PCIバス対応

デジタル積分フィルタ付 絶縁型汎用入出力ボード

PiDio デバイスドライバ

取扱説明書

2006. 7. 18　初版
2008. 10. 8　改訂
2012. 1. 16　改訂

対応ボード

PD5006P（入力 64 点）

PD5106P（出力 64 点）

PD5206P（入力 32 点／出力 32 点）

PD5306P（双方向 64 点）

NOVA electronics　株式会社 ノヴァ電子

■改訂履歴

版 数	改訂年月日	改訂内容
暫定版	2006年 6月20日	新規作成
初 版	2006年 7月18日	5章 評価ツールを追加。 一部の文章表現を変更。図、説明を追加。
改 訂	2008年10月 8日	C#対応部分を追加
改 訂	2012年 1月16日	2.3 節の注意を修正

はじめに

このたびは、PD5000Pシリーズをご検討いただきまして、ありがとうございます。

■ 安全にお使いいただくために

本製品を安全にお使いいただくために、本書に記述されている内容を必ずお守りください。
なお、注意事項をお守りいただかない場合、製品の故障、瑕疵担保責任、その他一切の保証をできかねる場合があります。
本製品をご使用いただく前に、必ず本書を熟読し理解した上でご使用ください。

また、本書の記載内容は、今後、機能の向上などのため予告なしに変更する場合があります。
最新の取扱説明書、ソフトウェアは、弊社ホームページ（URL: http://www.novaelec.co.jp/ ）からダウンロードできます。

■ マニュアルの併用

ボードの仕様等については、各ボードの取扱説明書を参照してください。

■ 本書で使用する用語

ポート	本ボードでは入／出力信号 8 点ごとにポート番号（0～7）を割り振り、本ドライバではそのポート番号を使用して入出力操作などを行うことができます。 ポート番号の詳細については、ボードの取扱説明書「コネクタピンアサイン」の章を参照してください。
入力ポート	入力信号のみのポートです。 PD5006P はポート0～7、PD5206P はポート0～3が入力ポートです。 パソコン起動時、PD5306P はポート0～7が入力ポートです。（Dio_SetIoDir 関数で変更可）
出力ポート	出力信号のみのポートです。 PD5106P はポート0～7、PD5206P はポート4～7が出力ポートです。
入出力混在ポート	入力信号と出力信号が混在するポートです。 PD5306P のみ Dio_SetIoDir 関数で設定することができます。
入力信号 Low	入力端子が外部電源 GND（GEX）と短絡された状態です。
入力信号 Hi	入力端子が開放された状態です。
出力信号 Low	オープンコレクタ出力トランジスタが ON し、シンク電流が流れ込む状態です。
出力信号 Hi	オープンコレクタ出力トランジスタが OFF し、シンク電流が流れない状態です。
信号の立ち上がり	信号が Low レベルから Hi レベルへ遷移することを指します。
信号の立ち下がり	信号が Hi レベルから Low レベルへ遷移することを指します。
ラッチ	外部ストローブ信号の立ち上がり・立ち下がり、あるいは入力同時ラッチ命令などをトリガとして、入力データを捕捉することを言います。

― 目 次 ―

1. 概　要

本ドライバは、ノヴァ電子製 PCI バス対応汎用入出力ボード共通ドライバです。
ボードの仕様は、各ボードの取扱説明書を参照してください。

1.1 対応ボード

本ドライバは、下記のボードに対応しています。1つの PC に複数の種類のボードをインストールできます。

対応ボード	入出力点数
PD5006P	入力 64 点
PD5106P	出力 64 点
PD5206P	入力 32 点／出力 32 点
PD5306P	双方向 64 点

1.2 対応 OS

本ドライバは、下記の OS に対応しています。

対応 OS
Windows 98
Windows 2000
Windows XP

1.3 対応言語

本ドライバは、下記の言語に対応しています。
アプリケーションから弊社が提供する DLL を呼び出す事で各ボードを制御できます。

対応言語	
Microsoft Visual C++ 6.0	Microsoft Visual Basic 2005
Microsoft Visual C++ .NET 2003	Microsoft Visual Basic 2008
Microsoft Visual C++ 2005	Microsoft Visual C# .NET 2003
Microsoft Visual C++ 2008	Microsoft Visual C# 2005
Microsoft Visual Basic 6.0	Microsoft Visual C# 2008
Microsoft Visual Basic .NET 2003	

1.4 最大ボード数

1つの PC に下表の全種類のボードを同時にインストールできます。
また、同じ型式のボードは同時に16枚までインストールできます。同じ型式のボードの識別はボード上のロータリスイッチで行います。(ロータリスイッチは0～Fまで16種類の番号を設定できます。)
ロータリスイッチの位置については、ボードの取扱説明書「基板外形」の章を参照してください。

ボード型式	同じ型式で 同時に使用できる最大ボード数
PD5006P	16
PD5106P	16
PD5206P	16
PD5306P	16

2. インストール

この章では、本ボードのパソコンへの組込みとドライバのインストール方法について説明します。

2.1 ドライバソフトウェアの準備

CD－ROMからドライバをインストールする場合は、PiDioドライバのCD－ROMを用意して下さい。
ホームページからダウンロードしたドライバをインストールする場合は、ダウンロードしたソフトウェアを解凍して下さい。

2.2 同一型式ボードを複数枚使用する場合の設定

同一型式のボードを1つのパソコンで複数枚使用するときは、ボードを個別に認識させる為に、2枚目以降のボードはボード上のロータリスイッチでボード番号を設定して下さい。
ロータリスイッチ（SW1）の位置は、ボードの取扱説明書「基板外形」の章を参照してください。
ロータリスイッチは0～Fのいずれかを設定できます。ロータリスイッチの番号は他の同一型式ボードと重複しないように設定して下さい。

出荷時は、ボード上のロータリスイッチに0が設定されています。

異なる型式のボードを複数枚使用する場合には、この設定は必要ありません。例えば、PD5206P1枚と PD5306P1枚を1つのパソコンで同時に使用する場合、それぞれのボードはロータリスイッチ0のままで使用できます。

2.3 パソコンへの本ボードの組込み

注 意:
パソコンへの取り付け作業は必ずパソコンの電源を切断してから行ってください。
パソコンの電源を入れたままで取り付け作業を行いますと、基板の内部回路などを破損する場合があります。

① パソコン本体の電源がOFFであることを確認してから、外装カバー、スロットカバー等を外します。
② 空いている拡張スロットへ本ボードを差し込みます。基板のエッジコネクタをパソコンのPCIバスコネクタに正しく挿入してください。（本ボードを設置できる PCI バスコネクタについては、ボードの取扱説明書「基板外形」の章を参照してください。）
③ 取付金具をネジ止めしてください。この時キチンとねじを締めないと後で抜け落ちたりするなどして、ショートや故障、誤動作の原因となります。
④ パソコン本体の外装カバーを元通りに取り付けます。

2.4 デバイスドライバのインストール

各OSに対応したインストール手順を説明します。

2.4.1 Windows98

① 2.1 の方法でインストールするデバイスドライバを準備して下さい。

② 2.2, 2.3 を実行し、ボードが確実にパソコンに組み込まれているか確認してください。

③ パソコン本体の電源をONし、Windows98 を起動します。

④ すぐに下の画面が表示されますので「次へ」をクリックします。

⑤ 下の画面が表示されますので、「使用中のデバイスに最適なドライバを検索する(推奨)」を選択し「次へ」をクリックします。

⑥ 下の画面が表示されますので、「検索場所の指定」をチェックします。
ドライバをCD－ROMからインストールする場合は、パソコンにCD－ROMをセットし、CD－ROMが認識されてから次の動作に進みます。
参照ボタンを押し、提供CD－ROMの Driver フォルダ（提供CD－ROMがDドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアの Driver フォルダを選択し、テキストボックスにフォルダが表示されたら、「次へ」をクリックします。

⑦ 下の画面にボード名（PD5306P 等）が表示されます。「次へ」をクリックします。

⑧ デバイスドライバのコピーが完了すると下の画面が表示されますので、「完了」をクリックします。

⑨ 以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「デバイスマネージャ」画面（下記左画面）を開き、「NOVA」の下のボード名（PD5306P 等）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。

また、デバイスドライバのインストール完了後はパソコンの起動時に手順④のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は 2.4 の手順から再度インストールをやり直してください。

※同一型式ボードを複数枚使用する場合、ボードのロータリスイッチ番号が重複していると、ここでエラーが表示されます。

インストールが完了したら、「コントロールパネル」－「システム」－「デバイスマネージャ」でボード名をダブルクリックして「リソース」タブを表示し、リソース(I／Oアドレス、割り込みレベル)の設定、競合の有無を確認してくだい。

2.4.2 Windows2000

デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。

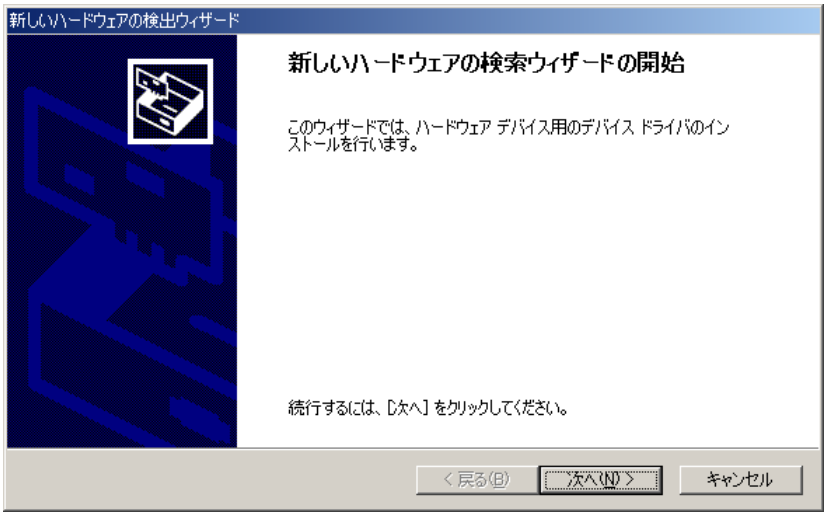
① 2.1 の方法でインストールするデバイスドライバを準備して下さい。

② 2.2, 2.3 を実行し、ボードが確実にパソコンに組み込まれているか確認してください。

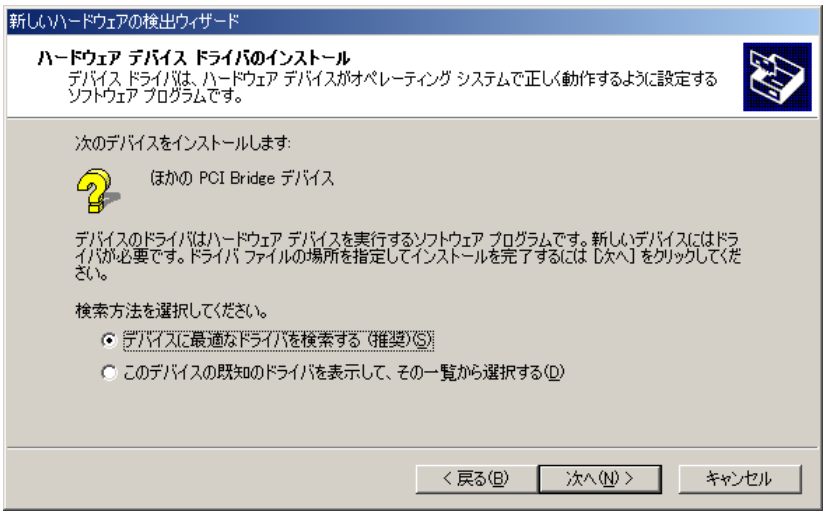
③ パソコン本体の電源をONし、Windows2000 を起動します。

④ アドミニストレーター権限を持ったユーザーでログインしてください。

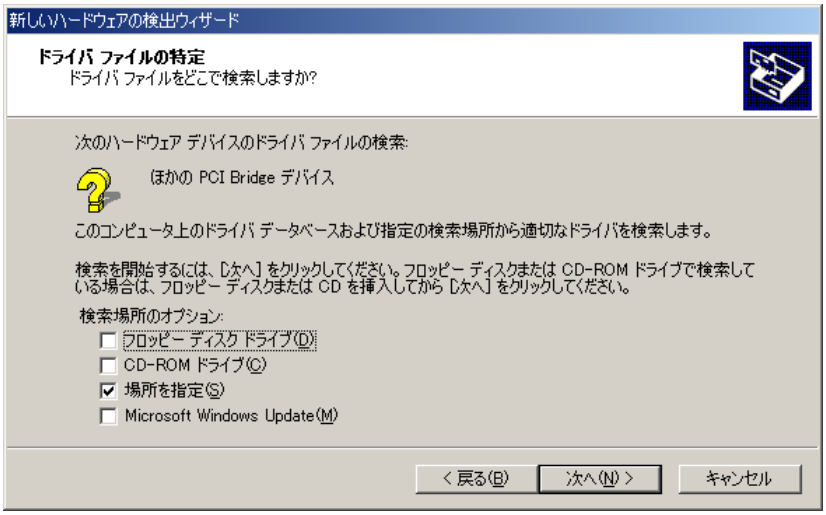
⑤ すぐに下の画面が表示されますので「次へ」をクリックします。



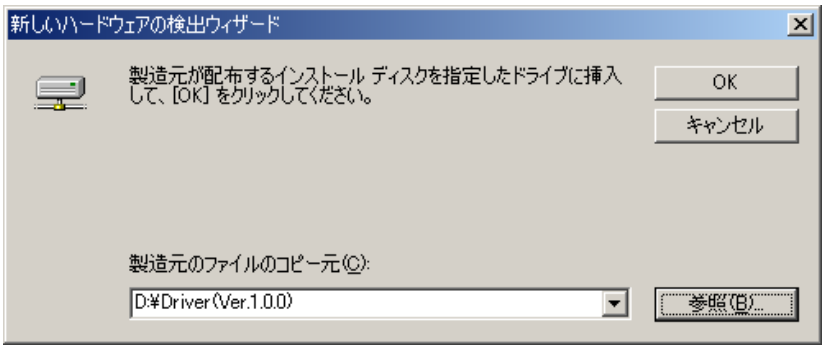
⑥ 下の画面が表示されますので、「デバイスに最適なドライバを検出する(推奨)」を選択し「次へ」をクリックします。



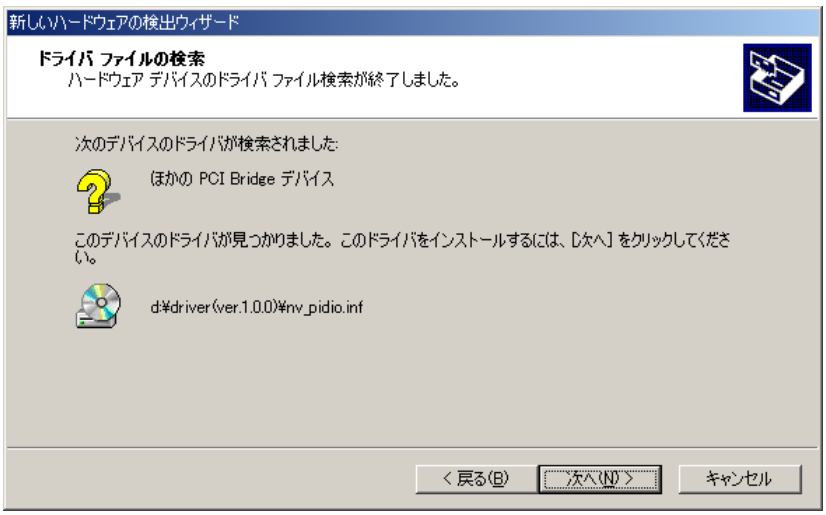
⑦ 下の画面が表示されますので、「場所を指定」をチェックし「次へ」をクリックします。



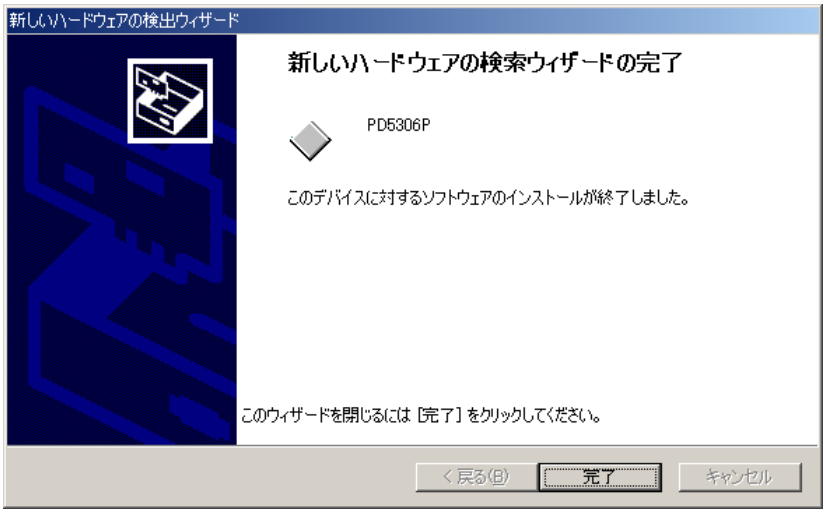
⑧ 下の画面が表示されます。ドライバをCD－ROMからインストールする場合は、パソコンにCD－ROMをセットし、CD－ROMが認識されてから次の動作に進みます。
参照ボタンを押し、提供CD－ROMの Driver フォルダ（提供CD－ROMがDドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアの Driver フォルダを選択し、テキストボックスにフォルダが表示されたら、OKを押してください。



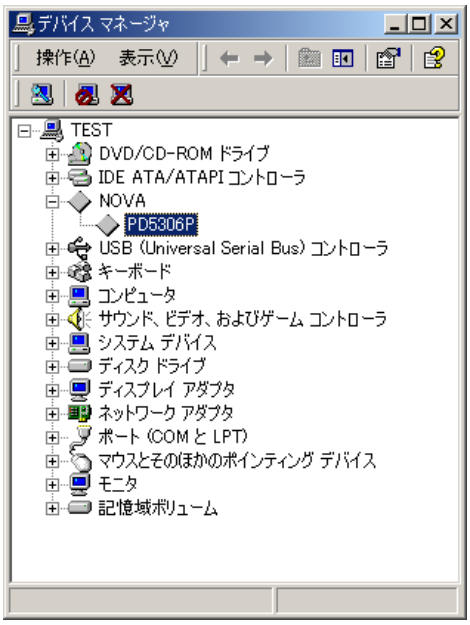
⑨ デバイスドライバの情報ファイルが発見されると確認の画面が出ますので、「¥driver¥nv_pidio.inf」が表示されたことを確認して、「次へ」をクリックします。

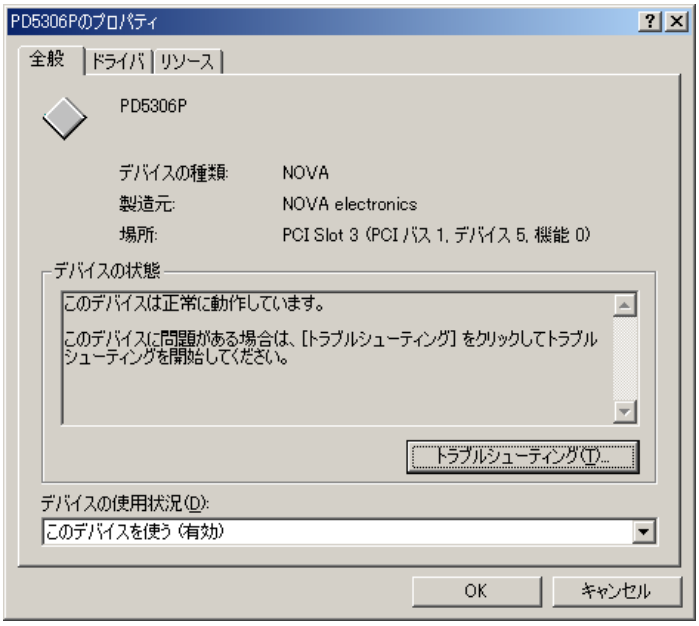


⑩ デバイスドライバのインストールが正常に完了すると、下の画面が表示されますので、「完了」をクリックします。



⑪ 以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、「NOVA」の下のボード名（PD5306P 等）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。

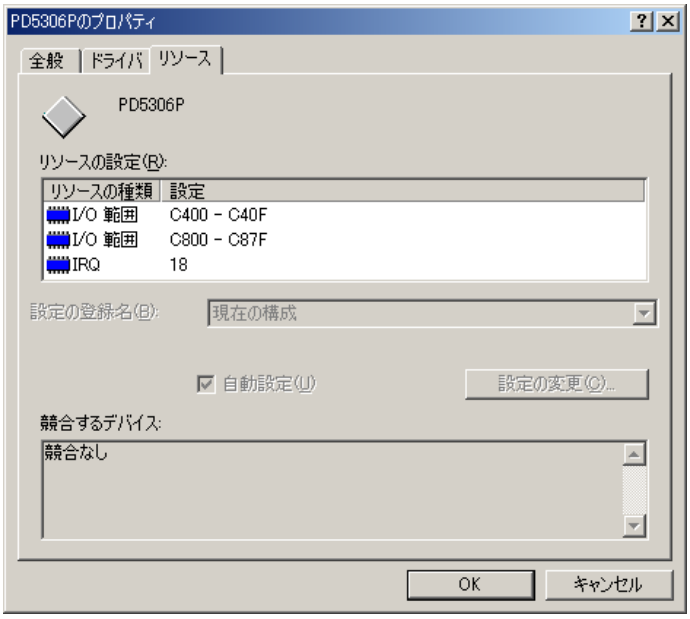




また、デバイスドライバのインストール完了後はパソコンの起動時に手順⑤のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は 2.4 の手順から再度インストールをやり直してください。

※同一型式ボードを複数枚使用する場合、ボードのロータリスイッチ番号が重複していると、ここでエラーが表示されます。

インストールが完了したら、「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」でボード名をダブルクリックして「リソース」タブを表示し、リソース(I／Oアドレス、割り込みレベル)の設定、競合の有無を確認してください。



2.4.3 WindowsXP

デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。

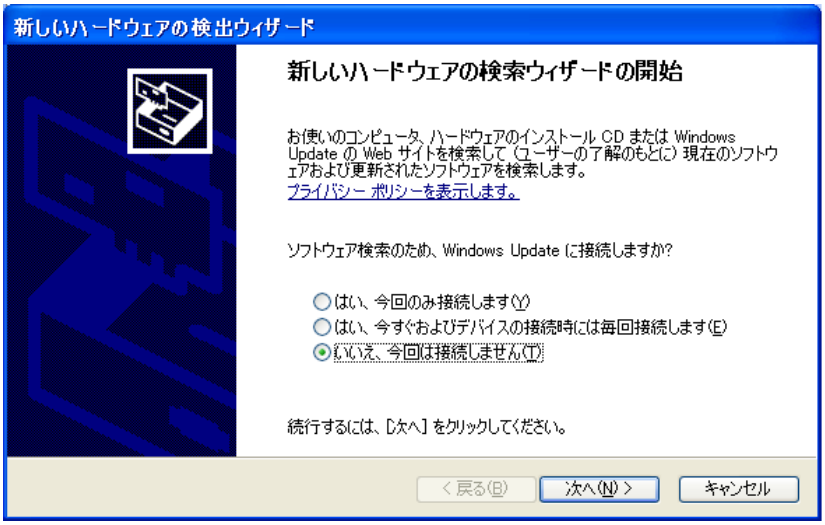
① 2.1 の方法でインストールするデバイスドライバを準備して下さい。

② 2.2, 2.3 を実行し、ボードが確実にパソコンに組み込まれているか確認してください。

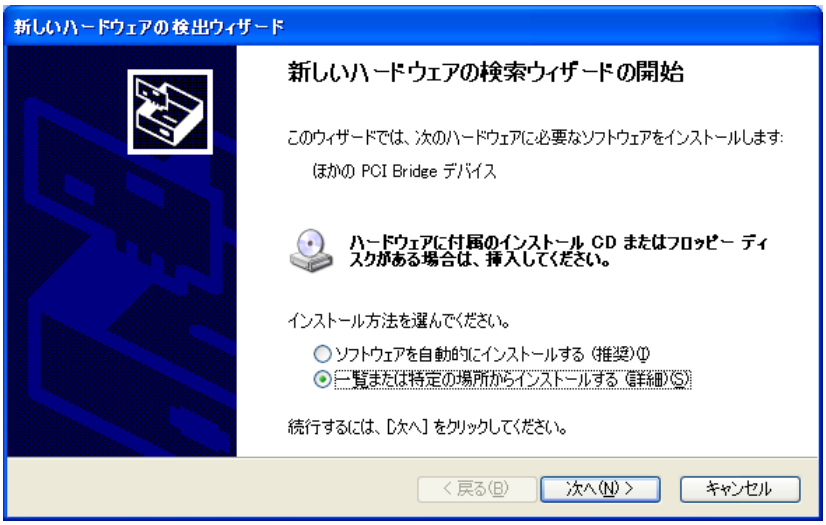
③ パソコン本体の電源をONし、WindowsXP を起動します。

④ アドミニストレーター権限を持ったユーザーでログインしてください。

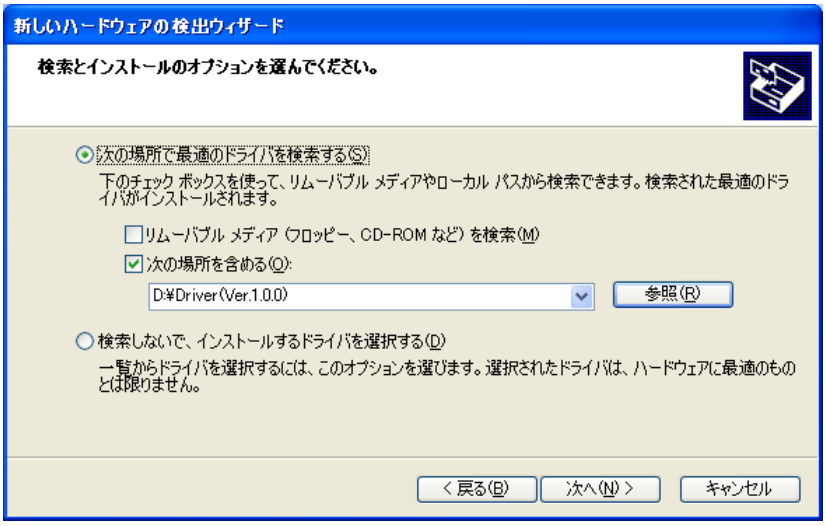
⑤ WindowsXP サービスパック2の場合は、下の画面が表示されますので、「いいえ、今回は接続しません」を選択し、「次へ」をクリックします。
WindowsXP サービスパック1の場合は、この画面は表示されませんので、次に進んで下さい。



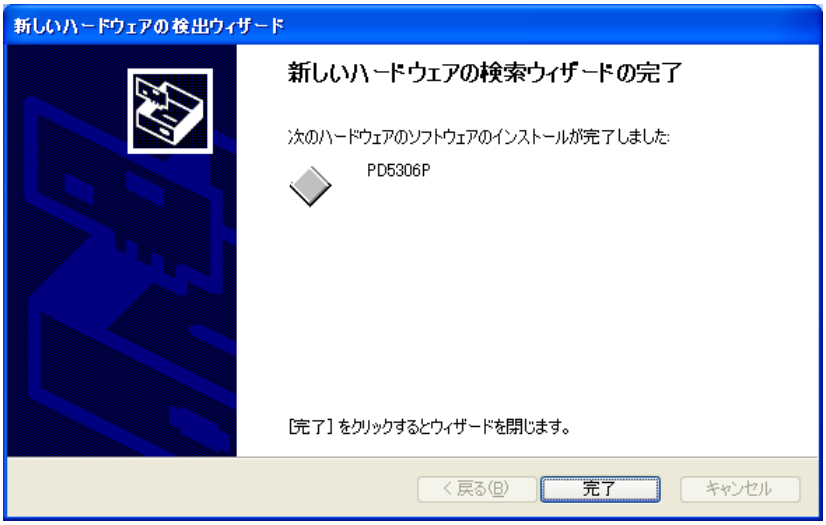
⑥ 次に下の画面が表示されますので「一覧または特定の場所からインストールする」を選択し、「次へ」をクリックします。



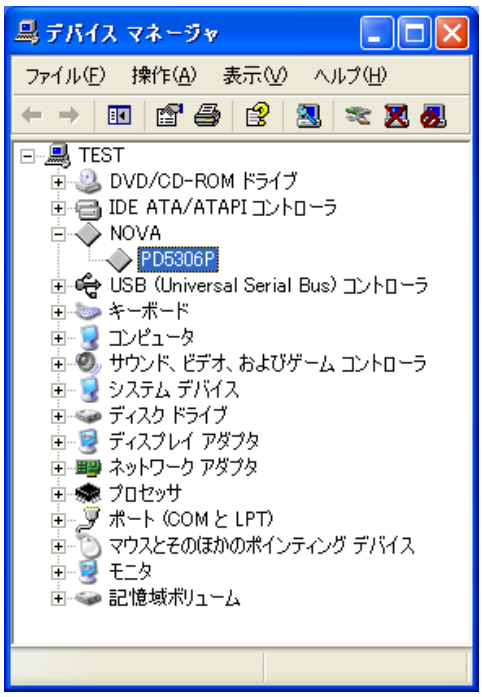
⑦ 下の画面が表示されますので、「次の場所で最適のドライバを検索する」を選択し、「次の場所を含める」をチェックします。ドライバをCD－ROMからインストールする場合は、パソコンにCD－ROMをセットし、CD－ROMが認識されてから次の動作に進みます。
参照ボタンを押し、提供CD－ROMの Driver フォルダ（提供CD－ROMがDドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアの Driver フォルダを選択し、テキストボックスにフォルダが表示されたら、「次へ」をクリックします。

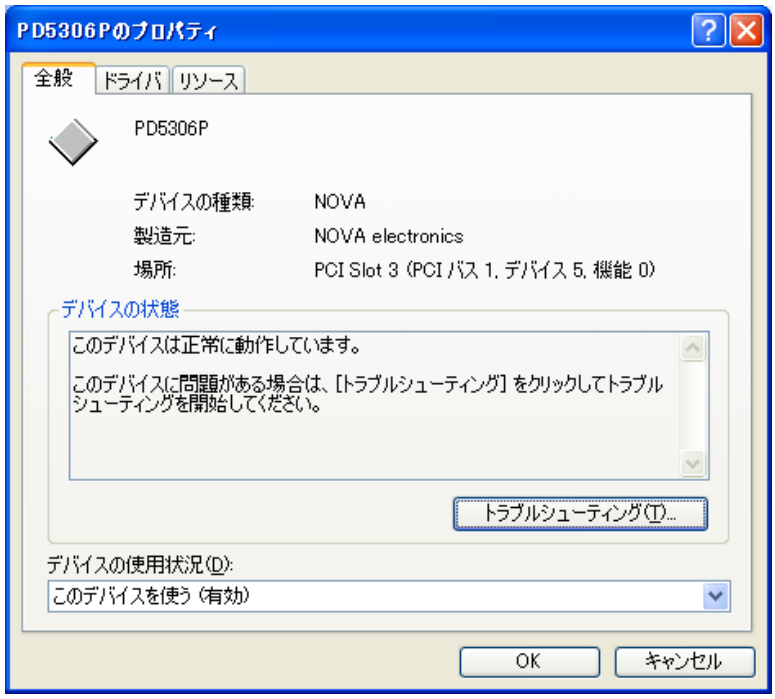


⑧ デバイスドライバのインストールが正常に完了すると、下の画面が表示されますので、「完了」をクリックします。



⑨ 以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、「NOVA」の下のボード名（PD5306P 等）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。

