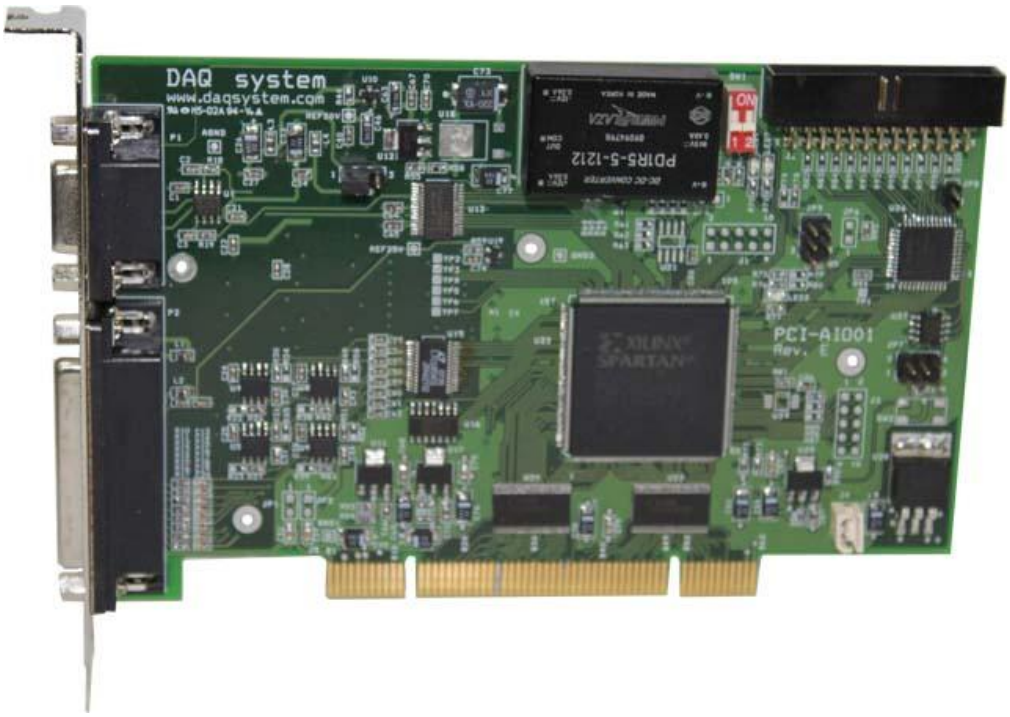
PCI-AIO01/02/04 API Programming (Rev 1.0)



API 설명

PCI-AIO 시리즈 보드를 사용하기 위한 API(Application Programming Interface)를 설명한다.

본 API는 PCI_AIO01, PCI_02, PCI_AIO04 제품을 지원한다.

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//  DLL export header file for PCI boards
//  (c) 2009,2010 DAQ system
//////////////////////////////////////////////////////////////////////////////////////////////////
#define MAX_MODEL   50                      // Maximum number of Model number
#define MAX_DEVICE  4                       // Maximum number of board to find

#define PCI_AIO00   0                       // Define PCI-AIO00 number
#define PCI_AIO01   1                       // Define PCI-AIO01 number
#define PCI_AIO02   2                       // Define PCI-AIO02 number
#define PCI_AIO03   3                       // Define PCI-AIO03 number
#define PCI_AIO04   4                       // Define PCI-AIO04 number

//************************************************************************************************
// Board level API functions
//************************************************************************************************
extern "C" __declspec(dllimport) BOOL  __stdcall OpenDAQDevice(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall ResetBoard(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall CloseDAQDevice(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall GetBoardVersion(int nModel, int nBoard, int *version);

//************************************************************************************************
// ADC API functions
//************************************************************************************************
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_Init(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_Reset(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_Close(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_GetData(int nModel, int nBoard, int nChannel, int *nData);

//////////////////////////////////////////////////////////////////////////////////////////////////
//  ADC Set Input Range
//  nRange = 0 : 0 to 5V
//  nRange = 1 : -5V to 5V
//  nRange = 2 : 0 to 10V
//  nRange = 3 : -10V to 10V
//////////////////////////////////////////////////////////////////////////////////////////////////
// **** ADC_SetRange() function has API bug. Now You should use ADC_ConfigChannelEx(). ****
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_SetRange(int nModel, int nBoard, int nChannel, int nRange);

