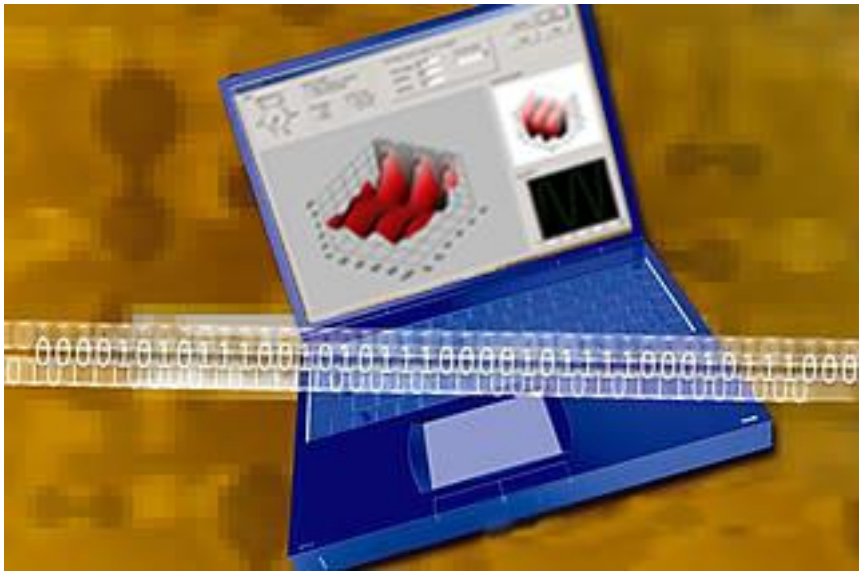
COMIZOA

SOFTWARE MANUAL



COMIZOA

COMIZOA

COMputer Innovation
is Zoomed by Our Affection!

2001 년 11 월 20 일 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

책의 구성

본 매뉴얼은 COMIDAS 디바이스를 제어하기 위해 필요한 드라이버, 테스트 프로그램 그리고 라이브러리의 설치 및 사용 방법에 대하여 설명하고 있습니다.

본 매뉴얼의 구성은 다음과 같습니다.

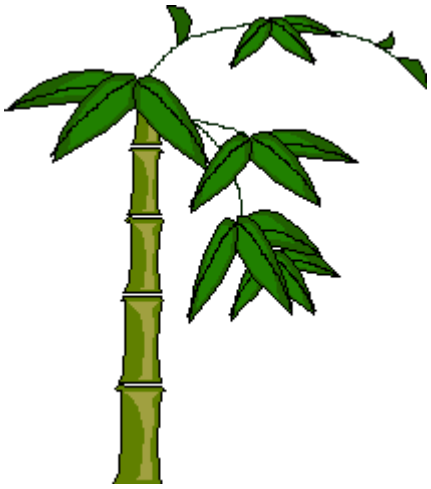
■ CHAPTER 1. 장치 제어기 및 소프트웨어 설치
이 장에서는 일반적으로 드라이버라 불리우는 장치 제어기를 설치하는 방법과 기타 소프트웨어를 설치하는 방법을 설명합니다.

■ CHAPTER 2. COMI-Master 유틸리티 소프트웨어
이 장에서는 COMIDAS 의 각종 기능을 테스트하거나 COMIDAS 를 이용하여 여러 가지 작업을 수행할 수 있도록 도와주는 유틸리티 소프트웨어의 사용방법에 대하여 설명합니다.

■ CHAPTER 3. Windows C/C++ 라이브러리
이 장에서는 COMIDAS 디바이스를 위한 Windows 용 C/C++ 라이브러리에 대하여 소개합니다. COMIDAS 라이브러리가 제공하는 모든 함수들에 대하여 자세한 설명을 수록하였으며, 이해를 돕기 위하여 각각의 함수들이 사용되는 예제를 제공하고 있습니다.

■ CHAPTER 4. DOS C/C++ 라이브러리
이 장에서는 COMIDAS 디바이스를 위한 DOS 용 C/C++ 라이브러리에 대하여 소개합니다. COMIDAS 라이브러리가 제공하는 모든 함수들에 대하여 자세한 설명을 수록하였으며, 이해를 돕기 위하여 각각의 함수들이 사용되는 예제를 제공하고 있습니다.

CHAPTER 1
장치 제어기 및 소프트웨어 설치

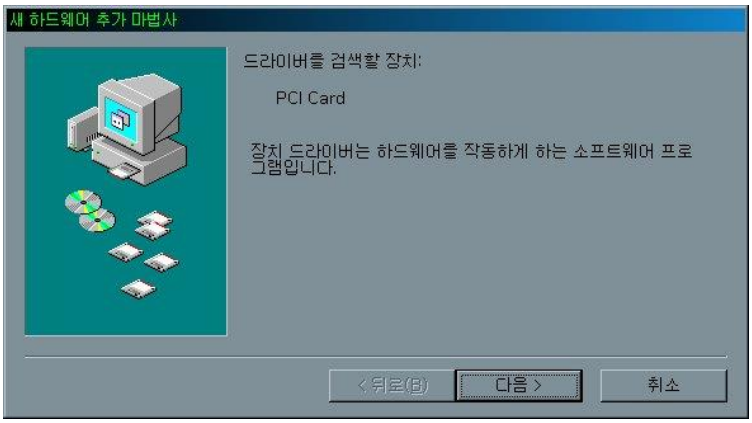


CHAPTER 1. 장치 제어기 및 소프트웨어 설치

본 장에서는 COMIDAS 장치 제어기(드라이버)를 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.

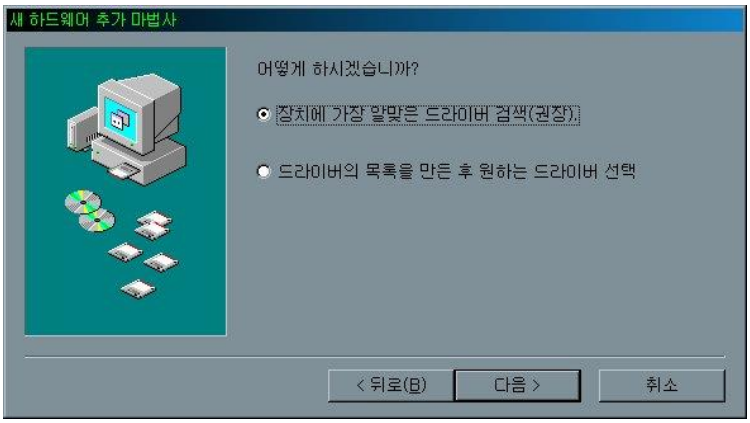
1-1 Windows98 및 WindowsME 드라이버 설치

먼저 컴퓨터 전원을 꺼주신 다음에 COMIDAS 디바이스를 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.



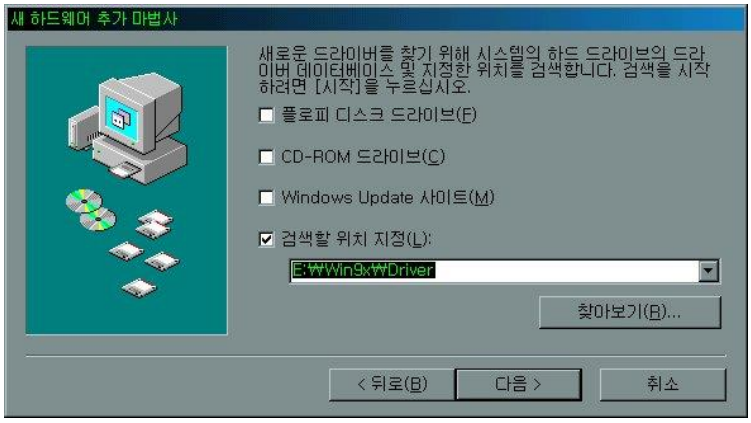
[그림 1-1] 디바이스 제어기 설치 화면 1

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



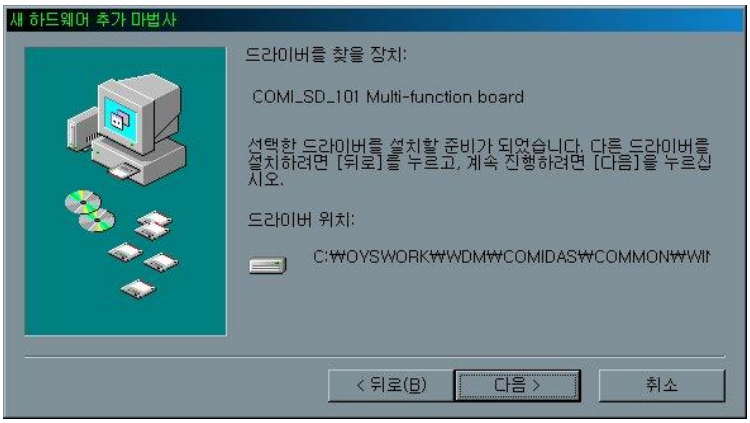
[그림 1-2] 디바이스 제어기 설치 화면 2

그림 1-3 과 같은 화면이 나타나면, '검색할 위치 지정(L):' 항목에 "E:₩Win9x₩Driver"를 입력한 후 '다음' 버튼을 클릭하십시오. 이때 "E:₩" 는 CDROM 드라이브를 의미합니다. 이미 소프트웨어 설치를 한 경우에는 설치된 디렉토리의 하위 디렉토리인 Window₩Driver 디렉토리를 지정하여도 됩니다.



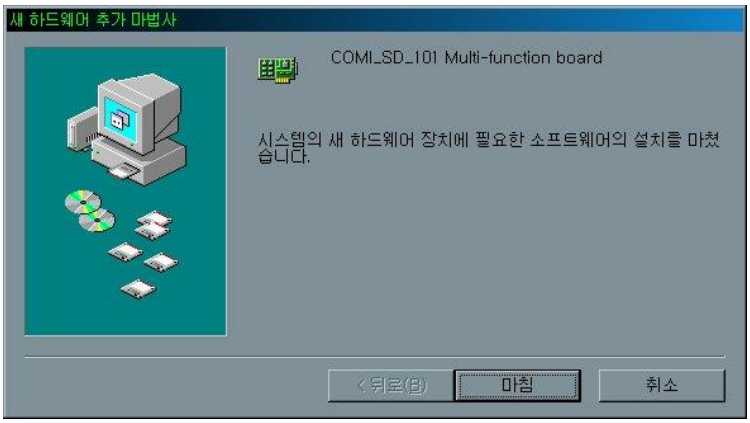
[그림 1-3] 디바이스 제어기 설치 화면 3

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

그림 1-5 와 같은 화면이 나타나면 '마침' 버튼을 클릭하여 제어기 설치를 마칩니다.

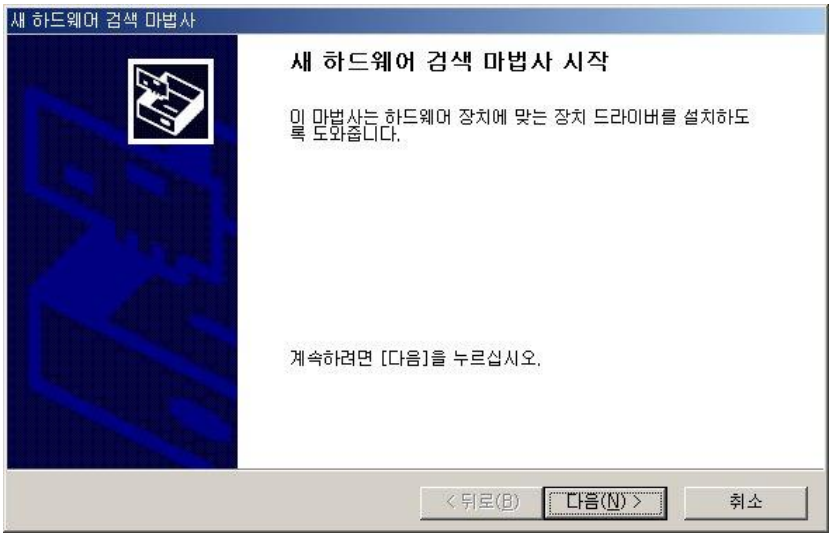


[그림 1-5] 디바이스 제어기 설치 화면 5

1-2 Windows2000 드라이버 설치

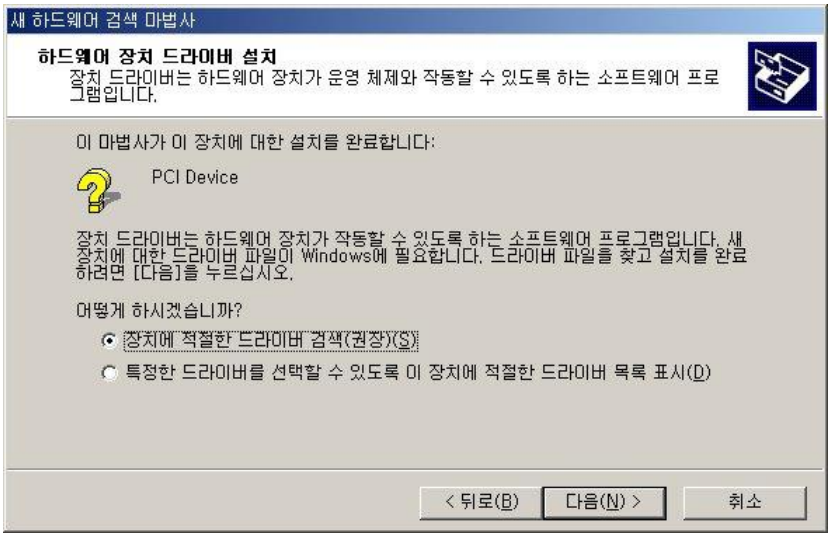
먼저 컴퓨터 전원을 꺼주신 다음에 COMIDAS 디바이스를 적당한 PCI 슬롯에 장착하고 시스템을 부팅시킵니다. 윈도우가 시작되면서 윈도우는 자동으로 COMIDAS 디바이스를 인식하여 그림 1-6 과 같이 PCI Card 를 찾았다는 화면을 나타낸 후 그림 1-7 과 같은 화면을 나타냅니다. 제공된 설치 CD 를 CDROM 드라이브에 삽입하신 후 '다음' 버튼을 클릭하십시오.

[그림 1-6] Win2000 디바이스 제어기 설치 화면 1



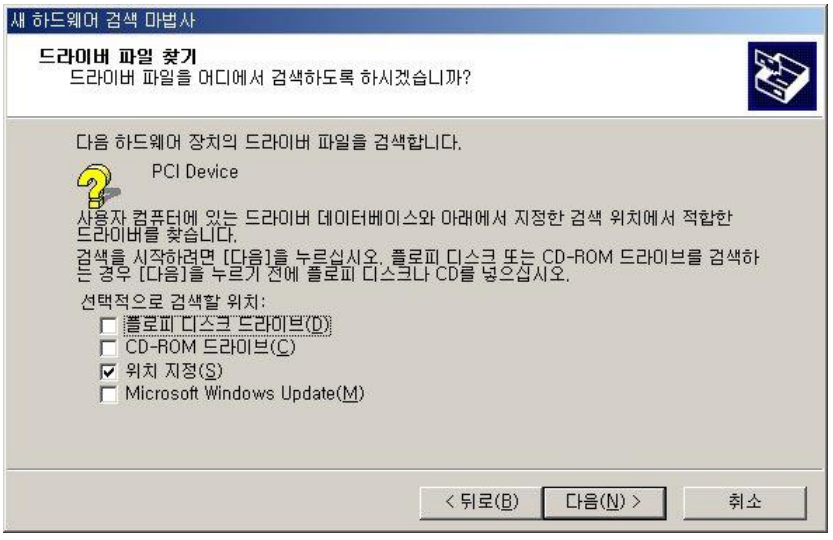
[그림 1-7] Win2000 디바이스 제어기 설치 화면 2

그림 1-8 과 같은 화면이 나타나면, '장치에 적절한 드라이버 검색(권장)'을 선택한 후 '다음' 버튼을 클릭하십시오.



[그림 1-8] Win2000 디바이스 제어기 설치 화면 3

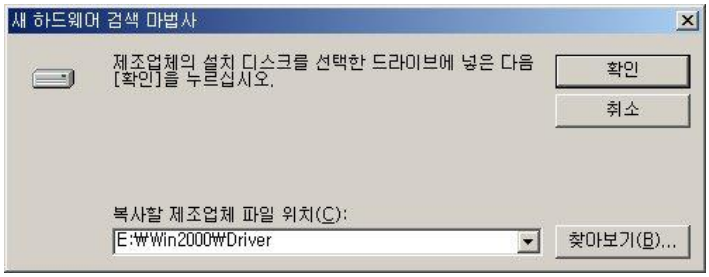
그림 1-9 와 같은 화면이 나타나면, '위치 지정'을 선택한 후 '다음' 버튼을 클릭하십시오.



[그림 1-9] Win2000 디바이스 제어기 설치 화면 4

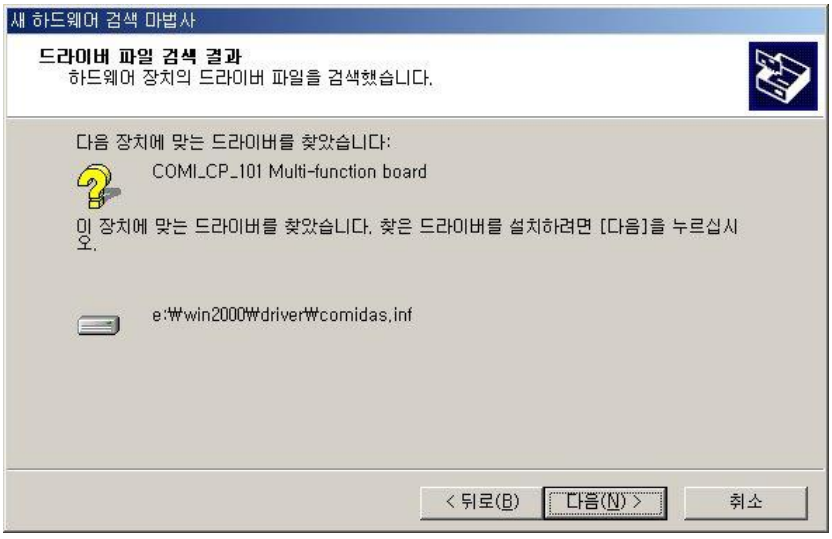
그림 1-10 과 같은 화면이 나타나면, '복사할 제조업체 파일 위치(C):' 항

목에 “E:₩Win2000₩Driver”를 입력한 후 ‘다음’ 버튼을 클릭하십시오. 이때 “E:₩”는 CDROM 드라이브를 의미합니다. 이미 소프트웨어 설치를 한 경우에는 설치된 디렉토리의 하위 디렉토리인 Window₩Driver 디렉토리를 지정하여도 됩니다.



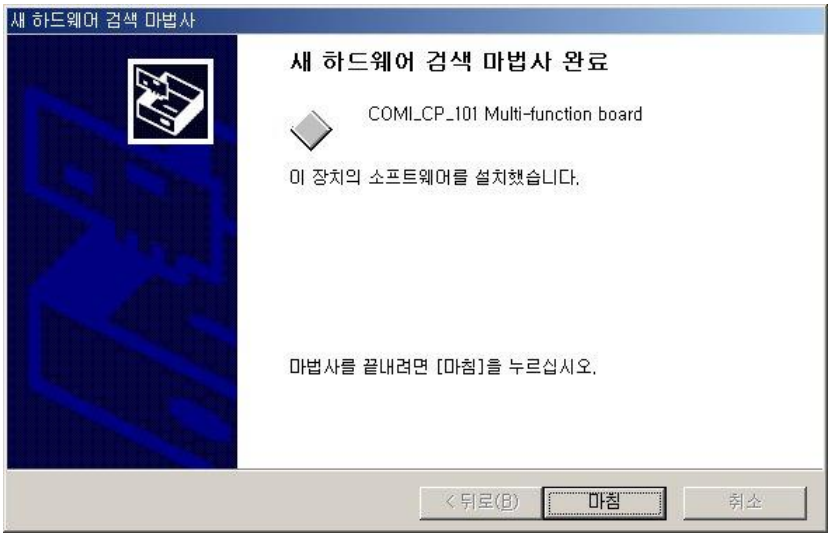
[그림 1-10] Win2000 디바이스 제어기 설치 화면 5

그림 1-11 과 같이 COMIDAS 디바이스 장치가 검색되었다는 화면이 나타나면, ‘다음’ 버튼을 클릭하십시오. 그러면 윈도우는 필요한 파일들을 설치합니다.



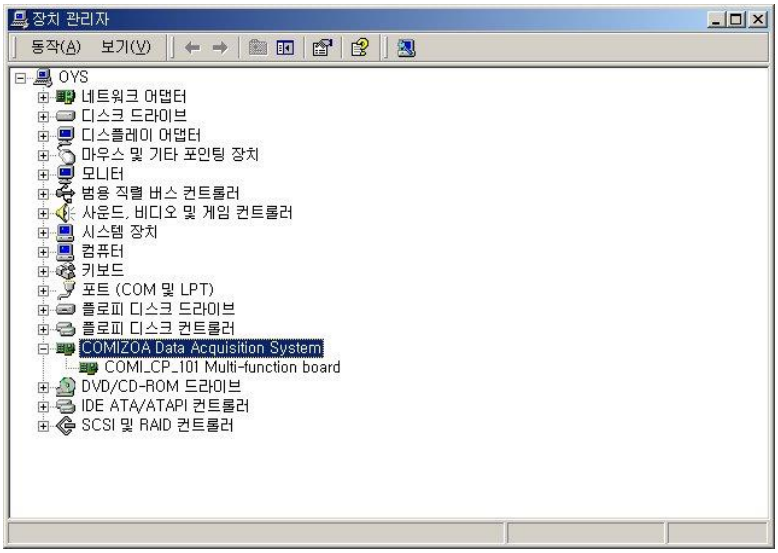
[그림 1-11] Win2000 디바이스 제어기 설치 화면 6

그림 1-12 와 같은 화면이 나타나면 ‘마침’ 버튼을 클릭하여 제어기 설치를 마칩니다.



[그림 1-12] Win2000 디바이스 제어기 설치 화면 7

제어기 설치를 마친 후 장치 관리자를 실행하면 그림 1-13 과 같이 COMIDAS 디바이스가 설치된 것을 확인할 수 있습니다.



[그림 1-13] Win2000 에서 COMIDAS 디바이스의 설치를 확인하는 화면

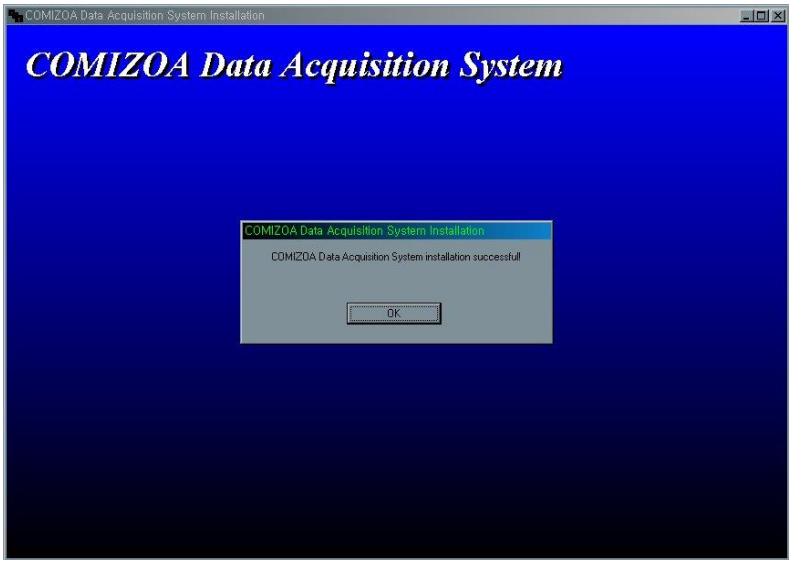
1-3 COMIDAS 소프트웨어 설치

본 장에서는 테스트 프로그램과 라이브러리 및 예제 등을 설치하는 방법을 설명합니다. 본 장에서 설명하는 설치를 수행하시면 도스용 라이브러리와 예제 파일들도 함께 설치됩니다.

제공된 설치 CD 에서 WindowsWPrograms 폴더에 있는 Setup.exe 를 실행시키십시오. 그러면 그림 1-14 와 같은 화면이 나타납니다.

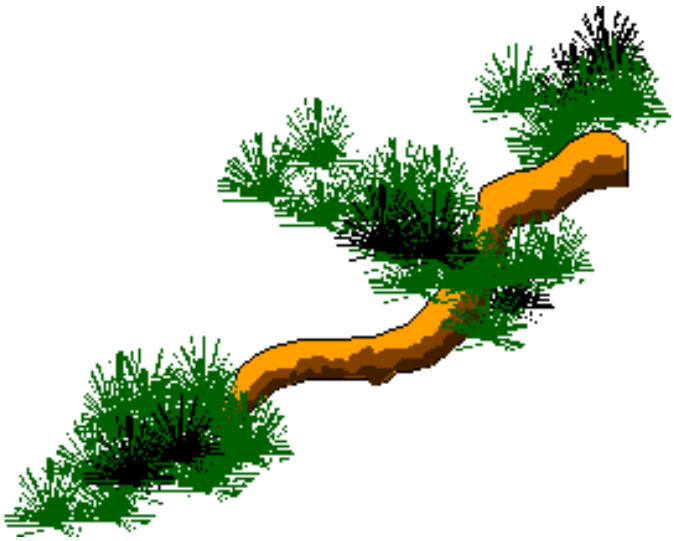
[그림 1-14] S/W 설치 화면 1

'COMIZOA Data Acquisition System Directory' 항목에서 COMIDAS 소프트웨어들을 설치할 디렉토리를 설정하십시오. 'LabWindows CVI Run Time Engine Directory' 항목에는 아무 디렉토리나 설정하셔도 됩니다. 특별한 이유가 없으면 기본값을 그냥 놓아 두시는 것이 좋습니다. 디렉터리 설정을 하셨으면 'Finish' 버튼을 클릭하십시오. 그러면 필요한 파일들을 복사한 후 그림 1-15 와 같이 설치가 끝났다는 메시지가 나타납니다. 여기서 'OK' 버튼을 클릭하면 설치가 끝납니다.



[그림 1-15] S/W 설치 종료 화면

CHAPTER 2
COMI-Master 유틸리티 소프트웨어



CHAPTER 2. COMI-Master 유틸리티 소프트웨어

1 장에서 설명한 소프트웨어 설치를 마치면 Comidasmaster.exe 라는 테스트 프로그램이 설치됩니다. 이 프로그램은 모든 종류의 COMIDAS 디바이스에 공통적으로 사용할 수 있는 프로그램으로써 COMIDAS 디바이스의 모든 기능을 테스트해볼 수 있으며, 여러 가지 방법으로 A/D 데이터를 저장, 디스플레이 해주는 기능이 있습니다.

윈도우의 시작메뉴에서 보면 'COMIZOA Data Acquisition System' 이란 메뉴가 나타날 것입니다. 이 메뉴를 선택하면 COMIDAS 테스트 프로그램을 실행할 수 있습니다.

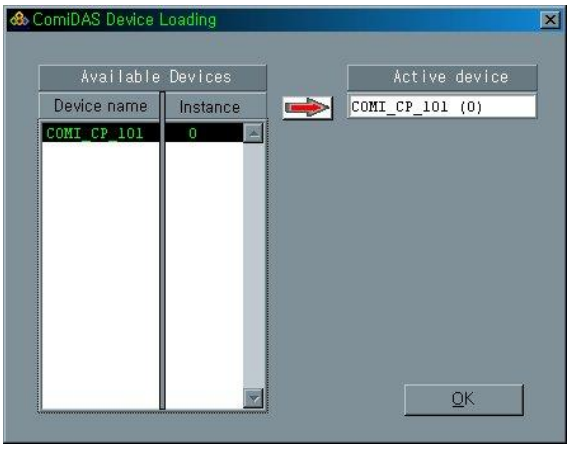
2-1 General

2-1-1 Device Loading

'Device Loading' 메뉴는 Active device 를 선택하는 메뉴입니다. Active device 는 본 테스트 프로그램에서 각 기능을 수행할 때 그 대상이 되는 디바이스를 말합니다. 'Device Loading' 메뉴를 선택하면 [그림 2-1]과 같은 화면이 나타납니다.

[그림 2-1]의 화면에서 'Available Devices' 항목은 현재 사용자의 컴퓨터에 장착되어 있는 COMIDAS 디바이스들을 리스트 해줍니다. 이 리스트에 나타나 있는 디바이스 중에서 원하는 디바이스를 마우스 더블 클릭하거나 화살표 버튼을 클릭하면 해당 디바이스가 'Active Device' 항목에 나타나며, Active device 로 선택됩니다. 'Active Device' 항목은 현재의 Active device 를 나타내 줍니다.

본 테스트 프로그램은 기본적으로 프로그램 시작 시에 첫 번째 디바이스를 Active device 로 설정합니다. 따라서 하나의 디바이스만이 컴퓨터에 장착되어 있는 경우에는 Device Loading 메뉴를 수행할 필요가 없습니다.



[그림 2-1] Device Loading 메뉴를 선택했을 때 나타나는 화면

2-1-2 General Test

'General Test' 는 Active device 선택된 디바이스의 전반적인 기능을 테스트해보는 메뉴입니다. General Test 메뉴를 선택하면 [그림 2-2]와 같은 화면이 나타납니다. 사용자는 이 곳에서 Analog Input, Analog Output, Digital Input/Output 등의 기능을 모두 테스트해볼 수 있습니다.

◈ Digital Output

Digital Output 항목을 통하여 사용자는 해당 디바이스의 각 Digital Output 채널에 출력을 내보낼 수 있습니다. 각 채널에 해당하는 버튼을 ON(눌려진 상태)시키면 해당 Digital Output 채널에서는 5 volt 가 출력이 되고, OFF 시키면 해당 Digital Output 채널에서는 0 volt 가 출력됩니다.

◈ Digital Input

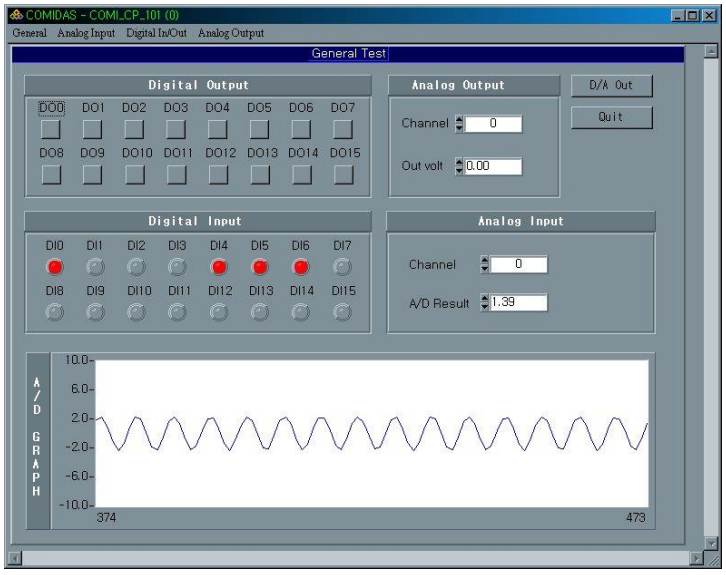
Digital Input 항목을 통하여 사용자는 해당 디바이스의 각 Digital Input 채널의 상태를 읽을 수 있습니다. 각 LED 에 적색 불이 켜지면 해당 Digital Input 채널의 상태가 ON 상태임을 의미하며, 회색이면 OFF 상태임을 의미합니다.

◈ Analog Output

Analog Output 항목을 통하여 사용자는 해당 디바이스의 각 Analog Output 채널에 원하는 전압을 출력할 수 있습니다. 'Channel' 항목에 원하는 Analog Output 채널을 입력하고, 'Out volt' 항목에 출력 전압을 입력한 후 'D/A Out' 버튼을 클릭하면 원하는 채널에 원하는 전압이 출력됩니다.

◈ Analog Input

Analog Input 항목을 통하여 사용자는 해당 디바이스의 각 Analog Input 채널에서 A/D 값을 읽을 수 있습니다. 'Channel' 항목에 원하는 Analog Input 채널을 입력하면 'A/D Result' 항목과 그래프에 A/D 결과값이 표시됩니다.

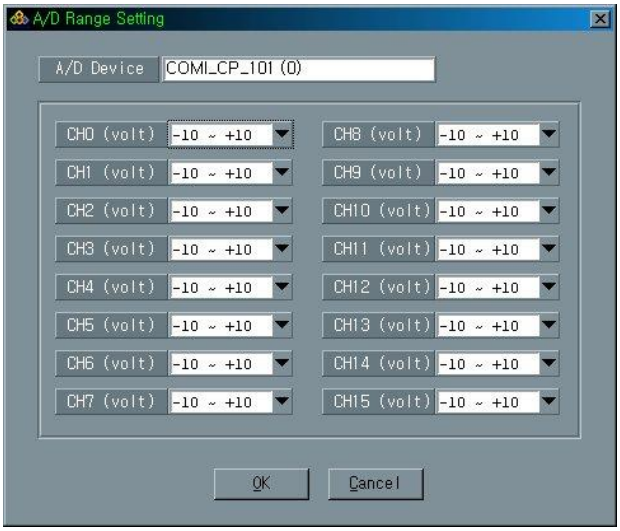


[그림 2-2] General Test 화면

2-2 Analog Input

2-2-1 A/D range setting

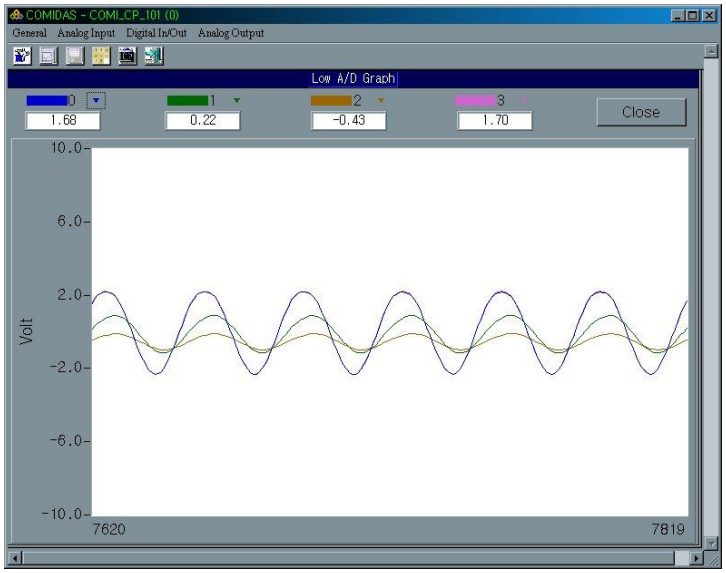
Active device 의 각 A/D 채널의 입력 범위를 설정한다. 'A/D range setting' 메뉴를 선택하면 [그림 2-3]과 같은 화면이 나타난다. 여기서 각 채널별로 입력 범위를 선택한 후 'OK' 버튼을 클릭하면 된다.



[그림 2-3] A/D range setting 화면

2-2-2 Low frequency A/D

Low frequency A/D 는 저속으로 A/D 를 수행하고, 그 결과를 그래프로 디스플레이 하거나 파일에 저장하는 기능이다. Low frequency A/D 는 윈도우 타이머 이벤트를 받아서 타이머 이벤트가 발생할 때마다 각 채널의 A/D 를 수행합니다. 따라서 A/D 하는 주기가 정확하게 지켜지지는 않습니다.



[그림 2-4] Low frequency A/D 가 수행되는 화면

'Low frequency A/D' 메뉴를 선택하면 [그림 2-4]에서와 같이 몇 개의 버튼으로 구성된 Control bar 가 나타납니다. 이 Control bar 는 Low frequency A/D 를 수행하는데 있어서 여러 가지 옵션이나 동작을 설정합니다. Control bar 의 각 버튼의 기능은 다음과 습니다.

◈ 환경설정

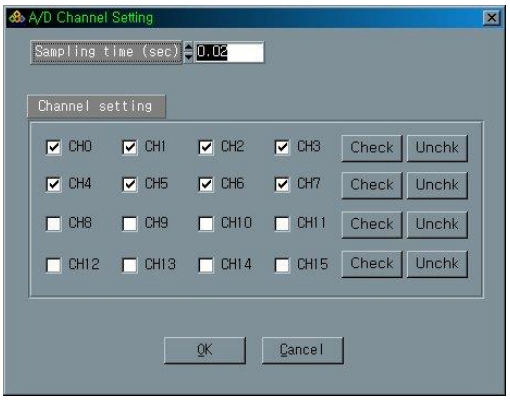
 의 그림으로 나타나는 '환경설정' 버튼은 A/D 를 수행할 채널과 샘플링 주기를 설정합니다. 이 버튼을 클릭하면 [그림 2-5]와 같은 화면이 나타납니다.

'Sampling time' 항목은 A/D 를 수행할 주기를 나타냅니다. 기본적으로 이 값은 0.02 sec 로 입력되어 있는데, 이는 0.02 초마다 각 채널의 A/D 값을 읽어 들인다는 의미입니다. 단, 앞에서 언급한 바와 같이 Low frequency A/D 는 윈도우 타이머 이벤트를 받아서 하므로 그 주기가 정확하게 보장되지는 않습니다.

'Channel setting' 항목은 A/D 를 수행할 채널을 설정하는 것입니다. A/D 를

수행할 채널을 체크 합니다. 오른쪽의 'Check' 버튼과 'Unchk' 버튼은 4 개 채널단위로 채널을 체크 또는 언체크(Uncheck) 합니다.

샘플링 주기와 채널을 설정하였으면 'OK' 버튼을 클릭하면 설정 내용이 적용됩니다.



[그림 2-5] Low frequency A/D 환경 설정 화면

◈ Create a graph

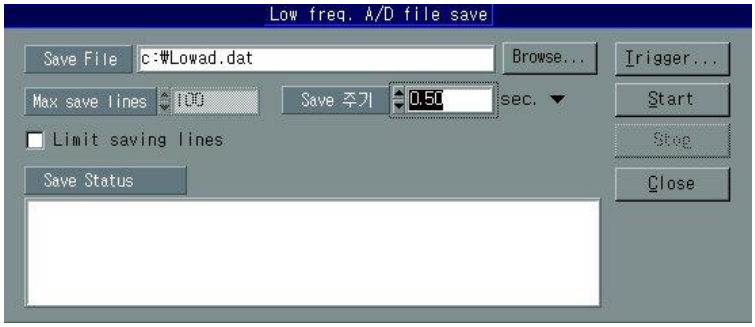

의 그림으로 나타나는 'Create a graph' 버튼은 새로운 그래프 윈도우를 생성합니다. Low frequency A/D 에서는 그래프를 사용자가 원하는 수 만큼 생성할 수 있습니다.

◈ Save data


의 그림으로 나타나는 'Save data' 버튼은 A/D 데이터를 텍스트 파일로 저장하는 기능을 수행합니다. 'Save data' 버튼을 누르면 [그림 2-6]과 같은 화면이 나타납니다. 'Save File' 항목에 원하는 파일명을 입력한 후 'Start' 버튼을 누르면 A/D 데이터를 파일에 저장하기 시작하며, 'Stop' 버튼을 누르면 저장이 종료됩니다. 파일이 저장되는 동안에는 'Scan Status' 항목에 파일에 저장된 데이터 수를 보여줍니다.

