



MICROCHIP

Regional Training Centers

Section 8
Interrupt

What's Interrupt ?

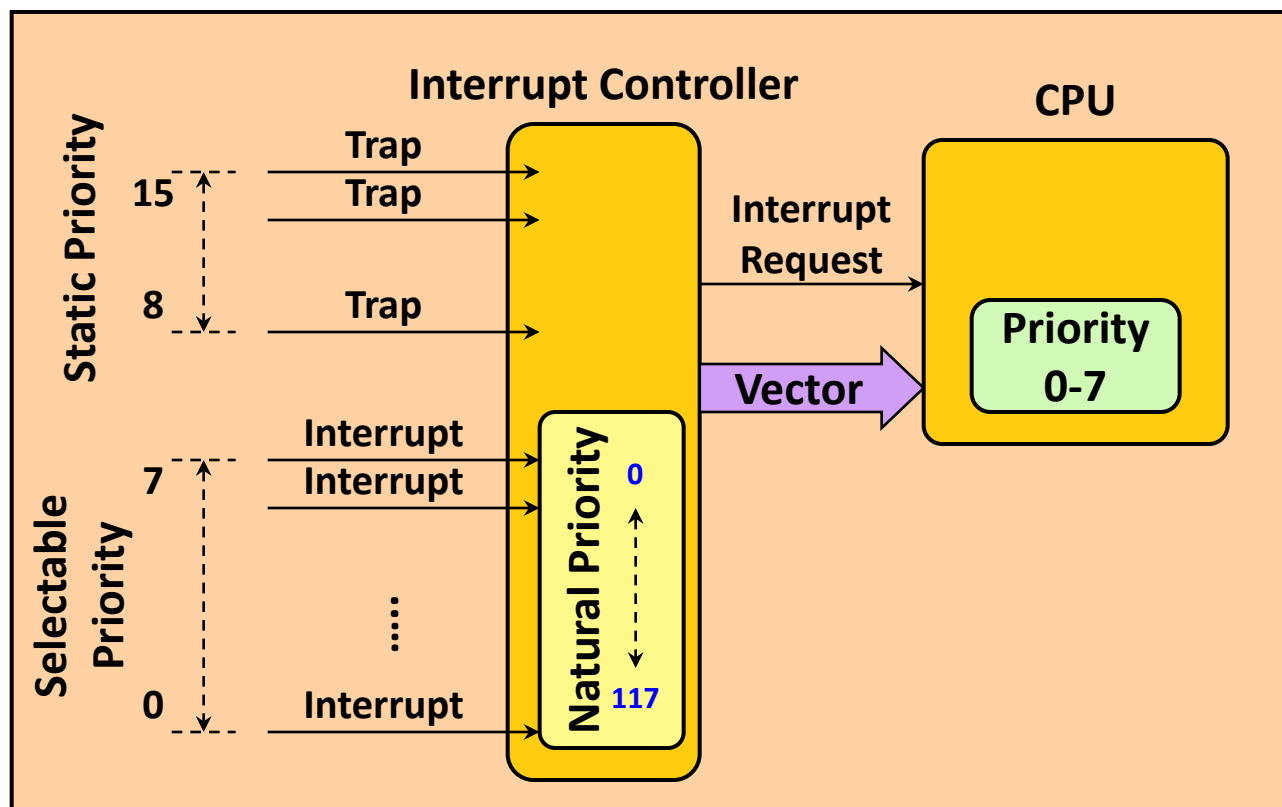
- 在一般的情況下,CPU是按照程式的流程依序地執行。但如果此時某些周邊希望取得CPU的服務,此時就可以透過中斷,打斷目前正在執行的程式片斷,跳到該中斷的服務常式中 (ISR, Interrupt Service Routine)。
- 中斷的來源,有許多可能如Timer, ADC, UART, SPI, etc., 都可以打斷CPU。這些中斷來源在16-Bits MCU中被區分為兩大類:
Interrupt:一般周邊所產生的中斷,可以被致能或禁能如上述的Timer, ADC, UART, SPI, 外部中斷, etc..
Trap:特殊的中斷類型或稱不可遮罩中斷(NMI), 這類型的中斷發生,代表著某些可能危害程式運行的狀況出現,如除"0",Stack Overflow / Underflow等。

What's Interrupt ?

- 中斷條件的成立,通常必須要滿足幾個要素:
 - **中斷需求(Interrupt Request):**
在MCU中, 周邊會透過設定中斷旗標(xxIF)來提出中斷需求。
 - **中斷致能(Interrupt Enable):**
使用者可以透過中斷致能位元(xxIE)來決定是否接受周邊的中斷需求。
簡單的說,就是當xxIF跟xxIE都為1時, 才能打斷CPU的執行, 跳到中斷服務常式中。
- Trap的發生代表著某些可能危害程式運行的狀況出現,因此Trap是無法拒絕的,CPU一定必須接受。
 - 16-Bits dsPIC MCU定義以下幾種Trap :
Stack Error, Oscillator Fail Error,
Address Error, Math Error, DMA Error。

16-Bits Interrupt Architecture

- 16-Bits的中斷架構,如圖所示。16-Bits MCU支援巢狀中斷,也就是中斷可以設定優先權,優先權高的可以打斷低的。
- 各周邊的中斷優先權可以個別設定,範圍為0~7(IPCx)。
- Trap優先權則為8~15,每個Trap都有固定的優先權。
- CPU也擁有自己的優先權,範圍是0-7。



(Trap 的優先權都比 CPU 高)

16-Bits Interrupt Architecture

- 在16-Bits MCU的中斷架構中, 要使中斷條件成立, 除了前述的xxIF跟xxIE都必須為1以外, 中斷(Interrupt)的優先權還必需大於CPU的優先權才可以成立。
- 因為CPU最大的優先權可以設定到7, 但Trap從8開始, 這也說明了為何CPU無法拒絕Trap。
- CPU的優先權設定位元(IPL)在狀態暫存器的Bit7-Bit5(SRL <7:5>)。中設定。bits 透過更改CPU的優先權設定位元(IPL)的值, 可以抑制較低優先權的中斷發生(IPL=7, Disable All Interrupt)。當中斷發生時, 舊的CPU優先權(IPL)會被自動存到堆疊裡保存起來。
- 當兩個相同優先權的中斷“同時”發生時,CPU要先服務誰?

