IMP Series

驅動函式庫

使用手冊

版本 ：V.1.01

日期 ：2013.01

http://www.epcio.com.tw

目 錄

I. 驅動程式函式庫簡介

IMP Series 驅動程式函式庫可用來驅動利用 IMC ASIC 所設計開發的 IMP Series 運動控制平台 IMP-2。

依功能的不同，後面章節將劃分為下面幾個部分說明 IMP Series 驅動程式函式庫的使用方式。

▲Global Control Interface	中斷及重置功能設定
▲Pulse Generator Control Interface	設定運動脈波輸出控制
▲Encoder Counter Interface	設定編碼器輸入與計數器之控制
▲Local I/O Control Interface	近端輸出入接點控制
▲Remote I/O Interface	遠端輸出入接點控制
▲ADC Control Interface	類比轉數位輸入控制
▲PCL Control Interface	硬體位置閉迴路設定
▲DAC Control Interface	數位轉類比輸出控制

相關參考手冊：

硬體相關資訊

IMP – 2	硬體使用手冊
IMP–WB–2	硬體使用手冊

驅動程式使用導引

IMP Series 驅動程式函式庫參考手冊

IMP Series 驅動程式函式庫範例手冊

IMP Series 驅動程式函式庫測試軟體使用手冊

II. 運動控制卡基址，中斷及重致功能設定

使用 IMP Series 驅動程式函式庫的第一步需先初始化 IMP 運動控制平台，可以使用下列函式初始化運動控制卡：

```
IMC_OpenDevice();
```

IMC_OpenDevice()的函式傳回值若為 TRUE(1)，表示初始化運動控制平台成功，此時方可繼續使用其他函式。

而要關閉 IMP Series 運動控制平台，需使用 IMC_CloseIfOpen()。本函式會關閉 IMP 內所有的功能模組，若使用運動控制平台時有啟動中斷功能，此時亦會還原中斷向量。

下面程式碼說明如何使用 IMP Series 驅動程式函式庫，其中利用 IMC_GLB_ResetModule()重置所指定的 IMC 模組，此函式通常會與初始化函式搭配使用。

```
if (IMC_OpenDevice())
{
    //   將 IMP Series 運動控制平台重置
    IMC_GLB_ResetModule(RESET_ALL);
    /*
    使用者欲執行的程式碼
    */

    IMC_CloseIfOpen();//   關閉 IMP Series 運動控制平台
}
```

III. 脈波輸出控制

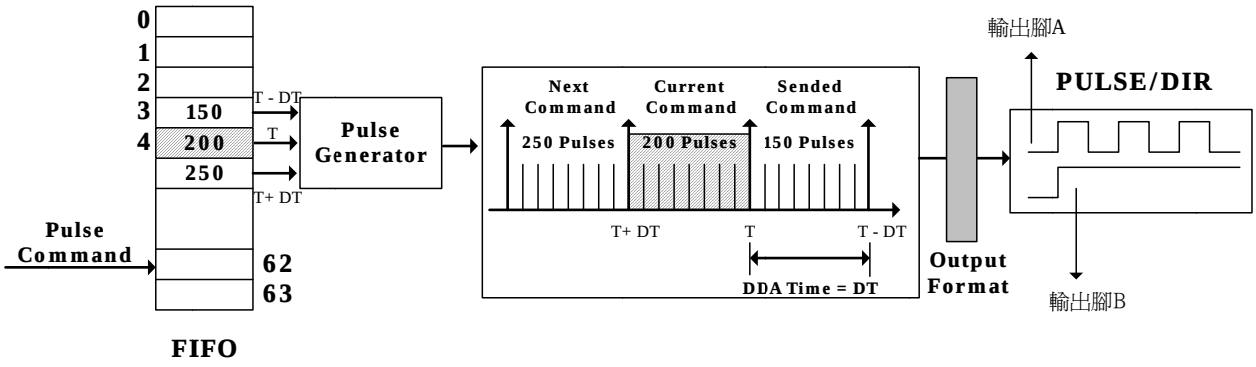
III.1 基本的脈波輸出控制

1 套 IMP Series 運動控制平台最多可擁有 8 個輸出 Channel(channel 0 ~ channel 7)，並使用 PGE 模組控制脈波輸出，此模組控制機制主要分為下列兩個步驟：

1. 將脈波命令(pulse command)送至所指定 Channel 的**脈波命令暫存器(FIFO)**，脈波命令暫存器共可儲存 64 筆脈波命令。

2. 脈波產生器使用 IPO Time 為時間間隔(IPO Time 可彈性設定)，每次由脈波命令暫存器中自動讀取 1 筆脈波命令，並在 IPO Time 時間內，依照所設定的輸出格式，由指定的 channel 均勻送出這些脈波。

下圖顯示這兩個步驟。需特別注意，每個輸出 Channel 擁有各自的 FIFO，以 IMP Series 運動控制平台為例，共有 8 個輸出 Channel，因此擁有 8 個 FIFO。此圖也表示每一個 IPO Time 均消耗一筆脈波命令。



由上面的說明可知，要送出脈波命令至少需完成下面的步驟，包括：

1. 使用 IMC_PGE_SetIPOTime()設定 IPO Time

2. 設定所指定 Channel 的脈波輸出格式，包括設定：

   甲、 設定脈波輸出格式。

   ➔ IMC_PGE_SetOutputFormat()

3. 使用 IMC_PGE_Start()啟動 PGE 機制

   ➔ *See Also* IMC_PGE_Start()

4. 使用 IMC_PGE_SendPulse()將脈波命令送到指定 Channel 的 FIFO。

下面程式碼說明如何在初始化運動控制卡成功後，送出一筆脈波命令。在 IMP Series 運動控制平台中，當有位置(脈波)或速度(電壓)命令輸出，均需使用 IMC_LIO_SetMotionEnable()開啟輸出功能；某些伺服系統可能需要呼叫 IMC_LIO_SetServoOn()，以開啟伺服馬達驅動器的 servo on 接點，系統才能正常運作。完整的呼叫程序如下：

```
//  開啟脈波及速度運動命令輸出功能
IMC_LIO_SetMotionEnable(1);

//  開啟 Channel 0 之 Servo On 接點
IMC_LIO_SetServoOn(0);

//  設定 IPO Time = 10 ms
IMC_PGE_SetIPOTime(10);

//  設定 Channel 0 之脈波輸出格式為 Pulse / Dircetion 格式
```

```
IMC_PGE_SetOutputFormat(0, PGE_FMT_PD);

//  啟動 PGE 機制
IMC_PGE_Start(1);

//  發出脈波命令，要求 Channel 0 在 1 個 IPO Time 內送出 200
//  個脈波
IMC_PGE_SendPulse(0, 200);
```

此外，使用 IMC_PGE_SetClockDivider()與 IMC_PGE_SetClock Number()