//////////////////////////////////////////////////////////////////////////////////////////////////
//  ADC Input Mode
//  nChannel = Don't care
//  dwConfig = 0 : Single-ended
//  dwConfig = 1 : Differential
//////////////////////////////////////////////////////////////////////////////////////////////////
// **** ADC_ConfigChannel() function has API bug. Now You should use ADC_ConfigChannelEx(). ****
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_ConfigChannel(int nModel, int nBoard, int nChannel, DWORD dwConfig);

//////////////////////////////////////////////////////////////////////////////////////////////////
//  ADC Input Mode
//  nChannel    = x : selected channel
//  nPol        = 0 : Single-ended
//  nPol        = 1 : Differential
//  nRange      = 0 : 0 to 5V
//  nRange      = 1 : -5V to 5V
//  nRange      = 2 : 0 to 10V
//  nRange      = 3 : -10V to 10V
//////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_ConfigChannelEx(int nModel, int nBoard, int nChannel, int nPol, int nRange);

////////////////////////////////////////////////////////////////////////////////////////
//      ADC Resolution Set
//  dwResol = 0 : 12-bit ADC
//  dwResol = 1 : 14-bit ADC
//  dwResol = 2 : 16-bit ADC
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_ConfigResolution(int nModel, int nBoard, DWORD dwResol);
////////////////////////////////////////////////////////////////////////////////////////
// Start Buffered Read
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_StartBufferedRead(int nModel, int nBoard);
////////////////////////////////////////////////////////////////////////////////////////
//  Stop Buffered Read
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_StoptBufferedRead(int nModel, int nBoard);
////////////////////////////////////////////////////////////////////////////////////////
//      ADC BUFFERED DATA READ : raw-data(int type)
//  nCount   : read request numbers
//  *Data    : integer type data buffer
//  return   : readable numbers in Data buffer
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) int  __stdcall ADC_GetBufferedData(int nModel, int nBoard, int nChannel, int nCount, int* Data);
////////////////////////////////////////////////////////////////////////////////////////
//      ADC BUFFERED DATA READ : voltage converted-data(float type)
//  nCount   : read request numbers
//  *fData   : float type data buffer
//  return   : readable numbers in fData buffer
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) int  __stdcall ADC_GetBufferedDataEx(int nModel, int nBoard, int nChannel, int nCount,float* fData);
////////////////////////////////////////////////////////////////////////////////////////
//  Set ADC Average data count
//  nCount = 1 - 256;
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) BOOL  __stdcall ADC_SetAvgCounter(int nModel, int nBoard, int nChannel, int nCount);

