5	O1.4	O1.3	O1.2	O1.1	O1.0
default	1	1	1	1	1	1	1	1

This register is an output-only port. It reflects the outgoing logic levels of the pins defined as outputs by Register 6 and 7. Bit values in this register have no effect on pins defined as inputs. In turn, reads from this register reflect the value that is in the flip-flop controlling the output selection, NOT the actual pin value.

Registers 4 and 5 — Polarity Inversion Registers

bit	N0.7	N0.6	N0.5	N0.4	N0.3	N0.2	N0.1	N0.0
default	0	0	0	0	0	0	0	0
bit	N1.7	N1.6	N1.5	N1.4	N1.3	N1.2	N1.1	N1.0
default	0	0	0	0	0	0	0	0

This register allows the user to invert the polarity of the Input Port register data. If a bit in this register is set (written with '1'), the Input Port data polarity is inverted. If a bit in this register is cleared (written with a '0'), the Input Port data polarity is retained.

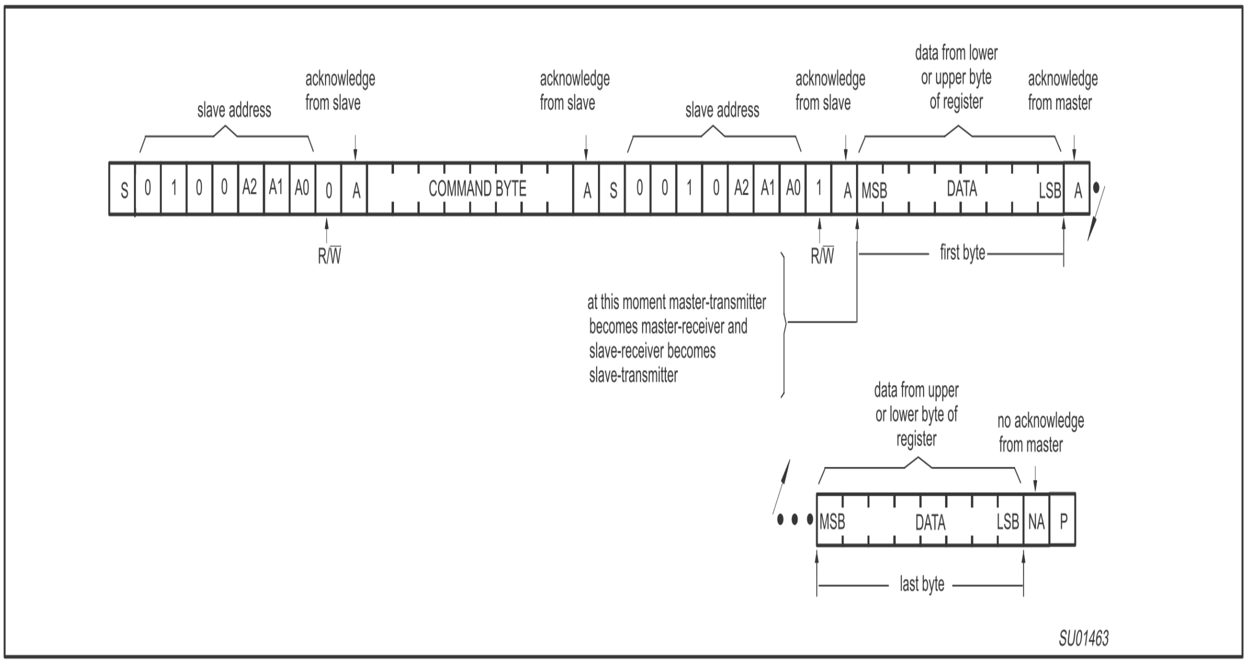
Registers 6 and 7 — Configuration Registers

bit	C0.7	C0.6	C0.5	C0.4	C0.3	C0.2	C0.1	C0.0
default	1	1	1	1	1	1	1	1
bit	C1.7	C1.6	C1.5	C1.4	C1.3	C1.2	C1.1	C1.0
default	1	1	1	1	1	1	1	1

This register configures the directions of the I/O pins. If a bit in this register is set (written with '1'), the corresponding port pin is enabled as an input with high impedance output driver. If a bit in this register is cleared (written with '0'), the corresponding port pin is enabled as an output. Note that there is a high value resistor tied to V_{DD} at each pin. At reset the device's ports are inputs with a pull-up to V_{DD}.

