MICROCHIP

Regional Training Centers

Normal & Vocational School Training

Microchip PIC18F Family

MICROCHIP
Regional Training Centers

Tools Requirement

- **Software**
  - MPLAB X IDE v2.30 (Or New)
  - MPLAB C Compiler C18 v3.40 Lite Mode (Or New)
- **Hardware**
  - MPLAB PICKit 3
  - APP001/APP001T Rev.3 EVM
  - MCU PIC18F4550-I/P
- **Reference**
  - MPLAB C18 Compiler User's Guide & Library
  - APP001 Rev.3 User Manual

- You can download all at microchip Taiwan website.
  - www.microchip.com.tw

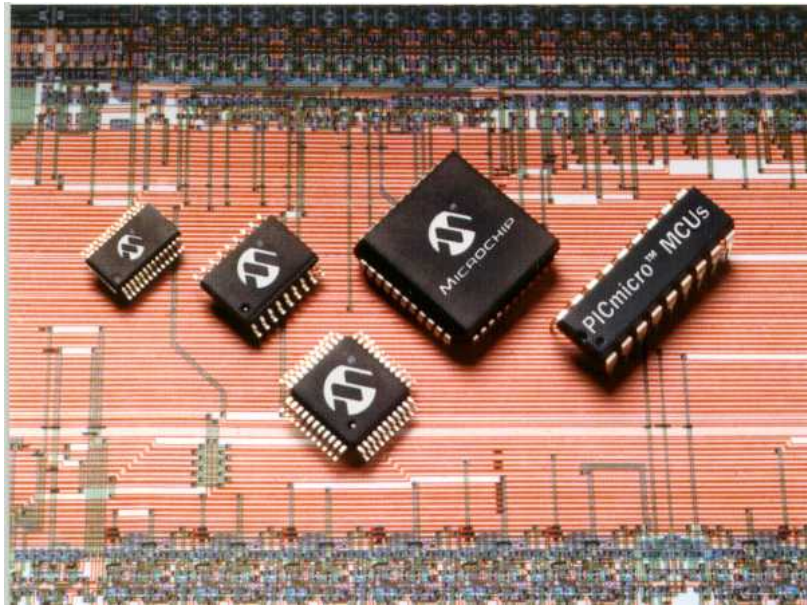
MICROCHIP

Regional Training Centers

PIC18F Family Architecture

PIC18 Family 架構

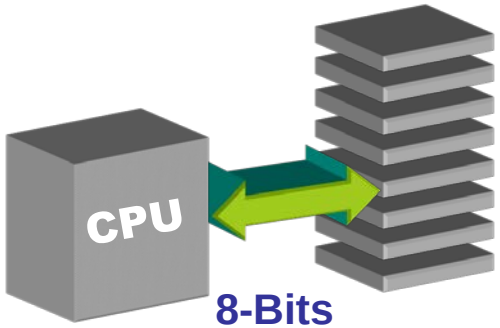
- The high performance of the PICmicro® microcontroller can be attributed to the following architectural features:
  - Harvard Architecture
  - Instruction Pipelining
  - Large Register File
  - Single Cycle Instructions
  - Single Word Instructions
  - Long Word Instructions
  - Reduced Instruction Set
  - Orthogonal Instruction Set



PICmicro® 的架構

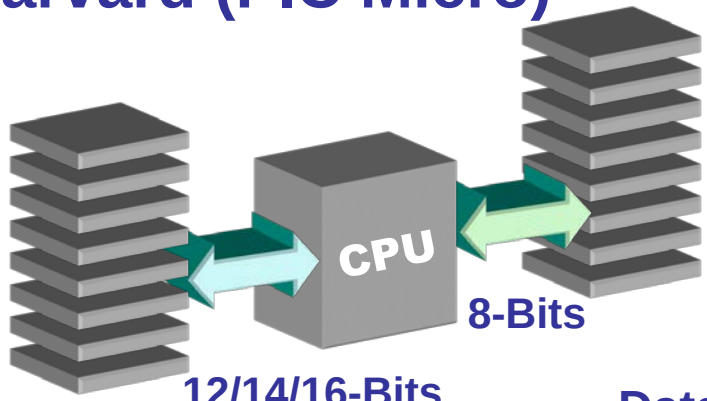
MICROCHIP
Regional Training Centers

Von Neumann (一般MCU)



Program 以及 Data Memory

Harvard (PIC Micro)

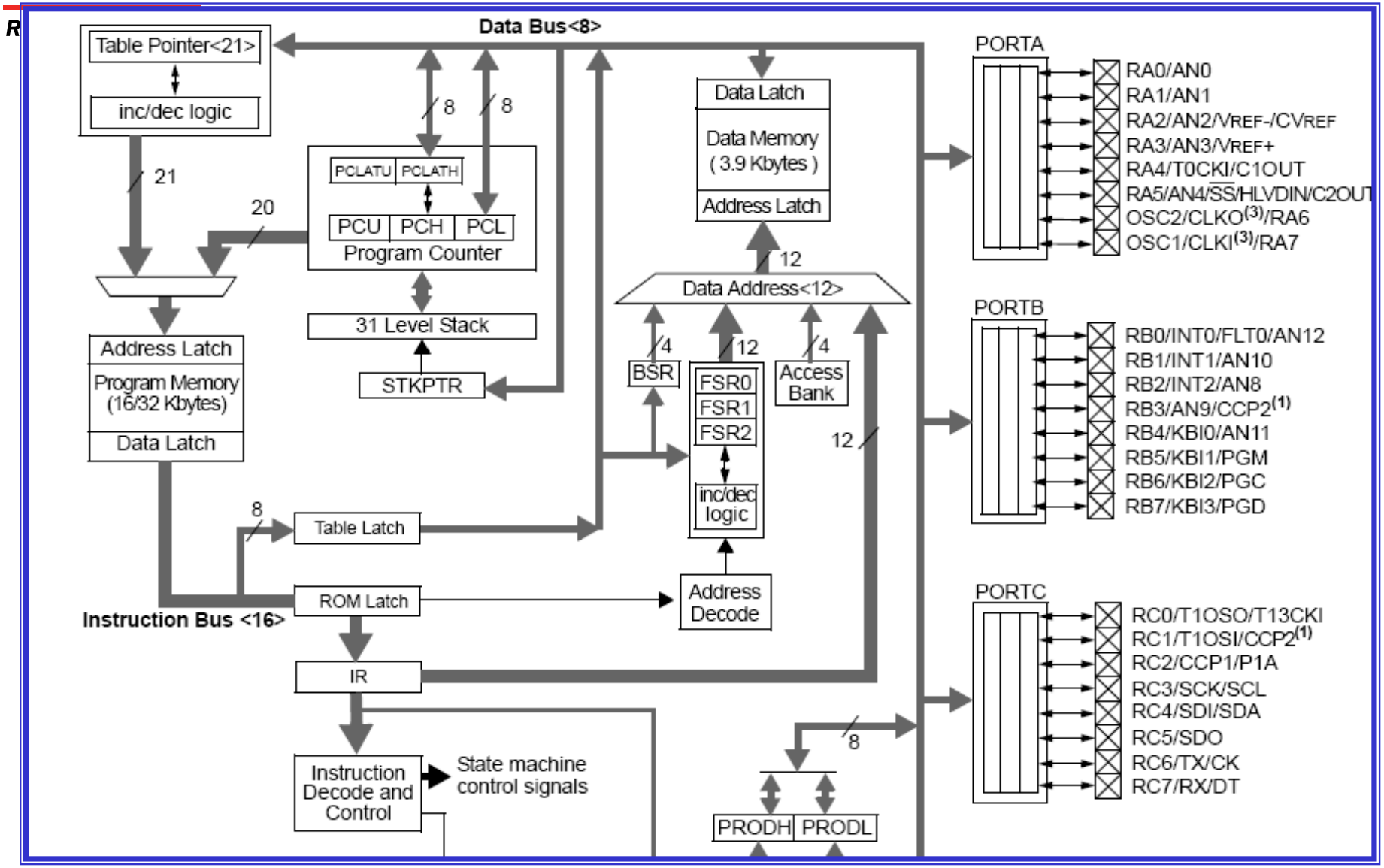


Program Memory

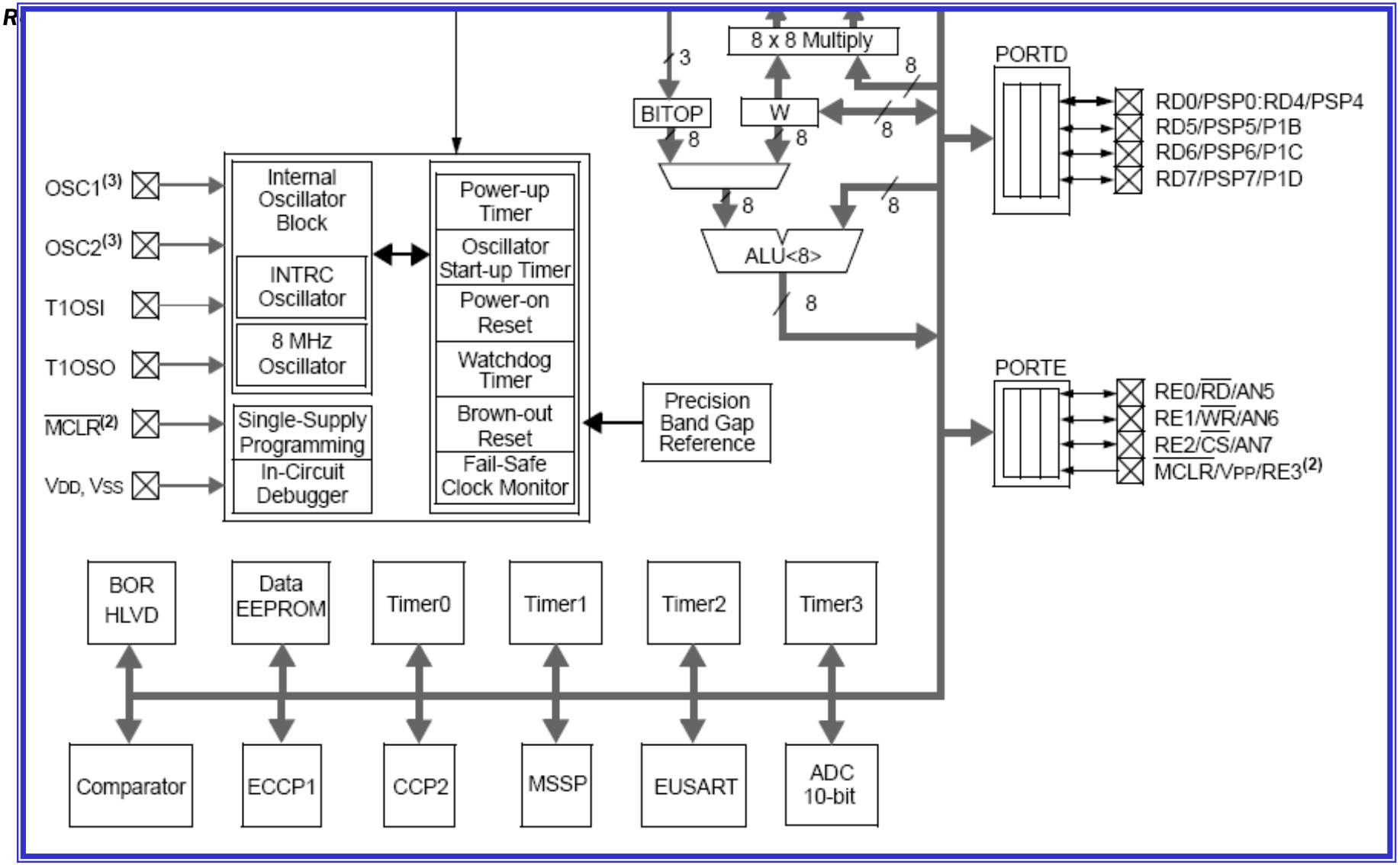
Data Memory

- 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
- 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

PIC18F4520 架構圖(一)

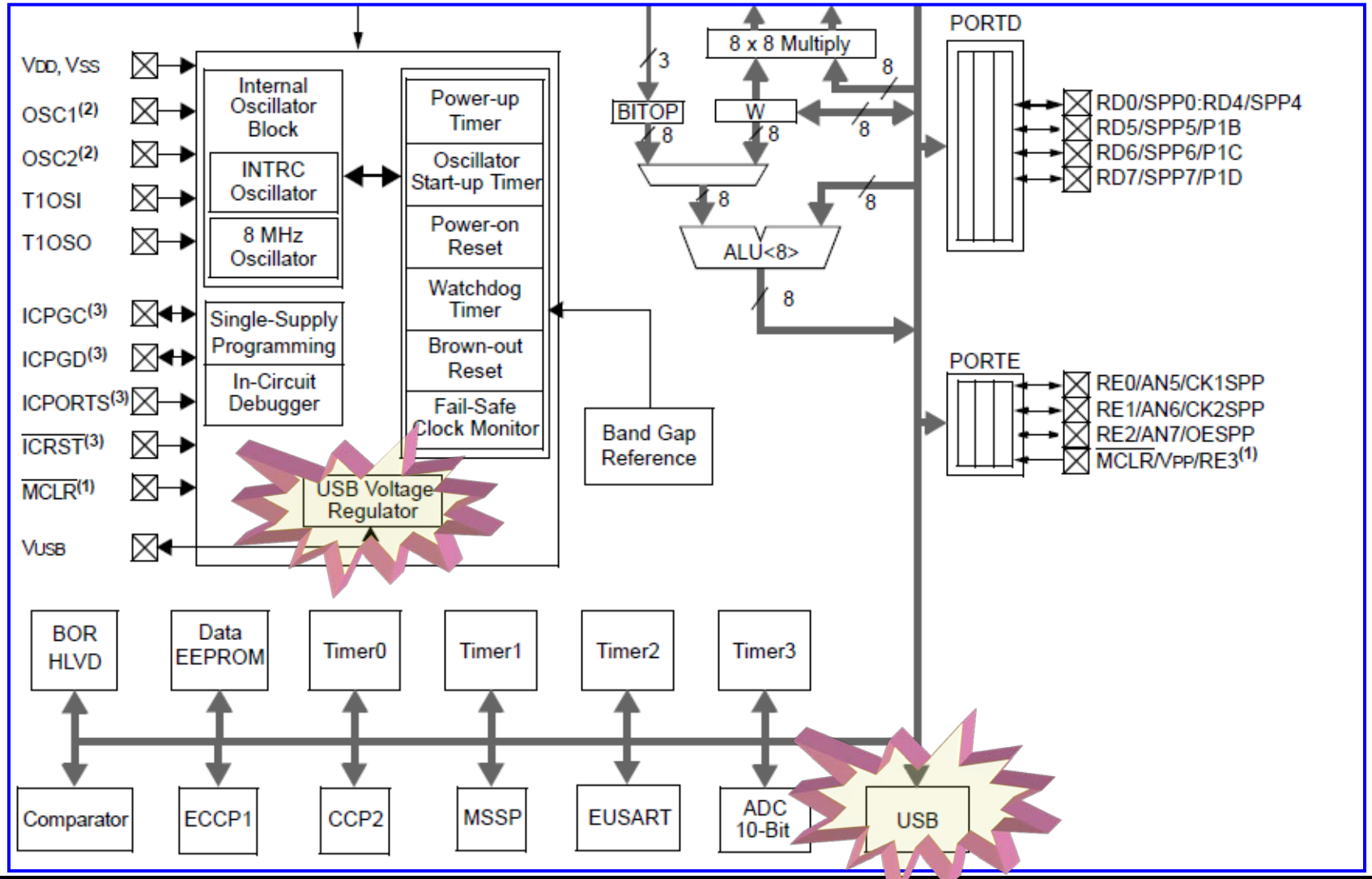


PIC18F4520 架構圖(二)