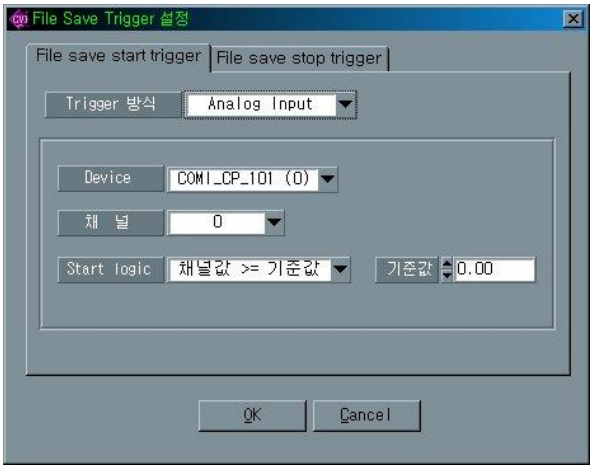
'Limit saving lines' 항목을 체크하면 파일에 저장되는 데이터 수를 제한해 줍니다. 또한 'Save 주기' 는 데이터 저장 주기를 결정하는데, 이 값은 샘플

링 주기의 배수이어야 합니다.



[그림 2-6] Low freq. A/D file save control panel 화면

'Trigger..' 버튼은 파일 저장을 시작하고 종료하는 신호를 선택하는 데 사용된다. 'Trigger..' 버튼을 클릭하면 [그림 2-7]과 같이 대화상자가 나타난다.



[그림 2-7] File save trigger 설정 화면

여기서 사용자는 파일저장 시작/종료 신호를 설정할 수 있다. File save start trigger 페이지의 'Trigger 방식' 항목에는 세가지 옵션이 있는데 각각의 의미는 다음과 같다.

▷ **Button** : 'Start' 버튼이 눌리면 바로 파일저장을 시작한다.

▷ **Digital Input** : 'Start' 버튼이 눌린 후 파일저장을 바로 시작하지 않고,

'채널' 항목에서 지정한 D/I 채널의 State 를 모니터링하여 'Start Logic' 항목에서 지정한 State 상태(High 또는 Low)가 되면 파일저장을 시작한다.

▷ **Analog Input** : 'Start' 버튼이 눌린 후에 '채널' 항목에서 지정한 A/D 채널의 값을 계속 계측하고, 그 값을 '기준값' 항목에서 지정한 값과 비교하여 'Start Logic' 항목에서 지정한 logic 이 참이면 파일 저장을 시작한다.

※ Low frequency A/D 저장 데이터 파일 형식 : 데이터는 아스키 파일에 저장되며, 아스키 파일의 형식은 다음과 같습니다. 아래에서 ElapsedT 열은 각 데이터가 저장되는 순간에 저장 시작 시간으로부터 경과된 시간을 나타냅니다.

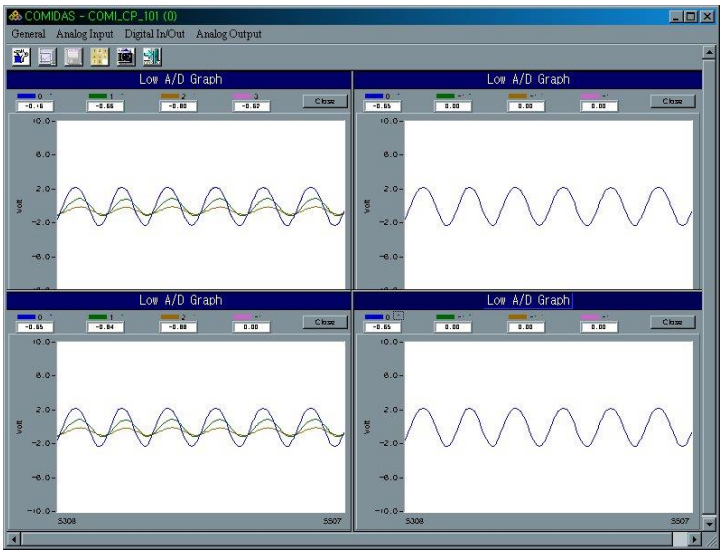
Start Date : 01-04-2001

Start Time : 10:39:24

ElapsedT	CH 0	CH 1	CH 2	CH 3	CH 4	CH 5	CH 6	CH 7
0.01	0.574	2.322	-0.217	-0.212	-0.271	-0.339	-0.540	-0.759
0.06	0.574	1.600	0.159	-0.090	-0.188	-0.281	-0.422	-0.598
0.16	0.569	2.747	0.569	0.081	-0.139	-0.271	-0.496	-0.764
0.26	0.574	2.537	0.789	0.193	-0.081	-0.247	-0.481	-0.745
0.36	0.574	0.872	0.618	0.154	-0.051	-0.222	-0.466	-0.730
0.46	0.574	-1.360	0.105	-0.007	-0.071	-0.212	-0.452	-0.720
0.56	0.574	-2.762	-0.462	-0.212	-0.115	-0.208	-0.369	-0.554
0.66	0.574	-2.391	-0.750	-0.330	-0.168	-0.227	-0.383	-0.564
0.76	0.574	-0.510	-0.593	-0.310	-0.188	-0.237	-0.393	-0.574
0.86	0.574	1.673	-0.105	-0.178	-0.183	-0.242	-0.398	-0.584
0.96	0.574	2.757	0.383	-0.032	-0.159	-0.247	-0.471	-0.745
1.06	0.574	2.488	0.632	0.071	-0.115	-0.242	-0.462	-0.730
1.16	0.569	0.764	0.481	0.037	-0.100	-0.227	-0.452	-0.716
1.26	0.574	-1.463	-0.007	-0.115	-0.129	-0.232	-0.452	-0.706
.....								

◈ Auto arrange

의 그림으로 나타나는 'Auto arrange' 버튼은 여러 개의 그래프 윈도우가 생성되었을 경우 자동으로 각 그래프 윈도우의 크기나 위치를 조절하여 배치하는 기능이다. [그림 2-8]화면은 4 개의 그래프를 자동 배치한 화면입니다.



[그림 2-8] Auto arrange 기능을 이용하여 자동 배치된 화면

◈ Save settings

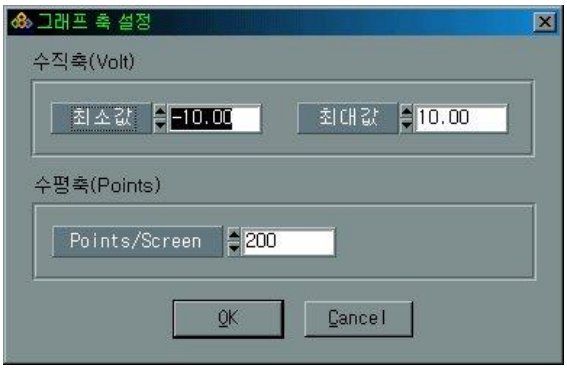
의 그림으로 나타나는 'Save settings' 버튼은 현재의 채널, 샘플링 주기, 화면 배치 등의 설정 값을 저장하는 기능을 합니다. 이렇게 저장된 설정 값은 다음에 다시 Low frequency A/D 를 수행할 때 로드 됩니다.

◈ Exit

의 그림으로 나타나는 'Exit' 버튼은 Low frequency A/D 를 종료합니다.

◈ 그래프 축 설정

각 그래프를 오른쪽 마우스 클릭하면 '그래프 축 설정'이라는 팝업 메뉴가 나타납니다. 이 메뉴를 선택하면 [그림 2-9]와 같이 각 그래프의 축을 설정할 수 있는 대화상자가 나타납니다. 이 곳에서 각 그래프의 축을 설정할 수 있습니다.



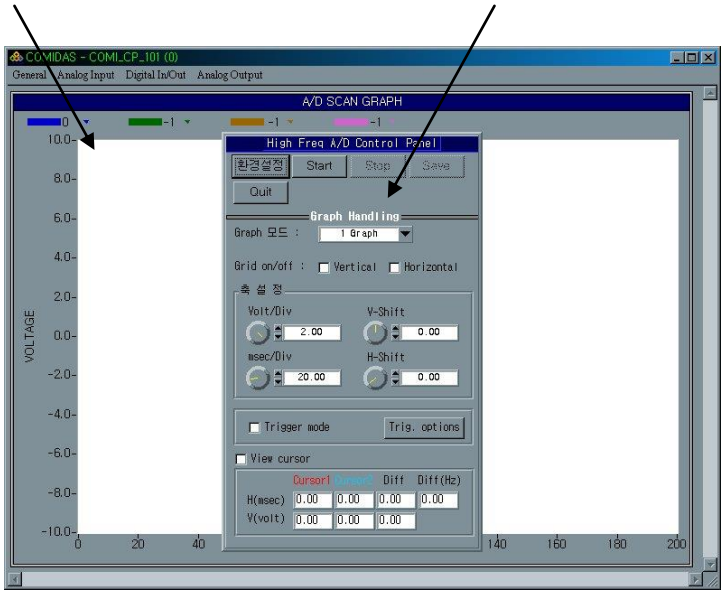
[그림 2-9] Low frequency A/D 그래프 축 설정 대화상자

2-2-3 High frequency A/D

High frequency A/D 는 고속으로 A/D 를 수행하고, 그 결과를 그래프로 디스플레이 하거나 파일에 저장하는 기능입니다. 특히 High frequency A/D 는 COMIDAS 라이브러리의 Unlimited A/D scan 기능을 사용하여 샘플링 주기가 정확하게 보장됩니다. High frequency A/D 는 오실로스코프와 비슷한 기능을 수행할 수 있으며, 고속 A/D 데이터를 실시간으로 파일저장을 할 수 있습니다.

'High frequency A/D' 메뉴를 선택하면 [그림 2-10]과 같이 그래프 윈도우와 Control panel 이 나타납니다. High frequency A/D 의 모든 작업은 Control panel 에서 제어하며, 그래프 윈도우는 High frequency A/D data 를 그래프로 보여줍니다.

그래프 윈도우 High freq. A/D Control panel



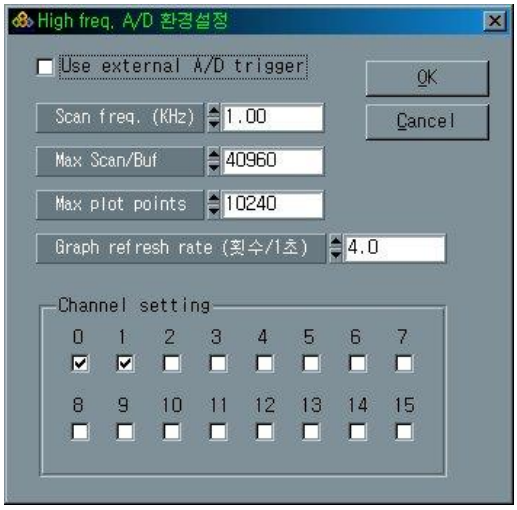
[그림 2-10] High frequency A/D 그래프 윈도우와 Control panel

1) High frequency A/D 환경 설정

High frequency A/D 를 시작하기 전에 먼저 환경 설정을 해주어야 합니다. 앞에서 언급한 바와 같이 High frequency A/D 는 Unlimited scan 기능을 이용합니

다. 따라서 High frequency A/D 의 환경 설정은 대부분 Unlimited scan 시작함수의 파라미터들입니다. Unlimited scan 에 대한 자세한 내용은 라이브러리 매뉴얼을 참조하시기 바랍니다.

Control panel 에서 '환경설정' 버튼을 클릭하면 [그림 2-11]과 같이 환경설정을 하는 대화상자가 나타납니다. 각 항목에 적절한 값을 입력한 후 'OK' 버튼을 누르면 환경설정이 적용됩니다.



[그림 2-11] High frequency A/D 환경설정 대화상자

환경설정 대화상자에서 각 항목의 의미는 다음과 같다.

▷ Use external A/D trigger - 이 항목을 체크하면 A/D trigger 소스클럭을 외부에서 입력한다. 즉, 외부의 펄스 신호를 이용하여 A/D trigger 를 하는 것입니다.

▷ Scan freq. - 이 항목은 A/D Scanning frequency 를 의미합니다. 이 값은 각 채널 당 Sampling frequency 가 된다는 것을 명심하셔야 합니다. CP 시리즈 보드의 경우에는 동시 샘플링이 지원되지 않으므로 이 값과 채널 수를 곱한 값이 20 KHz 이상이 되면 인터럽트가 너무 자주 발생되는 관계로 시스템이 정지될 수 있습니다.

▷ Max Scan/Buf - 이 항목은 Scan buffer 의 크기를 결정합니다. 이 항목은 COMIDAS 라이브러리 함수 중 COMI_US_Start()함수의 'bufSize' 파라미터와 같은 의미입니다. 자세한 내용은 COMIDAS 라이브러리 매뉴얼을 참조하시기 바랍니다.

▷ Max plot points - 이 항목은 한 화면에 보여줄 수 있는 데이터 포인트의 최대 수를 지정합니다. 이 값은 반드시 'Max Scan/Buf' 항목의 입력 값보다 작아야 합니다.

▷ Graph refresh rate - 이 항목은 그래프를 갱신하는 속도를 지정합니다.

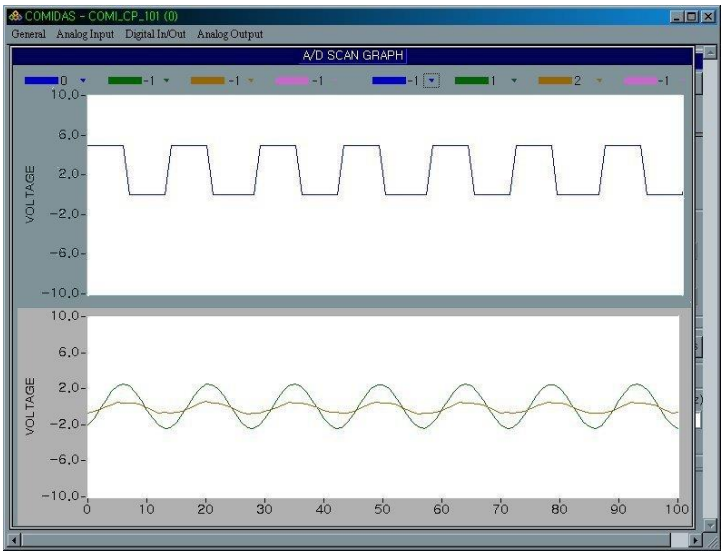
▷ Channel setting - A/D scan 을 수행할 채널을 선택합니다.

2) Graph control 하기

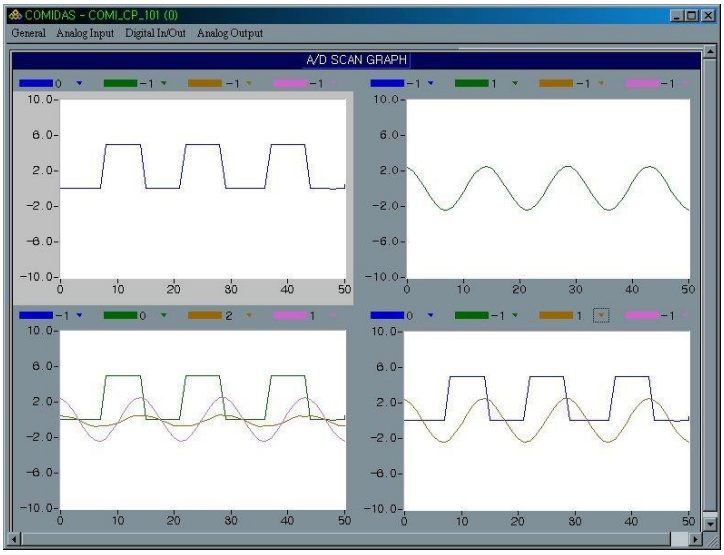
Control panel 에서 'Start' 버튼을 클릭하면 A/D Scan 이 시작되고 그래프 윈도우에는 Scan 된 A/D 데이터들이 디스플레이 됩니다. 사용자는 Control panel 에 있는 여러 가지 컨트롤을 이용하여 그래프를 다양하게 제어할 수 있습니다. Control panel 에서 그래프를 제어하는 각 컨트롤들은 다음과 같습니다.

◈ Graph 모드

'Graph 모드' 항목은 그래프 윈도우의 그래프 수를 지정합니다. High frequency A/D 에서 그래프 윈도우는 1 개의 그래프 또는 2 개의 그래프 또는 4 개의 그래프로 구성될 수 있습니다. 1 Graph 모드를 선택하면 그래프 윈도우는 [그림 2-10]과 같이 나타나며, 2 Graph 모드와 4 Graph 모드는 각각 [그림 2-12]와 [그림 2-13]과 같이 나타납니다. 2 Graph 와 4 Graph 모드에서 테두리가 밝은 회색으로 표시되는 그래프는 활성 그래프임을 표시하는 것이며 Control panel 에서 이루어지는 그래프 제어는 이 활성 그래프에 적용됩니다. 원하는 그래프를 활성 그래프로 하는 방법은 해당 그래프를 클릭하면 됩니다.



[그림 2-12] 2 Graph 모드를 선택했을 때의 그래프 윈도우 화면



[그림 2-13] 4 Graph 모드를 선택했을 때의 그래프 윈도우 화면

◈ Grid on/off

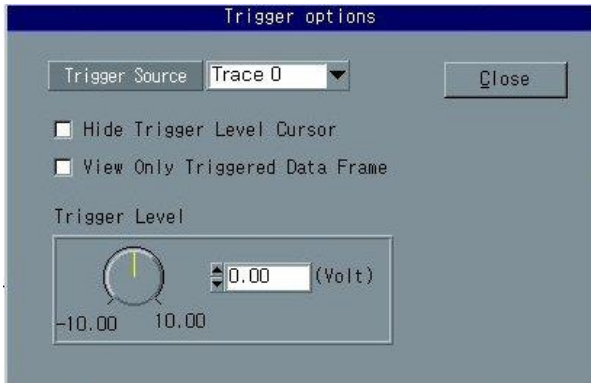
'Grid on/off' 항목은 활성 그래프에 Grid 를 표시하거나 숨기는 역할을 합니다.

◈ 축 설정

이 그룹의 각 항목은 오실로스코프의 Volt/Div, Sec/Div, Position 제어 손잡이에 해당합니다. 이 각 항목들은 수평/수직축의 범위를 설정하거나, trace를 shift 하는 기능을 수행합니다.

◈ Trigger mode

여기에서 Trigger 모드는 오실로스코프에서의 Trigger 기능과 같은 기능을 하는 모드입니다. 이 항목을 체크하면 Trigger 모드가 Enable 됩니다. 우측에 있는 'Trig. Options' 버튼을 클릭하면 [그림 2-14]와 같은 화면이 나타나는데, 이 곳에서 Trigger 레벨이나, Trigger 소스 등을 설정합니다.



[그림 2-14] Trigger options 설정 화면

▷ Trigger Source - Trigger source 를 선택합니다. Trigger source 는 채널 번호로 선택하지 않고 그래프상의 Trace 번호로 선택한다.

▷ Hide Trigger Level Cursor - 이 항목을 체크하면 Trigger level 을 그래프상에 표시하지 않습니다.

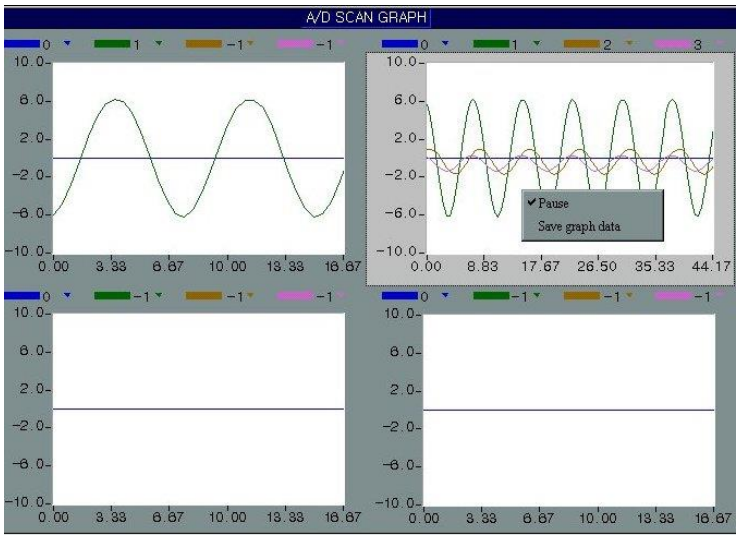
▷ View Only Triggered Data Frame - 이 항목을 체크하면 Trigger 가 일어날 때만 그래프를 갱신합니다. 이 것은 노이즈 등과 같이 가끔 발생되는 이상 데이터를 캐치(catch)하고자 할 때 유용하게 사용될 수 있습니다.

◈ View cursor

이 항목을 체크하면 그래프에는 2 개의 십자 모양 커서가 나타납니다. 이 커서는 그래프상에 그려진 데이터의 값을 숫자로 나타내보기 위한 것입니다. 이 커서의 위치 값은 Control panel 의 하단에 있는 숫자 창에 나타납니다. Cursor1 과 Cursor2 의 항목에서는 각 커서의 전압축과 시간 축상의 위치 값을 나타낸다. Diff 항목에서는 두 커서 사이의 차이 값 나타냅니다. 또한 Diff2 는 두 커서간의 시간차를 주파수(Hz)로 계산해서 나타내줍니다.

◈ 팝업 메뉴

각 그래프를 오른쪽 마우스 클릭하면 [그림 2-15]와 같이 'Pause' 와 'Save graph data' 메뉴로 구성된 팝업 메뉴가 나타납니다.



[그림 2-15] 그래프를 마우스 오른쪽 클릭하여 나타나는 팝업 메뉴

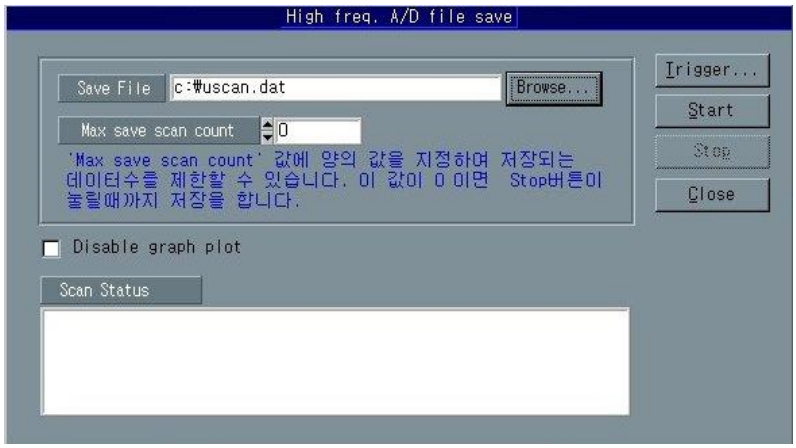
▷ 'Pause' 메뉴 - 이 메뉴는 해당 그래프에 그리기를 일시 중단하고자 할 때 사용합니다. Pause 된 상태에서 다시 한번 선택하면 다시 그리기를 시작합니다.

▷ 'Save graph data' 메뉴 - 이 메뉴는 그래프가 Pause 되었거나 'Stop' 버튼이 눌렸을 경우에 선택할 수 있는 메뉴로써 이 메뉴를 선택하면 현재

그래프에 그려진 데이터를 텍스트 파일로 저장하게 됩니다. 이 메뉴를 선택하면 파일을 선택할 수 있도록 하는 대화상자가 나타나고, 사용자가 파일을 선택하면 그 파일에 그래프에 그려진 데이터를 저장합니다. 단, 이것은 현재 그래프에 보여지는 데이터만 저장이 됩니다. 따라서 모든 데이터를 실시간으로 저장하기 위해서는 다음 단원에서 소개되는 기능을 사용하셔야 합니다.

3) 데이터를 파일로 저장하기

Control panel 에서 'Save' 버튼을 클릭하면 현재 수행되고 있는 A/D Scan 에서 획득한 데이터를 실시간으로 파일에 저장할 수 있습니다. 'Save' 버튼을 클릭하면 [그림 2-16]과 같은 화면이 나타납니다. 이 곳에서 파일 저장을 수행할 수 있습니다. High freq. A/D 에서 파일 저장하는 방법은 Low freq. A/D 에서와 거의 동일합니다. 따라서 자세한 내용은 생략하고 Low freq. A/D 에서 없었던 기능만 설명하도록 하겠습니다.



[그림 2-16] High freq. A/D 파일 저장 화면

'Max save scan count' 항목은 파일에 저장되는 데이터 수를 제한하는 것입니다. 'Disable graph plot' 항목은 파일 저장하는 동안에는 그래프에 데이터를 그리지 않도록 합니다.

2-3 Digital In/Out

2-3-1 DIO Usage Selection

이 메뉴는 COMI-CP401 Digital I/O Board 에만 해당하는 기능입니다. COMI-CP401 보드는 총 32 개의 DIO 채널이 있습니다. 사용자에 따라 다음의 3 가지 모드 중 하나를 선택하여 32 채널의 용도를 선택할 수 있습니다.

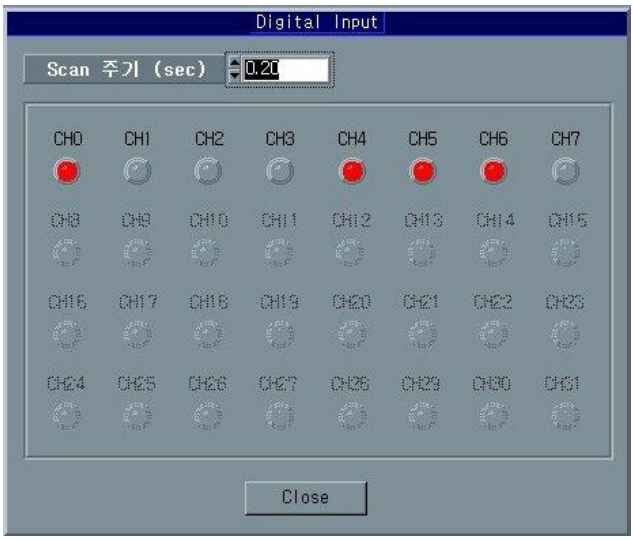
- 32 채널 모두 Digital Input 전용으로 사용
- 32 채널 모두 Digital Output 전용으로 사용
- Digital In/Out 각각 16 채널씩 사용

[그림 2-17] DIO Usage Selection 화면

2-3-2 Digital Input

이 메뉴는 Digital Input 기능을 수행합니다. 'Digital Input' 메뉴를 선택하면 [그림 2-18]과 같은 화면이 나타납니다. 화면에서 각 LED 는 해당 D/I 채널의 상태를 표시합니다. 그림에서 Dimmed 처리된 LED 는 사용 불가능한 채널을 표시합니다. 디바이스에 따라 가용한 D/I 채널의 수는 달라집니다.

'Scan 주기' 항목은 각 Digital Input 채널의 상태를 체크 하는 주기를 지정합니다.



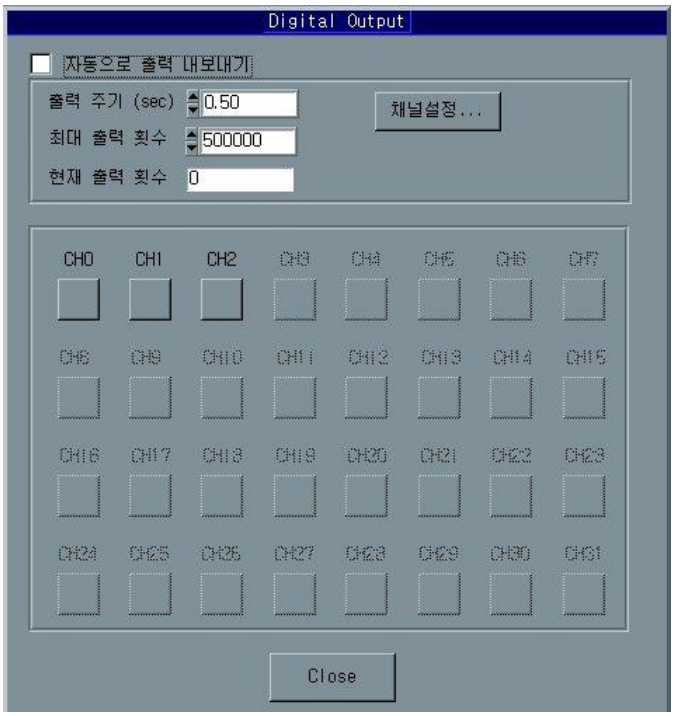
[그림 2-18] Digital Input 화면

2-3-3 Digital Output

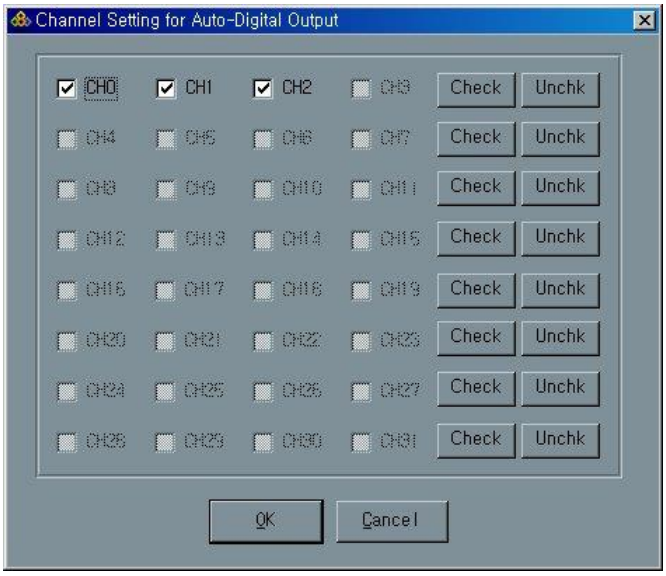
이 메뉴는 Digital output 기능을 수행합니다. 'Digital Output' 메뉴를 선택하면 [그림 2-19]와 같은 화면이 나타납니다. 화면에서 각 채널에 해당하는 버튼을 클릭하여 Digital output 채널의 ON/OFF 상태를 제어할 수 있습니다.
만일 주기적으로 ON/OFF 상태를 반복하여 출력하고 싶다면 화면 상단의 "자동으로 출력 내보내기" 항목을 선택하십시오. 그러면 선택한 Digital output 채널의 출력(ON 또는 OFF)이 지정한 주기에 따라 주기적으로 변하게 됩니다.

자동 출력을 적용할 채널을 선택하려면 "채널설정" 버튼을 클릭하십시오. 그러면 [그림 2-20]과 같이 채널을 선택할 수 있는 화면이 나타납니다. [그림 2-20]에서 필요한 채널들을 선택하신 후 "OK" 버튼을 누르면 해당 채널들이 선택됩니다.

"최대 출력 횟수" 항목은 Digital output 출력 변화 횟수를 제한하고자 할 때 사용됩니다. 이 항목에서 지정한 횟수만큼 Digital output 의 출력 변화가 이루어지면 자동으로 출력을 중지합니다.



[그림 2-19] Digital Output 화면

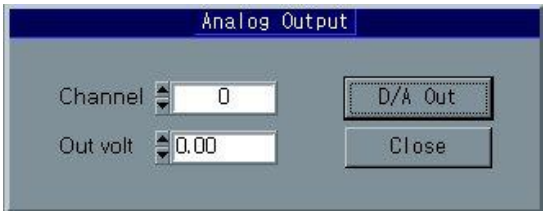


[그림 2-20] 자동 Digital Output 채널 선택 화면

2-4 Analog Output

2-4-1 Single Point D/A

이 메뉴는 Analog output 채널에 사용자가 원하는 전압을 출력합니다. 주메뉴에서 Analog Output => Single Point D/A 메뉴를 선택하면 [그림 2-21]과 같은 화면이 나타납니다. 화면에서 'Channel' 항목에 D/A 채널을, 'Out volt' 항목에 출력 전압을 입력하고 'D/A Out' 버튼을 클릭하면 해당 채널에 지정한 전압이 출력됩니다.

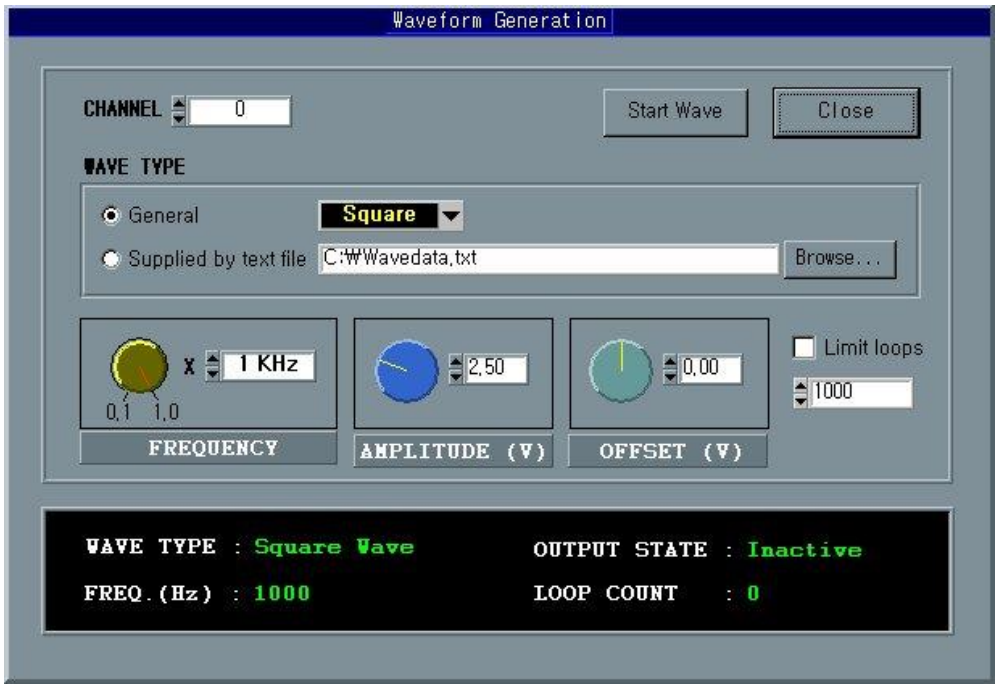


[그림 2-21] Single Point D/A 화면

2-4-2 Waveform Generation

이 메뉴는 Waveform Generation 기능을 수행한는 기능으로써 COMI-SD301 보드에서만 지원가능한 기능입니다. Waveform Generation 은 사용자가 지정한 임의의 주기신호를 Analog Output 채널을 통하여 출력하는 기능입니다. 본 응용프로그램에서는 COMI-SD301 보드가 지원하는 Waveform Generation 기능을 이용하여 Square Wave, Triangle Wave, Sine Wave 형태의 Waveform 신호를 발생하는 기능을 구현하였습니다.

주 메뉴에서 Analog Output => Waveform Out 메뉴를 선택하시면 [그림 2-22]와 같은 화면이 나타납니다. 'Start Wave' 버튼을 누르면 지정한 Waveform 신호가 지정된 D/A Channel 을 통해 출력되게 됩니다.



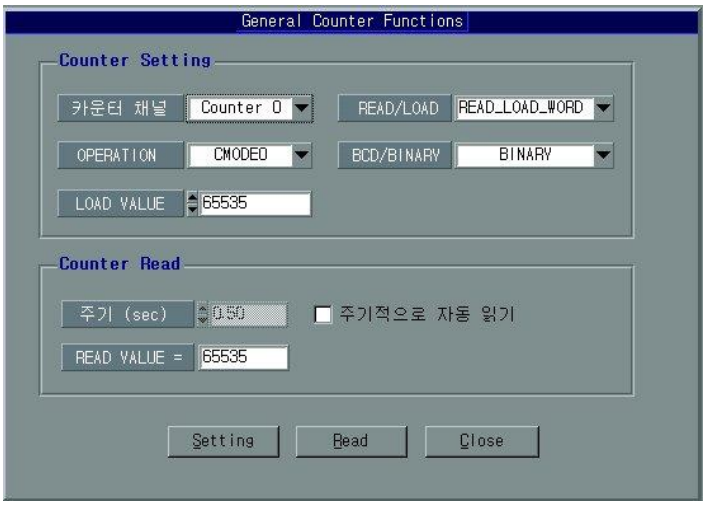
[그림 2-22] Single Point D/A 화면

2-5 Counter/Timer

2-5-1 General Functions

이 기능은 Counter 채널을 사용자가 직접 셋팅해보고, 읽어보도록 하기 위한 것입니다. 이 메뉴를 선택하면 [그림 2-23]과 같은 화면이 나타납니다. 'Counter Setting' 그룹에 있는 각 항목들을 설정하고 'Setting' 버튼을 클릭하면 카운터가 셋팅됩니다. 카운터 셋팅에 대해서는 라이브러리 매뉴얼의 부록편을 참조하십시오. 카운터가 셋팅되면 카운터는 소스 클럭을 카운트하기 시작하고 동작(Operation) 모드에 따라 카운터 출력 단자에 출력을 내보냅니다. 카운터의 출력은 동작(Operation) 모드에 따라 다릅니다.

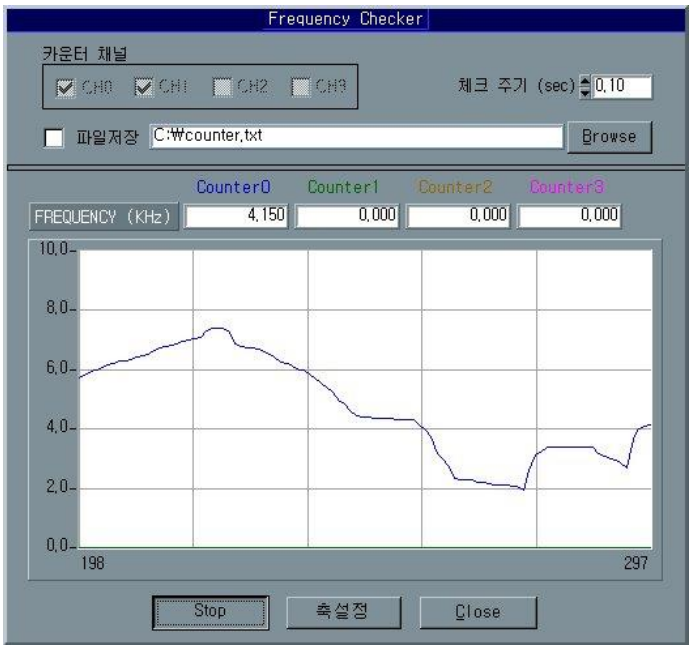
'Read' 버튼을 클릭할 때마다 카운터에서 카운트 값을 읽어서 'READ VALUE' 항목에 나타냅니다. 8254 카운터는 Decrease 방식으로 카운트하게 됩니다. 따라서 실제 카운트 값은 LOAD VALUE 에서 READ VALUE 를 뺀 값이 됩니다. '주기적으로 자동읽기' 항목을 체크하시면 'Read' 버튼을 누르지 않아도 주기적으로 카운트 값을 읽어서 화면에 나타내줍니다.



[그림 2-23] 카운터의 General Functions 화면

2-5-2 Frequency Checker

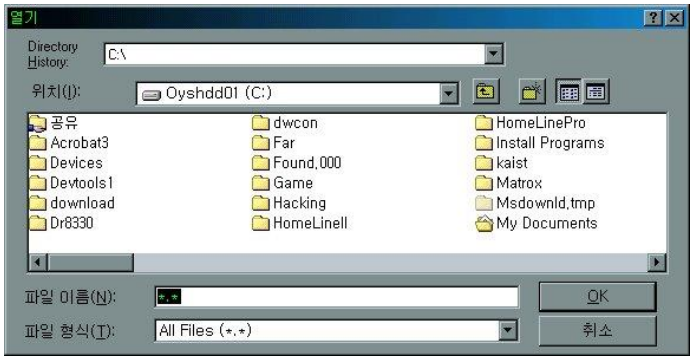
이 기능은 카운터를 이용하여 펄스의 주파수를 측정해주는 기능입니다. 이 기능은 모터 등의 속도측정시 유용하게 사용될 수 있습니다. 'Frequency checker' 메뉴를 선택하면 [그림 2-24]와 같은 화면이 나타납니다. 카운터 채널 항목에서 원하는 카운터 채널을 선택합니다. 동시에 여러 개의 카운터 채널을 선택해도 관계없습니다. 카운터를 선택했으면 'Start' 버튼(화면에서는 'Stop'으로 나와 있는 버튼, 이 버튼은 토글 버튼으로써 'Start'와 'Stop' 공용입니다)을 클릭하면, 체크 주기에서 지정한 시간마다 지정 카운터 채널에 입력되는 펄스의 주파수를 측정하여 숫자와 그래프로 나타내줍니다.



