MICROCHIP

Regional Training Centers

PIC 微控制器初學者教育訓練

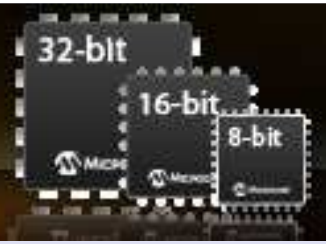
MPLAB X IDE 版本

教材的出版

- **PIC101 Starter RTC** 為台灣繁體中文版本，如有需修改之處，請直接來電:
  - 0800-717718
  - 連絡人: Richard Yang

本教材目前版本為 **v1.0**

教材發行日期 **: 2016/06/01**



本課程為基礎課程，可協助有興趣的學員了解 PIC 微控制器的程式設計及週邊應用

使用 MPLAB X IDE 及 XC8 的程式開發及基本除錯

課程內容

- **Microchip PIC18 MCU** 架構
- **MPLAB X IDE** 基本安裝與使用
- **XC8** 基本介紹
- **PIC101 Lab1.x** 建立新專案
  - PIC101 Lab1.x
  - X IDE 基本除錯
- **PIC** 的工作配置需求的設定
  - PIC101 Lab2.x
  - PIC101 Lab2-1 MCC.x
- 了解 **Timer1** 的工作原理
  - PIC101 Lab3.x
  - PIC101 Lab3 MCC.x
- **PIC18F** 中斷說明 **(**附錄 **A)**
  - PIC101 Lab4.x
  - PIC101 Lab4-1 MCC.x
- **PIC** 特殊功能簡介 **(**附錄 **B)**

PIC101 Starter RTC 教材

- 軟體
  - MPLAB X IDE v3.26 (含) 以上版本
  - MCC v3.05 (含) 以上版本
  - XC8 編譯器 v1.34 (含) 以上版本
- 硬體
  - APP001 v3.a (PIC18F45K22)
  - 9V 電源供應器 或 USB 電纜供電
  - PICKit3 或 ICD3
  - 示波器 (測量時間)

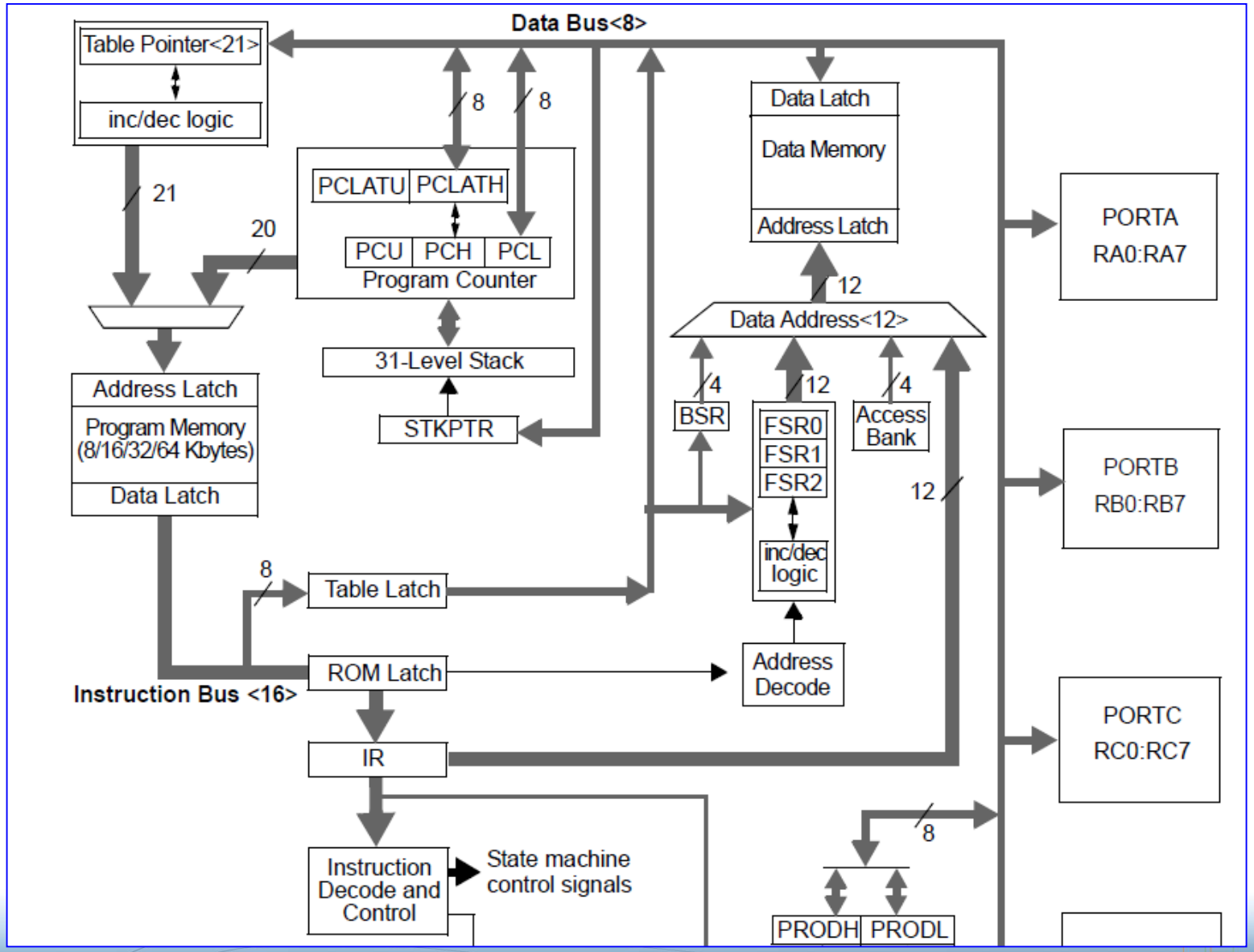
PIC18F45K22 功能

- **32KB Flash Memory**
- **1536 Byte RAM**
- **256 Byte EEPROM**
- 工作電壓: **1.8~5.5V (LF/F)**
- 最高工作頻率: **64MHz**
- 最高執行效能 : **16MIPS**
- 內建高準度 **16MHz** 震盪器
  - 誤差 1%
  - 31KHz ~ 16MHz 可選擇
- 睡眠模式 : **20nA**

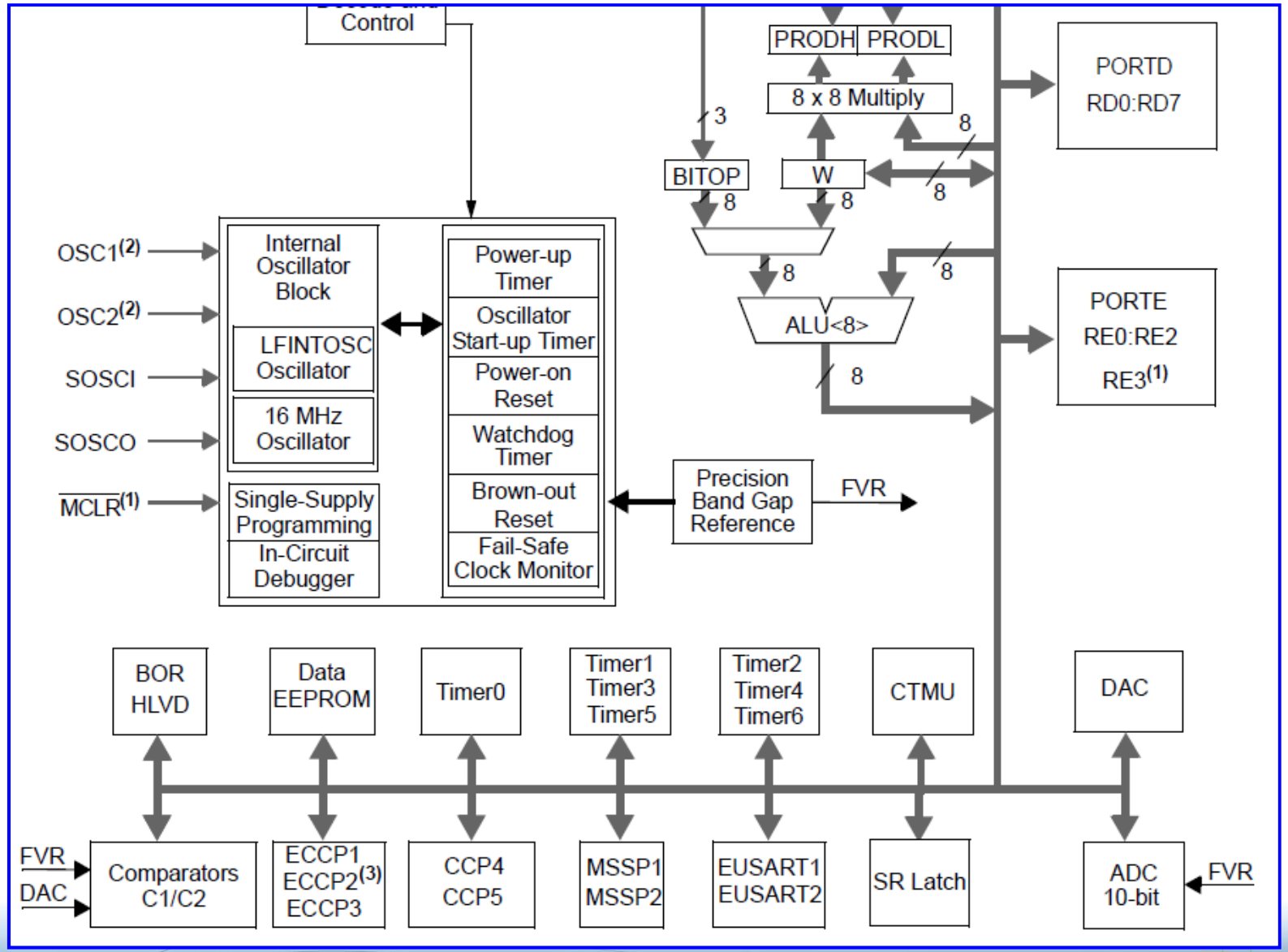
- **ADC : 28** 個輸入
  - PORTA ~ PORTE
- 電壓比較器 : **2**
- 參考電壓源
- **CTMU**

- **CCP x2**
- **ECCP x2**
- **Timer x5**
- **EUART x2**
- **I2C x2**
- **SPI x2**

PIC18F45K22 架構圖(一)

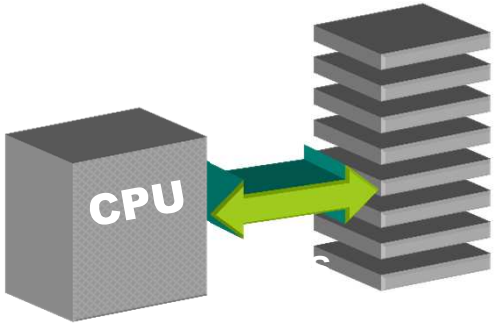


PIC18F45K22 架構圖(二)



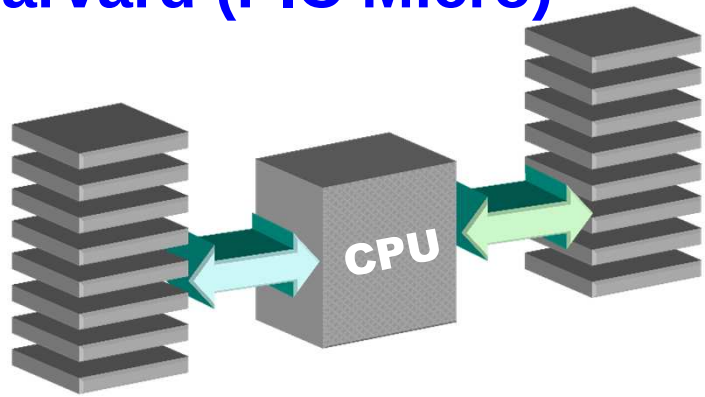
PICmicro® 的架構

Von Neumann (一般MCU)



- 經由相同的記憶體及匯流排來提取程式及存取資料
  - 指令與資料無法有效率的同時被處理
  - 運作效率受到此結構影響而變慢

Harvard (PIC Micro)

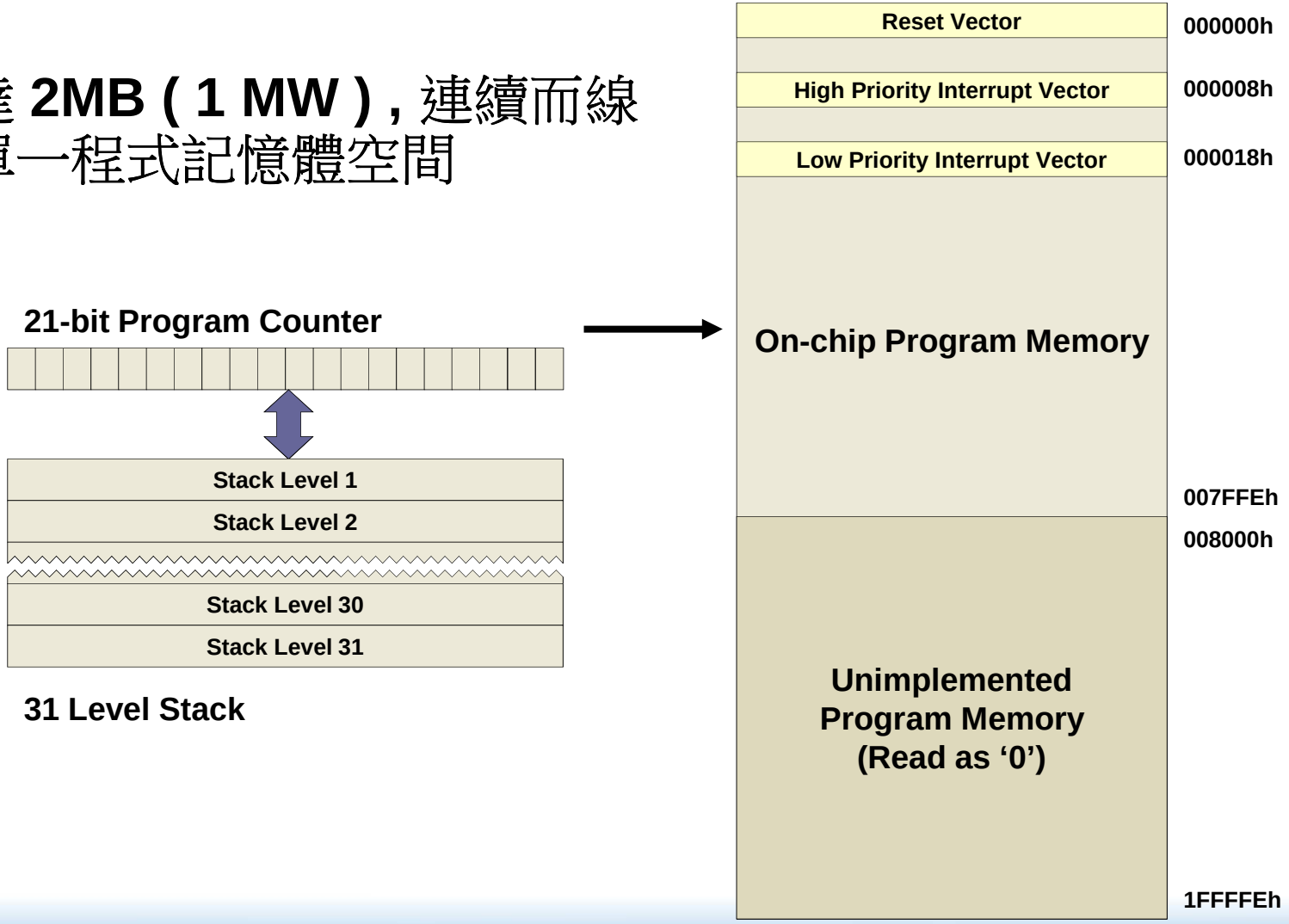


- 使用兩個不同的記憶空間與匯流排來存取程式及存取資料
  - 增加處理資料的效能與執行效率
  - 使得**MCU**可以具有不同寬度的程式記憶體與資料記憶體

Memory

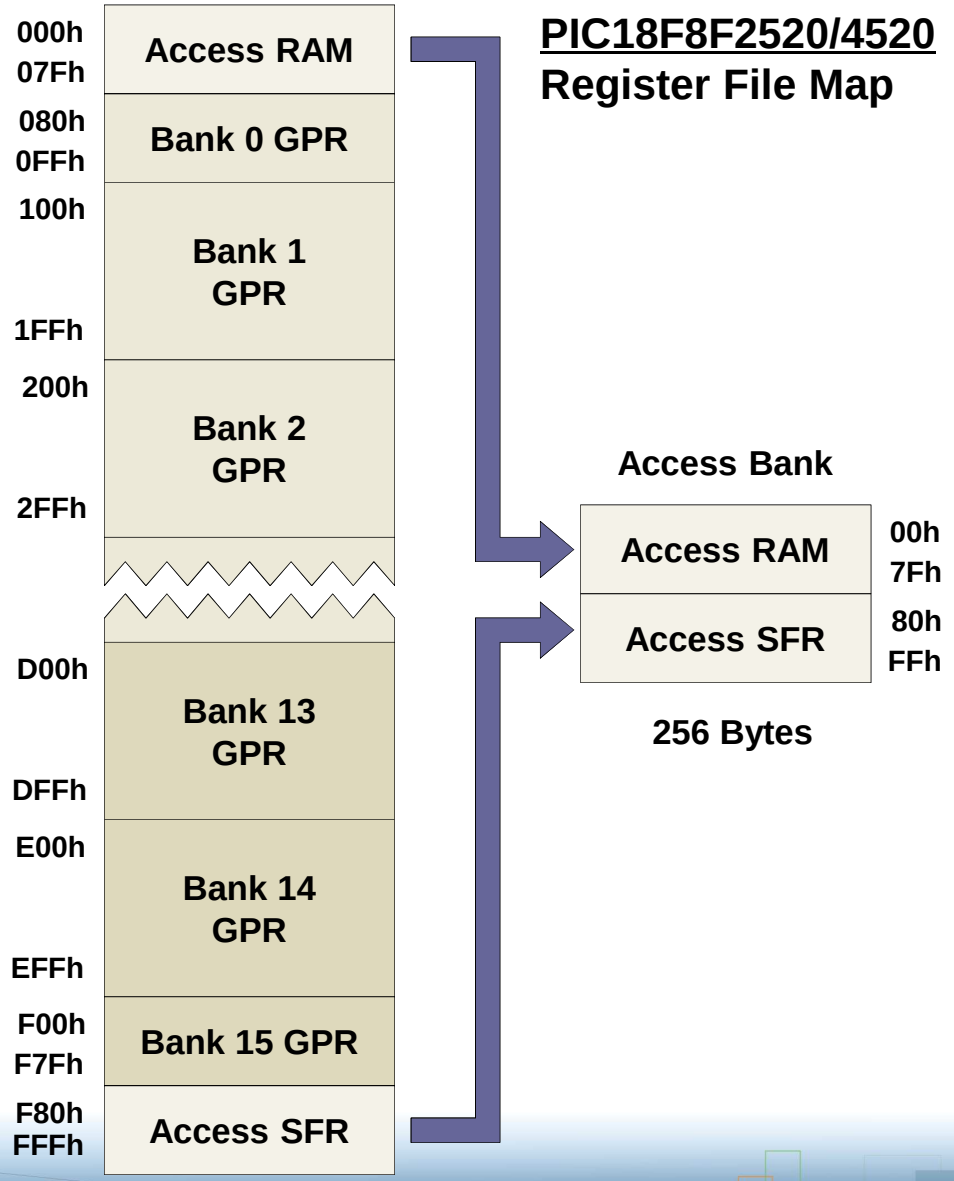
PIC18 Family 程式記憶體架構

- 最高達 **2MB (1 MW)**, 連續而線性的單一程式記憶體空間



Data Memory Organization

- **Data Memory** 最大定址空間達 **4k bytes**
- 使用間接索引定址做線性的存取 **(FSRx)**
- 直接定址以 **256 byte** 為單位。使用 **bank** 的方式來定址全區域 **4KB** 的 **Data Memory** (配合 **BSR** 暫存器)
- **bank 0** 的前段部份與 **bank 15** 的後段部份組合成一個特殊的 **bank** 稱之為 **Access Bank**。
- 使用 **Access Bank** 時，不受 **BSR** 切換的影響。



PIC18 系列標準指令集總覽

Byte Oriented Operations		
addwf	f,d,a	Add WREG and f
addwfc	f,d,a	Add WREG and Carry bit to f
andwf	f,d,a	AND WREG with f
clrf	f,a	Clear f
comf	f,d,a	Complement f
cpfseq	f,a	Compare f with WREG, skip =
cpfsgt	f,a	Compare f with WREG, skip >
cpfslt	f,a	Compare f with WREG, skip <
decf	f,d,a	Decrement f
decfsz	f,d,a	Decrement f, Skip if 0
dcfsnz	f,d,a	Decrement f, Skip if Not 0
incf	f,d,a	Increment f
incfsz	f,d,a	Increment f, Skip if 0
infsnz	f,d,a	Increment f, Skip if Not 0
iorwf	f,d,a	Inclusive OR WREG with f
movf	f,d,a	Move f
movff	f_s,f_d	Move f_s (src) to f_d (dst)
movwf	f,a	Move WREG to f
mulwf	f,a	Multiply WREG with f
negf	f,a	Negate f
rlcf	f,d,a	Rotate Left f through Carry
rlncf	f,d,a	Rotate Left f (No Carry)
rrcf	f,d,a	Rotate Right f through Carry
rrncf	f,d,a	Rotate Right f (No Carry)
setf	f,a	Set f
subfwb	f,d,a	Subtract f from WREG with borrow
subwf	f,d,a	Subtract WREG from f
subwfb	f,d,a	Subtract WREG from f with borrow
swapf	f,d,a	Swap nibbles in f
tstfsz	f,a	Test f, skip if 0
xorwf	f,d,a	Exclusive OR WREG with f

Bit Oriented Operations		
bcf	f,b,a	Bit Clear f
bsf	f,b,a	Bit Set f
btfsc	f,b,a	Bit Test f, Skip if Clear
btfss	f,b,a	Bit Test f, Skip if Set
btg	f,b,a	Bit Toggle f

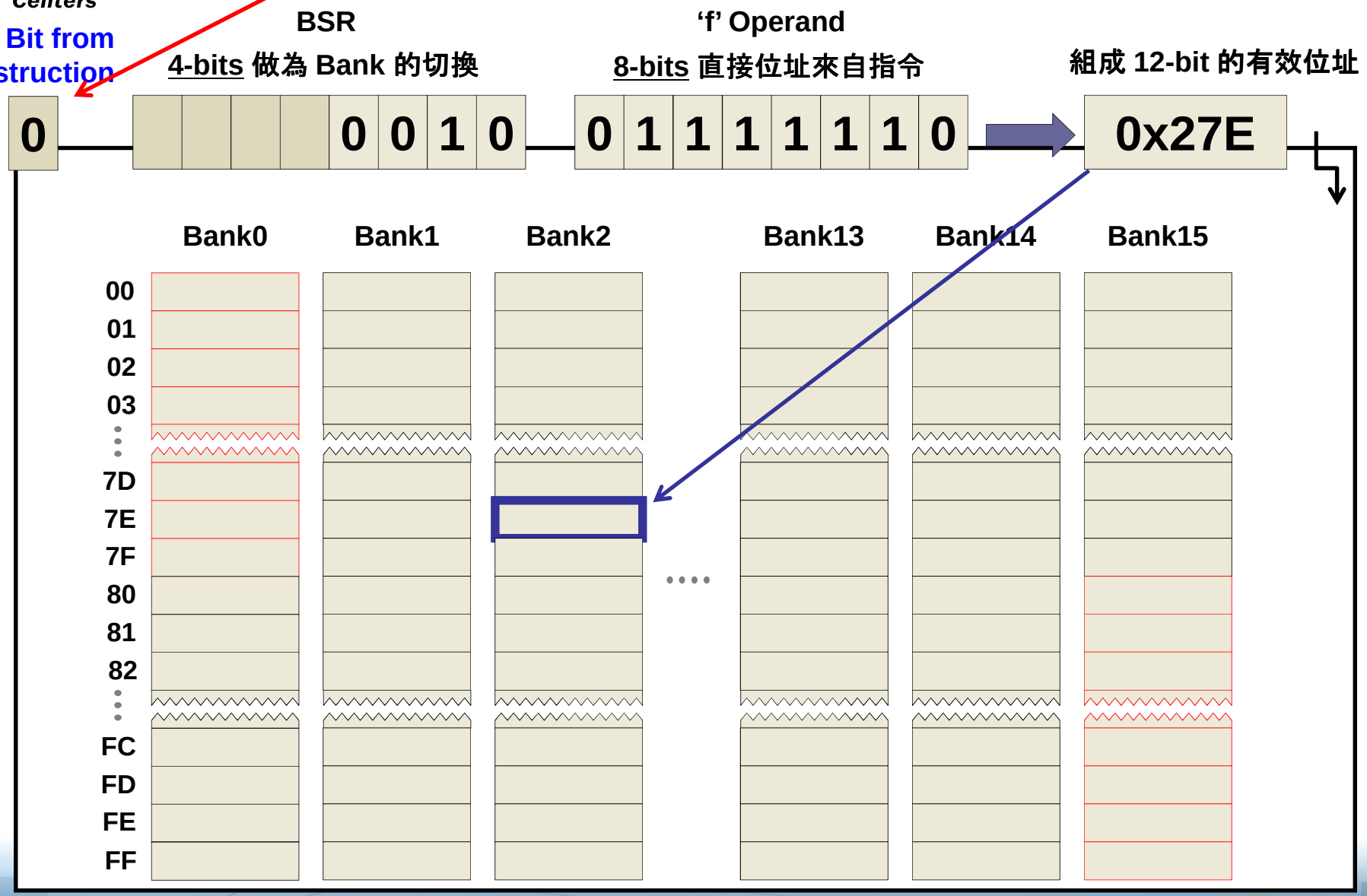
PIC18 系列標準指令集總覽

Control Operations		
bc	n	Branch if Carry
bn	n	Branch if Negative
bnc	n	Branch if Not Carry
bnn	n	Branch if Not Negative
bnov	n	Branch if Not Overflow
bnz	n	Branch if Not Zero
bov	n	Branch if Overflow
bra	n	Branch Always
bz	n	Branch if Zero
call	n,s	Call subroutine
clrwdt		Clear Watchdog Timer
daw		Decimal Adjust WREG
goto	n	Go to address
nop		No Operation
pop		Pop top of return stack (TOS)
push		Push top of return stack (TOS)
rcall	n	Relative Call
reset		Software device RESET
retfie	s	Return from interrupt
return	s	Return from subroutine
sleep		Go into standby mode

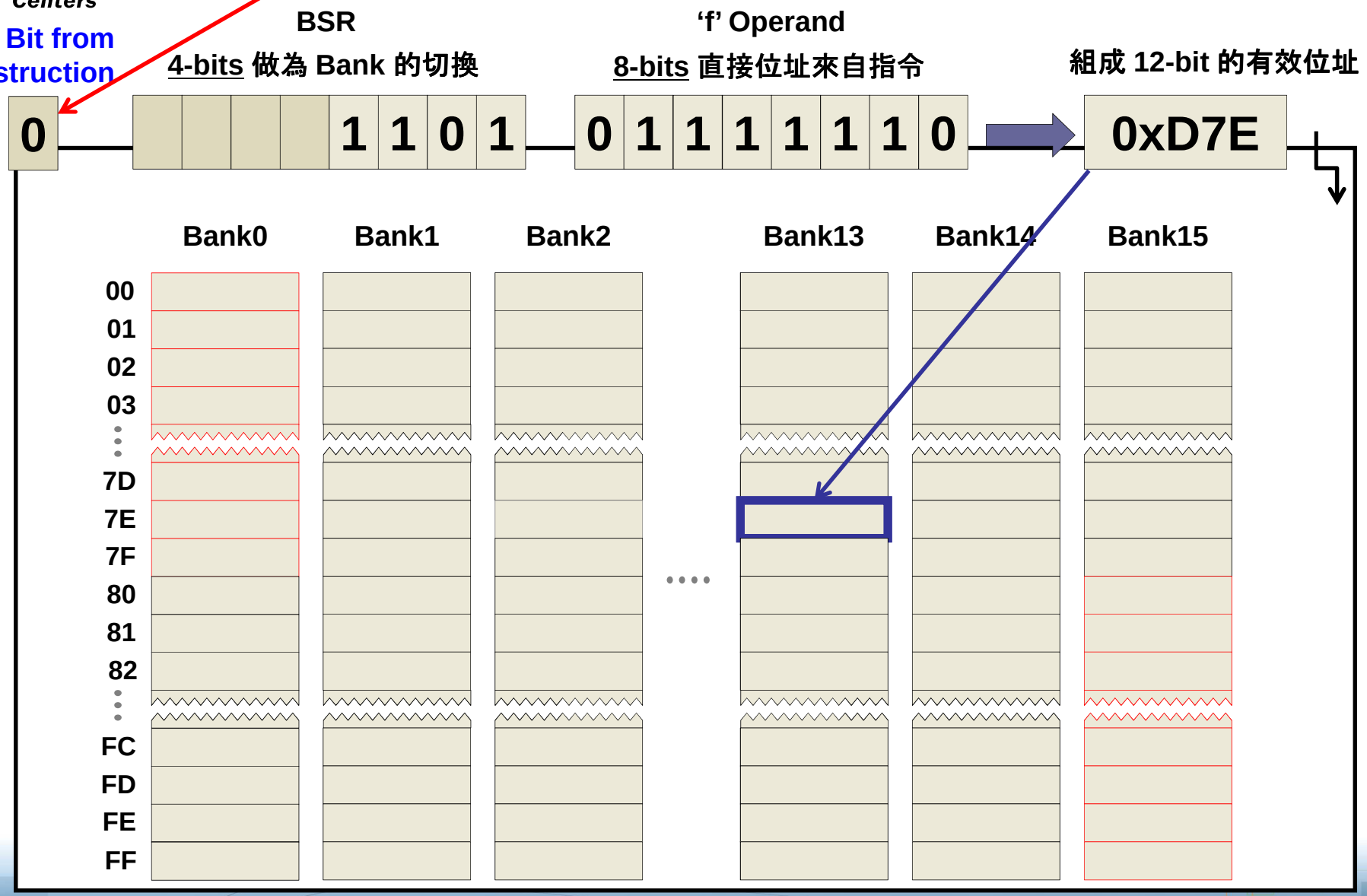
Literal Operations		
addlw	k	Add literal and WREG
andlw	k	AND literal with WREG
iorlw	k	Inclusive OR literal with WREG
lfsr	f,k	Move 12-bit literal to FSR
movlb	k	Move literal to BSR<3:0>
movlw	k	Move literal to WREG
mullw	k	Multiply literal with WREG
retlw	k	Return with literal in WREG
sublw	k	Subtract WREG from literal
xorlw	k	Exclusive OR literal with WREG

Data Memory ⇔ Program Memory Operations	
tblrd*	Table Read
tblrd*+	Table Read with post-increment
tblrd*-	Table Read with post-decrement
tblrd+*	Table Read with pre-increment
tblwt*	Table Write
tblwt*+	Table Write with post-increment
tblwt*-	Table Write with post-decrement
tblwt+*	Table Write with pre-increment

Banked 直接定址模式



Banked 直接定址模式



Access Bank 直接定址

‘a’ Bit from Instruction

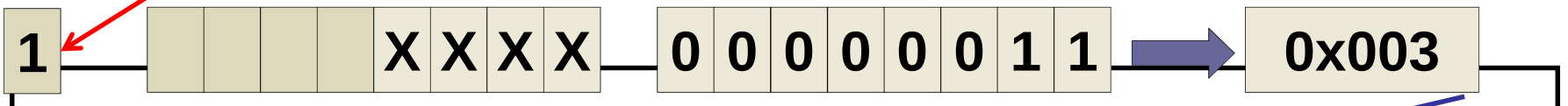
BSR

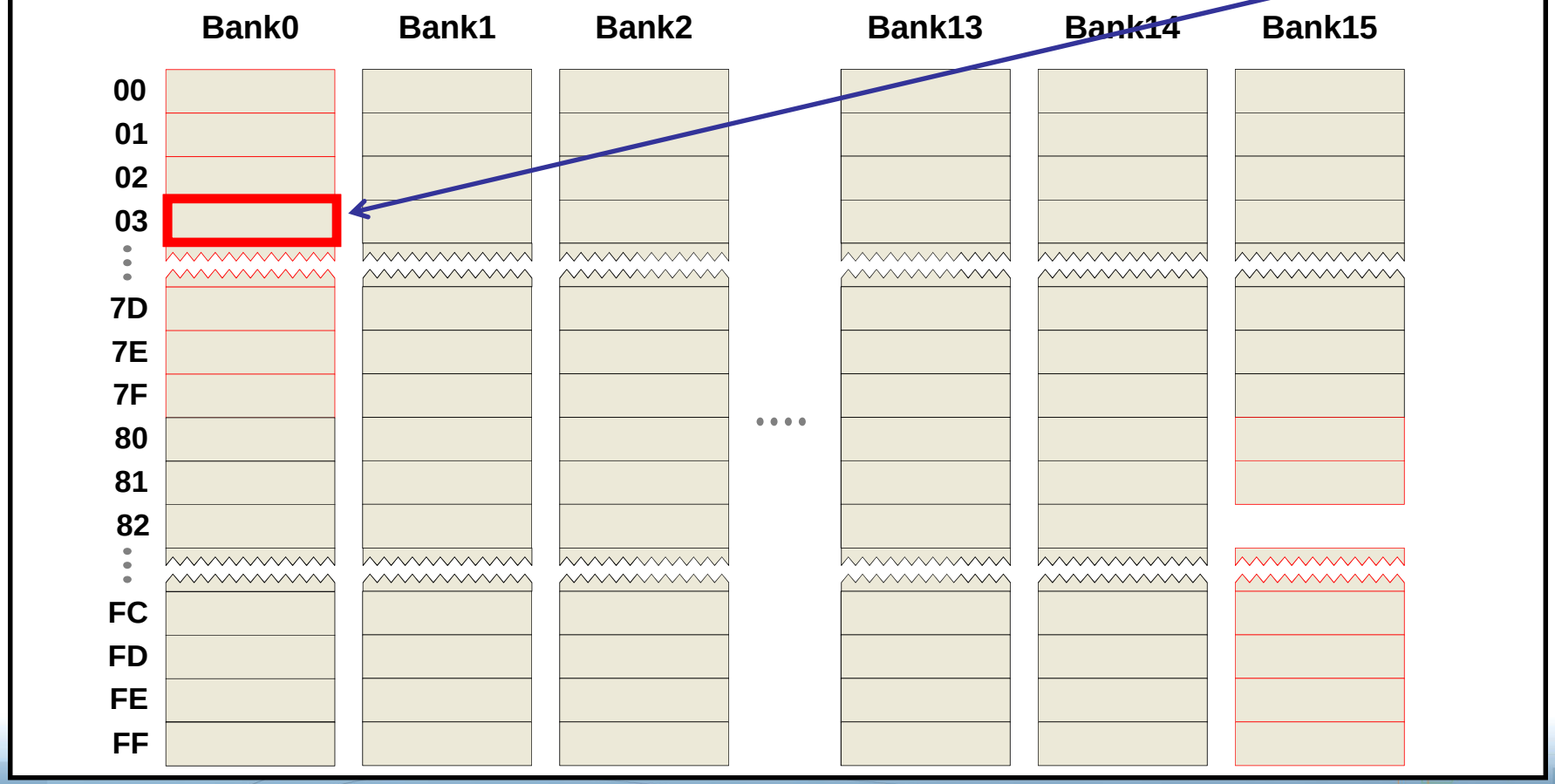
4-bits 做為 Bank 的切換

‘f’ Operand

8-bits 直接位址來自指令

組成 12-bit 的有效位址





Access Bank 直接定址

‘a’ Bit from Instruction

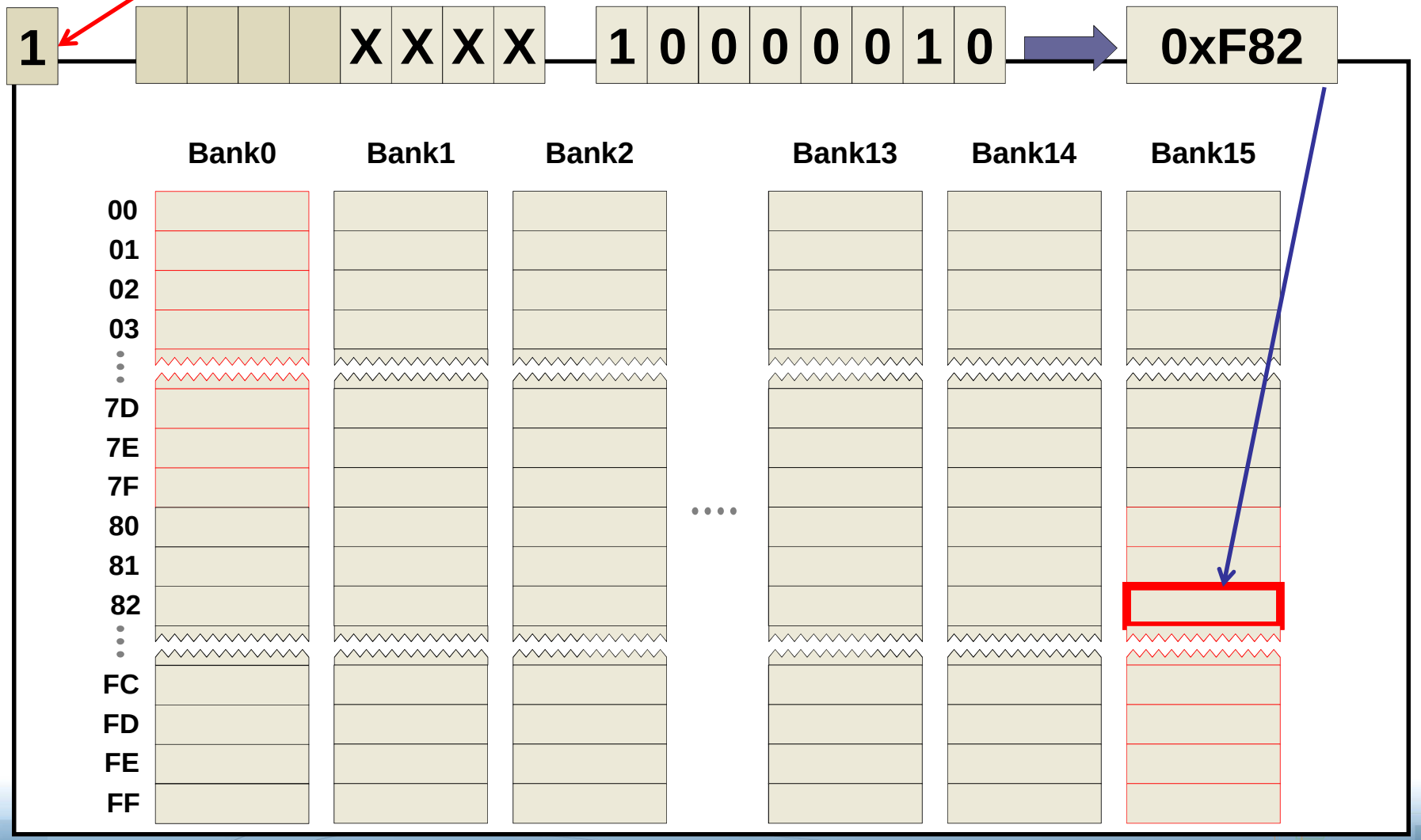
BSR

4-bits 做為 Bank 的切換

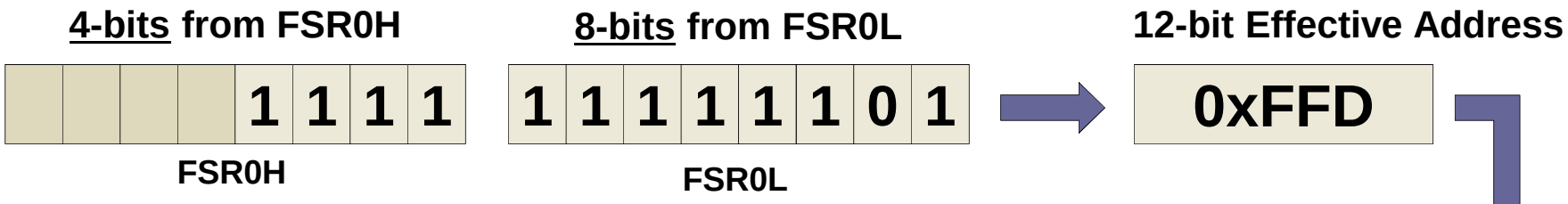
‘f’ Operand

8-bits 直接位址來自指令

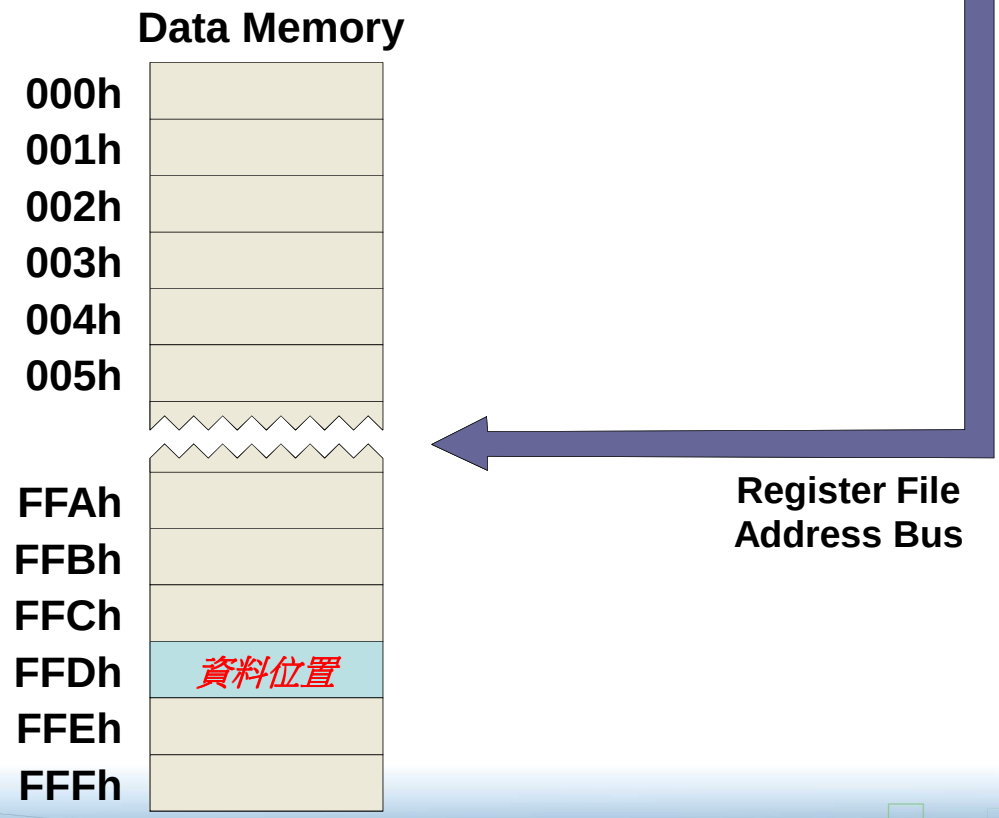
組成 12-bit 的有效位址



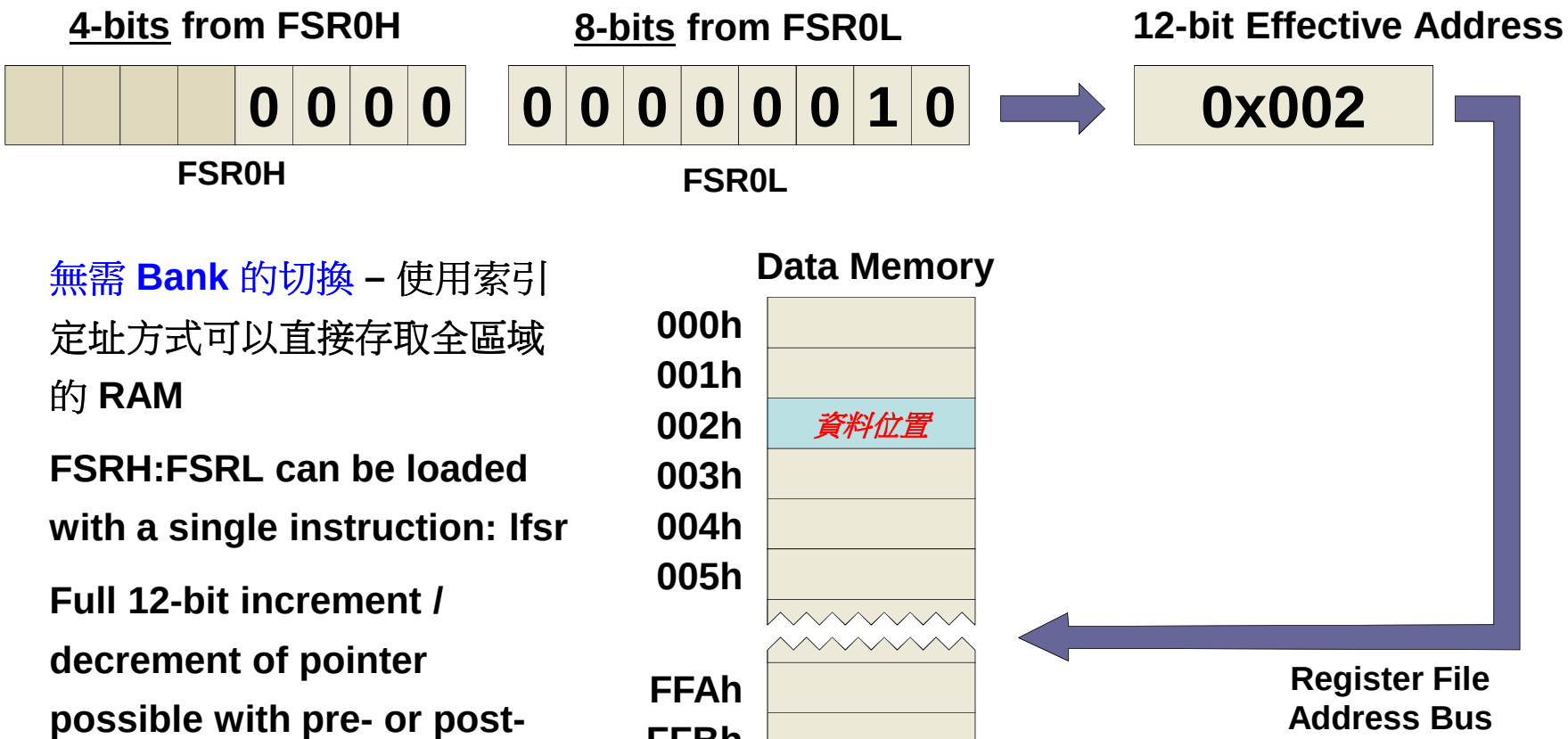
暫存器間接定址模式



- 無需 **Bank** 的切換 – 使用索引定址方式可以直接存取全區域的 **RAM**
- **FSRH:FSRL can be loaded with a single instruction: lfsr**
- **Full 12-bit increment / decrement of pointer possible with pre- or post-modification modes**
- 三對 **FSR** 暫存器可以使用 : **FSR0, 1 & 2**



暫存器間接定址模式



- 無需 **Bank** 的切換 – 使用索引定址方式可以直接存取全區域的 **RAM**
- **FSRH:FSRL can be loaded with a single instruction: lfsr**
- **Full 12-bit increment / decrement of pointer possible with pre- or post-modification modes**
- 三對 **FSR** 暫存器可以使用 : **FSR0, 1 & 2**

MICROCHIP
Regional Training Centers

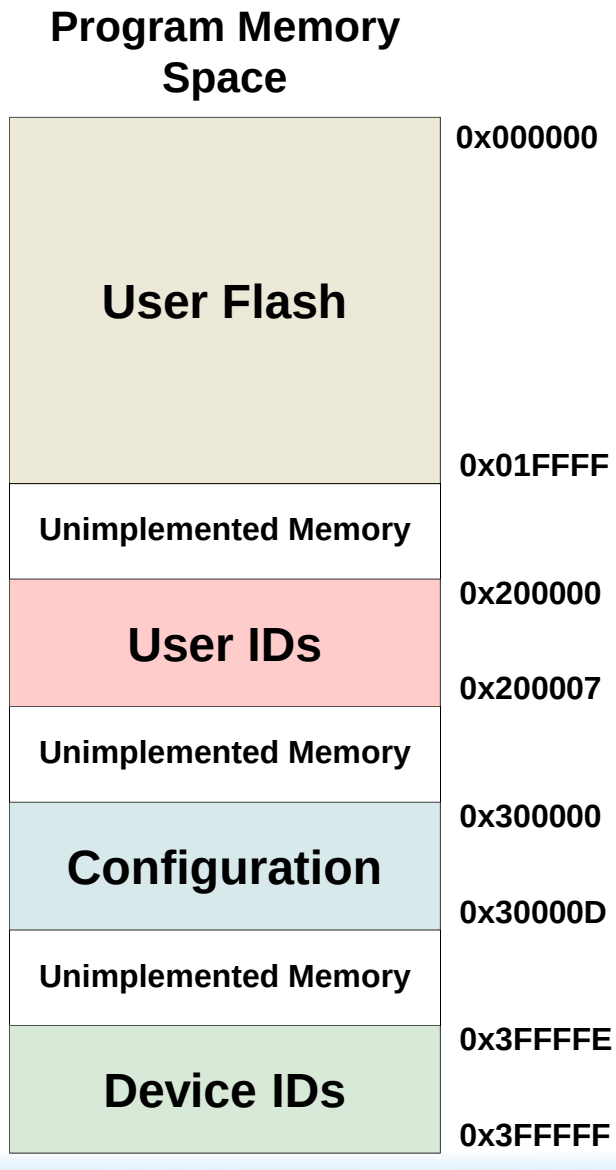
PIC18 Family 的定址模式

查表與跳躍指令

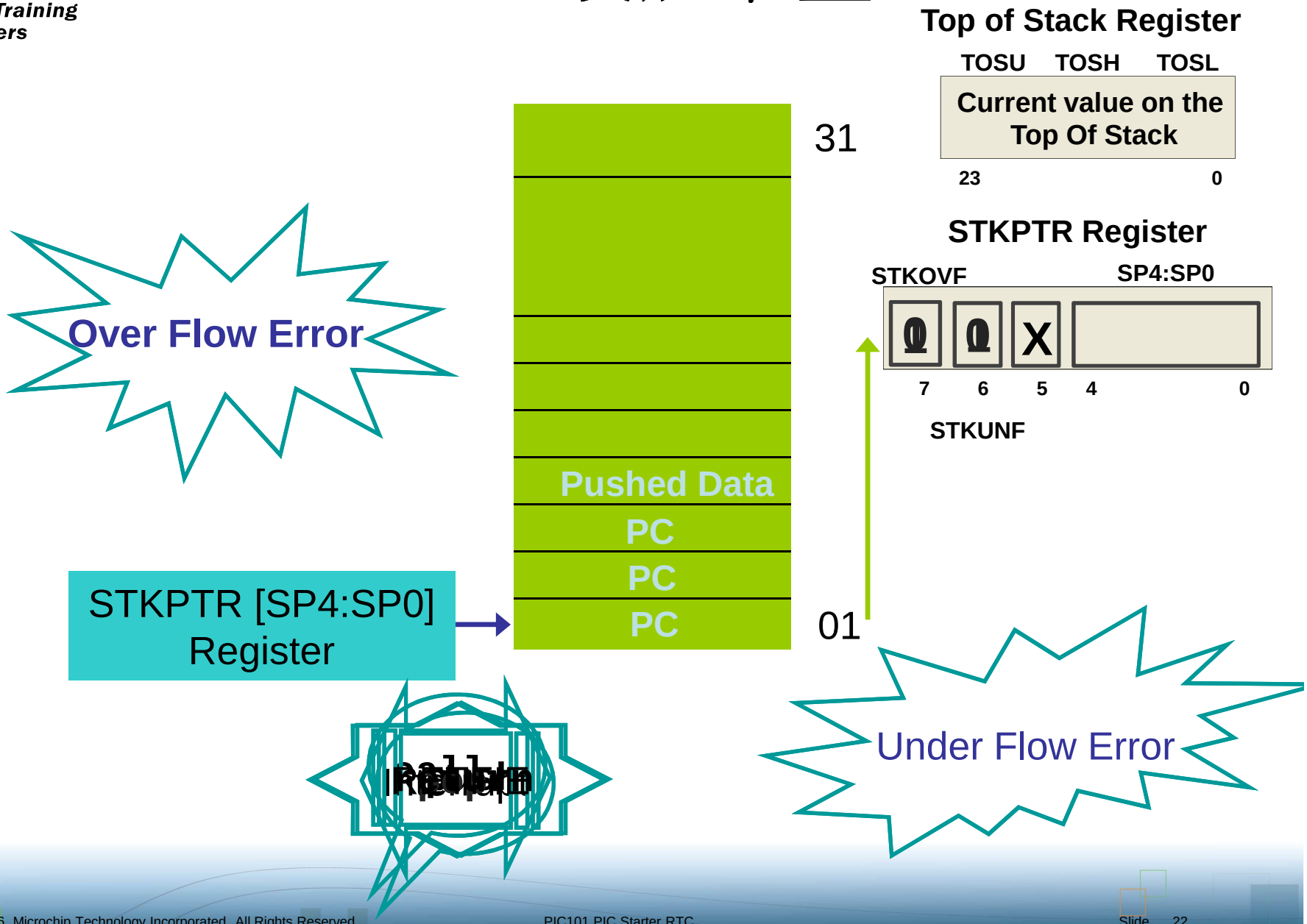
Mode	Example Syntax
Absolute 絕對定址	`goto <addr>`
Relative 相對定址	`bra <addr>`
Table Read / Write	`tblrd*` `tblwt*`
Table Read / Write Post Increment	`tblrd*+` `tblwt*+`
Table Read / Write Post Decrement	`tblrd*-` `tblwt*-`
Table Read / Write Pre Increment	`tblrd+*` `tblwt+*`

