EPCIO Series

Motion Control Command

Library Example Manual

(Applicable to Motion Control Command Library V.5.10)

Version: V.5.10

Date: 2009.10

http://www.epcio.com.tw

Table of Contents

1. Description of Motion Control Command Library Examples

The examples provided of the installation CD-ROM are console modes that users can integrate into their own applications. While the Motion Control Command Library (MCCL) can support a maximum of 12 EPCIO Series motion control cards and 72 groups, most examples only use 1 motion control card (motion control card number ***CARD_INDEX***) and 1 group (group number ***g_nGroupIndex***) to increase 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 Programming

InitSys.cpp

Description

This example describes the process for setting the group, mechanism, and encoder parameters. First use MCC_SetSysMaxSpeed() to set the maximum feed rate. Then use MCC_SetMacParam() and MCC_SetEncoderConfig() to set the mechanism and encoder parameters for each axis. Finally, use MCC_CreateGroup() to establish a new group.

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

3. Interpolation Time Adjustment

Related Commands

MCC_InitSystem()

MCC_GetCurPulseStockCount()

Example Program

CheckHWStock.cpp

Description

Shorter interpolation time create better motion control performance. While the interpolation time can be set to a minimum of 1 ms, this minimum interpolation time is not applicable to all PCs, as it is related to PC performance. To obtain the most appropriate interpolation time, use MCC_GetCurPulseStockCount() to acquire the pulse stock count in the EPCIO Series motion control card. A pulse stock count greater than or equal to 60 is required in a continuous motion process to guarantee stable motion performance. If the stock count appears to equal 0, the interpolation time must be extended (interpolation time is a necessary parameter for MCC_InitSystem()). The interpolation time must also be extended if a delay occurs on the user interface operations screen.

4. Motion Control Command Library Open and Closed

Related Commands

MCC_InitSystem()

MCC_CloseSystem()

MCC_GetMotionStatus()

Example Program

InitSys.cpp

Description

After completing setup of the group and mechanism parameters, use MCC_InitSystem() to initiate the motion control command library. For the necessary parameters, please refer to "**EPCIO Series Motion Control Command Library User Manual.**" Directions for this example are outlined below:

Step 1: Provide control card hardware parameters

```
SYS_CARD_CONFIG stCardConfig[MAX_CARD_NUM];
..
stCardConfig[CARD_INDEX].wCardAddress = BASE_ADDRESS
stCardConfig[CARD_INDEX].wCardType = wCardType;
stCardConfig[CARD_INDEX].wIRQ_No = IRQ_NO;
```

Step 2: Initiate MCCL

```
nRet = MCC_InitSystem(INTERPOLATION_TIME, // set interpolation time to 10ms
                      stCardConfig,       // hardware parameters
                      1);                 // use only 1 EPCIO card

if (nRet == NO_ERR)// motion control command library initiation is successful
{
    /*
    This can be used to perform other actions for initialization, such as
    setting the unit of movement or the feed rate.
    */
}
```

Step 3:

MCC_CloseSystem() is used to disable the MCCL and the driver command library. Two methods can be used to disable the system:

i. System shutdown after the entire motion command is completed

First examine whether the system status is at “stop”. If the returned value from the command MCC_GetMotionStatus() is GMS_STOP, the system has stopped.

```
while ((nRet = MCC_GetMotionStatus(g_nGroupIndex)) != GMS_STOP)
{
MCC_TimeDelay(1); // Sleep 1 ms
// because the “while” command was used to avoid system lockup, impacting system operations,
// MCC_TimeDelay () is required to free the CPU usage rights.
}

MCC_CloseSystem(); // shuts down MCCL system
```

ii. Directly shutdown the motion control library

Only MCC_CloseSystem() is required to immediately stop system operations.

5. Setting System Status

Related Commands

MCC_SetUnit()
MCC_GetUnit()
MCC_SetAbsolute()
MCC_SetIncrease()
MCC_Get_CoordType()
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 Program

SetStatus.cpp

Description

This example describes how to change the system status. If the system status is not specified, the system will use the default status operations. For the system's default status, please refer to "**EPCIO Series Motion Control Command Library Reference Manual.**" The commands are described below.

MCC_SetUnit(UNIT_MM, g_nGroupIndex); // use mm as the unit of movement

MCC_SetAbsolute(g_nGroupIndex); // use absolute coordinate terms to express the position of each axis

// use the T curve for the line, curve, and circular motion acceleration types
MCC_SetAccType ('T', g_nGroupIndex);

```
// use the S curve for the line, curve, and circular motion deceleration types
MCC_SetDecType('S', g_nGroupIndex);

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

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

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

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

Enabling servo system requires MCC_SetServoOn() for the system to operate normally. Consider the actual situation to determine whether MCC_EnablePosReady() is required.

6. Acquisition of Information Regarding Motion Speed, Coordinates, and Motion Commands

Related Commands

MCC_GetCurFeedSpeed()

MCC_GetFeedSpeed()

MCC_GetCurPos()

MCC_GetPulsePos()

MCC_GetCurCommand()

MCC_GetCommandCount()

Example Program

GetStatus.cpp

Description

MCC_GetCurFeedSpeed() can be used to acquire the current feed rate; MCC_GetSpeed() can then be used to obtain the current feed rates for each axis. MCC_GetCurPos() can be used to acquire the cartesian coordinate values for the current positions of each axis; MCC_GetPulsePos() can then be used to obtain the motor coordinate values (also referred to as pulse coordinate values) for the current positions of each axis. Cartesian coordinate values and motor coordinate values can also be obtained using mechanism parameter conversion, or motor coordinate value = cartesian coordinate value × (dfGearRatio / dfPitch) × dwPPR. The coordinates for each axis acquired using MCC_GetCurPos() and MCC_GetPulsePos() only have significance when the given axis corresponds to a hardware output channel.

The example used is below:

Step 1: Declare the variables

double dfCurPosX, dfCurPosY, dfCurPosZ, dfCurPosU, dfCurPosV, dfCurPosW, dfCurSpeed;

double dfCurSpeedX, dfCurSpeedY, dfCurSpeedZ, dfCurSpeedU, dfCurSpeedV, dfCurSpeedW;

long lCurPulseX, lCurPulseY, lCurPulseZ, lCurPulseU, lCurPulseV, lCurPulseW;

Step 2: Acquire the current feed rate

dfCurSpeed = MCC_GetCurFeedSpeed(g_nGroupIndex);

Step 3: Acquire the current feed rate for each axis

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

Step 4: Acquire the cartesian coordinates for the current position of each axis

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

Step 5: Acquire the motor coordinates for the current position of each axis

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

MCC_GetCurCommand() can obtain information related to operational motion commands currently being executed, including the motion command type, the motion command code, the feed rate, and the destination position. MCC_GetCommandCount() can obtain the items in the command buffer yet to be executed .

7. Motion Status Inspection

Related Commands

MCC_GetMotionStatus()

Example Program

MotionFinished.cpp

Description

The value returned by the command MCC_GetMotionStatus() check the machine's current motion status. If the value returned is GMS_RUNNING, the machine is running. If the value returned is GMS_STOP, the machine has stopped and no commands are in the command buffer. If MCC_HoldMotion() is successfully called and the returned value for the command MCC_GetMotionStatus() is GMS_HOLD, the machine is on temporary hold with unexecuted motion commands. If the value returned is GMS_DELAYING, the system is currently delayed because MCC_DelayMotion() had been called. An example of command usage follows below:

Step 1: Declare the motion status parameters acquired