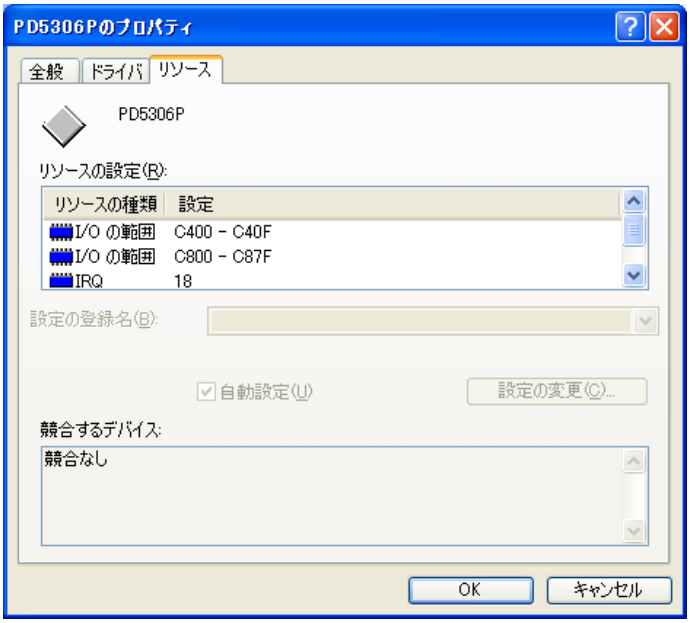




また、デバイスドライバのインストール完了後はパソコンの起動時に手順⑤、⑥のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は 2.4 の手順から再度インストールをやり直してください。

※同一型式ボードを複数枚使用する場合、ボードのロータリスイッチ番号が重複していると、ここでエラーが表示されます。

インストールが完了したら、「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」でボード名をダブルクリックして「リソース」タブを表示し、リソース(I／Oアドレス、割り込みレベル)の設定、競合の有無を確認してください。



2.5 ボードの取り外し

① パソコン本体の電源がOFFであることを確認してから、外装カバー、スロットカバー等を外します。
② ボードを止めているビスを外します。
③ 本ボードを指先でつまんで軽く左右にゆするようにしながら引き出します。

2.6 デバイスドライバの更新

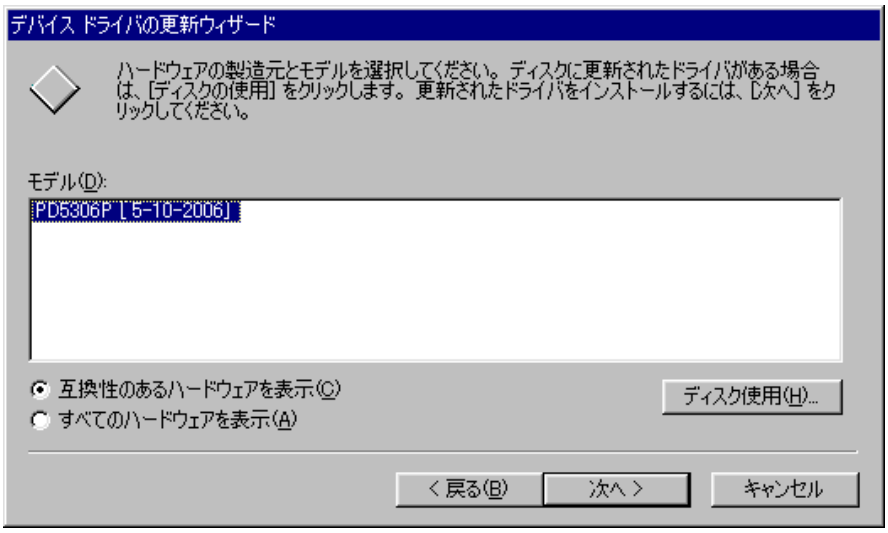
デバイスドライバのバージョンが新しくなった場合、次の手順でドライバの更新を行って下さい。
以下に、各OSに対応した更新手順を説明します。

2.6.1 Windows98

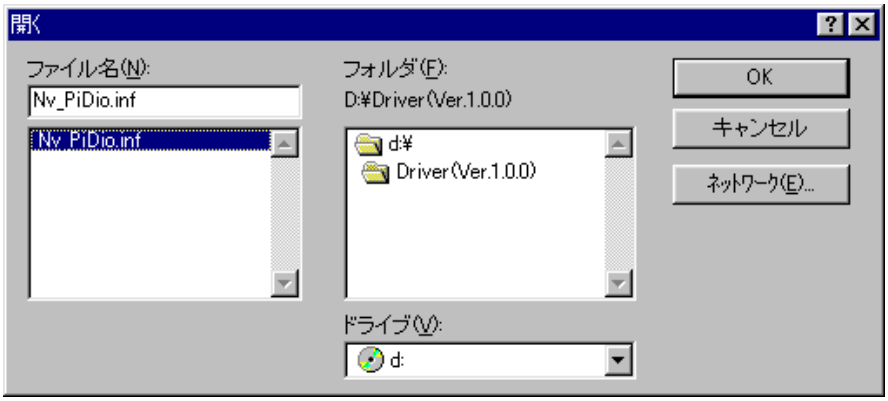
① 「コントロールパネル」－「システム」－「デバイスマネージャ」画面（下記左画面）を開き、「NOVA」の下のボード名（PD5306P 等）をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
② 次に「ドライバの更新」ボタンをクリックし、その次の画面で「次へ」ボタンをクリックします。

③ 下記画面が表示されますので、「特定の場所にあるすべてのドライバの一覧を作成し、インストールするドライバを選択する」を選択し、「次へ」ボタンをクリックします。

④ 下記画面で、「ディスク使用」ボタンをクリック後、「参照」ボタンをクリックします。



⑤ 下記画面で、更新するドライバソフトのフォルダ(¥Driver)を選択後、「OK」ボタンをクリックし、その次の画面でも「OK」ボタンをクリックします。



⑥ 下記画面で、「次へ」ボタンをクリック後、その次の画面でも「次へ」ボタンをクリックすると、ドライバが更新されます。

⑦ 正常に更新されると、下記画面が表示されます。

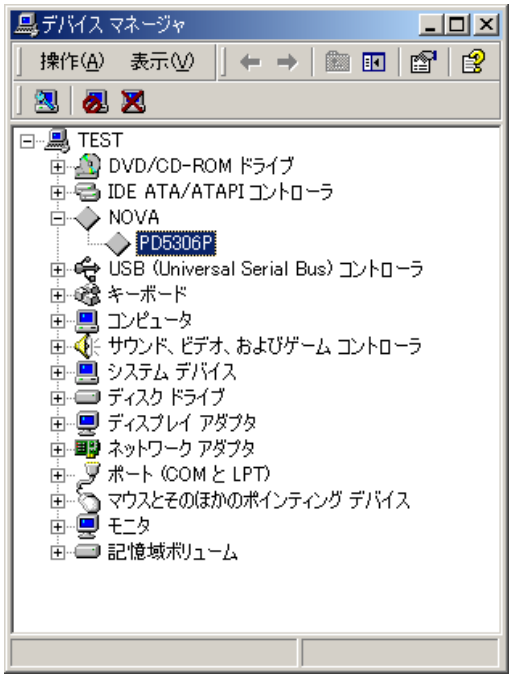
⑧ 「コントロールパネル」－「システム」－「デバイスマネージャ」でボード名をダブルクリックし、「ドライバ」タブで「ドライバの詳細」ボタンをクリックすると、下記画面が表示されます。
下記画面で、更新したドライバのファイルバージョンを確認します。
¥Driver¥Version.txt ファイルの「2. ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している2つのファイルのバージョンが正しいか確認して下さい。

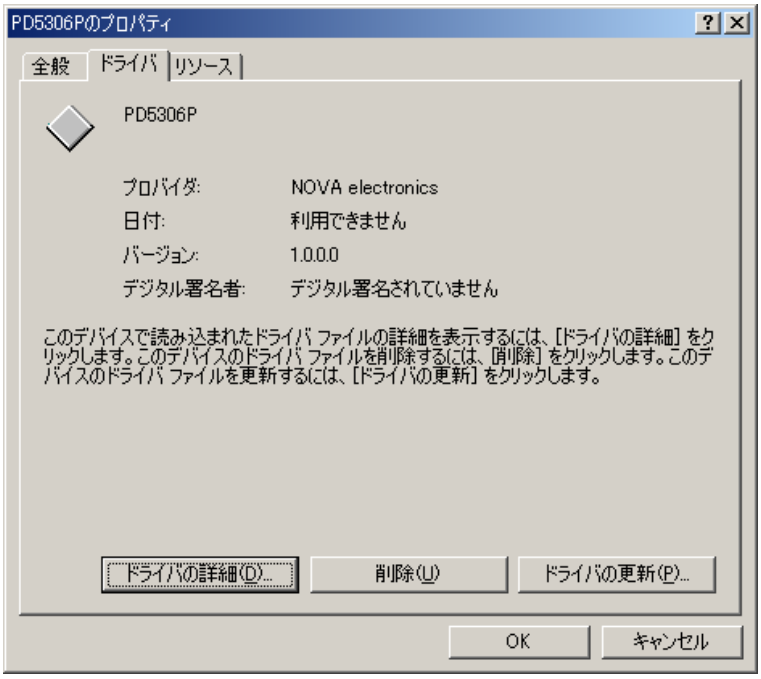
⑨ ボードが複数ある場合は、「デバイスマネージャ」画面の「NOVA」の下に表示される全てのボードについて、①からの手順でドライバを更新して下さい。

⑩ パソコンを再起動して下さい。これで更新作業は終了です。

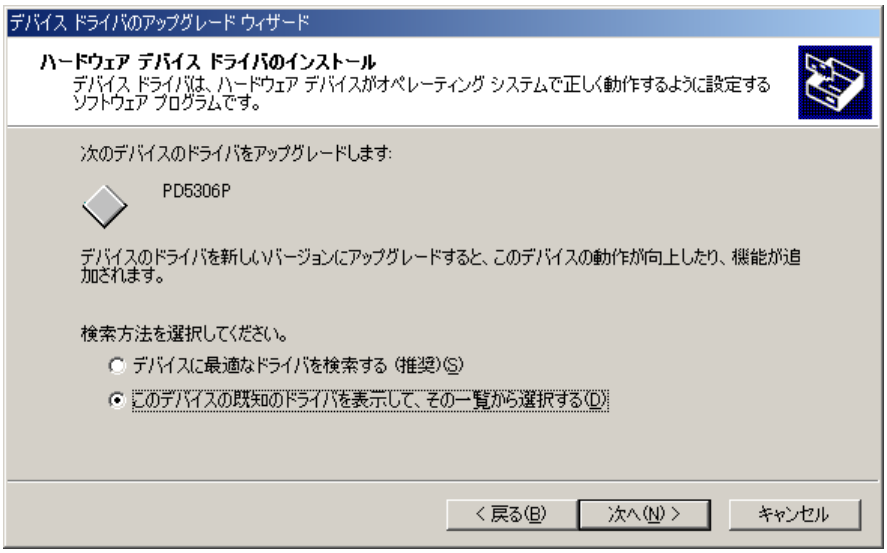
2.6.2 Windows2000

① 「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、「NOVA」の下のボード名（PD5306P 等）をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
② 次に「ドライバの更新」ボタンをクリックし、次に表示された画面では「次へ」ボタンをクリックします。

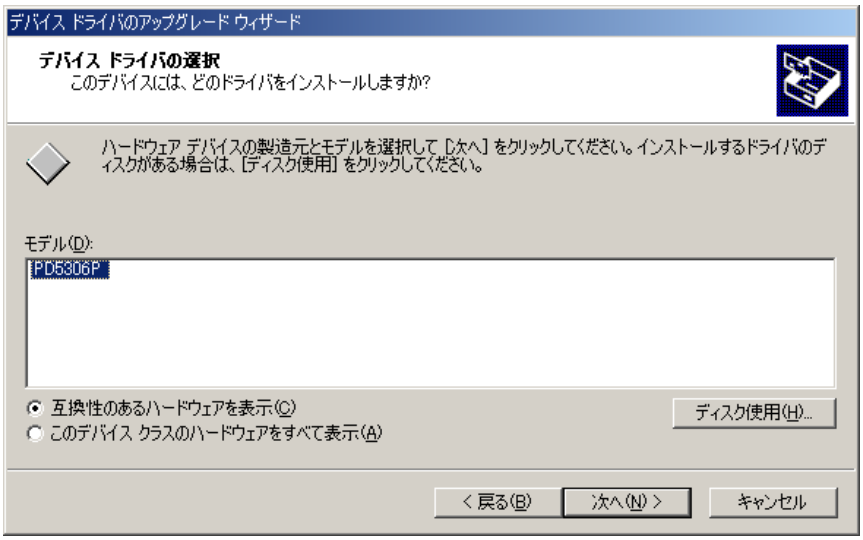




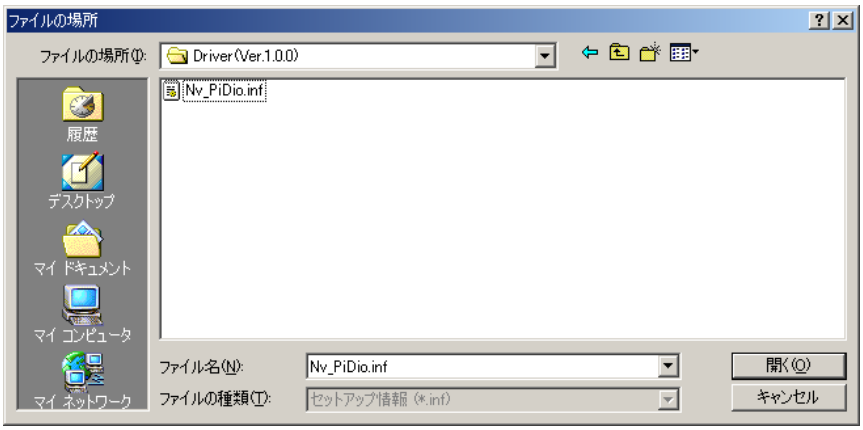
③ 下記画面が表示されますので、「このデバイスの既知のドライバを表示して、その一覧から選択する」を選択し、「次へ」ボタンをクリックします。



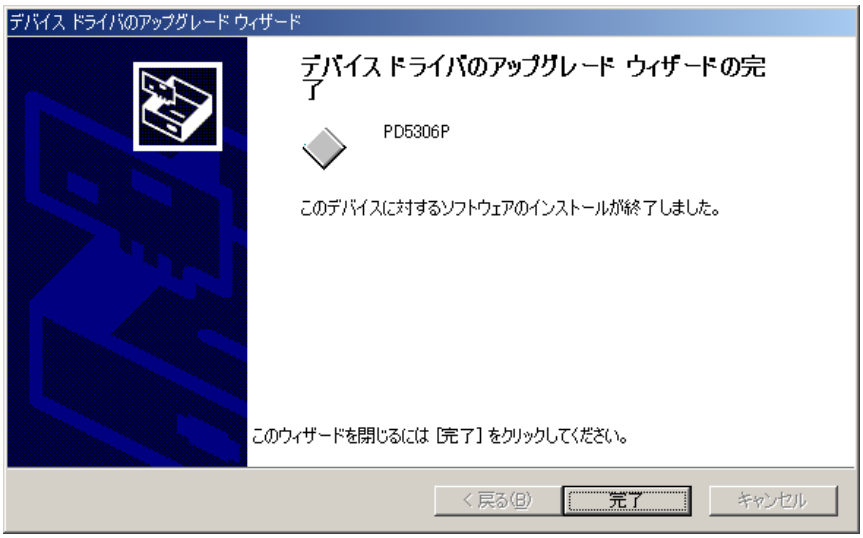
④ 下記画面で、「ディスク使用」ボタンをクリック後、「参照」ボタンをクリックします。



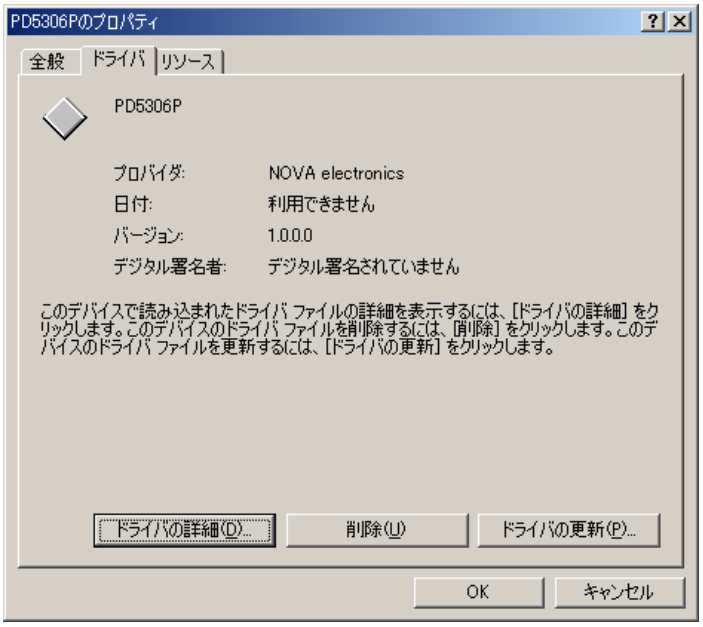
⑤ 下記画面で、更新するドライバソフトのフォルダ(¥Driver)を選択し、「開く」ボタンクリック後、「OK」ボタンをクリックします。その次の画面では「次へ」ボタンクリック後、更に次の画面でも「次へ」ボタンをクリックします。



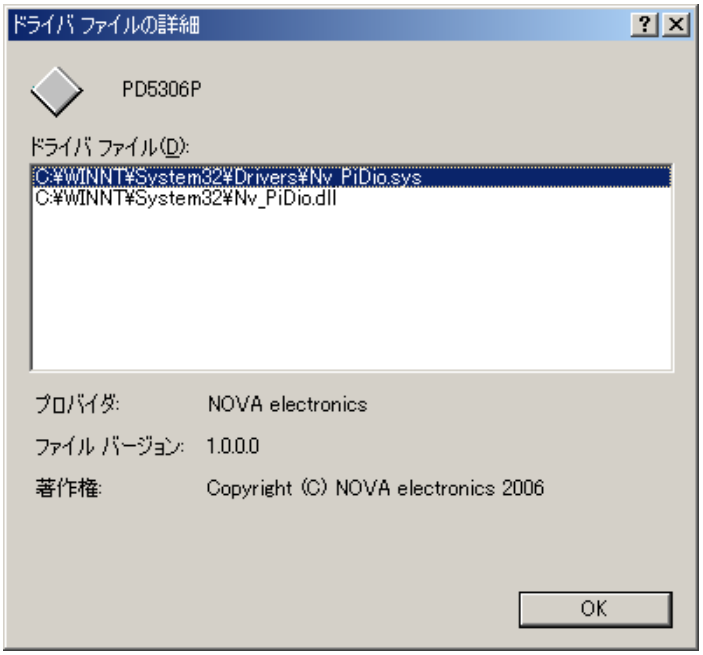
⑥ 正常に更新されると、下記画面が表示されます。



⑦ 下記画面(デバイスマネージャの更新したボードのプロパティ画面)で、更新したドライバのバージョンを確認します。
¥Driver¥Version.txt ファイルの「1. ドライババージョン」のバージョンが下記画面に表示されますので正しいか確認して下さい。 次に「ドライバの詳細」ボタンをクリックします。



⑧ 下記画面で、更新したドライバのファイルバージョンを確認します。
¥Driver¥Version.txt ファイルの「2. ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している2つのファイルのバージョンが正しいか確認して下さい。

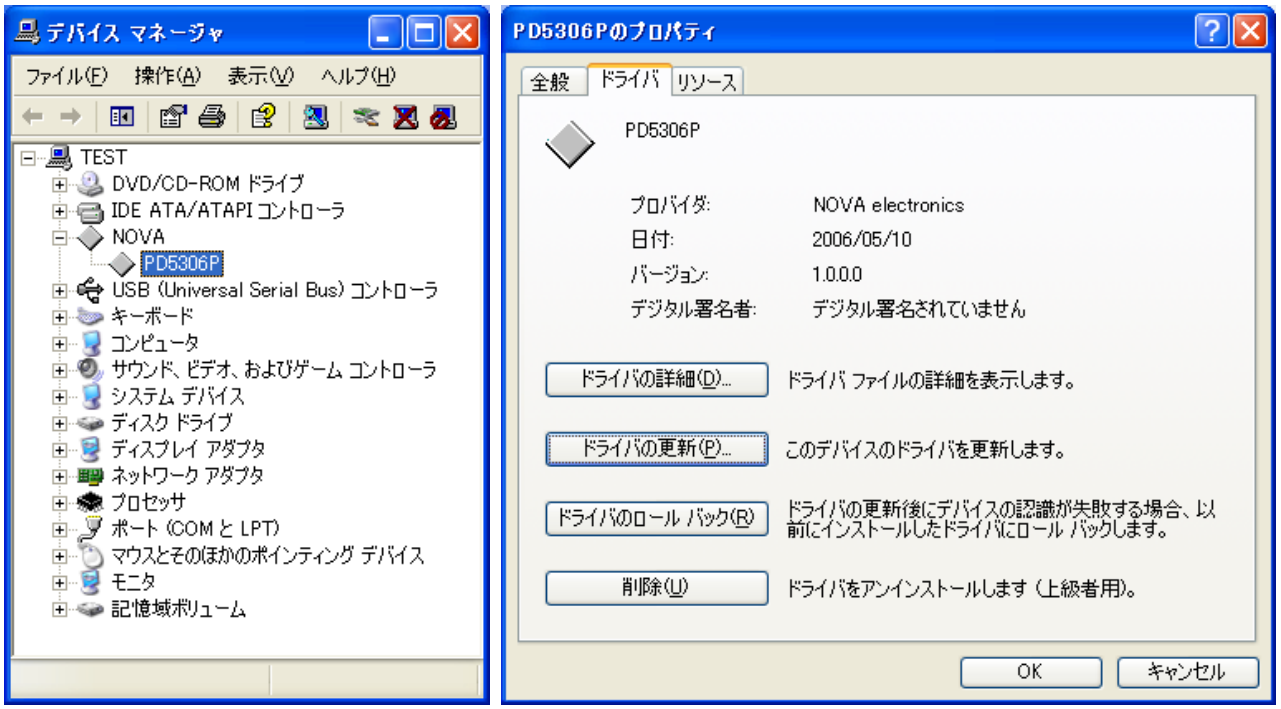


⑨ ボードが複数ある場合は、「デバイスマネージャ」画面の「NOVA」の下に表示される全てのボードについて、①からの手順でドライバを更新して下さい。

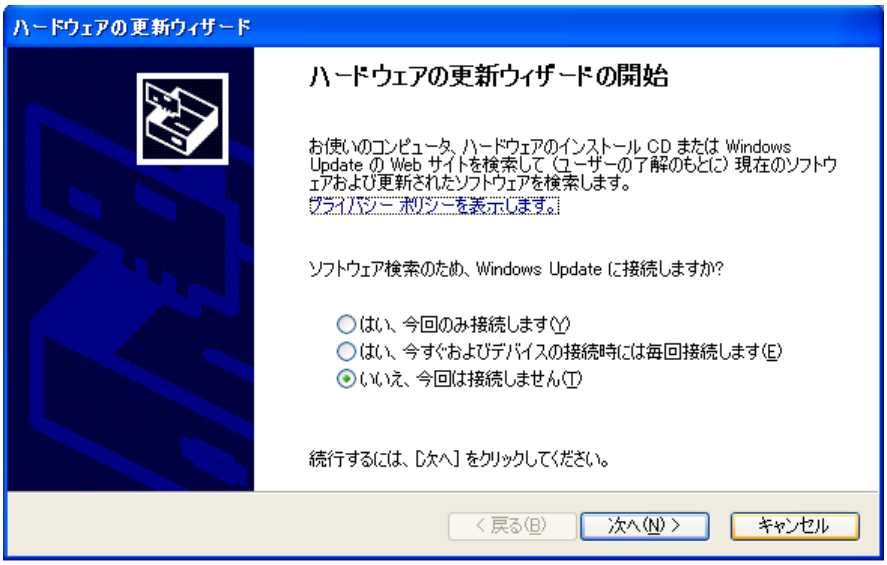
⑩ パソコンを再起動して下さい。これで更新作業は終了です。

2.6.3 WindowsXP

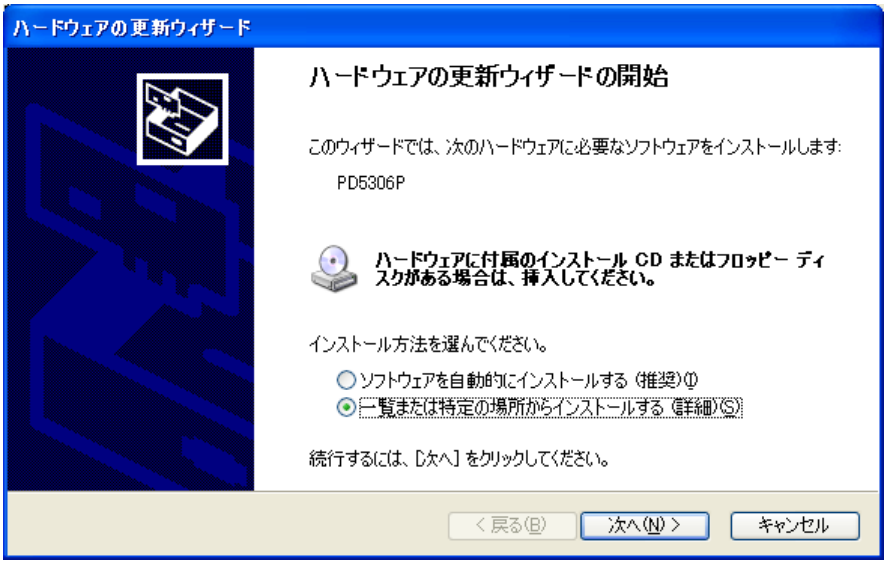
① 「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、「NOVA」の下のボード名（PD5306P 等）をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
② 次に「ドライバの更新」ボタンをクリックします。



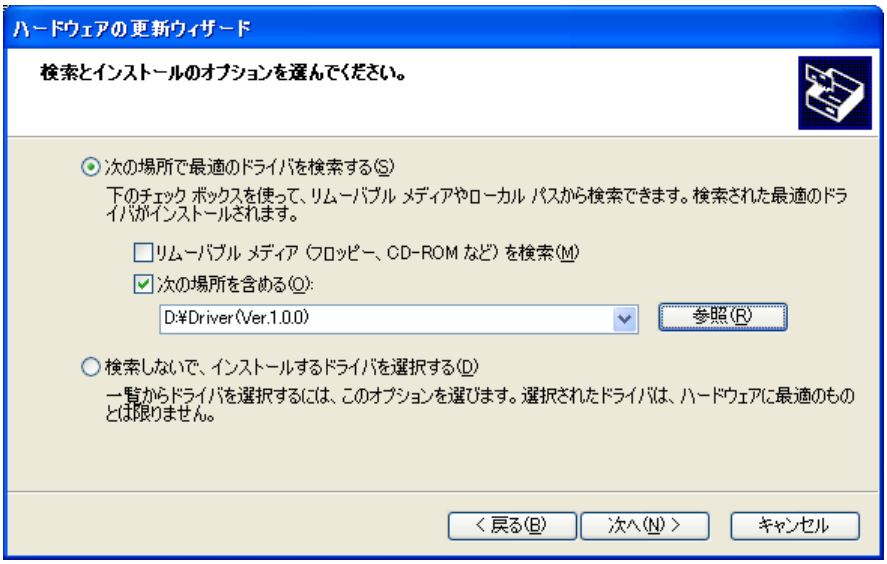
③ WindowsXP サービスパック2の場合は、下の画面が表示されますので、「いいえ、今回は接続しません」を選択し、「次へ」をクリックします。
WindowsXP サービスパック1の場合は、この画面は表示されませんので、次に進んで下さい。



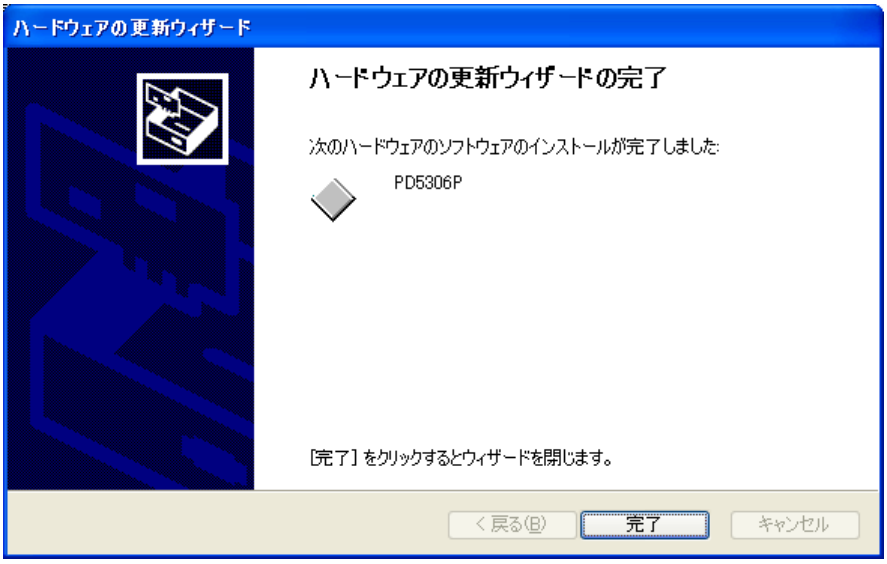
④ 下記画面で、「一覧または特定の場所からインストールする」を選択し、「次へ」ボタンをクリックします。



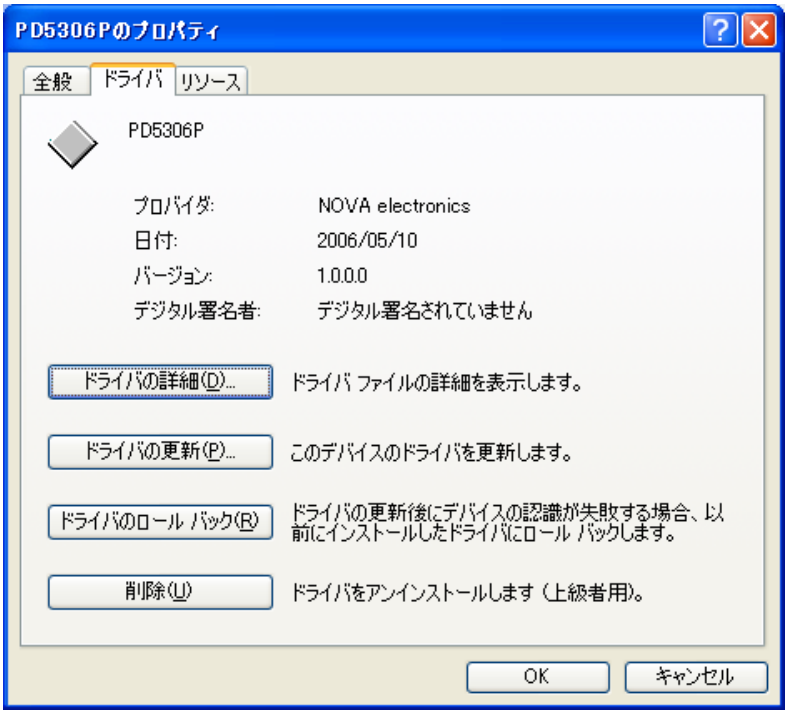
⑤ 下記画面で、「次の場所で最適のドライバを検索する」を選択し、「次の場所を含める」をチェックします。
参照ボタンから更新するドライバソフトのフォルダ(¥Driver)を選択し、下記画面の「次へ」をクリックします。



