



MICROCHIP

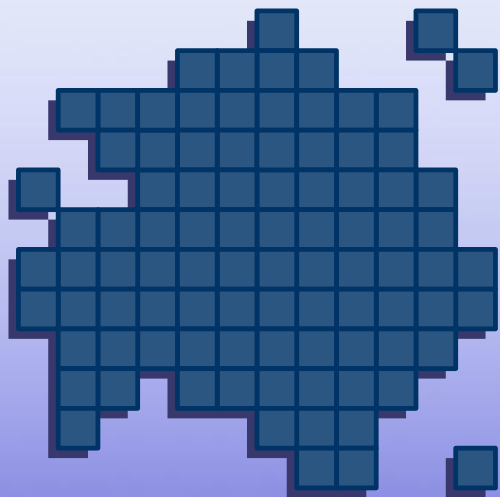
Regional Training Centers

課程編號 : TLS2118

**Microchip PIC18 MCU 架構
與基礎 C18 程式開發**

TLS2118 課程內容

- 開發工具的介紹
- MPLAB C18 語言工具簡介
- 基礎周邊說明 I/O
 - 練習一：建立 MPLAB® C 的專案
- PIC18F 內建的特殊功能
- 如何設定Configuration Bits
- 變數與暫存器的存取
 - 練習二：I/O 控制與函數庫的使用
- 基礎周邊說明 Interrupts
- 基礎周邊說明 Timers
 - 練習三：Timer1 規劃及中斷函數
- 基礎周邊說明 10-bit ADC
 - 練習四：使用 ADC 函數庫轉換
- 參考資料：
 - PIC18F Family 架構
 - C18 Memory Models



Microchip

開發工具簡介

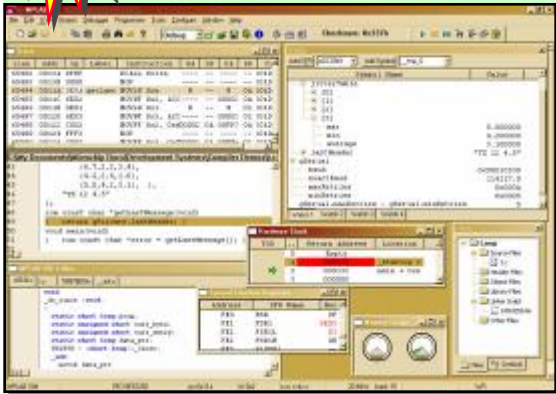
TLS2118 使用工具

- 使用軟體工具
 - MPLAB IDE v8.70 (或更新版本)
 - MPLAB C18 v3.38 Lite Mode (或更新版本)
 - MPASM , MPLINK , MPLIB
- 使用硬體工具
 - MPLAB ICD 3
 - Microchip APP001 Rev. 3 實驗版 (PIC18F4520)
- 參考書籍
 - MPLAB C18 Compiler User's Guide & Library
 - MPASM User's Guide with MPLINK and MPLIB
 - MPLAB IDE User's Guide
 - APP001 Vre. 3 使用手冊

容易入門的開發環境及工具安裝

Download the MPLAB® IDE (Free)

Free



Select a Debugger/ Programmer



Select Your C Compiler



**MPLAB®
CERTIFIED**

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

Free



Free

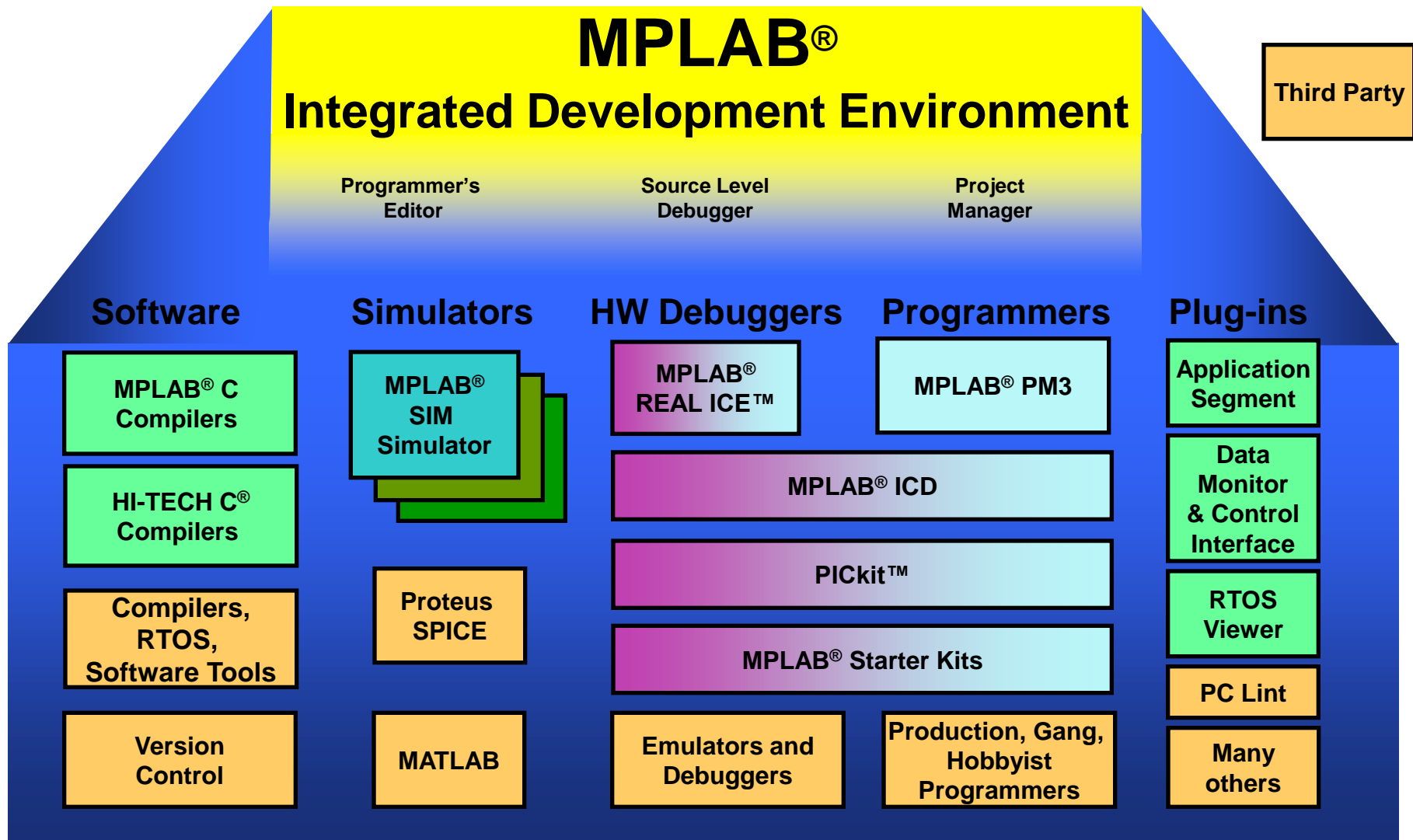
**HI-TECH C®
Lite**

**HI-TECH C®
STD**

**HI-TECH C®
PRO**

Support
for all
PIC®
MCUs

MPLAB® IDE and Components



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
- V_{DD} range 3 - 5.5V
- V_{PP} range 3 - 13V
- CE and RoHS-compliant



MPLAB®
CERTIFIED

MPLAB ICD 3

New In-Circuit Debugger

- **Full PIC portfolio support**
- **Many other Improvements**
 - Can provide limited target power (100mA)
 - Fast – USB HS, HW accelerator, SRAM
 - SW breakpoints (1000)
- **Completely USB powered**
- **No external power supply required**
- **High Voltage protection on ICSP**
- **Over Current Protection**
- **Test Interface Module**
 - Validate interface and connection integrity
 - Helps isolate target faults vs. tool issues
 - Test module includes small PIC



MPLAB[®] REAL ICE[™] Probe Kit

- **MPLAB IDE compatible**
- **Standard Driver Board**
- **Complex Breakpoints**
- **Real Time Data Watchpoints**
- **Stack/WDT/Sleep Breakpoints**
- **USB High Speed Connection**



DV244005 MPLAB REAL ICE List \$499.98

各式除錯工具基本功能 比較表

Feature	PICKit 2	ICD 2	PICKit 3	ICD 3	REAL ICE
USB Speed	Full	Full	Full	High and Full	High and Full
USB Driver	HID	Custom	HID	Custom	Custom
USB powered	Yes	Yes	Yes	Yes	Yes
External power supply required	No	Yes	No	No	No
Programmable VPP Range	(3-13V)	(10-14V)	(2-14v)	(2-14V)	(2-14V)
VPP Current Supplied	30ma	30ma	30ma	100ma	100ma
Power to Target	Yes	5V only	Yes	Yes	No
Programmable VDD	3 -5V	No	2-5V	2-5V	2-5V internal
VDD Current Supplied	30ma	200ma	30ma	100ma	100ma
VDD drain from target	20ma	20ma	20ma	<1ma	<1ma
Over voltage Protection	No	Yes	Yes	Yes	Yes
Over current Protection	Yes	Yes	Yes	Yes	Yes
Emulation Support	Full Speed	Full Speed	Full Speed	Full Speed	Full Speed
Software Breakpoints	No	No	No	Yes	Yes
Program Image	128K bytes	No	512K bytes	No	No

除錯工具功能選擇表

 有支援

 有限支援

 無法支援

Selection criteria / Recomm. tool	Advantage / comment	Pickit2	Pickit3	ICD2	ICD3	REAL-ICE
Debug capability		limited	limited			
Can program all PICs & dsPICs				adap. Need		
Fast Project Development	Faster daily debug / programming					
Tool cost						
PIC10/12/16 (except PIC16E)						
PIC16 Enhanced	Need Vpp = 8.5V					
PIC18 std (non J, non K)						
PIC18FxxJxx						
PIC18FxxKxx	Need Vpp = 8.5V					
PIC18 fast programming	max 15s up to 128KB					
16 bits fast programming	max 15s up to 256KB					
32 bits fast programming	max 15s up to 512KB					
Programmer-To-Go	Standalone programming					
Real-Time variables watch	PIC18J, 16/32 bits except dsPIC30					
Real-Time measurement of program execution time (STOPWATCH)	PIC18, 16/32 bits except dsPIC30					
SW breakpoints	PIC18, 16/32 bits except dsPIC30					
Real-Time Data Monitoring (RTDM)	dsPIC30/dsPIC33 PIC24H					
TRACE (Instructions)	PIC32 only					
TRACE (Ports / SPI)	PIC18, 16 bits (not PIC32)					
External breakpoint Trigger	For all PICs					

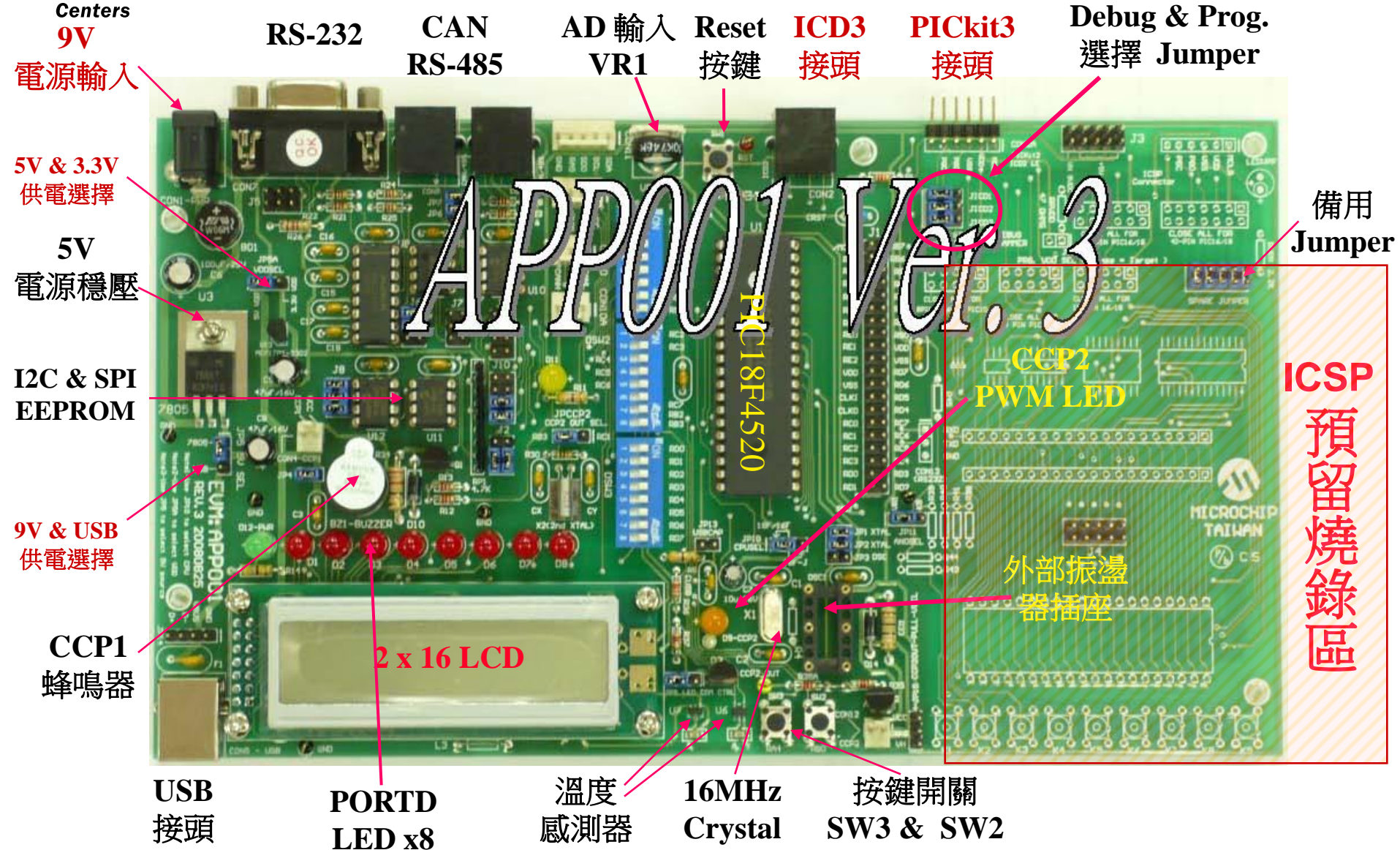
APP001 實驗板使用手冊

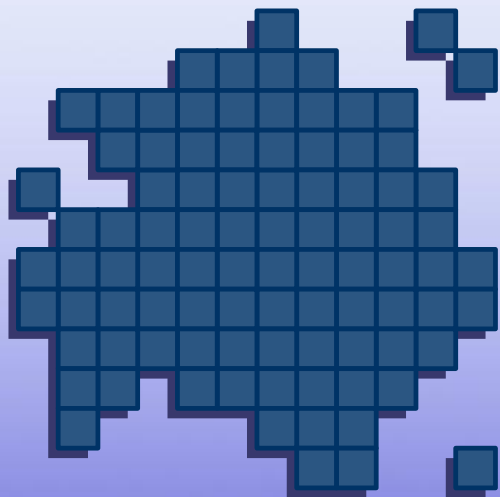
- **APP001 實驗板的使用手冊、電路圖及出廠測試程式可以在 **Microchip** 台灣的網站下載:**

<http://www.microchip.com.tw/TaiwanCAE/EVM/Pages/Index1.htm>

- **APP001 實驗板有三種版本 v1.0, v2.0 及最新的 v3.0；本教材均可適用**

認識 APP001 REV.3a 實驗板





MPLAB[®] C18

語言工具簡介

MPLAB® C18

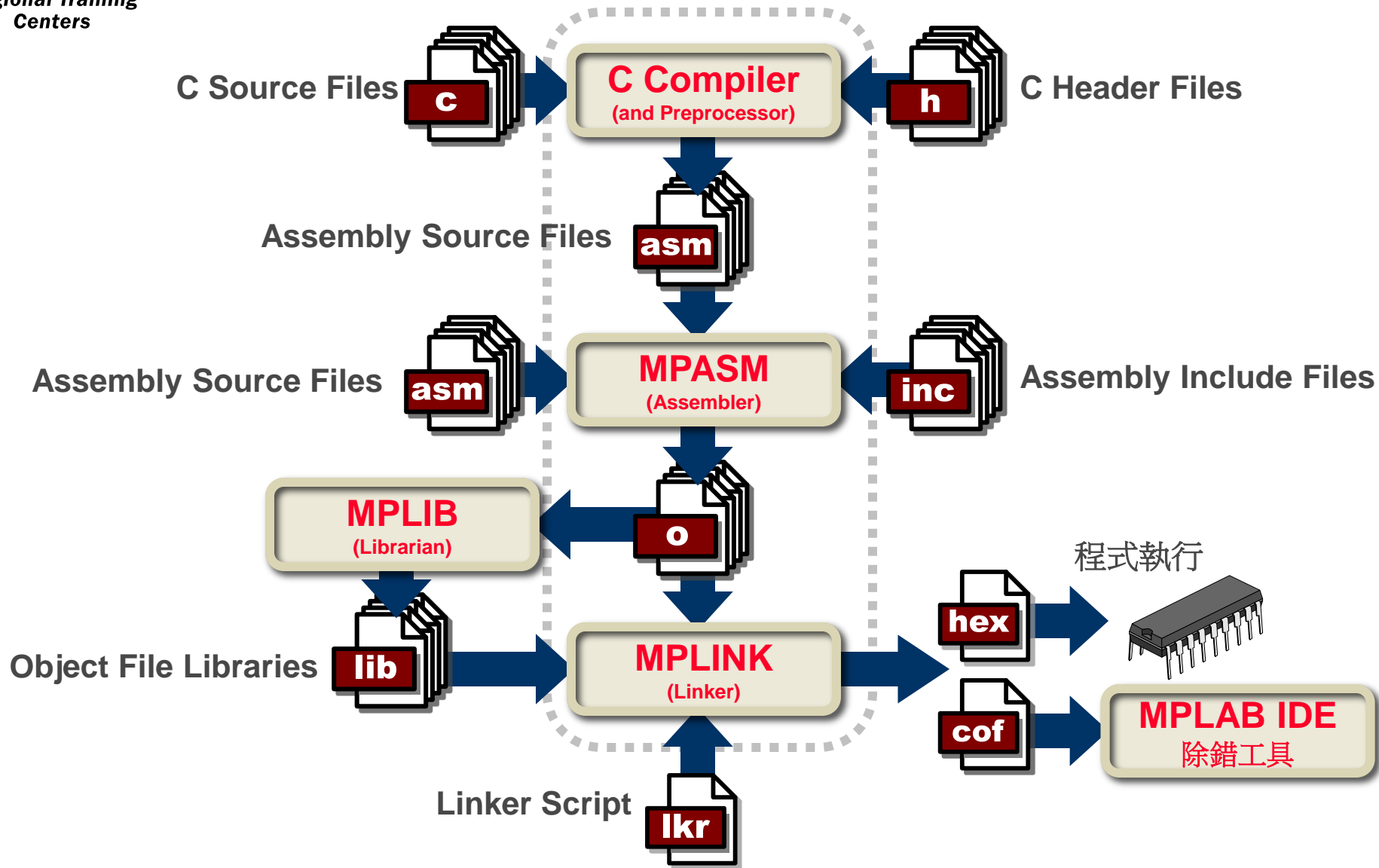
概要說明

- 相容於 **ANSI x3.159-1989** 的 **C** 編譯器
- 具有最佳化處理功能
- 加入一些為 **PIC18** 特定周邊功能
- 跟 **MPLAB® IDE** 結合為一體
- 相容於 **MPASM** 及 **MPLINK**



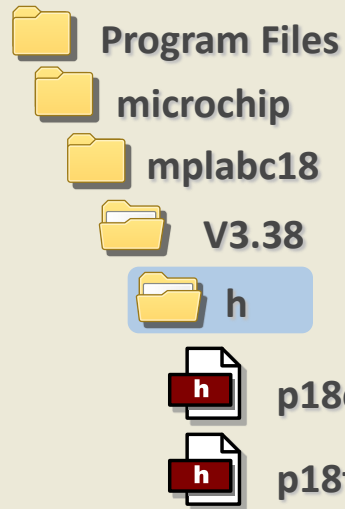
MPLAB® C for PIC18 Lite /Evaluation 的版本可以免費自 **Microchip** 的官網下載安裝。

C18 編譯流程

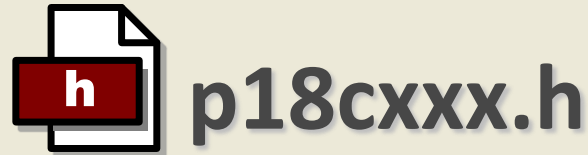


MPLAB[®] C for PIC18

標頭檔 (Header Files)



Header files 包含安裝 **MPLAB C18** 後所支援的所有的元件的名稱定義：



p18cxxx.h 是一個通用型的標頭檔，他會依據 **MPLAB IDE** 所傳進來的使用元件料號加以判斷後，加入該元件的名稱定義標頭檔 (**p18f4520.h**)。

- 各元件的標頭檔提供了：
 - 提供所有暫存器的定義名稱
 - 定義周邊暫存器的位元操作名稱
 - 定義巨集指令以方便 **C** 來操作一些特殊指令

p18f4520.h 定義檔

- 定義**18F4520** 特殊功能暫存器(SFR) 的名稱及相關位元名稱

➤ 位置 :C:\Program Files
\\Microchip\mplabc18\v3.38\h

- 定義一些常用的巨集

➤ `Nop()`
➤ `ClrWdt()`
➤ `Sleep() ...`

- 使用時用 **#include**
<p18fxxx.h>

➤ **P18fxxx.h** 為通用性標頭檔，
他會一 **MPLAB IDE** 傳入的
元件名稱自行判斷加入該
元件的 **h** 檔。

有關**ADCON0**暫存器之定義

```
extern near unsigned char ADCON0;  
extern near union {  
    struct {  
        unsigned ADON:1;  
        unsigned :1;  
        unsigned GO:1;  
        unsigned CHS0:1;  
        unsigned CHS1:1;  
        unsigned CHS2:1;  
        unsigned ADCS0:1;  
        unsigned ADCS1:1;  
    } ;  
    struct {  
        unsigned :2;  
        unsigned NOT_DONE:1;  
    } ;  
    struct {  
        unsigned :2;  
        unsigned DONE:1;  
    } ;  
    struct {  
        unsigned :2;  
        unsigned GO_DONE:1;  
    } ;  
} ADCON0bits ;
```

為何要用
extern 的宣告?

C18 的位元定義 (p18f4520.h)

- 所有的周邊暫存器及相關位元均在各標頭檔 (**Header file**) 中定義
 - 例: **PIC18F4520** 的各種周邊是定義在 “ p18f4520.h ”
- 撰寫程式時，只要按照 **Data Book** 所標示的名稱使用。對應的位元結構為暫存器名稱加上 “**bits**”

1. 設定 **PORTD** 為輸出，並將 **0x55** 的16進制值輸出到 **PORTD**：

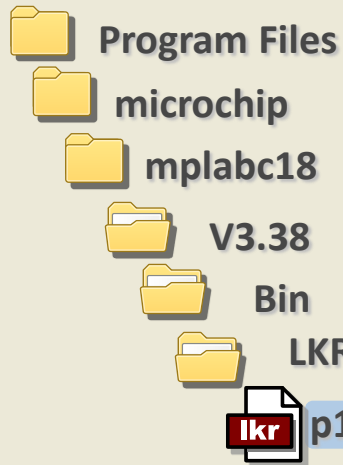
```
TRISD = 0 ;  
PORTD = 0x55 ;
```

2. 將 **PORTD** 的 bit3 & bit5 設定為 1：

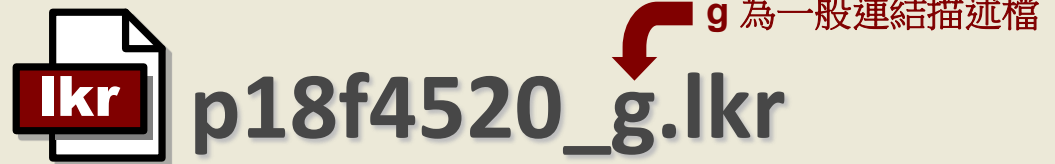
```
PORTDbits.RD3 = 1 ;  
PORTDbits.RD5 = 1 ;
```

MPLAB® C for PIC18

連結描述檔 (Linker Scripts)



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



- 每一個元件都有相對應連結描述檔：
 - 定義該元件的記憶大小、節區及邊界位址
 - 定義 **debug** 和 **release** 的工作模式
- **MPLAB IDE** 將自動選用連結描述檔，除非你有特別的指定使用

18f4520_g.lkr 描述檔

LIBPATH .

