COMIZOA DAQ System

LX•Series Manual

-Motion Controller part-

COMIZOA

COMIZOA

COMputer Innovation
is Zoomed by Our Affection!

저작권자 : ㈜커미조아

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

2003 년 4 월 7 일 중판 인쇄

이 사용자 설명서는 저작권법에 의해 보호되고 있습니다.

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

Hardware Support : Hardware@comizoa.co.kr

Software Support : Software@comizoa.co.kr

㈜커미조아

www.comizoa.co.kr

www.comizoa.com

Tel) 042 - 861 - 3301~3

Fax) 042 - 861 - 3304

차 례

CHAPTER 1 COMI-LX50X 모션제어 전용 보드 소개 2

1.1 COMI-LX50x 시리즈 제품 사양 및 특징 3

1.2 COMI-LX50x 시리즈를 이용한 모션제어 시스템 구성 5

1.2.1 COMI-LX501 을 이용한 모션제어 시스템 구성 5

1.2.2 COMI-LX502/4/8 을 이용한 모션제어 시스템 구성 7

1.3 소프트웨어 지원 11

CHAPTER 2 장치 제어기 및 소프트웨어 설치 14

1.1 WINDOWS98 및 WINDOWSME 드라이버 설치 14

1.2 WINDOWS2000 드라이버 설치 18

1.3 유틸리티 프로그램 및 라이브러리 설치 22

CHAPTER 3 하드웨어 가이드 30

3.1 MOTION CONTROLLER 30

3.1.1 COMI-LX501 4 축 제어용 Motion Controller 30

3.1.2 COMI-LX502 2 축 제어용 Motion Controller 36

3.1.3 COMI-LX504 4 축 제어용 Motion Controller 43

3.1.4 COMI-LX508 8 축 제어용 Motion Controller 51

3.2 MOTION TERMINAL 62

3.2.1 COMI-LXT5 Motion Terminal Board 62

3.2.2 COMI-STT5B Motion Terminal Board 65

3.2.3 COMI-LXT5M 미쯔비시 서보드라이버 전용 Terminal Board 69

3.2.3 COMI-LXT5S 삼성 서보드라이버 전용 Terminal Board *73*
3.3 입/출력 신호 설명 78
3.3.1 펄스 출력 신호 (CW/CCW) *79*
3.3.2 엔코더 입력 신호 (EA, EB, EZ) *82*
3.3.3 Servo-ON(SVON) *85*
3.3.4 ALARM 입력 신호(ALM) *86*
3.3.5 In-position 입력 신호(INP) *86*
3.3.6 편차카운터 클리어 출력 신호(ERC) *87*
3.3.7 End Limit 입력 신호 (+EL, -EL) *88*
3.3.8 원점 센서 입력 신호 (HOME or ORG) *90*
3.3.9 Ramping down 입력 신호 (SD) *91*
3.3.10 External Switch Operation 입력 신호 (+DR, -DR) *93*
3.3.11 Manual Pulser 입력 신호(PA, PB) *95*
3.3.12 카운터 클리어 입력 신호(CLR) *96*
3.3.13 General Digital Input/Output(DI#, DO#) *97*
3.4 하드웨어 배선 예 99
3.4.1 MITSUBISHI MR-J2 서보 드라이버 배선 예 *99*
3.4.2 YASKAWA SGDM 서보 드라이버 배선 예 *100*
3.4.3 PANASONIC 서보 드라이버 배선 예 *101*
3.4.4 SANYO AC Servo PY2 배선 예 *102*
3.4.5 SAMSUNG CSD Seriese 서보드라이버 배선예 *103*

CHAPTER 4 COMI-XMASTER 유틸리티 프로그램 106

4.1 DIGITAL IN/OUT 107
4.2 MOTION CONTROL 109
4.2.1 Motion Status 창 *111*
4.2.2 Digital I/O 창 *112*
4.2.3 Graph 창 *112*
4.2.4 Pulse Input/Output 설정 *114*
4.2.5 Motion I/O 환경설정 *116*

CHAPTER 5 C/C++라이브러리 120

5.1 라이브러리 사용 방법 121

5.1.1 Visual C/C++에서의 라이브러리 사용법 121

5.1.2 Borland C++ Builder 에서의 라이브러리 사용법 122

5.1.3 Labwindos CVI 에서의 라이브러리 사용법 123

5.2 라이브러리 및 디바이스 시작/종료 124

5.3 디지털 입출력(DIGITAL INPUT/OUTPUT) 130

5.4 모션 제어(MOTION CONTROL) 141

5.4.1 모션 초기화 및 환경설정 함수 142

5.4.2 Single Axis 모션 제어 함수 160

5.4.3 Multi-Axis 동시제어 함수 198

5.4.4 Coordinated Motion 함수 215

5.4.5 속도 및 위치 오버라이딩(Overriding) 함수 266

5.4.6 원점 복귀(Home Return) 함수 274

5.4.7 Manual Pulser 모드 모션 제어 함수 287

5.4.8 External Switch Operation 모드 모션 제어 함수 302

5.4.9 리스트 모션(Listed Motion) 함수 308

5.4.10 상태 감시 및 제어 함수 317

5.4.11 I/O(입출력) 환경설정 함수 341

5.4.12 인터럽트 관련 함수 366

APPENDIX A 라이브러리 함수 리스트 380

A.1 함수 색인 380

CHAPTER 1

COMI-LX50x 모션제어 전용 보드 소개

본 장에서는 COMI-LX50x Advanced Motion Controller 의 개요 및 특징을 간단히 설명합니다.

1.1 COMI-LX50x 시리즈 제품 사양 및 특징 ····· 3
1.2 COMI-LX50x 시리즈를 이용한 모션제어 시스템 구성 ····· 5
1.3 소프트웨어 지원 ····· 11

CHAPTER 1 COMI-LX50x 모션제어 전용 보드 소개

	COMI-LX501	COMI-LX502	COMI-LX504	COMI-LX508
버 스	PCI	PCI	PCI	PCI
제어축수	4 축	2 축	4 축	8 축
출력속도범위	0.1PPS ~ 6.55MPPS	0.1PPS ~ 6.55MPPS	0.1PPS ~ 6.55MPPS	0.1PPS ~ 6.55MPPS
가감속 모드	대칭/비대칭 사다리꼴, S-커브	대칭/비대칭 사다리꼴, S-커브	대칭/비대칭 사다리꼴, S-커브	대칭/비대칭 사다리꼴, S-커브
보간기능	직선, 원호	직선, 원호	직선, 원호	직선, 원호
다축동기구동	Programmed or External switch	Programmed or External switch	Programmed or External switch	Programmed or External switch
원점복귀모드	13 가지 모드 지원	13 가지 모드 지원	13 가지 모드 지원	13 가지 모드 지원
펄서(Pulser) 구동	지원	지원	지원	지원
외부스위치 구동	지원	지원	지원	지원
슬립/백래쉬 보정	지원	지원	지원	지원
위치비교 펄스 출력	지원안함	지원	지원	지원
General DIO	각 16 채널	각 3 채널	각 6 채널	각 12 채널
터미널 보드	필수, COMI-LXT5	선택, COMI-LXT5S, COMI-LXT5M	선택, COMI-LXT5S, COMI-LXT5M	선택, COMI-LXT5S, COMI-LXT5M

[표 1-1] COMI-LX50x 시리즈 제품 목록과 주요 사양

COMI-LX50x 시리즈는 PCI 기반의 고성능 모션제어 전용 보드입니다. COMI-LX50x 시리즈 모션제어 전용 보드에는 2 축 제어용인 COMI-LX502, 4 축 제어용인 COMI-LX501, COMI-LX504, 8 축 제어용인 COMI-LX508 이 있습니다. [표 1-1]은 COMI-LX50x 시리즈 모션제어 전용 보드 제품 리스트 및 각 제품의 주요 사항을 리스트한 것입니다.

COMI-LX50x 시리즈 모션제어 전용 보드는 6.55MHz 까지의 펄스를 자유자재로 생성하므로써 스텝 모터 또는 서보 모터를 정밀 제어할 수 있도록 해주는 장치입니다. Trapezoidal 및 S-curve 가감속 속도 프로파일을 지원하며 2 축 원호 보간 및 2~4 축

직선 보간 기능, 그리고 속도의 연속성을 가지는 보간 기능을 제공합니다. 또한 속도나 위치를 구동중에 보정할 수 있도록 되어 있어서 구동 중 환경의 변화에 의해 속도나 위치를 바꿔야하는 상황에 능동적으로 대처하는 것이 가능합니다. 모든 축에 대해서 각각 제공되는 엔코더 인터페이스는 부정확한 기계적 메커니즘에 의해 발생되는 위치 오차를 교정할 수 있는 기능을 제공하며, 특히 다양하게 제공되는 각종 비교기 기능은 위치의 에러 검출 뿐 아니라 트리거 기능으로 사용될 수 있어서 Machine Vision 장치와 같은 외부장치와의 상호 연계작업을 수행하는데 정확한 타이밍 제어에 유용한 기능을 제공할 수 있습니다. 그밖에 보다 정확하고 안정한 시스템 구축을 지원하기 위하여 다양한 기계적 센서 인터페이스, 서보모터 인터페이스 그리고 범용적으로 사용될 수 있는 I/O 인터페이스를 제공하고 있습니다.

1.1 COMI-LX50x 시리즈 제품 사양 및 특징

▷ 32-Bit PCI-Bus plug and play
▷ 스텝모터 및 서보모터 제어용 Step/Direction 펄스 출력
▷ 펄스 출력 옵션 : OUT/DIR, CW/CCW
▷ 최고 6.55 Mpps 의 고속 제어
▷ 사다리꼴, S 자 가감 속도 제어
▷ PTP 구동
▷ 2 축에서 4 축까지의 직선보간 구동
▷ 원호보간 구동
▷ 각축 개별구동 및 다축동기 구동
▷ 모터 구동중 속도 및 이동량 오버라이드
▷ 13 가지 모드의 다양한 자동 원점 복귀 기능
▷ 실제 이동 거리 Count 기능
▷ 현재 지령 위치 및 속도 실시간 모니터링 기능
▷ 엔코더 입력 (1 상 Up/Down, 2 상 1, 2, 4 체배)
▷ 원점 검색 및 신호 검색 기능
▷ 백래쉬 및 슬립 보정 기능
▷ 지령(실제) 위치 설정에 따른 Trigger 신호 출력 기능
▷ H/W Limit 신호 입력시 서보 드라이버 잔류 펄스 클리어 기능
▷ 각 축의 모션 구동 종료 시 인터럽트 발생기능
▷ 외부 기계 신호 입력의 동작 Level 설정 기능
▷ 위치 결정 완료 신호, 알람 신호, 감속 정지 신호, 급정지 신호
▷ 외부 센서에 의한 감속 및 급정지 기능
▷ Slow Down Stop, Emergency Stop
▷ Stepping 및 Servo Driver 인터페이스

Applicable Motors

▷ Stepping motors.
▷ AC or DC servomotors with pulse train input servo drivers.

Performance

▷ Number of controllable axes: 2, 4, 8 axes[1].
▷ Pulse rate setting range : 0.1 PPS ~ 6.55 MPPS
▷ Velocity profile : Linear, Trapezoidal, S-Curve velocity profile drive.
▷ Internal reference clock: 20 MHz
▷ Position pulse setting range: -134,217,728~ +134,217,728 pulses.
▷ Up / down counter counting range: 0~268,435,455 (28-bit) or -134,217,728 to +134,217,727

I/O Signals

▷ Input/Output Signals for each axis
▷ All I/O signal are optically isolated with 2500Vrms isolation voltage
▷ Command pulse output pins: OUT and DIR.
▷ Incremental encoder signals input pins: EA and EB.
▷ Encoder index signal input pin: EZ.
▷ Mechanical limit/switch signal input pins: ±EL, SD/PCS and ORG.
▷ Servomotor interface I/O pins: SVON[2], RDY[3], INP, ALM and ERC.
▷ Position latch input pin: LTC
▷ Position compare output pin: CMP[4].
▷ Pulser signal input pin: PA and PB.
▷ Simultaneous Start/Stop signal I/O pins: STA and STP.

General-Purposed Digital Input/Output[5]

▷ Isolated Digital Input
▷ Open Collector Digital Output

---

[1] COMI-LX502:2 축, COMI-LX501/LX504:4 축, COMI-LX508:8 축
[2] COMI-LX501 은 SVON 출력이 따로 정의 되어있지 않고 디지털 출력 채널중 하나를 사용합니다.
[3] COMI-LX501 은 RDY 입력이 따로 정의 되어있지 않고 디지털 입력 채널중 하나를 사용합니다.
[4] COMI-LX501 에서는 CMP 출력을 지원하지 않습니다.
[5] COMI-LX501:각 16 채널, COMI-LX502:각 3 채널, COMI-LX504:각 6 채널, COMI-LX508:각 12 채널

General Specifications

▷ Connectors: 68-pin VHDCI-type connector[6)]
▷ Operating Temperature: 0°C ~ 50°C
▷ Storage Temperature: -20°C ~ 80°C
▷ Humidity: 5 ~ 85%, non-condensing

Power Consumption:

▷ Slot power supply(input): +5V DC ±5%, 900mA max.
▷ External power supply(input): +24V DC ±5%, 500mA max.
▷ External power supply(output): +5V DC ±5%, 500mA, max.

Dimension

▷ COMI-LX501 : 200mm X 114mm
▷ COMI-LX502 : 256mm X 114mm
▷ COMI-LX504 : 256mm X 114mm
▷ COMI-LX508 : 256mm X 114mm

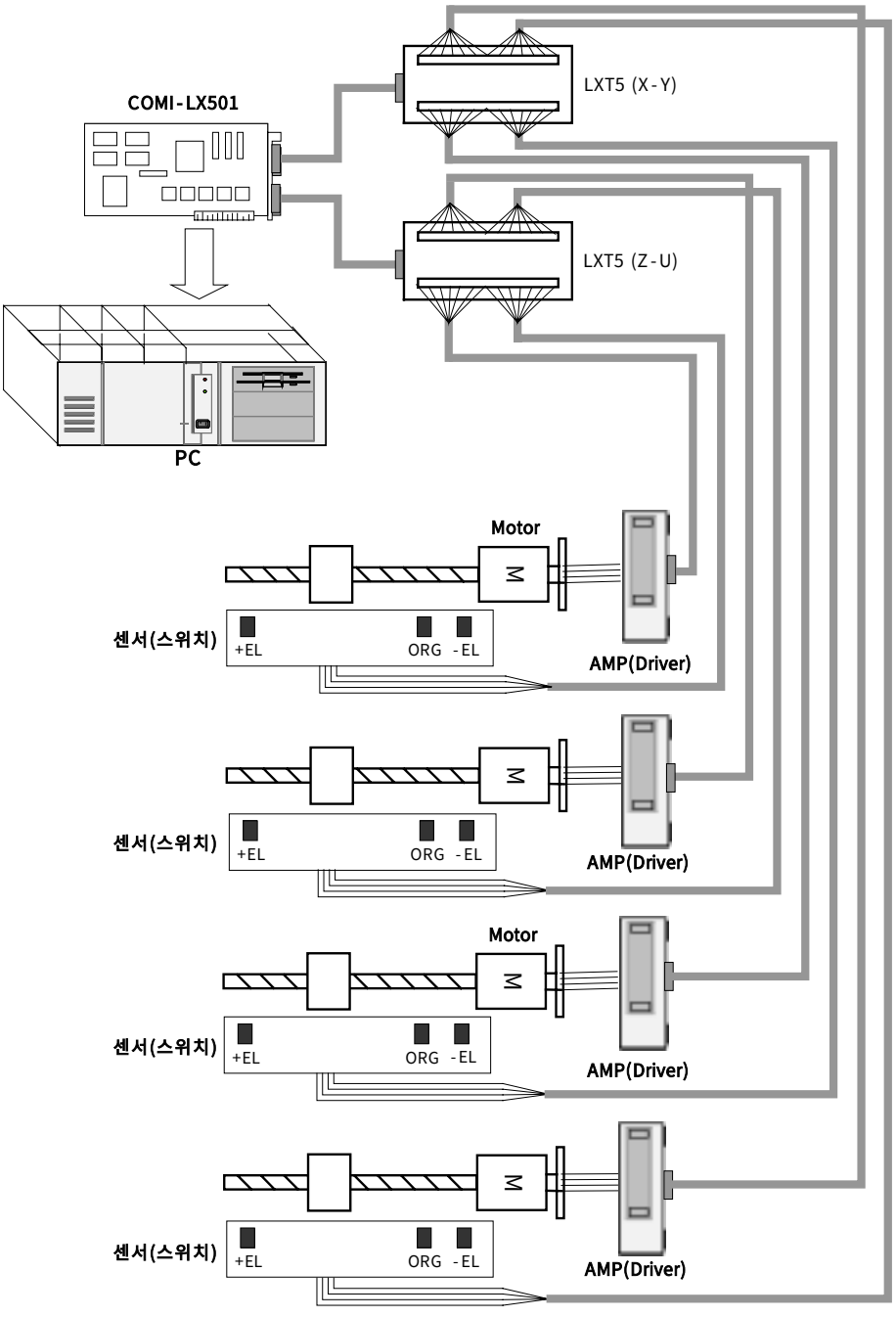
1.2 COMI-LX50x 시리즈를 이용한 모션제어 시스템 구성

1.2.1 COMI-LX501 을 이용한 모션제어 시스템 구성

COMI-LX501 은 하나의 보드에서 4 축을 동시에 또는 개별적으로 제어할 수 있는 모션 제어 보드입니다. COMI-LX501 과 입출력되는 신호들은 COMI-LXT5 터미널 보드에서 Isolation 처리됩니다. 따라서 COMI-LX501 은 COMI-LXT5 와 쌍을 이루어 사용되어야 하며 신호선은 COMI-LXT5 단자에 연결합니다. COMI-LXT5 는 2 축 모션 인터페이스를 제공하므로 4 축을 제어하기 위해서는 두 개의 COMI-LXT5 터미널 보드가 필요합니다. COMI-LX501 을 이용하여 모션제어 시스템을 구성하는 예는 [그림 1-1]과 같습니다.

COMI-LX501 및 COMI-LXT5 터미널 보드에 대한 자세한 사양은 “CHAPTER 2 하드웨어 가이드” 편을 참조하십시오.

6) COMI-LX502:1*68-pin VHDCI, COMI-LX501/4:2*68-pin VHDCI, COMI-LX508:4*68-pin VHDCI



[그림 1-1] COMI-LX501을 이용한 4축 모션제어 시스템 구성도

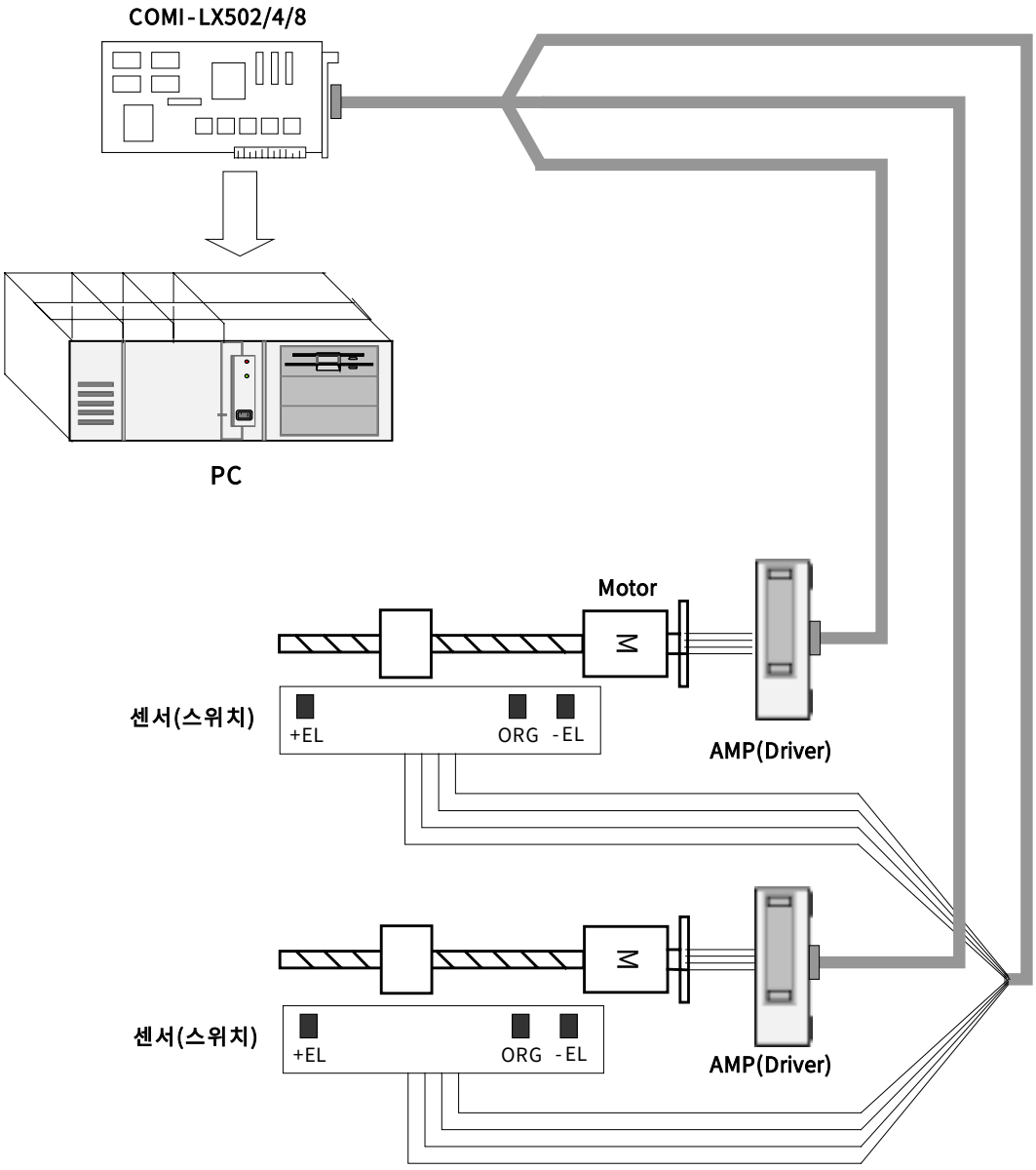
1.2.2 COMI-LX502/4/8 을 이용한 모션제어 시스템 구성

COMI-LX502/4/8 은 각각 하나의 보드에서 2 축, 4 축, 8 축을 동시에 또는 개별적으로 제어할 수 있는 모션제어 보드입니다. COMI-LX502 는 68-Pin 커넥터가 하나 있으며, COMI-LX504 는 2 개, COMI-LX508 은 4 개가 있습니다. 하나의 68-Pin 커넥터는 2 축의 모션을 제어할 수 있습니다

COMI-LX502/4/8 은 모두 보드내에서 모든 신호의 Isolation 처리를 하였기 때문에 터미널보드를 사용하지 않고도 시스템을 구성할 수 있습니다. [그림 1-2]는 COMI-LX502/4/8 를 이용하여 모션제어 시스템을 구성하는 일반적인 방법으로써 터미널을 사용하지 않고 COMI-LX502/4/8 의 68 핀 커넥터에 각종 신호선을 직접 결선하는 예입니다(그림에서는 2 축 연결만 도시하였으나, COMI-LX504 는 4 축, COMI-LX508 은 8 축을 같은 방법으로 연결할 수 있습니다). 이러한 경우에 사용자가 직접 결선작업을 하거나 커미조아에서 유상 제공하는 전용케이블을 사용하실 수 있습니다.

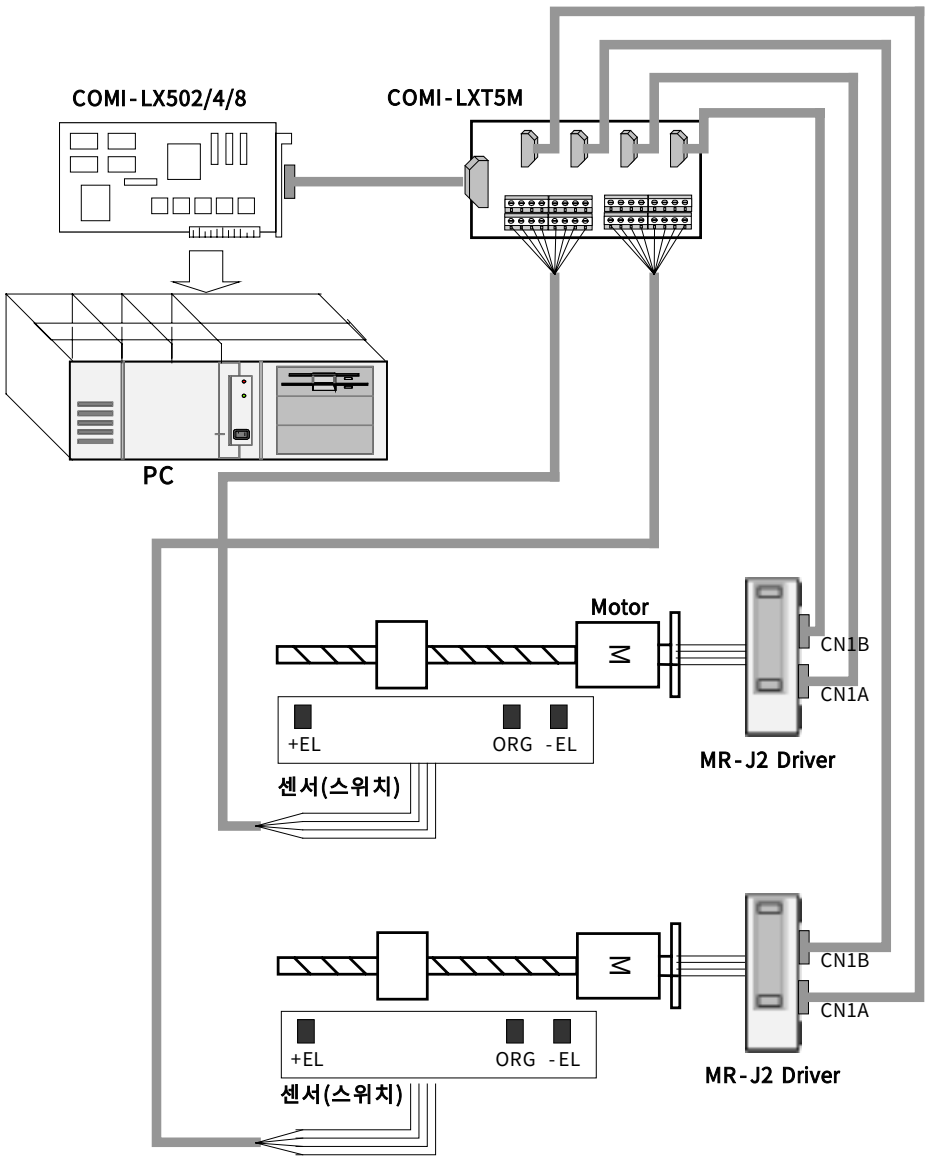
COMI-LX502/4/8 을 미쯔비시의 MR-J2 서보모터 드라이버나 삼성의 서보모터 드라이버 등과 함께 사용하는 경우에는 사용자가 직접 결선작업시에 겪는 불편함과 부주의로 인해 발생되는 여러가지 문제를 해결하기 위하여 전용 터미널 및 케이블을 따로 제공합니다. [그림 1-3]과 [그림 1-4]와 같이 전용 터미널 및 케이블을 사용하시면 서보 모터 드라이버와의 각종 신호 연결을 간단히 처리할 수 있습니다.

[그림 1-3]과 같이 미쯔비시의 MR-J2 서보모터 드라이버를 사용하는 경우에는 COMI-LXT5M MR-J2 전용 터미널 보드를 사용하실 수 있습니다. 그리고 [그림 1-4]와 같이 삼성의 서보모터 드라이버를 사용하는 경우에는 COMI-LXT5S 삼성 서보 전용 터미널 보드를 사용하실 수 있습니다. 기타 서보 드라이버의 경우에는 ㈜커미조아에 전용 터미널 보드에 관하여 문의하시기 바랍니다. 이 전용 터미널 보드들은 각각의 서보 드라이버에 맞게 서보 인터페이스 I/O 신호들을 커넥터 처리하고 각각의 커넥터에 전용 케이블을 장착 연결하기만 하면 서보 드라이버와 모든 인터페이스 신호 연결이 모두 이루어지도록 설계되었습니다. 따라서 사용자께서는 외부 센서들(Machin I/O)에 관련된 결선 작업만 수행하시면 됩니다.



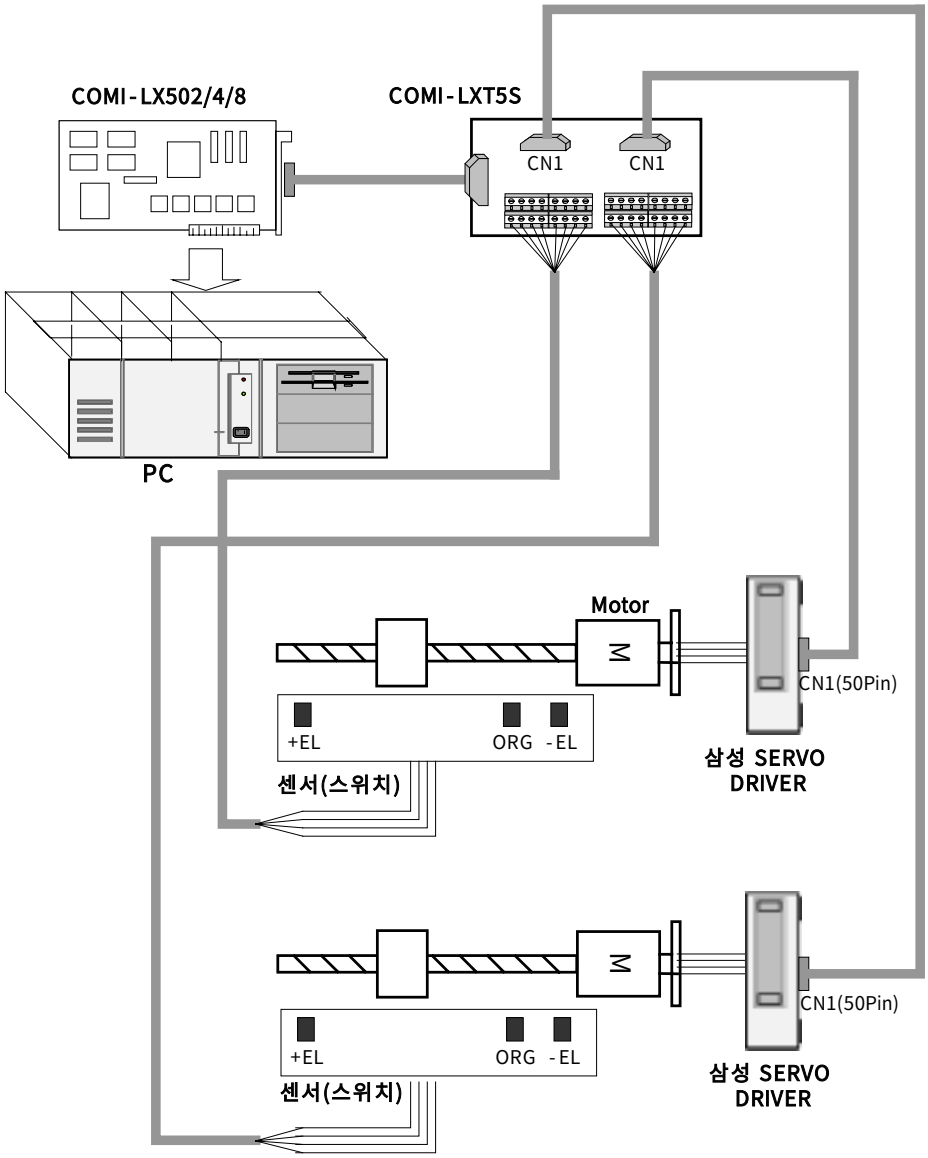
[그림 1-2] COMI-LX502/4/8 을 이용한 일반적인 모션제어 시스템 구성도 [1)]

[1)] 그림에서는 2 축 연결만 도시하였으나, COMI-LX504 는 4 축, COMI-LX508 은 8 축을 같은 방법으로 연결할 수 있습니다



[그림 1-3] COMI-LX502/4/8 과 미쯔비시 MR-J2 서보 드라이버 전용 터미널(COMI-LXT5M)을 이용한 모션제어 시스템 구성도 [1)]

[1)] 그림에서는 2 축 연결만 도시하였으나, COMI-LX504 는 4 축, COMI-LX508 은 8 축을 같은 방법으로 연결할 수 있습니다



[그림 1-4] COMI-LX502/4/8 과 삼성 서보 드라이버 전용 터미널(COMI-LXT5S)을 이용한 모션제어 시스템 구성도 [1)]

[1)] 그림에서는 2 축 연결만 도시하였으나, COMI-LX504 는 4 축, COMI-LX508 은 8 축을 같은 방법으로 연결할 수 있습니다

1.3 소프트웨어 지원

Utility Program

㈜커미조아는 COMI-LX50x Motion Controller 의 기능을 테스트하고 간단한 조작을 해볼 수 있도록 COMI-XMaster 유틸리티 소프트웨어를 제공합니다. 이에 대한 자세한 내용은 CHAPTER 4 를 참조하시기 바랍니다.

Programming Library

㈜커미조아는 사용자가 COMI-LX50x 을 이용하는 프로그램을 보다 쉽게 개발할 수 있도록 라이브러리를 제공합니다. 지원되는 라이브러리는 Visual C++, Borland C++ (Builder), Labwindows CVI 등에서 사용할 수 있는 DLL 형태의 C/C++라이브러리와 Visual Basic, Delphi 등에서 사용할 수 있는 OCX 형태의 라이브러리로 구성됩니다. C/C++라이브러리에 대한 자세한 내용은 CHAPTER 5 를 참조하시기 바랍니다. 그리고 OCX 형태의 라이브러리에 대한 설명은 PDF 파일로 제공되며, 본 매뉴얼에는 수록되지 않았습니다.

CHAPTER 2

장치 제어기 및 소프트웨어 설치

본 장에서는 COMIDAS 장치 제어기를 설치하는 방법을 설명합니다. 사용자는 제어기를 설치하셔야 윈도우에서 COMIDAS 디바이스를 사용하실 수 있습니다(도스에서는 제어기를 설치할 필요가 없습니다). 모든 종류의 COMIDAS 디바이스는 그 설치 방법이 동일합니다.

2.1 Windows98 및 Windows ME 드라이버 설치 ····· 14
2.2 Windows 2000 드라이버 설치 ····· 18
2.3 유틸리티 프로그램 및 라이브러리 설치 ····· 22

CHAPTER 2 장치 제어기 및 소프트웨어 설치

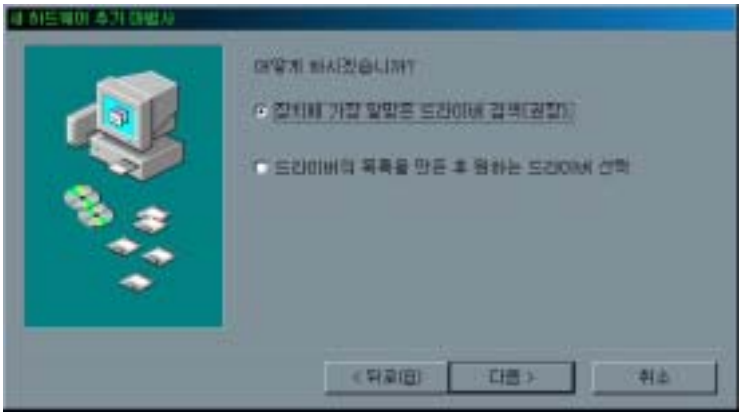
본 장에서는 COMIDAS 장치 제어기를 설치하는 방법을 설명합니다. 사용자는 제어기를 설치하셔야 윈도우에서 COMIDAS 디바이스를 사용하실 수 있습니다(도스에서는 제어기를 설치할 필요가 없습니다). 모든 종류의 COMIDAS 디바이스는 그 설치 방법이 동일합니다.

1.1 Windows98 및 WindowsME 드라이버 설치

먼저 컴퓨터 전원을 꺼주신 다음에 COMIDAS 디바이스 적당한 PCI 슬롯에 장착하고 시스템을 부팅시킵니다. 윈도우가 시작되면서 윈도우는 자동으로 COMIDAS 디바이스를 인식하여 [그림 2-1]과 같이 PCI Card 를 찾았다는 화면을 나타냅니다. 제공된 설치 CD 를 CDROM 드라이브에 삽입하신 후 '다음' 버튼을 클릭하십시오.

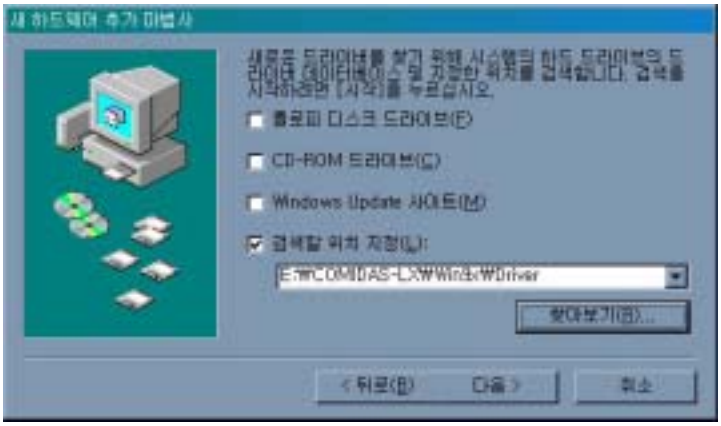
[그림 2-1] 디바이스 제어기 설치 화면 1

[그림 2-2]와 같은 화면이 나타나면, '장치에 가장 알맞은 드라이버 검색(권장)'을 선택한 후 '다음' 버튼을 클릭하십시오.



[그림 2-2] 디바이스 제어기 설치 화면 2

[그림 2-3]과 같은 화면이 나타나면, '검색할 위치 지정(L):' 항목에 "E:WCOMIDAS-LXWWin9xWDriver"를 입력한 후 '다음' 버튼을 클릭하십시오. 이때 "E:W"는 CDROM 드라이브를 의미합니다. 이미 소프트웨어 설치를 한 경우에는 설치된 디렉토리의 하위 디렉토리인 WindowWDriver 디렉토리를 지정하여도 됩니다.

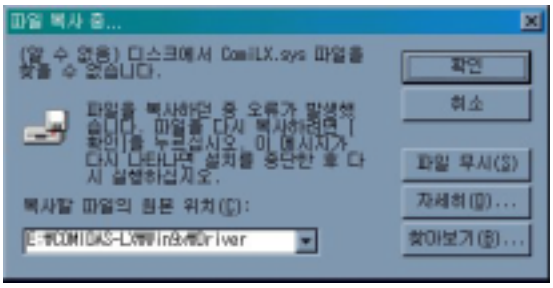


[그림 2-3] 디바이스 제어기 설치 화면 3

[그림 2-4]와 같이 COMIDAS 디바이스 장치가 검색되었다는 화면이 나타나면, '다음' 버튼을 클릭하십시오. 그러면 윈도우는 필요한 파일들을 설치합니다.

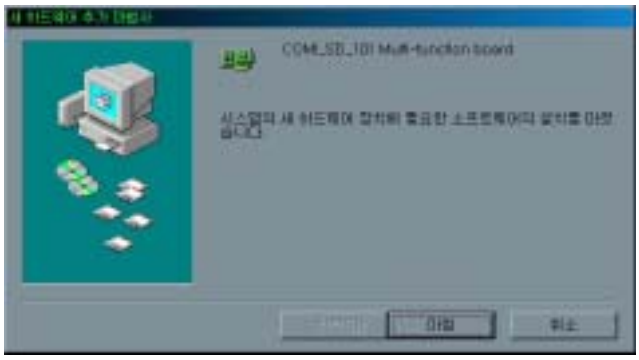
[그림 2-4] 디바이스 제어기 설치 화면 4

경우에 따라서는 [그림 2-5]와 같이 파일을 찾을 수 없다는 메시지가 나타날 수 있습니다. 이 때에는 무시하고 그냥 “확인” 버튼을 누르면 설치를 계속 진행합니다.



[그림 2-5] 파일을 찾을 수 없다는 오류 메시지

필요한 드라이버 파일의 설치가 완료되면 [그림 2-6]과 같은 화면이 나타나면 ‘마침’ 버튼을 클릭하여 제어기 설치를 마칩니다.



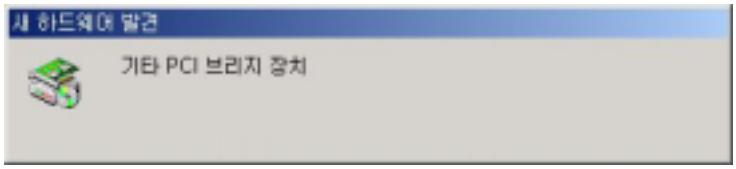
[그림 2-6] 디바이스 제어기 설치 화면 5

제어기 설치를 마친 후 제어판의 시스템 정보를 실행하면 [그림 2-7]과 같이 COMIDAS 디바이스가 설치된 것을 확인할 수 있습니다.

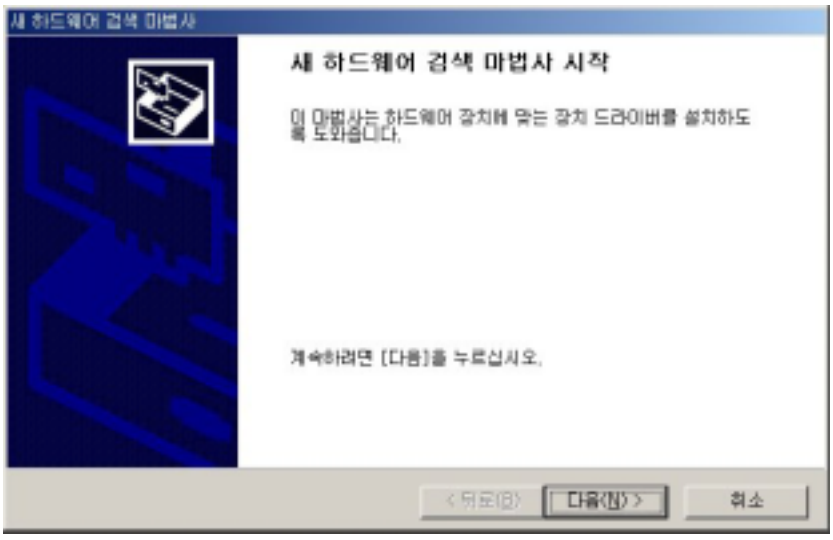
[그림 2-7] 시스템 등록 정보에 디바이스 설치 확인

1.2 Windows2000 드라이버 설치

먼저 컴퓨터 전원을 꺼주신 다음에 COMIDAS 디바이스를 적당한 PCI 슬롯에 장착하고 시스템을 부팅시킵니다. 윈도우가 시작되면서 윈도우는 자동으로 COMIDAS 디바이스를 인식하여 [그림 2-8]과 같이 PCI Card 를 찾았다는 화면을 나타낸 후 [그림 1-9]와 같은 화면을 나타냅니다. 제공된 설치 CD 를 CDROM 드라이브에 삽입하신 후 '다음' 버튼을 클릭하십시오.

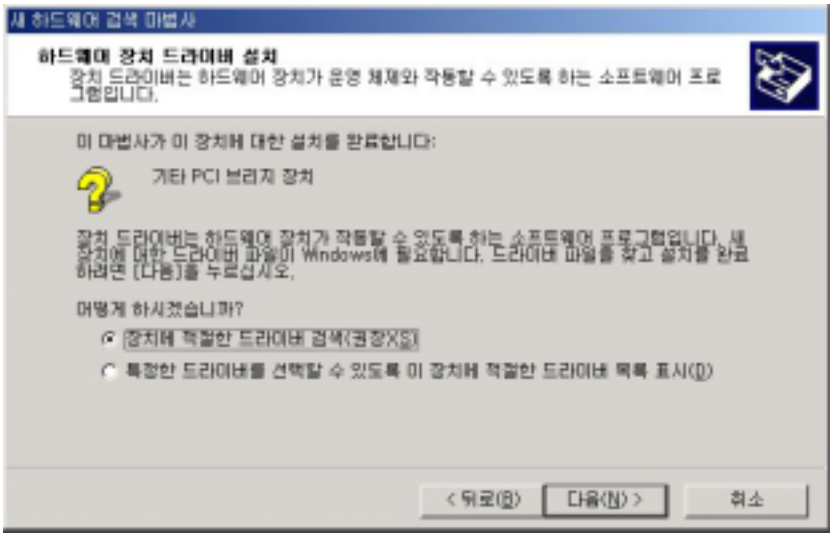


[그림 2-8] Win2000 디바이스 제어기 설치 화면 1



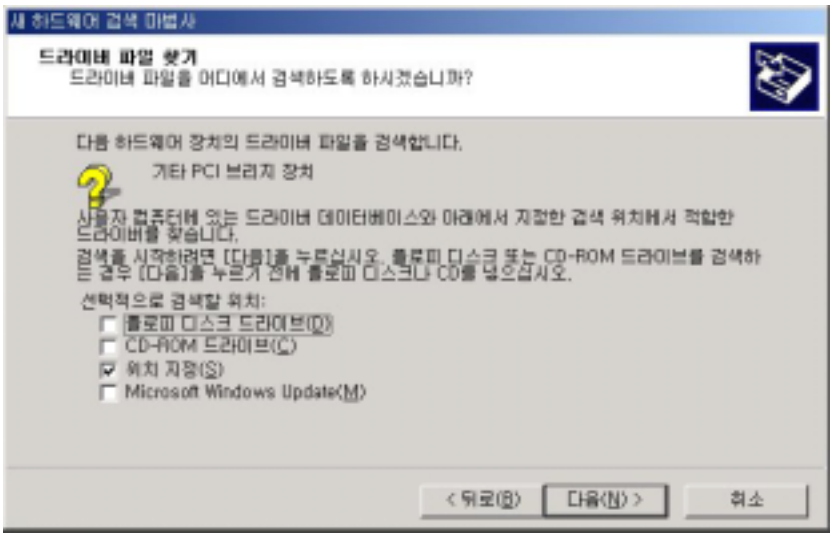
[그림 2-9] Win2000 디바이스 제어기 설치 화면 2

[그림 2-10]과 같은 화면이 나타나면, '장치에 적절한 드라이버 검색(권장)' 을 선택한 후 '다음' 버튼을 클릭하십시오.



[그림 2-10] Win2000 디바이스 제어기 설치 화면 3

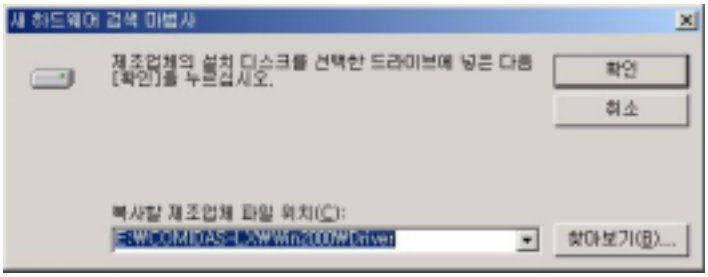
[그림 2-11]과 같은 화면이 나타나면, '위치 지정'을 선택한 후 '다음' 버튼을 클릭하십시오.



[그림 2-11] Win2000 디바이스 제어기 설치 화면 4

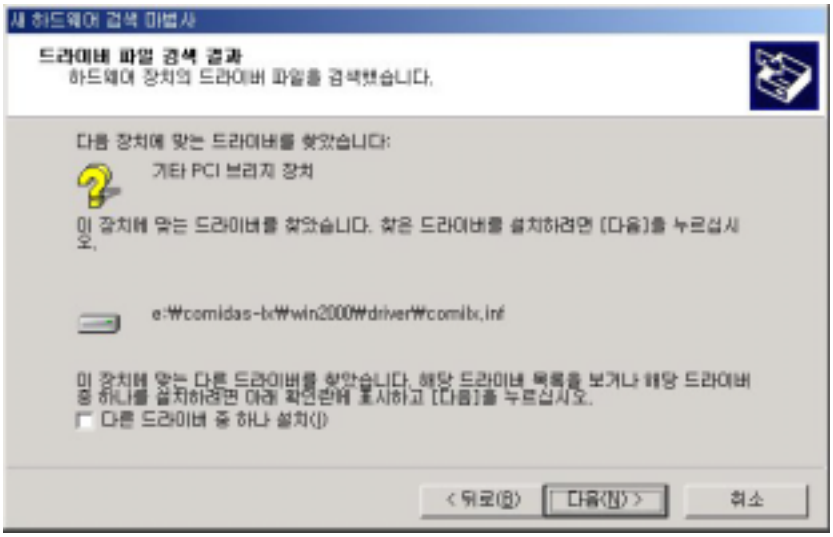
[그림 2-12]와 같은 화면이 나타나면, '복사할 제조업체 파일 위치(C):' 항목에

"E:₩COMIDAS-LX₩Win2000₩Driver"를 입력한 후 '다음' 버튼을 클릭하십시오. 이때 "E:₩"는 CDROM 드라이브를 의미합니다. 이미 소프트웨어 설치를 한 경우에는 설치된 디렉토리의 하위 디렉토리인 Window₩Driver 디렉토리를 지정하여도 됩니다.



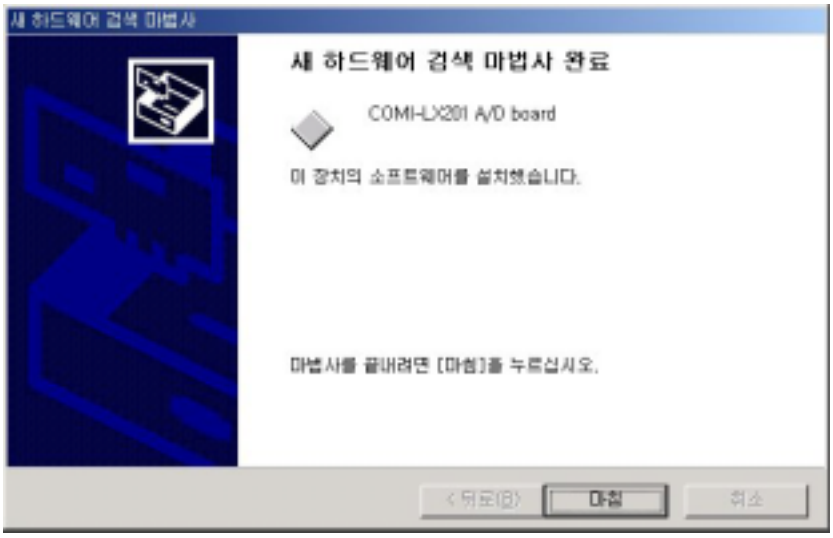
[그림 2-12] Win2000 디바이스 제어기 설치 화면 5

[그림 2-13]과 같이 기타 PCI 브리지 장치(또는 COMIDAS 모델명)가 검색되었다는 화면이 나타나면, '다음' 버튼을 클릭하십시오. 그러면 윈도우는 필요한 파일들을 설치합니다.



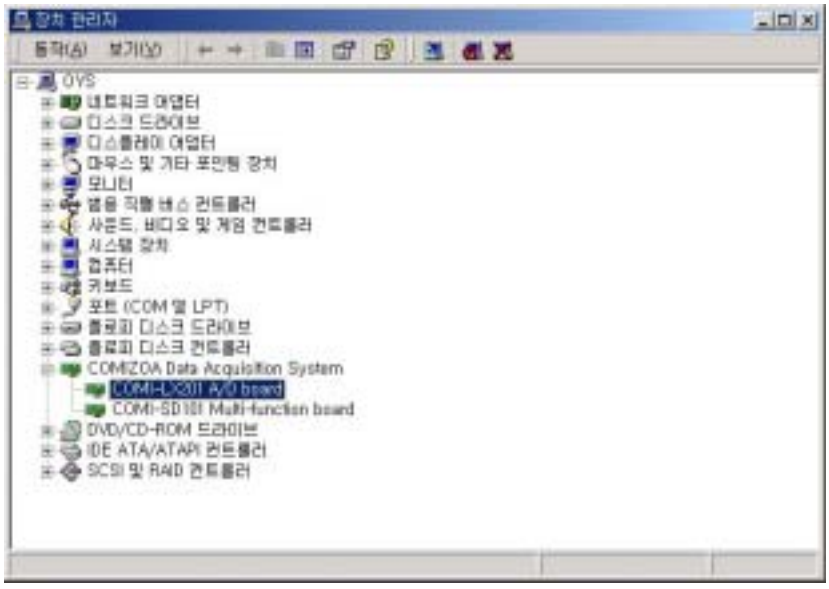
[그림 2-13] Win2000 디바이스 제어기 설치 화면 6

[그림 2-14]와 같은 화면이 나타나면 '마침' 버튼을 클릭하여 제어기 설치를 마칩니다.



[그림 2-14] Win2000 디바이스 제어기 설치 화면 7

제어기 설치를 마친 후 장치 관리자를 실행하면 [그림 2-15]와 같이 COMIDAS 디바이스가 설치된 것을 확인할 수 있습니다.



[그림 2-15] Win2000 에서 COMIDAS 디바이스의 설치를 확인하는 화면

1.3 유틸리티 프로그램 및 라이브러리 설치

제공된 설치 CD 를 CDROM 드라이브에 삽입하면 자동 실행 프로그램이 실행되어 [그림 2-16]과 같은 화면이 나타납니다.

[그림 2-16] 설치 CD 의 자동실행 초기화면

[그림 2-16]화면의 상단 메뉴에서 Install 메뉴를 선택하면 [그림 2-17]과 같은 화면이 나타납니다. Win98 이나 WinME 사용자는 “COMIDAS-LX for W9x” 버튼을 Win2000 사용자는 “COMIDAS-LX for W2000”버튼을 클릭하면 COMIDAS-LX 소프트웨어 설치 프로그램이 실행됩니다.

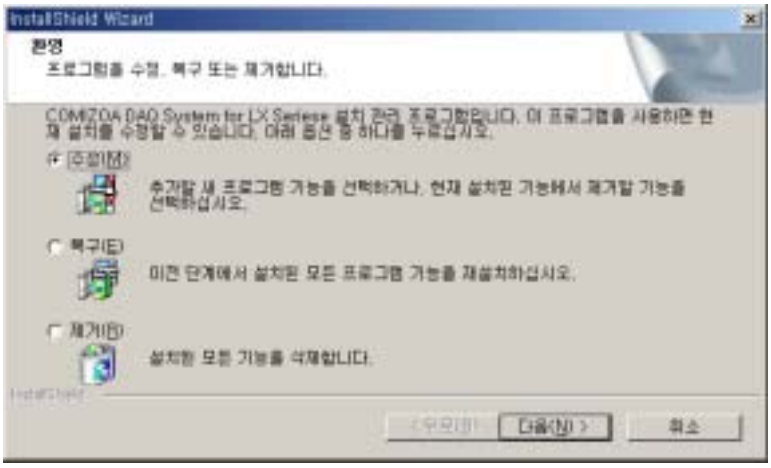
[그림 2-17] 설치 CD 의 자동실행 프로그램의 Install 메뉴 화면

[그림 2-18]과 같은 화면이 나타나면 “다음” 버튼을 클릭하여 프로그램 설치를 시작합니다.

[그림 2-18] 프로그램 설치 시작 화면

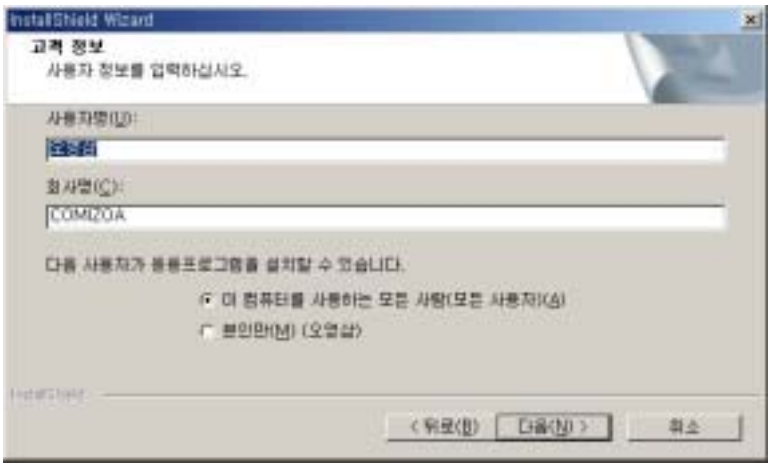
만일 COMIDAS-LX 프로그램이 이미 설치되어 있으면 [그림 2-18]화면 대신 [그림 2-19] 화면이 나타나게 됩니다. [그림 2-19] 화면에서 프로그램을 다시 설치하려면

"복구" 옵션을 선택한 후 "다음" 버튼을 클릭하십시오.



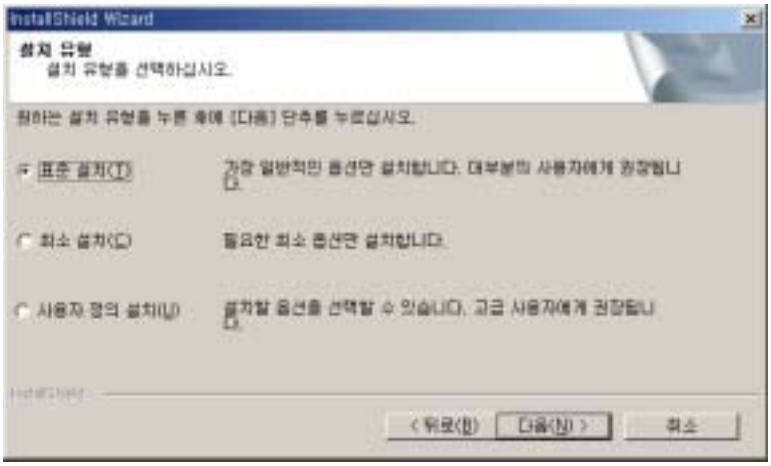
[그림 2-19] 이전에 프로그램이 설치되어 있는 경우의 설치 시작 화면

[그림 2-20]과 같은 화면이 나타나면 고객 정보를 입력한 후 "다음" 버튼을 클릭하십시오.



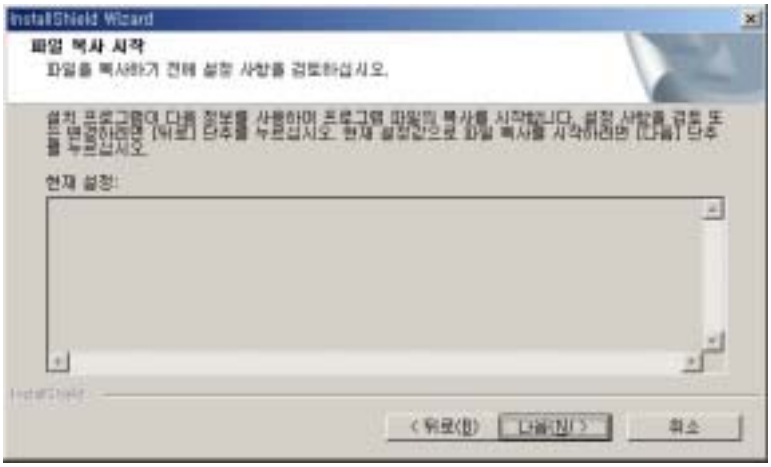
[그림 2-20] 고객 정보 입력 화면

[그림 2-21]과 같은 화면이 나타나면 "표준 설치(T)" 옵션을 선택한 후 "다음" 버튼을 클릭하십시오.



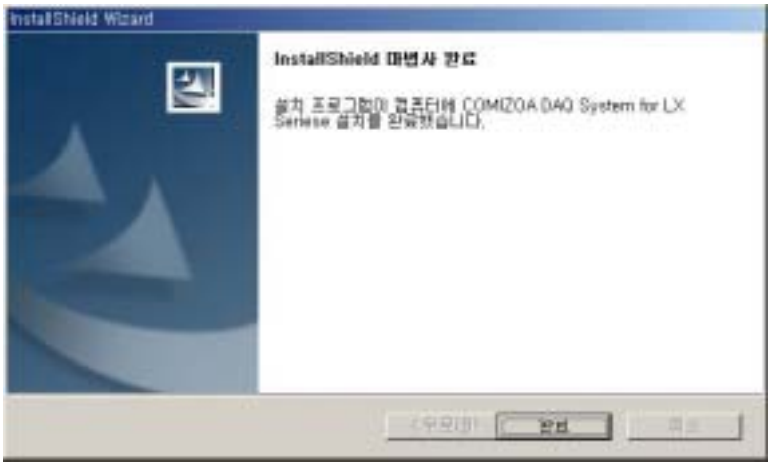
[그림 2-21] 설치 유형 선택 화면

[그림 2-22]와 같은 화면이 나타나면 “다음” 버튼을 클릭하십시오.



[그림 2-22] 현재 설정 확인 화면

[그림 2-22]화면에서 “다음” 버튼을 클릭하면 필요한 파일 복사가 진행됩니다. 프로그램 설치가 완료되면 [그림 2-23]과 같은 화면이 나타납니다. 이상이 없으면 “완료” 버튼을 클릭하여 프로그램 설치를 완료하십시오.



[그림 2-23] 프로그램 설치 완료 화면

프로그램은 사용자가 설치 시에 변경하지 않았다면 기본적으로 C:WProgram FilesWCOMIZOAWCOMIDAS-LX 폴더에 설치됩니다. 각 하위 폴더의 내용은 다음과 같습니다.

▶ COMI-XMaster 유틸리티 프로그램
-. 폴더명 : ROOT 디렉토리에 설치됨
-. 프로그램명 : ComiXMaster.exe
-. 설명 : COMIDAS-LX 시리즈 디바이스의 전반적인 기능을 테스트해보거나 신호 분석을 할 수 있도록 하는 유틸리티 프로그램

▶ 드라이버
-. 폴더명 : Driver
-. 설명 : COMIDAS-LX 시리즈 디바이스의 드라이버 프로그램입니다. 드라이버를 설치할 때 CD 대신 이 폴더에 있는 Comilx.sys 와 Comilx.inf 를 사용할 수 있습니다.

▶ 라이브러리
-. 폴더명 : C_CPPWLibrary
-. 설명 : COMIDAS-LX 시리즈 디바이스의 C/C++ 라이브러리 파일들이 설치되는 폴더입니다.

▶ 예제
-. 폴더명 : C_CPPWExamples

-. 설명 : COMIDAS-LX 시리즈 디바이스의 C/C++ 예제들이 설치되는 폴더입니다.

CHAPTER 3

하드웨어 가이드

본 장에서는 ㈜커미조아에서 개발한 COMI-LX50x Advanced Motion Controller 의 하드웨어적인 사항들을 설명합니다. COMI-LX50x 의 하드웨어적인 상세사양 및 하드웨어 구조, 커넥터 핀 배열 등에 대한 내용을 수록했을 뿐 아니라 각종 모터와의 하드웨어적인 연결 방법에 대한 예 등을 수록하여 사용자가 COMI-LX50x 의 특성 및 사용방법을 쉽게 파악할 수 있도록 하였습니다.

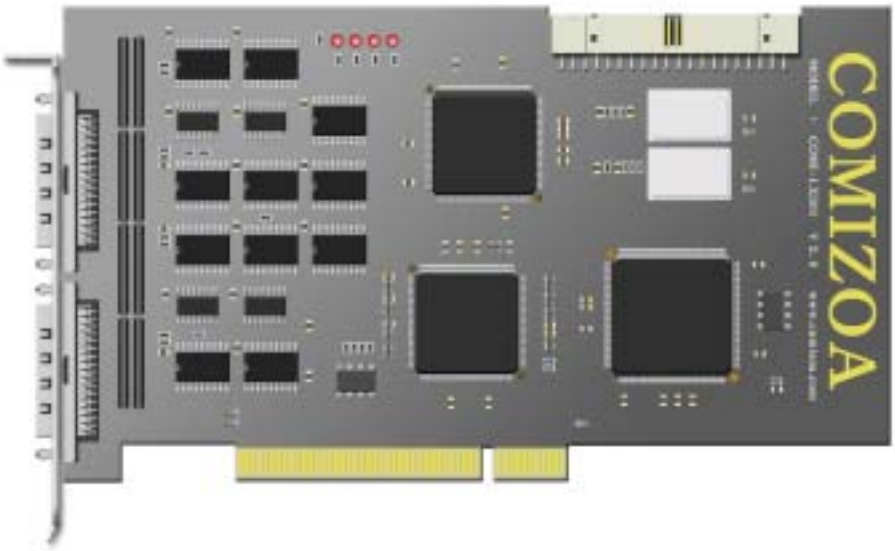
3.1 Motion Controller ····· 30
3.2 Motion Terminal ····· 62
3.3 입/출력 신호 설명 ····· 78
3.4 하드웨어 배선 예 ····· 99

CHAPTER 3 하드웨어 가이드

본 장에서는 ㈜커미조아에서 개발한 COMI-LX50x Advanced Motion Controller 의 하드웨어적인 사항들을 설명합니다. COMI-LX50x 의 하드웨어적인 상세사양 및 하드웨어 구조, 커넥터 핀 배열 등에 대한 내용을 수록했을 뿐 아니라 각종 모터와의 하드웨어적인 연결 방법에 대한 예 등을 수록하여 사용자가 COMI-LX50x 의 특성 및 사용방법을 쉽게 파악할 수 있도록 하였습니다.

3.1 Motion Controller

3.1.1 COMI-LX501 4 축 제어용 Motion Controller



[그림 3-1] COMI-LX501 4 축 모션제어 전용 보드

COMI-LX501 은 PCI 기반 4 축 모션제어 전용 보드로써 6.55MHz 까지의 펄스를 자유자재로 생성하므로써 스텝 모터 또는 서보 모터를 정밀 제어할 수 있도록해주는 장치입니다. COMI-LX501 은 4 축 동기 구동이 가능하며 Trapezoidal 및 S-curve 가감속 속도 프로파일을 지원하며 2 축 원호 보간 및 2~4 축 직선 보간 기능, 그리고 속도의

연속성을 가지는 보간 기능을 제공합니다. 또한 속도나 위치를 구동중에 보정할 수 있도록 되어 있어서 구동 중 환경의 변화에 의해 속도나 위치를 바꿔야하는 상황에 능동적으로 대처하는 것이 가능합니다. 4 축 모두에 제공되는 엔코더 인터페이스는 부정확한 기계적 메커니즘에 의해 발생되는 위치 오차를 교정할 수 있는 기능을 제공하며, 특히 다양하게 제공되는 각종 비교기 기능은 위치의 에러 검출 뿐 아니라 트리거 기능으로 사용될 수 있어서 Machine Vision 장치와 같은 외부장치와의 상호 연계작업을 수행하는데 정확한 타이밍 제어에 유용한 기능을 제공할 수 있습니다. 그밖에 보다 정확하고 안정한 시스템 구축을 지원하기 위하여 다양한 기계적 센서 인터페이스, 서보모터 인터페이스 그리고 범용적으로 사용될 수 있는 I/O 인터페이스를 제공하고 있습니다.

이 제품은 COMI-LX502/4/8 제품과는 달리 COMI-LXT5 또는 COMI-STT5B 터미널보드와 함께 사용되어야 합니다.

1) COMI-LX501 제품 사양

Performance

▷ Number of controllable axes: 4 axes.
▷ Pulse rate setting range : 0.1 PPS ~ 6.55 MPPS
▷ Velocity profile : Linear, Trapezoidal, S-Curve velocity profile drive.
▷ Internal reference clock: 20 MHz
▷ Position pulse setting range: -134,217,728~ +134,217,728 pulses.
▷ Up / down counter counting range: 0~268,435,455 (28-bit) or -134,217,728 to +134,217,727

I/O Signals

▷ Input/Output Signals for each axis
▷ All I/O signal are optically isolated with 2500Vrms isolation voltage
▷ Command pulse output pins: OUT and DIR.
▷ Incremental encoder signals input pins: EA and EB.
▷ Encoder index signal input pin: EZ.
▷ Mechanical limit/switch signal input pins: ±EL, SD/PCS and ORG.
▷ Servomotor interface I/O pins: INP, ALM and ERC.
▷ Position latch input pin: LTC
▷ Pulser signal input pin: PA and PB.

▷ Simultaneous Start/Stop signal I/O pins: STA and STP.

