IMP Series

運動控制函式庫

範例手冊

版本 ：V.1.01

日期 ：2013.01

http://www.epcio.com.tw

目 錄

1. 運動控制函式庫範例內容說明

安裝光碟中所提供的範例皆為 console mode 類型，使用者可將這些範例整合到自己的應用程式中。MCCL 最多能支援 6 張 IMP Series 運動控制卡與 48 個 groups，但為了增加這些範例的可讀性，大部分的範例只使用 1 張運動控制卡(運動控制卡編號使用 ***CARD_INDEX*** 來表示)與 1 個 group(group 編號使用 ***g_nGroupIndex*** 來表示)。

2. GROUP 參數與機構、編碼器參數設定

相關函式

MCC_SetSysMaxSpeed()

MCC_GetSysMaxSpeed()

MCC_SetMacParam()

MCC_GetMacParam()

MCC_SetEncoderConfig()

MCC_CloseAllGroups()

MCC_CreateGroup()

MCC_UpdateParam()

範例程式

InitSys.cpp

內容說明

本範例說明 group、機構與編碼器參數的設定過程。先使用 MCC_SetSysMaxSpeed()設定進給速度的上限，接著使用 MCC_SetMacParam()與 MCC_SetEncoderConfig() 設定各軸的機構與編碼器參數，最後再使用 MCC_CreateGroup()建立一新的 group。

有關 group 使用方式與機構參數更詳細的說明請參考”**IMP Series 運動控制函式庫使用手冊**”。

3. 插值時間調整

相關函式

MCC_InitSystem()

MCC_GetCurPulseStockCount()

MCC_SetMaxPulseStockNum()

範例程式

CheckHWStock.cpp

內容說明

較小的插值時間擁有較佳的運動控制性能，插值時間可設定的最小值為 1ms。為了減少因使用 FIFO 造成的命令延遲，可以使用 MCC_SetMaxPulseStockNum()設定 FIFO 的使用個數。為了獲得最適當的插值時間，可以使用 MCC_GetCurPulseStockCount()讀取 IMP Series 運動控制卡上的 pulse 庫存筆數。在持續運動過程中 pulse 庫存筆數必須等於設定的最大 FIFO 使用個數，才能保證穩定的運動性能。若庫存筆數出現等於 0 的現象，則必須延長插值時間(插值時間為 MCC_InitSystem()所需的參數之一)。另外，若人機操作畫面的顯示出現遲滯的現象，也必須延長插值時間。

4. 啟動與結束運動控制函式庫

相關函式

MCC_InitSystem()

MCC_CloseSystem()

MCC_GetMotionStatus()

範例程式

InitSys.cpp

內容說明

本範例說明在完成 group 參數與機構參數的設定後，使用 MCC_InitSystem() 啟動運動控制函式庫，所需的參數請參考”**IMP Series 運動控制函式庫使用手冊**”。下面說明範例的內容。

Step 1：給定控制卡硬體參數

```
SYS_CARD_CONFIG    stCardConfig[MAX_CARD_NUM];
..
stCardConfig[CARD_INDEX].wCardType      = wCardType;
```

Step 2：啟動運動控制函式庫

```
nRet = MCC_InitSystem( INTERPOLATION_TIME,  //   插值補間時間設為  2 ms
                       stCardConfig,        //   硬體參數
                       1);                  //   只使用  1  張  IMP

if (nRet == NO_ERR)//   啟動運動控制函式庫成功
{
    /*
    使用者可於此執行其他初始化的動作，例如設定位移單位、進給
    速度。
    */
}
```

Step 3：

MCC_CloseSystem() 被使用來關閉 MCCL 與驅動程式函式庫，兩種方式可用來關閉系統：

i. 全部運動命令執行完成才關閉系統

需檢查系統是否處於停止狀態，MCC_GetMotionStatus()的函式傳回值若為 GMS_STOP，則系統處於停止狀態。

```
while ((nRret = MCC_GetMotionStatus(g_nGroupIndex)) != GMS_STOP)
{
MCC_TimeDelay(1);   //    Sleep 1 ms
//    因使用”while’’命令，為避免系統鎖死，影響系統的操作
//    需呼叫 MCC_TimeDelay ()釋放 CPU 的使用權。
}

MCC_CloseSystem();          //    結束 MCCL 與驅動程式函式庫
```

ii. 直接結束運動控制庫

只需呼叫 MCC_CloseSystem()即可，系統將立刻停止運作。

5. 系統狀態設定

相關函式

MCC_SetAbsolute()
MCC_SetIncrease()
MCC_GetCoordType()
MCC_SetAccType()
MCC_GetAccType()
MCC_SetDecType()
MCC_GetDecType()
MCC_SetPtPAccType()
MCC_GetPtPAccType()
MCC_SetPtPDecType()
MCC_GetPtPDecType()
MCC_SetServoOn()
MCC_SetServoOff()
MCC_EnablePosReady()
MCC_DisablePosReady()

範例程式

SetStatus.cpp

內容說明

此範例說明如何改變系統狀態。未特別設定系統狀態，系統將使用預設狀態運作，系統的預設狀態可參閱"**IMP Series 運動控制函式庫參考手冊**"。下面說明函式的內容。

```
MCC_SetAbsolute(g_nGroupIndex);   //  使用絕對座標型態表示各軸位置

//  使用'T'型曲線為直線、圓弧、圓運動的加速型式
MCC_SetAccType('T', g_nGroupIndex);

//  使用'S'型曲線為直線、圓弧、圓運動的減速型式
```

```
MCC_SetDecType('S' , g_nGroupIndex);

//  使用'T'型曲線為點對點運動的加速型式
MCC_SetPtPAccType('T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', g_nGroupIndex);

//  使用'S'型曲線為點對點運動的減速型式
MCC_SetPtPDecType('S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', g_nGroupIndex);

MCC_SetServoOn(0, CARD_INDEX); //  啟動第 0 軸伺服系統

//  開啟 Position Ready 輸出接點功能
MCC_EnablePosReady(CARD_INDEX);
```

啟動伺服系統需呼叫 MCC_SetServoOn()，系統才能正常運作；是否需呼叫 MCC_EnablePosReady()視實際情況而定。

6. 讀取運動速度、座標與運動命令資訊

相關函式

MCC_GetCurFeedSpeed()
MCC_GetFeedSpeed()
MCC_GetCurPos()
MCC_GetPulsePos()
MCC_GetCurCommand()
MCC_GetCommandCount()

範例程式

GetStatus.cpp

內容說明

MCC_GetCurFeedSpeed()用來讀取目前的進給速度，MCC_GetSpeed()則可以用來讀取目前各軸的進給速度。MCC_GetCurPos()用來讀取各軸目前位置之直角座標值，MCC_GetPulsePos()則用來讀取各軸目前位置之馬達座標值(或稱為 pulse 座標值)。直角座標值與馬達座標值可以利用機構參數換算而得，也就是馬達座標值 ＝直角座標值 × (dfGearRatio / dfPitch) × dwPPR。使用 MCC_GetCurPos()與 MCC_GetPulsePos()所讀取之各軸座標值，只有在該軸有實際對應至硬體輸出 Channel 時才有意義。

