IMP Series
Motion Control Command
Library
Example Manual

Version: V.1.01

Date: 2013.01

http://www.epcio.com.tw

Table of Contents

1. Description of Motion Control Command Library Examples

The examples provided in the installation CD-ROM are console modes. These examples can be integrated into applications of the user. The MCCL can support a maximum of six IMP Series motion control cards but most examples use only one motion control card (motion control card number represented by ***CARD_INDEX***) and one group (group number represented by ***g_nGroupIndex***) to increase the readability.

2. Setting Group, Mechanism and Encoder Parameters

Related Commands

MCC_SetSysMaxSpeed()

MCC_GetSysMaxSpeed()

MCC_SetMacParam()

MCC_GetMacParam()

MCC_SetEncoderConfig()

MCC_CloseAllGroups()

MCC_CreateGroup()

MCC_UpdateParam()

Example

InitSys.cpp

Description

This example describes the process of setting the group, mechanism and encoder parameters. First, use MCC_SetSysMaxSpeed() to set the maximum feed speed; then use MCC_SetMacParam() and MCC_SetEncoderConfig() to set the mechanism and encoder parameters for each axis. Lastly, use MCC_CreateGroup() to create a new group.

For more details regarding group usage and mechanism parameters, please refer to "**IMP Series Motion Control Command Library User Manual**".

3. Interpolation Time Adjustment

Related Commands

MCC_InitSystem()

MCC_GetCurPulseStockCount()

MCC_SetMaxPulseStockNum()

Example

CheckHWStock.cpp

Description

Shorter interpolation time have better motion control performance. The interpolation time can be set as a minimum of 1ms. MCC_SetMaxPulseStockNum() can be used to set the FIFO number to decrease the command delay caused by using FIFO. MCC_GetCurPulseStockCount() can be used to get the pulse stock count in the IMP Series motion control card to obtain the most appropriate interpolation time. During continuous motion, the pulse stock count must equal to the set maximum FIFO number to ensure the stable motion performance. If the pulse stock count equals 0, the interpolation time must be extended (the interpolation time is one of the necessary parameters for MCC_InitSystem()). Meanwhile, the interpolation time must also be extended if a delay occurs on the user interface.

4. Enabling and Disabling the Motion Control Command Library

Related Commands

MCC_InitSystem()

MCC_CloseSystem()

MCC_GetMotionStatus()

Example

InitSys.cpp

Description

This example shows using MCC_InitSystem () to initiate the motion control command library after completing settings of group and mechanism parameters. For the required parameters, please refer to "**IMP Series Motion Control Command User Manual**". Directions for this example is detailed as follows.

Step 1: Set motion control card hardware parameters

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

Step 2: Enabling the Motion Control Command Library

```
nRet = MCC_InitSystem( INTERPOLATION_TIME,  // Interpolation time set as 2 ms
                       stCardConfig,                // Hardware parameters
                       1);                          // Only use one motion control card

if (nRet == NO_ERR) // Successfully Enabling the Motion Control Command Library
{
    /*
    The user can execute other initializations here, such as setting the
    movement unit and the feed speed.
    */
}
```

Step 3:

MCC_CloseSystem() is used to disable the MCCL and the driver command library. There are two ways to shutdown the system:

i. Shutdown the system after the entire motion command is completed

First examine if the system is in "stop" status. If the return value of MCC_GetMotionStatus() is GMS_STOP, then the system has stopped.

```
while ((nRret = MCC_GetMotionStatus(g_nGroupIndex)) != GMS_STOP)
{
        MCC_TimeDelay(1); //    Sleep 1 ms
        //    because the "while" command was used to avoid system lockup
        //    and affect system operations, MCC_TimeDelay () is required to
        //    be called to release the CPU usage rights.
}

MCC_CloseSystem(); // Shutdown the MCCL and driver command library
```

ii. Directly shutdown the motion control library

Call MCC_CloseSystem() and the system will immediately stop the operation.

5. Setting System Status

Related Commands

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()

Example

SetStatus.cpp

Description

This example shows how to change the system status. If the status is not specified, the system will use the default to operate. For system default statuses, please refer to "**IMP Series Motion Control Command Library Reference Manual**". Content of commands are detailed below.

```
MCC_SetAbsolute(g_nGroupIndex);    //    use absolute coordinates to show the
                                          position of each axis

//    use the T curve as the acceleration type for linear, curve and circular motions
MCC_SetAccType('T', g_nGroupIndex);
```

```
//    use the S curve as the deceleration type for linear, curve and circular motions
MCC_SetDecType(‘S’ , g_nGroupIndex);

//    use the T curve as the acceleration type for point-to-point motion
MCC_SetPtPAccType('T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', g_nGroupIndex);

//    use the S curve as the deceleration type for point-to-point motion
MCC_SetPtPDecType('S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', g_nGroupIndex);

MCC_SetServoOn(0, CARD_INDEX);        // enable the servo system of axis 0

//    enable Position Ready output connection function
MCC_EnablePosReady(CARD_INDEX);
```

When activating the servo system, MCC_SetServoOn() needs to be called so that the system can function normally. Depending the actual situation to determine if MCC_EnablePosReady() needs to be called.

6. Getting Information of Motion Speed, Coordinates and Motion Commands

Related Commands

MCC_GetCurFeedSpeed()

MCC_GetFeedSpeed()

MCC_GetCurPos()

MCC_GetPulsePos()

MCC_GetCurCommand()

MCC_GetCommandCount()

Example

GetStatus.cpp

Description

MCC_GetCurFeedSpeed() can be used to get current feed speed; MCC_GetSpeed() can be used to get the current feed speed of each axis. MCC_GetCurPos() can be used to get the Cartesian coordinates of current position of each axis; MCC_GetPulsePos() can be used to get the motor coordinates (also referred to as pulse coordinates) of current position of each axis. Cartesian and motor coordinates can be calculated by using mechanism parameter; motor coordinates = Cartesian coordinates × (dfGearRatio / dfPitch) × dwPPR. Coordinates of each axis got by using MCC_GetCurPos() and MCC_GetPulsePos() are only meaning when the given axis actually corresponds to a hardware output channel.