General-Purposed Digital In/Out

▷ 16 TTL level Digital Input
▷ 16 TTL level Digital Output

General Specifications

▷ Connectors: 2 * 68-pin VHDCI-type connector
▷ Operating Temperature: 0°C ~ 50°C
▷ Storage Temperature: -20°C ~ 80°C
▷ Humidity: 5 ~ 85%, non-condensing

Power Consumption:

▷ Slot power supply(input): +5V DC ±5%, 900mA max.
▷ External power supply(input): +24V DC ±5%, 500mA max.
▷ External power supply(output): +5V DC ±5%, 500mA, max.

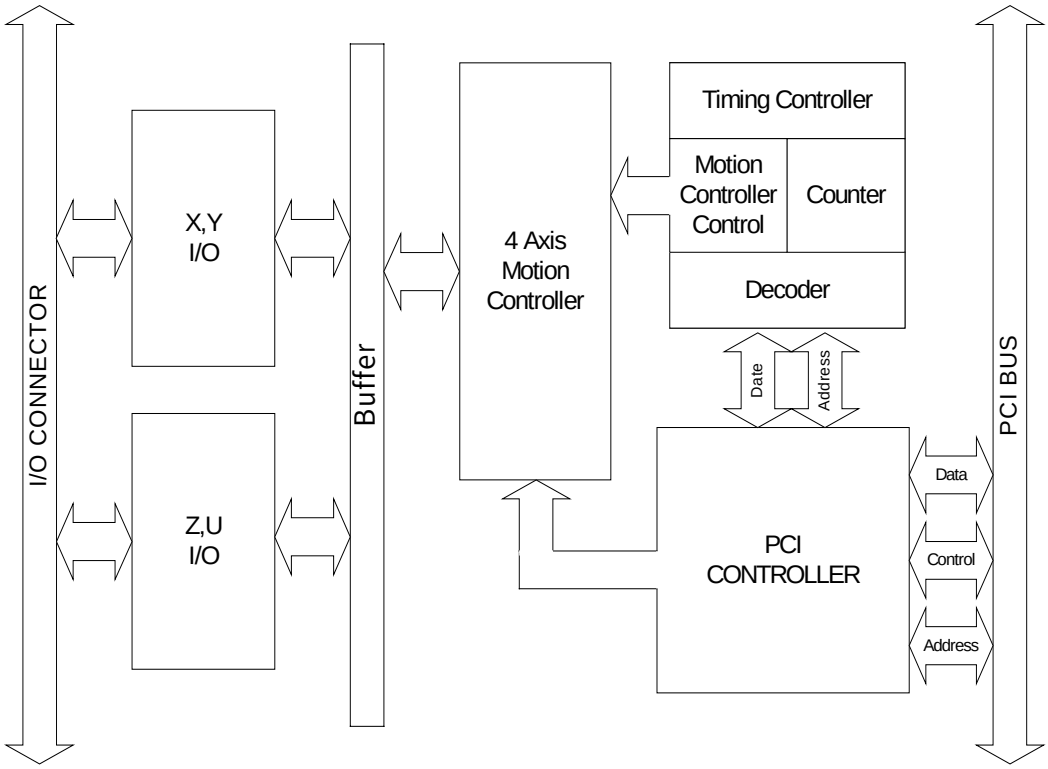
Dimension

▷ 200mm X 114mm

Accessories

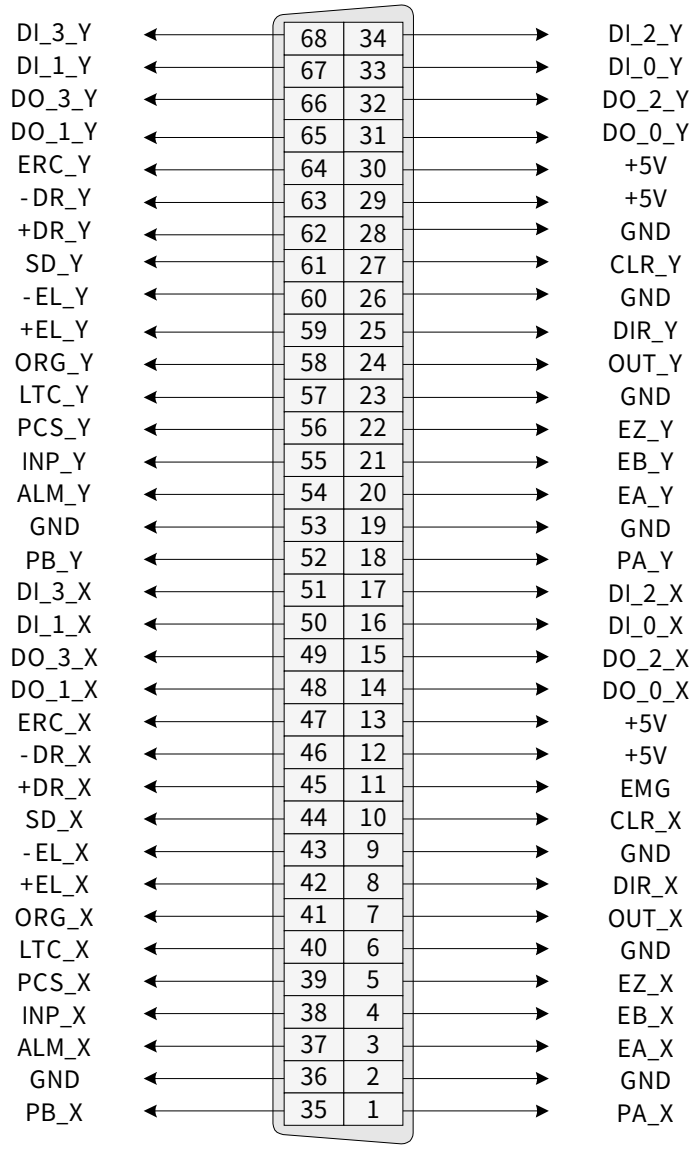
▷ Terminal : COMI-LXT5, COMI-STT5B
▷ Cable : COMI-LXCABLE5, COMI-LXCABLE6

2) COMI-LX501 구조

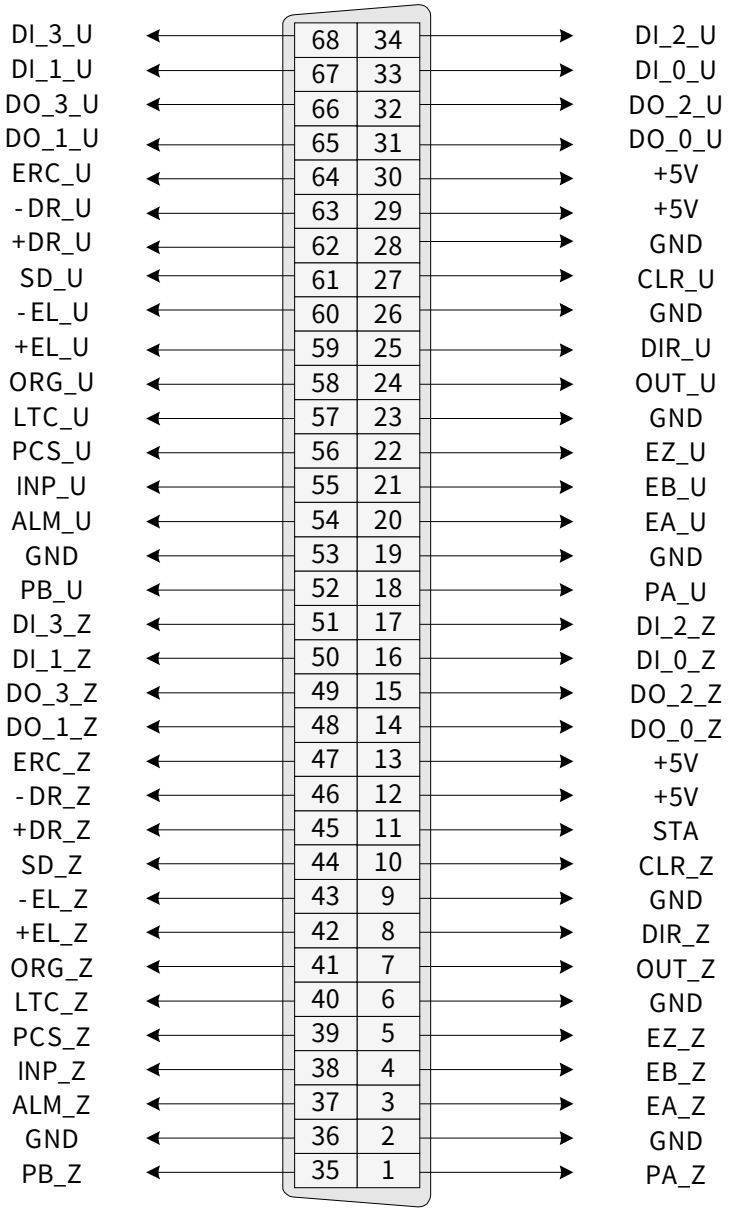


[그림 3-2] COMI-LX501 Hardware 개념도

3) COMI-LX501 커넥터 PIN 배열

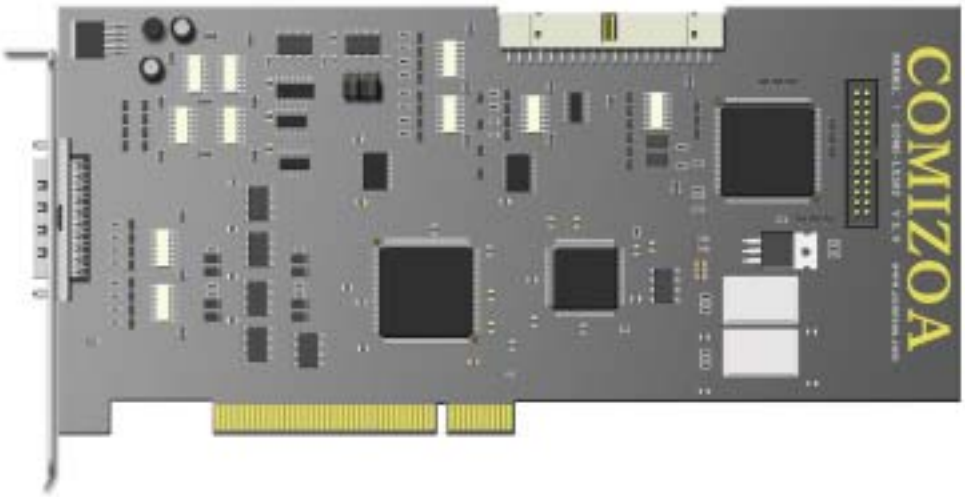


[그림 3-3] COMI-LX501 커넥터 핀 배열 (X-Y 축)



[그림 3-4] COMI-LX501 커넥터 핀 배열 (Z-U 축)

3.1.2 COMI-LX502 2 축 제어용 Motion Controller



[그림 3-5] COMI-LX502 2 축 모션제어 전용 보드

COMI-LX502 는 PCI 기반 2 축 모션제어 전용 보드로써 6.55MHz 까지의 펄스를 자유자재로 생성하므로써 스텝 모터 또는 서보 모터를 정밀 제어할 수 있도록해주는 장치입니다. Trapezoidal 및 S-curve 가감속 속도 프로파일을 지원하며 2 축 원호 보간 및 2 축 직선 보간 기능, 그리고 속도의 연속성을 가지는 보간 기능을 제공합니다. 또한 속도나 위치를 구동중에 보정할 수 있도록 되어 있어서 구동 중 환경의 변화에 의해 속도나 위치를 바꿔야하는 상황에 능동적으로 대처하는 것이 가능합니다. 2 축 모두에 제공되는 엔코더 인터페이스는 부정확한 기계적 메커니즘에 의해 발생되는 위치 오차를 교정할 수 있는 기능을 제공하며, 특히 다양하게 제공되는 각종 비교기 기능은 위치의 에러 검출 뿐 아니라 트리거 기능으로 사용될 수 있어서 Machine Vision 장치와 같은 외부장치와의 상호 연계작업을 수행하는데 정확한 타이밍 제어에 유용한 기능을 제공할 수 있습니다. 그밖에 보다 정확하고 안정한 시스템 구축을 지원하기 위하여 다양한 기계적 센서 인터페이스, 서보모터 인터페이스 그리고 범용적으로 사용될 수 있는 I/O 인터페이스를 제공하고 있습니다.

COMI-LX502 는 보드내에서 모든 신호의 Isolation 처리를 하였기 때문에 터미널보드를 사용하지 않고 시스템을 구성할 수 있습니다.

1) COMI-LX502 제품 사양

Performance

▷ Number of controllable axes: 2 axes.
▷ Pulse rate setting range : 0.1 PPS ~ 6.55 MPPS
▷ Velocity profile : Linear, Trapezoidal, S-Curve velocity profile drive.
▷ Internal reference clock: 20 MHz
▷ Position pulse setting range: -134,217,728~ +134,217,728 pulses.
▷ Up / down counter counting range: 0~268,435,455 (28-bit) or -134,217,728 to +134,217,727

I/O Signals

▷ Input/Output Signals for each axis
▷ All I/O signal are optically isolated with 2500Vrms isolation voltage
▷ Command pulse output pins: OUT and DIR.
▷ Incremental encoder signals input pins: EA and EB.
▷ Encoder index signal input pin: EZ.
▷ Mechanical limit/switch signal input pins: ±EL, SD/PCS and ORG.
▷ Servomotor interface I/O pins: SVON, RDY, INP, ALM and ERC.
▷ Position latch input pin: LTC
▷ Position compare output pin: CMP.
▷ Pulser signal input pin: PA and PB.
▷ Simultaneous Start/Stop signal I/O pins: STA and STP.

General-Purposed Digital In/Out

▷ 3 채널 Isolated Digital Input
▷ 3 채널 Open Collector Digital Output

General Specifications

▷ Connectors: 68-pin VHDCI-type connector
▷ Operating Temperature: 0°C ~ 50°C
▷ Storage Temperature: -20°C ~ 80°C
▷ Humidity: 5 ~ 85%, non-condensing

Power Consumption:

▷ Slot power supply(input): +5V DC ±5%, 900mA max.
▷ External power supply(input): +24V DC ±5%, 500mA max.
▷ External power supply(output): +5V DC ±5%, 500mA, max.

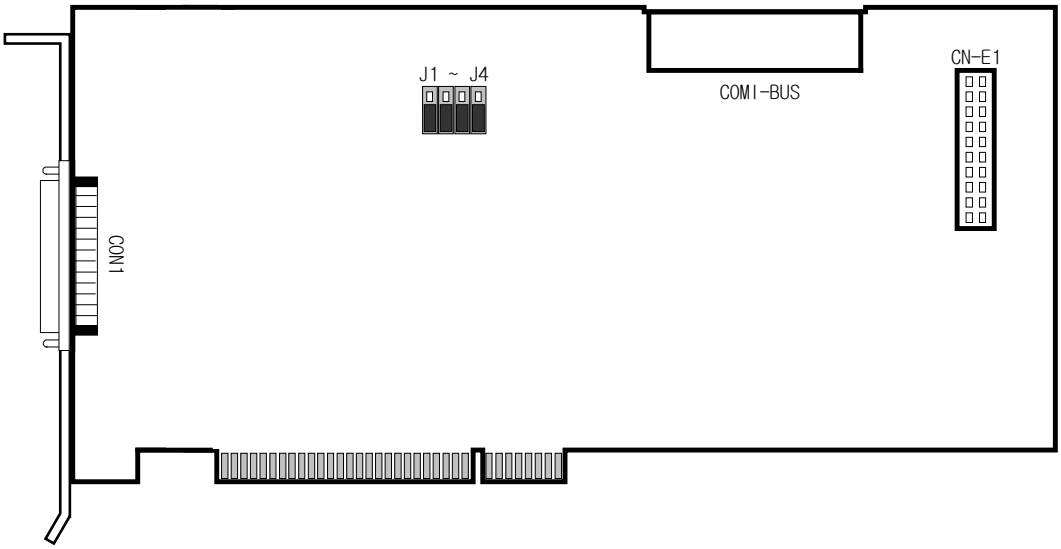
Dimension

▷ 256mm X 114mm

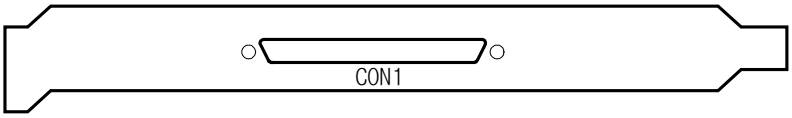
Accessories

▷ Terminal(Optional) : COMI-STT1, COMI-LXT5M, COMI-LXT5S
▷ Cable : COMI-LXCABLE5, COMI-LXCABLE6, COMI-LXCABLE7, COMI-LXCABLE8, COMI-LXCABLE5M, COMI-LXCABLE5S

2) COMI-LX502 구조



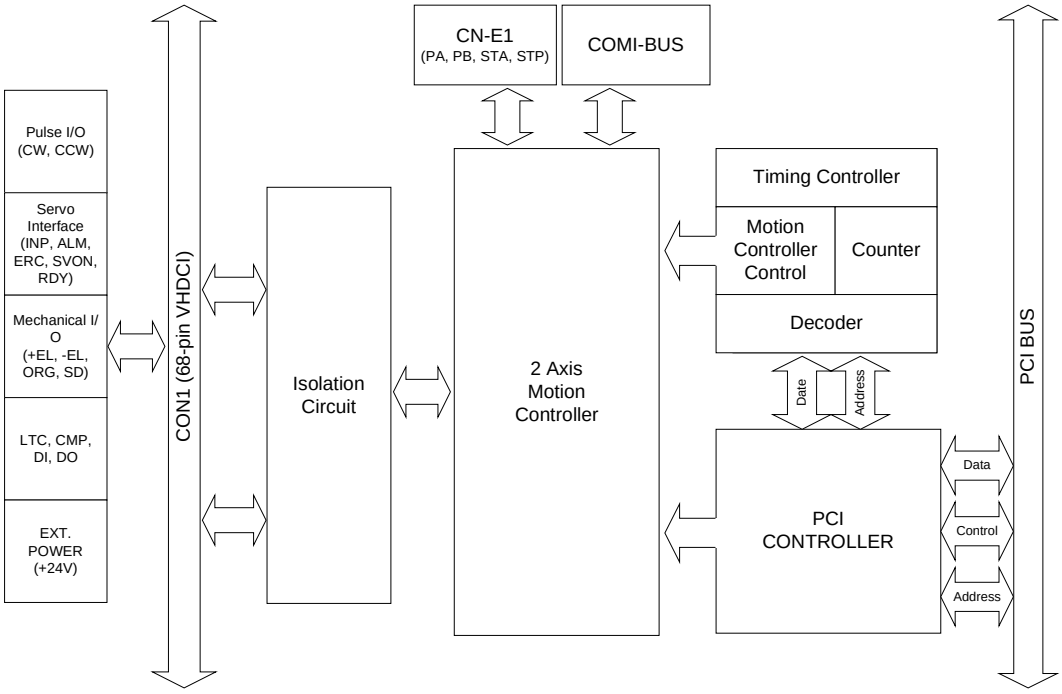
[그림 3-6] COMI-LX502 외관도



[그림 3-7] COMI-LX502 Front Panel

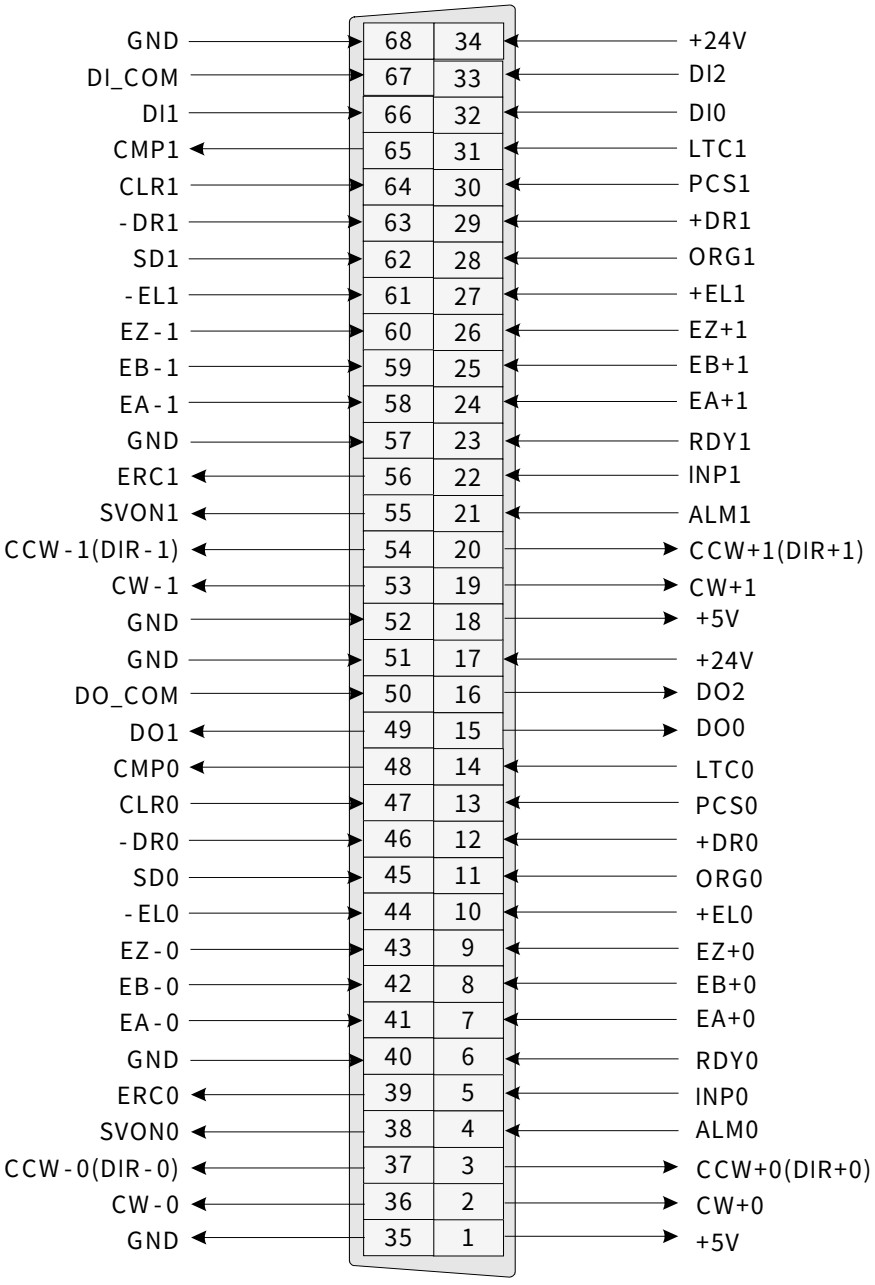
▷ **CON 1** : 68-pin VHDCI Connector, Motion I/O interface for Axis #0, #1
▷ **CN-E1** : 20-pin Connector, PA, PB, STA, STP Input
▷ **J1 ~ J4** : CW, CCW output signal type selection jumper

출력신호	OPEN COLLECTOR	LINE DRIVE (Default)
CW #0	J1 (1,2 번 점퍼연결)	J1 (2,3 번 점퍼연결)
CCW #0	J2 (1,2 번 점퍼연결)	J2 (2,3 번 점퍼연결)
CW #1	J3 (1,2 번 점퍼연결)	J3 (2,3 번 점퍼연결)
CCW #1	J4 (1,2 번 점퍼연결)	J4(2,3 번 점퍼연결)



[그림 3-8] COMI-LX502 Hardware 개념도

3) CON1 68-pin 커넥터 핀 배열

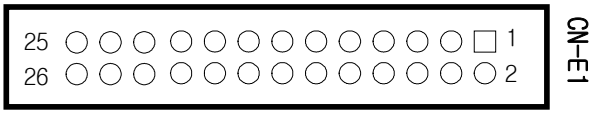


[그림 3-9] CON1 68-pin VHDCI 커넥터 핀 배열

4) CN-E1 커넥터 핀 배열

CN-E1 은 Manual Pulse 입력 신호(PA, PB)와 다축 동기 구동 신호(STA, STP)를 입력할 수 있도록 하는 26-pin 커넥터입니다.

사용자는 일반적인 26 핀 Flat cable 을 이용하거나 ㈜커미조아에서 따로 제공하는 D-Sub 25 핀 커넥터(Front panel 포함)를 사용하여 신호연결을 하실 수 있습니다.

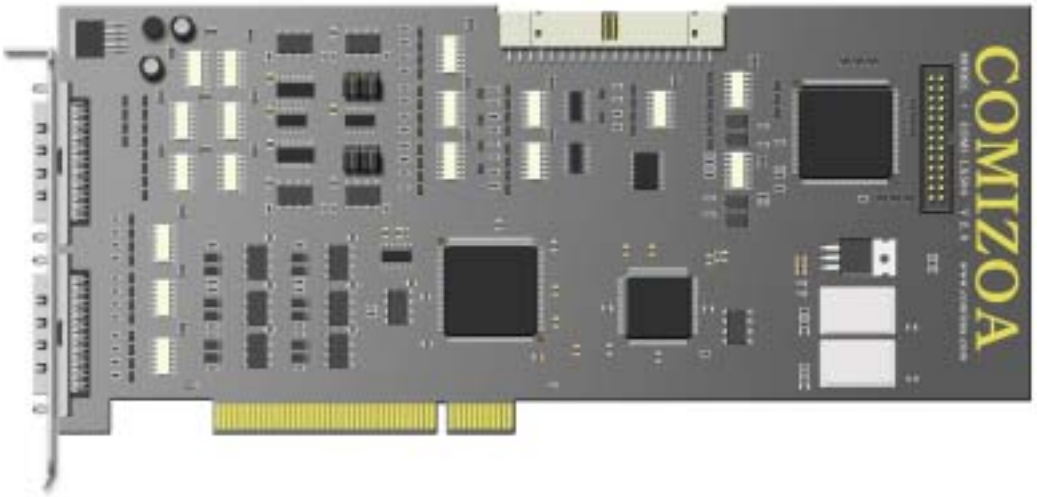


[그림 3-10] CN-E1 26-pin connector

Pin No.	Pin Name	Pin No.	Pin Name
1	PA0	2	GND
3	PB0	4	GND
5	PA1	6	GND
7	PB1	8	GND
9	NC	10	GND
11	NC	12	GND
13	NC	14	GND
15	NC	16	GND
17	STA	18	GND
19	STP	20	GND
21	+5V	22	GND
23	+5V	24	GND
25	+5V	26	NC

[표 3-1] CN-E1 26-pin connector pin description

3.1.3 COMI-LX504 4 축 제어용 Motion Controller



[그림 3-11] COMI-LX504 4 축 모션제어 전용 보드

COMI-LX504 는 PCI 기반 4 축 모션제어 전용 보드로써 6.55MHz 까지의 펄스를 자유자재로 생성하므로써 스텝 모터 또는 서보 모터를 정밀 제어할 수 있도록해주는 장치입니다. COMI-LX504 는 4 축 동기 구동이 가능하며, Trapezoidal 및 S-curve 가감속 속도 프로파일을 지원하며 2 축 원호 보간 및 4 축 직선 보간 기능, 그리고 속도의 연속성을 가지는 보간 기능을 제공합니다. 또한 속도나 위치를 구동중에 보정할 수 있도록 되어 있어서 구동 중 환경의 변화에 의해 속도나 위치를 바꿔야하는 상황에 능동적으로 대처하는 것이 가능합니다. 2 축 모두에 제공되는 엔코더 인터페이스는 부정확한 기계적 메커니즘에 의해 발생되는 위치 오차를 교정할 수 있는 기능을 제공하며, 특히 다양하게 제공되는 각종 비교기 기능은 위치의 에러 검출 뿐 아니라 트리거 기능으로 사용될 수 있어서 Machine Vision 장치와 같은 외부장치와의 상호 연계작업을 수행하는데 정확한 타이밍 제어에 유용한 기능을 제공할 수 있습니다. 그밖에 보다 정확하고 안정한 시스템 구축을 지원하기 위하여 다양한 기계적 센서 인터페이스, 서보모터 인터페이스 그리고 범용적으로 사용될 수 있는 I/O 인터페이스를 제공하고 있습니다.

COMI-LX504 는 보드내에서 모든 신호의 Isolation 처리를 하였기 때문에 터미널보드를 사용하지 않고 시스템을 구성할 수 있습니다.

1) COMI-LX504 제품 사양

Performance

▷ Number of controllable axes: 4 axes.
▷ Pulse rate setting range : 0.1 PPS ~ 6.55 MPPS
▷ Velocity profile : Linear, Trapezoidal, S-Curve velocity profile drive.
▷ Internal reference clock: 20 MHz
▷ Position pulse setting range: -134,217,728~ +134,217,727 pulses.
▷ Up / down counter counting range: 0~268,435,455 (28-bit) or -134,217,728 to +134,217,727

I/O Signals

▷ Input/Output Signals for each axis
▷ All I/O signal are optically isolated with 2500Vrms isolation voltage
▷ Command pulse output pins: OUT and DIR.
▷ Incremental encoder signals input pins: EA and EB.
▷ Encoder index signal input pin: EZ.
▷ Mechanical limit/switch signal input pins: ±EL, SD/PCS and ORG.
▷ Servomotor interface I/O pins: SVON, RDY, INP, ALM and ERC.
▷ Position latch input pin: LTC
▷ Position compare output pin: CMP.
▷ Pulser signal input pin: PA and PB.
▷ Simultaneous Start/Stop signal I/O pins: STA and STP.

General-Purposed Digital In/Out

▷ 6 채널 Isolated Digital Input
▷ 6 채널 Open Collector Digital Output

General Specifications

▷ Connectors: 2 * 68-pin VHDCI-type connector
▷ Operating Temperature: 0°C ~ 50°C
▷ Storage Temperature: -20°C ~ 80°C
▷ Humidity: 5 ~ 85%, non-condensing

Power Consumption:

▷ Slot power supply(input): +5V DC ±5%, 900mA max.
▷ External power supply(input): +24V DC ±5%, 500mA max.
▷ External power supply(output): +5V DC ±5%, 500mA, max.

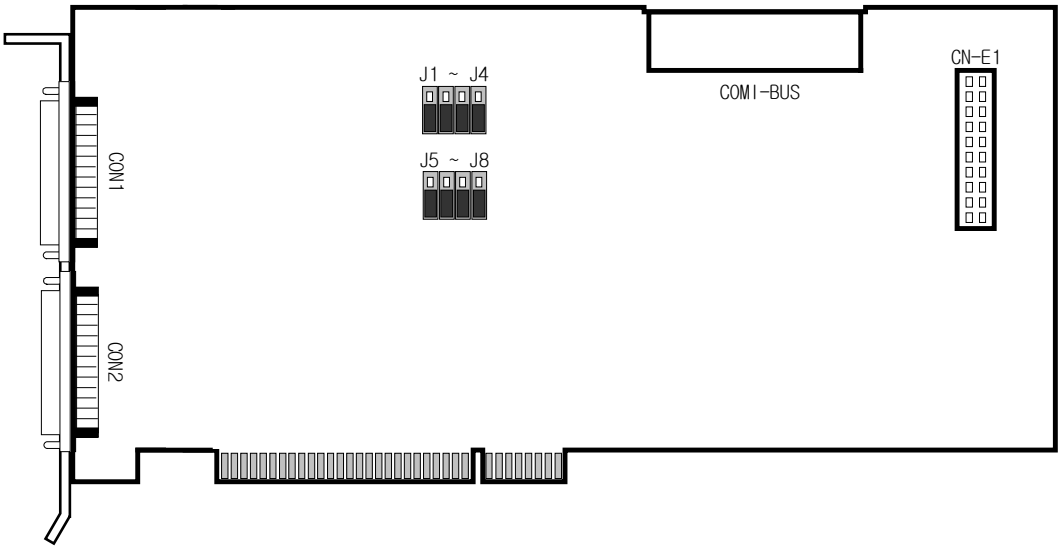
Dimension

▷ 256mm X 114mm

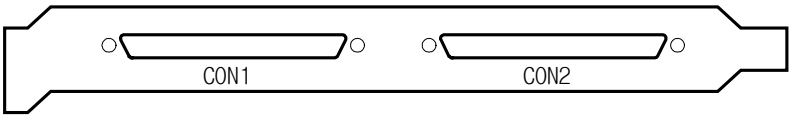
Accessories

▷ Terminal(Optional) : COMI-STT1, COMI-LXT5M, COMI-LXT5S
▷ Cable : COMI-LXCABLE5, COMI-LXCABLE6, COMI-LXCABLE7, COMI-LXCABLE8, COMI-LXCABLE5M, COMI-LXCABLE5S

2) COMI-LX504 구조



[그림 3-12] COMI-LX504 외관도



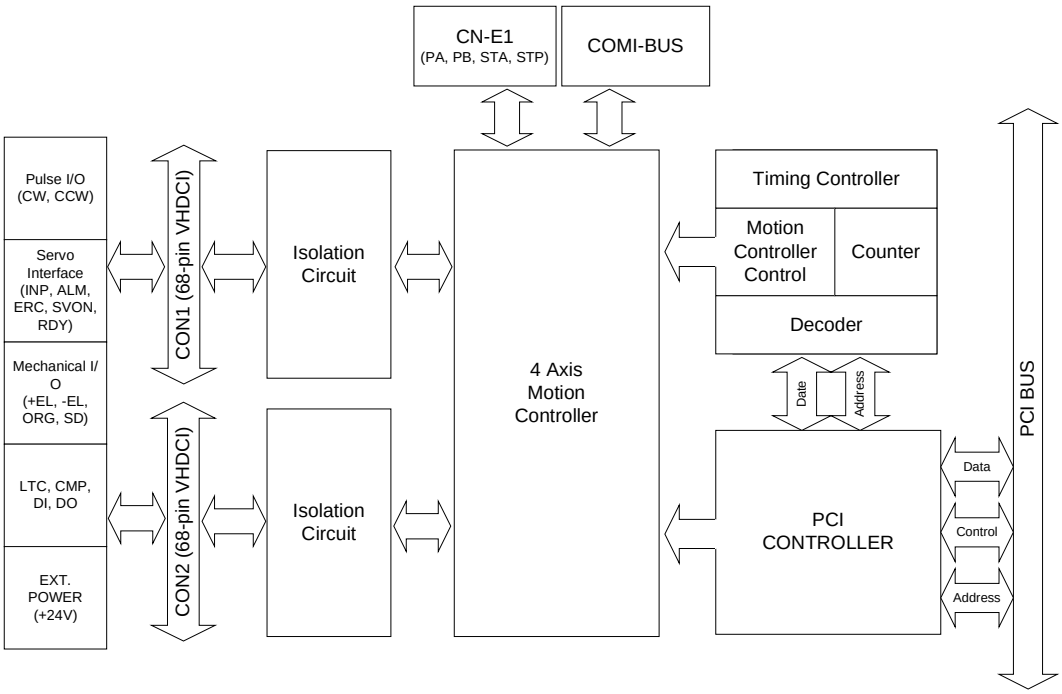
[그림 3-13] COMI-LX504 Front Panel

▷ **CON 1** : 68-pin VHDCI Connector, Motion I/O interface for Axis #0, #1
▷ **CON 2** : 68-pin VHDCI Connector, Motion I/O interface for Axis #2, #3
▷ **CN-E1** : 20-pin Connector, PA, PB, STA, STP Input
▷ **J1 ~ J4** : CW, CCW output signal type selection jumper (Axis #0, #1)

출력신호	OPEN COLLECTOR	LINE DRIVE (Default)
CW #0	J1 (1,2 번 점퍼연결)	J1 (2,3 번 점퍼연결)
CCW #0	J2 (1,2 번 점퍼연결)	J2 (2,3 번 점퍼연결)
CW #1	J3 (1,2 번 점퍼연결)	J3 (2,3 번 점퍼연결)
CCW #1	J4 (1,2 번 점퍼연결)	J4(2,3 번 점퍼연결)

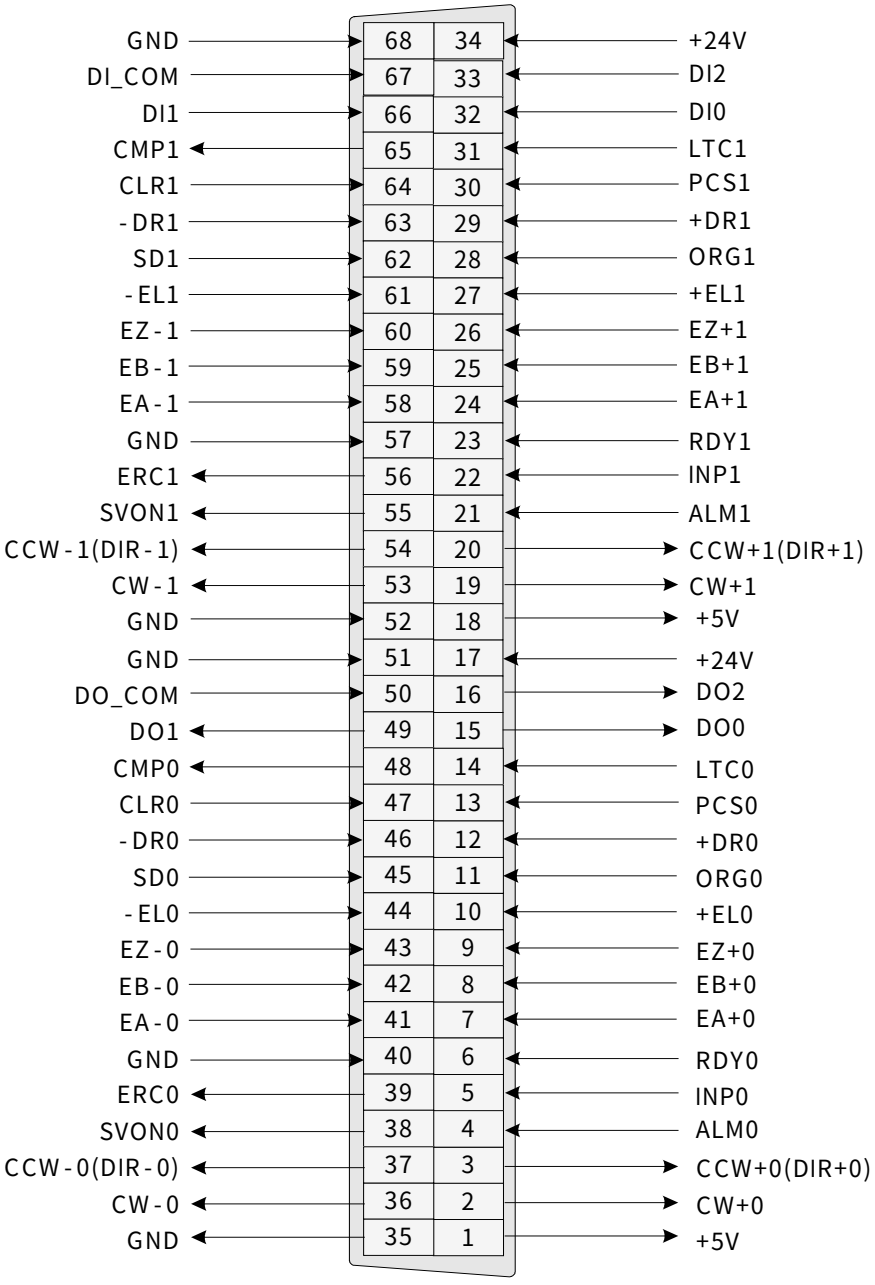
▷ J5 ~ J8 : CW, CCW output signal type selection jumper (Axis #3, #4)

출력신호	OPEN COLLECTOR	LINE DRIVE (Default)
CW #2	J5 (1,2 번 점퍼연결)	J5 (2,3 번 점퍼연결)
CCW #2	J6 (1,2 번 점퍼연결)	J6 (2,3 번 점퍼연결)
CW #3	J7 (1,2 번 점퍼연결)	J7 (2,3 번 점퍼연결)
CCW #3	J8 (1,2 번 점퍼연결)	J8(2,3 번 점퍼연결)

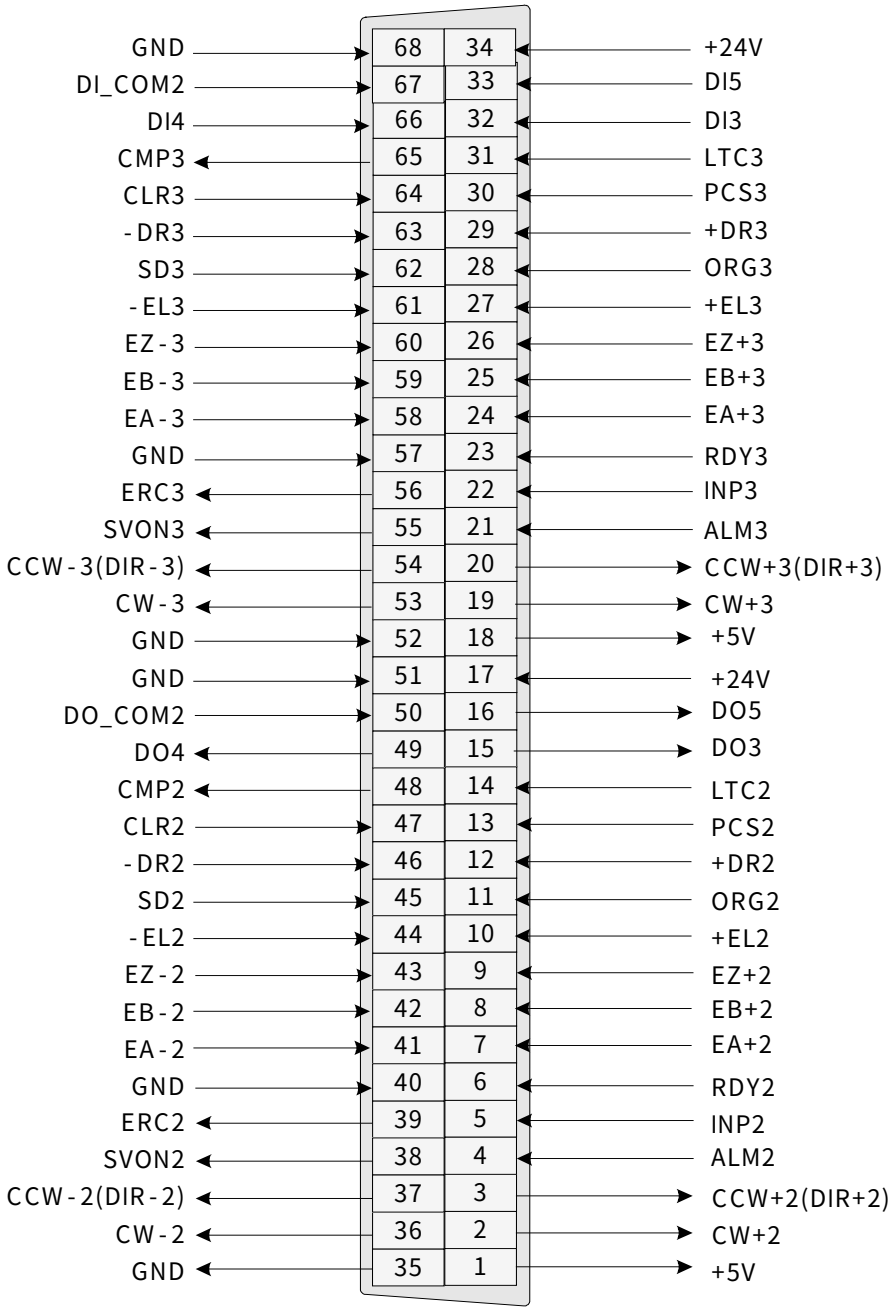


[그림 3-14] COMI-LX504 Hardware 개념도

3) CON1,2 68-pin 커넥터 핀 배열



[그림 3-15] CON1 68-pin VHDCI 커넥터 핀 배열

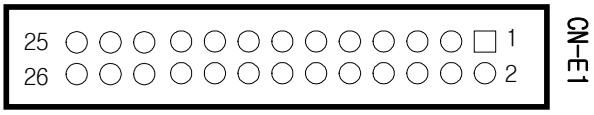


[그림 3-16] CON2 68-pin VHDCI 커넥터 핀 배열

4) CN-E1 커넥터 핀 배열

CN-E1 은 Manual Pulse 입력 신호(PA, PB)와 다축 동기 구동 신호(STA, STP)를 입력할 수 있도록 하는 26-pin 커넥터입니다.

사용자는 일반적인 26 핀 Flat cable 을 이용하거나 ㈜커미조아에서 따로 제공하는 D-Sub 25 핀 커넥터(Front panel 포함)를 사용하여 신호연결을 하실 수 있습니다.



[그림 3-17] CN-E1 26-pin connector

Pin No.	Pin Name	Pin No.	Pin Name
1	PA0	2	GND
3	PB0	4	GND
5	PA1	6	GND
7	PB1	8	GND
9	PA2	10	GND
11	PB2	12	GND
13	PA3	14	GND
15	PB3	16	GND
17	STA	18	GND
19	STP	20	GND
21	+5V	22	GND
23	+5V	24	GND
25	+5V	26	NC

[표 3-2] CN-E1 26-pin connector pin description

3.1.4 COMI-LX508 8 축 제어용 Motion Controller

[그림 3-18] COMI-LX508 8 축 모션제어 전용 보드

COMI-LX508 은 PCI 기반 8 축 모션제어 전용 보드로써 6.55MHz 까지의 펄스를 자유자재로 생성하므로써 스텝 모터 또는 서보 모터를 정밀 제어할 수 있도록해주는 장치입니다. COMI-LX508 은 8 축 동기 구동이 가능하며, Trapezoidal 및 S-curve 가감속 속도 프로파일을 지원하며 2 축 원호 보간 및 4 축 직선 보간 기능, 그리고 속도의 연속성을 가지는 보간 기능을 제공합니다. 또한 속도나 위치를 구동중에 보정할 수 있도록 되어 있어서 구동 중 환경의 변화에 의해 속도나 위치를 바꿔야하는 상황에 능동적으로 대처하는 것이 가능합니다. 2 축 모두에 제공되는 엔코더 인터페이스는 부정확한 기계적 메커니즘에 의해 발생되는 위치 오차를 교정할 수 있는 기능을 제공하며, 특히 다양하게 제공되는 각종 비교기 기능은 위치의 에러 검출 뿐 아니라 트리거 기능으로 사용될 수 있어서 Machine Vision 장치와 같은 외부장치와의 상호 연계작업을 수행하는데 정확한 타이밍 제어에 유용한 기능을 제공할 수 있습니다. 그밖에 보다 정확하고 안정한 시스템 구축을 지원하기 위하여 다양한 기계적 센서 인터페이스, 서보모터 인터페이스 그리고 범용적으로 사용될 수 있는 I/O 인터페이스를 제공하고 있습니다.

COMI-LX508 은 보드내에서 모든 신호의 Isolation 처리를 하였기 때문에 터미널보드를 사용하지 않고 시스템을 구성할 수 있습니다.

1) COMI-LX508 제품 사양

Performance

▷ Number of controllable axes: 8 axes.
▷ Pulse rate setting range : 0.1 PPS ~ 6.55 MPPS
▷ Velocity profile : Linear, Trapezoidal, S-Curve velocity profile drive.
▷ Internal reference clock: 20 MHz
▷ Position pulse setting range: -134,217,728~ +134,217,728 pulses.
▷ Up / down counter counting range: 0~268,435,455 (28-bit) or -134,217,728 to +134,217,727

I/O Signals

▷ Input/Output Signals for each axis
▷ All I/O signal are optically isolated with 2500Vrms isolation voltage
▷ Command pulse output pins: OUT and DIR.
▷ Incremental encoder signals input pins: EA and EB.
▷ Encoder index signal input pin: EZ.
▷ Mechanical limit/switch signal input pins: ±EL, SD/PCS and ORG.
▷ Servomotor interface I/O pins: SVON, RDY, INP, ALM and ERC.
▷ Position latch input pin: LTC
▷ Position compare output pin: CMP.
▷ Pulser signal input pin: PA and PB.
▷ Simultaneous Start/Stop signal I/O pins: STA and STP.

General-Purposed Digital In/Out

▷ 12 채널 Isolated Digital Input
▷ 12 채널 Open Collector Digital Output

General Specifications

▷ Connectors: 4 * 68-pin VHDCI-type connector
▷ Operating Temperature: 0°C ~ 50°C
▷ Storage Temperature: -20°C ~ 80°C
▷ Humidity: 5 ~ 85%, non-condensing

Power Consumption:

▷ Slot power supply(input): +5V DC ±5%, 900mA max.
▷ External power supply(input): +24V DC ±5%, 500mA max.
▷ External power supply(output): +5V DC ±5%, 500mA, max.

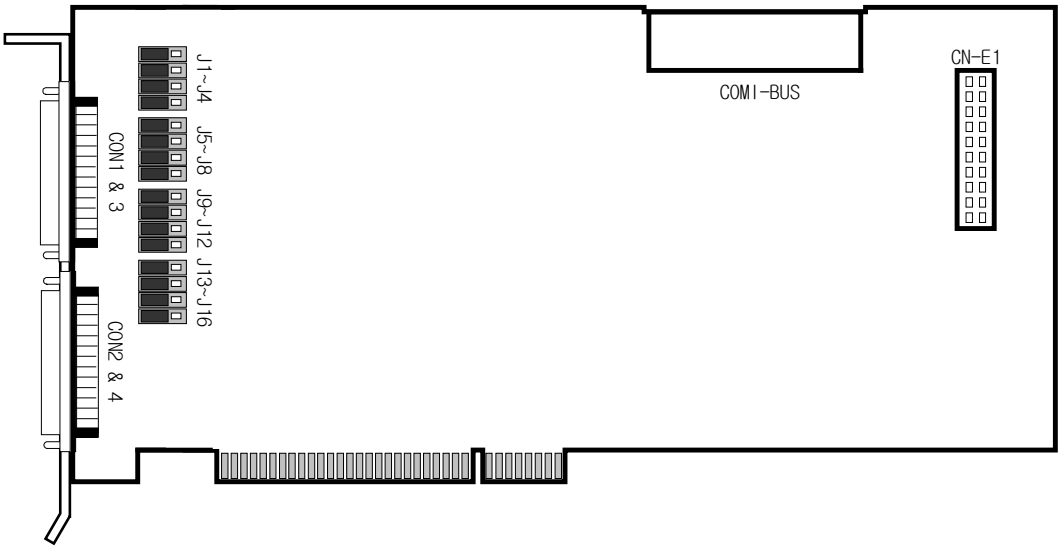
Dimension

▷ 256mm X 114mm

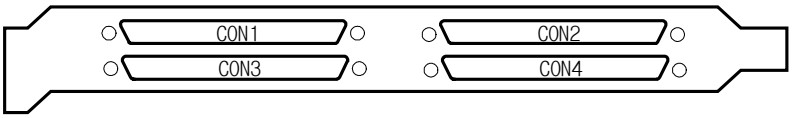
Accessories

▷ Terminal(Optional) : COMI-STT1, COMI-LXT5M, COMI-LXT5S
▷ Cable : COMI-LXCABLE5, COMI-LXCABLE6, COMI-LXCABLE7, COMI-LXCABLE8, COMI-LXCABLE5M, COMI-LXCABLE5S

2) COMI-LX508 구조



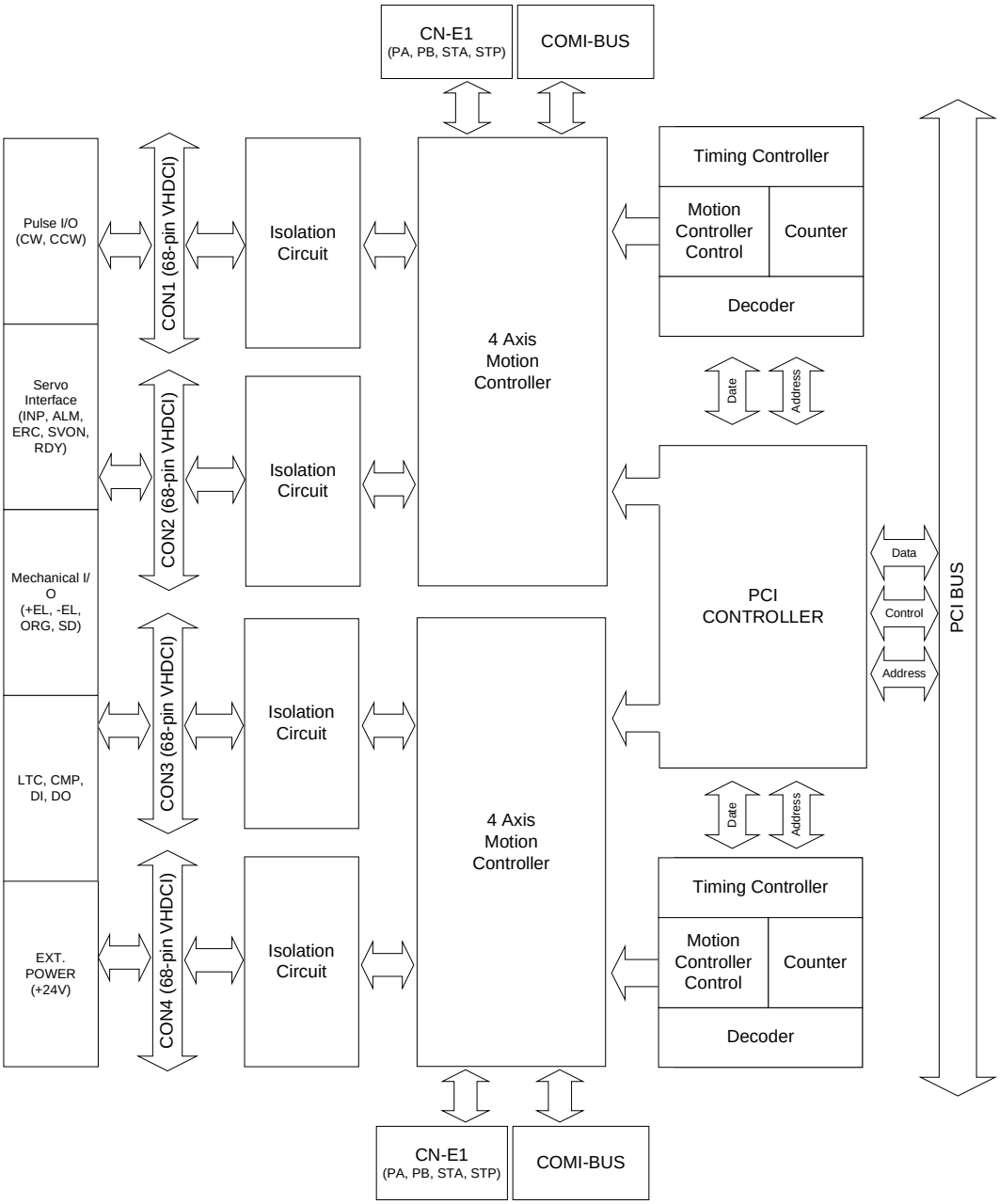
[그림 3-19] COMI-LX504 외관도



[그림 3-20] COMI-LX504 Front Panel

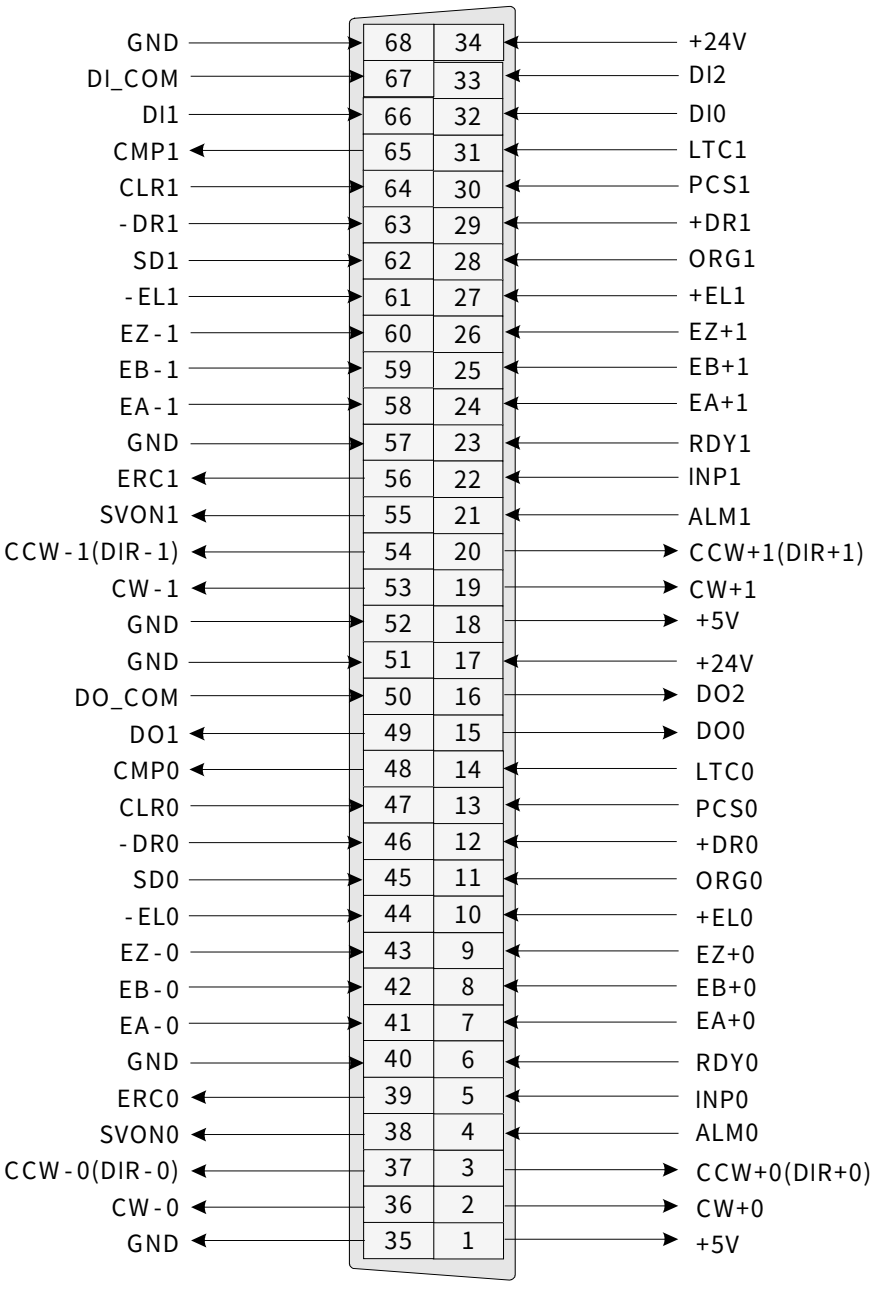
▷ **CON 1** : 68-pin VHDCI Connector, Motion I/O interface for Axis #0, #1
▷ **CON 2** : 68-pin VHDCI Connector, Motion I/O interface for Axis #2, #3
▷ **CON 3** : 68-pin VHDCI Connector, Motion I/O interface for Axis #4, #5
▷ **CON 4** : 68-pin VHDCI Connector, Motion I/O interface for Axis #6, #7
▷ **CN-E1** : 20-pin Connector, PA, PB, STA, STP Input
▷ **J1 ~ J16** : CW, CCW output signal type selection jumper for 8 Axes

출력신호	OPEN COLLECTOR	LINE DRIVE (Default)
CW #0	J1 (1,2 번 점퍼연결)	J1 (2,3 번 점퍼연결)
CCW #0	J2 (1,2 번 점퍼연결)	J2 (2,3 번 점퍼연결)
CW #1	J3 (1,2 번 점퍼연결)	J3 (2,3 번 점퍼연결)
CCW #1	J4 (1,2 번 점퍼연결)	J4(2,3 번 점퍼연결)
CW #2	J5 (1,2 번 점퍼연결)	J5 (2,3 번 점퍼연결)
CCW #2	J6 (1,2 번 점퍼연결)	J6 (2,3 번 점퍼연결)
CW #3	J7 (1,2 번 점퍼연결)	J7 (2,3 번 점퍼연결)
CCW #3	J8 (1,2 번 점퍼연결)	J8(2,3 번 점퍼연결)
CW #4	J9 (1,2 번 점퍼연결)	J9 (2,3 번 점퍼연결)
CCW #4	J10 (1,2 번 점퍼연결)	J10 (2,3 번 점퍼연결)
CW #5	J11 (1,2 번 점퍼연결)	J11 (2,3 번 점퍼연결)
CCW #5	J12 (1,2 번 점퍼연결)	J12(2,3 번 점퍼연결)
CW #6	J13 (1,2 번 점퍼연결)	J13 (2,3 번 점퍼연결)
CCW #6	J14 (1,2 번 점퍼연결)	J14 (2,3 번 점퍼연결)
CW #7	J15 (1,2 번 점퍼연결)	J15 (2,3 번 점퍼연결)
CCW #7	J16 (1,2 번 점퍼연결)	J16(2,3 번 점퍼연결)

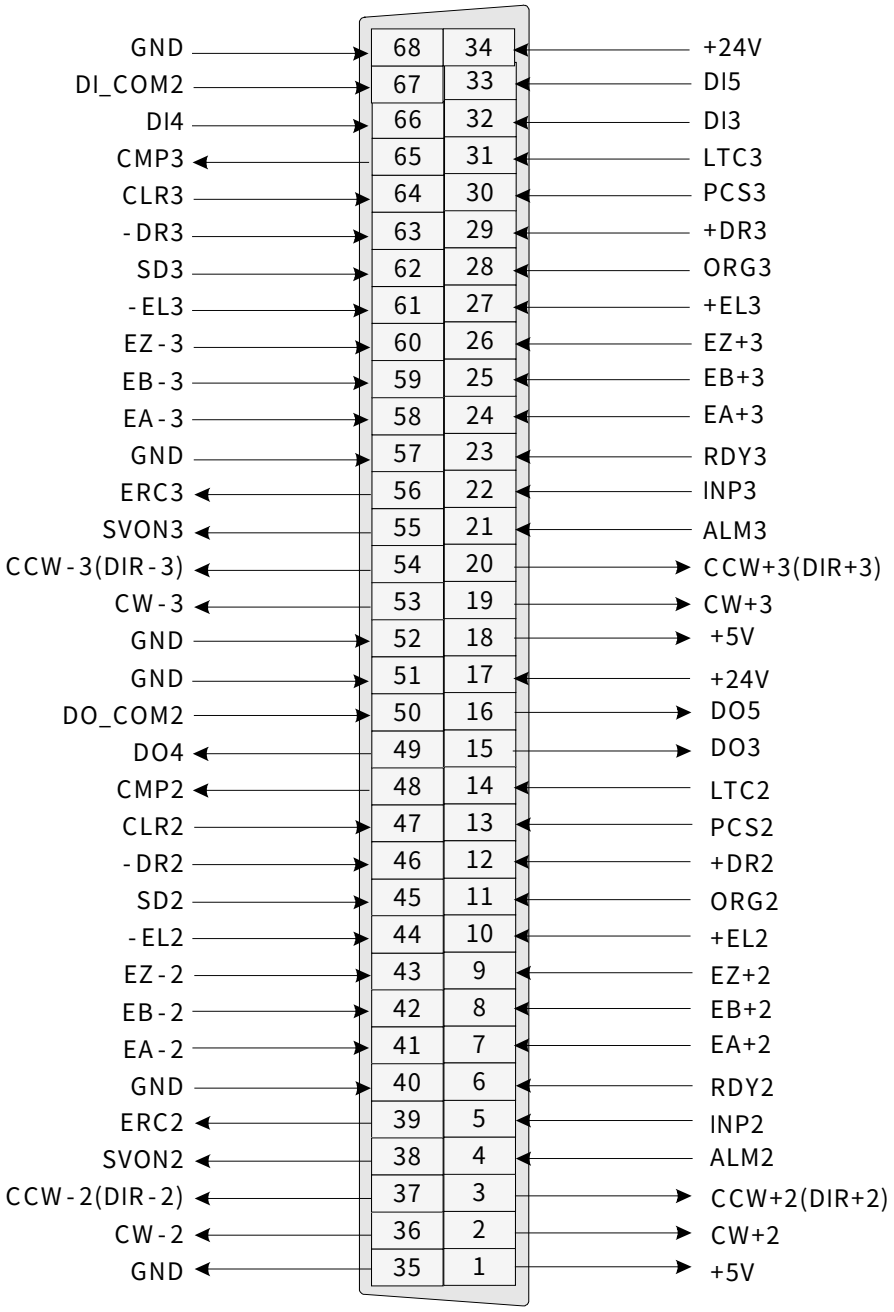


[그림 3-21] COMI-LX508 Hardware 개념도

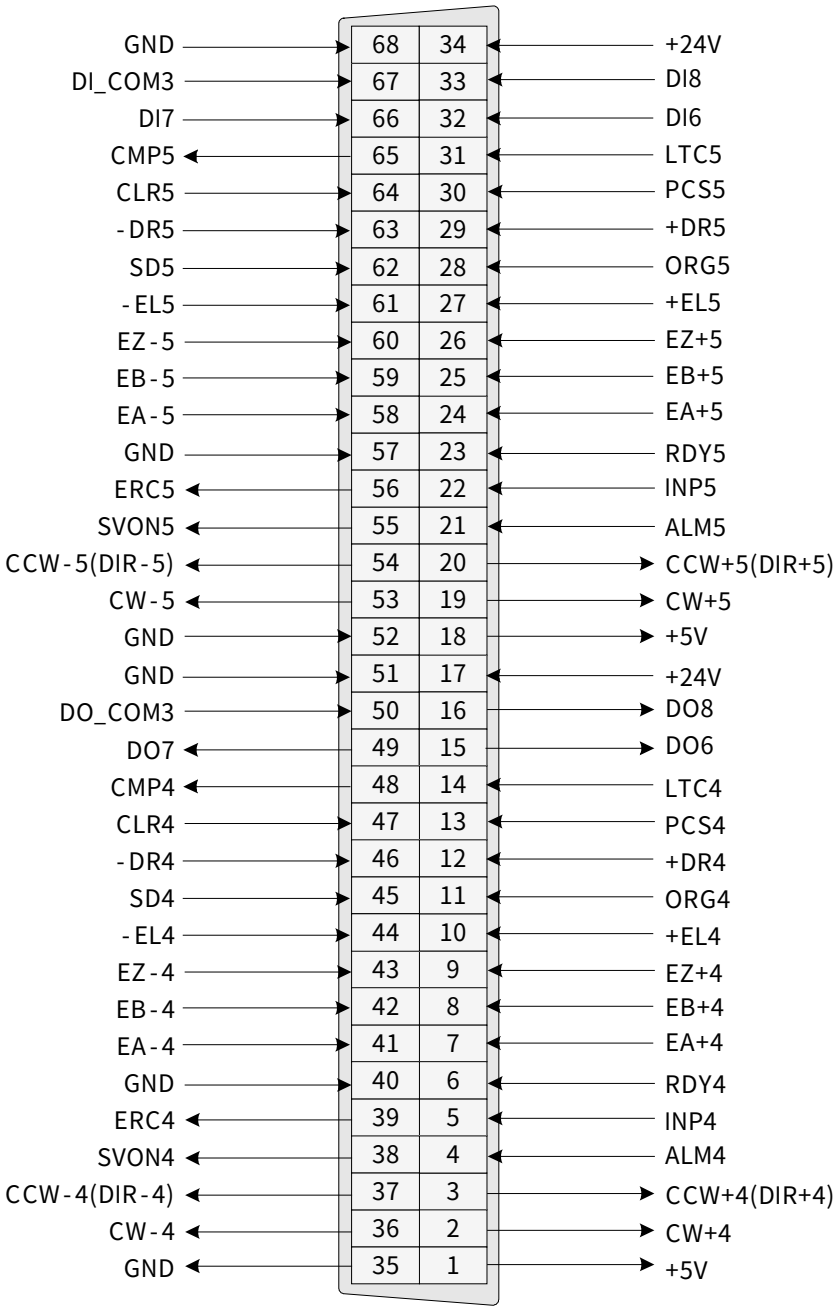
3) COMI-LX508 커넥터 PIN 배열



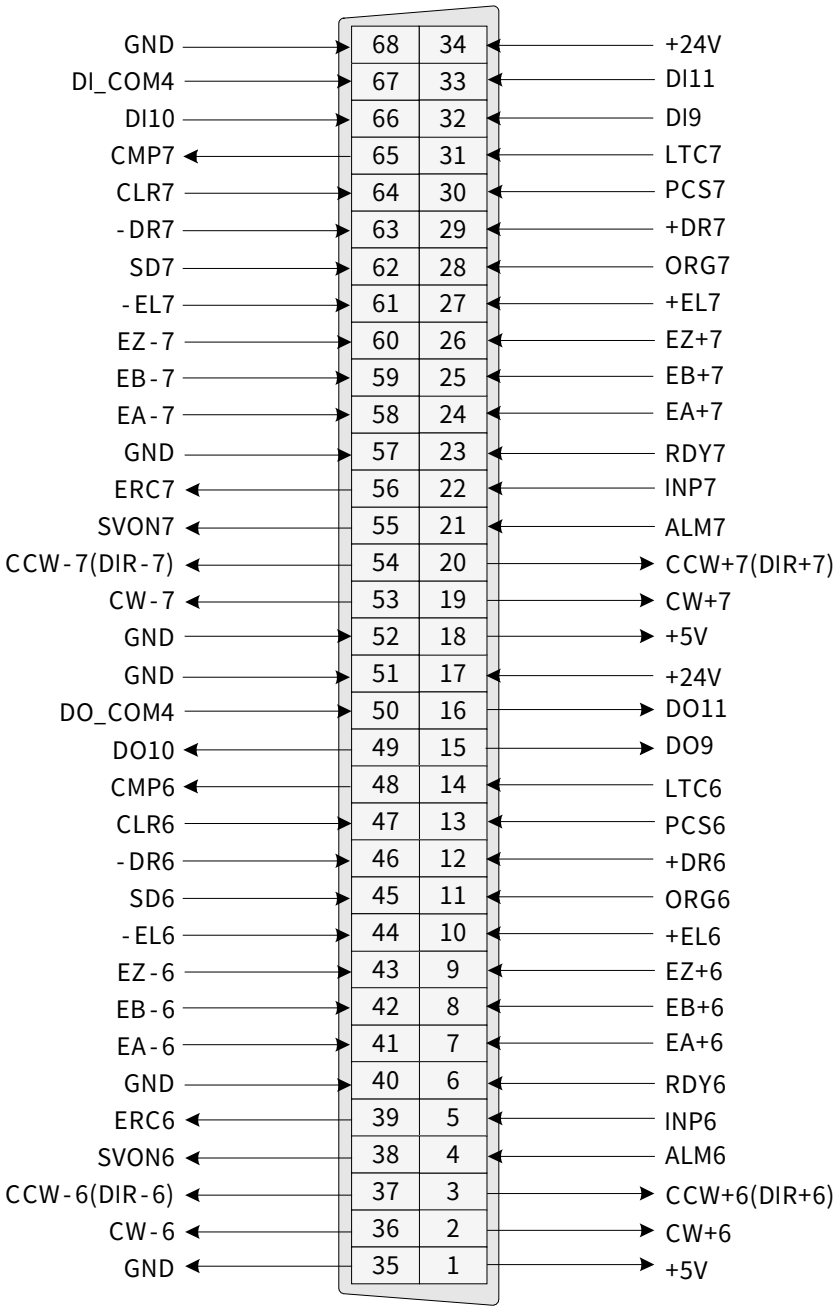
[그림 3-22] CON1 68-pin VHDCI 커넥터 핀 배열



[그림 3-23] CON2 68-pin VHDCI 커넥터 핀 배열



[그림 3-24] CON3 68-pin VHDCI 커넥터 핀 배열

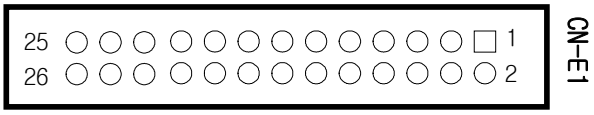


[그림 3-25] CON4 68-pin VHDCI 커넥터 핀 배열

4) CN-E1 커넥터 핀 배열

CN-E1 은 Manual Pulse 입력 신호(PA, PB)와 다축 동기 구동 신호(STA, STP)를 입력할 수 있도록 하는 26-pin 커넥터입니다.