```
int nStatus;
```

Step 2: Enable servo

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

Step 3: Straight line motion

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

Step 4: Wait for MCC_Line() to be completed. Execute the following command after post-production GMS_STOP exits the back of the loop

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

Step 5: Delay motion command with the motion status GMS_DELAYING

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

Step 6: Do the line moation again, the motion status will be changed.

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

Step 7: Press the H button to temporarily hold motion. Motion status will appear as GMS_HOLD

nRet = MCC_HoldMotion(g_nGroupIndex);

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

nRet = MCC_ContiMotion(g_nGroupIndex);

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

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 Program

AccStep.cpp

Description

The default acceleration and deceleration time for general motion (including line, curve, and circular motion) and point-to-point motion are 300 ms. However, this time can be adjusted using MCC_SetAccTime(), MCC_SetDecTime(), MCC_SetPtPAccTime(), and MCC_SetPtPDecTime() to ensure a steady acceleration or deceleration process.

Different speeds should be applied to 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 the use of different motors and mechanisms. The following formulas can be used to obtain the acceleration and deceleration time:

operational acceleration time = required speed/required acceleration

operational deceleration time = required speed/required deceleration

9. Setting Feed Rates

Related Commands

MCC_SetFeedSpeed()

MCC_GetFeedSpeed()

MCC_SetPtPSpeed()

MCC_GetPtPSpeed()

Example Program

SetSpeed.cpp

Description

The feed rate must be set before conducting line, curve, and circular motion. All set feed rates should not exceed the MCC_SetSysMaxSpeed() set value.

Use MCC_SetFeedSpeed() to set the feed rates for line, curve, circular, and helix motion. For example, when MCC_SetFeedSpeed (20, g_nGroupIndex) is called, the feed rate is 20 mm/sec or 20 inch/sec, depending on the unit.

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

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

Related Commands

MCC_SetAbsolute()

MCC_SetFeedSpeed()

MCC_Line()

MCC_ArcXY()

MCC_CircleXY()

Example Program

GeneralMotion.cpp

Description

After the group, mechanism, and encoder parameters have been set, the system initiated, the maximum feed rate set, the servo circuit enabled (this action is unnecessary when using a stepper motor), and the feed rate set, then line, curve, circular, and helix motion can be conducted. When using curve commands, ensure that the given parameters are proper (the point of origin, reference point, and destination point cannot be located on the same line). Below is an example of the commands.

Step 1: Use absolute coordinate terms to express the position for each axis and to set the feed rate

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

Step 2: Execute the line motion command

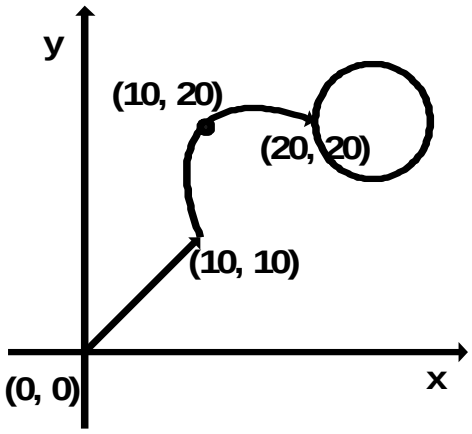
```
MCC_Line(10, 10, 0, 0, 0, 0, g_nGroupIndex);
```

Step 3: Execute curve motion, please make sure that the points of origin, reference, and destination are not located on the same line