下面為使用範例：

Step 1：宣告變數

```
double  dfCurPosX, dfCurPosY, dfCurPosZ, dfCurPosU, dfCurPosV, dfCurPosW,
dfCurPosA, dfCurPosB, dfCurSpeed;
double  dfCurSpeedX, dfCurSpeedY, dfCurSpeedZ, dfCurSpeedU, dfCurSpeedV,
dfCurSpeedW, dfCurSpeedA, dfCurSpeedB;
long    lCurPulseX, lCurPulseY, lCurPulseZ, lCurPulseU, lCurPulseV, lCurPulseW,
lCurPulseA, lCurPulseB;
```

Step 2：讀取目前的進給速度

```
dfCurSpeed = MCC_GetCurFeedSpeed(g_nGroupIndex);
```

Step 3：讀取目前各軸的進給速度

```
MCC_GetSpeed(   &dfCurSpeedX, &dfCurSpeedY, &dfCurSpeedZ, &dfCurSpeedU,
                &dfCurSpeedV, &dfCurSpeedW, &dfCurSpeedA, &dfCurSpeedB,
                g_nGroupIndex);
```

Step 4：讀取各軸目前位置之直角座標值

```
MCC_GetCurPos( &dfCurPosX,   &dfCurPosY,   &dfCurPosZ,   &dfCurPosU,
               &dfCurPosV,   &dfCurPosW,   &dfCurPosA,   &dfCurPosB,
               g_nGroupIndex);
```

Step 5：讀取各軸目前位置之馬達座標值

```
MCC_GetPulsePos(&lCurPulseX,   &lCurPulseY,   &lCurPulseZ,   &lCurPulseU,
                &lCurPulseV,   &lCurPulseW,   &lCurPulseA,   &lCurPulseB,
                g_nGroupIndex);
```

使用 MCC_GetCurCommand() 可以獲得目前正在執行的運動命令相關的資訊，包括運動命令類型、運動命令編碼、進給速度、目的點位置座標等。使用 MCC_GetCommandCount() 可以獲得運動命令緩衝區中庫存且尚未執行的運動命令之數目。

7. 運動狀態檢視

相關函式

MCC_GetMotionStatus()

範例程式

MotionFinished.cpp

內容說明

利用 MCC_GetMotionStatus()的函式傳回值可檢視機器目前的運動狀態。若函式傳回值為 GMS_RUNNING，表示機器處於運動狀態；若函式傳回值為 GMS_STOP，表示機器處於停止狀態，運動命令緩衝區中已無命令；若呼叫 MCC_HoldMotion() 成功，此時 MCC_GetMotionStatus() 的函式傳回值為 GMS_HOLD，表示機器處於暫停狀態，仍有運動命令尚未執行完成；若 MCC_GetMotionStatus() 的函式傳回值為 GMS_DELAYING，表示因呼叫 MCC_DelayMotion()，系統目前處於運動延遲狀態。下面為使用範例：

Step 1：宣告讀取運動狀態參數

```
int nStatus;
```

Step 2：啟動伺服

```
MCC_SetServoOn(0, CARD_INDEX);
MCC_SetServoOn(1, CARD_INDEX);
```

Step 3：直線運動，後執行運動狀態讀取

```
MCC_Line(20, 20, 0, 0, 0, 0, 0, 0, g_nGroupIndex);
```

Step 4：等待 MCC_Line()執行完，產生 GMS_STOP 後方跳出迴圈，再繼續執行下面的命令

```
while (MCC_GetMotionStatus(g_nGroupIndex) != GMS_STOP);
{…….}
```

Step 5：延遲運動命令，此時運動狀態為 GMS_DELAYING

MCC_DelayMotion(10000); // delay 10000 ms

Step 6：再次運動改變運動狀態

MCC_Line(50, 50, 0, 0, 0, 0, 0, 0, g_nGroupIndex);

Step 7：當按下 H 鍵，運動暫停，狀態呈現 GMS_HOLD

nRet = MCC_HoldMotion(g_nGroupIndex);

Step 8：當按下 C 鍵，將繼續未完成之運動，狀態呈現 GMS_RUNNING

nRet = MCC_ContiMotion(g_nGroupIndex);

printf("Motion status : %d \r", status);

8. 加、減速段使用時間設定

相關函式

MCC_SetAccTime()
MCC_SetDecTime()
MCC_GetAccTime()
MCC_GetDecTime()
MCC_SetPtPAccTime()
MCC_SetPtPDecTime()
MCC_GetPtPAccTime()
MCC_GetPtPDecTime()

範例程式

AccStep.cpp

內容說明

一般運動(包括直線、圓弧、圓運動)與點對點運動的加、減速時間預設值為300 毫秒，但可使用 MCC_SetAccTime()、MCC_SetDecTime()、MCC_SetPtPAccTime()、MCC_SetPtPDecTime()調整加、減速的時間，使這些運動有較為平順的加、減速過程。

不同速度應採用不同的加、減速時間。使用 MCCL 時，使用者需自行設計各種速度下的加、減速時間，適當的加、減速時間會因為使用不同的馬達與機構而有所差異。加、減速時間可以利用下面的公式獲得：

運動時的加速時間 = 要求的速度 / 要求的加速度
運動時的減速時間 = 要求的速度 / 要求的減速度

9. 進給速度設定

相關函式

MCC_SetFeedSpeed()

MCC_GetFeedSpeed()

MCC_SetPtPSpeed()

MCC_GetPtPSpeed()

範例程式

SetSpeed.cpp

內容說明

在進行直線、圓弧、圓運動前需先設定進給速度，所設定的進給速度不應超過 MCC_SetSysMaxSpeed() 的設定值。

使用 MCC_SetFeedSpeed() 設定直線、圓弧、圓、螺線運動的進給速度，例如呼叫 MCC_SetFeedSpeed (20, g_nGroupIndex)時，表示進給速度為 20 UU/sec。

使用 MCC_SetPtPSpeed() 來設定點對點運動的速度，第一個參數為”各軸最大速度的百分比再乘以 100”，範圍從 0 ~ 100。例如執行 MCC_SetPtPSpeed(50, g_nGroupIndex) 時，表示要求各軸的點對點運動速度為(RPM/60 × Pitch / GearRatio) × 50%。RPM、Pitch、GearRatio 定義在機構參數中。

10. 直線、圓弧、圓、螺線運動(一般運動)

相關函式

MCC_SetAbsolute()
MCC_SetFeedSpeed()
MCC_Line()
MCC_ArcXY()
MCC_CircleXY()

範例程式

GeneralMotion.cpp

內容說明

在完成 group、機構與編碼器參數設定、啟動系統、設定進給速度的上限、開啟伺服迴路(使用步進馬達時不需此動作)與設定進給速度後，即可進行直線、圓弧、圓、螺線運動。在使用圓弧函式時需注意給定的參數是否合理 (起始點、參考點與目的點等三點的位置不能在同一直線上)。下面為函式使用範例。

Step 1：使用絕對座標型態表示各軸位置並設定進給速度

```
MCC_SetAbsolute(g_nGroupIndex);
MCC_SetFeedSpeed(10, g_nGroupIndex);
```