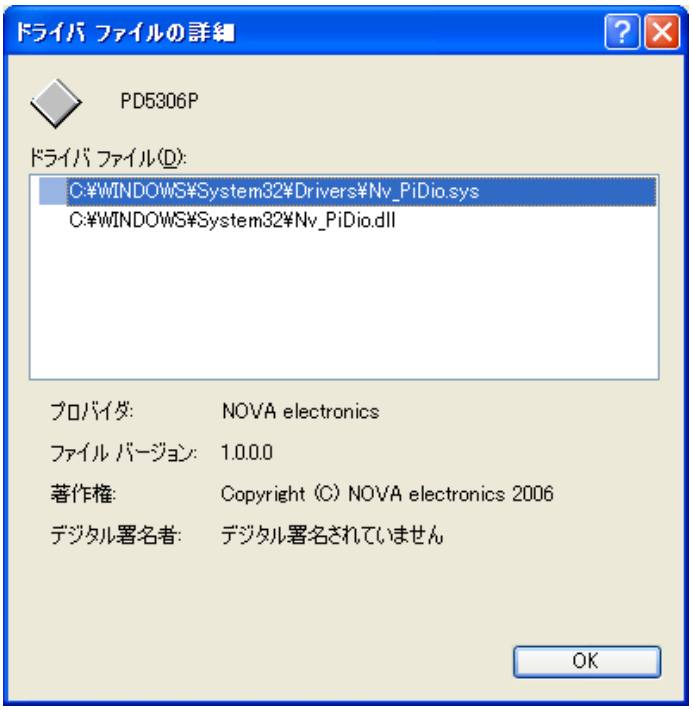
⑥ 正常に更新されると、下記画面が表示されます。



⑦ 下記画面(デバイスマネージャの更新したボードのプロパティ画面)で、更新したドライバのバージョンを確認します。¥Driver¥Version.txt ファイルの「1. ドライババージョン」のバージョンが下記画面に表示されますので正しいか確認して下さい。次に「ドライバの詳細」ボタンをクリックします。



⑧ 下記画面で、更新したドライバのファイルバージョンを確認します。
¥Driver¥Version.txt ファイルの「2. ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している2つのファイルのバージョンが正しいか確認して下さい。



⑨ ボードが複数ある場合は、「デバイスマネージャ」画面の「NOVA」の下に表示される全てのボードについて、①からの手順でドライバを更新して下さい。

⑩ パソコンを再起動して下さい。これで更新作業は終了です。

3. 操作方法

この章で説明している関数の詳細については、4.4 APIを参照してください。
各機能の仕様については、ボードの取扱説明書も合わせて参照してください。

3.1 一覧

No.	項目		操作内容	関数名
1	開始・終了処理		開始処理	Dio_Open
			終了処理	Dio_Close, Dio_CloseAll
2	通常の入力／出力動作		入力／出力の設定（PD5306P のみ）	Dio_SetIoDir
			入力論理設定	Dio_SetInputLogic
			入力フィルタの指定	Dio_SetInputFilter
			フィルタ時定数の設定	Dio_SetFilterTime
			入力値・出力値の読み出し	Dio_In_1Bit, Dio_In_1Byte, Dio_In_2Byte Dio_In_4Byte, Dio_In_8Byte Dio_In_nBit, Dio_In_nByte
			出力値の書き込み（信号出力）	Dio_Out_1Bit, Dio_Out_1Byte, Dio_Out_2Byte Dio_Out_4Byte, Dio_Out_8Byte Dio_Out_nBit, Dio_Out_nByte
3	入力同時ラッチ	INSTB	外部ストローブ信号変化方向設定	Dio_SetStrobeDir
			入力同時ラッチ有効設定（INSTB、命令）	Dio_SetSmlInStbCmd
			入力同時ラッチデータ読み出し	Dio_ReadLatch
		命令	入力同時ラッチ有効設定（INSTB、命令）	Dio_SetSmlInStbCmd
			入力同時ラッチ命令	Dio_ExecSmlInLatch
			入力同時ラッチデータ読み出し	Dio_ReadLatch
		タイマ（PD5206P のみ）	入力同時ラッチ有効設定（タイマ）	Dio_SetSmlInTimer
			タイマ値設定、タイマ起動、停止など	
			入力同時ラッチデータ読み出し	Dio_ReadLatch
4	出力同時セット	OTSTB	外部ストローブ信号変化方向設定	Dio_SetStrobeDir
			出力同時セット有効設定（OTSTB、命令）	Dio_SetSmlOutStbCmd
			出力値の書き込み	Dio_Out_1Byte, Dio_Out_2Byte Dio_Out_4Byte, Dio_Out_8Byte, Dio_Out_nByte
		命令	出力同時セット有効設定（OTSTB、命令）	Dio_SetSmlOutStbCmd
			出力値の書き込み	Dio_Out_1Byte, Dio_Out_2Byte Dio_Out_4Byte, Dio_Out_8Byte, Dio_Out_nByte
			出力同時セット命令	Dio_ExecSmlOut
		タイマ（PD5206P のみ）	出力同時セット有効設定（タイマ）	Dio_SetSmlOutTimer
			出力値の書き込み	Dio_Out_1Byte, Dio_Out_2Byte Dio_Out_4Byte, Dio_Out_8Byte, Dio_Out_nByte
			タイマ値設定、タイマ起動、停止など	
5	入力変化		入力変化有効設定	Dio_SetInTransitionMode
			入力変化方向設定	Dio_SetInTransitionDir
			入力変化情報読み出し	Dio_ReadInTransition
			入力変化情報クリア	Dio_ClearInTransition
6	タイマ		タイマ値設定	Dio_SetTimer
			タイマ起動（単一、連続）	Dio_StartTimer
			タイマ停止（即停止、サイクル停止）	Dio_StopTimer
			タイマ実行中の動作タイマ値読み出し	Dio_ReadCurrentTimer

No.	項目	操作内容	関数名
7	割り込み	オープン時に割り込み使用モード指定	Dio_Open
		割り込み通知設定(関数呼び出し通知)	Dio_SetEventFunc
		割り込み通知設定(メッセージ送信通知)	Dio_SetEventMsg
		割り込み通知設定の解除	Dio_ResetEvent
		割り込み設定	Dio_SetIntrptMode
		割り込み要因読み出し(入力変化情報は除く)	Dio_ReadIntrpt
		入力変化情報読み出し	Dio_ReadInTransition
8	設定値の読み出し	ボードに設定したモード・パラメータ設定値の読み出し	モード・パラメータ設定読み出し関数
9	エラーコード	エラーコード取得	Dio_GetLastError
		エラーコードクリア	Dio_ClearLastError

3.2 詳細

3.2.1 開始・終了処理

プログラム開始時などボードの使用を開始するときは、Dio_Open関数を一度実行してください。
1つのボードに対して複数のアプリケーションから同時にオープンすることができますが、そのボードに対して割り込みを使用したオープンができるのは1つのアプリケーションのみです。

プログラム終了時などボードの使用を終了するときは、Dio_Close、またはDio_CloseAll関数を実行してください。クローズ後、再度ボードを使用する場合はDio_Open関数を実行してください。

3.2.2 通常の入力／出力動作

入力／出力に必要なモード設定、入力値・出力値の読み出し、出力値の書き込み（信号出力）の説明をします。

（1） 入力／出力の設定（PD5306P のみ）

各信号を入力として使用するか、出力として使用するかを1点ごとに設定します。
入力／出力の設定（Dio_SetIoDir）関数で設定します。

（2） 入力論理設定

入力信号の Hi レベルを入力値1とするか、Low レベルを入力値1とするかの論理を1点ごとに設定します。
入力論理設定（Dio_SetInputLogic）関数で設定します。

（3） 入力フィルタの指定

入力ポート（または入力に設定したポート）について、積分フィルタを使用するか、スルーで通す（フィルタなし）かを4点ごと（ポートの上位4点／下位4点）に指定します。積分フィルタを使用する場合には、本ボードは 3 種類のフィルタ時定数を持っていますので、3 種類のうちのどの時定数を使用するかを指定します。入力フィルタ指定（Dio_SetInputFilter）関数で設定します。

（4） フィルタ時定数の設定

本ボードの積分フィルタの時定数を設定します。時定数は 3 種類あります。それぞれの時定数 1,2,3 について、遅延時間1μsecから 32msecの範囲で設定します。フィルタ時定数の設定（Dio_SetFilterTime）関数で設定します。

（5） 入力値・出力値の読み出し

入力信号の場合は現在の入力信号の状態（フィルタ通過後の論理設定された入力値）を、出力信号の場合は現在の出力値を読み出すことができます。
信号番号を指定した1点読み出しと、先頭ポート番号を指定した8，16，32，64点の読み出しを行うことができます。

関数名	内容
Dio_In_1Bit	1ビット読み出し（1点）
Dio_In_1Byte	1バイト読み出し（8点）
Dio_In_2Byte	2バイト読み出し（16点）
Dio_In_4Byte	4バイト読み出し（32点）
Dio_In_8Byte	8バイト読み出し（64点）
Dio_In_nBit	任意複数ビット読み出し
Dio_In_nByte	任意複数バイト読み出し

（6） 出力値の書き込み（信号出力）

出力ポート（または出力に設定したポート）に対して、信号番号を指定した1点出力と、先頭ポート番号を指定した8，16，32，64点の出力を行うことができます。出力同時セットを無効にしている場合は、下記関数で出力値を書いたときに信号出力されます。出力同時セットを有効にしている場合については、出力同時セットの章を参照してください。

関数名	内容
Dio_Out_1Bit	1ビット出力（1点）
Dio_Out_1Byte	1バイト出力（8点）
Dio_Out_2Byte	2バイト出力（16点）
Dio_Out_4Byte	4バイト出力（32点）
Dio_Out_8Byte	8バイト出力（64点）
Dio_Out_nBit	任意複数ビット出力
Dio_Out_nByte	任意複数バイト出力

3.2.3 入力同時ラッチ

すべての入力信号を同時にラッチする機能です。入力同時ラッチには、INSTB 信号、命令、タイマの3つの方法があります。PD5206P の場合、3つの方法を使用できます。PD5006P, PD5306P の場合、INSTB 信号と命令のどちらか1つだけを使用することができ、ボードのジャンパーで設定します。

（1） INSTB 信号によるラッチ

INSTB 信号による入力同時ラッチの手順は次の通りです。

●ボードのジャンパー設定

ボードの JP3 ジャンパーで INSTB 信号を使用する設定（出荷時はこの設定です）にしてください。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

① 外部ストローブ信号変化方向設定（Dio_SetStrobeDir）関数でINSTB信号の立ち上がりを使用するか立ち下がりを使用するかを指定します。
② Dio_SetSmlInStbCmd関数で、入力同時ラッチ有効設定（INSTB、命令）を有効にします。
③ INSTB 信号が変化したとき、入力に設定されているすべての信号（フィルタ通過後の入力の値）がラッチされます。

割り込み設定（Dio_SetIntrptMode）関数で、INSTB信号変化時の割り込みを有効にすると、信号変化時に割り込みを発生させることができます。

注意：

INSTB 信号状態によっては、Dio_SetSmlInStbCmd、Dio_SetStrobeDir 関数実行時に入力同時ラッチされる場合があります。例えば、INSTB 立ち上がり指定状態で INSTB 信号が Hi の時は、Dio_SetSmlInStbCmd 関数で入力同時ラッチ有効状態に変更した場合、関数実行時に入力同時ラッチされます。このとき、INSTB 信号変化時の割り込みを有効にしていると割り込みも発生します。

（2） 命令によるラッチ

命令による入力同時ラッチの手順は次の通りです。

●ボードのジャンパー設定

PD5006P, PD5306P の場合、ボードの JP3 ジャンパーで命令を使用する設定にしてください。PD5206P の場合、JP3 ジャンパーの設定を出荷時（INSTB 信号を使用する設定）のままにしてください。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

① Dio_SetSmlInStbCmd関数で、入力同時ラッチ有効設定（INSTB、命令）を有効にします。
② 入力同時ラッチ命令（Dio_ExecSmlInLatch）関数を実行すると、入力に設定されているすべての信号（フィルタ通過後の入力の値）がラッチされます。

注意：

PD5006P, PD5306P の JP3 ジャンパーで命令を選択したときは、入力同時ラッチ有効設定（Dio_SetSmlInStbCmd）関数で有効状態に変更した場合、関数実行時に入力同時ラッチされます。

PD5006P, PD5306P の JP3 ジャンパーで命令を選択したときは、次の処理を行わないでください。
・外部ストローブ信号変化方向設定（Dio_SetStrobeDir）関数で INSTB 信号の方向を変更する
・割り込み設定（Dio_SetIntrptMode）関数で INSTB 信号変化時の割り込みを有効にする

（3） タイマによるラッチ（PD5206P のみ）

本ボードは 1μsec～32sec の範囲で設定できるタイマを持っています。タイマを起動させ、タイムアウトしたときにすべての入力信号を同時ラッチします。タイマ動作は、1 回だけ動作する単一起動と、停止させるまで繰り返し動作する連続起動があります。連続起動の場合にはタイムアウトごとに入力値がラッチされます。
タイマによる入力同時ラッチはタイマ0（タイマ番号0）のタイマを使用します。手順は次の通りです。

① Dio_SetSmlInTimer関数で、入力同時ラッチ有効設定（タイマ）を有効にします。タイムアウトで割り込みを発生させる場合には、割り込み設定（Dio_SetIntrptMode）関数でタイマ0のタイマ割り込みを有効にします。
② タイマ値設定（Dio_SetTimer）関数でタイマ0のタイマ値を設定します。
③ タイマ起動（Dio_StartTimer）関数でタイマ0のタイマを単一起動、または連続起動します。
④ タイマのタイムアウトで入力に設定されているすべての信号（フィルタ通過後の入力値）がラッチされます。Dio_ReadCurrentTimer関数で動作タイマ値を読み出すか、割り込みの発生でタイムアウトを確認します。割り込みの場合には、割り込み要因読み出し（Dio_ReadIntrpt）関数でタイマ0の割り込みを確認します。

（4） ラッチした入力値の読み出し

同時ラッチした入力の値は、入力同時ラッチデータ読み出し（Dio_ReadLatch）関数で読み出すことができます。

注意：
入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力論理設定を変更してはいけません。
PD5306P の場合：入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力／出力の設定を変更してはいけません。

3.2.4 出力同時セット

すべての出力信号を同時に出力させる機能です。出力同時セットには、OTSTB 信号、命令、タイマの3つの方法があります。PD5206P の場合、3つの方法を使用できます。PD5106P, PD5306P の場合、OTSTB 信号と命令のどちらか1つだけを使用することができ、ボードのジャンパーで設定します。

出力同時セット有効設定を有効にすると、Dio_Out_xxByte 関数(注1)で出力値を書いても出力信号は変化しません。あらかじめ Dio_Out_xxByte 関数ですべてのポートに出力値を書いておき、出力同時セット機能(OTSTB 信号、命令、タイマのいずれか)の各タイミングで、それらの出力値が全出力信号に対して同時に出力されます。

注1: Dio_Out_xxByte の xx には 1,2,4,8,n のいずれかが入ります。また、出力同時セット有効設定を有効にしている場合、Dio_Out_1Bit と Dio_Out_nBit 関数は使用できません。

(1) OTSTB 信号による出力同時セット

OTSTB 信号による出力同時セットの手順は次の通りです。

●ボードのジャンパー設定
ボードの JP2 ジャンパーで OTSTB 信号を使用する設定(出荷時はこの設定です)にしてください。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

④ 外部ストローブ信号変化方向設定(Dio_SetStrobeDir)関数でOTSTB信号の立ち上がりを使用するか立ち下がりを使用するかを指定します。
⑤ Dio_SetSmlOutStbCmd関数で、出力同時セット有効設定(OTSTB、命令)を有効にします。
⑥ Dio_Out_xxByte 関数ですべてのポートに出力値を書きます。
⑦ OTSTB 信号を指定した方向に変化させると、あらかじめ書いた出力値が全出力信号に対して同時に出力されます。

割り込み設定(Dio_SetIntrptMode)関数で、OTSTB信号変化時の割り込みを有効にすると、信号変化時に割り込みを発生させることができます。

注意:
OTSTB 信号状態によっては、Dio_SetSmlOutStbCmd、Dio_SetStrobeDir 関数実行時に出力同時セットされる場合があります。例えば、OTSTB 立ち上がり指定状態で OTSTB 信号が Hi の時は、Dio_SetSmlOutStbCmd 関数で出力同時セット有効状態に変更した場合、関数実行時に出力同時セットされます。このとき、OTSTB 信号変化時の割り込みを有効にしていると割り込みも発生します。

(2) 命令による出力同時セット

命令による出力同時セットの手順は次の通りです。

●ボードのジャンパー設定
PD5106P, PD5306P の場合、ボードの JP2 ジャンパーで命令を使用する設定にしてください。PD5206P の場合、JP2 ジャンパーの設定を出荷時(OTSTB 信号を使用する設定)のままにしてください。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

① Dio_SetSmlOutStbCmd関数で、出力同時セット有効設定(OTSTB、命令)を有効にします。
② Dio_Out_xxByte 関数ですべてのポートに出力値を書きます。
③ 出力同時セット命令(Dio_ExecSmlOut)関数を実行すると、あらかじめ書いた出力値が全出力信号に対して同時に出力されます。

注意:
PD5106P, PD5306P の JP2 ジャンパーで命令を選択したときは、次の処理を行わないでください。
・外部ストローブ信号変化方向設定(Dio_SetStrobeDir)関数で OTSTB 信号の方向を変更する
・割り込み設定(Dio_SetIntrptMode)関数で OTSTB 信号変化時の割り込みを有効にする

（3） タイマによる出力同時セット（PD5206P のみ）

本ボードは 1μsec～32sec の範囲で設定できるタイマを持っています。タイマを起動させ、あらかじめ Dio_Out_xxByte 関数で書いた出力値を、タイムアウトしたときに同時に出力させることができます。タイマ動作は、1 回だけ動作する単一起動と、停止させるまで繰り返し動作する連続起動があります。連続起動の場合には、あらかじめ Dio_Out_xxByte 関数で書いた出力値が、タイムアウトごとに同時に出力されます。
タイマによる出力同時セットはタイマ1（タイマ番号1）のタイマを使用します。手順は次の通りです。

① Dio_SetSmlOutTimer関数で、出力同時セット有効設定（タイマ）を有効にします。タイムアウトで割り込みを発生させる場合には、割り込み設定（Dio_SetIntrptMode）関数でタイマ1のタイマ割り込みを有効にします。
② タイマ値設定（Dio_SetTimer）関数でタイマ1のタイマ値を設定します。
③ Dio_Out_xxByte 関数ですべてのポートに出力値を書きます。
④ タイマ起動（Dio_StartTimer）関数でタイマ1のタイマを単一起動、または連続起動します。
⑤ Dio_ReadCurrentTimer関数で動作タイマ値を読み出すか、割り込みの発生でタイムアウトを確認します。タイムアウト時に、あらかじめ書いた出力値が全出力信号に対して同時に出力されます。
⑥ 割り込みの場合には、割り込み要因読み出し（Dio_ReadIntrpt）関数でタイマ1の割り込みを確認します。
⑦ タイマ連続起動の場合には、Dio_Out_xxByte 関数で次の出力値を書き込み、⑤、⑥を繰り返します。

3.2.5 入力変化

入力変化とは、すべての入力信号について、信号の変化を捉える機能です。

（1） 入力変化有効設定・方向設定

入力信号 1 点ごとに、変化を捉えることを有効にするか無効にするかの設定と、変化する方向の指定を行います。
変化方向は、入力論理設定前の入力信号の状態が Low から Hi に変化したときの変化を捉えるのか、Hi から Low に変化したときの変化を捉えるのかを指定します。
有効／無効の設定はDio_SetInTransitionMode関数で、変化の方向設定はDio_SetInTransitionDir関数で行います。

（2） 変化捕捉の動作

入力変化有効設定で有効になった信号に対しては直ちに変化保持機能が働きます。入力値が指定の方向に変化した場合に、入力変化は”1”となります。入力変化情報を読み出すまでに、同じ入力信号が何度変化しても”1”のままです。

（3） 入力変化の読み出し

入力変化情報は、入力変化情報読み出し（Dio_ReadInTransition）関数で読み出します。入力変化情報は一度読み出すとクリアされます。

（4） 変化情報のクリア

入力変化情報は、入力変化情報読み出し（Dio_ReadInTransition）関数で読み出すとクリアされますが、入力変化情報クリア（Dio_ClearInTransition）関数でクリアすることもできます。

3.2.6 タイマ

本ボードは 1μsec ～ 32sec の範囲で設定できるタイマを持っています。PD5306P の場合は4個のタイマを使用できます。PD5006P，PD5106P，PD5206P の場合は2個のタイマを使用できます。タイマのタイムアウト時に次の3つの動作を行わせることができます。それぞれの節を参照してください。

- 入力同時ラッチ（PD5206P のみ） 3.2.3 節
- 出力同時セット（PD5206P のみ） 3.2.4 節
- 割り込みの発生 3.2.7 節

タイマの操作は、次の手順で行います。

① タイマ値設定（Dio_SetTimer）関数でタイマ値を設定します。
② タイムアウトで割り込みを発生させる場合には、割り込み設定（Dio_SetIntrptMode）関数で指定タイマのタイマ割り込みを有効にします。
③ タイマ起動（Dio_StartTimer）関数で指定タイマを単一起動、または連続起動します。
④ タイマ実行中の動作タイマ値はDio_ReadCurrentTimer関数でを読み出します。
⑤ 単一起動させたタイマを途中で停止させるには、タイマ停止（Dio_StopTimer）関数でタイマ即停止を実行します。また、連続起動させたタイマを停止させるには、タイマ停止（Dio_StopTimer）関数でタイマ即停止またはタイマサイクル停止を実行します。

3.2.7 割り込み

割り込みは次の4つの動作で発生させることができます。

- 入力同時ラッチ時の INSTB 信号の変化
- 出力同時セット時の OTSTB 信号の変化
- 入力変化
- タイマのタイムアウト

それぞれの割り込みは、割り込み設定（Dio_SetIntrptMode）関数で有効／無効の設定をします。割り込み発生時、入力変化情報はDio_ReadInTransition関数で読み出し、それ以外の割り込み要因はDio_ReadIntrpt 関数で読み出します。
アプリケーションで割り込みを使用する場合、手順は次の通りです。

① Dio_Open関数で割り込み使用モードを指定してオープンします。
② Dio_SetEventFunc関数、またはDio_SetEventMsg関数を実行し、割り込み通知を行う設定にします。割り込み発生時、指定した関数を呼び出して通知する場合はDio_SetEventFunc関数、ウィンドウにメッセージを送信して通知する場合はDio_SetEventMsg関数を実行します。（VBの場合はDio_SetEventFunc関数を使用できません。）
③ 割り込み設定（Dio_SetIntrptMode）関数で割り込みを有効にします。
④ 割り込みが発生すると、指定した方法でアプリケーションに通知されます。（指定した関数が呼び出されるかメッセージが送信されます）
⑤ 割り込み要因読み出し（Dio_ReadIntrpt）関数、または入力変化情報読み出し（Dio_ReadInTransition）関数で割り込み要因を確認します。
⑥ アプリケーションへの割り込み通知設定を解除する場合は、Dio_ResetEvent関数を使用します。

3.2.8 設定値の読み出し

本ボードに設定したタイマ値設定、割り込み設定、各有効／無効設定などのすべての設定値を読み出すことができます。

3.2.9 エラーコード

各関数が失敗した場合、エラーコードはDio_GetLastError関数で取得します。
また、Dio_ClearLastError関数でエラーコードをクリアできます。

3.2.10 パソコン起動時の設定内容

パソコン起動時、本ボードのモード・パラメータ設定内容は下表のようになります。

モード・パラメータ設定項目	パソコン起動時の設定内容	関連する関数
入力／出力の設定（PD5306P のみ）	すべての信号が入力	Dio_SetIoDir
入力信号の入力論理	すべての入力信号が Hi レベルを入力値1	Dio_SetInputLogic
入力フィルタ指定	すべての入力信号がフィルタ時定数1（注1）	Dio_SetInputFilter
フィルタ時定数1，2，3	時定数1は 7，時定数2は 10，時定数3は 14（注1）	Dio_SetFilterTime
タイマ値	すべてのタイマが0μs	Dio_SetTimer
入力変化有効／無効	すべての信号が無効	Dio_SetInTransitionMode
入力変化方向	すべて0（入力論理設定前の入力信号の状態が Low から Hi への変化）	Dio_SetInTransitionDir
INSTB 信号の変化方向	0（立ち上がり）	Dio_SetStrobeDir
OTSTB 信号の変化方向	0（立ち上がり）	
入力同時ラッチ有効／無効（INSTB、命令）	無効	Dio_SetSmlInStbCmd
入力同時ラッチ有効／無効（タイマ）（PD5206P のみ）	無効	Dio_SetSmlInTimer
出力同時セット有効／無効（OTSTB、命令）	無効	Dio_SetSmlOutStbCmd
出力同時セット有効／無効（タイマ）（PD5206P のみ）	無効	Dio_SetSmlOutTimer
入力同時ラッチ INSTB 信号割り込み	無効	Dio_SetIntrptMode
出力同時セット OTSTB 信号割り込み	無効	
入力変化割り込み	無効	
タイマ割り込み	すべてのタイマが無効	

上表の中で、入力フィルタ指定とフィルタ時定数については、デバイスドライバがパソコン起動時にボードに対して設定している内容です。また、入力／出力の設定と入力論理については、デバイスドライバ内部で情報の管理を行いますので、その設定内容を示しています。

注1： デバイスドライバは、パソコン起動時にフィルタ時定数と入力フィルタ指定を下表のように初期設定しています。

フィルタ時定数	設定値	信号遅延時間	
1	7	128μsec	←入力フィルタ指定は時定数1を設定しています。
2	10	1.02 msec	
3	14	16.4 msec	

注意：

本ボードにおける実際の信号遅延時間は、上表の遅延時間にフォトカプラの遅延時間（50～80μsec 程度）が加算されます。

4. プログラミング

この章では、アプリケーション開発のためのソフトウェア仕様とプログラミング方法について説明します。
アプリケーション開発は、Microsoft Visual C++ (以下 VC++)、あるいは Microsoft Visual Basic(以下 VB) 、あるいは Microsoft Visual C#(以下 C#)のいずれかを使用して行います。

4.1 動作環境

対応OS　Windows98 Windows2000 WindowsXP

対応言語

Microsoft Visual C++ 6.0	
Microsoft Visual C++.NET 2003	(Windows2000, WindowsXP)
Microsoft Visual C++ 2005	(Windows2000, WindowsXP)
Microsoft Visual C++ 2008	(Windows2000, WindowsXP)
Microsoft Visual Basic 6.0	
Microsoft Visual Basic.NET 2003	(Windows2000, WindowsXP)
Microsoft Visual Basic 2005	(Windows2000, WindowsXP)
Microsoft Visual Basic 2008	(Windows2000, WindowsXP)
Microsoft Visual C#.NET 2003	(Windows2000, WindowsXP)
Microsoft Visual C# 2005	(Windows2000, WindowsXP)
Microsoft Visual C# 2008	(Windows2000, WindowsXP)

注)アプリケーション開発を行う場合は、開発ツールのサポート状況など MSDN のサポートページを参考にソフトウェアを開発してください。サンプルプログラムを Visual Studio.NET2003、Visual Studio 2005、Visual Studio 2008 で動作させる場合は、MSDN のサポートページを参考にサンプルプログラムの移行を行ってください。

4.2 ソフトウェア構成

4.2.1 ソフトウェアー覧

項目	フォルダ	ファイル名・フォルダ名	説明
デバイスドライバ	Driver	Nv_PiDio.sys	デバイスドライバ本体
		Nv_PiDio.dll	ダイナミックリンクライブラリ　VC++, VB, C# 共通
		Nv_PiDio.inf	インストールファイル
ライブラリ	Lib¥VB6	Nv_PiDio_DLL.bas	Nv_PiDio.dll を使用するための Declare 宣言ファイル　VB6.0 専用
	Lib¥VB.NET2003_2005	Nv_PiDio_DLL.vb	Nv_PiDio.dll を使用するための Declare 宣言ファイル VB.NET2003 または VB2005 で使用します。
	Lib¥VC6	Nv_PiDio.lib	Nv_PiDio.dll を使用するためのライブラリ　VC++ 専用
		Nv_PiDio_DLL.h	Nv_PiDio.dll を使用するためのヘッダ定義ファイル　VC++ 専用
	Lib¥Csharp	Pd5000pWrap.dll	Nv_PiDio.dll を使用するためのクラスライブラリ C# 専用
サンプルプログラム（VB6.0）	Sample¥VB6	Input	入力値・出力値読み出しサンプル： 入力関数を実行し、読み出した値を表示します。
		Output	出力サンプル： 画面から入力した出力値を出力関数にて書き込みます。
		Interrupt	割り込みサンプル： タイマ割り込みを有効にし、タイマ停止時に割り込みを発生させます。
		Timer	タイマサンプル：4つのタイマの動作確認ができるサンプルです。
		InputChange	入力変化サンプル：入力変化に関する動作確認ができるサンプルです。
		InputLatch	入力同時ラッチサンプル：入力同時ラッチに関する動作確認ができるサンプルです。
		OutputLatch	出力同時セットサンプル：出力同時セットに関する動作確認ができるサンプルです。
サンプルプログラム（VB.NET2003）	Sample¥VB.NET2003	Input	入力値・出力値読み出しサンプル： 入力関数を実行し、読み出した値を表示します。
		Output	出力サンプル： 画面から入力した出力値を出力関数にて書き込みます。
		Interrupt	割り込みサンプル： タイマ割り込みを有効にし、タイマ停止時に割り込みを発生させます。
		Timer	タイマサンプル：4つのタイマの動作確認ができるサンプルです。
		InputChange	入力変化サンプル：入力変化に関する動作確認ができるサンプルです。
		InputLatch	入力同時ラッチサンプル：入力同時ラッチに関する動作確認ができるサンプルです。
		OutputLatch	出力同時セットサンプル：出力同時セットに関する動作確認ができるサンプルです。
サンプルプログラム（VC6.0）	Sample¥VC6	Input	入力値・出力値読み出しサンプル： 入力関数を実行し、読み出した値を表示します。
		Output	出力サンプル： 画面から入力した出力値を出力関数にて書き込みます。
		Interrupt	割り込みサンプル： タイマ割り込みを有効にし、タイマ停止時に割り込みを発生させます。
		Timer	タイマサンプル：4つのタイマの動作確認ができるサンプルです。
		InputChange	入力変化サンプル：入力変化に関する動作確認ができるサンプルです。
		InputLatch	入力同時ラッチサンプル：入力同時ラッチに関する動作確認ができるサンプルです。
		OutputLatch	出力同時セットサンプル：出力同時セットに関する動作確認ができるサンプルです。