FILES c018i.o
FILES cliib.lib
FILES p18f4520.lib

將所要用到函數庫加入

程式空間

```
#IFDEF _DEBUGCODESTART
CODEPAGE NAME=page START=0x0 END=_CODEEND
CODEPAGE NAME=debug START=_DEBUGCODESTART END=_CEND PROTECTED
#ELSE
CODEPAGE NAME=page START=0x0 END=0x7FFF
#FI

CODEPAGE NAME=idlocs START=0x200000 END=0x200007 PROTECTED
CODEPAGE NAME=config START=0x300000 END=0x30000D PROTECTED
CODEPAGE NAME=devid START=0x3FFFFE END=0x3FFFFF PROTECTED
CODEPAGE NAME=eedata START=0xF00000 END=0xF000FF PROTECTED
```

資料空間

```
ACCESSBANK NAME=accessram START=0x0 END=0x7F
#FI

DATABANK NAME=gpr0 START=0x80 END=0xFF
DATABANK NAME=gpr1 START=0x100 END=0x1FF
DATABANK NAME=gpr2 START=0x200 END=0x2FF
DATABANK NAME=gpr3 START=0x300 END=0x3FF
DATABANK NAME=gpr4 START=0x400 END=0x4FF

#IFDEF _DEBUGDATASTART
DATABANK NAME=gpr5 START=0x500 END=_DATAEND
DATABANK NAME=dbgspr START=_DEBUGDATASTART END=_DEND PROTECTED
#ELSE //no debug
DATABANK NAME=gpr5 START=0x500 END=0x5FF
#FI

ACCESSBANK NAME=accesssfr START=0xF80 END=0xFFFF PROTECTED
```

軟體堆疊
空間

```
#IFDEF _DEBUGDATASTART
STACK SIZE=0x100 RAM=gpr4
#ELSE
STACK SIZE=0x100 RAM=gpr5
```

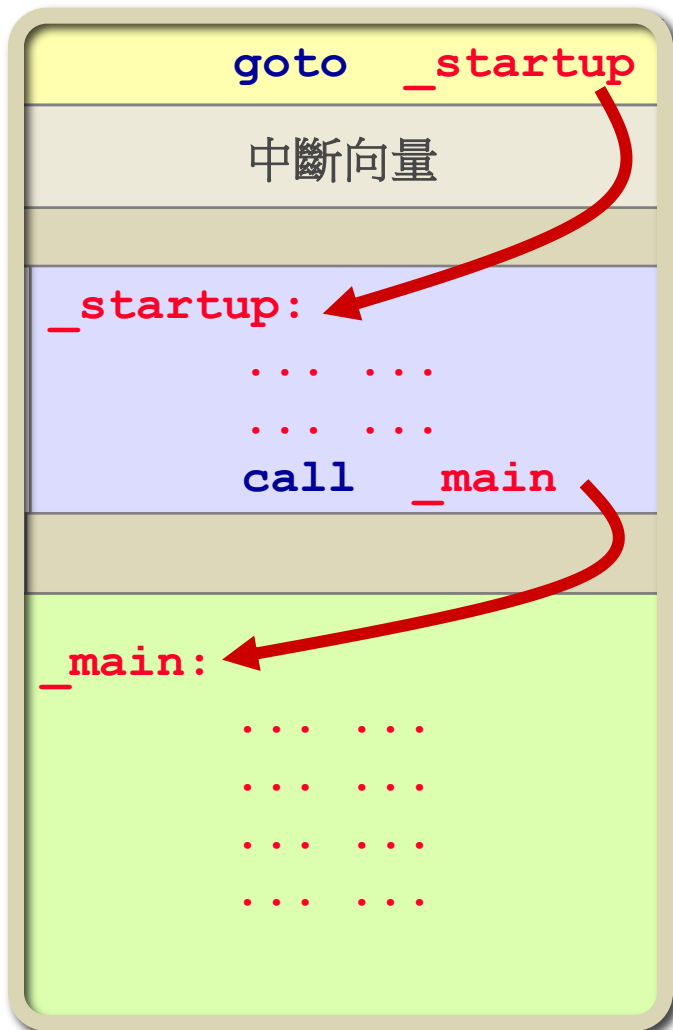
C 的 Linker 節區

Section Type	Default Name	Purpose
code	<code>.code_filename</code>	程式執行碼
romdata	<code>.romdata_filename</code>	程式記憶體의常數資料
udata	<code>.udata_filename</code>	無初始值的變數
idata	<code>.idata_filename</code>	有初始值的變數
stack (udata)	<code>.stack</code>	C 程式執行的軟體堆疊

- **Linker** 會對每一個輸入檔案 (*.o) 建立節區
- 所以你可以建立你自己的節區，並將相關的節區整合在一起放在記憶體裡

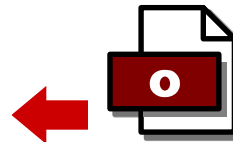
MPLAB[®] C for PIC18

啟動與初始設定



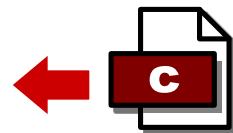
Reset Vector

MPLINK 會自動加入適當的啟動模組，一般是呼叫 `c018i.o` (`_startup` label)



`c018i.o` (內定使用)

C 啟動模組主要是做初始化的設定、軟體堆疊、指標設定等工作，設定完畢後再將控制權交給 `main()` (原始程式碼：`c018*.c`)



`main.c`

你的 `main()` 函數，Linker 會自動安排執行位址的。

C 的啟動及初始設定

如何使用正確的啟動模組

如有使用到其它的啟動模組可以修改 **lkr** 檔裡的啟動模組的名稱

```
// File: 18f4520i.lkr  
// Sample ICD2 linker script
```

```
LIBPATH .
```

```
FILES c018i.o  
FILES TLS2118.lib  
FILES clib.lib  
FILES p18f4520.lib
```

CODEPAGE	NAME=page	START=0x0
CODEPAGE	NAME=debug	START=0x7D
CODEPAGE	NAME=idlocs	START=0x20
CODEPAGE	NAME=config	START=0x30
CODEPAGE	NAME=devvid	START=0x3F
CODEPAGE	NAME=eedata	START=0xF0

修改啟動模組，共有六項可選用：

使用 PIC18F 傳統 指令集

c018i.o	idata & udata
c018iz.o	idata & zeroed udata
c018.o	udata only

使用 PIC18F 擴展型指令集

c018i_e.o	idata & udata
c018iz_e.o	idata & zeroed udata
c018_e.o	udata only

c018i.c 啟動模組程式

```
#pragma code _entry_scn = 0x000000  
static void  
entry (void)  
{ _asm goto _startup _endasm }
```

RESET 位址:
0x000000

```
#pragma code _startup_scn  
static void _startup (void)  
{  
  _asm  
  // Initialize the stack pointer  
  LFSR 1, _stack LFSR 2, _stack CLRFB TBLPTRU, 0  
  // Initialize rounding flag for floating point libs  
  BSF FPFLAGS,RND,0  
  _endasm
```

**給予 STACK 及
TBLPTRU 初使值**

```
_do_cinit ( );
```

設定初始變數值

```
loop:  
  // Call the user's main routine  
  main ( );  
  goto loop;  
} /* end _startup() */
```

將控制權交至 main()

p18f4520.lib 周邊函數庫檔

- 定義 **18F4520** 所有的特殊周邊暫存器 (SFR) 的位址
 - p18F4520.asm 組譯後以 obj 的型態存在於 p18f4520.lib
 - 原始程式為組合語言型態放在
 - C:\Program Files\Microchip\mplabc18\v3.38\src\traditional\proc
- **p18f4520.lib** 也提供各周邊控制函數
 - 相關的周邊控制函數，各自分門別類的以 obj 型態存在於 p18f4520.lib
 - 周邊控制函數請參閱 MPLAB C18 Reference Manual
 - A/D , USART , Timer x , EEPROM ... 等
 - 原始程式在 C:\Program Files\Microchip\mplabc18\v3.38\src\pmc_common 的目錄下

PIC18F4520 三個元件支援檔

有關特殊暫存器之位址定義

- **p18f4520.o (asm)**

- 定義周邊暫存器的位址給 C18 使用

```
LIST P=18F4520
NOLIST
```

```
-----
; MPLAB-Cxx PIC18F4520 processor definition module
;
; (c) Copyright 1999-2007 Microchip Technology,
;-----
```

- **p18f4520.h**

- 定義周邊暫存器及相關位元名稱

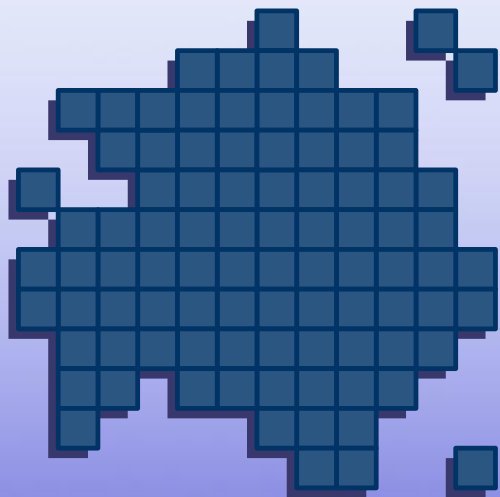
```
SFR_UNBANKED0          UDATA_ACS H'F80'

PORTA
PORTAbits               RES 1      ; 0xF80
PORTB
PORTBbits               RES 1      ; 0xF81
PORTC
PORTCbits               RES 1      ; 0xF82
PORTD
PORTDbits               RES 1      ; 0xF83
PORTE
PORTEbits               RES 1      ; 0xF84
                       RES 4

LATA
LATAbits                RES 1      ; 0xF89
LATB
LATBbits                RES 1      ; 0xF8A
LATC
LATCbits                RES 1      ; 0xF8B
LATD
LATDbits                RES 1      ; 0xF8C
LATE
LATEbits                RES 1      ; 0xF8D
```

- **p18f4520.inc**

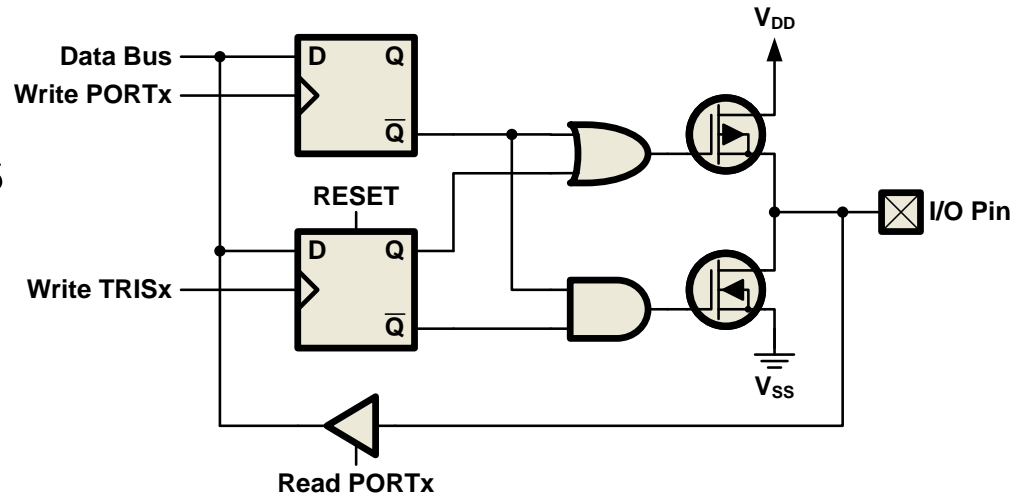
- 是給組合語言使用的周邊暫存器定義含入檔，不可混淆



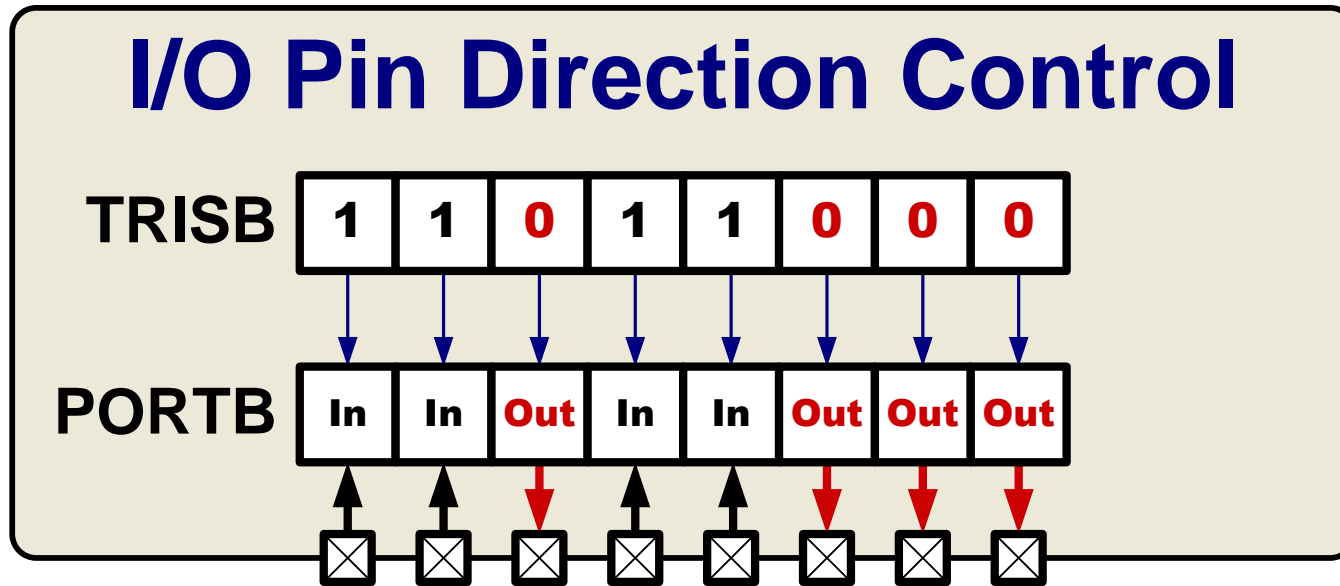
PIC18 Family 內建的 基礎周邊說明 - I/O

I/O Ports

- **High Drive Capability**
- **Can directly drive LEDs**
- **Direct, single cycle bit manipulation**
- **Each pin has individual direction control under software**
- **All pins have ESD protection diodes**
- **Pin RA4 is usually open drain**
- 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

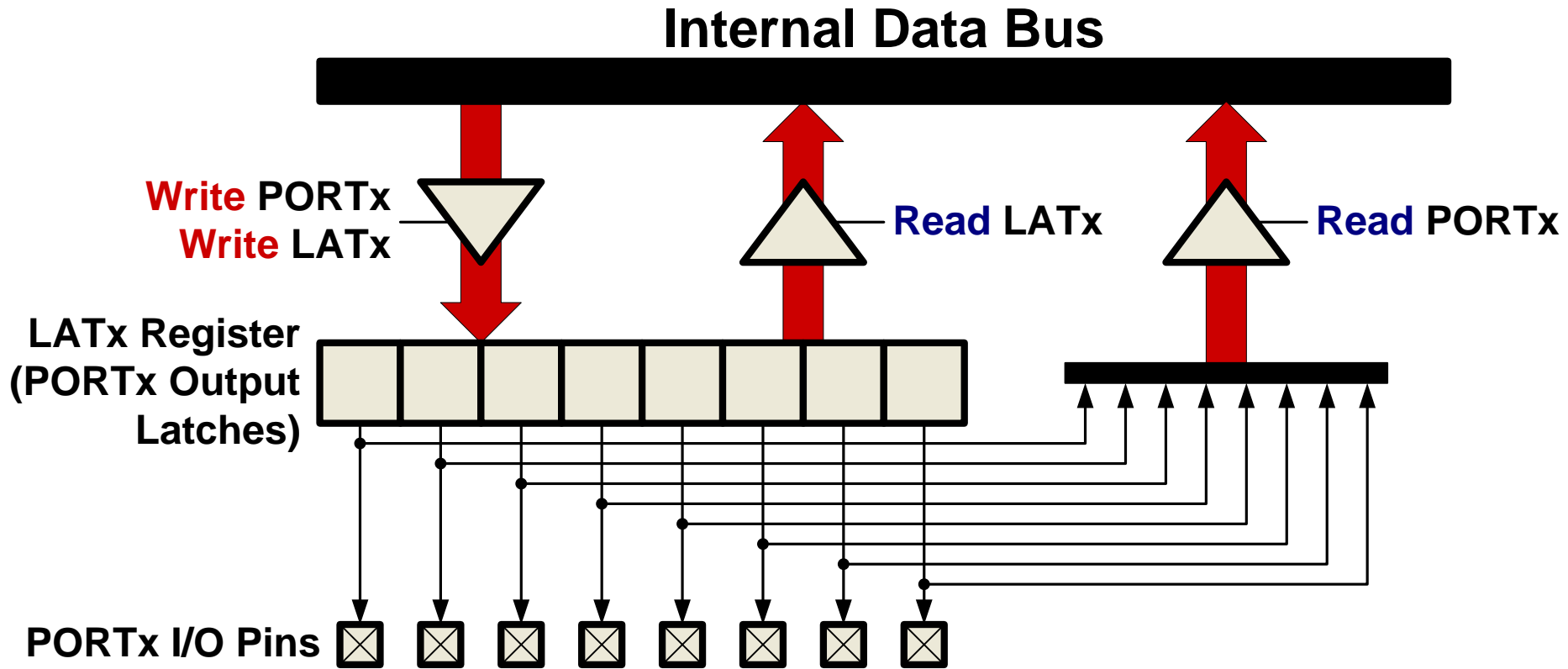


I/O Ports



- 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

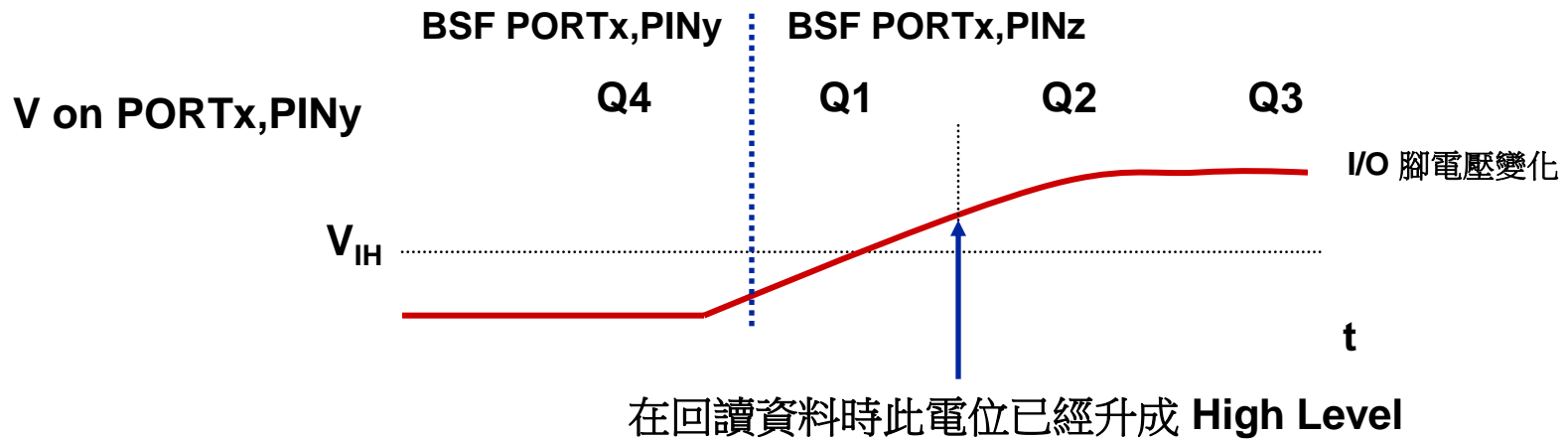


注意一下：**PORTx** 與 **LATx** 在做輸出與輸入時，功能上有何不同？

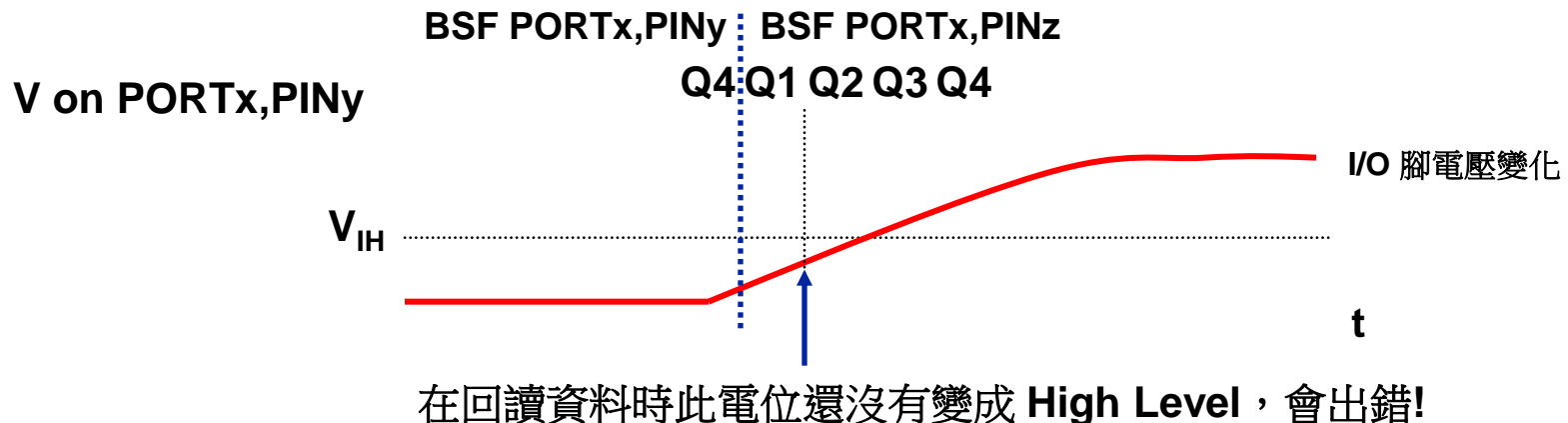
為何要加入 LATx 的暫存器

消除 Read-Modify-Write 動作的風險

在低工作頻率及較小的雜散電容



在高工作頻率及較大的雜散電容

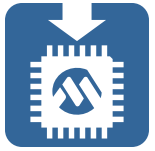


Analog or Digital I/O?

- 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 **PIC18F4520** 中, **ADCON1** 被用來設定這些共用接腳的操作模式為何.

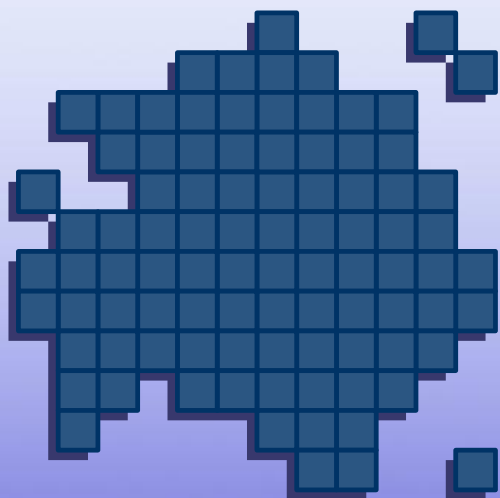
Bit 7				Bit 0			
-	-	X	X	1	1	1	1
VCFG1		VCFG0		PCFG3		PCFG2	
				PCFG1		PCFG0	

Set lower 4 bits to '1' to make all multiplexed pins digital



Summary

- 在 PIC18 Family 中使用於 I/O 控制的暫存器有
 - TRIS_x
 - 控制 I/O port 的方向
 - PORT_x
 - 對 I/O port 讀取/寫入用的暫存器
 - LAT_x
 - 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



練習一

請照著後面的投影片的順序
建立新的專案、編譯、燒錄



Lab Exercise 1

建立一個 MPLAB® C18 程式專案



練習一目標

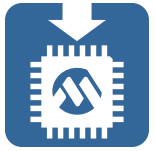
建立第一個 **MPLAB C18** 的專案，使用 **C18** 來編譯此專案並確定成功後執行程式可以正確的看到 “**Hello, TLS2118!**” 的字串顯示在 **APP001 Demo Board** 的 **LCD**。

Hello, TLS2118

這是一步一腳印的練習，使用實際範例按步就班的建立一個 **MPLAB C18** 的基本專案。

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

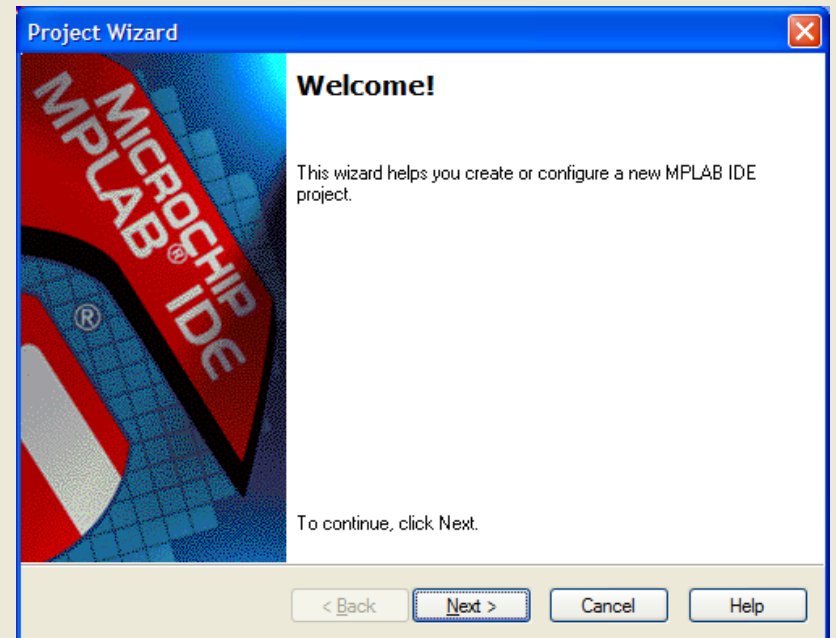
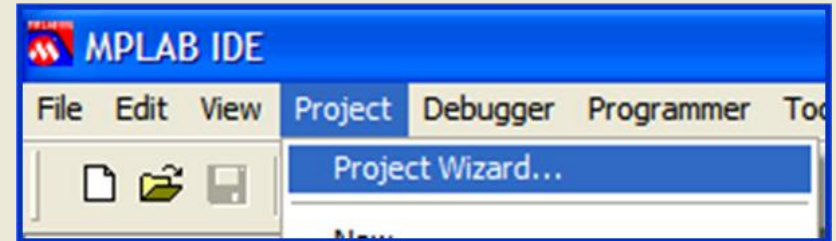


步驟

1

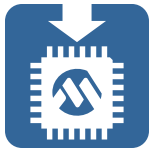
在 MPLAB IDE 下的 Project 選項
下，開啟專案精靈：
Project ► Project Wizard...

After the Project Wizard opens,
Click **Next >** to continue...



Lab Exercise 1

建立一個 MPLAB® C18 程式專案

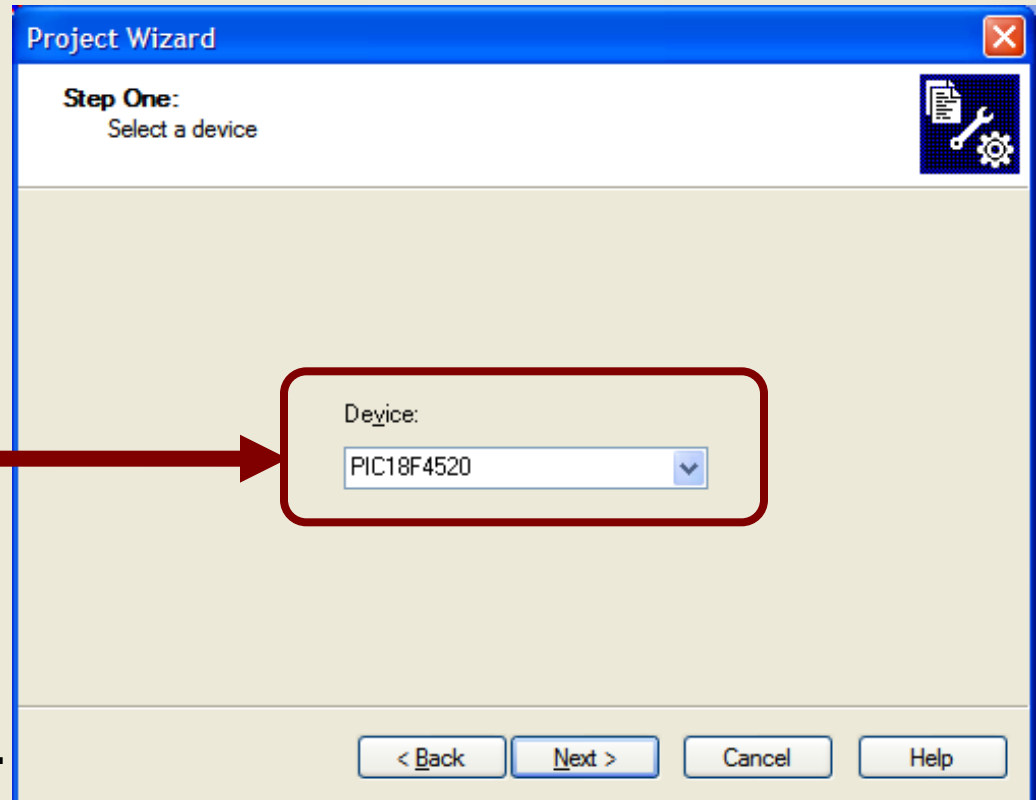


步驟

2

選擇正確的使用元件名稱：

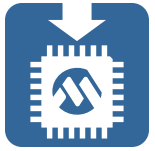
PIC18F4520



Click **Next >** to continue...