能詳細設定每個 IPO Time 可送出的脈波總數之上限，使用 IMC_PGE_SetIPOTime()將自動以所設定之 IPO Time 完成脈波總數上限之計算與設定，IMC_PGE_SendPulse()所送出的每筆脈波命令所包含之脈波總數需滿足此項限制。

III.2 控制脈波命令暫存器(FIFO)

IMP Series 驅動程式函式庫提供下列功能，用來控制與讀取 FIFO 的狀態。

1. 可使用 IMC_PGE_CheckFifoEmpty()檢查所指定 Channel 的 FIFO 中，目前是否已無儲存任何脈波命令。

2. 可使用 IMC_PGE_CheckFifoFull()檢查所指定 Channel 的 FIFO 中，目前是否已無多餘位置儲存脈波命令，每個 FIFO 共有 64 個儲存空間。

3. 可使用 IMC_PGE_GetStockCount()讀取所指定 Channel 的 FIFO 中，目前所儲存但尚未被執行之脈波命令筆數。

利用上面所提及的函式充分利用 FIFO 的空間，可預先將脈波命令置入 FIFO 中，避免因脈波命令不足造成運動不連續的現象出現，將可提昇系統運作的穩定性。

III.3 控制送出中的脈波命令

IMP Series 驅動程式函式庫提供下列功能，用來控制與讀取正在送出、已不在 FIFO 中的脈波命令。

1. 可使用 IMC_PGE_GetCurrentCommand()讀取所指定 Channel 目前正在送出的脈波命令，包括正負符號。利用此命令的正負符號，可判斷目前的運動方向。

2. 也可以使用 IMC_PGE_SetOutputFormat(指定的 Channel, PGE_FMT_NO)，使所指定 Channel 的輸出無效， FIFO 中的命令與正在執行中的命令皆會停止輸出。

III.4 緊急停止脈波輸出

某些情況下需緊急停止輸出脈波， IMP Series 驅動程式函式庫所提供的下列功能，能滿足此種需求。

1. 可使用 IMC_PGE_Start(0)關閉 PGE 機制，此函式將停止所有 Channel 的輸出功能。

2. 需緊急停止輸出 FIFO 中的命令與正在輸出的命令，可以使用 IMC_PGE_SetOutputFormat(指定的 Channel, PGE_FMT_NO)，使所指定 Channel 的輸出無效。

III.5 已輸出的脈波總數計數

IMP Series 驅動程式函式庫提供脈波計數功能，可獲得實際輸出的脈波總數。這些功能包括：

1. 使用 IMC_PGE_EnablePulseCounter()開啟所指定 Channel 的脈波計數功能。

2. 使用 IMC_PGE_ClearPulseCounter()使所指定 Channel 的脈波計數器之計數值歸零。

3. 使用 IMC_PGE_GetPulseCount()讀取所指定 Channel 的脈波計數器之計數內容

在使用 IMC_PGE_GetPulseCount()前需先開啟計數功能，也就是需先呼叫 IMC_PGE_EnablePulseCounter()。下面程式碼說明如何讀取 Channel 0 實際送出的脈波總數。

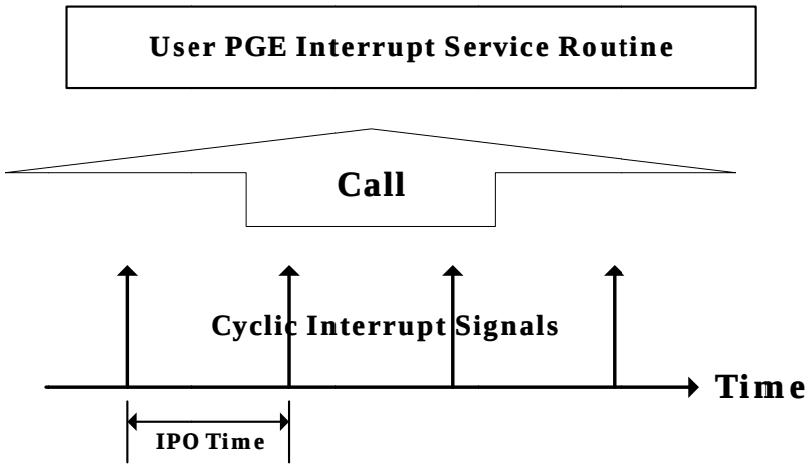
```
//  清除 Channel 0 之脈波計數器的計數值
IMC_PGE_ClearPulseCounter(0, 1);

//  開啟 Channel 0 之脈波計數功能
IMC_PGE_EnablePulseCounter(0, 1);
long lPulseCount;
//  讀取的 Channel 0 實際送出之脈波總數
lPulseCount = IMC_PGE_GetPulseCount(0, &lPulseCount);
```

通常會使用脈波計數器的計數內容與實際使用 IMC_PGE_SendPulse()所送出的脈波數相比較，以便驗證是否正確使用脈波輸出功能。

III.6 循環中斷功能

IMP Series 驅動程式函式庫提供循環中斷功能，當開啟循環中斷功能後，驅動程式函式庫將以 IPO Time 為發生週期，每隔 IPO Time 便自動觸發並執行使用者自訂的 PGE 中斷服務函式，如下圖。



要使用循環中斷功能需完成下列幾個步驟：

1. 宣告與定義使用者自訂的 PGE 中斷服務函式，此函式必須遵循下面的宣告：

```
typedef void(IMC_LIB_CALL *PGEISR)(PGEINT *p);
```

因此使用者自訂的 DDA 中斷服務函式可定義如下：

```
void _stdcall PGE_ISR_Function(PGEINT *pstINTSource)
{
    if (pstINTSource->CYCLE)//   判斷循環中斷是否發生
    {
      /*
      循環中斷發生後欲執行的程式碼
```

```
            */
        }
    }
```

在 PGE 中斷服務函式中需判斷此函式是否由循環中斷所觸發。此外 PGE 中斷服務函式的宣告需加上關鍵字_stdcall，但如果使用的是 Standalone 模式則需要加上 IMC_LIB_CALL 此關鍵字。

2. 串接使用者自訂的 PGE 中斷服務函式

在初始化運動控制卡後需串接 PGE 中斷服務函式，將中斷服務函式的函式指標傳入 IMC_PGE_SetISRFunction()，如下：

```
IMC_PGE_SetISRFunction(PGE_ISR_Function);
```

3. 使用 IMC_PGE_EnableCycleInterrupt()開啟循環中斷功能

➔ *See Also* IMC_PGE_EnableCycleInterrupt()

下面程式碼將說明如何使用循環中斷功能

```
void _stdcall PGE_ISR_Function(PGEINT *pstINTSource)
{
        if (pstINTSource->CYCLE)//    判斷循環中斷是否發生
        {
            /*
            循環中斷發生後欲執行的程式碼
            */
        }
       }
    •
    •
```

```
    •

if (IMC_OpenDevice())
{
    •
    IMC_PGE_SetISRFunction(PGE_ISR_Function);
    IMC_PGE_EnableCycleInterrupt(1);
    •
}
```