Step 2：執行直線運動命令

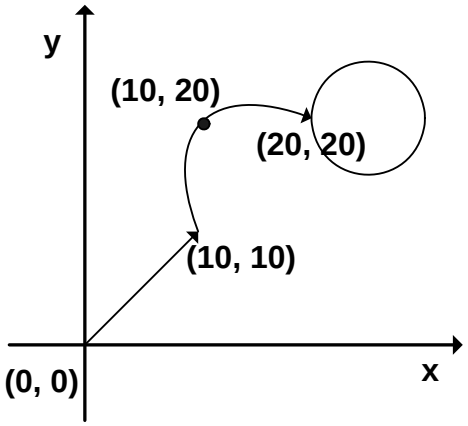
```
MCC_Line(10, 10, 0, 0, 0, 0, 0, 0, g_nGroupIndex);
```

Step 3：執行圓弧命令，請注意需避免起始點、參考點與目的點在同一直線上

```
nRet = MCC_ArcXY(10, 20, 20, 20, g_nGroupIndex);
if (nRet != NO_ERR)
{
    /*
    利用傳回值了解錯誤發生的原因，如果參數錯誤則傳回值為
    PARAMETER_ERR。
```

```
    */
}
```

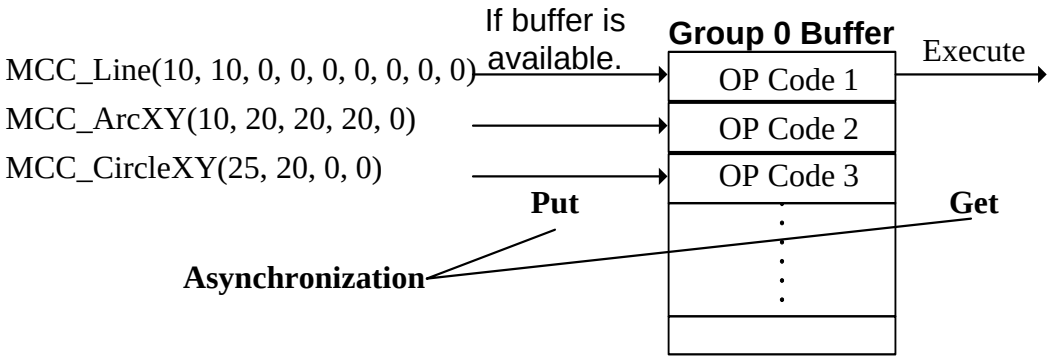
可以利用函式的傳回值了解錯誤發生的原因，傳回值的意義請參考”**IMP Series 運動控制函式庫參考手冊**”。行進的軌跡如下圖所示。



Step 3：執行圓命令

MCC_CircleXY(25, 20, 0, g_nGroupIndex);

運動命令的執行過程是運動函式先將運動命令(OP Code)置於各 group 專屬的運動命令緩衝區中，MCCL 再同時從不同 group 的運動命令緩衝區中擷取運動命令依序執行。這兩個動作並不是同步動作，也就是並不需等到前一筆運動命令執行完成，即可將新的運動命令送到運動命令緩衝區中。



若運動命令緩衝區已滿，則函式的傳回值為COMMAND_BUFFER_FULL_ERR，此筆運動命令將不被接受。預設每個運動命令緩衝區擁有 10000 個運動命令儲存空間。上圖顯示對 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，更詳細的說明請參考”**IMP Series 運動控制函式庫使用手冊**”。

11. 點對點運動

相關函式

MCC_SetAbsolute()
MCC_SetPtPSpeed()
MCC_PtP()

範例程式

PtPMotion.cpp

內容說明

在完成 group、機構與編碼器參數設定、啟動系統、設定進給速度的上限、開啟伺服迴路(使用步進馬達時不需此動作)與設定進給速度後，即可執行點對點運動。下面為函式使用範例。

Step 1：使用絕對座標與設定進給速度

```
MCC_SetAbsolute(g_nGroupIndex);
MCC_SetFeedSpeed(20, g_nGroupIndex);
```

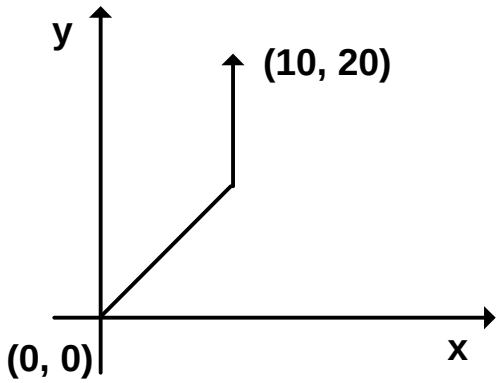
Step 2：設定各軸使用最大速度的 20%運動，也就是(RPM × Pitch / GearRatio) × 20%

```
MCC_SetPtPSpeed(20, g_nGroupIndex);
```

Step 3：各軸使用不同動的方式運動至(10, 20)

MCC_PtP(10, 20, 0, 0, 0, 0, 0, 0, g_nGroupIndex);

點對點運動採用不同動的運動方式，也就是各軸使用各自的速度運動，各軸在同時啟動後並不一定會同時到達目的點，這點與一般運動不同。一般運動使用同動的運動方式，各軸同時啟動後會同時到達目的點。下圖為點對點運動的運動軌跡，此時各軸的速度相同。



12. JOG 運動

相關函式

MCC_JogPulse()

MCC_JogSpace()

MCC_JogConti()

範例程式

JogMotion.cpp

內容說明

MCC_JogPulse()使用 pulse 為單位，對特定軸進行微動動作，但移動的 pulse 數不能超過 2048。MCC_JogSpace() 使用單位與一般運動相同，對特定軸進行吋動動作。MCC_JogConti() 則可運動至機構參數所設定的工作區間邊界。MCC_JogSpace() 與 MCC_JogConti() 所需的參數包括速度比例，設定方式與點對點運動類似。下面為使用範例。

Step 1：使 X 軸移動 100 pulses

```
MCC_JogPulse(0, 100, g_nGroupIndex) ;
```

Step 2：使用速度為(RPM × Pitch / GearRatio) × 10%，使 X 軸移動 -1 UserUnit 距離

```
MCC_JogSpace(-1, 10, 0, g_nGroupIndex);
```

Step 3：使用速度為(RPM × Pitch / GearRatio) × 10%，使 X 軸移動至工作區間的右邊界

```
MCC_JogConti(1, 10, 0, g_nGroupIndex);
```

13. 定位控制

相關函式

MCC_SetInPosMaxCheckTime()

MCC_EnableInPos

MCC_SetInPosToleranceEx

MCC_GetInPosStatus

範例程式

InPosCheck.cpp

內容說明

本範例程式利用編碼器的計數值(實際機台位置)與目標位置之誤差，檢查每一個運動軸是否滿足定位確認條件。

當運動命令執行完成將開始檢視是否滿足定位條件，若檢視時間超過設定值，如還存在某些運動軸的位置誤差無法滿足定位條件，則紀錄此現象，並停止執行其他運動命令。使用者可以強制馬達產生誤差並觀察運作情況。此項功能的使用步驟如下：