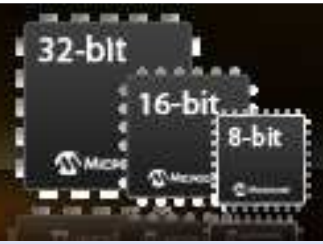
查表的能力

- **PIC18F** 程式記憶體可分成四部分:
  - User Memory
    - 最高到 **128kB** 內建程式記憶體
    - 最大可支援 **2MB (**外部擴充記憶體**)**
  - User IDs
    - **8** 客戶使用識別用區域
  - Configuration Memory
    - 元件的設定，程式保護設定**…**等
  - Device IDs
    - 元件或版本的 簽署區域
- 一般使用程式計數器 **(PC)** 最多只能存取 **21-bits** 的程式位址。
- 透過查表指令可以存取所有的程式記憶體 **2MB (TBLPTR is 22-bits wide)**



PIC18 硬體堆疊





先安裝 **MPLAB X IDE**

再安裝 **XC8** 編譯器

安裝 MPLAB X IDE

- 本教育訓練使用版本為: **X IDE v3.26**
- 請參照本教材的 **X IDE** 安裝附件，參考第 **1** 頁到第 **10** 頁的指引安裝 **MPLAB X IDE v3.xx** 的工作平台軟體。
  - X IDE 建議安裝在內定的路徑目錄。
  - X IDE 安裝時不能使用中文路徑。
  - X IDE 看不懂中文的登錄名稱。
  - X IDE 的專案檔不可放在桌面。(桌面只放圖示捷徑)

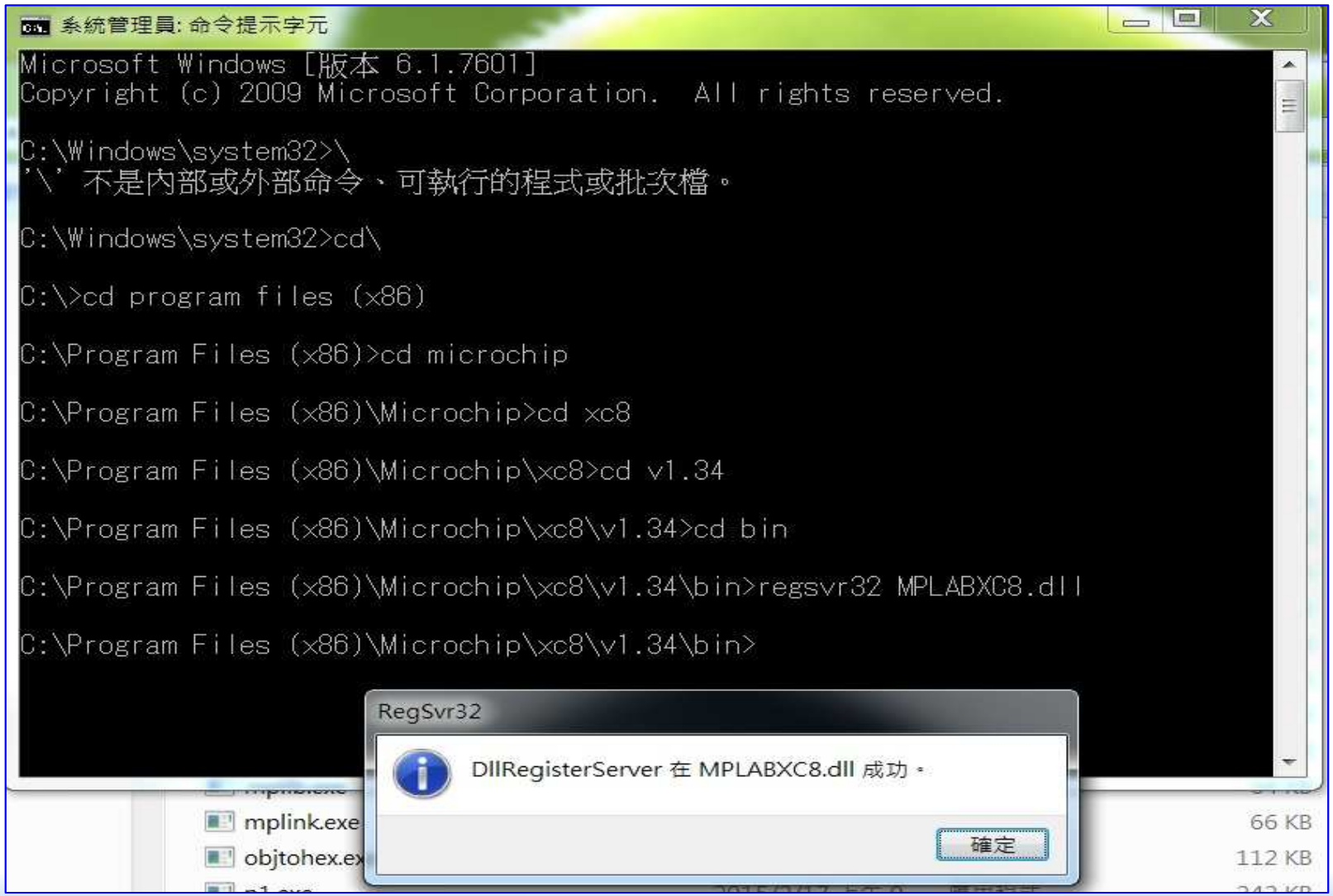
安裝 XC8 (Lite 版)

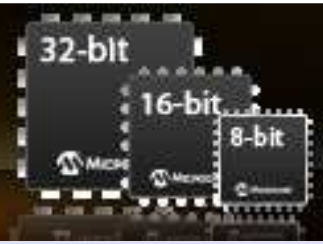
- 請參照本教材的 **X IDE** 安裝附件，參考第 **11**頁到第 **20** 頁的指引安裝 **XC8 v1.3x** 的 **C** 編譯器。
- **XC8** 建議安裝在內定的路徑目錄。
- 注意:
  - XC8 v1.35 (含) 以後版本不再內掛 PIC18F LIB 的週邊函式庫，如有需要請自行安裝 Peripheral Libraries (V2.00RC3)
  - 參考的 X IDE 安裝附件: page 20

MPLAB IDE v8.92 無法安裝 XC8 的問題

- 如果 **XC8** 無法安裝是因為螢幕字型大小設成 **150%,** 降低到**125%**後再安裝即可。
- 安裝 **XC8** 完成，在 **MPALB IDE v8.92** 看不到 **XC8** 的編譯器：
  - 到所有程式下 → 附屬應用程式 → 命令提示字元， 按老鼠右鍵以"系統管理員身分執行" DOS 的命令列。
  - 在 DOS 模式下該改執行的路徑到 “bin” 下，如下頁投影片的方式操作。
  - 執行 “regsvr32 MPLABXC8.dll” 後登錄 XC8 編譯器。
  - 開啟 MPLAB IDE v8.92 看看是否已加入 XC8 的編譯器

人工登錄 XC8 編譯器





XC8 基本介紹

關於 **XC8** 的更詳細說明請參考 “教育訓練光碟” 裡的：

XC8T v1.0 New!

MCU1121T HI-TECH PICC for PIC16F Series 2.0 New!

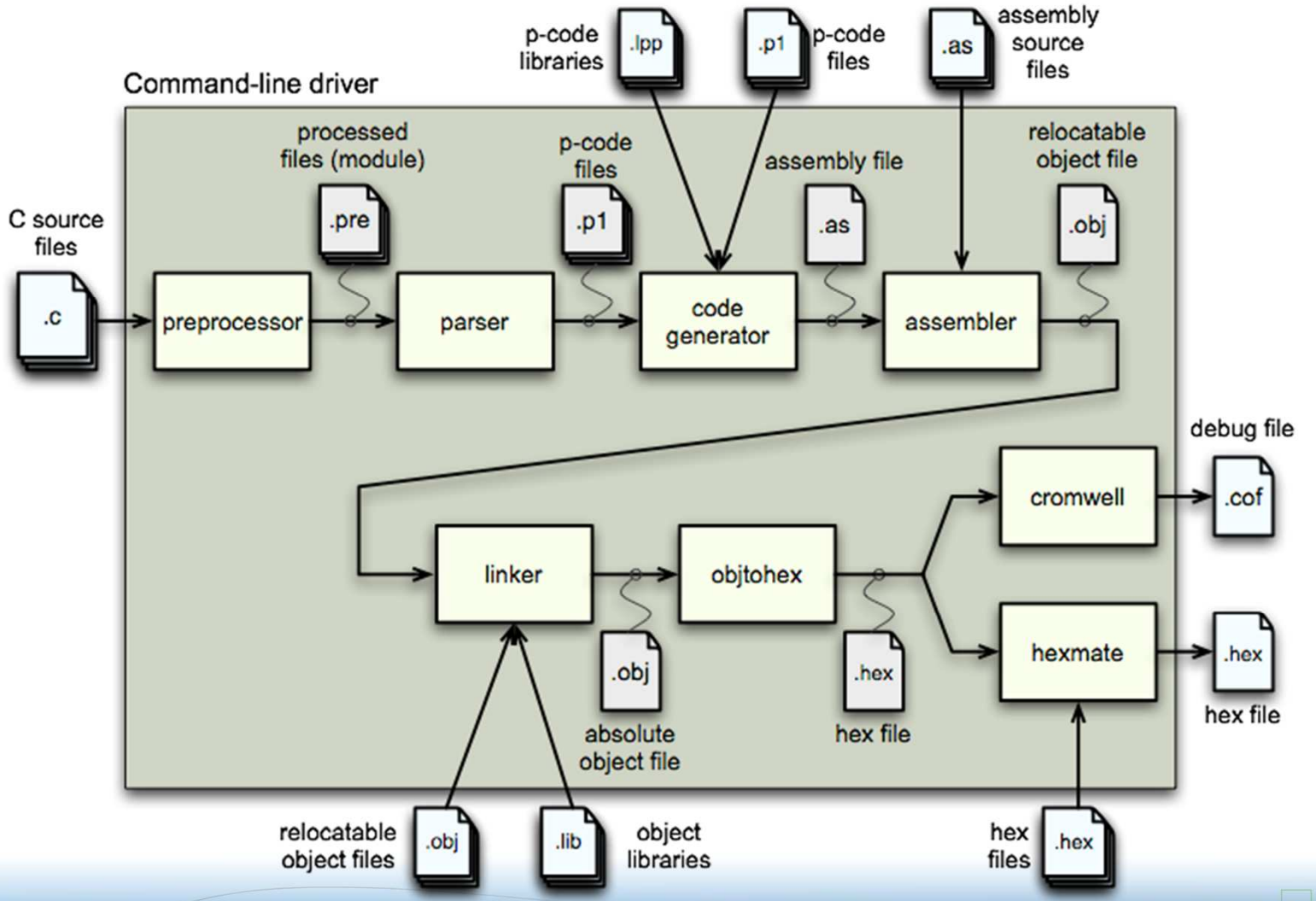
關於 XC8 C 編譯器

- **Microchip** 對 **8-bit PIC** 的 **C** 編譯器在 **XC8** 推出之前，主要是以 **Hi-Tech PICC (**支援 **PIC16F)** 及 **MPLAB C18 (**支援 **PIC18F)** 為主。**XC8** 是新推出的 **ANSI C** 編譯器。
- **XC** 編譯器有三種分為**: XC8 (8-bit PIC)**、**XC16 (16-bit PIC & dsPIC)**、**XC32 (32-bit PIC)** 為所有 **Microchip** 全系列的 **MCU** 提供一完整的 **C** 編譯器。
- **XC8** 經改良後繼承 **Hi-Tech C** 及 **C18** 雙方優良的語法，使得 **XC8** 更加容易使用也符合 **ANSI C** 的要求。 基本上 **XC8** 的精隨是延自於 **Hi-Tech C** 所以可以相容於 **Hi-Tech** 的語法。**XC8** 可支援 **PIC16F** 及 **PIC18F** 的元件，這也意味著 **XC8** 內建兩套編譯器以支援 **PIC16F & PIC18F**。
  - XC8 相容於舊的 Hi-Tech C 的與法 (可直接編譯)
- 有關 **XC8** 的詳細使用請參考**: XC8T v1.0** 的教育訓練

XC8 相關的執行檔

Name	Description
xc8 (calls PICC or PICC18)	Command line driver; the interface to the compiler
CLIST	Text file formatter
CPP	The C preprocessor
P1	C code parser
CGPIC or CGPIC18	Code generator (based on the target device)
ASPIC or ASPIC18	Assembler (based on the target device)
HLINK	Linker
OBJTOHEX	Conversion utility to create HEX files
CROMWELL	Debug file converter
HEXMATE	HEX file utility
LIBR	Librarian
DUMP	Object file viewer

標準版 XC8 編譯動作流程



XC8 編譯器版本

- **PRO mode (付費授權版)**
  - 全功能的 Omniscient Code Generation (OCG) 最佳化功能
- **Lite mode (免費版)**
  - 限制性的 OCG 最佳化功能
  - 不受記憶體大小、元件編號及使用時間的限制
  - 原始程式相容於 OCG 的語法
- 評估版 (免費版，限安裝一次)
  - 45 天內使用為 PRO 模式，超過時間則轉為 Lite 模式

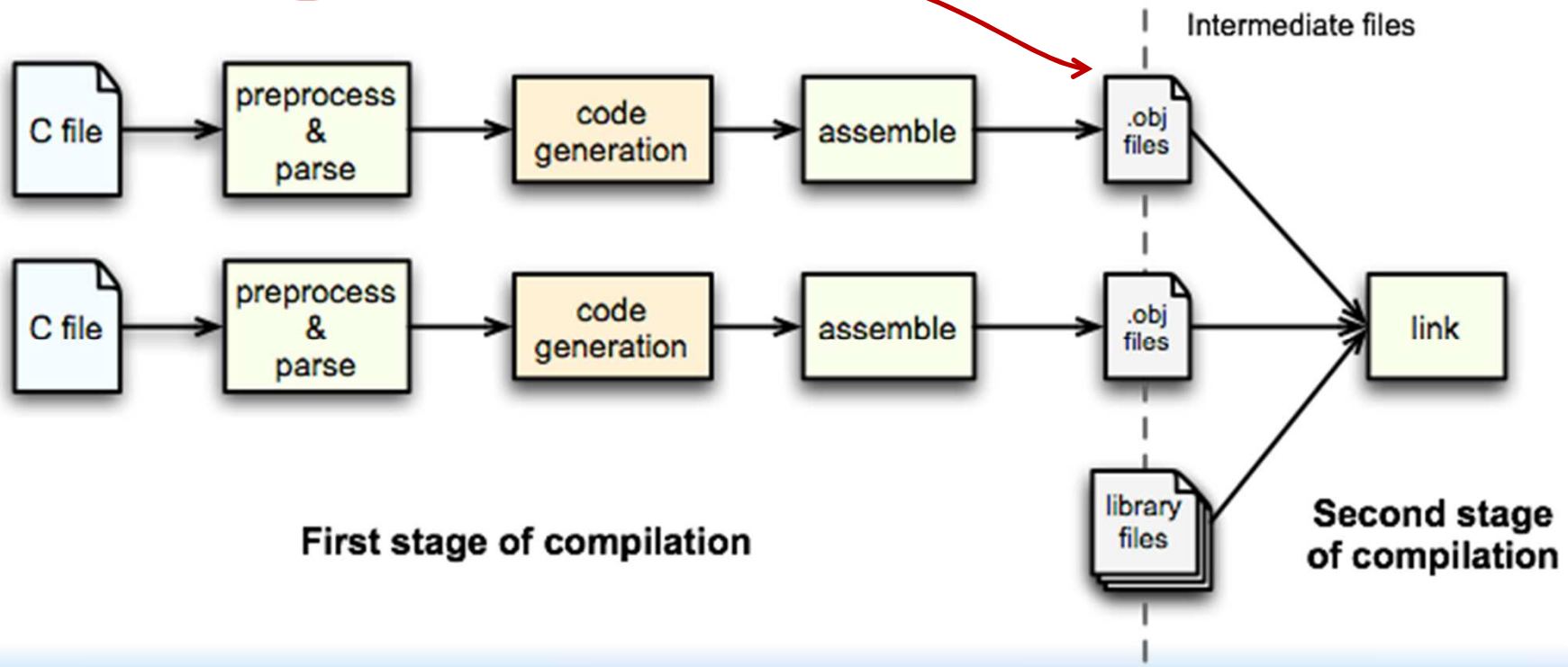
Microchip XC8
OCG Technology

- 所有 **Microchip** 的 **XC** 編譯器採用新的專利 **Omniscient Code Generation™ (OCG)** 全域性程式編譯技術。
- **OCG** 具有全視野範圍的編譯架構
- 在正式編譯前先掃描過所有 **C** 檔案的關聯
- 強化最佳化編譯功能，縮減程式碼的空間，加快執行的速度

編譯的動作流程

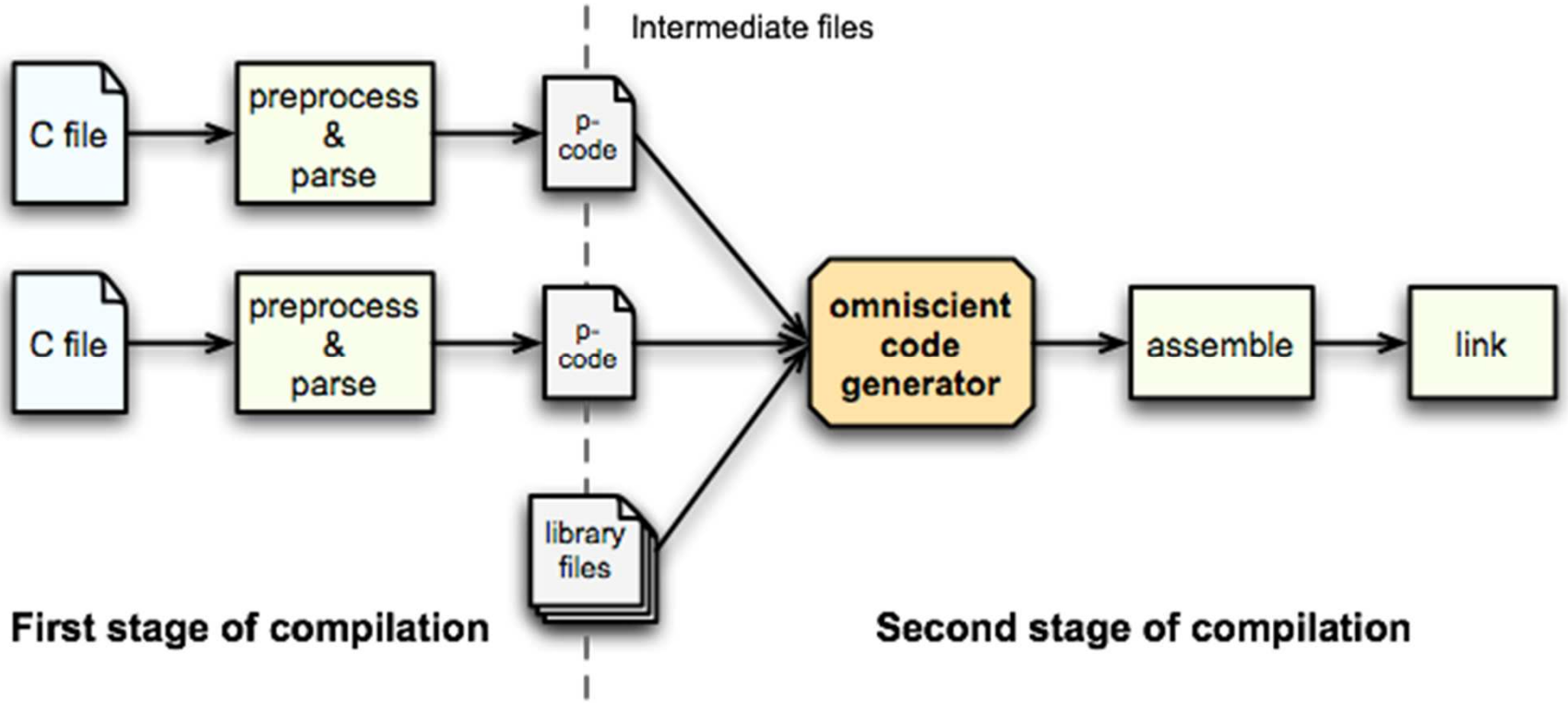
傳統方式編譯

- C 檔案是個別編譯處理的
  - 編譯出的中間檔案是要經連結器做連結處理的 obj 檔



編譯的動作流程
全域性編譯方式 (OCG)

- 所有的 **C** 程式的編譯是同時處理的
  - P-code 檔是中間檔案，一起送入 OCG 編譯



MICROCHIP
Regional Training Centers

XC8 整數的資料型別

Type	Bits	Min	Max
bit	1	0	1
char, unsigned char	8	0	255
signed char	8	-128	127
short, signed short	16	-32768	32767
unsigned short	16	0	65535
int, signed int	16	-32768	32767
unsigned int	16	0	65535
short long, signed short long	24	-8388608	8388607
unsigned short long	24	0	16777215
long, signed long	32	-2^{31}	$2^{31}-1$
unsigned long	32	0	$2^{32}-1$

XC8 只提供到 **32-bit** 的數值範圍
XC16 則提供到 **64-bit** 的數值範圍

浮點數的格式

- 支援兩種浮點數格式
  - 24-bit truncated IEEE format (Default)

sign	exp	mantissa
x	xxxx xxxx	xxx xxxx xxxx xxxx

  - 32-bit IEEE format

sign	exp	mantissa
x	xxxx xxxx	xxx xxxx xxxx xxxx xxxx xxxx

- 更大的儲存格式提供更精確的數值

標頭檔 (.h) 的路徑定義

- 編譯器提供的標頭檔 **(Header File)** 的路徑搜尋方式
  - ..\xc8\v1.34\include\　　一般的標頭檔 (安裝時內定路徑)
  - ..\xc8\v1.34\include\plib\　周邊函數庫的標頭檔
- **XC8 v1.35 (含)** 以後版本不再安裝 **PIC18F** 的周邊函數庫，如有需要需自行再安裝 **(v2.00RC3)**。
- **XC8** 不支援 **PIC16F** 的周邊函數庫

格式	內定的搜尋路徑
<xxx.h>	編譯器內定的標頭檔的目錄下
"xxx.h"	先搜尋專案下的原始檔的目錄； 再到編譯器設定的標頭檔的目錄

通用標頭檔
Generic Header File (xc.h)

- 使用 **XC8** 只需加入的標頭檔 **xc.h** 即可
- **xc.h** 標頭檔需在每個程式模組加入此檔案
  - #include <xc.h>

- 舊版 **Hi-Tech C** 編譯器使用 **<htc.h>** 與使用 **<xc.h>** 是一樣的
  - #include <xc.h> 等於 #include <htc.h>

pic18_chip_select.h

- **xc.h** 會自動帶入此標頭檔
- **pic18_chip_select.h** 是很重要的檔案
  - 依據 X IDE 所傳入的元件訊息找出所要使用的元件標頭檔
    - **PIC18F4520** 找出 **<pic18f4520.h>**
    - **PIC18F45K22** 找出 **<pic18f45k22.h>**
- **pic18f45k22.h** 融合 **Hi-Tech C** 及 **C18** 對內部暫存器名稱定義
  - 可直接使用”位元名稱” 或 “位元結構” 的方式做 bit 的存取

pic18f4520.h 裡的定義 (SFRs)

```
// Register: PORTA
extern volatile unsigned char       PORTA           @ 0xF80;
#ifndef _LIB_BUILD
asm("PORTA equ 0F80h");
#endif
// bitfield definitions
typedef union {
    struct {
        unsigned RA0                :1;
        unsigned RA1                :1;
        unsigned RA2                :1;
        unsigned RA3                :1;
        unsigned RA4                :1;
        unsigned RA5                :1;
        unsigned RA6                :1;
        unsigned RA7                :1;
    };
                :
                :
} PORTAbits_t;
extern volatile PORTAbits_t PORTAbits @ 0xF80;extern  volatile  PORTAbits_t  PORTAbits  @
0x005;
```

定義 **PORTA** 的位元結購
(用法上與 **C18** 相容**)**

定義 **PORTA** 的絕對位址

Hlink 所使用的元件連結描述檔

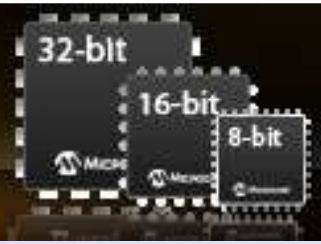
- 連結程式 **(hlink.exe)** 如何知道所選用的元件的訊息
  - PIC16Fxxx : picc.ini
  - PIC18Fxxxx : pic18-18.ini
- 修改 **ini** 的內容即改變該元件所預設的資源，如需修改需注意其內容。

PIC18F4520 記憶容量及位址設定

..\microchip\xc8\v1.xx\dat\picc-18.ini

- 底下以 **PIC18F4520** 為例，請參考 **picc-18.ini** 第一頁的說明

```
[18F4520]
ARCH=PIC18
CFGMEM=300000-30000D
COMMON=00-7F
DEVIDMEM=3FFFFE-3FFFFF
EEPROM=F00000-F000FF
ERRATA=FASTINTS
FAMILY=18f4520
FLASH_EW=40,20
GPRBANKS=080-0FF,100-1FF,200-2FF,300-3FF,400-4FF,500-5FF
ICD2RAM=5F4-5FF
ICD2ROM=7DC0-7FFF
INSTR=EXTENDED
MAKE=MICROCHIP
PLIB=1
PROCID=4520
RAMSIZE=600
REALICERAM=5EF-5FF,F9C-F9C,FD4-FD4,FDB-FDF,FE3-FE7,FEB-
FEF,FFC-FFF
REALICEROM=7D00-7FFF
ROMSIZE=8000
SFRBANKS=F80-FFF
STACKDEPTH=1F
USERIDMEM=200000-200007
```



存取特殊功能暫存器

SFR and Bit Access

MICROCHIP
Regional Training Centers

使用內建的暫存器名稱宣告

- 暫存器名稱與其單獨的位元宣告名稱都在相對的元件標頭檔裡宣告。以本課程所使用的 **PIC18F45K22** 為例，該元件所有的暫存器及位元的名稱都在 **pic18f45K22.h** 的標頭檔裡宣告。
- 程式裡只要使用 **#include <xc.h>** 即可， 所使用的元件標頭檔會自動含入
- 所定義的暫存器及其位元名稱與 **Data Sheet** 相同

使用特殊功能暫存器

- 特殊功能暫存器 **(SFR)** 的存取就像一般的全域變數一樣的簡易
  - 名稱的宣告與資料手冊相同 (pic18f45K22.h)
  - 同時也定義了位元名稱
  - <xc.h> 檔會自動找出所使用元件的定義檔

使用 **SFR** 的範例：

```
#include <xc.h>

ADCON0 = (channel << 3) + 0xC1;
GODONE = 1;          // ADC 開始轉換
while(GODONE);       // 轉換完成?
```

XC8 位元變數定址方式

1. 使用位元的絕對位址方式
   - 源自於 Hi-Tech C 的專用語法
   - 用來定義特殊功能暫存器(SFR)內的獨立位元
   - SFR內的位元定義 – 需要使用 pic18f45K22.h
2. 使用 **bit** 定義位元變數
   - 源自於 Hi-Tech C 的專用語法
   - 最簡單方便的使用方式
3. 位元結構方式
   - ANSI C 標準使用方式
   - 與 MPLAB C18 語法相容

使用絕對位元
絕對位址的設定 (PIC16F)

1. 使用絕對位址定址方式
   - 位元定址的起始位址為 0
   - 公式 : (8 bits * SFR 的位址) + 偏移位元

範例: 存取 PORTA 的 RA5

```
static volatile unsigned char PORTA @ 0x05 ;
static volatile bit RA5 @ (unsigned) & PORTA*8 + 5 ;
```

```
RA5 = 1;          // RA5 輸出 High
Nop( );           //  避免 Read-Modify-Write 現象
RA3= 0;           // RA3 輸出 Low
If (RB0) ….       // 判斷RB0
```

使用絕對位元

絕對位址的設定 (PIC18F)

1. **pic18F45k22.h** 檔的絕對位址定義

```
extern volatile __bit    LATA0        @ (((unsigned) &LATA)*8) + 0;
#define                  LATA0_bit     BANKMASK(LATA), 0
extern volatile __bit    LATA1        @ (((unsigned) &LATA)*8) + 1;
#define                  LATA1_bit     BANKMASK(LATA), 1
```

範例: 存取 **PORTA** 的 **I/O** 腳訊號

```
LATA0 = 1;          // RA0 輸出 High
LATA3 = 0;          // RA3 輸出 Low
If (RB0) ….         // 判斷RB0
```

使用 **bit** 型別

2. **bit** 型別使用布林數 **(boolean values)**

- 使用 bit 的定義

```
bit motor_is_on;  // motor state flag
```

- 會將零散 bit 位元變數整合在一個 Byte 的 RAM
- Bit 變數無法為 auto 變數型態
  - 可以宣告成 **local** 變數，但必須加上 **static** 的宣告
- Bit 變數無法透過指標存取
- 使用起來非常簡單

bit 型別範例

2. 透過 bit 的簡易的整數位元轉移

範例:

```
char value = 0x34; // 0b00111000
bit flag;

flag = value;      // flag will be assigned 0
```

簡易使用的 bit 型別的範例:

```
static bit Count_Flag ;
static bit Buzzer_1_Flag ;

Buzzer_1_Flag = 1 ;
if (Count_Flag) Count_Flag = 0 ;
```

使用位元結構方式
(與 C18 相容)

3. 將 **PORTD** 的 **bit3 & bit5** 設定為輸出腳並設定輸出為 **High**：

```
TRISDbits.TRISD3 = 0;          // 設定 RD3 為輸出腳功能
TRISDbits.TRISD5 = 0;          // 設定 RD5 為輸出腳功能
PORTDbits.RD3 = 1 ;            // 將 RD3 輸出 High
PORTDbits.RD5 = 1 ;            // 將 RD5 輸出 High
```

另一範例：

```
#define   SW1      PORTAbits.RA0
#define   LED1     PORTDbits.RD0
TRISDbits.TRISD0 = 0;          // 設定 RD0 為輸出腳功能
TRISAbits.TRISA0 = 1;          // 設定 RA0 為輸入腳功能
:
If (!SW1) LED1=1;              // SW1 被按下，點亮 LED1
else LED1=0;                   // SW1 放開，LED1 熄滅
```

XC8 指定變數擺放位址

- **XC8** 使用 **@ address** 的方式設定變數的絕對位址

- 範例：

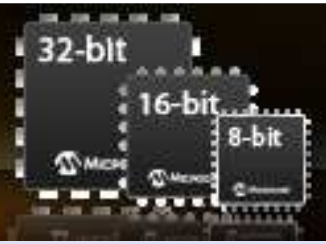
```
int foobar @ 0x100;
unsigned char ABC[10] @0x100;
```

函數的絕對位址

- 函數呼叫:
  - PIC16F/18F 函數的呼叫將使用內建的硬體堆疊
  - 中斷也會使用硬體堆疊
  - 參數的傳遞不是使用硬體堆疊，而是使用 RAM
- 使用絕對定址方式來設定函數的位址
  - "*函數原形宣告*" 上使用絕對定址法 ***@ address***
  - 函數進入點將會被設定在 address

設定函數在特定的絕對位址

```
void special(int a) @ 0x1000;
// 設定該函數在 Flash 位址 0x1000 的開始編譯
```



MPLAB X IDE 概述

關於 **X IDE** 的更詳細說明請參考 **“教育訓練光碟”** 裡的：

XIDET MPLAB X IDE 中文版 **New!**

的中文版的教育訓練

X IDE 的啟用

- X IDE 是個功能強大的 PIC 微控制器整合式的開發平台
  - 支援 8-bit，16-bit， 32-bit 全系列
- 使用 X IDE 需建立開發的專案
- 基本的使用及專案建立請參考 X IDE 安裝附件裡第 23 頁到 31 頁的操作說明。

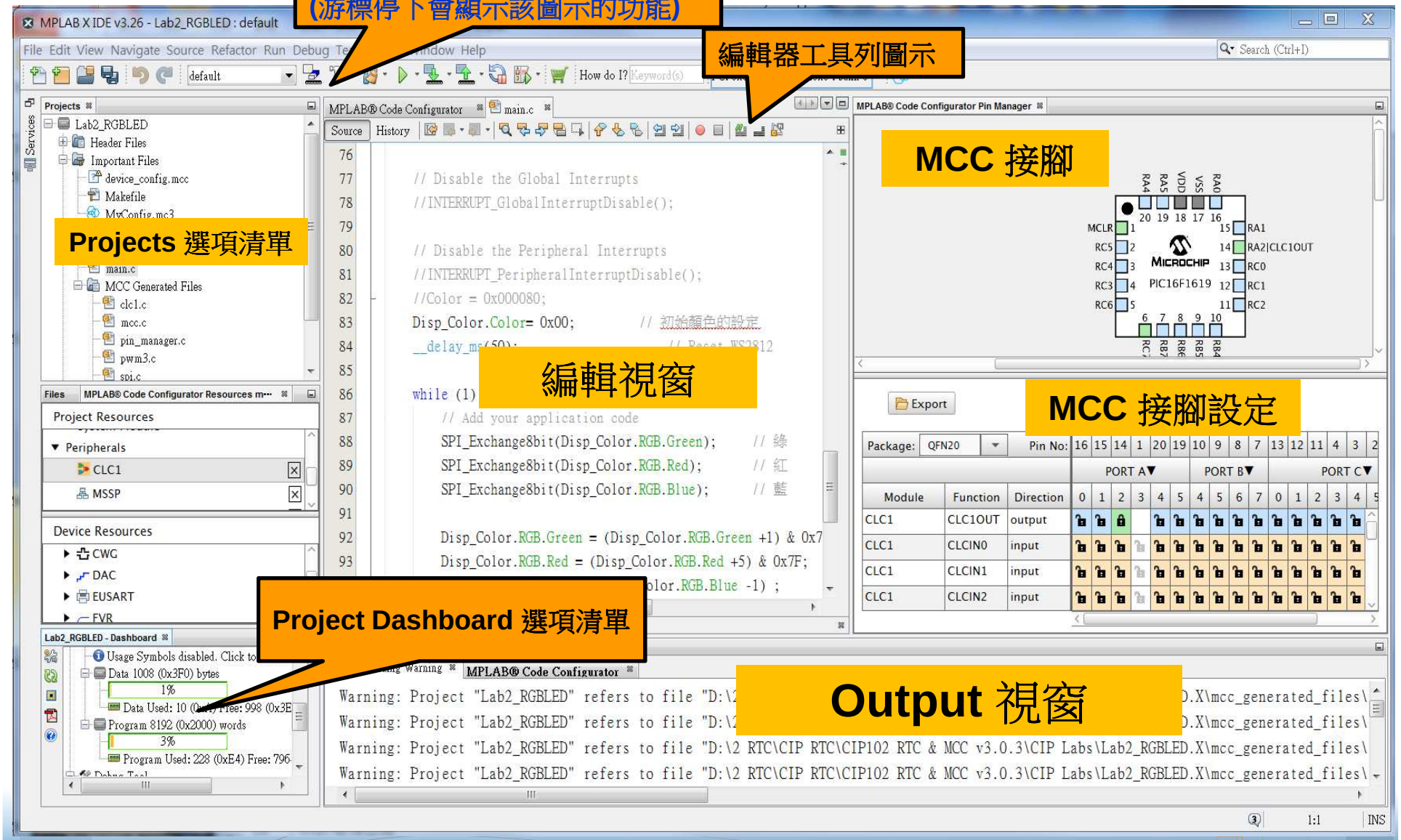
先了解切換步驟
開發工具 ICD 3 的切換

- **PICKit 3 使用 HID 驅動程式**
  - MPLAB IDE v8.x 和 MPLAB X IDE 一樣不用做切換
- 切換 **ICD 3 (確定 ICD3 已經連接 USB)**
  - 確定 MPLAB IDE 及 X IDE 都已經關閉
  - 以系統管理員身分執行 MPLAB Device Driver Switcher
  - 選擇 ICD 3 給 MPLAB 後，按下 Apply Changes 選項
  - 於對話視窗確定一下 ICD 3 切換成供訊息
- 此切換工具很笨，它不會提示目前工具的設定是給 **MPLAB IDE** 或 **MPLAB X IDE** 來使用

MPLAB X IDE 的版面配置

主工具列圖示
(游標停下會顯示該圖示的功能)

編輯器工具列圖示



如何設定 XC 編譯器

1 打開 Embedded 選項視窗

a 從 主功能表 ▶ **Tools**（工具）▶ **Options**（選項）

b 在打開的視窗中，選擇 **Embedded** 分類

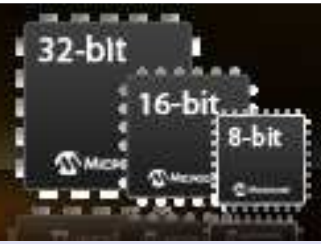
c 選擇 **Build Tools** 編譯工具）選項卡

為 XC8 v1.36 加入周邊函數庫

- 如前所述，**XC8** 自 **v1.34** 均不再內建 **PIC18F** 系列的周邊函數庫
- **XC8 v1.36** 需再安裝?
  - PIC18F Legacy Peripheral Libraries v2.0 - Windows
- 在台灣網站下的教育訓練光碟的連結:

安裝此軟體

開發軟體, 編譯器	
MPLAB® X IDE	v3.26 Windows(Local) Windows Version Linux Version Mac Version *Detail Info.*
MPLAB® IDE	v8.92(Local)
MPLAB® XC8	v1.36(Local) *Peripheral libraries not include
• Part Support Patch Files	v1.36(Local)
• Peripheral Libraries	v2.00RC3(Local) v1.34(Local)



PIC101 Lab1.x

如何新建獨立專案

PIC101 Lab1.x

如何建立建立獨立專案

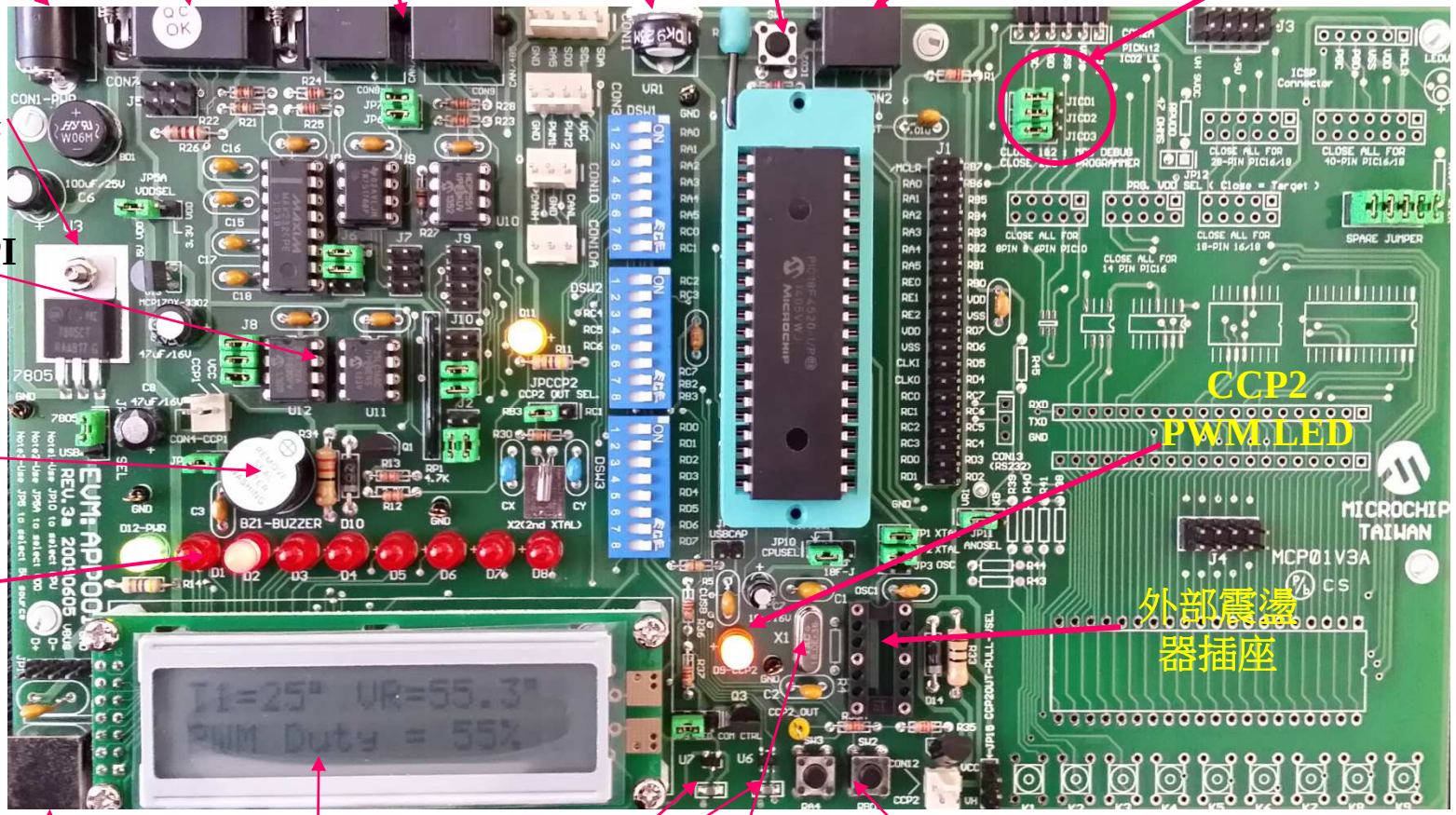
目的

- 使用專案精靈從頭開始建立專案
- 描述在專案精靈中選擇的各個選項
- 提供可編譯自己專案的平臺

APP001 實驗板使用手冊

- **APP001** 實驗板的使用手冊、電路圖及出廠測試程式可以在 **Microchip** 台灣的網站下載:
  - “教育訓練光碟”的連結
    - **APP001 v3.0** 實驗板電路圖
    - **APP001 v3.0** 中文使用手冊
    - **APP001 V3.0** 出廠測試程式

認識 APP001 v.3a 實驗板



PIC101 Lab1.x

如何建立獨立專案

目標

- 請先確定一下 **APP001** 上的元件是
  - PIC18F45K22
- 練習一所提供的原始檔案從頭開始建立新的 **MPLAB® X** 專案
  - 程式 : **Lab1_Basic_LED_Delay.c**
- 程式功能為每 **0.25** 秒 **LED1** 閃爍一次
  - 使用軟體延遲方式

PIC101 Lab1.x

如何建立獨立專案

步驟

1. 選擇專案類型
2. 指定開發的元件編號
3. 指定除錯器 / 程式燒錄器
4. 指定編譯器
5. 指定專案名稱和位置
6. 將檔加入到專案中
7. 執行專案

