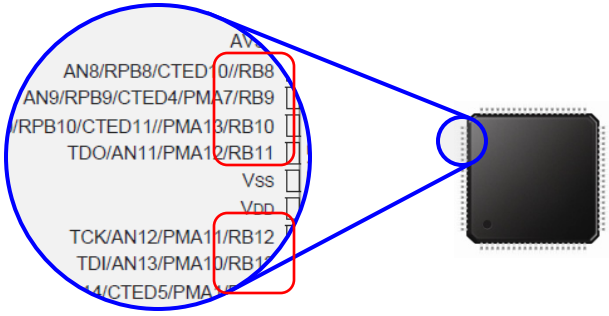
MICROCHIP

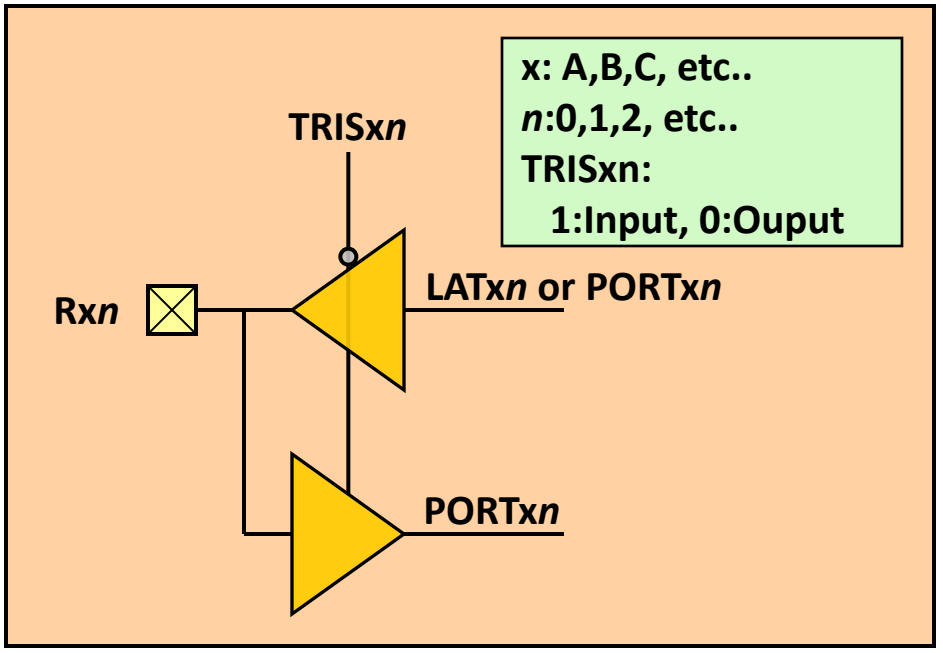
Regional Training Centers

Section 5
IO Port Architecture

I/O Port Block Diagram

- 32-Bits MCU的IO Port示意圖, 如圖所示。
- 所有的IO Port都有TRIS, LAT跟PORT特殊功能暫存器。TRIS用來設定IO的方向,要輸出時(Ex:控制LED)TRISx*n*必須設為"0" 。要輸入(Ex:讀取按鍵狀態)時TRISx*n*必須設為"1"。
- 設為輸出時,要輸出的狀態填入LATx*n*或PORTx*n*,對應的接腳Rx*n*就會有輸出。
- 設為輸入時,可以讀取PORTx*n*取得外部的實際狀態。

