



MICROCHIP

Regional Training Centers

MCC RTC

MPLAB® X IDE Code Configurator (MCC)

動手做實驗課程

PIC18F45K22

MPLAB® X IDE Code Configurator (MCC) 的特色

- 使用圖形化界面(GUI) 來規劃周邊的設定及使用方式後，透過程式碼產生器來產生周邊的定義檔 (.h) 及函數庫原始程式 (.c)
 - ◆ 取代 C 編譯器所提供的函式庫
 - ◆ 快速規劃周邊所需設定及所需操作函式庫
 - ◆ 圖形化的規劃設定，簡單迅速上手
 - ◆ 有錯隨時修正，重生產生原始程式
 - ◆ 產生的 .h 檔及 .c 檔可依所需加以更改



安裝 MCC 外掛程式 (Plugin)

MPLAB[®] X IDE Code Configurator (MCC)
外掛程式需事先安裝完成才可以做本教育訓練的的練習

關於 MCC 版本

- 注意：本投影片內容是使用 **MCC v3.36** 版本
 - ◆ 請確定是安裝此版本來做教材中實作練習
 - ◆ 搭配MPLAB X: v4.05
- **MCC 的下載來源：**
 - ◆ **www.microchip.com** 網站可以做依版本選擇的下載
 - ◆ 台灣網站下的教育訓練光碟連結
 - **http://www.microchip.com.tw/Data_CD/**
 - ◆ **MPLAB® X IDE** 中 **Tools** 功能表的 **Plug-in** 選項裡
- **Microchip** 隨時會更新 **MCC** 的版本，使用較新版本 **MCC** 開啟專案時可能會出現周邊設定的 **GUI** 畫面的差異，**MPLAB X IDE** 所開啟的專案在不同版本**MCC** 間可能須要重新設定。
 - ◆ 在 **www.microchip.com** 網頁，搜尋“**MCC**”

兩種安裝 MCC 外掛模組

- 離線安裝方式

- ◆ 需事先下載 **MCC**安裝模組於硬碟裡
- ◆ 可以自選要安裝的版本
- ◆ **MCC v3.36**

- **Windows : com-microchip-mcc-3.36.nbm**



- 線上安裝方式

- ◆ 直接於 **MPLAB XIDE** 中的 **Tools** 功能表選擇 “**Plugins**” 項目
- ◆ 在 **Available Plugins** 的選項卡裡可以找到 **MPLAB Code Configurator** 來安裝
- ◆ **Available Plugins** 永遠指向最新版本！

離線安裝方式

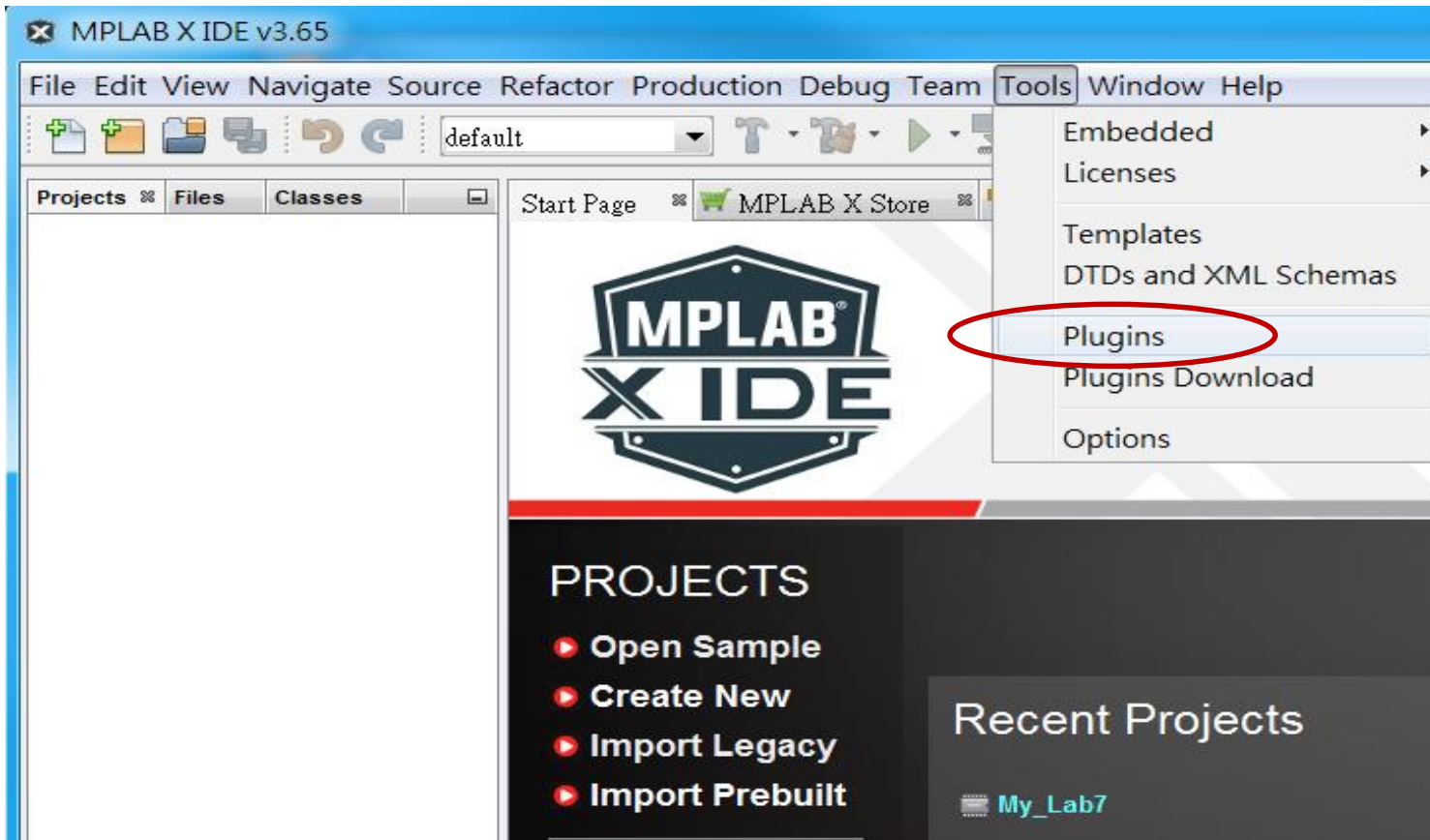
- 於台灣網站下載 **MCC** 外掛模組：
 - ◆ http://www.microchip.com.tw/Data_CD/
- 至美國 **Microchip** 網站搜尋” **MCC**” 後，選擇”**MPLAB® Code Configurator**”
 - ◆ 選擇要下載的版本 (本課程使用 V3.36)
 - ◆ 在 **MCC** 的首頁選擇 “**Current Download**” 的選卡項裡下載 “**MCC MPLAB® X Plugin**”到硬碟裡 (檔案名稱: **com-microchip-mcc-3.36.zip**)
 - ◆ 如果 **3.36** 已經非最新版本，請至 “**Archive Download**” 選項卡下載
 - 下載完成後起解壓縮以便安裝時使用

儲存 MCC 外掛程式

- 將 **com-microchip-mcc-3.36.zip** 解壓縮到硬碟裡，請記住檔案路徑
- 解壓縮後的檔名為：
 - ◆ **com-microchip-mcc-3.36.nbm**
- 啟動 **MPLAB® X IDE**

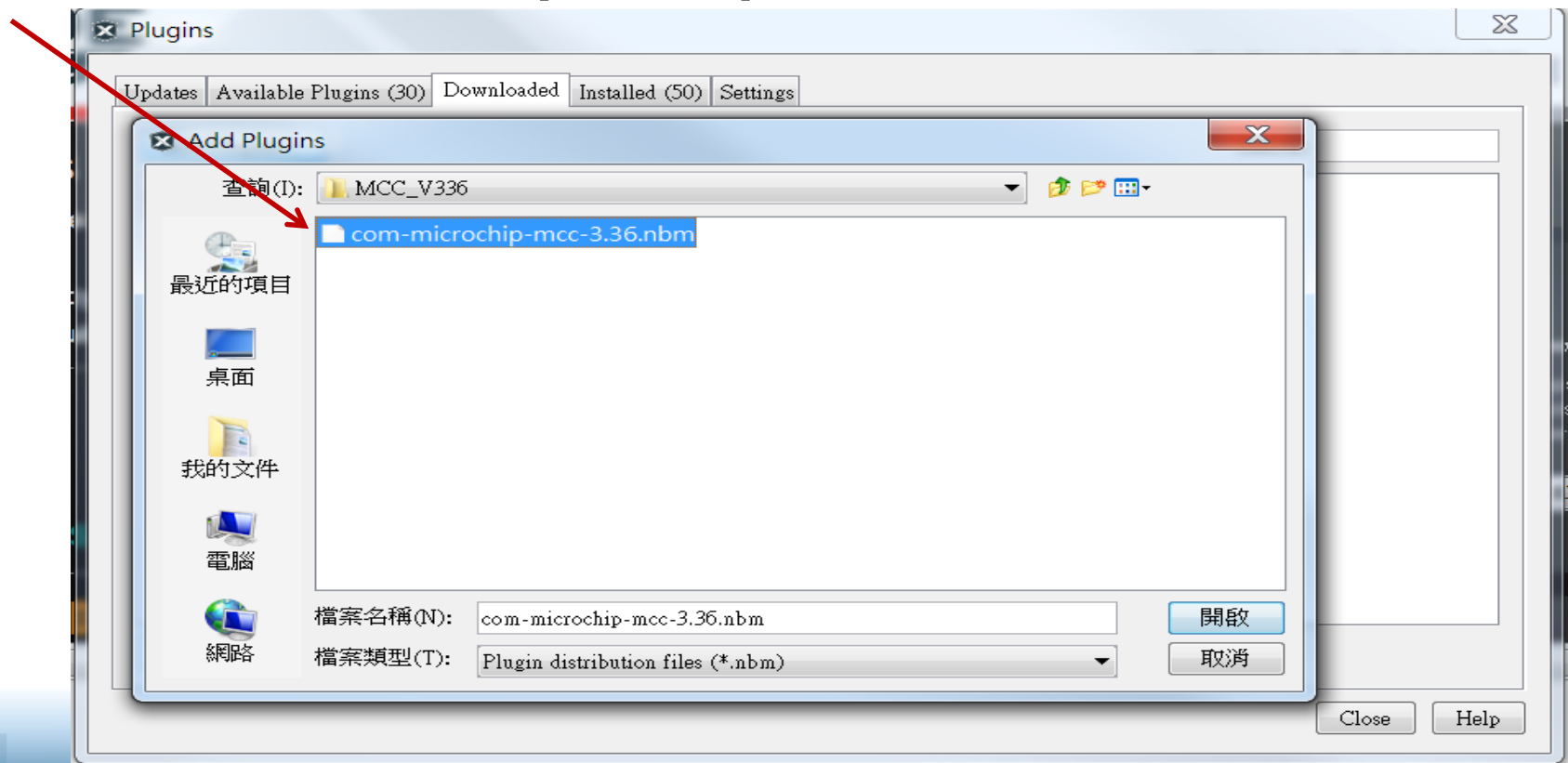
離線加掛 MCC 模組

- 點選 Tools 下的 “Plugins”



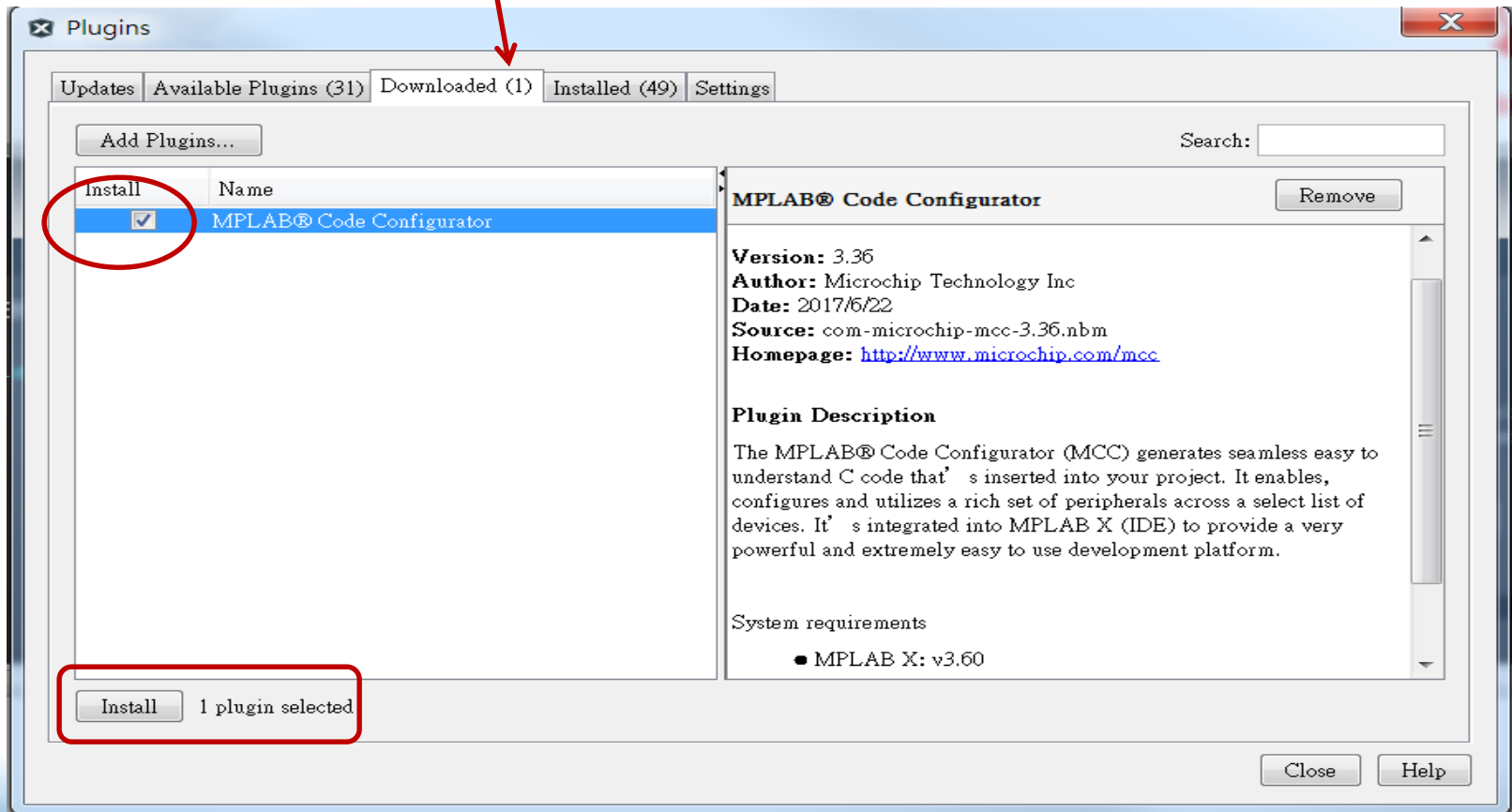
離線加掛 MCC 模組

- 到 Downloaded 選卡裡，點選“Add Plugings”後，找出剛解壓縮的路徑的 MCC Module 檔案 (.nbm)，點選該檔案後按開啟



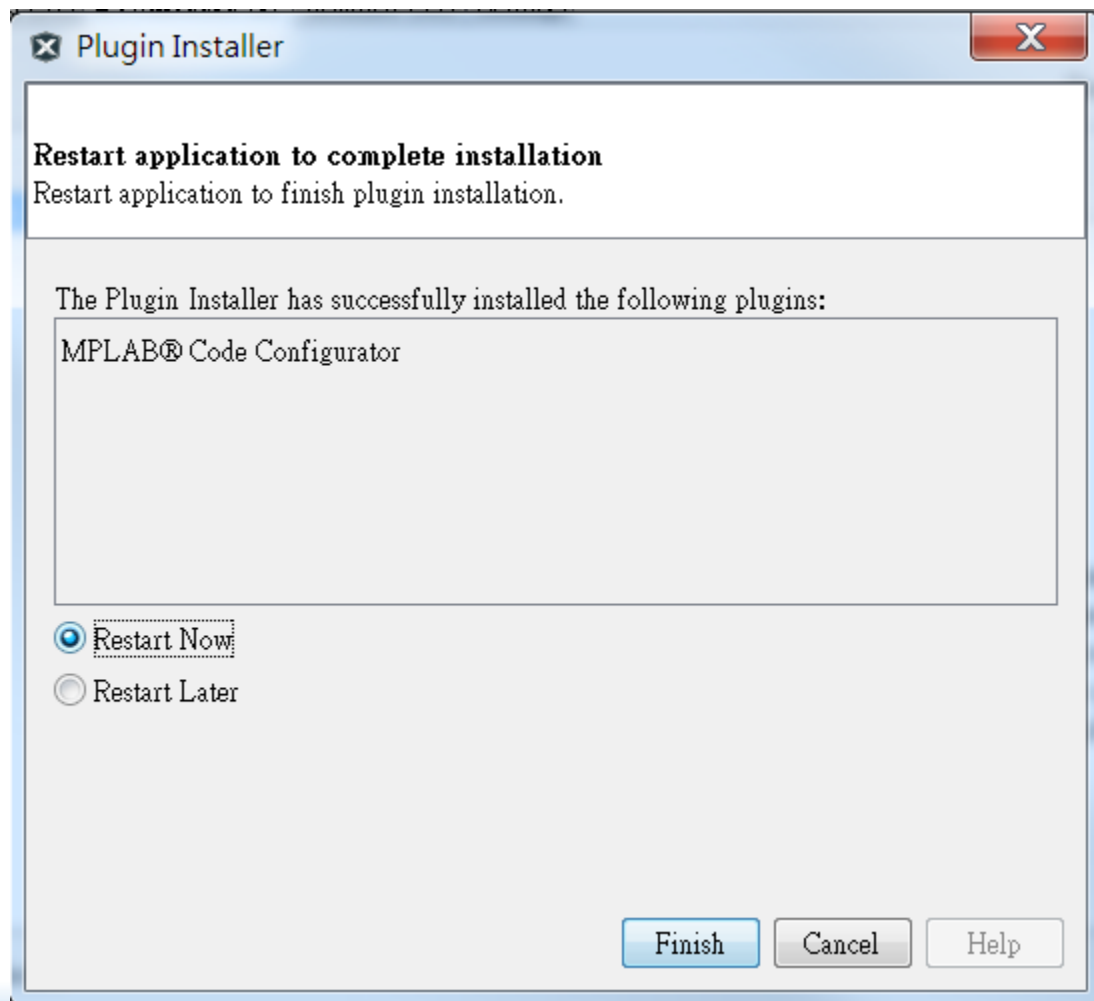
離線安裝 MCC 模組

- 這時 “Downloaded” 選卡會顯示有一個模組，勾選 MPLAB® Code Configurator 後，按下底下的 “Install” 來安裝



離線安裝 MCC 模組完成

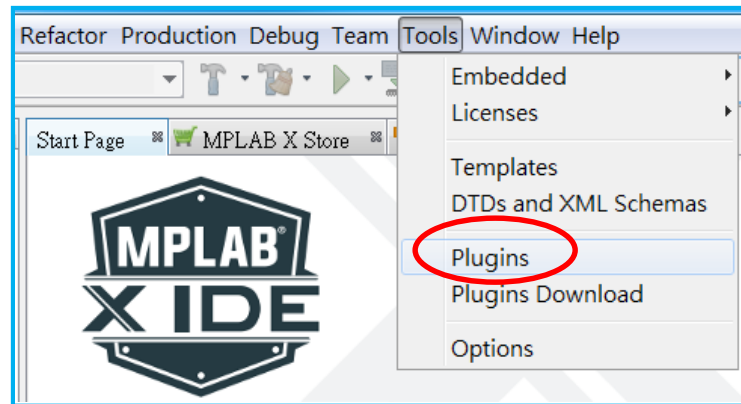
- 安裝完成選擇
“Restart Now”
- 重新啟動
MPLAB® X IDE
以便使 **MCC** 安
裝後時所做的改
變生效



線上安裝 MCC 步驟

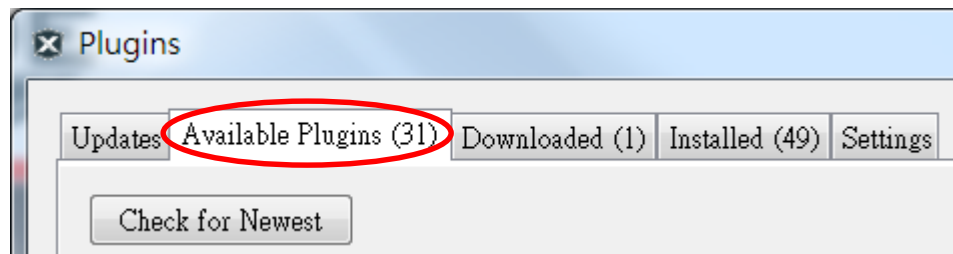
1.

到 “Tools” 功能表
點選 “Plugins” 的
選項



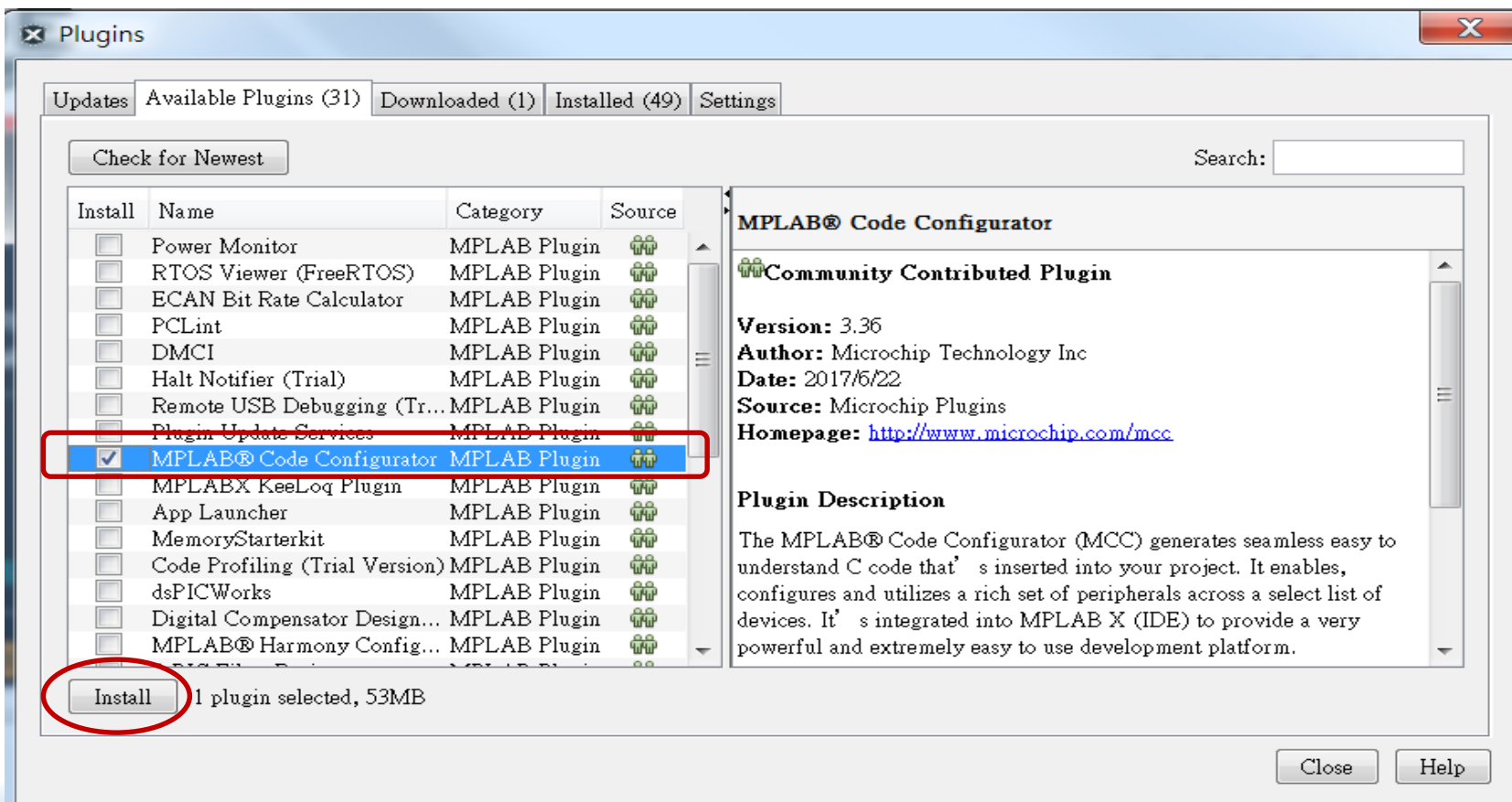
2.

選取 “Available
Plugins” “選項卡下
來找尋可用的外掛
模組



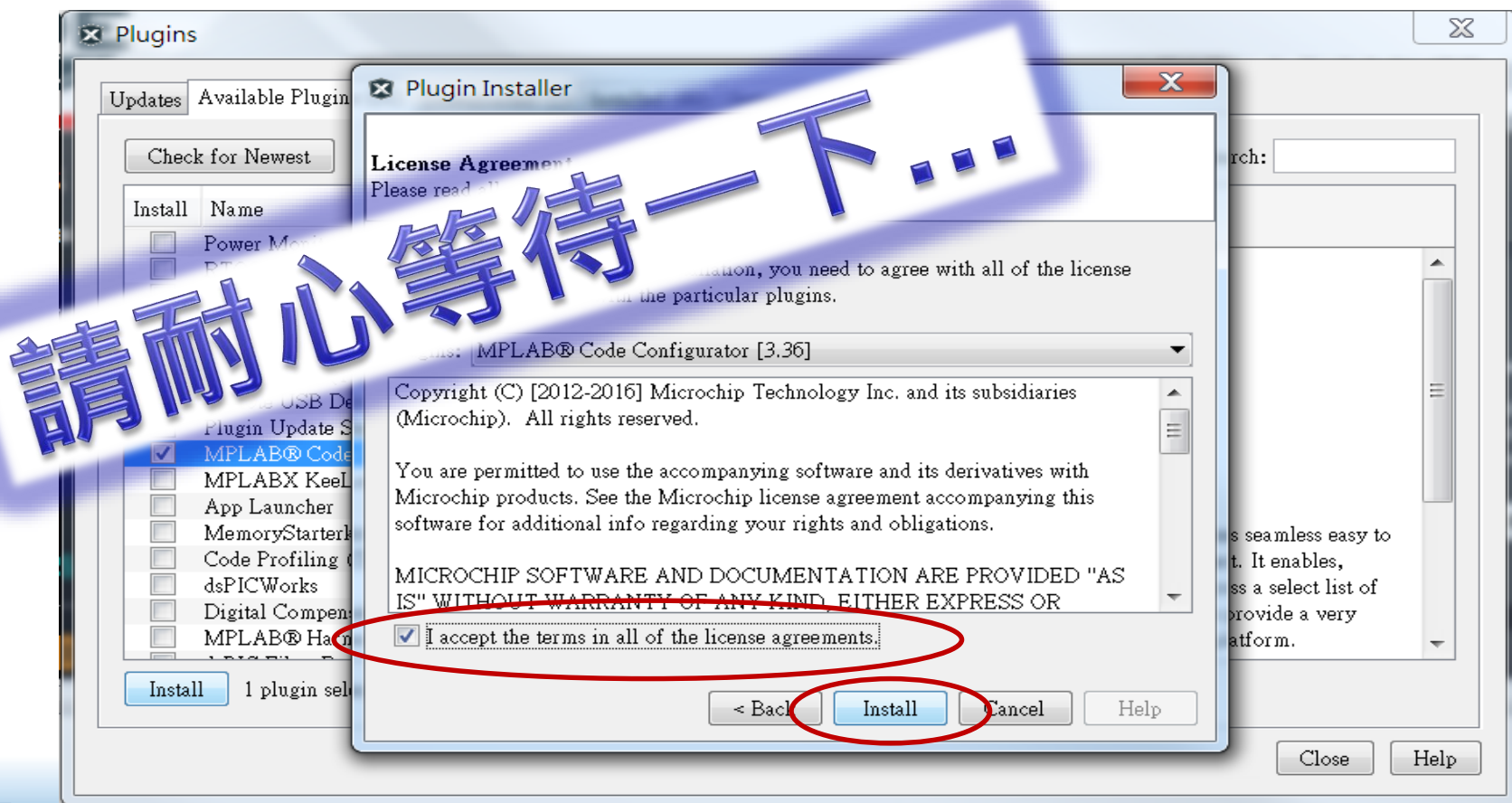
線上安裝 MCC 步驟 (Continue)

3. 在眾多的 Plugins 找到 “MPLAB® Code Configurator”，勾選後再按 “Install”



線上安裝 MCC 步驟 (Continue)

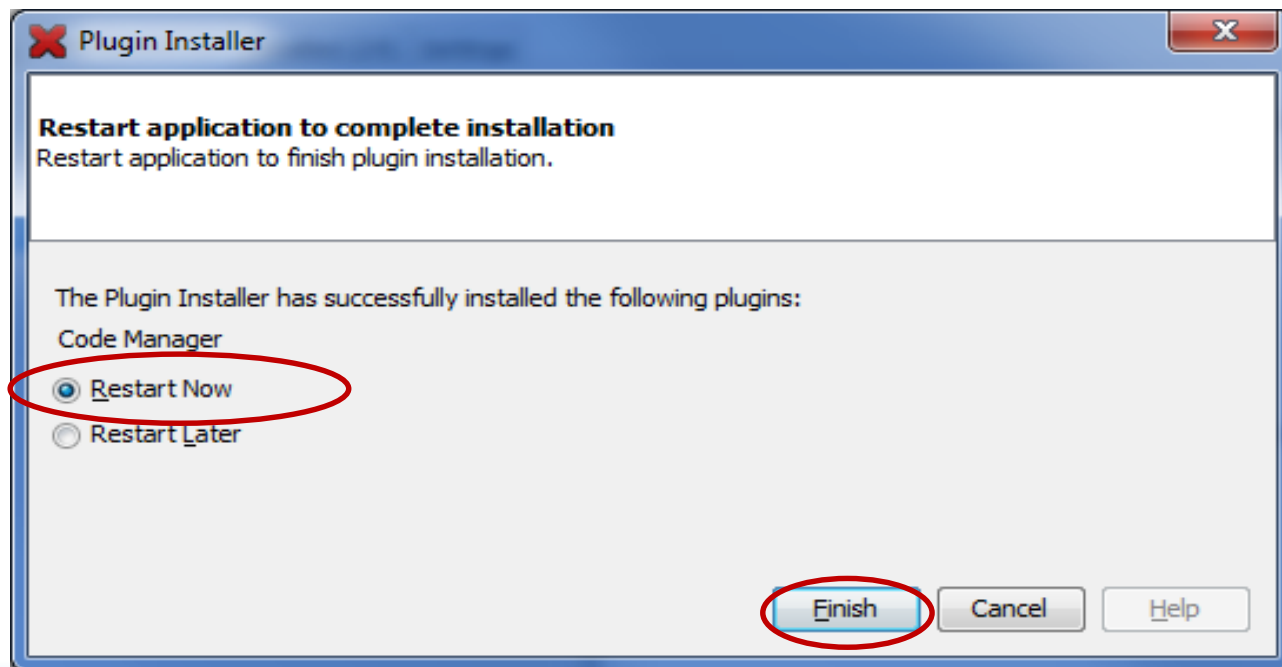
4. 勾選同意此授權協議後點選 "Install" 選項
後開使上網下載 MCC 模組並安裝 MCC



線上安裝 MCC 步驟 (Continue)

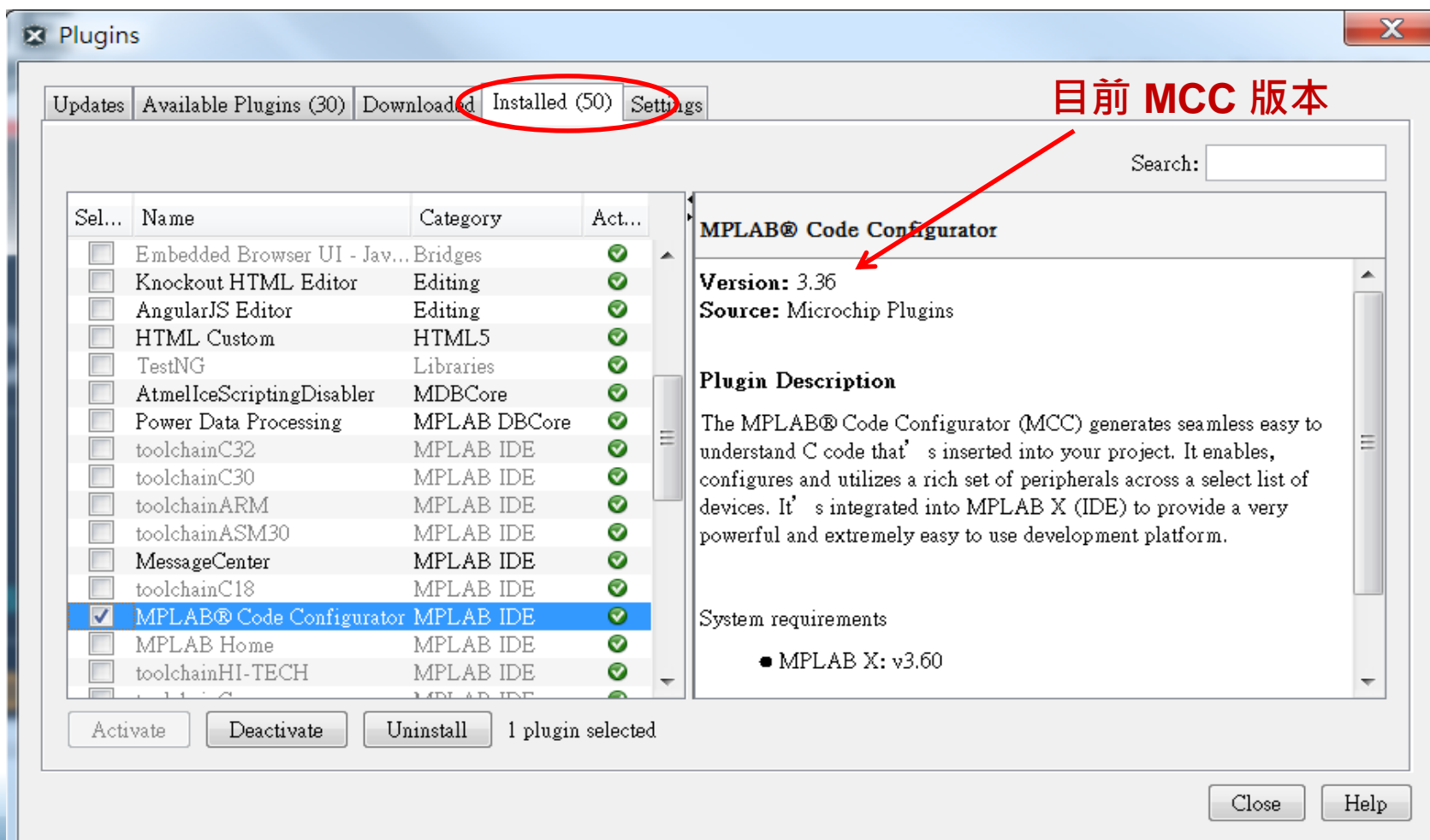
5.

安裝完成後需重新啟動 MPLAB® X IDE
請選擇 “Restart Now” 的選項
選擇重新啟動按 “Finish”



檢查 MCC 是否安裝完成

- 不管是線上或離線安裝，重新啟動後，請檢查外掛程式 (Plugins) 選項卡裡檢視 MCC 是否安裝就緒 (如藍色所示)





開發工具與實驗版

使用的開發工具

PICKit™ 3

- USB 2.0 Full Speed (12MHz)，提供及時除錯/模擬功能，有全速執行、暫停、單步執行、斷點功能
- 透過 MPLAB® X IDE 可檢查版本韌體自動升級
- 可直接開啟對外供電功能最高到 30mA
- V_{DD} range 3 - 5.5V
- V_{PP} range 3 - 13V
- CE and RoHS-compliant



MPLAB
CERTIFIED

使用的開發工具

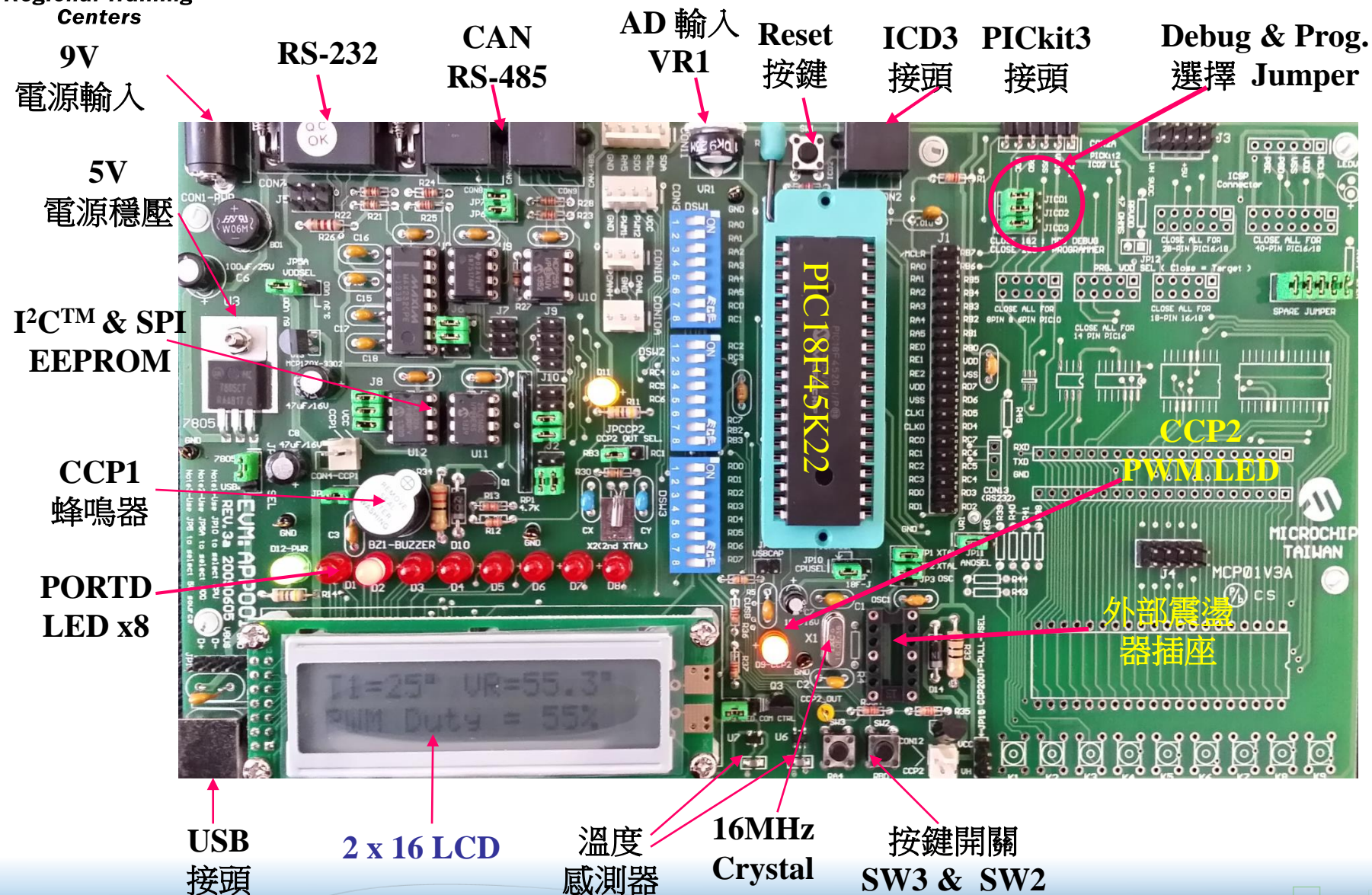
MPLAB® ICD 3

- 全系列 **PIC®** 產品的支援
- 多項增加的功能
 - ◆ 可供應外部電源 (100mA)
 - ◆ 更快速 – USB HS, HW 加速, SRAM
 - ◆ 軟體斷點設定(1000)
 - ◆ 及時除錯監視功能 (Real-Time Watch)
- 只需由 **USB** 供電
- 無須外加電源供應器
- 高壓保護電路 **ICSP** 除錯/燒錄腳位
- 過電流保護
- 自我測試模組
 - ◆ 檢測介面及連線是否正確
 - ◆ 協助判斷是工具的錯誤還是目標板的故障
 - ◆ 測試模組內建 **8-pin PIC®** 偕同測試



DV164035

認識 APP001 實驗板











有關實驗用零件申請

- **APP001 板子**

- ◆ 原安裝 PIC18F4520
須換成 PIC18F45K22
- ◆ 確認已換裝24LC02B 於 U12 才能做
課程中的Lab 4-1的練習

MCC v3.36目前支援的元件

- **MCC 支援 8/16/32 位元的 MCU, 詳細的元件列表請看各個系列的 Device Libraries 之 Release Notes**

Features				
Current Download				
Archive Download				
Documentation				
Current Version				
Title	Version	Date Published	Release Notes	D/L
Plug In				
MPLAB® Code Configurator	v3.36	7/11/2017		
Device Libraries				
PIC10/PIC12/PIC16/PIC18	v1.45	7/11/2017		
PIC24/dsPIC33/PIC32MM	v1.35.1	7/11/2017		
PIC32MX	v1.35	4/6/2017		

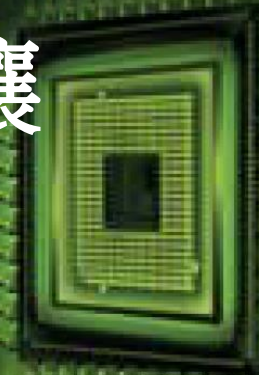


MICROCHIP

Regional Training Centers

PIC18 Lab 1:

使用 **Timer1** 做一秒計時讓
LED 轉態一次 (Toggle)



PIC18 Lab 1 目標

- 啟動 **MPLAB® X IDE** 的 **MCC** 功能
- 設定組態工作模式(**Config. Reg**) 及震盪器的各項設定
- 設定基本輸出、輸入腳 (**I/O**) 功能
- **Timer1** 一秒的計時設定
- 利用程式產生器來產生所設定的周邊程式
- 配合周邊的程式來撰寫主程式

PIC18 Lab 1 概述

1. **APP001 硬體上電並接上 PICKit3**
2. **開啟 MPLAB® X IDE**
3. **開啟專案，目錄及專案檔案名稱：**
 - ◆ **..\PIC18F MCC RTC\labs\lab1.X**
4. **開啟 MCC (MPLAB® Code Configurator)**
5. **PIC® 的配置設定 (Config. Bits)**
6. **I/O 的設定**
7. **Timer 的設定**



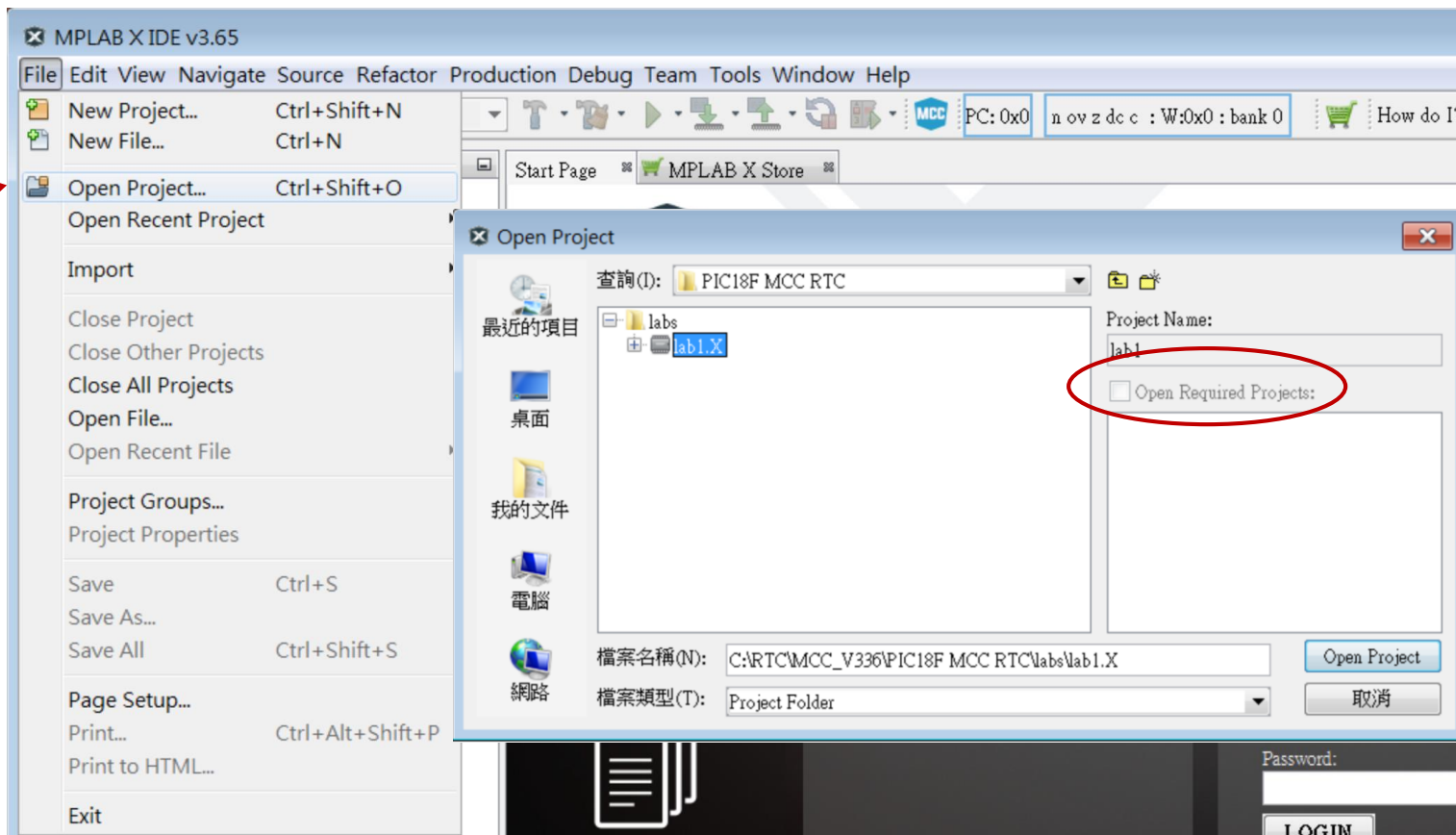
動手跟這做實驗

PIC18 Lab1

PIC18 Lab 1

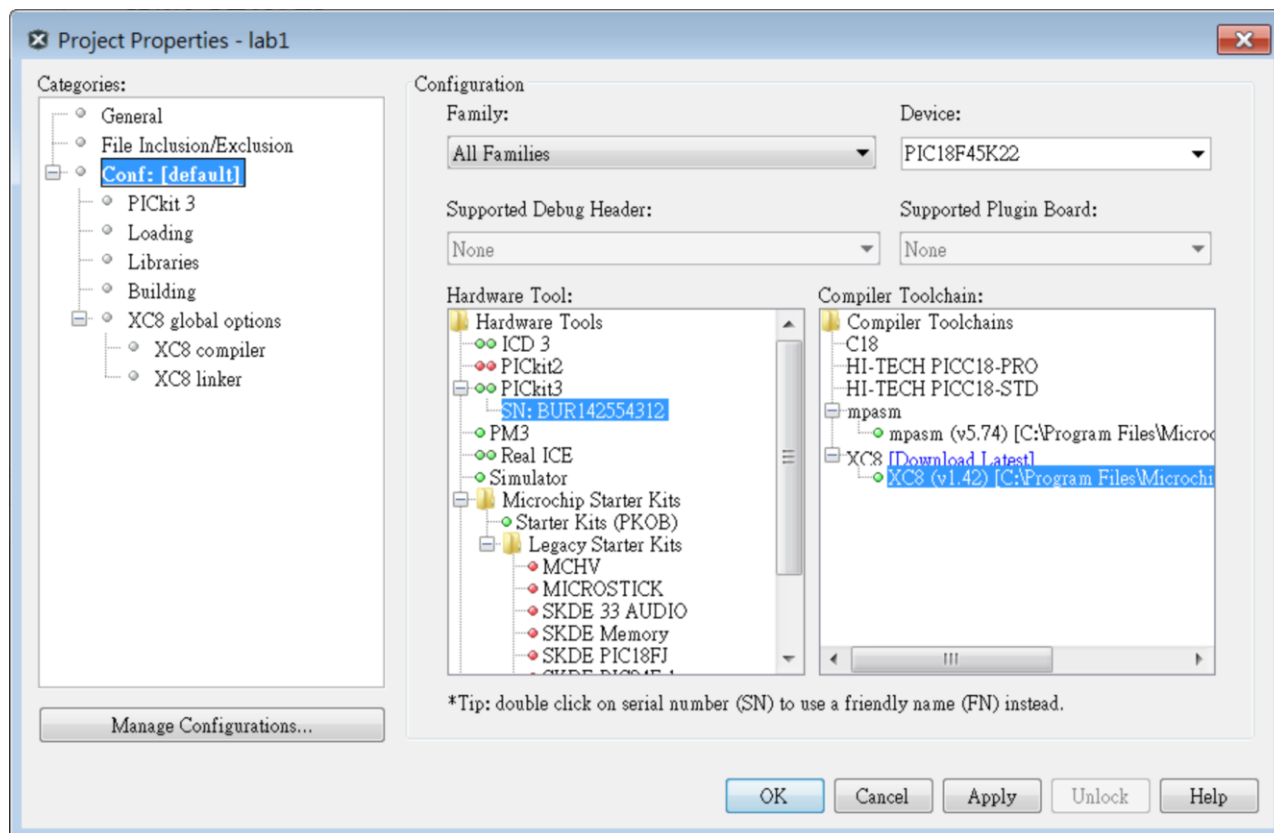
開啟 LAB1 的專案

- 在 MPLAB® X IDE 下，點選 ” **Open Project** ”
再選擇專案 “ **lab1.X** ”



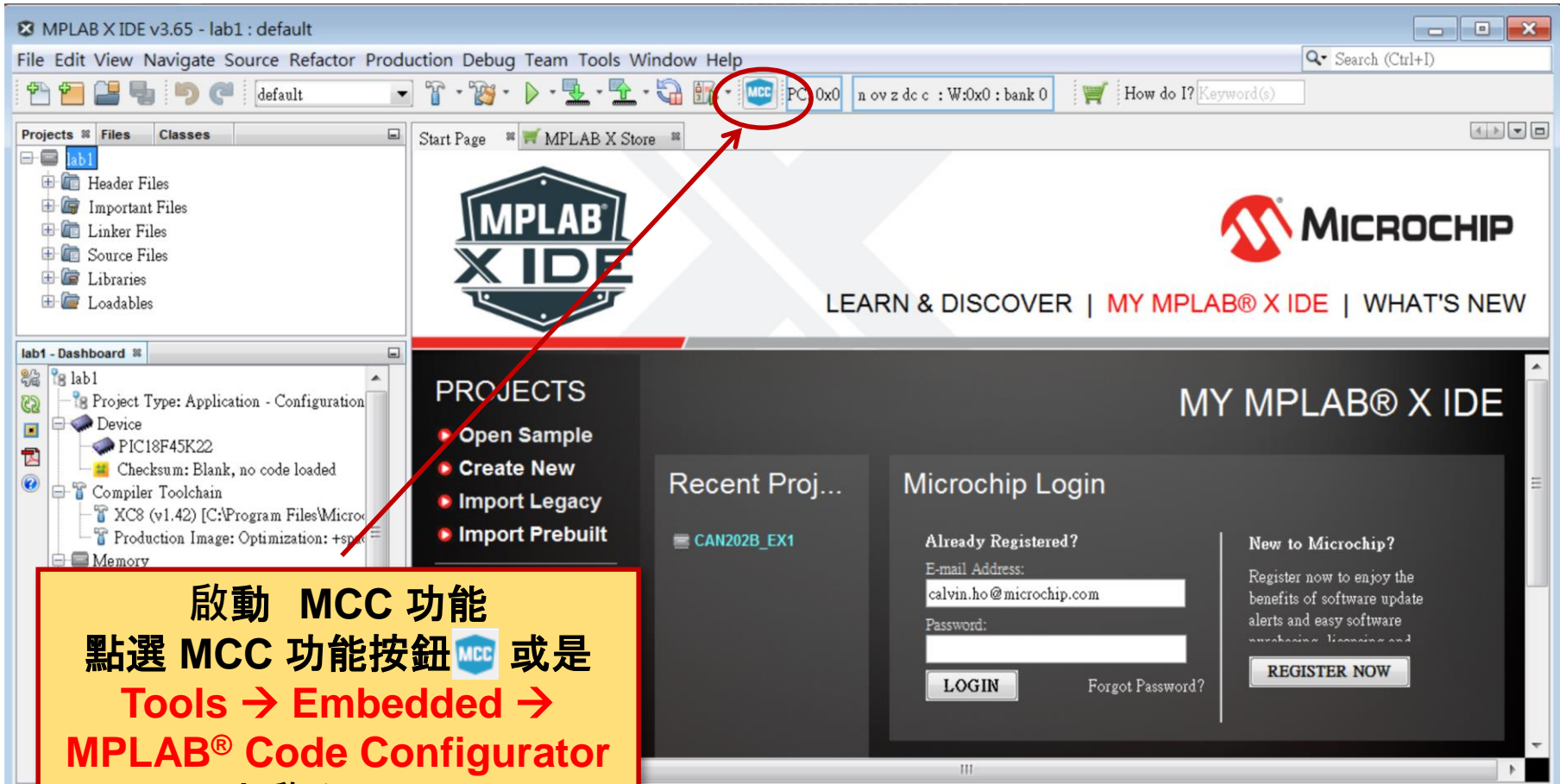
確定選用元件及開發工具


- 在專案視窗下用老鼠右鍵點選“**Lab1**”後，在點選“**內容**”選項 (Properties)
- **PIC18F45K22**
- **PICkit3**
 - 點選序號
- **XC8 (V1.4x)**



PIC18 Lab 1

● 啟動 MPLAB® Code Configurator



啟動 MCC 功能
點選 MCC 功能按鈕  或是
Tools → Embedded →
MPLAB® Code Configurator
來啟動 MCC

MPLAB® Code Configurator

Pure Engineering Nirvana



- **Generates seamless, easy to understand Drivers and Initializers that can be inserted into your project**
 - ◆ Enables, configures and utilizes a rich set of peripherals across many of Microchip's most popular PIC® microcontrollers
 - ◆ Generated C code can be easily modified and debugged
- **Leverage drivers and GUI interface to reduce time to market**
- **Reliable, Small Footprint and Efficient**
 - ◆ Generated C code is reliable and designed for efficient use MCU resources
- **Powerful, easy to use plug-in development tool for MPLAB® X IDE**
 - **Download this powerful development tool for FREE at:**
www.microchip.com/MCC

MPLAB® Code Configurator

Pure Engineering Nirvana

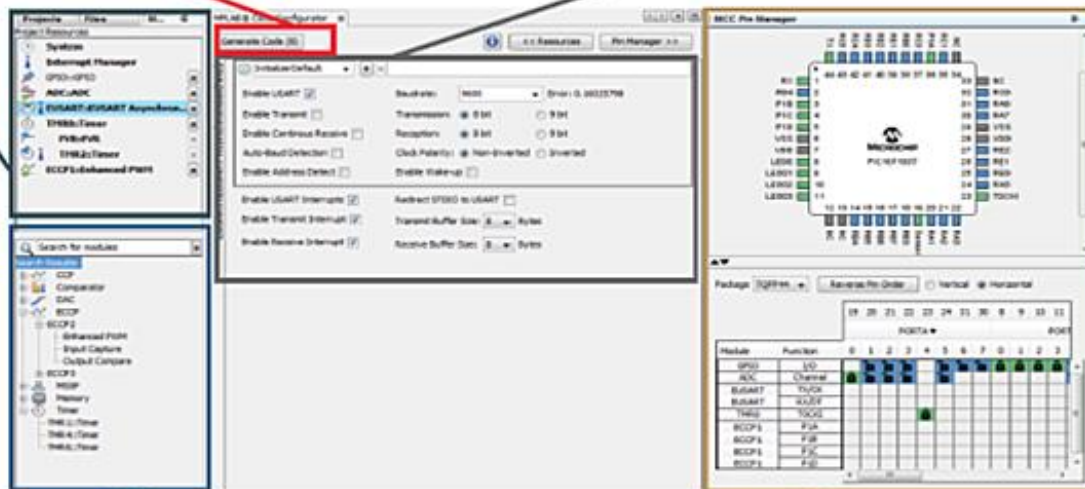
Project Resources Section

- Includes all of the peripherals or functions you want to use in your project
- Each peripheral has a simple user interface to set up the basic configuration and related pins