Following is the example of using this command:

Step 1: Declare variables

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: Get the current feed speed

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

Step 3: Get the current feed speed of each axis

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

Step 4: Get current position of each axis

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

Step 5: Get pulse counter value of each axis

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

MCC_GetCurCommand() can be used to obtain the related information of motion commands currently being executed, including the motion command type, the motion command code, the feed speed and the position of end point. MCC_GetCommandCount() can be used to obtain the number of unexecuted motion commands stored in the motion command buffer.

7. Motion Status

Related Commands

MCC_GetMotionStatus()

Example

MotionFinished.cpp

Description

The current status of the motion can be checked by using the return value of MCC_GetMotionStatus(). If the return value is GMS_RUNNING, it means the motion is running. If the return value is GMS_STOP, then it means the motion has stopped and there is no command in command buffer. When MCC_HoldMotion() is successfully called, and the return value of MCC_GetMotionStatus() is GMS_HOLD, it means the motion is paused with unexecuted motion commands. If the return value of MCC_GetMotionStatus() is GMS_DELAYING, it means the motion is currently delayed because MCC_DelayMotion() has been called. Following is the example of using this command:

Step 1: Declare the motion status parameters acquires
int nStatus;

Step 2: Start servo
MCC_SetServoOn(0, CARD_INDEX);
MCC_SetServoOn(1, CARD_INDEX);

Step 3: Linear motion command. The motion status will appear as GMS_RUNNING.
MCC_Line(20, 20, 0, 0, 0, 0, 0, 0, g_nGroupIndex);

Step 4: After MCC_Line() is completed. The motion status changes to GMS_STOP and while loop ends.
while (MCC_GetMotionStatus(g_nGroupIndex) != GMS_STOP);
{…….}
Step 5: Delay motion command. The motion status will appear as GMS_DELAYING

```
MCC_DelayMotion(10000);          // delay 10000 ms
```

Step 6: Run the motion again and change the motion status

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

Step 7: Press the H button to pause the motion. The motion status will appear as GMS_HOLD

```
nRet = MCC_HoldMotion(g_nGroupIndex);
```

Step 8: Press the C button to continue the uncompleted motion. The motion status will appear as GMS_RUNNING

```
nRet = MCC_ContiMotion(g_nGroupIndex);

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

8. Setting Acceleration and Deceleration Time

Related Commands

MCC_SetAccTime()

MCC_SetDecTime()

MCC_GetAccTime()

MCC_GetDecTime()

MCC_SetPtPAccTime()

MCC_SetPtPDecTime()

MCC_GetPtPAccTime()

MCC_GetPtPDecTime()

Example

AccStep.cpp

Description

The default acceleration and deceleration time for general motion (including linear, curve and circular) and point-to-point motion are 300ms. MCC_SetAccTime(), MCC_SetDecTime(), MCC_SetPtPAccTime() and MCC_SetPtPDecTime() can be used to adjust interval time to ensure a steady acceleration or deceleration process.

Different speeds should apply different acceleration and deceleration time. When using the MCCL, the user must manually set the acceleration and deceleration time for each speed. The appropriate acceleration and deceleration time will vary with different motors and mechanisms. The acceleration and deceleration time can be got by following formulas:

Acceleration time in motion = required speed / required acceleration

Deceleration time in motion = required speed / required deceleration

9. Setting Feed Speed

Related Commands

MCC_SetFeedSpeed()

MCC_GetFeedSpeed()

MCC_SetPtPSpeed()

MCC_GetPtPSpeed()

Example

SetSpeed.cpp

Description

The feed speed must be set before running linear, curve and circular motions. The feed speed set should not exceed the setting of MCC_SetSysMaxSpeed().

Use MCC_SetFeedSpeed() to set the feed speed of linear, curve, circular and helix motions. For example, the feed speed is 20 UU/sec when MCC_SetFeedSpeed (20, g_nGroupIndex) is called.

Use MCC_SetPtPSpeed() to set the speed of point-to-point motion. The first parameter is “the maximum speed ratio of each axis multiplied by 100” with the range from 0 to 100. For example, when MCC_SetPtPSpeed(50, g_nGroupIndex) is being executed, meaning the required point-to-point motion speed of each axis is (RPM / 60 × Pitch / GearRatio) × 50%. RPM, Pitch and GearRatio are defined in mechanism parameters.

10. Linear, Curve, Circular and Helix Motion (General Motion)

Related Commands

MCC_SetAbsolute()

MCC_SetFeedSpeed()

MCC_Line()

MCC_ArcXY()

MCC_CircleXY()

Example

GeneralMotion.cpp

Description

After completing setting group, mechanism and encoder parameters, starting the system, setting the maximum feed speed, enabling the servo loop (not necessary when using the stepper motor) and setting the feed speed, linear, curve, circular and helix motions can be conducted. When using curve commands, the user must ensure given parameters are logical (starting piont, the reference point and the end point should not be located on the same line). Following is the example of how to use this command.

Step 1: Use absolute coordinates to show the position of each axis and set the feed speed

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

Step 2: Execute the linear motion command

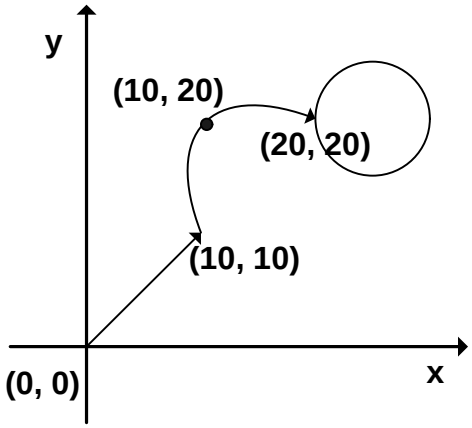
```
MCC_Line(10, 10, 0, 0, 0, 0, 0, 0, g_nGroupIndex);
```

Step 3: Execute the curve command. The user must ensure that starting point, the reference point and the end point are not located on the same line