```
nRet = MCC_ArcXY(10, 20, 20, 20, g_nGroupIndex);
if (nRet != NO_ERR)
```

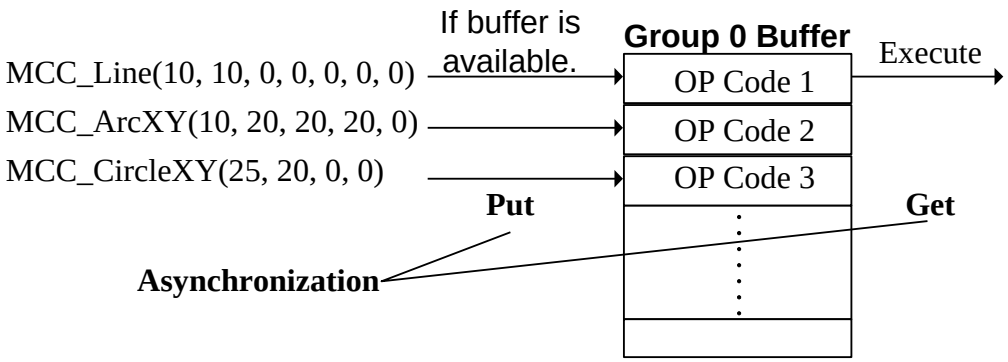
```
{
    /*
    Use the return value to understand the reason for errors. If a parameter error
    occurs, the return value will be PARAMETER_ERR.
    */
}
```

The return value from the command can be used to understand the reason for the error. For the meaning of the return value, please refer to "**EPCIO Series Motion Control Command Library Reference Manual.**"
The trajectory are displayed in the figure below.



Step 3: Execute circular command
MCC_CircleXY(25, 20, 0, g_nGroupIndex);

During motion command execution, the motion command first places the OP code in each group's exclusive motion command buffer. Then the MCCL simultaneously collects motion commands in order of execution from the buffers of different groups. These two actions are not synchronized, so it is unnecessary to wait for execution of the prior motion command to be completed before sending the new motion command to the motion command buffer.



If the motion command buffer is full, the command will return COMMAND_BUFFER_FULL_ERR. This motion command is not processed. Each motion command buffer has a default storage space of 10000 motion commands. The figure above displaying the Group 0 motion command buffer operational process shows that commands belonging to the same group will be executed in order.

Because each group has an exclusive motion command buffer zone, motion commands belonging to different groups can be executed simultaneously. For a detailed explanation, please refer to "**EPCIO Series Motion Control Command Library User Manual.**"

11. Point-to-Point Motion

Related Commands

MCC_SetAbsolute()

MCC_SetPtPSpeed()

MCC_PtP()

Example Program

PtPMotion.cpp

Description

After the group, mechanism, and encoder parameters have been set, the system initiated, the maximum feed rate set, the servo circuit enabled (this action is unnecessary when using a stepper motor), and the feed rate set, then point-to-point motion can be conducted. Below is an example use of the commands.

Step 1: Use absolute coordinates and set the feed rate

MCC_SetAbsolute(g_nGroupIndex);

MCC_SetFeedSpeed(20, g_nGroupIndex);

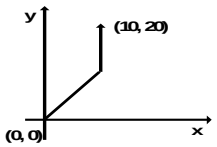
Step 2: Set each axis to 20 % maximum speed, which (RPM × Pitch / GearRatio) × 20 %

MCC_SetPtPSpeed(20, g_nGroupIndex);

Step 3: Each axis moves asynchronously (10, 20)

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

Point-to-point motion uses asynchronously motion. Each axis uses its own speed motion. Even if all axes are initiated simultaneously, they will not necessarily arrive at the synchronously destination at the same time. General motion, however, uses the synchronously motion, meaning that if all axes are initiated simultaneously, they will arrive at the synchronously destination at the same time. The following figure below displays the point-to-point trajectory when the speed of each axis is identical.



12. JOG Motion

Related Commands

MCC_SetUnit()

MCC_JogPulse()

MCC_JogSpace()

MCC_JogConti()

Example Program

JogMotion.cpp

Description

MCC_JogPulse() conducts pluse-motion on a specific axis in units of pulse not exceeding 2048 pulses of movement. MCC_JogSpace() conducts inch-motion on a specific axis using the same units as general motion. MCC_JogConti() can then move the axis to the work area border as set by mechanism parameters. The necessary parameters for MCC_JogSpace() and MCC_JogConti() include speed ratio, setting mode, and type of point-to-point movement. An example is outlined below:

Step 1: Set mm as the unit of movement

MCC_SetUnit(UNIT_MM, g_nGroupIndex);

Step 2: Move the X axis 100 pulses

MCC_JogPulse(100, 0, g_nGroupIndex) ;

Step 3: Use the speed (RPM × Pitch / GearRatio) × 10 % to move the X axis -1 mm

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

Step 4: Use the speed (RPM × Pitch / GearRatio) × 10 % to move the X axis to the right border of the work area

MCC_JogConti(1, 10, 0, g_nGroupIndex);

13. Position Control

Related Commands

MCC_SetInPosMaxCheckTime()

MCC_EnableInPos()

MCC_SetInPosToleranceEx()

MCC_GetInPosStatus()

Example Program

InPosCheck.cpp

Description

This example program uses the error between the encoder count (actual machine location) and the destination to inspect whether each motion axis meets the position confirmation criteria.

Position check will begin when the motion command is complete. If check time exceeds the set value, and if certain motion axis location errors are still unable to meet the position criteria, the situation will be recorded and other movement command executions will be stopped. Users can force motor error generation and observe operations by following the procedure below:

Step 1: Set the maximum position confirmation check time in units of ms

MCC_SetInPosMaxCheckTime(1000, g_nGroupIndex);

Step 2: Set the position control mode

MCC_SetInPosMode(IPM_ONETIME_BLOCK, g_nGroupIndex);

Step 3: Set the error value for each axis in units of mm or inches

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

Step 4: Enable position control

MCC_EnableInPos(g_nGroupIndex);

Step 5: Acquire the position control status for each axis, with an accurate positional status of 0xff(255)

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

Step 6: Obtain the error code

nErrCode = MCC_GetErrorCode(g_nGroupIndex);

14. Go Home Motion

Related Commands

MCC_SetHomeConfig()

MCC_Home()

MCC_GetGoHomeStatus()

MCC_AbortGoHome()

Example Program

GoHome.cpp

Description

The Go Home procedure depends on the SYS_HOME_CONFIG settings in the Go Home parameters. MCC_SetHomeConfig() can be used to set these parameters (please refer to "**EPCIO Series Motion Control Command Library User Manual**").

The MCC_GetGoHomeStatus() command can be used to acquire the status of completion of the Go Home procedure, and MCC_AbortGoHome() can be called during the Go Home process to forcefully stop the motion.

Currently, the Go Home function provided by the MCCL can only target one motion control card at a time. If multiple cards need to be operated, MCC_GetGoHomeStatus() must confirm that the current Go Home execution has been completed before MCC_Home() can be called to execute Go Home on the next card. An example is outlined below:

Step 1: Set the Go Home parameters

```
SYS_HOME_CONFIG stHomeConfig;

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

```
for (WORD wChannel = 0;wChannel < 6;wChannel++)
    MCC_SetHomeConfig(&stHomeConfig, wChannel, CARD_INDEX);
```

Step 3: 0xff means Go Home is not required for this axis

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

Step 4: If it is required, this command can be used to stop the Go Home motion

```
MCC_AbortGoHome();
```

Step 5: Use the returned value from this command to determine the status of completion of the Go Home motion; if nStatus equals 1, the Go Home motion is complete

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

15. Motion Hold, Continue, and Abort

Related Commands

MCC_ HoldMotion()

MCC_ContiMotion()

MCC_AbortMotionEx()

Example Program

CtrlMotion.cpp

Description

MCC_HoldMotion() is used to hold the motion command currently being executed. MCC_ContiMotion() is then used to continue executing the motion command being held. Therefore, MCC_ContiMotion() must be used in combination with MCC_HoldMotion() in the same group. MCC_AbortMotionEx() sets the decelerate/stop time and aborts the motion command being held or executed.

Currently, if MCC_HoldMotion() is called while no motion command is being executed, the returned value will be HOLD_ILLEGAL_ERR. Previously, if MCC_ContiMotion() was called when MCC_HoldMotion() had been unsuccessful, the returned value would be CONTI_ILLEGAL_ERR. Regardless of the current motion status, calling MCC_AbortMotionEx() will (decelerate) stop motion and clear the stored command from the command buffer.

16. Forced Delay Motion Command

Related Commands

MCC_InitSystem()

MCC_DelayMotion()

Example Program

DelayMotion.cpp

Description

MCC_DelayMotion() can be used to forcefully delay an execution of the subsequent motion command. The delay is calculated in terms of ms. In the following example, a 3000 ms delay occurs after the first command is completed, before the next command can be executed.