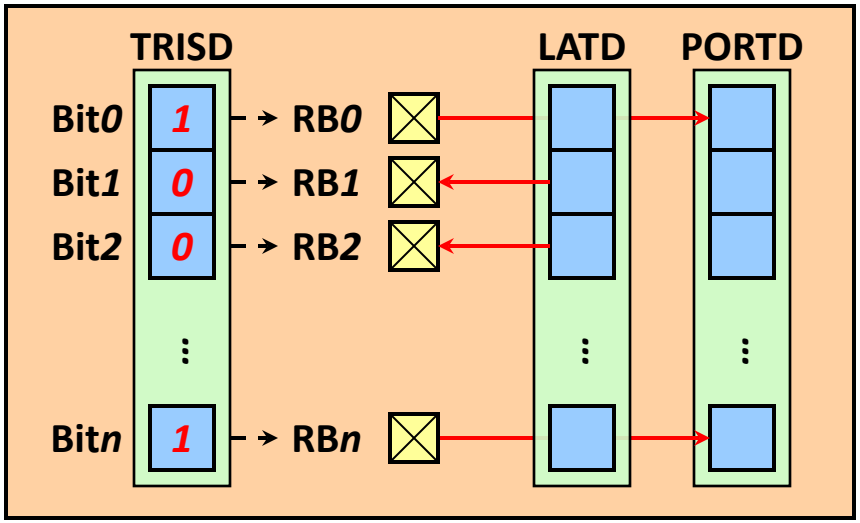




MICROCHIP
Regional Training Centers

I/O Port Manipulation

- TRISx, LATx跟PORTx特殊功能暫存器都是一個32-Bits的暫存器, 暫存器中每個Bit都控制著對應的I/O接腳。
- 以IO Port D為例,TRISD, LATD, PORTD的Bit2,控制RD2接腳。
- 要用RD8控制LED時, 必須先把TRISD的Bit2設為0(輸出), 然後把要輸出的狀態,填入LATD的Bit2。
- 要用RD2讀取按鍵狀態時,則必須把TRISD的Bit2設為1(輸入), 然後就可以從PORTD的Bit2取得外部的狀態。



XC32 I/O Port Function & Macro

- MPLAB XC32提供許多IO Function可供使用:
- mPORTxSetPinsDigitalIn( ); // 設定對應的I/O Portx Bit為數位輸入
  mPORTxSetPinsDigitalOut( ); // 設定對應的I/O Portx Bit為數位輸出
  mPORTxSetPinsAnalogIn( ); // 設定對應的I/O Portx Bit為類比輸入
  mPORTxSetPinsAnalogOut( ); // 設定對應的I/O Portx Bit為類比輸出
  mPORTxSetBits( ); // 設定I/O為High, Portx Bit = 1
  mPORTxClearBits( ); // 清除I/O為Low, Portx Bit = 0
  mPORTxToggleBits( ); // 反向I/O Portx Bit(“1”->”0”,”0”->”1”)
  mPORTxReadBits( ); // 讀取I/O Port Bit的資料…

XC32 I/O Code Example

- I/O設定為數位輸出, 輸出操作範例

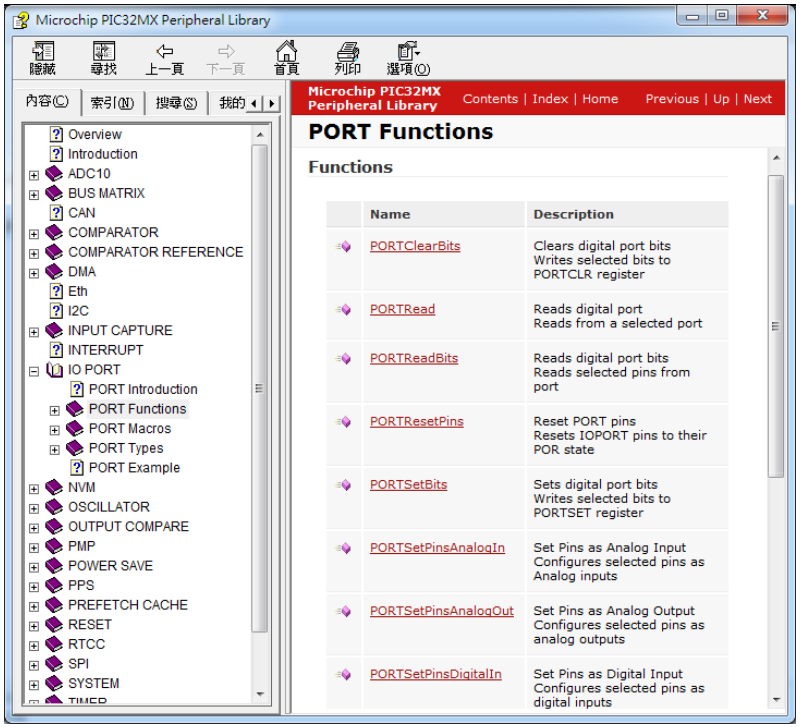
```
mPORTDSetPinsDigitalOut( BIT_2 | BIT_5 );
mPORTDSetBits( BIT_2 );
mPORTDClearBits( BIT_2 );
mPORTDToggleBits( BIT_2 | BIT_5 );
```

