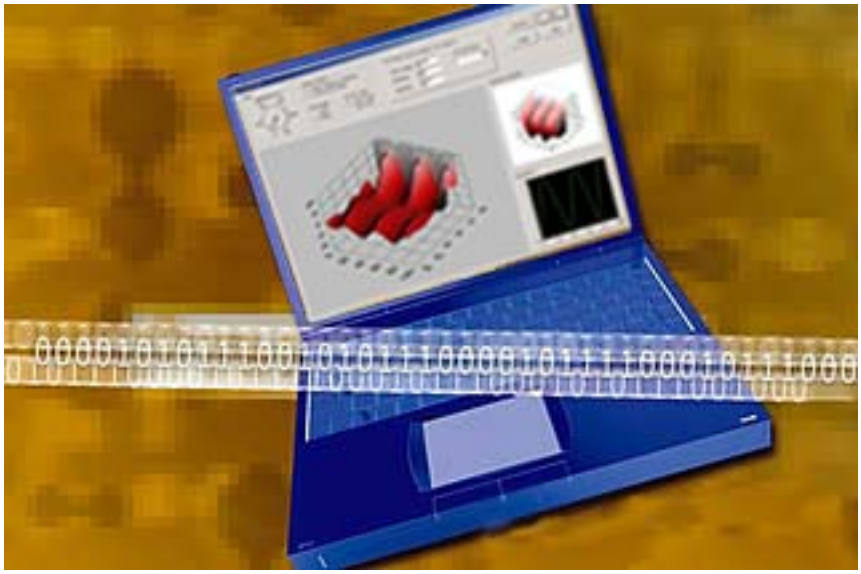
COMIZOA DAQ System (LX 시리즈)
소프트웨어 매뉴얼



COMIZOA

COMputer Innovation
is Zoomed by Our Affection!

2001년 11월 20일 중판 인쇄

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


COMIZOA

CHAPTER 1

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.

CHAPTER 1 장치 제어기 및 소프트웨어 설치

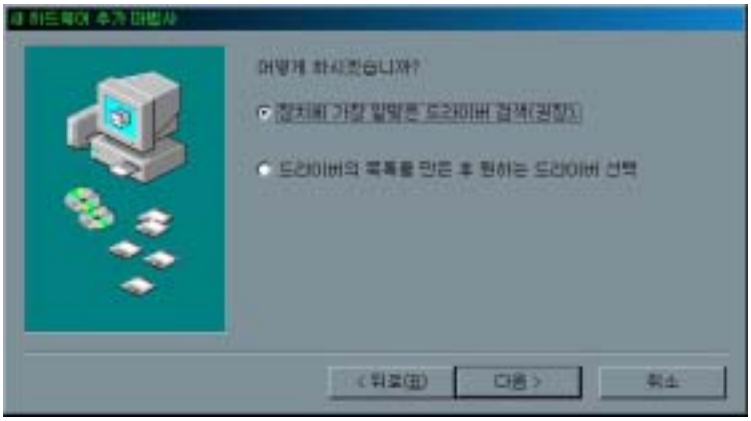
본 장에서는 COMIDAS 장치 제어기를 설치하는 방법을 설명합니다. 사용자는 제어기를 설치하셔야 윈도우에서 COMIDAS 디바이스를 사용하실 수 있습니다(도스에서는 제어기를 설치할 필요가 없습니다). 모든 종류의 COMIDAS 디바이스는 그 설치 방법이 동일합니다.

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 1-1]과 같이 PCI Card 를 찾았다는 화면을 나타냅니다. 제공된 설치 CD 를 CDROM 드라이브에 삽입하신 후 '다음' 버튼을 클릭하십시오.

[그림 1-1] 디바이스 제어기 설치 화면 1

[그림 1-2]와 같은 화면이 나타나면, '장치에 가장 알맞은 드라이버 검색(권장)'을 선택한 후 '다음' 버튼을 클릭하십시오.



[그림 1-2] 디바이스 제어기 설치 화면 2

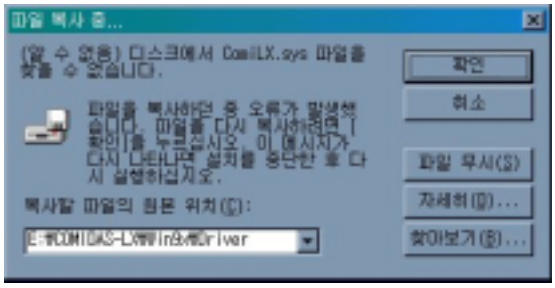
[그림 1-3]과 같은 화면이 나타나면, '검색할 위치 지정(L):' 항목에 "E:WCOMIDAS-LXWWin9xWDriver"를 입력한 후 '다음' 버튼을 클릭하십시오. 이때 "E:W"는 CDROM 드라이브를 의미합니다. 이미 소프트웨어 설치를 한 경우에는 설치된 디렉토리의 하위 디렉토리인 WindowWDriver 디렉토리를 지정하여도 됩니다.

[그림 1-3] 디바이스 제어기 설치 화면 3

[그림 1-4]와 같이 COMIDAS 디바이스 장치가 검색되었다는 화면이 나타나면, '다음' 버튼을 클릭하십시오. 그러면 윈도우는 필요한 파일들을 설치합니다.

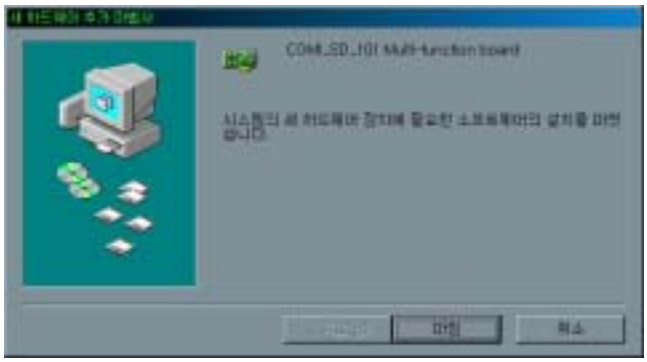
[그림 1-4] 디바이스 제어기 설치 화면 4

경우에 따라서는 [그림 1-5]와 같이 파일을 찾을 수 없다는 메시지가 나타날 수 있습니다. 이 때에는 무시하고 그냥 “확인”버튼을 누르면 설치를 계속 진행합니다.



[그림 1-5] 파일을 찾을 수 없다는 오류 메시지

필요한 드라이버 파일의 설치가 완료되면 [그림 1-6]과 같은 화면이 나타나면 ‘마침’ 버튼을 클릭하여 제어기 설치를 마칩니다.



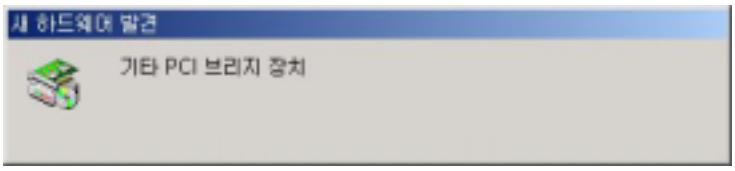
[그림 1-6] 디바이스 제어기 설치 화면 5

제어기 설치를 마친 후 제어판의 시스템 정보를 실행하면 [그림 1-7]과 같이 COMIDAS 디바이스가 설치된 것을 확인할 수 있습니다.

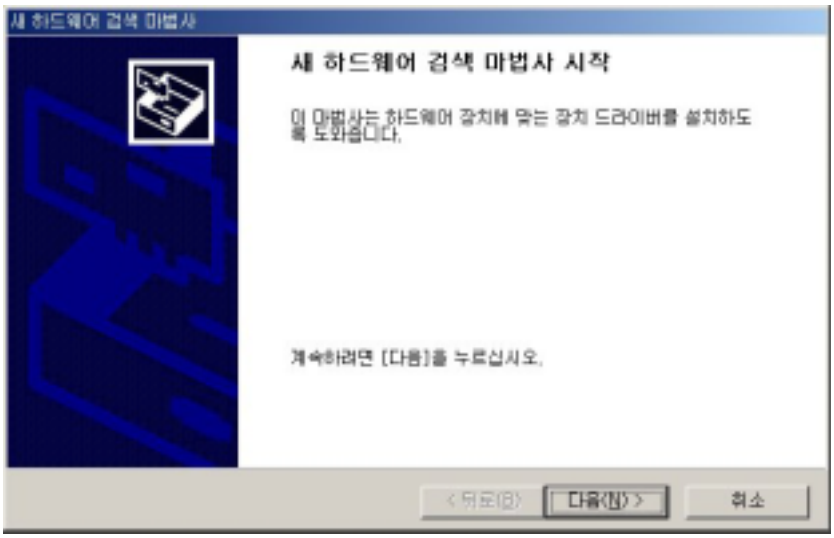
[그림 1-7] 시스템 등록 정보에 디바이스 설치 확인

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 1-8]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.

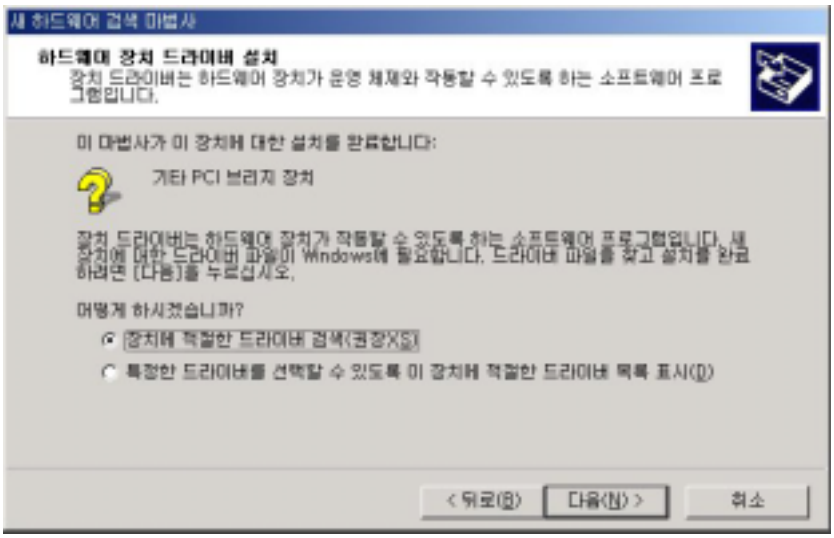


[그림 1-8] Win2000 디바이스 제어기 설치 화면 1



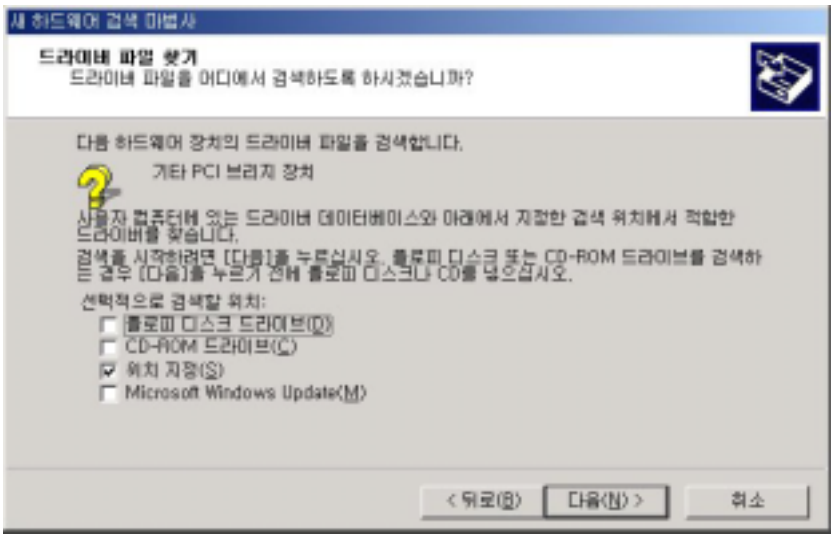
[그림 1-9] Win2000 디바이스 제어기 설치 화면 2

[그림 1-10]과 같은 화면이 나타나면, '장치에 적절한 드라이버 검색(권장)' 을 선택한 후 '다음' 버튼을 클릭하십시오.



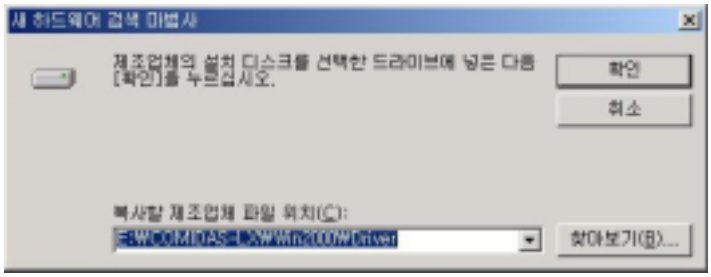
[그림 1-10] Win2000 디바이스 제어기 설치 화면 3

[그림 1-11]과 같은 화면이 나타나면, '위치 지정'을 선택한 후 '다음' 버튼을 클릭하십시오.



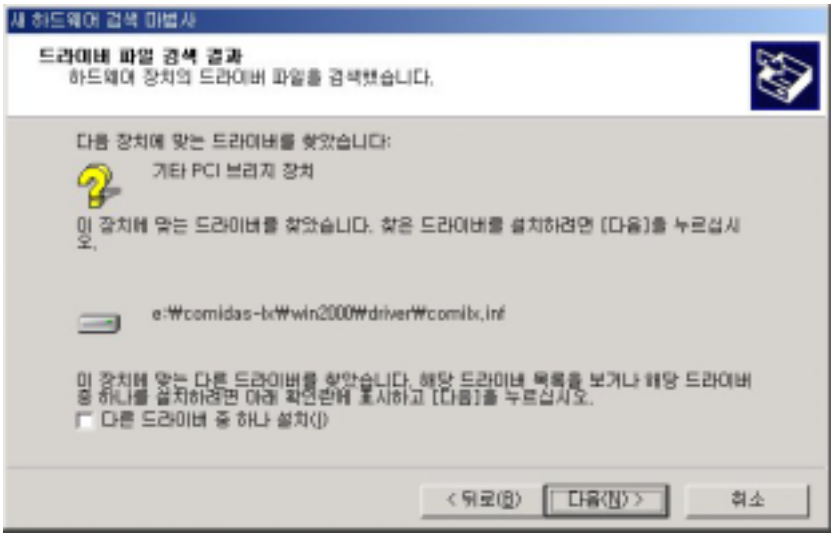
[그림 1-11] Win2000 디바이스 제어기 설치 화면 4

[그림 1-12]와 같은 화면이 나타나면, '복사할 제조업체 파일 위치(C):' 항목에 "E:WCOMIDAS-LXWWin2000WDriver"를 입력한 후 '다음' 버튼을 클릭하십시오. 이 때 "E:W"는 CDROM 드라이브를 의미합니다. 이미 소프트웨어 설치를 한 경우에는 설치된 디렉토리의 하위 디렉토리인 WindowWDriver 디렉토리를 지정하여도 됩니다.



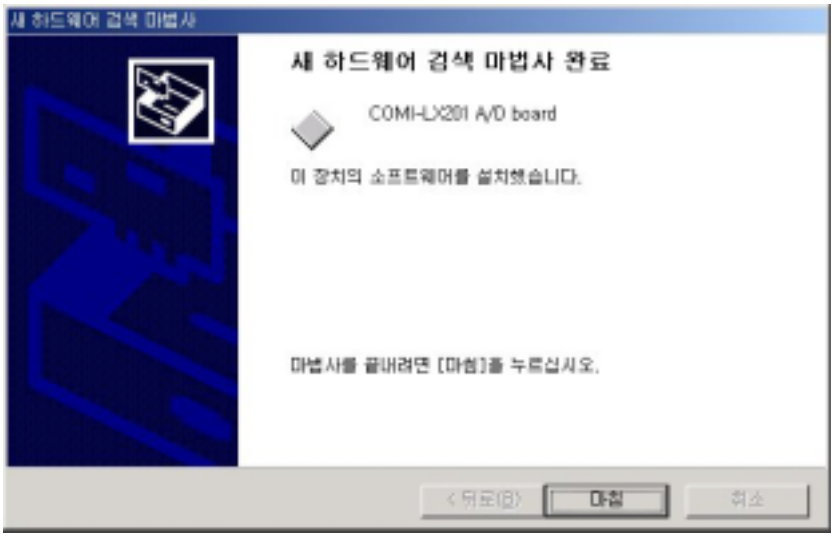
[그림 1-12] Win2000 디바이스 제어기 설치 화면 5

[그림 1-13]과 같이 기타 PCI 브리지 장치(또는 COMIDAS 모델명)가 검색되었다는 화면이 나타나면, '다음' 버튼을 클릭하십시오. 그러면 윈도우는 필요한 파일들을 설치합니다.



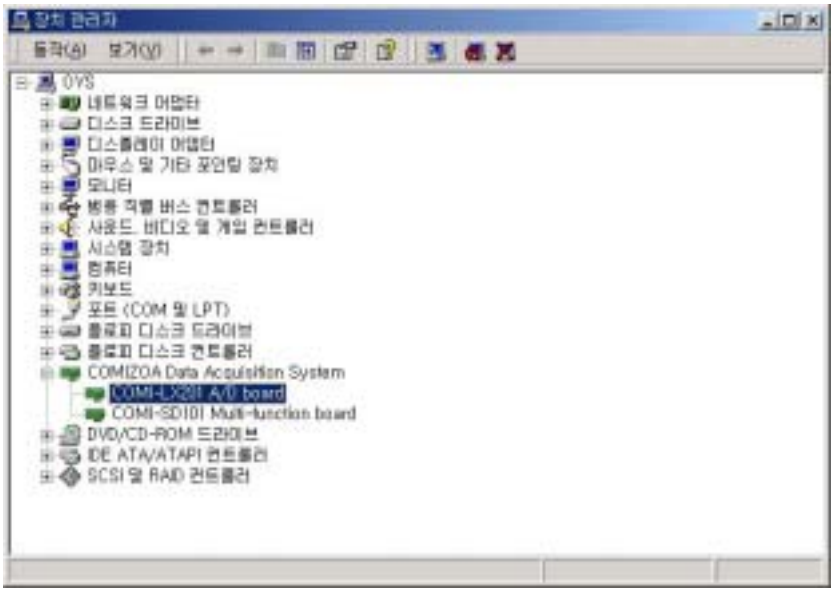
[그림 1-13] Win2000 디바이스 제어기 설치 화면 6

[그림 1-14]와 같은 화면이 나타나면 '마침' 버튼을 클릭하여 제어기 설치를 마칩니다.



[그림 1-14] Win2000 디바이스 제어기 설치 화면 7

제어기 설치를 마친 후 장치 관리자를 실행하면 [그림 1-15]와 같이 COMIDAS 디바이스가 설치된 것을 확인할 수 있습니다.



[그림 1-15] 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 1-16]과 같은 화면이 나타납니다.

[그림 1-16] 설치 CD 의 자동실행 초기화면

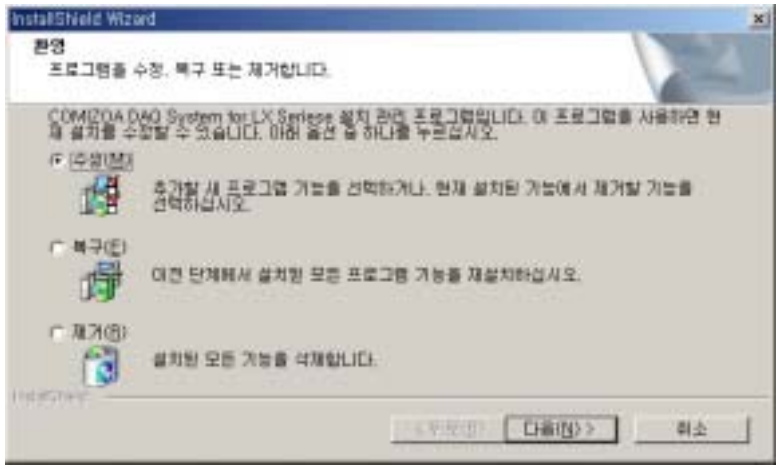
[그림 1-16]화면의 상단 메뉴에서 Install 메뉴를 선택하면 [그림 1-17]과 같은 화면이 나타납니다. Win98 이나 WinME 사용자는 “COMIDAS-LX for W9x” 버튼을 Win2000 사용자는 “COMIDAS-LX for W2000”버튼을 클릭하면 COMIDAS-LX 소프트웨어 설치 프로그램이 실행됩니다.

[그림 1-17] 설치 CD 의 자동실행 프로그램의 Install 메뉴 화면

[그림 1-18]과 같은 화면이 나타나면 “다음” 버튼을 클릭하여 프로그램 설치를 시작합니다.

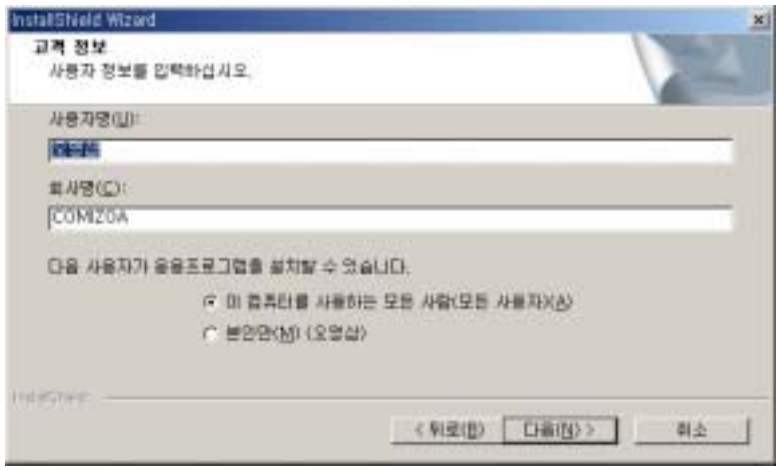
[그림 1-18] 프로그램 설치 시작 화면

만일 COMIDAS-LX 프로그램이 이미 설치되어 있으면 [그림 1-18]화면 대신 [그림 1-19] 화면이 나타나게 됩니다. [그림 1-19] 화면에서 프로그램을 다시 설치하려면 “복구” 옵션을 선택한 후 “다음” 버튼을 클릭하십시오.



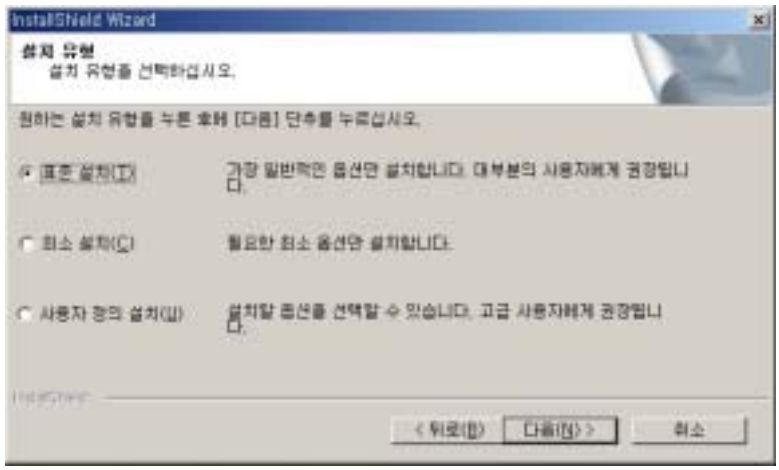
[그림 1-19] 이전에 프로그램이 설치되어 있는 경우의 설치 시작 화면

[그림 1-20]과 같은 화면이 나타나면 고객 정보를 입력한 후 “다음” 버튼을 클릭하십시오.



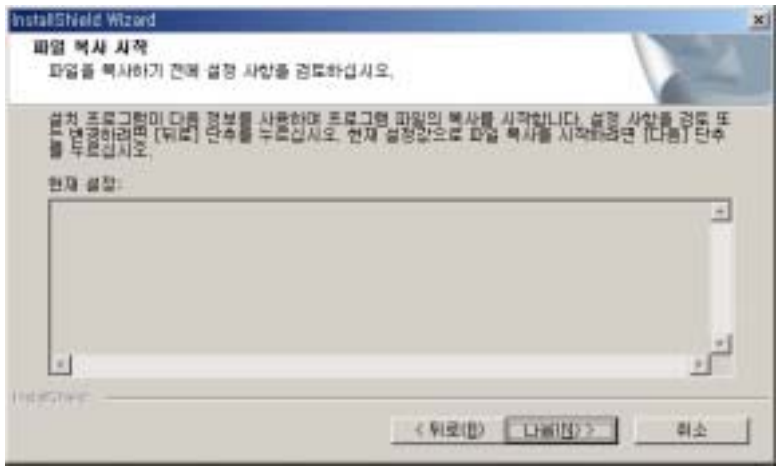
[그림 1-20] 고객 정보 입력 화면

[그림 1-21]과 같은 화면이 나타나면 “표준 설치(T)” 옵션을 선택한 후 “다음” 버튼을 클릭하십시오.



[그림 1-21] 설치 유형 선택 화면

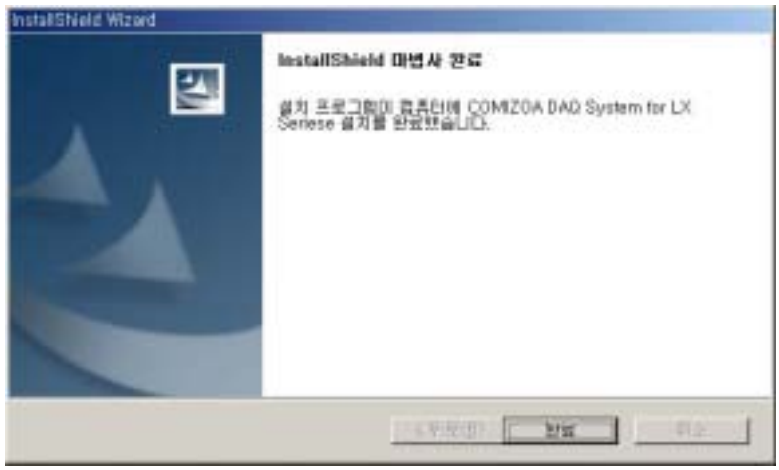
[그림 1-22]와 같은 화면이 나타나면 “다음” 버튼을 클릭하십시오.



[그림 1-22] 현재 설정 확인 화면

[그림 1-22]화면에서 “다음” 버튼을 클릭하면 필요한 파일 복사가 진행됩니다. 프

로그램 설치가 완료되면 [그림 1-23]과 같은 화면이 나타납니다. 이상이 없으면 “완료” 버튼을 클릭하여 프로그램 설치를 완료하십시오.



[그림 1-23] 프로그램 설치 완료 화면

프로그램은 사용자가 설치 시에 변경하지 않았다면 기본적으로 C:₩Program Files₩COMIZOA₩COMIDAS-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 2

COMI-XMaster 유틸리티 프로그램

본 장에서는 COMI-XMaster 유틸리티 프로그램을 설명합니다. COMI-XMaster 유틸리티 프로그램은 COMI-XMaster 는 COMI-LX 시리즈 DAQ 보드 전용 유틸리티 프로그램으로써 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.

CHAPTER 2 COMI-XMaster 유틸리티 프로그램

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

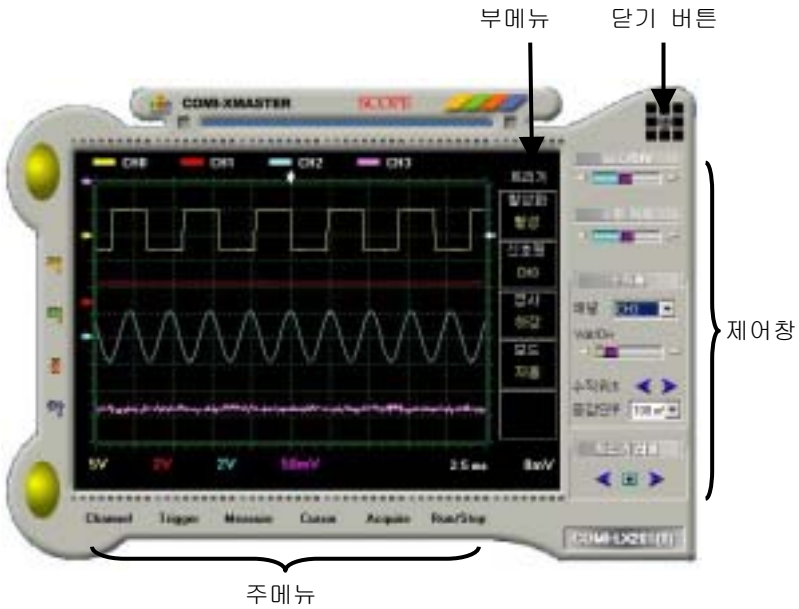
윈도우의 시작메뉴에서 COMIDAS-LXWCOMI-XMaster Utility for LX Seriese 메뉴를 선택하면 [그림 2-1]과 같은 화면이 나타납니다.

[그림 2-1] COMI-XMaster 주 화면

[그림 2-1]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.

2.1 Scope

Scope 는 COMI-LX10x 디바이스와 COMI-LX20x 디바이스들에 적용 가능한 프로그램 모듈로써 널리 사용되고 있는 오실로스코프의 기능을 제공하는 프로그램입니다. 이 프로그램은 오실로스코프의 거의 모든 기능을 제공할 뿐 아니라, 데이터를 텍스트 파일로 저장하거나 그래프를 BMP 파일로 저장하거나 프린터로 인쇄할 수 있는 기능을 제공합니다.



[그림 2-2] Scope 프로그램 모듈 화면

Scope 프로그램 모듈은 [그림 2-2]와 같이 구성됩니다. Scope 프로그램의 화면은 다음과 같이 크게 4 개의 영역으로 구분될 수 있습니다.

◈ 그래프 영역

입력된 신호를 그래프로 표시해주는 영역입니다.

◈ 주메뉴

Scope 의 주메뉴를 선택하는 영역입니다. 주메뉴 영역에 있는 버튼을 클릭하면 부

메뉴 영역은 선택된 주메뉴에 해당하는 부메뉴 항목들로 변경됩니다.

◈ 부메뉴

각 주메뉴에 따른 하부 명령 또는 상태 표시를 할 수 있도록 하는 영역입니다. 부메뉴의 각 항목은 항목의 특성에 따라 명령 버튼으로 사용되거나 상태 표시 항목으로 사용됩니다.

◈ 제어창

그래프의 축 확대/축소, 축 이동, 트리거 레벨 조정 등을 수행하는 영역입니다.

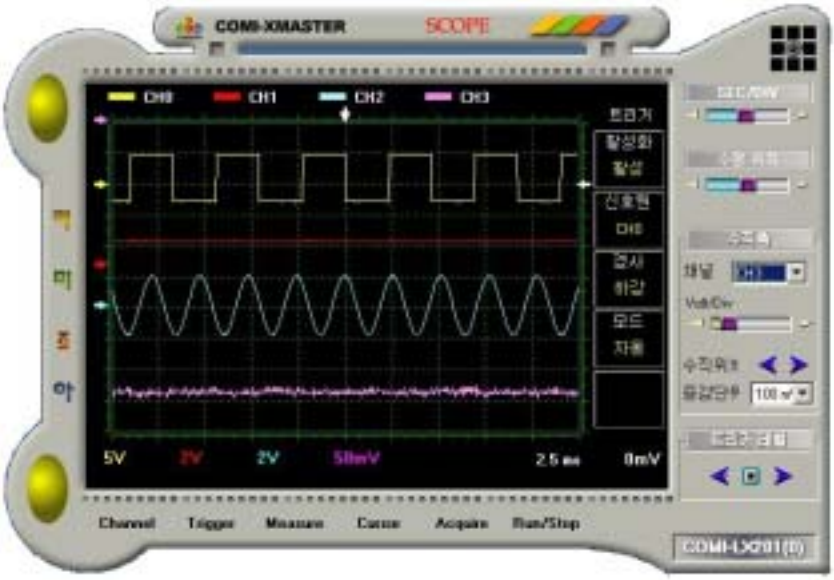
2.1.1 Channel 메뉴

스코프 화면에 보여줄 A/D 채널을 선택합니다. 최대 4 개의 채널 까지 선택할 수 있습니다. 주메뉴에서 “Channel” 메뉴를 선택하면 [그림 2-3]과 같이 대화상자가 나타납니다. 이 대화상자에서 A/D 채널을 선택합니다. None 으로 선택하면 해당 TRACE는 스코프 화면에 표시되지 않습니다.

