IMP Series

運動控制函式庫

使用手冊

版本 ：V.2.01

日期 ：2013.01

http://www.epcio.com.tw

目 錄

1. 運動控制函式庫簡介

IMP Series 運動控制平台，內部已嵌入硬即時(Hard real time)作業系統 VxWorks 與運算處理器(PowerPC 440)，並已內含 **IMP Series 運動控制函式庫**（Motion Control Command Library, MCCL），供使用者分別在 A+ PC **模式**或單機模式(Standalone Mode)下呼叫使用。

A+ PC 模式為使用者藉由個人電腦開發、編譯及執行應用程式，應用程式中所有與運動相關之運動控制函式，將透過PCI Bus或Enthernet與IMP Series運動控制平台溝通，運動控制功能皆由IMP負責運算處理；人機介面及其它應用函式，則由個人電腦負責運算處理。

如欲應用在 A+ PC 模式，可使用在 WINDOWS 98se、WINDOWS NT/2000/XP 等作業平台，並支援 Visual C++、Visual Basic 與 Borland C++ Builder 等開發環境。

在單機模式下之應用，使用者可安裝 IMP Series 運動控制平台安裝光碟，即可使用智慧型運動控制平台整合開發工具 IDK(Integrated Development Kits)，開發單機模式之應用程式。詳細單機應用整合開發環境使用，請參考 IMP Series 單機(Standalone)模式應用使用手冊。

MCCL 提供 3D 空間中點對點、直線、圓弧、圓、螺線等運動的軌跡規劃函式；除此之外，MCCL 並提供了 14 種原點復歸模式、運動空跑、運動延遲、微動/吋動/連續吋動、運動暫停、繼續、棄置等操作函式。

在軌跡規劃功能方面可設定不同的加/減速時間、加/減速曲線型式、進給速度、最大進給速度與最大加速度；MCCL 也包含軟、硬體過行程保護、平滑運動、動態調整進給速度及錯誤訊息處理等功能。

在定位控制方面，使用者可利用 MCCL 設定定位比例增益、定位誤差容許範圍，MCCL 也提供定位確認、齒輪齒隙、間隙補償等功能。

在 I/O 接點訊號處理方面，使用者可利用 MCCL 讀取 Home 接點與 Limit Switch 接點的訊號，也可輸出 Servo-On/Off 訊號；另外某些特定 I/O 接點的輸入訊號可自動觸發中斷服務函式，使用者可自訂此

函式的執行內容。

在編碼器的功能方面，使用者可以即時讀取編碼器的計數值，並設定編碼器的訊號輸入倍率。某些特定輸入訊號可自動閘鎖編碼器計數值，MCCL 並支援編碼器之計數值累積到特定值時自動觸發使用者自訂函式的功能。

在 D/A 轉換功能方面，使用者除了可利用 MCCL 輸出要求的電壓值外(-10V ~ 10V)，並可預先規劃欲輸出的電壓值，並在滿足觸發條件後自動輸出此電壓值。

在 A/D 轉換功能方面，使用者可利用 MCCL 讀取輸入的電壓值(-5V ~ 5V 或 0V ~ 10V)，並可設定單一 Channel 電壓轉換與標籤 Channel 電壓轉換。而在完成電壓轉換的動作或電壓值滿足比較條件時，皆可自動觸發中斷服務函式，使用者可自訂此函式的執行內容。

在計時功能方面，使用者可設定計時器的計時時間，當啟動計時功能並在計時終了時，可自動觸發使用者自訂的中斷服務函式，並重新開始計時，此過程將持續至關閉此項功能為止。MCCL 也提供 Watch Dog 的功能。

使用 MCCL 並不需深入了解運動控制中複雜的軌跡規劃、定位控制、即時多工環境，利用此函式庫即可快速開發、整合系統。

2. MCCL 功能

2.1 軟體規格

■ A^+ PC 模式作業系統環境

✓ WINDOWS XP/XP Embedded

✓ WINDOWS 7

■ A^+ PC 模式開發環境

✓ Borland C++ Builder (BCB)

✓ Visual C++ (VC++)

✓ Visual Basic (VB)

✓ Visual C# (VC#)

■ 單機模式開發環境

✓IDK (隨卡提供)

✓WindRiver WorkBench (使用者自行購置)

✓Green Hill – Multi (使用者自行購置)

■ A^+ PC 模式使用 MCCL 時，專案所需要的附加檔案

	檔案名稱
VC++	MCCL.h MCCL_Fun.h MCCLPCI_IMP.lib
VB	MCCLPCI_IMP.bas
VC#	MCCL.cs

■ 單機模式使用 MCCL 時，專案所需要的附加檔案

	檔案名稱
IDK	MCCL.h MCCL_Fun.h

2.2 運動軸定義與可搭配控制平台數目

2.2.1 運動軸定義

MCCL 的設計目的是針對三軸直角正交(X-Y-Z)，外加五軸輔助軸(U、V、W、A、B)的運動平台，提供運動控制的功能。如 Figure 2.2.1 所示，U、V、W、A、B 為五個輔助軸，代表五個獨立軸向。

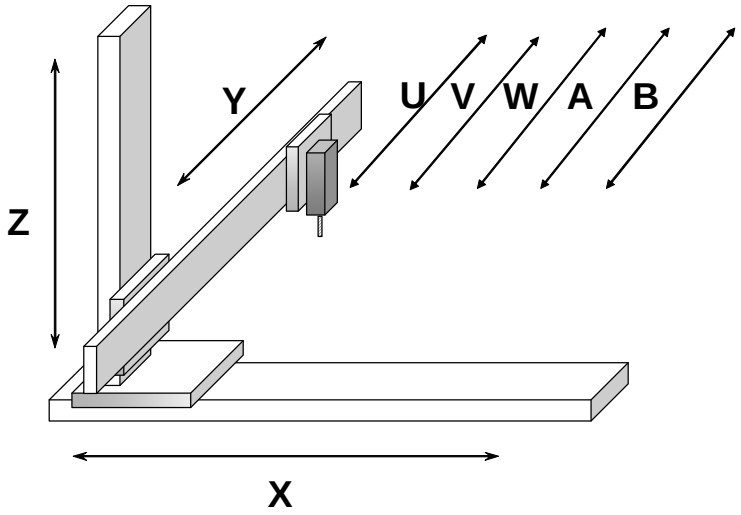


Figure 2.2.1 三軸直角正交(**X-Y-Z**)，外加五軸輔助軸(**U、V、W、A、**B)

MCCL 提供最大的同動控制軸數為 8 軸，使用者可以使用一張 IMP Series 運動控制平台執行 1 至 8 軸同動或不同動控制。使用者給定的運動命令可選用絕對、增量座標值，不論使用者選用何種座標值，本函式庫皆會記錄運動位置的絕對座標值(相對於原點)。

2.2.2 可搭配控制平台數目

每張運動控制平台依型號不同最多可控制八組系統(含馬達及驅動器)，而 MCCL 最多可同時控制 6 張運動控制平台，因此，最多可

同時控制 48 軸。IMP Series 運動控制平台可送出速度命令（V Command）或是脈衝命令(Pulse Command)，基本的組構如 Figure 2.2.2。

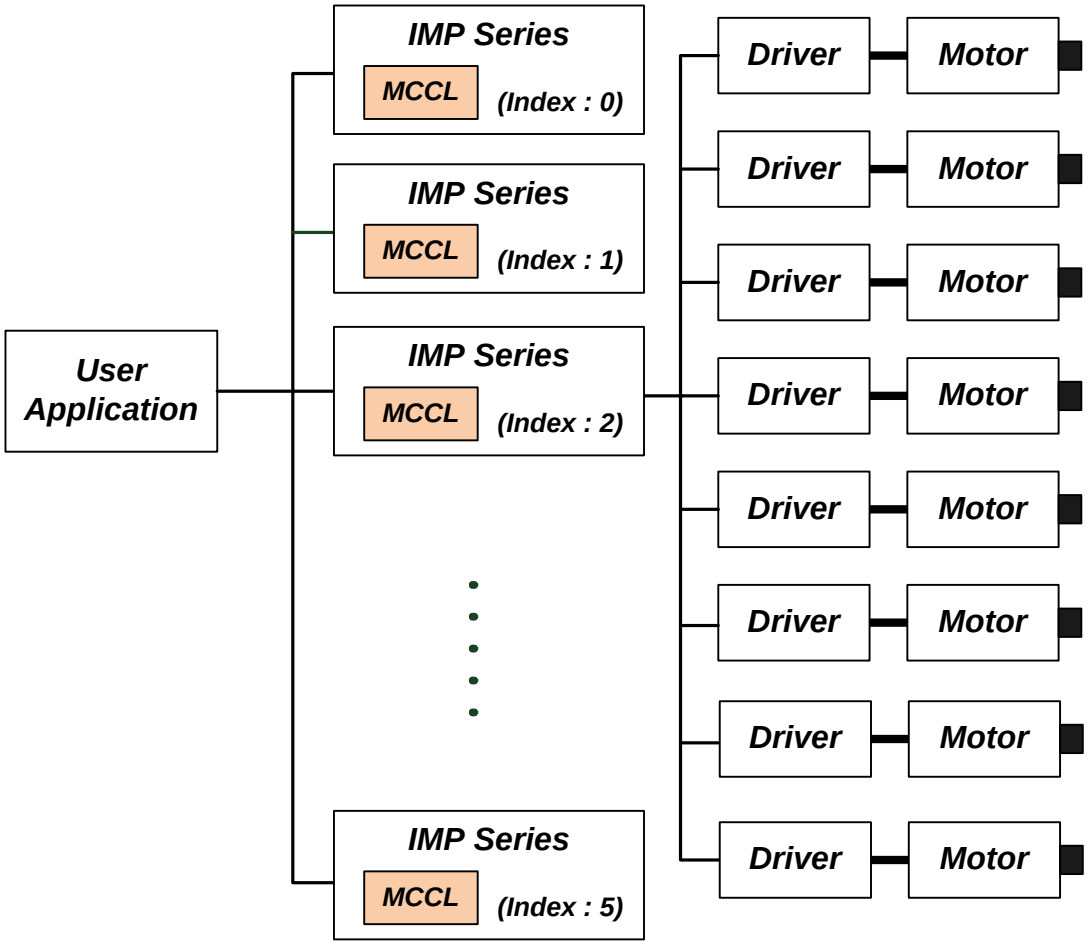


Figure 2.2.2 MCCL 可搭配 6 張 IMP 平台

2.3 函式庫操作特性

呼叫 MCCL 中的運動函式後會將相關的運動命令先存放(Put)在各 Group 專屬的運動命令緩衝區(Motion Command Queue)中，而***非立即執行***(有關 Group 的說明請參考"**2.5.1 啟動運動控制函式庫**")。然後 MCCL 會使用先入先出(First In First Out, FIFO)的方式，從緩衝區中抓取(Get)運動命令進行解譯及粗插值運算 (參考 Figure 2.3.1)。但這兩個行為並非順序與同步動作，也就是說並不需要等到運動命令執行完成，即可將新的運動命令送到運動命令緩衝區中。

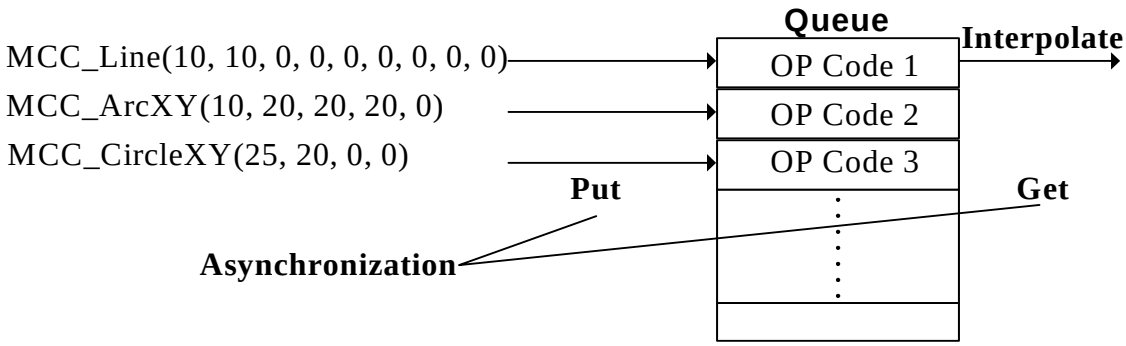


Figure 2.3.1 運動命令緩衝區

各 Group 之運動命令緩衝區預設可儲存 10000 筆命令，但可利用 MCC_SetCmdQueueSize 改變其大小，而 MCC_GetCmdQueueSize 可讀回目前緩衝區大小；注意，***改變緩衝區大小的動作必須在無庫存命令時始可執行***。

下面列出會增加運動命令庫存量之函式名稱：

呼叫這些函式，MCCL 會將命令先存放(Put)在運動命令緩衝區中，而後會在適當時機取出緩衝區中第一筆命令（並移除），然後執行對應動作：

A. 直線運動函式

MCC_Line()

B. 圓弧運動函式

MCC_ArcXYZ()　　MCC_ArcXYZ_Aux()
MCC_ArcXY()　　MCC_ArcXY_Aux ()
MCC_ArcYZ()　　MCC_ArcYZ_Aux ()
MCC_ArcZX()　　MCC_ArcZX_Aux ()
MCC_ArcThetaXY()　　MCC_ArcThetaYZ()
MCC_ArcThetaZX()

C. 圓運動函式

MCC_CircleXY()　　MCC_CircleYZ()
MCC_CircleZX()
MCC_CircleXY_Aux ()　　MCC_CircleYZ_Aux ()
MCC_CircleZX_Aux ()

D. 螺線運動函式

MCC_HelicaXY_Z()　　MCC_HelicaYZ_X()
MCC_HelicaZX_Y()
MCC_HelicaXY_Z_Aux ()　　MCC_HelicaYZ_X_Aux ()
MCC_HelicaZX_Y_Aux ()

E. 點對點運動函式

MCC_PtP()　　MCC_PtPX()　　MCC_PtPY()
MCC_PtPZ()　　MCC_PtPU()　　MCC_PtPV()
MCC_PtPW()　　MCC_PtPA()　　MCC_PtPB()

F. 吋動與連續吋動函式

MCC_JogSpace()　　MCC_JogConti()　　MCC_JogPulse()

G. 定位確認函式

MCC_EnableInPos()　　MCC_DisableInPos()

H. 平滑運動函式

MCC_EnableBlend()　　MCC_DisableBlend()

MCC_CheckBlend()

I. 運動延遲函式

MCC_DelayMotion()

若運動命令緩衝區已滿，則使用上述函式的傳回值將為 COMMAND_BUFFER_FULL_ERR，表示此筆運動命令不被接受。Figure 2.3.1 顯示對 Group 0 運動命令緩衝區的操作過程，可看出屬於同一個 Group 的運動命令將被依序執行。因為各個 Group 擁有專屬的運動命令緩衝區，因此可同時執行屬於不同 Group 的運動命令。

注意

若今有一需求：使 Group 0 之 X 軸移動到座標 10 的位置後，輸出一 servo-on 訊號，再將此軸移動到座標 20 的位置。程式可能的寫法為：

```
MCC_Line(10, 0, 0, 0, 0, 0, 0, 0, 0);
MCC_SetServoOn(1, 0);
MCC_Line(20, 0, 0, 0, 0, 0, 0, 0, 0);
```

則在 MCC_Line()被置入運動命令緩衝區後(尚未真正執行)，將立即執行 MCC_SetServoOn()，因 MCC_SetServoOn()並不被置於運動命令緩衝區中而是直接執行，因此在實際位置到達座標 10 之前，servo-on 訊號將早已送出，此項操作特性需特別注意。

如要求 X 軸移動到座標 10 的位置後才執行訊號輸出，則使用者需進行額外的判斷，也就是需自行檢查系統的運動狀態或目前座標，來控制訊號的輸出動作，下面為一個簡單的使用範例：

```
//  假使要求第 0 個 Group 中的 X 軸移動到座標 10 的位置，才輸出 servo-on
//  訊號
MCC_Line(10, 0, 0, 0, 0, 0, 0, 0, 0);

while( MCC_GetMotionStatus(0) != GMS_STOP )
//  MCC_GetMotionStatus() 的函式傳回值等於 GMS_STOP 表示目前全部的運動命令皆已執行完成

MCC_SetServoOn(1, 0);
MCC_Line(20, 0, 0, 0, 0, 0, 0, 0, 0);
```

2.4 機構、編碼器、原點復歸參數設定

2.4.1 機構參數

MCCL 利用機構參數來定義使用者的機構平台特性與驅動器使用型式；並利用機構參數規劃相對於邏輯原點的座標系統、座標系統邊界值及各軸的最大安全進給速度。

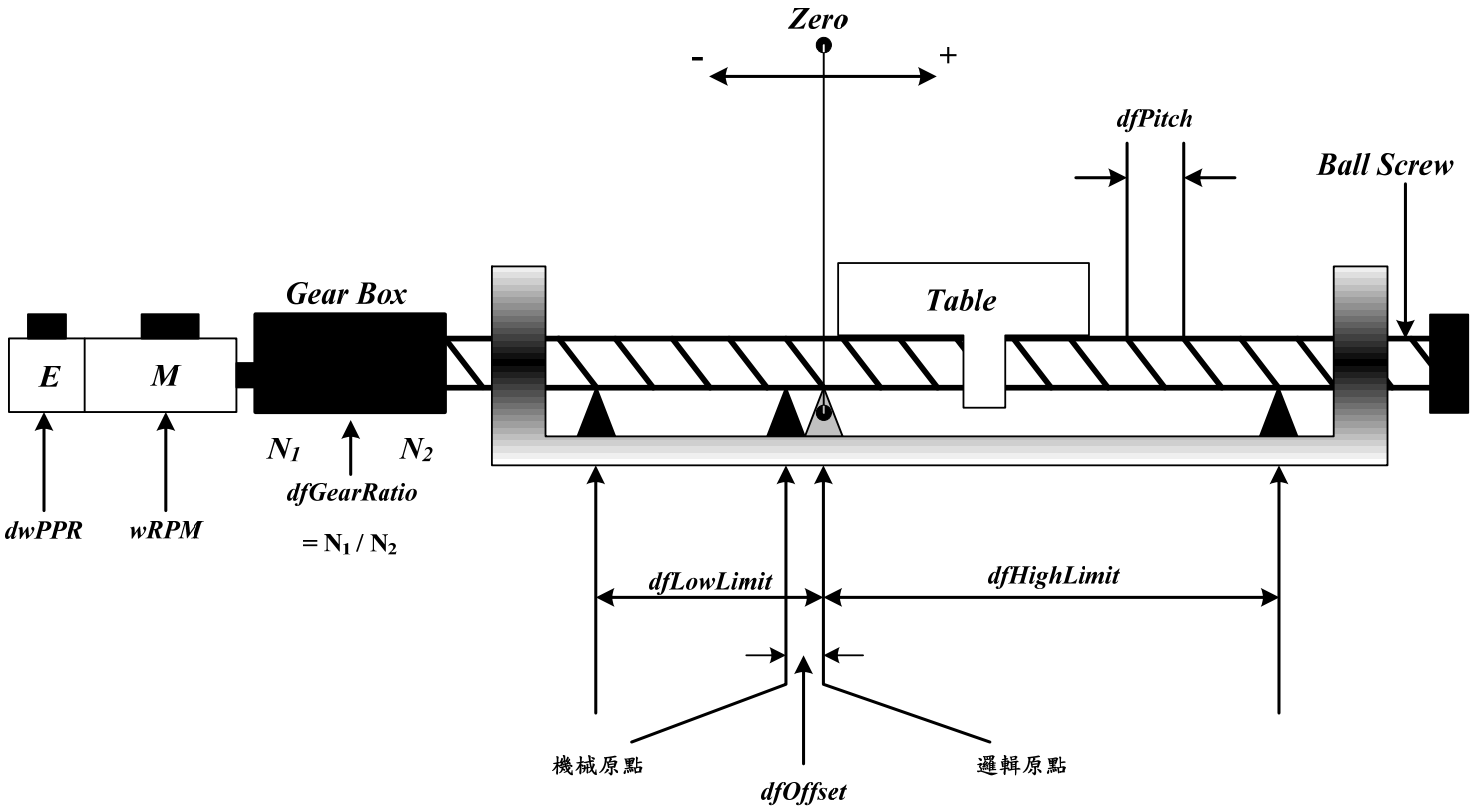


Figure 2.4.1 機構平台特性

以下為機構參數的內容與詳細說明：

```
typedef struct _SYS_MAC_PARAM
{
    WORD        wPosToEncoderDir;
    WORD        wRPM;
    DWORD       dwPPR;
```

```
    double      dfPitch;
    double      dfGearRatio;
    double      dfHighLimit;
    double      dfLowLimit;
    double      dfHighLimitOffset;
    double      dfLowLimitOffset;
    WORD        wPulseMode;
    WORD        wPulseWidth;
    WORD        wCommandMode;
    WORD        wPaddle;
    WORD        wOverTravelUpSensorMode;
    WORD        wOverTravelDownSensorMode;
} SYS_MAC_PARAM;
```

wPosToEncoderDir：方向調整參數

0　　輸出命令不反向

1　　輸出命令反向

此項參數用來修正當運動命令方向與期望的機構運動方向不同時的現象，即若送出正向運動命令例如使用 MCC_JogSpace(10, 10, 0, 0)，但因馬達配線的因素，使機構實際上往使用者定義的負方向移動，此時可設定此項參數為"1"，使運動命令方向與期望的機構運動方向保持一致。(不需更改馬達配線)

wRPM：馬達最大安全轉速

各軸馬達最大安全轉速。**各軸進行點對點快速移動時**，由所設定的速度換算而得的各軸馬達轉速將不會超過 *wRPM* 設定值。

➔ *See Also*　　MCC_SetPtPSpeed()

dwPPR：馬達軸心每旋轉一圈，編碼器所增加的計數值，或每旋轉一圈所需的 pulse 數。

如使用閉迴路控制，此值為馬達軸心每旋轉一圈，編碼器所增加的計

數值；但如為開迴路系統，則此值為馬達軸心每旋轉一圈所需的 pulse 數。

使用線性馬達時 *dfPitch* 與 *dfGearRatio* 皆應設定為 1。另外，線性馬達並無 *dwPPR* 相關的定義，且要求移動的距離通常是以 pulse 為單位，此時可將 *dwPPR* 設定為 1，如此可更改 MCCL 所使用的單位為 pulse。例如，當要求 X 軸移動 1000 pulses 時，可呼叫 MCC_Line(1000, 0,0,0,0,0,0,0,0) ， X 軸將輸出 1000 pulses ； 當使用 MCC_SetFeedSpeed(500)時，表示要求線性馬達的速度為每秒移動 500 pulses。

dfPitch：導螺桿間隙值

導螺桿每旋轉一圈，table 所移動的距離，**單位為 UU(其單位為使用者自訂之單位)。**如無配置導螺桿則此值應設定為 1。

dfGearRatio：齒輪箱減速比

連接馬達軸心與導螺桿之齒輪箱之雙向齒輪比，該值可用齒輪之齒數計算得知，或是簡單定義為「導螺桿每轉一圈，馬達所轉動的圈數」。如無配置齒輪箱則此值應設定為 1。

dfHighLimit：正方向過行程軟體邊界(或稱正方向邊界)

