



MICROCHIP

Regional Training Centers

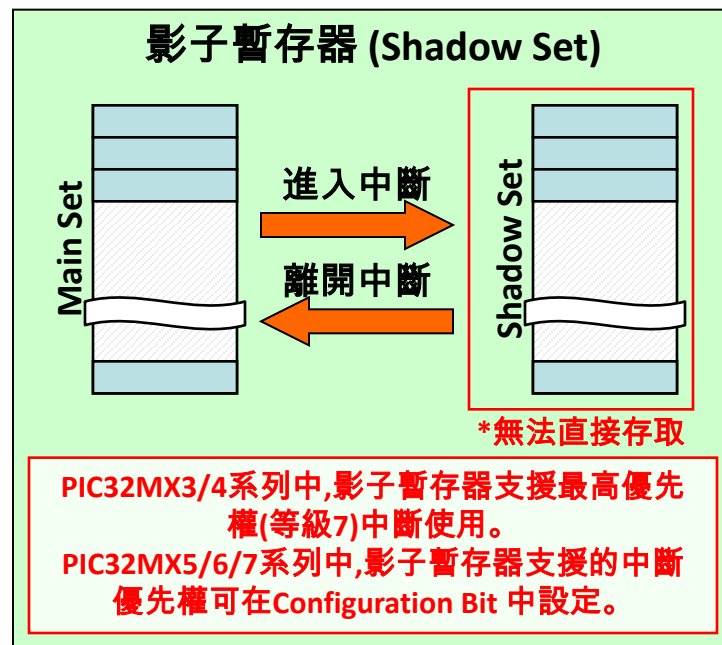
Section 8
Interrupt

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,外部中斷, etc.. etc.,都可以打斷CPU。
 - 中斷條件的成立,通常必須要滿足幾個要素:
 - 中斷需求(Interrupt Request):在MCU中,周邊會透過設定中斷旗標(xxIF)來提出中斷需求。
 - 中斷致能(Interrupt Enable):使用者可以透過中斷致能位元(xxIE)來決定是否接受周邊的中斷需求。
- 簡單的說,就是當xxIF跟xxIE都為1時,才能打斷CPU的執行,跳到中斷服務常式中。