사용자는 일반적인 26 핀 Flat cable 을 이용하거나 ㈜커미조아에서 따로 제공하는 D-Sub 25 핀 커넥터(Front panel 포함)를 사용하여 신호연결을 하실 수 있습니다.



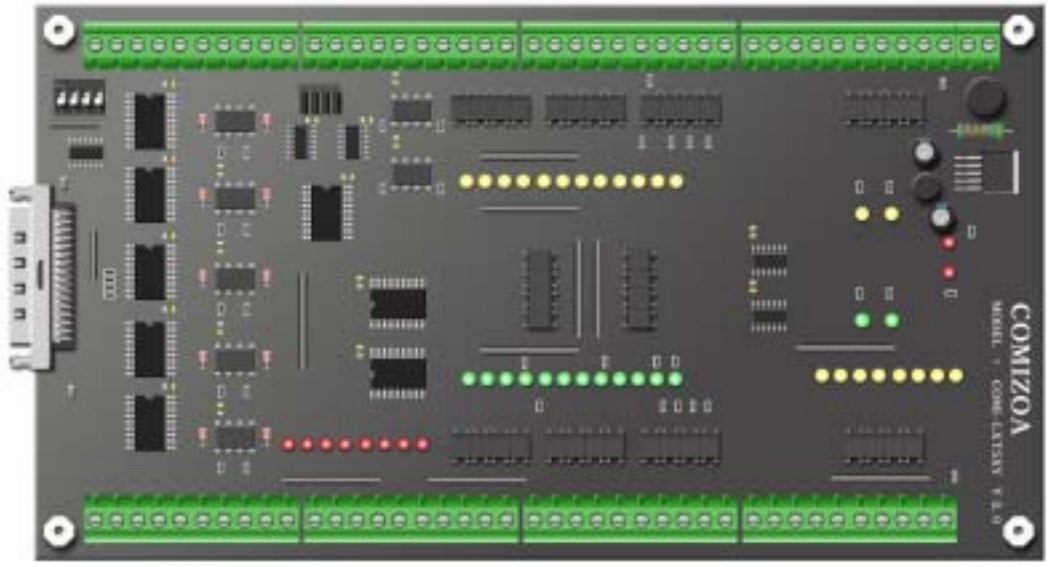
[그림 3-26] CN-E1 26-pin connector

Pin No.	Pin Name	Pin No.	Pin Name
1	PA0	2	PA4
3	PB0	4	PB4
5	PA1	6	PA5
7	PB1	8	PB5
9	PA2	10	PA6
11	PB2	12	PB6
13	PA3	14	PA7
15	PB3	16	PB7
17	STA	18	GND
19	STP	20	GND
21	+5V	22	GND
23	+5V	24	GND
25	+5V	26	NC

[표 3-3] CN-E1 26-pin connector pin description

3.2 Motion Terminal

3.2.1 COMI-LXT5 Motion Terminal Board

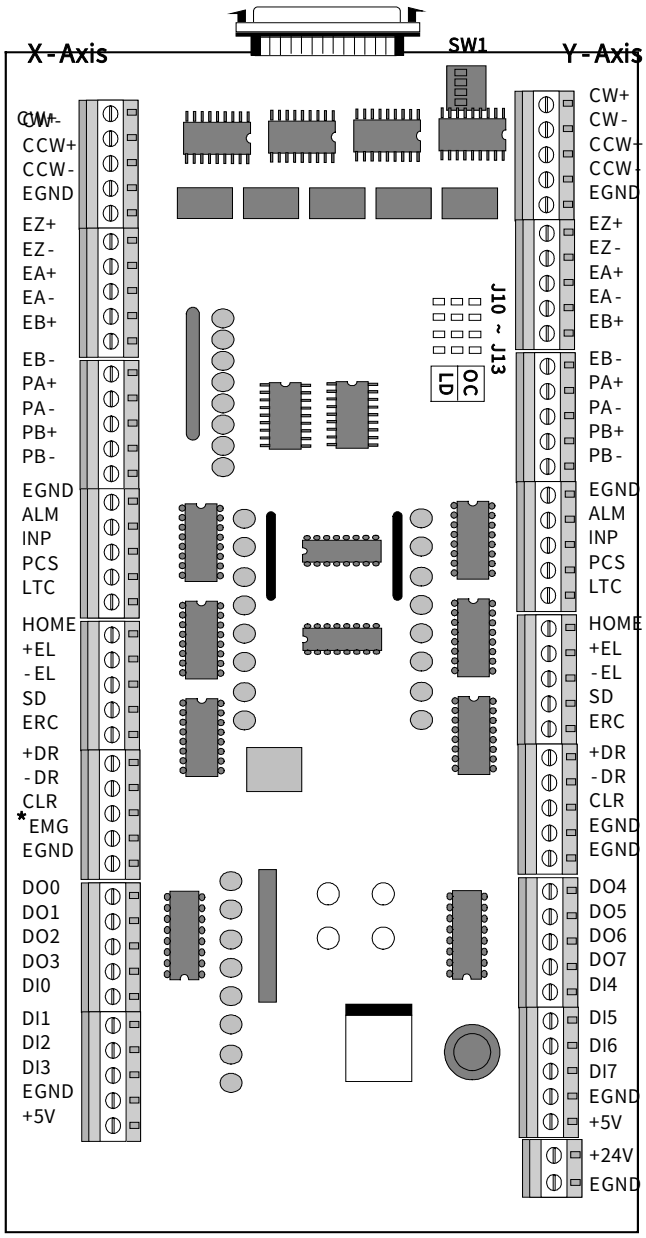


[그림 3-27] COMI-LXT5 Motion Terminal Board

COMI-LXT5 Motion Terminal Board 는 COMI-LX501 을 이용하여 모터제어 시스템을 구성할 때 각종 신호선을 연결하기 편하게 하기 위하여 사용되는 터미널 보드로써 각종 절연회로, LED Indicator, 신호 입출력 환경설정용 점퍼 등을 제공합니다.

COMI-LX501 용 터미널 보드로는 COMI-LXT5 와 COMI-STT5B 의 두 가지 종류를 사용할 수 있습니다. 신호선 연결을 위하여, COMI-LXT5 는 Screw type 의 터미널블럭을 제공하며 COMI-STT5B 는 커넥터 방식을 제공합니다.

COMI-LXT5 는 2 축 제어용 터미널 보드입니다. 따라서 4 축을 제어할 시에는 2 개의 COMI-LXT5 가 사용됩니다. COMI-LXT5 는 COMI-LXCABLE5 또는 COMI-LXCABLE6 을 통하여 COMI-LX501 과 연결됩니다.



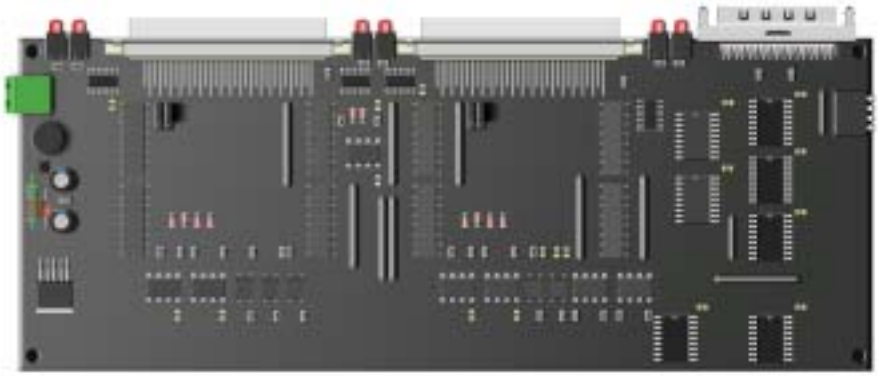
* EMG 단자는 Z-U 축 터미널보드에서는 SYNC(STA) 단자로 사용된다.

[그림 3-28] COMI-LXT5 Motion Terminal 배열

Pin	I/O	Description	Pin	I/O	Description
CW+	O	CW pulse(+)	PCS	I	Start In-position(override of target position 2)
CW-	O	CW pulse(-)	LTC	I	Latch the value of a counter
CCW+	O	CCW pulse(+) or direction	HOME	I	Home return signal
CCW-	O	CCW pulse(-) or direction	+EL	I	End limit signal (Positive direction)
EGND	O	Ext. ground	-EL	I	End limit signal (Negative direction)
EZ+	I	Encoder Z phase (+)	SD	I	Ramping down
EZ-	I	Encoder Z phase (-)	ERC	O	Reset the Deviation counter of servo driver
EA+	I	Encoder A phase (+)	+DR	I	Hard start by switch (+)
EA-	I	Encoder A phase (-)	-DR	I	Hard start by switch (-)
EB+	I	Encoder B phase (+)	CLR	I	Counter clear
EB-	I	Encoder B phase (-)	EMG	I	Emergency stop (X-Y 축 터미널에만 있음)
PA+	I	Pulser A phase (+)	SYNC	I	Simultaneous start(STA, Z-U 축 터미널에만 있음)
PA-	I	Pulser A phase (-)	DO#	O	General purpose digital output
PB+	I	Pulser B phase (+)	DI#	I	General purpose digital input
PB-	I	Pulser B phase (-)	+5V	O	+5Volt power supply output
ALM	I	Alarm signal	+24V	I	Ext. power supply input, D.C 24V
INP	I	In-position signal			

[표 3-4] COMI-LXT5 Motion Terminal 단자 설명

3.2.2 COMI-STT5B Motion Terminal Board



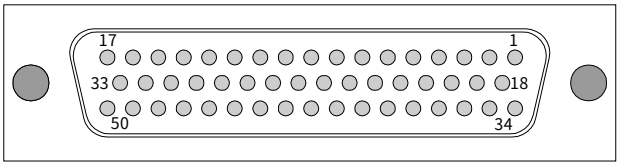
[그림 3-29] COMI-STT5B Motion Terminal Board

COMI-STT5B Motion Terminal Board 는 COMI-LX501 을 이용하여 모터제어 시스템을 구성할 때 각종 신호선을 연결하기 편하게 하기 위하여 사용되는 터미널 보드로써 각종 절연회로, LED Indicator, 신호 입출력 환경설정용 점퍼 등을 제공합니다.

COMI-LX501 용 터미널 보드로는 COMI-LXT5 와 COMI-STT5B 의 두 가지 종류를 사용할 수 있습니다. 신호선 연결을 위하여, COMI-LXT5 는 Screw type 의 터미널블럭을 제공하며 COMI-STT5B 는 커넥터 방식을 제공합니다.

COMI-STT5B 는 2 축 제어용 터미널 보드입니다. 따라서 4 축을 제어할 시에는 2 개의 COMI-STT5B 가 사용됩니다. COMI-STT5B 는 1 개의 68 핀 커넥터와 2 개의 50 핀 커넥터를 제공합니다. 68 핀 커넥터는 COMI-LXCABLE5 또는 COMI-LXCABLE6 을 이용하여 COMI-LX501 과 연결할 수 있도록 하는 커넥터이며, 50 핀 커넥터 2 개는 각종 모션 관련 입출력 신호를 연결할 수 있도록 하는 커넥터입니다.

1) COMI-STT5B PIN 배열



Pin50		Pin17	Pin50		Pin17
VCC	Pin33	Vcc	VCC	Pin33	Vcc
Pin49	VCC	Pin16	Pin49	VCC	Pin16
GND	Pin32	GND	GND	Pin32	GND
Pin48	GND	Pin15	Pin48	GND	Pin15
CLR_X	Pin31	-EL_X	CLR_Y	Pin31	-EL_Y
Pin47	-EB_X	Pin14	Pin47	-EB_Y	Pin14
PCS_X	Pin30	+EL_X	PCS_Y	Pin30	+EL_Y
Pin46	+EB_X	Pin13	Pin46	+EB_Y	Pin13
-DR_X	Pin29	ORG_X	-DR_Y	Pin29	ORG_Y
Pin45	GND	Pin12	Pin45	GND	Pin12
+DR_X	Pin28	LTC_X	+DR_Y	Pin28	LTC_Y
Pin44	-EA_X	Pin11	Pin44	-EA_Y	Pin11
SD_X	Pin27	INP_X	SD_Y	Pin27	INP_Y
Pin43	+EA_X	Pin10	Pin43	+EA_Y	Pin10
GND	Pin26	NC	GND	Pin26	EMG
Pin42	GND	Pin9	Pin42	GND	Pin9
DI_3_X	Pin25	ALM_X	DI_3_Y	Pin25	ALM_Y
Pin41	-EZ_X	Pin8	Pin41	-EZ_Y	Pin8
DI_2_X	Pin24	ERC_X	DI_2_Y	Pin24	ERC_Y
Pin40	+EZ_X	Pin7	Pin40	+EZ_Y	Pin7
DI_1_X	Pin23	GND	DI_1_Y	Pin23	GND
Pin39	-PB_X	Pin6	Pin39	-PB_Y	Pin6
DI_0_X	Pin22	-CCW_X	DI_0_Y	Pin22	-CCW_Y
Pin38	+PB_X	Pin5	Pin38	+PB_Y	Pin5
GND	Pin21	+CCW_X	GND	Pin21	+CCW_Y
Pin37	GND	Pin4	Pin37	GND	Pin4
DO_3_X	Pin20	GND	DO_3_Y	Pin20	GND
Pin36	-PA_X	Pin3	Pin36	-PA_Y	Pin3
DO_2_X	Pin19	-CW_X	DO_2_Y	Pin19	-CW_Y
Pin35	+PA_X	Pin2	Pin35	+PA_Y	Pin2
DO_1_X	Pin18	+CW_X	DO_1_Y	Pin18	+CW_Y
Pin34	GND	Pin1	Pin34	GND	Pin1
DO_0_X		GND	DO_0_Y		GND

[그림 3-30] COMI-STT5B 커넥터 핀 배열

No.	Name	I/O	Description
1	GND		Ext. ground
2	CW+	I	CW pulse (+)
3	CW-	I	CW pulse (-)
4	GND		Ext. ground
5	CCW+	I	CCW pulse (+) or direction
6	CCW-	I	CCW pulse (-) or direction
7	GND		Ext. ground
8	ERC		Reset the Deviation counter of servo driver
9	ALM	I	Alarm signal
*10	NC…	I	No Connection(X,Z), EMG(Y), STA(U)
11	INP	I	In-position signal
12	LTC		Latch the value of a counter
13	ORG	I	Home return signal
14	+EL	I	End limit signal (+)
15	-EL	I	End limit signal (-)
16	GND		Ext. ground
17	+5V	O	Ext. power
18	GND		Ext. ground
19	PA+	I	Pulser A phase (+)
20	PA-	I	Pulser A phase (-)
21	GND		Ext. ground
22	PB+	I	Pulser B phase (+)
23	PB-	I	Pulser B phase (-)
24	EZ+	I	Encoder Z phase (+)
25	EZ-	I	Encoder Z phase (-)
26	GND		Ext. ground
27	EA+	I	Encoder A phase (+)
28	EA-	I	Encoder A phase (-)

29	GND		Ext. ground
30	EB+	I	Encoder B phase (+)
31	EB-	I	Encoder B phase (-)
32	GND		Ext. ground
33	+5V	O	Ext. power
34	DO0	O	Digital output 0
35	DO1	O	Digital output 1
36	DO2	O	Digital output 2
37	DO3	O	Digital output 3
38	GND		Ext. ground
39	DI0	I	Digital input 0
40	DI1	I	Digital input 1
41	DI2	I	Digital input 2
42	DI3	I	Digital input 3
43	GND		Ext. ground
44	SD	I	Ramping down
45	+DR	I	Hard start by switch (+)
46	-DR	I	Hard start by switch (-)
47	PCS	I	Start In-position(override of target position 2)
48	CLR	I	Counter clear
49	GND		Ext. ground
50	+5V	O	Ext. power

★. 주의 : 10 번 핀은 X,Z 축을 연결할 때는 NC(No connect)이며 Y 축은 EMG 신호, U 축은 STA 신호로 사용 됩니다.

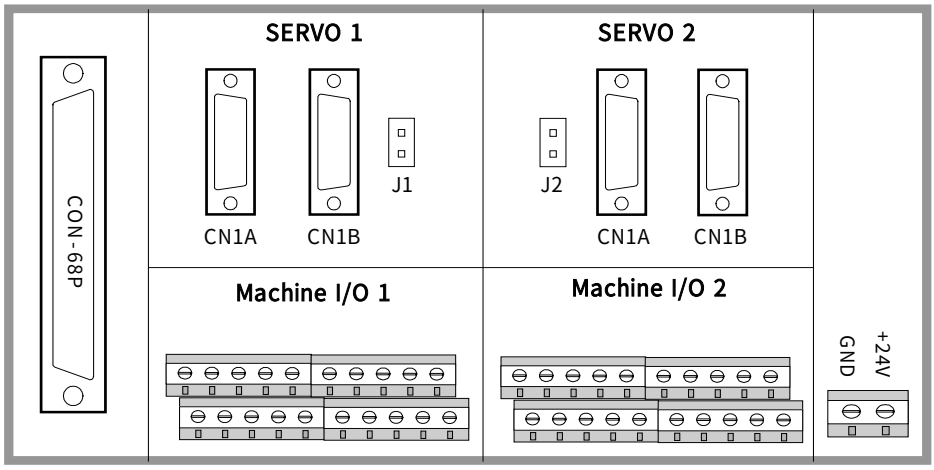
[표 3-5] COMI-STT5B 핀 배열 설명

3.2.3 COMI-LXT5M 미쯔비시 서보드라이버 전용 Terminal Board

[그림 3-31] COMI-LXT5M Motion Terminal Board

COMI-LXT5M 은 COMI-LX502/4/8 을 미쯔비시 MR-J2 시리즈 서보드라이버와 함께 사용할 때 결선작업을 용이하게 하기위하여 제공되는 터미널 보드입니다. COMI-LXT5M 은 MR-J2 서보드라이버와 연결되는 모든 신호를 커넥터 처리하였습니다. 따라서 사용자는 COMI-LXT5M 과 MR-J2 서보드라이버간에 20 핀 전용 케이블 2 개를 연결하기만 하면 모든 서보 인터페이스가 이루어집니다.

하나의 COMI-LXT5M 에서는 2 축 모션 인터페이스를 제공합니다. COMI-LXT5M 의 구조는 다음과 같습니다.



[그림 3-32] COMI-LXT5M 외관도

▷ **CON-68P** : COMI-LX502/4/8 과 연결되는 68-pin connector
▷ **CN1A** : MR-J2 서보 드라이버의 CN1A 커넥터와 연결되는 커넥터
▷ **CN1B** : MR-J2 서보 드라이버의 CN1B 커넥터와 연결되는 커넥터
▷ **J1, J2** : 디지털 출력 채널 중 하나를 이용하여 SERVO ALARM RESET(CN1B 의 14 번 핀, RES)신호로 사용할 지를 설정하는 점퍼.

JUMPER	Description
J1	Open 시키면 Servo1 CN1B 의 14 번 핀을 사용하지 않습니다. Close 시키면 디지털 출력채널 [1] 하나를 14 번 핀에 할당합니다. CN1B 의 14 번 핀은 서보드라이버의 Alarm reset 으로 사용됩니다.
J2	Open 시키면 Servo2 CN1B 의 14 번 핀을 사용하지 않습니다. Close 시키면 디지털 출력채널 [2] 하나를 14 번 핀에 할당합니다. CN1B 의 14 번 핀은 서보드라이버의 Alarm reset 으로 사용됩니다.

▷ **Machine I/O 1** : 첫 번째 축과 관련하여 서보 드라이버 인터페이스를 제외한 각종 I/O 신호 연결단자로서 [표 3-6]과 같습니다.

[1] COMI-LX502/4/8 의 CON1 에 연결되면 DO0, CON2 에 연결되면 DO3, CON3 에 연결되면 DO6, CON4 에 연결되면 DO9 채널이 할당됩니다.
[2] COMI-LX502/4/8 의 CON1 에 연결되면 DO1, CON2 에 연결되면 DO4, CON3 에 연결되면 DO7, CON4 에 연결되면 DO10 채널이 할당됩니다.

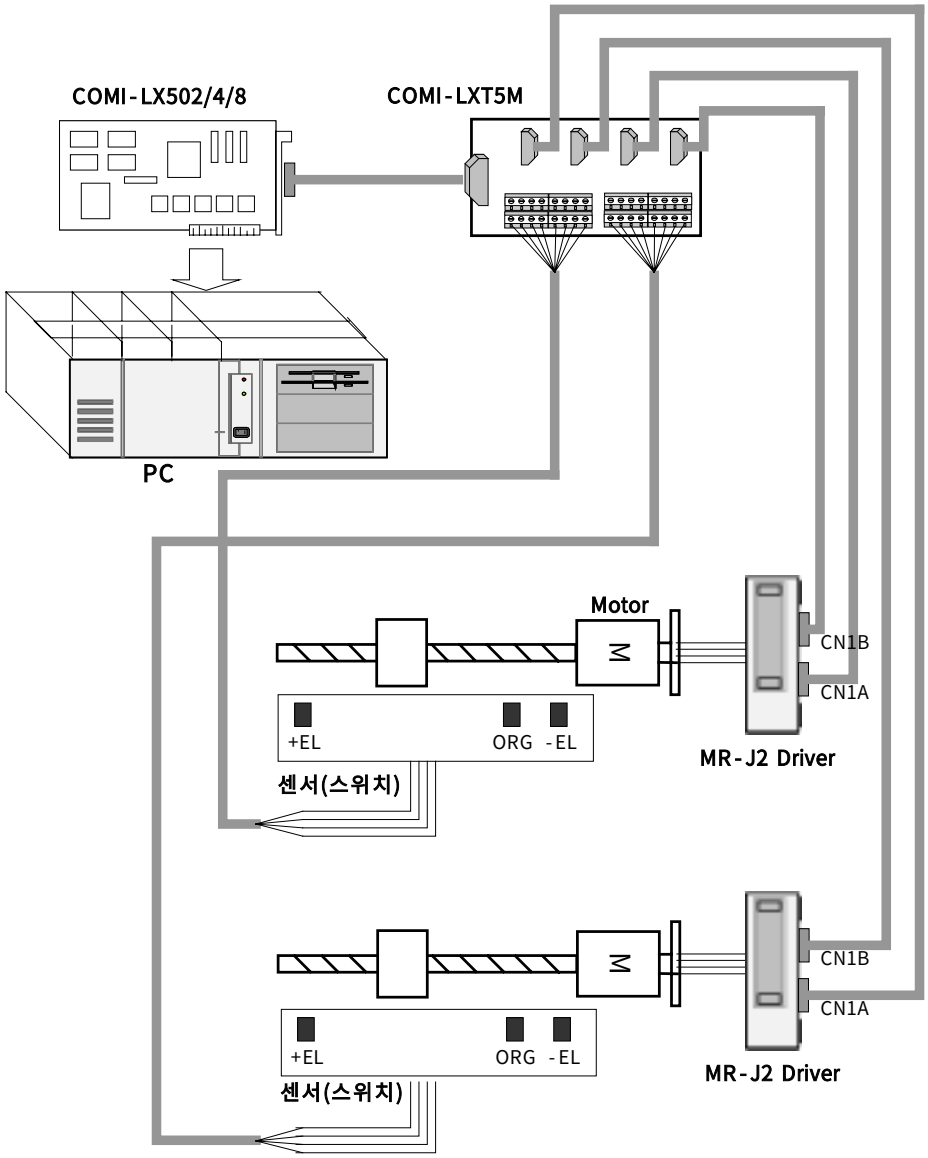
Pin No.	I/O	Pin Name	Pin No.	I/O	Pin Name
1	O	DO0(/3/6/9)	2	O	DO1(/4/7/10)
3	O	DO2(/5/8/11)	4	I/O	DO_COM
5	I	LTC1	6	O	CMP1
7	I	PCS1	8	I	CLR1
9	I	+DR1	10	I	-DR1
11	I	-EL1	12	I	SD1
13	I	+EL1	14	I	ORG1
15	I/O	GND	16	O	+24V
17	I/O	GND	18	O	+24V
19	I/O	GND	20	O	+24V

[표 3-6] Machine I/O 1 pin description

▷ **Machine I/O 2** : 두 번째 축과 관련하여 서보 드라이버 인터페이스를 제외한 각종 I/O 신호 연결단자로서 [표 3-7]과 같습니다.

Pin No.	I/O	Pin Name	Pin No.	I/O	Pin Name
1	O	DI0(/3/6/9)	2	O	DI1(/4/7/10)
3	O	DI2(/5/8/11)	4	I/O	DI_COM
5	I	LTC2	6	O	CMP2
7	I	PCS2	8	I	CLR2
9	I	+DR2	10	I	-DR2
11	I	-EL2	12	I	SD2
13	I	+EL2	14	I	ORG2
15	I/O	GND	16	O	+24V
17	I/O	GND	18	O	+24V
19	I/O	GND	20	O	+24V

[표 3-7] Machine I/O 2 pin description



[그림 3-33] COMI-LX502/4/8 과 미쯔비시 MR-J2 서보 드라이버 전용 터미널(COMI-LXT5M)을 이용한 모션제어 시스템 구성도 [1)]

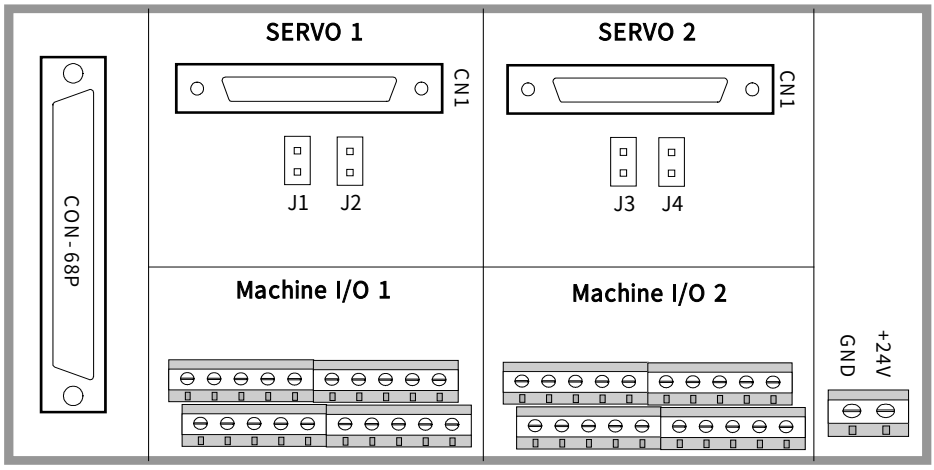
[1)] 그림에서는 2 축 연결만 도시하였으나, COMI-LX504 는 4 축, COMI-LX508 은 8 축을 같은 방법으로 연결할 수 있습니다

3.2.3 COMI-LXT5S 삼성 서보드라이버 전용 Terminal Board

[그림 3-34] COMI-LXT5M Motion Terminal Board

COMI-LXT5S 는 COMI-LX502/4/8 을 삼성 CSD-시리즈 서보드라이버와 함께 사용할 때 결선작업을 용이하게 하기위하여 제공되는 터미널 보드입니다. COMI-LXT5S 는 삼성 CSD 서보드라이버와 연결되는 모든 신호를 커넥터 처리하였습니다. 따라서 사용자는 COMI-LXT5S 와 CSD 서보드라이버간에 50 핀 전용 케이블 1 개를 연결하기만 하면 모든 서보 인터페이스가 이루어집니다.

하나의 COMI-LXT5S 에서는 2 축 모션 인터페이스를 제공합니다. COMI-LXT5S 의 구조는 다음과 같습니다.



[그림 3-35] COMI-LXT5S 외관도

▷ **CON-68P** : COMI-LX502/4/8 과 연결되는 68-pin connector
▷ **CN1** : CSD 서보 드라이버의 CN1 커넥터와 연결되는 커넥터
▷ **J1, J3** : 디지털 출력 채널 중 하나를 이용하여 SERVO ALARM RESET(CN1 의 7 번핀, ALM-RST)신호로 사용할 지를 설정하는 점퍼.

JUMPER	Description
J1	Open 시키면 Servo1 의 CN1 7 번 핀을 사용하지 않습니다. Close 시키면 디지털 출력채널 [1] 하나를 7 번 핀에 할당합니다. CN1 의 7 번 핀은 서보드라이버의 Alarm reset 으로 사용됩니다.
J3	Open 시키면 Servo2 의 CN1 7 번 핀을 사용하지 않습니다. Close 시키면 디지털 출력채널 [2] 하나를 7 번 핀에 할당합니다. CN1 의 7 번 핀은 서보드라이버의 Alarm reset 으로 사용됩니다.

[표 3-8] J1, J3 Jumper description

▷ **J2, J4** : COMI-LX502/4/8 의 ERC 출력을 CN1 의 10 번 핀에 연결할 지를 설정하는 점퍼

[1] COMI-LX502/4/8 의 CON1 에 연결되면 DO0, CON2 에 연결되면 DO3, CON3 에 연결되면 DO6, CON4 에 연결되면 DO9 채널이 할당됩니다.
[2] COMI-LX502/4/8 의 CON1 에 연결되면 DO1, CON2 에 연결되면 DO4, CON3 에 연결되면 DO7, CON4 에 연결되면 DO10 채널이 할당됩니다.

JUMPER	Description
J2	Open 시키면 Servo1 의 CN1 10 번 핀을 사용하지 않습니다. Close 시키면 ERC 출력을 CN1 10 번 핀에 할당합니다. CSDJ 플러스 서보드라이버에서는 CN1 의 10 번 핀을 위치카운터 클리어 신호로 사용할 수 있습니다.
J4	Open 시키면 Servo2 의 CN1 10 번 핀을 사용하지 않습니다. Close 시키면 ERC 출력을 CN1 10 번 핀에 할당합니다. CSDJ 플러스 서보드라이버에서는 CN1 의 10 번 핀을 위치카운터 클리어 신호로 사용할 수 있습니다.

[표 3-9] J1, J3 Jumper description

▷ **Machine I/O 1** : 첫 번째 축과 관련하여 서보 드라이버 인터페이스를 제외한 각종 I/O 신호 연결단자로서 [표 3-10]과 같습니다.

Pin No.	I/O	Pin Name	Pin No.	I/O	Pin Name
1	O	DO0(/3/6/9)	2	O	DO1(/4/7/10)
3	O	DO2(/5/8/11)	4	I/O	DO_COM
5	I	LTC1	6	O	CMP1
7	I	PCS1	8	I	CLR1
9	I	+DR1	10	I	-DR1
11	I	-EL1	12	I	SD1
13	I	+EL1	14	I	ORG1
15	I/O	GND	16	O	+24V
17	I/O	GND	18	O	+24V
19	I/O	GND	20	O	+24V

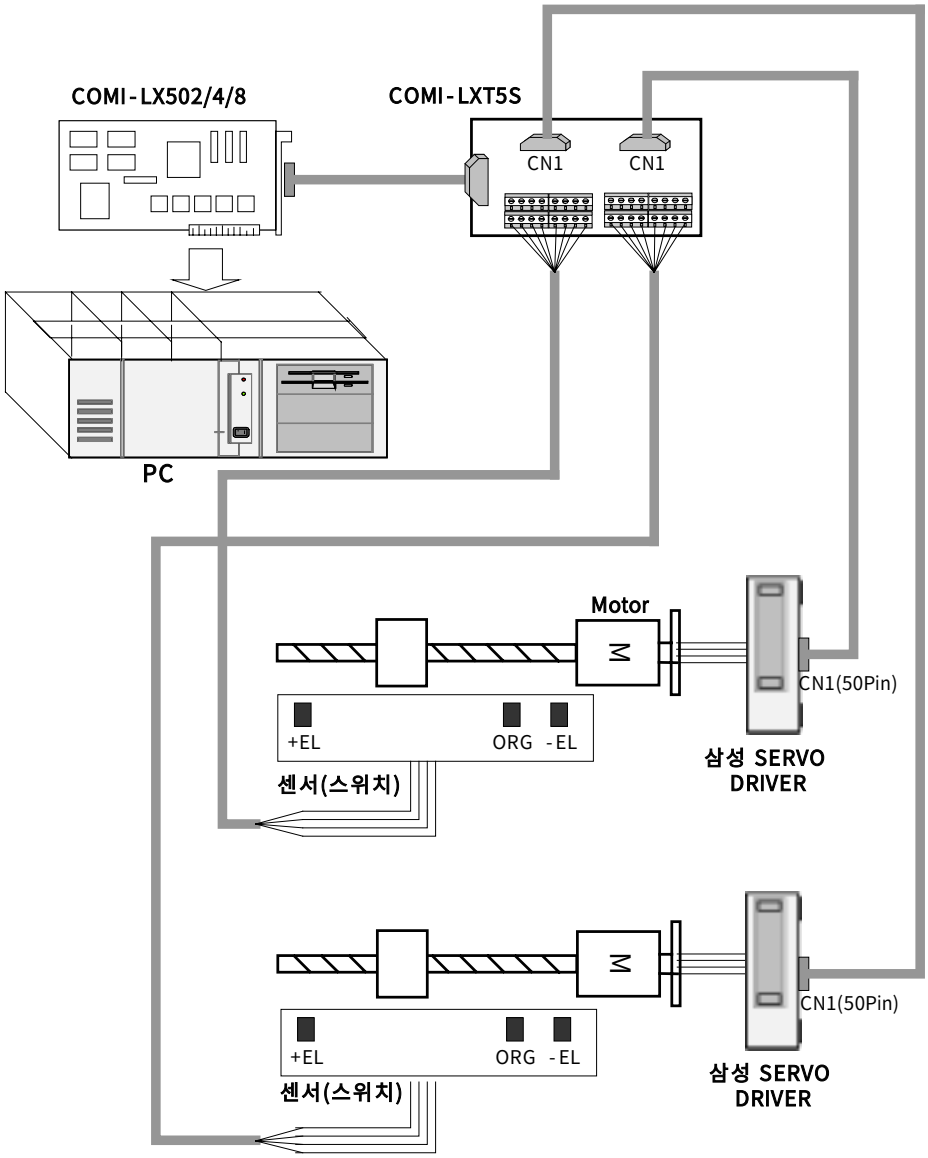
[표 3-10] Machine I/O 1 pin description

▷ **Machine I/O 2** : 두 번째 축과 관련하여 서보 드라이버 인터페이스를 제외한 각종 I/O 신호 연결단자로서 [표 3-11]과 같습니다.

Pin No.	I/O	Pin Name	Pin No.	I/O	Pin Name
1	O	DI0(/3/6/9)	2	O	DI1(/4/7/10)
3	O	DI2(/5/8/11)	4	I/O	DI_COM
5	I	LTC2	6	O	CMP2
7	I	PCS2	8	I	CLR2

Pin No.	I/O	Pin Name	Pin No.	I/O	Pin Name
9	I	+DR2	10	I	-DR2
11	I	-EL2	12	I	SD2
13	I	+EL2	14	I	ORG2
15	I/O	GND	16	O	+24V
17	I/O	GND	18	O	+24V
19	I/O	GND	20	O	+24V

[표 3-11] Machine I/O 2 pin description



[그림 3-36] COMI-LX502/4/8 과 삼성 서보 드라이버 전용 터미널(COMI-LXT5S)을 이용한 모션제어 시스템 구성도 [1)]

[1)] 그림에서는 2 축 연결만 도시하였으나, COMI-LX504 는 4 축, COMI-LX508 은 8 축을 같은 방법으로 연결할 수 있습니다

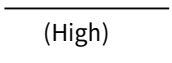
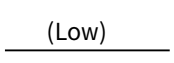
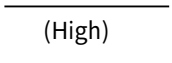
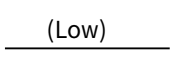
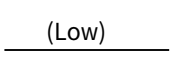
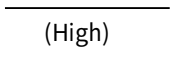
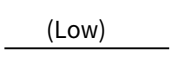
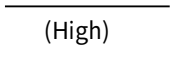
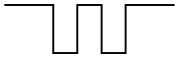
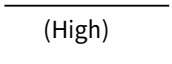
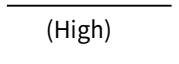
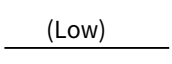
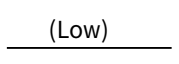
3.3 입/출력 신호 설명

Signal	I/O	Description
CW	O	모터 구동용 펄스 출력
CCW	O	모터 구동 방향 지시 신호 또는 역방향 모터 구동용 펄스 출력
ALM	I	각 축의 비상신호 입력
SVON	O	Servo-on 신호 출력
INP	I	Servo driver In-position 신호 입력
ERC	O	Deviation counter clear 신호 출력
RDY	I	Servo driver ready 신호 입력
EA	I	Enocder A상 신호 입력
EB	I	Encoder B상 신호 입력
EZ	I	Encoder Z상 신호 입력
+EL	I	Positive limit 센서 신호 입력
-EL	I	Negative limit 센서 신호 입력
ORG	I	원점센서 신호 입력
SD	I	외부 스위치에 의한 stop-after-deceleration 신호 입력
+DR	I	External switch operation 신호 입력(정방향)
-DR	I	External switch operation 신호 입력(역방향)
PCS	I	외부 스위치에 의한 start-in-positioning 신호 입력
CLR	I	카운터 클리어 신호 입력
LTC	I	카운트 값 래치 명령신호. 상승에지에서 래치
CMP	O	위치 비교에 의한 펄스 신호 출력
PA	I	Manual pulser mode 구동 펄스 신호 입력
PB	I	Manual pulser mode 구동 펄스 신호 입력
STA	I	다축 동기 구동 시작 신호 입력
STP	I	다축 동기 구동 종료 신호 입력

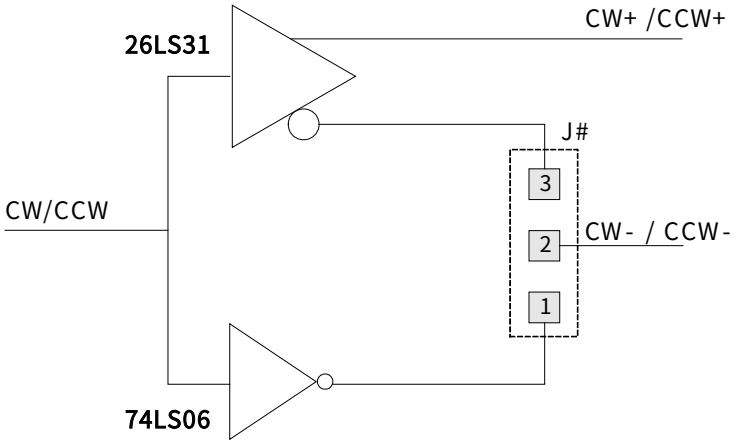
[표 3-12] COMI-LX50x의 입출력 신호 리스트

3.3.1 펄스 출력 신호 (CW/CCW)

COMI-LX50x Motion Controller 는 각 축마다 CW 과 CCW, 2 개의 펄스 출력을 제공합니다. 이 두 신호는 펄스 구동 방식의 스텝모터 또는 DC/AC 서보 모터의 구동 신호로 사용됩니다. 이 두 신호는 OUT/DIR 출력 방식과 CW/CCW 출력 방식을 모두 지원합니다. 여기서 OUT/DIR 출력 방식은 CW 신호는 펄스 출력을 CCW 신호는 방향지시(Low/High)를 출력하는 방식을 말합니다. 그리고 CW/CCW 출력 방식은 모터의 구동 방향에 따라 CW 과 CCW 신호가 모두 펄스 출력을 내보내는 방식을 말합니다. 이때, 두 신호중 하나는 펄스출력을 내보내고, 다른 하나는 LOW 또는 HIGH 상태를 유지합니다. COMI-LX50x Motion Controller 의 출력 방식은 소프트웨어적으로 설정 가능하며 지원 가능한 출력 방식은 아래의 표와 같이 6 가지입니다.

Value	출력 형태			
	(+) 방향 운전 시		(-) 방향 운전 시	
	CW pin	CCW pin	CW pin	CCW pin
0		 (High)		 (Low)
1		 (High)		 (Low)
2		(Low) 		 (High)
3		(Low) 		 (High)
4		 (High)	(High) 	
5		(Low) 	(Low) 	

CW 과 CCW 신호의 연결 방법은 Line Drive 방식과 Open Collector 방식의 두 가지 방식이 있으며 . 이 두 가지 방식의 선택은 점퍼 설정에 의해서 이루어지는데 COMI-LX501 의 경우에는 터미널 보드인 COMI-LXT5 에 출력 방식 설정 점퍼가 있으며, COMI-LX502/4/8 은 각 보드내에 출력 방식 설정 점퍼가 있습니다.



[그림 3-37] CW/CCW 출력 신호 회로 구성도

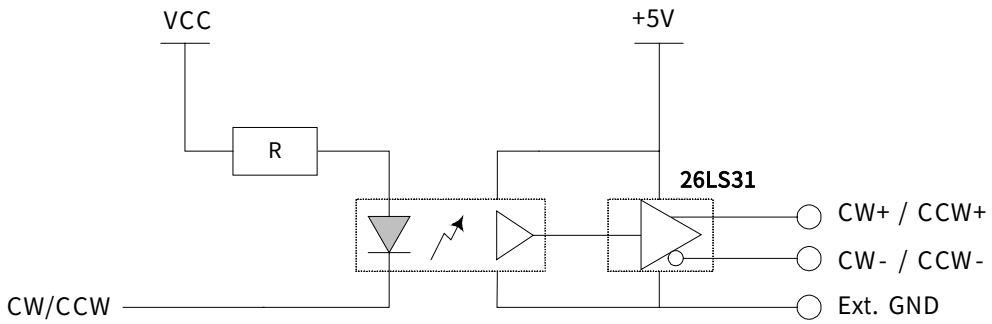
COMI-LXT5 의 출력 방식 설정 점퍼는 [표 3-13]과 같습니다. COMI-LX502/4/8 의 점퍼에 대한 설명은 앞 단원의 각 제품별 구조 설명편을 참조하십시오.

출력신호	O/C	L/D(Default)
CW (X or Z)	JP10 (1,2 번 점퍼연결)	JP10 (2,3 번 점퍼연결)
CCW (X or Z)	JP11 (1,2 번 점퍼연결)	JP11 (2,3 번 점퍼연결)
CW (Y or U)	JP12 (1,2 번 점퍼연결)	JP12 (2,3 번 점퍼연결)
CCW (Y or U)	JP13 (1,2 번 점퍼연결)	JP13 (2,3 번 점퍼연결)

[표 3-13] 출력 신호 형태 선택 점퍼 설정

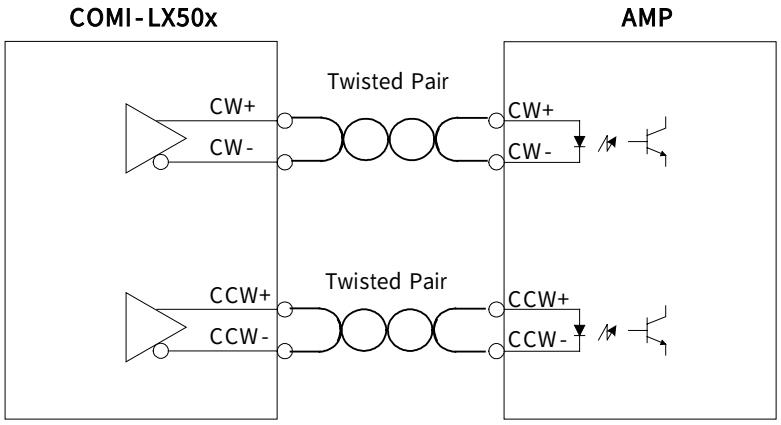
Line Drive 신호 연결

COMI-LX50x 에서 CW/CCW 의 Line Drive 출력 회로는 다음과 같이 구성되었습니다.



[그림 3-38] COMI-LX50x 의 Line Drive 출력 회로도

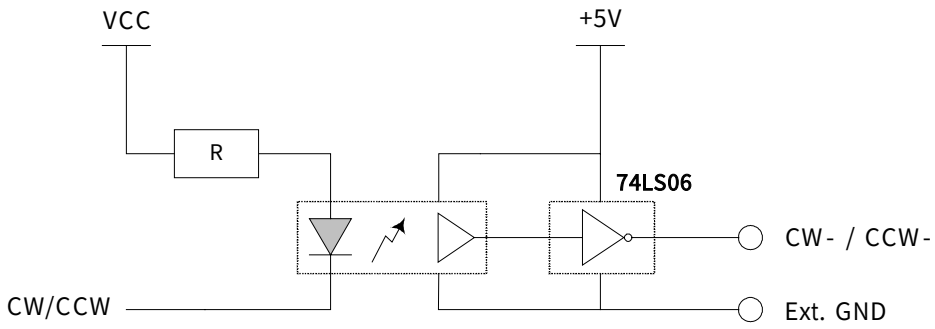
Line Drive 방식으로 사용할 경우에 AMP 와의 신호 연결은 다음과 같이 구성하면 됩니다. 이때 신호선은 Twisted pair 방식으로 배선하는 것이 바람직합니다.



[그림 3-39] Line Drive 출력 신호 연결법

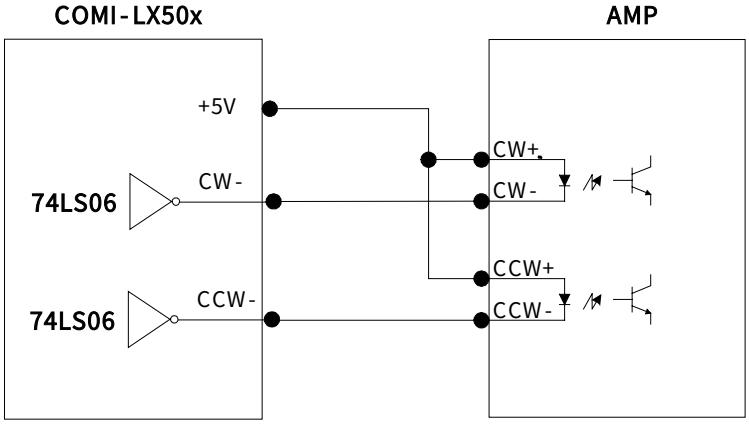
Open Collector 신호 연결

COMI-LX50x 에서 CW/CCW 의 Open Collector 출력 회로는 다음과 같이 구성되었습니다.



[그림 3-40] COMI-LX50x 의 Open collector 출력 회로도

Open Collector 방식으로 사용할 경우에 AMP 와의 신호 연결은 다음과 같이 구성하면 됩니다.



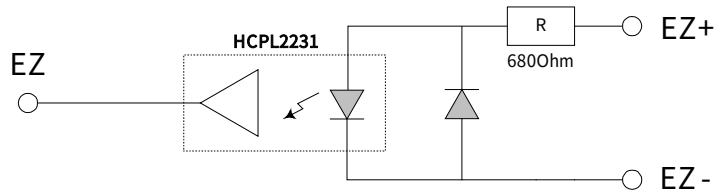
[그림 3-41] COMI-LX50x 의 Open collector 출력 신호 연결법

3.3.2 엔코더 입력 신호 (EA, EB, EZ)

COMI-LX50x Motion Controller 는 모터 또는 구조물의 실제 움직임을 계측할 수 있도록 하기 위하여 각 축마다 EA, EB, EZ 세 가지의 엔코더 입력 채널을 제공합니다. EA, EB, EZ 는 각가 엔코더의 Phase-A, Phase-B, Index(Zero)-Pulse 신호를 의미합

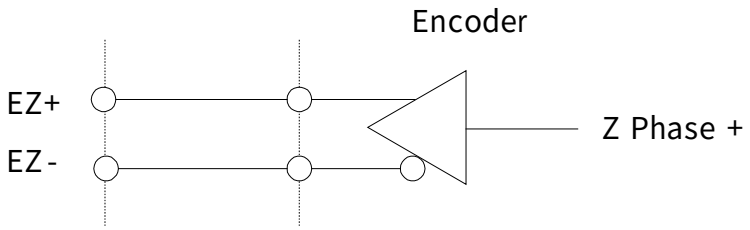
니다.

엔코더의 Z 상은 모터의 한 회전 당 한 번의 펄스신호가 출력됩니다. 모터의 RPM 이나 원점을 계측하고자 할 때 사용되어질 수 있습니다.



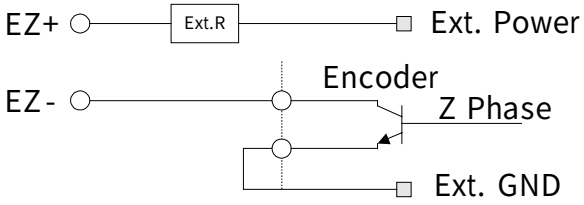
[그림 3-42] 엔코더 Z 신호 입력 회로도

[그림 3-42]는 모터에 장착된 엔코더의 Z 신호가 Isolation 회로를 통하여 입력되는 회로도 입니다. 포터커플러가 동작되면 COMI-LX50x 의 EZ 신호는 High 로 올라갑니다.



[그림 3-43] 엔코더 Z 신호 Line Drive

[그림 3-43]과 같이 Line Drive 형으로 엔코더의 Z 신호를 입력 받고자할 때에는 EZ+와 EZ-사의의 전압이 최소 3.5V 이상이어야 됩니다.



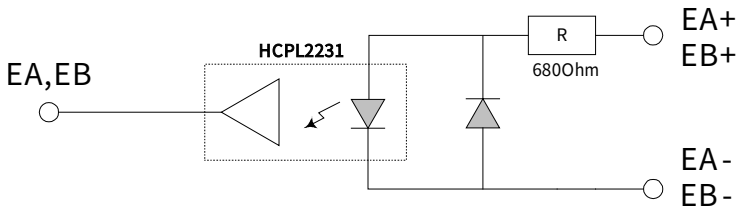
[그림 3-44] 엔코더 Z 신호 Open Collector

[그림 3-44]에서와 같이 Open Collector 형으로 신호를 입력 받을 땐 외부 전원에 의해 다음 표와 같이 저항값을 설정해야 됩니다.

외부 전원	외부 저항 R
+5V	0
+12V	1.8K ohm
+24V	4.3K ohm

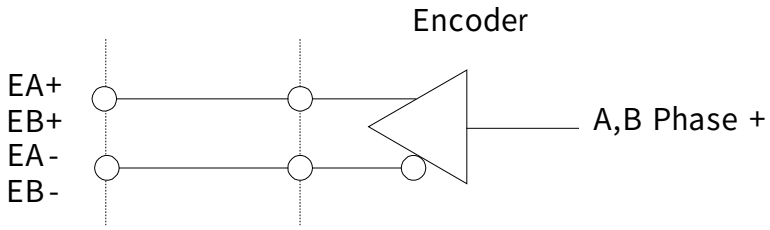
[표 3-14] Opec Collector 입력시 저항 선택(IF = 6mA max)

엔코더의 A,B 상은 모터의 한 회전 당 엔코더의 분해능과 같은 펄스신호가 출력됩니다. 정확한 RPM 이나 위치제어를 할 때 사용되어질 수 있습니다.



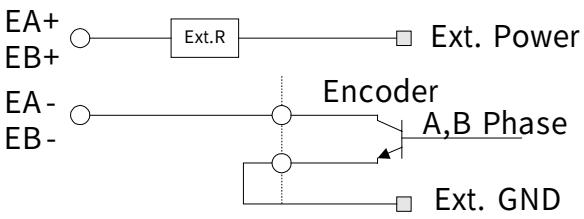
[그림 3-45] 엔코더 A,B 신호 입력 회로도

[그림 3-45]은 모터에 장착된 엔코더의 A,B 신호가 Isolation 회로를 통하여 입력되는 회로도 입니다. 포터커플러가 동작되면 COMI-LX50x 의 EA,EB 신호는 High 로 올라갑니다.



[그림 3-46] 엔코더 A,B 신호 Line Drive

[그림 3-46]와 같이 Line Drive 형으로 엔코더의 A,B 신호를 입력 받고자 할 땐 EA+(EB+)와 EA-(EB-)사의의 전압이 최소 3.5V 이상이어야 됩니다. 이때 신호선은 Twisted pair 방식으로 배선하는 것이 바람직합니다.



[그림 3-47] 엔코더 A,B 신호 Open Collector

[그림 3-47]에서와 같이 Open Collector 형으로 신호를 입력 받을 땐 외부 전원에 의해 다음 표와 같이 저항값을 설정해야 됩니다.

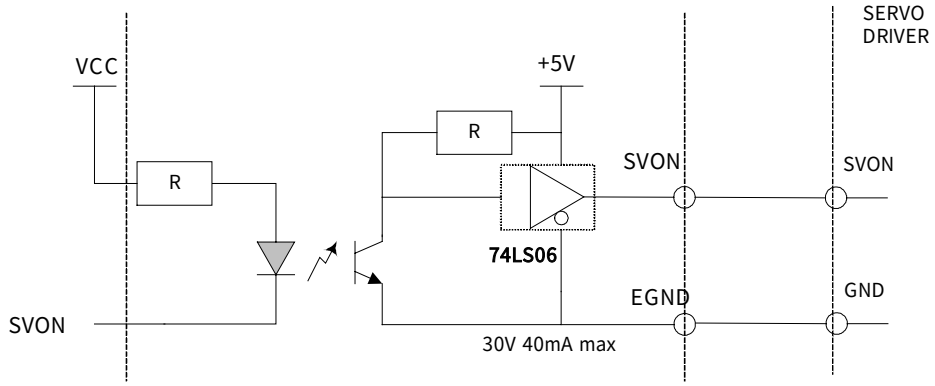
외부 전원	외부 저항 R
+5V	0
+12V	1.8K ohm
+24V	4.3K ohm

[표 3-15] Opec Collector 입력시 저항 선택(IF = 6mA max)

3.3.3 Servo-ON(SVON)

서보 드라이버를 사용하실 때는 외부에서 스위치를 이용하여 서보드라이버의 ON/OFF를 제어할 수 있도록 하는데, 이를 SERVO-ON 신호라 합니다. COMI-LX502/4/8 제품의 경우에는 SVON 출력이 따로 제공되며, COMI-LX501 에서는 SERVO-ON 출력 신호를 따로 제공하지는 않으며, 대신에 범용 디지털 출력을 서보 드라이버의 SERVO-ON 신호 입력핀과 연결하시고 디지털 출력을 이용하여 SERVO-ON 제어를 하실 수 있습니다.

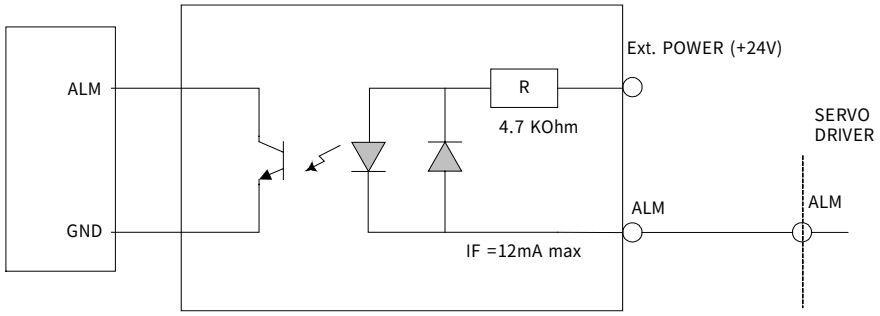
COMI-LX502/4/8 의 경우에 SVON 입력 회로는 다음과 같습니다.



[그림 3-48] SVON 출력 회로도

3.3.4 ALARM 입력 신호(ALM)

ALARM 신호는 각 축마다 하나씩 제공되는 서보모터 인터페이스 입력 신호로서 서보모터 드라이버에서 출력되는 ALARM 신호를 입력받는 신호입니다. ALARM 신호가 ON 이 되면 모션은 정지하게 됩니다.



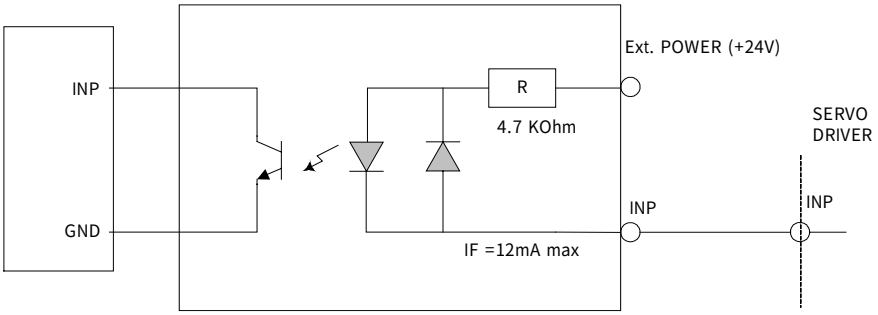
[그림 3-49] ALM 신호 입력 회로도 및 신호 연결법

3.3.5 In-position 입력 신호(INP)

INP 신호는 각 축마다 하나씩 제공되는 서보모터 인터페이스 입력 신호로서 서보모

터 드라이버의 편차 카운터(Deviation counter)가 0 이 됨을 의미합니다. 펄스구동 방식의 서보모터 드라이버는 지령펄스(Command pulse)와 궤환펄스(Feedback pulse)의 편차를 카운트하는 편차카운터(Deviation counter)를 내장하고 있습니다. 서보모터가 실제로 움직이는 것은 지령펄스보다 약간의 지연이 있어서 지령펄스와 궤환펄스 사이에는 약간의 편차가 발생하게 됩니다. 서보 드라이버는 편차카운트값이 0 이 될때까지 모터를 제어하게 됩니다. 따라서 편차카운트값이 0 이 되는 것은 모션 구동이 완료됐음을 의미하게 됩니다. 서보드라이버에서 출력되는 INP 신호는 편차값이 0 이 되면 ON 이 되고, 0 보다 커지면 OFF 가 됩니다. 단, 서보 드라이버의 설정에 따라 편차값 기준은 0 보다 큰값이 될 수도 있습니다. 예를 들어 MITSUBISHI-J2 서보 드라이버의 경우 Param No.5 “In-position range” 파라미터에서 설정한 값이 그 기준이 됩니다.

COMI-LX50x 는 INP 신호를 이용하여 모션의 완료를 판단할 수 있습니다. **COMILX_MC_SetMioCfgINP**() 함수 의 bInpEnable 파라미터를 TRUE 로 하면 COMI-LX50x 는 펄스 출력을 완료하였어도 INP 신호가 ON 이 아니면 모션이 완료되지 않은 것으로 간주합니다.

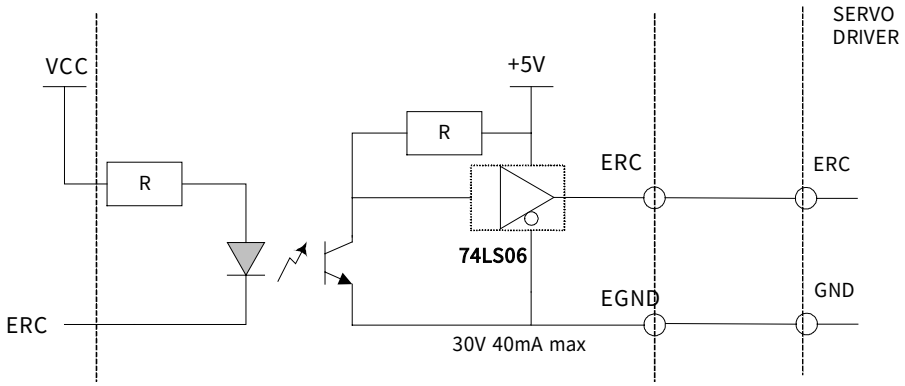


[그림 3-50] INP 신호 입력 회로도 및 신호 연결법

3.3.6 편차카운터 클리어 출력 신호(ERC)

편차카운터 신호는 각 축마다 하나씩 제공되는 서보모터 인터페이스 출력 신호로서 서보모터 드라이버의 편차 카운터(Deviation counter)를 클리어해줍니다. INP 신호 편에서 설명한 바와 서보모터 드라이버는 편차카운트값이 0 이 될 때까지 모터를 구동합니다. 따라서 COMMAND 펄스 출력이 완료되었어도 모터가 정지하기까지는 약간의 시간 지연이 있을 수 있습니다.

ERC 신호는 원점 복귀시에 사용되며 원점복귀가 완료되는 시점에 시간지연없이 모터를 즉시 정지시키고자하는 목적으로 사용됩니다. ERC 신호는 원점 복귀가 완료되면 자동으로 출력됩니다. 단, ERC 신호를 자동으로 출력하기 위해서는 **COMILX_MC_SetHomeConfig**() 함수의 *nErcOut* 파라미터값을 1 로 하여야 합니다.

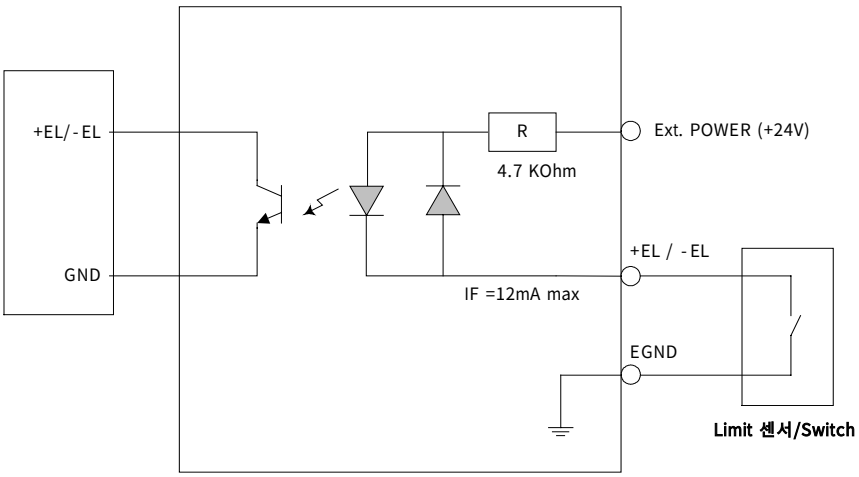


[그림 3-51] ERC 신호 출력 회로도 및 신호 연결법

3.3.7 End Limit 입력 신호 (+EL, -EL)

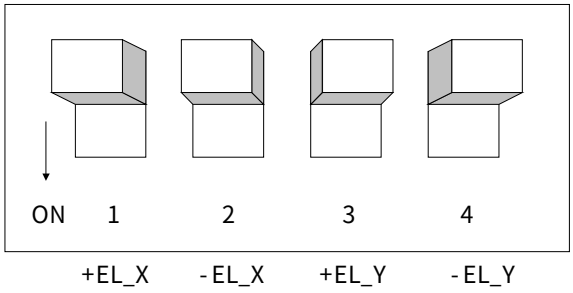
COMI-LX50x Motion Controller 는 모션의 이동 한계를 자동으로 감지할 수 있도록 각 축마다 +EL 과 -EL 의 두 가지 End Limit 신호 입력을 제공합니다. +EL 신호는 + 방향으로의 End limit 을 의미하며, -EL 신호는 -방향으로의 End limit 을 의미합니다. +EL 이나 -EL 신호 입력이 ON 이 되면 COMI-LX50x 은 자동적으로 모션을 중지합니다. 따라서 +EL 과 -EL 신호는 모션의 이동 한계를 구조적으로 제한하고자 할때 사용합니다. 그리고 +EL 신호와 -EL 신호는 원점 복귀시에 원점의 레퍼런스로도 사용할 수 있습니다. 원점복귀에 대한 자세한 사항은 “5.4.6 원점 복귀(Home Return) 함수” 편을 참조하십시오.

+EL 과 -EL 신호의 입력 회로 및 신호 연결은 다음과 같습니다.



[그림 3-52] +EL/-EL 신호 입력 회로도 및 신호 연결법

외부 전원은 24V 이고 이때의 저항 R 은 4.7 Kohm, 최대 전류량은 12mA 입니다. COMI-LX501 을 사용하는 경우에는 Limit 스위치의 접점 형식을 아래 그림과 같이 COMI-LXT5 의 SW1 을 이용하여 A 접점(Normal open) 또는 B 접점(Normal close)으로 설정할 수 있습니다.



OFF : Select ‘A’ Contact EL Switch (Normal Open)

ON : Select ‘B’ Contact EL Switch (Normal Close)

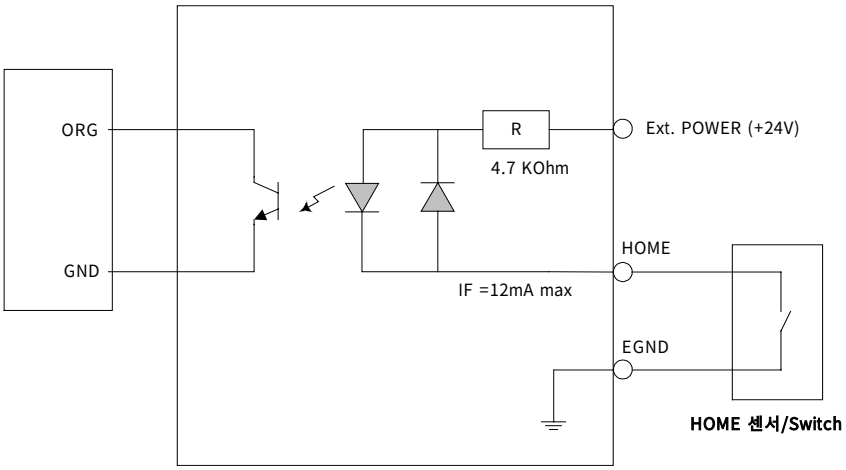
[그림 3-53] +EL/-EL 신호의 입력 로직 설정 점퍼

3.3.8 원점 센서 입력 신호 (HOME or ORG)

원점 신호(HOME or ORG)는 기구적인 원점을 알리는 신호를 말합니다. 정확한 위치 제어를 하기 위해서는 모션의 좌표 원점을 일관적으로 해줄 필요가 있습니다. 이를 위해서 일반적으로 원점 센서등을 이용하여 원점을 감지할 수 있도록 하고 작업 초기에 기구물을 원점으로 복귀시켜 모션의 좌표를 원점 좌표로 맞추어준 후에 본격적인 작업을 수행하는 방법을 사용합니다.