```
nRet = MCC_ArcXY(10, 20, 20, 20, g_nGroupIndex);
if (nRet != NO_ERR)
{
    /*
    Use the return value to verify the cause of error. If the parameter is incorrect, then
```

```
    the return value will be PARAMETER_ERR。
    */
}
```

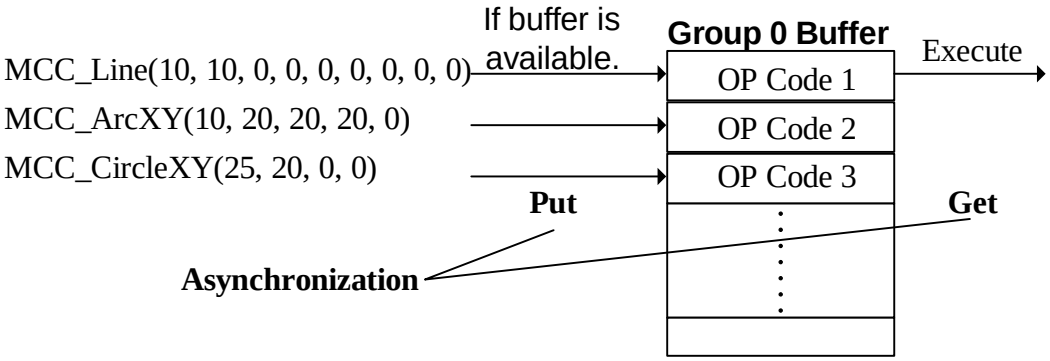
The user can use the return value of function to understand the cause of error. For meanings of the return value, please refer to "**IMP Series Motion Control Command Library Reference Manual**". The trajectory is showed in the following diagram.



Step 3: Execute the circular command

MCC_CircleXY(25, 20, 0, g_nGroupIndex);

During the motion command execution, the motion function first places the motion command (OP Code) in the exclusive motion command buffer of each group and then at the same time, the MCCL captures the motion command from the motion command buffer and execute orderly. These two motions are not synchronized, meaning that the new motion command can be set to the buffer before the execution of previous motion command is completed.



When the motion command buffer is full, the return value of the motion command function will be COMMAND_BUFFER_FULL_ERR, meaning that the command cannot be accepted. Each motion command buffer consists of the storage of 10000 motion commands. The above figure shows the operational process for Group 0 motion command buffer and demonstrates that motion commands of the same group will be executed in sequence.

Each group consists of its exclusive motion command buffer so that motion commands of different groups can be executed simultaneously. For more details, please refer to "**IMP Series Motion Control Command Library User Manual**".

11. Point-To-Point Motion

Related Commands

MCC_SetAbsolute()

MCC_SetPtPSpeed()

MCC_PtP()

Example

PtPMotion.cpp

Description

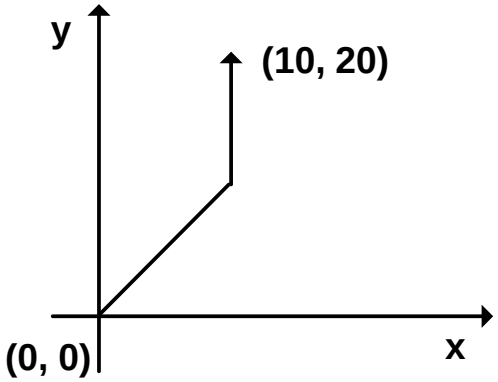
After completing setting group, mechanism and encoder parameters, starting the system, setting the maximum feed speed, enabling the servo loop (not necessary when using the stepper motor) and setting the feed speed, the point-to-point motion can be conducted. Following is the example of how to use this command.

Step 1: Use absolute coordinates and set the feed speed
MCC_SetAbsolute(g_nGroupIndex);
MCC_SetFeedSpeed(20, g_nGroupIndex);

Step 2: Set each axis to 20% of the maximum speed, meaning (RPM × Pitch / GearRatio) × 20%
MCC_SetPtPSpeed(20, g_nGroupIndex);

Step 3: Move asynchronously X-axis to position at 10 and Y-axis to position at 20
MCC_PtP(10, 20, 0, 0, 0, 0, 0, 0, g_nGroupIndex);

The point-to-point motion uses asynchronous motion, which means each axis use its own speed to move. After activating simultaneously, each axis will not necessarily reach the end point at the same time and this is different from the general motion. The general motion uses synchronous motion and each axis will arrive at the end point at the same time after activating. The following figures shows the point-to-point motion on a trajectory graph; at this point, all axes use the same speed.


y
(10, 20)
(0, 0)
x

12. Jog Motion

Related Commands

MCC_JogPulse()

MCC_JogSpace()

MCC_JogConti()

Example

JogMotion.cpp

Description

MCC_JogPulse() conducts the jog motion (pulse) on a specific axis in the unit of pulse not exceeding 2048 pulses of movement. MCC_JogSpace() conducts the jog motion (UU: User Unit) on a specific axis with the same unit of general motion. And MCC_JogConti() can move to the work area border set by mechanism parameters. The speed ratio is a necessary parameter for MCC_JogSpace() and MCC_JogConti(); the setting is similar to the point-to-point motion. The example is detailed as follows.

Step 1: Make the X axis move 100 pulses

MCC_JogPulse(100, 0, g_nGroupIndex) ;

Step 2: To move the X axis -1 User Unit distance by using the speed (RPM × Pitch / GearRatio) × 10%

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

Step 3: To move the X axis to the right border of work area by using the speed (RPM × Pitch / GearRatio) × 10%

MCC_JogConti(1, 10, 0, g_nGroupIndex);

13. In Position Control

Related Commands

MCC_SetInPosMaxCheckTime()

MCC_EnableInPos

MCC_SetInPosToleranceEx

MCC_GetInPosStatus

Example

InPosCheck.cpp

Description

This example uses the difference between encoder count (the actual machine position) and target position to check if each motion axis meets the in position confirmation criteria.

When the motion command is completed, the program will check whether the criteria of in position are met. If the check time exceeds the setting (such as the position tolerance of certain motion axes cannot meet the criteria of in position), then this position tolerance will be recorded and the execution of other motion commands will be stopped. The user can force the motor to produce error and observe its operation. The process of applying this function is as follows:

Step 1: Set the maximum in position confirmation check time; unit: ms
MCC_SetInPosMaxCheckTime(1000, g_nGroupIndex);

Step 2: Set the In Position Control mode
MCC_SetInPosMode(IPM_ONETIME_BLOCK, g_nGroupIndex);

Step 3: Set the tolerance of each axis; unit: mm or inch
MCC_SetInPosToleranceEx(0.5, 0.5, 1000, 1000, 1000, 1000, 1000, 1000, g_nGroupIndex);

Step 4: Enable the In Position Control function
MCC_EnableInPos(g_nGroupIndex);

Step 5: Get the In Position Control status of each axis; correct the status of in position is 0xff (255)

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

Step 6: Get the error code

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

14. Go Home Motion

Related Commands

MCC_SetHomeConfig()

MCC_Home()

MCC_GetGoHomeStatus()

MCC_AbortGoHome()

Example

GoHome.cpp

Description

The Go Home process will follow the settings of SYS_HOME_CONFIG in Go Home parameters. MCC_SetHomeConfig() can be used to set Go Home parameters (please refer to "**IMP Series Motion Control Command Library User Manual**").

MCC_GetGoHomeStatus() can be used to verify if the Go Home process has been completed. MCC_AbortGoHome() can be used to stop the Go Home motion forcibly.

Currently, the Go Home function provided by the MCCL can only target one motion control card at one time. If the user needs to operate multiple motion control cards, then it is necessary to use MCC_GetGoHomeStatus() to confirm the current Go Home motion has been completed so that MCC_Home() can be called to execute Go Home on the next motion control card. Following is the example of using this command:

Step 1: Set Go Home parameters