該值為正方向相對於邏輯原點所允許的最大位移量，單位為 UU。

➔ *See Also* MCC_SetOverTravelCheck()

dfLowLimit：負方向過行程軟體邊界(或稱負方向邊界)

該值為負方向相對於邏輯原點所允許的最大位移量，通常給定負值，單位為 UU。

dfHighLimitOffset：

保留欄位，使用者需設定為 0。

dfLowLimitOffset：

保留欄位，使用者需設定為 0。

wPulseMode：脈衝輸出格式

DDA_FMT_PD	Pulse/Direction
DDA_FMT_CW	CW/CCW
DDA_FMT_AB	A/B phase

wPulseWidth：脈衝輸出寬度(IMP 無作用)

wCommandMode：運動命令輸出型式

OCM_PULSE	脈衝命令(Pulse Command)
OCM_VOLTAGE	電壓命令(Voltage Command)

注意：僅在此值為 OCM_PULSE 時，***wPulseMode*** 與 ***wPulseWidth*** 才有意義。

wPaddle

保留欄位，使用者需設定為 0。

wOverTravelUpSensorMode：正極限開關(Limit Switch +)之配線方式，請參考下文有關如何檢查配線方式是否正確的說明。

SL_NORMAL_OPEN	Active High
SL_NORMAL_CLOSE	Active Low
SL_UNUSED	不檢查是否已碰觸極限開關。指定軸如未安裝極限開關可使用此選項

wOverTravelDownSensorMode：負極限開關(Limit Switch -)之配線方式，請參考下文有關如何檢查配線方式是否正確的說明。

SL_NORMAL_OPEN	Active High
SL_NORMAL_CLOSE	Active Low
SL_UNUSED	不檢查是否已碰觸極限開關。指定軸如

未安裝極限開關可使用此選項

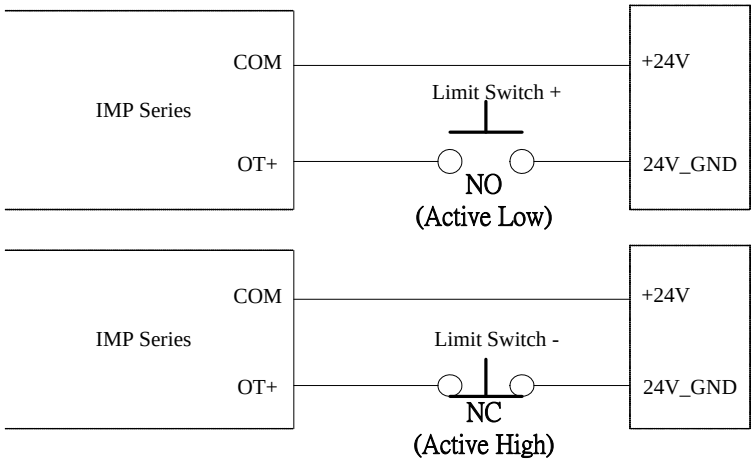


Figure 2.4.2 極限開關的配線方式

要使用極限開關功能需依據極限開關的配線方式(如 Figure 2.4.2)正確設定 *wOverTravelUpSensorMode* 與 *wOverTravelDownSensorMode*。可使用 MCC_GetLimitSwitchStatus()檢查配線方式的設定值是否正確。在未碰觸極限開關時，如果使用 MCC_GetLimitSwitchStatus()所獲得的極限開關為 Active 狀態，則表示配線方式設定值錯誤，應更改 *wOverTravelUpSensorMode* 或 *wOverTravelDownSensorMode* 的設定值。

但要使極限開關正常運作，除了必須正確設定極限開關的配線方式外，尚必須呼叫 MCC_EnableLimitSwitchCheck()，如此 *wOverTravelUpSensorMode* 與 *wOverTravelDownSensorMode* 的設定才能生效。

但 *wOverTravelUpSensorMode* 與 *wOverTravelDownSensorMode* 如設定為 SL_UNUSED，則呼叫 MCC_EnableLimitSwitchCheck()並無任何意義。

當功能開啟時，在碰觸到該軸運動方向的極限開關時(例如往正方向移動且觸到正向極限開關，或往負方向移動且碰觸到負向極限開關)，將會停止輸出 Group 的運動命令(並產生錯誤記錄)。

MCC_EnableLimitSwitchCheck()通常會與 MCC_GetErrorCode()搭配使用，利用不斷呼叫 MCC_GetErrorCode ()可獲知系統是否因碰觸到極限開關而產生錯誤記錄(代碼 0xF701~0xF708 分別代表 X~B 軸碰觸極限開關)；當發現碰觸極限開關之錯誤時，一般作法可能是：在螢幕上顯示訊息告知操作員，然後在程式中呼叫 MCC_ClearError()清除記錯，則此時系統可再往反方向退開極限開關。

在確定機構參數中各項欄位的內容後，可以使用 MCC_SetMacParam()設定機構參數，下面為使用範例：

```
SYS_MAC_PARAM     stAxisParam;
memset(&stAxisParam, 0, sizeof(SYS_MAC_PARAM)); // clear content to zero

stAxisParam.wPosToEncoderDir                = 0;
stAxisParam.dwPPR                           = 500;
stAxisParam.wRPM                            = 3000;
stAxisParam.dfPitch                         = 1.0;
stAxisParam.dfGearRatio                     = 1.0;
stAxisParam.dfHighLimit                     = 50000.0;
stAxisParam.dfLowLimit                      = -50000.0;
stAxisParam.wPulseMode                      = DDA_FMT_PD;
stAxisParam.wPulseWidth                     = 100; //任意值
stAxisParam.wCommandMode                    = OCM_PULSE;
stAxisParam.wOverTravelUpSensorMode         = SL_UNUSED; // not check
stAxisParam.wOverTravelDownSensorMode       = SL_UNUSED; // not check

MCC_SetMacParam(&stAxisParam, 0, 0); // 設定第 0 張平台的第 0 軸
```

一般需在使用 MCC_InitSystem()前將機構參數設定完成，各軸的

機構參數需分開設定。

➔ *See Also* MCC_GetMacParam()

2.4.2 編碼器參數

MCCL 利用編碼器參數來定義編碼器的使用特性，這些特性包括編碼器的訊號輸入格式、輸入訊號相位是否轉換與回授倍率(×1、×2、×4)等。以下為編碼器參數的內容與詳細說明：

```
typedef struct _SYS_ENCODER_CONFIG
{
    WORD    wType;
    WORD    wAInverse;
    WORD    wBInverse;
    WORD    wCInverse;
    WORD    wABSwap;
   WORD     wInputRate;
    WORD    wPaddle[2];
} SYS_ENCODER_CONFIG;
```

wType：輸入格式設定

ENC_TYPE_AB	A/B Phase
ENC_TYPE_CW	CW/CCW
ENC_TYPE_PD	Pulse / Direction

wAInverse：Phase A 訊號是否反相

0	不反相
1	反相

wBInverse：Phase B 訊號是否反相

0	不反相

1　　反相

wCInverse：Phase C (Phase Z) 訊號是否反相

0　　不反相

1　　反相

wABSwap：Phase A/B 訊號是否交換

0　　不交換

1　　交換

wInputRate：設定編碼器的回授倍率

1　　1 倍回授倍率(×1)

2　　2 倍回授倍率(×2)

4　　4 倍回授倍率(×4)

paddle：保留欄位，使用者需設定為 0

在確定編碼器參數中各項欄位的內容後，可以使用 MCC_SetEncoderConfig()設定編碼器參數，下面為使用範例：

```
SYS_ENCODER_CONFIG    stENCConfig;
memset(&stENCConfig, 0, sizeof(SYS_ENCODER_CONFIG));

stENCConfig.wType          = ENC_TYPE_AB;
stENCConfig.wAInverse      = 0; // not inverse
stENCConfig.wBInverse      = 0; // not inverse
stENCConfig.wCInverse      = 0; // not inverse
stENCConfig.wABSwap        = 0; // not swap
stENCConfig.wInputRate     = 4; // set encoder input rate: x4

MCC_SetEncoderConfig(&stENCConfig, 0, 0); // 設定第 0 張平台的第
```

0 軸

在使用 MCC_InitSystem()前須先設定編碼器參數，各軸的編碼器參數需分開設定。

注意

若在呼叫過 MCC_InitSystem()後再去改變機構或編碼器參數，則需另外呼叫 MCC_UpdateParam()，系統才能反應新的設定值。但須注意：***使用 MCC_UpdateParam()的結果與使用 MCC_ResetMotion()的結果相似，系統將回復到呼叫 MCC_InitSystem()後的初始狀態***。

2.4.3 原點復歸參數

MCCL 利用原點復歸參數來定義原點復歸動作，包括使用模式、原點復歸運動方向、原點開關(Home Sensor)的配線方式、編碼器 INDEX 訊號計數次數、加減速度設定等，更詳細的說明請參閱”**2.8 原點復歸**”。

原點復歸參數的內容說明如下：

```
typedef struct _SYS_HOME_CONFIG
{
    WORD        wMode;
    WORD        wDirection;
     WORD       wSensorMode;
     WORD       wPaddel0;
     int        nIndexCount;
     int        nPaddel1;
     double     dfAccTime;
     double     dfDecTime;
     double     dfHighSpeed;
     double     dfLowSpeed;
     double     dfOffset;
```

} *SYS_HOME_CONFIG*;

wMode：原點復歸模式

定義原點復歸使用模式，此項參數值必須大於等於 3 且小於等於 16，各模式的詳細說明請參閱後面與原點復歸有關的章節。

wDirection:原點復歸運動的起始方向

0　　正方向

1　　負方向

wSensorMode：原點開關(Home Sensor)的配線方式

SL_NORMAL_OPEN　　Active High

SL_NORMAL_CLOSE　　Active Low

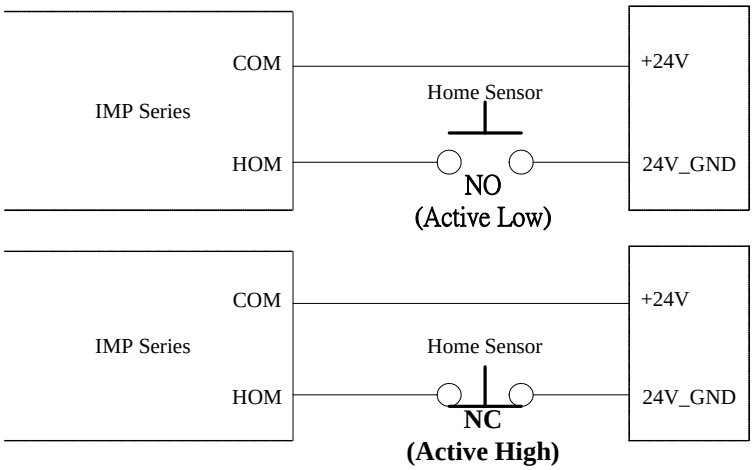


Figure 2.4.3 原點開關的配線方式

要使用原點復歸功能需依據原點開關的配線方式(如 Figure 2.4.3)正確設定 *wSensorMode*。可使用 MCC_GetHomeSensorStatus()檢查配線方式的設定值是否正確。在未碰觸原點開關時，如果使用 MCC_GetHomeSensorStatus()所獲得的原點開關為 Active 狀態，則表示配線方式設定值錯誤，應更改 *wSensorMode* 的設定值。

nIndexCount：指定編碼器 INDEX 訊號的編號

在原點復歸運動的過程中之 PHASE 2(尋找所指定編號的 INDEX)，第一個發生的 INDEX 訊號編號為 0，第二個發生的 INDEX 訊號編號為 1，往後依此類推。某些原點復歸模式需指定編碼器 INDEX 訊號的編號，當符合此設定值的 INDEX 訊號被觸發後，才能完成全部的原點復歸運動。

dfAccTime：進行原點復歸運動時，加速到 *dfHighSpeed* 或 *dfLowSpeed* 所需的時間，單位為 ms。

dfDecTime：進行原點復歸運動時，由速度 *dfHighSpeed* 或 *dfLowSpeed* 減速到停止所需的時間，單位為 ms。

dfHighSpeed：高速速度設定值，單位為 UU/sec。

通常是指原點復歸運動時第一階段所使用的速度

dfLowSpeed：低速速度設定值，單位為 UU/sec。

通常是指在完成原點復歸運動最後階段所使用的速度

dfOffset：邏輯原點位置偏移量，單位為 UU

一般而言機械原點與邏輯原點之間的偏移量會在校機的過程中找到。確認此一偏移量的方式是先將 *dfOffset* 設定為 0，並在完成原點復歸動作後(此時機台停留在「機械原點」位置)，再以 JOG 帶動方式找出與「邏輯原點」之間的偏移量，最後以此偏移量設定 *dfOffset*。再次執行原點復歸動作後，該運動軸將會移動至「邏輯原點」的位置，並且系統將以此點為運動命令的參考原點。

在確定原點復歸參數各項欄位的內容後，可以使用 MCC_SetHomeConfig()設定之，下面為使用範例：

```
SYS_HOME_CONFIG       stHomeConfig;
```

```
memset(&stHomeConfig, 0, sizeof(SYS_HOME_CONFIG));

stHomeConfig.wMode          = 3;     //  使用模式 3
stHomeConfig.wDirection     = 1;     //  往負方向進行復歸動作
stHomeConfig.wSensorMode    = 0;     //  使用 Active High 配線方式
stHomeConfig.nIndexCount    = 2;     //  INDEX 編號為 2
stHomeConfig.dfAccTime      = 300;   //  加速所需的時間，單位 ms
stHomeConfig.dfDecTime      = 300;   //  減速所需的時間，單位 ms
stHomeConfig.dfHighSpeed    = 10;    //  單位 UU/sec
stHomeConfig.dfLowSpeed     = 2;     //  單位 UU/sec
stHomeConfig.dfOffset       = 0;

MCC_SetHomeConfig(&stHomeConfig, 0, 0);//  設定第 0 張平台的第 0 軸
```

原點復歸參數須在執行原點復歸動作前設定，各軸的原點復歸參數需分開設定。

2.4.4 設定 Group(運動群組)參數

在使用 MCCL 前需先建立所需的 Group(運動群組)。Group 可視為一獨立之運動系統，當此系統運動時，其內部各運動軸通常存在相依的關係，明顯的例子為 X-Y-Z Table。

MCCL 使用 Group 的操作概念，所提供的運動控制函式大部分是以 Group 為單位來操作。每個 Group 皆包含了 X、Y、Z、U、V、W、A、B 等八個運動軸，但各運動軸並不一定須要實際對應到 IMP Series

運動控制平台上的輸出 Channel。MCCL 能同時控制 6 張 IMP Series 運動控制平台，而每張 IMP Series 運動控制平台最多可定義為 8 個 Group，因此最多能同時使用 48 個 Group，各 Group 間相互獨立，並不影響彼此間的運作。

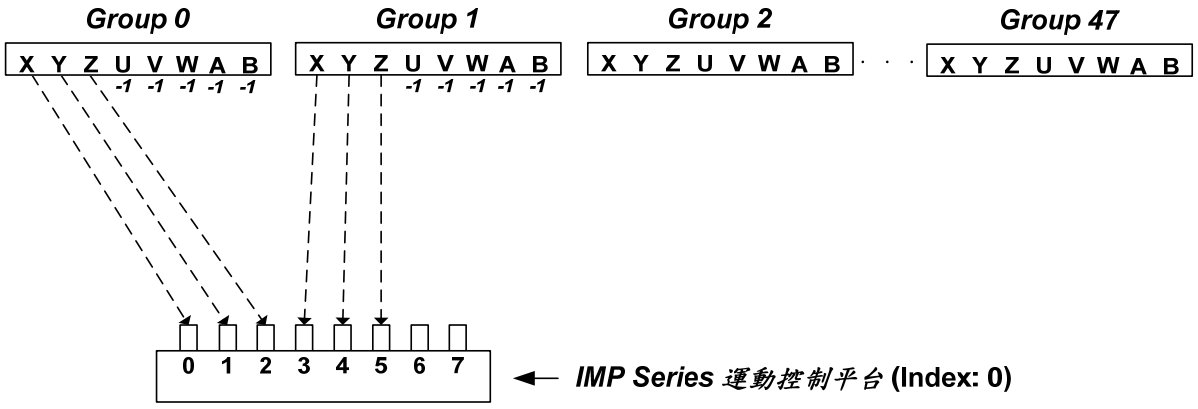


Figure 2.4.4 Group 參數設定

以 Figure 2.4.4 為例，目前使用了兩個 Group 與一張 IMP Series 運動控制平台，其中 Group(0)的 X、Y、Z 軸軌跡規劃的結果將分別從第 0 張平台的 0、1、2 之實體 Channel 輸出，並忽略 U、V、W、A、B 軸軌跡規劃的結果；而 Group(1)的 X、Y、Z 軸軌跡規劃的結果將分別從第 0 張平台的 3、4、5 個 Channel 輸出，並忽略 U、V、W、A、B 軸軌跡規劃的結果。

此時程式的寫法可能如下：

```
int    nGroup0, nGroup1;

MCC_CloseAllGroups(); //  關閉全部 Group
nGroup0 = MCC_CreateGroup(    0,    //  X  對應到實體輸出 Channel 0
                              1,    //  Y  對應到實體輸出
```

```
Channel 1
                                    2,  // Z 對應到實體輸出
Channel 2
                                    -1, // U 不作實際對應
                                    -1, // V 不作實際對應
                                    -1, // W 不作實際對應
                                    -1, // A 不作實際對應
                                    -1, // B 不作實際對應
                                    0); // 對應到第 0 張平台

    nGroup1 = MCC_CreateGroup(      3,  // X 對應到實體輸出
Channel 3
                                    4,  // Y 對應到實體輸出
Channel 4
                                    5,  // Z 對應到實體輸出
Channel 5
                                    -1, // U 不作實際對應
                                    -1, // V 不作實際對應
                                    -1, // W 不作實際對應
                                    -1, // A 不作實際對應
                                    -1, // B 不作實際對應
                                    0); // 對應到第 0 張平台
```

MCC_CreateGroup()之回傳值代表了新建立之 Group 的編號(0~47)，此編號將會在往後呼叫運動函式時用到。例：若要使 Group(1)的 X、Y、Z 軸移動到座標位置 10，在程式中就須寫成 MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, ***nGroup1***)；此時第 0 張 IMP 平台的第 3、4、5 個 Channel 將負責輸出此 Group 的 X、Y、Z 軸插值的結果。再以下例說明：

```
MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, nGroup0);    // Command 0
MCC_Line(20, 20, 20, 0, 0, 0, 0, 0, nGroup0);    // Command 1
MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, nGroup1);    // Command 2
MCC_Line(20, 20, 20, 0, 0, 0, 0, 0, nGroup1);    // Command 3
```

使用上述的 Group 設定內容，Group(0)將執行 Command 0，並從第 0 張平台的第 0、1、2 個 Channel 輸出 X、Y、Z 軸軌跡規劃的結果。Group(0)在完成 Command 0 後，才會再執行同屬於相同 Group 之 command 1。

因各 Group 間獨自運作，故 Group(1)不需等待 Group(0)完成 Command 0，將直接執行 Command 2，並從第 0 張平台的第 3、4、5 個 Channel 輸出 X、Y、Z 軸軌跡規劃的結果。而 Group(1)完成 Command 2 後，會再執行同屬於相同 Group 之 Command 3。