[그림 2-3] Scope 채널 설정 대화상자

2.1.2 Trigger 메뉴

본 프로그램은 오실로스코프의 트리거(Trigger)기능과 동일한 트리거 기능을 제공합니다. 트리거 기능을 사용하면 주기적인 신호를 계측할 때 파형이 그래프에서 흐르지 않고 정지된 것과 같은 효과를 나타낼 수 있어서 신호분석을 용이하게 하여줍니다. 주메뉴에서 “Trigger” 메뉴를 선택하면 [그림 2-4]화면과 같이 부메뉴가 트리거 메뉴로 나타납니다. 단, 트리거 레벨의 조정은 제어창의 “트리거레벨” 항목을 이용하여 설정합니다.



[그림 2-4] Trigger 메뉴를 선택하였을 때의 화면

트리거 부메뉴의 각 항목은 다음과 같습니다.

◈ 활성화

트리거 기능을 활성화(Enable) 또는 비활성화(Disable)하는 메뉴입니다. 이 부메뉴를 클릭하면 활성화/비활성화가 토글됩니다.

◈ 신호원

트리거 소스 채널을 설정합니다. 이 부메뉴를 클릭하면 소스 채널이 변경됩니다.

◈ 경사

트리거 포인트를 결정하는 경사를 설정합니다. 이 부메뉴를 클릭하면 “상승”과 “하강”이 토글됩니다.

-. 상승 : 상승에지(Rising Edge)에서 트리거 포인트를 잡습니다.

-. 하강 : 하강에지(Falling Edge)에서 트리거 포인트를 잡습니다.

◈ 모드

트리거 모드를 설정합니다.

-. 자동 : 트리거가 발생하지 않아도 화면을 갱신합니다.

-. 보통 : 트리거가 발생한 경우에만 화면을 갱신합니다.

-. 단발 : 트리거가 발생하기 전까지 화면을 갱신하다가 트리거가 발생하면 화면 갱신을 멈춥니다.

2.1.3 Measure 메뉴

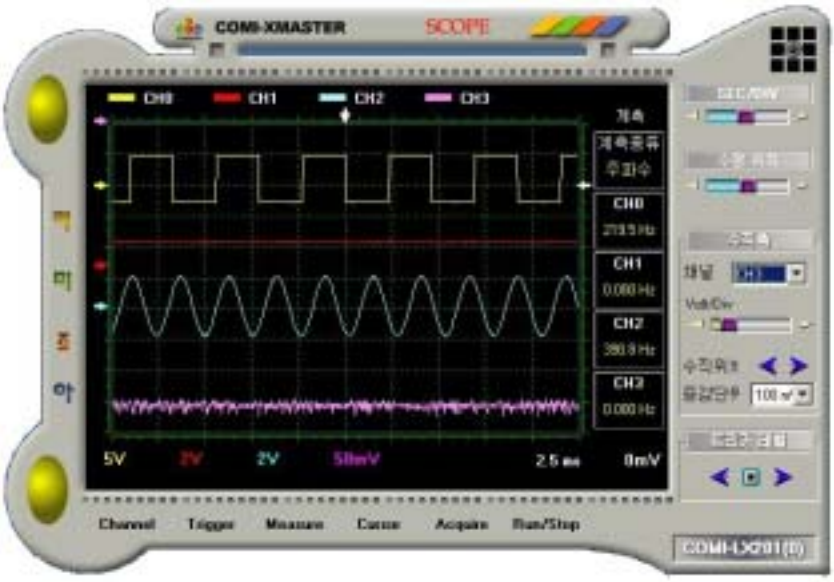
본 프로그램은 각 신호의 주파수, 실효치(RMS), 평균치, 첨두치(VPP) 등의 특성을 쉽게 관찰할 수 있도록 하는 기능을 제공합니다. 주메뉴에서 “Measure” 메뉴를 선택하면 [그림 2-5]화면과 같이 부메뉴가 계측 메뉴로 나타납니다. “계측종류” 부메뉴를 이용하여 다음과 같이 계측하고자 하는 신호의 특성을 선택할 수 있습니다.

-. 주파수 : 각 입력 신호의 주파수를 계측하여 나타냅니다.

-. 실효치 : 각 입력 신호의 실효치(Root Mean Square)를 계측하여 나타냅니다.

-. 평균치 : 각 입력 신호의 평균치를 계측하여 나타냅니다.

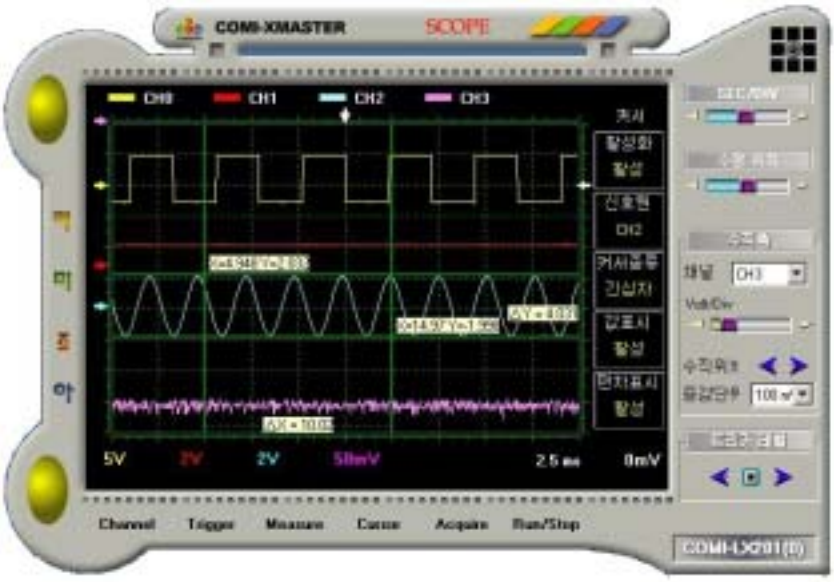
-. 첨두치 : 각 입력 신호의 첨두치(VPP)를 계측하여 나타냅니다.



[그림 2-5] Measure 메뉴를 선택하였을 때의 화면

2.1.4 Cursor 메뉴

본 프로그램은 그래프상에서 각 신호의 값, 전압차, 시간차 등을 쉽게 알아볼 수 있도록 하는 커서를 제공합니다. 주메뉴에서 “Cursor” 메뉴를 선택하면 [그림 2-6] 화면과 같이 그래프에 두개의 커서가 나타나면서 부메뉴가 커서 메뉴로 전환됩니다. 사용자는 두개의 커서를 이용하여 그래프상에서 각 데이터의 값을 확인할 수 있을 뿐 아니라, 두 커서간의 시간차(△X)와 전압차(△Y) 또한 한눈에 확인할 수 있습니다.



[그림 2-6] Cursor 메뉴를 선택하였을 때의 화면

커서 부메뉴의 각 항목은 다음과 같습니다.

◈ 활성화

커서를 활성화 또는 비활성화하는 메뉴입니다. 이 부메뉴를 클릭하면 활성화/비활성화가 토글됩니다.

◈ 신호원

커서를 이용하여 데이터를 확인할 채널을 설정합니다. 각 신호의 전압축 스케일이 다를 수 있으므로 커서의 신호원을 정확히 지정하여야 정확한 데이터값으로 표시됩니다.

◈ 커서종류

커서의 모양을 설정합니다.

◈ 값표시

커서가 가리키는 데이터의 값을 화면에 표시할 지를 설정합니다.

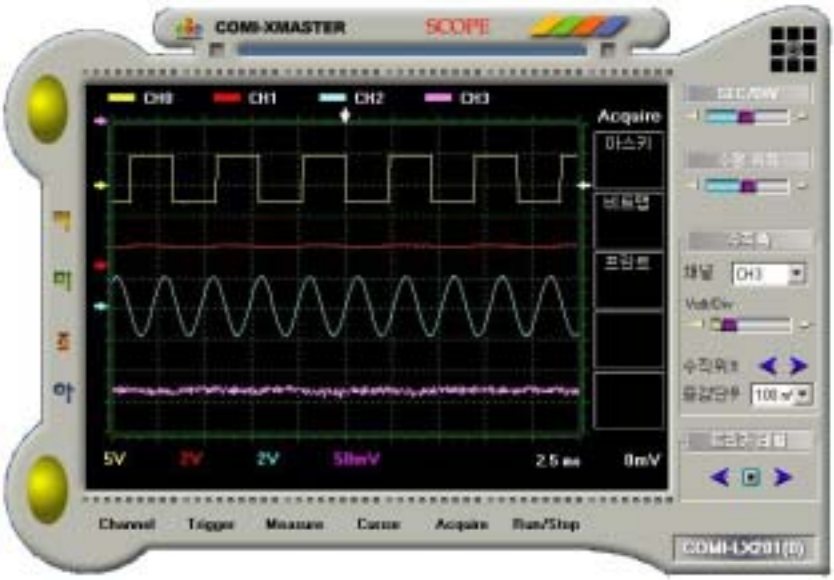
◈ 편차표시

두 커서간의 전압차 및 시간차를 화면에 표시할 지를 설정합니다.

2.1.5 Acquire 메뉴

“Acquire” 메뉴는 데이터를 텍스트 파일로 저장하거나 그래프를 BMP 파일로 저장 또는 인쇄하는 기능을 제공하는 메뉴입니다.

주메뉴에서 “Acquire” 메뉴를 선택하면 [그림 2-7]화면과 같이 부메뉴가 커서 메뉴로 전환됩니다.



[그림 2-7] Acquire 메뉴를 선택하였을 때의 화면

Acquire 부메뉴의 각 항목은 다음과 같습니다.

◈ 아스키

이 부메뉴를 클릭하면 그래프에 있는 각 채널의 데이터를 텍스트 파일로 저장합니다.

◈ 비트맵

이 부메뉴를 클릭하면 그래프를 BMP 파일로 저장합니다.

◈ 프린트

이 부메뉴를 클릭하면 그래프를 인쇄합니다.

2.1.6 그래프 수평축 조정

제어창 영역에서 “SEC/DIV” 항목과 “수평 위치” 항목을 이용하여 그래프의 수평축을 조정할 수 있습니다.

◈ SEC/DIV

이 항목은 그리드(Grid)의 시간 간격을 설정합니다. 이 항목을 이용하면 그래프의 시간축을 확대/축소할 수 있습니다. 시간축은 각 채널에 동일하게 적용됩니다. 설정된 그리드의 시간 간격 수치는 그래프의 하단에 표시됩니다.

◈ 수평위치

이 항목은 트리거 포인트의 수평축 상 위치를 설정합니다. 트리거 기능이 활성화된 경우 이 항목을 이용하면 신호를 수평축 상에서 스크롤해볼 수 있습니다.

2.1.7 그래프 수직축 조정

제어창 영역에서 “수직축” 항목에 있는 여러 가지 컨트롤들을 이용하여 수직축을 조정할 수 있습니다.

◈ 채널

이 항목은 수직축 제어의 대상이 되는 입력 채널을 선택합니다.

◈ Volt/Div

이 항목은 수직축 그리드(Grid)의 전압 간격을 설정합니다. 이 항목을 이용하면 입력신호를 전압축 기준으로 확대/축소해볼 수 있습니다.

◈ 수직위치

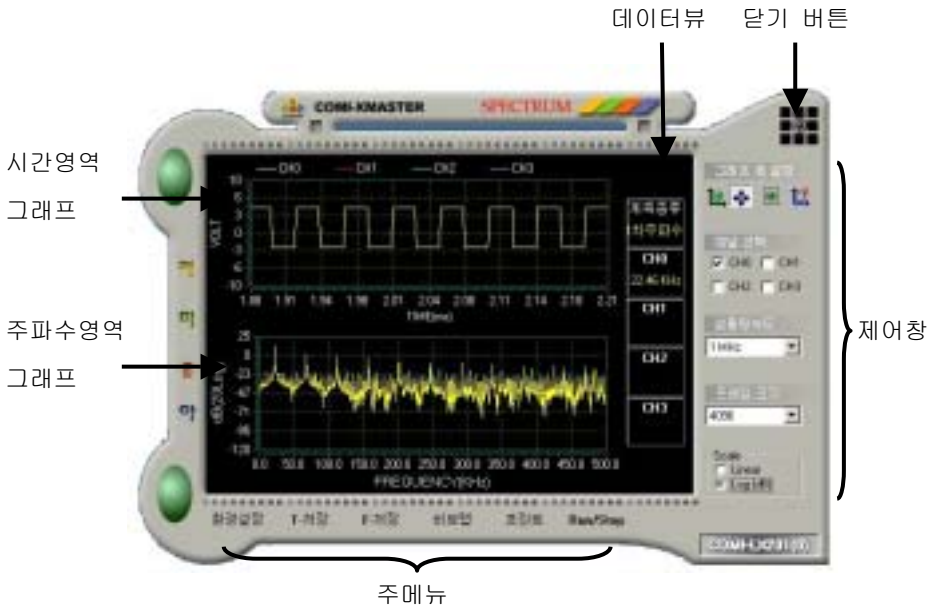
이 항목은 그래프에서 입력신호가 표시되는 전압축 상의 위치를 조정합니다. 왼쪽 화살표 버튼을 클릭하면 신호는 아래로 이동하고 오른쪽 화살표 버튼을 클릭하면 신호는 위로 이동합니다. 이때 이동량은 “Volt/Div” 항목과 “증감단위” 항목에 따라 결정됩니다. 각 입력신호의 전압축 원점은 그래프의 좌측에 있는 화살표로 표시됩니다.

◈ 증감단위

이 항목은 “수직위치” 항목을 이용하여 신호를 전압축 상에서 이동할 때 이동량을 결정합니다.

2.2 Spectrum Analyzer

Spectrum Analyzer 는 COMI-LX20x 디바이스들에 적용 가능한 프로그램 모듈로써 아날로그 입력신호를 주파수 분석할 수 있도록 하는 프로그램입니다.



[그림 2-8] Spectrum Analyzer 프로그램 모듈 화면

Spectrum Analyzer 프로그램 모듈의 화면은 [그림 2-8]과 구성됩니다.

시간영역 그래프

아날로그 입력 신호를 시간영역(Time-domain) 그래프로 표시하여줍니다. 이 그래프에서 X 축은 시간 축이 됩니다.

주파수영역 그래프

아날로그 입력 신호를 FFT 변환하여 주파수영역(Frequency-domain) 그래프로 표시하여 줍니다. 이 그래프에서 X 축은 주파수 축이 됩니다.

주메뉴

Spectrum Analyzer 의 여러 가지 메뉴 버튼들을 모아놓은 영역입니다.

데이터뷰

각 채널의 1차 주파수 값을 표시하여 줍니다.

제어창

그래프에 대한 옵션을 설정하거나 채널에 대한 옵션을 설정합니다.

2.2.1 환경설정

주메뉴에서 “환경설정” 버튼을 클릭하면 [그림 2-9]와 같은 대화상자가 나타납니다. 이 환경설정은 Trigger Event 에 대한 설정과 아날로그 입력 채널의 범위를 설정합니다.

[그림 2-9] Spectrum Analyzer 환경설정 대화상자

Trigger Event 설정

Trigger Event 는 A/D 데이터를 획득하는 시점을 결정합니다. Spectrum Analyzer 프로그램은 제어창의 “프레임 크기” 항목에서 지정한 수의 A/D 데이터를 획득하여 FFT 처리를 한 후 화면에 보여줍니다. 이 때 Trigger Event 는 지정된 수의

A/D 데이터를 획득하기 시작하는 시점에 대한 조건을 설정합니다.

Trigger Event 설정의 각 항목에 대한 자세한 사항은 “3-4-3 LX20-시리즈 디바이스 전용 A/D 스캔 함수” 단원을 참조하십시오.

Analog Input Range

각 아날로그 입력 채널에 대한 입력 범위를 설정합니다.

2.2.2 데이터 저장 및 인쇄

시간영역 데이터 저장

주메뉴의 “T-저장” 버튼을 클릭하면 시간영역 그래프에 있는 데이터를 텍스트 파일로 저장할 수 있습니다.

시간영역 데이터 저장

주메뉴의 “F-저장” 버튼을 클릭하면 주파수영역 그래프에 있는 데이터를 텍스트 파일로 저장할 수 있습니다.

그래프를 BMP 로 저장

주메뉴의 “비트맵” 버튼을 클릭하면 각 그래프를 비트맵 파일로 저장할 수 있습니다.

그래프 프린트

주메뉴의 “프린트” 버튼을 클릭하면 각 그래프를 인쇄할 수 있습니다.

2.2.3 그래프 제어

제어창의 “그래프 축 설정” 항목에 있는 컨트롤들을 사용하여 그래프의 각 축을 자유로이 확대 축소할 수 있습니다.

축 스크롤

축 스크롤 버튼()을 클릭한 상태에서 원하는 축을 마우스로 왼쪽클릭한 상태에서 스크롤하면 축이 스크롤됩니다.

축 확대/축소

축 확대/축소 버튼()을 클릭한 상태에서 원하는 축을 마우스로 왼쪽클릭한 상태에서 스크롤하면 축이 확대/축소됩니다.

영역 확대

영역 확대 버튼()을 클릭한 상태에서 그래프의 영역을 드래그하여 선택하면 선택된 영역이 확대되어 그래프에 나타납니다.

Auto Set

Auto Set 버튼()을 클릭하면 각 그래프의 축 설정이 기본값으로 복원됩니다. 이는 축을 확대/축소 또는 스크롤한 후 원래의 축 설정으로 복원하고자할 때 사용하는 기능입니다.

2.2.4 제어창의 기타 항목

채널 선택

그래프에 표시하고자 하는 채널을 선택합니다.

샘플링속도

아날로그 신호 샘플링 속도를 설정합니다.

프레임 크기

아날로그 신호 데이터 프레임 크기. 이 항목에서 설정한 수만큼의 데이터를 스캔하여 FFT 변환을 수행합니다.

Scale

주파수영역 그래프의 수직 축 Scale을 설정합니다.

2.3 Function Generator

Function Generator 프로그램 모듈은 COMI-LX10x 와 COMI-LX30x 시리즈 디바이스들에 적용 가능한 프로그램으로써 아날로그 출력 채널의 Waveform Generation 기능을 사용하여 널리 사용되고 있는 Function Generator 기능을 제공하는 프로그램입니다. 본 프로그램은 구형파, 삼각파, 정현파의 정규 파형외에 사용자가 임의로 구성한 주기 데이터를 출력할 수 있습니다.

[그림 2-10] Spectrum Analyzer 프로그램 모듈 화면

채널 설정

[그림 2-10]화면에서 “Channel” 항목은 아날로그 출력 채널을 설정합니다. 그림에서 “CH0”로 표시된 버튼을 클릭하면 채널이 토글됩니다.

Function 선택

[그림 2-10]화면에서 “Function”그룹의 버튼들을 이용하여 출력파형을 결정합니다. 출력 파형은 구형파, 삼각파, 정현파 그리고 사용자 정의 파형이 있습니다.

단, 사용자 정의 파형은 “User Data” 메뉴를 이용하여 데이터 파일을 먼저 입력하여야 합니다.

User Data

사용자 정의 파형을 출력하기 위해서는 먼저 사용자 정의 데이터를 입력하여야 합니다. 사용자 정의 데이터는 텍스트 파일로 입력하며 텍스트 파일의 구성은 출력하고자 하는 파형의 데이터를 1 열로 연속하여 써 주면 됩니다. 예를 들어 0 ~ 5 Volt 의 구형파를 출력하고자 한다면 다음과 같이 텍스트 파일에 적어주면 됩니다.

0

5

2.4 Digital In/Out

DIO 프로그램 모듈은 디지털 입출력 프로그램 모듈입니다. 메인 프로그램에서 DIO 버튼을 클릭하면 [그림 2-11]과 같은 화면이 나타납니다.

[그림 2-11] 디지털 입출력 프로그램 모듈 화면

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

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.

2.5 Motion Control

Motion Control 프로그램 모듈은 COMI-LX501 모션제어(Motion Control) 전용 보드의 기능을 테스트해볼 수 있도록하기 위하여 제작된 프로그램입니다. 메인 프로그램에서 Motion 버튼을 클릭하면 [그림 2-12]와 같은 화면이 나타납니다.

[그림 2-12] 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.

2.5.1 Motion Status 창

[그림 2-12] Motion Control 메인 화면에서 좌측 중앙에 있는 타원형의 버튼을 클릭하면 [그림 2-13]과 같이 Motion Status 를 감시할 수 있는 창이 확장되어 나타납니다.

[그림 2-13] 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 LED는 모션의 관련된 각 I/O 의 상태를 표시합니다.

2.5.2 Digital I/O 창

[그림 2-12] Motion Control 메인 화면에서 우측 중앙에 있는 타원형의 버튼을 클릭하면 [그림 2-14]와 같이 Digital Input/Output 을 감시 또는 제어할 수 있는 창이 나타납니다. COMI-LX501 모션제어 보드는 각각 16 채널씩의 Digital Input/Output 채널을 제공합니다.

[그림 2-14] Digital In/Out 창이 좌측에 확장되어 나타난 화면

2.5.3 Graph 창

[그림 2-12] Motion Control 메인 화면에서 하단의 우측에 있는 타원형의 버튼을 클릭하면 [그림 2-15]와 같이 위치 및 속도를 그래프로 표시해볼 수 있는 창이 나타납니다.

[그림 2-15] Graph 창이 하단에 확장되어 나타난 화면

Plot in motion

[그림 2-15]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 2-15]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 스크롤하면 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.

2.5.4 Pulse Input/Output 설정

[그림 2-12]화면에서 'Set In/Out' 버튼을 클릭하면 [그림 2-16]화면이 나타납니다. 이 화면에서 사용자는 Pulse 입/출력 신호의 환경을 설정할 수 있습니다.

[그림 2-16] Pulse Input/Output 설정 대화상자

[그림 2-16]화면에서 각 항목의 의미는 다음과 같습니다.

Encoder Input Mode

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

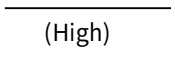
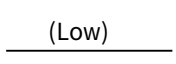
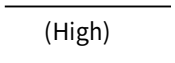
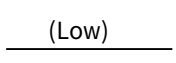
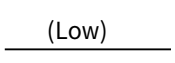
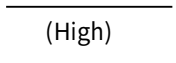
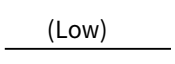
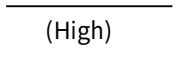
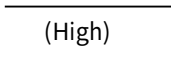
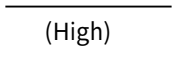
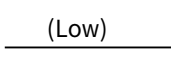
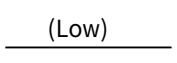
Item Value	Meaning
1X A/B	1X A/B (1 채배 엔코더 입력 모드)
2X A/B	2X A/B (2 채배 엔코더 입력 모드)
4X A/B	4X A/B (4 채배 엔코더 입력 모드)
CW/CCW	CW/CCW (A 펄스 - 카운트 증가, B 펄스 - 카운트 감소)

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)	

CHAPTER 3

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.

CHAPTER 3 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. 사용자는 각 함수 그룹의 설명 및 부록에서 제공되는 각 함수의 지원가능한 디바이스 리스트를 참조하여 각 디바이스에 맞는 함수들을 사용하시기 바랍니다.

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)프로그램으로써 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.

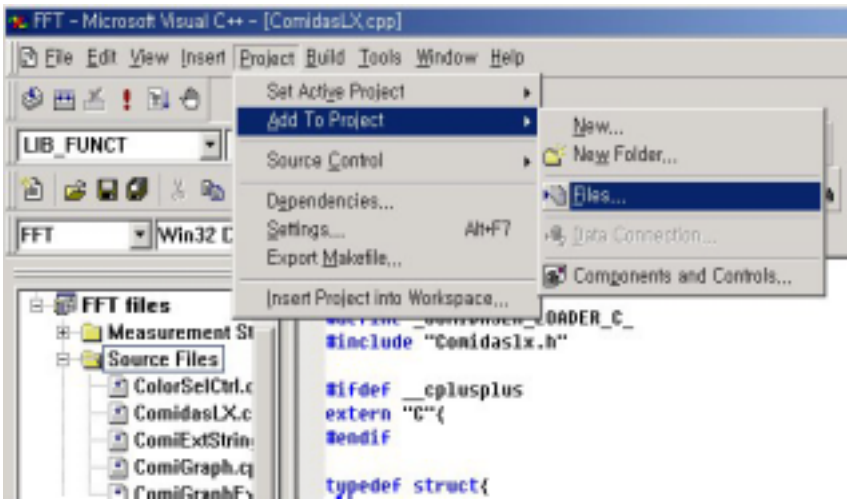
3.1 라이브러리 사용 방법

3.1.1 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 1-1]과 같이 Project ≫ Add To Project ≫ Files 메뉴를 선택하여 해당 파일들을 추가하면 됩니다.



[그림 1-1] 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”

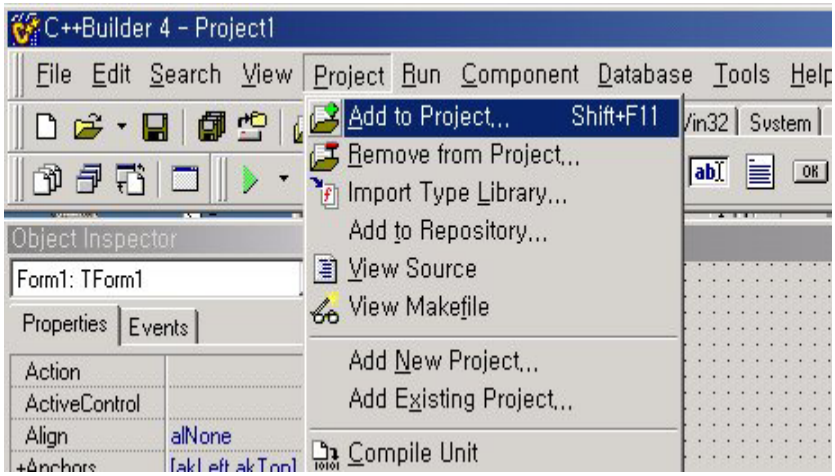
라이브러리 함수를 사용하고자 하는 소스 파일에 #include “ComidasLX.h” 구문을 추가합니다.

3.1.2 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 1-2]와 같이 Project ≫ Add To Project 메뉴를 선택하여 해당 파일들을 추가하면 됩니다.



[그림 1-2] C++ Builder 에서 프로젝트에 헤더 및 소스파일 추가하기

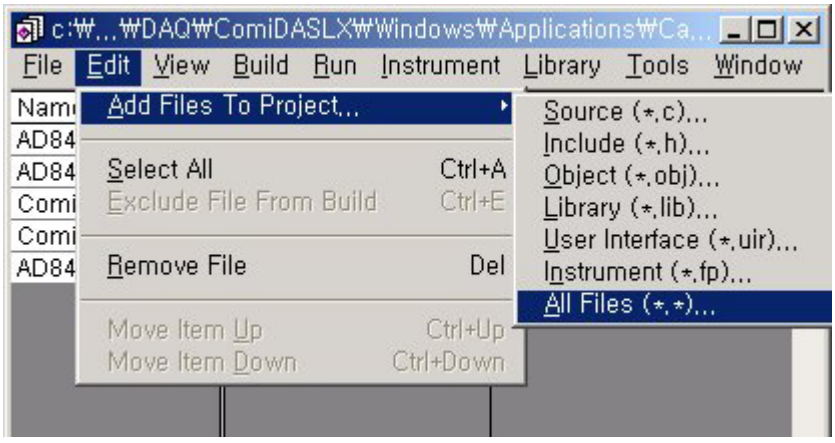
라이브러리 함수를 사용하고자 하는 소스 파일에 #include “ComidasLX.h” 구문을 추가합니다.

3.1.3 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 1-3]과 같이 Edit ≫ Add Files To Project ≫ All Files 메뉴를 선택하여 해당 파일들을 추가하면 됩니다.



[그림 1-3] 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).

3.2 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.

메쏘드명	각 보드별 적용 여부					
	LX10x	LX20x	LX301	LX401	LX402	LX501
COMILX_LoadDll	V	V	V	V	V	V
COMILX_UnloadDll	V	V	V	V	V	V
COMILX_LoadDevice	V	V	V	V	V	V
COMILX_UnloadDevice	V	V	V	V	V	V

[표 3-1] 라이브러리 및 디바이스 시작/종료 함수 리스트 및 각 보드별 지원 여부

▣ 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.

Return 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-LX101 Multi-function board 의 경우 COMI_LX101 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-CP101 보드가 장착되어 있다면 처음 장착된 보드의 instance 값은 0 이 되며, 두 번째 장착된 보드의 instance 값은 1 이 됩니다.

Return 값

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-LX201 디바이스를 이용하여 프로그램 할 때의 코드 예

```
HANDLE hDevice1, hDevice2;
if(!COMILX_LoadDll()){
```

```
printf("Comidas.dll load failure");
   exit(0);
}

hDevice1 = COMILX_LoadDevice (COMI_LX201, 0);
if(hDevice1 == INVALID_HANDLE_VALUE){
   printf("Can't load first COMI-LX201 device!");
   COMILX_UnloadDll();
   exit(0);
}

hDevice2 = COMILX_LoadDevice (COMI_LX201, 1);
if(hDevice2 == INVALID_HANDLE_VALUE){
   printf("Can't load second COMI-LX201 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_LX201, 0);
if(hDevice == INVALID_HANDLE_VALUE){
   printf("Can't load COMI-LX201 device!");
   COMILX_UnloadDll();
   exit(0);
}
……
……
COMILX_UnloadDevice(hDevice);
COMILX_UnloadDll();
```

3.3 COMI-BUS 제어

▣ COMILX_SetComiBus

함수 원형

void **COMILX_SetComiBus** (HANDLE hDevice, BOOL bEnable, BOOL blsMaster)

함수 설명

이 함수는 COMI-BUS 의 운용 방식을 결정합니다. COMI-BUS 는 여러 대의 디바이스를 동기화 시키기 위하여 사용됩니다. 각 디바이스를 독립적으로 사용하려면 이 함수를 사용하지 않거나, bEnable 매개변수를 FALSE 로 하여 이 함수를 수행시키므로써 COMI-BUS 를 Disable 시켜야 합니다.

매개 변수

▶ ***hDevice*** : COMI-BUS 를 셋팅할 디바이스의 핸들값입니다. 이 값은 COMILX_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

▶ ***bEnable*** : 이 값이 0 이면 해당 디바이스는 COMI-BUS 를 사용하지 않으며, 1 이면 해당 디바이스는 COMI-BUS 를 사용하게 됩니다.