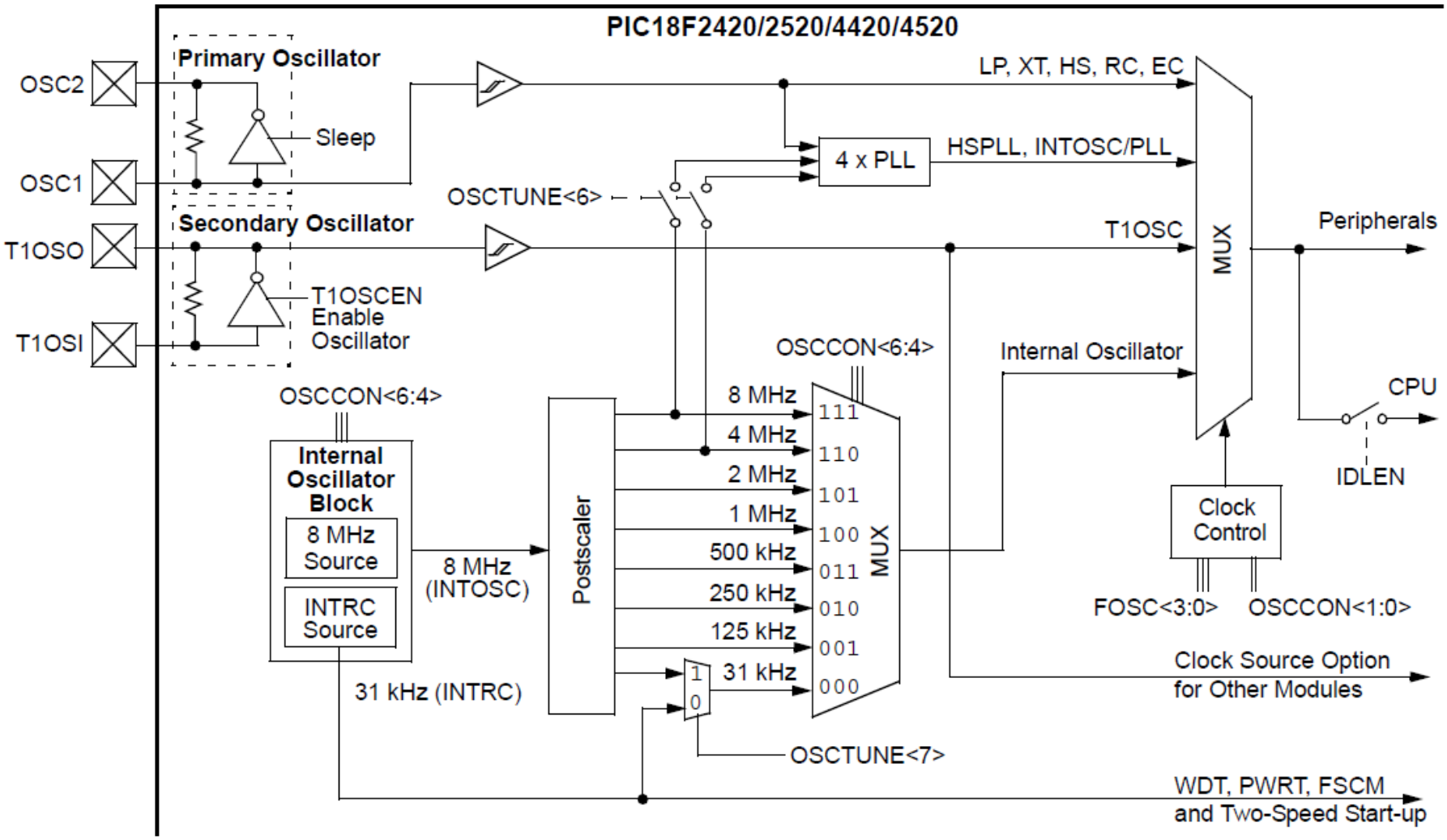


MICROCHIP
Regional Training Centers

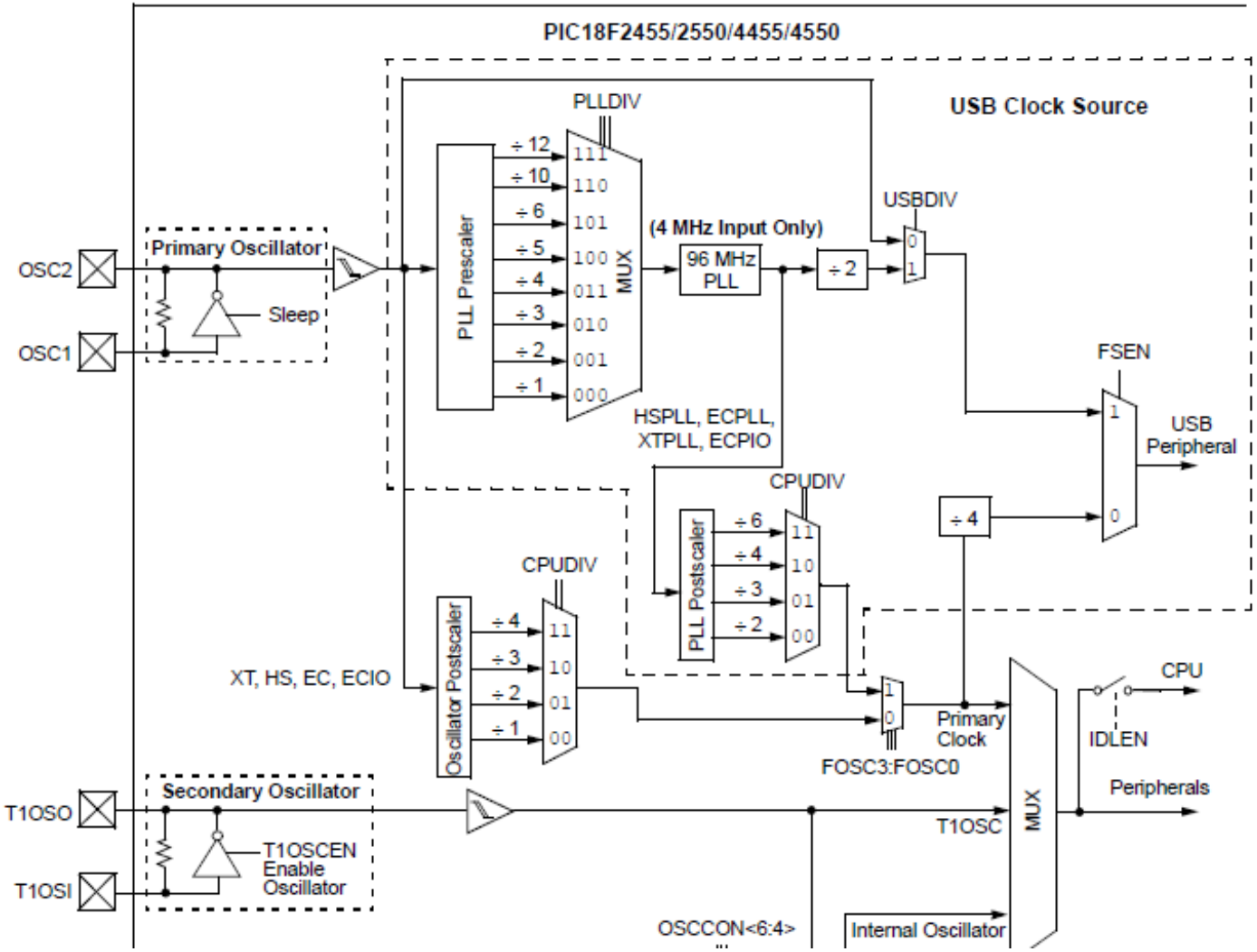
PIC18F4550 與標準 PIC18F4520 之差異



PIC18F4520 振盪器架構

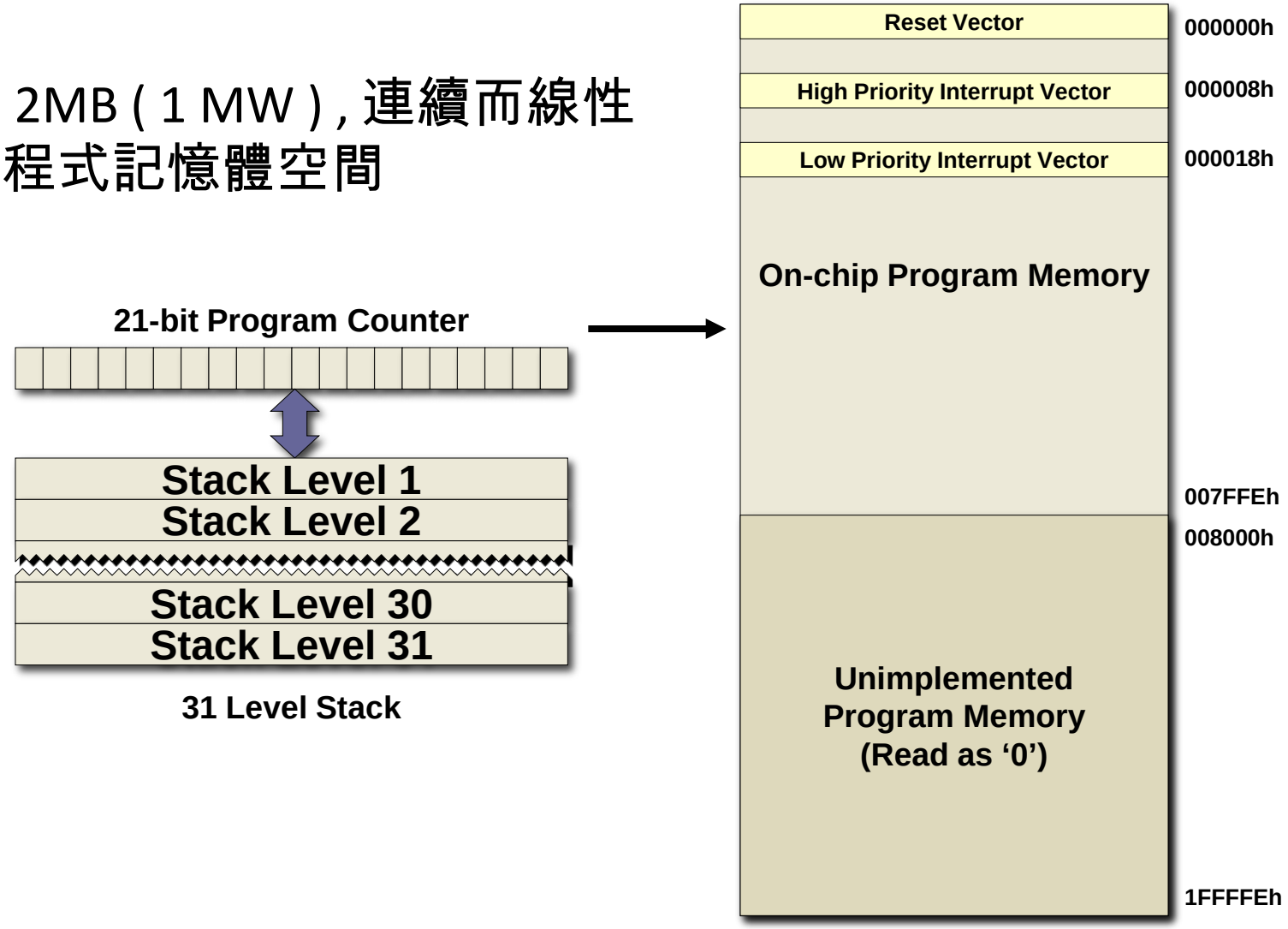


PIC18F4550 振盪器架構



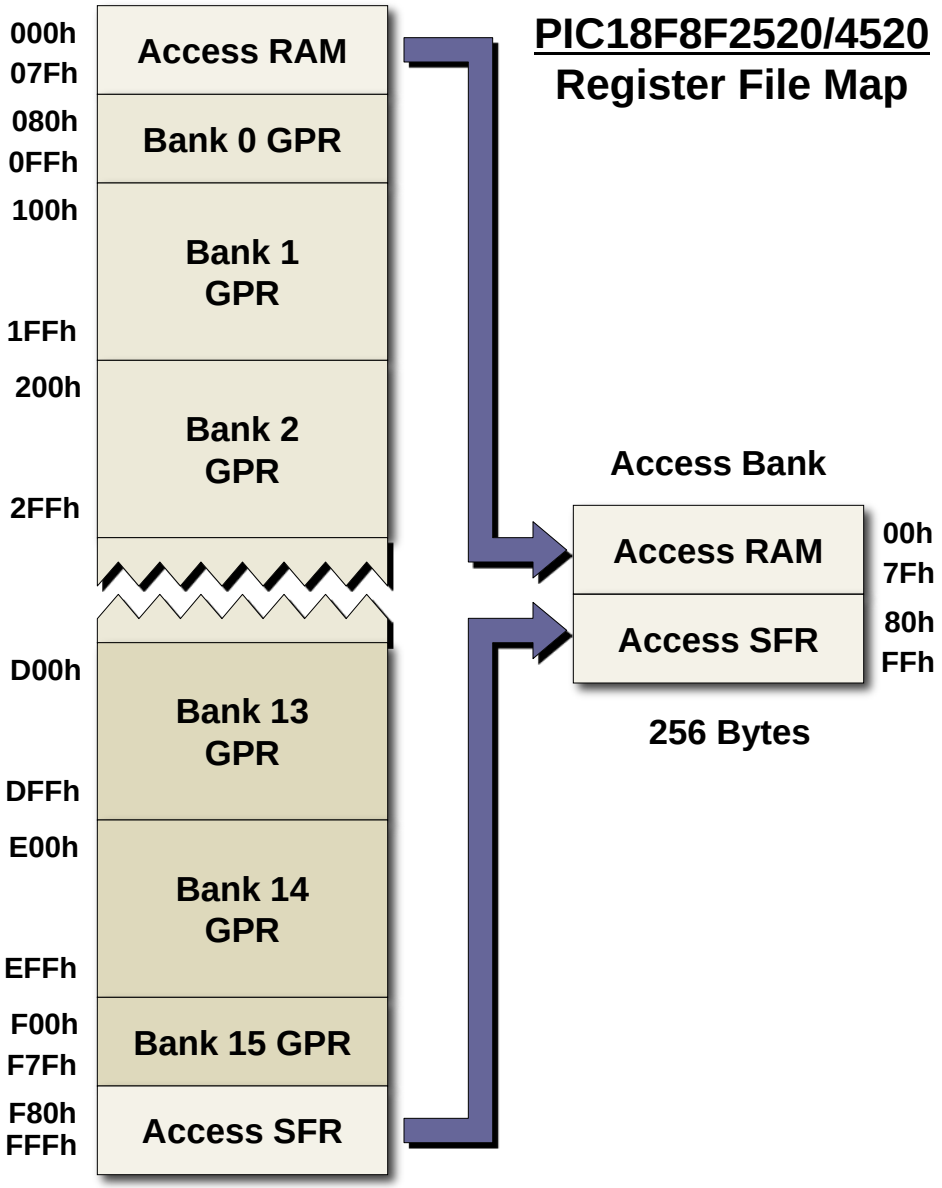
PIC18 Family 程式記憶體架構

- 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
- 可以使用特定指令做線性的存取(需要兩個 Instruction words)
- 也可以 256 byte 為單位,使用 bank 的方式來定址 Data Memory(只需一個 Instruction word 但要配合 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(Access Bank). 在特定指令中指定使用 Access Bank 時,不論 BSR 指向何處都會 Access 此 bank !



MICROCHIP

Regional Training Centers

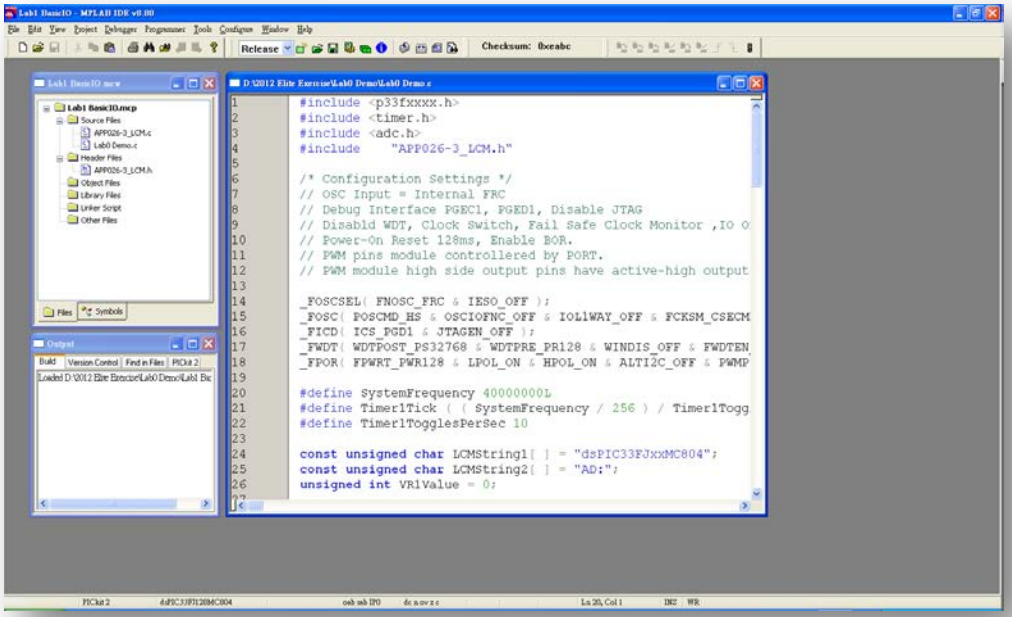
Development Tools

MPLAB IDE & MPLAB X IDE

MICROCHIP
Regional Training Centers

MPLAB® IDE

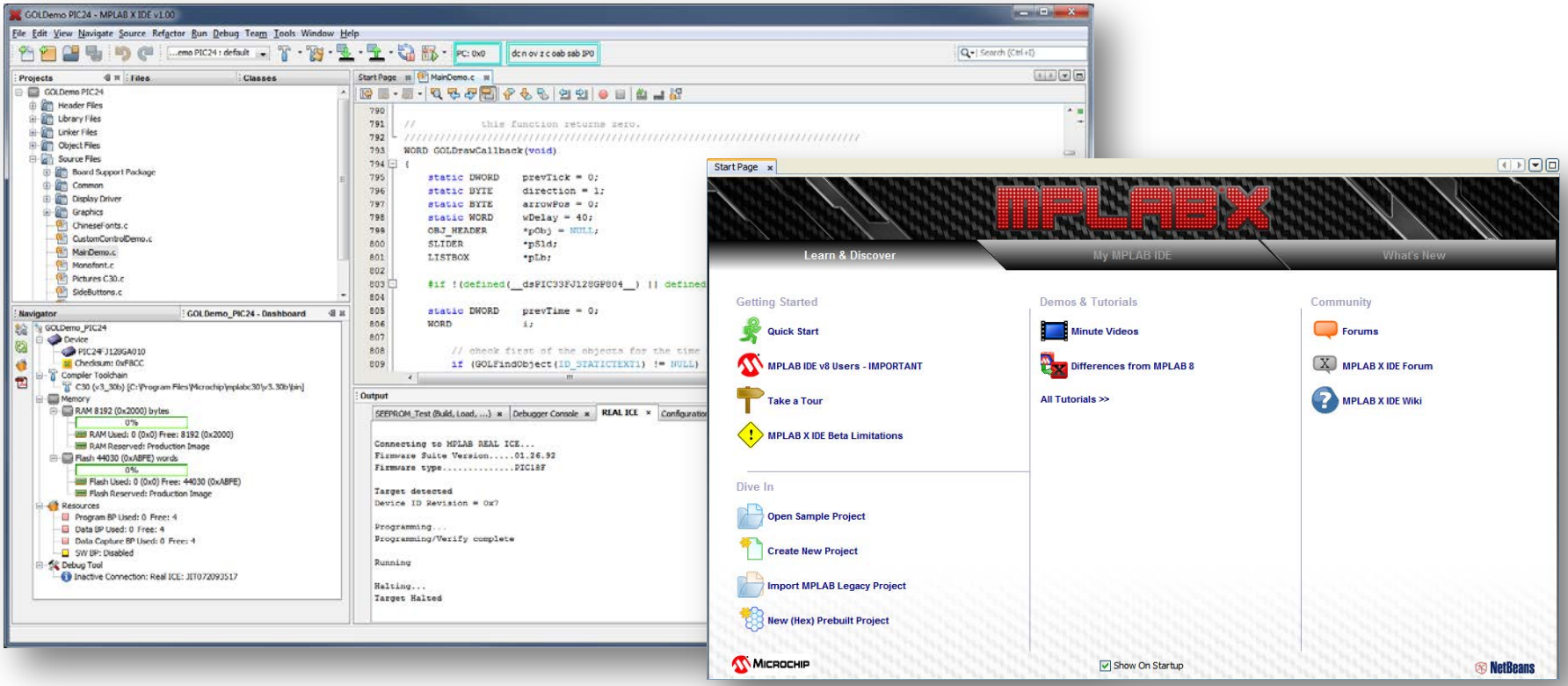
- Microchip提供的整合式開發環境, 支援全系列8-Bits, 16-Bits及32-Bits的MCU。所有的MCU都可透過相同的環境開發。目前最新的版本是v8.92。
- 可整合各式的組譯器/編譯器(MPLAB C, Hi-TECH C, etc.),開發工具(PICkit3, ICD3, Real ICE, etc..) 。



MICROCHIP
Regional Training Centers

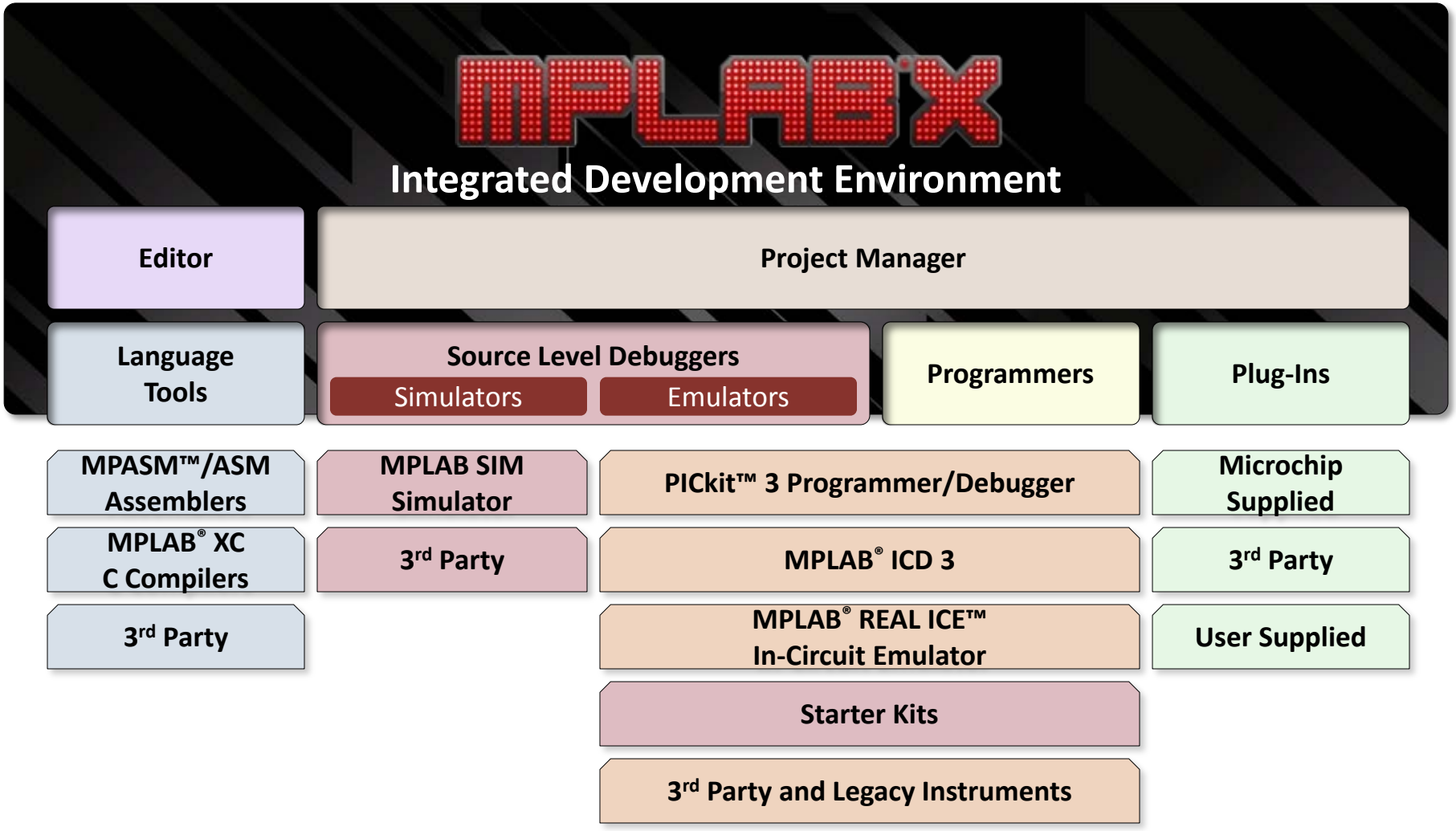
MPLAB® X IDE

- 新一代的整合式開發環境, 可支援全系列MCU, 擁有許便利的功能與特徵。Java Base, 可以跨平台, 目前最新的版本是v3.05。
- 不支援ICD2, ICE2000/4000, ProMATE II, PICStart Plus.