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

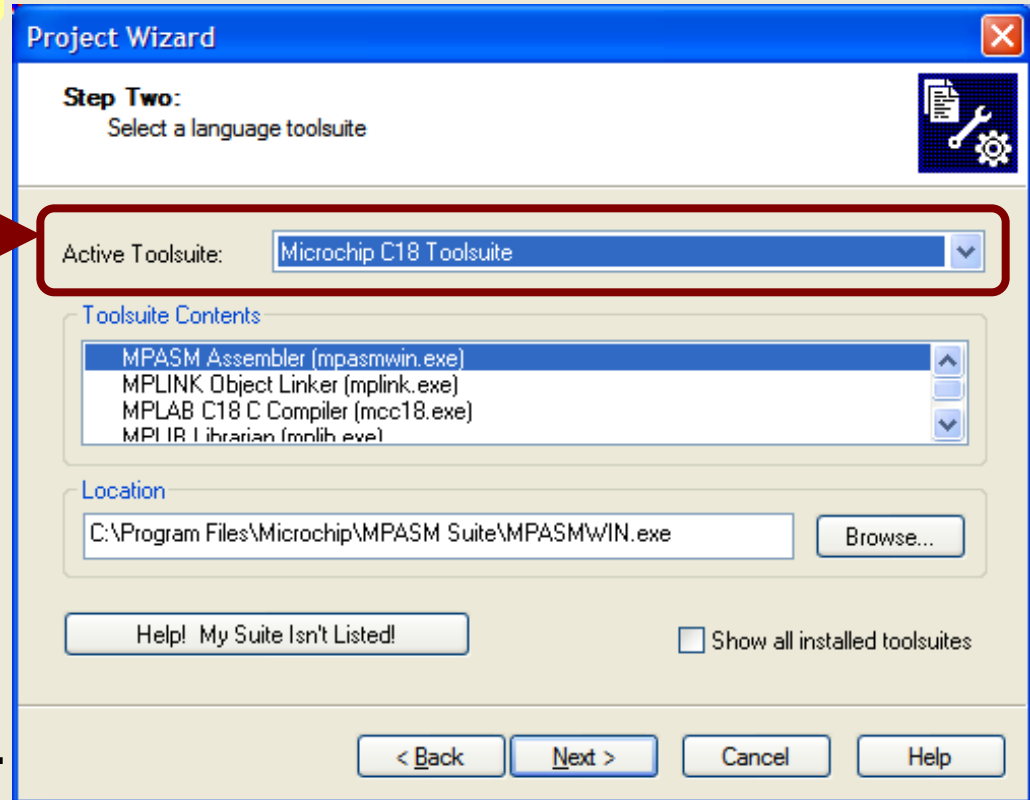


步驟

3

在已安裝編譯器工具箱 (Toolsuite) 選擇：

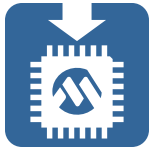
Microchip C18 Toolsuite



Click **Next >** to continue...

Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

4

按下

Browse...

給予專案名稱 及路徑：

C:\RTC\TLS2118\Lab1

專案的名稱為 Lab1.mcp

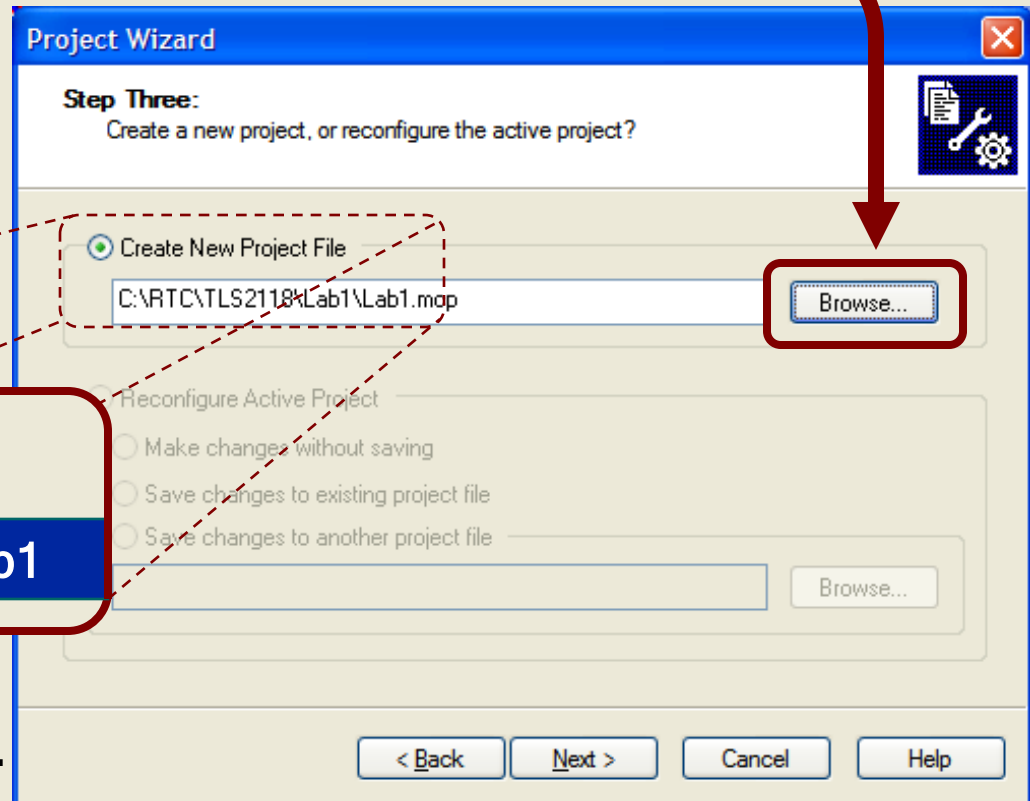
☒ Create New Project File

C:\RTC\TLS2118\Lab1\Lab1

Click

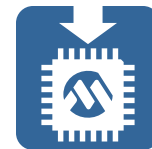
Next >

to continue...



Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

- 5** 加入原始程式到專案裡，在左邊表列裡可以選擇：

C:\RTC\TLS2118\Lab1

選擇 **Lab1.c**

按下

Add >>

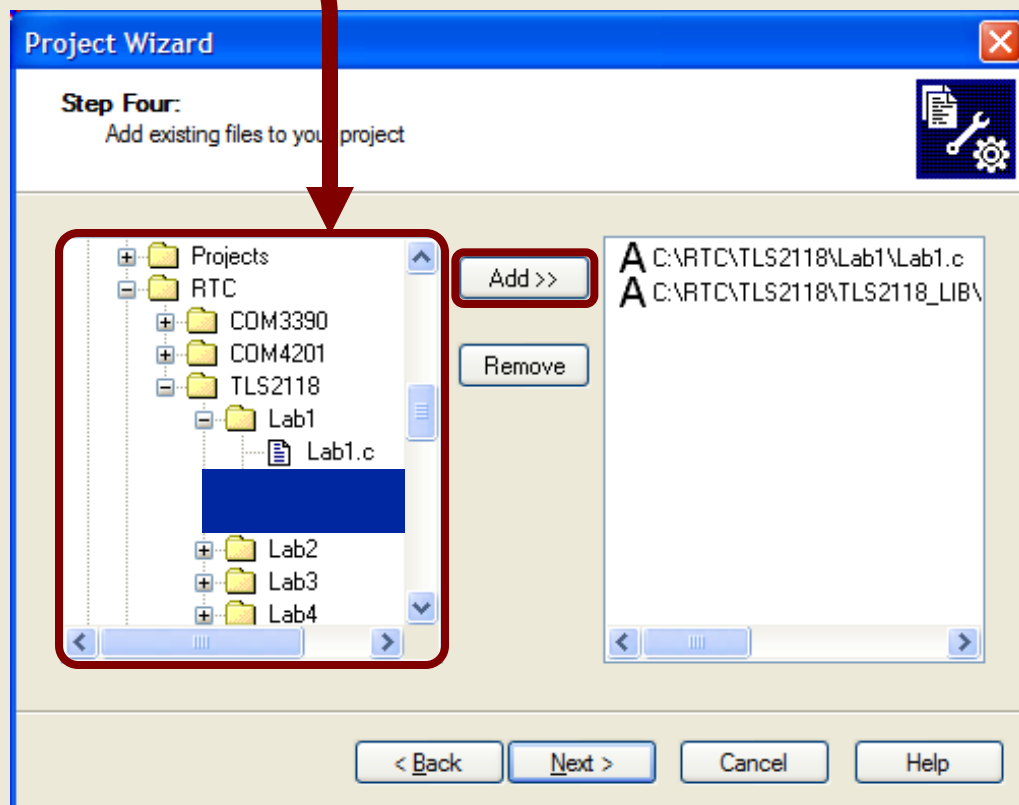
- 6** 接著到底下的路徑加入所要使用的 **Library**：

C:\RTC\TLS2118\TLS2118_LIB

選擇此檔案 **TLS2118.lib**

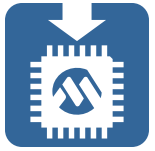
Click

Add >>



Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

- **MPLAB** 在較新的版本會自行為使用者選擇適當的連結器描述檔 (**LKR** 檔毋需指定)
- 使用者若自行加入連結器描述檔，將會取代原先 **MPLAB** 的預設選擇。
- 使用者可以為自己建立的專案加入一個連結器描述檔 – **Linker Script** (例如：**p18f4520_g.lkr**)
- 一般而言，除非使用者要修改程式區間的分配作業才須自行指定連結器描述檔

Lab Exercise 1

建立一個 MPLAB® C18 程式專案



連結器描述檔的內容範例（主要做為可用記憶區塊的宣告）

```
#IFDEF _DEBUGCODESTART
```

CODEPAGE	NAME=page	START=0x0	END=_CODEEND	
CODEPAGE	NAME=debug	START=_DEBUGCODESTART	END=_CEND	PROTECTED

```
#ELSE
```

CODEPAGE	NAME=page	START=0x0	END=0x7FFF	
----------	-----------	-----------	------------	--

```
#FI
```

CODEPAGE	NAME=idlocs	START=0x200000	END=0x200007	PROTECTED
CODEPAGE	NAME=config	START=0x300000	END=0x30000D	PROTECTED
CODEPAGE	NAME=devid	START=0x3FFFFE	END=0x3FFFFFF	PROTECTED
CODEPAGE	NAME=eedata	START=0xF00000	END=0xF000FF	PROTECTED

```
#IFDEF _EXTENDEDMODE
```

DATABANK	NAME=gpre	START=0x0	END=0x5F
ACCESSBANK	NAME=accessram	START=0x60	END=0x7F

```
#ELSE
```

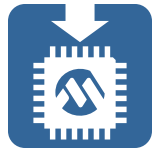
ACCESSBANK	NAME=accessram	START=0x0	END=0x7F
------------	----------------	-----------	----------

```
#FI
```

DATABANK	NAME=gpr0	START=0x80	END=0xFF
DATABANK	NAME=gpr1	START=0x100	END=0x1FF

Lab Exercise 1

建立一個 MPLAB® C18 程式專案

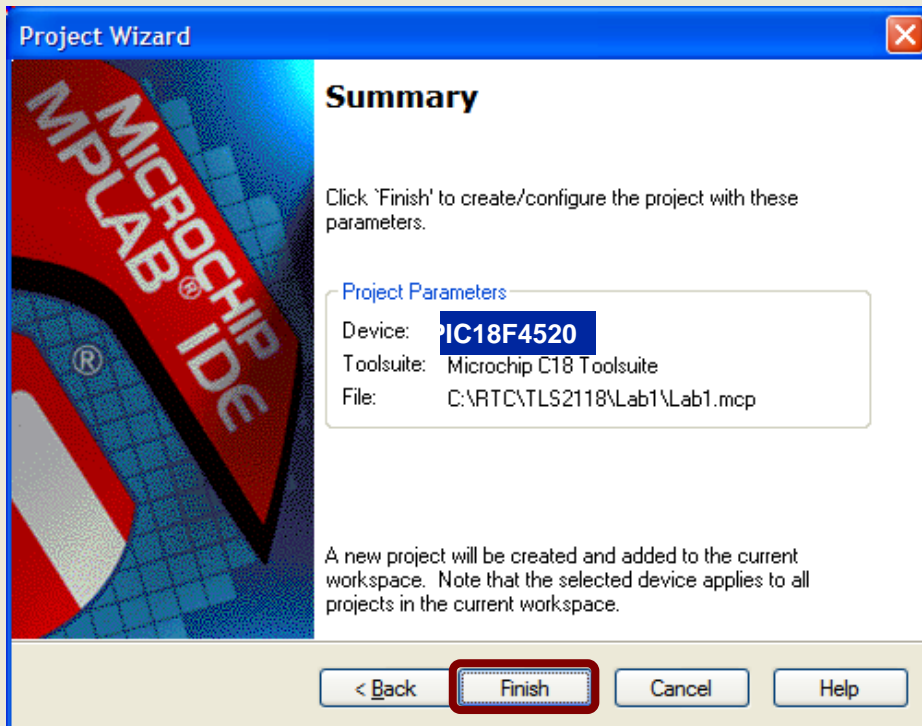


步驟

7

按下

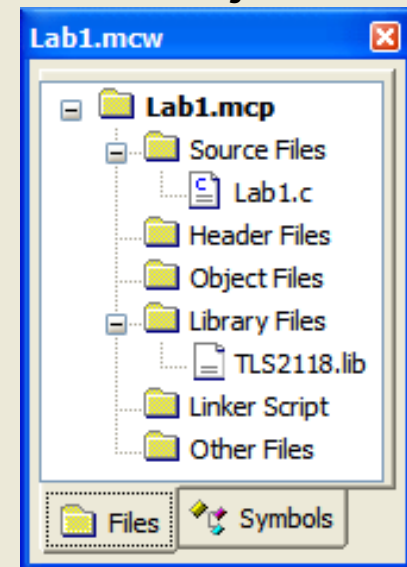
Finish



這時將可以看到專案視窗
如無法看到專案視窗：

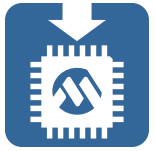
View ► Project

Lab 1 Project Tree



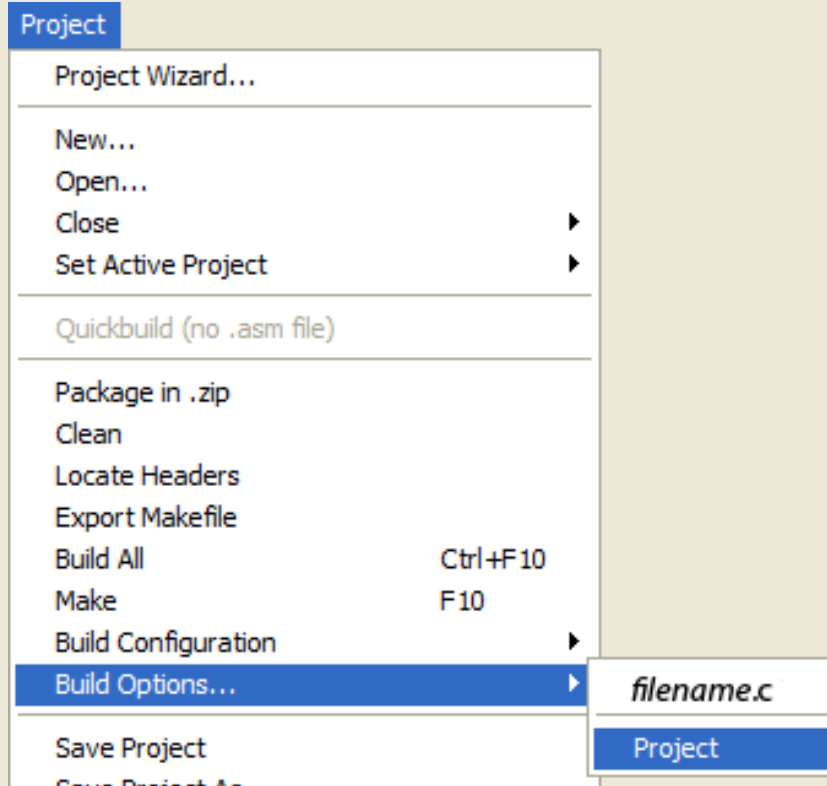
Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

8 設定專案內的編譯選項 **Project ► Build Options ► Project**

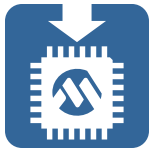


記住：
這個選項將會開啟該專案
選項的對話視窗，你可以
用它來變更編譯器的設定

這個編譯選項在本課程中
將常用到

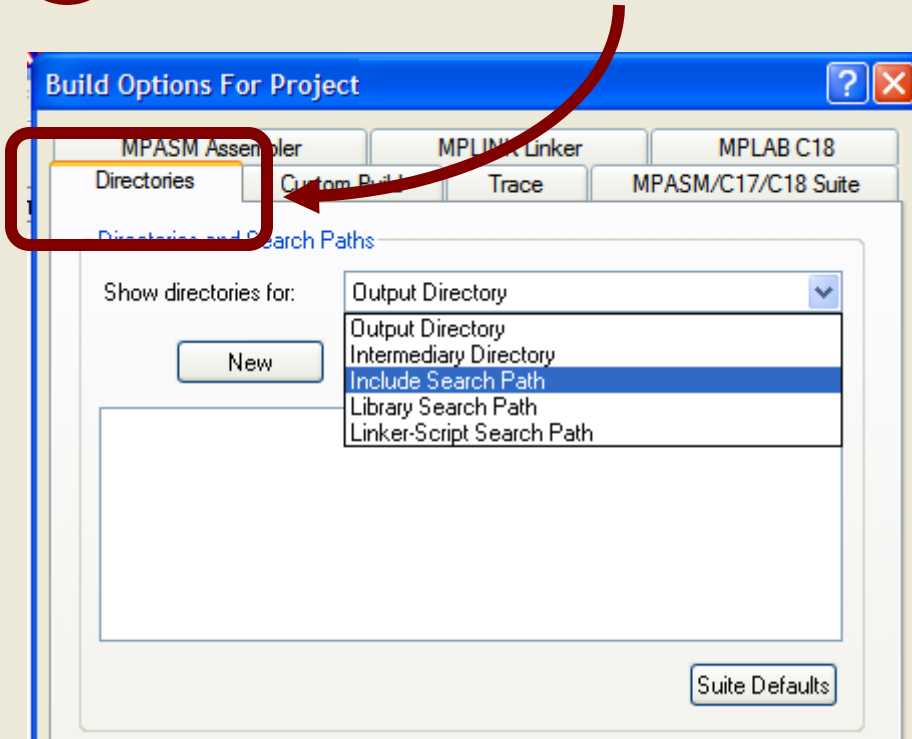
Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

9 選擇 **Directories** 對話項



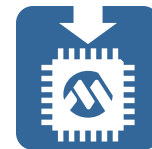
現在要設定在專案裡所使用到的檔案路徑，這樣編譯才可以成功

所有的練習的預設路徑都放在：
C:\RTC\TLS2118 的目錄下

練習一的 **Project** 擺在：
C:\RTC\TLS2118\lab1\lab1.mcp

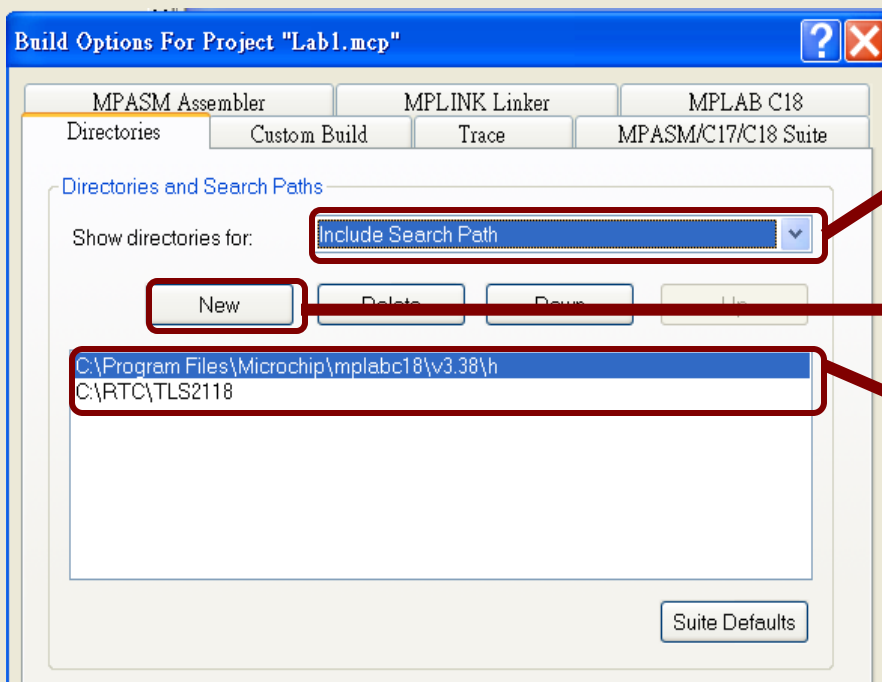
Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

10 設定 **Include** 搜尋路徑



a Select **Include Search Path** from the combo box

b Click on **New**

c 加入搜尋路徑：..**Microchip\mplabc18\v3.38\h** (Default 尋找路徑)
C:\RTC\TLS2118 (加入新的搜尋)



TLS2118 的課程除了內定的標頭檔路徑外還必須指定自行加入的標頭檔路徑

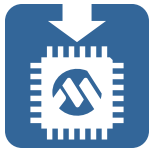
這些 Header 路徑有何不同？

1. **#include <p18cxxx.h>**
2. **#include "TLS2118_LIB/P18F_LCD.h"**
3. **#include "P18F_LCD.H"**

- 第一個使用 **< xxxx.h>** 的符號是表示使用：
..\Microchip\mplabc18\v3.38\h 路徑下的 **Header File**
- 第二個使用 **"xxxxxx/xxxxx.h"** 引號是指目前的 **Project** 目錄，再下一層的指定路徑
- 第三是使用單純的**"xxxx.h"** 引號是指使用目前 **Project** 的目錄

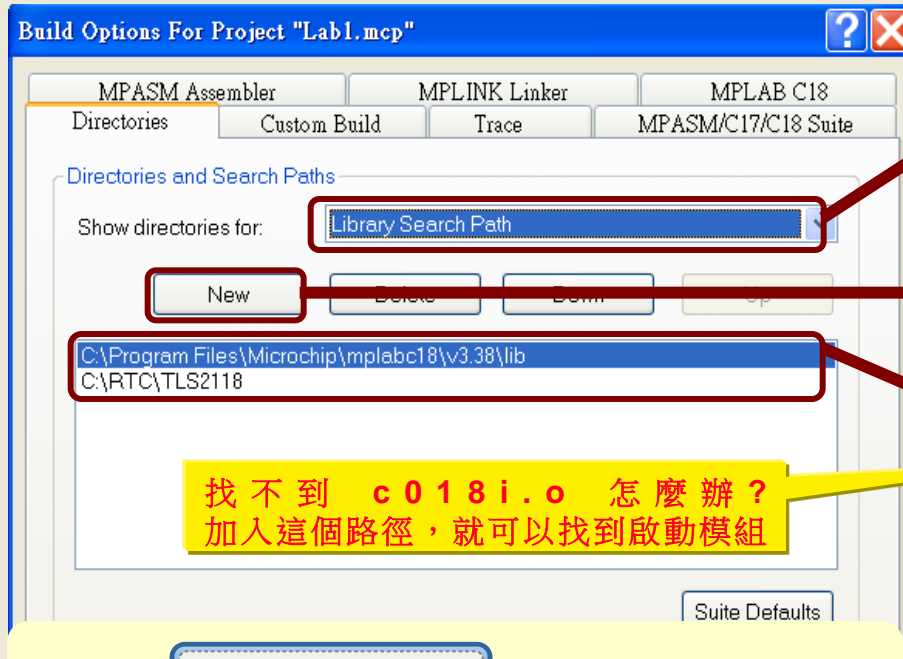
Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

11 設定 Library 搜尋路徑



a Select **Library Search Path** from the combo box

b Click on **New**

c 加入路徑:
..\\Microchip\\mplabc18\\v3.38\\lib
C:\\RTC\\TLS2118\\TLS2118_LIB

找不到 **c018i.o** 怎麼辦?
加入這個路徑, 就可以找到啟動模組

Click **OK** when finished



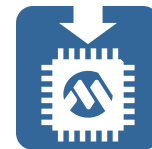
C18 v3.37 以後版本內定安裝路徑不再是 **c:\mcc18** 而是改到 **C:\Program Files** 下了



除了內定的資料庫路徑外, **TLS2118_LIB** 的路徑在本課程裡是需要特別指定的。

Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

12

針對Lab1.c 所加入最基礎所需的部分

在 project 視窗中, Double click **Lab1.c** 可檢視下列的敘述

#include 所要用到的標頭檔是一你的應用程式而定

- 元件定義標頭檔 (做為存取該元件的的定義名稱)
- 標準 C 及 Microchip 資料庫標頭檔 (**if used**)
- 自行定義的函數所要使用的標頭檔 (**if used**)



Lab1.c

```
13  #include <p18fcxxx.h>
14  #include "TLS2118_LIB/P18F_LCD.h"
15
```

所使用的相對路徑的設定
是在 **project options** 對
話視窗下設定

Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

13 針對 Lab1.c 所加入最基礎所需的部分

在程式中已加入對configuration bits 設定的敘述

- 使用 `#pragma config`
- 參考各 help file 中對各 device 選項的列表，選項間可用逗號來區隔
- 為被指定的選項將會使用其預設值



[hlpPIC18ConfigSet.chm](#)



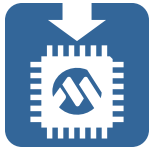
Lab1.c 如果使用 C18 Lite 版本建議將 **extend instruction** 功能關閉

17

```
#pragma config OSC=INTIO7, WDT=OFF, BOREN = ON, BORV = 1, LVP=OFF,  
PBADEN=OFF, XINST = OFF, MCLRE = ON
```

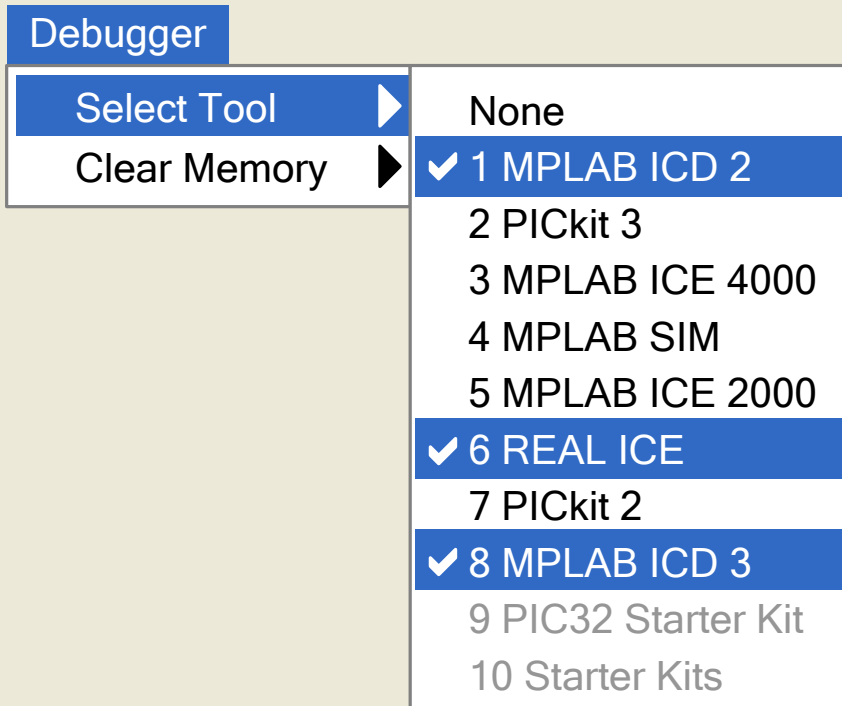
Lab Exercise 1

建立一個 MPLAB® C18 程式專案



步驟

14 開啟你的除錯工具選項： **Debugger** ▶ **Select Tool** ▶



選則除錯工具：

- **MPLAB PICKit 3**
- **MPLAB ICD 3**
- **REAL ICE**
- **MPLAB SIM**

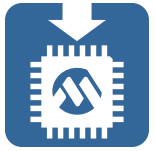
取決於何種工具連接
到 **APP001** 的板子



**DO NOT enable tool from
the Programmer menu.**

Lab Exercise 1

建立一個 MPLAB® C18 程式專案



執行 Debug 的步驟



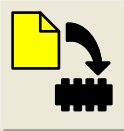
15 Select **Debug** mode.