サンプルプログラム（C#.NET2003）	Sample¥Csharp	input	入力値・出力値読み出しサンプル: 入力関数を実行し、読み出した値を表示します。
		output	出力サンプル:画面から入力した出力値を出力関数にて書き込みます。
		interrupt	割り込みサンプル:タイマ割り込みを有効にし、タイマ停止時に割り込みを発生させます。
		Timer	タイマサンプル:4つのタイマの動作確認ができるサンプルです。
		InputChange	入力変化サンプル:入力変化に関する動作確認ができるサンプルです。
		InputLatch	入力同時ラッチサンプル:入力同時ラッチに関する動作確認ができるサンプルです。
		OutputLatch	出力同時セットサンプル:出力同時セットに関する動作確認ができるサンプルです。
評価ツール	Tool	PD_Tool.exe	PD5000P シリーズの4種類のボードを評価するツールです。 入力値・出力値の読み出し、出力動作、モード・パラメータ設定＆読み出し、タイマ動作、割り込み処理、入力変化情報や入力ラッチデータの読み出しなどを行います。

注: サンプルアプリを VB2005 にて使用する場合

VB2005 開発ツールで VB.NET2003 のサンプルアプリを開き、変換ウィザードで変換すると、VB2005 で使用できます。

4.2.2 ファイルの詳細

ソフトウェアのフォルダ構成とファイル内容を以下に示します。

注意： CD－ROMからハードディスクにコピーする場合、ファイルやフォルダが読み取り専用になる場合がありますので、必要であれば読み取り専用を解除してから使用して下さい。

```
¥
+---Driver
|       +---Nv_PiDio.sys                    デバイスドライバ本体
|       +---Nv_PiDio.inf                    デバイスドライバのインストール用プログラム
|       +---Nv_PiDio.dll                    デバイスドライバを使用するためのダイナミックリンクライブラリ
|       +---Version.txt                     デバイスドライバのバージョン説明ファイル
|
+---LIB
|       +---VB6
|       |       +---Nv_PiDio_DLL.bas        VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |
|       +---VB.NET2003_2005
|       |       +---Nv_PiDio_DLL.vb         VB.NET2003, VB2005 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |
|       +---VC6
|       |       +---Nv_PiDio.lib            VC6.0 用 Nv_PiDio.dll のライブラリファイル
|       |       +---Nv_PiDio_DLL.h          VC6.0 用 Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|       |
|       +---Csharp
|               +---Pd5000pWrap.dll         C#.NET2003 用 Pd5000p C＃クラスライブラリファイル
+---Tool
|       +---PD_Tool.exe                     ボードの評価ツール
|       +---ReadMe.txt
|
+---Sample
        +---ReadMe.txt
        +---VB6                             VB6.0 用サンプルプログラム
        |       +---Input                   入力値・出力値読み出しサンプル
        |       |       +---Input.vbp       VB サンプルプログラム用プロジェクトファイル（VB6.0）
        |       |       +---Form1.frm       サンプルプログラム
        |       |       +---Module1.bas     サンプルプログラム
        |       |       +---Nv_PiDio_DLL.bas VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
        |       |       +---exe
        |       |               +---Input.exe   実行ファイル
        |       |
        |       +---Output                  出力サンプル
        |       |       +---Output.vbp      VB サンプルプログラム用プロジェクトファイル（VB6.0）
        |       |       +---Form1.frm       サンプルプログラム
        |       |       +---Module1.bas     サンプルプログラム
        |       |       +---Nv_PiDio_DLL.bas VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
        |       |       +---exe
        |       |               +---Output.exe  実行ファイル
        |       |
        |       +---Interrupt               割り込みサンプル
        |       |       +---Interrupt.vbp   VB サンプルプログラム用プロジェクトファイル（VB6.0）
        |       |       +---Form1.frm       サンプルプログラム
        |       |       +---Module1.bas     サンプルプログラム
        |       |       +---Nv_PiDio_DLL.bas VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
        |       |       +---exe
        |       |               +---Interrupt.exe   実行ファイル
```

```
|       |
|       +---Timer                           サンプルプログラム Timer
|       |       +--- Timer.vbp             VB サンプルプログラム用プロジェクトファイル（VB6.0）
|       |       +---Form1.frm              サンプルプログラム
|       |       +---Module1.bas            サンプルプログラム
|       |       +---Nv_PiDio_DLL.bas       VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +--- Timer.exe      実行ファイル
|       |
|       +---InputChange                     サンプルプログラム InputChange
|       |       +--- InputChange.vbp        VB サンプルプログラム用プロジェクトファイル（VB6.0）
|       |       +---Form1.frm              サンプルプログラム
|       |       +---Module1.bas            サンプルプログラム
|       |       +---Nv_PiDio_DLL.bas       VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +--- InputChange.exe 実行ファイル
|       |
|       +---InputLatch                      サンプルプログラム InputLatch
|       |       +--- InputLatch.vbp         VB サンプルプログラム用プロジェクトファイル（VB6.0）
|       |       +---Form1.frm              サンプルプログラム
|       |       +---Module1.bas            サンプルプログラム
|       |       +---Nv_PiDio_DLL.bas       VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +--- InputLatch.exe 実行ファイル
|       |
|       +---OutputLatch                     サンプルプログラム OutputLatch
|               +--- OutputLatch.vbp        VB サンプルプログラム用プロジェクトファイル（VB6.0）
|               +---Form1.frm              サンプルプログラム
|               +---Module1.bas            サンプルプログラム
|               +---Nv_PiDio_DLL.bas       VB6.0 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|               +---exe
|                       +--- OutputLatch.exe 実行ファイル
|
|
|
+---VB.NET2003                              VB.NET2003 用サンプルプログラム
|       +---ReadMe.txt
|       +---Input                           入力値・出力値読み出しサンプル
|       |       +---Input.sln               VB サンプルプログラム用ソリューションファイル（VB.NET2003）
|       |       +---Form1.vb               サンプルプログラム
|       |       +---Module1.vb             サンプルプログラム
|       |       +---Nv_PiDio_DLL.vb        VB.NET2003 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +---Input.exe       実行ファイル
|       |
|       +---Output                          出力サンプル
|       |       +---Output.sln              VB サンプルプログラム用ソリューションファイル（VB.NET2003）
|       |       +---Form1.vb               サンプルプログラム
|       |       +---Module1.vb             サンプルプログラム
|       |       +---Nv_PiDio_DLL.vb        VB.NET2003 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +---Output.exe      実行ファイル
|       |
|       +---Interrupt                       割り込みサンプル
|       |       +--- Interrupt.sln          VB サンプルプログラム用ソリューションファイル（VB.NET2003）
|       |       +---Form1.vb               サンプルプログラム
|       |       +---Nv_PiDio_DLL.vb        VB.NET2003 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +--- Interrupt.exe  実行ファイル
```

```
|       |
|       +--- Timer                          サンプルプログラム Timer
|       |       +--- Timer.sln              VB サンプルプログラム用ソリューションファイル（VB.NET2003）
|       |       +---Form1.vb                サンプルプログラム
|       |       +---Module1.vb              サンプルプログラム
|       |       +---Nv_PiDio_DLL.vb         VB.NET2003 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +--- Timer.exe      実行ファイル
|       |
|       +--- InputChange                    サンプルプログラム InputChange
|       |       +--- InputChange.sln        VB サンプルプログラム用ソリューションファイル（VB.NET2003）
|       |       +---Form1.vb                サンプルプログラム
|       |       +---Module1.vb              サンプルプログラム
|       |       +---Nv_PiDio_DLL.vb         VB.NET2003 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +--- InputChange.exe 実行ファイル
|       |
|       +--- InputLatch                     サンプルプログラム InputLatch
|       |       +--- InputLatch.sln         VB サンプルプログラム用ソリューションファイル（VB.NET2003）
|       |       +---Form1.vb                サンプルプログラム
|       |       +---Module1.vb              サンプルプログラム
|       |       +---Nv_PiDio_DLL.vb         VB.NET2003 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|       |       +---exe
|       |               +--- InputLatch.exe 実行ファイル
|       |
|       +--- OutputLatch                    サンプルプログラム OutputLatch
|               +--- OutputLatch.sln        VB サンプルプログラム用ソリューションファイル（VB.NET2003）
|               +---Form1.vb                サンプルプログラム
|               +---Module1.vb              サンプルプログラム
|               +---Nv_PiDio_DLL.vb         VB.NET2003 用 Nv_PiDio.dll Declare 宣言、各定義ファイル
|               +---exe
|                       +--- OutputLatch.exe 実行ファイル
|
|
+---VC6                                     VC6.0 用サンプルプログラム
|       +---Input                           入力値・出力値読み出しサンプル
|       |       +---Input.dsw               VC サンプルプログラム用プロジェクトワークスペース（VC6.0）
|       |       +---Input.cpp               アプリケーションクラスメンバ関数
|       |       +---InputDlg.cpp            ダイアログクラスメンバ関数
|       |       +---Input.h                 アプリケーションクラス宣言
|       |       +---InputDlg.h              ダイアログクラス宣言
|       |       +---Sample.H                サンプルプログラムヘッダファイル
|       |       +---Nv_PiDio.lib            Nv_PiDio.dll のライブラリファイル
|       |       +---Nv_PiDio_DLL.h          Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|       |       +---exe
|       |               +---Input.exe       実行ファイル
|       |
|       +---Output                          出力サンプル
|       |       +---Output.dsw              VC サンプルプログラム用プロジェクトワークスペース（VC6.0）
|       |       +---Output.cpp              アプリケーションクラスメンバ関数
|       |       +---OutputDlg.cpp           ダイアログクラスメンバ関数
|       |       +---Output.h                アプリケーションクラス宣言
|       |       +---OutputDlg.h             ダイアログクラス宣言
|       |       +---Sample.H                サンプルプログラムヘッダファイル
|       |       +---Nv_PiDio.lib            Nv_PiDio.dll のライブラリファイル
|       |       +---Nv_PiDio_DLL.h          Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|       |       +---exe
|       |               +---Output.exe      実行ファイル
|       |
```

```
|       |
|       +---Interrupt                   割り込みサンプル
|       |       +---Interrupt.dsw       VC サンプルプログラム用プロジェクトワークスペース（VC6.0）
|       |       +---Interrupt.cpp       アプリケーションクラスメンバ関数
|       |       +---InterruptDlg.cpp    ダイアログクラスメンバ関数
|       |       +---Interrupt.h         アプリケーションクラス宣言
|       |       +---InterruptDlg.h      ダイアログクラス宣言
|       |       +---Nv_PiDio.lib        Nv_PiDio.dll のライブラリファイル
|       |       +---Nv_PiDio_DLL.h      Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|       |       +---exe
|       |               +---Interrupt.exe   実行ファイル
|       |
|       +--- Timer                      サンプルプログラム Timer
|       |       +--- Timer.dsw          VC サンプルプログラム用プロジェクトワークスペース（VC6.0）
|       |       +--- Timer.cpp          アプリケーションクラスメンバ関数
|       |       +--- TimerDlg.cpp       ダイアログクラスメンバ関数
|       |       +--- Timer.h            アプリケーションクラス宣言
|       |       +--- TimerDlg.h         ダイアログクラス宣言
|       |       +---Sample.H            サンプルプログラムヘッダファイル
|       |       +---Nv_PiDio.lib        Nv_PiDio.dll のライブラリファイル
|       |       +---Nv_PiDio_DLL.h      Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|       |       +---exe
|       |               +---Input.exe   実行ファイル
|       |
|       +--- InputChange                サンプルプログラム InputChange
|       |       +--- InputChange.dsw    VC サンプルプログラム用プロジェクトワークスペース（VC6.0）
|       |       +--- InputChange.cpp    アプリケーションクラスメンバ関数
|       |       +--- InputChangeDlg.cpp ダイアログクラスメンバ関数
|       |       +--- InputChange.h      アプリケーションクラス宣言
|       |       +--- InputChangeDlg.h   ダイアログクラス宣言
|       |       +---Sample.H            サンプルプログラムヘッダファイル
|       |       +---Nv_PiDio.lib        Nv_PiDio.dll のライブラリファイル
|       |       +---Nv_PiDio_DLL.h      Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|       |       +---exe
|       |               +---Input.exe   実行ファイル
|       |
|       +--- InputLatch                 サンプルプログラム InputLatch
|       |       +---InputLatch.dsw      VC サンプルプログラム用プロジェクトワークスペース（VC6.0）
|       |       +---InputLatch.cpp      アプリケーションクラスメンバ関数
|       |       +---InputLatchDlg.cpp   ダイアログクラスメンバ関数
|       |       +---InputLatch.h        アプリケーションクラス宣言
|       |       +---InputLatchDlg.h     ダイアログクラス宣言
|       |       +---Sample.H            サンプルプログラムヘッダファイル
|       |       +---Nv_PiDio.lib        Nv_PiDio.dll のライブラリファイル
|       |       +---Nv_PiDio_DLL.h      Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|       |       +---exe
|       |               +---Input.exe   実行ファイル
|       |
|       +--- OutputLatch                サンプルプログラム OutputLatch
|               +---OutputLatch.dsw     VC サンプルプログラム用プロジェクトワークスペース（VC6.0）
|               +---OutputLatch.cpp     アプリケーションクラスメンバ関数
|               +---OutputLatchDlg.cpp  ダイアログクラスメンバ関数
|               +---OutputLatch.h       アプリケーションクラス宣言
|               +---OutputLatchDlg.h    ダイアログクラス宣言
|               +---Sample.H            サンプルプログラムヘッダファイル
|               +---Nv_PiDio.lib        Nv_PiDio.dll のライブラリファイル
|               +---Nv_PiDio_DLL.h      Nv_PiDio.dll のヘッダファイル（関数宣言、各定義）
|               +---exe
|                       +---Input.exe   実行ファイル
```

```
|
+---Csharp
      +--- Sample input                    サンプルプログラム input
      |        +---Form1.cs                サンプルプログラム input C#ソースファイル
      |        +---input.csproj            サンプルプログラム input プロジェクトワークスペース(C#のみ)
      |        +---exe
      |              +---input.exe         サンプル input 実行ファイル
      |
      +---Sample output                    サンプルプログラム output
      |        +---Form1.cs                サンプルプログラム output C#ソースファイル
      |        +--- output.csproj          サンプルプログラム output プロジェクトワークスペース(C#のみ)
      |        +---exe
      |              +--- output.exe       サンプル output 実行ファイル
      |
      +---Sample interrupt                 サンプルプログラム interrupt
      |        +---Form1.cs                サンプルプログラム interrupt C#ソースファイル
      |        +--- interrupt.csproj       サンプルプログラム interrupt プロジェクトワークスペース(C#のみ)
      |        +---exe
      |              +--- interrupt.exe    サンプル interrupt 実行ファイル
      |
      +---Sample Timer                     サンプルプログラム Timer
      |        +---Form1.cs                サンプルプログラム Timer C#ソースファイル
      |        +--- Timer.csproj           サンプルプログラム Timer プロジェクトワークスペース(C#のみ)
      |        +---exe
      |              +--- Timer.exe        サンプル Timer 実行ファイル
      |
      +---Sample InputChange               サンプルプログラム InputChange
      |        +---Form1.cs                サンプルプログラム InputChange C#ソースファイル
      |        +--- InputChange.csproj     サンプルプログラム InputChange プロジェクトワークスペース
      |        |                           (C#のみ)
      |        +---exe
      |              +-InputChange.exe     サンプル InputChange 実行ファイル
      |
      +---Sample InputLatch                サンプルプログラム InputLatch
      |        +---Form1.cs                サンプルプログラム InputLatch C#ソースファイル
      |        +--- InputLatch.csproj      サンプルプログラム InputLatch プロジェクトワークスペース
      |        |                           (C#のみ)
      |        +---exe
      |              +---InputLatch.exe    サンプル InputLatch 実行ファイル
      |
      +---Sample OutputLatch               サンプルプログラム OutputLatch
               +---Form1.cs                サンプルプログラム OutputLatch C#ソースファイル
               +--- OutputLatch.csproj     サンプルプログラム OutputLatch プロジェクトワークスペース
               |                           (C#のみ)
               +---exe
                     +-OutputLatch.exe     サンプル OutputLatch 実行ファイル
```

※ VC++ 、VB 、C# 開発ツールが自動的に生成するファイルに関しては説明を省略します。

4.3 開発手順

4.3.1 VC++でアプリケーションを開発する場合（VC++6.0, VC++.NET2003, VC++ 2005, VC++ 2008）

アプリケーションは Nv_PiDio.lib と Nv_PiDio_DLL.h ファイルを使用します。この2ファイルは VC++ 6.0 以降対応です。

① ¥Lib¥VC6 フォルダに入っている2つのファイル Nv_PiDio.lib と Nv_PiDio_DLL.h を開発するアプリケーションのフォルダにコピーしてください。

② 開発するアプリケーションのプロジェクトに Nv_PiDio_DLL.h ファイルを追加登録してください。
また、API 関数を使用するソースファイルに Nv_PiDio_DLL.h を include して下さい。

③ VC++6.0 の場合は、「プロジェクト」－「設定」で「リンク」タブを選択し「オブジェクト／ライブラリモジュール」に Nv_PiDio.lib を追加して下さい。（「図 4.3-1 VC++6.0 プロジェクトの設定」を参照。）

VC++.NET2003、VC++ 2005、VC++ 2008 の場合は、「プロジェクト」－「プロパティ」画面で「リンカ」－「入力」を選択し、「追加の依存ファイル」に Nv_PiDio.lib を追加して下さい。
（「図 4.3-2 VC++.NET2003 プロジェクトのプロパティ」、「図 4.3-3 VC++ 2005 プロジェクトのプロパティ」「図 4.3-4 VC++ 2008 プロジェクトのプロパティ」を参照。）

④ 4.4 APIの関数を使用してプログラミングを行って下さい。

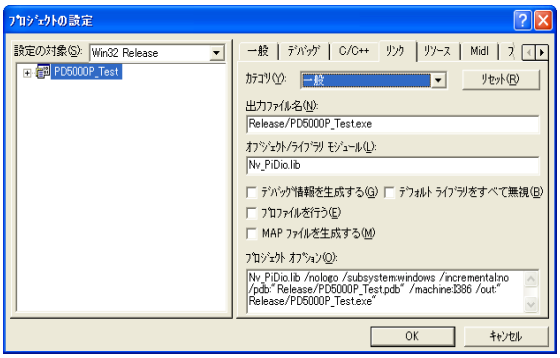


図 4.3-1 VC++6.0 プロジェクトの設定

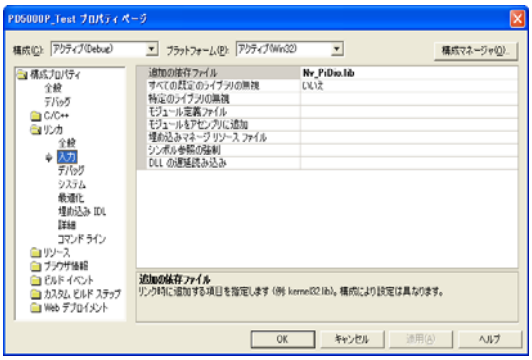


図 4.3-2 VC++.NET2003 プロジェクトのプロパティ

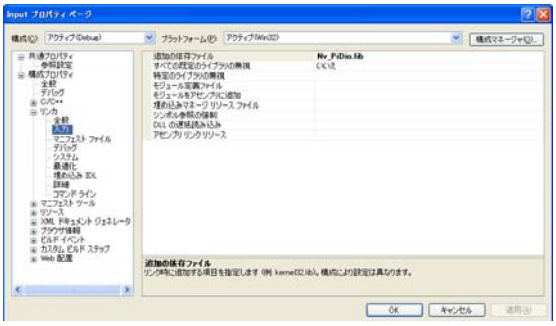


図 4.3-3 VC++ 2005 プロジェクトのプロパティ

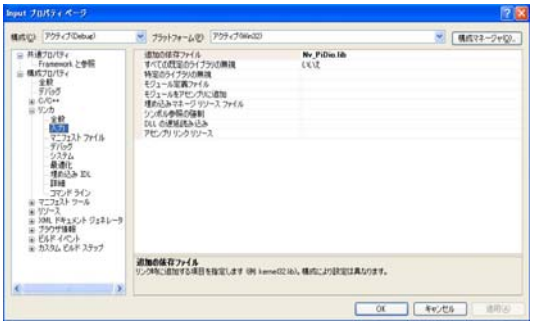


図 4.3-4 VC++ 2008 プロジェクトのプロパティ

4.3.2 VB6.0 でアプリケーションを開発する場合

① ¥Lib¥VB6 フォルダに入っている Nv_PiDio_DLL.bas ファイルを開発するアプリケーションのプロジェクトに標準モジュールとして追加してください。

② 4.4 APIの関数を使用してプログラミングを行って下さい。

まれにデバッグ中など Nv_PiDio.dll にリンクできないときは Nv_PiDio.dll をカレントフォルダにコピーして使用してください。

4.3.3 VB.NET2003, VB2005, VB2008 でアプリケーションを開発する場合

① ¥Lib¥VB.NET2003_2005 フォルダに入っている Nv_PiDio_DLL.vb ファイルを開発するアプリケーションのプロジェクトに追加してください。

② 4.4 APIの関数を使用してプログラミングを行って下さい。

まれにデバッグ中など Nv_PiDio.dll にリンクできないときは Nv_PiDio.dll をカレントフォルダにコピーして使用してください。

4.3.4 C#でアプリケーションを開発する場合

PD5000P アプリケーションでは. NETに対応した C#クラスライブラリ Pd5000pWrap.dll を使用します。

① ¥LIB¥Csharp フォルダに入っているファイル Pd5000pWrap.dll を開発するアプリケーションのフォルダにコピーしてください。

② C#.NET2003 の場合は[プロジェクト]－[参照の追加]で、Pd5000pWrap.dll への参照を追加してください。（「図4.3-5 C#.NET2003 参照の追加」を参照。）

C# 2005、C# 2008 の場合は、[プロジェクト]－[参照の追加]で「参照」タブを選択し Pd5000pWrap.dll への参照を追加して下さい。（「図 4.3-6 C# 2005、C# 2008 参照の追加」を参照。）

③ アプリケーションのソースファイルに using でネームスペース Pd5000pWrap を追加してください。

④ 4.4 API（ PD5000P ドライバ関数 ）の関数を使用してプログラミングを行って下さい。

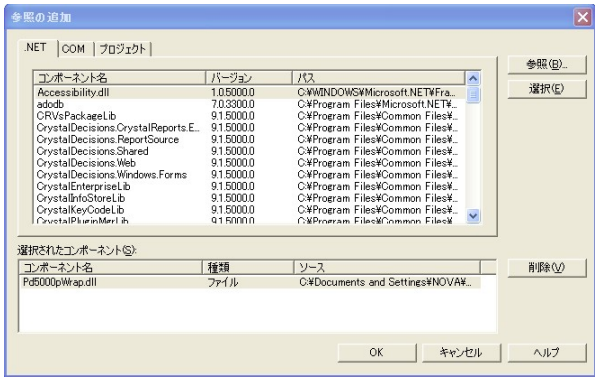


図4.3-5 C#.NET2003 参照の追加

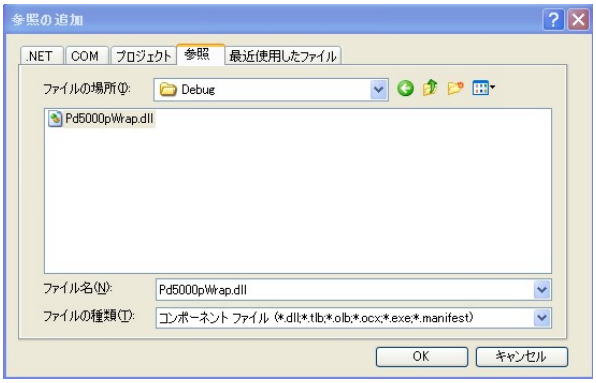


図 4.3-6 C# 2005、C# 2008 参照の追加

4.4 ＡＰＩ

この章では、Nv_PiDio.sys、Nv_PiDio.dll がアプリケーションに提供するAPIについて説明します。
API関数宣言は下記のファイルで行っています。

VC++	:	Nv_PiDio_DLL.h
VB6.0	:	Nv_PiDio_DLL.bas
VB. NET2003、VB2005、VB2008	:	Nv_PiDio_DLL.vb
C#.NET	:	Pd5000pWrap.dll

4.4.1 関数一覧

下表は、API 関数の一覧表です。
「PD5006P」「PD5106P」「PD5206P」「PD5306P」の欄は、各ボードにおいて各関数の使用可能状況を記載しています。

○ :使用できます。
× :目的の機能は実行されません。使用するとエラーが返ります。
△ :目的の機能は実行されません。使用してもエラーにはなりません。

（1） 基本関数

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_Open	ボードオープン	○	○	○	○
Dio_Close	ボードクローズ	○	○	○	○
Dio_CloseAll	すべてのボードクローズ	○	○	○	○
Dio_SetEventFunc	割り込み通知設定（関数呼び出し通知）**※VB は使用不可**	○	○	○	○
Dio_SetEventMsg	割り込み通知設定（メッセージ送信通知）	○	○	○	○
Dio_ResetEvent	割り込み通知設定の解除	○	○	○	○

（2） モード・パラメータ設定

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_SetIoDir	入力／出力の設定（1点ごとに設定可能）	×	×	×	○
Dio_SetInputLogic	入力信号の論理レベル設定（1点ごとに設定可能）	○	△	○	○
Dio_SetInputFilter	入力信号のフィルタ指定（4点ごとに設定可能）	○	△	○	○
Dio_SetFilterTime	フィルタ時定数（3種類）の設定	○	△	○	○
Dio_SetTimer	タイマ値設定	○	○	○	○
Dio_SetInTransitionMode	入力変化有効設定（1点ごとに設定可能）	○	△	○	○
Dio_SetInTransitionDir	入力変化方向設定（1点ごとに設定可能）	○	△	○	○
Dio_SetStrobeDir	外部ストローブ信号（INSTB, OTSTB）の変化方向設定	○	○	○	○
Dio_SetSmlInStbCmd	入力同時ラッチ有効設定（INSTB、命令）	○	△	○	○
Dio_SetSmlInTimer	入力同時ラッチ有効設定（タイマ）	×	×	○	×
Dio_SetSmlOutStbCmd	出力同時セット有効設定（OTSTB、命令）	△	○	○	○
Dio_SetSmlOutTimer	出力同時セット有効設定（タイマ）	×	×	○	×
Dio_SetIntrptMode	割り込み設定	○	○	○	○

（3） モード・パラメータ設定読み出し

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_GetIoDir	入力／出力設定の読み出し	×	×	×	○
Dio_GetInputLogic	入力信号の論理レベル設定の読み出し	○	○	○	○
Dio_GetInputFilter	入力信号のフィルタ設定の読み出し	○	○	○	○
Dio_GetFilterTime	フィルタ時定数設定の読み出し	○	○	○	○
Dio_GetTimer	タイマ値設定の読み出し	○	○	○	○
Dio_GetInTransitionMode	入力変化有効設定の読み出し	○	○	○	○
Dio_GetInTransitionDir	入力変化方向設定の読み出し	○	○	○	○
Dio_GetStrobeDir	外部ストローブ信号(INSTB, OTSTB)の変化方向設定の読み出し	○	○	○	○
Dio_GetSmlInStbCmd	入力同時ラッチ有効設定(INSTB、命令)の読み出し	○	○	○	○
Dio_GetSmlInTimer	入力同時ラッチ有効設定(タイマ)の読み出し	×	×	○	×
Dio_GetSmlOutStbCmd	出力同時セット有効設定(OTSTB、命令)の読み出し	○	○	○	○
Dio_GetSmlOutTimer	出力同時セット有効設定(タイマ)の読み出し	×	×	○	×
Dio_GetIntrptMode	割り込み設定の読み出し	○	○	○	○

（4） 入力値・出力値の読み出し

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_In_1Bit	1ビット読み出し(1点)	○	○	○	○
Dio_In_1Byte	1バイト読み出し(8点)	○	○	○	○
Dio_In_2Byte	2バイト読み出し(16点)	○	○	○	○
Dio_In_4Byte	4バイト読み出し(32点)	○	○	○	○
Dio_In_8Byte	8バイト読み出し(64点)	○	○	○	○
Dio_In_nBit	任意複数ビット読み出し	○	○	○	○
Dio_In_nByte	任意複数バイト読み出し	○	○	○	○

（5） 出力

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_Out_1Bit	1ビット出力(1点)	×	○	○	○
Dio_Out_1Byte	1バイト出力(8点)	△	○	○	○
Dio_Out_2Byte	2バイト出力(16点)	△	○	○	○
Dio_Out_4Byte	4バイト出力(32点)	△	○	○	○
Dio_Out_8Byte	8バイト出力(64点)	△	○	○	○
Dio_Out_nBit	任意複数ビット出力	×	○	○	○
Dio_Out_nByte	任意複数バイト出力	△	○	○	○

（6） データ読み出し

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_ReadIntrpt	割り込み要因読み出し	○	○	○	○
Dio_ReadInTransition	入力変化情報読み出し	○	○	○	○
Dio_ReadLatch	入力同時ラッチデータ読み出し	○	○	○	○
Dio_ReadCurrentTimer	タイマ実行中の動作タイマ値読み出し	○	○	○	○

（7） その他の命令

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_StartTimer	タイマ起動（単一、連続）	○	○	○	○
Dio_StopTimer	タイマ停止（即停止、サイクル停止）	○	○	○	○
Dio_ClearInTransition	入力変化情報クリア	○	△	○	○
Dio_ExecSmlInLatch	入力同時ラッチ命令の実行	○	△	○	○
Dio_ExecSmlOut	出力同時セット命令の実行	△	○	○	○

（8） エラーコード

関数名	説明	PD 5006P	PD 5106P	PD 5206P	PD 5306P
Dio_GetLastError	エラーコード取得	○	○	○	○
Dio_ClearLastError	エラーコードクリア	○	○	○	○

4.4.2 関数仕様

VC++ の場合 ・・・・・・・・・・・・・・・・・・・・・・・・・ **[VC]** の項目を参照して下さい。
VB6.0 の場合 ・・・・・・・・・・・・・・・・・・・・・・・・・ **[VB]** の項目を参照して下さい。
VB.NET2003、VB2005、VB2008 の場合 ・・・・ **[VB.NET]** の項目が有る場合は**[VB.NET]**を、無い場合は **[VB]** の項目を参照して下さい。
C#.NET の場合 ・・・・・・・・・・・・・・・・・・・・・・・・ **[C#.NET]** と **[C#]** の項目を参照して下さい。

指定のない項目は、各言語共通の内容です。

Dio_Open　ボードオープン

■機能

ボードの使用を開始します。

■書式

[VC] BOOL Dio_Open(USHORT DeviceID, USHORT SwitchNo, BOOL IntrptFlg, HANDLE* phBoard);

[VB] Function Dio_Open(ByVal DeviceID As Integer, ByVal SwitchNo As Integer, ByVal IntrptFlg As Long, ByRef phBoard As Long) As Long

[VB.NET] Function Dio_Open(ByVal DeviceID As Short, ByVal SwitchNo As Short, ByVal IntrptFlg As Integer, ByRef phBoard As Integer) As Integer

[C#.NET] bool PD5000P.Dio_Open(DEV DeviceID, ushort SwitchNo, bool IntrptFlg, out int phBoard);

■パラメータ

DeviceID ボードのデバイス ID を指定します。
PD5006P は ID_PD5006P を、PD5106P は ID_PD5106P を、PD5206P は ID_PD5206P を、
PD5306P は ID_PD5306P を指定します。
[C#]
PD5006P は DEV.ID_PD5006P を、PD5106P は DEV.ID_PD5106P を、
PD5206P は DEV.ID_PD5206P を、PD5306P は DEV.ID_PD5306P を指定します。

SwitchNo ボード上のロータリスイッチの値(0～F(16進数))を指定します。
出荷時、スイッチの値は0です。

IntrptFlg 割り込みを使用するかどうかを指定します。

[VC] TRUE ：使用する。 FALSE ：使用しない。
[VB] True ：使用する。 False ：使用しない。
[C#] True ：使用する。 False ：使用しない。

phBoard ボードハンドル変数のポインタを指定します。この変数にボードハンドルが格納されます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