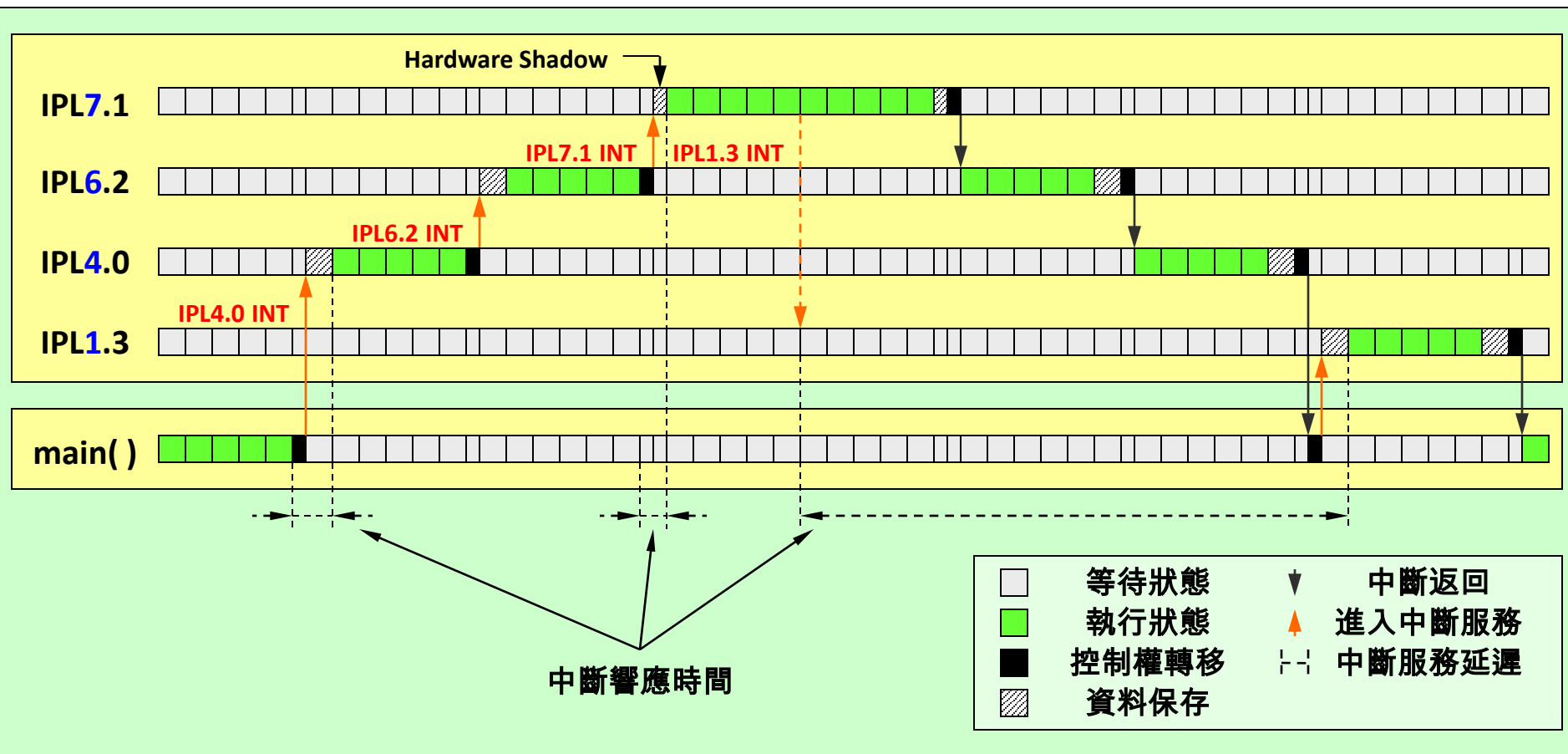
Interrupt Architecture

- PIC32MX470具強健的中斷硬體架構,內建向量中斷控制器,可提升中斷效能。
- 支援兩種中斷模式:
單一中斷向量模式(Single Vector)
(PIC32重置時預設值,通常使用在RTOS的架構上)
多重中斷模式(Multi Vector)
- 提供76個中斷來源,46個中斷向量。
- 提供7個主要優先權(Priorities),4個次要優先權(Sub- Priorities),及自然優先權。
- 對於高優先權之中斷,提供硬體的影子暫存器集(Shadow Set),加快中斷響應時間。