Step 1: Set the interpolation time to INTERPOLATION_TIME

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

Step 2: Initiate motion command

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

Step 3: Delay execution of the next command line for 3000 ms; please observe motion status

```
MCC_DelayMotion(3000);
```

17. Speed Override

Related Commands

MCC_SetOverrideSpeed()

MCC_GetOverrideRate()

MCC_OverridePtPSpeed()

MCC_GetPtPOverrideRate()

Example Program

OverrideSpeed.cpp

Description

MCC_OverrideSpeed() sets the speed override ratio for line, curve, circular, and helix motion. The updated speed as a percentage of the original speed × 100 is a necessary parameter. MCC_GetOverrideRate() can then be used to obtain the current speed override ratio. An example of this is outlined below:

Step 1: Set the feed rate for line, curve, circular, and helix motion to 20 mm/ sec
MCC_SetFeedSpeed(20, g_nGroupIndex);
MCC_Line(10, 10, 0,0,0,0,0, g_nGroupIndex)

Step 2: Set the motion speed override ratio, changing the current speed to 20 ×150 % = 30 mm /sec
MCC_OverSpeed(150, g_nGroupIndex);

Step 3: Acquire the override 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 Program

CheckOT.cpp

Description

The MCCL provides software over travel check (also referred to as software limit protection). When software over travel check is enabled, if the range of advancement for any axis exceeds work area borders, the system will stop motion (producing a record of the error). The record of the error must be deleted from the system before the system can move in the opposite direction and resumes normal status. Mechanism parameters dfHighLimit and dfLowLimit each set the software location limits. MCC_SetOverTravelCheck() enables and disables over travel check, while MCC_GetOverTravelCheck() is used to check the current set status. An example of the use of this command is outlined below:

Step 1: Enable X axis software over travel check
MCC_SetOverTravelCheck (1, 0, 0, 0, 0, 0, g_nGroupIndex);

Step 2: If over travel check is set, OT0 – OT5 equals 1; otherwise it equals 0
MCC_GetOverTravelCheck(&OT0, &OT1, &OT2, &OT3, &OT4, &OT5, g_nGroupIndex);

Step 3: Acquire information about possible errors generated
nErrCode = MCC_GetErrorCode(g_nGroupIndex);

The returned value of MCC_GetErrorCode() can be used to determine if the system is currently unable to move because its location will exceed software limitations. If the returned value is between 0xF301 to 0xF306, then precisely this

situation has occurred in order from the X axis to the W axis. The following example can be used to return the system to normal:

Step 4: Delete the record of the error from the system to return the system to a normal state

MCC_ClearError(g_nGroupIndex);

The MCCL also provides hardware limit switch check. For the limit switch to operate normally, in addition to accurately setting up the switch wiring, the command MCC_EnableLimitSwitchCheck() is required to allow the *wOverTravelUpSensorMode* and *wOverTravelDownSensorMode* settings to take effect. However, if *wOverTravelUpSensorMode* and *wOverTravelDownSensorMode* are set to 2, calling MCC_EnableLimitSwitchCheck() is completely ineffective.

If MCC_EnableLimitSwitchCheck(1) is used, the group motion will only be stopped when a limit switch for the direction of the given axis is touched (an axis moving in the positive direction touches a positive limit switch, or an axis moving in the reverse direction touches a reverse limit switch). If MCC_EnableLimitSwitchCheck(0) is used, the group motion is stopped whenever a limit switch is touched (regardless of direction).

The returned value for MCC_GetErrorCode() can determine whether motion is currently impossible because a limit switch has been activated (internally producing a record of the error). If the returned value is between 0xF701 to 0xF706, then precisely this situation has occurred, in order from the X axis to the W axis. The following example can be used to return the system to normal.

a. If the prior call was: MCC_EnableLimitSwitchCheck(2)

then: MCC_ClearError() → MCC_DisableLimitSwitchCheck() → reverse from the Limit Switch

b. If the prior call was: MCC_EnableLimitSwitchCheck(3)

then: MCC_ClearError()→reverse from the Limit Switch

19. Setting Path Blending

Related Commands

MCC_EnableBlend()

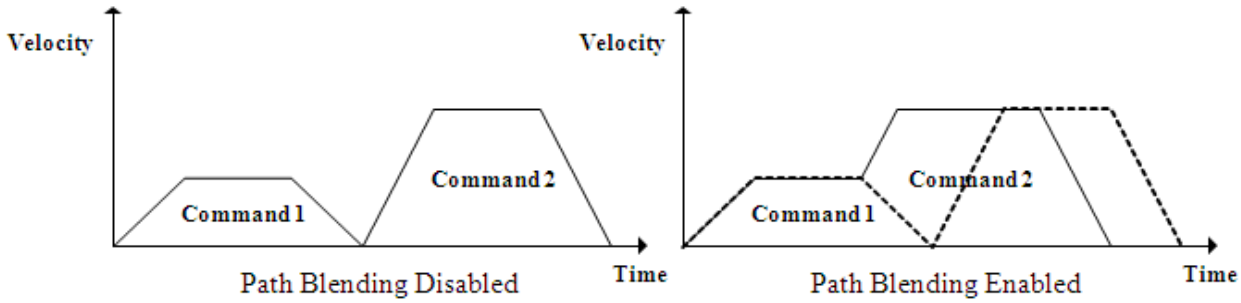
MCC_DisableBlend()

MCC_CheckBlend()

Example Program

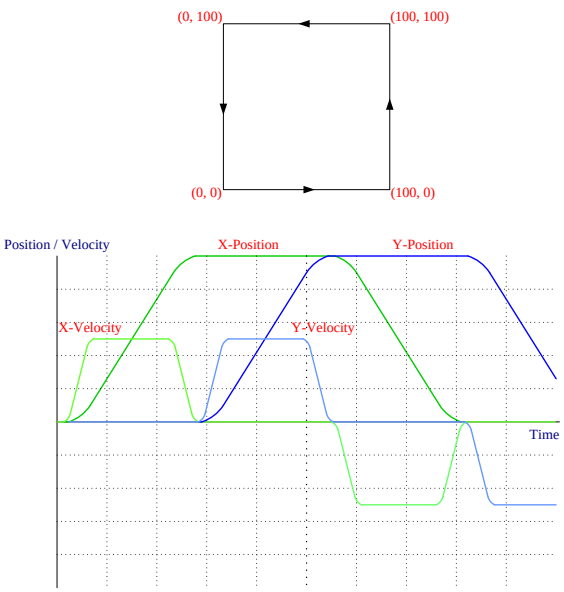
SetBlend.cpp

Description

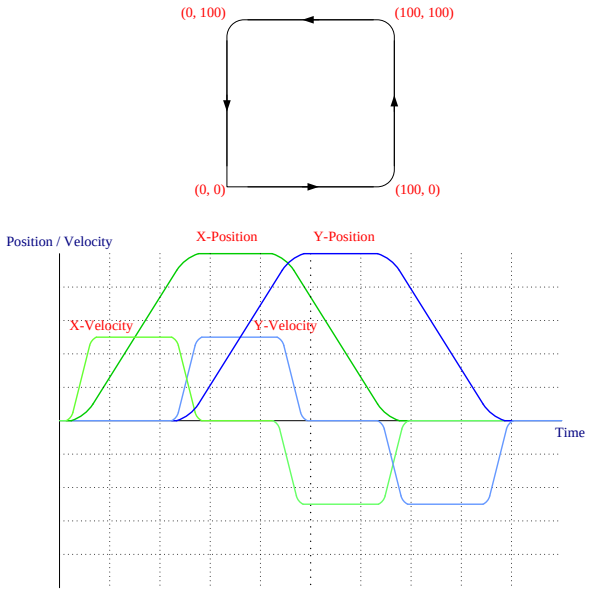


This figure shows the motion pattern after path blending has been enabled. Rather than decelerating after Command 1 reaches a constant speed, it accelerates directly to the constant speed for Command 2 (represented by the solid line in the figure on the right). In this way, command execution time is faster, but the connection between each command will distort the trajectory.