循環中斷為硬體中斷，擁有較精確的觸發週期，通常使用在具週期特性且要求準時執行的工作。利用循環中斷功能，搭配檢視 FIFO 狀態相關的函式，可保證 FIFO 中的命令庫存量可滿足連續運動的即時性需求，避免因 FIFO 中已無命令而造成運動中斷的現象。

假設總共需送出 200 筆脈波命令，但因 FIFO 最多能儲存 64 筆命令，因此使用循環中斷觸發中斷服務函式，並在此函式內讀取 FIFO 內目前儲存的命令筆數並計算剩餘的儲存空間，並藉以判斷與送出可送入 FIFO 的命令。在中斷服務程式中將重複這些動作，直到送完 200 筆命令為止。下面的程式碼說明此過程。

```
int nCount = 200;          //    共需送出 200 筆脈波命令
int nPulse[200] = 150;     //     200 筆命令，命令可預先規劃

void _stdcall PGE_ISR_Function(PGEINT *pstINTSource)
{
        WORD wStockNo;

        if (pstINTSource->CYCLE)//    判斷循環中斷是否發生
        {
           if (nCount)//    送完 200 筆命令時 nCount 等於 0
```

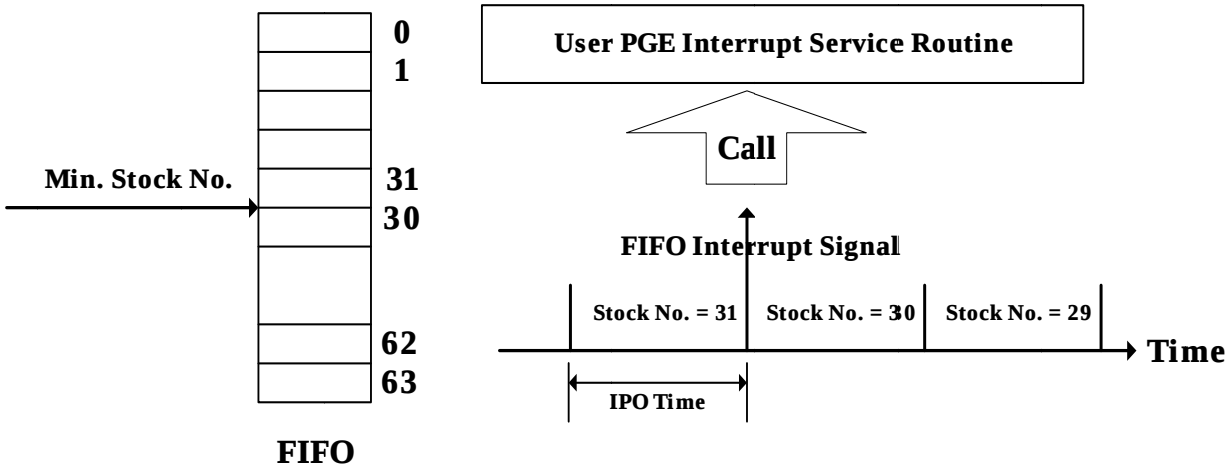
```
            {
                //  讀取 Channel 0 之 FIFO 中目前儲存的命令筆數
                IMC_PGE_GetStockCount(0, &wStockNo);

                //  “i < 64 – wStockNo”是因為 FIFO 只具 64 筆儲存
                //  空間
                for (int i = 0;i < 64 - wStockNo && nCount;i++)
                {
                    IMC_PGE_SendPulse(0, nPulse[200 - nCount])
                    nCount--;
                }
            }
        }
    }
```

循環中斷因具備週期發生的特性，因此也常用來檢查系統的各種狀態，例如 I/O 接點的輸出入狀態。

III.7 FIFO 最低庫存數目中斷

IMP Series 驅動程式函式庫提供 FIFO 最低庫存數目中斷(簡稱 FIFO 中斷)功能。當設定 FIFO 最低庫存數目並開啟所指定 Channel 的 FIFO 中斷功能後，則所指定 Channel 的 FIFO 庫存命令消耗到只剩最低庫存數目時，將觸發並執行使用者自訂的 PGE 中斷服務函式，如下圖。



要使用 FIFO 中斷功能需完成下列幾個步驟：

1. 宣告與定義使用者自訂的 PGE 中斷服務函式，此函式必須遵循下面的宣告：

```
typedef void(IMC_LIB_CALL *PGEISR)(PGEINT *p)
```

因此使用者自訂的 PGE 中斷服務函式可定義如下：

```
void _stdcall PGE_ISR_Function(PGEINT *pstINTSource)
{
    //  判斷是否發生 Channel 0 的 FIFO 中斷
    if (pstINTSource->FIFO0)
    {
       /*
       Channel 0  的 FIFO 中斷發生後欲執行的程式碼
       */
    }
    //  判斷是否發生 Channel 1 的 FIFO 中斷
    if (pstINTSource->FIFO1)
```

```
        {
            /*
            Channel 1 的 FIFO 中斷發生後欲執行的程式碼
            */
        }
    .
    .
}
```

在 PGE 中斷服務函式中需判斷此函式是否由 FIFO 中斷所觸發。pstINTSource->FIFO0 ~ pstINTSource->FIFO7 分別用來判斷是否發生 Channel 0 ~ Channel 7 的 FIFO 中斷。此外 PGE 中斷服務函式的宣告需加上關鍵字_stdcall，，但如果使用的是 Standalone 模式則需要加上 IMC_LIB_CALL 此關鍵字。

2. 串接使用者自訂的 PGE 中斷服務函式。
   在初始化運動控制卡後需串接 PGE 中斷服務函式，將中斷服務函式的函式指標傳入 IMC_PGE_SetISRFunction()，如下：

   IMC_PGE_SetISRFunction(PGE_ISR_Function);

3. 使用 IMC_PGE_EnableStockInterrupt()開啟所指定 Channel 的 FIFO 中斷功能。
   ➔ *See Also* IMC_PGE_EnableStockInterrupt()

下面程式碼說明如何使用 FIFO 中斷功能。

```
void _stdcall PGE_ISR_Function(PGEINT *pstINTSource)
{
        //   判斷是否發生 channel 0 的 FIFO 中斷
```

```
            if (pstINTSource->FIFO0)
            {
                /*
                FIFO 中斷發生後欲執行的程式碼
                */
            }
        }
    •
    •
    •

if (IMC_OpenDevice())
{
    •
    IMC_PGE_SetISRFunction(PGE_ISR_Function);
    IMC_PGE_EnableStockInterrupt(0, 1);
    •
}
```