PIC101 Lab1.x

如何建立獨立的專案

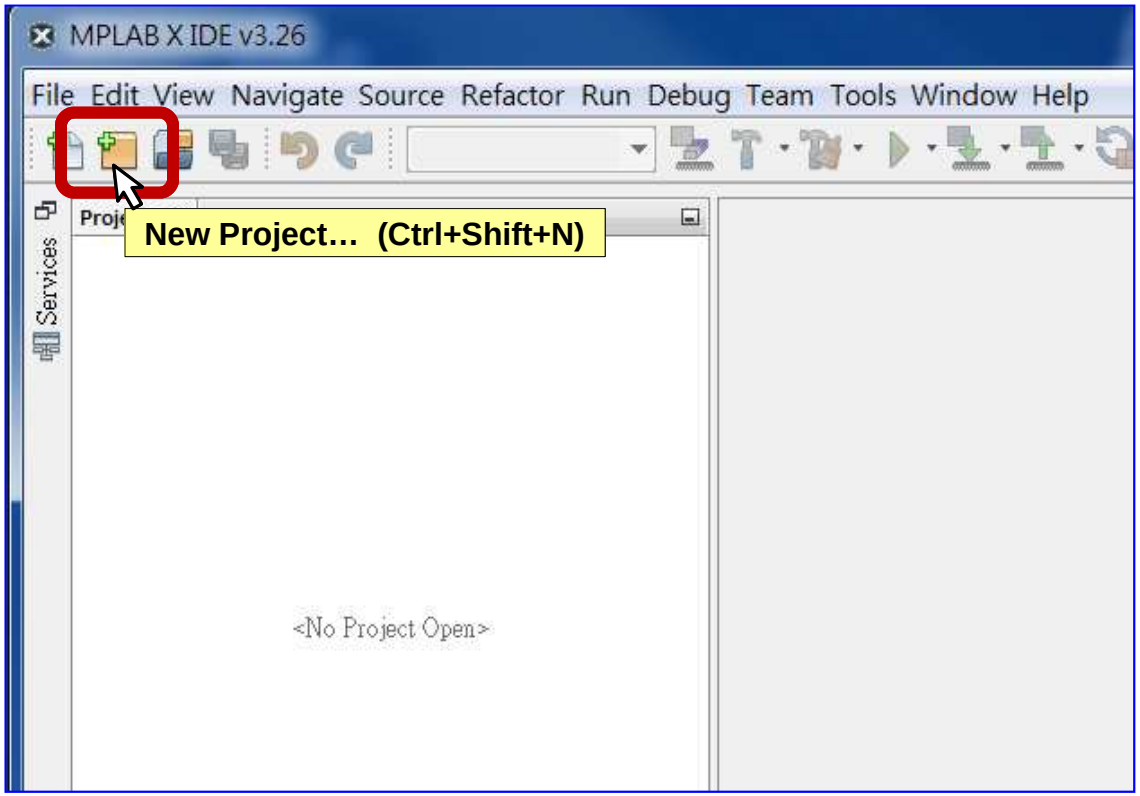
1 啟動新建專案精靈

工具列

選單

File（文件）▸
New Project...
（新建專案...）

鍵盤



PIC101 Lab1.x

如何建立獨立專案

2 選擇專案

a 在 “**Categories**”（類別）下選擇：

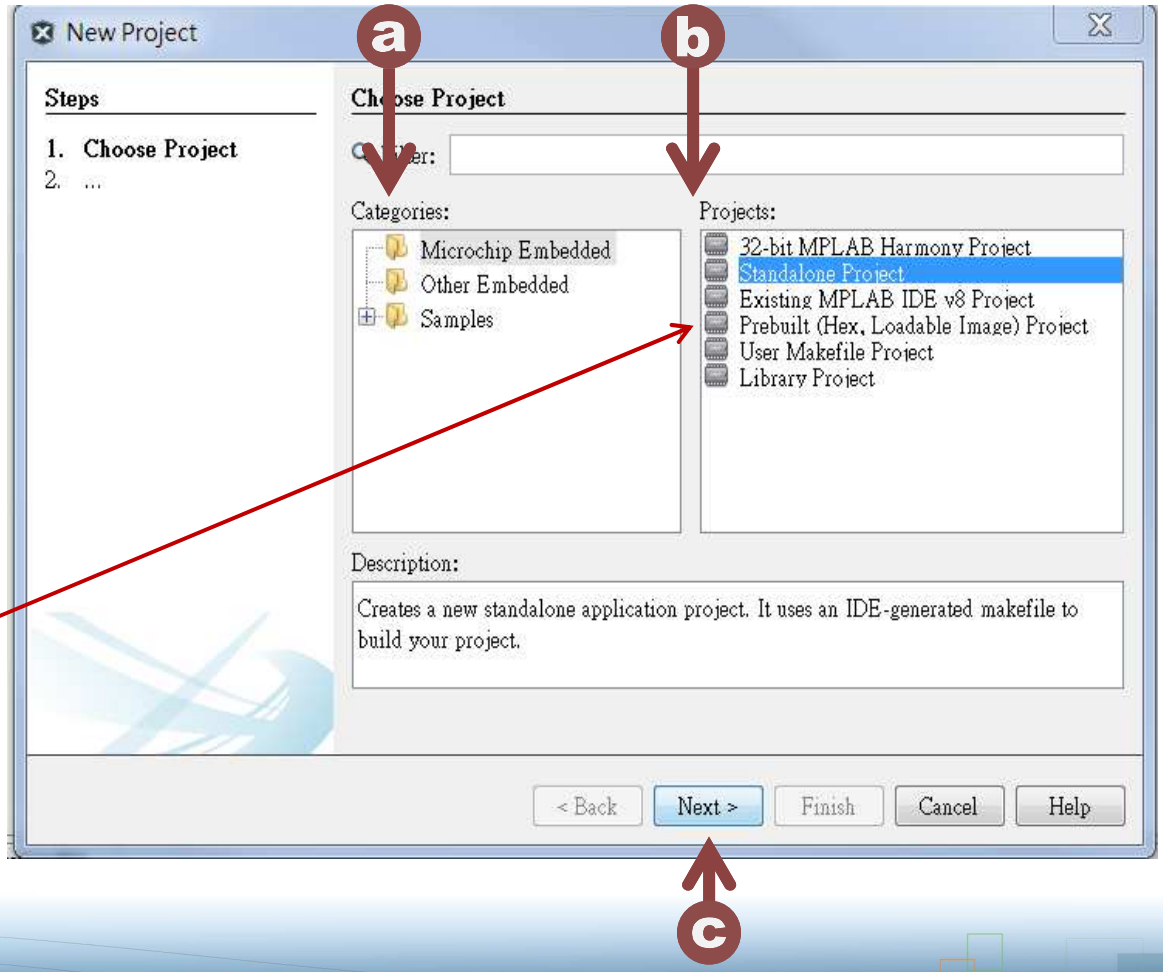
Microchip Embedded (Microchip 嵌入式）

b 在 “**Projects**”（專案）下選擇：

Standalone Project （獨立專案）

*** 在第四項 Prebuilt (Hex) Project 可以建立 Hex 檔的燒錄專案 (類似 IDE 的 Import / Export 的功能。**

c 按一下 **Next >**



PIC101 Lab1.x

如何建立獨立專案

3 選擇元件

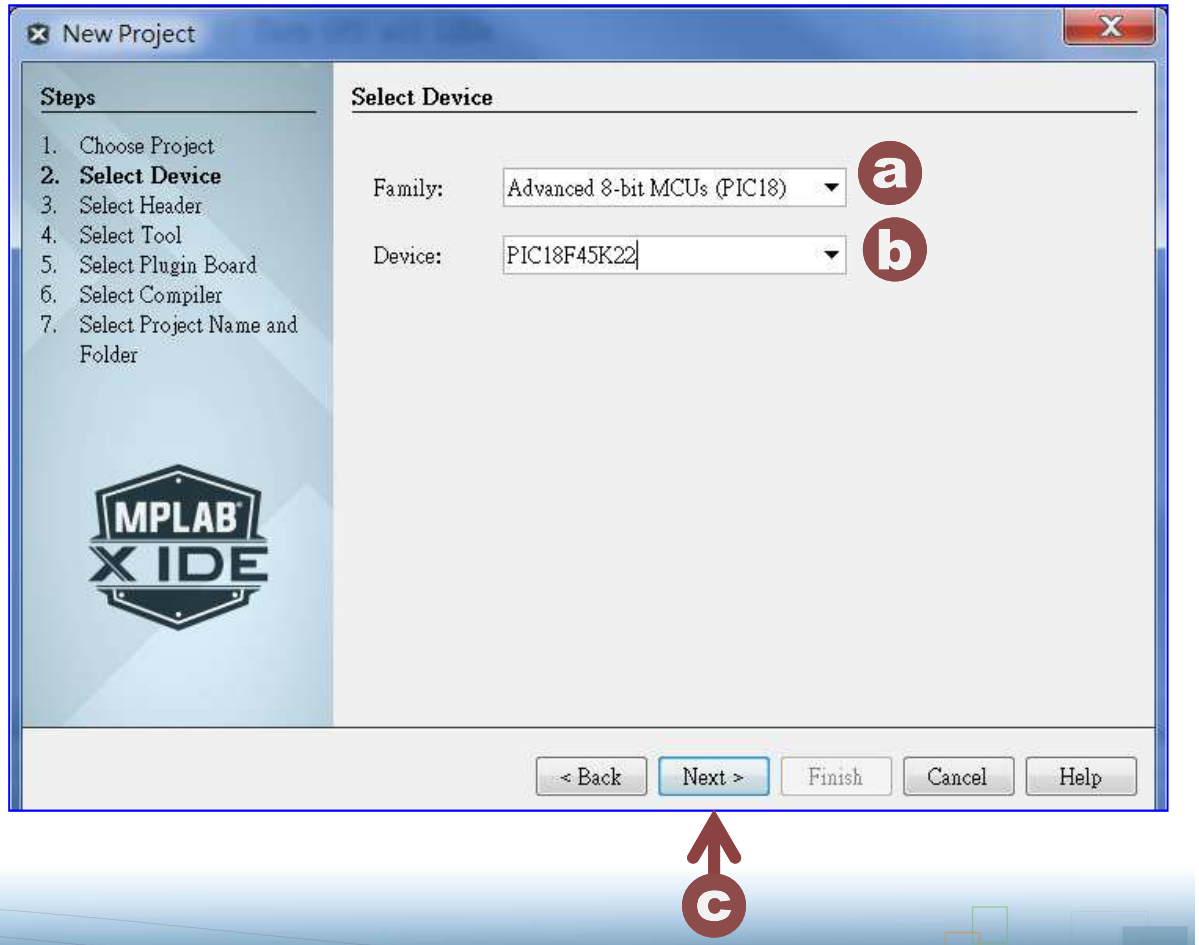
a 在 “**Family**”（系列）中選擇：

Advanced 8-bit MCUs (PIC18)

b 在 “**Device**”（元件）中選擇：

PIC18F45K22

c 按一下 **Next >**



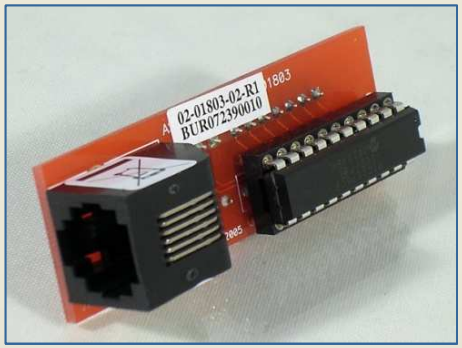
PIC101 Lab1.x

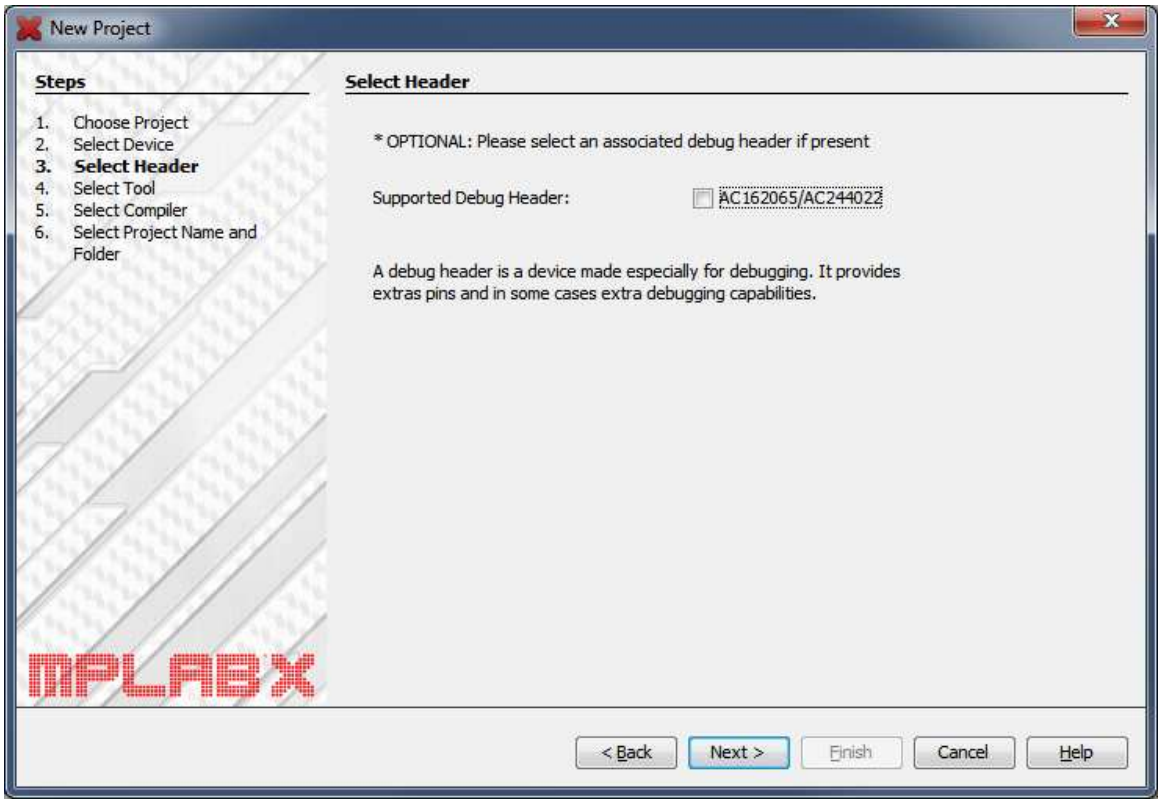
如何建立獨立專案

4 選擇元件轉接座 (Device Header)

本**練習**中使用的元件無需模擬頭。

自動略過此項目，但有些較少腳位的元件會有此需求。





按一下 **Next >**

PIC101 Lab1.x

如何建立獨立專案

5 選擇工具

a 選擇

點選 ICD3 下的序號

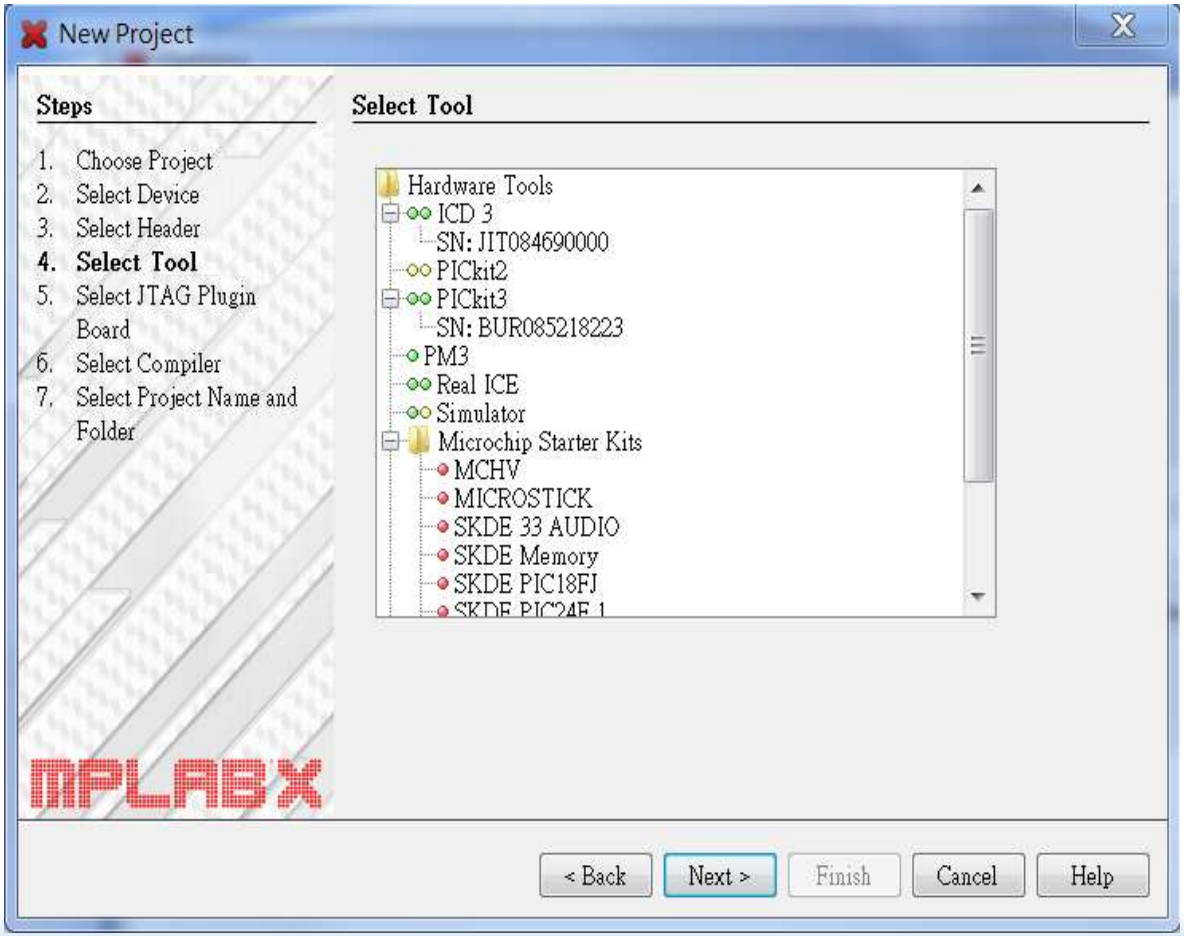
硬體工具必須插上 USB 槽，先讓 X IDE 辨識。

如果使用硬體除錯工具，則選擇其序號，如右側 **ICD 3** 下方所示。

在工具選用項裡可以看到有兩種工具可以選用:
一為 **ICD 3**
二為 **PICKit 3**

綠燈: 全功能已測試過
黃燈: 搶鮮版，部分功能待測
紅燈: 不支援

b 按一下 **Next >**

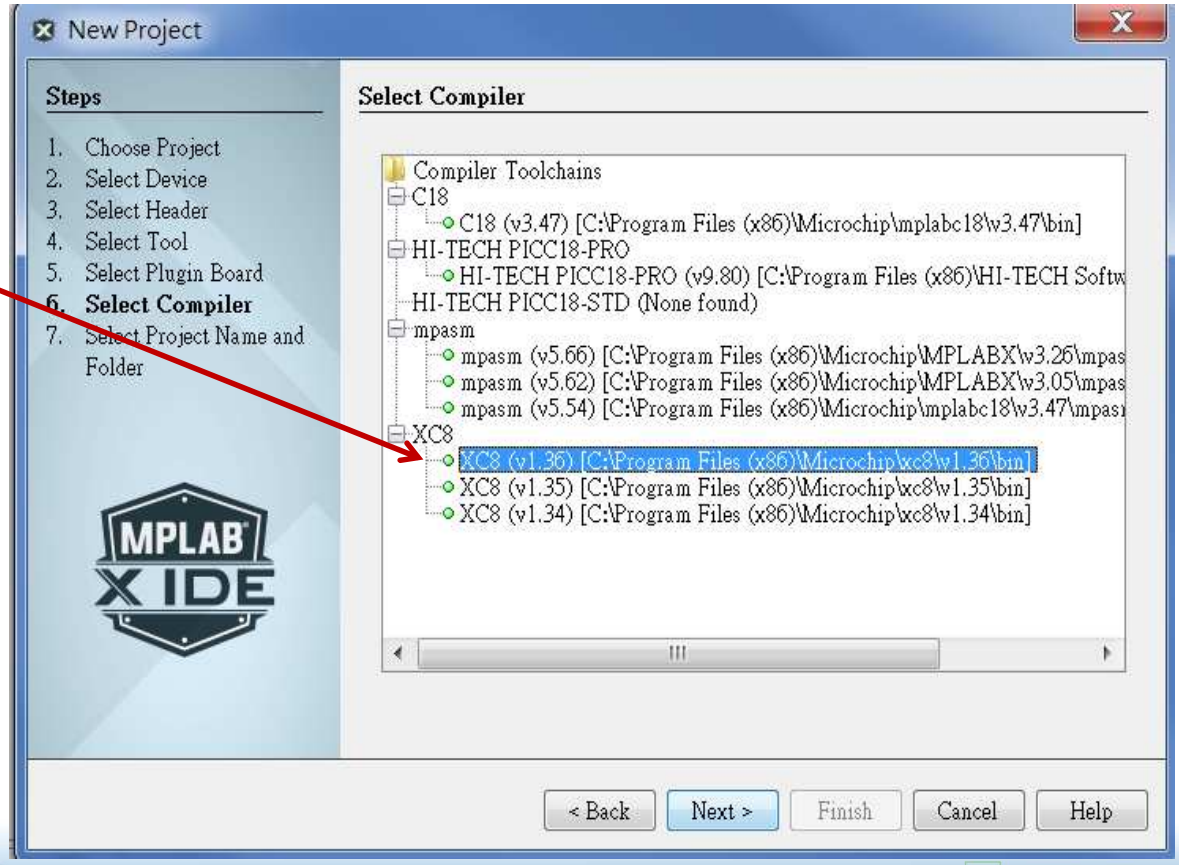


PIC101 Lab1.x

如何建立獨立專案

6 選擇 C 編譯器

a 按一下要使用的編譯器名稱下的編譯器版本。
實驗中請選擇 XC8 底下 XC8 (v1.3x)



如果在編譯器名稱下看不到版本號，則可能是編譯器未安裝或 X IDE 無法查找到該編譯器。

b 按一下 **Next >**

PIC101 Lab1.x

如何建立獨立專案

7 選擇專案名稱和資料夾及中文編碼格式

a 輸入*專案名稱*：
PIC101 Lab1

b 輸入*專案位置*：
D:\..\PIC101 Starter RTC

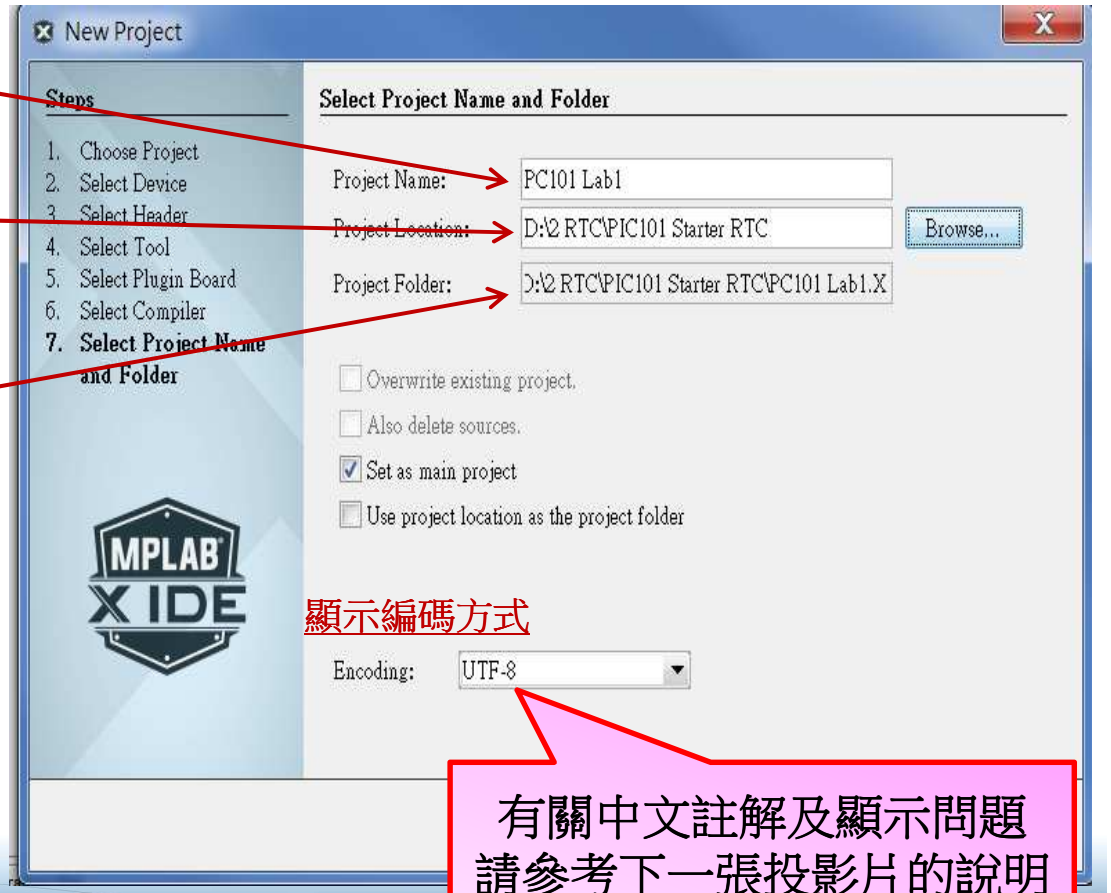
磁碟機 (D:)

xxxx

PIC101 Starter RTC

PIC101 Lab1.x ← 專案資料夾

c 按一下 **Finish**

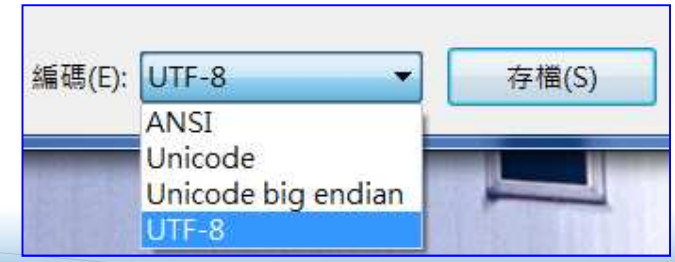


PIC101 Lab1.x

關於中文編輯與顯示

- 建議 **MPLAB X IDE** 使用 **UTF-8** 的中文編碼格式
  - 所以直接在 **X IDE** 下，直接撰寫程式是沒有中文顯示的問題
- 但 **MPLAB IDE v8.x** 的中文是使用 **Big-5** 編碼方式，除非在 **X IDE** 也一樣選用 **Big-5** 編碼來顯示，如果 **X IDE** 設成 **UTF-8** 會造成中文變成亂碼…
- 解決方法:
  - 將 X IDE 也設成 Big-5 碼的編碼方式
  - 將原先在 MPLAB v8.x 的原始程式檔 (*.C) 先用"記事本"開啟後再用 Save as 方式選擇 UTF-8 編碼後回存。

記事本選擇編碼後再回存



專案補充說明

- 到目前為止，我們建立了一個名稱為 **PIC101 Lab1.X** 的專案。但這專案仍然沒有程式的存在所以拷貝 **Lab1_Basic_LED_Delay.c** 到該專案的根目錄下。
  - ..\PIC101 Starter RTC\PC101 Lab1.X
- 接下來我們開始要加入相關的檔案到專案裡，當然這些檔案假設都已經事先寫好的程式並放置在 **PIC101 Lab.X** 的根目錄下。
- 當然你也可以開始用 **X IDE** 的編輯器開始撰寫程式後存檔在根目錄下後再加到專案裡。

PIC101 Lab1.x

將檔案分別加到專案裡

8 將檔案加入到新專案中

a 在專案夾中，利用老鼠右鍵後，從彈出功能表中選擇：**Add Existing Item…**（加入現存有的檔案…）

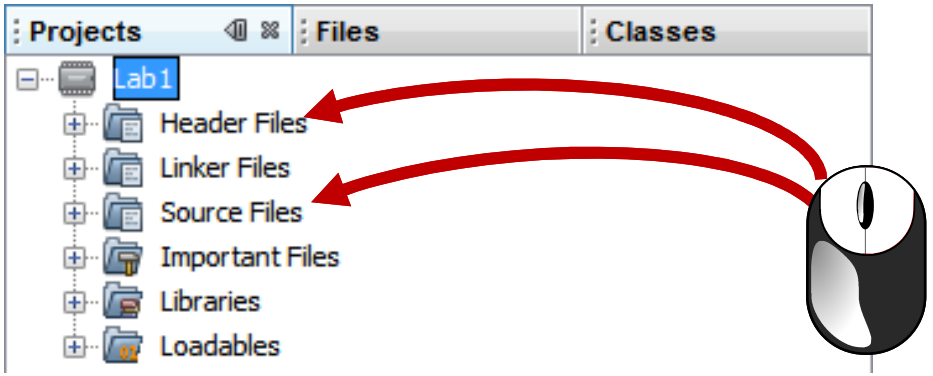
對於函式庫，從彈出選單中選擇:
Add Library/Object File（加入函式庫/目的檔…）

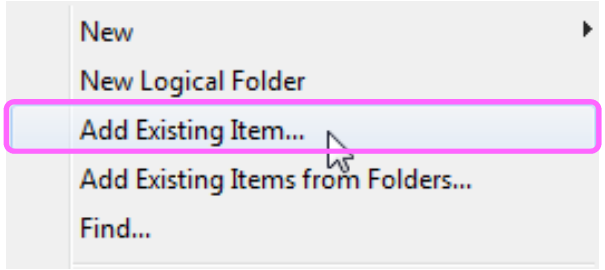
將相關檔案加到以下資料夾中：

b • **Header Files**
c • **Source Files**
d • **Libraries**

（詳見下一頁）

a





PIC101 Lab1.x

如何建立獨立專案

8 將相關檔案加入到新專案中（續.....）

*從 **PIC101 Lab1.X** 目錄中加入下列文件*

b **Header Files**

未使用

c **Source Files**

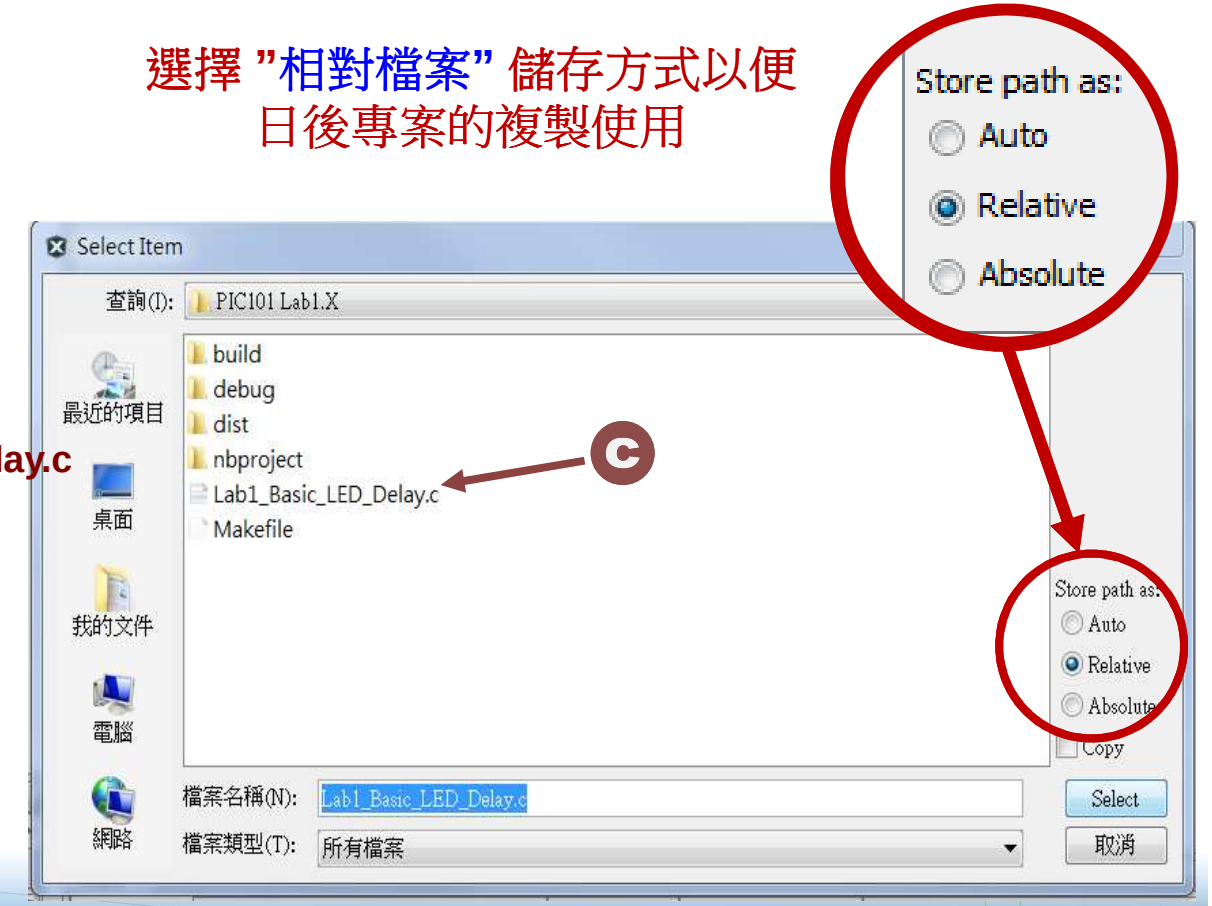
Lab1_Blank_KED_Delay.c

xxx.c

Libraries

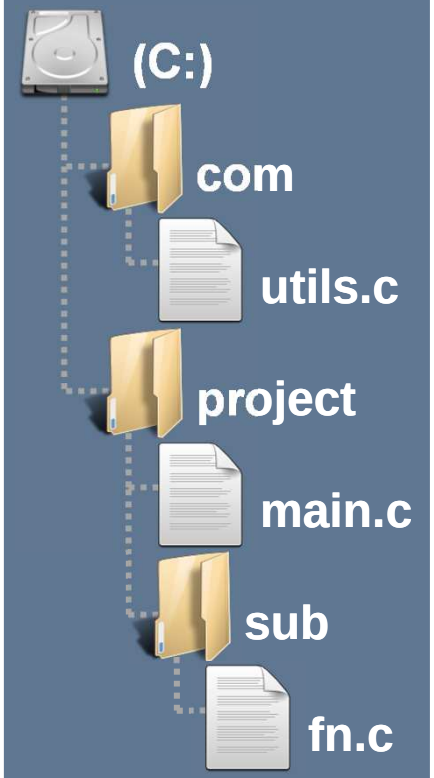
未使用

選擇 ”相對檔案” 儲存方式以便日後專案的複製使用



PIC101 Lab1.x

相對檔案路逕說明

儲存路徑 範例	絕對路徑 Absolute	相對路徑 Relative	自動 Auto
	..\ = 上移一層 .\ = 專案目錄		
專案目錄以外			
	C:\com\utils.c	..\com\utils.c	C:\com\utils.c
專案目錄內			使用絕對路徑
	C:\project\main.c	.\main.c	.\main.c
專案目錄的子目錄內			使用相對路徑
	C:\project\sub\fn.c	.\sub\fn.c	.\sub\fn.c
			使用相對路徑

PIC101 Lab1.x

編譯專案及燒錄

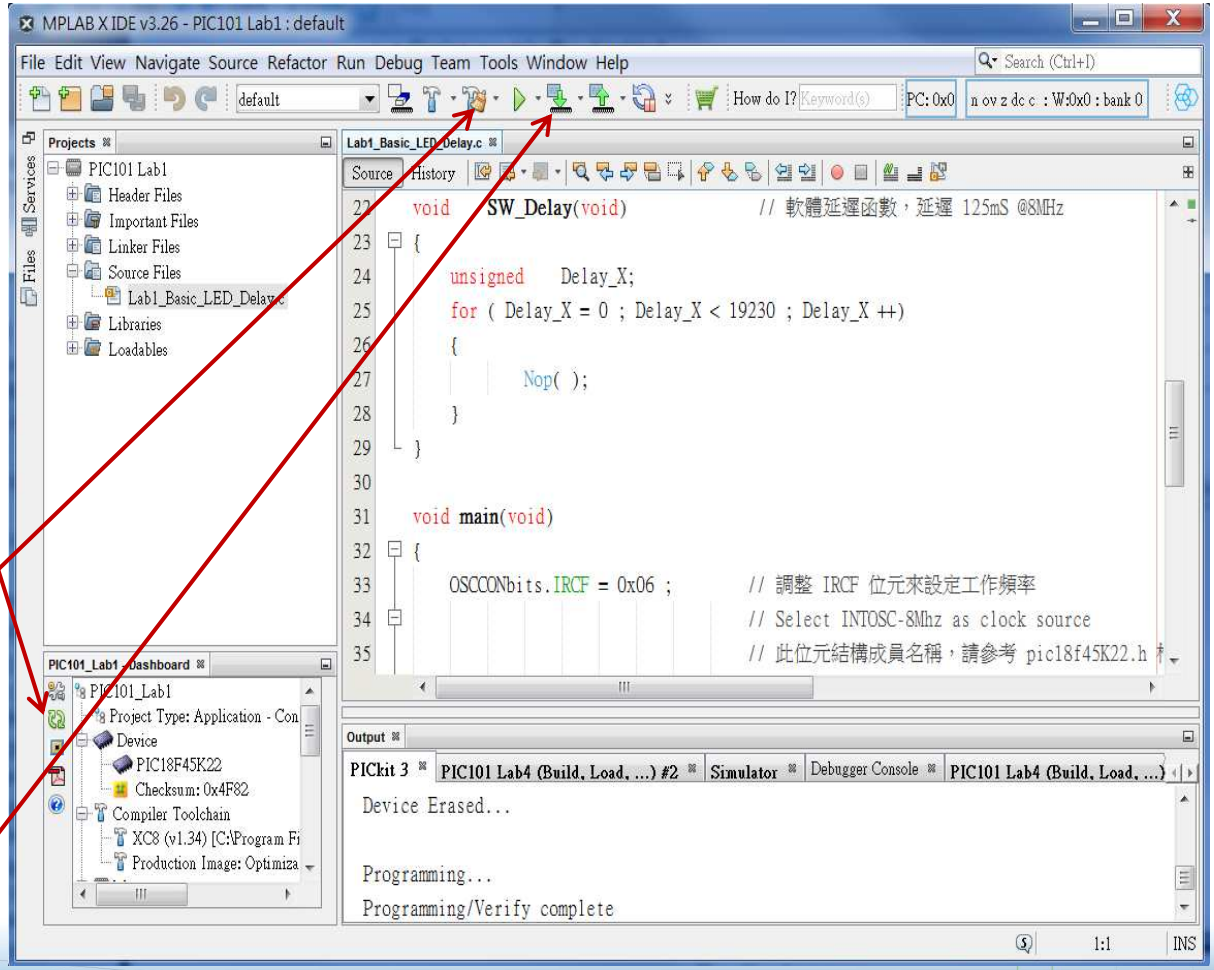
9 編譯、燒錄和執行

編譯專案，確保每一項都正確完成。

1. 按一下工具連線圖示確定韌體更新及連線的狀態

2. 按一下 Clean And Build Project (編譯專案) 圖示確定在 “Output” 視窗顯示編譯成功。

3. 編譯成功後再按右邊的圖示 Program Target（燒錄目標元件) 執行程式碼 (Hex) 的燒錄

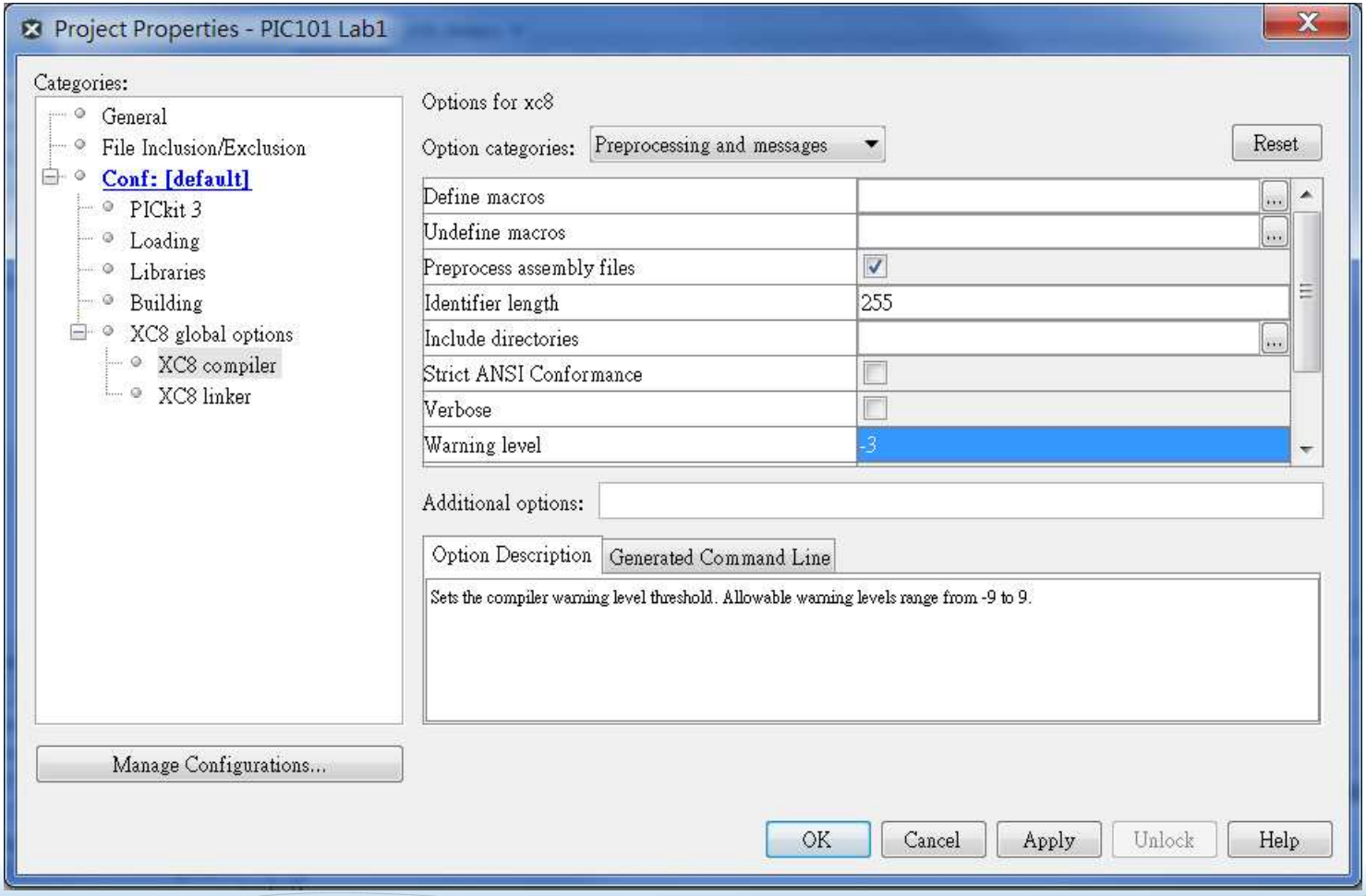


編譯的問題

- **Warning Level = -3 (-9 ~ 9 的範圍)**
  - Advisory Message (提示訊息)
  - Warning Message (警告訊息)
  - Error Message (錯誤訊息)
  - Fatal Error Message (致命錯誤訊息)
- **PC101 Lab1** 在 **XC8 v1.34** 以後的版本編譯後會產生一堆 **Warning Message**
  - 設定 Waring Level =0

修改 XC8 編譯警告訊息

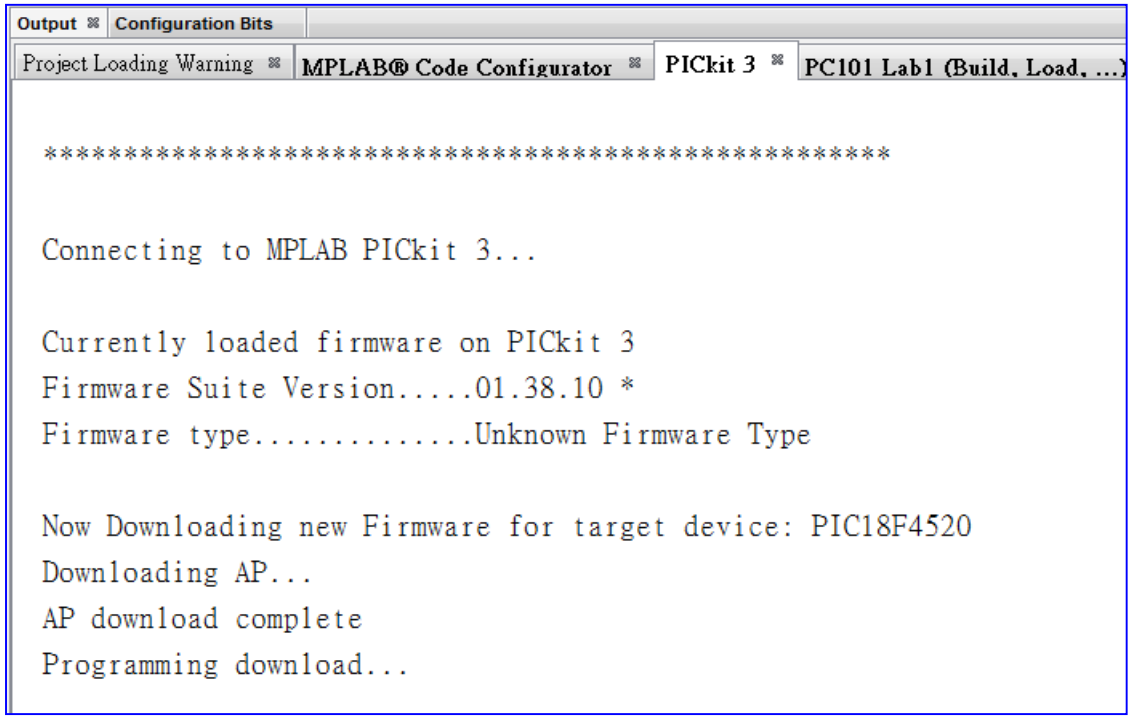
- 用右鍵點選專案名稱 **“PIC101 Lab1”**
- 在彈出的新視窗再點選 **“Properties”** 即可



燒錄時的問題

PICkit3 / ICD3 更新元件韌體

- **PICkit3/ ICD3** 會自動檢查目前 **X IDE**所使用的元件來決定是否要更新目前的 除錯/燒錄 韌體。
- 右圖為 **PIC18F** 系列的韌體更新的顯示。
- 韌體在更新時 ，不可中斷 **USB** 連線，否則會造成工具的損壞。



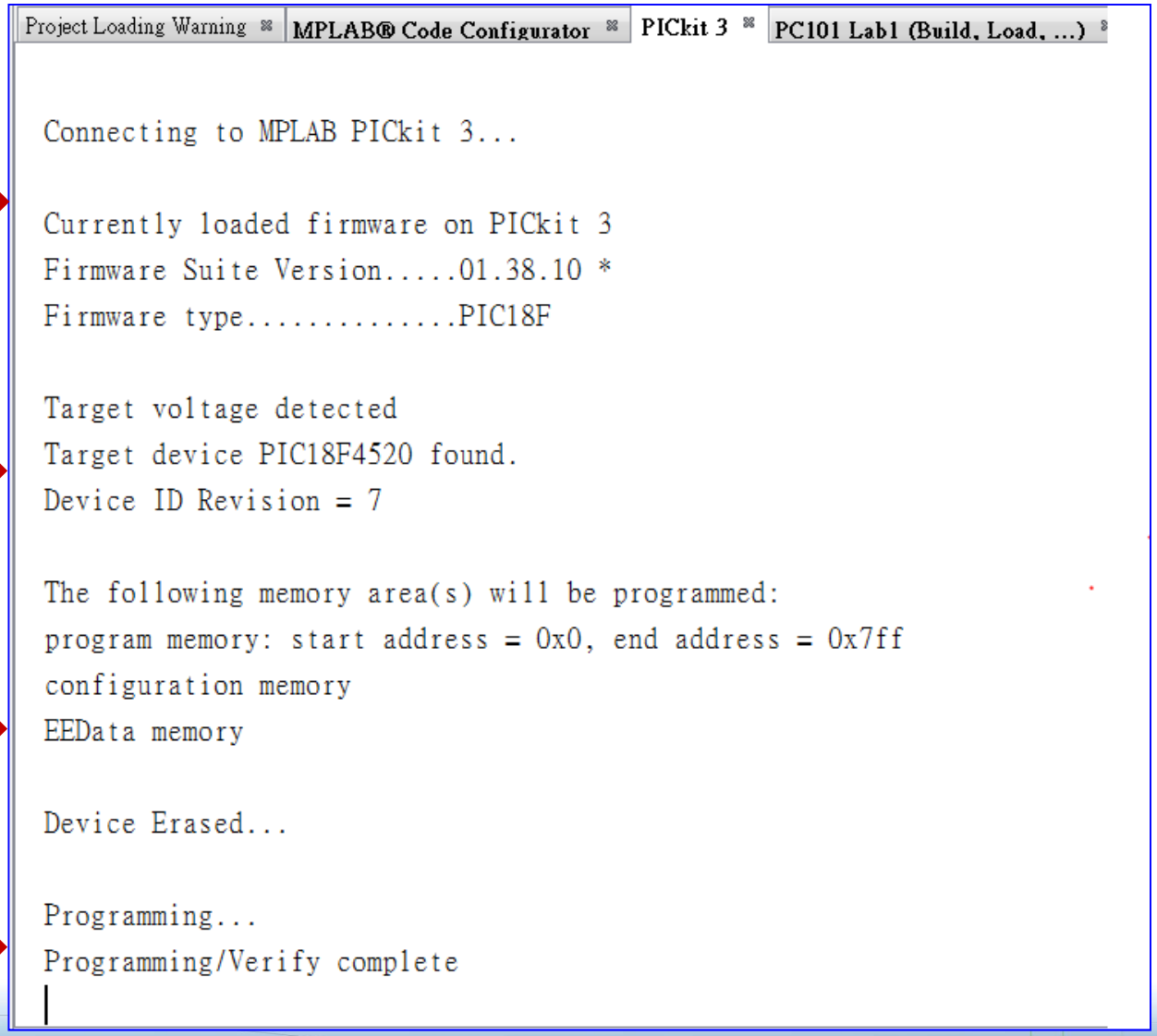
燒錄成功的顯示

建立連線

尋找燒錄元件及版本訊息

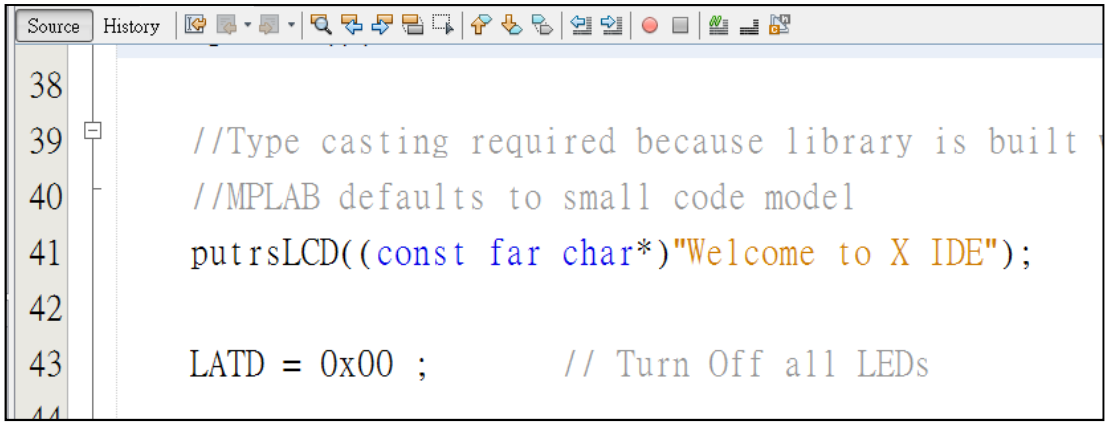
清除所有的記憶體

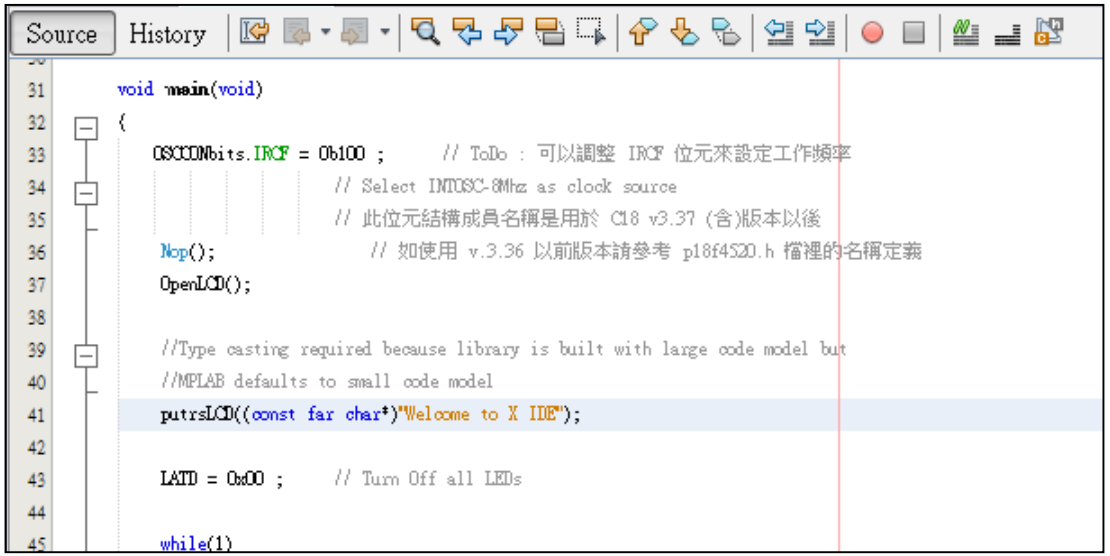
燒錄成功



編輯區的文字放大

在編輯區的所顯示的文字大小是可以改變的。
先按下 **“Alt + 滾輪”** 即可隨意改變字型大小。此方式也可以使用在 **“Output”** 視窗



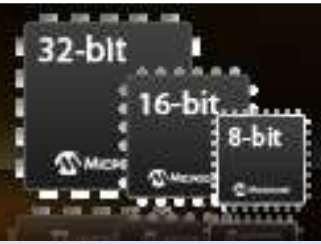


PIC101 Lab1.x

結果

如果所有操作 (編譯，燒錄) 均正確完成，則可以看到版子上的 LED1 ~ LED8 (PORTD) 以每 0.125 秒的時間做加一的計數。

PORTD LED 每 0.25 秒 進位一次



PIC101 Lab1.x

基本除錯

斷點的使用

PIC101 Lab1.x

試著用除錯模式

我們即將介紹除錯的基本功能，接下來請使用 **PIC101 Lab1** 的專案實作一下除錯的動作及變數的觀察

X IDE 基本除錯

- 了解除錯的圖示
  - 除錯模式的初始設定
  - 開起除錯模式
  - 離開除錯模式
- 變數的觀察
- 程式裡設定斷點
- 觀察 / 修改 記憶體內容值
- 計時碼表功能 **(Stopwatch) –** 軟體模擬下