Natural Priority

- 自然優先權(Natural Priority)是仲裁CPU該先服務那個中斷的最後手段。
- 當數個優先權相同的中斷“同時”發生時,由於自訂的優先權無法比較出高低,因此僅能透過自然優先權來仲裁。
- 在16-Bits MCU中自然優先權,是根據中斷在中斷向量表中安排的位置而定,位址越低的優先權越高,位址越高的優先權越低。

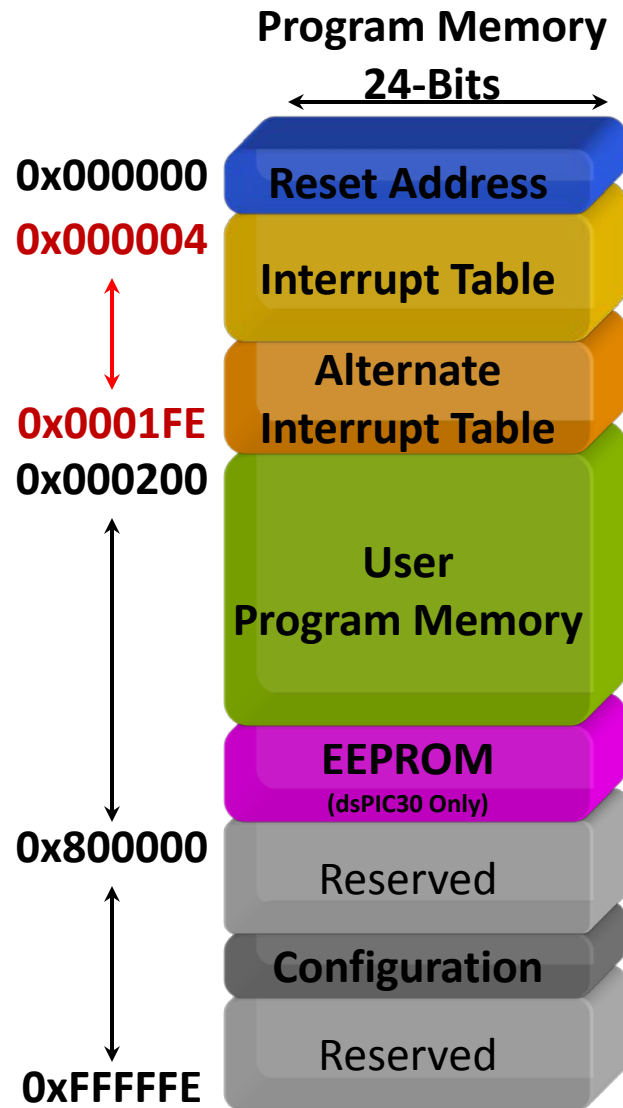
Vector Number	IVT Address	AIVT Address	Interrupt Source
0	0x000004	0x000104	Reserved
1	0x000006	0x000106	Oscillator Failure
2	0x000008	0x000108	Address Error
3	0x00000A	0x00010A	Stack Error
4	0x00000C	0x00010C	Math Error
5	0x00000E	0x00010E	DMA Error
6	0x000010	0x000110	Reserved
7	0x000012	0x000112	Reserved
8	0x000014	0x000114	INT0 – External Interrupt 0
9	0x000016	0x000116	IC1 – Input Capture 1
10	0x000018	0x000118	OC1 – Output Compare 1
11	0x00001A	0x00011A	T1 – Timer1
12	0x00001C	0x00011C	DMA0 – DMA Channel 0
13	0x00001E	0x00011E	IC2 – Input Capture 2
14	0x000020	0x000120	Reserved
15	0x000022	0x000122	Reserved
16	0x000024	0x000124	Reserved
17	0x000026	0x000126	Reserved
18	0x000028	0x000128	Reserved
19	0x00002A	0x00012A	Reserved
20	0x00002C	0x00012C	Reserved
21	0x00002E	0x00012E	Reserved
22	0x000030	0x000130	Reserved
23	0x000032	0x000132	Reserved
24	0x000034	0x000134	Reserved
25	0x000036	0x000136	Reserved
26	0x000038	0x000138	Reserved
27	0x00003A	0x00013A	Reserved
28	0x00003C	0x00013C	Reserved
29	0x00003E	0x00013E	Reserved

Reset – GOTO Instruction	0x000000
Reset – GOTO Address	0x000002
Reserved	0x000004
Oscillator Fail Trap Vector	
Address Error Trap Vector	
Stack Error Trap Vector	
Math Error Trap Vector	
DMA Error Trap Vector	
Reserved	
Reserved	
Interrupt Vector 0	0x000014
Interrupt Vector 1	
~	
~	
~	
Interrupt Vector 52	0x00007C
Interrupt Vector 53	0x00007E
Interrupt Vector 54	0x000080
~	
~	
~	