FIFO 中斷也為硬體中斷，且只在滿足中斷條件時中斷才會發生，與周期性發生的循環中斷相比，FIFO 中斷較不會增加系統執行時的負荷。通常利用 FIFO 中斷功能，搭配檢視 FIFO 狀態相關的函式，可保證 FIFO 中的命令不會低於某一設定值(即 FIFO 最低庫存數目)，如此可避免因 FIFO 中已無命令而造成運動中斷的現象。

假設總共需送出 200 筆脈波命令，但因 FIFO 最多能儲存 64 筆命令，因此使用 FIFO 中斷觸發中斷服務函式，並在此函式內讀取 FIFO 內目前儲存的命令筆數並計算剩餘的儲存空間，並藉以判斷與送出可送入 FIFO 的命令。在中斷服務程式中將重複這些動作，直到送完 200 筆命令為止。

但為觸發 FIFO 中斷，因此需先送出 64 筆命令至 FIFO，如此才可能出現滿足 FIFO 中斷觸發條件的情況(也就是 FIFO 所儲存的命令筆數由 31 筆消耗至 30 筆時)。下面的程式碼說明此過程。

```
int nCount = 200;          //   共需送出 200 筆脈波命令
int nPulse[200] = 150;     //    200 筆命令，命令可預先規劃
.
.
for (int i = 0;i < 64;i++) //   先送出 64 筆命令至 Channel 0 之
                           //    FIFO
{
      IMC_PGE_SendPulse(0, nPulse[200 - nCount])
      nCount--;
}

.
.
void _stdcall PGE_ISR_Function(PGEINT *pstINTSource)
{
        WORD wStockNo;

        if (pstINTSource->FIFO0)
        {
          if (nCount)//   送完 200 筆命令時 nCount 等於 0
          {
              //   讀取 Channel 0 之 FIFO 中目前儲存的命令筆數
              IMC_PGE_GetStockCount(0, &nStockNo);
```

```
                //   FIFO 具 64 筆命令儲存空間
                for (int i = 0;i < 64 - nStockNo && nCount;i++)
                {
                    IMC_PGE_SendPulse(0, nPulse[200 - nCount])
                    nCount--;
                }
            }
        }
    }
```

IV. 編碼器控制

IV.1 基本設定與功能

IMP Series 運動控制平台最多擁有 8 個 channel 可輸入編碼器訊號，編號分別為 channel 0 ~ channel 7，要使用與編碼器控制有關的函式，首先需完成下列各項步驟：

1. 使用 IMC_ENC_SetInputFormat()設定所指定 Channel 的輸入訊號型態，輸入訊號型態必須搭配硬體設定。當輸入訊號為馬達編碼器迴授訊號時，請參考馬達或驅動器設定；當接一般手輪時請設定為 A/B Phase 輸入(內定為 A/B Phase 輸入)。

2. 使用 IMC_ENC_SetInputRate()設定所指定 Channel 的計數器訊號解碼倍率。解碼倍率必須在所輸入的編碼器格式為 A/B Phase 時方為有效。本函式必須搭配 IMC_ENC_SetInputFormat()設定為 A/B Phase 輸入。

3. 使用 IMC_ENC_StartCounter()開啟計數器的計數功能，在使用此函式前通常會先呼叫 IMC_ENC_ClearCounter()使計數器的計數值歸零。

完成上面各項設定後即可利用 IMC_ENC_ReadCounter()讀取所指定 Channel 的計數值。

下面程式碼說明如何讀取 channel 0 的計數值。

```
long lCounter;
//  設定 Channel 0 之輸入格式為 A/B Phase
IMC_ENC_SetInputFormat(0, ENC_TYPE_AB);
```

```
//  設定 Channel 0 之訊號解碼倍率為 x4
IMC_ENC_SetInputRate(0, ENC_RATE_X4);

//  使 Channel 0 之計數器之計數值歸零
IMC_ENC_ClearCounter(0, 1);

//  開啟 Channel 0 計數功能
IMC_ENC_StartCounter(0, 1);

//  讀取 Channel 0 之計數器計數值
IMC_ENC_ReadCounter(0, &Counter);
```

為了配線的實際需要，IMP Series 驅動程式函式庫提供下列函式，可將送到計數器輸入腳位的訊號反向：

1. 使用 IMC_ENC_EnableInAInverse()可使所指定 Channel 的計數器輸入訊號中的 inA 腳位反相。預設狀態為不反相。
   ➔ *See Also* IMC_ENC_EnableInAInverse()

2. 使用 IMC_ENC_EnableInBInverse ()可使所指定 Channel 的計數器輸入訊號中的 inB 腳位反相。預設狀態為不反相。
   ➔ *See Also* IMC_ENC_EnableInBInverse()

3. 使用 IMC_ENC_EnableInCInverse()可使所指定 Channel 的計數器輸入訊號中的 inC 腳位反相。預設狀態為不反相。
   ➔ *See Also* IMC_ENC_EnableInCInverse()

4. 使用 IMC_ENC_EnableInABSwap()可使所指定 Channel 的計數器輸入訊號中的 inA 及 inB 腳位，在訊號進入計數器前先經過訊號交

換處理。預設狀態為不需經過訊號交換處理。

➔ *See Also* IMC_ENC_EnableInABSwap()

IV.2 編碼器計數值觸發中斷服務函式功能

IMP Series 驅動程式函式庫所提供的編碼器計數值觸發中斷服務函式功能(簡稱計數值觸發中斷功能)，可讓使用者針對所指定 Channel 設定比較值，當啟動所指定 Channel 此項功能，並在該 Channel 的計數值等於比較值時，比較器將自動觸發並執行使用者自訂的中斷服務函式。要使用計數值觸發中斷功能必須完成下列幾個步驟：

1. 定義與宣告使用者自訂的 ENC 中斷服務函式，ENC 中斷服務函式的函式宣告如下：

```
typedef void(IMC_LIB_CALL *ENCISR)(ENC INT *p);
```

因此使用者自訂的 ENC 中斷服務函式可定義如下：

```
void _stdcall ENC_ISR_Function (ENCINT *pstINTSource)
{
    //   判斷是否由 Channel 0 的計數值觸發中斷服務函式
    if (pstINTSource->COMP0)
    {
       /*
       計數值觸發中斷後欲執行的程式碼
       */
    }
 }
```

在 ENC 中斷服務函式中需判斷此函式是否由計數值所觸發。

在與編碼器相關的中斷功能中，Channel 0 ~ Channel 7 在計數值觸發 ENC 中斷服務函式後，使用 COMP0 ~ COMP7 判斷是由哪一個 Channel 的編碼器之計數值觸發此。各個 Channel 與 COMP0 ~ COMP7 的關係如下面所示。