使用主工具列圖示

預設按鈕

New File（建立新文件）

New Project（建立新專案）

Open Project（打開專案）

Save All（全部儲存）

Undo（恢復）

Redo（重做）

Build（編譯）

Rebuild (Clean and Build)
重新編譯（清除並編譯）

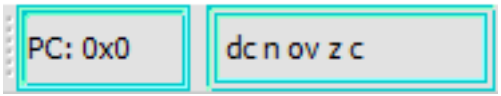
Program Target（燒錄目標元件）

Read Target（讀取目標元件）

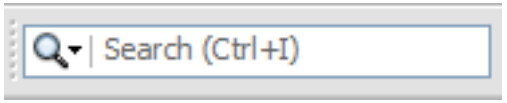
Hold in Reset（保持重置）

Debug (Build, Program, Run)
除錯模式（編譯、燒錄並執行）

專案設定



程式計數器和
狀態位元



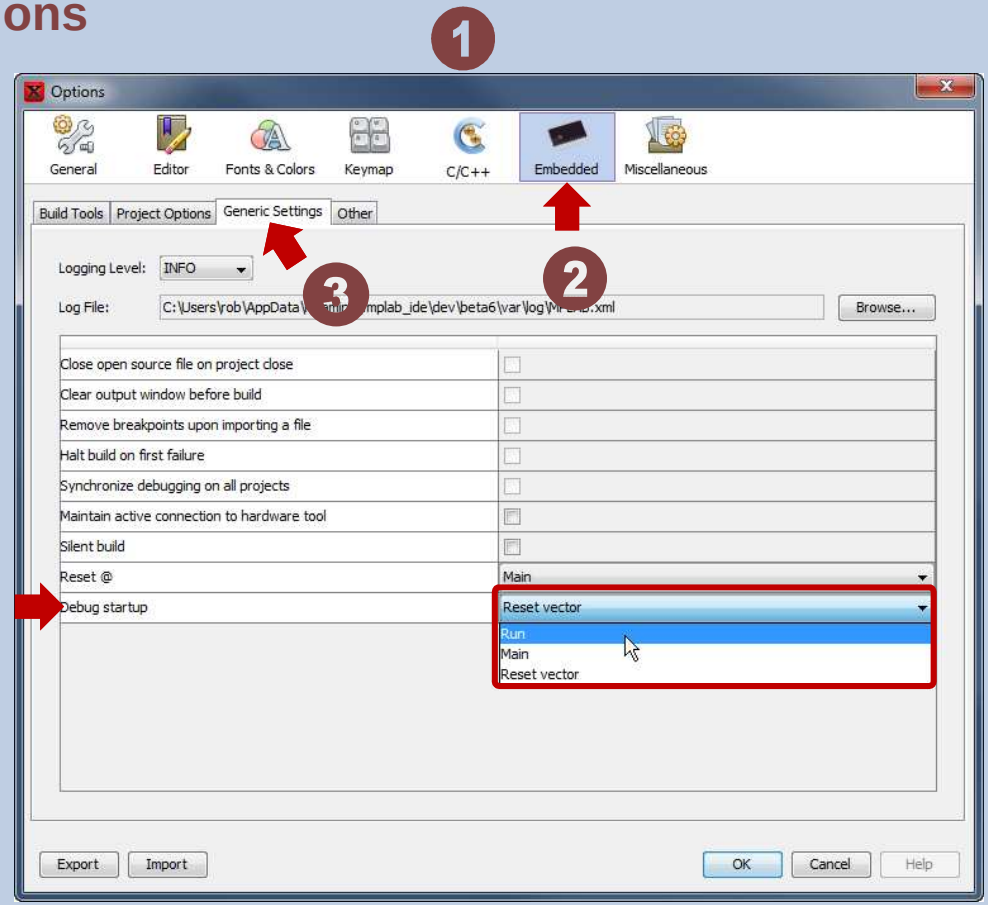
快速搜索

除錯模式的初始設定

如果您想或不想除錯器自動開始執行：

1. 從主功能表中選擇：**Tools ▸ Options**
2. 選擇 **Embedded** 圖示
3. 選擇 **Generic Settings**（普通設定）選項卡
4. 對於 **Debug startup**（除錯啟動）設置，選擇：

   1. ***Main (Reset 後到 Main)***
   2. ***Reset vector* (停在 Reset 位址)**
   3. ***Run (Reset 後直接執行程式)***



基本除錯 – 啟動除錯模式

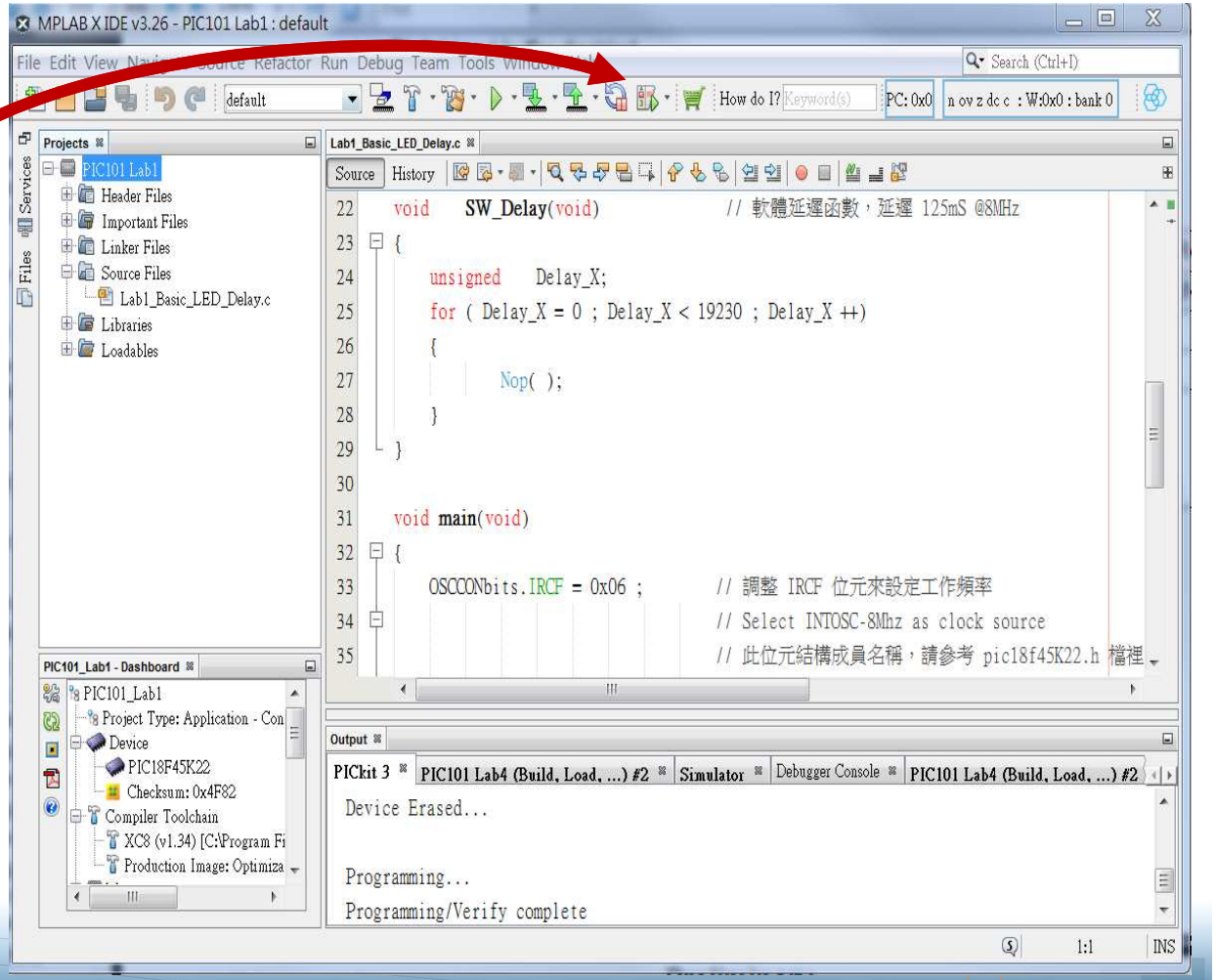
專案的基本除錯

編譯專案，確保每一項都正確完成。

按一下 **Debug Project 除錯專案）** 圖示。

該按鈕的功能：

1. 在 *除錯* 模式下編譯（make）專案
2. 重新與 ICD3 連線
3. 將程式燒錄至目標板上的PIC18F45K22 元件
4. 執行 C 啟動程式 (游標停在 main( ) 函式)
5. 按繼續執行圖示開始執行程式



如何啟動除錯程序

Debug Project（開啟除錯模式）

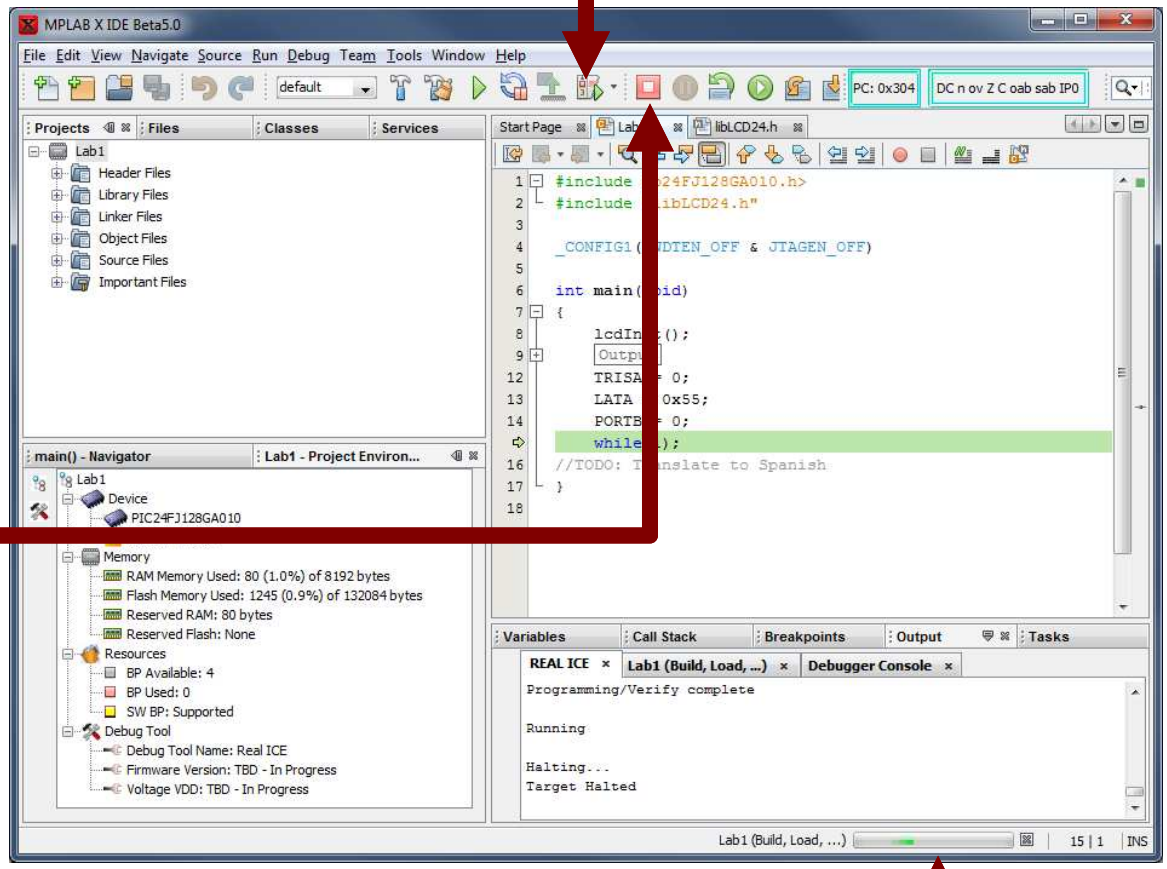
啟動除錯程序。

1. 在除錯模式下編譯專案。
2. 連接有硬體除錯器，該操作將建立USB連接
3. 燒錄目標元件
4. 並執行除錯監督程式。

End Debug Session（除錯程序結束）

中止除錯對話。

1. 斷開硬體除錯器
2. 關閉 USB 的連接。
3. 該操作必須在修改程式並繼續進行除錯前下達中止除錯程序。



對話啟動時，
進度條會有顯示

MICROCHIP
Regional Training Centers

除錯工具的圖示列

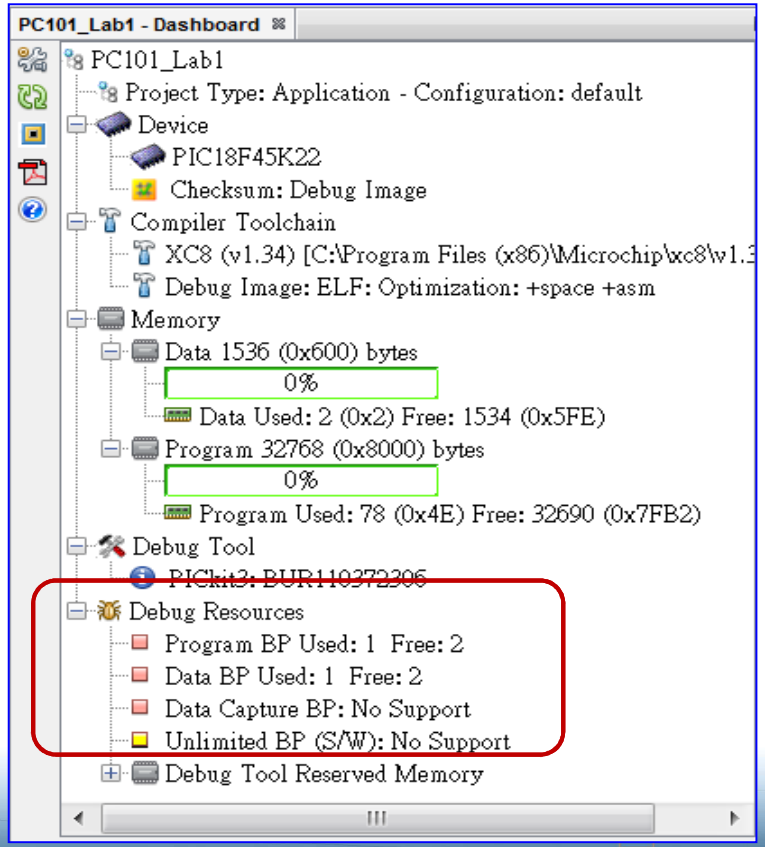
功能	MPLAB® 8	MPLAB® X
結束除錯步驟	無	
暫停		
執行/繼續		
連續單步執行		無
執行程式至游標位置處	在上下文選單中	
單步執行		
單步跳離函數執行		
重置		
游標回到目前 PC 位址	無	

元件的硬體斷點支援

定義說明

硬體斷點 **(Hardware Breakpoints)**：在目標微控制器裡使用硬體架構所建構出的比較暫存器. 一但與被監看的位址(或資料) 比對相符合時，程式立即暫停執行並將相關資料轉移到除錯用的監督程式。

- 有限制的硬體斷點數
  - 通常 1 ~ 4 個斷點可用
  - 實際可用的斷點數取決於元件的內部設計
- 使用儀表版視窗 顯示 **(Dashboard)** 可用斷點數的查看
  - Menu: Window ▸ Dashboard
  - PIC18F45K22 有三個程式硬體斷點，資料斷點也有三個



斷點 vs 中斷

- 斷點與中斷是不一樣的
  - 斷點是為除錯所設計的利器，用來暫停程式的執行以利分析程式執行的結果。
  - 中斷 (Interrupt) 程式執行中，因突發事件暫時跳離去處理事件，完畢後再返回原處。
- 斷點分類
  - 軟體斷點 – 會占用 CPU 的資源。
    - **軟體模擬、ICD3、Read ICE**
  - 硬體斷點 – 由 RAM 比較器等硬體電路完成
    - **PKOB 、PICKit3、ICD3、Read ICE**

除錯時斷點為何很重要

- 斷點的設定可以讓程式執行到這裡時暫時停止下來
  - 查看變數的內容，修改變數值
  - 檢驗周邊暫存器值
  - I/O 腳位的變化
  - 配合單步執行  模式分析程式執行及驗證其正確性
  - 只在除錯模式有效

其它 PIC 硬體斷點數
內建允許使用個數

Devices	Number of Hardware Breakpoints
PIC10F / 12F / 16F	1
PIC16F1xxx enhanced	3
PIC18F	1
PIC18FxxK enhanced	3
PIC18FxxJ	3 or 5 *
dsPIC30F / PIC24F	1 to 5
dsPIC33F / PIC24H	2, 4, 6, or 10
PIC32MX	6

*** There is a limitation for these devices that only 1 data capture is available.**

除錯斷點基本設定

如何設置或清除斷點

斷點設定

按一下圖符頁邊上的行號，以設置（當前無斷點）和清除（當前有斷點）此行上的斷點。

一個紅色正方形（■）表示斷點已設置。對應的程式行也將以紅色背景高亮顯示。

綠色為目前 PC 暫停的位置

在行號上按一下以切換斷點 有 / 無 的交互設定

行號斷點

如何配置行號的斷點

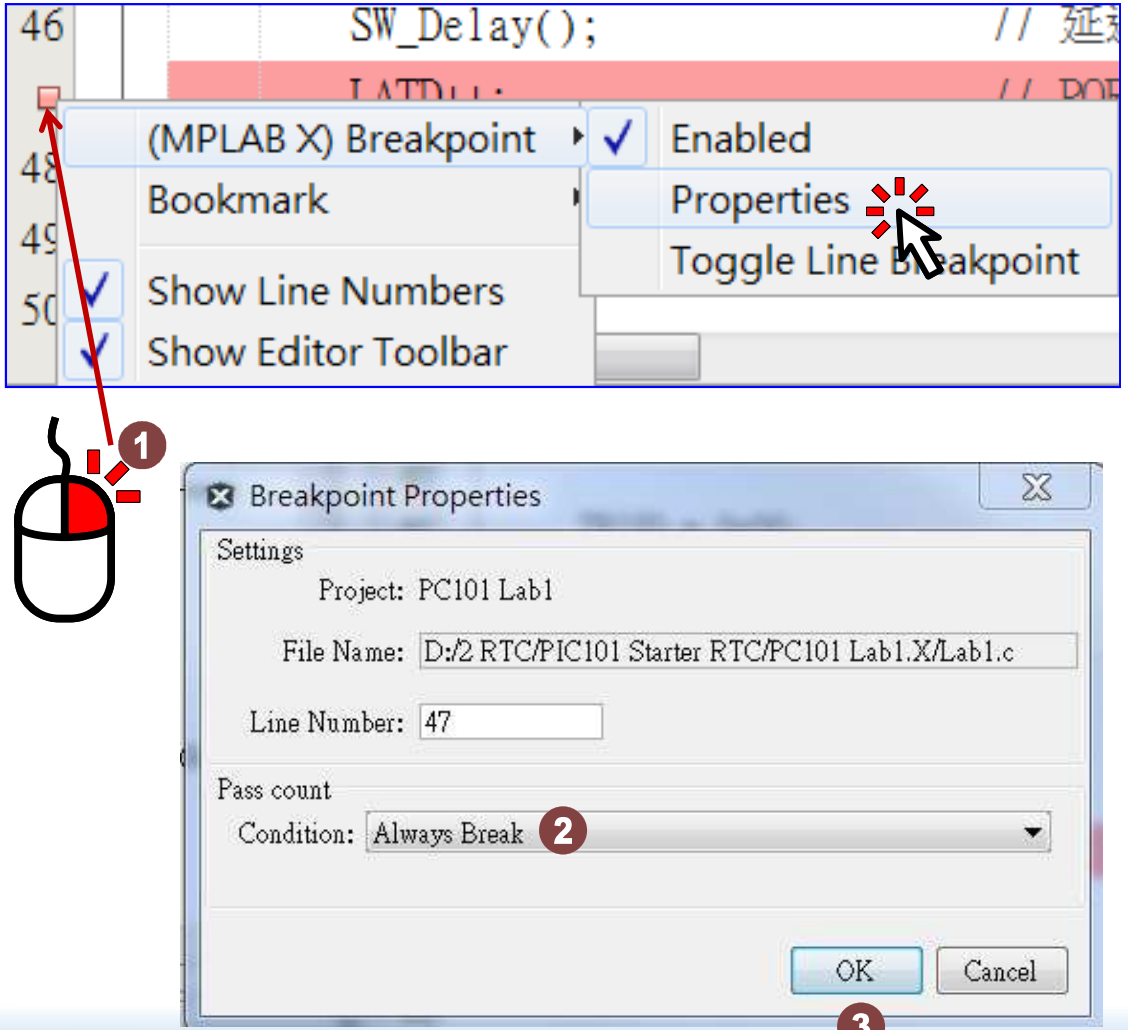
步驟

1. 在程式的編輯畫面，用老鼠的右鍵點選一下 的斷點設定圖示。
   Select ：(MPLAB X) Breakpoint ▸ Properties 彈出對話選擇視窗 **“Breakpoint Properties”**

2. 選擇 **“ Pass Count”** 條件

   Always Break
   - Always Break
   - Break occurs Count instructions after Event
   - Event must occur Count times

3. 最後選取 **OK**



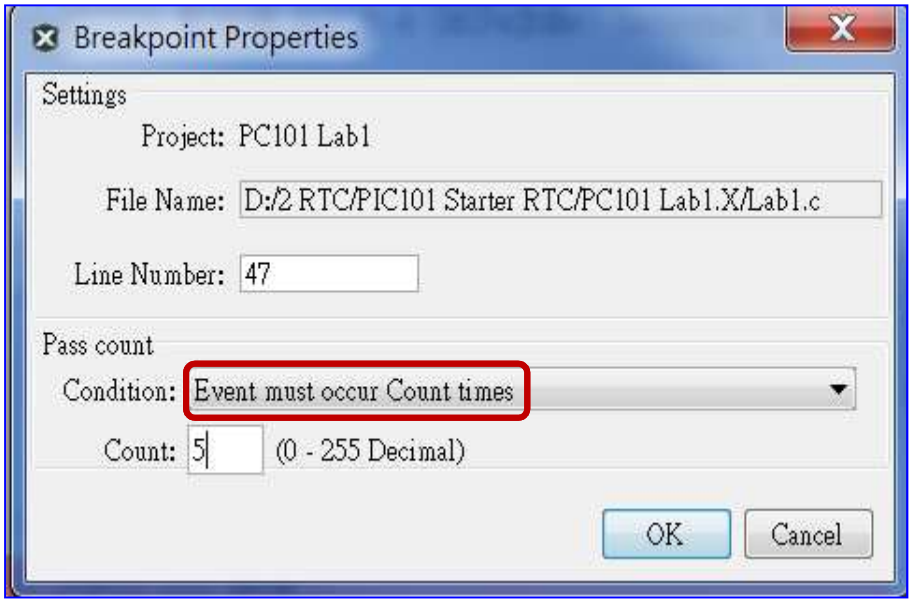
斷點次數的設定

客制化的選項：斷點通過次數

Definition

Pass Counter：是客制化的設定輸入值，主要是用來計測程式執行時通過此斷點的次數。通常程式經過此斷點並不會停下來直到你所特別指定的通過次數相符合時程式才會暫停下來。

- **PICKit3 “Pass Count”** 的選項:
  - Always Break (no pass count)
  - Event must occur *Count* times
- 不同的除錯工具所共的斷點除錯功能也會不同
  - REAL ICE > ICD3 > PK3

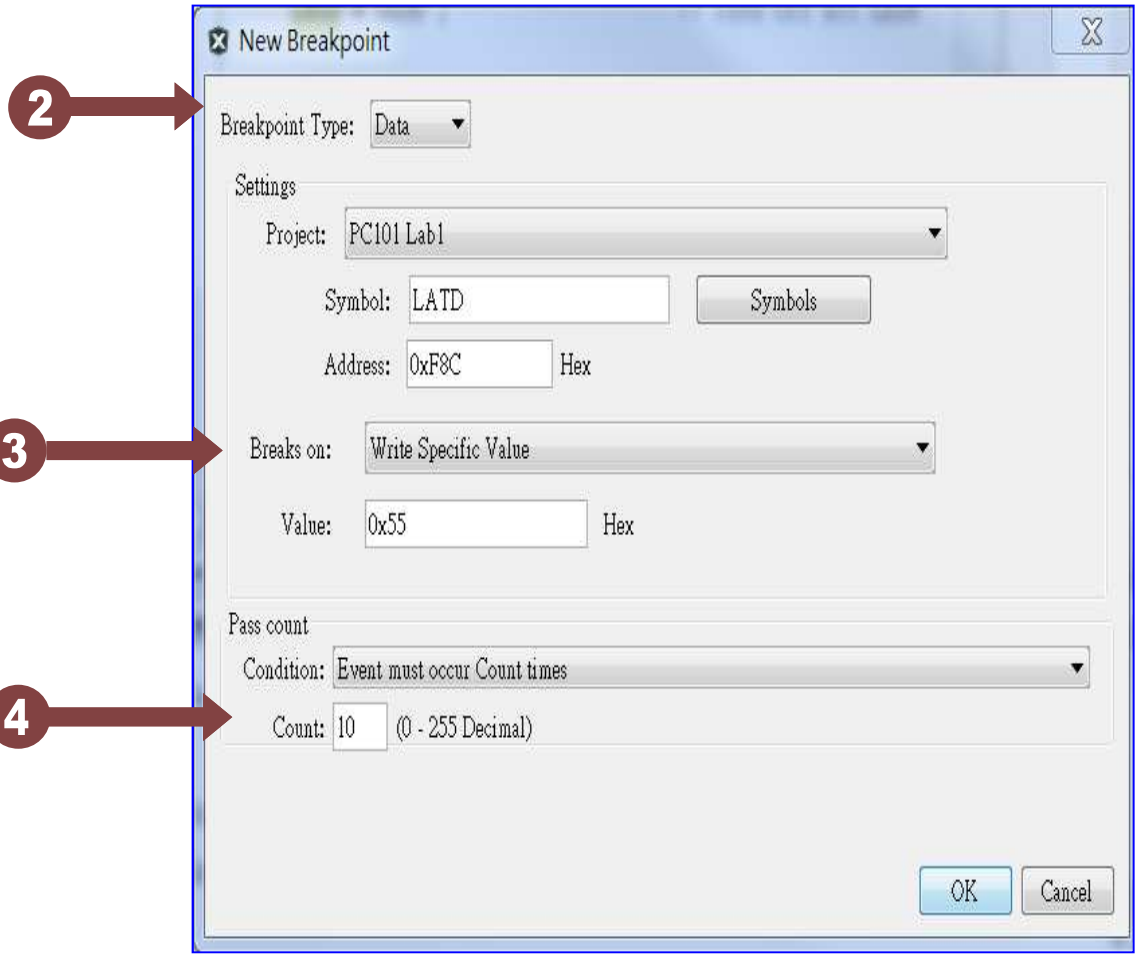


中斷點

如何設置條件性資料中斷點

條件性資料斷點設定

1. 從功能表中選擇 **Main: Debug▶New Breakpoint**
2. 選擇 **Data（資料）**作為中斷點類型
3. 指定資料 **Address（地址）**和 **Breaks on（中斷條件）**設置
4. 指定 **Pass count Condition**



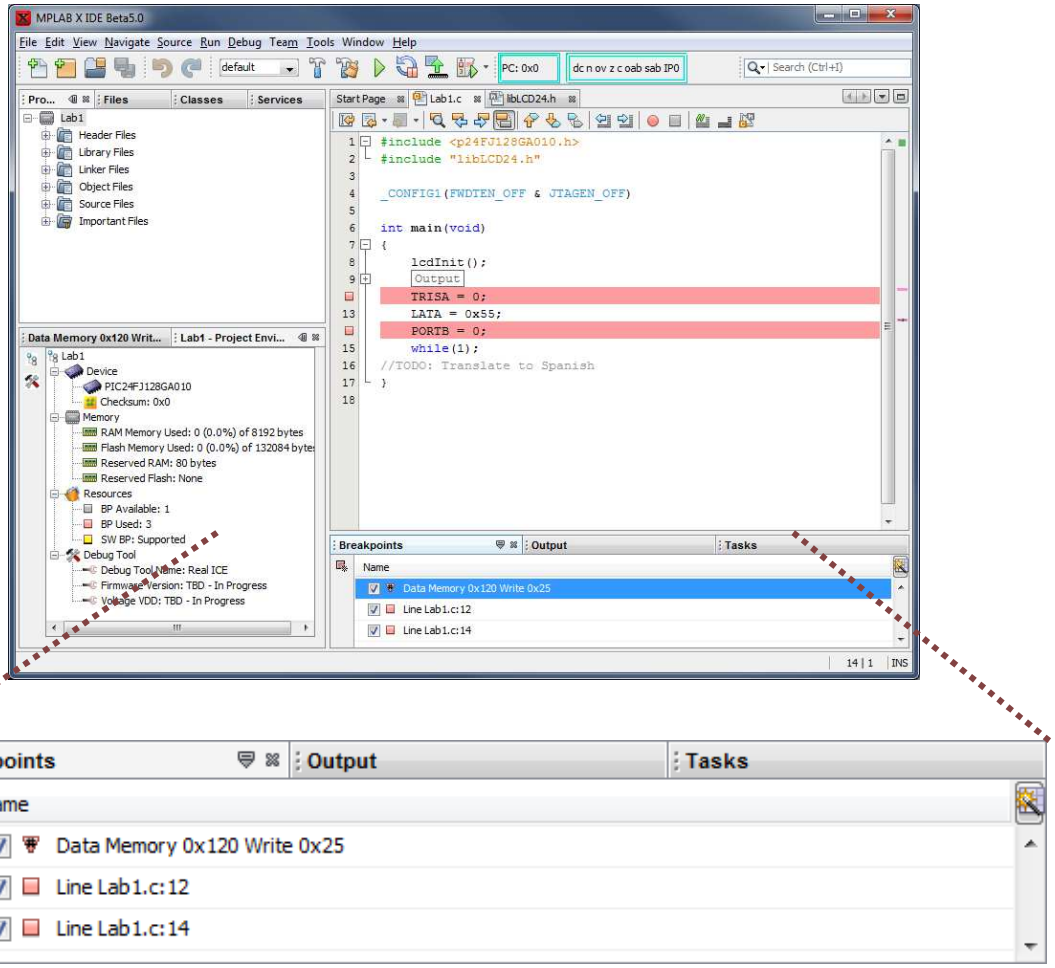
斷點查看

如何查看專案中的所有斷點

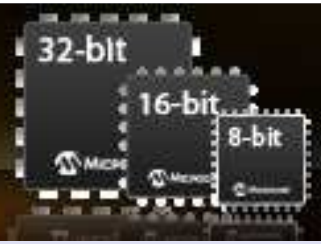
顯示Breakpoint（斷點）視窗

1. 選擇 **Windows ▸Debugging ▸Breakpoints**

2. 按右鍵以：
   - 建立新斷點
   - 使能所有斷點
   - 禁止所有斷點
   - 刪除所有斷點
   - 組合斷點（允許按組 使能/禁止）
   - 更改斷點設置



按兩下斷點可轉至原始檔案



PIC101 Lab1.x

基本除錯

變數的觀察

為何變數觀察很重要

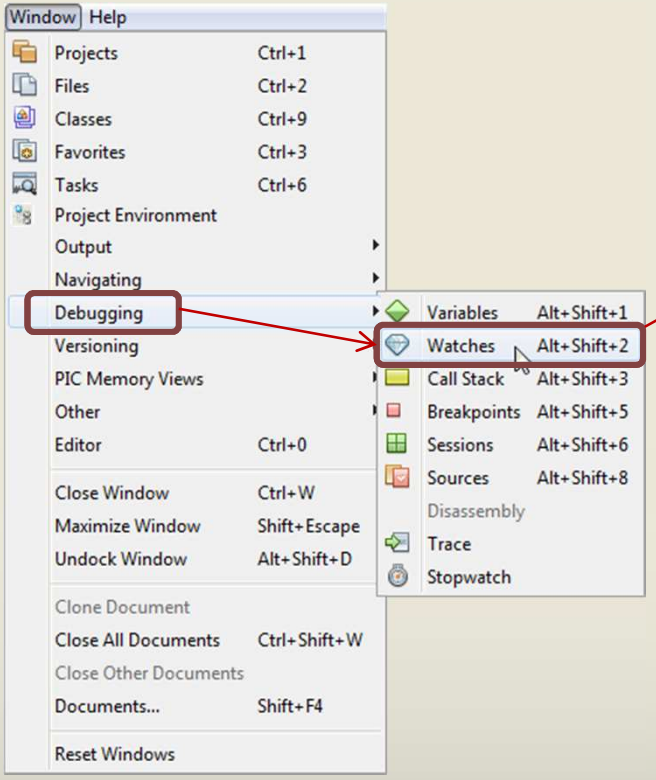
- 變數的觀察一定要將程式暫停下來，通常是配合斷點或按下暫停圖示  **(Halt)**
- 因為 **CPU** 在執行程式後內部的 **RAM** 被改變時不會即時更新到 **X IDE** 的 **RAM**。
- 建議直接縮小範圍直接用變數觀察視窗 **(Watch Window)** 來觀察變數，只要暫停資料立即更新。
- **Watch Window** 可觀測的內容
  - **bit, char, int, long, float, double**
  - **struct, union, array, ROM array, point**
  - 二進制，八進制，十進制，十六進制
  - 字元，字串，浮點數

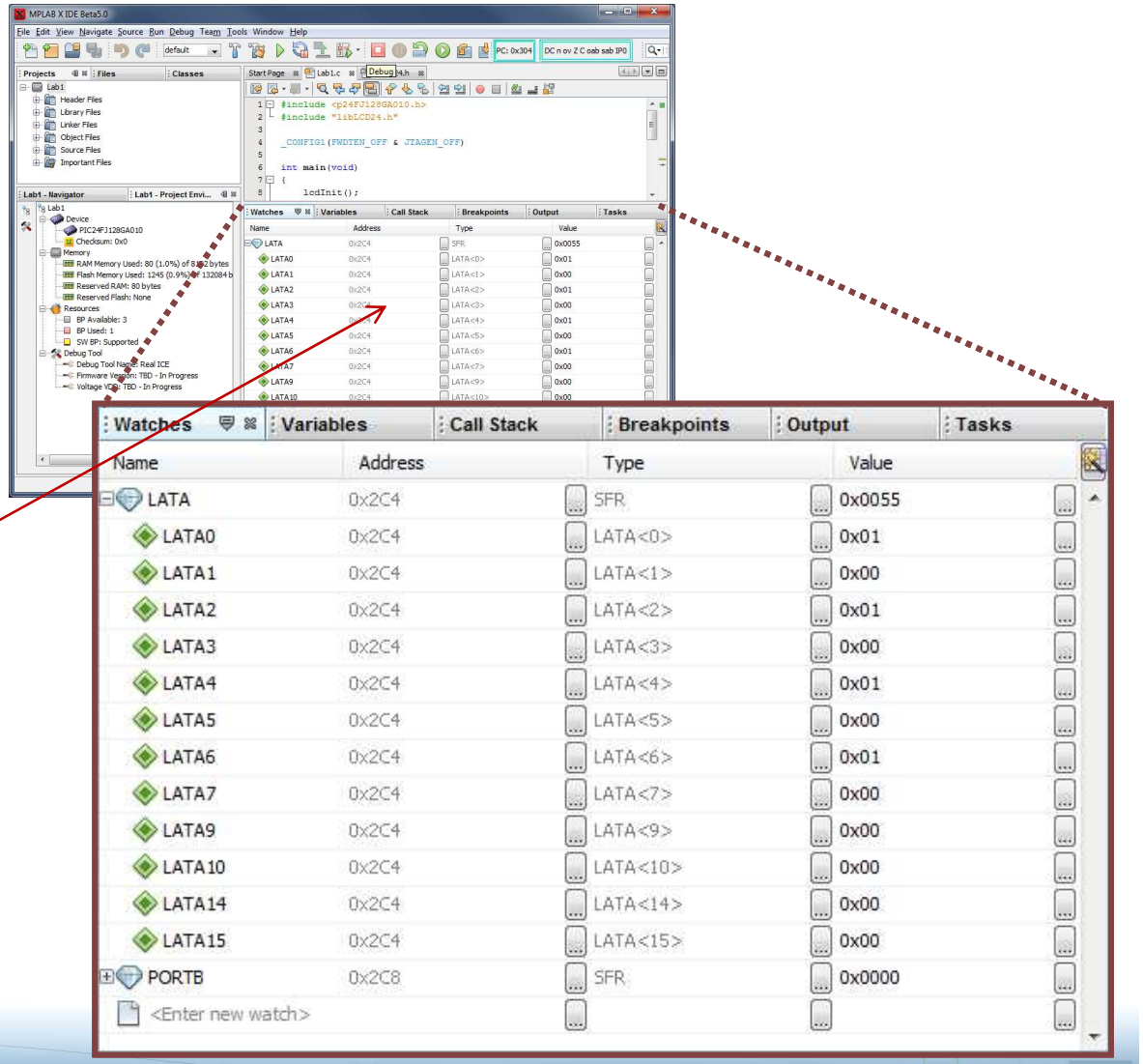
觀察變數

如何顯示 **Watches**（觀察）選項卡

從主功能表中選擇：

Window▸ Debugging▸ Watches





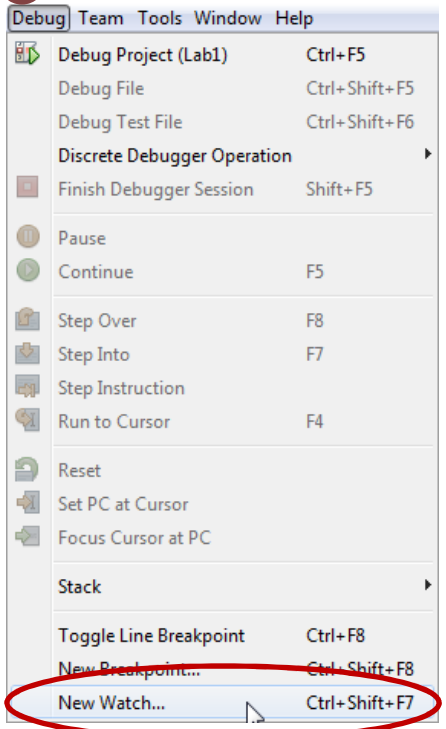
觀察變數

如何向 **Watches** 選項卡中加入變數

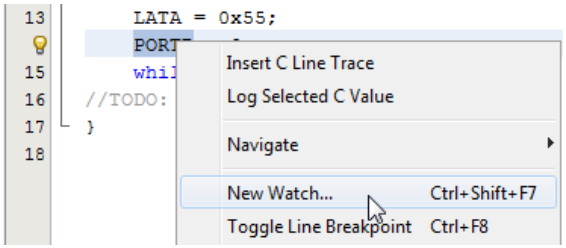
1. (可選）在編輯器中，使用老鼠來圈選變數
2. 執行以下一種操作：
   - a 在編輯器中按右鍵並從彈出功能表中選擇**New Watch...**（新建觀察變數……）
   - b 從主功能表中選擇：**Debug▸ New Watch...**
   - c 在編輯器中高亮顯示一個變數，然後點擊並將它拖動到**Watches**視窗
   - d 

3. 輸入變數名或僅接受所顯示的名稱並按一下 OK

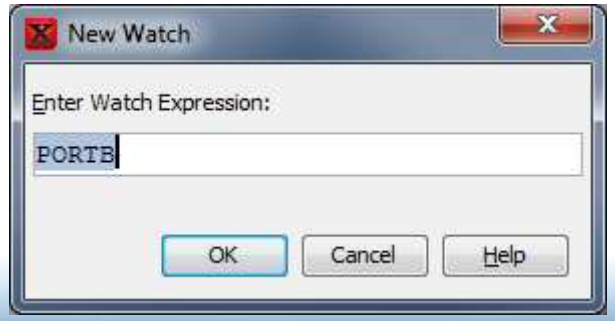
2b



2a



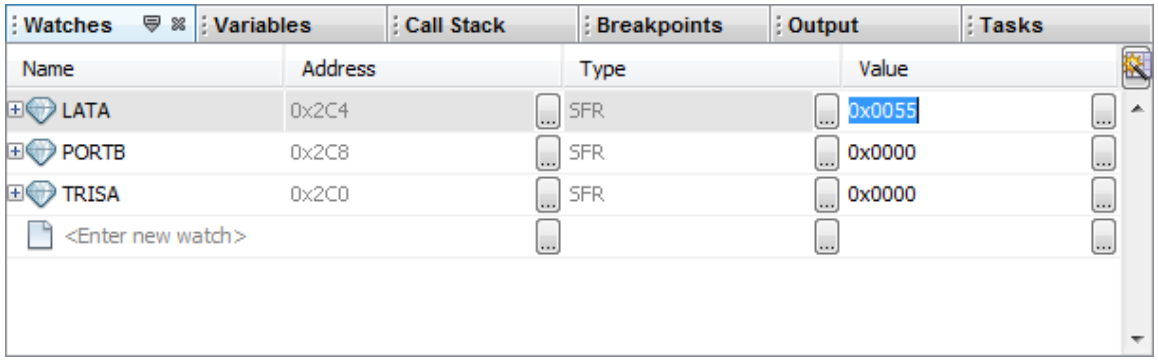
3



觀察變數

如何更改觀察變數的值

1. 按兩下**Value**列中的一個值
2. 輸入新值，完成後按**Enter**鍵

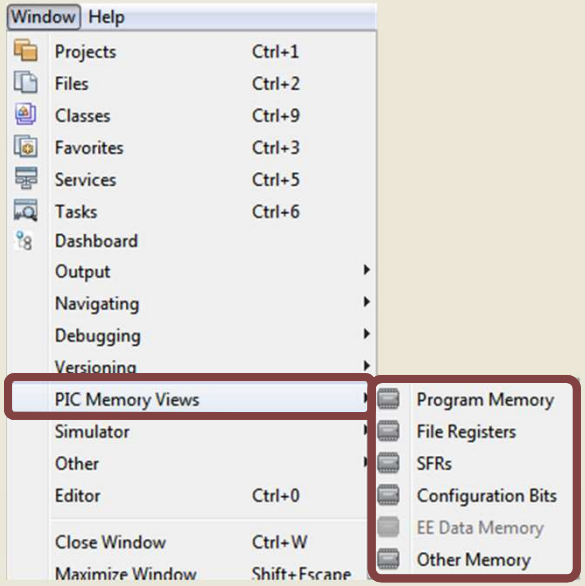


無法動態更改值。要進行更改，您必須首先暫停程式執行。

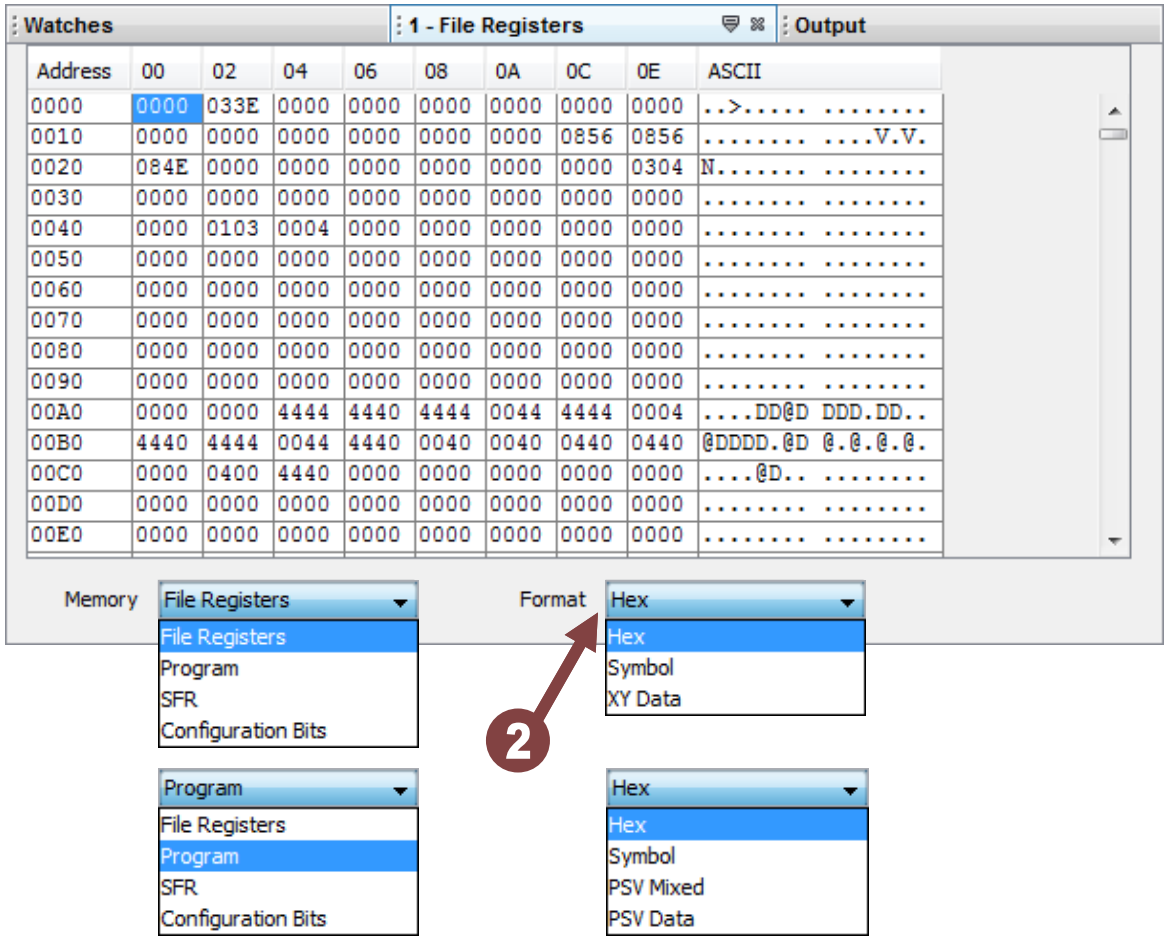
記憶體視窗

如何查看嵌入式記憶體

1. 從主功能表中選擇：
   Window ▸ PIC Memory Views ▸



2. 選擇顯示格式

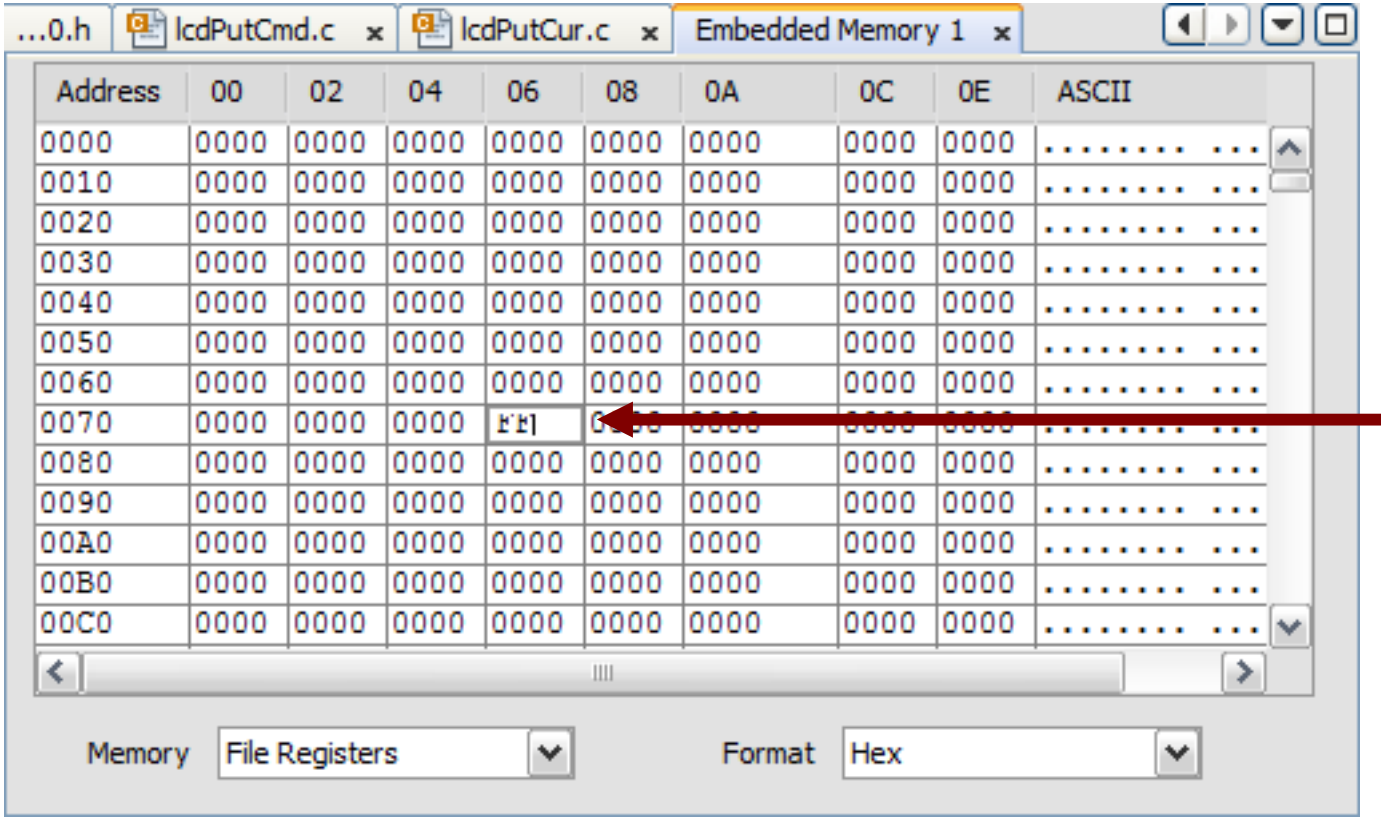


可對每個記憶體視圖視窗進行設定，以顯示任何支援的記憶體類型。

此顯示視窗以 **16-bit** 的元件為範例

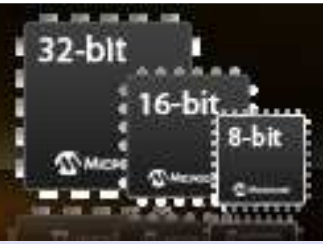
記憶體視窗

如何更改儲存單元的值



按兩下儲存格並輸入新值——在完成時按Enter鍵

此顯示視窗以 **16-bit** 的元件為範例



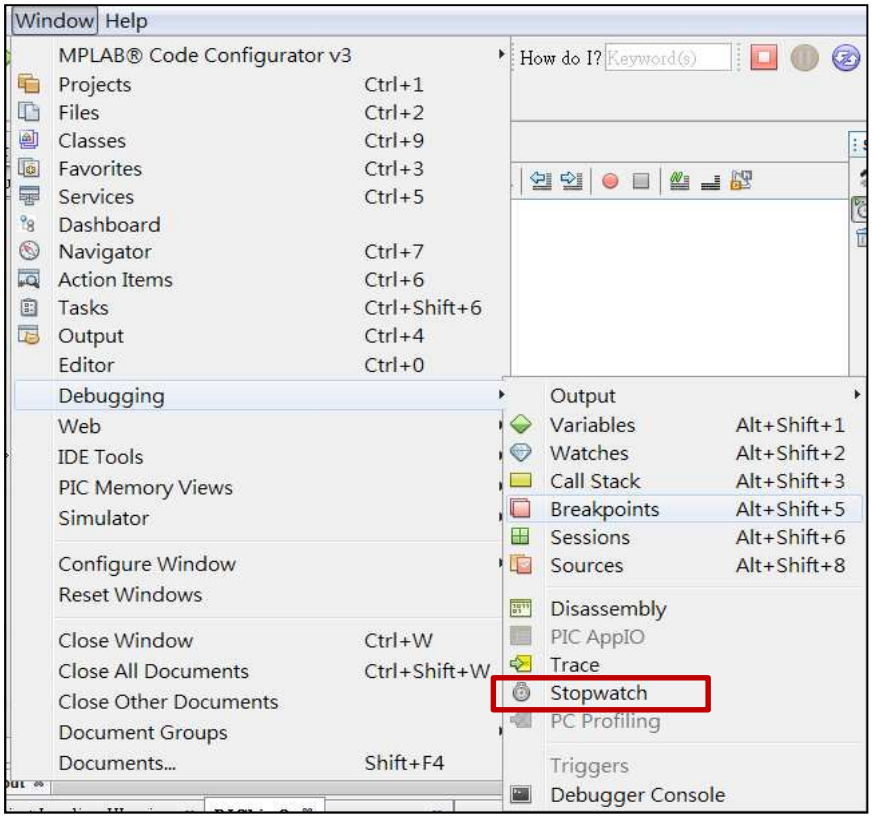
PIC101 Lab1.x

基本除錯

計時碼表功能

計時碼表功能
Stopwatch

- 關於計時碼表功能，主要是量測程式所執行的時間
- 計時碼表功能
  - PICKit3 不支援
  - MPLAB SIM
  - ICD3
  - Real ICE
- 必須在除錯模式下才有此功能