16 Click **Build All** button.



17 If no errors reported, Click **Program** button.



18 When programming completes, click **Reset** button.



19 Click **Run** button.



20 Click **Halt** button.



Exercise 1 的結果

- 將可以看到 LCD 顯示幕出現
“**Hello, TLS2118!**”
- **D1 LED 在閃爍著!**



建立自己的 LIB 練習

建立一個 LCD 函數庫
(TLS2118.LIB)

Library Exercise



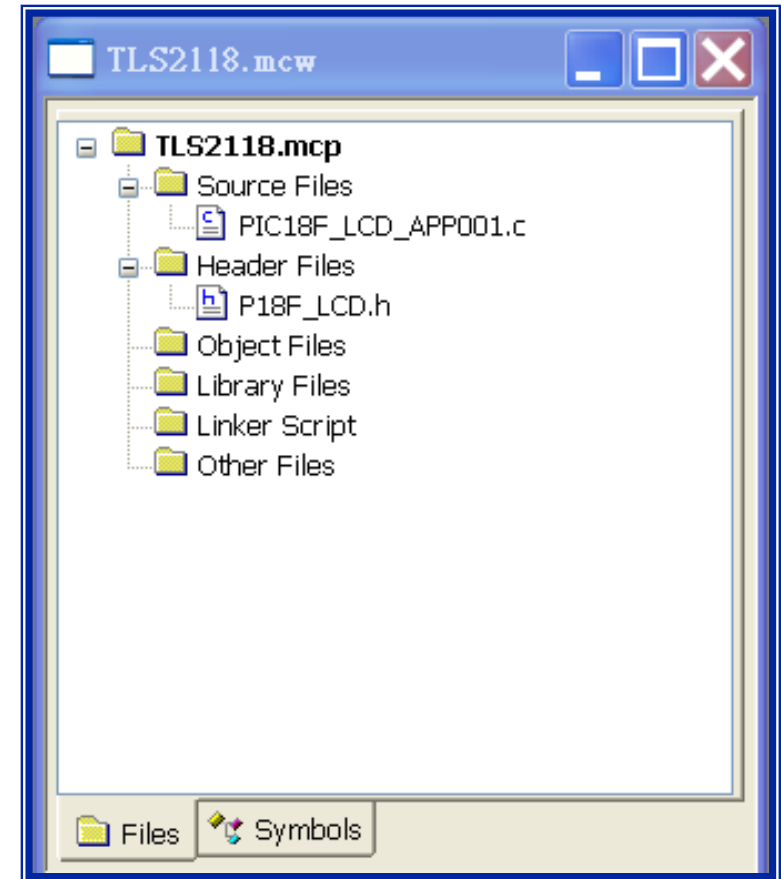
Lab1 功能 Review

使用自己的 **Library** 中的 **functions** 來操作 **LCD**

- 必須將所需要的 Library file 含入 project file 中
- 程式中必須含入欲使用該 Library 必須參考的 header file:
P18F_LCD.h
- Lab1 使用的 function 有 OpenLCD() 、 putsLCD()
- TLS2118.lib 中常用的 function 為
 - void OpenLCD (void) ;
 - void putsLCD(char *) ;
 - void putsLCD(const rom far char *) ;
 - void putcLCD(unsigned char) ;
 - void puthexLCD(unsigned char) ;
 - void LCD_Set_Cursor(unsigned char , unsigned char) ;

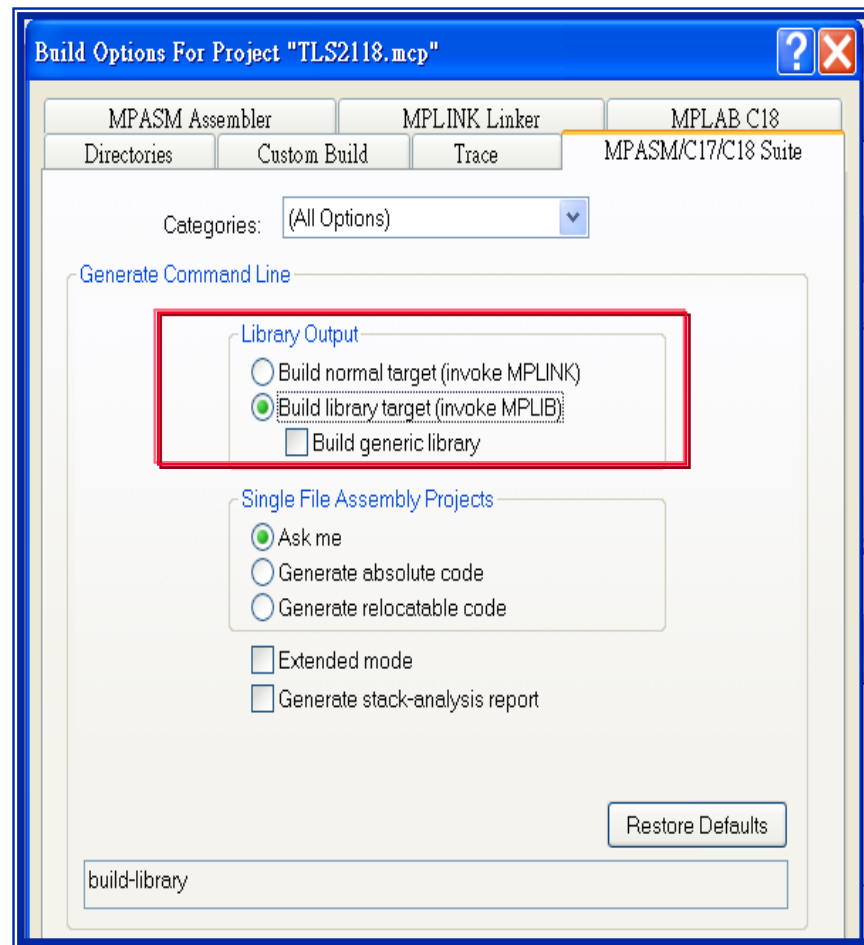
建立 LIB 步驟一

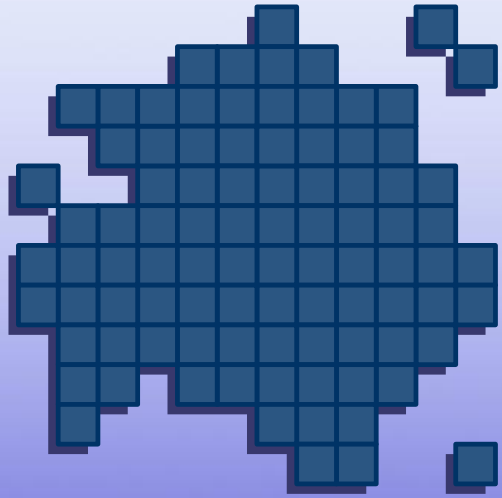
- 如同建立一般的 **Project** 一樣，將 **C** 程式加到 **Project** 裡。
- 每一個 **obj** 都可被 **Linker** 是為一個獨立模組，**Linker** 會將其抽出來使用，而不會將整個 **Library** 都拉進來佔空間。



建立 LIB 步驟二

- 在 **Project → Build Options → Project** 下開啟左側的對話視窗
- 在 **Library Output** 選項下，選擇 “**Build Library Target**”
- 按下套用後離開
- 啟動 **Build All** 開始建立 **TLS2118.LIB**





PIC18 Family 內建的 Special Features

PIC18 Oscillator Options

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

- **Selectable clock options provide greater flexibility for the designer:**
 - LP Oscillator designed to draw least amount of current
 - RC or INTRC provide ultra low cost oscillator solution
 - XT optimized for most commonly used oscillator frequencies
 - HS optimized to drive high frequency crystals or resonators
- **Speed ranges are guidelines only**

POR, OST, PWRT

- **POR: Power On Reset**

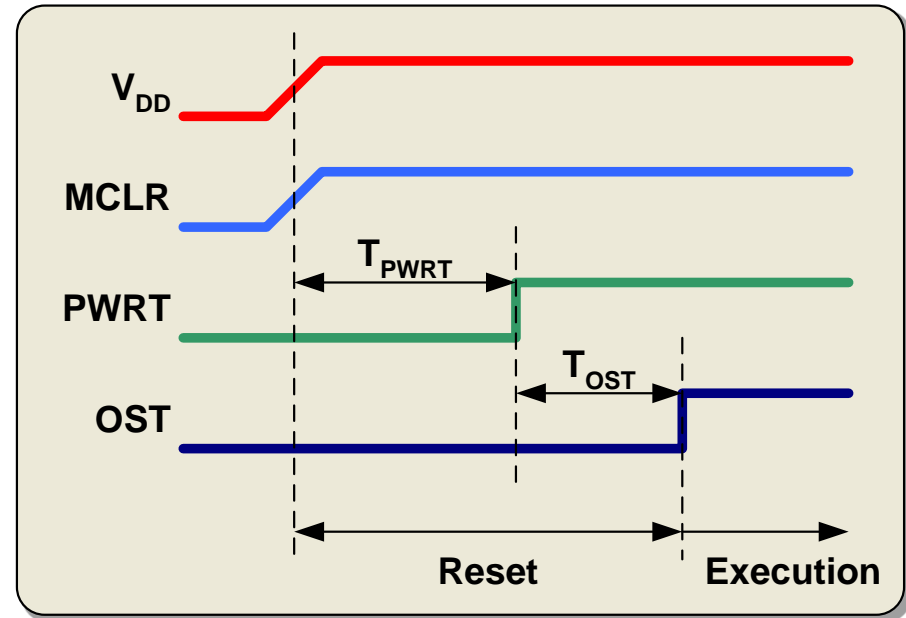
- With MCLR tied to V_{DD} , a reset pulse is generated when V_{DD} rise is detected

- **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

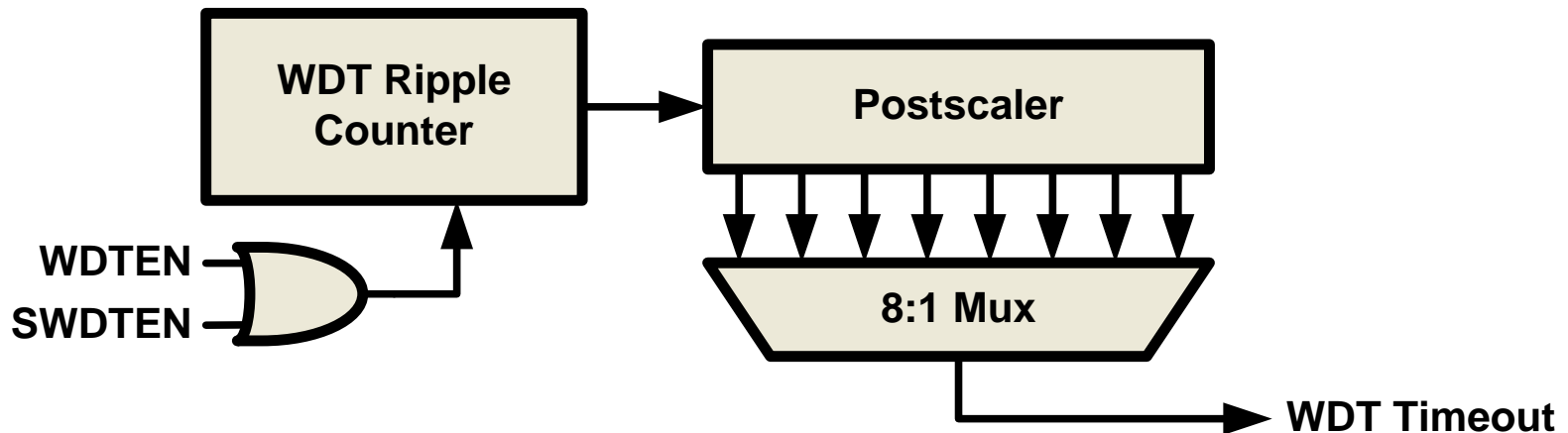
- **The processor can be put into a power-down mode by executing the SLEEP instruction**
 - System oscillator is stopped
 - Processor status is maintained (static design)
 - Watchdog timer continues to run, if enabled
 - Minimal supply current is drawn - mostly due to leakage (0.1 - 2.0 μ A typical) , **XLP device is much lower than std. device**

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 ² C Mode) Start / Stop Bit Detect Interrupt
PSP	Parallel Slave Port Read or Write

Watchdog Timer

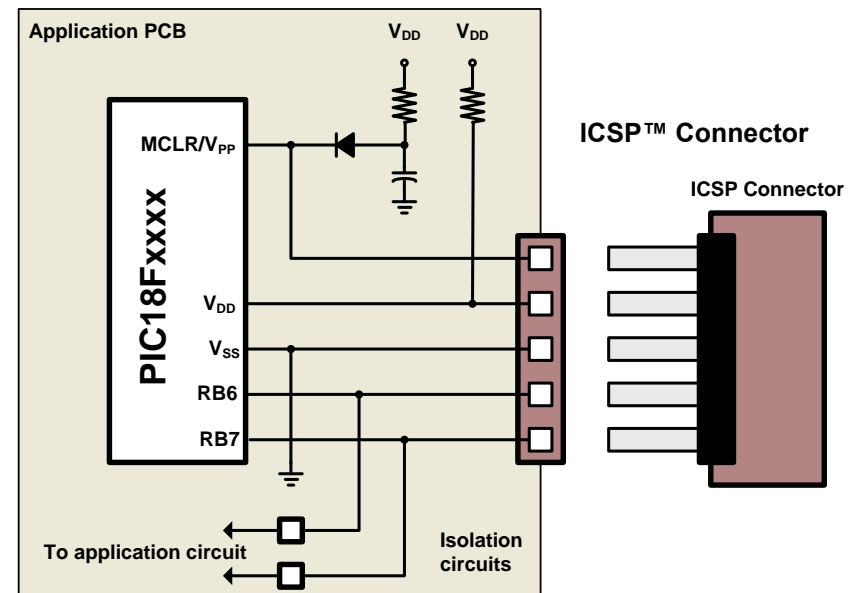
- Helps recover from software malfunction
- Uses its own free-running on-chip RC oscillator
- WDT is cleared by CLRWDT instruction
- Enabled WDT (WDTEN) cannot be disabled by software
- WDT overflow resets the chip
- Programmable timeout period: 18ms to 3.0s typical
- Operates in SLEEP; on time out,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



In-Circuit Serial Programming™

- 除了電源與 V_{pp} (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

- 當工作電壓降低到指定的電壓門檻時，該元件會自動進入重置狀態直到電壓高於電壓門檻
- 防止一些奇怪或不可預期的電壓變化
- 降低成本，避免使用外接的 **BOR** 電路
- 缺點：啟用 **BOR** 時，功耗會加大

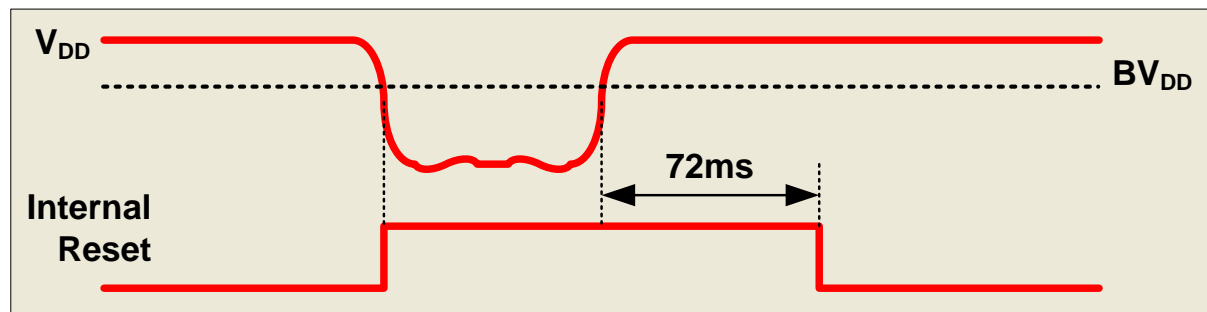
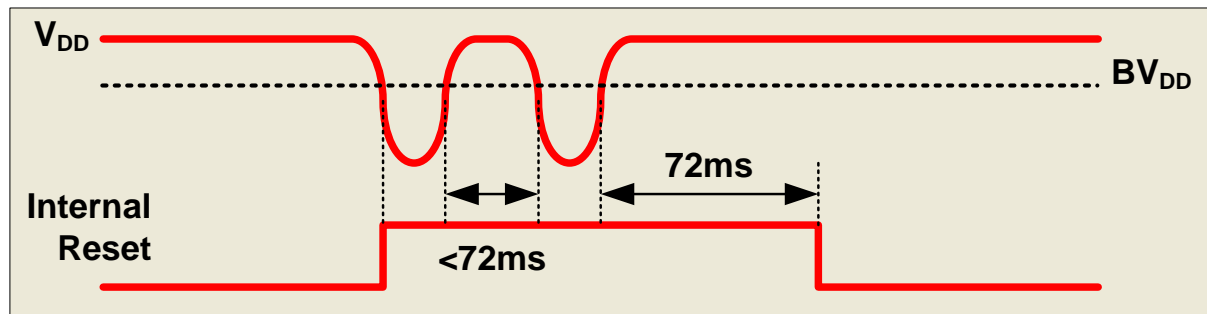
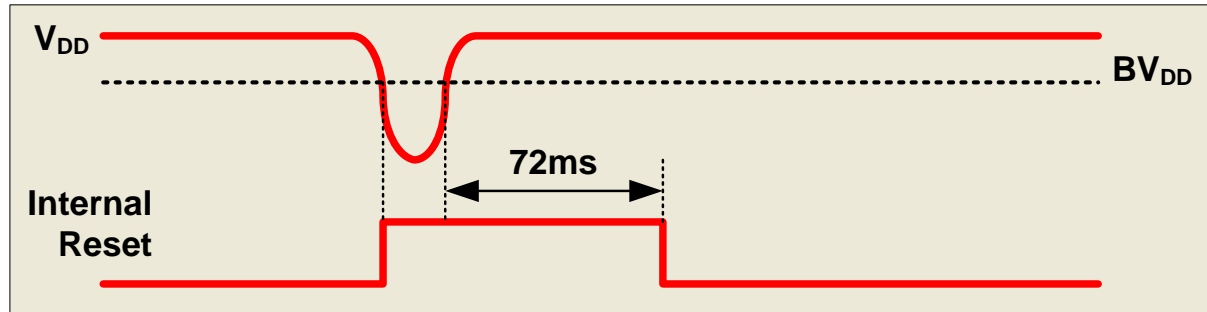
PBOR

Programmable Brown Out Reset

- Configuration 的選項 (在燒錄 IC 時一併被設定)
 - 無法使用軟體來 enabled / disabled (XLP 系列有支援軟體關閉 BOR 功能選項，可在 Sleep Mode 下關閉)
- 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)
 - 2.5V
 - 2.7V
 - 4.2V
 - 4.5V
- 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)

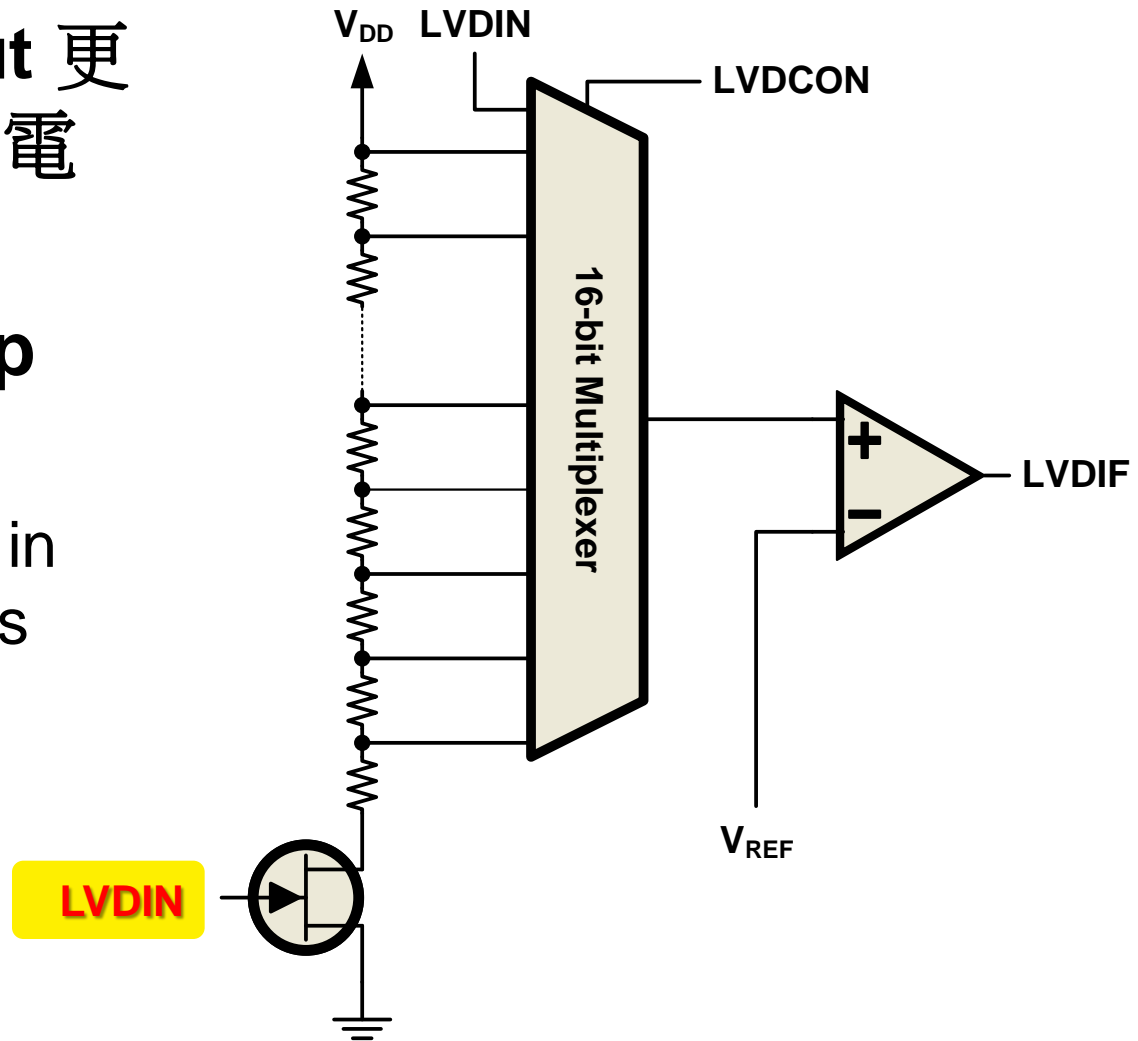
PBOR – Programmable Brown Out Reset

- 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}**



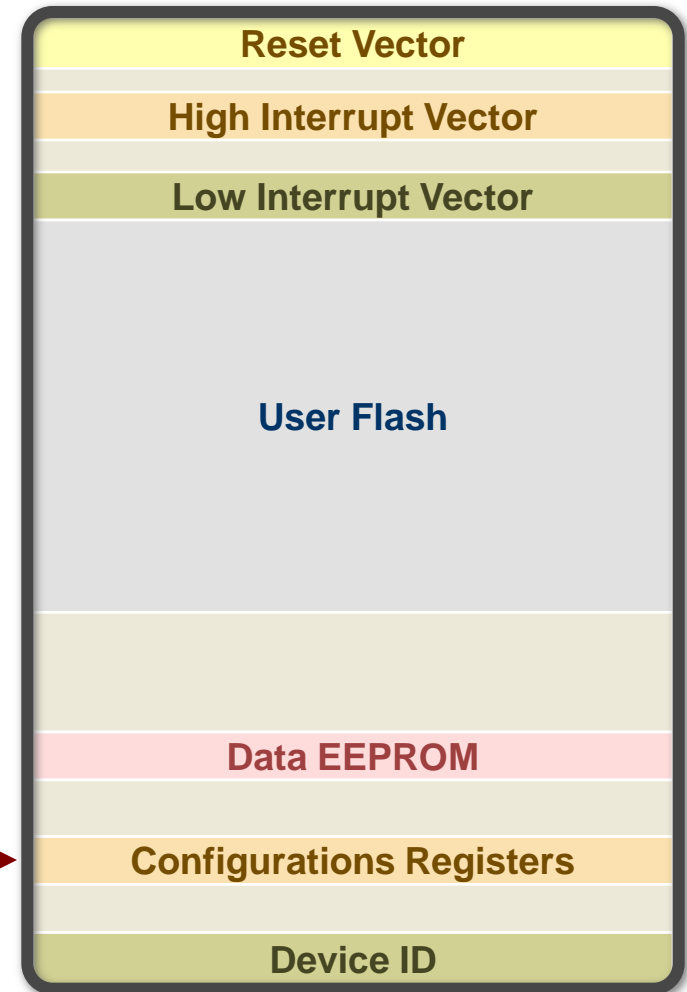


如何設定 PIC18F 的 Configuration Bits

Configuration Bits?

- 用來設定元件的執行功能：
 - Code Protect
 - Watchdog Timer
 - Oscillator Options
 - Debug Options
 - More...

CONFIG 暫存器位於程式記憶空間，
超過程式所能執行的空間範圍
(起始位址 @ **0x300001**)



Configuration Bits

PIC18F4520 Registers (4 of 11)

CONFIG1H Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IESO	FCMEN	—	—	FOSC3	FOSC2	FOSC1	FOSC0
bit 7				bit 0			

CONFIG2L Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	BORV1	BORV0	BOREN1	BOREN0	PWRTEN
bit 7				bit 0			

CONFIG2H Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7				bit 0			

CONFIG3H Register

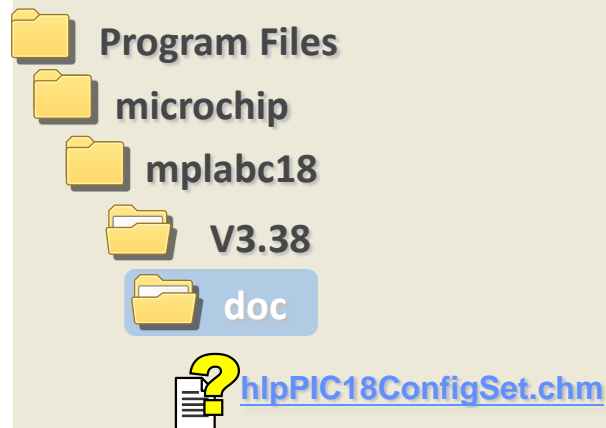
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
MCLRE	—	—	—	—	LPT1OSC	PBADEN	CCP2MX
bit 7				bit 0			

如何在程式中直接設定 Configuration Bits

Syntax

```
#pragma config setting-list
```

- 設定選項裡的各項名稱須用逗號分開，**Config** 的名稱定義在 **hlpPIC18ConfigSet.chm**



Values for the setting list are defined in the PIC18 Config Set help file.