在啟動 MCCL 前如未建立任何 Group，則 MCCL 會使用預設值運作，預設為僅開啟 Index 為 0 之 Group，且其 X、Y、Z、U、V、W、A、B 運動軸分別對應到第 0 張平台的 Channel 0 ~ 7 輸出。

資訊

1. Group 與 Group 間互相**不**影響。
2. Group 皆包含了 X、Y、Z、U、V、W、A、B 等八個運動軸，各運動軸可規劃是否對應至實體 Channel 輸出，但 **Group 中須至少有一運動軸作實際對應；且不允許有兩個以上的運動軸對應至同一實體 Channel**。
3. 為降低 MCCL 對 CPU 的使用率，**所使用的 Group 個數應愈少愈好**。

2.5 啟動與結束運動控制函式庫

2.5.1 啟動運動控制函式庫

在開始使用 MCCL 時需先設定下面幾項參數的內容，包括：

a. 設定機構參數　　使用 MCC_SetMacParam()
b. 設定編碼器參數　　使用 MCC_SetEncoderConfig()
c. 設定 Group 參數　　使用 MCC_CreateGroup() / MCC_CloseAllGroups()

未完成這幾項參數的內容設定，或是在進行這幾項步驟時產生錯誤，不得再使用 MCCL 中的其他函式。機構、編碼器與 **Group(**運動群組)參數的設定請參閱前面的說明與參考**"IMP Series 運動控制函式庫範例手冊"**，下面僅說明如何啟動 MCCL。

I. 啟動 MCCL

使用 MCC_InitSystem()啟動 MCCL，MCC_InitSystem()的函式宣告如下：

```
int MCC_InitSystem(   int                 nInterpolateTime,
                      SYS_CARD_CONFIG     *pstCardConfig,
                      WORD                wCardNo);
```

nInterpolateTime 為插值時間(請參考後面章節的說明)，單位為 ms，可設定範圍為 1 ms ~ 50 ms，建議值為 2ms。較小的插值時間可縮短兩個插值點間的距離，但會增加 CPU 的工作負荷。下面為插值時間的設定參考，這些建議值並非絕對，可依照實際需要加以調整。

系　　統　　特　　徵	插 值 時 間 建 議 值
只需直線運動的系統	5 ms ~ 10 ms
一般包含圓弧運動的系統	5 ms
對圓弧運動軌跡有真圓度要求的系統	1ms ~ 3ms

pstCardConfig 為前一個步驟所設定的 IMP Series 運動控制平台硬體參數，*wCardNo* 為此時所使用 IMP Series 運動控制平台的張數。

2.5.2 結束運動控制函式庫

如要結束 MCCL 只需呼叫 MCC_CloseSystem()函式即可。

2.6 運動控制

2.6.1 座標系統

座標系統包括下列功能：

I. 選擇絕對或增量座標系統。

➔ *See Also* MCC_SetAbsolute()
MCC_SetIncrease()
MCC_GetCoordType()

II. 讀取目前位置座標值

➔ *See Also* MCC_GetCurPos() MCC_GetCurRefPos()
MCC_GetPulsePos()

III. 開啟/關閉軟體過行程檢查功能

使用 MCC_SetOverTravelCheck()開啟此項功能後，MCCL 在計算完每一個插值點時，會檢查此插值點是否會超出各軸的有效工作區間；若判斷已超出工作區間時，則不再對運動控制平台送出命令。使用者可使用 MCC_GetErrorCode()查詢訊息代碼(訊息代碼的意義請參考 IMP Series 運動控制函式庫參考手冊)，獲知是否已超出各軸的有效工作區間。

➔ *See Also* MCC_GetOverTravelCheck()
MCC_GetErrorCode()

IV. 開啟/關閉硬體極限開關檢查功能

此項功能請參考”**2.4.1 機構參數**”此章節的說明。

➔ *See Also* MCC_EnableLimitSwitchCheck()
MCC_DisableLimitSwitchCheck()
MCC_GetLimitSwitchStatus()

2.6.2 基本軌跡規劃

MCCL 提供直線、圓弧、圓、螺線等運動(統稱為一般運動)與點對點運動軌跡規劃的功能，在使用這些功能前應先針對機構特性及特殊需求設定加減速形式(S 或梯形)、加減速時間與進給速度。

I. 一般運動 (直線、圓弧、圓、螺線運動)

一般運動包括直線、圓弧、圓、螺線等多軸同動運動，在使用一般運動的函式時，通常會檢查這些函數的傳回值，傳回值如果小於 0 表示運動命令不被接受，不被接受的原因請參考與傳回值定義相關的手冊(請參考"**IMP Series 運動控制函式庫參考手冊**")。傳回值如果大於或等於 0，表示 MCCL 給于此運動命令的命令編碼，使用者可從這些運動命令編碼追蹤命令的執行情況。使用 MCC_ResetCommandIndex()可以重置此編碼值，使重新從零開始計數。

A. 直線運動

使用直線運動函式時，只需給定各軸的目的位置或位移量，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。

➔ *See Also* MCC_Line()

B. 圓弧運動

呼叫圓弧運動函式時，只需給定參考點與目的點的座標值，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。**MCCL 也提供 3-D 圓弧運動函式**。

➔ *See Also* MCC_ArcXYZ() MCC_ArcXYZ_Aux()
MCC_ArcXY() MCC_ArcXY_Aux ()

MCC_ArcYZ() MCC_ArcYZ_Aux ()
MCC_ArcZX() MCC_ArcZX_Aux ()
MCC_ArcThetaXY() MCC_ArcThetaYZ()
MCC_ArcThetaZX()

C. 圓運動

呼叫圓運動函式時，只需給定圓心的座標值，並指定運動方向為順時針或逆時針，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。

➔ *See Also* MCC_CircleXY() MCC_CircleXY_Aux ()
MCC_CircleYZ() MCC_CircleYZ_Aux ()
MCC_CircleZX() MCC_CircleZX_Aux ()

D. 螺線運動

呼叫螺線運動函式時，只需給定圓心的座標值、直線進給軸目的點之座標值與 Pitch 值，並指定運動方向為順時針或逆時針，此時將依照給定的進給速度運動，預設的加減速時間為 300ms。

➔ *See Also* MCC_HelicaXY_Z()
MCC_HelicaYZ_X()
MCC_HelicaZX_Y()
MCC_HelicaXY_Z_Aux ()
MCC_HelicaYZ_X_Aux ()
MCC_HelicaZX_Y_Aux ()

E. 一般運動的加減速時間及進給速度設定

欲設定一般運動之加減速時間可使用 MCC_SetAccTime()及 MCC_SetDecTime()；欲設定進給速度時使用 MCC_SetFeedSpeed()。

須另外說明的是，MCCL 計算一般運動之進給速度時只考慮 X/Y/Z 三軸，而 U/V/W/A/B 軸僅是配合前三軸運動同時開始及結束(作直線運動)；但若此筆運動命令之 X/Y/Z 三軸無位移，則所設定之進給速度即改為指定 U/V/W/A/B 中行程最長之軸之速度，其餘四軸則配合同時開始及結束(類似點對點運動行為)。

進給速度的設定不能超過使用 MCC_SetSysMaxSpeed()對進給速度所作的限制，若超過將以 MCC_SetSysMaxSpeed()的設定值為進給速度。使用 MCC_SetSysMaxSpeed()必須在 InitSystem()之前呼叫。

➔ *See Also* MCC_GetFeedSpeed()
MCC_GetCurFeedSpeed()
MCC_GetSpeed()

II. 點對點運動

點對點運動與一般運動中之直線運動相當類似，差別僅在於一般運動之速度使用 MCC_SetFeedSpeed()設定之，單位為 UU/sec；而點對點運動採用最大安全速度***比例***的方式設定之，對應函式是 MCC_SetPtPSpeed()，計算方式為：

各軸點對點運動的進給速度 =
各軸最大安全速度 × (進給速度比例 / 100)

其中

各軸的最大安全速度 = (RPM / 60) × Pitch / GearRatio

有了各軸進給速度後，即可算出各軸所須之時間，MCCL 會以所須時間最長之軸為主，其餘各軸配合同時開始及結束。

點對點運動之加減速時間仍沿用一般運動中之設定。

➔ *See Also* MCC_PtP()
MCC_GetPtPSpeed()

III. 微動、吋動、連續 JOG

A. 微動 JOG：MCC_JogPulse()

要求特定軸移動指定的 pulse 量(最大的位移量為 2048 個 pulse)。在使用此函式時，運動狀態應先處於靜止情況(MCC_GetMotionStatus()的函式傳回值應為 GMS_STOP)。

```
MCC_JogPulse(    10,              0,              0);
                 位移量(pulse)    指定軸          Group 編號
```

B. 吋動 JOG：MCC_JogSpace()

要求特定軸依指定的進給速度比例(可參考點對點運動的說明)移動指定的位移量(單位：UU)。在使用此函式時，運動狀態應先處於靜止情況(使用 MCC_GetMotionStatus()的函式傳回值應為 GMS_STOP)。可以使用 MCC_AbortMotionEx()停止此項運動。下面為使用例子。

```
MCC_JogSpace(    1,        20,             0,          0);
                 位移量    進給速度比例    指定軸      Group 編號
```

C. 連續 JOG 運動：MCC_JogConti()

要求選定軸依指定的進給速度比例(可參考點對點運動的說明)與方向，移動到使用者設定的有效工作區間邊界才停止(有效工作區定義在機構參數中)。在使用此函式時，運動狀態應先處於靜止情況(使用 MCC_GetMotionStatus()的函式傳回值應為

GMS_STOP)。可以使用 MCC_AbortMotionEx()停止此運動。下面為使用例子。

```
MCC_JogConti(    1,          20      ,      0,          0);
                 位移方向  進給速度比例   指定軸   Group 編號
                 (1:正向, -1:反向)
```

IV. 運動暫停、持續、棄置

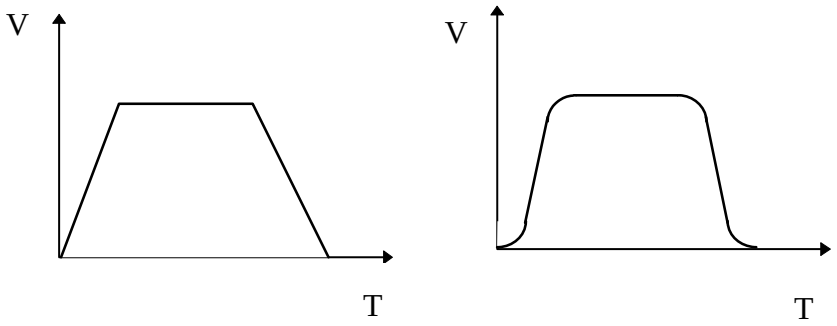
可使用 MCC_AbortMotionEx()捨棄目前正在執行中與庫存的所有運動命令。也可以使用 MCC_HoldMotion()暫停執行中的運動命令(此時將以等減速的方式停止運動)，待使用 MCC_ContiMotion()後，才繼續執行該筆命令尚未完成的部分；但此時也可以使用 MCC_AbortMotionEx()，捨棄尚未完成的部分。

MCC_AbortMotionEx()可使用指定的減速時間來停止運動，若目前已在 Hold 狀態則減速時間參數將被忽略。

➔ *See Also* MCC_GetMotionStatus()

2.6.3 進階軌跡規劃

為達到更有彈性、更具效率的定位控制，MCCL 提供了幾種進階軌跡規劃功能，例如當不同運動命令間不需精確定位，且需快速到達指定位置時，可使用平滑運動(Motion Blending)功能；又在控制系統中常見的追蹤問題，也提供速度強制(Override Speed)功能，允許動態調整進給速度。下面分別說明這些功能。



梯形曲線　　S 形曲線

Figure 2.6.1 加減速型式

I. 加減速型式設定

加減速型式可設定為梯形曲線或 S 形曲線(參考 Figure 2.6.1)。點對點、直線、圓弧、圓、螺線運動各軸的加減速型式均以相同方式設定。

➔ *See Also*

MCC_SetAccType()　　MCC_GetAccType()
MCC_SetDecType()　　MCC_GetDecType()
MCC_SetPtPAccType()
MCC_GetPtPAccType()
MCC_SetPtPDecType()
MCC_GetPtPDecType()

II. 開啟/關閉平滑運動(Motion Blending)

可以使用 MCC_EnableBlend()開啟平滑運動功能，此項功能可滿足不同運動命令間的等速段達到速度平滑連續的要求(也就是在完成前一段運動命令時速度不需減速到停，可直接加速或減速到下一段運動命令要求的速度)。平滑運動功能包括直線-直線、直線-圓弧、圓弧-圓弧間的平滑運動。

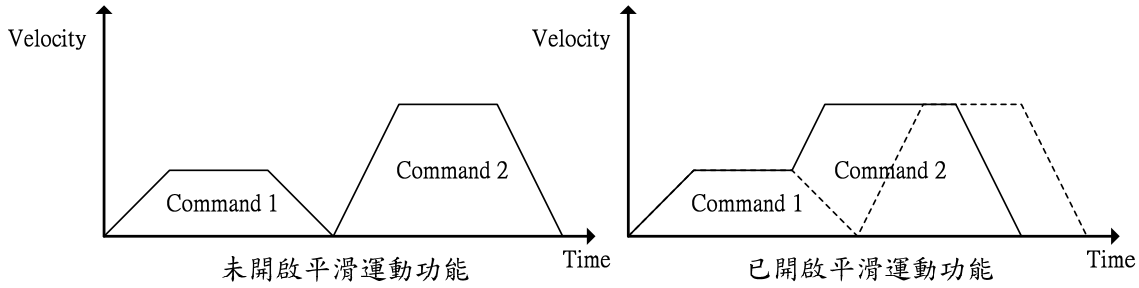
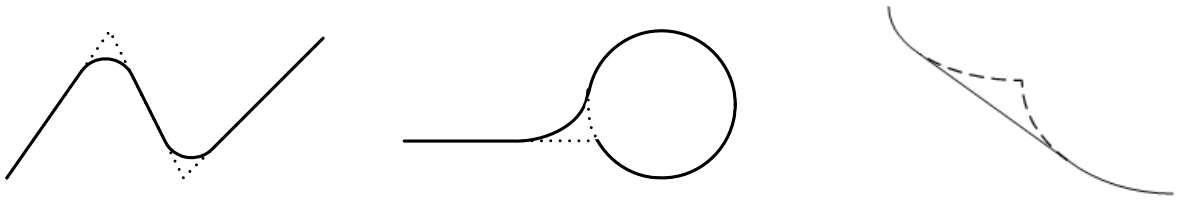


Figure 2.6.2 平滑運動時的速度

由 Figure 2.6.2 可以看出開啟平滑運動功能後的運動情形，第一筆運動命令在達到等速段後不經減速段，而直接加速至第二筆運動命令的等速段(如 Figure 2.6.2 右圖之實線所示)，如此命令的執行時間較快，但各筆命令的連接處會有軌跡失真的狀況存在。Figure 2.6.3 顯示在開啟平滑運動功能後的運動軌跡(虛線代表原先規劃的軌跡曲線)。



直線-直線　　直線-圓弧　　圓弧-圓弧

Figure 2.6.3 直線-直線、直線-圓弧、圓弧-圓弧平滑運動

➔ *See Also*　　MCC_DisableBlend()
MCC_CheckBlend()

III. 速度強制

如需在運動中動態變更進給速度時，可使用速度強制功能，此功

能可將執行中運動命令的速度 V_1 加速到要求的速度值 V_2(當 $V_1 < V_2$)，或由目前的速度 V_3 減速到要求的速度值 V_4 (當 $V_3 > V_4$)。

如 Figure 2.6.4 ， $V_2 = V_1 \times 175 / 100$(因使用 MCC_OverrideSpeed(175))；同理， $V_4 = V_3 \times 50 / 100$(因使用 MCC_OverrideSpeed(50))。

使用 MCC_OverrideSpeed()指定速度比例，即時強制變更切線速度。速度比例的定義為：

速度比例 = 要求變更後的進給速度 / 原進給速度 × 100

原進給速度是指使用 MCC_SetFeedSpeed()或 MCC_SetPtPSpeed()所設定的速度。注意：***使用 MCC_OverrideSpeed()後，將影響往後全部運動的速度，而不只是影響執行中的運動***。

➔ *See Also* MCC_GetOverrideRate()

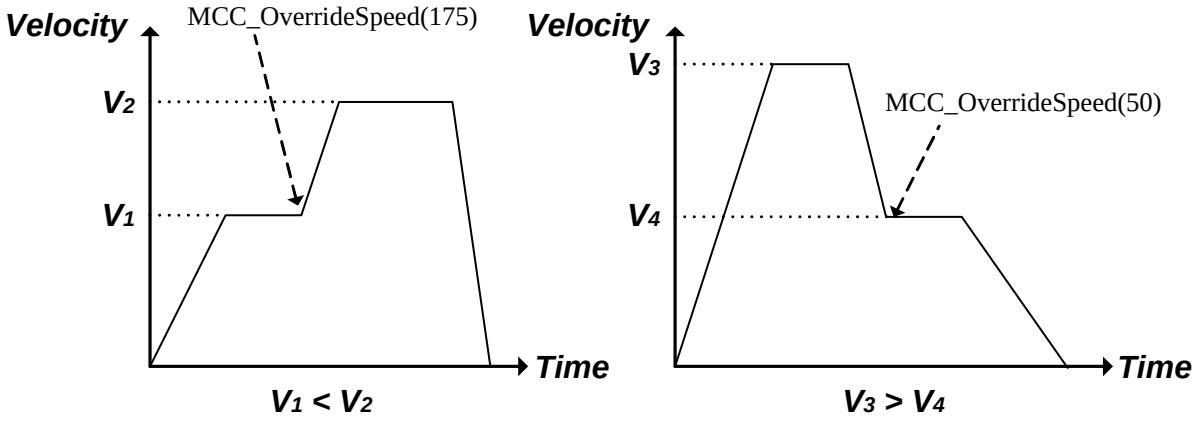


Figure 2.6.4 速度強制

點對點運動速度強制：

使用 MCC_OverridePtPSpeed()即時強制變更各軸速度，此函式所需的參數為變更各軸的速度比例為原來速度比例的多少百分比再乘以 100，可參考前面的說明。使用 MCC_OverridePtPSpeed()後，將影

響全部點對點運動的速度，而不只是影響執行中的點對點運動。

➔ *See Also* MCC_GetPtPOverrideRate()

IV. 運動空跑

使用 MCC_EnableDryRun()可開啟運動空跑功能，此時軌跡規劃的結果並不由運動控制平台送出，但使用者仍可利用 MCC_GetCurPos()與 MCC_GetPulsePos()讀取軌跡規劃的內容，除了可事先獲得運動路徑外，使用者並可利用這些資訊在螢幕上模擬運動軌跡。

➔ *See Also* MCC_DisableDryRun()
MCC_CheckDryRun()

V. 運動延遲

可以使用 MCC_DelayMotion()強迫延遲執行下一個運動命令，延遲的時間以毫秒(ms)為單位，以下面的範例為例：

```
MCC_Line(10, 10, 10, 0, 0, 0, 0, 0, 1);-------- A
MCC_DelayMotion(200, 1);
MCC_Line(15, 15, 15, 0, 0, 0, 0, 0, 1);-------- B
```

則在執行完運動命令 A 後將延遲 200 ms，再繼續執行運動命令 B。

➔ *See Also* MCC_GetMotionStatus()

VI. 錯誤訊息

當發生運動過行程(運動超出軟體邊界)、進給速度超出最大設定

值、加/減速度超出最大設定值、圓弧命令參數錯誤、圓弧命令執行錯誤等情況時，可利用 MCC_GetErrorCode()讀取錯誤碼獲知錯誤的內容(錯誤碼意義請參考"**IMP Series 運動控制函式庫參考手冊**")。

當 Group 發生錯誤時，此 Group 將不再執行運動命令。此時使用者需自行使用 MCC_GetErrorCode()判斷錯誤原因並排除之，爾後使用 MCC_ClearError()清除錯誤紀錄，使 Group 恢復至正常狀態。