Channel 0：pstINTSource ->COMP0 ----
Channel 1：pstINTSource ->COMP1 ----
Channel 2：pstINTSource ->COMP2 ----
Channel 3：pstINTSource ->COMP3 ----
Channel 4：pstINTSource ->COMP4 ----
Channel 5：pstINTSource ->COMP5 ----
Channel 6：pstINTSource ->COMP6 ----
Channel 7：pstINTSource ->COMP7 ----

因此 pstINTSource->COMP0 ~ pstINTSource->COMP7 分別用來判斷中斷服務函式是否由第 0 個 Channel ~ 第 7 個 Channel 的編碼器計數值所觸發。

此外 ENC 中斷服務函式的宣告需加上關鍵字_stdcall，但如果使用的是 Standalone 模式則需要加上 IMC_LIB_CALL 此關鍵字。

2. 串接使用者自訂的 ENC 中斷服務函式。

在初始化運動控制卡後需串接 ENC 中斷服務函式，將中斷服務函式的函式指標傳入 IMC_ENC_SetISRFunction()，如下：

```
IMC_ENC_SetISRFunction(ENC_ISR_Function);
```

3. 使用 IMC_ENC_SetComparator()設定所指定 Channel 的計數器比較值。

4. 使用 IMC_ENC_EnableComparatorInterrupt()開啟所指定 Channel 的編碼器計數值觸發中斷服務函式功能。

下面的程式碼只開啟 Channel 4 的編碼器計數值觸發中斷服務函式功能。

```
void _stdcall ENC_ISR_Function(ENCINT *pstINTSource)
    {
        //   判斷是否由第 4 個 channel 的編碼器計數值所觸發
        if (pstINTSource->COMP4)
        {
            /*
            編碼器計數值觸發中斷服務函式後欲執行的程式碼
            */
        }
    }
.
.
if (IMC_OpenDevice())
{
    .
    .
    IMC_ENC_SetISRFunction(ENC_ISR_Function);
    //   設定 Channel 4 之計數器比較值為 10000
    IMC_ENC_SetComparator(4, 10000);

    //   開啟 Channel 4 之編碼器計數值觸發中斷服務函式功能
    IMC_ENC_EnableComparatorInterrupt(4, 1);
    .
    .
}
```

IV.3 Index 中斷

IMP Series 驅動程式函式庫提供編碼器 Index 中斷功能，當編碼器之 Index(Z Phase)訊號輸入時，可觸發使用者自訂的中斷服務函式。要使用 Index 中斷功能必須完成下列幾項設定：

1. 定義與宣告使用者自訂的 ENC 中斷服務函式，ENC 中斷服務函式的函式宣告如下：

```
typedef void(IMC_LIB_CALL *ENCISR)(ENC INT *p);
```

因此使用者自訂的 ENC 中斷服務函式可定義如下：

```
void _stdcall ENC_ISR_Function (ENCINT *pstINTSource)
{
    //   判斷是否由第 0 個 Channel 的 Index 訊號所觸發
    if (pstINTSource->INDEX0)
    {
       /*
       Index 中斷發生後欲執行的程式碼
       */
    }
}
```

在 ENC 中斷服務函式中需判斷此函式是否由 Index 中斷所觸發。在與編碼器相關的中斷功能中，Channel 0 ~ Channel 7 在 Index 中斷發生後，使用 INDEX0 ~ INDEX7 判斷是由哪一個 Channel 的 Index 訊號所觸發。各個 Channel 與 INDEX0 ~ INDEX7 的關係如下面所示。

Channel 0：pstINTSource ->INDEX0 ----

Channel 1：pstINTSource ->INDEX1 ----

Channel 2：pstINTSource ->INDEX2 ----

Channel 3：pstINTSource ->INDEX3 ----

Channel 4：pstINTSource ->INDEX4 ----

Channel 5：pstINTSource ->INDEX5 ----

Channel 6：pstINTSource ->INDEX6 ----

Channel 7：pstINTSource ->INDEX7 ----

因此 pstINTSource->INDEX0 ~ pstINTSource->INDEX7 分別用來判斷是否發生第 0 個 channel ~ 第 7 個 channel Index 中斷。

此外 ENC 中斷服務函式的宣告需加上關鍵字_stdcall，但如果使用的是 Standalone 模式則需要加上 IMC_LIB_CALL 此關鍵字。

2. 串接使用者自訂的 ENC 中斷服務函式。

在初始化運動控制卡後需串接 ENC 中斷服務函式，將中斷服務函式的函式指標傳入 IMC_ENC_SetISRFunction()，如下：

IMC_ENC_SetISRFunction(ENC_ISR_Function);

3. 使用 IMC_ENC_EnableIndexInterrupt()開啟所指定 Channel 的 Index 中斷觸發功能。

➔ *See Also* IMC_ENC_GetIndexStatus()

下面的程式碼只開啟 Channel 5 的 Index 中斷功能。

```
void _stdcall ENC_ISR_Function(ENCINT *pstINTSource)
{
   //   判斷是否由第 5 個 Channel 之 Index 中斷所觸發
```

```
    if (pstINTSource->INDEX5)
    {
        /*
        Index 中斷發生後欲執行的程式碼
        */
    }
}
.
.
if (IMC_OpenDevice())
{
    .
    .
    IMC_ENC_SetISRFunction(ENC_ISR_Function);
    //   開啟 Channel 5 之 Index 中斷觸發功能
    IMC_ENC_EnableIndexInterrupt(5, 1);
    .
    .
}
```

IV.4 計數器計數值 Latch 功能

IMP Series 驅動程式函式庫提供計數器計數值 Latch 功能，使用者可設定觸發訊號源，這些觸發訊號源被用來觸發將計數器的計數值紀錄在閂鎖暫存器內的動作，使用者並可使用驅動程式函式庫提供的函式，讀取閂鎖暫存器內的紀錄值。

計數器計數 Latch 功能之觸發訊號源分為 Index 訊號觸發或外部條件觸發兩類，使用計數器計數 Latch 功能首先需使用 IMC_ENC_SetIndexLatchSource()或 IMC_ENC_SetExternalLatchSource()設定觸發訊號源是來自 Index 訊號或外部條件觸發訊號，函式的函式宣告如下：

```
void IMC_ENC_SetIndexLatchSource(    WORD Channel,
                                     WORD Source    );
void IMC_ENC_SetExternalLatchSource ( WORD Channel,
                                      WORD Source );
```

Channel 表示 Channel 的編號，範圍為 0～7。Source 則為觸發訊號源，Index 訊號觸發或外部條件觸發各有 8 種觸發源可觸發閂鎖(Latch)計數器計數值的動作，設定時可同時取多個條件的聯集。這些觸發訊號源包括：