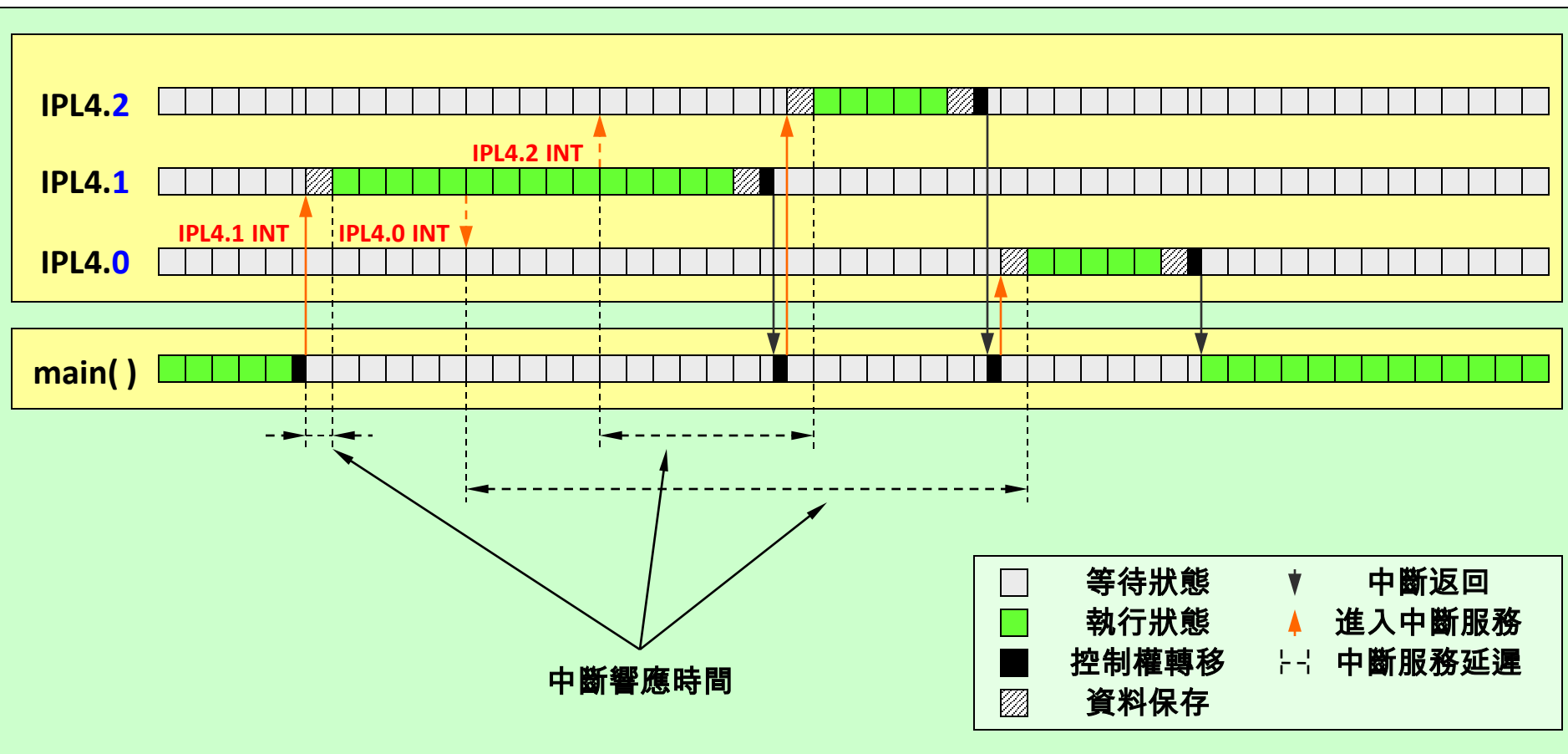
主要優先權 Priority

主要優先權較高者可強佔(Preemptive)執行權



次要優先權 Sub- Priority

中斷執行中不予理會，返回後再執行優先權最高者。



自然優先權 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PIC32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。

Interrupt Source ⁽¹⁾	IRQ #	Vector #	Interrupt Bit Location				Persistent Interrupt
			Flag	Enable	Priority	Sub-priority	
Highest Natural Order Priority							
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>	No
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>	No
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>	No
INT0 – External Interrupt	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>	No
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>	No
IC1E – Input Capture 1 Error	5	5	IFS0<5>	IEC0<5>	IPC1<12:10>	IPC1<9:8>	Yes
IC1 – Input Capture 1	6	5	IFS0<6>	IEC0<6>	IPC1<12:10>	IPC1<9:8>	Yes
OC1 – Output Compare 1	7	6	IFS0<7>	IEC0<7>	IPC1<20:18>	IPC1<17:16>	No
INT1 – External Interrupt 1	8	7	IFS0<8>	IEC0<8>	IPC1<28:26>	IPC1<25:24>	No
T2 – Timer2	9	8	IFS0<9>	IEC0<9>	IPC2<4:2>	IPC2<1:0>	No
IC2E – Input Capture 2	10	9	IFS0<10>	IEC0<10>	IPC2<12:10>	IPC2<9:8>	Yes
IC2 – Input Capture 2	11	9	IFS0<11>	IEC0<11>	IPC2<12:10>	IPC2<9:8>	Yes
OC2 – Output Compare 2	12	10	IFS0<12>	IEC0<12>	IPC2<20:18>	IPC2<17:16>	No
INT2 – External Interrupt 2	13	11	IFS0<13>	IEC0<13>	IPC2<28:26>	IPC2<25:24>	No
T3 – Timer3	14	12	IFS0<14>	IEC0<14>	IPC3<4:2>	IPC3<1:0>	No
IC3E – Input Capture 3	15	13	IFS0<15>	IEC0<15>	IPC3<12:10>	IPC3<9:8>	Yes
IC3 – Input Capture 3	16	13	IFS0<16>	IEC0<16>	IPC3<12:10>	IPC3<9:8>	Yes
OC3 – Output Compare 3	17	14	IFS0<17>	IEC0<17>	IPC3<20:18>	IPC3<17:16>	No
INT3 – External Interrupt 3	18	15	IFS0<18>	IEC0<18>	IPC3<28:26>	IPC3<25:24>	No
T4 – Timer4	19	16	IFS0<19>	IEC0<19>	IPC4<4:2>	IPC4<1:0>	No
IC4E – Input Capture 4 Error	20	17	IFS0<20>	IEC0<20>	IPC4<12:10>	IPC4<9:8>	Yes
IC4 – Input Capture 4	21	17	IFS0<21>	IEC0<21>	IPC4<12:10>	IPC4<9:8>	Yes
OC4 – Output Compare 4	22	18	IFS0<22>	IEC0<22>	IPC4<20:18>	IPC4<17:16>	No
INT4 – External Interrupt 4	23	19	IFS0<23>	IEC0<23>	IPC4<28:26>	IPC4<25:24>	No
T5 – Timer5	24	20	IFS0<24>	IEC0<24>	IPC5<4:2>	IPC5<1:0>	No
IC5E – Input Capture 5 Error	25	21	IFS0<25>	IEC0<25>	IPC5<12:10>	IPC5<9:8>	Yes
IC5 – Input Capture 5	26	21	IFS0<26>	IEC0<26>	IPC5<12:10>	IPC5<9:8>	Yes