▶ ***blsMaster*** : 이 값이 0 이면 해당 디바이스는 SLAVE 로, 1 이면 MASTER 로 동작하게 됩니다. 만일 bEnable 값이 0 이면 blsMaster 는 무시됩니다.

참 고

□ COMI-LX20x 의 경우에 여러 대의 디바이스들에 대하여 A/D 스캔의 동기를 맞추기 위하여 COMI-BUS 를 사용할 수 있습니다. 이 때는 A/D 스캔 Trigger 신호는 MASTER 로 설정된 디바이스에서 발생되게 됩니다.

□ 동기화를 하려고 하는 여러 대의 디바이스들 중에서 하나의 디바이스만 MASTER 로 설정되어야 하며 나머지는 SLAVE 로 설정되어야 합니다.

□ 동기화를 하려고 하는 각 디바이스들은 서로 COMI-BUS 케이블로 연결이 되어 있어야 합니다.

사용예

3 개의 COMI-LX201 디바이스를 동기화 하기 위해 COMILX_SetComiBus()함수를 사용하는 예.

```
HANDLE hDevice1, hDevice2, hDevice3;

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

hDevice1 = COMILX_LoadDevice (COMI_LX201, 0);
hDevice2 = COMILX_LoadDevice (COMI_LX201, 1);
hDevice3 = COMILX_LoadDevice (COMI_LX201, 2);

COMILX_SetComiBus (hDevice1, TRUE, TRUE);
COMILX_SetComiBus (hDevice2, TRUE, FALSE);
COMILX_SetComiBus (hDevice3, TRUE, FALSE);

// 이 이후에 A/D SCAN 함수들을 수행한다. //
……
……
COMILX_UnloadDevice(hDevice1);
COMILX_UnloadDevice(hDevice2);
COMILX_UnloadDevice(hDevice3);
COMILX_UnloadDll();
```

3.4 아날로그 입력(Analog Input)

이 장에서는 A/D 에 관련된 함수들을 소개합니다. A/D 는 아날로그(Analog) 신호를 입력 받아 디지털(Digital)값으로 변환해주는 기능입니다.

COMIDAS 에서 지원하는 A/D 방식에는 두 가지가 있습니다. 첫 번째는 Single point A/D 방식이며 두 번째는 A/D Scan 방식입니다.

Single point A/D 는 사용자가 원하는 시점에서 소프트웨어적으로 A/D trigger 를 하여 A/D 데이터를 획득하는 방식입니다. 단 LX20-시리즈 디바이스는 Single point A/D 방식은 지원하지 않습니다.

A/D Scan 방식은 사용자가 직접 A/D trigger 를 하지 않고, 디바이스에 내장된 타이머가 일정 주기로 A/D trigger 를 하고 변환된 A/D 데이터를 특정 버퍼에 저장하는 방식입니다. 이 방식은 Single point A/D 방식에 비해 속도가 빠르고 정확한 샘플링 주기를 보장할 수 있습니다.

LX10-시리즈 보드와 LX20-시리즈 보드는 A/D 스캔 방식에 있어서 많은 차이점이 있습니다. 따라서 A/D 스캔 함수는 LX10-시리즈 보드에 사용되는 함수와 LX20-시리즈 보드에 사용되는 함수가 구분되어 있습니다. COMILX_US1_XXXXX 의 형태로된 함수는 LX10-시리즈 보드에 적용되는 A/D 스캔 함수이며, COMILX_US2_XXXXX 의 형태로된 함수는 LX20-시리즈 보드에 적용되는 A/D 스캔 함수입니다.

본 매뉴얼의 3-4-2 단원에서 LX10-시리즈 보드용 A/D 스캔 함수를 수록하였으며, 3-4-3 단원에서는 LX20-시리즈 보드용 A/D 스캔 함수를 따로 수록하였습니다.

3.4.1 아날로그 입력(General)

이 단원에서는 Single point A/D 함수를 포함한 일반적으로 사용되는 A/D 관련 함수들을 소개합니다. 이에 관련된 함수들의 리스트는 다음과 같습니다.

함수명	각 보드별 적용 여부					
	LX10x	LX20x	LX301	LX401	LX402	LX501
COMILX_AD_SetInputType	V					
COMILX_AD_SetRange	V	V				
COMILX_AD_GetDigit	V					
COMILX_AD_GetVolt	V					

[표 3-2] Analog Input 일반 함수 리스트 및 각 보드별 지원 여부

▣ COMILX_AD_SetInputType

함수 원형

void **COMILX_AD_SetInputType** (HANDLE hDevice, int nInputMode)

함수 설명

이 함수는 아날로그 입력 신호의 연결 형식을 소프트웨어적으로 설정합니다. 아날로그 입력 신호의 연결 형식에는 Single ended 방식과 Differential 방식의 두 가지가 있습니다(하드웨어 매뉴얼 참조).

매개 변수

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

▶ *nInputMode* : 아날로그 입력 신호의 연결 형식을 설정합니다. 이 값은 다음 중 하나의 값이어야 합니다. 컴퓨터 부팅시에 연결 형식의 기본값은 AI_SINGLE 로 설정됩니다.

Value	Meaning
0 또는 AI_DIFF	아날로그 입력 신호의 연결 형식을 **Differential** 로 설정합니다.
1 또는 AI_SINGLE	아날로그 입력 신호의 연결 형식을 **Single ended** 로 설정합니다.

참 고

LX10-시리즈만이 소프트웨어적으로 연결 형식을 설정할 수 있습니다. LX20-시리즈는 하드웨어의 점퍼 셋팅을 통하여 연결 형식을 설정하여야 합니다.

▣ COMILX_AD_SetRange

함수 원형

BOOL **COMILX_AD_SetRange** (HANDLE hDevice, int ch, float vmin, float vmax)

함수 설명

이 함수는 각 A/D 채널의 입력 범위를 정해줍니다.

매개 변수

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

▶ *ch* : A/D 범위를 정해줄 채널 번호를 지정합니다. 채널 번호는 0 부터 시작합니다.

▶ *vmin* : A/D 범위의 최소값을 지정합니다. 유효한 vmin 값은 보드 종류에 따라 다음과 같습니다.

- COMI-LX10x 디바이스 ~ 0, -1, -2, -5, -10
- COMI-LX20x 디바이스 ~ -1, -2, -5, -10

▶ *vmax* : A/D 범위의 최대값을 지정합니다. 유효한 vmax 값은 1, 2, 5, 10 입니다.

Return 값

함수 수행의 성공 여부

Value	Meaning
0	함수 수행 실패
1	함수 수행 성공

참 고

디바이스별 설정 가능한 A/D 범위는 다음과 같습니다.

디바이스	설정 가능한 A/D 범위
COMI-LX10x	±1, ±2, ±5, ±10, 0~1, 0~2, 0~5, 0~10
COMI-LX20x	±1, ±2, ±5, ±10

사용예

A/D CH0 은 -10 ~ 10 의 입력 범위를, 그리고 A/D CH1 은 -5 ~ 5 의 입력 범위를 가지도록 설정하는 예

```
// 이 이전에 COMILX_LoadDevice(…)가 수행되어 있어야 한다 //
COMILX_AD_SetRange (hDevice, 0, -10, 10);
COMILX_AD_SetRange (hDevice, 1, -5, 5);
```

▣ COMILX_AD_GetDigit

함수 원형

short **COMILX_AD_GetDigit** (HANDLE hDevice, int ch)

함수 설명

이 함수는 주어진 채널에 대하여 A/D 변환을 수행하고 그 값을 정수값으로 반환합니다.

매개 변수

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

▶ *ch* : A/D를 수행할 채널 번호. 채널 번호는 0부터 시작합니다.

Return 값

□ 정수형의 A/D 결과값. 이 값의 범위는 A/D 분해능에 따라 다르며 디바이스별로 보면 다음과 같습니다.

디바이스	분해능	반환되는 정수값의 범위
COMI-LX101	12 Bit	0 ~ 4095
COMI-LX102	14 Bit	0 ~16383
COMI-LX103	16 Bit	-32768 ~ 32767

□ 이 함수에서 반환하는 정수값을 Voltage 값으로 변환하려면 다음과 같은 식이 적용되어야 합니다.

$$V = \frac{V_{\max} - V_{\min}}{D_{\max} - D_{\min}}(D - D_{\min}) + V_{\min}$$

여기서

V : 정수값으로부터 환산되는 Voltage 값

D : 환산하고자 하는 대상 정수값

V_{max} : A/D 범위의 최대값 (COMILX_AD_SetRange 함수 참조)

V_{min} : A/D 범위의 최소값 (COMILX_AD_SetRange 함수 참조)

D_{max} : 정수값의 최대 범위값(A/D 분해능에 따라 다르며 앞의 표 참조)

D_{min} : 정수값의 최소 범위값(A/D 분해능에 따라 다르며 앞의 표 참조)

사용예

A/D CH0의 A/D 값을 정수값으로 받는 예

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

#define DEV_ID    COMI_LX101
#define CHAN   0
#define VMIN  -10
#define VMAX   10

void main (void)
{
   HANDLE hDevice;
   int ad_digit;

   if(!COMILX_LoadDll()){
       printf("Comidas.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);
   }

   // Analog Input type을 'Single ended'방식으로 설정한다. //
   // Differential 입력으로 하고자 한다면 AI_SINGLE대신에      //
   //          AI_DIFF          로          변경하여야          한다.
//
   COMILX_AD_SetInputType (hDevice, AI_SINGLE);

   COMILX_AD_SetRange (hDevice, CHAN, VMIN, VMAX);
   printf("A/D 변환을을 시작하려면 아무키나 누르십시오.\n");
   printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
```

```
    _getch();
    while (!_kbhit())
    {
        ad_digit = COMILX_AD_GetDigit (hDevice, CHAN);
        printf("A/D Result (Digit) = %d\n", ad_digit);
        Sleep(500);
    }

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

▣ COMILX_AD_GetVolt

함수 원형

float **COMILX_AD_GetVolt** (HANDLE hDevice, int ch)

함수 설명

이 함수는 주어진 채널에 대하여 A/D 변환을 수행하고 그 값을 voltage 값으로 반환합니다.

매개 변수

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

▶ *ch* : A/D를 수행할 채널 번호. 채널 번호는 0부터 시작합니다.

Return 값

A/D 결과값을 Voltage값으로 반환합니다.

사용예

A/D CH0의 A/D 값을 Voltage 값으로 받는 예

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

#define DEV_ID    COMI_LX101
#define CHAN   0
#define VMIN  -10
#define VMAX   10

void main (void)
{
   HANDLE hDevice;
   float ad_volt;

   if(!COMILX_LoadDll()){
       printf("Comidas.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);
    }

    // Analog Input type을 'Single ended'방식으로 설정한다. //
    // Differential 입력으로 하고자 한다면 AI_SINGLE 대신에    //
    // AI_DIFF로 변경하여야 한다.                              //
    COMILX_AD_SetInputType (hDevice, AI_SINGLE);

    COMILX_AD_SetRange (hDevice, CHAN, VMIN, VMAX);
    printf("A/D 변환을을 시작하려면 아무키나 누르십시오.\n");
    printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
    _getch();
    while (!_kbhit())
    {
        ad_ volt = COMILX_AD_GetVolt (hDevice, CHAN);
        printf("A/D Result (volt) = %.3f\n", ad_volt);
        Sleep(500);
    }

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

3.4.2 COMI-LX10x 시리즈 전용 A/D 스캔

이 단원에서는 LX10-시리즈 디바이스 전용으로 사용되는 A/D 스캔 함수들을 소개합니다. LX10-시리즈 전용 A/D 스캔에 관련된 함수들은 함수명이 COMILX_US1_xxxx 의 형식으로 구성됩니다.

Unlimited A/D Scan 방식은 사용자가 직접 A/D trigger 를 하지 않고, 디바이스에 내장된 타이머가 일정 주기로 A/D trigger 를 해주고 변환된 A/D 데이터를 특정 버퍼에 저장하는 방식입니다. 이 때 Scan 데이터를 저장하는 버퍼는 환형 버퍼 형식으로 운용되며 사용자는 필요시에 이 버퍼로부터 데이터를 취하게 됩니다. 이 방식은 Single point A/D 방식에 비해 속도가 빠르고 정확한 샘플링 주기를 보장할 수 있습니다. 따라서 이 방식은 고속 A/D를 할 때 아주 유용하게 사용될 수 있습니다.

LX10-시리즈 디바이스 전용 A/D 스캔 함수의 리스트는 다음과 같습니다.

함수명	각 보드별 적용 여부								
	LX101	LX102	LX103	LX201	LX202	LX203	LX301	LX401	LX501
COMILX_US1_Start	V	V	V						
COMILX_US1_Stop	V	V	V						
COMILX_US1_CurCount	V	V	V						
COMILX_US1_GetBuffer	V	V	V						
COMILX_US1_SBPos	V	V	V						
COMILX_US1_RetrvOne	V	V	V						
COMILX_US1_RetrvChannel	V	V	V						
COMILX_US1_RetrvBlock	V	V	V						
COMILX_US1_ReleaseBuf	V	V	V						

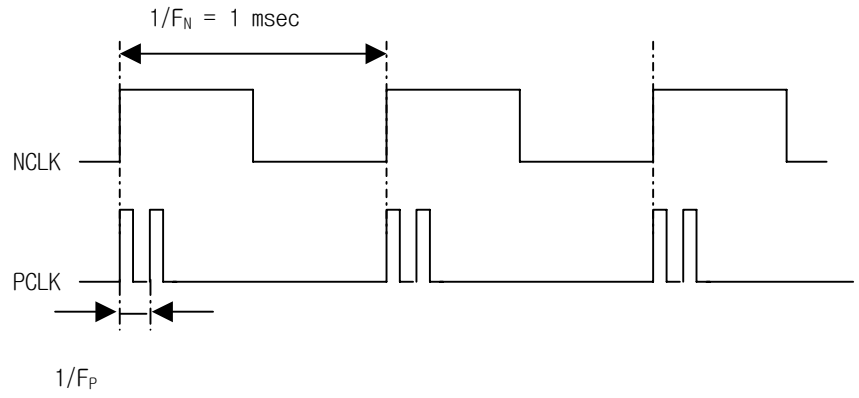
[표 3-3] LX10-시리즈 디바이스 전용 A/D 스캔 함수 리스트 및 각 보드별 지원 여부

우선 Unlimited A/D Scan 함수들을 소개하기 앞서 몇 가지 미리 숙지해야할 사항들을 수록합니다.

□ SCAN 이란 ?

A/D SCAN 이라함은 지정된 여러 채널을 순차적으로 A/D 변환한다는 뜻입니다. 따라서 1 회의 SCAN 은 사용자가 지정한 모든 채널에 대하여 1 회씩 A/D 변환이 완료되었을 때를 1 회의 SCAN 으로 정의합니다. 따라서 이후에 사용되는 Scan rate(또는 Scan frequency)라는 용어의 의미는 스캔채널로 지정된 각각의 채널에 대하여 1 초당 변환되는 A/D 횟수를 말하게 됩니다. 이는 동일 스캔내에서의 각 채널간 A/D 변환 주기를 결정해주는 Sampling rate(또는 Sampling frequency)와는 구분이 되어야 합니다. External trigger 를 사용하는 경우를 제외하곤 Scan rate 와 Sampling rate 는 디바이스 내부의 타이머에 의해 제어됩니다.

0 번 채널과 1 번 채널을 스캔 채널로 지정하고 Scan rate 를 1 KHz 로 지정한 경우 각 디바이스에 따른 Scan rate 와 Sampling rate 를 그림으로 표시하면 다음과 같습니다.



NCLK : 스캔 타이머 신호

PCLK : 샘플링 타이머 신호, 이 신호가 실제 A/D Trigger 를 한다.

F_N : Scan frequency

F_P : Sampling frequency (이 값은 디바이스의 A/D 칩이 지원하는 최대 주파수로 설정되며 디바이스에 따라 달라진다.)

□ **환형 버퍼**

Unlimited A/D Scan 에서 A/D 데이터가 저장되는 버퍼는 환형 버퍼 형식으로 운용됩니다. 환형 버퍼는 한정된 버퍼에 무한히 데이터를 기록하기 위해 사용되는 것으로써, 데이터가 버퍼의 마지막 위치까지 다 채워지면 버퍼의 처음 위치부터 다시 채워 나가는 방식을 말합니다.

▣ COMILX_US1_Start

함수 원형 :

long **COMILX_US1_Start** (HANDLE hDevice, int nNumChannel, int *pChanList, UINT dwScanFreq, UINT nBufSize, int nTrsMethod)

함수 설명 :

이 함수는 Unlimited scan 기능을 시작합니다.

매개 변수 :

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

▶ ***nNumChannel*** : A/D scan 할 A/D 채널의 수.

▶ ***pChanList*** : A/D scan 을 수행할 채널 리스트를 담고 있는 배열 또는 포인터.

▶ ***dwScanFreq*** : A/D scan frequency 를 Hz 단위로 지정합니다. 이 값은 SCAN 과 SCAN 사이의 시간차를 결정합니다.

▶ ***nBufSize*** : 스캔 데이터를 저장할 환형버퍼의 크기를 결정하는 값으로써 각 채널의 데이터가 환형버퍼에 오버랩(Overlap)되지 않고 담길 수 있는 최대 데이터 수를 의미합니다. 환형 버퍼는 디바이스 드라이버에서 자동으로 할당하며 실제 크기는

환형 버퍼의 실제 크기(bytes) = nNumChannel * nBufSize * sizeof(short)

가 됩니다.

▶ ***nTrsMethod*** : A/D 디바이스에서 스캔버퍼로 데이터를 전송하는 방식을 지정합니다. 이 값은 다음의 두 값 중의 하나이어야 합니다.

Value	Meaning
1 또는 TRS_SINGLE	1 회의 SCAN 이 완료될 때마다 인터럽트를 발생시켜 데이터를 전송합니다. 인터럽트의 한계에 따라 Scan frequency 가 30 KHz 이상이 되면 이 방식이 적절히 작동

	하지 않을 수 있습니다.
2 또는 TRS_BLOCK	이 방식을 선택하면, A/D 디바이스는 A/D Conversion 데이터를 디바이스에 내장되어 있는 FIFO 메모리에 일단 저장하고, 1024 개의 데이터가 쌓이면 인터럽트를 발생시켜 데이터를 1024 개 단위로 사용자 버퍼에 전송합니다. 이 방식은 고속 A/D scan 시에 적합합니다.

Return 값

실제로 설정되는 스캔 주파수를 Hz 단위로 반환합니다. 사용자가 지정한 스캔 주파수와 실제로 설정되는 스캔 주파수는 약간의 차이가 있을 수 있습니다. 이 값이 0 보다 작으면 수행도중 에러가 발생하였음을 의미합니다.

사용예

A/D CH0 의 A/D 값을 Voltage 값으로 받는 예

□ 0 번 채널 한 채널을 1KHz 로 SCAN 할 때의 Start 함수 사용예

```
long dwActFreq;
int nChanNo = 0;
…
dwActFreq = COMILX_US1_Start (hDevice, 1, &nChanNo, 1000, 10240,
TRS_SINGLE);
```

□ 32 채널 모두에 대하여 채널당 10KHz 로 SCAN 할 때의 Start 함수 사용예 (TRS_BLOCK 모드 사용)

```
long dwActFreq;
int nChanList[32];
…
COMILX_AD_SetInputType (hDevice, AI_SINGLE); // 32 채널을 사용하기
위해서는 연결 형식이 SINGLE-ENDED 로 설정되어야함
for(i=0; i<32; i++){
 nChanList[i] = i;
}
dwActFreq = COMILX_US1_Start (hDevice, 32, nChanList, 10000,
8192, TRS_BLOCK);
```

▣ COMILX_US1_Stop

함수 원형

void COMILX_US1_Stop (HANDLE hDevice, BOOL bReleaseBuf)

함수 설명

이 함수는 Unlimited scan 을 종료합니다.

매개 변수

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

▶ *bReleaseBuf* : COMILX_US1_Start()가 수행될 때 할당되었던 스캔 버퍼를 메모리 해제시킬것인지를 결정합니다. 만일 이 값을 FALSE 로 지정하면 후에 반드시 COMILX_US1_ReleaseBuf() 사용하여 버퍼를 해제하여야 합니다. 이 값을 TRUE 로 지정하면 COMILX_US1_ReleaseBuf() 함수를 수행할 필요가 없습니다.

▣ COMILX_US1_CurCount

함수 원형

ULONG **COMILX_US1_CurCount** (HANDLE hDevice)

함수 설명

이 함수는 현재까지 수행된 SCAN 횟수를 반환합니다. 사용자는 버퍼에서 데이터를 취할 때에 이 함수를 참조하여 가장 최근 스캔된 데이터의 위치를 알아낼 수 있습니다.

매개 변수

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

Return 값

현재까지 수행된 총 SCAN 횟수

▣ COMILX_US1_GetBuffer

함수 원형 :

short* **COMILX_US1_GetBuffer**(HANDLE hDevice)

함수 설명 :

이 함수는 스캔 버퍼를 가리키는 포인터를 반환합니다. 사용자는 이 포인터를 이용하여 스캔 데이터를 직접 Access 할 수 있습니다.

매개 변수 :

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

Return 값:

스캔 버퍼에 대한 포인터

참 고 :

□ 이 함수를 이용하여 데이터를 취할 때는 **COMILX_US1_SBPos()** 함수를 함께 사용하는 것이 편리합니다.

□ 이 함수 대신에 COMILX_US1_RetrvOne() 또는 COMILX_US1_RetrvChannel() 또는 COMILX_US1_RetrvBlock() 함수를 사용할 수 있습니다.

사용예 :

이프로그램은 A/D CH0 와 CH1 의 두채널을 Unlimited scan 을 이용하여 A/D 변환을 수행하고 그 결과를 파일로 저장하는 예제입니다. 예제는 scan 버퍼를 직접 handling 하는 방법을 보여주고 있습니다.

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

#define DEV_ID     COMI_LX101
#define NUM_CH     2  /* Number of channels */
#define S_FREQ     1000   /* Scan freq. -> 1000 Hz */
#define MSB       10240   /* Scan buffer 크기 */

void main (void)
```

```
{
   HANDLE hDevice;
   int ch_list[2] = {0, 4};/*Scan channel list : 0번 과 4번 채널*/
   FILE *fp;
   short *pScanBuf;
   ULONG c, prv_cnt, cur_cnt;
   UINT idx1, idx2;

   /* Load DLL */
   if(!COMILX_LoadDll()){
       printf("ComidasLX.dll load failure\n");
       printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
       _getch();
       exit(0);
   }
   /* Load Device */
   hDevice = COMILX_LoadDevice (DEV_ID, 0);
   if(hDevice == INVALID_HANDLE_VALUE){
       printf("Can't load specified device!\n");
       printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
       _getch();
       COMILX_UnloadDll();
       exit(0);
   }

   printf("A/D Scan을 시작하려면 아무키나 누르십시오.\n");
   printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
   _getch();

   COMILX_AD_SetInputType (hDevice, AI_SINGLE);

   /* start unlimited scan */
   long act_freq = COMILX_US1_Start (hDevice, NUM_CH, ch_list,
S_FREQ, MSB, TRS_SINGLE);

   /* Create a file to save data */
   fp = fopen ("c:\\ComiUscan.txt", "w");
   fprintf (fp,"   CH0    CH4\n");
   /* Get scan buffer pointer */
   pScanBuf = COMILX_US1_GetBuffer(hDevice);
   prv_cnt = 0; // initialize count variable

   while(!kbhit())
   {
       cur_cnt = COMILX_US1_CurCount (hDevice);

       /*prv_cnt: end count of previously processed data block */
       /*prv_cnt+1: initial count of newly scaned data block   */
       /*cur_cnt: end count of current newly scanned data block*/
       for(c = prv_cnt+1; c <= cur_cnt; c++)
       {
```

```
            /* COMILX_US1_SBPos (..) 함수를 이용하여 각 채널의     */
            /* 버퍼상의 인덱스를 얻는다.                          */
            /* 이때 두번째 파라미터는 채널번호가 아니라 채널 리스트상  */
            /* 의 순서이다. 즉, CH0 는 0 이고, CH4 는 1 이 된다.     */
            idx1 = COMILX_US1_SBPos (hDevice, 0, c);
            idx2 = COMILX_US1_SBPos (hDevice, 1, c);

            /* File 에 데이터를 저장한다. 아래의 코드는 Voltage 값으로 */
            /* 저장하는 것이 아니고 0~4095 의 정수값으로 저장된다.     */
            fprintf     (fp,     "%6d     %6d\n",     pScanBuf[idx1],
pScanBuf[idx2]);
        }
        printf("Saved scan count = %u\n", cur_cnt);
        prv_cnt = cur_cnt;
    }

    COMILX_US1_Stop (hDevice, TRUE);
    fclose(fp);

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

▣ COMILX_US1_SBPos

함수 원형

UINT **COMILX_US1_SBPos**(HANDLE hDevice, int chOrder, ULONG scanCount)

함수 설명

이 함수는 원하는 스캔 데이터가 있는 스캔 버퍼의 인덱스(Index)를 계산해줍니다. 스캔 버퍼는 환형 버퍼로 운용되므로 Scan count 와 버퍼상의 인덱스가 항상 일치하지는 않습니다. 원하는 데이터의 버퍼상 인덱스를 얻기 위해서는 채널 순서와 Scan count 를 가지고 약간의 조작이 필요한데, COMILX_US1_SBPos()는 이를 해주는 함수입니다.

매개 변수

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

▶ *chOrder* : 데이터를 취하기 원하는 채널의 채널 리스트 상의 순서(0 based)입니다. 이 값은 채널 번호가 아님을 주의하여야 한다.

▶ *scanCount* : 원하는 데이터의 Scan count.

Return 값

chOrder 와 scanCount 에 의해 지정된 데이터의 스캔 버퍼상의 인덱스.

참 고

COMI_US_SBPos() 함수의 사용예 : 스캔 채널 배열이 {CH0, CH5}인 경우, 가장 최근에 Scan 된 100 회의 데이터를 취하는 예는 다음과 같이 들 수 있습니다.

```
short *buffer = COMI_US_GetBufPtr(hDevice);
cur_cnt = COMI_US_CurCount(hDevice);
for(i=0; i<100; i++){
   index = COMI_US_SBPos (hDevice, 0, cur_cnt - i);
ch0_data[100-i] = buffer[index];
   index = COMI_US_SBPos (hDevice, 1, cur_cnt - i);
ch5_data[100-i] = buffer[index];
}
```

▣ COMILX_US1_RetrvOne

함수 원형

float COMILX_US1_RetrvOne (HANDLE hDevice, int chOrder, ULONG scanCount)

함수 설명

이 함수는 원하는 위치의 A/D Scan 데이터를 Voltage 값으로 반환합니다.

매개 변수

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

▶ *chOrder* : 데이터를 취하기 원하는 채널의 채널 리스트 상의 순서(0 based)입니다. 이 값은 채널 번호가 아님을 주의하여야 합니다.

▶ *scanCount* : 원하는 데이터의 Scan count.

Return 값

chOrder 와 scanCount 에 의해 지정된 데이터를 Voltage 형식으로 반환합니다.

▣ COMILX_US1_RetrvChannel

함수 원형 :

ULONG **COMILX_US1_RetrvChannel** (HANDLE hDevice, int chOrder, ULONG startCount, int maxNumData, void *pDestBuf, TComiVarType VarType)

함수 설명 :

이 함수는 A/D Scan 채널 중에서 하나의 채널에 대한 데이터 블럭을 Voltage 값으로 환산하여 전달합니다. 데이터 블럭은 사용자가 지정한 startCount 에서부터 maxNumData 에서 지정한 수만큼이 됩니다.

매개 변수 :

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

▶ ***chOrder*** : 데이터를 취하기 원하는 채널의 채널 리스트 상의 순서(0 based)입니다. 이 값은 채널 번호가 아님을 주의하여야 합니다.

▶ ***startCount*** : 전달 받고자 하는 데이터 블럭의 시작 Scan count.

▶ ***maxNumData*** : 전달 받고자 하는 데이터 블럭의 크기(데이터 수)를 지정 합니다. 이 값이 양수이면 startCount 부터 이후에 스캔된 데이터 중 maxNumData 에서 지정한 수만큼 데이터를 전달합니다. 이 값이 음수이면 startCount 부터 이전에 스캔된 데이터 중 maxNumData 에서 지정한 수만큼 데이터를 전달합니다.

▶ ***pDestBuf*** : 데이터를 전달 받을 버퍼 포인터를 지정합니다. 이 버퍼의 데이터형은 VarType 파라미터에서 지정한 데이터형과 일치해야 합니다. 또한 이 버퍼의 크기는 maxNumData 에서 지정한 값보다 크거나 같아야 한다.

▶ ***VarType*** : pDestBuf 의 데이터 형을 지정합니다. 이 값은 다음의 값 중 하나이어야 합니다.

Value	Meaning

0 또는 VT_SHORT	pDestBuf 가 short 형 포인터임을 의미하며, 데이터는 Voltage 로 환산되기 이전의 정수형값으로 전달됩니다.
1 또는 VT_FLOAT	pDestBuf 가 float 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.
2 또는 VT_DOUBLE	pDestBuf 가 double 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.

Return 값:

실제 전달된 데이터 수. 만일 startCount 이후에 현재까지 스캔된 데이터 수가 maxNumData 에서 지정한 수 보다 작으면, 현재 스캔된 데이터까지만 전달하게됩니다.

사용예 :

□ **사용예 1.**

이프로그램은 A/D CH0 와 CH1 의 두채널을 Unlimited scan 을 이용하여 A/D 변환을 수행하고 그 결과를 파일로 저장하는 예제입니다.

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

#define DEV_ID   COMI_LX101
#define NUM_CH    2       /* Number of channels */
#define S_FREQ    1000    /* Scan freq. -> 1000 Hz */
#define MSB     10240
#define CH_BUF_SIZE   (S_FREQ * 2)

void main (void)
{
   HANDLE hDevice;
   int ch_list[2] = {0, 4};  /* Scan channel list : 0 번 과 4 번 채널
*/
   FILE *fp;
   float *pChanBuf;
   ULONG prv_cnt=0, count;
   ULONG i;

   /* Load DLL */
   if(!COMILX_LoadDll()){
       … /* 에러 처리 */
       exit(0);
   }
   /* Load Device */
   hDevice = COMILX_LoadDevice (DEV_ID, 0);
```

```
    if(hDevice == INVALID_HANDLE_VALUE){
        … /* 에러 처리 */
        COMILX_UnloadDll();
        exit(0);
    }

    // Analog Input type을 'Single ended'방식으로 설정한다. //
    // Differential 입력으로 하고자 한다면 AI_SINGLE 대신에  //
    // AI_DIFF 로 변경하여야 한다.                          //
    COMILX_AD_SetInputType (hDevice, AI_SINGLE);

    // Set A/D range //
    COMILX_AD_SetRange(hDevice, 0, -10, 10);
    COMILX_AD_SetRange(hDevice, 4, -10, 10);

    printf("A/D Scan을 시작하려면 아무키나 누르십시오.\n");
    printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
    _getch();

    /* start unlimited scan */
    long act_freq = COMILX_US1_Start (hDevice, NUM_CH, ch_list,
S_FREQ, MSB, TRS_SINGLE);
    if(act_freq < 0){
        …/* Error 처리 */
        COMILX_UnloadDevice(hDevice);
        COMILX_UnloadDll();
        exit(0);
    }

    /* Create a file to save data */
    /* Create a file to save data */
    fp = fopen ("c:\\ComiUscan.txt", "w");
    fprintf (fp,"   CH0\n");
    // Allocate a buffer to retrieve scan data //
    pChanBuf = (float *)malloc(sizeof(float)*CH_BUF_SIZE);
    prv_cnt = 0;
    while(!kbhit())
    {
        // CH0의 SCAN 데이터를 얻어온다.                          //
        // prv_cnt 이후에 scan한 데이터수가 CH_BUF_SIZE 보다 적으면  //
        // 현재 scan된 것까지 버퍼에 담아주고 CH_BUF_SIZE 보다 크면   //
        // CH_BUF_SIZE 만큼만 담아준다.                           //
        count = COMILX_US1_RetrvChannel (hDevice, 0, prv_cnt+1,
CH_BUF_SIZE, pChanBuf, VT_FLOAT);
        for(i=0; i<count; i++)
            fprintf (fp, "%6.2f\n", pChanBuf[i]);
        prv_cnt += count;
        printf("Number of saved data = %d\n", prv_cnt);
        Sleep(100);
    }
```

```
   free(pChanBuf); // Free channel buffer
   COMILX_US1_Stop (hDevice, TRUE);
   fclose(fp);

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

□ **사용예 2.**

A/D CH0,CH1,CH2 의 3 채널을 Unlimited scan 을 이용하여 A/D convert 를 하고 3 채널 모두의 데이터를 파일에 저장하는 예제.