- I/O設定為數位輸入範例

```
mPORTFSetPinsDigitalIn(BIT_4);
```

- *BTValue* =

```
mPORTFReadBits(BIT_4) ;
```



- 詳細說明請參閱說明文件

Microchip PIC32MX Peripheral Library.chm。

C:\Program Files (x86)\Microchip\xc32\v1.32\docs\pic32-lib-help\

MICROCHIP
Regional Training Centers

Lab1 – GPIO Output

- 在Lab1的程式基礎上,嘗試加入IO Output的控制程式。
- 請嘗試控制TRISx*n*, LATx*n*, PORTx*n*, 或使用XC32的I/O函數, 讓LED(D8 , RD2)接腳可以不斷的轉態(Toggle, 1<->0) 。時間間隔為500 mS 。
- 前面所提過的, 要正確的存取SFRs必須 include MCU的標頭檔 Ex:#include <p32xxxx.h> 。使用XC32的函數庫時, 必須include 函數庫的標頭檔 Ex:#include <plib.h> 。
- 操作SFRs時,
  - 可以使用暫存器, 結構來存取

    Ex:TRISD &= ~0x0002; // PORTD Bit2設為輸出,
    LATD |= 0x0002; // PORTD Bit2 輸出High Level。
    Ex:TRISDbits.TRISD2 = 0;
    LATDbits.LATD2= 1;
  - 或透過XC32的I/O函數

    Ex:mPORTDSetPinsDigitalOut(BIT_2);
    mPORTDToggleBits(BIT_2);