程式架構

- 一個完整的中斷架構程式片斷,必須至少有1+3個部份:
- 啟動多重中斷向量模式:
`INTEnableSystemMultiVectoredInt();`
- 定義中斷服務常式與原型(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:Timer1 TMR1=PR1時, 要Toggle LED。
XC32使用`void __ISR(vector , ipln) ISR_Function_Name(void);`來設定ISR。
- 設定中斷主要與次要優先權並致能中斷:
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"。
XC32使用`ConfigIntxxxx();`來設定優先權。
- 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。XC32使用`Openxxxx();`設定周邊模組。

Interrupt Service Routine

- 使用XC32建立Timer的中段服務函式。

```
Ex: void __ISR( _TIMER_1_VECTOR, IPL4 ) ISR_Timer1( void )  
{  
    INTClearFlag( INT_T1 );  
    ....  
}
```

- 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。
- 為何知道是Timer1的中斷服務函式？

Interrupt Service Routine

- void **__ISR**(**vector** , **ipln**) ISR_Function_Name(void)

- vector中斷向量編號, 定義在對應的MCU標頭檔中。

C:\Program Files (x86)\Microchip\xc32\
v1.31\pic32mx\include\proc\p32mxxxxxxx.h

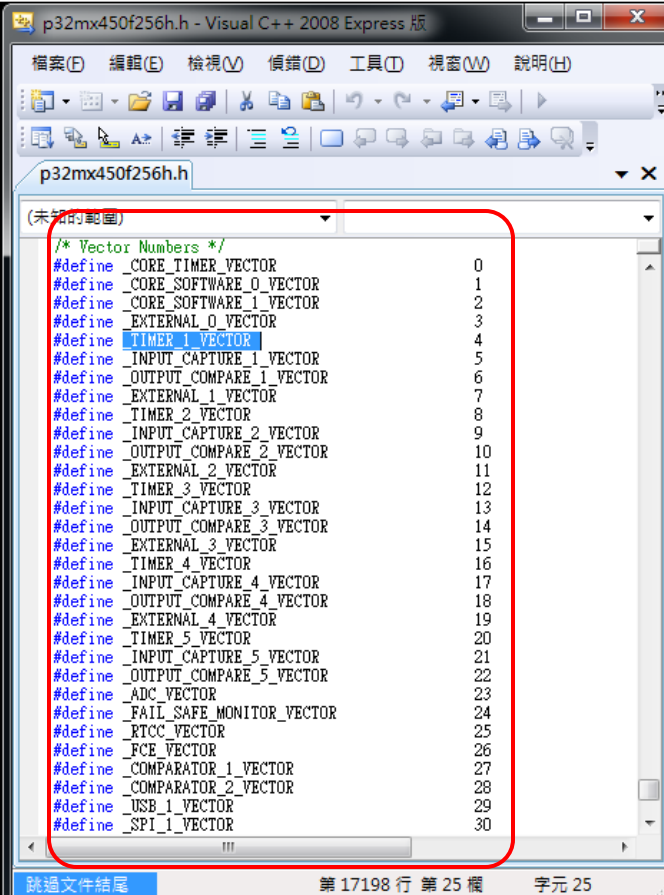
- ipln:中斷優先權,
(ipl1~ ipl7,ipl0為關閉中斷)。

***ISR所設定之ipln必須與
Configxxx();的Priority相同。**

- ISR_Function_Name:函數名稱。

- __ISR的詳細說明可參考