```
SYS_HOME_CONFIG     stHomeConfig;

stHomeConfig.wMode          = 3;    //  Set the Go Home mode
stHomeConfig.wDirection     = 1;    //  Set the Go Home motion in the
                                    //  negative direction
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: Set Go Home parameters

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

Step 3: 0xff means that the given axis does not need to execute Go Home

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

Step 4: This command can be used to stop the Go Home motion when it's necessary

```
MCC_AbortGoHome();
```

Step 5: Use the return value of this function to verify if the Go Home motion has been completed. If the value of nStatus equals 1, it means that the Go Home motion has been completed

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

15. Motion Hold, Continuation and Abortion

Related Commands

MCC_ HoldMotion()

MCC_ContiMotion()

MCC_AbortMotionEx()

Example

CtrlMotion.cpp

Description

MCC_HoldMotion() is used to pause the motion command currently being executed and MCC_ContiMotion() is used to continue executing the motion command being paused. Therefore, MCC_ContiMotion() needs to be used with MCC_HoldMotion() in the same group. MCC_AbortMotionEx() is used to set the deceleration time to pause motion and current remain motion commands will abort.

If there is no motion command currently being executed, the return value when calling MCC_HoldMotion() will be HOLD_ILLGEGAL_ERR; if MCC_HoldMotion() is unsuccessfully called, the return value when calling MCC_ContiMotion() will be CONTI_ILLGEGAL_ERR. Despite the current motion status, calling MCC_AbortMotionEx() will stop (decelerate) the motion and delete the stored motion commands in motion command buffer.

16. Forcibly Delaying Motion Command Execution

Related Commands

MCC_InitSystem()

MCC_DelayMotion()

Example

DelayMotion.cpp

Description

MCC_DelayMotion() can be used to forcibly delay the execution of the next motion command; the delay time is calculated in the unit of ms. In the following example, after completing the first command (Line), the next motion command will be executed after 3000 ms delay.

Step 1: Set the interpolation time as INTERPOLATION_TIME

nRet = MCC_InitSystem(INTERPOLATION_TIME, stCardConfig, 1);

Step 2: Start motion command

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

Step 3: Execute the next command after 3000 ms delay; please observe motion status

MCC_DelayMotion(3000);

17. Speed Override

Related Commands

MCC_OverrideSpeed()

MCC_GetOverrideRate()

MCC_OverrideSpeedEx()

Example

OverrideSpeed.cpp

Description

MCC_OverrideSpeed() can be used to set the feed speed ratio of linear, curve, circular and helix motions. The required parameter is the updated speed as the percentage of original speed × 100. MCC_GetOverrideRate() can be used to get current feed speed ratio. The example is detailed as follows.

Step 1: Set the feed speed of linear, curve, circular and helix motions as 20 mm/sec

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

Step 2: Set motion feed speed ratio; current speed will change to 20 × 150% = 30 mm/sec

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

Step 3: Get feed speed ratio; dfRate should equal 150

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

18. Software Over Travel Check and Hardware Limit Switch Check

Related Commands

MCC_SetOverTravelCheck()

MCC_GetOverTravelCheck()

MCC_EnableLimitSwitchCheck()

MCC_DisableLimitSwitchCheck()

MCC_GetLimitSwitchStatus()

Example

CheckOT.cpp

Description

The MCCL provides software over travel check (also referred to as software limit protection). When software over travel check is enabled, the system will stop the motion (and produce an error record) if the travel range of any axis exceeds the work area. The error record in the system must be cleared so that the system can move in the opposite direction and resume to normal operation. dfHighLimit and dfLowLimit in parameters are used to set the software location limits respectively. MCC_SetOverTravelCheck() is used to enable and disable this function while MCC_GetOverTravelCheck() is used to check current status set. The example is detailed as follows.

