

---

# **COMIZOA DAQ System**

## **LX-Series Manual**

### **–Motion Controller part–**

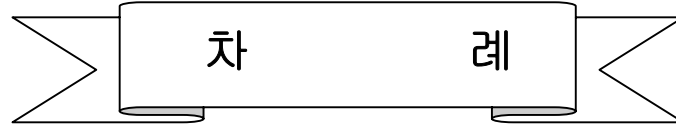
### **( Additional Informations )**



---

*COMputer Innovation  
is Zoomed by Our Affection!*





<b>PART I . 매뉴얼 수정 안내 .....</b>	<b>1</b>
■ COMILX_MC_ArcTo_p.....	2
■ COMILX_MC_GetMotionStatus.....	3
■ COMILX_MC_GetMioStatus.....	4
■ COMILX_MC_MaskInterrupt .....	6
■ COMILX_MC_GetAxisIntState .....	8
■ COMILX_MC_GetIntStatus .....	9
■ 직선보간시 초기 속도 설정에 관하여 .....	15
<b>PART II . 라이브러리 업데이트 안내 .....</b>	<b>17</b>
1. POSITION LATCH 관련 함수 .....	20
■ COMILX_MC_SetMioCfgLTC .....	21
■ COMILX_MC_GetMioCfgLTC .....	22
■ COMILX_MC_GetLatchState .....	23
■ COMILX_MC_ReadLatchCouner.....	24
2. 위치비교 출력(CMP) 기능 관련 함수 .....	26
■ COMILX_MC_SetMioCfgCMP .....	28
■ COMILX_MC_GetMioCfgCMP .....	29
■ COMILX_MC_SetTriggerCompare .....	30
■ COMILX_MC_RegTableCCMP.....	32
■ COMILX_MC_BuildTableCCMP .....	33
■ COMILX_MC_StartCCMP .....	35
■ COMILX_MC_StopCCMP .....	38
3. 스플라인(SPLINE) 보간 기능 관련 함수.....	39

■ COMILX_MC_BuildSpline.....	40
■ COMILX_MC_DeleteSpline .....	43
4. 헬리컬(HELICAL) 보간 기능 관련 함수 .....	44
■ COMILX_MC_StartHelical.....	45
■ COMILX_MC_AbortHelical .....	49
■ COMILX_MC_SetHelOnceSpeed .....	50
■ COMILX_MC_StartHelOnce .....	53
■ COMILX_MC_HelOnce.....	58
5. 라이브러리 V2.9 에서 추가된 기타 함수 .....	59
■ COMILX_MC_ServoOn.....	59
■ COMILX_MC_GetServoOn .....	60
■ COMILX_MC_SetELL .....	61
■ COMILX_MC_SetMioCfgSTA .....	62
■ COMILX_MC_SetMioCfgSTP .....	66
■ COMILX_MC_CompleteArc .....	69
■ COMILX_MC_SetListMotionAxes.....	71
6. 라이브러리 V3.1 에서 추가된 기타 함수 .....	74
■ COMILX_MC_InitFromFile .....	74
■ COMILX_MC_HomeMoveAuto.....	78
■ COMILX_MC_LmCurSequence .....	84
■ COMILX_MC_SetSpeedMx2 .....	87
■ COMILX_EnableDebugLog.....	91
■ COMILX_MC_SetOutputMask.....	93





## PART I . 매뉴얼 수정 안내

제품과 함께 제공되는 매뉴얼에서 몇 가지 내용이 부족하거나 잘못된 내용이 있습니다. 이 점에 대하여 사용자의 깊은 이해를 부탁드립니다.

본 단원에서는 이와 같이 내용이 부족하거나 잘못된 내용을 수정하오니 본 단원에서 수정된 내용을 반드시 참조하여 사용하시는데 혼선이 없기를 바랍니다.

수정된 내용은 [표 1-1]과 같습니다.

수정 내용	C/P	O/P
COMILX_MC_ArcTo_p 함수 설명 : nDir 파라미터 설명이 추가되었습니다.	2p	259p
COMILX_MC_GetMotionStatus 함수 설명 : 반환값의 BIT1, BIT2 설명 수정	3p	338p
COMILX_MC_GetMioStatus 함수 설명 : 반환값의 BIT16, BIT17 설명 추가	4p	339p
COMILX_MC_MaskInterrupt 함수 설명 : dwMask 파라미터의 BIT12 설명 수정	6p	367p
COMILX_MC_GetAxisIntState 함수 설명 : nChannel 파라미터 및 반환값 설명 수정	8p	371p
COMILX_MC_GetIntStatus 함수 설명 : pEventStatus 파라미터의 BIT12, BIT14 설명 수정	9p	373p

참고 : 표의 “C/P” 항목은 현재의 문서의 페이지 번호를 의미하며, “O/P” 항목은 인쇄된 매뉴얼에서 해당 내용이 수록된 페이지 번호를 의미합니다

[표 1-1] 매뉴얼 수정 사항 리스트

## ■ COMILX\_MC\_ArcTo\_p

### 함수 원형

```
void COMILX_MC_ArcTo_p(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fXEndPos, double fYEndPos, int nDir)
```

### 함수 설명

이 함수는 원호보간 이동을 수행합니다. 이 함수는 중심점의 좌표값을 절대좌표값으로 설정하며 원호보간 이동의 완료지점(End Point)에 대한 정보 또한 절대좌표값으로 설정합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nMapIndex* : 축 그룹 인덱스. 이 값은 0 또는 1 이어야 하며 COMILX\_MC\_MapAxes 함수를 통하여 제어 대상축들이 맵핑되어 있어야 합니다.
- ▶ *fXCent* : 중심점의 X 축 절대좌표값
- ▶ *fYCent* : 중심점의 Y 축 절대좌표값
- ▶ *fXEndPos* : 원호보간 이동을 완료할 목표지점(End point)의 X 축 절대좌표값
- ▶ *fYEndPos* : 원호보간 이동을 완료할 목표지점(End point)의 Y 축 절대좌표값
- ▶ *nDir* : 회전 방향을 지정합니다.

Value	Meaning
0 또는 음수	시계 방향(CW)으로 회전
양수	반시계 방향(CCW)으로 회전



---

## ■ COMILX\_MC\_GetMotionStatus

### 함수 원형

```
int COMILX_MC_GetMotionStatus (HANDLE hDevice, int nChannel)
```

### 함수 설명

이 함수는 현재 모션의 동작 상태를 반환합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들
- ▶ *nChannel* : 채널(축) 번호, 0 ~ 3

### 반환값

현재 모션의 동작 상태를 반환합니다. 반환 값의 의미는 다음과 같습니다.

Value	Meaning
0	Stop
1	Wait for DR input
2	Wait for STA input
3	Wait for an internal synchronous signal
4	Wait other axis
5	Wait ERC finished
6	Wait DIR change (DIR 은 외부입력 신호가 아닌 내부적인 신호임)
7	Reserved
8	Wait PA/PB
9	In home special speed motion
10	In start velocity motion
11	In acceleration
12	In working velocity
13	In deceleration
14	Wait INP
15	Reserved

## ■ COMILX\_MC\_GetMioStatus

### 함수 원형

```
int COMILX_MC_GetMioStatus(HANDLE hDevice, int nChannel)
```

### 함수 설명

이 함수는 현재 모션과 관련된 여러가지 I/O 상태를 반환합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들
- ▶ *nChannel* : 채널(축) 번호, 0 ~ 3

### 반환값

모션과 관련된 여러가지 I/O 상태를 32 비트 값으로 반환합니다. 반환되는 값의 각 비트의 값은 다음의 표와 같이 특정 I/O 핀의 상태를 나타냅니다.

Bit	Name	
BIT0	RDY	Servo ready signal input status(1=ON), 단 COMI-LX501에서는 지원되지 않음
BIT1	ALM	Alarm signal status(1=ON)
BIT2	+EL	Positive limit switch status(1=ON)
BIT3	-EL	Negative limit switch status(1=ON)
BIT4	ORG	Orgin switch status(1=ON)
BIT5	DIR	Operating direction status(1=ON)
BIT6	Reserved	
BIT7	PCS	PCS signal input status(1=ON)
BIT8	ERC	ERC pin output status(1=ON)
BIT9	EZ	Index signal status(1=ON)
BIT10	CLR	Clear input status(1=ON)
BIT11	Latch	Latch signal input status(1=ON)
BIT12	SD	Slow Down signal input status(1=ON)
BIT13	INP	In-Position signal input status(1=ON)

---

BIT14	DRP	+DR input signal status(1=ON)
BIT15	DRN	-DR input signal status(1=ON)
BIT16	STA	STA input signal status(1=ON)
BIT17	STP	STP input signal status(1=ON)
BIT16 ~BIT31	Reserved	

## ■ COMILX\_MC\_MaskInterrupt

### 함수 원형

```
void COMILX_MC_MaskInterrupt (HANDLE hDevice, int nChannel, long dwMask)
```

### 함수 설명

이 함수는 어떠한 조건의 인터럽트를 받아들일 것인지를 설정합니다. 인터럽트를 발생 시킬 조건을 dwMask 파라미터를 통하여 설정하십시오.

### 매개 변수

- ▶ **hDevice** : 디바이스 핸들.
- ▶ **nChannel** : 채널(축) 번호, 0 ~ 3. 채널별로 각각 다른 인터럽트 조건을 설정할 수 있습니다.
- ▶ **dwMask** : 인터럽트를 발생시킬 조건을 설정합니다. 이 값의 각 비트는 다음의 표와 같이 인터럽트 발생 조건을 설정합니다. 각 비트값이 1 이면 해당조건이 인터럽트를 발생시키며, 0 이면 발생시키지 않습니다.

Bit	인터럽트 발생 조건
BIT0	Normal stop
BIT1	Succesive start of the next operation cause event interrupt
BIT2	Reserved
BIT3	Reserved
BIT4	Start of acceleration
BIT5	End of acceleration
BIT6	Start of deceleration
BIT7	End of deceleration
BIT8	Reserved
BIT9	Reserved
BIT10	Position error tolerance exceed (COMILX_MC_SetErrorCompare 함수 참조)
BIT11	General Comparator (COMILX_MC_SetGeneralCompare 함수 참조)
BIT12	CMP trigger output

BIT13	CLR signal input resetting counter value
BIT14	LTC input making counter value latched
BIT15	ORG input making counter value latched
BIT16	SD input ON
BIT17	±DR input change
BIT18 ~ BIT31	Reserved

## 참 고

□ 다음과 같은 Error Interrupt 는 마스크되지 않습니다.

인터럽트 발생 조건
Stop by +SL(Software limit) stop motion
Stop by -SL(Software limit) stop motion
Reserved
Stop by General Comparator stop motion
Reserved
+EL signal is turning ON stop motion
-EL signal is turning ON stop motion
ALM signal is turning ON stop motion
Reserved
Reserved
SD signal turning ON after deceleration
Abnormal operation data stop motion
Reserved
Reserved
PA/PB input buffer counter overflows
In-position counter counts beyond the range at the time of interpolation
Reserved

## ■ COMILX\_MC\_GetAxisIntState

### 함수 원형

```
BOOL COMILX_MC_GetAxisIntState (HANDLE hDevice, int nChannel)
```

### 함수 설명

이 함수는 각 축에 대하여 인터럽트가 발생하였는지를 알려주는 함수입니다. 사용자는 인터럽트 이벤트가 발생하면 먼저 이 함수를 이용하여 어느 축에서 인터럽트가 발생하였는지를 체크한 후 COMILX\_MC\_GetIntStatus() 함수를 이용하여 인터럽트의 종류를 파악하여 적절한 대응을 합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호, 0 ~ 3.

### 반환값

각 축의 인터럽트 발생 상태

Value	Meaning
0	해당축에서 인터럽트가 발생되지 않음
1	해당축에서 인터럽트가 발생됨

### 참 고

- 사용에는 COMILX\_MC\_GetIntStatus() 함수 설명편을 참조하십시오.

## ■ COMILX\_MC\_GetIntStatus

### 함수 원형

```
void COMILX_MC_GetIntStatus (HANDLE hDevice, int nChannel, long *pErrorStatus,
                             long *pEventStatus)
```

### 함수 설명

이 함수는 각 축의 인터럽트가 발생한 원인을 알려주는 함수입니다. 사용자는 인터럽트 이벤트가 발생하면 먼저 COMILX\_MC\_GetAxisIntState() 함수를 이용하여 어느 축에서 인터럽트가 발생하였는지를 체크한 후 이 함수를 이용하여 인터럽트의 종류를 파악하여 적절한 대응을 합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호, 0 ~ 3
- ▶ *pErrorStatus* : 에러에 관련된 인터럽트의 상태를 나타내는 값을 받아들이는 변수의 주소값. 이 변수에 전달되는 값은 32 비트 정수값이며 각 비트의 값의 의미는 다음과 같습니다.

Bit	인터럽트 발생 조건
BIT0	Stop by +SL(Software limit) stop motion
BIT1	Stop by -SL(Software limit) stop motion
BIT2	Reserved
BIT3	Stop by General Comparator stop motion
BIT4	Reserved
BIT5	+EL signal is turning ON stop motion
BIT6	-EL signal is turning ON stop motion
BIT7	ALM signal is turning ON stop motion
BIT8	Reserved
BIT9	Reserved
BIT10	SD signal turning ON after deceleration
BIT11	Abnormal operation data stop motion
BIT12	Reserved

BIT13	Reserved
BIT14	PA/PB input buffer counter overflows
BIT15	In-position counter counts beyond the range at the time of interpolation
BIT16 ~ BIT31	Reserved

▶ **pEventStatus** : 에러 인터럽트 이외의 인터럽트(이벤트 인터럽트)의 상태를 나타내는 값을 받아들이는 변수의 주소값. 이벤트 인터럽트는 COMILX\_MC\_MaskInterrupt() 함수를 통하여 마스크(Mask) 가능합니다. 이 변수에 전달되는 값은 32 비트 정수 값이며 각 비트의 값의 의미는 다음과 같습니다.

Bit	인터럽트 발생 조건
BIT0	Normal Stop
BIT1	Succesive start of the next operation
BIT2	Reserved
BIT3	Reserved
BIT4	Start of acceleration
BIT5	End of acceleration
BIT6	Start of deceleration
BIT7	End of deceleration
BIT8	Reserved
BIT9	Reserved
BIT10	Position error tolerance exceed (COMILX_MC_SetErrorCompare 함수 참조)
BIT11	General Comparator (COMILX_MC_SetGeneralCompare 함수 참조)
BIT12	CMP output triggered
BIT13	Reserved
BIT14	Latched
BIT15	ORG input signal ON
BIT16	SD input signal ON



## 사용예

본 예제는 인터럽트 이벤트를 받을 수 있도록 하는 것에 대한 예제입니다. 본 예제에서는 x 축에 대하여 인터럽트를 Enable 하고 인터럽트가 발생하면 인터럽트 상태를 화면에 표시하는 예제입니다.

인터럽트 이벤트처리를 위하여 Application 쪽에서 해야하는 작업은 다음과 같습니다.

1) COMILX\_MC\_MaskInterrupt()함수를 이용하여 이벤트를 마스크하여 처리할 인터럽트 이벤트를 선택한다.

2) CreateEvent() API 함수를 이용하여 Event 핸들을 생성한다.

3) COMILX\_MC\_EnableInterrupt()함수를 이용하여 인터럽트를 Enable 시키고, 이벤트핸들을 COMI-LX501 드라이버에 전달한다.

4) WaitForSingleObject()등의 API 함수를 이용하여 이벤트를 기다린다.

5) WaitForSingleObject()함수가 Return 되면 인터럽트 이벤트가 발생한 것이므로 COMILX\_MC\_GetAxisIntState() 함수와 COMILX\_MC\_GetIntStatus() 함수를 이용하여 인터럽트 상태를 체크하고 상태에 따라 적절한 조치를 취한다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

// 축(채널) 번호 //
#define X_AXIS 0

HANDLE hIntEvent;
BOOL bAbortThread;

//*****
*// DisplayInterrupt () : 인터럽트 상태를 화면에 표시한다.
//*****
*
void DisplayInterrupt(long dwErrState, long dwEvtState)
{
    char *szErrState[] = {
        "Stop by +SL(Software limit) stop motion",
        "Stop by -SL(Software limit) stop motion",
        "",
        "Stop by General Comparator stop motion",
        "",
        "+EL signal is turning ON stop motion",
        "-EL signal is turning ON stop motion",
        "ALM signal is turning ON stop motion",
        "",
    }
```

```

        " ",
        "SD signal turning ON after deceleration",
        "Abnormal operation data stop motion",
        " ",
        " ",
        "PA/PB input buffer counter overflows",
        "In-position counter counts beyond the range at the time
of interpolation",
    };
    char *szEvtState[] = {
        "Normal Stop",
        "Succesive start of the next operation",
        " ",
        " ",
        "Start of acceleration",
        "End of acceleration",
        "Start of deceleration",
        "End of deceleration",
        " ",
        " ",
        "Position error tolerance exceed",
        "General Comparator",
        "Compared triggered for axis 0, 1",
        " ",
        "Latched for axis2,3",
        "ORG input signal ON",
        "SD input signal ON"
    };

    for(int i=0; i<16; i++){
        if(dwErrState & (1<<i))
            printf("Error Interrupt : %s\n", szErrState[i]);
    }

    for(i=0; i<16; i++){
        if(dwEvtState & (1<<i))
            printf("Event Interrupt : %s\n", szEvtState[i]);
    }
}

//*****
*// InterruptEventThread() : Interrupt Event 를 Waiting 하고 이벤트
가// 발생하면 이벤트를 처리하는 함수
//*****
*DWORD WINAPI InterruptEventThread( LPVOID pParam )
{
    HANDLE hDevice = (HANDLE)pParam;
    long nErrState, nEvtState;
    char szMessage[300];
    int i;
    bAbortThread = FALSE;

```

```

while(!bAbortThread)
{
    // Waiting for interrupt event //
    WaitForSingleObject(hIntEvent, INFINITE);
    // 이벤트가 발생하면 x 축에서 인터럽트가 발생된 것인지를 체크하고 //
    // x 축에서 발생한 것이면 인터럽트가 발생한 이유를 화면에 표시한다. //
    // 단, 프로그램이 종료할 때 이벤트를 강제로 발생시키므로 //
    // bAbortThread 가 TRUE 이면 강제로 발생된 이벤트이므로 처리하지 //
    // 않는다. //
    if(!bAbortThread &&
        COMILX_MC_GetAxisIntState(hDevice,X_AXIS))
    {
        COMILX_MC_GetIntStatus(hDevice, 0, &nErrState,
&nEvtState);
        DisplayInterrupt(nErrState, nEvtState);
    }
}
bAbortThread=FALSE;
return TRUE;
}

void main()
{
    double fDistList[2];

    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX501, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_Reset(hDevice);

    hIntEvent = CreateEvent(NULL, 0, 0, "");
    COMILX_MC_MaskInterrupt(hDevice, 0, 0xffff);
    COMILX_MC_EnableInterrupt(hDevice, hIntEvent);
    CreateThread(0, 0, InterruptEventThread, hDevice, 0, NULL);