Interrupt Vector Table (IVT)⁽¹⁾

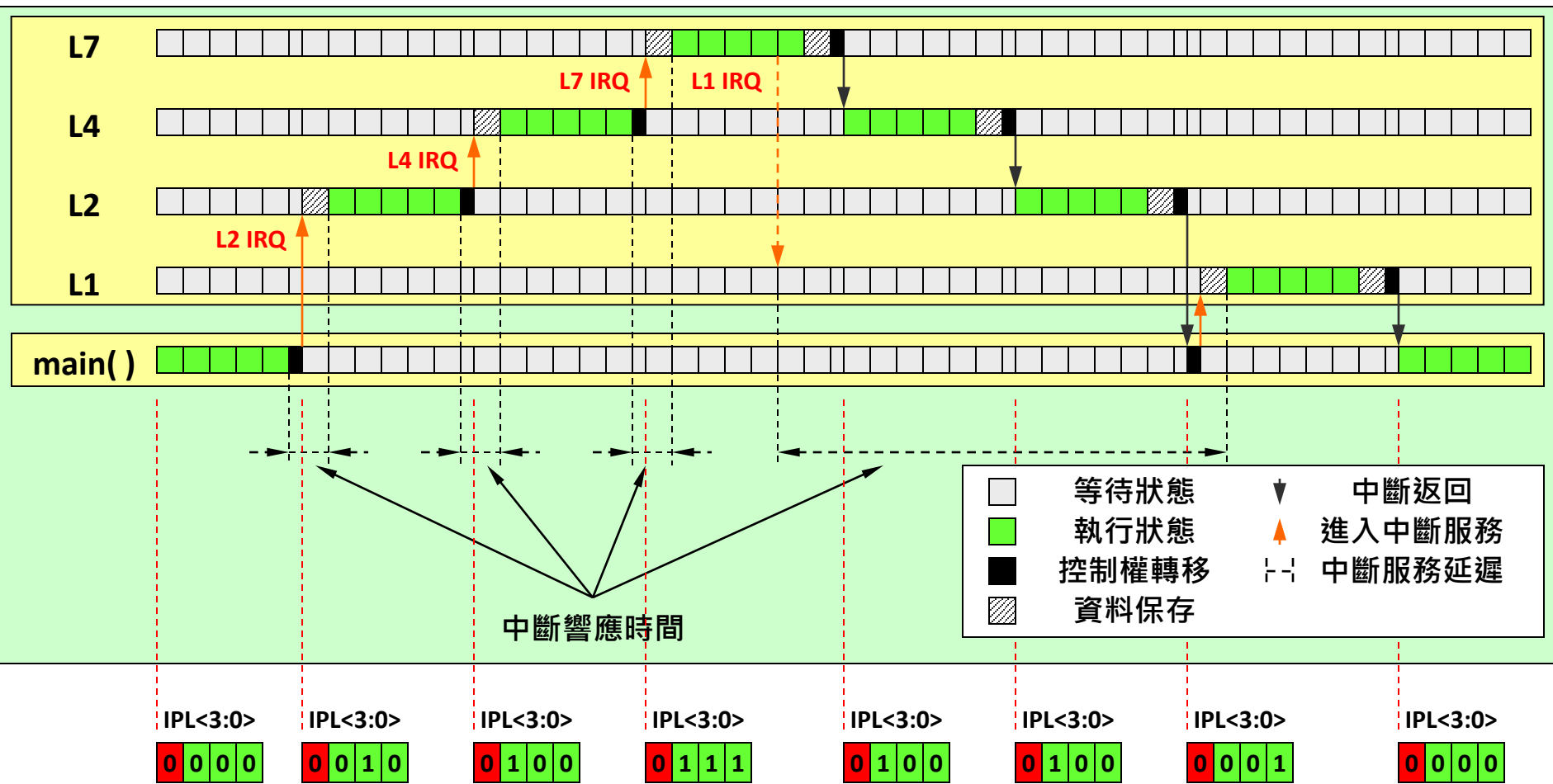
Interrupt Vector Table

- 16-Bits MCU將中斷向量表(IVT), 設定在程式記憶體中0x000004-0x0001FE的區域。當中斷服務常式宣告成功後, Linker就會將中斷服務常式的進入點, 填入中斷向量表中。
- 中斷發生時, 如果中斷被致能, CPU就會先到中斷向量表(IVT)中取出, 對應中斷的中斷服務常式的進入點, 接著跳躍到ISR中。





Interrupt Flow



Coding for Interrupt

- 一個完整的中斷架構程式片斷, 必須至少有三個部份:
- 中斷服務常式(Interrupt Service Routine):
中斷需求被接受後, CPU會跳至ISR中執行程式,因此必須針對所啟用的中斷設計ISR。
例如:Timer計數時, 要Toggle LED。
在MPLAB XC16中, 透過 `__attribute__((interrupt, auto_psv))` 來指定ISR。
- 設定中斷優先權, 開啟中斷:
如開頭所說,中斷必須要被致能, 周邊發出需求時, 才會被CPU接受, 因此必須在啟用中斷時, 將對應的(xxIE)設為"1"。
- 初始化周邊模組,設定發出中斷需求的模式:
設定周邊模組要在什麼情形下, 發出中斷需求。
例如:ADC每次轉換完成後, 發出中斷需求。

Interrupt Support for XC16

- MPLAB XC16透過attribute來指定ISR。
Ex: void `__attribute__((interrupt, auto_psv))` **`_T1Interrupt`**(void)
{
 IFSObits.T1IF = 0;

}
- 宣告一個函式, 作為Timer1的中斷服務函式。
注意!中斷服務函式不可以有傳入跟傳回值, 所以都必須設定為void型態。
- 中斷服務函式, 一定要把對應的中斷旗標清除為零。在中斷的架構上, 中斷旗標會由硬體設為"1", 但不會自動清除, 必須透過軟體手動清除。
- 如何得知該中斷服務函式是Timer1的 ?