[그림 2-24] Frequency checker 화면

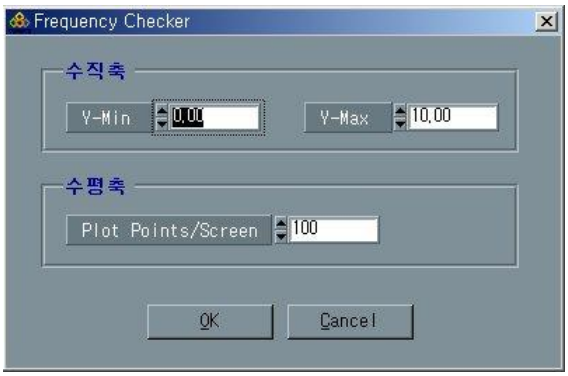
속도 데이터를 저장하고자 한다면 '파일저장' 체크박스를 선택하십시오. 그러면 체크박스 오른쪽에 스트링으로 지정된 파일에 데이터를 아스키형태로 저장합니다. 파일명을 바꾸시려면 에디트박스에 직접 파일명을 입력하거나, 'Browse' 버튼을 눌러 원하는 파일을 선택하십시오. 'Browse' 버튼을 누르면

[그림 2-25]와 같이 파일을 선택할 수 있도록 하는 대화상자가 나타납니다.



[그림 2-25] File browse 대화상자

그래프의 축을 변경하려면 '축설정' 버튼을 클릭하십시오. 그러면 [그림 2-26]과 같이 그래프의 축을 변경할 수 있도록 하는 대화상자가 나타납니다. 여기서 수직축의 단위는 KHz 가 됩니다.

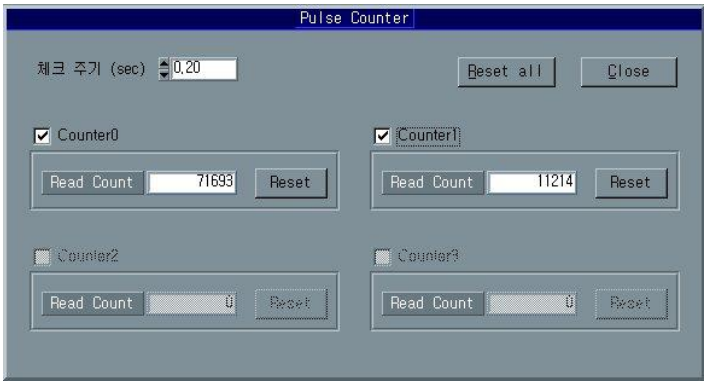


[그림 2-26] Frequency Checker 의 그래프 축설정 화면

2-5-3 Pulse Counter

이 기능은 펄스의 입력 수를 카운트하는 기능입니다. 'Pulse counter' 를 선택하면 [그림 2-27]과 같은 화면이 나타납니다. 원하는 카운터를 선택하면 선택한 순간부터 '체크 주기' 에서 설정한 시간마다 펄스 입력 수를 카운트하

여 화면에 숫자로 나타냅니다. 카운트 값을 0 으로 Reset 하려면 각 카운터의 'Reset' 버튼을 클릭하십시오. 또한 'Reset all' 버튼을 클릭하면 모든 카운터의 카운트 값이 Reset 됩니다.

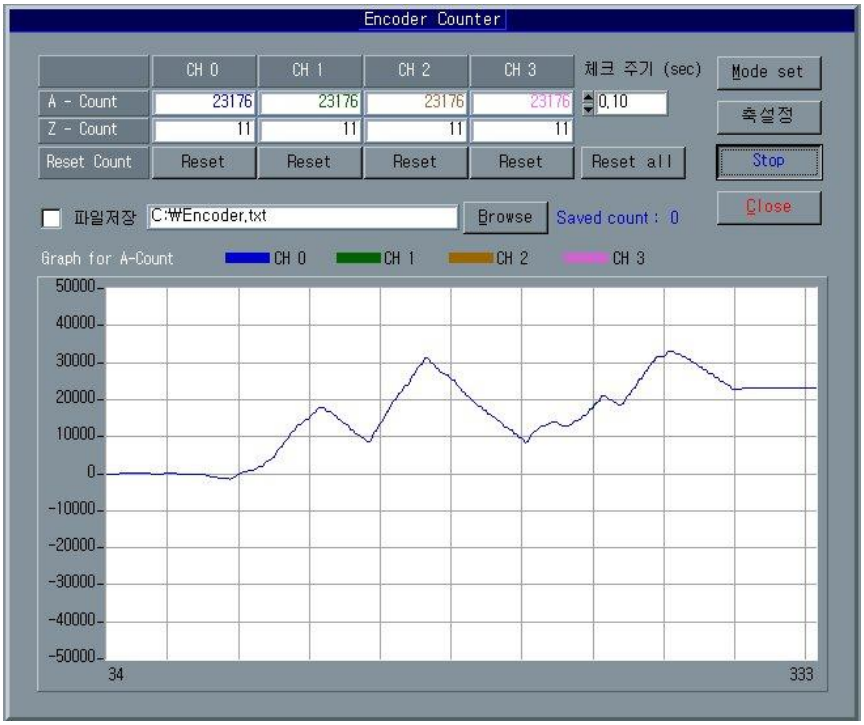


[그림 2-27] Pulse counter 화면

2-5-4 Encoder Counter

이 기능은 엔코더 센서를 계측하여 모니터링하거나 파일에 그 데이터를 저장하는 기능입니다.

메뉴바에서 **Counter/Timer ⇒Encoder counter** 메뉴를 선택하면 [그림 2-28]과 같은 화면이 나타납니다. 'Start' 버튼 을 누르면 엔코더 센서를 계측하여 화면에 카운트 값을 표시합니다. 참고로 'Start' 버튼은 버튼의 상태에 따라 버튼 라벨이 "Start" 문자와 "Stop" 문자로 토글되며, 화면에서는 "Stop" 문자로 나타내어지고 있습니다.



[그림 2-28] Encoder counter 메인 화면

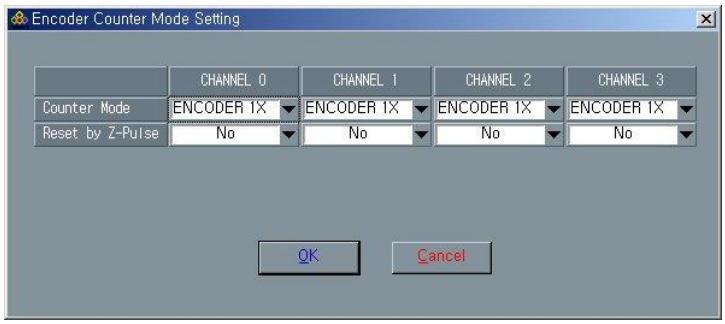
◈ 모드 설정

모드 설정은 엔코더 카운터의 다음 두 가지에 대한 옵션을 설정한다.

① 엔코더 카운터의 채배 기능

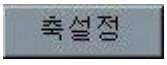
② Z-Pulse 에 의한 A/B 상 카운트 값의 리셋 여부

위의 두 가지 옵션에 대한 자세한 사항은 라이브러리 매뉴얼의 COMI_ENC_Config(…)함수의 설명을 참조하십시오. 모드 설정을 하기 위해서는 [그림 2-28]에서 'Mode set' 버튼()을 클릭하십시오. 그러면 [그림 2-29]화면이 나타납니다. 이 화면에서 위의 두 가지 옵션을 설정하십시오.

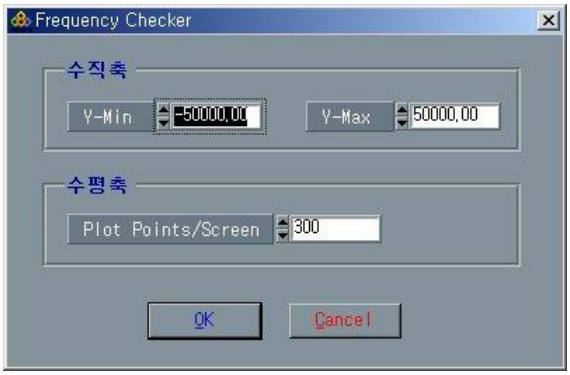


[그림 2-29] 엔코더 카운터의 모드 설정 화면

◈ 축 설정

축 설정은 엔코더 카운트 값을 화면에 나타내는 그래프의 축을 설정합니다. [그림 2-28]에서 '축설정' 버튼(

)을 클릭하면 [그림 2-30]화면이 나타납니다. 이 화면에서 그래프의 수직 축과 수평축의 범위를 설정하십시오. 여기서 수평축은 데이터 포인트 수가 됩니다. 따라서 [그림 2-28]에서 '체크주기'를 0.1 초로 하고, [그림 2-30]과 같이 'Plot Points/Screen' 값을 300 으로 지정하였다면 수평축의 범위는 30 초가 됩니다.



[그림 2-30] Encoder counter 그래프 축 설정 화면

◈ 리셋(Reset)

엔코더 카운트 값을 0 으로 리셋하려면 [그림 2-28]화면에서 각 채널에 해당하는 'Reset' 버튼(

)을 클릭하거나, 'Reset all' 버튼

(Reset all)을 클릭하십시오.

◈ 데이터 저장

[그림 2-28]화면에서 그래프의 위쪽에 [그림 2-31]과 같은 항목들을 보실 수 있습니다.

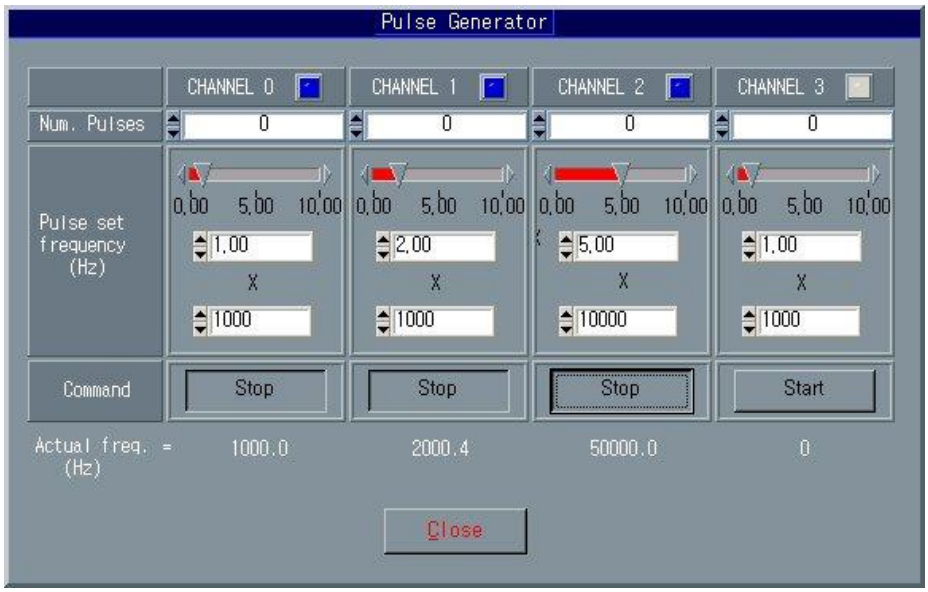
[그림 2-31]데이터 저장에 관련된 항목들

[그림 2-31]에서 파일저장 체크박스를 체크하면 그 순간부터 엔코더 카운트 값은 [그림 2-31]의 텍스트박스에 기입된 파일에 저장이 됩니다. 파일명을 바꾸고자 한다면 텍스트박스에 직접 기입하거나 'Browse' 버튼을 클릭하여 파일을 선택할 수 있습니다.

2-5-5 Pulse Generator

Pulse Generator(이후 PG) 기능은 COMI-SD501 보드에서만 제공하는 기능으로써 사용자가 원하는 주파수로 펄스를 생성시켜주는 기능입니다. 이 기능은 생성되는 펄스의 주파수를 제어할 수 있을 뿐 아니라 펄스의 수까지 제어할 수 있습니다. 이 기능은 주로 서보모터 및 스텝모터의 제어에 사용되는 기능입니다.

메뉴바에서 **Counter/Timer⇒Pulse generator** 메뉴를 선택하면 [그림 2-32] 화면이 나타납니다.



[그림 2-32] Pulse Generator 메인 화면

[그림 2-32]화면에서 각 채널에 해당하는 'Start' 버튼을 클릭하면 해당 채널을 통하여 펄스가 출력되며 'Stop' 버튼을 클릭하면 펄스 출력이 중지됩니다. 참고로, 'Start' 버튼과 'Stop' 버튼은 동일 버튼으로써 버튼의 상태에 따라 버튼 라벨이 "Start" 문자와 "Stop" 문자로 토글됩니다. 출력되는 펄스의 개수와 주파수는 사용자가 동적으로 지정할 수 있는데 이 들을 지정하는 방법은 다음을 참조하십시오. 다음은 [그림 2-32]화면에서 각 항목에 대한 설명입니다.

◈ Num. Pulses 항목

이 항목은 출력되는 펄스의 개수를 지정하는 항목입니다. 이 값에 **양의 값**을 지정하면 지정한 개수만큼만 펄스가 출력되고 자동으로 중지됩니다. 그러나 이 값에 **0** 을 지정하면 'Stop' 버튼이 눌리기 전까지 계속해서 펄스가 출력됩니다.

단, 이 값은 펄스 출력이 시작되기 전에 값을 설정하여야 하며 'Start' 버튼이 눌린 상태에서는 이 값을 변경하여도 영향을 미치지 않습니다.

◈ Pulse set frequency 항목

이 항목은 출력되는 펄스의 주파수를 지정합니다. 출력 주파수는 펄스가 출력되고 있는 도중에도 변경이 가능합니다.

주파수를 지정할 때는 화면에 나타나 있는 슬라이드바를 이용하거나 에디트 콘트롤(Edit control)을 이용하여 값을 변경할 수 있습니다. 두 개의 에디트 콘트롤 중에서 아래쪽에 있는 것은 주파수를 설정하는 게인(Gain)값입니다. 즉, 주파수는 슬라이드바나 위쪽에 있는 에디트 콘트롤의 값과 이 Gain 값이 곱해져서 계산됩니다. [그림 2-32]화면을 예로 들면 채널 0 는 1000 Hz, 채널 1 은 2000 Hz, 채널 2 는 50000 Hz 의 주파수로 설정된 것입니다.

◈ Actual frequency 항목

이 항목은 펄스의 실제 출력 주파수를 나타냅니다. 펄스의 실제 출력 주파수는 사용자가 지정한 주파수와 약간의 차이가 있을 수 있습니다.

2-6 Controls

2-6-1 고속 PID Controls

이 기능은 COMI-SD101, COMI-SD102, COMI-SD103 보드에서만 지원되는 기능으로써 A/D 와 D/A 채널을 이용하여 고속 PID 제어를 할 수 있도록 하는 기능입니다. 본 프로그램에서는 동시에 2 개의 시스템에 대한 PID 제어가 가능합니다. 본 프로그램에 적용된 PID 제어 알고리즘은 위치형(Position form) 제어 알고리즘으로써 수식으로 표현하면

$$u(t) = K_p[e(t) + \frac{1}{T_i}\int e(t)dt + T_d\frac{de(t)}{dt}] \qquad \text{(식 2-1)}$$

으로 나타납니다. (식 2-1)을 이산식으로 변형하면

$$u(n) = K_p[e(n) + \frac{T_s}{T_i}\sum_{j=1}^{n-1} e(j) + \frac{T_d}{T_s}\{e(n) - e(n-1)\}] \qquad \text{(식 2-2)}$$

과 같습니다. 여기서 K_p 는 비례 게인, T_i 는 적분 시간, T_d 는 미분시간이며, T_s 는 샘플링 시간입니다.

본 고속 PID 제어 기능은 디바이스 드라이버 내부에서 인터럽트를 이용하여 제어하므로써 제어 주기가 정확하며, 고속 실시간 제어가 가능하도록 되어 있습니다. 본 고속 PID 제어 기능은 1 ~ 15000Hz 의 범위 내에서 제어 주파수를 선택할 수 있습니다.

1) 고속 PID Controls 화면 구성

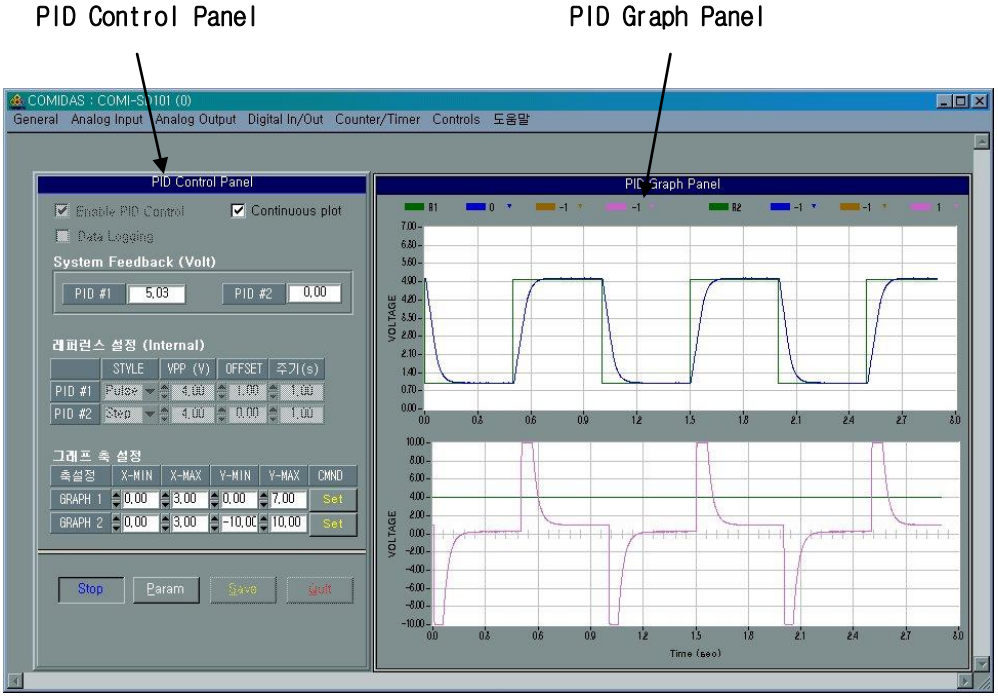
고속 PID Controls 기능의 화면 구성은 [그림 2-33]과 같이 Control panel 과 Graph panel 로 구성됩니다.

Control panel 은 PID Control 을 하기위한 여러 가지 환경설정을 구성하고 PID Control 을 하기위한 명령을 내리기 위한 Panel 입니다. Graph panel 은 각 A/D 입력 및 D/A 출력 데이터를 모니터하기 위한 Panel 입니다.

본 프로그램에서 Control panel 과 Graph panel 의 두개의 화면으로 분리한 이유는 상황에 따라 그래프화면을 최대한 크게 하여 볼 수 있도록 하기 위함입니다. Graph panel 은 그 크기를 사용자가 마음대로 조정할 수 있습니다. Graph panel

의 크기에 따라 두개의 그래프도 함께 변하게 됩니다.

두 Panel 의 위치나 크기는 프로그램이 종료될 때 자동으로 저장되어 다음에 실행될 때 마지막에 실행되었던 상황에서의 각 Panel 의 위치나 크기가 복원됩니다.



[그림 2-33] 고속 PID Controls 기능의 화면 구성

2) 고속 PID Controls 기능의 옵션(Options)

고속 PID Controls 의 옵션항목은 Control panel 에서 Checkbox 로 이루어진 항목들로써 'Enable PID Control', 'Continuous plot'과 'Data logging'항목이 있습니다.

◈ Enable PID Control

이 항목은 PID Control 을 적용할 것인지를 결정합니다. 이 항목을 체크하지 않으면 'Start'버튼을 클릭하여도 시스템에 PID 제어를 적용하지 않습니다. 단지 A/D 채널들만 계측하여 화면에 표시해줍니다. 1 차계의 특성을 알아보기 위해서는 PID 제어를 수행하지 않고 시스템의 출력만을 모니터링해 볼 필요가

있습니다. 이 때에는 시스템의 입력은 D/A 채널 또는 외부 디바이스를 이용하여 직접 입력하여주어야 합니다.

◈ Continuous plot

이 항목은 그래프를 업데이트하는 방식을 결정합니다. 이 항목을 체크하면 그래프는 계속해서 업데이트가 됩니다. 이 항목을 체크하지 않으면 그래프의 시간축의 최대값까지 그래프가 그려지면 이후에는 그래프를 업데이트하지 않습니다.

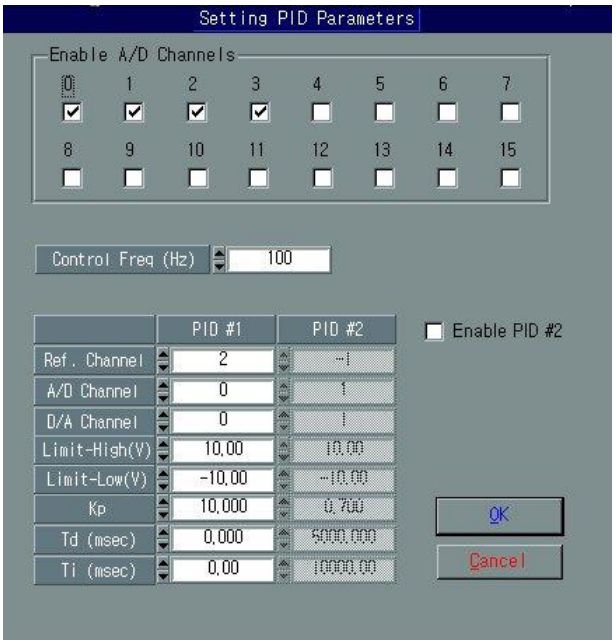
◈ Data logging

이 항목은 데이터의 저장여부를 결정합니다. 이 항목을 선택하면 'Start' 버튼이 눌린 후에 모든 A/D 데이터가 임시 바이너리 파일에 저장됩니다. 이렇게 Log 파일이 생성되면 사용자는 'Stop' 버튼이 눌린 후에 'Save' 버튼을 클릭하여 데이터를 텍스트파일 형식으로 저장할 수 있습니다. 단, Data logging을 하는 경우에는 사양이 낮은 컴퓨터 환경에서 속도의 저하요인이 될 수 있습니다.

이 항목을 선택하지 않으면 Log 파일이 생성되지 않으며, 따라서 텍스트 파일로의 데이터 저장도 되지 않습니다. 데이터를 저장할 필요가 있을 때만 이 옵션을 선택하는 것이 좋습니다.

3) 고속 PID Controls 기능의 파라미터(Parameters)

고속 PID Controls 기능에는 여러 가지 환경을 파라미터로써 지정할 수 있습니다. 파라미터 설정을 하기 위해서는 Control panel 에서 'Param' 버튼을 클릭하여야 합니다. 그러면 [그림 2-34]와 같은 화면이 나타납니다. [그림 2-34]화면에서 원하는 파라미터들을 설정하신 후에 'OK' 버튼을 클릭하시면 설정된 파라미터들이 적용됩니다.



[그림 2-34] 고속 PID Controls 파라미터 입력화면

◈ Enable A/D Channels

이 항목은 계측할 A/D 채널들을 선택합니다. PID 제어에 직접 사용되는 채널이 아니더라도 계측 및 모니터링이 가능합니다. 단, 제어 주파수를 3 KHz 이상으로 하는 경우에는 계측 채널 수가 다음과 같이 제한됩니다.

$$N < \frac{110000(Hz)}{2F_c} \qquad \text{(식 2-3)}$$

여기서, N : 선택 가능한 A/D 채널 수

F_c : 제어 주파수 (Hz)

예를 들면 제어 주파수를 5 KHz 로 하는 경우, $N < \frac{110000(Hz)}{2F_c}$ =11 입니다.

즉, 계측 가능한 최대 A/D 채널 수는 10 개가 됩니다.

'Enable A/D Channels' 파라미터 그룹은 PID Control 이 시작된 이후에는 변경할 수 없습니다.

◈ Control Freq

이 항목은 PID 제어 주파수를 결정합니다. 제어 주파수는 1Hz ~ 15KHz 의 범위에서 설정 가능합니다.

◈ PID 시스템 파라미터

PID 시스템 파라미터 그룹은 [그림 2-34]화면에서 하단에 표 형식으로 구성되어 있는 파라미터 항목들을 말합니다. 본 프로그램에서는 동시에 2 개의 시스템에 대하여 PID 제어가 가능합니다. 각 파라미터 항목들은 각 PID 시스템에 다르게 적용할 수 있습니다.

각 파라미터 항목에 대한 자세한 의미는 다음과 같습니다.

▷ **Enable PID #2 -** 이 항목은 2 개의 시스템에 대하여 PID 제어를 수행할 것인지를 결정합니다. 최대 2 개까지 동시에 PID 제어가 가능합니다.

▷ **Ref. Channel -** 이 항목은 레퍼런스(Reference) 입력 값의 소스를 결정합니다. 이 값을 -1 값으로 지정하면 [그림 2-33] 의 Control panel 에서 'Ref. (V)' 항목에 입력되는 값을 레퍼런스 입력 값으로 사용합니다. 이 값을 0 이상의 값으로 지정하면 해당 A/D 채널에서 계측되는 값을 레퍼런스 값으로 사용하게 됩니다. 단, 이때에는 해당 채널이 'Enable A/D Channel' 항목에서 선택되어진 채널이어야 합니다.

▷ **A/D Channel -** 시스템의 출력을 계측하여 Feedback 할 A/D 채널을 지정합니다. 이 채널의 값이 레퍼런스와 비교되어 에러(Error)값으로 계산됩니다. 단, 해당 채널이 'Enable A/D Channel' 항목에서 선택되어진 채널이어야 합니다.

▷ **D/A Channel -** 시스템의 입력으로 사용될 D/A 채널을 지정합니다. PID 제어 알고리즘인 (식 2-2)를 적용하여 계산된 조작량 $u(n)$ 값이 이 채널을 통하여 시스템에 입력되게 됩니다.

▷ **Limit-High -** D/A 채널을 통하여 출력될 최대값을 지정합니다. 이 값의 단위는 Volt 이며, 그 값이 10 이하이어야 합니다.

▷ **Limit-Low -** D/A 채널을 통하여 출력될 최소값을 지정합니다. 이 값의 단위는 Volt 이며, 그 값이 -10 이하이어야 합니다.

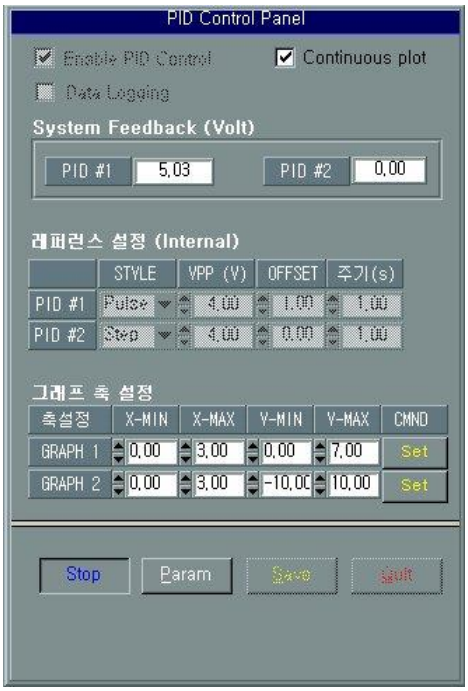
▷ **Kp -** 비례게인 값을 지정합니다(식 2-2 참조).

▷ **Td** - 미분시간을 지정합니다(식 2-2 참조). 이 값의 단위는 msec 입니다.

▷ **Ti** - 적분시간을 지정합니다(식 2-2 참조). 이 값의 단위는 msec 입니다.

4) 내부 레퍼런스 입력 설정

내부 레퍼런스 입력이란 PID 시스템이 추종해야할 레퍼런스를 프로그램 내부에서 지정하는 것을 말합니다. 이를 위해서는 앞의 'PID 시스템 파라미터' 중에서 'Ref. Channel' 항목이 -1 로 설정되어야 합니다.



[그림 2-35] PID Control Panel

내부 레퍼런스 입력을 사용하는 경우 프로그램은 Step, Pulse, Ramp, Sine 의 4 가지 형태의 파형을 레퍼런스 입력으로 지정할 수 있습니다.

[그림 2-35]와 같은 PID Control Panel 에서 '레퍼런스 설정(Internal)' 그룹에 포함된 각 항목들을 지정함으로써 레퍼런스 입력을 구성할 수 있습니다.

각 항목의 의미는 다음과 같습니다.

▷ **STYLE** - 파형의 종류를 지정합니다.

▷ **VPP** - Peak to Peak Voltage 값을 지정합니다. 이 값은 파형의 최대값과 최소값의 차에 해당합니다.

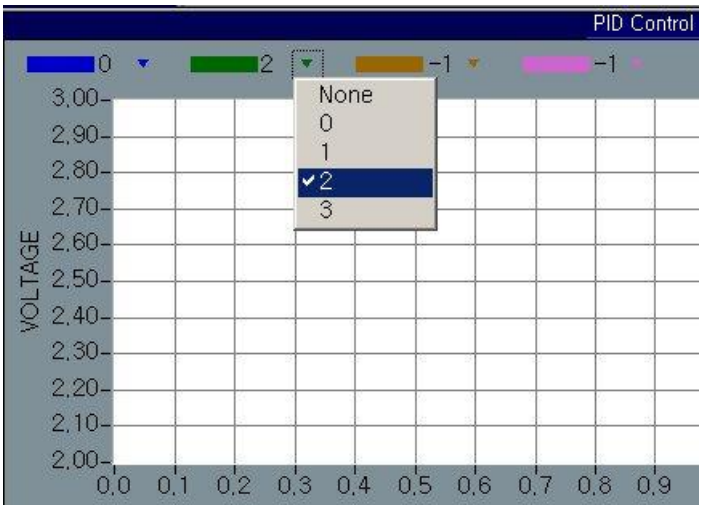
▷ **OFFSET** - 파형의 최소값의 Voltage 값을 지정합니다.

▷ **주기** - 파형의 주기를 Sec 단위로 지정합니다. 단, 레퍼런스 파형의 최소 주기값은 0.2 초이며 Ramp 나 Sine 파의 경우는 1 초 이상으로 하는 것이 좋습니다.

5) 고속 PID 제어의 실행 및 모니터링

고속 PID 제어 기능에 대한 각 파라미터와 옵션들을 적절히 설정한 후 Control panel 의 'Start' 버튼을 클릭하면 PID 제어가 시작됩니다. 'Start' 버튼은 토글버튼(Toggle button)으로써 눌리면 'Stop' 버튼으로 바뀌게 됩니다. 즉, 버튼의 글씨가 'Start' 로 되어있는 경우에는 이 버튼이 PID 제어를 시작하는 용도로 사용되며, 'Stop' 으로 되어있는 경우에는 이 버튼이 PID 제어를 종료하는 용도로 사용됩니다.

PID 제어가 시작되면 각 State 는 A/D 채널을 통하여 계측되어 두개의 그래프에 나누어 그릴 수가 있습니다. PID 제어에서 그래프는 [그림 2-33]화면에서와 같이 두개가 사용됩니다. 각 그래프에는 최대 4 개의 채널을 동시에 그릴 수 있으며 각 그래프에 그려질 A/D 채널은 [그림 2-36]과 같이 화면의 상단에서 선택할 수 있습니다.

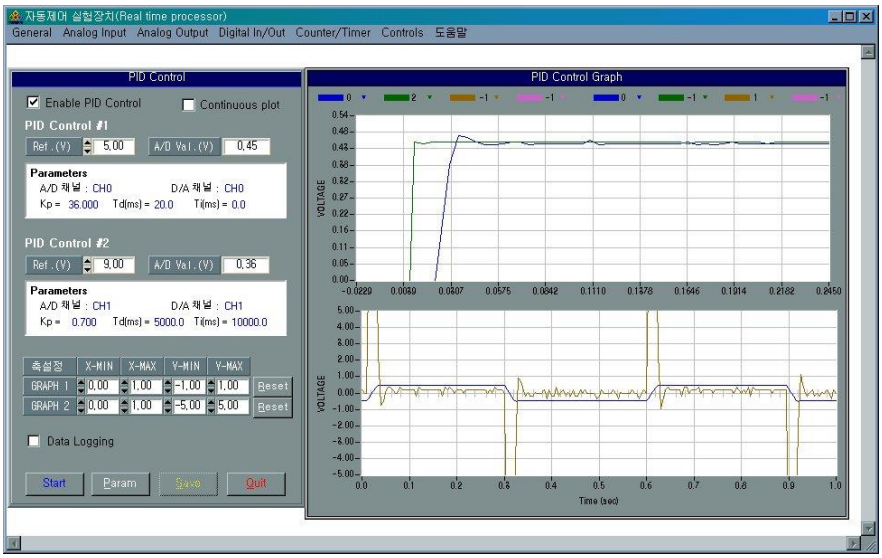


[그림 2-36] 그래프에 그려질 A/D 채널을 선택하는 화면

본 프로그램에서는 레퍼런스 입력 값과 D/A 출력 값을 따로 화면에 표시하지는 않습니다. D/A 출력 값을 모니터링하고자 하는 경우에는 A/D 채널 중 하나를 D/A 채널과 연결하여 계측하므로써 모니터링할 수 있습니다. 레퍼런스 값은 외부 입력인 경우에는 해당 A/D 채널을 그래프에 그리도록 선택하므로써 모니터링

할 수 있습니다.

[그림 2-37]은 실제 PID 제어를 실시하여 모니터링한 화면입니다. 이 경우에는 A/D CH0 을 시스템 출력 Feedback 용으로 사용하였으며, A/D CH1 을 시스템 입력으로 연결된 D/A 채널과 연결하여 D/A 출력을 모니터링 하도록 하였고, 레퍼런스 입력을 A/D CH2 로 지정하여 Function Generator 를 이용하여 레퍼런스 입력을 주는 예입니다. 위의 그래프에는 시스템 출력(AD CH0)과 레퍼런스 입력(AD CH2) 을 표시한 것이며, 아래 그래프에는 시스템 출력(AD CH0)과 계측된 시스템 입력(AD CH0, 실제로는 D/A 값)을 표시한 것입니다.



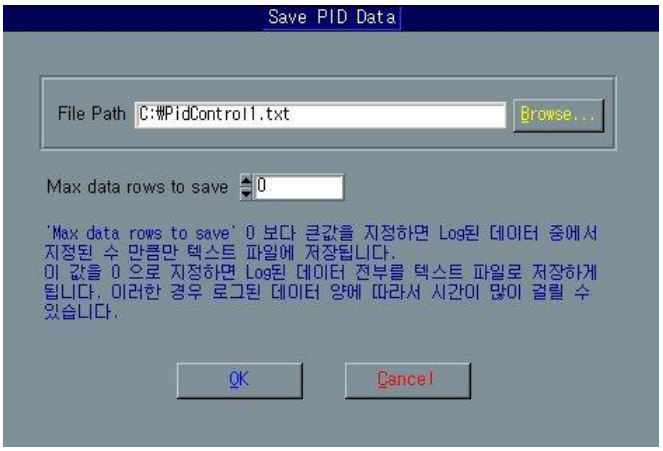
[그림 2-37] PID 제어의 각 STATE 를 모니터링한 화면

▷ **그래프 축 설정** - 그래프의 축 설정은 두 가지 방법으로 할 수 있습니다. 첫 번째는 Control panel 의 '축설정' 항목에서 X-Y 축의 범위를 지정한 후 오른쪽의 'Reset' 버튼을 클릭하면 지정된 값으로 그래프의 축이 설정됩니다. 두 번째는 마우스와 키보드를 이용하여 ZOOM IN/OU 과 SCROLL 을 하는 것입니다. 「CTRL + 마우스 왼쪽 클릭」을 하면 그래프가 ZOOM IN 되며, 「CTRL + 마우스 오른쪽 클릭」을 하면 그래프가 ZOOM OUT 되고, 「CTRL + SHIFT + 마우스 끌기」를 하면 그래프가 SCROLL 됩니다.

6) 고속 PID 제어 데이터 저장

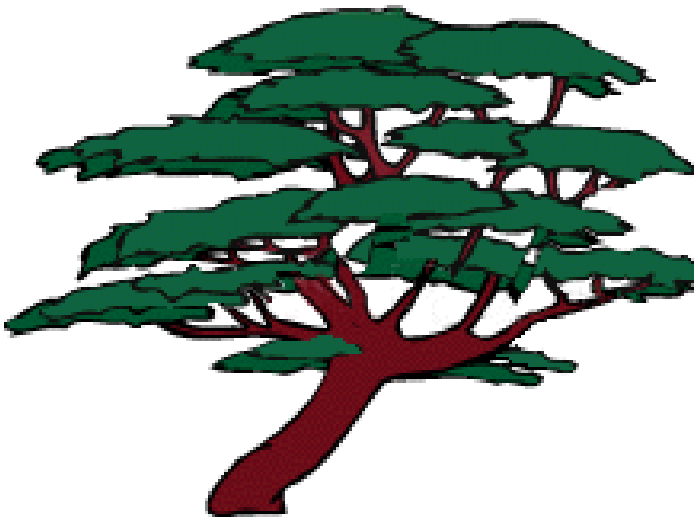
Control panel 에서 'Data logging' 옵션을 선택하면 PID 제어를 시작하면서 각 A/D 채널의 값이 임시 바이너리 파일에 저장됩니다. 'Stop' 버튼이 눌리어 PID 제어가 중지되면 'Save' 버튼이 클릭 가능하게 됩니다(PID 제어를 하는 중에는 'Save' 버튼은 dimmed 됩니다). Data logging 이 되었으면 'Save' 버튼을 클릭하여 텍스트 파일로 데이터를 저장할 수 있습니다. 'Save' 버튼을 누르면 [그림 2-38]과 같은 화면이 나타납니다. 'File Path' 항목에 원하는 파일명을 입력한 후 'OK' 버튼을 클릭하면 데이터가 해당 텍스트 파일에 저장이 됩니다. 단, 데이터의 저장은 'Data logging' 옵션을 선택한 후 PID 제어를 한 경우에만 가능합니다.

'Max data rows to save' 항목은 텍스트 파일에 저장할 데이터 수를 지정합니다. 이 값을 0 으로 지정하면 로깅(Logging)된 전체 데이터를 텍스트 파일로 저장하게 되며, 이 값을 양의 값으로 지정하면 지정된 데이터 수만큼만 텍스트 파일에 저장됩니다.



[그림 2-38] PID DATA 를 텍스트 파일로 저장할 때 나타나는 대화상자

CHAPTER 3
Windows C/C++ 라이브러리



CHAPTER 3. Windows C/C++ 라이브러리

본 장에서는 COMIDAS 디바이스의 윈도우용 C/C++ 라이브러리를 소개합니다. COMIDAS 디바이스의 윈도우용 C/C++ 라이브러리는 Comidas.sys 라는 디바이스 드라이버 프로그램과 ComiDll.dll 파일로 구성됩니다. Comidas.sys 는 COMIDAS 디바이스의 통합 드라이버이며 모든 종류의 COMIDAS 디바이스를 제어하는 프로그램입니다. ComiDll.dll 파일은 Comidas.sys 프로그램과 사용자 프로그램과의 통신을 담당해주는 DLL(Dynamic Link Library)프로그램으로써 이 프로그램을 통해 COMIDAS 제어기를 Access 할 수 있습니다. 이 두 파일들은 디바이스 설치 시에 윈도우의 적정 디렉토리에 설치되므로 사용자가 따로 설치할 필요는 없습니다.