    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);
    // (+)방향으로 Velocity Move 수행 //
    COMILX_MC_StartVMove(hDevice, X_AXIS, 1);
    // Stop 명령(키보드)이 있을 때 까지 Velocity Move 지속 //
    while(!kbhit())
    ;
    // 감속 후 정지 //
    COMILX_MC_Stop(hDevice, X_AXIS);

    bAbortThread=TRUE; // Monitor_Log_Thread 쓰레드 종료시킨다.
}

```

```
// Monitor_Log_Thread 쓰레드가 Event 를 Waiting 하고 있는 상태이면
//
// Wait 상태에서 빠져 나오도록 강제로 이벤트를 발생시킨다.      //
SetEvent(hIntEvent);
while(bAbortThread) // Monitor_Log_Thread 쓰레드가 종료될 때까지
기다린다.
;
CloseHandle(hIntEvent);
COMILX_UnloadDevice(hDevice);
COMILX_UnloadDll();
}
```

---

## ▣ 직선보간시 초기 속도 설정에 관하여

### 변경 사항

단축 모션에서는 속도 설정 함수인 COMILX\_MC\_SetSpeed() 함수에서는 초기속도 설정을 할 수 있으나, 직선보간 속도 설정 함수인 COMILX\_MC\_SetSpeedMx() 함수에서는 초기속도를 설정할 수 없습니다.

DLL 버전 3.0.0.7 이후에서는 COMILX\_MC\_SetSpeed()에서 지정한 초기속도가 직선보간시에도 적용되도록 수정되었습니다. 따라서 이전의 버전에서는 직선보간시에 가속을 0 (PPS)으로부터 시작하여 작업속도까지 수행해야하는데 반하여 DLL V3.0.0.7 이후에는 초기속도를 지정하여 사용자가 원하는 속도로부터 가속을 시작할 수 있어서 가속 시간을 단축할 수 있습니다.

## ■ 8 축 모션보드(COMI-LX508) 사용시 주의 사항

### 변경 사항

단축 모션에서는 속도 설정 함수인 COMILX\_MC\_SetSpeed() 함수에서는 초기속도 설정을 할 수 있으나, 직선보간 속도 설정 함수인 COMILX\_MC\_SetSpeedMx() 함수에서는 초기속도를 설정할 수 없습니다.

DLL 버전 3.0.0.7 이후에서는 COMILX\_MC\_SetSpeed()에서 지정한 초기속도가 직선보간시에도 적용되도록 수정되었습니다. 따라서 이전의 버전에서는 직선보간시에 가속을 0 (PPS)으로부터 시작하여 작업속도까지 수행해야하는데 반하여 DLL V3.0.0.7 이후에는 초기속도를 지정하여 사용자가 원하는 속도로부터 가속을 시작할 수 있어서 가속 시간을 단축할 수 있습니다.

---

## PART II. 라이브러리 업데이트 안내

제품과 함께 제공되는 매뉴얼이 인쇄된 이후에 사용자에게 보다 나은 기능을 제공하기 위하여 몇 가지 라이브러리의 함수가 추가되었습니다. 이 점에 대하여 사용자의 깊은 이해를 부탁드립니다.

추가된 라이브러리 함수는 본 문서에 함수별로 수록하였으므로 사용자께서는 인쇄된 매뉴얼과 더불어 본 문서를 함께 참조하시기 바랍니다.

제품과 함께 제공된 매뉴얼은 라이브러리 버전 “2,8,0,1”을 기준으로 작성되었습니다. 라이브러리(DLL) 버전은 윈도우 시스템 폴더(Win98 또는 ME의 경우에는 C:\Windows\System 폴더, Win2000 또는 XP의 경우에는 C:\Windows\System32)에 있는 “ComidasLX.dll” 파일의 등록정보에서 확인하실 수 있습니다.

매뉴얼이 인쇄된 이후에 추가된 라이브러리 함수는 [표 2-1]와 같습니다.

## Motion Control 라이브러리 업데이트 안내

함수설명	함수구분	라이브러리 버전	중요도
COMILX_MC_SetMioCfgLTC() : LATCH 입력 신호의 유형 및 LTC2 대상 카운터를 선택합니다.	Position Latch	V2.9.0.1	★
COMILX_MC_GetMioCfgLTC() : 현재 설정된 LATCH 입력 신호의 유형 및 LTC2 대상 카운터의 설정값을 얻어옵니다.	Position Latch	V2.9.0.1	★
COMILX_MC_ReadLatchCouner() : 래치된 카운트값을 반환합니다.	Position Latch	V2.9.0.1	★
COMILX_MC_GetLatchState() : 현재의 래치 상태를 반환합니다.	Position Latch	V2.9.0.1	★
COMILX_MC_SetMioCfgCMP() : 위치비교 출력 펄스(CMP 신호)의 출력 방식을 설정합니다.	위치비교 출력	V2.9.0.1	★
COMILX_MC_GetMioCfgCMP() : 위치비교 출력 펄스(CMP 신호)의 출력 설정값을 얻어옵니다.	위치비교 출력	V2.9.0.1	★
COMILX_MC_SetTriggerCompare() : 위치비교기를 설정합니다.	위치비교 출력	V2.9.0.1	★★
COMILX_MC_RegTableCCMP() : 연속적인 위치 비교 출력 위치 데이터를 등록합니다.	위치비교 출력	V2.9.0.1	★★
COMILX_MC_BuildTableCCMP() : 연속적인 위치 비교 출력 위치 데이터를 자동 생성하여 등록합니다.	위치비교 출력	V2.9.0.1	★★
COMILX_MC_StartCCMP() : 연속적인 위치 비교 출력 기능을 시작합니다.	위치비교 출력	V2.9.0.1	★★
COMILX_MC_StopCCMP() : 연속적인 위치 비교 출력 기능을 종료합니다.	위치비교 출력	V2.9.0.1	★★
COMILX_MC_ServoOn() : 지정한 채널(축)의 SERVO-ON 신호를 제어합니다.	V2.9 기타	V2.9.0.1	★★★★★
COMILX_MC_GetServoOn() : 지정한 채널(축)의 SERVO-ON 신호의 출력 상태를 반환합니다.	V2.9 기타	V2.9.0.1	★
COMILX_MC_SetELL() : +EL, -EL 신호의 입력 로직을 설정합니다.	V2.9 기타	V2.9.0.1	★
COMILX_MC_SetMioCfgSTA() : STA신호에 대한 환경을 설정합니다.	V2.9 기타	V2.9.0.1	★
COMILX_MC_SetMioCfgSTP() : STP신호에 대한 환경을 설정합니다.	V2.9 기타	V2.9.0.1	★
COMILX_MC_CompleteArc() : 원호보간의 종점좌표가 목표좌표가 불일치하는 경우 사용하면 좌표보정을 해줍니다.	V2.9 기타	V2.9.0.1	★
COMILX_MC_SetListMotionAxes() : 리스트모션에서 사용되는 모든 축들을 등록하는 함수입니다.	V2.9 기타	V2.9.0.1	★★★
COMILX_MC_BuildSpline() : 사용자가 입력한 데이터를 기반으로 Cubic spline 보간을 수행한 후 곡선 데이터를 담은 버퍼를 생성합니다.	스플라인보간	V3.0.0.1	★



함수설명	함수구분	라이브러리 버전	중요도
<b>COMILX_MC_DeleteSpline()</b> : COMILX_MC_BuildSpline() 함수에 의하여 생성된 스플라인 버퍼를 메모리 해제합니다.	스플라인보간	V3.0.0.1	★
<b>COMILX_MC_StartHelical()</b> : 헬리컬 보간 구동을 시작합니다.	헬리컬보간	V3.0.0.1	★
<b>COMILX_MC_AbortHelical()</b> : 현재 구동되고 있는 헬리컬 보간 작업을 취소합니다.	헬리컬보간	V3.0.0.1	★
<b>COMILX_MC_SetHelOnceSpeed()</b> : COMILX_MC_StartHelOnce() 함수나 COMILX_MC_HelOnce() 함수를 사용하여 헬리컬보간 작업을 수행할 때의 속도패턴을 설정합니다.	헬리컬보간	V3.1.0.1	★
<b>COMILX_MC_StartHelOnce()</b> : 2축 원호보간과 1축 또는 2축 직선보간을 동시에 시작하고 동시에 종료하는 헬리컬보간 구동을 시작합니다.	헬리컬보간	V3.1.0.1	★
<b>COMILX_MC_HelOnce()</b> : COMILX_MC_StartHelOnce() 함수와 동일한 작업을 수행합니다.	헬리컬보간	V3.1.0.1	★
<b>COMILX_MC_InitFromFile()</b> : 장치 초기화 파일로부터 정보를 전달받아서 모션 제어보드를 초기화하는 함수입니다.	V3.1 기타	V3.1.0.1	★★★★★
<b>COMILX_MC_HomeMoveAuto()</b> : 기구물이 어떤 위치에 있든 자동으로 원점을 찾아가도록 하는 기능이 보완된 원점복귀작업 명령입니다.	V3.1 기타	V3.1.0.1	★★★★★
<b>COMILX_MC_LmCurSequence()</b> : 리스트모션을 사용할 때 현재 수행되고 있는 작업을 알아보고자할 때 사용할 수 있도록 만들어진 함수입니다.	V3.1 기타	V3.1.0.1	★
<b>COMILX_MC_SetSpeedMx2</b> : Coordinated Motion의 속도 및 가/감속도를 설정합니다.	V3.1 기타	V3.1.0.1	★★
<b>COMILX_MC_EnableDebugLog()</b> : 라이브러리 디버그 로그 기능을 Enable 시키는 함수입니다.	V3.1 기타	V3.1.0.1	★★
<b>COMILX_MC_SetOutputMask</b> : 펄스 출력을 마스크하는 기능을 제공합니다.	V3.1 기타	V3.1.0.1	★

[표 2-1] 추가된 라이브러리 함수 리스트

## 1. Position Latch 관련 함수

Position Latch는 특정 순간에 Motion의 위치 관련 카운트값을 래치(Latch)하여 읽을 수 있도록하는 기능입니다. Position Latch는 LTC 입력핀에 LATCH 신호<sup>1</sup>가 입력되면 그 순간의 각 카운트값을 래치(Latch)합니다. 사용자는 COMILX\_MC\_GetLatchState()함수를 이용하여 래치 상태를 체크할 수 있으며, 래치가 되었으면 COMILX\_MC\_ReadLatchCouner() 함수를 이용하여 래치된 위치 카운트값을 읽을 수 있습니다.

LATCH 신호가 입력되었을 때 래치되는 카운터는 다음과 같이 4 가지입니다.

- **LTC0** : 명령위치 카운터(Command position counter)
- **LTC1** : 실제위치 카운터(Feedback position counter)
- **LTC2** : 편차 카운터(Deviation counter) 또는 현재의 속도값(PPS). LTC2 대상은 사용자가 선택할 수 있습니다.
- **LTC3** : General Counter

```
void COMILX_MC_SetMioCfgLTC(HANDLE hDevice, int nChannel, int nLtcLogic, int nLtc3Src)
```

LATCH 신호의 유형 및 LTC2 대상값을 설정합니다.

```
void COMILX_MC_GetMioCfgLTC(HANDLE hDevice, int nChannel, int *pnLtcLogic, int *pnLtc3Src)
```

현재 설정된 LATCH 신호의 유형 및 LTC2 대상의 설정값을 얻어옵니다.

```
BOOL COMILX_MC_GetLatchState(HANDLE hDevice, int nChannel)
```

현재의 Position Latch 상태를 반환합니다.

```
long COMILX_MC_ReadLatchCouner(HANDLE hDevice, int nChannel, int nCounter)
```

래치된 카운트값을 읽어옵니다.

<sup>1</sup> LATCH 신호는 LTC 입력 핀에 입력되는 신호를 의미하며 COMILX\_MC\_SetMioCfgLTC()함수를 이용하여 신호의 유형을 Falling Edge 또는 Rising Edge 중에서 선택할 수 있습니다.

## ■ COMILX\_MC\_SetMioCfgLTC

### 함수 원형

```
void COMILX_MC_SetMioCfgLTC(HANDLE hDevice, int nChannel, int nLtcLogic, int nLtc2Src)
```

### 함수 설명

LATCH 입력 신호의 유형 및 LTC2 대상 카운터를 선택합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *nLtcLogic* : LATCH 신호의 유형을 설정합니다.

Value	Meaning
0	Falling edge
1	Rising edge

- ▶ *nLtc2Src* : LTC2 대상 카운터를 선택합니다. 래치되는 카운터는 총 4 가지인데 이 중에서 3 번째 카운터(LTC2)는 다음 둘 중 하나를 선택할 수 있습니다.

Value	Meaning
0	편차카운터
1	현재의 펄스 출력 속도(PPS)

## ■ COMILX\_MC\_GetMioCfgLTC

### 함수 원형

```
void COMILX_MC_GetMioCfgLTC(HANDLE hDevice, int nChannel, int *pnLtcLogic, int *pnLtc2Src)
```

### 함수 설명

현재 설정된 LATCH 입력 신호의 유형 및 LTC2 대상 카운터의 설정값을 얻어옵니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *pnLtcLogic* : LATCH 신호의 유형 설정값을 반환받을 변수의 주소값. 반환되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	Falling edge
1	Rising edge

- ▶ *pnLtc2Src* : LTC2 대상 카운터의 설정값을 반환받을 변수의 주소값. 반환되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	편차카운터
1	현재의 펄스 출력 속도(PPS)

## ■ COMILX\_MC\_GetLatchState

### 함수 원형

```
BOOL COMILX_MC_GetLatchState(HANDLE hDevice, int nChannel)
```

### 함수 설명

현재의 래치 상태를 반환합니다. 사용자는 이 함수를 이용하여 래치상태를 체크한 후 래치되었으면 카운트값을 읽어야 합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)

### 반환값

현재 래치 상태를 반환합니다.

### 참 고

- 이 함수가 제대로 사용되기 위해서는 COMILX\_MC\_MaskInterrupt() 함수를 이용하여 BIT14 를 마스크(1 로 만들어줌) 해주어야 합니다.
- 래치상태를 읽으면 이전의 래치상태는 리셋(0)됩니다.
- COMILX\_MC\_GetIntStatus() 함수를 이용해서도 래치상태를 체크할 수 있으며, COMILX\_MC\_GetIntStatus() 함수를 수행하면 마찬가지로 이전의 래치상태는 리셋(0)됩니다.

### 사용예

- 아래의 예제는 유사 코드로써 실제로는 COMI\_LoadDll(), COMI\_LoadDevice() 등을 수행해야 합니다.
- 아래의 예제에서는 편의상 while() 루프를 이용하여 래치상태를 체크하고 있습니다만 실제로는 스레드 또는 타이머 이벤트 등을 이용하는 것이 좋습니다.

```
COMILX_MC_MaskInterrupt(hDevice, 0, 0xfff); /*BIT14 MASKING(본 예제에서는 전부다 마스크를 한 것이며 BIT14 만 MASKING 할려면 0xfff 대신에 (1<<14)를 입력하면 됩니다.*/
while(!COMILX_MC_GetLatchState(hDevice, 0))
;
comand_pos = COMILX_MC_ReadLatchCouner(hDevice, 0, 0);
feed_pos = COMILX_MC_ReadLatchCouner(hDevice, 0, 1);
deviation = COMILX_MC_ReadLatchCouner(hDevice, 0, 2);
gen_count = COMILX_MC_ReadLatchCouner(hDevice, 0, 3);
```

## ■ COMILX\_MC\_ReadLatchCounter

### 함수 원형

```
long COMILX_MC_ReadLatchCounter(HANDLE hDevice, int nChannel, int nCounter)
```

### 함수 설명

래치된 카운트값을 반환합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *nCounter* : 읽을 래치 카운터를 지정합니다. 이 값은 다음과 같습니다.

Value	Meaning
0	명령위치 카운터(Command position counter)
1	실제위치 카운터(Feedback position counter)
2	Deviation 또는 펄스 출력 속도.
3	General Counter

### 반환값

지정한 래치 카운터의 카운트값을 반환합니다.

### 참 고

- LTC2 카운터는 COMILX\_MC\_SetMioCfgLTC() 함수에서의 설정에 따라 Deviation counter 또는 펄스 출력 속도가 될 수 있습니다.

### 사용예

- 아래의 예제는 유사 코드로써 실제로는 COMI\_LoadDll(), COMI\_LoadDevice()등을 수행해야합니다.
- 아래의 예제에서는 편의상 while() 루프를 이용하여 래치상태를 체크하고 있습니다만 실제로는 스레드 또는 타이머 이벤트를 이용하는 것이 좋습니다.

```
COMILX_MC_MaskInterrupt(hDevice, 0, 0xffff); /*BIT14 MASKING(본 예제에서는 전부다 마스크를 한 것이며 BIT14 만 MASKING 할려면 0xffff 대신에 (1<<14)를 입력하면 됩니다.*/
while(!COMILX_MC_GetLatchState(hDevice, 0))
    ;
```

```
comand_pos = COMILX_MC_ReadLatchCouner(hDevice, 0, 0);  
feed_pos = COMILX_MC_ReadLatchCouner(hDevice, 0, 1);  
deviation = COMILX_MC_ReadLatchCouner(hDevice, 0, 2);  
gen_count = COMILX_MC_ReadLatchCouner(hDevice, 0, 3);
```

## 2. 위치비교 출력(CMP) 기능 관련 함수

COMI-LX502/4/8 모션 제어 보드는 각 축마다 위치비교 출력 기능을 제공합니다. 위치비교 출력 기능은 Command counter 또는 Position counter 의 카운트값이 사용자가 지정한 조건에 만족되면 **CMP** 출력핀을 통하여 트리거 펄스를 출력해주는 기능입니다.

이 기능을 사용하면 모션을 구동하면서 연속적으로 원하는 위치에서 외부기기에 하드웨어적인 트리거 신호를 제공할 수 있습니다. 특히 Machine Vision 시스템에서 유용하게 사용될 수 있습니다.

사용자는 COMILX\_MC\_SetTriggerCompare()함수를 이용하여 비교 조건을 설정할 수 있습니다. 비교기가 설정되고 비교조건이 만족되면 CMP 출력이 나가게 됩니다. CMP 출력 형태는 비교조건에 따라 One shot 펄스이거나 Active 상태를 유지하는 형태 두 가지가 될 수 있습니다. 이에 대해서는 COMILX\_MC\_SetTriggerCompare()함수를 참조하십시오.

여러 위치에서 연속적으로 위치비교 출력을 내고자하는 경우 COMILX\_MC\_StartCCMP() 함수를 사용하십시오. 이 함수를 사용하면 COMILX\_MC\_RegTableCCMP() 또는 COMILX\_MC\_BuildTableCCMP()함수를 통하여 등록된 위치 데이터를 순차적으로 자동 로딩>Loading)하여 연속적인 위치비교 출력을 낼 수 있습니다.

<b>void COMILX_MC_SetMioCfgCMP(HANDLE hDevice, int nChannel, ULONG nCmpPulseWidth, int nCmpLogic)</b> CMP 출력 신호의 유형을 설정합니다.
<b>void COMILX_MC_GetMioCfgCMP(HANDLE hDevice, int nChannel, ULONG *pnCmpPulseWidth, int *pnCmpLogic)</b> CMP 출력 신호 유형의 설정값을 얻어옵니다.
<b>void COMILX_MC_SetTriggerCompare(HANDLE hDevice, int nChannel, int nCmpSrc, int nCmpMethod, double fData)</b> 위치비교 조건을 설정합니다. 이 함수는 하나의 위치비교 조건만을 설정합니다.
<b>BOOL COMILX_MC_RegTableCCMP(HANDLE hDevice, int nChannel, double *pDataBuffer, int nNumData)</b> 임의의 연속적인 다중 위치데이터를 등록합니다.
<b>BOOL COMILX_MC_BuildTableCCMP(HANDLE hDevice, int nChannel, double fStartData, double fInterval, int nNumData)</b> 일정 간격을 가지는 연속적인 위치데이터를 자동생성하여 등록합니다.
<b>BOOL COMILX_MC_StartCCMP(HANDLE hDevice, int nChannel, int nCmpSrc, int nCmpMethod)</b> 연속적인 위치비교 출력 기능을 시작합니다.



**BOOL COMILX\_MC\_StopCCMP(HANDLE hDevice, int nChannel)**

연속적인 위치비교 출력 기능을 종료합니다.

## ■ COMILX\_MC\_SetMioCfgCMP

### 함수 원형



```
void COMILX_MC_SetMioCfgCMP(HANDLE hDevice, int nChannel, ULONG nCmpPulseWidth,
int nCmpLogic)
```

### 함수 설명

위치비교 출력 펄스(CMP 신호)의 출력 방식을 설정합니다. 연속적인 위치비교 출력 기능을 사용하는 경우에는 위치조건이 만족되면 One shot 펄스가 출력이 됩니다. 이때의 펄스의 폭 및 펄스의 출력 로직(Logic)을 설정합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *nCmpPulseWidth* : One shot 펄스의 펄스폭을 설정합니다. 실제 펄스폭은  $nCmpPulseWidth * 1.5(us)$  가 됩니다. 즉, 이 값을 1 로 하면 1.5us, 2 로 하면 3us... 와 같이 됩니다.
- ▶ *nCmpLogic* : CMP 펄스의 출력 로직(Logic)을 설정합니다.

Value	Meaning
0	Low active (  )
1	High active (  )

### 참 고

□ *nCmpPulseWidth* 파라미터값을 0 으로 하면 CMP 출력 신호의 펄스폭은 COMMAND 펄스의 펄스폭과 같게 됩니다.

□ COMILX\_MC\_SetTriggerCompare() 또는 COMILX\_MC\_StarttCCMP() 함수에서 *nCmpMethod* 파라미터를 4 또는 5 의 값으로 설정하면 CMP 출력 신호는 One shot 펄스가 아니고, 조건이 만족되는한 Active 상태를 유지하게 됩니다. 자세한 사항은 COMILX\_MC\_SetTriggerCompare() 함수 설명을 참조하시기 바랍니다.

## ▣ COMILX\_MC\_GetMioCfgCMP

### 함수 원형

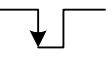

```
void COMILX_MC_GetMioCfgCMP (HANDLE hDevice, int nChannel, ULONG
*pnCmpPulseWidth, int *pnCmpLogic)
```

### 함수 설명

위치비교 출력 펄스(CMP 신호)의 출력 설정값을 얻어옵니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *pnCmpPulseWidth* : One shot 펄스의 펄스폭 설정값을 반환받을 변수의 주소값.
- ▶ *pnCmpLogic* : CMP 펄스의 출력 로직(Logic) 설정값을 반환받을 변수의 주소값

Value	Meaning
0	Low active (  )
1	High active (  )

## ■ COMILX\_MC\_SetTriggerCompare

### 함수 원형

```
void COMILX_MC_SetTriggerCompare(HANDLE hDevice, int nChannel, int nCmpSrc,
int nCmpMethod, double fData)
```

### 함수 설명

위치비교기를 설정합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *nCmpSrc* : 비교 대상 카운터를 설정합니다.

Value	Meaning
0	Command counter
1	Feedback counter
2	Deviation counter
3	General counter

- ▶ *nCmpMethod* : CMP 신호 출력 조건을 설정합니다. 즉, 비교 대상 카운트값과 사용자가 지정한 데이터 값을 어떻게 비교하여 CMP 신호 출력을 내보낼지를 결정합니다.

Value	Meaning
0	위치비교 출력 기능 Disable
1	Counting direction 과 관계없이 Count = fData 이면 One shot 펄스 출력.
2	카운트가 증가(Counting up)하는 시점에서 Count=fData 이면 One shot 펄스 출력.
3	카운트가 감소(Counting down)하는 시점에서 Count=fData 이면 One shot 펄스 출력.
4	Count < fData 이면 CMP 출력을 내보냅니다. 이때에는 One shot 펄스가 아니며, Count < fData 인 동안에는 CMP 출력이 Active 가 됩니다.

5	Count > fData 이면 CMP 출력을 내보냅니다. 이때에는 One shot 펄스가 아니며, Count > fData 인 동안에는 CMP 출력이 Active 가 됩니다.
---	---

▶ **fData** : 비교 대상 레퍼런스(Reference)값. 비교기는 지정한 카운트값을 이 값과 비교하여 CMP 출력을 내보낸다. 주의할 것은 이 값은 항상 펄스단위로 설정하여야 한다는 것입니다. 일부 사용자는 논리적 거리 단위를 펄스가 아닌 다른 단위로 설정할 수 있습니다. 이러한 때에도 이 값은 반드시 펄스단위로 설정하여야 합니다.

## 사용예

- 아래의 예제는 유사 코드로써 실제로는 COMI\_LoadDll(), COMI\_LoadDevice()등을 수행해야합니다.
- 아래의 예제에서는 X 축의 Command counter 값이 10000 이 되면 150us 펄스폭을 가지는 트리거 펄스를 출력하도록 하는 것입니다.