//**************************************************************************************
// DAC API functions
//**************************************************************************************
extern "C" __declspec(dllimport) BOOL  __stdcall DAC_Init(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall DAC_Reset(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall DAC_Close(int nModel, int nBoard);
extern "C" __declspec(dllimport) BOOL  __stdcall DAC_SetOutput(int nModel, int nBoard, int nChannel, DWORD buf);
////////////////////////////////////////////////////////////////////////////////////////
//      DAC Channel config
//  AIO04
//  dwConfig = 0 : bipolar
//  dwConfig = 1 ; unipolar
////////////////////////////////////////////////////////////////////////////////////////
extern "C" __declspec(dllimport) BOOL  __stdcall DAC_ConfigChannel(int nModel, int nBoard, int nChannel, DWORD dwConfig);
```

Board Level API Functions

Overview

BOOL **OpenDAQDevice (int nModel, int nBoard)**

BOOL **ResetBoard (int nModel, int nBoard)**

BOOL **CloseDAQDevice (int nModel, int nBoard)**

BOOL **GetBoardVersion (int nModel, int nBoard, int *version)**

OpenDAQDevice

디바이스를 Open한다. PCI-AIO 시리즈 보드를 이용하는 프로그램에서 초기에 반드시 한번 함수를 호출하여 디바이스를 Open 하여야 한다.

BOOL **OpenDAQDevice(int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ResetBoard

현재 시스템(PC)에 장착된 디바이스를 초기화 한다.

BOOL **ResetBoard(int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

CloseDAQDevice

오픈된 모든 PCI-AIO 시리즈 디바이스를 Close한다. 장치의 사용이 끝나게 되면, 반드시 장치를 Close 하여 다른 프로그램에서 사용할 수 있도록 한다.

BOOL **CloseDAQDevice(int nModel, int nBoard)**

Parameters:

nModel : PCI_ AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

GetBoardVersion

PCI-AIO 디바이스의 하드웨어 버전 정보를 얻는다. .

BOOL **GetBoardVersion(int nModel, int nBoard, int *version)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

version : 버전 정보를 받을 변수의 포인터이다. 정상적인 경우 양의 정수 값을 나타낸다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC(Analog to Digital Convertor) API Functions

Overview

BOOL **ADC_Init (int nModel, int nBoard)**

BOOL **ADC_Reset (int nModel, int nBoard)**

BOOL **ADC_Close (int nModel, int nBoard)**

BOOL **ADC_GetData (int nModel, int nBoard, int nChannel, int *nData)**

BOOL **ADC_SetRange (int nModel, int nBoard, int nChannel, int nRange)**

BOOL **ADC_ConfigChannel (int nModel, int nBoard, int nChannel, DWORD dwConfig)**

BOOL **ADC_ConfigChannelEx (int nModel, int nBoard, int nChannel, int nPol, int nRange)**

BOOL **ADC_ConfigResolution (int nModel, int nBoard, DWORD dwResol)**

BOOL **ADC_StartBufferedRead (int nModel, int nBoard)**

BOOL **ADC_StopBufferedRead (int nModel, int nBoard)**

int **ADC_GetBufferedData (int nModel,int nBoard,int nChannel,int nCount,int *nData)**

int **ADC_GetBufferedDataEx (int nModel,int nBoard,int nChannel,int nCount,float *fData)**

BOOL **ADC_SetAvgCounter (int nModel, int nBoard,int nChannel,int nCount)**

ADC_Init

ADC의 설정을 초기화 한다.

BOOL **ADC_Init (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_Reset

ADC의 기능을 리셋 시킨다.

BOOL **ADC_Reset (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_Close

ADC 기능을 더 이상 사용하지 않을 경우 호출하여 사용한 리소스를 해제한다.