Click the "Generate Code" button and view the source file for your project

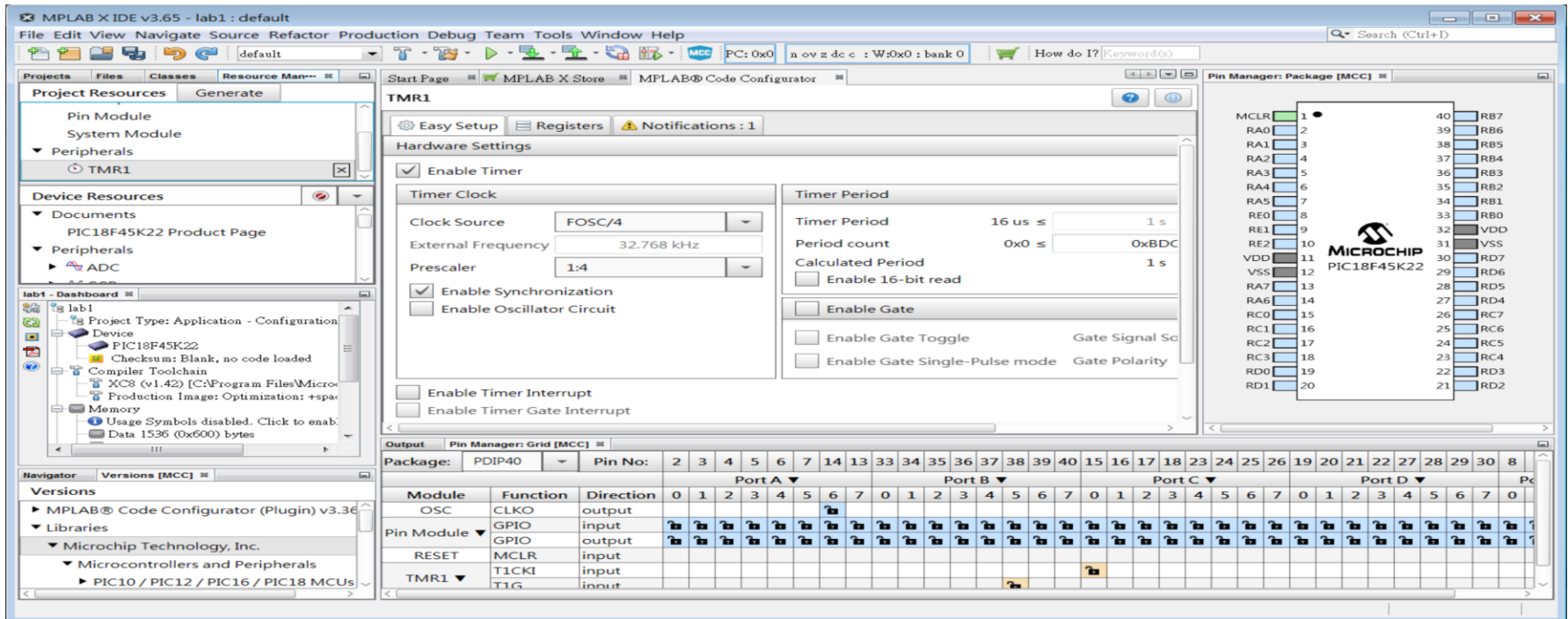
Module Composer Window

- This window changes depending on the peripheral selected in the project resources section so there is a customized GUI to setup each different peripheral
- Easily set up your GPIOs as inputs or outputs and with the snap of a button—configure them to start high or low and show which pin you want them to be active on
- Easy graphical setup of logic gates for the Configurable Logic Cells (CLC)
- Allows saving custom names for easy code readability



MPLAB® Code Configurator

Pure Engineering Nirvana



- **Device support**
 - Support 200+ devices
 - 40+ PIC24 microcontrollers tool
 - PIC32 MCU supported too

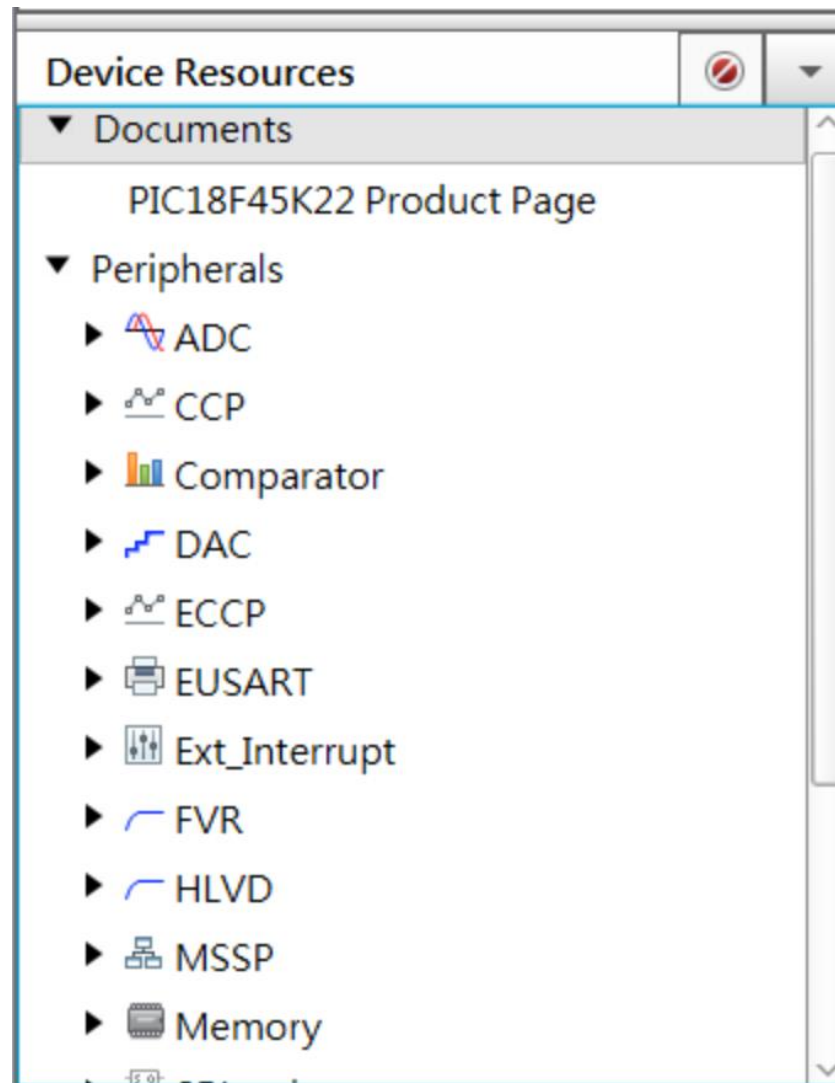




Device Resource (PIC18)

此PIC內部硬體及相關文件

Device Resource視窗



Device Resource

- **ADC**
- **CCP**
- **Comparator**
- **DAC(Digital-to-Analog Converter)**
- **ECCP**
- **EUSART**
- **FVR**

Device Resource

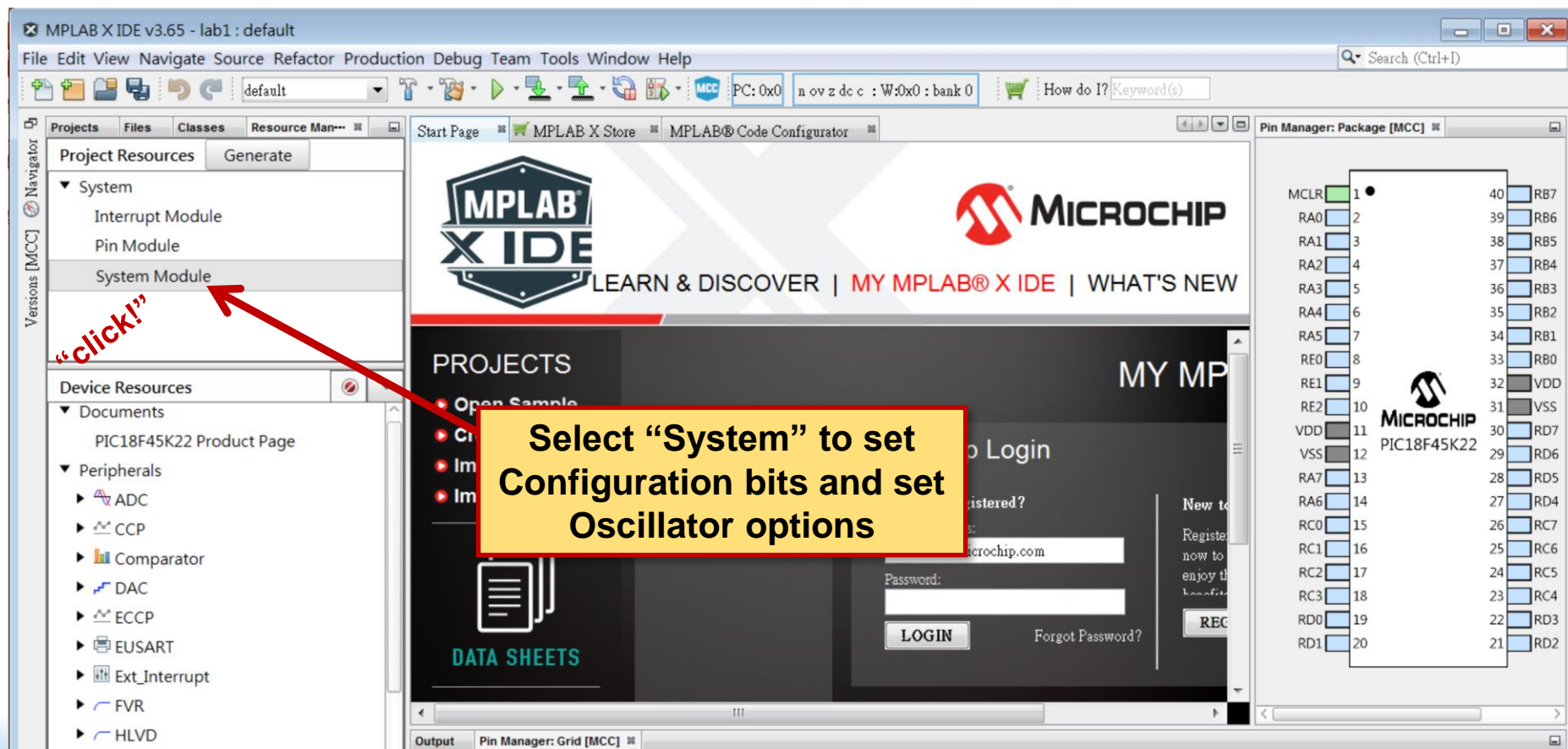
- **GPIO**
- **HLVD**
 - ◆ High/Low-voltage Detection
 - ◆ Programmable 16-Level
 - ◆ Interrupt on High/Low-Voltage Detection
- **MSSP**
- **Memory**
 - ◆ DataEE (Up to 1024 Bytes Data EEPROM)
- **SRLatch**
- **Timer**



Configuration Bits 的設定

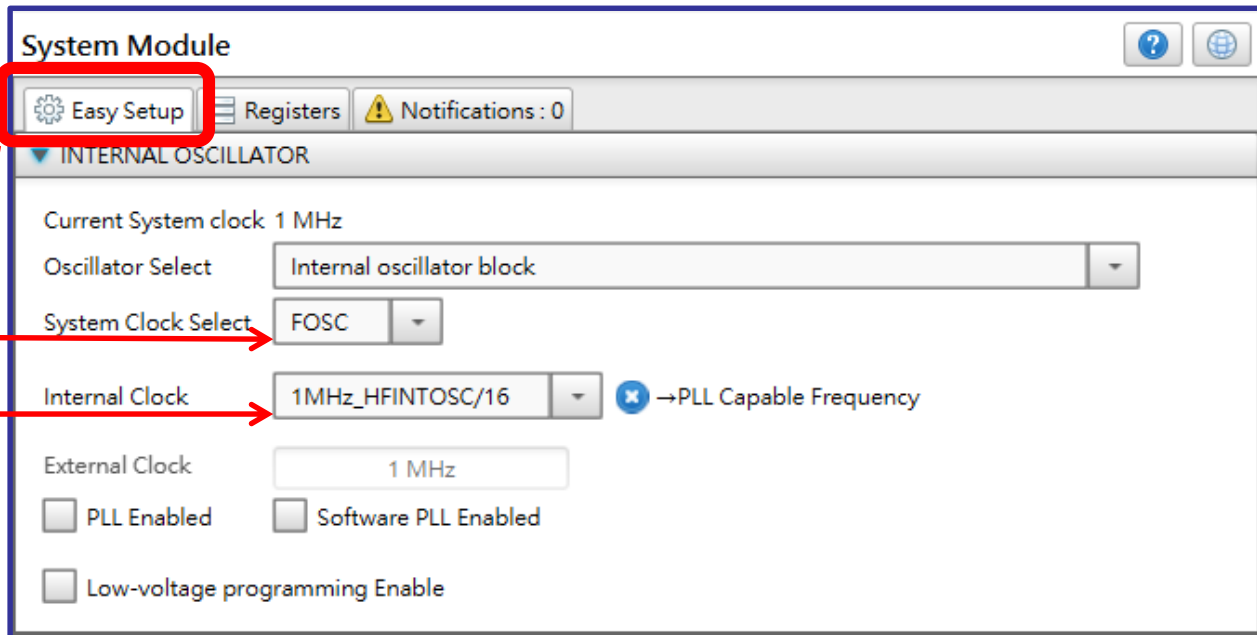
Config. Bits 的設定

- 點選 “System” 開始設定 Configuration bits 及 Oscillator



PIC18 Lab 1

- **System Clock 選擇 Internal Oscillator**
 - ◆ INTOSC(內部振盪器)
- **Internal Clock 選擇內定的**
 - ◆ 1MHz_HF



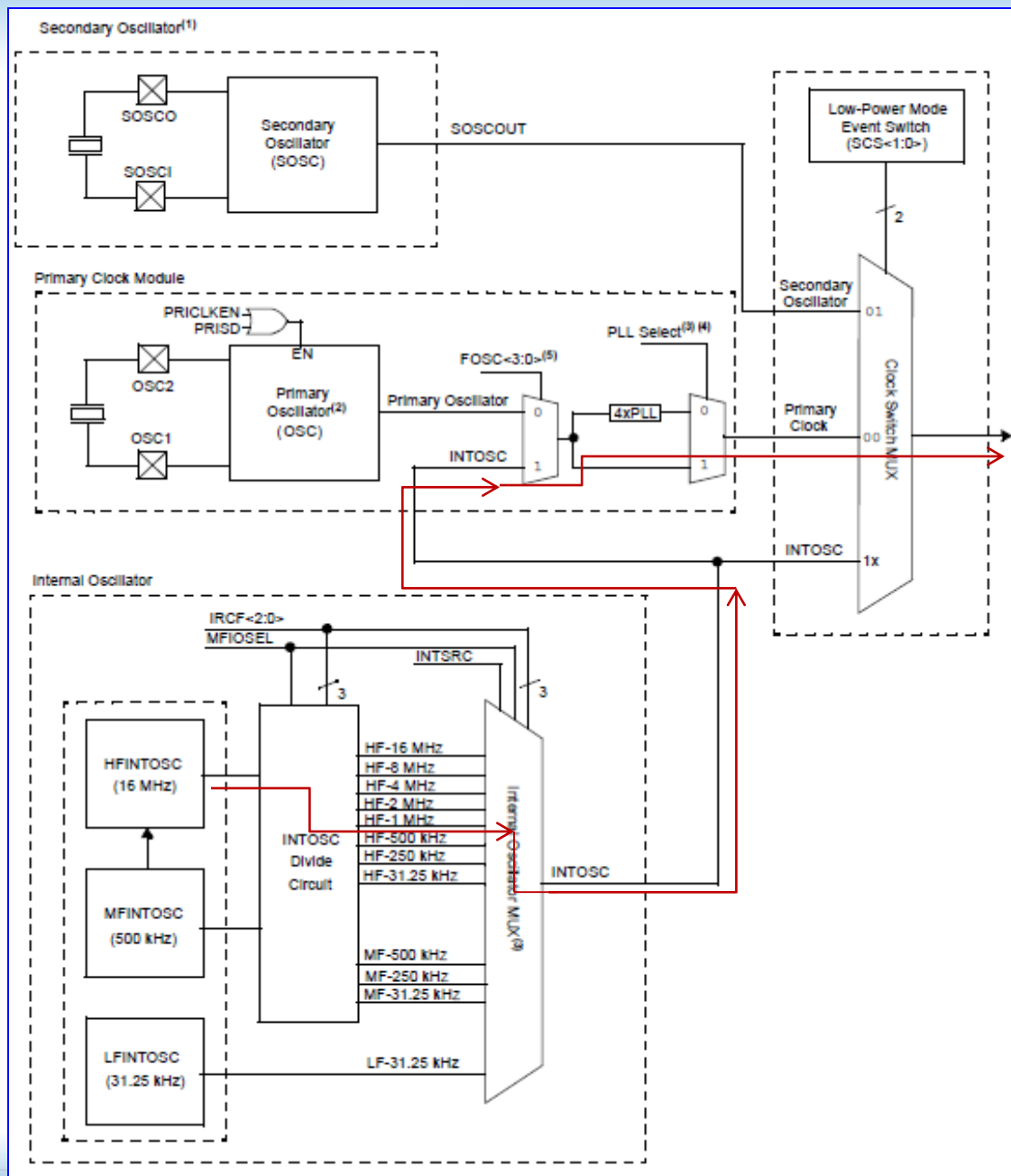
在Easy Setup頁面裡
確定工作頻率 (Fosc)
為1 MHz



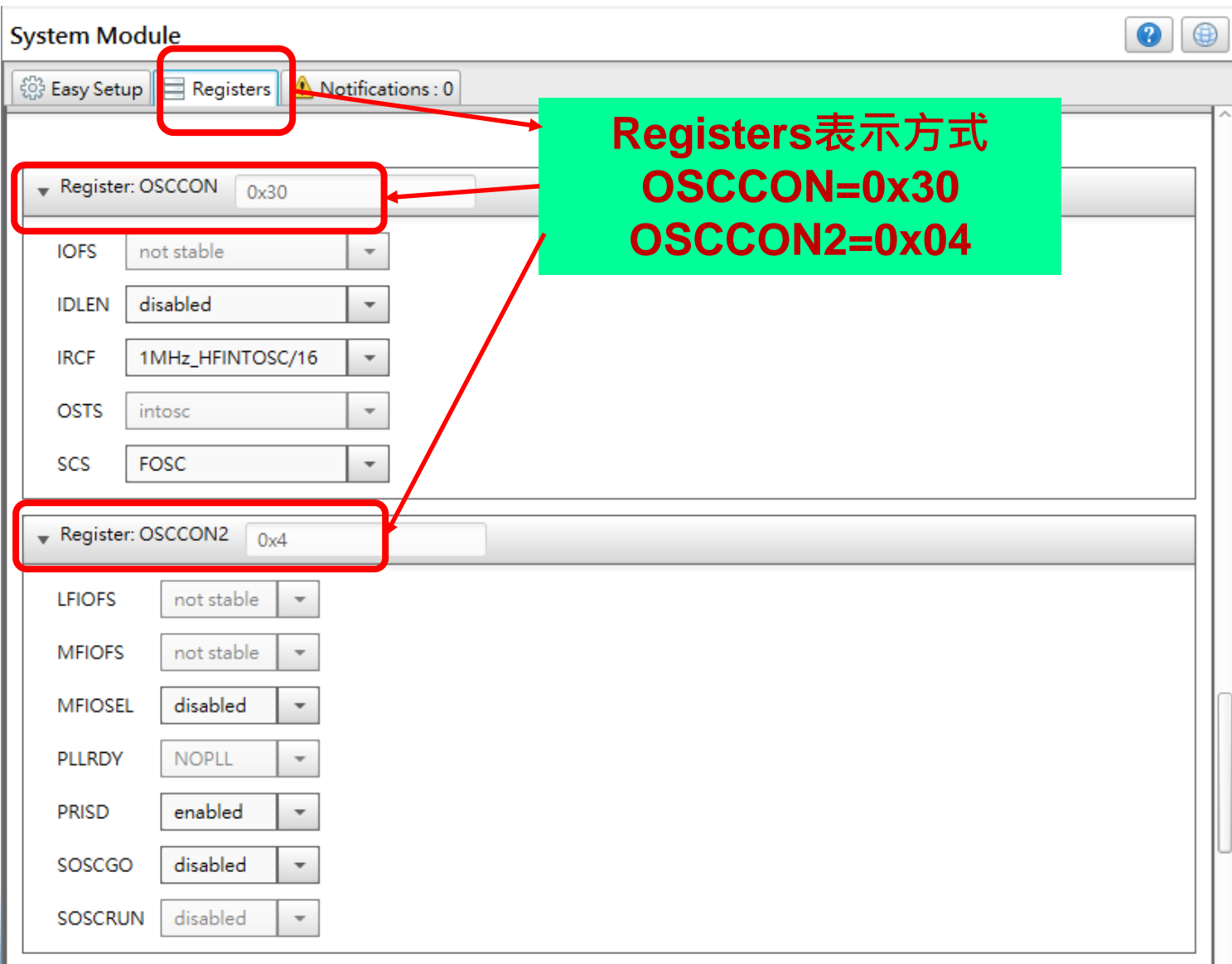
● PIC18F45K22 震盪器方塊圖

➤ **FOSC**

➤ **HF- 1MHz**



震盪器的設定 (OSCCON 及 OSCCON2)



System Module

Easy Setup Registers Notifications : 0

Register: OSCCON 0x30

IOFS not stable

IDLEN disabled

IRCF 1MHz_HFINTOSC/16

OSTS intosc

SCS FOSC

Register: OSCCON2 0x4

LFIOFS not stable

MFIOFS not stable

MFIOSEL disabled

PLLRDY NOPLL

PRISD enabled

SOSCGO disabled

SOSCRUN disabled

**Registers表示方式
OSCCON=0x30
OSCCON2=0x04**

2.3 Register Definitions: Oscillator Control

REGISTER 2-1: OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0	R/W-0	R/W-1	R/W-1	R-q	R-0	R/W-0	R/W-0
IDLEN	IRCF<2:0>			OSTS ⁽¹⁾	HFIOFS	SCS<1:0>	
bit 7							bit 0

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

q = depends on condition

-n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

bit 7

IDLEN: Idle Enable bit

1 = Device enters Idle mode on *SLEEP* instruction

0 = Device enters Sleep mode on *SLEEP* instruction

bit 6-4

IRCF<2:0>: Internal RC Oscillator Frequency Select bit

111 = HFINTOSC – (16 MHz)

110 = HFINTOSC/2 – (8 MHz)

101 = HFINTOSC/4 – (4 MHz)

100 = HFINTOSC/8 – (2 MHz)

011 = HFINTOSC/16 – (1 MHz)⁽³⁾

If INTSRC = 0 and MFIOSEL = 0:

010 = HFINTOSC/32 – (500 kHz)

001 = HFINTOSC/64 – (250 kHz)

000 = LFINTOSC – (31.25 kHz)

If INTSRC = 1 and MFIOSEL = 0:

010 = HFINTOSC/32 – (500 kHz)

001 = HFINTOSC/64 – (250 kHz)

000 = HFINTOSC/512 – (31.25 kHz)

If INTSRC = 0 and MFIOSEL = 1:

010 = MFINTOSC – (500 kHz)

001 = MFINTOSC/2 – (250 kHz)

000 = LFINTOSC – (31.25 kHz)

對照Datasheet資料

HEX 30

DEC 48

OCT 60

BIN 0011 0000

11 0000

震盪器及Reset的設定 (OSCTUNE及RCON)

▼ Register: OSCTUNE 0x0

INTSRC	disabled	▼
PLLEN	disabled	▼
TUN	0x0	

▼ RESET

▼ Register: RCON 0x50

BOR	occurred	▼
IPEN	disabled	▼
PD	POR or CLRWDT	▼
POR	occurred	▼
RI	did not occur	▼
SBOREN	enabled	▼
TO	POR or CLRWDT	▼

震盪器的設定 (CONFIG1H)

System Module

⚙ Easy Setup 📄 Registers ⚠ Notifications : 0

▼ System Module

▼ Register: CONFIG1H 0x28

FCMEN	Fail-Safe Clock Monitor disabled	▼
FOSC	Internal oscillator block	▼
IESO	Oscillator Switchover mode disabled	▼
PLLCFG	Oscillator used directly	▼
PRICKEN	Primary clock is always enabled	▼

其它 Config. Bits 的設定 (一)

- **CONFIG2L**

- ◆ BOREN SBORDIS, BORV 285, PWRTEN ON

- **CONFIG2H**

- ◆ WDTEN OFF (除錯時的必要選項)

- **CONFIG3H**

- ◆ CCP2MX PORTC1, PBADEN OFF,
MCLRE EXTMCLR (除錯模式下需使用 MCLR 腳)

- **CONFIG4L**

- ◆ LVP OFF, STVREN ON, XINTST OFF
 - 如使用 **Lite** 版的 **XC8** 要將 **XINTST** 設成 **OFF**

其它 Config. Bits 的設定 (一)

▼ Register: CONFIG2H	0x3C	
WDTEN	Watch dog timer is always disabled. SWDTEN has no effect.	▼
WDTPS	1:32768	▼

▼ Register: CONFIG2L	0x7	
BOREN	Brown-out Reset enabled in hardware only (SBOREN is disabled)	▼
BORV	VBOR set to 2.85 V nominal	▼
PWRTEN	Power up timer disabled	▼

其它 Config. Bits 的設定 (二)

▼ Register: CONFIG3H 0xBF

CCP2MX	CCP2 input/output is multiplexed with RC1	▼
CCP3MX	P3A/CCP3 input/output is multiplexed with RB5	▼
HFOFST	HFINTOSC output and ready status are not delayed by the oscillator stable status	▼
MCLRE	MCLR pin enabled, RE3 input pin disabled	▼
P2BMX	P2B is on RD2	▼
PBADEN	PORTB<5:0> pins are configured as analog input channels on Reset	▼
T3CMX	T3CKI is on RC0	▼

▼ Register: CONFIG4L 0x81

DEBUG	Disabled	▼
LVP	Single-Supply ICSP disabled	▼
STVREN	Stack full/underflow will cause Reset	▼
XINST	Instruction set extension and Indexed Addressing mode disabled (Legacy mode)	▼

其它 Config. Bits 的設定 (三)

- **CONFIG5L (設成OFF)**

- ◆ 區塊程式碼保護設定，除錯階段關閉所有的保護設定

- **CONFIG5H (設成OFF)**

- ◆ Boot 區塊保護，EEPROM 資料保護

- **CONFIG6L (設成OFF)**

- ◆ 程式區塊禁止程式執行的寫入

- **CONFIG6H (設成OFF)**

- ◆ Config. Reg, EEPROM, Boot 區塊寫入的保護

除錯階段請關閉所有的保護設定

其它 Config. Bits 的設定 (四)

▼ Register: CONFIG5H 0xC0	
CPB	Boot block (000000-0007FFh) not code-protected ▼
CPD	Data EEPROM not code-protected ▼
▼ Register: CONFIG5L 0xF	
CP0	Block 0 (000800-001FFFh) not code-protected ▼
CP1	Block 1 (002000-003FFFh) not code-protected ▼
CP2	Block 2 (004000-005FFFh) not code-protected ▼
CP3	Block 3 (006000-007FFFh) not code-protected ▼
▼ Register: CONFIG6H 0xE0	
WRTB	Boot Block (000000-0007FFh) not write-protected ▼
WRTC	Configuration registers (300000-3000FFh) not write-protected ▼
WRTD	Data EEPROM not write-protected ▼
▼ Register: CONFIG6L 0xF	
WRT0	Block 0 (000800-001FFFh) not write-protected ▼
WRT1	Block 1 (002000-003FFFh) not write-protected ▼
WRT2	Block 2 (004000-005FFFh) not write-protected ▼
WRT3	Block 3 (006000-007FFFh) not write-protected ▼

其它 Config. Bits 的設定 (五)

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

除錯階段如請關閉所有的保護設定

其它 Config. Bits 的設定 (六)

▼ Register: CONFIG7H 0x40

EBTRB Boot Block (000000-0007FFh) not protected from table reads executed in other blocks ▼

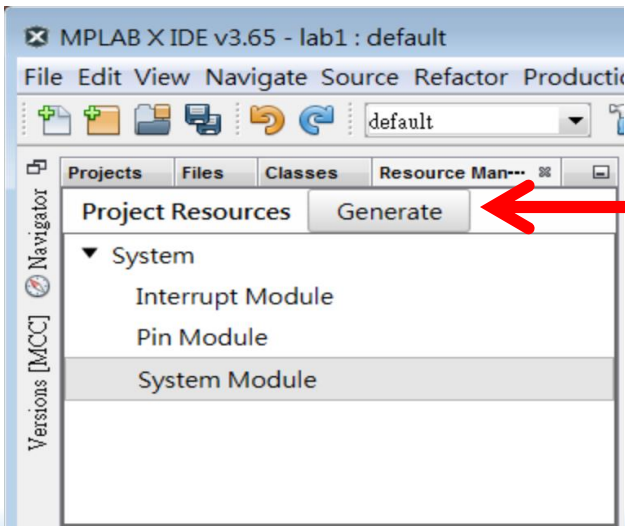
▼ Register: CONFIG7L 0xF

EBTR0 Block 0 (000800-001FFFh) not protected from table reads executed in other blocks ▼

EBTR1 Block 1 (002000-003FFFh) not protected from table reads executed in other blocks ▼

EBTR2 Block 2 (004000-005FFFh) not protected from table reads executed in other blocks ▼

EBTR3 Block 3 (006000-007FFFh) not protected from table reads executed in other blocks ▼



Press Generate Button

MCC 產生的 Config 設定

// CONFIG1H

```
#pragma config FOSC = INTIO67 // Oscillator Selection bits->Internal oscillator block
#pragma config PLLCFG = OFF // 4X PLL Enable->Oscillator used directly
#pragma config PRCLKEN = ON // Primary clock enable bit->Primary clock is always enabled
#pragma config FCMEN = OFF // Fail-Safe Clock Monitor Enable bit->Fail-Safe Clock Monitor disabled
#pragma config IESO = OFF // Internal/External Oscillator Switchover bit->Oscillator Switchover mode dis
```

// CONFIG2L

```
#pragma config PWRTEN = OFF // Power-up Timer Enable bit->Power up timer disabled
#pragma config BOREN = SBORDIS // Brown-out Reset Enable bits->Brown-out Reset enabled in hardware
#pragma config BORV = 285 // Brown Out Reset Voltage bits->VBOR set to 2.85 V nominal
```

// CONFIG2H

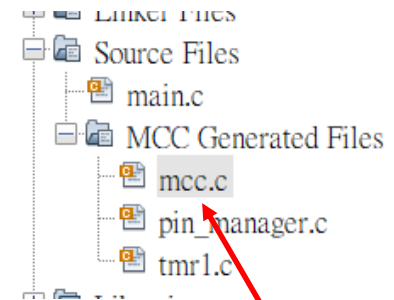
```
#pragma config WDTCN = OFF // Watchdog Timer Enable bits->Watch dog timer is always disabled. SWDTEN has no effect.
#pragma config WDTCS = 32768 // Watchdog Timer Postscale Select bits->1:32768
```

// CONFIG3H

```
#pragma config CCP2MX = PORTC1 // CCP2 MUX bit->CCP2 input/output is multiplexed with RC1
#pragma config PBAEN = ON // PORTB A/D Enable bit->PORTB<5:0> pins are configured as analog input channels on Reset
#pragma config CCP3MX = PORTB5 // P3A/CCP3 Mux bit->P3A/CCP3 input/output is multiplexed with RB5
#pragma config HFOFST = ON // HFINTOSC Fast Start-up->HFINTOSC output and ready status are not delayed by the oscillator stable status
#pragma config T3CMX = PORTC0 // Timer3 Clock input mux bit->T3CKI is on RC0
#pragma config P2BMX = PORTD2 // ECCP2 B output mux bit->P2B is on RD2
#pragma config MCLRE = EXTMCLR // MCLR Pin Enable bit->MCLR pin enabled, RE3 input pin disabled
```

// CONFIG4L

```
#pragma config STVREN = ON // Stack Full/Underflow Reset Enable bit->Stack full/underflow will cause Reset
#pragma config LVP = OFF // Single-Supply ICSP Enable bit->Single-Supply ICSP disabled
#pragma config XINST = OFF // Extended Instruction Set Enable bit->Instruction set extension and Indexed Addressing mode disabled (Legacy m
#pragma config DEBUG = OFF // Background Debug->Disabled
```



存放在mcc.c中

MCC 產生的 Config 設定

// CONFIG5L

```
#pragma config CP0 = OFF // Code Protection Block 0->Block 0 (000800-001FFFh) not code-protected
#pragma config CP1 = OFF // Code Protection Block 1->Block 1 (002000-003FFFh) not code-protected
#pragma config CP2 = OFF // Code Protection Block 2->Block 2 (004000-005FFFh) not code-protected
#pragma config CP3 = OFF // Code Protection Block 3->Block 3 (006000-007FFFh) not code-protected
```

// CONFIG5H

```
#pragma config CPB = OFF // Boot Block Code Protection bit->Boot block (000000-0007FFFh) not code-protected
#pragma config CPD = OFF // Data EEPROM Code Protection bit->Data EEPROM not code-protected
```

// CONFIG6L

```
#pragma config WRT0 = OFF // Write Protection Block 0->Block 0 (000800-001FFFh) not write-protected
#pragma config WRT1 = OFF // Write Protection Block 1->Block 1 (002000-003FFFh) not write-protected
#pragma config WRT2 = OFF // Write Protection Block 2->Block 2 (004000-005FFFh) not write-protected
#pragma config WRT3 = OFF // Write Protection Block 3->Block 3 (006000-007FFFh) not write-protected
```

// CONFIG6H

```
#pragma config WRTC = OFF // Configuration Register Write Protection bit->Configuration registers (300000-3000FFFh) not write-protected
#pragma config WRTB = OFF // Boot Block Write Protection bit->Boot Block (000000-0007FFFh) not write-protected
#pragma config WRTD = OFF // Data EEPROM Write Protection bit->Data EEPROM not write-protected
```

// CONFIG7L

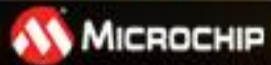
```
#pragma config EBTR0 = OFF // Table Read Protection Block 0->Block 0 (000800-001FFFh) not protected from table reads executed in other blocks
#pragma config EBTR1 = OFF // Table Read Protection Block 1->Block 1 (002000-003FFFh) not protected from table reads executed in other blocks
#pragma config EBTR2 = OFF // Table Read Protection Block 2->Block 2 (004000-005FFFh) not protected from table reads executed in other blocks
#pragma config EBTR3 = OFF // Table Read Protection Block 3->Block 3 (006000-007FFFh) not protected from table reads executed in other blocks
```

// CONFIG7H

```
#pragma config EBTRB = OFF // Boot Block Table Read Protection bit->Boot Block (000000-0007FFFh) not protected from table reads executed in other blocks
```

Reduce your development time • Reuse your code • Scale up or down

ONE DEVELOPMENT ENVIRONMENT



ONE PIC[®] MCU PLATFORM



GPIO 腳位 的設定

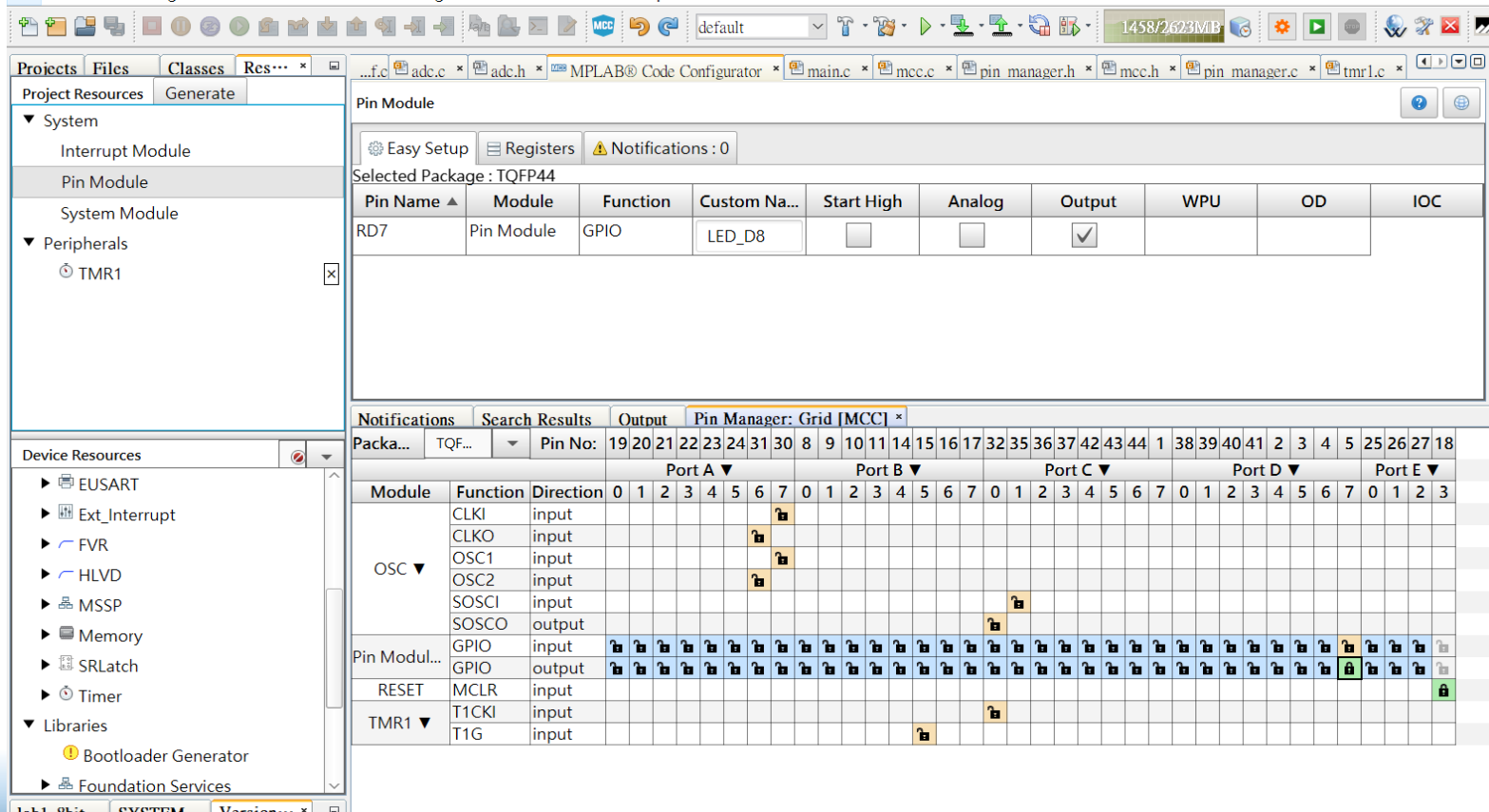
GPIO 腳位設定

● 在 Project Resources 視窗下

◆ 點選 “Pin Module”

✕ MPLAB X IDE v3.55 - lab1_8bit : default

File Edit View Navigate Source Refactor Run Debug Team Tools Window Help



The screenshot shows the MPLAB X IDE interface. The 'Project Resources' window is open, and the 'Pin Module' is selected under the 'System' category. The 'Pin Module' configuration window is displayed, showing the 'Easy Setup' tab. The 'Selected Package' is 'TQFP44'. The 'Pin Name' is 'RD7', the 'Module' is 'Pin Module', and the 'Function' is 'GPIO'. The 'Custom Name' is 'LED_D8'. The 'Start High' checkbox is unchecked, the 'Analog' checkbox is unchecked, and the 'Output' checkbox is checked. The 'WPU', 'OD', and 'IOC' checkboxes are also unchecked.

Below the configuration window, the 'Pin Manager: Grid [MCC]' is visible. It shows a table of pins and their functions. The table has columns for 'Pin No.' (19-28), 'Port A', 'Port B', 'Port C', 'Port D', and 'Port E'. The rows show the 'Module', 'Function', and 'Direction' for each pin. The 'Pin Module' is highlighted in blue, and the 'TMR1' module is highlighted in green.

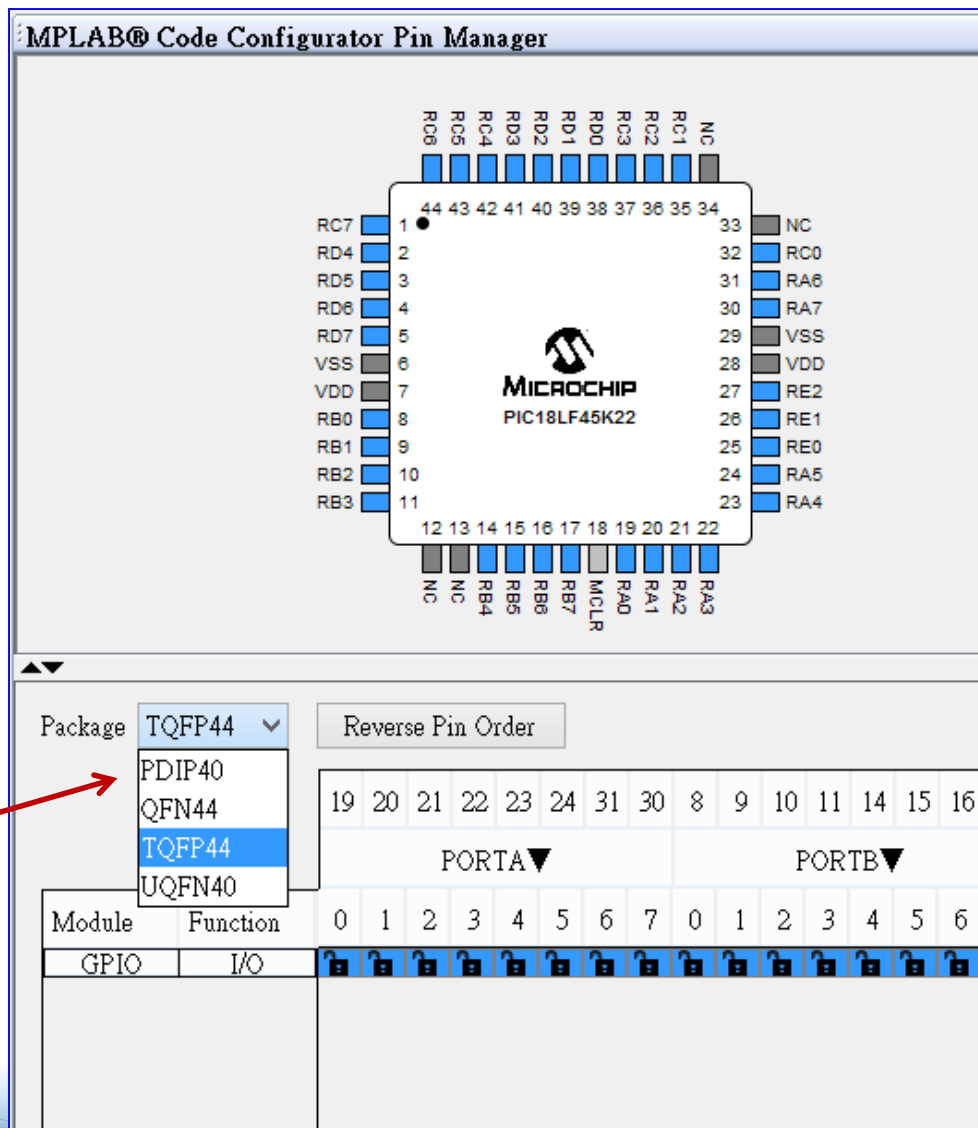
Pin No.	Port A	Port B	Port C	Port D	Port E
19	0	1	2	3	4
20	5	6	7	8	9
21	10	11	12	13	14
22	15	16	17	18	19
23	20	21	22	23	24
24	25	26	27	28	29
25	30	31	32	33	34
26	35	36	37	38	39
27	40	41	42	43	44
28	45	46	47	48	49

GPIO 腳位設定

● 元件包裝的變更

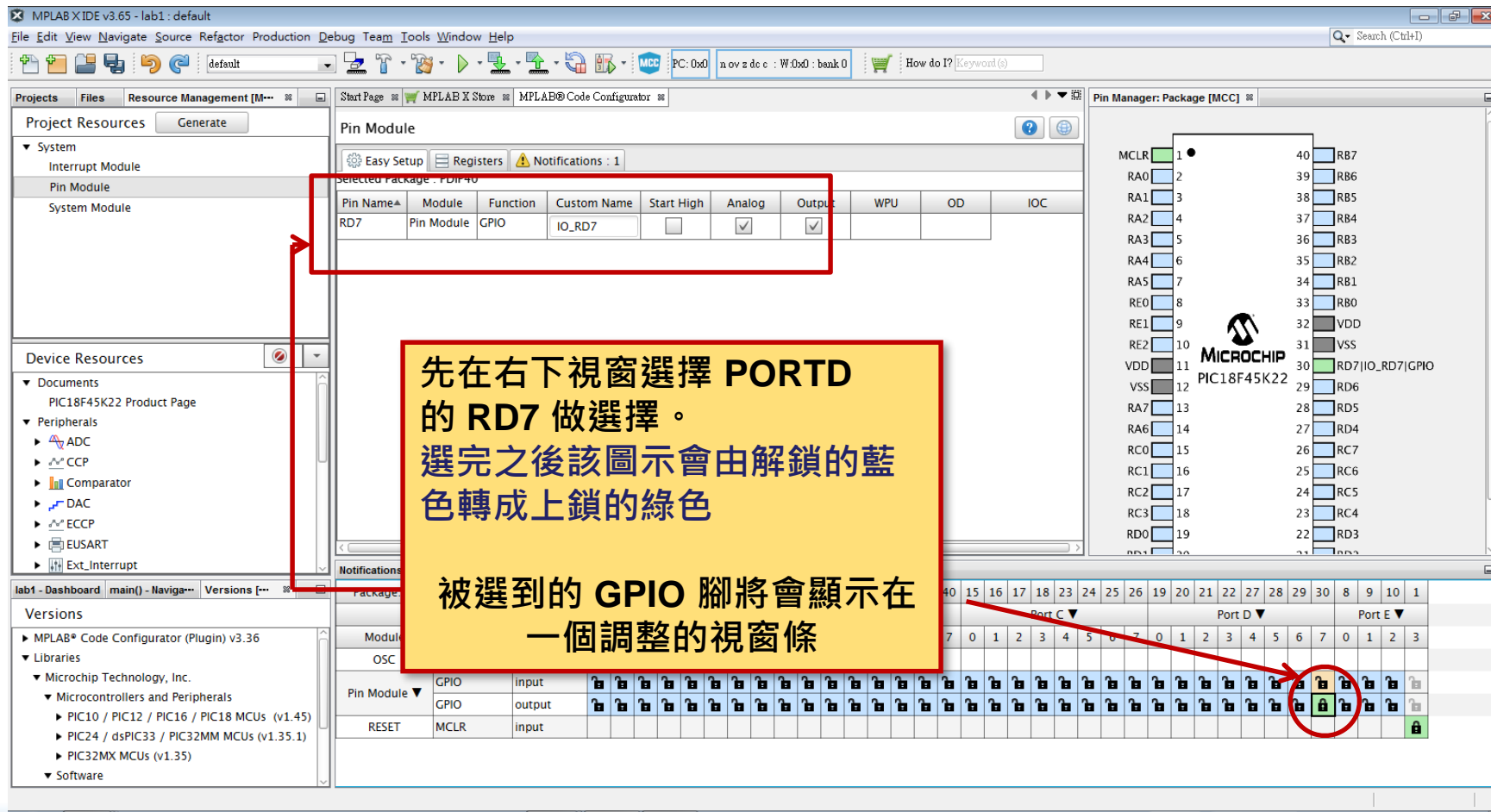
- ◆ PDIP40
- ◆ QFN44
- ◆ TQFP44
- ◆ UQFN40

選擇所需的包裝



GPIO 腳位設定

- 在 MCC Pin Manager 視窗下選擇 **RD7**



Pin Module

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RD7	Pin Module	GPIO	IO_RD7	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			

Pin Manager: Package [MCC]

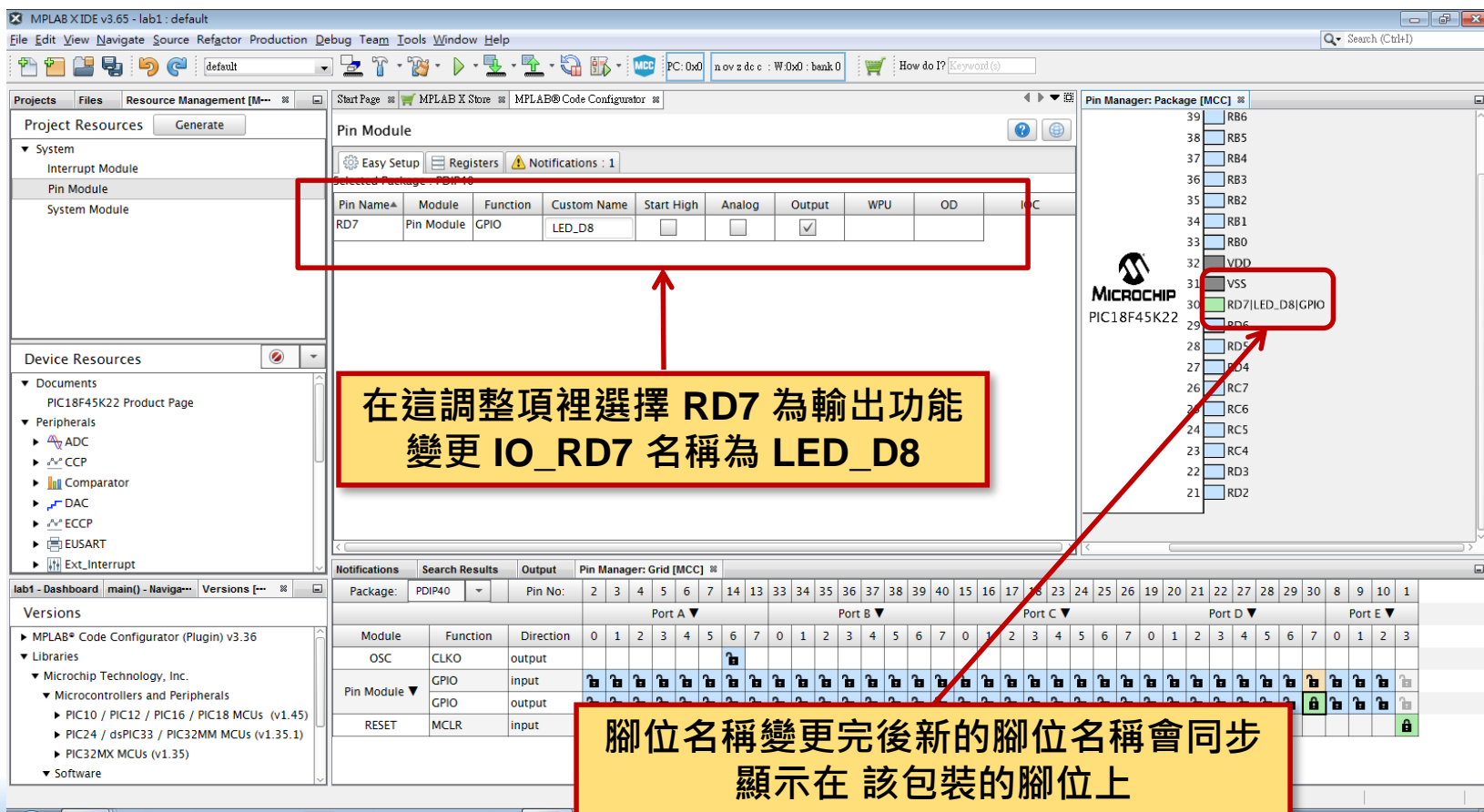
MCLR 1 40 RB7
RA0 2 39 RB6
RA1 3 38 RB5
RA2 4 37 RB4
RA3 5 36 RB3
RA4 6 35 RB2
RA5 7 34 RB1
RE0 8 33 RB0
RE1 9 32 VDD
RE2 10 31 VSS
VDD 11 30 RD7|IO_RD7|GPIO
VSS 12 29 RD6
RA7 13 28 RD5
RA6 14 27 RD4
RC0 15 26 RC7
RC1 16 25 RC6
RC2 17 24 RC5
RC3 18 23 RC4
RD0 19 22 RD3
RD1 20 21 RD2

先在下視窗選擇 PORTD 的 RD7 做選擇。
選完之後該圖示會由解鎖的藍色轉成上鎖的綠色

被選到的 GPIO 腳將會顯示在一個調整的視窗條

GPIO 腳位設定

- 設定 RD7 為輸出腳功能，並更改名稱為 **LED_D8** (程式使用)



Pin Module

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IO/C
RD7	Pin Module	GPIO	LED_D8	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			

在這調整項裡選擇 RD7 為輸出功能
變更 IO_RD7 名稱為 LED_D8

Pin Manager: Package [MCC]

Pin No.	Pin Name
39	RB6
38	RB5
37	RB4
36	RB3
35	RB2
34	RB1
33	RB0
32	VDD
31	VSS
30	RD7(LED_D8)GPIO
29	RD6
28	RD5
27	RD4
26	RC7
25	RC6
24	RC5
23	RC4
22	RD3
21	RD2

腳位名稱變更完後新的腳位名稱會同步
顯示在 該包裝的腳位上



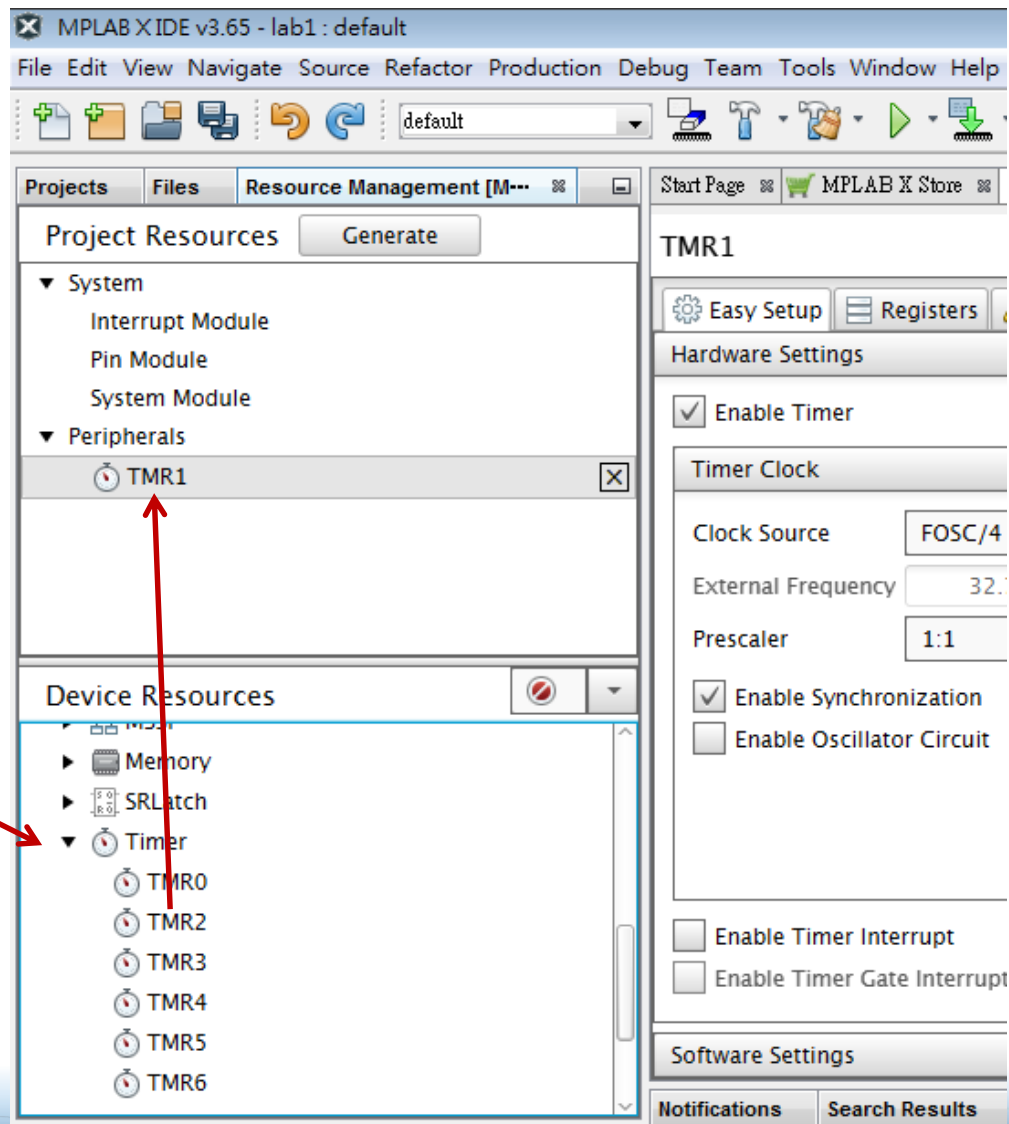
Timer 的設定

Timer1 設定

- 選擇 TMR1

雙點選 “**Timer**” 後顯示
所有的 Timer 模組

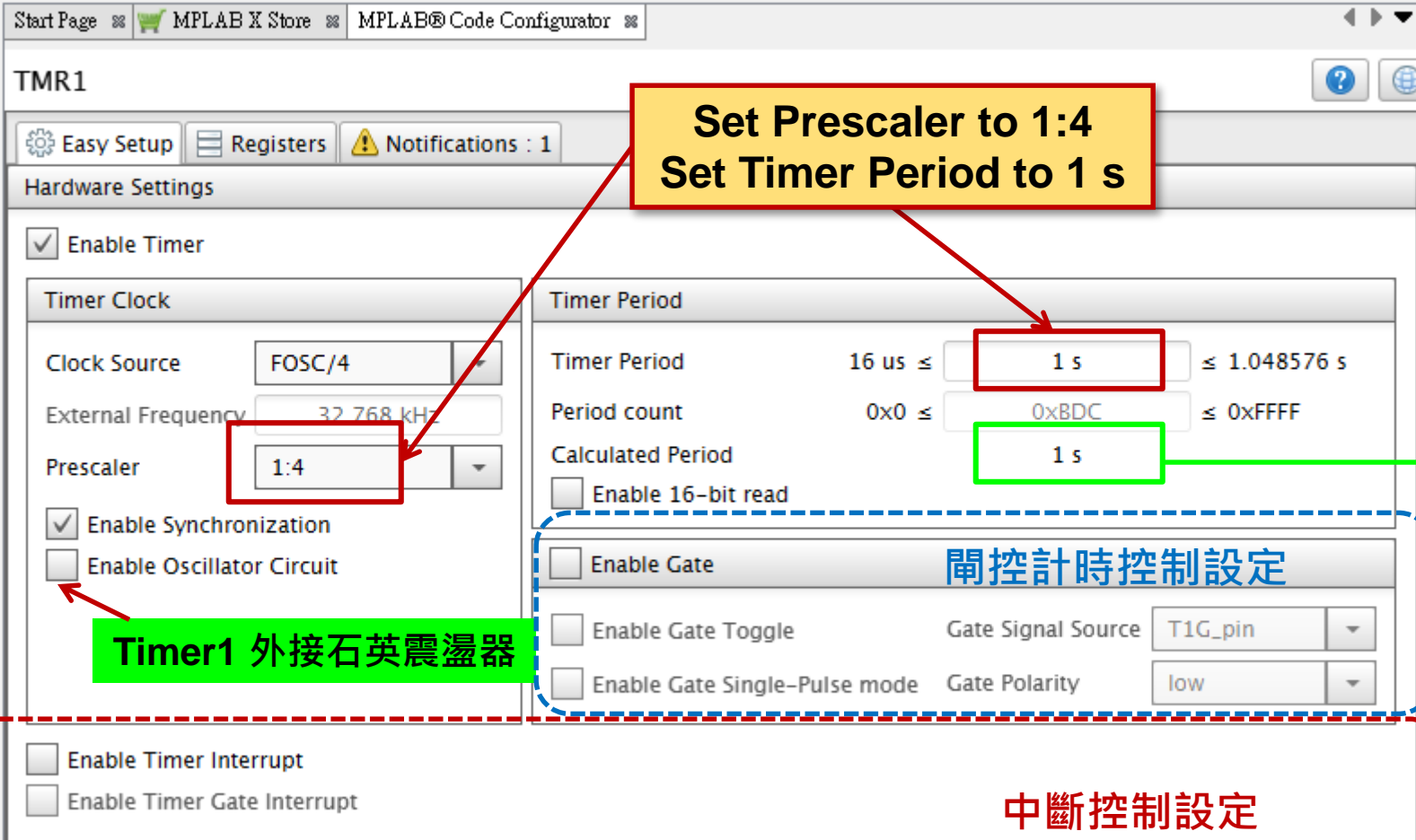
點選 **TMR1::TMR** 的模組
後將會移到 “Project
Resources” 的視窗





Timer1 設定

● Timer1 設定說明



The screenshot shows the TMR1 configuration window in MPLAB X IDE. The window is titled "TMR1" and has tabs for "Easy Setup", "Registers", and "Notifications : 1". The "Easy Setup" tab is selected, showing "Hardware Settings".