MICROCHIP
Regional Training Centers

MPLAB® X IDE Overview



MICROCHIP
Regional Training Centers

MPLAB® PICkit™ 3

Debugger/Programmer Probe

- Full-speed real-time emulation with run/halt, step and breakpoints
- Firmware upgradable via the free MPLAB® IDE
- USB 2.0 Full Speed communications for fast downloads
- USB powered with target power, up to 30mA
- VDD range 3 - 5.5V
- VPP range 3 - 13V
- CE and RoHS-compliant

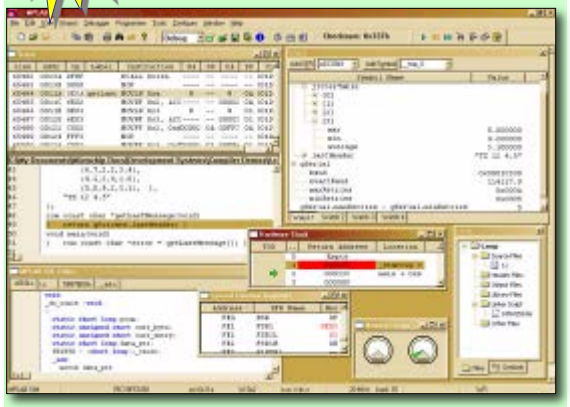
MPLAB® CERTIFIED

容易入門的開發環境及工具安裝

MICROCHIP
Regional Training

Download the MPLAB® IDE (Free)

Free



+

Select a Debugger/ Programmer

+

Select Your C Compiler

Free

MPLAB® CERTIFIED

MPLAB® C Compilers
Lite & Standard Versions
Support for PIC18, PIC24, dsPIC & PIC32

HI-TECH C COMPILERS by Microchip Technology

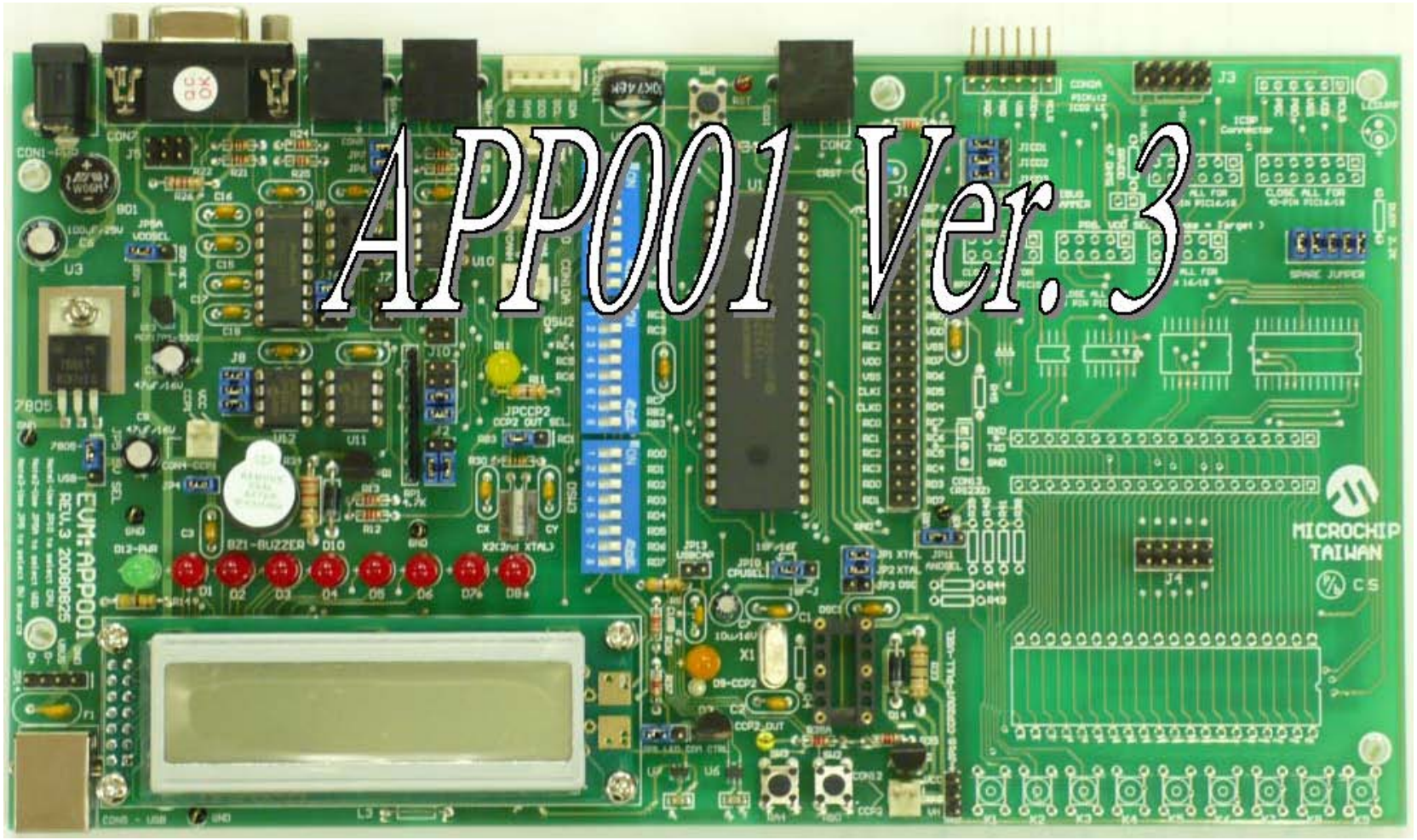
Free

- HI-TECH C Lite
- HI-TECH C STD
- HI-TECH C PRO

Support for all PIC® MCUs

MICROCHIP
Regional Training Centers

EVM:APP001 Rev.3



MICROCHIP

Regional Training Centers

Development Tools

C Compiler : MPLAB C18

MICROCHIP
Regional Training Centers

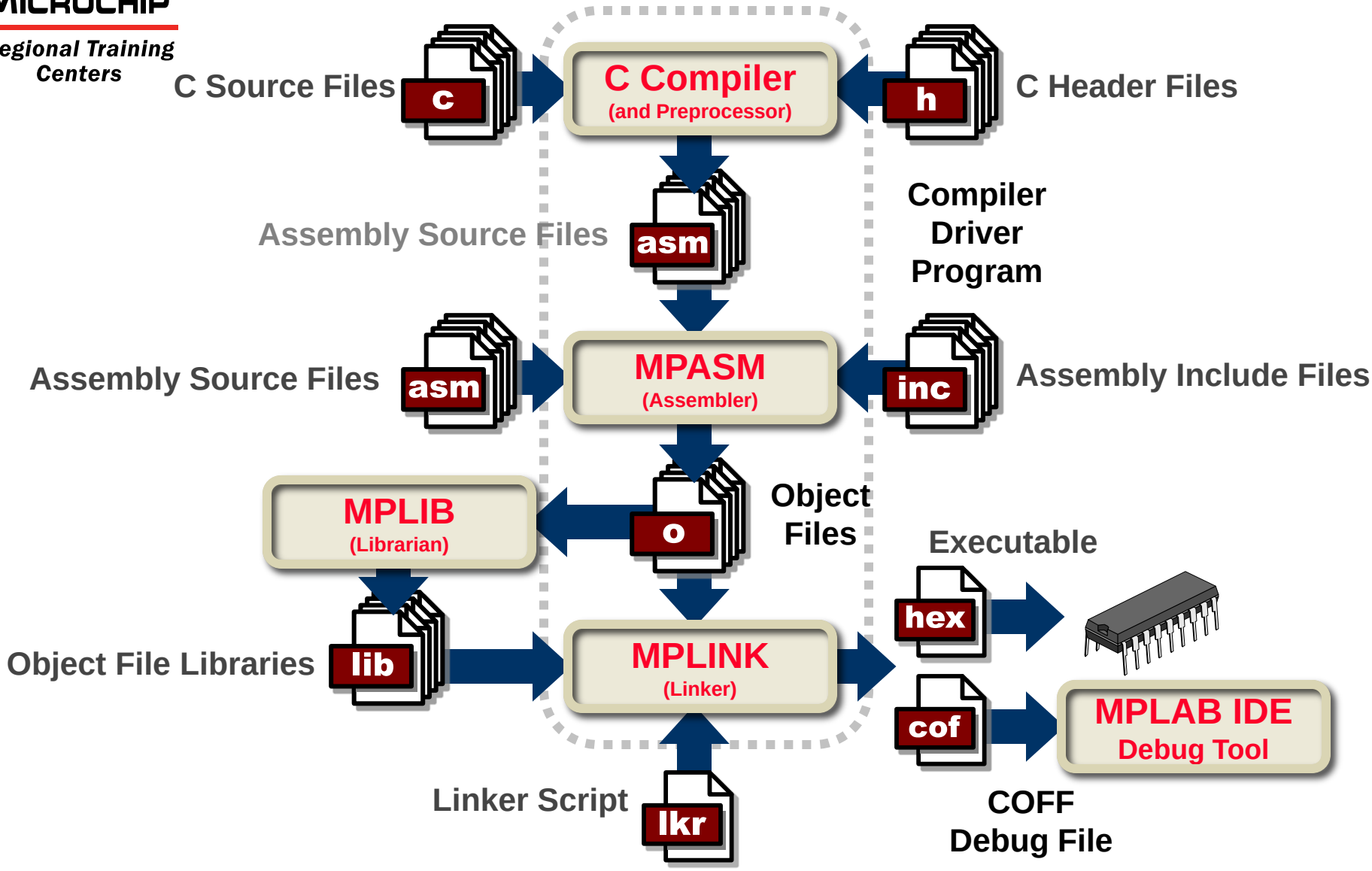
MPLAB® C for PIC18

Overview

- ANSI x3.159-1989 compliant except when standard conflicts with efficient PIC® MCU support
- Optimizing compiler
- Includes language extensions for PIC18
- Works as a component of MPLAB® IDE
- Compatible with MPASM object modules

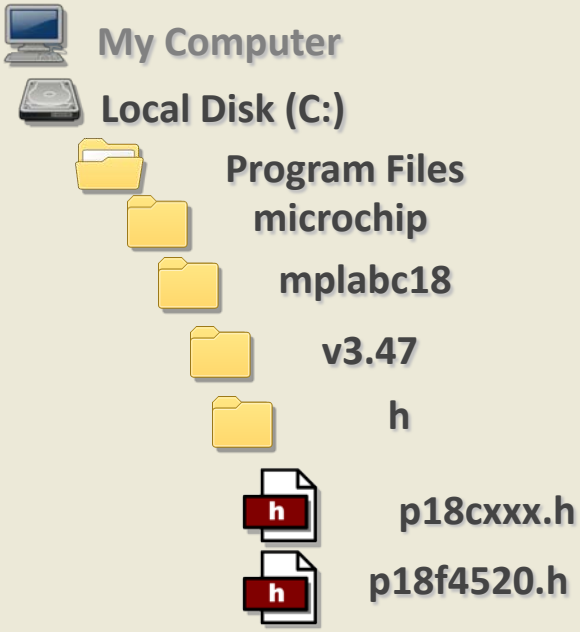
MPLAB® C for PIC18 Lite /Evaluation version are available for free from the Microchip web site.

Development Tools Data Flow



MPLAB® C for PIC18

Header Files



Header files are included as part of the MPLAB C installation:

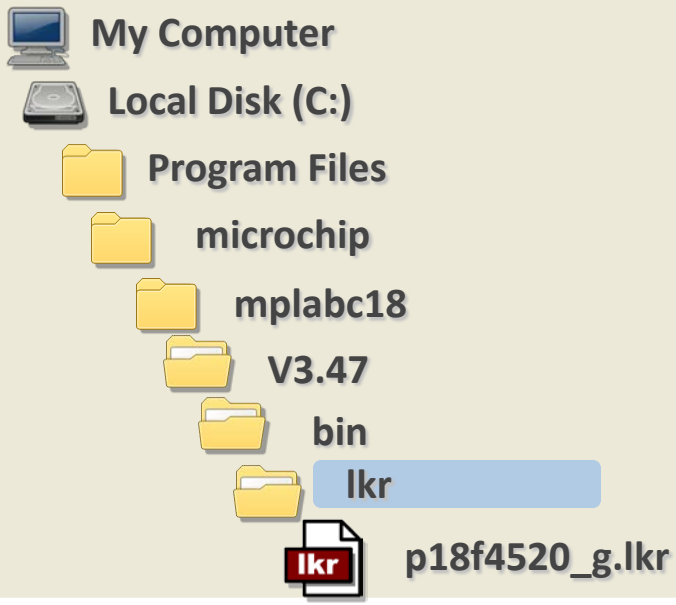
p18cxxx.h

p18cxxx.h is a generic header file which automatically includes processor specific header files based on project settings

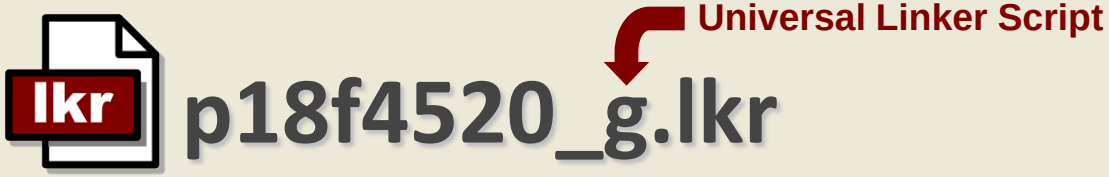
- One header file per device:
  - Provides access to registers as C variables
  - Defines labels for bit manipulation
  - Defines macros to utilize instructions not normally accessible from C

MPLAB® C for PIC18

Linker Scripts



Linker Script files are included as part of the MPLAB installation :



- One linker script file per device:
  - Defines memory sections and boundaries
  - Defines both debug and release modes
- Automatically selected and used by MPLAB unless specified in project tree

MICROCHIP

Regional Training Centers

Tools Installation

MPLAB X IDE & MPLAB C18

MICROCHIP
Regional Training Centers

Tools Installation

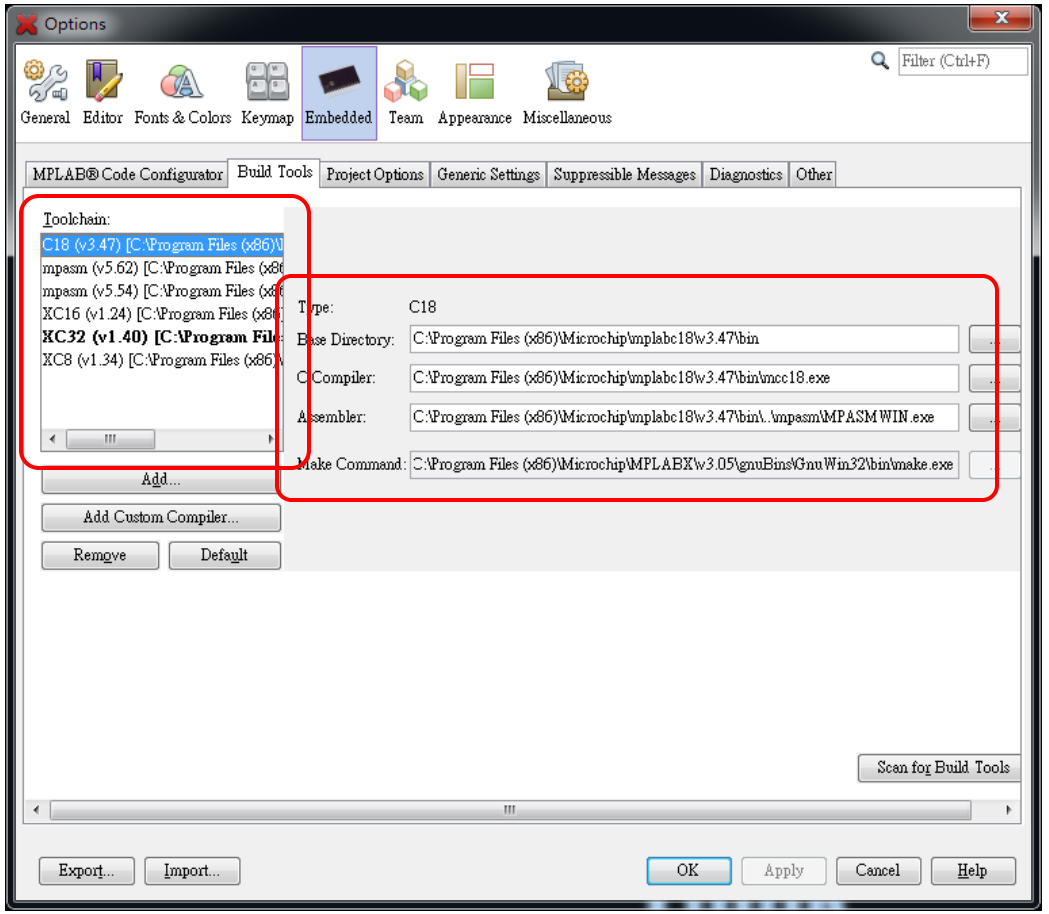
- 使用 Microchip RTC 教育訓練光碟或至以下連結安裝所需軟體
  http://www.microchip.com.tw/Data_CD/
- MPLAB IDE v3.05
  MPLAB C18 v3.47

開發軟體, 編譯器 (www.microchip.com/developmentools)	
MPLAB® X IDE	v3.05 Windows(Local)
	Windows Version Linux Version Mac Version *Detail Info.*
MPLAB® IDE	v8.92(Local)
MPLAB® XC8	v1.34(Local)
MPLAB® XC16	v1.24(Local)
MPLAB® XC32	v1.40(local) v1.34(local)
MPLAB® C18 Lite	v3.47(Local)
MPLAB® C30 Lite	v3.31(Local)
MPLAB® C32 Lite	v2.02a(Local)
HI-TECH C for PIC10/12/16	v9.83(Local)
HI-TECH C for PIC18	v9.80(Local)

MICROCHIP
Regional Training Centers

Install Status Check

- 安裝完成後, 可以使用MPLAB X IDE功能表中的Tools\Options來確認。
- 先選擇Tools\Options, 然後點選 “Embedded”, 再選擇 “Build Tools”, 確認 C18 已正確安裝。



MICROCHIP

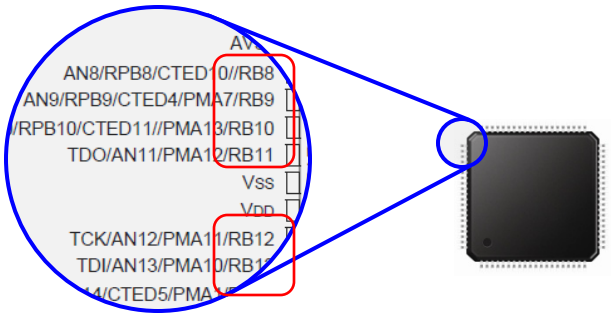
Regional Training Centers

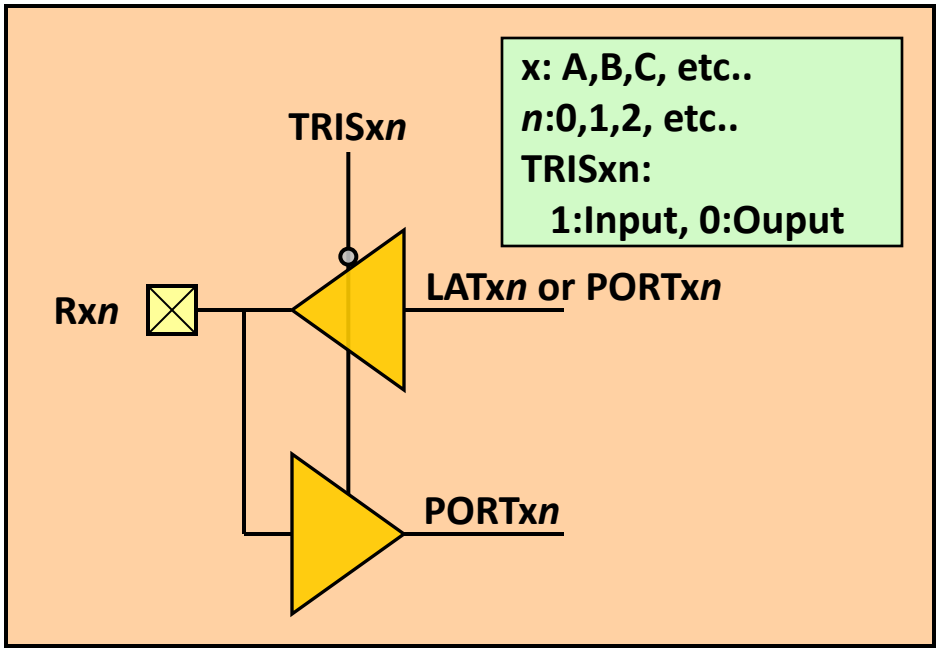
PIC18 GPIO Review

** Exercises, First Project & LED Control*

I/O Port Block Diagram

- PIC18的IO Port示意圖, 如圖所示。
- 所有的IO Port都具有TRIS*x*, LAT*x*跟PORT*x*三個特殊功能暫存器。TRIS*x*用來設定輸出或輸入。
  設定為"0", 表示輸出;設定為"1", 表示輸入。
- 設為輸出時, 欲輸出的狀態填入LAT*xn*或PORT*xn*,對應的接腳R*xn*就會有輸出高準位或低準位。
- 設為輸入時, 可以讀取PORT*xn*取得外部的實際狀態。