Lab1 – GPIO Output *Step1*

- 程式要如何設計才能達到Toggle的功能？

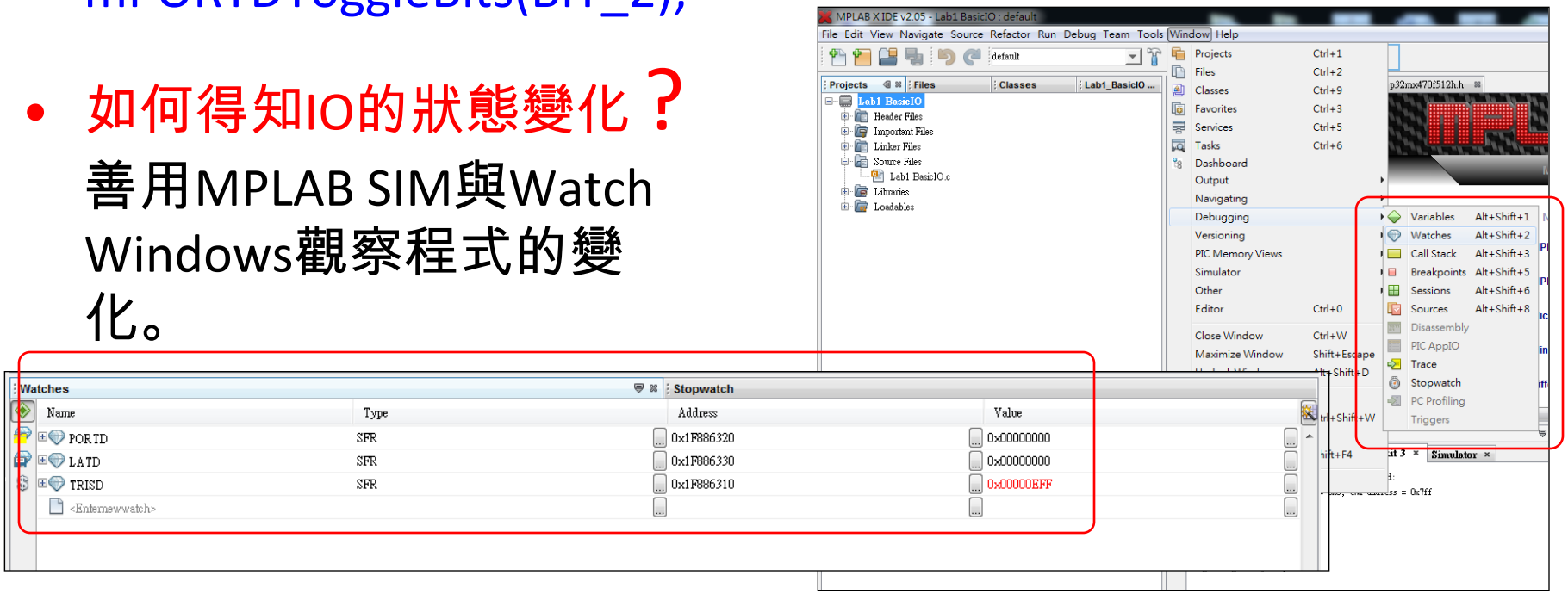
  LATD = LATD ^ 0x0004; // 利用XOR

  LATDbits.LATD2= !LATDbits.LATD2; // 利用NOT

  mPORTDToggleBits(BIT_2);