실제로 사용자가 COMIDAS 의 윈도우용 C/C++ 라이브러리를 사용하여 프로그램하는 방법은 아주 간단합니다. 다음의 절차에 따라 필요한 파일들을 사용자의 프로젝트에 추가하고 일반 API 함수 사용하듯 필요한 함수를 호출하면 됩니다.

1. COMIDAS_ROOT₩Window₩C_Cpp₩Lib 폴더에 있는 Comidas.cpp 또는 Comidas.c 파일과 ComidasCommon.h, Comidas.h 파일을 사용자 소스 폴더에 복사합니다. 여기서 COMIDAS_ROOT 는 COMIDAS 파일들이 설치된 루트 디렉토리를 가리킵니다.
2. Comidas.cpp 또는 Comidas.c 파일과 Comidas.h 파일을 사용자 프로젝트에 추가합니다.
3. 사용하고자 하는 소스 파일에 #include “Comidas.h” 구문을 추가합니다.
4. Visual C++ 에서는 Comidas.cpp 파일의 맨 처음 부분에 #include “stdafx.h” 구문을 추가합니다. 단, Project 옵션에서 "not using precompiled header" 옵션을 선택한 경우에는 이 작업은 필요하지 않습니다.

3-1 라이브러리 및 디바이스 시작/종료 함수

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.

- BOOL COMI_LoadDll(void)
- void COMI_UnloadDll(void)
- HANDLE COMI_LoadDevice (COMIDAS_DEVID deviceID, ULONG instance)
- void COMI_UnloadDevice (HANDLE hDevice)

함수명	보드별 지원 여부										
	CP101	CP201	CP301	CP401	CP501	SD10x	SD201	SD301	SD4xx	SD501	SD502
COMI_LoadDll	V	V	V	V	V	V	V	V	V	V	V
COMI_UnloadDll	V	V	V	V	V	V	V	V	V	V	V
COMI_LoadDevice	V	V	V	V	V	V	V	V	V	V	V
COMI_UnloadDevice	V	V	V	V	V	V	V	V	V	V	V

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

◈ BOOL COMI_LoadDll (void)

이 함수는 COMIDAS DLL(Dynamic Link Library)을 로드(load)합니다.

- *Return* : 1 => 성공
  0 => 실패
- *Remarks* :
  1. 이 함수는 그 어떤 다른 COMIDAS Library 함수들 보다도 먼저 수행되어져야 합니다. 보통 프로그램 시작 시에 수행하면 됩니다.
  2. 여러 개의 디바이스를 제어하더라도 이 함수는 한번만 실행 되어야 합니다.

☞ **지원 디바이스 :** All devices

◈ void COMI_UnloadDll (void)

이 함수는 COMIDAS DLL(Dynamic Link Library)을 언로드(unload)합니다.

- *Remarks* : 이 함수는 보통 프로그램 종료 시에 수행하면 됩니다.

☞ **지원 디바이스 :** All devices

◈ HANDLE COMI_LoadDevice (COMIDAS_DEVID deviceID, ULONG instance)

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

- *deviceID* : 각 디바이스의 고유한 아이디값입니다. 이 값은 각 디바이스 명칭과 동일하게 적어주면 됩니다. 예를 들어 COMI-CP101 Multi-function board 의 경우 COMI_CP101 을 적어주면 됩니다. 각 디바이스 아이디는 Comidas.h 헤더파일의 앞부분에 정의되어 있으므로 Comidas.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 1이 됩니다.

- *Return* : 이 함수는 디바이스 핸들을 반환합니다. 이 값은 디바이스를 제어하는 각 함수의 첫번째 파라미터로 사용됩니다. 만일 이 값이 INVALID_HANDLE_VALUE 이면 디바이스 로딩이 실패한 것입니다.
- *Remarks* : 이 함수는 제어하고자 하는 디바이스 수만큼 수행되어야 합니다.

☞ **지원 디바이스 :** All devices

◈ void COMI_UnloadDevice (HANDLE hDevice)

이 함수는 하나의 COMIDAS 디바이스를 언로드(unload)한다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값입니다.

☞ **지원 디바이스 :** All devices

[리스트 3-1] 라이브러리 및 디바이스 시작/종료 함수 사용 예 1

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Library & Deivce loading/unloading
/* - Contents: 이 프로그램은 COMI-CP101 보드 하나를 사용할 때 시작과 종료 시 필요한
/*       절차를 보여주는 예제 입니다.
/*************************************************************************/

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

void main (void)
{
  HANDLE hDevice;

  //------ 프로그램 시작 시 수행해야 할 루틴 ------//
  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }
  //--------------------------------------//

  // 필요한 디바이스 제어 루틴을 구현한다. //
  ……
  ……

  //------ 프로그램 종료 시 수행해야 할 루틴 ------//
  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
  //--------------------------------------//
}
```

[리스트 3-2] 라이브러리 및 디바이스 시작/종료 함수 사용 예 2

```
/************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Library & Deivce loading/unloading
/* - Contents: 이프로그램은 COMI-CP101 보드 2 개와 COMI-CP301 보드 하나를 사용할 때
/*     시작과 종료시 필요한 절차를 보여주는 예제 입니다.
/************************************************************************/

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

void main (void)
{
  HANDLE hDevice[3];

  //------ 프로그램 시작시 수행해야할 루틴 ------//
  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }

  hDevice[0] = COMI_LoadDevice (COMI_CP101, 0);
  hDevice[1] = COMI_LoadDevice (COMI_CP101, 1);
  hDevice[2] = COMI_LoadDevice (COMI_CP301, 0);
  // 여기에서 Loading 에러처리를 해야하나 생략한다. //
  //--------------------------------------//

  // 필요한 디바이스 제어 루틴을 구현한다. //
  ……
  ……

  //------ 프로그램 종료시 수행해야할 루틴 ------//
  COMI_UnloadDevice(hDevice[0]);
  COMI_UnloadDevice(hDevice[1]);
  COMI_UnloadDevice(hDevice[2]);
  COMI_UnloadDll();
  //--------------------------------------//

}
```

3-2 아날로그 입력

이 장에서는 아날로그 입력(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. 이에 관련된 함수들은 단원 3-2-2 에서 소개됩니다.

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. 이에 관해서는 단원 3-2-3 에서 자세히 소개됩니다.

3-2-1 아날로그 입력 공통 함수

이 단원에서는 Single point A/D 방식과 A/D Scan 방식 모두에서 사용될 수 있는 함수들을 소개합니다. 이에 관련된 함수들의 리스트는 다음과 같습니다.

■ BOOL COMI_AD_SetRange (HANDLE hDevice, int ch, float vmin, float vmax)

■ float COMI_DigitToVolt (short digit, float vmin, float vmax)

■ float COMI_Digit16ToVolt (short digit, float vmin, float vmax)

함수명	보드별 지원 여부										
	CP101	CP201	CP301	CP401	CP501	SD10x	SD20x	SD301	SD4xx	SD501	SD502
COMI_AD_SetRange	V	V				V	V				
COMI_DigitToVolt	V	V				V					
COMI_Digit16ToVolt							V				

[표 3-2] 아날로그 입력 공통 함수 리스트 및 각 보드별 지원 여부

◈ BOOL COMI_AD_SetRange (HANDLE hDevice, int ch, float vmin, float vmax)

이 함수는 각 A/D 채널의 입력 범위를 정해줍니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_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.

  CP 시리즈 보드 ~ -1, -2, -5, -10

  SD 시리즈 보드 ~ 0, -1, -2, -5, -10
- *vmax* : A/D 범위의 최대값을 지정합니다. 유효한 vmax 값은 1, 2, 5, 10 입니다.
- *Return* : 1 => 성공

  0 => 실패

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ float COMI_DigitToVolt (short digit, float vmin, float vmax)

이 함수는 12 Bit resolution 의 A/D 디바이스(COMI-CP101, COMI-CP201, COMI-SD101)의 정수형 A/D 변환값(0 ~ 4095)을 voltage 값으로 환산해주는 함수입니다.

- *digit* : 변환하고자 하는 digit 값을 지정합니다. 이 값은 0~4095 사이의 값이어야 합니다.
- *vmin* : A/D 범위의 최소값을 지정합니다.
- *vmax* : A/D 범위의 최대값을 지정합니다.
- *Return* : digit 값을 voltage 값으로 환산한 값.
- *Remarks* : digit 값을 voltage 값으로 환산하는 것은 다음과 같은 식에 의해서 이루어집니다.

Voltage = digit*(vmax - vmin)/4095 + vmin

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104

◈ float COMI_Digit16ToVolt (short digit, float vmin, float vmax)

이 함수는 16 Bit resolution 의 A/D 디바이스(COMI-SD201)의 정수형 A/D 변환값(-32768 ~ 32767)을 voltage 값으로 환산해주는 함수입니다.

- *digit* : 변환하고자 하는 -32768 ~ 32767 사이의 digit 값.
- *vmin* : A/D 범위의 최소값을 지정합니다.
- *vmax* : A/D 범위의 최대값을 지정합니다.
- *Return* : digit 값을 voltage 값으로 환산한 값.
- *Remarks* : digit 값을 voltage 값으로 환산하는 것은 다음과 같은 식에 의해서 이루어집니다.

Voltage = (digit+32768)*(vmax - vmin)/65535 + vmin

☞ **지원 디바이스 :** COMI-SD201

3-2-2 Single point A/D

이 단원에서는 Single point A/D 를 수행하는 함수들을 소개합니다.

◈ short COMI_AD_GetDigit (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : A/D 를 수행할 채널 번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 정수형의 A/D 결과값. 12 Bit resolution 디바이스(COMI-CP101, COMI-CP201, COMI-SD101)는 0 ~ 4095 사이의 값을 가지며, 16 Bit resolution 디바이스(COMI-SD201)는 -32768 ~ 32767 사이의 값을 갖는다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ float COMI_AD_GetVolt (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : A/D 를 수행할 채널 번호. 채널 번호는 0 부터 시작합니다.
- *Return* : A/D 결과값 (voltage)

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

[리스트 3-3] Single point A/D 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/* - Subject : Single point A/D
/* - Contents: 이프로그램은 COMI_AD_GetVolt(...)함수를 사용하여 Single point
/*     A/D 를 수행하는 프로그램입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*     사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은
/*     디바이스 ID 로 바꾸어야 합니다.
/*************************************************************************/
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "Comidas.h"
#define CHAN  0
#define VMIN  -10
#define VMAX  10

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

  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }
  COMI_AD_SetRange (hDevice, CHAN, VMIN, VMAX);

  while (!_kbhit()){
    ad_volt = COMI_AD_GetVolt (hDevice, CHAN);
    printf("%6.2f\n", ad_volt); // 결과를 화면에 보여준다.
    Sleep(500); // 0.5 초 delay
  }

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

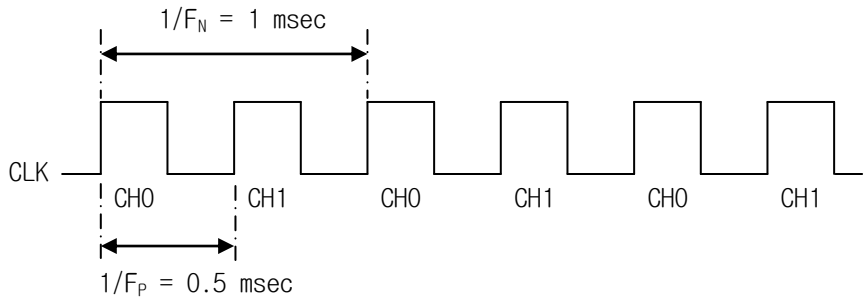
3-2-3 Unlimited A/D Scan

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/D 를 할 때 아주 유용하게 사용될 수 있습니다. 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. CP 시리즈 보드의 경우에는 Scan rate 와 Sampling rate 가 동일 타이머에 의해 제어되며, Sampling rate = (Scan rate) * (채널수) 의 관계를 가지게 됩니다. SD 나 LX 시리즈 보드의 경우는 Scan rate 와 Sampling rate 가 각각 다른 타이머에 의해 제어됩니다. 이러한 경우, 기본적으로 Sampling rate 는 각 디바이스가 지원하는 최대 Sampling rate 로 설정되어 최대한 동시 샘플링을 할 수 있도록 설정됩니다. 그러나 본 라이브러리에서는 사용자에 따라 이 값을 바꿀 수 있도록 함수를 지원하고 있습니다. 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.

▷ CP 시리즈 보드

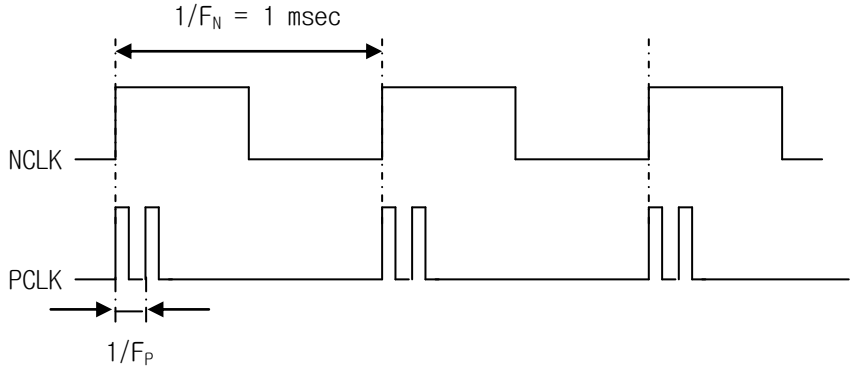


CLK : 스캔 및 샘플링 타이머 신호

F_N : Scan frequency

F_P : Sampling frequency

▷ SD 또는 LX 시리즈 보드



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 디바이스에 따라 또는 사용자의 정의에 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.

Unlimited A/D Scan 에 관련된 함수들의 리스트는 다음과 같다.

1) Unlimited Scan 기본 함수

■ long COMI_US_Start (HANDLE hDevice, int numCh, int *chanList, UINT scanFreq, UINT bufSize, int trsMethod)
■ BOOL COMI_US_Stop(HANDLE hDevice, BOOL bReleaseBuf)
■ BOOL COMI_US_ReleaseBuf(HANDLE hDevice)
■ ULONG COMI_US_CurCount (HANDLE hDevice)

2) Scan 버퍼 직접 access 함수

■ short *COMI_US_GetBufPtr(HANDLE hDevice)
■ UINT COMI_US_SBPos(HANDLE hDevice, int chOrder, ULONG scanCount)

3) Scan 버퍼 간접 access 함수

■ float COMI_US_RetrvOne (HANDLE hDevice, int chOrder, ULONG scanCount)
■ ULONG COMI_US_RetrvChannel (HANDLE hDevice, int chOrder, ULONG startCount, UINT maxNumData, void *pDestBuf, TVarType VarType)
■ UINT COMI_US_RetrvBlock (HANDLE hDevice, UINT startCount, UINT maxNumScan, void *pDestBuf, TVarType VarType)

4) 자동 파일 저장 함수

■ BOOL COMI_US_FileSaveFirst (HANDLE hDevice, char *szFilePath)
■ ULONG COMI_US_FileSaveNext(HANDLE hDevice)
■ BOOL COMI_US_FileSaveStop()
■ void COMI_US_FileConvert(char *szBinFilePath, char *szTextFilePath)

■ float COMI_US_CheckFileConvert(void)

■ void COMI_US_CancelFileConvert(void)

함수명	보드별 지원 여부										
	CP101	CP201	CP301	CP401	CP501	SD10x	SD20x	SD301	SD4xx	SD501	SD502
COMI_US_Start	v	v				v	v				
COMI_US_Stop	v	v				v	v				
COMI_US_ReleaseBuf	v	v				v	v				
COMI_US_CurCount	v	v				v	v				
COMI_US_GetBufPtr	v	v				v	v				
COMI_US_SBPos	v	v				v	v				
COMI_US_RetrvOne	v	v				v	v				
COMI_US_RetrvChannel	v	v				v	v				
COMI_US_RetrvBlock	v	v				v	v				
COMI_US_FileSaveFirst	v	v				v	v				
COMI_US_FileSaveNext	v	v				v	v				
COMI_US_FileSaveStop	v	v				v	v				
COMI_US_FileConvert	v	v				v	v				
COMI_US_CheckFileConvert	v	v				v	v				
COMI_US_CancelFileConvert	v	v				v	v				

[표 3-3] Unlimited A/D Scan 함수 리스트 및 각 보드별 지원 여부

1) A/D Scan 기본 함수

◈ **long COMI_US_Start (HANDLE hDevice, int numCh, int *chanList, UINT scanFreq, UINT bufSize, int trsMethod)**

이 함수는 Unlimited scan 기능을 시작합니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *numCh* : scan 할 A/D 채널의 수.
- *chanList* : A/D scan 을 수행할 채널 리스트를 담고 있는 배열 또는 포인터.
- *scanFreq* : A/D scan frequency 를 지정합니다. 이 값은 SCAN 과 SCAN 사이의 시간차를 결정합니다. Scan frequency 에 대한 자세한 내용은 본 매뉴얼의 69 페이지를 참조하십시오. 단, 이 값을 0 으로 하면 External Trigger 모드로 동작합니다.
- *bufSize* : 스캔 데이터를 저장할 환형버퍼의 크기를 지정합니다. 이 값의 단위는 바이트가 아니고 SCAN 횟수입니다. 예를 들어, 채널이 2 채널일 때 bufSize 값을 10240 으로 지정하면 실제로는 2 * 10240 *sizeof(short) = 40960 바이트 크기의 버퍼가 할당됩니다. 이 값은 1024 의 배수로 지정하는 것이 좋습니다.
- *trsMethod* : A/D 디바이스에서 스캔버퍼로 데이터를 전송하는 방식을 지정합니다. 이 값은 다음의 두 값 중의 하나이어야 합니다. 단, CP 시리즈 보드에서는 TRS_SINGLE 값만 사용할 수 있습니다.
  - ▷ TRS_SINGLE => CP 시리즈 보드의 경우에는 매 채널 A/D 변환이 완료될 때마다 그리고 SD 시리즈 보드의 경우에는 1 회의 SCAN 이 완료될 때마다 인터럽트를 발생시켜 데이터를 전송합니다. 인터럽트의 한계에 따라 Scan frequency 가 30 KHz 이상이 되면 이 방식이 적절히 작동하지 않을 수 있습니다.
  - ▷ 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 단

위로 사용자 버퍼에 전송합니다. 이 방식은 고속 A/D scan 시에 적합합니다. 단, 이 방식은 CP 시리즈 보드에서는 사용할 수 없다.

- *Return* : 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.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ long COMI_US_ChangeScanFreq (HANDLE hDevice, UINT scanFreq)

이 함수는 Unlimited scan 을 수행하는 중에 Scan 주파수만을 변경하고자할 때 사용합니다. 이 함수를 수행한 후에 COMI_US_ResetCount()함수를 수행해 주는 것이 좋습니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *scanFreq* : A/D scan frequency 를 지정합니다. 이 값은 SCAN 과 SCAN 사이의 시간차를 결정합니다. Scan frequency 에 대한 자세한 내용은 본 매뉴얼의 69 페이지를 참조하십시오.
- *Return* : 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.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ void COMI_US_ResetCount (HANDLE hDevice)

이 함수는 Scan Count 를 0 으로 초기화합니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ BOOL COMI_US_Stop(HANDLE hDevice, BOOL bReleaseBuf)

이 함수는 Unlimited scan 을 종료합니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *bReleaseBuf* : COMI_US_Start()가 수행될 때 할당되었던 스캔 버퍼를 메모리 해제시킬것인지를 결정합니다. 만일 이 값을 FALSE 로 지정하면 후에 반드시 COMI_US_ReleaseBuf() 사용하여 버퍼를 해제하여야 합니다. 이 값을 TRUE 로 지정하면 COMI_US_ReleaseBuf() 함수를 수행할 필요가 없습니다.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ BOOL COMI_US_ReleaseBuf (HANDLE hDevice)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해

얻어진 값이어야 합니다.

- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ ULONG COMI_US_CurCount (HANDLE hDevice)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 현재까지 수행된 총 SCAN 횟수

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

2) Scan 버퍼 직접 Access 함수

Unlimited Scan 이 시작되면 디바이스는 스캔 버퍼에 스캔된 데이터를 차례대로 저장합니다. 사용자는 이 버퍼에서 데이터를 취하여 원하는 처리를 하게 되는데, 스캔 버퍼에서 데이터를 취하는 방법은 직접 Access 방법과 간접 Access 방법이 있습니다. 이 단원에서 소개할 함수들은 스캔 버퍼를 직접 Access 하여 데이터를 취하는데 필요한 함수들입니다.

◈ short* COMI_US_GetBufPtr (HANDLE hDevice)

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

- *hDevice* : 디바이스 핸들값이다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야한다.
- *Return* : 스캔 버퍼에 대한 포인터

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ UINT COMI_US_SBPos (HANDLE hDevice, int chOrder, ULONG scanCount)

이 함수는 원하는 스캔 데이터가 있는 스캔 버퍼의 인덱스(Index)를 계산해 줍니다. 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, COMI_US_SBPos()는 이를 해주는 함수입니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_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.
- *scanCount* : 원하는 데이터의 Scan count.
- *Return* : chOrder 와 scanCount 에 의해 지정된 데이터의 스캔 버퍼상의 인

덱스.

- *Remarks* :

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];
}
```

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

[리스트 3-4] Unlimited Scan 사용예 1 : 스캔 버퍼 직접 Access 방법을 통하여 데이터를 취하는 방법

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Unlimited Scan
/* - Contents: 이프로그램은 A/D CH0 와 CH4 의 두채널을 Unlimited scan 을 이용하여 A/D
/*     convert 를 하고 그 결과를 파일로 저장하는 예제입니다.  이 프로그램에서는 scan 버퍼를
/*     직접  handling 하는 방법을 보여주고  있습니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를 사
/*      용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*************************************************************************/

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

#define NUM_CH  2  /* Number of channels */
#define S_FREQ  1000 /* Scan freq. -> 1000 Hz */
#define MSB 40960  /* Max scans/buffer => 이 값은 scan 버퍼의 크기를 결정하는데  */
                   /* 1024 의 배수로 지정하는것이 좋다.  */

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

  /* Load DLL */
  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }
  /* Load Device */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
```

```
  }

  /* start unlimited scan */
  long act_freq = COMI_US_Start (hDevice, NUM_CH, ch_list, S_FREQ, MSB,
TRS_SINGLE)
  if(act_freq < 0){
    /* Error 처리 */
    COMI_ErrorString (COMI_LastError());
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    exit(0);
  }

  /* Create a file to save data */
  fp = fopen ("c:\\ComiUscan.txt", "w");
  fprintf (fp,"  CH0   CH4\n");
  printf("Unlimited scan has started !!\nPress any key to quit");

  pScanBuf = COMI_US_GetBufPtr(hDevice); // Get scan buffer pointer
  prv_cnt = 0; // initialize count variable

  while(!kbhit())
  {
    cur_cnt = COMI_US_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++)
    {
      /* COMI_US_SBPos (..) 함수를 이용하여 각 채널의 버퍼상 인덱스를 얻는다. */
      /* 이때 두번째 파라미터는 채널번호가 아니라 채널 리스트상의 순서이다.  */
      /* 즉, CH0 는 0 이고, CH4 는 1 이 된다.                          */
      idx1 = COMI_US_SBPos (hDevice, 0, c);
      idx2 = COMI_US_SBPos (hDevice, 1, c);

      /* File 에 데이터를 저장한다. 아래의 코드는 Voltage 값으로 저장하는 */
      /* 것이 아니고  정수값으로 저장된다. 이를 Voltage 값으로  환산해서   */
      /* 저장하려면 아래의 주석처리된 코드를 이용하면된다 .                */
      /*(단, COMI-SD201 보드는 COMI_Digit16ToVolt()를                  */
      /*  이용해야 한다.)                                              */
      // float ch0_volt, ch4_volt;
      // ch0_volt = COMI_DigitToVolt (pScanBuf[idx1], -10, 10);
      // ch4_volt = COMI_DigitToVolt (pScanBuf[idx2], -10, 10);
      // fprintf (fp, "%6.2f %6.2f\n", ch0_volt, ch4_volt);

      fprintf (fp, "%6d %6d\n", pScanBuf[idx1], pScanBuf[idx2]);
    }
```

```
      prv_cnt = cur_cnt;
    }

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

    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
}
```

3) Scan 버퍼 간접 Access 함수

이 단원에서는 스캔 버퍼를 직접 Handling 하지 않고 스캔 데이터를 취하는 함수들을 소개합니다. 버퍼를 직접 Access 하는 방법으로 데이터를 취하려면 버퍼 인덱스를 계산하고, 정수 값을 Voltage 값으로 환산하여야 하므로 약간 번거로울 수 있습니다. 따라서 이 단원에서 소개되는 함수들을 이용하면 보다 편하게 스캔 데이터를 취할 수 있습니다.

◈ **float COMI_US_RetrvOne (HANDLE hDevice, int chOrder, ULONG scanCount)**

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

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *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* : chOrder 와 scanCount 에 의해 지정된 데이터를 Voltage 형식으로 반환합니다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ **ULONG COMI_US_RetrvChannel (HANDLE hDevice, int chOrder, ULONG startCount, int maxNumData, void *pDestBuf, TVarType VarType)**

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.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *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.
  - ▷ **VT_SHORT** => pDestBuf 가 short 형 포인터임을 의미하며, 데이터는 Voltage 로 환산되기 이전의 정수형값으로 전달됩니다.
  - ▷ **VT_FLOAT** => pDestBuf 가 float 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.
  - ▷ **VT_DOUBLE** => pDestBuf 가 double 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.
- *Return* : 실제 전달된 데이터 수. 만일 startCount 이후에 현재까지 스캔된 데이터 수가 maxNumData 에서 지정한 수 보다 작으면, 현재 스캔된 데이터까지만 전달하게됩니다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ ULONG COMI_US_RetrvBlock (HANDLE hDevice, UINT startCount, UINT maxNumScan, void *pDestBuf, TVarType VarType)

이 함수는 모든 A/D Scan 채널에 대한 데이터 블럭을 전달합니다. 데이터 블록은 각 채널별로 사용자가 지정한 startCount 에서부터 maxNumScan 에서 지정한 수만큼이 됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의

해 얻어진 값이어야 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 전달받을 버퍼 포인터를 지정합니다. 이 버퍼는 short 형이나 float 형, 또는 double 형이어야 하며, VarType 파라미터에서 지정한 데이터형과 일치해야 합니다. 또한 이 버퍼의 크기는 maxNumScan 에서 지정한 값보다 크거나 같아야 합니다.
- *VarType* : pDestBuf 의 데이터 형을 지정한다. 이 값은 다음의 세 값 중 하나이어야 합니다.
  - ▷ **VT_SHORT** => pDestBuf 가 short 형 포인터임을 의미하며, 데이터는 Voltage 로 환산되기 이전의 정수형값으로 전달됩니다.
  - ▷ **VT_FLOAT** => pDestBuf 가 float 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.
  - ▷ **VT_DOUBLE** => pDestBuf 가 double 형 포인터임을 의미합니다. 데이터는 Voltage 값으로 전달됩니다.
- *Return* : 실제 전달된 데이터 블록의 크기(스캔 횟수).

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

[리스트 3-5] Unlimited Scan 사용예 2 : COMI_US_RetrvChannel () 함수를 사용하여 데이터를 취하는 예제

```
/*********************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Unlimited Scan
/* - Contents: 이프로그램은 A/D CH0 와 CH4 의 두채널을 Unlimited scan 을 이용하여
/*      A/D convert 를 하고 그 중 CH0 의 데이터를 파일에 저장하는 예제이다. 이 함수는 스캔
/*     버퍼를 직접 Access 하지 않고, COMI_US_RetrvChannel () 함수를 사용하여 데이터를
/*     취한다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는  경우에는 COMI_LoadDevice(...) 함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*********************************************************************/


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

#define NUM_CH  2 /* Number of channels */
#define S_FREQ  1000 /* Scan freq. -> 1000 Hz */
#define MSB   40960/* Max scans/buffer => 이 값은 scan 버퍼의 크기를 결정하는데 */
                  /* 1024 의 배수로 지정하는것이 좋다.                          */
#define CH_BUF_SIZE  (S_FREQ)

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(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }
  /* Load Device */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
```

```
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

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

  /* start unlimited scan */
  long act_freq = COMI_US_Start (hDevice, NUM_CH, ch_list, S_FREQ, MSB,
TRS_SINGLE)
  if(act_freq < 0){
    /* Error 처리 */
    COMI_ErrorString (COMI_LastError());
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    exit(0);
  }

  /* Create a file to save data */
  fp = fopen ("c:\\ComiUscan.txt", "w");
  fprintf (fp,"  CH0\n");
  printf("Unlimited scan has started !!\nPress any key to quit");
  // Allocate a buffer to retrieve scan data //
  if((pChanBuf = (float *)malloc(sizeof(float)*CH_BUF_SIZE)) == NULL){
    printf("Memory allocation failed !");
    COMI_US_Stop (hDevice, TRUE);
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    exit(0);
  }
  prv_cnt = 0;
  while(!kbhit())
  {
    // CH0 의 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 = COMI_US_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("%d\n", prv_cnt);
    Sleep(100);
  }

  free(pChanBuf); // Free channel buffer
```

```
    COMI_US_Stop (hDevice, TRUE);
    fclose(fp);

    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
}
```

[리스트 3-6] Unlimited Scan 사용예 3 : COMI_US_RetrvBlock () 함수를 사용하여 데이터를 취하는 예제

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Unlimited Scan
/* - Contents: 이프로그램은 A/D CH0 와 CH4 의 두채널을 Unlimited scan 을 이용하여
/*     A/D convert 를 하고  데이터를 파일에 저장하는 예제이다. 이 함수는 스캔 버퍼를 직접
/*     Access 하지 않고, COMI_US_RetrvBlock () 함수를 사용하여 데이터를 취한다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*************************************************************************/


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

#define NUM_CH  2 /* Number of channels */
#define S_FREQ  1000 /* Scan freq. -> 1000 Hz */
#define MSB   40960/* Max scans/buffer => 이 값은 scan 버퍼의 크기를 결정하는데 */
                   /* 1024 의 배수로 지정하는것이 좋다.                    */
#define CH_BUF_SIZE  (S_FREQ * NUM_CH)

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(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }
  /* Load Device */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
```

```
    exit(0);
  }

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

  /* start unlimited scan */
  long act_freq = COMI_US_Start (hDevice, NUM_CH, ch_list, S_FREQ, MSB,
TRS_SINGLE)
  if(act_freq < 0){
    /* Error 처리 */
    COMI_ErrorString (COMI_LastError());
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    exit(0);
  }

  /* Create a file to save data */
  fp = fopen ("c:\\ComiUscan.txt", "w");
  fprintf (fp,"  CH0   CH4\n");
  printf("Unlimited scan has started !!\nPress any key to quit");
  // Allocate a buffer to retrieve scan data //
  if((pChanBuf = (float *)malloc(sizeof(float)*CH_BUF_SIZE)) == NULL){
    printf("Memory allocation failed !");
    COMI_US_Stop (hDevice, TRUE);
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    exit(0);
  }
  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 = COMI_US_RetrvBlock (hDevice, prv_cnt+1, CH_BUF_SIZE/NUM_CH,
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("%d\n", prv_cnt);
    Sleep(100);
  }

  free(pChanBuf); // Free channel buffer
  COMI_US_Stop (hDevice, TRUE);
```

```
    fclose(fp);

    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
}
```

4) 자동 파일 저장 함수

이 단원에서는 스캔 데이터를 자동으로 파일에 저장하는 함수들을 소개합니다. 이 기능을 이용하면 사용자는 쉽게 데이터를 바이너리 파일이나 텍스트 파일로 저장할 수 있습니다. 이 기능 사용 시 스캔 데이터는 우선 바이너리 파일로 저장됩니다. 사용자는 이 바이너리 파일을 읽어서 필요한 처리를 하거나 COMI_US_FileConvert() 함수를 이용해서 텍스트 파일로 변환할 수 있습니다. 이 기능에 의해 생성되는 바이너리 파일은 처음 부분에 TScanFileHead 구조체 형식의 헤더 정보가 위치하며 이어서 스캔 데이터가 차례로 저장됩니다. TScanFileHead 구조체는 Comidas.h 헤더 파일에서 정의 되어 있습니다. 바이너리 파일을 직접 읽는 방법은 [리스트 3-8]의 예제를 참조하면 됩니다.

◈ **BOOL COMI_US_FileSaveFirst (HANDLE hDevice, char *szFilePath, BOOL bIsFromStart)**

이 함수는 자동 파일 저장 기능을 시작합니다. 이 함수는 자동 저장을 준비시켜주는 함수이며, 실제로 스캔 데이터를 저장하지는 않습니다. 실제 데이터의 저장은 COMI_US_FileSaveNext() 함수가 호출될 때 이루어집니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *szFilepath* : 데이터를 저장할 바이너리 파일의 Path를 지정합니다.
- *bIsFromStart* : 이 값이 1 이면 Scan 이 시작된 처음 데이터부터 저장하게 되며, 0 이면 File save 시작한 순간의 데이터부터 저장합니다.
- *Return* : 1 => 성공
  0 => 실패
- *주의* : 이 함수는 반드시 COMI_US_Start(..)함수가 실행된 이후에 수행되어야 합니다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ ULONG COMI_US_FileSaveNext (HANDLE hDevice)

이 함수는 바이너리 파일에 데이터 블록을 저장한다. 데이터 블록은 마지막 저장되었던 Scan count 의 다음부터 현재 이루어진 스캔 데이터까지가 됩니다. 단, 이함수는 COMI_US_FilseSaveFirst() 함수가 수행된 이후에 사용되어져야 합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 파일에 저장된 데이터의 스캔 횟수

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD201, COMI-LX101, COMI-LX201

◈ BOOL COMI_US_FileSaveStop(HANDLE hDevice)

이 함수는 바이너리 파일에 데이터를 저장하는 작업을 종료합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ void COMI_US_FileConvert(char *szBinFilePath, char *szTextFilePath, ULONG nMaxDataRow)

자동 파일 기능에 의해 생성된 바이너리 파일을 텍스트 파일로 변환하여 줍니다. 이 함수를 호출하면 COMIDAS 라이브러리 내부에서 하나의 쓰레드(Thread)가 생성되고 그 쓰레드 내부에서 파일 변환 작업이 이루어집니다. 이는 파일의 크기가 크면 파일 변환 작업하는데 시간이 오래 걸릴 수 있으므로 쓰레드를 쓰지 않을 경우에는 장시간동안 다른 작업을 수행할 수 없기 때문입니다.

- *szBinFilePath* : 바이너리 파일의 Path
- *szTextFilePath* : 텍스트 파일의 Path
- *nMaxDataRow* : 이 값이 0 이면 Binary file 에서 저장된 모든 데이터를 텍스트 파일로 변환하고, 양수 값이면 이 값이 지정한 Scan 횟수에 해당하는 데이터만 텍스트 파일로 저장됩니다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ float COMI_US_CheckFileConvert(void)

파일 변환 진행율을 % 값으로 반환합니다.

- *Return* : 파일 변환 작업 진행율(%).