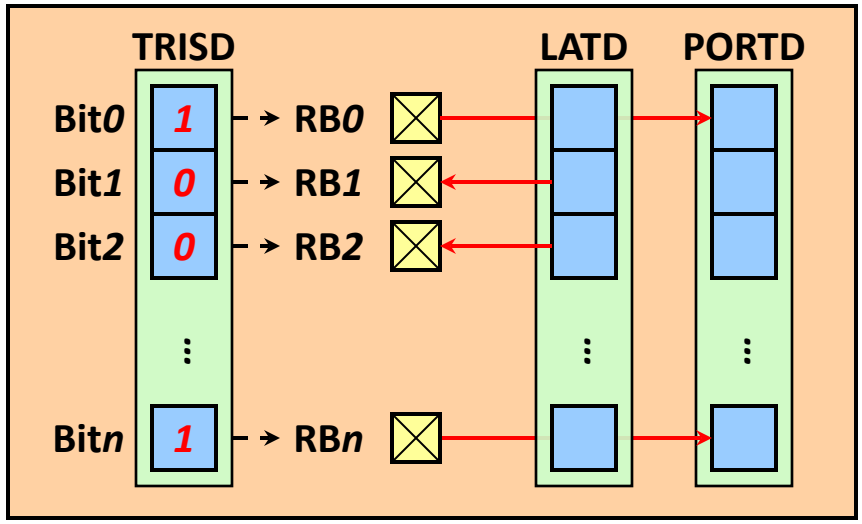




MICROCHIP
Regional Training Centers

I/O Port Manipulation

- TRISx, LATx跟PORTx特殊功能暫存器都是一個8 Bits的暫存器,暫存器中每個Bit都控制著對應的I/O接腳。
- 以PortD為例,TRISD, LATD, PORTD的Bit 0,控制RD0接腳。
- 要用RD0控制LED時, 必須先把TRISD的Bit 0設為0(輸出), 然後把要輸出的狀態,填入LATD的Bit 0。
- 要用RD0讀取按鍵狀態時, 則必須把TRISD的Bit 0設為1(輸入), 然後就可以從PORTD的Bit 0取得外部的狀態。



MICROCHIP
Regional Training Centers

Access SFRs (位元組操作)

- 透過MCU標頭檔(Header File) 的定義,可以使用與Datasheet相同的名稱來讀取SFRs。
- For Example:

```
TRISD = 0x00;          // 設定所有的 PORTD 腳位為 Output
LATD = 0x88;           // 設定所有的 PORTD 腳位輸出 High
TRISB = 0xff;          // 設定所有的 PORTB 腳位為 Input
Value =  PORTB;        //讀取所有的 PORTB 的狀態
```

- For Example:

```
TRISD &= ~0x01;        // 設定 PORTD Bit0 (RD0)腳位為 Output
LATD |= 0x01;          // 設定 PORTD Bit0 (RD0)腳位輸出 High
TRISB |= 0x01 ;        // 設定 PORTB Bit0 腳位為 Input
Value = PORTB & ~0x01; // 讀取 PORTB Bit0 的狀態
```

MICROCHIP
Regional Training Centers

Access SFRs (位元操作)

- MCU標頭檔(Header File)中, 也定義了各個SFRs的結構, 因此也可以透過結構, 存取SFRs內的特定成員(*SFRNAME*bits.*BITNAME*)。
- For Example:

```
TRISDbits.TRISD0 = 0;          // 設定 RD0 腳位為 Output
LATDbits.LATD0 = 1;            // 設定 RD0 腳位輸出 High
TRISBbits.TRISB0 = 1;          // 設定 RB0 腳位為 Input
 Value = PORTBbits.RB0;        // 讀取 RB0 的狀態
```

- TRIS*x*及TRIS*x*bits都參考到同一SFRs的位址, 因此操作TRIS*x*或TRIS*x*bits結果是相同的。
- 每個SFRs的結構型態定義都不同,建議直接打開標頭檔(Header File)來觀察看看。

MICROCHIP
Regional Training Centers

MCU's Header File

- P18f4550.h MCU標頭檔(Head File)的內容片段
  C:\Program Files (x86)\Microchip\mplabc18\v3.47\h\

```
extern volatile near unsigned char      PORTD;
extern volatile near union
{
  Struct
  {
    unsigned RD0:1;
    unsigned RD1:1;
    unsigned RD2:1;
    unsigned RD3:1;
    unsigned RD4:1;
    unsigned RD5:1;
    unsigned RD6:1;
    unsigned RD7:1;
  };
  Struct
  {
    unsigned SPP0:1;
    unsigned SPP1:1;
    unsigned SPP2:1;
    unsigned SPP3:1;
    unsigned SPP4:1;
    unsigned SPP5:1;
    unsigned SPP6:1;
    unsigned SPP7:1;
  };
} PORTDbits;
```

PORTE	RDPU(3)	—	—	—	RE3(5)	RE2(3)	RE1(3)	RE0(3)	0--- x000	56, 125
PORTD(3)	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	56, 124
PORTC	RC7	RC6	RC5(6)	RC4(6)	—	RC2	RC1	RC0	xxxx -xxx	56, 121
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	56, 118
PORTA	—	RA6(4)	RA5	RA4	RA3	RA2	RA1	RA0	-x0x 0000	56, 115

MICROCHIP
Regional Training Centers

MCU's Header File

- P18f4550.h MCU標頭檔(Head File)的內容片段
  C:\Program Files (x86)\Microchip\mplabc18\v3.47\h\

```
extern volatile near unsigned char      ADCON1;
extern volatile near union
{
  struct
  {
    unsigned PCFG:4;
    unsigned VCFG:2;
  };

  Struct
  {
    unsigned PCFG0:1;
    unsigned PCFG1:1;
    unsigned PCFG2:1;
    unsigned PCFG3:1;
    unsigned VCFG0:1;
    unsigned VCFG1:1;
  };
} ADCON1bits;
```

ADRESH	A/D Result Register High Byte								xxxx xxxx	54, 274
ADRESL	A/D Result Register Low Byte								xxxx xxxx	54, 274
ADCON0	—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON	--00 0000	54, 265
ADCON1	—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0	--00 0qqq	54, 266
ADCON2	ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0	0-00 0000	54, 267

MICROCHIP
Regional Training Centers

Lab1 – LED Control

- 先透過專案精靈, 一步一步的建立您的第一個專案。 ***(Refer Appendix)***
- 接著再利用Lab1的程式基礎,嘗試控制LED(D1, RD0)的動作, 並利用軟體迴圈來控制間隔時間 。

MICROCHIP
Regional Training Centers

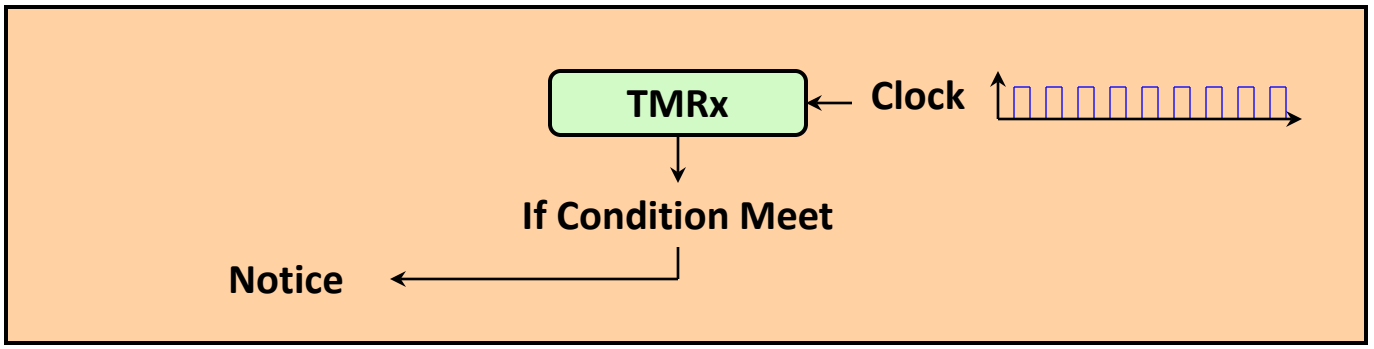
PIC18 Timer Architecture

* *Exercises, Timer2 Polling*
* *Exercises, Timer2 Polling with Scale*
* *Exercises, Timer0 16 Bits Mode Polling*

MICROCHIP
Regional Training Centers

What's Timer ?

- Timer簡單的說就是一個會持續不斷的計算進入Timer中Clock數量的模組。
- Timer一般有兩種稱呼Timer或Counter。
  Clock頻率未知時, 僅能得知計數到多少個Clock,稱為Counter。
  Clock頻率已知時, 便可以換算出時間, 稱為Timer。
  其實是一樣的東西。



MICROCHIP
Regional Training Centers

What's Timer ?

- PIC18的的Timer為遞增型,也就是會固定從"0"開始計數。模組可以設定計數到多少Clock後要通知CPU。
  條件根據不同的Timer結構設計有所不同, PIC18是以計數到特定數值或溢位為滿足條件。例如可以設定, 要計數1,000個Clock後, 發出通知。
- 此處指的通知, 其實就是中斷旗標(Interrupt Flag) 。透過中斷旗標可以對CPU發出中斷請求 (Interrupt Request)。
- 要取得此一狀態, 在輪詢架構下, 程式可以自行檢查該旗標的狀態, 來確認計數的狀態。
  或者透過致能中斷, 讓CPU自行監控, 並執行中斷服務常式。

MICROCHIP
Regional Training Centers

What's Timer ?

- Timer簡單的說就是一個會持續不斷的計算進入Timer中Clock數量的模組。
- Timer一般有兩種稱呼Timer或Counter。其實是一樣的東西。
  Clock頻率未知:僅能得知計數到多少個Clock,稱為Counter。
  Clock頻率已知:可以進一步換算出時間, 稱為Timer。
- 16 Bits MCU的Timer為遞增型,也就是會固定從"0"開始計數。
  模組可以設定數多少時要通知CPU。
  例如:設定計數到1,000個Clock後, 通知CPU。
- 此處指的通知, 就是中斷旗標(Interrupt Flag) 。當數到期望值時, 中斷旗標會被模組設定為"1", 對CPU發出中斷請求(Interrupt Request)。
  如果此時中斷是除能的, 則可由程式檢查事件是否發生。
  如果此時中斷是致能的, 就會自動進入中斷服務常式。

MICROCHIP
Regional Training Centers

PIC18的Timers種類

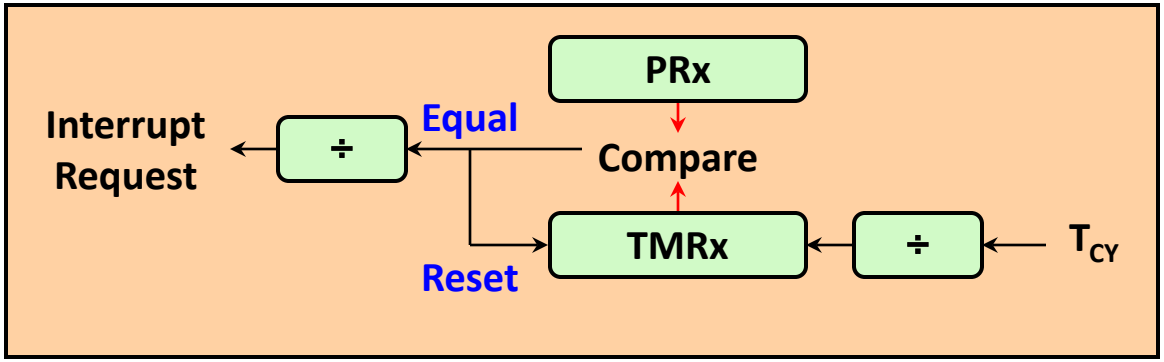
- PIC18F4520擁有四組Timer。
  Timer0, Timer1, Timer2及Timer3。

- Timer0, Timer1與Timer3屬於上數溢位型(溢位時發出中斷),
  Timer2則是上數匹配型(與特定數值相同時發出中斷)。

- Timer2只支援8 Bits(0~255)模式,
  Timer0可支援8 Bits(0~255)與16 Bits(0~65535)模式,
  Timer1與Timer3僅支援16 Bits(0~65535)模式。

Timers's 計時/計數模式

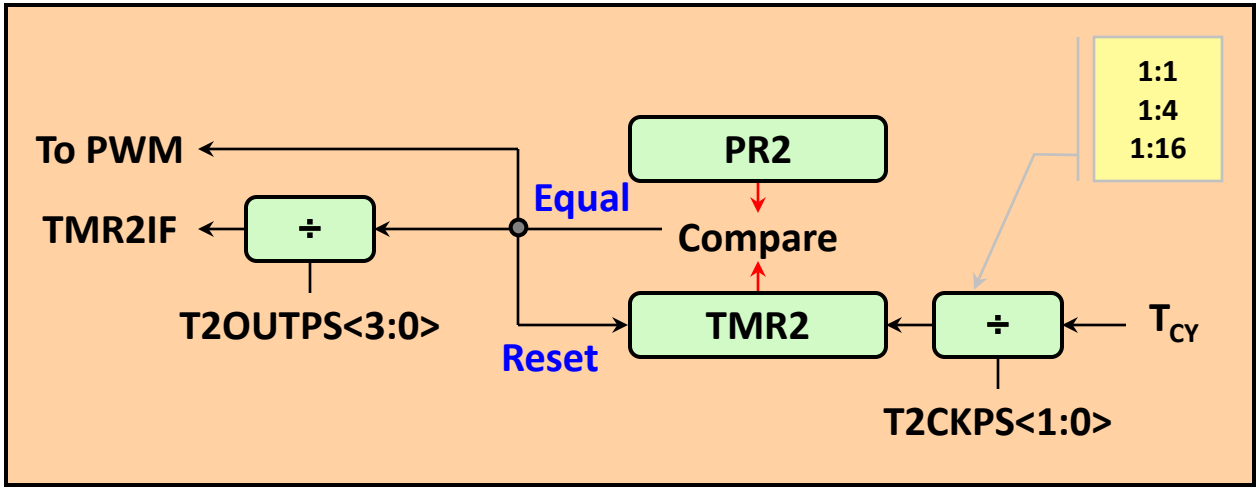
- 上數匹配型Timer的基本方塊圖, 如圖所示。
- Clock經過預除器後,送入TMRx。TMRx從"0"開始遞增, Compare會不斷比較PRx與TMRx的值, 與PRx相同時發出中斷需求, 並且將TMRx的值歸零。
- 舉例來說,假設T_{cy}是0.25uS, 想要計算50uS的時間。則可將預除器設為除1, PRx設為200。如此一來, 每次數到200個 Clock時, Timer就會設定中斷旗標, 發出中斷需求, 並清除TMRx 。
- 如果有致能中斷, MCU就會自動呼叫中斷服務常式。或者也可透過輪詢(Polling)中斷旗標得知事件是否發生。



MICROCHIP
Regional Training Centers

Timer2 Architecture

- Timer2方塊圖如下。Timer2屬於上數匹配型Timer。時脈來源則僅能選擇由內部的T_{CY}提供。最大的特點在於可以透過除頻器, 降低中斷的頻率。



MICROCHIP
Regional Training Centers

Timer2 Example

- Timer2的初始化範例:
  E.g.:
  …
  T2CONbits.T2OUTPS = 0x00;
  T2CONbits.T2CKPS = 0x00;
  T2CONbits.TMR2ON = 1;
  PR2 = 200;
  …
- Timer2的初始化後,可以利用Polling TMR2IF (Timer2的中斷旗標)的方式,來得知Timer2是否計數到預期值。
  E.g.:

```
if(PIR1bits.TMR2IF)
{
   PIR1bits.TMR2IF = 0;
   ....
}
```

REGISTER 13-1: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS<3:0>:** Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS<1:0>:** Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

MICROCHIP
Regional Training Centers

Lab2 – Timer2 Polling

- 在Lab1的程式基礎上,嘗試使用Timer2來控制Buzzer(RC2)的, 讓蜂鳴器(Buzzer)產生出10KHz(100uS)的聲音。

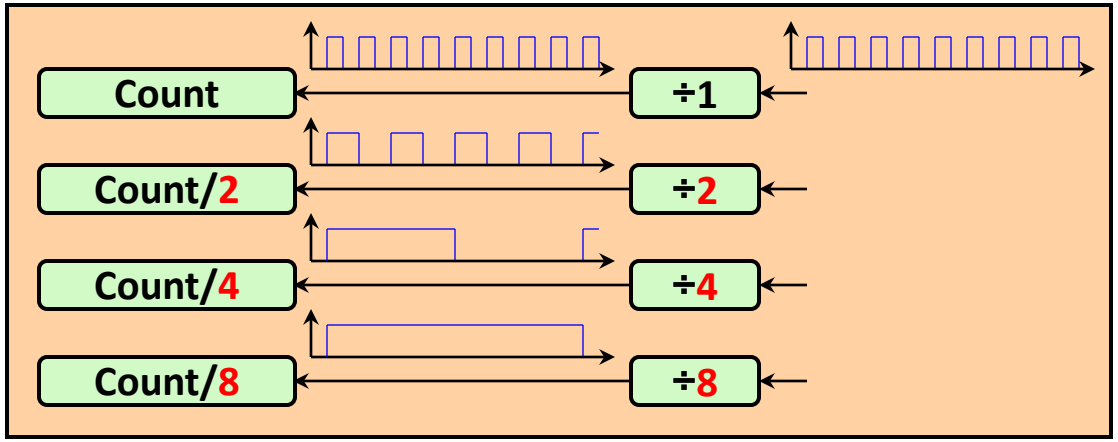
Prescaler & Postscaler

- PIC18 的Timer, 計數範圍為0~255(8 Bits)與0~65,535(16 Bits)。TMRx, PRx都不可超過。如果需要的計數值超過時, 必須透過預除器修正。
- 預除器可以用來擴大計數範圍。提高預除器比率, 減少計數值。
- 以Timer2為例, 如果Clock為0.25uS, 要達成100mS的周期, 計數值必須設定為400。即400 * 0.25uS = 100mS。但此數值已經超過PR2範圍。此時可以透過預除器的設定, 減少計數值。

$$\left(2^n - 1\right) \le PRx = \frac{Peroid}{Clock \times Scale}$$

✗ $$PRx = \frac{50mS}{0.25uS \times 1}, PRx > 255$$

✓ $$PRx = \frac{50mS}{0.25uS \times 2}, PRx < 255$$



MICROCHIP
Regional Training Centers

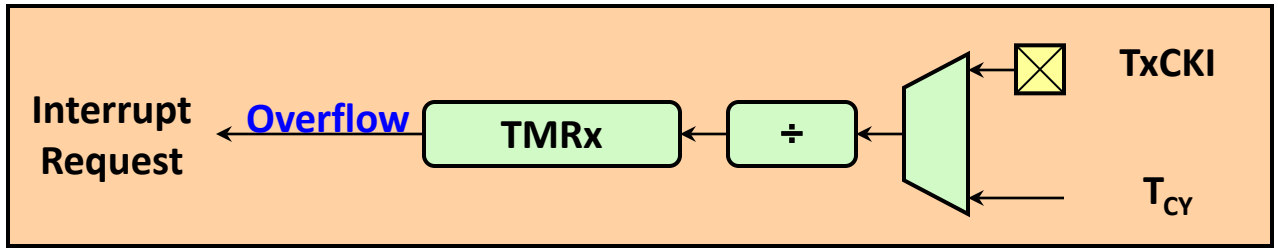
Lab3 – Timer2 Polling with Scale

- 在Lab2的程式基礎上,嘗試使用Timer2來控制LED(D1, RD0)的轉態(Toggle, 1->0->1->…) 。 Timer2的週期設為1mS, 再搭配變數的計數, 達成LED Toggle間隔為500 mS的要求 。