```
#define X_AXIS      0
#define LOW_ACTIVE  0
#define CMND_CNTR   0

...
...
COMILX_MC_SetMioCfgCMP(hDevice, X_AXIS, 100, LOW_ACTIVE);
COMILX_MC_SetTriggerCompare(hDevice,X_AXIS,CMND_CNTR,          1,
10000) ;
...
...
```

## ■ COMILX\_MC\_RegTableCCMP

### 함수 원형

```
BOOL COMILX_MC_RegTableCCMP(HANDLE hDevice, int nChannel, double *pDataBuffer,
    int nNumData)
```

### 함수 설명

연속적인 위치 비교 출력 기능을 사용하기 위해서 임의의 연속적인 위치 데이터를 등록합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *pDataBuffer* : 연속적인 위치 데이터를 담고 있는 버퍼(배열). 주의할 것은 이 버퍼에 설정되는 위치데이터는 항상 펄스단위로 설정하여야 한다는 것입니다. 일부 사용자는 논리적 거리 단위를 펄스가 아닌 다른 단위로 설정할 수 있습니다. 이러한 때에도 이 값은 반드시 펄스단위로 설정하여야 합니다.
- ▶ *nNumData* : 버퍼에 담겨있는 위치 데이터 수

### 사용예

□ 아래의 예제는 유사 코드로써 실제로는 COMI\_LoadDll(), COMI\_LoadDevice()등을 수행해야합니다.

□ 아래의 예제에서는 x 축을 0 에서 50000 좌표로 이동시키면서 Command counter 값이 1000, 5000, 10000, 20000, 30000 이 될 때 각각 트리거 펄스가 출력되도록 하는 예입니다. 따라서 연속적으로 총 5 회의 트리거 펄스가 출력되게 됩니다.

```
#define X_AXIS          0
#define LOW_ACTIVE      0
#define CMND_CNTR      0

double fDataBuf [5]={1000, 5000, 10000, 20000, 30000};
...
COMILX_MC_MoveTo(hDevice, X_AXIS, 0); // 원점으로 이동
...
// CMP 신호 설정 : 150us 펄스폭, low active //
COMILX_MC_SetMioCfgCMP(hDevice, X_AXIS, 100, LOW_ACTIVE);
// 연속적인 위치데이터 등록 //
COMILX_MC_RegTableCCMP(hDevice, X_AXIS, fDataBuf, 5);
// 연속적인 위치 비교 기능 시작 //
COMILX_MC_StartCCMP(hDevice, X_AXIS, CMND_CNTR, 1);
```

```
...
COMILX_MC_MoveTo(hDevice, X_AXIS, 50000); /* X 축을 50000 포인트로
이동한다. 이동하면서 지정한 위치마다 CMP 트리거 펄스가 출력되게 된다. */
COMILX_MC_StopCCMP(hDevice, X_AXIS); // 연속위치비교 기능 종료
```

## ■ COMILX\_MC\_BuildTableCCMP

### 함수 원형

```
BOOL COMILX_MC_BuildTableCCMP(HANDLE hDevice, int nChannel, double fStartData,
double fInterval, int nNumData)
```

### 함수 설명

연속적인 위치 비교 출력 기능을 사용하기 위해서 일정한 위치 간격을 가지는 연속적인 위치 데이터를 자동으로 생성하여 등록하도록 합니다.

이 함수는 일정한 위치 간격으로 CMP 출력을 내보낼 때 COMILX\_MC\_RegTableCCMP() 함수 대신에 사용할 수 있습니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *fStartData* : 시작 위치값. 주의할 것은 이 값은 항상 펄스단위로 설정하여야 한다는 것입니다. 일부 사용자는 논리적 거리 단위를 펄스가 아닌 다른 단위로 설정할 수 있습니다. 이러한 때에도 이 값은 반드시 펄스단위로 설정하여야 합니다.
- ▶ *fInterval* : 위치 간격. 이 또한 펄스 단위로 설정해야 합니다.
- ▶ *nNumData* : 총 데이터 수

### 사용예

□ 아래의 예제는 유사 코드로써 실제로는 COMI\_LoadDll(), COMI\_LoadDevice()등을 수행해야 합니다.

□ 아래의 예제에서는 x 축을 0 에서 50000 좌표로 이동시키면서 Command counter 값이 5000Q 부터 1000 씩 증가될 때마다 트리거 펄스가 출력되도록 하는 예입니다. 즉, 5000, 6000, 7000, ..., 44000, 45000 의 좌표에서 CMP 출력이 나가게 됩니다.

```
#define X_AXIS      0
#define LOW_ACTIVE  0
#define CMND_CNTR   0
```

...

## Motion Control 라이브러리 업데이트 안내

---

```
COMILX_MC_MoveTo(hDevice, X_AXIS, 0) ;// 원점으로 이동
...
// CMP 신호 설정 : 150us 펄스폭, low active //
COMILX_MC_SetMioCfgCMP(hDevice, X_AXIS, 100, LOW_ACTIVE);
// 연속적인 위치데이터 자동 생성 및 등록 : 5000 부터 1000 씩 증가하여 //
// 40 개의 데이터 포인트, 즉 45000 까지 데이터를 자동 생성하여 위치 //
// 데이터로 등록한다. //

COMILX_MC_BuildTableCCMP(hDevice, X_AXIS, 5000, 1000, 40);
// 연속적인 위치 비교 기능 시작 //
COMILX_MC_StartCCMP(hDevice, X_AXIS, CMND_CNTR, 1);
...
COMILX_MC_MoveTo(hDevice, X_AXIS, 50000); /* X 축을 50000 포인트로
이동한다. 이동하면서 지정한 위치마다 CMP 트리거 펄스가 출력되게 된다. */
COMILX_MC_StopCCMP(hDevice, X_AXIS); // 연속위치비교 기능 종료
...
```



## ■ COMILX\_MC\_StartCCMP

### 함수 원형

```
BOOL COMILX_MC_StartCCMP(HANDLE hDevice, int nChannel, int nCmpSrc, int nCmpMethod)
```

### 함수 설명

연속적인 위치 비교 출력 기능을 시작합니다. COMILX\_MC\_RegTableCCMP() 또는 COMILX\_MC\_BuildTableCCMP() 함수를 이용하여 등록된 연속적인 위치데이터를 비교기에 순차적으로 자동로딩하면서 연속적인 위치비교 출력기능을 수행합니다.

연속적인 위치비교 출력 기능은 현재 비교기에 로드된 비교조건이 만족되어 CMP 출력이 나가게 되면 인터럽트가 발생되어 드라이버 프로그램에서 사용자가 등록한 다음 데이터를 비교기에 자동으로 로드하도록 하는 기능입니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)
- ▶ *nCmpSrc* : 비교 대상 카운터를 설정합니다.

Value	Meaning
0	Command counter
1	Feedback counter
2	Deviation counter
3	General counter

- ▶ *nCmpMethod* : CMP 신호 출력 조건을 설정합니다. 즉, 비교 대상 카운트값과 사용자가 지정한 데이터 값을 어떻게 비교하여 CMP 신호 출력을 내보낼지를 결정합니다.

Value	Meaning
0	위치비교 출력 기능 Disable
1	Counting direction 과 관계없이 Count = fData 이면 One shot 펄스 출력.
2	카운트가 증가(Counting up)하는 시점에서 Count=fData 이면 One shot

	펄스 출력.
3	카운트가 감소(Counting down)하는 시점에서 Count=fData 이면 One shot 펄스 출력.

### 참 고

□ 연속적인 위치비교 출력 기능이 시작된 이후에 COMILX\_MC\_MaskInterrupt() 함수를 이용하여 BIT12 를 언마스크(0 으로 만듦)하면 인터럽트를 받을 수 없으므로 연속적인 위치비교 출력이 기능이 진행되지 않습니다.

□ 사용자가 등록한 모든 위치 데이터에 대하여 CMP 출력이 완료되었으면 COMILX\_MC\_StopCCMP() 함수를 호출하여 중지하지 않는한 처음 데이터부터 다시 비교기에 로드되어 연속적인 비교출력이 자동으로 재개됩니다. 그러나 현재 로드된 비교조건이 만족되기 전까지는 다음 위치데이터가 로드되지 않는다는 점에 유의하시기 바랍니다. 즉, 1000, 5000, 10000, 15000 의 연속적인 위치데이터를 등록한 후 모션이 10000 위치까지 이동 후 다시 0 위치로 복귀하고 다시 10000 위치로 이동했을 때 처음 10000 위치로 이동시에는 1000, 5000, 10000 의 위치에서 CMP 출력이 나가지만 두번째 10000 위치로 이동시에는 CMP 출력이 나가지 않습니다. 이유는 현재 비교기에 로드된 위치데이터가 15000 인데 이 조건이 만족되지 않았으므로 계속해서 비교기에 이 데이터가 남아있기 때문입니다.

### 사용예

□ 아래의 예제는 유사 코드로써 실제로는 COMI\_LoadDll(), COMI\_LoadDevice() 등을 수행해야 합니다.

□ 아래의 예제에서는 x 축을 0 에서 50000 좌표로 이동시키면서 Command counter 값이 1000, 5000, 10000, 20000, 30000 이 될 때 각각 트리거 펄스가 출력되도록 하는 예입니다. 따라서 연속적으로 총 5 회의 트리거 펄스가 출력되게 됩니다.

```
#define X_AXIS          0
#define LOW_ACTIVE      0
#define CMND_CNTR      0

double fDataBuf [5]={1000, 5000, 10000, 20000, 30000};
...
COMILX_MC_MoveTo(hDevice, X_AXIS, 0); // 원점으로 이동
...
// CMP 신호 설정 : 150us 펄스폭, low active //
COMILX_MC_SetMioCfgCMP(hDevice, X_AXIS, 100, LOW_ACTIVE);
// 연속적인 위치데이터 등록 //
COMILX_MC_RegTableCCMP(hDevice, X_AXIS, fDataBuf, 5);
```

```
// 연속적인 위치 비교 기능 시작 //
```

```
COMILX_MC_StartCCMP(hDevice, X_AXIS, CMND_CNTR, 1);
```

```
...
```

```
COMILX_MC_MoveTo(hDevice, X_AXIS, 50000); /* x 축을 50000 포인트로
```

```
이동한다. 이동하면서 지정한 위치마다 CMP 트리거 펄스가 출력되게 된다. */
```

```
COMILX_MC_StopCCMP(hDevice, X_AXIS); // 연속위치비교 기능 종료
```

## ■ COMILX\_MC\_StopCCMP

### 함수 원형

BOOL COMILX\_MC\_StopCCMP (HANDLE hDevice, int nChannel)

### 함수 설명

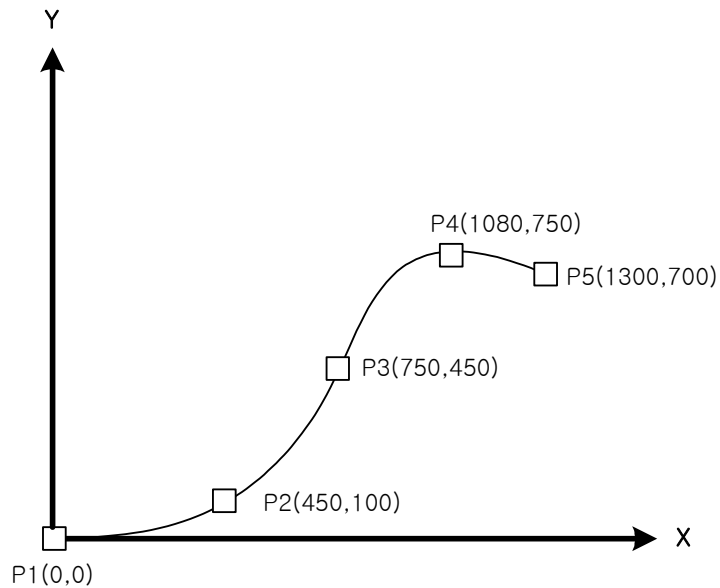
연속적인 위치 비교 출력 기능을 종료합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다)

### 3. 스플라인(Spline) 보간 기능 관련 함수

COMI-LX50x 모션 제어 보드는 Cubic spline interpolation 기능을 지원합니다. Cubic spline interpolation 은 아래 그림과 같이 사용자가 지정하는 데이터 포인트를 자유곡선으로 보간해주는 기능입니다.



본 라이브러리에서 제공하는 스플라인 보간 기능은 리스트모션과 함께 사용하면 효과적으로 스플라인 보간 구동을 수행할 수 있습니다.

```
double* COMILX_MC_BuildSpline(double fInPoints[], int nNumInPoints, int nNumOutPoints)
```

사용자가 지정한 데이터들을 기반으로 Cubic spline 보간 기법을 사용하여 보간된 데이터들을 담고 있는 버퍼를 생성해줍니다.

```
void COMILX_MC_DeleteSpline(double *pSplineBuffer)
```

COMILX\_MC\_BuildSpline() 함수에 의해 생성된 버퍼를 메모리 해제합니다.

## ■ COMILX\_MC\_BuildSpline

### 함수 원형

```
double* COMILX_MC_BuildSpline(double fInPoints[], int nNumInPoints, int nNumOutPoints)
```

### 함수 설명

사용자가 입력한 데이터를 기반으로 Cubic spline 보간을 수행한 후 곡선 데이터를 담은 버퍼를 생성합니다. 이 함수는 시간축을 매개변수로 하여 Cubic spline 보간을 수행합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *fInPoints* : 스플라인 보간을 수행할 샘플 데이터 포인트 배열
- ▶ *nNumInPoints* : 샘플 데이터의 수.
- ▶ *nNumOutPoints* : 스플라인 보간을 수행하여 자동 생성할 데이터 수. 이 값은 전체 곡선을 몇 개의 데이터 포인트로 곡선화할 것인지를 결정합니다.

### 참 고

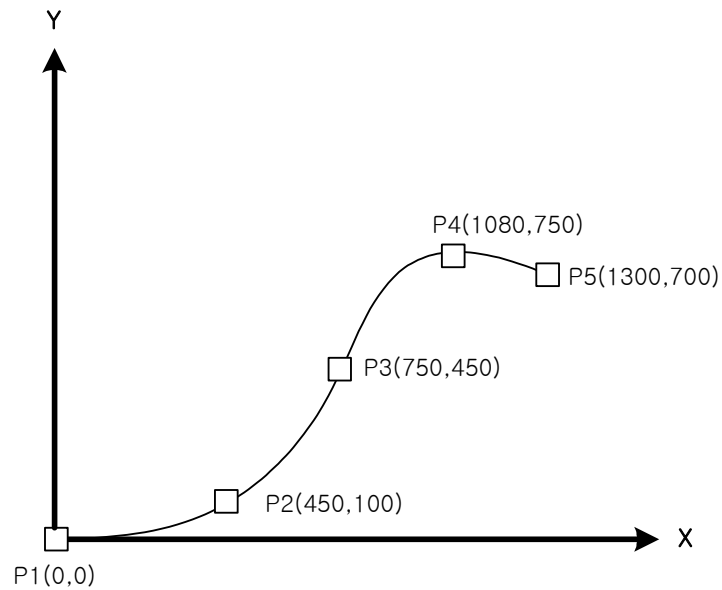
□ 이 함수는 스플라인 보간 기법을 이용하여 데이터를 생성해주는 역할만 하며 실제로 모션을 구동하지는 않습니다. 스플라인 보간 구동을 하기 위해서는 여기서 생성된 각 데이터 포인트를 COMILX\_MC\_LineTo() 함수 등을 이용하여 구동해주어야 합니다. 이 때 리스트모션(Listed Motion) 기법과 함께 사용하면 효과적으로 구동할 수 있습니다.

□ 이 함수는 nNumOutPoints 에서 지정한 크기만큼 버퍼를 동적으로 생성하게 됩니다. 따라서 스플라인 보간 구동이 완료된 후에는 COMILX\_MC\_DeleteSpline() 함수를 이용하여 메모리 해제를 해주어야 합니다.

□ 이 함수를 이용하면 2 차원 스플라인 보간뿐만 아니라 3 차원, 4 차원 스플라인 보간도 가능합니다.

### 사용예

□ 아래 그림과 같이 x,y 두 축에 대하여 5 개의 데이터 포인트를 이용하여 그림과 같은 곡선을 만들어내는 예제입니다.



```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#define X_AXIS    0
#define Y_AXIS    1
#define N_IN_POINTS  5
#define N_OUT_POINTS 100

void main()
{
    double x_point[N_IN_POINTS], y_point[N_IN_POINTS];
    double *spl_buffer[2], li_points[2];

    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_ServoOn(hDevice, X_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Y_AXIS, 1);

    // 샘플데이터 입력 //
    x_point[0] = 0;        y_point[0]=0;
    x_point[1] = 4500;     y_point[1]=1000;
```

## Motion Control 라이브러리 업데이트 안내

```
x_point[2] = 7500;    y_point[2]=4500;
x_point[3] = 10800;   y_point[3]=7500;
x_point[4] = 13000;   y_point[4]=7000;
// x 축 좌표에 대하여 스플라인 보간 수행 //
spl_buffer[0] = COMILX_MC_BuildSpline(x_point,  N_IN_POINTS,
N_OUT_POINTS);
// y 축 좌표에 대하여 스플라인 보간 수행 //
spl_buffer[1] = COMILX_MC_BuildSpline(y_point,  N_IN_POINTS,
N_OUT_POINTS);

COMILX_MC_MapAxes (hDevice, MAP0, X_MASK|Y_MASK);
COMILX_MC_SetSpeedModeMx (hDevice, MAP0, 1);
COMILX_MC_SetSpeedMx (hDevice, MAP0, 50000, 500000);

COMILX_MC_SetListMotionAxes (hDevice, X_MASK|Y_MASK);
COMILX_MC_BeginList(hDevice); // 모션 리스트 등록 시작 //
for(int i=0; i<N_OUT_POINTS; i++){
    li_points[0] = spl_buffer[0][i];
    li_points[1] = spl_buffer[1][i];
    COMILX_MC_LineTo (hDevice, MAP0, li_points);
}
COMILX_MC_EndList(hDevice); // 모션 리스트 등록을 마침 //

COMILX_MC_StartListMotion(hDevice); // 리스트 모션 수행 //

while(!COMILX_MC_CheckListMotionDone(hDevice) && !kbhit())
    Sleep(0);

COMILX_MC_DeleteSpline(spl_buffer[0]); // 스플라인 버퍼 해제
COMILX_MC_DeleteSpline(spl_buffer[1]); // 스플라인 버퍼 해제

COMILX_UnloadDevice(hDevice);
COMILX_UnloadDll();
}
```



## ■ COMILX\_MC\_DeleteSpline

### 함수 원형

```
void COMILX_MC_DeleteSpline(double *pSplineBuffer)
```

### 함수 설명

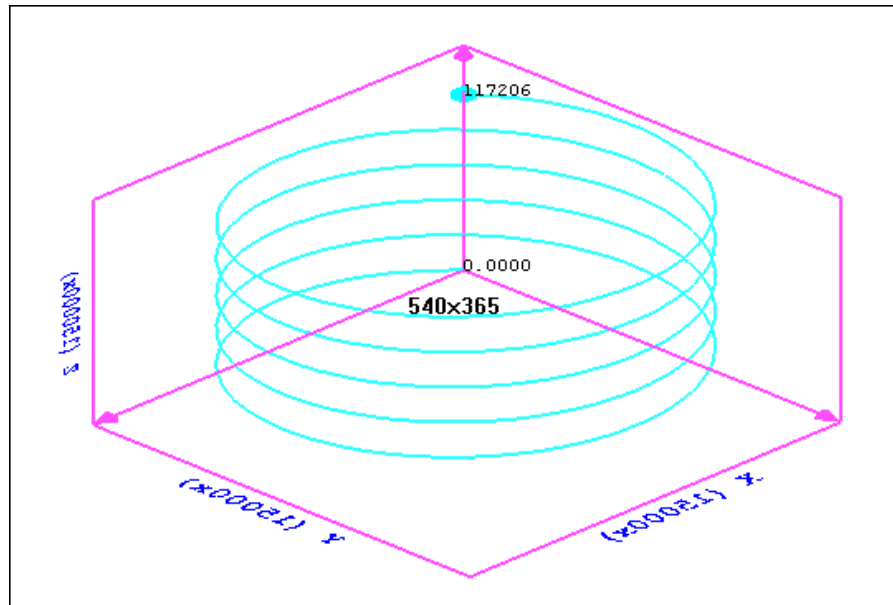
COMILX\_MC\_BuildSpline() 함수에 의하여 생성된 스플라인 버퍼를 메모리 해제합니다.

### 매개 변수

▶ *pSplineBuffer* : 스플라인 버퍼 포인터

## 4. 헬리컬(Helical) 보간 기능 관련 함수

COMI-LX50x 모션 제어 보드는 헬리컬 보간(Helical interpolation) 기능을 지원합니다. 헬리컬 보간 기능은 아래 그림과 같이 2 축 원호 보간과 1 축 직선 보간이 복합적으로 구동되는 기능입니다.



## ■ COMILX\_MC\_StartHelical

### 함수 원형

```
BOOL COMILX_MC_StartHelical (HANDLE hDevice, ThelicalUserInfo
*pHelicalUserInfo)
```

### 함수 설명

헬리컬 보관 구동을 시작합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *pHelicalUserInfo* : 헬리컬 보관 구동 정보를 담은 ThelicalUserInfo 형 구조체 변수의 주소값.

### 참 고

□ ThelicalUserInfo 구조체는 다음과 같이 구성됩니다. 단, 여기서 각 변수의 이름 또는 설명에서 언급하는 X,Y,Z 축은 개념상의 X,Y,Z 축입니다. 여기서 X,Y 축은 원호보간에 사용되는 두 축을 의미하며, Z 축은 직선보간으로 사용되는 축을 의미합니다.