Step 1：設定定位確認最長的檢查時間，單位ms

MCC_SetInPosMaxCheckTime(1000, g_nGroupIndex);

Step 2：設定定位控制模式

MCC_SetInPosMode(IPM_ONETIME_BLOCK, g_nGroupIndex);

Step 3：設定各軸誤差值，單位為 mm 或 inch

MCC_SetInPosToleranceEx(0.5, 0.5, 1000, 1000, 1000, 1000, 1000, 1000, g_nGroupIndex);

Step 4：啟動定位控制功能

MCC_EnableInPos(g_nGroupIndex);

Step 5：讀取各軸定位控制狀態，正確到位狀態為 0xff(255)

MCC_GetInPosStatus(&byInPos0, &byInPos1, &byInPos2, &byInPos3, &byInPos4, &byInPos5, &byInPos6, &byInPos7, g_nGroupIndex);

Step 6：抓取錯誤代碼

nErrCode = MCC_GetErrorCode(g_nGroupIndex);

14. 原點復歸運動

相關函式

MCC_SetHomeConfig()
MCC_Home()
MCC_GetGoHomeStatus()
MCC_AbortGoHome()

範例程式

GoHome.cpp

內容說明

原點復歸的程序將依照原點復歸參數中 SYS_HOME_CONFIG 的設定內容，可以使用 MCC_SetHomeConfig() 設定原點復歸參數(請參考"**IMP Series 運動控制函式庫使用手冊**")。

利用 MCC_GetGoHomeStatus() 可獲得原點復歸程序是否已經完成，另外在原點復歸運動過程中可呼叫 MCC_AbortGoHome() 強迫停止原點復歸運動。

目前 MCCL 所提供的原點復歸功能，一次只能針對一張運動控制卡，如需操作多張運動控制卡，則需使用 MCC_GetGoHomeStatus() 來確定目前進行的原點復歸運動已經完成，才能對下一張運動控制卡呼叫 MCC_Home() 進行原點復歸的動作。下面為使用範例：

Step 1：設定原點復歸參數

```
SYS_HOME_CONFIG      stHomeConfig;

stHomeConfig.wMode          = 3;      //   設定原點復歸模式
stHomeConfig.wDirection     = 1;      //   設定往負方向做原點復歸運動
stHomeConfig.wSensorMode    = 0;      //   Normal Open
stHomeConfig.nIndexCount    = 0;
stHomeConfig.dfAccTime      = 300;    //   ms
stHomeConfig.dfDecTime      = 300;    //   ms
stHomeConfig.dfHighSpeed    = 10;     //   mm/s
```

```
stHomeConfig.dfLowSpeed      = 2;      //  mm/s
stHomeConfig.dfOffset        = 0;
```

Step 2：設定原點復歸參數

```
for (WORD wChannel = 0;wChannel < MAX_AXIS_NUM;wChannel++)
    MCC_SetHomeConfig(&stHomeConfig, wChannel, CARD_INDEX);
```

Step 3：原點復歸，0xff 表示該軸不需進行原點復歸動作

```
MCC_Home(0, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, CARD_INDEX);
```

Step 4：如果有需要可使用此函式，停止原點復歸運動

```
MCC_AbortGoHome();
```

Step 5：利用函式傳回值判斷原點復歸運動是否已經完成, nStatus 的值若為 1，表示原點復歸運動已經完成

```
nStatus = MCC_GetGoHomeStatus();
```

15. 運動暫停、持續、棄置

相關函式

MCC_ HoldMotion()

MCC_ContiMotion()

MCC_AbortMotionEx()

範例程式

CtrlMotion.cpp

內容說明

MCC_HoldMotion() 用來暫停目前正在執行的運動命令，MCC_ContiMotion() 則用來繼續執行被暫停執行的運動命令，因此 MCC_ContiMotion() 需與 MCC_HoldMotion() 搭配使用，且需使用在相同的 group 中。MCC_AbortMotionEx() 則用來設定減速停止的時間並棄置被暫停或執行中的運動命令。

目前若無執行中的運動命令，呼叫 MCC_HoldMotion() 時的函式傳回值將為 HOLD_ILLGEGAL_ERR；先前若呼叫 MCC_HoldMotion() 不成功，呼叫 MCC_ContiMotion() 時的函式傳回值將為 CONTI_ILLGEGAL_ERR。無論目前運動狀態為何，呼叫 MCC_AbortMotionEx() 皆會使運動(減速)停止並清除運動命令緩衝區中的庫存命令。

16. 強迫延遲執行運動命令

相關函式

MCC_InitSystem()

MCC_DelayMotion()

範例程式

DelayMotion.cpp

內容說明

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 ms 為計時單位；下面的範例中，在執行完第一筆運動命令(Line)後，將延遲 3000 ms，才會再執行下一筆運動命令。

Step 1：插值時間設為 INTERPOLATION_TIME

```
nRet = MCC_InitSystem(INTERPOLATION_TIME, stCardConfig, 1);
```

Step 2：開始運動命令

```
MCC_Line(10, 10, 0, 0, 0, 0, 0, 0, g_nGroupIndex);
```

Step 3：延遲 3000 ms 才執行下一筆命令，請觀察運動狀態

```
MCC_DelayMotion(3000);
```

17. 速度強制控制

相關函式

MCC_OverrideSpeed()

MCC_GetOverrideRate()

MCC_OverrideSpeedEx()

範例程式

OverrideSpeed.cpp

內容說明

MCC_OverrideSpeed() 可用來設定直線、圓弧、圓、螺線運動速度強制比例，所需的參數為更新速度為原來速度之百分比 × 100。MCC_GetOverrideRate() 則用來獲得目前的速度強制比例。下面為使用範例。

Step 1：設定直線、圓弧、圓、螺線運動的進給速度為 20 mm/ sec

```
MCC_SetFeedSpeed(20, g_nGroupIndex);
MCC_Line(10, 10, 0, 0, 0, 0, 0, 0, 0, g_nGroupIndex)
```

Step 2：設定運動速度強制比例，目前的速度將變為 20 ×150% = 30 mm /sec

```
MCC_OverrideSpeedEx(150, 1, g_nGroupIndex);
```

Step 3：讀取強制比例，dfRate 應等於 150

```
dfRate = MCC_GetOverrideRate(g_nGroupIndex);
```

18. 軟體過行程檢查與硬體極限開關檢查

相關函式

MCC_SetOverTravelCheck()
MCC_GetOverTravelCheck()
MCC_EnableLimitSwitchCheck()
MCC_DisableLimitSwitchCheck()
MCC_GetLimitSwitchStatus()

範例程式

CheckOT.cpp

內容說明

MCCL 提供軟體過行程檢查功能(或稱為軟體極限保護功能)，當啟動軟體過行程檢查功能後，若任一軸的行進範圍將超出工作區間，系統將停止運動(並產生一錯誤記錄)。此時若要使系統恢復正常狀態，必須先清除系統中的錯誤紀錄，然後才能往反方向移動。機構參數中的 dfHighLimit、dfLowLimit 分別用來設定軟體左右極限的位置；MCC_SetOverTravelCheck() 用來啟動與關閉此項功能，MCC_GetOverTravelCheck() 則用來檢查目前的設定狀態。下面為使用範例。