MCC_EnableBlend() and MCC_DisableBlend() will enable and disable path blending, respectively. MCC_CheckBlend() can then obtain the current status settings. If the returned value is 0, path blending is enabled; if the return value is 1, path blending is disabled.



Path Blending Disabled



Path Blending Enabled

The above figures display the trajectory change and motion conditions for a square track with path blending enabled or disabled. The figure on the left shows that when path blending is disabled, the Y axis line motion command can only initiate after the X axis motion command has decelerated and stopped. The figure on the right shows that when path blending is enabled, the Y axis line motion command will initiate acceleration simultaneously as the X axis command begins deceleration. Therefore, enabling path blending can effectively reduce the execution time for motion commands, but will also create distortion at the trajectory connection points between each motion command.

20. Acquiring and Deleting Error Status

Related Commands

MCC_GetErrorCode()

MCC_ClearError()

Example Program

ErrorStatus.cpp

Description

If an error status is removed after a system error has occurred, MCC_ClearError() is still required to delete the record of the error in the system. Otherwise, the system will be unable to continue executing subsequent motions. Generally, the user should acquire the current error code at any time during system operations to check for errors that may have occurred during operation. An example of this is outlined below. Also, please refer to the sections concerning the command use for **"Software Over Travel Check and Hardware Limit Switch Check."**

This section differs from the example program. The user can consult the following section to deal with error generation.

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

21. Gear Backlash and Gap Compensation

Related Commands

MCC_SetCompParam()

MCC_UpdateCompParam()

Example Program

Compensate.cpp

Description

The gear backlash and gap compensation function provided by the MCCL can compensate for errors created by manufacturing deficiencies in gears or screws and other mechanical linkages. For backlash errors or back gap errors, please refer to the **"EPCIO Series Motion Control Command Library User Manual."**

22. How to Complete Continuous Motion Among the Six Axes

Related Commands

MCC_CreateGroup()

MCC_SetFeedSpeed()

MCC_EnableBlend()

MCC_Line()

Example Program

SyncLine.cpp

Description

When MCC_EnableBlend() is used to enable path blending in one group (fulfilling the conditions for path and speed continuity), if MCC_Line() is called multiple times, the requirements for 6-axis synchronization (the 6 axes initiate and pause simultaneously) are met. However, only 3 axes, X, Y, and Z, can achieve the conditions for path and speed continuity; while the last 3 axes, U, V, and W, can only meet the requirements for synchronization.

Two groups should be used if both 6-axis synchronization and the conditions for path and speed continuity are required. The 1st group is responsible for the trajectory of the first 3 axes, while the 2nd group is responsible for the last 3. However, to meet the demands of 6-axis synchronization, the speed of the 2nd group can be converted by multiplying the ratio between the required movement distances in both groups by the feed rate of the 1st group. The programming code for this process is shown below. Here, fnSyncLine() must replace MCC_Line().

Step 1: Declare fnSyncLine

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

Step 2: Set and use two groups

```
int g_nGroupIndex0 = -1;
int g_nGroupIndex1 = -1;
// set group parameters
MCC_CloseAllGroups();
g_nGroupIndex0 = MCC_CreateGroup(0, 1, 2, -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, CARD_INDEX);
if( g_nGroupIndex1 < 0 )
{
   printf("Groups create error !\n\n");
   return;
}
```

Step 3: Enable path blending

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

Step 4: Call fnSyncLine

```
fn SyncLine(10, 20, 30, 40, 50, 60, 10);
fnSyncLine( 40, 50, 60, 10, 20, 30, 10);
```

Step 5: Define fnSyncLine

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

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

   if (dfDistance0 && dfXYZSpeed)
   {
      dfDistance1 = u * u + v * v + w * w;
      // The proper speed for the converted last three axes
      dfUVWSpeed = dfXYZSpeed * sqrt(dfDistance1/ dfDistance0);
```

```
    MCC_SetFeedSpeed(dfXYZSpeed, g_nGroupIndex0);


    // Known from the definition of group, the 1st group (g_nGroupIndex0) will
       appear like this
    // Command from the output of the first 3 axes
    MCC_Line(x, y, z, 0, 0, 0, g_nGroupIndex0);

    MCC_SetFeedSpeed(dfUVWSpeed, g_nGroupIndex1);
    // Known from the definition of group, the 2nd group (g_nGroupIndex1) will
       appear like this
    // Command from the output of the last 3 axes
    MCC_Line(u, v, w, 0, 0, 0, g_nGroupIndex1);
    }
}
```

23. Commands for Triggering Service Interrupt with Encoder Counts

Related Commands

MCC_SetENCRoutineEx()

MCC_SetENCCompValue()

MCC_EnableENCCompTrigger()

MCC_DisableENCCompTrigger()

MCC_SetENCInputRate()

MCC_GetENCValue()

Example Program

ENCCompare.cpp

Description

The command allowing the encoder count to trigger an ISR **(Note 1)** provided by the MCCL can set the comparative value of the encoder count. If the function is enabled, once the encoder count reaches this comparative value (MCC_GetENCValue() can be used to acquire the encoder count value), the MCCL will automatically call the ISR serially connected by the user. An example of this is outlined below:

Note 1: ISR indicates Interrupt Service Rutine.

Step 1: Declare the ISR

void _stdcall ENC_ISR_Function(ENCINT_EX *pstINTSource);

Step 2: Serially connect the ISR

MCC_SetENCRoutineEx(ENC_ISR_Function, CARD_INDEX);

Step 3: Set the comparative value to 20000 pulses

MCC_SetENCCompValue(20000, CHANNEL_INDEX, CARD_INDEX);

Step 4: Enable the command triggering the ISR by encoder count

MCC_EnableENCCompTrigger(CHANNEL_INDEX, CARD_INDEX);

MCC_Line(100, 0, 0, 0, 0, 0, g_nGroupIndex);

Step 5: Define the ISR