各関数(Dio_GetLastError、Dio_ClearLastError は除く)を使用する前に本関数を一度実行してください。

割り込みを使用する場合は、まず本関数で割り込みを使用する設定にし、Dio_SetEventFuncまたはDio_SetEventMsg関数で割り込み通知設定を行い、Dio_SetIntrptMode関数で発生させたい割り込みを有効にします。

1つのボードに対して複数のアプリケーションから同時にオープンすることができますが、そのボードに対して割り込みを使用したオープンができるのは1つのアプリケーションのみです。

■使用例

スイッチ番号0の PD5306P ボードを開始します。(割り込みを使用しない)

[VC]	HANDLE hBoard; Ret = Dio_Open(ID_PD5306P, 0, FALSE, &hBoard);
[VB]	Dim hBoard As Long Ret = Dio_Open(ID_PD5306P, 0, False, hBoard)
[VB.NET]	Dim hBoard As Integer Ret = Dio_Open(ID_PD5306P, 0, False, hBoard)
[C#.NET]	int hBoard Ret = PD5000P.Dio_Open(DEV.ID_PD5306P, 0, false, out hBoard);

Dio_Close ボードクローズ

■機能

ボードの使用を終了します。

■書式

[VC] BOOL Dio_Close(HANDLE hBoard);
[VB] Function Dio_Close(ByVal hBoard As Long) As Long
[VB.NET] Function Dio_Close(ByVal hBoard As Integer) As Integer
[C#.NET] bool PD5000P.Dio_Close(int hBoard);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

[VC]	HANDLE hBoard; Ret = Dio_Open(ID_PD5306P, 0, FALSE, &hBoard); Ret = Dio_Close(hBoard);
[VB]	Dim hBoard As Long Ret = Dio_Open(ID_PD5306P, 0, False, hBoard) Ret = Dio_Close(hBoard)
[VB.NET]	Dim hBoard As Integer Ret = Dio_Open(ID_PD5306P, 0, False, hBoard) Ret = Dio_Close(hBoard)
[C#.NET]	int hBoard Ret = PD5000P.Dio_Close(hBoard);

Dio_CloseAll　全ボードクローズ

■機能

アプリケーション内で既に開始しているすべてのボードの使用を終了します。

■書式

[VC]　BOOL Dio_CloseAll();
[VB]　Function Dio_CloseAll() As Long
[VB.NET]　Function Dio_CloseAll() As Integer
[C#.NET]　bool PD5000P. Dio_CloseAll();

■パラメータ

パラメータはありません。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

[VC]　Ret = Dio_CloseAll();
[VB]　Ret = Dio_CloseAll()
[C#.NET]　Ret = PD5000P.Dio_CloseAll()

Dio_SetEventFunc　割り込み通知設定（関数呼び出し通知）

■機能

割り込み発生時にユーザー関数を呼び出す設定にします。
この関数を実行すると、割り込みが発生した時にユーザー関数が呼び出され、指定したパラメータが1つ渡されます。このユーザー関数は別スレッドから起動されます。
割込み通知設定を解除する場合はDio_ResetEvent関数を実行して下さい。

■書式

[VC] BOOL Dio_SetEventFunc(HANDLE hBoard, void (CALLBACK* UserFunc)(LPVOID lpParam), LPVOID lpParam);

[VB] 使用できません。

[VB.NET] 使用できません。

[C#.NET] bool PD5000P.Dio_SetEventFunc(int hBoard, UserThread Callback, int param);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

[VC]
UserFunc 割り込み発生時に呼び出すユーザー関数のアドレスを指定します。
割り込みが発生すると、ここで指定したユーザー関数が呼び出されます。

[VC]
lpParam 割り込み発生時にユーザー関数に渡す1つのパラメータ(ポインタ、または数値等)を指定します。
ポインタの場合、ユーザー関数呼び出し時に使用可能なポインタを設定して下さい。

[C#]
Callback ユーザーメソッド（デリゲート型）

[C#]
param 割り込み発生時にユーザー関数に渡す1つのパラメータ(数値等 int 限定)を指定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

本関数は、割り込みが発生していない状態で実行して下さい。
本関数を実行する場合、Dio_Open関数で割り込みを指定したオープンを行ってください。
割り込み発生要因は、Dio_ReadIntrpt関数とDio_ReadInTransition関数で確認できます。
呼び出されたユーザー関数で、必要に応じて割り込み要因を確認してください。

■使用例

[VC]

```
{
   Dio_Open(ID_PD5306P, 0, TRUE, &hBoard);                  // 割り込みを使用する設定でオープンする
   Dio_SetEventFunc(hBoard, IntrptFunc, lpParameter); // 割り込み通知設定(ユーザー関数使用)
   Dio_SetIntrptMode(hBoard, 0x10);                         // タイマ0割り込み設定
}

●割り込みユーザー関数例
void CALLBACK IntrptFunc(LPVOID Param)
{
   ・・・・・・・・・
   ULONG  IntrptData , InTransiData1, InTransiData2 ;

   Dio_ReadIntrpt(hBoard, &IntrptData);                                    // 割り込み要因読み出し
   Dio_ReadInTransition(hBoard, &InTransiData1, &InTransiData2); // 入力変化情報読み出し
   ・・・・・・・・・
}
```

[C#]

```
{
  // 割り込みを使用する設定でオープンする
  bool Ret = PD5000P.Dio_Open(DEV.ID_PD5306P, 0, true, out hBoard);
  // 割込みユーザーメソッド IntrptFunc をデリゲート型変数に代入
  PD5000P.callback[0] = new PD5000P.UserThread(IntrptFunc);
  // 割り込み通知設定(ユーザー関数使用)
  bool Ret = PD5000P.Dio_SetEventFunc(hBoard, PD5000P.callback[0], param)
  // タイマ0割り込み設定
  bool Ret = PD5000P.Dio_SetIntrptMode(hBoard, 0x00000010)
}

●割り込みユーザー関数例
private void IntrptFunc(int lParam)
{
   ・・・・・・・・・
   uint    IntrptData=0;
   uint    InTransiData1;
   uint    InTransiData2;

   // 割り込み要因読み出し
   PD5000P.Dio_ReadIntrpt(hBoard, out IntrptData);
   // 入力変化情報読み出し
   PD5000P.Dio_ReadInTransition(hBoard, out InTransiData1, out InTransiData2) ;
   ・・・・・・・・・
}
```

Dio_SetEventMsg　割り込み通知設定（メッセージ送信通知）

■機能

割り込み発生時にアプリケーションのウィンドウにメッセージを送信する設定にします。
この関数を実行すると、割り込みが発生した時に割り込み通知メッセージ（WM_INTRPT）がアプリケーションウィンドウに送信され、ボードハンドルと指定した1つのパラメータが渡されます。（wParam にボードハンドルがセットされます。）
割込み通知設定を解除する場合はDio_ResetEvent関数を実行して下さい。

■書式

[VC] BOOL Dio_SetEventMsg(HANDLE hBoard, HWND hWnd, LPARAM lParam);

[VB] Function Dio_SetEventMsg(ByVal hBoard As Long, ByVal hWnd As Long, ByVal lParam As Long) As Long

[VB.NET] Function Dio_SetEventMsg(ByVal hBoard As Integer, ByVal hWnd As Integer, ByVal lParam As Integer) As Integer

[C#.NET] bool PD5000P.Dio_SetEventMsg(int hBoard, int hWnd, int lParam);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

hWnd 割り込み発生時にメッセージを受け取るウィンドウのハンドルを指定します。
割り込みが発生すると、ここで指定したウィンドウに割り込み通知メッセージ（WM_INTRPT(※)）が送信されます。
※WM_INTRPTはユーザー定義メッセージです。定義ファイルで定義しています。定義ファイルについては、「4.4.3 補足説明（5）定義内容」を参照してください。

lParam 割り込み発生時にウィンドウに渡す1つのパラメータを指定します。
ここで指定した値は、ウィンドウメッセージ（WM_INTRPT）の lParam パラメータにセットされます。
wParam パラメータにはボードハンドルがセットされます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

本関数は、割り込みが発生していない状態で実行して下さい。
本関数を実行する場合、Dio_Open関数で割り込みを指定したオープンを行ってください。
割り込み発生要因は、Dio_ReadIntrpt関数とDio_ReadInTransition関数で確認できます。
割り込み通知メッセージが送信されたとき、必要に応じて割り込み要因を確認してください。

■使用例

[VC]

```
{
   Dio_Open(ID_PD5306P, 0, TRUE, &hBoard);              // 割り込みを使用する設定でオープンする
   Dio_SetEventMsg(hBoard, (HWND)m_hWnd, lParam);// 割込み通知設定(メッセージ使用)
   Dio_SetIntrptMode(hBoard, 0x10);                     // タイマ0割り込み設定
}

// WM_INTRPT メッセージ受信関数設定
BEGIN_MESSAGE_MAP(CTestDlg, CDialog)
   ON_MESSAGE( WM_INTRPT, OnMsgIntrpt )
END_MESSAGE_MAP()

// WM_INTRPT メッセージ受信関数
afx_msg LRESULT CTestDlg::OnMsgIntrpt(HANDLE hBoard, LPARAM lParam)
{
   ULONG  IntrptData , InTransiData1, InTransiData2 ;

   Dio_ReadIntrpt(hBoard, &IntrptData);                                  // 割り込み要因読み出し
   Dio_ReadInTransition(hBoard, &InTransiData1, &InTransiData2);  // 入力変化情報読み出し
   ・・・・・・・・・
   return 0;
}
```

[VB]

```
   ・・・・・・・・・
   Ret = Dio_Open(ID_PD5306P, 0, True, hBoard)     ' 割り込みを使用する設定でオープンする
   Ret = Dio_SetEventMsg(hBoard, hwnd, lParam)   ' 割込み通知設定(メッセージ使用)
   Ret = Dio_SetIntrptMode(hBoard, &H10)           ' タイマ0割り込み設定
End Sub

' ウィンドウメッセージ受信関数
Function WindowProc(ByVal hw As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal
lParam As Long) As Long

   Dim hBoard As Long
   Dim IntrptData As Long
   Dim InTransiData1 As Long
   Dim InTransiData2 As Long

   If uMsg = WM_INTRPT Then      ' 割込みメッセージ

      hBoard = wParam
      Ret = Dio_ReadIntrpt(hBoard, IntrptData)                                  ' 割り込み要因読み出し
      Ret = Dio_ReadInTransition(hBoard, InTransiData1, InTransiData2) ' 入力変化情報読み出し
      ・・・・・・・・・

   End If

   WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)

End Function
```

[VB.NET]

```
   ・・・・・・・・・
   Ret = Dio_Open(ID_PD5306P, 0, True, hBoard)          ’割り込みを使用する設定でオープンする
   Ret = Dio_SetEventMsg(hBoard, Handle.ToInt32, lParam)’割込み通知設定（メッセージ使用）
   Ret = Dio_SetIntrptMode(hBoard, &H10)                ’タイマ0割り込み設定
End Sub

’ウィンドウメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)

   Dim hBoard As Integer
   Dim IntrptData As Integer
   Dim InTransiData1 As Integer
   Dim InTransiData2 As Integer

   If m.Msg = WM_INTRPT Then      ’割込みメッセージ

      hBoard = m.WParam.ToInt32()
      Ret = Dio_ReadIntrpt(hBoard, IntrptData)                                  ’割り込み要因読み出し
      Ret = Dio_ReadInTransition(hBoard, InTransiData1, InTransiData2)’入力変化情報読み出し
      ・・・・・・・・・

   End If

   MyBase.WndProc(m)

End Sub
```

[C#]

```
{
  // 割り込みを使用する設定でオープンする
  bool Ret = PD5000P.Dio_Open(DEV.ID_PD5306P, 0, true, out hBoard);
  // 割込み通知設定（メッセージ使用）
  int hWnd = this.Handle.ToInt32();
  int lParam = 0;
  bool Ret = PD5000P.Dio_SetEventMsg(hBoard, hWnd, lParam);
  // タイマ0割り込み設定
  bool Ret = PD5000P.Dio_SetIntrptMode(hBoard, 0x10)
}

// WM_INTRPT メッセージ受信関数
protected override void WndProc(ref Message m)
 {
   base.WndProc(ref m);
   if (m.Msg == (int)MSG_ID.WM_INTRPT)
   {
   uint   IntrptData=0;
   uint   InTransiData1;
   uint   InTransiData2;

   // 入力変化以外の割り込みについては、本関数で割り込み要因を確認してください。
   if(PD5000P.Dio_ReadIntrpt(hBoard, out IntrptData) == false)
      ・・・・・・・・・;    // エラー処理
   // 入力変化の割り込みについては、本関数で割り込み要因を確認してください。
   if(PD5000P.Dio_ReadInTransition(hBoard, out InTransiData1, out InTransiData2) == false)
      ・・・・・・・・・;    // エラー処理
   //割り込み要因と入力変化情報を表示する
   StrBuf = "割り込み発生！¥n¥n 割り込み要因  = " + IntrptData.ToString("X")+
            "h¥n¥n 入力変化情報  = "+InTransiData1.ToString("X")+
            " h, "+InTransiData2.ToString("X")+ " h";
   MessageBox.Show(this,StrBuf,"PD5000P 割込みサンプル", MessageBoxButtons.OK,
   MessageBoxIcon.Exclamation);
   ・・・・・・・・・
   }
 }
```

Dio_ResetEvent　割り込み通知設定の解除

■機能

割り込み発生時にアプリケーションに対して割り込み通知を行わない設定にします。
この関数を実行すると、割り込みが発生しても「ユーザー関数の呼び出し」と「割り込み通知メッセージの送信」を両方とも行いません。この関数は、Dio_SetEventFunc関数とDio_SetEventMsg関数で設定した内容をすべて無効にします。

■書式

[VC]　BOOL Dio_ResetEvent(HANDLE hBoard);
[VB]　Function Dio_ResetEvent(ByVal hBoard As Long) As Long
[VB.NET]　Function Dio_ResetEvent(ByVal hBoard As Integer) As Integer
[C#.NET]　bool PD5000P.Dio_ResetEvent(int hBoard);

■パラメータ

hBoard　Dio_Open 関数で取得したボードハンドルを指定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

本関数は、割り込みが発生していない状態で実行して下さい。

■使用例

[VC]　Ret = Dio_ResetEvent(hBoard);
[VB]　Ret = Dio_ResetEvent(hBoard)
[C#]　Ret = PD5000P.Dio_ResetEvent(hBoard)

Dio_SetIoDir　入力／出力の設定

■対応ボード　:　PD5306P

■機能

すべての入／出力信号（全64点）について、入力信号として使用するか、出力信号として使用するかを1点ごとに設定します。
パソコン起動時は、すべての信号が入力の設定です。

■書式

[VC]　BOOL Dio_SetIoDir(HANDLE hBoard, ULONG WtData1, ULONG WtData2);

[VB]　Function Dio_SetIoDir(ByVal hBoard As Long, ByVal WtData1 As Long, ByVal WtData2 As Long) As Long

[VB.NET]　Function Dio_SetIoDir(ByVal hBoard As Integer, ByVal WtData1 As Integer, ByVal WtData2 As Integer) As Integer

[C#.NET]　bool PD5000P.Dio_SetIoDir(int hBoard, uint WtData1, uint WtData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

WtData1 ポート0～3の信号32点の入力／出力を1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

0 :入力 （入力信号として使用する）
1 :出力 （出力信号として使用する）

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

例えば、SN00, SN11, SN22, SN33 信号を出力に、それ以外を入力にする場合、08040201h（16 進数）を設定します。

WtData2 ポート4～7の信号32点の入力／出力を1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

0 :入力 （入力信号として使用する）
1 :出力 （出力信号として使用する）

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

例えば、SN44, SN55, SN66, SN77 信号を出力に、それ以外を入力にする場合、80402010h（16 進数）を設定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

出力信号を入力信号に変更する場合、必ず変更前に該当する信号の出力値を0の状態にしてください。

■使用例

SN00, SN11, SN66, SN77 信号を出力信号にします。

[VC] Ret = Dio_ SetIoDir(hBoard, 0x0201, 0x80400000);
[VB] Ret = Dio_ SetIoDir(hBoard, &H0201, &H80400000)
[C#] Ret = PD5000P.Dio_ SetIoDir(hBoard, 0x0201, 0x80400000)

Dio_SetInputLogic　入力論理設定

■機能

すべての入力信号について、論理レベルを1点ごとに設定します。
パソコン起動時の入力論理は、すべての入力信号が Hi レベルを入力値1とする設定です。

■書式

[VC]　BOOL Dio_SetInputLogic(HANDLE hBoard, ULONG WtData1, ULONG WtData2);

[VB]　Function Dio_SetInputLogic(ByVal hBoard As Long, ByVal WtData1 As Long, ByVal WtData2 As Long) As Long

[VB.NET]　Function Dio_SetInputLogic(ByVal hBoard As Integer, ByVal WtData1 As Integer, ByVal WtData2 As Integer) As Integer

[C#.NET]　bool PD5000P.Dio_SetInputLogic(int hBoard, uint WtData1, uint WtData2);

■パラメータ

hBoard　Dio_Open 関数で取得したボードハンドルを指定します。

WtData1　ポート0～3の入力信号32点の論理レベルを1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

0 :入力信号 Hi レベルを入力値1
1 :入力信号 Low レベルを入力値1

出力信号（または出力に設定している信号）には0を設定してください。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

例えば、SN00, SN11, SN22, SN33 信号を1に、それ以外を0にする場合、08040201h（16 進数）を設定します。

WtData2 ポート4～7の入力信号32点の論理レベルを1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

0 :入力信号 Hi レベルを入力値1
1 :入力信号 Low レベルを入力値1

出力信号(または出力に設定している信号)には0を設定してください。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

例えば、SN44, SN55, SN66, SN77 信号を1に、それ以外を0にする場合、80402010h(16 進数)を設定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

出力信号に対する論理設定の機能はありませんので、本関数では、出力信号(または出力に設定している信号)に対しては常に0がセットされます。(出力信号に対して 0 以外を指定した場合でも、0がセットされます)

■注意点

PD5306P の場合:
この関数を実行する時点で入力信号に設定されている信号に対してのみ、入力論理を設定できます。
この関数を実行する時点で出力に設定されている信号に入力論理1を設定しても、1は設定されず0が設定されます。
また、入力信号に入力論理1を設定しても、その後その信号を出力に変更した場合、設定した入力論理は出力信号に変更した時点で0クリアされます。アプリケーション実行中に入力／出力の設定を頻繁に変更する場合は注意してください。

■使用例

SN00, SN11, SN66, SN77 信号に対して入力論理1(入力信号 Low レベルを入力値1)を、それ以外の信号に対して入力論理0(入力信号 Hi レベルを入力値1)を設定します。

[VC] Ret = Dio_SetInputLogic(hBoard, 0x0201, 0x80400000);
[VB] Ret = Dio_SetInputLogic(hBoard, &H0201, &H80400000)
[C#] Ret = PD5000P. Dio_SetInputLogic(hBoard, 0x0201, 0x80400000);

Dio_SetInputFilter　入力フィルタ指定

■機能

すべての入力信号に対して、信号4点ごとに積分フィルタの時定数を指定します。
フィルタは、3種類のフィルタ時定数（1，2，3）とフィルタなしの中から選択でき、4点（各ポートの上位4点／下位4点）ごとに設定します。（フィルタ時定数は、Dio_SetFilterTime関数で、16個の信号遅延時間から3個を選択しフィルタ時定数1，2，3に設定します。）
パソコン起動時は、すべての入力信号にフィルタ時定数1が設定されます。

■書式

[VC] BOOL Dio_SetInputFilter(HANDLE hBoard, ULONG WtData);

[VB] Function Dio_SetInputFilter(ByVal hBoard As Long, ByVal WtData As Long) As Long

[VB.NET] Function Dio_SetInputFilter(ByVal hBoard As Integer, ByVal WtData As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetInputFilter(int hBoard, uint WtData);

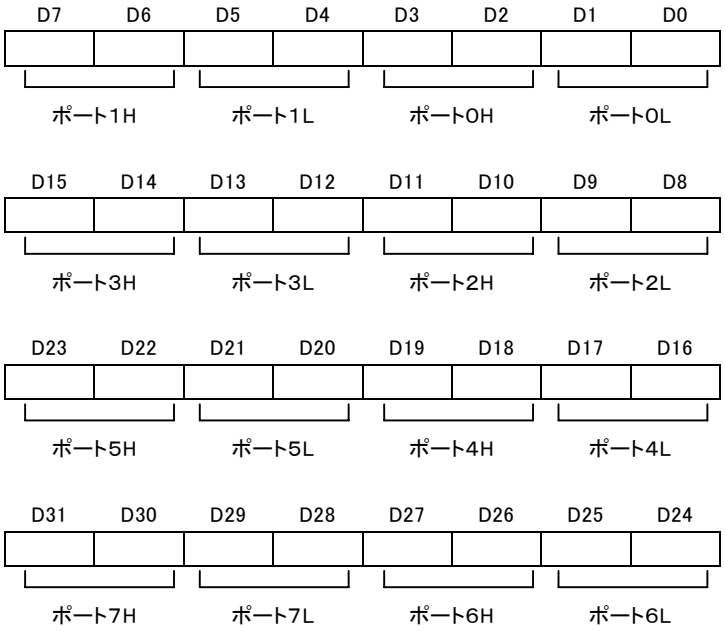
■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

WtData 全ポートの入力信号4点ごとのフィルタ時定数を指定します。
4点ごとに、下表の3種類のフィルタ時定数(1, 2, 3)とスルーの中から1つを指定します。

指定値 (上位ビット、下位ビット)	フィルタ指定内容
0, 0	スルー(フィルタなし)
0, 1	時定数1のフィルタが設定される
1, 0	時定数2のフィルタが設定される
1, 1	時定数3のフィルタが設定される

この変数の D0～31 ビットに、各ポートの上位4点／下位4点に指定するフィルタ時定数を、それぞれ2ビット単位(表の指定値)で設定します。下図のポート**H はポート**の上位4点、ポート**L はポート**の下位4点を表します。ポート番号(H／L)の所に、その4点に対する値を設定してください。



例えば、ポート0～3が時定数1、ポート4～7が時定数2の場合は、AAAA5555h(16 進数)を設定します。

注: 出力信号に対しては、フィルタは機能しません。

PD5206P の場合:
出力ポート(ポート4～7)は 00 を設定してください。
出力ポートに 00 以外を設定しても 00 が設定されます。

PD5306P の場合:
設定した入力フィルタは、入力信号に対してのみ機能します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

出力信号に対しては、フィルタは機能しません。

■使用例

ポート0～3にフィルタ時定数1を、ポート4～7にフィルタ時定数2を設定します。

[VC] Ret = Dio_SetInputFilter(hBoard, 0xAAAA5555);
[VB] Ret = Dio_SetInputFilter(hBoard, &HAAAA5555)
[C#] Ret = PD5000P. Dio_SetInputFilter(hBoard, 0xAAAA5555);

Dio_SetFilterTime　フィルタ時定数の設定

■機能

フィルタ時定数1, 2, 3の値を設定します。
本ボードは3種類のフィルタ時定数(1, 2, 3)を持っています。各々のフィルタ時定数は、信号遅延時間を表す16個の数値の中から1つを選択することができます。
パソコン起動時、時定数1は7, 時定数2は10, 時定数3は14が設定されます。

■書式

[VC] BOOL Dio_SetFilterTime(HANDLE hBoard, ULONG Filter1, ULONG Filter2, ULONG Filter3);

[VB] Function Dio_SetFilterTime(ByVal hBoard As Long, ByVal Filter1 As Long, ByVal Filter2 As Long, ByVal Filter3 As Long) As Long

[VB.NET] Function Dio_SetFilterTime(ByVal hBoard As Integer, ByVal Filter1 As Integer, ByVal Filter2 As Integer, ByVal Filter3 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetFilterTime(int hBoard, FILTER Filter1, FILTER Filter2, FILTER Filter3);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Filter1 フィルタ時定数1に設定する値を指定します。
下表の16個の設定値(0～15)の中から1つを選択します。

Filter2 フィルタ時定数2に設定する値を指定します。
下表の16個の設定値(0～15)の中から1つを選択します。

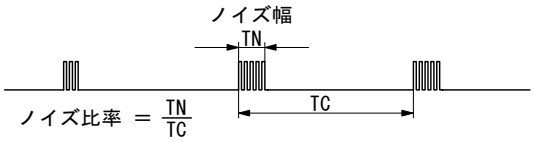
Filter3 フィルタ時定数3に設定する値を指定します。
下表の16個の設定値(0～15)の中から1つを選択します。

設定値	[C#] 設定値	信号遅延時間 (μsec)	除去ノイズ幅 (μsec)	設定値	[C#] 設定値	信号遅延時間 (msec)	除去ノイズ幅 (msec)
0	FILTER. F00_00001	1.00	0.875	8	FILTER. F08_00256	0.256	0.224
1	FILTER. F01_00002	2.00	1.75	9	FILTER. F09_00512	0.512	0.448
2	FILTER. F02_00004	4.00	3.50	10	FILTER. F10_01020	1.02	0.896
3	FILTER. F03_00008	8.00	7.00	11	FILTER. F11_02050	2.05	1.79
4	FILTER. F04_00016	16.0	14.0	12	FILTER. F12_04100	4.10	3.58
5	FILTER. F05_00032	32.0	28.0	13	FILTER. F13_08190	8.19	7.17
6	FILTER. F06_00064	64.0	56.0	14	FILTER. F14_16400	16.4	14.3
7	FILTER. F07_00128	128	112	15	FILTER. F15_32800	32.8	28.7

注意:

本ボードにおける実際の信号遅延時間は、上表の遅延時間にフォトカプラの遅延時間（50～80μsec 程度）が加算されます。

除去ノイズ幅とは、本フィルタが除去可能なノイズの最大時間幅をいいます。設定値を上げると除去可能な最大ノイズ幅は上がりますが、信号の遅延時間が大きくなりますので、設定値は適切な値を設定します。



■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

本関数で設定したフィルタ時定数を各ポートの上位／下位に設定する方法についてはDio_SetInputFilter関数を参照してください。

■使用例

フィルタ時定数1に7を、フィルタ時定数2に10を、フィルタ時定数3に14を設定します。

```
[VC]    Ret = Dio_SetFilterTime(hBoard, 7, 10, 14);
[VB]    Ret = Dio_SetFilterTime(hBoard, 7, 10, 14)
[C#]    Ret = PD5000P. Dio_SetFilterTime(hBoard, FILTER.F07_00128, FILTER.F10_01020,
        FILTER.F14_16400);
```

Dio_SetTimer　タイマ値設定

■機能

タイマの値を 0～32,767 の範囲で、μsec 単位、または msec 単位で設定します。
PD5306P は4個のタイマを、それ以外のボードは2個のタイマを使用でき、それぞれのタイマにタイマ値を設定します。
パソコン起動時、すべてのタイマ値は0μsec です。

■書式

[VC] BOOL Dio_SetTimer(HANDLE hBoard, ULONG TimerNo, ULONG TimerValue, ULONG Unit);

[VB] Function Dio_SetTimer(ByVal hBoard As Long, ByVal TimerNo As Long, ByVal TimerValue As Long, ByVal Unit As Long) As Long

[VB.NET] Function Dio_SetTimer(ByVal hBoard As Integer, ByVal TimerNo As Integer, ByVal TimerValue As Integer, ByVal Unit As Integer) As Integer

[C#.NET] bool Dio_SetTimer(int hBoard, TIMER TimerNo, uint TimerValue, TSCALE Unit);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

TimerNo タイマ値を設定するタイマ番号を指定します。

PD5306P の場合：
タイマ番号として0～3のいずれかを指定します。(4個のタイマを使用できます)

PD5006P, PD5106P, PD5206P の場合：
タイマ番号として0、または1を指定します。(2個のタイマを使用できます)

TimerValue 設定するタイマ値を 0～32,767 の範囲で指定します。
ここで1を設定すると、1μsec、または1msec となります。単位は Unit 変数で指定します。

Unit 設定するタイマ値の単位を指定します。

[VC] [VB] 0 ：μsec 単位、 1 ：msec 単位
[C#] TSCALE . MICRO ：μsec 単位、 TSCALE . MILLI ：msec 単位

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

タイマの起動・停止は Dio_StartTimer 、Dio_StopTimer 関数で行い、動作中タイマのタイマ値読み出しは Dio_ReadCurrentTimer関数で行います。
タイマは、割り込み、入力同時ラッチ(PD5206P のみ)、出力同時セット(PD5206P のみ)などで使用します。

■使用例

タイマ番号0のタイマ値として 1000msec を設定します。

```
[ VC ]  Ret = Dio_SetTimer(hBoard, 0, 1000, 1);
[ VB ]  Ret = Dio_SetTimer(hBoard, 0, 1000, 1)
[ C# ]  Ret = PD5000P. Dio_SetTimer(hBoard, 0, 1000, TSCALE . MILLI);
```

Dio_SetInTransitionMode　入力変化有効設定

■ 機能

入力変化とは、入力信号の変化を捉える機能です。
この関数は、すべての入力信号について、入力変化有効／無効の設定を1点ごとに行います。
入力変化有効(変化を捉える)に設定した入力信号について、設定した方向に入力信号が変化した場合、変化を捉えます。方向は、入力変化方向設定(Dio_SetInTransitionDir)関数で設定します。
パソコン起動時は、すべての入力信号について、入力変化無効設定(入力変化を捉えない)です。

■書式

[VC] BOOL Dio_SetInTransitionMode(HANDLE hBoard, ULONG WtData1, ULONG WtData2);

[VB] Function Dio_SetInTransitionMode(ByVal hBoard As Long, ByVal WtData1 As Long, ByVal WtData2 As Long) As Long

[VB.NET] Function Dio_SetInTransitionMode(ByVal hBoard As Integer, ByVal WtData1 As Integer, ByVal WtData2 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetInTransitionMode(int hBoard, uint WtData1, uint WtData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

WtData1 ポート0～3の入力信号32点の入力変化有効／無効を1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

0 :入力変化無効(入力変化を捉えない)
1 :入力変化有効(入力変化を捉える)

注意: 出力信号(または出力に設定している信号)には必ず0を設定してください。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

例えば、SN00, SN11, SN22, SN33 信号を1に、それ以外を0にする場合、08040201h(16 進数)を設定します。

WtData2 ポート4～7の入力信号32点の入力変化有効／無効を1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

0 :入力変化無効(入力変化を捉えない)
1 :入力変化有効(入力変化を捉える)

注意: 出力信号(または出力に設定している信号)には必ず0を設定してください。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

例えば、SN44, SN55, SN66, SN77 信号を1に、それ以外を0にする場合、80402010h(16 進数)を設定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

各入力信号について、入力信号LowからHiの変化を捉えるか、入力信号HiからLowの変化を捉えるかは、入力変化方向設定(Dio_SetInTransitionDir)関数で指定します。
入力変化情報読み出しはDio_ReadInTransition関数、入力変化情報クリアはDio_ClearInTransition関数を使用します。

■使用例

SN00, SN11, SN66, SN77 信号に対して1(入力変化有効)を、それ以外の信号に対して0(入力変化無効)を設定します。

```
[ VC ]   Ret = Dio_SetInTransitionMode(hBoard, 0x0201, 0x80400000);
[ VB ]   Ret = Dio_SetInTransitionMode(hBoard, &H0201, &H80400000)
[ C# ]   Ret = PD5000P. Dio_SetInTransitionMode(hBoard, 0x0201, 0x80400000);
```