MICROCHIP
Regional Training Centers

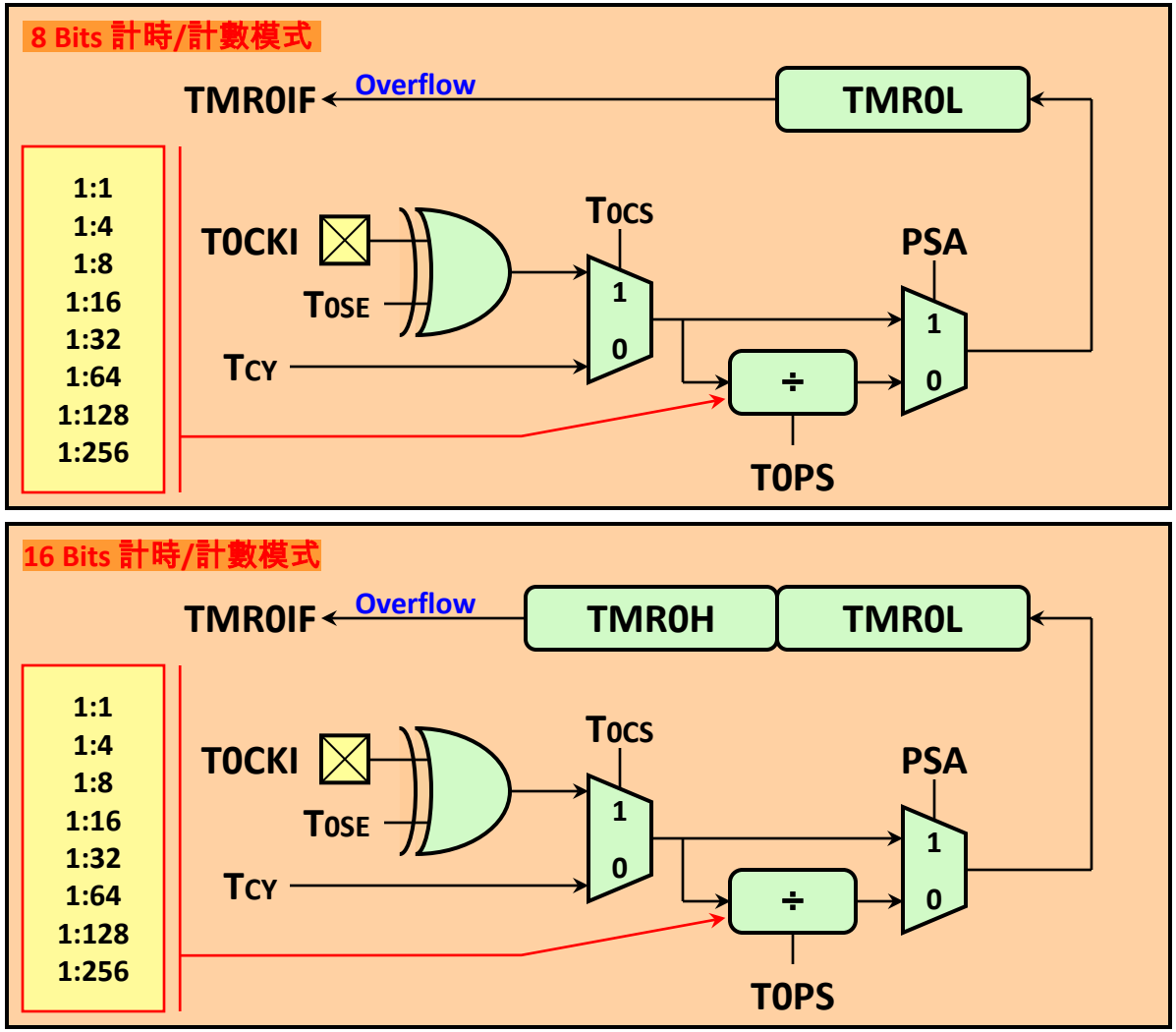
Timers's 計時/計數模式

- 上數溢位型Timer的基本方塊圖, 如圖所示。
- Clock可以選擇內部, 或者由外部接腳輸入。經過預除器後, 送入TMRx。TMRx從某數值開始遞增, 直到溢位時發出中斷需求。
- 以16 Bits Timer模式來舉例來說,假設T_{CY}是0.25uS, 想要計算50uS的時間。則可將Clock輸入設定為T_{CY}, 預除器設為除1, PRx設為65,536-200。
  如此一來, TMRx會從65,336開始計算, 數了200個Clock後, 發生溢位, 然後設定中斷旗標, 發出中斷需求。
- 如果有致能中斷, MCU就會自動呼叫中斷服務常式。
  或者也可透過輪詢(Polling)中斷旗標得知事件是否發生。



Timer0 Architecture

- Timer0方塊圖如下。Timer0最大的特點是可選擇8 Bits或16 Bits 計時/計數模式。
- 時脈來源則可選擇內部的T_{CY}或由外部輸入。
- 16 Bits 下, 具有同步讀寫機制, 可以避免寫入/讀取TMR0H/TMR0L不同步的問題。



MICROCHIP
Regional Training Centers

Timer0 Example

- Timer0 16Bits Mode的初始化範例:

E.g.:

```
...
T0CONbits.T08BIT = 0;
T0CONbits.T0CS = 0;
T0CONbits.T0SE = 0;
T0CONbits.PSA = 0;
T0CONbits.T0PS = 0x00;

TMR0H = 0xFF;
TMR0L = 0x38;  // 65,536 - 200

T0CONbits.TMR0ON = 1;
...
```

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **TMR0ON**: Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS<2:0>**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

MICROCHIP
Regional Training Centers

Timer0 Example

- Timer0的初始化後,可以利用Polling TMR0IF(Timer0的中斷旗標)的方式, 來得知Timer0是否計數到預期值。
  上數溢位型Timer必須自行Reload新值, 以確保下次的計數值正確。
- E.g.:

```
if(INTCONbits.TMR0IF)
{
   INTCONbits.TMR0IF = 0;
   TMR0H = 0xFF;
   TMR0L = 0x38;
   ....
}
```

MICROCHIP
Regional Training Centers

Lab4 – Timer0 16 Bits Polling

- 在Lab3的程式基礎上,嘗試使用Timer0來控制LED(D8, RD7)的轉態(Toggle, 1->0->1->…) 。 每次Toggle的間隔設為1S 。

MICROCHIP

Regional Training Centers

PIC18 Interrupt

* *Exercises, Timer0 Interrupt*

MICROCHIP
Regional Training Centers

What's Interrupt ?

- 在一般的情況下,CPU是按照程式的流程依序地執行。但如果此時某些周邊希望取得CPU的服務,此時就可以透過中斷,打斷目前正在執行的程式片斷,跳到該中斷的服務常式中(ISR, Interrupt Service Routine) 。
- 中斷的來源, 有許多可能如Timer, ADC, UART, SPI,外部中斷, etc.. etc..,都可以打斷CPU。
- 中斷條件的成立,通常必須要滿足幾個要素:
  中斷需求(Interrupt Request):在MCU中,周邊會透過設定中斷旗標(xxIF)來提出中斷需求。
  中斷致能(Interrupt Enable):使用者可以透過中斷致能位元(xxIE)來決定是否接受周邊的中斷需求。
  簡單的說,就是當xxIF跟xxIE都為1時,才能打斷CPU的執行,跳到中斷服務常式中。

MICROCHIP
Regional Training Centers

Interrupt Architecture

- PIC18的中斷採”加強型單一中斷向量模式”。
- 單一中斷向量模式:
  當任一中斷發生時, 都會跳到同一個向量, 因此必須在程式中自行判斷中斷由那個周邊發出。
- 加強型:
  PIC18的中斷模式可以選擇是否支援優先權仲裁。
  選擇開啟優先權仲裁時, 使用者可以定義中斷的優先權層級(高, 低)如果兩個以上中斷需求同時發生, 則優先權高者先獲得服務。
- 開啟優先權仲裁時, 高優先權中斷向量點為0x0008, 低優先權則為0x0018。
- 關閉優先權仲裁時, 中斷向量點均為0x0008。

MICROCHIP
Regional Training Centers

Programs Architecture

一個完整的中斷架構程式片斷,必須至少有3個部份:

- 定義中斷服務函式(Interrupt Service Routine):
  中斷需求被接受後, CPU會跳至ISR中執行程式, 因此必須針對所啟用的中斷設計ISR。例如:Timer2, TMR2=PR2時, 要Toggle LED。
- 致能個別中斷以及全域中斷:
  如開頭所說,中斷必須要被致能, 周邊發出需求時, 才會被CPU接受, 因此必須在啟用中斷時, 將對應的(xxIE)設為"1"。
  可以使用*xxIE* = 1;來致能中斷。如果有開啟優先權仲裁模式時, 還須透過*xxIP* = 1; or *xxIP* = 0;來設定優先權。
  完成個別中斷的致能後, 還必須將全域中斷致能開啟,中斷架構才能正常運行。
- 初始化周邊模組, 設定發出中斷需求的時機:
  設定周邊模組要在什麼情形下, 發出中斷需求。例如: ADC轉換完成後,Timer0溢後, 或Timer2, TMR2=PR2時, 發出中斷需求。

Timer0 Code Example

```
Void ISRHigh( void ){
   //在高優先權的中斷服務函式中, 添加Timer0的事件判斷。
   if( TMR0IF && TMR0IE )
   {
      // 清除Timer0中斷旗標。
      TMR0IF = 0;
      TMR0H = 0x7B;
      TMR0L = 0xE1;
   }
}
```

高優先權中斷服務函式

```
void main( void ){
   // 初始化Timer0。
   T0CONbits.T0PS = 0x07;
   T0CONbits.TMR0ON = 1;
   TMR0H = 0x7B;
   TMR0L = 0xE1;
   // 致能個別與全域中斷。
   INTCONbits.TMR0IE = 1;
   INTCONbits.PEIE = 1;
   INTCONbits.GIE = 1;
   while( 1 );
}
```

初始化Timer0與致能中斷程式片斷

MICROCHIP
Regional Training Centers

Interrupt Service Routine

```
Void ISRHigh( void )
{
   //在高優先權的中斷服務函式中, 添加Timer0的事件判斷。
   if( TMR0IF && TMR0IE )
   {
      // 清除Timer0中斷旗標。
      TMR0IF = 0;
      TMR0H = 0x7B;
      TMR0L = 0xE1;
   }
}
```

- 新加入一個事件檢查, 判斷Timer0是否提出中斷需求。
  注意,中斷服務函式不可以有傳入跟傳回值,所以都必須設定為void型態。
- 在中斷服務函式中,一定要把對應的中斷旗標清除為零。
  在中斷的架構上, 中斷旗標會由硬體設為”1”, 但不會自動清除,必須透過軟體手動清除。
- 上數溢位型Timer必須自行Reload TMR的值, 才能確保下次的事件正確發生。

MICROCHIP
Regional Training Centers

Priority arbitration

- 雖然有簡單的優先權設定, 但如果相同優先權的中斷同時發生時, 該決定先處理何者呢?
- 由於PIC18的中斷架構屬於單一中斷向量模式, 因此此時在中斷服務函式的檢查順序, 就自然產生優先權層級的規劃。先被檢查的中斷源, 其優先權自然就比較高。
- void ISRHigh(void)

```
{
  if( xxIF && xxIE )
     // ...
 if( yyIF && yyIE )
  // ...
 if( zzIF && zzIE )
  // ...
}
```

MICROCHIP
Regional Training Centers

Interrupt Vector Redirect

- 在C18的編譯器架構內, PIC18的兩個中斷向量點, 必須先重新導向到兩個中斷服務函式, 中斷才能正確運行。

```
#pragma code HighIVT = 0x0008
void IVTHigh(void){
   _asm goto ISRHigh _endasm // 重新導向到ISRHigh函式(高優先權)。
}
#pragma code
#pragma interrupt ISRHigh
void ISRHigh (void){
   // …
}
#pragma code
```

高優先權中斷服務函式

```
#pragma code LowIVT = 0x0018
void IVTHigh(void){
   _asm goto ISRLow _endasm // 重新導向到ISRLow函式(低優先權) 。
}
#pragma code
#pragma interruptlow ISRLow
void ISRLow(void){
   // …
}
#pragma code
```

低優先權中斷服務函式

vloatile Qualification

- 如果在ISR與主程式中會共用到同一變數,則此變數在宣告時,必須加上*volatile*的關鍵字。
  Ex: *volatile* unsigned int Ticks = 0;
- *volatile*關鍵字是用來避免C Compiler在進行最佳化時,將特定變數在最佳化的過程刪除,導致程式的流程出錯。

```
extern volatile unsigned int        PORTD __attribute__((section("sfrs")));
typedef union {
  struct {
    unsigned RD0:1;
    unsigned RD1:1;
    unsigned RD2:1;
    unsigned RD3:1;
    unsigned RD4:1;
    unsigned RD5:1;
    unsigned RD6:1;
    unsigned RD7:1;
    unsigned RD8:1;
    unsigned RD9:1;
    unsigned RD10:1;
    unsigned RD11:1;
  };
  struct {
    unsigned w:32;
  };
} __PORTDbits_t;
extern volatile __PORTDbits_t PORTDbits __asm__ ("PORTD") __attribute__((section("sfrs")));
extern volatile unsigned int        PORTDCLR __attribute__((section("sfrs")));
extern volatile unsigned int        PORTDSET __attribute__((section("sfrs")));
extern volatile unsigned int        PORTDINV __attribute__((section("sfrs")));
extern volatile unsigned int        LATD __attribute__((section("sfrs")));
typedef union {
  struct {
    unsigned LATD0:1;
    unsigned LATD1:1;
    unsigned LATD2:1;
    unsigned LATD3:1;
    unsigned LATD4:1;
    unsigned LATD5:1;
    unsigned LATD6:1;
    unsigned LATD7:1;
    unsigned LATD8:1;
    unsigned LATD9:1;
    unsigned LATD10:1;
    unsigned LATD11:1;
  };
  struct {
    unsigned w:32;
  };
} __LATDbits_t;
extern volatile __LATDbits_t LATDbits __asm__ ("LATD") __attribute__((section("sfrs")));
```

一般變數的最佳化,結果符合原意!

```
B=A;
C=B;
最佳化後
C=A;
```

SFR的最佳化, 結果變成TMR1完全消失,不符合原意!

```
TMR1=A;
C=TMR1;
最佳化後
C=A;
```

!

*所有的SFRs在MCU標頭檔中,都已經宣告為volatile,但記住!自訂的共用變數必須自行加上。

MICROCHIP
Regional Training Centers

Lab5 Timer0 Interrupt

- 在Lab4的程式基礎上,嘗試將Timer0 的控制方式由原來的Polling(輪詢)修改為中斷架構。

MICROCHIP

Regional Training Centers

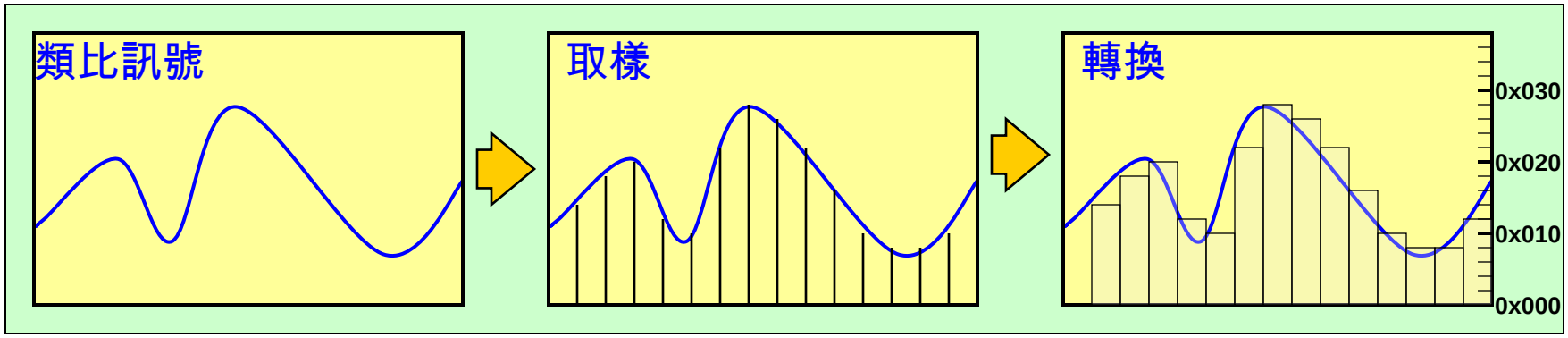
PIC18 10 Bits ADC

* *Exercises, ADC Basic*
* *Exercises, Bits Read/Write Mode*

MICROCHIP
Regional Training Centers

What's ADC ?

- ADC：Analog Digital Conversion,類比數位轉換器。
  一個可將類比訊號轉換成數位資料的模組。
- ADC的轉換過程,可以分為兩個步驟,如下圖所示,首先對類比信號進行"取樣",利用外部的類比訊號對ADC內部的小電容充電,已取得外部類比訊號的複本,接著再將獲得的資料加以"轉換",獲得量化後的數位資料。

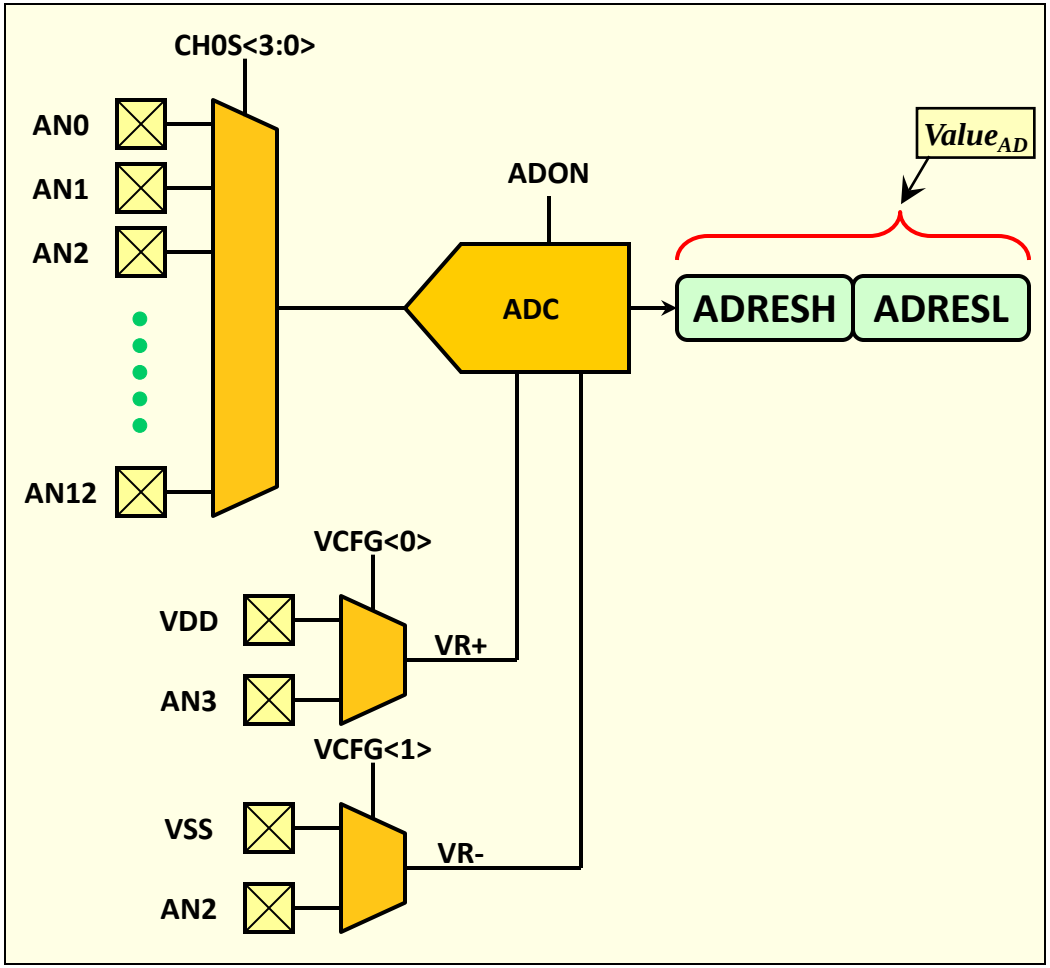


PIC18 ADC Architecture

- PIC18具有一組採用SAR (連續近似法)的 10-Bits ADC。 搭配12對1之類比多工器, 達成多通道轉換功能。
- 類比信號轉換公式

$$Value_{AD} = \left(\frac{V_{AD} - V_{R-}}{V_{R+} - V_{R-}} \times 2^n \right) - 1$$

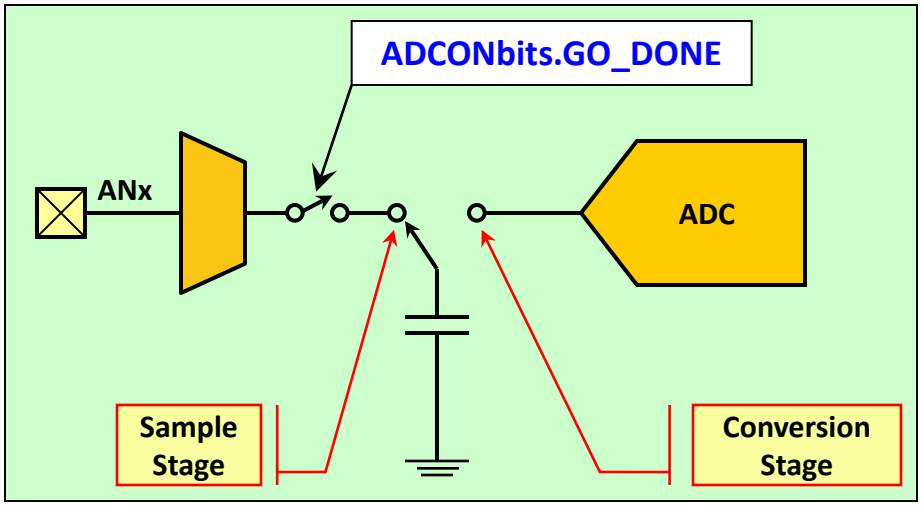
- 參考電壓可使用 V_{REF+} (AN3), V_{REF-}(AN2) 或AV_{DD} , AV_{SS} 。



