EPCIO Series

驅動函式庫

使用手冊

版本：V.6.00

日期：2020.06

http://www.epcio.com.tw

目 錄

1. 驅動函式庫簡介

EPCIO Series 驅動函式庫可用來驅動利用 EPCIO ASIC 所設計開發的 PCI 與 PCIe 介面之控制板以及輸出入控制模組，PCI 介面之運動控制卡包括 EPCIO-4000、EPCIO-4005、EPCIO-6000 與 EPCIO-6005，PCIe 之運動控制卡包括 EPCIO-6000e 與 EPCIO-6005e，輸出入控制模組包括 EDIO-S003、EDIO-S005 與 EDIO-S003H。

EPCIO Series 驅動函式庫共有超過 100 個函式可供使用者呼叫，依驅動運動控制卡上不同輸出或輸入功能區分成八大部份。

▲Bus Interface	運動控制卡初始化、中斷及重置功能(系統功能)
▲DDA Control Interface	脈波輸出控制(DDA)
▲Encoder Counter Interface	編碼器控制(ENC)
▲Local I/O Control Interface	近端輸出入接點控制(LIO)
▲Remote I/O Control Interface	遠端輸出入接點控制(RIO)
▲ADC Control Interface	類比轉數位輸入控制(ADC)
▲DAC Control Interface	數位轉類比輸出控制(DAC)
▲PCL Control Interface	硬體位置閉迴路控制(PCL)

相關參考手冊：

硬體相關資訊

EPCIO –4000/4005　硬體使用手冊

EPCIO –6000/6005　硬體使用手冊

EPCIO –6000e/6005e　硬體使用手冊

驅動程式使用導引

EPCIO Series 驅動函式庫參考手冊

EPCIO Series 驅動函式庫範例手冊

EPCIO Series 驅動函式庫整合測試環境使用手冊

2. 運動控制卡初始化、中斷及重置功能(系統功能)

使用 EPCIO Series 驅動函式庫的第一步須先初始化運動控制卡，可以使用下列函式初始化運動控制卡：

EPCIO4000_Init()	適用於 EPCIO-4000、EPCIO-4005
EPCIO6000_Init()	適用於 EPCIO-6000、EPCIO-6005
EPCIO6000e_Init()	適用於 EPCIO-6000e、EPCIO-6005e

以 EPCIO-6000 運動控制卡為例，使用 EPCIO6000_Init()進行初始化，下面將說明如何初始化運動控制卡。此函式的函式宣告如下：

```
BOOL EPCIO6000_Init( DDAISR     pfnDDAInterrupt,
                     ENCISR     pfnENC0Interrupt ,
                     ENCISR     pfnENC1Interrupt ,
                     ENCISR     pfnENC2Interrupt ,
                     RIOISR     pfnRIO0Interrupt ,
                     RIOISR     pfnRIO1Interrupt ,
                     ADCISR     pfnADCInterrupt,
                     LIOISR     pfnLIOInterrupt,
                     PCLISR     pfnPCLInterrupt,
                     WORD       wCardIndex)
```

pfnDDAInterrupt ~ pfnPCLInterrupt 為使用者自訂的中斷服務函式的函式指標，如不需使用相關模組的中斷功能，只需要在相關的參數位置傳入 NULL 即可。

wCardIndex 為運動控制卡編號，編號範圍 0 ~ 11，由使用者自行選定。在 EPCIO Series 驅動函式庫中利用此項編號來識別運動控制卡，因此不同的運動

控制卡需選擇不同的編號。因編號範圍的限制，在同一部 PC 中，最多只能同時使用 12 張 EPCIO Series 運動控制卡。下面程式碼為初始化運動控制卡的函式：

```
EPCIO6000_Init( DDA_ISR_Function, NULL, NULL, NULL, NULL,
                NULL, NULL, NULL, NULL, 0);
```

DDA_ISR_Function 為使用者自訂的 DDA 中斷服務函式的函式指標，使用者除了可使用初始化函式(例如 EPCIO6000_Init())串接使用者自訂的中斷服務程式外，也可使用 EPCIO_SetISRFunction()串接中斷服務程式，不過此函式需使用在呼叫初始化函式之前。更詳細的說明請參閱"***EPCIO Series 驅動函式庫參考手冊***"。

EPCIO6000_Init()的函式傳回值若為 TRUE(1)，表示初始化運動控制卡成功，此時方可繼續使用其他函式。

而要關閉 EPCIO Series 運動控制卡，需使用 EPCIO_Close()。本函式會關閉 EPCIO 模組內所有的功能，若使用運動控制卡時有啟動中斷功能，此時亦會還原中斷向量。

下面程式碼說明如何使用 EPCIO Series 驅動函式庫，並利用 EPCIO_ResetModule()重置所指定的 EPCIO 模組，此函式通常會與初始化函式搭配使用。

```
if (EPCIO6000_Init(DDA_ISR_Function, NULL, NULL, NULL, NULL,
                   NULL, NULL, NULL, NULL, 0))
{
    //   重置編號為 0 的 EPCIO Series 運動控制卡
    EPCIO_ResetModule(RESET_ALL, 0);

    /*
    使用者欲執行的程式碼
    */
```

```
    EPCIO_Close(0);//    關閉編號為 0 的 EPCIO Series 運動控制卡
}
```

3. 脈波輸出控制(DDA)

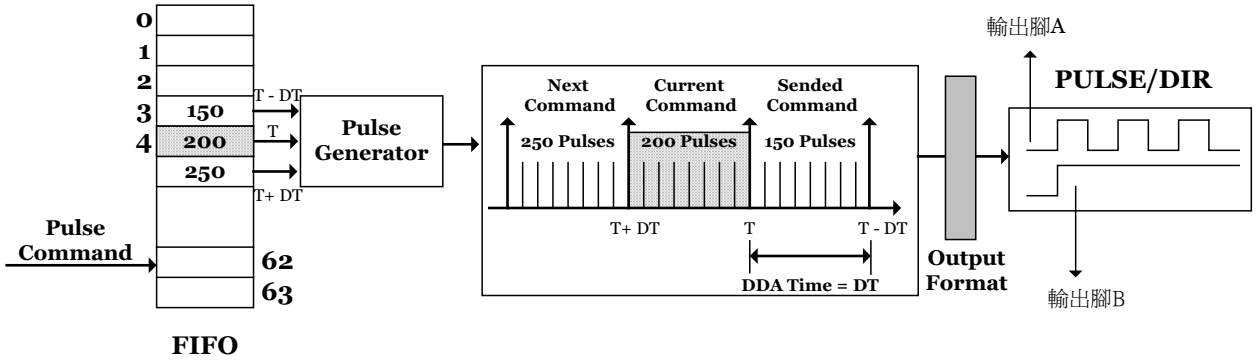
3.1 基本的脈波輸出控制

1 張 EPCIO Series 運動控制卡最多可擁有 6 個輸出 Channels(Channel 0 ~ Channel 5)，並使用 DDA 機制控制脈波輸出，此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 DDA Time 為時間間隔(DDA 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 DDA 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 EPCIO-6000 運動控制卡為例，共有 6 個輸出 Channel，因此擁有 6 個 FIFO。此圖也表示每一個 DDA 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 EPCIO_DDA_SetTime()設定 DDA Time

2. 設定所指定 Channel 的脈波輸出格式，包括設定：

(i) 使用 EPCIO_DDA_SetPulseWidth()設定脈波輸出寬度，單位為 40ns。

(ii) 輸出腳 A 訊號線是否反相。

➔ *See Also* EPCIO_DDA_EnableOutAInverse()
EPCIO_DDA_DisableOutAInverse()

(iii)輸出腳 B 訊號線是否反相。

➔ *See Also* EPCIO_DDA_EnableOutBInverse()
EPCIO_DDA_DisableOutBInverse()

(iv) 輸出腳 A 及 B 兩訊號線是否對調。

➔ *See Also* EPCIO_DDA_EnableABSwap()
EPCIO_DDA_DisableABSwap()

3. 使用 EPCIO_DDA_EnableOutputChannel()開啟指定 Channel 的輸出功能
➔ *See Also* EPCIO_DDA_DisableOutputChannel()

4. 使用 EPCIO_DDA_StartEngine()啟動 DDA Engine 機制
➔ *See Also* EPCIO_DDA_StopEngine()

5. 使用 EPCIO_DDA_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 EPCIO Series 運動控制卡中，當有位置(脈波)或速度(電壓)命令輸出，**均需使用 EPCIO_LIO_EnablePulseDAC()開啟輸出功能**；某些伺服系統可能需要呼叫 EPCIO_LIO_ServoOn()，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 Card 0 的脈波及速度命令輸出功能
EPCIO_LIO_EnablePulseDAC(0);
```

```
//  開啟 Card 0 的 Channel 0 之 Servo-On 接點
EPCIO_LIO_ServoOn(0);
//  設定 DDA Time = 10 ms，DDA Length = 15 bits
EPCIO_DDA_SetTime(10, DDA_LEN15, 0);

//  設定 Card 0 的 Channel 0 之脈波輸出格式為 Pulse/Dircetion 格式
EPCIO_DDA_SetOutputFormat(0, DDA_FMT_PD, 0);

//  設定 Card 0 的 Channel 0 之脈波輸出寬度為 100 個 System Clocks
EPCIO_DDA_SetPulseWidth(0, 100, 0);

//  開啟 Card 0 的 Channel 0 輸出功能
EPCIO_DDA_EnableOutputChannel(0, 0);

//  啟動 DDA Engine 機制
EPCIO_DDA_StartEngine(0);