- 進入除錯模式下，**Windows → Debugging → Stopwatch**

量測程式的執行時間

- 使用 **Stopwatch** 的計時碼表功能
- 將沒用到 **I/O** 腳在待測時間的程式段設定轉態變化，程式執行時用示波器來測量時間
- 低速用 **LED** 做即時顯示
- 高速可以用 **UART** 送出，也可即時觀察暫存器的變化

計時碼表功能

Definition

計時碼表功能 **(Stopwatch)** 主要是提供指令周期執行時間的量測，如斷點之間的執行時間，或其它事件的時間量測。確定的程式執行時間可確保程式的穩定度。

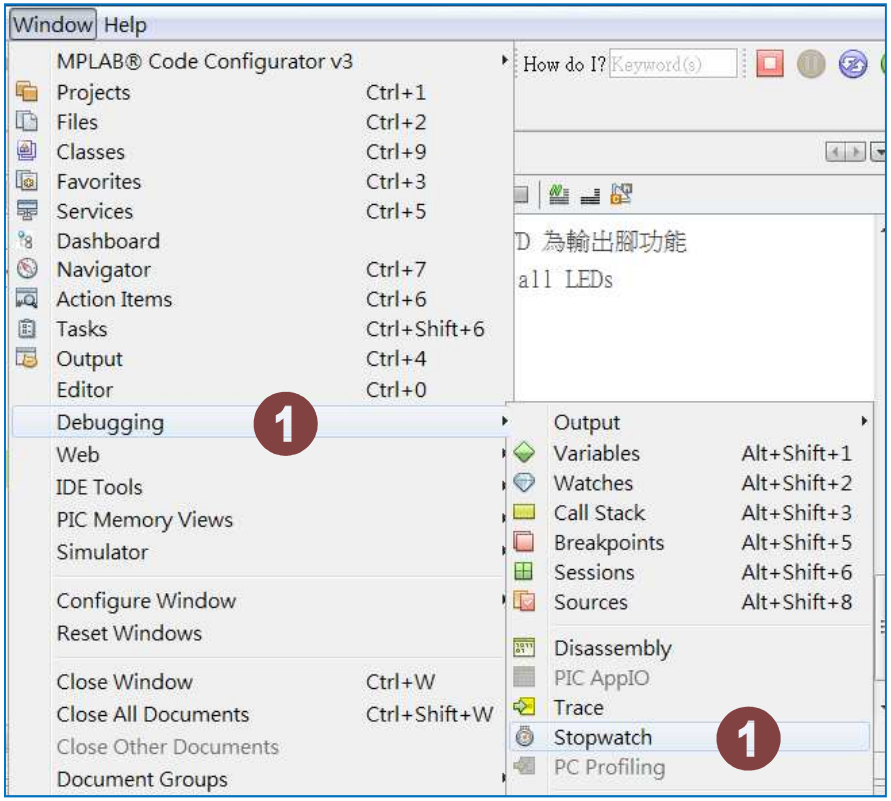
- 使用場合
  - 驗證 延遲迴圈 的時間
  - 確定中斷執行時間
  - 確定 task 完成的時間
  - 確認變數讀寫的時間
  - ….有關程式執行時間的即時量測

使用 Stopwatch

1. 在主目錄：
   Window → Debugging → Stopwatch 開啟計時碼表觀察視窗

2. **Stopwatch** 視窗的顯示
   1. 第一行的 **5 uS** 為 **XC8** 的啟動模組所執行的時間 **(Reset** 到 **main( ))**
   2. 自 **main** 到斷點的時間
   3. 純粹的 **Delay** 迴圈時間

注意: 右邊所顯示的時間或次數，是以 **Instruction Cycle** 為計算主體，且是以 **1MHz** 的指令周期所算出來的。
本練習是使用 **8MHz Fosc** ，所以其時間還要再除二為 **125mS** 才是正確的。

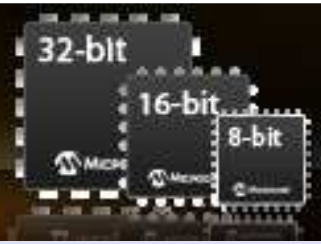


PIC101 Lab1.x

心得與結論

結論

- 您現在應已瞭解建立獨立專案要執行的操作包括：
  - 選擇目標元件的編號
  - 選擇編譯器和除錯工具
  - 選擇專案目錄
  - 編譯後燒錄，並在目標板上或除錯器中執行程式
  - 請繼續接受下一堂課的挑戰



PIC Device Configuration

PIC 的工作配置需求的設定

設定 Configuration Bits

定義

Configuration Bits - 暫存器中的特殊控制位元，配合程式設計的需求。這些設定位元主控制著 **MCU** “永久” 功能的設定，只能在燒錄程式時以燒錄方式設定。

- 設定如下功能：
  - 振盪器類型
  - 看門狗計時器
  - 掉電偵測
  - 程式保護 …
- 更多詳細資訊，請參見資料手冊中的 **“PIC** 的特殊功能暫存器”

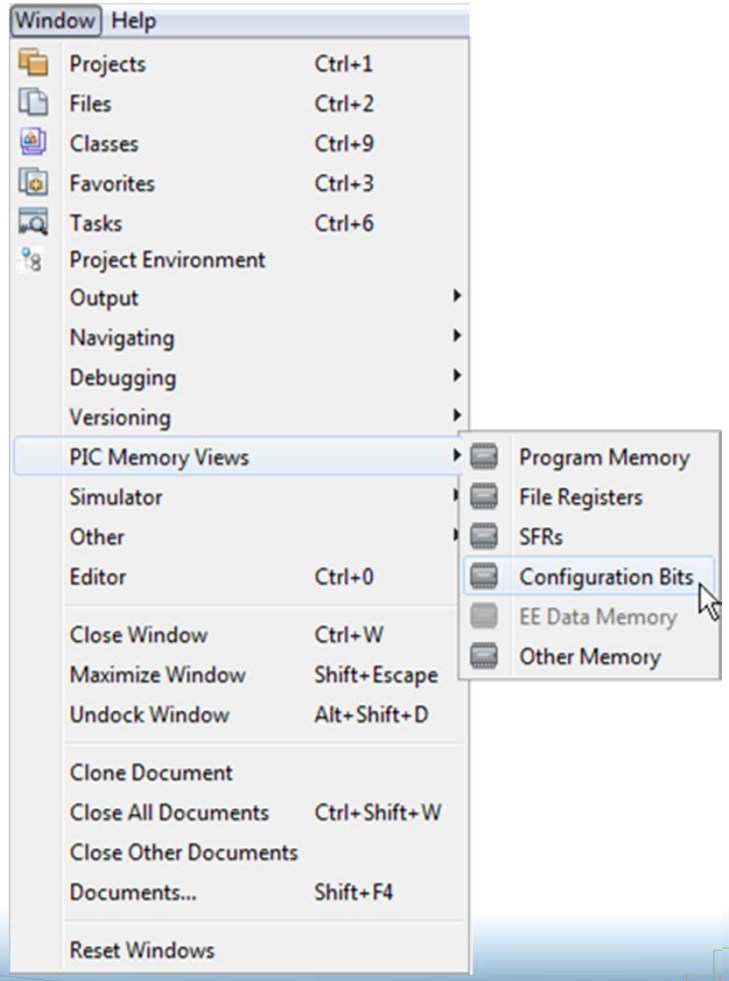
如何查看 Configuration Bits

1 打開 Configuration Bits（設定位元）視窗

從主功能表中選擇：

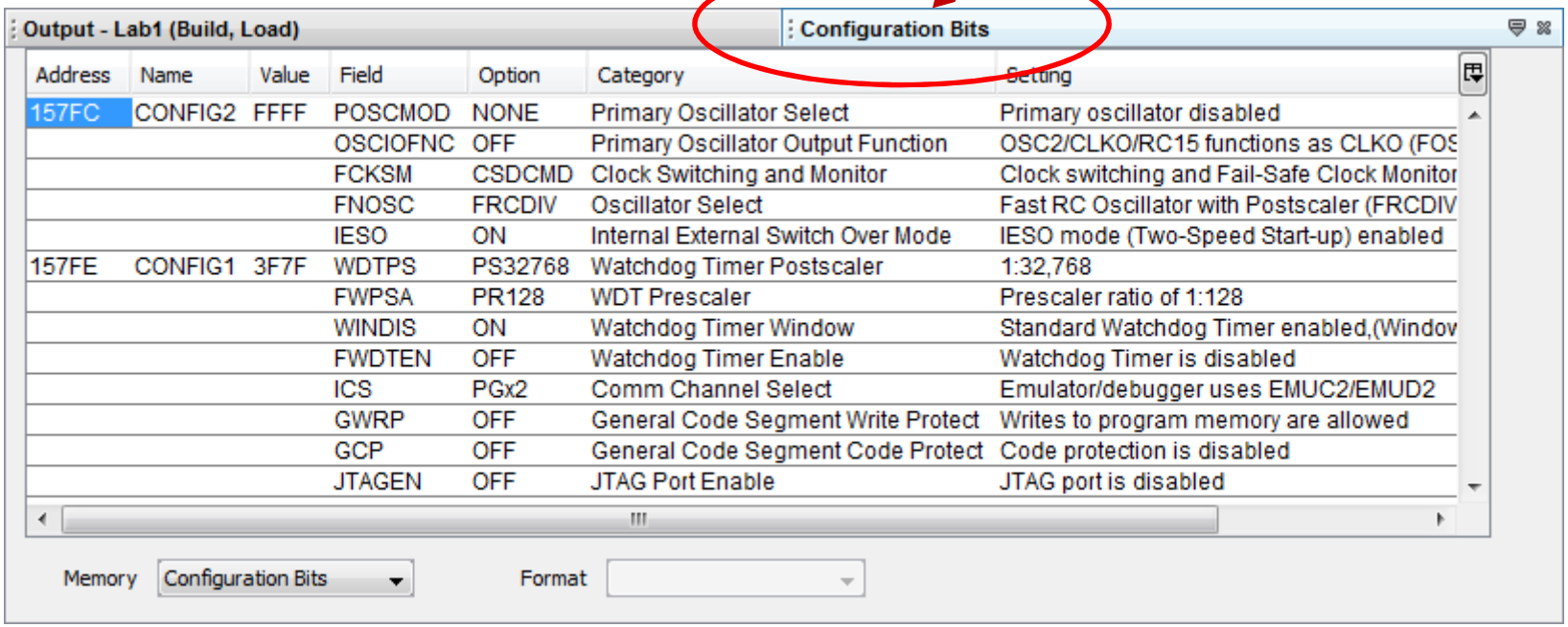
- **Window**
- **PIC Memory Views**

  （PIC 記憶體視圖）
- **Configuration Bits**



如何查看 Configuration Bits

2 該視窗以選項卡的形式在 **Output** 視窗旁邊開啟即可。



MICROCHIP
Regional Training Centers

設定 Config. Bits 的方法

1. 使用 **XC8** 所提供的 **Config. Bits** 的宣告檔案來設定
2. 使用 **X IDE** 的 **Config. Bits** 產生器來產生設定檔
3. 使用 **MCC** 的 **System Module** 的設定，自己產生 **Config.** 的設定 **(請參考 MCC201 v1.0 的教育訓練教材)**

使用 XC8 的 Config. Bits 設定

- **PIC16F** 系列
  - 相關的定義在:
    - **C:\Program Files\Microchip\xc8\v1.2\docs \pic_chipinfo.html**

- **PIC18F** 系列
  - 相關的定義在:
    - **C:\Program Files\Microchip\xc8\v1.12\docs \pic18_chipinfo.html**

Config. Bits 元件的選擇

Microchip MPLAB XC8 C Compiler supported devices				
10F200	10F202	10F204	10F206	10F220
10F222	10F320	10F322	10LF320	10LF322
12C508	12C508A	12C509	12C509A	12C671
12C672	12CE518	12CE519	12CE673	12CE674
12CR509A	12F1501	12F1571	12F1572	12F1822
12F1840	12F508	12F509	12F510	12F519
12F520	12F529T39A	12F529T48A	12F609	12F615
12F617	12F629	12F635	12F675	12F683
12F752	12HV609	12HV615	12HV752	12LF1501
12LF1552	12LF1571	12LF1572	12LF1822	12LF1840
12LF1840T39A	12LF1840T48A	16C432	16C433	16C505
16C54	16C54A	16C54C	16C55	16C554
16C557	16C558	16C55A	16C56	16C56A
16C57	16C57C	16C58A	16C58B	16C620
16C620A	16C621	16C621A	16C622	16C622A
16C62A	16C62B	16C63	16C63A	16C642
16C64A	16C65A	16C65B	16C66	16C662

pic_chipinfo.html

Microchip MPLAB XC8 C Compiler supported devices				
18C242	18C252	18C442	18C452	18C601
18C658	18C801	18C858	18F1220	18F1230
18F1320	18F1330	18F13K22	18F13K50	18F14K22
18F14K22LIN	18F14K50	18F2220	18F2221	18F2320
18F2321	18F2331	18F23K20	18F23K22	18F2410
18F242	18F2420	18F2423	18F2431	18F2439
18F2450	18F2455	18F2458	18F248	18F2480
18F24J10	18F24J11	18F24J50	18F24K20	18F24K22
18F24K50	18F2510	18F2515	18F252	18F2520
18F2523	18F2525	18F2539	18F2550	18F2553
18F258	18F2580	18F2585	18F25J10	18F25J11
18F25J50	18F25K20	18F25K22	18F25K50	18F25K80
18F2610	18F2620	18F2680	18F2682	18F2685
18F26J11	18F26J13	18F26J50	18F26J53	18F26K20
18F26K22	18F26K80	18F27J13	18F27J53	18F4220
18F4221	18F4320	18F4321	18F4331	18F43K20
18F43K22	18F4410	18F442	18F4420	18F4423

pic18_chipinfo.html

pic18_chipinfo.html 裡的 PIC18F4520 Config.

使用範例: **#pragma config IESO=OFF, OSC=INTIO67, FCMEN = OFF ….**

Register: CONFIG1H @ 0x300001

IESO =	**Internal/External Oscillator Switchover bit**	
	OFF	**Oscillator Switchover mode disabled**
	ON	**Oscillator Switchover mode enabled**
OSC =	**Oscillator Selection bits**	
	RCIO6	**External RC oscillator, port function on RA6**
	RC	**External RC oscillator, CLKO function on RA6**
	INTIO7	**Internal oscillator block, CLKO function on RA6, port function on RA7**
	XT	**XT oscillator**
	LP	**LP oscillator**
	HSPLL	**HS oscillator, PLL enabled (Clock Frequency = 4 x FOSC1)**
	ECIO6	**EC oscillator, port function on RA6**
	EC	**EC oscillator, CLKO function on RA6**
	INTIO67	**Internal oscillator block, port function on RA6 and RA7**
	HS	**HS oscillator**
FCMEN =	**Fail-Safe Clock Monitor Enable bit**	
	OFF	**Fail-Safe Clock Monitor disabled**
	ON	**Fail-Safe Clock Monitor enabled**

其它 Config. Bits 的設定（一）

- **CONFIG2L**
  - BOREN = SBORDIS, BORV = 2, PWRT = ON
- **CONFIG2H**
  - WDT = OFF **(**除錯時的必要選項）
- **CONFIG3H**
  - CCP2MX = PORTC, PBADEN = OFF, MCLRE = ON **(**除錯模式下需使用 **MCLR** 腳**)**
- **CONFIG4L**
  - LVP = OFF, STVREN = ON, XINTST = OFF
    - 如使用 **Lite** 版的 **XC8** 要將 **XINTST** 設成 **OFF**

其它 Config. Bits 的設定 (二)

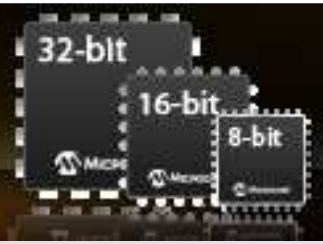
- **CONFIG5L (設成OFF)**
  - 區塊程式碼保護設定，除錯階段關閉所有的保護設定
- **CONFIG5H (設成OFF)**
  - Boot 區塊保護，EEROM 資料保護
- **CONFIG6L (設成OFF)**
  - 程式區塊禁止程式執行的寫入
- **CONFIG6H (設成OFF)**
  - Config. Reg, EEPROM, Boot 區塊寫入的保護

除錯階段請關閉所有的保護設定

PIC101 Lab1.x

程式裡的 Config. Bits 設定

```
#include <xc.h>      // XC8 萬用型標頭檔

// Lab1 的 Config. Bits 的設定
#pragma config PLLCFG = OFF, FOSC=INTIO67,
BOREN = SBORDIS, BORV = 190, PWRTEN = OFF,
WDTEN=OFF, PBADEN=OFF,  LVP = OFF, XINST = OFF,
MCLRE = EXTMCLR

//內部 EEPROM 的初始設定值
__EEPROM_DATA
(0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07);
__EEPROM_DATA
(0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F);
```



PIC101 Lab2.x

使用 X IDE
設定 Config. Bits 的方式

PIC101 Lab2.x

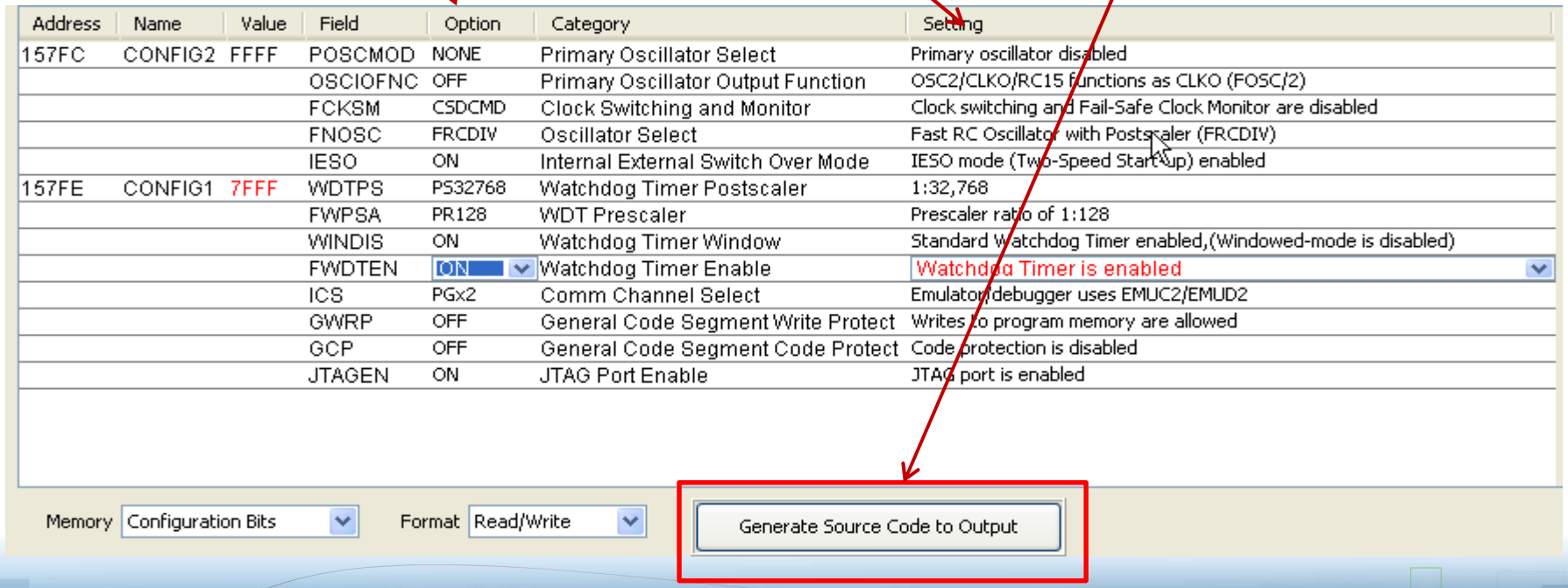
第二種方式產生 Config. Bits

- 利用 **X IDE** 來產生
  - 在主目錄的 Windows → PIC Memory Views → 開啟 Configuration Bits 的視窗
  - 修改所需的 Config. Bits 設定

PIC101 Lab2.x

手動修改產生 Config. Bits

- 在 **MPLAB® X IDE** 下手動修改設定
  - 在 Option 或 Setting 下，點選要更改的設定
  - 在下拉式功能表中選擇值，全都完成後
  - 按一下按鈕 “Generate Source Code to Output”
    （產生設定 Config. 的 C 原始程式）



PIC101 Lab2.x

加入 Conf. Bits 到專案裡

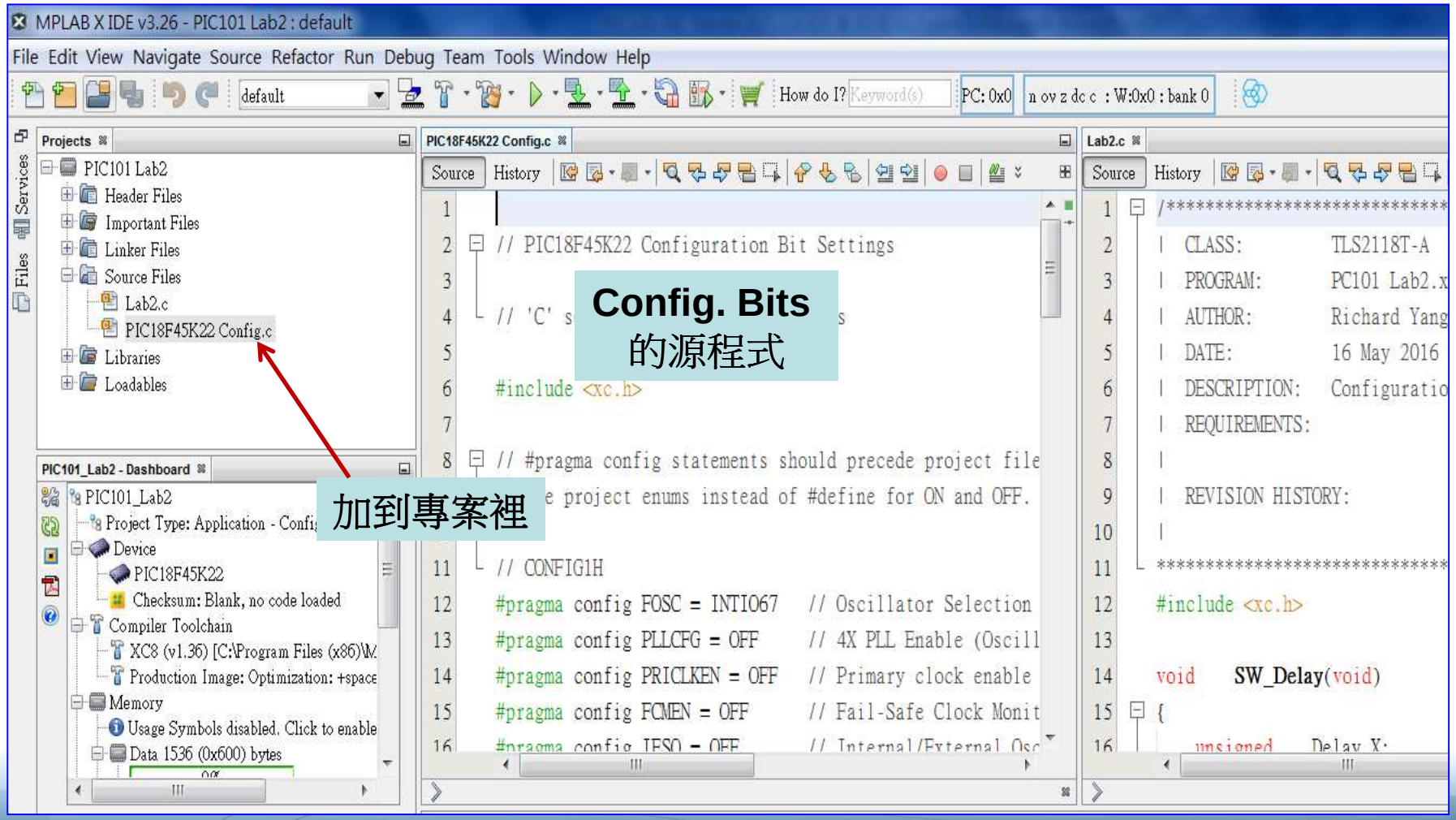
- 這時到 “Output” 視窗將可看見所產生 Config. 原始程式
- 在編輯區域，按老鼠右鍵用”Save As” 存檔到專案目錄下，檔案名稱為 PIC18F45K22 Config.c
- 將此 C 檔加到專案中並編譯。

Configuration Loading Error | Config Bits Source | PIC101 Lab2 (Clean, Build, ...)

```
// PIC18F45K22 Configuration Bit Settings

// 'C' source line config statements

#include <xc.h>

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

// CONFIG1H
#pragma config FOSC = ECHPIO6   // Oscillator Selection bits (EC oscillator (high power, >16 MHz))
#pragma config PLLCFG = OFF     // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON    // Primary clock enable bit (Primary clock is always enabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)
```

PIC101 Lab2.x

- 按  圖示做編譯及燒錄，程式是否一樣



PIC101 Lab2.x 練習的延伸

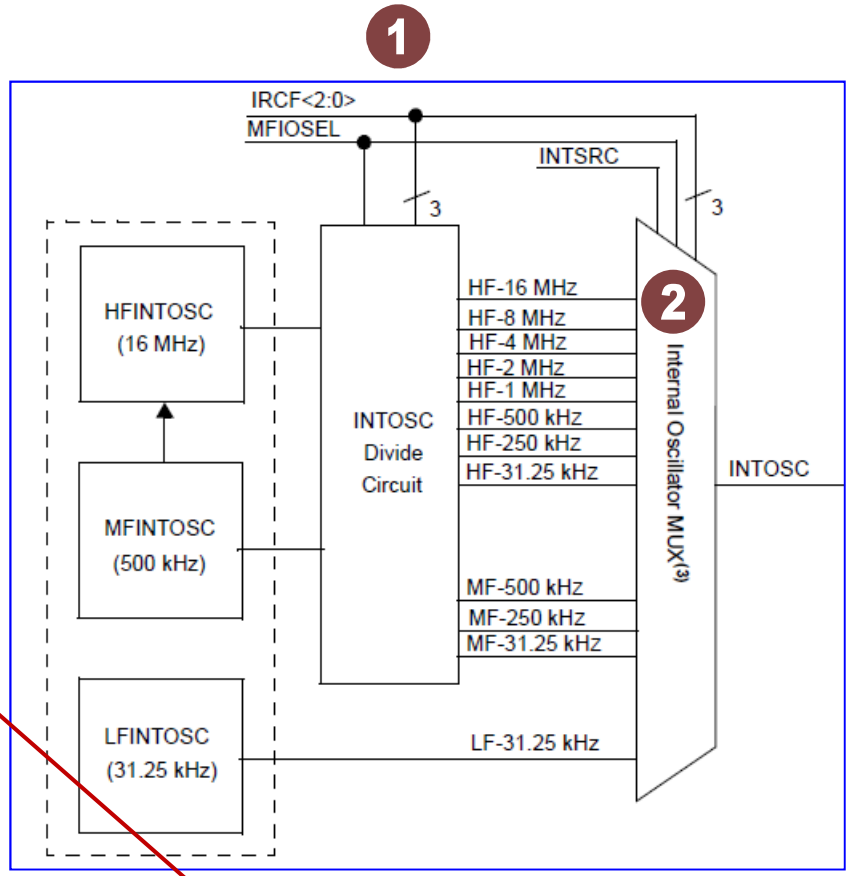
- 目前程式是以 **8MHz (Fosc)** 的速度在執行 **LED** 的加一動作
  - 想以 1MHz 的速度來執行?
  - 以最高速 64MHz 的速度來執行?
  - 以上要怎樣做才能實現?
- 可以使用 **MCC** 設定嗎?

選擇內部 RC 的輸出

1. 內部 **RC** 震盪器
   透過 **IRCF<2:0>** 選擇輸出頻率

2. 開機內定輸出為 **1MHz**

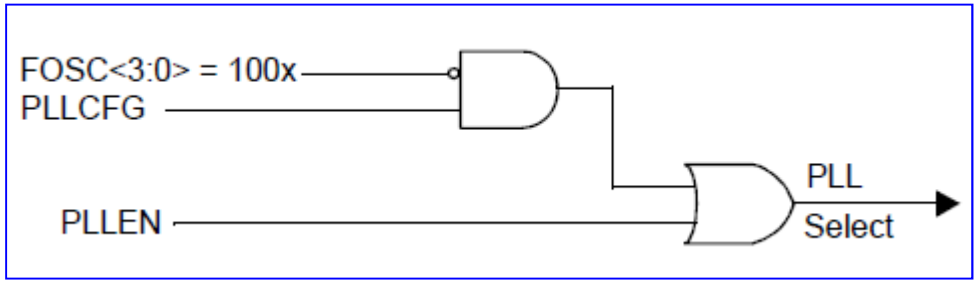
IRCF<2:0>: 內部 **RC** 頻率選擇位元
111 = HFINTOSC – (16 MHz)
110 = HFINTOSC/2 – (8 MHz)
101 = HFINTOSC/4 – (4 MHz)
100 = HFINTOSC/8 – (2 MHz)
011 = HFINTOSC/16 – (1 MHz)
010 = HFINTOSC/32 – (500 kHz)
001 = HFINTOSC/64 – (250 kHz)
000 = LFINTOSC – (31.25 kHz)




```
void main(void)
{
    OSCCONbits.IRCF = 0x06 ;    // 調整 IRCF 位元來設定工作頻率
                                // Select INTOSC-8Mhz as clock source
```

內建 RC + x4 PLL

- 欲使用 **64MHz** 的 **Fosc**



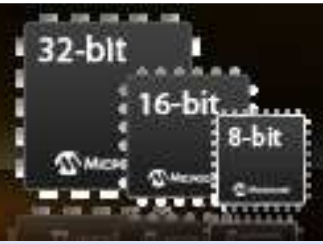
PLLCFG 在 **Config. Bits** 燒錄時設定
只要將該為設定為一後，
就可以啟動外部震盪器使用 **x4 PLL**
這時 **PLLEN** 軟體控制無效

OSCTUN<6> : PLLEN 為軟體設定 **PLL** 啟動位元，只要將該為設定為一後，
不管使用外部或內部震盪
x4 PLL 立即啟動

PLLCFG: 4 x PLL Enable bit
1 = 4 x PLL always enabled, Oscillator multiplied by 4
0 = 4 x PLL is under software control, PLLEN (OSCTUNE<6>)

PLLEN: Frequency Multiplier 4xPLL for HFINTOSC Enable bit
1 = PLL enabled for HFINTOSC (8 MHz and 16 MHz only)
0 = PLL disabled

- 所需的設定為：選用內部 **16MHz**，再設定 **PLLEN** 位元 **= 1**，這樣就可以的到 **64MHz**的執行速度。



MCC

基本的 Config. Bits 設定

請事先安裝 **MCC v.xx** 版本，安裝方法請參考 **X IDE** 安裝附件

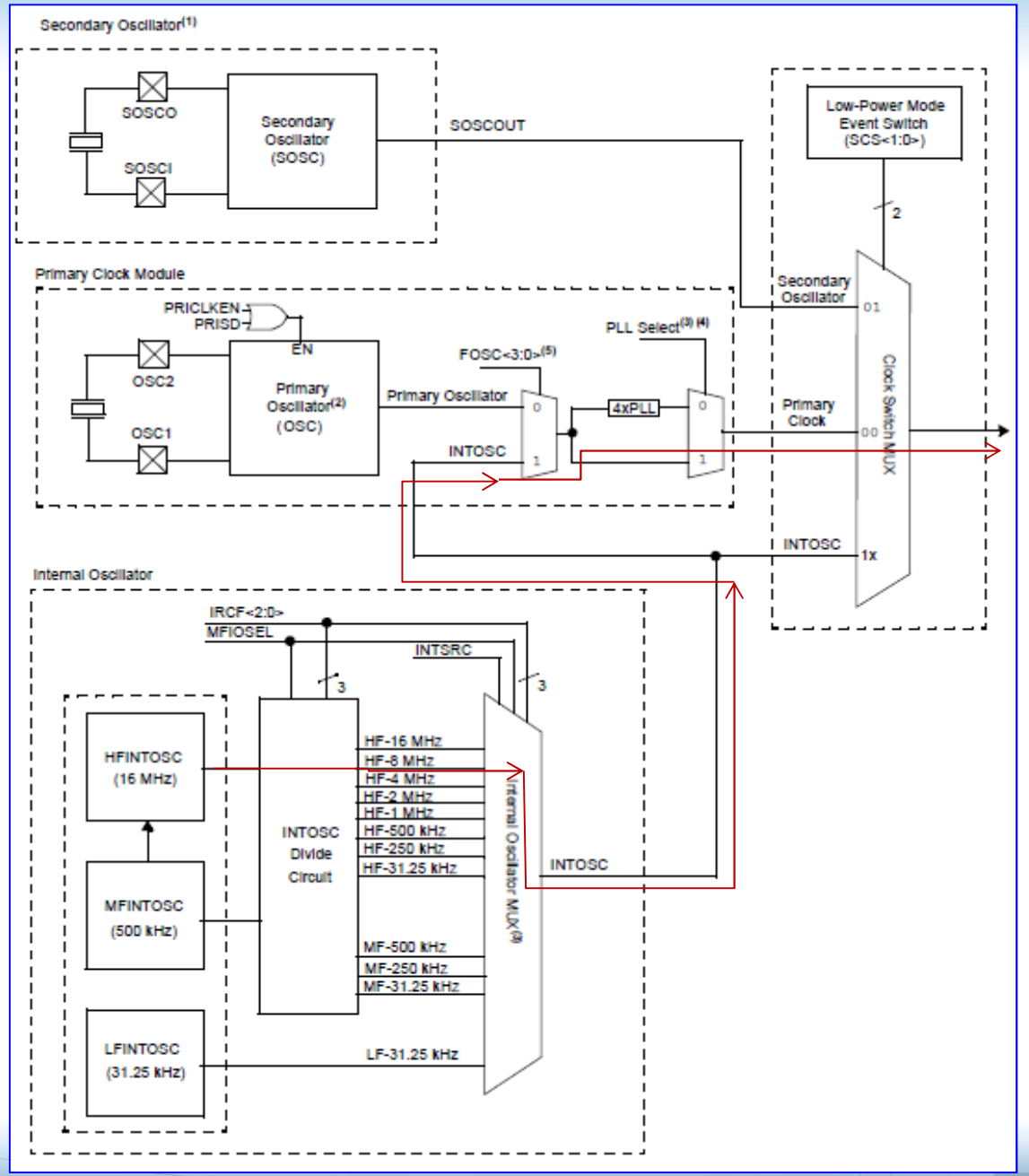
第 **32**頁：安裝 **MCC** 外掛模組的方法

使用 MCC 的基本配置
Config. Bits

- 這裡將繼續延用 **PIC101 Lab2.x** 的 **125mS** 的軟體計時方式 **Fosc @8MHz**
- 使用 **MCC v3.x** 來設定 **Config. Bits**
- 專案名稱: **PIC101 Lab2-1 MCC.x**
  - 先設定執行速度為 8MHz (內部RC)
  - 使用 MCC 設定 x4 PLL 設定 32MHz
  - 內建 RC 如何執行 64MHz 的執行速度
  - 16MHz 石英震盪設定 Fosc=64MHz

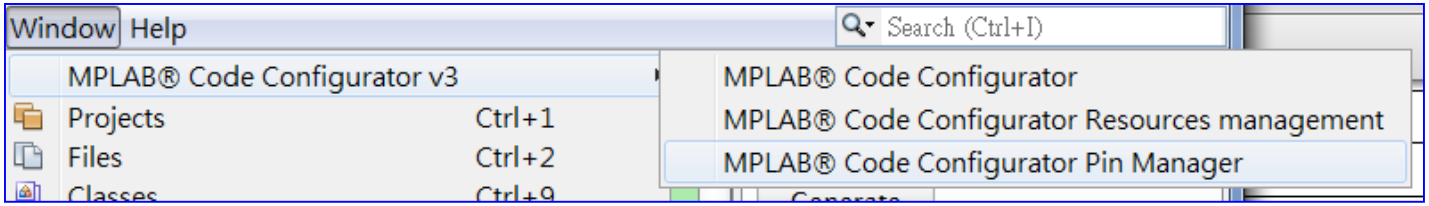
MICROCHIP
Regional Training Centers

- **PIC18F45K22**
  震盪器方塊圖
  - **FOSC**
  - **HF- 8MHz**

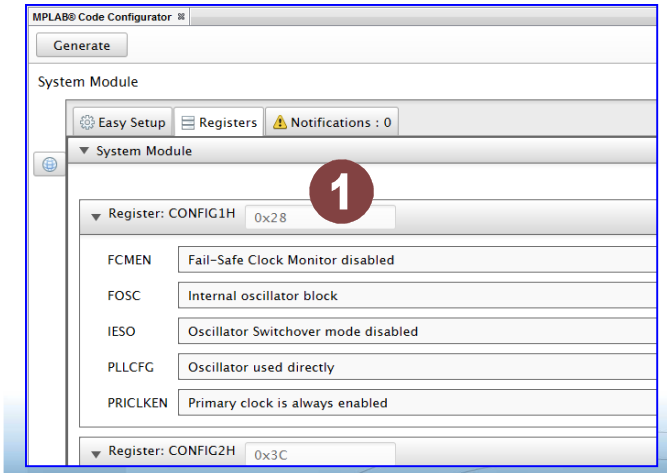


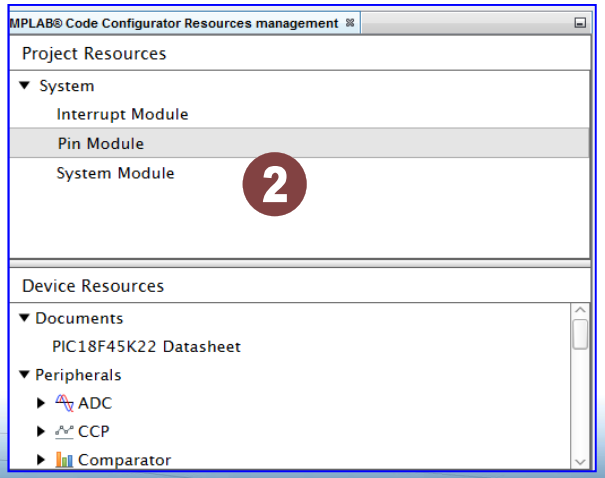
MCC 視窗

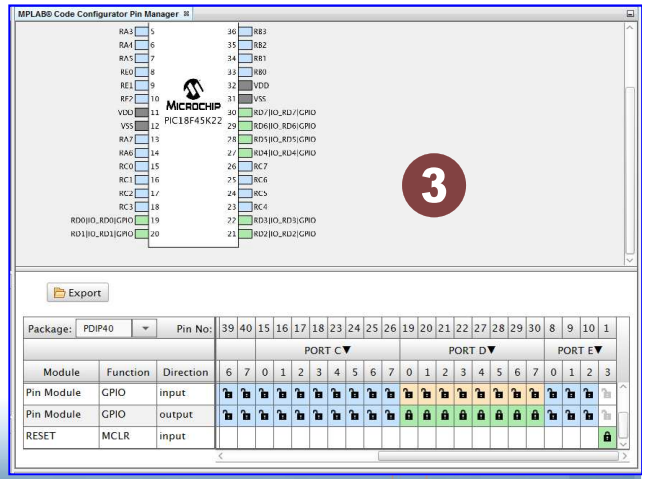
- **MCC 主要有三個設定視窗**



1. Code Configurator (主視窗)
2. Resource Management (周邊資源選項視窗)
3. Pin Manager (腳位管理視窗)

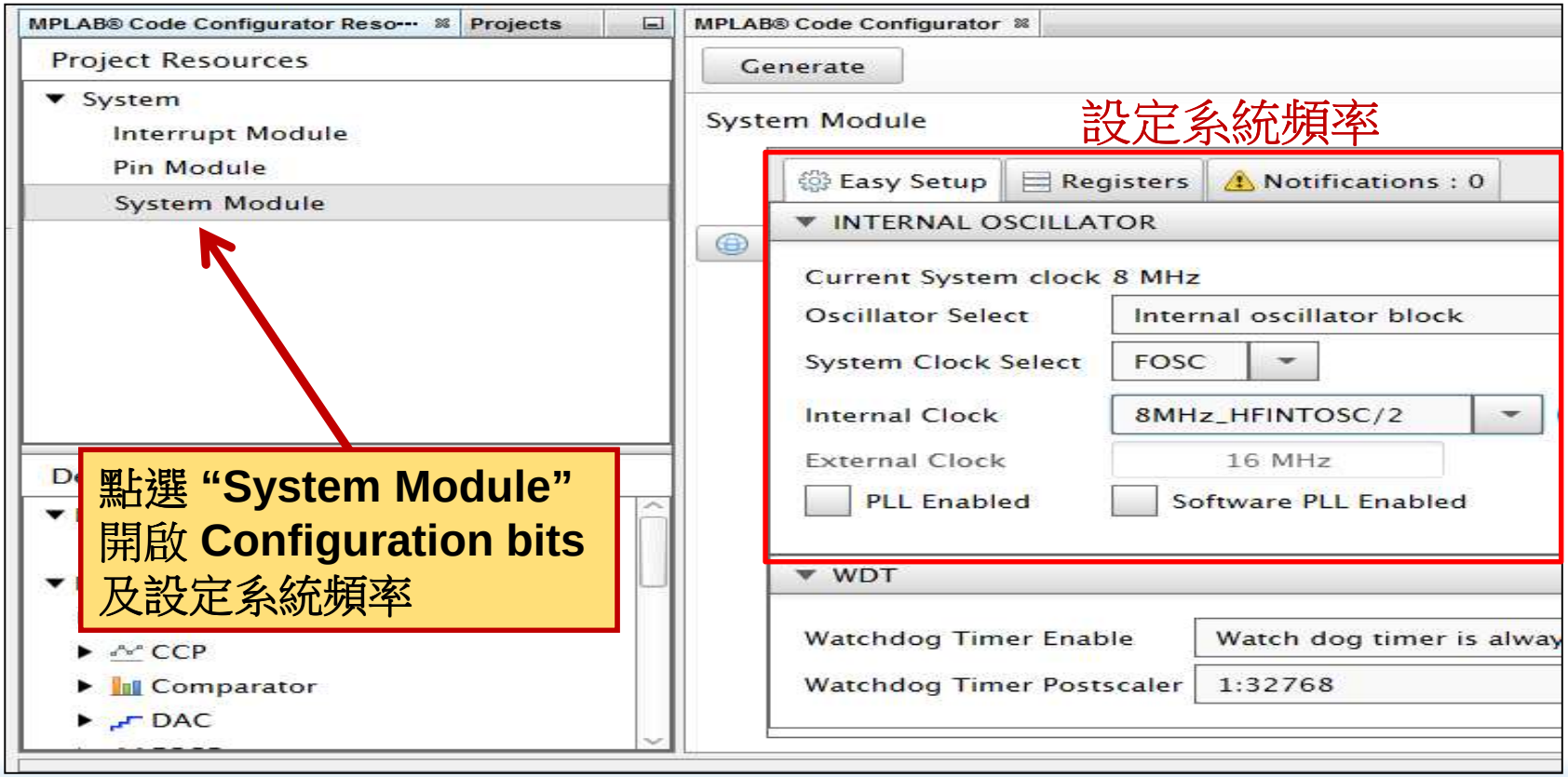






1. Config. Bits 的設定

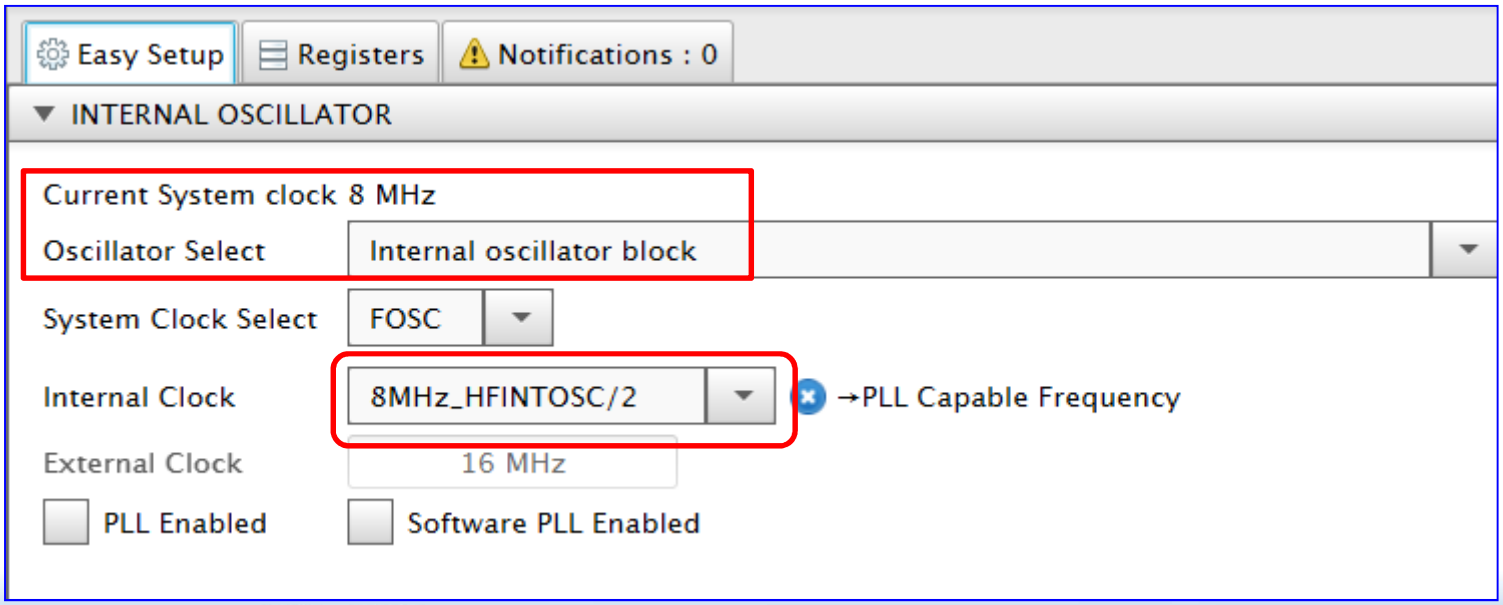
- 點選 “ **System Module**” 開始設定 **Configuration bits** 及 **Fosc = 8MHz**



PIC101 Lab2-1 MCC.x

Fosc @8MHz

- **System Clock** 選擇內定的
  - FOSC = 8MHz
- **Internal Clock** 選擇內定的
  - 8MHz_HFINTOS/2

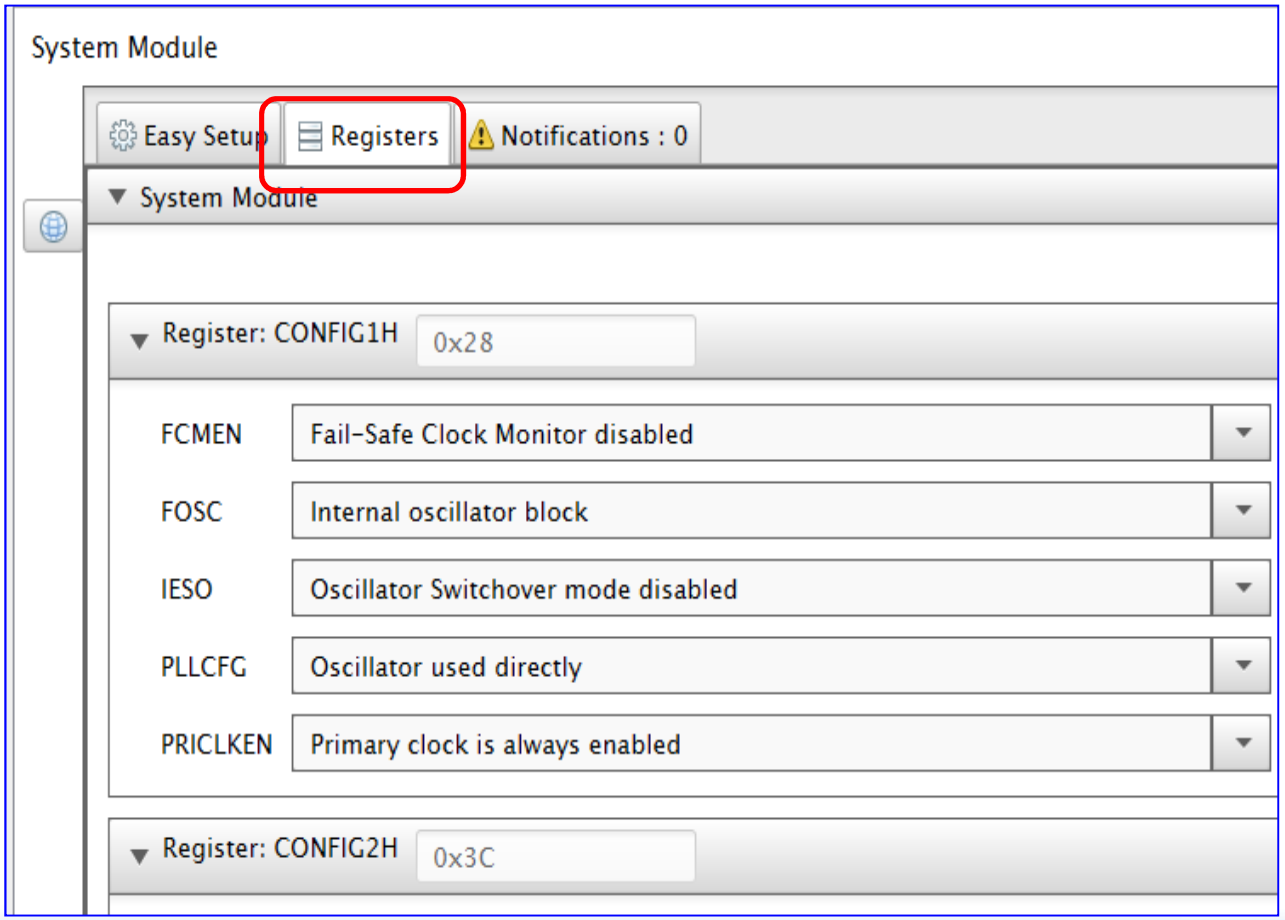


震盪器的設定 (COFIG1H)