```
void _stdcall ENC_ISR_Function(ENCINT_EX *pstINTSource)
{
   if (pstINTSource->COMP0)// Determine whether the source of the trigger was the
   comparative conditions in Channel 0
   // Abort motion commands currently being executed and those in the buffer
   MCC_AbortMotionEx(0, g_nGroupIndex);
   ENC_ISR++;
   MCC_DisableENCCompTrigger(CHANNEL_INDEX);//Disable the command
   triggering the ISR by encoder count
}
```

The above example shows that after this command has been enabled, once the encoder count equals 20000 pulses for line motion, incomplete motions will be stopped. When the first parameter for MCC_AbortMotionEx is set to 0, the deceleration time is 0, allowing the encoder location to approach 20000 once it has stopped.

24. Latch Encoder Count and Service Interrupt Triggered by INDEX signals

Related Commands

MCC_SetENCRoutineEx()

MCC_GetENCValue()

MCC_SetENCLatchType(()

MCC_SetENCLatchSource()

MCC_EnableENCIndexTrigger()

Example Program

GetENCLatch.cpp

Description

The latch encoder count function provided by the MCCL uses MCC_SetENCLatchSource() to appoint the trigger conditions (sources). After the trigger conditions or latch mode are satisfied (using the trigger mode set by MCC_SetENCLatchType()), the encoder count can be recorded in the latch register, and MCC_GetENCLatchValue() can be used to acquire the latch register recorded value. An example of this is outlined below:

Step 1: Set the encoder count latch mode

ENC_TRIG_FIRST

The first time the trigger conditions are met, the latch count is no longer altered

ENC_TRIG_LAST

When the trigger conditions are met, the count is latched; and when conditions are repeatedly met, a new count is repeatedly latched

MCC_SetENCLatchType(ENC_TRIG_LAST, CHANNEL_INDEX, ARD_INDEX);

Step 2: Set the encoder trigger source. A total of 15 trigger sources (conditions) can act as the latch count conditions. The setting can simultaneously unite multiple conditions, at which point the encoder INDEX signal selected is the trigger source (condition)

MCC_SetENCLatchSource(ENC_TRIG_INDEX0, CHANNEL_INDEX, CARD_INDEX);

The above example shows that the encoder INDEX signal can be used as the trigger source (condition). When the command allowing the encoder INDEX signal to trigger an ISR is enabled, MCC_GetENCLatchValue() can be used immediately after the encoder INDEX signal occurs to acquire the recorded value in the latch register. To use this function, the user must first serially connect, customize, and enable the ISR.

Step 3: Declare the ISR

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

Step 4: Serially connect the ISR

```
MCC_SetENCRoutineEx(ENC_ISR_Function, CARD_INDEX);
```

Step 5: Enable the command 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)// Determine if the source of the trigger was the
   INDEX signal
   {
      // Acquire the value recorded in the latch temporary storage
      MCC_GetENCLatchValue(&lLatchValue, CHANNEL_INDEX,
      CARD_INDEX);
   }
}
```

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

25. Commands for Triggering Service Interrupt with Local I/O Signal Control

Related Commands

MCC_SetServoOn()

MCC_SetServoOff()

MCC_EnablePosReady()

MCC_DisablePosReady()

MCC_EnablePosReady()

MCC_GetLimitSwitchStatus()

MCC_GetHomeSensorStatus()

MCC_SetLIORoutineEx()

MCC_SetLIOTriggerType()

MCC_EnableLIOTrigger()

Example Program

LIOTrigger.cpp

Description

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

Some limit switch input connection signals can trigger the customized ISR, including:

a. EPCIO-601/605/6000/6005: 7 points

Channel 0 Limit Switch +

Channel 1 Limit Switch +

Channel 2 Limit Switch +

Channel 3 Limit Switch +

Channel 4 Limit Switch +

Channel 5 Limit Switch +

Channel 1 Limit Switch -

b. EPCIO-400/405/4000/4005: 7 points

Channel 0 Limit Switch +

Channel 1 Limit Switch +

Channel 2 Limit Switch +

Channel 3 Limit Switch +

Channel 0 Limit Switch -

Channel 1 Limit Switch -

Channel 2 Limit Switch -

The procedure for using “ISR triggered by input connection signal” is outlined below:

Step 1: Use MCC_SetRIORoutineEx() to serially connect the customized ISR.
The customized ISR must first be designed. The routine declaration must abide by the following definitions:

typedef void(_stdcall *LIOISR_EX)(LIOINT_EX*)

For example, the customized ISR could be designed as follows:

```
_stdcall MyLIOFunction(LIOINT_EX *pstINTSource)
{
   // Determine whether this ISR was triggered by touching channel 0 limit switch +
   if (pstINTSource->LDI0)
   {
      // process when  channel 0 limit switch + is touched
   }

   // Determine whether this ISR was triggered by touching channel 1 limit switch +
   if (pstINTSource->LDI1)
   {
     // process when channel 0 limit switch + is touched
   }
}
```

Language similar to “else if (pstINTSource->LDI1)” cannot be used because pstINTSource->LDI0 and pstINTSource->LDI1 might not equal 0 simultaneously.

Next, use MCC_SetLIORoutineEx(MyLIOFunction) to serially connect the customized ISR. When the customized ISR is triggered and executed, the incoming customized ISR is declared to be the pstINTSource parameters of LIOINT_EX to determine the touched input connector that called this customized ISR. LIOINT_EX is defined as the following:

```
typedef struct _LIO_INT_EX
{
    BYTE LDI0;
    BYTE LDI1;
    BYTE LDI2;
    BYTE LDI3;
    BYTE LDI4;
    BYTE LDI5;
    BYTE LDI6;
    BYTE TIMER;
} LIOINT_EX;
```

The definitions for the connectors corresponding to each field in LIOINT_EX are as follows:

	EPCIO-601/605/6000/6005	EPCIO-400/405/4000/4005
LDI0	Channel 0 Limit Switch+	Channel 0 Limit Switch+
LDI1	Channel 1 Limit Switch+	Channel 1 Limit Switch+
LDI2	Channel 2 Limit Switch+	Channel 2 Limit Switch+
LDI3	Channel 3 Limit Switch+	Channel 3 Limit Switch+
LDI4	Channel 4 Limit Switch+	Channel 0 Limit Switch-
LDI5	Channel 5 Limit Switch+	Channel 1 Limit Switch-
LDI6	Channel 0 Limit Switch-	Channel 2 Limit Switch-

If the values for these fields are non-zero, the corresponding connectors of the field currently have a signal input. For example, if the input parameter pstINTSource->LDI2 in MyLIOFunction() is not zero, the channel 2 limit switch + has been touched.

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

The trigger type can be set as rising edge trigger, falling edge trigger, or level change trigger. The MCC_SetLIOTriggerType() input parameters could be:

LIO_INT_RISE	rising edge trigger (Default)
LIO_INT_FALL	falling edge trigger
LIO_INT_LEVEL	level change trigger

Step 3: Finally, use MCC_EnableLIOTrigger() to enable the “input connector signal to trigger the ISR.”
MCC_DisableLIOTrigger() can be used to disable this function.