```
typedef struct _ThelicalUserInfo{
    int c_map, z_axis;
    double c_xcen, c_ycen;
    int c_dir;
    int c_num;
    double c_la;
    double z_dist;
}ThelicalUserInfo;
```

**c\_map** : 원호보간으로 사용할 Axis map index

**z\_axis** : 직선보간으로 사용될 축 번호를 지정합니다. 이 변수의 이름은 편의상 Z 축으로 되어있으나 꼭 Z 축일 필요는 없습니다.

**c\_xcen** : 현재 좌표로부터 원호보간의 중심점까지의 X 축 상대 좌표값. 여기서 X 축은 편의상 X 축으로 표현한 것이며, 실제로는 원호보간으로 사용되는 두 축 중에서 축 번호가 낮은 축을 의미합니다.

**c\_ycen** : 현재 좌표로부터 원호보간의 중심점까지의 Y 축 상대 좌표값. 여기서 Y 축은 편의상 Y 축으로 표현한 것이며, 실제로는 원호보간으로 사용되는 두 축 중에서 축 번호가 높은 축을 의미합니다.

**c\_dir** : 원호보간의 구동 방향을 설정합니다. 양수 - 반시계방향(CCW), 0 또는음수-시계방향(CW)

**c\_num** : 전체 구동 중에 포함되어야할 원의 수(마지막 ARC 포함). 여기서 지정한 수

## Motion Control 라이브러리 업데이트 안내

의 원(마지막 ARC 포함)이 그려지면 Z 축에 지정한 거리에 도달하지 못하였다라도 헬리컬보간은 자동으로 종료됩니다.

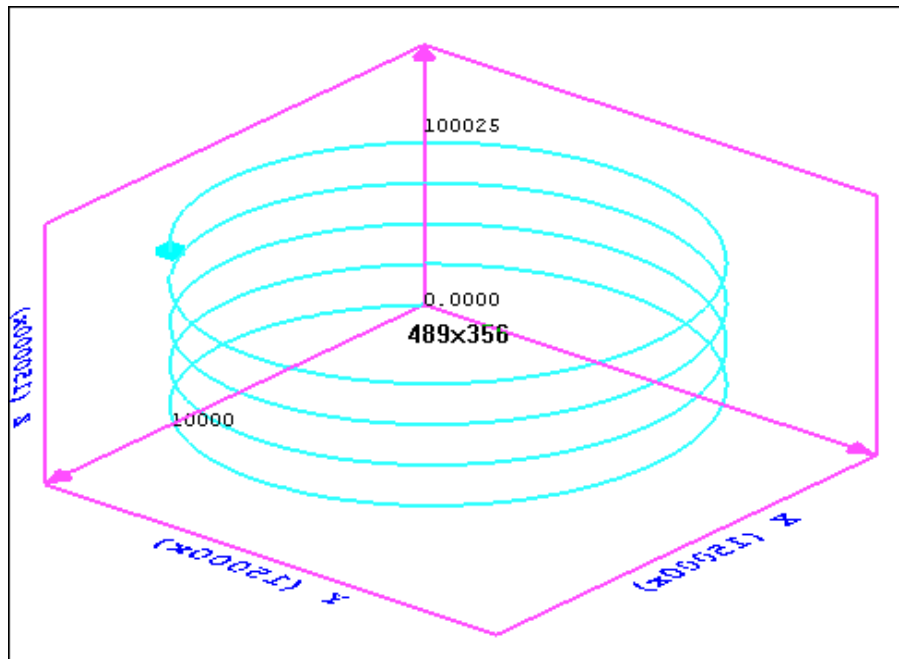
**c\_la** : 마지막 ARC 의 각도입니다. 이 각도는 시작점을 기준으로 X-Y 평면상의 각도를 의미합니다. 마지막 원호 보간은 완전한 원이 아니도록 할 수 있습니다. 예를 들어 처음 좌표를 기준으로 90 도 회전한 위치에서 헬리컬보간을 완료하도록 하고자 한다면 이 값을 90 또는 -90 으로 해주면 됩니다.

**z\_dist** : Z 축이 이동해야할 거리(상대좌표)를 지정합니다. Z 축이 지정한 거리만큼 이동이 완료되면 헬리컬보간은 자동으로 종료됩니다.

□ 헬리컬 보간이 종료되는 조건은 Z 축이 z\_dist 만큼 이동하거나 Z 축이 강제 종료된 경우, 또는 c\_num 에서 지정한 수만큼 원호보간을 완료하면 자동종료됩니다. 세가지 조건 중에서 하나만 만족되면 자동 종료됩니다. 따라서 만일 원호보간의 수나 마지막 ARC 의 각도가 중요한 경우에는 처음에는 z\_dist 를 충분히 하여 시뮬레이션해보는 것이 좋습니다.

### 사용예

□ 아래 그림과 같이 x,y 축에 대하여 5 회의 원호보간을 수행하면서 동시에 z 축을 이동하면서 헬리컬보간을 수행하는 예제입니다. 이때 마지막 ARC 의 각도는 90 도 하여 처음 시작점을 기준으로 90 도 원호를 그리고 종료하도록 합니다.



```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

// 축번호 //
#define X_AXIS 0
#define Y_AXIS 1
#define Z_AXIS 2
// 축마스크값 Axis Map 구성시 사용//
#define X_MASK 1
#define Y_MASK 2

void main()
{
    THelicalUserInfo HelicalInf;

    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_ServoOn(hDevice, X_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Y_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Z_AXIS, 1);

    COMILX_MC_SetSpeedMode(hDevice, Z_AXIS, 0);
    COMILX_MC_SetSpeed(hDevice, Z_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, Z_AXIS, 100000, 100000);

    // Map X & Y axis //
    COMILX_MC_MapAxes(hDevice, MAP0, X_MASK|Y_MASK);
    COMILX_MC_SetSpeedModeMx(hDevice, MAP0, 0);
    COMILX_MC_SetSpeedMx(hDevice, MAP0, 20000, 50000);

    HelicalInf.c_dir = 1; // 원호보관 방향 : 반시계방향
    HelicalInf.c_map = MAP0; // 원호보관 맵번호
    HelicalInf.c_xcen = 5000; // 원호보관 중심점 x 좌표
    HelicalInf.c_ycen = 5000; // 원호보관 중심점 y 좌표
    HelicalInf.c_num = 5; // 원의 수 : 5 개 (마지막 90 도 원호포함)
    HelicalInf.c_la = 90; // 마지막 ARC 각도 : 90 도
    HelicalInf.z_axis = Z_AXIS; // Up/Down 축 : Z 축
    HelicalInf.z_dist = 150000; // Z 축 이동거리

    COMILX_MC_StartHelical(hDevice, &HelicalInf);
    Sleep(500);
    while(!COMILX_MC_Done(hDevice, Z_AXIS))
        Sleep(0);
}
```

## Motion Control 라이브러리 업데이트 안내

---

```
Sleep(200);  
COMILX_UnloadDevice(hDevice);  
COMILX_UnloadDll();  
}
```

## ■ COMILX\_MC\_AbortHelical

### 함수 원형

BOOL COMILX\_MC\_AbortHelical (HANDLE hDevice)

### 함수 설명

현재 구동되고 있는 헬리컬 보간 작업을 취소합니다.

### 매개 변수

▶ *hDevice* : 디바이스 핸들.

## ■ COMILX\_MC\_SetHelOnceSpeed

### 함수 원형

```
void COMILX_MC_SetHelOnceSpeed (HANDLE hDevice, int nMaster, int nSpeedMode,
double fSpeed, double fAcc, double fDec)
```

### 함수 설명

COMILX\_MC\_StartHelOnce() 함수나 COMILX\_MC\_HelOnce() 함수를 사용하여 헬리컬보간 작업을 수행할 때의 속도패턴을 설정합니다.

### 매개 변수

- ▶ **hDevice** : 디바이스 핸들.
- ▶ **nMaster** : 속도 설정의 마스터를 설정합니다. 이 값에 따라서 지정한 속도패턴의 의미가 다음과 같이 달라집니다.

Value	Meaning
0	설정된 속도는 벡터 속도로 적용됩니다. 이때 원호보간 속도와 Z 축의 속도는 벡터거리에 대한 상대적인 거리비에 따라 결정됩니다.
1	설정된 속도는 원호보간의 속도로 적용됩니다. Z 축의 속도는 원호보간 거리에 대한 상대적인 거리비에 따라 결정됩니다.
2	설정된 속도는 Z 축의 속도로 적용됩니다. 원호보간 속도는 Z 축 이동 거리에 대한 상대적인 거리비에 따라 결정됩니다.

- ▶ **nSpeedMode** : 속도 모드를 설정합니다.

Value	Meaning
0	Constant speed mode
1	Trapezoidal speed mode
2	S-curve speed mode

- ▶ **fAccel** : Coordinated Motion 의 가속도를 지정합니다. 단, 이 값을 0 으로 설정하면 가속구간이 없이 즉시 작업속도로 올라갑니다(실제로는 가속값이 무한대).

- ▶ **fDecel** : Coordinated Motion 의 감속도를 지정합니다. 단, 이 값을 0 으로 설정하면 감속구간이 없이 즉시 정지합니다(실제로는 감속값이 무한대).



## 사용예

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

// 축번호 //
#define X_AXIS    0
#define Y_AXIS    1
#define Z_AXIS    2
#define U_AXIS    3

void main()
{
    int nChanList[4] = {0,1,2,3};
    double fCoords[4];

    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    //////////////////////////////////////
    // CME 파일을 이용한 장치 초기화
    // 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
    // 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
    // 존재한다.
    char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
    GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
    패스를 얻는다.
    sprintf(szFilePath, "%s\\Default.cme", szSysDir);
    if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
        printf("장치 초기화 파일을 로드할 수 없습니다.");
    }

    COMILX_MC_ServoOn(hDevice, X_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Y_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Z_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, U_AXIS, 1);

    // 헬리컬보간 속도 설정 : z 축 속도를 v=5000, acc=25000, //
    // dec=25000 으로 설정한다. 원호보간 속도는 원호보간 이동거리와 //
    // z 축 이동거리의 비에 따라 자동 설정된다. //
    COMILX_MC_SetHelOnceSpeed (hDevice, 2, 1, 5000, 25000, 25000);
    // 헬리컬보간 이동 : 원호중심=(5000,5000), 원호이동각도=360, //
    // z 축 이동량=3000, U 축 Direction=+방향 //
    fCoords[0] = 5000; fCoords[1] = 5000;
```

## Motion Control 라이브러리 업데이트 안내

---

```
fCoords[2] = 3000; fCoords[3] = 1;
COMILX_MC_HelOnce (hDevice, nChanList, 4, fCoords, 360);

COMILX_UnloadDevice(hDevice);
COMILX_UnloadDll();
}
```

## ■ COMILX\_MC\_StartHelOnce

### 함수 원형

```
double COMILX_MC_StartHelOnce (HANDLE hDevice, int nAxisList[], int nNumAxis,
double fCoordList[], double fArcAngle)
```

### 함수 설명

2 축 원호보간과 1 축 또는 2 축 직선보간을 동시에 시작하고 동시에 종료하는 헬리컬보간 구동을 시작합니다. 헬리컬보간 구동에서 원호보간 이동은 그 속도와 이동량이 U 축(4 축 모션보드인 경우 CH3, 8 축 모션보드인 경우 CH3 또는 CH8)과 동기되어 움직입니다. 따라서 U 축은 헬리컬보간에서 반드시 포함되어야 하며, 채널리스트의 마지막 축이 반드시 U 축으로 설정되어야 합니다. 그리고 U 축은 원호보간과 동기되어 움직이므로 이동량이나 속도가 원호보간과 다르게 사용할 수는 없습니다.

원호보간을 위한 2 축과 Z 축은 4 축 모션보드인 경우에 CH0, CH1, CH2 그리고 8 축 모션보드인 경우에 CH0, CH1, CH2, CH4, CH5, CH6 축을 임의로 조합하여 사용할 수 있습니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nAxisList* : 헬리컬보간에 사용되는 축 배열의 주소값. 이 배열의 마지막 Element 값은 4 축 모션 보드인 경우에는 반드시 3 이어야 하며 8 축 모션 보드인 경우에는 반드시 3 또는 7 이어야 합니다. 이 배열의 구성은 다음과 같이 하면 됩니다.
  - nAxisList*[0] : 원호보간의 X 축.
  - nAxisList*[1] : 원호보간의 Y 축.
  - nAxisList*[2] : Z 축 (3 축만 사용하는 경우에는 U 축)
  - nAxisList*[3] : U 축 (3 축만 사용하는 경우에는 무시됨)
- ▶ *nNumAxis* : 헬리컬보간에 사용되는 축 수. 이 값은 3 또는 4 이어야 합니다.
- ▶ *fCoordList* : 좌표 배열 주소. 3 축을 사용하는 경우와 4 축을 사용하는 경우에 이 배열의 구성은 다음과 같이 하면 됩니다.

#### □ 3 축을 사용하는 경우

- nCoordList*[0] : 원호 중심의 X 상대좌표값
- nCoordList*[1] : 원호 중심의 Y 상대좌표값
- nCoordList*[2] : U 축 방향 (0 또는 음수 - 음의 방향, 양수 - 양의 방향)

#### □ 4 축을 사용하는 경우

nCoordList[0] : 원호 중심의 X 상대좌표값  
nCoordList[1] : 원호 중심의 Y 상대좌표값  
nCoordList[2] : Z 축 이동거리(상대좌표)  
nCoordList[3] : U 축 방향 (0 또는 음수 - 음의 방향, 양수 - 양의 방향)

▶ *fArcAngle* : 원호보간 이동 각도. 이 값이 음수이며 시계방향으로 양수이면 반 시계 방향으로 원호를 그리게 되며, 이 값의 절대값이 360 보다 클수 없다.

## 사용예 1

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

// 축번호 //
#define X_AXIS    0
#define Y_AXIS    1
#define Z_AXIS    2
#define U_AXIS    3

void main()
{
    THelicalUserInfo HelicalInf;

    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    //////////////////////////////////////
    // CME 파일을 이용한 장치 초기화
    // 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
    // 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
    // 존재한다.
    char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
    GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
    패스를 얻는다.
    sprintf(szFilePath, "%s\\Default.cme", szSysDir);
    if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
        printf("장치 초기화 파일을 로드할 수 없습니다.");
    }

    COMILX_MC_ServoOn(hDevice, X_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Y_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Z_AXIS, 1);
}
```

```

COMILX_MC_ServoOn(hDevice, U_AXIS, 1);

int nChanList[4] = {0,1,2,3};
double fCoords[4];
// 헬리컬보간 속도 설정 : z 축 속도를 v=5000, acc=25000,          //
// dec=25000 으로 설정한다. 원호보간 속도는 원호보간 이동거리와    //
// z 축 이동거리의 비에 따라 자동 설정된다.                        //
COMILX_MC_SetHelOnceSpeed (hDevice, 2, 1, 5000, 25000, 25000);
// 헬리컬보간 이동 : 원호중심=(5000,5000), 원호이동각도=360,      //
// z 축 이동량=3000, U 축 Direction=+방향                          //
fCoords[0] = 5000; fCoords[1] = 5000;
fCoords[2] = 3000; fCoords[3] = 1;
COMILX_MC_HelOnce (hDevice, nChanList, 4, fCoords, 360);

COMILX_UnloadDevice(hDevice);
COMILX_UnloadDll();
}

```

## 사용예 2 : 연속적인 헬리컬 보간

□ 아래 그림과 같이 x,y 축에 대하여 5 회의 원호보간을 수행하면서 동시에 z 축을 이동하면서 헬리컬보간을 수행하는 예제입니다. 이때 마지막 ARC 의 각도는 90 도 로 하여 처음 시작점을 기준으로 90 도 원호를 그리고 종료하도록 합니다.

```

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

// 축번호 //
#define X_AXIS    0
#define Y_AXIS    1
#define Z_AXIS    2
#define U_AXIS    3

void main()
{
    int nChanList[4] = {X_AXIS, Y_AXIS, Z_AXIS, U_AXIS};
    double fCoords[4];

    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    //////////////////////////////////////
    // CME 파일을 이용한 장치 초기화

```

## Motion Control 라이브러리 업데이트 안내

```
// 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
// 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
// 존재한다.
char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
패스를 얻는다.
sprintf(szFilePath, "%s\\Default.cme", szSysDir);
if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
    printf("장치 초기화 파일을 로드할 수 없습니다.");
}

COMILX_MC_ServoOn(hDevice, X_AXIS, 1);
COMILX_MC_ServoOn(hDevice, Y_AXIS, 1);
COMILX_MC_ServoOn(hDevice, Z_AXIS, 1);
COMILX_MC_ServoOn(hDevice, U_AXIS, 1);

COMILX_MC_SetListMotionAxes(hDevice, X_MASK|Y_MASK);
COMILX_MC_BeginList(hDevice); // 모션 리스트 등록 시작 //
// 처음 이동은 가속=25000, 감속구간 없음으로 설정 //
COMILX_MC_SetHelOnceSpeed (hDevice, 2, 1, 5000, 25000, 0);
// 헬리컬보간 이동 : 원호중심=(5000,5000), 원호이동각도=360, //
// z 축 이동량=3000, u 축 Direction=+방향 //
fCoords[0] = 5000; fCoords[1] = 5000;
fCoords[2] = 3000; fCoords[3] = 1;
COMILX_MC_StartHelOnce (hDevice, nChanList, 4, fCoords, 360);
// 두번째 부터는 Constant 속도 모드로 설정 //
COMILX_MC_SetHelOnceSpeed (hDevice, 2, 0, 5000, 0, 0);
fCoords[0] = 5000; fCoords[1] = 5000;
fCoords[2] = 3000; fCoords[3] = 1;
COMILX_MC_StartHelOnce (hDevice, nChanList, 4, fCoords, 360);
fCoords[0] = 5000; fCoords[1] = 5000;
fCoords[2] = 3000; fCoords[3] = 1;
COMILX_MC_StartHelOnce (hDevice, nChanList, 4, fCoords, 360);
fCoords[0] = 5000; fCoords[1] = 5000;
fCoords[2] = 3000; fCoords[3] = 1;
COMILX_MC_StartHelOnce (hDevice, nChanList, 4, fCoords, 360);
// 마지막 이동은 가속구간 없음, 감속=25000 으로 설정 //
COMILX_MC_SetHelOnceSpeed (hDevice, 2, 1, 5000, 25000, 0);
fCoords[0] = 5000; fCoords[1] = 5000;
fCoords[2] = 3000; fCoords[3] = 1;
COMILX_MC_StartHelOnce (hDevice, nChanList, 4, fCoords, 360);

COMILX_MC_EndList(hDevice); // 모션 리스트 등록을 마침 //

COMILX_MC_StartListMotion(hDevice); // 리스트 모션 수행 //
// 리스트 모션이 모두 완료될 때까지 기다림 //
while(!COMILX_MC_ChekcListMotionDone(hDevice)){
    // 현재 수행되는 작업 번호를 체크하여 화면에 표시 //
    nCurOper = COMILX_MC_LmCurSequence(hDevice);
    printf("Current Operation = %d\n", nCurOper);
}
```

```
        Sleep(0);  
    }  
  
    COMILX_UnloadDevice(hDevice);  
    COMILX_UnloadDll();  
}
```

## ■ COMILX\_MC\_HeIOnce

### 함수 원형