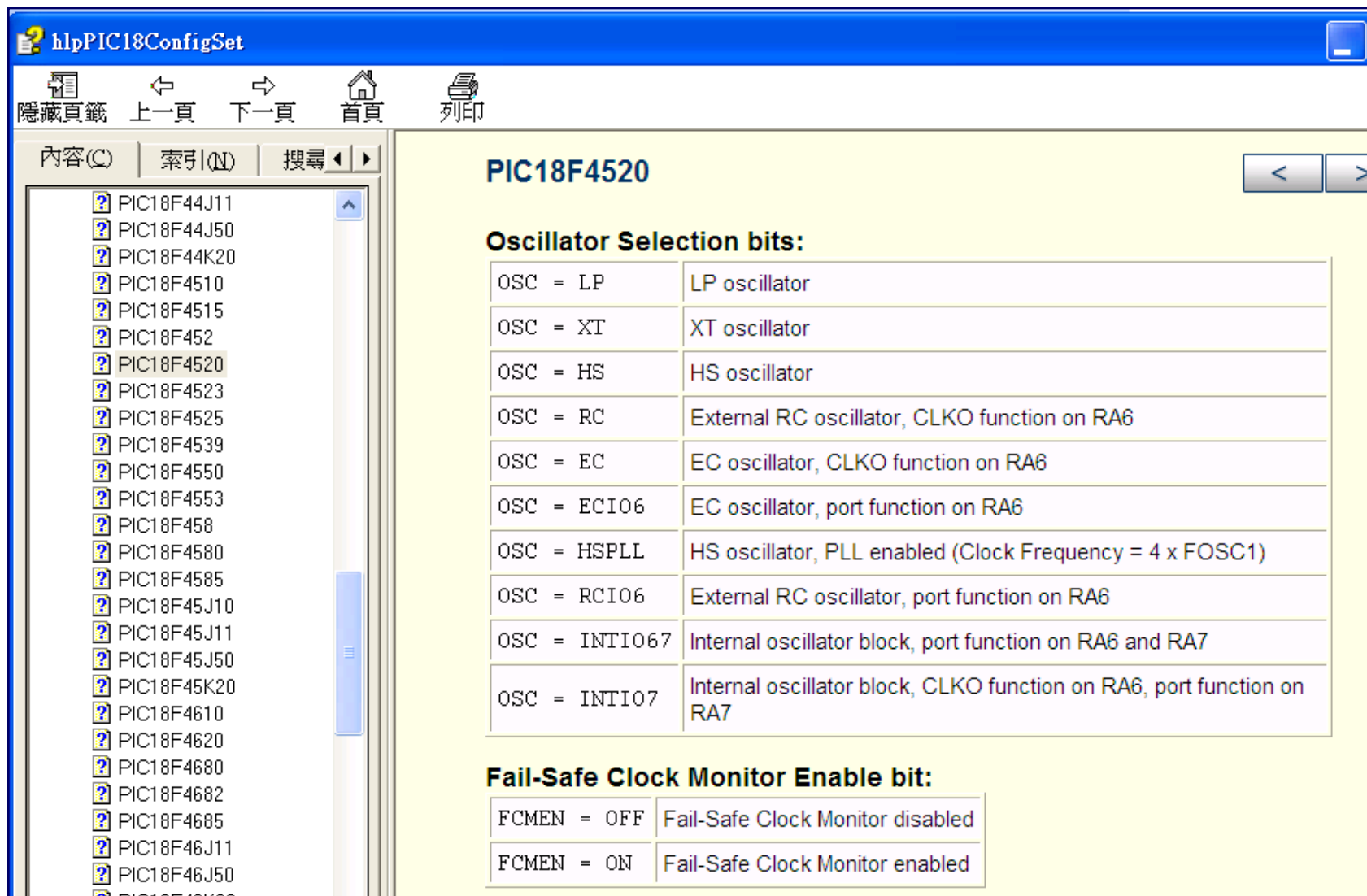


hlpPIC18ConfigSet.chm

使用範例

```
#pragma config OSC = INTIO67, WDT = OFF, BOREN = OFF
```

Configuration Bits 設定項



The screenshot shows the hlpPIC18ConfigSet application window. On the left is a tree view of PIC18 devices, with PIC18F4520 selected. The main area displays the configuration bits for PIC18F4520.

PIC18F4520

Oscillator Selection bits:

OSC = LP	LP oscillator
OSC = XT	XT oscillator
OSC = HS	HS oscillator
OSC = RC	External RC oscillator, CLKO function on RA6
OSC = EC	EC oscillator, CLKO function on RA6
OSC = ECI06	EC oscillator, port function on RA6
OSC = HSPLL	HS oscillator, PLL enabled (Clock Frequency = 4 x FOSC1)
OSC = RCIO6	External RC oscillator, port function on RA6
OSC = INTIO67	Internal oscillator block, port function on RA6 and RA7
OSC = INTIO7	Internal oscillator block, CLKO function on RA6, port function on RA7

Fail-Safe Clock Monitor Enable bit:

FCMEN = OFF	Fail-Safe Clock Monitor disabled
FCMEN = ON	Fail-Safe Clock Monitor enabled

How to Set Configuration Bits

PIC18F4520 擁有的選項 (Part 1)

Feature Symbol	Description	Setting Options
OSC	Oscillator Selection	LP, XT, HS, RC, EC, ECIO6, HSPLL, RCIO6, INTIO67, INTIO7
FCMEN	Fail-Safe Clock Monitor Enable	ON, OFF
IESO	Internal/External Oscillator Switchover	ON, OFF
PWRT	Power Up Timer Enable	ON, OFF
BOREN	Brown Out Reset Enable	ON, OFF, NOSLP, SBORDIS
BORV	Brown Out Reset Voltage	0, 1, 2, 3
WDT	Watchdog Timer Enable	ON, OFF
WDTPS	Watchdog Timer Postscale Select	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768
MCLRE	MCLR Pin Enable	ON, OFF
LPT1OSC	Low Power TMR1 Oscillator Enable	ON, OFF
PBADEN	PORTB A/D Input Enable	ON, OFF
CCP2MX	CCP2 Mux	PORTBE, PORTC
STVREN	Stack Overflow/Underflow Reset Enable	ON, OFF
LVP	Single Supply ICSP Enable	ON, OFF
XINST	Extended Instruction Set Enable	ON, OFF
DEBUG	Background Debugger Enable	ON, OFF
CP_n	Code Protection Enable Block <i>n</i> (<i>n</i> = 0-3)	ON, OFF
CPB	Boot Block Code Protection Enable	ON, OFF
WRT_n	Write Protection Enable Block <i>n</i> (<i>n</i> = 0-3)	ON, OFF
WRTB	Boot Block Write Protection Enable	ON, OFF

How to Set Configuration Bits

PIC18F4520 擁有的選項 (Part 2)

Feature Symbol	Description	Setting Options
WRTC	Configuration Register Write Protection	ON , OFF
WRD	Data EEPROM Write Protection	ON , OFF
EBTR_n	Table Read Protection Block <i>n</i> (<i>n</i> = 0-3)	ON , OFF
EBTRB	Boot Block Table Read Protection Enable	ON , OFF

- 各元件的設定選項所不同，使用時要特別注意
- 使用除錯工具時須將 **Watch-Dog Timer** 關閉
- 在編譯時須正確使用 **Debug/Release** 模式，以正確啟動 **Config.** 裡的 “**Debug Bit**” 的設定
- **hlpPIC18ConfigSet.chm** 裡支援所有 **PIC18F** 的元件
 - 如有新元件無法支援，請更新 **MPLAB C18** 的版本
- 請參考各元件的 **Data Sheet** 裡關於 **configuration** 暫存器的各項設定說明

TLS2118 課程中 Configuration Bits 的設定

- 底下的程式使用在 TLS2118 所有的練習裡，
用來設定 **Configuration Bits**

```
#pragma config OSC=INTIO7, WDT=OFF, BOREN = ON,  
BORV = 1, LVP=OFF, PBADEN=OFF, XINST = OFF,  
MCLRE = ON
```

注意:

- 在除錯模式下有些限制: **WDT** 必須是關閉的。 **LVP** 也是要關閉。 **MCLRE** 必須設 **ON** 以啟用 **MCLR Pin** 的外部 **Reset** 功能。
- 還有如果 **C18** 是使用 **Lite** 版本， **Extended Instruction Sets** 功能也要關閉。



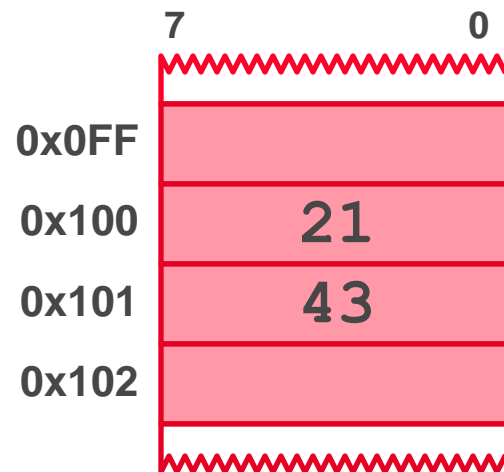
變數與暫存器的存取

MPLAB[®] C for PIC18

資料的擺放原則

- 對超過 **Byte** 以上的資料擺放是以 “**little endian**” 的格式：
 - LSB 擺放在較低的位址
 - MSB 擺放在較高的位址

0x4321 的資料 @ 0x100 (RAM)



MPLAB[®] C for PIC18

資料型態

Type	Bits	Min	Max
char, signed char	8	-128	127
unsigned char	8	0	255
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	-8,388,608	8,388,607
unsigned short long	24	0	16,777,215
long, signed long	32	-2 ³¹	2 ³¹ - 1
unsigned long	32	0	2 ³² - 1

MPLAB[®] C for PIC18

浮點數的格式

- MPLAB C18 採用 IEEE 754 的格式**

32位單精度

單精度二進制小數，使用32個位元存儲。

1	8	23 位長
S	Exp	Fraction
31	30至23 偏正值（實際的 指數大小+127）	22至0 位編號（從右邊開始為0）

MPLAB® C for PIC18

浮點數的資料型態

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
float	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{128} * (2 - 2^{-15}) \approx 6.80564693e + 38$
double	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{128} * (2 - 2^{-15}) \approx 6.80564693e + 38$

- double 和 float 的資料型態是一樣的
- C18不支援 64-bit 的 double 資料型別
- C30 支援 64-bit 的 double

MPLAB[®] C for PIC18

指標形態

- 指標可以存取 **data memory** 或使用 **Table Read** 方式抓取 **program memory**
- 指標的範圍取決於 **memory model**
- 指標的範圍可以使用保留字 "**near**" 或 "**far**" 來宣告

	程式記憶區(rom)	資料記憶區(ram)
far	2M bytes 定址模式 (用 24-bit 的指標)	4K bytes 定址模式 (用 16-bit 的指標)
near	<64K bytes 定址模式 (用 16-bit 的指標)	Access RAM 定址模式 (用 8-bit 的指標)

位元組 (byte) 的存取

語法

暫存器名稱

- 使用這類的暫存器名稱通常是用是使用 **char** 資料形態的變數
- 元件標頭檔所使用的暫存器名稱定義與 **Data Sheet** 相同

Example

```
PORTB = 0x31;           // Write 0x31 to PORTB
AtoD_Result = ADRESH;    // Read A/D Result (High)
TXREG = 'a';             // Send 'a' out UART
if (RXREG == 'x') { ... } // If received char is 'x'
while (RXREG) { ... }    // While char is not '\0'
```

暫存器變數宣告

以 PIC18F4520 為例

PORTA Variable Declaration from p18f4520.h Header File

```
extern volatile near unsigned char PORTA;
```

- **extern**: **PORTA** 的位址定義是在 **p18f4520.o**
- **volatile**: 這變數會不須經程式執行來變動，他本身會自己改變 (像：硬體週邊或中斷函數所用的變數)
- **near**: 變數宣告在 **Access RAM** 這樣就無須做 **Bank** 的切換動作

安排變數的位置 (RAM Location)

- 在 **Access Bank** 內的變數執行速度較快
- 利用前置處理指令 “**#pragma {udata/idata}**”來指定變數在**RAM**的特定位置

```
#pragma udata [data-qualifier] [section-name [=address]]  
#pragma idata [data-qualifier] [section-name [=address]]
```
- 利用前置處理指令 “**#pragma udata/idata**”來結束指定節區位置的作業

```
#pragma udata/idata
```

#pragma udata

- **#pragma udata [data-qualifier] [section-name [= addr]]**
 - **#pragma udata [xxx]** 會將以下所宣告的變數作特定的位置安排，直到遇到指令 “#pragma udata” 才進行常態安排
 - **udata** : 設定無初始值的變數(idata有初始值的變數)
 - Option* – **[data-qualifier]** : 變數要擺在那一區域
 - “access” ==> 安排在 ACCESS Bank 的 RAM (0x00-0x7F)中
 - “空白” ==> 安排在非 ACCESS Bank (GPR)
 - Option* – **[section-name [= addr]]** : 變數放置的位址
 - 指定section-name: 與 Linker的連結描述檔內的section-name相同，則會依連結描述檔內的指定進行安排
 - 不指定section-name: 即獨立的名字，並不與 Linker的連結描述檔內的section-name相同，此時變數會放在unprotected區
 - = addr : 可以強制指定節區位址

#pragma udata 宣告（範例）

```
#pragma udata access AccessSection  
near unsigned char Temp_Code[4];  
near unsigned char Rec_Data;  
near unsigned char PWM_Duty;  
near unsigned char On_Flag;
```

宣告以下之變數放在Access Bank
中，由Linker自行安排位址

```
#pragma udata abc=0x100  
unsigned char j;  
unsigned char i;  
unsigned char e;  
unsigned char f;
```

宣告以下之變數放在GPR中
位址為0x100的地方

```
#pragma udata test  
unsigned char EE_Write_Data;  
unsigned char EE_Addr;  
unsigned char Send_UR;  
unsigned char Err;
```

宣告以下之變數放在GPR中，由Linker自行
安排位址，又 section name 有特別指定故會
被安排在 Bank 2 的位址
SECTION NAME=test RAM=gpr2

```
#pragma udata
```

如何存取暫存器的位元變數

語法

REGNAMEbits.BITNAME

- 暫存器位元變數在元件的標頭檔裡定義
 - p18f4520.h
- 將暫存器的標準名稱加上 **'bits'**
- 位元的名稱定義與 **data sheet** 相同

Example

```
LATBbits.LATB5 = 1;           // Set bit 5 of PORTB
Flag = PORTBbits.RB5;         // Read bit 5 of PORTB
while (!ADCONbits.DONE) { ... } // While A/D converting
ADCON0bits.ADON = 1;          // Enable A/D
```

位元變數使用位元結構方式宣告

Example for PIC18F4520



TRISEbits Variable Declaration from p18f4520.h Header File

```
extern volatile near unsigned char TRISE;
```

```
extern volatile near union {
```

```
    struct {
```

```
        unsigned TRISE0:1;
```

```
        unsigned TRISE1:1;
```

```
        unsigned TRISE2:1;
```

```
    };
```

```
    struct {
```

```
        unsigned :4;
```

```
        unsigned PSPMODE:1;
```

```
        unsigned IBOV:1;
```

```
        unsigned OBF:1;
```

```
        unsigned IBF:1;
```

```
    };
```

```
} TRISEbits;
```

← Bit Field Structure Definitions



Primary Bit Names



Secondary Bit Names

Bit Field
Variable
Declaration



Register and Bit Field Variables

Definition in Processor Specific Library

- Both *REGNAME* and *REGNAMEbits* defined in a processor specific library
- Both are allocated at the same address



Excerpt from p18f4520.asm (p18f4520.lib) Library File

```
SFR_UNBANKED0          UDATA_ACS  H'F80'

PORTA
PORTAbits              RES 1        ; 0xF80
PORTB
PORTBbits              RES 1        ; 0xF81
PORTC
PORTCbits              RES 1        ; 0xF82
...
```



練習二

I/O Port 的規劃與運用

MPLAB SIM – 軟體模擬器的應用

LCD 函數庫的應用

C18 標準函式數庫的應用

Lab Exercise 2

I/O 與 Library 函式的活用



練習二目標

專案位置: **C:\RTC\TLS2118\Lab2\Lab2.mcp**

了解如何使用 **Microchip** 所提供的 **Head File (p18f4520.h)** 裡的名稱定義來使用各項暫存器變數。

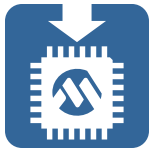
了解位元結構是如何對各周邊位元來定義及其位元變數的名稱及其使用的方法。

練習二動手實驗

- 啟用 MPLAB SIM + Stopwatch 計算出 100mS 的軟體 Delay
 - 設定軟體模擬的頻率為 8MHz
 - Debugger → Setting...
 - 使用斷點設定，觀察 Stopwatch 所執行的時間
- Debug 工具切回 ICD3 後實際編譯過後燒錄程式執行

Lab Exercise 2

I/O 與 Library 函式的活用



Lab Exercise 2 練習

- **學會 I/O 的規劃**
 - 參考 APP001 的電路圖, 將按鍵與 LED 的控制腳位找出來並予以正確規劃 (注意: LCD 與 LED 是共用腳位的設計)
 - 使用 C18 的共用型位元結構, 練習輸入腳位的讀取與輸出腳位的控制
- **練習使用 MPLAB SIM 來觀察 delay() 程式所花費的時間**
- **運用 Library – TLS2118.lib 中提供的 LCD function libraries 來做游標的設定及顯示程式中的變數值.**
- **運用 MPLAB C18 提供的 standard library**
 - 知道 itoa(int value, char *string) 這標準函數的功能嗎?
 - 將程式中的變數 Key_Count 及 Loop_Count 轉換為字串格式後使用 putsLCD() 印出至 LCD



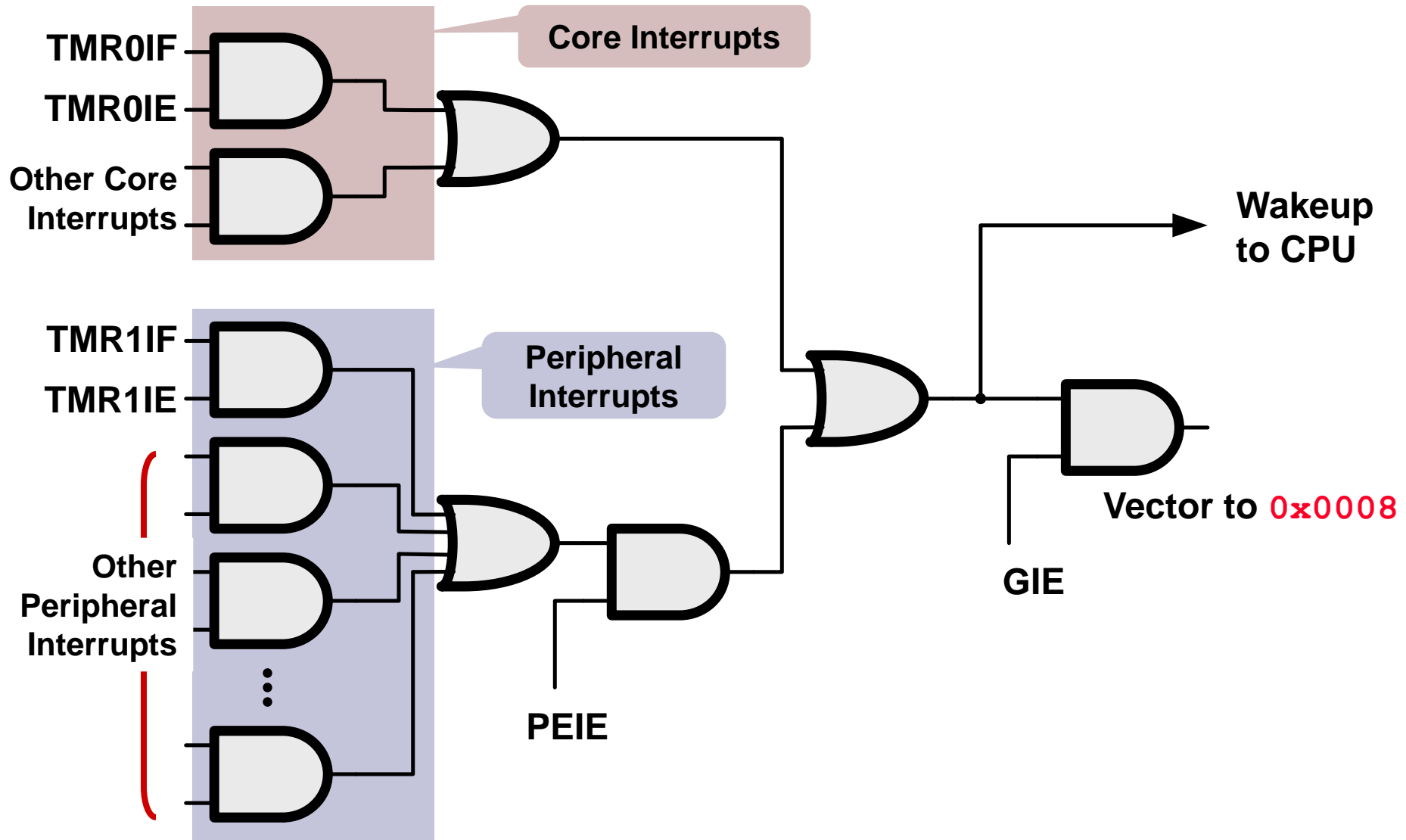
PIC18 Family 內建的 基礎周邊說明 Interrupts

18F4520 中斷處理

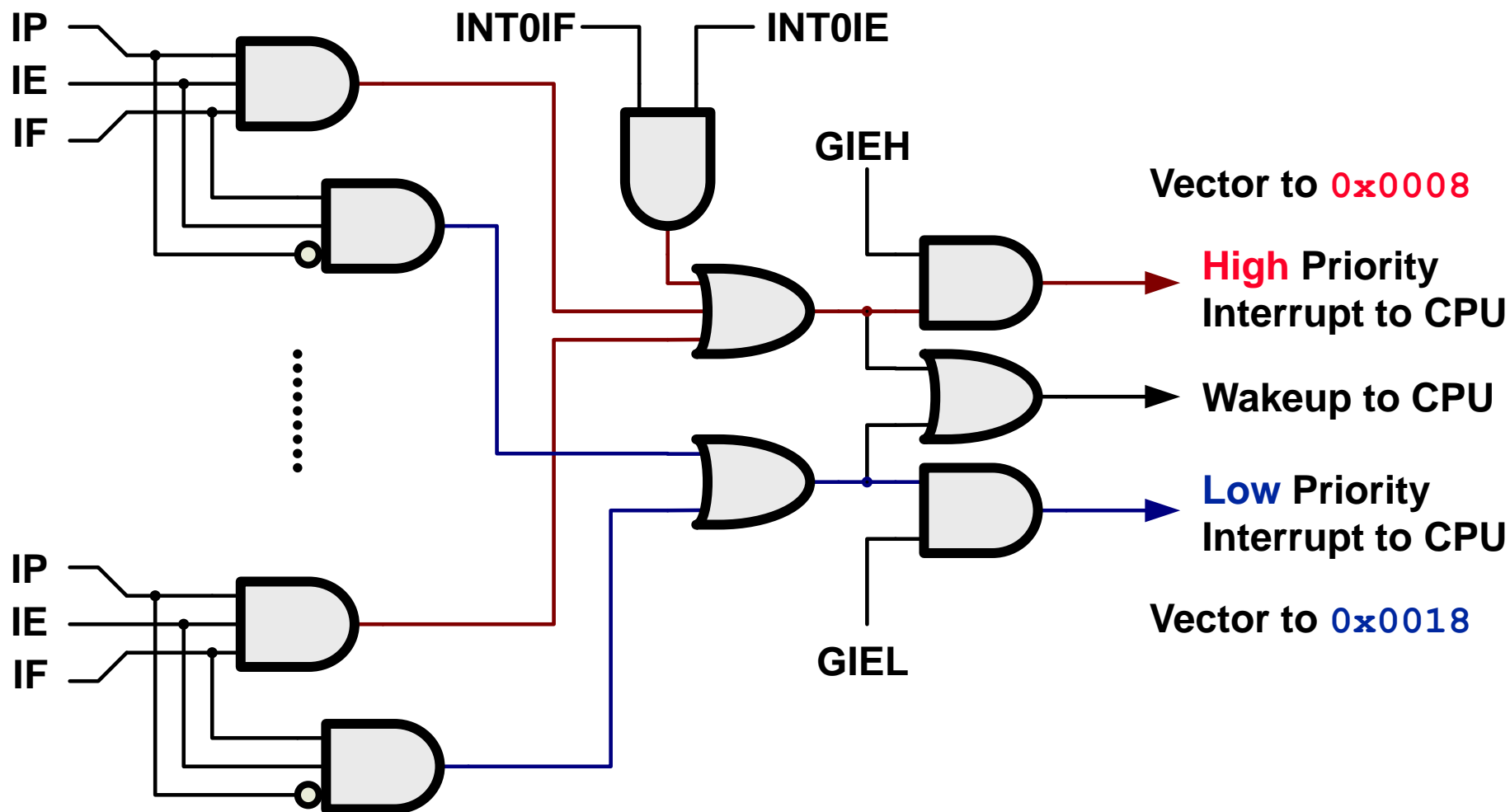
- **18F4520有兩個中斷向量點**
 - 高優先權==>中斷向量位址0x0008
 - 低優先權==>中斷向量位址0x0018
 - 每個中斷源均可選擇其中斷優先權（二選一）
 - 每個中斷源均有獨立的中斷旗標(Flag)
 - 中斷旗標的清除==>自行用軟體清除
 - 每個中斷源均可Enable或Disable
- **當然PIC18系列也可設定與PIC16Fxxx系列的中斷相容（關掉優先權的設定）**

中斷邏輯電路

傳統模式



中斷邏輯電路 優先權模式



Shadow 暫存器

- **18F4520**有“**Shadow Register**”的設計，能提高中斷程式對事件的反應速度
- 高優先權中斷
 - W，BSR，STATUS 的存入/取出使用Shadow Register
 - 程式的返回：retfie FAST
- 低優先權中斷
 - W，BSR，STATUS 的存入/取出則必需透過軟體堆疊
 - 程式的返回：retfie 0
- 以上處理，在 **C18** 編譯器均打點完畢，程式設計者無需傷神考慮
- 結論：**C** 的中斷處理比組合語言更簡單方便

設定高優先權中斷服務程式

- 利用前置處理指令 “`#pragma code`” 將程式區間的起點設定於高優先權中斷向量位址 - `0x0008`，再使用 `goto` 指令將控制權轉移給中斷服務程式
- 利用前置處理指令 “`#pragma interrupt`” 來指定函數為高優先權中斷服務程式 (可在程式任何位址)，處理完畢會自行用 `retfie FAST` 返回

#pragma interrupt func-name save=symbol list

- `func-name` : 高優先權中斷服務程式名稱
- `save= symbol list` : 在中斷服務程式中，須被額外保存的變數 (例: `save= FSR0, PRODL`)

高優先中斷及其 ISR 的設定範例

```
#pragma code hi_vector=0x0008    // 設定中斷進入點
void isr_high_code(void)
{
    _asm
        goto        isr_high
    _endasm
}
#pragma code
/**/*****
/** Function: isr_high(void)
/**      - Received a serial data from RS-232
/**      - Save the received data to Rec_Data
/**/*****
#pragma interrupt isr_high
void isr_high(void)
{
    Rec_Data=ReadUSART();
    PORTD=Rec_Data;
}
#pragma code
```

使用內建組合語言功能，轉移控制權到中斷服務程式(*isr_high*)

中斷服務程式(*isr_high*)

中斷的前置設定

- 由上一頁來看中斷服務的設定是很簡單
- 但別忘了還有一些暫存器須設定後，中斷才會真正的動作(以USART的接收為例)

1. 開啟中斷優先權的設定: `RCONbits.IPEN=1;`
2. 設定USART的接收為高優先權: `IPR1bits.RCIP=1;`
3. enable USART的接收中斷: `PIE1bits.RCIE=1;`
4. 啟動高優先權中斷: `INTCONbits.GIEH=1;`

- 完成中斷的基本設定

設定低優先權中斷服務程式

- 利用前置處理指令 “`#pragma code`” 將程式區間的起點設定於高優先權中斷向量位址 - `0x0018`，再使用 `goto` 指令將控制權轉移給中斷服務程式
- 利用 “`#pragma interruptlow`” 來指定函數為低優先權中斷服務程式，返回方式是 `retfie`