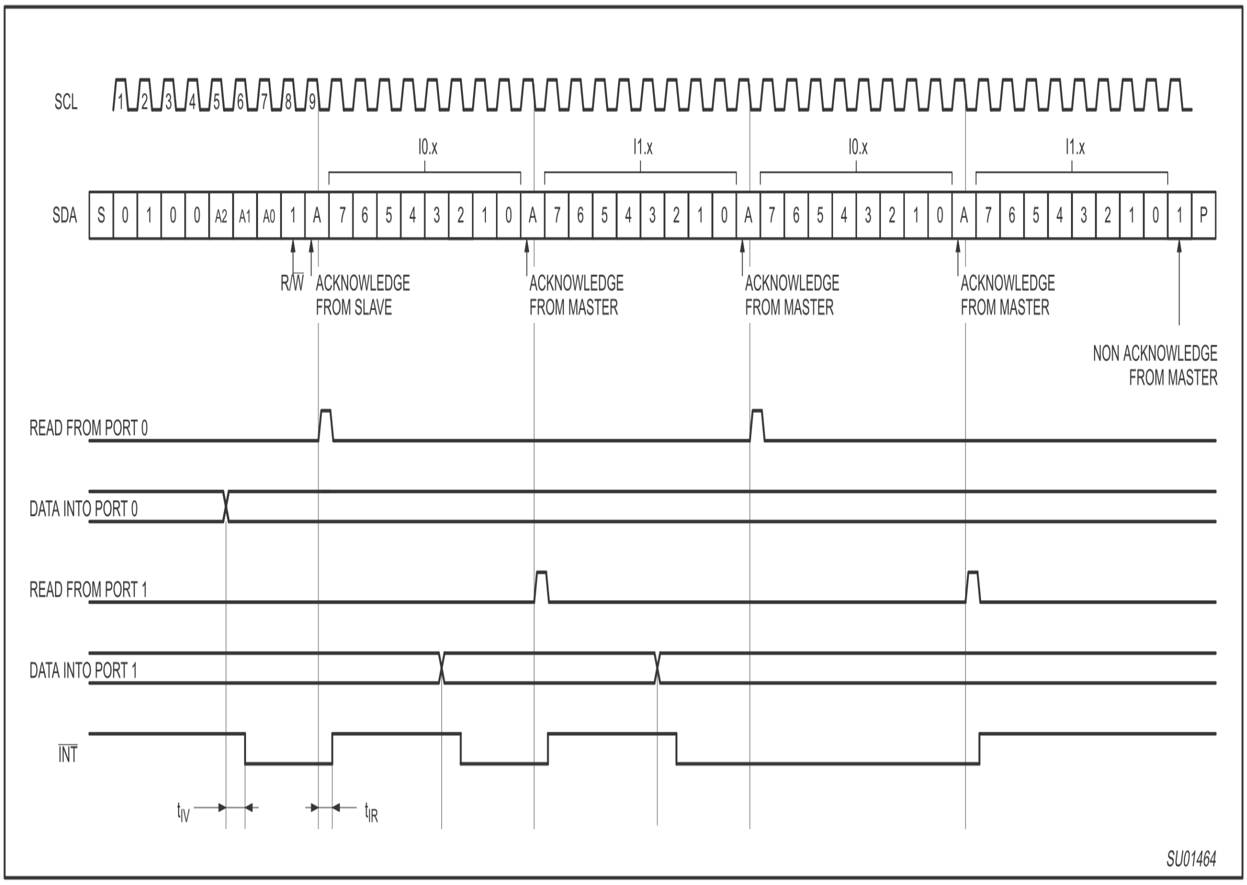
POWER-ON RESET

When power is applied to V_{DD}, an internal power-on reset holds the PCA9555 in a reset condition until V_{DD} has reached V_{POR}. At that point, the reset condition is released and the PCA9555 registers and SMBus state machine will initialize to their default states. The power-on reset typically completes the reset and enables the part by the time the power supply is above V_{POR}. However, when it is required to reset the part by lowering the power supply, it is necessary to lower it below 0.2 V.


slave address
acknowledge from slave
acknowledge from slave
slave address
acknowledge from slave
data from lower or upper byte of register
acknowledge from master
S 0 1 0 0 A2 A1 A0 0 A
COMMAND BYTE
A S 0 0 1 0 A2 A1 A0 1 A MSB DATA LSB A
R/$\overline{W}$
R/$\overline{W}$
first byte
at this moment master-transmitter becomes master-receiver and slave-receiver becomes slave-transmitter
data from upper or lower byte of register
no acknowledge from master
MSB DATA LSB NA P
last byte
SU01463

NOTE: Transfer can be stopped at any time by a STOP condition.

Figure 8. READ from register



NOTES: Transfer of data can be stopped at any moment by a STOP condition. When this occurs, data present at the latest acknowledge phase is valid (output mode).
It is assumed that the command byte has previously been set to 00 (read input port port register).

Figure 9. READ input port register — scenario 1