BOOL **ADC_Close (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_GetData

현재 ADC 입력에 대한 한 개의 데이터 값을 읽어 온다.

읽어올 데이터는 정밀도, 입력 극성 및 범위에 영향을 받으므로 함수 ADC_ConfigResolution(), ADC_ConfigChannelEx() 를 호출해야 한다.

정밀도에 따른 ADC의 값의 범위는 극성 설정에 따라 다음 [표 3]과 같다.

[표 3. 정밀도에 따른 AD 값의 범위]

정밀도	극성	
	Unipolar	Bipolar
12-비트	0 ~ +4095	-2048 ~ +2047
14-비트	0 ~ +16383	-8192 ~ +8191
16-비트	0 ~ +65535	-32768 ~ +32767

이때 실제 전압의 계산은 다음과 같다.

전압 = (읽은 값/Resolution 범위) *(Range 최대값-최소값)

예) 0 에서 5V 범위, 12-비트로 설정한 경우

1024를 읽었을 때 실제 전압은

(1024 /4096) * (5-0) = 1.25 [V] 가 된다.

-10 에서 + 10V 범위, 16-비트 설정한 경우

-16384를 읽었을 때 실제 전압은

(-16384 /65536) * (10-(-10)) = -5 [V] 가 된다.

BOOL **ADC_GetData (int nModel, int nBoard, int nChannel, int *nData)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : ADC 채널 값을 적는다. PCI-AIO01의 채널 번호는 0에서 7까지 이다.

nData : 읽어올 ADC값을 받을 변수의 포인터이다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_SetRange

ADC의 아날로그 입력 범위를 지정한다.

BOOL **ADC_SetRange (int nModel, int nBoard, int nChannel, int nRange)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : 무의미하며 ‘0’을 설정한다. 이 함수에서는 모든 채널의 입력범위가 동일하게 적용되므로 채널 별 설정을 위해서는 ADC_ ConfigChannelEx () 함수를 사용한다.

nRange : ADC의 입력 범위를 설정한다. 디폴트는 0 에서 10V 이다.

0 : 0 에서 5V

1 : -5 에서 5V

2 : 0 에서 10V

3 : -10 에서 10V

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_ConfigChannel

ADC 채널의 아날로그 입력 극성을 설정한다.

BOOL **ADC_ConfigChannel (int nModel, int nBoard, int nChannel, DWORD dwConfig)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : 무의미하며 '0'을 설정한다. 이 함수에서는 모든 채널의 입력특성이 동일하게 적용되므로 채널 별 설정을 위해서는 ADC_ConfigChannelEx() 함수를 사용한다.

dwConfig : 보드의 아날로그 입력 극성.
0 : Single-Ended(SE)
1 : Differential(DI)

Return Value:

함수 호출에 실패할 경우 "FALSE" 성공일 경우 "TRUE"을 리턴함.

ADC_ConfigChannelEx

채널 별 ADC 입력 극성, 입력 범위를 설정한다.

BOOL **ADC_ConfigChannelEx (int nModel, int nBoard, int nChannel, int nPol, int nRange)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : 적용 채널 값을 지정한다.

nPol : 아날로그 입력 극성.
0 : Single-Ended(SE)
1 : Differential(DI)

nRange : ADC의 입력 범위를 설정한다. 디폴트는 0 에서 10V 이다.
0 : 0 에서 5V
1 : -5 에서 5V
2 : 0 에서 10V
3 : -10 에서 10V

Return Value:

함수 호출에 실패할 경우 "FALSE" 성공일 경우 "TRUE"을 리턴함.

ADC_ConfigResolution

AD 데이터 폭(비트)를 설정한다

보드의 AD 컨버터가 12,14,16 비트 3가지 정밀도(Resolution)를 지원하는 것과 별도로 읽는 데이터의 정밀도를 지정한다.

BOOL **ADC_ConfigResolution (int nModel, int nBoard, DWORD dwResol)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

dwResol : 데이터 비트 설정 값. 지원하고 있는 AD 컨버터의 데이터 비트에 따라 설정한다.
12-bit ADC 일 때 0(디폴트), 14-bit ADC 일 때 1, 16-bit ADC 일 때 2이다.

Return Value :

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_StartBufferedRead

AD 데이터를 버퍼에 저장을 시작한다.

라이브러리에 채널당 32,768개의 데이터 버퍼가 할당된다. 이 함수를 호출하게 되면 버퍼에 AD 데이터가 연속으로 링 버퍼 형식으로 연속 저장된다. 사용자는 데이터가 덮어써지기 전에 읽어야만 유효한 데이터를 취득할 수 있다.

BOOL **ADC_StartBufferedRead (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_StopBufferedRead

AD 데이터를 버퍼에 저장을 중지한다.

BOOL **ADC_StopBufferedRead (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

ADC_GetBufferedData

버퍼에 저장된 AD 데이터를 읽어 온다.

ADC_GetData()와 같이 정밀도, 입력 극성 및 범위는 함수 ADC_ConfigResolution(), ADC_ConfigChannelEx() 를 이용하여 설정한다.

정밀도에 따른 ADC의 값의 범위는 극성 설정에 따라 표 3과 같다.

이때 실제 전압의 계산은 다음과 같다.

전압 = (읽은 값/Resolution 범위) *(Range 최대값-최소값)

예) 0 에서 5V 범위, 12-비트로 설정한 경우