26. Service Interrupt Triggered by Timer Ending

Related Commands

MCC_SetLIORoutineEx();

MCC_SetTimer()

MCC_EnableTimer()

MCC_EnableTimerTrigger()

Example Program

TimerTrigger.cpp

Description

Using the MCCL, the timing for the 24 bit timer in the EPCIO Series motion control card can be set. When the timer function is enabled and the timer ends (when the timer value is equal to the set value), the customized ISR will be triggered and the timer will be reset. This process will continue until the function is disabled. The procedure to use this function is outlined below:

Step 1: Use MCC_SetLIORoutineEx() to serially connect the customized ISR

If MCC_SetLIORoutineEx() has not been called, please refer to the above explanation for this procedure (please refer to the **Section 25 "Command for Triggering Service Interruption with Local I/O Signal Control"**). If MCC_SetLIORoutineEx() has been called, simply add the determination of the incoming parameter (pstINTSource) "timer ending" field in the customized ISR. Please refer to the procedure below:

```
_ stdcall MyLIOFunction(LIOINT_EX *pstINTSource)
{
    // Determine if the ISR was triggered by the ending of the timer
    if (pstINTSource->TIMER)
    {
     // Process when the timer ends
     }
```

Step 2: Use MCC_SetTimer() to set the timer, using the unit System Clock(25ns)

Step 3: Use MCC_EnableTimerTrigger() to enable the “ISR by Timer Ending”

Step 4: Use MCC_EnableTimer() to enable the timer

27. Watchdog Function

Related Commands

MCC_SetLIORoutineEx()

MCC_SetTimer()

MCC_SetWatchDogTimer()

MCC_SetWatchDogResetPeriod()

MCC_EnableTimer()

MCC_EnableWatchDogTimer()

Example Program

WatchDog.cpp

Description

After the watchdog function is enabled, MCC_RefreshWatchDogTimer() must be used to refresh the watchdog timer before it ends (in other words, before the watchdog timer value equals the set comparative value). Otherwise, once the watchdog timer equals the set comparative value, the hardware will be reset. The procedure for using the watchdog is outlined below:

Step 1: Use MCC_SetTimer() to set the timer in units of System Clock(25ns).

Step 2: Use MCC_SetWatchDogTimer() to set the comparative value for the watchdog timer

The watchdog timer comparative value is 16-bit, using the time on the timer as the time base. If the following programming code is used:

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

The comparative value for the watchdog timer for Card 0 is (25 ns × 1000000) × 2000 = 50 s.

Step 3: Use MCC_SetWatchDogResetPeriod() to set the reset signal period

This command can program the reset hardware period generated by the watchdog function, using units of system clock (25 ns).

Step 4: Use MCC_EnableTimer() to enable the timing function of the timer

Step 5: Use MCC_RefreshWatchDogTimer() to refresh the watchdog timer content before the timer ends

Users can combine “ISR by Timer Ending” functions as a warning prior to watchdog hardware reset action, to conduct necessary processing in the timer ISR.

28. Set and Acquire Remote I/O Output and Input Connection Signal

Related Commands

MCC_EnableRIOSetControl()

MCC_EnableRIOSlaveControl()

MCC_GetRIOInputValue()

MCC_SetRIOOutputValue()

Example Program

RIOCtrl.cpp

Description

Each EPCIO-6000 possesses two Remote I/O card connectors (referred to as Remote I/O Set 0 and Remote I/O Set 1, collectively known as Remote I/O Master terminal). The two Remote I/O cards (also referred to as Remote I/O Slave terminal) can be controlled simultaneously. Each Remote I/O card provides 64 ouput and 64 input connections.

EnableRIOSetControl() and EnableRIOSlaveControl() enable data transmission. This example is outlined below: enable the Remote I/O Set 0 on the motion control card, and enable data transmission function of slave.

```
EnableRIOSetControl(RIO_SET0, CARD_INDEX);
EnableRIOSlaveControl(RIO_SET0, CARD_INDEX);
```

When initial settings are complete, low potential (ECOM-) can be used to contact connectors, and MCC_GetRIOInputValue() can acquire the input connector signal status; MCC_SetRIOOutputValue() can also be used to set the output connector signal status.

29. Acquire Remote I/O Signal Transmission Status

Related Commands

MCC_EnableRIOSetControl()

MCC_EnableRIOSlaveControl()

MCC_GetRIOTransStatus()

MCC_GetRIOMasterStatus()

MCC_GetRIOSlaveStatus()

Example Program

RIOStatus.cpp

Description

MCC_GetRIOTransStatus() can be used to monitor the data transmission status for each Remote I/O Set at any time. When a data transmission error occurs, the data transmission error information obtained by MCC_GetRIOMasterStatus() and MCC_GetRIOSlaveStatus() comes from the motion control card or the Remote I/O card.

If the status is acquired using MCC_GetRIOTransStatus(), MCC_GetRIOMasterStatus(), and MCC_GetRIOSlaveStatus() equals 1, the transmission status is normal; if it equals 0, a data transmission error occurred. This example is outlined below.

```
WORD wTransStatus;

// Acquire transmission status
MCC_GetRIOTransStatus( &wTransStatus, RIO_SET0, CARD_INDEX);
```

If wTransStatus equals 1, the transmission status is normal; if it equals 0, a data transmission error occurred.

30. Commands for Triggering Service Interrupt by Remote I/O Input Connection Signal

Related Commands

MCC_EnableRIOSetControl()

MCC_EnableRIOSlaveControl()

MCC_SetRIORoutineEx()

MCC_SetRIOTriggerType()

MCC_EnableRIOInputTrigger()

Example Program

RIOInput.cpp

Description

The signals for the first four input connections (RIO_DI0, RIO_DI1, RIO_DI2, and RIO_DI3*)* in each Remote I/O card can trigger the customized ISR. The procedure for using the "Command for Triggering Service Interruption by Input Connection Signal" is as follows:

Step 1: Use MCC_SetRIORoutineEx() to serially connect the customized ISR.
The customized ISR must first be designed. The ISR declaration must abide by the following definitions:

typedef void(_stdcall *RIOISR_EX)(RIOINT_EX*)

For example, the customized ISR could be designed as follows:

```
_stdcall MyRIOFunction(RIOINT_EX *pstINTSource)
{
   // Determine whether the trigger came from Set 0 Digital Input 0
   if (pstINTSource->SET0_DI0)
   {
      // Process when Digital Input 0 signal changes
   }
}
```

Next, use MCC_SetRIORoutineEx() to serially connect the customized ISR. The routine prototype is as follows; where pfnRIORoutine is the routine specification for the customized ISR, for example MyRIOFunction.

int MCC_SetRIORoutineEx (RIOISR_EX pfnRIORoutine, WORD wCardIndex)

.

Step 2: Set the method for the command triggering ISR by Remote I/O Digital Input signals

Use MCC_SetRIOTriggerType() to set the method for triggering ISR by Remote I/O Digital Input signals to “Front Edge Trigger,” “Back Edge Trigger,” or “Level Change Trigger.”

Step 3: Use MCC_EnableRIOInputTrigger() to enable the command triggering ISR by input connection signals

Below is an example using MCC_EnableRIOInputTrigger(), where the Remote I/O Set 0 input connection signal triggers the ISR.