在我第一篇文章中有介紹過硬體的 IIC 控制 EEPROM，而 PCA9555 的控制封包與 EEPROM 不太相同，標準的 IIC 封包是先給裝置位置，EEPROM 是 A0 然後再加上 PAGE 頁碼以及讀寫碼，再就是給要讀寫的記憶體位置，之後就是跟隨資料讀或是寫的程序，但是 PCA9555 並不是，一開始也是給裝置位置 0x40 再加上 A0-A2 位置(一組 IIC 共可以接 8 顆裝置)再加上一位讀寫碼，之後就是給 COMMAND 就是上一頁介紹的 COMMAND 位置，共有 0-7 個控制碼，控制碼之後接著傳送控制的資料，最後才是資料 DATA 的讀寫部份·

在資料讀寫部份是可以連續送的，所以 COMMAND 給 P0 的命令後，可以繼續送 P1 的資料，但是我的寫法是透過 MARCO 單獨去設 I/O 再去送值，所以我是分開讀寫·

3. 程式導讀－主程式：

```
void main(void)
{
            // 寫入部份
            temp[0]=PCA9555_CM_Config_P0;
            // 這裡設定 P0 不用 PULL-HI，並且為 OUYPUT.
            temp[1]=0x00;      // P0 有兩組 8Bit 的 I/O
            temp[2]=0x00;
            PCA9555_DEVICE_Write(&temp,3);

            LED_NTSC1(ON),LED_NTSC2(OFF),LED_PAL1(OFF);
            LED_PALN(OFF),LED_PAL2(OFF),LED_PALM(OFF);
            // 這裡是用 MACRO 來把 I/O 帶的結構 I/O 定義變成
            // 直接的 PCA9555_LED_PORT.p0&=~0x01 這樣一行
            // 透過這樣的巨集寫法實際上在編譯後我在 KEIL C 上看到是與
            // ASM 1:1 的行數置換，因此這部份執行效率與 ASM 等效
            temp[0]=PCA9555_CM_OUT_P0;
            temp[1]=PCA9555_LED_PORT.p0;
            temp[2]=PCA9555_LED_PORT.p1;
            PCA9555_DEVICE_Write(&temp,3);
            // 這一段就是把命令與資料透過 IIC 去寫入到 PCA9555 中·


            // 讀取部份
            temp[0]=PCA9555_CM_Config_P0; //先設 I/O 為 PULL-HI
            temp[1]=0xFF;
            temp[2]=0xFF;
            PCA9555_DEVICE_Write(&temp,3);
```

```
                temp[0]=PCA9555_CM_IN_P0;
                temp[1]=0xFF;
                temp[2]=0xFF;
                temp[2]=PCA9555_DEVICE_Read(&temp,3);

                printf("\t P0=%d\n ", temp[2])
}
```