COMI-LX50x 는 각 축마다 1 개씩의 원점 신호 입력(HOME or ORG)을 제공하여 원점을 감지할 수 있도록 합니다. 그리고 원점으로 자동 복귀할 수 있도록하는 자동 원점 복귀 기능을 제공합니다. 자동 원점 복귀 기능은 ORG 신호외에 EZ, EL 신호와 조합하여 13 가지의 다양한 원점 복귀 모드를 제공하므로 시스템과 센서의 상황에 따라 적당한 모드를 선택하여 사용하면 정확한 원점 복귀 작업을 수행할 수 있습니다. 원점 복귀에 대한 자세한 사항은 “5.4.6 원점 복귀(Home Return) 함수” 편을 참조하십시오.

원점 신호의 입력 회로 및 신호 연결은 다음과 같습니다.



[그림 3-54] 원점 신호 입력 회로도 및 신호 연결법

3.3.9 Ramping down 입력 신호 (SD)

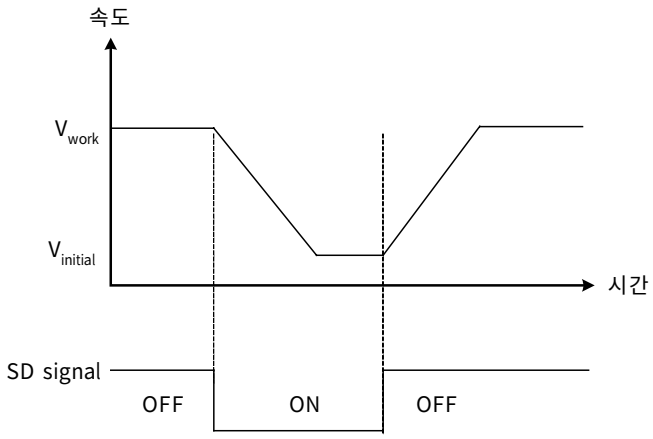
SD(Start of Deceleration) 신호는 외부 신호에 의해 강제적으로 감속을 시켜주는 신호입니다. 이 신호는 속도 모드가 Constant speed mode 일 때에는 무시되며, Trapezoidal 또는 S-curve 속도 모드에서만 적용됩니다. SD 신호 입력 기능이 Enable 되었을 때, 모션 구동 중에 SD 신호가 ON 이 되면 초기 속도값까지 감속됩니다. SD 신호가 다시 OFF 상태로 바뀔 때의 동작 방식은 SD 모드에 따라 다릅니다.

SD 신호 입력과 관련된 설정은 **COMILX_MC_SetMioCfgSD(…)** 함수에 의해서 이루어지며, 초기속도와 감속도는 **COMILX_MC_SetSpeed(…)** 함수와 **COMILX_MC_SetAccel(…)** 함수에 의해서 설정됩니다.

SD 입력 환경 설정(**COMILX_MC_SetMioCfgSD** 함수의 nSdMode, nSdLatch 파라미터)에 따른 동작 방식은 다음과 같습니다.

(1) Deceleration [nSdMode=0, nSdLatch=0]

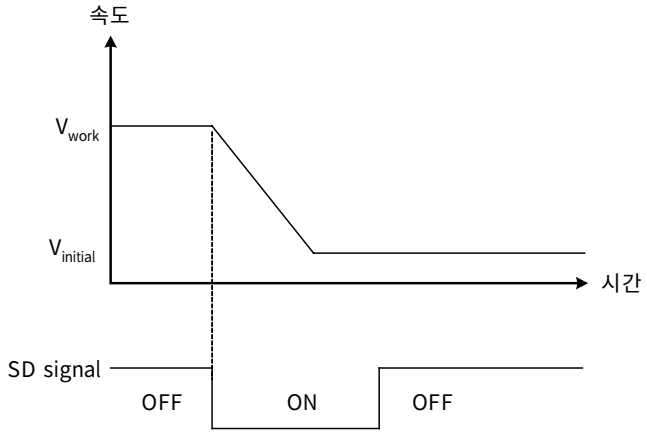
Constant speed mode 에서는 SD 신호 입력은 무시됩니다. Trapezoidal 또는 S-Curve 속도 모드에서 모션 구동중에 SD 신호가 ON 이 되면 모션의 속도가 초기 속도($V_{initial}$)로 감속됩니다. SD 신호가 다시 OFF 상태로 되면 모션의 속도는 다시 작업속도(V_{work})로 가속됩니다.



(2) Latch and Deceleration [nSdMode=0, nSdLatch=1]

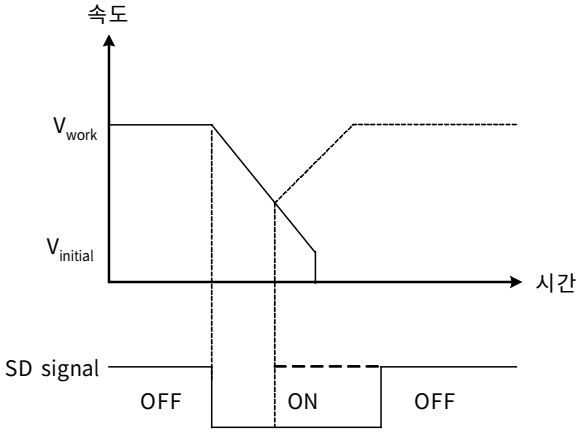
Constant speed mode 에서는 SD 신호 입력은 무시됩니다. Trapezoidal 또는 S-Curve

속도 모드에서 모션 구동중에 SD 신호가 ON 이 되면 모션의 속도가 초기 속도($V_{initial}$)로 감속됩니다. 이 모드에서는 SD 입력 신호가 Latch 되어 SD 신호가 다시 OFF 상태로 되더라도 모션의 속도는 가속되지 않습니다.



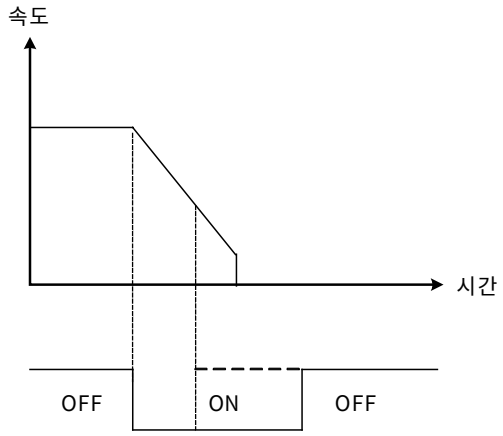
(3) Stop after Deceleration [nSdMode=1, nSdLatch=0]

Constant speed mode 에서는 SD 신호 입력은 무시됩니다. Trapezoidal 또는 S-Curve 속도 모드에서 모션 구동중에 SD 신호가 ON 이 되면 모션의 속도가 초기속도($V_{initial}$)로 감속된 후 모션 구동을 종료합니다. 단, 속도가 초기속도까지 감속되기 이전에 SD 신호가 다시 OFF 상태로 된다면 다시 작업속도로 가속된 후 모션을 지속합니다.



(4) Latch and Stop after Deceleration [nSdMode=1, nSdLatch=1]

Constant speed mode 에서는 SD 신호 입력은 무시됩니다. Trapezoidal 또는 S-Curve 속도 모드에서 모션 구동중에 SD 신호가 ON 이 되면 모션의 속도가 초기속도($V_{initial}$)로 감속된 후 모션 구동을 종료합니다. nSdLatch 가 1 로 설정되면 SD 신호의 ON 상태가 Latch 되어 감속중에 SD 신호가 OFF 로 다시 바뀌어도 가속하지 않습니다.

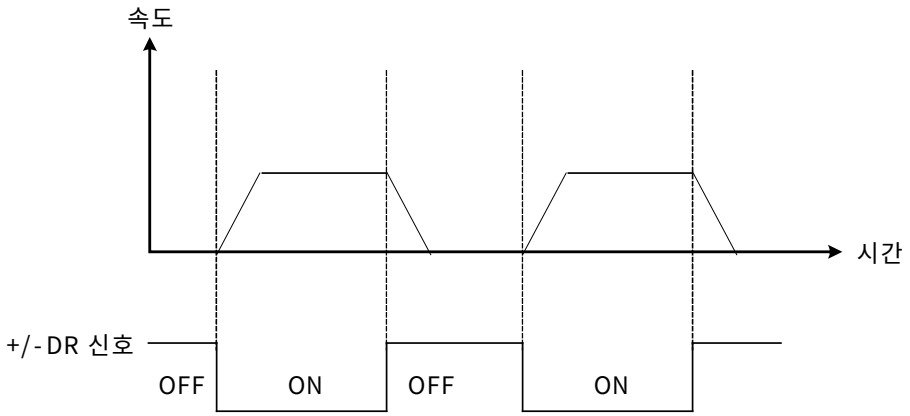


※. SD 입력 회로 및 신호 연결 방법은 원점 신호나 End Limit 신호와 동일합니다.

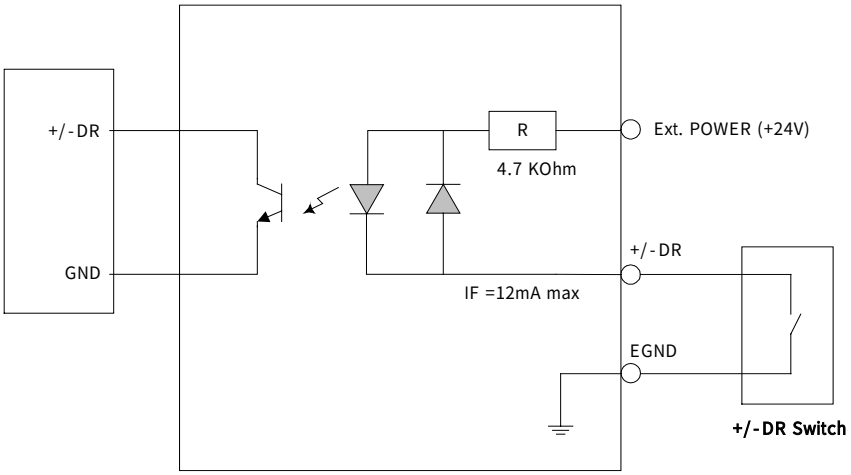
3.3.10 External Switch Operation 입력 신호 (+DR, -DR)

+DR, -DR 신호는 각 축마다 하나씩 제공되는 입력 신호로서 외부 스위치에 의해 모션의 구동을 수행할 수 있도록 하는 신호입니다. COMILX_MC_StartVMoveEx() 함수 또는 COMILX_MC_StartMoveEx() 함수를 이용하여 모션 구동 명령을 내린 경우에는 실제 모션의 동작은 +DR, -DR 신호에 의해 제어됩니다. +DR 신호는 정방향으로 모션을 구동하고 -DR 신호는 역방향으로 모션을 구동합니다.

+DR 또는 -DR 신호와 모션 구동의 관계는 다음 그림과 같습니다.



+DR 신호와 -DR 신호의 입력 회로 및 신호 연결은 다음과 같습니다.



[그림 3-55] +/- DR 신호 입력 회로도 및 신호 연결법

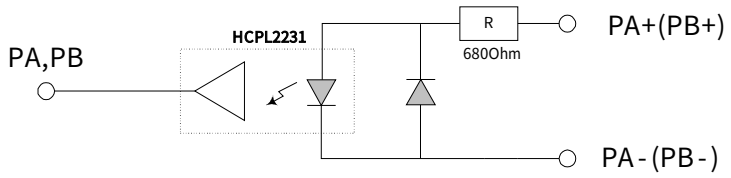
참고 COMILX_MC_StartVMoveEx() 함수 또는 COMILX_MC_StartMoveEx() 함수를 사용한 모션 구동에서만 External Switch(+/-DR)에 의해 모션이 제어됩니다.

3.3.11 Manual Pulser 입력 신호(PA, PB)

PA, PB 입력 신호는 외부에서 입력되는 펄스 신호에 의해 모션이 제어되도록 하는 신호입니다. COMI-LX50x 은 엔코더와 같은 펄스 생성 장치를 이용하여 외부에서 수동으로 모션을 제어할 수 있는 기능을 제공합니다. 앞서 설명한 External switch operation(+DR, -DR)기능이 모션의 시작과 종료만을 제어하는 기능인데 반해서 Manual pulser 모드는 외부에서 모션의 속도 및 구동 거리까지 제어할 수 있습니다. 이 기능은 수동 조작기능으로 사용하거나, 외부 모터 또는 구조물과 동기 제어를 할 때 유용하게 사용할 수 있습니다.

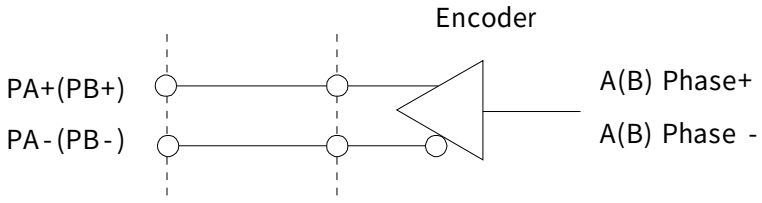
Manual pulser 기능과 관련된 자세한 사항은 “5.4.7 Manual Pulser 모드 모션 제어 함수” 편을 참조하십시오.

PA, PB 신호의 회로도는 [그림 3-56]와 같습니다.



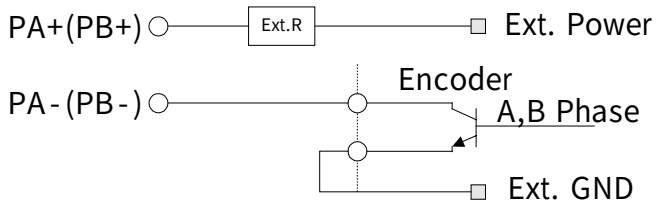
[그림 3-56] Pulser A,B 신호 입력 회로도

PA, PB 신호에 Line drive 형식으로 신호선을 연결할 때 신호 연결법은 [그림 3-57]과 같습니다.



[그림 3-57] Pulser A,B 신호 Line Drive

PA, PB 신호에 Open collector 형식으로 신호선을 연결할 때 신호 연결법은 [그림 3-58] 과 같습니다.



[그림 3-58] Pulser A,B 신호 Open Collector

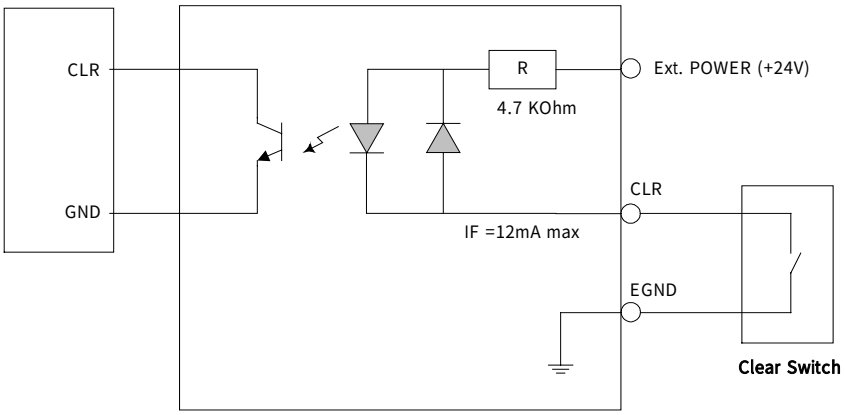
Open Collector 형으로 사용하실 때엔 외부 전원에 의해 다음표와 같은 저항 값을 사용하셔야 됩니다.

외부 전원	외부 저항 R
+5V	0
+12V	1.8K ohm
+24V	4.3K ohm

(IF = 6mA max)

3.3.12 카운터 클리어 입력 신호(CLR)

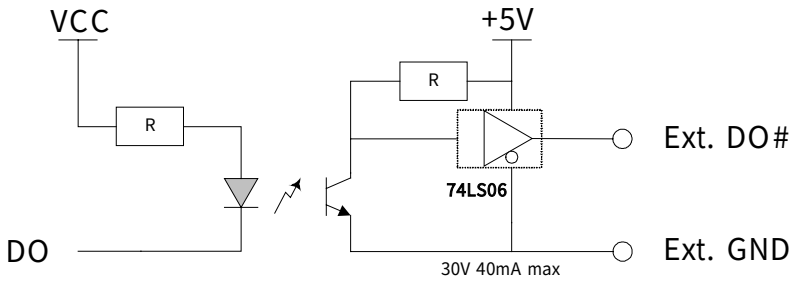
CLR 신호는 COMI-LX50x 의 각 축마다 제공되는 입력 신호로서 각 축의 카운터를 클리어해주는 신호입니다. COMI-LX50x 는 Command position counter, Actual position counter, Deviation counter 그리고 General counter 의 4 개 카운터를 제공합니다. 각 카운터가 CLR 신호에 의해 클리어되게 할지는 소프트웨어적으로 선택할 수 있습니다. 또한 CLR 신호의 입력 타입도 소프트웨어적으로 선택할 수 있습니다. 자세한 사항은 Page 301 COMILX_MC_SetMioCfgCLR() 함수 설명편을 참조하십시오.



[그림 3-59] CLR 신호 입력 회로도 및 신호 연결법

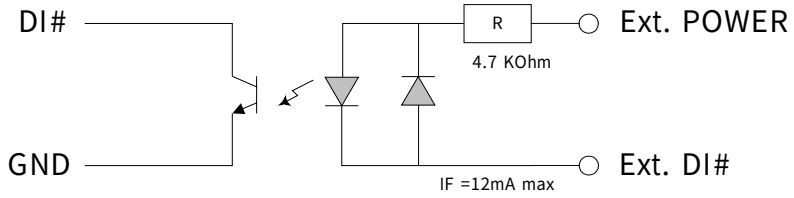
3.3.13 General Digital Input/Output(DI#, DO#)

COMI-LX50x 는 범용 디지털 입출력 채널을 제공합니다. 제공되는 채널 수는 각 제품마다 다르므로 제품 사양 설명편을 참조하십시오. [그림 3-60]은 COMI-LX50x 의 디지털 출력 회로도입니다. 각 단은 포토커플러에 의해 분리되었으며 외부 출력사양은 최대 30V, 40mA 입니다.



[그림 3-60] 디지털 출력 회로도

[그림 3-61]은 COMI-LX50x 의 디지털 입력 회로도 입니다.



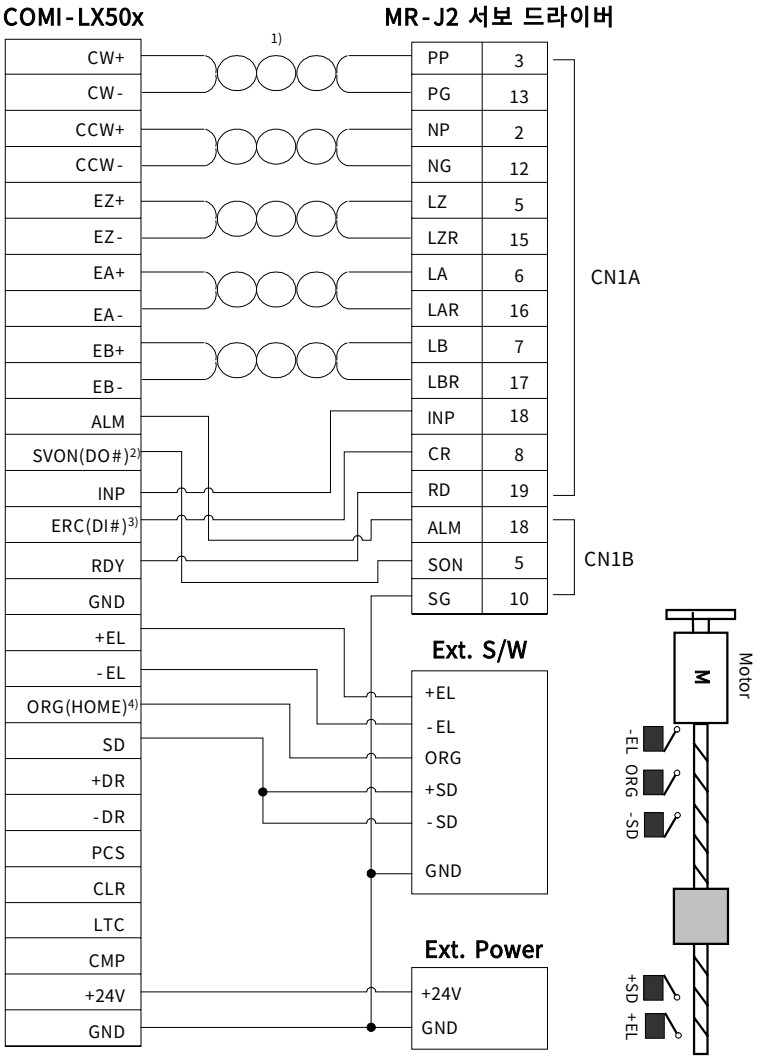
[그림 3-61] 디지털 입력 회로도

외부 전원은 24V 이고 이때의 저항 R 은 4.7 Kohm, 최대 전류량은 12mA 입니다. 포토 커플러가 동작되면 DI#핀은 Low 로 떨어집니다.

※ COMI-LX501 의 경우에 디지털 입출력 채널을 편의상 각 축에 4 채널씩 배치되어 있습니다. 그러나 실제로는 각 축별로 독립적인 디지털 입출력 채널로 구분되지 않고 16 개의 디지털 입출력 채널이 단일 그룹으로 구성된 것입니다. 따라서 프로그램을 하실 때는 일반적인 디지털 입출력 함수를 사용하며 채널 번호는 0 에서부터 15 번까지로 구성됩니다. 예를 들어 X 축에 할당된 채널들은 CH0 ~ CH3, Y 축에 할당된 채널들은 CH4 ~ CH7 로 분류됩니다.

3.4 하드웨어 배선 예

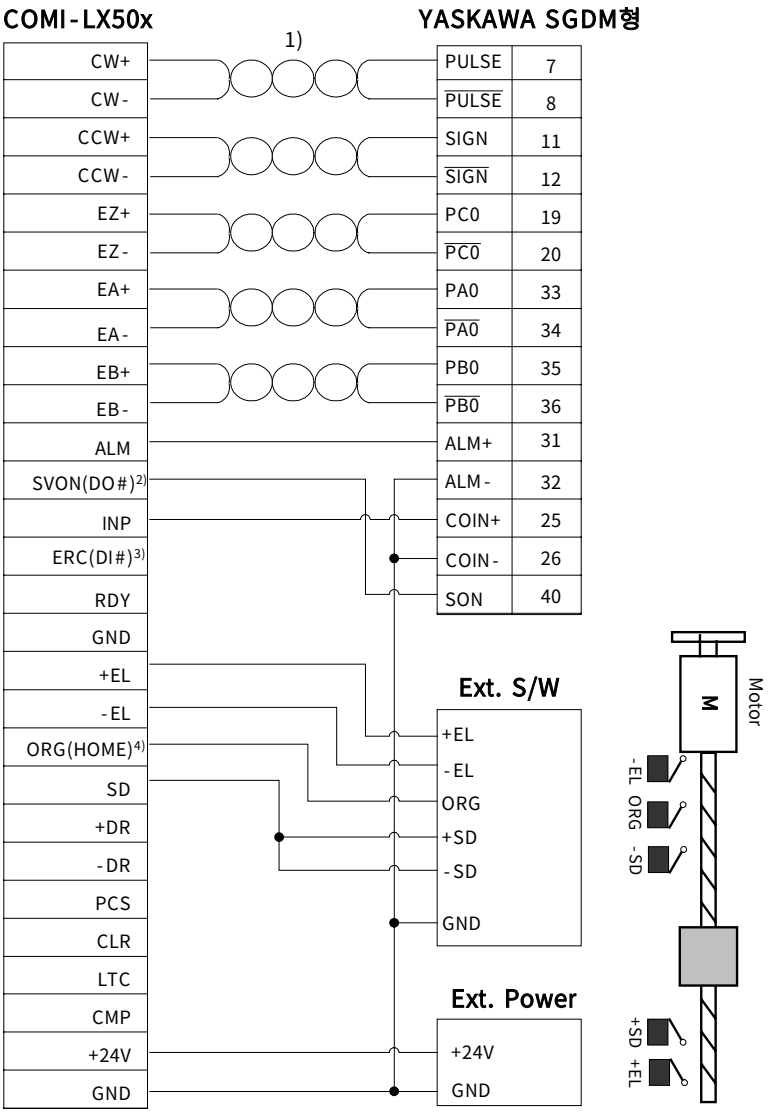
3.4.1 MITSUBISHI MR-J2 서보 드라이버 배선 예



註 1) ⟩⟨ 표시는 Twisted Pair를 의미합니다.
註 2) COMI-LX501은 SVON 출력이 따로 없고 디지털 출력 채널 중 하나를 사용합니다.
註 3) COMI-LX501은 RDY 입력이 따로 없고 디지털 입력 채널 중 하나를 사용합니다.
註 4) COMI-LXT5는 HOME, 나머지는 ORG로 표기합니다.

[그림 3-62] MITSUBISH MR-J2 시리즈 서보드라이버 배선 예

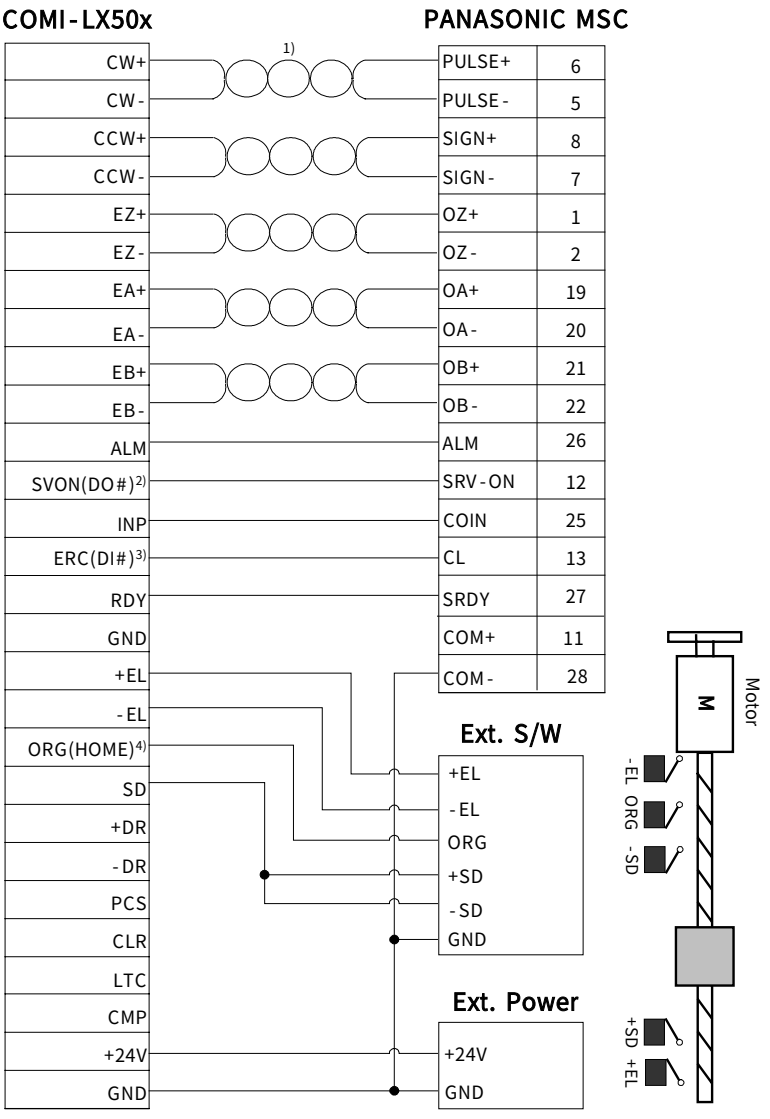
3.4.2 YASKAWA SGDM 서보 드라이버 배선 예



註 1) ⟩⟨ 표시는 Twisted Pair를 의미합니다.
註 2) COMI-LX501은 SVON 출력이 따로 없고 디지털 출력 채널 중 하나를 사용합니다.
註 3) COMI-LX501은 RDY 입력이 따로 없고 디지털 입력 채널 중 하나를 사용합니다.
註 4) COMI-LXT5는 HOME, 나머지는 ORG로 표기합니다.

[그림 3-63] YASKAWA SGDM 서보드라이버 배선 예

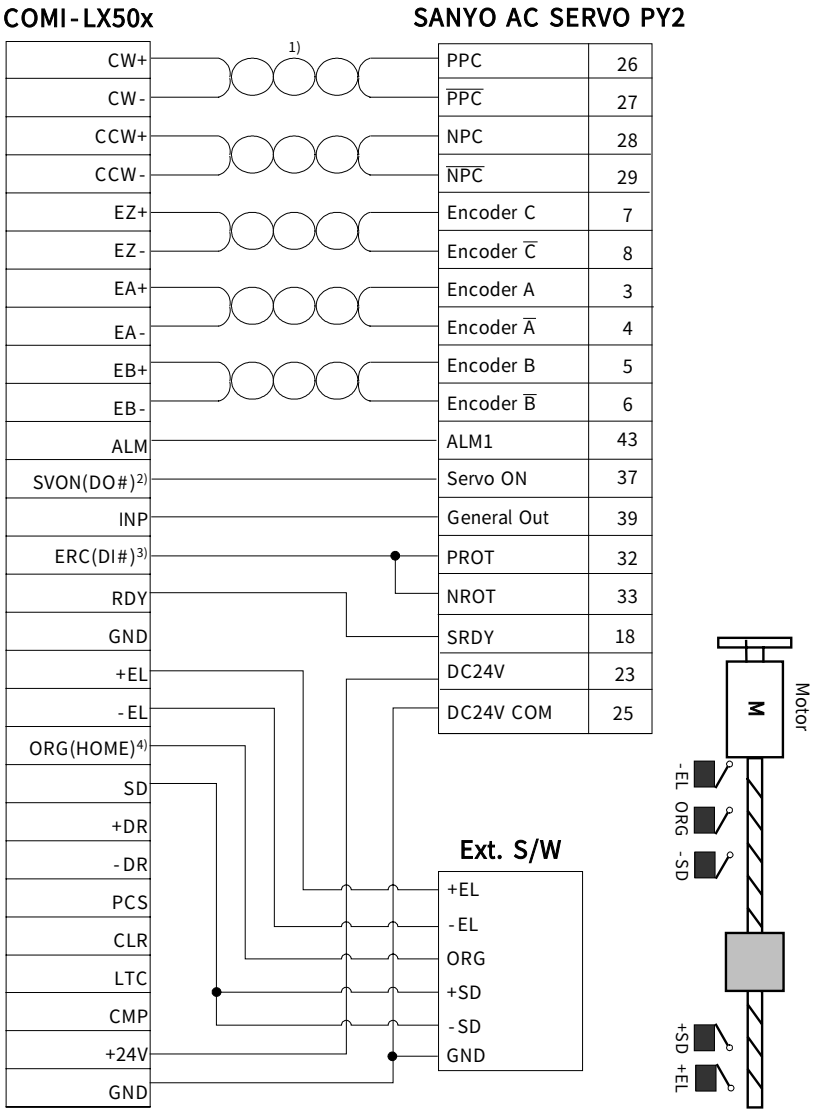
3.4.3 PANASONIC 서보 드라이버 배선 예



註 1) ⟩⟨ 표시는 Twisted Pair를 의미합니다.
註 2) COMI-LX501은 SVON 출력이 따로 없고 디지털 출력 채널 중 하나를 사용합니다.
註 3) COMI-LX501은 RDY 입력이 따로 없고 디지털 입력 채널 중 하나를 사용합니다.
註 4) COMI-LXT5는 HOME, 나머지는 ORG로 표기합니다.

[그림 3-64] PANASONIC 서보드라이버 배선 예

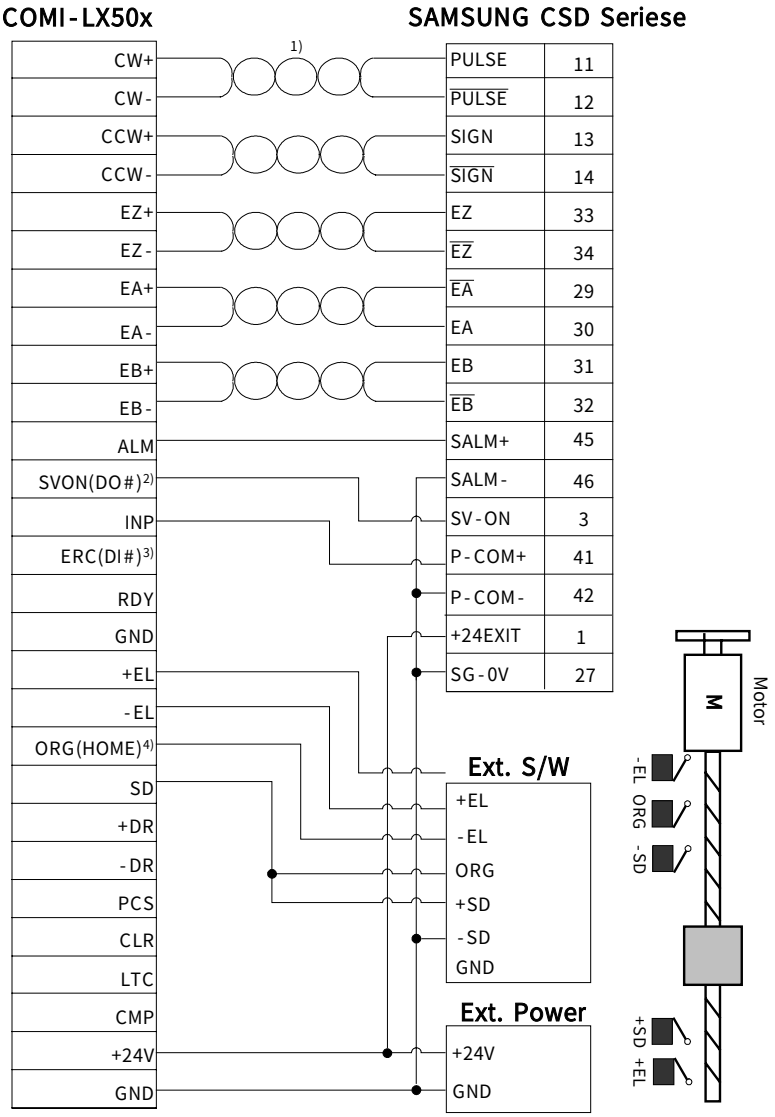
3.4.4 SANYO AC Servo PY2 배선 예



註 1) ⟩⟨ 표시는 Twisted Pair를 의미합니다.
註 2) COMI-LX501은 SVON 출력이 따로 없고 디지털 출력 채널 중 하나를 사용합니다.
註 3) COMI-LX501은 RDY 입력이 따로 없고 디지털 입력 채널 중 하나를 사용합니다.
註 4) COMI-LXT5는 HOME, 나머지는 ORG로 표기합니다.

[그림 3-65] SANYO AC Servo PY2 배선 예

3.4.5 SAMSUNG CSD Seriese 서보드라이버 배선예



註 1) ⟩⟨ 표시는 Twisted Pair를 의미합니다.
註 2) COMI-LX501은 SVON 출력이 따로 없고 디지털 출력 채널 중 하나를 사용합니다.
註 3) COMI-LX501은 RDY 입력이 따로 없고 디지털 입력 채널 중 하나를 사용합니다.
註 4) COMI-LXT5는 HOME, 나머지는 ORG로 표기합니다.

[그림 3-66] SAMSUNG CSD Seriese 배선 예

CHAPTER 4

COMI-XMaster 유틸리티 소프트웨어

본 장에서는 COMI-XMaster 유틸리티 프로그램을 설명합니다. COMI-XMaster 유틸리티 프로그램은 COMI-XMaster 는 COMI-LX 시리즈 DAQ 보드 전용 유틸리티 프로그램으로서 보드 구입시 무상으로 제공됩니다. 본 프로그램은 각 디바이스의 전반적인 기능 테스트를 손쉽게 수행할 수 있도록 하였으며 사용자 친화적이면서도 강력한 기능을 가지는 그래픽 사용자 인터페이스를 제공하여 신호의 심도 있는 분석을 용이하게 해줍니다.

4.1 Digital In/Out ····· 107

4.2 Motion Control ····· 109

CHAPTER 4 COMI-XMaster 유틸리티 프로그램

2 장에서 설명한 소프트웨어 설치를 마치면 ComiXMaster.exe 라는 유틸리티 프로그램이 설치됩니다. 이 프로그램은 모든 LX 시리즈 디바이스에 적용되는 프로그램으로서 디바이스의 전반적인 기능을 테스트해볼 수 있도록 하며, 다양한 신호 분석 및 제어 기능을 제공하는 프로그램입니다.

윈도우의 시작메뉴에서 COMIDAS-LXWCOMI-XMaster Utility for LX Seriese 메뉴를 선택하면 [그림 4-1]과 같은 화면이 나타납니다.

[그림 4-1] COMI-XMaster 주 화면

[그림 4-1]화면에서 좌측에는 PC 에 장착되어 있는 LX 시리즈 디바이스들이 리스트됩니다. 우측에 있는 버튼들은 각 프로그램 모듈을 실행할 수 있는 메뉴 버튼들입니다. DEVICE LIST 에서 디바이스를 선택하면 각 디바이스에 적용 가능한 메뉴버튼이 활성화 됩니다.

COMI-XMASTER 는 DIO, Motion Control 기능 외에도 선택된 디바이스에 따라 Analog Signal Scope, Spectrum Analyzer, Function Generator 등도 지원하지만, COMI-LX50x 디바이스만 설치되어 있을 경우, 이 기능은 사용하실 수 없습니다.

DIO, Motion Control 기능 외의 계측, 분석, 제어 기능을 원하시면 COMI-LX 제품군의 다른 모델을 구매하셔야합니다.

4.1 Digital In/Out

DIO 프로그램 모듈은 디지털 입출력 프로그램 모듈입니다. 메인 프로그램에서 DIO 버튼을 클릭하면 [그림 4-2]와 같은 화면이 나타납니다. COMI-LX501 의 경우, 모션 제어 전용보드는 각 축당 4 개씩, 총 16 개씩의 디지털 입력과 출력 채널을 제공합니다.[1] COMI-LX501 모션제어 전용보드에서 제공하는 디지털 입출력 채널은 다른 모션 제어 보드의 기능과 달리 축에 종속된 개념으로 생각하지 않고 CH0 ~ CH15 의 채널 번호로 구성되는 단일 채널 그룹으로 간주됩니다. 즉, X 축에 있는 4 개의 디지털 입력 채널들은 D/I CH0 ~ D/I CH3 으로, Y 축에 있는 4 개의 디지털 입력 채널들은 D/I CH4 ~ D/I CH7 로 간주됩니다. 마찬가지로 Z 축과 U 축에 있는 디지털 입력 채널들도 각각 D/I CH8 ~ D/I CH11, D/I CH12 ~ D/I CH15 로 간주됩니다. 디지털 출력의 채널 개념 또한 디지털 입력의 채널 개념과 동일합니다. 이는 ㈜커미조아에서 개발한 다른 디지털 입출력 디바이스와의 호환성(동일한 라이브러리 함수를 쓸수 있도록 하기 위함)을 고려하였기 때문이며, 이는 COMI-LX502/4/8 에도 동일하게 적용됩니다.

[그림 4-2] 디지털 입출력 프로그램 모듈 화면

프로그램의 상단은 디지털 출력을 제어하는 부분이면 하단은 디지털 입력을 모니터링하는 부분입니다.

[1] COMI-LX501 : 16 채널, COMI-LX502 : 3 채널, COMI-LX504 : 6 채널, COMI-LX508 : 12 채널

자동 출력

자동 출력은 주기적으로 출력의 State 를 Toggle 시켜주는 기능입니다. State 변경 주기는 “주기” 항목에서 결정하며 “반복회수” 항목의 값이 0 보다 큰 값이면 지정한 회수만큼만 출력을 내보냅니다.

입력 로그

입력 로그는 디지털 입력 채널의 State 가 변화하면 리스트 박스에 디지털 입력 채널의 값들을 시간과 함께 기록하는 기능입니다. 입력 로그 기능을 사용하려면 “Enable logging” 항목을 체크하십시오.

4.2 Motion Control

Motion Control 프로그램 모듈은 COMI-LX50x 모션제어(Motion Control) 전용 보드의 기능을 테스트해볼 수 있도록하기 위하여 제작된 프로그램입니다. [그림 4-1]의 메인 프로그램에서 Motion 버튼을 클릭하면 [그림 4-3]과 같은 화면이 나타납니다.

[그림 4-3] Motion Control 메인 화면

Axis 선택

이 항목은 제어대상 축을 선택하는 항목입니다. 콤보박스에서 제어하고자 하는 축을 선택하십시오.

Operation Mode

이 항목은 모션의 동작 모드를 선택하는 항목입니다. Operation Mode 에서 선택할 수 있는 모드는 다음의 3 가지 모드입니다.

□ **Velocity Move**

이 모드에서는 사용자가 'Stop' 또는 'E-Stop' 버튼을 클릭할 때까지 모션을 계속 수행합니다. Velocity Move 모드에서 '+Move' 버튼을 클릭하면 설정된 속도 프로파일에 따라서 (+)방향으로 모션을 수행하고 '-Move' 버튼을 클릭하면 설정된 속도 프로파일에 따라서 (-)방향으로 모션을 수행합니다.

□ **Relative In-Position**

이 모드에서는 상대좌표값(Distance)에서 지정하는 거리만큼 이동을 수행합니다. '+Move' 버튼을 클릭하면 설정된 속도 프로파일에 따라서 (+)방향으로 지정한

거리만큼 이동하고 '-Move' 버튼을 클릭하면 설정된 속도 프로파일에 따라서 (-)방향으로 지정한 거리만큼 이동합니다.

□ Absolute In-Position

이 모드에서는 절대좌표값(Position1, Position2)에서 지정하는 절대좌표로의 이동을 수행합니다. '+Move' 버튼을 클릭하면 'Position1' 항목에서 지정하는 좌표로 이동하고 'Position2' 항목에서 지정하는 좌표로 이동합니다.

속도 프로파일 설정

□ Velocity Profile

이 콤보박스에서 속도 모드를 지정합니다. 속도모드는 Constant, Trapezoidal, S-curve 모드 중에서 선택할 수 있습니다. 각 속도모드의 특성은 3 장의 COMILX_MC_SetSpeedMode 함수 설명편을 참조하십시오.

□ Initial Speed

모션의 초기 속도를 지정합니다. 여기서 속도의 단위는 PPS(Pulses/sec)입니다.

□ Work Speed

모션의 작업 속도를 지정합니다. 여기서 속도의 단위는 PPS(Pulses/sec)입니다.

□ Acceleration

모션의 가속도를 지정합니다. 여기서 가속도의 단위는 PPS/sec 입니다. 속도모드가 Trapezoidal 또는 S-curve 모드로 지정된 경우에는 모션은 이 항목에서 지정한 가속도로 초기 속도로부터 작업 속도로 가속을 수행합니다. 이 값이 0 인 경우에는 가속구간 없이 바로 작업속도로 모션을 제어합니다. 단, 이 항목은 Contant Speed Mode 에서는 무시됩니다.

□ Deceleration

모션의 감속도를 지정합니다. 여기서 감속도의 단위는 PPS/sec 입니다. 속도모드가 Trapezoidal 또는 S-curve 모드로 지정된 경우에는 모션은 이 항목에서 지정한 감속도로 작업 속도로부터 초기 속도로 감속을 수행한 후 모션을 정지합니다. 단, 이 항목은 Contant Speed Mode 에서는 무시됩니다.

□ S-section(acc)

가속 구간에서의 S-cruve 범위를 설정합니다. 이 값은 속도모드가 S-curve 모드로 지정된 경우에만 적용됩니다. S-curve 범위에 대한 자세한 사항은 3 장의 COMILX_MC_SetScurve 함수 설명편을 참조하십시오.

□ S-section(dec)

감속 구간에서의 S-cruve 범위를 설정합니다. 이 값은 속도모드가 S-curve 모드로 지정된 경우에만 적용됩니다. S-curve 범위에 대한 자세한 사항은 3 장의

COMILX_MC_SetScurve 함수 설명편을 참조하십시오.

4.2.1 Motion Status 창

[그림 4-3] Motion Control 메인 화면에서 좌측 중앙에 있는 타원형의 버튼을 클릭하면 [그림 4-4]과 같이 Motion Status 를 감시할 수 있는 창이 확장되어 나타납니다.

[그림 4-4] Motion Status 창이 좌측에 확장되어 나타난 화면

위치 감시

Motion Status 창 화면에서 'Position' 이라는 이름으로 그룹화된 컨트롤 항목들은 모션의 명령 위치(Command position), 실제 위치(Feedback position) 그리고 두 위치간의 오차값을 표시합니다.

속도 감시

Motion Status 창 화면에서 'Velocity' 라는 이름으로 그룹화된 컨트롤 항목들은 모션의 명령 속도(Command speed), 실제 속도(Feedback speed) 를 표시합니다.

모션의 상태 감시

Motion Status 창 화면에서 'Motion Status' 라는 이름으로 그룹화된 컨트롤 항목들은 모션의 진행 상태 및 각 I/O 상태를 표시합니다. 이 중에서 텍스트 표시부는 현재 진행되고 있는 모션의 과정 또는 상태를 텍스트로 표시합니다. 그리고 하단의 LED 는 모션의 관련된 각 I/O 의 상태를 표시합니다.

4.2.2 Digital I/O 창

[그림 4-3] Motion Control 메인 화면에서 우측 중앙에 있는 타원형의 버튼을 클릭하면 [그림 4-5]와 같이 Digital Input/Output 을 감시 또는 제어할 수 있는 창이 나타납니다. 입력과 출력이 각각 16 채널씩 표시되어 있습니다. 모든 모션 보드를 하나의 프로그램에서 처리하기 위한 것이며, 각 보드에 따라 입력, 출력 채널수는 차이가 있습니다.[1] 예를 들어, COMI-LX502 의 경우, 입력/출력 채널수가 각각 3 개이므로, 프로그램에서는 0 번 채널에서 2 번채널까지만 사용하시면 됩니다.

[그림 4-5] Digital In/Out 창이 좌측에 확장되어 나타난 화면

4.2.3 Graph 창

[그림 4-3] Motion Control 메인 화면에서 하단의 우측에 있는 타원형의 버튼을 클릭하면 [그림 4-6]과 같이 위치 및 속도를 그래프로 표시해볼 수 있는 창이 나타납니다.

[1] COMI-LX501 : 16 채널, COMI-LX502 : 3 채널, COMI-LX504 : 6 채널, COMI-LX508 : 12 채널

[그림 4-6] Graph 창이 하단에 확장되어 나타난 화면

Plot in motion

[그림 4-6]화면에서 좌측에 있는 “Plot in motion” 항목은 모션이 진행되는 순간에만 Plot 을 할 것인지 아니면 항상 Plot 을 할 것인지를 결정합니다. 이 항목을 체크하면 모션이 진행되는 순간에만 Plot 을 하게되며 모션이 진행되지 않는 순간에는 Plotting 을 하지 않습니다..

Plot target

[그림 4-6]화면에서 좌측에 있는 “Plot target” 항목은 그래프에 Plotting 할 대상을 선택합니다. “Position” 을 선택하면 그래프에는 Command Position 과 Actual Position 이 Plotting 되며, “Velocity” 를 선택하면 그래프에는 Command speed 와 Actual speed 가 Plotting 되게 됩니다.

그래프 제어

그래프 상단에 있는 여러 가지 버튼들을 이용하여 그래프를 제어하여 데이터를 자세히 분석할 수 있습니다.

□ **축 스크롤(** **)**

축 스크롤 버튼을 클릭하고 원하는 축을 마우스로 왼쪽클릭한 상태에서 스크롤하

면 축이 스크롤됩니다.

□ **축 확대/축소( )**

축 확대/축소 버튼을 클릭하고 원하는 축을 마우스로 왼쪽클릭한 상태에서 스크롤하면 축이 확대/축소됩니다.

□ **영영 확대( )**

영역 확대 버튼을 클릭하고 그래프의 영역을 드래그하여 선택하면 선택된 영역이 확대되어 그래프에 나타납니다. 영역 확대가 된 상태에서 이 버튼을 다시 클릭하여 OFF 상태로 만들면 그래프는 원래의 축 상태로 복원됩니다.

□ **커서 1 표시( )**

커서는 그래프에서 데이터의 값을 확인하기 위해 사용되는 십자선(+)을 의미합니다. 본 그래프는 2 개의 커서를 제공하는데 이 버튼은 이중에서 첫번째 커서를 보이거나 혹은 보이지 않거나할 수 있습니다. 사용자가 커서 1 버튼을 ON 시키면 그래프에는 십자선이 하나 나타나며 십자점이 가리키는 지점의 X, Y 데이터를 화면에 표시할 수 있습니다.

□ **커서 2 표시( )**

커서는 그래프에서 데이터의 값을 확인하기 위해 사용되는 십자선(+)을 의미합니다. 본 그래프는 2 개의 커서를 제공하는데 이 버튼은 이중에서 두 번째 커서를 보이거나 혹은 보이지 않거나할 수 있습니다. 사용자가 커서 1 버튼을 ON 시키면 그래프에는 십자선이 하나 나타나며 십자점이 가리키는 지점의 X, Y 데이터를 화면에 표시할 수 있습니다

□ **Pause/Resume( )**

이 버튼은 그래프가 자동 스크롤되는 것을 멈추거나 재개하는데 사용됩니다. 이 버튼이 OFF 인 상태에서는 그래프가 자동 스크롤됩니다. 이 버튼이 ON 인 상태에서는 그래프가 자동 스크롤되지 않습니다. 또한 그래프의 축을 확대, 축소, 스크롤한 경우에는 자동으로 Pause 됩니다. 따라서 이러한 경우에는 이 버튼을 OFF 시켜서 자동 스크롤을 Enable 시켜야 자동 스크롤됩니다.

4.2.4 Pulse Input/Output 설정

[그림 4-3] 화면에서 'Set In/Out' 버튼을 클릭하면 [그림 4-7]화면이 나타납니다. 이 화면에서 사용자는 Pulse 입/출력 신호의 환경을 설정할 수 있습니다.

[그림 4-7] Pulse Input/Output 설정 대화상자

[그림 4-7]화면에서 각 항목의 의미는 다음과 같습니다.

Encoder Input Mode

엔코더 피드백(Encoder Feedback) 신호의 입력모드를 설정합니다. 여기서 설정할 수 있는 입력모드는 다음과 같습니다.

Item Value	Meaning
1X A/B	1X A/B (1 채배 엔코더 입력 모드)
2X A/B	2X A/B (2 채배 엔코더 입력 모드)
4X A/B	4X A/B (4 채배 엔코더 입력 모드)
CW/CCW	CW/CCW (A 펄스 - 카운트 증가, B 펄스 - 카운트 감소)

Encoder Input Logic

엔코더 피드백(Encoder Feedback) 신호의 입력 로직을 설정합니다.

Pulse Output Mode

커맨드 출력(Command Output) 펄스의 출력모드 번호를 설정합니다. 설정가능한 출력모드는 다음과 같습니다.

Value	출력 형태			
	(+) 방향 운전 시		(-) 방향 운전 시	
	CW pin	CCW pin	CW pin	CCW pin
0		(High)		(Low)
1		(High)		(Low)
2		(Low)		(High)
3		(Low)		(High)
4		(High)	(High)	
5		(Low)	(Low)	

4.2.5 Motion I/O 환경설정

[그림 4-3] 화면에서 'MIO Config' 버튼을 클릭하면 [그림 4-8] 화면이 나타납니다. 이 화면에서는 사용자가 모션에 관련된 입출력 신호에 대한 환경을 설정할 수 있도록 합니다. [그림 4-8]화면에서 각 항목의 의미는 다음과 같습니다.

ALARM

ALARM(ALM) 입력 신호에 대한 입력 로직(Logic) 및 동작 모드를 설정합니다.

입력 로직은 "Acitve Low"로 선택하면 신호의 입력이 Low 상태일 때 ON 이 되며, "Acitve High"로 선택하면 신호의 입력이 High 상태일 때 ON 이 됩니다.

동작모드는 ALARM 신호가 ON 이 되었을 때 모션을 즉시 정지할 것인가, 아니면 감속 후 정지할 것인가를 결정합니다.

[그림 4-8] MIO Config 설정 대화상자

INP

INP 입력 신호에 대한 입력 로직(Logic) 및 INP 신호의 적용 여부를 설정합니다.

“Enable INP” 를 선택하면 모션의 완료를 INP 신호에 의해 결정합니다. COMI-LX50x 에서 펄스 출력을 완료하고, INP 신호가 ON 상태이어야만 모션이 완료된 것으로 간주됩니다.

입력 로직은 “Acitve Low” 로 선택하면 신호의 입력이 Low 상태일 때 ON 이 되며, “Acitve High” 로 선택하면 신호의 입력이 High 상태일 때 ON 이 됩니다.

ERC

ERC 출력신호의 출력 로직 및 펄스폭을 설정합니다. ERC 신호는 서보 드라이버의 편차 카운터를 클리어하는 신호로써 설정에 따라서 원점 복귀가 완료되면 출력됩니다.

EL

+/-EL(External Limit) 입력신호에 대한 모션의 동작 모드를 설정합니다. +/-EL 신호가 ON 이 되면 모션은 정지합니다. 이 때 설정에 따라서 즉시 정지 또는 감속 후 정지합니다.

ORG

ORG(HOME) 신호의 입력 로직을 설정합니다. ORG 입력 신호는 원점 복귀시에 원점을 감지하는 센서의 입력 신호입니다.

입력 로직은 “Acitve Low” 로 선택하면 신호의 입력이 Low 상태일 때 ON 이 되며, “Acitve High” 로 선택하면 신호의 입력이 High 상태일 때 ON 이 됩니다.

EZ

EZ 신호의 입력 로직을 설정합니다. EZ 입력 신호는 엔코더의 Z-Phase 펄스 신호로써 원점 복귀시에 사용될 수 있습니다.

입력 로직은 “Acitve Low”로 선택하면 신호의 입력이 Low 상태일 때 ON 이 되며, “Acitve High”로 선택하면 신호의 입력이 High 상태일 때 ON 이 됩니다.

CHAPTER 5

C/C++ 라이브러리

본 장에서는 사용자가 COMI-LX 시리즈 디바이스를 이용하여 프로그램을 구현할 때 유용하게 사용될 수 있는 C/C++ 라이브러리에 대한 내용을 설명합니다. ㈜커미조아에서는 사용자가 사용하기 쉬우면서도 각 디바이스의 기능을 극대화할 수 있도록 심혈을 기울여 라이브러리를 제작하였습니다.

5.1 라이브러리 사용 방법 ····· 121
5.2 라이브러리 및 디바이스 시작/종료 ····· 124
5.3 디지털 입출력(Digital Input/Output) ····· 130
5.4 모션 제어(Motion Control) ····· 141

CHAPTER 5 C/C++라이브러리

본 장에서는 사용자가 COMI-LX 시리즈 디바이스를 이용하여 프로그램을 구현할 때 유용하게 사용될 수 있는 라이브러리에 대한 내용을 설명합니다. ㈜커미조아에서 제공하는 COMI-LX 시리즈용 C/C++ 라이브러리는 모든 종류의 COMI-LX 시리즈 디바이스에 적용 가능한 통합 라이브러리입니다. 각 함수들은 그 기능에 따라 분류되어 수록되었습니다.

본 장에서는 우선 먼저 각 컴파일러에 따라 ㈜커미조아의 라이브러리를 사용하는 방법을 설명하고, 각 함수에 대한 설명을 수록하였습니다. COMI-LX 시리즈 디바이스의 윈도우용 C/C++ 라이브러리는 ComiLX.sys 라는 디바이스 드라이버 프로그램과 ComidasLX.dll 파일로 구성됩니다. ComiLX.sys 는 COMI-LX 시리즈 디바이스의 통합 드라이버이며 모든 종류의 COMI-LX 시리즈 디바이스를 제어하는 프로그램입니다. ComidasLX.dll 파일은 ComiLX.sys 프로그램과 사용자 프로그램과의 통신을 담당해주는 DLL(Dynamic Link Library)프로그램으로서 이 프로그램을 통해 드라이버를 Access 할 수 있습니다. 드라이버 파일은 디바이스 설치 시에 윈도우의 적정 디렉토리에 설치되고, DLL 파일은 소프트웨어 설치 시에 윈도우의 적정 디렉토리에 설치됩니다. 따라서 이들을 사용자가 따로 설치할 필요는 없습니다.

본 매뉴얼은 라이브러리 Version 2.6 기준으로 작성된 것입니다. 사용자들에게 보다 나은 기능을 제공하기 위하여 라이브러리는 업그레이드될 수 있습니다. 라이브러리의 버전업이 이루어지는 경우에는 버전업된 라이브러리와 함께 수정 또는 추가된 사항을 일목 요연하게 정리한 문서를 제공합니다. 인터넷이 가능한 사용자는 Smart update 기능을 사용하여 드라이버, 라이브러리, 유틸리티 등의 버전 상태를 체크할 수 있고 각 프로그램을 최신 버전으로 업데이트하는 작업을 간단히 수행할 수 있습니다.

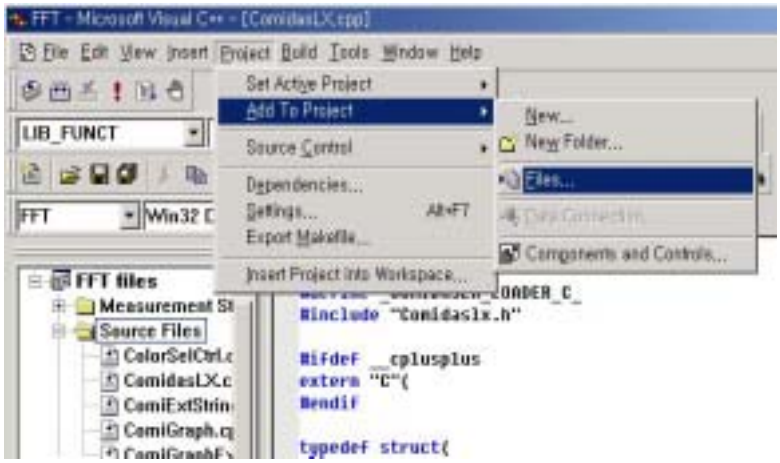
5.1 라이브러리 사용 방법

5.1.1 Visual C/C++에서의 라이브러리 사용법

Visual C/C++ 에서 본 라이브러리를 사용하려면 다음의 절차에 따라 사용하시면 됩니다.

COMIDAS_ROOTWWindowsWC_CppWLib 폴더에 있는 ComidasCommon.h 와 ComidasLX.h 그리고 ComidasLX.cpp 파일을 사용자 소스 폴더에 복사합니다. 여기서 COMIDAS_ROOT 는 COMIDAS 파일들이 설치된 루트 디렉토리를 가리킵니다.

ComidasCommon.h 와 ComidasLX.h 그리고 ComidasLX.cpp 파일을 사용자 프로젝트에 추가합니다. 그러기 위해서는 [그림 5-1]과 같이 Project ≫ Add To Project ≫ Files 메뉴를 선택하여 해당 파일들을 추가하면 됩니다.



[그림 5-1] Visual C++에서 프로젝트에 헤더 및 소스파일 추가하기

만일 프로젝트 옵션 중에 “Use precompiled header file” 옵션을 선택하였다면 ComidasLX.cpp 파일에 다음의 구문을 추가하여야 합니다.

#include “stdafx.h”

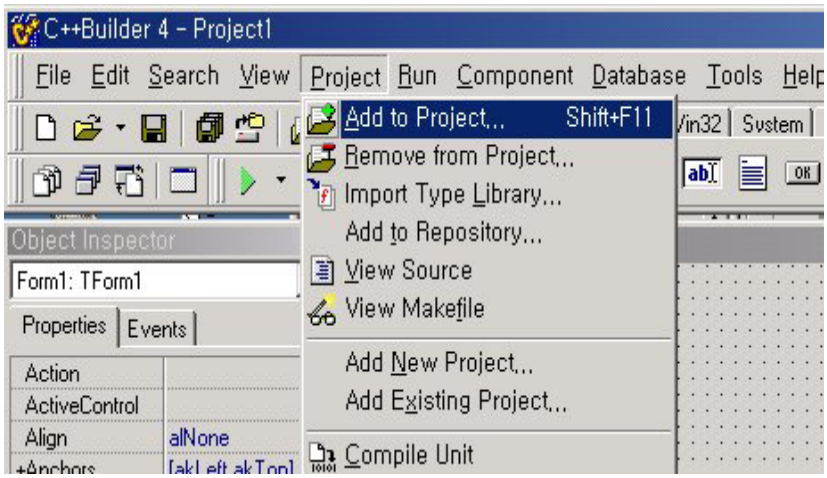
라이브러리 함수를 사용하고자 하는 소스 파일에 #include “ComidasLX.h” 구문을 추가합니다.

5.1.2 Borland C++ Builder 에서의 라이브러리 사용법

Borland C++ Builder 에서 본 라이브러리를 사용하려면 다음의 절차에 따라 사용하시면 됩니다.

COMIDAS_ROOT₩Windows₩C_Cpp₩Lib 폴더에 있는 ComidasCommon.h 와 ComidasLX.h 그리고 ComidasLX.cpp 파일을 사용자 소스 폴더에 복사합니다. 여기서 COMIDAS_ROOT 는 COMIDAS 파일들이 설치된 루트 디렉토리를 가리킵니다.

ComidasCommon.h 와 ComidasLX.h 그리고 ComidasLX.cpp 파일을 사용자 프로젝트에 추가합니다. 그러기 위해서는 [그림 5-2]와 같이 Project ≫ Add To Project 메뉴를 선택하여 해당 파일들을 추가하면 됩니다.



[그림 5-2] C++ Builder 에서 프로젝트에 헤더 및 소스파일 추가하기

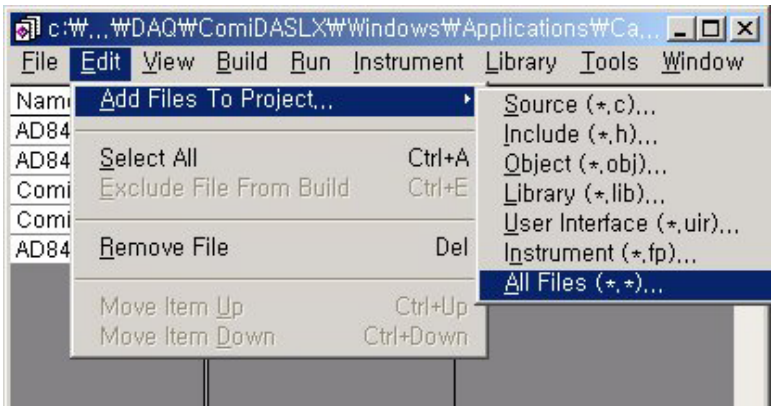
라이브러리 함수를 사용하고자 하는 소스 파일에 #include “ComidasLX.h” 구문을 추가합니다.

5.1.3 Labwindos CVI 에서의 라이브러리 사용법

Labwindows CVI 에서 본 라이브러리를 사용하려면 다음의 절차에 따라 사용하시면 됩니다.

1. COMIDAS_ROOT₩Windows₩C_Cpp₩Lib 폴더에 있는 ComidasCommon.h 와 ComidasLX.h 그리고 ComidasLX.c 파일을 사용자 소스 폴더에 복사합니다. 여기서 COMIDAS_ROOT 는 COMIDAS 파일들이 설치된 루트 디렉토리를 가리킵니다.

2. ComidasCommon.h 와 ComidasLX.h 그리고 ComidasLX.c 파일을 사용자 프로젝트에 추가합니다. 그러기 위해서는 [그림 5-3]과 같이 Edit ≫ Add Files To Project ≫ All Files 메뉴를 선택하여 해당 파일들을 추가하면 됩니다.



[그림 5-3] Labwindows 에서 프로젝트에 헤더 및 소스파일 추가하기

3. 라이브러리 함수를 사용하고자 하는 소스 파일에 #include “ComidasLX.h” 구문을 추가합니다.

주의 사항

Labwindows 에서 본 라이브러리를 사용하기 위해서는 Labwindows 를 설치할 때 Windows SDK Library 설치가 선택되었어야 합니다. Labwindows 에서 Windows SDK Library 를 설치하기 위해서는 Custom 설치를 통하여 Windows SDK Library 옵션을 체크하여야 설치됩니다. 라이브러리 함수를 사용하고자 하는 소스 파일에서는 #include “ComidasLX.h” 구문 이전에 #include “windows.h” 구문이 추가되어야 합니다(소스의 가장 처음 부분에 위치해야함).

5.2 라이브러리 및 디바이스 시작/종료

이 단원에서는 COMIDAS 라이브러리와 각 디바이스를 로드(Load)/언로드(Unload)하는 함수들을 소개합니다. 이 함수들은 COMIDAS 라이브러리를 사용하기 위해 필수적으로 적용되어져야 할 함수들입니다. 이에 관련된 함수들의 리스트는 다음과 같습니다.

▣ COMILX_LoadDll

함수 원형

BOOL COMILX_LoadDll (void)

함수 설명

이 함수는 라이브러리 프로그램을 로드(load)합니다. 라이브러리 프로그램은 DLL(Dynamic Link Libray)형태의 프로그램이며 사용자 프로그램에게 디바이스를 제어할 수 있는 함수를 제공합니다. 그러나 실제로는 디바이스를 제어하는 역할은 디바이스 드라이버가 수행하여 주며, DLL 은 사용자 프로그램이 디바이스 드라이버를 ACCESS 할 수 있도록 해주는 매개체 역할을 해줍니다.

반환 값

성공적으로 DLL 을 Load 했는지를 알려주는 값을 Return 합니다.

Value	Meaning
0	DLL 을 Load 하는데 실패하였음을 의미합니다.
1	DLL 을 성공적으로 Load 했음을 의미합니다.

참 고

이 함수는 그 어떤 다른 COMIDAS Library 함수들 보다도 먼저 수행되어져야 합니다. 보통 프로그램 시작 시에 수행하면 됩니다. 여러 개의 디바이스를 제어하더라도 이 함수는 한번만 실행 되어야 합니다.

사용예

```
// 프로그램 시작 부분 : 다른 함수들이 사용되기 전에 수행 //
if(!COMILX_LoadDll()){
printf("Comidas.dll load failure");
   exit(0);
}
……
```

```
……
// 프로그램 종료 부분 //
COMILX_UnloadDll();
```

▣ COMILX_UnloadDll

함수 원형

```
void COMILX_UnloadDll (void)
```

함수 설명

이 함수는 라이브러리 프로그램을 언로드(load)합니다.

참 고

이 함수는 보통 프로그램 종료 시에 수행하면 됩니다.

사용예

```
// 프로그램 시작 부분 : 다른 함수들이 사용되기 전에 수행 //
if(!COMILX_LoadDll()){
printf("Comidas.dll load failure");
   exit(0);
}
……
……

// 프로그램 종료 부분 //
COMILX_UnloadDll();
```

▣ COMILX_LoadDevice

함수 원형

HANDLE **COMILX_LoadDevice** (COMIDAS_DEVID deviceID, ULONG instance)

함수 설명

이 함수는 하나의 COMIDAS 디바이스를 로드(load)합니다. 각 디바이스를 제어하기 위해서는 먼저 이 함수를 이용하여 해당 디바이스에 대한 핸들을 얻어와야 합니다.

매개 변수

▶ *deviceID* : 각 디바이스의 고유한 아이디값입니다. 이 값은 각 디바이스 명칭과 동일하게 적어주면 됩니다. 예를 들어 COMI-LX501 모션제어 전용 보드의 경우 COMI_LX501 을 적어주면 됩니다. 각 디바이스 아이디는 ComidasLX.h 헤더파일의 앞부분에 정의되어 있으므로 ComidasLX.h 파일을 참조하기 바랍니다.