//  發出脈波命令，要求 Card 0 的 Channel 0 在 1 個 DDA Time 內送出 200 個
//  脈波
EPCIO_DDA_SendPulse(0, 200, 0);
```

EPCIO_DDA_SetTime()與 EPCIO_DDA_SetBitLength()能設定每個 DDA Time 可送出的脈波總數之上限，因此使用 EPCIO_DDA_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。

3.2 控制脈波命令暫存器(FIFO)

EPCIO Series 驅動函式庫提供下列功能，用來控制與讀取 FIFO 的狀態：

1. 可使用 EPCIO_DDA_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 EPCIO_DDA_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 EPCIO_DDA_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。

4. 可使用 EPCIO_DDA_EraseFIFOCmd()移除所指定 Channel 的 FIFO 中，所儲存但尚未被執行的命令，一次最多可刪除 64 筆命令，注意，正在執行的命令並不受影響。

5. 可使用 EPCIO_DDA_ShiftOutFIFOCmd()移除所指定 Channel 的 FIFO 中下一筆待執行的命令。使用此功能移除 FIFO 中的命令時，必須先停止該 Channel 的輸出功能(也就是需先呼叫 EPCIO_DDA_DisableOutputChannel())，只有已停止輸出的 Channel， FIFO 中下一筆待送出的命令才能被移除，執行中的 Channel 將不受影響。

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，將可提升系統運作的穩定性，尤其是在使用 WINDOWS 作業系統並未搭配其它即時函式庫時。

3.3 控制送出中的脈波命令

EPCIO Series 驅動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 EPCIO_DDA_GetCurrentCmd()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 EPCIO_DDA_SetOutputFormat(指定 Channel, **DDA_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。

3.4 緊急停止脈波輸出

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

1. 可使用 EPCIO_DDA_DisableOutputChannel()關閉所指定 Channel 的輸出功能，但執行中的命令並不受影響，仍將執行完畢後，才停止送出庫存的脈波命令。

2. 可使用 EPCIO_DDA_StopEngine()關閉 DDA Engine 機制，此函式將停止所有 Channel 的輸出功能。
   ➔ *See Also* EPCIO_DDA_StartEngine()

3. 可使用 EPCIO_DDA_EnableEmgcStop()啟動緊急停止功能。本功能可在命令執行中途停止所有 Channel 輸出脈波，執行中之命令將立即停止輸出，但 EPCIO 內部仍會繼續執行脈波命令的計算。在取消緊急停止功能後，脈波命令將於下一個 DDA 周期開始輸出脈波。
   ➔ *See Also* EPCIO_DDA_DisableEmgcStop()

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

當使用上面所提及的緊急停止輸出功能時，通常會伴隨使用 EPCIO_DDA_EraseFIFOCmd()，一併清除 FIFO 中所儲存但尚未執行的脈波命

令，如下面程式碼：

```
//  先使 Card 0 的 Channel 0 之輸出無效
EPCIO_DDA_SetOutputFormat (0, DDA_FMT_NO, 0);

//  關閉 Card 0 的 Channel 0 輸出功能
EPCIO_DDA_DisableOutputChannel(0, 0);

//  清除 Card 0 的 Channel 0 之 FIFO 中所有儲存的脈波命令
EPCIO_DDA_EraseFIFOCmd(0, 64, 0);
```

3.5 已輸出的脈波總數計數

EPCIO Series 驅動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 EPCIO_DDA_EnablePulseCounter()開啟所指定 Channel 的脈波計數功能。

   ➔ *See Also* EPCIO_DDA_DisablePulseCounter()

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

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

   ➔ *See Also* EPCIO_DDA_GetOutputPulse()

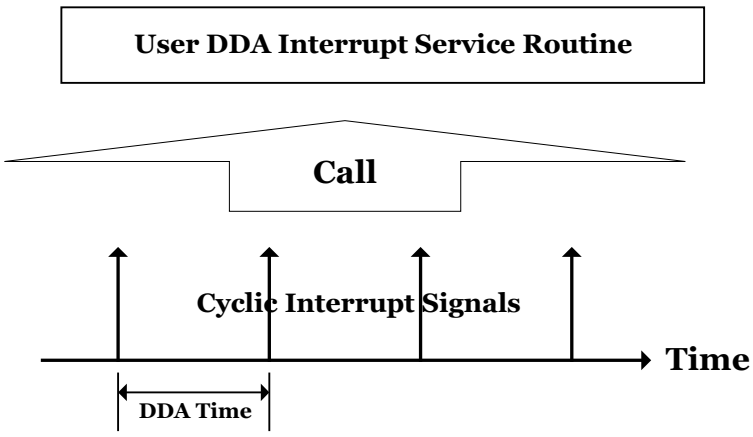
在使用 EPCIO_DDA_GetOutputPulse()前需先開啟計數功能，也就是需先呼叫 EPCIO_DDA_EnablePulseCounter()。下面程式碼說明如何讀取 Channel 0 實際送出的脈波總數。

```
//  清除 Card 0 的 Channel 0 之脈波計數器的計數值
EPCIO_DDA_ClearPulseCounter(0, 0);

//  開啟 Card 0 的 Channel 0 之脈波計數功能
EPCIO_DDA_EnablePulseCounter(0, 0);
.
.
long lPulseCount;

//  讀取 Card 0 的 Channel 0 實際送出之脈波總數
EPCIO_DDA_GetOutputPulse(0, &lPulseCount, 0);
```

通常會使用脈波計數器的計數內容與實際使用 EPCIO_DDA_SendPulse() 所送出的脈波數相比較，以便驗證是否正確的使用 EPCIO_DDA_SetPulseWidth()、EPCIO_SetTime()與 EPCIO_SetBitLength()。

3.6 循環中斷功能

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



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 DDA 中斷服務函式，此函式必須遵循下面的宣告：

```
typedef void(_stdcall *DDAISR)(DDAINT*);
```

使用者自訂的 DDA 中斷服務函式可定義如下：

```
void _stdcall DDA_ISR_Function(DDAINT *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 DDA 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 DDA 中斷服務函式的宣告需加上關鍵字_stdcall。

2. 串接使用者自訂的 DDA 中斷服務函式

在初始化運動控制卡時需串接 DDA 中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_init()，如下：

```
EPCIO6000_Init( DDA_ISR_Function, NULL, NULL, NULL, NULL,   NULL,
NULL, NULL, NULL, 0);
```

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

➔ *See Also* EPCIO_DDA_DisableCycleInt()

下面程式碼將說明如何使用循環中斷功能：

```
void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
    //  判斷循環中斷是否發生
    if (pstINTSource->CYCLE)
    {
        /*
        循環中斷發生後欲執行的程式碼
        */
      }
}
    .
    .
if (EPCIO6000_Init(DDA_ISR_Function, NULL, NULL, NULL, NULL,
                   NULL, NULL, NULL, NULL, 0))
{
    .
    .
    //  開啟循環中斷功能
    EPCIO_DDA_EnableCycleInt(0);
    .
}
```

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運動的需要，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：

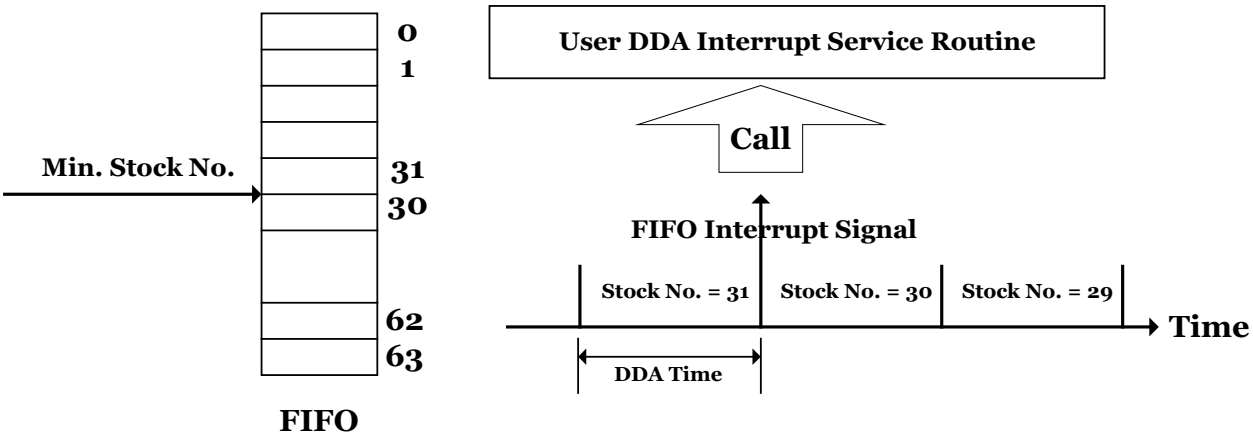
```
int nCount = 200;          //   共需送出 200 筆脈波命令
int nPulse[200] = 150;     //   200 筆命令，命令可預先規劃

void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
   WORD wSockNo;

   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 Card 0 的 Channel 0 之 FIFO 中目前儲存的命令筆數
         EPCIO_DDA_GetStockNo(0, &wStockNo, 0);

         //   “i < 64 – wStockNo”是因為 FIFO 只具 64 筆儲存空間
         for (int i = 0;i < 64 - wStockNo && nCount;i++)
         {
            EPCIO_DDA_SendPulse(0, nPulse[200 - nCount], 0)
            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。需注意的是從循環中斷發生到觸發中斷服務函式執行，中間存在未確定的中斷延遲(Interrupt Latency)，尤其是使用 WINDOWS 作業系統須特別注意此項時間延遲對系統性能的影響。

3.7 FIFO 最低庫存數目中斷

EPCIO Series 驅動函式庫提供 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 DDA 中斷服務函式，如下圖。



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 DDA 中斷服務函式，此函式必須遵循下面的宣告：

```
typedef void(_stdcall *DDAISR)(DDAINT*);
```

使用者自訂的 DDA 中斷服務函式可定義如下：

```
void _stdcall DDA_ISR_Function(DDAINT *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 DDA 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->FIFO5)分別用來判斷是否發生 Channel 0 ~ Channel 5 的 FIFO 中斷。此外 DDA 中斷服務函式的宣告需加上關鍵字_stdcall。

2. 串接使用者自訂的 DDA 中斷服務函式。

   在初始化運動控制卡時需串接 DDA 中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_Init()，如下：

```
EPCIO6000_Init(  DDA_ISR_Function, NULL, NULL, NULL, NULL,
                 NULL, NULL, NULL, NULL, 0);
```