Step 1: Enable X axis software over travel check

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

Step 2: If over travel check has been set, OT0 ~ OT7 equal 1; otherwise the value equals 0

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

Step 3: Get the error code that may have produced

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

The return value of MCC_GetErrorCode() can be used to verify if the motion is currently unable to move (because an error record has been produced internally) since

its location has already exceeded software limits. If the return value is between 0xF301 to 0xF308, it means this situation is occurring on X axis to B axis correspondingly. Under this situation, the user can use the following example to make the system back to normal status.

Step 4: Clear the error record in the system to return the system to normal status
MCC_ClearError(g_nGroupIndex);

The MCCL also provides hardware limit switch check. To ensure the normal operation of limit switch, except the wiring of limit switch must be correctly set, it is also necessary to call MCC_EnableLimitSwitchCheck() so that settings of *wOverTravelUpSensorMode* and *wOverTravelDownSensorMode* will become effective. However, if *wOverTravelUpSensorMode* and *wOverTravelDownSensorMode* are set as 2, then it will be meaningless to call MCC_EnableLimitSwitchCheck().

If MCC_EnableLimitSwitchCheck(1) is used, then the group motion will only be stopped when the limit switch of moving direction of the given axis is triggered (for example, triggering the positive limit switch when moving in the forward direction or triggering the negative limit switch when moving in the backward direction); if MCC_EnableLimitSwitchCheck(0) is called, the group motion will be stopped once the limit switch is triggered (despite the moving direction).

The return value of MCC_GetErrorCode() can verify if the motion is stop when the limit switch has been triggered (because an error record has been produced internally). If the return value is between 0xF701 to 0xF708, it means this situation is occurring on X axis to B axis correspondingly. Under this situation, the user can use the following example to make the system back to normal status.

a. If the previous call was: MCC_EnableLimitSwitchCheck(0)
   then: exit the limit switch in the reverse direction
b. If the previous call was: MCC_EnableLimitSwitchCheck(1)
   then: exit the limit switch in the reverse direction
c. If the previous call was: MCC_EnableLimitSwitchCheck(2)
   then: MCC_ClearError() → exit the limit switch in the reverse direction

d. If the previous call was: MCC_EnableLimitSwitchCheck(3)

then: MCC_ClearError() → exit the limit switch in the reverse direction

19. Setting Path Blending

Related Commands

MCC_EnableBlend()

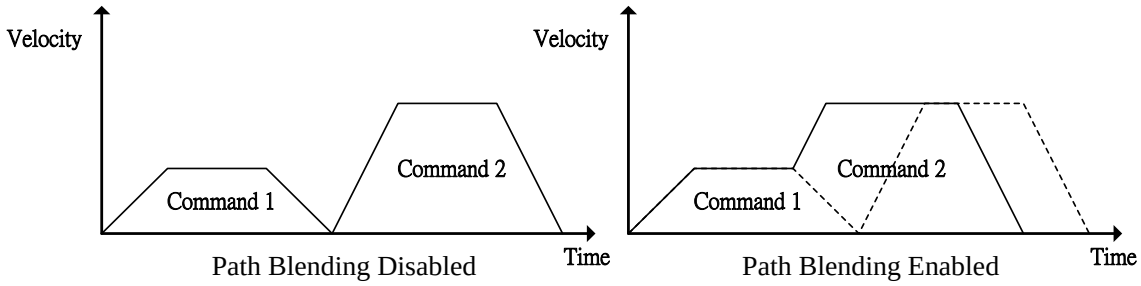
MCC_DisableBlend()

MCC_CheckBlend()

Example

SetBlend.cpp

Description



The figure shows the motion status after motion blending is enabled. The 1st motion command directly accelerate to the stable speed of the 2nd motion command from its own stable speed without decelerating (shown as the solid line in the figure). With this method, the command is executed faster while the trajectory error will exist at the connection points between each command.

Calling MCC_EnableBlend() and MCC_DisableBlend() can respectively enable and disable path blending. Calling MCC_CheckBlend() can get current status setting. If the return value of this command is 0, it means that path blending has been enabled; if the return value is 1, it means path blending has been disabled.

20. Getting and Clearing Error Status

Related Commands

MCC_GetErrorCode()

MCC_ClearError()

Example

ErrorStatus.cpp

Description

If the system error has been removed after occurring, it is still necessary to call MCC_ClearError() to clear the error record in the system. Otherwise, the system will be unable to execute the subsequent motions normally. Generally, the user should get the current error code during system operation to check if any error occurs. Following is an example. As for the application of these two commands, the user can also refer to the section "**Software Over Travel Check and Hardware Over Travel Check**".

The user can refer to the following command when handling the error.

```
if (MCC_GetErrorCode(g_nGroupIndex))
{
    /*
    Remove error status here
    */
    MCC_ClearError(g_nGroupIndex);//    clear the error record in the system
}
```

21. Gear Backlash and Gap Compensation

Related Commands

MCC_SetCompParam()

MCC_UpdateCompParam()

Example

Compensate.cpp

Description

The gear backlash and gap compensation function provided by the MCCL can compensate errors (such as the backlash error and gap error) resulting from the gear and screw during transmission. For a detailed description, please refer to "**IMP Series Motion Control Command Library User Manual**".

22. How to Complete 8-Axis Continuous Motion

Related Commands

MCC_CreateGroup()

MCC_SetFeedSpeed()

MCC_EnableBlend()

MCC_Line()

Example

SyncLine.cpp

Description

After one group is using MCC_EnableBlend() to enable path blending (fulfilling the conditions of path and speed continuity), although the 8-axis synchronization requirement (8 axes start and stop at the same time) can be met by calling MCC_Line() repeatedly, only the first 3 axes (X, Y, Z) can fulfill the conditions of path and speed continuity. The last 5 axes (U, V, W, A, B) can only fulfill the synchronization requirement.

When it is required to fulfill the conditions of 8-axis synchronization as well as path and speed continuity, three groups must be used. The 1st group is responsible for the trajectory of the first three axes, the 2nd group for the trajectory of the middle three axes and the 3rd group for the last two axes.