Annotations:

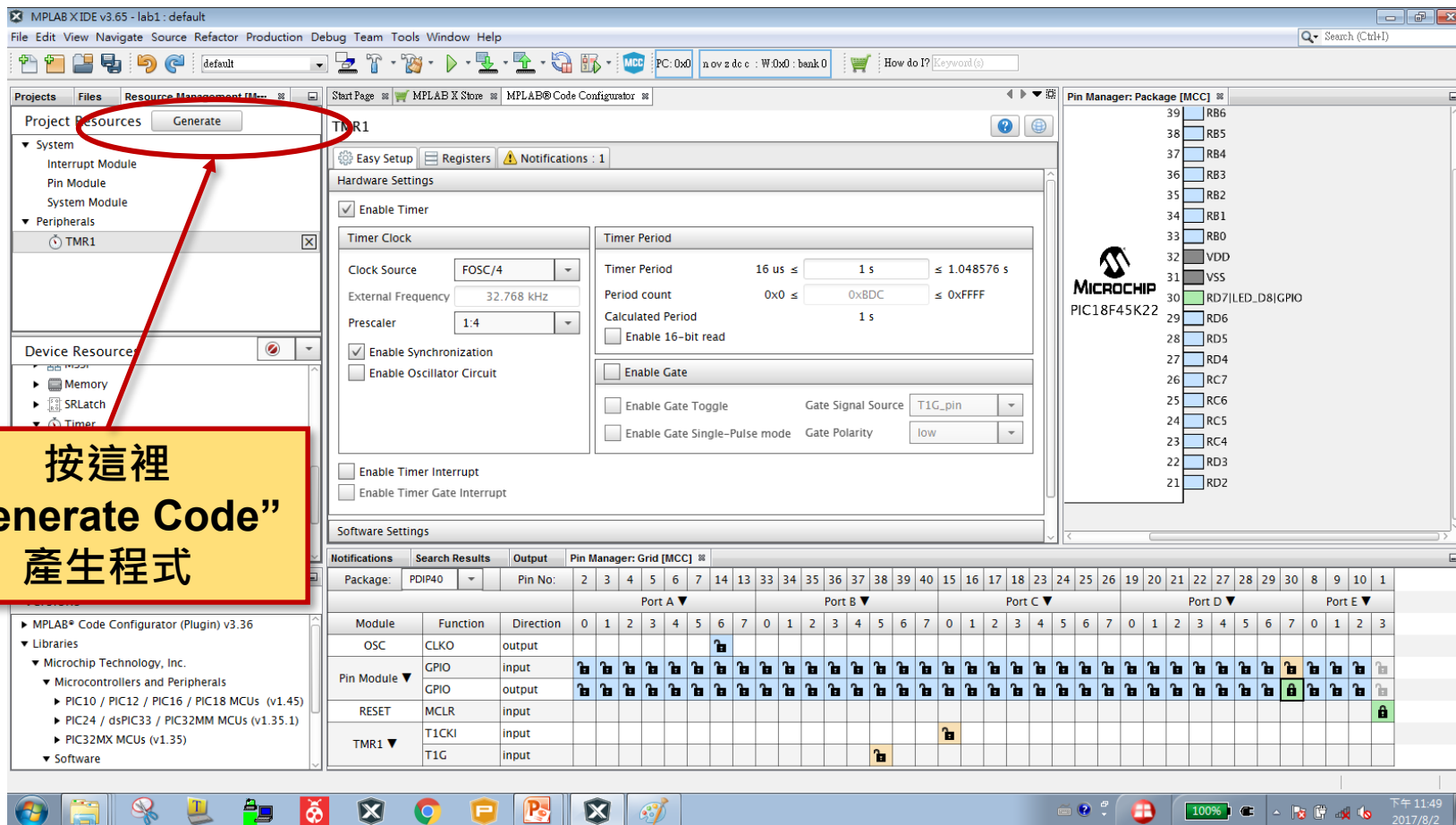
- Set Prescaler to 1:4 Set Timer Period to 1 s** (Yellow box): Points to the "Prescaler" dropdown (set to 1:4) and the "Timer Period" input (set to 1 s).
- 確認最終系統設定出來的值** (Vertical text on the right): Points to the "Timer Period" input.
- Timer1 外接石英震盪器** (Green box): Points to the "Enable Synchronization" checkbox, which is checked.
- 閘控計時控制設定** (Blue dashed box): Encloses the "Enable Gate", "Enable Gate Toggle", and "Enable Gate Single-Pulse mode" checkboxes, all of which are unchecked.
- 中斷控制設定** (Red dashed box): Encloses the "Enable Timer Interrupt" and "Enable Timer Gate Interrupt" checkboxes, both of which are unchecked.

Configuration Details:

- Timer Clock:**
 - Clock Source: FOSC/4
 - External Frequency: 32.768 kHz
 - Prescaler: 1:4
 - ☒ Enable Synchronization
 - ☐ Enable Oscillator Circuit
- Timer Period:**
 - Timer Period: 1 s
 - Period count: 0x0 ≤ 0xBDC ≤ 0xFFFF
 - Calculated Period: 1 s
 - ☐ Enable 16-bit read
- Gate Control:**
 - ☐ Enable Gate
 - ☐ Enable Gate Toggle
 - ☐ Enable Gate Single-Pulse mode
 - Gate Signal Source: T1G_pin
 - Gate Polarity: low
- Interrupt Control:**
 - ☐ Enable Timer Interrupt
 - ☐ Enable Timer Gate Interrupt

程式產生器

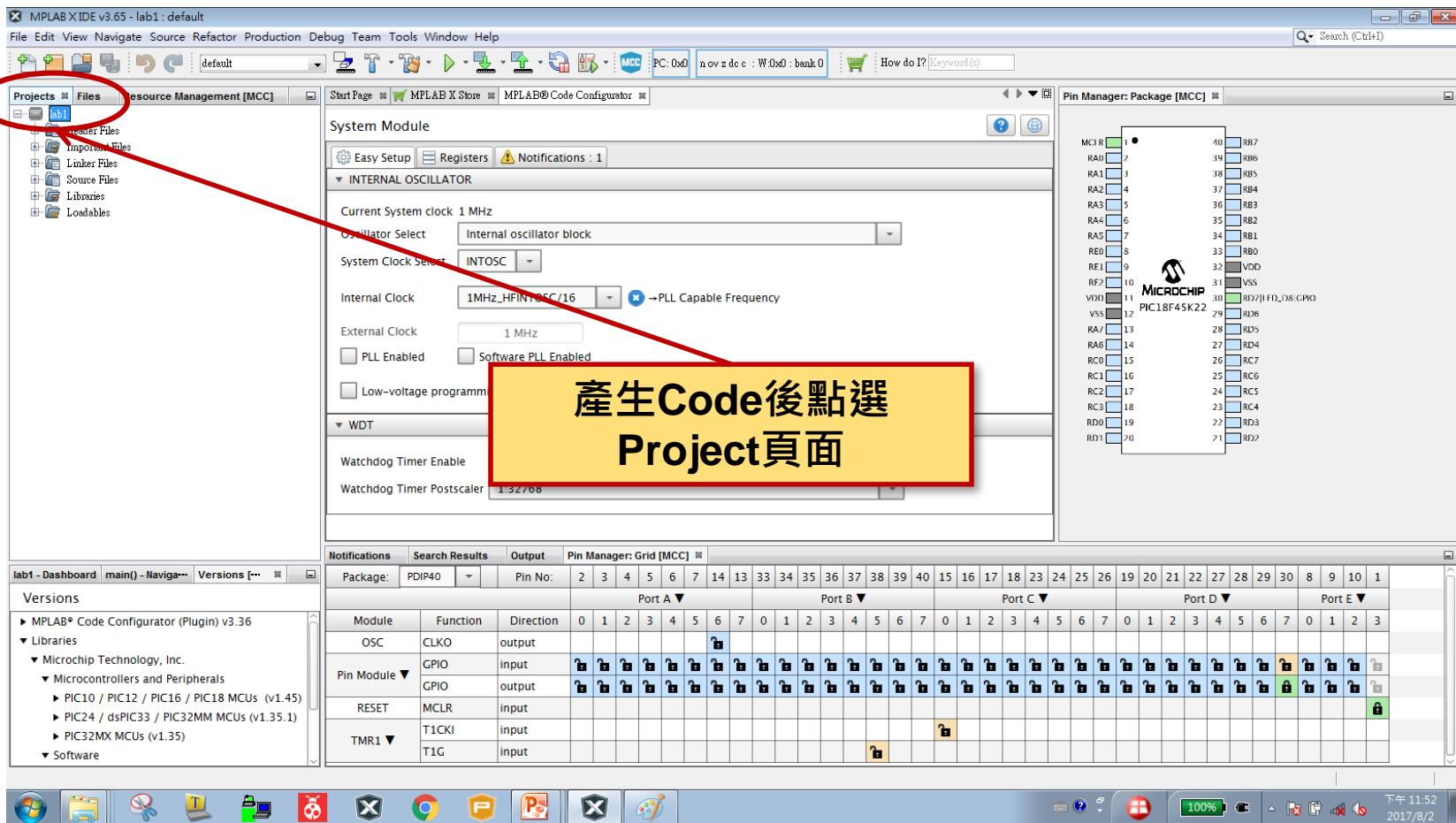
- 開始產生 C 程式並加到專案裡



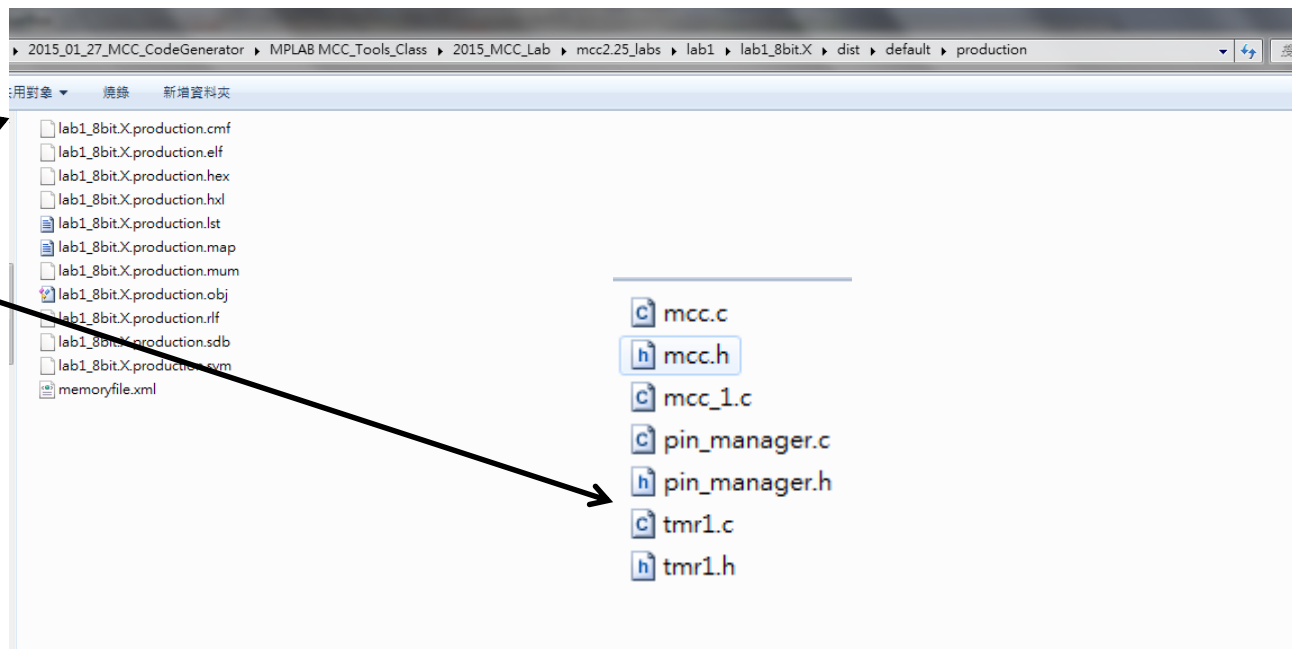
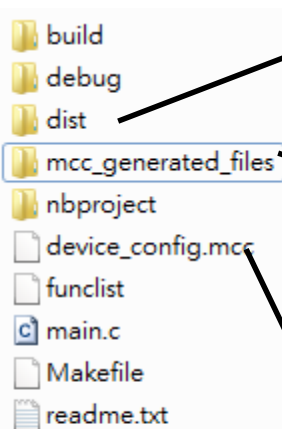
按這裡
“Generate Code”
產生程式

PIC18 Lab 1

- 產生Code後點選Project頁面

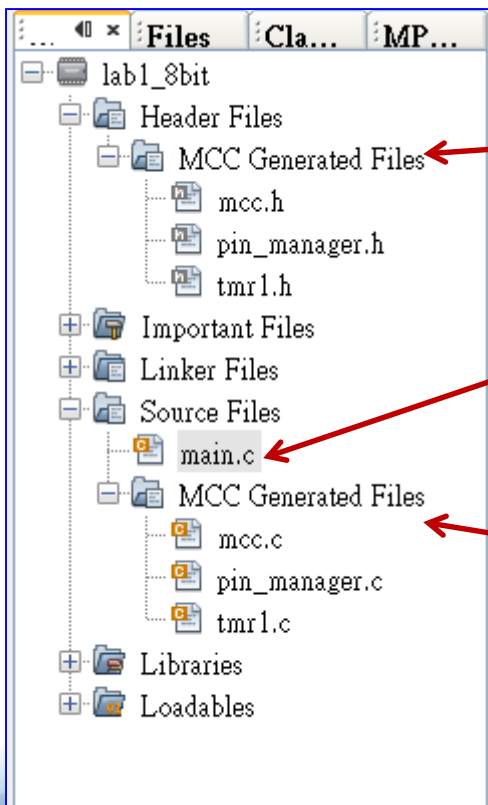


專案目錄下檔案的分析



PIC18 Lab 1

- 專案視窗下程式產生器所產生的檔案
 - ◆ 程式產生器所產生的檔案在 **MCC Generated Files** 的目錄下



展開標頭檔的檔案

mcc.h
pin_manager.h
tmr1.h

你的原始程式區
main.c

展開原始程式 (C code) 檔案
裡的周邊程式設定檔

mcc.c
pin_manager.c
tmr1.c

如何使用 MCC 所產生的標頭檔 **mcc.h**

- **mcc.h** – 為所有周邊標頭檔的總管，將其加在 **main.c** 其它 MCC 所產生的 **h** 檔將一併函入

```
#ifndef          MCC_H
#define          MCC_H
#include          <xc.h>           // 加入 XC8 的主標頭檔
#include          "pin_manager.h"  // 加入腳位設定的標頭檔
#include          "tmr1.h"         // 加入 Timer1 的設定標頭檔

#define _XTAL_FREQ 1000000        // 定義工作頻率 Fosc
void SYSTEM_Initialize(void);      // 函數原型宣告
void OSCILLATOR_Initialize(void);
```

標頭檔的說明

pin_manager.h

- **pin_manager.h** – 為所有的 I/O 腳位定義動作的巨集，I/O 函數原型宣告

```
void PIN_MANAGER_Initialize (void); // 原型宣告
```

```
//get/set LED_D8 aliases
```

```
#define LED_D8_TRIS      TRISD7
```

```
#define LED_D8_LAT       LATD7
```

```
#define LED_D8_PORT      PORTDbits.RD7
```

```
#define LED_D8_ANS       ANSD7
```

```
#define LED_D8_SetHigh()  do { LATD7 = 1; } while(0)
```

```
#define LED_D8_SetLow()   do { LATD7 = 0; } while(0)
```

```
#define LED_D8_Toggle()   do { LATD7 = ~LATD7; } while(0)
```

```
#define LED_D8_GetValue() PORTDbits.RD7
```

```
#define LED_D8_SetDigitalInput()  do { TRISD7 = 1; } while(0)
```

```
#define LED_D8_SetDigitalOutput() do { TRISD7 = 0; } while(0)
```

```
#define LED_D8_SetAnalogMode()    do { ANSD7 = 1; } while(0)
```

```
#define LED_D8_SetDigitalMode()   do { ANSD7 = 0; } while(0)
```

標頭檔的說明

tmr1.h

- **tmr1.h** – 為 **Timer1** 函數原型宣告

```
#include <stdbool.h>
```

```
#include <stdint.h>
```

```
void TMR1_Initialize(void);
```

```
void TMR1_StartTimer(void);
```

```
void TMR1_StopTimer(void);
```

```
uint16_t TMR1_ReadTimer(void);
```

```
void TMR1_WriteTimer(uint16_t timerVal);
```

```
void TMR1_Reload(void);
```

```
void TMR1_StartSinglePulseAcquisition(void);
```

```
uint8_t TMR1_CheckGateValueStatus(void);
```

```
bool TMR1_HasOverflowOccured(void);
```

Timer1 使用基本範例

```
TMR1_Initialize();  
TMR1_WriteTimer(0x055);  
TMR1_StartTimer();
```

MCC 主程式說明

mcc.c

- **mcc.c** – 為 **MCC** 主要的程式

```
#pragma config IESO = OFF
#pragma config PLLCFG = OFF
#pragma config PRICKEN = OFF
#pragma config FOSC = INTIO67
#pragma config FCMEN = OFF
:
#include "mcc.h"
```

上方為產生的 **Config.**
Bits 的設定

```
void SYSTEM_Initialize(void)
{
    OSCILLATOR_Initialize();
    PIN_MANAGER_Initialize();
    TMR1_Initialize();
}
```

MCC 所產生的各式周
邊函數，使用時只要
在 **main.c** 呼叫函數
SYSTEM_Initialize()
即可完成周邊設定



pin_manager.c & tmr1.c

- **pin_Manager.c - I/O 腳的設定函數**

PIN_MANAGER_Initialize()

- **tmr1.c – Tmer1 的函數**

void TMR1_Initialize(void)

void TMR1_StartTimer(void)

void TMR1_StopTimer(void)

uint16_t TMR1_ReadTimer(void)

void TMR1_WriteTimer(uint16_t timerVal)

void TMR1_Reload(void)

void TMR1_StartSinglePulseAcquisition(void)

uint8_t TMR1_CheckGateValueStatus(void)

bool TMR1_HasOverflowOccured(void)

加入 MCC 到 main.c

- 從前面幾張投影片的檔案說明
 - ◆ 標頭檔只要加入：**mcc.h**
 - ◆ 程式裡呼叫 **SYSTEM_Initialize()** 即可

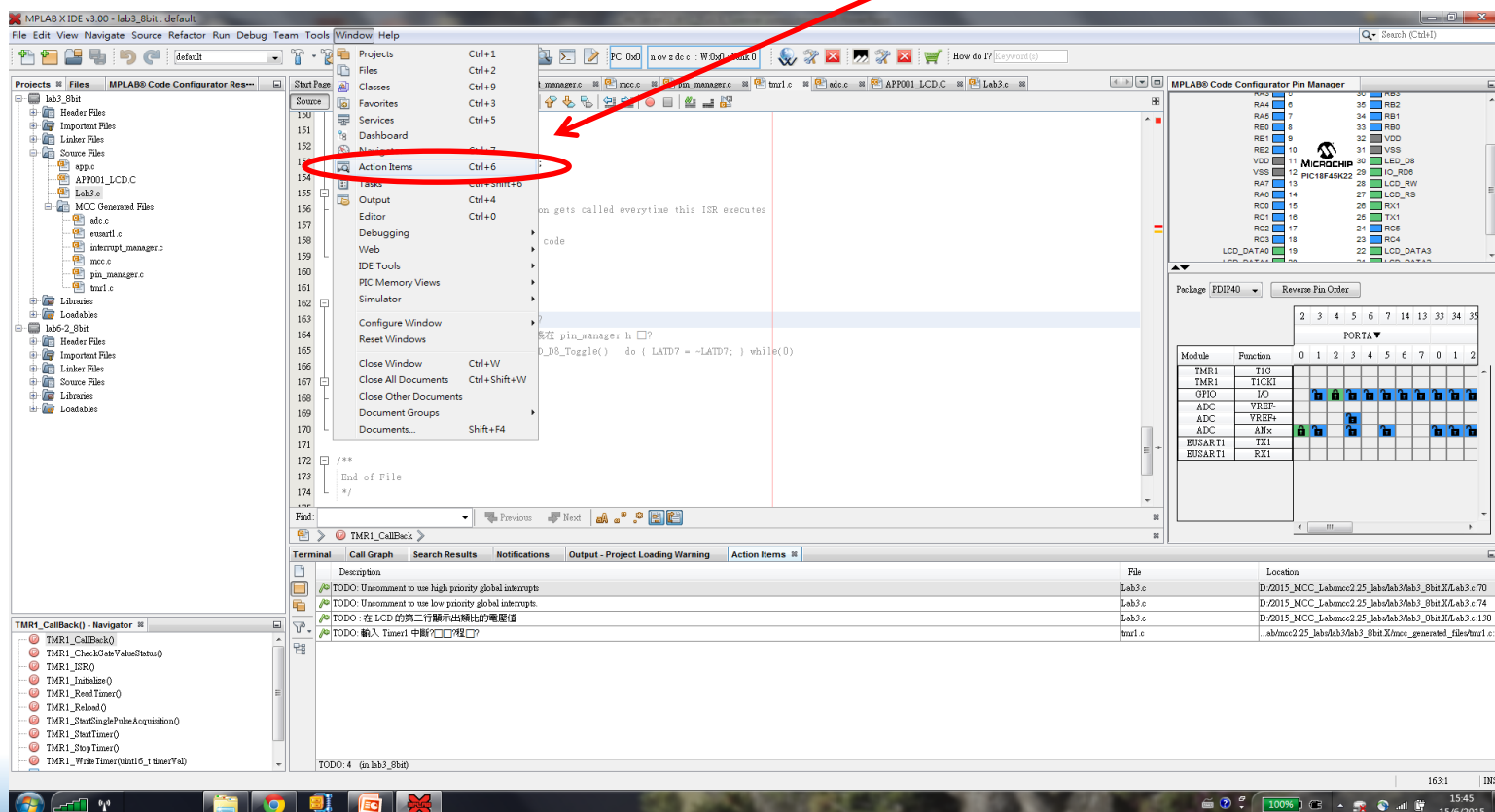
```
#include "mcc_generated_files/mcc.h"
```

```
// Main application
```

```
void main(void)  
{  
    // Initialize the device  
    SYSTEM_Initialize( );  
    :  
    :  
}
```

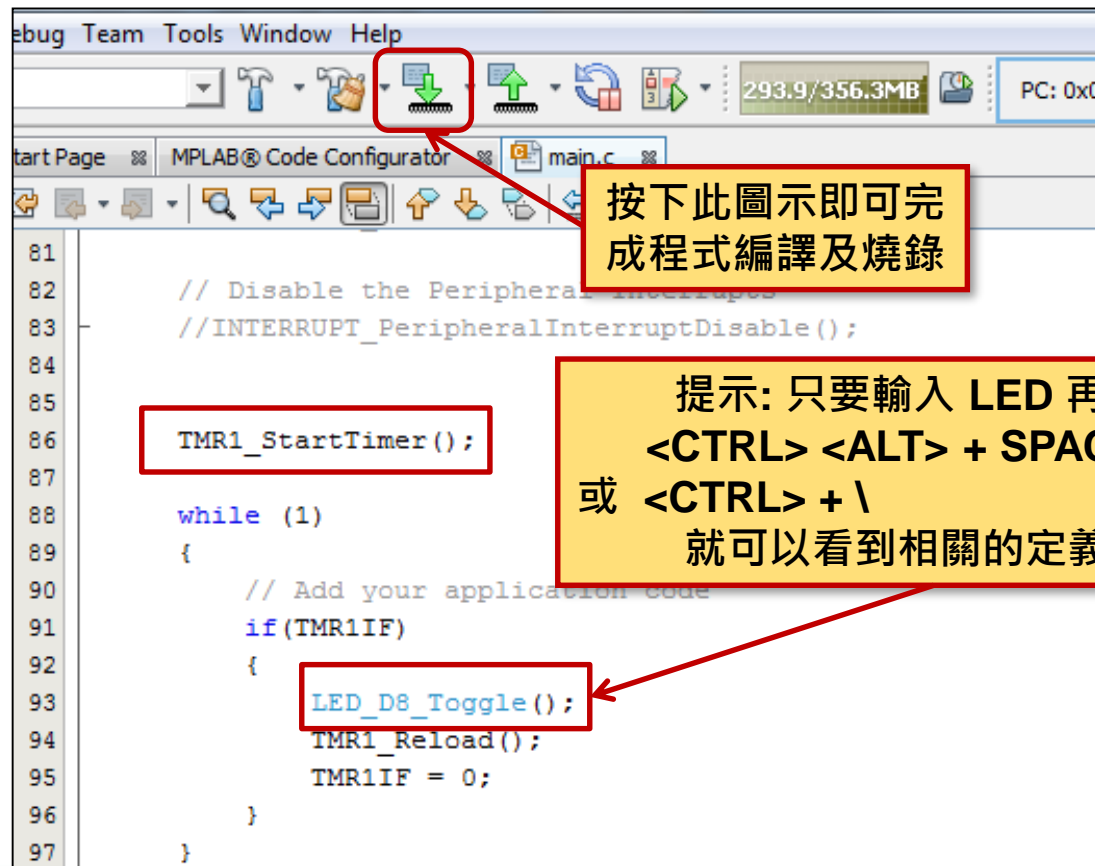
PIC18 Lab 1 動手寫程式

- 開啟 MPLAB® X Tasks 視窗，有兩個 TODO 事項需要完成



PIC18 Lab 1

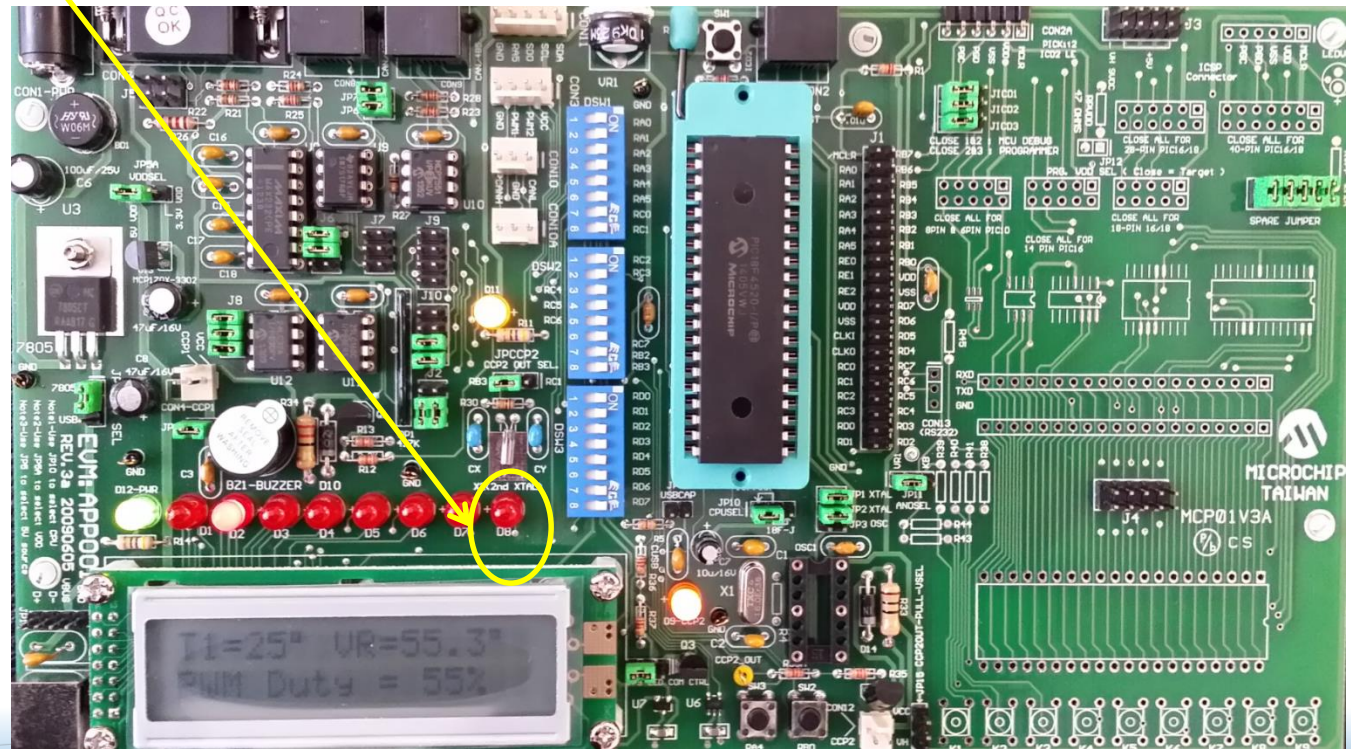
- 加入您的程式，按下燒錄圖示即可做**程式編譯及燒錄**。完成後應可以看到 **LED_D8** 間隔 1 秒閃爍。



PIC18 Lab 1

- 檢視 LED D8 的位置是否有每秒閃爍一次。

注意 LED D8 閃爍
(程式中取名為 LED_D8)



PIC18 Lab 1 結論

- 快速的設定 **GPIO** 來控制一個 **LED**
- 快速的設定 **Timer1** 做為 1 秒的計時器
- 快速地建立一個每秒閃爍的 **LED** 程式
- 以上是初體驗 **MCC**，再來就要更深入的來討論 **MCC** 的使用



MICROCHIP

Regional Training Centers

PIC18 Lab 2:

使用 Timer1 做一秒計時基準

LED 轉態一次、讀取可變變阻電壓值、將
電壓值經 UART 傳送終端機上顯示

PIC18 Lab 2 目標

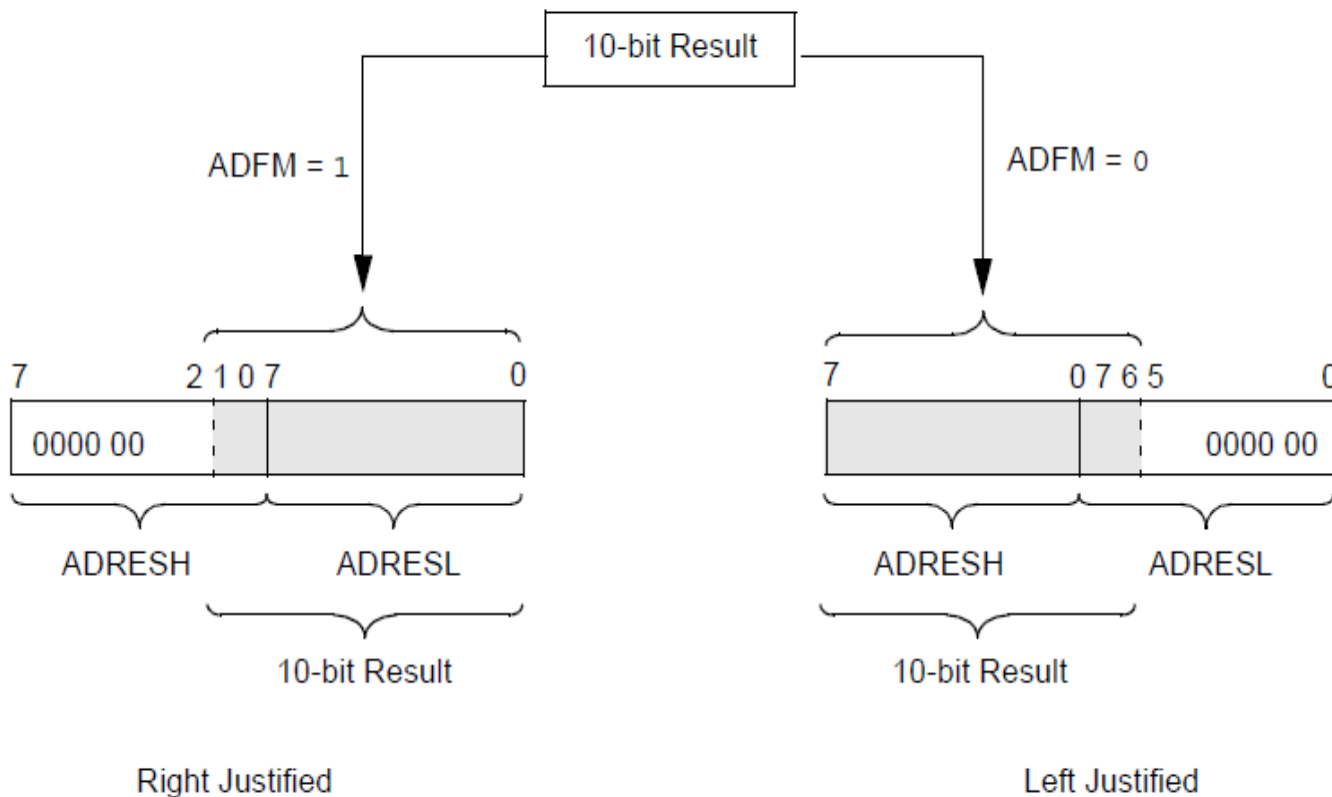
- 設定 **ADC** 模組的工作模式，並讀取可變電阻上的電壓值
- 設定 **UART** 模組的工作模式，使用 **Timer1** 做一秒計時基準 **LED** 轉態一次、讀取可變變阻電壓值、將電壓值經 **UART** 傳送終端機上顯示

PIC18 Lab 2 操作概述

- 確定關閉編輯器所開啟的檔案及關閉 **MPLAB® X IDE** 下的所有專案
- 開啟 **lab2_8bit.X**
- 設定 **ADC1** 模組，輸入腳為 **AN0**
 - ◆ 設定輸出格式為**向右靠齊** (right aligned)
 - ◆ 將 AN0 名稱變更為 **POT_CHANNEL**
- 設定 **EUSART1** 的通訊格式
 - ◆ **9600 Baud**, Parity None and 1 Stop Bit
 - ◆ 將 **STDIO** 輸出轉向到 **USART1**

PIC18 Lab 2-ADC

FIGURE 11-4: A/D RESULT JUSTIFICATION



向右靠齊 (right aligned)

PIC18 Lab 2-UART BAUD RATE

EXAMPLE 16-1: CALCULATING BAUD RATE ERROR

For a device with FOSC of 16 MHz, desired baud rate of 9600, Asynchronous mode, 8-bit BRG:

$$\text{Desired Baud Rate} = \frac{F_{osc}}{64([SPBRGHx:SPBRGx] + 1)}$$

Solving for SPBRGHx:SPBRGx:

$$X = \frac{\frac{F_{osc}}{\text{Desired Baud Rate}}}{64} - 1$$

$$= \frac{\frac{16000000}{9600}}{64} - 1$$

$$= [25.042] = 25$$

$$\text{Calculated Baud Rate} = \frac{16000000}{64(25 + 1)}$$

$$= 9615$$

$$\text{Error} = \frac{\text{Calc. Baud Rate} - \text{Desired Baud Rate}}{\text{Desired Baud Rate}}$$

$$= \frac{(9615 - 9600)}{9600} = 0.16\%$$

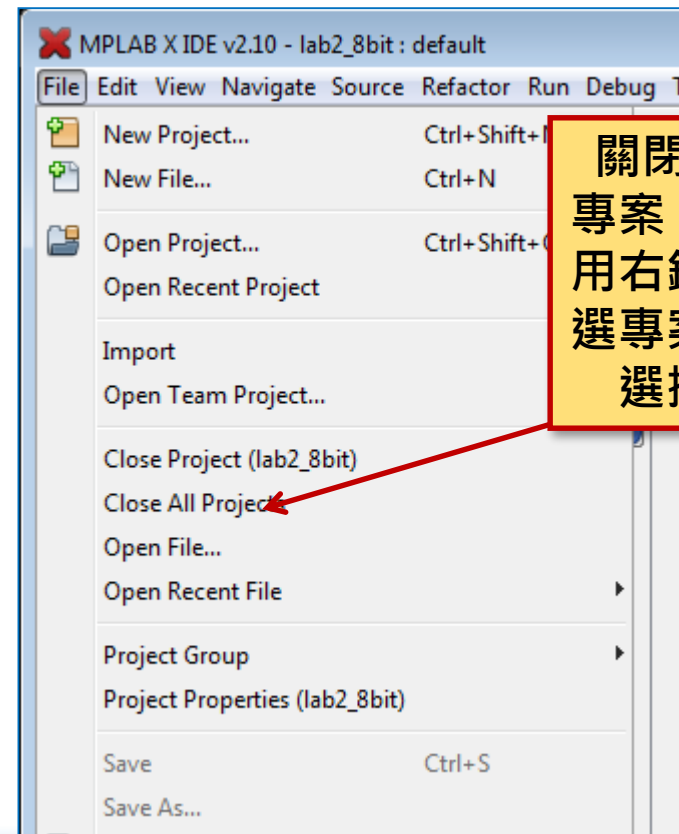
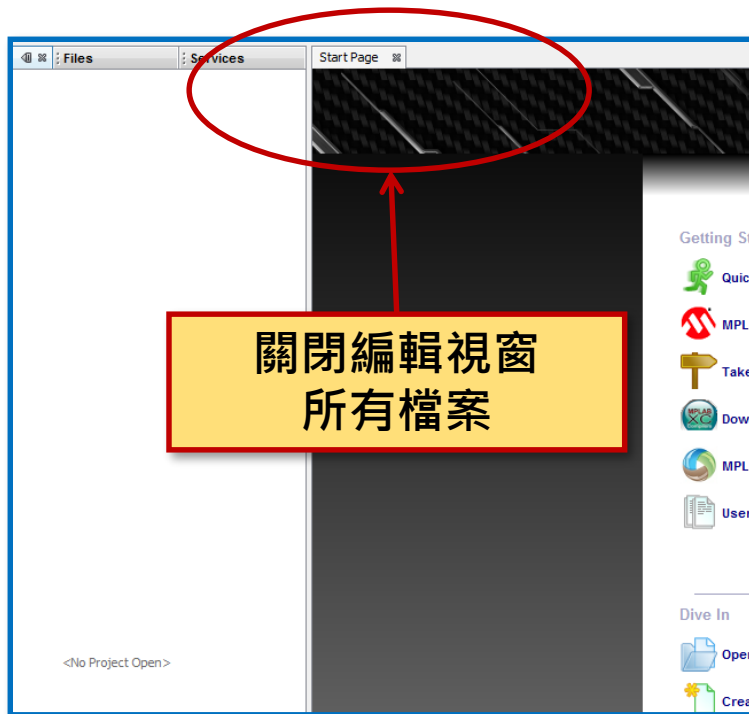


動手做實驗

PIC18 Lab2

PIC18 Lab 2

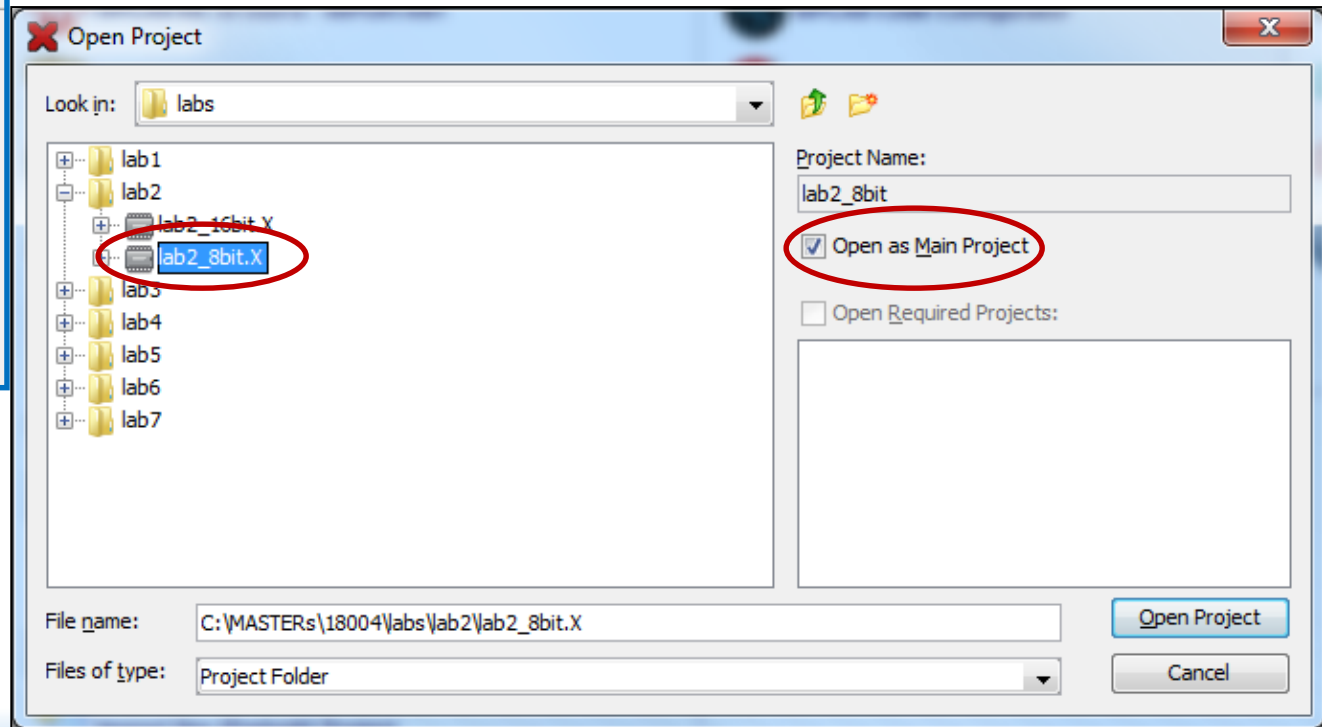
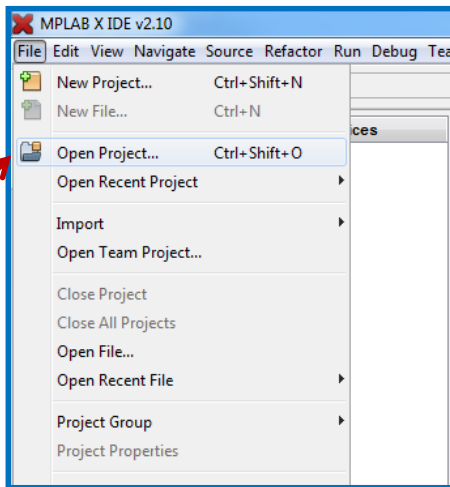
- 關閉編輯器所開啟的檔案及關閉 **MPLAB® X IDE** 下的所有專案



關閉所有的
專案，或直接
用右鍵直接點
選專案名稱後
選擇關閉

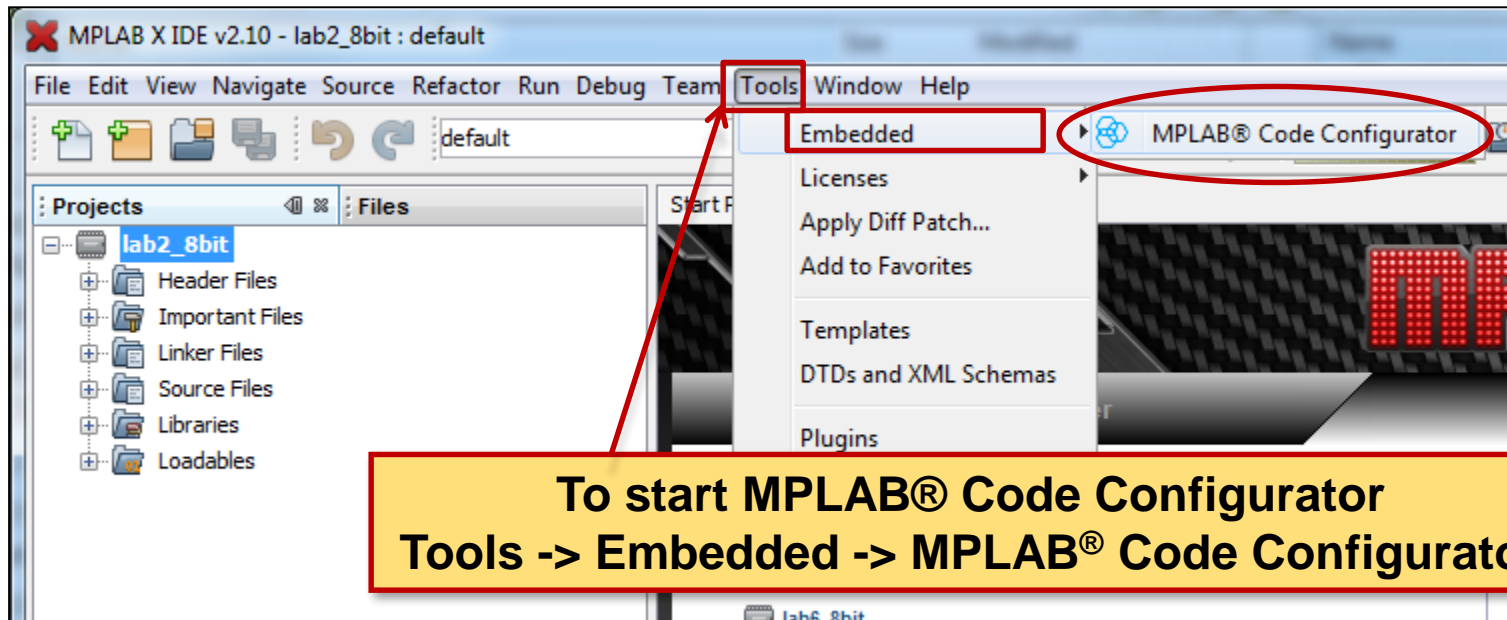
PIC18 Lab 2

- 開啟 lab2_8bit.X



PIC18 Lab 2

- 啟動 MPLAB® Code Configurator



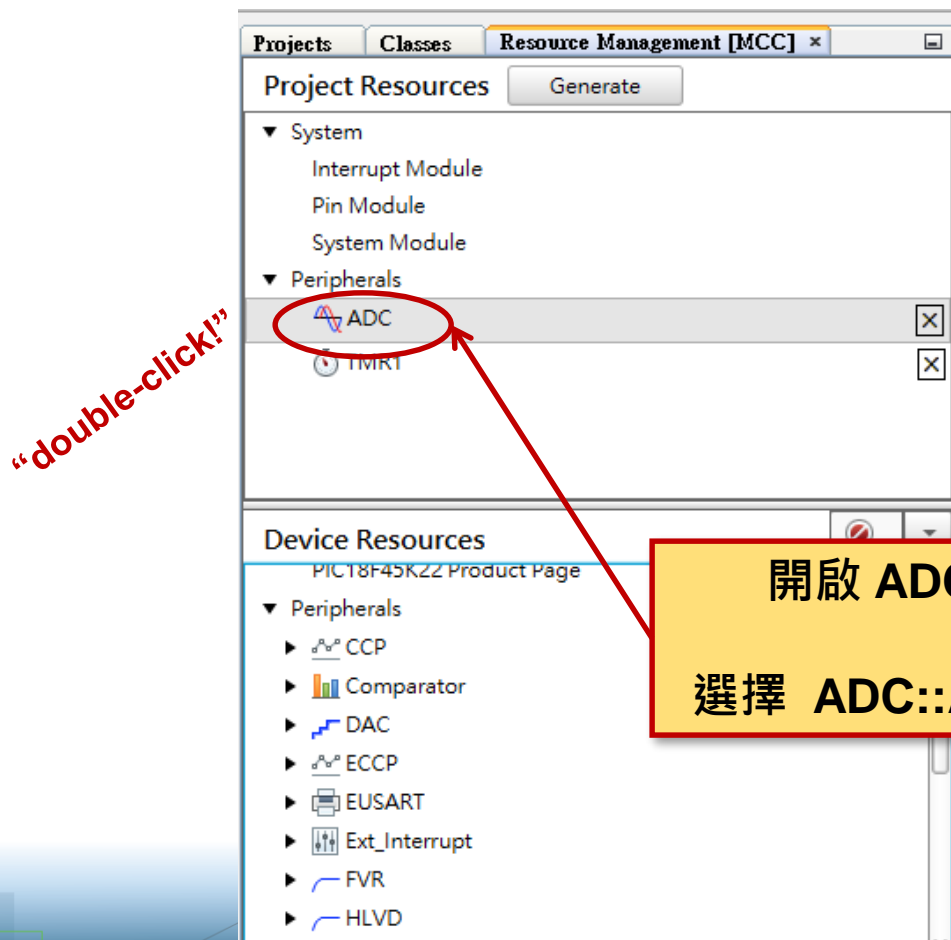


10-bit ADC 的設定

PIC18 Lab 2

ADC

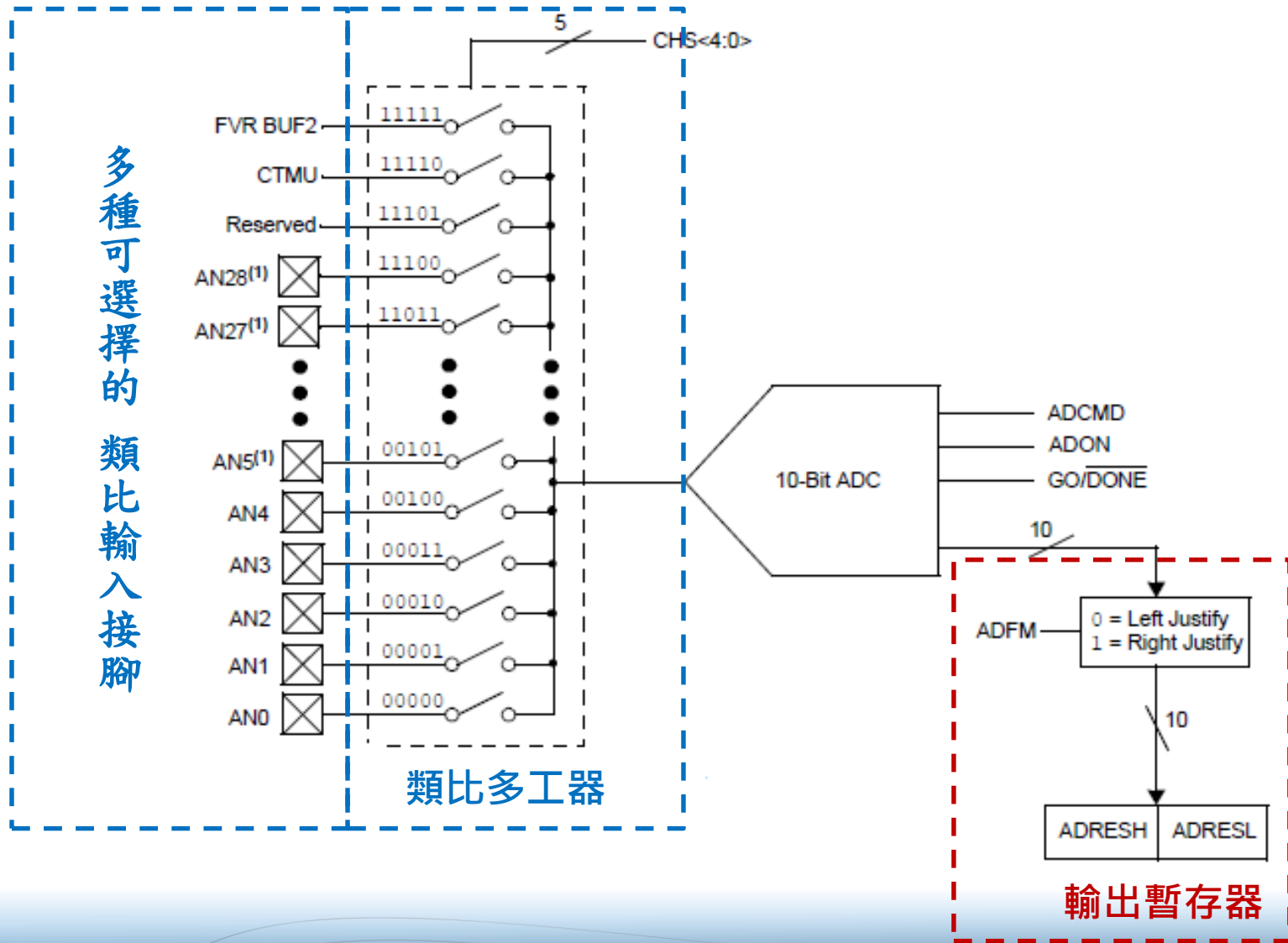
- 點選 **ADC::ADC** 選項，開啟 ADC 模組的設定



Lab 1 所設定的：
System
GPIO
Timer1

Lab2是MCC設定
是Lab1的解答

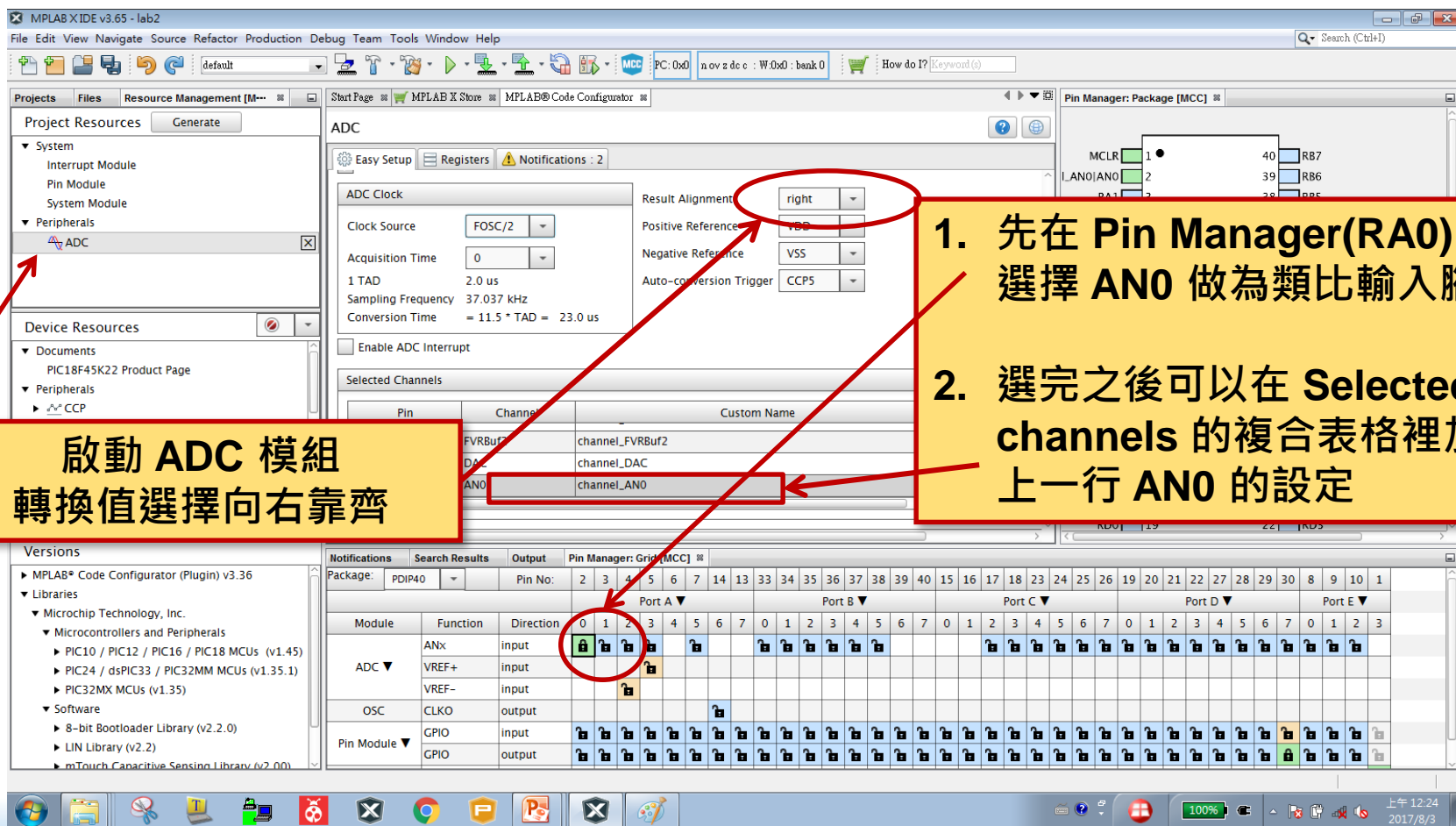
10-bit ADC 方塊圖



PIC18 Lab 2

ADC

- 選擇 ADC::ADC 就會秀出 ADC 設定視窗



ADC 設定視窗

ADC Clock: FOSC/2

Acquisition Time: 0

1 TAD: 2.0 us

Sampling Frequency: 37.037 kHz

Conversion Time: 11.5 * TAD = 23.0 us

Result Alignment: right

Positive Reference: VDD

Negative Reference: VSS

Auto-conversion Trigger: CCP5

Enable ADC Interrupt: ☐

Selected Channels:

Pin	Channel	Custom Name
FVRBuf2	channel_FVRBuf2	
DAC	channel_DAC	
AN0	channel_AN0	

Pin Manager: Package [MCC]

Grid:

Module	Function	Direction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ADC	ANx	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	
	VREF+	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	
	VREF-	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	
OSC	CLKO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
Pin Module	GPIO	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
	GPIO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	

“click!”