▶ *Instance* : 동일 deviceID 를 가진 여러 개의 디바이스를 구분하기 위한 값입니다. 같은 종류의 디바이스가 동일 컴퓨터에 여러 개 장착된다면 장착된 순서대로 instance 번호가 부여됩니다. Instance 번호는 0 번부터 차례로 부여됩니다. 예를 들어 2 개의 COMI-LX501 보드가 장착되어 있다면 처음 장착된 보드의 instance 값은 0 이 되며, 두 번째 장착된 보드의 instance 값은 1 이 됩니다.

반환값

이 함수는 디바이스 핸들을 반환합니다. 이 값은 디바이스를 제어하는 각 함수의 첫 번째 파라미터로 사용됩니다. 만일 이 값이 INVALID_HANDLE_VALUE 이면 디바이스 로딩이 실패한 것입니다.

참 고

이 함수는 제어하고자 하는 디바이스 수만큼 수행되어야 합니다.

사용예

2 대의 COMI-LX501 디바이스를 이용하여 프로그램 할 때의 코드 예

```
HANDLE hDevice1, hDevice2;

if(!COMILX_LoadDll()){
printf("Comidas.dll load failure");
   exit(0);
}

hDevice1 = COMILX_LoadDevice (COMI_LX501, 0);
if(hDevice1 == INVALID_HANDLE_VALUE){
   printf("Can't load first COMI-LX501 device!");
```

```
   COMILX_UnloadDll();
   exit(0);
}

hDevice2 = COMILX_LoadDevice (COMI_LX501, 1);
if(hDevice2 == INVALID_HANDLE_VALUE){
   printf("Can't load second COMI-LX501 device!");
   COMILX_UnloadDll();
   exit(0);
}

……
……
COMILX_UnloadDevice(hDevice1);
COMILX_UnloadDevice(hDevice2);
COMILX_UnloadDll();
```

▣ COMILX_UnloadDevice

함수 원형

void **COMILX_UnloadDevice** (HANDLE hDevice)

함수 설명

이 함수는 하나의 COMIDAS 디바이스를 언로드(unload)한다.

매개 변수

▶ *hDevice* : 디바이스 핸들값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값입니다.

사용예

```
HANDLE hDevice;

if(!COMILX_LoadDll()){
printf("Comidas.dll load failure");
   exit(0);
}

hDevice = COMILX_LoadDevice (COMI_LX501, 0);
if(hDevice == INVALID_HANDLE_VALUE){
   printf("Can't load COMI-LX501 device!");
   COMILX_UnloadDll();
   exit(0);
}
……
……
COMILX_UnloadDevice(hDevice);
COMILX_UnloadDll();
```

5.3 디지털 입출력(Digital Input/Output)

이 단원에서는 Digital Input 과 Output 에 관한 함수를 소개합니다. 일반적으로 Digital Input 은 스위치(Switch)의 상태를 읽어들이는데 사용되고, Digital Output 은 스위치의 상태를 제어하는데 사용됩니다.

COMI-LX50x 모션제어 보드는 모션 전용 I/O 인터페이스외에 사용자가 범용적으로 사용할 수 있는 디지털 입력/출력 채널을 제공합니다.[1)]

COMI-LX50x 모션제어 전용보드에서 지원하는 여러가지 함수들이 대부분 축(X,Y,Z,U)에 종속하는 함수로 구성된것과 달리 디지털 입출력 기능은 각축에 종속되지 않은 것으로 간주됩니다. 이는 ㈜커미조아에서 개발한 다른 디지털 입출력 디바이스와의 호환성(동일한 라이브러리 함수를 쓸수 있도록 하기 위함)을 고려하였기 때문입니다.

따라서, COMI-LX501 의 경우, 모션제어 전용보드에서 제공하는 디지털 입출력 채널은 CH0 ~ CH15 의 채널 번호로 구성되는 단일 채널 그룹으로 간주됩니다. 즉, X 축 터미널에 있는 4 개의 디지털 입력 채널들은 D/I CH0 ~ D/I CH3 으로, Y 축에 있는 4 개의 디지털 입력 채널들은 D/I CH4 ~ D/I CH7 로 간주됩니다. 마찬가지로 Z 축과 U 축에 있는 디지털 입력 채널들도 각각 D/I CH8 ~ D/I CH11, D/I CH12 ~ D/I CH15 로 간주됩니다. 디지털 출력의 채널 개념 또한 디지털 입력의 채널 개념과 동일합니다. 물론, 이러한 개념들은 COMI-LX502/4/8 보드에도 동일하게 적용됩니다.

[1)] COMI-LX501 : 16 채널, COMI-LX502 : 3 채널, COMI-LX504 : 6 채널, COMI-LX508 : 12 채널

▣ COMILX_DI_GetOne

함수 원형

int COMILX_DI_GetOne (HANDLE hDevice, int ch)

함수 설명

이 함수는 지정한 Digital Input 채널의 Status 를 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

▶ *ch* : Digital Input 채널번호. 채널번호는 0 부터 시작합니다.

반환값

Digital Input 채널의 Status.

Value	Meaning
0	OFF
1	ON

예 제

이 프로그램은 COMILX_DO_PutOne(..)과 COMILX_DI_GetOne(..) 함수를 사용하여 D/O CH0 을 통하여 STATUS 를 반복적으로 반전하면서 출력을 내보내고 D/I CH0 의 STATUS 를 반복적으로 체크하는 것입니다. D/O CH0 와 D/I CH0 를 서로 연결한다면 D/O CH0 의 출력을 D/I CH0 를 통하여 반복적으로 체크할 수 있습니다.

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

#define DEV_ID    COMI_LX501
#define DO_CH 0
#define DI_CH 0

void main (void)
{
   HANDLE hDevice;
   int do_state=0, di_state;

   if(!COMILX_LoadDll()){
```

```
            printf("ComidasLX.dll load failure");
            printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
            _getch();
            exit(0);
        }

        hDevice = COMILX_LoadDevice (DEV_ID, 0);
        if(hDevice == INVALID_HANDLE_VALUE){
            printf("Can't load specified device!");
            printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
            _getch();
            COMILX_UnloadDll();
            exit(0);
        }

        printf("DIO 테스트를 시작하려면 아무키나 누르십시오.\n");
        printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
        _getch();

        while(!kbhit())
        {
            do_state ^= 1;  // state 반전
            COMILX_DO_PutOne (hDevice, DO_CH, do_state); // Put D/O

            /* Get D/I and print on screen */
            di_state = COMILX_DI_GetOne(hDevice, DI_CH);
            printf("Status of D/I CH0 = %d\n", di_state);
            Sleep(500);
        }


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

▣ COMILX_DI_GetAll

함수 원형

DWORD COMILX_DI_GetAll (HANDLE hDevice)

함수 설명

이 함수는 해당 디바이스의 모든 Digital Input 채널의 Status 를 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

반환값

32 개의 채널에 대한 Input Status 를 32 비트 값으로 반환합니다. 각비트는 비트 순서와 일치하여 각 채널의 ON/OFF 상태를 나타냅니다. 단, 디바이스에 따라 32 채널 미만의 Digital 채널을 지원하는 경우에는 BIT0 부터 해당 채널 수 만큼의 비트만 사용하시면 됩니다.

예 제

이 프로그램은 COMILX_DI_GetAll(..)과 COMILX_DO_PutAll(..) 함수를 사용하여 D/I 와 D/O 의 8 채널을 동시에 콘틀롤하는 예제입니다.

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

#define DEV_ID    COMI_LX501

void main (void)
{
   HANDLE hDevice;
   ULONG do_states=0, di_states;
   int di_each[8], i;

   if(!COMILX_LoadDll()){
       printf("ComidasLX.dll load failure");
       printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
       _getch();
       exit(0);
   }

   hDevice = COMILX_LoadDevice (DEV_ID, 0);
   if(hDevice == INVALID_HANDLE_VALUE){
       printf("Can't load specified device!");
```

```
            printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
            _getch();
            COMILX_UnloadDll();
            exit(0);
        }

        printf("DIO 테스트를 시작하려면 아무키나 누르십시오.\n");
        printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
        _getch();

        while(!kbhit())
        {
            do_states = ~do_states;  // 모든 D/O 채널 On/Off state 반전
            COMILX_DO_PutAll (hDevice, do_states); // Put D/O

            /* Get D/I and print on screen */
            di_states = COMILX_DI_GetAll(hDevice);
            /* di_states 는 전채널의 state를 담고 있다.      */
            /* 각 채널의 상태를 얻으려면 다음과 같이 bit mask   */
            /* 를 하면 된다.                                 */
            for(i=0; i<8; i++)
                di_each[i] = (di_states >> i) & 0x1;
            printf("States of DI0 ~ DI7 = %d %d %d %d %d %d %d %d\n",
                   di_each[0], di_each[1], di_each[2], di_each[3],
                   di_each[4], di_each[5], di_each[6], di_each[7]);
            Sleep(500);
        }


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

▣ COMILX_DO_PutOne

함수 원형

void COMILX_DO_PutOne (HANDLE hDevice, int ch, int status)

함수 설명

이 함수는 지정한 Digital Output 채널에 지정한 Status로 출력을 내보냅니다.

매개 변수

▶ *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

▶ *ch* : Digital Output 채널번호. 채널 번호는 0부터 시작한다.

▶ *status* : 출력 Status. 0 - OFF, 1 - ON.

예 제

이 프로그램은 COMILX_DO_PutOne(..)과 COMILX_DI_GetOne(..) 함수를 사용하여 D/O CH0 을 통하여 STATUS 를 반복적으로 반전하면서 출력을 내보내고 D/I CH0 의 STATUS 를 반복적으로 체크하는 것입니다. D/O CH0 와 D/I CH0 를 서로 연결한다면 D/O CH0 의 출력을 D/I CH0 를 통하여 반복적으로 체크할 수 있습니다.

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

#define DEV_ID    COMI_LX501
#define DO_CH 0
#define DI_CH 0

void main (void)
{
   HANDLE hDevice;
   int do_state=0, di_state;

   if(!COMILX_LoadDll()){
       printf("ComidasLX.dll load failure");
       printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
       _getch();
       exit(0);
   }

   hDevice = COMILX_LoadDevice (DEV_ID, 0);
   if(hDevice == INVALID_HANDLE_VALUE){
       printf("Can't load specified device!");
```

```
            printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
            _getch();
            COMILX_UnloadDll();
            exit(0);
        }

        printf("DIO 테스트를 시작하려면 아무키나 누르십시오.\n");
        printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
        _getch();

        while(!kbhit())
        {
            do_state ^= 1;  // state 반전
            COMILX_DO_PutOne (hDevice, DO_CH, do_state); // Put D/O

            /* Get D/I and print on screen */
            di_state = COMILX_DI_GetOne(hDevice, DI_CH);
            printf("Status of D/I CH0 = %d\n", di_state);
            Sleep(500);
        }


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

▣ COMILX_DO_PutAll

함수 원형

void COMILX_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)

함수 설명

이 함수는 해당 디바이스의 모든 Digital Output 채널에 출력을 내보낸다.

매개 변수

▶ *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

▶ *dwStatuses* : 모든 Digital Output 채널의 출력 Status 를 나타내는 32 bit 값. 이 값의 각 비트의 값이 각 채널의 Status 를 나타냅니다.

예 제

이 프로그램은 COMILX_DI_GetAll(..)과 COMILX_DO_PutAll(..) 함수를 사용하여 D/I 와 D/O 의 8 채널을 동시에 콘틀롤하는 예제입니다.

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

#define DEV_ID    COMI_LX501

void main (void)
{
   HANDLE hDevice;
   ULONG do_states=0, di_states;
   int di_each[8], i;

   if(!COMILX_LoadDll()){
       printf("ComidasLX.dll load failure");
       printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
       _getch();
       exit(0);
   }

   hDevice = COMILX_LoadDevice (DEV_ID, 0);
   if(hDevice == INVALID_HANDLE_VALUE){
       printf("Can't load specified device!");
       printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
       _getch();
       COMILX_UnloadDll();
       exit(0);
```

```
    }

    printf("DIO 테스트를 시작하려면 아무키나 누르십시오.\n");
    printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
    _getch();

    while(!kbhit())
    {
        do_states = ~do_states;  // 모든 D/O 채널 On/Off state 반전
        COMILX_DO_PutAll (hDevice, do_states); // Put D/O

        /* Get D/I and print on screen */
        di_states = COMILX_DI_GetAll(hDevice);
        /* di_states 는 전채널의 state를 담고 있다.      */
        /* 각 채널의 상태를 얻으려면 다음과 같이 bit mask   */
        /* 를 하면 된다.                                 */
        for(i=0; i<8; i++)
            di_each[i] = (di_states >> i) & 0x1;
        printf("States of DI0 ~ DI7 = %d %d %d %d %d %d %d %d\n",
               di_each[0], di_each[1], di_each[2], di_each[3],
               di_each[4], di_each[5], di_each[6], di_each[7]);
        Sleep(500);
    }


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

▣ COMILX_DO_GetOne

함수 원형

int COMILX_DO_GetOne (HANDLE hDevice, int ch)

함수 설명

이 함수는 지정한 Digital Output 채널의 현재 출력값을 알아볼 수 있도록 하는 함수입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

▶ *ch* : Digital Output 채널번호. 채널 번호는 0 부터 시작한다.

반환값

Digital Output 채널의 현재 출력 상태

Value	Meaning
0	OFF
1	ON

▣ COMILX_DO_GetAll

함수 원형

DWORD COMILX_DO_GetAll (HANDLE hDevice)

함수 설명

이 함수는 CH0 에서 CH31 까지의 Digital Output 채널의 현재 출력값을 알아볼 수 있도록 하는 함수입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

반환값

32 개의 채널에 대한 현재 출력 상태를 32 비트 값으로 반환합니다. 각비트는 비트 순서와 일치하여 각 채널의 ON/OFF 상태를 나타냅니다. 단, 디바이스에 따라 32 채널 미만의 Digital 채널을 지원하는 경우에는 BIT0 부터 해당 채널 수 만큼의 비트만 사용하시면 됩니다.

5.4 모션 제어(Motion Control)

이 단원에서 설명하는 모션제어 기능은 ㈜커미조아에서 개발한 COMI-LX50x 모션제어(Motion Control) 전용 보드에서 제공하는 기능입니다. 따라서 이 단원에서 소개되는 모든 함수는 COMI-LX50x 보드에만 적용가능합니다.

COMI-LX50x 모션제어 보드는 펄스 구동 방식에 의하여 스텝모터나 서보 모터의 정밀 위치제어 및 속도제어를 하기 위한 기능을 제공합니다. 스텝모터나 서보 모터의 위치 제어는 디지털 출력등의 일반적인 방법으로 펄스를 만들어서 제어할 수도 있습니다. 그러나 이러한 방식으로는 정확한 속도 제어, 가/감속 제어, 두 축 이상의 보간(Interpolation)등은 거의 불가능합니다. 모션제어 보드는 단일축의 위치제어는 물론이고 정밀 속도 제어, 가/감속 제어, 두 축 이상의 보간 등을 모두 자동화하여 사용자들이 이러한 기능을 아주 쉽게 구현할 수 있도록 해줍니다.

5.4.1 모션 초기화 및 환경설정 함수

이 단원에서는 모션을 초기화하고 모션을 수행하기 이전에 환경을 설정하는 함수들을 소개합니다. 이와 관련된 함수들은 다음과 같습니다.

void COMILX_MC_Reset (HANDLE hDevice) 모션 제어 보드의 하드웨어와 소프트웨어적인 모든 상태를 리셋합니다.
void COMILX_MC_SetBlockingMode (HANDLE hDevice, BOOL bBlocking) 모션이 완료될때까지 루프를 돌 때 윈도우 또는 시스템 이벤트를 처리할 수 있도록할지를 결정하는 함수
void COMILX_MC_SetOutputMode (HANDLE hDevice, int nChannel, int nOutputMode) 현재 설정된 Command 펄스의 출력 모드를 얻어옵니다.
int COMILX_MC_GetOutputMode (HANDLE hDevice, int nChannel) 현재 설정된 Command 펄스의 출력 모드를 반환합니다.
void COMILX_MC_SetInputMode (HANDLE hDevice, int nChannel, int nInputMode, int nPulseLogic) Feedback 펄스의 입력 모드를 설정합니다.
void COMILX_MC_GetInputMode (HANDLE hDevice, int nChannel, int *pInputMode, int *pPulseLogic) 현재 설정된 Feedback 펄스의 입력 모드를 얻어옵니다.
void COMILX_MC_SetSpeedRange(HANDLE hDevice, int nChannel, double fMaxSpeed) 모션에 적용할 수 있는 최저/최고 속도를 제한합니다.
void COMILX_MC_SetUnitDistance (HANDLE hDevice, int nChannel, double fUnitDist) 논리적 단위 거리에 대한 펄스 수를 설정합니다.
double COMILX_MC_GetUnitDistance (HANDLE hDevice, int nChannel) 현재 설정된 논리적 단위 거리에 대한 펄스 수를 반환합니다.
void COMILX_MC_SetUnitSpeed (HANDLE hDevice, int nChannel, double fUnitSpeed) 논리적 단위 속도에 대한 펄스 출력 속도(PPS)를 설정합니다.
double COMILX_MC_GetUnitSpeed (HANDLE hDevice, int nChannel) 현재 설정된 논리적 단위 속도에 대한 펄스 출력 속도(PPS)를 반환합니다.
double COMILX_MC_SetInOutRatio (HANDLE hDevice, int nChannel, double fRatio) Feedback 펄스와 Command 펄스의 분해능 비율(Resolution ratio)을 설정합니다.

▣ COMILX_MC_Reset

함수 원형

void COMILX_MC_Reset (HANDLE hDevice)

함수 설명

이 함수는 모션 제어 보드의 하드웨어와 소프트웨어적인 모든 상태를 리셋합니다. 프로그램 초기와 종료 부분에서는 이 함수를 사용하여 모션을 리셋시켜주는 것이 좋습니다.

매개 변수

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

예 제

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

void main()
{
   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);
   //Set constant speed mode //
   COMILX_MC_SetSpeedMode(hDevice, 0, 0);
   // Set speed as 5000 PPS //
   COMILX_MC_SetSpeed(hDevice, 0, 0, 1000);
   COMILX_MC_Move(hDevice, 0, 5000); // 60 rpm의 속도로 100mm 이동

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

▣ COMILX_MC_SetBlockingMode

함수 원형

void COMILX_MC_SetBlockingMode (HANDLE hDevice, BOOL bBlocking)

함수 설명

이 함수는 Blocking 모드를 결정합니다. COMILX_MC_Move()와 같은 함수들은 Motion을 시작해준 후에 내부적으로 while loop 를 돌면서 모션이 완료되는 것을 체크한 후 모션이 완료되면 함수에서 리턴(Return)됩니다. Blocking 이란 while loop 를 수행하게 되면 키보드, 마우스, 타이머 이벤트 등의 윈도우 이벤트가 처리되지 않는 현상을 말합니다.

Blocking 모드를 FALSE 로 하면 COMILX_MC_Move () 등의 함수 내부에서 while loop를 돌때에 PeekMessage 와 TranslateMessage 등의 API 함수를 수행하여 루프 수행중에도 키보드, 마우스, 타이머 이벤트 등과 같은 윈도우 이벤트나 메시지를 처리할 수 있도록 합니다.

매개 변수

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

▶ *bBlocking* : Blocking 모드를 결정합니다.

참고

다음과 같은 함수들은 Motion이 완료될 때까지 함수에서 Return되지 않습니다.

COMILX_MC_Move, COMILX_MC_MoveTo, COMILX_MC_Line, COMILX_MC_LineTo, COMILX_MC_Arc, COMILX_MC_ArcTo

예 제

다음의 예제는 소스의 간결성을 위하여 **Console application** 형태로 구성되었으며 **COMILX_MC_SetBlockingMode()** 함수의 사용법만 예로 들기 위해서 만들어진 것입니다. 실제로는 **COMILX_MC_SetBlockingMode()**는 프로그램이 윈도우 기반이거나 쓰레드 기반일 때 그 효과를 발휘할 수 있습니다.

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

#ifndef FALSE
#define FALSE 0
```

```
#endif

void main()
{
   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_SetBlockingMode (hDevice, FALSE);
   //Set constant speed mode //
   COMILX_MC_SetSpeedMode(hDevice, 0, 0);
   // Set speed as 5000 PPS //
   COMILX_MC_SetSpeed(hDevice, 0, 0, 1000);
   // Blocking이 않되도록 설정되었으므로 Move()함수가 내부적으로 //
   // Loop를 돌면서 모션이 완료되기를 기다릴 때에도 키보드나 마우스 //
   // 등의 시스템 및 윈도우 이벤트를 처리할 수 있다.              //
   COMILX_MC_Move(hDevice, 0, 5000);

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

▣ COMILX_MC_SetOutputMode

함수 원형

void COMILX_MC_SetOutputMode (HANDLE hDevice, int nChannel, int nOutputMode)

함수 설명

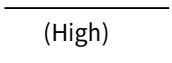
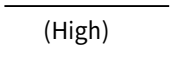
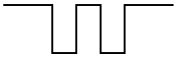
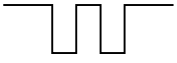
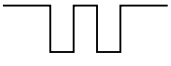
Command 펄스의 출력 모드를 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nOutputMode* : Command 펄스의 출력 모드를 설정합니다. 출력 모드는 다음과 같이 6가지로 설정할 수 있습니다.

Value	출력 형태			
	(+) 방향 운전 시		(-) 방향 운전 시	
	CW pin	CCW pin	CW pin	CCW pin
0		 (High)		(Low)
1		 (High)		(Low)
2		(Low)		(High)
3		(Low)		(High)
4		(High)	(High)	
5		(Low)	(Low)	

▣ COMILX_MC_GetOutputMode

함수 원형

int COMILX_MC_GetOutputMode(HANDLE hDevice, int nChannel)

함수 설명

현재 설정된 Command 펄스의 출력 모드를 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재 설정된 Command 펄스의 출력 모드를 반환합니다. 반환되는 값은 0~5 의 정수이며 각 값의 의미는 COMILX_MC_SetOutputMode() 함수를 참조하십시오.

▣ COMILX_MC_SetInputMode

함수 원형

void COMILX_MC_SetInputMode(HANDLE hDevice, int nChannel, int nInputMode, int nPulseLogic)

함수 설명

Feedback 펄스의 입력 모드를 설정합니다. Feedbakc 펄스는 실제 모터 또는 구조물의 구동거리를 체크하기 위하여 사용되며 거의 대부분 엔코더 입력을 사용합니다. 사용자는 4 가지 형태의 Feedback 펄스의 입력모드를 설정할 수 있습니다. 또한 이 함수는 입력 펄스의 입력 로직(Logic)을 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nInputMode* : Feedback 펄스의 입력 모드를 설정합니다. 입력 모드는 다음과 같이 4 가지로 설정할 수 있습니다.

Value	Meaning
0	1X A/B (1 채배 엔코더 입력 모드)
1	2X A/B (2 채배 엔코더 입력 모드)
2	4X A/B (4 채배 엔코더 입력 모드)
3	CW/CCW (A 펄스 - 카운트 증가, B 펄스 - 카운트 감소)

▶ *nPulseLogic* : 입력 펄스의 로직(Logic)을 설정합니다.

Value	Meaning
0	Normal low
1	Normal high

▣ COMILX_MC_GetInputMode

함수 원형

void COMILX_MC_GetInputMode(HANDLE hDevice, int nChannel, int *pInputMode, int *pPulseLogic)

함수 설명

현재 설정된 Feedback 펄스의 입력 모드를 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pInputMode* : Feedback 펄스의 입력 모드를 반환 받을 변수의 주소값(포인터). 반환되는 값은 0~3 의 정수이며 각 값의 의미는 COMILX_MC_SetInputMode() 함수를 참조하십시오. 이 값이 NULL 이면 입력 모드를 반환하지 않습니다.

▶ *pPulseLogic* : Feedback 펄스의 입력 로직(Logic)을 반환 받을 변수의 주소값(포인터). 반환되는 값은 0~1 의 정수이며 각 값의 의미는 COMILX_MC_SetInputMode() 함수를 참조하십시오. 이 값이 NULL 이면 입력 로직을 반환하지 않습니다.

▣ COMILX_MC_SetSpeedRange

함수 원형

void COMILX_MC_SetSpeedRange(HANDLE hDevice, int nChannel, double fMaxSpeed)

함수 설명

모션에 적용할 수 있는 최저/최고 속도를 제한합니다. 이 함수는 실제적으로는 출력 펄스의 주파수 범위를 설정하는 역할을 합니다. 출력 펄스의 주파수는 최대 6.5MHz 까지 설정가능하며 기본적으로 설정되는 주파수 범위는 10Hz ~ 655,350Hz 입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fMaxSpeed* : 모션의 최고 속도를 설정합니다. 이 값에 따라 최저 속도는 자동으로 설정됩니다. 이 값의 단위는 COMILX_MC_SetUnitSpeed()함수에 의하여 설정된 단위가 됩니다. 만일 COMILX_MC_SetUnitSpeed()함수를 사용하지 않았다면 PPS 단위가 됩니다. fMaxSpeed 값에 따라 설정되는 출력 펄스의 주파수 범위를 몇 가지 예로 들면 아래의 표와 같습니다.

fMaxSpeed 값(Hz)	출력 펄스의 주파수 범위(Hz)
6,553.5	0.1 to 6,553.5
13,107	0.2 to 13,107
32,767.5	0.5 to 32,767.5
65,535	1 to 65,535
131,070	2 to 131070
327,650	5 to 327,650
655,350	10 to 655,350
1,310,700	20 to 1,310,700
3,276,750	50 to 3,276,750
6,553,500	100 to 6,553,500

참 고

□ 최저 속도는 최대 속도 설정에 따라 다음과 같은 식에 의하여 자동으로 결정됩니

다.

$$V_{\min} = \frac{V_{\max}}{65535}$$

예를 들어 V_{max}(fMaxSpeed) 값을 1000000(Hz)로 지정하였다면 V_{min} 값은 1000000/65535 = 15.26(Hz) 으로 자동 설정됩니다. 따라서 사용자는 15.26 ~ 1000000 (Hz)의 범위에서 속도를 설정할 수 있습니다. 만일 COMILX_MC_SetSpeed()등의 함수를 이용하여 속도를 설정할 때 최저 속도보다 작은 값으로 설정하면 자동으로 최저 속도로 조정되어 적용됩니다.

▣ COMILX_MC_SetUnitDistance

함수 원형

void COMILX_MC_SetUnitDistance(HANDLE hDevice, int nChannel, double fUnitDist)

함수 설명

논리적 단위 거리에 대한 펄스 수를 설정합니다. 여기서 논리적 단위 거리라 함은 Move 함수에서 사용하는 거리 또는 위치에 대한 단위량을 의미합니다. 이 함수를 사용하여 특별히 지정하지 않는 경우에는 논리적 단위 거리에 대한 펄스 수는 1 로 사용됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fUnitDist* : 논리적 단위 거리에 대한 펄스 수를 지정합니다.

참고

모터나 모터 드라이버의 특성에 따라 단위 거리 이동에 필요한 펄스의 수는 다르게 됩니다. 또는 사용자의 특성에 따라 이동량에 대한 단위가 다를 수 있습니다. 즉, 어떤 사용자는 이동량의 단위를 각도로 표현하는 것이 용이할 수 있고 어떤 사용자는 mm 또는 cm 등으로 표현하는 것이 용이할 수 있습니다. COMILX_MC_SetUnitDistance 함수는 사용자가 이동량의 단위를 결정하도록 하는 함수입니다. 이 함수를 다음의 예를 참고하여 사용하십시오.

Ex 1) 1 회전에 필요한 펄스 수가 3600 펄스인 경우에 이동량의 단위를 1° 로 하고자 한다면 fUnitDist 값을 10 으로 하면 됩니다.

Ex 2) 1mm 이송에 펄스 수가 20 펄스인 경우에 이동량의 단위를 1mm 로 하고자 한다면 fUnitDist 값을 20 으로 하면 됩니다.

예제

1mm 이동하는데 필요한 펄스수가 100 펄스이고 1 회전에 필요한 펄스수가 10000 펄스일때 거리의 단위를 mm 로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

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

void main()
```

```
{
   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

   // Set 100 pulses for unit distance             //
   // 이 예제에서는 1mm 이동에 필요한 펄스수를 100 펄스로   //
   // 가정하고 단위 거리를 1mm 로 설정한 것이다.          //
   COMILX_MC_SetUnitDistance(hDevice, 0, 100);
   // Set 10000/60(=166.7) PPS for unit speed //
   // 이 예제에서는 1 회전에 필요한 펄스수를 10000         //
   // 펄스로 가정하고 단위 속도를 1rpm 로 설정한 것이다.  //
   COMILX_MC_SetUnitSpeed(hDevice, 0, 10000./60);

   COMILX_MC_SetSpeed(hDevice, 0, 0, 60); // Set speed as 60 rpm
//
   COMILX_MC_SetSpeedMode(hDevice, 0, 0); // Set contant speed
mode //

   COMILX_MC_Move(hDevice, 0, 100); // 60 rpm 의 속도로 100mm 이동
//

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

▣ COMILX_MC_GetUnitDistance

함수 원형

double COMILX_MC_GetUnitDistance(HANDLE hDevice, int nChannel)

함수 설명

현재 설정된 논리적 단위 거리에 대한 펄스 수를 반환합니다. 여기서 논리적 단위 거리라 함은 Move 함수에서 사용하는 거리 또는 위치에 대한 단위량을 의미합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재 설정된 단위 거리에 대한 펄스 수를 반환합니다.

▣ COMILX_MC_SetUnitSpeed

함수 원형

void COMILX_MC_SetUnitSpeed(HANDLE hDevice, int nChannel, double fUnitSpeed)

함수 설명

논리적 단위 속도에 대한 실제 펄스 출력 속도(PPS)를 설정합니다. 여기서 논리적 단위 속도라 함은 속도 지정함수에서 사용하는 속도 또는 가속도에 대한 단위량을 의미합니다. 이 함수를 사용하여 특별히 지정하지 않는 경우에는 단위 속도에 대한 펄스 출력 속도는 1PPS 로 사용됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fUnitSpeed* : 단위 속도에 대한 펄스 출력 속도(PPS)를 지정합니다.

참고

사용자의 특성에 따라 속도에 대한 단위가 다를 수 있습니다. 즉, 어떤 사용자는 속도 단위를 RPM 으로 표현하는 것이 용이할 수 있고 어떤 사용자는 m/sec 로 표현하는 것이 용이할 수 있습니다. COMILX_MC_SetUnitSpeed 함수는 사용자가 속도의 단위를 결정하도록 하는 함수입니다. 이 함수를 다음의 예를 참고하여 사용하십시오.

Ex 1) 1 회전에 필요한 펄스 수가 3600 펄스인 경우에 속도의 단위를 RPM 으로 하고자 한다면 fUnitDist 값을 3600/60, 즉 60 PPS 로 설정합니다(여기서 60 으로 나누는 것은 RPM 은 분당 회전수이므로 초당 3600/60 펄스를 출력해야 1 분에 3600 펄스가 나가기 때문입니다).

Ex 2) 1cm 이송에 필요한 펄스 수가 1000 펄스인 경우에 이동량의 단위를 cm/sec 로 하고자 한다면 fUnitDist 값을 1000 PPS 로 설정합니다.

관련 함수 :

COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve,
COMILX_MC_SetSpeedMx, COMILX_MC_SetAccelMx, COMILX_MC_SetScurveMx,
COMILX_MC_GetActualSpeed, COMILX_MC_GetCurSpeed

예제

▣ 예제 1

1 회전에 필요한 펄스수가 3600 펄스일 때 거리의 단위를 각도(1°)로, 속도의 단

위를 rpm 으로 설정하는 예제입니다.

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

void main()
{
   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

   // Set 10 pulses for unit distance              //
   // 이 예제에서는 1 회전에 필요한 펄스 수를 3600 펄스로   //
   // 가정하고 단위 거리를 1 ° 로 설정한 것이다.          //
   COMILX_MC_SetUnitDistance(hDevice, 0, 10);
   // Set 3600/60(=60) PPS for unit speed          //
   // 이 예제에서는 1 회전에 필요한 펄스수를 3600 펄스로    //
   // 가정하고 단위 속도를 1rpm 으로 설정한 것이다.       //
   COMILX_MC_SetUnitSpeed(hDevice, 0, 3600./60);
   // Set trapezoidal speed mode //
   COMILX_MC_SetSpeedMode(hDevice, 0, 1);
   // Set speed as 100 rpm //
   COMILX_MC_SetSpeed(hDevice, 0, 0, 100);
   //가속도와 감속도를 각가 200rpm/s 로 설정한다. 이렇게 하면 작업속도가//
   //100rpm 이므로 가속 및 감속 시간은  각각 0.5 초  걸린다.          //
   COMILX_MC_SetAccel(hDevice, 0, 200, 200);
   // 모터를 720 ° 회전한다. 실제로는 720*10 펄스가 출력된다. //
   COMILX_MC_Move(hDevice, 0, 720);

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

▣ 예제 2

1cm 이동하는데 필요한 펄스수가 1000 펄스일 때 거리의 단위를 cm 로, 속도의 단위를 cm/sec 로 설정하는 예제입니다.

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

void main()
{
   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

   // Set 1000 pulses for unit distance             //
   // 이 예제에서는 1cm 이동에 필요한 펄스수를 1000 펄스로  //
   // 가정하고 단위 거리를 1cm 로 설정한 것이다.              //
   COMILX_MC_SetUnitDistance(hDevice, 0, 1000);
   // Set 1000 PPS for unit speed                       //
   // 이 예제에서는 1cm 이동에 필요한 펄스수를 1000 펄스로  //
   // 펄스로 가정하고 단위 속도를 1rpm 로 설정한 것이다.     //
   COMILX_MC_SetUnitSpeed(hDevice, 0, 1000);

   COMILX_MC_SetSpeed(hDevice, 0, 0, 50); // Set speed as 50
cm/sec/ //
   COMILX_MC_SetSpeedMode(hDevice, 0, 0); // Set constant speed
mode //

  COMILX_MC_Move(hDevice, 0, 10); // 50 cm/sec 의 속도로 10cm 이동 .
실제로는 10*1000=10000 펄스가 출력된다.//

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

▣ COMILX_MC_GetUnitSpeed

함수 원형

double COMILX_MC_GetUnitSpeed (HANDLE hDevice, int nChannel)

함수 설명

현재 설정된 논리적 단위 속도에 대한 실제 펄스 출력 속도(PPS)를 반환합니다. 여기서 논리적 단위 속도라 함은 속도 지정함수에서 사용하는 속도 또는 가속도에 대한 단위량을 의미합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재 설정된 단위 거리에 대한 펄스 수를 반환합니다.

▣ COMILX_MC_SetInOutRatio

함수 원형

void COMILX_MC_SetInOutRatio (HANDLE hDevice, int nChannel, double fRatio)

함수 설명

Feedback 펄스와 Command 펄스의 분해능 비율(Resolution ratio)을 설정합니다. 여기서 Feedback 펄스의 분해능이란 엔코더의 1 회전시에 발생되는 펄스 수를 의미합니다. 그리고 Command 펄스의 분해능이란 모터를 1 회전시키기 위해 필요한 Command 펄스수를 의미합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fRatio* : Feedback 펄스와 Command 펄스의 분해능 비를 지정합니다. 이 값은

fRatio = (Feedback 펄스 분해능)/(Command 펄스의 분해능)

참 고

□ In/Out Ratio 는 Actual(Feedback) position 또는 Actual speed 를 논리 단위로 읽을 때 적용됩니다. 논리적 단위 거리나 단위 속도는 Command 펄스기준으로 설정되므로 Command 펄스와 Feedback 펄스의 분해능이 서로 다르다면 Actual position 이나 Acutal speed 의 논리값 계산이 잘못되게 됩니다.

□ In/Out Ratio 는 **COMILX_MC_GetPosition_A**() 함수와 **COMILX_MC_GetActualSpeed**() 함수에만 영향을 미칩니다. In/Out Ratio 가 적용되는 로직은 이 두 함수의 설명을 참조하십시오.

5.4.2 Single Axis 모션 제어 함수

이 단원에서는 Single Axis 모션 제어에 관련된 함수들을 소개합니다. Single Axis 모션은 한 축만을 독립적으로 제어하는 작업을 의미합니다. Single Axis 모션은 먼저 속도설정 함수들을 이용하여 속도를 설정하고 이동 함수를 사용하여 이동 작업을 수행합니다. 그리고 필요에 따라 정지 함수를 사용하여 모션을 정지합니다. Single Axis Motion 에 관련된 함수들은 다음과 같습니다.

void COMILX_MC_SetSpeedMode(HANDLE hDevice, int nChannel, int nModeIndex) Motion 의 속도 모드를 설정합니다.
void COMILX_MC_SetSpeed (HANDLE hDevice, int nChannel, double fIniSpeed, double fWorkSpeed) Motion 의 속도를 설정합니다.
void COMILX_MC_SetAccel(HANDLE hDevice, int nChannel, double fAccel, double fDecel) Motion 의 가/감속도를 설정합니다.
void COMILX_MC_SetScurve (HANDLE hDevice, int nChannel, double fSVacc, double fSVdec) 속도모드를 S-curve 속도 패턴으로 설정한 경우에 S-curve Section 의 범위를 속도단위로 설정합니다.
void COMILX_MC_StartVMove (HANDLE hDevice, int nChannel, int nDirection) 작업속도까지 가속한 후에 작업속도를 유지하며 정지함수가 호출될 때까지 지정한 방향으로의 모션을 계속 수행합니다.
void COMILX_MC_StartMove (HANDLE hDevice, int nChannel, double fDistance) 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 함수는 모션을 시작시킨 후 바로 Return 합니다.
void COMILX_MC_Move (HANDLE hDevice, int nChannel, double fDistance) 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 함수는 모션이 완료될때까지 Return 되지 않습니다.
void COMILX_MC_StartMoveTo (HANDLE hDevice, int nChannel, double fPosition) 지정한 절대좌표로의 이동을 수행합니다. 이 함수는 모션(Motion)을 시작 시킨 후에 바로 Return 합니다.
void COMILX_MC_MoveTo (HANDLE hDevice, int nChannel, double fPosition) 지정한 절대좌표로의 이동을 수행합니다. 이 함수는 모션이 완료될때까지 Return 되지 않습니다.
void COMILX_MC_Stop (HANDLE hDevice, int nChannel) 지정한 축에 대한 모션을 감속 후 정지합니다.
void COMILX_MC_EmgStop (HANDLE hDevice, int nChannel)

지정한 축에 대한 모션을 감속없이 즉시 정지합니다.
BOOL COMILX_MC_Done (HANDLE hDevice, int nChannel) 하나의 축에 대하여 모션이 완료됐는지를 체크합니다.

▣ COMILX_MC_SetSpeedMode

함수 원형

void COMILX_MC_SetSpeedMode(HANDLE hDevice, int nChannel, int nModeIndex)

함수 설명

Motion 의 속도 모드를 설정합니다. 단, 이 함수는 Motion 에 바로 영향을 주는 것이 아니고 Move, MoveTo 등의 이송 함수가 수행될 때 설정된 내용이 적용됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nModeIndex* : 속도 모드를 지정합니다. 속도 모드는 다음과 같이 3 가지로 설정할 수 있습니다.

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

관련 함수

COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve

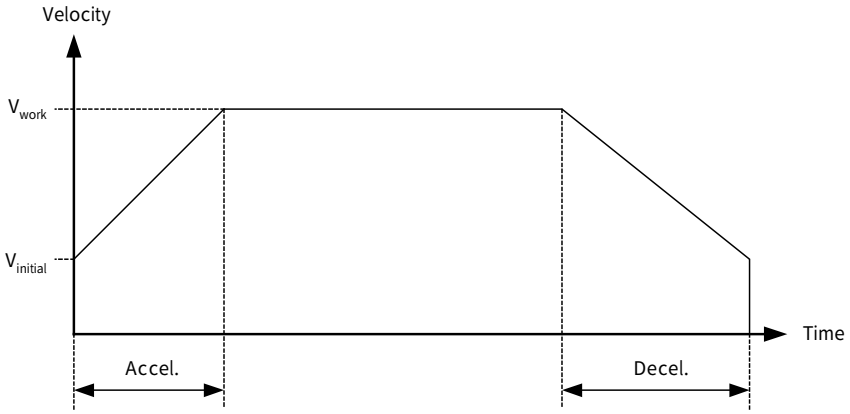
참 고

□ Constant speed mode

Constant speed mode 에서는 Motion 을 수행할 때 가속/감속을 적용하지 않고 일정속도로 Motion 을 수행합니다. 여기서 적용되는 일정 속도는 COMILX_MC_SetSpeed 함수에서 주어지는 fEndSpeed 에서 주어진 값이 적용됩니다.

□ Trapezoidal speed mode

Trapezoidal speed mode 에서는 Motion 을 수행하는데 있어서 속도의 패턴을 [그림 5-4]와 같이 Linear acceleration -> Working speed(constant) -> Linear deceleration 의 형태로 운용하는 모드입니다.



[그림 5-4] Trapezoidal speed pattern

여기서 Acceleration time 은

$T_{acc} = (V_{work} - V_{initial})/a$

Where,

T_{acc} : Acceleration time

$V_{initial}$: Initial speed

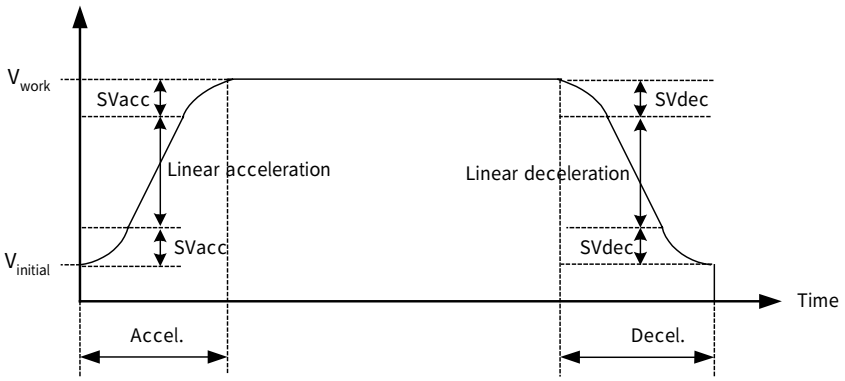
V_{work} : Working speed

a : Acceleration setting value

과 같으며 Deceleration time 또한 위와 같은 계산식이 적용됩니다.

□ S-curve speed mode

S-curve speed mode 에서는 Motion 을 수행할 때 S 자형 형태로 가속과 감속을 수행합니다. S-curve speed mode 에서 가(감)속 구간은 [그림 5-5]과 같이 S-curve section 과 Linear acceleration section 으로 구성됩니다.



[그림 5-5] S-curve speed pattern

※ **S-curve section** : S-curve 형식의 가/감속이 이루어지는 구간. 이 구간은 COMILX_MC_SetSCurve 함수의 fSVacc 와 fSVdec 파라미터에 의해 설정됩니다. fSVacc 값이 0 이거나 속도 범위(Working speed - Initial speed)의 50%로 설정되면 가속구간은 Linear acceleration section 이 없이 모두 S-curve section 으로 구성됩니다. 마찬가지로, fSVdec 값이 0 이거나 속도 범위(Working speed - Initial speed)의 50%로 설정되면 감속구간은 Linear deceleration section 이 없이 모두 S-curve section 으로 구성됩니다.

※ S-curve speed mode 에서 COMILX_MC_SetAccel 함수를 통하여 설정한 가(감)속 값은 S-curve section 을 포함한 전체 가(감)속 시간을 결정하는 파라미터로 사용되며 실제 가(감)속도 또는 Jerk 는 자동으로 계산됩니다. 전체 가속 시간 Tacc 는

Tacc = (Vwork - Vinitial)/a

여기서,

Tacc : Acceleration time

Vinitial : Initial speed

Vwork : Working speed

a : Acceleration setting value

과 같으며 Deceleration time 또한 위와 같은 계산식이 적용됩니다.

예 제

▣ 예제 1

다음의 예제는 Trapezoidal 속도 모드를 설정하는 예제입니다.

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

#define X_AXIS    0 // X 축 채널번호
void main()
{
   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

   // Set trapezoidal speed mode //
   COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
   // Set speed as 1000 //
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 1000);
   //가속도와 감속도를 각가 2000 으로 설정한다. 이렇게 하면 작업속도가  //
   //1000 이므로 가속 및 감속 시간은  각각 0.5 초  걸린다.        //
   COMILX_MC_SetAccel(hDevice, X_AXIS, 2000, 2000);
   // 현재의 위치로부터 5000 만큼 이동 //
   COMILX_MC_Move(hDevice, X_AXIS, 5000);

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

▣ 예제 2

다음의 예제는 S-Curve 속도 모드를 설정하는 예제입니다.

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

#define X_AXIS    0 // X 축 채널번호
void main()
{
   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

    // Set S-Curve speed mode //
    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 2);
    // Set speed as 1000 //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 1000);
    //가속도와 감속도를 각가 2000으로 설정한다. 이렇게 하면 작업속도가  //
    //1000 이므로 가속 및 감속 시간은  각각 0.5초  걸린다.            //
    COMILX_MC_SetAccel(hDevice, X_AXIS, 2000, 2000);
    // Set S-Curve section range : 본 예제는 Linear section이    //
    // 없는 완// 전한 S-curve 가/감속 모드가 되도록 SVacc, SVdec 값을 //
    // 모두 0으로 설정함                                          //
    COMILX_MC_SetScurve(hDevice, X_AXIS, 0, 0);
    // 현재의 위치로부터 5000만큼 이동 //
    COMILX_MC_Move(hDevice, X_AXIS, 5000);

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

▣ COMILX_MC_SetSpeed

함수 원형

void COMILX_MC_SetSpeed (HANDLE hDevice, int nChannel, double fIniSpeed, double fWorkSpeed)

함수 설명

Motion 의 속도를 설정합니다. 단, 이 함수는 Motion 에 바로 영향을 주는 것이 아니고 Move, MoveTo 등의 이송 함수가 수행될 때 설정된 내용이 적용됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fIniSpeed* : 초기 속도를 설정합니다. 단, Constant 속도 모드에서는 이 값이 무시됩니다.

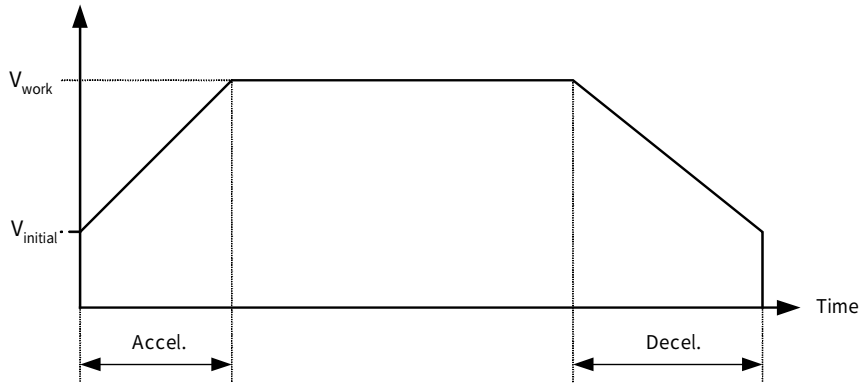
▶ *fWorkSpeed* : 작업 속도를 설정합니다.

참 고

□ 속도의 단위는 COMILX_MC_SetUnitSpeed 함수에 의하여 결정되며 기본적으로는 Pulses/sec 입니다.

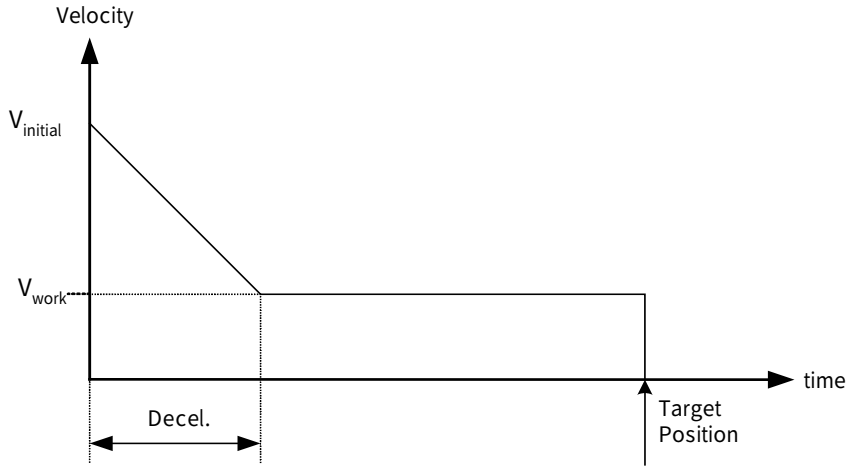
□ 초기속도(fIniSpeed)가 작업속도(fWorkSpeed)가 설정 가능한 속도 범위보다 작거나 크면 자동으로 속도 범위의 최소값 또는 최대값으로 설정됩니다. 속도 범위는 COMILX_MC_SetSpeedRange 함수에 의해 결정됩니다.

□ Trapezodial 또는 S-curve speed mode 에서 In-Position 명령을 수행할 때 작업속도(fWorkSpeed)가 초기속도(fIniSpeed)보다 크면 [그림 5-6]과 같이 초기속도⇒가속⇒작업속도⇒감속⇒정지의 동작을 수행합니다.



[그림 5-6] 작업속도가 초기속도보다 크게 설정된 경우의 속도 구성

□ Trapezodial 또는 S-curve speed mode 에서 In-Position 명령을 수행할 때 작업속도(fWorkSpeed)가 초기속도(flniSpeed)보다 작으면 [그림 5-7]과 같이 초기속도로부터 출발하여 작업속도까지 감속 후에 작업속도를 유지하고 목표 위치까지 이동한 후에는 감속 없이 바로 정지하게 됩니다.



[그림 5-7] 작업속도가 초기속도보다 작게 설정된 경우의 속도 구성

예 제

▣ 예제 1

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

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

#define X_AXIS    0
void main()
{
   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_SetSpeedMode(hDevice, X_AXIS, 0);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 1000);
   COMILX_MC_StartVMove(hDevice, X_AXIS, 1);
   while(!kbhit())
       ;
   COMILX_MC_EmgStop(hDevice, X_AXIS);

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

▣ 예제 2

1 회전에 필요한 펄스수가 3600 펄스일 때 거리의 단위를 각도(1°)로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

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

void main()
{
   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

   // Set 10 pulses for unit distance              //
   // 이 예제에서는 1 회전에 필요한 펄스 수를 3600 펄스로   //
   // 가정하고 단위 거리를 1° 로 설정한 것이다.           //
   COMILX_MC_SetUnitDistance(hDevice, 0, 10);
   // Set 3600/60(=60) PPS for unit speed          //
   // 이 예제에서는 1 회전에 필요한 펄스수를 3600 펄스로    //
```

```
    // 가정하고 단위 속도를 1rpm 으로 설정한 것이다.       //
    COMILX_MC_SetUnitSpeed(hDevice, 0, 3600./60);
    // Set trapezoidal speed mode //
    COMILX_MC_SetSpeedMode(hDevice, 0, 1);
    // Set speed as 100 rpm //
    COMILX_MC_SetSpeed(hDevice, 0, 0, 100);
    //가속도와 감속도를 각가 200rpm/s 로 설정한다. 이렇게 하면 작업속도가//
    //100rpm 이므로 가속 및 감속 시간은  각각 0.5 초  걸린다.          //
    COMILX_MC_SetAccel(hDevice, 0, 200, 200);
    // 모터를 720 ° 회전한다. 실제로는 720*10 펄스가 출력된다. //
    COMILX_MC_Move(hDevice, 0, 720);

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

▣ COMILX_MC_SetAccel

함수 원형

void COMILX_MC_SetAccel(HANDLE hDevice, int nChannel, double fAccel, double fDecel)

함수 설명

Motion 의 가/감속도를 설정합니다. 단, 이 함수는 Motion 에 바로 영향을 주는 것이 아니고 Move, MoveTo 등의 이송 함수가 수행될 때 설정된 내용이 적용됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fAccel* : 가속도를 설정합니다. 가속도의 단위는 COMILX_MC_SetUnitSpeed 함수에 의하여 결정되며 기본적으로는 1 PPS/SEC 입니다. 이 값이 0 이면 가속은 생략됩니다.

▶ *fDecel* : 감속도를 설정합니다. 감속도의 단위는 COMILX_MC_SetUnitSpeed 함수에 의하여 결정되며 기본적으로는 1 PPS/SEC 입니다. 이 값이 0 이면 감속은 생략됩니다.

참 고

□ 가/감속은 COMILX_MC_SetSpeedMode 함수에서 속도 패턴(Speed Pattern)을 Trapezoidal 또는 S-Curve 로 지정한 경우에 적용됩니다.

□ Trapezoidal 속도 패턴에서는 가/감속 구간의 가/감속도와 일치하게 됩니다. 그러나 S-curve 속도 패턴에서는 이 함수에서 지정한 가/감속 값은 전체 가/감속 구간(S-curve section 포함)의 시간을 결정하는 파라미터로 사용되며 가속도 값은 S-curve 의 range 에 따라 자동으로 결정됩니다. Trapezoidal 속도 패턴에서는 가/감속 구간의 가/감속도와 일치하게 됩니다. 그러나 S-curve 속도 패턴에서는 이 함수에서 지정한 가/감속 값은 전체 가/감속 구간(S-curve section 포함)의 시간을 결정하는 파라미터로 사용되며 가속도 값은 S-curve 의 range 에 따라 자동으로 결정됩니다.

전체 가속 시간 Tacc 는
  Tacc = (Vwork - Vinitial)/a
여기서,
  Tacc : Acceleration time
  Vinitial : Initial speed
  Vwork : Working speed
  a : Acceleration setting value

와 같으며 Deceleration time 또한 위와 같은 계산식이 적용됩니다.

예 제

▣ 예제 1

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

#define X_AXIS    0
void main()
{
   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_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 1000);
   COMILX_MC_SetAccel(hDevice, X_AXIS, 2000, 2000);
   // V=1000, Acc=2000, Dec=2000 의 속도 패턴으로 //
   // 4000 만큼 이동                              //
   COMILX_MC_Move(hDevice, X_AXIS, 4000);

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

▣ 예제 2

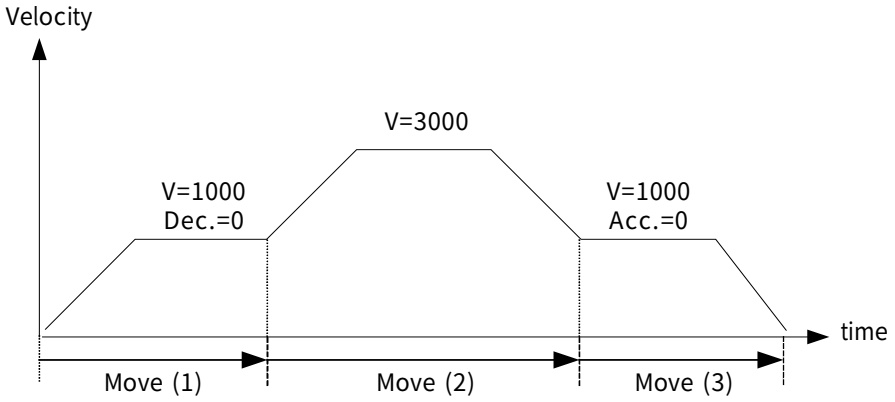
1 회전에 필요한 펄스수가 3600 펄스일 때 거리의 단위를 각도(1°)로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

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

void main()
{
   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
```

```
    // Set 10 pulses for unit distance              //
    // 이 예제에서는 1 회전에 필요한 펄스 수를 3600 펄스로   //
    // 가정하고 단위 거리를 1 ° 로 설정한 것이다.           //
    COMILX_MC_SetUnitDistance(hDevice, 0, 10);
    // Set 3600/60(=60) PPS for unit speed          //
    // 이 예제에서는 1 회전에 필요한 펄스수를 3600 펄스로    //
    // 가정하고 단위 속도를 1rpm 으로 설정한 것이다.        //
    COMILX_MC_SetUnitSpeed(hDevice, 0, 3600./60);
    // Set trapezoidal speed mode //
    COMILX_MC_SetSpeedMode(hDevice, 0, 1);
    // Set speed as 100 rpm //
    COMILX_MC_SetSpeed(hDevice, 0, 0, 100);
    //가속도와 감속도를 각가 200rpm/s 로 설정한다. 이렇게 하면 작업속도가//
    //100rpm 이므로 가속 및 감속 시간은  각각 0.5 초  걸린다.          //
    COMILX_MC_SetAccel(hDevice, 0, 200, 200);
    // 모터를 720 ° 회전한다. 실제로는 720*10 펄스가 출력된다. //
    COMILX_MC_Move(hDevice, 0, 720);

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

▣ 예제 3

아래 그림과 같이 속도의 연속성을 가지는 3 단계의 In-Position 작업을 리스트 모션(Listed Motion) 기능을 이용하여 구현하는 예입니다. 리스트 모션은 하나의 작업과 다음 작업간의 지연시간을 없애고 연속적으로 수행될 수 있도록 일괄처리하는 기능입니다. 이 때 초기 속도, 작업 속도, 가속도, 감속도를 적절히 설정하면 여러 단계의 In-Position 작업을 속도의 연속성을 가지고 연속적으로 수행할 수 있습니다.



작 업	초기속도	작업속도	Acceleration	Deceleration
Move (1)	0	1000	2000	0
Move (2)	1000	3000	2000	2000
Move (3)	0	1000	0	2000

[그림 5-8] 속도의 연속성을 가지는 연속적인 In-Position 모션

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

void main()
{
   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_BeginList(hDevice); // 모션 리스트 등록 시작 //
   // Set Trapezoidal Speed Mode //
   COMILX_MC_SetSpeedMode(hDevice, 0, 1);
   // Move (1) //
   COMILX_MC_SetSpeed(hDevice, 0, 0, 1000);
   COMILX_MC_SetAccel(hDevice, 0, 2000, 0);
   COMILX_MC_MoveTo(hDevice, 0, 3000);
   // Move (2) //
   COMILX_MC_SetSpeed(hDevice, 0, 1000, 3000);
   COMILX_MC_SetAccel(hDevice, 0, 2000, 2000);
   COMILX_MC_MoveTo(hDevice, 0, 8000);
   // Move (3) //
   COMILX_MC_SetSpeed(hDevice, 0, 0, 1000);
   COMILX_MC_SetAccel(hDevice, 0, 0, 2000);
   COMILX_MC_MoveTo(hDevice, 0, 3500);
   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))
       ;

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

▣ COMILX_MC_SetScurve

함수 원형

void COMILX_MC_SetScurve (HANDLE hDevice, int nChannel, double fSVacc, double fSVdec)

함수 설명

속도모드를 S-curve 속도 패턴으로 설정한 경우에 S-curve Section 의 범위를 속도 단위로 설정합니다. 단, 이 함수는 Motion 에 바로 영향을 주는 것이 아니고 Move, MoveTo 등의 이송 함수가 수행될 때 설정된 내용이 적용됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

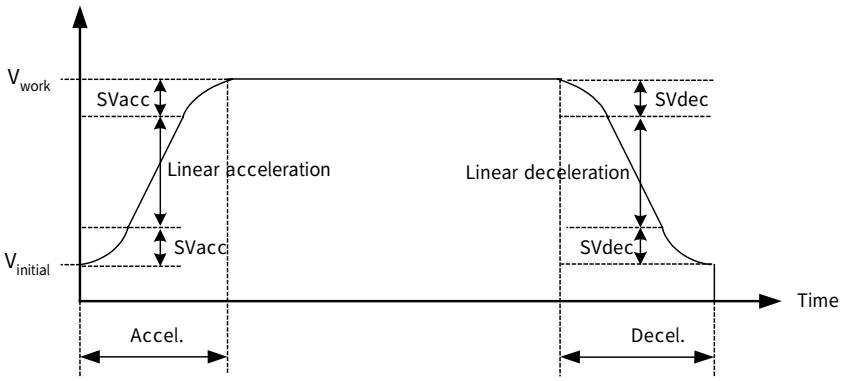
▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fSVacc* : 가속구간의 S-curve Section 의 범위를 속도단위로 지정합니다.

▶ *fSVdec* : 감속구간의 S-curve Section 의 범위를 속도단위로 지정합니다.

참 고

S-curve speed mode 에서는 Motion 을 수행할 때 S 자형 형태로 가속과 감속을 수행합니다. S-curve speed mode 에서 가(감)속 구간은 [그림 5-9]과 같이 S-curve section 과 Linear acceleration section 으로 구성됩니다.



[그림 5-9] S-curve speed pattern

※ **S-curve section** : S-curve 형식의 가/감속이 이루어지는 구간. 이 구간은

COMILX_MC_SetSCurve 함수의 fSVacc 와 fSVdec 파라미터에 의해 설정됩니다. fSVacc 값이 0 이거나 속도 범위(Working speed - Initial speed)의 50%로 설정되면 가속구간은 Linear acceleration section 이 없이 모두 S-curve section 으로 구성됩니다. fSVdec 값이 0 이거나 속도 범위(Working speed - Initial speed)의 50%로 설정되면 감속구간은 Linear deceleration section 이 없이 모두 S-curve section 으로 구성됩니다.

※ S-curve speed mode 에서 COMILX_MC_SetAccel 함수를 통하여 설정한 가(감)속 값은 S-curve section 을 포함한 전체 가(감)속 시간을 결정하는 파라미터로 사용되며 실제 가(감)속도 또는 Jerk 는 자동으로 계산됩니다. 전체 가속 시간 Tacc 는

Tacc = (Vwork - Vinitial)/a

여기서,

Tacc : Acceleration time

Vinitial : Initial speed

Vwork : Working speed

a : Acceleration setting value

과 같으며 Deceleration time 또한 위와 같은 계산식이 적용됩니다.

예 제

▣ 예제 1

본 예제는 다음과 같은 속도 조건을 가지고 S-Curve 가/감속 모드로 In-Position 작업을 수행하는 예입니다.

```
Vinitial=0
Vwork=10000
Acc Time=0.5 초 => Acc = 10000/0.5 = 20000
Dec Time=0.5 초 => Dec = 10000/0.5 = 20000
SVacc=0 => No linear section in acceleration
SVdec=0 => No linear section in deceleration
```

이 예제는 SVacc, SVdec 값을 모두 0 으로 하므로써 가/감속시에 완전한 S-Curve 를 그리는 가/감속 모드를 수행합니다.

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

#define X_AXIS    0
```

```
void main()
{
   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

   // 속도 모드를 S-curve 모드로 설정 //
   COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 2);

   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
   COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);
   COMILX_MC_SetScurve(hDevice, X_AXIS, 0, 0);

   COMILX_MC_Move(hDevice, X_AXIS, 50000);

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

▣ 예제 2

본 예제는 다음과 같은 속도 조건을 가지고 S-Curve 가/감속 모드로 In-Position 작업을 수행하는 예입니다.

```
Vinitial=0
Vwork=10000
Acc Time=0.5 초 => Acc = 10000/0.5 = 20000
Dec Time=0.5 초 => Dec = 10000/0.5 = 20000
SVacc=2000
SVdec=2000
```

이 예제는 SVacc, SVdec 값을 각각 2000 으로 설정하므로써 0~2000 과 8000~10000 의 속도 구간은 S-curve 가/감속을, 2000 ~ 8000 의 속도 구간은 Linear 가/감속을 적용하도록 하는 예제입니다.

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

#define X_AXIS    0
void main()
{
   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

    // 속도 모드를 S-curve 모드로 설정 //
    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 2);

    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);
    COMILX_MC_SetScurve(hDevice, X_AXIS, 2000, 2000);

    COMILX_MC_Move(hDevice, X_AXIS, 50000);

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

▣ COMILX_MC_StartVMove

함수 원형

void COMILX_MC_StartVMove (HANDLE hDevice, int nChannel, int nDirection)

함수 설명

작업속도까지 가속한 후에 작업속도를 유지하며 정지함수가 호출될 때까지 지정한 방향으로의 모션을 계속 수행합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nDirection* : 모션의 방향을 설정합니다.

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

참 고

□ 속도 모드를 Constant Speed Mode 로 지정한 경우에는 가속 구간이 없이 작업속도로 모션을 시작합니다.

□ Velocity Move 를 정지할 때 COMILX_MC_Stop 또는 COMILX_MC_EmgStop 을 사용합니다.

예 제

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

#define X_AXIS    0
void main()
{
   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_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);

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

▣ COMILX_MC_StartMove

함수 원형

void COMILX_MC_StartMove (HANDLE hDevice, int nChannel, double fDistance)

함수 설명

하나의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 함수는 모션(Motion)을 시작 시킨 후에 바로 Return 합니다. 속도 패턴은 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fDistance* : 이동할 거리를 지정합니다. 이 값은 현재의 위치에 대한 상대 좌표이며 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, fDistance 값 1 은 1Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_Move 함수가 모션이 완료될 때까지 Return 되지 않는데 반하여, 이 함수는 지정한 모션을 시작시킨 후에 바로 Return 하게 됩니다.

□ COMILX_MC_StartMoveTo 함수가 절대좌표로의 이동을 수행하는데 반하여, 이 함수는 현재 위치에서 상대적인 거리를 파라미터로하여 이동을 수행합니다.

예 제

▣ 예제 1

본 예제는 **Trapezoidal** 속도모드를 적용하여 **(+)**방향으로 **30000** 이동한 후 다시 **(-)**방향으로 **30000** 이동하는 예제입니다. 본 예제에서는 거리나 속도에 대한 논리 단위를 특별히 지정하지 않고 거리와 속도의 단위를 기본단위인 **Pulses** 와 **PPS** 단위로 사용합니다.

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

#define X_AXIS    0
void main()
{
```

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

    COMILX_MC_StartMove(hDevice, X_AXIS, 30000);
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;
    COMILX_MC_StartMove(hDevice, X_AXIS, -30000);
    // 모션이 완료될때까지 기다린다 //
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;

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

▣ 예제 2

본 예제는 COMILX_MC_SetUnitDistance()함수와 COMILX_MC_SetUnitSpeed() 함수를 사용하여 논리거리(Logic Distance)와 논리속도(Logic Speed)를 정의하여 Move 작업을 수행하는 예입니다. 1mm 이동하는데 필요한 펄스수가 100 펄스이고 1 회전에 필요한 펄스수가 10000 펄스일때 거리의 단위를 mm 로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"
#define X_AXIS    0
void main()
{
    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

    // Set 100 pulses for unit distance              //
    // 이 예제에서는 1mm 이동에 필요한 펄스수를 100 펄스로  //
    // 가정하고 단위 거리를 1mm 로 설정한 것이다.            //
    COMILX_MC_SetUnitDistance(hDevice, X_AXIS, 100);
    // Set 10000/60(=166.7) PPS for unit speed //
    // 이 예제에서는 1 회전에 필요한 펄스수를 10000        //
```

```
    // 펄스로 가정하고 단위 속도를 1rpm 로 설정한 것이다.  //
    COMILX_MC_SetUnitSpeed(hDevice, X_AXIS, 10000./60);

    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
    // Set speed as 60 rpm //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 60);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 60, 60);
    // 60 rpm 의 속도로 100mm 이동 //
    COMILX_MC_StartMove(hDevice, X_AXIS, 100);
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;

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

▣ COMILX_MC_Move

함수 원형

void COMILX_MC_Move (HANDLE hDevice, int nChannel, double fDistance)

함수 설명

하나의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 때의 속도 패턴은 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다. 이 함수는 지정한 위치로의 이동이 완료되기 전까지 Return 되지 않습니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fDistance* : 이동할 거리를 지정합니다. 이 값은 현재의 위치에 대한 상대 좌표이며 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, fDistance 값 1 은 1Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_StartMove 함수가 모션이 완료되는 것을 기다리지 않고 바로 Return 하는데 반하여, 이 함수는 지정한 상대좌표로의 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_MoveTo 함수가 절대좌표로의 이동을 수행하는데 반하여, 이 함수는 현재 위치에서 상대적인 거리를 파라미터로하여 이동을 수행합니다.

예 제

▣ 예제 1

본 예제는 **Trapezoidal** 속도모드를 적용하여 **(+)**방향으로 **30000** 이동한 후 다시 **(-)**방향으로 **30000** 이동하는 예제입니다. 본 예제에서는 거리나 속도에 대한 논리 단위를 특별히 지정하지 않고 거리와 속도의 단위를 기본단위인 **Pulses** 와 **PPS** 단위로 사용합니다.

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

```
#include "comidaslx.h"

#define X_AXIS    0
void main()
{
   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_SetBlockingMode (hDevice, FALSE);

   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_Move(hDevice, X_AXIS, 30000);
   COMILX_MC_Move(hDevice, X_AXIS, -30000);

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

▣ 예제 2

본 예제는 COMILX_MC_SetUnitDistance() 함수와 COMILX_MC_SetUnitSpeed() 함수를 사용하여 논리거리(Logic Distance)와 논리속도(Logic Speed)를 정의하여 Move 작업을 수행하는 예입니다. 1mm 이동하는데 필요한 펄스수가 100 펄스이고 1 회전에 필요한 펄스수가 10000 펄스일때 거리의 단위를 mm 로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"
#define X_AXIS    0
void main()
{
   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

   // Set 100 pulses for unit distance             //
   // 이 예제에서는 1mm 이동에 필요한 펄스수를 100 펄스로  //
   // 가정하고 단위 거리를 1mm 로 설정한 것이다.          //
   COMILX_MC_SetUnitDistance(hDevice, X_AXIS, 100);
   // Set 10000/60(=166.7) PPS for unit speed      //
```

```
    // 이 예제에서는 1 회전에 필요한 펄스수를 10000        //
    // 펄스로 가정하고 단위 속도를 1rpm 로 설정한 것이다.   //
    COMILX_MC_SetUnitSpeed(hDevice, X_AXIS, 10000./60);

    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
    // Set speed as 60 rpm //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 60);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 60, 60);
    // 60 rpm 의 속도로 100mm 이동 //
    COMILX_MC_Move(hDevice, X_AXIS, 100);

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

▣ COMILX_MC_StartMoveTo

함수 원형

void COMILX_MC_StartMoveTo (HANDLE hDevice, int nChannel, double fPosition)

함수 설명

하나의 축에 대하여 지정한 절대좌표로의 이동을 수행합니다. 이 함수는 모션(Motion)을 시작 시킨 후에 바로 Return 합니다. 속도 패턴은 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fPosition* : 이동할 절대 좌표 값을 지정합니다. 좌표의 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다.