3.1 程式導讀 －PCA9555 副程式部份 (定義與巨集)：

```
#define PCA9555_READ    1

#define PCA9555_IIC_SDA_SET(x)            (x)?(P5_OPDN|=0x3F):(P5_OPDN&=~0xFF)
// 這裡介紹一下巨集寫法透過巨集可以把行式左邊的部份直接用右邊的行去替換掉，
// 因此雖然看起來在呼叫是用到 PCA9555_IIC_SDA_SET(1or0);但是實際編譯器會轉成行數，
// 就代換成右邊行式的替換原本的呼叫行，而用?:可以選擇 x 為真就用:左邊換假用右邊換.

#define PCA9555_IIC_SDA(x)                (x)?(P5_DATA|=0x40):(P5_DATA&=~0x40)
#define PCA9555_IIC_SCL(x)                (x)?(P5_DATA|=0x80):(P5_DATA&=~0x80)
#define PCA9555_IIC_SDA_READ()            (P5_DATA&0x40)

#define PCA9555_IIC_Delay_Time 0x01
#define PCA9555_DEVID 0x4E

#define PCA9555_CM_IN_P0                          0x00
#define PCA9555_CM_IN_P1                          0x01
#define PCA9555_CM_OUT_P0                         0x02
#define PCA9555_CM_OUT_P1                         0x03
#define PCA9555_CM_Polarity_Inversion_P0          0x04
#define PCA9555_CM_Polarity_Inversion_P1          0x05
#define PCA9555_CM_Config_P0                      0x06
#define PCA9555_CM_Config_P1                      0x07

typedef unsigned char BYTE;
typedef bit BOOL;

typedef struct PCA9555_LED_PO_OUT
{
      unsigned char p0;
```

```
        unsigned char p1;
}PCA9555_LED_POSTR;

// I/O 使用結構的寫法是 ARM Cortex BSP 常用的 MEMORY MAPPED 的方式,

// 將實際對應到 9555 的接腳電路上的定義與巨集定義相同,這樣比較好寫也較不容易錯.
extern xdata PCA9555_LED_POSTR PCA9555_LED_PORT;


#define LED_POWER(x) x?(PCA9555_LED_PORT.p0&=~0x01,PCA9555_LED_PORT.p1=~0x00):
(PCA9555_LED_PORT.p0=~0x00,PCA9555_LED_PORT.p1=~0x00)
#define LED_VGA(x) x?(PCA9555_LED_PORT.p0&=~0x02):(PCA9555_LED_PORT.p0|=0x02)
#define LED_SV(x) x?(PCA9555_LED_PORT.p0&=~0x04):(PCA9555_LED_PORT.p0|=0x04)
#define LED_AV(x) x?(PCA9555_LED_PORT.p0&=~0x08):(PCA9555_LED_PORT.p0|=0x08)

#define LED_NTSC1(x) x?(PCA9555_LED_PORT.p0&=~0x10):(PCA9555_LED_PORT.p0|=0x10)
#define LED_NTSC2(x) x?(PCA9555_LED_PORT.p0&=~0x20):(PCA9555_LED_PORT.p0|=0x20)

#define LED_PAL1(x) x?(PCA9555_LED_PORT.p0&=~0x40):(PCA9555_LED_PORT.p0|=0x40)
#define LED_PALN(x) x?(PCA9555_LED_PORT.p0&=~0x80):(PCA9555_LED_PORT.p0|=0x80)

#define LED_PAL2(x) x?(PCA9555_LED_PORT.p1&=~0x01):(PCA9555_LED_PORT.p1|=0x01)
#define LED_PALM(x) x?(PCA9555_LED_PORT.p1&=~0x02):(PCA9555_LED_PORT.p1|=0x02)
#define LED_ALL(x) x?(PCA9555_LED_PORT.p0=0x00,PCA9555_LED_PORT.p1=0x00):
(PCA9555_LED_PORT.p0=~0x00,PCA9555_LED_PORT.p1=~0x00)
// 定義電路上實際 9555 的 I/O 名稱為巨集.

// ******************************************************************************
//                                       Functions
// ******************************************************************************
extern void PCA9555_DEVICE_Write(BYTE *,BYTE );

#if PCA9555_READ
extern BYTE PCA9555_DEVICE_Read(BYTE *,BYTE );
#endif
```