啟動 ADC 模組
轉換值選擇向右靠齊

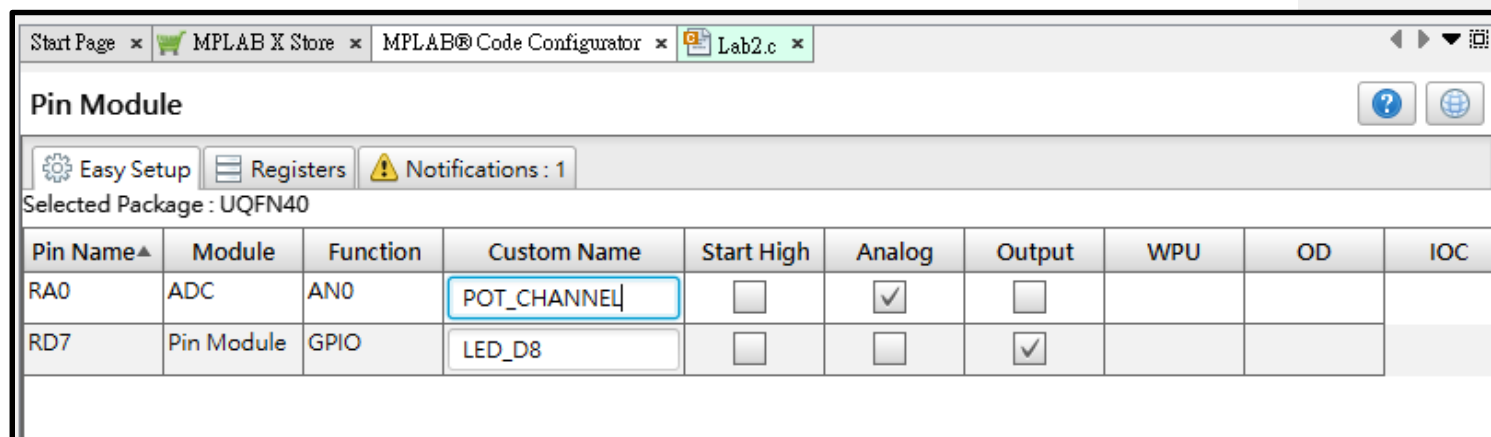
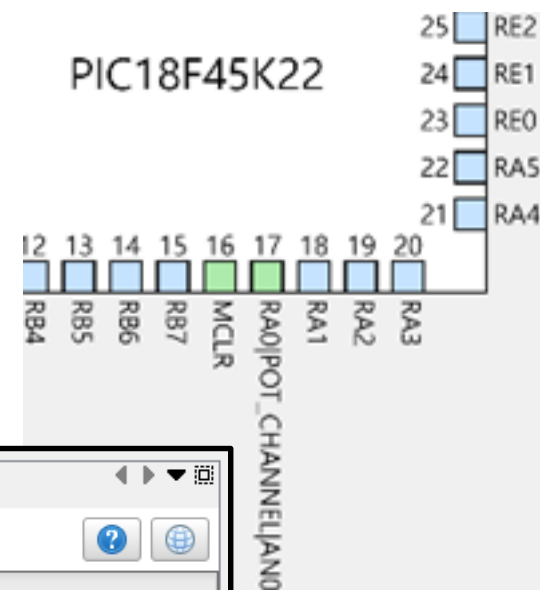
1. 先在 Pin Manager(RA0)
選擇 AN0 做為類比輸入腳

2. 選完之後可以在 Selected
channels 的複合表格裡加
上一行 AN0 的設定

PIC18 Lab 2

ADC

- 到Pin Module，更改 AN0 的名稱為 POT_CHANNEL



PIC18 Lab 2

ADC

- 更改 AN0 的名稱為 POT_CHANNEL

typedef enum

```
{  
    channel_CTMU = 0x1D,  
    channel_DAC = 0x1E,  
    channel_FVRBuf2 = 0x1F,  
    POT_CHANNEL = 0x00  
} adc_channel_t;
```

adc.h :
在列舉型別裡將
POT_CHANNEL
設成 0x00 連結
到 AN0 的輸入腳

修改名稱 AN0 為
POT_CHANNEL

				channel_CTMU
				channel_DAC
				channel_FVRBuf2
RA0	19	AN0		POT_CHANNEL

有關類比的 I/O 腳

- **PIC18F45K22 有 30 ADC 輸入通道**
 - ◆ ADC輸入腳散佈在 PORTA, PORTB, PORTC, PORTD & PORTE
- 沒有被設定的腳為依舊維持在類比輸入功能
- **Lab1 RD7 已經被設成 I/O 的輸出腳**
- **Lab2 已將 RA0 (AN0) 設為 ADC 輸入**
- 注意: 目前 **RD7 & RA0** 有設定，其餘腳位的設定均維持開機時的設定



EUSART1

的設定

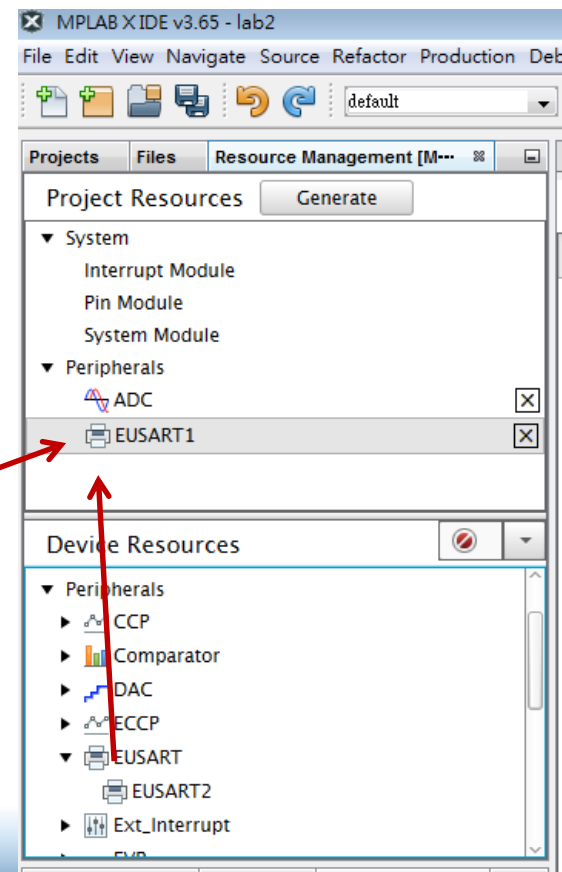
PIC18 Lab 2

UART

- 開啟 EUSART1 並選擇 EUSART
UART 通訊格式
 - ◆ 9600,N,8,1

開啟 EUSART 功能並選擇 EUSART1 模組

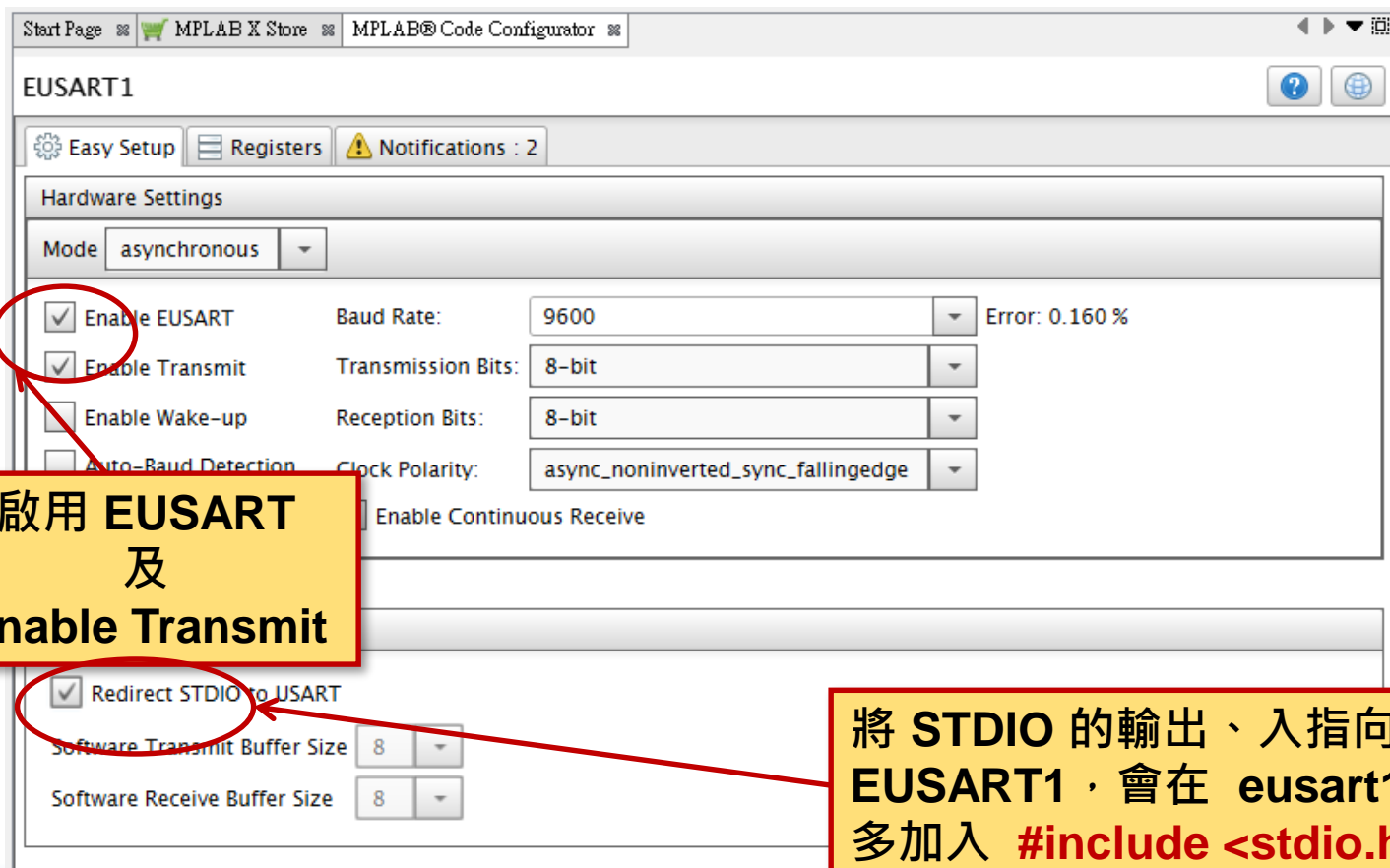
“double-click!”



PIC18 Lab 2

UART

- 依據底下的圖示選擇設定 **EUSART** 的工作模式



PIC18 Lab 2

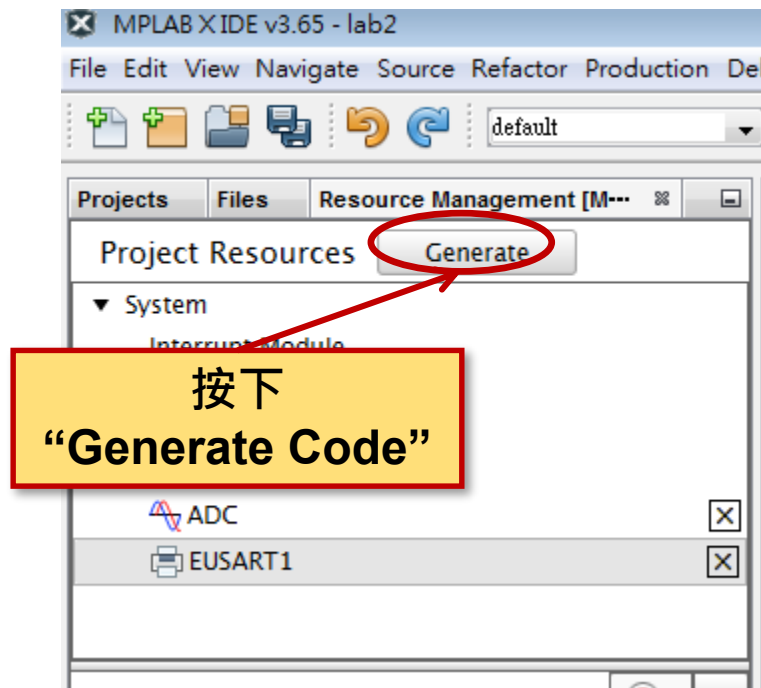
UART

- 傳輸與接收腳位將會被啟用(綠色上鎖的圖示)

Notifications			Search Results		Output	Pin Manager: Grid [MCC] ⓘ																																			
Package:		PDIP40	▼	Pin No:	2	3	4	5	6	7	14	13	33	34	35	36	37	38	39	40	15	16	17	18	23	24	25	26	19	20	21	22	27	28	29	30	8	9	10	1	
					Port A ▼							Port B ▼							Port C ▼							Port D ▼							Port E ▼								
Module		Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3		
ADC ▼	ANx	input		🔒	🔒	🔒	🔒		🔒			🔒	🔒	🔒	🔒	🔒	🔒					🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	
	VREF+	input					🔒																																		
	VREF-	input				🔒																																			
EUSART1 ▼	RX1	input																																							
	TX1	output																																							
OSC	CLKO	output									🔒																														
Pin Module ▼	GPIO	input		🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	
	GPIO	output		🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	
RESET	MCLR	input																																						🔒	

PIC18 Lab 2

- 最後按下 “**Generate Code**” 產生 **Project Resources** 項目裡的程式共五項周邊



PIC18 Lab 2

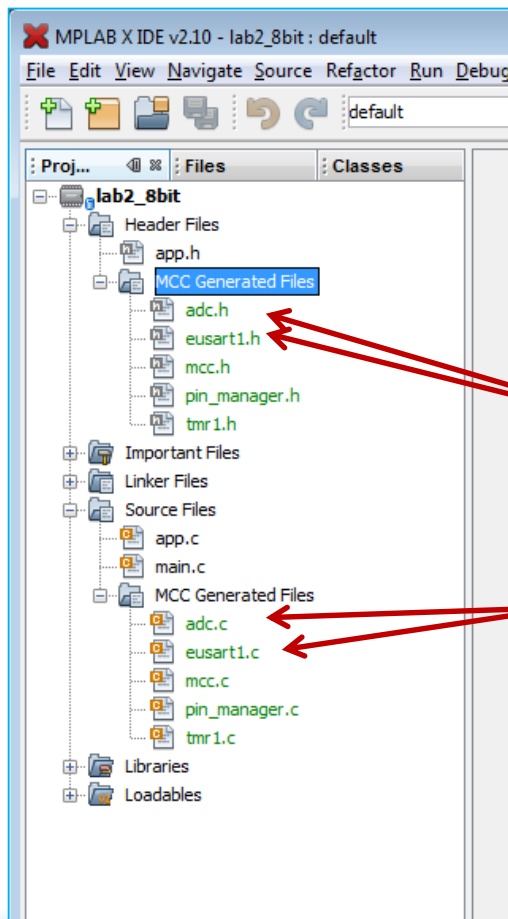
相關程式說明

- 加入了兩個 h 檔及兩個 c 檔

- ◆ adc.h
- ◆ adc.c
- ◆ eusart1.h
- ◆ eusart1.c

mcc.h 檔案裡將會加入:

```
#include <xc.h>
#include "pin_manager.h"
#include "tmr1.h"
#include "adc.h"
#include "eusart1.h"
```



**MPLAB® Code
Configurator driver
files for ADC and
EUSART1 added**



eusart1.c

- **void EUSART1_Initialize(void)**
 - ◆ 初始設定 UART1 函數
 - ◆ 使用 UART1 功能之便須先呼叫此函數
- **uint8_t EUSART1_Read(void)**
 - ◆ 自接收模組讀取一個 Byte 的資料
 - ◆ Polling RC1IF 位元模式方式
- **void EUSART1_Write(uint8_t txData)**
 - ◆ 傳送一個 Byte 出去

自己加入的應用程式

app.c & Lab2.c

- **app.c**

- ◆ ConvertADCVoltage()
 - 將 **ADC** 的值乘上單位電壓後轉換成電壓值
 - 將電壓值轉成十進制的值
- ◆ puts1USART()
 - 從 **UART1** 傳送 **const** 字串出去

- **Lab2.c**

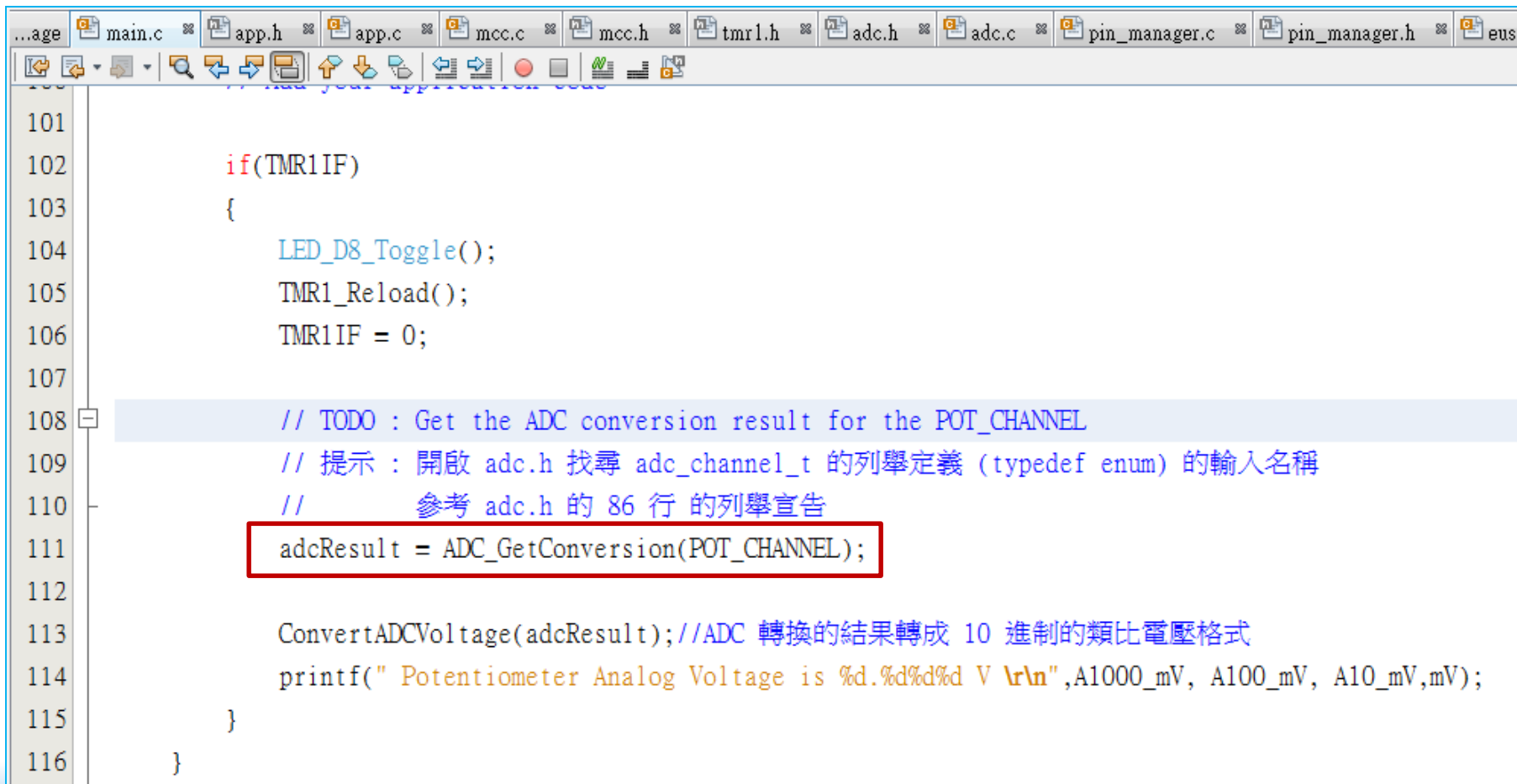
```
printf("\r\nMCC: It's Easy!\r\nPIC18 Lab 2\r\n");  
puts1USART("\r\nMCC: It's Easy!\r\nPIC18 Lab 2\r\n");
```

- ◆ 以上兩行敘述，功能一樣，但實際上有何不同？

PIC18 Lab 2

相關程式說明

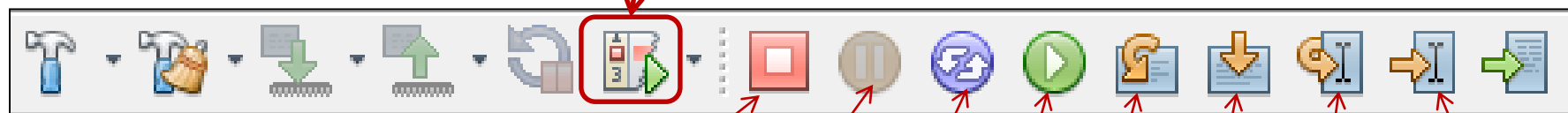
- 開啟 Lab2.c 搜尋 TODO 加入正確的程式



```
...age main.c app.h app.c mcc.c mcc.h tmr1.h adc.h adc.c pin_manager.c pin_manager.h eus
101
102     if(TMR1IF)
103     {
104         LED_D8_Toggle();
105         TMR1_Reload();
106         TMR1IF = 0;
107
108         // TODO : Get the ADC conversion result for the POT_CHANNEL
109         // 提示：開啟 adc.h 找尋 adc_channel_t 的列舉定義 (typedef enum) 的輸入名稱
110         // 參考 adc.h 的 86 行的列舉宣告
111         adcResult = ADC_GetConversion(POT_CHANNEL);
112
113         ConvertADCVoltage(adcResult); //ADC 轉換的結果轉成 10 進制的類比電壓格式
114         printf(" Potentiometer Analog Voltage is %d.%d%d V \r\n", A1000_mV, A100_mV, A10_mV, mV);
115     }
116 }
```


PIC18 Lab 2

- 按下 “**Debug Main Project**”
 - ◆ 程式會先編譯、燒錄後進入除錯模式



退出除錯模式

Pause(暫停)

Reset

Run

Step Over

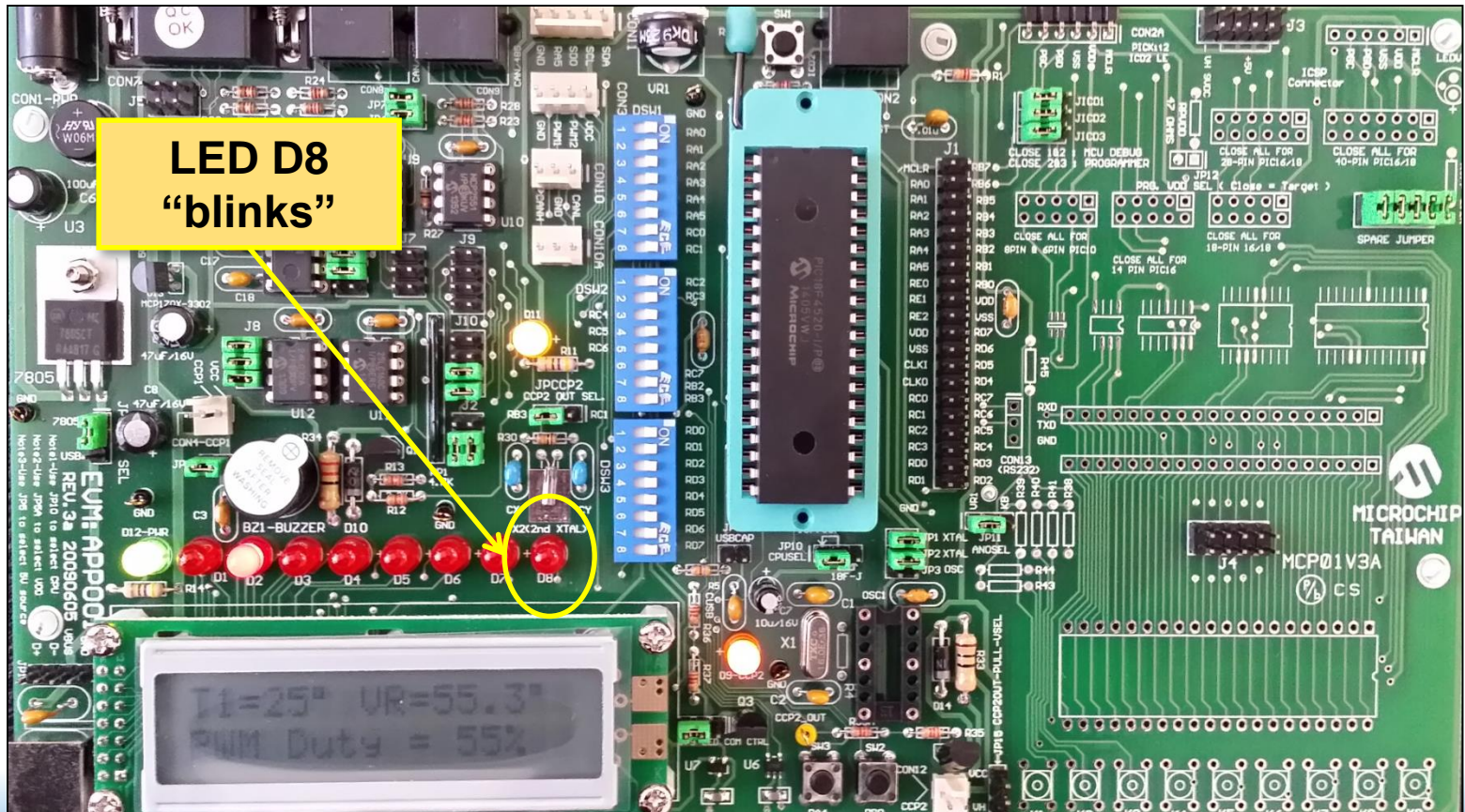
Step Into

Run To Course

Set PC
To Course

PIC18 Lab 2

- 按下“Run”的圖示，LED_D8 會每秒閃爍一次，那 UART 呢？



PIC18 Lab 2

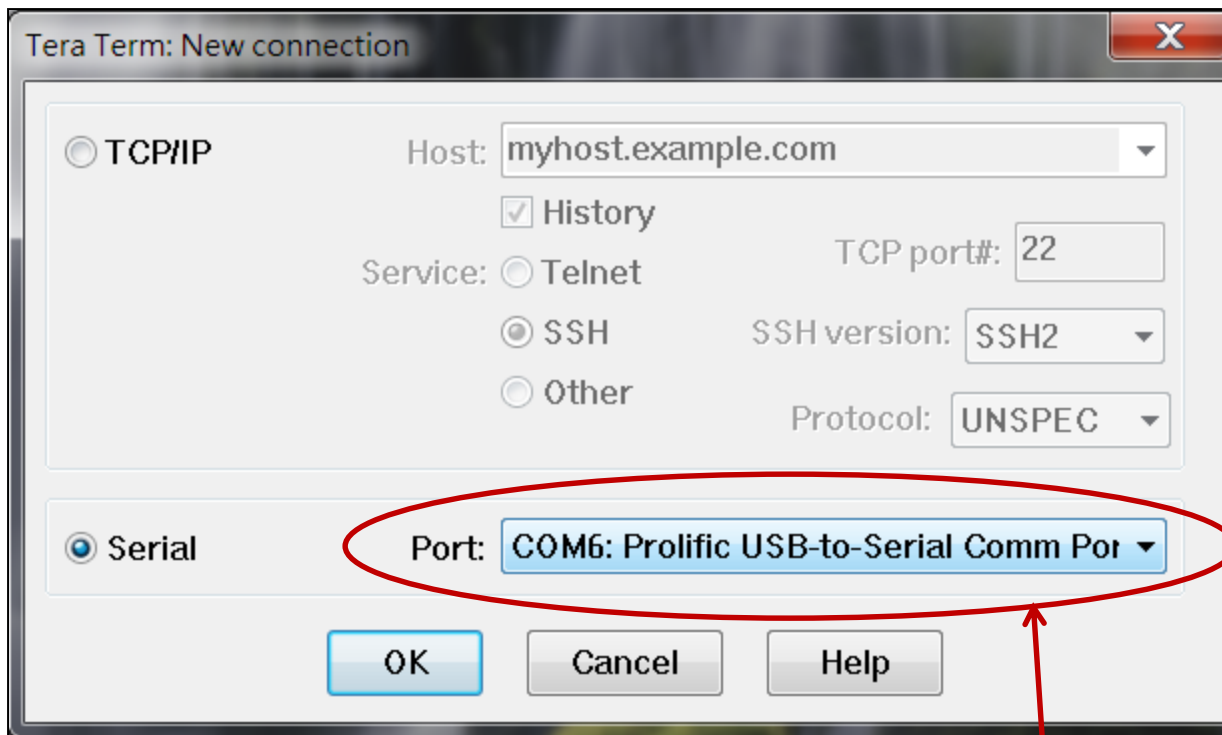
終端機顯示

- 使用 Tera Term 終端機模擬程式
- 因現存電腦 RS-232 介面幾乎消失了
 - ◆ 使用 RS-232 轉 USB 纜線做連接
 - ◆ 確定驅動程式已安裝
- 開啟裝置管理員
 - ◆ 右圖是模擬出一個 COM6



PIC18 Lab 2

終端機開啟時的顯示



選擇裝置管理員
所模擬的 COMx

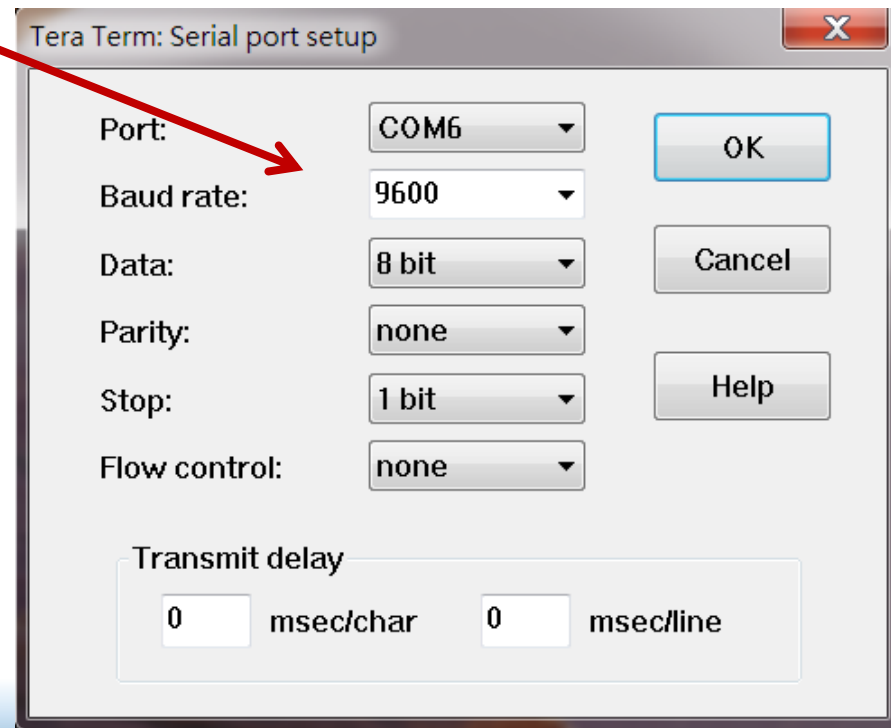
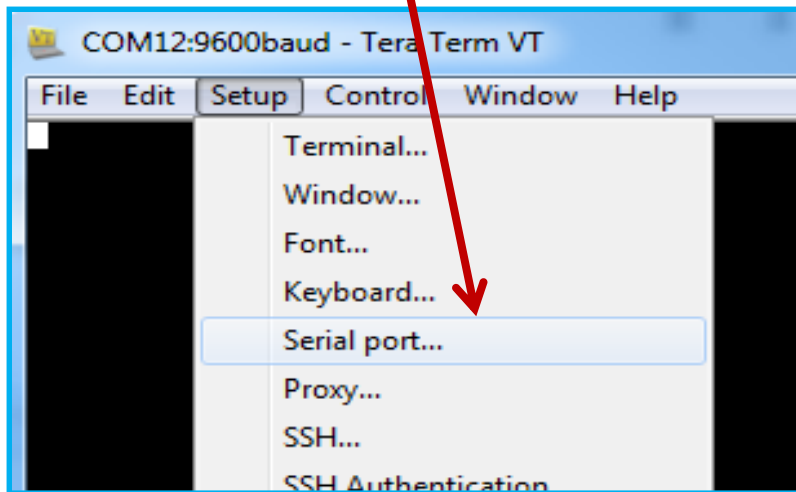
注意: 所模擬出的通訊口會不一樣

PIC18 Lab 2

終端機通訊協定設定

- 設定串列通訊協定

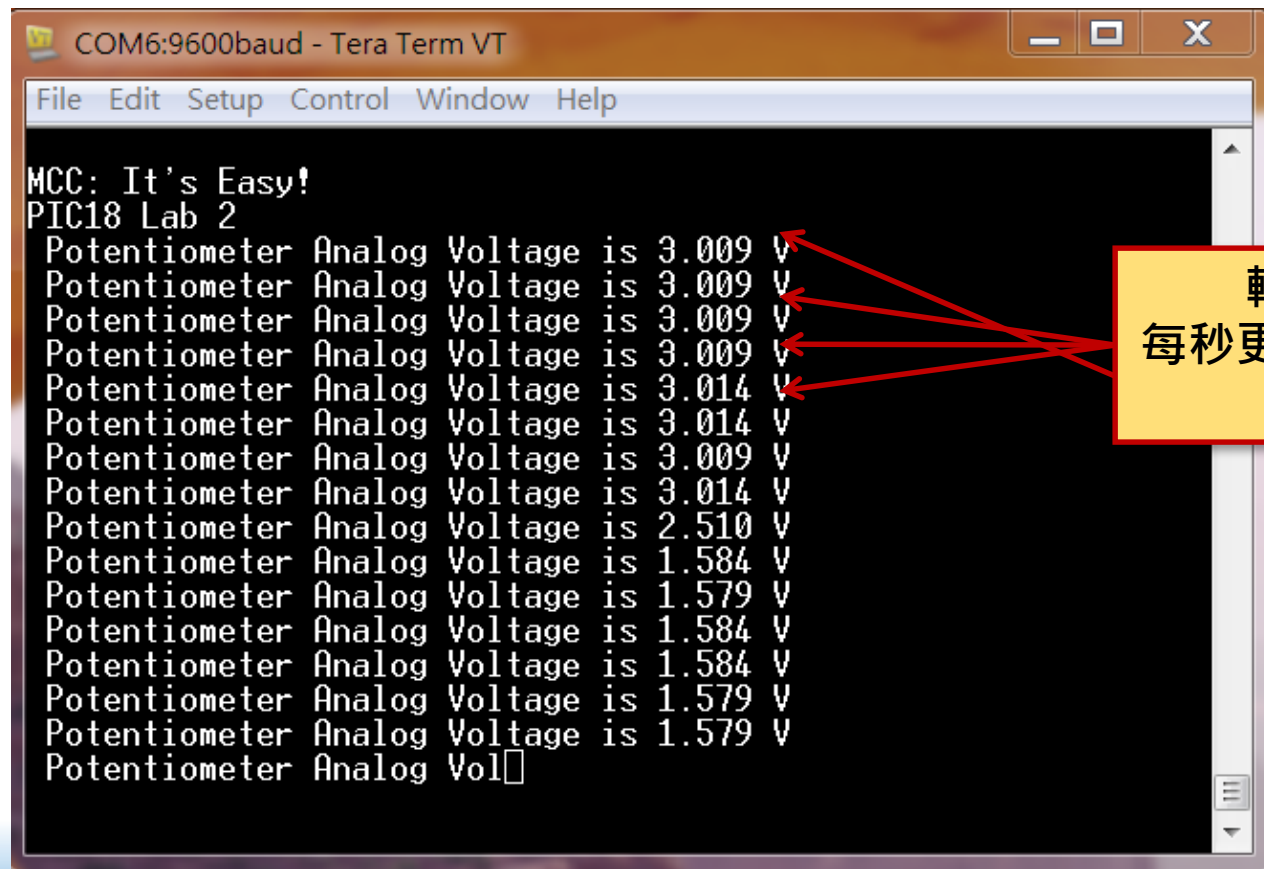
Choose Serial Port
under “Setup” and
configure the
following settings



PIC18 Lab 2

終端機顯示

- **Tera Term** 會每秒顯示 **UART** 所傳送過來的可變電阻的電壓值



```
COM6:9600baud - Tera Term VT
File Edit Setup Control Window Help
MCC: It's Easy!
PIC18 Lab 2
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.014 V
Potentiometer Analog Voltage is 3.014 V
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.014 V
Potentiometer Analog Voltage is 2.510 V
Potentiometer Analog Voltage is 1.584 V
Potentiometer Analog Voltage is 1.579 V
Potentiometer Analog Voltage is 1.584 V
Potentiometer Analog Voltage is 1.584 V
Potentiometer Analog Voltage is 1.579 V
Potentiometer Analog Voltage is 1.579 V
Potentiometer Analog Vol
```

轉動可變動阻(VR1)
每秒更新一次，AN0當下偵測
到的電壓

PIC18 Lab 2 結論

- 簡易的設定 **ADC** 讀取可變電阻的數值並將其換算成電壓值
- 設定 **EUSART1** 的傳輸模式，將可變電阻的電壓透過 **UART** 傳給終端機 (Tera Term) 來顯示



MICROCHIP

Regional Training Centers

PIC18 Lab 2-1

加入 LCD 模組的顯示



APP001_LCD.c

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



動手做實驗

PIC18 Lab 2-1

開啟 Lab2-1_8bit.x 專案

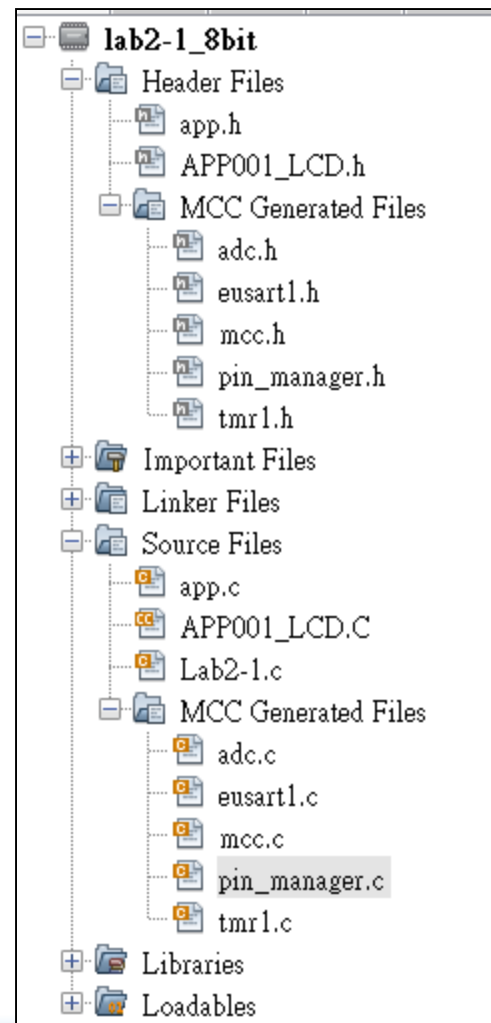
- 右側為 **Lab2-1_8bit.x** 專案下的相關檔案

- ◆ 自己的應用程式:

- **App.c**
- **APP001_LCD.c**
- **Lab2-1.c**

- ◆ **MCC 產生的程式**

- **Mcc.c (System 的項目)**
- **Adc.c**
- **Eusart1.c**
- **Pin_manager.c**
- **Tmr1.c**



APP001_LCD.c 所提供的函數

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

設定 GPIO::GPIO

- 設定 RA2 為 Digital I/O
- 設定 PORTD 為 Digital I/O

Package: PDIP40 Reverse Pin Order

		PORTA▼								PORTB▼								PORTC▼								PORTD▼							
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Module	Function																																
TMR1	T1G																																
TMR1	T1CKI																																
GPIO	I/O	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
ADC	VREF-																																
ADC	VREF+																																
ADC	ANx	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
EUSART1	TX1																																
EUSART1	RX1																																

Diagram illustrating the pin configuration for the PIC18F MCC RTC_TW device. The package is PDIP40. The pin order is 2, 3, 4, 5, 6, 7, 14, 13, 33, 34, 35, 36, 37, 38, 39, 40, 15, 16, 17, 18, 23, 24, 25, 26, 19, 20, 21, 22, 27, 28, 29, 30, 8. The diagram shows the configuration for the GPIO module, specifically for the RA2 pin (pin 2) and the PORTD module (pins 19-26). The RA2 pin is configured as a Digital I/O pin. The PORTD module is configured as a Digital I/O module. The diagram also shows the configuration for the ADC module (pins 0-7) and the EUSART1 module (pins 15-18).

GPIO 腳位定義

- 將目前所選擇到的 I/O 腳都設成輸出模式，初始輸出設為 Hi
- 賦予每個 I/O 腳新的名稱

Start Page x MPLAB X Store x MPLAB® Code Configurator x Lab2-1.c x

Pin Module

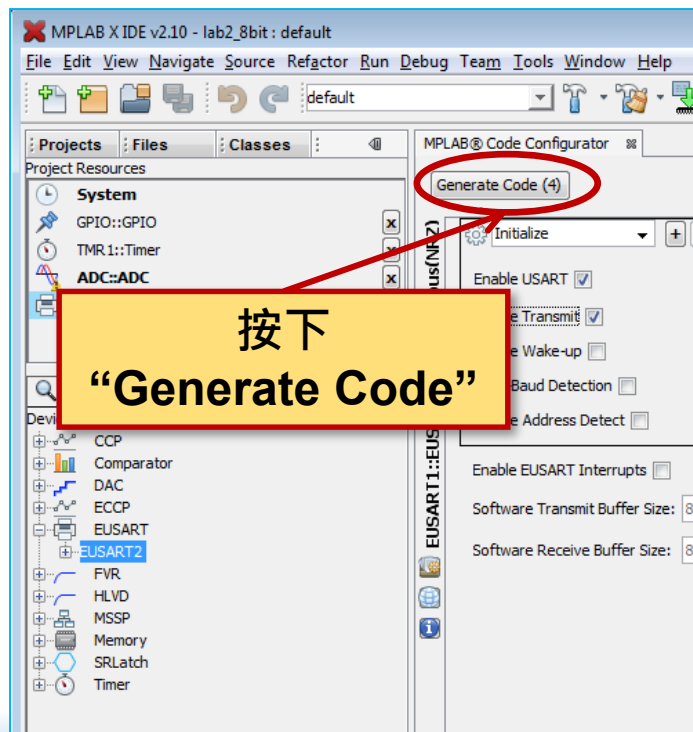
Easy Setup Registers Notifications : 1

Selected Package : UQFN40

Pin Name▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA0	ADC	AN0	POT_CHANNE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
RA2	Pin Module	GPIO	LCD_E	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RC6	EUSART1	TX1		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RC7	EUSART1	RX1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
RD0	Pin Module	GPIO	LCD_DATA0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD1	Pin Module	GPIO	LCD_DATA1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD2	Pin Module	GPIO	LCD_DATA2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD3	Pin Module	GPIO	LCD_DATA3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD4	Pin Module	GPIO	LCD_RS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD5	Pin Module	GPIO	LCD_RW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD7	Pin Module	GPIO	LED_D8	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			

PIC18 Lab 2-1

- 跟之前的動作一樣完成 MCC 設定後，按下 **“Generate Code”** 產生 Project Resources 項目裡的程式共五項周邊



Lab2-1 程式說明

- **Pin_manager.c & pin_manager.h**
 - ◆ 會加入 RA2 & PORTD 的設定
- 所以只需針對 **Lab2-1.c** 的主程式進行修改
- 找出 **Lab2-1.c** 裡的 **TODO**，依照提示修改，讓 **LCD** 第二行可以顯示類比電壓值

```
ADC Hex = 0x03FF  
ADC Volt= 4.910V
```

LCD 顯示幕

Lab2-1 TODO

- 依照提示，輸入底下程式

```
// 在 LCD 第二行顯示 "ADC Volt= 4.910V"
LCD_Set_Cursor(1,0);
putrsLCD((const far char*)"ADC Volt=      V");
LCD_Set_Cursor(1,10);
// TODO : 在 LCD 的第二行顯示出類比的電壓值
// 提示：使用 putcLCD( ) 函數來顯示一個字元，
//          目前經十進制轉換後的值為 BCD 型態，需再轉成 ASCII 後才可以在 LCD 上顯示
putcLCD(A1000_mV + '0');           // 印出個位數的電壓值
putcLCD('.');                       // 印出小數點
putcLCD(A100_mV + '0');            // 印出十分位的電壓值
putcLCD(A10_mV + '0');             // 印出百分位的電壓值
putcLCD(mV + '0');                 // 印出千分位的電壓值 (mV)
```



MICROCHIP

Regional Training Centers

PIC18 Lab 3:

使用 Timer1 中斷

PIC18 Lab 3 目標

- 將 **Lab 2** 的詢問模式轉換成中斷模式
- 在 **Timer1** 一秒的中斷函數裡做 **LED** 轉態，設定中斷處理旗號後，由主程式判斷將可變電阻的電壓值經 **UART** 傳送終端機上顯示
- **UART** 採中斷接收模式，接收終端機輸入的“**0 ~ 9**” **ASCII** 字元並顯示在 **LCD**

PIC18 Lab 3 操作概述

- 關閉 **MPLAB® X IDE** 所有已開啟的專案
- 開啟 **lab3_8bit.X** 的專案
- 啟動 **TMR1** 的中斷設定
 - ◆ 設定中斷裡還要再呼叫一個函數 (Callback Function)
- 開啟 **EUSART1** 的接收中斷
- 配置中斷管理方式
 - ◆ 啟用 高、低中斷優先權方式
 - ◆ TMR1 設成**高**優先權中斷
 - ◆ EUSART 接收 (RxD)設成**低**優先權中斷

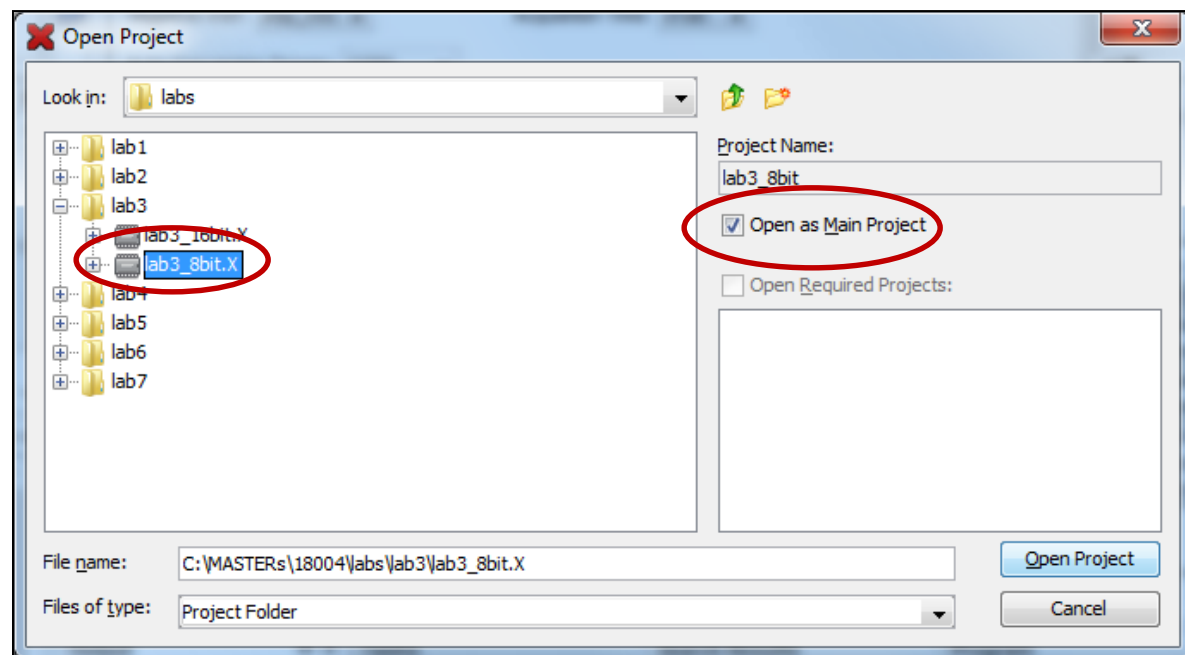
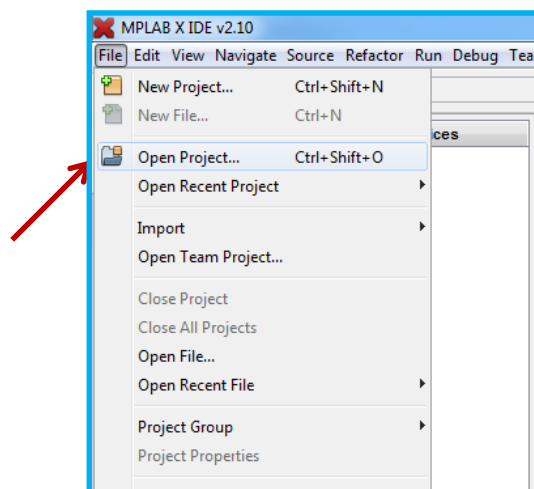


動手做實驗

PIC18 Lab 3

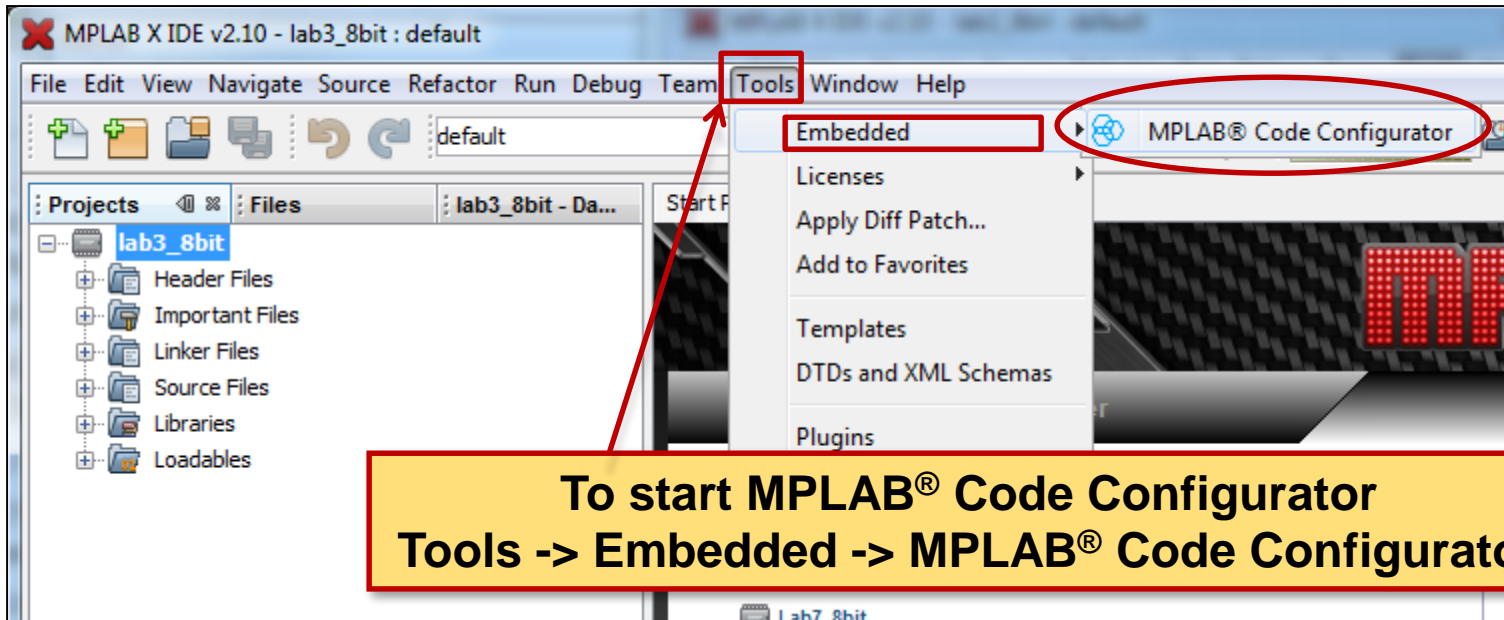
PIC18 Lab 3

- 先關閉舊的專案
- 開啟 lab3_8bit.X



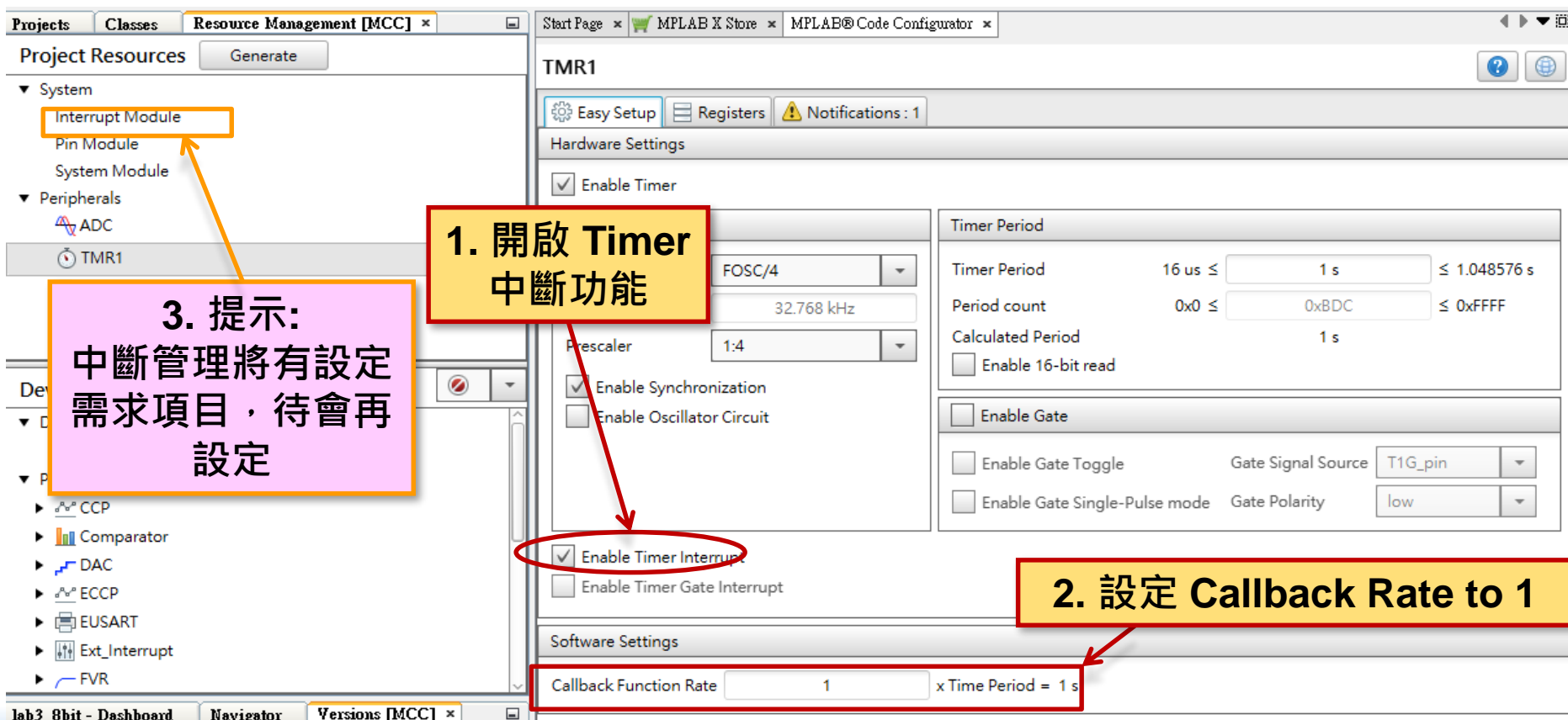
PIC18 Lab 3

- 啟動 MPLAB® Code Configurator



PIC18 Lab 3

- 設定 TMR1 的周邊需求
 - ◆ 開啟中斷功能, 設定 Callback 為 1



3. 提示:
中斷管理將有設定需求項目, 待會再設定

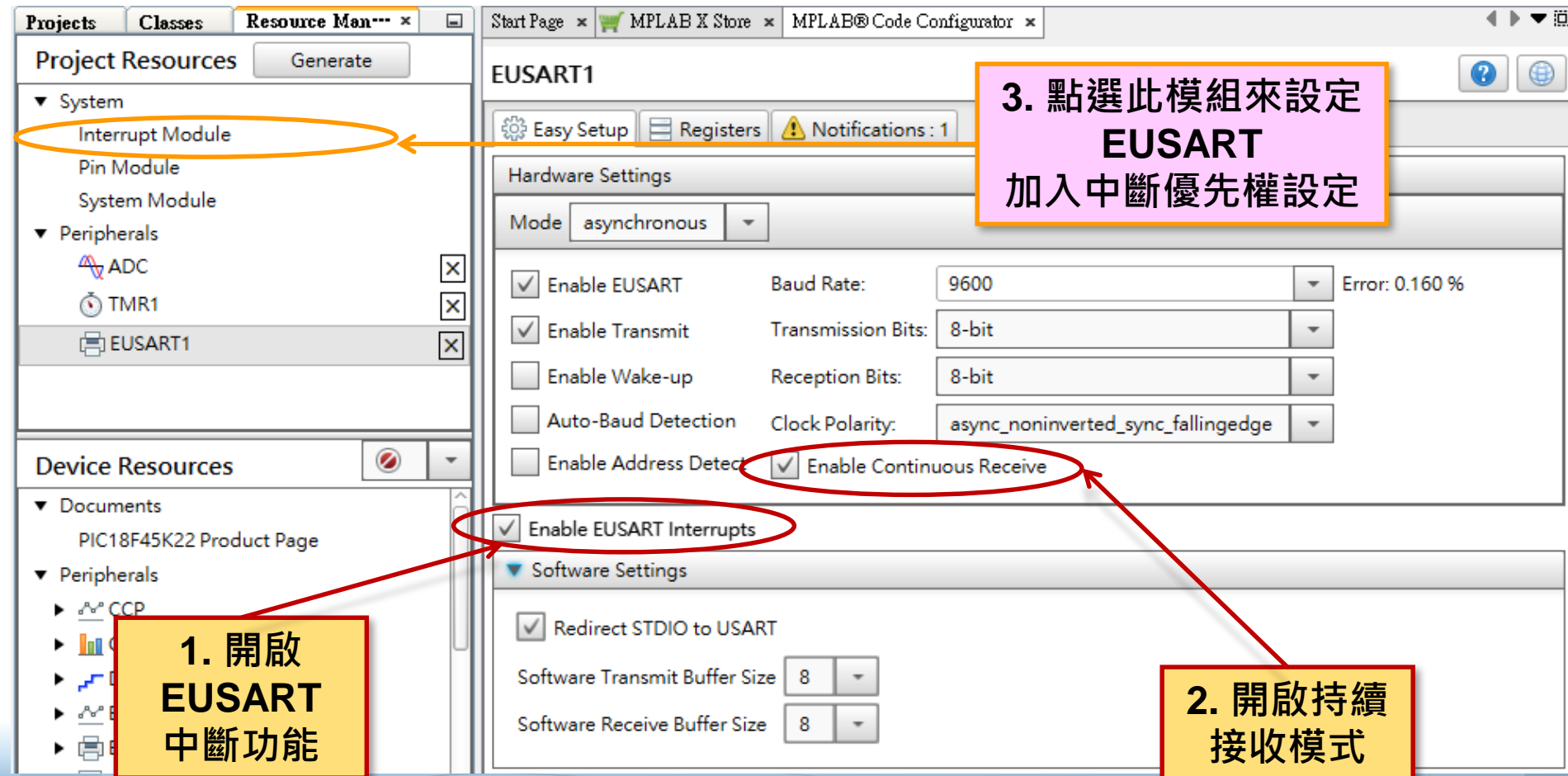
1. 開啟 Timer 中斷功能

2. 設定 Callback Rate to 1

Callback Function Rate: 1 x Time Period = 1 s

PIC18 Lab 3

- 選擇 EUSART1 並開啟 EUSART1 的接收中斷，勾選 Continuous Receive



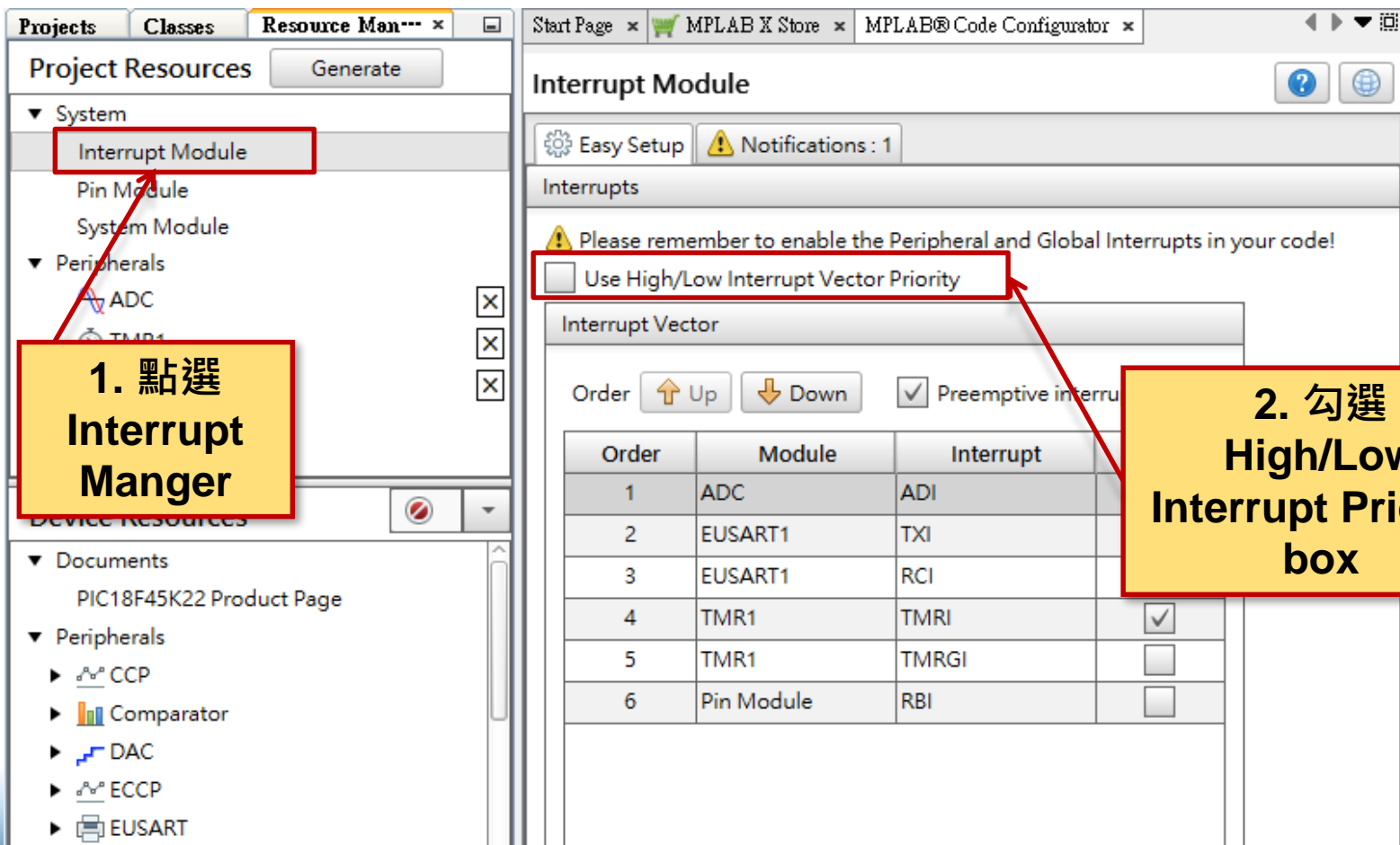
1. 開啟 EUSART 中斷功能

2. 開啟持續接收模式

3. 點選此模組來設定 EUSART 加入中斷優先權設定

PIC18 Lab 3

- 點選 “Interrupt Manager” 開啟中斷對話視窗
- 開啟 高、低中斷優先權向量



1. 點選 Interrupt Manager

2. 勾選 High/Low Interrupt Priority box

Interrupt Module

Easy Setup Notifications : 1

Interrupts

Please remember to enable the Peripheral and Global Interrupts in your code!