3. 使用 EPCIO_DDA_SetMinStockNo()設定 FIFO 最低庫存數目。假設此時呼叫 EPCIO_DDA_SetMinStockNo(30, 0)，意味只要當 FIFO 所儲存的命令筆數由 31 筆消耗至 30 筆時，FIFO 中斷才會發生；**這意味著 FIFO 所儲存的命令筆數若一直等於或小於 30 筆，則中斷將永遠不會發生**。

4. 使用 EPCIO_DDA_EnableStockInt()開啟所指定 Channel 的 FIFO 中斷功能。

   ➔ *See Also*　EPCIO_DDA_DisableStockInt()

下面程式碼說明如何使用 FIFO 中斷功能：

```
void _stdcall DDA_ISR_Function(DDAINT *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 (EPCIO6000_Init(DDA_ISR_Function, NULL, NULL, NULL, NULL,
                   NULL, NULL, NULL, NULL, 0))
{
    •
    •
   //   設定 Card 0 的 Channel 0 之 FIFO 最低庫存數目為 30
   EPCIO_DDA_SetMinStockNo(30, 0);
   EPCIO_DDA_EnableStockInt(0);
    •
    •
}
```

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。

但為了產生觸發 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
```

```
//   先送出 64 筆命令至 Card 0 的 Channel 0 之 FIFO
for (int i = 0;i < 64;i++)
{
    EPCIO_DDA_SendPulse(0, nPulse[200 - nCount], 0)
    nCount--;
}
•
•
void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
    WORD wStockNo;
    if (pstINTSource->FIFO0)
    {
       if (nCount)//   送完 200 筆命令時 nCount 等於 0
       {
          //   讀取 Card 0 的 Channel 0 之 FIFO 目前儲存的命令筆數
          EPCIO_DDA_GetStockNo(0, &nStockNo, 0);

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

FIFO 中斷與循環中斷相同，從 FIFO 中斷發生到觸發中斷服務函式執行，中間存在未確定的中斷延遲，尤其是使用 WINDOWS 作業系統須特別注意此項時間延遲對系統性能的影響。

4. 編碼器控制(ENC)

4.1 基本設定與功能

EPCIO Series 運動控制卡最多擁有 9 個 Channels 可輸入編碼器(ENC)訊號，編號分別為 Channel 0 ~ Channel 8，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 EPCIO_ENC_SetFilterClock()開啟編碼器濾波取樣功能並設定取樣頻率，取樣頻率為 System Clock / (Divider + 1)。當取樣頻率設定完成後，輸入訊號必須滿足連續 3 個以上取樣值為 High 或 Low，才算是有效的輸入值。

2. 使用 EPCIO_ENC_SetInputType()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回授訊號時，請參考馬達或驅動器設定；當接一般手輪(MPG)時請設定為 A/B Phase 輸入(預設為 A/B Phase 輸入)。

3. 使用 EPCIO_ENC_SetInputRate()設定所指定 Channel 的編碼器計數器訊號回授倍率。回授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 EPCIO_ENC_SetInputType()設定為 A/B Phase 輸入。

4. 使用 EPCIO_ENC_StartInput()開啟編碼器計數器的計數功能，在使用此函式前通常會先呼叫 EPCIO_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 EPCIO_ENC_GetValue()讀取所指定 Channel 的編碼器計數值。

下面程式碼說明如何讀取 Channel 0 的計數值：

```
long lCounter;
```

```
//  設定 Card 0 濾波取樣時脈
EPCIO_ENC_SetFilterClock(2, 0);

//  設定 Card 0 的 Channel 0 之輸入格式為 A/B Phase
EPCIO_ENC_SetInputType(0, ENC_TYPE_AB, 0);

//  設定 Card 0 的 Channel 0 之訊號回授倍率為 x4
EPCIO_ENC_SetInputRate(0, ENC_RATE_X4, 0);

//  使 Card 0 的 Channel 0 之計數器之計數值歸零
EPCIO_ENC_ClearCounter(0, 0);

//  開啟 Card 0 計數功能
EPCIO_ENC_StartInput(0);

//  讀取 Card 0 的 Channel 0 之計數器計數值
EPCIO_ENC_GetValue(0, &counter, 0);
```

為了配線的實際需要，EPCIO Series 驅動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 EPCIO_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* EPCIO_ENC_DisableInAInverse()

2. 使用 EPCIO_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* EPCIO_ENC_DisableInBInverse()

3. 使用 EPCIO_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* EPCIO_ENC_DisableInCInverse()

4. 使用 EPCIO_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* EPCIO_ENC_DisableInABSwap()

4.2 編碼器計數值觸發中斷服務函式功能

EPCIO Series 驅動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(_stdcall *ENCISR)(ENC INT*);
```

使用者自訂的 ENC 中斷服務函式可定義如下：

```
void _stdcall ENC_ISR_Function1(ENCINT *pstINTSource)
{
   //   判斷是否由該組第 0 個 Channel 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，將 Channel 0 ~ Channel 8 以三個 Channels 為一組(也就是分為 Channel 0 ~ Channel 2、Channel 3 ~ Channel 5、Channel 6 ~ Channel 8 共三組)，各自使用相對應的 ENC 中斷服務函式，並在計數值觸發中斷服務函式後，使用 COMP0 ~ COMP2 判斷是由哪一個 Channel 的編碼器之計數值觸發此函式。各 Channel 與 COMP0 ~ COMP2 的關係如下面所示：

|---ENC_ISR_Function1()---|
Channel 0：pstINTSource ->COMP0
Channel 1：pstINTSource ->COMP1
Channel 2：pstINTSource ->COMP2

|---ENC_ISR_Function2()---|
Channel 3：pstINTSource ->COMP0
Channel 4：pstINTSource ->COMP1
Channel 5：pstINTSource ->COMP2

|---ENC_ISR_Function3()---|
Channel 6：pstINTSource ->COMP0
Channel 7：pstINTSource ->COMP1
Channel 8：pstINTSource ->COMP2

因此(pstINTSource->COMP0) ~ (pstINTSource->COMP2)分別用來判斷中斷服務函式是否由該組第 0 個 Channel ~ 第 2 個 Channel 的編碼器計數值所觸發，至於 Channel 編號為何(範圍 0 ~ 8)，需視 ENC 中斷服務函式之函式指標傳入初始化函式(例如 EPCIO6000_Init())時的位置而定，下一步驟會再與予說明。此外 ENC 中斷服務函式的宣告需加上關鍵字_stdcall。

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時需串接 ENC 中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_Init()，如下所示：

```
EPCIO6000_Init( NULL,
                ENC_ISR_Function1,//  針對 Channel 0 ~ Channel 2
                ENC_ISR_Function2,//  針對 Channel 3 ~ Channel 5
                ENC_ISR_Function3,//  針對 Channel 6 ~ Channel 8
                NULL, NULL, NULL, NULL, NULL, 0);
```

其中 ENC_ISR_Function1、ENC_ISR_Function2、ENC_ISR_Function3 分別表示 Channel 0 ~ Channel2、Channel 3 ~ Channel 5 、Channel 6 ~ Channel 8 所使用的中斷服務函式。

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

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

➔ *See Also* EPCIO_ENC_DisableCompInt()

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，注意 ENC_ISR_Function()的函式指標傳入 EPCIO6000_Init()時的位置。

```
void _stdcall ENC_ISR_Function2(ENCINT *pstINTSource)
{
   //   判斷是否由第 4 個 Channel 的編碼器計數值所觸發
   if (pstINTSource->COMP1)
   {
      /*
      編碼器計數值觸發中斷服務函式後欲執行的程式碼
      */
   }
}

if (EPCIO6000_Init( NULL, NULL, ENC_ISR_Function2,
                    NULL, NULL,NULL, NULL, NULL, NULL, 0))
{
    //   設定 Card 0 的 Channel 4 之計數器比較值為 10000
    EPCIO_ENC_SetCompValue(4, 10000, 0);

    //   開啟 Card 0 的 Channel 4 之編碼器計數值觸發中斷服務函式
    //   功能
    EPCIO_ENC_EnableCompInt(4, 0);
     .
     .
}
```

4.3 編碼器 INDEX 中斷

EPCIO Series 驅動函式庫提供編碼器 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(_stdcall *ENCISR)(ENC INT*);
```

使用者自訂的 ENC 中斷服務函式可定義如下：

```
void _stdcall ENC_ISR_Function1(ENCINT *pstINTSource)
{
   //   判斷是否由此組第 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，將 Channel 0 ~ Channel 8 以三個 Channels 為一組(也就是分為 Channel 0 ~ Channel 2、Channel 3 ~ Channel 5、Channel 6 ~ Channel 8 共三組)，各自使用相對應的 ENC 中斷服務函式，並在 INDEX 中斷發生後，使用 INDEX0 ~ INDEX2 判斷是由哪一個 Channel 的 INDEX 訊號所觸發。各 Channel 與 INDEX0 ~ INDEX2 的關係如下面所示：

```
|---ENC_ISR_Function1()---|
Channel 0：pstINTSource ->INDEX0
Channel 1：pstINTSource ->INDEX1
Channel 2：pstINTSource ->INDEX2
```

|---ENC_ISR_Function2()---|
Channel 3：pstINTSource ->INDEX0
Channel 4：pstINTSource ->INDEX1
Channel 5：pstINTSource ->INDEX2

|---ENC_ISR_Function3()---|
Channel 6：pstINTSource ->INDEX0
Channel 7：pstINTSource ->INDEX1
Channel 8：pstINTSource ->INDEX2

因此(pstINTSource->INDEX0) ~ (pstINTSource->INDEX2)分別用來判斷是否發生該組第 0 個 Channel ~ 第 2 個 Channel 的 INDEX 中斷，至於 Channel 編號為何(Channel 0 ~ 8)，需視 ENC 中斷服務函式之函式指標傳入初始化函式(例如 EPCIO6000_Init())時的位置而定，下一步驟會再與予說明。此外 ENC 中斷服務函式的宣告需加上關鍵字_stdcall。