```
double COMILX_MC_HeIOnce (HANDLE hDevice, int nAxisList[], int nNumAxis,  
double fCoordList[], double fArcAngle)
```

### 함수 설명

이 함수는 COMILX\_MC\_StartHeIOnce() 함수와 동일한 작업을 수행합니다. 단, COMILX\_MC\_StartHeIOnce() 함수가 헬리컬보간을 시작한 후 바로 리턴되는데 반하여, 이 함수를 사용하면 내부적으로 모션이 완료될때까지 기다리며 모션이 완료된 후 리턴된다는 것이 다릅니다. 따라서 이 함수에 대한 설명은 COMILX\_MC\_StartHeIOnce() 함수 설명을 참조하시기 바랍니다.

리스트모션에 등록할 때에는 COMILX\_MC\_HeIOnce() 함수와 COMILX\_MC\_StartHeIOnce() 함수는 완전히 동일하게 취급됩니다.



## 5. 라이브러리 V2.9 에서 추가된 기타 함수

이 단원에서는 라이브러리(DLL) V2.9.0.1 버전에서 추가된 함수들 중에서 앞에서 소개되지 않은 기타 함수들을 소개합니다.

### ▣ COMILX\_MC\_ServoOn

#### 함수 원형

```
void COMILX_MC_ServoOn(HANDLE hDevice, int nChannel, BOOL bEnable)
```

#### 함수 설명

지정한 채널(축)의 SERVO-ON 신호를 제어합니다.

이 함수는 COMI-LX501 제품에는 적용되지 않습니다.

#### 매개 변수

- ▶ *hDevice* : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다).
- ▶ *bEnable* : SERVO-ON 제어값. 0=>OFF, 1=>ON

#### 참 고

서보 드라이버를 사용하실 때는 외부에서 스위치를 이용하여 서보드라이버의 ON/OFF를 제어할 수 있도록 하는데, 이를 SERVO-ON 신호라 합니다. 따라서 서보드라이버를 사용하여 모터를 구동하는 경우에는 반드시 이 함수를 이용하여 SERVO-ON 신호를 Enable 시켜야 합니다.

단, 이 함수는 COMI-LX501 보드에서는 지원하지 않습니다. COMI-LX501에서는 디지털 출력 채널을 이용하여 SERVO-ON 제어를 하셔야합니다.

## ■ COMILX\_MC\_GetServoOn

### 함수 원형

```
int COMILX_MC_ServoOn(HANDLE hDevice, int nChannel)
```

### 함수 설명

지정한 채널(축)의 SERVO-ON 신호의 출력 상태를 반환합니다. 이 함수는 COMI-LX501 제품에는 적용되지 않습니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다).

### 반환값

현재 출력되고 있는 SERVO-ON 신호의 상태를 반환합니다.

## ■ COMILX\_MC\_SetELL

### 함수 원형

```
void COMILX_MC_SetELL(HANDLE hDevice, int nChannel, int nLogic)
```

### 함수 설명

+EL, -EL 신호의 입력 로직을 설정합니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.
- ▶ *nChannel* : 채널(축) 번호(0 부터 시작합니다).
- ▶ *nLogic* : +EL, -EL 신호의 입력 로직 설정

Value	Meaning
0	Low active 또는 Normal Open
1	High active 또는 Normal Close

## ■ COMILX\_MC\_SetMioCfgSTA

### 함수 원형

```
void COMILX_MC_SetMioCfgSTA(HANDLE hDevice, int nChannel, int nMode, int nInputType)
```

### 함수 설명

STA 신호에 대한 환경을 설정합니다. STA 신호는 외부에서 입력되는 모션 시작 신호입니다. 이 함수에서 nMode 를 0 으로 하면 STA 입력 신호는 무시됩니다. nMode 를 1 로 하면 COMILX\_MC\_Move(), COMILX\_MC\_MoveAll()과 같은 모든 이동 명령이 외부에서 입력되는 STA 입력 신호와 동기되어 시작됩니다.

이 함수는 COMI-LX501 제품에는 적용되지 않습니다.

### 매개 변수

▶ **hDevice** : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

▶ **nChannel** : 채널(축) 번호(0 부터 시작합니다).

▶ **nMode** : STA 신호의 적용 여부를 결정합니다.

Value	Meaning	Description
0	Start by software	STA 입력 신호는 무시됩니다. 즉, 모든 이동 명령이 호출되면 바로 이동을 시작합니다.
1	Start by STA hardware signal input	COMILX_MC_Move(), COMILX_MC_MoveAll, COMILX_MC_StartVmoveAll() 등과 이동 명령이 호출되면 바로 이동을 시작하지 않고 STA 입력핀에 트리거 신호가 입력되면 이동을 시작합니다.

▶ **nInputType** : STA 핀에 입력되는 트리거 신호 유형을 설정합니다.

Value	Meaning	Description
0	Level trigger(Low)	STA 입력핀이 Low level 이 되면 모션을 시작합니다.
1	Edge trigger(Falling)	STA 입력핀에 Falling edge 신호가 입력되면 모션을 시작합니다.

## 참 고

□ nMode 값을 1 로 하면 모든 이동 명령이 STA 입력이 ON 되기전까지 시작되지 않습니다. 따라서 STA 동기 구동이 필요없는 경우에는 nMode 값을 0 으로 하여주어야 합니다.

□ STA 입력핀은 모든 축에 대해 공통으로 사용됩니다. 따라서 STA 입력을 각 축별로 따로 입력할 수는 없습니다. 그러나 STA 입력 환경은 각 축별로 다르게 설정할 수 있습니다.

## 사용예 1

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#define X_AXIS      0
#define STA_BY_SW   0 // Start by software
#define STA_BY_HW   1 // Start by hardware(STA input)
#define TRG_EDGE    1 // Falling edge trigger
void main()
{
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_ServoOn(hDevice, X_AXIS, 1);

    // x 축에 대하여 STA 신호의 환경을 설정한다. STA 입력에 Falling- //
    // edge 신호가 입력되면 이동이 시작되도록 한다. //
    COMILX_MC_SetMioCfgSTA(hDevice, X_AXIS, STA_BY_HW, TRG_EDGE);

    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);

    // x 축을 현재위치로부터 10000 만큼 이동한다. 단, STA 신호가 //
    // 입력되기 전까지는 이동을 시작하지 않는다. //
    COMILX_MC_Move(hDevice, X_AXIS, 10000);

    // x 축에 대하여 STA 신호를 무시하도록 환경을 설정한다. //
    COMILX_MC_SetMioCfgSTA(hDevice, X_AXIS, STA_BY_SW, 0);
```

## Motion Control 라이브러리 업데이트 안내

```
// x 축을 현재위치로부터 10000 만큼 이동한다. 이때는 STA 신호와 //  
// 관계없이 곧바로 이동을 시작한다. //  
COMILX_MC_Move(hDevice, X_AXIS, 10000);  
  
COMILX_UnloadDevice(hDevice);  
COMILX_UnloadDll();  
}
```

### 사용예 2

□ 아래의 예제에서는 x,y 축을 STA 입력 신호와 동기를 맞추어 동시에 구동하는 예입니다.

```
#include <windows.h>  
#include <stdio.h>  
#include <conio.h>  
#include "comidaslx.h"  
  
#define X_AXIS 0  
#define Y_AXIS 1  
#define STA_BY_SW 0 // Start by software  
#define STA_BY_HW 1 // Start by hardware(STA input)  
#define TRG_EDGE 1 // Falling edge trigger  
  
void main()  
{  
    int nAxisList[2]={X_AXIS, Y_AXIS};  
    double fDistList[2]={20000, 40000};  
    if(!COMILX_LoadDll())  
        exit(-1); // Load Dll Failure  
  
    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);  
    if(hDevice == INVALID_HANDLE_VALUE)  
        exit(-1); // Load Device Failure  
  
    COMILX_MC_ServoOn(hDevice, X_AXIS, 1);  
    COMILX_MC_ServoOn(hDevice, Y_AXIS, 1);  
  
    // x,y 축에 대하여 STA 신호의 환경을 설정한다. STA 입력에 Falling- //  
    // edge 신호가 입력되면 이동이 시작되도록 한다. //  
    COMILX_MC_SetMioCfgSTA(hDevice, X_AXIS, STA_BY_HW, TRG_EDGE);  
    COMILX_MC_SetMioCfgSTA(hDevice, Y_AXIS, STA_BY_HW, TRG_EDGE);  
  
    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);  
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);  
    COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);  
  
    COMILX_MC_SetSpeedMode(hDevice, Y_AXIS, 1);  
    COMILX_MC_SetSpeed(hDevice, Y_AXIS, 0, 20000);
```

```
COMILX_MC_SetAccel(hDevice, Y_AXIS, 40000, 40000);  
COMILX_MC_MoveAll (hDevice, 2, nAxisList, fDistList);  
COMILX_UnloadDevice(hDevice);  
COMILX_UnloadDll();  
}
```

## ■ COMILX\_MC\_SetMioCfgSTP

### 함수 원형

```
void COMILX_MC_SetMioCfgSTP(HANDLE hDevice, int nChannel, int nMode)
```

### 함수 설명

STP 신호에 대한 환경을 설정합니다. STP 신호는 외부에서 입력되는 모션 종료 신호입니다. 이 함수에서 nMode 를 0 으로 하면 STP 입력 신호는 무시됩니다. nMode 를 1 로 하면 외부에서 STP 에 신호를 입력하여 모션을 종료합니다.

STP 신호는 외부에서 하드웨어적으로 모션 구동을 종료할 수 있도록 하기 위해 마련된 입력신호입니다.

이 함수는 COMI-LX501 제품에는 적용되지 않습니다.

### 매개 변수

- ▶ **hDevice** : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- ▶ **nChannel** : 채널(축) 번호(0 부터 시작합니다).
- ▶ **nMode** : STA 신호의 적용 여부를 결정합니다.

Value	Meaning	Description
0	Ignore STP	STP 입력 신호는 무시됩니다.
1	Immediate stop by STP	STP 입력핀에 Low level 신호가 입력되면 모션을 즉시 정지합니다
2	Stop after deceleration by STP	STP 입력핀에 Low level 신호가 입력되면 모션을 감속 후 정지합니다

### 참 고

- nMode 를 1 또는 2 로 하더라도 COMILX\_MC\_Stop()과 같은 소프트웨어적인 정지 명령은 여전히 유효합니다. 즉, STP 신호가 입력되지 않아도 소프트웨어적인 정지 명령으로 모션을 정지할 수 있습니다.
- STP 신호는 STA 신호와 달리 Level trigger 방식만 지원합니다. 그리고 입력 로직은 Low active(low level 일때 ON)입니다.
- STP 입력핀은 모든 축에 대해 공통으로 사용됩니다. 따라서 STP 입력을 각 축별로



따로 입력할 수는 없습니다. 그러나 STP 입력 환경은 각 축별로 다르게 설정할 수 있습니다.

## 사용예

□ 아래의 예제에서는 X,Y 축을 STA 입력 신호와 동기를 맞추어 동시에 시작하고 STP 입력 신호와 동기를 맞추어 동시에 종료되는 예입니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#define X_AXIS    0
#define Y_AXIS    1

void main()
{
    int nAxisList[2]={X_AXIS, Y_AXIS};
    int nDirList[2]={1,1};
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure
    // X,Y 축에 대하여 STA 신호의 환경을 설정한다. STA 입력에 Falling- //
    // edge 신호가 입력되면 이동이 시작되도록 한다. //
    COMILX_MC_SetMioCfgSTA(hDevice, X_AXIS, 1, TRG_EDGE);
    COMILX_MC_SetMioCfgSTA(hDevice, Y_AXIS, 1, TRG_EDGE);
    // X,Y 축에 대하여 STP 신호의 환경을 설정한다. STP 입력핀이 LOW LEVEL
    // 이 되면 모션이 종료됩니다. //
    COMILX_MC_SetMioCfgSTP(hDevice, X_AXIS, 2);
    COMILX_MC_SetMioCfgSTP(hDevice, Y_AXIS, 2);

    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);

    COMILX_MC_SetSpeedMode(hDevice, Y_AXIS, 1);
    COMILX_MC_SetSpeed(hDevice, Y_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, Y_AXIS, 20000, 20000);

    COMILX_MC_StartVMoveAll (hDevice, 2, nAxisList, nDirList);
    // STA 신호가 ON 이 되면 모션을 시작한다. 그리고 STP 신호가 ON 이 되면
    // 모션은 자동 종료됩니다. //
    while(!COMILX_MC_AllDone (hDevice, 2, nAxisList))
        ;

    COMILX_UnloadDevice(hDevice);
    COMILX_UnloadDll();
}
```

## Motion Control 라이브러리 업데이트 안내

---

}

## ■ COMILX\_MC\_CompleteArc

### 함수 원형

```
void COMILX_MC_CompleteArc (HANDLE hDevice, int nMapIndex)
```

### 함수 설명

이 함수는 COMILX\_MC\_StartArc\_a(), COMILX\_MC\_StartArc\_p(), COMILX\_MC\_StartArcTo\_a(), COMILX\_MC\_StartArcTo\_p() 함수를 이용하여 원호보간 작업을 수행하였을 때 Ending 좌표가 목표 좌표와 약간의 차이를 가질 수 있는데 이를 보정해주는 함수입니다. 따라서 앞에서 언급된 4 개의 함수를 이용하여 원호보간 작업을 수행했을 때 종점좌표가 목표좌표와 일치하지 않는 경우에는 원호보간이 완료된 후에 이 함수를 한번 수행해주면 해당 오차를 보정해줍니다.

이 함수는 COMI-LX501 과 COMI-LX504 제품을 사용하는 경우, 그리고 원호보간의 종점좌표가 불일치하는 경우에만 사용하십시오.

COMI-LX502, COMI-LX504A, COMI-LX508 제품을 사용하는 경우에는 이 함수를 사용하지 않아도 됩니다.

COMI-LX504A 제품과 COMI-LX504 제품을 구분하는 방법은 모션보드에 부착되어 있는 모션칩의 명칭을 보면 알 수 있습니다. COMI-LX504 는 모션칩에 PCL6045 로 표기되어 있으며, COMI-LX504A 는 PCL6045A 로 표기되어 있습니다.

### 매개 변수

▶ *hDevice* : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.

▶ *nMapIndex* : 축 그룹 인덱스. 이 값은 0 또는 1 이어야 하며 COMILX\_MC\_MapAxes 함수를 통하여 제어 대상축들이 맵핑되어 있어야 합니다.

### 사용예

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#define X_MASK    1
#define Y_MASK    2
#define Z_MASK    4
#define U_MASK    8

#define MAP0      0
```

## Motion Control 라이브러리 업데이트 안내

---

```
void main()
{
    double fDistList[2];
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure
    // Map X&Y axis to MAP0 //
    COMILX_MC_MapAxes(hDevice, MAP0, X_MASK|Y_MASK);
    // Set speed mode as Trapezoidal //
    COMILX_MC_SetSpeedModeMx(hDevice, MAP0, 1);
    // Set speed & accel => V=500, Acc=500 //
    COMILX_MC_SetSpeedMx(hDevice, MAP0, 500, 500);
    // 반원을 그리는 원호보간 시작 //
    COMILX_MC_StartArc_a(hDevice, MAP0, 0, 500, 180);
    // 원호보간이 완료될 때까지 기다린다. //
    while(!COMILX_MC_MxDone(hDevice, MAP0))
        Sleep(0);
    COMILX_MC_CompleteArc (hDevice, MAP0); // 원호보간을 완료한다.

    COMILX_UnloadDevice(hDevice);
    COMILX_UnloadDll();
}
```

## ■ COMILX\_MC\_SetListMotionAxes

### 함수 원형

```
void COMILX_MC_SetListMotionAxes(HANDLE hDevice, unsigned char bMapMask)
```

### 함수 설명

이 함수는 리스트모션에서 사용되는 모든 축들을 등록하는 함수입니다. 이는 리스트모션을 수행하면서 독립적으로 다른 축들을 구동하고자할 때 리스트모션 작업이 독립적으로 구동되는 축에 영향을 주지 않도록 하기 위함입니다.

예를 들어, X,Y 축을 리스트 모션을 이용하여 연속으로 여러 단계의 작업을 수행하면서 동시에 Z 축은 독립적으로 계속 구동되도록하고자 할 때 리스트 모션 작업의 영향으로 Z 축이 중간에 멈춰지는 현상이 벌어질 수 있습니다. COMILX\_MC\_BeginList()함수를 호출하기 전에 먼저 COMILX\_MC\_SetListMotionAxes(hDevice, 0x3)과 같이 리스트 모션에서는 X,Y 축만 영향을 주도록 하면 Z 축은 오류없이 리스트모션과 독립적으로 구동할 수 있습니다.

### 매개 변수

- ▶ **hDevice** : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- ▶ **nChannel** : 채널(축) 번호(0 부터 시작합니다).
- ▶ **bMapMask** : 리스트모션에서 사용되는 모든 축을 지정할 마스크 값. 이 값의 BIT0~BIT3 을 이용하여 그룹에 포함할 축들을 지정합니다. 각 비트의 값이 0 이면 해당 축(비트의 순서와 일치하는 축)은 배제되는 것이며 1 이면 해당 축이 포함되는 것입니다.

### 참 고

□ 이 함수는 리스트모션 기능을 보완하기 위하여 추가된 함수입니다. 리스트모션에 대한 자세한 설명은 인쇄된 매뉴얼의 “5.4.9 리스트모션(Listed Motion)” 단원을 참조하시기 바랍니다.

### 사용예

□ 아래 예제는 x,y 축에 대하여 연속적인 직선 및 원호 보간 작업을 리스트모션을 이용하여 연속적으로 수행하는 동시에 z 축은 독립적으로 다른 작업을 하도록 하는 예제입니다.

```
#include <windows.h>
#include <stdio.h>
```

## Motion Control 라이브러리 업데이트 안내

```
#include <conio.h>
#include "comidaslx.h"

// 축번호 //
#define X_AXIS 0
#define Y_AXIS 1
#define Z_AXIS 2
// 축마스크값 Axis Map 구성시 사용//
#define X_MASK 1
#define Y_MASK 2

void main()
{
    double fDistList[2];

    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_ServoOn(hDevice, X_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Y_AXIS, 1);
    COMILX_MC_ServoOn(hDevice, Z_AXIS, 1);

    COMILX_MC_SetSpeedMode(hDevice, Z_AXIS, 1);
    COMILX_MC_SetSpeed(hDevice, Z_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, Z_AXIS, 100000, 100000);

    // 리스트모션에서는 x,y 축만 구동하고 다른 축에는 영향을 주지 않도록 //
    // COMILX_MC_SetListMotionAxes()함수를 이용하여 x, y 축을 등록한다.
    COMILX_MC_SetListMotionAxes(hDevice, X_MASK|Y_MASK);
    COMILX_MC_BeginList(hDevice); // 모션 리스트 등록 시작 //
    COMILX_MC_MapAxes(hDevice, 0, X_MASK|Y_MASK); // Map X&Y axis //
    // Set Trapezoidal Speed Mode //
    COMILX_MC_SetSpeedModeMx(hDevice, 0, 1);
    // Set work speed=500, Acceleration=1000 //
    COMILX_MC_SetSpeedMx(hDevice, 0, 500, 1000);
    // (1000,0)만큼 직선 보간 이동 //
    fDistList[0]=1000; fDistList[1]=0;
    COMILX_MC_Line(hDevice, 0, fDistList);
    // 중심점 Offset = (0, 500), End Angle = 90 DEG //
    // 의 조건으로 원호보간 이동 //
    COMILX_MC_Arc_a(hDevice, 0, 0, 500, 90);
    // (0,1000)만큼 직선 보간 이동 //
    fDistList[0]=0; fDistList[1]=1000;
    COMILX_MC_Line(hDevice, 0, fDistList);
    COMILX_MC_EndList(hDevice); // 모션 리스트 등록을 마침 //
    // 리스트모션과 별개로 z 축을 구동한다. //
    COMILX_MC_StartVMove(hDevice, Z_AXIS, 1);
```

```
COMILX_MC_StartListMotion(hDevice); // 리스트 모션 수행 //
```

---

```
// 리스트 모션이 모두 완료될 때까지 기다림 //
```

```
while(!COMILX_MC_CheckListMotionDone(hDevice))
```

```
    ;
```

```
// 리스트모션과 별개로 동작되고 있는 z 축 작업을 종료한다 //
```

```
COMILX_MC_Stop(hDevice, Z_AXIS);
```

```
COMILX_UnloadDevice(hDevice);
```

```
COMILX_UnloadDll();
```

```
}
```

## 6. 라이브러리 V3.1 에서 추가된 기타 함수

이 단원에서는 라이브러리(DLL) V3.1.0.1 버전에서 추가된 함수들 중에서 앞에서 소개되지 않은 기타 함수들을 소개합니다.

### ▣ COMILX\_MC\_InitFromFile

#### 함수 원형

```
int COMILX_MC_InitFromFile (HANDLE hDevice, char *szFilePath)
```

#### 함수 설명

이 함수는 장치 초기화 파일로부터 정보를 전달받아서 모션제어보드를 초기화하는 함수입니다. 모션제어보드는 펄스 입출력 모드 및 각종 I/O 신호들의 환경 등, 초기화해주어야 할 사항들이 많습니다. 이전 버전의 라이브러리에서는 이러한 환경들을 설정해주기 위해서 각각의 관련함수들을 수행해주어야 했습니다. 그러나 라이브러리(DLL) V3.1.0.1 이후 버전에서는 이 함수를 이용하여 쉽게 장치초기화를 할 수 있습니다.

장치 초기화 파일은 ㈜커미조아의 모션제어 전용 프로그램인 “모션빌더”에서 생성됩니다. 사용자는 모션빌더의 “Device Init & Config” 메뉴를 통하여 장치의 환경을 설정할 수 있으며, 각 설정값을 파일로 저장할 수 있습니다. 단, 이 기능은 라이브러리(DLL) V3.1.0.1 이후 버전과 모션빌더 V1.1 이후 버전부터 지원됩니다. 라이브러리(DLL) V3.1.0.1 보다 낮은 버전의 라이브러리를 사용하거나, 모션빌더 V1.0 을 사용하시는 사용자께서 이 기능을 사용하시려면 프로그램을 업데이트하셔야 합니다.