Step 1：啟動 X 軸軟體過行程檢視功能

```
MCC_SetOverTravelCheck (1, 0, 0, 0, 0, 0, 0, 0, g_nGroupIndex);
```

Step 2：OT0 ~ OT5 的值若為 1 表示已設定過行程檢查功能，否則為 0

```
MCC_GetOverTravelCheck( &OT0, &OT1, &OT2, &OT3, &OT4, &OT5, &OT6,
&OT7, g_nGroupIndex);
```

Step 3：讀取可能產生的錯誤訊息

```
nErrCode = MCC_GetErrorCode(g_nGroupIndex);
```

利用 MCC_GetErrorCode() 的傳回值，可判斷系統目前位置是否已經超出軟體極限值，導致無法運動(因內部已產生錯誤記錄)。傳回值若為 0xF301 ~

0xF308，則依序代表 X 軸 ~ B 軸出現此種情形，此狀況下可依下面範例使系統回復正常狀態。

Step 4：清除系統中的錯誤紀錄, 使系統回復正常狀態

MCC_ClearError(g_nGroupIndex);

MCCL 也提供硬體極限開關(Limit Switch) 檢查功能，要使極限開關能正常運作，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 2，則呼叫 MCC_EnableLimitSwitchCheck() 並無任何意義。

若使用 MCC_EnableLimitSwitchCheck(1)，則只有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)，才會停止該 Group 之運動；若呼叫 MCC_EnableLimitSwitchCheck(0)，則只要碰觸到極限開關(不管行進方向)，皆會停止該 Group 之運動。

利用 MCC_GetErrorCode() 的傳回值可判斷目前是否因碰觸到極限開關而無法運動(因內部已產生錯誤記錄)。傳回值若為 0xF701 ~ 0xF708，則依序代表 X 軸 ~ B 軸出現此種情形，此狀況下可依下面範例使系統回復正常狀態。

a. 若之前呼叫：MCC_EnableLimitSwitchCheck(0)
   則：反方向退出 Limit Switch
b. 若之前呼叫：MCC_EnableLimitSwitchCheck(1)
   則：反方向退出 Limit Switch
c. 若之前呼叫：MCC_EnableLimitSwitchCheck(2)
   則：MCC_ClearError() → 反方向退出 Limit Switch
d. 若之前呼叫：MCC_EnableLimitSwitchCheck(3)
   則：MCC_ClearError() → 反方向退出 Limit Switch

19. 連續運動功能設定

相關函式

MCC_EnableBlend()

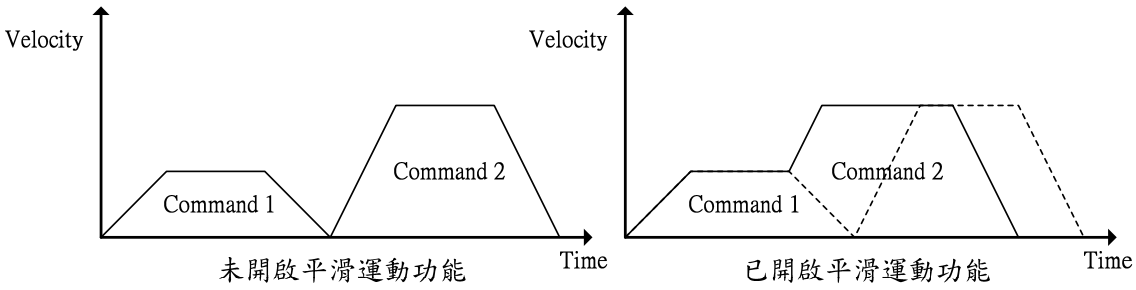
MCC_DisableBlend()

MCC_CheckBlend()

範例程式

SetBlend.cpp

內容說明



由圖中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。

呼叫 MCC_EnableBlend() 與 MCC_DisableBlend() 可分別開啟與關閉速度連續功能。呼叫 MCC_CheckBlend()則可獲得目前的狀態設定，若函式傳回值為 0，表示已開啟速度連續功能；若函式傳回值為 1，則表示此時關閉速度連續功能。

20. 讀取與清除錯誤狀態

相關函式

MCC_GetErrorCode()

MCC_ClearError()

範例程式

ErrorStatus.cpp

內容說明

系統錯誤發生後，若已排除錯誤狀況，仍必須呼叫 MCC_ClearError()，來清除系統中的錯誤記錄，否則無法正常繼續執行往後的運動。通常在系統運作中使用者應隨時讀取目前的錯誤代碼，以檢查在系統運作時是否發生錯誤。下面為使用範例，另外也可參閱”**軟體過行程檢查與硬體極限開關檢查**”此節關於這兩個函式的使用方式。

此部分與範例程式不同，使用者可以參考以下寫法去處理錯誤產生。

```
if (MCC_GetErrorCode(g_nGroupIndex))
{
    /*
    在此排除錯誤狀況
    */
    MCC_ClearError(g_nGroupIndex);//   清除系統中的錯誤記錄
}
```

21. 齒輪齒隙與間隙補償

相關函式

MCC_SetCompParam()

MCC_UpdateCompParam()

範例程式

Compensate.cpp

內容說明

MCCL 所提供的齒輪齒隙與間隙補償功能，能彌補機台在傳動時因齒輪、螺桿製造上的缺陷所造成的誤差，如齒隙誤差、背隙誤差，詳細的說明請參考”**IMP Series 運動控制函式庫使用手冊**”。

22. 如何完成 8 軸連續運動

相關函式

MCC_CreateGroup()

MCC_SetFeedSpeed()

MCC_EnableBlend()

MCC_Line()

範例程式

SyncLine.cpp

內容說明

當 1 個 group 已使用 MCC_EnableBlend() 開啟連續運動功能後(滿足路徑與速度連續條件)，如多次呼叫 MCC_Line() 時，雖能達到 8 軸同動要求(也就是 8 軸同時啟動且同時靜止)，但只有前 3 軸也就是 X、Y、Z 軸能滿足路徑與速度連續的條件，而後五軸也就是 U、V、W、A、B 軸僅能滿足同動要求。

如需 8 軸同動且滿足路徑與速度連續的條件，則可使用 3 個 group；第 1 個 group 負責前 3 軸的軌跡規劃，第 2 個 group 負責中間三軸的軌跡規劃，第 3 個 group 負責最後兩軸的軌跡規劃。

但為了滿足 8 軸同動要求，第 2 個 group 的速度可使用第 2 個 group 要求移動的距離與第 1 個 group 要求移動的距離之比值，再乘上第 1 個 group 的進給速度而換算得到；第 3 個 group 的速度可使用第 3 個 group 要求移動的距離與第 1 個 group 要求移動的距離之比值，再乘上第 1 個 group 的進給速度而換算得到。