#pragma interruptlow func-name save=symbol list

- `func-name`：低優先權中斷服務程式名稱
- `save= symbol list`：在中斷服務程式中，須被額外保存的變數（例：`save= FSR0, PRODL`）

中斷處理程式 (ISR) 注意事項

- 中斷函數無法傳遞參數
- 中斷所使用到的變數需加入 volatile 的宣告
- 中斷函數儘量不要使用 Local 變數，影響中斷響應時間
- ISR 應越短越好，不要做太多的處理，可以設定 Flag 後交給主程式處理
- 許多運算都會用到 tmpdata 節區，若 ISR 中有複雜的計算 (乘、除、比較的運算)，就必須要 save tmpdata 這個節區，避免與主程式的暫存資料相衝突。
- 新版的 C18，會自動儲存 tmpdata

tmpdata vs. 中斷

- tmpdata 在主程式與中斷函數共享
- 新版的 C18 會自動儲存 tmpdata，可防止中斷程式出錯，但也因要儲存 tmpdata 所以響應速度會變慢
- 如果中斷程式只是很簡單的處理並未使用到 tmpdata 就可以用底下的方式以加快中斷響應速度

```
#pragma interrupt isr nosave=section(".tmpdata")  
void isr (void)  
{  
  
...  
}
```

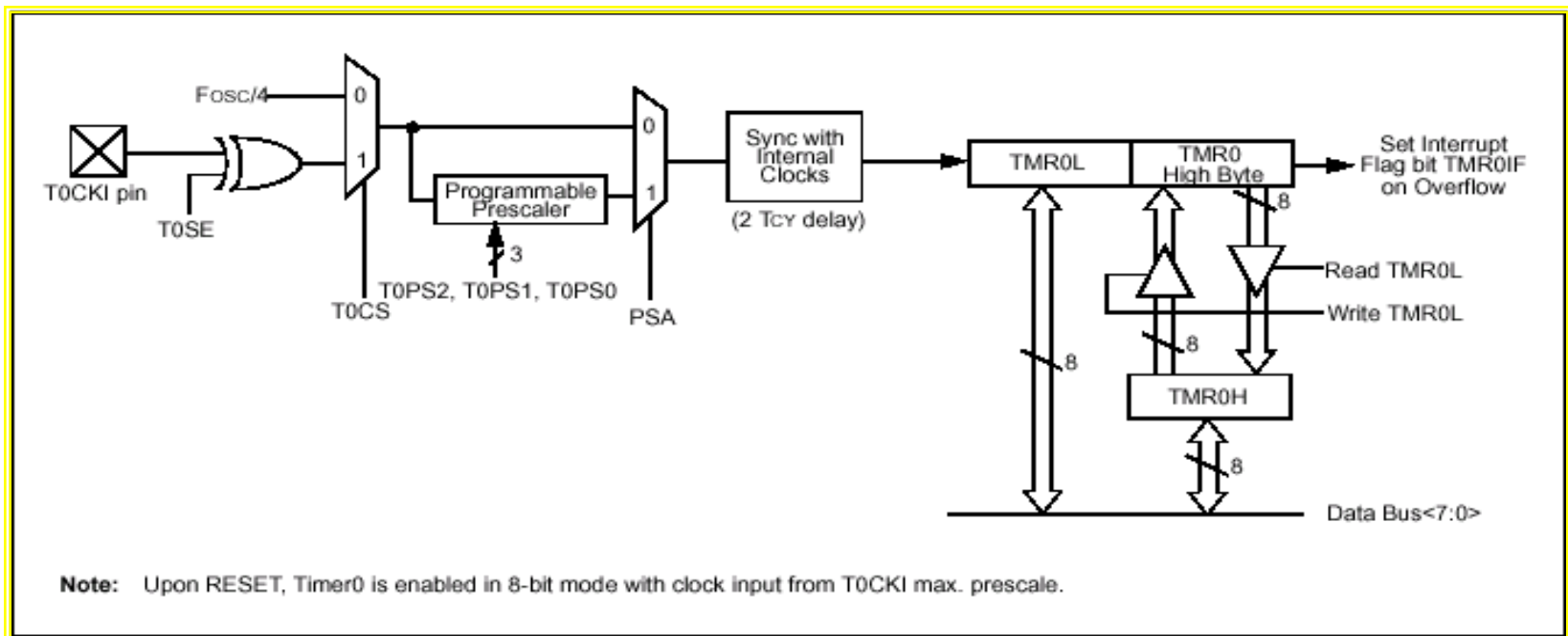


PIC18 Family 內建的 基礎周邊說明 Timers

PIC18 系列的周邊：

Timer0

- Timer0 可設定為 8-Bits 或 16-Bits 模式
- 16-bit mode 時，TMR0H 會在讀寫 TMR0L 時才真正的被讀出或寫入 Timer0
 - 可準確的由 8-bit 的 Data Bus 來讀取 16-bit 的 Timer 值
- 計時器產生溢位時 FFh to 00h (FFFFh to 0000h)，即產生中斷



PIC18 系列的周邊

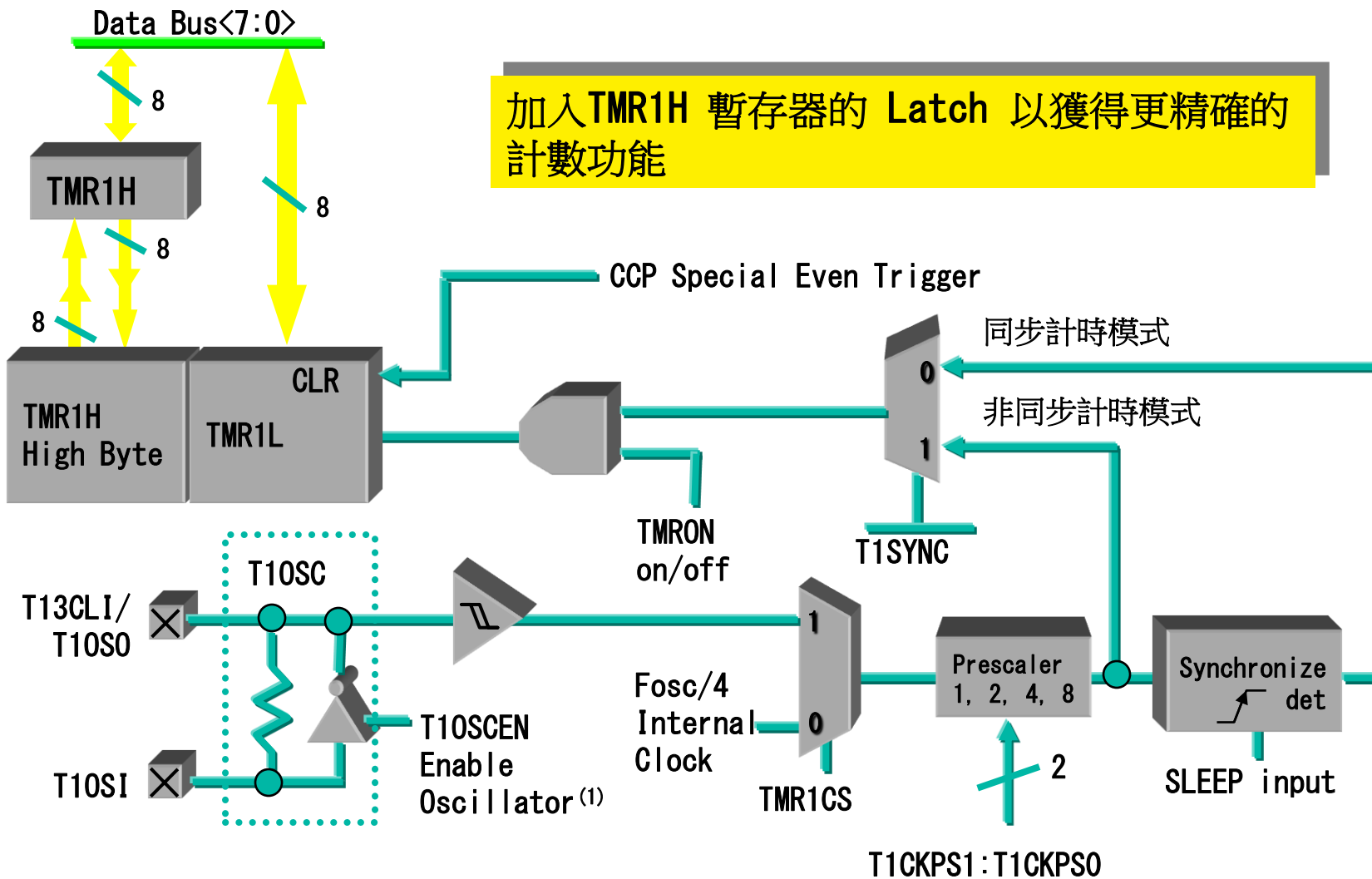
Timer1 and Timer3

- 16-bit 模式的計數器或計時器
- 由兩個可讀/寫的 8-bit 計數器串聯而成
- 預除器有四種選擇： $\div 1$ ， $\div 2$ ， $\div 4$ ，or $\div 8$
- 三種功能：計時器，同步模式計數器，非同步模式計數器（睡眠模式下使用非同步時序喚醒）
- 專用石英振盪電路可作為外部計數時序或第二系統時序（System Clock）選擇
- 當計數器或計時器產生溢位時 FFFFh to 0000h，即產生中斷

PIC18 系列的周邊

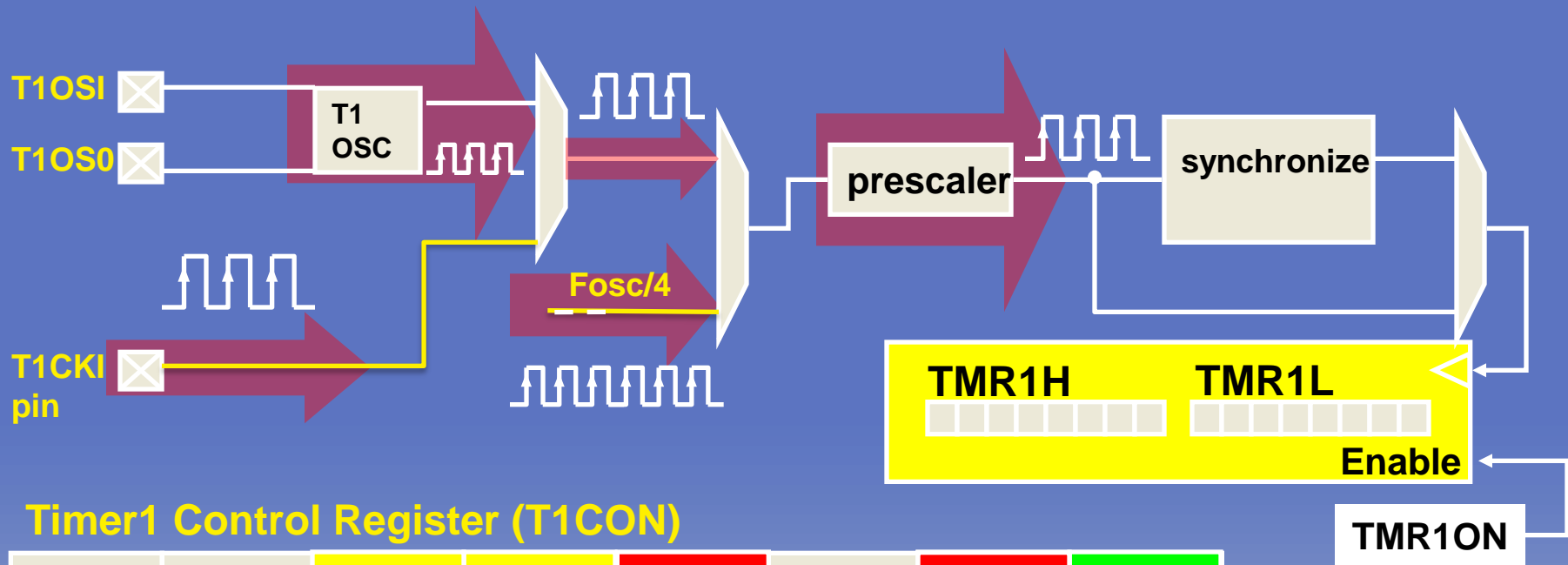
Timer1 and Timer3 (continued)

加入TMR1H 暫存器的 Latch 以獲得更精確的計數功能





Timer1 的運行圖解



Timer1 Control Register (T1CON)



T1CKPS1	T1CKPS0	scale
1	1	1:8
1	0	1:4
0	1	1:2
0	0	1:1

LP Oscillator Enable

1 = T1OSC selected
0 = T1CKI can be used

Clock Source Select

1 = External (T1CKI)
0 = Internal ($F_{osc}/4$)

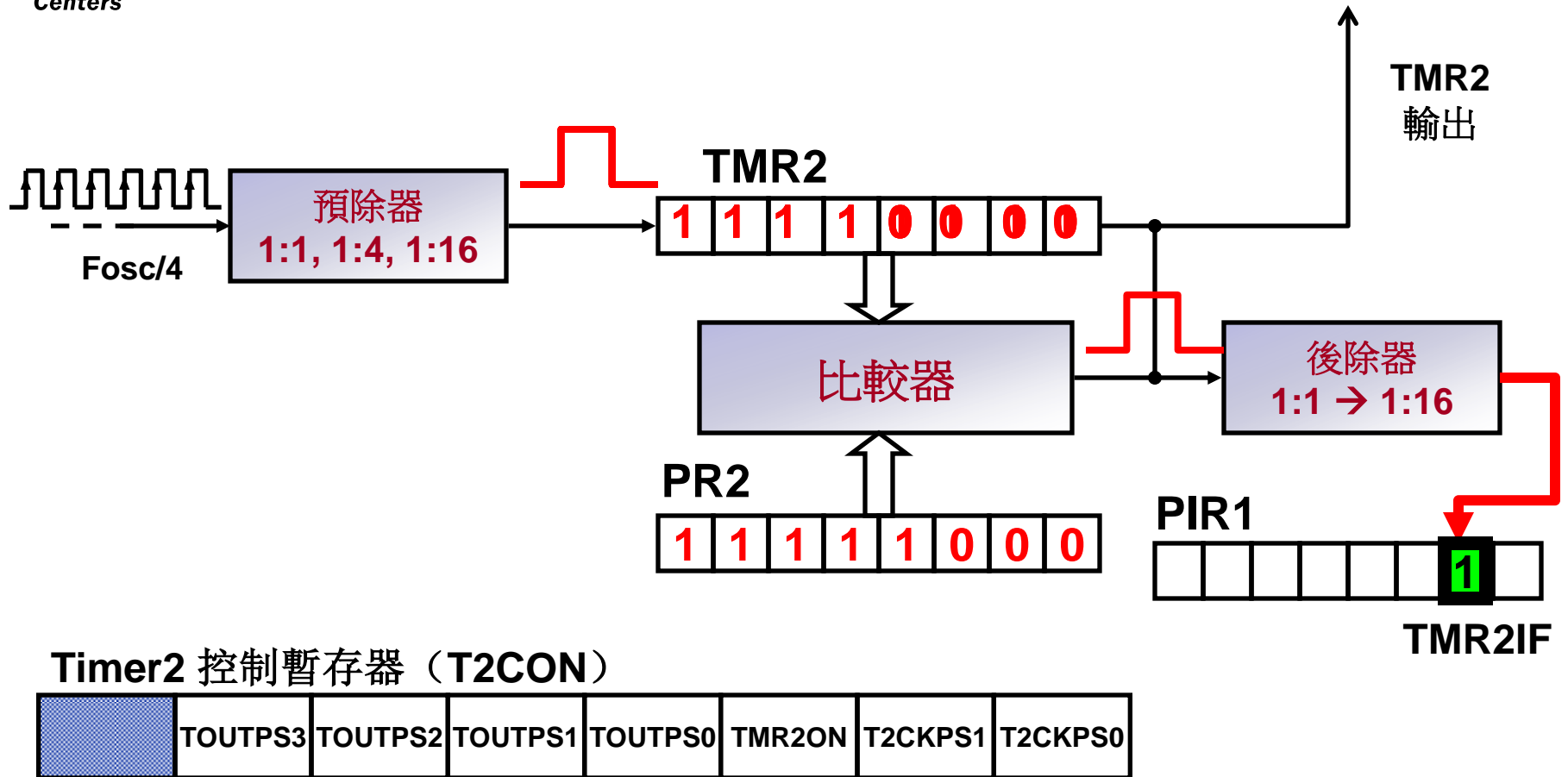
Timer1 On
1 = Enable Timer1

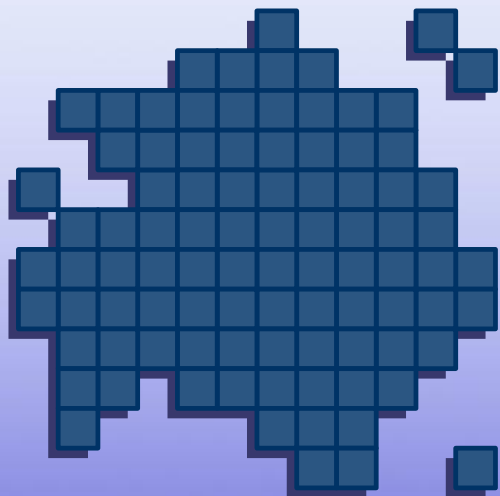
PIC18 系列的周邊

Timer2

- **8-bit** 模式的計時器，有預除器及後除器之功能
- **PWM** 輸出模式下，基本的頻率來源
- **TMR2** 為一可讀、寫具有自動載入功能的計時器
- **TMR2** 會自動加一並與設定的值相比；若相等則送出訊號至後除器或產生中斷，並自動將自己清除為零，重新計時
- 可做為**MSSP (SPI™)** 傳送速率的設定

Timer2 方塊圖





Lab Exercise 3

Timer1 的規劃與應用
中斷服務程式的安排與處理
C18 對於 Timers 函式的應用

Lab Exercise 3

Timer 與 Interrupt 的應用



練習三目標

專案位置: **C:\RTC\TLS2118\Lab3\Lab3.mcp**

了解如何使用 **MPLAB C18** 的中斷函數使用方法及中斷的向量轉移

Timer 結合中斷方式來顯示時間

練習三 動手做實驗

練習 3-1

建立 Timer1 的中斷向量轉移函式，中斷時間設定為 1mS

1. 利用 #pragma code 的方式訂出位址 0x0008 的向量轉移
2. 定義 TIMER1_1ms_VALUE 的 1mS 設定值

練習 3-2

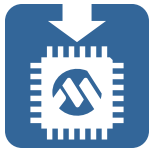
建立 Timer1 的中斷函式，中斷時間設定為 1mS

1. 利用 #pragma interrupt 的方式訂出高優先權中斷函式
2. 重新載入 TIMER1_1ms_VALUE 的 1mS 設定值
3. 相關旗號的清除與設定

重點：Timer1 重新載入的值會受一些背景儲存動作而有所誤差
請利用 MPLAB SIM + Stopwatch 的功能做細部時間的修正

Lab Exercise 3

Timer 與 Interrupt 的應用



Lab Exercise 3 練習

- 學習使用 **C18** 的 **Library** 來規劃 **Timer1**
 - 讓 **Timer1** 能每 1 ms 中斷 CPU 一次
- 學習使用 **MPLAB SIM** 來觀察 **Timer1** 中斷的時間
- 在 **C** 程式中安排高優先權的中斷服務程式
- 使用軟體的技巧來計算時間，得到 **HH:MM:SS** 的資訊
- 使用 **LCD** 將時間顯示

Lab Exercise 3

Timer1 的設定



深入了解 Timer1 的函數庫與時間設定

- 使用 **Timer1** 的函數庫

- 需加入 **Timer** 的定義檔 `#include <timers.h>`

```
WriteTimer1(TIMER1_1ms_VALUE);           // 設定中斷時間
OpenTimer1( TIMER_INT_ON & T1_16BIT_RW & \
T1_SOURCE_INT & T1_PS_1_1 & \
T1_OSC1EN_OFF & T1_SYNC_EXT_OFF );
```

- **1mS** 的中斷時間怎樣計算

一般會這樣計算 `#define TIMER1_1ms_VALUE 65536-(1000*2) // @8MHz`

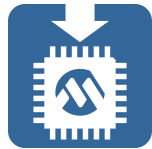
因為 $T_{osc} = 8\text{MHz}$ ， $F_{cy} = 2\text{MHz}$ 。

因預除器設定為 1:1 的除率，所以 1mS 的延遲為 $2\text{MHz} / 1000 \text{ Hz} = 2000$

Timer1 為往上計數方式，1mS 要計數 2000 次，載入 **Timer1** 的值為 $(65536 - 2000)$

Lab Exercise 3

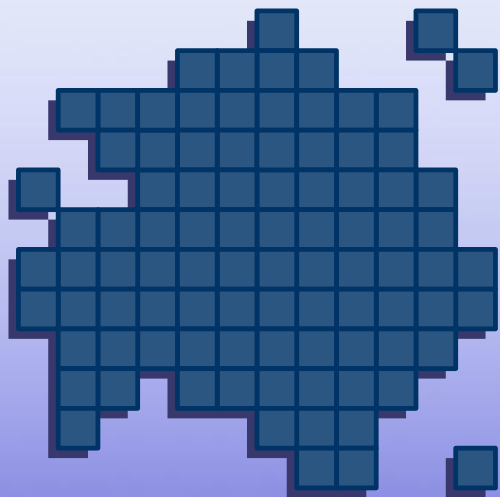
Timer1 的中斷延遲



深入了解 Timer1 的中斷延遲

- 中斷發生時會有固定的反應時間
 - 中斷發生時要儲存一些背景資料，返回時也要取回背景資料
 - 在中斷設定裡，關閉 **Temp Data** 的存取，以加快中斷處理
- #pragma interrupt HighISR nosave=section(".tmpdata")**
- 那 **Timer1** 的時間設定是否還要再調整
 - 計算值 **TIMER1_1ms_VALUE** $65536 - (1000 * 2)$
 - 可以用 **MPLAB SIM** 做此細微的時間調整
 - 實際在程式的設定值為：

TIMER1_1ms_VALUE $65536 - (1000 * 2) + 156$



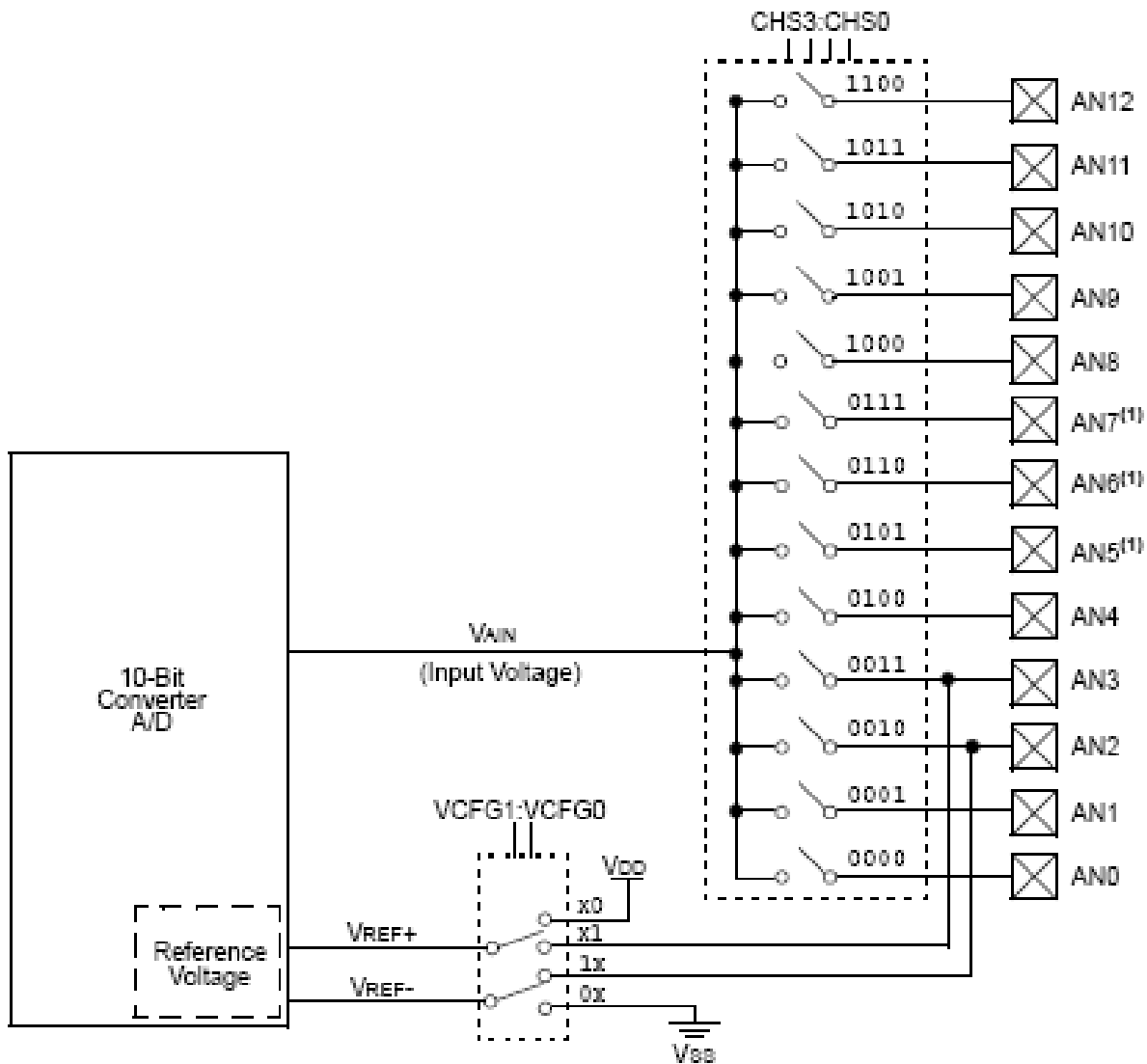
PIC18 Family 內建的 基礎周邊說明

10-Bit Analog-to-Digital Converter

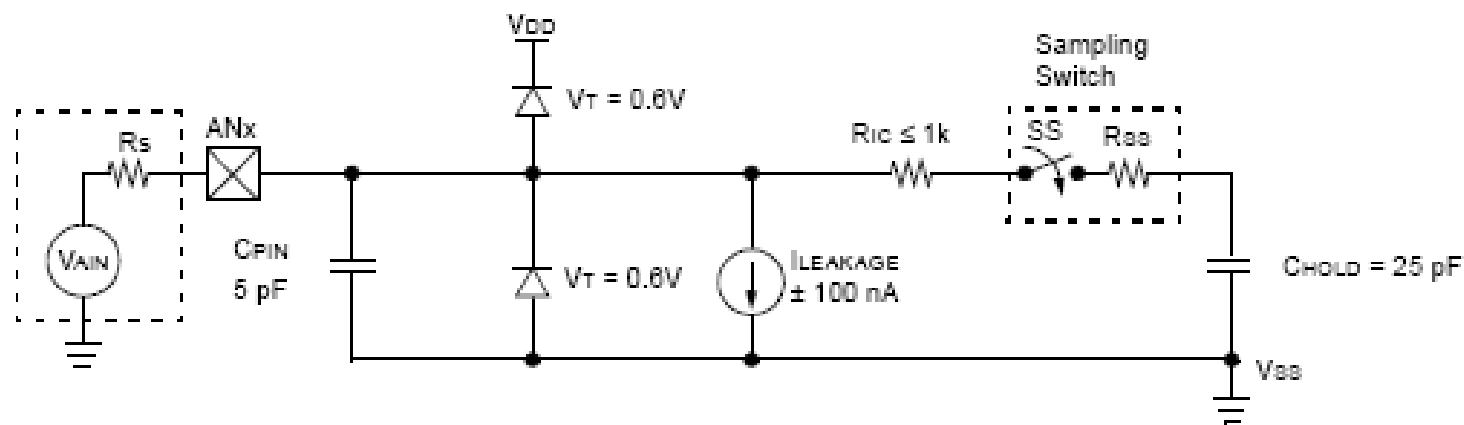
PIC18F4520 中 10-bit A/D 轉換器

- 使用 **SAR** 的轉換方式 (須設定 **TAD**)
- **13** 組類比轉換多工輸入選擇，**10 bits** 解析度
- 類比輸入取樣時間：**2.4 μ S** (信號源阻抗<10K)
- 類比輸入轉換時間：**11 ~ 12 個 T_{AD}**
 - **8.4 μ S (12 T_{AD} , T_{AD} 的最小時間為 0.7uS)**
- **10-bit** 解析度時，只有一位元的誤差
- 允許使用外部參考電壓：**VREF+ & VREF-**
- 轉換的結果允許被設定為向左、向右對齊修正