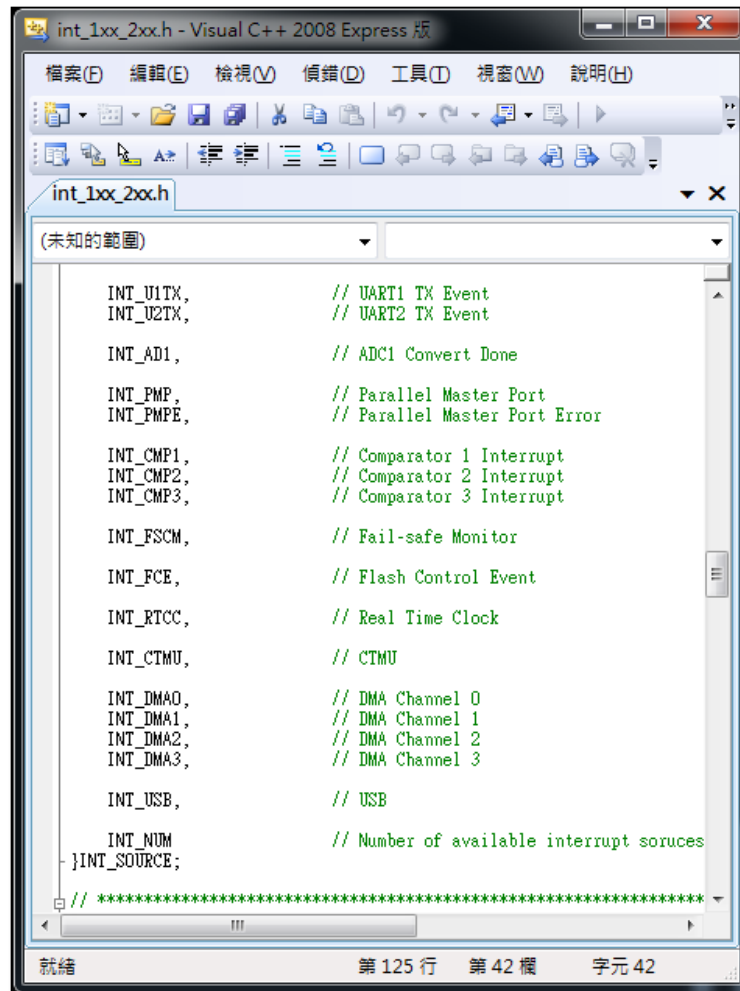
C:\Program Files (x86)\Microchip\xc32\
v1.32\docs\MPLAB-XC32-Users-Guide.pdf



```
/* Vector Numbers */
#define CORE_TIMER_VECTOR 0
#define CORE_SOFTWARE_0_VECTOR 1
#define CORE_SOFTWARE_1_VECTOR 2
#define EXTERNAL_0_VECTOR 3
#define TIMER_1_VECTOR 4
#define INPUT_CAPTURE_1_VECTOR 5
#define OUTPUT_COMPARE_1_VECTOR 6
#define EXTERNAL_1_VECTOR 7
#define TIMER_2_VECTOR 8
#define INPUT_CAPTURE_2_VECTOR 9
#define OUTPUT_COMPARE_2_VECTOR 10
#define EXTERNAL_2_VECTOR 11
#define TIMER_3_VECTOR 12
#define INPUT_CAPTURE_3_VECTOR 13
#define OUTPUT_COMPARE_3_VECTOR 14
#define EXTERNAL_3_VECTOR 15
#define TIMER_4_VECTOR 16
#define INPUT_CAPTURE_4_VECTOR 17
#define OUTPUT_COMPARE_4_VECTOR 18
#define EXTERNAL_4_VECTOR 19
#define TIMER_5_VECTOR 20
#define INPUT_CAPTURE_5_VECTOR 21
#define OUTPUT_COMPARE_5_VECTOR 22
#define ADC_VECTOR 23
#define FAIL_SAFE_MONITOR_VECTOR 24
#define RTCC_VECTOR 25
#define FCE_VECTOR 26
#define COMPARATOR_1_VECTOR 27
#define COMPARATOR_2_VECTOR 28
#define USB_1_VECTOR 29
#define SPI_1_VECTOR 30
```