2.6.4 插值時間與加減速時間

I. 設定插值時間

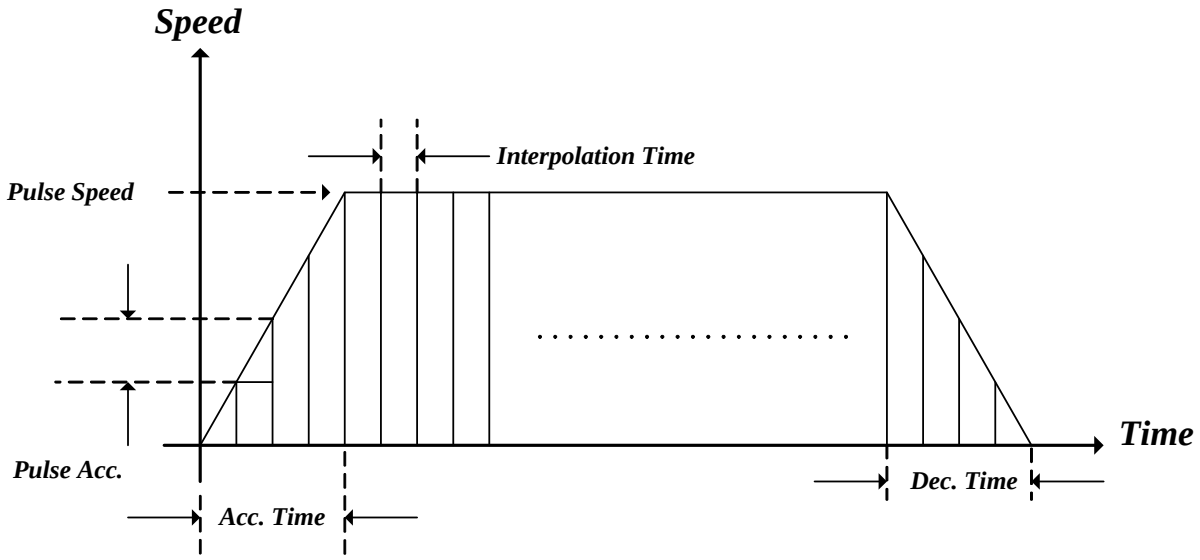


Figure 2.6.5 軌跡規劃設定參數

插值時間(Interpolation Time)是指距離下一個插值點的時間距離(參考 Figure 2.6.5)，可設定值最小為 1 ms，最大為 50 ms。

II. 最大 pulse 速度設定

最大 pulse 速度(Max. Pulse Speed)用來限制各軸在每一個插值時間內所能送出的最大 pulse 量，也就是會限制各軸最大的進給速度。

可以使用 MCC_SetMaxPulseSpeed()來設定最大 pulse 速度，可設定的範圍為 1 ~ 32767，系統預設值為 32767。

➔ *See Also* MCC_GetMaxPulseSpeed()

III. 最大 pulse 加、減速度設定

最大 pulse 加、減速度(Max. Pulse Acceleration / Deceleration)用來限制相鄰插值時間之間所送出 pulse 的最大差量。在運動過程中若加、減速時間不足，有可能加、減速度會超過機構的容許值，如此可能因為運動慣量太大，而造成機構的損傷；此項設定值可用來限制所送出 pulse 的差量在機構的容許範圍內。使用者可以利用 MCC_GetErrorCode()判斷在運動過程中加、減速度是否超出設定範圍。可以使用 MCC_SetMaxPulseAcc()來設定最大 pulse 加、減速度，可設定的範圍為 1 ~ 32767，預設值為 32767 pulses。

➔ *See Also* MCC_GetMaxPulseAcc()

IV. 加、減速段所需時間

可以設定一般運動與點對點運動加速到穩定速度所需的時間，也可以設定由穩定速度減速到停止運動所需的時間。使用 MCC_SetAccTime()與 MCC_SetDecTime()設定直線、圓弧、圓、螺線運動所需的加、減速時間；使用 MCC_SetPtPAccTime()與 MCC_SetPtPDecTime()設定點對點運動所需的加、減速時間。通常在給定較快的進給速度時會要求較長的加速度時間，因此 MCC_SetAccTime() 與 MCC_SetDecTime() 通常會與 MCC_SetFeedSpeed()搭配使用；同樣的，MCC_SetPtPAccTime()與 MCC_SetPtPDecTime()通常會與 MCC_SetPtPSpeed()搭配使用。

下面的範例說明在不同的進給速度時，要求不同的加、減速時

間。通常使用者需依照機構特性自訂 SetSpeed()的內容，在要求變更進給速度時需使用 SetSpeed()，而非直接呼叫 MCC_SetFeedSpeed()，尤其是在使用步進馬達時，以免造成失步的情況。

```
void SetSpeed(double dfSpeed)
{
    double dfAcc, dfTime;

    dfAcc = 0.04; //  設定加速度為  0.04 (UU/sec^2)

    if (dfSpeed > 0)
    {
        dfTime = dfSpeed / dfAcc;

        MCC_SetAccTime(dfTime);
        MCC_SetDecTime(dfTime);

        MCC_SetFeedSpeed(dfSpeed);
    }
}
```

2.6.5 系統狀態檢視

MCCL 所提供的函式能檢視目前的實際位置、規劃速度與實際速度、運動狀態、運動命令庫存量、硬體 FIFO 中的細運動命令(FMC)庫存量、執行中運動命令的內容。

使用 MCC_GetCurPos()可以獲得目前的命令位置，單位為 UU。

MCC_GetPulsePos()可獲得目前已從控制平台輸出的 pulse 量，此值與使用 MCC_GetCurPos()所讀到之值只差別在後者經過機構參數的轉換。

假使系統有安裝編碼器，則可以使用 MCC_GetENCValue()讀回

目前的實際位置(讀取值為編碼器計數值)。

使用 MCC_GetPtPSpeed()可以獲得點對點運動規劃的進給速度比例，而利用 MCC_GetFeedSpeed()可以獲得一般運動規劃的進給速度；而對於一般運動，尚可以使用 MCC_GetCurFeedSpeed()獲得目前實際的切線速度，使用 MCC_GetSpeed()則可以獲得目前各軸實際的進給速度。

利用呼叫 MCC_GetMotionStatus()所獲得的傳回值可以判斷目前的運動狀態；傳回值若為 GMS_RUNNING，表示系統正處於運動狀態；傳回值若為 GMS_STOP，表示系統處於停止狀態，已無任何未執行的庫存命令；傳回值若為 GMS_HOLD，表示系統因使用 MCC_HoldMotion()暫停中；傳回值若為 GMS_DELAYING，表示系統因使用 MCC_DelayMotion()目前正在延遲中。

使用 MCC_GetCurCommand()可以獲得目前正在執行的運動命令相關的資訊，MCC_GetCurCommand()的函式宣告如下：

```
MCC_GetCurCommand(COMMAND_INFO *pstCurCommand,
                  WORD wGroupIndex)
```

COMMAND_INFO 儲存目前執行中的運動命令內容，他被定義為：

```
typedef struct _COMMAND_INFO
{
    int         nType;
    int         nCommandIndex;
    double      dfFeedSpeed;
    double      dfPos[8];
} COMMAND_INFO;
```

其中

nType：運動命令類型

0. 點對點運動
1. 直線運動
2. 順時針圓弧、圓運動
3. 逆時針圓弧、圓運動
4. 順時針螺線運動
5. 逆時針螺線運動
6. 運動延遲
7. 開啟平滑運動
8. 關閉平滑運動
9. 開啟定位確認
10. 關閉定位確認

nCommandIndex：此運動命令編碼

dfFeedSpeed：

一般運動	進給速度
點對點運動	進給速度比例
運動延遲	目前剩餘的延遲時間(單位：ms)

dfPos[]：要求的目的點位置

使用 MCC_GetCommandCount()可以獲得目前尚未被執行的運動命令之庫存量，此庫存量不包括正在執行的運動命令。

使用 MCC_GetCurPulseStockCount()可以讀取 IMP Series 運動控制平台上的細運動命令(Fine Movement Command, FMC)庫存筆數。在持續運動過程中，FMC 庫存筆數預設值為 60，使用者可自行依需求設定其 FMC 庫存筆數，確保穩定的運動性能。若存在 FMC 庫存筆數等於 0 的現象，必須延長插值時間(請參考前面對插值時間的介紹)。另外，若人機操作畫面的顯示出現遲滯的現象，也可考慮延長插值時間。

2.7 定位控制

MCCL 提供之定位控制功能包括：

1. 閉迴路控制器增益設定
2. 定位確認
3. 跟隨誤差偵測
4. 位置閉迴路控制失效處理
5. 齒輪齒隙、背隙補償

以下各節將一一介紹其內容及使用方式。

2.7.1 閉迴路比例積分微分前饋增益(PID+FF Gain)設定

使用 MCC_SetPGain()、MCC_SetIGain()、MCC_SetDGain()與 MCC_SetFGain()，設定控制閉迴路中的比例積分微分前饋增益參數，可設定範圍為 0 ~ 255。比例積分微分前饋增益參數的調整方式為：在調整完驅動器為電壓命令後，使用安裝光碟所提供的整合測試環境(ITE)中的[View Profile]功能，利用跟隨誤差調整比例增益(跟隨誤差為命令位置與實際位置間的誤差量)。

➔ *See Also* MCC_GetPGain()
MCC_GetIGain()
MCC_GetDGain()
MCC_GetFGain()

2.7.2 定位確認

MCCL 所提供之定位確認功能在確保執行中之運動命令已實際到達目標位置(容許誤差範圍內)後，才繼續執行下一筆命令，否則棄置後續命令並產生錯誤記錄(亦可選擇忽略)。

欲開啟此功能可呼叫 MCC_EnableInPos()，爾後 MCCL 會在送出

該運動命令(Motion Command)之最後一筆細運動命令(FMC)後，開始檢查是否滿足定位確認條件；若是，則執行下一筆運動命令，但若一直等到所設定之最大檢查時間(利用 MCC_SetInPosMaxCheckTime()函式設定)截止前仍未定位，則棄置後續命令並產生錯誤記錄(最大檢查時間定義可參考 Figure 2.7.1)。

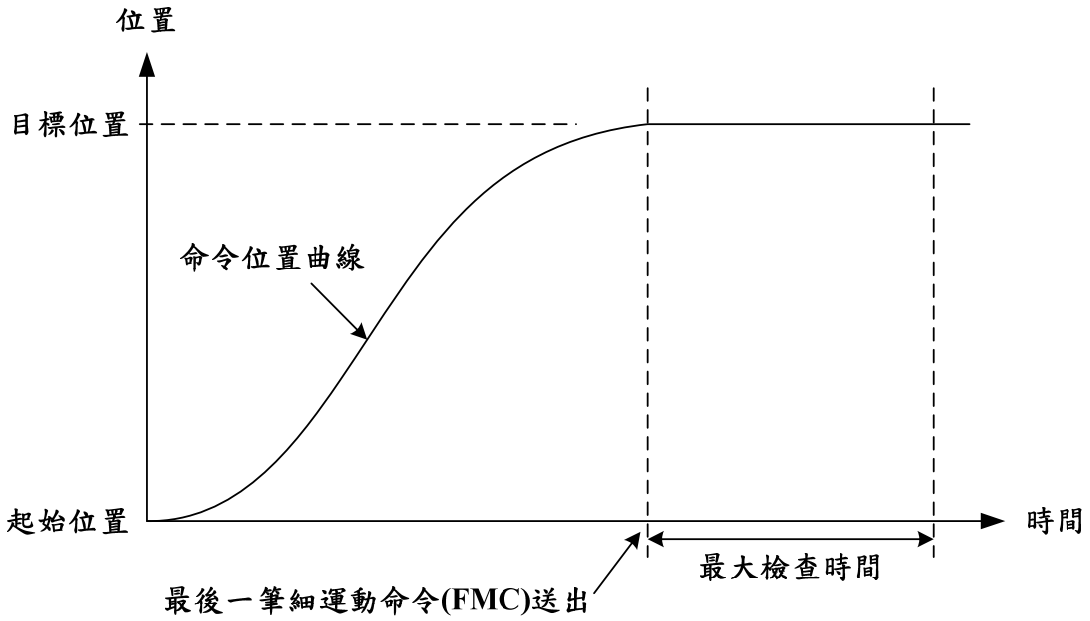


Figure 2.7.1 最大檢查時間示意圖

MCCL 提供了四種定位確認模式，使用者可透過 MCC_SetInPosMode()函式來選擇適合的方式。以下將一一介紹各模式之定義：

模式 IPM_ONETIME_BLOCK：

當 Group 中各軸的位置誤差皆小於或等於定位誤差容許範圍(可以利用 MCC_SetInPosToleranceEx()設定之，單位為 UU)，即滿足此模式定位條件(參考 Figure 2.7.2)。

若到最大檢查時間截止前皆未滿足上述條件，則棄置後續命令並產生錯誤記錄(可以 MCC_GetErrorCode()獲得)。

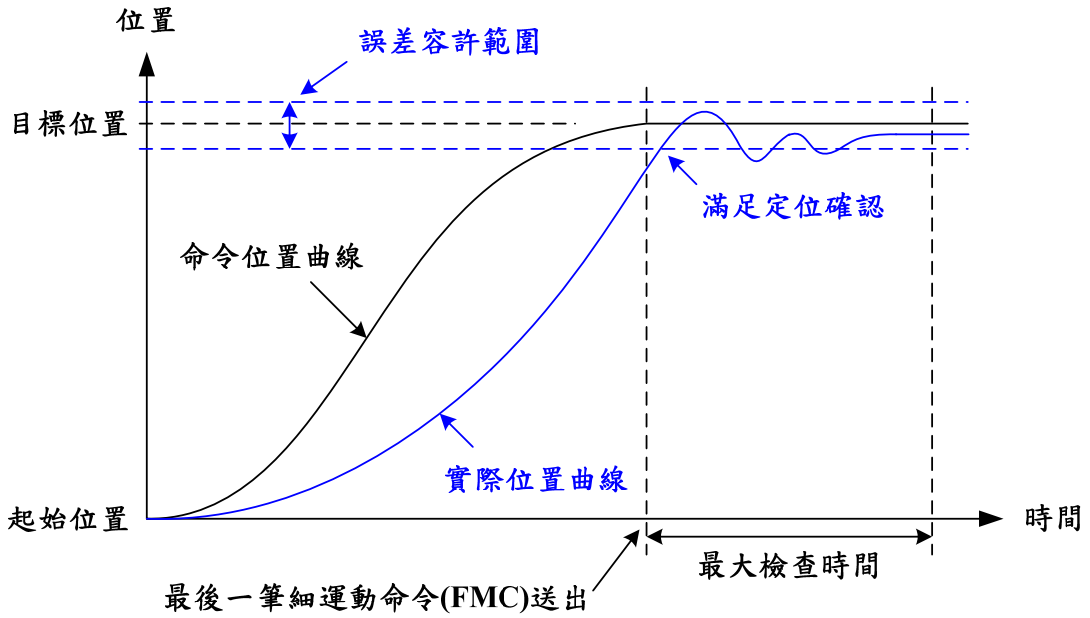


Figure 2.7.2 IPM_ONETIME_BLOCK 模式定位成功示意圖

模式 IPM_ONETIME_UNBLOCK：

此模式與 IPM_ONETIME_BLOCK 模式之定位條件相同，差別僅在於當最大檢查時間截止時，若未滿足定位條件，並不產生錯誤記錄而直接執行後續命令。

模式 IPM_SETTLE _BLOCK：

當 Group 中各軸的位置誤差皆小於或等於定位誤差容許範圍(可以利用 MCC_SetInPosToleranceEx()設定之，單位為 UU)，***並持續一段穩定時間***(可以利用 MCC_SetInPosSettleTime()設定之，單位為 ms)，即滿足此模式定位條件(參考 Figure 2.7.3)。

若到最大檢查時間截止前皆未滿足上述條件，則棄置後續命令並產生錯誤記錄(可以 MCC_GetErrorCode()獲得)。

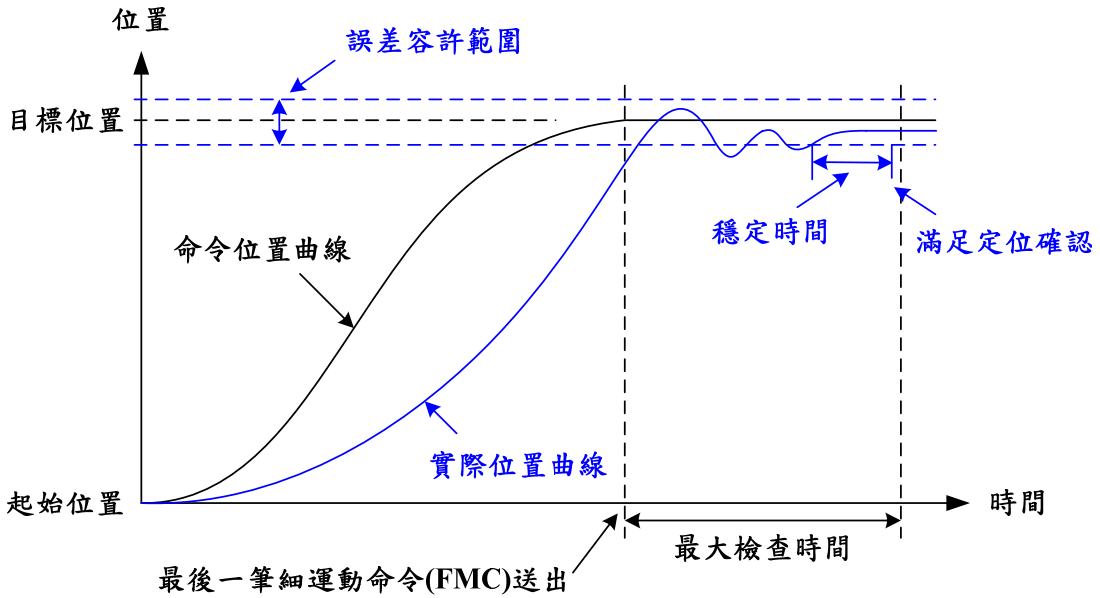


Figure 2.7.3 IPM_SETTLE _BLOCK 模式定位成功示意圖

模式 IPM_SETTLE _UNBLOCK：

此模式與 IPM_SETTLE_BLOCK 模式之定位條件相同，差別僅在於當最大檢查時間截止時若未滿足定位條件，並不產生錯誤記錄而直接執行後續命令。

定位容許誤差愈大，則完成定位確認所須時間相對愈少，但在運動命令連接點處與規劃路徑之誤差會較大(反之則誤差較小)。由圖 Figure 2.7.4 可看出，較小的定位容許誤差將產生較小的軌跡誤差(Error 1 < Error 2)；因此定位容許誤差需視不同功能的系統而做適當的設定。另外，可透過 MCC_GetInPosStatus()來得到 Group 中各運動軸之定位狀態。

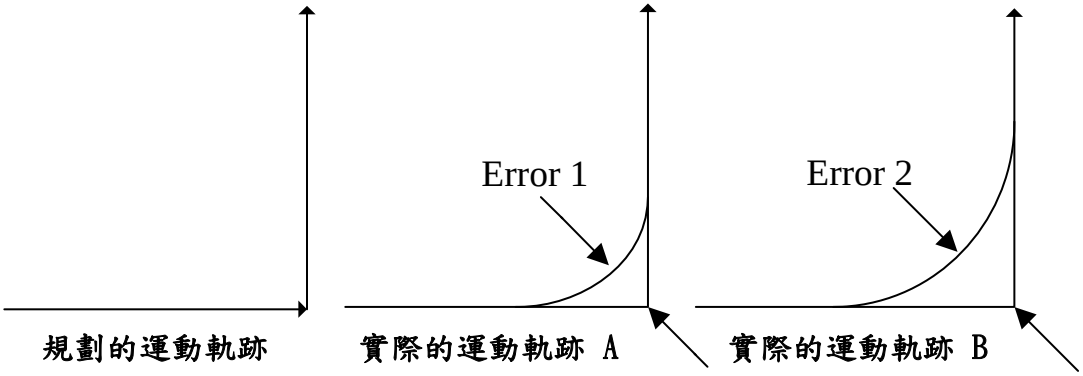


Figure 2.7.4 定位誤差對路徑誤差影響

➔ *See Also* MCC_GetInPosToleranceEx()
MCC_DisableInPos()

注意

1. 因定位確認功能是比對機台***實際位置***與目標位置是否進入容許誤差範圍，故開啟此功能之運動軸必須配接編碼器，否則將永遠無法完成定位確認。
2. 系統一旦判定定位確認成功，則不再進行定位確認判斷(表示會保持在定位成功的狀態，即使實際位置又離開定位容許範圍，參考 Figure 2.7.2)，直到有新的運動命令下達。

2.7.3 跟隨誤差(Tracking Error)偵測

任何時刻下，運動軸之命令位置與實際位置之誤差，稱為跟隨誤差(參考 Figure 2.7.5)。

一般情況中，跟隨誤差大小與機構特性、閉迴路比例增益值及運動加速度等設定有關；過大的跟隨誤差表示運動已偏離(或落後)規劃路徑太遠，更甚者有可能已經撞機。

欲使用此功能，可先透過 MCC_SetTrackErrorLimit()設定誤差容許範圍，再利用 MCC_EnableTrackError()函式來開啟此功能。功能開

啟後，一旦發現運動軸之跟隨誤差超出範圍，則停止對該 Group 之後續命令規劃並產生錯誤記錄，使用者可使用 MCC_GetErrorCode()得到錯誤代碼：0xF801 ~ 0xF808 分別代表 X、Y、Z、U、V、W、A、B 出現過大之跟隨誤差。

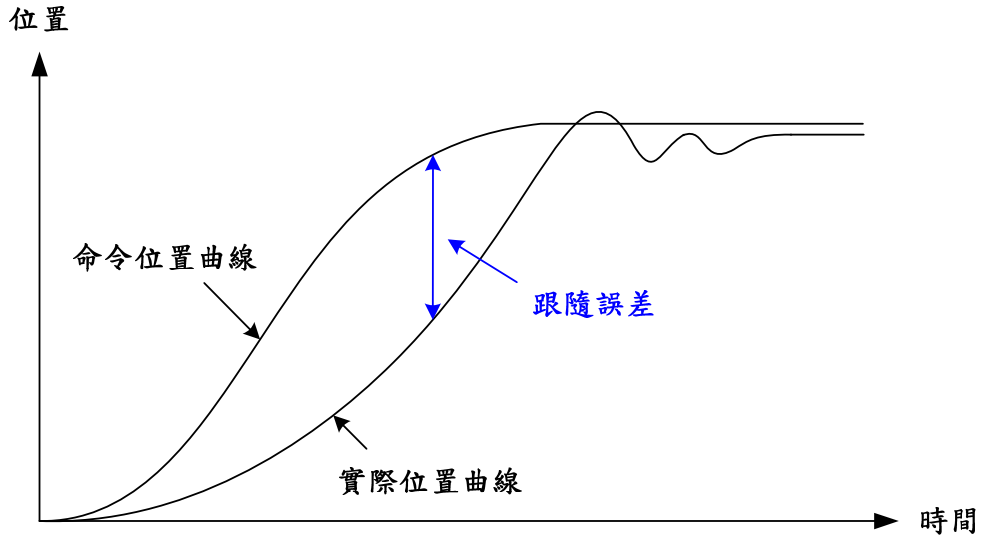


Figure 2.7.5 跟隨誤差示意圖

➔ *See Also* MCC_DisableTrackError()
MCC_GetTrackErrorLimit()