#### 매개 변수

- ▶ **hDevice** : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.
- ▶ **szFilePath** : 장치 초기화 파일(.cme)의 경로를 포함한 파일명을 스트링으로 지정합니다.

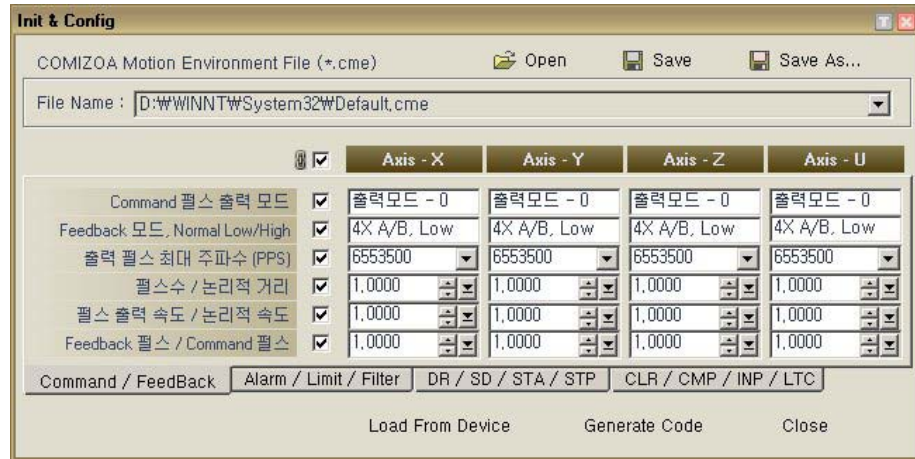
#### 반환값

Value	Meaning
0	초기화 파일을 이용한 장치 초기화에 성공
음수	초기화 파일을 로드하는데 에러가 발생함



## 참고

□ 모션빌더에서 “Device Init & Config” 메뉴를 선택하면 아래와 같은 화면이 나타납니다.



[그림 3-8] 모션빌더의 “Device Init & Config” 화면

사용자는 [그림 3-8] 화면에서 사용자의 시스템에 맞게 입출력 모드 및 각 신호의 환경을 설정한후 “Save” 버튼을 이용하여 저장하면 지정한 파일에 저장이 됩니다. 이렇게 저장된 파일을 **COMILX\_MC\_InitFromFile()** 함수를 이용하여 참조할 수 있습니다.

모션빌더는 윈도우의 시스템폴더에 “Default.cme”의 기본 장치 초기화 파일을 제공합니다. 사용자는 이 기본 파일을 사용하거나 “Save As” 버튼을 이용하여 다른 이름으로 초기화 파일을 저장할 수 있습니다.

□ **COMILX\_MC\_InitFromFile()** 함수를 이용하도록 제작된 사용자 프로그램을 배포할 경우에는 **COMILX\_MC\_InitFromFile()** 함수에서 참조하는 파일도 함께 배포하여야 합니다.

□ **COMILX\_MC\_InitFromFile()** 함수는 내부적으로 다음과 같은 함수들을 자동으로 수행해줍니다.

COMILX\_MC\_SetOutputMode(), COMILX\_MC\_SetInputMode(), COMILX\_MC\_SetSpeedRange(),  
COMILX\_MC\_SetUnitDistance(), COMILX\_MC\_SetUnitSpeed(),  
COMILX\_MC\_SetInOutRatio(), COMILX\_MC\_SetMioCfgALM(), COMILX\_MC\_SetSoftLimit(),  
COMILX\_MC\_EnableSoftLimit() 또는 COMILX\_MC\_DisableSoftLimit(),  
COMILX\_MC\_SetMioCfgEL(), COMILX\_MC\_SetELL(), COMILX\_MC\_SetFilterLogic(),

## Motion Control 라이브러리 업데이트 안내

COMILX\_MC\_SetMioCfgDR(), COMILX\_MC\_SetMioCfgSD(), COMILX\_MC\_SetMioCfgSTA(),  
COMILX\_MC\_SetMioCfgSTP(), COMILX\_MC\_SetMioCfgCLR(), COMILX\_MC\_SetMioCfgCMP(),  
COMILX\_MC\_SetMioCfgINP(), COMILX\_MC\_SetMioCfgLTC()

□ **COMILX\_MC\_InitFromFile()** 함수는 원점센서에 대한 환경설정은 수행하지 않습니다. 따라서 원점복귀 작업을 수행할 때에는 COMILX\_MC\_SetHomeConfig() 함수를 사용하여 원점센서에 대한 환경을 설정해야 합니다.

## 예 제

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#ifdef ON
#define ON 1
#endif

void main()
{
    int i;
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_Reset(hDevice);
    //////////////////////////////////////
    // CME 파일을 이용한 장치 초기화
    // 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
    // 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
    // 존재한다.
    char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
    GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
    패스를 얻는다.
    sprintf(szFilePath, "%s\\Default.cme", szSysDir);
    if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
        printf("장치 초기화 파일을 로드할 수 없습니다.");
    }
    // Servo 모터를 사용하는 경우에는 Servo-On을 시켜줍니다. 본 예제는 //
    // 4축 모터를 사용하는 예제이므로 4축에 대하여 Servo-On을 수행함 //
    // 니다. //
    for(i=0; i<4; i++){
        COMILX_MC_ServoOn(hDevice, i, ON);
    }
}
```

```
}  
// 이후 필요한 모션 작업을 수행합니다. //  
...  
...  
}
```

## ■ COMILX\_MC\_HomeMoveAuto

### 함수 원형

```
void COMILX_MC_HomeMoveAuto (HANDLE hDevice, int nChannel, int nDirection,  
double fRvsVel, double fEscapeDist)
```

### 함수 설명

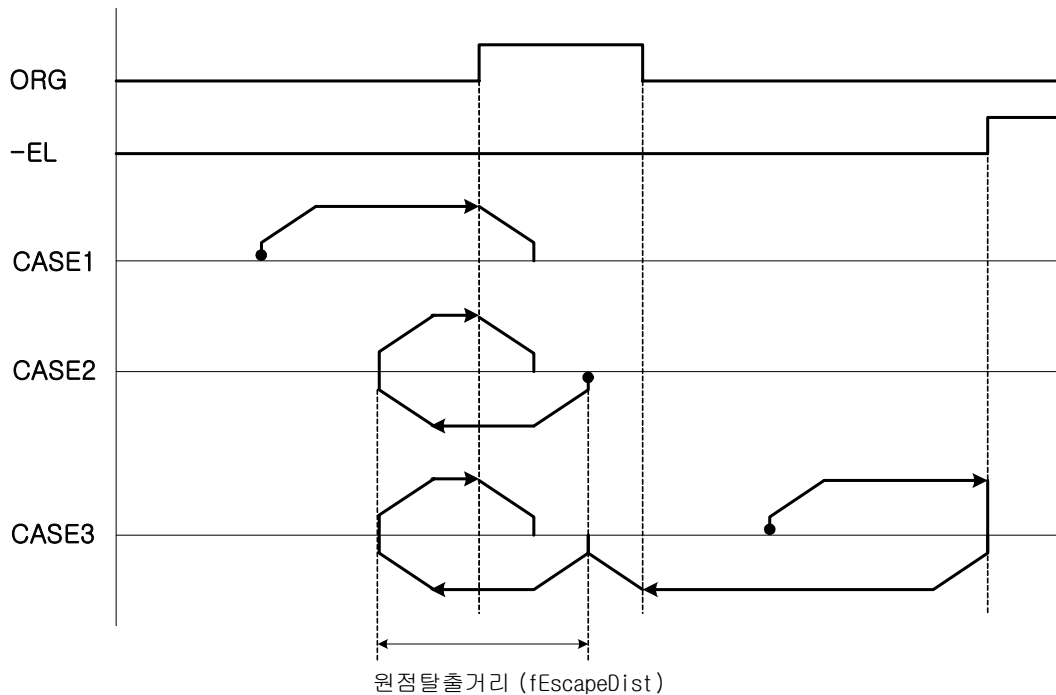
이 함수는 기구물이 어떤 위치에 있든 자동으로 원점을 찾아가도록 하는 기능이 보완된 원점복귀작업 명령입니다.

COMILX\_MC\_HomeMove() 함수를 사용할 때는 이미 원점센서가 ON 되어 있는 경우, 그리고 원점센서와 Negative Limit 센서 중간에 기구물이 위치해 있는 경우에는 사용자가 기구물을 원점센서를 기준으로 양의 방향으로 이동시킨 후 원점복귀작업을 수행해야 합니다.

COMILX\_MC\_HomeMoveAuto() 함수는 이러한 단점을 보완하여 기구물이 어느 위치에 있든 자동으로 원점을 찾아가도록 합니다. 기구물의 위치를 크게 3 가지의 경우로 나눌 수 있는데 각각의 경우에 대하여 COMILX\_MC\_HomeMoveAuto() 함수를 사용한 경우 동작되는 방식은 다음과 같습니다(단, 원점복귀 작업의 방향을 음의 방향(nDirection = -1)로 한 경우에 대한 것입니다).

- CASE 1. 원점센서를 기준으로 양의 방향 위치에서 원점복귀를 시작한 경우  
정상적인 원점복귀 작업을 수행합니다.
- CASE 2. 원점센서가 ON 인 상태에서 원점복귀를 시작한 경우  
이러한 경우에는 먼저 원점탈출거리(fEscapeDist) 만큼 양의 방향으로 이동한 후 다시 정상적인 원점복귀 작업을 수행합니다.
- CASE 3. 원점센서를 기준으로 양의 방향 위치에서 원점복귀를 시작한 경우  
이러한 경우에는 먼저 음의 방향으로 먼저 이동을 시작합니다. 그리고 -EL(Negative Limit) 센서가 ON 되면 정지 후 양의 방향으로 이동합니다. 원점센서가 ON 되면 다시 정지 후 CASE 2 에서와 같이 원점탈출거리(fEscapeDist) 만큼 양의 방향으로 이동한 후 다시 정상적인 원점복귀 작업을 수행합니다.

[그림 2-1]은 원점복귀모드를 0, 속도 패턴을 사다리꼴 방식으로 하고 COMILX\_MC\_HomeMoveAuto() 함수를 이용하여 원점복귀 작업을 수행할 때 위의 각 경우에 대하여 동작하는 방식을 도식적으로 나타낸 것입니다.



[그림 2-1] COMILX\_MC\_HomeMoveAuto() 함수를 이용한 원점복귀 작업

### 매개 변수

- ▶ ***hDevice*** : 디바이스 핸들값입니다. 이 값은 `COMILX_LoadDevice()` 함수에 의해 얻어진 값이어야 합니다.
- ▶ ***nChannel*** : 채널(축) 번호, 0 ~ 3
- ▶ ***nDirection*** : 원점 복귀 모션을 수행할 방향을 지정합니다.

Value	Meaning
0 또는 음수	(-) 방향
양수	(+) 방향

- ▶ ***fRvsVel*** : Reverse Speed 를 설정합니다. 복귀 모드에 따라 Reverse Speed 를 필요로 하는 모드가 있습니다. 앞의 복귀 모드 설명에서 Reverse Speed 는 `Vr` 로 표기되었습니다.
- ▶ ***fEscapeDist*** : 원점탈출거리를 지정합니다. 거리의 단위는 논리적 거리 단위를

사용합니다.

### 참 고

□ 이 함수는 원점 복귀 작업을 시작시킨 후에 바로 리턴(Return)합니다. 따라서 사용자는 COMILX\_MC\_Done() 함수를 사용하여 복귀 작업이 완료되었는지를 체크하여야 합니다.

### 예 제

본 예제는 x 축에 대하여 원점복귀 작업을 수행하는 예제입니다. 본 예제에서는 원점 복귀 모드 4 번을 설정하여 작업을 수행합니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#ifdef ON
#define ON 1
#endif

#define X_AXIS 0

void main()
{
    int i;
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_Reset (hDevice);

    //////////////////////////////////////
    // CME 파일을 이용한 장치 초기화
    // 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
    // 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
    // 존재한다.
    char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
    GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
    패스를 얻는다.
    sprintf(szFilePath, "%s\\Default.cme", szSysDir);
    if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
        printf("장치 초기화 파일을 로드할 수 없습니다.");
    }

    COMILX_MC_ServoOn(hDevice, X_AXIS, ON);
```

```
COMILX_MC_SetHomeConfig (hDevice, X_AXIS, MODEL, LOW_ACTIVE, 0,
LOW_ACTIVE, 0);

COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
COMILX_MC_SetSpeed(hDevice, X_AXIS, 200, 10000);
COMILX_MC_SetAccel(hDevice, X_AXIS, 100000, 100000);

COMILX_MC_HomeMoveAuto(hDevice, X_AXIS, 0, 1000, 2000);
while(!COMILX_MC_Done (hDevice, X_AXIS))
    ;

COMILX_UnloadDevice(hDevice);
COMILX_UnloadDll();
}
```

## ■ COMILX\_MC\_HomeMoveAutoAll

### 함수 원형

```
void COMILX_MC_HomeMoveAutoAll (HANDLE hDevice, int nNumAxis, int nAxisList[],
int nDirList[], double fRvsVelList[], double fEscapeDistList[])
```

### 함수 설명

이 함수는 여러축에 대하여 COMILX\_MC\_HomeMoveAuto() 작업을 동시에 시작하는 함수입니다. 동작 방식은 COMILX\_MC\_HomeMoveAuto() 함수와 동일합니다. 원점복귀의 자세한 내용은 COMILX\_MC\_HomeMove() 함수 및 COMILX\_MC\_HomeMoveAuto() 함수 설명편을 참조하시기 바랍니다.

### 매개 변수

- ▶ **hDevice** : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.
- ▶ **nNumAxis** : 동시에 작업을 수행할 대상 축의 수
- ▶ **nAxisList** : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.
- ▶ **nDirList** : 원점 복귀 모션을 수행할 방향을 지정하는 배열 주소값.

Value	Meaning
0 또는 음수	(-) 방향
양수	(+) 방향

- ▶ **fRvsVelList** : Reverse Speed 를 설정하는 배열 주소값.
- ▶ **fEscapeDistList** : 원점탈출거리를 지정하는 배열 주소값. 거리의 단위는 논리적 거리 단위를 사용합니다.

### 예 제

본 예제는 x 축에 대하여 원점복귀 작업을 수행하는 예제입니다. 본 예제에서는 원점 복귀 모드 4 번을 설정하여 작업을 수행합니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#ifdef ON
```



```

#define ON 1
#endif

#define X_AXIS    0

void main()
{
    int i;
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_Reset (hDevice);

    //////////////////////////////////////
    // CME 파일을 이용한 장치 초기화
    // 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
    // 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
    // 존재한다.
    char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
    GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
    패스를 얻는다.
    sprintf(szFilePath, "%s\\Default.cme", szSysDir);
    if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
        printf("장치 초기화 파일을 로드할 수 없습니다.");
    }

    for(I=0; I<4; I++){
        COMILX_MC_ServoOn(hDevice, i, ON); // 서보모터인 경우
        COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
        COMILX_MC_SetSpeed(hDevice, X_AXIS, 200, 10000);
        COMILX_MC_SetAccel(hDevice, X_AXIS, 100000, 100000);
        COMILX_MC_SetHomeConfig(hDevice, i, MODEL, LOW_ACITVE, 0,
        LOW_ACTIVE, 0);

    }
    int nAxes[4]={0,1,2,3};
    int nDirs[4] = {0,0,0,0};
    double fRvsVels[4]={1000, 1000, 1000, 1000};
    double fEscDists[4] = {1000, 1000, 1000, 1000};
    COMILX_MC_HomeMoveAutoAll(hDevice, 4, nAxes, nDirs, fRvsVels,
    fEscDists);
    while(!COMILX_MC_AllDone (hDevice, 4, nAxes))
        ;

    COMILX_UnloadDevice(hDevice);
    COMILX_UnloadDll();
}

```

## ■ COMILX\_MC\_LmCurSequence

### 함수 원형

```
int COMILX_MC_LmCurSequence (HANDLE hDevice)
```

### 함수 설명

이 함수는 리스트모션을 사용할 때 현재 수행되고 있는 작업을 알아보고자할 때 사용할 수 있도록 만들어진 함수입니다.

이 함수는 COMILX\_MC\_StartListMotion () 함수가 호출된 이후에 현재 수행되고 있는 작업의 인덱스를 반환합니다. 작업의 인덱스는 사용자가 리스트모션을 등록한 순서대로 부여됩니다. 단, COMILX\_MC\_SetSpeed(), COMILX\_MC\_SetAcce() 등과 같이 실제 이동을 수행하는 명령이 아닌 함수들은 인덱스 부여에서 무시됩니다.

### 매개 변수

▶ *hDevice* : 디바이스 핸들.

### 반환 값

현재 수행되고 있는 작업의 인덱스(0 - Based)를 반환합니다.

### 예 제

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#ifndef ON
#define ON 1
#endif

#define X_AXIS 0
#define X_MASK 1

void main()
{
    int i;
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX504, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    COMILX_MC_Reset (hDevice);
```

```

////////////////////////////////////
// CME 파일을 이용한 장치 초기화
// 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
// 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
// 존재한다.
char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
패스를 얻는다.
sprintf(szFilePath, "%s\\Default.cme", szSysDir);
if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
    printf("장치 초기화 파일을 로드할 수 없습니다.");
}

COMILX_MC_ServoOn(hDevice, X_AXIS, ON);

COMILX_MC_SetListMotionAxes(hDevice, X_MASK);
COMILX_MC_BeginList(hDevice); // 모션 리스트 등록 시작 //
// Set Trapezoidal Speed Mode //
COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1); // 인덱스부여 무시
// Move (1) //
COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 1000); // 인덱스부여 무시
COMILX_MC_SetAccel(hDevice, X_AXIS, 2000, 0); // 인덱스부여 무시
COMILX_MC_MoveTo(hDevice, X_AXIS, 3000); // Operation[0]으로 등
록됨
// Move (2) //
COMILX_MC_SetSpeed(hDevice, X_AXIS, 1000, 3000); // 인덱스부여
무시
COMILX_MC_SetAccel(hDevice, X_AXIS, 2000, 2000); // 인덱스부여
무시
COMILX_MC_MoveTo(hDevice, X_AXIS, 8000); // Operation[1]로 등록
됨
// Move (3) //
COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 1000); // 인덱스부여 무시
COMILX_MC_SetAccel(hDevice, X_AXIS, 0, 2000); // 인덱스부여 무시
COMILX_MC_MoveTo(hDevice, X_AXIS, 11000); // Operation[2]로 등
록됨
COMILX_MC_EndList(hDevice); // 모션 리스트 등록을 마침 //

COMILX_MC_StartListMotion(hDevice); // 리스트 모션 수행 //
// 리스트 모션이 모두 완료될 때까지 기다림 //
while(!COMILX_MC_ChekcListMotionDone(hDevice)){
    // 현재 수행되는 작업 번호를 체크하여 화면에 표시 //
    nCurOper = COMILX_MC_LmCurSequence(hDevice);
    printf("Current Operation = %d\n", nCurOper);
    Sleep(0);
}

```

## Motion Control 라이브러리 업데이트 안내

---

```
COMILX_UnloadDevice(hDevice);  
COMILX_UnloadDll();  
}
```

## ■ COMILX\_MC\_SetSpeedMx2

### 함수 원형

```
void COMILX_MC_SetSpeedMx2 (HANDLE hDevice, int nMapIndex, double fSpeed,
double fAccel, double fDecel)
```

### 함수 설명

Coordinated Motion 의 속도 및 가/감속도를 설정합니다. COMILX\_MC\_SetSpeedMx() 함수가 가속값과 감속값을 동일하게 설정해야만하는것과 달리 이 함수를 사용하면 Coordinated Motion 에서도 가속값과 감속값을 다르게 설정할 수 있습니다.

Coordinated Motion 의 속도 설정에 대한 자세한 사항은 COMILX\_MC\_SetSpeedMx() 함수 설명편을 참조하시기 바랍니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들.
- ▶ *nMapIndex* : 축 그룹 인덱스, 이 값은 0 또는 1 이어야 합니다.
- ▶ *fSpeed* : 작업속도를 벡터 속도값으로 지정합니다. 벡터 속도에 대한 자세한 내용은 “참고” 항목을 참조하십시오.
- ▶ *fAccel* : Coordinated Motion 의 가속도를 지정합니다. 단, 이 값을 0 으로 설정하면 가속구간이 없이 즉시 작업속도로 올라갑니다(실제로는 가속값이 무한대).
- ▶ *fDecel* : Coordinated Motion 의 감속도를 지정합니다. 단, 이 값을 0 으로 설정하면 감속구간이 없이 즉시 정지합니다(실제로는 감속값이 무한대).

### 참 고

□ Coordinated Motion 의 속도모드를 S-curve 속도모드로 설정하면 가/감속 구간에서 Linear Section 이 없는 완전한 S-curve 가/감속을 수행하게 됩니다.