☐ Use High/Low Interrupt Vector Priority

Interrupt Vector

Order Up Down ☒ Preemptive interrupt

Order	Module	Interrupt	
1	ADC	ADI	
2	EUSART1	TXI	
3	EUSART1	RCI	
4	TMR1	TMRI	<input checked="" type="checkbox"/>
5	TMR1	TMRGI	<input type="checkbox"/>
6	Pin Module	RBI	<input type="checkbox"/>

PIC18 Lab 3

- 設定 **Timer1** 為高優先權中斷
- 移動 **EUSART** **ISRs** 為低優先權中斷

Interrupts

⚠ Please remember to enable the Peripheral and Global Interrupts in your code!

☒ Use High/Low Interrupt Vector Priority

**Timer 1
High Priority**

High Priority Interrupt Vector

Order ☒ Preemptive interrupt routine

Order	Module	Interrupt	Enabled
1	ADC	ADI	<input type="checkbox"/>
2	TMR1	TMRI	<input checked="" type="checkbox"/>
3	TMR1	TMRGI	<input type="checkbox"/>
4	Pin Module	RBI	<input type="checkbox"/>

Low Priority Interrupt Vector

Order ☒ Preemptive interrupt routine

Order	Module	Interrupt	Enabled
1	EUSART1	RCI	<input checked="" type="checkbox"/>
2	EUSART1	TXI	<input checked="" type="checkbox"/>

Move the EUSART ISRs to Low Priority

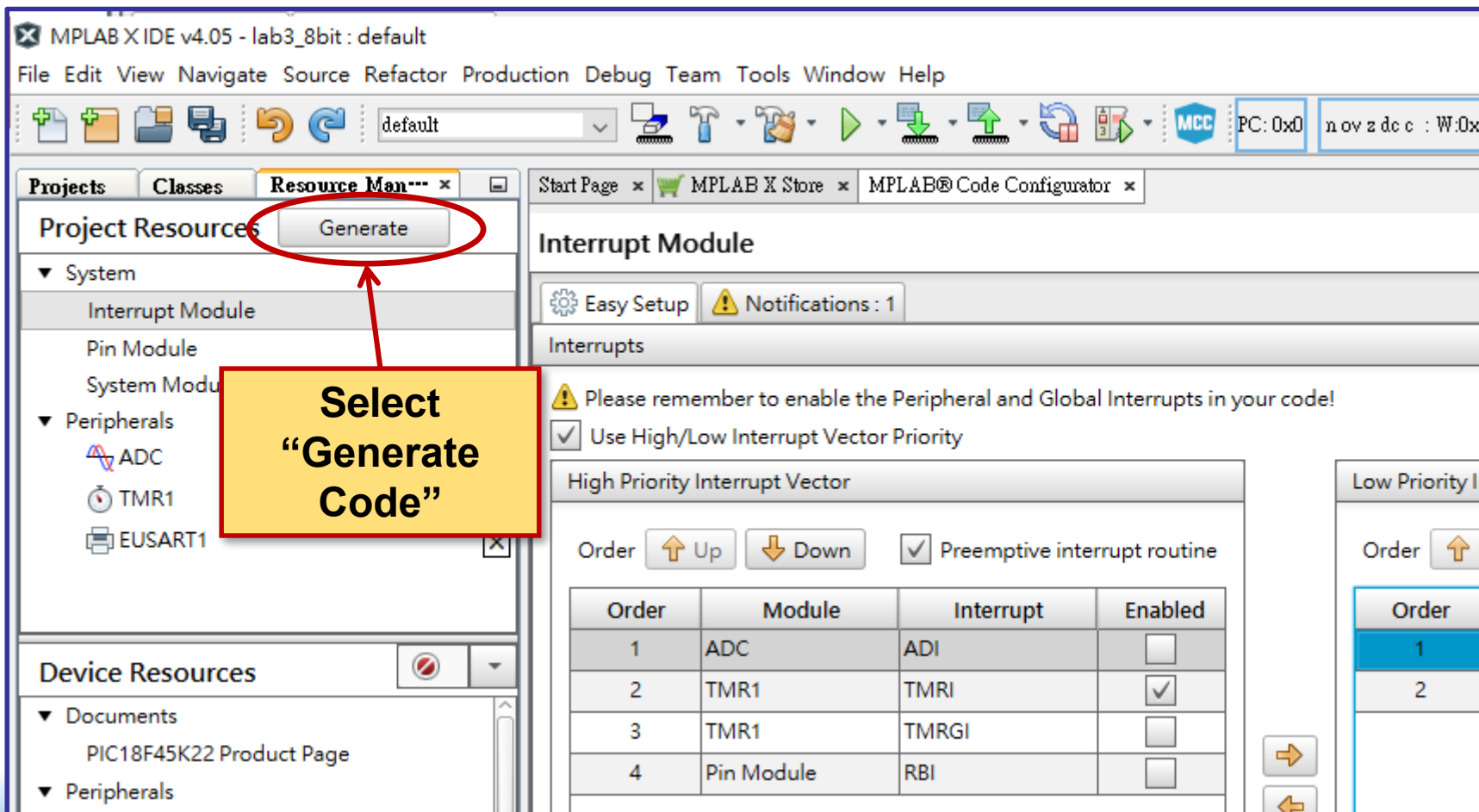
此時 **Output** 視窗會有提示:

Interrupt Manager: Please remember to enable the Peripheral and Global Interrupts in your code!

請別忘了，程式裡要開啟相關的周邊中斷致能 (xxIE) 位元及主中斷 (GIEH & GIEL) 位元

PIC18 Lab 3

- 最後按下 “Generate code” 的選項

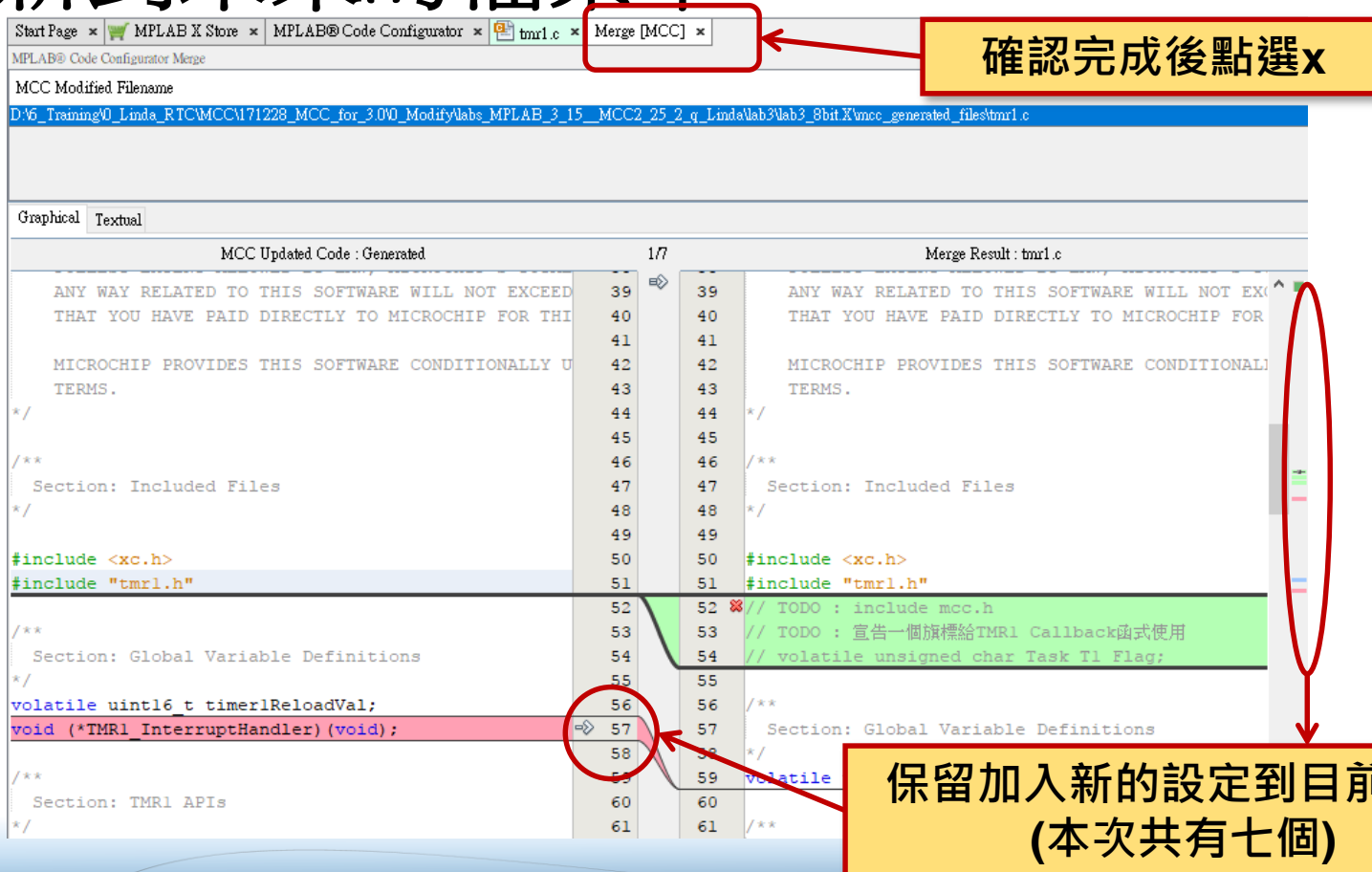


The screenshot shows the MPLAB X IDE v4.05 interface. The 'Resource Manager' window is open, displaying a tree view of project resources. The 'Generate' button is highlighted with a red circle, and a red arrow points from it to a yellow callout box containing the text 'Select "Generate Code"'. The 'Interrupt Module' window is also visible, showing the 'Easy Setup' tab and a table of interrupt modules.

Order	Module	Interrupt	Enabled
1	ADC	ADI	<input type="checkbox"/>
2	TMR1	TMRI	<input checked="" type="checkbox"/>
3	TMR1	TMRGI	<input type="checkbox"/>
4	Pin Module	RBI	<input type="checkbox"/>

PIC18 Lab 3

- 左畫面為新產生的設定，移到右畫面使其更新到未來的檔案中



Start Page x MPLAB X Store x MPLAB Code Configurator x **tmr1.c x Merge [MCC] x**

MPLAB Code Configurator Merge

MCC Modified Filename
D:\6_Training0_Linda_RTC\MCC171228_MCC_for_30W_Modify\abs_MPLAB_3_15_MCC2_25_2_q_Linda\lab3\lab3_8bit_X\mcc_generated_files\tmr1.c

Graphical Textual

MCC Updated Code : Generated	1/7	Merge Result : tmr1.c
ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED	39	39 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EX
THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THI	40	40 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR
	41	41
MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY U	42	42 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONAL
TERMS.	43	43 TERMS.
*/	44	44 */
/**	45	45
Section: Included Files	46	46 /**
*/	47	47 Section: Included Files
	48	48 */
#include <xc.h>	49	49
#include "tmr1.h"	50	50 #include <xc.h>
	51	51 #include "tmr1.h"
	52	52 // TODO : include mcc.h
	53	53 // TODO : 宣告一個旗標給TMR1 Callback函式使用
	54	54 // volatile unsigned char Task Ti Flag;
	55	55
/**	56	56 /**
Section: Global Variable Definitions	57	57 Section: Global Variable Definitions
*/	58	58 */
volatile uint16 t timer1ReloadVal;	59	59 volatile
void (*TMR1 InterruptHandler)(void);	60	60
	61	61 /**

確認完成後點選x

保留加入新的設定到目前檔案 (本次共有七個)

中斷這樣就設定好了嗎？

- 這樣好像太簡單了，那 **FAE** 不就要失業了...
- 還早呢！還有一些中斷的事項要在設定
 - ◆ 主程式裡的中斷開啟
 - ◆ 中斷函數裡要執行的中斷即時事件處理

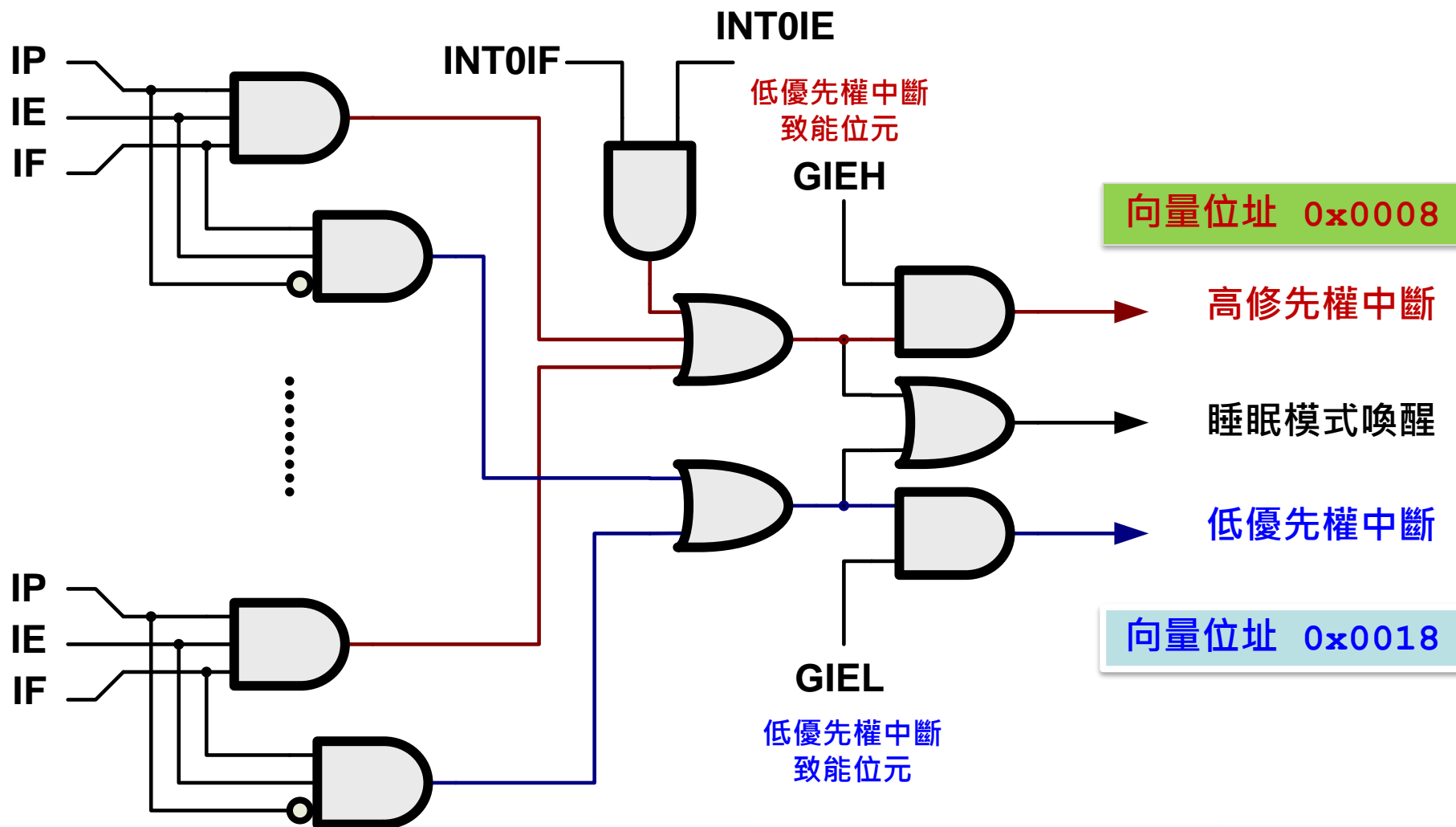


XC8 的中斷介紹

- 再複習一下有關 **PIC18F** 的中斷
- **XC8** 對 **PIC18** 的高、低優先權中斷的宣告
- 有關中斷控制的相關位元

PIC18Fxxx 中斷邏輯電路

高、低優先權模式



PIC18F 中斷處理

- **PIC18Fxxxx** 有兩個中斷進入點
 - ◆ 高優先權==>中斷位址0x0008
 - ◆ 低優先權==>中斷位址0x0018
 - ◆ 每個中斷源均可選擇其中斷優先權 (xxIP)
 - ◆ 每個中斷源均有獨立的中斷旗標 (xxIF)
 - ◆ 每個中斷源均可單獨 Enable 或 Disable (xxIE)
 - ◆ 中斷旗標的清除 ==> 需自行用軟體方式清除
 - **USART** 產生的 **TXIF** 及 **RCIF** 旗標，無法直接用軟體清除 **Flag**
 - 清除 **TXIF** → 寫入 **TXREG**
 - 清除 **RCIF** → 讀取 **RCREG**

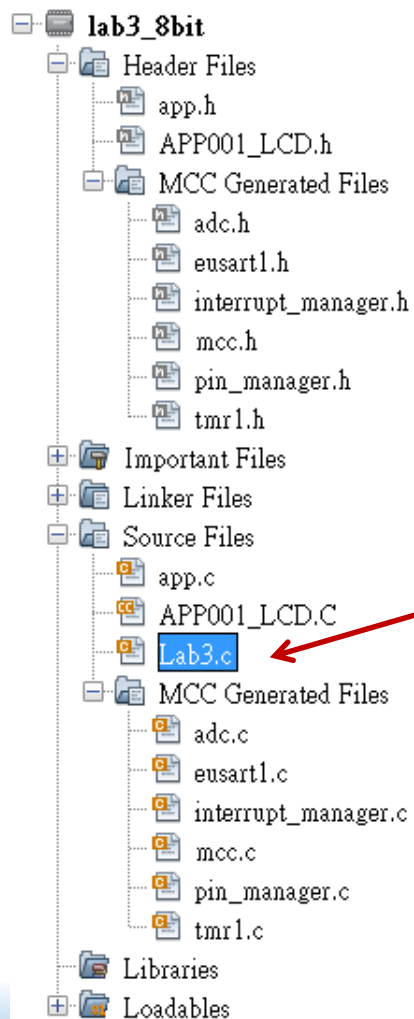
XC8 的中斷函數宣告

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

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

PIC18 Lab 3

- **Lab3 專案下的檔案**
 - ◆ 加入了中斷管理程式



開啟 Lab3.c 的
編輯視窗

專案下：
interrupt_manager.h
及
interrupt_manager.c
檔案已加入專案項裡

LAB3 裡 Timer1 中斷位元

- 要啟用 **Timer1** 高優先權中斷還要做那些設定

底下是以位元架構下的寫法:

```
PIE1bits.TMR1IE = 1;           // 啟用 Timer1 中斷
RCONbits.IPEN = 1;             // 啟用高、低優先權控制機制
IPR1bits.TMR1IP = 1;           // 設定 Timer1 為高優先權中斷
INTCONbits.GIEH = 1;           // 開啟高優先權中斷總控制位元
```

- **MCC** 裡，這些位元在哪裡被設定了

MCC 中斷位元的設定在哪?

```
PIE1bits.TMR1IE = 1;           // TMR1_Initialize( ) @ tmr1.c
RCONbits.IPEN = 1;             // INTERRUPT_Initialize ( ) @ interrupt_manager.c
IPR1bits.TMR1IP = 1;           // INTERRUPT_Initialize ( ) @ interrupt_manager.c
INTCONbits.GIEH = 1;           // main( ) @ Lab3.c
```

PIC18 Lab 3

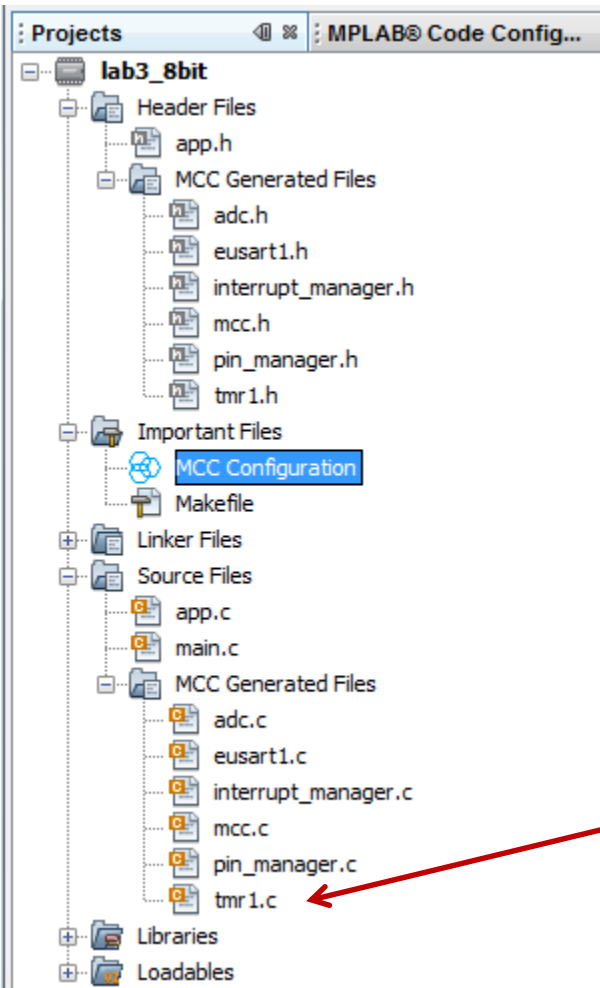
- 檢視上頁的 **Timer1** 中斷相關位元，除 **GIEH** 外其餘位元 **MCC** 都已設定完畢

```
59 void main(void)
60 {
61     int adcResult;
62
63     // Initialize the device
64     SYSTEM_Initialize();
65
66     // If using interrupts in PIC18 High Priority
67     // If using interrupts in PIC Mid-Range or Low-Byte
68     // Use the following macros to:
69
70     // TODO: 刪除註解(//)符號，啟動高優先權中斷
71     // Enable high priority global interrupts
72     INTERRUPT_GlobalInterruptHighEnable();
73
74     // TODO: 刪除註解(//)符號，啟動低優先權中斷
75     // Enable low priority global interrupts.
76     INTERRUPT_GlobalInterruptLowEnable();
77
78     // Disable high priority global interrupts
79     //INTERRUPT_GlobalInterruptHighDisable();
```

依照TODO的說明，將註解移除，啟用
函數：High/Low Priority Interrupts()；

PIC18 Lab 3

Open tmr1.c



“雙點擊” tmr1.c
開啟 tmr1.c 於編輯視窗中

檢視 interrupt_manager.c

- 提供 **INTERRUPT_Initialize()**
 - ◆ 提供系統高、低優先權的設定
- 高優先權中斷函式的宣告

```
void interrupt INTERRUPT_InterruptManagerHigh (void)
{
    // interrupt handler
    if (PIE1bits.TMR1IE == 1 && PIR1bits.TMR1IF == 1)
    {
        TMR1_ISR(); // 加入 Timer1 的即時處理中斷函式
    }
    else
    {
        //Unhandled Interrupt
    }
}
```

檢視 TMR1_ISR()

tmr1.c

→ MCC產生的tmr1.c檔案中開頭部分

```
#include <xc.h>
```

```
#include "tmr1.h"
```

```
// TODO : 為了呼叫點亮LED函式，加入pin管理標頭檔
```

```
#include "pin_manager.h"
```

```
/**
```

```
Section: Global Variable Definitions
```

```
*/
```

```
volatile uint16_t timer1ReloadVal;
```

```
// TODO : 宣告一個旗標給TMR1 Callback函式使用
```

```
volatile unsigned char Task_T1_Flag;
```



檢視 TMR1_ISR()

tmr1.c

```
void TMR1_ISR(void)
{
    PIR1bits.TMR1IF = 0;           // 清除Timer1 的中斷旗號
    TMR1 += timer1ReloadVal;       // 載入 Timer1 一秒的計時值
    TMR1_Callback( );             // 使用者加入 Timer1 的中斷服務程式
}

void TMR1_Callback(void)
{
    // Add your custom callback code here
    if(TMR1_InterruptHandler)
    {
        TMR1_InterruptHandler();
    }
}

void TMR1_SetInterruptHandler(void* InterruptHandler){
    TMR1_InterruptHandler = InterruptHandler;
}
```

檢視 TMR1_ISR()

tmr1.c

MCC產生的tmr1.c檔案中，中斷函數

```
void TMR1_DefaultInterruptHandler(void)  
{  
    // add your TMR1 interrupt custom code  
    // or set custom function using TMR1_SetInterruptHandler()  
  
    //TODO: 輸入Timer1 中斷處理程式  
    LED_D8_Toggle( );           // 此巨集定義在 pin_manager.h 裡  
    Task_T1_Flag = 1;  
}
```

TMR1_ISR()

- 使用者加入的中斷即時處理程式
- 在這裡要做：
 - ◆ **LED_D8** 的反轉動作
 - ◆ 設定 **T1_Task_Flag** ，再交由主程式處理
 - 因為 **LCD** 顯示需較長時間處理
 - 中斷裡只處理簡短的動作
 - ◆ 注意: **Task_T1_Flag** 是在中斷裡所使用的變數一定要加入**volatile** 的宣告

所以說 MCC Timer1 的中斷

- 中斷發生先跳到
 - ◆ interrupt_manager.c 裡的中斷函數:
`interrupt INTERRUPT_InterruptManagerHigh (void)`
- 再來跳到 `tmr.c`
 - ◆ `TMR1_ISR(void)` 執行清除旗號及載入Timer 值
 - ◆ 執行 `TMR1_CallBack(void)` 內的使用者所加入的中斷程式
- 對於新手就照著做吧!
- 對老手而言，這太複雜了! 直接就在 `interrupt INTERRUPT_InterruptManagerHigh (void)` 的中斷函數裡就處理完畢了?? 減少中斷響應時間。

好了，開始改 Lab3 程式

- 首先在 **tmr1.c** 的 **TMR1_CallBack ()**
 - ◆ 加入 `#include "pin_manager.h"`
 - 加入 I/O 腳位的宣告 (`pin_manager.h`) 這是 `GPIO::GPIO` 的設定，例如: `LED_D8_Toggle ()`
 - ◆ 加入 `volatile unsigned char Task_T1_Flag;`
 - 因為是中斷裡所使用的變數 (`volatile`)
 - ◆ 加入 `LED_D8` 轉態控制
 - `LED_D8_Toggle()`
 - ◆ 設定 `Task_T1_Flag = 1`

TMR1_CallBack ()

- **TMR1_CallBack** 函數是 **Timer1** 中斷裡的函數，所要處理的動作有兩項
 - ◆ 立即處理的動作：
 - **LED_D8** 轉態控制
 - ◆ 無立即性的執行需求，但需要處理的動作：
 - 如設定 **Task_T1_Flag = 1**
 - 交由主程式檢查旗號來決定是否執行

在 Lab3.c 的修改

- 加入 **Task_T1_Flag** 的宣告
 - ◆ `extern volatile unsigned char Task_T1_Flag;`
 - 為何要使用 **extern** 因為主宣告是在 **tmr1.c** 裡完成的

```
51  #include "app.h"
52  #include "APP001_LCD.h"
53
54  extern volatile unsigned char Task_T1_Flag;
```

```
105  while (1)
106  {
107      // Add your application code
108
109      if(Task_T1_Flag)
110      {
111          Task_T1_Flag=0;
112
113          adcResult = ADC_GetConversion(POT_CHANNEL);    // ADC 轉換，再讀取 ADC 的 16 進制值
114
115          // 在 LCD 第一行顯示 "ADC Hex = 0x03FF"
116          LCD_Set_Cursor(0,0);
117          putsLCD((const far char*)"ADC Hex = 0x  ");
```

透過Task_T1_Flag在主程式
中，對LCD做定時更新

在 Lab3.c 的修改

```
59 void main(void)
60 {
61     int adcResult;
62
63     // Initialize the device
64     SYSTEM_Initialize();
65
66     // If using interrupts in PIC18 High/Low Priority Mode you need to enable the
67     // If using interrupts in PIC Mid-Range Compatibility Mode you need to enable
68     // Use the following macros to:
69
70     // TODO: 刪除註解 (//) 符號，啟動高優先權中斷
71     // Enable high priority global interrupts
72     INTERRUPT_GlobalInterruptHighEnable();
73
74     // TODO: 刪除註解 (//) 符號，啟動低優先權中斷
75     // Enable low priority global interrupts.
76     INTERRUPT_GlobalInterruptLowEnable();
77
78     // Disable high priority global interrupts
79     //INTERRUPT_GlobalInterruptHighDisable();
```

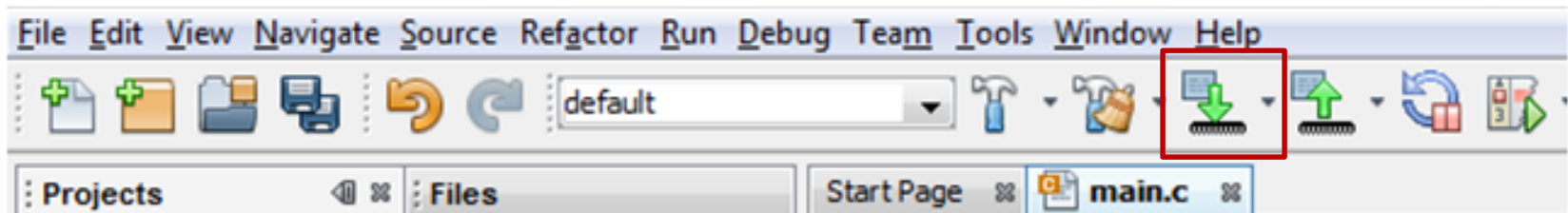
依照TODO的說明，將註解移除，啟用
函數：High/Low Priority Interrupts()；

// TODO: 刪除註解 (//) 符號，啟動高優先權中斷
// Enable high priority global interrupts
INTERRUPT_GlobalInterruptHighEnable();

// TODO: 刪除註解 (//) 符號，啟動低優先權中斷
// Enable low priority global interrupts.
INTERRUPT_GlobalInterruptLowEnable();

編譯及燒錄 Lab3_8bit 專案

- 啟動 Tera Term 終端機模擬程式
 - ◆ File → Disconnect → New connection
 - ◆ 啟用 Serial Port 的設定
 - ◆ Setup → Serial port 裡設定 (9600,N,8,1)
- 按下 “**Make and Program**” 的圖示開始編譯及燒錄



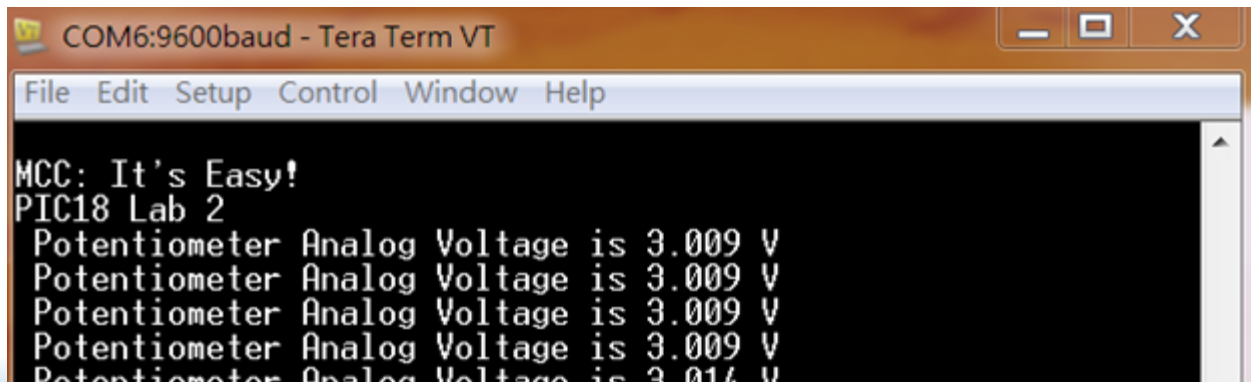
Lab3_8bit 專案 的執行

- LED _D8 每一秒閃爍一次
- LCD 會顯示

ADC Hex = 0x03FF
ADC Volt= 4.910V

LCD 顯示幕

- Tera Term 終端機每隔一秒顯示一次電壓值



```
COM6:9600baud - Tera Term VT
File Edit Setup Control Window Help
MCC: It's Easy!
PIC18 Lab 2
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.009 V
Potentiometer Analog Voltage is 3.014 V
```

PIC18 Lab 3 結論

- 我們配置了 **Timer1** 並使用高優先權的中斷模式
- 我們配置了 **EUSART1** 並使其為低優先權的中斷模式
- 我們在 **tmr1.c** 的 **callback ()** 函數裡加入 **Timer1** 發生中斷後須執行的程式



MICROCHIP

Regional Training Centers

PIC18 Lab 3-1:

EUSART 中斷接收模式

EUSART 接收相關的程式

- **eusart1.h**

- ◆ #define **EUSART1_DataReady** (eusart1RxCount)
- ◆ EUSART1 的函數原型宣告

- **eusart1.c**

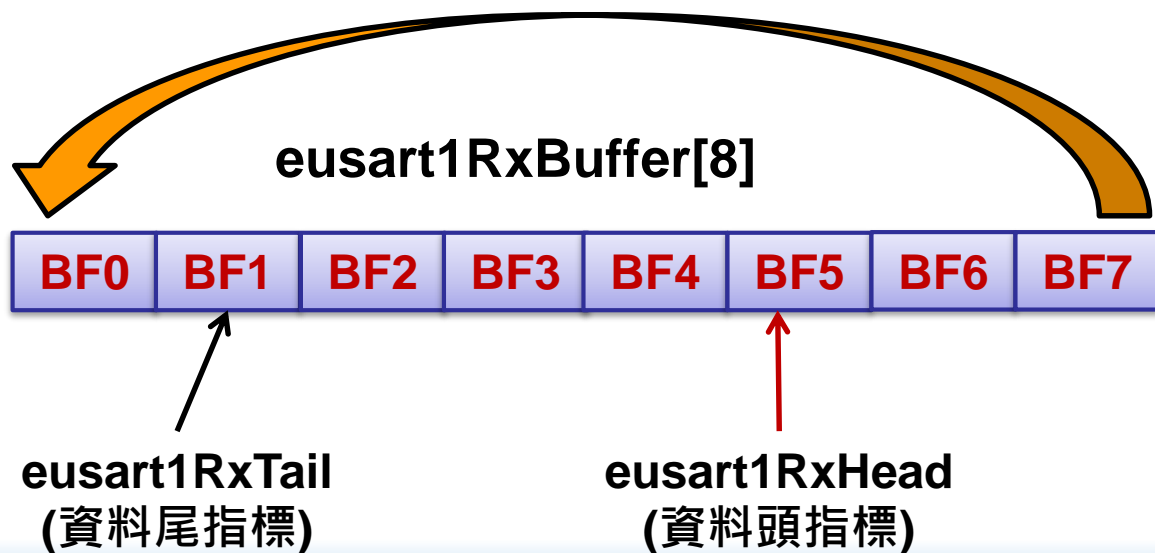
- ◆ 接收的 Ring Buffer 宣告與設定
 - **EUSART1_Initialize(viod)**
 - **EUSART1_Read(void)**
 - **EUSART1_Receiver_ISR(void)**

- **Lab3-1**

- ◆ 主程式，將接收到的資料顯示在 LCD 的第二行

接收的 Buffer 架構

- 使用 8 Bytes 陣列，用 Ring Buffer 的架構
 - ◆ static uint8_t eusart1RxHead = 0; // 資料頭的索引
 - ◆ static uint8_t eusart1RxTail = 0; // 資料尾的索引
 - ◆ static uint8_t eusart1RxBuffer[8];
 - ◆ volatile uint8_t **eusart1RxCount**; (主程式檢查此 Byte)
// 目前 Buffer 裡有幾個 Bytes 的資料還沒被拿走



eusart1.c 接收程式解說

中斷接收端

```
void EUSART1_Receive_ISR(void)
```

```
{
```

```
    if (1 == RC1STAbits.OERR)           // 檢查是否有 Over run 的發生
```

```
    {
```

```
        // EUSART1 error - restart
```

```
        RC1STAbits.CREN = 0;           // Overrun 發生只能重設 CREN 位元來清除
```

```
        RC1STAbits.CREN = 1;
```

```
    }
```

```
    // buffer overruns are ignored      // 注意此接收程式不檢查資料是否會被覆蓋
```

```
    eusart1RxBuffer[eusart1RxHead++] = RC1REG;
```

```
    // 接收資料存到 Ring Buffer 裡，資料頭索引值加一，指向下一個 Buffer 位置
```

```
    if (sizeof(eusart1RxBuffer) <= eusart1RxHead) // 資料頭索引值 等於 Buffer 的大小
```

```
    {
```

```
        eusart1RxHead = 0;           // 資料頭索引值 = 0，資料頭索引值歸零
```

```
    }
```

```
    eusart1RxCount++;                // 接收計數器加一 (收到一筆資料)
```

```
}
```

eusart1.c 接收程式解說

主程式呼叫的接收函數

```
uint8_t EUSART1_Read(void)
```

```
{  
    uint8_t readValue = 0;  
  
    while (0 == eusart1RxCount) // 主程式呼叫，檢查是否有新資料在 Ring Buffer 裡  
    {  
        //沒有新資料所要處理的程式寫在這裡  
    }
```

```
    PIE1bits.RC1IE = 0; // 暫時關閉接收中斷以避免資料被中斷打亂 (很重要)  
    readValue = eusart1RxBuffer[eusart1RxTail++]; // 自 Ring Buffer 的資料尾索引位置拿取資料  
    if(sizeof(eusart1RxBuffer) <= eusart1RxTail) // 資料尾索引值 等於 Buffer 的大小(到底了嗎?)  
    {  
        eusart1RxTail = 0; // 資料尾索引值 到底了要歸零(從頭開始)  
    }  
    eusart1RxCount--; // 接收計數器減一 (拿走一筆資料)  
    PIE1bits.RC1IE = 1; // 重先開啟接收中斷  
    return readValue; // 回傳一個接收資料  
}
```

Lab3-1.c 的 EUSART 接收程式

- 所以主程式只要檢查 **EUSART1_DataReady** 不等於零時，代表有資料在接收的 **Ring Buffer** 裡

Lab3-1.c Main Function

while (EUSART1_DataReady)

// 此定義在 eusart1.h 裡: #define EUSART1_DataReady (eusart1RxCount)

// 不為零，代表有資料在 **UART1** 接收的 **Buffer** 裡

{

LCD_Set_Cursor(1,LCD_Count); // 設定顯示位置

putcLCD(EUSART1_Read());

// 讀取 **RxD** 的 **Ring Buffer** 的值後顯示在 **LCD** 上。

if (LCD_Count++ ==15) LCD_Count=0;

// 控制 **LCD** 顯示的位置，位置 = 15 時 **Cursor** 歸零。

}

Lab3-1_8bit 專案 的執行

- LED _D8 一樣每一秒閃爍一次
- Tera Term 終端機一樣會每隔一秒顯示一次電壓值
- 用老鼠切換到 Tera Term 後，輸入鍵盤上的字就可以看到 LCD 會在第二行顯示所輸入的字元

ADC Hex = 0x03FF
1234567890ABCDEF

LCD 顯示幕

Tera Term
所輸入的字

好像發送端還沒有講

- **EUSART1_Write(uint8_t txData)** 為傳送函數
 - ◆ 不使用中斷傳輸模式 (一般使用)
 - 確定 **TX1IE = 0**
 - 呼叫 **EUSART1_Write()** 傳輸資料直接寫入 **TX1REG** 直接傳送出去
 - ◆ 中斷方式傳輸，用 **Tx Ring Buffer** 來做傳輸
 - 設定 **TX1IE = 1**
 - 呼叫 **EUSART1_Write()**，傳入資料將被送到 **Ring Buffer** 後再啟動 **TX1IE**
 - 傳輸中斷立即發生，經 **EUSART1_Transmit_ISR()** 做中斷傳送

euasrt1.c 傳送程式解說

```
void EUSART1_Write(uint8_t txData)
{
    while(0 == eusart1TxBufferRemaining) // 檢查 Tx Ring Buffer 是否還有暫存資料
    { // 沒有，在這裡輸入您的程式
    }
    if (0 == PIE1bits.TX1IE)           // 中斷傳輸模式是否開啟？
    {
        TX1REG = txData;                // 沒有使用中斷模式傳輸，送出傳輸資料，
        // 這裡沒有做TX Busy 的檢查，所以呼叫者要做
    }
    else
    {
        // 使用中斷傳輸模式
        PIE1bits.TX1IE = 0;
        eusart1TxBuffer[eusart1TxHead++] = txData; // 資料寫到 Tx Ring Buffer 裡
        if (sizeof(eusart1TxBuffer) <= eusart1TxHead) // 檢查索引值是否到頂了
        {
            eusart1TxHead = 0; // 索引值到頂，歸零
        }
        eusart1TxBufferRemaining--;
    }
    PIE1bits.TX1IE = 1; // 啟動傳輸中斷
}
```



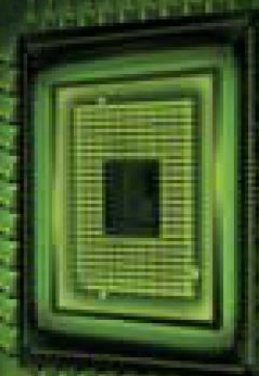
MICROCHIP

Regional Training Centers

PIC18 Lab 4:

I²C™ 串列通訊

用 Master 模式存取 24LC32A



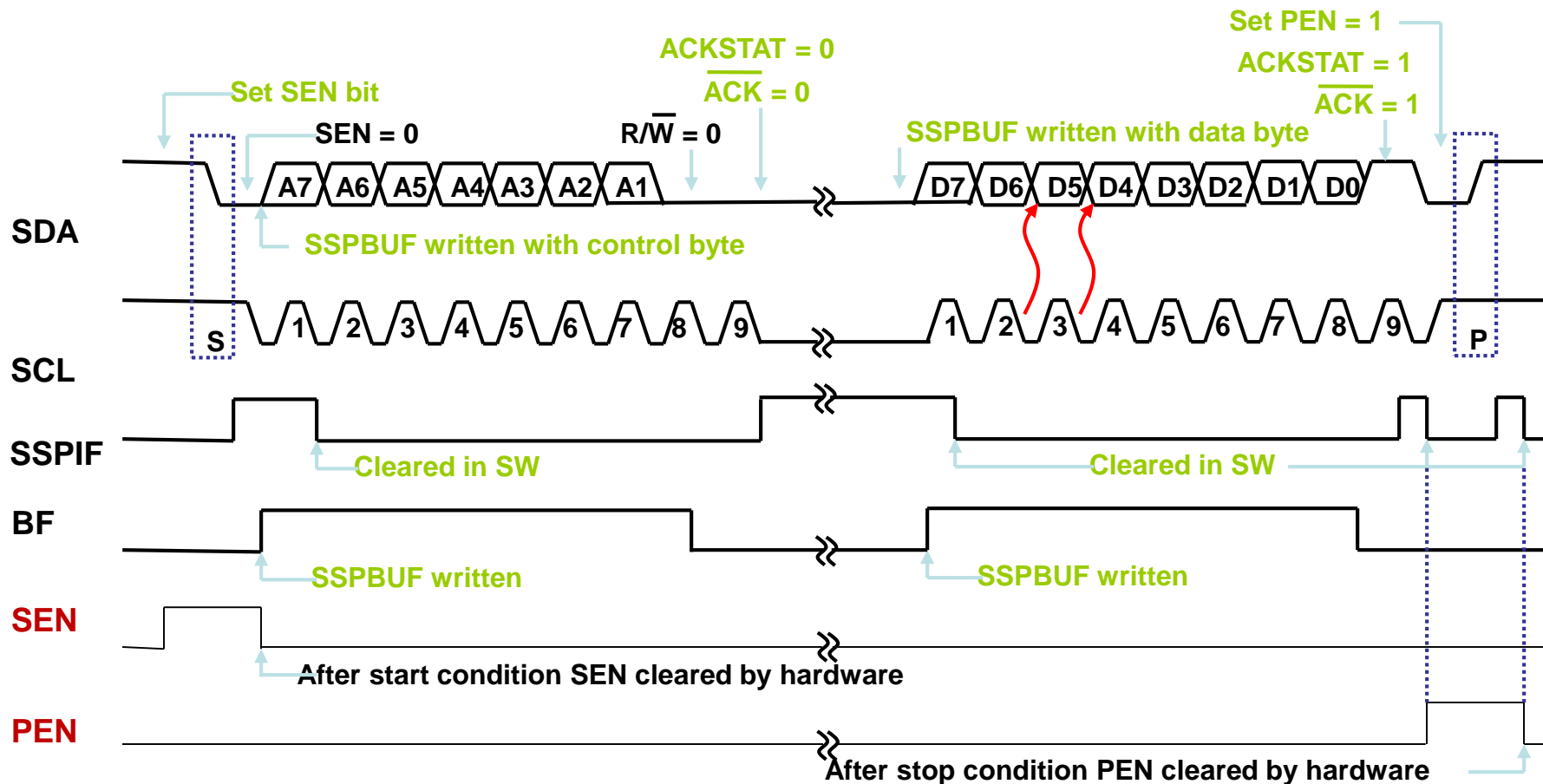
PIC18 Lab 4 目標

- 使用 **I²C™** 同步型串列通訊
- 設置 **I²C™ Master** 模式做寫入和讀取一個 **I²C™ Slave** 元件 **24LC32A** **EEPROM**
- 重設 **EUSART** 利用終端機來顯示各項訊息

PIC18 Lab 4 概述

- 先關閉 **MPLAB® X IDE** 下所有的專案及檔案
- 開啟 **..\Labs\Lab4\lab4_8bit.X** 的專案
- 設定 **Timer1, ESUART**
- 設定 **I²C™ Module**

I²C™ Master 模式下 發送資料的時序



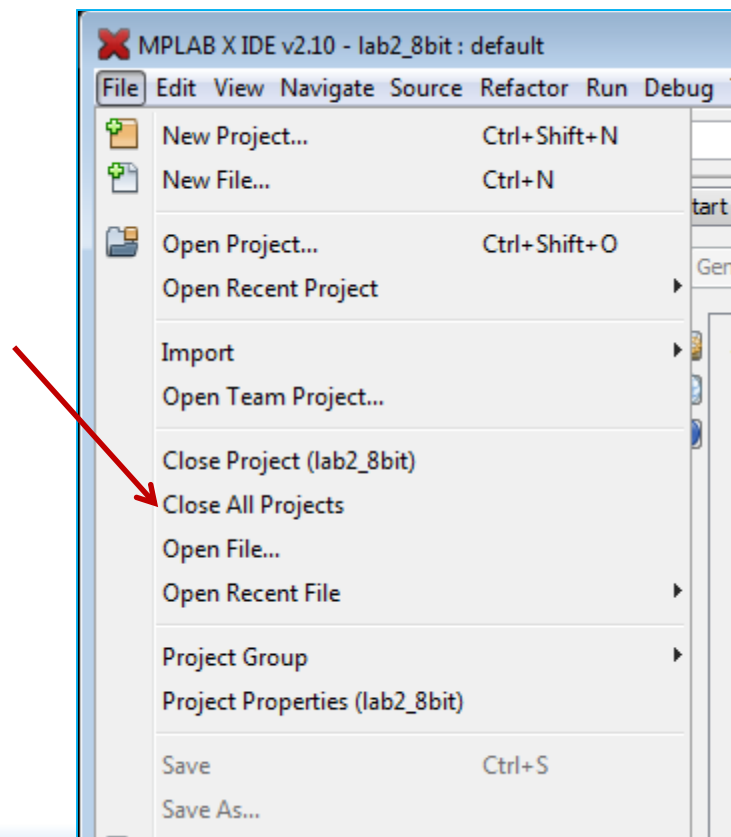


動手做實驗

PIC18 Lab 4

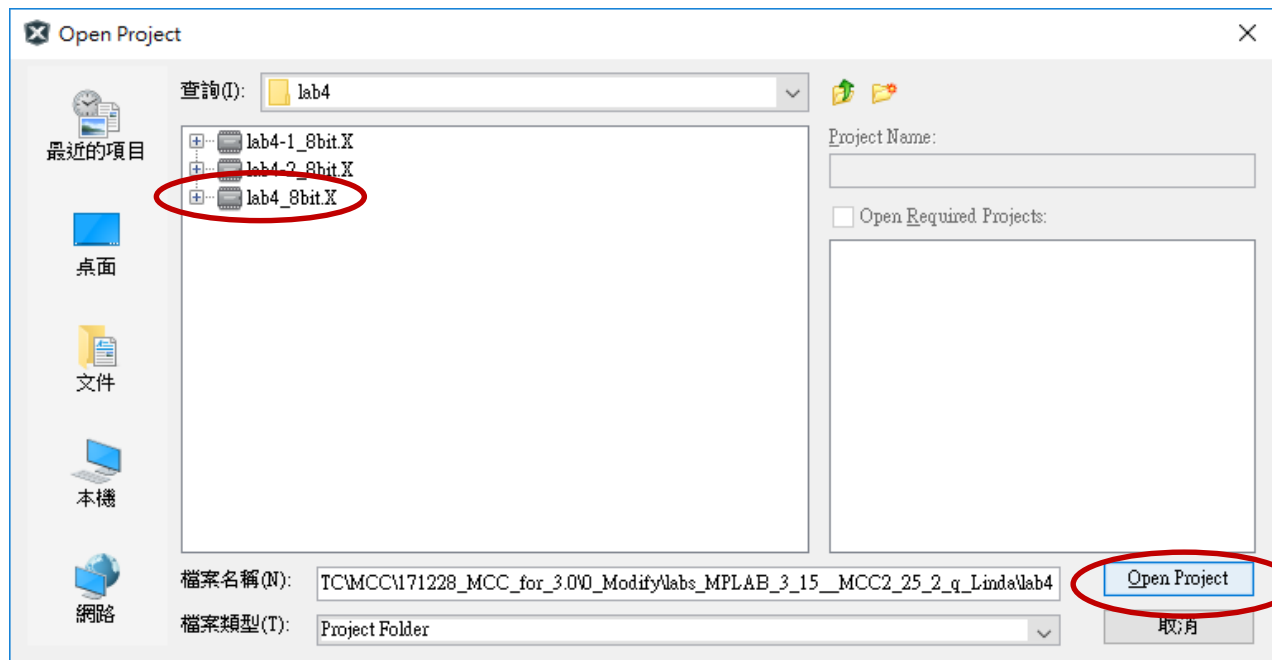
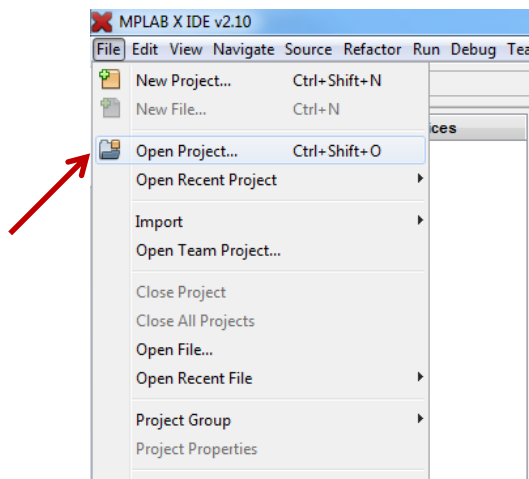
PIC18 Lab 4

- 為避免混淆，請確定關閉所有的專案及編輯視窗裡的檔案



PIC18 Lab 4

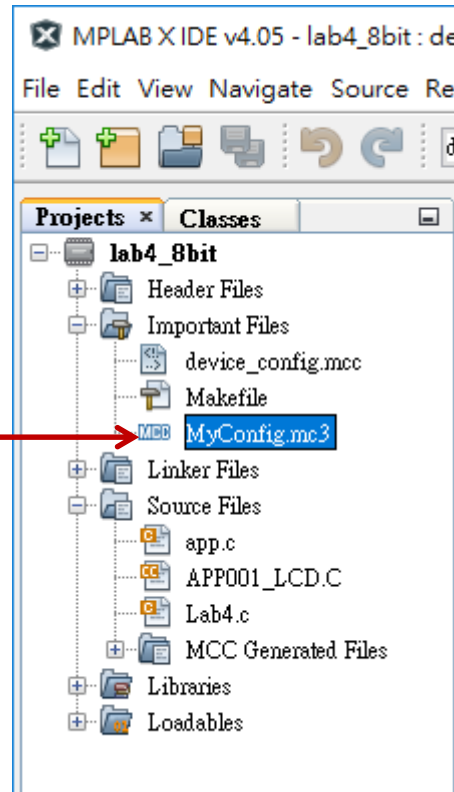
- 開啟 ..\Labs\Lab4\lab4_8bit.X



PIC18 Lab 4

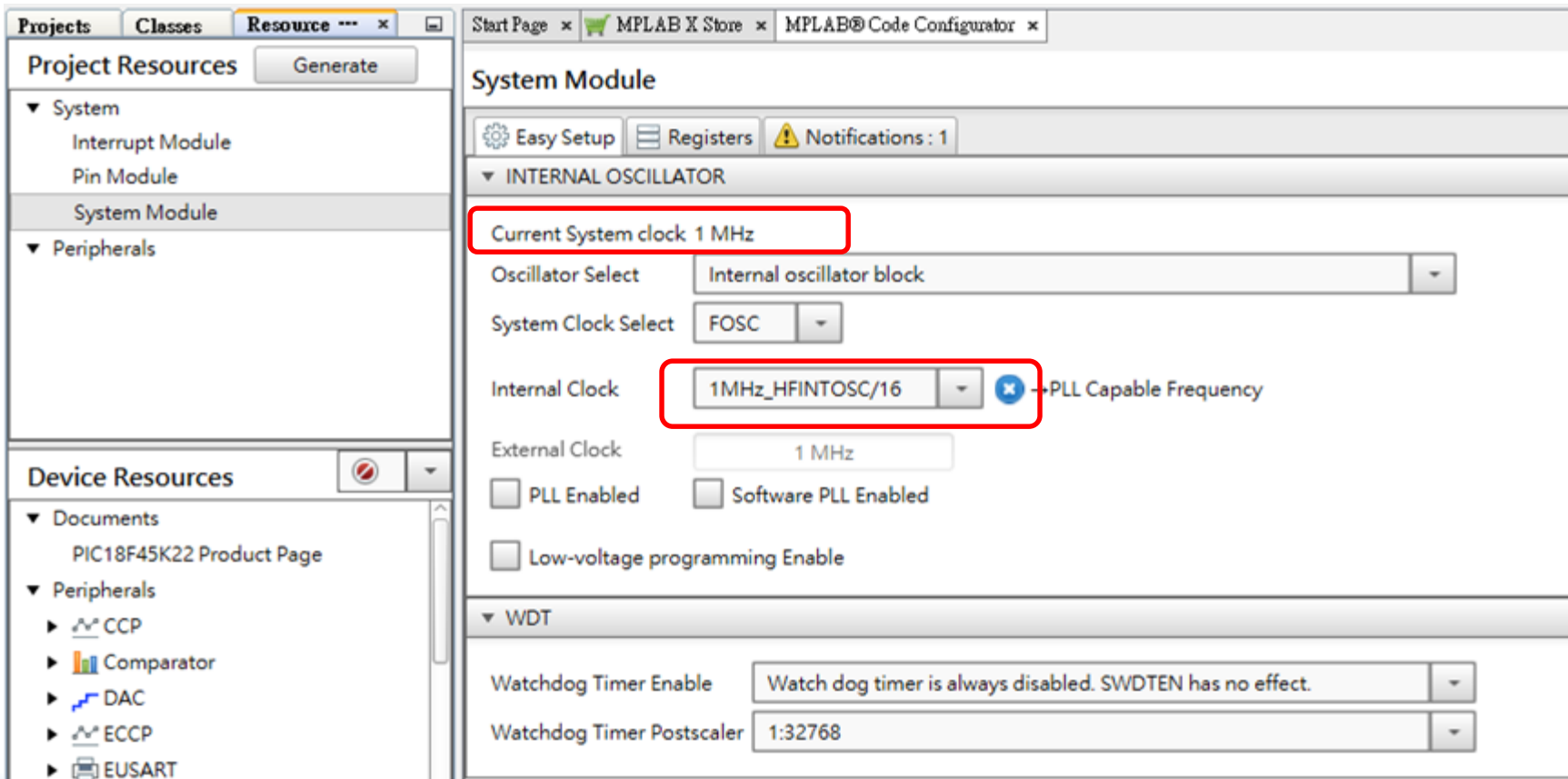
- 啟用 MPLAB® Code Configurator

“double
click”



檢視MCC 的設定 System




- 使用內建 1MHz 的 RC 振盪器



The screenshot displays the MPLAB X IDE interface with the MPLAB Code Configurator (MCC) open. The 'System Module' tab is selected, showing the 'INTERNAL OSCILLATOR' configuration. The 'Current System clock' is set to 1 MHz. The 'Oscillator Select' is 'Internal oscillator block'. The 'System Clock Select' is 'FOSC'. The 'Internal Clock' is set to '1MHz_HFINTOSC/16', which is highlighted with a red box. The 'External Clock' is set to '1 MHz'. The 'PLL Enabled' and 'Software PLL Enabled' checkboxes are unchecked. The 'Low-voltage programming Enable' checkbox is also unchecked. The 'WDT' (Watchdog Timer) settings are visible at the bottom, with 'Watchdog Timer Enable' set to 'Watch dog timer is always disabled. SWDTEN has no effect.' and 'Watchdog Timer Postscaler' set to '1:32768'.