Dio_SetInTransitionDir　入力変化方向設定

■機能

入力変化とは、入力信号の変化を捉える機能です。
この関数は、すべての入力信号について、入力変化方向（入力信号 Low から Hi の変化を捉える、または入力信号 Hi から Low の変化を捉える）の設定を1点ごとに行います。
入力変化有効設定（Dio_SetInTransitionMode）関数で入力変化有効にした入力信号について、本関数で設定した方向に入力信号が変化した場合、変化を捉えます。
パソコン起動時、すべての入力信号について、入力変化方向は「入力信号 Low から Hi の変化を捉える」です。

■書式

[VC] BOOL Dio_SetInTransitionDir(HANDLE hBoard, ULONG WtData1, ULONG WtData2);

[VB] Function Dio_SetInTransitionDir(ByVal hBoard As Long, ByVal WtData1 As Long, ByVal WtData2 As Long) As Long

[VB.NET] Function Dio_SetInTransitionDir(ByVal hBoard As Integer, ByVal WtData1 As Integer, ByVal WtData2 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetInTransitionDir(int hBoard, uint WtData1, uint WtData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

WtData1 ポート0～3の入力信号32点の入力変化方向を1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

入力変化方向は、入力論理設定前の入力信号の状態について設定します。
0 ：入力論理設定前の入力信号の状態が Low から Hi に変化したときの変化を捉える
1 ：入力論理設定前の入力信号の状態が Hi から Low に変化したときの変化を捉える

出力信号（または出力に設定している信号）には0を設定してください。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00
	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10
	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20
	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

0：Low から Hi の変化
1：Hi から Low の変化

例えば、SN00, SN11, SN22, SN33 信号を1に、それ以外を0にする場合、08040201h（16 進数）を設定します。

WtData2 ポート4～7の入力信号32点の入力変化方向を1点ごとに設定します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値を設定してください。

入力変化方向は、入力論理設定前の入力信号の状態について設定します。
0 ：入力論理設定前の入力信号の状態が Low から Hi に変化したときの変化を捉える
1 ：入力論理設定前の入力信号の状態が Hi から Low に変化したときの変化を捉える

出力信号（または出力に設定している信号）には0を設定してください。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40
	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50
	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60
	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

0：Low から Hi の変化
1：Hi から Low の変化

例えば、SN44, SN55, SN66, SN77 信号を1に、それ以外を0にする場合、80402010h（16 進数）を設定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

入力変化有効／無効の設定については、入力変化有効設定（Dio_SetInTransitionMode）関数で行います。
入力変化情報読み出しはDio_ReadInTransition関数、入力変化情報クリアはDio_ClearInTransition関数を使用します。

■注意点

入力変化有効設定後に、本関数で入力変化方向を変更した場合、変更内容によっては入力変化が1になる場合があります。
本関数実行後に、入力変化情報クリア（Dio_ClearInTransition）関数を実行し、入力変化情報をクリアしてください。

■使用例

SN00, SN11, SN66, SN77 信号に対して1（Hi から Low の変化を捉える）を、それ以外の信号に対して0（Low から Hi の変化を捉える）を設定します。

```
[ VC ]   Ret = Dio_SetInTransitionDir(hBoard, 0x0201, 0x80400000);
[ VB ]   Ret = Dio_SetInTransitionDir(hBoard, &H0201, &H80400000)
[ C# ]   Ret = PD5000P. Dio_SetInTransitionDir(hBoard, 0x0201, 0x80400000);
```

Dio_SetStrobeDir　外部ストローブ信号変化方向設定

■機能

外部ストローブ信号(INSTB, OTSTB)の変化方向として、信号の立ち上がりを使用するか立ち下がりを使用するかを、それぞれの信号に対して設定します。
INSTB 信号は入力同時ラッチと割り込みで、OTSTB 信号は出力同時セットと割り込みで使用します。
パソコン起動時、外部ストローブ信号(INSTB, OTSTB)の変化方向は、両方とも「立ち上がりを使用する」です。

■書式

[VC] BOOL Dio_SetStrobeDir(HANDLE hBoard, ULONG Instb, ULONG Otstb);

[VB] Function Dio_SetStrobeDir(ByVal hBoard As Long, ByVal Instb As Long, ByVal Otstb As Long) As Long

[VB.NET] Function Dio_SetStrobeDir(ByVal hBoard As Integer, ByVal Instb As Integer, ByVal Otstb As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetStrobeDir(int hBoard, EDGE_TRG Instb, EDGE_TRG Otstb);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Instb INSTB 信号の変化方向を指定します。

[VC] [VB] 0 :立ち上がり、 1 :立ち下がり
[C#] EDGE_TRG.EDGE_P :立ち上がり、 EDGE_TRG. EDGE_M :立ち下がり

Otstb OTSTB 信号の変化方向を指定します。

[VC] [VB] 0 :立ち上がり、 1 :立ち下がり
[C#] EDGE_TRG.EDGE_P :立ち上がり、 EDGE_TRG. EDGE_M :立ち下がり

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

INSTB信号による入力同時ラッチ有効設定はDio_SetSmlInStbCmd関数で、OTSTB信号による出力同時セット有効設定はDio_SetSmlOutStbCmd関数で指定します。INSTB,OTSTB信号変化時の割り込みはDio_SetIntrptMode関数で指定します。

■注意点

INSTB 信号状態によっては、本関数実行時に入力同時ラッチされる場合があります。
例えば、入力同時ラッチ(INSTB、命令)有効状態で INSTB 信号が Hi のときは、本関数で INSTB を立ち上がりに変更した場合、関数実行時に入力同時ラッチされます。このとき、INSTB 信号変化時の割り込みを設定していると割り込みも発生します。
OTSTB 信号状態によっては、本関数実行時に出力同時セットされる場合があります。
例えば、出力同時セット(OTSTB、命令)有効状態で OTSTB 信号が Hi のときは、本関数で OTSTB を立ち上がりに変更した場合、関数実行時に出力同時セットされます。このとき、OTSTB 信号変化時の割り込みを設定していると割り込みも発生します。

PD5006P, PD5106P, PD5306P の場合:
ボードの JP3 ジャンパーで命令を選択している場合、本関数で INSTB 信号の方向を変更しないでください。
ボードの JP2 ジャンパーで命令を選択している場合、本関数で OTSTB 信号の方向を変更しないでください。

■使用例

INSTB 信号の変化方向に0(立ち上がり)を、OTSTB 信号の変化方向に1(立ち下がり)を設定します。

```
[ VC ]    Ret = Dio_SetStrobeDir(hBoard, 0, 1);
[ VB ]    Ret = Dio_SetStrobeDir(hBoard, 0, 1)
[ C# ]    Ret = PD5000P. Dio_SetStrobeDir(hBoard, EDGE_TRG.EDGE_P, EDGE_TRG. EDGE_M);
```

Dio_SetSmlInStbCmd　入力同時ラッチ有効設定(INSTB、命令)

■機能

INSTB 信号、または命令による入力同時ラッチ機能の有効／無効を設定します。
この機能を有効にすると、INSTB信号の変化時(立ち上がりまたは立ち下がりを設定)、または入力同時ラッチ命令(Dio_ExecSmlInLatch)実行時に、全入力が内部に取り込まれます。
パソコン起動時、入力同時ラッチ有効／無効設定(INSTB、命令)は無効設定になります。

■書式

[VC] BOOL Dio_SetSmlInStbCmd(HANDLE hBoard, ULONG Enabled);

[VB] Function Dio_SetSmlInStbCmd(ByVal hBoard As Long, ByVal Enabled As Long) As Long

[VB.NET] Function Dio_SetSmlInStbCmd(ByVal hBoard As Integer, ByVal Enabled As Integer) As Integer

[C#] bool PD5000P. Dio_SetSmlInStbCmd(int hBoard, ENABLED Enabled);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Enabled INSTB 信号、または命令による入力同時ラッチ機能の有効／無効を指定します。

[VC] [VB]　0 :無効、 1 :有効
[C#]　ENABLED.FALSE :無効、 ENABLED.TRUE :有効

有効にすると、INSTB 信号変化時、または入力同時ラッチ命令実行時に、入力同時ラッチされます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

INSTB信号の立ち上がりを使用するか立ち下がりを使用するかは、外部ストローブ信号変化方向設定(Dio_SetStrobeDir)関数で指定し、INSTB信号変化時の割り込みはDio_SetIntrptMode関数で指定します。入力同時ラッチされた値は、入力同時ラッチデータ読み出し(Dio_ReadLatch)関数で読み出すことができます。

PD5006P, PD5306P の場合:
INSTB 信号、または命令のどちらの機能を使用するかは、ボードの JP3 ジャンパーで設定し、設定した機能による入力同時ラッチのみを行うことができます。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

PD5206P の場合:
ボードの JP3 ジャンパーで INSTB 信号を使用する設定(出荷時はこの設定です)にしてください。この設定にすると、INSTB 信号と命令の両方の機能を使用できます。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

■注意点

入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力論理設定を変更してはいけません。

INSTB 信号状態によっては、本関数実行時に入力同時ラッチされる場合があります。
例えば、INSTB 立ち上がり指定状態で INSTB 信号が Hi のときは、本関数で入力同時ラッチ有効状態に変更した場合、関数実行時に入力同時ラッチされます。このとき、INSTB 信号変化時の割り込みを有効にしていると割り込みも発生します。

PD5006P, PD5306P の場合:
JP3 ジャンパーで命令を選択しているときは、本関数で入力同時ラッチ有効状態に変更した場合、関数実行時に入力同時ラッチされます。
JP3 ジャンパーで命令を選択している場合は、次の処理を行わないでください。
・外部ストローブ信号変化方向設定(Dio_SetStrobeDir)関数で INSTB 信号の方向を変更する
・割り込み設定(Dio_SetIntrptMode)関数で INSTB 信号変化時の割り込みを有効にする

PD5306P の場合:
入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力／出力の設定を変更してはいけません。

■使用例

INSTB 信号、または命令による入力同時ラッチ機能を有効にします。

```
[ VC ]    Ret = Dio_SetSmlInStbCmd(hBoard, 1);
[ VB ]    Ret = Dio_SetSmlInStbCmd(hBoard, 1)
[ C# ]    Ret = PD5000P. Dio_SetSmlInStbCmd(hBoard, ENABLED.TRUE);
```

Dio_SetSmlInTimer　入力同時ラッチ有効設定(タイマ)

■対応ボード　:　PD5206P

■機能

タイマによる入力同時ラッチ機能の有効／無効を設定します。タイマはタイマ0(タイマ番号0のタイマ)を使用します。
この機能を有効にすると、タイマ0を起動させ、タイマカウンタが設定したタイマ値になったときに、全入力が内部に取り込まれます。
パソコン起動時、入力同時ラッチ有効／無効設定(タイマ)は無効設定になります。

■書式

[VC]　BOOL Dio_SetSmlInTimer(HANDLE hBoard, ULONG Enabled);

[VB]　Function Dio_SetSmlInTimer(ByVal hBoard As Long, ByVal Enabled As Long) As Long

[VB.NET]　Function Dio_SetSmlInTimer(ByVal hBoard As Integer, ByVal Enabled As Integer) As Integer

[C#.NET]　bool PD5000P. Dio_SetSmlInTimer(int hBoard, ENABLED Enabled);

■パラメータ

hBoard　Dio_Open 関数で取得したボードハンドルを指定します。

Enabled　タイマ0(タイマ番号0のタイマ)による入力同時ラッチ機能の有効／無効を指定します。

[VC] [VB]　0 :無効、 1 :有効
[C#]　ENABLED.FALSE :無効、 ENABLED.TRUE :有効

有効にすると、タイマ0を起動させ、タイマカウンタが設定したタイマ値になったときに、入力同時ラッチされます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

タイマ0のタイマ値はタイマ値設定(Dio_SetTimer)関数で設定し、タイマ0の起動・停止はDio_StartTimer関数、Dio_StopTimer関数で行います。入力同時ラッチされた値は、入力同時ラッチデータ読み出し(Dio_ReadLatch)関数で読み出すことができます。タイマ割り込みはDio_SetIntrptMode関数で設定します。

■注意点

入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力論理設定を変更してはいけません。

PD5306P の場合:
入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力／出力の設定を変更してはいけません。

■使用例

タイマ0(タイマ番号0のタイマ)による入力同時ラッチ機能を有効にします。

[VC]　Ret = Dio_SetSmlInTimer(hBoard, 1);
[VB]　Ret = Dio_SetSmlInTimer(hBoard, 1)
[C#]　Ret = PD5000P. Dio_SetSmlInTimer(hBoard, ENABLED.TRUE);

Dio_SetSmlOutStbCmd　出力同時セット有効設定(OTSTB、命令)

■機能

OTSTB 信号、または命令による出力同時セット機能の有効／無効を設定します。
この機能を有効にすると、Dio_Out_xxByte関数(注1)で出力値を書いても、出力信号は変化しません。OTSTB信号の変化時(立ち上がりまたは立ち下がりを設定)、または出力同時セット命令(Dio_ExecSmlOut)実行時に、あらかじめDio_Out_xxByte関数で書き込んだ値が出力信号に反映されます。
パソコン起動時、出力同時セット有効／無効設定(OTSTB、命令)は無効設定になります。
注1： Dio_Out_xxByte の xx には 1,2,4,8,n のいずれかが入ります。

■書式

[VC] BOOL Dio_SetSmlOutStbCmd(HANDLE hBoard, ULONG Enabled);

[VB] Function Dio_SetSmlOutStbCmd(ByVal hBoard As Long, ByVal Enabled As Long) As Long

[VB.NET] Function Dio_SetSmlOutStbCmd(ByVal hBoard As Integer, ByVal Enabled As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetSmlOutStbCmd(int hBoard, ENABLED Enabled);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Enabled OTSTB 信号、または命令による出力同時セット機能の有効／無効を指定します。

[VC] [VB]　0 :無効、 1 :有効
[C#]　ENABLED.FALSE :無効、 ENABLED.TRUE :有効

有効にすると、あらかじめ Dio_Out_xxByte 関数で書いた値が、OTSTB 信号変化時、または出力同時セット命令実行時に出力信号に反映されます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

OTSTB信号の立ち上がりを使用するか立ち下がりを使用するかは、外部ストローブ信号変化方向設定(Dio_SetStrobeDir)関数で指定し、OTSTB信号変化時の割り込みはDio_SetIntrptMode関数で指定します。
出力同時セット有効設定(OTSTB、命令)を有効にしている場合、Dio_Out_1BitとDio_Out_nBit関数は使用できません。

PD5106P, PD5306P の場合：
OTSTB 信号、または命令のどちらの機能を使用するかは、ボードの JP2 ジャンパーで設定し、設定した機能による出力同時セットのみを行うことができます。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

PD5206P の場合：
ボードの JP2 ジャンパーで OTSTB 信号を使用する設定(出荷時はこの設定です)にしてください。この設定にすると、OTSTB 信号と命令の両方の機能を使用できます。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。

■注意点

OTSTB 信号状態によっては、本関数実行時に出力同時セットされる場合があります。
例えば、OTSTB 立ち上がり指定状態で OTSTB 信号が Hi のときは、本関数で出力同時セット有効状態に変更した場合、関数実行時に出力同時セットされます。このとき、OTSTB 信号変化時の割り込みを有効にしていると割り込みも発生します。

PD5106P, PD5306P の場合：
JP2 ジャンパーで命令を選択している場合は、次の処理を行わないでください。
・外部ストローブ信号変化方向設定（Dio_SetStrobeDir）関数で OTSTB 信号の方向を変更する
・割り込み設定（Dio_SetIntrptMode）関数で OTSTB 信号変化時の割り込みを有効にする

■使用例

OTSTB 信号、または命令による出力同時セット機能を有効にします。

```
[ VC ]    Ret = Dio_SetSmlOutStbCmd(hBoard, 1);
[ VB ]    Ret = Dio_SetSmlOutStbCmd(hBoard, 1)
[ C# ]    Ret = PD5000P. Dio_SetSmlOutStbCmd(hBoard, ENABLED.TRUE);
```

Dio_SetSmlOutTimer　出力同時セット有効設定(タイマ)

■対応ボード　:　PD5206P

■機能

タイマによる出力同時セット機能の有効／無効を設定します。タイマはタイマ1(タイマ番号1のタイマ)を使用します。
この機能を有効にすると、Dio_Out_xxByte 関数(注1)で出力値を書いても、出力信号は変化しません。タイマ1を起動させ、タイマカウンタが設定したタイマ値になったときに、あらかじめ Dio_Out_xxByte 関数で書き込んだ値が出力信号に反映されます。パソコン起動時、出力同時セット有効／無効設定(タイマ)は無効設定になります。
注1： Dio_Out_xxByte の xx には 1,2,4,8,n のいずれかが入ります。

■書式

[VC] BOOL Dio_SetSmlOutTimer(HANDLE hBoard, ULONG Enabled);

[VB] Function Dio_SetSmlOutTimer(ByVal hBoard As Long, ByVal Enabled As Long) As Long

[VB.NET] Function Dio_SetSmlOutTimer(ByVal hBoard As Integer, ByVal Enabled As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetSmlOutTimer(int hBoard, ENABLED Enabled);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Enabled タイマ1(タイマ番号1のタイマ)による出力同時セット機能の有効／無効を指定します。

[VC] [VB]　0 :無効、 1 :有効
[C#]　ENABLED.FALSE :無効、 ENABLED.TRUE :有効

有効にすると、あらかじめ Dio_Out_xxByte 関数で書いた値が、タイマ1を起動させ、タイマカウンタが設定したタイマ値になったときに、出力信号に反映されます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

タイマ1のタイマ値はタイマ値設定(Dio_SetTimer)関数で設定し、タイマ1の起動・停止は Dio_StartTimer 関数、Dio_StopTimer関数で行います。タイマ割り込みは Dio_SetIntrptMode関数で設定します。
出力同時セット有効設定(タイマ)を有効にしている場合、Dio_Out_1Bit と Dio_Out_nBit関数は使用できません。

■使用例

タイマ1(タイマ番号1のタイマ)による出力同時セット機能を有効にします。

```
[ VC ]　Ret = Dio_SetSmlOutTimer(hBoard, 1);
[ VB ]　Ret = Dio_SetSmlOutTimer(hBoard, 1)
[ C# ]　Ret = PD5000P. Dio_SetSmlOutTimer(hBoard, ENABLED.TRUE);
```

Dio_SetIntrptMode　割り込み設定

■機能

割り込みの有効／無効を設定します。
設定できる割り込みは、INSTB信号変化時の割り込み、OTSTB信号変化時の割り込み、入力変化時の割り込み、タイマ割り込み(タイマごと)です。この機能を有効にすると、指定した条件のとき、ボード内で割り込みが発生します。発生した割り込みをアプリケーションに通知する場合は、Dio_SetEventFunc関数、またはDio_SetEventMsg関数を使用します。
パソコン起動時、すべての割り込み設定は無効設定になります。

■書式

[VC] BOOL Dio_SetIntrptMode(HANDLE hBoard, ULONG IntrptMode);

[VB] Function Dio_SetIntrptMode(ByVal hBoard As Long, ByVal IntrptMode As Long) As Long

[VB.NET] Function Dio_SetIntrptMode(ByVal hBoard As Integer, ByVal IntrptMode As Integer) As Integer

[C#.NET] bool PD5000P. Dio_SetIntrptMode(int hBoard, uint IntrptMode);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

IntrptMode 各割り込みの有効／無効を設定します。
有効にすると、指定した条件のとき、割り込みが発生します。
この変数の D1～7 ビットに、1ビットごと有効／無効を設定します。その他のビットには0を設定してください。

0 :無効、 1 :有効

D7	D6	D5	D4	D3	D2	D1	D0
TIM3	TIM2	TIM1	TIM0	TRN	OTS	INS	0

割り込み設定

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

D23	D22	D21	D20	D19	D18	D17	D16
0	0	0	0	0	0	0	0

D31	D30	D29	D28	D27	D26	D25	D24
0	0	0	0	0	0	0	0

● 各ビットの説明

D1　INS　　INSTB 信号変化時の割り込み有効
このビットに1をセットすると、INSTB信号変化時の割り込みが有効になります。INSTB信号の変化方向(立ち上がり／立ち下がり)は、Dio_SetStrobeDir関数で指定します。INSTB信号が変化すると、割り込みが発生し、割り込み要因(Dio_ReadIntrpt関数参照)がセットされます。

注意: 入力同時ラッチ有効設定(INSTB、命令)を有効にしないと、この割り込みは有効になりません。

D2 OTS OTSTB 信号変化時の割り込み有効
このビットに1をセットすると、OTSTB信号変化時の割り込みが有効になります。OTSTB信号の変化方向（立ち上がり／立ち下がり）は、Dio_SetStrobeDir関数で指定します。OTSTB信号が変化すると、割り込みが発生し、割り込み要因（Dio_ReadIntrpt関数参照）がセットされます。

注意： 出力同時セット有効設定（OTSTB、命令）を有効にしないと、この割り込みは有効になりません。

D3 TRN 入力変化割り込み有効
このビットに1をセットすると、入力変化有効設定で有効にした入力信号の内、いずれかの信号が変化したときに割り込みが発生します。入力変化有効にした信号が変化（変化方向は入力変化方向設定で設定）すると、割り込みが発生し、入力変化情報（Dio_ReadInTransition関数参照）がセットされます。

D4 TIM0 タイマ割り込み有効（タイマ0）

このビットに1をセットすると、タイマ番号0のタイマ割り込みが有効になります。タイマ0を起動させ、タイマがタイムアウトすると割り込みが発生し、割り込み要因（Dio_ReadIntrpt関数参照）がセットされます。
タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。

D5 TIM1 タイマ割り込み有効（タイマ1）

このビットに1をセットすると、タイマ番号1のタイマ割り込みが有効になります。タイマ1を起動させ、タイマがタイムアウトすると割り込みが発生し、割り込み要因（Dio_ReadIntrpt関数参照）がセットされます。
タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。

D6 TIM2 タイマ割り込み有効（タイマ2）

このビットに1をセットすると、タイマ番号2のタイマ割り込みが有効になります。タイマ2を起動させ、タイマがタイムアウトすると割り込みが発生し、割り込み要因（Dio_ReadIntrpt関数参照）がセットされます。
タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。
注意： タイマ2が無いボードでは使用できません。

D7 TIM3 タイマ割り込み有効（タイマ3）

このビットに1をセットすると、タイマ番号3のタイマ割り込みが有効になります。タイマ3を起動させ、タイマがタイムアウトすると割り込みが発生し、割り込み要因（Dio_ReadIntrpt関数参照）がセットされます。
タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。
注意： タイマ3が無いボードでは使用できません。

例えば、TIM0 と OTS の割り込みを有効にし、その他を無効にする場合、14h（16 進数）を設定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

入力変化情報はDio_ReadInTransition関数で読み出し、読み出すとクリアされます。
入力変化以外の割り込み要因はDio_ReadIntrpt 関数で読み出し、読み出すとクリアされます。

本関数で割り込みを有効にした場合、発生した割り込みをアプリケーションに通知しない設定にした場合でも、指定した割り込みはボード内部で発生します。

■注意点

PD5006P, PD5106P, PD5306P の場合：
ボードの JP3 ジャンパーで命令を選択している場合、本関数で INSTB 信号変化時の割り込みを有効にしないでください。
ボードの JP2 ジャンパーで命令を選択している場合、本関数で OTSTB 信号変化時の割り込みを有効にしないでください。

■使用例

入力変化割り込みとタイマ1割り込みを有効に、それ以外の割り込みを無効に設定します。

[VC] Ret = Dio_SetIntrptMode(hBoard, 0x28);
[VB] Ret = Dio_SetIntrptMode(hBoard, &H28)
[C#] Ret = PD5000P. Dio_SetIntrptMode(hBoard, 0x28);

Dio_GetIoDir　入力／出力の設定読み出し

■対応ボード　：　PD5306P

■機能

Dio_SetIoDir関数で設定した入力／出力の設定を読み出します。
パソコン起動時は、すべての信号が入力の設定です。

■書式

[VC]　BOOL Dio_GetIoDir(HANDLE hBoard, ULONG* pRdData1, ULONG* pRdData2);

[VB]　Function Dio_GetIoDir(ByVal hBoard As Long, ByRef pRdData1 As Long, ByRef pRdData2 As Long) As Long

[VB.NET]　Function Dio_GetIoDir(ByVal hBoard As Integer, ByRef pRdData1 As Integer, ByRef pRdData2 As Integer) As Integer

[C#.NET]　bool PD5000P. Dio_GetIoDir(int hBoard, out uint RdData1, out uint RdData2);

■パラメータ

hBoard　Dio_Open 関数で取得したボードハンドルを指定します。

pRdData1　読み出すデータを格納する変数のポインタを指定します。
ポート0～3の各信号の入力／出力の設定を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 ：入力 （入力信号として使用する）
1 ：出力 （出力信号として使用する）

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

pRdData2 読み出すデータを格納する変数のポインタを指定します。
ポート4～7の各信号の入力／出力の設定を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力 （入力信号として使用する）
1 :出力 （出力信号として使用する）

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

入力／出力の設定を読み出します。Data1 変数にポート0～3の値を, Data2 変数にポート4～7の値を読み出します。

[VC] Ret = Dio_GetIoDir(hBoard, &Data1, &Data2);
[VB] Ret = Dio_GetIoDir(hBoard, Data1, Data2)
[C#] Ret = PD5000P. Dio_GetIoDir(hBoard, out Data1, out Data2);

Dio_GetInputLogic　入力論理設定読み出し

■機能

Dio_SetInputLogic関数で設定した入力論理の設定を読み出します。
パソコン起動時の入力論理は、すべての入力信号が Hi レベルを入力値1とする設定です。

■書式

[VC] BOOL Dio_GetInputLogic(HANDLE hBoard, ULONG* pRdData1, ULONG* pRdData2);

[VB] Function Dio_GetInputLogic(ByVal hBoard As Long, ByRef pRdData1 As Long, ByRef pRdData2 As Long) As Long

[VB.NET] Function Dio_GetInputLogic(ByVal hBoard As Integer, ByRef pRdData1 As Integer, ByRef pRdData2 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetInputLogic(int hBoard, out uint RdData1, out uint RdData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pRdData1 読み出すデータを格納する変数のポインタを指定します。
ポート0～3の各入力信号の論理レベルを読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力信号 Hi レベルを入力値1
1 :入力信号 Low レベルを入力値1

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

pRdData2 読み出すデータを格納する変数のポインタを指定します。
ポート4～7の各入力信号の論理レベルを読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力信号 Hi レベルを入力値1
1 :入力信号 Low レベルを入力値1

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

入力論理の設定を読み出します。Data1 変数にポート0～3の値を, Data2 変数にポート4～7の値を読み出します。

[VC] Ret = Dio_GetInputLogic(hBoard, &Data1, &Data2);
[VB] Ret = Dio_GetInputLogic(hBoard, Data1, Data2)
[C#] Ret = PD5000P. Dio_GetInputLogic(hBoard, out Data1, out Data2);

Dio_GetInputFilter　入力フィルタ指定読み出し

■機能

Dio_SetInputFilter関数で設定した入力フィルタの設定を読み出します。
パソコン起動時は、すべての入力信号にフィルタ時定数1が設定されます。

■書式

[VC] BOOL Dio_GetInputFilter(HANDLE hBoard, ULONG* pRdData);

[VB] Function Dio_GetInputFilter(ByVal hBoard As Long, ByRef pRdData As Long) As Long

[VB.NET] Function Dio_GetInputFilter(ByVal hBoard As Integer, ByRef pRdData As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetInputFilter(int hBoard, out uint RdData);

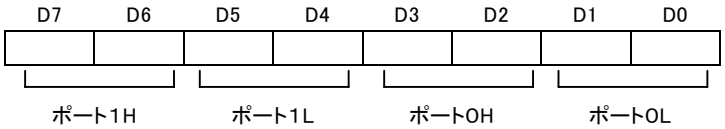
■パラメータ

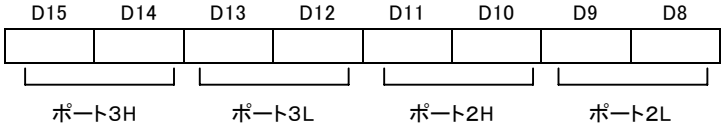
hBoard Dio_Open 関数で取得したボードハンドルを指定します。

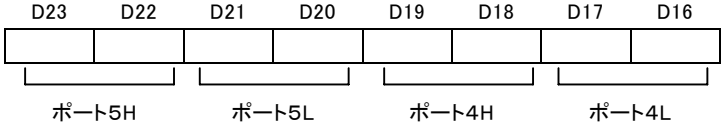
pRdData 読み出すデータを格納する変数のポインタを指定します。
全ポートの入力信号4点ごとのフィルタ時定数を読み出します。
下表の3種類のフィルタ時定数(1，2，3)とスルーのうち1つの値が、4点ごとに設定されています。

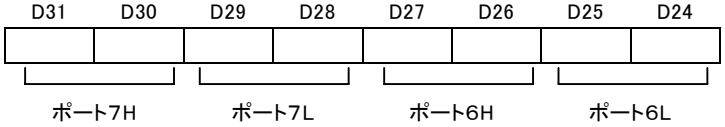
指定値 (上位ビット、下位ビット)	フィルタ指定内容
0，0	スルー(フィルタなし)
0，1	時定数1のフィルタが設定される
1，0	時定数2のフィルタが設定される
1，1	時定数3のフィルタが設定される

この変数の D0～31 に、各ポートの上位4点／下位4点のフィルタ時定数(2ビット単位でセットされています)を読み出します。下図のポート**H はポート**の上位4点、ポート**L はポート**の下位4点を表します。
ポート番号(H／L)の所に、その4点に対する値がセットされます。









PD5206P の場合:

Dio_SetInputFilter 関数で出力ポート(ポート4～7)に 00 以外を設定しても 00 が設定されます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

Data 変数に入力フィルタの設定を読み出します。

```
[ VC ]    Ret = Dio_GetInputFilter(hBoard, &Data);
[ VB ]    Ret = Dio_GetInputFilter(hBoard, Data)
[ C# ]    Ret = PD5000P. Dio_GetInputFilter(hBoard, out Data);
```

Dio_GetFilterTime　フィルタ時定数の設定読み出し

■機能

Dio_SetFilterTime関数で設定したフィルタ時定数の設定を読み出します。
パソコン起動時、時定数1は7, 時定数2は10, 時定数3は14が設定されます。

■書式

[VC] BOOL Dio_GetFilterTime(HANDLE hBoard, ULONG* pFilter1, ULONG* pFilter2, ULONG* pFilter3);

[VB] Function Dio_GetFilterTime(ByVal hBoard As Long, ByRef pFilter1 As Long, ByRef pFilter2 As Long, ByRef pFilter3 As Long) As Long

[VB.NET] Function Dio_GetFilterTime(ByVal hBoard As Integer, ByRef pFilter1 As Integer, ByRef pFilter2 As Integer, ByRef pFilter3 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetFilterTime(int hBoard, out FILTER Filter1, out FILTER Filter2, out FILTER Filter3);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pFilter1 読み出すデータを格納する変数のポインタを指定します。
フィルタ時定数1に設定した値を読み出します。
設定値についてはDio_SetFilterTime関数の表を参照してください。

pFilter2 読み出すデータを格納する変数のポインタを指定します。
フィルタ時定数2に設定した値を読み出します。
設定値についてはDio_SetFilterTime関数の表を参照してください。

pFilter3 読み出すデータを格納する変数のポインタを指定します。
フィルタ時定数3に設定した値を読み出します。
設定値についてはDio_SetFilterTime関数の表を参照してください。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