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

#define DEV_ID   COMI_LX101
#define NUM_CH    3       /* Number of channels */
#define S_FREQ    5000    /* Scan freq. -> 1000 Hz */
#define MSB     10240

#define CH_BUF_SIZE  1024 // 사용자 버퍼 크기 정의 => 이값은 사용자가 마음대로 정하여도 좋다 //
float Buffer[NUM_CH][CH_BUF_SIZE];
void main (void)
{
   HANDLE hDevice;
   int ch_list[2] = {0, 1, 2}; /* Scan channel list */
   FILE *fp;
   float *pChanBuf;
   ULONG prv_cnt=0, count;
   ULONG i;

   /* Load DLL */
   if(!COMILX_LoadDll()){
       … /* 에러 처리 */
       exit(0);
   }
   /* Load Device */
   hDevice = COMILX_LoadDevice (DEV_ID, 0);
   if(hDevice == INVALID_HANDLE_VALUE){
       … /* 에러 처리 */
       COMILX_UnloadDll();
       exit(0);
   }

   // Analog Input type 을 'Single ended'방식으로 설정한다. //
   // Differential 입력으로 하고자 한다면 AI_SINGLE 대신에  //
```

```
   // AI_DIFF 로 변경하여야 한다.                    //
   COMILX_AD_SetInputType (hDevice, AI_SINGLE);

   // Set A/D range //
   COMILX_AD_SetRange(hDevice, 0, -10, 10);
   COMILX_AD_SetRange(hDevice, 1, -10, 10);
COMILX_AD_SetRange(hDevice, 2, -10, 10);


   printf("A/D Scan 을 시작하려면 아무키나 누르십시오.\n");
   printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
   _getch();

   /* start unlimited scan */
   long act_freq = COMILX_US1_Start (hDevice, NUM_CH, ch_list,
S_FREQ, MSB, TRS_BLOCK);
   if(act_freq < 0){
       /* Error 처리 */
       COMILX_UnloadDevice(hDevice);
       COMILX_UnloadDll();
       exit(0);
   }

/* Create a file to save data */
   fp = fopen ("c:\\ComiUscan.txt", "w");
fprintf (fp,"   CH0    CH1    CH2\n");
   prv_cnt = 0;
   while(!kbhit())
   {
       // 각 채널의 스캔 데이터를 가져온다.                          //
       // prv_cnt 이후에 scan 한 데이터수가 CH_BUF_SIZE 보다 적으면  //
       // 현재 scan 된 것까지 버퍼에 담아주고 CH_BUF_SIZE 보다 크면   //
       // CH_BUF_SIZE 만큼만 담아준다.                               //
       // 그리고 데이터를 전달받는 중에도 SCAN 이 진행되므로 각 채널마다 //
       // 전달되는 데이터 수가 다를 수 있다. 이를 방지하기 위하여 첫번째 //
       // 채널의 전달된 데이터 수와 동일한 수의 데이터를 나머지 채널이  //
       // 받을 수 있도록 두 번째 채널부터는 maxNumData 파라미터를      //
       // 변수로 처리하였음에 유의할 것                              //
       count = COMILX_US1_RetrvChannel (hDevice, 0, prv_cnt+1,
CH_BUF_SIZE, &Buffer[0], VT_FLOAT);
       COMILX_US1_RetrvChannel (hDevice, 1, prv_cnt+1, count,
&Buffer[1], VT_FLOAT);
       COMILX_US1_RetrvChannel (hDevice, 2, prv_cnt+1, count,
&Buffer[2], VT_FLOAT);
       if(count > 0){
           for(i=0; i<count; i++)
               fprintf (fp, "%6.2f %6.2f %6.2f\n", Buffer[0][i],
Buffer[1][i], Buffer[2][i]);
           prv_cnt += count;
           printf("Number of saved data = %d\n", prv_cnt);
```

```
            }
        }
        free(pChanBuf); // Free channel buffer
        COMILX_US1_Stop (hDevice, TRUE);
        fclose(fp);

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

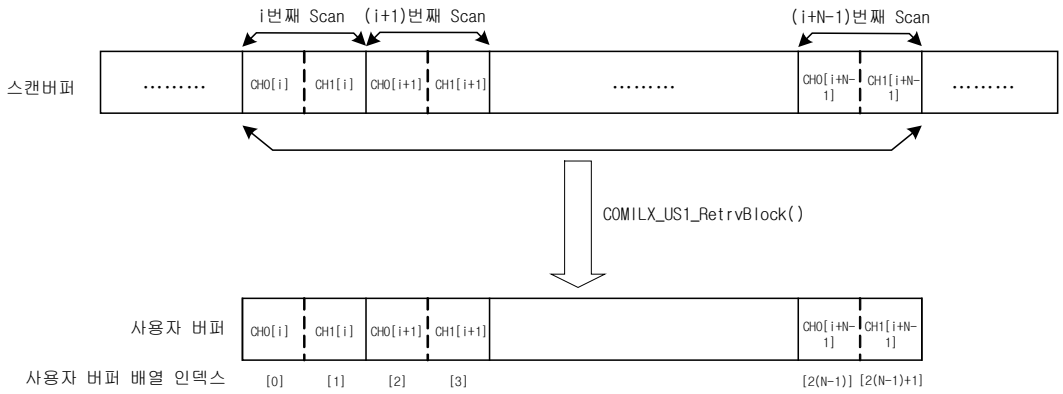
▣ COMILX_US1_RetrvBlock

함수 원형

ULONG COMILX_US1_RetrvBlock (HANDLE hDevice, UINT startCount, int maxNumScan, void *pDestBuf, TComiVarType VarType)

함수 설명

이 함수는 A/D Scan 전 채널에 대한 데이터를 사용자가 지정하는 버퍼에 전달합니다. 전달되는 데이터 블록의 시작 위치는 startCount 에 의해 결정되며, 그 크기는 maxNumScan 에 의하여 결정됩니다. 데이터 블록의 실제 크기는 maxNumScan * 채널수가 됩니다. 예를 들어 CH0 와 CH1 의 두 채널에 대하여 A/D 스캔을 수행할 때 startCount 를 1, maxNumScan 을 N 으로 하였다면 스캔 버퍼에서 사용자 버퍼로 데이터가 전달되는 것은 다음 그림과 같습니다.



매개 변수

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

▶ *chOrder* : 데이터를 취하기 원하는 채널의 채널 리스트 상의 순서(0 based)입니다. 이 값은 채널 번호가 아님을 주의하여야 합니다.

▶ *startCount* : 전달 받고자 하는 데이터 블럭의 시작 Scan count.

▶ *maxNumScan* : 전달 받고자 하는 데이터 블럭의 크기(스캔 횟수)를 지정합니다. 이 값이 양수이면 startCount 부터 이후에 스캔된 데이터 중 maxNumScan 에서 지정한 수만큼 데이터를 전달합니다. 이 값이 음수이면 startCount 부터 이전에 스캔된 데이터 중 maxNumScan 에서 지정한 수만큼 데이터를 전달합니다.

▶ *pDestBuf* : 데이터를 전달 받을 버퍼 포인터를 지정합니다. 이 버퍼의 데이터형은 VarType 파라미터에서 지정한 데이터형과 일치해야 합니다. 또한 이 버퍼의 크기는 maxNumScan*채널수 보다 크거나 같아야 합니다.

▶ *VarType* : pDestBuf 의 데이터 형을 지정합니다. 이 값은 다음의 값 중 하나이어야 합니다.

Value	Meaning
0 또는 VT_SHORT	pDestBuf 가 short 형 포인터임을 의미하며, 데이터는 Voltage 로 환산되기 이전의 정수형값으로 전달됩니다.
1 또는 VT_FLOAT	pDestBuf 가 float 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.
2 또는 VT_DOUBLE	pDestBuf 가 double 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.

Return 값

실제 전달된 데이터 수. 만일 startCount 이후에 현재까지 스캔된 데이터 수가 maxNumData 에서 지정한 수 보다 작으면, 현재 스캔된 데이터까지만 전달하게됩니다.

참 고 :

□ 사용자 버퍼의 배열 크기는 반드시 (채널수 X maxNumData) 보다 크거나 같아야 합니다.

사용예

A/D CH0 와 CH1 의 두채널을 Unlimited scan 을 이용하여 A/D 변환을 수행하고 그 결과를 파일로 저장하는 예제입니다.

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

#define DEV_ID    COMI_LX101
#define NUM_CH    2   /* Number of channels */
#define S_FREQ    1000    /* Scan freq. -> 1000 Hz */
#define MSB       40960
#define CH_BUF_SIZE   (S_FREQ * 2)

void main (void)
{
   HANDLE hDevice;
   int ch_list[2] = {0, 4};  /* Scan channel list : 0번 과 4번 채널
*/
   FILE *fp;
   float *pChanBuf;
   ULONG prv_cnt=0, count;
   ULONG i;

   /* Load DLL */
   if(!COMILX_LoadDll()){
       … /* 에러 처리 */
       exit(0);
   }
   /* Load Device */
   hDevice = COMILX_LoadDevice (DEV_ID, 0);
   if(hDevice == INVALID_HANDLE_VALUE){
       … /* 에러 처리 */
       exit(0);
   }

   // Set A/D range //
   COMILX_AD_SetRange(hDevice, 0, -10, 10);
   COMILX_AD_SetRange(hDevice, 4, -10, 10);

   printf("A/D Scan을 시작하려면 아무키나 누르십시오.\n");
   printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
   _getch();

   /* start unlimited scan */
   long  act_freq  =  COMILX_US1_Start  (hDevice,  NUM_CH,  ch_list,
S_FREQ, MSB, TRS_SINGLE);
   if(act_freq < 0){
       … /* 에러 처리 */
       COMILX_UnloadDevice(hDevice);
       COMILX_UnloadDll();
       printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
       _getch();
       exit(0);
```

```
    }

    /* Create a file to save data */
    fp = fopen ("c:\\ComiUscan.txt", "w");
    fprintf (fp,"   CH0    CH4\n");
    // Allocate a buffer to retrieve scan data //
    pChanBuf = (float *)malloc(sizeof(float)*CH_BUF_SIZE* NUM_CH);
    prv_cnt = 0;
    while(!kbhit())
    {
        // 모든 채널의 SCAN 데이터를 얻어온다.                    //
        // prv_cnt 이후에 scan 한 데이터수가 CH_BUF_SIZE 보다 적으면   //
        // 현재 scan 된 것까지 버퍼에 담아주고 CH_BUF_SIZE 보다 크면   //
        // CH_BUF_SIZE 만큼만 담아준다.                          //
        count    =    COMILX_US1_RetrvBlock    (hDevice,    prv_cnt+1,
CH_BUF_SIZE, pChanBuf, VT_FLOAT);;
        for(i=0; i<count; i++)
            fprintf   (fp,   "%6.2f   %6.2f\n",   pChanBuf[i*NUM_CH],
pChanBuf[i*NUM_CH+1]);
        prv_cnt += count;
        printf("Number of saved data = %d\n", prv_cnt);
        Sleep(100);
    }

    free(pChanBuf); // Free channel buffer
    COMILX_US1_Stop (hDevice, TRUE);
    fclose(fp);

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

▣ COMILX_US1_ReleaseBuf

함수 원형

BOOL **COMILX_US1_ReleaseBuf**(HANDLE hDevice)

함수 설명

COMILX_US1_Start()가 수행될 때 할당되었던 스캔 버퍼를 메모리 해제 시킵니다. 이 함수는 COMILX_US1_Stop() 함수가 호출되기 전에 수행되어서는 안되며, COMILX_US1_Stop() 함수를 호출할 때 두 번째 파라미터를 TRUE 로 지정했을 때는 사용하지 않아도 됩니다.

매개 변수

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

Return 값

Value	Meaning
0	함수 수행 실패
1	함수 수행 성공

3.4.3 COMI-LX20x 시리즈 전용 A/D 스캔

이 단원에서는 LX20-시리즈 디바이스 전용으로 사용되는 A/D 스캔 함수들을 소개합니다. LX20-시리즈 전용 A/D 스캔에 관련된 함수들은 함수명이 **COMILX_US2_**xxxx 의 형식으로 구성됩니다. LX20-시리즈 디바이스는 고속 샘플링을 구현하기 위하여 LX10-시리즈 디바이스들과 여러 가지 차이점을 가지게 되어 그 함수를 구분하여 구현하였습니다.

함수명	각 보드별 적용 여부								
	LX101	LX102	LX103	LX201	LX202	LX203	LX301	LX401	LX501
COMILX_US2_SetTriggerEvent				V	V	V			
COMILX_US2_Start				V	V	V			
COMILX_US2_Resume				V	V	V			
COMILX_US2_IsBufFull				V	V	V			
COMILX_US2_ChangeScanFreq				V	V	V			
COMILX_US2_DmaCount				V	V	V			
COMILX_US2_GetBuffer				V	V	V			
COMILX_US2_RetrvChannel				V	V	V			
COMILX_US2_Stop				V	V	V			
COMILX_US2_ReleaseBuf				V	V	V			

[표 3-4] LX20-시리즈 디바이스 전용 A/D 스캔 함수 리스트 및 각 보드별 지원 여부

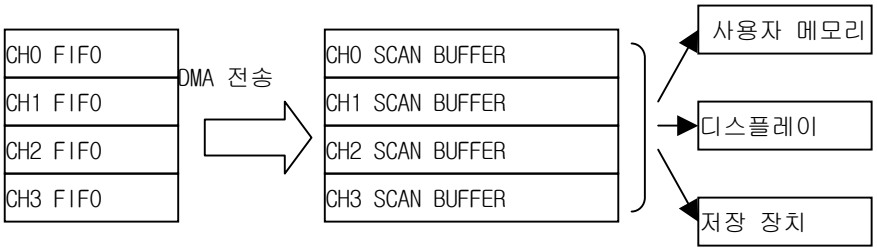
우선 LX20-시리즈 전용 A/D 스캔 함수들을 설명하기 전에 자주 사용되는 용어에 대하여 먼저 설명합니다. 이 용어들을 먼저 숙지하고 각 함수들을 파악한다면 좀더 쉽게 이해하실 수 있을 것입니다.

□ **스캔 버퍼 (Scan Buffer)**

이 단원에서 사용되는 스캔 버퍼란 용어는 드라이버가 자동 할당하여 A/D 된 데이터들을 임시 저장하는 PC 메모리 공간을 말합니다. 스캔 버퍼는 드라이버에서

자동적으로 할당합니다. 단, 그 크기는 사용자가 지정하게 되는데 이 것은 COMILX_US2_Start() 함수의 nBufSizeGain 매개 변수가 스캔 버퍼의 크기를 결정하게 됩니다.

LX20-시리즈 디바이스를 사용할 때 A/D 데이터의 흐름을 그림으로 표현하면 다음과 같습니다.



[LX20x 디바이스 내장 [PC 메모리 공간] [사용자의 데이터 처리]

[그림 3-1] COMI-LX20 시리즈 디바이스를 이용한 A/D 스캔 시에 데이터의 흐름도

디바이스에 내장된 FIFO 는 각 채널당 8192 개의 데이터를 담을 수 있는 크기이며 4096 개씩으로 나뉘어 더블 버퍼링(Double Buffering)방식으로 운용됩니다. DMA 전송은 각 채널당 4096 개의 데이터 블록 단위로 전송됩니다. 따라서 1 회의 DMA 전송이 이루어졌으면 각 채널당 4096 개의 데이터가 전송되었음을 의미합니다.

□ **환형 버퍼**

환형 버퍼는 한정된 버퍼에 무한히 데이터를 기록하기 위해 사용되는 것으로써, 데이터가 버퍼의 마지막 위치까지 다 채워지면 버퍼의 처음 위치부터 다시 채워나가는 방식을 말합니다.

COMILX_US2_Start()함수의 매개 변수중에서 bPauseBufFull 값을 FALSE 로 설정하면 드라이버는 스캔버퍼를 환형버퍼 형식으로 운용하여 데이터를 채워나갑니다.

※버퍼크기가 N 인 경우 스캔 버퍼에 데이터가 저장되는 예 (여기서 nBufSizeGain 값을 n 이라 하면 N = n * 4096 이 된다)

- 현재 N개의 데이터가 스캔된 경우

Buf[0]	Buf[1]	Buf[2]		Buf[N-1]
DATA (1)	DATA (2)	DATA (3)	.....	DATA (N)

- 현재 N+2개의 데이터가 스캔된 경우

Buf[0]	Buf[1]	Buf[2]		Buf[N-1]
DATA (N+1)	DATA (N+2)	DATA (3)	.....	DATA (N)

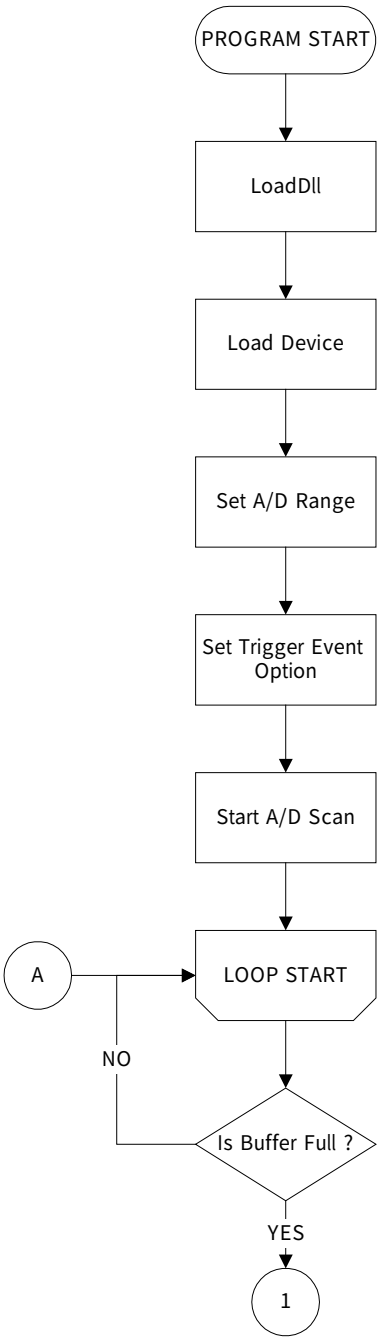
□ Frame Scan 과 Continuous Scan

본 라이브러리를 이용하여 고속으로 A/D 스캔을 수행할 때에 COMILX_US2_Start() 함수의 bPauseAtBufFull 매개 변수의 값에 따라 다음의 두 가지 방식으로 수행됩니다.

Frame Scan : 일정 크기의 데이터 블록이 스캔되면 사용자가 COMILX_US2_Resume()함수를 이용하여 재개하기 전까지 자동으로 스캔을 일시 중지합니다. 이 것은 고속으로 A/D 스캔하는 경우 데이터의 Overlap 을 방지하기 위한 방식입니다.

Continuous Scan : 스캔 버퍼를 환형 버퍼 형식으로 운용하여 사용자가 중지할 때까지 계속하여 A/D 스캔을 수행하는 방식입니다. 이 방식을 사용하면 데이터를 끊김 없이 계측할 수 있지만 데이터의 처리 속도에 따라 데이터가 Overlap 되는 경우가 발생할 수 있습니다.

☐ Frame Scan 방식을 이용할 때의 일반적인 순서도



함수명 : COMILX_LoadDll

함수명 : COMILX_LoadDevice

함수명 : COMILX_AD_SetRange

-10 ~ +10 인 경우에는 생략 가능

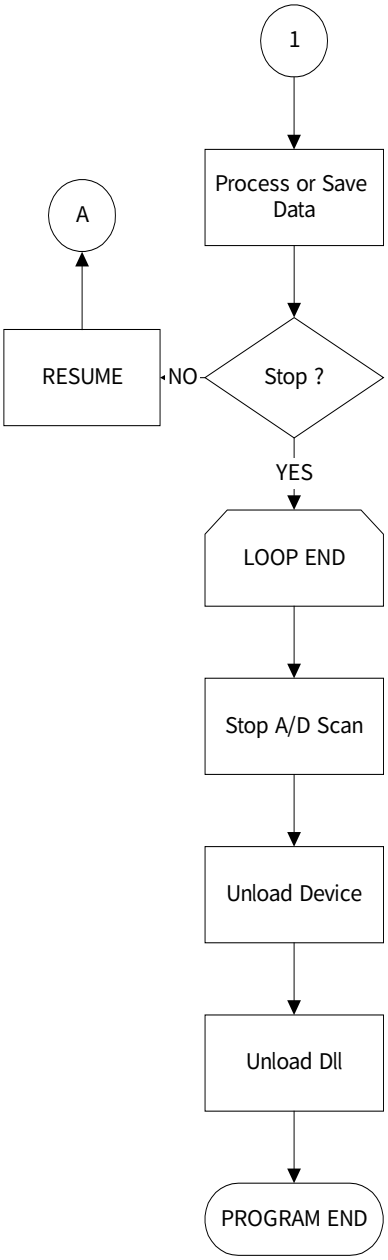
함수명 : COMILX_AD_SetTriggerEvent

T i E t 사용 안하는 경우 생략

함수명 : COMILX_US2_Start

bPauseAtBufFull 매개변수값이 1 이어야 한다.

함수명 : COMILX_US2_IsBufFull



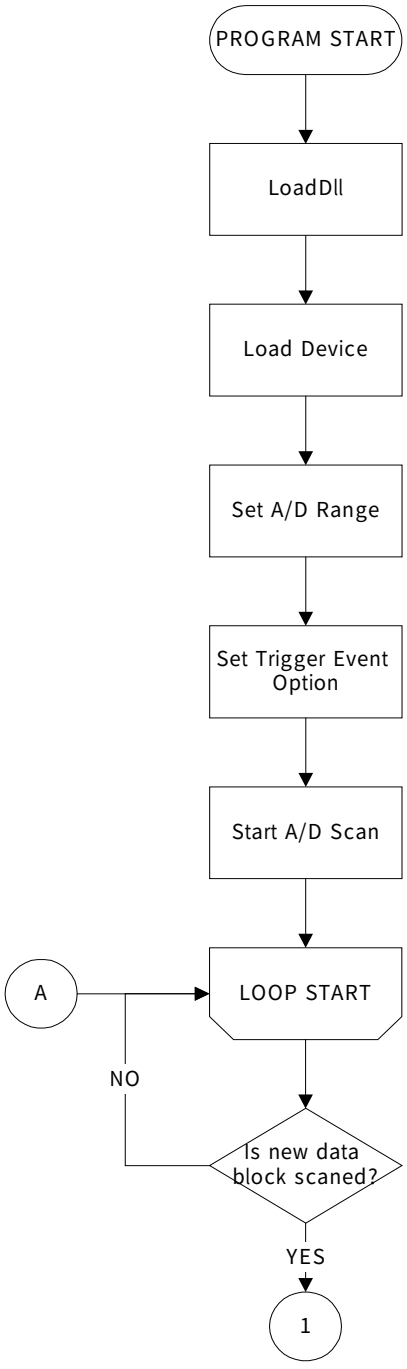
이 곳은 사용자가 스캔된 데이터를 처리하는 부분임
이 곳에서 COMILX_US2_GetBuffer 또는 COMILX US2 RetrvChannel 함수를 이용하 사용자가 STOP 시키지 않으면 COMILX_Resume 함수를 이용하여 스캔을

함수명 : COMILX_US2_Stop
이때 버퍼를 해제하지 않으면 다음에 COMILX_US2_ReleaseBuf 함수를 이용하여

함수명 : COMILX_UnloadDevice

함수명 : COMILX_UnloadDll

□ Continuous Scan 방식을 이용할 때의 순서도



함수명 : COMILX_LoadDll

함수명 : COMILX_LoadDevice

함수명 : COMILX_AD_SetRange

-10 ~ +10 인 경우에는 생략 가능

함수명 : COMILX_AD_SetTriggerEvent

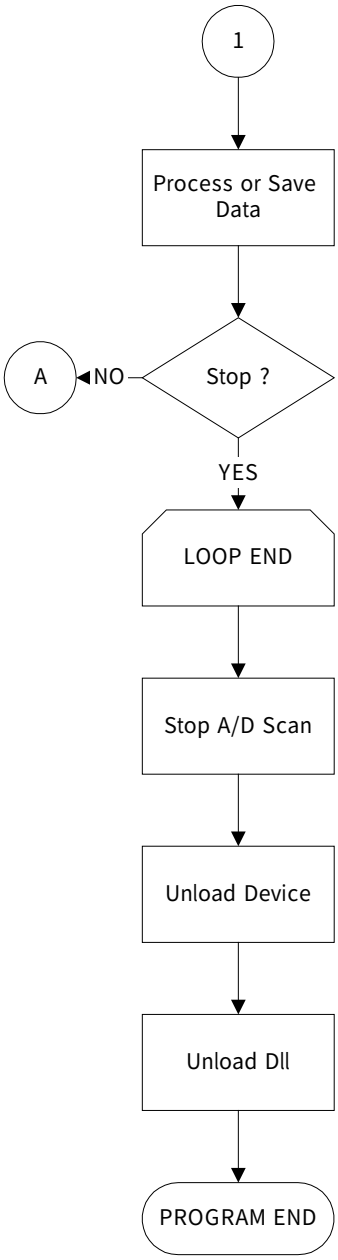
Trigger Event 사용 안하는 경우 생략

함수명 : COMILX_US2_Start

bPauseAtBufFull 매개변수값이 1 이어야 한다.

함수명 : COMILX_US2_DmaCount

DMA Count 가 증가했는지를 체크하므로써 새로운 데이터 블록이 스캔됐는지를 체크할 수 있다.



이 곳은 사용자가 스캔된 데이터를 처리하는 부분임
이 곳에서 COMILX_US2_GetBuffer 또는 COMILX US2 RetrvChannel 함수를 이용하

함수명 : COMILX_US2_Stop
이때 버퍼를 해제하지 않으면 다음에 COMILX_US2_ReleaseBuf 함수를 이용하여

함수명 : COMILX_UnloadDevice

함수명 : COMILX_UnloadDll

▣ COMILX_US2_SetTriggerEvent

함수 원형 :

void COMILX_US2_SetTriggerEvent (HANDLE hDevice, int nInputSource, int nEdgeType, int nTrgMode, float fAiRef, float fAiRefBand)

함수 설명 :

이 함수는 Trigger Event 를 사용할 것인지를 결정하고, Trigger Event 신호의 종류와 운용 방법을 설정합니다.

Trigger Event 는 A/D 스캔을 시작하는 신호를 의미합니다. Trigger Event 를 사용하지 않는 경우에는 A/D 스캔 시작 함수를 호출함과 동시에 A/D 스캔이 시작됩니다. 그러나 Trigger Event 를 사용하면 A/D 스캔 시작 함수를 호출하여도 외부에서 Trigger Event 신호가 입력되기 전까지 스캔 데이터를 버퍼에 저장하지 않습니다.

매개 변수

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

▶ *nInputSource* : Trigger Event 신호로 사용되는 신호원(Signal Source)을 설정합니다. 이 값은 다음 중 하나의 값이어야 합니다.

Value	Meaning
0 또는 TS_NONE	이 값을 지정하면 Trigger Event 를 사용하지 않음을 의미합니다. 즉, A/D Scan 시작 함수가 호출됨과 동시에 스캔 데이터는 버퍼에 저장됩니다.
1 또는 TS_ANALOG	이 값을 지정하면 A/D CH0 의 신호를 Trigger Event 의 신호원으로 사용합니다.
2 또는 TS_DIGITAL	이 값을 지정하면 Digital 신호(ON/OFF)가 Trigger Event 의 신호원으로 사용됩니다. COMI-LX20x 디바이스는 Trigger Event 전용 Digital Input 단자를 제공합니다. 자세한 사항은 H/W 매뉴얼을 참조하십시오.

▶ *nEdgeType* : Trigger Event 신호원에서 발생되는 신호가 Trigger Event 로서 동작하기 위한 신호의 상태를 설정합니다. 이 값은 다음 중 하나의 값이어야 하며 자세한 사항은 아래의 참고 항목을 참조하십시오.

Value	Meaning
0 또는 TE_POSITIVE	Positive Edge
1 또는 TE_NEGATIVE	Negative Edge

▶ *nTrgMode* : 이 값은 나중을 위하여 예약된 것이며 현재는 0 으로 지정하여야 합니다.

▶ *fAiRef* : 이 값은 nInputSource 가 TS_ANALOG 로 지정되었을 때만 의미를 갖는 값으로써, 이 때에는 A/D CH0 의 값이 레퍼런스값으로 사용됩니다.

▶ *fAiRefBand* : 이 값은 nInputSource 가 TS_ANALOG 로 지정되었을 때만 의미를 갖는 값으로써, 유효한 Trigger 신호를 검출하기 위하여 A/I Reference 의 Band 를 설정하기 위한 값입니다 (참고 항목 참조). 이 값의 단위는 A/D 입력 범위에 대한 백분율(%)입니다. 예를 들어, 만일 Input Range 가 -10 ~ +10 이고, fAiRefBand 의 값이 1 인 경우에는 Range 크기가 20 이므로 Band 크기는 20 * 0.01 = 0.2 Volt 가 됩니다.

Return 값

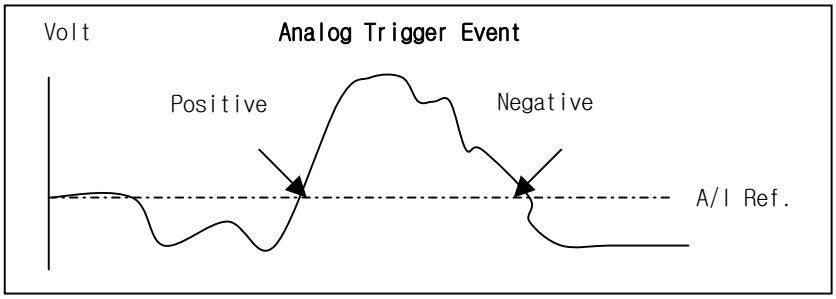
함수 수행의 성공 여부

Value	Meaning
0	함수 수행 실패
1	함수 수행 성공

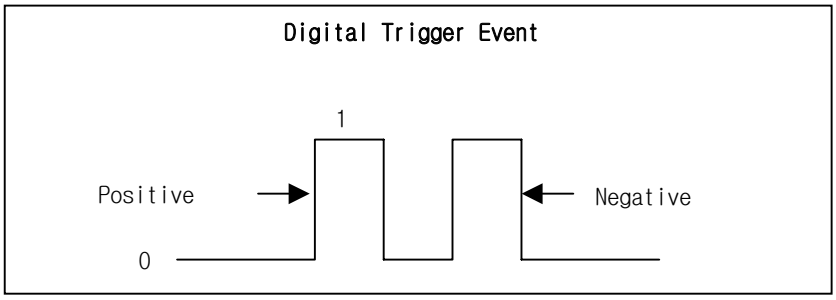
참 고

□ 컴퓨터가 처음 부팅될 때에 COMI-LX20x 디바이스는 기본적으로 Trigger Event 를 사용하지 않는 것으로 설정됩니다.