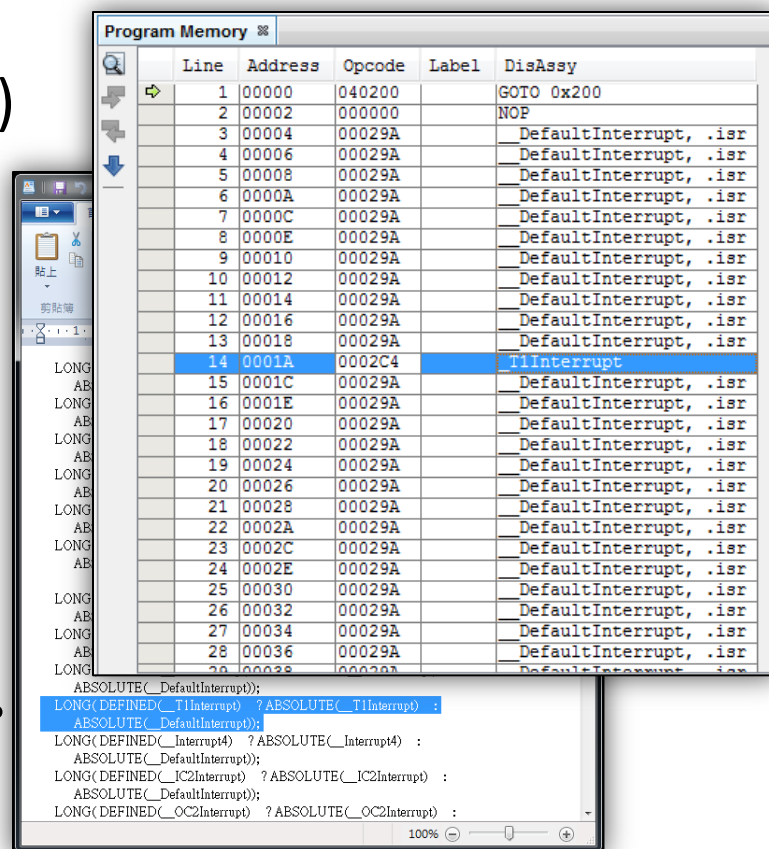
Interrupt Support for XC16

- MPLAB XC16中規定了所有中斷服務函式的名稱, 每個中斷都有專屬的名稱。宣告ISR時, 名稱必須一模一樣。
- 中斷服務函式的名稱可以開啟對應MCU的連結檔(Linker Script, *.gld)來尋找。
- 宣告的ISR名稱不同時會出現Warning的提示。如果宣告成功, Linker會自動將對應ISR的位址填入中斷向量表中。