- **CONFIG1H** 使用內定的震盪器

設定值選項：

- **FCMEN OFF**
- **FOSC INTIO67**
- **IESO OFF**
- **PLLCFG OFF**
- **PRICLKEN OFF**



CONFIG1H (燒錄決定)

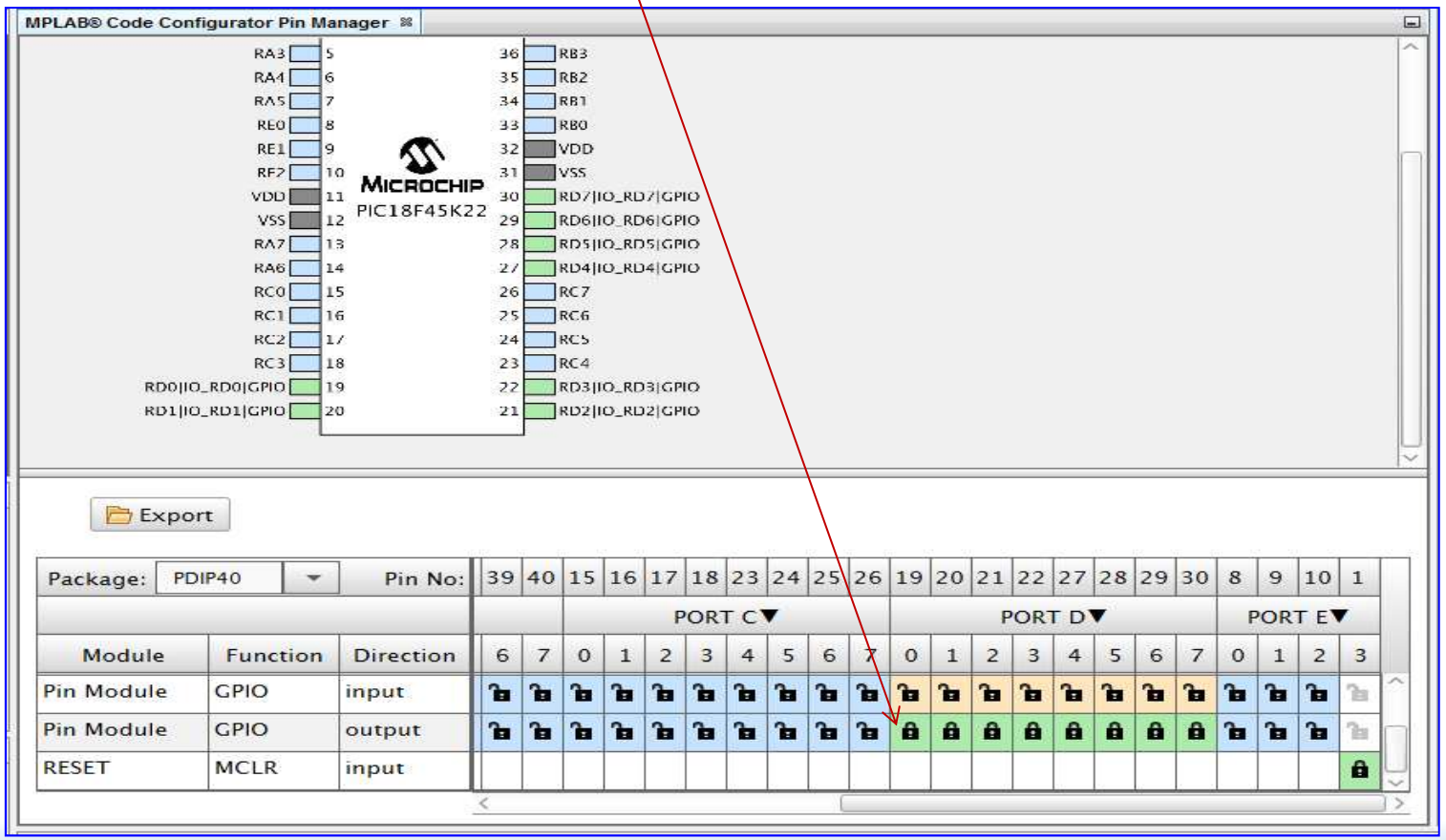
震盪模式的選擇

FOSC<3:0>: Oscillator Selection bits

- 1111 = External RC oscillator, CLKOUT function on RA6
- 1110 = External RC oscillator, CLKOUT function on RA6
- 1101 = EC oscillator **(low power, <500 kHz)**
- 1100 = EC oscillator, CLKOUT function on OSC2 **(low power, <500 kHz)**
- 1011 = EC oscillator **(medium power, 500 kHz-16 MHz)**
- 1010 = EC oscillator, CLKOUT function on OSC2 **(medium power, 500 kHz-16 MHz)**
- 1001 = Internal oscillator block, CLKOUT function on OSC2
- 1000 = Internal oscillator block
- 0111 = External RC oscillator
- 0110 = External RC oscillator, CLKOUT function on OSC2
- 0101 = EC oscillator **(high power, >16 MHz)**
- 0100 = EC oscillator, CLKOUT function on OSC2 **(high power, >16 MHz)**
- 0011= HS oscillator **(medium power, 4 MHz-16 MHz)**
- 0010= HS oscillator **(high power, >16 MHz)**
- 0001= XT oscillator
- 0000= LP oscillator

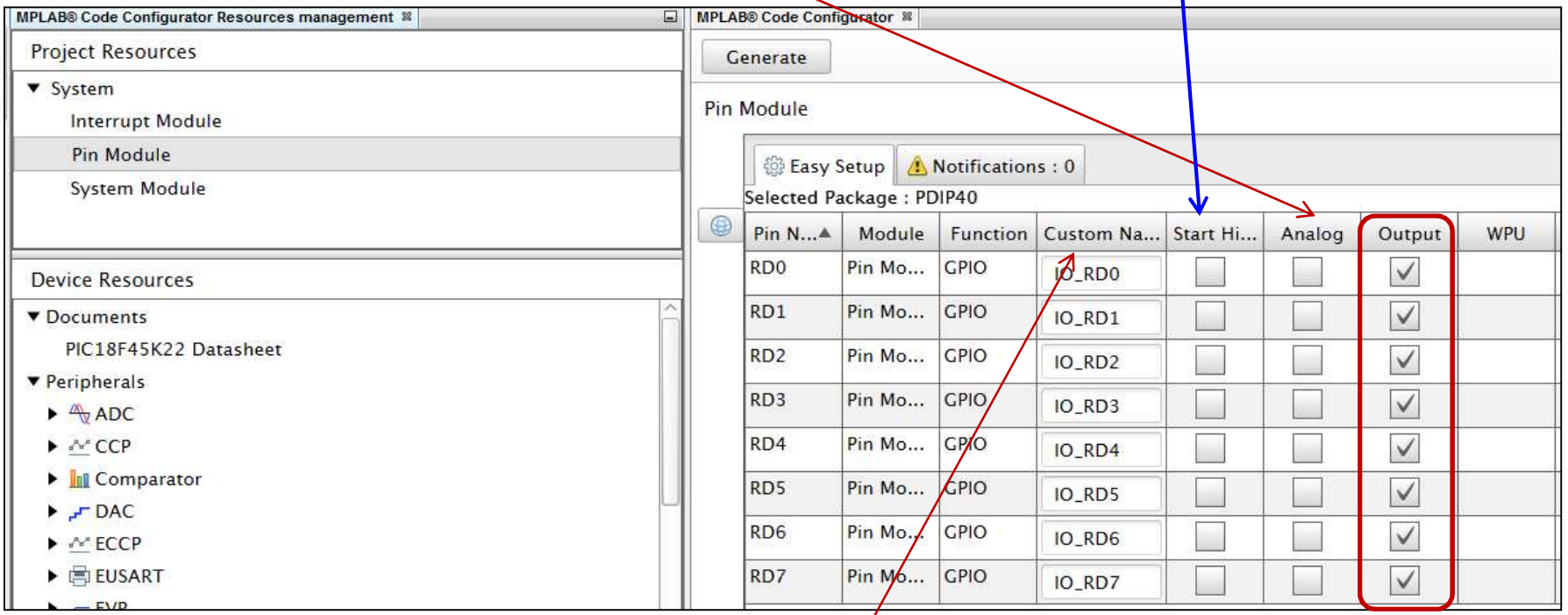
2. 設定 PORTD 為輸出

- 開啟 **Pin Manager** 視窗，將 **PORTD** 的輸出欄上的圖示標示成上鎖 (綠色)



3. 設定 PORTD 的腳位功能

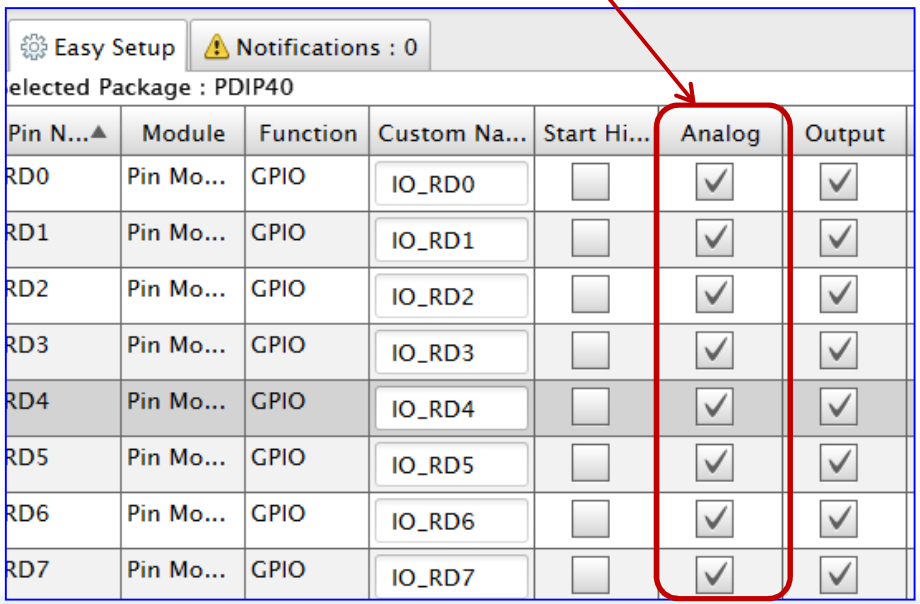
- 切換回 **MCC** 主視窗，將 **PORTD** 初始設定為 **0**，類比輸入功能關閉



這裡先不用客製化腳位名稱的設定，暫時使用內定腳位名稱

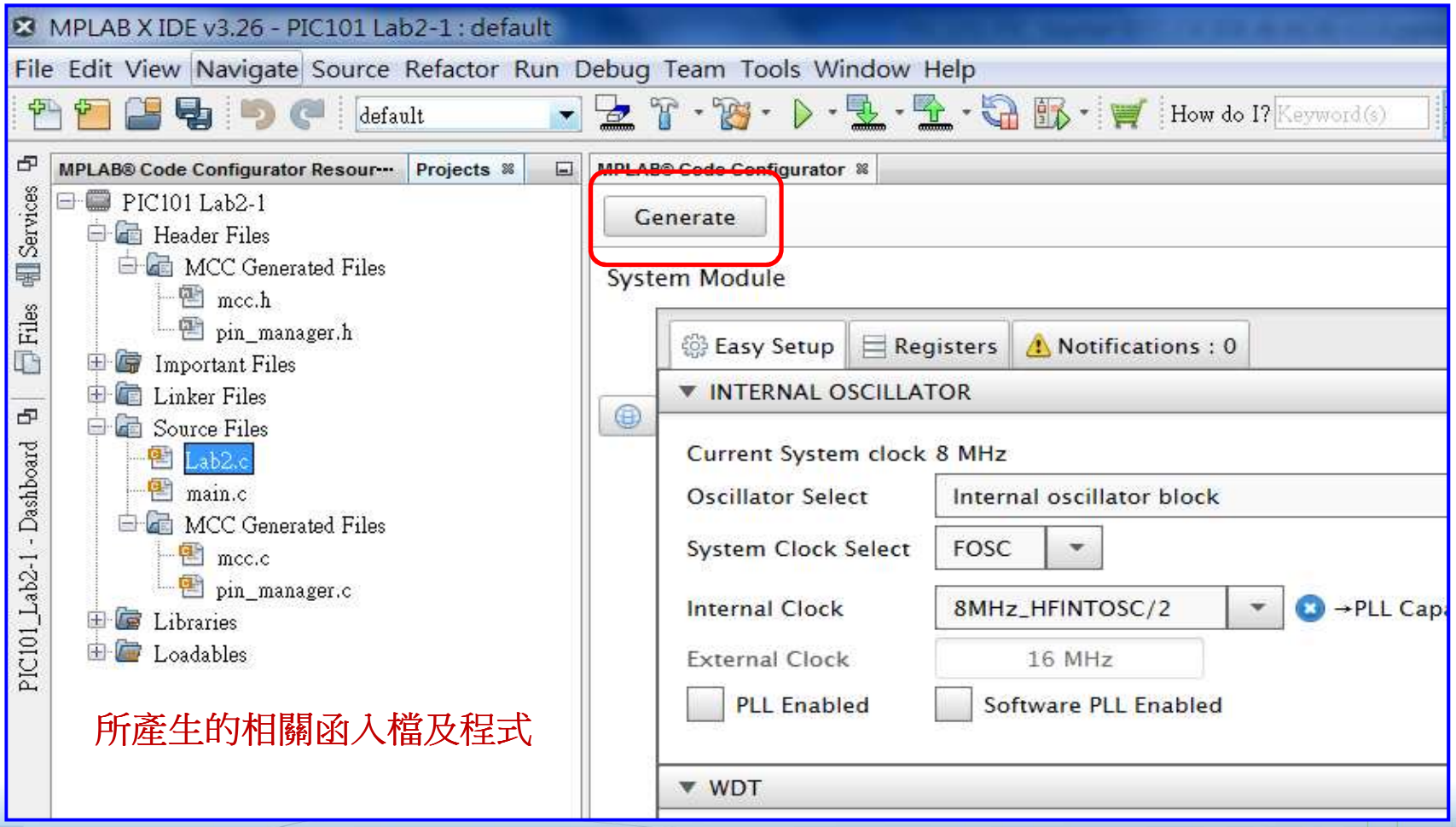
關於腳位的設定問題

- 很重要的提示:
  - 要是腳位功能有再被修改或是再加入新的腳位設定時，在腳位功能設定視窗下發現所有的腳位的類比輸入功能都會被重新勾選。
  - 所以在按 Code Generate 前一定要再檢查一下腳位功能視窗的設定



4. MCC 程式產生

- 按下 **Generate** 方塊產生周邊函數

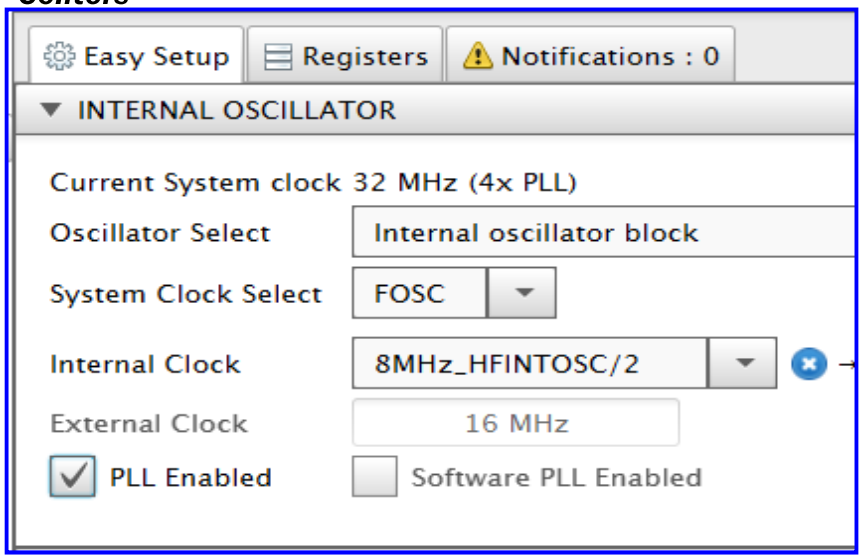


執行 PIC101 Lab2-1 MCC.x

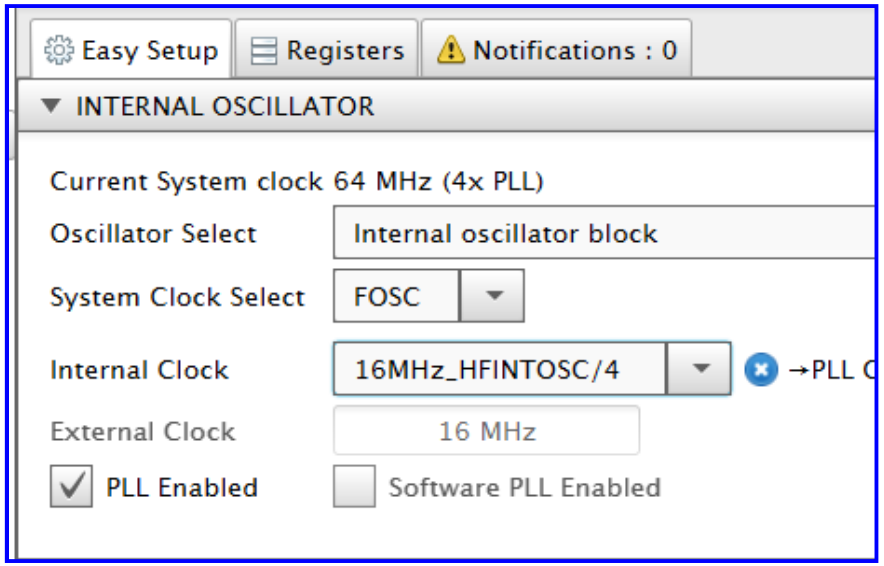
- 直接編譯並燒錄到 **PIC18F45K22** 去執行，**PORTD** 的 **LED** 進位速度是否為 **Fosc = 8MHz** 的速度

- 以示波器量測 **LED0 (LTAD0)** 的輸出為 **125mS** 轉態一次

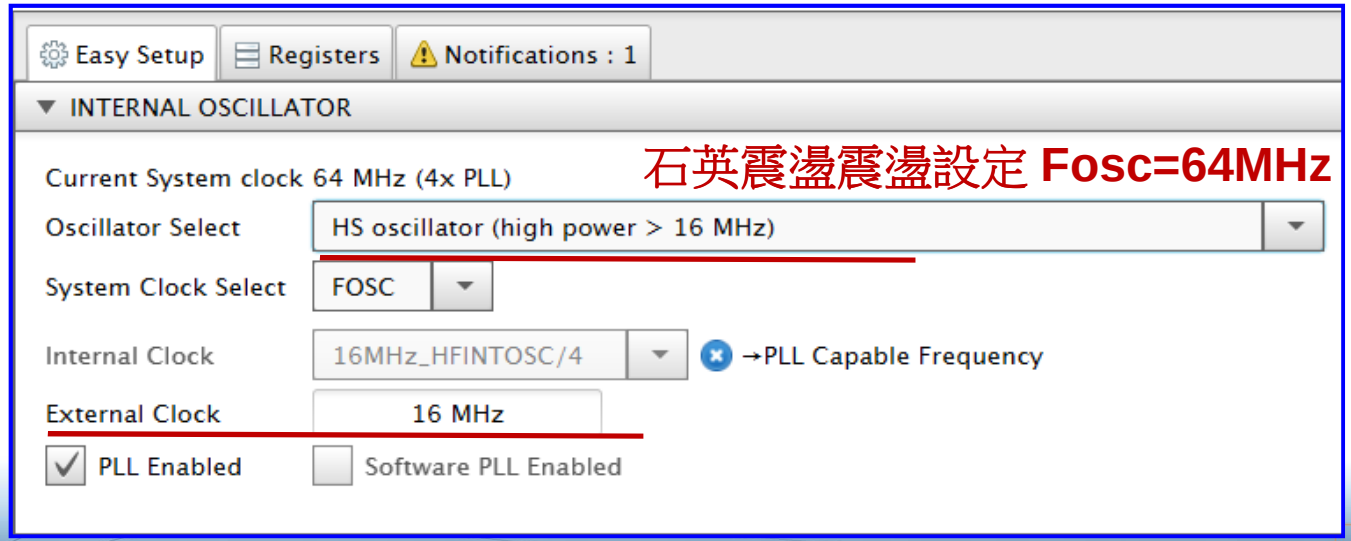
MCC 的修改的方便性



內建 **RC** 震盪設定 **Fosc=64MHz**



內建 **RC** 震盪震盪設定 **Fosc=64MHz**



其它 Config. Bits 的設定（一）

- **CONFIG2L**
  - BOREN SBORDIS, BORV 285, PWRTEN ON
- **CONFIG2H**
  - WDTEN OFF **(**除錯時的必要選項）
- **CONFIG3H**
  - CCP2MX PORTC1, PBADEN OFF,
    MCLRE EXTMCLR **(**除錯模式下需使用 **MCLR** 腳**)**
- **CONFIG4L**
  - LVP OFF, STVREN ON, XINTST OFF
    - 如使用 **Lite** 版的 **XC8** 要將 **XINTST** 設成 **OFF**

其它 Config. Bits 的設定 (二)

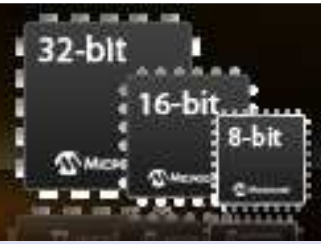
- **CONFIG5L (設成OFF)**
  - 區塊程式碼保護設定，除錯階段關閉所有的保護設定
- **CONFIG5H (設成OFF)**
  - Boot 區塊保護，EEROM 資料保護
- **CONFIG6L (設成OFF)**
  - 程式區塊禁止程式執行的寫入
- **CONFIG6H (設成OFF)**
  - Config. Reg, EEPROM, Boot 區塊寫入的保護

除錯階段請關閉所有的保護設定

其它 Config. Bits 的設定 (三)

- **CONFIG7L (設成OFF)**
  - 程式碼區塊禁止 Table Read 指令讀取的保護
- **CONFIG7H (設成OFF)**
  - Boot 區塊禁止 Table Read 指令讀取的保護

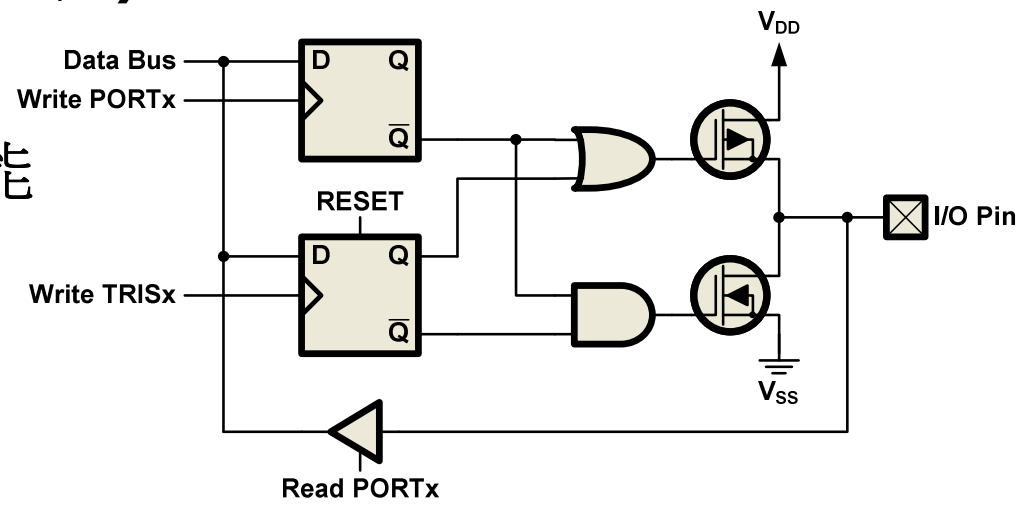
除錯階段如請關閉所有的保護設定



PIC18 Family 內建的基礎周邊說明 - I/O

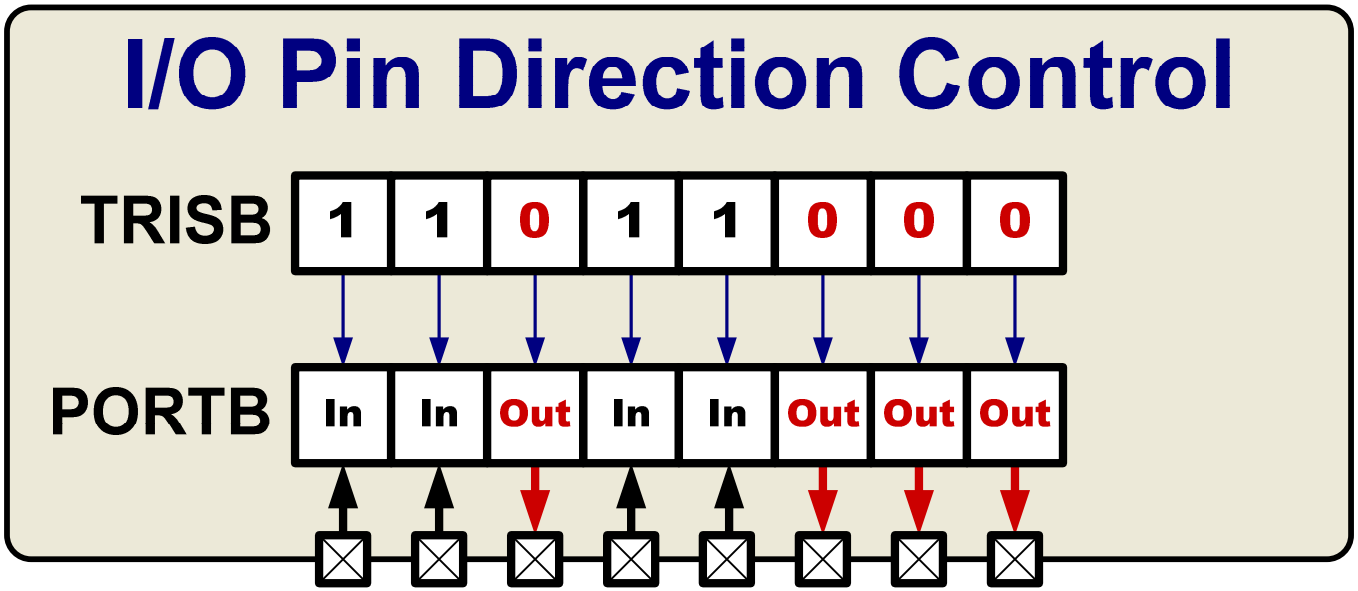
I/O Ports

- 高驅動能力 **(25mA)**
- 可直接驅動 **LEDs (需串電阻)**
- 每支腳位都可以個別使用軟體來設定輸出入功能
- 所有的腳位都具有 **ESD** 二極體保護功能



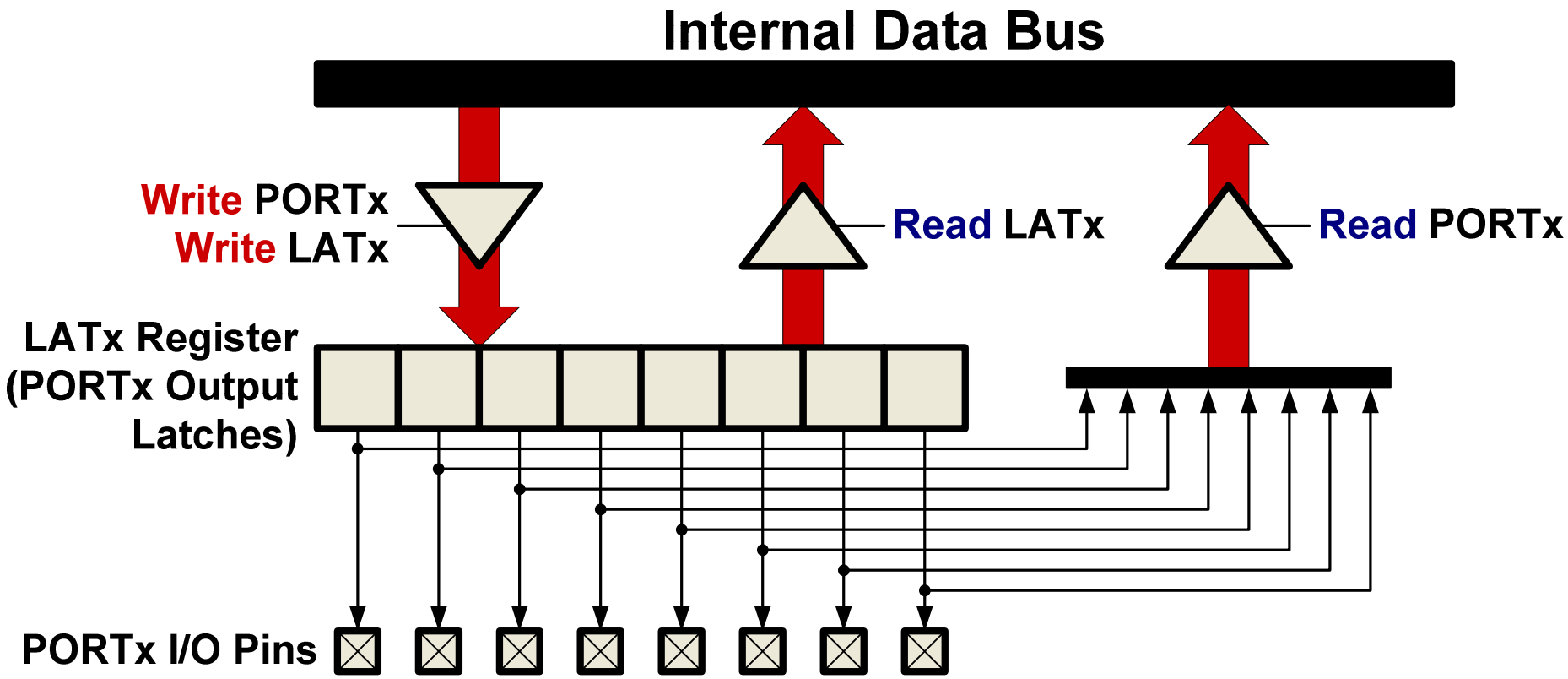
- 開機時，所有的 **I/O** 腳內定均設為輸入 **(**高輸入阻抗**)**，所以要注意沒有被使用到的 **I/O** 腳浮接狀態
- 注意：如果該 **I/O** 腳位有類比功能 **(**如 **AD** 輸入**)**，開機時其內定功能為類比的輸入功能

I/O Ports



- 位於 **TRISx** 暫存器中的 **Bit n** 決定在 **PORTx** 中的第 **n** 個腳位的方向
- **1 = Input, 0 = Output**

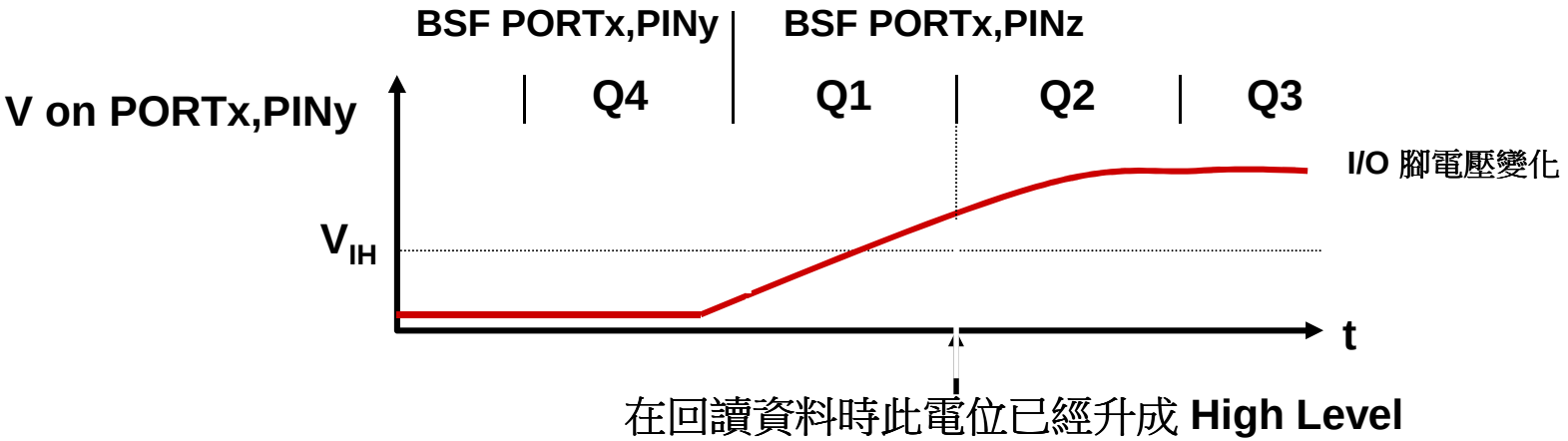
Digital I/O Ports – PIC18



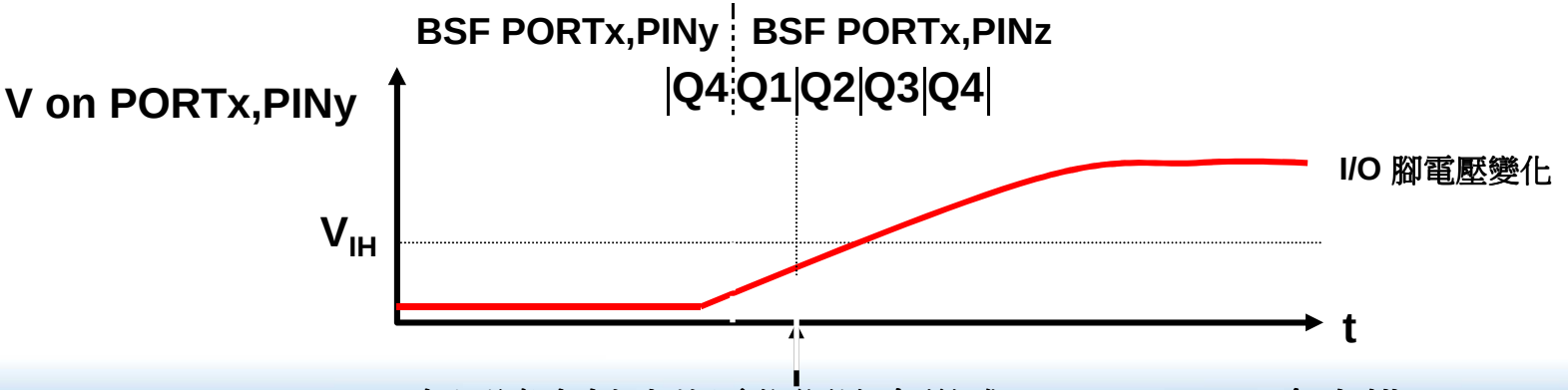
注意一下：**PORTx** 與 **LATx** 在做輸出與輸入時，功能上有何不同？

為何要加入 LATx 的暫存器

消除 Read-Modify-Write 動作的風險



在回讀資料時此電位已經升成 **High Level**



在回讀資料時此電位還沒有變成 **High Level**，會出錯!

Analog 或 Digital I/O?

- 有些 **I/O** 腳位的功能與類比輸入是多工使用的 **(Power On** 後的預設值為 **“analog mode”)**
  - 所以在初始化程式段落並須將欲使用為 Digital I/O 的腳位規劃為 Digital mode !
- 在 **PIC18F45K22** ，使用新的設定方式
  - 所有的 A/D 輸入都會有相對應的設定暫存器
  - ANSELx – x 為 PORTx
  - 該位元開機內定為 1，啟用該腳位的 A/D 輸入功能

REGISTER 10-3: ANSELA – PORTA ANALOG SELECT REGISTER

U-0	U-0	R/W-1	U-0	R/W-1	R/W-1	R/W-1	R/W-1
—	—	ANSA5	—	ANSA3	ANSA2	ANSA1	ANSA0
bit 7							bit 0

如何使用 I/O 腳位

- 所有的周邊，**PORTx** 及位元的定義在:
  - C:\Program Files (x86)\Microchip\xc8\v1.36\include\pic18f45k22.h
  - 使用時只需要加入 #include <xc.h> 即可

- 範例: **PORTA** 的操作

```
ANSELA = 0x00;    // 關閉 PORTA 類比輸入功能

TRISA = 0x00;     // 將 PORTA 設為輸出模式
LATA = 0x55;      // PORTA 送出 0x55 的值

TRISA = 0xFF;     // 將 PORTA 設為輸入模式
BUF = PORTA;      // 讀取 PORTA 的輸入值
```

使用位元的控制

- 一樣所有的位元定義在：**pic18f45k22.h**

```
typedef union {
    struct {
        unsigned RA0            :1;
        unsigned RA1            :1;
        unsigned RA2            :1;
        unsigned RA3            :1;
        unsigned RA4            :1;
        unsigned RA5            :1;
        unsigned RA6            :1;
        unsigned RA7            :1;
    };
        :
        :
        :
} PORTAbits_t;
extern volatile PORTAbits_t PORTAbits @ 0xF80;
```

範例: **I/O** 腳位元的操作

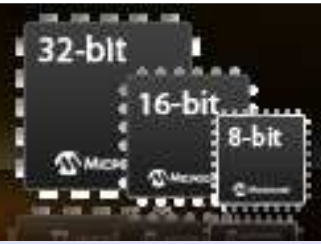
```
PORTAbits.RA0 = 1;
LATAbits.RA1 = 0;

RA0 = 1;
Nop( );    // 避免 Read-Modify-Writ 錯誤
RA1 = 1;

LATA0 = 1;
LATA1 = 0;  // = LATAbits.RA1 = 0;
```

```
extern volatile __bit _DEPRECATED    RA0        @ (((unsigned) &PORTA)*8) + 0;
#define                              RA0_bit      BANKMASK(PORTA), 0
extern volatile __bit _DEPRECATED    RA1        @ (((unsigned) &PORTA)*8) + 1;
#define                              RA1_bit      BANKMASK(PORTA), 1
```

I/O Ports 總結

Summary

- 在 PIC18 Family 中使用於 I/O 控制的暫存器有
  - TRISx
    - 控制 I/O port 的方向
  - PORTx
    - 對 I/O port 讀取/寫入用的暫存器
  - LATx
    - 對 I/O port 寫入時的栓鎖暫存器 – 建議使用此暫存器進行位元運算的操作



了解 **Timer1** 的工作

PIC101 Lab3.x

為何使用 **Timer** 來計時

- 前面 **PIC101 Lab1.x** 的練習採用軟體迴圈來計時
  - MCU 的執行能力都浪費在做 Delay 工作
  - MCU 無法在做其它的工作

- 如果使用 **Timer** 的硬體來協助計算時間**?**
- 可以使用 **Timer** 中斷方式來計時**?**

使用經驗分享**:**

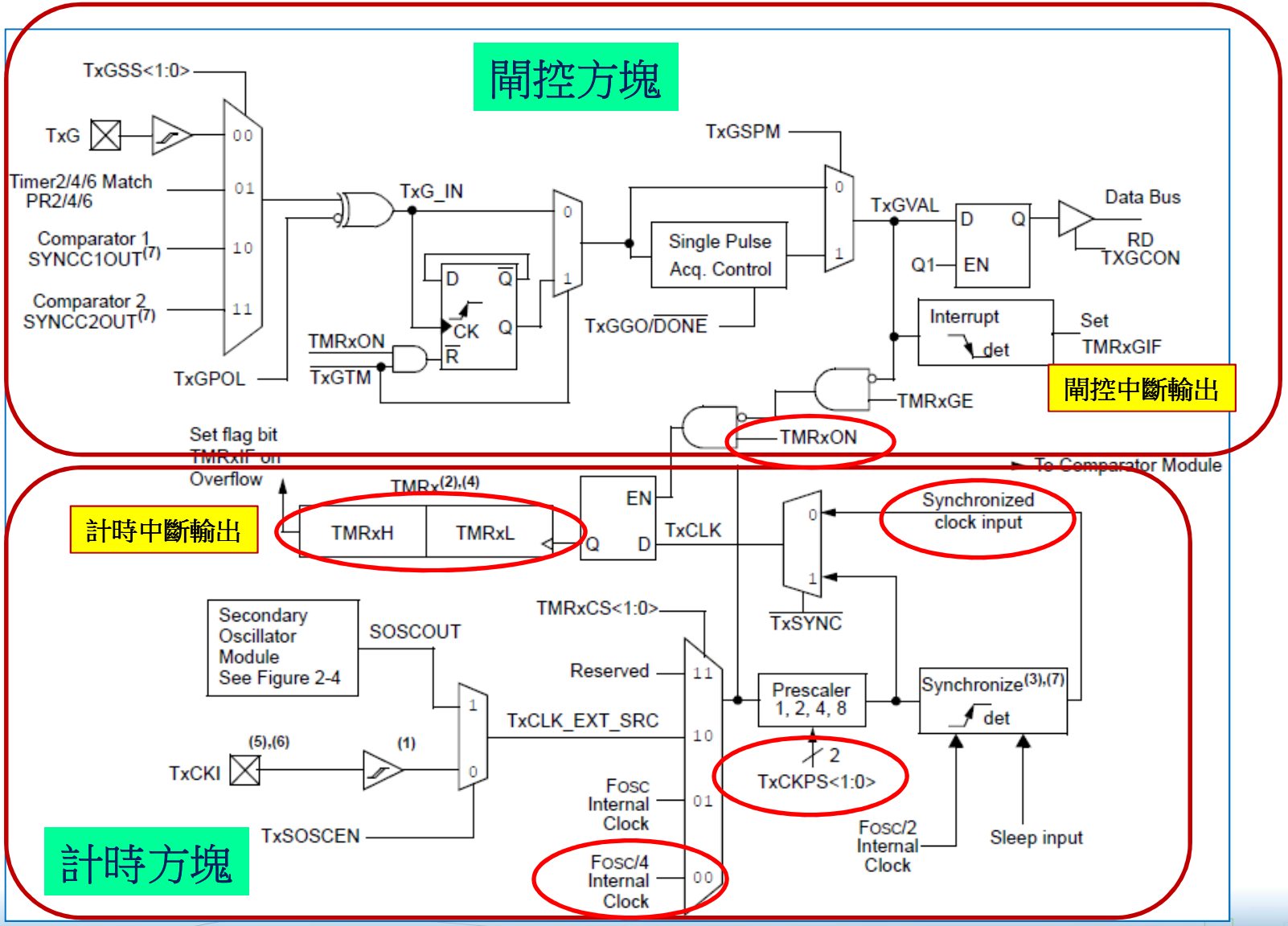
使用 **Timer** 中斷的計時方式會是最佳的解決方案

MICROCHIP
Regional Training Centers

Timer 的工作？

- **Timer / Counter 差異**
  - Timer 使用已知的頻率來計數做時間的量測
  - Counter 利用計數器量測外部輸入的訊號
    - 頻率、時間、週期、Duty Cycle
- **Timer 0/1/3/5 屬於計數溢位型**
  - 0xFFFF → 0x0000 溢位時， TxIF =1
  - 溢位後需重新載入計時值
- **Timer 2/4/6 屬於計數比較型**
  - 計數值與設定值做比較，相等時計數器歸零、TxIF = 1
  - 無需重新載入，無延遲的計時方式

Timer1/3/5 方塊圖



Timer1 的計時方塊

- 初學者，先了解計時方塊的使用及設定
  - 閘控計時功能 (Gate Time) – 先保留

REGISTER 12-1: TXCON: TIMER1/3/5 CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/0	R/W-0/u
TMRxCS<1:0>		TxCKPS<1:0>		TxSOSCEN	$\overline{\text{TxSYNC}}$	TxRD16	TMRxON
bit 7 0	0	1	1	0	1	1	1 bit 0

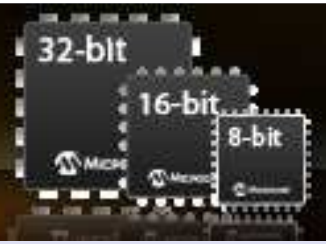
- **TMRxCS<1:0>:** Timer1/3/5 Clock Source Select bits **：選擇 Fosc/4**
- **TxCKPS<1:0>:** Timer1/3/5 Input Clock Prescale Select bits **：選擇 1:8 的除頻比率**
- **TxSOSCEN:** Secondary Oscillator Enable Control bit **：選擇 關閉**
- **TxSYNC:** Timer1/3/5 External Clock Input Synchronization Control bit **：選擇 非同步**

  TxRD16: 16-Bit Read/Write Mode Enable bit **：選擇 16-bit 時間讀寫模式**
- **TMRxON:** Timer1/3/5 On bit **：選擇 啟動 Timer1**

其他相關的 Timer1 控制位元

- **IPR1bits.TMR1IP (Data Sheet Page 127)**
  - 中斷高、低優先權設定位元
- **PIE1bits.TMR1IE (Page 123)**
  - 中斷致能設定位元
- **PIR1bits.TMR1IF (Page 118)**
  - 計時溢位指示旗號 (需用軟體清除)

練習 **PIC101 Lab3.x** 使用 **Timer1** 來計時
使用**Polling** 方式來執行每 **125mS** 將
PORTD 的 **LED** 加一的計數



實驗
PIC101 Lab3.x

使用 **Timer1 Polling** 計時方式
每 **125mS** 將 **PORTD** 的 **LED** 加一

125mS 的時間計算

- **Timer1 使用：Fosc/4 (Fcy)，1:8 預除器，系統頻率 (Fosc) = 8MHz**

➢ 如何計算 **125mS** 的計時:

(8MHz/4) / 8 (預除比) = 250　即每 **1mS** 所需的計數值

➢ 那要計時 **125mS** 時所需的計數值為:

250 x 125mS = 31250

➢ 因為是使用溢位的計時方式，所以需要用:

65536- 32150 = 34286 = 0x85EE

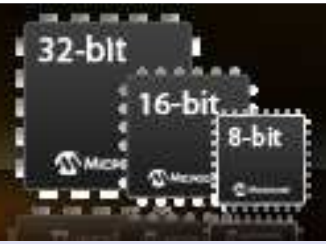
TMR1H = 0x85

TMR1L = 0xEE

PIC101 Lab3.x

Lab3.c 解說

```
void main(void)
{
  OSCCONbits.IRCF = 0b110 ;          // 調整 IRCF 位元來設定工作頻率
                                     // Select INTOSC-8Mhz as clock source
                                     // 此位元結構成員名稱，請參考 pic18f45K22.h 檔裡的名稱定義
                                     // 參考一下 Data Sheet REGISTER 2-1: OSCCON: OSCILLATOR CONTROL REGISTER
                                     // OSCCON 暫存器 bit 6-4 IRCF<2:0>: Internal RC Oscillator Frequency Select bits(2)
   OSCTUNE = 0x00;                   // 關閉軟體 x4 PLL (PLLEN 位元) 功能
  Nop();

  T1CON = 0b00110111;      //T1CKPS 1:8; T1OSCEN 關閉; T1SYNC 非同步模式; TMR1CS FOSC/4; TMR1ON 開啟; T1RD16 開啟;
  T1GCON = 0x00;           //T1GSS T1G; TMR1GE 關閉; T1GTM 關閉; T1GPOL low; T1GGO done; T1GSPM 關閉;
  TMR1H = 0x85;            //TMR1H 133，預載入 MSB (125mS)
  TMR1L = 0xEE;            //TMR1L 238，同時載入載入 125mS 計數值到 Timer1(16-bit 載入)

  TRISD = 0x00;                      // 設定 PORTD 為輸出腳功能
  LATD = 0x00 ;                      // Turn Off all LEDs

  while (1)
  {
    while(!PIR1bits.TMR1IF);         // 檢查 Timer1 250mS 計時溢位了嗎? (沒有，繼續等待)
    PIR1bits.TMR1IF=0;               // 時間溢位，清除旗號
    TMR1H = 0x85;                    //TMR1H 133，預載入 MSB (125mS) (重新載入Timer1 125mS 的計時)
    TMR1L = 0xEE;                    //TMR1L 238，同時載入載入 125mS 計數值到 Timer1(16-bit 載入)
    LATD++;                          // PORTD 計數 LED 加一
  }
}
```



實驗
PIC101 Lab3-1 MCC.x

將 **PIC101 Lab3.x** 的練習
改成使用 **MCC** 的設定方式

一樣是使用 **Timer1 Polling** 計時方式
每 **125mS** 將 **PORTD** 的 **LED** 加一

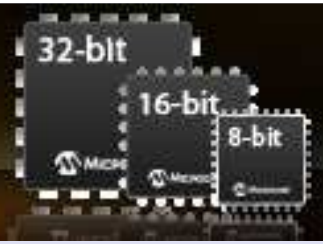
比較一下 使用 **MCC** 的設定

- **PIC101 Lab3-1 MCC.x** 是使用 **MCC v3.05** 來做 **Config. Bits**、**Timer1** 及 **PORTD** 腳位的設定。
- 與 **PIC101 Lab3.x** 的功能一模一樣

比較一下這兩種設定方式的優點是?