MICROCHIP
Regional Training Centers

ADC Sample and Conversion

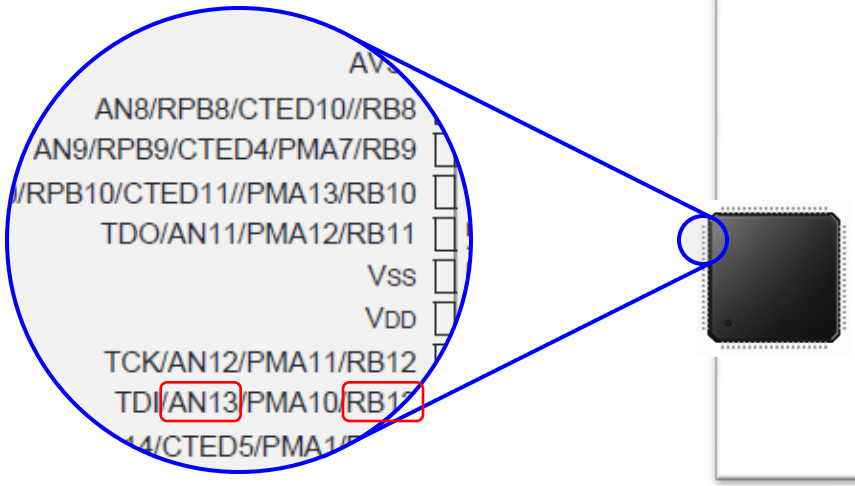
- ADC Module透過GO/DONE Bit的設定, 來進行AD轉換。
  起始AD轉換後, ADC Module會先
  進入取樣階段(Sample Stage), 對電容充電, 取得外部電壓值。
  充電完成後,
  再進入轉換階段(Conversion Stage), 根據取得的電壓進行量化, 獲得數值。
- 完成轉換後, 會將數值存入ADRESH:ADRESL。
- 可以透過檢查GO/DONE Bit, 來判斷AD是否完成轉換。



MICROCHIP
Regional Training Centers

Analog Mode

- 類比輸入腳(AN*n*)跟數位IO接腳(Rx*n*)是共用的。要當作ADC的輸入接腳時, 必須設定為類比模式(Analog Mode)。
- 類比模式的設定可以透過ADCON1bits.PCFG<3:0>決定。



bit 3-0 **PCFG<3:0>:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

MICROCHIP
Regional Training Centers

10 Bits ADC Example

- 10 Bits ADC的初始化範例:

  E.g.:

  …

```
ADCON0bits.CHS = 0x00;  // Set to AN0
ADCON1bits.PCFG = 0x0e; // Enable Analog Mode for AN0
ADCON0bits.ADON = 1;    // ADC Module Turn On
```

  …

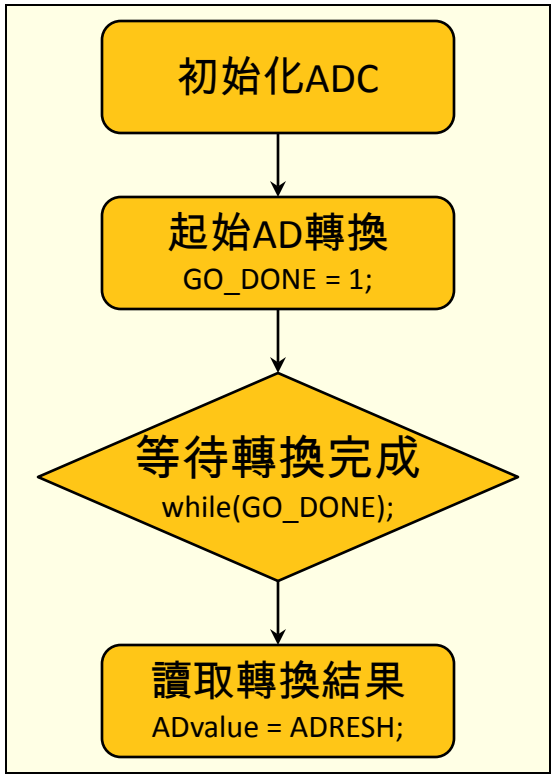
- ADC初始化後,可以利GO_DONE來啟始AD轉換與判斷是否完成轉換。

  E.g.:

  …

```
ADCON0bits.GO_DONE = 1;
while(ADCON0bits.GO_DONE);
ADValue = ADRESH;
```

  …



MICROCHIP
Regional Training Centers

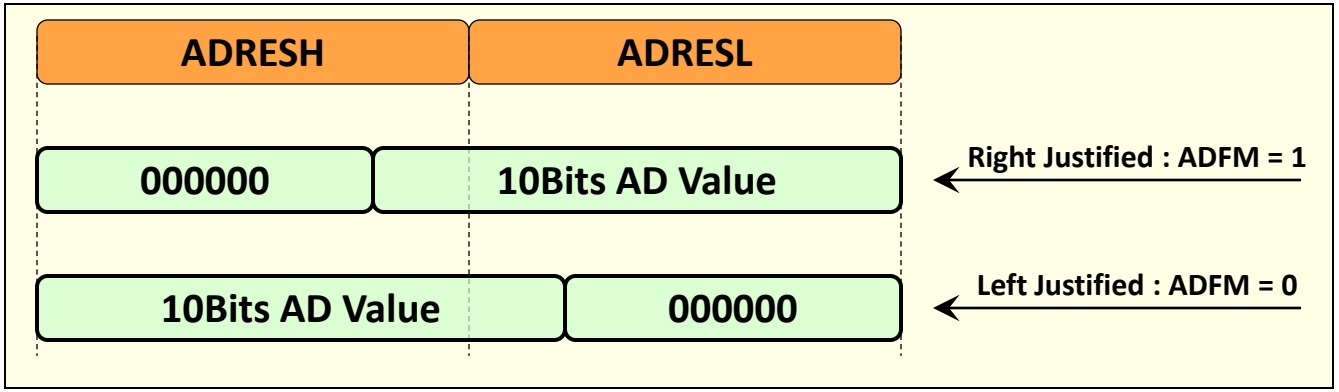
Lab6 – ADC Basic

- 在Lab5的程式基礎上,嘗試初始化ADC, 並使用ADC Module轉換VR1的電壓值, 將結果中較高的8 Bits, 顯示在LED(D1~D8)上。ADC轉換的頻率請用Timer0控制, 每秒10次。

MICROCHIP
Regional Training Centers

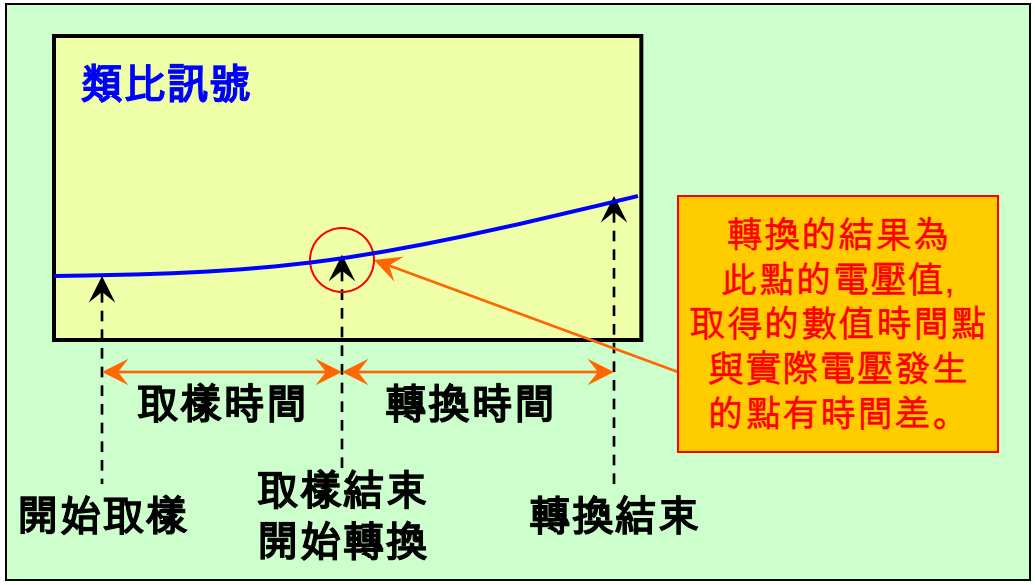
Right and Left Justified

- 10 Bits ADC的結果, 需要兩個Bytes存放, ADC Module會將其結果存放在ADRESH:ADRESL中。
- 使用者可以根據需求, 選擇資料的擺放方式。
- 一般來說,
  Right Justified設定求的是精度, 取用的是整個10 Bits的數值。
  Left Justified則是針對訊號較不穩定的場合, 僅取出較高的8 Bits數值。



Sample and Conversion Sequence

- ADC的取樣時間與轉換時間都有最短需求時間的規範,設計時必須滿足才能確保轉換結果正確。時間規範可查詢Datasheet電氣特性章節。
- 取樣時間必須大於1T_{AD}
  轉換時間固定需要12T_{AD}。
- PIC18F系列的T_{AD}最少必須為0.7uS, 實際的設定設定, 可以透過ADCON2bits.ADCS與ADCON2bits.ADQT決定。
- ADCS :設定T_{AD}時間。
- ADQT:轉換需要幾個T_{AD}。



MICROCHIP
Regional Training Centers

How to Display Additional Information ?

- 可以直接使用提供給各位的LCD Module Control Function, (APP001_LCM.c, APP001_LCM.h)。
- 提供以下幾個Function:

  void LCM_Init( ) //初始化LCD Module。
  Void LCM_SetCursor(char X , char Y) // 設定遊標位置。
  void LCM_PutASCII(unsigned char) // 輸出字元。
  Void LCM_PutHex(unsigned char Hex) //將變數轉為Hex輸出。
  Void LCM_PutROMString(const unsigned char *String) // 輸出const 字串。
  Void LCM_PutRAMString(unsigned char *String) //輸出字串。
  void LCM_PutNumber(unsigned int Number , unsigned char Digit);
  // 輸出n位數的整數。
- 提供原始碼,如果需要其它的功能,可以自行修改。

MICROCHIP
Regional Training Centers

Lab7 – 10 Bits ADC and LCM

- 在Lab6的程式基礎上,嘗試初始化ADC, 並使用ADC Module轉換VR1的電壓值, 將完整結果(10 Bits)顯示在LCD Module上。ADC轉換的頻率請用Timer0控制, 每秒10次。

MICROCHIP
Regional Training Centers

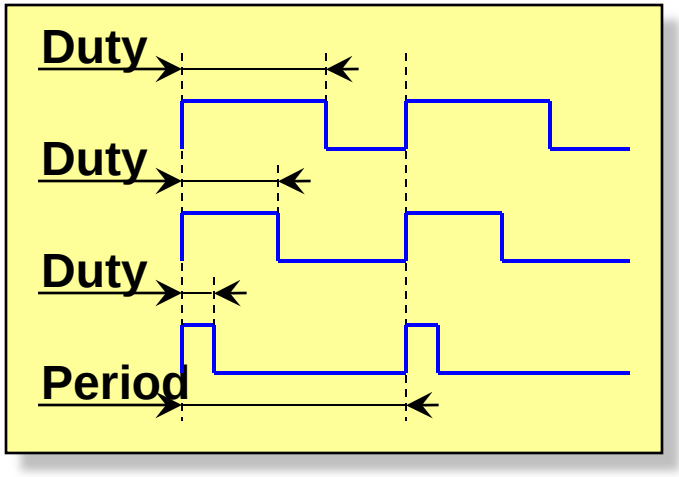
PIC18 CCP/ECCP - PWM Arch.

* *Exercises, PWM Output*
* *Exercises, PWM Output By VR Value*
* *Exercises, Breathing Light*

MICROCHIP
Regional Training Centers

What's PWM

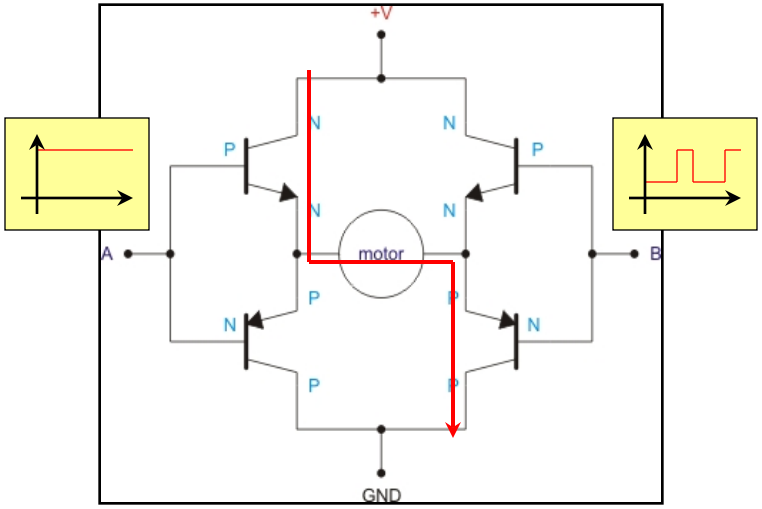
- 脈波寬度調變(PWM, Pulse-width modulation)。一種調變工作週期的技術。透過高解析度的計數器,將方波的工作週期進行調變,用來對一個類比信號的電壓進行編碼。
- PWM是以數位控制來控制類比訊號一種非常有效的技術。廣泛的被應用在測量、通信及功率控制與變換等領域中。
- PWM控制兩個重要參數
  週期(Period) 、比例(Duty) 。

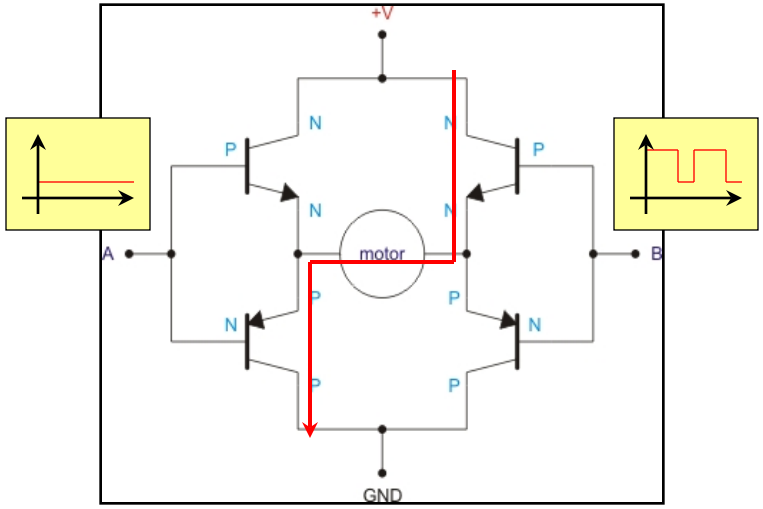


MICROCHIP
Regional Training Centers

Motor Control

- 以馬達控制為例,如圖所示的H Bridge結構馬達控制電路。可使用PWM控制正反轉與旋轉的速度。





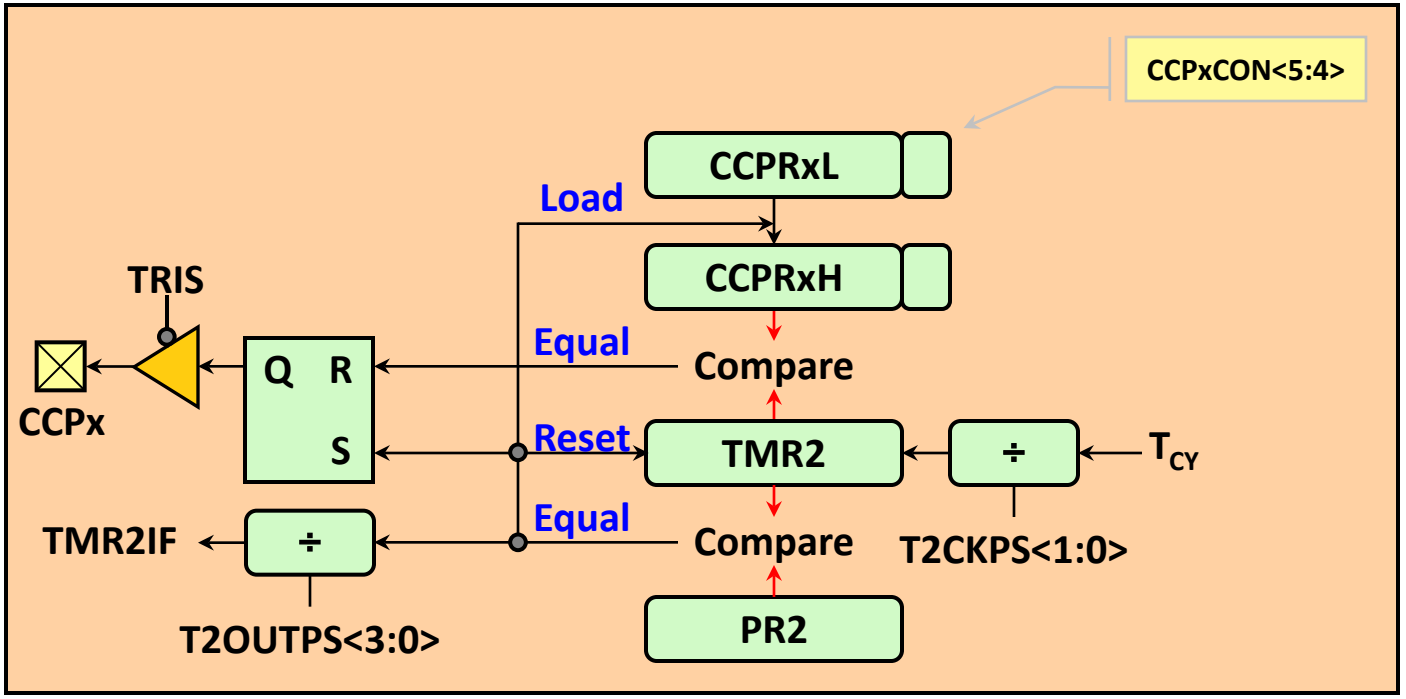
MICROCHIP
Regional Training Centers

PIC18 CCP/ECCP Module

- PIC18 具有一個複合式的波形產生與擷取模組CCP/ECCP。
- CCP, **C**apture/ **C**ompare/**P**WM Module。
  Capture用來量測外部訊號的寬度, 週期等。Compare用來產生特定寬度或工作週期的訊號輸出。PWM專門用來產生PWM訊號。
- ECCP, **E**nhanced **C**apture/ **C**ompare/**P**WM Module。主要是增加了PWM的互補輸出與更多的工作模式。基本的功能與CPP相同。

PIC18 CCP - PWM Architecture

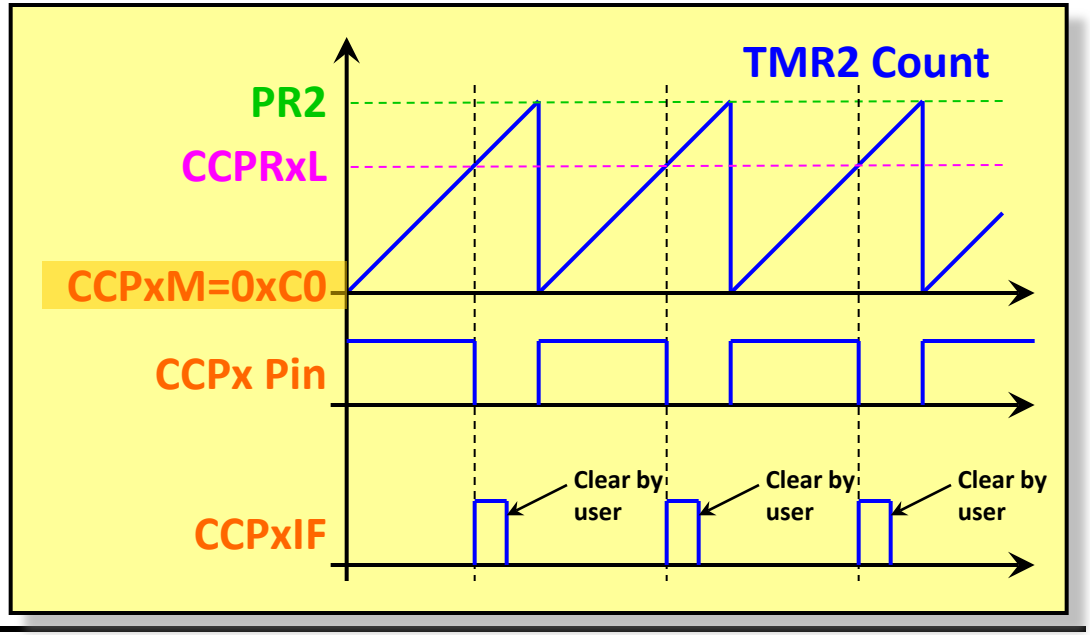
- PWM使用Timer2作為Timer Base(Period), Duty Cycle則由CCPRxL決定。其方塊圖如下圖所示:



MICROCHIP
Regional Training Centers

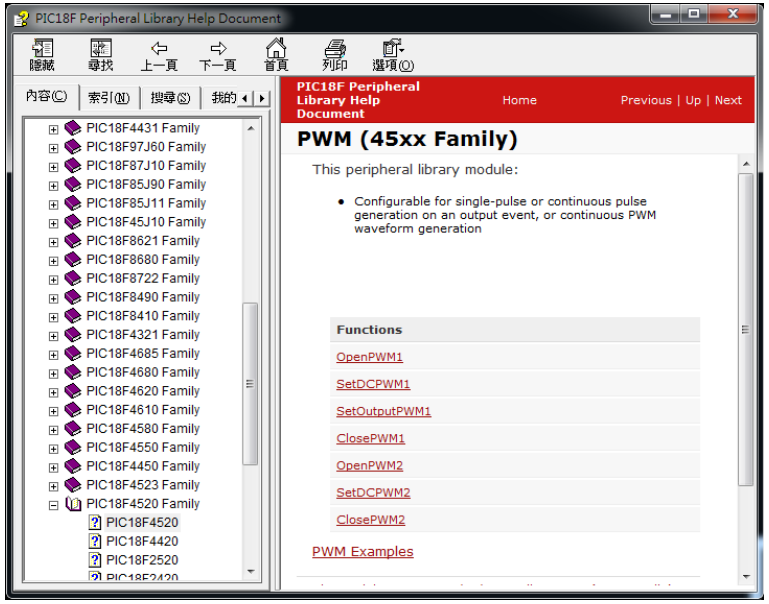
PWM Mode

- CCPxM<3:0> = 0xc0時, CCP會進入PWM Mode。
- 此模式下, TMR2會根據T_{CY}遞增。模組會比較CCPRxL與TMR2的數值。
- 當CCPRxL = TMR2時CCPx pin會變成Low輸出,
  當PR2 = TMR2時CCPx pin會變成High輸出。
- PR2決定週期(Peroid),
  CCPRxL則決定工作
  週期(Duty Cycle)。



C18 PWM Function & Macro

- C18提供的PWM操作 Function, 只要搞懂其中兩個就能控制PWM:
  - **void OpenPWMx(char period);**
    // 初始化PWMx;
  - **void SetDCPWMx(unsigned int dutycycle);**
    // 設定Duty Cycle;
  - **void ClosePWMx(void);**
    // 關閉PWM。



MICROCHIP
Regional Training Centers

PWM Initial & Example

- **Initial Example**

```
#include <pwm.h>
#define InstructionFrequency 16000000 / 4
#define T2Peroid 1000
#define TMR2Value InstructionFrequency / 16 / 1 / T2Peroid

T2CONbits.T2OUTPS = 0x00;                  // 1:1
T2CONbits.T2CKPS = 0x02;                   // 1:16
PR2 = TMR2Value;
T2CONbits.TMR2ON = 1;                      // Timer2 Turn On

OpenPWM1( TMR2Value );
SetDCPWM1( ( TMR2Value / 2) << 2 );        // 50% Duty Cycle
```

MICROCHIP
Regional Training Centers

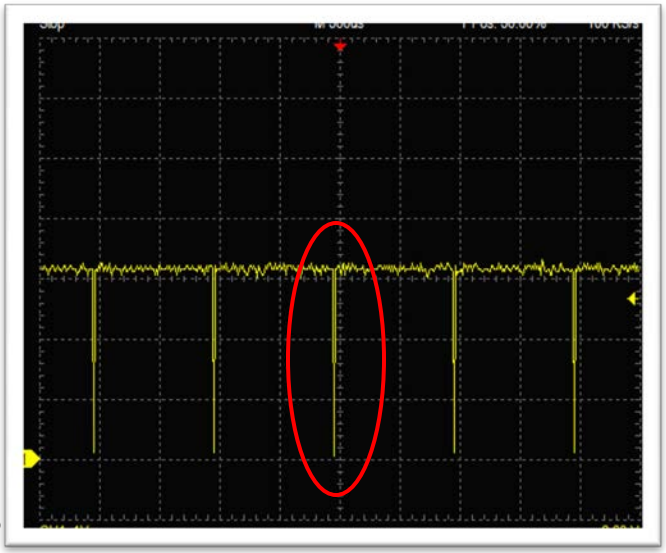
Lab8 PWM Output

- 在Lab7的程式基礎上, 初始化PWM1。
  嘗試輸出一的1KHz, Duty Cycle 20%的PWM訊號。
  記住, PWM1的Timer Base是Timer2, 所以Timer要記得設定。
- PWM1預設連接至APP001的Buzzer, 記得設定連接Buzzer的I/O Pin(RC2)為輸出。成功設定後, 可以聽到Buzzer發出的聲響。
- 建議使用示波器觀察CCP1的輸出波形。
- 完成後, 請嘗試不同Duty Cycle
  所造成的效果。
  0%, 2%, 5%, 50%, 100%。

  (Note:Buzzer可以透過JP4關閉, 如果嫌吵!!)

PWM Duty Cycle Control

- 100% Duty要如何設定?
  - PWM Mode下, 兩個關鍵條件
    1.TMR2 = CCPRxL Pin =Low, 2.TMR2= PR2 Pin = High。
  - 試想100% Duty的情況, 則 CCPRxL必須PR2 。上述兩個條件會同時成立了。造成下陷的突波(Glitch)出現。
  - 為了避免此現象發生。必須避免則 CCPRxL=PR2 。
  - 因此在此情況下, 可技巧性的將CCPRxL設定的比PR2大一點。
    如此一來CCPRxL=PR2的條件就沒機會成立。
    (TMR2=PR2時, TMR2會自動歸零, Timer的架構設定)。



MICROCHIP
Regional Training Centers

PWM Set Example

- **CCP1 PWM Duty Cycle Set Example**

```
if( DutyValue >= PR2  )
        SetDCPWM1(  PR2 + 1 );
else
        SetDCPWM1( DutyValue << 2);
```

  - 設定CCP1的Duty Cycle。
    當 DutyValue >= PR2時(100% Duty Cycle), 將CCPRxL填入PR2+1。以消除突波(Glitch)問題。
    其他他情形時(0~99.999%Duty Cycle)則直接填入DutyValue值。

MICROCHIP
Regional Training Centers

Lab9 PWM Output By VR Value

- 在Lab8的程式基礎上, 嘗試加入ADC的功能。並利用VR的電壓來調整PWM2的Duty Cycle。PWM2頻率同樣設定為1KHz。
- 請正規化為
  VR:0V ~ 5V <=> PWM Duty:0% ~ 100%
- CCP2(RC1)可以透過調整JPCCP2與LED(D9)。連接。請直接透過LED來觀察PWM Duty Cycle對於LED的影響。也可以使用示波器觀察CCP2的輸出波形。

MICROCHIP
Regional Training Centers

Lab10 Breathing Light

- 嘗試利用所學, 實作呼吸燈。
- 簡單來說就是PWM Duty會自動變化的狀態。透過自動變化可以控制LED的亮度。控制得當時, 看起來就像人在呼吸一般。
- 利用Lab7的程式, 將PWM的功能改變為自動調整。
  讓PWM的Duty Cycle可以從
  0% -> 100% -> 0% -> … 不斷的變化。
- PWM自動調整的速率請設定在2~5秒間。
- **Hint:**
  Timer Event to do below:

```
unsigned int Duty = 50;
char DutyDistance = 2;
Duty +=DutyDistance;
if ( Duty >= 100 || Duty <= 0 )
        DutyDistance = -DutyDistance;
```

MICROCHIP

Regional Training Centers

Thank you a lot !

MICROCHIP

Regional Training Centers

Appendix

建立第一個 MPLAB X IDE 專案

Regional Training

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
- 為 Lab1 的 LED 控制實習建立專案

如何建立獨立專案

目標

- 練習一所提供的原始檔案從頭開始建立新的 MPLAB® X 專案
  - 使用單一的 source code :
    - Lab1 LED Control.c
- 使編號為 D1 的 LED1 定時閃爍

建立獨立專案步驟

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

建立獨立專案

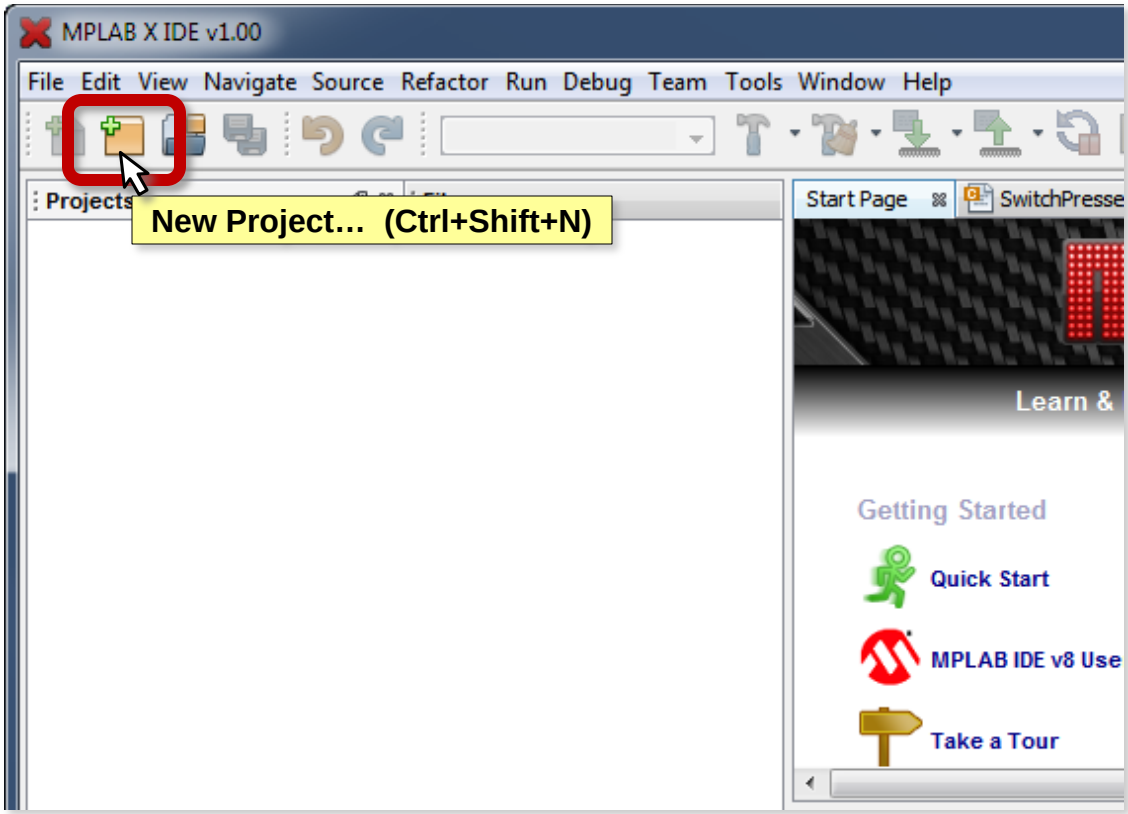
1 啟動新建專案精靈

工具列

選單

File（文件）►
New Project...
（新建專案...）

鍵盤



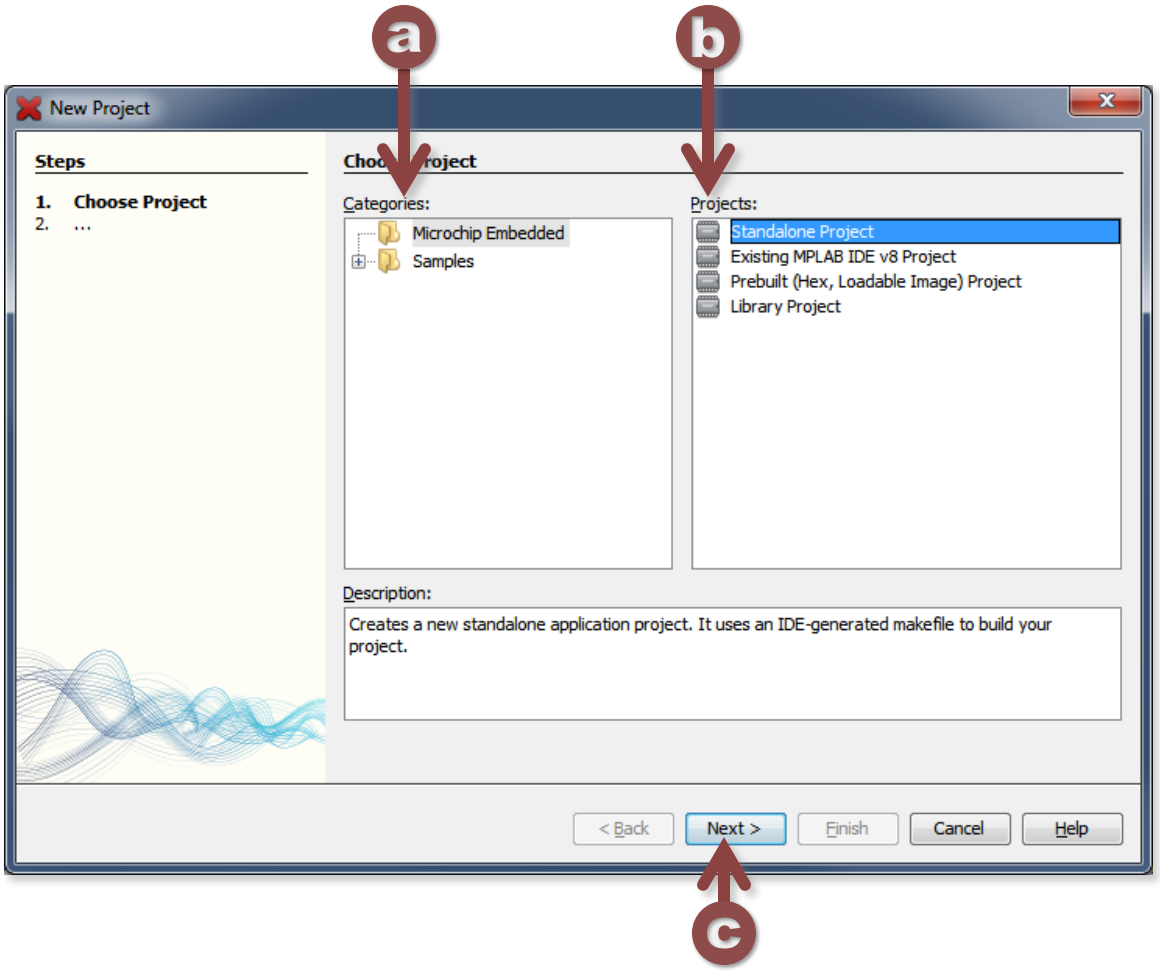
建立獨立專案

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）

c 按一下 **Next >**



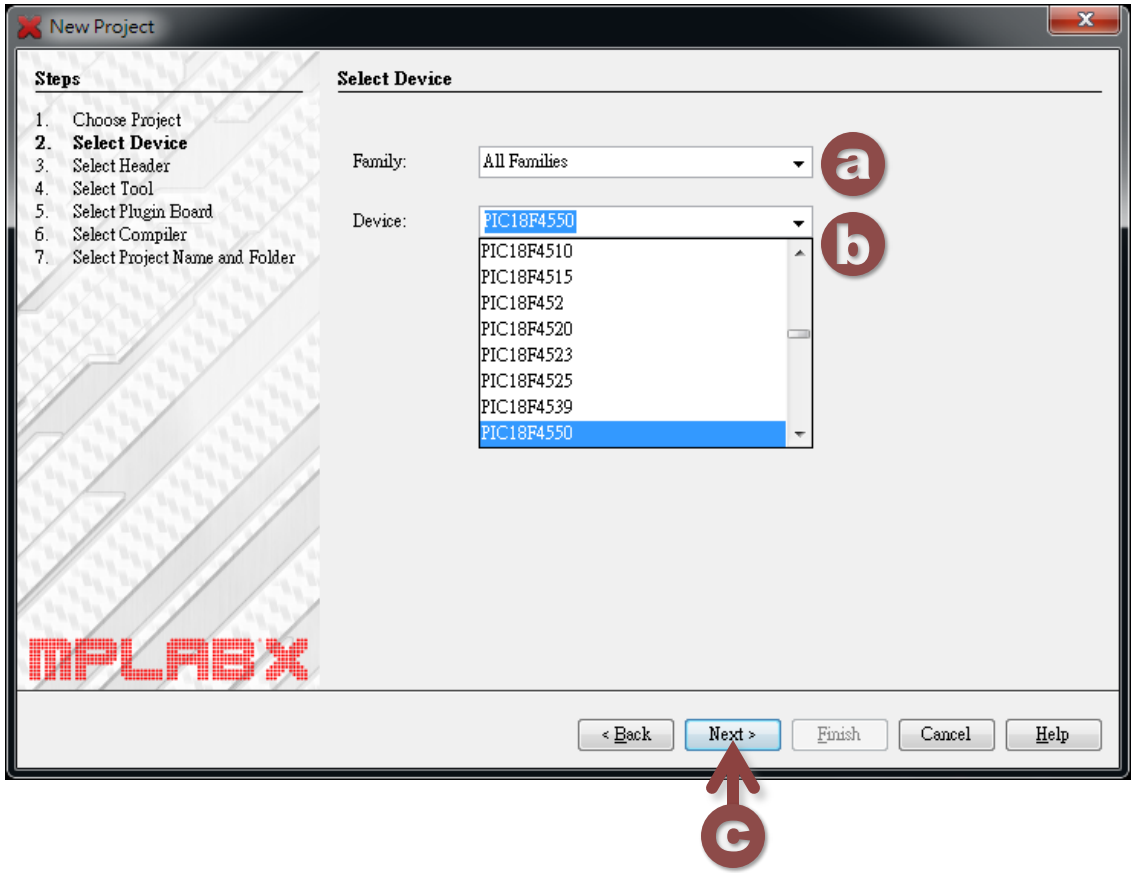
建立獨立專案

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：
PIC18F4550

c 按一下 **Next >**



建立獨立專案

5 選擇工具

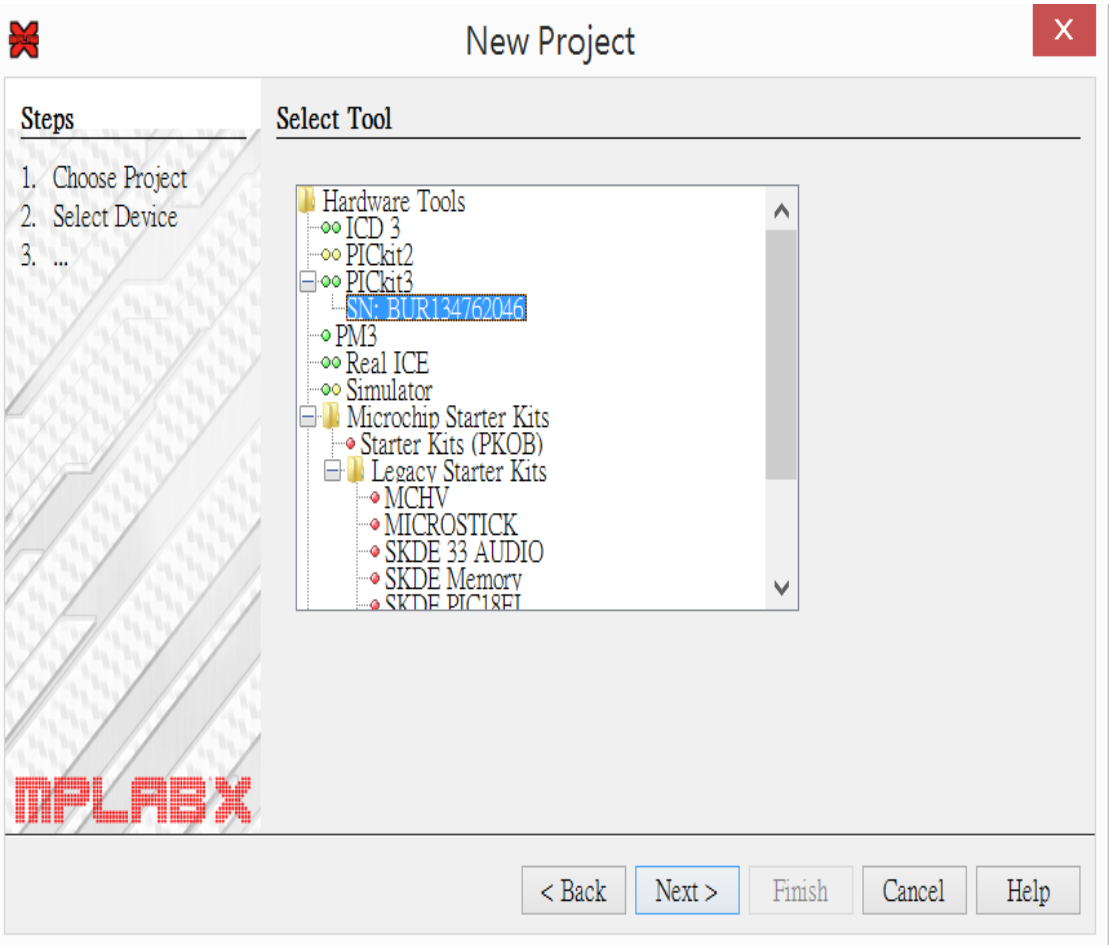
a 選擇
點選 PICkit 3下的序號

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 PICkit 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 PICkit 3
二為 ICD 3