資訊

當使用 MCCL，在任何情況下若呼叫 MCC_GetErrorCode()得到非零之回傳值，表示該 Group 已產生錯誤記錄，處理方式為：

1. 判斷錯誤種類並做相對應錯誤排除動作(使用者應自行定義之)
2. 呼叫 MCC_ClearError()清除錯誤記錄
3. 系統繼續正常運作

2.7.4 位置閉迴路控制失效處理

當比例增益參數設定不當，或其他操作上的原因使得位置閉迴路

控制功能失效時，系統將處於不受控狀態。使用者可由MCC_SetErrorCountThreshold()函式設定正負脈波誤差容忍大小，當系統誤差暫存器大於使用者設定大小，系統即時通知使用者系統處於不受控狀態，運動控制平台會自動產生中斷信號，使用者可自訂一個位置控制閉迴路中斷處理函式，並串接至系統，當出現運動軸位置閉迴路控制功能失效時，此自訂函式將被呼叫，使用者可將處理程序設計在此自訂函式中。此項功能的使用步驟如下：

***Step 1*：使用 MCC_SetPCLRoutine()串接自訂的中斷服務函式**

需先設計自訂的中斷服務函式，函式的宣告必須遵循下列的定義：

typedef void(_stdcall *_PCLISR_)(_PCLINT_*)

例如自訂的函式可設計如下：

```
_stdcall MyPCLFunction(PCLINT *pstINTSource)
{

    //  判斷是否因 Channel 0 的位置誤差暫存器大於設定正誤差大小而觸發此函式
    if (pstINTSource->OVP0)
    {
     //  Channel 0 的位置閉迴路控制功能失效的處理程序
     }

    //  判斷是否因 Channel 0 的位置誤差暫存器小於設定負誤差大小而觸發此函式
    if (pstINTSource->OVN0)
    {
     //  Channel 0 的位置閉迴路控制功能失效的處理程序
     }
```

```
    //  判斷是否因 Channel 1 的位置誤差暫存器大於設定正誤差大小
而觸發此函式

    if (pstINTSource->OVP1)
    {
     //  Channel 1 位置閉迴路控制功能失效的處理程序
     }

    //  判斷是否因 Channel 1 的位置誤差暫存器小於設定負誤差大小
而觸發此函式
    if (pstINTSource->OVN1)
    {
     //  Channel 1 位置閉迴路控制功能失效的處理程序
     }

    //  判斷是否因 Channel 2 的位置誤差暫存器大於設定正誤差大小
而觸發此函式
    if (pstINTSource->OVP2)
    {
     //  Channel 2 的位置閉迴路控制功能失效的處理程序
     }

    //  判斷是否因 Channel 2 的位置誤差暫存器小於設定負誤差大小
而觸發此函式
    if (pstINTSource->OVN2)
    {
     //  Channel 2 的位置閉迴路控制功能失效的處理程序
     }
```

```
    //  判斷是否因 Channel 3 的位置誤差暫存器大於設定正誤差大小
而觸發此函式
    if (pstINTSource->OVP3)
    {
     //  Channel 3 的位置閉迴路控制功能失效的處理程序
     }

    //  判斷是否因 Channel 3 的位置誤差暫存器小於設定負誤差大小
而觸發此函式
    if (pstINTSource->OVN3)
    {
     //  Channel 3 的位置閉迴路控制功能失效的處理程序
     }
```

不可以使用”else if (pstINTSource->*OVP1*)”類似的語法，因 pstINTSource->*OVP0* 與 pstINTSource->*OVP1* 有可能同時不為 0。

接著使用 MCC_SetPCLRoutine(MyPCLFunction)串接自訂的中斷服務函式。當自訂函式被觸發執行時，可以利用傳入自訂函式中、被宣告為 *PCLINT* 的 pstINTSource 參數，判斷此刻自訂函式被呼叫是因滿足何種觸發條件。*PCLINT* 的定義如下：

```
typedef struct _PCL_INT
{
     BYTE OVP0;
     BYTE OVP1;
     BYTE OVP2;
     BYTE OVP3;
     BYTE OVP4;
     BYTE OVP5;
     BYTE OVP6;
     BYTE OVP7;
```

BYTE *OVN0*;
BYTE *OVN1*;
BYTE *OVN2*;
BYTE *OVN3*;
BYTE *OVN4*;
BYTE *OVN5*;
BYTE *OVN6*;
BYTE *OVN7*;

} *PCLINT*;

PCLINT 中的欄位值如果不為 0，表示自訂函式被呼叫的原因，各欄位所表示的觸發原因如下：

OVP0	Channel 0 位置誤差暫存器大於設定之正誤差
OVP1	Channel 1 位置誤差暫存器大於設定之正誤差
OVP2	Channel 2 位置誤差暫存器大於設定之正誤差
OVP3	Channel 3 位置誤差暫存器大於設定之正誤差
OVP4	Channel 4 位置誤差暫存器大於設定之正誤差
OVP5	Channel 5 位置誤差暫存器大於設定之正誤差
OVP6	Channel 6 位置誤差暫存器大於設定之正誤差
OVP7	Channel 7 位置誤差暫存器大於設定之正誤差
OVN0	Channel 0 位置誤差暫存器小於設定之負誤差
OVN1	Channel 1 位置誤差暫存器小於設定之負誤差
OVN2	Channel 2 位置誤差暫存器小於設定之負誤差
OVN3	Channel 3 位置誤差暫存器小於設定之負誤差
OVN4	Channel 4 位置誤差暫存器小於設定之負誤差
OVN5	Channel 5 位置誤差暫存器小於設定之負誤差
OVN6	Channel 6 位置誤差暫存器小於設定之負誤差
OVN7	Channel 7 位置誤差暫存器小於設定之負誤差

2.7.5 齒輪齒隙、背隙補償

機台做定位控制時，齒輪、螺桿製造上的缺陷會造成機台傳動時的位置誤差，如齒隙誤差、背隙誤差(參考 Figure 2.7.6)。

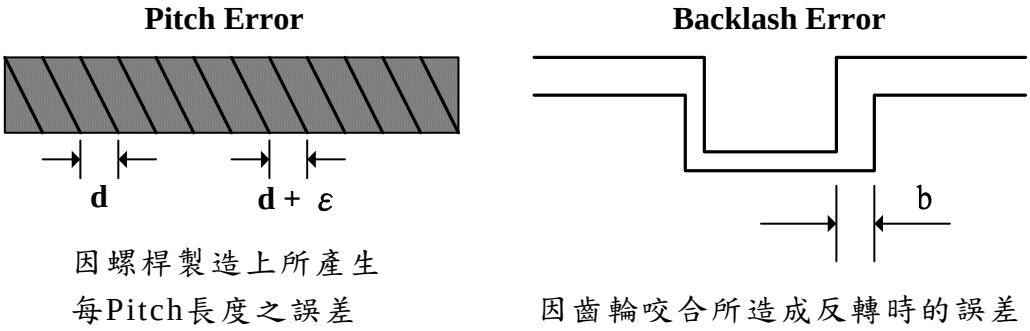


Figure 2.7.6 齒輪齒隙、間隙誤差

使用者可將機台分成多個小區段(參考 Figure 2.7.7)，使用雷射量測儀，在正負向來回掃瞄一次，將區段點的誤差量記錄下來，並建成正、負向補償表。正、負向補償量表為一個二維陣列，存放各軸所有補償點的補償量，所有補償點皆使用一個量測點為基準(參考 Figure.2.7.7)。使用者需給定 *dwInterval*、*wHome_No*、正、負向補償表 (*nForwardTable* 與 *nBackwardTable*)，並呼叫補償設定函式 MCC_SetCompParam()與 MCC_UpdateCompParam()，即可進行補償功能。MCCL 提供各軸 256 個補償點，最多可將機台的各軸分為 255 個補償區段，在各區段間會使用線性補償的方法。

使用補償功能時，補償參數的內容必須涵蓋機台全部的工作行程，以避免產生不正常的動作，因此應在尚未完成原點復歸動作前啟動補償功能，可以搭配使用 MCC_GetGoHomeStatus()檢查原點復歸動作是否已完成(函式傳回值如為 1，表示原點復歸動作已完成)。

停止補償功能的方法是將補償參數中的 *dwInterval* 欄位設定為 0，例如停止 Channel 0 的補償功能可執行下面的程式碼：

```
SYS_COMP_PARAM      stUserCompParam;
```

```
stUserCompParam.dwInterval = 0;

MCC_SetCompParam(&stUserCompParam, 0, 0);
MCC_UpdateCompParam();
```

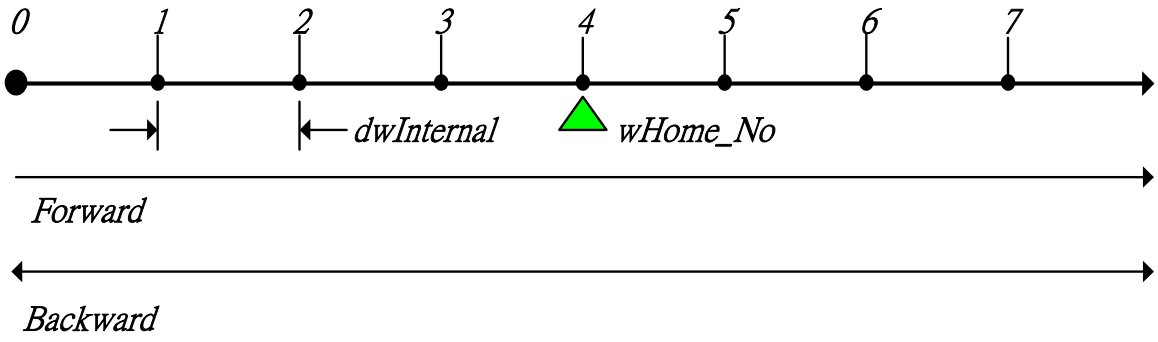


Figure 2.7.7 補償區段

在使用補償功能前需先設定補償參數，補償參數的定義如下：

```
typedef struct _SYS_COMP_PARAM
{
    DWORD       dwInterval;
    WORD        wHome_No;
    WORD        wPaddle;
    int         nForwardTable[256];
    int         nBackwardTable[256];
} SYS_COMP_PARAM;
```

dwInterval :

補償區段的間距，以 pulse 量為單位，此值若小於 0 或等於 0，表示不進行補償功能。

wHome_No： 為各軸原點所在位置之補償點編號。

wPaddle：保留欄位

nForwardTable：指向正向補償量表的指標變數

nBackwardTable：指向負向補償量表的指標變數

以 Figure.2.7.7 為例，假使將 X 軸的工作區域劃分為 7 個補償區段，也就是共需量測 8 個補償點(0 ~ 7)，其中原點位於編號 4 的補償點上，也就是在完成原點復歸動作後，系統會認為目前處於編號 4 的位置上。假使目前將 *dwInterval* 設為 10000(pulses)，表示正向的工作範圍為 10000 × (7 – 4) = 30000 (pulses)，負向的工作範圍為 10000 × (4 - 0) = 40000 (pulses)。機構參數中的 *dwHighLimit* 與 *dwLowLimit* 需配合這些設定值。各軸的補償參數要分開設定，下面為設定 X 軸補償參數的例子。

```
SYS_COMP_PARAM     stUserCompParam;

stUserCompParam.dwInterval          = 10000;
stUserCompParam.wHome_No            = 4;

stUserCompParam.nForwardTable[0]    = 22;//   單位為 pulse
stUserCompParam.nForwardTable[1]    = 20;
stUserCompParam.nForwardTable[2]    = 15;
stUserCompParam.nForwardTable[3]    = 11;
stUserCompParam.nForwardTable[4]    = 0; // 原點所在位置，設為 0
stUserCompParam.nForwardTable[5]    = 10;
stUserCompParam.nForwardTable[6]    = 12;
stUserCompParam.nForwardTable[7]    = 15;
```

```
MCC_SetCompParam(&stUserCompParam, 0, CARD_INDEX);
MCC_UpdateCompParam();
```

如同上面所言，使用者最多可將機台分成 255 個補償區段，而在每一個補償區段中將使用線性補償的方法的方式進行補償功能。例如目前機台的 X 軸(正處於編號 4 的位置上)需向右前進 15000 pulses，由齒隙誤差補償表(參考 stUserCompParam)可得知此位置位於 nForwardTable[5]與 nForwardTable[6]所定義的區段之間(因所在位置介於 10000 pulses 與 20000 pulses 之間)，因 nForwardTable[5]之值為 10、 nForwardTable[6] = 12，且 nForwardTable[6]- nForwardTable[5]= 12 - 10 = 2，因此實際上系統共送出 15000 + 10 + (int)((15000 – 10000)/ 10000 × 2) = 15000 + 10 + 1 = 15011 pulses。

2.8 原點復歸

配合原點復歸參數的設定內容，使用者可設定各軸原點復歸的順序、加減速度時間、速度、原點復歸方向及模式。原點復歸參數的內容如下，各項參數的意義請參考”**2.4.3 原點復歸參數**“的說明。

原點復歸參數(*SYS_HOME_CONFIG*)：

```
typedef struct _SYS_HOME_CONFIG
{
    WORD        wMode;
    WORD        wDirection;
    WORD        wSensorMode;
    WORD        wPaddle0;
    int         nIndexCount;
    int         nPaddle1;
    double      dfAccTime;
    double      dfDecTime;
    double      dfHighSpeed;
    double      dfLowSpeed;
    double      dfOffset;
} SYS_HOME_CONFIG;
```

2.8.1 原點復歸模式說明

原點復歸參數中的 *wMode* 用來指定原點復歸運動的使用模式，對於那些需搭配檢測 Home Sensor 訊號的使用模式，會在進行原點復歸動作前先行檢查起始點是否在正確位置上，下面兩種情形被定義為起始點位置在不正確位置上 (假設原點復歸運動的起始方向為向右)：

a. 原點復歸動作的起始點位置在 Home Sensor 區域中(如 Figure 2.8.1

的 Case 2)

b. 依照指定的方向運動將無法進入 Home Sensor 區域，且將碰觸到極限開關(如 Figure 2.8.1 的 Case 3)

假使出現上述兩種起始位置不正常情況，MCCL 將進行下列的處理程序：

a. 以 *dfHighSpeed* 速度往指定方向移動，直到觸發極限開關後急停。

b. 以 *dfHighSpeed* 速度往相反方向移動，直到進入 Home Sensor 區域，並繼續移動，直到出了 Home Sensor 區域後才減速停止。

c. 開始進行真正的原點復歸動作(也就是 *Case 1 的動作)*

文後所介紹的各種原點復歸模式如需搭配檢測 Home Sensor 訊號，皆有可能出現 Case 2 和 Case 3 之情形，將不再多做說明，僅針對一般情況 Case 1 講解。

另外，加速時間 *dfAccTime*，代表速度從 0 加速到 *dfHighSpeed*(或 *dfLowSpeed*) 的時間值；減速時間 *dfDecTime*，代表速度從 *dfHighSpeed*(或 *dfLowSpeed*)減速到 0 的時間值。而所謂”急停”指的是無減速動作，要求運動軸即刻停止。

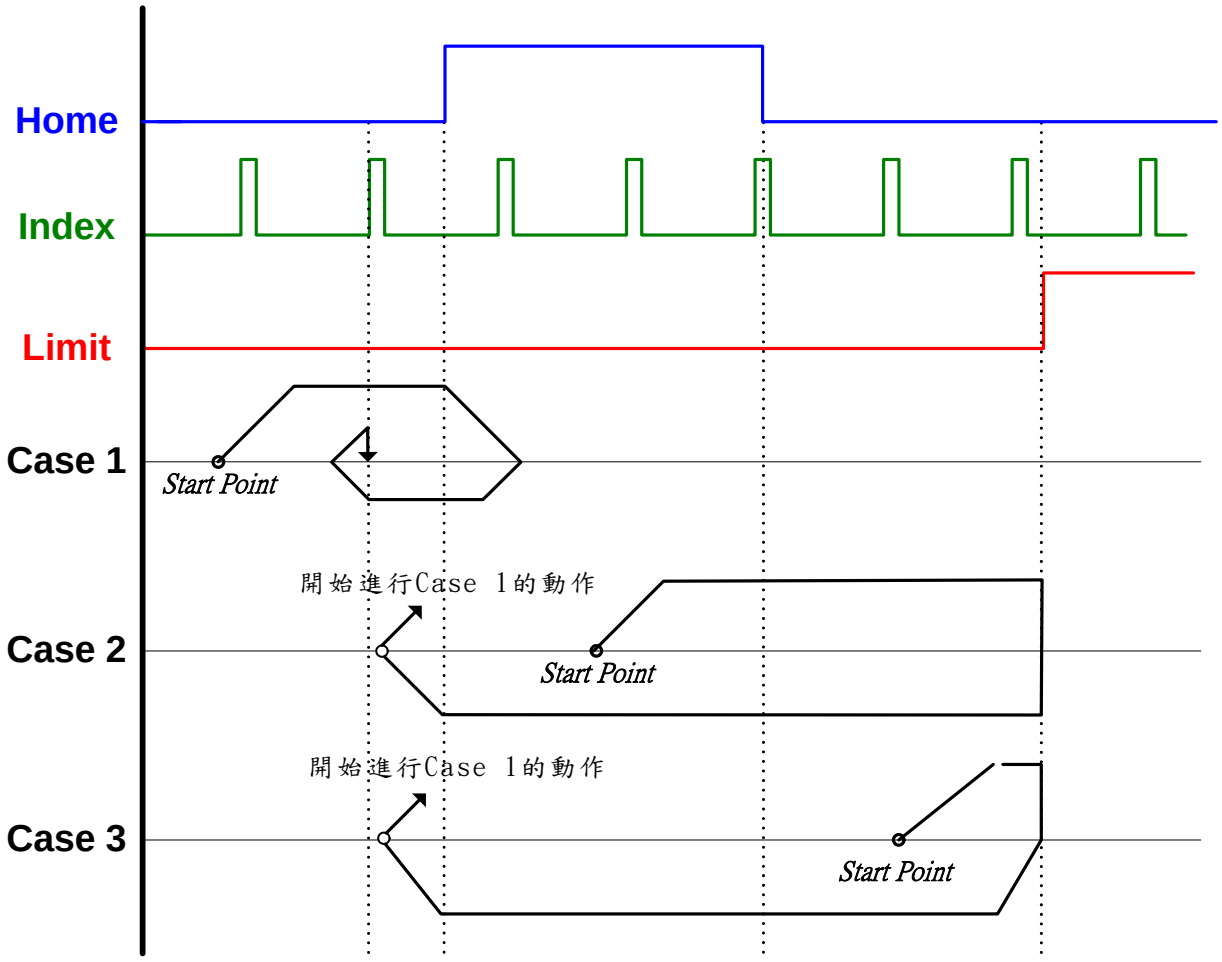
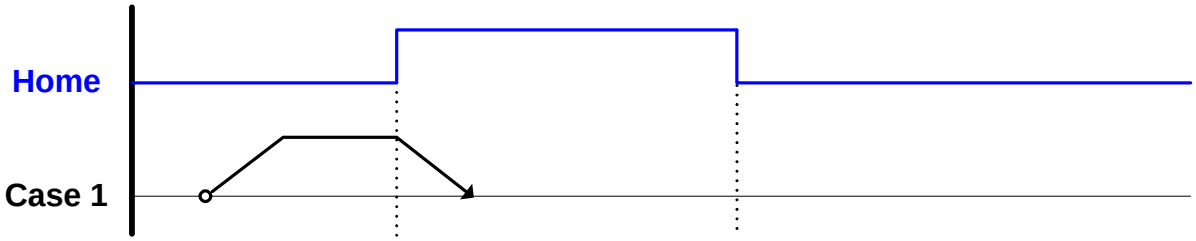


Figure 2.8.1 不同起始點對原點復歸動作的影響

下面分別說明各模式的操作特性：

a. 模式 3 (*wMode* = 3) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

以 *dfHighSpeed* 往指定的方向移動，當進入 Home Sensor 區域時減速停止，動作完成。(此時機台將停於機械原點上，接下來 MCCL 會根據參數 *dfOffset* 的設定值【詳見 2.4.1&2.4.3】，將機台移動到邏輯原點上，如此才算完成原點復歸全部的動作；以下所有模式皆同)。