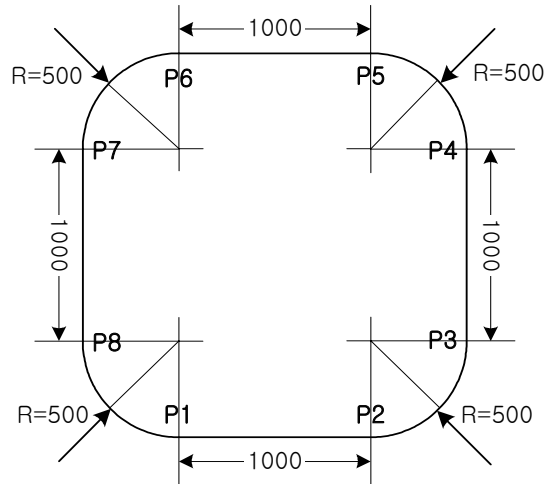
□ 이 함수는 여러 단계의 Coordinated Motion 작업을 중지없이 연속적으로 수행하고자할 때 유용하게 사용될 수 있는데, 이러한 경우에는 리스트모션과 함께 사용하는 것이 효과적입니다.

### 예 제

본 예제는 아래 그림과 같이 직선보간 이동과 원호보간 이동을 조합하는 Coordinated Motion 을 수행하는 예제입니다. p1 점으로부터 출발하여 p8 점을 거쳐 다시 p1 으로 복귀하는 작업입니다.  
이 때 각 포인트에서 중지하지 않고 연속적으로 다음 포인트로 이동하기 위해서 리

## Motion Control 라이브러리 업데이트 안내

스트모션을 사용하며, 처음 이동(P1→P2)에서는 감속없이 가속만 수행하고 P2에서 P8로 이동하는동안에는 가/감속이 없이 동작하게 합니다. 그리고 마지막 이동(P8→P1)에서는 가속없이 감속만 수행하게 합니다.



```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#define X_AXIS    0
#define Y_AXIS    2

#define X_MASK    1
#define Y_MASK    2

#define MAP0      0

void main()
{
    double fDistList[2];
    if(!COMILX_LoadDll())
        exit(-1); // Load Dll Failure

    HANDLE hDevice = COMILX_LoadDevice(COMI_LX501, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    //////////////////////////////////////
    // CME 파일을 이용한 장치 초기화
    // 본 예제에서는 모션빌더 프로그램이 기본적으로 사용하는 Default.cme
    // 파일을 사용하는 것으로 하였다. 이 파일은 윈도우의 시스템 폴더에
```

```

// 존재한다.
char szSysDir[MAX_PATH], szFilePath[MAX_PATH];
GetSystemDirectory(szSysDir, MAX_PATH); // 윈도우즈의 시스템 폴더
패스를 얻는다.
sprintf(szFilePath, "%s\\Default.cme", szSysDir);
if(COMILX_MC_InitFromFile(hDevice, szFilePath) < 0){
    printf("장치 초기화 파일을 로드할 수 없습니다.");
}

COMILX_MC_ServoOn(hDevice, X_AXIS, ON);
COMILX_MC_ServoOn(hDevice, Y_AXIS, ON);

COMILX_MC_SetListMotionAxes(hDevice, X_MASK|Y_MASK);
COMILX_MC_BeginList(hDevice); // 모션 리스트 등록 시작 //

// Map X&Y axis to MAP0 //
COMILX_MC_MapAxes(hDevice, MAP0, X_MASK|Y_MASK);
// Set speed mode as Trapezoidal //
COMILX_MC_SetSpeedModeMx(hDevice, MAP0, 1);
// Set speed & accel => V=5000, Acc=25000, Dec=0(무한대) //
COMILX_MC_SetSpeedMx2(hDevice, MAP0, 5000, 25000, 0);
// Move from P1 to P2 //
fDistList[0]=1000; fDistList[1]=0;
COMILX_MC_Line(hDevice, MAP0, fDistList);
// Move from P2 to P3 //
COMILX_MC_SetSpeedModeMx(hDevice, MAP0, 0); // Constant 속도모드
COMILX_MC_Arc_a(hDevice, MAP0, 0, 500, 90);
// Move from P3 to P4 //
fDistList[0]=0; fDistList[1]=1000;
COMILX_MC_Line(hDevice, MAP0, fDistList);
// Move from P4 to P5 //
COMILX_MC_Arc_a(hDevice, MAP0, -500, 0, 90);
// Move from P5 to P6 //
fDistList[0]=-1000; fDistList[1]=0;
COMILX_MC_Line(hDevice, MAP0, fDistList);
// Move from P6 to P7 //
COMILX_MC_Arc_a(hDevice, MAP0, 0, -500, 90);
// Move from P7 to P8 //
fDistList[0]=0; fDistList[1]=-1000;
COMILX_MC_Line(hDevice, MAP0, fDistList);
// Move from P8 to P1 //
COMILX_MC_SetSpeedModeMx(hDevice, MAP0, 1); //Trapezoidal 속도모드

// Set speed & accel => V=5000, Acc=0(무한대), Dec=25000 //
COMILX_MC_SetSpeedMx2(hDevice, MAP0, 5000, 0, 25000);
COMILX_MC_Arc_a(hDevice, MAP0, 500, 0, 90);

COMILX_MC_EndList(hDevice); // 모션 리스트 등록을 마침 //

```

## Motion Control 라이브러리 업데이트 안내

---

```
COMILX_MC_StartListMotion(hDevice); // 리스트 모션 수행 //
```

// 리스트 모션이 모두 완료될 때까지 기다림 //

```
while(!COMILX_MC_ChekcListMotionDone(hDevice)){
    // 현재 수행되는 작업 번호를 체크하여 화면에 표시 //
```

nCurOper = COMILX\_MC\_LmCurSequence(hDevice);

```
    printf("Current Operation = %d\n", nCurOper);
    Sleep(0);
}
```

COMILX\_UnloadDevice(hDevice);

```
COMILX_UnloadDll();
}
```



## ■ COMILX\_EnableDebugLog

### 함수 원형

```
int COMILX_EnableDebugLog (HANDLE hDevice, char *szLogFile, int nDebugLevel)
```

### 함수 설명

이 함수는 라이브러리 디버그 로그 기능을 Enable 시키는 함수입니다. nDebugLevel 값을 1,2,3 중의 하나의 값으로 설정하면 라이브러리 디버그 로그 기능이 Enable 됩니다. nDebugLevel 값을 0으로 설정하면 로그 기능이 Disable 됩니다.

로그 기능이 Enable 되면 모션 라이브러리의 함수가 호출될 때마다 그 시각과 호출된 내용이 지정한 로그 파일에 기록됩니다. 이때 기록 대상 라이브러리 함수들은 지정한 레벨에 따라서 달라집니다.

프로그램이 비정상적으로 동작하는 경우에 로그 파일이 유용하게 사용될 수 있습니다. 그러나 로그 기능을 사용하면 프로그램의 수행속도에 영향을 미칠 수 있으므로 정상적으로 동작하는 경우에는 Disable 시키는 것이 바람직합니다.

### 매개 변수

- ▶ **hDevice** : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.
- ▶ **szLogFile** : 로그 파일의 경로를 포함한 파일명을 스트링으로 지정합니다.
- ▶ **nDebugLevel** : 디버그 레벨을 설정합니다. 이 레벨에 따라서 로그되는 대상 함수들이 결정됩니다.

Value	Meaning
0	라이브러리 디버그 로그 기능을 Disable 시킵니다.
1	모든 COMILX_MC_SetXXX 함수와 이송 명령을 로깅합니다.
2	Level1 에서 로깅되는 함수와 더불어 COMILX_MC_GetXXX 함수들도 로깅합니다. 단, 반복적으로 사용될 수 있는 다음과 같은 함수들은 제외됩니다. COMILX_MC_GetPosition_#, COMILX_MC_GetCount_#, COMILX_MC_GetActualSpeed, COMILX_MC_GetAxisIntState, COMILX_MC_GetIntStatus, COMILX_MC_LmCurSequence, COMILX_MC_Done, COMILX_MC_AllDone, COMILX_MC_MxDone,

## Motion Control 라이브러리 업데이트 안내

---

	COMILX_MC_CheckListMotionDone,
3	모든 라이브러리 함수가 로딩됩니다.

### 반환값

Value	Meaning
0	함수 수행 성공
음수	함수 수행 실패

## ■ COMILX\_MC\_SetOutputMask

### 함수 원형

```
void COMILX_MC_SetOutputMask (HANDLE hDevice, int nChannel, BOOL bMask)
```

### 함수 설명

이 함수는 펄스 출력을 마스크하는 기능을 제공합니다. 펄스 출력을 마스크하게 되면 논리적으로는 모든 명령이 정상적으로 동작하지만 실제 펄스는 출력되지 않도록 하는 것을 의미합니다. 이는 안전을 위하여 제어 대상을 실제로 동작 시키지 않으면서 프로그램만 정상적으로 동작시켜볼 수 있도록 하기 위해서 제공되는 기능입니다.

### 매개 변수

- ▶ *hDevice* : 디바이스 핸들값입니다. 이 값은 COMILX\_LoadDevice() 함수에 의해 얻어진 값이어야 합니다.
- ▶ *nChannel* : 마스크를 적용할 축번호
- ▶ *bMask* : 펄스 출력 마스크 여부

Value	Meaning
0	출력 펄스를 마스크하지 않습니다. 따라서 정상적으로 펄스가 출력되도록 합니다.
1	출력 펄스를 마스크합니다. 따라서 논리적인 동작은 정상적으로 동작되지만 실제로 펄스는 출력되지 않습니다.

## Motion Control 라이브러리 업데이트 안내

---

---

## PART III. 알아두기

본 단원에서는 ㈜커미조아의 모션제어보드를 사용하면서 사용자들께서 주의하셔야 할 사항을 수록하고, 인쇄매뉴얼의 내용만으로는 이해하기 어려운 부분들을 보충설명하고 있습니다.

## ♣. 8 축 모션보드에서의 보간구동

8 축 모션보드는 Axis 0 ~ 3 의 4 개의 축이 **축그룹 1**로 구분되고, Axis 4 ~ 7 의 4 개의 축이 **축그룹 2**로 구분됩니다. 동일 축그룹내에서의 보간 구동은 제약없이 사용할 수 있습니다. 그러나 서로 다른 축그룹에 속한 축들간의 보간 구동은 허용되지 않거나 제약이 있습니다. 아래의 내용을 참조하시기 바랍니다.

### 8 축 모션보드에서의 직선보간

동일축그룹에 포함되는 축들간의 직선보간은 속도모드와 관계없이 사용할 수 있습니다. Axis 4 ~ 7 간의 직선보간을 사용하는 경우에는 다음의 예와 같이 사용하시면 됩니다.

```
#define CH4_MASK 0x10 // Axis4의 축마스크값
#define CH5_MASK 0x20 // Axis5의 축마스크값
#define CH6_MASK 0x40 // Axis6의 축마스크값
#define CH7_MASK 0x80 // Axis7의 축마스크값

#define MAP0 0
// MAP0_MASK는 아래와 같이 OR 연산을 통해서 설정해도 되고 0xf0 값을 사용해도 된다. //
#define MAP0_MASK (CH4_MASK|CH5_MASK|CH6_MASK|CH7_MASK)

double fDists[4];
COMILX_MC_MapAxes(hDevice, MAP0, MAP0_MASK);
COMILX_MC_SetSpeedModeMx(hDevice, MAP0, 1);
COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 50000);
fDists[0]=3000; fDists[1]=4000;
fDists[2]=5000; fDists[3]=6000;
COMILX_MC_Line(hDevice, MAP0, fDists);
fDists[0]=-3000; fDists[1]=-4000;
fDists[2]=-5000; fDists[3]=-6000;
COMILX_MC_Line(hDevice, MAP0, fDists);
```

그러나 서로다른 축그룹에 포함된 축들간의 직선보간구동을 사용하는 경우에는 속도모드를 **Constant speed mode**로 설정하여야 합니다. 예를 들어 아래와같이 Axis 0, 1, 4, 5 의 4 개의 축을 직선보간구동하는 경우에는 아래와 같이 속도모드를 Constant speed mode로 설정하여야 합니다.

```
#define CH0_MASK 0x01 // Axis4의 축마스크값
#define CH1_MASK 0x02 // Axis5의 축마스크값
#define CH4_MASK 0x10 // Axis6의 축마스크값
#define CH5_MASK 0x20 // Axis7의 축마스크값

#define MAP0 0
```

```
// MAP0_MASK 는 아래와 같이 OR 연산을 통해서 설정해도 되고 0x33 값을 사용해도 된다. //
#define MAP0_MASK (CH0_MASK|CH1_MASK|CH4_MASK|CH5_MASK)
#define SM_CONSTANT 0 // Constant speed mode

double fDists[4];
COMILX_MC_MapAxes(hDevice, MAP0, MAP0_MASK);
COMILX_MC_SetSpeedModeMx(hDevice, MAP0, SM_CONSTANT);
COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 50000);
fDists[0]=3000; fDists[1]=4000;
fDists[2]=5000; fDists[3]=6000;
COMILX_MC_Line(hDevice, MAP0, fDists);
fDists[0]=-3000; fDists[1]=-4000;
fDists[2]=-5000; fDists[3]=-6000;
COMILX_MC_Line(hDevice, MAP0, fDists);
```

## 8 축 모션보드에서의 원호보간 및 헬리컬보간

서로 다른 축그룹에 포함되어 있는 축들간의 원호보간 및 헬리컬보간은 허용되지 않습니다. 예를 들어 Axis 0 과 Axis 4 의 두 축간의 원호보간은 오동작하게 됩니다.

## ♣. 8 축 모션보드에서의 다축 동기구동

보간구동과는 달리 다축 동기구동시에는 서로 다른 축그룹에 속한 채널들을 함께 사용하는데 있어서 제약사항이 없습니다. 아래의 예는 8 개의 축을 동시에 구동하는 예입니다.

```
int nAxisList[8]={0,1,2,3,4,5,6,7};
double fDistList[8]={10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000};

for(int i=0; i<8; i++){
    COMILX_MC_SetSpeedMode(hDevice, i, 1);
    COMILX_MC_SetSpeed(hDevice, i, 0, 10000);
    COMILX_MC_SetAccel(hDevice, i, 100000, 100000);
}
COMILX_MC_MoveAll (hDevice, 8, nAxisList, fDistList);
```

## ♣. 매뉴얼 펄서(Manual Pulser) 구동시 주의할 점

□ COMILX\_MC\_StartPulserMove() 또는 COMILX\_MC\_StartPulserVMove () 함수가 실행된 상태에서 COMILX\_MC\_EmgStop() 함수 수행없이 다시 COMILX\_MC\_StartPulserMove()

또는 COMILX\_MC\_StartPulserVMove () 함수가 호출되면 오동작할 수 있습니다. COMILX\_MC\_StartPulserMove() 또는 COMILX\_MC\_StartPulserVMove() 함수를 수행할 때에는 COMILX\_MC\_EngStop()을 호출해준 후에 수행하는 것이 바람직합니다.

□ PA/PB 에 입력되는 펄스의 주파수가 COMILX\_MC\_SetSpeed()함수를 이용하여 설정한 속도보다 높게 되면 오버플로우가 발생하여 오동작하게 됩니다. 매뉴얼 펄스 동작모드에서 COMILX\_MC\_SetSpeed() 함수는 펄서입력의 최대속도를 제한하는 역할을 합니다.

## ♣. 장치초기화 간단히 하기

모션제어보드는 사용자가 프로그램을 개발할 때 펄스 입출력 모드 및 각종 I/O 신호들의 환경 등, 초기화해주어야 할 사항들이 많습니다. 라이브러리(DLL) V3.0.0.1 이전 버전의 라이브러리에서는 이러한 환경들을 설정해주기 위해서 각각의 관련함수들을 수행해주어야 했습니다. 그러나 라이브러리(DLL) V3.1.0.1 이후 버전에서는 COMILX\_MC\_InitFromFile() 함수를 이용하여 쉽게 장치초기화를 할 수 있습니다. 이에 대한 자세한 사항은 본 문서의 74 페이지, COMILX\_MC\_InitFromFile 함수 설명편을 참조하시기 바랍니다.

## ♣. 논리적거리가 적용되지 않는 예외 경우

커미조아의 모션제어보드를 사용할 때에는 COMILX\_MC\_SetUnitDistance() 함수를 사용하여 논리적거리 단위를 펄스 단위가 아닌 물리적 단위로 설정할 수 있습니다. 모든 이동 명령은 논리적거리 단위로 위치 데이터를 지정하게 됩니다. 그러나 위치비교출력의 위치데이터는 반드시 펄스 단위로 지정해야 합니다. 위치비교출력과 관련된 함수들은 본 문서의 26~38 페이지에 수록되어 있습니다.

## ♣. 디지털입력 채널 신호 연결법

커미조아의 모션제어보드에서는 범용 디지털 입력 채널을 제공하며 포토커플러로 절연되어 있습니다. 범용 디지털 입력 회로는 COMI-LX501 제품과 COMI-LX502/4/8 제품이 약간 다르게 설계되어 있습니다. 제품에 따라서 아래 그림을 참조하여 연결하시기 바랍니다.

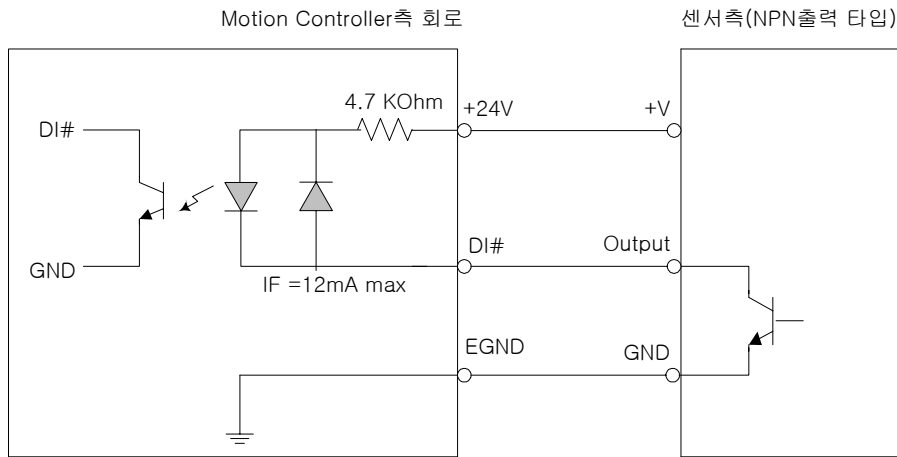


### COMI-LX501 보드에서의 디지털입력 신호 연결법

COMI-LX501 은 DI-COM 단자가 따로 없으며 터미널단자에 입력되는 +24V 전원을 DI-COM 으로 사용합니다. 따라서 디지털입력 센서에 공급되는 전원은 터미널보드에 사용되는 +24V 전원과 같은 것을 사용해야 합니다. 만일 센서에 입력되는 전원을 터미널보드와 인가되는 전원과 다른 것을 사용하고자 한다면 두 전원의 GND 를 연결해주어야 합니다.

COMI-LX501 에서 디지털입력으로 연결되는 신호는 Current sink 타입이어야 합니다. 따라서 오픈콜렉터 출력의 센서를 사용하는 경우에는 NPN 출력 형식을 사용하여야 하며 PNP 출력 형식의 센서는 사용할 수 없습니다.

다음 그림은 COMI-LX501 에서 포토센서출력(NPN 타입)을 디지털입력으로 사용할 때의 연결법입니다.

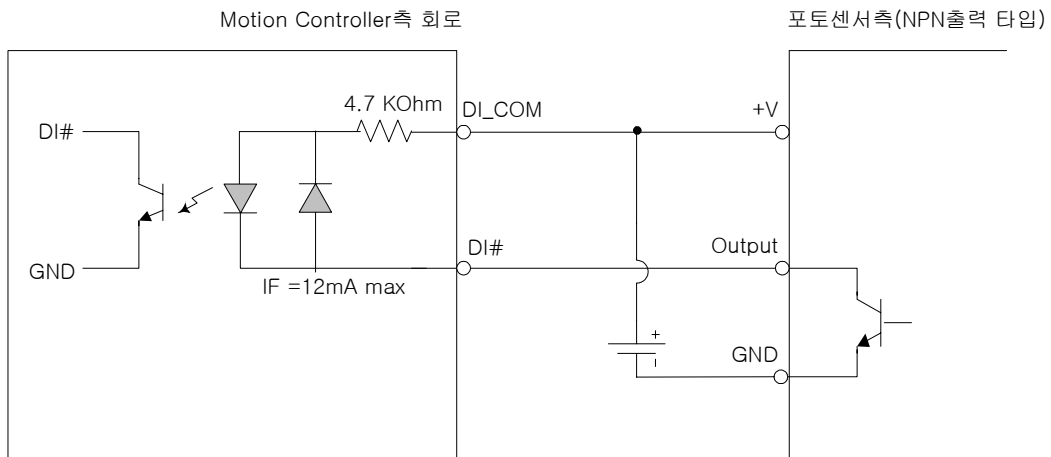


[그림 3-1] COMI-LX501 디지털 입력 신호연결 예 (NPN 포토센서)

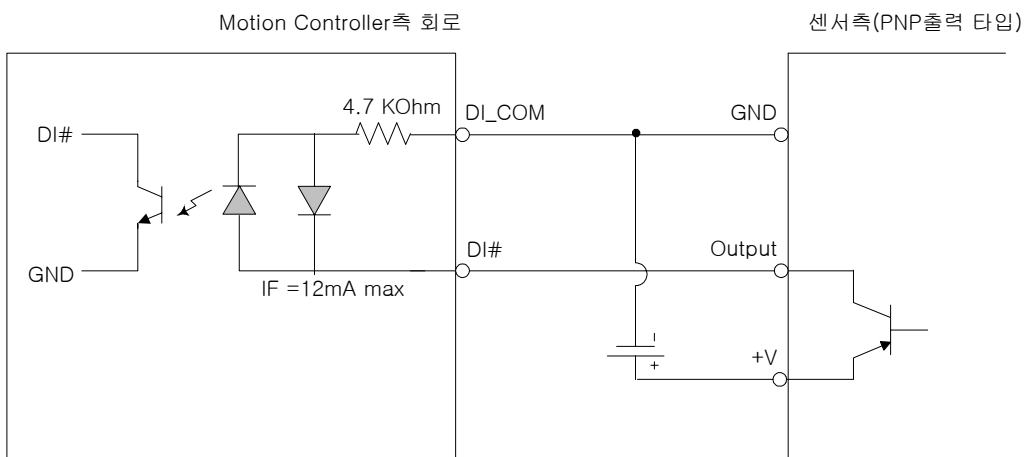
### COMI-LX502/4/8 보드에서의 디지털입력 신호 연결법

COMI-LX502/4/8 은 DI-COM 단자가 제공됩니다. DI-COM 단자에는 센서쪽에 인가되는 +전원을 연결하면 됩니다. COMI-LX502/4/8 에서는 센서와 터미널보드에 공급되는 전원이 서로 달라도 무방합니다. COMI-LX502/4/8 에서는 Current sink 와 Current source 형식을 모두 연결할 수 있습니다.