3.2 程式導讀 – PCA9555 副程式部份

```
// ================================================================================
// Name        : PCA9555_IIC_Delay
// Description :
// Parameters  :
// Return Value: None
// See Also     : ALL
// ===========================================================================
void PCA9555_IIC_Delay(void)
{
 idata unsigned char n;

 n=PCA9555_IIC_Delay_Time;
 while(n--)
      {
      _nop_();
      }
}
// 這部份的時序要靠輸出一個 I/O 用視波器抓看看，我學長教我的，用 C 語言編譯的話建議看一下.

// ===========================================================================
// Name         : PCA9555_IIC_Start
// Description :
// Parameters   : None
// Return Value: None
// See Also     : ALL
// ===========================================================================
void PCA9555_IIC_Start(void)
{
      PCA9555_IIC_SCL(HI); PCA9555_IIC_SDA(HI);
      PCA9555_IIC_Delay();
      PCA9555_IIC_SDA(LO); PCA9555_IIC_SCL(HI);
      PCA9555_IIC_Delay();
      PCA9555_IIC_SDA(LO); PCA9555_IIC_SCL(LO);
      PCA9555_IIC_Delay();
}
// IIC 的資料送由 START 開始發，最後 STOP 結束.

// ===========================================================================
// Name         : IIC_Stop
// Description :
// Parameters   : None
// Return Value: None
// See Also     : ALL
// ===========================================================================
void PCA9555_IIC_Stop(void)
{
      PCA9555_IIC_SDA(LO); PCA9555_IIC_SCL(LO);
      PCA9555_IIC_Delay();
      PCA9555_IIC_SDA(LO); PCA9555_IIC_SCL(HI);
      PCA9555_IIC_Delay();
      PCA9555_IIC_SDA(HI); PCA9555_IIC_SCL(HI);
      PCA9555_IIC_Delay();
}

#if PCA9555_READ
void PCA9555_IIC_Acknowledge(BOOL ack)
{
      PCA9555_IIC_SDA(LO); PCA9555_IIC_SCL(LO);
      PCA9555_IIC_SDA_READ(ack ? 0:1);  PCA9555_IIC_Delay();
      PCA9555_IIC_SCL(HI);  PCA9555_IIC_Delay();
      PCA9555_IIC_SCL(LO);  PCA9555_IIC_Delay();
}
#endif
// 讀取 ACK 的方式，因為 ACK 是 1 所以我再轉一下.


// ==============================================================================
Name        :
Description :
Parameters  :
Return Value:
See Also    :
// ==============================================================================
BOOL PCA9555_IIC_Set_Data(unsigned char Data)
{
 idata unsigned char i,Byte;
 BOOL ACK;
 Byte=Data;
  for(i=0; i<8; i++)
  {
   PCA9555_IIC_SDA(Byte&0x80); PCA9555_IIC_SCL(LO);
   PCA9555_IIC_Delay();
   PCA9555_IIC_SCL(HI); PCA9555_IIC_Delay();

   Byte<<=1;

   PCA9555_IIC_SCL(LO); PCA9555_IIC_Delay();

  }
      PCA9555_IIC_SDA(HI); PCA9555_IIC_SCL(LO);
      PCA9555_IIC_Delay();
      PCA9555_IIC_SCL(HI); PCA9555_IIC_Delay();
      ACK=(PCA9555_IIC_SDA_READ() ? 0:1); PCA9555_IIC_Delay();
      PCA9555_IIC_SCL(LO); PCA9555_IIC_Delay();
      PCA9555_IIC_SDA(LO); PCA9555_IIC_Delay();

 return ACK;
}
// 傳送資料 8 位碼