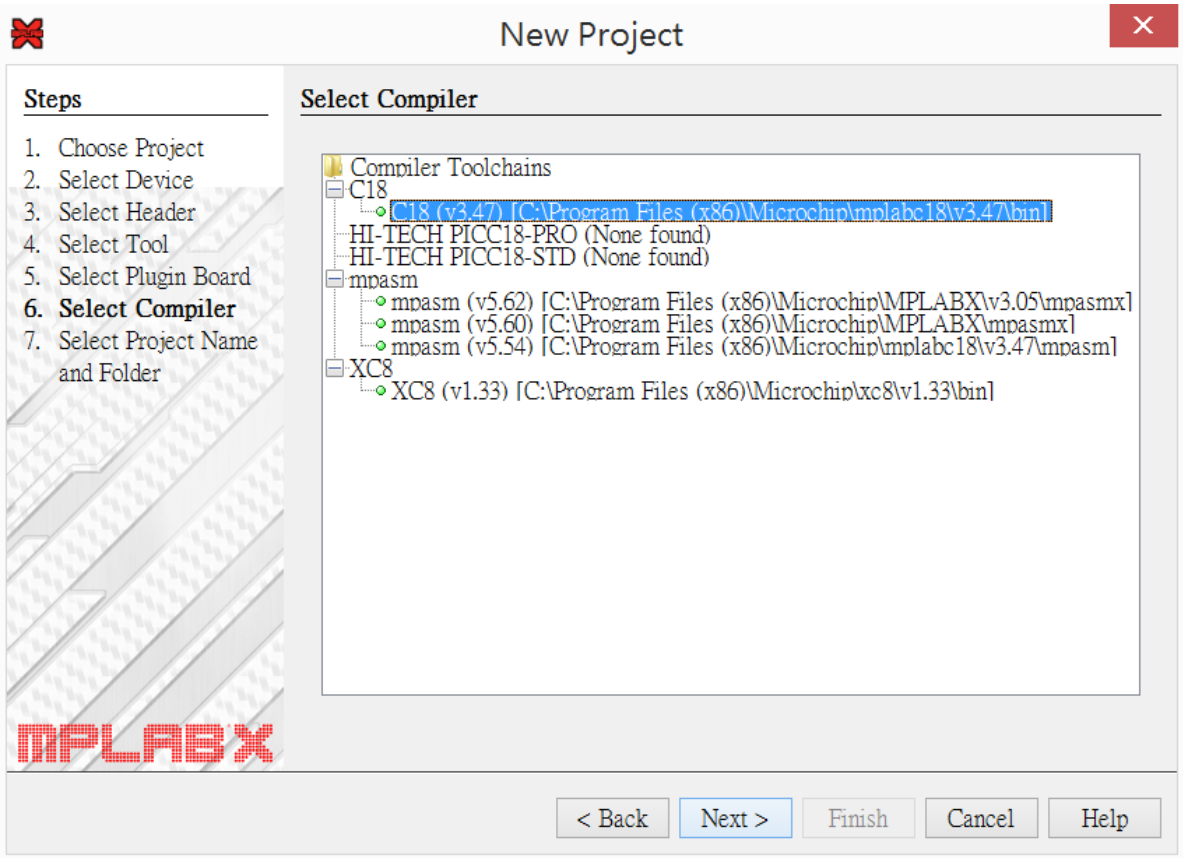
b 按一下 **Next >**



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 C18 底下 C18 (v3.47)



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 >**

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*：

Lab1 LED Control

b 輸入*專案位置*：

C:\..\Exercise\Lab1

會在專案文件**夾**下建立一個具有專案名稱的資料夾。

磁碟機 (C☺

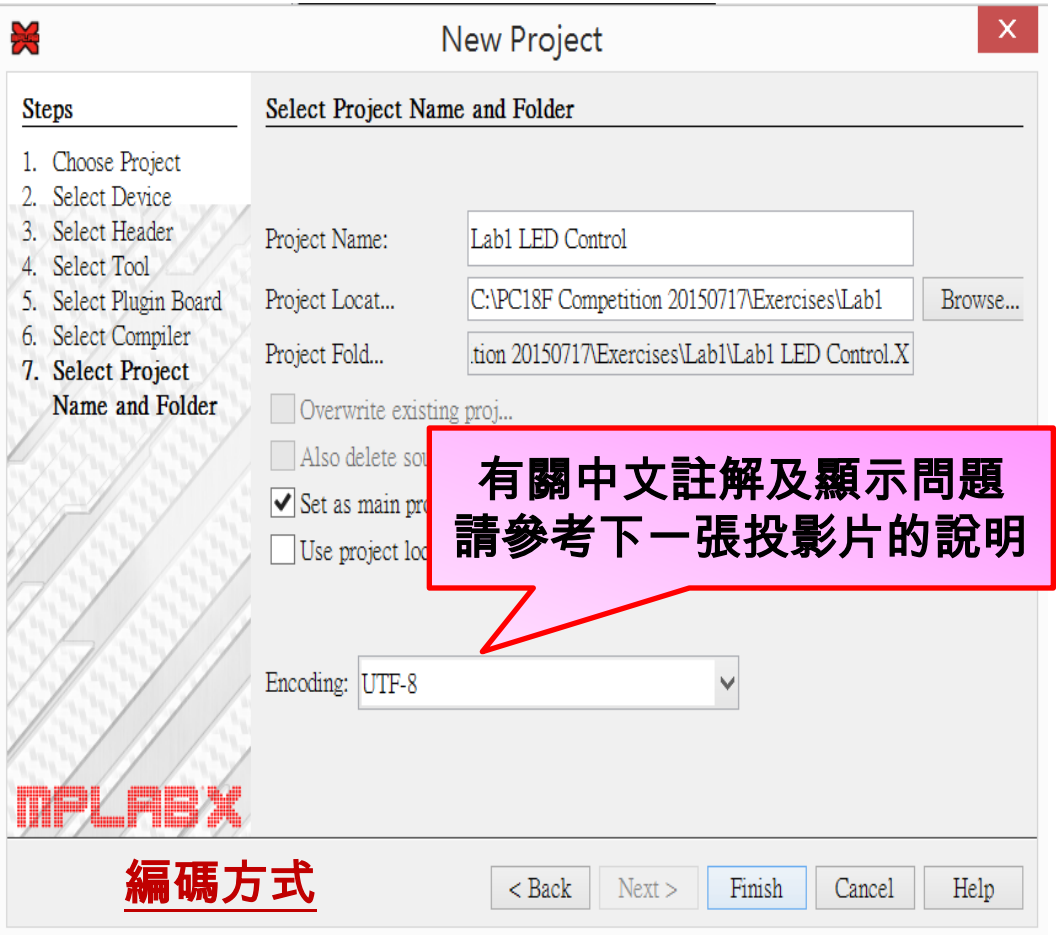
PC18F Competition 20150717

Exercise

Lab1 ⟵ 專案資料夾

Lab1 LED Control.X ⟵ 專案名稱.X

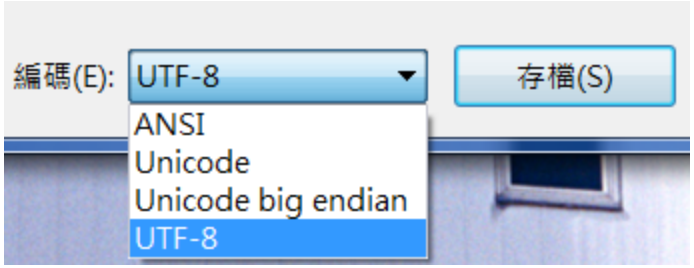
c 按一下 **Finish**



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



建立獨立專案

8 將檔加入到新專案中

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**…）

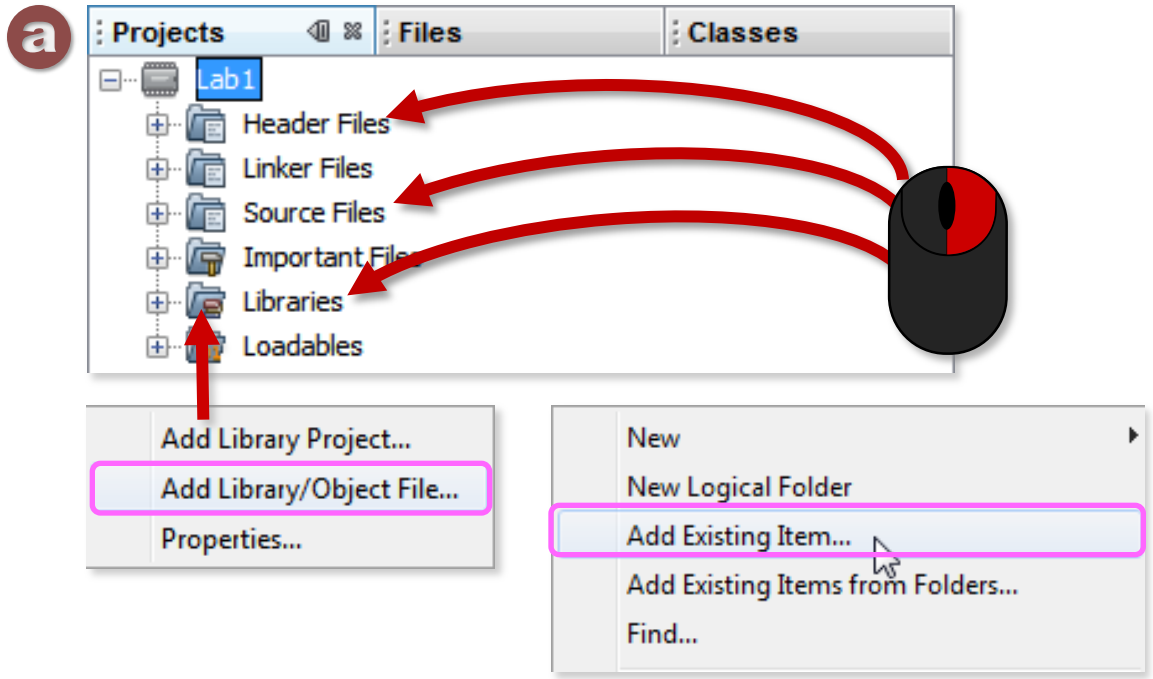
將相關檔案加到以下資料夾中：

b •**Header Files**

c •**Source Files**

d •**Libraries**

（詳見下一頁）



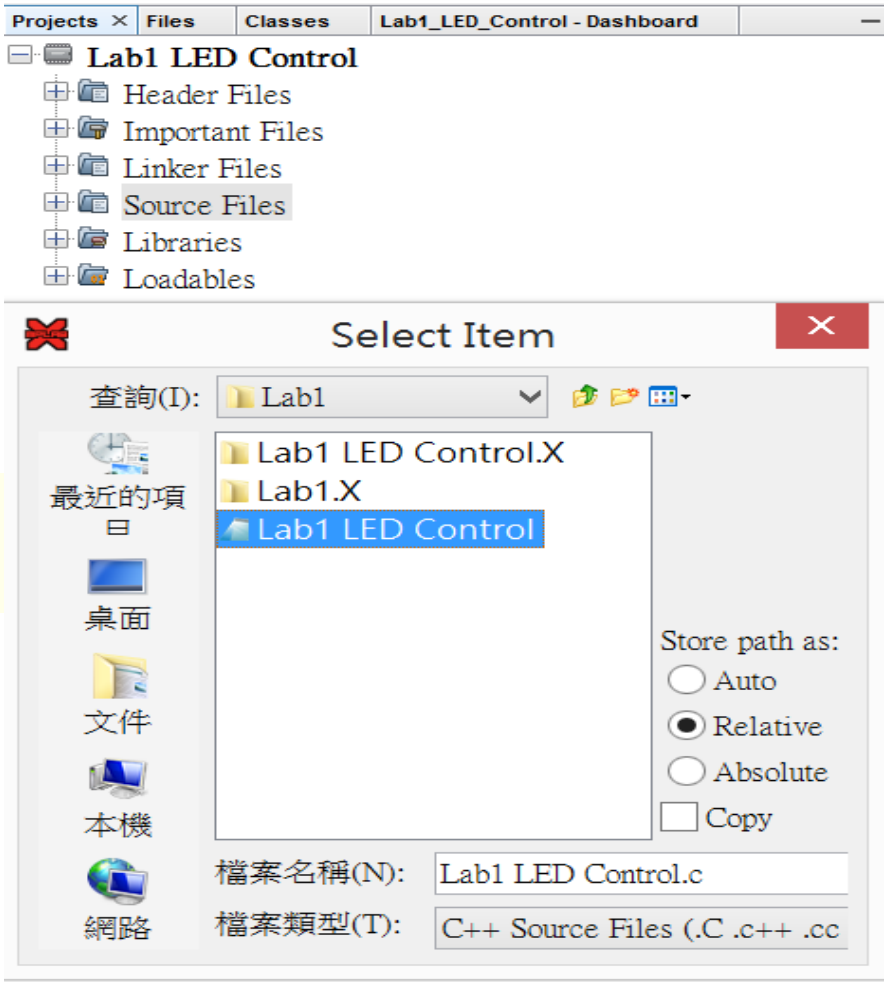
建立獨立專案案

8 將相關檔案加入到新專案中（續……）

從 X IDE Lab1.X目錄中加入下列文件

可被加入的 檔案

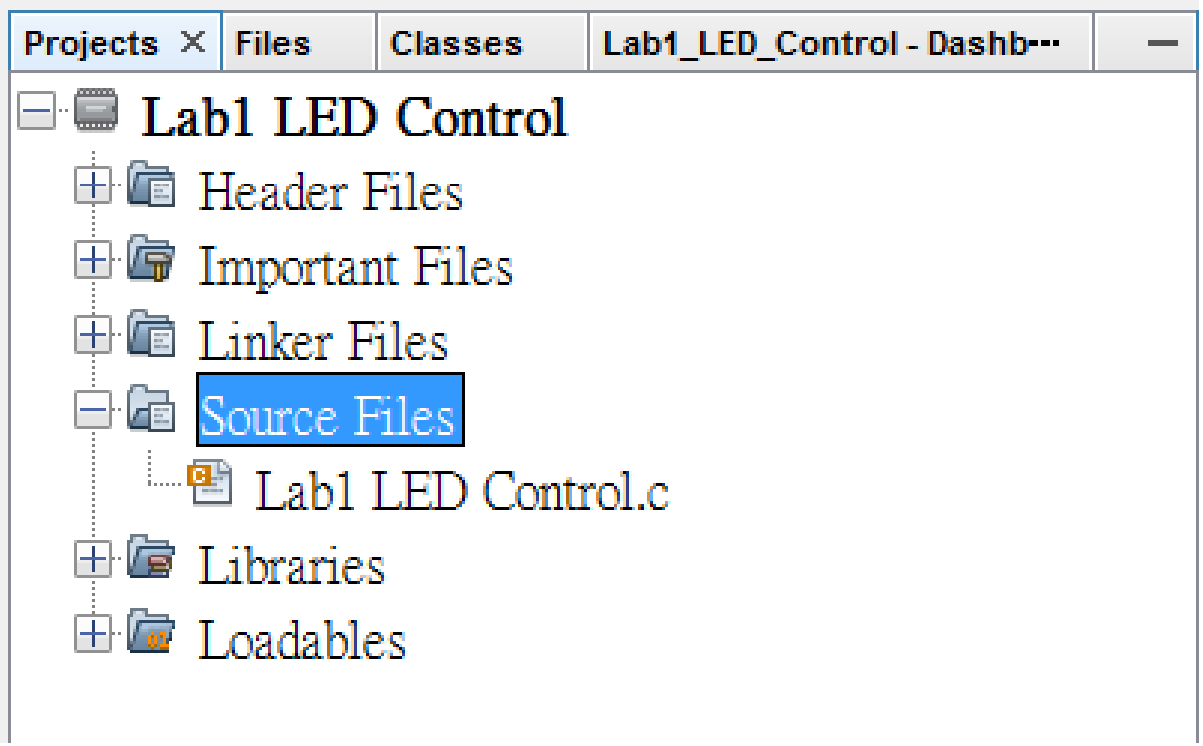
- Header Files
- Source Files
  - Lab1 LED Control.c
- Libraries
  - 暫時不用



建立獨立專案

8 將相關檔案加入到新專案中（續……）

完成後的簡單專案



開始對 Lab1 除錯

9 編譯和執行專案

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：

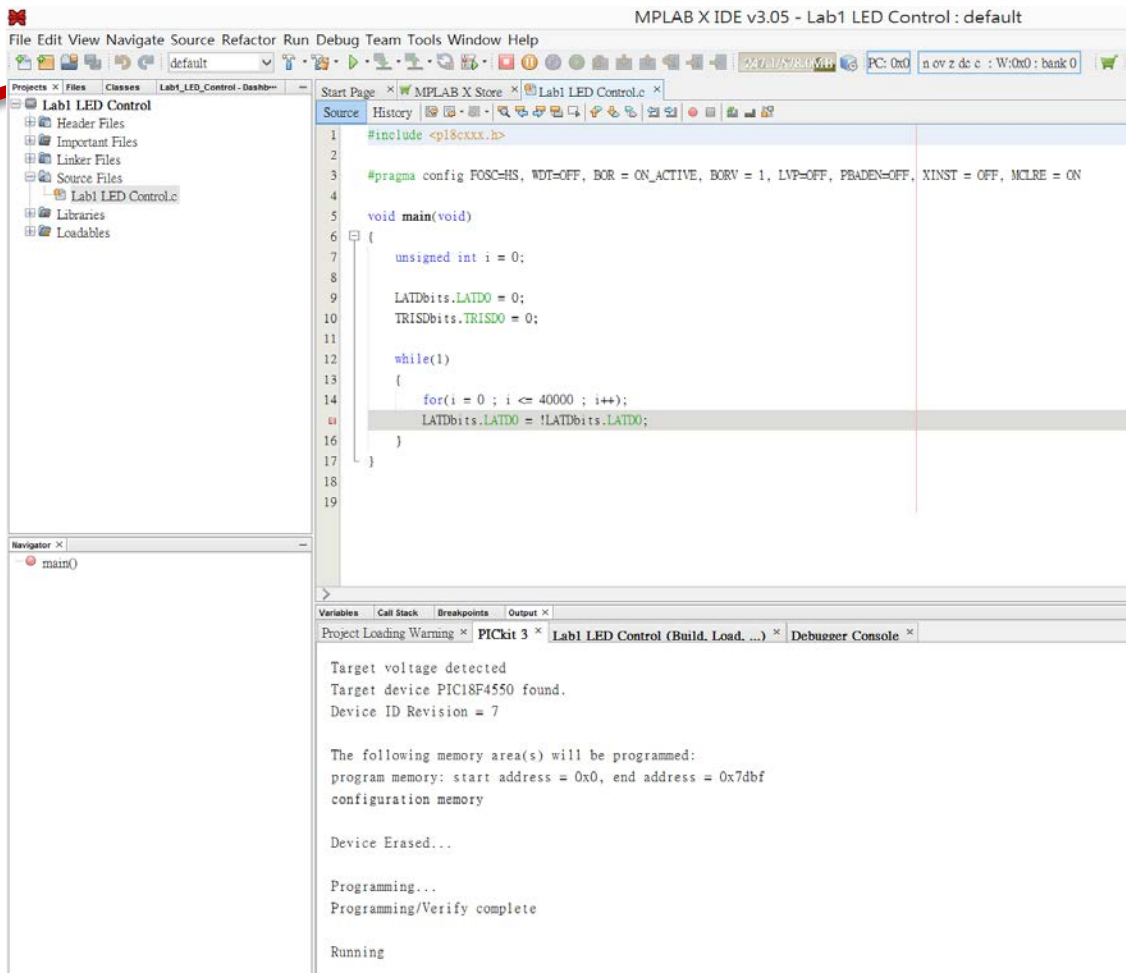
1.在 ***除錯*** 模式下編譯（**make**）專案

2.重新與 PICkit 3連線

3.將程式燒錄至目標板上的 PIC18F4550 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



如何善用編譯應用程式
使用除錯器

如果您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)