PIC101 Lab3-1 MCC.x 概述

1. **APP001** 硬體上電並接上 **PICKit3**
2. 開啟 **MPLAB® X IDE**
3. 開啟專案，目錄及專案檔案名稱:
   - **..\PIC101 Starter RTC**
     PIC101 Lab3-1 MCC.X
4. 開啟 **MCC (MPLAB® Code Configurator)**
5. **PIC®** 的配置設定 **(Config.)** (如前之設定)
6. **Timer1** 的設定
7. **PORTD** 的設定



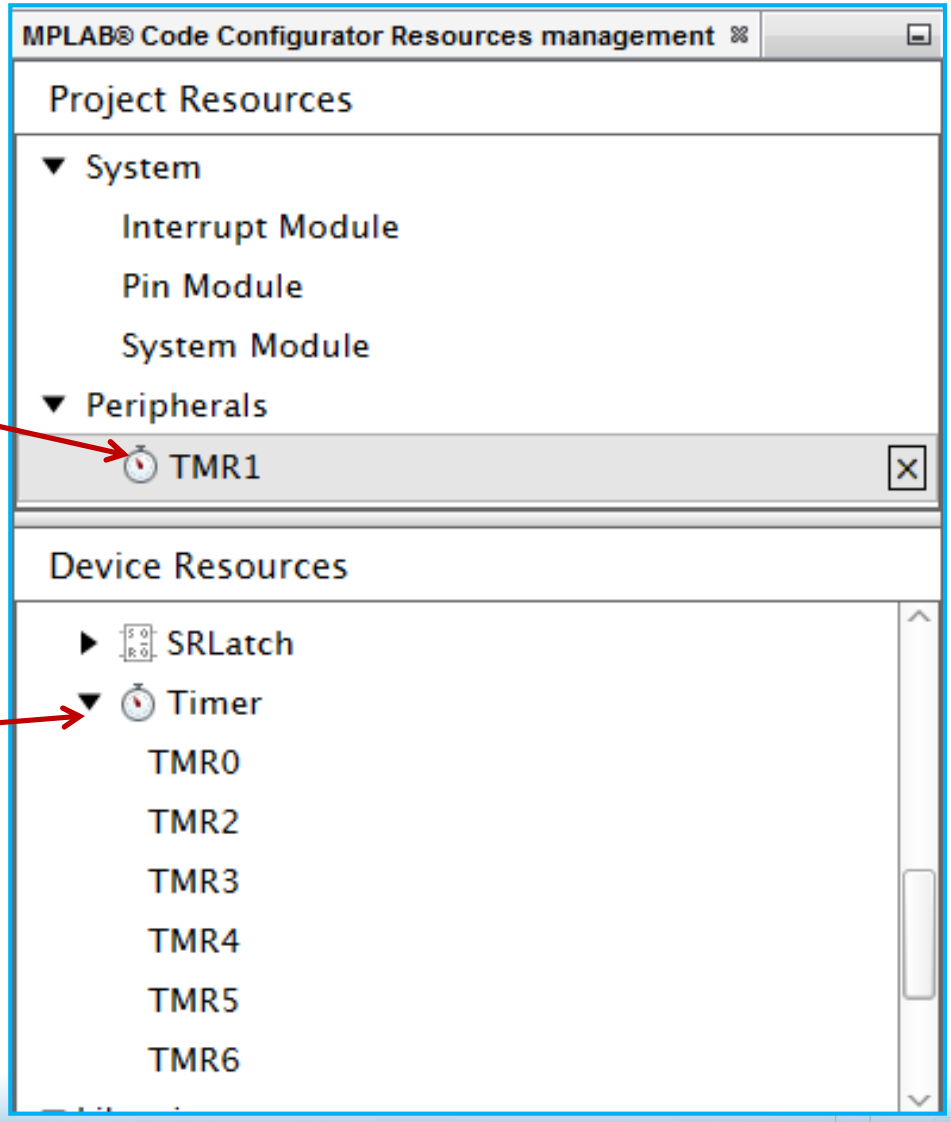
MCC
Timer1 的設定

Timer1 設定

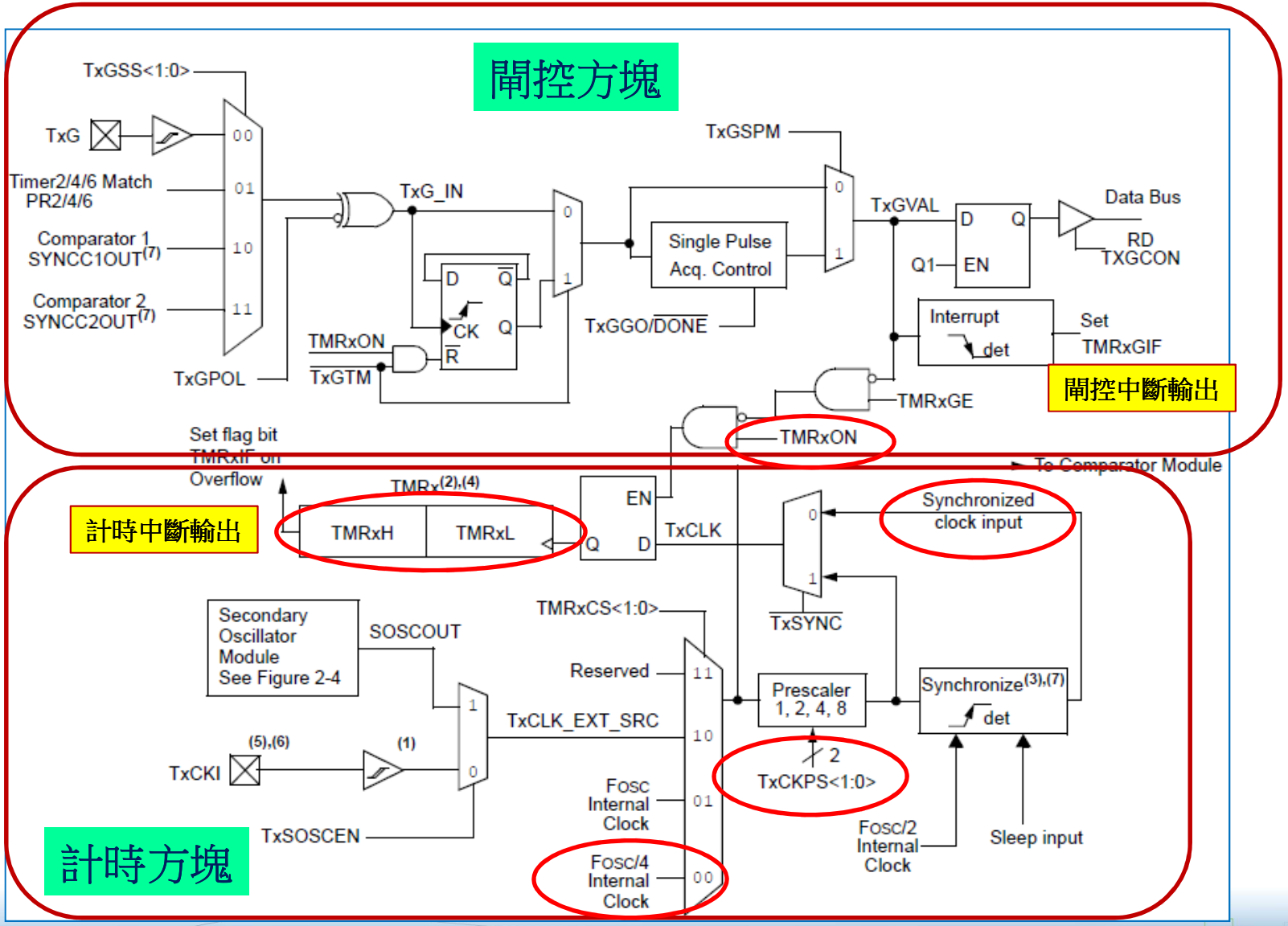
- 選擇 **TMR1**

2. Project Resources
點選 **TMR1** 的模組後將會開啟 **Timer1** 的設定視窗

1. Device Resources
雙點選 **“Timer”** 後顯示所有的 **Timer** 模組

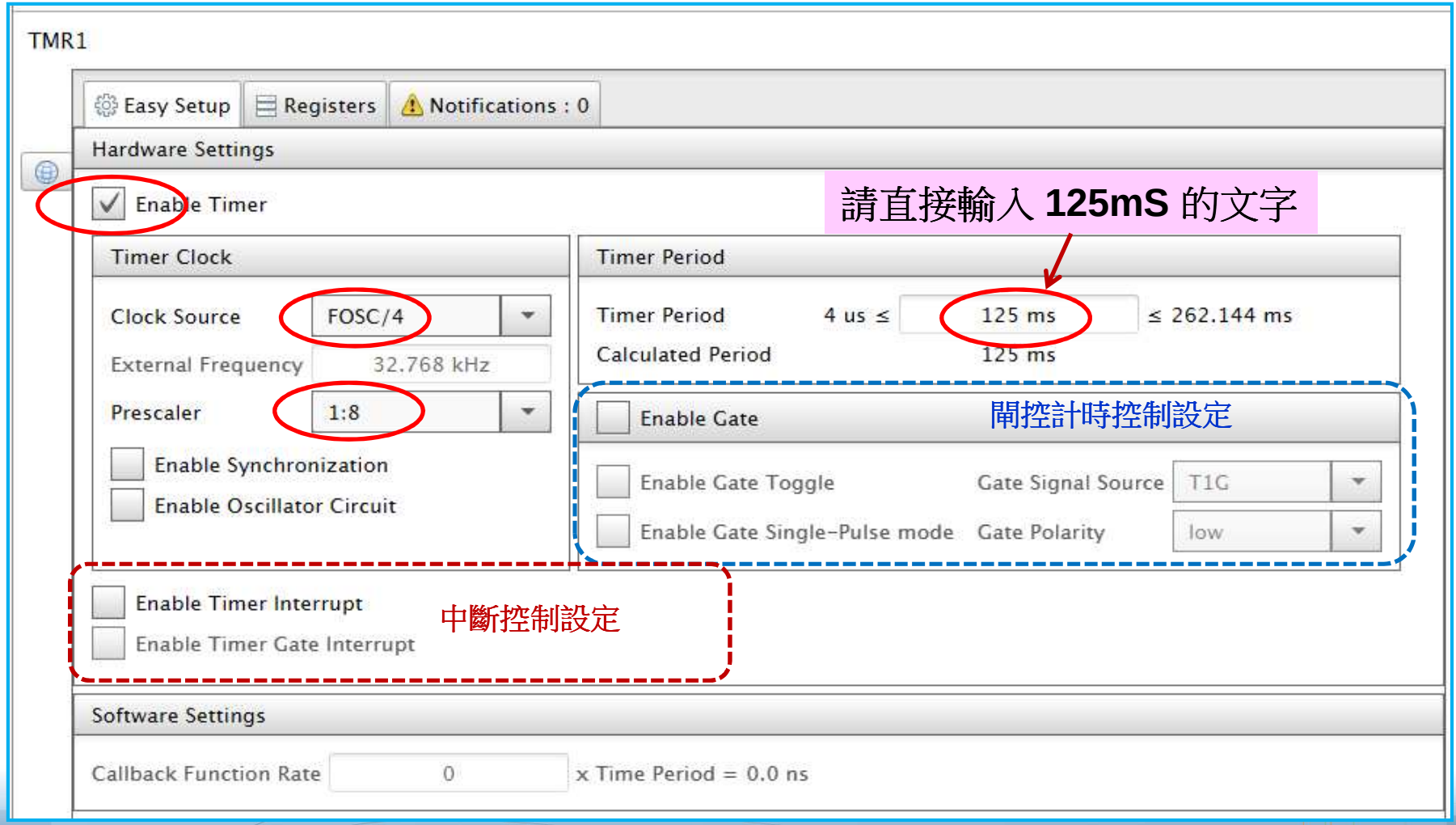


Timer1/3/5 方塊圖



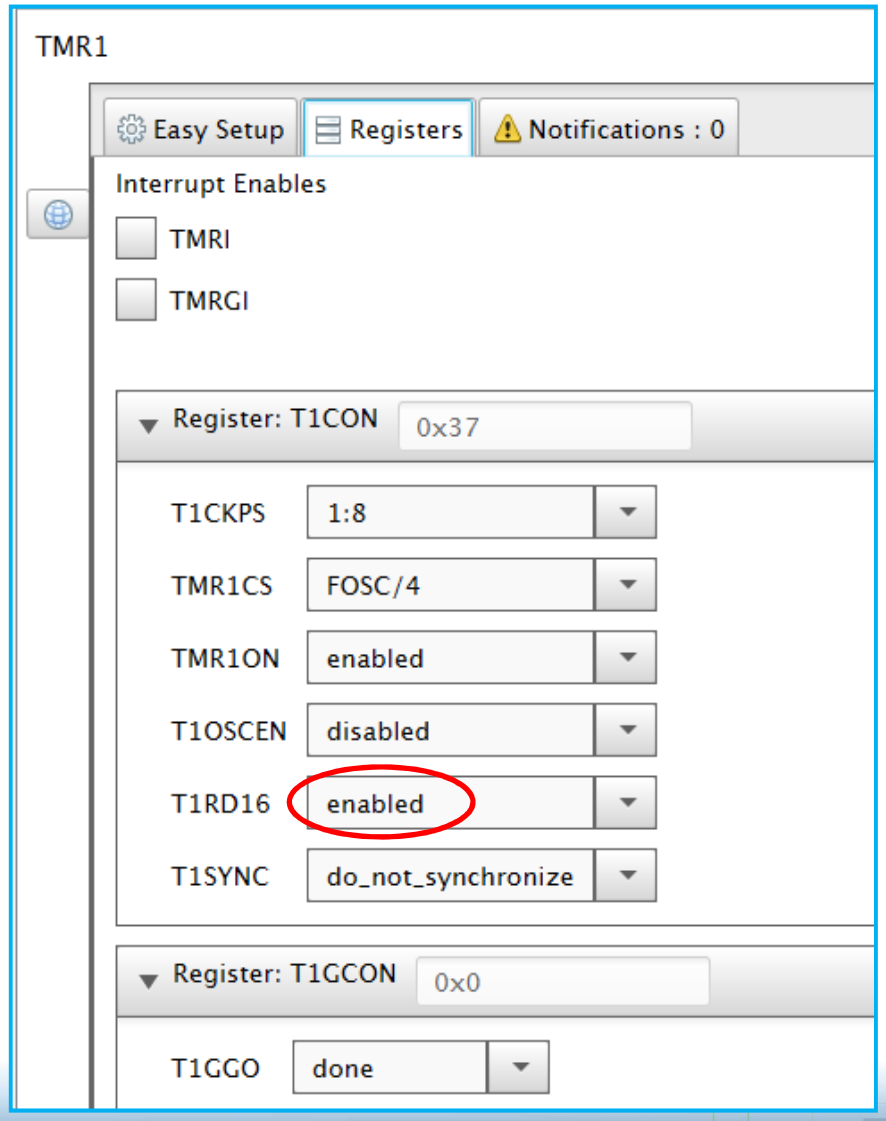
Timer1 設定

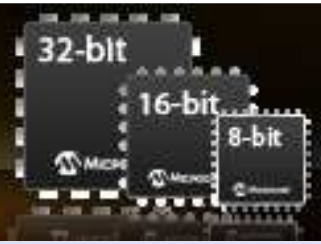
- **Timer1** 設定說明



Timer1 暫存器檢視功能

- **MCC** 提供暫存器檢視功能，按一下 **Registers** 選卡
- 右圖顯示 **Timer1** 的暫存器目前的設定值
- 請將 **T1RD16** 開啟，啟用 **16-bit** 讀、寫模式



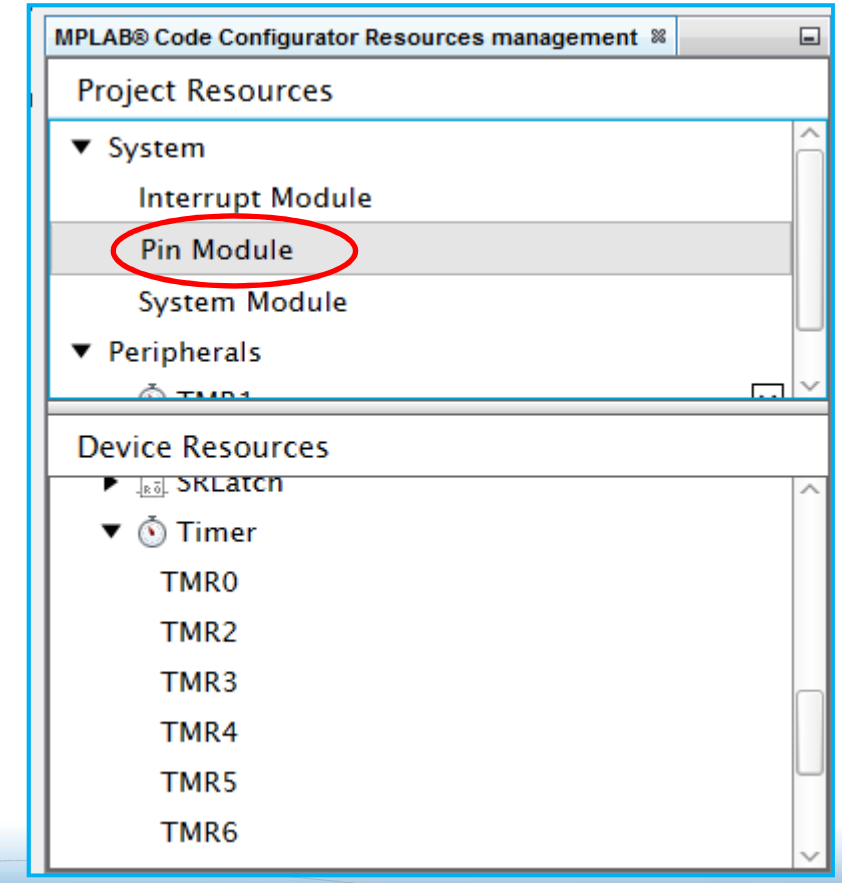


MCC

PORTD 腳位的設定

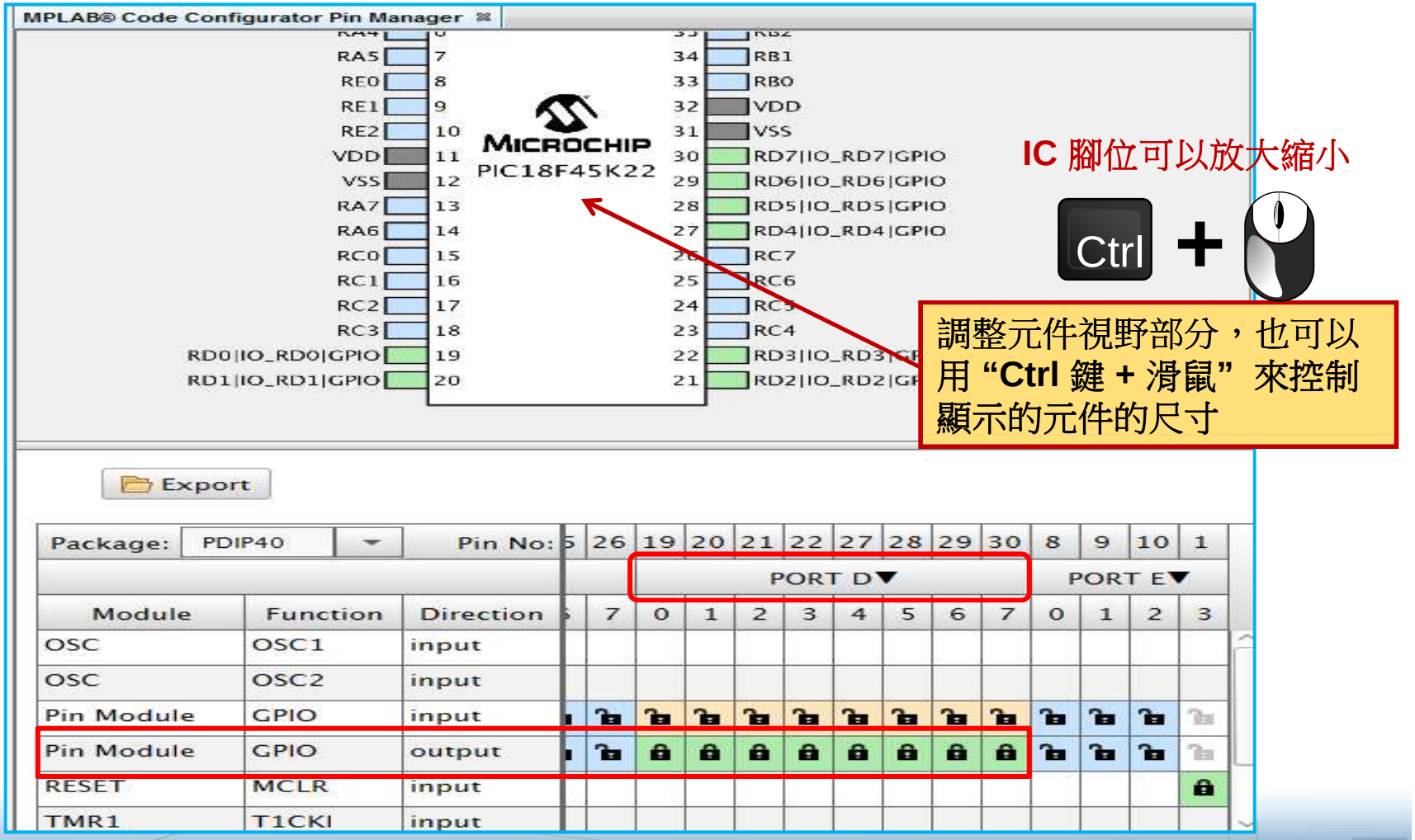
PORTD 腳位設定

- 在 **Project Resources** 視窗下
  - 點選 “Pin module” 開啟腳位設定視窗



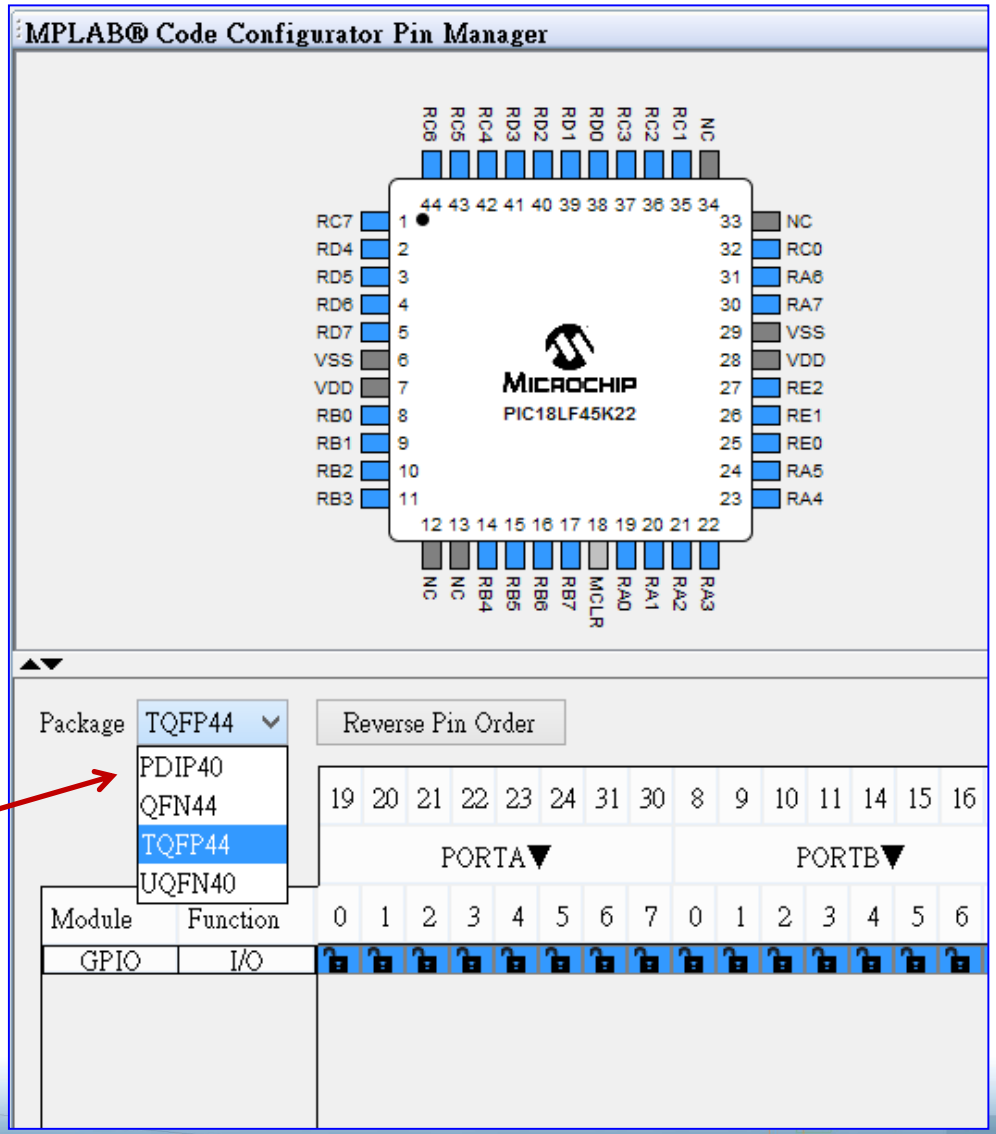
PORTD 腳位設定

- PORTD 為 LED 的輸出腳位



PORTD 腳位設定

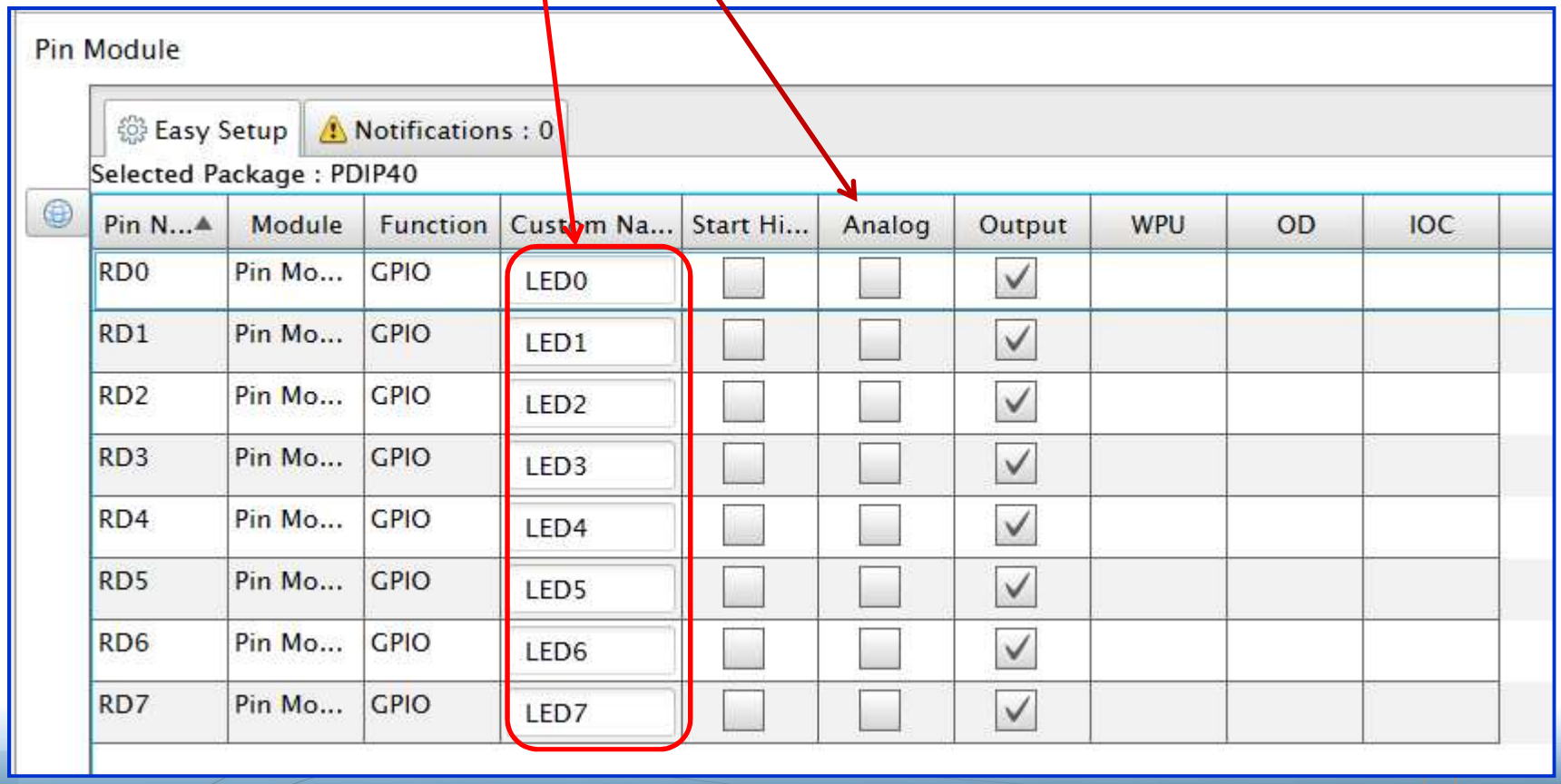
- 元件包裝的變更
  - PDIP40
  - QFN44
  - TQFP44
  - UQFN40



選擇所需的包裝

PORTD 腳位設定

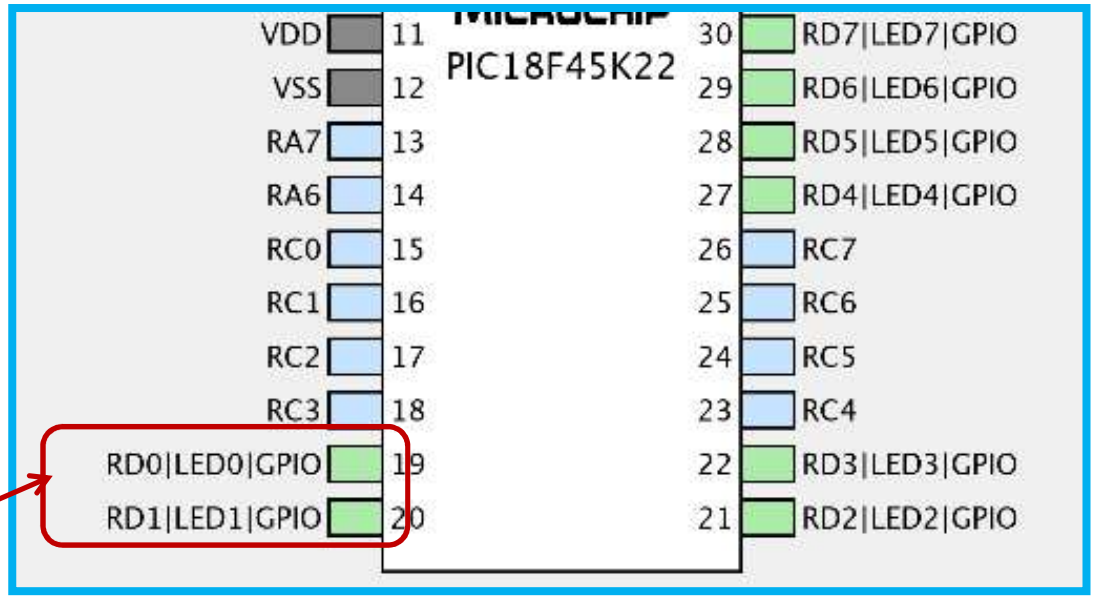
- 在 **Pin Module** 視窗下設定 **PORTD**
  - 關閉 **Analog** 輸入功能
  - 在 **Customer Name** 賦予腳位名稱



Pin N...	Module	Function	Custom Na...	Start Hi...	Analog	Output	WPU	OD	IOC
RD0	Pin Mo...	GPIO	LED0	☐	☐	☑			
RD1	Pin Mo...	GPIO	LED1	☐	☐	☑			
RD2	Pin Mo...	GPIO	LED2	☐	☐	☑			
RD3	Pin Mo...	GPIO	LED3	☐	☐	☑			
RD4	Pin Mo...	GPIO	LED4	☐	☐	☑			
RD5	Pin Mo...	GPIO	LED5	☐	☐	☑			
RD6	Pin Mo...	GPIO	LED6	☐	☐	☑			
RD7	Pin Mo...	GPIO	LED7	☐	☐	☑			

PORTD 腳位設定

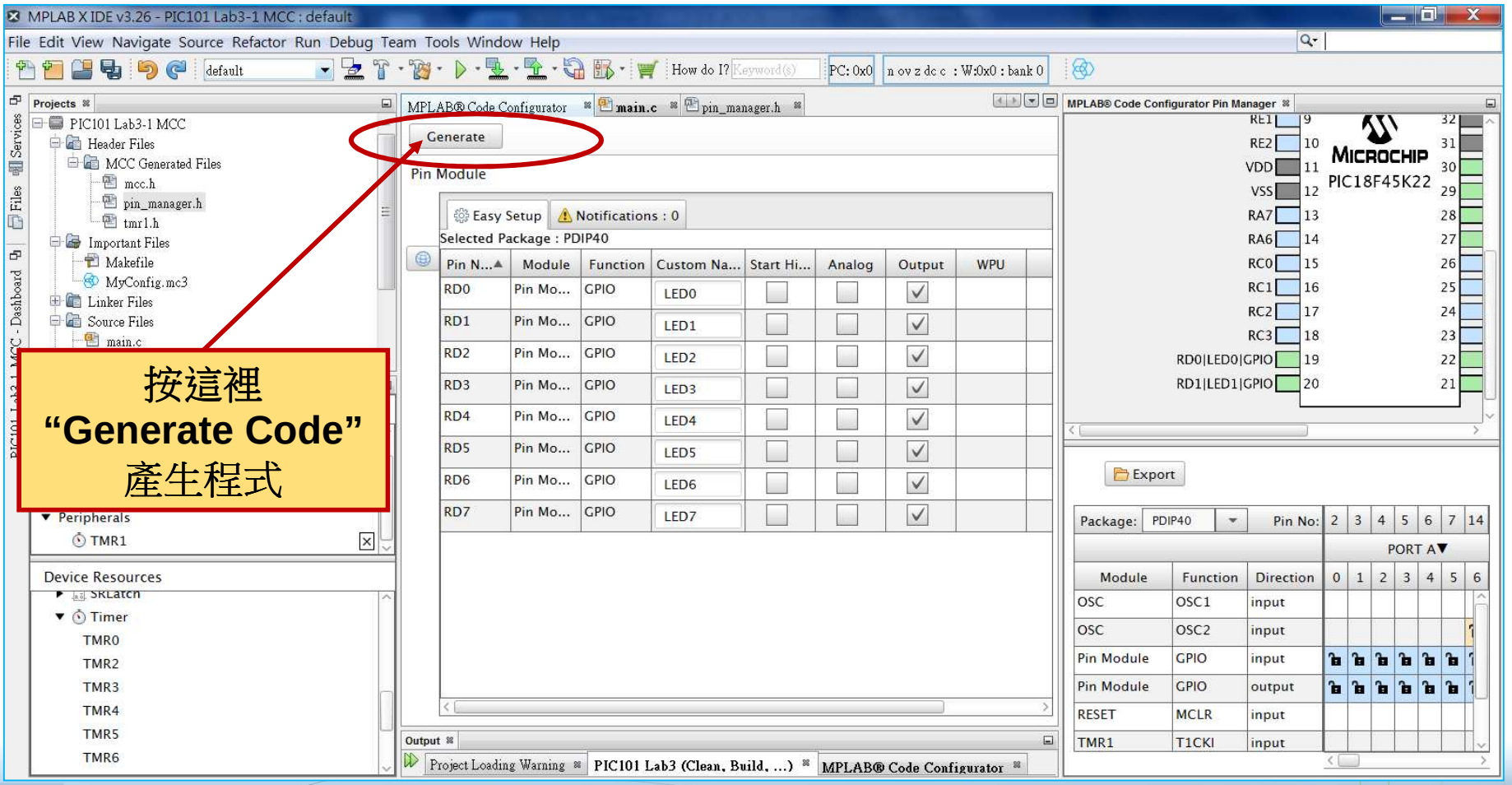
- 設定 **PORTD** 為輸出腳功能，並更改名稱為 **LED0 ~ LED7 (**程式使用**)**
- 新設定腳位名稱會在腳位定義檔 **Pin-Manager.h**



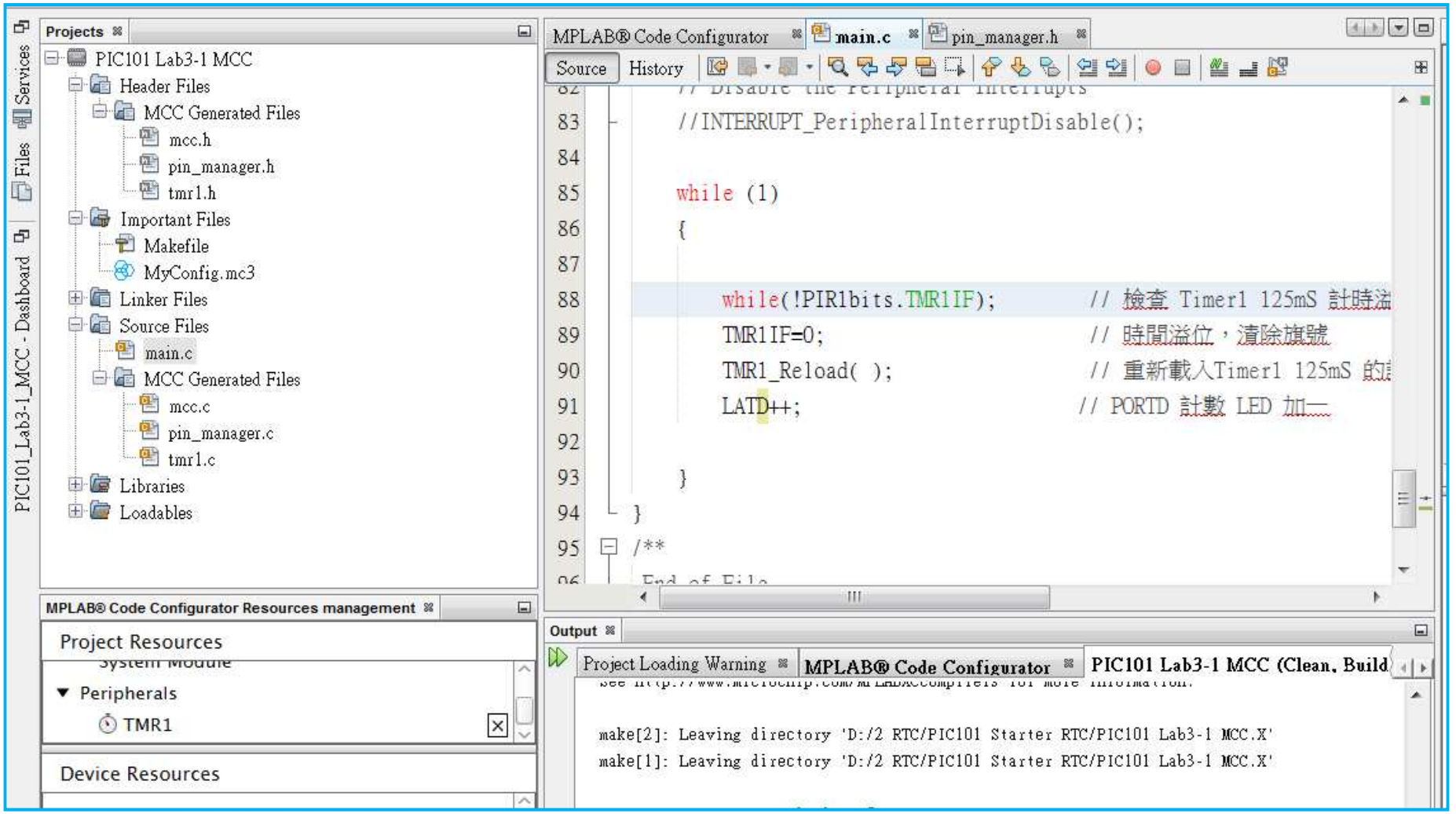
腳位名稱變更完後新的腳位名稱會同步顯示在 該包裝的腳位上

程式產生器

- 開始產生 C 程式並加到專案裡



專案下的檔案
main.c

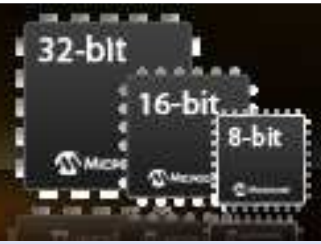


MCC 產生的檔案

- **MCC** 所產生的程式
  - 周邊函數的 C 原始程式(函數)
  - 使用方法: 在該周邊的 h 檔裡的原型宣告裡有使用方法

- 更多 **MCC** 的說明請參考教育訓練光碟下的 **8-bit** 教材:

MCC201 v1.00 MPLAB Code Configurator
New!



PIC18F 中斷說明

(附錄 A)

何謂中斷 (突發事件發生)

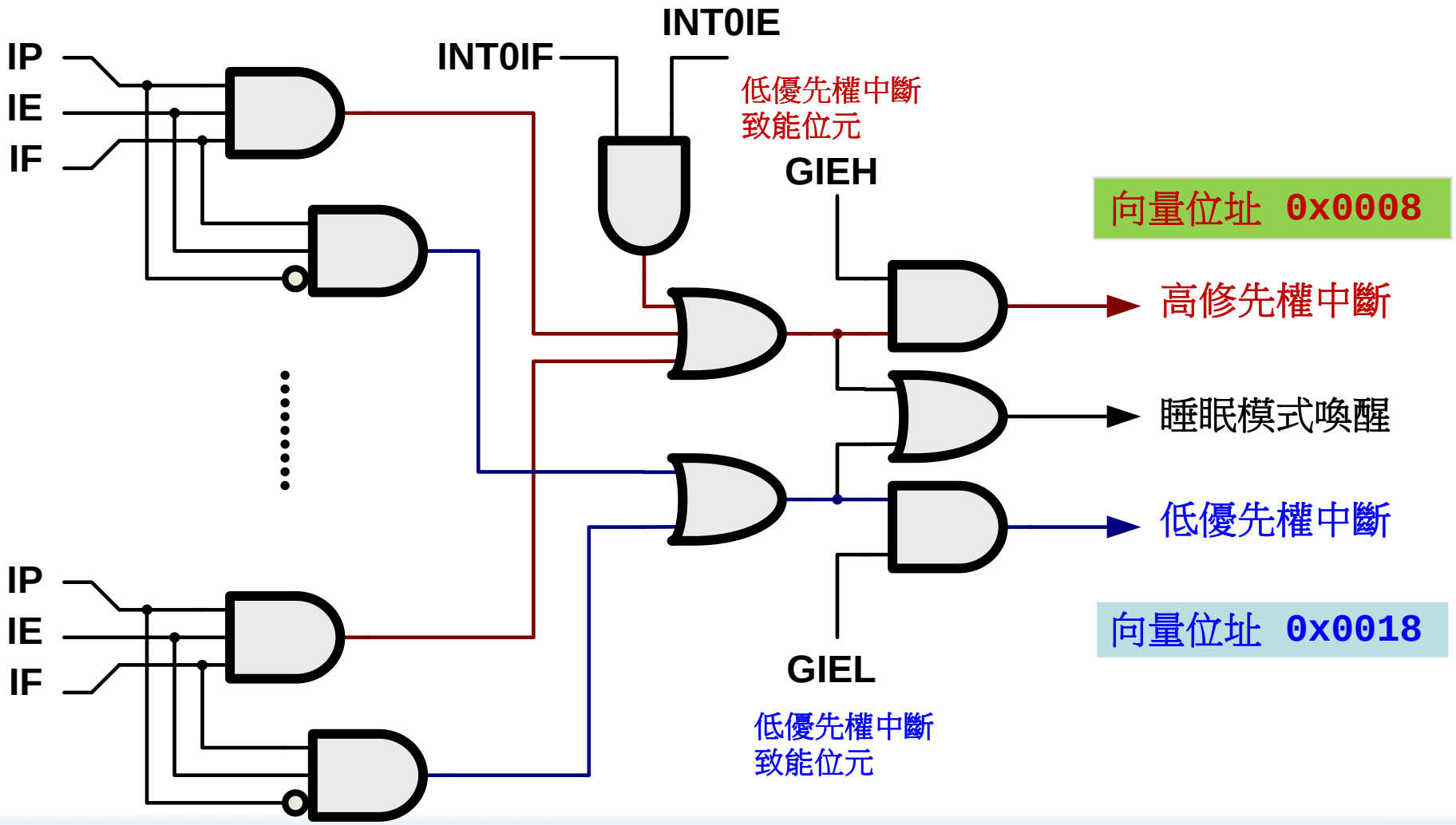
- 程式的執行一般是按照流程一步一步的在順序執行。如果這時周邊有事件發生需要程式(立即或待會)去做短暫的執行，這短暫跳至突發事件的執行稱為中斷。
- 程式在一般的執行時，中斷可被關閉或開啟。也即中斷是可以被控制的。
  - Timer 的計時、UART 接收到資料，ADC 轉換完成，I/O 腳位中斷需求、.....等。

XC8 的中斷

- PIC18F 的中斷方塊圖
- XC8 對 PIC18 的高、低優先權中斷的宣告
- 有關中斷控制的相關位元

PIC18Fxxxx 中斷邏輯電路

高、低優先權模式



PIC18F 中斷處理

- **PIC18Fxxxx** 有兩個中斷進入點
  - 高優先權的中斷服務位址: 0x0008
    - 高優先權控制位元： **GIEH**
  - 低優先權的中斷服務位址: 0x0018
    - 低優先權控制位元：**GIEL**
  - 每個中斷源均可選擇其中斷優先權 (xxIP)
  - 每個中斷源均有獨立的中斷旗標 (xxIF)
  - 每個中斷源均可單獨開啟或關閉 (xxIE)
  - 中斷旗標 (xxIF) 的清除 ==> 大部分需自行用軟體方式清除(詳細請看 Data Sheet)

PIC18F 的 Shadow 暫存器

- **Shadow Register**
  - 提高中斷程式對事件的反應速度
- 高優先權中斷
  - W，BSR，STATUS 自動存入Shadow Register
  - 程式的返回 RETFIE FAST
    - 自 **Shadow Register** 取回暫存值
- 低優先權中斷
  - W，BSR，STATUS 的存入、取出需透過軟體堆疊
  - 程式的返回 : RETFIE 0

中斷注意事項

- 中斷函數無參數傳遞功能
- 中斷所使用的變數需加入 volatile 的宣告
- 中斷函數儘量不要使用 Local 變數，引響中斷響應時間
- 中斷越短越好，不要做太多的處理，可以設定 Flag 後交給主程式處理
- 中斷函數最好不要做額外的數學的運算
- 中斷裡不要再呼叫主程式有在使用的函數

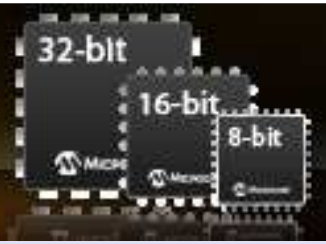
XC8 的中斷函數宣告

- 高優先權中斷函數
  - void interrupt HighISR(void)
  - 無法傳入及回傳參數
- 低優先權中斷函數
  - void interrupt low_priority LowISR(void)
  - 無法傳入及回傳參數

注意: 除了中斷函數的設定，其他相關的
中斷啟用位元 **(GIEH、GIEL & xxIE)** 及
優先權控制位元 **(xxIP)**也要設定

XC8 支援 PIC18 函數庫

- **XC8** 自 **v1.35 (含)** 以後的版本不內建 **PIC18F** 的周邊函數庫
  - 使用 MCC 產生周邊函數
  - MCC : PIC16F1xxx, PIC18 K系列
- 如需 **PIC18F** 周邊函數庫 請自行安裝
- **Lab4.x** 裡有用到 **Write Timer0 /1**
  - 所以請到教育訓練光碟安裝一下:
    XC8 Peripheral Libraries v2.0RC3



中斷的練習

PIC101 Lab4.x

(附錄 A)

PIC101 Lab4.x 中斷練習

PIC18F 高、低優先權中斷練習

- 在 **MPALB X IDE** 下開啟底下的專案
  - **..\ PIC101 Starter RTC\PIC101 Lab4.x**
- 本實驗的 **Fosc** 使用 內建 **RC 1MHz** 的速度
- 程式執行後的結果:
  - LED0 使用 Timer0 並設成高優先權中斷，每 500mS 閃爍一次
  - LED7 使用 Timer1 並設成低優先權中斷，每 250mS 閃爍一次

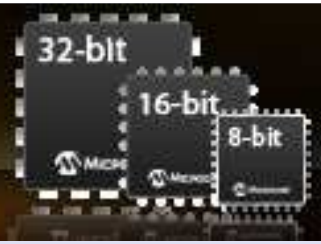
Lab4.x 裡相關的中斷位元

- 單單只有中斷函數設定中斷還是無法啟動的
- 有那些中斷位元還要做設定
  - 要使用 Timer0 & Timer1 的中斷前置設定

```
INTCON2bits.TMR0IP = 1;     // 設定 Timer0 為高優先權中斷
INTCONbits.T0IE = 1;        // 啟用 Timer0 中斷

IPR1bits.TMR1IP = 0;        // 設定 Timer1 為低優先權中斷
PIE1bits.TMR1IE = 1;        // 啟用 Timer1 中斷

RCONbits.IPEN = 1;          // 啟用高、低優先權控制機制
INTCONbits.GIEH = 1;        // 開啟高優先權中斷總控制位元
INTCONbits.GIEL = 1;        // 開啟低優先權中斷總控制位元
```

Lab4.c 中斷範例程式
Fosc = 1MHz

```
void interrupt HighISR(void)          // XC8 高優先權中斷函數
{
   if (TMR0IE && TMR0IF)
   {
      WriteTimer0(65535-487);         // 500mS Period, 500mS/4uS/256 = 488
      LATD0 = !LATD0;                 // LED1 (D1) 250mS 轉態一次
      TMR0IF=0;                       // 清除 Timer0 的中斷旗號
   }
}

void interrupt low_priority LowISR(void)    // XC8 低優先權中斷函數
{
   if (TMR1IF && TMR1IE)
   {
      WriteTimer1(65535-7811);        // 250mS Period, 250mS/4uS/8 = 7812
      LATD7 = !LATD7;                 // LED7 (D8) 500mS 轉態一次
      TMR1IF = 0;                     // 清除 Timer1 的中斷旗號
   }
}
```



使用 MCC 的設定
PIC18F 中斷

PIC101 Lab4-1 MCC.x

(附錄 A)

實驗 PIC101 Lab4-1 MCC.x

- **Lab4.x** 採用一般的寫法
  - 周邊暫存器直接設定
  - 呼叫周邊函數庫
- 將 **Lab4.x** 的計時功能改用 **MCC v3.x** 重新改寫
  - LED0 使用 Timer0 並設成高優先權中斷，每 500mS 閃爍一次
  - LED7 使用 Timer1 並設成低優先權中斷，每 250mS 閃爍一次
  - 比較一下哪種寫法較容易

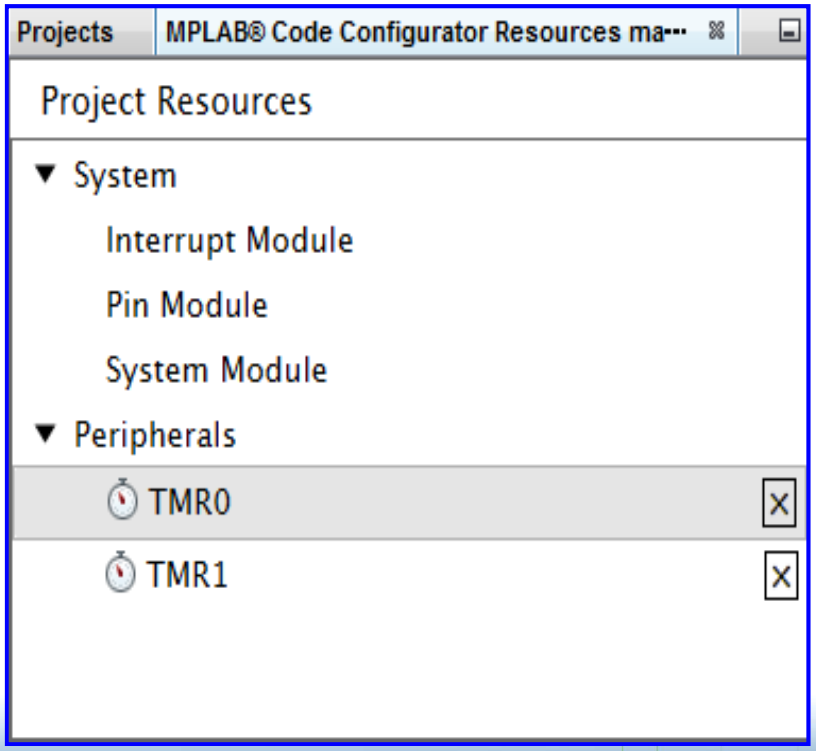
PIC101 Lab4-1 MCC.x 中斷練習

PIC18F 高、低優先權中斷練習

- 在 **MPALB X IDE** 下開啟底下的專案
  - **..\ PIC101 Starter RTC\PIC101Lab4-1 MCC.x**
- 本實驗的 **Fosc** 使用 內建 **RC 1MHz** 的速度
- 使用 **MCC** 來設定
  - Config. Bits (Fosc=1MHz)
  - Timer0 (高優先權中斷設定)
  - Timer1 (低優先權中斷設定)
  - Interrupt
  - PORTD & LCD Module (腳位最後才設定)