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ void COMI_US_CancelFileConvert(void)

파일 변환 작업을 취소합니다. 그러나 이 함수가 호출될 때까지 이루어진 변환 결과는 텍스트 파일에 그대로 남게 됩니다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

[리스트 3-7] Unlimited Scan 사용예 4 : 자동 파일 저장 기능을 이용하여 스캔 데이터를 텍스트 파일로 저장하는 예제

```
/*********************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Unlimited Scan
/* - Contents: 이프로그램은 A/D CH0 와 CH4 의 두채널을 Unlimited scan 을
/*   이용하여 A/D convert 를 하고 그 결과를 COMI_US_FileSaveFirst(...),
/*   COMI_US_FileSaveNext(...), COMI_US_FileSaveStop(...) 함수들을 이용
/*   하여 파일에 저장하는 예제입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는 경우에는 COMI_LoadDevice(...) 함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*********************************************************************/


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

#define NUM_CH  2 /* Number of channels */
#define S_FREQ  1000 /* Scan freq. -> 1000 Hz */
#define MSB   40960/* Max scans/buffer => 이 값은 scan 버퍼의 크기를 결정하는데 */
                   /* 1024 의 배수로 지정하는것이 좋다.                    */

void main (void)
{
  HANDLE hDevice;
  int ch_list[2] = {0, 4}; /* Scan channel list : 0 번 과 4 번 채널 */
  float progress;
  ULONG dwSavedScanCnt=0;

  /* Load DLL */
  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }
  /* Load Device */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
```

```
  }

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

  /* start unlimited scan */
  if(COMI_US_Start (hDevice, NUM_CH, ch_list, S_FREQ, MSB, TRS_SINGLE) <
0){
    /* Error 처리 */
    COMI_ErrorString (COMI_LastError());
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    exit(0);
  }

  // Start file save //
  printf("A/D Scan data file save is started !\n");
  dwSavedScanCnt = 0;
  if(COMI_US_FileSaveFirst (hDevice, "c:\\ComiUscan.bin", TRUE))
  {
    while(!kbhit())
    {
      Sleep(100);
      // Continue file save //
      dwSavedScanCnt += COMI_US_FileSaveNext(hDevice);
      printf("Total save scan count = %u\n", dwSavedScanCnt);
    }
  }
  COMI_US_FileSaveStop(hDevice); // Stop file save
  COMI_US_Stop (hDevice, TRUE); // Stop unlimited scan

  // Start binary file to text file conversion //
  COMI_US_FileConvert ("c:\\ComiUscan.bin", "c:\\ComiUscan.txt", 0);

  // Conversion 진행률을 체크하여 사용자에게 보여준다.       //
  while((progress = COMI_US_CheckFileConvert()) < 100.f){
    printf("File conversion progress = %.1f %c\n", progress, '%');
  }

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

[리스트 3-8] Unlimited Scan 사용예 5 - 자동 파일 저장 기능에 의해 생성된 바이너리 파일을 사용자가 직접 읽어서 처리하는 예제

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Unlimited Scan
/* - Contents: 이프로그램은 UnlimScan4 의 예제와 같은 기능을 수행한다.
/*   단, 이 프로그램에서는 COMI_US_FileConvert () 함수를 사용하지 않고,
/*   바이너리 파일을 직접 읽어들여 텍스트 파일로 변환하게 된다.
/*   이 예제의 의도는 COMI_US_FileSaveFirst(), COMI_US_FileSaveNext()
/*   등의 함수를 이용하여 저장된 바이너리 파일을 사용자가 직접 읽는
/*   방법을 보이기 위한 것이다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*************************************************************************/

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

#define NUM_CH  2 /* Number of channels */
#define S_FREQ  1000 /* Scan freq. -> 1000 Hz */
#define MSB   40960/* Max scans/buffer => 이 값은 scan 버퍼의 크기를 결정하는데 */
                   /* 1024 의 배수로 지정하는것이 좋다.                    */


//*************************************************************************
//* BinToTextFileConversion() : COMI_US_FileSaveFirst() 등의 함수를 통해
//*   저장된 바이너리 파일을 읽어들여 텍스트 파일로 변환하는 함수
//* - szBinFilePath : 바이너리 파일의 Path
//* - szTextFilePath : 텍스트 파일의 Path
//* - Return : 0 => 실패, 1 => 성공
//*************************************************************************/

BOOL BinToTextFileConversion(char *szBinFilePath, char *szTextFilePath)
{
  TScanFileHead Header; // Binary file 에 저장된 Header information 구조체
  FILE *fpBin, *fpTxt;
  ULONG dwCnvtedCnt = 0;
  short digits[16];
  float volt, fProgress;
```

```
  int i;

  if((fpBin = fopen(szBinFilePath, "rb")) == NULL)
    return FALSE;

  if((fpTxt = fopen(szTextFilePath, "w")) == NULL){
    fclose(fpBin);
    return FALSE;
  }

  fseek(fpBin, 0L, SEEK_SET); // File pointer 를 처음으로 위치시킨다.

  // Binary file header 정보를 얻는다. TScanFileHead 형식의 구조체는 //
  // Comidas.h 파일에서 정의되어있다.                    //
  fread(&Header, sizeof(TScanFileHead), 1, fpBin);

  // Text file 에 타이틀 필드를 작성한다. //
  fprintf(fpTxt, "Date : %s,  Time : %s\n", Header.szDate, Header.szTime);
  for(i=0; i<Header.nNumChan; i++)
    fprintf(fpTxt, "  CH%2d ", Header.nChanList[i]);
  fprintf(fpTxt, "\n");

  // Binary file 에서 1 scan line 씩 읽어들여 그 값을 Text file 에 //
  // 프린트한다.                                                  //
  while(fread(digits, sizeof(short), Header.nNumChan, fpBin))
  {
    for(i=0; i<Header.nNumChan; i++){
      if(Header.dmin < 0) // 16 Bit resolution : -32768 ~ 32767 //
        volt = (float)COMI_Digit16ToVolt(digits[i], Header.vmin[i],
                    Header.vmax[i]);
      else // 12 Bit resolution : 0 ~ 4095 //
        volt = (float)COMI_DigitToVolt(digits[i], Header.vmin[i],
                    Header.vmax[i]);
      fprintf(fpTxt, "%7.3f ", volt);
    }
    fprintf(fpTxt, "\n");
    dwCnvtedCnt++;
    if(!(dwCnvtedCnt % 100)){ // 100 회의 scan line 데이터가 저장될때마다 진행률을
표시
      fProgress= 100.f*(float)dwCnvtedCnt/(float)Header.dwSavedScanCnt;
      // Conversion 진행률을 화면에 보여준다. //
      printf("Conversion progress = %.0f %c\n", fProgress, '%');
    }
  }
  fclose(fpTxt);
  fclose(fpBin);
  return TRUE;
}
```

```
void main (void)
{
  HANDLE hDevice;
  int ch_list[2] = {0, 4}; /* Scan channel list : 0번과 4번 채널 */
  ULONG dwSavedScanCnt=0;
  BOOL bSuccess;

  /* Load DLL */
  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }
  /* Load Device */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

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

  /* start unlimited scan */
  if(COMI_US_Start (hDevice, NUM_CH, ch_list, S_FREQ, MSB, TRS_SINGLE) <
0){
    /* Error 처리 */
    COMI_ErrorString (COMI_LastError());
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    exit(0);
  }

  // Start file save //
  printf("A/D Scan data file save is started !\n");
  dwSavedScanCnt = 0;
  if(COMI_US_FileSaveFirst (hDevice, "c:\\ComiUscan.bin", TRUE))
  {
    while(!kbhit())
    {
      Sleep(100);
      dwSavedScanCnt += COMI_US_FileSaveNext(hDevice); // Continue file
save
      printf("Total save scan count = %u\n", dwSavedScanCnt);
    }
  }
```

```
  COMI_US_FileSaveStop(hDevice); // Stop file save
  COMI_US_Stop (hDevice, TRUE); // Stop unlimited scan

  // Convert binary file to text file //
  bSuccess = BinToTextFileConversion ("c:\\ComiUscan.bin",
"c:\\ComiUscan.txt");
  if(!bSuccess)
    printf("File conversion failed !");

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

3-3 고속 PID 제어

이 기능은 COMI-SD101, COMI-SD102, COMI-SD103 보드에서만 지원되는 기능으로써 A/D 와 D/A 채널을 이용하여 고속 PID 제어를 할 수 있도록 하는 기능입니다. 동시에 2 개의 시스템에 대한 PID 제어가 가능합니다. 본 라이브러리에서 적용된 PID 제어 알고리즘은 위치형(Position form) 제어 알고리즘으로써 수식으로 표현하면

$$u(t) = K_p[e(t) + \frac{1}{T_i}\int e(t)dt + T_d\frac{de(t)}{dt}] \qquad \text{(식 2-1)}$$

으로 나타납니다. (식 2-1)을 이산식으로 변형하면

$$u(n) = K_p[e(n) + \frac{T_s}{T_i}\sum_{j=1}^{n-1} e(j) + \frac{T_d}{T_s}\{e(n) - e(n-1)\}] \qquad \text{(식 2-2)}$$

과 같습니다. 여기서 K_p 는 비례 게인, T_i 는 적분 시간, T_d 는 미분시간이며, T_s 는 샘플링 시간입니다.

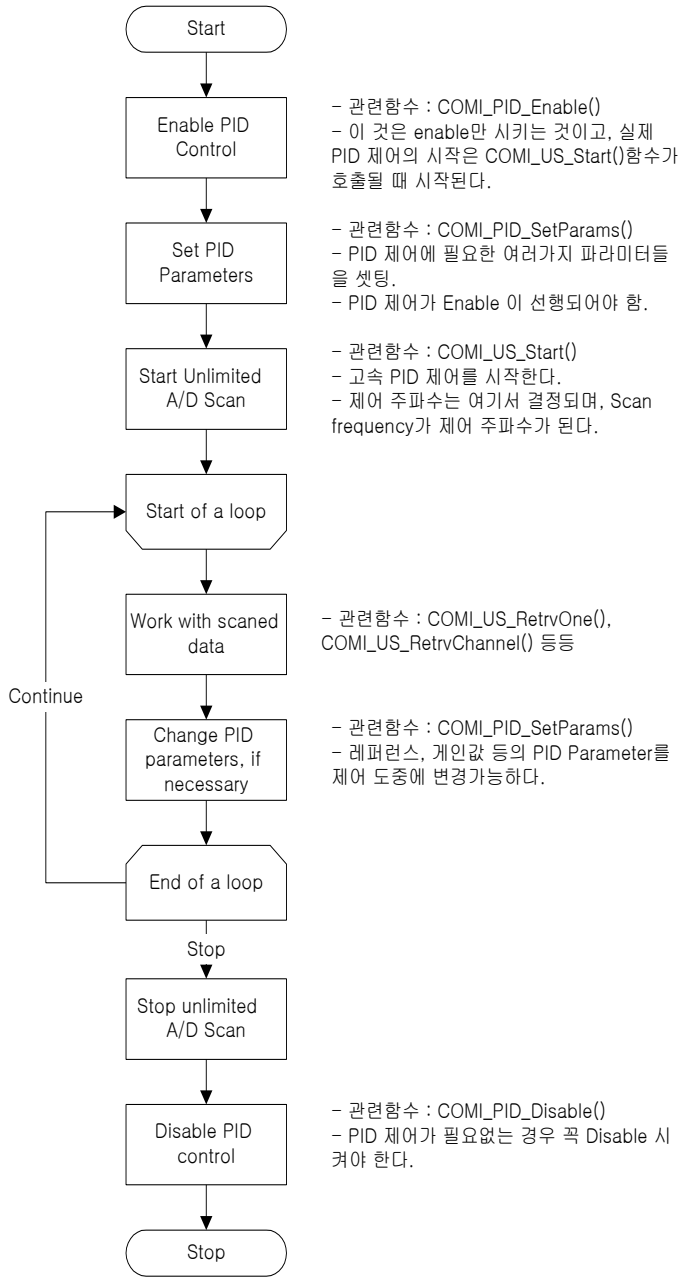
본 고속 PID 제어 기능은 디바이스 드라이버 내부에서 인터럽트를 이용하여 제어하므로써 제어 주기가 정확하며, 고속 실시간 제어가 가능하도록 되어 있습니다. 본 고속 PID 제어 기능은 1 ~ 15000Hz 의 범위내에서 제어 주파수를 선택할 수 있습니다.

본 고속 PID 제어 기능은 Unlimited Scan 기능을 이용합니다. 따라서 고속 PID 제어 기능을 사용하기 위해서는 앞 단원에 소개된 Unlimited A/D Scan 기능을 숙지하셔야 합니다.

Unlimited A/D Scan 함수 이외에 고속 PID 제어 기능과 관련된 함수들은 다음과 같습니다.

- BOOLCOMI_PID_Enable (HANDLE hDevice)
- BOOLCOMI_PID_SetParams (HANDLE hDevice, int nNumCtrls, TPidParams *pPidParams)
- BOOLCOMI_PID_Disable (HANDLE hDevice)

고속 PID 제어 함수를 이용하여 제어를 하는 방법을 간단한 순서도로 표현하면 다음과 같습니다.



◈ BOOL COMI_PID_Enable (HANDLE hDevice)

이 함수는 고속 PID 제어가 가능하도록 합니다. PID 제어를 Enable 시킨 후 Unlimited Scan 을 시작하면 고속 PID 제어가 수행됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스** : COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104

◈ BOOL COMI_PID_SetParams (HANDLE hDevice, int nNumCtrls, TPidParams *pPidParams)

이 함수는 고속 PID 제어에 필요한 여러 가지 파라미터들을 셋팅하는 함수입니다. 파라미터 셋팅은 제어가 이루어지고 있는 도중에도 변경이 가능합니다. 단, 이 함수는 PID 제어가 Enable 되지 않은 상태에서 수행하면 오류를 유발할 수 있습니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nNumCtrls* : PID 제어 시스템의 수를 지정합니다. PID 시스템을 2 개까지 동시에 제어할 수 있습니다.
- *pPidParams* : 고속 PID 제어에 필요한 여러가지 파라미터들을 셋팅하는 TPidParams 형 구조체 포인터입니다. 만일 nNumCtrls 를 2 로 지정했다면 pPidParams 는 TPidParams[2]의 배열이어야 합니다.

  TPidParams 구조체는 다음과 같이 구성됩니다.

```
typedef struct{
float Ref, lim_h, lim_l;
float Kp, Td, Ti;
int ch_ref, ch_ad, ch_da;
}TPidParams;
```

  - **Ref** : 레퍼런스값을 지정합니다. 만일 ch_ref 가 0 또는 양수값으로 지정되면 이 값은 사용되지 않습니다.

- **lim_h** : 시스템에 인가되는 조작량(D/A 출력)의 Maximum 값.
- **lim_l** : 시스템에 인가되는 조작량(D/A 출력)의 Minimum 값.
- **Kp** : 비례게인
- **Td** : 미분시간. 단 이 값은 msec 단위임에 주의하십시오.
- **Ti** : 적분시간. 단 이 값은 msec 단위임에 주의하십시오.
- **ch_ref** : 레퍼런스 입력을 A/D 채널을 통해 외부에서 주고자 할 때 사용되는 A/D 채널을 지정합니다. 이 값을 -1 로 지정하면 레퍼런스는 Ref 멤버 변수에서 지정한 값으로 설정됩니다.
- **ch_ad** : 제어대상의 출력을 Feedback 받을 A/D 채널 번호를 지정합니다.
- **ch_da** : 제어대상에 조작량을 인가할 D/A 채널 번호를 지정합니다.

- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :** COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104

◈ BOOL COMI_PID_Disable (HANDLE hDevice)

이 함수는 고속 PID 제어를 Disable 시킵니다. 고속 PID 제어를 종료하는 경우에는 반드시 이 함수를 수행해야 합니다.

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

- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :** COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104

[리스트 3-9] 고속 PID 제어를 수행하는 예제 (지면 관계상 에러 처리는 생략)

```
/*********************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Unlimited Scan
/* - Contents: COMI-SD101 보드를 이용하여 PID Control 을 하는 예제입니다.
/*   본 예제는 다음과 같은 내용을 담고 있습니다.
/*   1. A/D CH0 을 제어대상의 출력을 Feedback 받는 채널로 사용한다.
/*   2. D/A CH0 을 제어대상에 조작량을 인가하는 채널로 사용한다.
/*   3. 레퍼런스 입력은 기본적으로 파라미터를 통하여 입력되도록 되어 있으며, 0V 와 4V 가
/*      약 2 초 간격으로 레퍼런스값으로 설정된다. 만일 외부에서 레퍼런스 입력을 주고자 한다면
/*      '#define REF_CH'의 값을 원하는 A/D 채널로 설정하면 된다.
/*  3. 제어 주파수는 선언된 '#define S_FREQ' 의 값이 결정하며 Default 는 500Hz 로 지정
되었다.
/*   4. 제어가 시작되면 레퍼런스값 및 각 A/D 채널의 값이 C:\COMIPID.TXT 파일에 저장된다.
/*   5. 제어를 시작한 후 아무키나 누르면 프로그램이 종료된다.
/*
/* - Remarks :
/*   1. 이 예제는 COMI-SD101, COMI-SD102, COMI-SD103 에서만 적용되는 예제입니다.
/*********************************************************************/

#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <mmsystem.h>
#include "Comidas.h"

#define DEV_ID  COMI_SD101
#define NUM_CH  3 /* Number of channels */
#define S_FREQ  500  /* Scan freq. -> 500 Hz */
#define MSB   40960  /* Max scans/buffer => 이 값은 scan 버퍼의 크기를 결정. */

#define FEED_CH    0 // - 제어대상의 출력을 Feedback 받는 A/D 채널 설정
#define DA_CH      0 // - 제어대상에 조작량을 인가할 D/A 채널 설정
#define REF_CH    -1 // - 레퍼런스 채널 : -1 => 레퍼런스값을 파라미터에 의해 설정

#define LIMIT_H   10.f // Limiter high value - D/A 출력의 Limit
#define LIMIT_L   0.f // Limiter low value - D/A 출력의 Limit
#define Kp 5
#define Td 0
#define Ti 0

#define ToggleValue(v, l, h) ((v==l) ? h: l)
```

```
#define MAX_RET_SIZE 1024   // Scan buffer 에서 데이터를 전달받을때 최대로 받을 수 있는 크기 //
float Buffer[MAX_RET_SIZE*NUM_CH];
void main (void)
{
  HANDLE hDevice;
  FILE *fp;
  ULONG prv_cnt, ret_count, prv_time, cur_time;
 TPidParams PidParams = {0.f, LIMIT_H, LIMIT_L, Kp, Td, Ti, REF_CH, DA_CH, FEED_CH};
  int ch_list[NUM_CH] = {0, 1, 2};  // CH0 - 제어대상의 출력을 FEEDBACK
                                    // CH1, CH2 - 사용자에 따라  필요한 환경 계측

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

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

  COMI_PID_Enable(hDevice);
  COMI_PID_SetParams (hDevice, 1, &PidParams); /* PID_SetParams(..) 함수는 반드시 PID_Enable(..)함수가 수행된  이후에 그리고 PID_Disable()함수 이전에 수행되어야 한다. */
  /* start unlimited scan */
  if(COMI_US_Start (hDevice, NUM_CH, ch_list, S_FREQ, MSB, TRS_SINGLE) < 0){
    /* Error 처리 */
    printf("Error : Failed to start unlimited scan!");
    printf("%s", COMI_ErrorString(COMI_LastError()));
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
    printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
    _getch();
```

```
    exit(0);
  }

  prv_time = timeGetTime();
  /* Create a file to save data */
  fp = fopen ("c:\\ComiPID.txt", "w");
  fprintf (fp," Refer    CH0    CH1    CH2\n");
  printf("Unlimited scan has started !!\nPress any key to quit\n");

  prv_cnt = 0L; // initialize count variable
  while(!kbhit())
  {
    // 약 2 초마다 레퍼런스 값을 바꾸어 펄스 형태의 레퍼런스 입력이 되도록 한다. //
    // 만일 외부 레퍼런스 입력으로 설정한 경우에는 다음의 구문은 무의미하다.  //
    cur_time = timeGetTime();
    if(cur_time - prv_time > 1000 ){
      PidParams.Ref = ToggleValue(PidParams.Ref, 0.f, 4.f);
      COMI_PID_SetParams (hDevice, 1, &PidParams);
      prv_time = cur_time;
    }

    // 결과를 파일에 저장한다. //
    ret_count = COMI_US_RetrvBlock (hDevice, prv_cnt+1, MAX_RET_SIZE,
Buffer, VT_FLOAT);
    for(ULONG i=0; i<ret_count; i++)
    {
      fprintf(fp,   "%7.2f   %8.3f   %8.3f   %8.3f\n",   PidParams.Ref,
Buffer[i*NUM_CH], Buffer[i*NUM_CH+1], Buffer[i*NUM_CH+2]);
    }
    prv_cnt = prv_cnt + ret_count;
    printf("Reference = %6.2f      Number of saved data = %u\n",
PidParams.Ref, prv_cnt);
  }

  COMI_US_Stop (hDevice, TRUE);
  COMI_PID_Disable(hDevice);
  fclose(fp);

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

3-4 디지털 입출력

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. Digital Input/Output 에 관련된 함수는 다음과 같습니다.

- void COMI_DIO_SetUsage (HANDLE hDevice, int usage)
- int COMI_DI_GetOne (HANDLE hDevice, int ch)
- DWORD COMI_DI_GetAll (HANDLE hDevice)
- DWORD COMI_DI_GetAllEx (HANDLE hDevice, int nGroupIdx)
- BOOL COMI_DO_PutOne (HANDLE hDevice, int ch, int status)
- BOOL COMI_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)
- BOOL COMI_DO_PutAllEx (HANDLE hDevice, int nGroupIdx, DWORD dwStatuses)
- int COMI_DO_GetOne (HANDLE hDevice, int ch)
- DWORD COMI_DO_GetAll (HANDLE hDevice)
- DWORD COMI_DO_GetAllEx (HANDLE hDevice, int nGroupIdx)

함수명	보드별 지원 여부							
	CP101/201/301	CP401	SD10x/201/501	SD301	SD402	SD403	SD4x4	SD502
COMI_DIO_SetUsage		v		v		v		v
COMI_DI_GetOne	v	v	v	v		v	v	v
COMI_DI_GetAll	v	v	v	v		v	v	v
COMI_DI_GetAllEx	v	v	v	v		v	v	v
COMI_DO_PutOne	v	v	v	v	v		v	v
COMI_DO_PutAll	v	v	v	v	v		v	v
COMI_DO_PutAllEx	v	v	v	v	v		v	v
COMI_DO_GetOne	v	v	v	v	v		v	v
COMI_DO_GetAll	v	v	v	v	v		v	v
COMI_DO_GetAllEx	v	v	v	v	v		v	v

[표 3-4] 디지털 입출력 함수 리스트 및 각 보드별 지원 여부

◈ void COMI_DIO_SetUsage (HANDLE hDevice, int usage)

이 함수는 디지털 입출력 채널의 용도를 설정하는 함수입니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *usage* : 디바이스의 용도를 선택합니다. 이 값은 다음의 세 값 중 하나이어야 합니다.
  - ▷ **DI_ONLY** => DIO 채널 모두를 Digital Input 채널로 사용.
  - ▷ **DO_DI** => DIO 채널의 전반부를 Output 으로 후반부를 Input 으로 사용.
  - ▷ **DI_DO** => DIO 채널의 전반부를 Input 으로 후반부를 Output 으로 사용.
  - ▷ **DO_ONLY** => DIO 채널 모두를 Digital Output 채널로 사용.
- *Remarks* : 모든 제품이 디지털 입출력 채널의 용도를 소프트웨어적으로 설정할 수 있는 것은 아닙니다. 이 함수를 지원하는 제품과 각 설정값에 따른 채널 구분은 다음과 같습니다.

<table>
<tr><th rowspan="2">제품명</th><th rowspan="2">채널
구분</th><th colspan="4">용도 설정값</th></tr>
<tr><th>DI_ONLY</th><th>DO_DI</th><th>DI_DO</th><th>DO_ONLY</th></tr>
<tr><td rowspan="2">COMI-CP401</td><td>Input</td><td>CH0~CH15</td><td>CH8~CH15</td><td>CH0~CH7</td><td>NONE</td></tr>
<tr><td>Output</td><td>NONE</td><td>CH0~CH7</td><td>CH8~CH15</td><td>CH0~CH15</td></tr>
<tr><td rowspan="2">COMI-SD301</td><td>Input</td><td>CH0~CH31</td><td>CH16~CH31</td><td rowspan="2">지원안함</td><td>NONE</td></tr>
<tr><td>Output</td><td>NONE</td><td>CH0~CH15</td><td>CH0~CH31</td></tr>
<tr><td rowspan="2">COMI-SD502</td><td>Input</td><td>CH0~CH2</td><td rowspan="2">지원안함</td><td rowspan="2">지원안함</td><td>NONE</td></tr>
<tr><td>Output</td><td>NONE</td><td>CH0~CH2</td></tr>
</table>

◈ int COMI_DI_GetOne (HANDLE hDevice, int ch)

이 함수는 지정한 Digital Input 채널의 Status 를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Digital Input 채널번호. 채널번호는 0 부터 시작합니다.
- *Return* : Digital Input 채널의 Status. 0 - OFF, 1 - ON.

☞ **지원 디바이스** : All devices.

◈ DWORD COMI_DI_GetAll (HANDLE hDevice)

이 함수는 해당 디바이스의 모든 Digital Input 채널의 Status 를 반환합니다. 단, 32 채널보다 많은 디지털 입력 채널을 가진 제품에서는 CH0~CH31 의 Status 만 반환됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 모든 Digital Input 채널의 Status 를 나타내는 32 bit 값. 이 값의 각 비트의 값이 각 채널의 Status 를 나타냅니다.

☞ **지원 디바이스** : All devices.

◈ DWORD COMI_DI_GetAllEx (HANDLE hDevice, int nGroupIdx)

이 함수는 지정한 디지털 입력 디바이스로부터 32 개 채널의 Status 를 32 비트값으로 반환합니다. 이 함수는 32 채널보다 많은 디지털 입력 채널을 제공하는 장치에서 32 채널씩 데이터를 읽어들일 수 있도록 하기 위한 함수로써, 이때의 채널은 nGroupIndex 파라미터에 따라 달라집니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nGroupIdx* : 32 채널 단위의 채널 그룹인덱스를 지정합니다. 예를 들어 이 값이 0 이면 CH0~CH31 의 Status 를, 1 이면 CH32~CH63 의 Status 를 반환하도록 합니다.
- *Return* : 32 개의 채널에 대한 Input Status 를 32 비트 값으로 반환합니다. 각비트는 비트 순서와 일치하여 각 채널의 ON/OFF 상태를 나타냅니다.

☞ **지원 디바이스** : All devices.

◈ BOOL COMI_DO_PutOne (HANDLE hDevice, int ch, int status)

이 함수는 지정한 Digital Output 채널에 지정한 Status 로 출력을 내보냅니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Digital Output 채널번호. 채널 번호는 0 부터 시작한다.

- *status* : 출력 Status. 0 - OFF, 1 - ON.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스** : All devices.

◈ BOOL COMI_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)

해당 디바이스의 모든 Digital Output 채널에 출력을 내보낸다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_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.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스** : All devices.

◈ BOOL COMI_DO_PutAllEx (HANDLE hDevice, int nGroupIdx, DWORD dwStatuses)

이 함수는 32 개 Digital Output 채널에 출력을 내보냅니다. 이 함수는 32 채널보다 많은 디지털 출력 채널을 제공하는 장치에서 32 채널씩 출력을 제어할 수 있도록 하기 위한 함수로써, 이때의 채널은 nGroupIndex 파라미터에 따라 달라집니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nGroupIdx*: 32 채널 단위의 채널 그룹인덱스를 지정합니다. 예를 들어 이 값이 0 이면 CH0~CH31 에 출력을 내보내고, 1 이면 CH32~CH63 에 출력을 내보내도록 합니다.
- *dwStatuses* : 32 채널의 Digital Output 채널의 출력 Status 를 나타내는 32 bit 값. 이 값의 각 비트의 값이 각 채널의 Status 를 나타내며, 이때의 해당 채널그룹은 nGroupIndex 파라미터에 의해 결정됩니다.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :** All devices.

◈ int COMI_DO_GetOne (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 모든 Digital Output 채널
- *Return* : Digital Output 채널의 출력 Status. 0 - OFF, 1 - ON.

☞ **지원 디바이스 :** All devices.

◈ DWORD COMI_DO_GetAll (HANDLE hDevice)

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

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 32 개의 채널에 대한 현재 출력 상태를 32 비트 값으로 반환합니다. 각비트는 비트 순서와 일치하여 각 채널의 ON/OFF 상태를 나타냅니다. 단, 디바이스에 따라 32 채널 미만의 Digital 채널을 지원하는 경우에는 BIT0 부터 해당 채널 수 만큼의 비트만 사용하시면 됩니다.

☞ **지원 디바이스 :** All devices.

◈ DWORD COMI_DO_GetAllEx (HANDLE hDevice, int nGroupIdx)

이 함수는 32 채널보다 많은 디지털 출력 채널을 제공하는 장치에서 32 채널씩 출력 상태를 알아 볼수 있도록 하기 위한 함수로써, 이때의 채널은 nGroupIndex 파라미터에 따라 달라집니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nGroupIdx* : 32 채널 단위의 채널 그룹인덱스를 지정합니다. 예를 들어 이 값이 0 이면 CH0~CH31 디지털 출력채널의 현재 출력 상태를 반환하고, 1 이면 CH32~CH63 디지털 출력 채널의 현재 출력 상태를 반환합니다.

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

☞ **지원 디바이스 :** All devices.

[리스트 3-9] Single Channel Digital Input/Output

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : D/I and D/O
/* - Contents: 이 프로그램은 COMI_DO_PutOne(..)과 COMI_DI_GetOne(..) 함수를
/*    사용하여 각 1 채널에 대한 D/I 와 D/O 를 수행하는 예제입니다.
/* - Remarks :
/*    1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*       사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*       ID 로 바꾸어야 합니다.
/*    2. 이 프로그램은 D/O 채널 0 번을 통하여 On/Off 신호를 번갈아가면서 내보내고
/*      D/I 채널 0 번을 읽어들이는 프로그램입니다. D/O 0 번 채널과 D/I 0 번 채널을
/*       서로 연결해놓으면 D/O 에서 내보낸 신호를 D/I 채널에서 읽어들일수 있습니다.
/*************************************************************************/

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

#define DI_CH 0
#define DO_CH 0

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

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

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  // COMI-CP401 보드의 경우에는 DI/DO 기능을 모두 사용하기 위해 //
  // 여기서 COMI_DIO_SetUsage(DO_DI) 구문을 써주어야 합니다. //

  while(!kbhit())
```

```
  {
    do_state ^= 1; // state 반전
    COMI_DO_PutOne (hDevice, DO_CH, do_state); // Put D/O

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


  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

[리스트 3-10] Multi Channel Digital Input/Output

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : D/I and D/O
/* - Contents: 이 프로그램은 COMI_DI_GetAll(..)과 COMI_DO_PutAll(..) 함수를
/*   사용하여 D/I 와 D/O 의 8 채널을 동시에 콘틀롤하는 예제입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*************************************************************************/


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

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

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

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  // COMI-CP401 보드의 경우에는 DI/DO 기능을 모두 사용하기 위해 //
  // 여기서 COMI_DIO_SetUsage(DO_DI) 구문을 써주어야 합니다. //

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

```
    /* Get D/I and print on screen */
    di_states = COMI_DI_GetAll(hDevice);
    /* di_states 는 전채널의 state 를 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);
  }


  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

[리스트 3-10-2] 64 채널 디지털 입력 예제

```
/*****************************************************************
* [COMIDAS sample program by COMIZOA Inc., Ltd]
*
* - Subject : 64 비트 Digital Input
* - Contents: 이 프로그램은 COMI_DI_GetAllEx()함수를 사용하는 예제로써, 32 채널씩 나누
* 어서 총 64 채널의 디지털 입력값을 읽어들여 화면에 HEXA-DECIMAL 값으로 그 상태를 표시합니다.
*****************************************************************/

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

#define DEV_ID  COMI_SD403

void main (void)
{
  HANDLE hDevice;
  DWORD di_states[2];

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

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

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

  while(!kbhit())
  {
    /* Read first 32 D/I channels (CH0~CH31) */
    di_states[0] = COMI_DI_GetAllEx(hDevice, 0);
    /* Read second 32 D/I channels (CH32~CH63) */
    di_states[1] = COMI_DI_GetAllEx(hDevice, 1);
```

```
    printf("D/I States : CH0~CH31=%08lx  CH32~CH63=%08lx\n", di_states[0],
di_states[1]);
    Sleep(10);
  }

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

[리스트 3-10-3] 64 채널 디지털 출력 예제

```
/*****************************************************************
* [COMIDAS sample program by COMIZOA Inc., Ltd]
*
* - Subject : 64 비트 Digital Output
* - Contents: 이 프로그램은 COMI_DO_PutAllEx() 함수를 사용하는 예제로써 32 채널씩 나누
*   어서 총 64 채널의 디지털 출력을 내보냅니다. 이 때의 출력 상태는 전 채널 모두 on/off 를 반
*   복하여 출력됩니다.
/*****************************************************************/

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

#define DEV_ID  COMI_SD402

void main (void)
{
  HANDLE hDevice;
  DWORD do_states[2]={0x0, 0x0};

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

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

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

  while(!kbhit())
  {
    do_states[0] = ~do_states[0]; /* toggle the states of D/O channels */
    COMI_DO_PutAllEx(hDevice,  0,  do_states[0]);/*  Put  first  32  D/O
```

```
channels (CH0~CH31) */

    do_states[1] = ~do_states[1]; /* toggle the states of D/O channels */
    COMI_DO_PutAllEx(hDevice, 1, do_states[1]); /* Put second 32 D/O
channels (CH32~CH63) */

    printf("D/O States : CH0~CH31=%08lx CH32~CH63=%08lx\n",
      do_states[0], do_states[1]);
    Sleep(500);
  }

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

3-5 아날로그 출력

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	보드별 지원 여부										
	CP101	CP201	CP301/2	CP401	CP501	SD10x	SD201	SD301	SD4xx	SD501	SD502
COMI_DA_Out	V		V			V		V			
COMI_WFM_Start								V			
COMI_WFM_RateChange								V			
COMI_WFM_GetCurPos								V			
COMI_WFM_GetCurLoops								V			
COMI_WFM_Stop								V			

[표 3-5] 아날로그 출력 함수 리스트 및 각 보드별 지원 여부

3-5-1 일반적인 Analog Output 함수

◈ BOOL COMI_DA_Out (HANDLE hDevice, int ch, float volt)

이 함수는 지정한 Analog Output 채널에 지정한 Voltage 를 출력합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Analog Output 채널번호. 채널 번호는 0 부터 시작합니다.
- *volt* : Analog Output 출력 Voltage.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP301, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD301

[리스트 3-11] Single Point Analog Output 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Single Point D/A
/* - Contents: 이프로그램은 COMI_DA_Out(...)함수를 사용하여 Single point D/A를
/*     수행하는 프로그 램입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*     사용하는  경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*     ID로 바꾸어야 합니다.
/*************************************************************************/

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

#define CHAN  0
#define VMIN  -5
#define VMAX  5

void main (void)
{
  HANDLE hDevice;

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

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  while(!kbhit())
  {
    printf("D/A Out 5 volt !\n");
    COMI_DA_Out (hDevice, 0, 5.f); /* D/A CH0 에 5 volt를 내보낸다. */
    Sleep(1000); /* 1 sec delay */
    printf("D/A Out 0 volt !\n");
    COMI_DA_Out (hDevice, 0, 0.f); /* D/A CH0 에 0 volt를 내보낸다. */
```

```
    Sleep(1000); /* 1 sec delay */
  }

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

3-5-2 Waveform Generation

Waveform Generation 기능은 Sine Wave 또는 Square Wave 등과 같이 사용자가 지정하는 주기성을 가지는 신호를 아날로그 출력(D/A) 채널을 통하여 자동으로 생성해주는 기능입니다. 이 기능은 COMI-SD301 보드의 0 번 채널과 1 번채널에서만 지원하는 기능입니다.

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

이 함수는 Waveform Generation 을 시작합니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Waveform 신호를 출력할 D/A 채널 번호. 이 값은 0 또는 1 이어야 합니다.
- *pDataBuffer* : Waveform 데이터를 담은 버퍼의 주소값(포인터). Waveform Generation 기능을 지원하는 보드는 이 버퍼에 담겨진 데이터를 보드에 내장된 FIFO 메모리에 로드하므로 이 버퍼는 지역변수이어도 상관 없습니다.
- *nNumData* : 버퍼에 담겨진 데이터의 수
- *nPPS* : Waveform Generation 의 주기를 결정합니다. 이 값은 Points/Second 입니다. 예를 들어 100 개의 데이터로 한 주기를 구성하였다면 10Hz 의 신호를 만들기 위해서는 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* : 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.

☞ **지원 디바이스** : COMI-SD301

◈ long COMI_WFM_RateChange (HANDLE hDevice, int ch, ULONG nPPS)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Waveform 신호를 출력할 D/A 채널 번호. 이 값은 0 또는 1 이어야 합니다.
- *nPPS* : Waveform Generation 의 주기를 결정합니다. 이 값은 Points/Second 입니다. 예를 들어 100 개의 데이터로 한 주기를 구성하였다면 10Hz 의 신호를 만들기 위해서는 nPPS 는 1000 이 되어야 합니다.
- *Return* : 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.

☞ **지원 디바이스** : COMI-SD301

◈ long COMI_WFM_GetCurPos (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Waveform 신호를 출력할 D/A 채널 번호. 이 값은 0 또는 1 이어야 합니다.
- *Return* : 현재 출력되고 있는 주기 데이터의 인덱스(Index).

☞ **지원 디바이스** : COMI-SD301

◈ long COMI_WFM_GetCurLoops (HANDLE hDevice, int ch)

이 함수는 COMI_WFM_Start()함수의 nMaxLoops 파라미터를 양의 값으로 설정하였을 때만 사용가능한 함수입니다. 이 함수는 현재 남아있는 Wave 신호의 주기 수를 반환합니다. 예를 들어 nMaxLoops 를 1000 으로 하였을 때 이 함수

가 100 을 반환한다면 현재까지 900 회의 Wave 신호가 발생하였으며, 100 회의 Wave 신호가 남았음을 의미합니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Waveform 신호를 출력할 D/A 채널 번호. 이 값은 0 또는 1 이어야 합니다.
- *Return* : Wave 신호의 출력 횟수를 제한한 경우 현재 남아있는 Wave 신호의 출력 횟수. 이 값이 0 이면 지정한 횟수의 Wave 신호가 모두 출력되었음을 의미합니다.

☞ **지원 디바이스** : COMI-SD301

◈ void COMI_WFM_Stop (HANDLE hDevice, int ch)

Waveform Generation 을 종료합니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Waveform 신호를 출력할 D/A 채널 번호. 이 값은 0 또는 1 이어야 합니다.

☞ **지원 디바이스** : COMI-SD301

[리스트 3-12] Waveform Generation 사용예 1 : 이 예제는 COMI-SD301 보드의 D/A CH0 을 통하여 5Volt Amplitude 와 1KHz 의 주파수를 가지는 펄스 신호를 발생시키는 예제입니다.

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

#define DEV_ID  COMI_SD301
#define CHAN  0
#define AMP   5 // AMPLITUDE
#define FREQ  1000 // FREQUENCY

void main (void)
{
  HANDLE hDevice;
  float fWaveBuf[2]={AMP, -AMP};

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

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

  printf("Waveform Generation 을 시작하려면 아무키나 누르십시오.\n");
  printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
  _getch();
  COMI_WFM_Start(hDevice, CHAN, fWaveBuf, 2, 2*FREQ, 0);
  while (!_kbhit())
    ;
  COMI_WFM_Stop(hDevice, CHAN);
  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

[리스트 3-13] Waveform Generation 사용예 2 : 이 예제는 COMI-SD301 보드의 D/A CH0 을 통하여 5Volt Amplitude 와 1KHz 의 주파수를 가지는 Sine wave 신호를 발생시키는 예제입니다. 이 예제에서는 Sine wave 신호를 1000 주기씩 끊어서 발생시킵니다. 즉, 1000 주기의 Sine wave 신호를 내보낸 후 1 초 쉬고 다시 1000 주기의 Sine wave 신호를 출력합니다.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "Comidas.h"

#define DEV_ID  COMI_SD301
#define CHAN  0
#define AMP   5 // AMPLITUDE
#define FREQ  1000 // FREQUENCY
#define N   50

void main (void)
{
  HANDLE hDevice;
  float fWaveBuf[N], rad;
  long nCount;

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

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

  rad = 2*3.141592f/N;
  for(int i=0; i<N; i++)
    fWaveBuf[i] = AMP * sin(i*rad);

  printf("Waveform Generation 을 시작하려면 아무키나 누르십시오.\n");
```

```
  printf("아무키나 다시 누르면 프로그램이 종료됩니다.\n");
  _getch();

  while (!_kbhit()){
    COMI_WFM_Start(hDevice, CHAN, fWaveBuf, N, N*FREQ, 1000);
    printf("Waveform Generation has started!\n");
    while((nCount = COMI_WFM_GetCurLoops(hDevice, CHAN)) > 0){
      printf("Remainning Loops = %d\n", nCount);
    }
    printf("Waveform Generation has finished!");
    Sleep(1000);
  }
  COMI_WFM_Stop(hDevice, CHAN);
  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

3-6 카운터

카운터는 펄스 신호를 카운트하거나 펄스 신호를 출력하는데 사용되는 기능입니다. COMIDAS 는 다양한 형태의 카운터 기능을 제공합니다. 카운터와 관련된 함수들은 다음과 같습니다.

1) 8254 카운터 함수

- void COMI_SetCounter(HANDLE hDevice, int ch, int rw, int op, int bcd_bin, USHORT load_value)
- void COMI_LoadCount(HANDLE hDevice, int ch, USHORT load_value)
- USHORT COMI_ReadCount(HANDLE hDevice, int ch)

2) 32 비트 COMI-SD 카운터 함수

- ULONG COMI_ReadCounter32(HANDLE hDevice, int ch)
- void COMI_ClearCounter32(HANDLE hDevice, int ch)

3) 엔코더 카운터 함수

- void COMI_ENC_Config(HANDLE hDevice, int ch, int mode, BOOL bResetByZ)
- void COMI_ENC_Reset(HANDLE hDevice, int ch)
- void COMI_ENC_Load(HANDLE hDevice, int ch, long count)
- long COMI_ENC_Read(HANDLE hDevice, int ch)
- void COMI_ENC_ResetZ(HANDLE hDevice, int ch)
- void COMI_ENC_LoadZ(HANDLE hDevice, int ch, short count)
- short COMI_ENC_ReadZ(HANDLE hDevice, int ch)

4) 펄스 발생기 함수

- double COMI_PG_Start(HANDLE hDevice, int ch, double freq, UINT num_pulses)
- double COMI_PG_ChangeFreq(HANDLE hDevice, int ch, double freq)
- BOOL COMI_PG_IsActive(HANDLE hDevice, int ch)
- void COMI_PG_Stop(HANDLE hDevice, int ch)

5) COMI-SD502 카운터 전용 함수

- void COMI_SD502_SetCounter(HANDLE hDevice, int ch, int nMode,

UINT nClkSource)

- ULONG COMI_SD502_ReadNowCount(HANDLE hDevice, int ch)
- ULONG COMI_SD502_ReadOldCount(HANDLE hDevice, int ch)
- BOOL COMI_SD502_GetGateState(HANDLE hDevice, int ch)
- double COMI_SD502_GetClkFreq(int nClkSrcIdx)
- void COMI_SD502_Clear(HANDLE hDevice, int ch)
- void COMI_SD502_ClearMulti(HANDLE hDevice, ULONG dwCtrlBits)

함수명	보드별 지원 여부										
	CP101	CP201	CP301	CP401	CP501	SD10x	SD201	SD301	SD4xx	SD501	SD502
COMI_SetCounter	V	V			V						
COMI_LoadCount	V	V			V						
COMI_ReadCount	V	V			V						
COMI_ReadCounter32						V	V	V		V	
COMI_ClearCounter32						V	V	V		V	
COMI_ENC_Config										V	
COMI_ENC_Reset										V	
COMI_ENC_Load										V	
COMI_ENC_Read										V	
COMI_ENC_ResetZ										V	
COMI_ENC_LoadZ										V	
COMI_ENC_ReadZ										V	
COMI_PG_Start										V	
COMI_PG_ChangeFreq										V	
COMI_PG_IsActive										V	
COMI_PG_Stop										V	
COMI_SD502_SetCounter											V
COMI_SD502_ReadNowCount											V
COMI_SD502_ReadOldCount											V
COMI_SD502_GetGateState											V
COMI_SD502_GetClkFreq											V
COMI_SD502_Clear											V
COMI_SD502_ClearMulti											V

[표 3-6] 카운터 관련 함수 리스트 및 각 보드별 지원 여부

3-6-1 8254 카운터

COMI-CP101, COMI-CP201, COMI-CP501 보드에는 사용자가 사용할 수 있는 Intel 8254 카운터가 장착되어 있습니다. 8254 카운터는 펄스를 카운트하거나, 일정 주기의 펄스를 출력하는데 사용됩니다. Intel 8253/4 카운터에 대한 자세한 내용은 하드웨어 매뉴얼의 부록을 참조하기 바랍니다.

◈ void COMI_SetCounter (HANDLE hDevice, int ch, int rw, int op, int bcd_bin, USHORT load_value)

이 함수는 지정한 카운터 채널의 동작 방식을 결정하고, 원하는 카운트 값을 로드(Load)합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *rw* : Read/Write mode 를 선택합니다. 이 값은 다음 중 하나의 값이어야 합니다.
  - ▷ **READ_LOAD_MSB** => 상위 1 바이트값을 로드합니다.
  - ▷ **READ_LOAD_LSB** => 하위 1 바이트값을 로드합니다.
  - ▷ **READ_LOAD_WORD** => 2 바이트값을 로드합니다.
- *op* : 카운터의 동작 모드(Operation mode)를 결정합니다. 이 값은 CMODE0 , CMODE1, CMODE2, CMODE3, CMODE4, CMODE5 중의 하나이어야 합니다. Operation mode 에 관한 자세한 내용은 하드웨어 매뉴얼의 부록을 참조하십시오.
- *bcd_bin* : 카운트를 BCD 값으로 할 것인지, BINARY 값으로 할 것인지를 결정합니다. 이 값은 BCD 나 BINARY 중의 하나이어야 합니다. 일반적으로 BINARY 값을 입력합니다.
- *load_value* : 카운터에 로드할 값을 지정합니다. 이 값은 0 ~ 65535 사이의 값이어야 합니다.

☞ **지원 디바이스** : COMI-CP101, COMI-CP201, COMI-CP501

◈ void COMI_LoadCount (HANDLE hDevice, int ch, USHORT load_value)

이 함수는 지정한 카운터에 카운트값을 로드합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *load_value* : 카운터에 로드할 값을 지정합니다. 이 값은 0 ~ 65535 사이의 값이어야 합니다.

☞ **지원 디바이스** : COMI-CP101, COMI-CP201, COMI-CP501

◈ USHORT COMI_ReadCount (HANDLE hDevice, int ch)

이 함수는 지정한 카운터에서 카운트값을 읽어옵니다. 8254 카운터는 Decrease 카운팅을 합니다. 따라서 COMI_SetCounter() 또는 COMI_LoadCount()함수에서 로드(Load)한 값에서 이 함수를 통하여 읽은 카운트값을 뺀 값이 실제 카운트된 값이 됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 카운트 값

☞ **지원 디바이스** : COMI-CP101, COMI-CP201, COMI-CP501

[리스트 3-14] 8254 Counter 함수 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : Counter
/* Contents: 이 프로그램은 GATE0 에 5 volt 를 입력하고 ExtClk 에 일정
/*    Frequency 의 Pulse 신호를 입력하여 약 1 초동안의 Pulse 수를 반
/*    복해서 count 하는 프로그램입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는 경우에는 COMI_LoadDevice(...)함수의 첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*************************************************************************/

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

void main (void)
{
  HANDLE hDevice;
  int status=0;
  USHORT count;

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

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  COMI_SetCounter (0, READ_LOAD_WORD, CMODE0, BINARY, 65535);
  while(!kbhit())
  {
    Sleep(1000);
    COMI_LoadCount(0, 65535);
    count = 65535 - COMI_ReadCount(0) + 1;
    printf("Count = %u\n", count);
  }
```

```
    COMI_UnloadDevice(hDevice);
    COMI_UnloadDll();
}
```

3-6-2 32 비트 COMI-SD 카운터

COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD201 보드에는 사용자가 사용할 수 있는 COMI-SD 카운터가 각각 2 채널씩 장착되어 있습니다. 이 카운터는 Increment 방식을 사용하는 32 비트 카운터입니다.