□ Edge Type 을 그림과 함께 설명하면 다음과 같습니다.



[그림 3-2] Analog Trigger Event 시에 Edge Type 의 종류

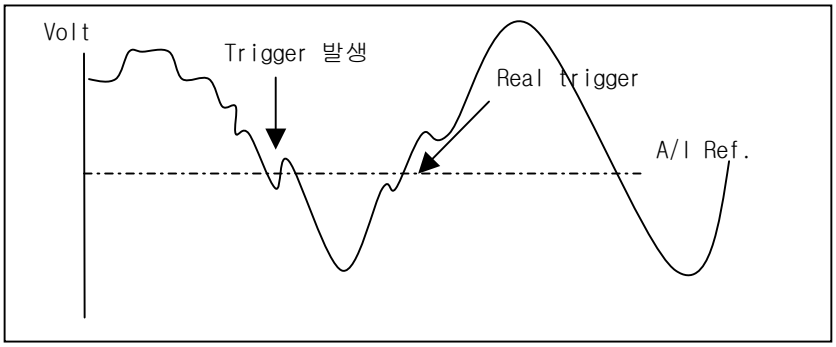


[그림 3-3] Digital Trigger Event 시에 Edge Type 의 종류

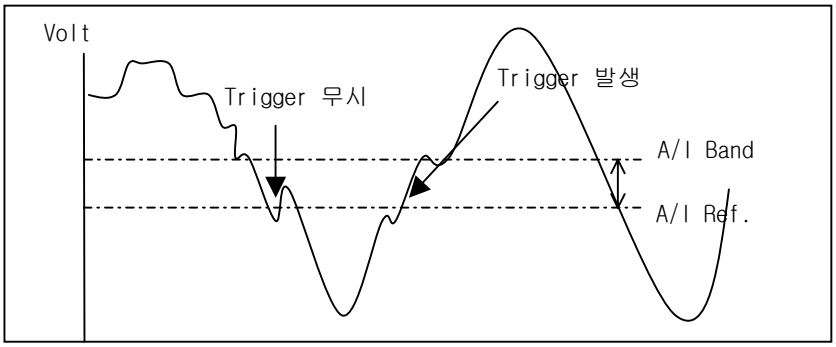
□ Analog Input Reference Band 의 역할

Analog Input Reference Band 는 Noise 등에 의하여 발생될 수 있는 잘못된 Trigger 를 방지하기 위한 것입니다. 예를 들어 [그림 3-4]와 같이 Positive Edge Trigger 모드에서 Negative Edge 순간임에도 불구하고 Noise 의 영향에 의하여 순간적으로 Positive Edge 가 발생할 수 있습니다. 이 때에는 원하지 않는 Trigger 가 발생하게 됩니다.

그러나 [그림 3-5]와 같이 Band 를 지정하면 A/I Reference 에서뿐만 아니라 Band limit 값에서도 Positive Edge 가 발생 해야만 진정한 Trigger Event 로 간주되므로 잘못된 Trigger 를 방지할 수 있습니다.



[그림 3-4] Positive Edge 방식에서 Analog Input Reference Band 를 사용하지 않아 잘못된 Trigger 가 발생되는 예



[그림 3-5] Positive Edge 방식에서 Analog Input Reference Band 를 사용하므로써 잘못된 Trigger 의 발생을 방지하는 예

사용예

1) Digital 신호가 LOW 에서 HIGH 로 되는 순간부터 스캔이 시작되도록 하는 경우.

```
// 이 이전에 COMILX_LoadDevice(…)가 수행되어 있어야 한다 //
COMILX_US2_SetTriggerEvent (hDevice, TS_DIGITAL, TE_POSITIVE, 0,
0, 0);
COMILX_US2_Start (…)
……
……
```

2) Analog Input(CH0) 신호가 5 Volt 이상이 되는 순간부터 스캔이 시작되도록 하되, 0.1 volt 의 Reference Band 크기를 갖도록 하는 경우.

```
// 이 이전에 COMILX_LoadDevice(…)가 수행되어 있어야 한다 //
COMILX_AD_SetRange (hDevice, 0, -10, 10); /* A/D 레인지에 따라 Band 폭이 달라짐 */
COMILX_US2_SetTriggerEvent (hDevice, TS_ANALOG, TE_POSITIVE, 0, 5, 0.5);
COMILX_US2_Start(…)
……
……
```

3) Trigger Event 를 사용하지 않는 경우

```
// 이 이전에 COMILX_LoadDevice(…)가 수행되어 있어야 한다 //
COMILX_US2_SetTriggerEvent(hDevice, TS_NONE, 0, 0, 0, 0);
COMILX_US2_Start(…)
……
……
```

▣ COMILX_US2_Start

함수 원형

double COMILX_US2_Start (HANDLE hDevice, int nNumChannel, int *pChanList, UINT nScanFreq, USHORT nBufSizeGain, BOOL bPauseAtBufFull)

함수 설명

이 함수는 Unlimited scan 기능을 시작합니다.

매개 변수

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

▶ *nNumChannel* : 스캔에 사용되는 A/D 채널 수를 지정합니다.

▶ *pChanList* : A/D scan 을 수행할 채널 리스트를 담고 있는 배열 또는 포인터.

▶ *nScanFreq* : A/D scan 주파수를 설정합니다. 단위는 Hz 입니다.

▶ *nBufSizeGain* : 이 값은 스캔 버퍼의 크기를 결정합니다. 그러나 이 값이 스캔 버퍼의 직접적인 크기를 지정하지는 않습니다. 실제 버퍼의 크기는 *nBufSizeGain * 4096* 개의 데이터를 저장할 수 있는 크기로 할당됩니다(여기서 4096 값은 한번의 DMA 가 수행될 때 전송되는 데이터 개수입니다). 예를 들어 채널 당 4096 개의 데이터를 저장할 수 있는 크기로 버퍼를 할당하려면 이 값을 1 로 하여야 하며, 40960 개의 데이터를 저장할 수 있도록 하려면 이 값을 10 으로 하여야 합니다.

▶ *bPauseAtBufFull* : 이 값은 스캔 버퍼에 데이터가 꽉 찬 경우에 스캔을 일시 중지할 것인지를 결정합니다. 이 값을 1 로 하면 Frame Scan, 0 으로 하면 Continuous Scan 방식으로 운용됩니다. 자세한 사항은 단원 앞 부분의 “Frame Scan 과 Continuous Scan” 설명을 참조하십시오.

Return 값

실제 스캔 주파수를 Hz 단위로 반환합니다. 스캔 주파수를 결정하는 내장 타이머가 정수값을 분주하여 주파수를 결정하므로 사용자가 지정한 스캔 주파수와 실제 스캔

주파수는 약간의 차이가 있을 수 있습니다.

참고

□ nBufSizeGain 및 bPauseBufFull 값을 설정할 때 주의 사항

① COMI-LX20x 디바이스는 각 채널당 8192 개의 데이터를 저장할 수 있는 FIFO 메모리를 가지고 있습니다. 따라서 nBufSizeGain 값을 2 로 하고 bPauseBufFull 을 TRUE 로 하면 속도와 상관없이 8 K 의 연속 데이터를 얻을 수 있습니다.

② (채널수 * 스캔 주파수) 가 5 MHz 보다 큰 경우에는 nBufSizeGain 을 2 로하고 bPauseBufFull 을 TRUE 로 하는 것이 좋습니다. 이는 DMA 속도가 스캔속도보다 느리게 되어 연속적으로 데이터를 전송할 수 없기때문입니다.

③ CH0 와 CH3 만을 사용하는 경우에 nBufSizeGain 을 4 로 하고 bPauseBufFull 을 TRUE 로 하면 자동적으로 2 개의 FIFO 를 각 채널에 할당하는 CASCADE 방식으로 운용하여 스캔 속도와 관계없이 16384 (16K)개의 연속 데이터를 얻을 수 있습니다.

④ COMILX_US2_SetTrgEvent()함수를 수행하여 트리거 이벤트를 TM_MIDDLE 로 설정한 경우에는 nBufSizeGain 을 2 로 하고 bPauseBufFull 을 TRUE 로 하여야 합니다.

▣ COMILX_US2_Resume

함수 원형

void COMILX_US2_Resume (HANDLE hDevice)

함수 설명

이 함수는 일시 중지된 A/D 스캔을 재개하여 줍니다. COMILX_US2_Start()함수에서 bPauseAtBufFull 값을 TRUE 로 지정한 경우에는 스캔 버퍼에 데이터가 다 차게 되면 스캔이 일시 중지됩니다. 이 때 사용자는 필요에 따라 스캔 데이터를 처리하고 COMILX_US2_Resume () 함수를 이용하여 스캔을 재개할 수 있습니다.

매개 변수

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

사용예

```
#define NUM_CH       4 // 채널 수 = 4 채널
#define SCAN_FREQ 100000  // 100 KHz sampling
#define BUF_SIZE_GAIN 2  // 채널당 버퍼 크기 = 2 * 4096 = 8192 개의 데이터를 담을 수 있는 공간
……
HANDLE hDevice;
int nChanList[NUM_CH]={0,1,2,3};
ULONG nCurCount;
……
……
COMILX_US2_Start(hDevice,    NUM_CH,    nChanList,    SCAN_FREQ, BUF_SIZE_GAIN, TRUE);
while(!IsStop()) // IsStop()은 강제 종료에 해당하는 가상의 함수임.
{
   nCurCount = COMILX_US2_DmaCount(hDevice);
   /* 버퍼가 차면 데이터를 처리하고, 다시 스캔을 재개한다. */
   if(nCurCount == BUF_SIZE_GAIN){
       /* ProcessData()는 스캔 데이터를 취하여 처리하는 가상의 함수 */
ProcessData(…);
       ……
       COMILX_US2_Resume(hDevice);
}
}
COMILX_US2_Stop(hDevice, TRUE);
……
```

▣ COMILX_US2_IsBufFull

함수 원형

BOOL COMILX_US2_IsBufFull (HANDLE hDevice)

함수 설명

이 함수는 지정한 크기(개수)의 스캔 버퍼에 데이터가 다 찼는지를 알려주는 함수입니다. 이 것은 COMILX_US2_Start(…)함수에서 bPauseAtBufFull 매개 변수 값을 TRUE 또는 1로 하였을 경우에만 해당하는 것입니다.

매개 변수

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

사용예

```
#define NUM_CH        4 // 채널 수 = 4 채널
#define SCAN_FREQ 100000  // 100 KHz sampling
#define BUF_SIZE_GAIN 2  // 채널당 버퍼 크기 = 2 * 4096 = 8192 개의 데이터를 담을 수 있는 공간
……
HANDLE hDevice;
int nChanList[NUM_CH]={0,1,2,3};
……
……
COMILX_US2_Start(hDevice,    NUM_CH,    nChanList,    SCAN_FREQ, BUF_SIZE_GAIN, TRUE);
while(!IsStop()) // IsStop()은 강제 종료에 해당하는 가상의 함수임.
{
   /* 버퍼가 차면 데이터를 처리하고, 다시 스캔을 재개한다. */
   if(COMILX_US2_IsBufFull(hDevice)){
       /* ProcessData()는 스캔 데이터를 취하여 처리하는 가상의 함수 */
ProcessData(…);
       ……
       COMILX_US2_Resume(hDevice);
}
}
COMILX_US2_Stop(hDevice, TRUE);
……
……
```

▣ COMILX_US2_ChangeScanFreq

함수 원형

double COMILX_US2_ChangeScanFreq (HANDLE hDevice, UINT nScanFreq)

함수 설명

이 함수는 스캔이 진행되는 중에 스캔 주파수를 변경합니다.

매개 변수

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

▶ *nScanFreq* : A/D scan 주파수를 설정합니다. 단위는 Hz 입니다.

Return 값

실제 스캔 주파수를 Hz 단위로 반환합니다. 스캔 주파수를 결정하는 내장 타이머가 정수값을 분주하여 주파수를 결정하므로 사용자가 지정한 스캔 주파수와 실제 스캔 주파수는 약간의 차이가 있을 수 있습니다.

▣ COMILX_US2_DmaCount

함수 원형 :

```
ULONG   COMILX_US2_DmaCount (HANDLE hDevice)
```

함수 설명 :

이 함수는 스캔이 시작된 후에 현재까지 몇회의 DMA 데이터블록이 전송됐는지를 알려주는 함수입니다.

매개 변수 :

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

Return 값:

현재까지 진행된 DMA 전송 횟수를 반환합니다. 1 회의 DMA 전송에 의해 각 채널당 4096 개의 데이터가 전송되므로 실제로 획득된 데이터수 수는

N = 4096 * C 가 됩니다.

여기서

N : 각 채널당 획득된 데이터 수

C : DMA 전송 횟수

▣ COMILX_US2_GetBuffer

함수 원형

short* COMILX_US2_GetBuffer (HANDLE hDevice, int chOrder)

함수 설명

이 함수는 각 채널에 대한 스캔 버퍼의 포인터를 반환합니다. 사용자는 여기서 반환된 포인터를 이용하여 스캔 데이터를 직접 취할 수 있습니다. 참고로, COMI-LX20x 디바이스는 다른 디바이스들과 달리 각 채널마다 독립적인 버퍼를 가지게 됩니다.

매개 변수

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

▶ *chOrder* : 데이터를 취하기 원하는 채널의 채널 리스트 상의 순서(0 based)입니다. 이 값은 채널 번호가 아님을 주의하여야 한다.

Return 값

해당 채널에 대한 스캔 버퍼의 포인터.

참 고

이 버퍼에 담겨진 A/D 데이터는 직접적인 Voltage 값이 아니고, 0 ~ 4095 사이의 정수값입니다. 사용자는 이 값을 A/D 입력 범위에 따라 Voltage 로 변환하여야 합니다.

사용예 1 : Frame Scan 방식을 사용할 때 COMILX_US2_GetBuffer 함수를 이용하여 데이터를 취하는 예

```
#define ADR_MIN   -10  // A/D Range 의 최소값
#define ADR_MAX   10    // A/D Range 의 최대값
#define NUM_CH        4 // 채널 수 = 4 채널
#define SCAN_FREQ 100000  // 100 KHz sampling
#define BUF_SIZE_GAIN 2  // 채널당 버퍼 크기 = 2 * 4096 = 8192 개의 데이터를 담을 수 있는 공간
#define DigitToVolt(d, vmin, vmax) (d * (vmax - vmin)/4095. + vmin)
……
```

```
HANDLE hDevice;
int i, j;
int nChanList[NUM_CH]={0,1,2,3};
short *pBuffer;
double volts[BUF_SIZE_GAIN   *4096];
ULONG nCurCount;
……
COMILX_US2_Start(hDevice,    NUM_CH,    nChanList,    SCAN_FREQ,
BUF_SIZE_GAIN, TRUE);
while(!IsStop()) // IsStop()은 가상의 함수임.
{
   nCurCount = COMILX_US2_DmaCount(hDevice);
   if(nCurCount == BUF_SIZE_GAIN){// 버퍼가 차면 데이터를 처리를 하고,
다시 스캔을 재개한다.
       for(i=0; i<NUM_CH; i++){
           pBuffer = COMILX_US2_GetBuffer(hDevice, nChanList[i]);
           for(j=0; j<BUF_SIZE_GAIN*4096; j++){
               volts[j]=DigitToVolt      (pBuffer[j],      ADR_MIN,
ADR_MAX);
}
PlotData(volts); // PlotData()함수는 가상의 함수임.
}
       ……
       COMILX_US2_Resume(hDevice);
}
}
COMILX_US2_Stop(hDevice, TRUE);
……
```

사용예 2: Continuous Scan 방식을 사용할 때 COMILX_US2_GetBuffer 함수를 사용하여 데이터를 취하는 예

```
#define ADR_MIN   -10  // A/D Range 의 최소값
#define ADR_MAX   10    // A/D Range 의 최대값
#define NUM_CH       4 // 채널 수 = 4 채널
#define SCAN_FREQ 100000  // 100 KHz sampling
#define BUF_SIZE_GAIN 10  // 채널당 버퍼 크기 = 10 * 4096 = 40960 개의
데이터를 담을 수 있는 공간
#define USER_BUF_SZIE 2*4096  // 사용자 버퍼 크기
#define  DigitToVolt(d,  vmin,  vmax)  (d  *  (vmax  –  vmin)/4095.  +
vmin)
……
HANDLE hDevice;
int i, j;
int nChanList[NUM_CH]={0,1,2,3};
short *pBuffer;
double volts[USER_BUF_SZIE];
ULONG nPrvCount, nCurCount, nNumData;
……
```

```
COMILX_US2_Start(hDevice,     NUM_CH,     nChanList,     SCAN_FREQ,
BUF_SIZE_GAIN, FALSE);
nPrvCount = 0;
while(!IsStop()) // IsStop()은 가상의 함수임.
{
   nCurCount = COMILX_US2_DmaCount(hDevice);
   // 새로운 스캔 데이터가 있는지 체크하고 있으면 데이터를 처리한다. //
   if(nCurCount > nPrvCount){
       /* 새로운 데이터의 수 계산 , 단 이 수가 사용자 버퍼 크기보다 크면 처
리할 수 없으므로 데이터수의 최대 크기는 USER_BUF_SIZE 로 제한한다. */
       nNumData = (nCurCount – nPrvCount) * 4096;
       if(nNumData > USER_BUF_SIZE)
           nNumData = USER_BUF_SZIE;
       for(i=0; i<NUM_CH; i++){
           pBuffer = COMILX_US2_GetBuffer(hDevice, nChanList[i]);
           // 버퍼상에서 현재의 데이터 블록이 시작 되는 인덱스 계산 //
           si = (nPrvCount*4096) % (BUF_SIZE_GAIN *4096);
           for(j=0; j< nNumData; j++){
              volts[j]=DigitToVolt     (pBuffer[si+j],     ADR_MIN,
ADR_MAX);
           }
           PlotData(volts, nNumData);//PlotData()함수는 가상의 함수임.
       }
       nPrvCount = nCurCount;
   }
}
COMILX_US2_Stop(hDevice, TRUE);
……
……
```

▣ COMILX_US2_RetrvChannel

함수 원형

ULONG COMILX_US2_RetrvChannel (HANDLE hDevice, int nChanOrder, ULONG nStartCount, int nMaxNumData, void *pDestBuf, TComiVarType VarType)

함수 설명

이 함수는 A/D Scan 채널 중에서 하나의 채널에 대한 데이터 블럭을 Voltage 값으로 환산하여 전달합니다. 데이터 블럭은 사용자가 지정한 nStartCount 에서부터 nMaxNumData 에서 지정한 수 만큼이 됩니다.

매개 변수

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

▶ *nChanOrder* : 데이터를 취하기 원하는 채널의 채널 리스트 상의 순서(0 based)입니다. 이 값은 채널 번호가 아님을 주의하여야 한다.

▶ *nStartCount* : 전달 받고자 하는 데이터 블럭의 시작 Scan count.

▶ *nMaxNumData* : 전달 받고자 하는 데이터 블럭의 크기(데이터 수)를 지정 합니다. 이 값이 양수이면 nStartCount 부터 이후에 스캔된 데이터 중 nMaxNumData 에서 지정한 수만큼 데이터를 전달합니다. 이 값이 음수이면 nStartCount 부터 이전에 스캔된 데이터 중 nMaxNumData 에서 지정한 수만큼 데이터를 전달합니다.

▶ *pDestBuf* : 데이터를 전달 받을 버퍼 포인터를 지정합니다. 이 버퍼는 float 형이나 double 형이어야 하며, VarType 파라미터에서 지정한 데이터형과 일치해야 합니다. 또한 이 버퍼의 크기는 nMaxNumData 에서 지정한 값보다 크거나 같아야 한다.

▶ *VarType* : pDestBuf 의 데이터 형을 지정합니다. 이 값은 다음의 두 값 중 하나이어야 합니다.

Value	Meaning

0 또는 VT_SHORT	pDestBuf 가 short 형 포인터임을 의미하며, 데이터는 Voltage 로 환산되기 이전의 정수형값으로 전달됩니다.
1 또는 VT_FLOAT	pDestBuf 가 float 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.
2 또는 VT_DOUBLE	pDestBuf 가 double 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.

Return 값

실제 전달된 데이터 수를 반환합니다. 만일 nStartCount 이후에 현재까지 스캔된 데이터 수가 nMaxNumData 에서 지정한 수 보다 작으면, 현재 스캔된 데이터까지만 전달하게됩니다.

사용예 :

☐ **사용예 1 :** Frame Scan 방식을 사용할 때 COMILX_US2_RetrvChannel 함수를 이용하여 데이터를 취하는 예

```
#define ADR_MIN   -10  // A/D Range 의 최소값
#define ADR_MAX   10    // A/D Range 의 최대값
#define NUM_CH        4 // 채널 수 = 4 채널
#define SCAN_FREQ 100000  // 100 KHz sampling
#define BUF_SIZE_GAIN 2  // 채널당 버퍼 크기 = 2 * 4096 = 8192 개의 데이터를 담을 수 있는 공간
……
HANDLE hDevice;
int i, j;
int nChanList[NUM_CH]={0,1,2,3};
double volts[BUF_SIZE_GAIN   *4096];
ULONG nCurCount;
……
COMILX_US2_Start(hDevice,    NUM_CH,    nChanList,    SCAN_FREQ, BUF_SIZE_GAIN, TRUE);
while(!IsStop()) // IsStop()은 가상의 함수임.
{
   /* 버퍼가 차면 데이터를 처리하고, 다시 스캔을 재개한다. */
   if(COMILX_US2_IsBufFull(hDevice)){
       for(i=0; i<NUM_CH; i++){
           count   =   COMILX_US2_RetrvChannel   (hDevice,   i,   1, BUF_SIZE_GAIN   *4096, volts, VT_DOUBLE);
PlotData(volts, count); // PlotData()함수는 가상의 함수임.
}
       ……
       COMILX_US2_Resume(hDevice);
```

```
}
}
COMILX_US2_Stop(hDevice, TRUE);
……
```

□ **사용예 2 :** Continuous Scan 방식을 사용할 때 COMILX_US2_RetrvChannel 함수를 사용하여 데이터를 취하는 예

```
#define ADR_MIN   -10  // A/D Range 의 최소값
#define ADR_MAX   10    // A/D Range 의 최대값
#define NUM_CH        4 // 채널 수 = 4 채널
#define SCAN_FREQ 100000  // 100 KHz sampling
#define BUF_SIZE_GAIN 10  /* 채널당 버퍼 크기 = 10 * 4096 = 40960 개의
데이터를 담을 수 있는 공간 */
#define USER_BUF_SZIE 2*4096  // 사용자 버퍼 크기

……
HANDLE hDevice;
int i, j;
int nChanList[NUM_CH]={0,1,2,3};
double volts[BUF_SIZE_GAIN   *4096];
ULONG nPrvCount, nCurCount, nNumData;
……
COMILX_US2_Start(hDevice,     NUM_CH,     nChanList,     SCAN_FREQ,
BUF_SIZE_GAIN, FALSE);
nPrvCount = 0;
while(!IsStop()) // IsStop()은 가상의 함수임.
{
   nCurCount = COMILX_US2_DmaCount(hDevice);
// 새로운 스캔 데이터가 있는지 체크하고 있으면 데이터를 처리한다. //
   if(nCurCount > nPrvCount){
       nNumData = (nCurCount – nPrvCount) * 4096;
       for(i=0; i<NUM_CH; i++){
           count    =    COMILX_US2_RetrvChannel    (hDevice,    i,
nPrvCount*4096+1, USER_BUF_SZIE, volts, VT_DOUBLE);
PlotData(volts, count); // PlotData()함수는 가상의 함수임.
}
nPrvCount = nCurCount;
}
}
COMILX_US2_Stop(hDevice, TRUE);
……
……
```

▣ COMILX_US2_Stop

함수 원형

```
void    COMILX_US2_Stop (HANDLE hDevice, BOOL bReleaseBuf)
```

함수 설명

이 함수는 A/D 스캔을 종료합니다.

매개 변수

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

▶ *bReleaseBuf* : COMILX_US2_Start()가 수행될 때 할당되었던 스캔 버퍼를 메모리 해제시킬것인지를 결정합니다. 만일 이 값을 FALSE 로 지정하면 후에 반드시 COMILX_US2_ReleaseBuf()를 사용하여 버퍼를 해제하여야 합니다. 이 값을 TRUE 로 지정하면 COMILX_US2_ReleaseBuf() 함수를 수행할 필요가 없습니다.

▣ COMILX_US2_ReleaseBuf

함수 원형 :

BOOL COMILX_US2_ReleaseBuf (HANDLE hDevice)

함수 설명 :

COMILX_US2_Start()가 수행될 때 할당되었던 스캔 버퍼를 메모리 해제 시킵니다. 이 함수는 COMILX_US2_Stop() 함수가 호출되기 전에 수행되어서는 안되며, COMILX_US2_Stop() 함수를 호출할 때 두 번째 파라미터를 TRUE 로 지정했을 때는 사용하지 않아도 됩니다.

매개 변수 :

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

Return 값

Value	Meaning
0	함수 수행 실패
1	함수 수행 성공

3.5 아날로그 출력(Analog Output)

이 단원에서는 Analog Output 에 관한 함수를 소개합니다. COMIDAS 에서는 두 가지 형태의 Analog Output 기능이 있습니다.

첫 번째는 일반적인 Analog Output 기능으로써 사용자가 지정한 전압을 출력하는 기능입니다.

두 번째는 Waveform Generation 기능입니다. Waveform Generation 기능은 Sine Wave 또는 Square Wave 등과 같이 사용자가 지정하는 주기성을 가지는 신호를 자동으로 생성해주는 기능입니다.

Analog Output 과 관련된 함수들의 리스트는 다음과 같습니다.

함수명	각 보드별 지원 여부								
	LX101	LX102	LX103	LX201	LX202	LX203	LX301	LX401	LX501
COMILX_DA_Out	V	V	V				V		
COMILX_WFM_Start	V	V	V				V		
COMILX_WFM_Reload	V	V	V				V		
COMILX_WFM_RateChange	V	V	V				V		
COMILX_WFM_GetCurPos	V	V	V				V		
COMILX_WFM_GetCurLoops	V	V	V				V		
COMILX_WFM_Stop	V	V	V				V		

[표 3-5] Analog Output 관련 함수 리스트 및 각 보드별 지원 여부

3.5.1. 일반적인 Analog Output 함수

▣ COMILX_DA_Out

함수 원형

BOOL **COMILX_DA_Out** (HANDLE hDevice, int ch, float volt)

함수 설명

이 함수는 지정한 Analog Output 채널에 지정한 Voltage를 출력합니다.

매개 변수

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

▶ ***ch*** : Analog Output 채널번호. 채널 번호는 0부터 시작합니다.

▶ ***volt*** : Analog Output 출력 Voltage.

Return 값

Value	Meaning
0	함수 수행 실패
1	함수 수행 성공

3.5.2. Waveform Generation

Waveform Generation 기능은 Sine Wave 또는 Square Wave 등과 같이 주기성을 가지는 신호를 아날로그 출력을 이용하여 자동으로 생성해주는 기능입니다. Waveform 데이터는 사용자가 배열로 지정하도록 되어 있으며 한 주기의 Waveform 을 구성하는 데이터수는 4096 개 이하에서 자유롭게 설정할 수 있습니다. Waveform Generation 기능을 지원하는 디바이스는 사용자가 지정하는 Waveform 데이터를 디바이스에 내장된 FIFO 메모리에 로드한 후 사용자가 지정한 출력 주파수에 따라 내부 타이머를 이용하여 반복적으로 아날로그 출력을 업데이트합니다. 사용자는 출력되는 Waveform 의 횟수를 제한할 수도 있습니다.

▣ COMILX_WFM_Start

함수 원형

long COMILX_WFM_Start (HANDLE hDevice, int ch, float *pDataBuffer, UINT nNumData, UINT nPPS, int nMaxLoops)

함수 설명

Waveform Generation 을 시작합니다.

매개 변수

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

▶ *ch* : Analog Output 채널번호. 채널 번호는 0 부터 시작합니다.

▶ *pDataBuffer* : Waveform 데이터를 담은 버퍼의 주소값(포인터)

▶ *nNumData* : 버퍼에 담겨진 데이터의 수. 이 값은 4096 보다는 작거나 같아야 합니다.

▶ *nPPS* : Waveform Generation 의 주기를 결정합니다. 이 값은 Points/Second 입니다. 예를 들어 100 개의 데이터로 한 주기를 구성하였다면 10Hz 의 신호를 만들기 위

해서는 nPPS 는 1000 이 되어야 합니다.

▶ *nMaxLoops* : 이 값이 0 보다 크면 생성되는 Wave 신호의 수를 제한합니다. 이 값이 0 이면 COMILX_WFM_Stop()함수가 수행되기 전까지 계속하여 Wave 신호를 생성합니다.

Return 값

실제로 설정되는 Points/Second 를 반환합니다. 사용자가 지정한 PPS 와 실제로 설정되는 PPS 는 약간의 차이가 있을 수 있습니다. 이 값이 0 보다 작으면 수행도중 에러가 발생하였음을 의미합니다.

사용예

□ 10 KHz, -5 ~ 5 Volt, Square Wave 신호를 계속 발생시킬 때의 예.

```
float DataBuffer[2]={-5, 5};
fActFreq = COMILX_WFM_Start (hDevice, CH0, DataBuffer, 2, 10000
* 2, 0);
printf("Waveform Generation Starts!\n");
printf("Actual Freq(Hz) = %f\n", fActFreq);
```

□ 1 KHz, -5 ~ 5 Volt, Sine Wave 신호를 계속 발생시킬 때의 예.

```
#define NUM_DATA    50
float DataBuffer[NUM_DATA];
rad = 2*3.141592/ NUM_DATA;
for(int i=0; i<NUM_DATA; i++)
 DataBuffer[i] = 5 * sin(i*rad);
fActFreq = COMILX_WFM_Start (hDevice, CH0, DataBuffer, NUM_DATA,
1000 * NUM_DATA, 0);
printf("Waveform Generation Starts!\n");
printf("Actual Freq(Hz) = %f\n", fActFreq);
```

□ 10 KHz, -5 ~ 5 Volt, Square Wave 신호를 1000 개 발생시킬 때의 예.

```
float DataBuffer[2]={-5, 5};
fActFreq = COMILX_WFM_Start (hDevice, CH0, DataBuffer, 2, 10000
* 2, 1000);
```

▣ COMILX_WFM_Reload

함수 원형 :

BOOL COMILX_WFM_Reload (HANDLE hDevice, int ch, float *pDataBuffer, UINT nNumData)

함수 설명 :

Waveform Generation 이 진행되고 있는 중에 주기(Wave 신호) 데이터를 변경합니다.