- 如何得知IO的狀態變化？

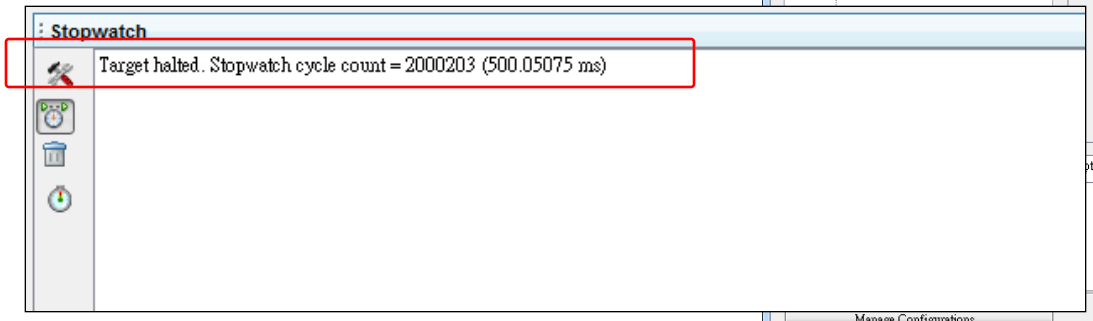
  善用MPLAB SIM與Watch Windows觀察程式的變化。

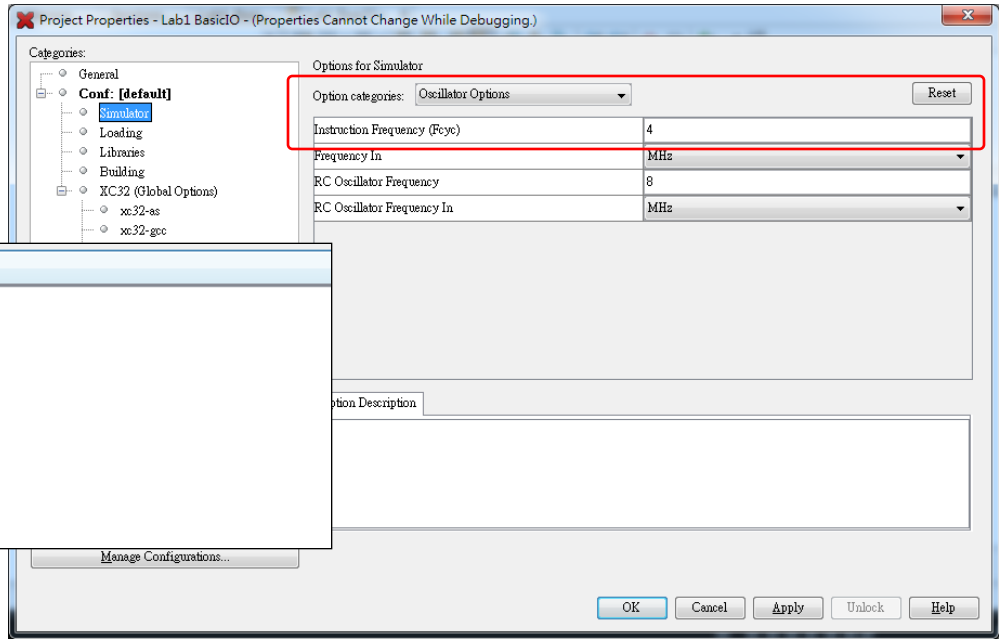


Lab1 – GPIO Output *Step2*

- 如何控制IO每次Toggle的間隔時間?

  控制for迴圈來決定Delay的時間。
  時間量測可以利用Simulator下的Stopwatch功能來測量迴圈的時間, 計算前必須先設定預計的系統時脈。此處設定Processor Frequency 為 4MHz (File\Project Properties\ Simulator)。





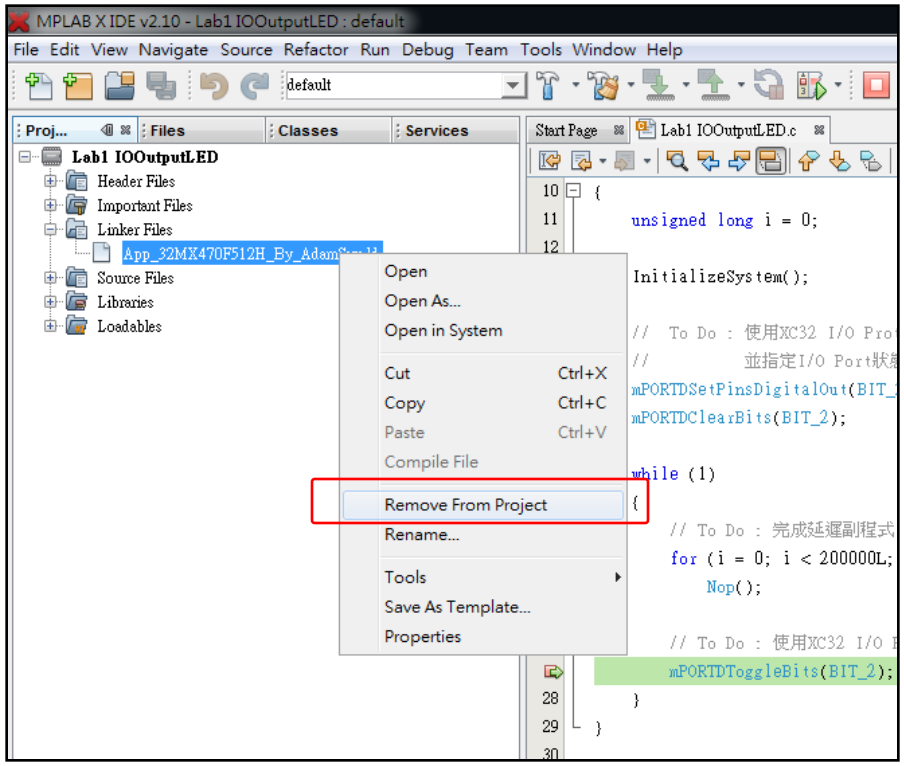
Lab1 – GPIO Output *Step3*

- 為何Simulator無法正常正常?