檢視MCC 的設定 GPIO

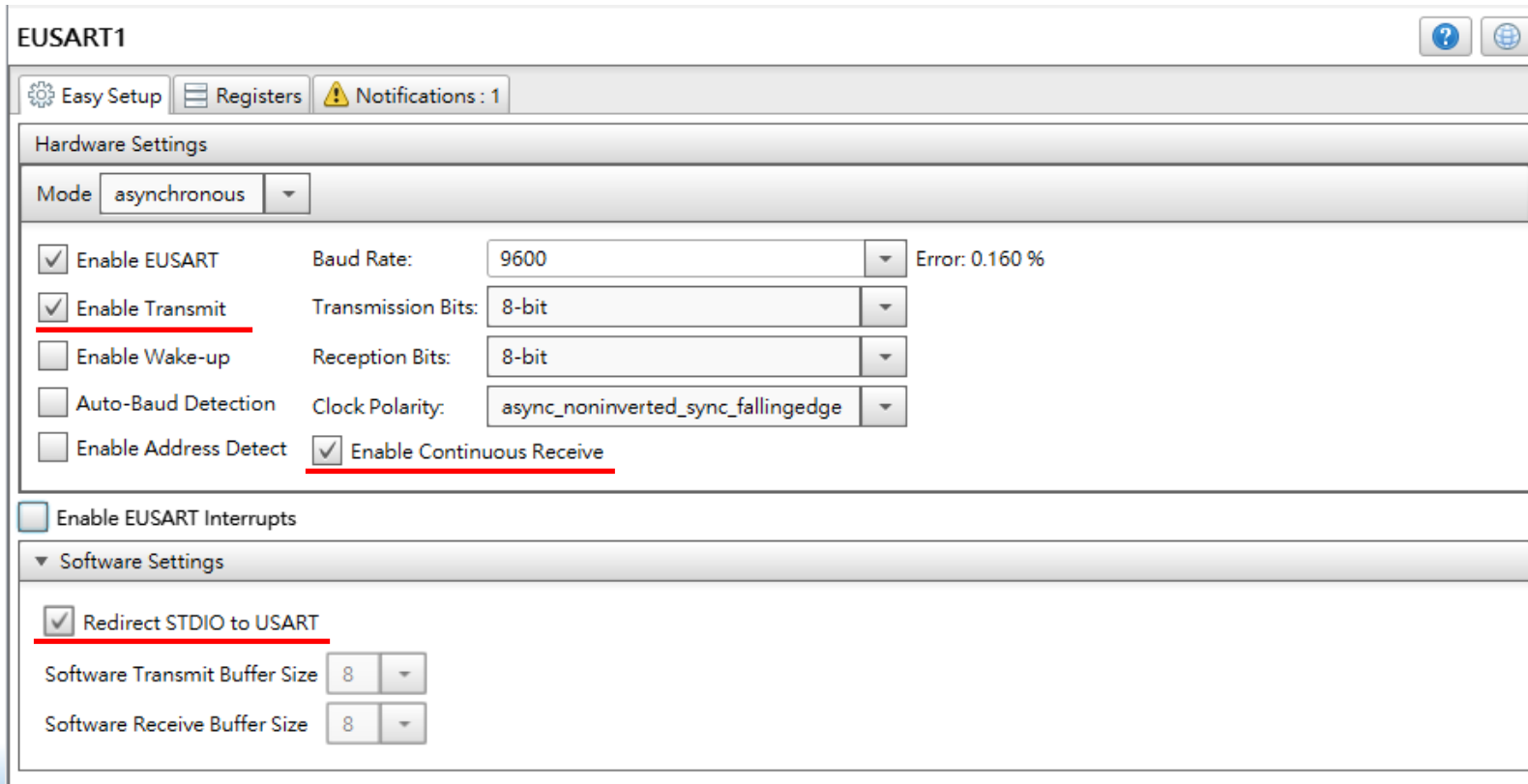
● 與 Lab3 (新增RB0=SW2,RA4=SW3)

Pin Module									
<div>  Easy Setup  Registers  Notifications : 0 </div>									
Selected Package : PDIP40									
Pin Name ▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA2	Pin Module	GPIO	LCD_E	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RA4	Pin Module	GPIO	SW3	<input type="checkbox"/>		<input type="checkbox"/>			
RB0	Pin Module	GPIO	SW2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
RC3	MSSP1	SCL1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
RC4	MSSP1	SDA1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
RC6	EUSART1	TX1		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RC7	EUSART1	RX1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
RD0	Pin Module	GPIO	LCD_DATA0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD1	Pin Module	GPIO	LCD_DATA1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD2	Pin Module	GPIO	LCD_DATA2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD3	Pin Module	GPIO	LCD_DATA3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD4	Pin Module	GPIO	LCD_RS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
RD5	Pin Module	GPIO	LCD_RW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			

檢視MCC 的設定 EUSART1

● 設定EUSART1

- ◆ 關閉中斷，開啟傳送模組，STDIO 轉向
- ◆ 9600, N, 8, 1



EUSART1

Easy Setup Registers Notifications : 1

Hardware Settings

Mode asynchronous

☒ Enable EUSART Baud Rate: 9600 Error: 0.160 %

☒ Enable Transmit Transmission Bits: 8-bit

☐ Enable Wake-up Reception Bits: 8-bit

☐ Auto-Baud Detection Clock Polarity: async_noninverted_sync_fallingedge

☐ Enable Address Detect ☒ Enable Continuous Receive

☐ Enable EUSART Interrupts

Software Settings

☒ Redirect STDIO to USART

Software Transmit Buffer Size 8

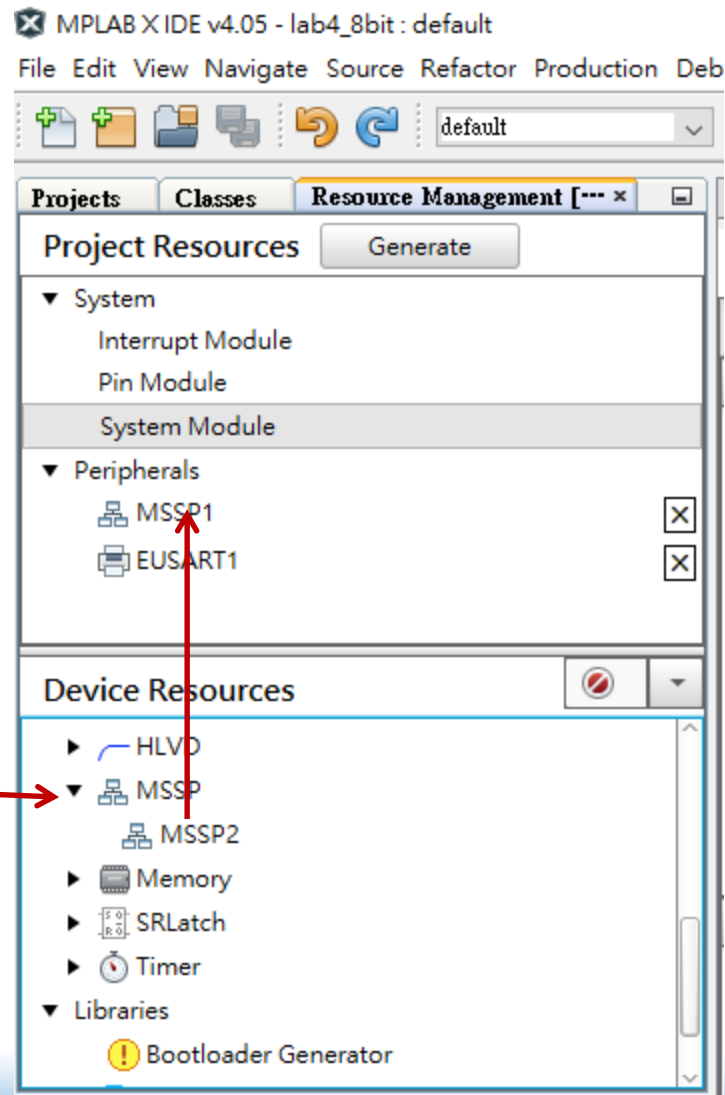
Software Receive Buffer Size 8

檢視MCC 的設定

加入I²C™ 1 模組

- 在 **Device Resources** 下選擇 **MSSP1**
- 加入**Project Resources**

雙擊 MSSP1



檢視MCC 的設定

I²C™ 1

- 設定 I²C™ 1 模組
 - ◆ 這是中斷傳輸的模式
 - ◆ 使用相容於 PIC16F 的中斷模式

Projects Classes Resource Management [--- x]

Project Resources Generate

- System
 - Interrupt Module
 - Pin Module
 - System Module
- Peripherals
 - MSSP1
 - EUSART1

“click!”

Device Resources

- HLVD
- MSSP
 - MSSP2
- Memory
- SRLatch
- Timer
- Libraries
 - Bootloader Generator

MSSP1

Easy Setup Register

Hardware Settings

Mode I2C Master

1. 選擇I²C™ Master Module

Enable MSSP

Enable SM Bus Input

Slew Rate Control High Speed

SDA Hold Time 100ns

I2C Clock

Baud Rate Generator Value 0x03 ≤ 0x3 ≤ 0xFF

I2C Clock Frequency 62.5 kHz

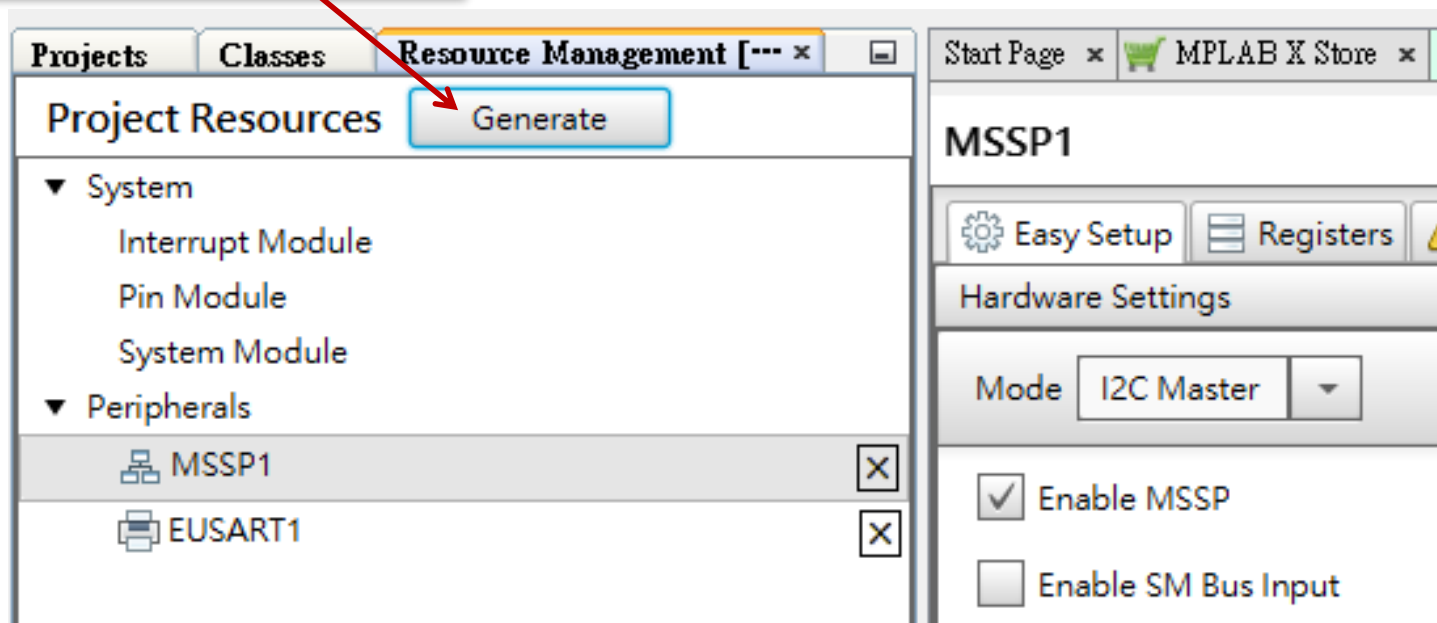
2. 填入數值調整I2C Clock頻率
· 下方會出現實際數值

Slave Address 7 bit 10 bit

完成 Lab 4 的 MCC 設定

- 按下 “Generate Code” 並關閉 MPLAB® Code Configurator

按下 “Generate” 圖示

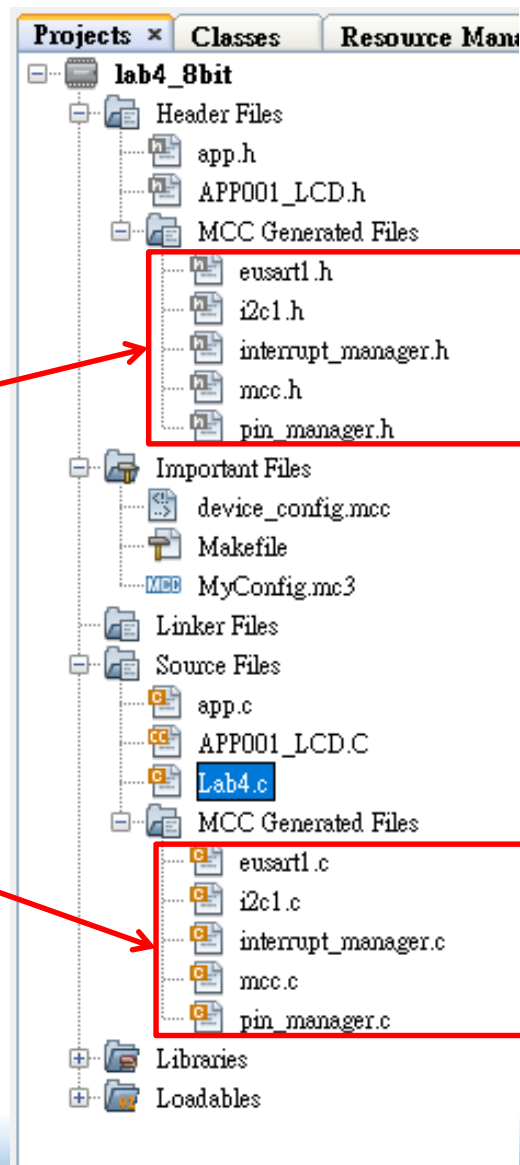


PIC18 Lab 4

- 開啟 Lab4.c

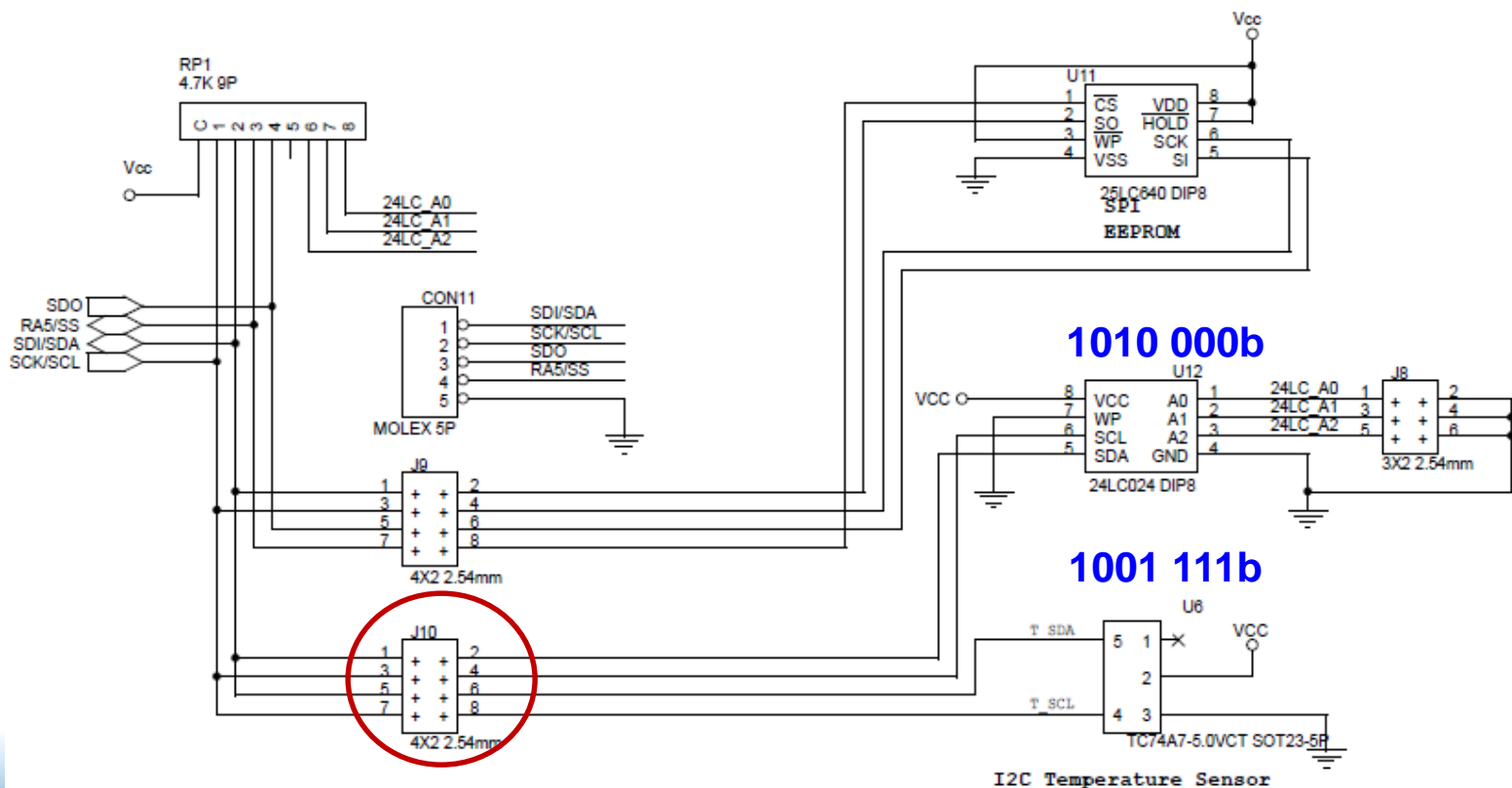
MCC 所產生的
周邊含入檔

MCC 所產生的
周邊函數

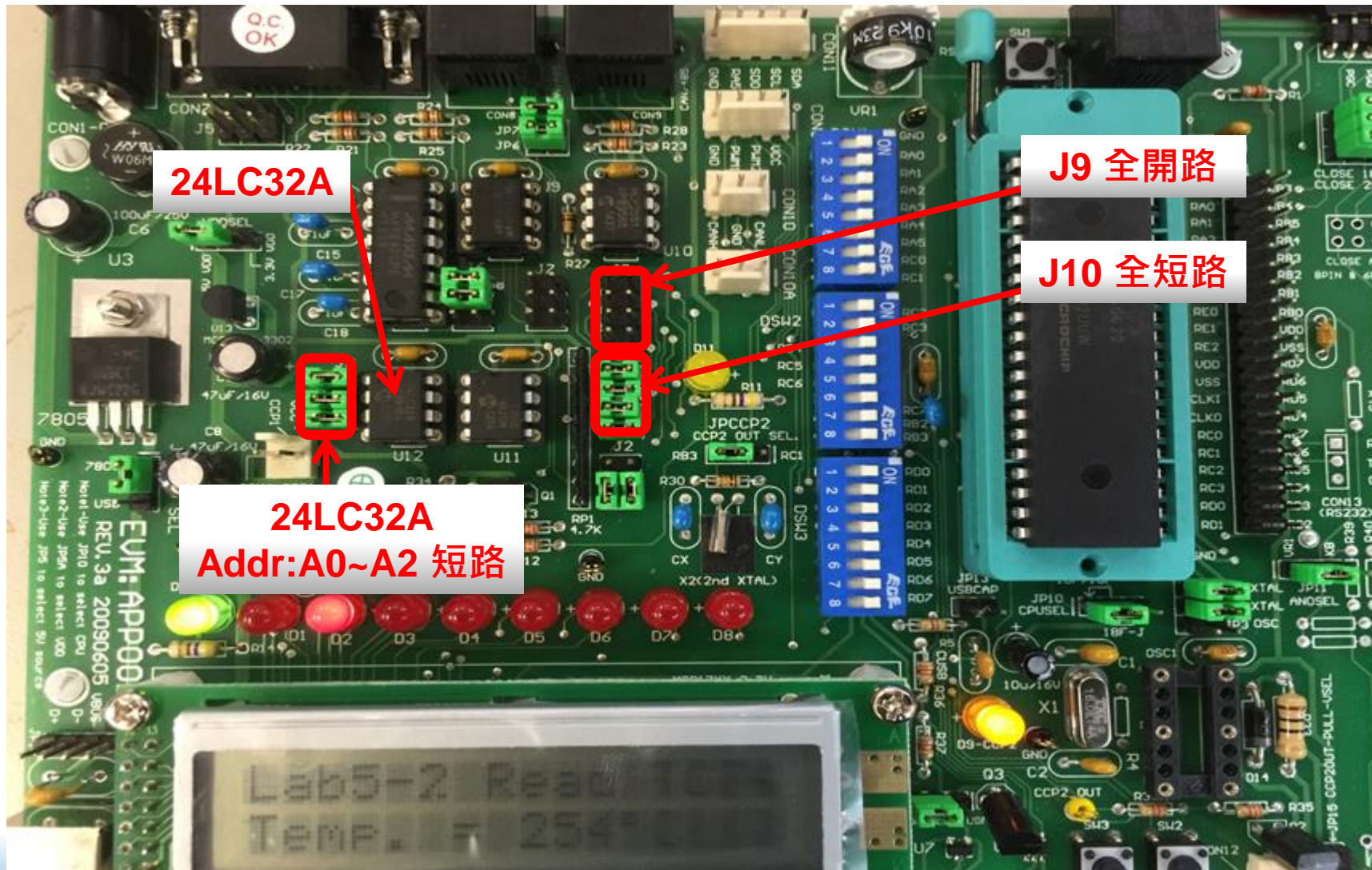


APP001 J10 的設定

- J10 用來設定 I²C™ 的元件選擇
 - ◆ 確定 J10 的 1 & 2, 3 & 4 是短路的
 - ◆ J10 全部短路也可以，因為 TC74 位址不同



APP001 Jump 的設定



Lab4 的功能說明

- 利用 **MCC** 所產生的 **I²C™** 函數，存取 **EEPROM 24LC32A** 的用法
- 按下 **SW2** 時，將預設的資料寫到 **24LC32A 0x0000** 的位址裡
 - ◆ 預設資料：**GOOD DAY !**
- 按下 **SW3** 時，讀取 **24LC32A 0x0000** 位址的連續資料並做比較
- 顯示比較的結果

Lab4 程式說明

- **Lab4.c – User's** 自己寫的主程式
 - ◆ 主程式，呼叫 MCC 所產生的 I²C™ 函數做 24LC32A 的儲存，讀取，比對
- **App.c – User's** 自己寫的應用程式
 - ◆ 按鍵彈跳處理，回傳按鍵值給 main()
- **I2C1.c – MCC** 所產生的函數
 - ◆ I2C1_Initialize() : I²C™ 初始設定 (中斷模式)
 - ◆ I2C1_ISR() : I²C™ 中斷傳送函數
 - ◆ I2C1_Master_Write() : I²C™ 寫入函數
 - ◆ I2C1_Master_Read() : I²C™ 讀取函數

Lab4 對 24LC32A 的宣告

- **#define I2C_SLAVE_ADDRESS 0x50**
 - ◆ 定義 24LC32A 的 Slave Address,
 - ◆ 實際位址 : 0b1010 000 (7-Address bits) + R/W (1 bit)
 - ◆ 程式會向左移一個位元再補上 R/W 位元
 - ◆ 存取有 Block 的元件, 0x51 會存取 Block1 (24LC04)
- **#define EEPROM_ADDRESS_HIGH 0x00**
- **#define EEPROM_ADDRESS_LOW 0x00**
 - ◆ 定義要存取的 24LC32A 的資料位址
 - ◆ 24LC32A 以上的 EEPROM 有 16 bits 位址
- **#define NUM_ADDRESS_BYTES 0x02**
 - ◆ 定義為 24LC32A 以上的元件位址
 - ◆ 如存取 24LC16 以下的元件則設成 0x01

I2C1_Master_Write()

I²C™ 寫入函數的參數說明

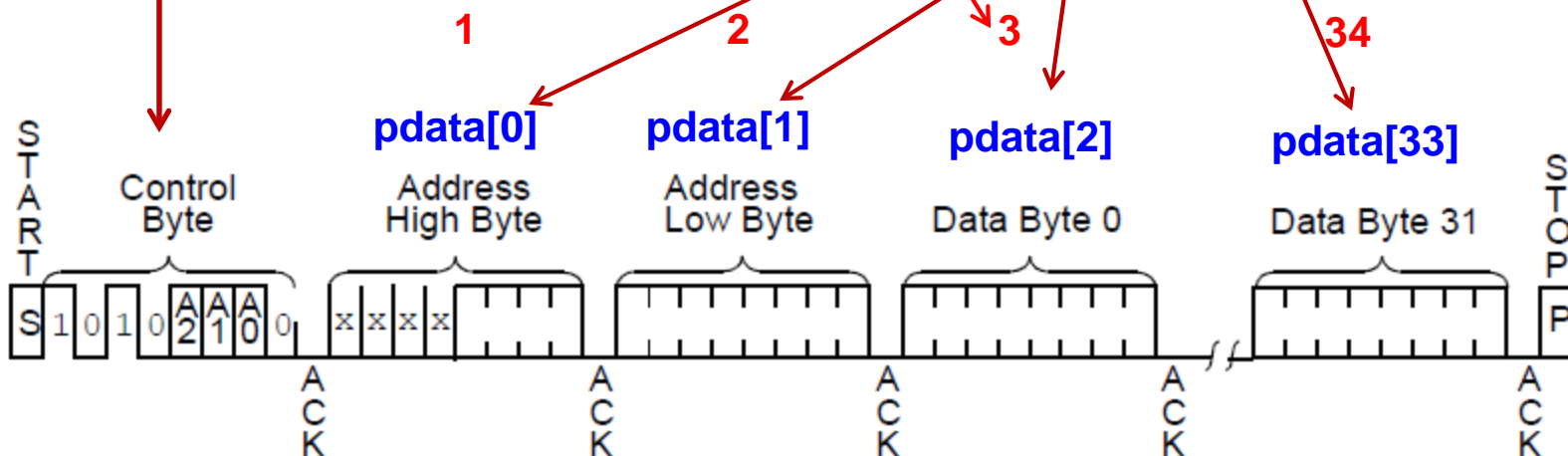
void I2C1_MasterWrite(

uint8_t **pdata*, // 寫入資料的陣列 (含位址)

uint8_t *length*, // 寫入資料的長度 (含位址)

uint16_t *address*, // Slave Device Address

I2C1_MESSAGE_STATUS **pstatus*);



24LC32A Page Write

Lab4.c

使用 I2C1_MasterWrite () 的範例

- **#define I2C_SLAVE_ADDRESS 0x50**
- **uint8_t eepromWriteBuffer[] =**
{EEPROM_ADDRESS_HIGH,
EEPROM_ADDRESS_LOW, 'G', 'O', 'O', 'D', ' ', 'D', 'A',
'Y', ' ', '!'}; // 共 12 Bytes 資料
- **I2C1_MasterWrite(**
eepromWriteBuffer, // 欲寫入的資料的指標
sizeof(eepromWriteBuffer), // { } 內資料總共 12 個
I2C_SLAVE_ADDRESS, // 0xA0 , 0x50 << 1
&i2cStatus); // 函數工作結束的狀態

關於寫入函數的狀態

i2cStatus

```
I2C1_MESSAGE_STATUS i2cStatus;    // Lab4.c
```

```
typedef enum {                                // 宣告在 i2C1.h 裡
    I2C1_MESSAGE_COMPLETE,
    I2C1_MESSAGE_FAIL,
    I2C1_MESSAGE_PENDING,
    I2C1_STUCK_START,
    I2C1_MESSAGE_ADDRESS_NO_ACK,
    I2C1_DATA_NO_ACK,
    I2C1_LOST_STATE
} I2C1_MESSAGE_STATUS;

// I2C 狀態的列舉宣告
```

i2cStatus 的使用範例

寫入一個 Byte

```
timeOut = 0;
while(status != I2C1_MESSAGE_FAIL)
{
    I2C1_MasterWrite( writeBuffer, 3, slaveDeviceAddress, &i2cstatus);

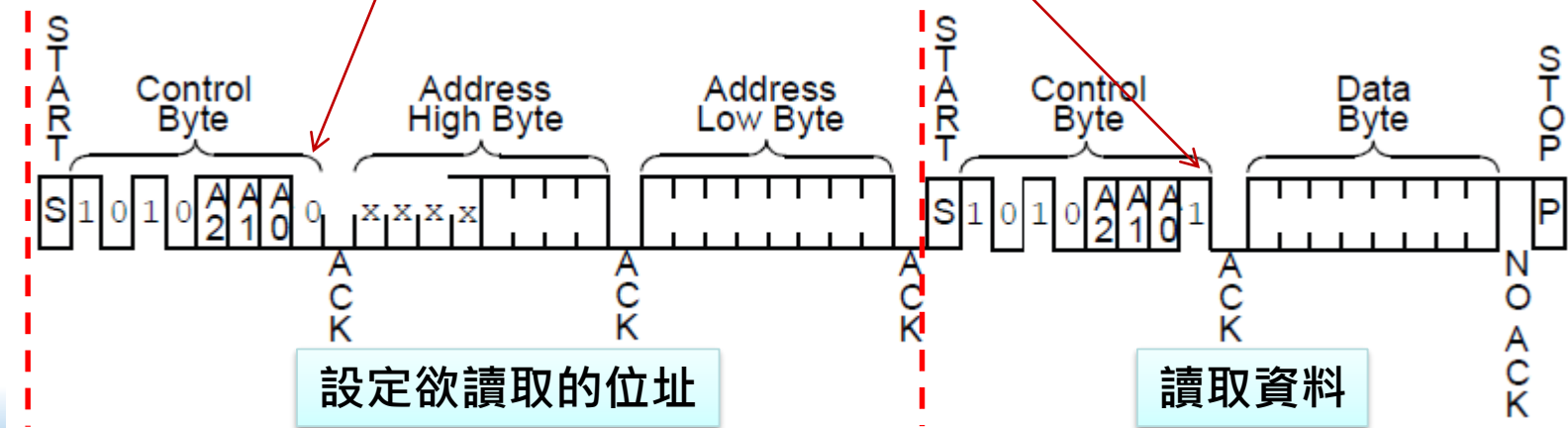
    while(i2cstatus == I2C1_MESSAGE_PENDING); // 等待寫入的狀態改變
    if (i2cstatus == I2C1_MESSAGE_COMPLETE) break; // 寫入動作完成，跳離此迴圈

    // 假如回傳的狀態為 I2C1_MESSAGE_ADDRESS_NO_ACK 或 I2C1_DATA_NO_ACK,
    // 表示元件還在忙，需要再做寫入直到 設定的 Time-Out 次數

    if (timeOut == SLAVE_I2C_GENERIC_RETRY_MAX) break; // Time-Out 發生，跳離迴圈
    else
        timeOut++; // Slave 沒有回應，Time-Out 計數值加一
}
if (i2cstatus == I2C1_MESSAGE_FAIL)
{
    break; // 寫入錯誤，離開迴圈
}
```

24LC32A Random Read

- 先用 **Write Command** 設定 EEPROM 要讀取的位址
 - ◆ 如要用 Sequential Read 則要注意邊界位址
- 再用 **Read Command** 讀取資料
 - ◆ 可以只讀取一個 Byte
 - ◆ 也可以讀取小於一個 Page 的資料 (32 Bytes)



Lab4.c 的定義

讀取 24LC32A

#define I2C_SLAVE_ADDRESS 0x50 // 24LC32A 的 Slave 位址

#define EEPROM_ADDRESS_HIGH 0x00 // 24LC32A 要讀取資料

#define EEPROM_ADDRESS_LOW 0x00 // 的內部位址設定

#define NUM_ADDRESS_BYTES 0x02 // 兩個 Byte 的位址 (容量)

**uint8_t eepromReadBuffer [sizeof(eepromWriteBuffer) -
NUM_ADDRESS_BYTES] = {0x00};**

// 定義 eepromReadBuffer 陣列的大小，此處為 $12 - 2 = 10$ 個

// 陣列初始值設為 0x00

Lab4 的 24LC32A 資料讀取

- 寫入位址的設定:

```
I2C1_MasterWrite(eepromWriteBuffer, NUM_AD  
DRESS_BYTES, I2C_SLAVE_ADDRESS, &i2cStat  
us);
```

NUM_ADDRESS_BYTES = 2, 只設定 16-bit 的位址

- 讀取 24LC32A 的資料

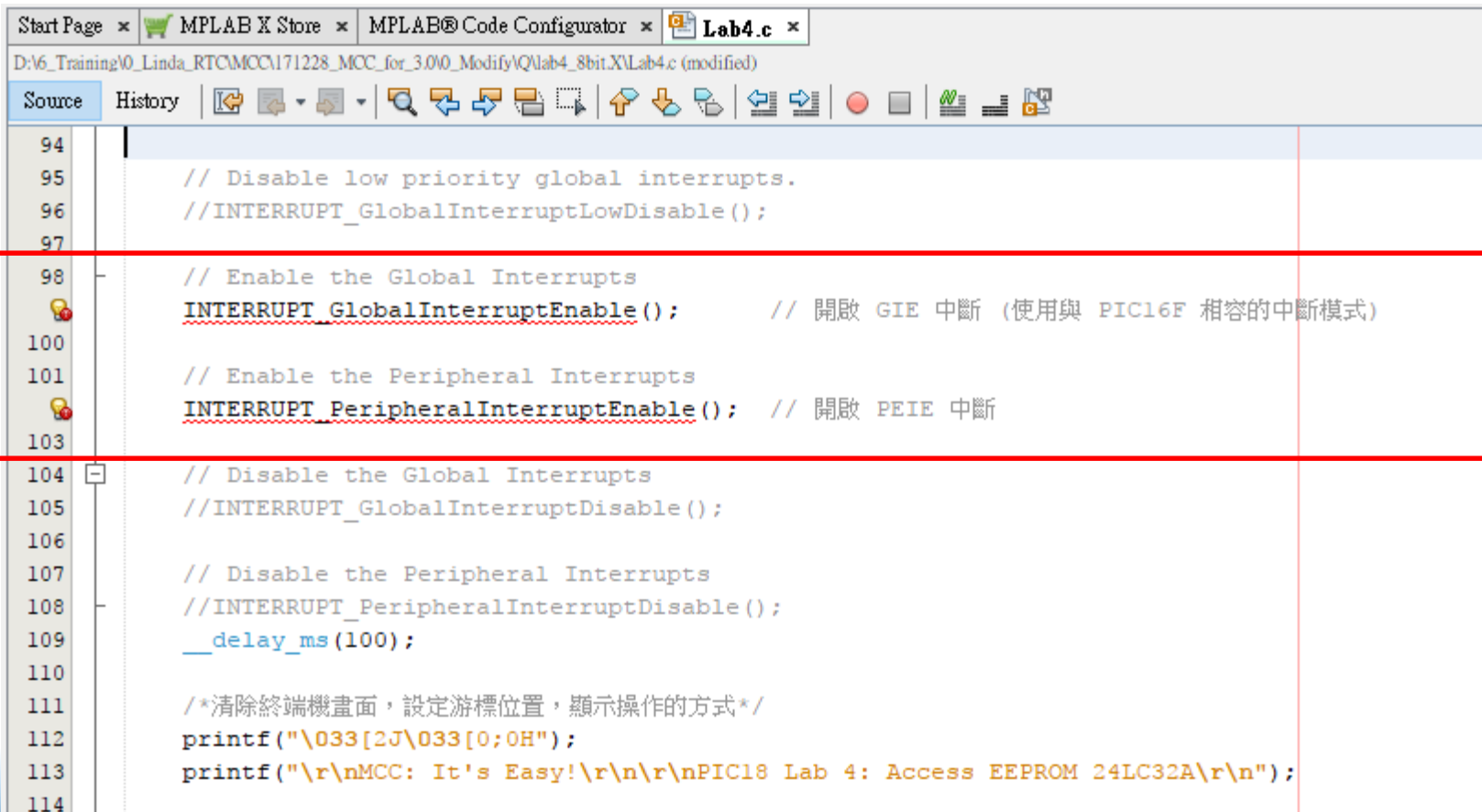
```
I2C1_MasterRead(eepromReadBuffer, sizeof(ee  
promReadBuffer), I2C_SLAVE_ADDRESS,  
&i2cStatus);
```

sizeof(eepromReadBuffer) = 10,
連續讀取 10 Bytes 資料到 eepromReadBuffer[]

PIC18 Lab 4

● 編輯 Lab4.c

- ◆ 此處MCC產生的I2C的函式是由中斷方式撰寫，所以必須要確認已開啟中斷



```
Start Page x MPLAB X Store x MPLAB® Code Configurator x Lab4.c x
D:\6_Training\0_Linda_RTC\MCC171228_MCC_for_3.0\0_Modify\QLab4_8bit.X\Lab4.c (modified)
Source History
94
95 // Disable low priority global interrupts.
96 //INTERRUPT_GlobalInterruptLowDisable();
97
98 // Enable the Global Interrupts
99 INTERRUPT_GlobalInterruptEnable(); // 開啟 GIE 中斷 (使用與 PIC16F 相容的中斷模式)
100
101 // Enable the Peripheral Interrupts
102 INTERRUPT_PeripheralInterruptEnable(); // 開啟 PEIE 中斷
103
104 // Disable the Global Interrupts
105 //INTERRUPT_GlobalInterruptDisable();
106
107 // Disable the Peripheral Interrupts
108 //INTERRUPT_PeripheralInterruptDisable();
109 __delay_ms(100);
110
111 /*清除終端機畫面，設定游標位置，顯示操作的方式*/
112 printf("\033[2J\033[0;0H");
113 printf("\r\nMCC: It's Easy!\r\n\r\nPIC18 Lab 4: Access EEPROM 24LC32A\r\n");
114
```

PIC18 Lab 4

- 編輯 Lab4.c
- 一樣在 Window 下的 Tasks 有三個 TODOs

Call Graph		Output
	Description ▲	File
	TODO: 將陣列 eepromReadBuffer的資料，...	Lab5.c
	TODO: 將陣列 eepromWriteBuffer的資料，...	Lab5.c
	TODO: 將陣列 eepromWriteBuffer的資料，...	Lab5.c

```

131 //將資料寫到 24LC32A EEPROM
132 //TODO: 將陣列 eepromWriteBuffer的資料，寫入資料的長度，24LC32A Slave Address 及狀態回傳變數 (共四項)
133 //      使用 I2C1_MasterWrite( ) 函數來實現資料讀寫入
134 I2C1_MasterWrite(eepromWriteBuffer,sizeof(eepromWriteBuffer),I2C_SLAVE_ADDRESS,&i2cStatus);

```

```

151 //將欲讀取資料的位址寫到 24LC32A EEPROM
152 //TODO: 將陣列 eepromWriteBuffer的資料，幾個位址長度 (2)，24LC32A Slave Address 及狀態回傳變數 (共四項)
153 //      使用 I2C1_MasterWrite( ) 函數來實現位址的設定
154 I2C1_MasterWrite(eepromWriteBuffer,NUM_ADDRESS_BYTES,I2C_SLAVE_ADDRESS,&i2cStatus);

```

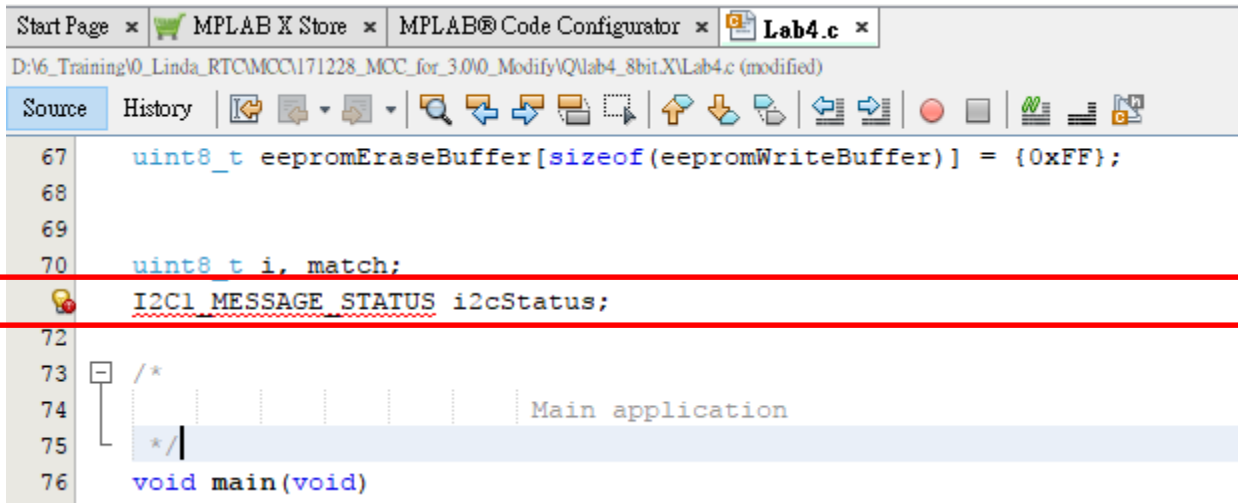
```

162 //依據所輸入的讀取長度來讀取 EEPROM 的資料
163 //TODO: 將陣列 eepromReadBuffer的資料，讀取資料個數，24LC32A Slave Address 及狀態回傳變數 (共四項)
164 //      使用 I2C1_MasterRead( ) 函數來實現資料的讀取
165 I2C1_MasterRead(eepromReadBuffer,sizeof(eepromReadBuffer),I2C_SLAVE_ADDRESS, &i2cStatus);

```

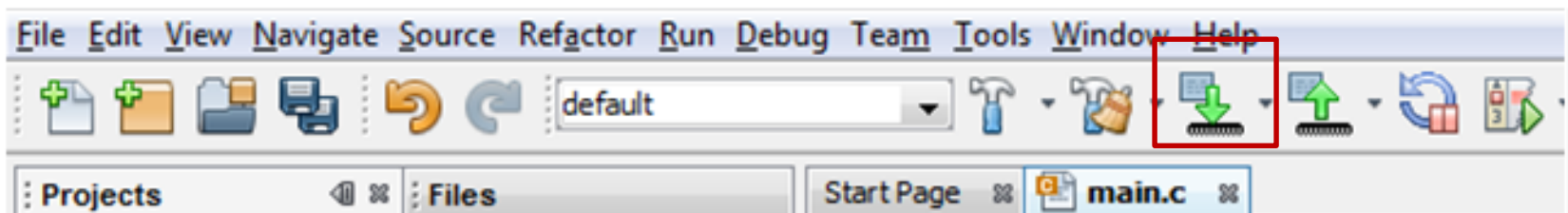
PIC18 Lab 4

- 確認宣告I2Cx_MESSAGE_STATUS承接I2C函數產生的狀態



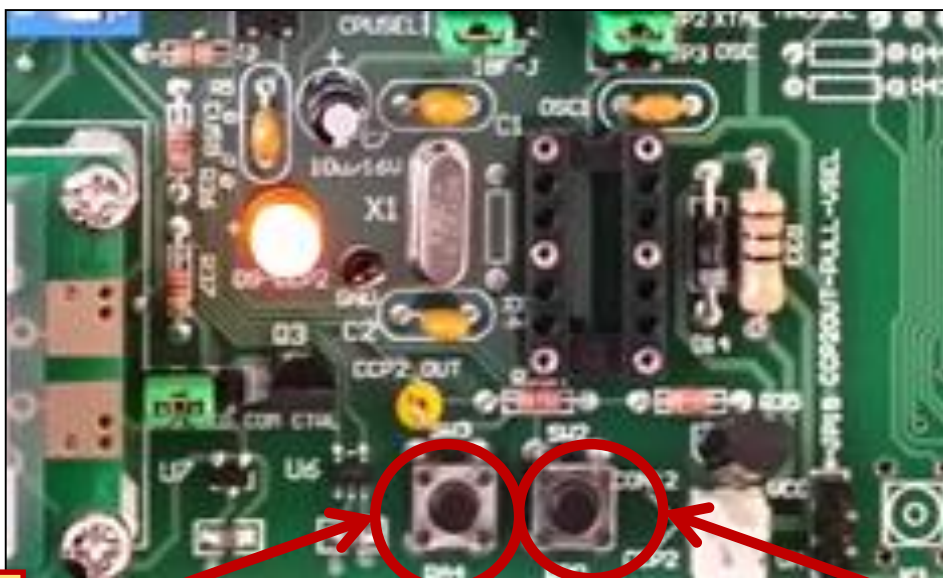
```
Start Page x MPLAB X Store x MPLAB® Code Configurator x Lab4.c x
D:\6_Training\0_Linda_RTC\MCC\171228_MCC_for_3.0\0_Modify\Q\lab4_8bit.X\Lab4.c (modified)
Source History
67 uint8_t eeepromEraseBuffer[sizeof(eepromWriteBuffer)] = {0xFF};
68
69
70 uint8_t i, match;
71 I2C1_MESSAGE_STATUS i2cStatus;
72
73 /*
74 | Main application
75 */
76 void main(void)
```

- “編譯及燒錄” 程式



Lab 4 程式執行

- 按下 **SW2** 執行 **EEPROM** 寫入動作
- 按下 **SW3** 執行 **EEPROM** 讀取及比對

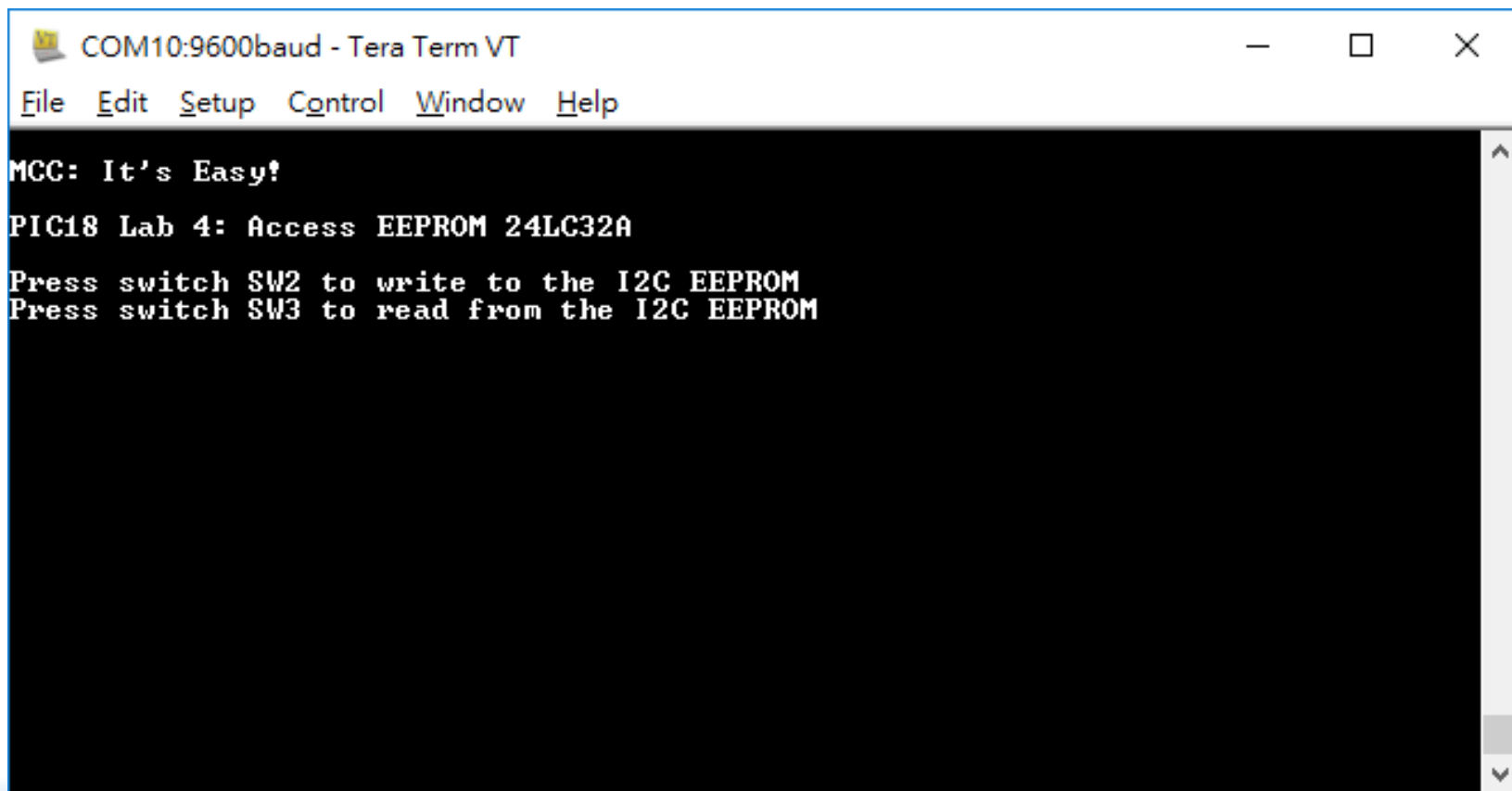


EEPROM
讀取

EEPROM
寫入

Lab 4 程式執行

- Tera Term 顯示 Lab 4 歡迎訊息



COM10:9600baud - Tera Term VT

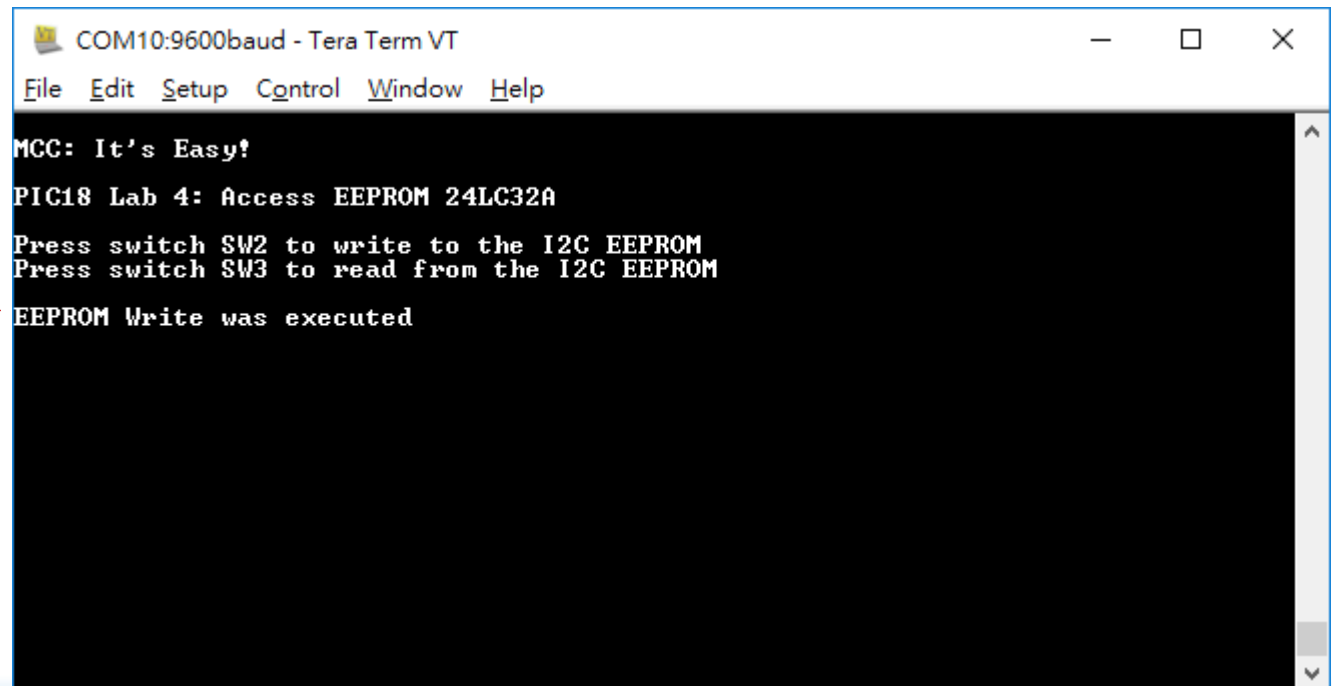
File Edit Setup Control Window Help

```
MCC: It's Easy!  
PIC18 Lab 4: Access EEPROM 24LC32A  
Press switch SW2 to write to the I2C EEPROM  
Press switch SW3 to read from the I2C EEPROM
```

PIC18 Lab 4

- 按下 **SW2** 將陣列資料寫到 **EEPROM**
 - ◆ 寫入資料為 “**GOOD Day !**”