매개 변수 :

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

▶ ***ch*** : Analog Output 채널번호. 채널 번호는 0 부터 시작합니다.

▶ ***pDataBuffer*** : Waveform 데이터를 담은 버퍼의 주소값(포인터)

▶ ***nNumData*** : 버퍼에 담겨진 데이터의 수

▣ COMILX_WFM_RateChange

함수 원형

long COMILX_WFM_RateChange (HANDLE hDevice, int ch, ULONG nPPS)

함수 설명

Waveform Generation 이 진행되고 있는 중에 주파수(PPS)를 변경합니다.

매개 변수

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

▶ *ch* : Analog Output 채널번호. 채널 번호는 0 부터 시작합니다.

▶ *nPPS* : Points/Second. COMILX_WFM_Start 함수 참조.

Return 값

실제로 설정되는 Points/Second 를 반환합니다. 사용자가 지정한 PPS 와 실제로 설정되는 는 약간의 차이가 있을 수 있습니다. 이 값이 0 보다 작으면 수행도중 에러가 발생하였음을 의미합니다.

▣ COMILX_WFM_GetCurPos

함수 원형

long COMILX_WFM_GetCurPos (HANDLE hDevice, int ch)

함수 설명

현재 출력되고 있는 주기 데이터의 위치를 반환합니다. 즉, 현재 출력되고 있는 데이터 포인트가 주기 데이터의 몇 번째 데이터인지를 알려줍니다.

매개 변수

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

▶ *ch* : Analog Output 채널번호. 채널 번호는 0 부터 시작합니다.

Return 값

현재 출력되고 있는 주기 데이터의 인덱스(Index).

▣ COMILX_WFM_GetCurLoops

함수 원형

long COMILX_WFM_GetCurLoops (HANDLE hDevice, int ch)

함수 설명

현재 남아있는 Wave 신호의 주기 수를 반환합니다.

매개 변수

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

▶ *ch* : Analog Output 채널번호. 채널 번호는 0 부터 시작합니다.

Return 값

이 함수는 현재 남아있는 Wave 신호의 주기 수를 반환합니다. 예를 들어 nMaxLoops 를 1000 으로 하였을 때 이 함수가 100 을 반환한다면 현재까지 900 회의 Wave 신호가 발생하였으며, 100 회의 Wave 신호가 남았음을 의미합니다.

▣ COMILX_WFM_Stop

함수 원형

```
void    COMILX_WFM_Stop (HANDLE hDevice, int ch)
```

함수 설명

Waveform Generation 을 종료합니다.

매개 변수

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

▶ ***ch*** : Analog Output 채널번호. 채널 번호는 0 부터 시작합니다.

3.6 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.

㈜커미조아에서 제공하는 COMI-LX 시리즈 디바이스는 크게 두 가지로 구분되는 디지털 입출력 방식을 제공합니다.

하나는 일반적인 디지털 입출력 기능입니다. 일반적인 디지털 입출력은 PC 에 장착된 PCI 보드에서 디지털 입출력을 직접제어하는 것을 의미하며 이와 관련된 함수들은 [3.6.1 일반적인 디지털 입출력] 단원에서 설명됩니다. COMI-LX402 Serial DIO Master Board 를 제외한 대부분의 COMI-LX 시리즈 디바이스들이 일반적인 디지털 입출력 기능을 지원하며 각 디바이스별 일반적인 디지털 입출력 기능 지원 여부는 [표 3-6]을 참조하십시오.

또 하나의 디지털 입출력 방식은 시리얼 통신(RS-422)을 이용하여 디지털 입출력을 제어하는 방식입니다. 이 방식은 PC 에 장착되어 시리얼 통신을 관장하는 마스터보드(COMI-LX402 보드)와 외부에 설치되어 실제 디지털 입출력을 제어하는 터미널 모듈(COMI-LXTM4A)이 RS-422 시리얼 통신으로 연결되어 제어되는 방식입니다. 이 방식은 사용자가 프로그램 상에서 디지털 입출력 명령을 수행하면 마스터보드가 해당 터미널 모듈에 시리얼통신을 이용하여 명령을 전달하고 해당 터미널 모듈이 디지털 입출력 명령을 수행하는 메카니즘을 사용합니다. 이 방식은 하나의 마스터보드에 최대 16 개까지 터미널모듈을 확장하여 사용할 수 있다는 큰 장점이 있습니다. 이와 관련된 함수들은 [3.6.2 시리얼 통신을 이용한 디지털 입출력] 단원에서 설명됩니다.

3.6.1 일반적인 디지털 입출력

이 단원에서는 일반적인 디지털 입출력 기능에 관련된 함수들을 소개합니다. 일반적인 디지털 입출력은 PC 에 장착된 PCI 보드에서 디지털 입출력을 직접제어하는 것을 의미합니다. COMI-LX402 Serial DIO Master Board 를 제외한 대부분의 COMI-LX 시리즈 디바이스들이 일반적인 디지털 입출력 기능을 지원하며 각 디바이스별 일반적인 디지털 입출력 기능 지원 여부는 [표 3-6]을 참조하십시오.

함수명	각 보드별 적용 여부					
	LX10x	LX20x	LX301	LX401	LX402	LX501
COMILX_DIO_SetUsage	V	V	V	V		
COMILX_DI_GetOne	V	V	V	V		V
COMILX_DI_GetAll	V	V	V	V		V
COMILX_DO_PutOne	V	V	V	V		V
COMILX_DO_PutAll	V	V	V	V		V

[표 3-6] Digital Input/Output 에 관련된 함수 리스트 및 각 보드별 지원 여부

▣ COMILX_DIO_SetUsage

함수 원형

void COMILX_DIO_SetUsage (HANDLE hDevice, int usage)

함수 설명

이 함수는 Digital Input/Output 단자의 용도를 설정합니다. COMILX-시리즈 디바이스는 Digital Input/Output 단자가 따로 구분되어 있지 않고 필요에 따라 용도를 설정할 수 있도록 되어 있습니다. 예를 들어 COMI-LX201 보드에는 8 개의 Digital Input/Output 단자가 있는데, 8 개의 단자를 모두 Input 전용으로 사용하거나 Output 전용으로 사용할 수 있습니다.

매개 변수

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

▶ ***usage*** : DIO 단자의 용도를 설정합니다. 이 값은 다음의 세 값 중 하나이어야 합니다.

Value	Meaning
0 또는 DI_ONLY	전 채널을 Digital Input 채널로 사용합니다. 이 때 채널 수는 디바이스에 따라 다르므로 H/W 매뉴얼을 참조하십시오.
1 또는 DI_DO	전 채널의 반은 Digital Input 으로, 나머지 반은 Digital Output 으로 사용합니다. 단, COMI-LX20 시리즈에서는 이 모드를 지원하지 않습니다.
2 또는 DO_DI	전 채널의 반은 Digital Output 으로, 나머지 반은 Digital Input 으로 사용합니다. 단, COMI-LX20 시리즈에서는 이 모드를 지원하지 않습니다.
3 또는 DO_ONLY	전 채널을 Digital Output 채널로 사용합니다. 이 때 채널 수는 디바이스에 따라 다르므로 H/W 매뉴얼을 참조하십시오.

참 고 :

□ D/I 와 D/O 채널 번호에 관하여

DIO 단자의 용도 설정에 따라 디지털 입력과 디지털 출력 채널 번호가 어떻게 되는지에 대해 혼동될 수 있습니다. 디지털 입력과 출력의 채널번호는 언제나 각각 0 번부터 시작합니다. 예를 들어 COMI-LX101 보드는 16 채널의 DIO 단자를 제공하는데 이를 DI_DO 로 설정하였다면 DIO0 ~ DIO7 의 단자는 디지털 입력 CH0 ~ CH7 로 사용되고 DIO8 ~ DIO15 의 단자는 디지털 출력 CH0 ~ CH7 로 사용됩니다. 반대로 DO_DI 모드로 설정하였다면 DIO0 ~ DIO7 의 단자는 디지털 출력 CH0 ~ CH7 로 사용되고 DIO8 ~ DIO15 의 단자는 디지털 입 력 CH0 ~ CH7 로 사용됩니다.

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_LX101
#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();

    COMILX_DIO_SetUsage(hDevice, DI_DO);
    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_GetOne

함수 원형 :

int COMILX_DI_GetOne (HANDLE hDevice, int ch)

함수 설명 :

이 함수는 지정한 Digital Input 채널의 Status를 반환합니다.

매개 변수 :

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

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

Return

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_LX101
#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();

    COMILX_DIO_SetUsage(hDevice, DI_DO);
    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.

Return 값

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_LX101

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();

    COMILX_DIO_SetUsage(hDevice, DI_DO);

    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_LX101
#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();

    COMILX_DIO_SetUsage(hDevice, DI_DO);
    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_LX101

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();

    COMILX_DIO_SetUsage(hDevice, DI_DO);

    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();
}
```

3.6.2 시리얼 통신을 이용한 디지털 입출력

이 단원에서는 시리얼 통신을 이용한 디지털 입출력 기능에 관련된 함수들을 소개합니다. 시리얼 통신을 이용한 디지털 입출력 방식은 시리얼 통신(RS-422)을 이용하여 디지털 입출력을 제어하는 방식입니다. 이 방식은 PC 에 장착되어 시리얼 통신을 관장하는 마스터보드(COMI-LX402 보드)와 외부에 설치되어 실제 디지털 입출력을 제어하는 터미널 모듈(COMI-STM4A)이 RS-422 시리얼 통신으로 연결되어 제어되는 방식입니다. 이 방식은 사용자가 프로그램 상에서 디지털 입출력 명령을 수행하면 마스터보드가 해당 터미널 모듈에 시리얼통신을 이용하여 명령을 전달하고 해당 터미널 모듈이 디지털 입출력 명령을 수행하는 메카니즘을 사용합니다. 이 방식은 하나의 마스터보드에 최대 16 개까지 터미널모듈을 확장하여 사용할 수 있다는 큰 장점이 있습니다.

각 COMI-STM4A 터미널 모듈은 16 채널의 디지털 입출력 채널을 제공하며, 하나의 COMI-ST401 마스터 보드에 16 개의 COMI-STM4A 터미널 모듈이 확장되어 연결될 수 있어서 하나의 마스터 보드가 총 256 개의 디지털 입출력 채널을 제어할 수 있습니다.

함수명	각 보드별 적용 여부					
	LX10x	LX20x	LX301	LX401	LX402	LX501
COMILX_SDIO_InitComm					V	
COMILX_SDIO_CheckModule					V	
COMILX_SDIO_CheckModule					V	
COMILX_SDIO_SetDioUsage					V	
COMILX_SDIO_ReadLowByte					V	
COMILX_SDIO_ReadHighByte					V	
COMILX_SDIO_WriteLowByte					V	
COMILX_SDIO_WriteHighByte					V	

[표 3-7] Digital Input/Output 에 관련된 함수 리스트 및 각 보드별 지원 여부

▣ COMILX_SDIO_InitComm

함수 원형 :

BOOL COMILX_SDIO_InitComm (HANDLE hDevice)

함수 설명 :

이 함수는 COMI-LX402 마스터 보드의 통신 포트를 초기화합니다. 일반적으로 컴퓨터가 부팅되면서 통신 초기화는 자동으로 이루어집니다. 따라서 사용자는 통신초기화를 별도로 하지 않아도 상관은 없으나 프로그램 시작부분에서 통신 초기화를 해주는 것이 좋습니다.

매개 변수 :

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

Return

함수 수행의 성공 여부

Value	Meaning
0	실패
1	성공

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_ST401, 0);
   if(hDevice == INVALID_HANDLE_VALUE)
       exit(-1); // Load Device Failure

   // RS-422 통신 초기화 //
   COMILX_SDIO_InitComm (hDevice);
```

```
    // 여기에서 필요한 SDIO 함수들을 수행한다. //
    // …… //

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

▣ COMILX_SDIO_CheckModule

함수 원형 :

BOOL COMILX_SDIO_CheckModule (HANDLE hDevice, int nModuleNo)

함수 설명 :

이 함수는 지정한 주소값(모듈 번호)을 가지는 COMI-STM4A 디지털 입출력 터미널 모듈이 COMI-LX402 마스터 보드에 현재 연결되어 있는지를 체크하는 함수입니다.

매개 변수 :

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

▶ *nModuleNo* : 검색하고자 하는 COMI-STM4A 디지털 입출력 터미널 모듈 번호를 지정합니다. 모듈 번호는 0 ~ 15 까지 지정할 수 있으며 해당 COMI-STM4A 터미널 모듈에서 점퍼로 설정되는 모듈번호와 일치하여야 합니다.

Return

지정한 COMI-STM4A 터미널 모듈이 연결되있는지를 나타내는 값

Value	Meaning
0	연결되어 있지 않음
1	연결되어 있음

예 제

다음의 예제는 **0** 번 모듈부터 **15** 번 모듈까지 모두 검색하여 각 모듈의 연결상태를 화면에 표시해주는 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_ST401, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    // RS-422 통신 초기화 //
    COMILX_SDIO_InitComm (hDevice);

    // 여기에서 필요한 SDIO 함수들을 수행한다. //
    for(int i=0; i<16; i++){
        if(COMILX_SDIO_CheckModule(hDevice, i))
            printf("Module #%d : 연결됨\n", i);
        else
            printf("Module #%d : 연결되지 않음\n", i);
    }

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

▣ COMILX_SDIO_SetDioUsage

함수 원형 :

```
BOOL    COMILX_SDIO_SetDioUsage (HANDLE hDevice, int nModuleNo, int nUsage)
```

함수 설명 :

이 함수는 지정한 COMI-STM4A 터미널 모듈의 Digital Input/Output 단자의 용도를 설정합니다. 각 COMI-STM4A 터미널 모듈은 16 개의 디지털 입출력 채널을 제공하며 16 채널의 입출력 모드를 4 가지 방법으로 설정할 수 있습니다.

매개 변수

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

▶ ***nModuleNo*** : 설정하고자 하는 COMI-STM4A 디지털 입출력 터미널 모듈 번호를 지정합니다. 모듈 번호는 0 ~ 15 까지 지정할 수 있으며 해당 COMI-STM4A 터미널 모듈에서 점퍼로 설정되는 모듈번호와 일치하여야 합니다.

▶ ***nUsage*** : DIO 단자의 용도를 설정합니다. 이 값은 다음의 값 중 하나이어야 합니다.

Value	Meaning
0 또는 DI_ONLY	전 채널을 디지털 입력 채널로 사용합니다.
1 또는 DI_DO	CH0~CH7 : 디지털 입력, CH8~CH15 : 디지털 출력
2 또는 DO_DI	CH0~CH7 : 디지털 출력, CH8~CH15 : 디지털 입력
3 또는 DO_ONLY	전 채널을 디지털 출력 채널로 사용합니다.

Return

함수 수행의 성공 여부

Value	Meaning
0	실패
1	성공

예 제

다음의 예제는 0 번 모듈의 디지털 입출력 모드를 DI_DO 로 설정하고, CH0~CH7 로부터 디지털 입력 상태를 읽어들여 각 채널의 상태를 화면에 표시하고 이를 다시 디지털 출력(CH8~CH15)으로 내보내는 예이다.

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

#define MOD0  0

// GetBitFromByte() : 바이트 데이터의 특정 비트의 값을 반환하는 함수 //
int GetBitFromByte(unsigned char Byte, int nBitNo)
{
   return((Byte>>nBitNo) & 0x1);
}

// SetBitOfByte () : 바이트 데이터의 특정 비트의 값을 변경하는 함수 //
unsigned char SetBitOfByte(unsigned char Byte, int nBitNo, int
nState)
{
   if(nState)// 비트값을 1 로 설정하는 경우
       Byte |= (1<<nBitNo);
   else // 비트값을 0 으로 설정하는 경우
       Byte &= ~(1<<nBitNo);
   return (Byte);
}


void main()
{
   unsigned char di_byte, do_byte;
   int di_each[16];

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

   HANDLE hDevice = COMILX_LoadDevice(COMI_ST401, 0);
   if(hDevice == INVALID_HANDLE_VALUE)
       exit(-1); // Load Device Failure

   // RS-422 통신 초기화 //
   COMILX_SDIO_InitComm (hDevice);
   // 제어대상 터미널 모듈이 연결되어 있는지 체크 //
   if(!COMILX_SDIO_CheckModule(hDevice, MOD0))
       exit(-1); // 제어대상 모듈이 연겨되지 않았으므로 프로그램 종료 //

   COMILX_SDIO_SetDioUsage (hDevice, MOD0, DI_DO);
```

```
    while (!_kbhit()){
        di_byte = COMILX_SDIO_ReadLowByte(hDevice, MOD0);
        for(int i=0; i<8; i++){
            di_each[i] = GetBitFromByte(di_byte, i);
        }
        printf("D/I States(CH0~CH7) = %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]);
        // 다음의 구문은 읽어들인 CH0~CH7 의 D/I 상태를 CH8~CH15 를    //
        // 통하여 D/O 출력으로 내보내는 것이다. 사실 do_byte = di_byte//
        // 를 대입하면 되지만 디지털 출력 각 채널을 개별적으로 설정하는 것//
        // 을 예로 보이기 위해서 실제로는 불필요한 아래의 for 루프를     //
        // 사용한 것이다.                                              //
        do_byte = 0;
        for(i=0; i<8; i++){
            do_byte = SetBitOfByte(do_byte, i, di_each[i]);
        }
        COMILX_SDIO_WriteHighByte(hDevice, MOD0, do_byte);
    }

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

▣ COMILX_SDIO_ReadLowByte

함수 원형 :

BYTE COMILX_SDIO_ReadLowByte (HANDLE hDevice, int nModuleNo)

함수 설명 :

이 함수는 지정한 COMI-STM4A 터미널 모듈의 0 번 채널부터 7 번 채널까지의 현재 입력 또는 출력 상태를 읽어들입니다. 이 함수는 CH0 ~ CH7 의 채널 그룹이 디지털 입력용으로 설정되었을때뿐 아니라 디지털 출력용으로 설정된 경우에도 사용할 수 있습니다. 만일 디지털 출력용으로 설정된 경우에 이 함수를 사용하시면 현재 CH0 ~ CH7 의 출력 상태를 반환받을 수 있습니다.

매개 변수

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

▶ ***nModuleNo*** : 설정하고자 하는 COMI-STM4A 디지털 입출력 터미널 모듈 번호를 지정합니다. 모듈 번호는 0 ~ 15 까지 지정할 수 있으며 해당 COMI-STM4A 터미널 모듈에서 점퍼로 설정되는 모듈번호와 일치하여야 합니다.

Return

지정한 COMI-STM4A 터미널 모듈의 0 번 채널부터 7 번 채널까지의 현재 입력 또는 출력 상태. 이 값은 8 비트값으로써 각 비트의 상태가 각 채널의 상태를 나타냅니다.

예 제

다음의 예제는 **0** 번 모듈의 디지털 입출력 모드를 **DI_DO** 로 설정하고, **CH0~CH7** 로부터 디지털 입력 상태를 읽어들여 각 채널의 상태를 화면에 표시하고 이를 다시 디지털 출력**(CH8~CH15)**으로 내보내는 예이다.

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

#define MOD0  0

// GetBitFromByte() : 바이트 데이터의 특정 비트의 값을 반환하는 함수 //
```

```
int GetBitFromByte(unsigned char Byte, int nBitNo)
{
   return((Byte>>nBitNo) & 0x1);
}

// SetBitOfByte () : 바이트 데이터의 특정 비트의 값을 변경하는 함수 //
unsigned char SetBitOfByte(unsigned char Byte, int nBitNo, int
nState)
{
   if(nState)// 비트값을 1 로 설정하는 경우
       Byte |= (1<<nBitNo);
   else // 비트값을 0 으로 설정하는 경우
       Byte &= ~(1<<nBitNo);
   return (Byte);
}


void main()
{
   unsigned char di_byte, do_byte;
   int di_each[16];

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

   HANDLE hDevice = COMILX_LoadDevice(COMI_ST401, 0);
   if(hDevice == INVALID_HANDLE_VALUE)
       exit(-1); // Load Device Failure

   // RS-422 통신 초기화 //
   COMILX_SDIO_InitComm (hDevice);
   // 제어대상 터미널 모듈이 연결되어 있는지 체크 //
   if(!COMILX_SDIO_CheckModule(hDevice, MOD0))
       exit(-1); // 제어대상 모듈이 연겨되지 않았으므로 프로그램 종료 //

   COMILX_SDIO_SetDioUsage (hDevice, MOD0, DI_DO);

   while (!_kbhit()){
       di_byte = COMILX_SDIO_ReadLowByte(hDevice, MOD0);
       for(int i=0; i<8; i++){
           di_each[i] = GetBitFromByte(di_byte, i);
       }
       printf("D/I States(CH0~CH7) = %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]);
       // 다음의 구문은 읽어들인 CH0~CH7 의 D/I 상태를 CH8~CH15 를    //
       // 통하여 D/O 출력으로 내보내는 것이다. 사실 do_byte = di_byte//
       // 를 대입하면 되지만 디지털 출력 각 채널을 개별적으로 설정하는 것//
       // 을 예로 보이기 위해서 실제로는 불필요한 아래의 for 루프를     //
       // 사용한 것이다.                                              //
       do_byte = 0;
```

```
            for(int i=0; i<8; i++){
                do_byte = SetBitOfByte(do_byte, i, di_each[i]);
            }
            COMILX_SDIO_WriteHighByte(hDevice, MOD0, do_byte);
        }

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

▣ COMILX_SDIO_ReadHighByte

함수 원형 :

```
BYTE    COMILX_SDIO_ReadHighByte (HANDLE hDevice, int nModuleNo)
```

함수 설명 :

이 함수는 지정한 COMI-STM4A 터미널 모듈의 8 번 채널부터 15 번 채널까지의 현재 입력 또는 출력 상태를 읽어들입니다. 이 함수는 CH8 ~ CH15 의 채널 그룹이 디지털 입력용으로 설정되었을 때뿐 아니라 디지털 출력용으로 설정된 경우에도 사용할 수 있습니다. 만일 디지털 출력용으로 설정된 경우에 이 함수를 사용하시면 현재 CH8 ~ CH15 의 출력 상태를 반환받을 수 있습니다.

매개 변수

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

▶ ***nModuleNo*** : 설정하고자 하는 COMI-STM4A 디지털 입출력 터미널 모듈 번호를 지정합니다. 모듈 번호는 0 ~ 15 까지 지정할 수 있으며 해당 COMI-STM4A 터미널 모듈에서 점퍼로 설정되는 모듈번호와 일치하여야 합니다.

Return

지정한 COMI-STM4A 터미널 모듈의 8 번 채널부터 15 번 채널까지의 현재 입력 또는 출력 상태. 이 값은 8 비트값으로써 각 비트의 상태가 각 채널의 ON/OFF 상태를 나타냅니다.

예 제

다음의 예제는 **0** 번 모듈의 디지털 입출력 모드를 **DO_DI** 로 설정하고, **CH8~CH15** 로부터 디지털 입력 상태를 읽어들여 각 채널의 상태를 화면에 표시하고 이를 다시 디지털 출력**(CH0~CH7)**으로 내보내는 예입니다.

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

#define MOD0  0
```

```
// GetBitFromByte() : 바이트 데이터의 특정 비트의 값을 반환하는 함수 //
int GetBitFromByte(unsigned char Byte, int nBitNo)
{
   return((Byte>>nBitNo) & 0x1);
}

// SetBitOfByte () : 바이트 데이터의 특정 비트의 값을 변경하는 함수 //
unsigned char SetBitOfByte(unsigned char Byte, int nBitNo, int
nState)
{
   if(nState)// 비트값을 1 로 설정하는 경우
       Byte |= (1<<nBitNo);
   else // 비트값을 0 으로 설정하는 경우
       Byte &= ~(1<<nBitNo);
   return (Byte);
}


void main()
{
   unsigned char di_byte, do_byte;
   int di_each[16];

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

   HANDLE hDevice = COMILX_LoadDevice(COMI_ST401, 0);
   if(hDevice == INVALID_HANDLE_VALUE)
       exit(-1); // Load Device Failure

   // RS-422 통신 초기화 //
   COMILX_SDIO_InitComm (hDevice);
   // 제어대상 터미널 모듈이 연결되어 있는지 체크 //
   if(!COMILX_SDIO_CheckModule(hDevice, MOD0))
       exit(-1); // 제어대상 모듈이 연겨되지 않았으므로 프로그램 종료 //

   COMILX_SDIO_SetDioUsage (hDevice, MOD0, DI_DO);

   while (!_kbhit()){
       di_byte = COMILX_SDIO_ReadHighByte(hDevice, MOD0);
       for(int i=0; i<8; i++){
           di_each[i] = GetBitFromByte(di_byte, i);
       }
       printf("D/I States(CH7~CH8) = %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]);
       // 다음의 구문은 읽어들인 CH0~CH7 의 D/I 상태를 CH8~CH15 를     //
       // 통하여 D/O 출력으로 내보내는 것이다. 사실 do_byte = di_byte//
       // 를 대입하면 되지만 디지털 출력 각 채널을 개별적으로 설정하는 것//
       // 을 예로 보이기 위해서 실제로는 불필요한 아래의 for 루프를     //
```

```
            // 사용한 것이다.                                                    //
            do_byte = 0;
            for(int i=0; i<8; i++){
                do_byte = SetBitOfByte(do_byte, i, di_each[i]);
            }
            COMILX_SDIO_WriteLowByte(hDevice, MOD0, do_byte);
        }

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

▣ COMILX_SDIO_WriteLowByte

함수 원형 :

BOOL COMILX_SDIO_WriteLowByte (HANDLE hDevice, int nModuleNo, BYTE bValue)

함수 설명 :

이 함수는 지정한 COMI-STM4A 터미널 모듈의 CH0 ~ CH7 에 디지털 출력을 내보냅니다. 이 함수를 사용하기 전에 COMILX_SDIO_SetDioUsage()함수를 사용하여 CH0 ~ CH7 을 디지털 출력 채널로 설정하여야 합니다.

매개 변수

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

▶ ***nModuleNo*** : 설정하고자 하는 COMI-STM4A 디지털 입출력 터미널 모듈 번호를 지정합니다. 모듈 번호는 0 ~ 15 까지 지정할 수 있으며 해당 COMI-STM4A 터미널 모듈에서 점퍼로 설정되는 모듈번호와 일치하여야 합니다.

▶ ***bValue*** : CH0 ~ CH7 의 디지털 출력 상태를 지정합니다. 이 값은 8 비트 값으로써 각 비트의 상태가 각채널의 ON/OFF 상태를 의미합니다.

Return

함수 수행의 성공 여부

Value	Meaning
0	실패
1	성공

참고

특정 채널의 상태만 변경하고자 하는 경우 COMILX_SDIO_ReadLowByte() 함수를 통하여 CH0 ~ CH7 의 현재 출력 상태를 읽어들인 후 원하는 채널에 해당하는 비트만 변경하여 출력하면 됩니다. CH0 ~ CH7 의 채널 중에 특정 채널만 출력을 변경하는 함수를 다음과 같이 구성할 수 있습니다.

```
void PutDO_LowByte(HANDLE hDevice, int nModule, int nChannel, int
nState)
{
   unsigned char do_states;
   do_states = COMILX_SDIO_ReadLowByte(hDevice, nModule);
   if(nState)// 비트값을 1 로 설정하는 경우
       do_states |= (1<<nChannel);
   else // 비트값을 0 으로 설정하는 경우
       do_states &= ~(1<<nChannel);
   COMILX_SDIO_WriteLowByte(hDevice, nModule, do_states);
}
```

예 제

다음의 예제는 0 번 모듈의 디지털 입출력 모드를 DO_DI 로 설정하고, CH8~CH15 로부터 디지털 입력 상태를 읽어들여 각 채널의 상태를 화면에 표시하고 이를 다시 디지털 출력(CH0~CH7)으로 내보내는 예입니다.

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

#define MOD0  0

// GetBitFromByte() : 바이트 데이터의 특정 비트의 값을 반환하는 함수 //
int GetBitFromByte(unsigned char Byte, int nBitNo)
{
   return((Byte>>nBitNo) & 0x1);
}

// SetBitOfByte () : 바이트 데이터의 특정 비트의 값을 변경하는 함수 //
unsigned char SetBitOfByte(unsigned char Byte, int nBitNo, int
nState)
{
   if(nState)// 비트값을 1 로 설정하는 경우
       Byte |= (1<<nBitNo);
   else // 비트값을 0 으로 설정하는 경우
       Byte &= ~(1<<nBitNo);
   return (Byte);
}


void main()
{
   unsigned char di_byte, do_byte;
   int di_each[16];

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

```
    HANDLE hDevice = COMILX_LoadDevice(COMI_ST401, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    // RS-422 통신 초기화 //
    COMILX_SDIO_InitComm (hDevice);
    // 제어대상 터미널 모듈이 연결되어 있는지 체크 //
    if(!COMILX_SDIO_CheckModule(hDevice, MOD0))
        exit(-1); // 제어대상 모듈이 연겨되지 않았으므로 프로그램 종료 //

    COMILX_SDIO_SetDioUsage (hDevice, MOD0, DI_DO);

    while (!_kbhit()){
        di_byte = COMILX_SDIO_ReadHighByte(hDevice, MOD0);
        for(int i=0; i<8; i++){
            di_each[i] = GetBitFromByte(di_byte, i);
        }
        printf("D/I States(CH7~CH8) = %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]);
        // 다음의 구문은 읽어들인 CH0~CH7 의 D/I 상태를 CH8~CH15 를    //
        // 통하여 D/O 출력으로 내보내는 것이다. 사실 do_byte = di_byte//
        // 를 대입하면 되지만 디지털 출력 각 채널을 개별적으로 설정하는 것//
        // 을 예로 보이기 위해서 실제로는 불필요한 아래의 for 루프를     //
        // 사용한 것이다.                                              //
        do_byte = 0;
        for(int i=0; i<8; i++){
            do_byte = SetBitOfByte(do_byte, i, di_each[i]);
        }
        COMILX_SDIO_WriteLowByte(hDevice, MOD0, do_byte);
    }

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

▣ COMILX_SDIO_WriteHighByte

함수 원형 :

BOOL COMILX_SDIO_WriteHighByte (HANDLE hDevice, int nModuleNo, BYTE bValue)

함수 설명 :

이 함수는 지정한 COMI-STM4A 터미널 모듈의 CH8 ~ CH15 에 디지털 출력을 내보냅니다. 이 함수를 사용하기 전에 COMILX_SDIO_SetDioUsage()함수를 사용하여 CH8 ~ CH15 를 디지털 출력 채널로 설정하여야 합니다.

매개 변수

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

▶ ***nModuleNo*** : 설정하고자 하는 COMI-STM4A 디지털 입출력 터미널 모듈 번호를 지정합니다. 모듈 번호는 0 ~ 15 까지 지정할 수 있으며 해당 COMI-STM4A 터미널 모듈에서 점퍼로 설정되는 모듈번호와 일치하여야 합니다.

▶ ***bValue*** : CH8 ~ CH15 의 디지털 출력 상태를 지정합니다. 이 값은 8 비트 값으로써 각 비트의 상태가 각채널의 ON/OFF 상태를 의미합니다.