透過MPLAB X IDE 功能表

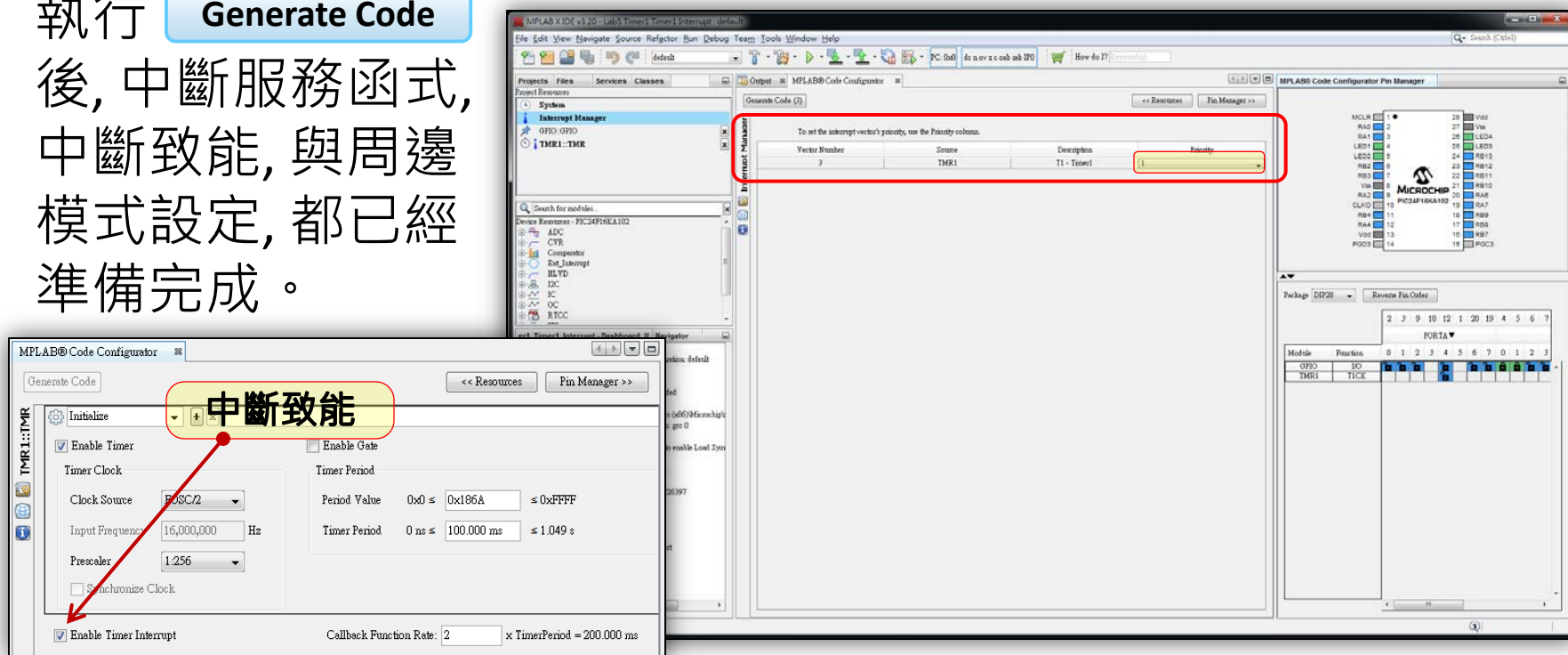
► **Window ► PIC Memory Views**

► **Program Memory** 看到安排細節。



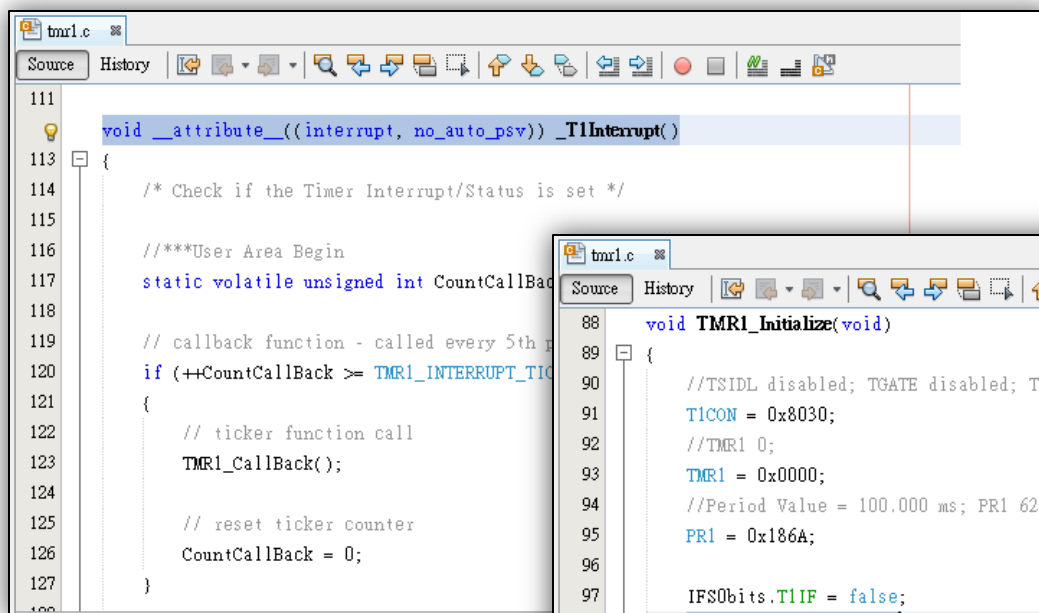
MCC's Interrupt Manager

- 在MCC只要開啟任何一個周邊的中斷($xxIE = 1$), 此時**Interrupt Manager**(中斷管理器)就會自動加入。在**Interrupt Manager**中可以檢視目前開啟的中斷, 也可以設定中斷的優先權。
- 執行 **Generate Code** 後, 中斷服務函式, 中斷致能, 與周邊模式設定, 都已經準備完成。

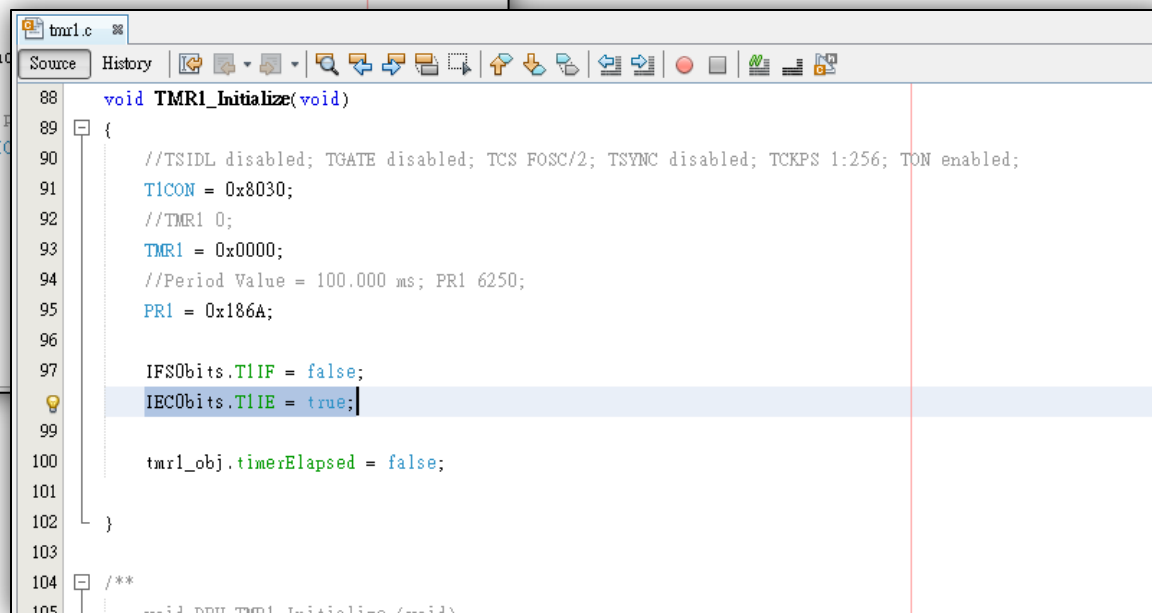


MCC's Interrupt Function

- 以Timer1為例, MCC會自動產生ISR。在初始化函式中也會一併開啟中斷(中斷致能)。



```
111 void __attribute__((interrupt, no_auto_psv)) _T1Interrupt()  
113 {  
114     /* Check if the Timer Interrupt/Status is set */  
115  
116     /***User Area Begin  
117     static volatile unsigned int CountCallBack  
118  
119     // callback function - called every 5th p  
120     if (++CountCallBack >= TMR1_INTERRUPT_TIC  
121     {  
122         // ticker function call  
123         TMR1_CallBack();  
124  
125         // reset ticker counter  
126         CountCallBack = 0;  
127     }  
128 }
```

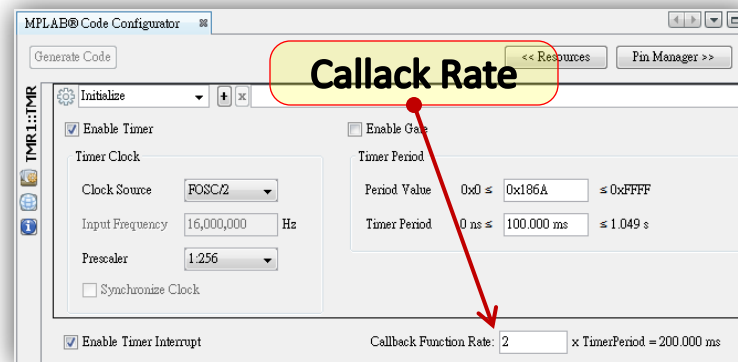


```
88 void TMR1_Initialize(void)  
89 {  
90     //TSIDL disabled; TGATE disabled; TCS FOSC/2; TSYNC disabled; TCKPS 1:256; TON enabled;  
91     T1CON = 0x8030;  
92     //TMR1 0;  
93     TMR1 = 0x0000;  
94     //Period Value = 100.000 ms; PR1 6250;  
95     PR1 = 0x186A;  
96  
97     IFS0bits.T1IF = false;  
98     IEC0bits.T1IE = true;  
99  
100     tmr1_obj.timerElapsed = false;  
101  
102 }  
103  
104 /**  
105 void DRV_TMR1_Initialize(void)
```