  實驗用的專案是使用Bootloader的架構。在此架構下無法使用Simulator功能。

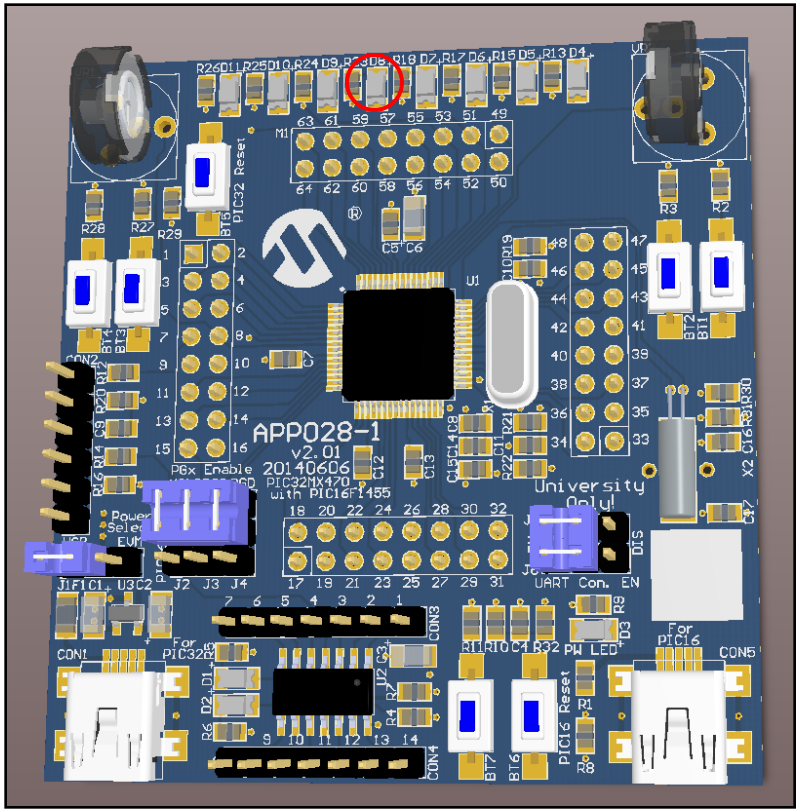
  必須先移除Linker Script Files (App_32MX470F512H_By_AdamSyu.ld), 才能進行Simulator。

- 要燒錄程式時, 記得要再把Linker Script Files加回去, 重新編譯, 燒錄。



Lab1 – GPIO Output *Result!*

- 使用Bootloader將程式燒錄進APP028-1。觀察LED的動作狀態。驗證看看LED(D8, RD2)是否每500mS亮/滅一次(1 Hz)?



MICROCHIP
Regional Training Centers

Lab2 – GPIO Input

- 在Lab1的程式基礎上,嘗試加入IO Input的控制程式。
- 請嘗試控制TRISx*n*, LATx*n*, PORTx*n*, 或使用XC32的I/O函數, 來讓BT1(RF4)接腳可以獲得按鍵的狀態, 並將BT1的狀態直接反映在LED上(D11, RD5)。 不按鍵時LED熄滅, 按鍵時亮起。

- 操作SFRs時,
  - 可以使用暫存器, 結構來存取

    Ex:TRISF |= 0x0010; // PORTF Bit4設為輸入,

    BTValue = PORTF & 0x0010; // 讀取PORTF Bit4狀態。

    Ex:TRISFbits.TRISF4 = 1;

    BTValue = PORTFbits.RF4;
  - 或透過XC32的I/O函數

    Ex:mPORTFSetPinsDigitalIn(BIT_4);

    BTValue = mPORTFReadBits(BIT_4);