참 고

☐ COMILX_MC_MoveTo 함수가 모션이 완료될 때까지 Return 되지 않는데 반하여, 이 함수는 지정한 모션을 시작시킨 후에 바로 Return 하게 됩니다.

☐ COMILX_MC_StartMove 함수가 현재위치에 대한 상대좌표로의 이동을 수행하는데 반하여, 이 함수는 절대좌표로의 이동을 수행합니다.

예 제

▣ 예제 1

본 예제는 현재 좌표를 0 이라 가정하고 Trapezoidal 속도모드를 적용하여 2 회의 MoveTo 작업을 수행하므로써 절대좌표 70000 으로 이동하는 예입니다.

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

#define X_AXIS    0
void main()
{
   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
    // Command Position의 현재 좌표를 0으로 초기화한다. //
    COMILX_MC_SetPosition_C (hDevice, X_AXIS, 0);

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

    // 좌표 0에서 30000으로 이동 //
    COMILX_MC_StartMoveTo(hDevice, X_AXIS, 30000);
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;
    // 좌표 30000에서 70000으로 이동 //
    COMILX_MC_StartMoveTo(hDevice, X_AXIS, 70000);
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;

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

▣ 예제 2

본 1210 예제는 COMILX_MC_SetUnitDistance() 함수와 COMILX_MC_SetUnitSpeed() 함수를 사용하여 논리거리(Logic Distance)와 논리속도(Logic Speed)를 정의하여 Move 작업을 수행하는 예입니다. 1mm 이동하는데 필요한 펄스수가 100 펄스이고 1 회전에 필요한 펄스수가 10000 펄스일때 거리의 단위를 mm 로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"
#define X_AXIS    0
void main()
{
    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

    // Set 100 pulses for unit distance             //
    // 이 예제에서는 1mm 이동에 필요한 펄스수를 100 펄스로  //
    // 가정하고 단위 거리를 1mm 로 설정한 것이다.          //
    COMILX_MC_SetUnitDistance(hDevice, X_AXIS, 100);
    // Set 10000/60(=166.7) PPS for unit speed      //
    // 이 예제에서는 1 회전에 필요한 펄스수를 10000        //
```

```
    // 펄스로 가정하고 단위 속도를 1rpm 로 설정한 것이다.   //
    COMILX_MC_SetUnitSpeed(hDevice, X_AXIS, 10000./60);

    // Command Position 의 현재 좌표를 0 으로 초기화한다. //
    COMILX_MC_SetPosition_C (hDevice, X_AXIS, 0);

    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
    // Set speed as 60 rpm //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 60);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 60, 60);
    // 60 rpm 의 속도로 좌표 100(mm)으로 이동 //
    COMILX_MC_StartMoveTo(hDevice, X_AXIS, 100);
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;

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

▣ COMILX_MC_MoveTo

함수 원형

void COMILX_MC_MoveTo (HANDLE hDevice, int nChannel, double fPosition)

함수 설명

하나의 축에 대하여 지정한 절대좌표로의 이동을 수행합니다. 속도 패턴은 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된대로 이루어집니다. 이 함수는 지정한 절대좌표로의 이동이 완료되기 전까지 Return 되지 않습니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fPosition* : 이동할 절대 좌표 값을 지정합니다. 좌표의 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다.

참 고

□ COMILX_MC_StartMoveTo 함수가 모션이 완료되는 것을 기다리지 않고 바로 Return 하는데 반하여, 이 함수는 지정한 절대좌표로의 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_MoveTo 함수가 절대좌표로의 이동을 수행하는데 반하여, 이 함수는 현재 위치에서 상대적인 거리를 파라미터로하여 이동을 수행합니다.

예 제

▣ 예제 1

본 예제는 현재 좌표를 0 이라 가정하고 Trapezoidal 속도모드를 적용하여 2 회의 MoveTo 작업을 수행하므로써 절대좌표 70000 으로 이동하는 예입니다.

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

#define X_AXIS    0
void main()
{
```

```
    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
    // Command Position의 현재 좌표를 0으로 초기화한다. //
    COMILX_MC_SetPosition_C (hDevice, X_AXIS, 0);

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

    // 좌표 0에서 30000으로 이동 //
    COMILX_MC_MoveTo(hDevice, X_AXIS, 30000);
    // 좌표 30000에서 70000으로 이동 //
    COMILX_MC_MoveTo(hDevice, X_AXIS, 70000);

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

▣ 예제 2

본 예제는 COMILX_MC_SetUnitDistance() 함수와 COMILX_MC_SetUnitSpeed() 함수를 사용하여 논리거리(Logic Distance)와 논리속도(Logic Speed)를 정의하여 Move 작업을 수행하는 예입니다. 1mm 이동하는데 필요한 펄스수가 100 펄스이고 1 회전에 필요한 펄스수가 10000 펄스일때 거리의 단위를 mm 로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"
#define X_AXIS    0
void main()
{
    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

    // Set 100 pulses for unit distance             //
    // 이 예제에서는 1mm 이동에 필요한 펄스수를 100 펄스로  //
    // 가정하고 단위 거리를 1mm 로 설정한 것이다.          //
    COMILX_MC_SetUnitDistance(hDevice, X_AXIS, 100);
    // Set 10000/60(=166.7) PPS for unit speed      //
```

```
    // 이 예제에서는 1 회전에 필요한 펄스수를 10000        //
    // 펄스로 가정하고 단위 속도를 1rpm 로 설정한 것이다.   //
    COMILX_MC_SetUnitSpeed(hDevice, X_AXIS, 10000./60);

    // Command Position 의 현재 좌표를 0 으로 초기화한다. //
    COMILX_MC_SetPosition_C (hDevice, X_AXIS, 0);

    COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
    // Set speed as 60 rpm //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 60);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 60, 60);
    // 60 rpm 의 속도로 좌표 100(mm)으로 이동 //
    COMILX_MC_MoveTo(hDevice, X_AXIS, 100);

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

▣ COMILX_MC_Stop

함수 원형

```
void COMILX_MC_Stop (HANDLE hDevice, int nChannel)
```

함수 설명

지정한 축에 대한 모션을 감속 후 정지합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

예 제

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

#define X_AXIS    0
void main()
{
   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_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);

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

▣ COMILX_MC_EmgStop

함수 원형

void COMILX_MC_EmgStop (HANDLE hDevice, int nChannel)

함수 설명

지정한 축에 대한 모션을 감속없이 즉시 정지합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

예 제

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

#define X_AXIS    0
void main()
{
   int bActive;
   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_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);
   BActive = TRUE;
   // Stop 명령(키보드)이 있을 때 까지 Velocity Move 지속 //
   while(!kbhit()){
       // DI CH0 가 ON 되면 즉시 정지 //
       if(COMILX_DI_GetOne(hDevice, 0)){
           COMILX_MC_EmgStop(hDevice, X_AXIS);
           bActive = FALSE;
       }
   }

   // 감속없이 즉시 정지 //
   if(bActive) COMILX_MC_EmgStop(hDevice, X_AXIS);
```

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

▣ COMILX_MC_Done

함수 원형

BOOL COMILX_MC_Done (HANDLE hDevice, int nChannel)

함수 설명

하나의 축에 대하여 모션이 완료됐는지를 체크합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

지정한 축의 모션이 완료됐는지를 알려줍니다.

Value	Meaning
0	모션이 완료되지 않았음
1	모션이 완료됨

참 고

COMILX_MC_SetMioCfgINP() 함수를 통하여 INP 기능이 Enable 되면 command 펄스 출력이 완료되도 INP 입력이 ON 이 되지 않으면 모션이 완료되지 않은 것으로 간주됩니다.

예 제

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

#define X_AXIS    0
void main()
{
   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_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
```

```
    COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);

    COMILX_MC_StartMove(hDevice, X_AXIS, 30000);
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;
    COMILX_MC_StartMove(hDevice, X_AXIS, 40000);
    // 모션이 완료될때까지 기다린다 //
    while(!COMILX_MC_Done (hDevice, X_AXIS))
        ;

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

5.4.3 Multi-Axis 동시제어 함수

이 단원에서는 Multi-Axis 동시 제어에 관련된 함수들을 소개합니다. Multi-Axis 동시 제어는 여러 개의 축을 완전한 동기를 맞추어 동시에 제어하는 기능을 말합니다. 만일 속도 패턴을 동일하게 설정하였다면 여러 개의 제어 대상 축이 시작 및 종료 시점은 물론이고 가속/감속 구간까지 완전히 동기를 맞추어 제어될 수 있습니다.

Multi-Axis 동시제어 기능은 Velocity Move 와 In-position Move 모두에 적용 가능합니다. 이와 관련된 함수들은 다음과 같습니다.

함수 / 설명
void COMILX_MC_StartVMoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], int nDirList[]) 여러 개의 축에 대하여 Velocity Move 작업을 동시에 시작합니다.
void COMILX_MC_StartMoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fDistList[]) 여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 동시에 시작합니다.
void COMILX_MC_MoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fDistList[]) 여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 수행합니다.
void COMILX_MC_StartMoveToAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fPosList[]) 여러 개의 축에 대하여 지정한 절대좌표로의 이동을 동시에 시작합니다.
void COMILX_MC_MoveToAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fPosList[]) 여러 개의 축에 대하여 지정한 절대좌표로의 이동을 수행합니다.
void COMILX_MC_StopAll (HANDLE hDevice, int nNumAxis, int nAxisList[]) 여러 개의 축에 대한 모션을 동시에 감속 후 정지합니다.
void COMILX_MC_EmgStopAll (HANDLE hDevice, int nNumAxis, int nAxisList[]) 여러 개의 축에 대한 모션을 동시에 감속없이 즉시 정지합니다.
BOOL COMILX_MC_AllDone (HANDLE hDevice, int nNumAxis, int nAxisList[]) 여러 개의 축에 대하여 지정한 모든 축의 모션이 완료됐는지를 체크합니다.

▣ COMILX_MC_StartVMoveAll

함수 원형

void COMILX_MC_StartVMoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], int nDirList[])

함수 설명

여러 개의 축에 대하여 Velocity Move 작업을 동시에 시작합니다. Velocity Move 는 작업속도까지 가속한 후에 작업속도를 유지하며 정지함수가 호출될 때까지 지정한 방향으로의 모션을 계속 수행합니다. 이 함수를 사용하면 여러 개의 축이 동시에 작업을 시작합니다. 따라서 이 함수는 여러축이 동기를 맞추어 작업을 시작해야하는 경우에 유용하게 사용될 수 있습니다. 속도 패턴은 각 축에 대하여 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

▶ *nDirList* : 방향을 지시하는 값의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다. 모션의 방향을 지시하는 값은 다음과 같습니다.

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

예 제

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

#define X_AXIS    0
#define Y_AXIS    0

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_LX501, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    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);
    // Stop 명령(키보드)이 있을 때 까지 Velocity Move 지속 //
    while(!kbhit())
        ;
    // 감속 후 정지 //
    COMILX_MC_StopAll(hDevice, 2, nAxisList);

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

▣ COMILX_MC_StartMoveAll

함수 원형

void COMILX_MC_StartMoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fDistList[])

함수 설명

여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 동시에 시작합니다. 이 함수를 사용하면 여러 개의 축이 동시에 작업을 시작합니다. 따라서 이 함수는 여러축이 동기를 맞추어 작업을 시작해야하는 경우에 유용하게 사용될 수 있습니다.

이 함수는 모션(Motion)을 시작 시킨 후에 바로 Return 합니다. 속도 패턴은 각 축에 대하여 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

▶ *fDistList* : 이동할 거리값의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다. 이동 거리값은 현재의 위치에 대한 상대 좌표이며 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리값 1은 1Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_MoveAll 함수가 모션이 완료될 때까지 Return 되지 않는데 반하여, 이 함수는 지정한 모션을 시작시킨 후에 바로 Return 하게 됩니다.

□ COMILX_MC_StartMoveToAll 함수가 절대좌표로의 이동을 수행하는데 반하여, 이 함수는 현재 위치에서 상대적인 거리를 파라미터로하여 이동을 수행합니다.

예 제

본 예제는 X 축과 Y 축을 동시에 제어하는 것에 대한 예제입니다. 본 예제에서는 X 축과 Y 축의 모션은 동시에 시작하되 각 축의 속도와 이동 거리는 다르게 설정할 수 있다는 것을 보여주기 위한 예제입니다.

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

#define X_AXIS    0
#define Y_AXIS    0

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_LX501, 0);
   if(hDevice == INVALID_HANDLE_VALUE)
       exit(-1); // Load Device Failure

   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_StartMoveAll (hDevice, 2, nAxisList, fDistList);
   while(!COMILX_MC_AllDone (hDevice, 2, nAxisList))
       ;

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

▣ COMILX_MC_MoveAll

함수 원형

void COMILX_MC_MoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fDistList[])

함수 설명

여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 함수를 사용하면 여러 개의 축이 동시에 작업을 시작합니다. 따라서 이 함수는 여러축이 동기를 맞추어 작업을 시작해야하는 경우에 유용하게 사용될 수 있습니다.

속도 패턴은 각 축에 대하여 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

▶ *fDistList* : 이동할 거리값의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다. 이동 거리값은 현재의 위치에 대한 상대 좌표이며 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, fDistance 값 1 은 1Pulse 출력을 의미합니다.

참 고

□ 이 함수는 지정된 모든 축의 모션이 완료될 때까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_MoveToAll 함수가 절대좌표로의 이동을 수행하는데 반하여, 이 함수는 현재 위치에서 상대적인 거리를 파라미터로하여 이동을 수행합니다.

예 제

본 예제는 X 축과 Y 축을 동시에 제어하는 것에 대한 예제입니다. 본 예제에서는 X 축과 Y 축의 모션은 동시에 시작하되 각 축의 속도와 이동 거리는 다르게 설정할 수

있다는 것을 보여주기 위한 예제입니다.

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

#define X_AXIS    0
#define Y_AXIS    0

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_LX501, 0);
   if(hDevice == INVALID_HANDLE_VALUE)
       exit(-1); // Load Device Failure

   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_StartMoveToAll

함수 원형

void COMILX_MC_StartMoveToAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fPosList[])

함수 설명

여러 개의 축에 대하여 지정한 절대좌표로의 이동을 시작합니다. 이 함수를 사용하면 여러 개의 축이 동시에 작업을 시작합니다. 따라서 이 함수는 여러축이 동기를 맞추어 작업을 시작해야하는 경우에 유용하게 사용될 수 있습니다. 속도 패턴은 각 축에 대하여 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

▶ *fPosList* : 절대좌표값의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다. 좌표에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 좌표의 단위는 Pulse 수가 됩니다. 즉, 좌표값 1 은 1Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_MoveToAll 함수가 모션이 완료될 때까지 Return 되지 않는데 반하여, 이 함수는 지정한 모션을 시작시킨 후에 바로 Return 하게 됩니다.

□ COMILX_MC_StartMoveAll 함수가 현재위치에 대한 상대좌표로의 이동을 수행하는데 반하여, 이 함수는 절대좌표로의 이동을 수행합니다.

예 제

본 예제는 X 축과 Y 축을 동시에 제어하는 것에 대한 예제입니다. 본 예제에서는 X 축과 Y 축의 모션은 동시에 시작하되 각 축의 속도와 이동 거리는 다르게 설정할 수 있다는 것을 보여주기 위한 예제입니다.

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

```
#include "comidaslx.h"

#define X_AXIS    0
#define Y_AXIS    0

void main()
{
   int nAxisList[2]={X_AXIS, Y_AXIS};
   double fPosList [2]={20000, 40000};
   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_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_StartMoveToAll (hDevice, 2, nAxisList, fPosList);
   while(!COMILX_MC_AllDone (hDevice, 2, nAxisList))
       ;

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

▣ COMILX_MC_MoveToAll

함수 원형

void COMILX_MC_MoveToAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fPosList[])

함수 설명

여러 개의 축에 대하여 지정한 절대좌표로의 이동을 수행합니다. 이 함수를 사용하면 여러 개의 축이 동시에 작업을 시작합니다. 따라서 이 함수는 여러축이 동기를 맞추어 작업을 시작해야하는 경우에 유용하게 사용될 수 있습니다. 속도 패턴은 각 축에 대하여 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 함수등에 의해 설정된 대로 이루어집니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

▶ *fPosList* : 절대좌표값의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다. 좌표에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 좌표의 단위는 Pulse 수가 됩니다. 즉, 좌표값 1 은 1Pulse 출력을 의미합니다.

참 고

□ 이 함수는 모션이 완료될 때까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_MoveAll 함수가 현재위치에 대한 상대좌표로의 이동을 수행하는데 반하여, 이 함수는 절대좌표로의 이동을 수행합니다.

예 제

본 예제는 X 축과 Y 축을 동시에 제어하는 것에 대한 예제입니다. 본 예제에서는 X 축과 Y 축의 모션은 동시에 시작하되 각 축의 속도와 이동 거리는 다르게 설정할 수 있다는 것을 보여주기 위한 예제입니다.

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

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

#define X_AXIS    0
#define Y_AXIS    0

void main()
{
   int nAxisList[2]={X_AXIS, Y_AXIS};
   double fPosList [2]={20000, 40000};
   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_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_MoveToAll (hDevice, 2, nAxisList, fPosList);

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

▣ COMILX_MC_StopAll

함수 원형

void COMILX_MC_StopAll (HANDLE hDevice, int nNumAxis, int nAxisList[])

함수 설명

여러 개의 축에 대한 모션을 동시에 감속 후 정지합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

예 제

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

#define X_AXIS    0
#define Y_AXIS    0

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_LX501, 0);
   if(hDevice == INVALID_HANDLE_VALUE)
       exit(-1); // Load Device Failure

   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);
   // Stop 명령(키보드)이 있을 때 까지 Velocity Move 지속 //
   while(!kbhit())
```

```
            ;
        // 감속 후 정지 //
        COMILX_MC_StopAll(hDevice, 2, nAxisList);

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

▣ COMILX_MC_EmgStopAll

함수 원형

void COMILX_MC_EmgStopAll (HANDLE hDevice, int nNumAxis, int nAxisList[])

함수 설명

여러 개의 축에 대한 모션을 동시에 감속없이 즉시 정지합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

예 제

본 예제는 X 축과 Y 축을 동시에 Velocity Move 작업을 수행하면서 Digital Input CH0 가 ON 이 되면 즉시 정지하고, 사용자가 키입력을 하면 정상 종료(감속 후 정지)를 하는 것에 대한 예제입니다.

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

#define X_AXIS    0
#define Y_AXIS    0

void main()
{
   int nAxisList[2]={X_AXIS, Y_AXIS};
   int nDirList[2]={1,1};
   int bActive;
   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_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);
    bActive = TRUE;
    // Stop 명령(키보드)이 있을 때 까지 Velocity Move 지속 //
    while(!kbhit()){
        // DI CH0 가 ON 되면 즉시 정지 //
        if(COMILX_DI_GetOne(hDevice, 0)){
            COMILX_MC_EmgStopAll(hDevice, 2, nAxisList);
            bActive = FALSE;
        }
    }
    // 감속 후 정지 //
    if(bActive) COMILX_MC_StopAll(hDevice, 2, nAxisList);

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

▣ COMILX_MC_AllDone

함수 원형

BOOL COMILX_MC_AllDone (HANDLE hDevice, int nNumAxis, int nAxisList[])

함수 설명

여러 개의 축에 대하여 지정한 모든 축의 모션이 완료됐는지를 체크합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

반환값

지정한 모든 축의 모션이 완료됐는지를 알려줍니다.

Value	Meaning
0	모션이 완료되지 않았음
1	모션이 완료됨

참 고

COMILX_MC_SetMioCfgINP() 함수를 통하여 INP 기능이 Enable 되면 command 펄스 출력이 완료되도 INP 입력이 ON 이 되지 않으면 모션이 완료되지 않은 것으로 간주됩니다.

예 제

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

#define X_AXIS    0
#define Y_AXIS    0

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_LX501, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    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_StartMoveAll (hDevice, 2, nAxisList, fDistList);
    while(!COMILX_MC_AllDone (hDevice, 2, nAxisList))
        ;

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

5.4.4 Coordinated Motion 함수

이 단원에서는 Coordinated Motion 에 관련된 함수들을 소개합니다. Coordinated Motion 이란 두 축 이상의 축이 연동되어 직선 보간(Linear Interpolation), 원호 보간(Circular Interpolation) 등의 모션을 수행하는 것을 의미합니다.

Multi-Axis 동시 제어 기능도 여러 개의 축을 제어하되 각 축이 서로 연동되어서 모션을 수행하는 것이 아니고 각 축이 개별적으로 모션을 수행하되 동시에 시작하는 것임에 반하여 Coordinated Motion 은 여러 개의 축이 서로 연동되어서 보간 이동을 수행한다는 것이 Multi-Aixs 동시 제어와 차이가 있다.

Coordinated Motion 에 관련된 함수들은 다음과 같습니다.

BOOL COMILX_MC_MapAxes (HANDLE hDevice, int nMapIndex, unsigned char bMapMask) Coordinated Motion 을 수행할 축들을 그룹화합니다
void COMILX_MC_SetSpeedModeMx (HANDLE hDevice, int nMapIndex, int nModeIndex) Coordinated Motion 의 속도 모드를 설정합니다.
void COMILX_MC_SetSpeedMx (HANDLE hDevice, int nMapIndex, double fSpeed, double fAccel) Coordinated Motion 의 속도 및 가속도를 설정합니다
void COMILX_MC_StartLine(HANDLE hDevice, int nMapIndex, double fDistList[]) 현재 위치로부터의 상대 좌표로의 직선 보간 이동을 수행합니다.
void COMILX_MC_Line (HANDLE hDevice, int nMapIndex, double fDistList[]) 현재 위치로부터의 상대 좌표로의 직선 보간 이동을 수행합니다
void COMILX_MC_StartLineTo(HANDLE hDevice, int nMapIndex, double fPosList[]) 절대 좌표로의 직선 보간 이동을 수행합니다.
void COMILX_MC_LineTo(HANDLE hDevice, int nMapIndex, double fPosList[]) 절대 좌표로의 직선 보간 이동을 수행합니다.
void COMILX_MC_StartArc_a (HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fEndAngle) 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
void COMILX_MC_Arc_a (HANDLE hDevice, int nMapIndex, double fXCentOffset, double

fYCentOffset, double fEndAngle) 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
void COMILX_MC_StartArc_p (HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fXEndPointDist, double fYEndPointDist) 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
void COMILX_MC_Arc_p(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fXEndPointDist, double fYEndPointDist) 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
void COMILX_MC_StartArcTo_a(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fEndAngle) 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
void COMILX_MC_ArcTo_a(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fEndAngle) 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
void COMILX_MC_StartArcTo_p(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fXEndPos, double fYEndPos) 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
void COMILX_MC_ArcTo_p(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fXEndPos, double fYEndPos) 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.
BOOL COMILX_MC_MxDone (HANDLE hDevice, int nMapIndex) Coordinated Motion 이 완료됐는지를 체크합니다.

▣ COMILX_MC_MapAxes

함수 원형

BOOL COMILX_MC_MapAxes (HANDLE hDevice, int nMapIndex, unsigned char bMapMask)

함수 설명

Coordinated Motion 을 수행할 축들을 그룹화합니다. Coordinated Motion 축 그룹은 최대 2 개(0 과 1)까지 지정할 수 있으며 nMapIndex 가 그룹을 지정하는 인덱스값입니다. 각 축 그룹은 최대 4 개의 축을 포함할 수 있습니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nMapIndex* : 축 그룹 인덱스, 이 값은 0 또는 1 이어야 합니다.

▶ *bMapMask* : 그룹에 포함할 축들을 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

□ Coordinated Motion 에 관련된 모든 함수들을 사용하기 전에 먼저 이 함수를 이용하여 축들을 그룹화하여야 합니다. Coordinated Motion 에 관련된 모든 함수들은 Map Index 를 파라미터로 입력받게 되어 있는데 이 함수의 nMapIndex 에 지정한 값을 입력하면 됩니다.

예 제

▣ 예제 1

본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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

void main()
```

```
{
   double fDistList [2]={10000, 20000};
   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
   // 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=8000 //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
   // Move to relative coord. (10000, 20000) //
   COMILX_MC_Line(hDevice, MAP0, fDistList);

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

▣ 예제 2

본 예제는 2 개의 축그룹을 동시에 Coordinated Motion 을 수행하는 것을 예로 보이기 위한 것입니다. X 축과 Y 축을 MAP0 에 맵핑하고 Z 축과 U 축을 MAP1 에 맵핑하여 두개의 축 그룹을 동시에 Coordinated Motion 을 수행합니다.

```
#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
#define MAP1  1

void main()
{
   double fDistList1 [2]={10000, 20000};
double fDistList2 [2]={8000, 5000};

   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
```

```
    // Map X&Y axis to MAP0 //
    COMILX_MC_MapAxes(hDevice, MAP0, X_MASK|Y_MASK);
    // Map Z&U axis to MAP1 //
    COMILX_MC_MapAxes(hDevice, MAP1, Z_MASK|U_MASK);

    // Set speed mode of MAP0 as Trapezoidal //
    COMILX_MC_SetSpeedModeMx(hDevice, MAP0, 1);
    // Set speed & accel of MAP0 => V=5000,  Acc=8000 //
    COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
    // Move to relative coord. (10000, 20000) //

    // Set speed mode of MAP1 as Trapezoidal //
    COMILX_MC_SetSpeedModeMx(hDevice, MAP1, 1);
    // Set speed & accel of MAP1 => V=2000,  Acc=5000 //
    COMILX_MC_SetSpeedMx(hDevice, MAP1, 2000, 5000);

    // Move to relative coord of (X, Y) => (10000, 20000) //
    COMILX_MC_StartLine(hDevice, MAP0, fDistList1);
    // Move to relative coord of (Z, U) => (8000, 5000) //
    COMILX_MC_StartLine(hDevice, MAP1, fDistList2);

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

▣ COMILX_MC_SetSpeedModeMx

함수 원형

void COMILX_MC_SetSpeedModeMx (HANDLE hDevice, int nMapIndex, int nModeIndex)

함수 설명

Coordinated Motion 의 속도 모드를 설정합니다. 단, 이 함수는 Motion 에 바로 영향을 주는 것이 아니고 Line, Arc 등의 Coordinated Motion 이송 함수가 수행될 때 설정된 내용이 적용됩니다.

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.

▶ *nModeIndex* : 속도 모드를 지정합니다. 속도 모드는 아래의 표와 같이 3 가지로 설정할 수 있습니다. Coordinated Motion 에서는 S-curve 속도 모드로 설정하면 가/감속 구간에서 Linear Section 이 없는 완전한 S-curve 가/감속 모드로 구성됩니다. 각각의 속도 모드에 대한 자세한 사항은 COMILX_MC_SetSpeedMode 함수를 참조하십시오.

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

예 제

본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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

void main()
{
   double fDistList [2]={10000, 20000};
   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
   // 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=8000 //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
   // Move to relative coord. (10000, 20000) //
   COMILX_MC_Line(hDevice, MAP0, fDistList);

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

▣ COMILX_MC_SetSpeedMx

함수 원형

void COMILX_MC_SetSpeedMx (HANDLE hDevice, int nMapIndex, double fSpeed, double fAccel)

함수 설명

Coordinated Motion 의 속도 및 가속도를 설정합니다. 단, 이 함수는 Motion 에 바로 영향을 주는 것이 아니고 Line, Arc 등의 Coordinated Motion 이송 함수가 수행될 때 설정된 내용이 적용됩니다.

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* : 가속도와 감속도를 지정합니다. Coordinated Motion 에서는 가속도와 감속도가 같은값으로 설정됩니다.

참 고

☐ Coordinated Motion 에서 초기속도는 자동으로 최저 속도로 설정됩니다.

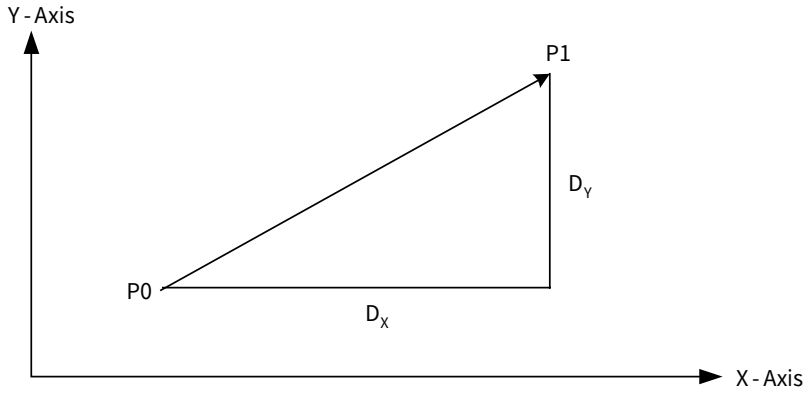
☐ Coordinated Motion 에서 초기속도는 자동으로 최저 속도로 설정됩니다.

☐ 직선 보간 함수(Line 또는 LineTo 함수)에서는 Constant, Trapezoidal, S-Curve 의 세 가지 속도 모드를 모두 적용할 수 있습니다. 단, 다음과 같은 제약사항이 있습니다.

1. 감속도(Deceleration)는 가속도와 같은값으로 자동 설정됩니다.
2. S-curve 속도모드로 설정하면 가/감속 구간에서 Linear Section 이 없는 완전한 S-curve 가/감속을 수행하게 됩니다.
3. 원형 보간 함수(Arc 또는 ArcTo 함수)에서는 가/감속도가 적용되지 않습니다.

☐ **직선 보간 이동시의 벡터 속도**

[그림 5-10]은 2 축(편의상 X, Y 축으로 가정) 직선 보간 이동을 그래프로 나타낸 것입니다.



[그림 5-10] X, Y 축간의 직선 보간 이송

그래프와 같이 P0 지점에서 P1 으로 이송시에 X 축 이송 거리 D_X 와 Y 축 이송 거리 D_Y 사이의 관계는 다음과 같습니다.

$$\Delta P = \sqrt{D_X{}^2 + D_Y{}^2}$$

각 축의 이송 거리와 각 축의 속도는 정비례하므로 벡터 속도 V, X 축의 속도 V_X 그리고 Y 축의 속도 V_Y 간의 관계는 다음과 같이 됩니다.

$$V_X = \frac{D_X \times V}{\sqrt{D_X{}^2 + D_Y{}^2}}$$

$$V_Y = \frac{D_Y \times V}{\sqrt{D_X{}^2 + D_Y{}^2}}$$

마찬가지로 3 축과 4 축 직선 보간 이동에서도 벡터 속도와 각 축의 속도간의 관계는 다음과 같은 관계식이 성립됩니다.

3 축(편의상 X, Y, Z 축으로 가정)의 경우 각 축의 속도는

$$V_i = \frac{D_i \times V}{\sqrt{D_X{}^2 + D_Y{}^2 + D_Z{}^2}}$$

과 같이 되며 4 축의 경우에는

$$V_i = \frac{D_i \times V}{\sqrt{D_X{}^2 + D_Y{}^2 + D_Z{}^2 + D_U{}^2}}$$

과 같이 됩니다.

샘플코드를 예를 들어 설명하면 다음과 같습니다.

```
#define X_Axis 1
#define Y_Axis 2
#define MAP_IDX0
// 코드의 간결성을 위하여 앞에서 행해져야할 초기화 루틴은 모두 생략 //
……
// X 축과 Y 축을 0 번 그룹으로 그룹화함 //
COMILX_MC_MapAxes (hDevice, MAP_IDX, (X_AXIS|Y_AXIS));
// Trapezoidal 속도모드로 설정 //
COMILX_MC_SetSpeedModeMx(hDevice, MAP_IDX, 1);
// 벡터속도 1000 PPS, 벡터가속도 2000 PPS/sec 로 설정 //
COMILX_MC_SetSpeedMx(hDevice, MAP_IDX, 1000, 2000);
double fDistList[2]={3000, 4000};
COMILX_MC_Line(hDevice, MAP_IDX, fDistList)
```

위의 코드는 현재 위치가 (0,0)이라고 가정할 때 (3000, 4000)의 좌표로 직선 보간 이동을 수행합니다. 벡터 속도를 1000 으로 지정하였으므로 각 축의 속도를 계산해본다면

$$V_X = \frac{D_X \times V}{\sqrt{D_X{}^2 + D_Y{}^2}} = \frac{3000 \times 1000}{\sqrt{3000^2 + 4000^2}} = 600$$

$$V_Y = \frac{D_Y \times V}{\sqrt{D_X{}^2 + D_Y{}^2}} = \frac{4000 \times 1000}{\sqrt{3000^2 + 4000^2}} = 800$$

과 같이 됩니다.

예 제

본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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

void main()
{
   double fDistList [2]={10000, 20000};
   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
   // 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=8000 //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
   // Move to relative coord. (10000, 20000) //
   COMILX_MC_Line(hDevice, MAP0, fDistList);
   COMILX_UnloadDevice(hDevice);
   COMILX_UnloadDll();
   }
```

▣ COMILX_MC_StartLine

함수 원형

void COMILX_MC_StartLine(HANDLE hDevice, int nMapIndex, double fDistList[])

함수 설명

이 함수는 현재 위치로부터의 상대 좌표로의 직선 보간 이동을 수행합니다.

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.

▶ *fDistList* : 현재 위치로부터의 상대적인 이동 좌표값(각 축의 이동 거리값)의 배열 주소. 이 배열의 크기는 COMILX_MC_MapAxes 함수를 통하여 맵핑된 축의 수와 일치하여야 합니다. 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리값 1 은 1 Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_Line 함수가 모션이 완료될 때까지 Return 되지 않는데 반하여, 이 함수는 지정한 모션을 시작시킨 후에 바로 Return 하게 됩니다.

□ COMILX_MC_StartLineTo 함수가 절대좌표로의 직선 보간 이동을 수행하는데 반하여, 이 함수는 현재 위치에서 상대적인 거리를 파라미터로하여 직선 보간 이동을 수행합니다.

예 제

▣ 예제 1

본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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

void main()
{
   double fDistList [2]={10000, 20000};
   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
   // 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=8000 //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
   // Move to relative coord. (10000, 20000) //
   COMILX_MC_StartLine(hDevice, MAP0, fDistList);
   // Coordinated Motion이 완료될때까지 기다린다. //
   while(!COMILX_MC_MxDone(hDevice, MAP0))
       ;

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

▣ COMILX_MC_Line

함수 원형

void COMILX_MC_Line (HANDLE hDevice, int nMapIndex, double fDistList[])

함수 설명

이 함수는 현재 위치로부터의 상대 좌표로의 직선 보간 이동을 수행합니다.

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.

▶ *fDistList* : 현재 위치로부터의 상대적인 이동 좌표값(각 축의 이동 거리값)의 배열 주소. 이 배열의 크기는 COMILX_MC_MapAxes 함수를 통하여 맵핑된 축의 수와 일치하여야 합니다. 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리값 1 은 1 Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_StartLine 함수가 모션이 완료되는 것을 기다리지 않고 바로 Return 하는데 반하여, 이 함수는 지정한 상대좌표로의 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_LineTo 함수가 절대좌표로의 직선 보간 이동을 수행하는데 반하여, 이 함수는 현재 위치에서 상대적인 거리를 파라미터로하여 직선 보간 이동을 수행합니다.

예 제

본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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

void main()
{
   double fDistList [2]={10000, 20000};
   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
   // 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=8000 //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
   // Move to relative coord. (10000, 20000) //
   COMILX_MC_Line(hDevice, MAP0, fDistList);

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

▣ COMILX_MC_StartLineTo

함수 원형

void COMILX_MC_StartLineTo(HANDLE hDevice, int nMapIndex, double fPosList[])

함수 설명

이 함수는 절대 좌표로의 직선 보간 이동을 수행합니다.

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.

▶ *fPosList* : 이동할 목표 절대좌표값(각 축의 절대좌표값)의 배열 주소. 이 배열의 크기는 COMILX_MC_MapAxes 함수를 통하여 맵핑된 축의 수와 일치하여야 합니다. 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리값 1은 1 Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_LineTo 함수가 모션이 완료될 때까지 Return 되지 않는데 반하여, 이 함수는 지정한 모션을 시작시킨 후에 바로 Return 하게 됩니다.

□ COMILX_MC_StartLine 함수가 상대좌표로의 직선 보간 이동을 수행하는데 반하여, 이 함수는 절대 좌표로의 직선 보간 이동을 수행합니다.

예 제

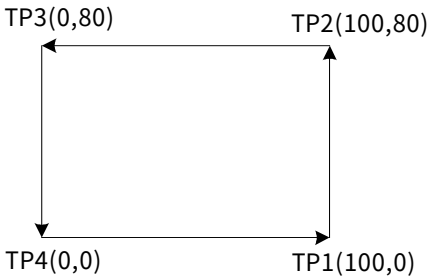
본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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
```

```
void main()
{
   double fPosList [2]={10000, 20000};
   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
   // 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=8000 //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
   // Move to absolute coord. (10000, 20000) //
   COMILX_MC_StartLineTo(hDevice, MAP0, fPosList);
   // Coordinated Motion이 완료될때까지 기다린다. //
   while(!COMILX_MC_MxDone(hDevice, MAP0))
       ;

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

▣ 예제 2

본 예제는 X 축과 Y 축을 그룹화하여 아래 그림과 같이 Coordinated Motion 을 수행하는 예제입니다. 그리고 1 회전에 필요한 펄스수가 3600 펄스라 가정하여 거리의 단위를 각도(1°)로, 속도의 단위를 rpm 으로 설정합니다.




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

```
#include "comidaslx.h"

#define X_AXIS    0
#define Y_AXIS    1

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

#define MAP0  0

void main()
{
   double fPosList[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
   // X 축과 Y 축에 대하여 논리거리 및 논리속도 정의 //
   COMILX_MC_SetUnitDistance(hDevice, X_AXIS, 10);
   COMILX_MC_SetUnitDistance(hDevice, Y_AXIS, 10);
   COMILX_MC_SetUnitSpeed(hDevice, X_AXIS, 3600./60);
   COMILX_MC_SetUnitSpeed(hDevice, Y_AXIS, 3600./60);

   // X&Y 축의 Command Position 의 현재 좌표를 0 으로 초기화한다. //
   COMILX_MC_SetPosition_C (hDevice, X_AXIS, 0);
   COMILX_MC_SetPosition_C (hDevice, Y_AXIS, 0);


   // 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=60(RPM),  Acc=100(RPM/SEC) //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 60, 100);

   // Move to (100,0) //
   fPosList[0]=100;   fPosList [1]=0;
   COMILX_MC_StartLineTo(hDevice, MAP0, fPosList);
   while(!COMILX_MC_MxDone(hDevice, MAP0))
       ;
   // Move to (100,80) //
   fPosList[0]=100;   fPosList[1]=80;
   COMILX_MC_StartLineTo(hDevice, MAP0, fPosList);
   while(!COMILX_MC_MxDone(hDevice, MAP0))
       ;
   // Move to (0,80) //
   fPosList[0]=0; fPosList[1]=80;
   COMILX_MC_StartLineTo(hDevice, MAP0, fPosList);
```

```
    while(!COMILX_MC_MxDone(hDevice, MAP0))
        ;
    // Move to (0,0) //
    fPosList[0]=0; fPosList[1]=0;
    COMILX_MC_StartLineTo(hDevice, MAP0, fPosList);
    while(!COMILX_MC_MxDone(hDevice, MAP0))
        ;

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

▣ COMILX_MC_LineTo

함수 원형

void COMILX_MC_LineTo(HANDLE hDevice, int nMapIndex, double fPosList[])

함수 설명

이 함수는 절대 좌표로의 직선 보간 이동을 수행합니다.

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.

▶ *fPosList* : 이동할 목표 절대좌표값(각 축의 절대좌표값)의 배열 주소. 이 배열의 크기는 COMILX_MC_MapAxes 함수를 통하여 맵핑된 축의 수와 일치하여야 합니다. 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리값 1은 1 Pulse 출력을 의미합니다.

참 고

□ COMILX_MC_StartLineTo 함수가 모션이 완료되는 것을 기다리지 않고 바로 Return 하는데 반하여, 이 함수는 지정한 상대좌표로의 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_Line 함수가 상대좌표로의 직선 보간 이동을 수행하는데 반하여, 이 함수는 절대 좌표로의 직선 보간 이동을 수행합니다.

예 제

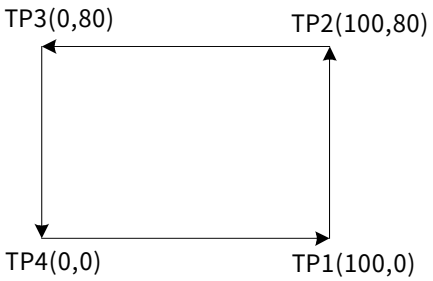
본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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

void main()
{
   double fPosList[2]={10000, 20000};
   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
   // 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=8000 //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 5000, 8000);
   // Move to absolute coord. (10000, 20000) //
   COMILX_MC_StartLineTo(hDevice, MAP0, fPosList);
   // Coordinated Motion이 완료될때까지 기다린다. //
   while(!COMILX_MC_MxDone(hDevice, MAP0))
       ;

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

▣ 예제 2

본 예제는 X 축과 Y 축을 그룹화하여 아래 그림과 같이 Coordinated Motion 을 수행하는 예제입니다. 그리고 1 회전에 필요한 펄스수가 3600 펄스라 가정하여 거리의 단위를 각도(1°)로, 속도의 단위를 rpm 으로 설정합니다.



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

#define X_AXIS    0
#define Y_AXIS    1

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

#define MAP0  0

void main()
{
   double fPosList[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
   // X 축과 Y 축에 대하여 논리거리 및 논리속도 정의 //
   COMILX_MC_SetUnitDistance(hDevice, X_AXIS, 10);
   COMILX_MC_SetUnitDistance(hDevice, Y_AXIS, 10);
   COMILX_MC_SetUnitSpeed(hDevice, X_AXIS, 3600./60);
   COMILX_MC_SetUnitSpeed(hDevice, Y_AXIS, 3600./60);

   // X&Y 축의 Command Position 의 현재 좌표를 0 으로 초기화한다. //
   COMILX_MC_SetPosition_C (hDevice, X_AXIS, 0);
   COMILX_MC_SetPosition_C (hDevice, Y_AXIS, 0);


   // 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=60(RPM),  Acc=100(RPM/SEC) //
   COMILX_MC_SetSpeedMx(hDevice, MAP0, 60, 100);

   // Move to (100,0) //
   fPosList[0]=100;   fPosList[1]=0;
   COMILX_MC_LineTo(hDevice, MAP0, fPosList);
   // Move to (100,80) //
   fPosList[0]=100;   fPosList[1]=80;
   COMILX_MC_LineTo(hDevice, MAP0, fPosList);
   // Move to (0,80) //
   fPosList[0]=0; fPosList[1]=80;
   COMILX_MC_LineTo(hDevice, MAP0, fPosList);
```

```
    // Move to (0,0) //
    fPosList[0]=0; fPosList[1]=0;
    COMILX_MC_LineTo(hDevice, MAP0, fPosList);

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

▣ COMILX_MC_StartArc_a

함수 원형

void COMILX_MC_StartArc_a(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fEndAngle)

함수 설명

이 함수는 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다. 이 함수는 End Point 에 대한 정보를 각도값으로 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.

▶ *fXCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 X 축 상대좌표값

▶ *fYCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 Y 축 상대좌표값

▶ *fEndAngle* : 원호보간 이동을 완료할 목표지점의 현재 위치에 대한 각도값을 Degree(°)값으로 지정합니다. 각도의 부호가 (+)이면 반시계방향, (-)이면 시계방향으로의 이동을 의미합니다.

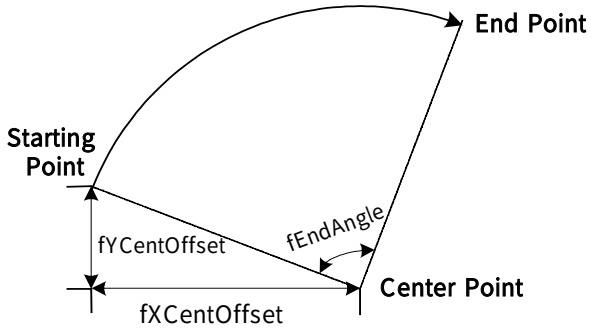
참 고

□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 이 함수는 원호 보간 이동을 시작시킨후 바로 Return 합니다.

□ COMILX_MC_StartArc_p 함수가 원호 보간 이동을 완료할 목표지점의 좌표값을 파라미터로 사용하는데 반하여 이 함수는 각도값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_StartArc_p 나 COMILX_MC_StartArc_a 함수중에 하나를 사용할 수 있습니다.

□ COMILX_MC_StartArc_a 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 5-11]과 같습니다.



[그림 5-11] COMILX_MC_StartArc_a 함수를 사용한 원호 보간 이동

▣ COMILX_MC_Arc_a

함수 원형

void COMILX_MC_Arc_a(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fEndAngle)

함수 설명

이 함수는 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다. 이 함수는 End Point 에 대한 정보를 각도값으로 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.

▶ *fXCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 X 축상 상대 좌표

▶ *fYCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 Y 축상 상대 좌표

▶ *fEndAngle* : 원호보간 이동을 완료할 목표지점의 현재 위치에 대한 각도값을 Degree(°)값으로 지정합니다. 각도의 부호가 (+)이면 반시계방향, (-)이면 시계방향으로의 이동을 의미합니다.

참 고

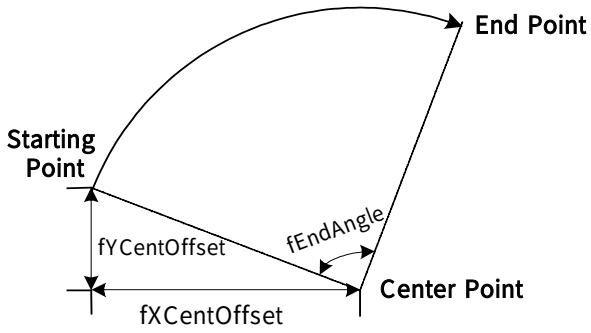
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 이 함수는 원호 보간 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_Arc_p 함수가 원호 보간 이동을 완료할 목표지점의 좌표값을 파라미터로 사용하는데 반하여 이 함수는 각도값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_Arc_p 나 COMILX_MC_Arc_a 함수중에 하나를 사용할 수 있습니다.

□ COMILX_MC_Arc_a 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의

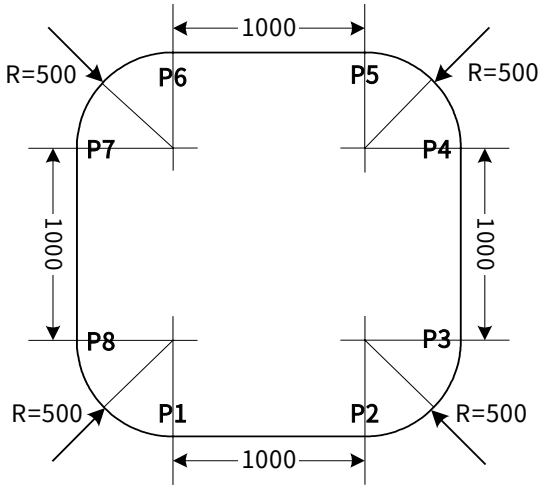
미는 [그림 5-12]과 같습니다.



[그림 5-12] COMILX_MC_Arc_a 함수를 사용한 원호 보간 이동

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.




```
#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

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
   // 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);
   // Move from P1 to P2 //
   fDistList[0]=1000; fDistList[1]=0;
   COMILX_MC_Line(hDevice, MAP0, fDistList);
   // Move from P2 to P3 //
   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_Arc_a(hDevice, MAP0, 500, 0, 90);

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

▣ COMILX_MC_StartArc_p

함수 원형

void COMILX_MC_StartArc_p(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fXEndPointDist, double fYEndPointDist, int nDir)

함수 설명

이 함수는 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다. 이 함수는 End Point 에 대한 정보를 상대좌표값으로 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.

▶ *fXCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 X 축상의 거리

▶ *fYCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 Y 축상의 거리

▶ *fXEndPointDist* : 원호보간 이동을 완료할 목표지점의 현재 위치로부터 X-축상 거리값.

▶ *fYEndPointDist* : 원호보간 이동을 완료할 목표지점의 현재 위치로부터 Y-축상 거리값.

▶ *nDir* : 회전 방향을 지정합니다.

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

참 고

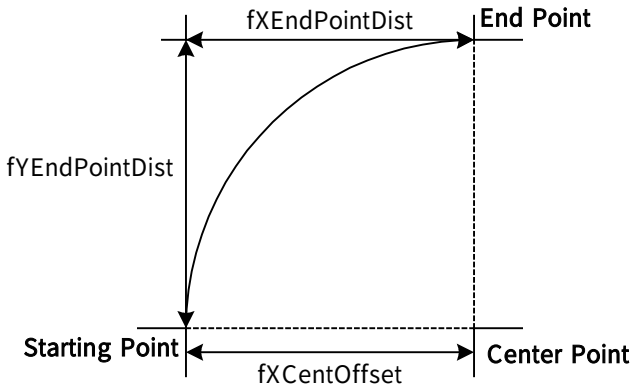
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 이 함수는 원호 보간 이동을 시작시킨후 바로 Return 합니다.

□ COMILX_MC_StartArc_a 함수가 원호 보간 이동을 완료할 목표지점의 각도를 파라미터로 사용하는데 반하여 이 함수는 상대 좌표값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_StartArc_p 나 COMILX_MC_StartArc_a 함수중에 하나를 사용할 수 있습니다.

□ fXEndPointDist 값과 fYEndPointDist 값이 모두 0 으로 지정되면 완전한 원을 그리게 됩니다.

□ COMILX_MC_StartArc_p 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 5-13]과 같습니다.



[그림 5-13] COMILX_MC_StartArc_p 함수를 사용한 원호 보간 이동

▣ COMILX_MC_Arc_p

함수 원형

void COMILX_MC_Arc_p(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fXEndPointDist, double fYEndPointDist, int nDir)

함수 설명

이 함수는 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다. 이 함수는 End Point 에 대한 정보를 상대좌표값으로 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.

▶ *fXCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 X 축상의 거리.

▶ *fYCentOffset* : 현재 위치(시작 위치)로부터 원의 중심까지 Y 축상의 거리

▶ *fXEndPointDist* : 원호보간 이동을 완료할 목표지점의 현재 위치로부터 X-축상 거리값.

▶ *fYEndPointDist* : 원호보간 이동을 완료할 목표지점의 현재 위치로부터 Y-축상 거리값.

▶ *nDir* : 회전 방향을 지정합니다.

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

참 고

□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

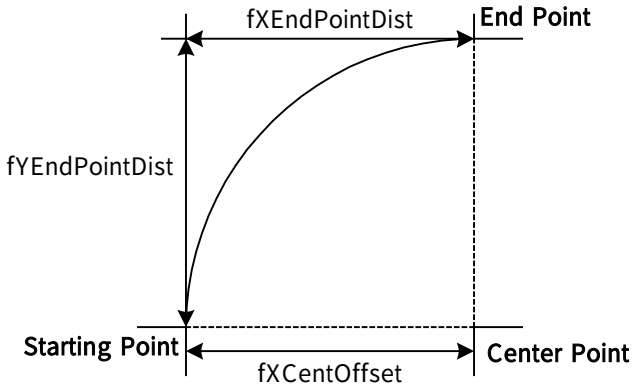
□ 이 함수는 원호 보간 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됨

니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ fXEndPointDist 값과 fYEndPointDist 값이 모두 0 으로 지정되면 완전한 원을 그리게 됩니다.

□ COMILX_MC_StartArc_a 함수가 원호 보간 이동을 완료할 목표지점의 각도를 파라미터로 사용하는데 반하여 이 함수는 상대 좌표값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_StartArc_p 나 COMILX_MC_StartArc_a 함수중에 하나를 사용할 수 있습니다.

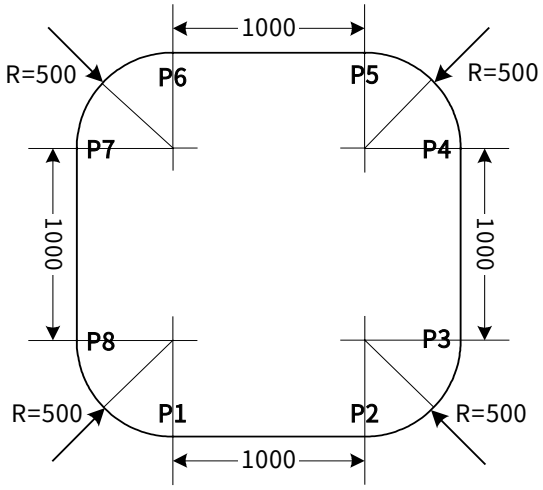
□ COMILX_MC_StartArc_p 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 5-14]과 같습니다.



[그림 5-14] COMILX_MC_StartArc_p 함수를 사용한 원호 보간 이동

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.




```
#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

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
   // 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);
   // Move from P1 to P2 //
   fDistList[0]=1000; fDistList[1]=0;
   COMILX_MC_Line(hDevice, MAP0, fDistList);
   // Move from P2 to P3 //
   COMILX_MC_Arc_p(hDevice, MAP0, 0, 500, 500, 500, 1);
```

```
    // 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_p(hDevice, MAP0, -500, 0, -500, 500, 1);
    // 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_p(hDevice, MAP0, 0, -500, -500, -500, 1);
    // Move from P7 to P8 //
    fDistList[0]=0; fDistList[1]=-1000;
    COMILX_MC_Line(hDevice, MAP0, fDistList);
    // Move from P8 to P1 //
    COMILX_MC_Arc_p(hDevice, MAP0, 500, 0, 500, -500, 1);

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

▣ COMILX_MC_StartArcTo_a

함수 원형

void COMILX_MC_StartArcTo_a(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fEndAngle)

함수 설명

이 함수는 원호보간 이동을 수행합니다. 이 함수는 중심점의 좌표값을 절대좌표값으로 설정하며 원호보간 이동의 완료지점에 대한 정보를 각도로 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 절대좌표

▶ *fYCent* : 중심점의 Y 축 절대좌표

▶ *fEndAngle* : 원호보간 이동을 완료할 목표지점의 현재 위치에 대한 각도값을 Degree(°)값으로 지정합니다. 각도의 부호가 (+)이면 반시계방향, (-)이면 시계방향으로의 이동을 의미합니다.

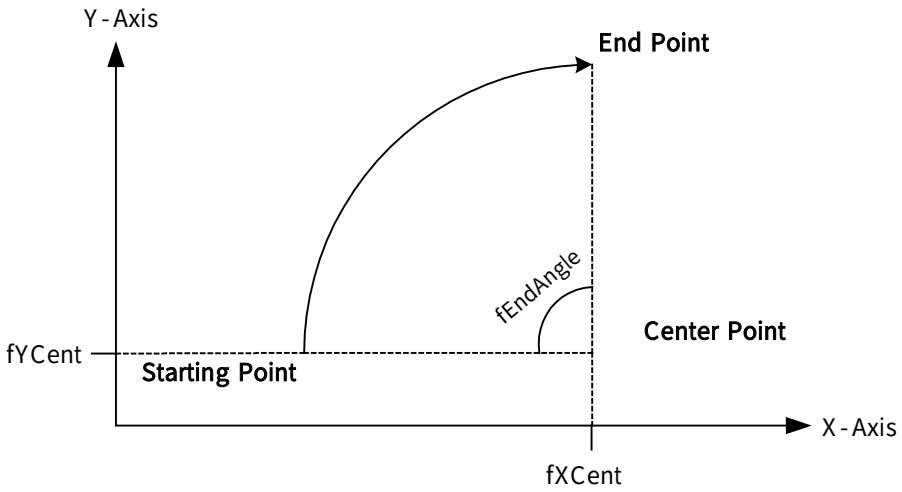
참 고

□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 이 함수는 원호 보간 이동을 시작시킨후 바로 Return 합니다.

□ COMILX_MC_StartArcTo_p 함수가 원호 보간 이동을 완료할 목표지점의 좌표값을 파라미터로 사용하는데 반하여 이 함수는 각도값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_StartArc_p 나 COMILX_MC_StartArc_a 함수중에 하나를 사용할 수 있습니다.

□ COMILX_MC_StartArcTo_a 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 5-15]와 같습니다.



[그림 5-15] COMILX_MC_StartArcTo_a 함수를 사용한 원호 보간 이동

▣ COMILX_MC_ArcTo_a

함수 원형

void COMILX_MC_ArcTo_a(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fEndAngle)

함수 설명

이 함수는 원호보간 이동을 수행합니다. 이 함수는 중심점의 좌표값을 절대좌표값으로 설정하며 원호보간 이동의 완료지점에 대한 정보를 각도로 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 절대좌표

▶ *fYCent* : 중심점의 Y 축 절대좌표

▶ *fEndAngle* : 원호보간 이동을 완료할 목표지점의 현재 위치에 대한 각도값을 Degree(°)값으로 지정합니다. 각도의 부호가 (+)이면 반시계방향, (-)이면 시계방향으로의 이동을 의미합니다.

참 고

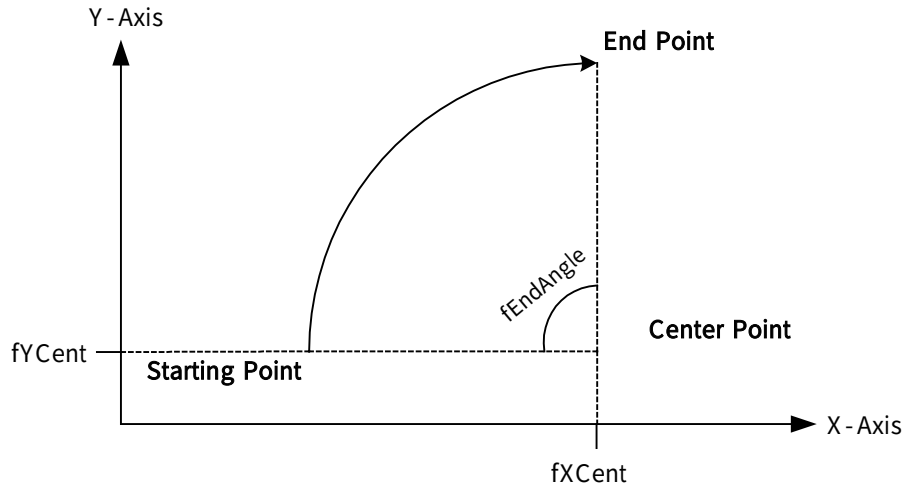
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 이 함수는 원호 보간 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ COMILX_MC_ArcTo_p 함수가 원호 보간 이동을 완료할 목표지점의 좌표값을 파라미터로 사용하는데 반하여 이 함수는 각도값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_Arc_p 나 COMILX_MC_Arc_a 함수중에 하나를 사용할 수 있습니다.

□ COMILX_MC_StartArcTo_a 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미

터의 의미는 [그림 5-16]과 같습니다.

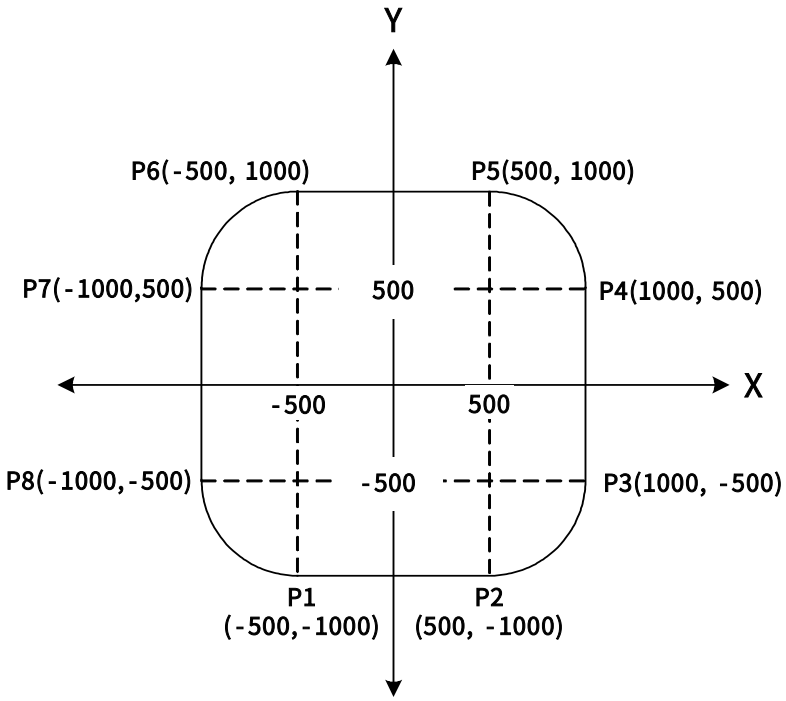


[그림 5-16] COMILX_MC_ArcTo_a 함수를 사용한 원호 보간 이동

예 제

▣ 예제 1

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. 그리고 현재 위치가 P1 의 위치에 있다고 가정합니다.

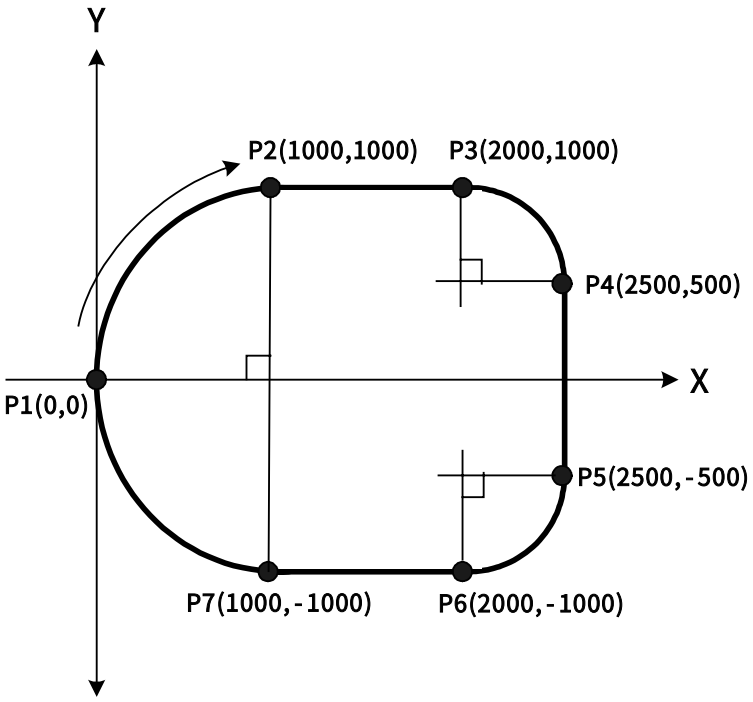



```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

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

#define MAP0  0

void main()
{
   double fPosList[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
   // 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);
    // Move from P1 to P2 //
    fPosList[0]=500; fPosList[1]=-1000;
    COMILX_MC_LineTo(hDevice, MAP0, fPosList);
    // Move from P2 to P3 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, 500, -500, 90);
    // Move from P3 to P4 //
    fPosList[0]=1000; fPosList[1]=500;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P4 to P5 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, 500, 500, 90);
    // Move from P5 to P6 //
    fPosList[0]=-500; fPosList[1]=1000;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P6 to P7 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, -500, 500, 90);
    // Move from P7 to P8 //
    fPosList[0]=-1000; fPosList[1]=-500;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P8 to P1 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, -500, -500, 90);