Clear Interrupt

- 進入ISR後,必須清除中斷旗標,
MPLAB XC32可透過以下方式
清除中斷旗標:
直接寫入暫存器方式:
 $IFS0bits.T1IF = 0;$
使用XC32的清除函數:
`INTClearFlag(yyyy);`
yyyy為XC32為該週邊
所定義之列舉型別
(INT_SOURCE)之成員。
其定義可在int_3xx_4xx.h中找到。



```
int_1xx_2xx.h - Visual C++ 2008 Express 版
檔案(F) 編輯(E) 檢視(V) 偵錯(D) 工具(T) 視窗(W) 說明(H)
int_1xx_2xx.h
(未知的範圍)
INT_UTX,           // UART1 TX Event
INT_UTX,           // UART2 TX Event
INT_AD1,           // ADC1 Convert Done
INT_PMP,           // Parallel Master Port
INT_PMP,           // Parallel Master Port Error
INT_CMP1,          // Comparator 1 Interrupt
INT_CMP2,          // Comparator 2 Interrupt
INT_CMP3,          // Comparator 3 Interrupt
INT_FSCM,          // Fail-safe Monitor
INT_FCE,           // Flash Control Event
INT_RTCC,          // Real Time Clock
INT_CTMU,          // CTMU
INT_DMA0,          // DMA Channel 0
INT_DMA1,          // DMA Channel 1
INT_DMA2,          // DMA Channel 2
INT_DMA3,          // DMA Channel 3
INT_USB,           // USB
INT_NUM            // Number of available interrupt sources
}INT_SOURCE;
// *****
```

Priority Setup & Enable Interrupt

- XC32 透過以下方式來設定周邊中斷的開啟/關閉及主副優先權。

```
ConfigIntxxxx ( INT ON/OFF | INT_Priority | INT_Sub_Priority );
```

參數一:開啟 / 關閉該周邊的中斷

參數二:設定主要中斷優先權等級

參數三:設定次要中斷優先權等級

```
INTSetVectorPriority( INT_UART_3A_VECTOR ,  
    INT_PRIORITY_LEVEL_5 );
```

// 設定主要中斷優先權等級。

```
INTSetVectorSubPriority( INT_UART_3A_VECTOR ,  
    INT_SUB_PRIORITY_LEVEL_0 );
```