◈ void COMI_ClearCounter32 (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.

☞ **지원 디바이스** : COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ ULONG COMI_ReadCounter32 (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 카운트 값을 읽어옵니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 카운트 값

☞ **지원 디바이스** : COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

[리스트 3-15] COMI-SD 32 비트 펄스 카운터 함수 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : 32 Bit Counter - SD 시리즈 보드에만 적용 가능
/* Contents: 이 프로그램은 0 번 카운터에 펄스를 입력하고,
/*     계속해서 Pulse 수를 count 하는 프로그램입니다.
/* - Remarks :
/*  1. 이 예제는 COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD201 보드에만 적용가능
합니다.
/*  2. 이 예제는 COMI-SD101 보드를 사용하는 것으로 작성되었습니다.
/*      따라서 다른 보드를 사용하는 경우에는 COMI_LoadDevice(...)함수의
/*      첫 번째 파라미터를 알맞은 디바이스 ID 로 바꾸어야 합니다.
/*************************************************************************/

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

#define DEV_ID  COMI_SD101
#define CNTR_CH 0


void main (void)
{
  HANDLE hDevice;
  ULONG count;

  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    exit(0);
  }
  hDevice = COMI_LoadDevice (DEV_ID, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }
  COMI_ClearCounter32 (hDevice, CNTR_CH); // Reset count to 0
  while(!kbhit()){
    count = COMI_ReadCounter32 (hDevice, CNTR_CH);
    printf("Read Count = %u\n", count);
  }
  COMI_UnloadDevice(hDevice);
```

```
   COMI_UnloadDll();
}
```

3-6-3 엔코더 카운터

이 단원에서 설명하는 카운터 기능은 COMI-SD501 보드에서만 적용되는 카운터 기능으로써 주로 엔코더 센서를 계측하는데 사용되는 카운터 기능입니다. 일반적으로 엔코더 센서는 회전체의 위치나 속도를 계측하기 위해 사용되는 센서입니다.

엔코더에서 발생되는 신호는 A 상, B 상 그리고 Z 상으로 구분되는 3 개의 신호이며 이들은 모두 펄스 형태로 출력되게 됩니다. A 상과 B 상 신호는 회전체의 미소 위치 변화(엔코더의 분해능에 따라 그 값은 다름)가 발생할 때마다 발생되는 펄스 신호로써, 이 두 신호는 일정한 위상차를 가지게 되어 회전 방향까지도 함께 알수 있습니다. COMI-SD501 은 회전 방향에 따라 자동으로 UP/DOWN 카운트가 됩니다. Z 상 신호는 회전체가 1 회전할 때마다 발생되는 펄스 신호입니다. COMI-SD501 은 Z 상 신호가 A/B 상 카운터의 값을 자동으로 리셋되도록 설정될 수 있습니다. 이 방식을 이용하면 회전체의 절대 위치를 파악하는데 용이합니다.

COMI-SD501 보드는 엔코더의 3 가지 신호를 모두 입력받을 수 있으며, A/B 상과 Z 상을 동시에 계측할 수 있습니다. 이 단원에서 소개되는 엔코더 카운터 함수중에서 'Z' 자가 함수명 끝에 붙은 함수는 Z 상 신호를 계측하는데 사용되는 함수이며, 나머지 함수들은 모두 A/B 상 신호를 계측하는데 사용되는 함수입니다.

※ COMI-SD501 디바이스에서 일반 펄스를 카운트하는 방법

COMI-SD501 에서는 엔코더 신호뿐만 아니라 일반 펄스 신호도 카운트할 수 있습니다. 일반 펄스 신호는 A 상과 B 상이 따로 존재하지 않으므로 신호선을 어떻게 연결해야 할지 혼동될 수 있습니다. 일반 펄스 신호를 카운트하기 위해서는 펄스 신호를 터미널 보드의 A 상 입력단에 연결하면 됩니다. 사용되는 라이브러리 함수는 엔코더 신호를 카운트하는 것과 동일합니다.

한편, 일반 펄스 신호를 카운트할 때에도 B 상 입력단에 0 Volt 또는 5 Volt 를 인가하여 Up/down 카운트를 할 수도 있습니다. B 상 입력단에 0 Volt 가 인가되면 업카운트(Up-count)가 되고, 5 Volt 가 인가되면 다운카운트(Down-count)가 됩니다. 만일 B 상 단자에 아무 신호도 연결되어 있지 않으면 업카운트를 하게 되며 펄스가 입력될 때마다 카운트는 증가하게 됩니다.

◈ void COMI_ENC_Config (HANDLE hDevice, int ch, int mode, BOOL bResetByZ)

이 함수는 지정한 A/B 상 카운터 채널의 모드를 설정합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *mode* : 채배모드를 설정합니다.
  - ▷ **0 또는 ENCODER_1X** => 엔코더의 기본 분해능대로 카운트합니다.
  - ▷ **1 또는 ENCODER_2X** => 2 채배 모드. 이 모드에서 카운터는 엔코더의 기본 분해능의 2 배 분해능을 가질 수 있습니다.
  - ▷ **2 또는 ENCODER_4X** => 4 채배 모드. 이 모드에서 카운터는 엔코더의 기본 분해능의 4 배 분해능을 가질 수 있습니다.
- *bResetByZ* : A/B 상 카운트값을 Z 상 입력단자에 펄스가 입력될 때마다 리셋(Reset)할 것인지를 지정합니다.
  - ▷ **0** => 이 값으로 지정하면 A/B 상 카운트값은 Z 상에 의해 영향을 받지 않습니다.
  - ▷ **1** => 이 값으로 지정하면 A/B 상 카운트값은 Z-펄스가 발생할 때마다 리셋(Reset)됩니다.

☞ **지원 디바이스** : COMI-SD501

◈ void COMI_ENC_Reset (HANDLE hDevice, int ch)

이 함수는 지정한 A/B 상 카운터 채널의 카운트값을 0 으로 리셋합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.

☞ **지원 디바이스** : COMI-SD501

◈ void COMI_ENC_Load (HANDLE hDevice, int ch, long count)

이 함수는 지정한 A/B 상 카운터 채널의 카운트값을 지정한 값으로 로드(Load)합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *count* : 카운터에 로드할 값. 이 값의 범위는 -2147483648 ~ 2147483647 입니다.
- *참고* : 일반적으로 이 함수는 회전체가 원점에서 시작되지 않는 경우 회전체의 초기 위치를 지정하기 위하여 사용됩니다. 카운트값을 0 으로 초기화하기 위해서는 이 함수를 사용하는 것보다 COMI_ENC_Reset()함수를 사용하는 것이 속도면에서 유리합니다.

☞ **지원 디바이스** : COMI-SD501

◈ long COMI_ENC_Read (HANDLE hDevice, int ch)

이 함수는 지정한 A/B 상 카운터 채널의 카운트값을 읽어서 그 값을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : A/B 상 카운트값.
- *참고* : 카운트값의 범위는 32 비트 정수값으로써 -2147483648 ~ 2147483647 입니다. 반환된 카운트값이 음의 값이면 회전체가 역방향으로 회전했음을 의미하며, 양의 값이면 정방향으로 회전했음을 의미합니다.

☞ **지원 디바이스** : COMI-SD501

◈ void COMI_ENC_ResetZ (HANDLE hDevice, int ch)

이 함수는 지정한 Z 상 카운터 채널의 카운트값을 0 으로 리셋합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.

☞ **지원 디바이스** : COMI-SD501

◈ void COMI_ENC_LoadZ(HANDLE hDevice, int ch, int count)

이 함수는 지정한 Z 상 카운터 채널의 카운트값을 지정한 값으로 로드(Load)합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *count* : 카운터에 로드할 값. 이 값의 범위는 -32767 ~ 32768 입니다.
- *참고* : 일반적으로 이 함수는 회전체가 한방향으로만 회전하는 경우에 Z 상 카운터의 16 비트값의 전체 영역을 모두 사용하기 위한 것입니다. 기본적으로 Z 상 카운터의 초기값은 0 이므로 정방향으로 회전시 최대 계측 가능한 펄스 수는 32767 회가 됩니다. 그러나 초기값을 -32767 값으로 로드한 후 계측한다면 정방향으로 회전시 65535 회의 펄스를 계측할 수 있습니다.

☞ **지원 디바이스** : COMI-SD501

◈ int COMI_ENC_ReadZ(HANDLE hDevice, int ch)

이 함수는 지정한 Z 상 카운터 채널의 카운트값을 읽어서 그 값을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : Z 상 카운트값.
- *참고* : 카운트값의 범위는 16 비트 정수값으로써 -32768 ~ 32767 입니다. 반환된 카운트값이 음의 값이면 회전체가 역방향으로 회전했음을 의미하며, 양의 값이면 정방향으로 회전했음을 의미합니다.

☞ **지원 디바이스** : COMI-SD501

[리스트 3-16] Encoder counter 함수 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Encoder Counter.
/* - Contents: 이 프로그램은 1 개의 Encoder counter 채널을 읽어서 화면에
/*       보여주는 프로그램입니다.
/* - Remarks :
/*  1. 이 예제는 COMI-SD501 보드에만 적용가능합니다.
/*************************************************************************/

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

#define CH 0

void main (void)
{
  HANDLE hDevice;
  long count;

  COMI_LoadDll();
  hDevice = COMI_LoadDevice (COMI_SD501, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }

  COMI_ENC_Config(hDevice, CH, ENCODER_1X, FALSE);
  COMI_ENC_Reset(hDevice, CH); /* Reset counter to 0 */

  while(!kbhit())
  {
    count = COMI_ENC_Read (hDevice, CH);
    printf("Read Encoder Count = %9ld\n", count);
  }

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

3-6-4 펄스 발생기

Pulse Generator(이후 PG) 기능은 COMI-SD501 보드에서만 제공하는 기능으로써 사용자가 원하는 주파수로 펄스를 생성시켜주는 기능입니다. 이 기능은 생성되는 펄스의 주파수를 제어할 수 있을 뿐 아니라 펄스의 수까지 제어할 수 있습니다. 이 기능은 주로 서보모터 및 스텝모터의 제어에 사용되는 기능입니다.

◈ double COMI_PG_Start (HANDLE hDevice, int ch, double freq, unsigned int num_pulses)

이 함수는 지정한 PG 채널을 통하여 지정한 주파수 및 펄스 수에 의거한 펄스를 출력합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.
- *freq* : 출력할 펄스의 주파수를 Hz 단위로 지정합니다.
- *num_pulses* : 출력할 펄스의 수를 지정합니다. 만일 이 값을 0 으로 지정하면 COMI_PG_Stop()함수가 수행되기 전까지 계속해서 펄스를 출력하게 됩니다.
- *Return* : 출력되는 펄스의 실제 주파수. 펄스의 주파수는 장치 내부에 장착된 타이머의 Base clock 을 정수로 분주하여 결정하므로 사용자가 지정한 주파수와 약간의 차이가 있을 수 있습니다.

☞ **지원 디바이스 :** COMI-SD501

◈ double COMI_PG_ChangeFreq (HANDLE hDevice, int ch, double freq)

이 함수는 지정한 PG 채널을 통하여 현재 출력되고 있는 펄스의 주파수를 Runtime 상에서 변경하여 줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.
- *freq* : 변경하고자 하는 펄스의 주파수를 Hz 단위로 지정합니다.

- *Return* : 출력되는 펄스의 실제 주파수.

☞ **지원 디바이스 :** COMI-SD501

◈ BOOL COMI_PG_IsActive(HANDLE hDevice, int ch)

이 함수는 지정한 PG 채널에 현재 펄스가 출력되고 있는지를 알려줍니다. 이 함수를 이용하면 출력 펄스 수를 제한한 경우에 원하는 펄스의 출력이 완료되었는지를 알아볼 수 있습니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 현재 펄스가 출력되고 있는지를 알려줍니다.
  0 => 현재 펄스가 출력되고 있지 않음
  1 => 현재 펄스가 출력되고 있음

☞ **지원 디바이스 :** COMI-SD501

◈ void COMI_PG_Stop(HANDLE hDevice, int ch)

이 함수는 지정한 PG 채널의 펄스 출력을 중지합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.

☞ **지원 디바이스 :** COMI-SD501

[리스트 3-17] Pulse Generator 함수 사용 예 1

```
/*********************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Pulse Generator
/* - Contents: 이 프로그램은 사용자가 지정하는 주파수를 가진 펄스를  사용자가 중단할 때까지
/*     계속하여 발생하는 프로그램입니다.
/* - Remarks :
/*  1. 이 예제는 COMI-SD501 보드에만 적용가능합니다.
/*********************************************************************/
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidas.h"

#define FREQ  1000.f /* Pulse freq. = 1 kHz */
#define PG_CH 0  /* Pulse Generator Channel number */

void main()
{
  HANDLE hDevice;
  double ActFreq;

  COMI_LoadDll();
  hDevice = COMI_LoadDevice (COMI_SD501, 0); /* Load device */

  printf("Generating infinite number of pulses through CH%d\n", PG_CH);
  printf("Set Freq.(Hz)   = %.0f\n", FREQ);
  /* Start pulse generation */
  ActFreq = COMI_PG_Start(hDevice, PG_CH, FREQ, 0);
  printf("Actual Freq.(Hz) = %.0f\n", ActFreq);
  printf("Press any key to exit !\n");
  while(!kbhit())
    ;
  COMI_PG_Stop (hDevice, PG_CH); /* Stop pulse generation */
  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

[리스트 3-18] Pulse Generator 함수 사용 예 2

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : Pulse Generator
/* Contents: 이 프로그램은 사용자가 지정하는 주파수의 펄스를 발생시킵니다.
/*     단, 이프로그램에서는 펄스의 수를 NBR_PULSES 값에 의해 제한합니다.
/* - Remarks :
/*  1. 이 예제는 COMI-SD501 보드에만 적용가능합니다.
/*************************************************************************/
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "comidas.h"

#define FREQ  1000.f /* Pulse freq. = 1 kHz */
#define NBR_PULSES 1000 /* 0 => Infinite number of pulses */
              /* others => The number of pulses to generate */
#define PG_CH 0 /* Pulse Generator Channel number */

void main()
{
  HANDLE hDevice;
  double ActFreq;

  COMI_LoadDll();
  hDevice = COMI_LoadDevice (COMI_SD501, 0); /* Load device */

  printf("Generating pulses through CH%d\n", PG_CH);
  printf("Set Freq.(Hz)   = %.0f\n", FREQ);
  if(NBR_PULSES == 0)
    printf("Num pulses = Infinite\n");
  else
    printf("Num pulses = %d\n", NBR_PULSES);
  /* Start pulse generation */
  ActFreq = COMI_PG_Start(hDevice, PG_CH, FREQ, NBR_PULSES);
  printf("Actual Freq.(Hz) = %.0f\n\n", ActFreq);
  while(!kbhit()){
    /* Check if generation is completed */
    if(!COMI_PG_IsActive(hDevice, PG_CH))
      break;
  }
  printf("Pulse generation is completed!\n");
  printf("Press any key to exit!\n");
  getch();
```

```
  COMI_PG_Stop (hDevice, PG_CH); /* Stop pulse generation */
  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

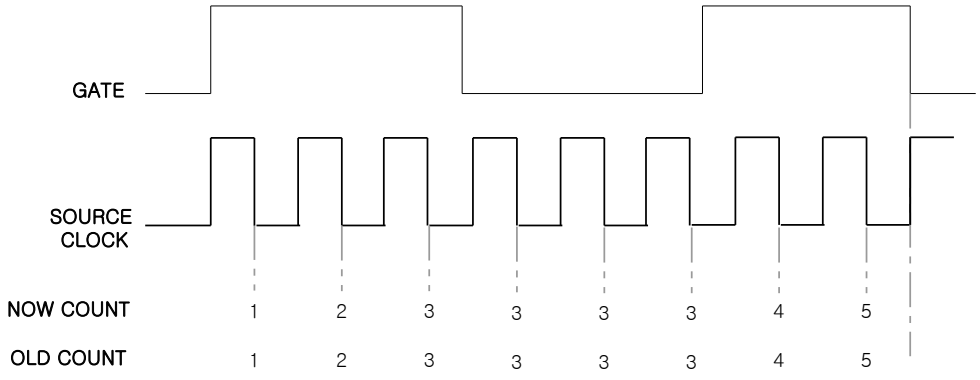
3-6-5 COMI-SD502 카운터

COMI-SD502 Counter 보드는 10 채널 24 비트 카운터 보드입니다. 특히 COMI-SD502 보드는 일반 카운터 기능외에 펄스 신호의 주파수를 쉽게 계측할 수 있는 기능을 제공합니다.

COMI-SD502 Counter 보드는 다음과 같이 두 가지 모드로 동작합니다.

■ MODE 0

이 모드는 일반적인 카운터 모드입니다. GATE 신호가 HIGH 상태로 유지되는 동안에 소스클럭(Source Clock)단에 입력되는 펄스의 수를 카운트합니다. 이 모드에서는 GATE 신호가 카운트 값을 자동으로 0 으로 초기화하지 않습니다. 그리고 GATE 신호에 아무 신호도 연결되지 않으면 GATE 신호는 자동적으로 High 상태로 유지됩니다. MODE 0 에서 카운터의 동작을 그림으로 표현하면 다음과 같습니다.

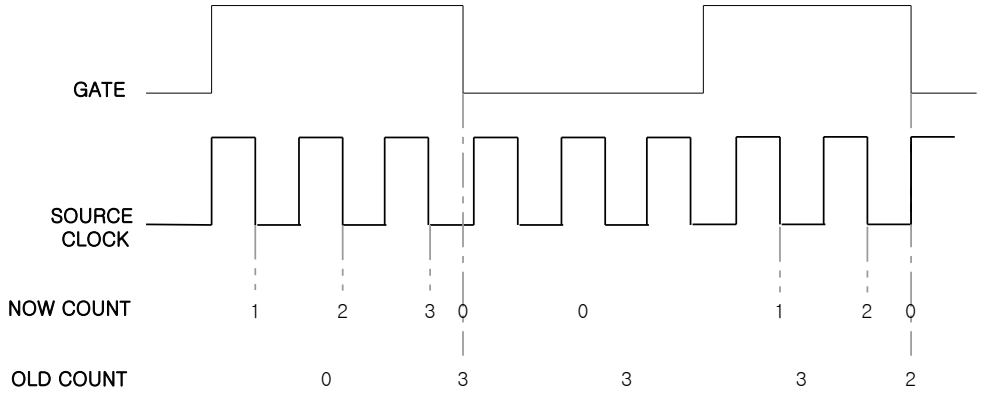


■ MODE 1

이 모드는 GATE 단자에 입력되는 펄스의 상태가 High 로 유지되는 시간을 계측하기 위해 주로 사용되는 모드입니다. 이 모드에서는 GATE 신호가 HIGH 상태로 유지되는 동안에 소스클럭(Source Clock)단에 입력되는 펄스의 수를 카운트하는 것은 MODE 1 과 동일합니다. 그러나 이 모드에서는 GATE 신호의 하강에지(Falling Edge)에서 현재의 카운트(Now count) 값을 Latch 에 저장(Old count) 한 후 0 으로 초기화하고, 다시 카운트를 시작합니다. 따라서

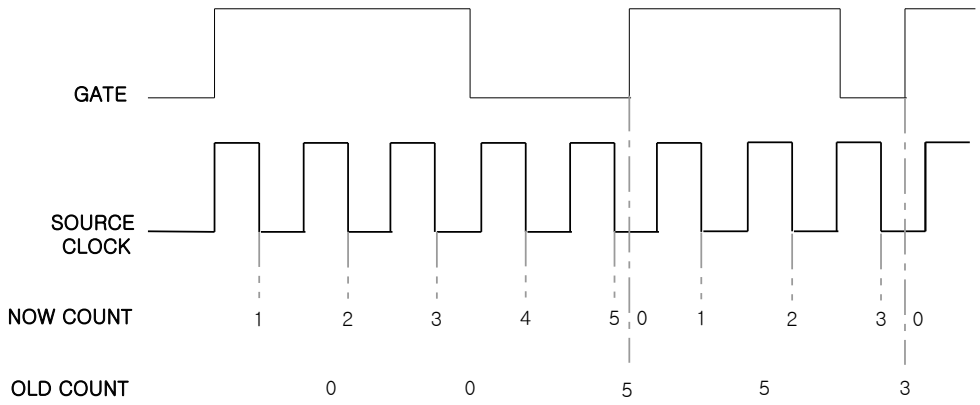
Old count 값을 참조하면 GATE 신호에 입력되는 펄스 신호의 주기 또는 주파수를 알 수 있습니다.

MODE 1 에서 카운터의 동작을 그림으로 표현하면 다음과 같습니다.



■ MODE 2

이 모드는 펄스의 주파수를 측정하는데 용이하게 사용될 수 있는 모드로써 GATE 신호의 상승에지(Rising Edge)가 새로운 카운트 시작을 트리거하고, 동시에 이전의 카운트값을 OLD COUNT 에 래치(Latch)합니다. 이 모드에서는 GATE 신호가 LOW 상태일 때에도 카운트는 계속됩니다.



◈ void COMI_SD502_SetCounter (HANDLE hDevice, int ch, int nMode, int nClkSource)

이 함수는 지정한 카운터 채널을 셋팅합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *nMode* : 카운터 모드를 지정합니다. 이 값은 0 ~ 2 사이의 값이어야 하며 각 모드에 대한 설명은 앞 장을 참조하십시오.
- *nClkSource* : 소스클럭(Source clock)을 지정합니다. 소스클럭은 외부에서 입력하거나 내부의 클럭을 사용할 수 있습니다. *nClkSource* 에 지정할 수 있는 값은 -1 또는 0 ~ 15 사이의 값입니다. 각 값의 의미는 다음과 같습니다.

값	소스클럭(Source Clock)	값	소스클럭(Source Clock)
-1	External Clock	8	19.531 KHz Internal Clock
0	5000.0 KHz Internal Clock	9	9.7656 KHz Internal Clock
1	2500.0 KHz Internal Clock	10	4.8828 KHz Internal Clock
2	1250.0 KHz Internal Clock	11	2.4414 KHz Internal Clock
3	625.00 KHz Internal Clock	12	1.2207 KHz Internal Clock
4	312.50 KHz Internal Clock	13	0.6104 KHz Internal Clock
5	156.30 KHz Internal Clock	14	0.3052 KHz Internal Clock
6	78.125 KHz Internal Clock	15	0.1526 KHz Internal Clock
7	39.063 KHz Internal Clock		

◈ ULONG COMI_SD502_ReadNowCount (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 현재 카운트 값을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *Return* : 현재 카운트(Now Count)값을 반환합니다.

◈ ULONG COMI_SD502_ReadOldCount (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 Old Count 값(GATE 신호의 Falling Edge 에서 Latch 저장된 값, 141~142 페이지 참조)을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *Return* : Old Count 값을 반환합니다.

◈ BOOL COMI_SD502_GetGateState (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 GATE 신호의 상태를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *Return* : 0 => GATE 신호가 LOW 상태임을 의미
  1 => GATE 신호가 HIGH 상태임을 의미

◈ double COMI_SD502_GetClkFreq (int nClkSrcIdx)

이 함수는 지정한 카운터 채널의 소스클럭을 내부클럭으로 설정한 경우에 내부클럭의 주파수를 반환합니다.

- *nClkSrcIdx* : 내부클럭의 인덱스(Index)값. 이 값은 COMI_SD502_SetCounter(..) 함수에서 사용되는 *nClkSource* 값과 동일한 값이어야 합니다.
- *Return* : 내부 소스클럭의 주파수 (Hz)
- *Remarks* : 소스클럭 인덱스와 주파수의 관계는 COMI_SD502_SetCounter() 함수 설명 참조하십시오.

◈ void COMI_SD502_Clear (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 New Count 와 Old Count 값을 0 으로 초기화 합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *Remarks* : 카운터 모드를 0 으로 한 경우에는 New Count 값은 GATE 신호의 Falling Edge 에 의하여 자동으로 Clear 됩니다.

◈ void COMI_SD502_ClearMulti (HANDLE hDevice, ULONG dwCtrlBits)

이 함수는 여러 채널을 동시에 Clear 해주는 함수입니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *dwCtrlBits* : 이 값의 하위 10 비트가 순서에 따라 각 채널의 Clear 를 제어합니다. 비트값이 1 이면 해당 채널은 Clear 됩니다. BIT0 가 Channel 0 에 해당하며 BIT9 가 Channel 9 에 해당합니다.
- *Remarks* : 카운터 모드를 0 으로 한 경우에는 New Count 값은 GATE 신호의 Falling Edge 에 의하여 자동으로 Clear 됩니다.

[리스트 3-19] COMI-SD502 Counter 함수 사용 예

```
/*********************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : SD502 Counter 함수 사용 예
/* Contents: 이 프로그램은 사용자가 지정하는 channel 과 mode 그리고
/*    source clock 으로 counter 를 셋팅한 후 지정한 채널의 카운트값을
/*    읽어서 화면에 표시합니다.
/* - Remarks :
/*   1. 이 예제는 COMI-SD502 보드에만 적용가능합니다.
/*********************************************************************/

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

#define DEV_ID  COMI_SD502

void main (void)
{
  HANDLE hDevice;
  int channel, mode, clk_src;
  ULONG now_count, old_count;
  double clk_freq, freq;
  char resp;
  // Load DLL //
  if(!COMI_LoadDll()){
    printf("Comidas.dll load failure");
    printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
    _getch();
    exit(0);
  }
  // Load Device //
  hDevice = COMI_LoadDevice (DEV_ID, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    printf("아무키나 누르면 프로그램이 종료됩니다.. \n");
    _getch();
    COMI_UnloadDll();
    exit(0);
  }
  // Get channel number //
  printf("Input Channel Number : ");
  scanf("%d", &channel);
  // Get counter mode //
```

```
  printf("Select Counter Mode (0 or 1) : ");
  scanf("%d", &mode);
  // Get source clock (only if counter mode is 0) //
  if(mode == 0){
    printf("Select Source Clock (-1 ~ 15) : ");
    scanf("%d", &clk_src);
  }
  // Set counter channel //
  COMI_SD502_SetCounter(hDevice, channel, mode, clk_src);
  COMI_SD502_Clear(hDevice, channel);
  while(1)
  {
    now_count = COMI_SD502_ReadNowCount(hDevice, channel); // Read now
count
    old_count = COMI_SD502_ReadOldCount(hDevice, channel); // Read old
count
    printf("\n");
    printf("Now Count = %lu\n", now_count);
    printf("Old Count = %lu\n", old_count);
    if(mode==0 && clk_src >= 0){// Display frequency of GATE signal
      clk_freq = COMI_SD502_GetClkFreq(clk_src);
      if(old_count == 0)
        freq = 0.;
      else
        freq = 0.5 * clk_freq/old_count;
      printf("Freq (Hz) = %.1f\n", freq);
    }
    printf("\nPress ESC key to exit or press any other key to
continue!\n");
    resp = _getch();
    if(resp == 27) // ESC Key가 눌리면 종료
      break;
  }

  COMI_UnloadDevice(hDevice);
  COMI_UnloadDll();
}
```

3-7 인터럽트

이 단원은 COMI-SD434 Digital I/O Board 에만 지원되는 기능입니다. 인터럽트는 특정 상황이 발생되었을 때 사용자(또는 프로그래머)에게 이것을 알려주기 위한 것입니다. 인터럽트는 폴링(Polling) 방식과 달리 CPU 에 부하를 주지 않는 것이 장점입니다. 윈도우 환경에서는 일반 Application 레벨에서 인터럽트를 처리할 수 없으므로 이벤트를 통하여 Application 에게 인터럽트가 발생하였음을 알려줍니다. 인터럽트와 관련된 함수들은 다음과 같습니다.

- long COMI_INT_Start (HANDLE hDevice)
- long COMI_INT_Stop (HANDLE hDevice)
- long COMI_INT_Clear (HANDLE hDevice)
- long COMI_INT_SetIntChan (HANDLE hDevice, int Numch, int nState)
- long COMI_INT_GetIntChan (HANDLE hDevice, int Numch)
- long COMI_INT_GetIntState (HANDLE hDevice)
- long COMI_INT_SetHandler (HANDLE hDevice, long HandlerType, UINT nMessage, LPVOID lParam)

◈ long COMI_INT_Start (HANDLE hDevice)

이 함수는 인터럽트 기능을 사용가능하도록 Enable 해줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 함수의 수행 결과를 반환합니다.
  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_Stop (HANDLE hDevice)

이 함수는 인터럽트 기능을 사용하지 않도록 Disable 해줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 함수의 수행 결과를 반환합니다.
  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_GetIntState (HANDLE hDevice)

이 함수는 인터럽트의 현재 동작 상태를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 현재 인터럽트 동작 상태를 반환합니다.
  0 - 사용 안함, 1 - 사용.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_Clear (HANDLE hDevice)

이 함수는 인터럽트의 현재 상태를 초기화 하여줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 함수의 수행 결과를 반환합니다.

0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_SetIntChan (HANDLE hDevice, int NumCh, int nState)

이 함수는 각 채널별로 인터럽트 사용여부를 설정합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *NumCh* : 인터럽트 상태를 감지할 채널 번호입니다. COMI-SD43 에서는 베이스채널인 Digital I/O 16 채널에 대하여 인터럽트 기능을 사용할 수 있습니다.
- *nState* : 채널의 인터럽트 기능의 사용 여부를 설정합니다.
  0 - 사용 안함, 1 - 사용.
- *Return* : 함수의 수행 결과를 반환합니다.
  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_GetIntChan (HANDLE hDevice, int NumCh)

이 함수는 해당 채널의 인터럽트 사용여부를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *NumCh* : 인터럽트 사용여부를 확인할 채널 번호입니다. COMI-SD43 에서는 베이스채널인 Digital I/O 16 채널에 대하여 인터럽트 기능을 사용할 수 있습니다.
- *Return* : 해당 채널의 인터럽트 사용여부를 반환합니다.
  0 - 사용 안함, 1 - 사용.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_SetHandler (HANDLE hDevice, long Type, HANDLE Handler, UINT nMessage, LPVOID lParam)

이 함수는 인터럽트 이벤트 핸들러를 등록합니다. 이벤트 핸들러는 윈도우

메시지 방식, 이벤트 객체 방식, 콜백 함수 방식등의 인터럽트 처리를 수행할 수 있으며, 해당 인터럽트가 발생되었을 때, 지정된 HandlerType 에 따른 처리가 이루어지게 됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Type* : 인터럽트 핸들러의 종류를 지정합니다. 이 값은 종류는 아래와 같이 3 가지로 지정할 수 있습니다.

Value	Meaning
0(IHT_MESSAGE)	윈도우 메시지 전달방식을 통하여 인터럽트를 통지합니다. ·장점 : GUI 관련 작업을 이벤트 핸들러에서 직접 수행할 수 있다. ·단점 : 윈도우가 메시지루프 상황에 따라서 상당한 지연시간을 가질 수 있다.
1(IHT_EVENT)	이벤트 객체를 통하여 인터럽트를 통지합니다. ·장점 : 메시지 전달 방식보다 비교적 적은 지연시간을 가질 수 있다. ·단점 : 사용하기가 복잡하다.
2(IHT_CALLBACK)	콜백 함수를 통하여 인터럽트를 통지합니다. 이 방식이 가장 권장되는 방식입니다. ·장점 : 메시지 전달의 지연시간이 3 가지 방식 중에서 가장 적다. ·단점 : GUI 관련 작업을 수행할 수 없다.

- *Handler* : 이 매개변수의 의미는 Type 설정에 따라서 다음과 같이 달라집니다.

Value	Meaning
0(메시지 방식) 일때	Handler 매개변수는 윈도우 핸들을 의미합니다.
1(이벤트 방식) 일때	Handler 매개변수는 이벤트 객체를 의미합니다.
2(콜백 방식) 일때	Handler 매개변수는 콜백 함수를 의미합니다.

- *uMessage* : 이 매개변수는 Type 이 IHT_MESSAGE 로 설정되었을 때만 유효한 것으로서 윈도우 메시지 번호를 설정합니다.
- *lParam* : 인터럽트가 처리되는 함수에 전달될 핸들값을 설정합니다.
- *Return* : 함수의 수행 결과를 반환합니다.

  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

[리스트 3-20] Interrupt 함수 사용 예

```
/*********************************************************************
/* [COMIDAS sample code by COMIZOA Inc., Ltd]
/*
/* Subject : Interrupt
/* Contents: 이 프로그램은 인터럽트 감지를 시작하고 난 뒤부터 인터럽트가 발생되었을 때
/*     사용자에게 알려주는 예제 코드입니다.
/*
/* - Remarks :
/*   1. 이 예제는 인터럽트 기능을 지원하는 COMI-CP434 보드를 기준으로 작성되었습니다.
/*
/*********************************************************************/
#include "stdafx.h"
#include "Comidas.h"

#define WMU_Interrupt  (WM_APP + 1)
Void WINAPI InterruptCallback (LPVOID lParam);

void main (void)
{
  HANDLE hDevice;

  if(!COMI_LoadDll()){
    AfxMessageBox("Comidas.dll load failure");
    exit(0);
  }

  hDevice = COMI_LoadDevice (COMI_SD434, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    AfxMessageBox("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  COMI_INT_SetHandler (hDevice, IHT_CALLBACK, InterruptCallback,
                                                WMU_Interrupt, this);
  if( COMI_INT_Start (hDevice))
    AfxMessageBox("Interrupt Activate!");
  COMI_INT_SetIntChan (hDevice, Ch0, TRUE);
  COMI_DO_PutOne (hDevice, Ch0, TRUE);
}

Void WINAPI InterruptCallback (LPVOID lParam)
{
  AfxMessageBox ("Interrupt detection!);
}
```

3-8 필터

이 단원은 COMI-SD434 Digital I/O Board 에만 지원되는 기능입니다. 디지털 입력의 외부 노이즈 및 원하지 않는 신호를 구분하여 받지 않게끔 하는 기능입니다. 모드 변경에 따라 필터되는 타이밍 간격이 조절 가능합니다. 필터와 관련된 함수들은 다음과 같습니다.

■ long COMI_SetFilterMode (HANDLE hDevice, int nMode)
■ long COMI_GetFilterMode (HANDLE hDevice)

◈ long COMI_SetFilterMode (HANDLE hDevice, int nMode)

이 함수는 필터 클럭 모드를 변경하여 필터링 되는 타이밍 간격을 설정 합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nMode*: 필터 클럭 모드를 설정 합니다. 필터를 적용하지 않는 디폴트 값은 12 MHz 입니다.

Value	Meaning
0(CLOCK_12MHz)	12 Mhz
1(CLOCK_3KHz)	3 Khz
2(CLOCK_760Hz)	760 Hz
3(CLOCK_95Hz)	95 Hz

- *Return* : 함수의 수행 결과를 반환합니다.