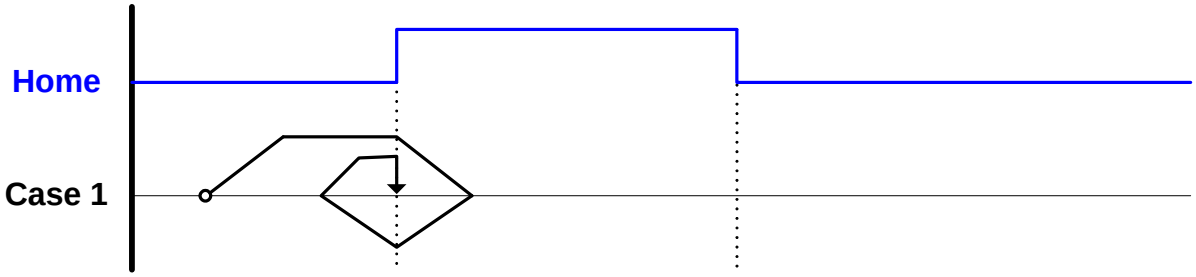


b. 模式 4 (*wMode* = 4) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時減速停止。

Step 2： 以 *dfHighSpeed* 速度往相反方向移動，離開 Home Sensor 區域後減速停止。

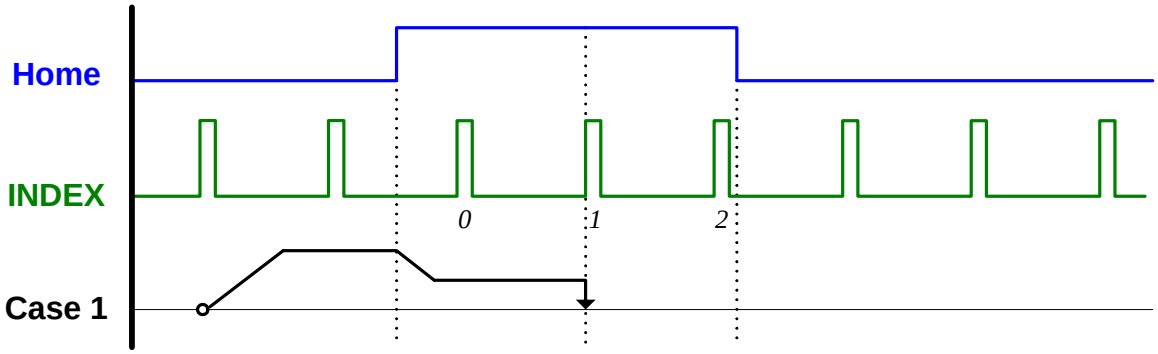
Step 3： 以 *dfLowSpeed* 速度往指定方向移動，進入 Home Sensor 區域後急停，動作完成。



c. 模式 5 (*wMode* = 5) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時開始減速至 *dfLowSpeed*；同時尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

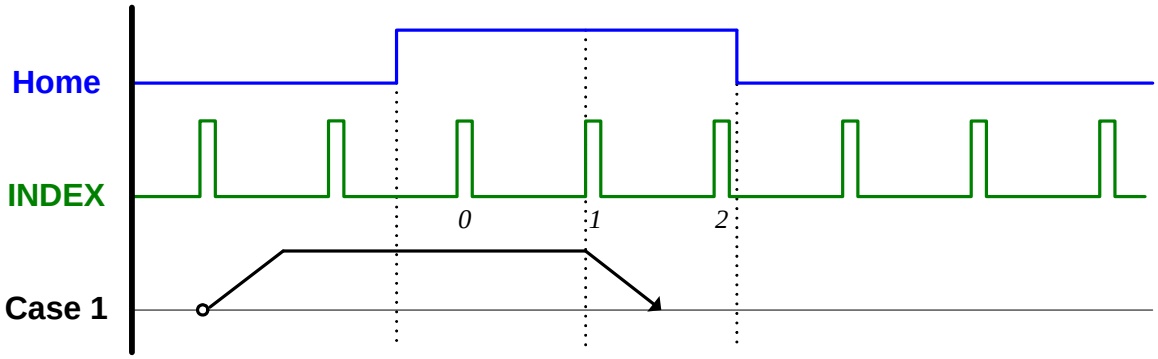
Step 2： 當觸發指定之 INDEX 後急停，動作完成。



d. 模式 6 (*wMode* = 6) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 2： 當觸發指定之 INDEX 後減速停止，動作完成。



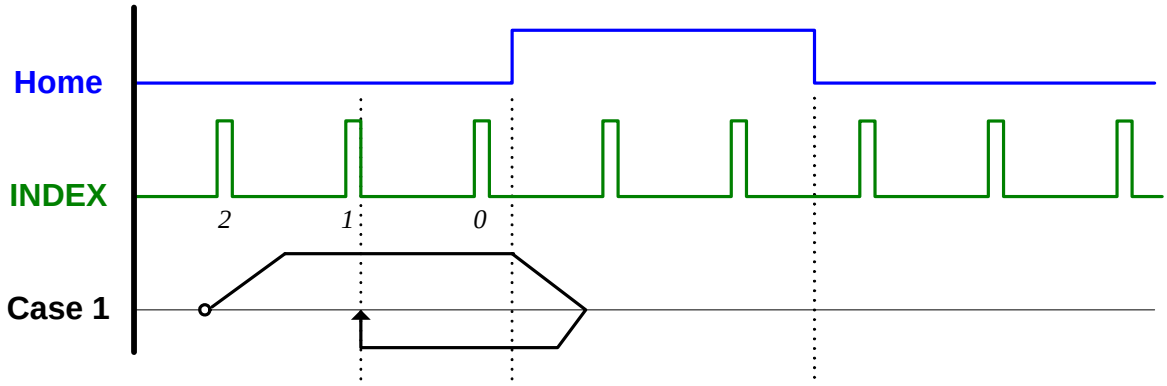
e. 模式 7 (*wMode* = 7) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時減速停止。

Step 2： 以 *dfLowSpeed* 速度往相反方向離開 Home Sensor 區域，

當離開後開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 3： 當觸發指定之 INDEX 後急停，動作完成。

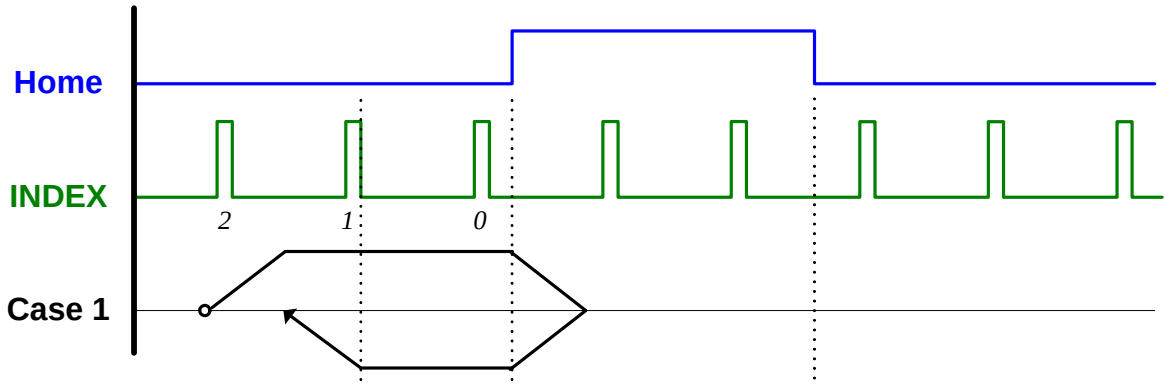


f. 模式 8 (*wMode* = 8) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時減速停止。

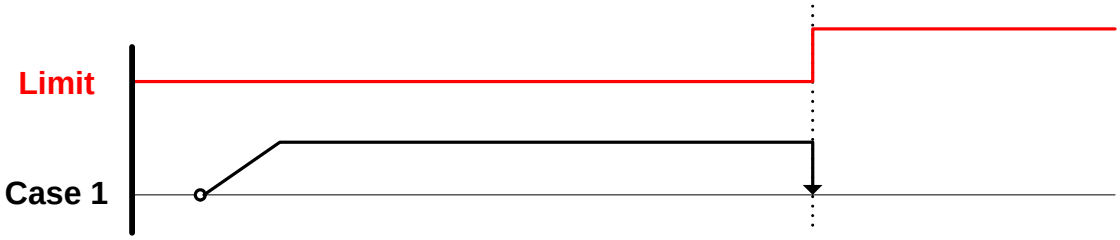
Step 2： 以 *dfHighSpeed* 速度往相反方向離開 Home Sensor 區域，當離開後開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 3： 當觸發指定之 INDEX 後減速停止，動作完成。



g. 模式 9 (*wMode* = 9) (此模式無 Case 2 和 Case 3)

以 *dfHighSpeed* 速度往指定的方向移動，當碰觸極限開關時急停，動作完成。

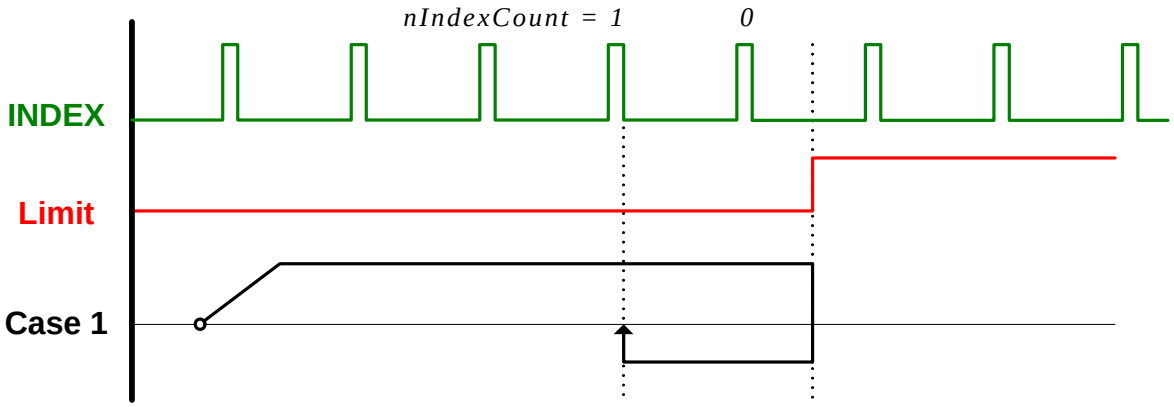


h. 模式 10 (*wMode* = 10) (此模式無 Case 2 和 Case 3)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當碰觸極限開關時急停。

Step 2： 以 *dfLowSpeed* 速度往相反方向移動，並開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 3： 當觸發指定之 INDEX 後急停，動作完成。

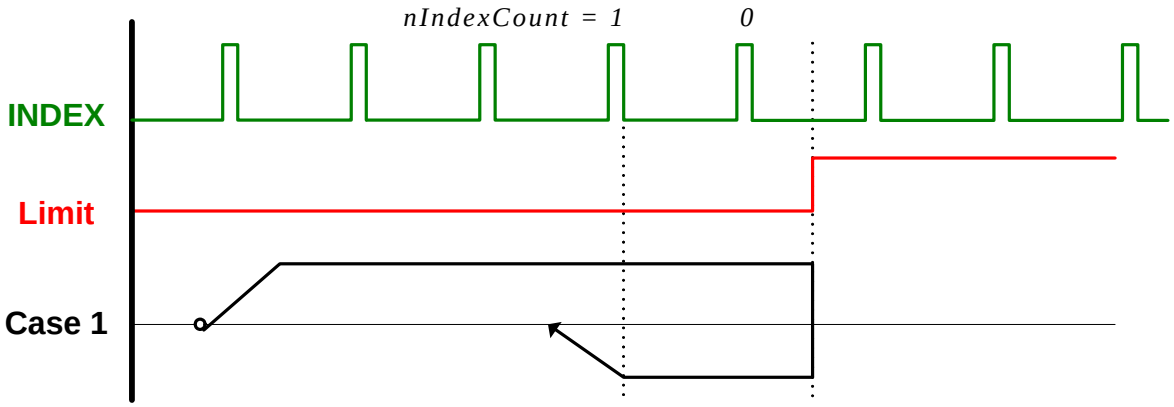


i. 模式 11 (*wMode* = 11) (此模式無 Case 2 和 Case 3)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當碰觸極限開關時急停。

Step 2： 以 *dfHighSpeed* 速度往相反方向移動，並開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 3： 當觸發指定之 INDEX 後減速停止，動作完成。

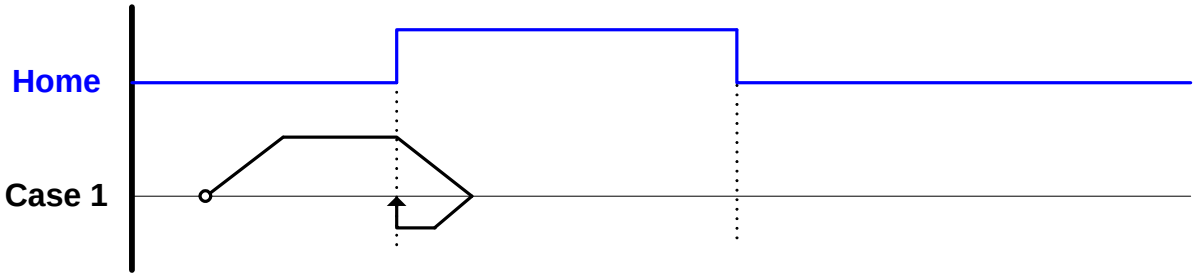


j. 模式 12 (*wMode* = 12) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時減速停止。

Step 2： 以 *dfLowSpeed* 速度往相反方向移動，以便離開 Home Sensor 區域。

Step 3： 一離開 Home Sensor 區域後急停，動作完成。

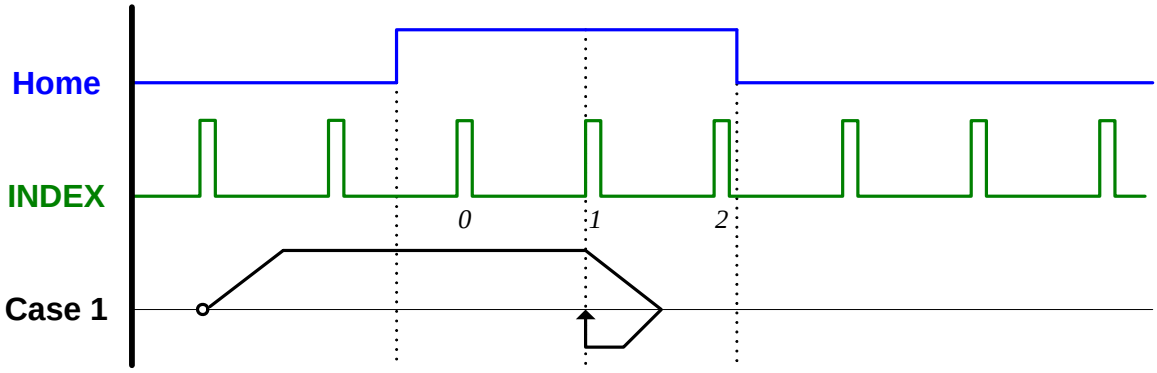


k. 模式 13 (*wMode* = 13) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 2： 當觸發指定之 INDEX 後減速停止。

Step 3： 以 *dfLowSpeed* 往相反方向回到觸發 INDEX 的位置，動作完成。



l. 模式 14 (*wMode* = 14) (下文僅說明 Case 1；Case 2 和 Case 3 請參考前面說明)

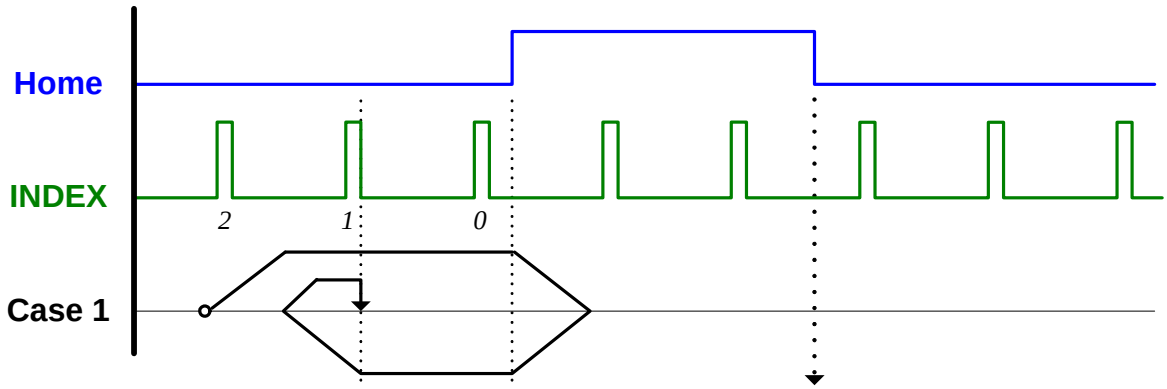
Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當進入 Home Sensor 區域時減速停止。

Step 2： 以 *dfHighSpeed* 速度往相反方向離開 Home Sensor 區域，

在離開 Home Sensor 區域後開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 3： 當觸發指定之 INDEX 後減速停止。

Step 4： 以 *dfLowSpeed* 往相反方向回到觸發 INDEX 的位置，動作完成。



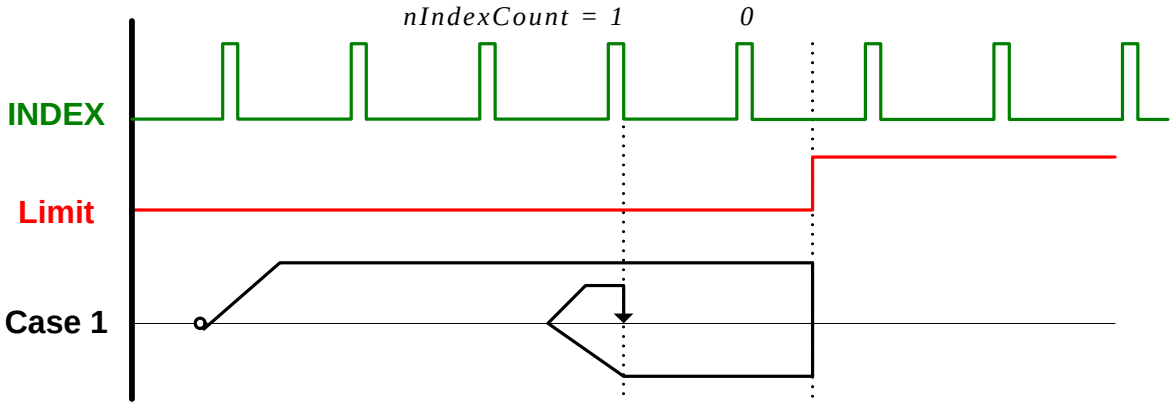
m. 模式 15 (*wMode* = 15) (此模式無 Case 2 和 Case 3)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當碰觸極限開關時急停。

Step 2： 以 *dfHighSpeed* 速度往相反方向移動，並開始尋找所指定編號的 INDEX（圖例設定尋找編號為 1 的 INDEX，也就是 *nIndexCount* = 1）。

Step 3： 當觸發指定之 INDEX 後減速停止。

Step 4： 以 *dfLowSpeed* 往相反方向回到觸發 INDEX 的位置，動作完成。

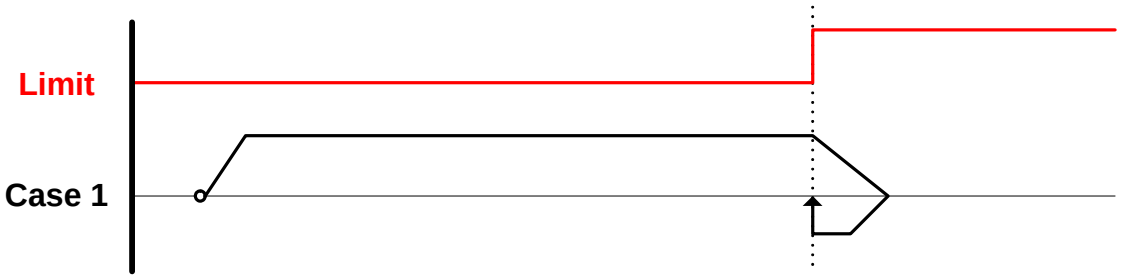


n. 模式 16 (*wMode* = 16) (此模式無 Case 2 和 Case 3)

Step 1： 以 *dfHighSpeed* 速度往指定的方向移動，當碰觸極限開關時減速停止。

Step 2： 以 *dfLowSpeed* 速度往相反方向移動，以便離開極限開關區域。

Step 3： 當離開極限開關區域時急停，動作完成。



2.8.2 啟動原點復歸動作

下面為啟動原點復歸動作的步驟。

1.需先使用 MCC_SetHomeConfig()設定原點復歸參數(請參考前面章節的說明)

2.呼叫 MCC_Home(

int *nOrder0*, int *nOrder1*, int *nOrder2*,
int *nOrder3*, int *nOrder4*, int *nOrder5*, ,
int *nOrder6*, int *nOrder7*, WORD *wCardIndex*)

其中

nOrder0 ~ nOrder7	各軸執行原點復歸動作的順序
wCardIndex	運動控制平台編號

各軸執行原點復歸動作的順序可設定為 0 ~ 7，設定值可重複。MCCL 將先對順序設定值為 0 的運動軸執行原點復歸動作，完成後再執行設定值為 1 的運動軸，依此原則完成所有運動軸的復歸動作。順序設定值如為 255 表示不對該運動軸執行原點復歸的動作。