## 알아두기



[그림 3-2] COMI-LX502/4/8 디지털 입력 신호연결 예 1 (NPN 포토센서)

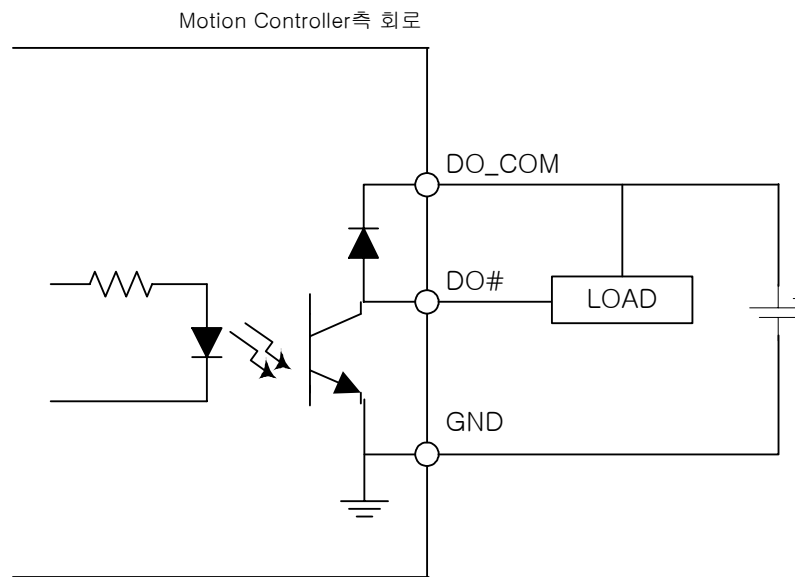


[그림 3-3] COMI-LX502/4/8 디지털 입력 신호연결 예 2 (PNP 포토센서)

## ♣. 디지털출력 채널 신호 연결법

커미조아의 모션제어보드에서는 범용 디지털 출력 채널을 제공합니다. 디지털 출력 회로 및 신호 연결 방식은 [그림 3-4]과 같습니다. 그림에서와 같이 디지털 출력 신호 연결 방식은 “Common ground 연결” 방식을 사용합니다. 이 회로에서는 디지털

출력이 “ON” 상태가 되면 싱크전류(Sink current)가 트랜지스터를 통하여 전도되게 됩니다. 그리고 디지털 출력이 “OFF” 상태가 되면 트랜지스터를 통하여 전류가 흐르지 않게 됩니다. 주의할 것은 릴레이, 코일 또는 모터등과 같이 인덕턴스(Inductance) 성질을 가지는 부하(Load)를 구동할 때에는 외부 소스 전원을 DO\_COM 핀에도 연결해주어야 합니다. 이 것은 “Fly-wheel Diode” 를 사용하여 부하가 “ON” 에서 “OFF” 상태로 변할 때 발생하는 역기전압으로부터 트랜지스터를 보호하기 위함입니다.



[그림 3-4] COM1-LX502/4/8 디지털 출력 신호연결법

## ♣. CMP 출력신호 연결법 및 주의사항

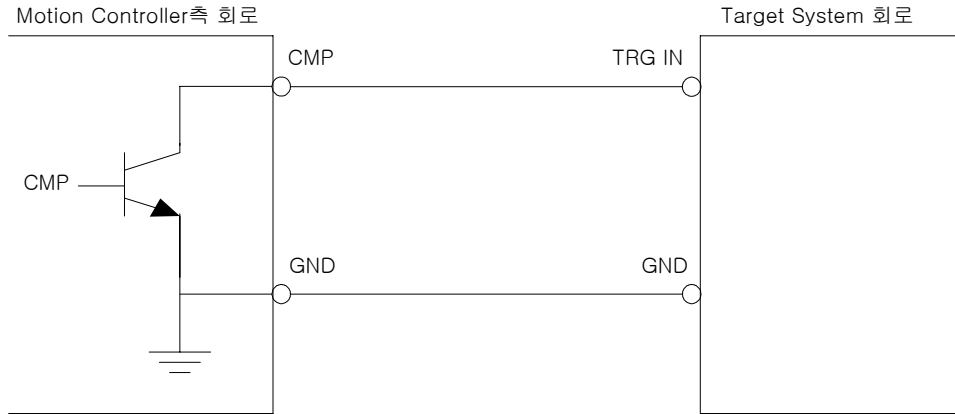
□ CMP 출력 신호를 트리거 신호로 사용할 때 Low Active(Falling Edge 감지) 방식으로 사용하는 경우와 High Active(Rising Edge 감지) 방식으로 사용하는 각각의 경우에 신호의 지연시간은 아래와 같습니다(Low Active 방식을 사용하는 것이 바람직함).

- Low Active 방식 :  $\leq 2 \mu s$
- High Active 방식 :  $\leq 40 \mu s$

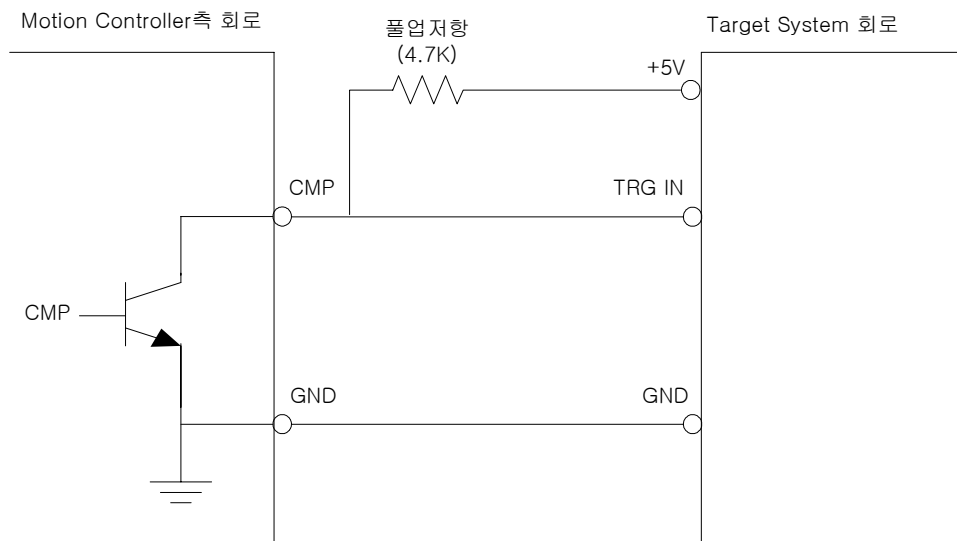
□ CMP 출력 신호는 NPN 오픈콜렉터 출력입니다. Target System 의 신호입력형식이 Isolated Input 형식이면 [그림 3-5]과 같이 연결하시고, TTL Input 형식이면

## 알아두기

[그림 3-6]과 같이 Pull-up 을 해주어야 합니다.



[그림 3-5] Isolated Input 형식일때의 CMP 신호 연결도



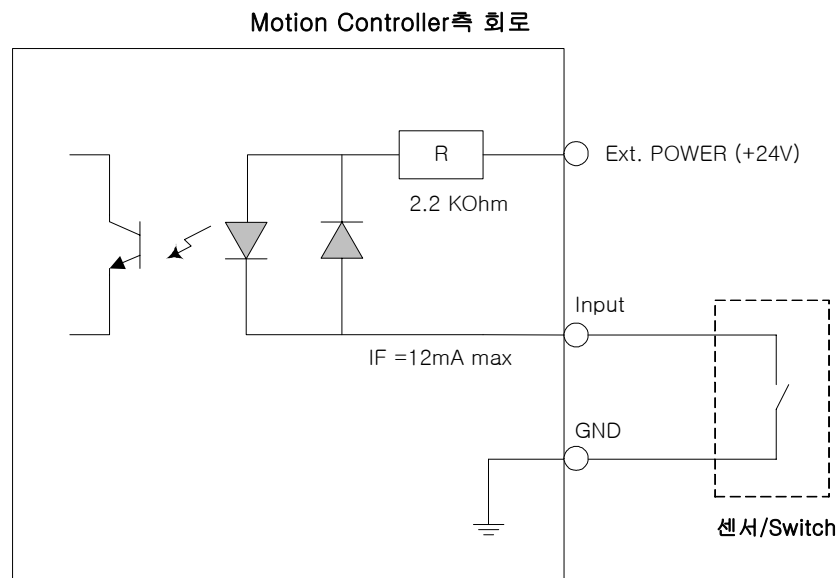
[그림 3-6] TTL Input 형식일때의 CMP 신호 연결도

## ♣. I/O 신호 입력로직과 센서형식

커미조아의 모션제어보드에서는 펄스입출력 신호외에 많은 I/O 신호를 제공합니다. COMI-LX50x 모션보드는 각 입력 신호의 ON/OFF 상태를 센서와 일치시키기 위해서 입력로직을 사용자가 설정할 수 있도록하는 인터페이스를 제공하며 입력로직은 **Low-active (0)** 또는 **High-active (1)**로 설정하도록 되어 있습니다.

COMI-LX50x 모션보드의 입력 신호는 모두 [그림 3-7]과 같이 센서에 의해서 GND 와 OPEN 되거나 CLOSE 되는 스위치입력 형식으로 구성되어 있습니다. 각 입력신호의 입력로직 설정에 따라 스위치가 Open 상태일때 입력 State 가 ON 이 될수도 있고 OFF 가 될수도 있습니다. 따라서 센서의 형식(A 접점, B 접점)에 따라 입력로직을 적절히 설정해주어야 합니다.

각 입력 신호의 로직과 센서의 상태에 따른 입력의 ON/OFF 관계는 [표 3-1]과 같습니다.



	LOW ACTIVE 설정시		HIGH ACTIVE 설정시		비고
	OPEN	CLOSE	OPEN	CLOSE	
ALM	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A *. 일반적으로 서보의 Alarm출력은 B접점(Normal Close)형식이 많습니다.
EL	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A *. B접점센서를 사용하는것이 바람직합니다.
ORG	ON	OFF	OFF	ON	*. A접점센서 - H/A, B접점센서 - L/A
CLR	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A
DR	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A
LTC	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A
PCS	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A
SD	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A
STA	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A
STP	OFF	ON	ON	OFF	*. A접점센서 - L/A, B접점센서 - H/A

[표 3-1] 입력신호의 로직과 센서상태에 따른 ON/OFF 관계

#### □ 용어 설명

- L/A : Low-active, 라이브러리 함수에서는 0 값을 지정
- H/A : High-active, 라이브러리 함수에서는 1 값을 지정
- A 접점센서 : 정상(Normal)시 Open 상태이고, 감지되면 Close 상태가 되는 센서
- B 접점센서 : 정상(Normal)시 Close 상태이고, 감지되면 Open 상태가 되는 센서

□ ORG 신호만 입력로직이 반대이고 나머지 I/O 신호들은 모두 입력로직과 센서상태에 따른 ON/OFF 관계가 동일합니다.

### ♣. 5V 출력핀 사용시 주의할 점

COMI-LX50x 모션컨트롤러는 5V 전압 출력단자를 제공합니다. 이 출력단자를 외부기에 연결하여 사용하고자 할때는 200mA 이상의 전류가 부하되지 않도록 주의하시기 바랍니다.

---

## PART IV. Trouble Shooting 및 FAQ

본 단원에서는 ㈜커미조아의 모션제어보드를 사용하면서 자주 질문되는 내용이나 문제 발생시 대처해야하는 사항들에 대해서 수록하였습니다. 문제 발생시에 본 단원의 내용을 참조하여 해결하시기 바랍니다. 만일 본 단원에 수록되지 않은 문제가 발생되거나 본 단원의 내용만으로 해결되지 않는 경우에는 ㈜커미조아의 기술지원부에 문의하시기 바랍니다.

## ♣. 이동명령을 내렸는데 모터가 구동되지 않습니다.

이동명령을 내렸는데 모터가 구동되지 않는 경우는 여러가지 원인이 있습니다. 아래와 같이 증상을 살핀후에 대처를 하시기 바랍니다.

### Command position 값이 바뀌지 않는 경우

☐ Alarm 이나 Limit 입력이 ON 이 되어있지 않는지 확인하십시오. Alarm 이나 Limit 입력이 ON 이 되면 모션은 구동되지 않도록 설계되어 있습니다.

☐ Software Limit 에 의해서 이러한 현상이 발생할 수도 있습니다. Software Limit 설정이 올바른지 확인하십시오.

### Command position 값이 바뀌는데 모터가 구동되지 않는 경우

☐ Motion Controller 에 24V 전원이 공급되는지 확인하십시오. 24V 전원이 공급되지 않으면 모터를 구동할 수 없습니다.

☐ 펄스출력방식이 모터드라이버와 일치하는지 확인하십시오. COMI-LX50x 는 Line-driver 출력 방식과 Open-collector 출력 방식의 두가지 출력방식을 제공하며 이는 보드에 있는 점퍼로 설정할 수 있습니다(인쇄매뉴얼의 80 페이지 참조).

☐ 서보모터를 사용하는 경우 SERV0-ON 을 시켰는지 확인하십시오. SERV0-ON 이 아니면 서보모터는 구동되지 않습니다.

☐ 서보모터의 Alarm Clear 를 사용한다면 Alarm Clear 신호가 Active 되어 있는지 확인하십시오. 서보드라이버는 Alarm Clear 신호가 Active 된 상태에서는 구동되지 않습니다.

☐ 서보모터를 사용하는 경우에 서보드라이버의 제어모드가 위치제어모드로 설정되었는지 확인하십시오.

☐ 각 케이블 및 커넥터가 단단히 연결되었는지 확인하십시오.

☐ 속도설정이나 이동거리가 너무작지는 않는지 확인하십시오. 너무 저속으로 구동되어 움직이지 않는것처럼 보일 수 있습니다.

## ♣. 모터가 한쪽방향으로만 구동됩니다.

이러한 경우는 COMI-LX50x 의 펄스출력모드와 모터드라이버의 입력모드가 다르기 때문에 발생하는 문제가 대부분입니다. 펄스출력모드는 크게 Pulse & Direction 출력



모드와 Two pulse 출력모드로 나뉩니다. 이러한 모드 설정이 모션제어보드와 모터드라이버간에 일치하여야 합니다. COMI-LX50x 의 출력모드설정은 인쇄매뉴얼 146 페이지를 참조하십시오.

#### ♣. Limit(EL)센서나 원점센서가 감지되지 않습니다.

**증상 및 원인 :** 포토인터럽트 센서를 사용하는 경우에 센서에서 권장하는 충분한 전류가 공급되지 않으면 센서가 정상적으로 반응하지 않습니다. 센서가 Close 상태 일때 GND 와 입력신호간의 전압은 0V 이어야 하지만 전압을 체크해보면 0V 보다 높은 전압이 걸릴수 있습니다. 이러한 현상은 센서에 충분한 전류가 공급되지 않아서 발생하는 경우가 많습니다.

**해결책 :** 센서에 공급되는 Power 를 센서에서 권장하는 전류세기만큼 충분히 공급하도록 합니다.

#### ♣. Limit(EL)센서가 감지되어도 즉시 정지하지 않습니다.

☐ +/-EL 의 정지모드를 “감속후 정지” 방식으로 설정하지 않았는지 체크하십시오. 그러한 경우에는 EL 센서가 감지되면 감속후 정지하기 때문에 감속하는 동안에는 모터가 구동됩니다.

☐ 서보모터를 사용하는 경우 서보드라이버의 게인설정이 잘못되어서 반응속도가 너무느린경우에도 이러한 현상이 발생할 수 있습니다. Motion Controller 의 명령은 중지되었으나 반응속도가 느린 서보드라이버는 Feedback 이 아직 Command 에 미치지 않았기 때문에 이미 출력된 Command Position 까지는 제어를 계속하게 됩니다. 이러한 경우에는 서보드라이버의 게인 조정을 다시해주어야 합니다.

#### ♣. 원점복귀시에 EL 위치까지 이동합니다.

**증상 및 원인 :** 속도모드를 Trapezoidal 또는 S-curve speed mode 로 설정한 상태에서 원점센서에서 멈추지 않고 EL 위치까지 밀리는 경우가 발생하는 경우는 감속도가 너무작기 때문입니다. 속도모드를 Trapezoidal 또는 S-curve speed mode 로 설정한 상태에서 원점복귀를 시작하면 원점센서가 감지되면 감속후 정지하게 되는데 감속구간동안의 이동에 의해 EL 위치까지 밀리게 되는 것입니다. 특히 원점복귀모드 1 로 구동한 경우에 이러한 현상이 발생하기 쉽습니다.

**해결책** : 작업속도를 줄이거나, 감속도를 크게하거나, Constant speed 모드로 설정하면 해결됩니다.

**♣. 직선보간 명령을 실행하면 치명적인 에러가 발생한다.**

**증상 및 원인** : 직선보간이동 거리가 모두 0 인지를 체크하십시오. 특히 절대좌표구동인 경우 이전의 위치와 현재 이동할 위치가 같은지를 확인하십시오. 이러한 경우에 구버전의 라이브러리에서는 에러가 발생할 수 있습니다.

**해결책** : 라이브러리 V3.1.0.0 이후의 버전을 사용하시면 이러한 에러가 발생되지 않습니다.

**♣. 명령내린 좌표와 실제 이동한 좌표가 불일치합니다.**

**증상 및 원인** : COMILX\_MC\_StartXXXXTo() 함수군(절대좌표 이동명령을 시작하고 바로 반환되는 함수들)을 사용하는 경우에는 반드시 현재 이동명령이 완료되었는지를 COMILX\_MC\_Done() 함수를 이용하여 체크한 후 다음 명령을 수행해야 합니다. 그렇지 않은 경우 절대좌표체계가 불일치할 수 있습니다.

**해결책** : COMILX\_MC\_Done() 함수를 이용하여 체크한 후 다음 명령을 수행해야 합니다.

**♣. 맨 처음 이동명령에서만 좌표가 불일치합니다.**

☐ 서보온(Servo-ON) 명령을 수행한 직후에 지연없이 이동명령을 내리면 목표좌표보다 덜 이동할 수 있습니다. 이는 서보드라이버가 Enable 되는데 약간의 시간이 필요한데 이동안에 명령이 출력되기 때문입니다.

**♣. 지정한속도까지 작업속도가 도달하지 않습니다.**

☐ 지정한 속도가 모터드라이버의 속도 범위를 넘지 않는지 체크하십시오.

☐ 이동명령의 이동량이 너무작지 않은가 체크하십시오. 이동량이 가속구간과 감속구간내에서 끝날수 있을정도의 적은 양이면 COMI-LX50x 는 “Triangular Drive” 를

---

피하기 위하여 작업속도를 자동으로 낮춥니다.

☐ 논리속도설정 (COMILX\_MC\_SetUnitSpeed)이 올바르게 되었는지 체크하십시오.

### **♣. 모션빌더프로그램에서 3D-Graph 가 표시되지 않습니다.**

☐ 모션빌더의 3D-Graph 가 표시되지 않으면 디스플레이 등록정보에서 “하드웨어 가속” 항목을 조정하십시오. 대부분의 경우 “없음” 으로 설정하면 문제가 해결될 것입니다.

## Trouble Shooting 및 FAQ

---

저작권자 : ㈜커미조아

*Copyright (c) by COMIZOA CO.,LTD. All right reserved.*

수정 날짜 : 2004 년 2 월 16 일

이 사용자 설명서는 저작권법에 의해 보호되고 있습니다.

㈜커미조아의 사전 서면 동의 없이 사용자설명서의 일부 또는 전체를 어떤 형태로든 복사, 전  
재할 수 없습니다.

Hardware Support : [Hardware@comizoa.co.kr](mailto:Hardware@comizoa.co.kr)

Software Support : [Software@comizoa.co.kr](mailto:Software@comizoa.co.kr)



㈜커미조아

[www.comizoa.co.kr](http://www.comizoa.co.kr)

[www.comizoa.com](http://www.comizoa.com)

Tel) 042 - 861 - 3301~3

Fax) 042 - 861 - 3304