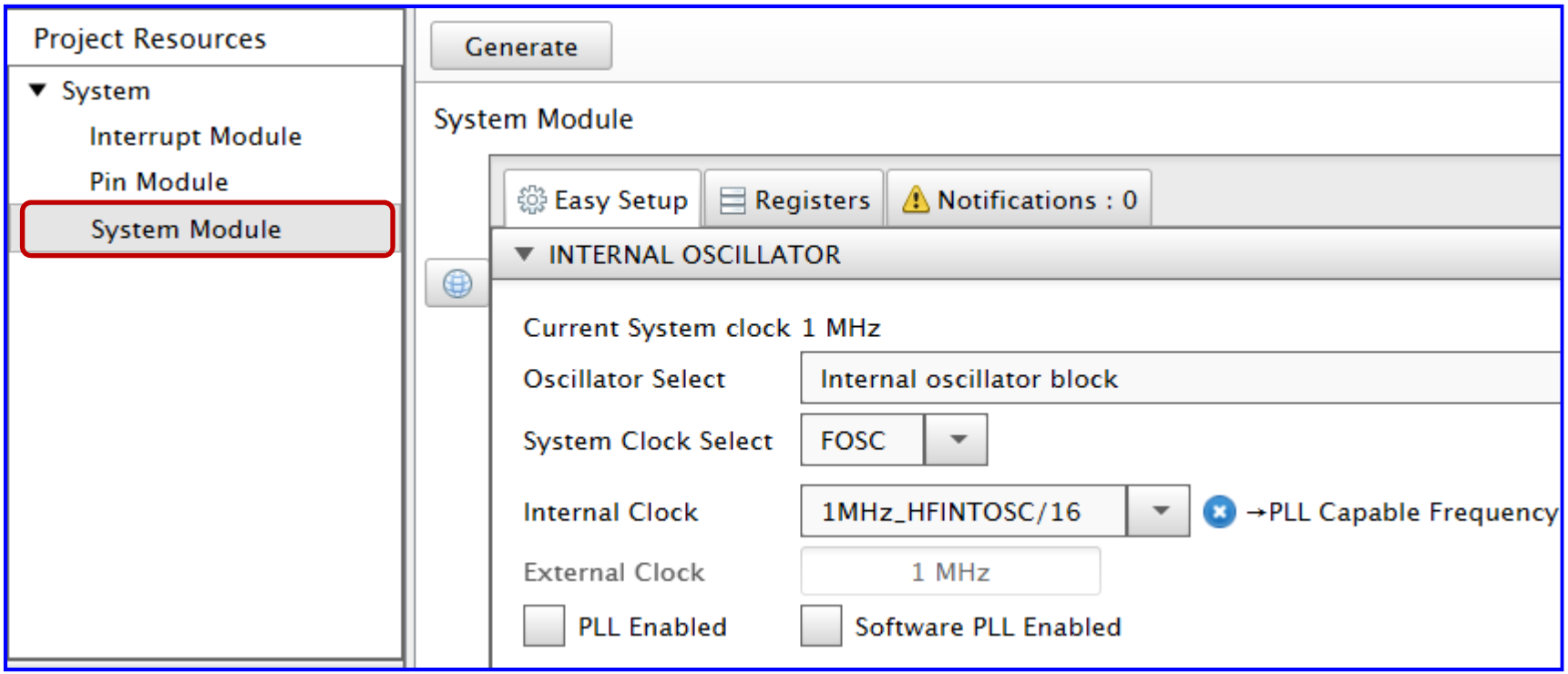
開始用 MCC 來設定

- 下圖為”專案資源(MCC)視窗“，點選適當模組開啟相關設定功能
  - System Module – 工作頻率 & Config. Bit
  - TMR0 – Timer0 的設定
  - TMR1 – Timer1 的設定
  - Interrupt Module
  - Pin Module – 腳位規劃設定 (該項請最後規劃)



Configuration Bits 設定

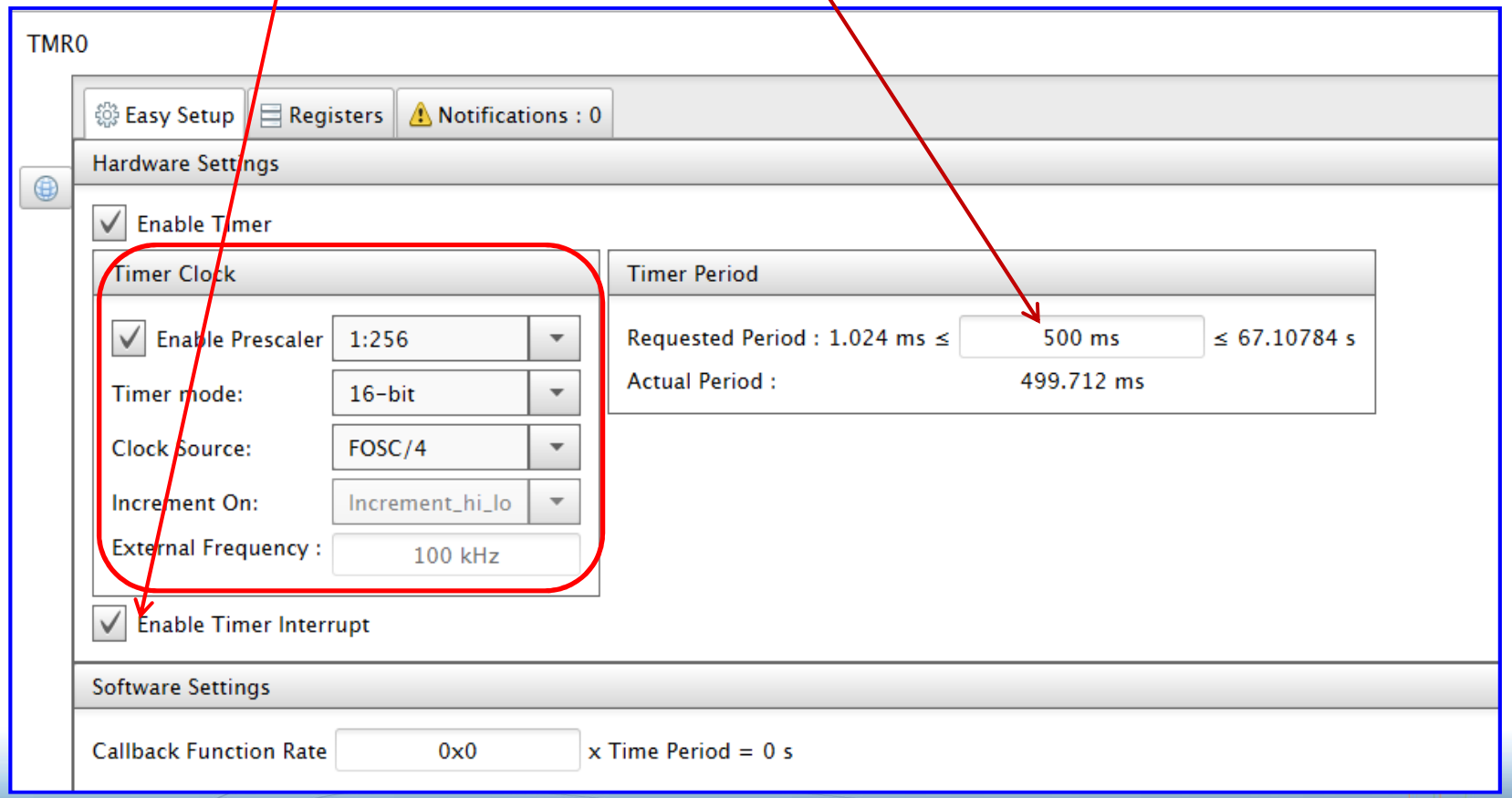
- **Fosc = Internal RC @ 1MHz**
- **Config. Bits 如前所設定一樣**



Timer0

Delay 500mS & 開啟中斷

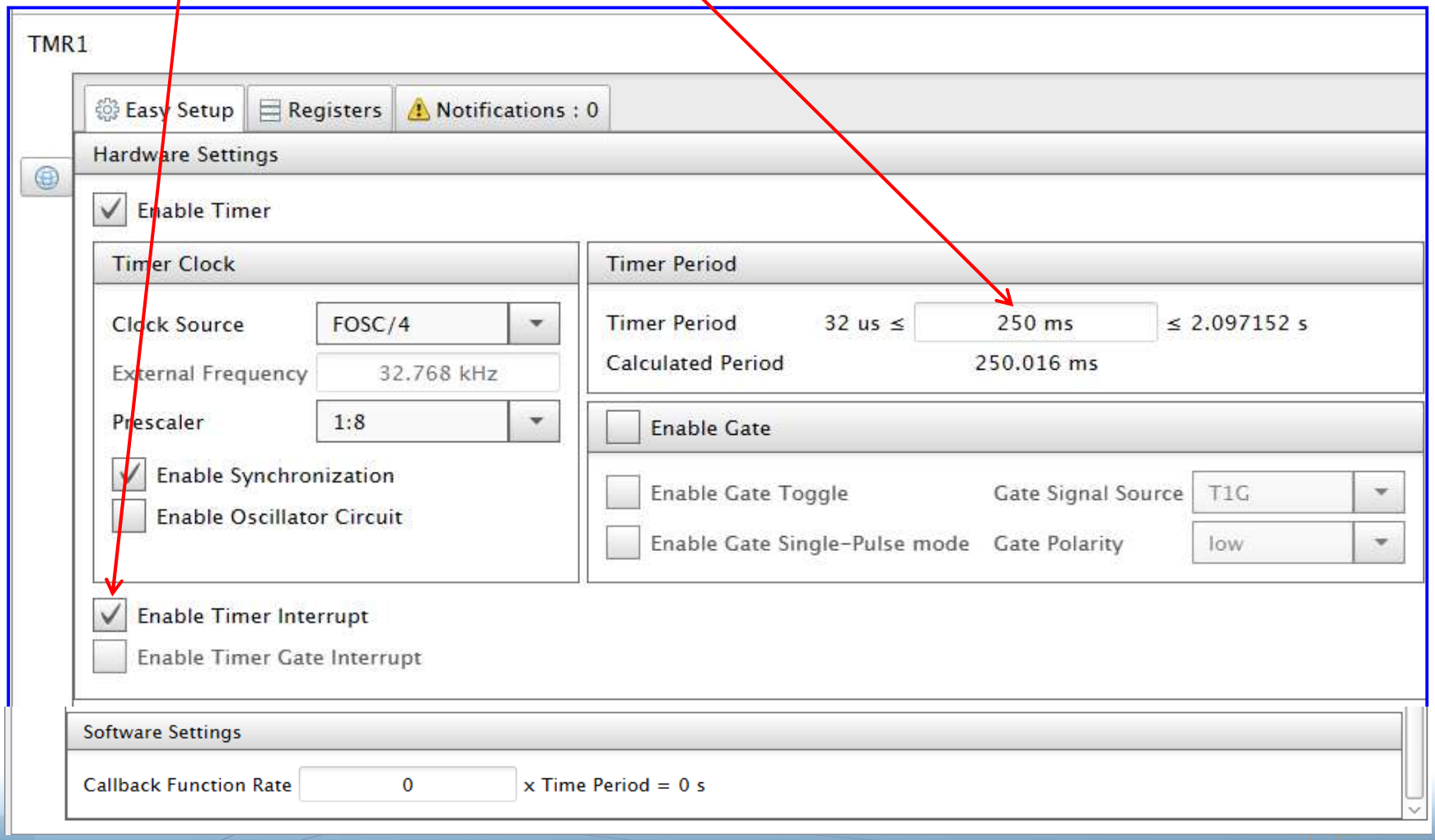
- **TMR0** 的設定
  - 開啟中斷，計時 500mS



Timer1

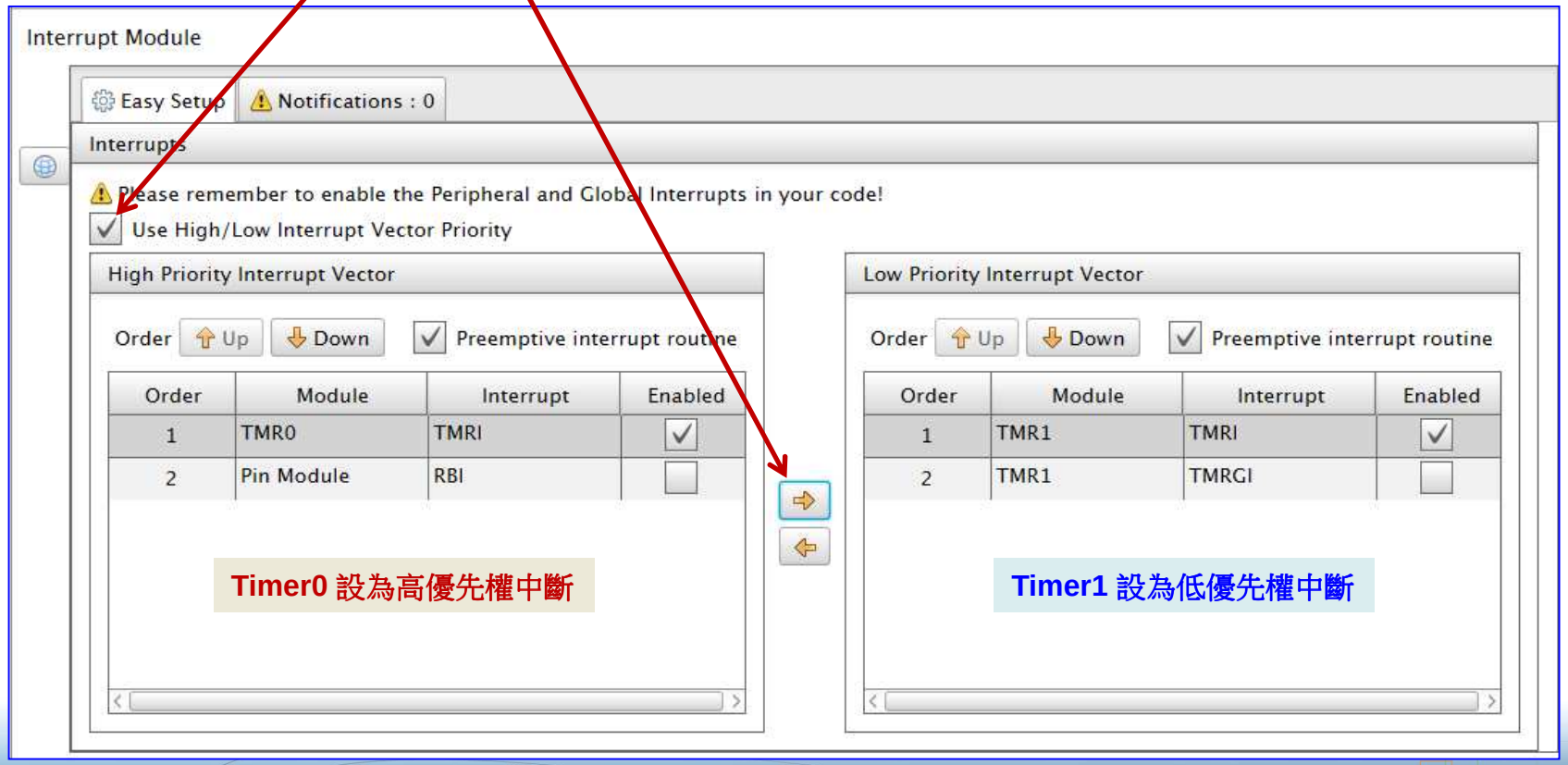
Delay 250mS & 開啟中斷

- 開啟中斷，計時 **250mS**



高、低優先權中斷設定

- **Interrupt Module** 的設定
  - 先啟用 High/Low Interrupt Vector Priority
  - 利用左右箭頭來變換中斷優先權



APP001_LCD.c

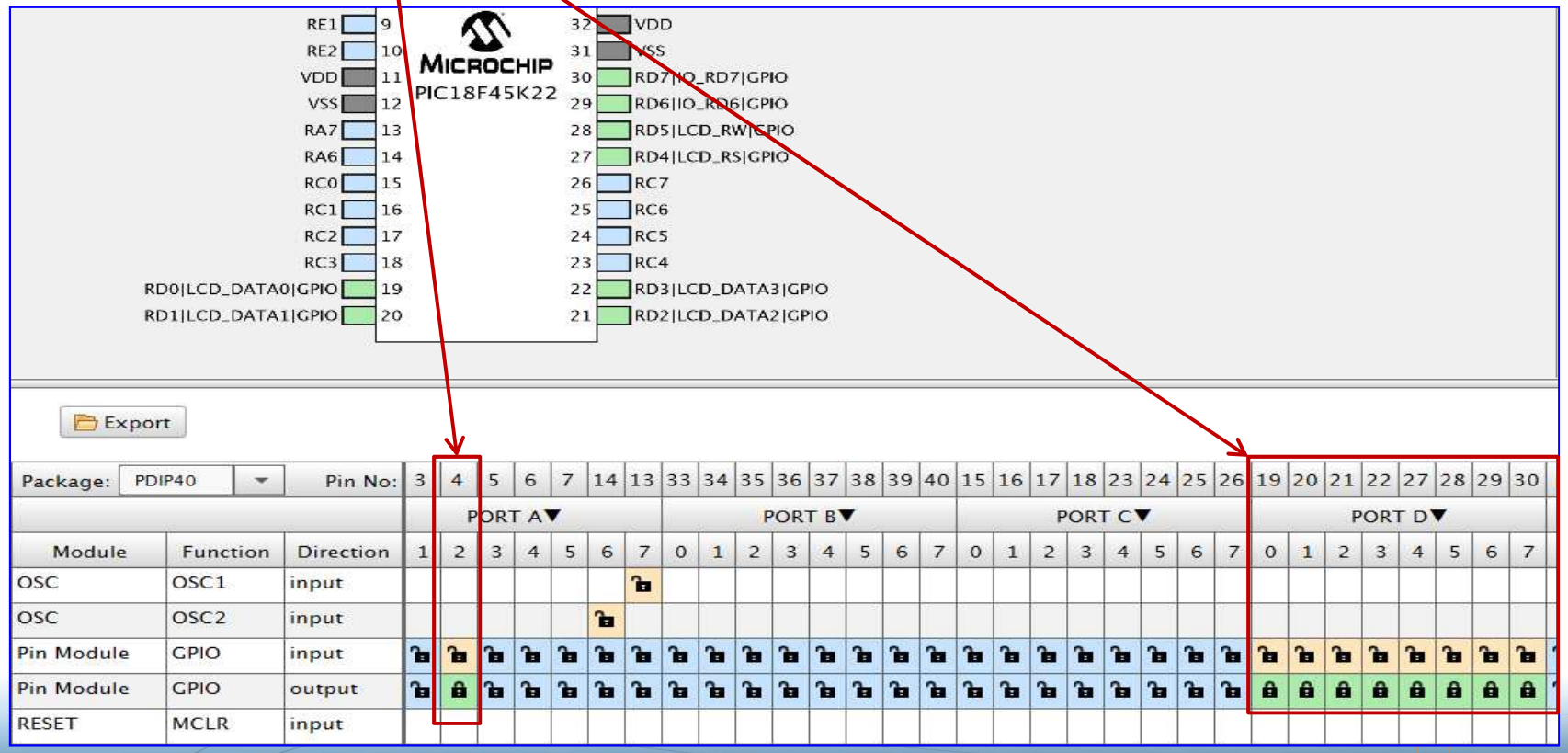
- 利用現成的 **W402T** 教育訓練的 **LCD** 函數加以修改
- 使用 **MCC** 來加入**GPIO** 的腳位設定
  - **RD4** 設為 **LCD_RS**
  - **RD5** 設為 **LCD_RW**
  - **RA2** 設為 **LCD_E**
  - **RD<3:0>** 設為 **LCD_DATA<3:0>**

APP001_LCD.c 所提供的函數

```
void    OpenLCD (void) ;
void    WriteCmdLCD ( unsigned char ) ;
void    WriteDataLCD( unsigned char ) ;
void    putsLCD( char * ) ;
void    putrsLCD( const far char * ) ;
void    putcLCD( unsigned char ) ;
void    puthexLCD( unsigned char ) ;
void    LCD_Set_Cursor( unsigned char , unsigned char ) ;
void    LCD_CMD_W_Timing( void ) ;
void    LCD_DAT_W_Timing ( void ) ;
void    LCD_L_Delay( void ) ;
void    LCD_S_Delay( void ) ;
```

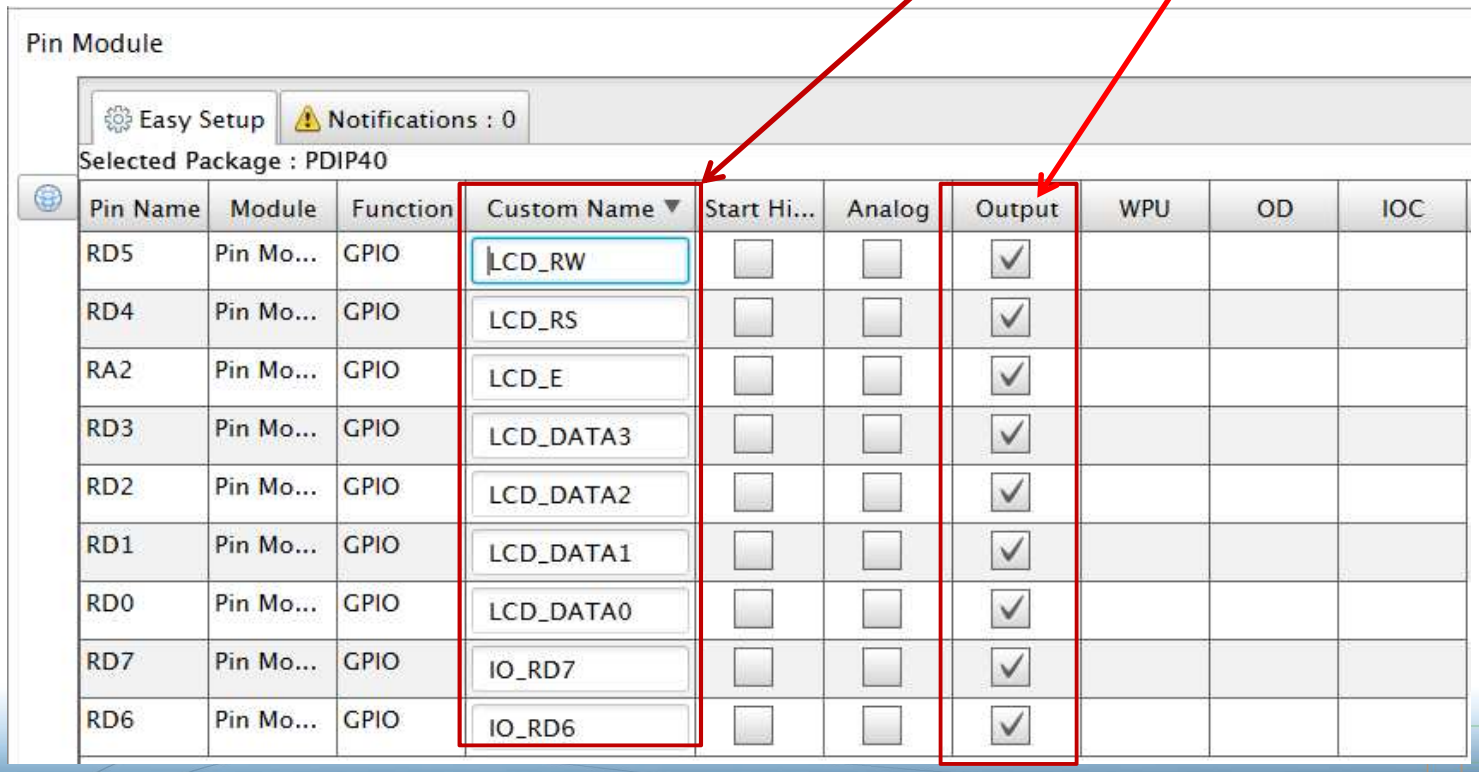
設定 LCD Module 腳位

- 開啟 “ **MCC Pin Manager**” 設定腳位
  - 設定 RA2 為 Digital I/O (綠色上鎖)
  - 設定 PORTD 為 Digital I/O (綠色上鎖)



GPIO 腳位定義

- **Pin Module** 的設定
  - 將目前所選擇到的 I/O 腳都設成 輸出模式，初始輸出設為 Low，關閉類比輸入功能
  - 賦予每個 I/O 腳新的名稱 (Custom Name)



Pin Name	Module	Function	Custom Name	Start Hi...	Analog	Output	WPU	OD	IOC
RD5	Pin Mo...	GPIO	LCD_RW	☐	☐	☑			
RD4	Pin Mo...	GPIO	LCD_RS	☐	☐	☑			
RA2	Pin Mo...	GPIO	LCD_E	☐	☐	☑			
RD3	Pin Mo...	GPIO	LCD_DATA3	☐	☐	☑			
RD2	Pin Mo...	GPIO	LCD_DATA2	☐	☐	☑			
RD1	Pin Mo...	GPIO	LCD_DATA1	☐	☐	☑			
RD0	Pin Mo...	GPIO	LCD_DATA0	☐	☐	☑			
RD7	Pin Mo...	GPIO	IO_RD7	☐	☐	☑			
RD6	Pin Mo...	GPIO	IO_RD6	☐	☐	☑			

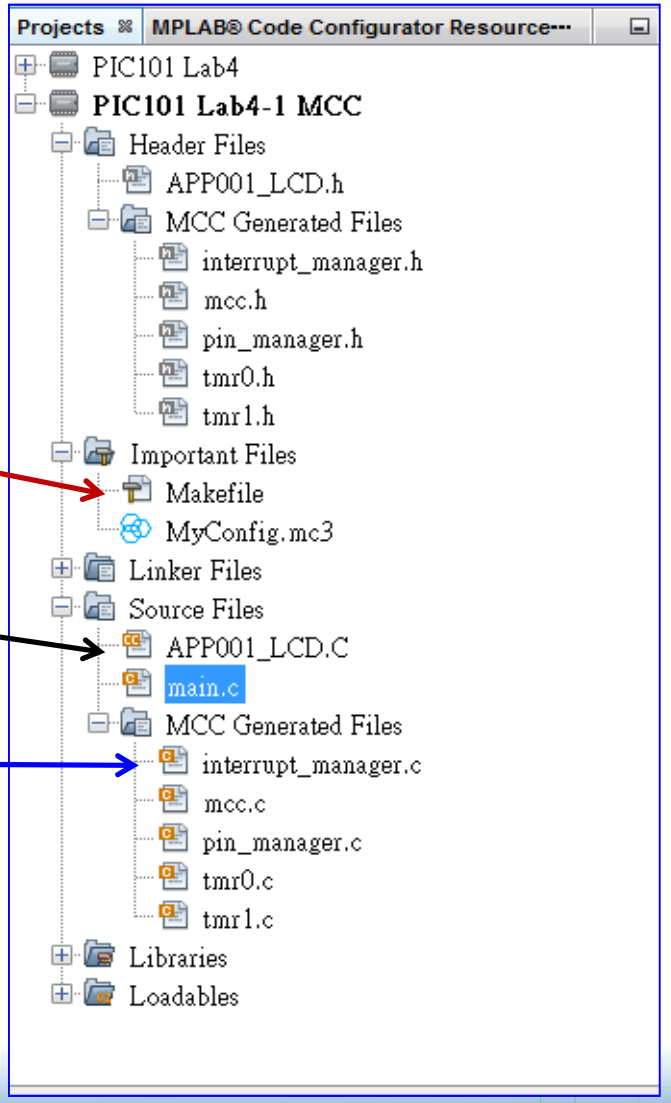
PIC101 Lab4-1 MCC.x 程式產生

- 按下 **“Generate”** 後產生相關的周邊程式如右圖專案所示

Makefile : 專案編譯設定檔
MyConfig.mc3 : **MCC** 配置檔

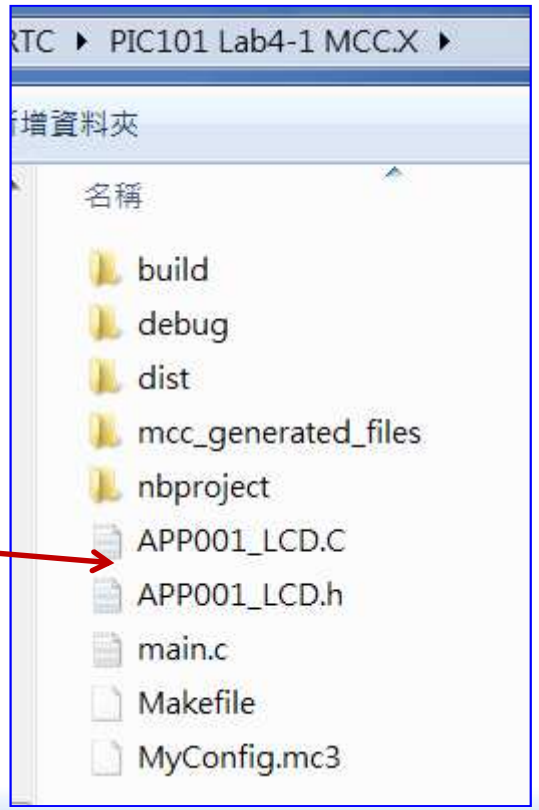
APP001_LCD.C : **LCD Module** 涵式
main.c : 主程式

interrupt_manager.c : 中斷設定函式
mcc.C : 系統頻率及 **Config.**
pin_manager.c : 腳位的規劃及初始設定
tmr0.c : **Timer0** 的函式
tmr1.c : **Timer1** 的函式



關於 h 檔的搜尋路徑

- 所使用到的 **APP001_LCD.c** 裡的 **h** 檔有三種來源路徑，這些是需要設定搜尋路徑的:
  - #include <xc.h>
    - v1.34\include 的內定搜尋路徑
  - #include <..\include\plib\delays.h>
    - 周邊函數庫所使用的標頭檔路徑
  - #include “APP001_LCD.h"
    - 專案目錄下的路徑



還有三步驟要完成

1. 在 **main.c** 開啟中斷 **GIEH & GIEL**
   - // Enable high priority global interrupts
   - INTERRUPT_GlobalInterruptHighEnable(); // 啟動高優先權中斷
   - // Enable low priority global interrupts.
   - INTERRUPT_GlobalInterruptLowEnable(); // 啟動低優先權中斷
2. 在所產生的 **tmr0.c** 裡的 **TMR0_ISR** 涵式裡加上一行 **LED0 (D1)** 的轉態。
   - LATD0 = !LATD0
3. 在所產生的 **tmr1.c** 裡的 **TMR1_ISR** 涵式裡加上一行 **LED7 (D8)** 的轉態。
   - LATD7 = !LATD7;

PIC101 Lab4-1 MCC.x 結論

- 以上我們寫過了傳統周邊使用暫存器或周邊函數庫的寫法
  - PIC101 Lab4.x
- 我們也使用 **MCC** 來設定所需的周邊並產生相關的設定及函數
  - PIC101 LAB4-1 MCC.x

比較一下，何種方式較為好用!

PIC 特殊功能簡介

(附錄 B)

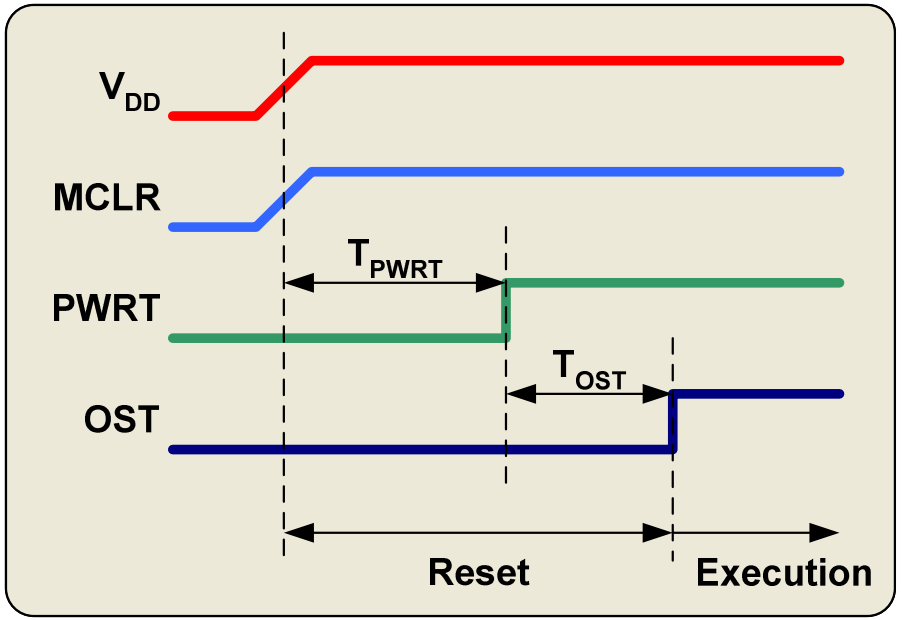
PIC18 震盪器的選擇

XT	Standard frequency crystal oscillator	100kHz - 4MHz
HS	High frequency crystal oscillator	DC - 40MHz
HS+PLL	High frequency crystal with 4x PLL	4MHz - 10MHz
LP	Low frequency crystal oscillator	5kHz - 200kHz
RC	External RC oscillator	DC - 4MHz
RCIO	External RC oscillator, OSC2=RA6	DC - 4MHz
INTRC	Internal RC oscillator	Various
EC	External Clock, OSC2=f_{osc}/4	DC - 40MHz
ECIO	External Clock, OSC2=RA6	DC - 40MHz

- 可供選擇的工作頻率設定，提供一高彈性的運作
  - LP 低頻率工作模式，可提供低功耗操作模式
  - RC or INTRC 提供低成本的震盪模式
  - XT 對於最常用的振盪器頻率做最佳化
  - HS 驅動高頻率振盪器做最佳化
- 表上的工作頻率範圍僅做為設計參考準則

POR, OST, PWRT

- **POR: Power On Reset**
  - With MCLR tied to V_{DD}, a reset pulse is generated when V_{DD} rise is detected
- **PWRT: Power Up Timer**
  - Device is held in reset for 72ms (nominal) to allow V_{DD} to rise to an acceptable level (after POR only)
- **OST: Oscillator Start-up Timer**
  - Holds device in reset for 1024 cycles to allow crystal or resonator to stabilize in frequency and amplitude; not active in RC modes; used only after POR or Wake Up from SLEEP



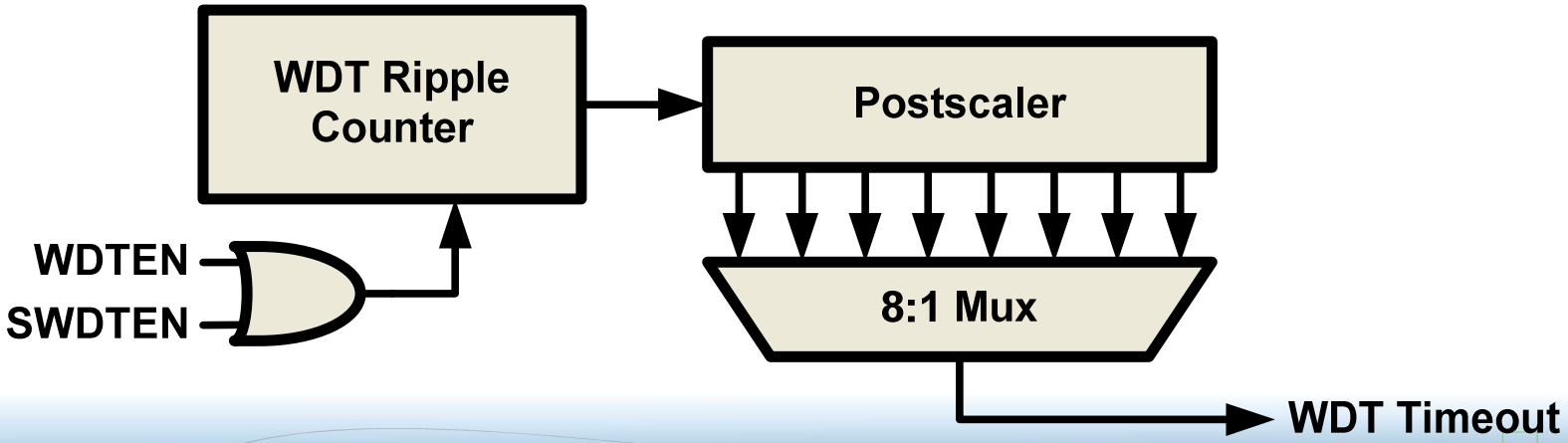
Sleep Mode

- 當微處理器執行 **SLEEP** 指令後將會關閉主震盪器，進入省電模式
  - 系統主震盪器停止震盪
  - 處理器的狀態將保持不變 (靜態設計)
  - 看門狗繼續運作(假如在開啟模式)
  - MCU 工作電流將降至 0.1uA 以下

可以使用於在 Sleep Mode 喚醒 CPU 的周邊來源	
MCLR	**Master Clear Pin Asserted (pulled low)**
WDT	**Watchdog Timer Timeout**
INT	**INT Pin Interrupt**
TMR1	**Timer 1 Interrupt (or also TMR3 on PIC18)**
ADC	**A/D Conversion Complete Interrupt**
CMP	**Comparator Output Change Interrupt**
CCP	**Input Capture Event**
PORTB	**PORTB Interrupt on Change**
SSP	**Synchronous Serial Port (I^2C Mode) Start / Stop Bit Detect Interrupt**
PSP	**Parallel Slave Port Read or Write**

Watchdog Timer

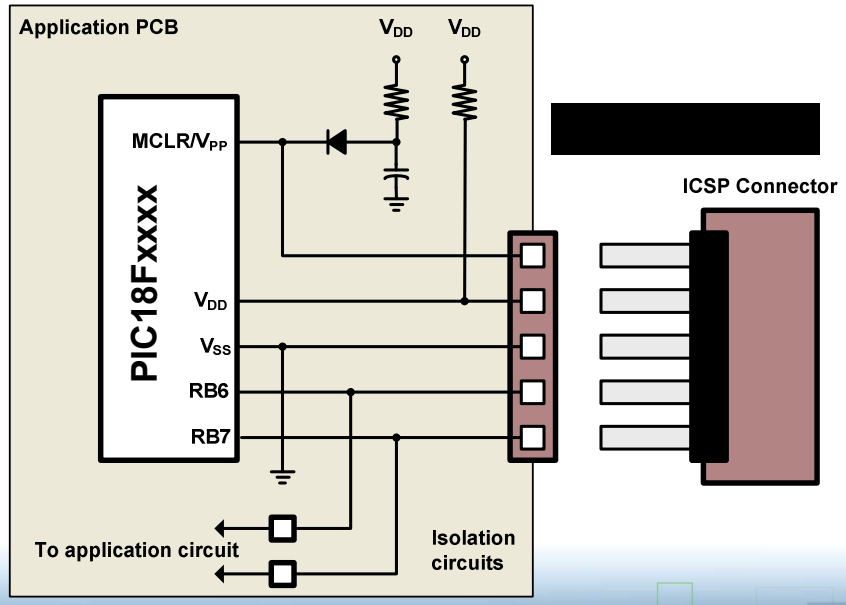
- Helps recover from software malfunction
- Uses its own free-running on-chip RC oscillator
- WDT is cleared by CLRWDT instruction
- Enabled WDT (WDTEN) cannot be disabled by software
- WDT overflow resets the chip
- Programmable timeout period: 18ms to 3.0s typical
- Operates in SLEEP; on time out, wakes up CPU



In-Circuit Serial Programming™

- 除了電源與 **Vpp(MCLR)** 接腳外，只需要另外兩個接腳即可進行對 **PIC18 device** 的燒錄與除錯
  - ICSP Data
  - ICSP Clock
- 也可以很方便地進行對下列的記憶體做 **In-System Programming**
  - Calibration Data
  - Serialization Data
- **MPLAB® ICD3 ，PICKit 3 ，Real ICE** 等燒錄/除錯器都可以使用此種方式為 **PIC18 device** 燒錄程式並進行除錯

Pin	Function
V_{PP}	Programming Voltage = 13V
V_{DD}	Supply Voltage
V_{SS}	Ground
RB6	Clock Input
RB7	Data I/O & Command Input



BOR – Brown Out Reset

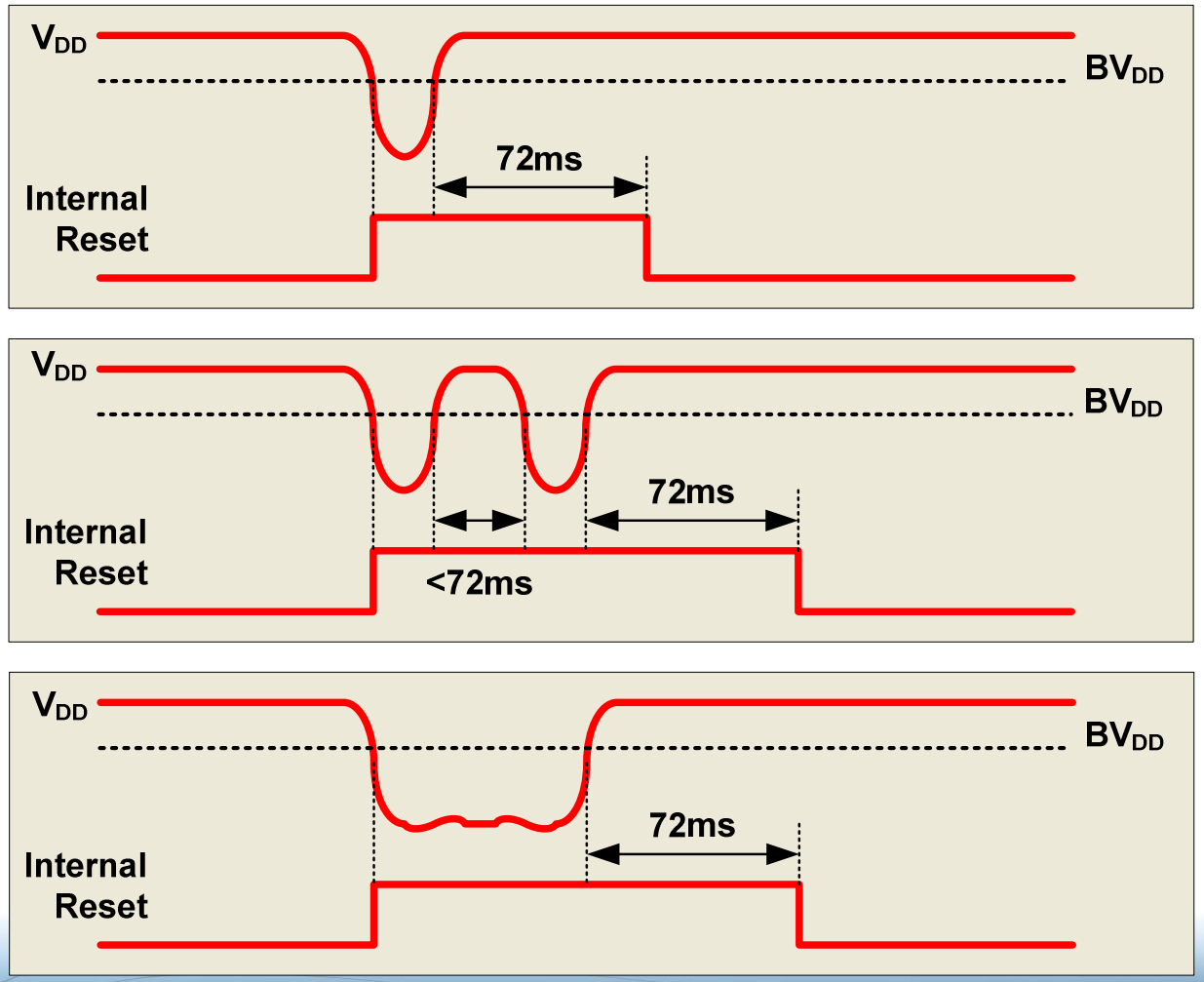
- 當工作電壓掉至所設定的門檻值時，**MCU** 將會進入重置模式
- 防止不穩定的或意外操作
- 無需外加 **BOR** 的電路降低成本
- **BOR** 的消耗電流在 **Sleep Mode** 下的控制
- 為系統穩定，一般使用建議啟用 **BOR**

PBOR
Programmable Brown Out Reset

- **Configuration** 的選項 **(**在燒錄 **IC** 時一併被設定**)**
  - 有些元件可以使用軟體來 enabled / disabled
- 有四個可選擇的 **BV_{DD}** 觸發位準選項 **: (**不同 **device** 間可能有部同的規格**) PIC18F45K22** 為例**:**
  - BORV<1:0> = 11 為 1.9V
  - 2.2V
  - 2.5V
  - BORV<1:0> = 00 為 2.85V
- 如果 **device** 內設的臨界值未能符合需求**,** 可以使用外部的 **supervisor (MCP1xx, MCP8xx/TCM8xx, or TC12xx)**

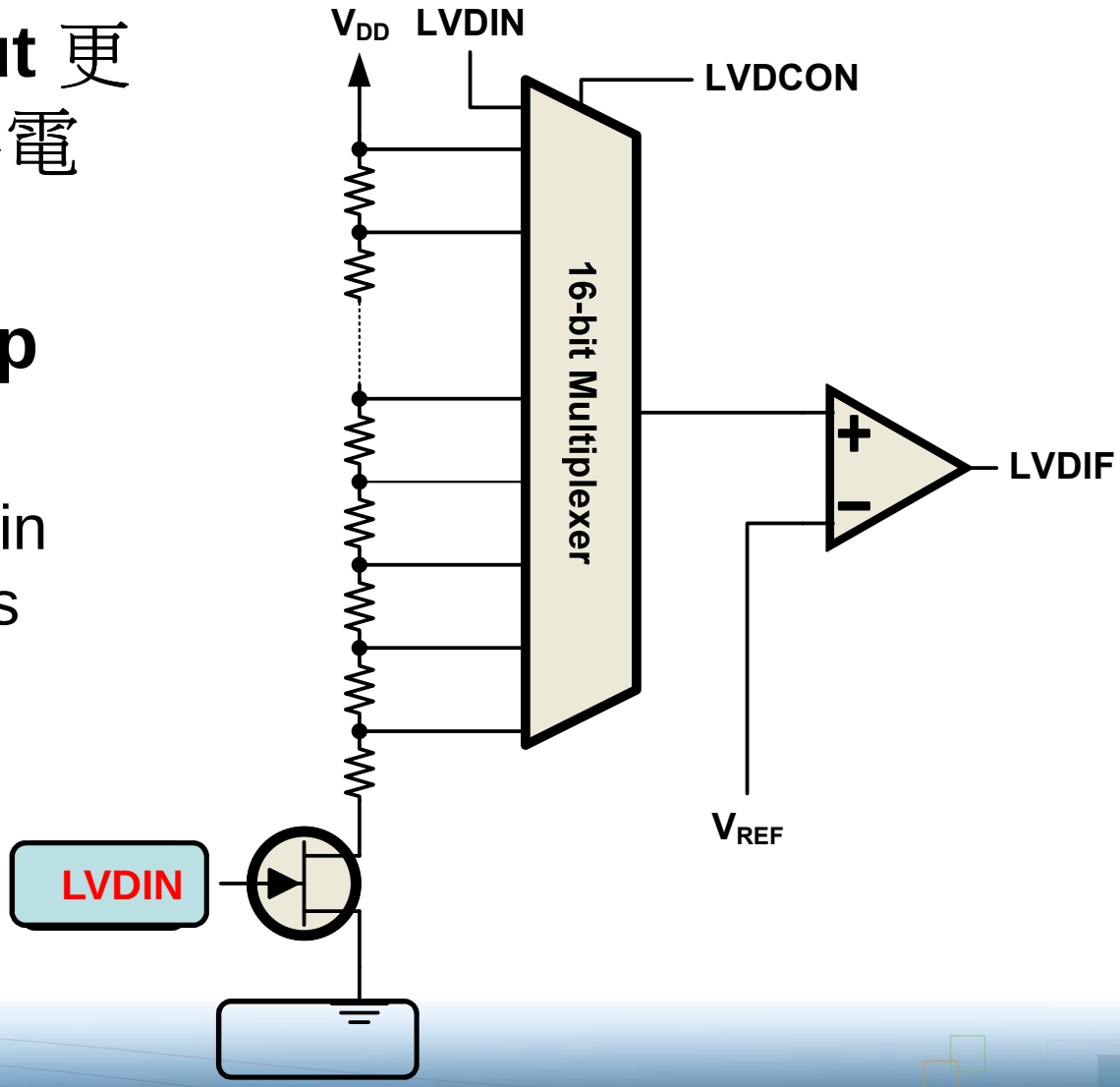
PBOR – Programmable Brown Out Reset

- Holds PICmicro® MCU in reset until ~72ms after V_{DD} rises back above threshold



PLVD – Programmable Low Voltage Detect

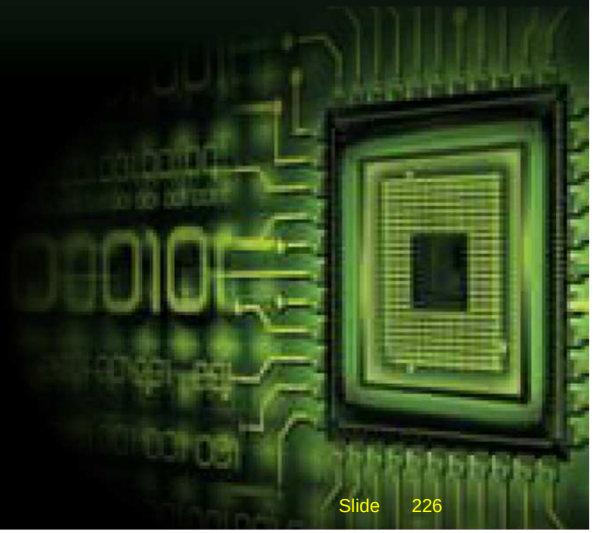
- 可以比 **brown out** 更早對 **device** 做出電源異常警示
- **16 selectable trip points:**
  - 1.8V up to 4.5V in 0.1 to 0.2V steps
  - External analog input
- **Internal V_{REF}**



MICROCHIP

Regional Training Centers

Thank You



MICROCHIP
Regional Training Centers

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KeeLoq, KeeLoq logo, MPLAB, PIC, PICmicro, PICSTART, PIC^{32} logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.