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時需串接 ENC 中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_Init()，如下：

```
EPCIO6000_Init( NULL,
                ENC_ISR_Function1,//  針對 Channel 0~Channel 2
                ENC_ISR_Function2,//  針對 Channel 3~Channel 5
                ENC_ISR_Function3,//  針對 Channel 6~Channel 8
                NULL, NULL, NULL, NULL, NULL, 0);
```

其中 ENC_ISR_Function1、ENC_ISR_Function2、ENC_ISR_Function3 分別表示 Channel 0~Channel2、Channel 3~Channel 5 、Channel 6~Channel 8 所使用的 INDEX 中斷服務函式。

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

➔ *See Also* EPCIO_ENC_DisableIndexInt()
EPCIO_ENC_GetIndexStatus()

下面的程式碼只開啟 Channel 5 的 INDEX 中斷功能，注意 ENC_ISR_Function()的函式指標傳入 EPCIO6000_Init()時的位置。

```
void _stdcall ENC_ISR_Function(ENCINT *pstINTSource)
{
   //   判斷是否由 Card 0 的第 5 個 Channel 之 INDEX 中斷所觸發
   if (pstINTSource->INDEX2)
   {
         /*
         INDEX 中斷發生後欲執行的程式碼
         */
   }
}
.
.
if (EPCIO6000_Init( NULL, NULL, ENC_ISR_Function, NULL, NULL,
                    NULL, NULL, NULL, NULL, 0))
{
     .
     .
    //   開啟 Card 0 的 Channel 5 之 INDEX 中斷觸發功能
    EPCIO_ENC_EnableIndexInt(5, 0);
}
```

4.4 計數器計數值 Latch 功能

EPCIO Series 驅動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記錄在閂鎖暫存器內的動作，使用者並可使用驅動函式庫提供的函式，讀取閂鎖暫存器內的紀錄值。

使用計數器計數 Latch 功能需先使用 EPCIO_ENC_SetTrigSource()設定觸發訊號源，此函式的函式宣告如下：

```
BOOL EPCIO_ENC_SetTrigSource(  WORD wChannel,
                               WORD wSource,
                               WORD wCardIndex);
```

wChannel 表示 Channel 的編號，範圍為 0 ~ 8。wSource 則為觸發訊號源，共有 15 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：

ENC_TRIG_NO	沒有選擇任何觸發訊號源
ENC_TRIG_INDEX0	Channel 0 編碼器的 INDEX 訊號
ENC_TRIG_INDEX1	Channel 1 編碼器的 INDEX 訊號
ENC_TRIG_INDEX2	Channel 2 編碼器的 INDEX 訊號
ENC_TRIG_INDEX3	Channel 3 編碼器的 INDEX 訊號
ENC_TRIG_INDEX4	Channel 4 編碼器的 INDEX 訊號
ENC_TRIG_INDEX5	Channel 5 編碼器的 INDEX 訊號
ENC_TRIG_INDEX6	Channel 6 編碼器的 INDEX 訊號
ENC_TRIG_INDEX7	Channel 7 編碼器的 INDEX 訊號
ENC_TRIG_INDEX8	Channel 8 編碼器的 INDEX 訊號
ENC_TRIG_LIO0	發生近端的輸入接點 DI 0 中斷
ENC_TRIG_LIO1	發生近端的輸入接點 DI 1 中斷
ENC_TRIG_RDI0	發生遠端輸入接點 Set 0 的 DI 0 中斷
ENC_TRIG_RDI1	發生遠端輸入接點 Set 1 的 DI 1 中斷

ENC_TRIG_ADC0　　發生 ADC Channel 0 的比較條件成立

ENC_TRIG_ADC1　　發生 ADC Channel 1 的比較條件成立

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記錄在閂鎖暫存器內。不過在使用 EPCIO_ENC_StartInput()開始計數器 Latch 功能前，需先呼叫 EPCIO_ENC_SetTrigMode()設定 Latch 模式。此函式的函式宣告如下：

```
EPCIO_ENC_SetTrigMode( WORD wChannel,
                       WORD wMode,
                       WORD wCardIndex)
```

wChannel 表示 wChannel 的編號，範圍為 0 ~ 8。wMode 為 Latch 觸發模式，設定值可為：

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 EPCIO_ENC_GetLatchValue()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
        EPCIO_ENC_GetLatchValue(0, &lLatchValue, 0)
    }
}

if (EPCIO6000_Init(NULL, ENC_ISR_Function, NULL, NULL, NULL,
                   NULL, NULL, NULL, NULL, 0))
{
    .
    .
    //   設定 Card 0 的 Channel 0 閂鎖計數器的觸發源為該 Channel 編碼器之
    //   INDEX 訊號
    EPCIO_ENC_SetTrigSource(0, ENC_TRIG_INDEX0, 0);

    //   設定 Card 0 的 Channel 0 閂鎖計數器之 Latch 觸發模式為連續觸發
    EPCIO_ENC_SetTrigMode(0, ENC_TRIG_LAST, 0);
    .
    .
}
```

5. 近端輸出入接點控制(LIO)

近端輸出入接點(Local Input/Output, LIO)共有 28 個可規劃為輸出或輸入的 IO 接點，不過在 EPCIO Series 運動控制卡上已規劃好這些 IO 的特定用途，包括：

輸入接點總共 20 個接點，包括：

Home Sernsor	共 6 個輸入接點
Limit Switch Plus(+)	共 6 個輸入接點
Limit Switch Minus(-)	共 6 個輸入接點
Status for Getting 24V	共 1 個輸入接點
Status for Emgergency Stop	共 1 個輸入接點

輸出接點總共 8 個接點，包括：

Servo-On/Off	共 6 個輸出接點
Enabling Position Ready	共 1 個輸出接點
Enabling Pulse DAC	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。

5.1 基本設定與功能

使用近端輸出接點前，必須先使用 EPCIO_LIO_EnableLDOOutput()啟動該點的輸出功能，此函式的函式宣告如下：

```
BOOL EPCIO_LIO_EnableLDOOutput(  WORD wPort,
                                 WORD wCardIndex);
```

近端數位輸出以 4 點為一個 Port，28 個 I/O 接點共分為 Port 0 ~ Port 6，此函式可單獨開啟或關閉某一個 Port 的輸出功能。所有 Port 的預設輸出狀態

設定為關閉狀態。此函式以每 4 點為一個 Port，參數 Port 可為：

LIO_OUT_EN0 表示 Port 0(LDO 0 ~ LDO 3)
LIO_OUT_EN1 表示 Port 1(LDO 4 ~ LDO 7)
LIO_OUT_EN2 表示 Port 2(LDO 8 ~ LDO 11)
LIO_OUT_EN3 表示 Port 3(LDO 12 ~ LDO 15)
LIO_OUT_EN4 表示 Port 4(LDO 16 ~ LDO 19)
LIO_OUT_EN5 表示 Port 5(LDO 20 ~ LDO 23)
LIO_OUT_EN6 表示 Port 6(LDO 24 ~ LDO 27)

➔ *See Also* EPCIO_LIO_DisableLDOOutput()

可以使用 EPCIO_LIO_GetLDIInput()來讀取 LDI 0 ~ LDI 27 的數位訊號輸入值，此函式的宣告如下：

```
BOOL EPCIO_LIO_GetLDIInput( DWORD *pdwInput,
                            WORD wCardIndex);
```

利用此函式所獲得的指標變數*pdwInput，其中 bit 0 ~ bit 27 用來表示 LDI 0 ~ LDI 27 的輸入狀態，bit 28 ~ bit 31 無任何意義。因此若使用 EPCIO_LIO_GetLDIInput(&dwInput, 0)所獲得的 dwInput 輸入值為 0x0002，表示 Card 0 的 LDI 1 目前的數位訊號輸入值為 1，因為 0x0002 換成 2 進位制等於 0b0000000000000010，LDI 1 相關的 bit 位置之值為 1。

同樣的，可以使用 EPCIO_LIO_SetLDOOutput()來設定 LDO 0 ~ LDO 27 的數位訊號輸出值，此函式的函式宣告如下：