0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_GetFilterMode (HANDLE hDevice)

이 함수는 설정한 필터 클럭 모드를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 설정한 필터 클럭 모드를 반환합니다.

Value	Meaning
0(CLOCK_12MHz)	12 Mhz
1(CLOCK_3KHz)	3 Khz
2(CLOCK_760Hz)	760 Hz
3(CLOCK_95Hz)	95 Hz

☞ **지원 디바이스 :** COMI-SD434.

[리스트 3-21] Filter 함수 사용 예

```
/*********************************************************************
/* [COMIDAS sample code by COMIZOA Inc., Ltd]
/*
/* Subject : Filter
/* Contents: 이 코드는 사용자가 디지털 필터 값을 다른 값으로 변경을 하도록 하는 예제
/*    코드 입니다.
/* - Remarks :
/*   1. 이 예제는 인터럽트 기능을 지원하는 COMI-CP434 보드를 기준으로 작성되었습니다.
/*********************************************************************/

#include "stdafx.h"
#include "Comidas.h"

void main (void)
{
  HANDLE hDevice;

  if(!COMI_LoadDll()){
    AfxMessageBox("Comidas.dll load failure");
    exit(0);
  }

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    AfxMessageBox("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  COMI_SetFilterMode (hDevice, CLOCK_12MHz);

  If(COMI_GetFilterMode (hDevice) == CLOCK_12MHz)
    AfxMessageBox ("12MHz Digital Filter Mode!")
}
```

CHAPTER 4
DOS C/C++ 라이브러리

CHAPTER 4. DOS C/C++ 라이브러리

본 장에서는 COMIDAS 디바이스의 도스용 C/C++ 라이브러리를 소개합니다. COMIDAS 디바이스의 도스용 C/C++ 라이브러리는 Comidas.c 와 Comidas.h 파일로 이루어져 있습니다.

도스에서 COMIDAS 라이브러리를 사용하기 위해서는 다음과 같은 절차를 거친 후 일반 함수 사용하듯 필요한 함수를 호출하면 됩니다.

1. 컴파일러의 Option=>Directories 메뉴를 선택한 후 Include Directories 항목에 COMIDAS_ROOT₩Dos₩Lib 폴더를 추가합니다. 여기서 COMIDAS_ROOT 는 COMIDAS 파일들이 설치된 루트 디렉토리를 가리킵니다.
2. 사용하고자 하는 소스 파일에 #include “Comidas.c” 구문을 추가합니다.

※ 본 COMIDAS 도스용 라이브러리에서는 윈도우용 라이브러리와의 호환성을 위하여 일부 데이터형을 정의하여 사용하고 있습니다. 윈도우 프로그램에 익숙하지 않은 사용자께서는 이러한 데이터형이 생소할 수도 있는데 이들은 다음과 같이 기존의 데이터형을 편의에 따라 재정의 한것입니다.

- ▷ UCHAR, BYTE, BOOL - unsigned char
- ▷ USHORT - unsigned short int
- ▷ UINT, WORD - unsigned int
- ▷ ULONG, DWORD - unsigned long

4-1 디바이스 시작/종료 함수

이 단원에서는 각 디바이스를 Load/Unload 하는 함수들을 소개합니다. 이 함수들은 COMIDAS 라이브러리를 사용하기 위해 필수적으로 적용되어져야할 함수들입니다. 윈도우와 달리 도스에서는 COMI_LoadDll()과 COMI_UnloadDll() 함수가 필요치 않습니다. 디바이스 시작/종료 함수 리스트는 다음과 같습니다.

■ HANDLE COMI_LoadDevice (COMIDAS_DEVID deviceID, ULONG instance)

■ void COMI_UnloadDevice (HANDLE hDevice)

함수명	보드별 지원 여부										
	CP101	CP201	CP301	CP401	CP501	SD10x	SD201	SD301	SD4xx	SD501	SD502
COMI_LoadDevice	V	V	V	V	V	V	V	V	V	V	V
COMI_UnloadDevice	V	V	V	V	V	V	V	V	V	V	V

[표 4-1] 디바이스 시작/종료 함수 리스트 및 각 보드별 지원 여부

◈ HANDLE COMI_LoadDevice (WORD deviceID, WORD instance)

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

- *deviceID* : 각 디바이스의 고유한 아이디값입니다. 이 값은 각 디바이스 명칭과 동일하게 적어주면 됩니다. 예를 들어 COMI-CP101 Multi-function board 의 경우 COMI_CP101 을 적어주면 된다. 각 디바이스 아이디는 Comidas.h 헤더파일의 앞부분에 정의되어 있으므로 Comidas.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, 두번째 장착된 보드의 instance 값은 1 이 됩니다.
- *Return* : 이 함수는 디바이스 핸들을 반환합니다. 이 값은 각 디바이스를 제어하는 함수의 첫번째 파라미터로 사용됩니다. 만일 이 값이 INVALID_HANDLE_VALUE 이면 디바이스 로딩이 실패한 것입니다.
- *Remarks* : 이 함수는 제어하고자 하는 디바이스 수만큼 수행되어야 합니다.

☞ **지원 디바이스 :** All devices

◈ void COMI_UnloadDevice (HANDLE hDevice)

이 함수는 하나의 COMIDAS 디바이스를 언로드(unload)한다. Device 가 언로드되지 않으면 메모리 손실(Loss)이 발생할 수 있다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값입니다.

☞ **지원 디바이스 :** All devices

[리스트 4-1] 디바이스 시작/종료 함수 사용예 1

```
/*********************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Deivce loading/unloading
/* - Contents: 이프로그램은 COMI-CP101 보드 하나를 사용할 때 시작과 종료시 필요한 절차를
/*     보여주는  예제 입니다.
/*********************************************************************/

#include <stdio.h>
#include <conio.h>
#include "Comidas.c"

void main (void)
{
  HANDLE hDevice;

  //------ 프로그램 시작시 수행해야할 루틴 ------//
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }
  //-------------------------------------//

  // 필요한 디바이스 제어 루틴을 구현한다. //
  ……
  ……

  //------ 프로그램 종료시 수행해야할 루틴 ------//
  COMI_UnloadDevice(hDevice);
  //-------------------------------------//
}
```

[리스트 4-2] 디바이스 시작/종료 함수 사용예 2

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Deivce loading/unloading
/* - Contents: 이프로그램은 COMI-CP101 보드 2 개와 COMI-CP301 보드 하나를 사용할 때
/*      시작과 종료시 필요한 절차를 보여주는 예제 입니다.
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "Comidas.C"

void main (void)
{
  HANDLE hDevice[3];

  //------ 프로그램 시작시 수행해야할 루틴 ------//
  hDevice[0] = COMI_LoadDevice (COMI_CP101, 0);
  hDevice[1] = COMI_LoadDevice (COMI_CP101, 1);
  hDevice[2] = COMI_LoadDevice (COMI_CP301, 0);
  // 여기에서 Loading 에러처리를 해야하나 생략한다. //
  //-------------------------------------//

  // 필요한 디바이스 제어 루틴을 구현한다. //
  ……
  ……

  //------ 프로그램 종료시 수행해야할 루틴 ------//
  COMI_UnloadDevice(hDevice[0]);
  COMI_UnloadDevice(hDevice[1]);
  COMI_UnloadDevice(hDevice[2]);
  //-------------------------------------//

}
```

4-2 아날로그 입력

이 장에서는 A/D 에 관련된 함수들을 소개합니다. A/D 는 아날로그(Analog) 신호를 입력받아 디지털(Digital)값으로 변환해주는 기능입니다. 도스 라이브러리에서는 A/D Scan 방식을 지원하지 않습니다. Analog Input 과 관련된 함수들의 리스트는 다음과 같습니다.

- BOOL COMI_AD_SetRange (HANDLE hDevice, int ch, float vmin, float vmax)
- float COMI_DigitToVolt (short digit, float vmin, float vmax)
- float COMI_Digit16ToVolt (short digit, float vmin, float vmax)
- short COMI_AD_GetDigit (HANDLE hDevice, int ch)
- float COMI_AD_GetVolt (HANDLE hDevice, int ch)

함수명	보드별 지원 여부										
	CP101	CP201	CP301	CP401	CP501	SD10x	SD201	SD301	SD4xx	SD501	SD502
COMI_AD_SetRange	V	V				V	V				
COMI_DigitToVolt	V	V				V					
COMI_Digit16ToVolt							V				
COMI_AD_GetDigit	V	V				V	V				
COMI_AD_GetVolt	V	V				V	V				

[표 4-2] 아날로그 입력 함수 리스트 및 각 보드별 지원 여부

◈ BOOL COMI_AD_SetRange (HANDLE hDevice, int ch, float vmin, float vmax)

이 함수는 각 A/D 채널의 입력 범위를 정해줍니다.

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_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.

  CP 시리즈 보드 ~ -1, -2, -5, -10

  SD/LX 시리즈 보드 ~ 0, -1, -2, -5, -10
- *vmax* : A/D 범위의 최대값을 지정합니다. 유효한 vmax 값은 1, 2, 5, 10 입니다.
- *Return* : 1 => 성공

  0 => 실패

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ int COMI_AD_GetDigit (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야합니다.
- *ch* : A/D 를 수행할 채널 번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 정수형의 A/D 결과값. 12 Bit resolution 디바이스(COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103)는 0 ~ 4095 사이의 값을 가지며, 16 Bit resolution 디바이스(COMI-SD201)는 -32768 ~ 32767 사이의 값을 갖는다.

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ float COMI_AD_GetVolt (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야합니다.
- *ch* : A/D 를 수행할 채널 번호. 채널 번호는 0 부터 시작합니다.
- *Return* : A/D 결과값 (voltage)

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ float COMI_DigitToVolt (short digit, float vmin, float vmax)

이 함수는 12 Bit resolution 의 A/D 디바이스(COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103)의 정수형 A/D 변환값(0 ~ 4095)을 voltage 값으로 환산해주는 함수입니다.

- *digit* : 변환하고자 하는 digit 값을 지정합니다. 이 값은 0~4095 사이의 값이어야 합니다.
- *vmin* : A/D 범위의 최소값을 지정합니다.
- *vmax* : A/D 범위의 최대값을 지정합니다.
- *Return* : digit 값을 voltage 값으로 환산한 값.
- *Remarks* : digit 값을 voltage 값으로 환산하는 것은 다음과 같은 식에 의해서 이루어집니다.

  Voltage = digit*(vmax - vmin)/4095 + vmin

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104

◈ float COMI_Digit16ToVolt (short digit, float vmin, float vmax)

이 함수는 16 Bit resolution 의 A/D 디바이스(COMI-SD201)의 정수형 A/D 변환값(-32768 ~ 32767)을 voltage 값으로 환산해주는 함수입니다.

- *digit* : 변환하고자 하는 -32768 ~ 32767 사이의 digit 값.
- *vmin* : A/D 범위의 최소값을 지정합니다.
- *vmax* : A/D 범위의 최대값을 지정합니다.
- *Return* : digit 값을 voltage 값으로 환산한 값.
- *Remarks* : digit 값을 voltage 값으로 환산하는 것은 다음과 같은 식에 의해서 이루어집니다.

  Voltage = (digit+32768)*(vmax - vmin)/65535 + vmin

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP201, COMI-SD101, COMI-SD102, COMI-SD103

[리스트 4-3] Single point A/D 사용예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Single point A/D
/* - Contents: 이프로그램은 COMI_AD_GetVolt(...)함수를 사용하여 Single point
/*      A/D를 수행하는 프로그램입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID로 바꾸어야 합니다.
/*************************************************************************/
#include <stdio.h>
#include <conio.h>
#include "Comidas.c"
#define CHAN  0
#define VMIN  -10
#define VMAX  10

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

  /* If you use other than COMI-CP101 board, you must replace */
  /* the first parameter of COMI_LoadDevice() function with a  */
  /* proper device ID.                          */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }
  /* Setting the A/D range of the specified channel */
  COMI_AD_SetRange (hDevice, CHAN, VMIN, VMAX);
  while (!kbhit())
  {
    ad_volt = COMI_AD_GetVolt(hDevice, 0);
    gotoxy (10, 10);
    printf("A/D volt = %-6.2f\n", ad_volt);
    delay(500);
  }

  COMI_UnloadDevice(hDevice);
}
```

4-3 디지털 입출력

- void COMI_DIO_SetUsage (HANDLE hDevice, int usage)
- int COMI_DI_GetOne (HANDLE hDevice, int ch)
- DWORD COMI_DI_GetAll (HANDLE hDevice)
- DWORD COMI_DI_GetAllEx (HANDLE hDevice, int nGroupIdx)
- BOOL COMI_DO_PutOne (HANDLE hDevice, int ch, int status)
- BOOL COMI_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)
- BOOL COMI_DO_PutAllEx (HANDLE hDevice, int nGroupIdx, DWORD dwStatuses)

함수명	보드별 지원 여부							
	CP101/201/301	CP401	SD10x/201/501	SD301	SD402	SD403	SD4x4	SD502
COMI_DIO_SetUsage		V		V		V		V
COMI_DI_GetOne	V	V	V	V		V	V	V
COMI_DI_GetAll	V	V	V	V		V	V	V
COMI_DI_GetAllEx	V	V	V	V		V	V	V
COMI_DO_PutOne	V	V	V	V	V		V	V
COMI_DO_PutAll	V	V	V	V	V		V	V
COMI_DO_PutAllEx	V	V	V	V	V		V	V

[표 4-3] 디지털 입출력 함수 리스트 및 각 보드별 지원 여부

◈ void COMI_DIO_SetUsage (HANDLE hDevice, int usage)

이 함수는 디지털 입출력 채널의 용도를 설정하는 함수입니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *usage* : 디바이스의 용도를 선택합니다. 이 값은 다음의 세 값 중 하나이어야 합니다.
  - ▷ **DI_ONLY** => DIO 채널 모두를 Digital Input 채널로 사용.
  - ▷ **DO_DI** => DIO 채널의 전반부를 Output 으로 후반부를 Input 으로 사용.
  - ▷ **DI_DO** => DIO 채널의 전반부를 Input 으로 후반부를 Output 으로 사용.
  - ▷ **DO_ONLY** => DIO 채널 모두를 Digital Output 채널로 사용.
- *Remarks* : 모든 제품이 디지털 입출력 채널의 용도를 소프트웨어적으로 설정할 수 있는 것은 아닙니다. 이 함수를 지원하는 제품과 각 설정값에 따른 채널 구분은 다음과 같습니다.

<table>
<tr><th rowspan="2">제품명</th><th rowspan="2">채널
구분</th><th colspan="4">용도 설정값</th></tr>
<tr><th>DI_ONLY</th><th>DO_DI</th><th>DI_DO</th><th>DO_ONLY</th></tr>
<tr><td rowspan="2">COMI-CP401</td><td>Input</td><td>CH0~CH15</td><td>CH8~CH15</td><td>CH0~CH7</td><td>NONE</td></tr>
<tr><td>Output</td><td>NONE</td><td>CH0~CH7</td><td>CH8~CH15</td><td>CH0~CH15</td></tr>
<tr><td rowspan="2">COMI-SD301</td><td>Input</td><td>CH0~CH31</td><td>CH16~CH31</td><td rowspan="2">지원안함</td><td>NONE</td></tr>
<tr><td>Output</td><td>NONE</td><td>CH0~CH15</td><td>CH0~CH31</td></tr>
<tr><td rowspan="2">COMI-SD502</td><td>Input</td><td>CH0~CH2</td><td rowspan="2">지원안함</td><td rowspan="2">지원안함</td><td>NONE</td></tr>
<tr><td>Output</td><td>NONE</td><td>CH0~CH2</td></tr>
</table>

◈ int COMI_DI_GetOne (HANDLE hDevice, int ch)

이 함수는 지정한 Digital Input 채널의 Status 를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Digital Input 채널번호. 채널번호는 0 부터 시작합니다(앞의 COMI_DIO_SetUsage() 함수 참조).
- *Return* : Digital Input 채널의 Status. 0 - OFF, 1 - ON.

☞ **지원 디바이스** : All devices.

◈ DWORD COMI_DI_GetAll (HANDLE hDevice)

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

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : CH0 ~ CH31 의 32 채널의 Digital Input 채널의 Status 를 나타내는 32 bit 값. 이 값의 각 비트의 값이 각 채널의 Status 를 나타냅니다.

☞ **지원 디바이스** : All devices.

◈ DWORD COMI_DI_GetAllEx (HANDLE hDevice, int nGroupIdx)

이 함수는 지정한 디지털 입력 디바이스로부터 32 개 채널의 Status 를 32 비트값으로 반환합니다. 이 함수는 32 채널보다 많은 디지털 입력 채널을 제공하는 장치에서 32 채널씩 데이터를 읽어들일 수 있도록 하기 위한 함수로써, 이때의 채널은 nGroupIndex 파라미터에 따라 달라집니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nGroupIdx* : 32 채널 단위의 채널 그룹인덱스를 지정합니다. 예를 들어 이 값이 0 이면 CH0~CH31 의 Status 를, 1 이면 CH32~CH63 의 Status 를 반환하도록 합니다.
- *Return* : 32 개의 채널에 대한 Input Status 를 32 비트 값으로 반환합니다. 각비트는 비트 순서와 일치하여 각 채널의 ON/OFF 상태를 나타냅니다.

☞ **지원 디바이스** : All devices.

◈ BOOL COMI_DO_PutOne (HANDLE hDevice, int ch, int status)

이 함수는 지정한 Digital Output 채널에 지정한 Status 로 출력을 내보낸다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Digital Output 채널번호. 채널 번호는 0 부터 시작한다.
- *status* : 출력 Status. 0 - OFF, 1 - ON.

- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스** : All devices.

◈ BOOL COMI_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)

해당 디바이스의 모든 Digital Output 채널에 출력을 내보낸다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *dwStatuses* : CH0~CH31 의 32 채널의 Digital Output 출력 Status 를 나타내는 32 bit 값. 이 값의 각 비트의 값이 각 채널의 Status 를 나타냅니다.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스** : All devices.

◈ BOOL COMI_DO_PutAllEx (HANDLE hDevice, int nGroupIdx, DWORD dwStatuses)

이 함수는 32 개 Digital Output 채널에 출력을 내보냅니다. 이 함수는 32 채널보다 많은 디지털 출력 채널을 제공하는 장치에서 32 채널씩 출력을 제어할 수 있도록 하기 위한 함수로써, 이때의 채널은 nGroupIndex 파라미터에 따라 달라집니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nGroupIdx*: 32 채널 단위의 채널 그룹인덱스를 지정합니다. 예를 들어 이 값이 0 이면 CH0~CH31 에 출력을 내보내고, 1 이면 CH32~CH63 에 출력을 내보내도록 합니다.
- *dwStatuses* : 32 채널의 Digital Output 채널의 출력 Status 를 나타내는 32 bit 값. 이 값의 각 비트의 값이 각 채널의 Status 를 나타내며, 이때의 해당 채널그룹은 nGroupIndex 파라미터에 의해 결정됩니다.
- *Return* : 1 => 성공, 0 => 실패

☞ **지원 디바이스** : All devices.

[리스트 4-4] Single Channel Digital Input/Output

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : D/I and D/O
/* - Contents: 이 프로그램은 COMI_DO_PutOne(..)과 COMI_DI_GetOne(..) 함수를
/*    사용하여 각 1 채널에 대한 D/I 와 D/O 를 수행하는 예제입니다.
/* - Remarks :
/*    1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*       사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*       ID 로 바꾸어야 합니다.
/*    2. 이 프로그램은 D/O 채널 0 번을 통하여 On/Off 신호를 번갈아가면서 내보내고
/*      D/I 채널 0 번을 읽어들이는 프로그램입니다. D/O 0 번 채널과 D/I 0 번 채널을
/*       서로 연결해놓으면 D/O 에서 내보낸 신호를 D/I 채널에서 읽어들일수 있습니다.
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "Comidas.c"

#define DO_CH 0
#define DI_CH 0

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

  /* If you use other than COMI-CP101 board, you must replace */
  /* the first parameter of COMI_LoadDevice() function with a  */
  /* proper device ID.                                         */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }

  /* COMI-CP401 보드의 경우에는 DI/DO 기능을 모두 사용하기 위해 */
  /* 여기서 COMI_DIO_SetUsage(DO_DI) 구문을 써주어야 합니다. */

  clrscr();
  while(!kbhit())
  {
    do_state ^= 1; /* toggle the state of D/O channel */
    COMI_DO_PutOne (hDevice, DO_CH, do_state); // Put D/O
```

```
    /* Get D/I and print on screen */
    di_state = COMI_DI_GetOne(hDevice, DI_CH);
    gotoxy (10, 10);
    printf("Status of D/I CH%d = %d\n", DI_CH, di_state);
    delay(500);
  }

  COMI_UnloadDevice(hDevice);
}
```

[리스트 4-5] Multi Channel Digital Input/Output

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : D/I and D/O
/* - Contents: 이 프로그램은 COMI_DI_GetAll(..)과 COMI_DO_PutAll(..) 함수를
/*   사용하여 D/I 와 D/O 의 8 채널을 동시에 콘틀롤하는 예제입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*      사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*      ID 로 바꾸어야 합니다.
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "Comidas.c"

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

  /* If you use other than COMI-CP101 board, you must replace */
  /* the first parameter of COMI_LoadDevice() function with a  */
  /* proper device ID.                                         */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }

  /* COMI-CP401 보드의 경우에는 DI/DO 기능을 모두 사용하기 위해 */
  /* 여기서 COMI_DIO_SetUsage(DO_DI) 구문을 써주어야  합니다.  */

  clrscr();
  while(!kbhit())
  {
    do_states = ~do_states; /* toggle the states of D/O channels */
    COMI_DO_PutAll (hDevice, do_states); /* Put D/O for all channels */

    /* Get D/I and print on screen */
    di_states = COMI_DI_GetAll(hDevice);
    /* di_states contains the states of all channels      */
    /* If you want to get the state of each channel, perform */
```

```
    /* a bit mask like following codes              */
    for(i=0; i<8; i++)
      di_each[i] = (di_states >> i) & 0x1;
    gotoxy (10, 10);
    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]);
    delay(500);
  }

  COMI_UnloadDevice(hDevice);
}
```

[리스트 4-5-2] 64 채널 디지털 입력 예제

```
/*****************************************************************
* [COMIDAS sample program by COMIZOA Inc., Ltd]
*
* - Subject : 64 비트 Digital Input
* - Contents: 이 프로그램은 COMI_DI_GetAllEx() 함수를 사용하는 예제로써, 32 채널씩 나누
* 어서 총 64 채널의 디지털 입력값을 읽어들여 화면에 HEXA-DECIMAL 값으로 그 상태를 표시합니다.
*****************************************************************/

  #include <stdio.h>
  #include <conio.h>
  #include "Comidas.c"

  #define DEV_ID  COMI_SD403

  void main (void)
  {
    HANDLE hDevice;
    DWORD di_states[2];
    int i, di_each[8];

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

    clrscr();
    while(!kbhit())
    {
      /* Read first 32 D/I channels (CH0~CH31) */
      di_states[0] = COMI_DI_GetAllEx(hDevice, 0);
      /* Read second 32 D/I channels (CH32~CH63) */
      di_states[1] = COMI_DI_GetAllEx(hDevice, 1);
      gotoxy (10,10);
      printf("D/I States(CH0~CH31) = %08lx", di_states[0]);
      gotoxy (10,11);
      printf("D/I States(CH31~CH63) = %08lx\n", di_states[1]);
    }

    COMI_UnloadDevice(hDevice);
  }
```

[리스트 4-5-3] 64 채널 디지털 출력 예제

```
/*****************************************************************
* [COMIDAS sample program by COMIZOA Inc., Ltd]
*
* - Subject : 64 비트 Digital Output
* - Contents: 이 프로그램은 COMI_DO_PutAllEx() 함수를 사용하는 예제로써 32 채널씩 나누
*   어서 총 64 채널의 디지털 출력을 내보냅니다. 이 때의 출력 상태는 전 채널 모두 on/off 를 반
*   복하여 출력됩니다.
*****************************************************************/

  #include <stdio.h>
  #include <conio.h>
  #include "Comidas.c"

  #define DEV_ID  COMI_SD402

  void main (void)
  {
    HANDLE hDevice;
    DWORD do_states[2]={0x0, 0x0};

    hDevice = COMI_LoadDevice (DEV_ID, 0);
    if(hDevice == INVALID_HANDLE_VALUE){
      printf("Can't load specified device!");
      exit(0);
    }
    clrscr();
    while(!kbhit())
    {
      do_states[0]=~do_states[0]; /* toggle the states of D/O channels */
      COMI_DO_PutAllEx(hDevice,  0,  do_states[0]);/*  Put  first  32  D/O
channels (CH0~CH31) */

      do_states[1]=~do_states[1]; /* toggle the states of D/O channels */
      COMI_DO_PutAllEx(hDevice,  1,  do_states[1]);  /*  Put  second  32  D/O
channels (CH32~CH63) */

      gotoxy (10,10);
      printf("D/O States(CH0~CH31) = %08lx", do_states[0]);
      gotoxy (10,11);
      printf("D/O States(CH31~CH63) = %08lx\n", do_states[1]);

      delay(500);
    }
    COMI_UnloadDevice(hDevice);
  }
```

4-4 아날로그 출력

◈ BOOL COMI_DA_Out (HANDLE hDevice, int ch, float volt)

이 함수는 지정한 Analog Output 채널에 지정한 Voltage를 출력합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Analog Output 채널번호. 채널 번호는 0부터 시작합니다.
- *volt* : Analog Output 출력 Voltage.
- *Return* : 1 => 성공
  0 => 실패

☞ **지원 디바이스 :**

COMI-CP101, COMI-CP301, COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD301

[리스트 4-6] Single Point Analog Output 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Single Point D/A
/* - Contents: 이프로그램은 COMI_DA_Out(...)함수를 사용하여 Single point D/A를
/*     수행하는 프로그램입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*     사용하는 경우에는 COMI_LoadDevice(...)함수의  첫 번째 파라미터를 알맞은 디바이스
/*     ID로 바꾸어야 합니다.
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "Comidas.c"

#define DA_CH 0

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

  /* If you use other than COMI-CP101 board, you must replace */
  /* the first parameter of COMI_LoadDevice() function with a  */
  /* proper device ID.                                         */
  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }

  clrscr();
  while(!kbhit())
  {
    printf("D/A Out 5 volt !\n");
    COMI_DA_Out  (hDevice,  DA_CH,  5.f);  /*  Output  5  volt  through  D/A
channel 0 */
    delay(1000); /* 1 sec delay */
    printf("D/A Out 0 volt !\n");
    COMI_DA_Out  (hDevice,  DA_CH,  0.f);  /*  Output  0  volt  through  D/A
channel 0 */
    delay(1000); /* 1 sec delay */
```

```
    }

    COMI_UnloadDevice(hDevice);
}
```

4-5 카운터

카운터는 펄스 신호를 카운트하거나 펄스 신호를 출력하는데 사용되는 기능입니다. COMIDAS 는 다양한 형태의 카운터 기능을 제공합니다. 카운터와 관련된 함수들은 다음과 같습니다.

1) 8254 카운터 함수

- void COMI_SetCounter(HANDLE hDevice, int ch, int rw, int op, int bcd_bin, USHORT load_value)
- void COMI_LoadCount(HANDLE hDevice, int ch, USHORT load_value)
- USHORT COMI_ReadCount(HANDLE hDevice, int ch)

2) 32 비트 COMI-SD 카운터 함수

- ULONG COMI_ReadCounter32(HANDLE hDevice, int ch)
- void COMI_ClearCounter32(HANDLE hDevice, int ch)

3) 엔코더 카운터 함수

- void COMI_ENC_Config(HANDLE hDevice, int ch, int mode, BOOL bResetByZ)
- void COMI_ENC_Reset(HANDLE hDevice, int ch)
- void COMI_ENC_Load(HANDLE hDevice, int ch, long count)
- long COMI_ENC_Read(HANDLE hDevice, int ch)
- void COMI_ENC_ResetZ(HANDLE hDevice, int ch)
- void COMI_ENC_LoadZ(HANDLE hDevice, int ch, short count)
- short COMI_ENC_ReadZ(HANDLE hDevice, int ch)

4) 펄스 발생기 함수

- double COMI_PG_Start(HANDLE hDevice, int ch, double freq, UINT num_pulses)
- double COMI_PG_ChangeFreq(HANDLE hDevice, int ch, double freq)
- BOOL COMI_PG_IsActive(HANDLE hDevice, int ch)
- void COMI_PG_Stop(HANDLE hDevice, int ch)

5) COMI-SD502 카운터 전용 함수

- void COMI_SD502_SetCounter(HANDLE hDevice, int ch, int nMode,

UINT nClkSource)

- ULONG COMI_SD502_ReadNowCount(HANDLE hDevice, int ch)
- ULONG COMI_SD502_ReadOldCount(HANDLE hDevice, int ch)
- BOOL COMI_SD502_GetGateState(HANDLE hDevice, int ch)
- double COMI_SD502_GetClkFreq(int nClkSrcIdx)
- void COMI_SD502_Clear(HANDLE hDevice, int ch)
- void COMI_SD502_ClearMulti(HANDLE hDevice, ULONG dwCtrlBits)

함수명	보드별 지원 여부										
	CP101	CP201	CP301	CP401	CP501	SD10x	SD201	SD301	SD4xx	SD501	SD502
COMI_SetCounter	V	V			V						
COMI_LoadCount	V	V			V						
COMI_ReadCount	V	V			V						
COMI_ReadCounter32						V	V	V		V	
COMI_ClearCounter32						V	V	V		V	
COMI_ENC_Config										V	
COMI_ENC_Reset										V	
COMI_ENC_Load										V	
COMI_ENC_Read										V	
COMI_ENC_ResetZ										V	
COMI_ENC_LoadZ										V	
COMI_ENC_ReadZ										V	
COMI_PG_Start										V	
COMI_PG_ChangeFreq										V	
COMI_PG_IsActive										V	
COMI_PG_Stop										V	
COMI_SD502_SetCounter											V
COMI_SD502_ReadNowCount											V
COMI_SD502_ReadOldCount											V
COMI_SD502_GetGateState											V
COMI_SD502_GetClkFreq											V
COMI_SD502_Clear											V
COMI_SD502_ClearMulti											V

[표 4-4] 카운터 관련 함수 리스트 및 각 보드별 지원 여부

4-5-1 8254 카운터 함수

COMI-CP101, COMI-CP201, COMI-CP501 보드에는 사용자가 사용할 수 있는 Intel 8254 카운터가 장착되어 있습니다. 8254 카운터는 펄스를 카운트하거나, 일정 주기의 펄스를 출력하는데 사용됩니다. Intel 8253/4 카운터에 대한 자세한 내용은 하드웨어 매뉴얼의 부록을 참조하기 바랍니다.

◈ void COMI_SetCounter (HANDLE hDevice, int ch, int rw, int op, int bcd_bin, UINT load_value)

이 함수는 지정한 카운터 채널의 동작 방식을 결정하고, 원하는 카운트 값을 로드(Load)합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *rw* : Read/Write mode 를 선택합니다. 이 값은 다음 중 하나의 값이어야 합니다.
  - ▷ **READ_LOAD_MSB** => 상위 1 바이트값을 로드합니다.
  - ▷ **READ_LOAD_LSB** => 하위 1 바이트값을 로드합니다.
  - ▷ **READ_LOAD_WORD** => 2 바이트값을 로드합니다.
- *op* : 카운터의 동작 모드(Operation mode)를 결정합니다. 이 값은 CMODE0 , CMODE1, CMODE2, CMODE3, CMODE4, CMODE5 중의 하나이어야 합니다. Operation mode 에 관한 자세한 내용은 하드웨어 매뉴얼의 부록을 참조하십시오.
- *bcd_bin* : 카운트를 BCD 값으로 할 것인지, BINARY 값으로 할 것인지를 결정합니다. 이 값은 BCD 나 BINARY 중의 하나이어야 합니다. 일반적으로 BINARY 값을 입력합니다.
- *load_value* : 카운터에 로드할 값을 지정합니다. 이 값은 0 ~ 65535 사이의 값이어야 합니다.

☞ **지원 디바이스** : COMI-CP101, COMI-CP201, COMI-CP501

◈ void COMI_LoadCount (HANDLE hDevice, int ch, UINT load_value)

이 함수는 지정한 카운터에 카운트값을 로드합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *load_value* : 카운터에 로드할 값을 지정합니다. 이 값은 0 ~ 65535 사이의 값이어야 합니다.

☞ **지원 디바이스 :** COMI-CP101, COMI-CP201, COMI-CP501

◈ UINT COMI_ReadCount (HANDLE hDevice, int ch)

이 함수는 지정한 카운터에서 카운트값을 읽어옵니다. 8254 카운터는 Decrease 카운팅을 합니다. 따라서 COMI_SetCounter() 또는 COMI_LoadCount()함수에서 로드(Load)한 값에서 이 함수를 통하여 읽은 카운트값을 뺀 값이 실제 카운트된 값이 됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 카운트 값

☞ **지원 디바이스 :** COMI-CP101, COMI-CP201, COMI-CP501

[리스트 4-7] Counter 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : Counter
/* Contents: 이 프로그램은 GATE0 에 5 volt 를 입력하고 ExtClk 에 일정
/*    Frequency 의 Pulse 신호를 입력하여 약 1 초동안의 Pulse 수를 반
/*    복해서 count 하는 프로그램입니다.
/* - Remarks :
/*   1. 이 예제는 COMI-CP101 보드를 사용하는 것으로 작성되었습니다. 따라서 다른 보드를
/*     사용하는 경우에는 COMI_LoadDevice(...) 함수의 첫 번째 파라미터를 알맞은 디바이스
/*     ID 로 바꾸어야 합니다.
/*************************************************************************/

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

void main (void)
{
  HANDLE hDevice;
  int status=0;
  UINT count;

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }

  clrscr ();
  COMI_SetCounter (0, READ_LOAD_WORD, CMODE0, BINARY, 65535);
  while(!kbhit())
  {
    delay(1000);
    COMI_LoadCount(0, 65535);
    count = 65535 - COMI_ReadCount(0) + 1;
    gotoxy (10, 10);
    printf("Count = %u\n", count);
  }

  COMI_UnloadDevice(hDevice);
}
```

4-5-2 32 비트 COMI-SD 카운터 함수

COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD201 보드에는 사용자가 사용할 수 있는 COMI-SD 카운터가 각각 2 채널씩 장착되어 있습니다. 이 카운터는 Increment 방식을 사용하는 32 비트 카운터입니다.

◈ void COMI_ClearCounter32 (HANDLE hDevice, int ch)

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

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.

☞ **지원 디바이스** : COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD104, COMI-SD201

◈ ULONG COMI_ReadCounter32 (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 카운트 값을 읽어옵니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 카운트 값

☞ **지원 디바이스** : COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD201

[리스트 4-8] COMI-SD 32 비트 펄스 카운터 함수 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : 32 Bit Counter - SD 시리즈 보드에만 적용 가능
/* Contents: 이 프로그램은 0 번 카운터에 펄스를 입력하고,
/*    계속해서 Pulse 수를 count 하는 프로그램입니다.
/* - Remarks :
/*  1. 이 예제는 COMI-SD101, COMI-SD102, COMI-SD103, COMI-SD201 보드에만 적용가
/*     능합니다.
/*  2. 이 예제는 COMI-SD101 보드를 사용하는 것으로 작성되었습니다.
/*     따라서 다른 보드를 사용하는 경우에는 COMI_LoadDevice(...)함수의
/*     첫 번째 파라미터를 알맞은 디바이스 ID 로 바꾸어야 합니다.
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "Comidas.c"

#define CNTR_CH 0

void main (void)
{
  HANDLE hDevice;
  ULONG count;

  hDevice = COMI_LoadDevice (COMI_SD101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }

  clrscr();
  COMI_ClearCounter32(hDevice, CNTR_CH); /* Reset counter to 0 */
  while(!kbhit())
  {
    count = COMI_ReadCounter32 (hDevice, CNTR_CH);
    gotoxy(10,10);
    printf("Read Count = %10lu", count);
  }

  COMI_UnloadDevice(hDevice);
}
```

4-5-3 엔코더 카운터

이 단원에서 설명하는 카운터 기능은 COMI-SD501 보드에서만 적용되는 카운터 기능으로써 주로 엔코더 센서를 계측하는데 사용되는 카운터 기능입니다. 일반적으로 엔코더 센서는 회전체의 위치나 속도를 계측하기 위해 사용되는 센서입니다.

엔코더에서 발생되는 신호는 A 상, B 상 그리고 Z 상으로 구분되는 3 개의 신호이며 이들은 모두 펄스 형태로 출력되게 됩니다. A 상과 B 상 신호는 회전체의 미소 위치 변화(엔코더의 분해능에 따라 그 값은 다름)가 발생할 때마다 발생되는 펄스 신호로써, 이 두 신호는 일정한 위상차를 가지게 되어 회전 방향까지도 함께 알수 있습니다. COMI-SD501 은 회전 방향에 따라 자동으로 UP/DOWN 카운트가 됩니다. Z 상 신호는 회전체가 1 회전할 때마다 발생되는 펄스 신호입니다. COMI-SD501 은 Z 상 신호가 A/B 상 카운터의 값을 자동으로 리셋되도록 설정될 수 있습니다. 이 방식을 이용하면 회전체의 절대 위치를 파악하는데 용이합니다.

COMI-SD501 보드는 엔코더의 3 가지 신호를 모두 입력받을 수 있으며, A/B 상과 Z 상을 동시에 계측할 수 있습니다. 이 단원에서 소개되는 엔코더 카운터 함수중에서 'Z' 자가 함수명 끝에 붙은 함수는 Z 상 신호를 계측하는데 사용되는 함수이며, 나머지 함수들은 모두 A/B 상 신호를 계측하는데 사용되는 함수입니다.

※ COMI-SD501 디바이스에서 일반 펄스를 카운트하는 방법

COMI-SD501 에서는 엔코더 신호뿐만 아니라 일반 펄스 신호도 카운트할 수 있습니다. 일반 펄스 신호는 A 상과 B 상이 따로 존재하지 않으므로 신호선을 어떻게 연결해야 할지 혼동될 수 있습니다. 일반 펄스 신호를 카운트하기 위해서는 펄스 신호를 터미널 보드의 A 상 입력단에 연결하면 됩니다. 사용되는 라이브러리 함수는 엔코더 신호를 카운트하는 것과 동일합니다.

한편, 일반 펄스 신호를 카운트할 때에도 B 상 입력단에 0 Volt 또는 5 Volt 를 인가하여 Up/down 카운트를 할 수도 있습니다. B 상 입력단에 0 Volt 가 인가되면 업카운트(Up-count)가 되고, 5 Volt 가 인가되면 다운카운트(Down-count)가 됩니다. 만일 B 상 단자에 아무 신호도 연결되어 있지 않으면 업카운트를 하게 되며 펄스가 입력될 때마다 카운트는 증가하게 됩니다.

◈ void COMI_ENC_Config (HANDLE hDevice, int ch, int mode, BOOL bResetByZ)

이 함수는 지정한 A/B 상 카운터 채널의 모드를 설정합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *mode* : 채배모드를 설정합니다.
  - ▷ 0 또는 ENCODER_1X => 엔코더의 기본 분해능대로 카운트합니다.
  - ▷ 1 또는 ENCODER_2X => 2 채배 모드. 이 모드에서 카운터는 엔코더의 기본 분해능의 2 배 분해능을 가질 수 있습니다.
  - ▷ 2 또는 ENCODER_4X => 4 채배 모드. 이 모드에서 카운터는 엔코더의 기본 분해능의 4 배 분해능을 가질 수 있습니다.
- *bResetByZ* : A/B 상 카운트값을 Z 상 입력단자에 펄스가 입력될 때마다 리셋(Reset)할 것인지를 지정합니다.
  - ▷ 0 => 이 값으로 지정하면 A/B 상 카운트값은 Z 상에 의해 영향을 받지 않습니다.
  - ▷ 1 => 이 값으로 지정하면 A/B 상 카운트값은 Z-펄스가 발생할 때마다 리셋(Reset)됩니다.

☞ **지원 디바이스** : COMI-SD501

◈ void COMI_ENC_Reset (HANDLE hDevice, int ch)

이 함수는 지정한 A/B 상 카운터 채널의 카운트값을 0 으로 리셋합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.

☞ **지원 디바이스** : COMI-SD501

◈ void COMI_ENC_Load (HANDLE hDevice, int ch, long count)

이 함수는 지정한 A/B 상 카운터 채널의 카운트값을 지정한 값으로 로드(Load)합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *count* : 카운터에 로드할 값. 이 값의 범위는 -2147483648 ~ 2147483647 입니다.
- *참고* : 일반적으로 이 함수는 회전체가 원점에서 시작되지 않는 경우 회전체의 초기 위치를 지정하기 위하여 사용됩니다. 카운트값을 0 으로 초기화하기 위해서는 이 함수를 사용하는 것보다 COMI_ENC_Reset()함수를 사용하는 것이 속도면에서 유리합니다.

☞ **지원 디바이스 :** COMI-SD501

◈ long COMI_ENC_Read (HANDLE hDevice, int ch)

이 함수는 지정한 A/B 상 카운터 채널의 카운트값을 읽어서 그 값을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : A/B 상 카운트값.
- *참고* : 카운트값의 범위는 32 비트 정수값으로써 -2147483648 ~ 2147483647 입니다. 반환된 카운트값이 음의 값이면 회전체가 역방향으로 회전했음을 의미하며, 양의 값이면 정방향으로 회전했음을 의미합니다.

☞ **지원 디바이스 :** COMI-SD501

◈ void COMI_ENC_ResetZ (HANDLE hDevice, int ch)

이 함수는 지정한 Z 상 카운터 채널의 카운트값을 0 으로 리셋합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.

☞ **지원 디바이스 :** COMI-SD501

◈ void COMI_ENC_LoadZ(HANDLE hDevice, int ch, int count)

이 함수는 지정한 Z 상 카운터 채널의 카운트값을 지정한 값으로 로드(Load) 합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *count* : 카운터에 로드할 값. 이 값의 범위는 -32767 ~ 32768 입니다.
- *참고* : 일반적으로 이 함수는 회전체가 한방향으로만 회전하는 경우에 Z 상 카운터의 16 비트값의 전체 영역을 모두 사용하기 위한 것입니다. 기본적으로 Z 상 카운터의 초기값은 0 이므로 정방향으로 회전시 최대 계측 가능한 펄스 수는 32767 회가 됩니다. 그러나 초기값을 -32767 값으로 로드한 후 계측한다면 정방향으로 회전시 65535 회의 펄스를 계측할 수 있습니다.

☞ **지원 디바이스 :** COMI-SD501

◈ int COMI_ENC_ReadZ(HANDLE hDevice, int ch)

이 함수는 지정한 Z 상 카운터 채널의 카운트값을 읽어서 그 값을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : Counter 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : Z 상 카운트값.
- *참고* : 카운트값의 범위는 16 비트 정수값으로써 -32768 ~ 32767 입니다. 반환된 카운트값이 음의 값이면 회전체가 역방향으로 회전했음을 의미하며, 양의 값이면 정방향으로 회전했음을 의미합니다.

☞ **지원 디바이스 :** COMI-SD501

[리스트 4-9] Encoder Counter 함수 사용 예

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* - Subject : Encoder Counter.
/* - Contents: 이 프로그램은 4 개의 Encoder counter 채널을 읽어서 화면에
/*       보여주는 프로그램입니다.
/* - Remarks :
/*  1. 이 예제는 COMI-SD501 보드에만 적용가능합니다.
/*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include "Comidas.c"

void main (void)
{
  HANDLE hDevice;
  int i;
  long count[4];

  hDevice = COMI_LoadDevice (COMI_SD501, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    exit(0);
  }

  clrscr();
  for(i=0; i<4; i++){
    COMI_ENC_Config(hDevice, i, ENCODER_1X, FALSE);
    COMI_ENC_Reset(hDevice, i); /* Reset counter to 0 */
  }
  while(!kbhit())
  {
    for(i=0; i<4; i++)
      count[i] = COMI_ENC_Read (hDevice, i);
    gotoxy(4,10);
    printf("Read Encoder Count\n\n");
    printf("   CH0 =%9ld\n", count[0]);
    printf("   CH1 =%9ld\n", count[1]);
    printf("   CH2 =%9ld\n", count[2]);
    printf("   CH3 =%9ld\n", count[3]);
  }

  COMI_UnloadDevice(hDevice);
}
```

4-5-4 펄스 발생기

Pulse Generator(이후 PG) 기능은 COMI-SD501 보드에서만 제공하는 기능으로써 사용자가 원하는 주파수로 펄스를 생성시켜주는 기능입니다. 이 기능은 생성되는 펄스의 주파수를 제어할 수 있을 뿐 아니라 펄스의 수까지 제어할 수 있습니다. 이 기능은 주로 서보모터 및 스텝모터의 제어에 사용되는 기능입니다.