在原點復歸過程中可以使用 MCC_AbortGoHome()停止復歸動作；也可以利用 MCC_GetGoHomeStatus()的函式傳回值獲知原點復歸的動作是否已經完成，若傳回值為 1 表示原點復歸動作已經完成，若為 0 表示原點復歸的動作尚在進行中。

注意

1. 不管採用任何模式，原點復歸過程皆可劃分為三個 Phase：
   Phase 1：尋找 Home Sensor 或極限開關的階段
   Phase 2：尋找所指定編號的 INDEX 訊號的階段
   Phase 3：從機械原點移動到邏輯原點的階段
2. 多軸同時執行原點復歸時，必須各軸皆完成 PHASE 1 動作後，才會一起進入 PHASE 2；同理，各軸皆應完成 PHASE 2 動作後，才會一起進入 PHASE 3；因此在原點復歸過程中可能出現某一軸已完成特定階段動作後，停止運動並等待其他軸完成同一階段的情況，此為正常現象。

下表列出各種原點復歸模式所包含的 Phase：

模式	Phase 1	Phase 2	Phase 3	說明
3	V		V	不需執行 Phase 2，但仍應等待各軸皆完成 Phase 2 動作後，才會一起執行 Phase 3。
4	V		V	同模式 3
5	V	V	V	
6	V	V	V	
7	V	V	V	
8	V	V	V	
9	V		V	同模式 3
10	V	V	V	
11	V	V	V	
12	V		V	同模式 3
13	V	V	V	
14	V	V	V	
15	V	V	V	
16	V		V	同模式 3

2.9 近端輸入接點與輸出接點(I/O)控制

近端輸入與輸出接點(Local I/O)是指內建於 IMP Series 運動控制平台上的 I/O 點，與需另外選購、最多可擴充至 512 個輸入接點與 512 個輸出接點的遠端控制模組(Remote I/O Module)不同。這些 I/O 接點均有專屬用途，但在實際應用如不需這些專屬用途(例如不需極限開關檢查功能或輸出 Servo-On/Off 訊號)，這些 I/O 接點均可作為一般的 I/O 使用。

2.9.1 輸入接點狀態

IMP Series 運動控制平台上內建的輸入接點包括：

a. 8 個 home sensor 訊號輸入接點，可以使用 MCC_GetHomeSensorStatus()讀取 home sensor 的輸入訊號。

b. 8 個正極限開關訊號輸入接點，與 8 個負極限開關訊號輸入接點，可以使用 MCC_GetLimitSwitchStatus()讀取極限開關的輸入訊號。

c. 1 個緊急停止開關訊號輸入接點，可以使用 MCC_GetEmgcStopStatus()讀取其輸入狀態。

2.9.2 訊號輸出控制

IMP Series 運動控制平台上內建的輸出接點包括：

a. 8 個 servo-on/off 訊號控制點，可以使用 MCC_SetServoOn()與 MCC_SetServoOff()輸出 servo-on/off 訊號。

b. 1 個 position ready 訊號控制點，可以使用 MCC_EnablePosReady() 與 MCC_DisablePosReady()輸出或取消 position ready 訊號。基於安全考量，通常在使用 MCC_InitSystem()成功啟動系統後，並確定系統運作正常後，需再額外使用 position ready 訊號來啟動系統外部電路(例如驅動器或馬達電路)。

c. 8 個 LED 顯示，可以讓使用者作為輸出燈號顯示。

2.9.3 輸入接點訊號觸發中斷服務函式

某些極限開關輸入接點的訊號能自動觸發使用者自訂的中斷服務函式(Interrupt Service Routine，ISR)。可以觸發 ISR 的極限開關包括：

a. IMP-2 共 24 點，包括：

Channel 0 Limit Switch +(OTP0)
Channel 1 Limit Switch +(OTP1)
Channel 2 Limit Switch +(OTP2)
Channel 3 Limit Switch +(OTP3)
Channel 4 Limit Switch +(OTP4)
Channel 5 Limit Switch +(OTP5)
Channel 6 Limit Switch +(OTP6)
Channel 7 Limit Switch +(OTP7)
Channel 0 Limit Switch - (OTN0)
Channel 1 Limit Switch - (OTN1)
Channel 2 Limit Switch - (OTN2)
Channel 3 Limit Switch - (OTN3)
Channel 4 Limit Switch - (OTN4)
Channel 5 Limit Switch - (OTN5)
Channel 6 Limit Switch - (OTN6)
Channel 7 Limit Switch - (OTN7)
Channel 0 Home Switch (HOME0)

Channel 1 Home Switch (HOME1)
Channel 2 Home Switch (HOME2)
Channel 3 Home Switch (HOME3)
Channel 4 Home Switch (HOME4)
Channel 5 Home Switch (HOME5)
Channel 6 Home Switch (HOME6)
Channel 7 Home Switch (HOME7)

使用”輸入接點觸發中斷服務函式”的步驟如下：

***Step 1*：使用 MCC_SetLIORoutineEx()串接自訂的中斷服務函式**

需先設計自訂的中斷服務函式，函式的宣告必須遵循下列的定義：

typedef void(_stdcall *_LIOISR_)(_LIOINT_*)

例如自訂的函式可設計如下：

```
stdcall MyLIOFunction(LIOINT *pstINTSource)
{
    //  判斷是否因碰觸到 Channel 0 Limit Switch +而觸發此函式
    if (pstINTSource->OTP0)
    {
        //  碰觸到 Channel 0 Limit Switch +時的處理程序
    }

    //  判斷是否因碰觸到 Channel 1 Limit Switch +而觸發此函式
    if (pstINTSource->OTP1)
    {
        //  碰觸到 Channel 1 Limit Switch +時的處理程序
    }
}
```

不可以使用”else if (pstINTSource->*OTP1*)”類似的語法，因 pstINTSource->*OTP0* 與 pstINTSource->*OTP1* 有可能同時不為 0。

接著使用 MCC_SetLIORoutineEx(MyLIOFunction)串接自訂的中斷服務函式。當自訂函式被觸發執行時，可以利用傳入自訂函式中、被宣告為 *LIOINT* 的 pstINTSource 參數，判斷此刻自訂函式被呼叫是因碰觸到哪一個輸入接點。*LIOINT* 的定義如下：

```
typedef struct _LIO_INT
{
  BYTE OTP0;
   BYTE OTP1;
   BYTE OTP2;
   BYTE OTP3;
   BYTE OTP4;
   BYTE OTP5;
   BYTE OTP6;
   BYTE OTP7;
   BYTE OTN0;
   BYTE OTN1;
   BYTE OTN2;
   BYTE OTN3;
   BYTE OTN4;
   BYTE OTN5;
   BYTE OTN6;
   BYTE OTN7;
   BYTE HOME0;
   BYTE HOME1;
   BYTE HOME2;
   BYTE HOME3;
   BYTE HOME4;
   BYTE HOME5;
   BYTE HOME6;
```

BYTE HOME7;

} *LIOINT*;

LIOINT 中各欄位的對應接點定義如下：

	IMP-2
OTP0	Channel 0 Limit Switch+
OTP1	Channel 1 Limit Switch+
OTP2	Channel 2 Limit Switch+
LDI3	Channel 3 Limit Switch+
LDI4	Channel 4 Limit Switch+
OTP5	Channel 5 Limit Switch+
OTP6	Channel 6 Limit Switch+
OTP7	Channel 7 Limit Switch+
OTN0	Channel 0 Limit Switch-
OTN1	Channel 1 Limit Switch-
OTN2	Channel 2 Limit Switch-
OTN3	Channel 3 Limit Switch-
OTN4	Channel 4 Limit Switch-
OTN5	Channel 5 Limit Switch-
OTN6	Channel 6 Limit Switch-
OTN7	Channel 7 Limit Switch-
HOME0	Channel 0 HOME Switch
HOME1	Channel 1 HOME Switch
HOME2	Channel 2 HOME Switch
HOME3	Channel 3 HOME Switch
HOME4	Channel 4 HOME Switch
HOME5	Channel 5 HOME Switch
HOME6	Channel 6 HOME Switch
HOME7	Channel 7 HOME Switch

這些欄位的值如果不為 0，表示該欄位的對應接點目前有訊號輸入；例如在 MyLIOFunction()中所輸入的參數 pstINTSource->*OTP2* 如果不為 0，表示碰觸到 Channel 2 Limit Switch +。

***Step 2*：使用 MCC_SetLIOTriggerType()設定觸發型態**

觸發型態可設定為上升緣(Rising Edge)觸發、下降緣(Falling Edge)觸發或是轉態(Level Change)觸發。MCC_SetLIOTriggerType()的輸入參數可為：

LIO_INT_NO	不觸發
LIO_INT_RISE	上升緣觸發(Default)
LIO_INT_FALL	下降緣觸發
LIO_INT_LEVEL	轉態觸發

***Step 3*：最後使用 MCC_EnableLIOTrigger()開啟”輸入接點訊號觸發中斷服務函式”功能。也可以使用 MCC_DisableLIOTrigger()關閉此項功能。**

2.10 編碼器(Encoder)控制

MCCL 所提供的編碼器控制功能包含回授倍率更動、計數值讀取、計數值閂鎖(Latch)、INDEX 觸發與計數值自動比較與觸發功能。

在使用編碼器的控制功能前需先正確設定機構參數中與編碼器特性有關的欄位，詳細內容請參考"**2.4.2 編碼器參數**"的說明。

2.10.1 一般控制

假使編碼器參數(請參考 2.4.2 一節)中的 *wType* 設定為 ENC_TYPE_AB，也就是輸入格式設定為 A/B Phase，則可以使用 MCC_SetENCInputRate()設定編碼器的回授倍率，可設定值為 1、2、4，分別表示回授倍率為×1、×2、×4。如機構參數中的 wCommandMode 設定為 OCM_VOLTAGE(使用 V Command)，並更改了回授倍率，則需重新設定機構參數中 *dwPPR* 欄位的設定值。而使用 MCC_GetENCValue()可以讀取編碼器的計數值。

2.10.2 計數值閂鎖(Latch)

MCCL 提供"計數值閂鎖"功能，使用者可以設定觸發訊號來源，這些觸發訊號被用來觸發將編碼器的計數值紀錄在閂鎖暫存器內的動作，可以使用 MCC_GetENCLatchValue()讀取閂鎖暫存器內的紀錄值。使用"計數值閂鎖"功能的步驟如下：

***Step 1*：使用 MCC_SetENCLatchSource()設定何種觸發訊號來源可觸發計數值閂鎖動作。**

下面的觸發來源皆可觸發將編碼器計數值紀錄在閂鎖暫存器內的動作，MCC_SetENCLatchSource()被用來設定觸發來源條件，設定時可同時取多個條件的聯集。這些觸發訊號來源包括：

ENC_TRIG_NO	沒有選擇任何觸發訊號源
ENC_TRIG_INDEX0	編碼器 Channel 0 的 Index 訊號
ENC_TRIG_INDEX1	編碼器 Channel 1 的 Index 訊號
ENC_TRIG_INDEX2	編碼器 Channel 2 的 Index 訊號
ENC_TRIG_INDEX3	編碼器 Channel 3 的 Index 訊號
ENC_TRIG_INDEX4	編碼器 Channel 4 的 Index 訊號
ENC_TRIG_INDEX5	編碼器 Channel 5 的 Index 訊號
ENC_TRIG_INDEX6	編碼器 Channel 6 的 Index 訊號
ENC_TRIG_INDEX7	編碼器 Channel 7 的 Index 訊號
ENC_TRIG_OTP0	發生近端輸入接點 OT0+中斷
ENC_TRIG_OTP1	發生近端輸入接點 OT1+中斷
ENC_TRIG_OTP2	發生近端輸入接點 OT2+中斷
ENC_TRIG_OTP3	發生近端輸入接點 OT3+中斷
ENC_TRIG_OTP4	發生近端輸入接點 OT4+中斷
ENC_TRIG_OTP5	發生近端輸入接點 OT5+中斷
ENC_TRIG_OTP6	發生近端輸入接點 OT6+中斷
ENC_TRIG_OTP7	發生近端輸入接點 OT7+中斷
ENC_TRIG_OTN0	發生近端輸入接點 OT0-中斷
ENC_TRIG_OTN1	發生近端輸入接點 OT1-中斷
ENC_TRIG_OTN2	發生近端輸入接點 OT2-中斷
ENC_TRIG_OTN3	發生近端輸入接點 OT3-中斷
ENC_TRIG_OTN4	發生近端輸入接點 OT4-中斷
ENC_TRIG_OTN5	發生近端輸入接點 OT5-中斷
ENC_TRIG_OTN6	發生近端輸入接點 OT6-中斷
ENC_TRIG_OTN7	發生近端輸入接點 OT7-中斷

如果使用

MCC_SetENCTriggerSource(ENC_TRIG_INDEX0 | ENC_TRIG_OTP0,

0, 0)

表示在輸入編碼器 Channel 0 的 Index 訊號與碰觸到 Channel 0 的正方向 Limit 時，皆會將編碼器計數值紀錄在第 0 個平台的第 0 個 Channel 的閂鎖暫存器內。

***Step 2*：使用 MCC_SetENCLatchType()設定計數值閂鎖模式**

在完成 ***Step 1***，接著使用 MCC_SetENCLatchType()設定閂鎖計數值的模式，可選擇的模式包括；

ENC_TRIG_FIRST　第一次滿足觸發條件時，計數值即被閂鎖並不再變動。
ENC_TRIG_LAST　當觸發條件滿足時即更新閂鎖計數值，次數不限。

***Step 3*：使用 MCC_GetENCLatchValue()讀取閂鎖暫存器內的紀錄值**

MCCL 並無函式可用來判斷閂鎖暫存器內的紀錄值是否被更新，但這些可更新閂鎖暫存器紀錄值的觸發來源皆可觸發中斷服務函式，使用者可搭配此項功能獲知紀錄值已被更新，並使用 MCC_GetENCLatchValue()讀取新的紀錄值，實際應用情況請參考”**IMP Series 運動控制函式庫範例手冊**”。

2.10.3 編碼器計數值觸發中斷服務函式

MCCL 提供的”編碼器計數值觸發中斷服務函式”功能可以對編碼器 Channel 0 ~ 7 設定比較值，在開啟所選定的 Channel 此項功能後，當該 Channel 的計數值等於所設定的比較值時，將自動呼叫使用者自訂的中斷服務函式。使用”編碼器計數值觸發中斷服務函式”功能的步驟如下：

***Step 1*：使用 MCC_SetENCRoutine()串接自訂的中斷服務函式**

需先設計自訂的中斷服務函式，函式的宣告必須遵循下列的定義：

typedef void(_stdcall *_ENCISR_)(_ENCINT_*)

例如自訂的函式可設計如下：

```
stdcall MyENCFunction(ENCINT *pstINTSource)
{

    //  判斷是否因編碼器 Channel 0 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP0)
    {
     //  滿足 Channel 0 比較值條件時的處理程序
     }

    //  判斷是否因編碼器 Channel 1 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP1)
    {
        //  滿足 Channel 1 比較值條件時的處理程序
    }

    //  判斷是否因編碼器 Channel 2 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP2)
    {
        //  滿足 Channel 2 比較值條件時的處理程序
    }
```

```
    //  判斷是否因編碼器 Channel 3 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP3)
    {
        //  滿足 Channel 3 比較值條件時的處理程序
    }

    //  判斷是否因編碼器 Channel 4 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP4)
    {
        //  滿足 Channel 4 比較值條件時的處理程序
    }

    //  判斷是否因編碼器 Channel 5 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP5)
    {
        //  滿足 Channel 5 比較值條件時的處理程序
    }
    //  判斷是否因編碼器 Channel 6 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP6)
    {
        //  滿足 Channel 6 比較值條件時的處理程序
    }

    //  判斷是否因編碼器 Channel 7 的計數值等於比較值而觸發此函式
    if (pstINTSource->COMP7)
    {
```

```
        //  滿足 Channel 7 比較值條件時的處理程序
    }

}
```

不可以使用"else if (pstINTSource->*COMP1*)"類似的語法，因 pstINTSource->*COMP0* 與 pstINTSource->*COMP1* 有可能同時不為 0。

接著使用 MCC_SetENCRoutine(MyENCFunction)串接自訂的中斷服務函式。當自訂函式被觸發執行時，可以利用傳入自訂函式中、被宣告為 *ENCINT* 的 pstINTSource 參數，判斷此刻自訂函式被呼叫是因滿足何種觸發條件。*ENCINT* 的定義如下：

```
typedef struct _ENC_INT
{
    BYTE    COMP0;
    BYTE    COMP1;
    BYTE    COMP2;
    BYTE    COMP3;
    BYTE    COMP4;
    BYTE    COMP5;
    BYTE    COMP6;
    BYTE    COMP7;
    BYTE    INDEX0;
    BYTE    INDEX1;
    BYTE    INDEX2;
    BYTE    INDEX3;
    BYTE    INDEX4;
    BYTE    INDEX5;
    BYTE    INDEX6;
    BYTE    INDEX7;
} ENCINT;
```

ENCIN 中的欄位值如果不為 0，表示自訂函式被呼叫的原因，各欄位所表示的觸發原因如下：

COMP0	編碼器 Channel 0 的計數值等於所設定的比較值
COMP1	編碼器 Channel 1 的計數值等於所設定的比較值
COMP2	編碼器 Channel 2 的計數值等於所設定的比較值
COMP3	編碼器 Channel 3 的計數值等於所設定的比較值
COMP4	編碼器 Channel 4 的計數值等於所設定的比較值
COMP5	編碼器 Channel 5 的計數值等於所設定的比較值
COMP6	編碼器 Channel 6 的計數值等於所設定的比較值
COMP7	編碼器 Channel 7 的計數值等於所設定的比較值
INDEX0	編碼器 Channel 0 的 Index 訊號所觸發
INDEX1	編碼器 Channel 1 的 Index 訊號所觸發
INDEX2	編碼器 Channel 2 的 Index 訊號所觸發
INDEX3	編碼器 Channel 3 的 Index 訊號所觸發
INDEX4	編碼器 Channel 4 的 Index 訊號所觸發
INDEX5	編碼器 Channel 5 的 Index 訊號所觸發
INDEX6	編碼器 Channel 6 的 Index 訊號所觸發
INDEX7	編碼器 Channel 7 的 Index 訊號所觸發

***Step 2*：使用 MCC_SetENCCompValue()設定指定 Channel 的編碼器計數值之比較值**

***Step 3*：使用 MCC_EnableENCCompTrigger()開啟指定 Channel 的”編碼器計數值觸發中斷服務函式”功能，也可以使用 MCC_DisableENCCompTrigger()關閉指定 Channel 此項功能。**

2.10.4 編碼器 INDEX 觸發中斷服務函式

MCCL 提供的”編碼器 Index 觸發中斷服務函式”功能可以利用編碼器 Channel 0 ~ 7 的 Index 訊號觸發使用者自訂的中斷服務函式。使用”編碼器 Index 觸發中斷服務函式”功能的步驟如下：

***Step 1*：使用 MCC_SetENCRoutine()串接自訂的中斷服務函式**

如未曾呼叫過 MCC_SetENCRoutine()，請參考前面對此步驟的說明；如已呼叫過 MCC_SetENCRoutine()，則只需在自訂的函式中加入對傳入參數(pstINTSource)”INDEX 訊號輸入”欄位(*INDEX0* ~ *INDEX7*)的判斷即可，請參考下例：

```
stdcall MyENCFunction(ENCINT *pstINTSource)
{
    //  判斷函式被呼叫是否由 Channel 0 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX0)
    {
     //  在編碼器 Channel 0 的 INDEX 訊號輸入時的處理程序
    }

    //  判斷函式被呼叫是否由 Channel 1 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX1)
    {
          //  在編碼器 Channel 1 的 INDEX 訊號輸入時的處理程序
    }

    //  判斷函式被呼叫是否由 Channel 2 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX2)
    {
          //  在編碼器 Channel 2 的 INDEX 訊號輸入時的處理程序
    }
```

```
    //  判斷函式被呼叫是否由 Channel 3 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX3)
    {
        //  在編碼器 Channel 3 的 INDEX 訊號輸入時的處理程序
    }

    //  判斷函式被呼叫是否由 Channel 4 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX4)
    {
        //  在編碼器 Channel 4 的 INDEX 訊號輸入時的處理程序
    }

    //  判斷函式被呼叫是否由 Channel 5 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX5)
    {
        //  在編碼器 Channel 5 的 INDEX 訊號輸入時的處理程序
    }

    //  判斷函式被呼叫是否由 Channel 6 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX6)
    {
        //  在編碼器 Channel 6 的 INDEX 訊號輸入時的處理程序
    }

    //  判斷函式被呼叫是否由 Channel 7 的 INDEX 訊號所觸發
    if (pstINTSource->INDEX7)
    {
        //  在編碼器 Channel 7 的 INDEX 訊號輸入時的處理程序
    }

}
```

***Step 2*：使用 MCC_EnableENCIndexTrigger()開啟指定 Channel 的編碼器 INDEX 觸發功能，也可以使用 MCC_DisableENCIndexTrigger()關閉此項功能。**

可以搭配”編碼器計數值閂鎖”功能獲得 INDEX 訊號輸入時的計數器的計數值。(” 編碼器計數值閂鎖”功能請參考前面章節的說明)。而目前 motor 的位置是否正位於編碼器的 INDEX 上，可以利用 MCC_GetENCIndexStatus()來判斷。