Return

함수 수행의 성공 여부

Value	Meaning
0	실패
1	성공

참고

특정 채널의 상태만 변경하고자 하는 경우 COMILX_SDIO_ReadHighByte() 함수를 통하여 CH8 ~ CH15 의 현재 출력 상태를 읽어들인 후 원하는 채널에 해당하는 비트만 변경하여 출력하면 됩니다. CH8 ~ CH15 의 채널 중에 특정 채널만 출력을 변경하는 함수를 다음과 같이 구성할 수 있습니다.

```
void PutDO_HighByte(HANDLE hDevice, int nModule, int nChannel,
int nState)
{
   unsigned char do_states;
   do_states = COMILX_SDIO_ReadHighByte(hDevice, nModule);
   if(nState)// 비트값을 1 로 설정하는 경우
       do_states |= (1<<(nChannel-8));
   else // 비트값을 0 으로 설정하는 경우
       do_states &= ~(1<<(nChannel-8));
   COMILX_SDIO_WriteHighByte(hDevice, nModule, do_states);
}
```

예 제

다음의 예제는 0 번 모듈의 디지털 입출력 모드를 DI_DO 로 설정하고, CH0~CH7 로부터 디지털 입력 상태를 읽어들여 각 채널의 상태를 화면에 표시하고 이를 다시 디지털 출력(CH8~CH15)으로 내보내는 예이다.

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

#define MOD0  0

// GetBitFromByte() : 바이트 데이터의 특정 비트의 값을 반환하는 함수 //
int GetBitFromByte(unsigned char Byte, int nBitNo)
{
   return((Byte>>nBitNo) & 0x1);
}

// SetBitOfByte () : 바이트 데이터의 특정 비트의 값을 변경하는 함수 //
unsigned char SetBitOfByte(unsigned char Byte, int nBitNo, int
nState)
{
   if(nState)// 비트값을 1 로 설정하는 경우
       Byte |= (1<<nBitNo);
   else // 비트값을 0 으로 설정하는 경우
       Byte &= ~(1<<nBitNo);
   return (Byte);
}


void main()
{
   unsigned char di_byte, do_byte;
   int di_each[16];

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

```
    HANDLE hDevice = COMILX_LoadDevice(COMI_ST401, 0);
    if(hDevice == INVALID_HANDLE_VALUE)
        exit(-1); // Load Device Failure

    // RS-422 통신 초기화 //
    COMILX_SDIO_InitComm (hDevice);
    // 제어대상 터미널 모듈이 연결되어 있는지 체크 //
    if(!COMILX_SDIO_CheckModule(hDevice, MOD0))
        exit(-1); // 제어대상 모듈이 연겨되지 않았으므로 프로그램 종료 //

    COMILX_SDIO_SetDioUsage (hDevice, MOD0, DI_DO);

    while (!_kbhit()){
        di_byte = COMILX_SDIO_ReadLowByte(hDevice, MOD0);
        for(int i=0; i<8; i++){
            di_each[i] = GetBitFromByte(di_byte, i);
        }
        printf("D/I States(CH0~CH7) = %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]);
        // 다음의 구문은 읽어들인 CH0~CH7 의 D/I 상태를 CH8~CH15 를    //
        // 통하여 D/O 출력으로 내보내는 것이다. 사실 do_byte = di_byte//
        // 를 대입하면 되지만 디지털 출력 각 채널을 개별적으로 설정하는 것//
        // 을 예로 보이기 위해서 실제로는 불필요한 아래의 for 루프를     //
        // 사용한 것이다.                                              //
        do_byte = 0;
        for(int i=0; i<8; i++){
            do_byte = SetBitOfByte(do_byte, i, di_each[i]);
        }
        COMILX_SDIO_WriteHighByte(hDevice, MOD0, do_byte);
    }

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

3.7 카운터

이 단원에서는 카운터와 관련된 함수를 소개합니다. 카운터는 펄스신호를 카운트하는데 사용되는 함수입니다. 카운터에 관련된 함수는 다음과 같습니다.

함수명	각 보드별 적용 여부								
	LX101	LX102	LX103	LX201	LX202	LX203	LX301	LX401	LX501
COMILX_ReadCounter32	V	V	V				V	V	
COMILX_ClearCounter32	V	V	V				V	V	

[표 3-8] Digital Input/Output 에 관련된 함수 리스트 및 각 보드별 지원 여부

▣ COMILX_ReadCounter32

함수 원형

ULONG COMILX_ReadCounter32 (HANDLE hDevice, int ch)

함수 설명

이 함수는 지정한 카운터 채널의 카운트 값을 읽어옵니다.

매개 변수

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

▶ *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.

Return 값

32 비트 카운트 값

▣ COMILX_ClearCounter32

함수 원형

BOOL COMILX_ClearCounter32 (HANDLE hDevice, int ch)

함수 설명

이 함수는 지정한 카운터 채널의 카운트 값을 0으로 리셋(reset)하여 줍니다.

매개 변수

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

▶ *ch* : Counter 채널번호. 채널 번호는 0부터 시작합니다.

3.8 모션 제어(Motion Control)

이 단원에서 설명하는 모션제어 기능은 ㈜커미조아에서 개발한 COMI-LX501 모션제어(Motion Control) 전용 보드에서 제공하는 기능입니다. 따라서 이 단원에서 소개되는 모든 함수는 COMI-LX501 보드에만 적용가능합니다.

COMI-LX501 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.

3.8.1 모션 초기화 및 환경설정 함수

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

함수 / 설명	페이지
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_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 Motion 이 완료될 때까지 내부적으로 루프(Loop)를 돌면서 함수에서 Return 되지 않습니다. Blocking 모드를 FALSE 로 하면 이러한 경우에도 키보드, 마우스 이벤트 등과 같은 윈도우 이벤트나 메시지를 처리할 수 있습니다.

매개 변수

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

▶ *bBlocking* : Blocking 모드를 결정합니다.

참고

다음과 같은 함수들은 Motion 이 완료될 때까지 함수에서 Return 되지 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

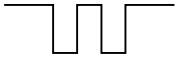
Command 펄스의 출력 모드를 설정합니다.

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

Return 값

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. 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 4가지로 설정할 수 있습니다.

Value	Meaning
0	1X A/B (1채배 엔코더 입력 모드)
1	2X A/B (2채배 엔코더 입력 모드)
2	4X A/B (4채배 엔코더 입력 모드)
3	CW/CCW (A펄스 - 카운트 증가, B펄스 - 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)
65,535	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 결정됩니다.

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

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

Return 값:

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. 이 것은 COMILX_MC_SetSpeed, COMILX_MC_SetAccel, COMILX_MC_SetScurve, COMILX_MC_SetSpeedMx, COMILX_MC_SetAccelMx, COMILX_MC_SetScurveMx 등의 함수에 영향을 미칩니다.

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 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 각각 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

Return 값

현재 설정된 단위 거리에 대한 펄스 수를 반환합니다.

▣ COMILX_MC_SetInOutRatio

함수 원형

double 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 펄스수를 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)

Return 값

현재 설정된 단위 거리에 대한 펄스 수를 반환합니다.

3.8.2 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.

함수 / 설명	페이지
void COMILX_MC_SetSpeedMode(HANDLE hDevice, int nChannel, int nModeIndex) Motion의 속도 모드를 설정합니다.	
void COMILX_MC_SetSpeed (HANDLE hDevice, int nChannel, double fIniSpeed, double fWorkSpeed) Motion의 속도를 설정합니다.	
void COMILX_MC_SetAccel(HANDLE hDevice, int nChannel, double fAccel, double fDecel) Motion의 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 Return합니다.	
void COMILX_MC_Move (HANDLE hDevice, int nChannel, double fDistance) 현재의 위치에서 지정한 거리만큼 이동을 수행합니다. 이 함수는 모션이 완료될때까지 Return되지 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

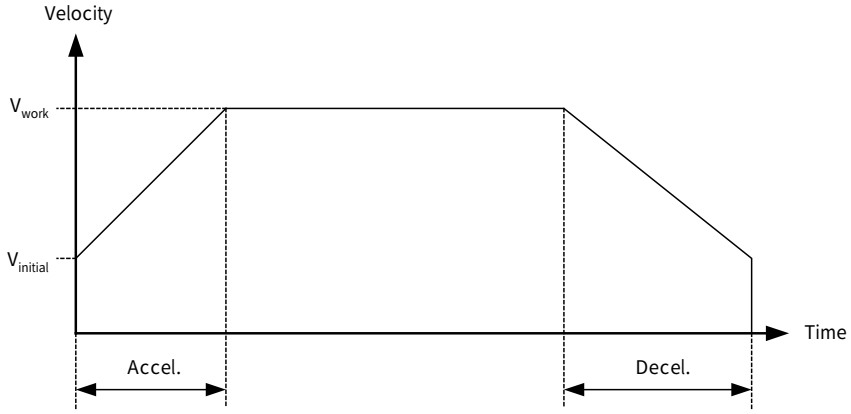
참 고

☐ 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 3-6]과 같이 Linear acceleration -> Working speed(constant) -> Linear deceleration 의 형태로 운용하는 모드입니다.



[

[그림 3-6] 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

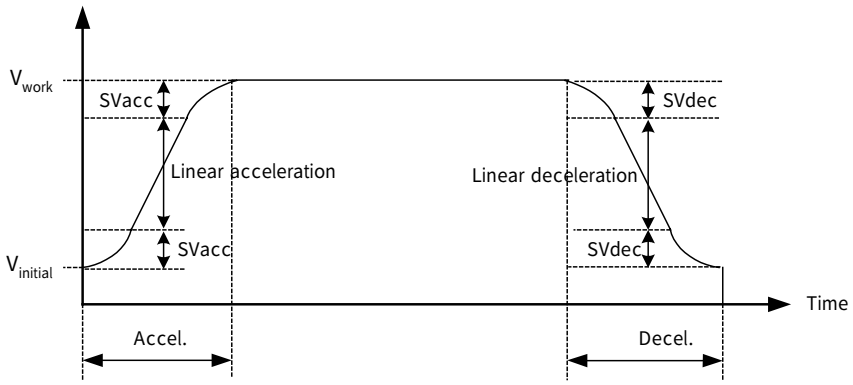
a : Acceleration setting value

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 3-7]과 같이 S-curve

section 과 Linear acceleration section 으로 구성됩니다.



[그림 3-7] 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

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 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);
   // Set S-Curve section range : 본 예제는 Linear section 이    //
   // 없는 완// 전한 S-curve 가/감속 모드가 되도록 SVacc, SVdec 값을 //
   // 모두 0 으로 설정함                                          //
   COMILX_MC_SetScurve(hDevice, X_AXIS, 0, 0);
   // 현재의 위치로부터 5000 만큼 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

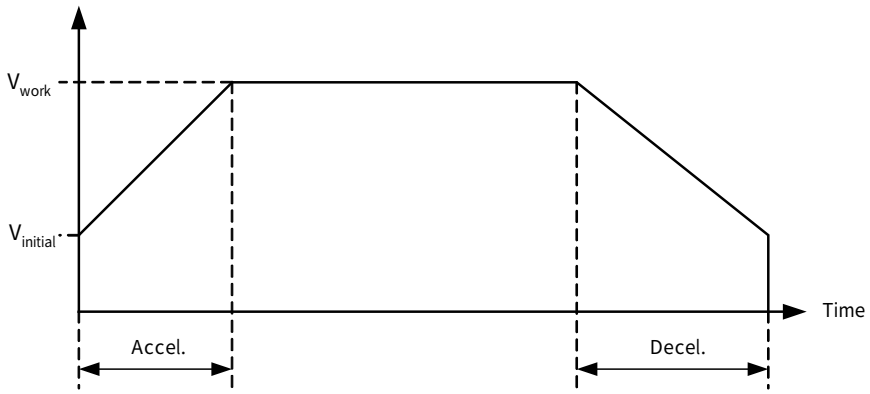
□ 속도의 단위는 COMILX_MC_SetUnitSpeed 함수에 의하여 결정되며 기본적으로는 Pulses/sec 입니다.

□ 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.

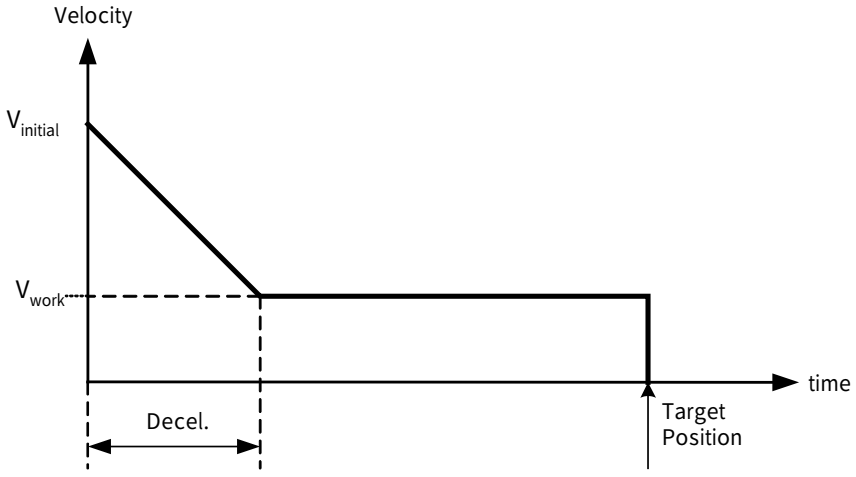
□ Trapezodial 또는 S-curve speed mode 에서 In-Position 명령을 수행할 때 작업속도(fWorkSpeed)가 초기속도(fIniSpeed)보다 크면 [그림 3-8]과 같이 초기속도⇒가

속⇒작업속도⇒감속⇒정지의 동작을 수행합니다.



[그림 3-8] 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 3-9]과 같이 초기속도로부터 출발하여 작업속도까지 감속 후에 작업속도를 유지하고 목표 위치까지 이동한 후에는 감속 없이 바로 정지하게 됩니다.



[그림 3-9] 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

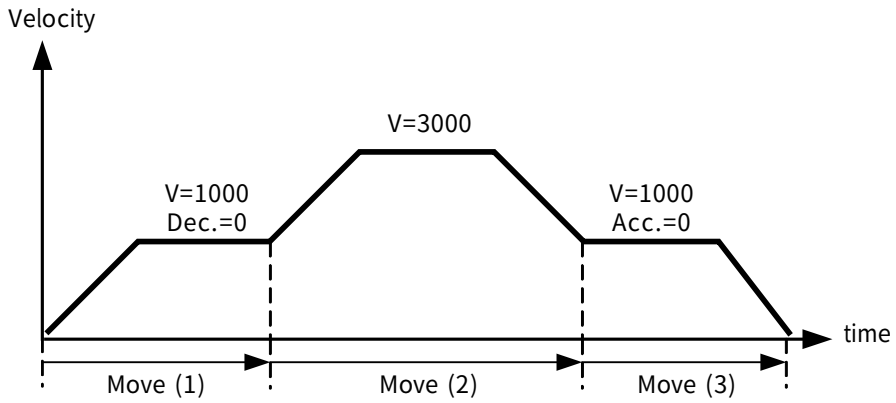
1 회전에 필요한 펄스수가 3600 펄스일 때 거리의 단위를 각도(1°)로, 속도의 단위를 rpm 으로 설정하는 예제입니다.

```
#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 3-10] 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.

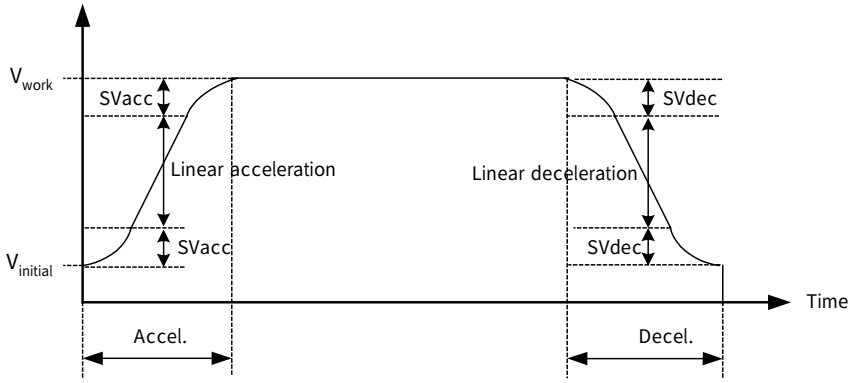
▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *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 3-11]과 같이 S-curve section 과 Linear acceleration section 으로 구성됩니다.



[그림 3-11] 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 각가 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 Position 의 현재 좌표를 0 으로 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 0 에서 30000 으로 이동 //
   COMILX_MC_StartMoveTo(hDevice, X_AXIS, 30000);
   while(!COMILX_MC_Done (hDevice, X_AXIS))
       ;
   // 좌표 30000 에서 70000 으로 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 Position 의 현재 좌표를 0 으로 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 0 에서 30000 으로 이동 //
   COMILX_MC_MoveTo(hDevice, X_AXIS, 30000);
   // 좌표 30000 에서 70000 으로 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

Return 값

지정한 축의 모션이 완료됐는지를 알려줍니다.

Value	Meaning
0	모션이 완료되지 않았음
1	모션이 완료됨

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();
}
```

3.8.3 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	페이지
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 Return하게 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 반

하여, 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.

Return 값

지정한 모든 축의 모션이 완료됐는지를 알려줍니다.

Value	Meaning
0	모션이 완료되지 않았음
1	모션이 완료됨

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();
}
```

3.8.4 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 같다.

함수 / 설명	페이지
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 Motion이 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.

▶ *nAixsList* : 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 1이어야 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.

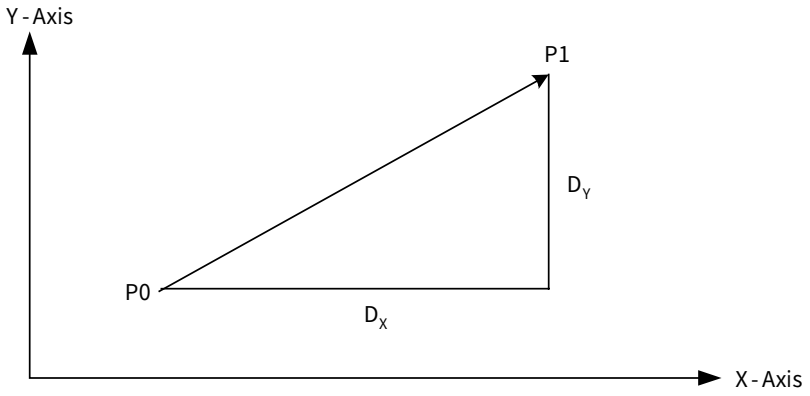
☐ Coordinated Motion 에서 초기속도는 자동으로 최저 속도로 설정됩니다.

☐ 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 3-12]은 2 축(편의상 X, Y 축으로 가정) 직선 보간 이동을 그래프로 나타낸 것입니다.



[그림 3-12] 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 1은 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 1은 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 Blocking이 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 1 은 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

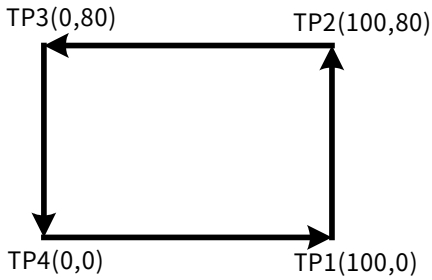
본 예제는 X 축과 Y 축을 그룹화하여 (10000, 20000)의 상대좌표로의 직선보간 이동을 수행하는 예제입니다.

```
#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 1 은 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 Blocking 이 일어나지 않도록 설정하여야 합니다.

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

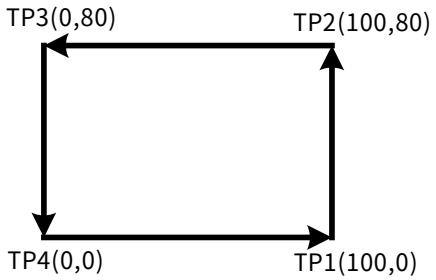
예 제

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

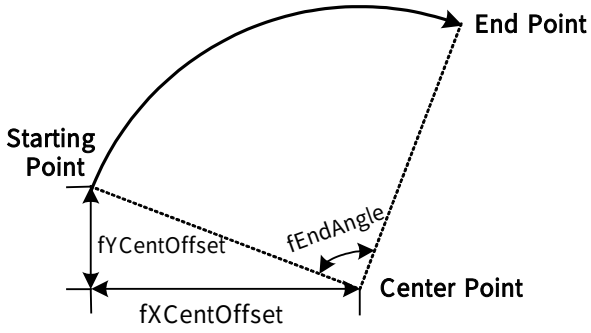
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 이 함수는 원호 보간 이동을 시작시킨후 바로 Return 합니다.

□ COMILX_MC_StartArc_p 함수가 원호 보간 이동을 완료할 목표지점의 좌표값을 파

라미터로 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 3-13]과 같습니다.



[그림 3-13] 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

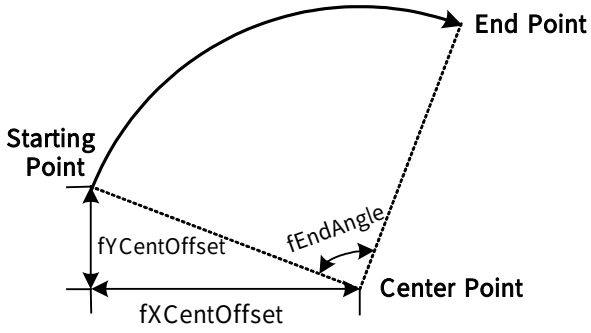
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 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 일

어나지 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.

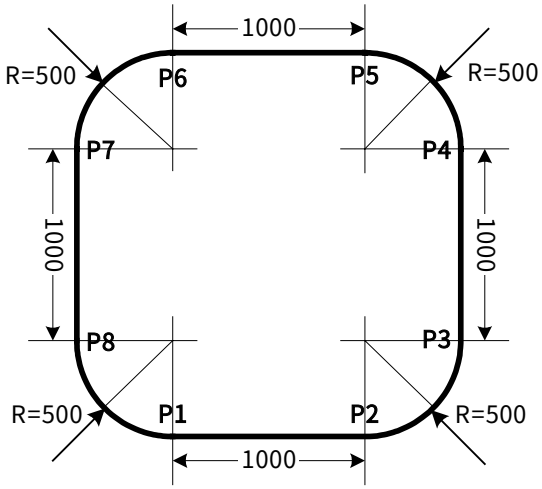
□ COMILX_MC_Arc_a 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 3-14]과 같습니다.



[그림 3-14] 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

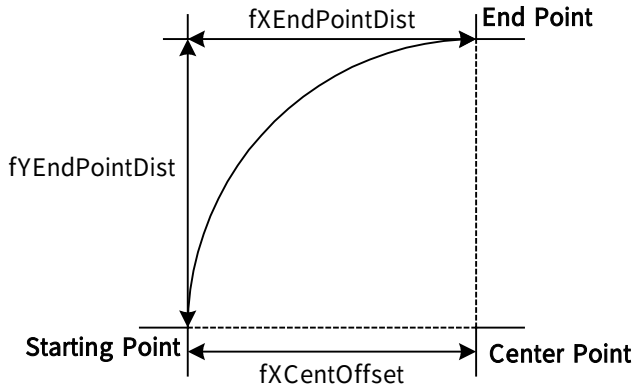
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 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 3-15]과 같습니다.



[그림 3-15] 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 맵핑

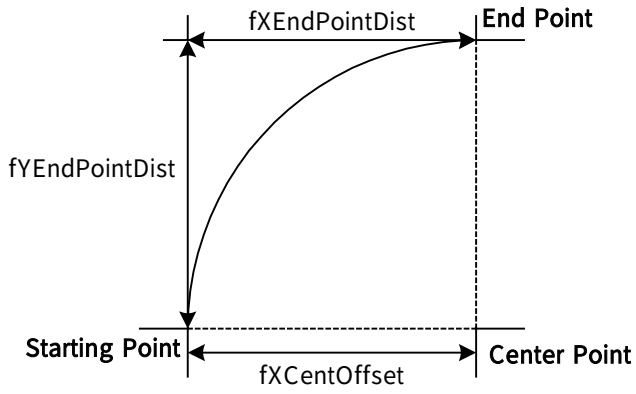
된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 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.

□ 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.

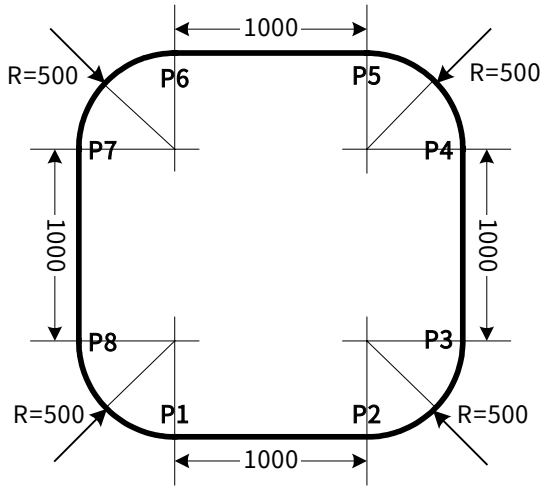
□ COMILX_MC_StartArc_p 함수를 사용하여 원호보간 이동을 수행할 때 각 파라미터의 의미는 [그림 3-16]과 같습니다.



[그림 3-16] 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

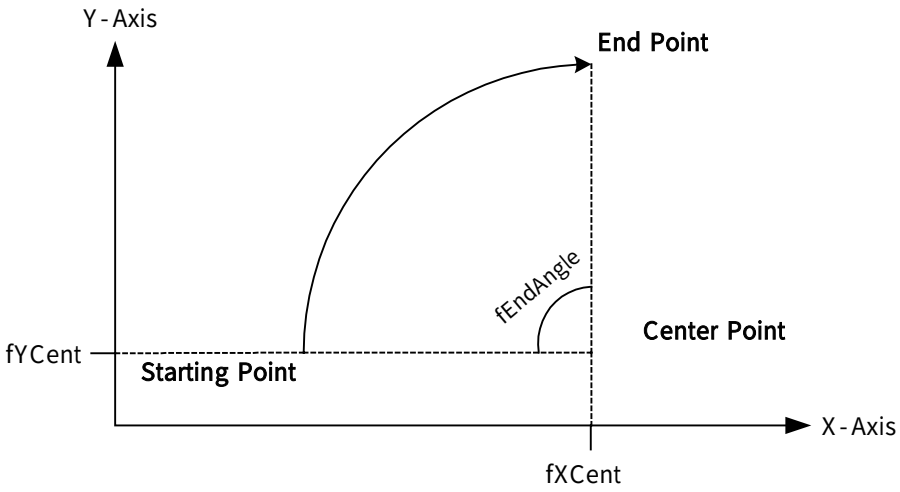
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 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 3-17]과 같습니다.



[그림 3-17] 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

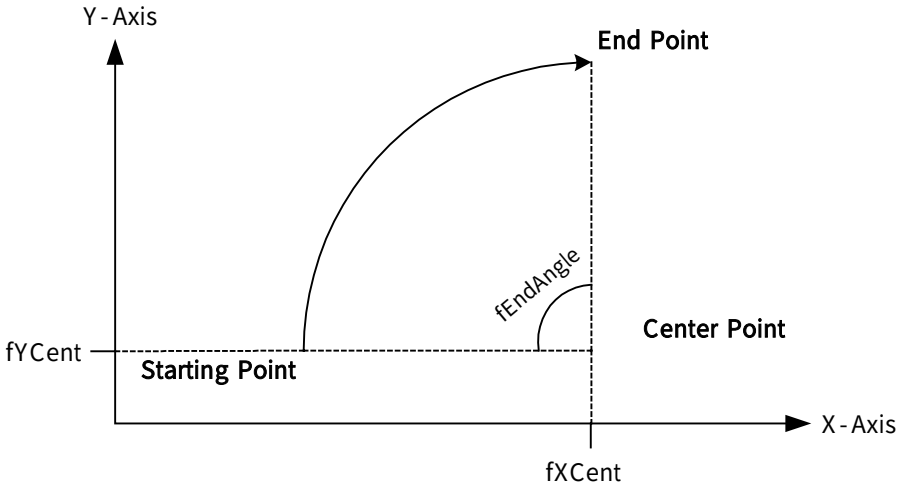
□ 원호보간 이동은 두 축에 대해서만 적용가능합니다. 따라서 본 단락에서는 맵핑된 두 축을 X, Y 축으로 간주하여 설명합니다. 여기서 X 축이라 함은 맵핑된 두 축 중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 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 일

어나지 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 파라미터의 의미는 [그림 3-18]과 같습니다.

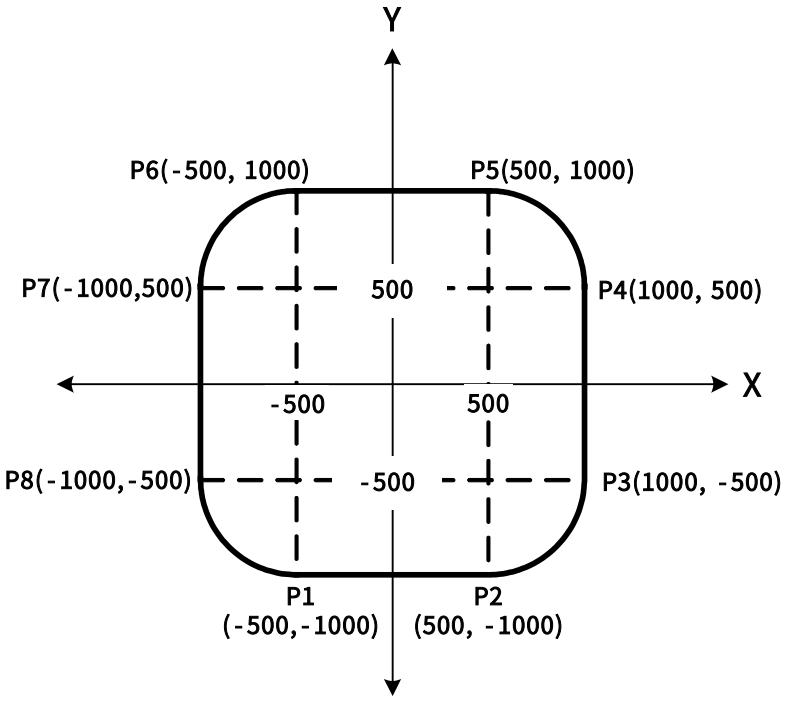


[그림 3-18] COMILX_MC_ArcTo_a 함수를 사용한 원호 보간 이동

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.

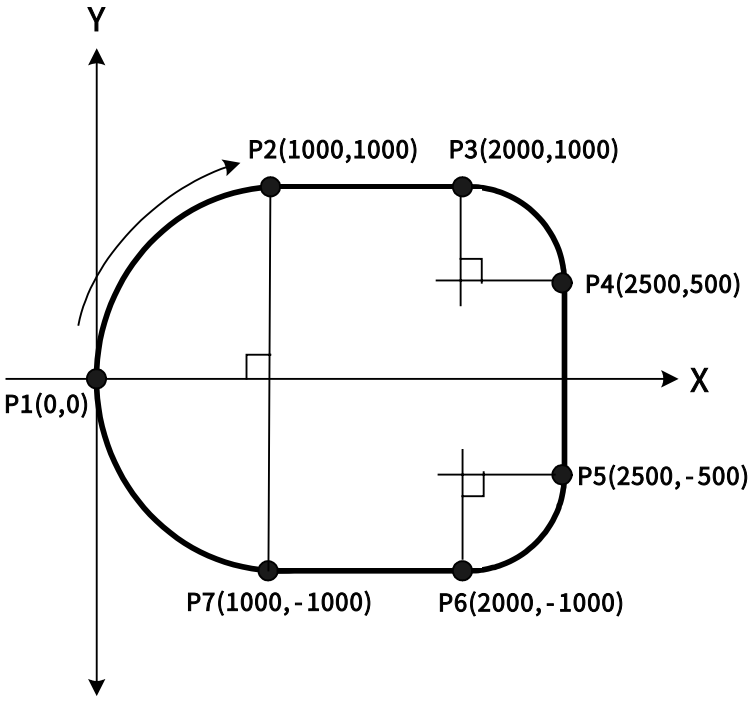



```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#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

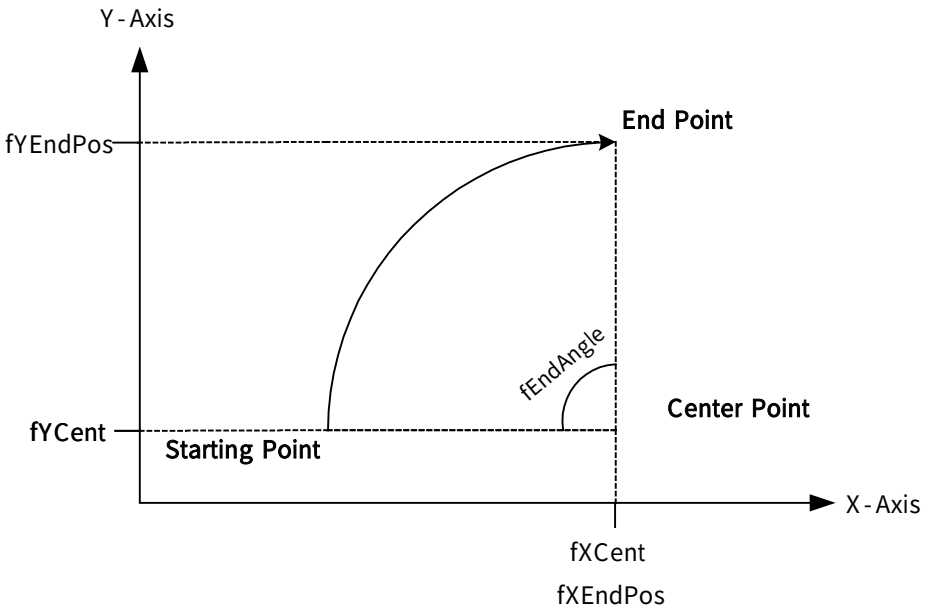
중에서 채널번호(X,Y,Z,U 순)가 낮은 축을 의미하며 Y 축은 채널번호가 높은 축을 의미합니다. 예를 들어 Z 축과 U 축이 맵핑된 두 축이라면 Z 축이 X 축에 해당하며 U 축이 Y 축에 해당합니다.