```
BOOL EPCIO_LIO_SetLDOOutput( DWORD dwValue,
                             WORD wCardIndex);
```

此函式的參數 dwValue 的 bit 0 ~ bit 27 用來表示 LDO 0 ~ LDO 27 的輸出

狀態，bit 28 ~ bit 31 並無任何意義。

因此使用 EPCIO_LIO_SetLDOOutput(0x0021, 0)表示對 Card 0 的 LDO 0 與 LDO 5 輸出訊號，因為 0x0021 換成 2 進位制等於 0b0000000000100001， LDO 0 與 LDO 5 相關的 bit 位置已設定為 1。需使用上面所提及的函式時，在 EPCIO-4000/4005 運動控制卡的各 bit 的意義如下所示：

LDI/LDO	定義	對應之 SCSI II 位置	備註
0	Channel 0 OT+	8（外接訊號點）	可觸發中斷
1	Channel 1 OT+	42（外接訊號點）	可觸發中斷
2	Channel 2 OT+	12（外接訊號點）	可觸發中斷
3	Channel 3 OT+	46（外接訊號點）	可觸發中斷
4	Channel 0 OT-	9（外接訊號點）	可觸發中斷
5	Channel 1 OT-	43（外接訊號點）	可觸發中斷
6	Channel 2 OT-	13（外接訊號點）	可觸發中斷
7	Channel 3 OT-	47（外接訊號點）	
8	Channel 0 HOME	7（外接訊號點）	
9	Channel 1 HOME	41（外接訊號點）	
10	Channel 2 HOME	11（外接訊號點）	
11	Channel 3 HOME	45（外接訊號點）	
12~15	保留(無使用)		
16	INH_O0	10（外接訊號點）	
17	INH_O1	44（外接訊號點）	
18	INH_O2	14（外接訊號點）	
19	INH_O3	48（外接訊號點）	
20	P_RDY(PCI Bus)	40（外接訊號點）	
21	保留(無使用)		
22	P_RDY(ISA Bus)	40（外接訊號點）	
23	PULSE_DA_OUT	非外接訊號點	

	PUT_ENABLE		
24	保留(內部使用)	非外接訊號點	
25	保留(內部使用)	非外接訊號點	
26	保留(內部使用)	非外接訊號點	
27	保留(內部使用)	非外接訊號點	

EPCIO-6000/6005/6000e/6005e 運動控制卡的各 bit 的意義如下所示：

LDI/LDO	定義	對應之 SCSI II 位置	備註
0	Channel 0 OT+	10（外接訊號點）	可觸發中斷
1	Channel 1 OT+	60（外接訊號點）	可觸發中斷
2	Channel 2 OT+	14（外接訊號點）	可觸發中斷
3	Channel 3 OT+	64（外接訊號點）	可觸發中斷
4	Channel 4 OT+	18（外接訊號點）	可觸發中斷
5	Channel 5 OT+	68（外接訊號點）	可觸發中斷
6	Channel 0 OT-	11（外接訊號點）	可觸發中斷
7	Channel 1 OT-	61（外接訊號點）	
8	Channel 2 OT-	15（外接訊號點）	
9	Channel 3 OT-	65（外接訊號點）	
10	Channel 4 OT-	19（外接訊號點）	
11	Channel 5 OT-	69（外接訊號點）	
12	Channel 0 HOME	9（外接訊號點）	
13	Channel 1 HOME	59（外接訊號點）	
14	Channel 2 HOME	13（外接訊號點）	
15	Channel 3 HOME	63（外接訊號點）	
16	INH_O0	12（外接訊號點）	
17	INH_O1	62（外接訊號點）	

18	INH_O2	16（外接訊號點）	
19	INH_O3	66（外接訊號點）	
20	INH_O4	20(外接訊號點)	
21	INH_O5	70（外接訊號點）	
22	P_RDY	58（外接訊號點）	
23	PULSE_DA_OUT_PUT_ENABLE	非外接訊號點	
24	PASS CHECK	非外接訊號點	
25	PASS CHECK	非外接訊號點	
26	PASS CHECK	非外接訊號點	
27	PASS CHECK	非外接訊號點	

EPCIO Series 運動控制卡因已規劃好這些 IO 接點的特定用途，所以如果使用 EPCIO-400-1、EPCIO-400-2、EPCIO-601-1 或 EPCIO-601-2 這些轉接版，也可以使用下面的函式，較方便對轉接板上相對應的接點作讀取或輸出的動作。使用這些函式來設定輸出接點的狀態時，因已自動開啟了輸出功能，並不需要再使用 EPCIO_LIO_EnableLDOOutput()。

EPCIO Series 驅動函式庫提供下列函式來讀取輸入接點的狀態：

1. 使用 EPCIO_LIO_GetHomeSensor()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 EPCIO_LIO_GetOverTravelUp()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, OT+)，若是則機台有可能發生撞機的危險，使用者應立即作緊急處置。EPCIO-6000、EPCIO-6005、EPCIO-6000e 與 EPCIO-6005e 的 Channel 0 ~ Channel 5 以及 EPCIO-4000 與 EPCIO-4005 的 Channel 0 ~ Channel 3，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 EPCIO_LIO_GetOverTravelDown()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, OT-)，若是則機台有可能發生撞機的危險，使用者應立即作緊急處置。EPCIO-6000、EPCIO-6005、EPCIO-6000e、EPCIO-6005e 的 Channel 0 ~ Channel 5 與 EPCIO-4000、EPCIO-4005 的 Channel 0 ~ Channel 2，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。

4. 使用 EPCIO_LIO_Get24VSensor()讀取 24 V 電壓輸入狀態。

5. 使用 EPCIO_LIO_GetEmgcStopStatus()讀取緊急停止開關狀態。

EPCIO Series 驅動函式庫提供下列函式來設定輸出接點的狀態：

1. 使用 EPCIO_LIO_ServoOff()開啟所指定 Channel 的禁止輸入接點功能。本接點可連接馬達驅動器的禁止輸入接點，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(例如呼叫 EPCIO4000_Init())，內定狀態為開啟禁止輸入功能。

2. 使用 EPCIO_LIO_ServoOn()關閉所指定 Channel 的禁止輸入接點功能。本接點可連接馬達驅動器的禁止輸入接點，當呼叫本函式設定後，所指定 Channel 將可接受來自 EPCIO Series 運動控制卡的位置或速度命令。

3. 使用 EPCIO_LIO_DisablePrdy()關閉 Position Ready 輸出接點功能。本輸出接點可連接電源開關控制接點，當呼叫本函式後，控制接點將被開路。在呼叫初始化函式成功後(例如呼叫 EPCIO4000_Init())，內定狀態為關閉 Position Ready 輸出功能。

4. 使用 EPCIO_LIO_EnblePrdy()開啟 Position Ready 輸出接點功能。本輸出接點可連接電源開關控制接點，當呼叫本函式後，控制接點將被導通。

5. 使用 EPCIO_LIO_DisablePulseDAC()關閉 EPCIO Series 運動控制卡的位置(脈波)與速度(電壓)命令輸出功能。當呼叫本函式後，輸出功能將被關閉。在呼叫初始化函式成功後(例如呼叫 EPCIO4000_Init())，內定狀態為關閉輸出功能。

6. 使用 EPCIO_LIO_DisablePulseDAC()開啟 EPCIO Series 運動控制卡的位置(脈波)與速度(電壓)命令輸出功能。當呼叫本函式後後，輸出功能將被開啟。

EPCIO Series 運動控制卡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。

5.2 硬體極限開關中斷

EPCIO Series 運動控制卡某些 Channel 的 Limit Switch Plus 與 Limit Switch Minus 接點(或稱為過行程極限接點)，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。

未提供極限中斷的 Channel，只能使用隨時檢查的方式，檢查是否碰觸到極限開關。或將未使用的 Limit Switch Plus 接點與 Limit Switch Minus 接點，作為其他 Channel 的過行程極限接點。當然，軟體也需配合，要能正確判斷發生的極限中斷來自哪一個 Channel，並判斷發生中斷的原因(是因為碰觸 Limit Switch Plus 接點或 Limit Switch Minus 接點)。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(_stdcall *LIOISR)(LIO INT*);
```

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-> LDI0)
      {
         /*
         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 LDI0 ~ LDI6，判斷此函式是否由極限中斷所觸發。LDI0 ~ LDI6 所表示的意義如下：

(i) EPCIO-4000、EPCIO-4005 之運動控制卡

pstINTSource-> LDI0 Channel 0的OT+ (Limit Switch Plus)
pstINTSource-> LDI1 Channel 1的OT+
pstINTSource-> LDI2 Channel 2的OT+
pstINTSource-> LDI3 Channel 3的OT+
pstINTSource-> LDI4 Channel 0的OT- (Limit dSwitch Minus)
pstINTSource-> LDI5 Channel 1的OT-
pstINTSource-> LDI6 Channel 2的OT-

(ii) EPCIO-6000、EPCIO-6005、EPCIO-6000e、EPCIO-6005e之運動控制卡

pstINTSource-> LDI0 Channel 0的OT+ (Limit Switch Plus)

pstINTSource-> LDI1 Channel 1的OT+

pstINTSource-> LDI2 Channel 2的OT+

pstINTSource-> LDI3 Channel 3的OT+

pstINTSource-> LDI4 Channel 4的OT+

pstINTSource-> LDI5 Channel 5的OT+

pstINTSource-> LDI6 Channel 0的OT- (Limit Switch Minus)

此外 LIO 中斷服務函式的宣告需加上關鍵字_stdcall。

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時需串接 LIO 中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_Init()，如下：

```
EPCIO6000_Init( NULL, NULL, NULL,
                NULL, NULL, NULL,
                NULL, LIO_ISR_Function, NULL, 0);
```

3. 使用 EPCIO_LIO_SetLDIIntType()設定 LDI 0 ~ LDI 6 接點，中斷觸發型態為上緣觸發(LIO_INT_RISE)或是下緣觸發(LIO_INT_FALL)或是轉態觸發(LIO_INT_LEVEL)。

4. 使用 EPCIO_LIO_EnableLDIInt()開啟 LIO 極限中斷觸發輸出功能。

下面的程式碼說明如何使用極限中斷，注意 LIO_ISR_Function()的函式指標傳入 EPCIO6000_Init()時的位置。

```
void _stdcall LIO_ISR_Function(LIOINT *pstINTSource)
{
```

```
    //   判斷是否發生極限中斷
    if (pstINTSource->LDI0)
    {
        /*
        發生過行程極限時的緊急處理動作
        */
    }
}

if (EPCIO600_Init( NULL, NULL, NULL, NULL, NULL
                   NULL, NULL, LIO_ISR_Function, NULL, 0))
{
      .
      .
    //   開啟 Card 0 的 LDI 0 中斷觸發功能
    EPCIO_LIO_SetLDIIntType(LIO_LDI0, LIO_INT_FALL , 0);
    EPCIO_LIO_EnableLDIInt(LIO_LDI0, 0);
}
```

5.3 計時器計時中斷

EPCIO Series 運動控制卡提供 24-bit 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(簡稱計時器中斷)，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(_stdcall *LIOISR)(LIO INT*);
```

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(LIOINT *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。

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時需串接計時器中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_Init()，如下：