// ==============================================================================
// Name        :
// Description :
// Parameters  :
// Return Value:
// See Also    :
// ==============================================================================
#if PCA9555_READ
BYTE PCA9555_IIC_Get_Data(void)
{
 idata unsigned int i,Byte;

  Byte=0;

   PCA9555_IIC_SDA(HI), PCA9555_IIC_Delay();
   for(i=0; i<8; i++)
   {
    PCA9555_IIC_SCL(HI), PCA9555_IIC_Delay();
    Byte<<=1;
    Byte|=PCA9555_IIC_SDA, PCA9555_IIC_Delay();
    PCA9555_IIC_SCL(LO), PCA9555_IIC_Delay();
   }
      PCA9555_IIC_Acknowledge(1);
      return Byte;
}
#endif

// ================================================================================
// Name        :
// Description :
// Parameters  :
// Return Value:
// See Also    :
// ================================================================================
void PCA9555_DEVICE_Write(BYTE *IIC_Data,BYTE Length)
{
 BYTE Byte;

 Byte=Length;

 PCA9555_IIC_SDA_SET(HI);

 PCA9555_IIC_Start();
 if(!(PCA9555_IIC_Set_Data(PCA9555_DEVID)))     // Detect ACK
      {
      #if (JK_DEBUG_EN)
      Print_Num(1, 7, PCA9555_DEVID, TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 NONE ACK DEVID",TEXT_COLOR);
      #endif
      return;
      }
 else
      {
      #if (JK_DEBUG_EN)
      Print_Num(1, 7, PCA9555_DEVID, TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 OK ACK DEVID",TEXT_COLOR);
      #endif
      }
```

```
if(!(PCA9555_IIC_Set_Data(IIC_Data[0]))) // Detect ACK
      {
      #if (JK_DEBUG_EN)
      Print_Num(2, 7, IIC_Data[0], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 NONE ACK COMMAND",TEXT_COLOR);
      #endif
      return;
      }
else
      {
      #if (JK_DEBUG_EN)
      Print_Num(2, 7, IIC_Data[0], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 OK ACK COMMAND",TEXT_COLOR);
      #endif
      }

if(!(PCA9555_IIC_Set_Data(IIC_Data[1]))) // Detect ACK
      {
      #if (JK_DEBUG_EN)
      Print_Num(3, 7, IIC_Data[1], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 NONE ACK DATA1",TEXT_COLOR);
      #endif
      return;
      }
else
      {
      #if (JK_DEBUG_EN)
      Print_Num(3, 7, IIC_Data[1], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 OK ACK DATA1",TEXT_COLOR);
      #endif
      }

if(!(PCA9555_IIC_Set_Data(IIC_Data[2]))) // Detect ACK
      {
      #if (JK_DEBUG_EN)
      Print_Num(4, 7, IIC_Data[2], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 NONE ACK DATA2",TEXT_COLOR);
      #endif
      return;
      }
 else
      {
      #if (JK_DEBUG_EN)
      Print_Num(4, 7, IIC_Data[2], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 OK ACK DATA2",TEXT_COLOR);
      #endif
      }
 PCA9555_IIC_Stop();
}

// =============================================================================