// 設定次要中斷優先權等級。

```
INTEnable( INT_U3ARX , INT_ENABLED );
```

// 開啟周邊中斷。

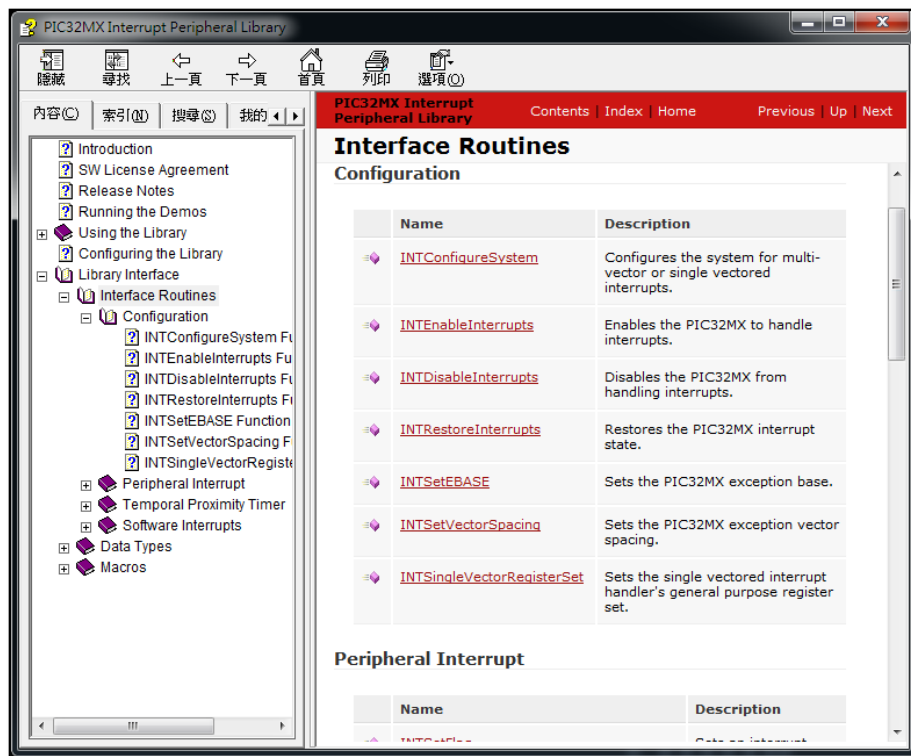
XC32 Interrupt Function & Macro

- MPLAB XC32提供許多Interrupt Function可供使用。

- `INTEnableSystemMultiVectoredInt();`
`INTEnableSystemSingleVectoredInt();`
`INTDisableInterrupts();`
`INTEnableInterrupts();`
`INTClearFlag();`
`INTSetFlag();`
`INTGetFlag();`
`INTSetPriority();`
`INTSetSubPriority();`
 ...

- 詳細說明請參閱

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



XC32 G.P. Timers Code Example

//建立Timer1的中斷服務函式, 並指定主要中斷優先權為(優先權必須相同)。

```
void __ISR ( _TIMER_1_VECTOR, ipl4 ) ISR_Timer1( void )
```

```
{
```

```
    // 清除Timer中斷旗標。
```

```
    INTClearFlag( INT_T1 );
```

```
}
```

```
int main ( void )
```

```
{
```

```
    // 使用XC32 Interrupt Function啟用多重中斷向量模式。
```

```
    INTEnableSystemMultiVectoredInt( );
```

```
    // 使用XC32 ConfigIntTimer1 Function啟用Timer1中斷, 並指定其主要中斷優先。
```

```
    ConfigIntTimer1( T1_INT_ON | T1_INT_PRIOR_4 | T1_INT_SUB_PRIOR_0 );
```

```
    // 使用XC32 OpenTimer1 Function 設定Timer1。
```

```
    OpenTimer1( T1_ON | T1_GATE_OFF | T1_PS_1_256 | T1_SOURCE_INT , 1000 );
```

```
    while( 1 );
```

```
}
```

ISR中變數的使用

volatile

- 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 Timer Interrupt

- 以Lab4的程式基礎, 將原本Polling Timer1中斷旗標的架構, 改成中斷架構, 由中斷服務函式來負責LED的Toggle。
- 先閱讀說明文件, 了解Timer Function中, 有關中斷的啟用與優先權設定方式。了解INTEnableSystemMultiVectoredInt();與ConfigIntTimer1()的用法。以及中斷服務函式的宣告與撰寫方式。
- 使用Bootloader將程式燒錄進APP028-1。觀察LED的動作狀態, 驗證是否每500mS亮/滅一次(1 Hz)?

Lab5 Timer Interrupt *Step1*

• 程式要如何改才能用改中斷架構 ?

先設定PIC32為多重中斷架構(`INTEnableSystemMultiVectoredInt();`)。

加入Timer1的中斷服務常式。將LED Toggle程式移入ISR中,
記住!, 進入中斷後必須清除T1IF(`INTClearFlag(INT_T1);`), 不然中斷狀態無法結束。

利用`ConfigIntTimer1()`啟用中斷, 設定優先權。中斷優先權設定為"4"。

將主程式的片段刪除, 只留下`while(1)`, 主程式完全不處理事情, LED的Toggle的功能也不在主程式內執行。主程式啥時都不做, 只等待Timer1中斷發生。