However, to satisfy the 8-axis synchronization requirement, the speed of the 2nd group can be converted by multiplying the ratio between required moving distances of the 2nd and the 1st group by the feed speed of the 1st group. The speed of the 3rd group can be converted by multiplying the ratio between the required moving distances of the 3rd and the 1st group by the feed speed of the 1st group.

The program code of this process is as follows. At this time, the user needs to call fnSyncLine() instead of using MCC_Line().

Step 1: Declare fnSyncLine

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

Step 2: Set and use three groups

```
int 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: Enable path blending

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

Step 4: Call fnSyncLine

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

Step 5: Define fnSyncLine

```
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);

        //    From the group definition, execute motion command of X, Y ,Z axis
        //    from group 1
        MCC_Line(x, y, z, 0, 0, 0, g_nGroupIndex0);

        //    Calculate the speed of the U, V, W axis
        dfDistance1 = u * u + v * v + w * w;
        dfUVWSpeed = dfXYZSpeed * sqrt(dfDistance1 / dfDistance0);
        MCC_SetFeedSpeed(dfUVWSpeed, g_nGroupIndex1);
```

```
            //  From the group definition, execute motion command of U, V ,W axis
            //  from group 2
                    MCC_Line(u, v, w, 0, 0, 0, g_nGroupIndex1);

            //  Calculate the speed of the A, B axis
            dfDistance2 = a * a + b * b;
            dfABSpeed = dfXYZSpeed * sqrt(dfDistance2 / dfDistance0);
            MCC_SetFeedSpeed(dfABSpeed, g_nGroupIndex2);

            //  From the group definition, execute motion command of A, B axis from
            //  group 3
            MCC_Line(a, b, 0, 0, 0, 0, g_nGroupIndex2);
        }
}
```

23. Triggering Interrupt of Encoder Count

Related Commands

MCC_SetENCRoutine()

MCC_SetENCCompValue()

MCC_EnableENCCompTrigger()

MCC_DisableENCCompTrigger()

MCC_SetENCInputRate()

MCC_GetENCValue()

Example

ENCCompare.cpp

Description

The command allowing the encoder count to trigger the ISR provided by the MCCL can set the comparative value of the encoder count. After enabling this function, when the encoder counter value equals to the set comparative value ((MCC_GetENCValue() can be used to get the encoder counter value), the MCCL will automatically call the ISR function that created by the user. The example is detailed as follows.

Step 1: Declare the ISR

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

Step 2: Serially connect the ISR

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

Step 3: Set the comparative value as 20000 pulses

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

Step 4: Enable encoder comparison interrupt function

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

Step 5: Define the ISR

```
void _stdcall ENC_ISR_Function(ENCINT_EX *pstINTSource)
{
    //  Determine whether the trigger source was the comparative conditions in
    //  Channel 0
    if (pstINTSource->COMP0)
      //  Abort motion commands currently being executed and those in the
      //  motion command buffer
      MCC_AbortMotionEx(0, g_nGroupIndex);

    ENC_ISR++;

    //  Disable encoder comparison interrupt function
    MCC_DisableENCCompTrigger(CHANNEL_INDEX);
}
```

This example shows that after the encoder count triggering ISR is enabled, a linear motion will be executed. When the encoder counter value reaches 20000 pulses, the unfinished linear motion will be stopped. Set the 1st parameter of MCC_AbortMotionEx as 0 to make the deceleration time equals 0 and allow the encoder counter value to approach 20000 pulses once it has stopped.

24. Encoder Counter Latch and Index Signals Trigger Interrupt

Related Commands

MCC_SetENCRoutine()

MCC_GetENCValue()

MCC_SetENCLatchType(()

MCC_SetENCLatchSource()

MCC_EnableENCIndexTrigger()

Example

GetENCLatch.cpp

Description

The encoder counter latch function provided by the MCCL can use MCC_SetENCLatchSource() to set a trigger conditions (sources); after the trigger conditions and latch mode are met (use MCC_SetENCLatchType() to set trigger mode), the encoder counter can be recorded in the latch register. MCC_GetENCLatchValue() can be used to get the value in latch register. The example is detailed as follows.

Step 1: Set the encoder counter latch mode

ENC_TRIG_FIRST	When trigger conditions are satisfied for the first time, the encoder counter value will be latched.
ENC_TRIG_LAST	When trigger conditions are satisfied, the encoder counter value will be latched; when conditions are repeatedly satisfied, then the new encoder counter value will be latched repeatly.

MCC_SetENCLatchType(ENC_TRIG_LAST, CHANNEL_INDEX,
CARD_INDEX);

Step 2: Set the encoder trigger source. There are 16 trigger sources (conditions) can be used to latch the encoder counter value. It is possible to unite multiple conditions at the same time during setting. At this time, select the encoder index signal as the trigger source (condition)

MCC_SetENCLatchSource(ENC_TRIG_INDEX0, CHANNEL_INDEX,

CARD_INDEX);

This function above shows that the encoder index signal can be used as the trigger source (condition). When the command allowing the encoder index signal to trigger the ISR is enabled, MCC_GetENCLatchValue() can be used to get the value in latch register after the encoder index signal occurs. To use this function, the user must serially connect the customized ISR as well as enable it first.

Step 3: Declare the ISR

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

Step 4: Serially connect the ISR

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

Step 5: Enable the function allowing the encoder index signal to trigger the ISR

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

Step 6: Define the ISR