此過程的程式碼如下，此時使用者需呼叫 fnSyncLine() 代替使用 MCC_Line()。

Step 1：宣告 fnSyncLine 函式

```
void fnSyncLine(double x, double y, double z, double u, double v, double w, double a,
      double b, double    dfXYZSpeed);
```

Step 2：設定並使用三個 Groups

```
nt g_nGroupIndex0 = -1;
int g_nGroupIndex1 = -1;
int g_nGroupIndex2 = -1;

//    set group parameters
MCC_CloseAllGroups();
g_nGroupIndex0 = MCC_CreateGroup(0, 1, 2, -1, -1, -1, -1, -1, CARD_INDEX);
if( g_nGroupIndex0 < 0 )
{
      printf("Groups create error !\n\n");
      return;
}

g_nGroupIndex1 = MCC_CreateGroup(3, 4, 5, -1, -1, -1, -1, -1, CARD_INDEX);
if( g_nGroupIndex1 < 0 )
{
      printf("Groups create error !\n\n");
      return;
}

g_nGroupIndex2 = MCC_CreateGroup(6, 7, -1, -1, -1, -1, -1, -1, CARD_INDEX);
if( g_nGroupIndex2 < 0 )
{
      printf("Groups create error !\n\n");
      return;
}
```

Step 3：啟用平滑運動

```
MCC_EnableBlend(g_nGroupIndex0);
MCC_EnableBlend(g_nGroupIndex1);
MCC_EnableBlend(g_nGroupIndex2);
```

Step 4：呼叫 fnSyncLine 函式

```
fnSyncLine (10, 20, 30, 40, 50, 60, 70, 80, 10);
fnSyncLine (40, 50, 60, 10, 20, 30, 70, 80, 10);
```

Step 5： fnSyncLine 函式定義

```
void fnSyncLine(double x, double y, double z, double u, double v, double w, double a,
double b, double    dfXYZSpeed)
{
    double    dfDistance0, dfDistance1, dfDistance2, dfUVWSpeed, dfABSpeed;

    dfDistance0 = x * x + y * y + z * z;

    if (dfDistance0 && dfXYZSpeed)
    {
        MCC_SetFeedSpeed(dfXYZSpeed, g_nGroupIndex0);

        //  由 group 的定義得知，第 1 個 group(也就是 g_nGroupIndex0)會
            將此命令由前三軸輸出
        MCC_Line(x, y, z, 0, 0, 0, g_nGroupIndex0);

        //  換算後中間三軸應有的速度
        dfDistance1 = u * u + v * v + w * w;
        dfUVWSpeed = dfXYZSpeed * sqrt(dfDistance1/ dfDistance0);
        MCC_SetFeedSpeed(dfUVWSpeed, g_nGroupIndex1);
```

```
            //  由 group 的定義得知，第 2 個 group(也就是 g_nGroupIndex1)會
                將此命令由中間三軸輸出
            MCC_Line(u, v, w, 0, 0, 0, g_nGroupIndex1);

            //  換算最後二軸應有的速度
            dfDistance2 = a * a + b * b;
            dfABSpeed = dfXYZSpeed * sqrt(dfDistance2/ dfDistance0);
            MCC_SetFeedSpeed(dfABSpeed, g_nGroupIndex2);

            //  由 group 的定義得知，第 3 個 group(也就是 g_nGroupIndex2)會
                將此命令由後二軸輸出
            MCC_Line(a, b, 0, 0, 0, 0, g_nGroupIndex2);
        }
    }
```

23. 編碼器(ENCODER)計數值觸發中斷服務函式功能

相關函式

MCC_SetENCRoutine()
MCC_SetENCCompValue()
MCC_EnableENCCompTrigger()
MCC_DisableENCCompTrigger()
MCC_SetENCInputRate()
MCC_GetENCValue()

範例程式

ENCCompare.cpp

內容說明

MCCL 所提供的編碼器計數值觸發中斷服務函式功能，可設定編碼器計數值的比較值，在開啟此項功能後，當編碼器的計數值等於設定的比較值時(可以使用 MCC_GetENCValue() 讀取編碼器的計數值)，MCCL 將自動呼叫使用者串接的中斷服務函式。下面為使用範例。

Step 1：宣告中斷服務函式

```
void _stdcall ENC_ISR_Function(ENCINT_EX *pstINTSource);
```

Step 2：串接中斷服務函式

```
MCC_SetENCRoutine(ENC_ISR_Function, CARD_INDEX);
```

Step 3：設定比較值為 20000 pulses

```
MCC_SetENCCompValue(20000, CHANNEL_INDEX, CARD_INDEX);
```

Step 4：開啟計數值觸發中斷服務函式功能

```
MCC_EnableENCCompTrigger(CHANNEL_INDEX, CARD_INDEX);
MCC_Line(100, 0, 0, 0, 0, 0, 0, 0, g_nGroupIndex);
```

Step 5：定義中斷服務常式

```
VOID _STDCALL ENC_ISR_FUNCTION(ENCINT_EX *PSTINTSOURCE)
{
    //  判斷是否因第 0 個 CHANNEL 的比較條件成立而觸發
    IF (PSTINTSOURCE->COMP0)
        //  放棄目前正在執行與運動緩衝區中所有的運動命令
        MCC_AbortMotionEx(0, g_nGroupIndex);

    ENC_ISR++;

    //  關閉計數值觸發中斷功能
    MCC_DisableENCCompTrigger(CHANNEL_INDEX);
}
```

上面的範例顯示在開啟編碼器計數值觸發中斷服務函式功能後，將進行直線運動，待編碼器的計數術值等於 20000 pulses 時，將停止未完成的直線運動。MCC_AbortMotionEx 第一參數設為 0，使減速時間為零，可以讓停止後編碼器位置接近 20000。

24. 閂鎖(LATCH)編碼器計數值與 INDEX 訊號觸發中斷服務函式功能

相關函式

MCC_SetENCRoutine()
MCC_GetENCValue()
MCC_SetENCLatchType(()
MCC_SetENCLatchSource()
MCC_EnableENCIndexTrigger()

範例程式

GetENCLatch.cpp

內容說明

MCCL 所提供的閂鎖(Latch)編碼器計數值功能，可使用 MCC_SetENCLatchSource() 指定觸發條件(來源)，在滿足觸發條件與閂鎖模式後(使用 MCC_SetENCLatchType() 設定觸發模式)，可將編碼器的計數值紀錄在閂鎖暫存器內，使用 MCC_GetENCLatchValue() 可以讀取閂鎖暫存器內的紀錄值。下面為使用範例。

Step 1：設定編碼器計數值閂鎖模式

ENC_TRIG_FIRST	第一次滿足觸發條件即 latch 計數值不再變動
ENC_TRIG_LAST	觸發條件滿足時即 latch 計數值且隨條件一再滿足即一再 latch 新的計數值

MCC_SetENCLatchType(ENC_TRIG_LAST, CHANNEL_INDEX, CARD_INDEX);

Step 2：設定編碼器觸發源。共有 16 種觸發來源(條件)可做為閂鎖計數值的條件。設定時可同時取多個條件的聯集，此時選取編碼器 index 訊號為觸發來源(條件)

MCC_SetENCLatchSource(ENC_TRIG_INDEX0, CHANNEL_INDEX, CARD_INDEX);