NO_TRIG_ENC	沒有選擇任何觸發訊號源
INDEX0_TRIG_ENC	Channel 0 編碼器的 Index 訊號
INDEX1_TRIG_ENC	Channel 1 編碼器的 Index 訊號
INDEX2_TRIG_ENC	Channel 2 編碼器的 Index 訊號
INDEX3_TRIG_ENC	Channel 3 編碼器的 Index 訊號
INDEX4_TRIG_ENC	Channel 4 編碼器的 Index 訊號
INDEX5_TRIG_ENC	Channel 5 編碼器的 Index 訊號
INDEX6_TRIG_ENC	Channel 6 編碼器的 Index 訊號
INDEX7_TRIG_ENC	Channel 7 編碼器的 Index 訊號
OTP0_TRIG_ENC	Channel 0 正極限輸入點發生訊號
OTP1_TRIG_ENC	Channel 1 正極限輸入點發生訊號
OTP2_TRIG_ENC	Channel 2 正極限輸入點發生訊號
OTP3_TRIG_ENC	Channel 3 正極限輸入點發生訊號
OTP4_TRIG_ENC	Channel 4 正極限輸入點發生訊號
OTP5_TRIG_ENC	Channel 5 正極限輸入點發生訊號
OTP6_TRIG_ENC	Channel 6 正極限輸入點發生訊號
OTP7_TRIG_ENC	Channel 7 正極限輸入點發生訊號
OTN0_TRIG_ENC	Channel 0 負極限輸入點發生訊號
OTN1_TRIG_ENC	Channel 1 負極限輸入點發生訊號

OTN2_TRIG_ENC	Channel 2 負極限輸入點發生訊號
OTN3_TRIG_ENC	Channel 3 負極限輸入點發生訊號
OTN4_TRIG_ENC	Channel 4 負極限輸入點發生訊號
OTN5_TRIG_ENC	Channel 5 負極限輸入點發生訊號
OTN6_TRIG_ENC	Channel 6 負極限輸入點發生訊號
OTN7_TRIG_ENC	Channel 7 負極限輸入點發生訊號

當觸發訊號源設定完成後，在這些訊號發生時會將計數器的計數值紀錄在閂鎖暫存器內。不過在使用 IMC_ENC_StartCounter()開始計數器 Latch 功能前，需先呼叫 IMC_ENC_SetCounterLatchMode()設定 Latch 模式。此函式的函式宣告如下：

```
IMC_ENC_SetCounterLatchMode(    WORD Channel,
                                WORD Mode)
```

Channel 表示 Channel 的編號，範圍為 0～7。Mode 為 Latch 觸發模式，可為：

ENC_TRIG_FIRST	第一次滿足觸發條件後即閂鎖住計數器的計數值並不再變動。
ENC_TRIG_LAST	觸發條件滿足時即閂鎖住計數器的計數值，但只要再次滿足條件即閂鎖住新的計數值。

使用者可使用 IMC_ENC_ReadLatchCounter()讀取指定 Channel 儲存在閂鎖暫存器中的紀錄值。

下面程式碼說明如何將觸發源設定為串接至 Channel 0 的編碼器之 Index 訊號，並在 Index 訊號發生，讀取閂鎖住的紀錄值，使用者可由此值獲得 Index 真正的位置。

```
void _stdcall ENC_ISR_Function(ENCINT *pstINTSource)
```

```
{
    //   判斷是否由第 0 個 Channel 的 Index 訊號所觸發
    if (pstINTSource->INDEX0)
    {
        //   Index 中斷發生後欲執行的程式碼
        long lLatchValue

        //   讀取閂鎖暫存器中的計數值
        IMC_ENC_ReadLatchCounter(0, &lLatchValue)
    }
}
 .
 .
if (IMC_OpenDevice())
{
      .
      .
     IMC_ENC_SetISRFunction(ENC_ISR_Function);
     //   設定 Channel 0 閂鎖計數器的觸發源為該 Channel 編碼
     //   器之 Index 訊號
     IMC_ENC_SetIndexLatchSource(0, INDEX0_TRIG_ENC);

     //   設定 Channel 0 閂鎖計數器之 Latch 觸發模式為連續觸發
     IMC_ENC_SetCounterLatchMode(0, ENC_TRIG_LAST);

      .
      .
}
```

V. 近端輸出入接點 (Local IO) 控制

近端輸出入接點共有 42 個可規劃為輸出或輸入的 IO 接點，不過在 IMP Series 運動控制平台上已規劃好這些 IO 的特定用途，包括：

輸入接點總共 25 個 Port，包括：

Home Sernsor	共 8 個輸入接點
Limit Switch Plus(+)	共 8 個輸入接點
Limit Switch Minus(-)	共 8 個輸入接點
Status for Emergency Stop	共 1 個輸入接點

輸出接點總共 17 個 Port 及 1 個 Motion Enable 專用輸出點，包括：

Servo On/Off	共 8 個輸出接點
LED Light	共 8 個輸出接點
Enabling Position Ready	共 1 個輸出接點
Motion Enable	共 1 個輸出接點

下面章節將說明如何使用這些近端輸、出入接點

V.1 基本設定與功能

近端數位輸出入以 8 點為一組，40 個 IO 接點依預設用途共分為 OT+、OT-、HOME、SERVO 以及 LED。其中 OT+ / OT- / HOME 預設規劃為輸入點，SERVO / LED 預設功能為輸出點，所有輸出入點的預設狀態皆設定為 Disable(0, 無輸出/入)狀態。

以 OT+為例，使用 IMC_LIO_GetPlusLimitLDIInput()可讀取 OT+ 0 ~ OT+ 7 的數位訊號輸入值，此函式的宣告如下：

```
IMC_LIO_GetPlusLimitLDIInput(DWORD *LDIState);
```

利用此函式所獲得*LDIState 的 Bit 0~Bit 7 即表示 OT+ 0 ~ OT+ 7 的輸入狀態，Bit 8 ~ Bit 31 則無任何意義。

因此若使用 IMC_LIO_GetPlusLimitLDIInput(&dwInput)所獲得的輸入值為 0x0002，表示 OT+ 1 輸入點目前的數位訊號輸入值為 1，因為 0x0002 換成 2 進位制等於 0b00000000**00000010**，OT+ 1 相關的 Bit 位置之值為 1。

➔ *See Also* IMC_LIO_GetMinusLimitLDIInput()
IMC_LIO_GetHomeSensorLDIInput()

需使用上述所提及函式之輸出入功能時，各點所代表的意義可參考下表：

IMP Series 運動控制平台近端輸出入接點 (Local I/O)