MCC_EnableRIOInputTrigger(RIO_SET0, CARD_INDEX);

31. Commands for Triggering Service Interrupt by Remote I/O Data Transmission Errors

Related Commands

MCC_EnableRIOSetControl()

MCC_EnableRIOSlaveControl()

MCC_SetRIORoutineEx()

MCC_EnableRIOTransTrigger()

Example Program

RIOError.cpp

Description

Besides using MCC_GetRIOTransStatus(), MCC_GetRIOMasterStatus(), and MCC_GetRIOSlaveStatus() to monitor the remote I/O data transmission status at any time, a data transmission error can also trigger the customized ISR. This function provides the user with immediate data transmission error processing. The procedures used in this function are outlined below:

Step 1: Use MCC_SetRIORoutineEx() to serially connect the customized ISR.
The customized ISR must first be designed. Routine declaration must abide by the following definitions:

```
typedef void(_stdcall *RIOISR_EX)(RIOINT_EX*)
```

For example, the customized ISR can be designed in the following way:

```
_stdcall MyRIOFunction(RIOINT_EX *pstINTSource)
{
    // Determine whether data error occurred in Set 0
    if (pstINTSource->SET0_FAIL)
    {
     // Process when data transmission error occurs
     }
}
```

Next, use MCC_SetRIORoutineEx() to serially connect the customized ISR. The original form of this command is as follows; where pfnRIORoutine is the command specification for the customized ISR, for example MyRIOFunction.

int MCC_SetRIORoutineEx(RIOISR_EX pfnRIORoutine, WORD wCardIndex)

where pfnRIORoutine is the routine specification for the customized ISR, for example MyRIOFunction.

Step 2: Use MCC_EnableRIOTransTrigger() to enable the command triggering ISR by data transmission error

Below is an example using MCC_EnableRIOInputTrigger(), where a Remote I/O Set 0 data transmission error triggers the ISR.

MCC_EnableRIOTransTrigger(RIO_SET0, CARD_INDEX);

32. Plan DAC Analog Voltage Output

Related Commands

MCC_StartDACConv()

MCC_SetDACOutput()

Example Program

DACOutput.cpp

Description

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

Use MCC_StartDACConv() to initiate DAC conversion. After MCC_InitSystem(...) is successfully called, the MCCL will automatically call this command. Finally, use MCC_SetDACOutput() to output the voltage value.

33. ADC Voltage Input: Single Conversion

Related Commands

MCC_SetADCConvMode()

MCC_SetADCConvType()

MCC_SetADCSingleChannel()

MCC_StartADCConv()

Example Program

ADC1Time.cpp

Description

This example program uses ADC Channel 0 to conduct single positive and negative voltage conversion (EPCIO-400/601: -10 to 10 V, EPCIO-4000/6000: -5 to 5 V) and to acquire the input voltage value. The procedures used in this function are outlined below:

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

Step 2: Set voltage conversion type to bipolar mode (EPCIO-400/601: -10V to 10V, EPCIO-4000/6000: -5V to 5V)
MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

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

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

If acquiring updated voltage values is required when using single voltage conversion, calling MCC_StartADCConv(CARD_INDEX) again is required. MCC_GetADCWorkStatus() could also be used to determine the completion status of the single voltage conversion.

34. ADC Voltage Input: Continual Conversion

Related Commands

MCC_SetADCConvMode()

MCC_SetADCConvType()

MCC_EnableADCConvChannel()

MCC_StartADCConv()

Example Program

ADCInput.cpp

Description

This example program uses ADC Channel 0 continuous positive and negative voltage conversion (ISA Bus: -10 V to 10 V, PCI Bus: -5 V to 5 V) and acquires the input voltage value. The procedures used in this function are outlined below:

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

Step 2: Set voltage conversion type to bipolar mode (ISA Bus: -10V -10V, PCI Bus: -5V -5V)
MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

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

Step 4: Enable voltage conversion
MCC_StartADCConv(CARD_INDEX)

35. ADC Comparator Interrupt Control

Related Commands

MCC_SetADCRoutine()

MCC_SetADCConvMode()

MCC_SetADCConvType()

MCC_SetADCCompValue()

MCC_SetADCCompType()

MCC_EnableADCCompTrigger()

MCC_EnableADCConvChannel()

MCC_StartADCConv()

Example Program

ADCComp.cpp

Description

This example program sets the comparative value for ADC's Channel 0 comparator. When the comparison conditions are established and set from high to low voltage, the customized ISR will be trigged. This example will continuously convert ADC, thereby continuously triggering interrupt when comparison conditions are established. The procedures used in this function are outlined below:

Step 1: Serially connect the customized ISR

```
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)// Determine if comparison conditions are met
nISRCount++;
}
```

Step 2: Set conversion mode to continuous conversion

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

Step 3: Set voltage conversion type to bipolar mode (EPCIO-400/601: -10V -10V, EPCIO-4000/6000: -5V -5V)
MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 4: Set voltage comparator comparative value
MCC_SetADCCompValue(2.0, 0, CARD_INDEX);

Step 5: Set voltage comparison conditions at from high voltage 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: Enable voltage conversion
MCC_StartADCConv(CARD_INDEX)

36. Commands for Triggering ISR by ADC Tag Channel

Related Commands

MCC_SetADCRoutine()

MCC_SetADCConvMode()

MCC_SetADCConvType()

MCC_SetADCTagChannel()

MCC_EnableADCTagTrigger()

MCC_EnableADCConvChannel()

MCC_StartADCConv()

Example Program

ADCTag.cpp

Description

This example program sets ADC Channel 0 as the tag channel. Voltage conversion in the tag channel will trigger the customized ISR. This example will continuously convert ADC, thereby continuously triggering the ISR. The procedures used in this function are outlined below:

Step 1: Serially connect the customized ISR

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

The customized ISR can be defined as follows:

```
void _stdcall ADC_ISR_Function(ADCINT *pstINTSource)// ADC ISR
{
// Determine the tag channel voltage conversion status; if converted, add 1 to the number of interrupt
if (pstINTSource->TAG)
nISRCount++;
}
```

Step 2: Set conversion mode to continuous conversion

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

Step 3: Set voltage conversion type to bipolar mode (EPCIO-400/601: -10V -10V, EPCIO-4000/6000: -5V -5V)

MCC_SetADCConvType(ADC_TYPE_BIP, 0, CARD_INDEX);

Step 4: Set tag channel

MCC_SetADCTagChannel(TAG_CHANNEL_INDEX);

Step 5: Enable tag channel to trigger the customized ISR

MCC_EnableADCTagTrigger(CARD_INDEX);

Step 6: Enable Channel 0 voltage conversion

MCC_EnableADCConvChannel(0, CARD_INDEX);

Step 7: Enable voltage conversion

MCC_StartADCConv(CARD_INDEX)