```
void _stdcall ENC_ISR_Function(ENCINT_EX *pstINTSource)
{
    if (pstINTSource->INDEX0)       //    Verify if the trigger source is index signal
    {
        //    Get the value recorded in latch register
        MCC_GetENCLatchValue(&lLatchValue, CHANNEL_INDEX,
                                        CARD_INDEX);
    }
}
```

For a detailed description, please refer to the "**IMP Series Motion Control Command Library User Manual**".

25. Triggering Interrupt with Local Input and Output Signal Control

Related Commands

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

Example

LIOTrigger.cpp

Description

The Local I/O provided by the MCCL includes servo on/off, position ready output signal control and check for home sensor and hardware limit switch input signal.

All input connection signals can trigger the customized ISR. The procedure of using "Input Signal Triggered Interrupt Service Routine" is detailed below:

Step 1: Use MCC_SetLIORoutine() to set the customized ISR of LIO
It is required to program the customized ISR and the routine declaration must follow the following prototype:

```
typedef void(_stdcall *LIOISR)(LIOINT*)
```

For example, the customized function can be designed as follows:

```
_stdcall MyLIOFunction(LIOINT *pstINTSource)
{
```

```
    //  Determine whether this ISR is triggered by touching channel 0 limit switch+
    if (pstINTSource->OTP0)
    {
        //  handling procedure when channel 0 limit switch+ is triggered
    }

    //  Determine whether this ISR is triggered by touching channel 1 limit switch+
    if (pstINTSource->OTP1)
    {
        //  handling procedure when channel 1 limit switch+ is triggered
    }
}
```

A routine such as ”else if (pstINTSource->OTP1)” cannot be used, because pstINTSource->OTP0 and pstINTSource->OTP1 may not be 0 simultaneously.

Later, use MCC_SetLIORoutineEx (MyLIOFunction) to set the customized ISR of LIO. When the customized is triggered during execution, the system can use the pstINTSource parameter declared as LIOINT in the customized routine to determine which input is triggered when the customized routine was called. The definition of LIOINT is as follows:

```
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;
```

If the value of these fields is not equal 0, then currently the connection corresponding to that field has a signal input. For example, if the MyLIOFunction() input parameter pstINTSource->OTP2 is not 0, the channel 2 limit switch+ is ON.

Step 2: Use MCC_SetLIOTriggerType() to set the trigger type

The trigger type can be set as Rising Edge, Falling Edge and Level Change. The input parameters of MCC_SetLIOTriggerType() can be:

LIO_INT_RISE	Rising edge (default)
LIO_INT_FALL	Falling edge
LIO_INT_LEVEL	Level change

Step 3: Finally, use MCC_EnableLIOTrigger() to enable the “Input Signal Triggered Interrupt Service Routine” function.
MCC_DisableLIOTrigger() can be used to disable this function.

26. Timer Triggered Interrupt Service Routine

Related Commands

MCC_SetTMRRoutine();

MCC_SetTimer()

MCC_EnableTimer()

MCC_EnableTimerTrigger()

Example

TimerTrigger.cpp

Description

The length of the 32-bit timer on IMP Series motion control card can be set by using the MCCL. When the timer function is enabled and the timing is completed (i.e. the value of timer equals the set value), it will trigger the customized ISR and restart timing. This process will continue until this function is disabled. The procedure of using "timer triggered interrupt service routine" is detailed below:

Step 1: Use MCC_SetTMRRoutine() to set the customized ISR of timer
It is required to program the customized ISR and the routine declaration must follow the following prototype:

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

For example, the customized command can be designed as follows:

```
stdcall MyTMRFunction(TMRINT *pstINTSource)
{
    //  determine whether the routine was triggered because the timer ends timing
    if (pstINTSource->TIMER)
    {
        //  handling procedure when the timer ends timing
     }
}
```

Later, use MCC_SetTMRRoutineEx(MyTMRFunction) to set the customized ISR of timer. When the customized is triggered during execution, the system can use the pstINTSource parameter declared as TMRINT in the customized routine to determine which input connection is triggered when the customized routine was called. The definition of TMRINT is as follows:

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

If the value in timer is not 0, it means the timer is time out.

Step 2: Use MCC_SetTimer() to set the timer; timing unit: 1us

Step 3: Use MCC_EnableTimer() to enable the timer function

Step 4: Use MCC_EnableTimerTrigger() to enable “Timer Triggered Interrupt Service Routine” function.

27. Setting Watchdog

Related Commands

MCC_SetWatchDogTimer()

MCC_SetWatchDogResetPeriod()

MCC_EnableWatchDogTimer()

Example

WatchDog.cpp

Description

After the user has enabled the watchdog function, it is necessary to use MCC_RefreshWatchDogTimer() to clear the watchdog timer before the timing ends (i.e. before the timing value of watchdog equals the set comparative value). Otherwise, once the timing value of watchdog equals to the set comparative value, the hardware will be reset. The procedure of using watchdog is as follows:

Step 1: Use MCC_SetWatchDogTimer() to set the comparative value of watchdog timer; unit: 1us and the setting range is between 1 to $2^{32.}$

In other words, if the following programming code is used:

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

At this point the comparative value of watchdog timer for card 0 is set as 1us * 10000000 = 10s.

Step 2: Use MCC_SetWatchDogResetPeriod() to set the reset signal duration.

This function can program the reset hardware duration produced by the watchdog function, using the unit of system clock (10ns).

Step 3: MCC_RefreshWatchDogTimer() must be used to clear the watchdog timer before the watchdog timing ends.

The user can combine this function with the “timer triggered interrupt service routine” function. The user will be alerted before the watchdog resets the hardware and

conduct the necessary handling within the timer ISR.

Note: When timer triggered interrupt service routine is combined to handle watchdog reset, it is necessary to set the escape time of timer < watchdog timer in advance. Otherwise, the system will be reset if the escape time of timer > watchdog timer.

28. Setting and Getting Remote Input and Output Signal

Related Commands

MCC_EnableARIOSetControl()

MCC_EnableARIOSlaveControl()

MCC_GetARIOInputValue()

MCC_SetARIOOutputValue()

Example

ARIOCtrl.cpp

Description

Each IMP-2 consists of one set of IMP-ARIO/ IMP-ARIO64 connectors (referred to as Async Remote I/O Master terminal, number RIO_SET0) and can simultaneously control IMP-ARIO/ IMP-ARIO64 with up to 512 Inputs and 512 Outputs. (referred to as Async Remote I/O Slave terminal, number RIO_SLAVE0 ~ RIO_SLAVE31). Each IMP-ARIO provides 16 output and 16 input connections. Each IMP-ARIO64 provides 32 output and 32 input connections.

EnableARIOSetControl() and EnableARIOSlaveControl() can be used to enable data transmission. The example is as follows. At this point, the 1st IMP-ARIO/ IMP-ARIO64 (number 0) and the data transmission of its slave terminal are enabled.

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

After completing the initial setting, use low potential (ECOM-) to contact connectors. MCC_GetARIOInputValue() can get the input signal status then. MCC_SetARIOOutputValue() can also be used to set the output signal status.

29. Getting Remote Input and Output Signal Transmission Status

Related Commands

MCC_EnableARIOSetControl()

MCC_EnableARIOSlaveControl()

MCC_GetARIOTransStatus()

MCC_GetARIOMasterStatus()

MCC_GetARIOSlaveStatus()

Example

ARIOStatus.cpp

Description

MCC_GetARIOTransStatus() can be used to monitor the data transmission status of each remote I/O set at any time. When the error of data transmission occurs, the information obtained by MCC_GetARIOMasterStatus() and MCC_GetARIOSlaveStatus() can be used to identify whether the error message comes from the IMP-2, IMP-ARIO or IMP-ARIO64.

If the status obtained by MCC_GetARIOTransStatus(), MCC_GetARIOMasterStatus() and MCC_GetARIOSlaveStatus() equals 1, it means the data transmission status is normal; if the status is 0, it means the data transmission error has occurred.

The example is detailed as follows.