◈ double COMI_PG_Start (HANDLE hDevice, int ch, double freq, unsigned int num_pulses)

이 함수는 지정한 PG 채널을 통하여 지정한 주파수 및 펄스 수에 의거한 펄스를 출력합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.
- *freq* : 출력할 펄스의 주파수를 Hz 단위로 지정합니다.
- *num_pulses* : 출력할 펄스의 수를 지정합니다. 만일 이 값을 0 으로 지정하면 COMI_PG_Stop()함수가 수행되기 전까지 계속해서 펄스를 출력하게 됩니다.
- *Return* : 출력되는 펄스의 실제 주파수. 펄스의 주파수는 장치 내부에 장착된 타이머의 Base clock 을 정수로 분주하여 결정하므로 사용자가 지정한 주파수와 약간의 차이가 있을 수 있습니다.

◈ double COMI_PG_ChangeFreq (HANDLE hDevice, int ch, double freq)

이 함수는 지정한 PG 채널을 통하여 현재 출력되고 있는 펄스의 주파수를 Runtime 상에서 변경하여 줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.
- *freq* : 변경하고자 하는 펄스의 주파수를 Hz 단위로 지정합니다.
- *Return* : 출력되는 펄스의 실제 주파수.

◈ BOOL COMI_PG_IsActive(HANDLE hDevice, int ch)

이 함수는 지정한 PG 채널에 현재 펄스가 출력되고 있는지를 알려줍니다. 이 함수를 이용하면 출력 펄스 수를 제한한 경우에 원하는 펄스의 출력이 완료되었는지를 알아볼 수 있습니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.
- *Return* : 현재 펄스가 출력되고 있는지를 알려줍니다.

  0 => 현재 펄스가 출력되고 있지 않음

  1 => 현재 펄스가 출력되고 있음

◈ void COMI_PG_Stop(HANDLE hDevice, int ch)

이 함수는 지정한 PG 채널의 펄스 출력을 중지합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : PG 채널번호. 채널 번호는 0 부터 시작합니다.

[리스트 4-10] Pulse Generator 함수 사용 예 1.

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : Pulse Generator
/* Contents: This program generates infinite number of pulses of which
/*     frequency is defined by user.
/* - Remarks :
/*  1. 이 예제는 COMI-SD501 보드에만 적용가능합니다.
/*************************************************************************/
#include <stdio.h>
#include <conio.h>
#include "comidas.c"

#define FREQ  1000.f /* Pulse freq. = 1 kHz */
#define PG_CH 0  /* Pulse Generator Channel number */

void main()
{
  HANDLE hDevice;
  double ActFreq;

  hDevice = COMI_LoadDevice (COMI_SD501, 0); /* Load device */

  clrscr();
  printf("Generating infinite number of pulses through CH%d\n", PG_CH);
  printf("Set Freq.(Hz)   = %.0f\n", FREQ);
  /* Start pulse generation */
  ActFreq = COMI_PG_Start(hDevice, PG_CH, FREQ, 0);
  printf("Actual Freq.(Hz) = %.0f\n", ActFreq);
  printf("Press any key to exit !\n");
  while(!kbhit())
    ;
  COMI_PG_Stop (hDevice, PG_CH); /* Stop pulse generation */
  COMI_UnloadDevice(hDevice);
}
```

[리스트 4-11] Pulse Generator 함수 사용 예 2.

```
/*************************************************************************
/* [COMIDAS sample program by COMIZOA Inc., Ltd]
/*
/* Subject : Pulse Generator
/* Contents: 이 프로그램은 사용자가 지정하는 주파수의 펄스를 발생시킵니다.
/*     단, 이프로그램에서는 펄스의 수를 NBR_PULSES 값에 의해 제한합니다.
/* - Remarks :
/*  1. 이 예제는 COMI-SD501 보드에만 적용가능합니다.
/*************************************************************************/
#include <stdio.h>
#include <conio.h>
#include "comidas.c"
#define FREQ  1000.f /* Pulse freq. = 1 kHz */
#define NBR_PULSES 1000 /* 0 => Infinite number of pulses */
              /* others => The number of pulses to generate */
#define PG_CH 0  /* Pulse Generator Channel number */
void main()
{
  HANDLE hDevice;
  double ActFreq;

  hDevice = COMI_LoadDevice (COMI_SD501, 0); /* Load device */

  clrscr();
  printf("Generating pulses through CH%d\n", PG_CH);
  printf("Set Freq.(Hz)   = %.0f\n", FREQ);
  if(NBR_PULSES == 0)
    printf("Num pulses = Infinite\n");
  else
    printf("Num pulses = %d\n", NBR_PULSES);
  /* Start pulse generation */
  ActFreq = COMI_PG_Start(hDevice, PG_CH, FREQ, NBR_PULSES);
  printf("Actual Freq.(Hz) = %.0f\n\n", ActFreq);
  while(!kbhit()){
    /* Check if generation is completed */
    if(!COMI_PG_IsActive(hDevice, PG_CH))
      break;
  }
  printf("Pulse generation is completed!\n");
  printf("Press any key to exit!\n");
  getch();

  COMI_PG_Stop (hDevice, PG_CH); /* Stop pulse generation */
  COMI_UnloadDevice(hDevice);
}
```

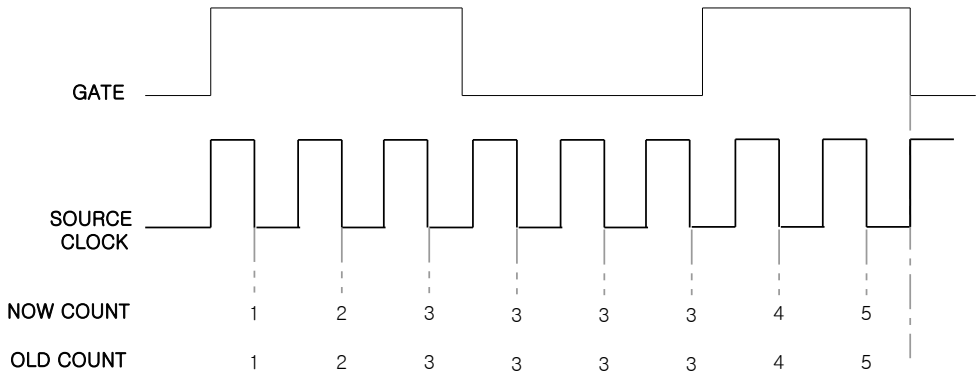
4-5-5 COMI-SD502 카운터

COMI-SD502 Counter 보드는 10 채널 24 비트 카운터 보드입니다. 특히 COMI-SD502 보드는 일반 카운터 기능외에 펄스 신호의 주파수를 쉽게 계측할 수 있는 기능을 제공합니다.

COMI-SD502 Counter 보드는 다음과 같이 두 가지 모드로 동작합니다.

■ MODE 0

이 모드는 일반적인 카운터 모드입니다. GATE 신호가 HIGH 상태로 유지되는 동안에 소스클럭(Source Clock)단에 입력되는 펄스의 수를 카운트합니다. 이 모드에서는 GATE 신호가 카운트 값을 자동으로 0 으로 초기화하지 않습니다. 그리고 GATE 신호에 아무 신호도 연결되지 않으면 GATE 신호는 자동적으로 High 상태로 유지됩니다. MODE 0 에서 카운터의 동작을 그림으로 표현하면 다음과 같습니다.

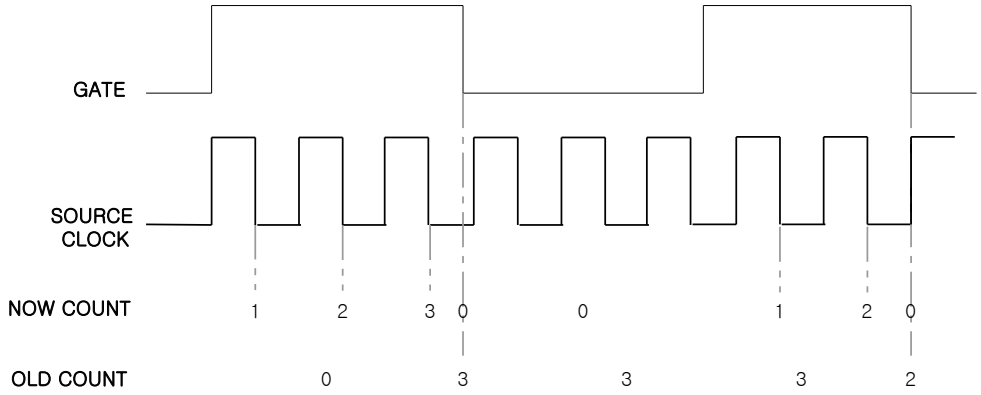


■ MODE 1

이 모드는 GATE 단자에 입력되는 펄스의 상태가 High 로 유지되는 시간을 계측하기 위해 주로 사용되는 모드입니다. 이 모드에서는 GATE 신호가 HIGH 상태로 유지되는 동안에 소스클럭(Source Clock)단에 입력되는 펄스의 수를 카운트하는 것은 MODE 1 과 동일합니다. 그러나 이 모드에서는 GATE 신호의 하강에지(Falling Edge)에서 현재의 카운트(Now count) 값을 Latch 에 저장(Old count) 한 후 0 으로 초기화하고, 다시 카운트를 시작합니다. 따라서

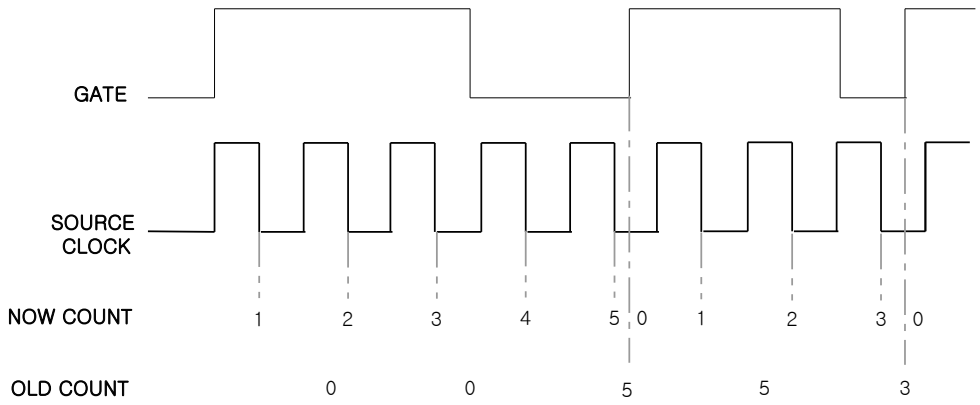
Old count 값을 참조하면 GATE 신호에 입력되는 펄스 신호의 주기 또는 주파수를 알 수 있습니다.

MODE 1 에서 카운터의 동작을 그림으로 표현하면 다음과 같습니다.



■ MODE 2

이 모드는 펄스의 주파수를 측정하는데 용이하게 사용될 수 있는 모드로써 GATE 신호의 상승에지(Rising Edge)가 새로운 카운트 시작을 트리거하고, 동시에 이전의 카운트값을 OLD COUNT 에 래치(Latch)합니다. 이 모드에서는 GATE 신호가 LOW 상태일 때에도 카운트는 계속됩니다.



◈ void COMI_SD502_SetCounter (HANDLE hDevice, int ch, int nMode, int nClkSource)

이 함수는 지정한 카운터 채널을 셋팅합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *nMode* : 카운터 모드를 지정합니다. 이 값은 0 또는 1 이어야 하며 각 모드에 대한 설명은 앞 장을 참조하십시오.
- *nClkSource* : 소스클럭(Source clock)을 지정합니다. 소스클럭은 외부에서 입력하거나 내부의 클럭을 사용할 수 있습니다. *nClkSource* 에 지정할 수 있는 값은 -1 또는 0 ~ 15 사이의 값입니다. 각 값의 의미는 다음과 같습니다.

값	소스클럭(Source Clock)	값	소스클럭(Source Clock)
-1	External Clock	8	19.531 KHz Internal Clock
0	5000.0 KHz Internal Clock	9	9.7656 KHz Internal Clock
1	2500.0 KHz Internal Clock	10	4.8828 KHz Internal Clock
2	1250.0 KHz Internal Clock	11	2.4414 KHz Internal Clock
3	625.00 KHz Internal Clock	12	1.2207 KHz Internal Clock
4	312.50 KHz Internal Clock	13	0.6104 KHz Internal Clock
5	156.30 KHz Internal Clock	14	0.3052 KHz Internal Clock
6	78.125 KHz Internal Clock	15	0.1526 KHz Internal Clock
7	39.063 KHz Internal Clock		

◈ ULONG COMI_SD502_ReadNowCount (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 현재 카운트 값을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *Return* : 현재 카운트(Now Count)값을 반환합니다.

◈ ULONG COMI_SD502_ReadOldCount (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 Old Count 값(GATE 신호의 Falling Edge 에서 Latch 저장된 값, 80 페이지 참조)을 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *Return* : Old Count 값을 반환합니다.

◈ BOOL COMI_SD502_GetGateState (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 GATE 신호의 상태를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.
- *Return* : 0 => GATE 신호가 LOW 상태임을 의미
  1 => GATE 신호가 HIGH 상태임을 의미

◈ double COMI_SD502_GetClkFreq (int nClkSrcIdx)

이 함수는 지정한 카운터 채널의 소스클럭을 내부클럭으로 설정한 경우에 내부클럭의 주파수를 반환합니다.

- *nClkSrcIdx* : 내부클럭의 인덱스(Index)값. 이 값은 COMI_SD502_SetCounter(..) 함수에서 사용되는 *nClkSource* 값과 동일한 값이어야 합니다.
- *Return* : 내부 소스클럭의 주파수 (Hz)
- *Remarks* : 소스클럭 인덱스와 주파수의 관계는 COMI_SD502_SetCounter() 함수 설명을 참조하십시오.

◈ void COMI_SD502_Clear (HANDLE hDevice, int ch)

이 함수는 지정한 카운터 채널의 New Count 와 Old Count 값을 0 으로 초기화 합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *ch* : 카운터 채널 번호. 이 값은 0 ~ 9 사이의 값이어야 합니다.

- *Remarks* : 카운터 모드를 0 으로 한 경우에는 New Count 값은 GATE 신호의 Falling Edge 에 의하여 자동으로 Clear 됩니다.

◈ void COMI_SD502_ClearMulti (HANDLE hDevice, ULONG dwCtrlBits)

이 함수는 여러 채널을 동시에 Clear 해주는 함수입니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *dwCtrlBits* : 이 값의 하위 10 비트가 순서에 따라 각 채널의 Clear 를 제어합니다. 비트값이 1 이면 해당 채널은 Clear 됩니다. BIT0 가 Channel 0 에 해당하며 BIT9 가 Channel 9 에 해당합니다.
- *Remarks* : 카운터 모드를 0 으로 한 경우에는 New Count 값은 GATE 신호의 Falling Edge 에 의하여 자동으로 Clear 됩니다.

4-6 인터럽트

이 단원은 COMI-SD434 Digital I/O Board 에만 지원되는 기능입니다. 인터럽트는 특정 상황이 발생되었을 때 사용자(또는 프로그래머)에게 이것을 알려주기 위한 것입니다. 인터럽트는 폴링(Polling) 방식과 달리 CPU 에 부하를 주지 않는 것이 장점입니다. 윈도우 환경에서는 일반 Application 레벨에서 인터럽트를 처리할 수 없으므로 이벤트를 통하여 Application 에게 인터럽트가 발생하였음을 알려줍니다. 인터럽트와 관련된 함수들은 다음과 같습니다.

- long COMI_INT_Start (HANDLE hDevice)
- long COMI_INT_Stop (HANDLE hDevice)
- long COMI_INT_Clear (HANDLE hDevice)
- long COMI_INT_SetIntChan (HANDLE hDevice, int Numch, int nState, int nMode)
- long COMI_INT_GetIntChan (HANDLE hDevice, int Numch, int nState, int nMode)
- long COMI_INT_GetIntState (HANDLE hDevice)
- long COMI_INT_SetHandler (HANDLE hDevice, long HandlerType, UINT nMessage, LPVOID lParam)

◈ long COMI_INT_Start (HANDLE hDevice)

이 함수는 인터럽트 기능을 사용가능하도록 Enable 해줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 함수의 수행 결과를 반환합니다.
  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_Stop (HANDLE hDevice)

이 함수는 인터럽트 기능을 사용하지 않도록 Disable 해줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 함수의 수행 결과를 반환합니다.
  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_GetIntState (HANDLE hDevice)

이 함수는 인터럽트의 현재 동작 상태를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 현재 인터럽트 동작 상태를 반환합니다.
  0 - 사용 안함, 1 - 사용.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_Clear (HANDLE hDevice)

이 함수는 인터럽트의 현재 상태를 초기화 하여줍니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 함수의 수행 결과를 반환합니다.
  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_SetIntChan (HANDLE hDevice, int NumCh, int nState, int nMode)

이 함수는 각 채널별로 인터럽트 사용여부를 설정합니다.

모드를 변경 시 인터럽트의 기능이 Disable 됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *NumCh* : 인터럽트 상태를 감지할 채널 번호입니다. COMI-SD43 에서는 베이스채널인 Digital I/O 16 채널에 대하여 인터럽트 기능을 사용할 수 있습니다.
- *nState* : 채널의 인터럽트 기능의 사용 여부를 설정합니다.

  0 - 사용 안함, 1 - 사용.
- *nMode* : 인터럽트 모드를 설정합니다. 이 값의 종류는 아래와 같이 5 가지로 지정할 수 있습니다.

Value	Meaning
0(INT_EDGE_BI)	설정한 채널의 Input 신호가 Rising 또는 Falling 일 경우 인터럽트가 발생합니다.
1(INT_EDGE_RISING)	설정한 채널의 Input 신호가 Rising 일 경우 인터럽트가 발생합니다.
2(INT_EDGE_FALLING)	설정한 채널의 Input 신호가 Falling 일 경우 인터럽트가 발생합니다.
3(INT_PATTERN_LOW)	패턴을 설정한 채널들의 Input 신호가 모두 Low 상태가 될 경우 인터럽트가 발생합니다.
4(INT_PATTERN_HIGH)	패턴을 설정한 채널들의 Input 신호가 모두 High 상태가 될 경우 인터럽트가 발생합니다.

- *Return* : 함수의 수행 결과를 반환합니다.

  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_GetIntChan (HANDLE hDevice, int NumCh, int nState, int nMode)

이 함수는 해당 채널의 인터럽트 사용여부를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *NumCh* : 인터럽트 사용여부를 확인할 채널 번호입니다. COMI-SD43 에서는 베이스채널인 Digital I/O 16 채널에 대하여 인터럽트 기능을 사용할 수 있습니다.
- *nState* : 현재 채널의 인터럽트 동작 상태를 반환 합니다.
  0 - 사용 안함, 1 - 사용.
- *nMode* : 현재 채널의 인터럽트 동작 모드를 반환 합니다.
  0 - Bidirection (Edge Mode)
  1 - Rising (Edge Mode), 2 - Falling (Edge Mode)
  3 - Low (Pattern Mode), 4 - High (Pattern Mode)
- *Return* : 해당 채널의 인터럽트 사용여부를 반환 합니다.
  0 - 사용 안함, 1 - 사용.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_INT_SetHandler (HANDLE hDevice, long Type, HANDLE Handler, UINT nMessage, LPVOID lParam)

이 함수는 인터럽트 이벤트 핸들러를 등록합니다. 이벤트 핸들러는 윈도우 메시지 방식, 이벤트 객체 방식, 콜백 함수 방식등의 인터럽트 처리를 수행할 수 있으며, 해당 인터럽트가 발생되었을 때, 지정된 HandlerType 에 따른 처리가 이루어지게 됩니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Type* : 인터럽트 핸들러의 종류를 지정합니다. 이 값의 종류는 아래와 같이 3 가지로 지정할 수 있습니다.

Value	Meaning
0(IHT_MESSAGE)	윈도우 메시지 전달방식을 통하여 인터럽트를 통지합니다. ·장점 : GUI 관련 작업을 이벤트 핸들러에서 직접 수행할 수 있다. ·단점 : 윈도우가 메시지루프 상황에 따라서 상당한 지연시간을 가질 수 있다.
1(IHT_EVENT)	이벤트 객체를 통하여 인터럽트를 통지합니다. ·장점 : 메시지 전달 방식보다 비교적 적은 지연시간을 가질 수 있다.

	·단점 : 사용하기가 복잡하다.
2(IHT_CALLBACK)	콜백 함수를 통하여 인터럽트를 통지합니다. 이 방식이 가장 권장되는 방식입니다. ·장점 : 메시지 전달의 지연시간이 3 가지 방식 중에서 가장 적다. ·단점 : GUI 관련 작업을 수행할 수 없다.

- *Handler* : 이 매개변수의 의미는 Type 설정에 따라서 다음과 같이 달라집니다.

Value	Meaning
0(메시지 방식) 일때	Handler 매개변수는 윈도우 핸들을 의미합니다.
1(이벤트 방식) 일때	Handler 매개변수는 이벤트 객체를 의미합니다.
2(콜백 방식) 일때	Handler 매개변수는 콜백 함수를 의미합니다.

- *uMessage* : 이 매개변수는 Type 이 IHT_MESSAGE 로 설정되었을 때만 유효한 것으로서 윈도우 메시지 번호를 설정합니다.
- *lParam* : 인터럽트가 처리되는 함수에 전달될 핸들값을 설정합니다.
- *Return* : 함수의 수행 결과를 반환합니다.

  0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

[리스트 4-12] Interrupt 함수 사용 예

```
/*********************************************************************
/* [COMIDAS sample code by COMIZOA Inc., Ltd]
/*
/* Subject : Interrupt
/* Contents: 이 프로그램은 인터럽트 감지를 시작하고 난 뒤부터 인터럽트가 발생되었을 때
/*    사용자에게 알려주는 예제 코드입니다.
/*
/* - Remarks :
/*   1. 이 예제는 인터럽트 기능을 지원하는 COMI-SD434 보드를 기준으로 작성되었습니다.
/*
/*********************************************************************/
#include <stdio.h>
#include "Comidas.h"
```

```
#define WMU_Interrupt  (WM_APP + 1)
Void WINAPI InterruptCallback (LPVOID lParam);

void main (void)
{
  HANDLE hDevice;

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

  hDevice = COMI_LoadDevice (COMI_SD434, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  COMI_INT_SetHandler (hDevice, IHT_CALLBACK, InterruptCallback,
                                                    WMU_Interrupt, this);
  if( COMI_INT_Start (hDevice))
    printf("Interrupt Activate!");
  COMI_INT_SetIntChan (hDevice, Ch0, TRUE);
  COMI_DO_PutOne (hDevice, Ch0, TRUE);
}

Void WINAPI InterruptCallback (LPVOID lParam)
{
  printf ("Interrupt detection!);
}
```

4-7 필터

이 단원은 COMI-SD434 Digital I/O Board 에만 지원되는 기능입니다. 디지털 입력의 외부 노이즈 및 원하지 않는 신호를 구분하여 받지 않게끔 하는 기능입니다. 모드 변경에 따라 필터되는 타이밍 간격이 조절 가능합니다. 필터와 관련된 함수들은 다음과 같습니다.

- long COMI_SetFilterMode (HANDLE hDevice, int nMode)
- long COMI_GetFilterMode (HANDLE hDevice)

◈ long COMI_SetFilterMode (HANDLE hDevice, int nMode)

이 함수는 필터 클럭 모드를 변경하여 필터링 되는 타이밍 간격을 설정 합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *nMode*: 필터 클럭 모드를 설정 합니다. 필터를 적용하지 않는 디폴트 값은 12 MHz 입니다.

Value	Meaning
0(CLOCK_12MHz)	12 Mhz
1(CLOCK_3KHz)	3 Khz
2(CLOCK_760Hz)	760 Hz
3(CLOCK_95Hz)	95 Hz

- *Return* : 함수의 수행 결과를 반환합니다.

0 - 수행 실패, 1 - 수행 성공.

☞ **지원 디바이스 :** COMI-SD434.

◈ long COMI_GetFilterMode (HANDLE hDevice)

이 함수는 설정한 필터 클럭 모드를 반환합니다.

- *hDevice* : 디바이스 핸들 값입니다. 이 값은 COMI_LoadDevice()함수에 의해 얻어진 값이어야 합니다.
- *Return* : 설정한 필터 클럭 모드를 반환합니다.

Value	Meaning
0(CLOCK_12MHz)	12 Mhz
1(CLOCK_3KHz)	3 Khz
2(CLOCK_760Hz)	760 Hz
3(CLOCK_95Hz)	95 Hz

☞ **지원 디바이스 :** COMI-SD434.

[리스트 4-13] Filter 함수 사용 예

```
/*************************************************************************
/* [COMIDAS sample code by COMIZOA Inc., Ltd]
/*
/* Subject : Filter
/* Contents: 이 코드는 사용자가 디지털 필터 값을 다른 값으로 변경을 하도록 하는 예제
/*    코드 입니다.
/* - Remarks :
/*   1. 이 예제는 인터럽트 기능을 지원하는 COMI-CP434 보드를 기준으로 작성되었습니다.
/*************************************************************************/

#include <stdio.h>
#include "Comidas.h"

void main (void)
{
  HANDLE hDevice;

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

  hDevice = COMI_LoadDevice (COMI_CP101, 0);
  if(hDevice == INVALID_HANDLE_VALUE){
    printf("Can't load specified device!");
    COMI_UnloadDll();
    exit(0);
  }

  COMI_SetFilterMode (hDevice, CLOCK_12MHz);

  If(COMI_GetFilterMode (hDevice) == CLOCK_12MHz)
    printf ("12MHz Digital Filter Mode!")
}
```

부 록

CP 시리즈 보드 관련 함수

구 분	함 수	COMI-CP101	COMI-CP201	COMI-CP301	COMI-CP401	COMI-CP501
시작/종료	BOOL COMI_LoadDll (void)	V	V	V	V	V
	void COMI_UnloadDll (void)	V	V	V	V	V
	HANDLE COMI_LoadDevice (COMIDAS_DEVID deviceID, ULONG instance)	V	V	V	V	V
	void COMI_UnloadDevice (HANDLE hDevice)	V	V	V	V	V
Analog Input	BOOL COMI_AD_SetRange (HANDLE hDevice, int ch, float vmin, float vmax)	V	V			
	float COMI_DigitToVolt (short digit, float vmin, float vmax)	V	V			
	short COMI_AD_GetDigit (HANDLE hDevice, int ch)	V	V			
	float COMI_AD_GetVolt (HANDLE hDevice, int ch)	V	V			
	long COMI_US_Start (HANDLE hDevice, int numCh, int *chanList, UINT scanFreq,□UINT msb, int trsMethod)	V	V			
	BOOL COMI_US_Stop (HANDLE hDevice, BOOL bReleaseBuf)	V	V			
	ULONG COMI_US_CurCount (HANDLE hDevice)	V	V			
	UINT COMI_US_SBPos (HANDLE hDevice, int chOrder, ULONG scanCount)	V	V			
	short* COMI_US_GetBufPtr (HANDLE hDevice)	V	V			
	BOOL COMI_US_ReleaseBuf (HANDLE hDevice)	V	V			
	short COMI_US_RetrvOne (HANDLE hDevice, int chOrder, ULONG scanCount)	V	V			
	ULONG COMI_US_RetrvChannel (HANDLE hDevice, int chOrder, ULONG startCount, int maxNumData, void *pDestBuf, TVarType VarType)	V	V			
	UINT COMI_US_RetrvBlock (HANDLE hDevice, UINT startCount, int maxNumScan, void *pDestBuf, TVarType VarType)	V	V			
	BOOL COMI_US_FileSaveFirst (HANDLE hDevice, char *szFilePath)	V	V			
	ULONG COMI_US_FileSaveNext (HANDLE hDevice)	V	V			
	BOOL COMI_US_FileSaveStop (HANDLE hDevice)	V	V			
	void COMI_US_FileConvert (char *szBinFilePath, char *szTextFilePath)	V	V			
Digital In /Out	void COMI_DIO_SetUsage (HANDLE hDevice, int usage)				V	
	int COMI_DI_GetOne (HANDLE hDevice, int ch)	V	V	V	V	
	DWORD COMI_DI_GetAll (HANDLE hDevice)	V	V	V	V	
	BOOL COMI_DO_PutOne (HANDLE hDevice, int ch, int status)	V	V	V	V	
	BOOL COMI_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)	V	V	V	V	
Analog Out	BOOL COMI_DA_Out (HANDLE hDevice, int ch, float volt)	V		V		
Counter	void COMI_SetCounter (HANDLE hDevice, int ch, int rw, int op, int bcd_bin, U	V	V			V
	void COMI_LoadCount (HANDLE hDevice, int ch, USHORT load_value)	V	V			V
	USHORT COMI_ReadCount (HANDLE hDevice, int ch)	V	V			V

SD 시리즈 보드 관련 함수

구 분	함 수	COMI-SD10x	COMI-SD201	COMI-SD301	COMI-SD501	COMI-SD502
시작/종료	BOOL COMI_LoadDll (void)	V	V	V	V	V
	void COMI_UnloadDll (void)	V	V	V	V	V
	HANDLE COMI_LoadDevice (COMIDAS_DEVID deviceID, ULONG instance)	V	V	V	V	V
	void COMI_UnloadDevice (HANDLE hDevice)	V	V	V	V	V
Analog Input	BOOL COMI_AD_SetRange (HANDLE hDevice, int ch, float vmin, float vmax)	V	V			
	float COMI_DigitToVolt (short digit, float vmin, float vmax)	V				
	float COMI_Digit16ToVolt (short digit, float vmin, float vmax)	X	V			
	short COMI_AD_GetDigit (HANDLE hDevice, int ch)	V	V			
	float COMI_AD_GetVolt (HANDLE hDevice, int ch)	V	V			
	long COMI_US_Start (HANDLE hDevice, int numCh, int *chanList, UINT scanFreq,□UINT msb, int trsMethod)	V	V			
	BOOL COMI_US_Stop (HANDLE hDevice, BOOL bReleaseBuf)	V	V			
	long COMI_US_ChangeScanFreq (HANDLE hDevice, UINT dwScanFreq)	V	V			
	void COMI_US_ResetCount (HANDLE hDevice)	V	V			
	ULONG COMI_US_CurCount (HANDLE hDevice)	V	V			
	UINT COMI_US_SBPos (HANDLE hDevice, int chOrder, ULONG scanCount)	V	V			
	short* COMI_US_GetBufPtr (HANDLE hDevice)	V	V			
	BOOL COMI_US_ReleaseBuf (HANDLE hDevice)	V	V			
	short COMI_US_RetrvOne (HANDLE hDevice, int chOrder, ULONG scanCount)	V	V			
	ULONG COMI_US_RetrvChannel (HANDLE hDevice, int chOrder, ULONG startCount, int maxNumData, void *pDestBuf, TVarType VarType)	V	V			
	UINT COMI_US_RetrvBlock (HANDLE hDevice, UINT startCount, int maxNumScan, void *pDestBuf, TVarType VarType)	V	V			
	BOOL COMI_US_FileSaveFirst (HANDLE hDevice, char *szFilePath, BOOL blsFro	V	V			
	ULONG COMI_US_FileSaveNext (HANDLE hDevice)	V	V			
	BOOL COMI_US_FileSaveStop (HANDLE hDevice)	V	V			
	void COMI_US_FileConvert (char *szBinFilePath, char *szTextFilePath, ULONG n	V	V			
Analog Output	BOOL COMI_DA_Out (HANDLE hDevice, int ch, float volt)	V		V		
	long COMI_WFM_Start (HANDLE hDevice, int ch, float *pDataBuffer, UINT nNumData, UINT nPPS, int nMaxLoops)			V		
	long COMI_WFM_RateChange (HANDLE hDevice, int ch, ULONG nPPS)			V		
	long COMI_WFM_GetCurPos (HANDLE hDevice, int ch)			V		
	long COMI_WFM_GetCurLoops (HANDLE hDevice, int ch)			V		
	void COMI_WFM_Stop (HANDLE hDevice, int ch)			V		

구 분	함 수	COMI-SD10x	COMI-SD201	COMI-SD301	COMI-SD501	COMI-SD502
Digital In/Out	void COMI_DIO_SetUsage (HANDLE hDevice, int usage)			V		V
	int COMI_DI_GetOne (HANDLE hDevice, int ch)	V	V	V		V
	DWORD COMI_DI_GetAll (HANDLE hDevice)	V	V	V		V
	BOOL COMI_DO_PutOne (HANDLE hDevice, int ch, int status)	V	V	V	V	V
	BOOL COMI_DO_PutAll (HANDLE hDevice, DWORD dwStatuses)	V	V	V	V	V
Counter	ULONG COMI_ReadCounter32 (HANDLE hDevice, int ch)	V	V			
	void COMI_ClearCounter32 (HANDLE hDevice, int ch)	V	V			
	void COMI_ENC_Config (HANDLE hDevice, int ch, int mode, BOOL bResetByZ)				V	
	void COMI_ENC_Reset (HANDLE hDevice, int ch)				V	
	void COMI_ENC_Load (HANDLE hDevice, int ch, long Count)				V	
	long COMI_ENC_Read (HANDLE hDevice, int ch)				V	
	void COMI_ENC_ResetZ (HANDLE hDevice, int ch)				V	
	void COMI_ENC_LoadZ (HANDLE hDevice, int ch, short count)				V	
	short COMI_ENC_ReadZ (HANDLE hDevice, int ch)				V	
	double COMI_PG_Start (HANDLE hDevice, int ch, double freq, UINT num_pulses)				V	
	BOOL COMI_PG_IsActive (HANDLE hDevice, int ch)				V	
	void COMI_PG_Stop (HANDLE hDevice, int ch)				V	
	void COMI_SD502_SetCounter (HANDLE hDevice, int ch, int nMode, UINT nClkSource)					V
	ULONG COMI_SD502_ReadNowCount (HANDLE hDevice, int ch)					V
	ULONG COMI_SD502_ReadOldCount (HANDLE hDevice, int ch)					V
	BOOL COMI_SD502_GetGateState (HANDLE hDevice, int ch)					V
	double COMI_SD502_GetClkFreq (int nClkSrcIdx)					V
	void COMI_SD502_Clear (HANDLE hDevice, int ch)					V
	void COMI_SD502_ClearMulti (HANDLE hDevice, ULONG dwCtrlBits)					V
PID Control	BOOL COMI_PID_Enable(HANDLE hDevice)	V				
	BOOL COMI_PID_SetParams(HANDLE hDevice, int nNumCtrls, TPidParams *pPidParams)	V				
	BOOL COMI_PID_Disable(HANDLE hDevice)	V				