□ 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 3-19]과 같습니다.



[그림 3-19] 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.

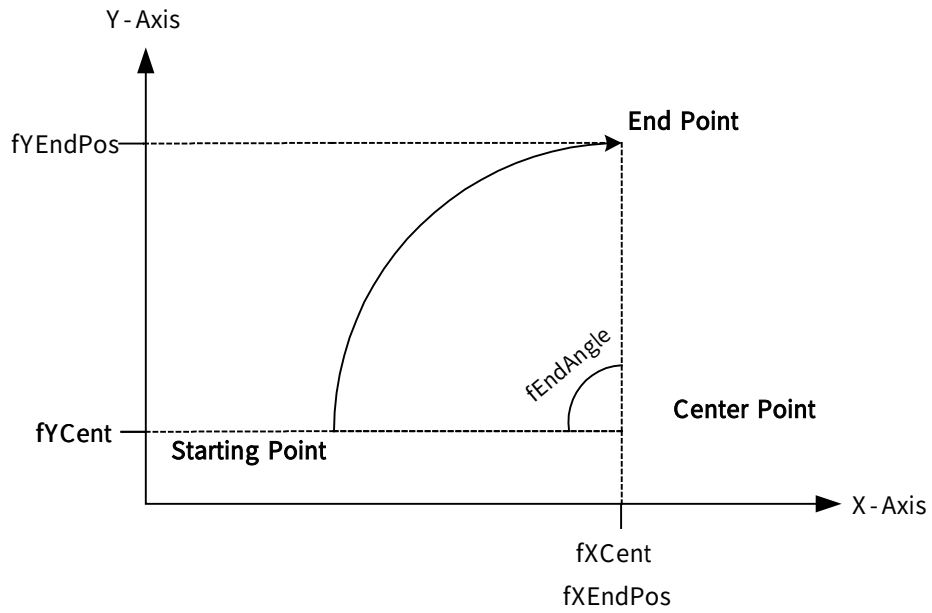
□ 이 함수는 원호 보간 이동이 완료되기 전까지 Return 되지 않고 루프를 돌게 됩니다. 루프를 도는 동안 윈도우 이벤트나 메시지가 처리될 수 있도록 하려면 이 함

수를 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 파라미터로 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 3-20]과 같습니다.



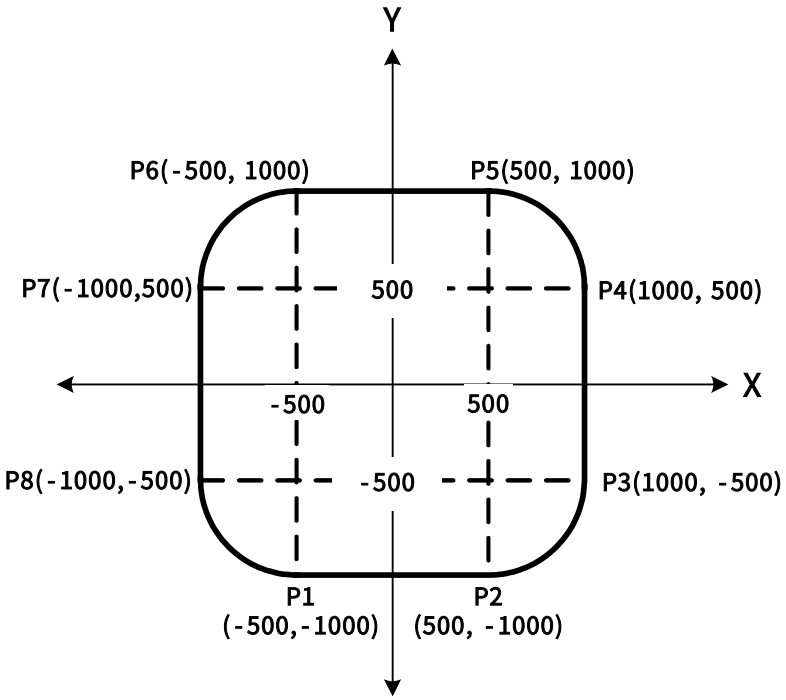
[그림 3-20] COMILX_MC_ArcTo_p 함수를 사용한 원호 보간 이동

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.

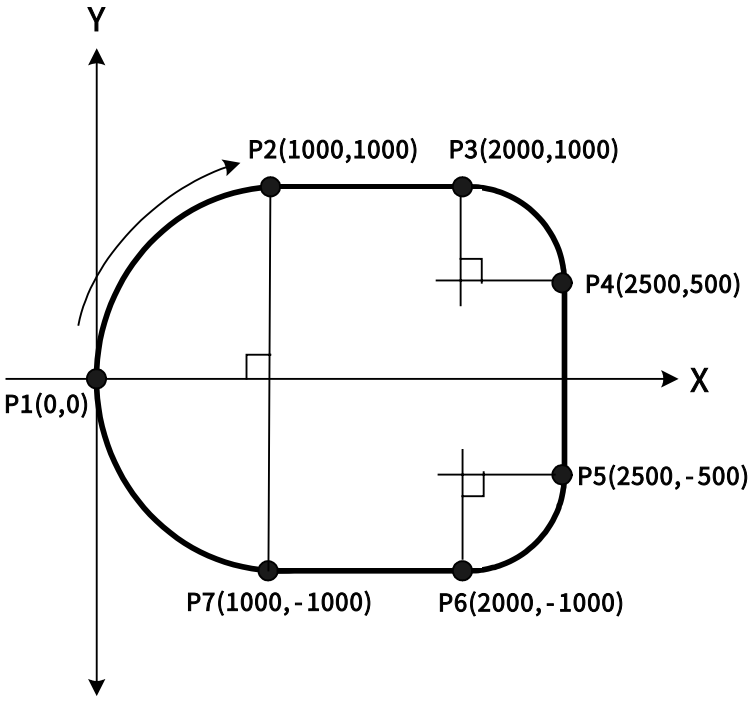



```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"

#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.

Return 값

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

3.8.5 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.

함수 / 설명	페이지
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.

3.8.6 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 Counter 는 자동으로 0 으로 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-LX501 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-LX501 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-LX501 보드에 입력되어야 합니다.

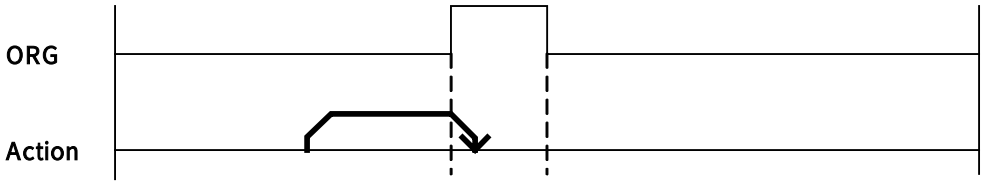
원점복귀모드

COMI-LX501 모션제어보드는 다음과 같이 13 가지의 다양한 원점 복귀 모드를 제공합

니다. 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.

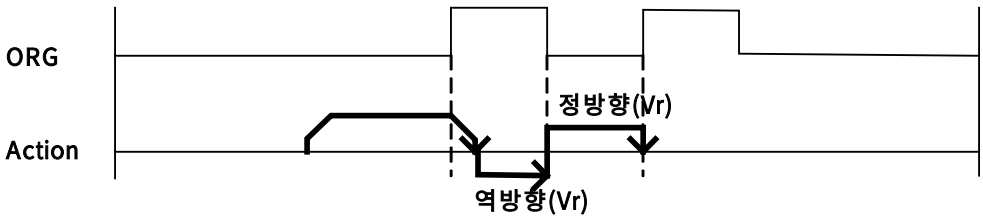
☐ MODE 0 : ORG -> Slow down -> Stop

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.

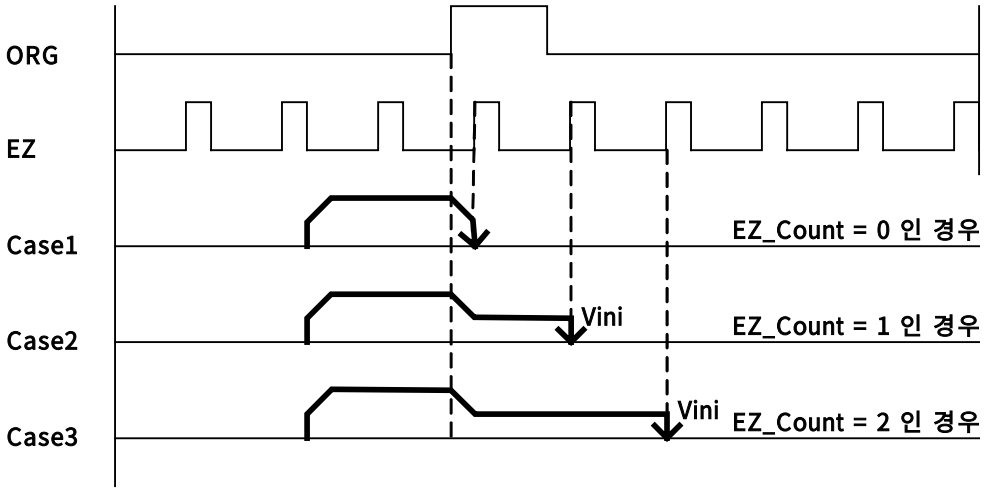


Vr : Reverse Speed를 의미하며 COMILX_MC_HomeMove 함수 참조

☐ MODE 2 : ORG -> Slow down -> Stop on EZ signal

MODE 2 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간에 모션을 감속한 후 초기속

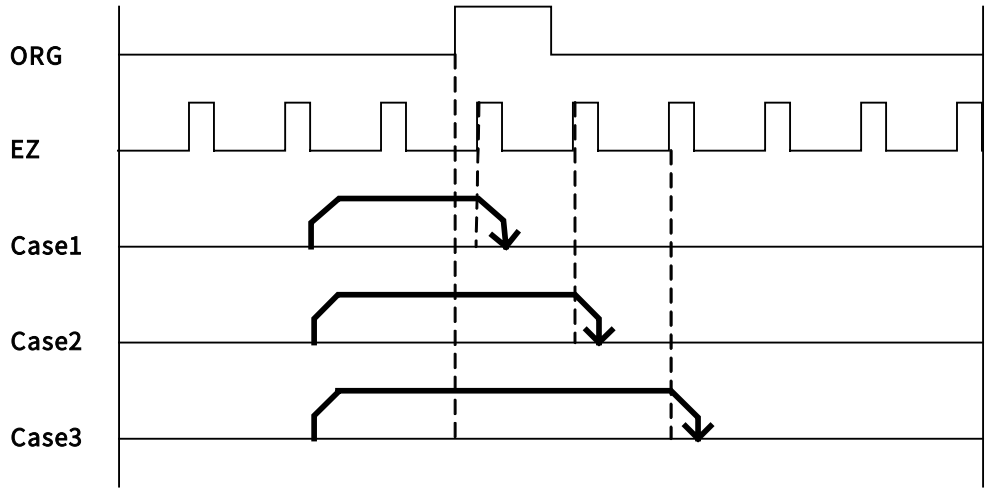
도값으로 모션을 진행하다가 EZ 신호에 따라 복귀작업을 종료합니다. COMILX_MC_SetHomeConfgi 함수를 통하여 미리 설정된 EzCount 값에 따라 종료하는 시점은 아래와같이 달라집니다.



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

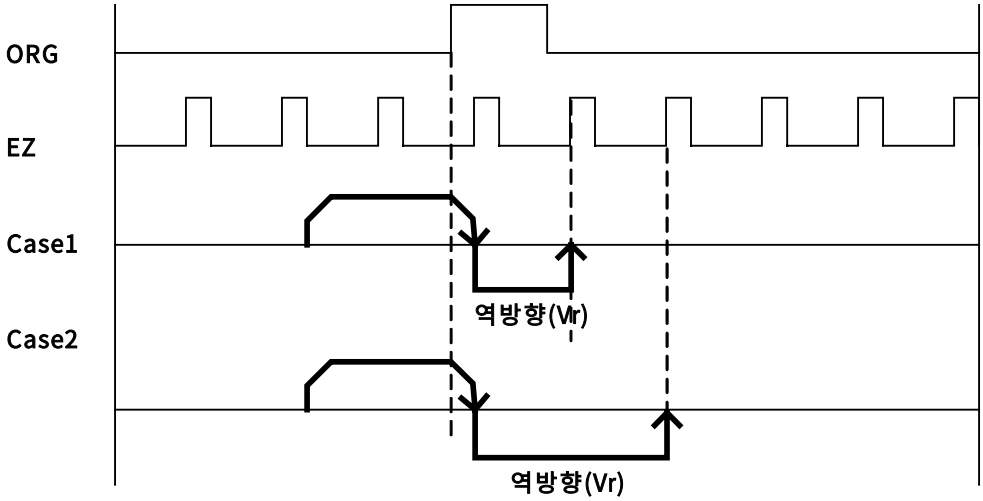
MODE 3 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌고 난 후 발생하는 EZ 신호에 따라 감속 후 정지합니다. COMILX_MC_SetHomeConfgi 함수를 통하여 미리 설정된 EzCount 값에 따라 감속을 시작하는 시점은 아래와같이 달라집니다.



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

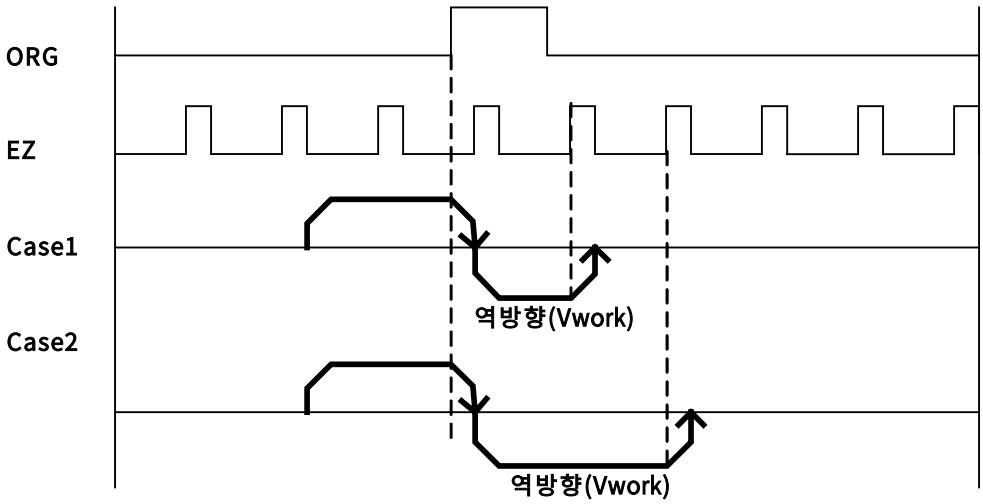
MODE 4 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간 감속 후 정지합니다. 그리고 Vr 의 속도로 역방향 회전한 후 EZ 신호에 따라 복귀 작업을 종료합니다.



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

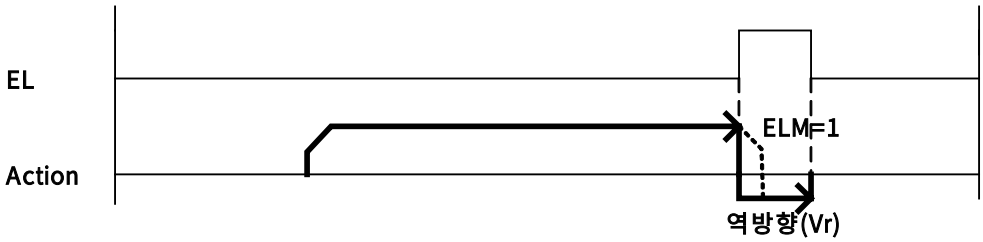
MODE 5 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간 감속 후 정지합니다. 그리고 작업속도까지 가속하여 역방향 회전한 후 EZ 신호에 따라 감속 후 복귀 작업을 종료합니다.



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

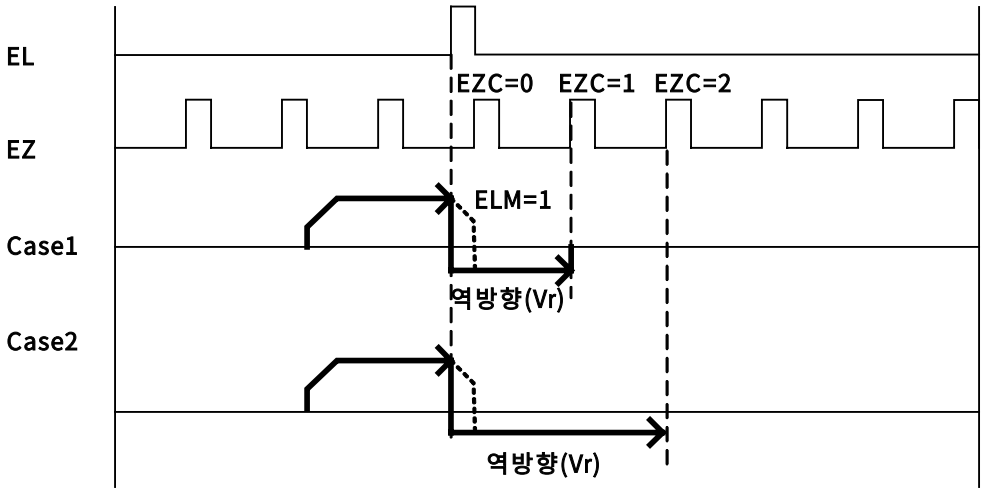
MODE 6 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속 후 정지)합니다. 그리고 반대 방향으로 Vr 속도로 회전하다가 EL 신호가 OFF 되는 순간에 복귀작업을 종료합니다. 여기서 ELM=1 은 COMILX_MC_SetMioCfgEL()함수에서 nElMode 를 1 로 설정했음을 의미합니다.



Vr : Reverse Speed를 의미하며 COMILX_MC_HomeMove 함수 참조

☐ MODE 7 : EL ON -> Go back at Vr -> Stop on EZ signal

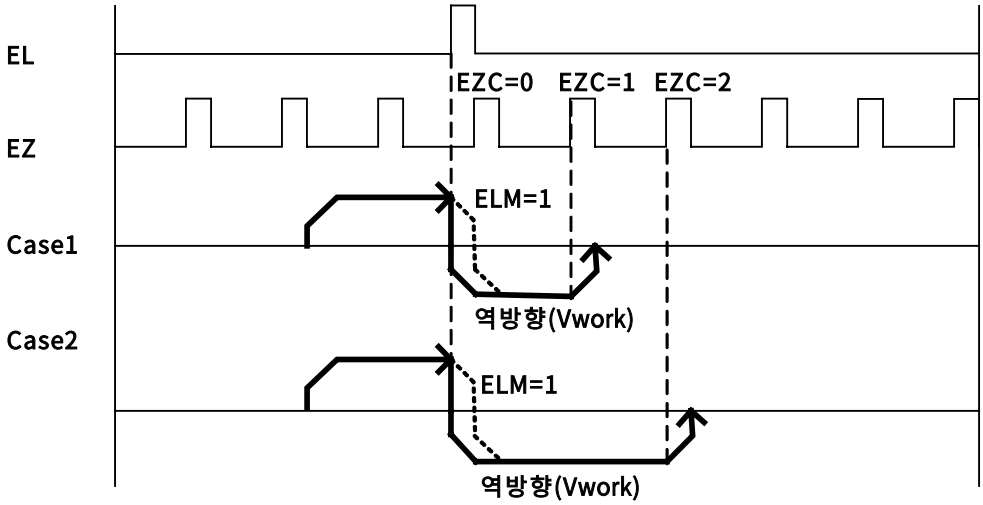
MODE 7 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속 후 정지)합니다. 그리고 반대 방향으로 Vr 속도로 회전하다가 EZ_Count 에 따라 복귀작업을 종료합니다. 여기서 ELM=1 은 COMILX_MC_SetMioCfgEL()함수에서 nElMode 를 1 로 설정했음을 의미합니다.



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 1 로 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 에서와 똑같은 복귀 작업을 수행한다. 그리고 난후 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 에서와 똑같은 복귀 작업을 수행한다. 그리고 난후 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 에서와 똑같은 복귀 작업을 수행한다. 그리고 난후 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 에서와 똑같은 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-LX501 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 제공합니다.

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();
}
```

3.8.7 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-LX501 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.

함수 / 설명	페이지
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_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);
    // 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);
   // 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_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_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

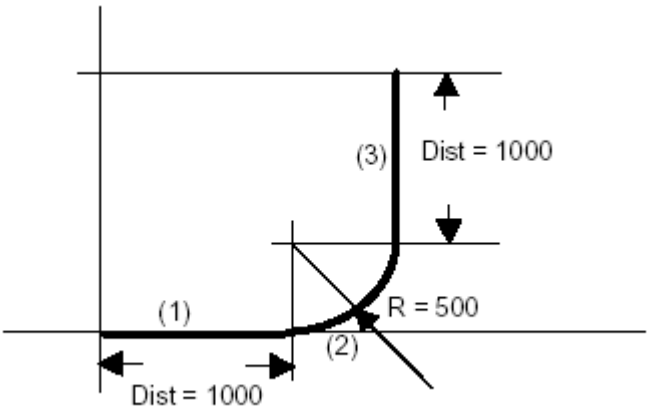
```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidaslx.h"
```

```
#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_SetPulserInputMode   (hDevice,   X_AXIS,   PHASE_1X,
FALSE);
   COMILX_MC_PulserMoveTo(hDevice, X_AXIS, 1000);

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

3.8.9 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



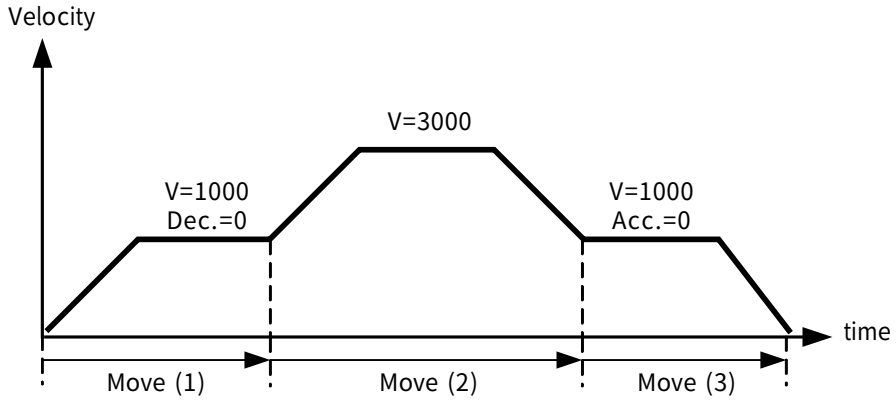
[그림 3-21] 3 회의 Coordinated Motion 으로 이루어지는 작업의 예

[그림 3-21]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 3-22] 속도의 연속성을 가지는 연속적인 In-Position 모션

리스트 모션을 이용하면 [그림 3-22]과 같이 In-Position 의 경우에도 작업 속도의

연속성을 확보할 수 있습니다. 다음의 코드는 [그림 3-22]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.

함수 / 설명	페이지
void COMILX_MC_BeginList (HANDLE hDevice) List Motion 에서 수행할 작업들의 등록을 시작합니다.	
void COMILX_MC_EndList (HANDLE hDevice) List Motion 에서 수행할 작업들의 등록을 종료합니다.	
void COMILX_MC_StartListMotion (HANDLE hDevice) List Motion 으로 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 Motion 으로 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.

Return 값

Value	Meaning
0	리스트 모션이 완료되지 않음
1	등록된 모든 리스트 모션이 완료됨

3.8.10 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.

함수 / 설명	페이지
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

Return 값

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

Return 값

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

Return 값

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

Return 값

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

Return 값

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

Return 값

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

Return 값

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

Return 값

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

Return 값

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

Return 값

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

Return 값

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	RDY pin input status
BIT1	ALM	Alarm signal status
BIT2	+EL	Positive limit switch status
BIT3	-EL	Negative limit switch status
BIT4	ORG	Orgin switch status
BIT5	DIR	DIR output status
BIT6	Reserved	
BIT7	PCS	PCS signal input status
BIT8	ERC	ERC pin output status
BIT9	EZ	Index signal status
BIT10	Reserved	
BIT11	Latch	Latch signal input status
BIT12	SD	Slow Down signal input status

BIT13	INP	In-Position signal input status
BIT14 ~BIT31	Reserved	

3.8.11 I/O(입출력) 환경설정 함수

이 단원에서는 I/O(입출력) 환경설정 함수에 관하여 설명합니다. COMI-LX501 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.

함수 / 설명	페이지
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_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_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_SetMioCfgERC (HANDLE hDevice, int nChannel, int nErcLogic, int nErcOnTime) ERC 신호의 입력 로직(Logic) 및 ON time을 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_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_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_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_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.

▶ *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.

▶ *pInpLogic* : INP 입력 신호의 로직(Logic)을 설정합니다.

Value	Meaning
0	Active low
1	Active high

▣ 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_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	Decleration 만 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.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▶ *nChannel* : 채널(축) 번호, 0 ~ 3

▶ *fLimitP* : (+)방향 Limit 값을 설정합니다.

▶ *fLimitN* : (-)방향 Limit 값을 설정합니다.

▣ 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.

3.8.12 인터럽트 관련 함수

이 단원에서는 인터럽트에 관련된 함수들을 소개합니다. 인터럽트는 특정 상황이 발생되었을 때 사용자(또는 프로그래머)에게 이 것을 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.

함수 / 설명	페이지
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

▣ 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.

▣ 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 Disable된 상태입니다.

매개 변수

▶ *hDevice* : 디바이스 핸들.

▣ 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.

Return 값

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

▣ 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

Appendix A

라이브러리 함수 리스트

본 부록에서는 COMI-LX 시리즈 라이브러리를 사용하는데 있어서 각 함수의 이해 및 검색에 도움을 주고자 라이브러리 함수에 대한 다양한 리스트를 제공합니다. 본 부록에서는 각 기능별로 함수를 구분하여 수록한 리스트와 각 함수별로 지원 가능한 디바이스에 관한 리스트를 수록하였습니다.

Appendix A 라이브러리 함수 리스트

A.1 기능별 함수 색인

A.2 함수별 지원 가능 디바이스 리스트

메쏘드명	각 보드별 지원 여부									
	LX101	LX102	LX103	LX201	LX202	LX203	LX301	LX401	LX402	LX501
COMILX_LoadDll	V	V	V	V	V	V	V	V	V	V
COMILX_UnloadDll	V	V	V	V	V	V	V	V	V	V
COMILX_LoadDevice	V	V	V							
COMILX_UnloadDevice	V	V	V	V	V	V				
COMILX_AD_SetInputType	V	V	V							
COMILX_AD_SetRange	V	V	V							
COMILX_AD_GetDigit	V	V	V							
COMILX_AD_GetVolt	V	V	V							
COMILX_US1_Start	V	V	V							
COMILX_US1_Stop	V	V	V							
COMILX_US1_CurCount	V	V	V							
COMILX_US1_GetBuffer	V	V	V							
COMILX_US1_SBPos				V	V	V				
COMILX_US1_RetrvOne				V	V	V				
COMILX_US1_RetrvChannel				V	V	V				
COMILX_US1_RetrvBlock				V	V	V				
COMILX_US1_ReleaseBuf				V	V	V				
COMILX_US2_SetTriggerEvent				V	V	V				
COMILX_US2_Start				V	V	V				
COMILX_US2_Resume				V	V	V				
COMILX_US2_IsBufFull				V	V	V				
COMILX_US2_ChangeScanFreq	V	V	V				V			
COMILX_US2_DmaCount	V	V	V				V			
COMILX_US2_GetBuffer	V	V	V				V			

메쏘드명	각 보드별 지원 여부									
	LX101	LX102	LX103	LX201	LX202	LX203	LX301	LX401	LX402	LX501
COMILX_US2_RetrvChannel	V	V	V				V			
COMILX_US2_Stop	V	V	V				V			
COMILX_US2_ReleaseBuf	V	V	V				V			
COMILX_DA_Out	V	V	V	V	V	V	V	V		V
COMILX_WFM_Start	V	V	V	V	V	V	V	V		V
COMILX_WFM_Reload	V	V	V	V	V	V	V	V		V
COMILX_WFM_RateChange	V	V	V	V	V	V	V	V		V
COMILX_WFM_GetCurPos	V	V	V	V	V	V	V	V		V
COMILX_WFM_GetCurLoops									V	
COMILX_WFM_Stop									V	
COMILX_DIO_SetUsage									V	
COMILX_DI_GetOne									V	
COMILX_DI_GetAll									V	
COMILX_DO_PutOne									V	
COMILX_DO_PutAll									V	
COMILX_ReadCounter32	V	V	V				V	V		
COMILX_ClearCounter32	V	V	V				V	V		

※ 모션에 관련된 함수는 COMI-LX501 에만 적용 가능하며 본 리스트에는 생략되었습니다.