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

▣ 예제 2

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. 그리고 현재 위치가 P1 의 위치에 있다고 가정합니다.




```
#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

void main()
{
   double fPosList[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
   // 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);
    // Move from P1 to P2 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, 1000, 0, 90);
    // Move from P2 to P3 //
    fPosList[0]=2000; fPosList[1]=1000;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P3 to P4 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, 2000, 500, 90);
    // Move from P4 to P5 //
    fPosList[0]=2500; fPosList[1]=-500;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P5 to P6 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, 2000, -500, 90);
    // Move from P6 to P7 //
    fPosList[0]=1000; fPosList[1]=-1000;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P7 to P1 //
    COMILX_MC_ArcTo_a(hDevice, MAP0, 1000, 0, 90);

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

▣ COMILX_MC_StartArcTo_p

함수 원형

void COMILX_MC_StartArcTo_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

참 고

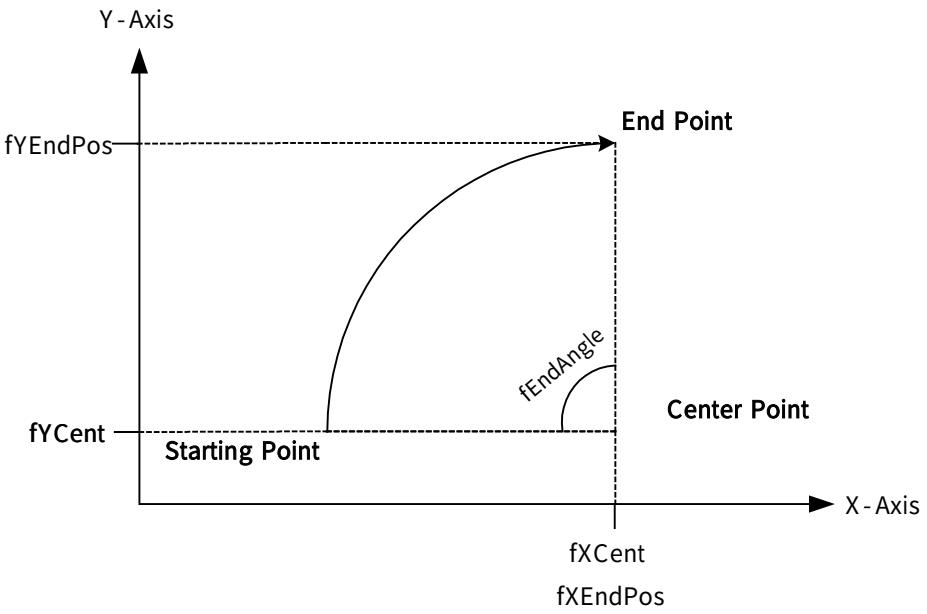
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 이 함수는 원호 보간 이동을 시작시킨후 바로 Return 합니다.

□ fXEndPos 값과 fYEndPos 값이 현재 위치(Starting Point)의 좌표값과 일치하면 완전한 원을 그리게 됩니다.

□ COMILX_MC_StartArcTo_a 함수가 원호 보간 이동을 완료할 목표지점의 각도를 파라미터로 사용하는데 반하여 이 함수는 상대 좌표값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_StartArcTo_p 나 COMILX_MC_StartArcTo_a 함수중에 하나를 사용할 수 있습니다.

□ COMILX_MC_StartArcTo_p 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 5-17]과 같습니다.



[그림 5-17] COMILX_MC_StartArcTo_p 함수를 사용한 원호 보간 이동

▣ 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

참 고

□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

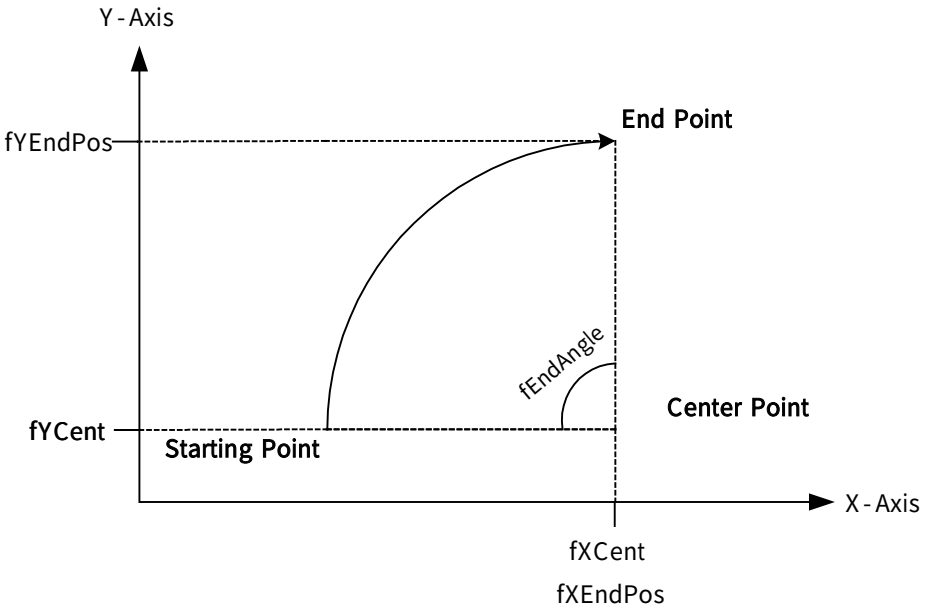
□ 이 함수는 원호 보간 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ fXEndPos 값과 fYEndPos 값이 현재 위치(Starting Point)의 좌표값과 일치하면 완전한 원을 그리게 됩니다.

□ COMILX_MC_ArcTo_a 함수가 원호 보간 이동을 완료할 목표지점의 각도를 파라미터

로 사용하는데 반하여 이 함수는 상대 좌표값을 파라미터로 사용합니다. 사용자는 편의에 따라 COMILX_MC_ArcTo_p 나 COMILX_MC_ArcTo_a 함수중에 하나를 사용할 수 있습니다.

□ COMILX_MC_ArcTo_p 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 5-18]과 같습니다.

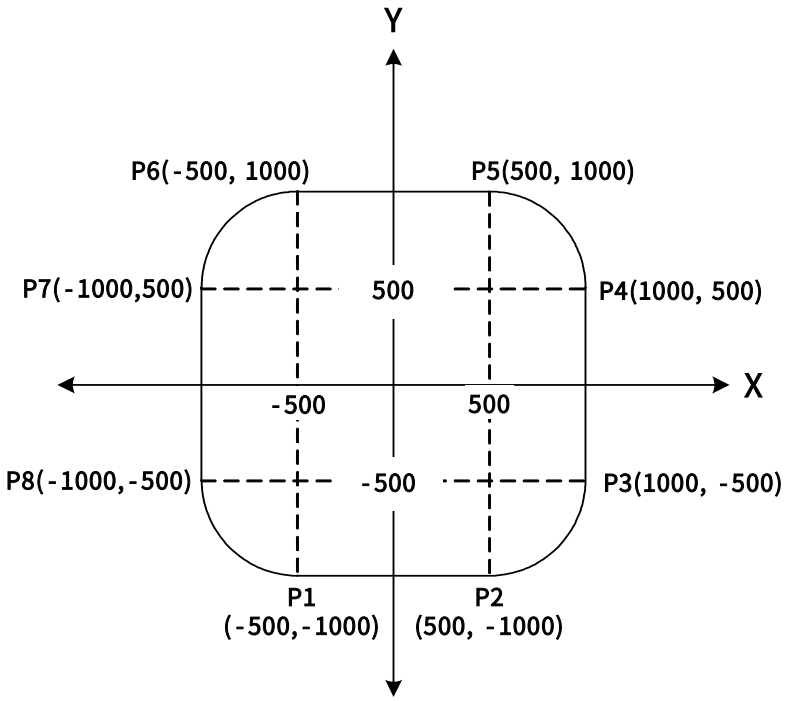


[그림 5-18] COMILX_MC_ArcTo_p 함수를 사용한 원호 보간 이동

예 제

▣ 예제 1

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. 그리고 현재 위치가 P1 의 위치에 있다고 가정합니다.

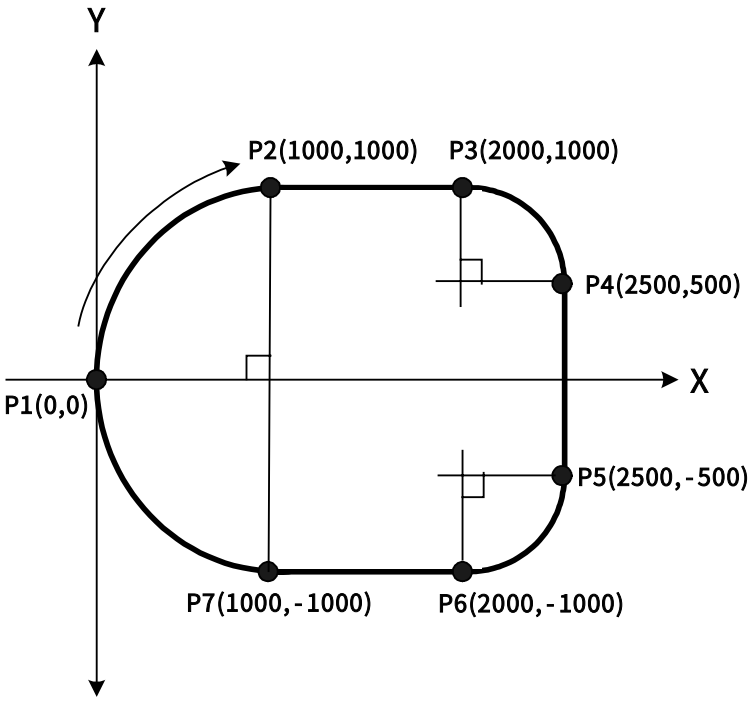



```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

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

#define MAP0  0

void main()
{
   double fPosList[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
   // 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);
    // Move from P1 to P2 //
    fPosList[0]=500; fPosList[1]=-1000;
    COMILX_MC_LineTo(hDevice, MAP0, fPosList);
    // Move from P2 to P3 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, 500, -500, 1000, -500, 1);
    // Move from P3 to P4 //
    fPosList[0]=1000; fPosList[1]=500;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P4 to P5 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, 500, 500, 500, 1000, 1);
    // Move from P5 to P6 //
    fPosList[0]=-500; fPosList[1]=1000;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P6 to P7 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, -500, 500, -1000, 500, 1);
    // Move from P7 to P8 //
    fPosList[0]=-1000; fPosList[1]=-500;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P8 to P1 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, -500, -500, -500, -1000, 1);

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

▣ 예제 2

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. 그리고 현재 위치가 P1 의 위치에 있다고 가정합니다.




```
#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

void main()
{
   double fPosList[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
   // 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);
    // Move from P1 to P2 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, 1000, 0, 1000, 1000, 0);
    // Move from P2 to P3 //
    fPosList[0]=2000; fPosList[1]=1000;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P3 to P4 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, 2000, 500, 2500, 500, 0);
    // Move from P4 to P5 //
    fPosList[0]=2500; fPosList[1]=-500;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P5 to P6 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, 2000, -500, 2000, -1000, 0);
    // Move from P6 to P7 //
    fPosList[0]=1000; fPosList[1]=-1000;
    COMILX_MC_LineTo (hDevice, MAP0, fPosList);
    // Move from P7 to P1 //
    COMILX_MC_ArcTo_p(hDevice, MAP0, 1000, 0, 0, 0, 0);

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

▣ COMILX_MC_MxDone

함수 원형

BOOL COMILX_MC_MxDone (HANDLE hDevice, int nMapIndex)

함수 설명

지정한 축그룹(nMapIndex)의 Coordinated Motion 이 완료됐는지를 체크합니다.

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.

반환값

지정한 축그룹(nMapIndex)에 맵핑되어 있는 모든 축이 모션을 완료하였으면 1 을 그렇지 않으면 0 을 반환합니다.

Value	Meaning
0	모션이 완료되지 않았음
1	모션이 완료됨

참 고

COMILX_MC_SetMioCfgINP() 함수를 통하여 INP 기능이 Enable 되면 command 펄스 출력이 완료되도 INP 입력이 ON 이 되지 않으면 모션이 완료되지 않은 것으로 간주됩니다.

5.4.5 속도 및 위치 오버라이딩(Overriding) 함수

이 단원에서는 속도 및 위치 오버라이딩 함수들을 소개합니다. 속도 오버라이딩은 모션이 진행되고 있는 중에 작업 속도를 변경하는 것을 의미합니다. 위치 오버라이딩은 Single Axis 모션 중에서 Move 나 MoveTo 와 같이 In-Position 모션을 수행하고 있는 중에 목표 거리 또는 목표 좌표를 수정하는 것을 의미 합니다.

속도 및 위치 오버라이딩에 관련된 함수는 다음과 같습니다.

void COMILX_MC_OverrideSpeedSet (HANDLE hDevice, int nChannel) Single Axis 모션이 진행되고 있는 중에 속도를 변경
void COMILX_MC_OverrideSpeedSetAll (HANDLE hDevice, int nNumAxis, int nAxisList[]) 여러 축에 대하여 동시에 속도를 변경.
void COMILX_MC_OverrideMove (HANDLE hDevice, int nChannel, double fNewDistance) COMILX_MC_StartMove 함수를 통하여 수행되는 상대좌표 In-position 모션에 대하여 상대좌표값, 즉 목표 거리값을 수정
void COMILX_MC_OverrideMoveTo (HANDLE hDevice, int nChannel, double fNewPosition) COMILX_MC_StartMoveTo 함수를 통하여 수행되는 절대좌표 In-position 모션에 대하여 목표 절대좌표값을 수정.

▣ COMILX_MC_OverrideSpeedSet

함수 원형

void COMILX_MC_OverrideSpeedSet (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 Single Axis 모션이 진행되고 있는 중에 속도를 변경(오버라이딩, Overriding)하고자할 때 사용하는 함수입니다. 속도를 오버라이딩(Overriding)하기 위해서는 먼저 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 등의 속도 패턴 설정 함수를 통하여 변경하고자 하는 속도 또는 가속도값을 설정하고 COMILX_MC_OverrideSpeedSet 함수를 통하여 설정된 속도 또는 가속도값을 실제 모션에 적용합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

참 고

□ 이 함수는 변경된 속도 패턴 설정을 실제 모션에 적용하는 역할을 합니다. 속도를 오버라이딩(Overriding)하기 위해서는 이 함수를 사용하기전에 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 등을 통하여 필요한 변경값을 설정하여야 합니다.

□ 여러축을 동시에 속도 오버라이딩(Overriding)하고자 한다면 이 함수 대신에 COMILX_MC_OverrideSpeedSetAll 함수를 사용하십시오.

□ Line 이나 Arc 와 같은 Interpolation(또는 Coordinated Motion) 함수를 사용한 경우에는 속도 오버라이딩을 사용할 수 없습니다.

예 제

본 예제는 **COMILX_MC_OverrideSpeedSet()**함수를 사용하여 속도를 오버라이딩 하는 것을 예로 보여주는 코드입니다. 본 예제는 사용자의 키보드 입력을 받아 미리 지정된 3 단계의 속도로 변경을 하면서 모션을 수행하는 예제입니다.

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

#define X_AXIS    0
void main()
```

```
{
   double fSpeed[3]={10000, 20000, 30000};
   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_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, fSpeed[0]);
   // (+)방향으로 Velocity Move 수행 //
   COMILX_MC_StartVMove(hDevice, X_AXIS, 1);
   printf("현재 %.0f(PPS)의 속도로 모션이 진행중입니다. \n"
       "아무키나 누르면 %.0f(PPS)의 속도로 변경됩니다.\n", fSpeed[0],
fSpeed[1]);
   getch();
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, fSpeed[1]);
   COMILX_MC_OverrideSpeedSet(hDevice, X_AXIS);
   printf("\n현재 %.0f(PPS)의 속도로 모션이 진행중입니다. \n"
       "아무키나 누르면 %.0f(PPS)의 속도로 변경됩니다.\n", fSpeed[1],
fSpeed[2]);

   getch();
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, fSpeed[2]);
   COMILX_MC_OverrideSpeedSet(hDevice, X_AXIS);
   printf("\n현재 %.0f(PPS)의 속도로 모션이 진행중입니다. \n"
       "아무키나 누르면 모션이 정지됩니다.", fSpeed[0], fSpeed[1]);
   getch();
   // 감속 후 정지 //
   COMILX_MC_Stop(hDevice, X_AXIS);

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

▣ COMILX_MC_OverrideSpeedSetAll

함수 원형

void COMILX_MC_OverrideSpeedSetAll (HANDLE hDevice, int nNumAxis, int nAxisList[])

함수 설명

이 함수는 Multi-Axis 동시 제어 모션이 진행되고 있는 중에 여러 축에 대하여 동시에 속도를 변경(오버라이딩, Overriding)하고자할 때 사용하는 함수입니다. 이 함수는 속도 오버라이딩을 여러축에 대하여 동시에 수행합니다. 속도를 오버라이딩(Overriding)하기 위해서는 먼저 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 등의 속도 패턴 설정 함수를 통하여 각 축에 대하여 변경하고자 하는 속도 또는 가속도값을 설정하고 COMILX_MC_OverrideSpeedSetAll 함수를 통하여 설정된 속도 또는 가속도값을 실제 모션에 적용합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nNumAxis* : 동시에 작업을 수행할 대상 축의 수

▶ *nAixsList* : 동시에 작업을 수행할 대상 축의 배열 주소값. 이 배열의 크기는 nNumAxis 값과 일치해야 합니다.

참 고

□ 이 함수는 변경된 속도 패턴 설정을 실제 모션에 적용하는 역할을 합니다. 속도를 오버라이딩(Overriding)하기 위해서는 이 함수를 사용하기전에 각 축에 대하여 COMILX_MC_SetSpeedMode, COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve 등을 통하여 필요한 변경값을 설정하여야 합니다.

□ 하나의 축에 대하여 속도 오버라이딩(Overriding)하고자 한다면 이 함수 대신에 COMILX_MC_OverrideSpeedSet 함수를 사용하십시오.

□ Line 이나 Arc 와 같은 Interpolation(또는 Coordinated Motion) 함수를 사용한 경우에는 속도 오버라이딩을 사용할 수 없습니다.

예 제

본 예제는 **COMILX_MC_OverrideSpeedSetAll()**함수를 사용하여 속도를 오버라이딩하는 것을 예로 보여주는 코드입니다. 본 예제는 **X,Y,Z** 축을 동시 제어하는 것으로써 사용자의 키보드 입력을 받아 미리 지정된 **3** 단계의 속도로 변경을 하면서

모션을 수행하는 예제입니다.

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

#define X_AXIS    0
#define Y_AXIS    1
#define Z_AXIS    2

void main()
{
   int nAxisList[3]={X_AXIS, Y_AXIS, Z_AXIS};
   int nDirList[3]={1,1,1};
   double fSpeed[3]={10000, 20000, 30000};
   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_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, fSpeed[0]);

   COMILX_MC_SetSpeedMode(hDevice, Y_AXIS, 1);
   COMILX_MC_SetAccel(hDevice, Y_AXIS, 20000, 20000);
   COMILX_MC_SetSpeed(hDevice, Y_AXIS, 0, fSpeed[0]);

   COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetAccel(hDevice, Z_AXIS, 20000, 20000);
   COMILX_MC_SetSpeed(hDevice, Z_AXIS, 0, fSpeed[0]);

   // (+)방향으로 Velocity Move 수행 //
   COMILX_MC_StartVMove(hDevice, X_AXIS, 1);
   COMILX_MC_StartVMoveAll(hDevice, 3, nAxisList, nDirList);
   printf("현재 %.0f(PPS)의 속도로 모션이 진행중입니다. \n"
       "아무키나 누르면 %.0f(PPS)의 속도로 변경됩니다.\n", fSpeed[0],
fSpeed[1]);
   getch();
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, fSpeed[1]);
   COMILX_MC_SetSpeed(hDevice, Y_AXIS, 0, fSpeed[1]);
   COMILX_MC_SetSpeed(hDevice, Z_AXIS, 0, fSpeed[1]);
   COMILX_MC_OverrideSpeedSetAll(hDevice, 3, nAxisList);
   printf("\n현재 %.0f(PPS)의 속도로 모션이 진행중입니다. \n"
       "아무키나 누르면 %.0f(PPS)의 속도로 변경됩니다.\n", fSpeed[1],
fSpeed[2]);

   getch();
```

```
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, fSpeed[2]);
    COMILX_MC_SetSpeed(hDevice, Y_AXIS, 0, fSpeed[2]);
    COMILX_MC_SetSpeed(hDevice, Z_AXIS, 0, fSpeed[2]);
    COMILX_MC_OverrideSpeedSetAll(hDevice, 3, nAxisList);
    printf("\n현재 %.0f(PPS)의 속도로 모션이 진행중입니다. \n"
        "아무키나 누르면 모션이 정지됩니다.", fSpeed[0], fSpeed[1]);
    getch();
    // 감속 후 정지 //
    COMILX_MC_StopAll(hDevice, 3, nAxisList);

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

▣ COMILX_MC_OverrideMove

함수 원형

void COMILX_MC_OverrideMove (HANDLE hDevice, int nChannel, double fNewDistance)

함수 설명

이 함수는 COMILX_MC_StartMove 함수를 통하여 수행되는 상대좌표 In-position 모션에 대하여 상대좌표값, 즉 목표 거리값을 수정(오버라이딩, Overriding)하는 함수입니다. 이 함수는 COMILX_MC_StartMove 함수를 사용하여 모션을 수행하고 있는 경우에만 사용가능한 함수입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fNewDistance* : 새로운 목표 거리값을 지정합니다. 이 값의 기준 위치는 COMILX_MC_StartMove 함수에서 사용한 기준과 같습니다. 즉 COMILX_MC_StartMove 를 실행하기 바로직전의 위치가 좌표값 0 으로 간주하여 fNewDistance 값을 설정하여야 합니다.

참 고

□ COMILX_MC_StartMoveTo 함수에 시작된 모션의 목표 위치(절대좌표)값을 오버라이딩하려면 COMILX_MC_OverrideMoveTo 함수를 사용하십시오.

▣ COMILX_MC_OverrideMoveTo

함수 원형

void COMILX_MC_OverrideMoveTo (HANDLE hDevice, int nChannel, double fNewPosition)

함수 설명

이 함수는 COMILX_MC_StartMoveTo 함수를 통하여 수행되는 절대좌표 In-position 모션에 대하여 목표 절대좌표값을 수정(오버라이딩, Overriding)하는 함수입니다. 이 함수는 COMILX_MC_StartMoveTo 함수를 사용하여 모션을 수행하고 있는 경우에만 사용 가능한 함수입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fNewPosition* : 새로운 목표 절대좌표값을 지정합니다.

참 고

□ COMILX_MC_StartMove 함수에 시작된 모션의 목표 거리값을 오버라이딩하려면 COMILX_MC_OverrideMove 함수를 사용하십시오.

5.4.6 원점 복귀(Home Return) 함수

이 단원에서는 원점 복귀(Home Return)에 관련된 함수들을 소개합니다. 원점 복귀는 모션제어의 대상이 되는 구조물이 원점 위치로 자동 복귀하도록 하는 작업입니다. 원점 복귀 작업이 완료되면 Command Counter, Feedback Counter, Deviation Counter는 자동으로 0으로 초기화됩니다.

원점 복귀 작업을 수행하기 위해서는 ORG(HOME), EZ 및 EL 신호가 참조되는데 이 신호들의 의미 및 작용은 다음과 같습니다.

☐ ORG (HOME) 신호

ORG 신호는 구조물이 원점에 복귀했는지를 센서로부터 입력받는 신호입니다. 일반적으로는 근접 센서와 같은 센서들을 이용하여 원점 복귀 여부를 감지하게 됩니다. 이 신호는 터미널 보드의 'HOME' 단자를 통하여 COMI-LX50x 보드에 입력되어야 합니다.

☐ EZ 신호

엔코더의 제로 펄스 신호를 의미합니다. 이 신호는 원점 복귀 모드에 따라 ORG 신호 또는 EL 신호와 함께 사용되어 보다 정밀한 원점복귀 작업을 수행할 수 있도록 해줍니다. 이 신호는 터미널 보드의 'EZ+' 단자와 'EZ-' 단자를 통하여 COMI-LX50x 보드에 입력되어야 합니다.

☐ EL 신호

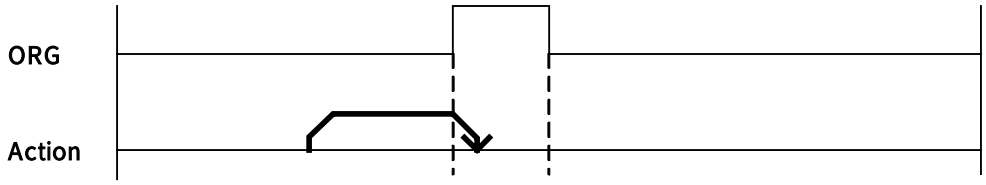
기계적인 리미트(Limit) 신호를 의미합니다. 이 신호는 일반적으로 구조물이 움직일 수 있는 한계점을 감지하기 위해 사용되나 원점 복귀 모드에 따라 ORG 신호의 대용으로도 사용될 수 있습니다. EL 신호는 (+)방향 리미트 신호와 (-)방향 리미트 신호의 두 가지 신호가 있습니다. (+)방향 리미트 신호는 터미널 보드의 '+EL' 단자, 그리고 (-)방향 리미트 신호는 '-EL' 단자를 통하여 COMI-LX50x 보드에 입력되어야 합니다.

원점복귀모드

COMI-LX50x 모션제어보드는 다음과 같이 13 가지의 다양한 원점 복귀 모드를 제공합니다. 원점 복귀 모드는 COMILX_MC_SetHomeConfig()함수를 통하여 설정됩니다. 아래의 그림은 모두 속도모드를 Trapezoidal 모드로 설정한 상태임을 가정하여 그려진 것이며, 만일 Constant 속도 모드로 설정된 경우에는 감속이 없이 즉시 정지하게 됩니다.

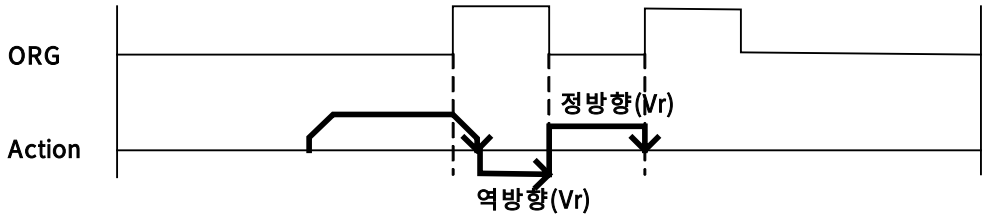
☐ MODE 0 : ORG -> Slow down -> Stop

MODE 0 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간에 모션을 감속 후 정지하고 복귀 작업을 종료합니다.



☐ MODE 1 : ORG -> Slow down -> Go back -> Go forward -> Stop

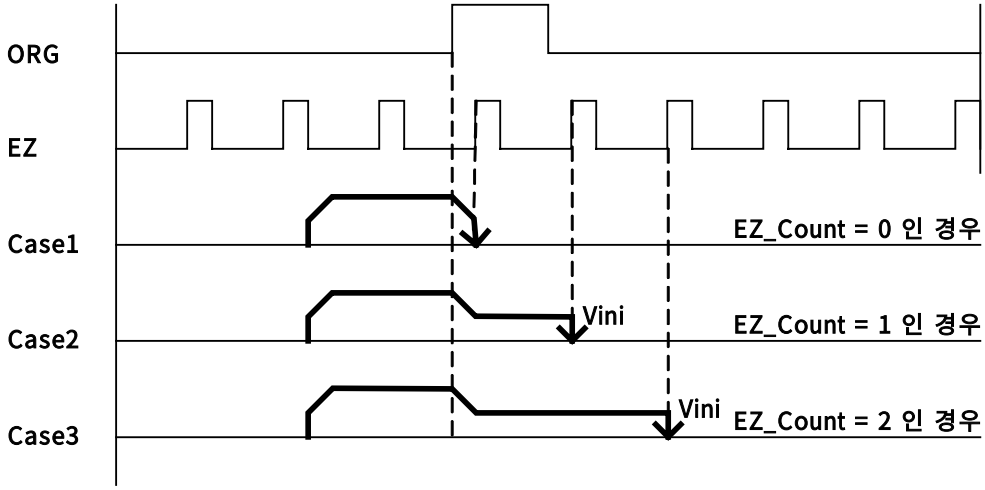
MODE 1 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간에 모션을 감속 후 정지한 후 ORG 신호가 OFF 가 될때까지 Vr(Reverse Speed)의 속도로 역방향 회전을 수행합니다. ORG 신호가 OFF 되는 순간에 다시 Vr 의 속도로 정방향 회전을 수행하다가 ORG 신호가 다시 ON 되는 순간에 복귀작업을 종료합니다.



Vr : Reverse Speed를 의미하며 COMILX_MC_HomeMove 함수 참조

☐ MODE 2 : ORG -> Slow down -> Stop on EZ signal

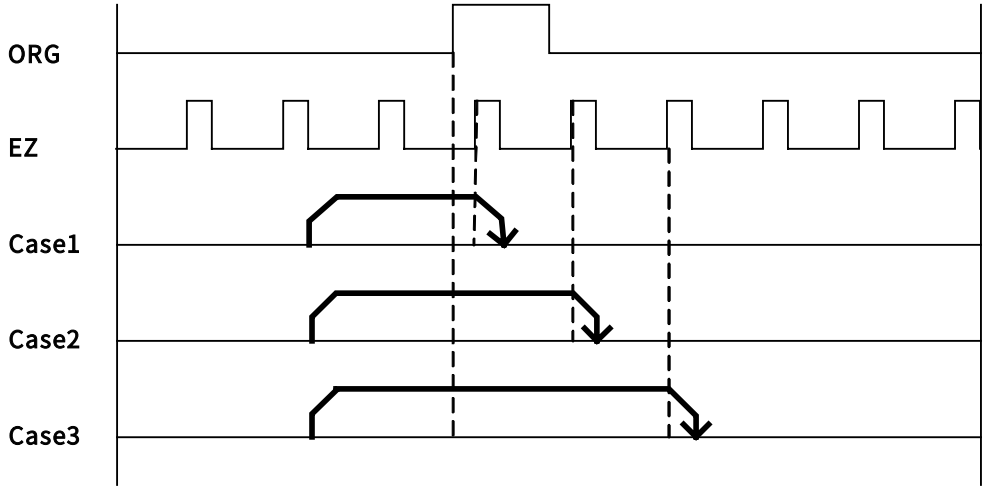
MODE 2 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간에 모션을 감속한 후 초기속도값으로 모션을 진행하다가 EZ 신호에 따라 복귀작업을 종료합니다. COMILX_MC_SetHomeConfgi 함수를 통하여 미리 설정된 EzCount 값에 따라 종료하는 시점은 아래와같이 달라집니다.



Case1 : EzCount = 0 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case2 : EzCount = 1 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case3 : EzCount = 2 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Vini : 초기속도를 의미하며, COMILX_MC_SetSpeed 함수 참조

☐ MODE 3 : ORG -> EZ -> Slow down -> Stop

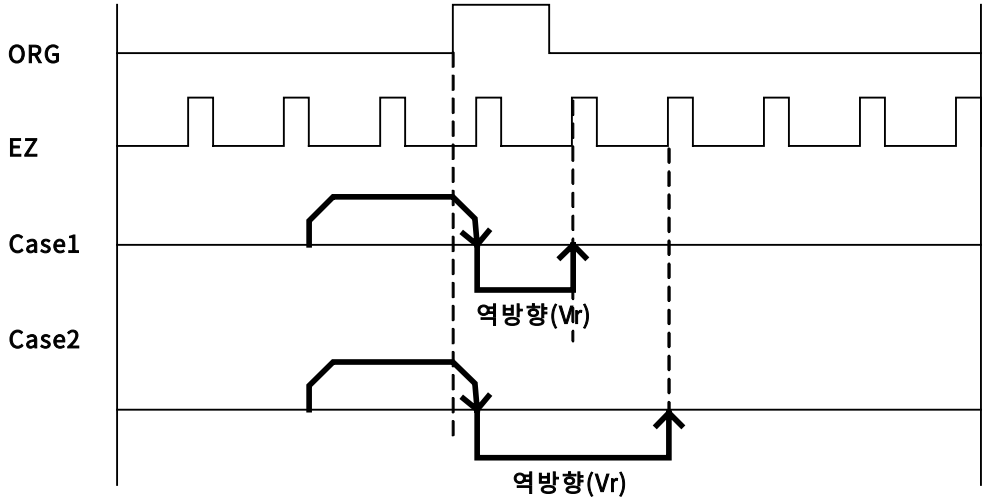
MODE 3 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌고 난 후 발생하는 EZ 신호에 따라 감속 후 정지합니다. COMILX_MC_SetHomeConfgi 함수를 통하여 미리 설정된 EzCount 값에 따라 감속을 시작하는 시점은 아래와같이 달라집니다.



Case1 : EzCount = 0 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case2 : EzCount = 1 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case3 : EzCount = 2 인 경우, COMILX_MC_SetHomeConfig 함수 참조

☐ MODE 4 : ORG -> Slow down -> Go back at Vr -> Stop on EZ signal

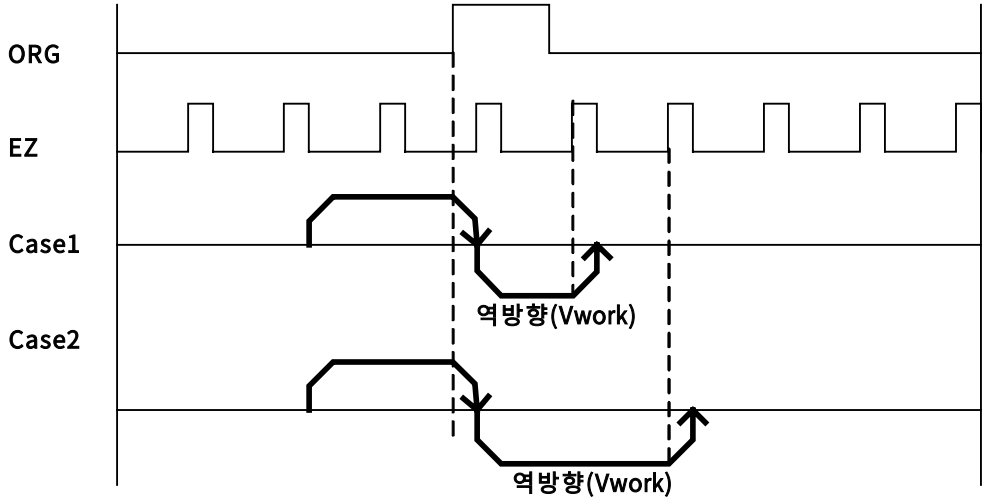
MODE 4 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간 감속 후 정지합니다. 그리고 Vr 의 속도로 역방향 회전한 후 EZ 신호에 따라 복귀 작업을 종료합니다.



Case1 : EzCount = 1 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case2 : EzCount = 2 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Vr : Reverse Speed를 의미하며 COMILX_MC_HomeMove 함수 참조

☐ MODE 5 : ORG -> Slow down -> Go back -> Accelerate to Vwork -> EZ -> Slow down -> Stop

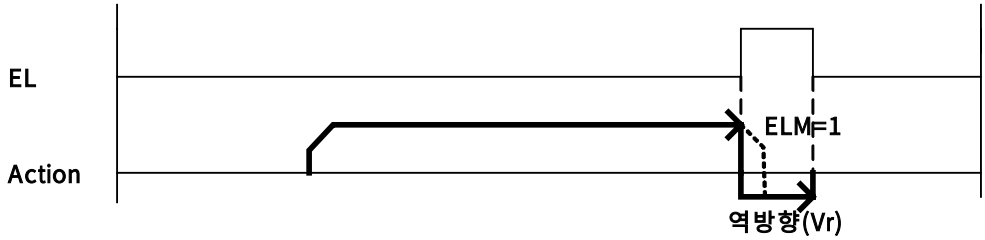
MODE 5 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간 감속 후 정지합니다. 그리고 작업속도까지 가속하여 역방향 회전한 후 EZ 신호에 따라 감속 후 복귀 작업을 종료합니다.



Case1 : EzCount = 1 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case2 : EzCount = 2 인 경우, COMILX_MC_SetHomeConfig 함수 참조

☐ MODE 6 : EL ON -> Stop -> Go back at Vr -> EL OFF -> Stop

MODE 6 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속 후 정지)합니다. 그리고 반대 방향으로 Vr 속도로 회전하다가 EL 신호가 OFF 되는 순간에 복귀작업을 종료합니다. 여기서 ELM=1 은 COMILX_MC_SetMioCfgEL()함수에서 nElMode 를 1 로 설정했음을 의미합니다.

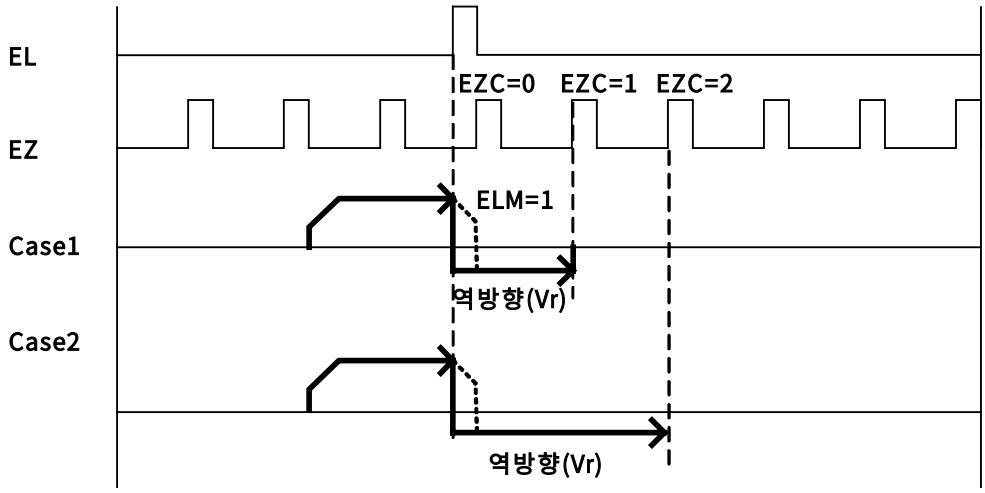


Vr : Reverse Speed를 의미하며 COMILX_MC_HomeMove 함수 참조

☐ MODE 7 : EL ON -> Go back at Vr -> Stop on EZ signal

MODE 7 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속 후 정지)합니다. 그리고 반대 방향으로 Vr 속도로 회전하다가 EZ_Count 에 따라 복귀작업을 종료합니다. 여기서 ELM=1 은 COMILX_MC_SetMioCfgEL()함수에서 nElMode 를

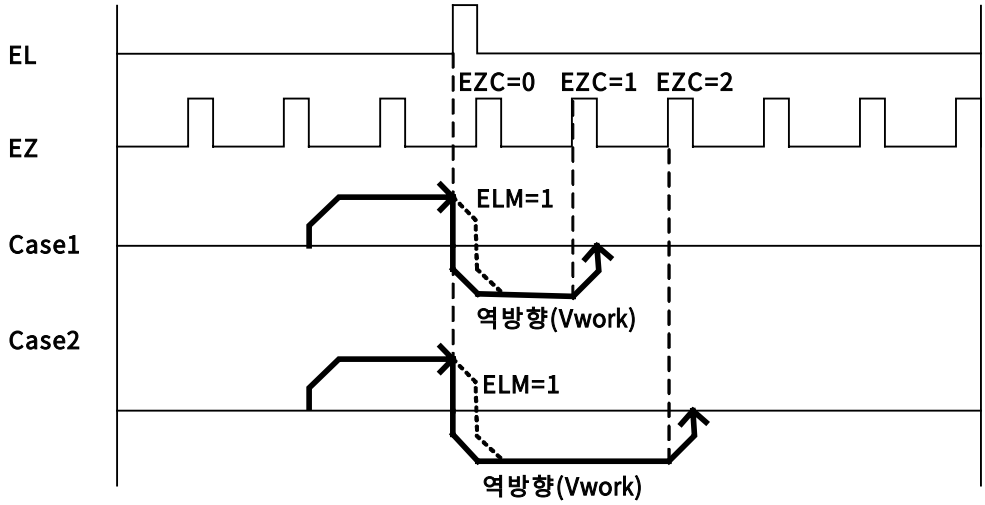
1로 설정했음을 의미합니다.



Case1 : EzCount = 1 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case2 : EzCount = 2 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Vr : Reverse Speed를 의미하며 COMILX_MC_HomeMove 함수 참조

☐ MODE 8 : EL ON -> Accelerate to Vwork -> EZ -> Slow down -> Stop

MODE 8 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속 후 정지)합니다. 그리고 작업속도까지 가속하여 반대 방향으로 회전한 후 EZ 신호에 따라 감속 후 복귀 작업을 종료합니다. 여기서 ELM=1 은 COMILX_MC_SetMioCfgEL()함수에서 nElMode 를 1로 설정했음을 의미합니다.



Case1 : EzCount = 1 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Case2 : EzCount = 2 인 경우, COMILX_MC_SetHomeConfig 함수 참조
Vwork : 작업속도를 의미하며 COMILX_MC_SetSpeed 함수 참조

☐ MODE 9 : EL ON -> Accelerate to Vwork -> EZ -> Slow down -> Stop

MODE 9 에서는 MODE 0 과 동일한 복귀 작업을 수행한다. 그리고, Feedback Counter 가 0 이 되도록 모션을 다시 취한 후에 복귀 작업을 종료한다.

☐ MODE 10 : EL ON -> Accelerate to Vwork -> EZ -> Slow down -> Stop

MODE 10 에서는 MODE 3 과 동일한 복귀 작업을 수행한다. 그리고, Feedback Counter 가 0 이 되도록 모션을 다시 취한 후에 복귀 작업을 종료한다.

☐ MODE 11 : EL ON -> Accelerate to Vwork -> EZ -> Slow down -> Stop

MODE 11 에서는 MODE 5 과 동일한 복귀 작업을 수행한다. 그리고, Feedback Counter 가 0 이 되도록 모션을 다시 취한 후에 복귀 작업을 종료한다.

☐ MODE 12 : EL ON -> Accelerate to Vwork -> EZ -> Slow down -> Stop

MODE 12 에서는 MODE 8 과 동일한 복귀 작업을 수행한다. 그리고 난후 Feedback

Counter 가 0 이 되도록 모션을 다시 취한 후에 복귀 작업을 종료한다.

▣ COMILX_MC_SetHomeConfig

함수 원형

void COMILX_MC_SetHomeConfig (HANDLE hDevice, int nChannel, int nOrgMode, int nOrgLogic, int nEzCount, int nEzLogic, int nErcOut)

함수 설명

이 함수는 원점 복귀 작업을 수행하기 위한 여러가지 환경설정을 수행합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nOrgMode* : 원점 복귀 모드 번호를 설정합니다. COMI-LX50x 모션 제어보드는 13가지(0 ~ 12)의 다양한 복귀 모드를 제공합니다. 각 복귀 모드에 대한 자세한 사항은 앞 페이지를 참조하십시오.

▶ *nOrgLogic* : ORG 신호의 Action Level 을 설정합니다. 즉 ORG 신호의 레벨(Level)이 High 상태일때 ON 인지, Low 상태일때 ON 인지를 설정합니다.

Value	Meaning
0	Low active, 즉 ORG 신호레벨이 Low 일때 Logic 1 이 된다.
1	High active, 즉 ORG 신호레벨이 High 일때 Logic 1 이 된다.

▶ *nEzCount* : 이 값은 ORG 신호 또는 EL 신호가 ON 이 된 후 실제로 복귀 작업을 완료하는데 필요한 EZ Count 값을 0 ~ 15 사이의 값으로 설정합니다. 이 값의 참조 여부는 복귀 모드에 따라서 다릅니다.

▶ *nEzLogic* : EZ 신호의 Action Level 을 설정합니다. 즉 EZ 신호의 레벨(Level)이 High 상태일때 ON 인지, Low 상태일때 ON 인지를 설정합니다.

Value	Meaning
0	Low active, 즉 EZ 신호레벨이 Low 일때 Logic 1 이 된다.
1	High active, 즉 EZ 신호레벨이 High 일때 Logic 1 이 된다.

▶ *nErcOut* : 원점 복귀 작업이 끝나는 시점에서 ERC 펄스를 출력할 것인지를 결정합니다. ERC 신호는 서보모터 드라이버의 Deviation Counter 를 리셋하는 기능을 제

공합니다.

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"

#define X_AXIS        0
#define MODE4         4
#define LOW_ACTIVE    0

void main()
{
   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_SetHomeConfig (hDevice, X_AXIS, MODE4, LOW_ACITVE, 0,
LOW_ACTIVE, 0);

   COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 200, 5000);
   COMILX_MC_SetAccel(hDevice, X_AXIS, 5000, 5000);

   COMILX_MC_HomeMove(hDevice, X_AXIS, 1, 500);
   while(!COMILX_MC_Done (hDevice, X_AXIS))
       ;

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

▣ COMILX_MC_HomeMove

함수 원형

void COMILX_MC_HomeMove (HANDLE hDevice, int nChannel, int nDirection, double fRvsVel)

함수 설명

이 함수는 원점 복귀 작업을 수행하기 위한 여러가지 환경설정을 수행합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nOrgMode* : 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.

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"

#define X_AXIS       0
#define MODE4        4
#define LOW_ACTIVE   0
```

```
void main()
{
   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_SetHomeConfig (hDevice, X_AXIS, MODE4, LOW_ACITVE, 0,
LOW_ACTIVE, 0);

   COMILX_MC_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 200, 5000);
   COMILX_MC_SetAccel(hDevice, X_AXIS, 5000, 5000);

   COMILX_MC_HomeMove(hDevice, X_AXIS, 1, 500);
   while(!COMILX_MC_Done (hDevice, X_AXIS))
       ;

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

5.4.7 Manual Pulser 모드 모션 제어 함수

이 단원에서는 Manual Pulser 모드 모션에 관련된 함수들을 소개합니다. Manual Pulser 모드는 로터리 엔코더(Rotary Encoder)와 같은 장치를 이용하여 수동으로 모션을 제어하는 모드를 의미합니다. COMI-LX50x 모션제어 보드는 PA/PB 단자를 통하여 Plus 펄스와 Minus 펄스(CW/CCW) 또는 90° 위상차를 갖는 AB Phase 펄스를 입력받아 수동으로 모션을 제어할 수 있습니다. PA/PB 단자에 입력되는 신호의 형태는 Pulser 입력모드 설정에 따라 달라지며 이는 COMILX_MC_SetPulserInputMode()함수에 의해 설정됩니다.

Manual Pulser 모드 모션에서의 속도는 입력 펄스의 주파수에 의해 결정됩니다. 따라서 속도모드나 가/감속 등은 설정할 필요가 없습니다. 그러나 Manual Pulser 입력신호의 최대 주파수는 COMILX_MC_SetSpeed()함수에 의해 설정되는 작업 속도에 의하여 제한됩니다. 따라서 Manual Pulser 모션을 수행하기 전에 COMILX_MC_SetSpeed() 함수를 이용하여 작업속도를 적정한 값으로 설정하여야 합니다.

Manual Pulser 모드 모션은 Velocity Motion 은 물론 상대좌표/절대좌표 In-Position 모션에도 적용가능합니다. 그러나 Coordinated Motion 에는 적용할 수 없습니다.

Manual Pulser 모드 모션에 관련된 함수들은 다음과 같습니다.

void COMILX_MC_SetPE(HANDLE hDevice, int nChannel, BOOL bEnable) Manual Puser 모드와 External Switch Operation 기능을 Enable/Disable 시킵니다.
void COMILX_MC_SetPulserInputMode (HANDLE hDevice, int nChannel, int nInputMode, BOOL bInverse) Pulser 입력신호 대한 환경을 설정합니다.
void COMILX_MC_PulserHomeMove (HANDLE hDevice, int nChannel, int nHomeType) Pulser Input 에 의한 원점 복귀 작업을 수행합니다.
void COMILX_MC_StartPulserVMove (HANDLE hDevice, int nChannel) Stop 함수가 호출될 때까지 Pulser 신호 입력에 맞추어 계속적인 모션을 수행합니다.
void COMILX_MC_StartPulserMove (HANDLE hDevice, int nChannel, double fDistance) Pulser 신호 입력에 맞추어 지정한 거리(상대좌표)만큼 이동을 수행합니다.
void COMILX_MC_PulserMove (HANDLE hDevice, int nChannel, double fDistance) Pulser 신호 입력에 맞추어 지정한 거리(상대좌표)만큼 이동을 수행합니다.

void COMILX_MC_StartPulserMoveTo (HANDLE hDevice, int nChannel, double fPosition)
Pulser 신호 입력에 맞추어 지정한 위치(절대좌표)로 이동을 수행합니다.
void COMILX_MC_PulserMoveTo (HANDLE hDevice, int nChannel, double fPosition)
Pulser 신호 입력에 맞추어 지정한 위치(절대좌표)로 이동을 수행합니다.

▣ COMILX_MC_SetPE

함수 원형

void COMILX_MC_SetPE(HANDLE hDevice, int nChannel, BOOL bEnable)

함수 설명

이 함수는 Manual Pulser 모드나 External Switch Operation 모드를 Enable/Disable 시킵니다. Manual Pulser 모드를 사용하기 위해서는 먼저 이 함수를 이용하여 Manaul Pulser 모드를 Enable 시켜야 합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *bEnable* : Manual Pulser 모드와 External Switch Operation 모드를 Enable/Disable 시킵니다.

Value	Meaning
0	Manual Pulser 모드와 External Switch Operation 모드 Disable
1	Manual Pulser 모드와 External Switch Operation 모드 Enable

▣ COMILX_MC_SetPulserInputMode

함수 원형

void COMILX_MC_SetPulserInputMode (HANDLE hDevice, int nChannel, int nInputMode, BOOL bInverse)

함수 설명

이 함수는 Pulser 입력 신호에 대한 환경을 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nInputMode* : PA 와 PB 입력 단자를 통하여 입력되는 Pulser 입력 신호의 입력모드를 설정합니다. 설정가능한 값은 다음과 같습니다.

Value	Meaning
0	1X A/B (1 채배 Phase type 입력 모드)
1	2X A/B (2 채배 Phase type 입력 모드)
2	4X A/B (4 채배 Phase type 입력 모드)
3	CW/CCW (PA - Plus direction move, PB - Minus direction move)

▶ *bInverse* : Pulser 입력 신호에 의해 결정되는 방향(Direction)을 모션에 반대로 적용할 지를 결정합니다. 설정가능한 값은 다음과 같습니다.

Value	Meaning
0	Pulser 입력 신호가 나타내는 방향과 모션의 방향 일치
1	Pulser 입력 신호가 나타내는 방향과 모션의 방향을 반대로 적용

▣ COMILX_MC_PulserHomeMove

함수 원형

void COMILX_MC_PulserHomeMove (HANDLE hDevice, int nChannel, int nHomeType)

함수 설명

이 함수는 Pulser Input 에 의한 원점 복귀 작업을 수행합니다. 원점 복귀 모드(Home Type)에 따라 원점 복귀가 완료되거나 COMILX_MC_Stop()함수 또는 COMILX_MC_EmgStop()함수가 호출되면 모션을 종료합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nHomeType* : Pulser Input 에 의해 원점 복귀를 수행하는 모드를 설정합니다. 이 값은 다음 중 하나의 값이어야 합니다.

Value	Meaning
0	Command 카운터가 0 이 될때까지 원점복귀를 종료합니다.
1	ORG 신호가 ON 이 되면 원점복귀를 종료합니다.

▣ COMILX_MC_StartPulserVMove

함수 원형

void COMILX_MC_StartPulserVMove (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 Stop 함수가 호출될 때까지 Pulser 신호 입력에 맞추어 계속적인 모션을 수행합니다. 모션의 속도는 Pulser 신호의 주파수에 따라 결정됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

참 고

□ Manual Pulser 모드 모션에서의 속도는 입력 펄스의 주파수에 의해 결정됩니다. 따라서 속도모드나 가/감속 등은 설정할 필요가 없습니다. 그러나 Manual Pulser 입력신호의 최대 주파수는 COMILX_MC_SetSpeed()함수에 의해 설정되는 작업 속도에 의하여 제한됩니다. 따라서 Manual Pulser 모션을 수행하기 전에 COMILX_MC_SetSpeed()함수를 이용하여 작업속도를 적정한 값으로 설정하여야 합니다. 만일 COMILX_MC_SetUnitSpeed()함수를 이용하여 논리속도를 1PPS 단위가 아닌 다른 값으로 설정하였다면 Pulser 입력속도의 단위는 PPS 단위임에 유의하여야 합니다.

□ Manual Pulser 모드 모션을 중지하기 위해서는 COMILX_MC_Stop() 함수대신 COMILX_MC_EmgStop()함수를 사용하여야 합니다.

예 제

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

#define X_AXIS       0
#define PHASE_1X  0 // Pulser input mode => 1X A/B (1 채배 Phase
type 입력 모드)
void main()
{
   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);
   COMILX_MC_SetPE(hDevice, X_AXIS, TRUE); // Enable Pulse Mode
   // Pulser 입력신호의 최대 주파수 제한 설정(650000PPS) //
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 650000);
   COMILX_MC_SetPulserInputMode   (hDevice,   X_AXIS,   PHASE_1X,
FALSE);
   // Manual Pulser 모드에서 Velocity Move 모션을 시작한다. //
   COMILX_MC_StartPulserVMove (hDevice, X_AXIS);
   // 사용자가 키보드 입력을 하기전까지 PA/PB 입력신호에 따라 모션이 수행된
다. //
   while(!kbhit())
       ;
   COMILX_MC_EmgStop(hDevice, X_AXIS);
   COMILX_UnloadDevice(hDevice);
   COMILX_UnloadDll();
}
```

▣ COMILX_MC_StartPulserMove

함수 원형

void COMILX_MC_StartPulserMove (HANDLE hDevice, int nChannel, double fDistance)

함수 설명

이 함수는 Pulser 신호 입력에 맞추어 지정한 거리(상대좌표)만큼 이동을 수행합니다. 이 함수는 모션을 시작시킨 후에 바로 Return 합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fDistance* : 이동할 거리를 지정합니다. 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리 값 1은 1 Pulse 출력을 의미합니다.

참 고

□ Manual Pulser 모드 모션에서의 속도는 입력 펄스의 주파수에 의해 결정됩니다. 따라서 속도모드나 가/감속 등은 설정할 필요가 없습니다. 그러나 Manual Pulser 입력신호의 최대 주파수는 COMILX_MC_SetSpeed()함수에 의해 설정되는 작업 속도에 의하여 제한됩니다. 따라서 Manual Pulser 모션을 수행하기 전에 COMILX_MC_SetSpeed()함수를 이용하여 작업속도를 적정한 값으로 설정하여야 합니다.

□ Manual Pulser 모드 모션을 취소하기 위해서는 COMILX_MC_Stop() 함수대신 COMILX_MC_EmgStop()함수를 사용하여야 합니다.

예 제

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

#define X_AXIS       0
#define PHASE_1X  0 // Pulser input mode => 1X A/B (1 채배 Phase type 입력 모드)
void main()
{
   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);
    COMILX_MC_SetPE(hDevice, X_AXIS, TRUE); // Enable Pulse Mode
    // Pulser 입력신호의 최대 주파수 제한 설정(650000PPS) //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 650000);
    COMILX_MC_SetPulserInputMode   (hDevice,   X_AXIS,   PHASE_1X,
FALSE);
    COMILX_MC_StartPulserMove(hDevice, X_AXIS, 1000);
    while(!COMILX_MC_Done())
        ;
    COMILX_UnloadDevice(hDevice);
    COMILX_UnloadDll();
}
```

▣ COMILX_MC_PulserMove

함수 원형

void COMILX_MC_PulserMove (HANDLE hDevice, int nChannel, double fDistance)

함수 설명

이 함수는 Pulser 신호 입력에 맞추어 지정한 거리(상대좌표)만큼 이동을 수행합니다. 이함수는 모션이 완료될 때까지 Return 되지 않고 루프를 돌게 됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fDistance* : 이동할 거리를 지정합니다. 거리에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리 값 1은 1 Pulse 출력을 의미합니다.

참 고

□ 이 함수는 모션이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ Manual Pulser 모드 모션에서의 속도는 입력 펄스의 주파수에 의해 결정됩니다. 따라서 속도모드나 가/감속 등은 설정할 필요가 없습니다. 그러나 Manual Pulser 입력신호의 최대 주파수는 COMILX_MC_SetSpeed()함수에 의해 설정되는 작업 속도에 의하여 제한됩니다. 따라서 Manual Pulser 모션을 수행하기 전에 COMILX_MC_SetSpeed()함수를 이용하여 작업속도를 적정한 값으로 설정하여야 합니다.

예 제

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

#define X_AXIS       0
#define PHASE_1X  0 // Pulser input mode => 1X A/B (1 채배 Phase type 입력 모드)
void main()
{
```

```
    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);
    // Pulser 입력신호의 최대 주파수 제한 설정(650000PPS) //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 650000);
    COMILX_MC_SetPE(hDevice, X_AXIS, TRUE); // Enable Pulse Mode
    COMILX_MC_SetPulserInputMode   (hDevice,   X_AXIS,   PHASE_1X,
FALSE);
    COMILX_MC_PulserMove(hDevice, X_AXIS, 1000);

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

▣ COMILX_MC_StartPulserMoveTo

함수 원형

void COMILX_MC_StartPulserMoveTo (HANDLE hDevice, int nChannel, double fPosition)

함수 설명

이 함수는 Pulser 신호 입력에 맞추어 지정한 위치(절대좌표)로 이동을 수행합니다. 단 이 함수는 모션을 시작시킨 후에 바로 Return 합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fPosition* : 이동할 목표 위치(절대좌표값)를 지정합니다. 위치에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리값 1은 1 Pulse 출력을 의미합니다.

참 고

□ Manual Pulser 모드 모션에서의 속도는 입력 펄스의 주파수에 의해 결정됩니다. 따라서 속도모드나 가/감속 등은 설정할 필요가 없습니다. 그러나 Manual Pulser 입력신호의 최대 주파수는 COMILX_MC_SetSpeed()함수에 의해 설정되는 작업 속도에 의하여 제한됩니다. 따라서 Manual Pulser 모션을 수행하기 전에 COMILX_MC_SetSpeed()함수를 이용하여 작업속도를 적정한 값으로 설정하여야 합니다.

□ Manual Pulser 모드 모션을 취소하기 위해서는 COMILX_MC_Stop() 함수대신 COMILX_MC_EmgStop()함수를 사용하여야 합니다.

예 제

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

#define X_AXIS       0
#define PHASE_1X  0 // Pulser input mode => 1X A/B (1 채배 Phase type 입력 모드)
void main()
{
   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);
    // Pulser 입력신호의 최대 주파수 제한 설정(650000PPS) //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 650000);
    COMILX_MC_SetPE(hDevice, X_AXIS, TRUE); // Enable Pulse Mode
    COMILX_MC_SetPulserInputMode   (hDevice,   X_AXIS,   PHASE_1X,
FALSE);
    COMILX_MC_StartPulserMoveTo(hDevice, X_AXIS, 1000);
    while(!COMILX_MC_Done())
        ;
    COMILX_UnloadDevice(hDevice);
    COMILX_UnloadDll();
}
```

▣ COMILX_MC_PulserMoveTo

함수 원형

void COMILX_MC_PulserMoveTo (HANDLE hDevice, int nChannel, double fPosition)

함수 설명

이 함수는 Pulser 신호 입력에 맞추어 지정한 위치(절대좌표)로 이동을 수행합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fPosition* : 이동할 목표 위치(절대좌표값)를 지정합니다. 위치에 대한 단위는 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다. COMILX_MC_SetUnitDistance 함수로 거리의 단위를 변경하지 않았다면 거리의 단위는 Pulse 수가 됩니다. 즉, 거리값 1은 1 Pulse 출력을 의미합니다.

참 고

□ 이 함수는 모션이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함수를 수행하기 이전에 COMILX_MC_SetBlockingMode 함수를 사용하여 Blocking 이 일어나지 않도록 설정하여야 합니다.

□ Manual Pulser 모드 모션에서의 속도는 입력 펄스의 주파수에 의해 결정됩니다. 따라서 속도모드나 가/감속 등은 설정할 필요가 없습니다. 그러나 Manual Pulser 입력신호의 최대 주파수는 COMILX_MC_SetSpeed()함수에 의해 설정되는 작업 속도에 의하여 제한됩니다. 따라서 Manual Pulser 모션을 수행하기 전에 COMILX_MC_SetSpeed()함수를 이용하여 작업속도를 적정한 값으로 설정하여야 합니다.

예 제

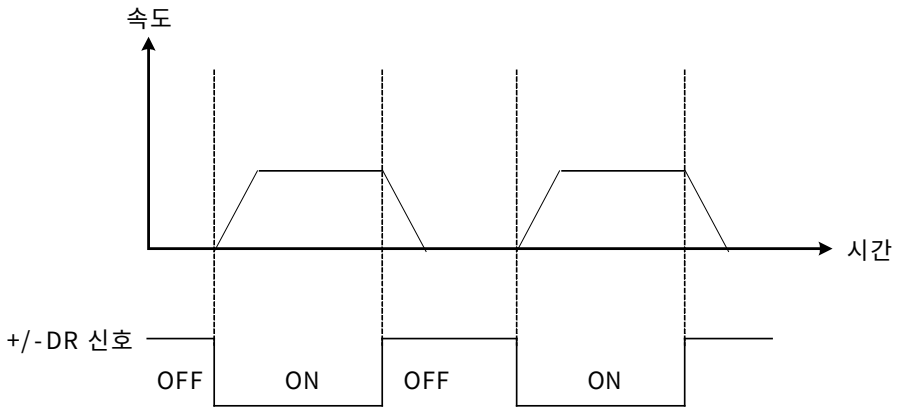
```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#define X_AXIS       0
#define PHASE_1X  0 // Pulser input mode => 1X A/B (1 채배 Phase
type 입력 모드)
void main()
{
   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);
    // Pulser 입력신호의 최대 주파수 제한 설정(650000PPS) //
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 650000);
    COMILX_MC_SetPE(hDevice, X_AXIS, TRUE); // Enable Pulse Mode
    COMILX_MC_SetPulserInputMode   (hDevice,   X_AXIS,   PHASE_1X,
FALSE);
    COMILX_MC_PulserMoveTo(hDevice, X_AXIS, 1000);

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

5.4.8 External Switch Operation 모드 모션 제어 함수

이 단원에서는 External Switch Operation 모드에 관련된 함수들을 소개합니다. 여기서 External Switch 는 +DR 과 -DR 신호를 의미합니다. External Switch Operation 모드는 +DR 또는 -DR 신호가 ON 상태인 경우에만 모션을 구동하고 OFF 인 상태에서는 모션을 정지하는 기능을 말합니다. 이 때의 속도 패턴은 Single Axis Motion 과 같이 COMILX_MC_SetSpeedMode(), COMILX_MC_SetSpeed(), COMILX_MC_SetAccel()의 함수에 의해 설정된 속도 패턴을 사용합니다. +DR 신호는 정방향으로 모션을 구동시키고 -DR 신호는 역방향으로 모션을 구동시킵니다.

+DR 또는 -DR 신호와 모션 구동의 관계는 다음 그림과 같습니다.



[그림 5-19] External Switch Operation

External Switch Operation 모드 모션에 관련된 함수들은 다음과 같습니다.

void COMILX_MC_SetPE(HANDLE hDevice, int nChannel, BOOL bEnable) Manual Puser 모드와 External Switch Operation 기능을 Enable/Disable 시킵니다.
void COMILX_MC_StartVMoveEx(HANDLE hDevice, int nChannel) External switch operation 모드하에 Velocity-Move 를 시작합니다.
void COMILX_MC_StartMoveEx(HANDLE hDevice, int nChannel, double fDistance) External switch operation 모드하에 Relative In-Position(상대좌표 위치결정) 모션을 시작합니다.

▣ COMILX_MC_SetPE

함수 원형

void COMILX_MC_SetPE(HANDLE hDevice, int nChannel, BOOL bEnable)

함수 설명

이 함수는 Manual Pulser 모드나 External Switch Operation 모드를 Enable/Disable 시킵니다. Manual Pulser 모드를 사용하기 위해서는 먼저 이 함수를 이용하여 Manaul Pulser 모드를 Enable 시켜야 합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *bEnable* : Manual Pulser 모드와 External Switch Operation 모드를 Enable/Disable 시킵니다.

Value	Meaning
0	Manual Pulser 모드와 External Switch Operation 모드 Disable
1	Manual Pulser 모드와 External Switch Operation 모드 Enable

▣ COMILX_MC_StartVMoveEx

함수 원형

void COMILX_MC_StartVMoveEx (HANDLE hDevice, int nChannel)

함수 설명

+/-DR 신호가 ON 상태가 되면 Velocity Move 모션을 수행합니다. +/-DR 신호가 OFF 가 되면 모션을 정지합니다. 이 때의 구동 방향은 +DR 신호와 -DR 신호에 의해 결정됩니다.

매개 변수

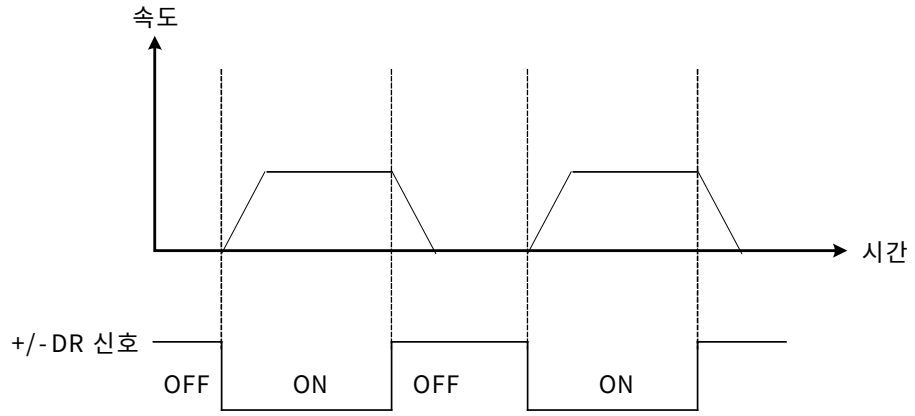
▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

참 고

□ 속도 모드를 Constant Speed Mode 로 지정한 경우에는 가속 구간이 없이 작업속도로 모션을 시작합니다.

□ +DR 또는 -DR 신호와 모션 구동의 관계는 다음 그림과 같습니다.



사용예

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

#define X_AXIS    0
void main()
```

```
{
   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_SetSpeedMode(hDevice, X_AXIS, 1);
   COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
   COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);

   COMILX_MC_SetPE(hDevice, X_AXIS, TRUE); // Enable Pulse Mode

   // External Switch 모드 Velocity Move 시작 //
   COMILX_MC_StartVMoveEx(hDevice, X_AXIS);
   // 이후부터는 모션의 구동은 +/- DR 신호의 상태에 따라 제어된다. //
   // Stop 명령(키보드)이 있을 때 까지 Velocity Move 지속 //
   while(!kbhit())
       ;
   // 감속 후 정지 //
   COMILX_MC_Stop(hDevice, X_AXIS);

   COMILX_MC_SetPE(hDevice, X_AXIS, FALSE); // Disable Pulse Mode

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

▣ COMILX_MC_StartMoveEx

함수 원형

void COMILX_MC_StartMoveEx (HANDLE hDevice, int nChannel, double fDistance)

함수 설명

External switch operation 모드하에 Relative In-Position(상대좌표 위치결정) 모션을 시작합니다. +/-DR 신호가 ON 상태일 때에만 모터를 구동합니다. +DR 신호가 ON 이 되면 정방향으로 회전하고 -DR 신호가 ON 이 되면 역방향으로 회전합니다.

지정한 거리만큼 이동이 완료되면 +/-DR 신호와 관계없이 모션을 종료합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

참 고

□ 속도 모드를 Constant Speed Mode 로 지정한 경우에는 가속 구간이 없이 작업속도로 모션을 시작합니다.

□ +DR 또는 -DR 신호와 모션 구동의 관계는 COMILX_MC_StartVMoveEx() 때와 같습니다. 단, 지정한 거리만큼 이동이 완료되면 +/-DR 신호가 ON 상태이어도 모션을 정지합니다.

예 제

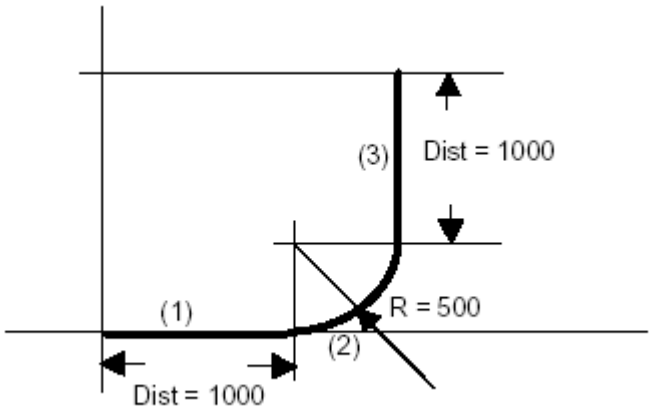
```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#define X_AXIS    0
void main()
{
   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_SetSpeedMode(hDevice, X_AXIS, 1);
```

```
    COMILX_MC_SetSpeed(hDevice, X_AXIS, 0, 10000);
    COMILX_MC_SetAccel(hDevice, X_AXIS, 20000, 20000);

    COMILX_MC_SetPE(hDevice, X_AXIS, TRUE); // Enable Pulse Mode

    // External Switch 모드 상대좌표 모션 시작 //
    COMILX_MC_StartMoveEx(hDevice, X_AXIS, 50000);
    // 이후부터는 모션의 구동은 +/- DR 신호의 상태에 따라 제어된다. //
    // Stop 명령(키보드)이 있을 때 까지 Velocity Move 지속 //
    while(!kbhit()){
        if(COMILX_MC_Done(hDevice, X_AXIS))
            break;
    }
    // 감속 후 정지 //
    COMILX_MC_Stop(hDevice, X_AXIS);

    COMILX_MC_SetPE(hDevice, X_AXIS, FALSE); // Disable Pulse Mode

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

5.4.9 리스트 모션(Listed Motion) 함수

이 단원에서는 리스트 모션(Listed Motion)제어에 관련된 함수들을 소개합니다. 리스트 모션은 수행해야할 여러 단계의 작업을 리스트로 등록시킨 후에 일괄적으로 처리하는 기능을 말합니다. 리스트 모션의 장점은 하나의 작업과 그 다음 작업간에 지연시간(Delay)이 없이 연속적인 작업을 수행할 수 있도록 한다는 것입니다. 또한 리스트 모션을 사용하면 Move 나 MoveTo 함수와 같은 In-Position 함수를 사용할 때에도 작업 속도의 연속성을 확보할 수 있습니다(적용예 2 참조).