PIC18F4520 ADC 方塊圖

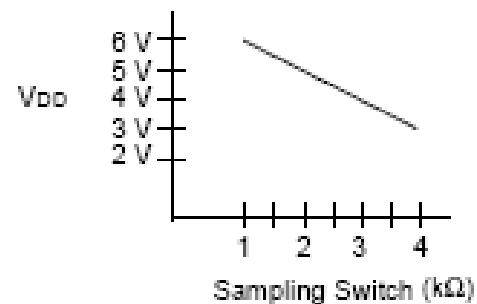


PIC18F4520 ADC 的輸入模型



Legend:

- CPIN = input capacitance
- VT = threshold voltage
- ILEAKAGE = leakage current at the pin due to various junctions
- RIC = interconnect resistance
- SS = sampling switch
- CHOLD = sample/hold capacitance (from DAC)
- RSS = sampling switch resistance



ADC 暫存器

- **ADCON0 控制暫存**

ADCON0



位元	功能
----	----

CHSx	類比通道選擇位元
GO/ $\overline{\text{DONE}}$	1 = A/D 轉換正在進行 0 = A/D 轉換完成
ADON	啟動 ADC 模組

ADC 暫存器

ADCON1

---	---	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
-----	-----	-------	-------	-------	-------	-------	-------

位元	功能
VCFG1	負參考電壓 1 = 來自 Vref- 接腳的外部電壓源， 0 = Vss
VCFG0	正參考電壓 1 = 來自 Vref+ 接腳的外部電壓源， 0 = Vdd
PCFGx	AD 輸入腳位的設定 設定 AN0 ~ AN12 腳位為數位功能或類比輸入

ADC 暫存器

ADCON2

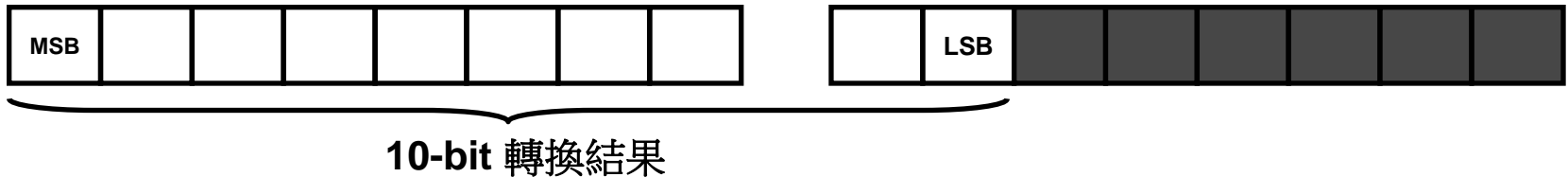
ADFM	---	ACQ2	ACQ1	ACQ0	ADCS2	ADCS1	ADCS0
------	-----	------	------	------	-------	-------	-------

位元	功能
ADFM	AD 轉換結果調整 1 = 向右靠齊調整， 0 = 向左靠齊調整
ACQx	AD 的取樣時間設定 0 T_{AD} 到 20 T_{AD} 的時間選擇

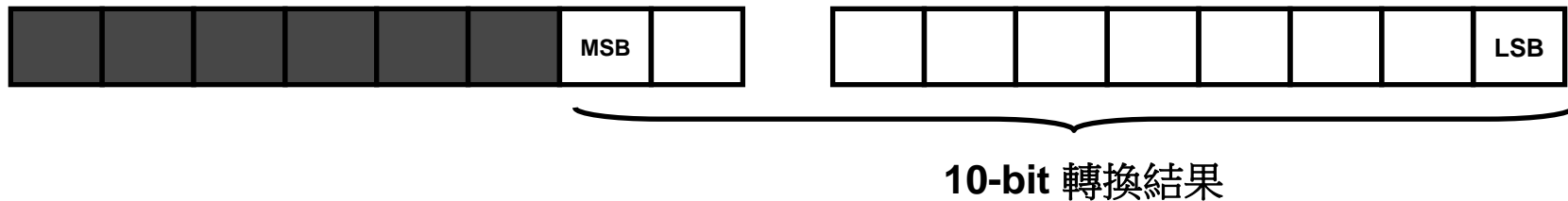
ADFM 位元

- 轉換完成後，**ADC** 轉換結果被放到個結果暫存器 **ADRESH** 和 **ADRESL** 中
- **10-bit ADC** 轉換結果可以向左對齊也可向右對齊

向左對齊



向右對齊



PCFGx 腳位設定

- 與 ADC 共用腳位的操作模式控制

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

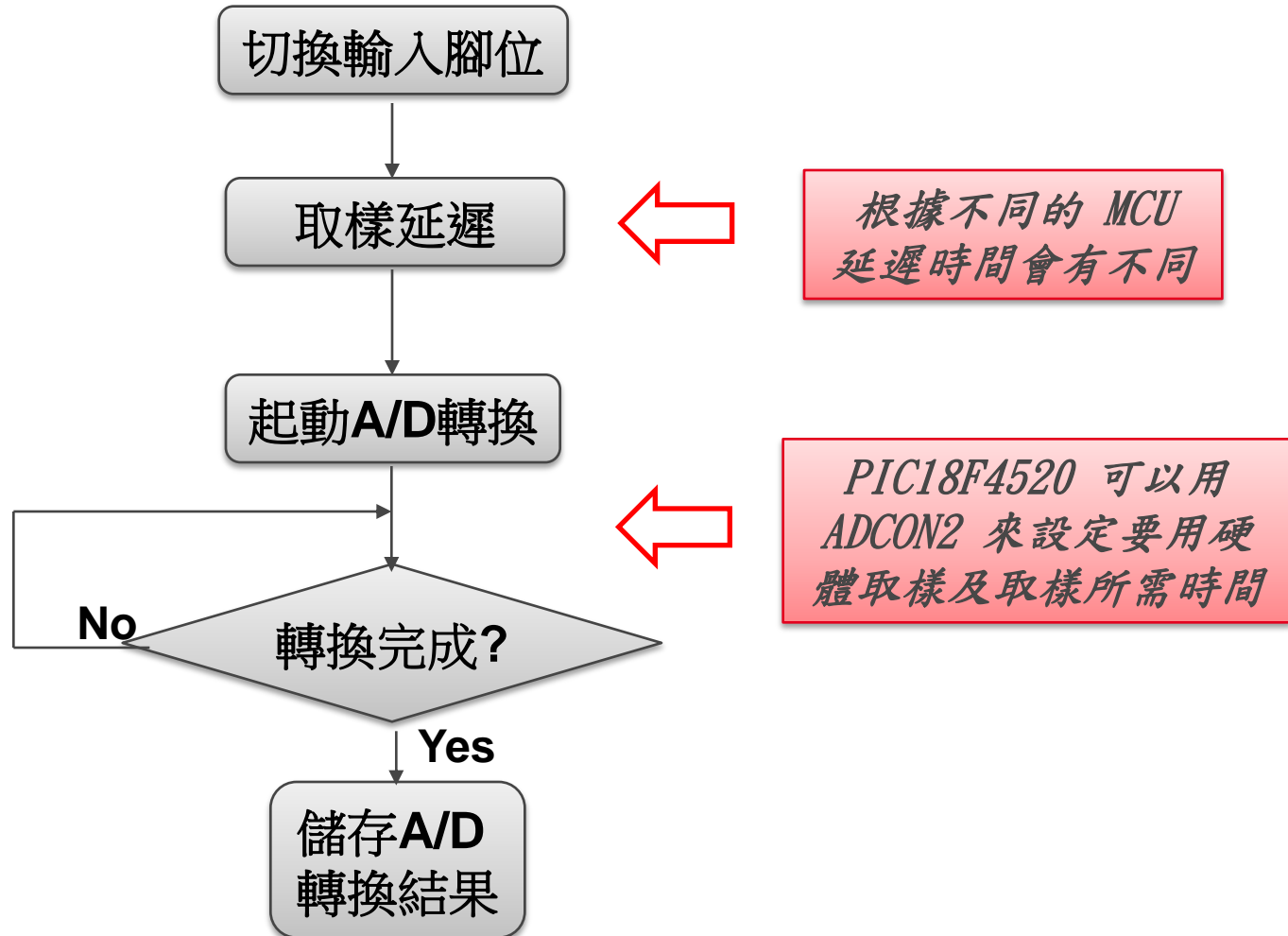
關於 C18 函數庫

- 周邊函數庫的使用參考說明 (**p18f4520.lib**)
 - C:\Program Files\Microchip\mplabc18\v3.38\doc\
PIC18F Peripheral Library Help Document.chm
- 通用函數庫 (**clib.lib**)
 - Software Peripheral Library
 - Software I²C™ Functions, Software UART Functions, ... 等
 - General Software Library
 - 字元轉換，記憶體操作，資料轉換，延遲函數 ... 等
 - Math Libraries
 - 32-bit 浮點數運算，數學運算函數
- 通用函數庫使用說明
 - ..\mplabc18\v3.38\doc\hlpC18lib.chm

ADC 函數庫

- **C18 支援所有的 PIC18F4520 周邊函數**
- **ADC 函數庫有：**
 - (void) OpenADC (unsigned char config,
unsigned char config2,
unsigned char portconfig)
 - (void) SetChanADC (unsigned char channel)
 - (void) SelChanConvADC (unsigned char channel)
 - (void) ConvertADC (void)
 - (void) BusyADC (void)
 - Int ReadADC (void)
 - (void) CloseADC (void)
- 需使用 **#include “adc.h”**

A/D 轉換基本流程





Lab Exercise 4

10-Bit Analog-to-Digital 轉換器的規劃與使用
活用 MPLAB C18 ADC 周邊函式

Lab Exercise 4

10-Bit ADC MPLAB C18 Library 的應用



練習四目標

專案位置: **C:\RTC\TLS2118\Lab4\Lab4.mcp**

了解如何使用 **MPLAB C18** 的周邊函數庫

練習四動手做實驗

- 使用OpenADC() 函式庫來設定 PIC18F4520 的 ADC

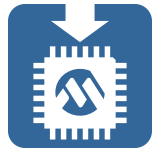
- 參數項請參考 : [PIC18F Peripheral Library Help Document.chm](#)

- 練習四對 ADC 設定知要求項目

1. 設定 T_{ad} 的時間為 $F_{osc}/16$
2. AD 轉換結果向右靠齊調整
3. 設定 AD 取樣時間為 4 個 T_{ad}
4. 選擇 Channel 0 為 AD 輸入
5. 關閉 AD 的中斷
6. 使用 VDD & VSS 當做 ADC 參考電壓源
7. 設定 AN0 為類比輸入功能，其餘腳位為數位功能

Lab Exercise 4

10-Bit ADC MPLAB C18 Library 的應用

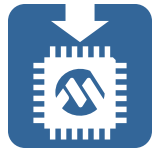


Lab Exercise 4 練習目標

- **學習 ADC module 的規劃 (參考後頁的說明)**
 - 使用 MPLAB C18 提供的 ADC 周邊 Library 來規劃 ADC module
- 使用 練習 3 完成的 **1ms Timer1** 中斷搭配軟體技巧得到一個 **100ms 的 Time Event**
- 當 **100ms 的 Time Event** 發生時，以 **ADC** 對 **APP001** 上的 **VR1** 做電壓的轉換 (參考後頁的說明)
- 當 **ADC** 轉換完成後以 **LCD** 將結果輸出

Lab Exercise 4

10-Bit ADC MPLAB C18 Library 的應用



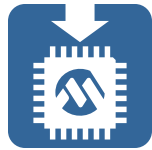
開啟 ADC

```
OpenADC ( ADC_FOSC_16 & ADC_RIGHT_JUST & ADC_4_TAD ,  
          ADC_CH0 & ADC_INT_OFF & ADC_REF_VDD_VSS ,  
          ADC_1ANA);
```

- 逐項說明 ADC 的設定：
 - ADC_FOSC_16 : 設定 T_{ad} 的時間為 $F_{osc}/16$
 - ADC_RIGHT_JUST : AD 轉換結果向右靠齊調整
 - ADC_4_TAD : 設定 AD 取樣時間為 4 個 T_{ad}
 - ADC_CH0 : 選擇 Channel 0 為 AD 輸入
 - ADC_INT_OFF : 關閉 AD 的中斷
 - ADC_REF_VDD_VSS : 使用 VDD & VSS 當做 ADC 參考電壓源
 - ADC_1ANA : 設定 AN0 為類比輸入功能，其餘腳位為數位功能

Lab Exercise 4

10-Bit ADC MPLAB C18 Library 的應用



ADC 的轉換

- 只要底下三行就可完成 **AD** 的轉換動作
SelChanConvADC(ADC_CH0) ;
while(BusyADC()) ;
Update_ADC(ReadADC()) ;
- 逐項說明 **ADC** 的轉換動作：
 - **SelChanConvADC(ADC_CH0)** :選擇 Channel 0 為轉換輸入，取樣延遲後開始做 AD 轉換動作
 - **while(BusyADC())** : 檢查 AD 轉換是否完成
 - **Update_ADC(ReadADC())** : 讀取 AD 轉換的結果到 Update_ADC 的變數裡

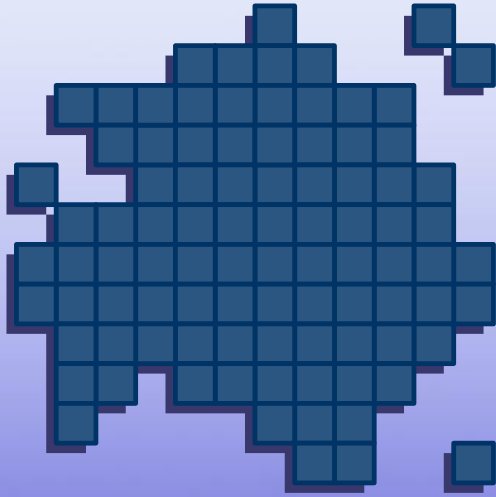


參考資料

MPLAB C18 Runtime environment

Memory Models

How to Override the Default Characteristics of Variables and Functions



C18 Memory Models

Object Allocation Schemes

Memory Models

Overview

Option

Description

Small
-ms

For programs < 64KB

Pointers to Data Space: 16-bits

Pointers to Program Space: 16-bits

For data < 128B 

All data resides in the
Access bank

Large
-ml

For programs > 64KB

Pointers to Data Space: 16-bits

Pointers to Program Space: 24-bits

For data > 128B 

Data may reside in any
bank

- May override on a case by case basis with the **near** and **far** qualifiers

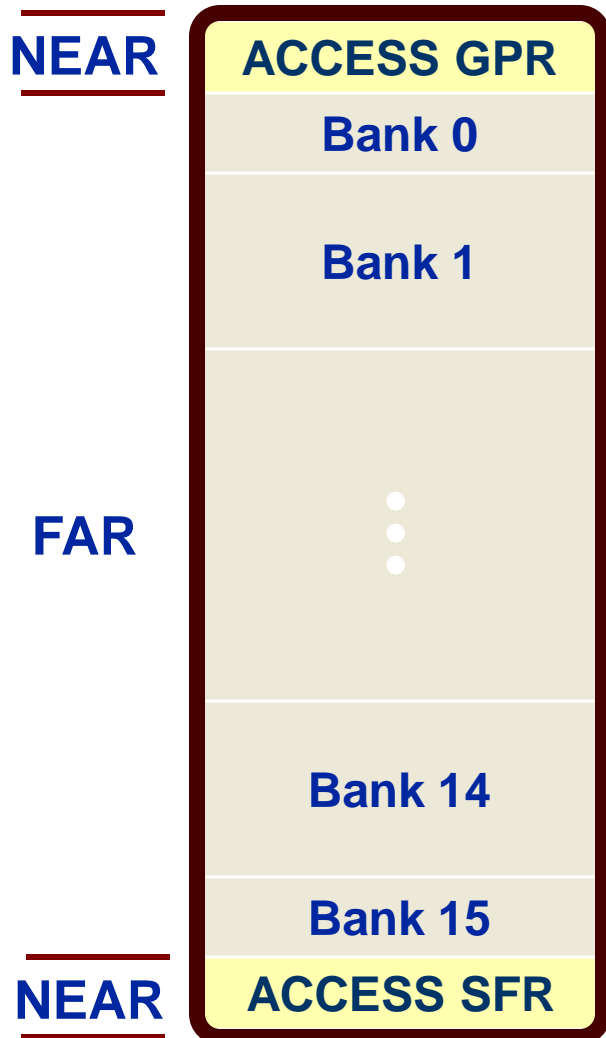


The General Purpose RAM portion of the Access bank is not 128 bytes on all devices. It may be smaller for devices with more than 128 bytes of Special Function Registers (e.g. PIC18F8720: 0xF60-0xFFFF vs PIC18F4520: 0xF80-0xFFFF).

Memory Models

Small Data

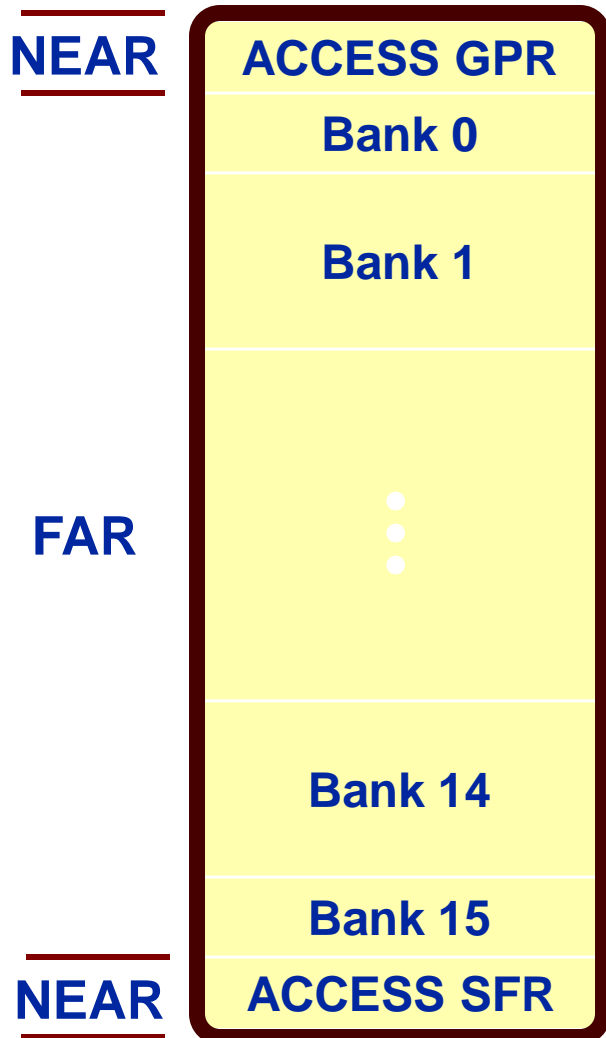
- All data fits in **ACCESS** bank (usually 128B)
- No bank switching required
- Fastest data access
- Smallest code to access data
- Causes warnings when using libraries



Memory Models

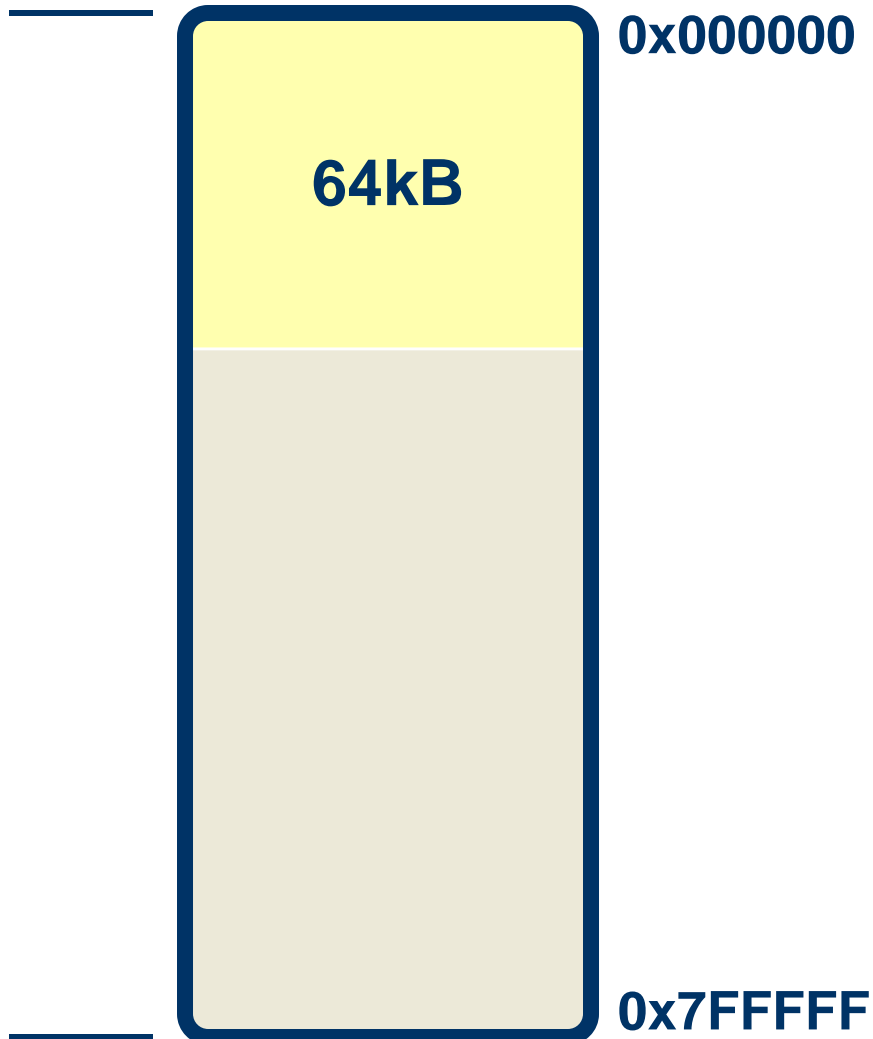
Large Data

- All data *cannot* fit in ACCESS bank
- Data can reside anywhere
- Bank switching required
- Data treated as if it were in **far** space
- Libraries are built using this memory model



Memory Models

Small Code

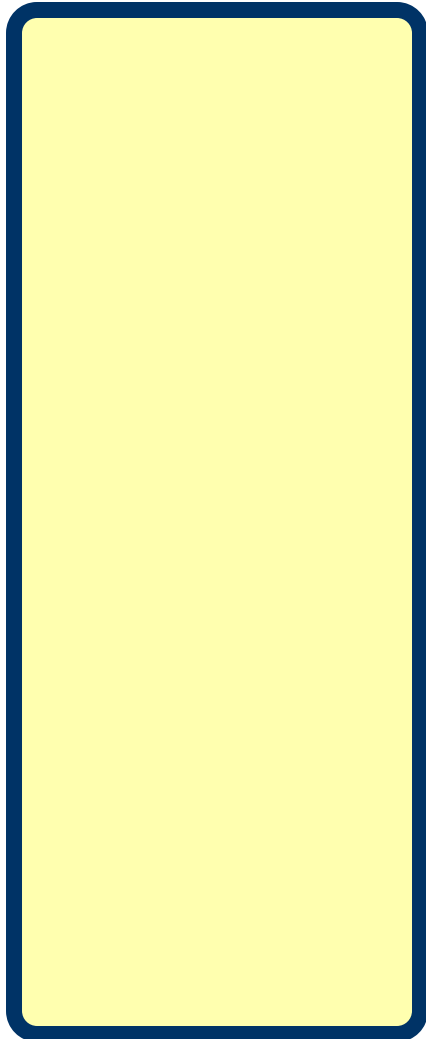


- Default Model
- Program fits within 64kB
- ROM pointers (data and function) are 16-bits (same as RAM)

Memory Models

Large Code

Program Memory (Flash)



0x000000

0x7FFFFFFF

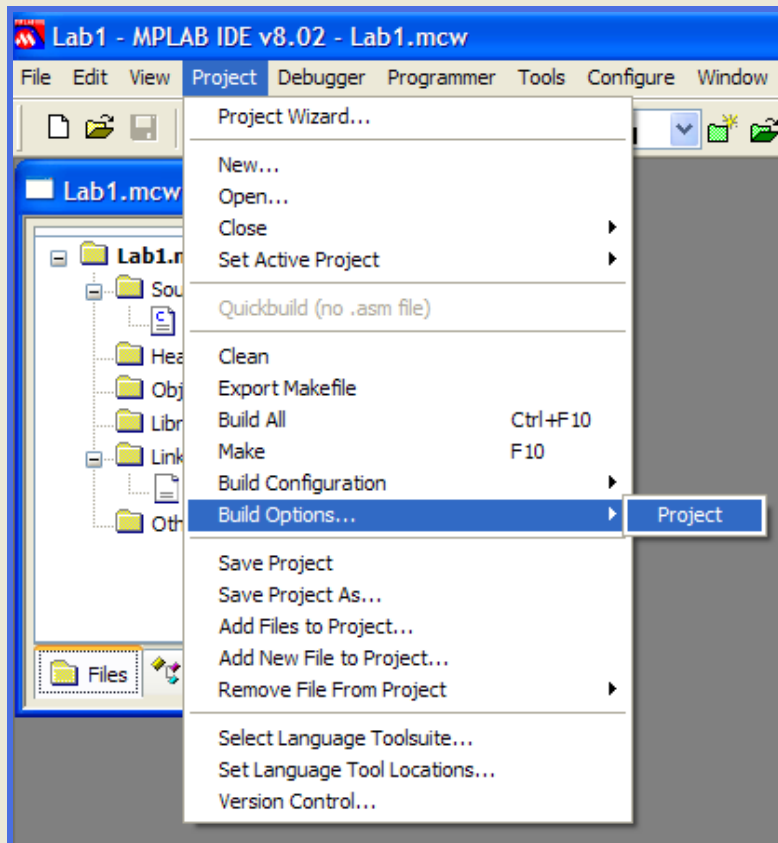
- Program *cannot* fit within 64kB
- ROM pointers (data and function) are 24-bits