Filter1～3 変数にフィルタ時定数1～3の設定値を読み出します。

[VC] Ret = Dio_GetFilterTime(hBoard, &Filter1, &Filter2, &Filter3);
[VB] Ret = Dio_GetFilterTime(hBoard, Filter1, Filter2, Filter3)
[C#] Ret = PD5000P. Dio_GetFilterTime(hBoard, out Filter1, out Filter2, out Filter3);

Dio_GetTimer　タイマ値設定読み出し

■機能

Dio_SetTimer関数で設定したタイマ値の設定を読み出します。
パソコン起動時、すべてのタイマ値は0μsec です。

■書式

[VC] BOOL Dio_GetTimer(HANDLE hBoard, ULONG TimerNo, ULONG* pTimerValue, ULONG* pUnit);

[VB] Function Dio_GetTimer(ByVal hBoard As Long, ByVal TimerNo As Long, ByRef pTimerValue As Long, ByRef pUnit As Long) As Long

[VB.NET] Function Dio_GetTimer(ByVal hBoard As Integer, ByVal TimerNo As Integer, ByRef pTimerValue As Integer, ByRef pUnit As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetTimer(int hBoard, TIMER TimerNo, out uint TimerValue, out TSCALE pUnit);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

TimerNo タイマ番号を指定します。

PD5306P の場合:
[VC] [VB] タイマ番号として0～3のいずれかを指定します。(4個のタイマを使用できます)
[C#] タイマ番号として0～3のいずれかを指定します。(4個のタイマを使用できます)
TIMER.TIMER0 :0 TIMER.TIMER1 :1 TIMER.TIMER2 :2

PD5006P, PD5106P, PD5206P の場合:
[VC] [VB] タイマ番号として0、または1を指定します。(2個のタイマを使用できます)
[C#] タイマ番号として0、または1を指定します。(2個のタイマを使用できます)
TIMER.TIMER0 :0 TIMER.TIMER1 :1

pTimerValue 読み出すデータを格納する変数のポインタを指定します。
タイマ値を読み出します。タイマ値は 0～32,767 の範囲で設定されています。

pUnit 読み出すデータを格納する変数のポインタを指定します。
タイマ値の単位を読み出します。

[VC] [VB] 0 :μsec 単位、 1 :msec 単位
[C#] TSCALE . MICRO :μsec 単位、 TSCALE . MILLI :msec 単位

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

タイマ0のタイマ値を読み出します。TimerValue 変数にタイマ値を, Unit 変数にタイマ値の単位を読み出します。

[VC] Ret = Dio_GetTimer(hBoard, 0, &TimerValue, &Unit);
[VB] Ret = Dio_GetTimer(hBoard, 0, TimerValue, Unit)
[C#] Ret = PD5000P. Dio_GetTimer(hBoard, TIMER.TIMER0, out TimerValue, out pUnit);

Dio_GetInTransitionMode　入力変化有効設定読み出し

■機能

Dio_SetInTransitionMode関数で設定した入力変化有効設定を読み出します。
パソコン起動時は、すべての入力信号について、入力変化無効設定（入力変化を捉えない）です。

■書式

[VC] BOOL Dio_GetInTransitionMode(HANDLE hBoard, ULONG* pRdData1, ULONG* pRdData2);

[VB] Function Dio_GetInTransitionMode(ByVal hBoard As Long, ByRef pRdData1 As Long, ByRef pRdData2 As Long) As Long

[VB.NET] Function Dio_GetInTransitionMode(ByVal hBoard As Integer, ByRef pRdData1 As Integer, ByRef pRdData2 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetInTransitionMode(int hBoard, out uint RdData1, out uint RdData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pRdData1 読み出すデータを格納する変数のポインタを指定します。
ポート0～3の各入力信号の入力変化有効／無効を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 ：入力変化無効（入力変化を捉えない）
1 ：入力変化有効（入力変化を捉える）

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

pRdData2 読み出すデータを格納する変数のポインタを指定します。
ポート4～7の各入力信号の入力変化有効／無効を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力変化無効(入力変化を捉えない)
1 :入力変化有効(入力変化を捉える)

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

入力変化有効設定を読み出します。Data1 変数にポート0～3の値を, Data2 変数にポート4～7の値を読み出します。

[VC] Ret = Dio_GetInTransitionMode(hBoard, &Data1, &Data2);
[VB] Ret = Dio_GetInTransitionMode(hBoard, Data1, Data2)
[C#] Ret = PD5000P. Dio_GetInTransitionMode(hBoard, out Data1, out Data2);

Dio_GetInTransitionDir　入力変化方向設定読み出し

■機能

Dio_SetInTransitionDir関数で設定した入力変化方向設定を読み出します。
パソコン起動時、すべての入力信号について、入力変化方向は「入力信号 Low から Hi の変化を捉える」です。

■書式

[VC] BOOL Dio_GetInTransitionDir(HANDLE hBoard, ULONG* pRdData1, ULONG* pRdData2);

[VB] Function Dio_GetInTransitionDir(ByVal hBoard As Long, ByRef pRdData1 As Long, ByRef pRdData2 As Long) As Long

[VB.NET] Function Dio_GetInTransitionDir(ByVal hBoard As Integer, ByRef pRdData1 As Integer, ByRef pRdData2 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetInTransitionDir(int hBoard, out uint RdData1, out uint RdData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pRdData1 読み出すデータを格納する変数のポインタを指定します。
ポート0～3の各入力信号の入力変化方向を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力論理設定前の入力信号の状態が Low から Hi に変化したときの変化を捉える
1 :入力論理設定前の入力信号の状態が Hi から Low に変化したときの変化を捉える

注意: 入力変化方向は、入力論理設定前の入力信号の状態について設定します。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00
	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10
	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20
	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

0 : Low から Hi の変化
1 : Hi から Low の変化

pRdData2 読み出すデータを格納する変数のポインタを指定します。
ポート4～7の各入力信号の入力変化方向を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力論理設定前の入力信号の状態が Low から Hi に変化したときの変化を捉える
1 :入力論理設定前の入力信号の状態が Hi から Low に変化したときの変化を捉える

注意: 入力変化方向は、入力論理設定前の入力信号の状態について設定します。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40
	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50
	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60
	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

0 : Low から Hi の変化
1 : Hi から Low の変化

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

入力変化方向設定を読み出します。Data1 変数にポート0～3の値を, Data2 変数にポート4～7の値を読み出します。

```
[ VC ]   Ret = Dio_GetInTransitionDir(hBoard, &Data1, &Data2);
[ VB ]   Ret = Dio_GetInTransitionDir(hBoard, Data1, Data2)
[ C# ]   Ret = PD5000P. Dio_GetInTransitionDir(hBoard, out Data1, out Data2);
```

Dio_GetStrobeDir　外部ストローブ信号変化方向設定読み出し

■機能

Dio_SetStrobeDir関数で設定した外部ストローブ信号変化方向設定を読み出します。
パソコン起動時、外部ストローブ信号(INSTB, OTSTB)の変化方向は、両方とも「立ち上がりを使用する」です。

■書式

[VC] BOOL Dio_GetStrobeDir(HANDLE hBoard, ULONG* pInstb, ULONG* pOtstb);

[VB] Function Dio_GetStrobeDir(ByVal hBoard As Long, ByRef pInstb As Long, ByRef pOtstb As Long) As Long

[VB.NET] Function Dio_GetStrobeDir(ByVal hBoard As Integer, ByRef pInstb As Integer, ByRef pOtstb As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetStrobeDir(int hBoard, out EDGE_TRG Instb, out EDGE_TRG Otstb);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pInstb 読み出すデータを格納する変数のポインタを指定します。
INSTB 信号の変化方向を読み出します。

[VC] [VB] 0 :立ち上がり、 1 :立ち下がり
[C#] EDGE_TRG.EDGE_P :立ち上がり、 EDGE_TRG.EDGE_M :立ち下がり

pOtstb 読み出すデータを格納する変数のポインタを指定します。
OTSTB 信号の変化方向を読み出します。

[VC] [VB] 0 :立ち上がり、 1 :立ち下がり
[C#] EDGE_TRG.EDGE_P :立ち上がり、 EDGE_TRG.EDGE_M :立ち下がり

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

Instb 変数に INSTB 信号の変化方向を, Otstb 変数に OTSTB 信号の変化方向を読み出します。

[VC] Ret = Dio_GetStrobeDir(hBoard, &Instb, &Otstb);
[VB] Ret = Dio_GetStrobeDir(hBoard, Instb, Otstb)
[C#] Ret = PD5000P. Dio_GetStrobeDir(hBoard, out Instb, out Otstb);

Dio_GetSmlInStbCmd　入力同時ラッチ有効設定読み出し(INSTB、命令)

■機能

Dio_SetSmlInStbCmd関数で設定した入力同時ラッチ有効設定(INSTB、命令)を読み出します。
パソコン起動時、入力同時ラッチ有効／無効設定(INSTB、命令)は無効設定になります。

■書式

[VC] BOOL Dio_GetSmlInStbCmd(HANDLE hBoard, ULONG* pEnabled);

[VB] Function Dio_GetSmlInStbCmd(ByVal hBoard As Long, ByRef pEnabled As Long) As Long

[VB.NET] Function Dio_GetSmlInStbCmd(ByVal hBoard As Integer, ByRef pEnabled As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetSmlInStbCmd(int hBoard, out ENABLED Enabled);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pEnabled 読み出すデータを格納する変数のポインタを指定します。
INSTB 信号、または命令による入力同時ラッチ機能の有効／無効設定を読み出します。

[VC] [VB] 0 :無効、 1 :有効
[C#] ENABLED.FALSE :無効、 ENABLED.TRUE :有効

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

Enabled 変数に入力同時ラッチ有効／無効設定(INSTB、命令)を読み出します。

[VC] Ret = Dio_GetSmlInStbCmd(hBoard, &Enabled);
[VB] Ret = Dio_GetSmlInStbCmd(hBoard, Enabled)
[C#] Ret = PD5000P. Dio_GetSmlInStbCmd(hBoard, out Enabled);

Dio_GetSmlInTimer　入力同時ラッチ有効設定読み出し(タイマ)

■対応ボード　:　PD5206P

■機能

Dio_SetSmlInTimer関数で設定した入力同時ラッチ有効設定(タイマ)を読み出します。
パソコン起動時、入力同時ラッチ有効／無効設定(タイマ)は無効設定になります。

■書式

[VC]　BOOL Dio_GetSmlInTimer(HANDLE hBoard, ULONG* pEnabled);

[VB]　Function Dio_GetSmlInTimer(ByVal hBoard As Long, ByRef pEnabled As Long) As Long

[VB.NET]　Function Dio_GetSmlInTimer(ByVal hBoard As Integer, ByRef pEnabled As Integer) As Integer

[C#.NET]　bool PD5000P. Dio_GetSmlInTimer(int hBoard, out ENABLED Enabled);

■パラメータ

hBoard　Dio_Open 関数で取得したボードハンドルを指定します。

pEnabled　読み出すデータを格納する変数のポインタを指定します。
タイマ0(タイマ番号0のタイマ)による入力同時ラッチ機能の有効／無効設定を読み出します。

[VC] [VB]　0 :無効、 1 :有効
[C#]　ENABLED.FALSE :無効、 ENABLED.TRUE :有効

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

Enabled 変数に入力同時ラッチ有効／無効設定(タイマ)を読み出します。

[VC]　Ret = Dio_GetSmlInTimer(hBoard, &Enabled);
[VB]　Ret = Dio_GetSmlInTimer(hBoard, Enabled)
[C#]　Ret = PD5000P. Dio_GetSmlInTimer(hBoard, out Enabled);

Dio_GetSmlOutStbCmd　出力同時セット有効設定読み出し(OTSTB、命令)

■機能

Dio_SetSmlOutStbCmd関数で設定した出力同時セット有効設定(OTSTB、命令)を読み出します。
パソコン起動時、出力同時セット有効／無効設定(OTSTB、命令)は無効設定になります。

■書式

[VC] BOOL Dio_GetSmlOutStbCmd(HANDLE hBoard, ULONG* pEnabled);

[VB] Function Dio_GetSmlOutStbCmd(ByVal hBoard As Long, ByRef pEnabled As Long) As Long

[VB.NET] Function Dio_GetSmlOutStbCmd(ByVal hBoard As Integer, ByRef pEnabled As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetSmlOutStbCmd(int hBoard, out ENABLED Enabled);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pEnabled 読み出すデータを格納する変数のポインタを指定します。
OTSTB 信号、または命令による出力同時セット機能の有効／無効設定を読み出します。

[VC] [VB] 0 :無効、 1 :有効
[C#] ENABLED.FALSE :無効、 ENABLED.TRUE :有効

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

Enabled 変数に出力同時セット有効／無効設定(OTSTB、命令)を読み出します。

[VC] Ret = Dio_GetSmlOutStbCmd(hBoard, &Enabled);
[VB] Ret = Dio_GetSmlOutStbCmd(hBoard, Enabled)
[C#] Ret = PD5000P. Dio_GetSmlOutStbCmd(hBoard, out Enabled);

Dio_GetSmlOutTimer　出力同時セット有効設定読み出し(タイマ)

■対応ボード　:　PD5206P

■機能

Dio_SetSmlOutTimer関数で設定した出力同時セット有効設定(タイマ)を読み出します。
パソコン起動時、出力同時セット有効／無効設定(タイマ)は無効設定になります。

■書式

[VC]　BOOL Dio_GetSmlOutTimer(HANDLE hBoard, ULONG* pEnabled);

[VB]　Function Dio_GetSmlOutTimer(ByVal hBoard As Long, ByRef pEnabled As Long) As Long

[VB.NET]　Function Dio_GetSmlOutTimer(ByVal hBoard As Integer, ByRef pEnabled As Integer) As Integer

[C#.NET]　bool　PD5000P. Dio_GetSmlOutTimer(int hBoard, out ENABLED Enabled);

■パラメータ

hBoard　Dio_Open 関数で取得したボードハンドルを指定します。

pEnabled　読み出すデータを格納する変数のポインタを指定します。
タイマ1(タイマ番号1のタイマ)による出力同時セット機能の有効／無効設定を読み出します。

[VC] [VB]　0 :無効、 1 :有効
[C#]　ENABLED.FALSE :無効、 ENABLED.TRUE :有効

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

Enabled 変数に出力同時セット有効／無効設定(タイマ)を読み出します。

[VC]　Ret = Dio_GetSmlOutTimer(hBoard, &Enabled);
[VB]　Ret = Dio_GetSmlOutTimer(hBoard, Enabled)
[C#]　Ret = PD5000P. Dio_GetSmlOutTimer(hBoard, out Enabled);

Dio_GetIntrptMode　割り込み設定読み出し

■機能

Dio_SetIntrptMode関数で設定した割り込み設定を読み出します。
パソコン起動時、すべての割り込み設定は無効設定になります。

■書式

[VC] BOOL Dio_GetIntrptMode(HANDLE hBoard, ULONG* pIntrptMode);

[VB] Function Dio_GetIntrptMode(ByVal hBoard As Long, ByRef pIntrptMode As Long) As Long

[VB.NET] Function Dio_GetIntrptMode(ByVal hBoard As Integer, ByRef pIntrptMode As Integer) As Integer

[C#.NET] bool PD5000P. Dio_GetIntrptMode(int hBoard, out uint pIntrptMode);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pIntrptMode 読み出すデータを格納する変数のポインタを指定します。
この変数の D1～7 ビットに、各割り込みの有効／無効設定を読み出します。
その他のビットには0がセットされます。

0 :無効、 1 :有効

D7	D6	D5	D4	D3	D2	D1	D0
TIM3	TIM2	TIM1	TIM0	TRN	OTS	INS	0

割り込み設定

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

D23	D22	D21	D20	D19	D18	D17	D16
0	0	0	0	0	0	0	0

D31	D30	D29	D28	D27	D26	D25	D24
0	0	0	0	0	0	0	0

● 各ビットの説明

D1	INS	INSTB 信号変化時の割り込み有効	1:有効、 0:無効
D2	OTS	OTSTB 信号変化時の割り込み有効	1:有効、 0:無効
D3	TRN	入力変化割り込み有効	1:有効、 0:無効
D4	TIM0	タイマ割り込み有効(タイマ0)	1:有効、 0:無効
D5	TIM1	タイマ割り込み有効(タイマ1)	1:有効、 0:無効
D6	TIM2	タイマ割り込み有効(タイマ2)	1:有効、 0:無効
D7	TIM3	タイマ割り込み有効(タイマ3)	1:有効、 0:無効

D1～7 ビットの詳細説明は、Dio_SetIntrptMode 関数の IntrptMode 変数の説明と同じです。そちらを参照してください。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■使用例

IntrptMode 変数に各割り込みの有効／無効設定を読み出します。

[VC]　Ret = Dio_GetIntrptMode(hBoard, &IntrptMode);
[VB]　Ret = Dio_GetIntrptMode(hBoard, IntrptMode)
[C#]　Ret = PD5000P. Dio_GetIntrptMode(hBoard, out IntrptMode);

Dio_In_1Bit　1ビット入力（1点）

■機能

指定した1点の入力値／出力値を読み出します。
指定した1点が入力信号の場合は入力値を、出力信号の場合は出力値を読み出します。
入力値／出力値については、「4.4.3 補足説明(1)」を参照してください。

■書式

[VC] BOOL Dio_In_1Bit(HANDLE hBoard, ULONG SignalNo, ULONG* pReadData);

[VB] Function Dio_In_1Bit(ByVal hBoard As Long, ByVal SignalNo As Long, ByRef pReadData As Long) As Long

[VB.NET] Function Dio_In_1Bit(ByVal hBoard As Integer, ByVal SignalNo As Integer, ByRef pReadData As Integer) As Integer

[C#.NET] bool PD5000P. Dio_In_1Bit(int hBoard, SIGNAL SignalNo, out uint pReadData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

SignalNo [VC] [VB] 読み出す信号(1点)の信号番号を指定します。
信号番号については、「4.4.3 補足説明(2)」を参照してください。

[C#] SIGNAL.SN00 :信号番号 00　SIGNAL.SN77 :信号番号 3F
信号番号については、「4.4.3 補足説明(2)」を参照してください。

pReadData 読み出すデータを格納する変数のポインタを指定します。
指定した信号(1点)の入力値／出力値を読み出します。読み出される値は0、または1です。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定した信号番号が存在しない場合、本関数はエラーになります。

■使用例

Data 変数に SN37 信号の入力値／出力値を読み出します。

[VC] Ret = Dio_In_1Bit(hBoard, 0x1F, &Data);
[VB] Ret = Dio_In_1Bit(hBoard, &H1F, Data)
[C#] Ret = PD5000P. Dio_In_1Bit(hBoard, SIGNAL.SN37, out Data);

Dio_In_1Byte　1バイト入力（8点）

■機能

指定したポートの入力値／出力値を1バイト（8点）読み出します。
指定したポートが入力ポートの場合は入力値を、出力ポートの場合は出力値を、入出力混在ポートの場合は入力値と出力値の両方（入力信号は入力値、出力信号は出力値）を読み出します。
入力値／出力値については、「4.4.3 補足説明（1）」を参照してください。

■書式

[VC] BOOL Dio_In_1Byte(HANDLE hBoard, ULONG PortNo, ULONG* pReadData);

[VB] Function Dio_In_1Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByRef pReadData As Long) As Long

[VB.NET] Function Dio_In_1Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByRef pReadData As Integer) As Integer

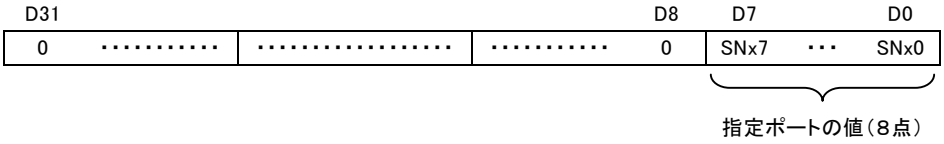
[C#.NET] bool PD5000P. Dio_In_1Byte(int hBoard, PORT PortNo, out uint pReadData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] ポート番号（0～7）を指定します。
[C#] ポート番号（0～7）を指定します。
PORT.PORT0 ：0　PORT.PORT7 ：7

pReadData 読み出すデータを格納する変数のポインタを指定します。
この変数の D0～7 ビットに、指定したポート番号の入力値／出力値を1バイト（8点）読み出します。
D8～31 ビットには0がセットされます。



下図の例のように、各信号の値は、D0～7 ビットに対して SNx0～SNx7 信号（x はポート番号）の順にセットされます。

例） ポート0を指定した場合

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0の値	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

例） ポート1を指定した場合

	D7	D6	D5	D4	D3	D2	D1	D0
ポート1の値	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定したポート番号が存在しない場合、本関数はエラーになります。

■使用例

Data 変数にポート3の入力値／出力値を8点分読み出します。

```
[ VC ]    Ret = Dio_In_1Byte(hBoard, 3, &Data);
[ VB ]    Ret = Dio_In_1Byte(hBoard, 3, Data)
[ C# ]    Ret = PD5000P. Dio_In_1Byte(hBoard, PORT.PORT3, out Data);
```

Dio_In_2Byte　2バイト入力（16点）

■機能

指定したポートの入力値／出力値を2バイト（16点）読み出します。
指定したポートが入力ポートの場合は入力値を、出力ポートの場合は出力値を、入出力混在ポートの場合は入力値と出力値の両方（入力信号は入力値、出力信号は出力値）を読み出します。
入力値／出力値については、「4.4.3 補足説明（1）」を参照してください。

■書式

[VC] BOOL Dio_In_2Byte(HANDLE hBoard, ULONG PortNo, ULONG* pReadData);

[VB] Function Dio_In_2Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByRef pReadData As Long) As Long

[VB.NET] Function Dio_In_2Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByRef pReadData As Integer) As Integer

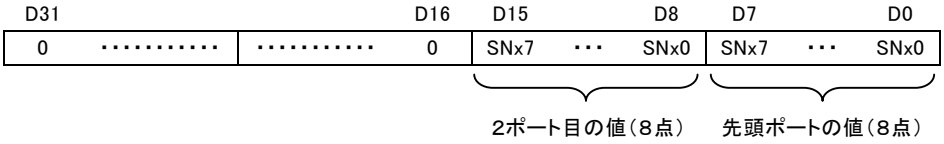
[C#.NET] bool PD5000P. Dio_In_2Byte(int hBoard, PORT PortNo, out uint pReadData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] ポート番号（0～7）を指定します。
[C#] ポート番号（0～7）を指定します。
PORT.PORT0 ：0　PORT.PORT7 ：7

pReadData 読み出すデータを格納する変数のポインタを指定します。
この変数の D0～15 ビットに、先頭ポート番号から2バイト（16点）の入力値／出力値を読み出します。
D16～31 ビットには0がセットされます。



D0～7 ビットには先頭ポートの値が、D8～15 ビットには2ポート目の値が入ります。
下図の例のように、各ポートの値は変数に8ビットずつセットされ、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号（x はポート番号）の順にセットされます。

例） 先頭ポートにポート0を指定した場合

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0の値	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1の値	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

2ポート目のポートが存在しない場合、2ポート目の値には0が入ります。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

先頭ポート番号が存在しない場合、本関数はエラーになります。
本関数は2バイト同時読み出しではないので、同時読み出しを行いたい場合は入力同時ラッチ機能を使用してください。

■使用例

Data 変数にポート2～3の入力値／出力値を16点分読み出します。

```
[ VC ]    Ret = Dio_In_2Byte(hBoard, 2, &Data);
[ VB ]    Ret = Dio_In_2Byte(hBoard, 2, Data)
[ C# ]    Ret = PD5000P. Dio_In_2Byte(hBoard, PORT.PORT2, out Data);
```

Dio_In_4Byte　4バイト入力（32点）

■機能

指定したポートの入力値／出力値を4バイト（32点）読み出します。
指定したポートが入力ポートの場合は入力値を、出力ポートの場合は出力値を、入出力混在ポートの場合は入力値と出力値の両方（入力信号は入力値、出力信号は出力値）を読み出します。
入力値／出力値については、「4.4.3 補足説明（1）」を参照してください。

■書式

［VC］ BOOL Dio_In_4Byte(HANDLE hBoard, ULONG PortNo, ULONG* pReadData);

［VB］ Function Dio_In_4Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByRef pReadData As Long) As Long

［VB.NET］ Function Dio_In_4Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByRef pReadData As Integer) As Integer

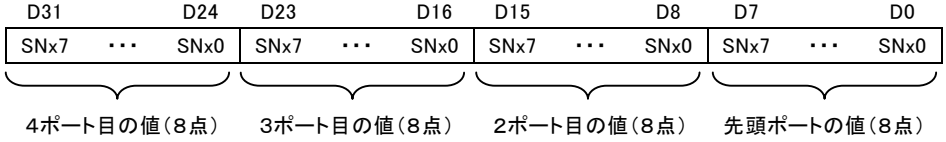
［C#.NET］ bool PD5000P. Dio_In_4Byte(int hBoard, PORT PortNo, out uint pReadData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] ポート番号(0～7)を指定します。
[C#] ポート番号(0～7)を指定します。
PORT.PORT0 :0 PORT.PORT7 :7

pReadData 読み出すデータを格納する変数のポインタを指定します。
この変数の D0～31 ビットに、先頭ポート番号から4バイト(32点)の入力値／出力値を読み出します。



D0～7 ビットには先頭ポートの値が、D8～15 ビットには2ポート目の値が、D16～23 ビットには3ポート目の値が、D24～31 ビットには4ポート目の値が入ります。下図の例のように、各ポートの値は変数に8ビットずつセットされ、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号(x はポート番号)の順にセットされます。

例) 先頭ポートにポート0を指定した場合

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0の値	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1の値	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2の値	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3の値	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

2ポート目以降のポートが存在しない場合、存在しないポートの値には0がセットされます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

先頭ポート番号が存在しない場合、本関数はエラーになります。
本関数は4バイト同時読み出しではないので、同時読み出しを行いたい場合は入力同時ラッチ機能を使用してください。

■使用例

Data 変数にポート0～3の入力値／出力値を32点分読み出します。

[VC] Ret = Dio_In_4Byte(hBoard, 0, &Data);
[VB] Ret = Dio_In_4Byte(hBoard, 0, Data)
[C#] Ret = PD5000P. Dio_In_4Byte(hBoard, PORT.PORT0, out Data);

Dio_In_8Byte　8バイト入力（64点）

■機能

指定したポートの入力値／出力値を8バイト（64点）読み出します。
指定したポートが入力ポートの場合は入力値を、出力ポートの場合は出力値を、入出力混在ポートの場合は入力値と出力値の両方（入力信号は入力値、出力信号は出力値）を読み出します。
入力値／出力値については、「4.4.3 補足説明（1）」を参照してください。

■書式

[VC] BOOL Dio_In_8Byte(HANDLE hBoard, ULONG PortNo, ULONG* pReadData1, ULONG* pReadData2);

[VB] Function Dio_In_8Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByRef pReadData1 As Long, ByRef pReadData2 As Long) As Long

[VB.NET] Function Dio_In_8Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByRef pReadData1 As Integer, ByRef pReadData2 As Integer) As Integer

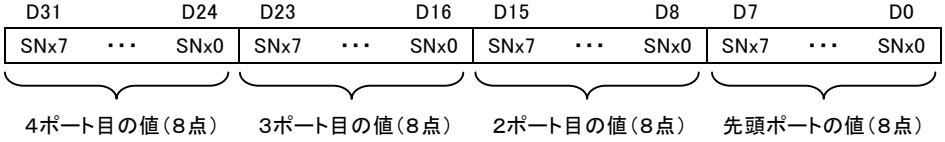
[C#.NET] bool PD5000P. Dio_In_8Byte(int hBoard, PORT PortNo, out uint pReadData1,
out uint pReadData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] ポート番号(0～7)を指定します。
[C#] ポート番号(0～7)を指定します。
PORT.PORT0 :0 PORT.PORT7 :7

pReadData1 読み出すデータを格納する変数のポインタを指定します。
この変数の D0～31 ビットに、先頭ポート番号から4バイト(32点)の入力値／出力値を読み出します。



D0～7 ビットには先頭ポートの値が、D8～15 ビットには2ポート目の値が、D16～23 ビットには3ポート目の値が、D24～31 ビットには4ポート目の値が入ります。下図の例のように、各ポートの値は変数に8ビットずつセットされ、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号(x はポート番号)の順にセットされます。

例) 先頭ポートにポート0を指定した場合

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0の値	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

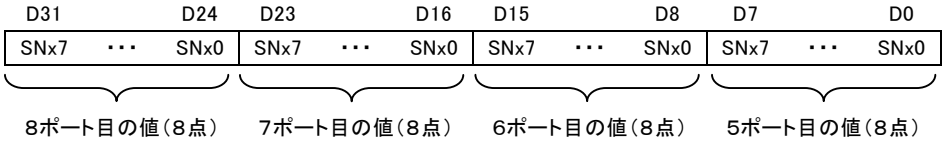
	D15	D14	D13	D12	D11	D10	D9	D8
ポート1の値	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2の値	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3の値	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

2ポート目以降のポートが存在しない場合、存在しないポートの値には0がセットされます。

pReadData2 読み出すデータを格納する変数のポインタを指定します。
この変数の D0～31 ビットに、5ポート目（先頭ポート番号＋4）から4バイト（32点）の入力値／出力値を読み出します。



D0～7 ビットには5ポート目の値が、D8～15 ビットには6ポート目の値が、D16～23 ビットには7ポート目の値が、D24～31 ビットには8ポート目の値が入ります。下図の例のように、各ポートの値は変数に8ビットずつセットされ、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号（x はポート番号）の順にセットされます。

例） 先頭ポートにポート0を指定した場合、ここではポート4～7の値を読み出します。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4の値	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5の値	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6の値	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7の値	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

5ポート目以降のポートが存在しない場合、存在しないポートの値には0がセットされます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

先頭ポート番号が存在しない場合、本関数はエラーになります。
本関数は8バイト同時読み出しではないので、同時読み出しを行いたい場合は入力同時ラッチ機能を使用してください。

■使用例

Data1 変数にポート0～3の値を32点分、Data2 変数にポート4～7の値を32点分読み出します。

```
[ VC ]   Ret = Dio_In_8Byte(hBoard, 0, &Data1, &Data2);
[ VB ]   Ret = Dio_In_8Byte(hBoard, 0, Data1, Data2)
[ C# ]   Ret = PD5000P. Dio_In_8Byte(hBoard, PORT.PORT0, out Data1, out Data2);
```

Dio_In_nBit　任意複数ビット入力

■機能

指定した複数の信号の入力値／出力値を1点ずつ読み出します。
指定した1点が入力信号の場合は入力値を、出力信号の場合は出力値を読み出します。
入力値／出力値については、「4.4.3 補足説明(1)」を参照してください。

■書式

[VC] BOOL Dio_In_nBit(HANDLE hBoard, ULONG Cnt, ULONG* pSignalNoArray, ULONG* pRdDataArray);

[VB] Function Dio_In_nBit(ByVal hBoard As Long, ByVal Cnt As Long, ByRef pSignalNoArray As Long, ByRef pRdDataArray As Long) As Long

[VB.NET] Function Dio_In_nBit(ByVal hBoard As Integer, ByVal Cnt As Integer, ByRef pSignalNoArray As Integer, ByRef pRdDataArray As Integer) As Integer

[C#.NET] bool PD5000P._Dio_In_nBit(int hBoard, uint Cnt, ref SIGNAL[] pSignalNoArray, ref uint[] pRdDataArray)
PD5000P._Dio_In_nBit関数については、「4.4.3 補足説明(5)」を参照してください。
関数名 Dio の前にアンダーバー “_”がついていますのでご注意ください。