LIO	定義	對應之 SCSI II 位置	備註
0	Channel 0 OT+	10（100Pin 外接輸入點）	可觸發中斷
1	Channel 1 OT+	60（100Pin 外接輸入點）	可觸發中斷
2	Channel 2 OT+	14（100Pin 外接輸入點）	可觸發中斷
3	Channel 3 OT+	64（100Pin 外接輸入點）	可觸發中斷
4	Channel 4 OT+	18（100Pin 外接輸入點）	可觸發中斷
5	Channel 5 OT+	68（100Pin 外接輸入點）	可觸發中斷
6	Channel 6 OT+	7（40Pin 外接輸入點）	可觸發中斷
7	Channel 7 OT+	8（40Pin 外接輸入點）	可觸發中斷
8	Channel 0 OT-	11（100Pin 外接輸入點）	可觸發中斷
9	Channel 1 OT-	61（100Pin 外接輸入點）	可觸發中斷
10	Channel 2 OT-	15（100Pin 外接輸入點）	可觸發中斷
11	Channel 3 OT-	65（100Pin 外接輸入點）	可觸發中斷
12	Channel 4 OT-	19（100Pin 外接輸入點）	可觸發中斷
13	Channel 5 OT-	69（100Pin 外接輸入點）	可觸發中斷

14	Channel 6 OT-	9（40Pin 外接輸入點）	可觸發中斷
15	Channel 7 OT-	10（40Pin 外接輸入點）	可觸發中斷
16	Channel 0 HOME	9（100Pin 外接輸入點）	可觸發中斷
17	Channel 1 HOME	59（100Pin 外接輸入點）	可觸發中斷
18	Channel 2 HOME	13（100Pin 外接輸入點）	可觸發中斷
19	Channel 3 HOME	63（100Pin 外接輸入點）	可觸發中斷
20	Channel 4 HOME	17（100Pin 外接輸入點）	可觸發中斷
21	Channel 5 HOME	67（100Pin 外接輸入點）	可觸發中斷
22	Channel 6 HOME	5（40Pin 外接輸入點）	可觸發中斷
23	Channel 7 HOME	6（40Pin 外接輸入點）	可觸發中斷
24	Channel 0 SERVO	12（100Pin 外接輸出點）	
25	Channel 1 SERVO	62（100Pin 外接輸出點）	
26	Channel 2 SERVO	16（100Pin 外接輸出點）	
27	Channel 3 SERVO	66（100Pin 外接輸出點）	
28	Channel 4 SERVO	20（100Pin 外接輸出點）	
29	Channel 5 SERVO	70（100Pin 外接輸出點）	
30	Channel 6 SERVO	11（40Pin 外接輸出點）	
31	Channel 7 SERVO	12（40Pin 外接輸出點）	
32	Channel 0 LED	非外接訊號輸出點	可條件觸發
33	Channel 1 LED	非外接訊號輸出點	可條件觸發
34	Channel 2 LED	非外接訊號輸出點	可條件觸發
35	Channel 3 LED	非外接訊號輸出點	可條件觸發
36	Channel 4 LED	非外接訊號輸出點	可條件觸發
37	Channel 5 LED	非外接訊號輸出點	可條件觸發
38	Channel 6 LED	非外接訊號輸出點	可條件觸發
39	Channel 7 LED	非外接訊號輸出點	可條件觸發
40	ESTOP	57（100Pin 外接輸入點）	
41	P_RDY	58（100Pin 外接輸出點）	

42	MOTION ENABLE	非外接訊號輸出點	

IMP Series 運動控制平台因已規劃好這些 IO 接點的特定用途，所以如果搭配 IMP-WB-1 或 IMP-WB-2 轉接板使用，可以使用下面的函式，對轉接板上相對應的接點作讀取或輸出的動作。

IMP Series 驅動程式函式庫提供下列函式來讀取輸入接點的狀態。

1. 使用 IMC_LIO_GetHomeSensorStatus()讀取所指定 channel 的 HOME 點狀態。HOME 點狀態改變時並不會產生中斷訊號，只能使用此函式檢查所指定 channel 的 HOME 點狀態。

2. 使用 IMC_LIO_GetPlusLimitStatus()讀取所指定的 Channel 是否已碰觸正方向的硬體極限開關(limit switch plus)，若是則機臺有可能發生撞機的危險，使用者應立即作緊急處置。在碰觸正方向的硬體極限開關時可觸發使用者自訂的中斷服務函式。

3. 使用 IMC_LIO_GetMinusLimitStatus()讀取所指定的 Channel 是否已碰觸負方向的硬體極限開關(limit switch minus)，若是則機臺有可能發生撞機的危險，使用者應立即作緊急處置。在碰觸負方向的硬體極限開關時可觸發使用者自訂的中斷服務函式。

IMP Series 驅動程式函式庫提供下列函式來設定輸出接點的狀態。

1. 使用 IMC_LIO_SetServoOn()開啟所指定 channel 的伺服驅動接點功能。本接點可連接馬達驅動器的伺服驅動接點，當呼叫本函式設定後，所指定 Channel 將可接受來自 IMP Series 運動控制平台的位置或速度命令。

2. 使用 IMC_LIO_SetServoOff()關閉所指定 Channel 的伺服驅動接點

功能。本接點可連接馬達驅動器的伺服驅動接點，當呼叫本函式後，所指定的 Channel 將不再接受位置或速度命令。在呼叫初始化函式成功後，內定狀態為關閉伺服驅動功能。

3. 使用 IMC_LIO_SetLedLightOn()設定所指定 channel 的 LED 點輸出功能。LED 輸出接點連接 IMP Series 運動控制平台上的 LED 指示燈，當呼叫本函式設定後，所指定 Channel 將可接受來自 IMP Series 運動控制平台的 LED 輸出命令。在呼叫初始化函式成功後，LED 內定為關閉狀態。

4. 使用 IMC_LIO_SetMotionEnable()開啟 IMP Series 運動控制平台的位置(脈波)與速度(電壓)命令輸出功能。當呼叫本函式後，輸出功能將被開啟。在呼叫初始化函式成功後，內定狀態為關閉輸出功能。

IMP Series 運動控制平台的輸出接點都有特定用途，但這些輸出接點也可用來作為一般的輸出用途。例如某些 Channel 使用的是步進馬達，並不需要 Servo On/Off 訊號控制，則這些 Channel 的 Servo On/Off 輸出接點可用來作為一般的輸出用途。

V.2 硬體極限開關中斷

IMP Series 運動控制平台的 Limit Switch Plus 與 Limit Switch Minus 接點(或稱為過行程極限接點)以及 Home Sensor(或稱 Home 點或原點)，提供硬體極限開關中斷功能(簡稱為極限中斷)，當發生碰觸極限開關的狀況時，將觸發使用者自訂的中斷服務函式，使用者可利用此功能規劃緊急處理動作的內容。

要使用極限中斷功能需完成下列各項步驟的內容：

1. 定義與宣告使用者自訂的 LIO 中斷服務函式，LIO 中斷服務函式

的宣告必需遵循下面的定義：

```
typedef void(IMC_LIB_CALL *LIOISR)(LIOINT *p);
```

因此使用者自訂的 LIO 中斷服務函式可定義如下：

```
void _stdcall LIO_ISR_Function(LIOINT *pstINTSource)
{
    //   判斷是否發生極限中斷
    if (pstINTSource-> OTP0)
    {
       /*
       發生過行程極限時的緊急處理動作
       */
    }
}
```