```
EPCIO6000_Init( NULL, NULL, NULL, NULL, NULL, NULL,
                NULL, Timer_ISR_Function, NULL, 0);
```

3. 使用 EPCIO_LIO_SetTimer()設定計時器之計時時間，計時單位為 System Clock(25ns)。

4. 使用 EPCIO_LIO_EnableTimerInt()開啟計時器中斷功能。

➔ *See Also* EPCIO_LIO_DisableTimerInt()

5. 使用 EPCIO_LIO_EnableTimer()開啟計時器計時功能。

➔ *See Also* EPCIO_LIO_DisableTimer()

下面程式碼說明如何使用計時中斷功能：

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

if (EPCIO6000_Init( NULL, NULL, NULL, NULL, NULL
                    NULL, NULL, Timer_ISR_Function, NULL, 0))
{
     .
     .
     //   設定 Card 0 的 LIO 計時器計時時間為 25ns x 1000000 = 25ms
     EPCIO_LIO_SetTimer(1000000, 0);
     EPCIO_LIO_EnableTimerInt(0);//   開啟 card 0 的計時器中斷功能
     EPCIO_LIO_EnableTimer(0);//   開啟 card 0 的計時器計時功能
}
```

5.4 Watch Dog

EPCIO Series 運動控制卡提供 Watch Dog 功能。當使用者開啟 Watch Dog 功能後，必須在 Watch Dog 計時終了前(也就是 Watch Dog 的計時值等於設定值前)，清除 Watch Dog 的計時內容。否則一旦計時終了將發生 Reset 硬體的動作。要使用 Watch Dog 必須完成下列幾個步驟：

1. 使用 EPCIO_LIO_SetTimer()設定計時器的計時時間，計時單位為 System Clock(25ns)。

2. 使用 EPCIO_LIO_SetWDogTimer()設定 Watch Dog 計時器比較值，Watch Dog 計時器比較值為 16-bit 數值，使用計時器的計時時間作為 Time Base。也就是說如果使用下列的程式碼：

```
EPCIO_LIO_SetTimer(1000000, 0);
EPCIO_LIO_SetWDogTimer(2000, 0);
```

此時表示 Card 0 的 Watch Dog 計時器的比較值設定為(25ns×1000000)×2000 = 50s。

3. 使用 EPCIO_LIO_SetWDogReset()設定 Watch Dog 計時器 Reset 訊號持續時間。Watch Dog 計時器計時終了後將觸發 Reset 硬體的動作，Reset 維持時間可透過本函式規劃，設定單位為 System Clock。

4. 使用 EPCIO_LIO_EnableWDogTimer()開啟 Watch Dog 功能。
   ➔ *See Also* EPCIO_LIO_DisableWDogTimer()

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

當開啟 Watch Dog 功能後，必須在計時終了前使用 EPCIO_LIO_RefreshWDogTimer()清除 Watch Dog 計時器的計數值，使用此函式後 Watch Dog 計時器的計數值將歸零，並重新計時。下面範例說明如何使用 Watch Dog：

```
//  設定 Card 0 的計時器計時時間為 25ns x 1000000 = 25ms
EPCIO_LIO_SetTimer(1000000, 0);

//  設定 Card 0 的 Watch Dog 計時器比較值為 2000 × 25ms = 50s
EPCIO_LIO_SetWDogTimer(2000, 0);

EPCIO_LIO_EnableWDogTimer(0);//  開啟 Card 0 的 Watch Dog 功能
EPCIO_LIO_EnableTimer(0);//  開啟 Card 0 計時器的計時功能
.
.
//  需在計時終了前清除 Card 0 Watch Dog 計時器的計數值
EPCIO_LIO_RefreshWDogTimer(0);
```

使用者可搭配計時器計時中斷功能，在 Watch Dog 產生 Reset 訊號前加以警示，並在計時中斷服務函式內進行必要的處理。

6. 遠端輸出入接點控制(RIO)

EPCIO Series 運動控制卡最多可支援兩組遠端輸出入接點(Remote Input/Output, RIO)，分別為 Set 0 和 Set 1，每組 Set 含有 1 個 Master 和 3 個 Slaves，每一個 Slave 具有 64 個輸入點與 64 個輸出點，故最多可支援 384 個輸入點與 384 個輸出點。

6.1 基本設定與讀取輸出、入點狀態

使用遠端輸出入接點前，須先使用 EPCIO_RIO_SetClockDivider()設定 Remote I/O 傳送資料的傳輸時脈，此函式的宣告如下：

```
BOOL EPCIO_RIO_SetClockDivider(WORD wSet,
                               WORD wDivider,
                               WORD wCardIndex);
```

使用 wSet 指定所使用的 Set，其參數可設定為：

RIO_SET0 表示 Remote I/O Set 0

RIO_SET1 表示 Remote I/O Set 1

傳輸時脈為 System Clock(40MHz)除以 2×(wDivider +1)，wDivider 的設定範圍為 0~255，預設值為 0。

使用 EPCIO_RIO_EnableSetControl()開啟指定 Remote I/O Set 之控制功能，此函式宣告如下：

```
BOOL EPCIO_RIO_EnableSetControl(WORD wSet,
                                WORD wCardIndex);
```

➔ *See Also* EPCIO_LIO_ DisableSetControl ()

開啟指定 Set 後，尚須呼叫 EPCIO_RIO_EnableSlaveControl()才可開啟該 Set 中欲指定的 Slave 功能，此函式宣告如下：

```
BOOL EPCIO_RIO_EnableSlaveControl(WORD wSet,
                                  WORD wSlave,
                                  WORD wCardIndex);
```

其中 wSlave 參數可設定為：

RIO_SLAVE0	Remote I/O 的 Slave 0
RIO_SLAVE1	Remote I/O 的 Slave 1
RIO_SLAVE2	Remote I/O 的 Slave 2

➔ *See Also* EPCIO_LIO_ DisableSlaveControl ()

可以使用 EPCIO_RIO_GetInputValue()來讀取 DI0 ~ DI63 的輸入訊號值，此函式的宣告如下：

```
BOOL EPCIO_RIO_GetInputValue(WORD wSet,
                             WORD wSlave,
                             WORD wPort,
                             WORD* pwInput,
                             WORD wCardIndex);
```

遠端輸入接點以 16 點為一個 Port，64 個輸入點共分為 4 個 Port(Port 0 ~ Port 3)，參數 Port 可為：

RIO_PORT0	Port 0 (DI 0 ~ DI 15)
RIO_PORT1	Port 1 (DI 16 ~ DI 31)
RIO_PORT2	Port 2 (DI 32 ~ DI 47)
RIO_PORT3	Port 3 (DI 48 ~ DI 63)

利用此函式所獲得*pwInput 的 bit0 ~ bit15 用來表示該 Port 的輸入狀態，因此若使用 EPCIO_RIO_GetInputValue(RIO_SET0, RIO_SLAVE0, RIO_PORT0, &wInput, 0)，即為讀取 RIO Set 0 的 Slave 0 的 Port 0(DI 0 ~ DI 15)的輸入狀態。

同樣的，可以使用 EPCIO_RIO_SetOutputValue()來設定 DO 0 ~ DO 63 的訊號輸出值，此函式的宣告如下：

```
BOOL EPCIO_RIO_SetOutputValue(WORD wSet,
                              WORD wSlave,
                              WORD wPort,
                              WORD *wOutput,
                              WORD wCardIndex)
```

遠端輸出接點以 16 點為一個 Port，64 個輸出點共分為 4 個 Port(Port 0 ~ Port 3)，參數 Port 可為：

RIO_PORT0	Port 0 (DO0 ~ DO 15)
RIO_PORT1	Port 1 (DO 16 ~ DO 31)
RIO_PORT2	Port 2 (DO 32 ~ DO 47)
RIO_PORT3	Port 3 (DO 48 ~ DO 63)

利用此函式所設定*wOutput 的 bit 0 ~ bit 15 用來表示該 Port 的輸出狀態，因此若使用 EPCIO_RIO_GetOutputValue(RIO_SET0, RIO_SLAVE0, RIO_PORT1, &pwOutput, 0)，即為讀取 RIO Set 0 中 Slave 0 其 Port 1(DO 16 ~ DO 31)的輸出狀態。

若要確認 Master 端是否有接收到來自 Slave 端所傳輸的資料，可以使用 EPCIO_RIO_GetTransStatus()來確認傳輸狀態，當傳輸過程發生錯誤時，可呼叫 EPCIO_RIO_GetMasterStatus()及 EPCIO_RIO_GetSlaveStatus()來分辨產生錯誤的為 Master 端或 Slave 端，並呼叫 EPCIO_RIO_GetSlaveFail()來辨別接收 Master 資料發生錯誤時的 Slave 編號。

使用 EPCIO_RIO_SetTransError()設定 Remote I/O 資料傳輸錯誤時最大允許資料重傳次數，當設定 i，代表自動重送 i 次，若設定為 0 時，表示系統將於錯誤發生後自動重送最多 16 次。

6.2 遠端輸入接點訊號觸發中斷功能

每個 Remote I/O Slave 模組的前 4 個輸入點(DI 0 ~ DI 3)可設定為中斷功能觸發源，故總共有 24 點(2 組 Sets 各可串 3 個 Slaves；2<Set>x3<Slave>x4<DI>)

可用來觸發使用者自訂的中斷服務函式，要使用遠端輸入點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 RIO 中斷服務函式，RIO 中斷服務函式的宣告必需遵循下面的定義：

```
typedef void(EDDL_CALL *RIOISR)(RIOINT*);
```

使用者自訂的 RIO 中斷服務函式可定義如下：

```
void _stdcall RIO_ISR_Function(RIOINT *pstINTSource)
{
      //   判斷是否發生 RIO Slave 0 的輸入接點 DI 0 中斷
      if (pstINTSource-> S0DI0)
      {
         /*
         有遠端輸入訊號時的緊急處理動作
         */
      }
}
```

在 RIO 中斷服務函式中需使用 RIOINT 結構參數，判斷此函式是否由遠端輸入中斷所觸發。各狀態變數所表示的意義如下：

S0DI0 ~ S0DI3 Remote I/O Slave 0 的輸入接點 DI 0 ~ DI 3 觸發中斷
S1DI0 ~ S1DI3 RIORemote I/O Slave 1 的輸入接點 DI 0 ~ DI 3 觸發中斷
S2DI0 ~ S2DI3 Remote I/O Slave 2 的輸入接點 DI 0 ~ DI 3 觸發中斷
FAIL   Remote I/O Set 資料傳輸狀態