Timer's Callback Function

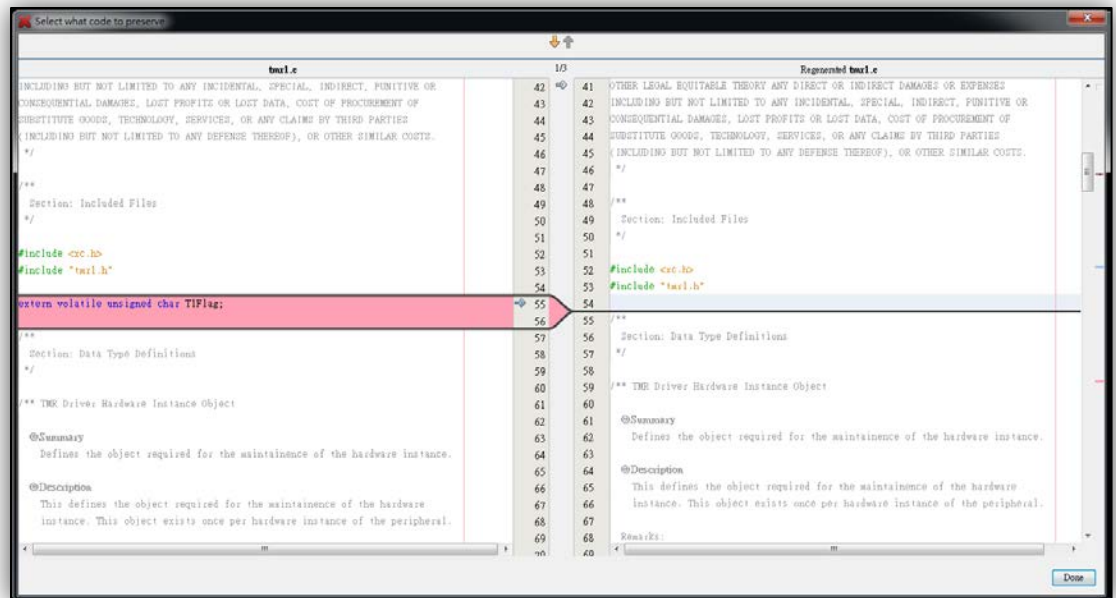
- 開啟Timer中斷後, MCC會自動產生Callback Function。Callback Function在這裡可以看成軟體Timer, 用來計算中斷的發生次數。
- 從Timer1的ISR中, 可以看出作用。透過累加CountCallBack變數, 可以計算中斷發生的次數, 達到設定的次數後, TMR1_CallBack() Function會被呼叫。
- 舉例來說, 如果Timer1設定為100ms發生中斷, Callback Rate設定為"2"。則TMR1_CallBack()每200ms(100ms x 2)會被呼叫一次。

```
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt()  
{  
    static volatile unsigned int CountCallBack = 0;  
  
    if (++CountCallBack >= TMR1_INTERRUPT_TICKER_FACTOR)  
    {  
        TMR1_CallBack();  
        CountCallBack = 0;  
    }  
    tmr1_obj.count++;  
    tmr1_obj.timerElapsed = true;  
    IFS0bits.T1IF = false;  
}
```



SubVersion's Version Control

- 透過MCC輔助的過程中, 我們會來回在MCC與程式碼之間反覆修改, 已達成目的。有時候也必須更進一步的修改MCC內部的程式碼。
- Version Control是一個輔助工具, 用來比較程式版本的差異。當我們修改了一份MCC的檔案後, 再次使用MCC產生新的檔案, 企圖覆蓋原先的檔案時, 就會出現提示。此時可以根據實際的需求, 決定最終要保留那份內容。也可透過插入或刪去, 來合併兩份檔案的內容。



volatile Qualification

- 如果在ISR與主程式中會共用到同一變數, 則此變數在宣告時, 必須加上**volatile**的關鍵字。

Ex: **volatile** unsigned int Ticks = 0;

- volatile**關鍵字是用來避免C Compiler在進行最佳化時, 將特定變數在最佳化的過程刪除, 導致程式的流程出錯。

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

一般變數的最佳化,
結果符合原意!

B=A;
C=B;
最佳化後
C=A;

SFR的最佳化, 結果變成
TMR1完全消失, 不符合原意!

TMR1=A;
C=TMR1;
最佳化後
C=A;