由上面的範例可以看出使用編碼器 INDEX 訊號做為觸發來源(條件)，為了在編碼器 INDEX 訊號發生後，立即使用 MCC_GetENCLatchValue() 讀取閂鎖暫存器內的紀錄值，可以使用編碼器 INDEX 訊號觸發中斷函式功能。要使用此項功能首先需串接自訂的中斷服務函式並開啟。

Step 3：宣告中斷服務函式

```
void _stdcall ENC_ISR_Function(ENCINT_EX *pstINTSource);
```

Step 4：串接中斷服務函式

```
MCC_SetENCRoutine(ENC_ISR_Function, CARD_INDEX);
```

Step 5：開啟編碼器 index 訊號觸發中斷服務函式功能

```
MCC_EnableENCIndexTrigger(CHANNEL_INDEX, CARD_INDEX);
```

Step 6：定義中斷服務常式

```
void _stdcall ENC_ISR_Function(ENCINT_EX *pstINTSource)
{
    if (pstINTSource->INDEX0)//   判斷是否由 INDEX 訊號所觸發
    {
        //   讀取閂鎖暫存器內的紀錄值
        MCC_GetENCLatchValue(&lLatchValue, CHANNEL_INDEX,
    CARD_INDEX);
    }
}
```

更詳細的說明請參考”**IMP Series 運動控制函式庫使用手冊**”。

25. 近端接點(LOCAL I/O)訊號控制與觸發中斷服務函式功能

相關函式

MCC_SetServoOn();
MCC_SetServoOff()
MCC_EnablePosReady()
MCC_DisablePosReady()
MCC_GetLimitSwitchStatus()
MCC_GetHomeSensorStatus()
MCC_SetLIORoutine ()
MCC_SetLIOTriggerType()
MCC_EnableLIOTrigger()

範例程式

LIOTrigger.cpp

內容說明

MCCL 提供對近端接點(Local I/O)包括 servo on/off、position ready 輸出訊號的控制函式，另外也提供對 home sensor 與 hardware limit switch 輸入訊號的檢視函式。

所有輸入接點的訊號皆能觸發使用者自訂的中斷服務函式，使用”輸入接點訊號觸發中斷服務函式”的步驟如下：

Step 1：使用 MCC_SetLIORoutine()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函式原型：

```
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_SetLIORoutine(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、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所引起。LIOINT 的定義如下：

```
typedef struct _LIO_INT
{
    BYTE OTP0;          //Channel 0 Limit Switch+
    BYTE OTP1;          //Channel 1 Limit Switch+
    BYTE OTP2;          //Channel 2 Limit Switch+
    BYTE OTP3;          //Channel 3 Limit Switch+
    BYTE OTP4;          //Channel 4 Limit Switch+
    BYTE OTP5;          //Channel 5 Limit Switch+
    BYTE OTP6;          //Channel 6 Limit Switch+
    BYTE OTP7;          //Channel 7 Limit Switch+
    BYTE OTN0;          //Channel 0 Limit Switch-
    BYTE OTN1;          //Channel 1 Limit Switch-
```

```
    BYTE OTN2;          //Channel 2 Limit Switch-
    BYTE OTN3;          //Channel 3 Limit Switch-
    BYTE OTN4;          //Channel 4 Limit Switch-
    BYTE OTN5;          //Channel 5 Limit Switch-
    BYTE OTN6;          //Channel 6 Limit Switch-
    BYTE OTN7;          //Channel 7 Limit Switch-
    BYTE HOME0;         //Channel 0 Home Sensor
    BYTE HOME1;         //Channel 1 Home Sensor
    BYTE HOME2;         //Channel 2 Home Sensor
    BYTE HOME3;         //Channel 3 Home Sensor
    BYTE HOME4;         //Channel 4 Home Sensor
    BYTE HOME5;         //Channel 5 Home Sensor
    BYTE HOME6;         //Channel 6 Home Sensor
    BYTE HOME7;         //Channel 7 Home Sensor
} LIOINT;
```

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緣(Rising Edge)觸發、下緣(Falling Edge)觸發或是轉態(Level Change)觸發。MCC_SetLIOTriggerType()的輸入參數可為：

LIO_INT_RISE	上緣觸發(Default)
LIO_INT_FALL	下緣觸發
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

26. 計時器計時終了觸發中斷服務函式功能

相關函式

MCC_SetTMRRoutine();
MCC_SetTimer()
MCC_EnableTimer()
MCC_EnableTimerTrigger()

範例程式

TimerTrigger.cpp

內容說明

利用 MCCL 可以設定 IMP Series 運動控制卡上 32 bits 計時器的計時時間，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。要使用”計時器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
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函式原型：

```
typedef void(_stdcall *TMRISR)(TMRINT*)
```

例如自訂的函式可設計如下：

```
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
    }
}
```

接著使用 MCC_SetTMRRoutine(MyTMRFunction) 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、被宣告為 TMRINT 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所引起。TMRINT 的定義如下：

```
typedef struct _TMR_INT
{
    BYTE TIMER;
} TMRINT;
```

TIMER 欄位的值如果不為 0，表示計時器發生 Time Out 訊號。

Step 2：使用設定 MCC_SetTimer() 計時器之計時時間，計時單位為 1us

Step 3： 使用 MCC_EnableTimer() 開啟計時器計時功能

Step 4： 使用 MCC_EnableTimerTrigger() 開啟”計時終了觸發中斷服務函式”功能

27. WATCH DOG 功能

相關函式

MCC_SetWatchDogTimer()

MCC_SetWatchDogResetPeriod()

MCC_EnableWatchDogTimer()

範例程式

WatchDog.cpp

內容說明

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：MCC_SetWatchDogTimer() 設定 watch dog 計時器比較值，單位係 1us，設定範圍為 $1 \sim 2^{32}$。

也就是說如果使用下列的程式碼：

```
MCC_SetWatchDogTimer(10000000, CARD_INDEX);
```

此時表示第 0 張卡的 watch dog 計時器的比較值設定為

1us x 10000000 = 10s。

Step 2：使用 MCC_SetWatchDogResetPeriod() 設定 reset 訊號持續時間。

透過本函式可規劃因 watch dog 功能所產生 reset 硬體動作的持續時間，設定單位為 system clock(10ns)。

Step 3： 必須在 watch dog 計時終了前，使用 MCC_RefreshWatchDogTimer() 清除 watch dog 的計時內容。

使用者在搭配”計時器計時終了觸發中斷服務函式”之功能時，可以在 watch dog reset 硬體動作前，先作警示，並於計時中斷服務函式內進行必要的處理。

注意,搭配計時器計時終了觸發中斷服務函式功能,用以處理 Watch dog reset 事件，事先必須設定計時器之 Time out 時間 < Watch Dog time out 時間，否則一旦 Reset 事件比計時器事件提早發生，將導致系統被重置(Reset)。

28. 設定與讀取 REMOTE I/O 輸出、入接點訊號

相關函式

MCC_EnableARIOSetControl()
MCC_EnableARIOSlaveControl()
MCC_GetARIOInputValue()
MCC_SetARIOOutputValue()

範例程式

ARIOCtrl.cpp

內容說明

每張 IMP-2 運動控制卡擁有一組 IMP-ARIO 卡的接頭(稱為 Async Remote I/O Master 端，編號 RIO_SET0)，可同時控制 32 張 IMP-ARIO 卡(或稱為 Async Remote I/O Slave 端，編號 RIO_SLAVE0 ~ RIO_SLAVE31)。每張 IMP-ARIO 卡各提供 16 個輸出接點與 16 個輸入接點。

使用 EnableARIOSetControl() 與 EnableARIOSlaveControl() 啟動資料傳輸功能，下面為使用範例，此時開啟第一張卡(編號為 0)與他的 Slave 端的資料傳輸功能。