此外 RIO 中斷服務函式的宣告需加上關鍵字_stdcall。

2. 串接使用者自訂的 RIO 中斷服務函式。

   在初始化運動控制卡時需串接 RIO 中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_Init()，如下所示：

```
EPCIO6000_Init( NULL, NULL, NULL,
                NULL, RIO_ISR_Function, NULL,
                NULL, NULL, NULL, 0);
```

3. 呼叫 EPCIO_RIO_SetIntType()來設定輸入點中斷訊號觸發型態，此函式的宣告如下：

```
BOOL EPCIO_RIO_SetIntType( WORD set,
                           WORD wSlave,
                           WORD wPoint,
                           WORD wType,
                           WORD wCardIndex);
```

   輸入接點參數 wPoint 可設定為：

RIO_DI0	RIO 輸入接點 DI 0
RIO_DI1	RIO 輸入接點 DI 1
RIO_DI2	RIO 輸入接點 DI 2
RIO_DI3	RIO 輸入接點 DI 3

   觸發型態可設定為：

RIO_INT_RISE	上緣觸發
RIO_INT_FALL	下緣觸發
RIO_INT_LEVEL	轉態觸發

4. 完成設定後，必須呼叫 EPCIO_RIO_EnableInputInt()開啟 RIO 中斷功能。

   ➔ *See Also* EPCIO_RIO_DisableInputInt()

下面的程式碼說明如何使用 RIO 中斷，注意 RIO_ISR_Function()的函式指標傳入 EPCIO6000_Init()時的位置：

```
void _stdcall RIO_ISR_Function(LIOINT *pstINTSource)
{
   //   判斷是否發生 Remote I/O Slave 0 的輸入接點 DI 0 中斷
   if (pstINTSource-> S0DI0)
   {
      /*
      有遠端輸入訊號時的緊急處理動作
      */
    }
}

if (EPCIO600_Init( NULL, NULL, NULL, NULL, RIO_ISR_Function
                   NULL, NULL, NULL, NULL, 0))
{
    .
    .
    //   設定 Remote I/O 第 0 個 Set 的第 0 個 Slave 的第 0 點輸入點中斷功能為
    //   上緣觸發
    EPCIO_RIO_SetIntType(RIO_SET0, RIO_SLAVE0, RIO_DI0,
    RIO_INT_RISE, 0);

    //   開啟 Remote I/O 第 0 個 Set 的第 0 個 Slave 的第 0 點輸入點中斷功能
    EPCIO_RIO_EnableInputInt(RIO_SET0, RIO_SLAVE0, RIO_DI0, 0);
    //   開啟 Remote I/O 第 0 個 Set 傳輸控制
    EPCIO_RIO_EnableSetControl(RIO_SET0, 0);
    //   開啟 Remote I/O 第 0 個 Set 的第 0 個 Slave 傳輸功能
```

```
    EPCIO_RIO_EnableSlaveControl(RIO_SET0, RIO_SLAVE0, 0);
    .
    .
}
```

6.3 傳輸錯誤觸發中斷功能

此外可利用各 Set 發生傳輸錯誤時觸發中斷功能，2 組 Sets 可設定 2 個觸發源，呼叫 EPCIO_RIO_EnableTransInt()開啟 Remote I/O 通訊失敗中斷功能。

➔ *See Also* EPCIO_RIO_ DisableTransInt()

7. 類比轉數位輸入控制(ADC)

EPCIO Series 運動控制卡之類比轉數位(A/D Converter, ADC)功能(此功能為選配，EPCIO-4005/6005/6005e 不支援 ADC 功能)，可將類比電壓訊號(-5 ~ 5 V 或 0 ~ 10 V)輸入至 ADC 模組(EPCIO-4000 為 6 組，EPCIO-6000/6000e 為 8 組)，再由運動控制卡之 ADC 介面讀取輸入電壓值。

7.1 基本設定與功能

使用 EPCIO_ADC_SetClockDivider()設定 ADC 串列介面的傳輸時脈，此函式的宣告如下：

```
BOOL EPCIO_ADC_SetClockDivider(WORD wDivider,
                               WORD wCardIndex);
```

傳輸時脈為 System Clock(40MHz)除以 4×(wDivider +1)，wDivider 的設定範圍為 0 ~ 255，預設值為 0。

使用 EPCIO_ADC_SetConvType()設定 ADC Channel 電壓轉換型式為雙極性(Bipolar)或單極性(Unipolar)轉換型式，此函式的宣告如下：

```
BOOL EPCIO_ADC_SetConvType(WORD wChannel,
                           WORD wType,
                           WORD wCardIndex);
```

利用參數 wType 設定 ADC 電壓轉換型式，可設定值有：

ADC_TYPE_BIP　雙極性(Bipolar)轉換型式

ADC_TYPE_UNI　單極性(Unipolar)轉換型式

可以使用 EPCIO_ADC_GetConvType()讀取目前的電壓轉換型式。

7.2 連續電壓與單次電壓轉換

使用 EPCIO_ADC_SetConvMode()設定電壓轉換型態，此函式的宣告如下：

```
BOOL EPCIO_ADC_SetConvMode( WORD wMode,
                            WORD wCardIndex);
```

其中參數 wMode，可設定值有：

ADC_MODE_SINGLE　單次轉換(Single Run)

ADC_MODE_FREE　　連續轉換(Free Run)

當設定為單次轉換(Single Run)模式時，需使用 EPCIO_ADC_SetSingleChannel()選定一 ADC Channel 作為唯一可進行輸入電壓轉換的 Channel，其他 Channel 則不會執行轉換功能。呼叫 EPCIO_ADC_StartConv()後，此選定的 Channel 會將電壓值轉換一次，若需要再次轉換則需重新呼叫 EPCIO_ADC_StartConv()。

當設定為連續轉換(Free Run)模式時，需使用 EPCIO_ADC_EnableConvChannel()開啟指定 ADC Channel 之類比轉數位功能，設定完成後須呼叫 EPCIO_ADC_StartConv()方可啟動 ADC 轉換功能。

➔ *See Also*　EPCIO_RIO_ DisableConvChannel ()、
EPCIO_ADC_StopConv()

使用 EPCIO_ADC_GetWorkStatus()讀取目前 ADC 的工作狀態，確認轉換過程是否完成，確認完成後可使用 EPCIO_ADC_GetInput()讀取輸入電壓值。

7.3 特定電壓值觸發中斷服務函式

EPCIO Series 運動控制卡提供電壓比較值觸發中斷功能，使用者可以對選定的 ADC Channel 設定電壓比較值，當比較功能開啟並滿足觸發條件後，將自動呼叫使用者自訂的中斷服務函式。要使用特定電壓值觸發中斷必須完成下列

幾項設定步驟：

1. 定義與宣告使用者自訂的電壓比較值中斷服務函式，中斷服務函式必須遵循下面的定義：

```
typedef void(_stdcall *ADCISR)(ADC INT*);
```

使用者自訂的電壓比較值中斷服務函式可定義如下：

```
void _stdcall ADC_ISR_Function(ADCINT *pstINTSource)
{
    //   判斷是否因 ADC Channel 0 的電壓值滿足比較條件而觸發此函式
    if (pstINTSource->COMP0)
    {
        /*
        滿足 Channel 0 比較值條件時的處理程序
        */
    }
 }
```

在電壓比較值中斷服務函式中需使用 pstINTSource->COMP0，判斷此函式是否由電壓比較值中斷所觸發。此外中斷服務函式的宣告需加上關鍵字 _stdcall。

2. 串接使用者自訂的電壓比較值中斷服務函式。

在初始化運動控制卡時需串接電壓比較值中斷服務函式，以初始化 EPCIO-6000 為例，需將中斷服務函式的函式指標傳入 EPCIO6000_Init()，如下：

```
EPCIO6000_Init( NULL, NULL, NULL, NULL, NULL, NULL,
                ADC_ISR_Function, NULL, NULL, 0);
```

3. 使用 EPCIO_ADC_SetCompValue()設定電壓比較器之比較值。

4. 使用 EPCIO_ADC_SetCompMask()設定電壓比較器之遮蔽值，藉由遮蔽電壓值 1~3 個 bits 不做比較動作，以降低比較器之靈敏度，避免中斷持續觸發。此函式可設定的參數包括：

   ADC_MASK_NO　不使用電壓遮蔽位元

   ADC_MASK_BIT1　使用 1 個電壓遮蔽位元

   ADC_MASK_BIT2　使用 2 個電壓遮蔽位元

   ADC_MASK_BIT3　使用 3 個電壓遮蔽位元

5. 使用 EPCIO_ADC_SetCompType()設定 ADC Channel 電壓比較型式，當比較條件成立便會觸發 ADC 中斷訊號。此函式可設定的參數包括：

   ADC_COMP_RISE　　ADC 輸入電壓由小到大，並通過比較值

   ADC_COMP_FALL　　ADC 輸入電壓由大到小，並通過比較值

   ADC_COMP_LEVEL　　ADC 輸入電壓值改變，並通過比較值

6. 使用 EPCIO_ADC_EnableCompInt()開啟電壓比較值中斷功能。

   ➔ *See Also*　EPCIO_ADC_DisableCompInt()