2.11 類比電壓輸出(D/A Converter，DAC)控制

假使被要求輸出電壓的運動軸在機構參數中已被規劃為 V Command 運動軸(也就是將 *nCommandMode* 設定為 OCM_VOLTAGE)，則無法使用下面所討論與 DAC 有關的所有函式，在呼叫這些函式後也將獲得錯誤的函式傳回值，此點需特別注意。

2.11.1 一般控制

在使用 MCC_InitSystem() 啟動 MCCL 後即可使用 MCC_SetDACOutput()輸出類比電壓，電壓輸出範圍為-10V ~ +10V。

另外，可以使用 MCC_StopDACConv()停止類比電壓輸出功能，也可以使用 MCC_StartDACConv()重新開啟此項功能。

2.11.2 輸出電壓硬體觸發模式

MCCL 提供的"輸出電壓硬體觸發模式"功能可針對選定的 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。此項功能在規劃後直接由硬體處理，因此保有最佳的即時特性。使用"輸出電壓硬體觸發模式"的步驟如下：

***Step 1*：使用 MCC_SetDACTriggerOutput()預先規劃輸出電壓值**

例如使用 MCC_SetDACTriggerOutput(2.0, 1, 0)對第 0 張平台的 DAC Channel 1 預先規劃 2.0 V 的輸出電壓。

***Step 2*：使用 MCC_SetDACTriggerSource()設定硬體觸發來源**

可設定的硬體觸發來源定義如下，並可同時設定多種觸發條件的聯集。需注意這些硬體觸發來源需皆來自相同的運動控制平台上。

1. DAC_TRIG_ENC0 編碼器 Channel 0 特定計數值
2. DAC_TRIG_ENC1 編碼器 Channel 1 特定計數值
3. DAC_TRIG_ENC2 編碼器 Channel 2 特定計數值
4. DAC_TRIG_ENC3 編碼器 Channel 3 特定計數值
5. DAC_TRIG_ENC4 編碼器 Channel 4 特定計數值
6. DAC_TRIG_ENC5 編碼器 Channel 5 特定計數值
7. DAC_TRIG_ENC6 編碼器 Channel 6 特定計數值
8. DAC_TRIG_ENC7 編碼器 Channel 7 特定計數值

在設定硬體觸發來源時也應啟動與這些硬體觸發來源有關的中斷服務功能，如此這些硬體觸發來源才能觸發輸出電壓，例如使用 MCC_SetDACTriggerSource(*DAC_TRIG_ENC0*, 1, 2)設定第 2 張平台的 Channel 1 之 DAC 硬體觸發來源為第 2 張平台 Channel 0 編碼器的特定計數值，此時也應開啟 Channel 0 的”編碼器計數值觸發中斷服務函式”功能，也就是需使用 MCC_SetENCCompValue() 與 MCC_EnableENCCompTrigger()函式啟動 Channel 0 的編碼器中斷服務功能，此項功能請參考”**2.10.3 編碼器計數值觸發中斷服務函式**”此章節；同樣地，硬體觸發來源如設定 Limit Switch 的訊號，也應使用 MCC_SetLIOTriggerType()與 MCC_EnableLIOTrigger()函式啟動輸入接點訊號觸發中斷服務功能，此項功能請參考” **2.9.3 輸入接點訊號觸發中斷服務函式**”此章節。

***Step 3*：使用 MCC_EnableDACTriggerMode()開啟此項功能，也可以使用 MCC_DisableDACTriggerMode()關閉此項功能。**

2.12 類比電壓輸入(A/D Converter，ADC)控制

2.12.1 初始設定

IMP-Series 在使用”類比電壓輸入控制”功能前需先完成下列步驟：

***Step 1*：使用 MCC_SetADCConvType()設定電壓轉換型式**

(1) 使用 MCC_SetADCConvType(*ADC_TYPE_BIP*)表示使用雙極性轉換型式(Bipolar Converter Type)，可讀取的電壓範圍為-5V ~ 5V。

(2) 使用 MCC_SetADCConvType(*ADC_TYPE_UNI*)表示使用單極性轉換型式(Unipolar Converter Type)，可讀取的電壓範圍為 0V ~ 10V。

***Step 2*：使用 MCC_SetADCConvMode()設定電壓轉換模式**

(1) 使用 MCC_SetADCConvMode(*ADC_MODE_FREE*)表示進行連續的電壓值讀取動作，所讀取的電壓值將隨不同的輸入電壓而改變，此函式需搭配 MCC_EnableADCConvChannel()使用，此項功能請參考”2.12.2 連續電壓轉換”。

(2) 使用 MCC_SetADCConvMode(*ADC_MODE_SINGLE*)表示僅進行一次電壓值讀取動作，除非再次呼叫 MCC_StartADCConv()，否則讀取值不再變動，此函式需搭配 MCC_SetADCSingleChannel()使用，此項功能請參考”2.12.3 單一 Channel 電壓轉換”。

2.12.2 連續電壓轉換

在完成前面的初始化設定後，如要讀取特定 Channel 的輸入電壓值，進行步驟如下：

***Step 1*： 呼叫 MCC_SetADCConvMode(*ADC_MODE_FREE*)**

***Step 2*：使用 MCC_EnableADCConvChannel()允許選定的 Channel 輸入類比電壓**

最多可同時允許 8 組 A/D Channel 輸入類比電壓，電壓轉換僅在允許輸入的 Channel 中輪替，也可以使用 MCC_DisableADCConvChannel()禁止選定的 Channel 輸入類比電壓。

***Step 3*：使用 MCC_StartADCConv()啟動類比電壓輸入功能，也可以使用 MCC_StopADCConv()停止類比電壓輸入功能。**

***Step 4*：使用 MCC_GetADCInput()讀取輸入電壓值**

2.12.3 單一 Channel 電壓轉換

MCCL 提供的 MCC_SetADCSingleChannel()函式可選定某一個 Channel 為唯一可進行輸入電壓轉換的 Channel，而其他 Channel 則停止轉換功能。

先利用 MCC_SetADCSingleChannel()選定 Channel，並呼叫 MCC_SetADCConvMode(*ADC_MODE_SINGLE*) 使用單次轉換模式，則在呼叫 MCC_StartADCConv()後，此選定的 Channel 會將電壓值轉換一次。轉換完成後即不再進行轉換，使用者需再次呼叫 MCC_StartADCConv()才會進行下一單次的轉換。轉換期間(約 8μs)可經由 MCC_GetADCWorkStatus()確認轉換動作是否完成。確認轉換動作完成後，可**使用 MCC_GetADCInput()讀取輸入電壓值。**

2.12.4 特定電壓值觸發中斷服務函式

MCCL 所提供的”特定電壓值觸發中斷服務函式”功能可以對選定的 ADC Channel 設定電壓比較值，當開啟此項功能並滿足觸發條件後，將自動呼叫使用者自訂的中斷服務函式。使用”特定電壓值觸發

中斷服務函式”功能的步驟如下：

***Step 1*：使用 MCC_SetADCRoutine()串接自訂的中斷服務函式**

需先設計自訂的中斷服務函式，函式的宣告必須遵循下列的定義：

typedef void(_stdcall **ADCISR*)(*ADCINT)**

例如自訂的函式可設計如下：

```
_stdcall MyADCFunction(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
     }

    //  判斷是否因 ADC Channel 1 的電壓值滿足比較條件而觸發此函式
    if (pstINTSource->COMP1)
    {
        //  滿足 Channel 1 比較值條件時的處理程序
    }
}
```

不可以使用”else if (pstINTSource->*COMP1*)”類似的語法，因 pstINTSource->*COMP0* 與 pstINTSource->*COMP1* 有可能同時不為 0。

接著使用 MCC_SetADCRoutine(MyADCFunction)串接自訂的中斷服務函式。當自訂函式被觸發執行時，可以利用傳入自訂函式中、被宣告為 *ADCINT* 的 pstINTSource 參數，判斷此刻自訂函式被呼叫是

因滿足何種觸發條件。*ADCINT* 的定義如下：

```
typedef struct _ADC_INT
{
    BYTE    COMP0;
    BYTE    COMP1;
    BYTE    COMP2;
    BYTE    COMP3;
    BYTE    COMP4;
    BYTE    COMP5;
    BYTE    COMP6;
    BYTE    COMP7;
    BYTE    CONV;
    BYTE    TAG;
} ADCINT;
```

ADCINT 中的欄位值如果不為 0，表示自訂函式被呼叫的原因，各欄位所表示的觸發原因如下：

COMP0	ADC Channel 0 的電壓值滿足觸發條件
COMP1	ADC Channel 1 的電壓值滿足觸發條件
COMP2	ADC Channel 2 的電壓值滿足觸發條件
COMP3	ADC Channel 3 的電壓值滿足觸發條件
COMP4	ADC Channel 4 的電壓值滿足觸發條件
COMP5	ADC Channel 5 的電壓值滿足觸發條件
COMP6	ADC Channel 6 的電壓值滿足觸發條件
COMP7	ADC Channel 7 的電壓值滿足觸發條件
CONV	ADC 的任意 Channel 完成電壓轉換
TAG	ADC 的標籤 Channel 完成電壓轉換(在同一時間只允許一個特定 Channel 被貼上標籤)

***Step 2*：參考前面的說明完成”初始設定”**

***Step 3*：使用 MCC_SetADCCompValue()設定電壓比較器之比較值**

***Step 4*：使用 MCC_SetADCCompMask()設定電壓遮蔽位元**

當輸入電壓與設定之比較值比較時，可遮蔽最小幾個 bit 不做比較動作，如此可降低比較器的靈敏度，避免因輸入電壓跳動造成中斷持續發生。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

***Step 5*：使用 MCC_SetADCCompType()設定電壓比較模式**

電壓比較模式用來設定觸發中斷的條件，可設定的電壓比較模式包括：

ADC_COMP_RISE	ADC 輸入電壓由小到大，並通過電壓比較值
ADC_COMP_FALL	ADC 輸入電壓由大到小，並通過電壓比較值
ADC_COMP_LEVEL	ADC 輸入電壓值改變，並通過電壓比較值

***Step 6*：使用 MCC_EnableADCCompTrigger()開啟此項功能**

***Step 7*：搭配使用”連續電壓轉換”或”單一 Channel 電壓轉換”功能**

2.12.5 電壓轉換完成觸發中斷服務函式

MCCL 提供的"電壓轉換完成觸發中斷服務函式”功能分為兩種，說明如下：

I. 任意 ADC Channel 完成電壓轉換動作後觸發中斷服務函式，此項功能的使用步驟如下：

***Step 1*：使用 MCC_SetADCRoutine()串接自訂的中斷服務函式**

如未曾呼叫過 MCC_SetADCRoutine()，請參考前面對此步驟的說明；如已呼叫過 MCC_SetADCRoutine()，則只需在自訂的函式中加入對傳入參數(pstINTSource)”電壓轉換完成”欄位(*CONV*)的判斷即可，請參考下例：

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

***Step 2*：使用 MCC_EnableADCConvTrigger()開啟此項功能，也可以使用 MCC_DisableADCConvTrigger()關閉此項功能。**

II. ADC 標籤 Channel 完成電壓轉換動作後觸發中斷服務函式，此項功能的使用步驟如下：

***Step 1*：使用 MCC_SetADCRoutine()串接自訂的中斷服務函式**

如未曾呼叫過 MCC_SetADCRoutine()，請參考前面對此步驟的說明；如已呼叫過 MCC_SetADCRoutine()，則只需在自訂的函式中加入對傳入參數(pstINTSource)”標籤 Channel 完成電壓轉換”欄位(*TAG*)的判斷即可，請參考下例：

```
_stdcall MyADCFunction(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
     }
}
```

***Step 2*：使用 MCC_SetADCTagChannel()選定標籤 Channel**

***Step 3*：使用 MCC_EnableADCTagTrigger()開啟此項功能，也可以使用 MCC_DisableADCTagTrigger()關閉此項功能。**

2.13 計時器(Timer)與Watch Dog控制

2.13.1 計時終了觸發中斷服務函式

利用 MCCL 可以設定 IMP Series 運動控制平台上 32 bits 計時器的計時時間之長度，在啟動計時功能並在計時終了時(也就是計時器的計時值等於設定值)，將觸發使用者自訂的中斷服務函式，並重新開始計時，此過程將持續至關閉此項功能為止。使用"計時終了觸發中斷服務函式"功能的步驟如下：

***Step 1*：使用 MCC_SetTMRRoutine()串接自訂的中斷服務函式**

如未曾呼叫過 MCC_SetTMRRoutine()；如已呼叫過 MCC_SetTMRRoutine()，則只需在自訂的函式中加入對傳入參數(pstINTSource)"計時器計時終了"欄位(*TIMER*)的判斷即可，請參考下例：

```
stdcall MyTMRFunction(TMRINT *pstINTSource)
{
    //   判斷是否因計時器計時終了而觸發此函式
    if (pstINTSource->TIMER)
    {
     //   計時器計時終了時的處理程序
     }
}
```

***Step 2*：使用設定 MCC_SetTimer()計時器之計時時間，計時單位為 System Clock(10ns)**

***Step 3*：使用 MCC_EnableTimerTrigger()開啟"計時終了觸發中斷服務函式"功能，也可以使用 MCC_DisableTimerTrigger()關閉此功能。**

***Step 4*：使用 MCC_EnableTimer()開啟計時器計時功能，也可以使用 MCC_DisableTimer()關閉此功能。**

2.13.2 Watch Dog 控制

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 的計時值等於設定的比較值前)，使用 MCC_RefreshWatchDogTimer()清除 watch dog 的計時內容。否則一但 watch dog 的計時值等於設定的比較值時，將發生 reset 硬體的動作。使用 watch dog 的步驟如下：

***Step 1*：使用設定 MCC_SetTimer()計時器之計時時間，計時長度為 32bit 數值，計時單位為 System Clock(10ns)。**

***Step 2*：MCC_SetWatchDogTimer()設定 watch dog 計時器長度**

Watch dog 計時器長度為 32-bit 數值，使用計時器之計時時間作為 time base。也就是說如果使用下列的程式碼：

```
MCC_SetTimer(1000000, 0);
MCC_SetWatchDogTimer(2000, 0);
```

此時表示第 0 張平台的 watch dog 計時器的比較值設定為(10ns × 1000000)× 2000 = 20s。

***Step 3*：使用 MCC_SetWatchDogResetPeriod()設定 reset 訊號持續時間**

可透過本函式可規劃因 watch dog 功能所產生 Reset 硬體動作的訊號持續時間(32-bit 數值，最大值為 4294967296，設定單位為 system clock(10ns)。

***Step 4*：使用 MCC_EnableTimer()開啟計時器計時功能**

***Step 5*：必須在 watch dog 計時終了前，使用 MCC_RefreshWatchDogTimer()清除 watch dog 的計時內容。**

使用者可搭配”計時終了觸發中斷服務函式”功能，在 Watch Dog Reset 硬體動作前先以警示，並在計時中斷服務函式內進行必要的處理。

2.14 Remote I/O控制

2.14.1 初始設定

每張 IMP-Series 運動控制平台則擁有 1 個 Remote I/O 平台的接頭，稱為 Remote I/O Master 端，可控制 32 張 Remote I/O 平台(編號 IMP-ARIO，稱為 Remote I/O Slave 端)。如下圖所示，每張 Remote I/O 卡各提供 16 個輸出接點與 16 個輸入接點。

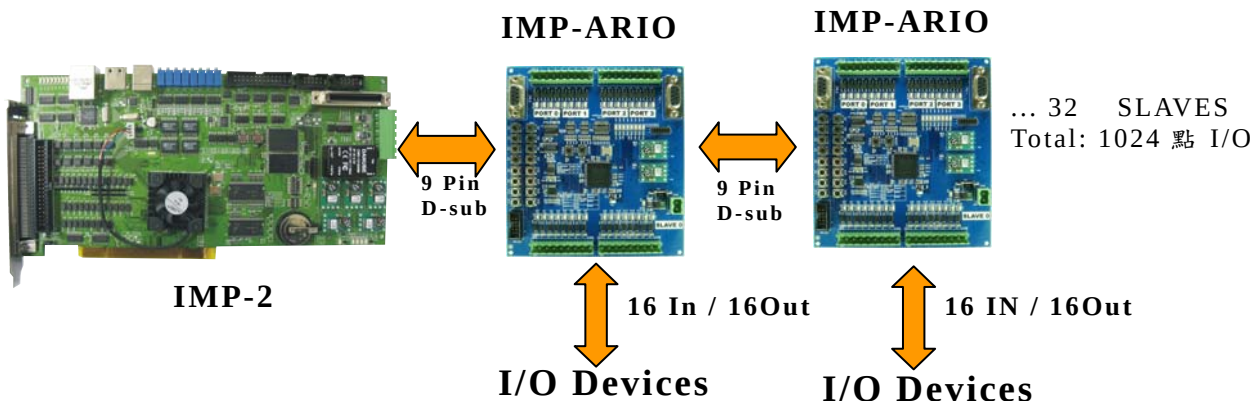


Figure 2.14.1 Remote Master 與 Slave

使用 MCC_EnableARIOSlaveControl()啟動資料傳輸功能，下面為使用範例。

MCC_EnableARIOSlaveControl(WORD wSet, WORD wSlave, WORD wCardIndex=0);

2.14.2 設定與讀取輸出、入點狀態

在完成初始設定後，即可使用 MCC_GetARIOInputValue()讀取輸入接點的訊號狀態；也可使用 MCC_SetARIOOutputValue()設定輸出接點的訊號狀態。MCC_GetARIOInputValue()的函式原型如下：

MCC_GetARIOInputValue(WORD* pwValue,

```
                                WORD    wSet,
                                WORD    wSlave,
                                WORD    wCardIndex);
```

Remote I/O 讀取功能可讀取 16/slave 個輸入點的狀態，*pwValue 的 bit 0 ~ bit 15 分別存放 Input 0 ~ Input 15 輸入點的狀態。參數 *wSet* 目前無意義，*wSlave* 是用來指定 Remote IO 卡要讀取的僕端。

Remote I/O 寫入功能可寫入 16/slave 點。因此 MCC_SetARIOOutputValue() 函式的使用方式與 MCC_GetARIOInputValue()類似，每次在設定輸出點狀態時，需同時設定指定組別的 16 個輸出點的狀態。MCC_SetARIOOutputValue()的函式原型如下：

```
MCC_SetARIOOutputValue( WORD    wValue,
                        WORD    wSet,
                        WORD    wSlave,
                        WORD    wCardIndex);
```

其中 wValue 所指定 16 個輸出點的狀態。

3. A+ PC 模式編譯環境

3.1 使用 Visual C++

Including Files

MCCL.h

MCCL_Fun.h

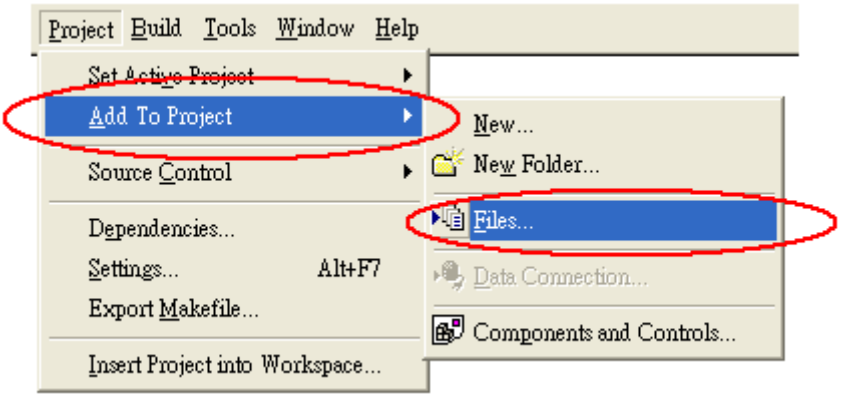
Import Library **(使用者需將此檔加入 Project 中)**

MCCLPCI_IMP.lib (for A+ PC 模式)

下面為使用 IMP Series 平台(IMP-Series)，並使用 VC++為發展工具，將加所需的 import library 也就是 MCCLPCI_IMP.lib 加入 project 的過程：

步驟一：

使用**[Project]**下的**[Add To Project]**



步驟二：

選取 MCCLPCI_IMP.lib 加入 project

3.2 使用 Visual Basic

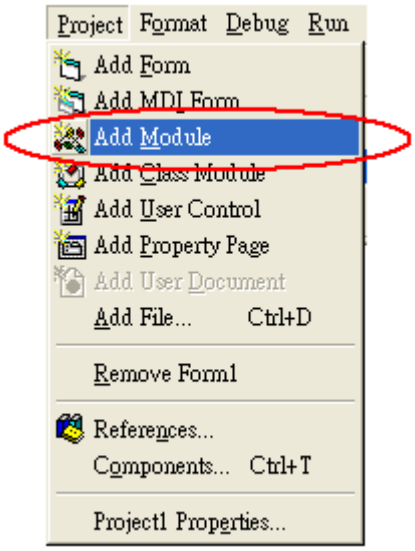
Including Files

MCCLPCI_IMP.bas (for A+ PC 模式)

下面為使用 IMP Series PCI 平台(IMP-Series)，並使用 VB 為發展工具，將所需的 import module 也就是 MCCLPCI_IMP.bas 加入 project 的過程

步驟一

使用[Project]下[Add]->[Module]



步驟二

選取 MCCLPCI_IMP.bas 加入 module 之中