在 LIO 中斷服務函式中需使用 OTP 0 ~ OTP 7 / OTN 0 ~ OTN 7 / HOME 0 ~ HOME 7，判斷此函式是否由極限中斷所觸發。OTP 0 ~ OTP 7 / OTN 0 ~ OTN 7 / HOME 0 ~ HOME 7 所表示的意義如下：

a. IMP Series 運動控制平台 Limit Switch Plus (OT+)

pstINTSource-> OTP0	Channel 0的OT+
pstINTSource-> OTP1	Channel 1的OT+
pstINTSource-> OTP2	Channel 2的OT+
pstINTSource-> OTP3	Channel 3的OT+
pstINTSource-> OTP4	Channel 4的OT+
pstINTSource-> OTP5	Channel 5的OT+

pstINTSource-> OTP6　Channel 6的OT+

pstINTSource-> OTP7　Channel 7的OT+

b. IMP Series 運動控制平台 Limit Switch Minus (OT-)

pstINTSource-> OTN0　Channel 0的OT-

pstINTSource-> OTN1　Channel 1的OT-

pstINTSource-> OTN2　Channel 2的OT-

pstINTSource-> OTN3　Channel 3的OT-

pstINTSource-> OTN4　Channel 4的OT-

pstINTSource-> OTN5　Channel 5的OT-

pstINTSource-> OTN6　Channel 6的OT-

pstINTSource-> OTN7　Channel 7的OT-

c. IMP Series 運動控制平台 Home Sensor

pstINTSource-> HOME0　Channel 0的Home Sensor

pstINTSource-> HOME1　Channel 1的Home Sensor

pstINTSource-> HOME2　Channel 2的Home Sensor

pstINTSource-> HOME3　Channel 3的Home Sensor

pstINTSource-> HOME4　Channel 4的Home Sensor

pstINTSource-> HOME5　Channel 5的Home Sensor

pstINTSource-> HOME6　Channel 6的Home Sensor

pstINTSource-> HOME7　Channel 7的Home Sensor

此外 LIO 中斷服務函式的宣告需加上關鍵字_stdcall，但如果使用的是 Standalone 模式則需要加上 IMC_LIB_CALL 此關鍵字。

2. 串接使用者自訂的 LIO 中斷服務函式。

在初始化運動控制卡後需串接 LIO 中斷服務函式，將中斷服務函式的函式指標傳入 IMC_LIO_SetISRFunction()，如下：

IMC_LIO_SetISRFunction(LIO_ISR_Function);

3. 使用 IMC_LIO_SetPlusLimitTriggerMode()/IMC_LIO_SetMinus LimitTriggerMode()/IMC_LIO_SetHomeSensorTriggerMode() 設定 OTP 0 ~ OTP 7 / OTN 0 ~ OTN 7 / HOME 0 ~ HOME 7 接點中斷觸發型態為上緣觸發或是下緣觸發或是轉態觸發。

4. 使用 IMC_LIO_EnablePlusLimitInterrupt()/IMC_LIO_EnableMinus LimitInterrupt()/IMC_LIO_EnableHomeSensorInterrupt() 開啟極限中斷功能。

下面的程式碼說明如何使用極限中斷。

```
void _stdcall LIO_ISR_Function(LIOINT *pstINTSource)
{
   //  判斷是否發生極限中斷
   if (pstINTSource-> OTP0)
   {
      /*
      發生過行程極限時的緊急處理動作
      */
   }
}
.
.
if (IMC_OpenDevice())
{
    .
    .
   IMC_LIO_SetISRFunction(LIO_ISR_Function);
```

```
    //   開啟 OTP0 中斷觸發功能
    IMC_LIO_SetPlusLimitTriggerMode(LIO_OTP0, LIO_INT_FALL);
    IMC_LIO_EnablePlusLimitInterrupt(LIO_OTP0, 1);
      .
      .
}
```

V.3 計時器計時中斷

IMP Series 運動控制卡提供 32 Bits 的計時器，使用者可設定計時器的計時時間，而在計時終了時將觸發計時器計時中斷(簡稱計時中斷)，並重新開始計時，此過程將持續至使用者關閉此項功能為止。要使用計時中斷必須完成下列幾項設定步驟：

1. 定義與宣告使用者自訂的計時器中斷服務函式，計時器中斷服務函式必須遵循下面的定義：

```
typedef void(IMC_LIB_CALL *TMRISR)(TMRINT *p);
```

因此使用者自訂的計時器中斷服務函式可定義如下：

```
void _stdcall Timer_ISR_Function(TMRINT *pstINTSource)
{
        //   判斷是否發生計時中斷
        if (pstINTSource->TIMER)
        {
            /*
            計時終了時欲執行的程式碼
            */
        }
```

}

在計時器中斷服務函式中需使用pstINTSource->TIMER，判斷此函式是否由計時中斷所觸發。此外計時器中斷服務函式的宣告需加上關鍵字_stdcall，但如果使用的是Standalone模式則需要加上IMC_LIB_CALL此關鍵字。

2. 串接使用者自訂的計時器中斷服務函式。
   在初始化運動控制卡後需串接計時器中斷服務函式，將中斷服務函式的函式指標傳入 IMC_TMR_SetISRFunction()，如下：

   IMC_TMR_SetISRFunction(Timer_ISR_Function);

3. 使用 IMC_TMR_SetTimer()設定計時器之計時時間，計時單位為System Clock(10ns)。

4. 使用 IMC_TMR_SetTimerIntEnable()開啟計時器中斷功能。
   ➔ *See Also* IMC_TMR_GetTimerIntEnable()

5. 使用 IMC_TMR_SetTimerEnable()開啟計時器計時功能。
   ➔ *See Also* IMC_TMR_GetTimerEnable()

下面程式碼說明如何使用計時中斷功能。

```
void _stdcall Timer_ISR_Function(TMRINT *pstINTSource)
{
   //   判斷是否發生計時中斷
   if (pstINTSource->TIMER)
   {
      /*
```

```
        計時終了時欲執行的程式碼
        */
    }
}
.
.
if (IMC_OpenDevice())
    {
     .
     .
    IMC_TMR_SetISRFunction(Timer_ISR_Function);
    //   設定 TMR 計時器計時時間為 10ns x 1000000 = 10ms
    IMC_TMR_SetTimer(1000000, 0);
    IMC_TMR_SetTimerIntEnable(1);    //   開啟計時中斷功能
    IMC_TMR_SetTimerEnable(1);       //   開啟計時器計時功能
     .
     .
    }
```