▣ 리스트 모션의 적용 예 1



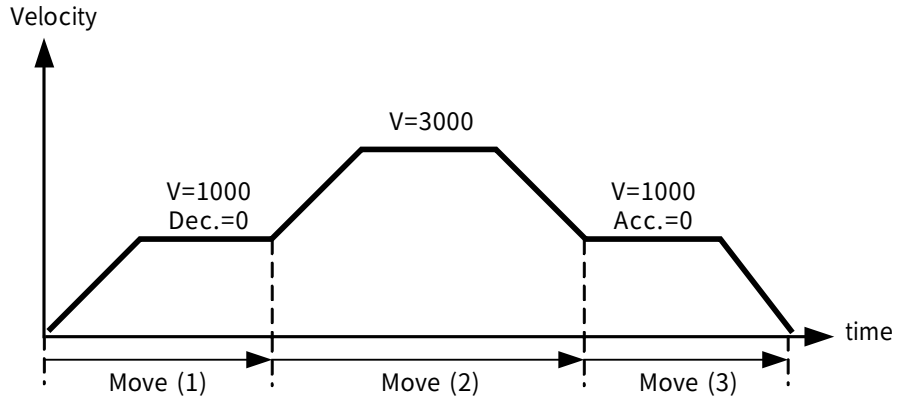
[그림 5-20] 3 회의 Coordinated Motion 으로 이루어지는 작업의 예

[그림 5-20]과 같이 3 회의 Coordinated Motion 을 연속적으로 수행하고자할 때 다음과 같이 리스트 모션을 적용하면 각 작업간의 Delay 가 최소화되어 연속적인 작업을 수행할 수 있습니다.

```
double fDistList[2];
COMILX_MC_BeginList(hDevice); // 모션 리스트 등록 시작 //
COMILX_MC_MapAxes(hDevice, 0, 0x3); // 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 //
COMILX_MC_StartListMotion(hDevice); // 리스트 모션 수행 //
// 리스트 모션이 모두 완료될 때까지 기다림 //
while(!COMILX_MC_CheckListMotionDone(hDevice))
    ;
```

▣ 리스트 모션의 적용 예 2



작 업	초기속도	작업속도	Acceleration	Deceleration
Move (1)	0	1000	2000	0
Move (2)	1000	3000	2000	2000
Move (3)	0	1000	0	2000

[그림 5-21] 속도의 연속성을 가지는 연속적인 In-Position 모션

리스트 모션을 이용하면 [그림 5-21]과 같이 In-Position 의 경우에도 작업 속도의 연속성을 확보할 수 있습니다. 다음의 코드는 [그림 5-21]의 작업을 리스트 모션을 적용하여 구현하는 예입니다.

```
COMILX_MC_BeginList(hDevice); // 모션 리스트 등록 시작 //
// Set Trapezoidal Speed Mode //
COMILX_MC_SetSpeedMode(hDevice, 0, 1);
// Move (1) //
```

```
COMILX_MC_SetSpeed(hDevice, 0, 0, 1000);
COMILX_MC_SetAccel(hDevice, 0, 2000, 0);
COMILX_MC_MoveTo(hDevice, 0, 3000);
// Move (2) //
COMILX_MC_SetSpeed(hDevice, 0, 1000, 3000);
COMILX_MC_SetAccel(hDevice, 0, 2000, 2000);
COMILX_MC_MoveTo(hDevice, 0, 8000);
// Move (3) //
COMILX_MC_SetSpeed(hDevice, 0, 0, 1000);
COMILX_MC_SetAccel(hDevice, 0, 0, 2000);
COMILX_MC_MoveTo(hDevice, 0, 11000);
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))
    ;
```

리스트 모션에 관련된 함수들은 다음과 같습니다.

void COMILX_MC_BeginList (HANDLE hDevice) List Motion에서 수행할 작업들의 등록을 시작합니다.
void COMILX_MC_EndList (HANDLE hDevice) List Motion에서 수행할 작업들의 등록을 종료합니다.
void COMILX_MC_StartListMotion (HANDLE hDevice) List Motion으로 등록된 작업들에 대하여 실제로 모션을 시작합니다.
void COMILX_MC_AbortListMotion (HANDLE hDevice) 리스트 모션을 수행하고 있는 중에 리스트 모션을 중지합니다.
void COMILX_MC_CheckListMotionDone (HANDLE hDevice) 리스트 모션 수행이 완료됐는지를 알려줍니다.

▣ COMILX_MC_BeginList

함수 원형

void COMILX_MC_BeginList (HANDLE hDevice)

함수 설명

이 함수는 List Motion 에서 수행할 작업들의 등록을 시작합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▣ COMILX_MC_EndList

함수 원형

```
void COMILX_MC_EndList (HANDLE hDevice)
```

함수 설명

이 함수는 List Motion 에서 수행할 작업들의 등록을 종료합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▣ COMILX_MC_StartListMotion

함수 원형

void COMILX_MC_StartListMotion (HANDLE hDevice)

함수 설명

이 함수는 List Motion으로 등록된 작업들에 대하여 실제로 모션을 시작합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▣ COMILX_MC_AbortListMotion

함수 원형

```
void COMILX_MC_AbortListMotion (HANDLE hDevice)
```

함수 설명

이 함수는 COMILX_MC_StartListMotion()을 이용하여 리스트 모션을 수행하고 있는 중에 리스트 모션을 중지하고자 할 때 사용합니다. 리스트 모션을 시작하면 등록된 작업들이 모두 수행되면 자동으로 중지됩니다. COMILX_MC_AbortListMotion() 함수는 리스트 모션이 진행되고 있는 중에 리스트 모션을 중지해야할 때 사용합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▣ COMILX_MC_CheckListMotionDone

함수 원형

void COMILX_MC_CheckListMotionDone (HANDLE hDevice)

함수 설명

이 함수는 리스트 모션 수행이 완료됐는지를 알려주는 함수입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

반환값

Value	Meaning
0	리스트 모션이 완료되지 않음
1	등록된 모든 리스트 모션이 완료됨

5.4.10 상태 감시 및 제어 함수

이 단원에서는 상태 감시 및 제어 함수에 관하여 설명합니다. 상태 감시 및 제어 함수들은 모션의 상태를 감시하는데 필요한 함수들을 그룹화한 것입니다. 상태 감시에는 모션의 속도, 위치 등을 체크하는 것을 포함하며 이외에도 현재 모션이 진행되고 있는지를 체크하고, 현재 진행되고 있는 모션의 단계 및 각 I/O 상태들을 체크하는 기능도 포함합니다.

상태 감시 및 제어 함수에 관련된 함수들은 다음과 같습니다.

double COMILX_MC_GetCurSpeed (HANDLE hDevice, int nChannel) 현재 출력되고 있는 Command 속도를 반환합니다.
void COMILX_MC_EnableActSpdChk (HANDLE hDevice, long dwInterval) 실제 모션의 속도를 체크하는 기능을 Enable 시킵니다.
void COMILX_MC_DisableActSpdChk (HANDLE hDevice) 실제 모션의 속도(Actual Speed)를 체크하는 기능을 Disable 시킵니다.
double COMILX_MC_GetActualSpeed (HANDLE hDevice, int nChannel) EA/EB 단자로 입력되는 Feedback 펄스를 계측하여 실제 모션의 속도를 반환합니다.
double COMILX_MC_GetPosition_A (HANDLE hDevice, int nChannel) 현재의 실제 위치(Actual Position)를 논리 단위로 반환합니다.
void COMILX_MC_SetPosition_A (HANDLE hDevice, int nChannel, double fPos) 현재의 실제 위치(Actual Position)값을 설정하며 단위는 논리단위입니다.
double COMILX_MC_GetPosition_C (HANDLE hDevice, int nChannel) 현재의 명령 위치(Command Position)를 논리 단위로 반환합니다.
void COMILX_MC_SetPosition_C (HANDLE hDevice, int nChannel, double fPos) 명령 위치(Command Position)값을 설정하며 단위는 논리단위입니다.
long COMILX_MC_GetCount_A (HANDLE hDevice, int nChannel) 현재의 실제 위치(Actual Position) 카운트값을 반환합니다.
void COMILX_MC_SetCount_A (HANDLE hDevice, int nChannel, long nCount) 현재의 실제 위치(Actual Position) 카운트값을 설정합니다.

long COMILX_MC_GetCount_C (HANDLE hDevice, int nChannel) 현재의 명령 위치(Command Position) 카운트값을 반환합니다.
void COMILX_MC_SetCount_C (HANDLE hDevice, int nChannel, long nCount) 현재의 명령 위치(Command Position) 카운트값을 설정합니다.
long COMILX_MC_GetCount_D (HANDLE hDevice, int nChannel) Deviation Counter 의 값을 읽어서 반환합니다.
void COMILX_MC_SetCount_D (HANDLE hDevice, int nChannel, long nCount) Deviation Counter 의 값을 새로이 설정합니다.
long COMILX_MC_GetCount_G (HANDLE hDevice, int nChannel) General Counter 의 값을 읽어서 반환합니다.
void COMILX_MC_SetCount_G (HANDLE hDevice, int nChannel, long nCount) General Counter 의 값을 새로이 설정합니다.
long COMILX_MC_GetCount_R (HANDLE hDevice, int nChannel) Remained Counter 의 값을 읽어서 반환합니다.
void COMILX_MC_SetCount_R (HANDLE hDevice, int nChannel, long nCount) Remained Counter 의 값을 새로이 설정합니다.
int COMILX_MC_GetMotionStatus (HANDLE hDevice, int nChannel) 현재 모션의 동작 상태를 반환합니다.
int COMILX_MC_GetMioStatus(HANDLE hDevice, int nChannel) 현재 모션과 관련된 여러가지 I/O 상태를 반환합니다.

▣ COMILX_MC_GetCurSpeed

함수 원형

double COMILX_MC_GetCurSpeed (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 현재 출력되고 있는 Command 속도를 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

지정한 축의 Command 속도를 반환합니다. 이 값은 모터의 실제속도가 아니며 스텝모터 또는 서보모터 드라이버에 전달되는 Command 속도입니다. 또한 이 값의 단위는 단위는 COMILX_MC_SetUnitSpeed() 함수에 의하여 결정되며 기본적으로는 Pulses/sec 입니다.

▣ COMILX_MC_EnableActSpdChk

함수 원형

void COMILX_MC_EnableActSpdChk (HANDLE hDevice, long dwInterval)

함수 설명

이 함수는 실제 모션의 속도를 체크하는 기능을 Enable 시킵니다. 모션의 실제속도는 Feedback 펄스수를 주기적으로 체크하여 펄스수의 변화량을 시간으로 나누어 계산됩니다. COMILX_MC_GetActualSpeed()함수를 사용하기전에 먼저 이함수를 사용하여 Actual Speed 체크 기능을 Enable 시켜야합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *dwInterval* : Feedback 펄스의 수를 체크하는 주기를 msec 단위로 설정합니다. Feedback 펄스의 주파수는 다음과 같은 식에 의해 계산됩니다.

$$V_a = \frac{\Delta C}{\Delta T} \times \frac{1}{R_u \times R_{io}}$$

여기서,

V_a : Feedback 펄스를 통하여 계측되는 모션의 실제 속도

ΔC : 체크 주기동안에 변화된 Feedback counter 값

ΔT : 체크주기, dwInterval 이 msec 단위로 설정되기 때문에 dwInterval/1000 를 의미합니다.

R_u : COMILX_MC_SetUnitSpeed()함수를 통하여 설정된 단위속도당 PPS 비

R_{io} : COMILX_MC_SetInOutRatio()함수를 통하여 설정된 Feedback 펄스와 Command 펄스의 분해능 비율(Resolution ratio)

▣ COMILX_MC_DisableActSpdChk

함수 원형

void COMILX_MC_DisableActSpdChk (HANDLE hDevice)

함수 설명

이 함수는 실제 모션의 속도(Actual Speed)를 체크하는 기능을 Disable 시킵니다. Actual Speed 를 체크할 필요가 없는 경우에는 Actual Speed Check 기능을 Disable 시키는 것이 좋습니다. 컴퓨터가 부팅되면 Actual Speed Check 기능은 기본적으로 Disable 됩니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▣ COMILX_MC_GetActualSpeed

함수 원형

double COMILX_MC_GetActualSpeed (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 EA/EB 단자로 입력되는 Feedback 펄스를 계측하여 실제 모션의 속도를 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

지정한 축의 Command 속도를 반환합니다. 이 값은 모터의 실제속도가 아니며 스텝모터 또는 서보모터 드라이버에 전달되는 Command 속도입니다. 또한 이 값의 단위는 단위는 COMILX_MC_SetUnitSpeed() 함수에 의하여 결정되며 기본적으로는 Pulses/sec 입니다.

참 고

□ 이 함수를 사용하기 전에 COMILX_MC_EnableActSpdChk()함수를 이용하여 Actual Speed 체크 기능을 Enable 시켜야합니다.

□ Feedback 펄스의 주파수는 다음과 같은 식에 의해 계산됩니다.

$$V_a = \frac{\Delta C}{\Delta T} \times \frac{1}{R_u \times R_{io}}$$

여기서,

V_a : Feedback 펄스를 통하여 계측되는 모션의 실제 속도

ΔC : 체크 주기동안에 변화된 Feedback counter 값

ΔT : 체크주기, dwInterval 이 msec 단위로 설정되기 때문에 dwInterval/1000 를 의미합니다.

R_u : COMILX_MC_SetUnitSpeed()함수를 통하여 설정된 단위속도당 PPS 비

R_{io} : COMILX_MC_SetInOutRatio()함수를 통하여 설정된 Feedback 펄스와 Command

펄스의 분해능 비율(Resolution ratio)

▣ COMILX_MC_GetPosition_A

함수 원형

double COMILX_MC_GetPosition_A (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 현재의 실제 위치(Actual Position)를 논리 단위로 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재의 실제 위치(Actual Position)를 논리 단위로 반환합니다. 논리 단위는 COMILX_MC_SetUnitDistance()함수에 의해 결정됩니다.

참 고

□ 실제 위치는 EA/EB 단자로 입력되는 Feedback 펄스를 카운트하여 계산됩니다.

□ Feedback 펄스의 카운트값과 논리적 실제 위치의 관계는 다음과 같습니다..

$$P_a = \frac{C}{R_u \times R_{io}}$$

여기서,

P_a : Actual Position

C : Feedback Count

R_u : COMILX_MC_SetUnitDistance()함수를 통하여 설정된 단위거리당 펄스 수

R_{io} : COMILX_MC_SetInOutRatio()함수를 통하여 설정된 Feedback 펄스와 Command 펄스의 분해능 비율(Resolution ratio)

▣ COMILX_MC_SetPosition_A

함수 원형

void COMILX_MC_SetPosition_A (HANDLE hDevice, int nChannel, double fPos)

함수 설명

이 함수는 현재의 실제 위치(Actual Position)값을 설정하며 단위는 논리단위입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fPos* : 새로이 설정할 실제 위치(Actual Positon)값. 이 값의 단위는 논리 단위이며 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다.

▣ COMILX_MC_GetPosition_C

함수 원형

double COMILX_MC_GetPosition_C (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 현재의 명령 위치(Command Position)를 논리 단위로 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재의 명령 위치(Command Position)를 논리 단위로 반환합니다. 논리 단위는 COMILX_MC_SetUnitDistance()함수에 의해 결정됩니다.

▣ COMILX_MC_SetPosition_C

함수 원형

void COMILX_MC_SetPosition_C (HANDLE hDevice, int nChannel, double fPos)

함수 설명

이 함수는 명령 위치(Command Position)값을 설정하며 단위는 논리단위입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fPos* : 새로이 설정할 명령 위치(Actual Positon)값. 이 값의 단위는 논리 단위이며 COMILX_MC_SetUnitDistance 함수에 의해 결정됩니다.

▣ COMILX_MC_GetCount_A

함수 원형

long COMILX_MC_GetCount_A (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 현재의 실제 위치(Actual Position) 카운트값을 반환합니다. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재의 실제 위치(Actual Position) 카운트값을 반환합니다. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

참 고

□ COMILX_MC_GetPosition_A() 함수가 COMILX_MC_SetUnitDistance()함수에 의해 결정되는 논리단위값을 기준으로 위치값을 반환하는데 반하여 COMILX_MC_GetCount_A() 함수는 순수한 펄스 카운트값을 반환합니다.

▣ COMILX_MC_SetCount_A

함수 원형

void COMILX_MC_SetCount_A (HANDLE hDevice, int nChannel, long nCount)

함수 설명

이 함수는 현재의 실제 위치(Actual Position) 카운트값을 설정합니다. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nCount* : 새로이 설정할 실제 위치(Actual Position) 카운트값. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

▣ COMILX_MC_GetCount_C

함수 원형

long COMILX_MC_GetCount_C (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 현재의 명령 위치(Command Position) 카운트값을 반환합니다. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재의 명령 위치(Command Position) 카운트값을 반환합니다. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

참 고

□ COMILX_MC_GetPosition_C() 함수가 COMILX_MC_SetUnitDistance()함수에 의해 결정되는 논리단위값을 기준으로 위치값을 반환하는데 반하여 COMILX_MC_GetCount_C() 함수는 순수한 펄스 카운트값을 반환합니다.

▣ COMILX_MC_SetCount_C

함수 원형

void COMILX_MC_SetCount_C (HANDLE hDevice, int nChannel, long nCount)

함수 설명

이 함수는 현재의 명령 위치(Command Position) 카운트값을 설정합니다. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nCount* : 새로이 설정할 명령 위치(Command Position) 카운트값. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

▣ COMILX_MC_GetCount_D

함수 원형

long COMILX_MC_GetCount_D (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 Deviation Counter 의 값을 읽어서 반환합니다. Deviation Counter 는 명령 위치(Command Position)와 실제 위치(Actual Position)간의 차이(Difference)를 카운트하는 카운터입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

현재의 명령 위치(Command Position)와 실제 위치(Actual Position)간의 차이(Difference)를 카운트하는 카운터의 값을 읽어서 반환합니다. 이 값은 논리단위가 아니며 순수한 펄스 수의 차를 나타냅니다. 만일 Command Pulse 와 Feedback Pulse 간의 분해능(Resolution)이 다르다면 이에 대한 보상은 이루어지지 않으므로 이 함수를 사용하는 것은 바람직하지 않습니다.

▣ COMILX_MC_SetCount_D

함수 원형

void COMILX_MC_SetCount_D (HANDLE hDevice, int nChannel, long nCount)

함수 설명

이 함수는 Deviation Counter 의 값을 새로이 설정합니다. Deviation Counter 는 명령 위치(Command Position)와 실제 위치(Actual Position)간의 차이(Difference)를 카운트하는 카운터입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nCount* : 새로이 설정할 Deviation Counter 의 카운트값. 이 값은 논리단위가 아닌 실제 펄스 카운트값입니다.

▣ COMILX_MC_GetCount_G

함수 원형

long COMILX_MC_GetCount_G (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 General Counter 의 값을 읽어서 반환합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

General Counter 의 카운트값.

▣ COMILX_MC_SetCount_G

함수 원형

void COMILX_MC_SetCount_G (HANDLE hDevice, int nChannel, long nCount)

함수 설명

이 함수는 General Counter 의 값을 새로이 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nCount* : 새로이 설정할 General Counter 의 카운트값.

▣ COMILX_MC_GetCount_R

함수 원형

long COMILX_MC_GetCount_R (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 Remained Counter 의 값을 읽어서 반환합니다. Remained Counter 는 In-Position 작업에서 현재 남은 출력 펄스의 수를 알려주는 카운터입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

반환값

Remained Counter 의 값을 읽어서 반환합니다.

▣ COMILX_MC_SetCount_R

함수 원형

void COMILX_MC_SetCount_R (HANDLE hDevice, int nChannel, long nCount)

함수 설명

이 함수는 Remained Counter 의 값을 새로이 설정합니다. Remained Counter 는 In-Position 작업에서 현재 남은 출력 펄스의 수를 알려주는 카운터입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nCount* : 새로이 설정할 General Counter 의 카운트값.

▣ 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	Reserved
2	Reserved
3	Reserved
4	Wait other axis
5	Wait ERC finished
6	Wait DIR change
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-LX501 에서는 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 stataus(1=ON)
BIT15	DRN	-DR input signal stataus(1=ON)
BIT16 ~BIT31	Reserved	

사용예

각종 모션 I/O 상태를 다음과 같이 체크할 수 있습니다.

```
……
……
char szState[2][4] = {“OFF”, “ON”};
ULONG dwMioStates = COMILX_MC_GetMioStatus(hDevice, 0)
Printf(“ALM state = %s\n”, szState[dwMioStates&0x2]);
Printf(“+EL state = %s\n”, szState[dwMioStates&0x4]);
Printf(“-EL state = %s\n”, szState[dwMioStates&0x8]);
Printf(“ORG state = %s\n”, szState[dwMioStates&0x10]);
……
……
```

5.4.11 I/O(입출력) 환경설정 함수

이 단원에서는 I/O(입출력) 환경설정 함수에 관하여 설명합니다. COMI-LX50x 모션제어 보드는 General Digital Input/Output 이외에 여러가지 모션 제어에 필요한 I/O 핀을 제공합니다. 여기에는 알람 신호(Alarm, ALM), 리미트 신호(Limit, EL) 등이 포함되며 서보 모터의 경우에는 INP, ERC 신호가 포함됩니다. 그리고 여러가지 Comparator(비교기) 또한 여기에 포함됩니다.

I/O(입출력) 환경설정에 관련된 함수들은 다음과 같습니다.

void COMILX_MC_SetFilterLogic(HANDLE hDevice, int nChannel, BOOL bEnable) 입력 신호에 필터로직을 적용할 지를 설정합니다. 필터 로직이 적용되는 I/O 신호 및 펄스 폭은 아래의 참고 항목을 참조하십시오.
void COMILX_MC_SetMioCfgALM (HANDLE hDevice, int nChannel, int nAlarmLogic, int nAlarmMode) Alarm 신호에 대한 입력모드를 설정합니다.
void COMILX_MC_GetMioCfgALM (HANDLE hDevice, int nChannel, int *pAlarmLogic, int *pAlarmMode) 현재 설정된 Alarm 신호에 대한 입력모드를 읽어옵니다.
void COMILX_MC_SetMioCfgCLR(HANDLE hDevice, int nChannel, int CntrSel, int nSignalType) CLR (카운터 클리어) 입력 신호에 대한 환경을 설정합니다.
void COMILX_MC_GetMioCfgCLR (HANDLE hDevice, int nChannel, int *pCntrSel, int *pnSignalType) 현재 설정된 CLR 입력 신호의 환경 설정값을 읽어옵니다.
void COMILX_MC_SetMioCfgDR(HANDLE hDevice, int nChannel, int nLogic) DR 입력 신호에 대한 환경을 설정합니다.
void COMILX_MC_GetMioCfgDR (HANDLE hDevice, int nChannel, int *pnLogic) 현재 설정된 DR 신호에 대한 환경 설정값을 읽어옵니다.
void COMILX_MC_SetMioCfgEL (HANDLE hDevice, int nChannel, int nElMode) EL 신호에 대한 모드를 설정합니다.

void COMILX_MC_GetMioCfgEL (HANDLE hDevice, int nChannel, int *pElMode) EL 신호에 대한 모드 설정값을 읽어옵니다.
void COMILX_MC_SetMioCfgERC (HANDLE hDevice, int nChannel, int nErcLogic, int nErcOnTime) ERC 신호의 입력 로직(Logic) 및 ON time 을 설정합니다.
void COMILX_MC_GetMioCfgERC (HANDLE hDevice, int nChannel, int *pErcLogic, int *pErcOnTime) ERC 신호의 입력 로직(Logic) 및 ON time 에 대한 설정값을 읽어옵니다.
void COMILX_MC_SetMioCfgINP (HANDLE hDevice, int nChannel, BOOL bInpEnable, int nInpLogic) INP 기능 및 신호에 대한 환경을 설정합니다.
void COMILX_MC_GetMioCfgINP (HANDLE hDevice, int nChannel, BOOL *pInpEnable, int *pInpLogic) INP 기능 및 신호에 대한 환경 설정값을 읽어옵니다.
void COMILX_MC_SetMioCfgSD (HANDLE hDevice, int nChannel, BOOL bSdEnable, int nSdLogic, int nSdLatch, int nSdMode) SD 신호에 대한 환경을 설정합니다. SD 신호는 Deceleration 시작점을 외부에서 입력하는 신호입니다.
void COMILX_MC_GetMioCfgSD (HANDLE hDevice, int nChannel, BOOL *pSdEnable, int *pSdLogic, int *pSdLatch, int *pSdMode) SD 신호에 대한 환경 설정값을 읽어옵니다.
void COMILX_MC_SetSoftLimit (HANDLE hDevice, int nChannel, double fLimitP, double fLimitN) 소프트웨어적인 Limit 를 설정합니다.
void COMILX_MC_EnableSoftLimit (HANDLE hDevice, int nChannel) 소프트웨어적인 Limit 의 적용을 Enable 시킵니다.
void COMILX_MC_DisableSoftLimit (HANDLE hDevice, int nChannel) 소프트웨어적인 Limit 의 적용을 Disable 시킵니다.
void COMILX_MC_SetErrorCompare (HANDLE hDevice, int nChannel, double fTol, int

bEnable) 명령 펄스(Command Pulse)와 Feedback 펄스간의 에러를 체크하는 기능을 설정합니다.
void COMILX_MC_SetGeneralCompare (HANDLE hDevice, int nChannel, int nCmpSrc, int nCmpMethod, int nCmpAction, double fData) General Comparator 의 비교 대상, 비교값 등을 설정합니다.

▣ COMILX_MC_SetFilterLogic

함수 원형

void COMILX_MC_SetFilterLogic(HANDLE hDevice, int nChannel, BOOL bEnable)

함수 설명

이 함수는 각종 I/O 신호에 필터 로직을 적용할지를 설정하는 함수입니다. 필터 로직이 적용되면 펄스폭이 너무 짧은 입력 신호는 무시되게 됩니다. 필터 로직이 적용되는 I/O 신호 및 펄스폭은 아래의 참고 항목을 참조하십시오.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *bEnable* : 필터로직을 적용할지를 결정합니다.

Value	Meaning
0	Filter disable
1	Filter enable

참 고

□ 필터 로직 적용이 Enable 됐을 때 필터 로직이 적용되는 신호 및 기준 펄스 폭은 다음과 같습니다.

필터 적용 I/O	필터 기준
+EL, -EL, SD, ORG, ALM, INP	펄스폭이 4μs 미만은 무시
+DR, -DR	펄스폭이 3.2ms 미만은 무시

▣ COMILX_MC_SetMioCfgALM

함수 원형

void COMILX_MC_SetMioCfgALM (HANDLE hDevice, int nChannel, int nAlarmLogic, int nAlarmMode)

함수 설명

이 함수는 Alarm 신호에 대한 입력모드를 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nAlarmLogic* : Alarm 신호 입력 펄스의 로직(Logic)을 설정합니다.

Value	Meaning
0	Active low
1	Active high

▶ *nAlarmMode* : Alarm 신호가 Logic ON 될때 모터가 반응하는 방식을 결정합니다.

Value	Meaning
0	Alarm 신호가 ON 되면 모션을 즉시 정지합니다.
1	Alarm 신호가 ON 되면 모션을 감속후에 정지합니다.

▣ COMILX_MC_GetMioCfgALM

함수 원형

void COMILX_MC_GetMioCfgALM (HANDLE hDevice, int nChannel, int *pAlarmLogic, int *pAlarmMode)

함수 설명

이 함수는 현재 설정된 Alarm 신호에 대한 입력모드를 읽어옵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pAlarmLogic* : Alarm 신호 입력 펄스의 로직(Logic) 설정값을 받아들일 변수의 주소값. 이 변수에 반환되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	Active low
1	Active high

▶ *pAlarmMode* : Alarm 신호에 대한 모터의 반응방식을 결정하는 모드 설정값을 받아들일 변수의 주소값. 이 변수에 반환되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	Alarm 신호가 ON 되면 모션을 즉시 정지합니다.
1	Alarm 신호가 ON 되면 모션을 감속후에 정지합니다.

▣ COMILX_MC_SetMioCfgCLR

함수 원형

void COMILX_MC_SetMioCfgCLR(HANDLE hDevice, int nChannel, int CntrSel, int nSignalType)

함수 설명

이 함수는 CLR 신호에 대한 환경을 설정합니다. CLR 신호는 COMI-LX50x 의 각종 카운터의 카운트값을 리셋합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *CntrSel* : CLR 신호에 의해 카운트값을 리셋하는 카운터를 지정합니다. COMI-LX50x 은 축당 4 개의 카운터를 제공합니다. CntrSel 파라미터의 각 비트는 다음과 같이 각 카운터의 Clear Enable/Disable 을 결정합니다.

BIT	Meaning
BIT0	Command position counter 의 클리어 여부를 결정합니다.(1=Enable)
BIT1	Feedback position counter 의 클리어 여부를 결정합니다.(1=Enable)
BIT2	Deviation counter 의 클리어 여부를 결정합니다.(1=Enable)
BIT3	General counter 의 클리어 여부를 결정합니다.(1=Enable)

▶ *nSignalType* : CLR 신호의 신호 유형을 결정합니다.

Value	Meaning
0	Clears at the falling edge
1	Clears at the rising edge
2	Clears at low level
3	Clears at high level

▣ COMILX_MC_GetMioCfgCLR

함수 원형

void COMILX_MC_GetMioCfgCLR (HANDLE hDevice, int nChannel, int *pCntrSel, int *pnSignalType)

함수 설명

이 함수는 CLR 신호에 대한 모드 설정값을 읽어옵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pCntrSel* : CLR 적용 카운터 설정값을 받아올 변수의 주소값.

▶ *pnSignalType* : CLR 신호의 신호 형태 설정값을 받아올 변수의 주소값

▣ COMILX_MC_SetMioCfgDR

함수 원형

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

함수 설명

이 함수는 +/-DR 신호에 대한 환경을 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nLogic* : +/-DR 신호 입력 펄스의 로직(Logic)을 설정합니다.

Value	Meaning
0	Active low
1	Active high

▣ COMILX_MC_GetMioCfgDR

함수 원형

```
void COMILX_MC_GetMioCfgDR (HANDLE hDevice, int nChannel, int *pnLogic)
```

함수 설명

이 함수는 +/-DR 신호에 대한 모드 설정값을 읽어옵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pnLogic* : +/-DR 신호의 입력 로직 설정값을 받아올 변수의 주소값. 이 변수에 반환되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	Active low
1	Active high

▣ COMILX_MC_SetMioCfgEL

함수 원형

void COMILX_MC_SetMioCfgEL (HANDLE hDevice, int nChannel, int nElMode)

함수 설명

이 함수는 EL 신호에 대한 모드를 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nElMode* : EL 신호가 Logic ON 될때 모터가 반응하는 방식을 결정합니다.

Value	Meaning
0	EL 신호가 ON 되면 모션을 즉시 정지합니다.
1	EL 신호가 ON 되면 모션을 감속후에 정지합니다.

▣ COMILX_MC_GetMioCfgEL

함수 원형

void COMILX_MC_GetMioCfgEL (HANDLE hDevice, int nChannel, int *pElMode)

함수 설명

이 함수는 EL 신호에 대한 모드 설정값을 읽어옵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pElMode* : EL 신호에 대한 모드 설정값을 받아들일 변수의 주소값. 이 변수에 반환되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	EL 신호가 ON 되면 모션을 즉시 정지합니다.
1	EL 신호가 ON 되면 모션을 감속후에 정지합니다.

▣ COMILX_MC_SetMioCfgERC

함수 원형

void COMILX_MC_SetMioCfgERC (HANDLE hDevice, int nChannel, int nErcLogic, int nErcOnTime)

함수 설명

이 함수는 ERC 신호의 입력 로직(Logic) 및 ON time을 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nErcLogic* : ERC 입력 신호의 로직(Logic)을 설정합니다.

Value	Meaning
0	Active low
1	Active high

▶ *nErcOnTime* : INP 입력 신호의 로직(Logic)을 설정합니다.

Value	ON time of ERC signal
0	12 us
1	102us
2	409us
3	1.6ms
4	13ms
5	52ms
6	104ms

▣ COMILX_MC_GetMioCfgERC

함수 원형

void COMILX_MC_GetMioCfgERC (HANDLE hDevice, int nChannel, int *pErcLogic, int *pErcOnTime)

함수 설명

이 함수는 ERC 신호의 입력 로직(Logic) 및 ON time 에 대한 설정값을 읽어옵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pErcLogic* : ERC 입력 신호의 로직(Logic) 설정값을 받아들일 변수의 주소값. 이 변수에 전달되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	Active low
1	Active high

▶ *pErcOnTime* : INP 입력 신호의 로직(Logic) 설정값을 받아들일 변수의 주소값. 이 변수에 전달되는 값의 의미는 다음과 같습니다.

Value	ON time of ERC signal
0	12 us
1	102us
2	409us
3	1.6ms
4	13ms
5	52ms
6	104ms

▣ COMILX_MC_SetMioCfgINP

함수 원형

void COMILX_MC_SetMioCfgINP (HANDLE hDevice, int nChannel, BOOL bInpEnable, int nInpLogic)

함수 설명

이 함수는 INP 기능 및 신호에 대한 환경을 설정합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *bInpEnable* : INP 기능을 Enable/Disable 시킵니다.

Value	Meaning
0	INP 기능을 Disable 시킵니다.
1	INP 기능을 Enable 시킵니다. INP 기능이 Enable 되면 command 펄스 출력이 완료되도 INP 입력이 ON 이 되지 않으면 모션이 완료되지 않은 것으로 간주됩니다.

▶ *nInpLogic* : INP 입력 신호의 로직(Logic)을 설정합니다.

Value	Meaning
0	Active low
1	Active high

▣ COMILX_MC_GetMioCfgINP

함수 원형

void COMILX_MC_GetMioCfgINP (HANDLE hDevice, int nChannel, BOOL *pInpEnable, int *pInpLogic)

함수 설명

이 함수는 INP 기능 및 신호에 대한 환경 설정값을 읽어옵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pInpEnable* : INP 기능 설정값을 읽어올 변수의 주소값. 이 변수에 전달되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	INP 기능을 Disable 시킵니다.
1	INP 기능을 Enable 시킵니다. INP 기능이 Enable 되면 command 펄스 출력이 완료되도 INP 입력이 ON 이 되지 않으면 모션이 완료되지 않은 것으로 간주됩니다.

▶ *pInpLogic* : INP 입력 신호의 로직(Logic)을 설정합니다.

Value	Meaning
0	Active low
1	Active high

▣ COMILX_MC_SetMioCfgSD

함수 원형

void COMILX_MC_SetMioCfgSD (HANDLE hDevice, int nChannel, BOOL bSdEnable, int nSdLogic, int nSdLatch, int nSdMode)

함수 설명

이 함수는 SD 신호에 대한 환경을 설정합니다. SD 신호는 Deceleration 시작점을 외부에서 입력하는 신호입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *bSdEnable* : SD 신호를 Enable/Disable 시킵니다.

Value	Meaning
0	SD 신호 Disable => Deceleration 시작점이 자동으로 결정됩니다.
1	SD 신호 Enable => 외부에서 입력되는 SD 신호에 의해 Deceleration 시작점이 결정됩니다.

▶ *nSdLogic* : SD 입력 신호의 로직(Logic)을 설정합니다.

Value	Meaning
0	Active low
1	Active high

▶ *nSdLatch* : SD 신호에 대한 Latch 여부를 결정합니다.

Value	Meaning
0	SD 신호를 Latch 하지 않음
1	SD 신호를 Latch 함

▶ *nSdMode* : SD 입력 신호에 대한 모터의 반응을 결정합니다.

Value	Meaning
0	Decleration 만 적용
1	Deceleration 후 정지

▣ COMILX_MC_GetMioCfgSD

함수 원형

void COMILX_MC_GetMioCfgSD (HANDLE hDevice, int nChannel, BOOL *pSdEnable, int *pSdLogic, int *pSdLatch, int *pSdMode)

함수 설명

이 함수는 SD 신호에 대한 환경 설정값을 읽어옵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *pSdEnable* : SD 신호의 Enable/Disable 설정값을 읽어들일 변수의 주소값. 이 변수에 전달되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	SD 신호 Disable => Deceleration 시작점이 자동으로 결정됩니다.
1	SD 신호 Enable => 외부에서 입력되는 SD 신호에 의해 Deceleration 시작점이 결정됩니다.

▶ *pSdLogic* : SD 입력 신호의 로직(Logic) 설정값을 읽어들일 변수의 주소값. 이 변수에 전달되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	Active low
1	Active high

▶ *pSdLatch* : SD 신호의 Latch 설정값을 읽어들일 변수의 주소값. 이 변수에 전달되는 값의 의미는 다음과 같습니다.

Value	Meaning
0	SD 신호를 Latch 하지 않음
1	SD 신호를 Latch 함

▶ *pSdMode* : SD 모드 설정값을 읽어들일 변수의 주소값. 이 변수에 전달되는 값의 의미는 다음과 같습니다.

Value	Meaning

0	Decleration만 적용
1	Deceleration 후 정지

▣ COMILX_MC_SetSoftLimit

함수 원형

void COMILX_MC_SetSoftLimit (HANDLE hDevice, int nChannel, double fLimitP, double fLimitN)

함수 설명

이 함수는 소프트웨어적인 Limit 를 설정합니다. 소프트웨어적인 Limit 은 리밋센서의 설치가 용이하지 않을 때 안전성을 위하여 소프트웨어적인 리미트를 설정하는 것입니다. 소프트웨어적인 Limit 은 Command pulse 카운터의 절대값이 지정한 +/- Limit 값보다 같거나 크게 되면 모션을 자동 정지하도록 합니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fLimitP* : (+)방향 Limit 값을 설정합니다.

▶ *fLimitN* : (-)방향 Limit 값을 설정합니다.

사용예

```
……
……
// 소프트웨어적인 리미트값을 설정한다. //
COMILX_MC_SetSoftLimit (hDevice, 0, -500000, 500000);
// 소프트웨어적인 리미트 적용을 Enable 시킨다. //
COMILX_MC_EnableSoftLimit (hDevice, 0);
……
……
```

▣ COMILX_MC_EnableSoftLimit

함수 원형

```
void COMILX_MC_EnableSoftLimit (HANDLE hDevice, int nChannel)
```

함수 설명

이 함수는 소프트웨어적인 Limit 의 적용을 Enable 시킵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▣ COMILX_MC_DisableSoftLimit

함수 원형

void COMILX_MC_DisableSoftLimit (HANDLE hDevice, int nChannel)

함수 설명

이 함수는 소프트웨어적인 Limit 의 적용을 Disable 시킵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▣ COMILX_MC_SetErrorCompare

함수 원형

void COMILX_MC_SetErrorCompare (HANDLE hDevice, int nChannel, double fTol, int bEnable)

함수 설명

이 함수는 명령 펄스(Command Pulse)와 Feedback 펄스간의 에러를 체크하는 기능을 설정합니다. 에러 체크 기능을 Enable 시키면 Command 펄스 수와 Feedback 펄스 수의 차가 지정한 Tolerance 보다 크면 인터럽트(Event Interrupt 의 BIT10)를 발생시킵니다. Event Interrupt 에 관해서는 COMILX_MC_GetIntStatus() 함수를 참조하십시오.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fTol* : Position error tolerance.

▶ *bEnable* : 에러 체크 기능을 Enable 또는 Disable 시킵니다.

▣ COMILX_MC_SetGeneralCompare

함수 원형

void COMILX_MC_SetGeneralCompare (HANDLE hDevice, int nChannel, int nCmpSrc, int nCmpMethod, int nCmpAction, double fData)

함수 설명

이 함수는 General Comparator 의 비교 대상, 비교값 등을 설정합니다. 비교 대상 카운터값과 지정한 데이터값이 비교 조건을 만족할 때 인터럽트(Event Interrupt 의 BIT11)를 발생시킵니다. Event Interrupt 에 관해서는 COMILX_MC_GetIntStatus() 함수를 참조하십시오.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *nCmpSrc* : 비교 대상 카운터를 설정합니다.

Value	Comapare Source
0	Command Counter
1	Feedback Counter
2	Error Counter
3	General Counter

▶ *nCmpMethod* : 비교 조건을 설정합니다.

Value	Comapare Source
0	비교를 하지 않습니다. => Disable
1	fData=Counter(Directionless) 이면 Action 을 취합니다.
2	fData=Counter(+ Direction) 이면 Action 을 취합니다.
3	fData=Counter(- Direction) 이면 Action 을 취합니다.
4	fData>Counter 이면 Action 을 취합니다.
5	fData<Counter 이면 Action 을 취합니다.

▶ *nCmpAction* : 비교 조건이 성립되었을 때 취할 Action 을 설정합니다.

Value	Comapare Source

0	인터럽트만 발생
1	즉시 정지 및 인터럽트 발생
2	감속 후 정지 및 인터럽트 발생
3	속도 변경 및 인터럽트 발생

▶ *fData* : 비교 기준 값.

5.4.12 인터럽트 관련 함수

이 단원에서는 인터럽트에 관련된 함수들을 소개합니다. 인터럽트는 특정 상황이 발생되었을 때 사용자(또는 프로그래머)에게 이것을 알려주기 위한 것입니다. 윈도우 환경에서는 일반 Application 레벨에서 인터럽트를 처리할 수 없으므로 이벤트를 통하여 Application 에게 인터럽트가 발생하였음을 알려줍니다.

사용자 프로그램에서 인터럽트를 처리하기 위해서는 먼저 CreateEvent() 등의 표준 윈도우 API 함수를 통하여 이벤트를 생성하고 COMILX_MC_MaskInterrupt() 함수를 이용하여 이벤트를 등록하여야 합니다. 이벤트가 등록된 후 사용자는 쓰레드(Thread) 또는 타이머 콜백 함수에서 이벤트가 발생했는지를 체크하고 이벤트가 발생하였으면 COMILX_MC_GetAxisIntState() 함수와 COMILX_MC_GetIntStatus() 함수를 통하여 인터럽트 Status 를 확인하여 각 Status 에 따라 적절한 Action 을 취하여야 합니다.

인터럽트 처리에 관련된 함수는 다음과 같습니다.

void COMILX_MC_MaskInterrupt (HANDLE hDevice, int nChannel, long dwMask) 어떠한 조건의 인터럽트를 받아들일 것인지를 설정합니다.
void COMILX_MC_EnableInterrupt (HANDLE hDevice, HANDLE hEvent) 인터럽트 이벤트 통지 기능을 Enable 시킵니다.
void COMILX_MC_DisableInterrupt (HANDLE hDevice) 인터럽트 이벤트 통지 기능을 Disable 시킵니다.
BOOL COMILX_MC_GetAxisIntState (HANDLE hDevice, int nChannel) 각 축에 대하여 인터럽트가 발생하였는지를 알려주는 함수입니다.
void COMILX_MC_GetIntStatus (HANDLE hDevice, int nChannel, long *pErrorStatus, long *pEventStatus) 각 축의 인터럽트가 발생한 원인을 알려주는 함수입니다.

▣ 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.

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	Reserved
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_EnableInterrupt

함수 원형

void COMILX_MC_EnableInterrupt (HANDLE hDevice, HANDLE hEvent)

함수 설명

이 함수는 인터럽트 이벤트 통지 기능을 Enable 시킵니다. 즉, 인터럽트가 발생하였을 때 사용자 Application 에게 인터럽트가 발생되었음을 알려주는 기능을 Enable 시킵니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *hEvent* : 인터럽트가 발생하였을 때 사용자 Application 에게 인터럽트가 발생되었음을 알려주기 위하여 사용될 이벤트 핸들. 이 핸들은 CreateEvent() 함수와 같이 이벤트 핸들을 생성해주는 표준 윈도우 API 함수를 통하여 먼저 이벤트 핸들을 생성한 후에 생성된 이벤트 핸들을 이 함수를 통하여 등록하여야 합니다.

참 고

□ 사용예는 COMILX_MC_GetIntStatus() 함수 설명편을 참조하십시오.

▣ COMILX_MC_DisableInterrupt

함수 원형

void COMILX_MC_DisableInterrupt (HANDLE hDevice)

함수 설명

이 함수는 인터럽트 이벤트 통지 기능을 Disable 시킵니다. 기본적으로 이벤트 통지 기능은 Disable 된 상태입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

참 고

□ 사용예는 COMILX_MC_GetIntStatus() 함수 설명편을 참조하십시오.

▣ 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. 채널별로 각각 다른 인터럽트 조건을 설정할 수 있습니다.

반환값

각 축의 인터럽트 발생 상태를 알려주는 32 비트 정수값

Bit	인터럽트 발생 조건
BIT0	X 축의 인터럽트 상태. 0=>No interrupt, 1=>Interrupt 발생
BIT1	Y 축의 인터럽트 상태. 0=>No interrupt, 1=>Interrupt 발생
BIT2	Z 축의 인터럽트 상태. 0=>No interrupt, 1=>Interrupt 발생
BIT3	U 축의 인터럽트 상태. 0=>No interrupt, 1=>Interrupt 발생
BIT4 ~ BIT31	Reserved

참 고

□ 사용예는 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. 채널별로 각각 다른 인터럽트 조건을 설정할 수 있습니다.

▶ *pErrorStatus* : 에러에 관련된 인터럽트의 상태를 나타내는 값을 받아들일 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 받아들일 변수의 주소값. 이벤트 인터럽트는 COMILX_MC_MaskInterrupt () 함수를 통하여 마스크(Mask) 가능합니다. 이 변수에 전달되는 값은 32 비트 정수값이며 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	Compared triggered for axis 0, 1
BIT13	Reserved
BIT14	Latched for axis2,3
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 이벤트가
// 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, 0xfff);
    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();
}
```

Appendix A

라이브러리 함수 리스트

본 부록에서는 COMI-LX50x 시리즈 라이브러리를 사용하는데 있어서 각 함수의 이해 및 검색에 도움을 주고자 라이브러리 함수에 대한 다양한 리스트를 제공합니다.

A.1 함수 색인 ····· 380

Appendix A 라이브러리 함수 리스트

A.1 함수 색인

라이브러리 및 디바이스 시작/종료		Page
BOOL	**COMILX_LoadDll (void)** 라이브러리 프로그램을 로드(load)합니다.	**124**
void	**COMILX_UnloadDll (void)** 라이브러리 프로그램을 언로드(load)합니다.	**126**
HANDLE	**COMILX_LoadDevice (COMIDAS_DEVID deviceID, ULONG instance)** 디바이스를 로드(load)합니다.	**127**
void	**COMILX_UnloadDevice (HANDLE hDevice)** 디바이스를 언로드(Unload)합니다.	**129**

디지털 입출력		Page
int	**COMILX_DI_GetOne (HANDLE hDevice, int ch)** 지정한 Digital Input 채널의 Status 를 반환합니다.	**131**
DWORD	**COMILX_DI_GetAll (HANDLE hDevice)** 디바이스의 모든 Digital Input 채널의 Status 를 반환합니다	**133**
void	**COMILX_DO_PutOne (HANDLE hDevice, int ch, int status)** 지정한 Digital Output 채널에 지정한 Status 로 출력을 내보냅니다.	**135**
void	**COMILX_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)** 해당 디바이스의 모든 Digital Output 채널에 출력을 내보냅니다.	**137**
int	**COMILX_DO_GetOne (HANDLE hDevice, int ch)** 지정한 Digital Output 채널의 현재 출력값을 반환합니다.	**139**
DWORD	**COMILX_DO_GetAll (HANDLE hDevice)** 모든 채널의 Digital Output 출력값을 반환합니다.	**140**

모션제어 (모션 초기화 및 환경설정)		Page
void	**COMILX_MC_Reset (HANDLE hDevice)** 모션 제어 보드의 하드웨어와 소프트웨어적인 모든 상태를 리셋합니다.	**143**
void	**COMILX_MC_SetBlockingMode (HANDLE hDevice, BOOL bBlocking)** Blocking 모드를 결정합니다.	**144**
void	**COMILX_MC_SetOutputMode (HANDLE hDevice, int nChannel, int nOutputMode)** Command 펄스의 출력 모드를 설정합니다.	**146**

int	**COMILX_MC_GetOutputMode(HANDLE hDevice, int nChannel)** 현재 설정된 Command 펄스의 출력 모드를 반환합니다.	147
void	**COMILX_MC_SetInputMode(HANDLE hDevice, int nChannel, int nInputMode, int nPulseLogic)** Feedback(엔코더) 펄스의 입력 모드를 설정합니다.	148
void	**COMILX_MC_GetInputMode(HANDLE hDevice, int nChannel, int *pInputMode, int *pPulseLogic)** 현재 설정된 Feedback 펄스의 입력 모드를 반환합니다.	149
void	**COMILX_MC_SetSpeedRange(HANDLE hDevice, int nChannel, double fMaxSpeed)** 이 함수는 출력 펄스의 주파수 범위를 설정합니다.	150
void	**COMILX_MC_SetUnitDistance(HANDLE hDevice, int nChannel, double fUnitDist)** 논리적 단위 거리에 대한 펄스 수를 설정합니다.	152
double	**COMILX_MC_GetUnitDistance(HANDLE hDevice, int nChannel)** 현재 설정된 논리적 단위 거리에 대한 펄스 수를 반환합니다.	154
void	**COMILX_MC_SetUnitSpeed(HANDLE hDevice, int nChannel, double fUnitSpeed)** 논리적 단위 속도에 대한 실제 펄스 출력 속도(PPS)를 설정합니다.	155
double	**COMILX_MC_GetUnitSpeed (HANDLE hDevice, int nChannel)** 현재 설정된 논리적 단위 속도에 대한 실제 펄스 출력 속도(PPS)를 반환합니다.	158
double	**COMILX_MC_SetInOutRatio (HANDLE hDevice, int nChannel, double fRatio)** Feedback 펄스와 Command 펄스의 분해능 비율(Resolution ratio)을 설정합니다.	159

모션제어 (Single Axis 모션)		Page
void	**COMILX_MC_SetSpeedMode(HANDLE hDevice, int nChannel, int nModeIndex)** Motion 의 속도 모드를 설정합니다.	162
void	**COMILX_MC_SetSpeed (HANDLE hDevice, int nChannel, double fIniSpeed, double fWorkSpeed)** Motion 의 속도를 설정합니다.	167
void	**COMILX_MC_SetAccel(HANDLE hDevice, int nChannel, double fAccel, double fDecel)** Motion 의 가/감속도를 설정합니다.	171
void	**COMILX_MC_SetScurve (HANDLE hDevice, int nChannel, double fSVacc, double fSVdec)** 속도모드를 S-curve 속도 패턴으로 설정한 경우에 S-curve Section 의 범위를 속도단위로 설정합니다.	175

void	**COMILX_MC_StartVMove (HANDLE hDevice, int nChannel, int nDirection)** 작업속도까지 가속한 후에 작업속도를 유지하며 정지함수가 호출될 때까지 지정한 방향으로의 모션을 계속 수행합니다.	**179**
void	**COMILX_MC_StartMove (HANDLE hDevice, int nChannel, double fDistance)** 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 함수는 모션을 시작시킨 후 바로 Return 합니다.	**181**
void	**COMILX_MC_Move (HANDLE hDevice, int nChannel, double fDistance)** 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 함수는 모션이 완료될때까지 Return 되지 않습니다.	**184**
void	**COMILX_MC_StartMoveTo (HANDLE hDevice, int nChannel, double fPosition)** 지정한 절대좌표로의 이동을 수행합니다. 이 함수는 모션(Motion)을 시작 시킨 후에 바로 Return 합니다.	**187**
void	**COMILX_MC_MoveTo (HANDLE hDevice, int nChannel, double fPosition)** 지정한 절대좌표로의 이동을 수행합니다. 이 함수는 모션이 완료될때까지 Return 되지 않습니다.	**190**
void	**COMILX_MC_Stop (HANDLE hDevice, int nChannel)** 지정한 축에 대한 모션을 감속 후 정지합니다.	**193**
void	**COMILX_MC_EmgStop (HANDLE hDevice, int nChannel)** 지정한 축에 대한 모션을 감속없이 즉시 정지합니다.	**194**
BOOL	**COMILX_MC_Done (HANDLE hDevice, int nChannel)** 하나의 축에 대하여 모션이 완료됐는지를 체크합니다.	**196**

모션제어 (Multi-Axis 동시제어)		Page
void	**COMILX_MC_StartVMoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], int nDirList[])** 여러 개의 축에 대하여 Velocity Move 작업을 동시에 시작합니다.	**199**
void	**COMILX_MC_StartMoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fDistList[])** 여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 동시에 시작합니다.	**201**
void	**COMILX_MC_MoveAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fDistList[])** 여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 수행합니다.	**203**
void	**COMILX_MC_StartMoveToAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fPosList[])** 여러 개의 축에 대하여 지정한 절대좌표로의 이동을 동시에 시작합니다.	**205**
void	**COMILX_MC_MoveToAll (HANDLE hDevice, int nNumAxis, int nAxisList[], double fPosList[])** 여러 개의 축에 대하여 지정한 절대좌표로의 이동을 수행합니다.	**207**

void	**COMILX_MC_StopAll (HANDLE hDevice, int nNumAxis, int nAxisList[])** 여러 개의 축에 대한 모션을 동시에 감속 후 정지합니다.	**209**
void	**COMILX_MC_EmgStopAll (HANDLE hDevice, int nNumAxis, int nAxisList[])** 여러 개의 축에 대한 모션을 동시에 감속없이 즉시 정지합니다.	**211**
BOOL	**COMILX_MC_AllDone (HANDLE hDevice, int nNumAxis, int nAxisList[])** 여러 개의 축에 대하여 지정한 모든 축의 모션이 완료됐는지를 체크합니다.	**213**

모션제어 (Coordinated Motion) **Page**

BOOL	**COMILX_MC_MapAxes (HANDLE hDevice, int nMapIndex, unsigned char bMapMask)** Coordinated Motion을 수행할 축들을 그룹화합니다	**217**
void	**COMILX_MC_SetSpeedModeMx (HANDLE hDevice, int nMapIndex, int nModeIndex)** Coordinated Motion의 속도 모드를 설정합니다.	**220**
void	**COMILX_MC_SetSpeedMx (HANDLE hDevice, int nMapIndex, double fSpeed, double fAccel)** Coordinated Motion의 속도 및 가속도를 설정합니다	**222**
void	**COMILX_MC_StartLine(HANDLE hDevice, int nMapIndex, double fDistList[])** 현재 위치로부터의 상대 좌표로의 직선 보간 이동을 수행합니다.	**226**
void	**COMILX_MC_Line (HANDLE hDevice, int nMapIndex, double fDistList[])** 현재 위치로부터의 상대 좌표로의 직선 보간 이동을 수행합니다	**228**
void	**COMILX_MC_StartLineTo(HANDLE hDevice, int nMapIndex, double fPosList[])** 절대 좌표로의 직선 보간 이동을 수행합니다.	**230**
void	**COMILX_MC_LineTo(HANDLE hDevice, int nMapIndex, double fPosList[])** 절대 좌표로의 직선 보간 이동을 수행합니다.	**234**
void	**COMILX_MC_StartArc_a(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fEndAngle)** 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	**238**
void	**COMILX_MC_Arc_a(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fEndAngle)** 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	**240**
void	**COMILX_MC_StartArc_p(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fXEndPointDist, double fYEndPointDist)** 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	**243**
void	**COMILX_MC_Arc_p(HANDLE hDevice, int nMapIndex, double fXCentOffset, double fYCentOffset, double fXEndPointDist, double fYEndPointDist)** 상대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	**245**

void	**COMILX_MC_StartArcTo_a(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fEndAngle)** 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	249
void	**COMILX_MC_ArcTo_a(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fEndAngle)** 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	251
void	**COMILX_MC_StartArcTo_p(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fXEndPos, double fYEndPos)** 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	257
void	**COMILX_MC_ArcTo_p(HANDLE hDevice, int nMapIndex, double fXCent, double fYCent, double fXEndPos, double fYEndPos)** 절대좌표를 파라미터로 하여 원호보간 이동을 수행합니다.	259
BOOL	**COMILX_MC_MxDone (HANDLE hDevice, int nMapIndex)** Coordinated Motion이 완료됐는지를 체크합니다.	265

모션제어 (속도 및 위치 오버라이딩(Overriding)) Page

void	**COMILX_MC_OverrideSpeedSet (HANDLE hDevice, int nChannel)** Single Axis 모션이 진행되고 있는 중에 속도를 변경	267
void	**COMILX_MC_OverrideSpeedSetAll (HANDLE hDevice, int nNumAxis, int nAxisList[])** 여러 축에 대하여 동시에 속도를 변경.	269
void	**COMILX_MC_OverrideMove (HANDLE hDevice, int nChannel, double fNewDistance)** COMILX_MC_StartMove 함수를 통하여 수행되는 상대좌표 In-position 모션에 대하여 상대좌표값, 즉 목표 거리값을 수정	272
void	**COMILX_MC_OverrideMoveTo (HANDLE hDevice, int nChannel, double fNewPosition)** COMILX_MC_StartMoveTo 함수를 통하여 수행되는 절대좌표 In-position 모션에 대하여 목표 절대좌표값을 수정.	273

모션제어 (원점 복귀(Home Return)) Page

void	**COMILX_MC_SetHomeConfig (HANDLE hDevice, int nChannel, int nOrgMode, int nOrgLogic, int nEzCount, int nEzLogic, int nErcOut)** 원점 복귀 작업을 수행하기 위한 여러가지 환경설정을 수행합니다.	283
void	**COMILX_MC_HomeMove (HANDLE hDevice, int nChannel, int nDirection)** 원점 복귀 작업을 수행하기 위한 여러가지 환경설정을 수행합니다.	285

모션제어 (Manual Pulser 모드 모션)		Page
void	**COMILX_MC_SetPE(HANDLE hDevice, int nChannel, BOOL bEnable)** Manual Puser 모드와 External Switch Operation 기능을 Enable/Disable 시킵니다.	289
void	**COMILX_MC_SetPulserInputMode (HANLE hDevice, int nChannel, int nInputMode, BOOL bInverse)** Pulser 입력신호 대한 환경을 설정합니다.	290
void	**COMILX_MC_PulserHomeMove (HANDLE hDevice, int nChannel, int nHomeType)** Pulser Input 에 의한 원점 복귀 작업을 수행합니다.	291
void	**COMILX_MC_StartPulserVMove (HANDLE hDevice, int nChannel)** Stop 함수가 호출될 때까지 Pulser 신호 입력에 맞추어 계속적인 모션을 수행합니다.	292
void	**COMILX_MC_StartPulserMove (HANDLE hDevice, int nChannel, double fDistance)** Pulser 신호 입력에 맞추어 지정한 거리(상대좌표)만큼 이동을 수행합니다.	294
void	**COMILX_MC_PulserMove (HANDLE hDevice, int nChannel, double fDistance)** Pulser 신호 입력에 맞추어 지정한 거리(상대좌표)만큼 이동을 수행합니다.	296
void	**COMILX_MC_StartPulserMoveTo (HANDLE hDevice, int nChannel, double fPosition)** Pulser 신호 입력에 맞추어 지정한 위치(절대좌표)로 이동을 수행합니다.	298
void	**COMILX_MC_PulserMoveTo (HANDLE hDevice, int nChannel, double fPosition)** Pulser 신호 입력에 맞추어 지정한 위치(절대좌표)로 이동을 수행합니다.	300

모션제어 (External Switch Operation)		Page
void	**COMILX_MC_SetPE(HANDLE hDevice, int nChannel, BOOL bEnable)** Manual Puser 모드와 External Switch Operation 기능을 Enable/Disable 시킵니다.	303
void	**COMILX_MC_StartVMoveEx(HANDLE hDevice, int nChannel)** External switch operation 모드하에 Velocity-Move 를 시작합니다.	304
void	**COMILX_MC_StartMoveEx(HANDLE hDevice, int nChannel, double fDistance)** External switch operation 모드하에 Relative In-Position(상대좌표 위치결정) 모션을 시작합니다.	306

모션제어 (리스트 모션(Listed Motion))		Page
void	**COMILX_MC_BeginList (HANDLE hDevice)** List Motion 에서 수행할 작업들의 등록을 시작합니다.	312
void	**COMILX_MC_EndList (HANDLE hDevice)** List Motion 에서 수행할 작업들의 등록을 종료합니다.	313

void **COMILX_MC_StartListMotion (HANDLE hDevice)** **314**
List Motion으로 등록된 작업들에 대하여 실제로 모션을 시작합니다.

void **COMILX_MC_AbortListMotion (HANDLE hDevice)** **315**
리스트 모션을 수행하고 있는 중에 리스트 모션을 중지합니다.

void **COMILX_MC_CheckListMotionDone (HANDLE hDevice)** **316**
리스트 모션 수행이 완료됐는지를 알려줍니다.

모션제어 (상태 감시 및 제어) Page

double **COMILX_MC_GetCurSpeed (HANDLE hDevice, int nChannel)** **319**
현재 출력되고 있는 Command 속도를 반환합니다.

void **COMILX_MC_EnableActSpdChk (HANDLE hDevice, long dwInterval)** **320**
실제 모션의 속도를 체크하는 기능을 Enable 시킵니다.

void **COMILX_MC_DisableActSpdChk (HANDLE hDevice)** **321**
실제 모션의 속도(Actual Speed)를 체크하는 기능을 Disable 시킵니다.

double **COMILX_MC_GetActualSpeed (HANDLE hDevice, int nChannel)** **322**
EA/EB 단자로 입력되는 Feedback 펄스를 계측하여 실제 모션의 속도를 반환합니다.

double **COMILX_MC_GetPosition_A (HANDLE hDevice, int nChannel)** **324**
현재의 실제 위치(Actual Position)를 논리 단위로 반환합니다.

void **COMILX_MC_SetPosition_A (HANDLE hDevice, int nChannel, double fPos)** **325**
현재의 실제 위치(Actual Position)값을 설정하며 단위는 논리단위입니다.

double **COMILX_MC_GetPosition_C (HANDLE hDevice, int nChannel)** **326**
현재의 명령 위치(Command Position)를 논리 단위로 반환합니다.

void **COMILX_MC_SetPosition_C (HANDLE hDevice, int nChannel, double fPos)** **327**
명령 위치(Command Position)값을 설정하며 단위는 논리단위입니다.

long **COMILX_MC_GetCount_A (HANDLE hDevice, int nChannel)** **328**
현재의 실제 위치(Actual Position) 카운트값을 반환합니다.

void **COMILX_MC_SetCount_A (HANDLE hDevice, int nChannel, long nCount)** **329**
현재의 실제 위치(Actual Position) 카운트값을 설정합니다.

long **COMILX_MC_GetCount_C (HANDLE hDevice, int nChannel)** **330**
현재의 명령 위치(Command Position) 카운트값을 반환합니다.

void **COMILX_MC_SetCount_C (HANDLE hDevice, int nChannel, long nCount)** **331**
현재의 명령 위치(Command Position) 카운트값을 설정합니다.

long **COMILX_MC_GetCount_D (HANDLE hDevice, int nChannel)** **332**
Deviation Counter 의 값을 읽어서 반환합니다.

void	**COMILX_MC_SetCount_D (HANDLE hDevice, int nChannel, long nCount)** Deviation Counter 의 값을 새로이 설정합니다.	**333**
long	**COMILX_MC_GetCount_G (HANDLE hDevice, int nChannel)** General Counter 의 값을 읽어서 반환합니다.	**334**
void	**COMILX_MC_SetCount_G (HANDLE hDevice, int nChannel, long nCount)** General Counter 의 값을 새로이 설정합니다.	**335**
long	**COMILX_MC_GetCount_R (HANDLE hDevice, int nChannel)** Remained Counter 의 값을 읽어서 반환합니다.	**336**
void	**COMILX_MC_SetCount_R (HANDLE hDevice, int nChannel, long nCount)** Remained Counter 의 값을 새로이 설정합니다.	**337**
int	**COMILX_MC_GetMotionStatus (HANDLE hDevice, int nChannel)** 현재 모션의 동작 상태를 반환합니다.	**338**
int	**COMILX_MC_GetMioStatus(HANDLE hDevice, int nChannel)** 현재 모션과 관련된 여러가지 I/O 상태를 반환합니다.	**339**

모션제어 (I/O(입출력) 환경설정) Page

void	**COMILX_MC_SetFilterLogic(HANDLE hDevice, int nChannel, BOOL bEnable)** 입력 신호에 필터로직을 적용할 지를 설정합니다. 필터 로직이 적용되는 I/O 신호 및 펄스폭은 아래의 참고 항목을 참조하십시오.	**344**
void	**COMILX_MC_SetMioCfgALM (HANDLE hDevice, int nChannel, int nAlarmLogic, int nAlarmMode)** Alarm 신호에 대한 입력모드를 설정합니다.	**345**
void	**COMILX_MC_GetMioCfgALM (HANDLE hDevice, int nChannel, int *pAlarmLogic, int *pAlarmMode)** 현재 설정된 Alarm 신호에 대한 입력모드를 읽어옵니다.	**346**
void	**COMILX_MC_SetMioCfgCLR(HANDLE hDevice, int nChannel, int CntrSel, int nSignalType)** CLR(카운터 클리어) 입력 신호에 대한 환경을 설정합니다.	**347**
void	**COMILX_MC_GetMioCfgCLR (HANDLE hDevice, int nChannel, int *pCntrSel, int *pnSignalType)** CLR 입력 신호에 대한 환경 설정값을 읽어들입니다.	**348**
void	**COMILX_MC_SetMioCfgDR(HANDLE hDevice, int nChannel, int nLogic)** DR 입력 신호에 대한 환경을 설정합니다.	**349**
void	**COMILX_MC_GetMioCfgDR (HANDLE hDevice, int nChannel, int *pnLogic)** 현재 설정된 DR 신호에 대한 환경 설정값을 읽어옵니다.	**350**
void	**COMILX_MC_SetMioCfgEL (HANDLE hDevice, int nChannel, int nElMode)** EL 신호에 대한 모드를 설정합니다.	**351**

void	**COMILX_MC_GetMioCfgEL (HANDLE hDevice, int nChannel, int *pElMode)** EL 신호에 대한 모드 설정값을 읽어옵니다.	**352**
void	**COMILX_MC_SetMioCfgERC (HANDLE hDevice, int nChannel, int nErcLogic, int nErcOnTime)** ERC 신호의 입력 로직(Logic) 및 ON time을 설정합니다.	**353**
void	**COMILX_MC_GetMioCfgERC (HANDLE hDevice, int nChannel, int *pErcLogic, int *pErcOnTime)** ERC 신호의 입력 로직(Logic) 및 ON time에 대한 설정값을 읽어옵니다.	**354**
void	**COMILX_MC_SetMioCfgINP (HANDLE hDevice, int nChannel, BOOL bInpEnable, int nInpLogic)** ERC 신호의 입력 로직(Logic) 및 ON time을 설정합니다.	**355**
void	**COMILX_MC_GetMioCfgINP (HANDLE hDevice, int nChannel, BOOL *pInpEnable, int *pInpLogic)** INP 기능 및 신호에 대한 환경 설정값을 읽어옵니다.	**356**
void	**COMILX_MC_SetMioCfgSD (HANDLE hDevice, int nChannel, BOOL bSdEnable, int nSdLogic, int nSdLatch, int nSdMode)** SD 신호에 대한 환경을 설정합니다. SD 신호는 Deceleration 시작점을 외부에서 입력하는 신호입니다.	**357**
void	**COMILX_MC_GetMioCfgSD (HANDLE hDevice, int nChannel, BOOL *pSdEnable, int *pSdLogic, int *pSdLatch, int *pSdMode)** SD 신호에 대한 환경 설정값을 읽어옵니다.	**358**
void	**COMILX_MC_SetSoftLimit (HANDLE hDevice, int nChannel, double fLimitP, double fLimitN)** 소프트웨어적인 Limit를 설정합니다.	**360**
void	**COMILX_MC_EnableSoftLimit (HANDLE hDevice, int nChannel)** 소프트웨어적인 Limit의 적용을 Enable시킵니다.	**361**
void	**COMILX_MC_DisableSoftLimit (HANDLE hDevice, int nChannel)** 소프트웨어적인 Limit의 적용을 Disable시킵니다.	**362**
void	**COMILX_MC_SetErrorCompare (HANDLE hDevice, int nChannel, double fTol, int bEnable)** 명령 펄스(Command Pulse)와 Feedback 펄스간의 에러를 체크하는 기능을 설정합니다.	**363**
void	**COMILX_MC_SetGeneralCompare (HANDLE hDevice, int nChannel, int nCmpSrc, int nCmpMethod, int nCmpAction, double fData)** General Comparator의 비교 대상, 비교값 등을 설정합니다.	**364**

모션제어 (인터럽트 관련 함수) Page

void	**COMILX_MC_MaskInterrupt (HANDLE hDevice, int nChannel, long dwMask)** 어떠한 조건의 인터럽트를 받아들일 것인지를 설정합니다.	**367**

void	**COMILX_MC_EnableInterrupt (HANDLE hDevice, HANDLE hEvent)** 인터럽트 이벤트 통지 기능을 Enable 시킵니다.	**369**
void	**COMILX_MC_DisableInterrupt (HANDLE hDevice)** 인터럽트 이벤트 통지 기능을 Disable 시킵니다.	**370**
BOOL	**COMILX_MC_GetAxisIntState (HANDLE hDevice, int nChannel)** 각 축에 대하여 인터럽트가 발생하였는지를 알려주는 함수입니다.	**371**
void	**COMILX_MC_GetIntStatus (HANDLE hDevice, int nChannel, long *pErrorStatus, long *pEventStatus)** 각 축의 인터럽트가 발생한 원인을 알려주는 함수입니다.	**372**