```
WORD wTransStatus;

//  Get transmission status
MCC_GetARIOTransStatus( &wTransStatus, RIO_SET0, CARD_INDEX);
```

If wTransStatus equals 1, it means the data transmission status is normal. If the status is 0, it means the data transmission error has occurred.

30. Digital to Analog Converter Voltage Output

Related Commands

MCC_StartDACConv()

MCC_SetDACOutput()

Example

DACOutput.cpp

Description

Suppose a certain motion axis does not use voltage command operation mode, then the corresponding D/A output channel of that axis can be used as the general analog voltage output channel.

Use MCC_StartDACConv() to start DAC conversion. After successfully calling MCC_InitSystem(...), the MCCL will call this function automatically. Finally, use MCC_SetDACOutput() to output the voltage.

31. Analog and Digital Converter Voltage Input: Single Conversion

Related Commands

MCC_SetADCConvMode()

MCC_SetDACOutput()

MCC_SetADCSingleChannel()

MCC_StartADCConv()

Example

ADC1Time.cpp

Description

This example uses ADC Channel 0 to conduct a single voltage and bipolar mode (-5V ~ 5V) and get the input voltage value. The process of applying this function is as follows:

Step 1: Set conversion mode as single voltage conversion
MCC_SetADCConvMode(ADC_MODE_SINGLE, CARD_INDEX);

Step 2: Set voltage conversion type as bipolar mode (-5V ~ 5V)
MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 3: Set the channel for single voltage conversion
MCC_SetADCSingleChannel(0, CARD_INDEX);

Step 4: Start single voltage conversion
MCC_StartADCConv(CARD_INDEX);

When using single voltage conversion mode, if the user intends to get the new voltage value, it is required to call MCC_StartADCConv(CARD_INDEX) again; it is also possible to use MCC_GetADCWorkStatus() to verify if signal voltage conversion has been completed.

32. Analog and Digital Converter Voltage Input: Continual Conversion

Related Commands

MCC_SetADCConvMode()

MCC_SetADCConvType()

MCC_EnableADCConvChannel()

MCC_StartADCConv()

Example

ADCInput.cpp

Description

This example uses ADC Channel 0 to conduct continuous voltage and bipolar mode (-5V ~ 5V) and get the input voltage value. The process of applying this function is as follows:

Step 1: Set conversion mode as continuous voltage conversion
MCC_SetADCConvMode(ADC_MODE_FREE, CARD_INDEX);

Step 2: Set voltage conversion type as bipolar mode (-5V ~ 5V)
MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 3: Enable Channel 0 voltage conversion
MCC_EnableADCConvChannel(0, CARD_INDEX);

Step 4: Start voltage conversion
MCC_StartADCConv(CARD_INDEX)

33. Analog to Digital Converter Comparator Interrupt Control

Related Commands

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

Example

ADCComp.cpp

Description

This example sets the comparative value of ADC Channel 0 comparator. When comparison conditions are met, with the voltage passing from high to low, the customized ISR will be triggered. This example will run ADC conversion continuously, meaning that when comparison conditions are met, the interrupt will be triggered continuously. The process of applying this function is as follows:

Step 1: Set the customized ISR of ADC
MCC_SetADCRoutine(ADC_ISR_Function, CARD_INDEX);

The customized ISR can be defined as follows:

```
void _stdcall ADC_ISR_Function(ADCINT *pstINTSource)    //    ADC ISR
{
    if (pstINTSource->COMP0)    //    if comparison conditions are satisfied
        nISRCount++;
}
```

Step 2: Set conversion mode as continuous conversion

MCC_SetADCConvMode(ADC_MODE_FREE, CARD_INDEX);

Step 3: Set voltage conversion type as bipolar mode (-5V ~ 5V)

MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 4: Set the comparative value of voltage comparator

MCC_SetADCCompValue(2.0, 0, CARD_INDEX);

Step 5: Set voltage comparison conditions as from high to low voltage

MCC_SetADCCompType(ADC_COMP_FALL, 0, CARD_INDEX);

Step 6: Enable voltage comparator to trigger the customized ISR

MCC_EnableADCCompTrigger(0, CARD_INDEX);

Step 7: Enable Channel 0 voltage conversion

MCC_EnableADCConvChannel(0, CARD_INDEX);

Step 8: Start voltage conversion

MCC_StartADCConv(CARD_INDEX)