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

Lab5 Timer1 Interrupt

目標

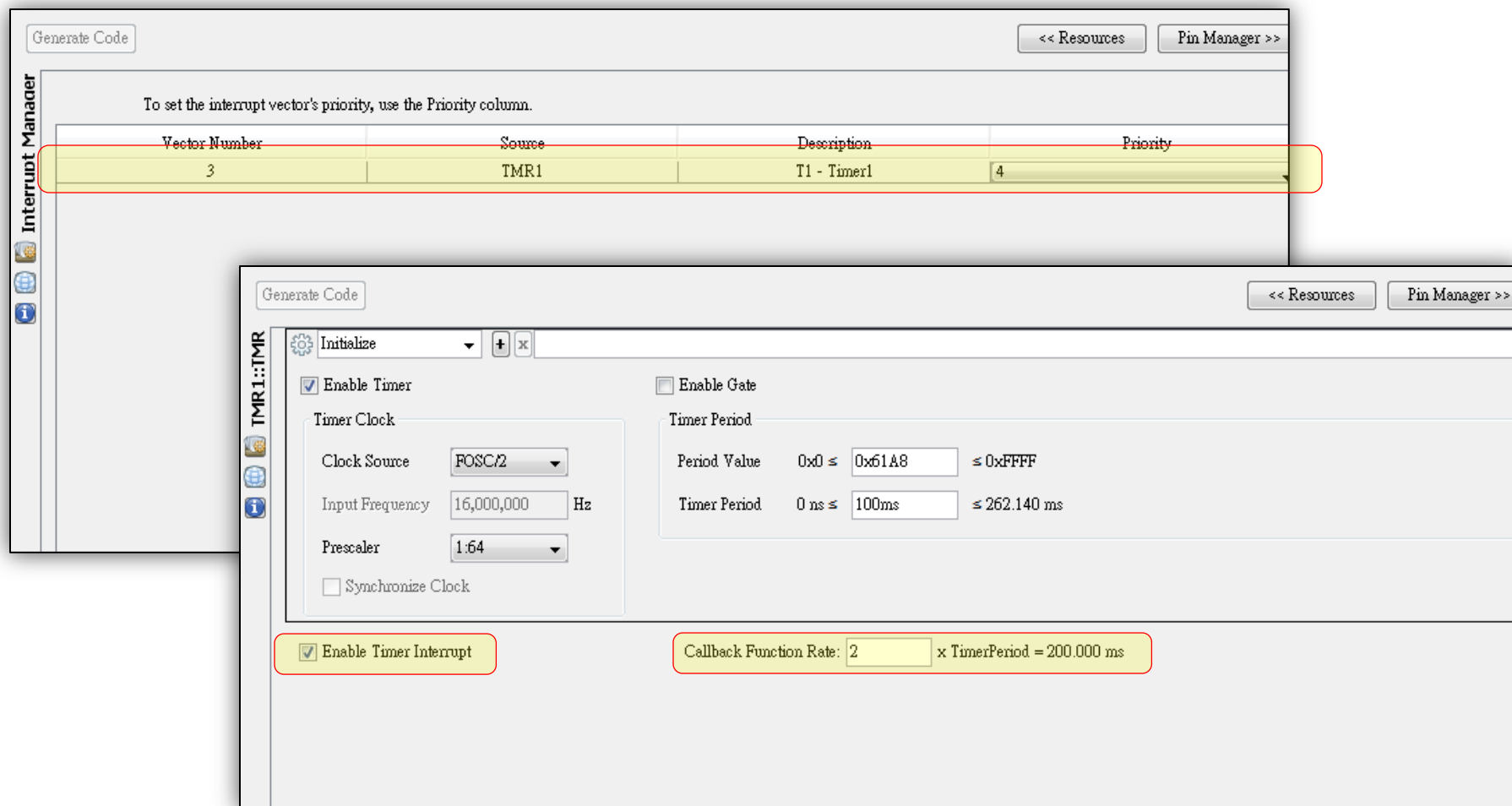
- 嘗試透過MCC的設定, 將Lab4的架構改為Timer Interrupt。
將Timer1的中斷優先權修改為"4", 並初始化Interrupt Manager。
 - 將原本Polling Timer1中斷旗標的架構, 改成Interrupt架構, 由中斷服務函式(ISR)來負責LED的控制。
 - 修改Timer1, 讓LED Toggle的時間間隔設定為200ms。
-
- 該如何開始？

Lab5 Timer1 Interrupt Step

-  開啟既有專案(<C:\Exercises\Exams\Lab5 Timer1 Interrupt.x>)
-  接著開啟開啟MCC, 修改**Timer1**資源, 開啟中斷需求, 並確認**Interrupt**資源的設定。

Lab5 Timer1 Interrupt

MCC's Setting



The image shows two overlapping screenshots from the Microchip Configurator software. The top screenshot displays the 'Interrupt Manager' window, which lists interrupt vectors. The bottom screenshot displays the 'TMR1::TMR' configuration window, showing various timer settings.

Interrupt Manager Window:

To set the interrupt vector's priority, use the Priority column.

Vector Number	Source	Description	Priority
3	TMR1	T1 - Timer1	4

TMR1::TMR Configuration Window:

Initialize

- ☒ Enable Timer
- ☐ Enable Gate

Timer Clock

- Clock Source: FOSC/2
- Input Frequency: 16,000,000 Hz
- Prescaler: 1:64
- ☐ Synchronize Clock

Timer Period

- Period Value: 0x0 ≤ 0x61A8 ≤ 0xFFFF
- Timer Period: 0 ns ≤ 100ms ≤ 262.140 ms

☒ Enable Timer Interrupt

Callback Function Rate: 2 x TimerPeriod = 200.000 ms

Lab5 Timer1 Interrupt

Code Example

tmr1.c

```
extern volatile unsigned char T1Flag;
```

```
void TMR1_CallBack(void)
{
    T1Flag = 1;
}
```

main.c

```
volatile unsigned char T1Flag = 0;
```

```
while (1)
{
    if (T1Flag)
    {
        T1Flag = 0;
        //LEDs Toggle
    }
}
```

Lab6 Multi-Timer Interrupt

目標

- 嘗試在Lab5的程式架構上, 多增加**Timer2**的中斷。將**Timer2**的中斷優先權修改為"5"。
- **Timer1**控制RB0(LED1), RB1(LED2)每200mS Toggle一次, **Timer2**控制RB14(LED3), RB15(LED4)每500mS Toggle一次。



© 2008 Microchip Technology Incorporated. All Rights Reserved. 2016 Winter Elite Slide 23