按下 SW2 的顯示



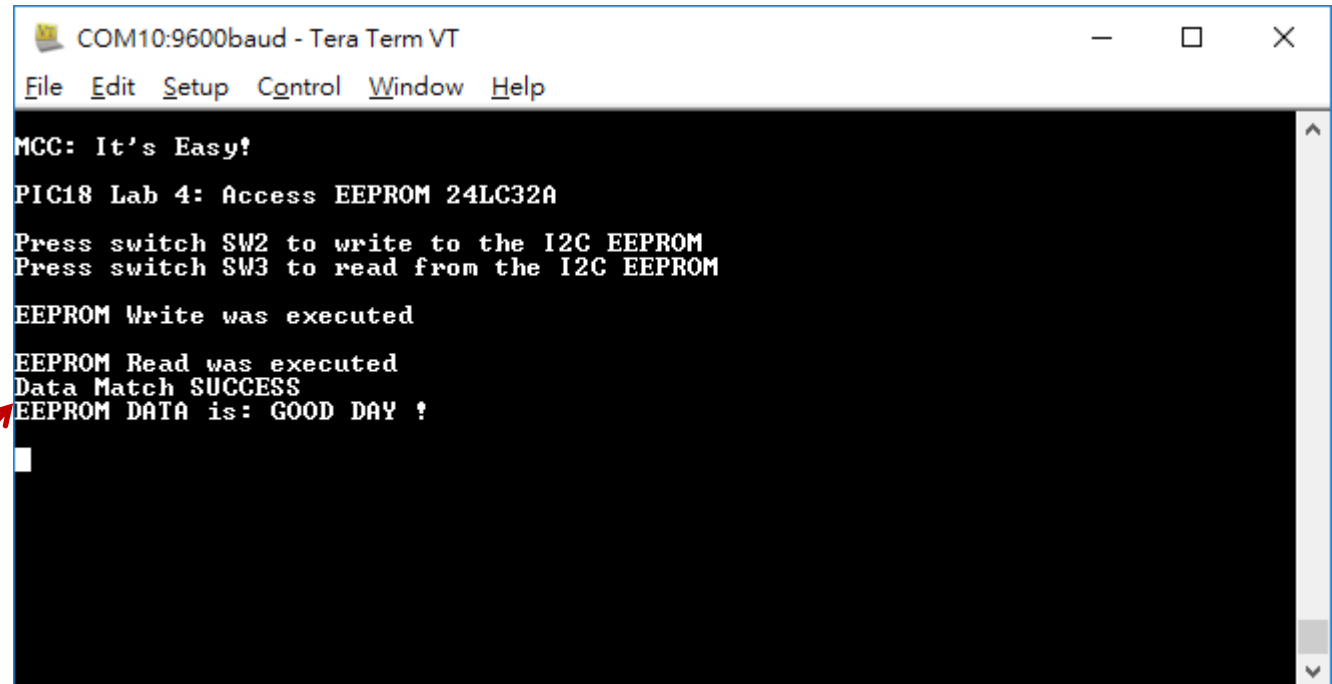
```
COM10:9600baud - Tera Term VT
File Edit Setup Control Window Help

MCC: It's Easy!
PIC18 Lab 4: Access EEPROM 24LC32A
Press switch SW2 to write to the I2C EEPROM
Press switch SW3 to read from the I2C EEPROM
EEPROM Write was executed
```

PIC18 Lab 4

- 按下 **SW3** 後，讀取 **EEPROM** 的內容值並做比對，正確後顯示訊息

按下 **SW3** 後的
顯示訊息



```
COM10:9600baud - Tera Term VT
File Edit Setup Control Window Help

MCC: It's Easy!
PIC18 Lab 4: Access EEPROM 24LC32A
Press switch SW2 to write to the I2C EEPROM
Press switch SW3 to read from the I2C EEPROM

EEPROM Write was executed
EEPROM Read was executed
Data Match SUCCESS
EEPROM DATA is: GOOD DAY !
```




MICROCHIP

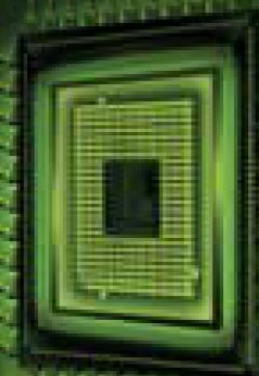
Regional Training Centers

PIC18 Lab 4-1:

I²C™ 串列通訊

用 Master 模式存取 24LC02B

(附加練習, 需要置換板上EEPROM)



我要如何改成其它 I²C™ 元件

- **Lab4.c** 的範例是用來存取 **24LC32A** 等 **16-bit** 位址的元件
- 如果我用的都只有 **8-bit Address** 元件或其他的 **I²C™** 元件呢
 - ◆ 例如:
 - **24LC02B**
 - **TC74** 溫度感應器

Lab4-1 的注意事項

- **Lab4-1 須將 APP001 的U12 (EEPROM)換成 24LC02B 或 24LC04B**
- **練習是將 Lab4 的 Page Write/Read 改成 Byte Write/Read 的方式**
- **MCC設定相同**

關於 24LC02B 的寫入

- **Page Write Buffer : 8 Bytes**

- ◆ Page Write 有起始位址的邊界

- **0bxxxxx000** 也就是必須是 **b2~b0** 都為零
 - 必須自起點算起才有 **8 Bytes Buffer**
 - 如果從位址 **0x33** 做 Page Write 會只有 **0x33 ~ 0x37** 共 **5 Bytes** 資料寫入，其餘資料資料將會遺失。

- **Byte Write** 可以隨意設定寫入的位址，也可以再繼續寫入下一筆資料直到邊界。

關於 24LC02B 的讀取

- **Current Address Read**

- ◆ 依目前 EEPROM 內部設定位址讀取一個 Byte 資料
- ◆ 依目前 EEPROM 內部設定位址連續讀取資料，直到邊界

- **Random Address Read**

- ◆ 先寫入欲讀取 EEPROM 的位址
- ◆ 再送出 Current Address Read 命令

關於 24LC02B 的讀取

- **Sequential Read Buffer : 8 Bytes**

- ◆ Sequ. Read 也有起始位址的邊界

- **0bxxxxx000** 也就是必須是 **b2~b0** 都為零
- 必須自起點算起才有 **8 Bytes Buffer**
- 位址自動加一直到邊界後會歸零，不會進位

例如：從位址 **0x00 ~ 0x0F** 分別設定資料 **0x00 ~ 0x0F**
，位址從 **0x05** 開始連續讀取 **10 Bytes**

所讀取到的資料如下：

0x05, 0x06, 0x07, 0x08, 0x00, 0x01, 0x02, 0x03, 0xFF, 0xFF

位址歸零

無法讀取

如何存取 24LC02B

Lab4-1.c

- **24LC32 以上的 EEPROM 需要有 16-bit Address 來存取所有的資料**
 - ◆ #define EEPROM_ADDRESS_HIGH 0x00
 - ◆ #define EEPROM_ADDRESS_LOW 0x00
 - ◆ #define NUM_ADDRESS_BYTES 0x02
- **24LC02 EEPROM 只需 8-bit Address 來存取資料**
 - ◆ // #define EEPROM_ADDRESS_HIGH 0x00
 - ◆ #define EEPROM_ADDRESS_LOW 0x00
 - ◆ #define NUM_ADDRESS_BYTES 0x01

24LC02B Byte Write

Lab4-1.c

- **Byte Write** 只要縮減/修改 `eeepromWriteBuffer[]`

```
uint8_t eeepromWriteBuffer[ ] = { 0x30, 0x61, '2', '3',  
'4','5','6','7','8'};
```

寫入 EEPROM 的位址

欲寫入 EEPROM 的資料

範例:

想在 **EEPROM 0x30** 位址存入 **0x61** 的資料

```
eeepromWriteBuffer[0] = 0x30; // 設定寫入位址
```

```
eeepromWriteBuffer[1] = 0x61; // 設定寫入資料
```

```
I2C1_MasterWrite(eeepromWriteBuffer, // 陣列的指標
```

```
2, // 第一為位址，第二個 Byte 為寫入資料
```

```
I2C_SLAVE_ADDRESS, // 0x50 << 1 = 0xA0
```

```
&i2cStatus); // 寫入狀態回報
```


24LC02B Byte Read

- 設定讀取位址，再寫入位址設

`eeepromWriteBuffer[0] = 0x30; // 設定寫入位址`

`I2C1_MasterWrite(eepromWriteBuffer, 1,
I2C_SLAVE_ADDRESS, &i2cStatus);`

// 因為只有一個 **EEPROM** 位址須設定，所以要設為 1

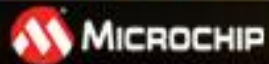
- 再用 **Sequential Read**

`I2C1_MasterRead(eepromReadBuffer, 1 ,
I2C_SLAVE_ADDRESS, &i2cStatus);`

- 這時 **EEPROM** 位址 **0x30** 的資料就會被讀到 `eeepromReadBuffer[0]` 裡

Reduce your development time • Reuse your code • Scale up or down

ONE DEVELOPMENT ENVIRONMENT



ONE PIC[®] MCU PLATFORM



動手做實驗

PIC18 Lab 4-1

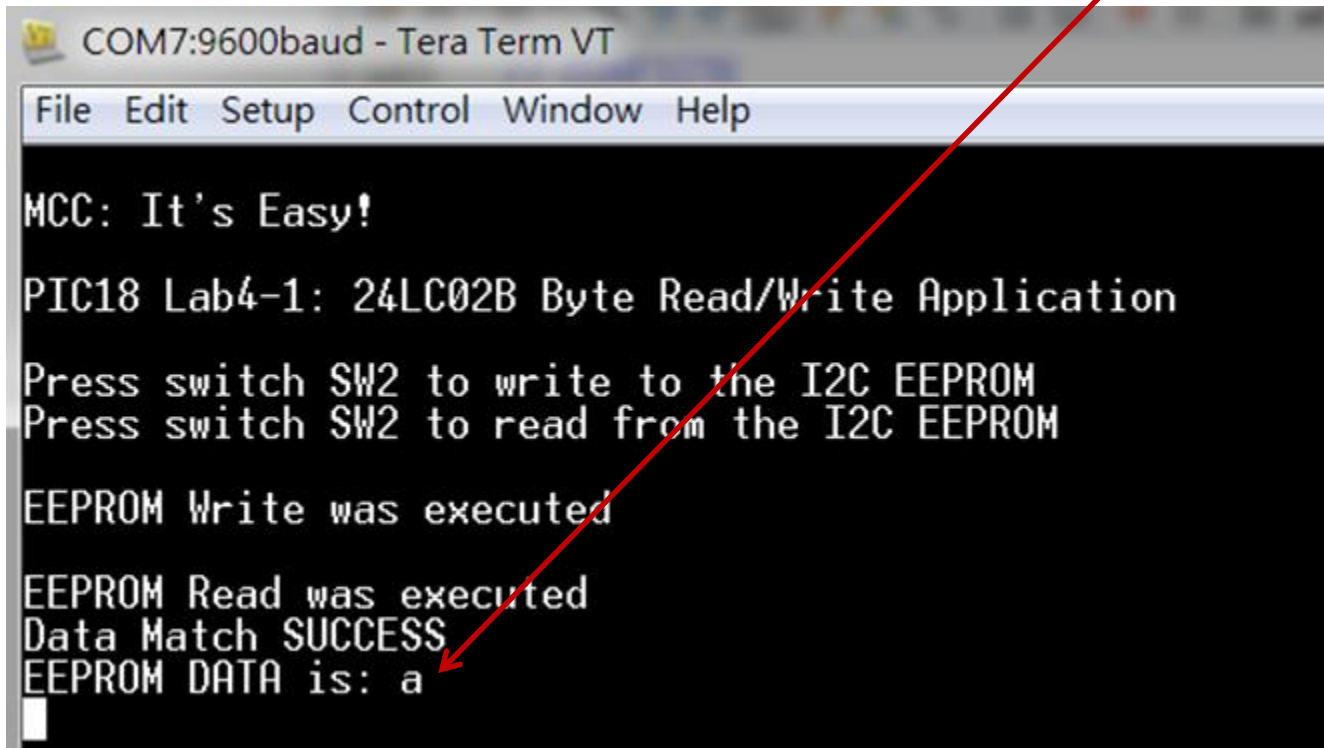
執行 Lab 4-1

- 開啟 **LAB4-1** 專案
 - ◆ 該專案的範例: 一樣使用 **Lab4** 裡的 **MCC** 所產生的周邊函數
 - ◆ 改用 **Byte Write** 及 **Byte Read** 的方式來讀取 **24LC02B**
 - ◆ 確定 **APP001** 將 **24LC32A** 換成 **24LC02B**
 - ◆ 訊息顯示在 終端機及 **LCD** 上

Lab4-1 結果

- 終端機的顯示

- ◆ 讀取 **EEPROM** 位址 **0x30** 的值為 'a' (**0x61**)



```
COM7:9600baud - Tera Term VT
File Edit Setup Control Window Help

MCC: It's Easy!
PIC18 Lab4-1: 24LC02B Byte Read/Write Application
Press switch SW2 to write to the I2C EEPROM
Press switch SW2 to read from the I2C EEPROM
EEPROM Write was executed
EEPROM Read was executed
Data Match SUCCESS
EEPROM DATA is: a
```

24LC02B Page Write

(附加資料)

- 用 Page Write 寫入 8 Bytes 資料，起始位址為：
EEPROM_ADDRESS_LOW

```
uint8_t eepromWriteBuffer[ ] = {  
EEPROM_ADDRESS_LOW, '1', '2', '3', '4', '5', '6', '7', '8'};
```

寫入 EEPROM 的位址

寫入 EEPROM 的資料

```
I2C1_MasterWrite(eepromWriteBuffer,    // 陣列的指標  
                  sizeof(eepromWriteBuffer), // 多少資料要寫入  
                  I2C_SLAVE_ADDRESS,    // 0x50 << 1 = 0xA0  
                  &i2cStatus);           // 寫入狀態回報
```

24LC02B Page Read

(附加資料)

● Page Read

- ◆ 使用 Write Address 方式設定 EEPROM 讀取的起始位址
- ◆ 再用 Sequ. Read 讀取資料直到邊界 (最多 8 Bytes) 且要注意起始的位址 ($b_2 \sim b_0 = 0$)

EEPROM 讀取陣列的宣告

```
uint8_t eepromReadBuffer[sizeof(eepromWriteBuffer)-  
NUM_ADDRESS_BYTES] = {0x00}; // 寫入的空間大小減去位  
址空間，此處為 8 Bytes
```

24LC02B Page Read

(附加資料)

- 先做 **Write Address**

```
I2C1_MasterWrite(eepromWriteBuffer, NUM_ADDRESS_  
BYTES, I2C_SLAVE_ADDRESS, &i2cStatus);
```

因為只有一個 **EEPROM** 位址須設定，所以：

NUM_ADDRESS_BYTES = 1

- 再用 **Sequential Read**

```
I2C1_MasterRead(eepromReadBuffer,  
sizeof(eepromReadBuffer) , I2C_SLAVE_ADDRESS,  
&i2cStatus);
```

sizeof(eepromReadBuffer) 計算為 8 個 **Bytes**，所以函數會讀取 8 **Bytes** 資料到 **eepromReadBuffer[]** 裡。



MICROCHIP

Regional Training Centers

PIC18 Lab 4-2:

I²C™ 串列通訊

用 Master 模式存取 溫度感應器

TC74

(附加練習)

Lab4-2 說明

- **MCC設定相同**
- **修改 Lab4-1 的 I²C™ Byte 的操作來
讀取 I²C™ 溫度感應器 TC74**
- **確定 APP001 的 JP10 全部要短路接
通 I²C™ EEPROM 及 TC74**

TC74 的應用

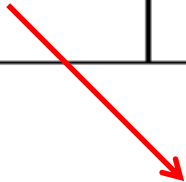
- CPU、硬碟溫度保護
- 電源供應器溫度保護
- PC週邊介面卡、筆記型電腦溫度保護
- 自動溫控系統
- 基板溫度量測、系統保護
- 一般室溫量測

TC74 的基本規格

- **SMBus™ / I²C™ 介面，傳送速率100KHZ_(Max.)**
- **量測溫度範圍**
 - ◆ +25°C ~ +85°C (精確度 +- 2°C)
 - ◆ 0°C ~ +125°C (精確度 +- 3°C)
- **內建溫度二極體，Delta-Sigma A/D 轉換器**
- **轉換速率：每秒 8 次**
- **工作電壓：2.7V ~ 5.5V**
- **消耗電流：200uA，靜態電流 5uA**
- **SOT-23 5-pin 包裝**

讀取 TC74 Configuration Data

Command	Code	Function
RTR	00h	Read Temperature (TEMP)
RWCR	01h	Read/Write Configuration (CONFIG)



CONFIG 暫存器

Bit	POR	Function	Type	Operation
D[7]	0	STANDBY Switch	Read/Write	1 = standby, 0 = normal
D[6]	0	Data Ready *	Read Only	1 = ready 0 = not ready
D[5]-D[0]	0	Reserved - Always returns zero when read	N/A	N/A

動作順序

先送出 0x01 讀取狀態
再送出 0x00 讀取溫度

讀取狀態資料後檢查
Config. 的 **Bit 6** 以了
解溫度轉換是否完成

讀取溫度資料

- 溫度採 **8-bit** 輸出，負溫採 **2'S** 格式
- **1 LSB** 代表 **1°C** 的溫度變化

Actual Temperature	Registered Temperature	Binary Hex
+130.00°C	+127°C	0111 1111
+127.00°C	+127°C	0111 1111
+126.50°C	+126°C	0111 1110
+25.25°C	+25°C	0001 1001
+0.50°C	0°C	0000 0000
+0.25°C	0°C	0000 0000
0.00°C	0°C	0000 0000
-0.25°C	-1°C	1111 1111
-0.50°C	-1°C	1111 1111
-0.75°C	-1°C	1111 1111
-1.00°C	-1°C	1111 1111
-25.00°C	-25°C	1110 0111

TC74-Ax

- **TC74-Ax 的讀取方式**
 - ◆ **TC74 為使用 SMBusTM / I²CTM 通信協定的溫度偵測器**
 - ◆ **TC74 的基本位址為 1001xxx0**
 - **xxx 為位址位元，範圍由 0 .. 7，料號中的 Ax 表示其使用的位址 (TC74-A0 .. TC74-A7)**
 - ◆ **TC74 的讀 / 寫格式與一般標準的 EEPROM 相同，只是位址不同而已**

TC74-A7 I²CTM 位址為 0b1001111x

定義 TC74-A7 的常數、變數

```
union  
{
```

```
    int                Word ;  
    char              Bytes[2] ;  
    struct {  
        unsigned      : 6 ;  
        unsigned      BitD6 : 1 ;  
    };
```

```
} TC_74 ;
```

```
#define TC74_Addr      0b01001111      // Define the TC74-A7 address with write command  
#define TC74_RWCR      0x01            // Define the Read/Write Configuration Command  
#define TC74_RTR        0x00           // Define the read temperature command
```

```
#define NUM_ADDRESS_BYTES 0x01
```

// 定義寫入的 **EEPROM** 的陣列資料，(寫入位置 + 資料)

```
uint8_t TC74WriteBuffer[3] = {0x00};
```

```
uint8_t TC74ReadBuffer[2] = {0x00};
```

TC74 I²C™ Protocol

● TC74 I²C™ 讀取資料的格式

Read Byte Format

S	Address	WR	ACK	Command	ACK	\$	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address

Command Byte: selects which register you are reading from.

Slave Address: repeated due to change in data-flow direction.

Data Byte: reads from the register set by the command byte.

寫入欲讀取暫存器
Command = 0 溫度
Command = 1 狀態

讀取的 **Cofig.** 狀態
或
讀取溫度資料



動手做實驗

PIC18 Lab 4-2

執行 Lab 4-2

- 開啟 **LAB4-2** 專案
 - ◆ 該專案的範例: 一樣使用 **Lab4** 裡的 **MCC** 所產生的周邊函數
 - ◆ 確認如何讀取 **TC74** (Lab4-2.c 176~204行)
 - ◆ 訊息顯示在 終端機及 **LCD** 上

讀取 TC74-A7 的溫度資料

```
unsigned Read_TC74_Temperature(void)
```

```
{
```

```
    I2C1_MESSAGE_STATUS i2cStatus;
```

```
    TC74WriteBuffer[0] = TC74_RWCR;
```

```
    I2C1_MasterWrite(TC74WriteBuffer, 1, TC74_Addr, &i2cStatus);
```

讀取狀態資料

// 寫入 Config. 暫存器位址 (0x01)

```
    I2C1_MasterRead(TC74ReadBuffer,1,TC74_Addr, &i2cStatus);
```

```
    TC_74.Word = TC74ReadBuffer[0];
```

```
    if ( TC_74.BitD6 )
```

```
        // b6 =1 , Read temperature
```

```
    {
```

```
        // 用 Random Read 方式讀取 TC74 的 溫度值
```

```
        TC74WriteBuffer[0] = TC74_RTR;
```

```
        I2C1_MasterWrite(TC74WriteBuffer, 1, TC74_Addr, &i2cStatus);
```

讀取溫度資料

// 寫入 Config. 暫存器位址 (0x00)

```
        I2C1_MasterRead(TC74ReadBuffer,1,TC74_Addr, &i2cStatus);
```

```
        TC_74.Word = TC74ReadBuffer[0];
```

```
        if ( TC_74.Word >= 0 ) return TC_74.Word +1;
```

// +1 度 C

```
        else return -1;
```

// 負溫度, return (-1)

```
    }
```

```
    else return -2;
```

// b6=0, TC74 處於忙碌狀態中 return (-2)

```
}
```

溫度顯示在 LCD

Lab 4-2.c

- 溫度值為一 **16** 進制值需經數值轉換後才能顯示
 - ◆ 數值必須轉換為 **ASCII code** 的字元或字串後才能顯示於終端機或 **LCD**
 - ◆ XC8 提供此轉換函數 (itoa())

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
char * itoa (char * buf, int val, int base)
```

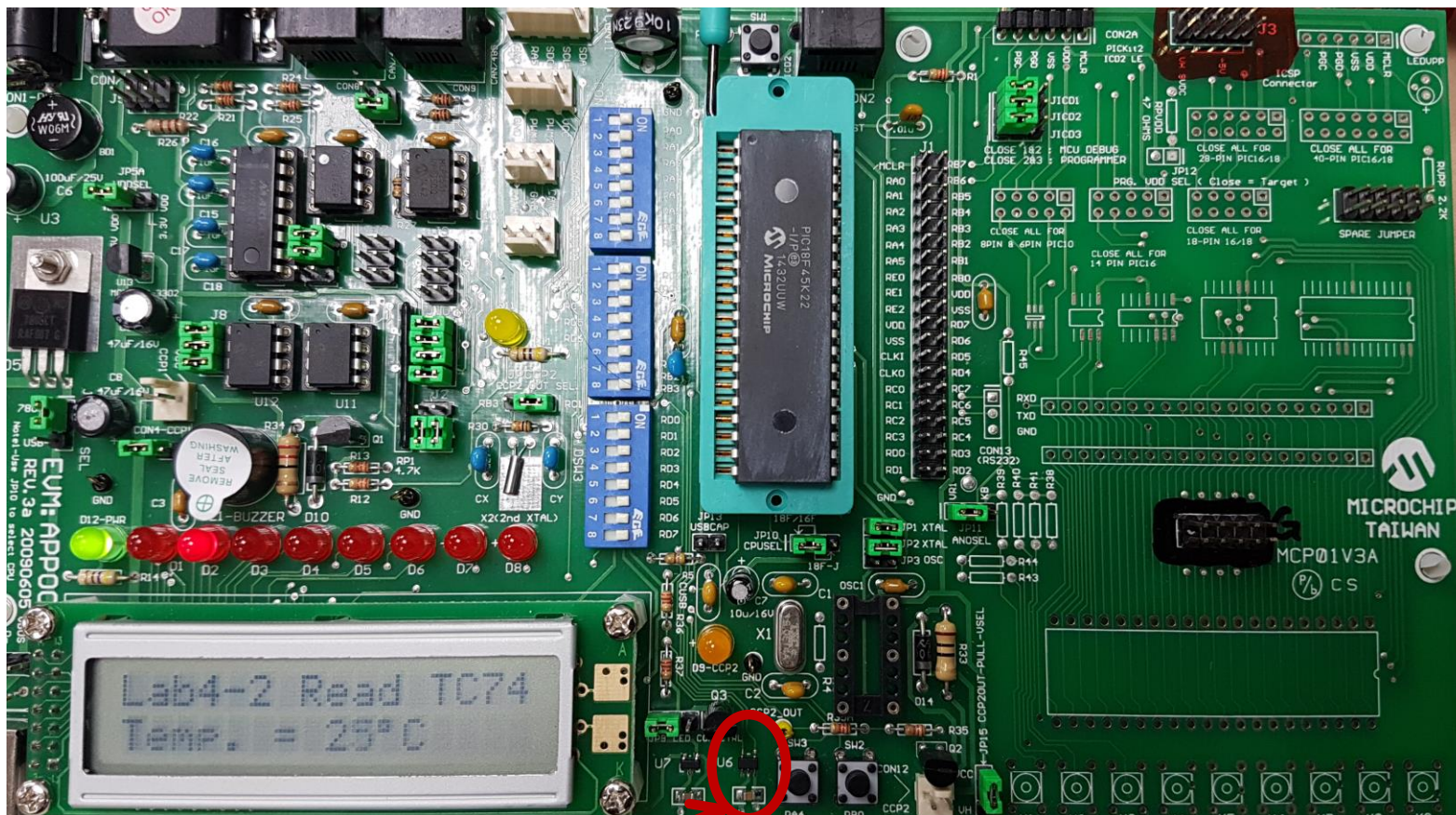
***buf** : 轉換成字串格式後儲存指標

val : 要轉換的 **16-bit** 資料

base : 轉換值的基底 (十進制 , 16 進制)

Lab 4-2 程式執行

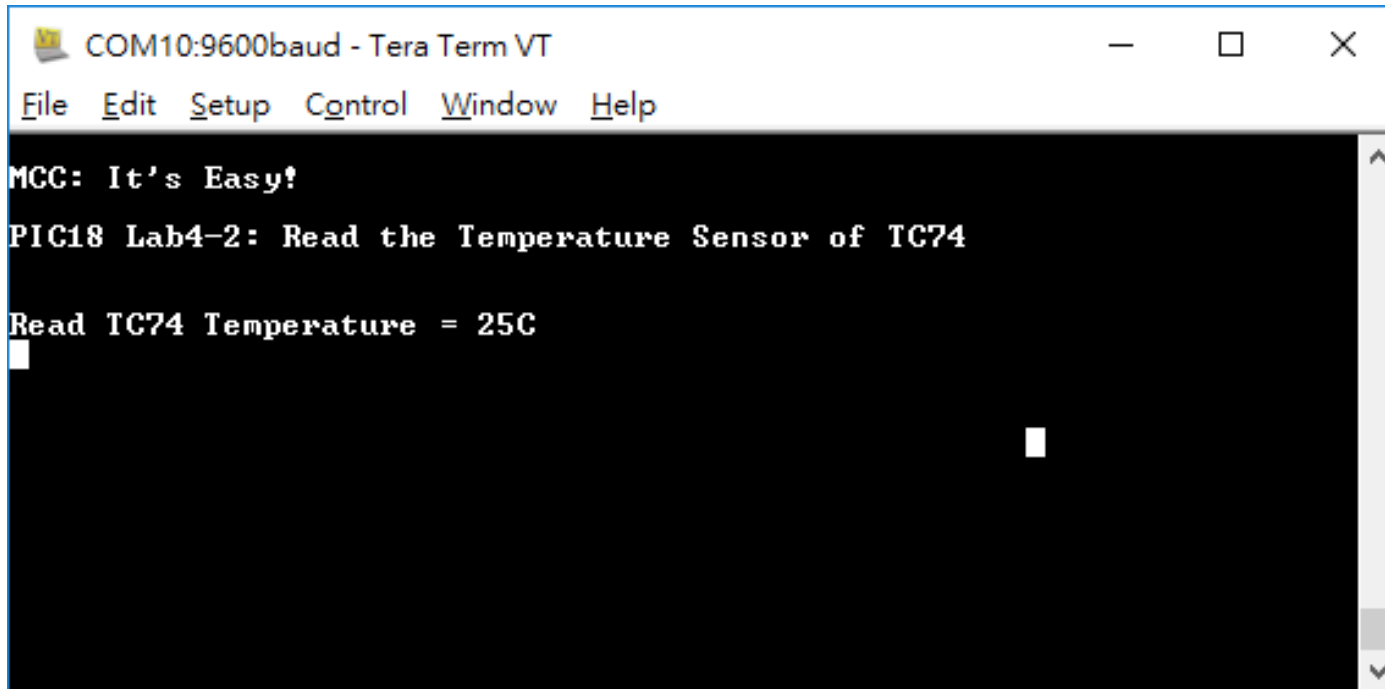
- 手按下U6 觀看溫度可以變化



TC74

Lab 4-2 執行顯示

- 本練習的終端機及 LCD 的顯示



```
COM10:9600baud - Tera Term VT
File Edit Setup Control Window Help
MCC: It's Easy!
PIC18 Lab4-2: Read the Temperature Sensor of TC74
Read TC74 Temperature = 25C
█
```

Lab4-2 Read TC74
Temp. = 25⁰C

LCD 顯示



MICROCHIP

Regional Training Centers

PIC18 Lab 5:

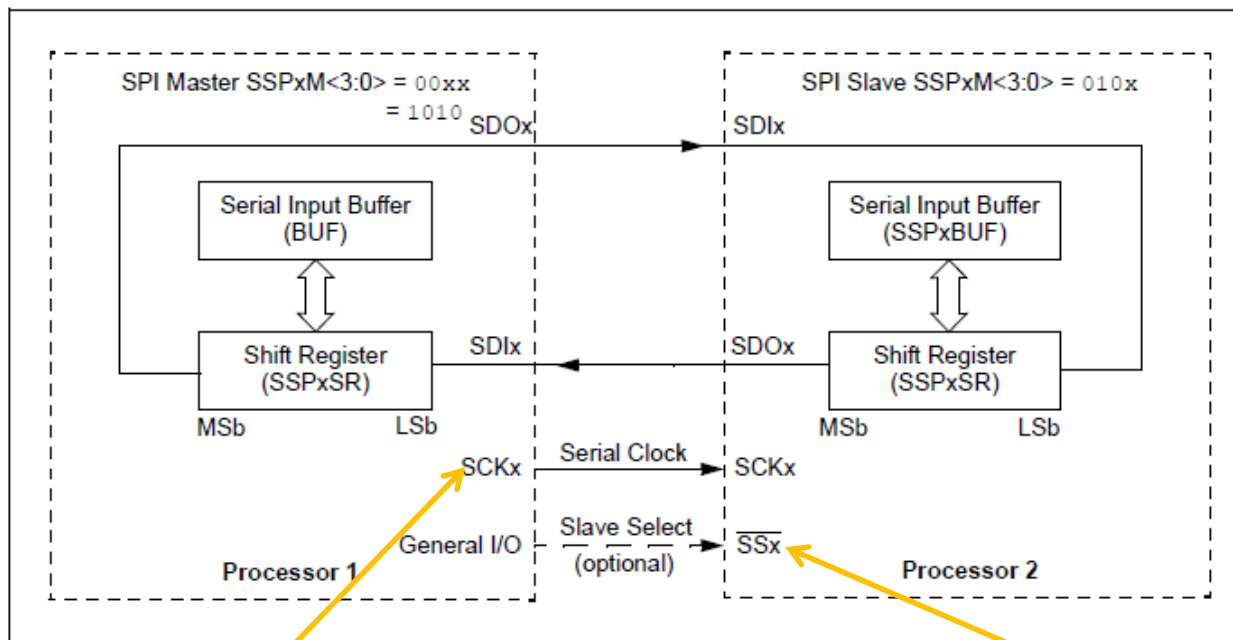
使用 SPI 介面讀取 25LC160A

PIC18 Lab 5 目標

- 使用 **MCC** 產生的 **SPI1** 函數存取 **25LC160C** 的資料
- **Byte Read/Write** 模式
- **Page Read/Write** 模式

了解 SPI 資料傳輸

- Master 傳送一 Byte 給 Slave，同時也會自 Slave 接收一Byte 資料。
- ◆ 效率比 UART、I²C™ 更好
- ◆ Dummy Write (Read a Byte)



SCK 由誰送出
他就是 Master

用 I/O 腳
做 CS 的選擇

SPI 四種模式

- **CKP** : 決定 SCLK idle 的 state , 即平時是在 low 還是 high
- **CKE**: 決定取樣點是在第一個 edge , 還是第二個 edge
 - 因此共有四種資料擷取的模式 (**Strobe**) , 要用那一種依你的 **SPI Device** 而定

Mode	極性 (POL)	取樣點(PHA)	Idle	Active	Latch
0,0 (0)	0	0	Hi	Low	Hi → Low
0,1 (1)	0	1	Hi	Low	Low → Hi
1,0 (2)	1	0	Low	Hi	Low → Hi
1,1 (3)	1	1	Low	Hi	Hi → Low

SPI 四種模式

● PIC 暫存器設定 (ARM 及PIC 跟其他元件的CKE定義不同)

Mode	極性 (CKP)	取樣點(CKE)	Idle	Active	Latch
0,1 (0)	0	1	Low	Hi	Hi → Low
0,0 (1)	0	0	Low	Hi	Low → Hi
1,1 (2)	1	1	Hi	Low	Low → Hi
1,0 (3)	1	0	Hi	Low	Hi → Low

CKP = 1:

Idle state for clock is a **high** level

CKE = 1 :

Transmit occurs on transition from **active to Idle** clock state

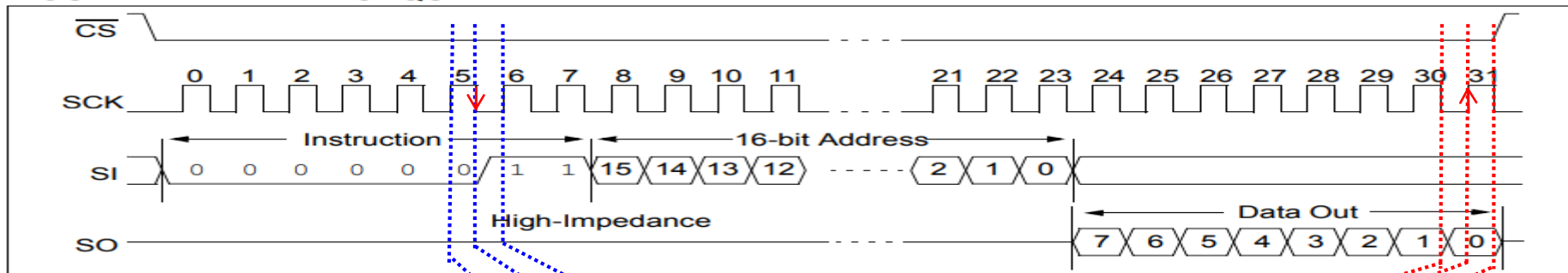
SMP = 1:

Input data sampled at **end** of data output time

25LC160C 時序

25LC160C 時序圖

FIGURE 2-1: READ SEQUENCE



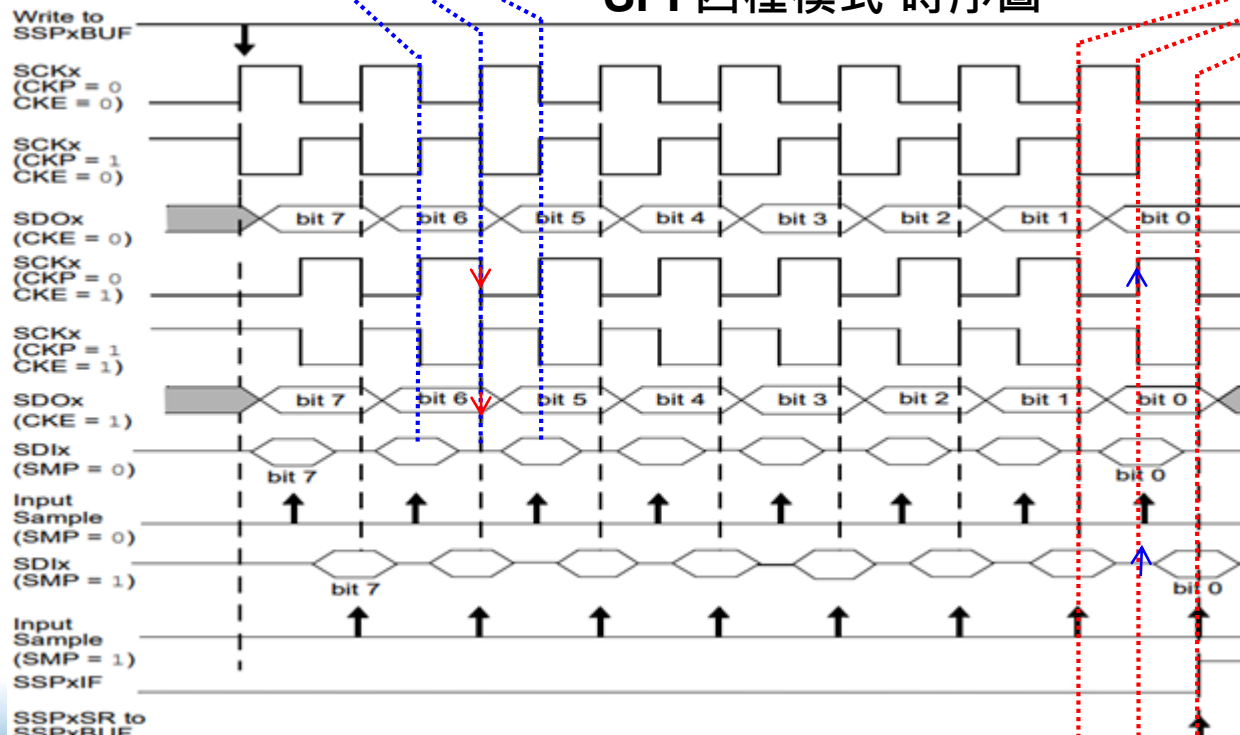
SPI 四種模式 時序圖

Mode 1

Mode 3

Mode 0

Mode 2



選 Mode 0
CKP=0
CKE=1

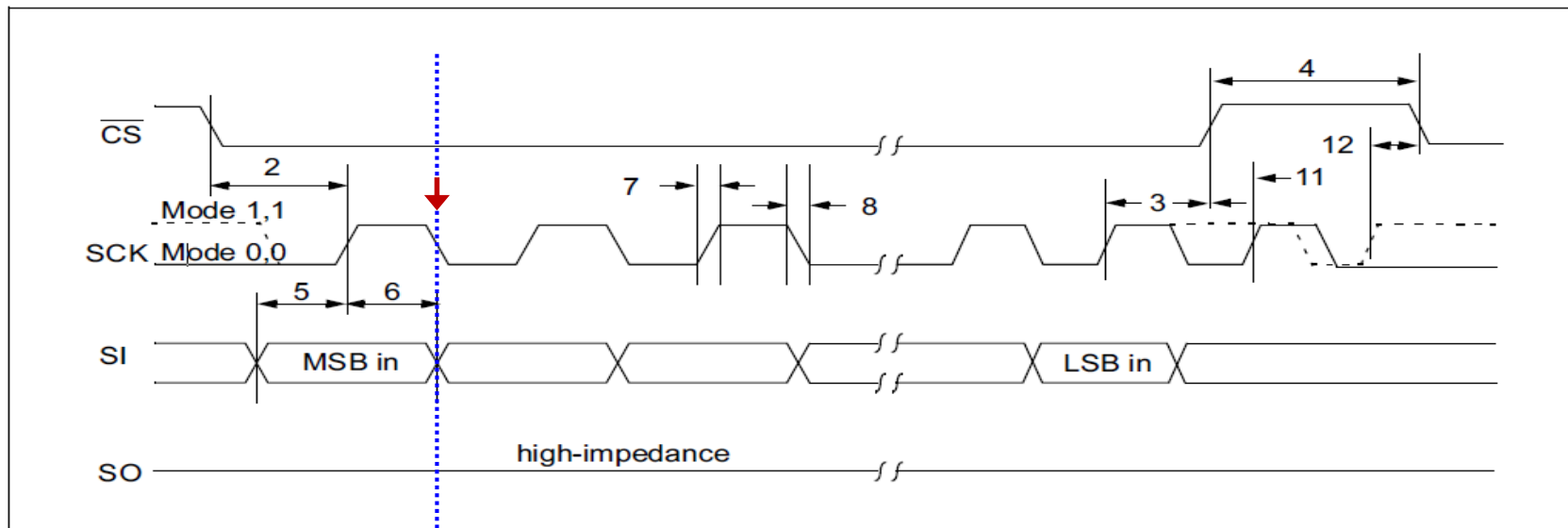
選 Middle
SMP=0

CKE = 1 : Transmit occurs on transition from **active to Idle** clock state

25LC160C 時序

25LC160C 時序圖

FIGURE 1-2: SERIAL INPUT TIMING



CKP

Idle=0



Idle=1



CKE

0: Idle -> Active
(第一次變化時傳資料)



Mode 1

1: Active -> Idle
(第二次變化時傳資料)



Mode 0



Mode 3



Mode 2

CKE = 1 : Transmit occurs on transition from active to Idle clock state

25LC160C 控制命令

- **25LC160C** 的寫入時間為 **5mS**
- 在 **WREN** 狀態下，資料才允許被寫入到 **EEPROM** 裡。
- 讀取動作隨時可執行，不受 **WREN** 控制

Instruction Name	Instruction Format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
WRDI	0000 0100	Reset the write enable latch (disable write operations)
WREN	0000 0110	Set the write enable latch (enable write operations)
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register

Lab5 相關接腳

- **25LC160A SPI EEPROM**
 - ◆ CS : RA5
 - ◆ SCK1 : RC3
 - ◆ SDO1: RC5
 - ◆ SDI1: RC4
- **PWM1: RC2 (Buzzer)**
- **可變電阻: RA0**

PIC18 Lab 5 概述

- 關閉 **MPLAB® X IDE** 下的所有專案
- 開啟 **lab5_8bit.X** 的專案
- 確定 **24LC160A** 已安裝在 **APP001** 板上
U11 位置
 - ◆ J9 全部短路，J10 全部開路
 - ◆ J13 USB Cap 開路，JP4 短路
- 設定 **SPI1** 模組
- 設定 **ECCP1::PWM** 來撥放音樂 (Lab5-1)



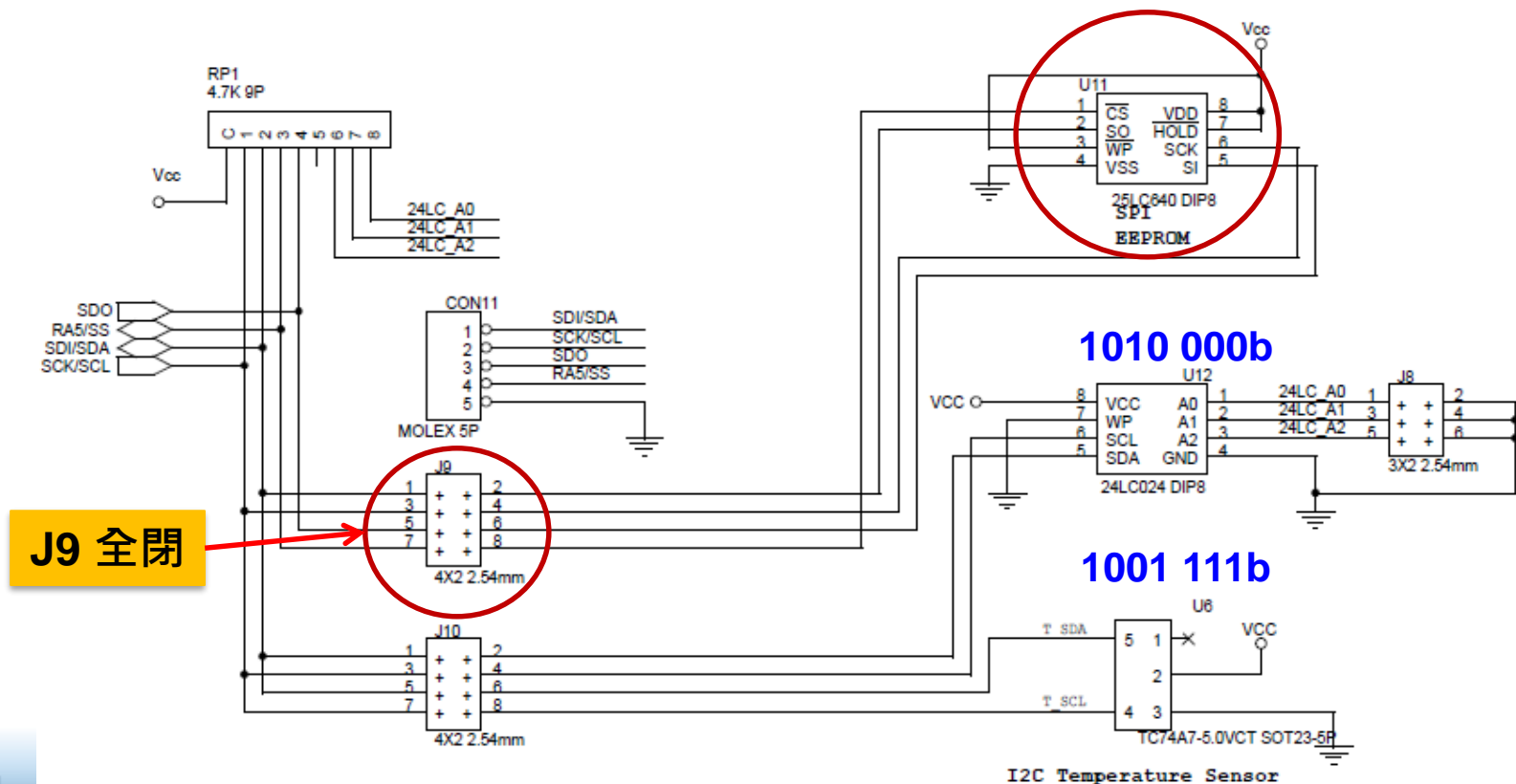
動手做實驗

Lab 5

25LC160C Byte 操作

APP001 J10 的設定

- J10 用來設定 I²C™ 的元件選擇
 - ◆ 確定 J10 的 1 & 2, 3 & 4 是短路的
 - ◆ J10 全部短路也可以，因為 TC74 位址不同

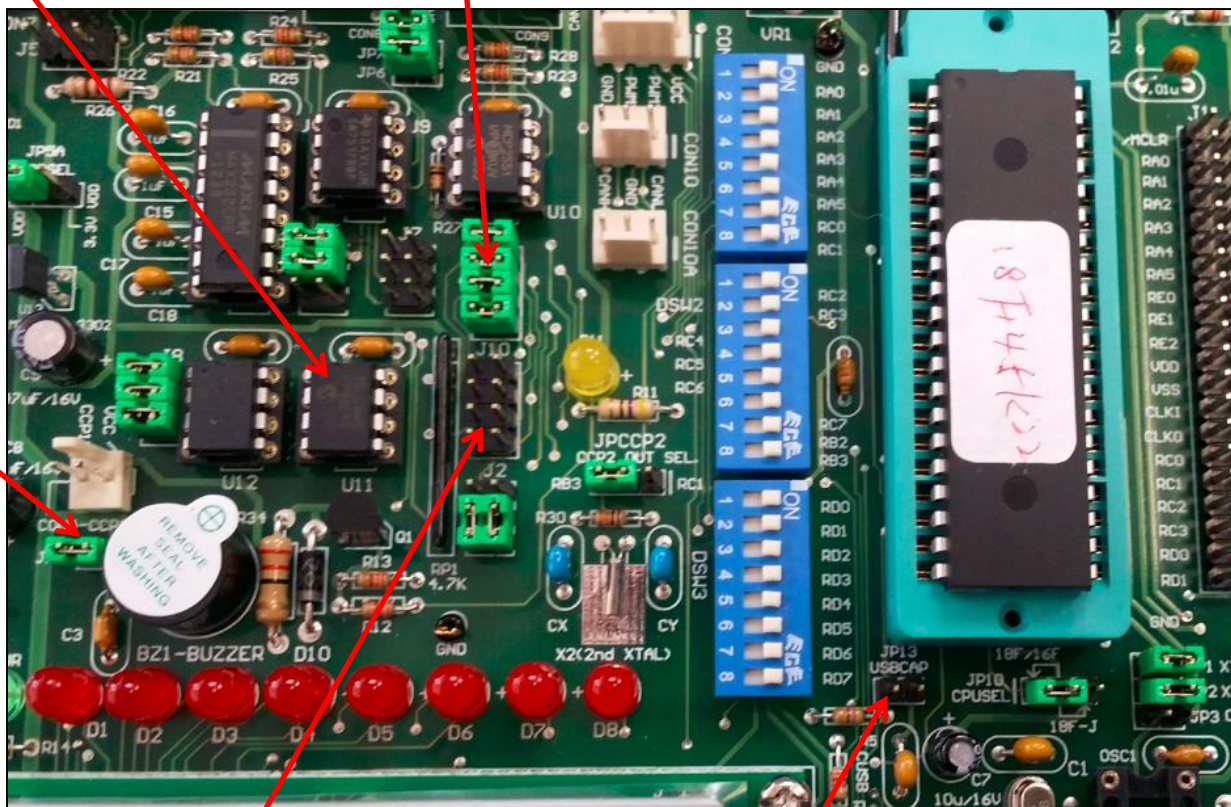


Lab5 板子設定

25LC160

J9 全閉

JP4 短路

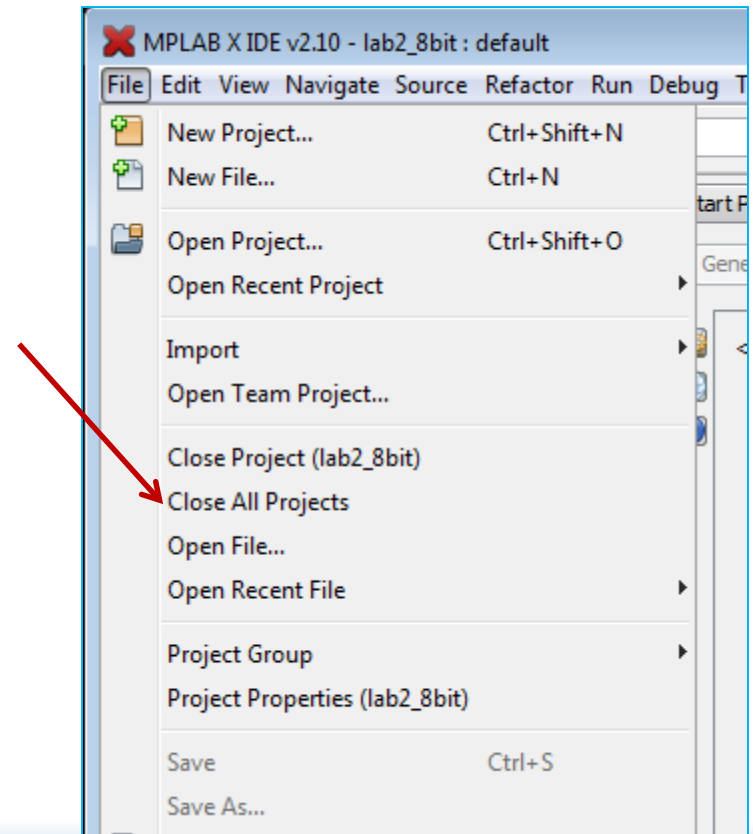
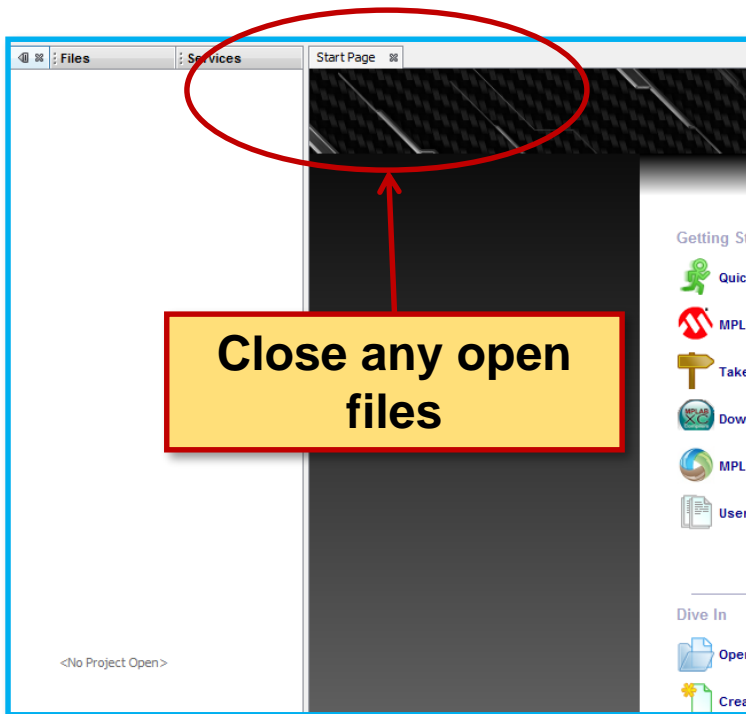


J10 全開路

JP13 開路

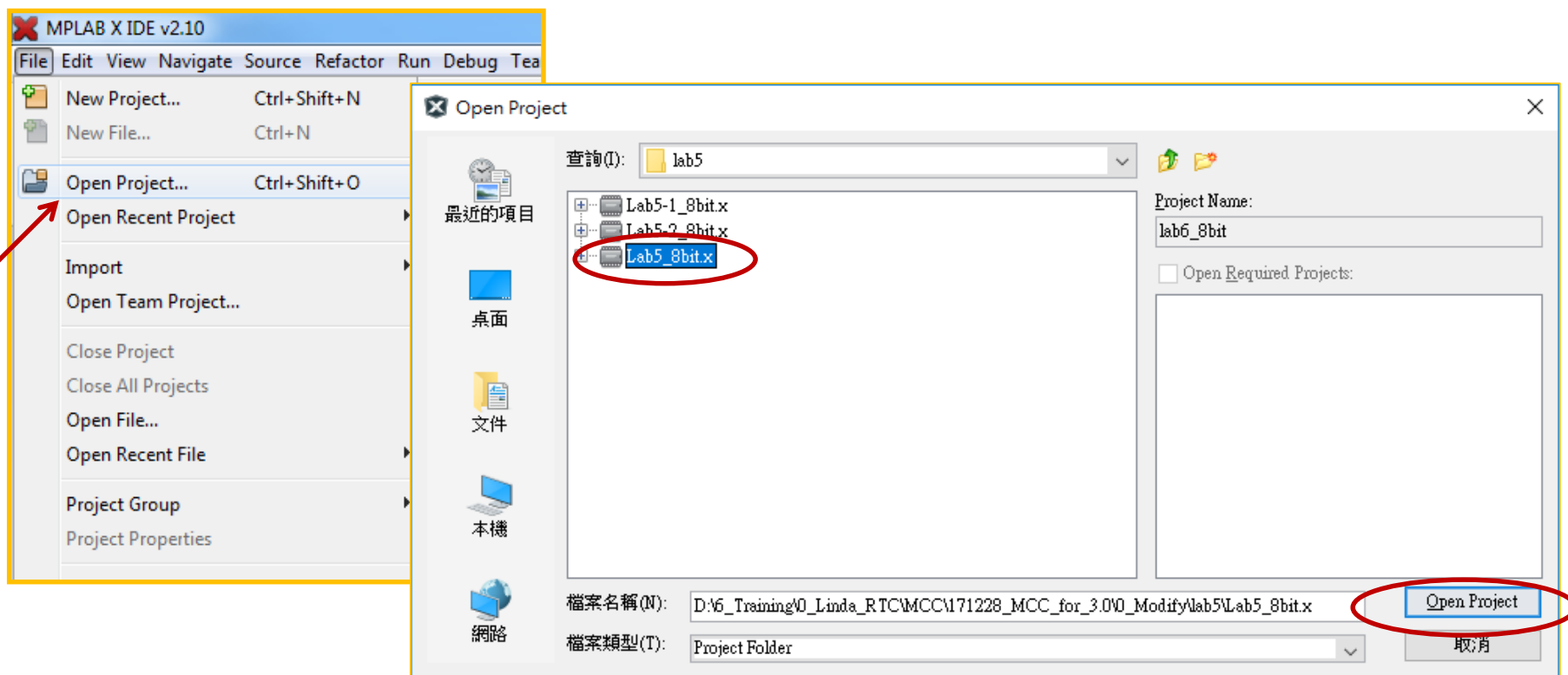
PIC18 Lab 5

- 關閉編輯視窗內的所有檔案及專案



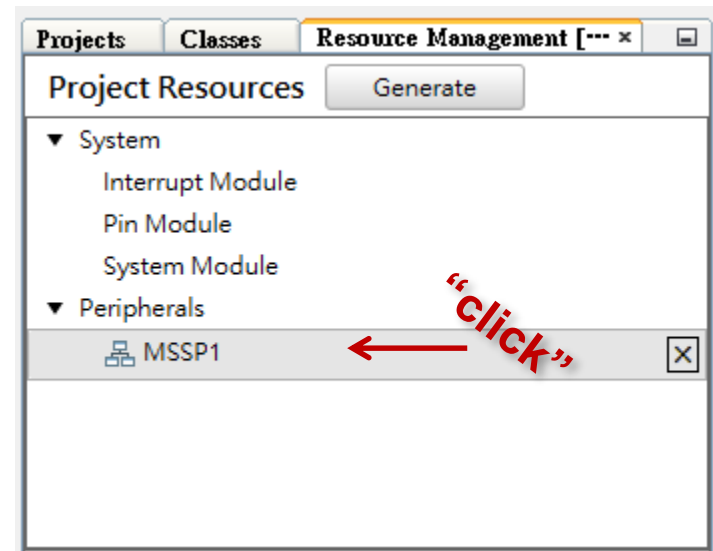
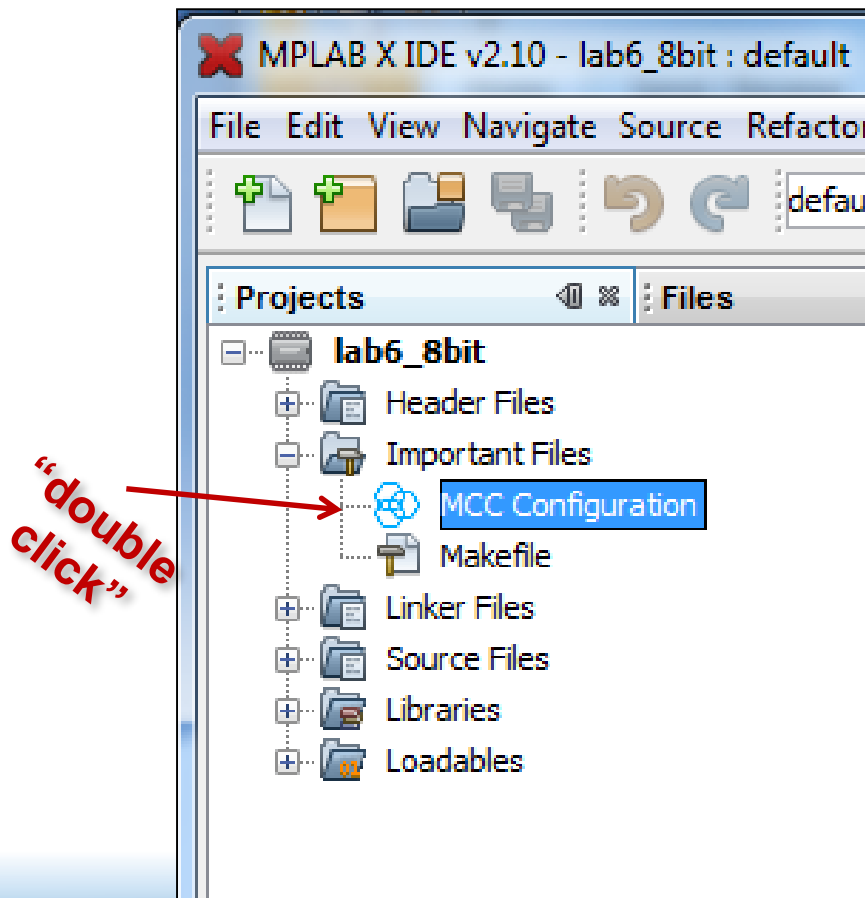
PIC18 Lab 5

- 開啟 lab5_8bit.X 的專案



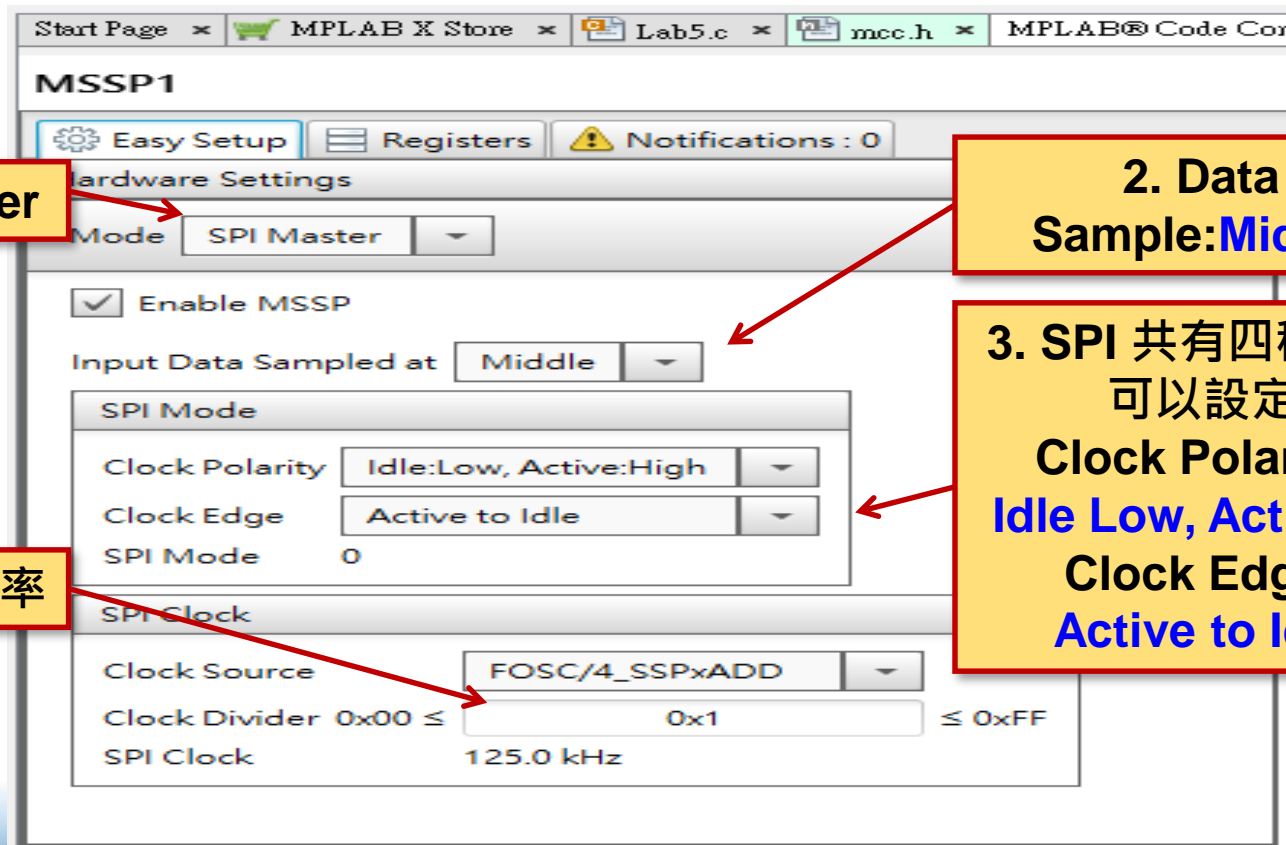
PIC18 Lab 5

1. 啟動 MPLAB® Code Configurator
2. 選擇 MSSP:MSSP1模式



PIC18 Lab 5

- 選擇 MSSP:MSSP1::SPI Master 模式
- 設定 SPI1 為 Mode 0，Sample: Middle



1. 選 SPI Master

2. Data
Sample: **Middle**




3. SPI 共有四種模式
可以設定
Clock Polarity:
Idle Low, Active Hi
Clock Edge:
Active to Idle

4. SPI Clock 頻率

PIC18 Lab 5













- 設定 **GPIO**: 加入 RA5 為 24LC160A 的 CS 控制腳，名稱為 **EEPROM_CHIP_SELECT**

Pin Module

<div>  Easy Setup  Registers  Notifications: 0 </div>						
Selected Package : UQFN40						
Pin Name^	Module	Function	Custom Name	Start High	Analog	Output
RA2	Pin Module	GPIO	LCD_E	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RA5	Pin Module	GPIO	EEPROM_CHIP_SELECT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RC3	MSSP1	SCK1		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RC4	MSSP1	SDI1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RC5	MSSP1	SDO1		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RD0	Pin Module	GPIO	LCD_DATA0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RD1	Pin Module	GPIO	LCD_DATA1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RD2	Pin Module	GPIO	LCD_DATA2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RD3	Pin Module	GPIO	LCD_DATA3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RD4	Pin Module	GPIO	LCD_RS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
RD5	Pin Module	GPIO	LCD_RW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

RA之為數位輸出腳，
初始設定值為 Hi，
更改 RA5 名稱

啟用 RA5 為數位腳

Variables	Output	Search Results	Pin Manager: Grid [MCC] ×									
Package:	UQFN40	▼	Pin No:	17	18	19	20	21	22	29	28	
				Port A ▼								
Module	Function	Direction	0	1	2	3	4	5	6	7		
OSC	CLKO	output										
Pin Module ▼	GPIO	input										
	GPIO	output										
RESET	MCLR	input										

SPI1_Exchange8bit()

- 透過 **SPI1** 傳送一個 **Byte** 資料
`readDummy = SPI1_Exchange8bit(Write date)`
- 透過 **SPI1** 讀取一個 **Byte** 資料後回傳
`readData = SPI1_Exchange8bit(DUMMY_DATA)`
 - ◆ 寫入 **DUMMY_DATA** 是假寫入，主要是送出 八個 **clock** 將資料自 **Slave** 傳回
- 使用 **MCC** 的 **SPI** 函數都要加入 **SPI1_Initialize()** 做初始的設定

Lab5.c 寫入資料

- 寫入動作

- ◆ 配合 CS 動作，送出 EEPROM 解鎖命令 (0x06)
- ◆ CS = Low，先送出 EEPROM 寫入命令 (0x02)，寫入 Msb 位址，再寫入 Lsb 位址
- ◆ 寫入儲存資料或連續寫入儲存資料
- ◆ CS = Hi
- ◆ Delay 5 ms 以上，等資料完成寫入動作

Lab5 EEPROM byte Write

寫入資料流程：