// Name        :
// Description :
// Parameters  :
// Return Value:
// See Also    :
// =============================================================================
#if PCA9555_READ
BYTE PCA9555_DEVICE_Read(BYTE *IIC_Data,BYTE Length)
{
 BYTE DATA;

 DATA=Length;

 PCA9555_IIC_SDA_SET(HI);
 PCA9555_IIC_Start();
 if(!(PCA9555_IIC_Set_Data(PCA9555_DEVID)))
      {
      #if (JK_DEBUG_EN)
      Print_Num(1, 7, PCA9555_DEVID, TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 NONE ACK DEVID",TEXT_COLOR);
      #endif
      return 0;
      }
 else
```

```
      {
      #if (JK_DEBUG_EN)
      Print_Num(1, 7, PCA9555_DEVID, TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 OK ACK DEVID",TEXT_COLOR);
      #endif
      }

if(!(PCA9555_IIC_Set_Data(IIC_Data[0]))) // Detect ACK
      {
      #if (JK_DEBUG_EN)
      Print_Num(2, 7, IIC_Data[0], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 NONE ACK COMMAND",TEXT_COLOR);
      #endif
      return 0;
      }
else
      {
      #if (JK_DEBUG_EN)
      Print_Num(2, 7, IIC_Data[0], TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 OK ACK COMMAND",TEXT_COLOR);
      #endif
      }

 if(!(PCA9555_IIC_Set_Data(IIC_Data[1])))  // Detect ACK
       {
 #if (JK_DEBUG_EN)
       Print_Num(2, 7, IIC_Data[1], TEXT_COLOR, HEX_VALUE);
       Print_String(5,7,"PCA9555 NONE ACK COMMAND",TEXT_COLOR);
 #endif
       return 0;
       }
 else
       {
 #if (JK_DEBUG_EN)
       Print_Num(2, 7, IIC_Data[1], TEXT_COLOR, HEX_VALUE);
       Print_String(5,7,"PCA9555 OK ACK COMMAND",TEXT_COLOR);
 #endif
       }
 if(!(PCA9555_IIC_Set_Data(IIC_Data[2])))  // Detect ACK
       {
 #if (JK_DEBUG_EN)
       Print_Num(2, 7, IIC_Data[2], TEXT_COLOR, HEX_VALUE);
       Print_String(5,7,"PCA9555 NONE ACK COMMAND",TEXT_COLOR);
 #endif
       return 0;
       }
 else
       {
 #if (JK_DEBUG_EN)
       Print_Num(2, 7, IIC_Data[2], TEXT_COLOR, HEX_VALUE);
       Print_String(5,7,"PCA9555 OK ACK COMMAND",TEXT_COLOR);
 #endif
       }

 if(!(PCA9555_IIC_Set_Data((PCA9555_DEVID|0x01))))
      {
      #if (JK_DEBUG_EN)
      Print_Num(1, 7, PCA9555_DEVID, TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 NONE ACK DEVID",TEXT_COLOR);
      #endif
      return 0;
      }
else
      {
      #if (JK_DEBUG_EN)
      Print_Num(1, 7, PCA9555_DEVID, TEXT_COLOR, HEX_VALUE);
      Print_String(5,7,"PCA9555 OK ACK DEVID",TEXT_COLOR);
      #endif
      }
DATA=PCA9555_IIC_Get_Data();
Print_Num(3, 7, DATA, TEXT_COLOR, HEX_VALUE);
Print_String(5,7,"KEY DATA",TEXT_COLOR);
PCA9555_IIC_Stop();
return DATA;
```

```
}
#endif
```

4. 程式說明：

 寫入的部份是已經驗證過了，因為之前的電路沒有用到讀的部份，程式是否有什麼問題可能要用的話再驗證一下，在 IIC 的部份我用到很多的巨集，是因為巨集是最單純的用 CODE 去代換呼叫行的，對於通訊的部份甚至是透過引用 ASM 來達成最簡潔的 1:1 行數都是常用的方式，巨集與涵式的差別就是在於速度，一個是透過 PUSH 和 POP 來跳過去執行，一個是直接代換呼叫的部份·