```
MCC_EnableARIOSetControl(RIO_SET0, CARD_INDEX);
MCC_EnableARIOSlaveControl(RIO_SET0, RIO_SLAVE0, CARD_INDEX);
```

在完成初始設定後，用低電位(ECOM-)接觸接點，MCC_GetARIOInputValue() 即可讀取輸入接點的訊號狀態；也可以使用 MCC_SetARIOOutputValue() 設定輸出接點的訊號狀態。

29. 讀取 REMOTE I/O 訊號傳輸狀態

相關函式

MCC_EnableARIOSetControl()
MCC_EnableARIOSlaveControl()
MCC_GetARIOTransStatus()
MCC_GetARIOMasterStatus()
MCC_GetARIOSlaveStatus()

範例程式

ARIOStatus.cpp

內容說明

使用 MCC_GetARIOTransStatus() 可以隨時監控各 Remote I/O Set 的資料傳輸狀態。當出現資料傳輸錯誤的情況時，可以使用 MCC_GetARIOMasterStatus() 與 MCC_GetARIOSlaveStatus() 所獲得的資訊獲知傳輸錯誤訊息是來自運動控制卡，或來自 IMP-ARIO 子板。

用 MCC_GetARIOTransStatus() 、 MCC_GetARIOMasterStatus() 與 MCC_GetARIOSlaveStatus() 所讀取的狀態，如果為 1 表示處於正常資料傳輸狀態，如果為 0 表示處於資料傳輸錯誤。

下面為使用範例。

```
WORD wTransStatus;

//  讀取傳輸狀態
MCC_GetARIOTransStatus( &wTransStatus,  RIO_SET0, CARD_INDEX);
```

wTransStatus 如果為 1 表示處於正常資料傳輸狀態，如果為 0 表示處於資料傳輸錯誤。

30. 規劃 DAC 類比電壓輸出

相關函式

MCC_StartDACConv()

MCC_SetDACOutput()

範例程式

DACOutput.cpp

內容說明

假使某一個運動軸不使用 Voltage Command 操作模式，則該運動軸相對的 D/A 輸出 Channel 可用來作為一般的類比電壓輸出 Channel。

使用 MCC_StartDACConv() 開始進行 DAC 轉換，呼叫 MCC_InitSystem(...) 成功後，MCCL 也會自動呼叫此函式；最後使用 MCC_SetDACOutput() 輸出電壓值。

31. ADC 電壓輸入單次轉換

相關函式

MCC_SetADCConvMode()
MCC_SetDACOutput()
MCC_SetADCSingleChannel()
MCC_StartADCConv()

範例程式

ADC1Time.cpp

內容說明

本範例程式規劃 ADC 的 Channel 0 進行單次的正負電壓型式(-5 ~ 5V)電壓轉換，並讀取輸入的電壓值。此項功能的使用步驟如下：

Step 1：設定轉換模式為單次電壓轉換模式
MCC_SetADCConvMode(ADC_MODE_SINGLE, CARD_INDEX);

Step 2：設定電壓轉換型式為雙極性模式(-5V ~ 5V)
MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 3：設定單次電壓轉換的 Channel
MCC_SetADCSingleChannel(0, CARD_INDEX);

Step 4：進行單次電壓轉換功能
MCC_StartADCConv(CARD_INDEX);

當使用單次電壓轉換模式的情形下，如需更新讀取的電壓值，需再次呼叫 MCC_StartADCConv(CARD_INDEX);也可以使用 MCC_GetADCWorkStatus() 判斷單次電壓轉換動作是否完成。

32. ADC 電壓輸入連續轉換

相關函式

MCC_SetADCConvMode()

MCC_SetADCConvType()

MCC_EnableADCConvChannel()

MCC_StartADCConv()

範例程式

ADCInput.cpp

內容說明

本範例程式規劃 ADC 的 Channel 0 進行連續的正負電壓型式(-5V ~ 5V)電壓轉換，並讀取輸入的電壓值。此項功能的使用步驟如下：

Step 1：設定轉換模式為連續電壓轉換模式

MCC_SetADCConvMode(ADC_MODE_FREE, CARD_INDEX);

Step 2：設定電壓轉換型式為雙極性模式(-5V ~ 5V)

MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 3：開啟 Channel 0 電壓轉換功能

MCC_EnableADCConvChannel(0, CARD_INDEX);

Step 4：開啟電壓轉換功能

MCC_StartADCConv(CARD_INDEX)

33. ADC 比較器中斷功能控制

相關函式

MCC_SetADCRoutine()
MCC_SetADCConvMode()
MCC_SetADCConvType()
MCC_SetADCCompValue()
MCC_SetADCCompType()
MCC_EnableADCCompTrigger()
MCC_EnableADCConvChannel()
MCC_StartADCConv()

範例程式

ADCComp.cpp

內容說明

本範例程式設定 ADC 的 Channel 0 比較器之比較值，當比較條件成立且電壓由高到低時將觸發使用者自訂的中斷處理函式。本範例將連續進行 ADC 轉換，也就是當比較條件成立時中斷將被連續觸發。此項功能的使用步驟如下：

Step 1：串接使用者自訂的中斷服務函式

```
MCC_SetADCRoutine(ADC_ISR_Function, CARD_INDEX);
```

使用者自訂的中斷服務函式可定義如下：

```
void _stdcall ADC_ISR_Function(ADCINT *pstINTSource)//  ADC 中斷服務程式
{
    if (pstINTSource->COMP0)//  判斷是否滿足比較條件
        nISRCount++;
}
```

Step 2：設定轉換模式為連續轉換

```
MCC_SetADCConvMode(ADC_MODE_FREE, CARD_INDEX);
```

Step 3：設定電壓轉換型式為雙極性模式(-5V ~ 5V)

MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 4：設定電壓比較器的比較值

MCC_SetADCCompValue(2.0, 0, CARD_INDEX);

Step 5：設定電壓比較條件為由高電壓到低電壓

MCC_SetADCCompType(ADC_COMP_FALL, 0, CARD_INDEX);

Step 6：開啟電壓比較器觸發使用者自訂的中斷處理函之功能

MCC_EnableADCCompTrigger(0, CARD_INDEX);

Step 7：開啟 Channel 0 電壓轉換功能

MCC_EnableADCConvChannel(0, CARD_INDEX);

Step 8：開啟電壓轉換功能

MCC_StartADCConv(CARD_INDEX)