1024를 읽었을 때 실제 전압은

(1024 /4096) * (5-0) = 1.25 [V] 가 된다.

-10 에서 + 10V 범위, 16-비트 설정한 경우

-16384를 읽었을 때 실제 전압은

(-16384 /65536) * (10-(-10)) = -5 [V] 가 된다.

int **ADC_GetBufferedData (int nModel,int nBoard,int nChannel,int nCount,int *nData)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : ADC 채널 값을 적는다. PCI-AIO01의 채널 번호는 0에서 7까지 이다.

nCount : 읽어 올 데이터 수를 지정한다. 최대 버퍼 수(32,768) 이내의 수를 지정한다.

nData : AD 데이터가 저장될 변수의 포인터이다. 버퍼 크기를 읽어 올 데이터 수 이상으로 할당한다.

Return Value:

실제 nData에 저장된 데이터 수이다.

ADC_GetBufferedDataEx

버퍼에 저장된 AD 데이터를 읽어 온다.

버퍼에 저장된 데이터를 전압레벨 float 형식으로 데이터를 취득한다. 정밀도, 입력 극성 및 입력 범위에 얻어지는 데이터 값이 영향을 받으므로 ADC_ConfigResolution(), ADC_ConfigChannelEx() 함수를 반드시 호출해야 한다.

int **ADC_GetBufferedDataEx (int nModel,int nBoard,int nChannel,int nCount,float *fData)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : ADC 채널 값을 적는다. PCI-AIO01의 채널 번호는 0에서 7까지 이다.

nCount : 읽어 올 데이터 수를 지정한다. 최대 버퍼 수(32,768) 이내의 수를 지정한다.

nData : 전압레벨 데이터가 저장될 변수의 포인터이다. 버퍼 크기를 읽어 올 데이터 수 이상으로 할당한다.

Return Value:

실제 nData에 저장된 데이터 수이다.

ADC_SetAvgCounter

이동평균에 사용할 AD 데이터 수를 설정한다.

데이터 수가 1일 경우에는 이동평균이 적용되지 않으며, ADC_GetBufferedData(), ADC_GetBufferedDataEx() 함수는 평균화된 데이터를 취득한다.

BOOL **ADC_SetAvgCounter (int nModel, int nBoard,int nChannel,int nCount)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : ADC 채널 값을 적는다. PCI-AIO01의 채널 번호는 0에서 7까지 이다.

nCount : 이동평균이 적용되는 데이터 수를 지정한다. 1 ~ 255의 수를 지정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

DAC(Digital to Analog Convertor) API Functions

Overview

BOOL **DAC_Init (int nModel, int nBoard)**

BOOL **DAC_Reset (int nModel, int nBoard)**

BOOL **DAC_Close (int nModel, int nBoard)**

BOOL **DAC_SetOutput (int nModel, int nBoard, int nChannel, DWORD dwData)**

BOOL **DAC_ConfigChannel (int nModel, int nBoard, int nChannel, DWORD dwConfig)**

DAC_Init

DAC 의 설정을 초기화 한다.

BOOL **DAC_Init (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

DAC_Reset

DAC 의 기능을 리셋한다.

BOOL **DAC_Reset (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

DAC_Close

DAC 기능을 더 이상 사용하지 않을 경우 호출하여 사용한 리소스를 해제한다.

BOOL **DAC_Close (int nModel, int nBoard)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

DAC_SetOutput

DAC 값을 출력 한다.

리턴되는 DAC 의 출력 되는 범위는 Unipolar의 경우 0 에서 10V 이며, Bipolar의 경우 -10 에서 10V 이다.

이때 설정하는 값의 범위는 0 에서 4095 이며 계산은 다음과 같다.

예) 0 에서 10V 범위로 설정되어 있는 경우

5V 를 출력할 경우 설정 값은

(5V/ 10V) * 4096 = 2048 가 된다.

-10 에서 + 10V의 경우

5V 를 출력할 경우 설정 값은

(10+5V/20V) * 4096 = 3072 가 된다.

-5V 를 출력할 경우 설정 값은

(10-5V/20V) * 4096 = 1024 가 된다.

BOOL **DAC_SetOutput (int nModel, int nBoard, int nChannel, DWORD dwData)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : DAC 채널 값을 적는다. PCI-AIO의 채널 번호는 0에서 1까지 이다.

dwData : DAC 에 출력할 값을 설정한다.

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.

DAC_ConfigChannel

각각의 ADC 채널에 대하여 출력 범위를 설정 한다.

BOOL **DAC_ConfigChannel (int nModel, int nBoard, int nChannel, DWORD dwConfig)**

Parameters:

nModel : PCI-AIO 모델번호를 적는다.

nBoard : 현재 시스템에 장착되어 있는 보드 번호를 알려준다.
보드 번호는 보드의 DIP 스위치를 이용하여 설정한다.

nChannel : DAC 채널 값을 적는다. PCI-AIO01의 채널 번호는 0에서 1까지 이다.

dwConfig : 0 : Bipolar (-10V 에서 + 10V)
1: Unipolar (0V 에서 + 10V) 디폴트 값

Return Value:

함수 호출에 실패할 경우 “FALSE” 성공일 경우 “TRUE”을 리턴함.