下面程式碼說明如何使用 ADC 中斷功能：

```
void _stdcall ADC_ISR_Function(ADCINT *pstINTSource)
{
   //   判斷是否發生 ADC Channel 0 比較值條件成立
   if (pstINTSource->COMP0)
   {
```

```
        /*
        滿足 ADC Channel 0 比較值條件時的處理程序
        */
    }
}

if (EPCIO6000_Init( NULL, NULL, NULL, NULL, NULL
                    NULL, ADC_ISR_Function, NULL, NULL, 0))
{
     .
     .
    //  設定 ADC 串列傳輸時脈
    EPCIO_ADC_SetClockDivider(10, 0);

    //  設定轉換模式為連續轉換
    EPCIO_ADC_SetConvMode(ADC_MODE_FREE, 0);

    //  設定 ADC Channel 0 轉換型式為雙極性模式(-5 ~ 5V)
    EPCIO_ADC_SetConvType(0, ADC_TYPE_BIP, 0);

    //  開啟 ADC Channel 0 之電壓值轉換功能
    EPCIO_ADC_EnableConvChannel(0, 0);

    //  設定電壓比較器之比較值為 3.0V
    EPCIO_ADC_SetCompValue(0, 3.0, 0);

    //  設定電壓比較模式由高到低
    EPCIO_ADC_SetCompType(0, ADC_COMP_FALL, 0);

    //  開啟 ADC 比較器觸發中斷功能
```

```
    EPCIO_ADC_EnableCompInt(0, 0);

    //  開啟 ADC 電壓轉換功能
    EPCIO_ADC_StartConv(0);
     .
     .
}
```

7.4 電壓轉換完成觸發中斷服務函式

EPCIO Series 運動控制卡提供兩種“電壓轉換完成觸發中斷服務函式”功能，分別為任意一 ADC Channel 轉換完成觸發中斷，以及 ADC 標籤 Channel 轉換完成觸發中斷，說明如下：

1. 任意一 ADC Channel 轉換完成觸發中斷服務函式

   參考 7.3 節，使用者自訂的任意 ADC Channel 完成電壓轉換動作後觸發之中斷服務函式可定義如下：

```
void _stdcall ADC_ISR_Function(ADCINT *pstINTSource)
{
   //  判斷是否因任意 ADCChannel 完成電壓轉換而觸發此函式
   if (pstINTSource->CONV)
   {
      /*
      任意 ADC Channel 完成電壓轉換時的處理程序
      */
   }
}
```

使用 EPCIO_ADC_EnableConvInt()開啟任意 ADC Channel 轉換完成後觸發中斷產生功能。

➔ *See Also* EPCIO_ADC_DisableConvInt()

2. ADC 標籤 Channel 轉換完成觸發中斷服務函式

使用者自訂的 ADC 標籤 Channel 完成電壓轉換動作後觸發之中斷服務函式可定義如下：

```
void _stdcall ADC_ISR_Function(ADCINT *pstINTSource)
{
   //  判斷是否因 ADC 標籤 Channel 完成電壓轉換而觸發此函式
   if (pstINTSource->TAG)
   {
      /*
      標籤 Channel 完成電壓轉換時的處理程序
      */
   }
 }
```

使用 EPCIO_ADC_SetTagChannel()設定某一 ADC Channel 為 Tag Channel，並呼叫 EPCIO_ADC_EnableTagInt()啟用該 ADC Tag Channel 完成電壓轉換時觸發中斷服務函式之功能。

➔ *See Also* EPCIO_ADC_DisableTagInt()

8. 數位轉類比輸出控制(DAC)

EPCIO Series 運動控制卡提供類比電壓輸出(D/A Converter, DAC)，輸出電壓範圍為-10 ~ 10 V，當指定 Channel 已規劃作為 V Command 的運動軸後，則無法作為 DAC Channel 來使用。請注意：EPCIO-4005、EPCIO-6005 和 EPCIO-6005e 不支援此功能

8.1 基本設定與功能

使用 EPCIO_DAC_SetClockDivider()設定 DAC 串列介面的傳輸時脈，此函式的宣告如下：

```
BOOL EPCIO_DAC_SetClockDivider( WORD wDivider,
                                WORD wCardIndex);
```

傳輸時脈為 System Clock(40MHz)除以 4×(wDivider+1)，wDivider 的設定範圍為 0 ~ 255，預設值為 0。

使用 EPCIO_DAC_SetCmdSource()設定 DAC Channel 輸出之命令來源為軟體規劃或由硬體閉迴路(PCL)輸入，當設定為軟體規劃模式時可呼叫 EPCIO_DAC_SetOutput()設定 DAC Channel 的輸出電壓值，並使用 EPCIO_DAC_StartConv()啟動 DAC Channel 進行電壓輸出轉換。

➔ *See Also* EPCIO_DAC_GetOutput()、EPCIO_DAC_StopConv()

8.2 輸出電壓硬體觸發模式功能

當 DAC 規劃為軟體命令模式時，可針對選定的 DAC Channel 預先規劃 1 個輸出電壓值，並由特定的硬體觸發來源觸發輸出此預設電壓。此功能的設定步驟說明如下：

使用 EPCIO_DAC_SetTrigOutput()規劃滿足觸發條件後立即輸出的電壓值，並使用 EPCIO_DAC_SetTrigSource()設定觸發預先規劃電壓的中斷條件，此函式的宣告如下：

```
BOOL EPCIO_DAC_SetTrigSource( WORD wChannel,
                              DWORD dwTrigSource,
                              WORD wCardIndex);
```

參數 dwTrigSource 可設定為：

DAC_TRIG_ENC0	編碼器 Channel 0 特定計數值
DAC_TRIG_ENC1	編碼器 Channel 1 特定計數值
DAC_TRIG_ENC2	編碼器 Channel 2 特定計數值
DAC_TRIG_ENC3	編碼器 Channel 3 特定計數值
DAC_TRIG_ENC4	編碼器 Channel 4 特定計數值
DAC_TRIG_ENC5	編碼器 Channel 5 特定計數值
DAC_TRIG_ENC6	編碼器 Channel 6 特定計數值
DAC_TRIG_ENC7	編碼器 Channel 7 特定計數值
DAC_TRIG_ADC0	ADC Channel 0 特定電壓輸入值
DAC_TRIG_ADC1	ADC Channel 1 特定電壓輸入值
DAC_TRIG_ADC2	ADC Channel 2 特定電壓輸入值
DAC_TRIG_ADC3	ADC Channel 3 特定電壓輸入值
DAC_TRIG_ADC4	ADC Channel 4 特定電壓輸入值
DAC_TRIG_ADC5	ADC Channel 5 特定電壓輸入值
DAC_TRIG_ADC6	ADC Channel 6 特定電壓輸入值

DAC_TRIG_ADC7	ADC Channel 7 特定電壓輸入值
DAC_TRIG_LDI0	近端輸入接點 LDI 0 訊號輸入
DAC_TRIG_LDI1	近端輸入接點 LDI 1 訊號輸入
DAC_TRIG_LDI2	近端輸入接點 LDI 2 訊號輸入
DAC_TRIG_LDI3	近端輸入接點 LDI 3 訊號輸入
DAC_TRIG_DFI0	近端輸入接點 DFI 0 訊號輸入
DAC_TRIG_DFI1	近端輸入接點 DFI 1 訊號輸入
DAC_TRIG_DFI2	近端輸入接點 DFI 2 訊號輸入
DAC_TRIG_DFI3	近端輸入接點 DFI 3 訊號輸入
DAC_TRIG_R0DI0	遠端輸入接點 Set 0 的 DI 0 訊號輸入
DAC_TRIG_R0DI1	遠端輸入接點 Set 0 的 DI 1 訊號輸入
DAC_TRIG_R0DI2	遠端輸入接點 Set 0 的 DI 2 訊號輸入
DAC_TRIG_R0DI3	遠端輸入接點 Set 0 的 DI 3 訊號輸入
DAC_TRIG_R1DI0	遠端輸入接點 Set 1 的 DI 0 訊號輸入
DAC_TRIG_R1DI1	遠端輸入接點 Set 1 的 DI 1 訊號輸入
DAC_TRIG_R1DI2	遠端輸入接點 Set 1 的 DI 2 訊號輸入
DAC_TRIG_R1DI3	遠端輸入接點 Set 1 的 DI 3 訊號輸入

設定完成後須呼叫 EPCIO_DAC_EnableTrigMode()開啟中斷觸發模式。

➔ *See Also* EPCIO_DAC_DisableTrigMode()

9. 硬體位置閉迴路控制(PCL)

EPCIO Series 運動控制卡提供硬體位置閉迴路輸出控制(Position Closed_Loop，PCL)功能(EPCIO-4005/6005/6005e 無此功能)，可驅動速度型伺服馬達，利用驅動函式可設定閉迴路補償增益(Compensator Gain)值，經由 P(比例增益)控制法則可將位置誤差值進行計算後得出欲輸出之電壓值。

9.1 基本設定與位置閉迴路控制失效處理

欲開啟閉迴路命令誤差計數(Error Counter)功能，需先呼叫 EPCIO_PCL_EnableErrorCounter()，再呼叫 EPCIO_PCL_StartControl()啟動 PCL 控制功能。

➔ *See Also* EPCIO_ PCL_DisableErrorCounter()
EPCIO_PCL_StopControl()

使用 EPCIO_PCL_GetErrorCounter()可讀取各軸位置命令輸出與編碼器回授命令之誤差計數值，並使用 EPCIO_PCL_SetScaleGain()設定閉迴路控制軸之增益值，閉迴路增益值可藉由設定比例項(Kp1)及倍率項(Kp2)組成，增益值= Kp1 × Kp2 / 16。其中 Kp1 為 wPGain，Kp2 則為 nSGain，此函式的宣告如下：

```
BOOL EPCIO_PCL_SetScaleGain(WORD wChannel,
                            WORD wPGain,
                            int nSGain,
                            WORD wCardIndex);
```

各參數意義為：

wChannel	PCL Channel 編號：0 ~ 5
wPGain	比例增益(P Gain)：0 ~ 127
nSGain	倍率增益(S Gain)：-7 ~ 7

使用 EPCIO_PCL_EnableOverflowInt()開啟指定 Channel 的命令誤差計數值溢位時產生中斷觸發之功能。當位置命令與編碼器位置的誤差量超過誤差計數值所能容許範圍時，命令誤差計數值產生溢位中斷通知，並自動輸出 DAC 電壓值為 0 V。此外可使用 EPCIO_PCL_ClearErrorCounter()清除命令誤差計數值以及溢位狀態。

➔ *See Also* EPCIO_PCL_DisableOverflowInt()