Memory Models

How to Select the Memory Model



1 Open Project Build Options



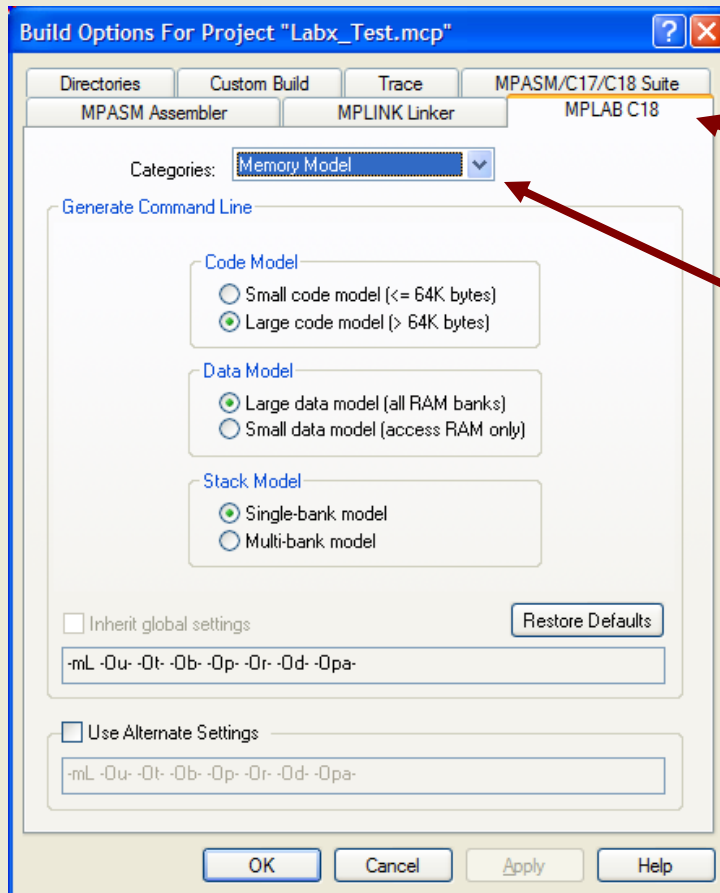
- From the menu bar, select:
Project ► Build Options... ► Project

Memory Models

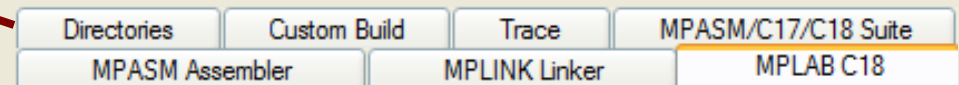
How to Select the Memory Model



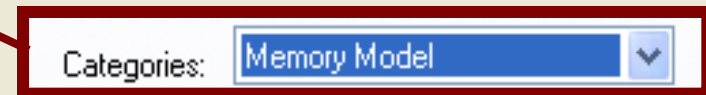
2 Go to Memory Models



► Select the **MPLAB C18** tab



► Select the **Memory Model** category

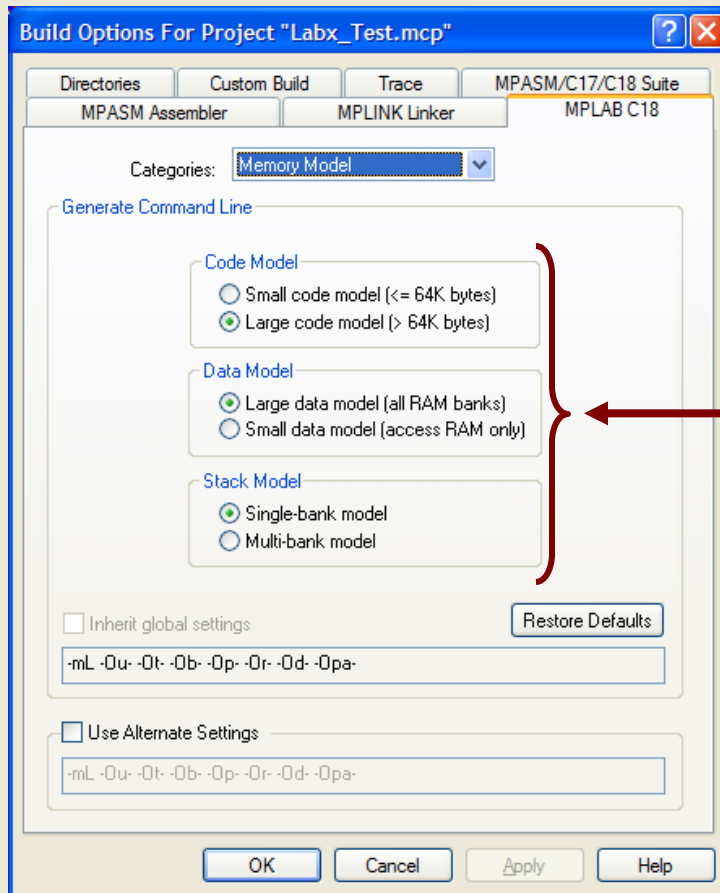


Memory Models

How to Select the Memory Model



3 Select Desired Models



Code Model

- ☐ Small code model (<= 64K bytes)
- ☒ Large code model (> 64K bytes)

Data Model

- ☒ Large data model (all RAM banks)
- ☐ Small data model (access RAM only)

Stack Model

- ☒ Single-bank model
- ☐ Multi-bank model

Memory Models

Tips & Tricks



- **Inappropriate model for your program can cause compile or link errors**
- **As your program grows, you may need to change the memory model**
- **If desired, you have full control over where objects are placed in memory**
 - Use small model, but force some objects into far memory
 - Compile different modules with different models

Memory Models

Tips & Tricks



- **Compiler can often generate more compact code if variables in the Access bank**

Option

Tips for Optimal Memory Use

Small Data

Use if all variables for the application can fit in Access bank

If all data doesn't fit in near space, define some variables with the **far** qualifier so others have space to fit in near data.

Large Data

1. Compile some individual modules using Small Data. Then include their compiled object modules in the Large Data project.

2. Define individual variables with the **near** qualifier



Following the use of a **far** data pointer in a small memory model program, the **TBLPTRU** byte must be cleared by the user. MPLAB C18 does not clear this byte.

Memory Models

Tips & Tricks



- Functions that are near (in first 64kB) may call each other more efficiently

Option

Tips for Optimal Memory Use

Small Code

1. Use if all functions are within first 64kB.
2. Compile some modules using Small Code and include their object files in a Large Code project.
3. If not all functions are in first 64kB, define some functions with the **far** qualifier.

Large Code

1. Define some functions with the **near** qualifier. An error will be generated if the function cannot be reached by one of its callers using the more efficient form of the function call.

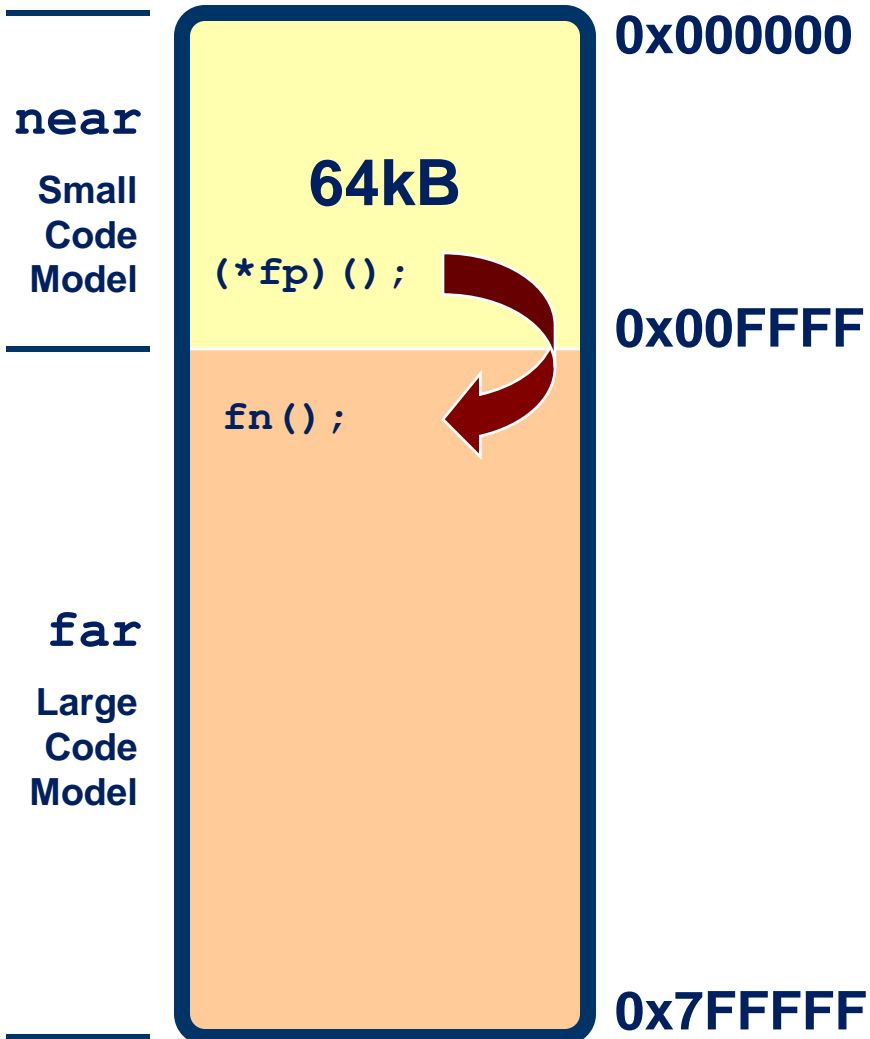


Following the use of a **far** function pointer in a small memory model program, the **PCLATU** byte must be cleared by the user. MPLAB C18 does not clear this byte.

Mixing Memory Models

Small Code application with Large Code libraries

Program Memory (Flash)



- Project built with small code model for efficiency
- Libraries built with large code model for flexibility
- Function pointers may be used to call far code library from small code application

Mixing Memory Models

Small Code application with Large Code libraries



Function in library built with large code model

```
void LCDPutStr(rom char *str) ;
```



Application built with small code model

- 1 Declare a function pointer and initialize it to point to the library function

```
far rom void (*fp) (far rom char *s) = (far rom void *) LCDPutStr;
```

Give function pointer and
pointer to parameter ability to
address beyond 64k

Typecast from pointer to `void` function to
a pointer to `far rom void` function
(Eliminates suspicious pointer conversion
warning)

Mixing Memory Models

Small Code application with Large Code libraries



Function in library built with large code model

```
void LCDPutStr(rom char *str);
```

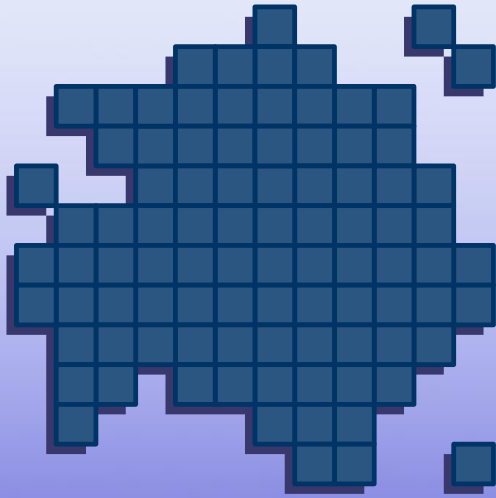


Application built with small code model

2 Call the library function via the pointer

```
fp((far rom char *) "Hello, world!");
```

Typecast from pointer to rom char to a pointer to far rom char
(Eliminates suspicious pointer conversion warning)



How to Override the Default Characteristics of Variables and Functions

Using type qualifiers and `#pragma`

Locating Variables and Code

Section Types

General Syntax

```
#pragma sectiontype sectionname=address
```

- Section types define the overall region of memory where an object is to be located

Section Name	Memory Region	Usage
udata	RAM	Static Uninitialized Data (e.g. <code>char x;</code>)
idata	RAM	Static Initialized Data (e.g. <code>char x = 55;</code>)
udata access	ACCESS RAM	Static Uninitialized Data in Access RAM
idata access	ACCESS RAM	Static Initialized Data in Access RAM
romdata	Program/Flash	Variables and Constants
code	Program/Flash	Executable Code

Locating Variables

How to place a variable in the Access Bank

Syntax

```
#pragma sectiontype access sectionname  
near type identifier;
```

- The **#pragma** tells linker to place the following variable(s) in the access bank
- **near** tells the compiler not to generate bank switching instructions
- Can be accessed more efficiently

Example

```
#pragma udata access mySection  
near int x;  
near char a[10];
```

Locating Variables

Manually Optimizing Data Memory Use



NEAR

ACCESS GPR

Bank 0

Bank 1

⋮

FAR

Bank 14

Bank 15

NEAR

ACCESS SFR

- Use large data model
- Define frequently accessed variables with **near** qualifier

Locating Variables

Variable Allocation Example



near Variables with the Large Data Model

```
#include <p18f4520.h>
```

```
char a[124]; ①
```

```
#pragma idata access myData  
near char x = 0xAA; ②
```

```
int main(void)  
{  
    ...  
}
```

- ① Variables are **far** by default (banked)
 - a[] may be located anywhere
- ② To create **near** variables with the large data model:
 - Declare a new access section and give it some name
 - Define variables with the **near** qualifier

Locating Variables

How to place a variable in far memory

Syntax

```
far type identifier;
```

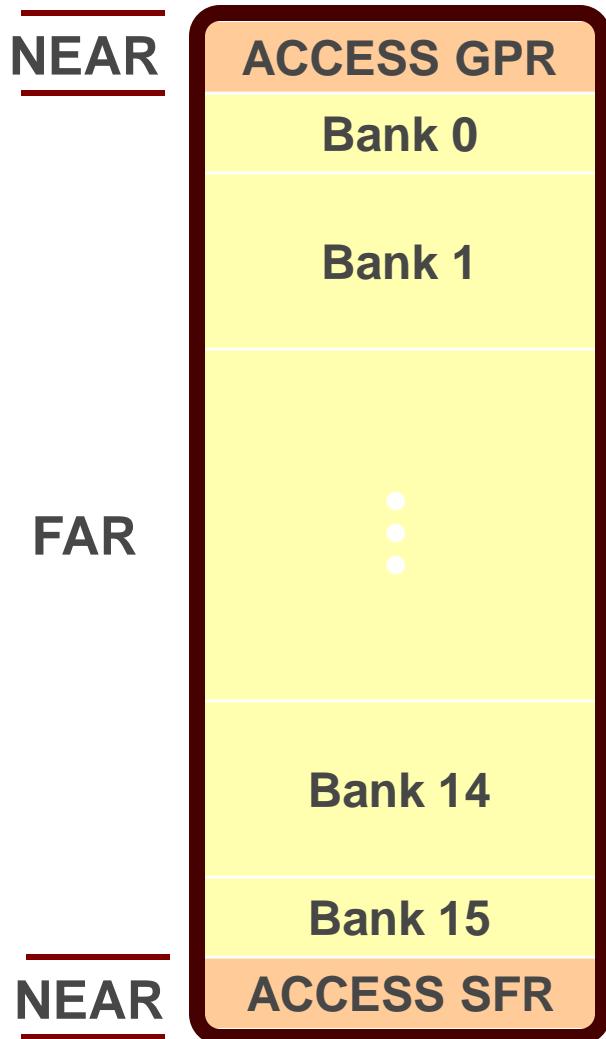
- Specifies that a variable *may* be located in far memory (non-Access RAM)
- Generates bank switching instructions to set the BSR before accessing the variable

Example

```
far int x;  
far char a[10];
```

Locating Variables

Manually Optimizing Data Memory Use



- Use small data model
- Let linker place most variables in the Access bank
- Define infrequently accessed variables with **far** qualifier
 - Makes room in access bank for frequently accessed variables

Locating Variables

Variable Allocation Example



far Variables with the Small Data Model

```
#include <p18f4520.h>
```

```
char a[124]; 1
```

```
#pragma idata myData 2  
far char x = 0xAA;
```

```
int main(void)  
{  
    ...  
}
```

- 1 Variables are **near** and in Access Bank by default
 - a[] fills Access bank along with 4 bytes claimed by tempdata and MATH_DATA
- 2 To create far variables with the small data model:
 - Declare a new section and give it some name
 - Define variables with the **far** qualifier

Locating Variables

How to place a variable in Flash

Syntax

```
rom type identifier;
```

- Specifies that a variable should be located in program memory
- Uses TBLRD to read variable
- Effectively becomes **const** though writes may be done with extra code using TBLWT

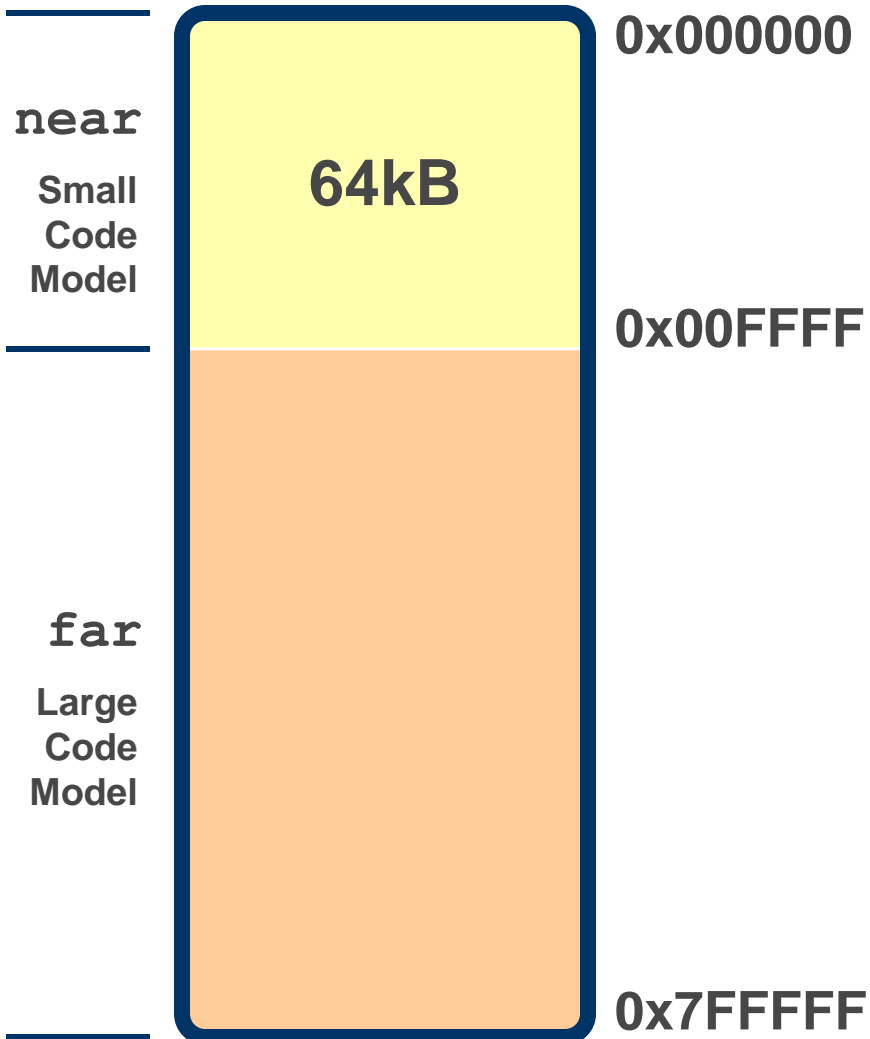
Example

```
rom int x = 320;  
rom char a[10] = "abcdefghi";
```

Locating Variables

How to place a variable in Flash

Program Memory (Flash)



- **near** and **far** also apply to **rom** variables
- **near rom:**
 - Variable in first 64kB
 - ROM Pointers are 16-bit
- **far rom:**
 - Variable anywhere
 - ROM Pointers are 24-bit

Locating Variables

Variable Allocation Example



far Variables in Flash with the Small Code Model

```
#include <p18cxxx.h>

#pragma romdata myNear 1
rom int a = 0x5555;

#pragma romdata myFar=0x10000
far rom int x = 0xAAAA; 2

int main(void)
{
    ...
}
```

- 1** Variables are **near** and in first 64kB by default
- 2** To create **far** variables with the small code model:
 - Declare a new section and give it some name
 - Define variables with the **far rom** qualifier

Locating Variables

Variable Allocation Example



near Variables in Flash with the Large Code Model

```
#include <p18cxxx.h>

#pragma romdata myFar 1
rom int a = 0x5555;

#pragma romdata myNear=0x100
near rom int x = 0xAAAA; 2

int main(void)
{
    ...
}
```

- 1** Variables are **far** and located anywhere by default
- 2** To create **near** variables with the large code model:
 - Declare a new section and give it some name
 - Define variables with the **near rom** qualifier

Locating Variables

Variable Allocation Example



Allocating Variables in Different Regions of Memory (large model)

```
#pragma udata access myUAccess
```

 ← Place uninitialized variables in Access bank

```
near char w;
```

```
near char a[5];
```

← Don't generate bank switching instructions

```
#pragma idata access myIAccess
```

 ← Place initialized variables in Access bank

```
near char x=0xAA;
```

```
near char b[] = "Microchip";
```

← Don't generate bank switching instructions

```
#pragma udata
```

 ← Place uninitialized variables anywhere

```
char y;
```

```
char c[5];
```

```
#pragma idata
```

 ← Place initialized variables anywhere

```
char z = 0x55;
```

```
char d[] = "PIC MCUs";
```

```
rom char e[] = "Hello, world!";
```

 ← Place variable in program memory

Locating Variables

How to place a variable in EEPROM data memory

Syntax

```
#pragma romdata eedata_scn = 0xF00000  
rom identifier = value;
```

- Specifies that a variable should be located in EEPROM data memory
- May be used to initialize EEPROM data
- EEPROM variables may not be read/written like ordinary variables...

Example

```
#pragma romdata eedata_scn = 0xF00000  
rom char eedata_values[4] = {0x01, 0x02, 0x03, 0x04};  
rom int xee = 0x1234;
```

Locating Variables

How to place a variable in EEPROM data memory

Reading variables stored in EEPROM data memory

```
#include <p18f4520.h>

#pragma romdata eedata_scn = 0xF00000
rom char xee = 0xAA;

#pragma udata
char x;

int main(void)
{
    EECON1bits.EEPGD = 0;    // Access EEPROM data memory
    EEADR = (int)&xee;        // Load address of data to read
    EECON1bits.RD = 1;       // Start read operation
    x = EEDATA;              // Result in EEDATA register
}
```

Locating Variables

How to place a variable in EEPROM data memory



Writing variables stored in EEPROM data memory

```
#include <p18f4520.h>

#pragma romdata eedata_scn = 0xF00000
rom char xee;

int main(void)
{
    EECON1bits.EEPGD = 0;    // Access EEPROM data memory
    EECON1bits.WREN = 1;    // Enable writes
    EEADR = (int)&xee;       // Load address to write to
    EEDATA = 0x99;          // Data to write in EEDATA
    EECON2 = 0x55;           // Unlock sequence
    EECON2 = 0xAA;           // Unlock sequence
    EECON1bits.WR = 1;       // Start write operation
    while (!PIR2bits.EEIF)   // Wait for write complete
    ;                       // before next write
    PIR2bits.EEIF = 0;       // Clear EEIF flag
}
```

Interrupts must be disabled for unlock sequence

Locating Variables

How to put related variables in the same bank



Hand Optimization of Variable Allocation

Global Variable Declarations in Program

```
#pragma udata MyRelatedVars
unsigned char MyVar1;
unsigned char MyVar2;
```

```
#pragma udata MyBigArray
unsigned char MyVar3[256];
```

```
#pragma udata
unsigned char MyVar4;
unsigned char MyVar5;
```

```
#pragma code
//Function Prototypes and
//Function Definitions
//Go Here
```

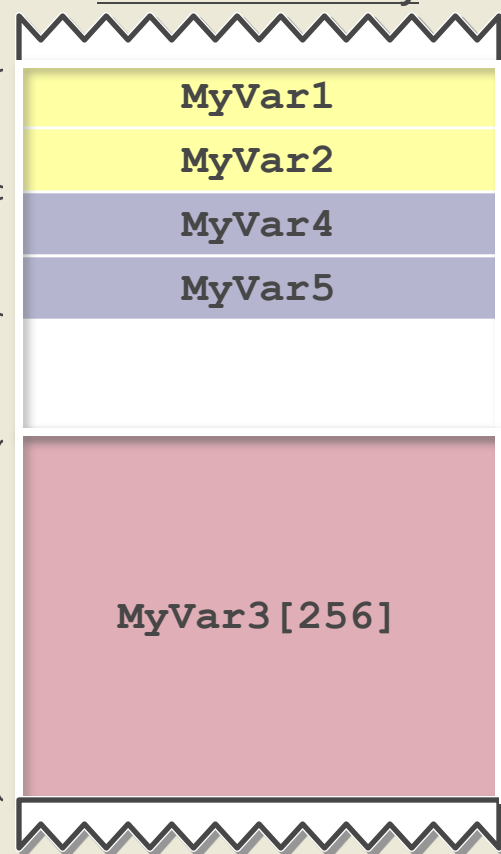
Section Name

MyRelatedVars

.udata

MyBigArray

8-bit Data Memory



Bank

BANK n

BANK n+1

Locating Variables

How to place a variable at a specific address

Syntax

```
#pragma sectiontype sectionname = address
```

- Specifies the address where a variable should be located
- Use sparingly – program will be harder to optimize by linker
- Won't work if *sectiontype* is wrong

Example

```
#pragma udata xSection = 0x100  
far int x;
```

Locating Code

How to place code at a specific address

Syntax

```
#pragma code sectionname = address
```

- Specifies the address where code should be located
- Use sparingly – program will be harder to optimize by linker

Example

```
#pragma code myCode = 0x2000  
int foo(void) { ... }
```

Resources

Microchip PIC18 material

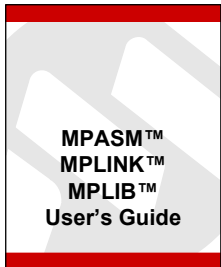
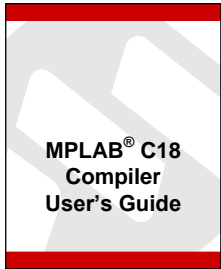
- **PICmicro® 18C MCU Family Reference Manual** (DS39500A)
- ***PIC18F4520 Data Sheet*** (DS39631E)
- ***PICDEM2 Plus Users Guide*** (DS51275D)
- ***More than 100 App notes, design guides and other reference material!***

www.microchip.com

Resources

Books – Compiler Specific

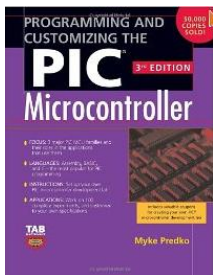
- **MPLAB® C18 C Compiler User's Guide**
Current Edition (PDF)
Microchip Technology
DS51288J
<http://www.microchip.com>
- **MPASM™ MPLINK™ and MPLIB™ User's Guide**
Current Edition (PDF)
Microchip Technology
DS33014K
<http://www.microchip.com>



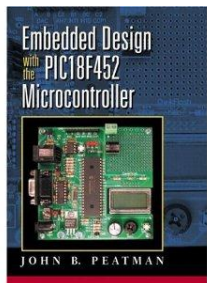
The older books on C are much more relevant to embedded C programming since they were written back when PCs and other computers had limited resources and programmers had to manage them carefully.

Reference Books

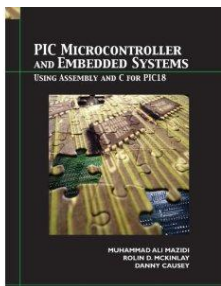
Books on PIC18



- **Programming and Customizing PIC Microcontrollers**
3rd Edition (September 27, 2007)
Myke Predko
ISBN-10: 0071472876
ISBN-13: 987-0071472876



- **Embedded Design with the PIC18F452 Microcontroller**
1st Edition (August 15, 2002)
John B. Peatman
ISBN-10: 0130462138
ISBN-13: 978-0130462138



- **PIC microcontrollers and Embedded Systems**
4th Edition (October 19, 2006)
Muhammad Ali Mazidi , Rolin McKinlay, Danny Causey
ISBN-10: 0131194045
ISBN-13: 978-0131194045



MICROCHIP

Regional Training Centers

Thank You

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KeeLoq, KeeLoq logo, MPLAB, PIC, PICmicro, PICSTART, PIC³² 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.

© 2010, Microchip Technology Incorporated, All Rights Reserved.