MICROCHIP
Regional Training Centers

Lab2 – GPIO Input *Step1*

- 程式要如何設計才能讀取按鍵的狀態？

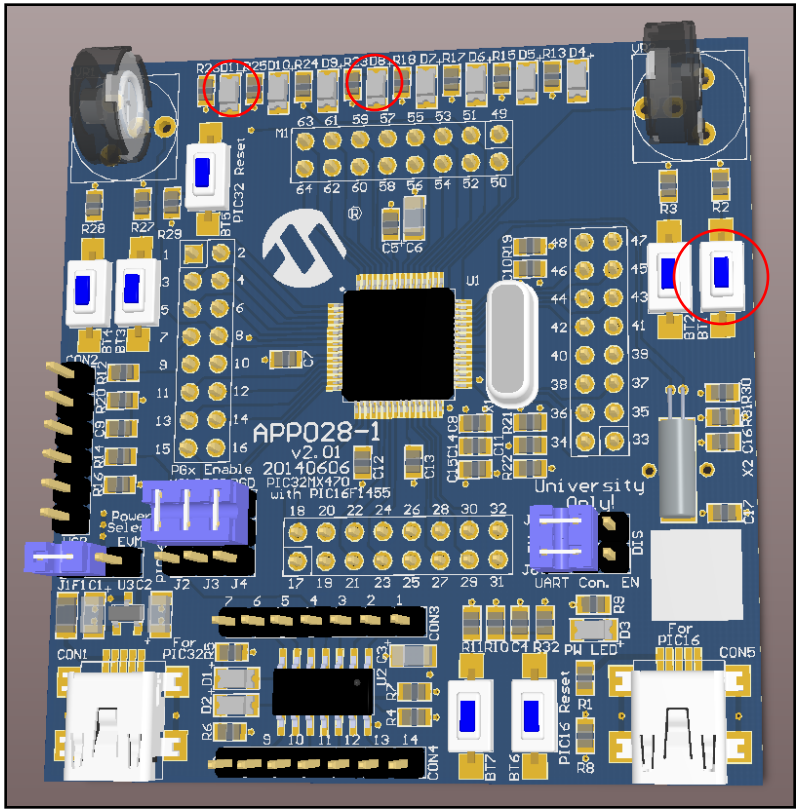
```
if ( PORTF & 0x0010 == 0x0010 ) { … }
if ( PORTFbits.RT4 ) { … }
if ( mPORTFReadBits(BIT_4) ) { … }
```

- 如何得知IO的狀態變化？

記住要善用MPLAB SIM與Watch Windows觀察程式的變化。
要使用Simulator時, 要先移除Linker Script Files。要燒錄程式時, 記得要再把Linker Script Files加回去,重新編譯, 燒錄。

Lab2 – GPIO Input *Result!*

- 使用Bootloader將程式燒錄進APP028-1。觀察LED的動作狀態。驗證看看LED(D8, RD2)是否每500mS亮/滅一次(1 Hz)?
- 按下按鈕時, 對應的LED是否會反映出按鈕的狀態, 還是感覺動作怪怪的, 不是很正常?



Lab2 – GPIO Input *fix Step1*

- 按鈕偵測感覺不是很正常?

  觀察程式架構, 按鈕狀態的取得, 是在每次Delay後才進行, 時間間隔太長, 當然會感覺按鈕很遲鈍。

```
while (1)
{
        for ( i = 0; i < 200000L; i++ )
                Nop();
        Toggle LED;
        Button Detection;
}
```

Lab2 – GPIO Input *fix Step2*

- 如何修改程式架構, 加快按鈕偵測速度?

  改以狀態機的架構來改寫程式, 也就是說讓程式不會傻傻的苦等Delay完成, 藉此提高按鈕偵測速度。

```
while (1)
{
    if( i++ >= 100000L )
    {
        i = 0;
        Toggle LED;
    }
    Button Detection;
}
```

程式架構不同,
延遲的計數值需要調整。

Lab2 – GPIO Input *fix Result!*

- 使用Bootloader將程式燒錄進APP028-1。觀察LED的動作狀態, 閃爍頻率是否為1MHz?
- 按下按鈕時, 對應的LED是否會反映出按鈕的狀態?
- 程式修改架構後, 按鈕反應是否變靈敏了?