```
EEPROM_CHIP_SELECT_SetLow( );  
SPI1_Exchange8bit(EEPROM_EN_CMD);  
EEPROM_CHIP_SELECT_SetHigh( );  
__delay_ms(1);  
  
EEPROM_CHIP_SELECT_SetLow( );  
EEPROM  
SPI1_Exchange8bit(EEPROM_WRITE_CMD);  
SPI1_Exchange8bit(0x00);  
SPI1_Exchange8bit(0x00);  
SPI1_Exchange8bit(0x55);  
SPI1_Exchange8bit(0xAA);  
EEPROM_CHIP_SELECT_SetHigh( );  
  
__delay_ms(10);
```

// EEPROM CS = Low
// EEPROM 寫入解鎖
// EEPROM CS = Hi

// CS = Low, Enable
// 寫入的命令
// EEPROM Msb 位址
// EEPROM Lsb 位址
// 第一個寫入資料 (0x55)
// 第二個寫入資料 (0xAA)
// CS = Hi, Disable EEPROM

// Delay 10mS for EEPROM Write

Lab6 EEPROM byte Read

讀取資料流程：

```
EEPROM_CHIP_SELECT_SetLow( );    // CS =Low , Enable EEPROM  
SPI1_Exchange8bit(EEPROM_READ_CMD);    // 讀取的命令  
SPI1_Exchange8bit(0x00);           // 設定 Msb Address  
SPI1_Exchange8bit(0x00);           // 設定 Lsb Address  
EE_Read_Buffer[0] = SPI1_Exchange8bit(DUMMY_DATA); // Read 0x0000 資料  
EE_Read_Buffer[1] = SPI1_Exchange8bit(DUMMY_DATA); // Read 0x0001 資料  
EEPROM_CHIP_SELECT_SetHigh( );    // CS = Hi , Disable EEPROM
```

Lab5_8bit 專案執行

- 燒錄後，產生結果
 - ◆ 螢幕顯示根據程式Lab5.c中這兩行變化

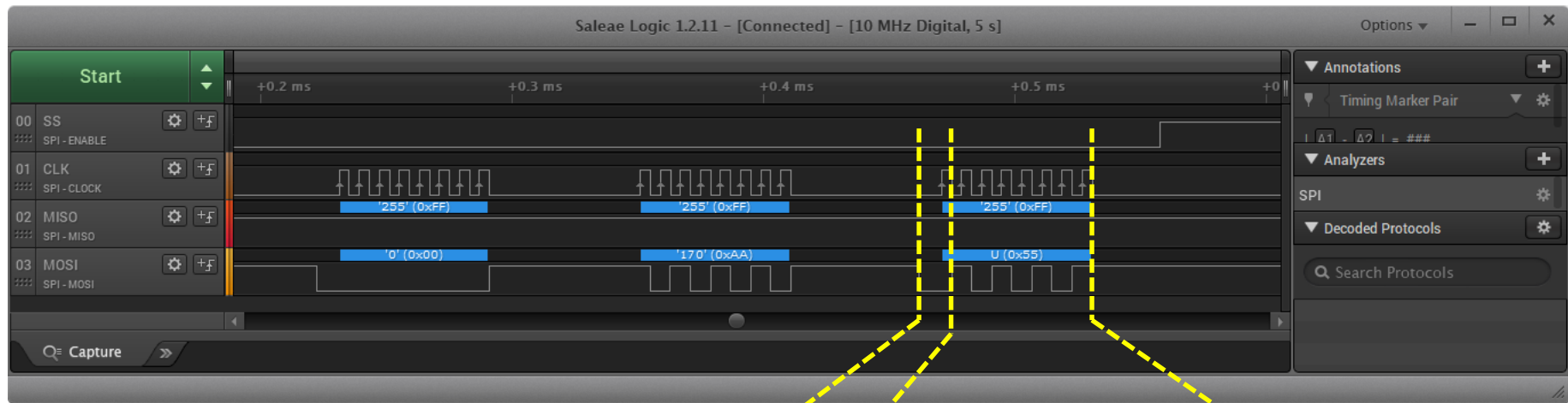
```
80 | EEPROM_CHIP_SELECT_SetLow();  
81 | SPI1_Exchange8bit(EEPROM_WRITE_CMD);  
82 | SPI1_Exchange8bit(0x00);  
83 | SPI1_Exchange8bit(0x00);  
84 | SPI1_Exchange8bit(0xAA);  
85 | SPI1_Exchange8bit(0x55);  
86 | EEPROM_CHIP_SELECT_SetHigh();
```

- LCD 會顯示

Lab5 EEPROM Byte
AA 55

LCD 顯示幕

Lab5_8bit 專案執行_0x55



Mode 0

Analyzer Settings

MOSI 3 - MOSI'

MISO 2 - MISO'

Clock 1 - CLK'

Enable 0 - SS'

Most Significant Bit First (Standard)

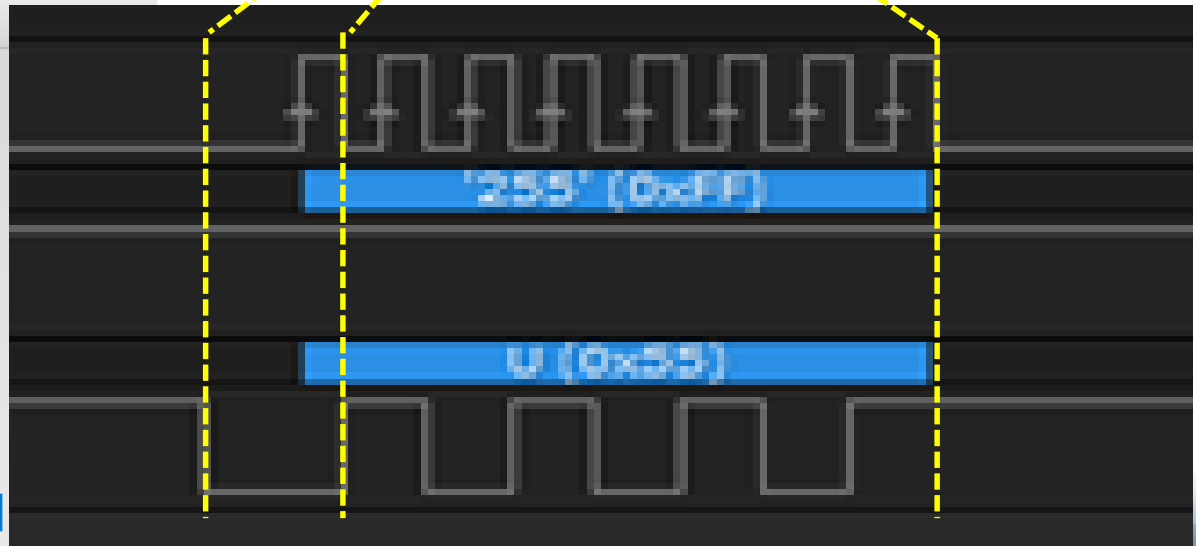
8 Bits per Transfer (Standard)

Clock is Low when inactive (CPOL = 0)

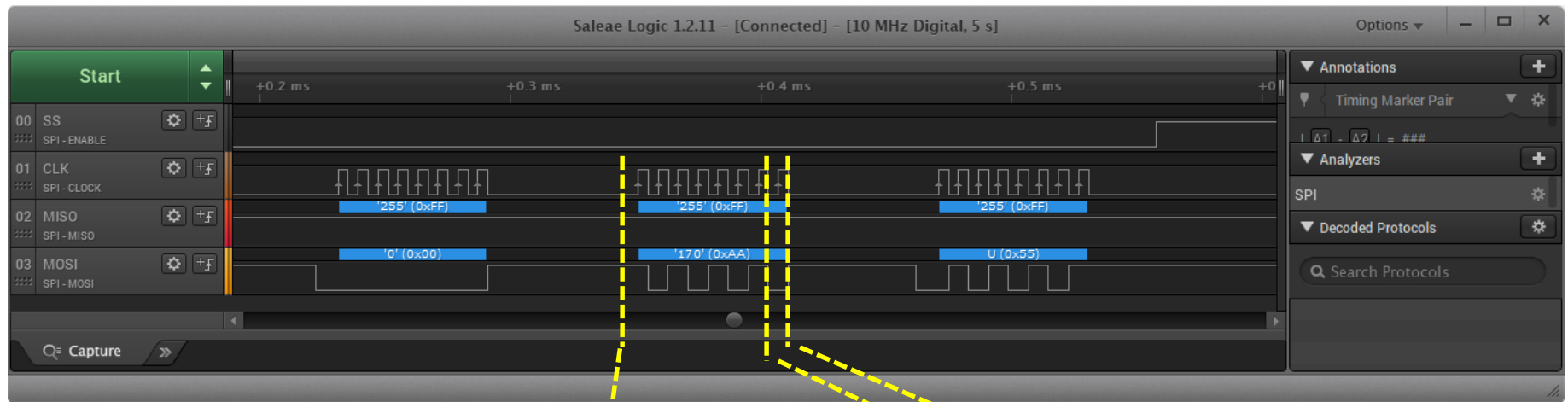
Data is Valid on Clock Leading Edge (CPHA = 0)

Enable line is Active Low (Standard)

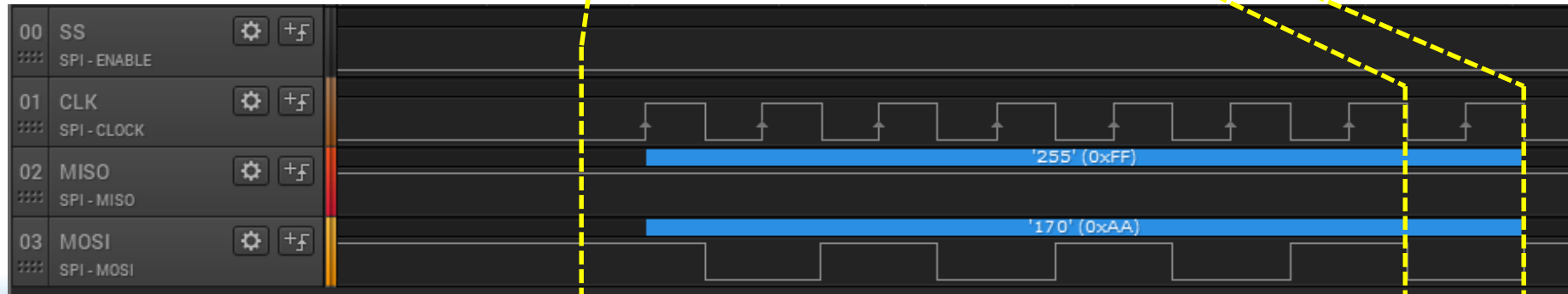
Save



Lab5_8bit 專案執行_0xAA



Mode 0





動手做實驗

Lab 5-1

25LC160C Page 操作

Lab5-1

Page Mode

- **25LC160C 內建 16 Bytes 的 Page Buffer**
- **Page 大小會隨編號及容量而不同**
- **了解如何使用 Page 模式來存取 EEPROM**

Lab5-1

Page Write

- **Page** 的寫入有邊界起始位址的限制
 - ◆ 以 25LC160C 為例，Page 有 16 Bytes
 - 寫入邊界位址為 **0xnnn0**，可完整寫入
 - 設定：寫入位址為 **0x0005**，資料會從 **0x0005** 開始寫入直到 **0x000F**，再來位址為指向同一 **Page** 的起始位址 **0x0000** 繼續寫入。

Addr...	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	4C	4E	4F	50	51	41	42	43	44	45	46	47	48	49	4A	4B
010	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
020	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
030	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
040	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

資料遭覆蓋了

本視窗是使用 **Serial Memory Starter Kit (DV243003)**
所截錄的 **25LC160C** 資料驗證畫面

Lab5-1

Page Write and Read

● 範例

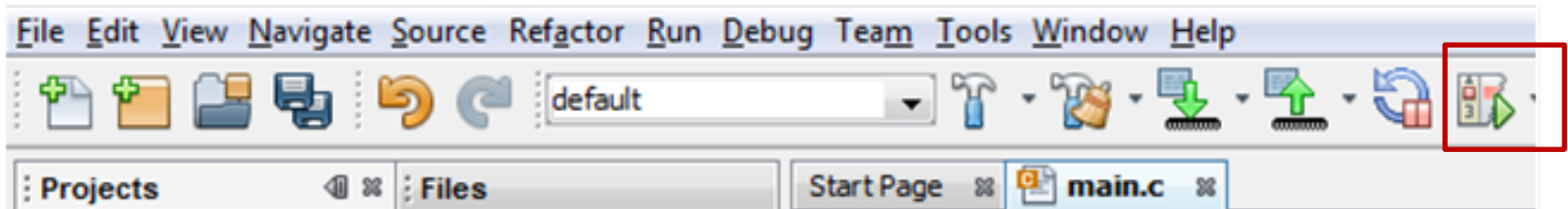
```
80 // 設定寫入命令及寫入的起始位址
81 EE_CMD_Buffer[0] = EEPROM_WRITE_CMD; // 設定寫入命令及起始位址 0x0000
82 EE_CMD_Buffer[1] = 0x00; // MSB Address
83 EE_CMD_Buffer[2] = 0x00; // LSB Address
84
85 // 先寫入命令及起始位址後再連續寫入 16 Bytes 的資料
86 EEPROM_CHIP_SELECT_SetLow();
87 SPI1_Exchange8bitBuffer(EE_CMD_Buffer,3,NULL);
88 SPI1_Exchange8bitBuffer(EE_Write_Buffer, 16, NULL);
89 EEPROM_CHIP_SELECT_SetHigh();
90
91 // delay 10mS 做寫入的延遲
92 __delay_ms(10);
93
94
95 // 設定讀取命令及起始位址
96 EE_CMD_Buffer[0] = EEPROM_READ_CMD;
97 EE_CMD_Buffer[1] = 0x00;
98 EE_CMD_Buffer[2] = 0x00;
99
100 // 送出讀取命令及起始位址，再連續讀取 16 Bytes 資料
101 EEPROM_CHIP_SELECT_SetLow();
102 SPI1_Exchange8bitBuffer(EE_CMD_Buffer,3,NULL);
103 SPI1_Exchange8bitBuffer(NULL, 16, EE_Read_Buffer);
104 EEPROM_CHIP_SELECT_SetHigh();
```

Page寫入

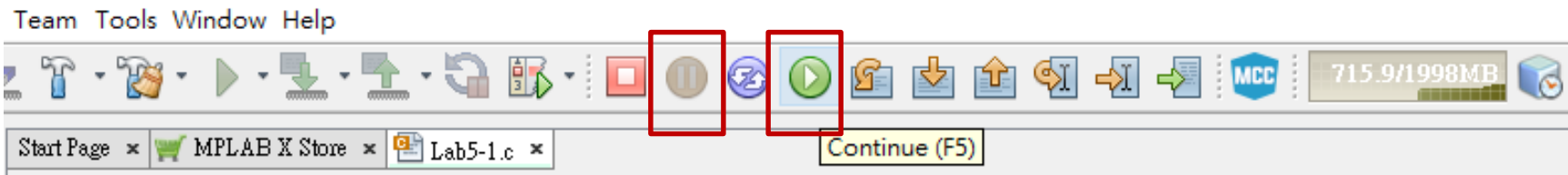
Page讀取

編譯及除錯 Lab5-1_8bit 專案

- 按下 “**Debug Main Project**” 的圖示開始編譯及燒錄並進入除錯模式

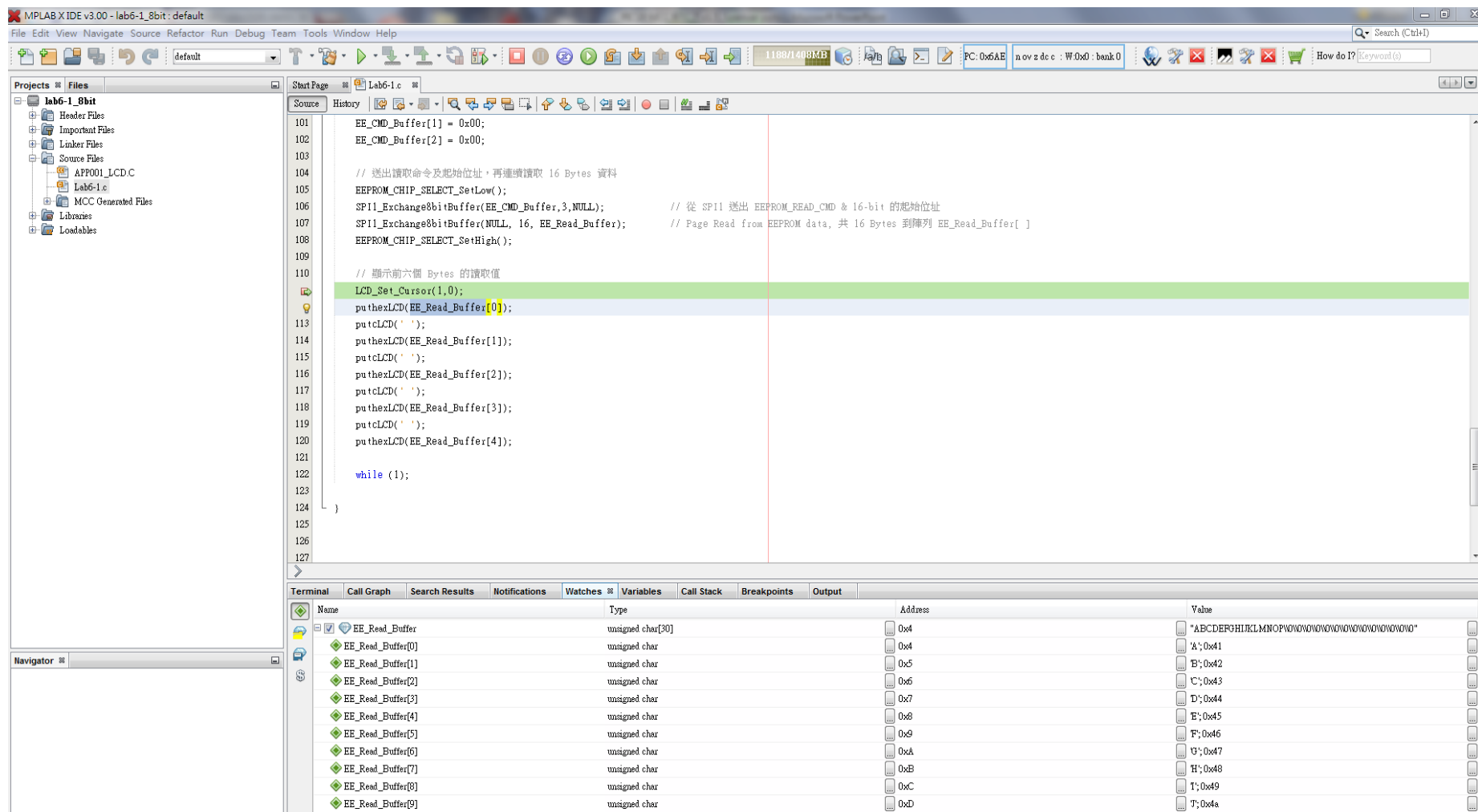


- 除錯模式執行、暫停



Lab5-1 執行結果

MPLAB® X IDE 變數顯示視窗



The screenshot shows the MPLAB X IDE v3.00 interface. The main window displays the source code for 'Lab5-1.c'. The code includes comments in Chinese and C code for initializing buffers, sending commands to an EEPROM, and displaying the read data on an LCD. The Watch window at the bottom shows the values of the 'EE_Read_Buffer' array elements, which are displayed as hexadecimal values from 0x4 to 0xD.

Source Code (Lab5-1.c):

```

101 EE_CMD_Buffer[1] = 0x00;
102 EE_CMD_Buffer[2] = 0x00;
103
104 // 送出讀取命令及起始位址，再連續讀取 16 Bytes 資料
105 EEPROM_CHIP_SELECT_SetLow();
106 SPI1_Exchange8bitBuffer(EE_CMD_Buffer, 3, NULL); // 從 SPI1 送出 EEPROM_READ_CMD & 16-bit 的起始位址
107 SPI1_Exchange8bitBuffer(NULL, 16, EE_Read_Buffer); // Page Read from EEPROM data, 共 16 Bytes 到陣列 EE_Read_Buffer[ ]
108 EEPROM_CHIP_SELECT_SetHigh();
109
110 // 顯示前六個 Bytes 的讀取值
111 LCD_Set_Cursor(1,0);
112 puthexLCD(EE_Read_Buffer[0]);
113 putcLCD(' ');
114 puthexLCD(EE_Read_Buffer[1]);
115 putcLCD(' ');
116 puthexLCD(EE_Read_Buffer[2]);
117 putcLCD(' ');
118 puthexLCD(EE_Read_Buffer[3]);
119 putcLCD(' ');
120 puthexLCD(EE_Read_Buffer[4]);
121
122 while (1);
123
124 }
  
```

Watch Window:

Name	Type	Address	Value
EE_Read_Buffer	unsigned char[30]	0x4	"ABCDER3HJIKLMNOPQWVWQWVWQWVWQWVWQWVWQW"
EE_Read_Buffer[0]	unsigned char	0x4	'A'; 0x41
EE_Read_Buffer[1]	unsigned char	0x5	'B'; 0x42
EE_Read_Buffer[2]	unsigned char	0x6	'C'; 0x43
EE_Read_Buffer[3]	unsigned char	0x7	'D'; 0x44
EE_Read_Buffer[4]	unsigned char	0x8	'E'; 0x45
EE_Read_Buffer[5]	unsigned char	0x9	'F'; 0x46
EE_Read_Buffer[6]	unsigned char	0xA	'G'; 0x47
EE_Read_Buffer[7]	unsigned char	0xB	'H'; 0x48
EE_Read_Buffer[8]	unsigned char	0xC	'I'; 0x49
EE_Read_Buffer[9]	unsigned char	0xD	'J'; 0x4a

Lab5-1 執行結果

MPLAB® X IDE debug模式下，變數顯示視窗

Variables					Call Stack					Breakpoints					Output				
Name	Type	Value	Address																
EE_Read_Buffer	unsigned char[30]	"ABCDEFGHJKLMNOPWOWOWC"	0x7																
EE_Read_Buffer[0]	unsigned char	'A'; 0x41	0x7																
EE_Read_Buffer[1]	unsigned char	'B'; 0x42	0x8																
EE_Read_Buffer[2]	unsigned char	'C'; 0x43	0x9																
EE_Read_Buffer[3]	unsigned char	'D'; 0x44	0xA																
EE_Read_Buffer[4]	unsigned char	'E'; 0x45	0xB																
EE_Read_Buffer[5]	unsigned char	'F'; 0x46	0xC																
EE_Read_Buffer[6]	unsigned char	'G'; 0x47	0xD																
EE_Read_Buffer[7]	unsigned char	'H'; 0x48	0xE																
EE_Read_Buffer[8]	unsigned char	'I'; 0x49	0xF																
EE_Read_Buffer[9]	unsigned char	'J'; 0x4a	0x10																
EE_Read_Buffer[10]	unsigned char	'K'; 0x4b	0x11																
EE_Read_Buffer[11]	unsigned char	'L'; 0x4c	0x12																
EE_Read_Buffer[12]	unsigned char	'M'; 0x4d	0x13																
EE_Read_Buffer[13]	unsigned char	'N'; 0x4e	0x14																
EE_Read_Buffer[14]	unsigned char	'O'; 0x4f	0x15																
EE_Read_Buffer[15]	unsigned char	'P'; 0x50	0x16																
EE_Read_Buffer[16]	unsigned char	NUL; 0x00	0x17																

LCD 顯示

Lab5-1 Page Mode
41 42 43 44 45

Lab5-1_8bit 專案執行

- 燒錄後，產生結果
 - ◆ 螢幕顯示根據程式Lab5-1.c中這兩行變化

```
59 unsigned char EE_Write_Buffer [ ] = { 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f,
60                                         0x50, 0x51, 0x52, 0x53, 0x00 };
61 unsigned char Read_Dummy = 0x00;

85 // 先寫入命令及起始位址後再連續寫入 16 Bytes 的資料
86 EEPROM_CHIP_SELECT_SetLow();
87 SPI1_Exchange8bitBuffer(EE_CMD_Buffer,3,NULL);
88 SPI1_Exchange8bitBuffer(EE_Write_Buffer, 16, NULL);
89 EEPROM_CHIP_SELECT_SetHigh();
```

- LCD 會顯示

Lab5-1 Page Mode
41 42 43 44 45

LCD 顯示幕



MICROCHIP

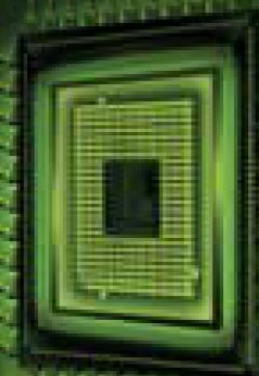
Regional Training Centers

PIC18 Lab5-2

使用 **SPI**、**PWM** 及 **ADC**

做個旋律撥放器

(附加的實驗)

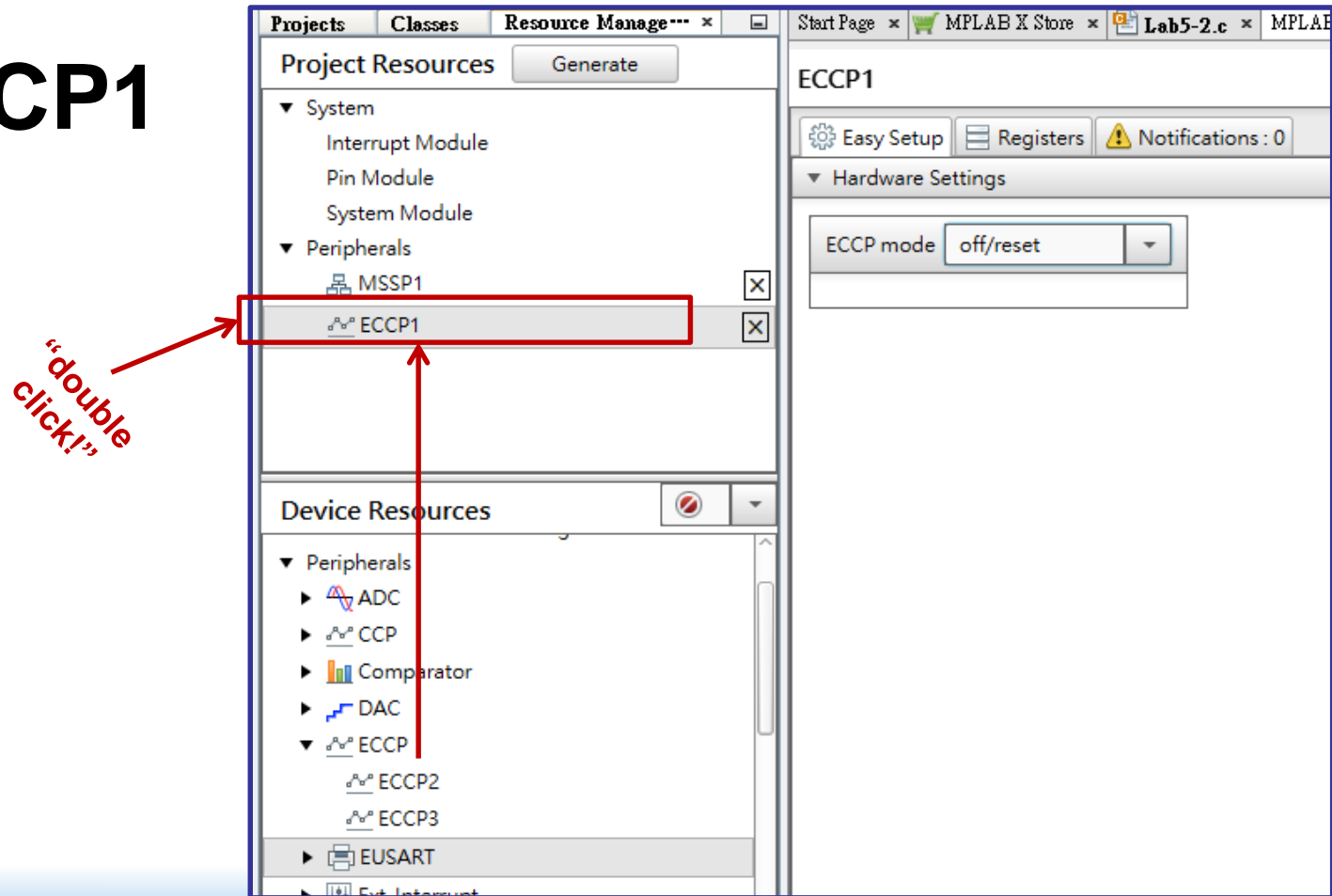


PIC18 Lab5-2 目標

- 設定 **SPI1** 介面來存取 **EEPROM** 的旋律資料
- 利用 **PWM1** 所接的蜂鳴器來撥放音樂
- 啟用 **ADC** 讀取變電阻 (**AN0**) 的電壓值做為撥放音樂節奏的快慢調整
- 整合 **MCC** 所產生的各個周邊函數完成音樂的撥放

PIC18 Lab5-2

- 加入 ECCP
ECCP1



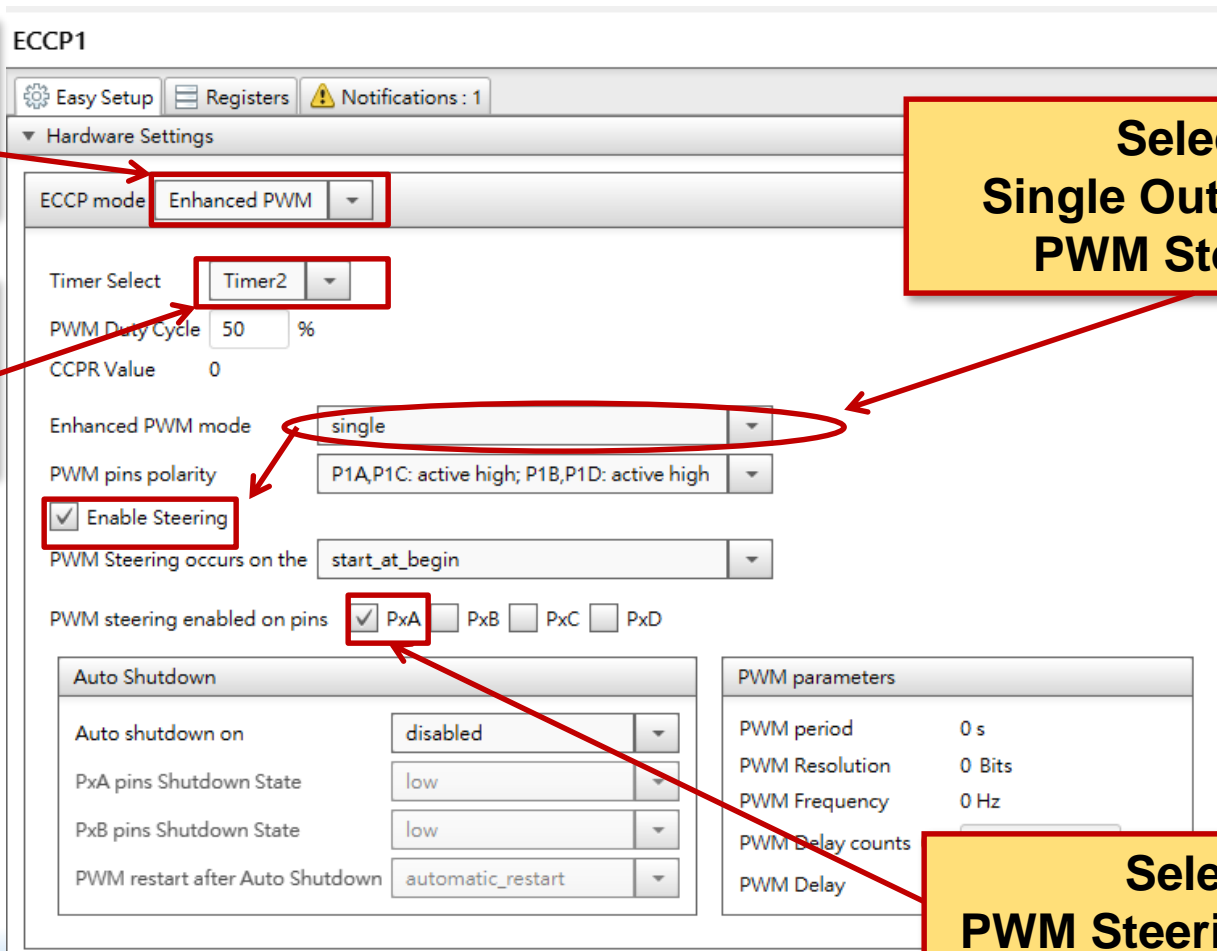
PIC18 Lab5-2

- 設定 ECCP:ECCP1: Enhanced PWM module

ECCP選
Enhanced
PWM模式

PWM
時脈來源
Timer2

Select:
Single Output with
PWM Steering



ECCP1

Easy Setup Registers Notifications: 1

Hardware Settings

ECCP mode: Enhanced PWM

Timer Select: Timer2

PWM Duty Cycle: 50 %

CCPR Value: 0

Enhanced PWM mode: single

PWM pins polarity: P1A,P1C: active high; P1B,P1D: active high

☒ Enable Steering

PWM Steering occurs on the: start_at_begin

PWM steering enabled on pins: ☒ PxA ☐ PxB ☐ PxC ☐ PxD

Auto Shutdown

Auto shutdown on	disabled
PxA pins Shutdown State	low
PxB pins Shutdown State	low
PWM restart after Auto Shutdown	automatic_restart

PWM parameters

PWM period	0 s
PWM Resolution	0 Bits
PWM Frequency	0 Hz
PWM Delay counts	
PWM Delay	

Select:
PWM Steering on PxA

PIC18 Lab5-2

- 設定 ECCP:ECCP1: Enhanced PWM module

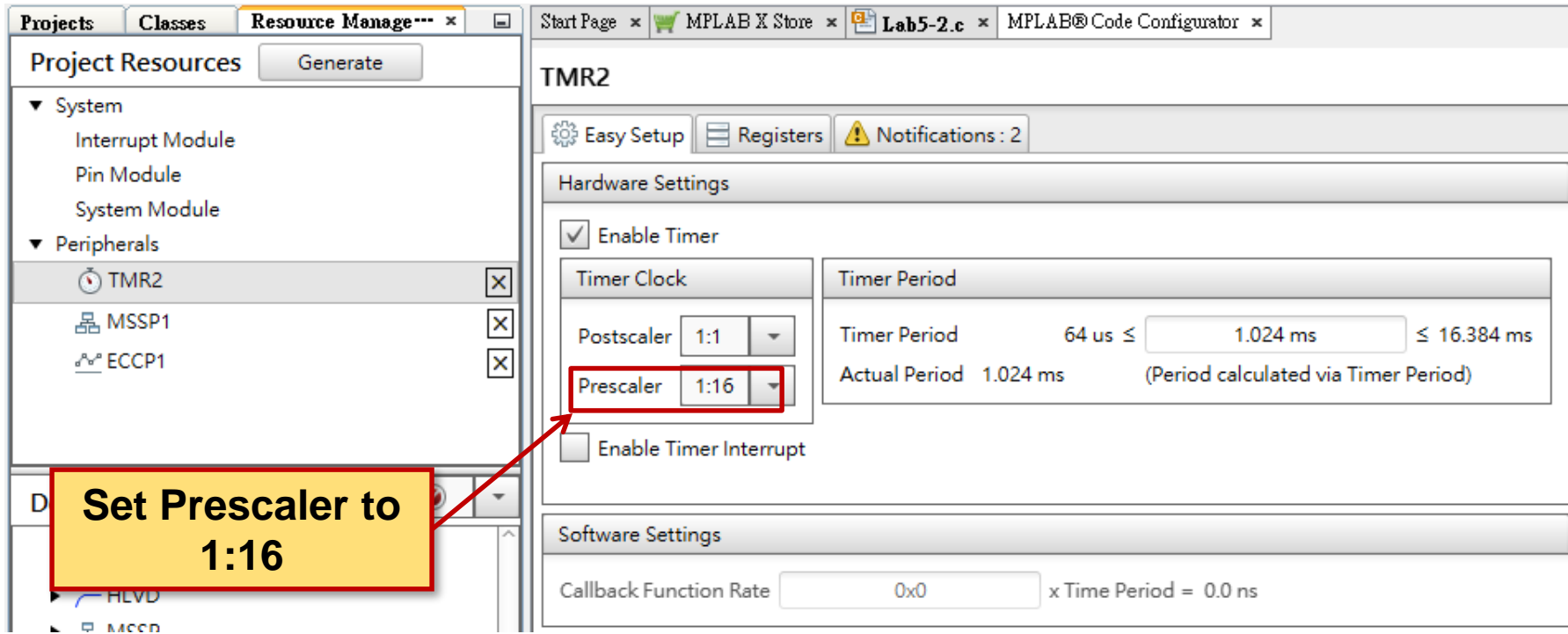
選Pin Manger頁面

Variables	Output	Search Results	Pin Manager: Grid [MCC] ×																																					
Package:	PDIP40	▼	Pin No:	2	3	4	5	6	7	14	13	33	34	35	36	37	38	39	40	15	16	17	18	23	24	25	26	19	20	21	22	27	28	29	30	8	9	10	1	
			Port A ▼								Port B ▼								Port C ▼								Port D ▼								Port E ▼					
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3		
ECCP1 ▼	FLT0	input																																						
	P1A	output																																						
	P1B	output																																						
	P1C	output																																						
	P1D	output																																						
	SCK1	output																																						

設定RC2 為P1A輸出

PIC18 Lab5-2

- 設定 Timer 2 module 為 PWM 時脈來源
 - ◆ Prescaler: 1:16



Set Prescaler to 1:16

TMR2

Easy Setup | Registers | Notifications : 2

Hardware Settings

☒ Enable Timer

Timer Clock

Postscaler 1:1

Prescaler 1:16

Timer Period

Timer Period 64 us ≤ 1.024 ms ≤ 16.384 ms

Actual Period 1.024 ms (Period calculated via Timer Period)

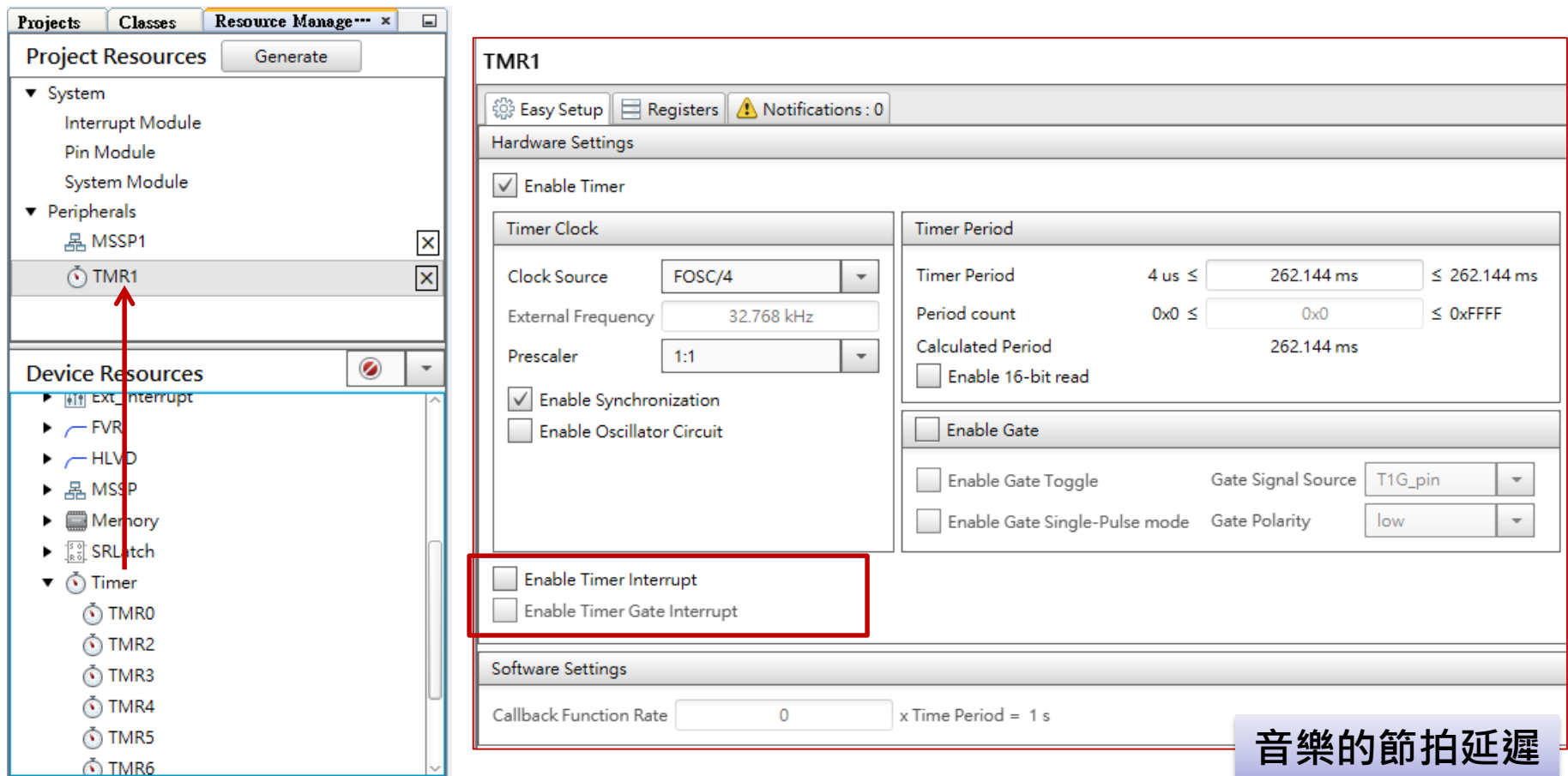
☐ Enable Timer Interrupt

Software Settings

Callback Function Rate 0x0 x Time Period = 0.0 ns

PIC18 Lab 5-2

- 設定 Timer1 (MusicNoteDelay) , 確認關閉中斷功能



Projects **Classes** **Resource Manage...** x

Project Resources **Generate**

- System
 - Interrupt Module
 - Pin Module
 - System Module
- Peripherals
 - MSSP1
 - TMR1**

Device Resources

- Ext. Interrupt
- FVR
- HLVD
- MSSP
- Memory
- SRLatch
- Timer
 - TMR0
 - TMR2
 - TMR3
 - TMR4
 - TMR5
 - TMR6

TMR1

Easy Setup **Registers** **Notifications : 0**

Hardware Settings

☒ Enable Timer

Timer Clock

Clock Source: FOSC/4

External Frequency: 32.768 kHz

Prescaler: 1:1

☒ Enable Synchronization

☐ Enable Oscillator Circuit

Timer Period

Timer Period: 4 us ≤ 262.144 ms ≤ 262.144 ms

Period count: 0x0 ≤ 0x0 ≤ 0xFFFF

Calculated Period: 262.144 ms

☐ Enable 16-bit read

☐ Enable Gate

☐ Enable Gate Toggle Gate Signal Source: T1G_pin

☐ Enable Gate Single-Pulse mode Gate Polarity: low

☐ Enable Timer Interrupt

☐ Enable Timer Gate Interrupt

Software Settings

Callback Function Rate: 0 x Time Period = 1 s

音樂的節拍延遲

PIC18 Lab5-2

● ADC 的設定

使用 VDD & VSS 做為參考電壓值

RA0 為可變電阻輸入，更名為 POT_CHANNEL(到Pin Module修改名字)

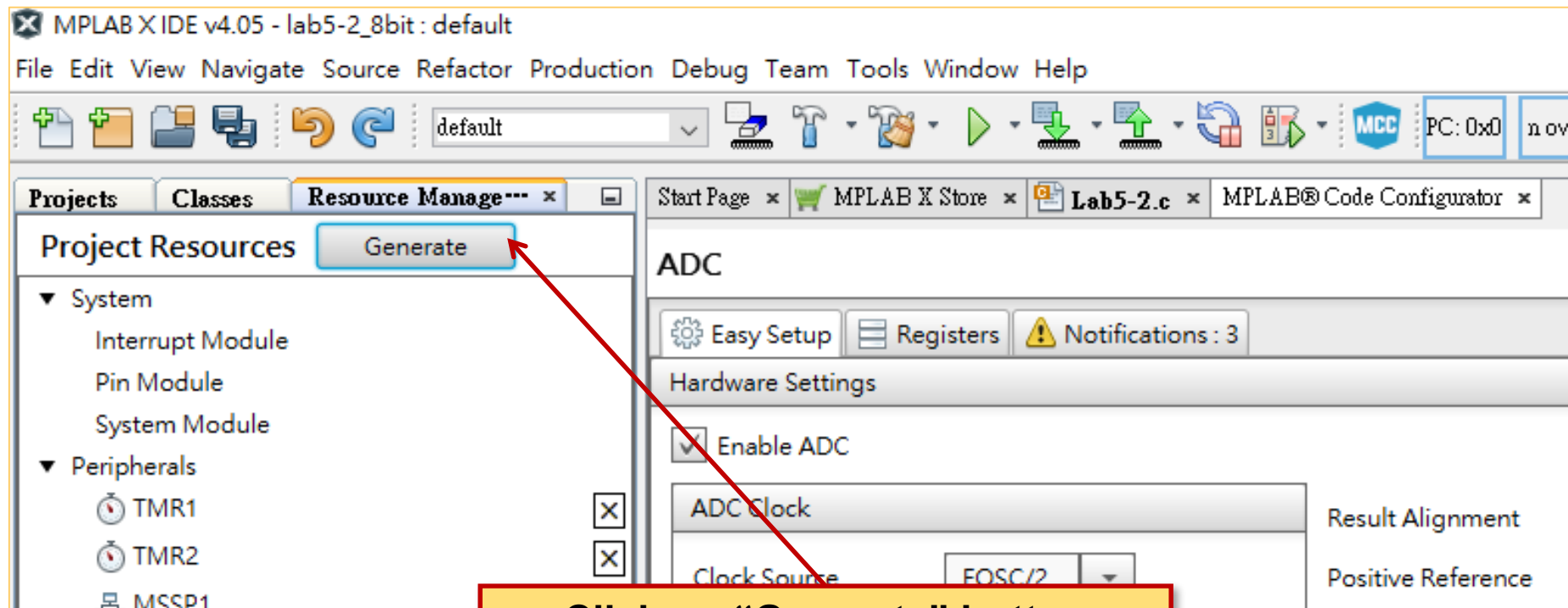
ADC Configuration:

- ☒ Enable ADC
- ADC Clock:**
 - Clock Source: FOSC/2
 - Acquisition Time: 0
 - 1 TAD: 2.0 us
 - Sampling Frequency: 37.037 kHz
 - Conversion Time: = 11.5 * TAD = 23.0 us
- ☐ Enable ADC Interrupt
- Result Alignment:** right
- Positive Reference:** VDD
- Negative Reference:** VSS
- Auto-conversion Trigger:** CCP5

Pin	Channel	Custom Name
Internal Channel	CTMU	channel_CTMU
Internal Channel	FVRBuf2	channel_FVRBuf2
Internal Channel	DAC	channel_DAC
RA0	AN0	POT_CHANNEL

PIC18 Lab5-2

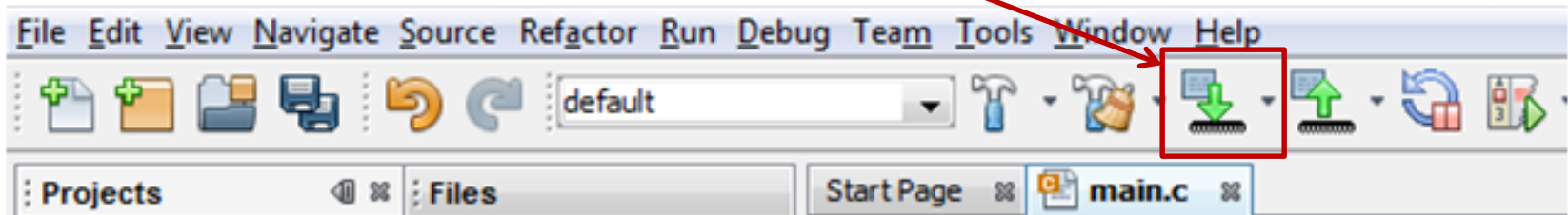
- 按下 **Generate** 來產生周邊設定的相關檔案



Click on “Generate” button

PIC18 Lab5-2

- 按下“編譯及燒錄”圖示來驗證結果



- 有沒有聽到蜂鳴器的音樂聲，調整 VR1 看看旋律是否改變？

PIC18 Lab5-2 結論

- 我們使用 **SPI** 介面來讀寫 **EEPROM** 的音樂資料
- 我們使用了 **EPWM** 模組來撥放音樂
- 我們利用 **ADC** 轉換可變電阻的電壓做為控制撥放音樂的速度控制

有問題嗎？



謝謝！



Trademarks

- The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, flexPWR, JukeBlox, KEELOQ, KEELOQ logo, Klear, LANCheck, MediaLB, MOST, MOST logo, MPLAB, OptoLyzer, PIC, PICSTART, PIC³² logo, RightTouch, SpyNIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
- The Embedded Control Solutions Company is a registered trademark of Microchip Technology Incorporated in the U.S.A.
- Analog-for-the-Digital Age, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, ECAN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, KlearNet, KlearNet logo, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mTouch, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, RightTouch logo, REAL ICE, SQI, Serial Quad I/O, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, 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.
- GestIC is a registered trademarks of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.
- All other trademarks mentioned herein are property of their respective companies.
- © 2014, Microchip Technology Incorporated, All Rights Reserved.