Lab6 Multi-Timer Interrupt

Code Example

tmr1.c

```
extern volatile unsigned char T1Flag;  
  
void TMR1_CallBack(void)  
{  
    T1Flag = 1;  
}
```

tmr2.c

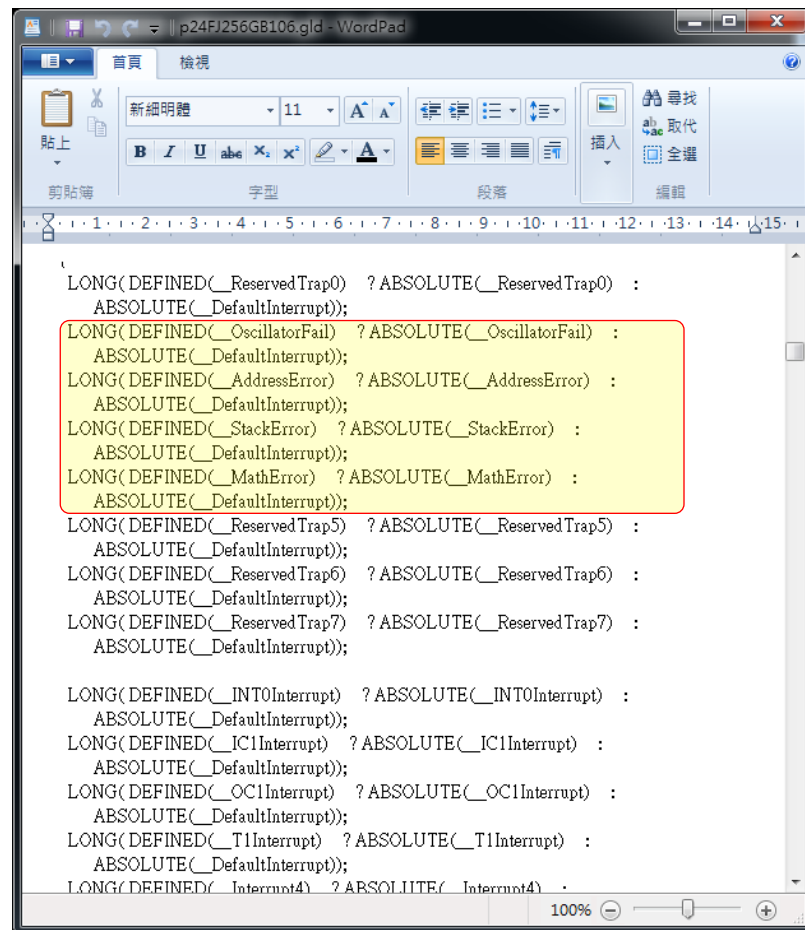
```
extern volatile unsigned char T2Flag;  
  
void TMR2_CallBack(void)  
{  
    T2Flag = 1;  
}
```

main.c

```
volatile unsigned char T1Flag = 0;  
volatile unsigned char T2Flag = 0;  
  
while (1)  
{  
    if (T1Flag)  
    {  
        T1Flag = 0;  
        //LEDs Toggle  
    }  
    if (T2Flag)  
    {  
        T2Flag = 0;  
        //LEDs Toggle  
    }  
}
```

Trap Service Routine

- Trap Service Routine, TSR沒有宣告時, 會套用DefaultInterrupt(), 當Trap發生時,直接Reset CPU。
- 我們可對每個Trap都改寫自訂的 Trap Service Routine,如此一來, 當Trap發生時,就會跳到各自的 Trap Service Routine,而不會直接Reset CPU。
- MPLAB XC16中對Trap Service Routine名字的定義,一樣可以在MCU的連結檔(Linker Script, *.gld)找到。
- **MCC目前尚未支援TSR產生。**



```
LONG(DEFINED(__ReservedTrap0) ? ABSOLUTE(__ReservedTrap0) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__OscillatorFail) ? ABSOLUTE(__OscillatorFail) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__AddressError) ? ABSOLUTE(__AddressError) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__StackError) ? ABSOLUTE(__StackError) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__MathError) ? ABSOLUTE(__MathError) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__ReservedTrap5) ? ABSOLUTE(__ReservedTrap5) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__ReservedTrap6) ? ABSOLUTE(__ReservedTrap6) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__ReservedTrap7) ? ABSOLUTE(__ReservedTrap7) :  
    ABSOLUTE(__DefaultInterrupt));  
  
LONG(DEFINED(__INT0Interrupt) ? ABSOLUTE(__INT0Interrupt) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__IC1Interrupt) ? ABSOLUTE(__IC1Interrupt) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__OC1Interrupt) ? ABSOLUTE(__OC1Interrupt) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__T1Interrupt) ? ABSOLUTE(__T1Interrupt) :  
    ABSOLUTE(__DefaultInterrupt));  
LONG(DEFINED(__Interrupt4) ? ABSOLUTE(__Interrupt4) :  
    ABSOLUTE(__DefaultInterrupt));
```


Trap Service Routine

- 目前尚未支援TSR產生。所以如果需要, 必須自行撰寫程式碼。宣告方式跟一般的ISR宣告方式一樣, 只是名字改成 *`_DefaultInterrupt`*。

E.g.:

Default Interrupt ISR

```
void __attribute__(( interrupt , auto_psv )) _DefaultInterrupt( void )  
{  
    while( 1 );  
}
```

- 改寫 `_DefaultInterrupt()` 有何好處?**

預設的 `_DefaultInterrupt()`, 在發生非預期中斷時, 會直接Reset, 這樣便成無法觀察MCU發生什麼事情, 只知道系統Reset。

如果改寫成上面的程式, 當程式停在 `while(1)` 時就知道有問題了, 這時候可以透過除錯工具, 去檢查相關暫存器, 確認問題。

Trap Service Routine

- 也可以針對個別的Trap撰寫TSR。宣告方式也跟一般的ISR宣告方式一樣, 都是修改名字即可。

E.g.:

Stack Error TSR

```
void __attribute__(( interrupt , auto_psv )) _StackError( void )  
{  
    while( 1 );  
}
```

- **改寫Stack Error Trap Service Routine有何好處?**

改寫成上面的程式, 當程式停在while(1)時可以馬上知道發生了Stack Error的錯誤。這時候可以透過除錯工具, 去檢查相關暫存器, 確認問題。