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Cnt 読み出す信号の数を指定します。

pSignalNoArray 信号番号配列変数のポインタを指定します。
Cnt 数の配列を確保し、読み出す信号番号を配列にセットしてください。
信号番号については、「4.4.3 補足説明(2)」を参照してください。

pRdDataArray 読み出すデータを格納する変数(Cnt 数の配列変数)のポインタを指定します。
指定した複数の信号の入力値／出力値を1点ずつ読み出し配列にセットします。
pSignalNoArray 変数で信号番号を指定した配列番号と同じ配列番号のところに対応した値がセットされます。読み出される値は0、または1です。

例えば、pSignalNoArray 配列の0番目に信号番号1をセットすると、pRdDataArray 配列の0番目に信号番号1の値がセットされます。

例） Cnt 数4で、SN00, SN02, SN04, SN06 信号を読み出す場合

	[0]	[1]	[2]	[3]	
pSignalNoArray 変数	0	2	4	6	←信号番号セット

	[0]	[1]	[2]	[3]	
pRdDataArray 変数	SN00 の値	SN02 の値	SN04 の値	SN06 の値	←読み出される値

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定した信号番号が存在しない場合、本関数はエラーになります。
本関数は複数の信号を同時に読み出してはいません。

■使用例

RdDataArray 配列に SN20, SN40, SN77 信号の入力値／出力値を読み出します。

[VC]	ULONG SignalNoArray[3] = {0x10, 0x20, 0x3F}; ULONG RdDataArray[3]; Ret = Dio_In_nBit(hBoard, 3, SignalNoArray, RdDataArray);
[VB]	Dim SignalNoArray(2) As Long Dim RdDataArray(2) As Long SignalNoArray(0) = &H10: SignalNoArray(1) = &H20: SignalNoArray(2) = &H3F Ret = Dio_In_nBit(hBoard, 3, SignalNoArray(0), RdDataArray (0))
[VB.NET]	Dim SignalNoArray(2) As Integer Dim RdDataArray(2) As Integer SignalNoArray(0) = &H10: SignalNoArray(1) = &H20: SignalNoArray(2) = &H3F Ret = Dio_In_nBit(hBoard, 3, SignalNoArray(0), RdDataArray(0))
[C#]	uint SignalNoArray[3] = {0x10, 0x20, 0x3F}; uint RdDataArray[3]; Ret = PD5000P. _Dio_In_nBit (hBoard, 3, SignalNoArray, RdDataArray);

Dio_In_nByte　任意複数バイト入力

■機能

指定した複数ポートの入力値／出力値を1バイト(8点)ずつ読み出します。
指定したポートが入力ポートの場合は入力値を、出力ポートの場合は出力値を、入出力混在ポートの場合は入力値と出力値の両方(入力信号は入力値、出力信号は出力値)を読み出します。
入力値／出力値については、「4.4.3 補足説明(1)」を参照してください。

■書式

[VC] BOOL Dio_In_nByte(HANDLE hBoard, ULONG Cnt, ULONG* pPortNoArray, ULONG* pRdDataArray);

[VB] Function Dio_In_nByte(ByVal hBoard As Long, ByVal Cnt As Long, ByRef pPortNoArray As Long, ByRef pRdDataArray As Long) As Long

[VB.NET] Function Dio_In_nByte(ByVal hBoard As Integer, ByVal Cnt As Integer, ByRef pPortNoArray As Integer, ByRef pRdDataArray As Integer) As Integer

[C#.NET] bool PD5000P._Dio_In_nByte(int hBoard, uint Cnt, ref PORT[] pPortNoArray, ref uint[] pRdDataArray)
PD5000P._Dio_In_ nByt関数については、「4.4.3 補足説明(5)」を参照してください。
関数名 Dio の前にアンダーバー “_”がついていますのでご注意ください。

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Cnt 読み出すポートの数を指定します。

pPortNoArray ポート番号配列変数のポインタを指定します。
Cnt 数の配列を確保し、読み出すポート番号（0～7）を配列にセットしてください。

pRdDataArray 読み出すデータを格納する変数（Cnt 数の配列変数）のポインタを指定します。
指定した複数ポートの入力値／出力値を1バイト（8点）ずつ読み出し配列にセットします。
pPortNoArray 変数でポート番号を指定した配列番号と同じ配列番号のところに対応した値がセットされます。

この変数の各配列 D0～7 ビットに、指定ポートの信号8点の値を読み出します。下図の例のように、各ポートの値は、D0～7 ビットに対して SNx0～SNx7 信号（x はポート番号）の順にセットされます。また、D8～31 ビットには0がセットされます。

例） pRdDataArray [0]にポート0の値を読み出した場合

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0の値	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

例えば、pPortNoArray 配列の0番目にポート番号1をセットすると、pRdDataArray 配列の0番目にポート番号1の値がセットされます。

例） Cnt 数4で、ポート0，2，4，6 の値を読み出す場合

	[0]	[1]	[2]	[3]	
pPortNoArray 変数	0	2	4	6	←ポート番号セット

	[0]	[1]	[2]	[3]	
pRdDataArray 変数	ポート0の値	ポート2の値	ポート4の値	ポート6の値	←読み出される値

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定したポート番号が存在しない場合、本関数はエラーになります。
本関数は複数ポートの値を同時に読み出してはいません。

■使用例

RdDataArray 配列にポート0, 2, 4, 6の入力値／出力値を8点ずつ読み出します。

[VC]	ULONG PortNoArray[4] = {0, 2, 4, 6}; ULONG RdDataArray[4]; Ret = Dio_In_nByte(hBoard, 4, PortNoArray, RdDataArray);
[VB]	Dim PortNoArray(3) As Long Dim RdDataArray(3) As Long PortNoArray(0) = 0: PortNoArray(1) = 2: PortNoArray(2) = 4: PortNoArray(3) = 6 Ret = Dio_In_nByte(hBoard, 4, PortNoArray(0), RdDataArray(0))
[VB.NET]	Dim PortNoArray(3) As Integer Dim RdDataArray(3) As Integer PortNoArray(0) = 0: PortNoArray(1) = 2: PortNoArray(2) = 4: PortNoArray(3) = 6 Ret = Dio_In_nByte(hBoard, 4, PortNoArray(0), RdDataArray(0))
[C#]	uint PortNoArray[4] = {0, 2, 4, 6}; uint RdDataArray[4]; Ret = PD5000P._ Dio_In_nByte(hBoard, 4, ref PortNoArray, ref RdDataArray);

Dio_Out_1Bit　1ビット出力（1点）

■機能

指定した信号1点に対する出力値を設定します。
本関数で出力値を設定すると、指定した信号1点が指定した出力レベル（出力信号 Hi、または Low）になります。

■書式

[VC] BOOL Dio_Out_1Bit(HANDLE hBoard, ULONG SignalNo, ULONG WriteData);

[VB] Function Dio_Out_1Bit(ByVal hBoard As Long, ByVal SignalNo As Long, ByVal WriteData As Long) As Long

[VB.NET] Function Dio_Out_1Bit(ByVal hBoard As Integer, ByVal SignalNo As Integer, ByVal WriteData As Integer) As Integer

[C#.NET] bool PD5000P. Dio_Out_1Bit(int hBoard, SIGNAL SignalNo, uint WriteData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

SignalNo [VC] [VB] 出力する信号（1点）の信号番号を指定します。
信号番号については、「4.4.3 補足説明（2）」を参照してください。

[C#] SIGNAL.SN00 ：信号番号 00　SIGNAL.SN77 ：信号番号 3F
信号番号については、「4.4.3 補足説明（2）」を参照してください。

WriteData 指定した信号（1点）に対する出力値を指定します。設定する値は0、または1です。

0 ：出力信号 Hi になります
1 ：出力信号 Low になります

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定した信号が入力信号、または存在しない信号の場合、本関数はエラーになります。

PD5306P の場合：
入力／出力の設定（Dio_SetIoDir）関数で各信号の入力／出力を出力に設定していないと、本関数で出力値を設定しても出力されません。

■注意点

出力同時セット有効設定（OTSTB、命令）、または出力同時セット有効設定（タイマ）を有効にしている場合には、本関数の処理は無効となりますので、実行しないでください。本関数で出力同時セット機能を使用することはできません。

■使用例

SN77 信号に対して出力値1を設定します。

[VC] Ret = Dio_Out_1Bit(hBoard, 0x3F, 1);
[VB] Ret = Dio_Out_1Bit(hBoard, &H3F, 1)
[C#] Ret = PD5000P. Dio_Out_1Bit(hBoard, 0x3F, 1);

Dio_Out_1Byte　1バイト出力（8点）

■機能

出力値を1バイト(8点)設定します。
本関数で出力値を設定すると、指定したポートの8点が指定した出力レベル(出力信号 Hi、または Low)になります。
指定したポートが入出力混在ポートの場合は、出力信号に対してのみ出力値が設定され、入力信号に対しては出力値が設定されません。

■書式

[VC] BOOL Dio_Out_1Byte(HANDLE hBoard, ULONG PortNo, ULONG WriteData);

[VB] Function Dio_Out_1Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByVal WriteData As Long) As Long

[VB.NET] Function Dio_Out_1Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByVal WriteData As Integer) As Integer

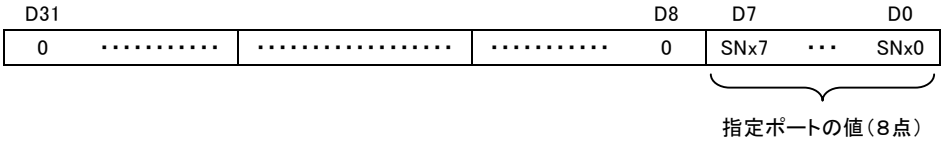
[C#.NET] bool PD5000P. Dio_Out_1Byte(int hBoard, PORT PortNo, uint WriteData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] ポート番号(0～7)を指定します。
[C#] ポート番号(0～7)を指定します。
PORT.PORT0 :0　PORT.PORT7 :7

WriteData 指定したポート番号の8点に対する出力値を1バイト(8ビット)指定します。
各信号に対して、出力値0をセットすると出力信号 Hi に、出力値1をセットすると出力信号 Low になります。
入力に設定されている信号に対してはどちらの値をセットしても構いません。
この変数の D0～7 ビットに、信号8点に対する出力値を設定します。D8～31 ビットには0を設定してください。



下図の例のように、各信号に対する出力値は、D0～7 ビットに対して SNx0～SNx7 信号(x はポート番号)の順にセットしてください。

例） ポート0を指定した場合

ポート0への出力値	D7	D6	D5	D4	D3	D2	D1	D0
	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

0 : 出力信号 Hi
1 : 出力信号 Low

例） ポート1を指定した場合

ポート1への出力値	D7	D6	D5	D4	D3	D2	D1	D0
	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定したポート番号が存在しない場合、本関数はエラーになります。
出力同時セット有効設定(OTSTB、命令)、または出力同時セット有効設定(タイマ)を有効にしている場合は、本関数で出力値を設定しても、直ちに出力信号は変化しません。詳細は出力同時セット有効設定を参照してください。

PD5306P の場合:
入力／出力の設定(Dio_SetIoDir)関数で各信号の入力／出力を出力に設定していないと、本関数で出力値を設定しても出力されません。

■使用例

ポート7の SN70,SN77 信号に対して出力値1を、ポート7のその他の信号に対して出力値0を設定します。

```
[ VC ]    Ret = Dio_Out_1Byte(hBoard, 7, 0x81);
[ VB ]    Ret = Dio_Out_1Byte(hBoard, 7, &H81)
[ C# ]    Ret = PD5000P. Dio_Out_1Byte(hBoard, PORT.PORT7, 0x81);
```

Dio_Out_2Byte　2バイト出力（16点）

■機能

出力値を2バイト（16点）設定します。
本関数で出力値を設定すると、指定した先頭ポートから16点が指定した出力レベル（出力信号 Hi、または Low）になります。
指定したポートに入出力混在ポート、または入力ポートが含まれる場合は、出力信号に対してのみ出力値が設定され、入力信号に対しては出力値が設定されません。

■書式

[VC] BOOL Dio_Out_2Byte(HANDLE hBoard, ULONG PortNo, ULONG WriteData);

[VB] Function Dio_Out_2Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByVal WriteData As Long) As Long

[VB.NET] Function Dio_Out_2Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByVal WriteData As Integer) As Integer

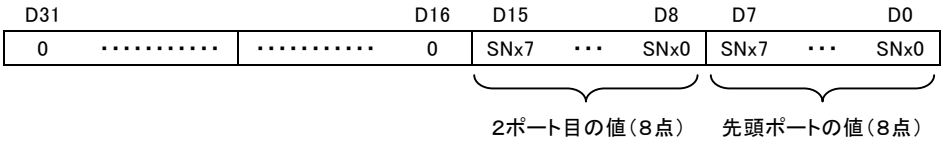
[C#.NET] bool PD5000P. Dio_Out_2Byte(int hBoard, PORT PortNo, uint WriteData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] 先頭ポート番号（0～7）を指定します。
[C#] ポート番号（0～7）を指定します。
PORT.PORT0 :0　PORT.PORT7 :7

WriteData 指定した先頭ポートからの16点に対する出力値を2バイト（16ビット）指定します。
各信号に対して、出力値0をセットすると出力信号 Hi に、出力値1をセットすると出力信号 Low になります。
入力に設定されている信号に対してはどちらの値をセットしても構いません。
この変数の D0～15 ビットに、信号16点に対する出力値を設定します。
D16～31 ビットには0を設定してください。



D0～7 ビットには先頭ポートの値を、D8～15 ビットには2ポート目の値を設定します。
下図の例のように、各ポートの値を変数に8ビットずつセットし、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号（x はポート番号）の順にセットしてください。

例） 先頭ポートにポート0を指定した場合、ここではポート0～1の値を設定します。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0への出力値	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

0 : 出力信号 Hi
1 : 出力信号 Low

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1への出力値	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

2ポート目のポートが存在しない場合、2ポート目は出力されません。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

先頭ポート番号が存在しない場合、本関数はエラーになります。
本関数は2バイト同時出力ではないので、同時出力を行いたい場合は出力同時セット機能を使用してください。

出力同時セット有効設定(OTSTB、命令)、または出力同時セット有効設定(タイマ)を有効にしている場合は、本関数で出力値を設定しても、直ちに出力信号は変化しません。詳細は出力同時セット有効設定を参照してください.。

PD5306P の場合:
入力／出力の設定(Dio_SetIoDir)関数で各信号の入力／出力を出力に設定していないと、本関数で出力値を設定しても出力されません。

■使用例

ポート4～5の SN40,SN57 信号に対して出力値1を、ポート4～5のその他の信号に対して出力値0を設定します。

```
[ VC ]      Ret = Dio_Out_2Byte(hBoard, 4, 0x8001);
[ VB ]      Ret = Dio_Out_2Byte(hBoard, 4, &H8001)
[ C# ]      Ret = PD5000P. Dio_Out_2Byte(hBoard, PORT.PORT4, 0x8001);
```

Dio_Out_4Byte　4バイト出力（32点）

■機能

出力値を4バイト（32点）設定します。
本関数で出力値を設定すると、指定した先頭ポートから32点が指定した出力レベル（出力信号 Hi、または Low）になります。
指定したポートに入出力混在ポート、または入力ポートが含まれる場合は、出力信号に対してのみ出力値が設定され、入力信号に対しては出力値が設定されません。

■書式

[VC] BOOL Dio_Out_4Byte(HANDLE hBoard, ULONG PortNo, ULONG WriteData);

[VB] Function Dio_Out_4Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByVal WriteData As Long) As Long

[VB.NET] Function Dio_Out_4Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByVal WriteData As Integer) As Integer

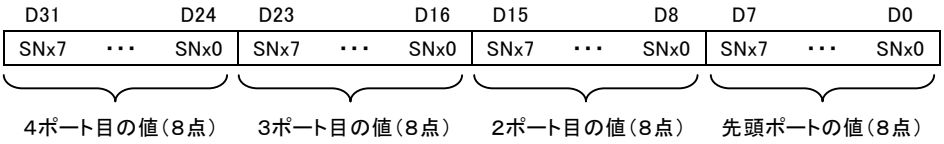
[C#.NET] bool PD5000P. Dio_Out_4Byte(int hBoard, PORT PortNo, uint WriteData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] 先頭ポート番号(0～7)を指定します。
[C#] ポート番号(0～7)を指定します。
PORT.PORT0 :0 PORT.PORT7 :7

WriteData 指定した先頭ポートからの32点に対する出力値を4バイト(32ビット)指定します。
各信号に対して、出力値0をセットすると出力信号 Hi に、出力値1をセットすると出力信号 Low になります。
入力に設定されている信号に対してはどちらの値をセットしても構いません。
この変数の D0～31 ビットに、信号32点に対する出力値を設定します。



D0～7 ビットには先頭ポートの値を、D8～15 ビットには2ポート目の値を、D16～23 ビットには3ポート目の値を、D24～31 ビットには4ポート目の値を設定します。
下図の例のように、各ポートの値を変数に8ビットずつセットし、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号(x はポート番号)の順にセットしてください。

例) 先頭ポートにポート0を指定した場合、ここではポート0～3の値を設定します。

ポート0への出力値	D7	D6	D5	D4	D3	D2	D1	D0
	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00
ポート1への出力値	D15	D14	D13	D12	D11	D10	D9	D8
	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10
ポート2への出力値	D23	D22	D21	D20	D19	D18	D17	D16
	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20
ポート3への出力値	D31	D30	D29	D28	D27	D26	D25	D24
	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

0 : 出力信号 Hi
1 : 出力信号 Low

2ポート目以降のポートが存在しない場合、存在しないポートには出力されません。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

先頭ポート番号が存在しない場合、本関数はエラーになります。
本関数は4バイト同時出力ではないので、同時出力を行いたい場合は出力同時セット機能を使用してください。

出力同時セット有効設定(OTSTB、命令)、または出力同時セット有効設定(タイマ)を有効にしている場合は、本関数で出力値を設定しても、直ちに出力信号は変化しません。詳細は出力同時セット有効設定を参照してください.。

PD5306P の場合:
入力／出力の設定(Dio_SetIoDir)関数で各信号の入力／出力を出力に設定していないと、本関数で出力値を設定しても出力されません。

■使用例

ポート4～7の SN40,SN77 信号に対して出力値1を、ポート4～7のその他の信号に対して出力値0を設定します。

[VC] Ret = Dio_Out_4Byte(hBoard, 4, 0x80000001);
[VB] Ret = Dio_Out_4Byte(hBoard, 4, &H80000001)
[C#] Ret = PD5000P. Dio_Out_4Byte(hBoard, PORT.PORT4, 0x80000001);

Dio_Out_8Byte　8バイト出力（64点）

■機能

出力値を8バイト（64点）設定します。
本関数で出力値を設定すると、指定した先頭ポートから64点が指定した出力レベル（出力信号 Hi、または Low）になります。
指定したポートに入出力混在ポート、または入力ポートが含まれる場合は、出力信号に対してのみ出力値が設定され、入力信号に対しては出力値が設定されません。

■書式

[VC] BOOL Dio_Out_8Byte(HANDLE hBoard, ULONG PortNo, ULONG WriteData1, ULONG WriteData2);

[VB] Function Dio_Out_8Byte(ByVal hBoard As Long, ByVal PortNo As Long, ByVal WriteData1 As Long, ByVal WriteData2 As Long) As Long

[VB.NET] Function Dio_Out_8Byte(ByVal hBoard As Integer, ByVal PortNo As Integer, ByVal WriteData1 As Integer, ByVal WriteData2 As Integer) As Integer

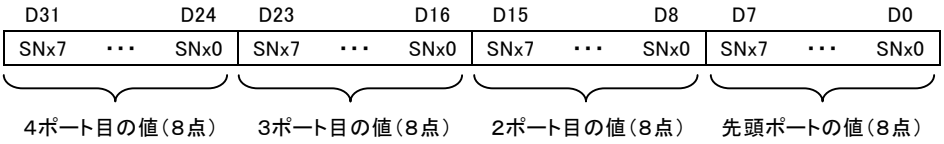
[C#.NET] bool PD5000P. Dio_Out_8Byte(int hBoard, PORT PortNo, uint WriteData1, uint WriteData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

PortNo [VC] [VB] 先頭ポート番号(0～7)を指定します。
[C#] ポート番号(0～7)を指定します。
PORT.PORT0 :0 PORT.PORT7 :7

WriteData1 先頭ポートからの32点に対する出力値を4バイト(32ビット)指定します。
各信号に対して、出力値0をセットすると出力信号 Hi に、出力値1をセットすると出力信号 Low になります。
入力に設定されている信号に対してはどちらの値をセットしても構いません。
この変数の D0～31 ビットに、信号32点に対する出力値を設定します。



D0～7 ビットには先頭ポートの値を、D8～15 ビットには2ポート目の値を、D16～23 ビットには3ポート目の値を、D24～31 ビットには4ポート目の値を設定します。
下図の例のように、各ポートの値を変数に8ビットずつセットし、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号(x はポート番号)の順にセットしてください。

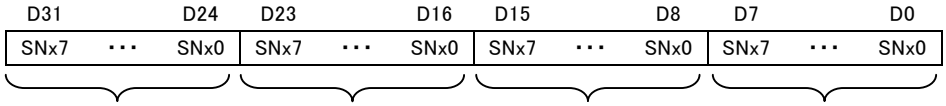
例) 先頭ポートにポート0を指定した場合、ここではポート0～3の値を設定します。

ポート0への出力値	D7	D6	D5	D4	D3	D2	D1	D0
	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00
ポート1への出力値	D15	D14	D13	D12	D11	D10	D9	D8
	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10
ポート2への出力値	D23	D22	D21	D20	D19	D18	D17	D16
	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20
ポート3への出力値	D31	D30	D29	D28	D27	D26	D25	D24
	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

0 : 出力信号 Hi
1 : 出力信号 Low

2ポート目以降のポートが存在しない場合、存在しないポートには出力されません。

WriteData2 5ポート目（先頭ポート番号＋4）からの32点に対する出力値を4バイト（32ビット）指定します。
各信号に対して、出力値0をセットすると出力信号 Hi に、出力値1をセットすると出力信号 Low になります。
入力に設定されている信号に対してはどちらの値をセットしても構いません。
この変数の D0～31 ビットに、信号32点に対する出力値を設定します。



D0～7 ビットには5ポート目の値を、D8～15 ビットには6ポート目の値を、D16～23 ビットには7ポート目の値を、D24～31 ビットには8ポート目の値を設定します。
下図の例のように、各ポートの値を変数に8ビットずつセットし、その8ビットの最下位ビット～最上位ビットに対して SNx0～SNx7 信号（x はポート番号）の順にセットしてください。

例） 先頭ポートにポート0を指定した場合、ここではポート4～7の値を設定します。

ポート4への出力値	D7	D6	D5	D4	D3	D2	D1	D0
	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40
ポート5への出力値	D15	D14	D13	D12	D11	D10	D9	D8
	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50
ポート6への出力値	D23	D22	D21	D20	D19	D18	D17	D16
	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60
ポート7への出力値	D31	D30	D29	D28	D27	D26	D25	D24
	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

0 ： 出力信号 Hi
1 ： 出力信号 Low

5ポート目以降のポートが存在しない場合、存在しないポートには出力されません。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

先頭ポート番号が存在しない場合、本関数はエラーになります。
本関数は8バイト同時出力ではないので、同時出力を行いたい場合は出力同時セット機能を使用してください。

出力同時セット有効設定（OTSTB、命令）、または出力同時セット有効設定（タイマ）を有効にしている場合は、本関数で出力値を設定しても、直ちに出力信号は変化しません。詳細は出力同時セット有効設定を参照してください.。

PD5306P の場合：
入力／出力の設定（Dio_SetIoDir）関数で各信号の入力／出力を出力に設定していないと、本関数で出力値を設定しても出力されません。

■使用例

ポート0～3の全信号に対して出力値1を、ポート4～7の全信号に対して出力値0を設定します。

```
[ VC ]   Ret = Dio_Out_8Byte(hBoard, 0, 0xFFFFFFFF, 0);
[ VB ]   Ret = Dio_Out_8Byte(hBoard, 0, &HFFFFFFFF, 0)
[ C# ]   Ret = PD5000P. Dio_Out_8Byte(hBoard, PORT.PORT0, 0xFFFFFFFF, 0);
```

Dio_Out_nBit　任意複数ビット出力

■機能

指定した複数の信号に対する出力値を1点ずつ設定します。
本関数で出力値を設定すると、指定した各信号が指定した出力レベル（出力信号 Hi、または Low）になります。

■書式

[VC] BOOL Dio_Out_nBit(HANDLE hBoard, ULONG Cnt, ULONG* pSignalNoArray, ULONG* pWtDataArray);

[VB] Function Dio_Out_nBit(ByVal hBoard As Long, ByVal Cnt As Long, ByRef pSignalNoArray As Long, ByRef pWtDataArray As Long) As Long

[VB.NET] Function Dio_Out_nBit(ByVal hBoard As Integer, ByVal Cnt As Integer, ByRef pSignalNoArray As Integer, ByRef pWtDataArray As Integer) As Integer

[C#.NET] bool PD5000P._Dio_Out_nBit(int hBoard, uint Cnt, ref SIGNAL[] pSignalNoArray, ref uint[] pWtDataArray)
PD5000P._ Dio_Out_nBit関数については、「4.4.3 補足説明（5）」を参照してください。
関数名 Dio の前にアンダーバー “_”がついていますのでご注意ください。

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Cnt 出力する信号の数を指定します。

pSignalNoArray 信号番号配列変数のポインタを指定します。
Cnt 数の配列を確保し、出力する信号番号を配列にセットしてください。
信号番号については、「4.4.3 補足説明（2）」を参照してください。

pWtDataArray 出力値配列変数のポインタを指定します。
Cnt 数の配列を確保し、各信号への出力値を1点ずつ配列にセットしてください。
pSignalNoArray 変数で信号番号を指定した配列番号と同じ配列番号のところに対応した値をセットしてください。設定する値は0、または1です。

0 ：出力信号 Hi になります
1 ：出力信号 Low になります

例えば、pSignalNoArray 配列の0番目に信号番号1をセットした場合、pWtDataArray 配列の0番目に信号番号1の出力値（0、または1）をセットします。

例） Cnt 数4で、SN00, SN02, SN04, SN06 信号に対して出力する場合

	[0]	[1]	[2]	[3]	
pSignalNoArray 変数	0	2	4	6	←信号番号セット

	[0]	[1]	[2]	[3]	
pWtDataArray 変数	SN00 の値	SN02 の値	SN04 の値	SN06 の値	←出力値セット

出力値 0 ： 出力信号 Hi
出力値 1 ： 出力信号 Low

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定した信号が入力信号、または存在しない信号の場合、本関数はエラーになりますが、配列の途中でエラーになった場合、途中までは出力されます。
本関数は複数の信号に対して同時に出力していません。

PD5306P の場合：
入力／出力の設定（Dio_SetIoDir）関数で各信号の入力／出力を出力に設定していないと、本関数で出力値を設定しても出力されません。

■注意点

出力同時セット有効設定（OTSTB、命令）、または出力同時セット有効設定（タイマ）を有効にしている場合には、本関数の処理は無効となりますので、実行しないでください。本関数で出力同時セット機能を使用することはできません。

■使用例

SN40 信号に出力値1を、SN60 信号に出力値0を、SN77 信号に出力値1を書き込みます。

[VC]	ULONG SignalNoArray[3] = {0x20, 0x30, 0x3F}; ULONG WtDataArray[3] = {1,0,1}; Ret = Dio_Out_nBit(hBoard, 3, SignalNoArray, WtDataArray);
[VB]	Dim SignalNoArray(2) As Long Dim WtDataArray(2) As Long SignalNoArray(0) = &H20: SignalNoArray(1) = &H30: SignalNoArray(2) = &H3F WtDataArray(0) = 1: WtDataArray(1) = 0: WtDataArray(2) = 1 Ret = Dio_Out_nBit(hBoard, 3, SignalNoArray(0), WtDataArray(0))
[VB.NET]	Dim SignalNoArray(2) As Integer Dim WtDataArray(2) As Integer SignalNoArray(0) = &H20: SignalNoArray(1) = &H30: SignalNoArray(2) = &H3F WtDataArray(0) = 1: WtDataArray(1) = 0: WtDataArray(2) = 1 Ret = Dio_Out_nBit(hBoard, 3, SignalNoArray(0), WtDataArray(0))
[C#]	ULONG SignalNoArray[3] = {0x20, 0x30, 0x3F}; ULONG WtDataArray[3] = {1,0,1}; Ret = PD5000P._Dio_Out_nBit(hBoard, 3, SignalNoArray, WtDataArray);

Dio_Out_nByte　任意複数バイト出力

■機能

指定した複数ポートに対する出力値を1バイト(8点)ずつ設定します。
本関数で出力値を設定すると、指定した各ポートの8点が指定した出力レベル(出力信号 Hi、または Low)になります。
指定したポートに入出力混在ポート、または入力ポートが含まれる場合は、出力信号に対してのみ出力値が設定され、入力信号に対しては出力値が設定されません。

■書式

[VC] BOOL Dio_Out_nByte(HANDLE hBoard, ULONG Cnt, ULONG* pPortNoArray, ULONG* pWtDataArray);

[VB] Function Dio_Out_nByte(ByVal hBoard As Long, ByVal Cnt As Long, ByRef pPortNoArray As Long, ByRef pWtDataArray As Long) As Long

[VB.NET] Function Dio_Out_nByte(ByVal hBoard As Integer, ByVal Cnt As Integer, ByRef pPortNoArray As Integer, ByRef pWtDataArray As Integer) As Integer

[C#.NET] bool PD5000P._Dio_Out_nByte(int hBoard, uint Cnt, ref PORT[] pPortNoArray, ref uint[] pWtDataArray)
PD5000P._Dio_ Out _ nByte関数については、「4.4.3 補足説明(5)」を参照してください。
関数名 Dio の前にアンダーバー “_”がついていますのでご注意ください。

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

Cnt 出力するポートの数を指定します。

pPortNoArray ポート番号配列変数のポインタを指定します。
Cnt 数の配列を確保し、出力するポート番号(0～7)を配列にセットしてください。

pWtDataArray 出力値配列変数のポインタを指定します。
Cnt 数の配列を確保し、各ポートへの出力値を1バイト(8点)ずつ配列にセットしてください。
pPortNoArray 変数でポート番号を指定した配列番号と同じ配列番号のところに対応した値をセットしてください。各信号に対して、出力値0をセットすると出力信号 Hi に、出力値1をセットすると出力信号 Low になります。入力に設定されている信号に対してはどちらの値をセットしても構いません。

この変数の各配列 D0～7 ビットに、指定ポートの信号8点に対する出力値をセットしてください。下図の例のように、各ポートの値は、D0～7 ビットに対して SNx0～SNx7 信号(x はポート番号)の順にセットしてください。また、D8～31 ビットには0を設定してください。

例) pWtDataArray[0]にポート0への出力値をセットする場合

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0への出力値	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

0 : 出力信号 Hi
1 : 出力信号 Low

例えば、pPortNoArray 配列の0番目にポート番号1をセットした場合、pWtDataArray 配列の0番目にポート番号1の出力値をセットします。

例) Cnt 数4で、ポート0, 2, 4, 6に対して出力する場合

	[0]	[1]	[2]	[3]	
pPortNoArray 変数	0	2	4	6	←ポート番号セット

	[0]	[1]	[2]	[3]	
pWtDataArray 変数	ポート0の値	ポート2の値	ポート4の値	ポート6の値	←出力値セット

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

指定したポート番号が存在しない場合本関数はエラーになりますが、配列の途中でエラーになった場合、途中までは出力されます。
本関数は複数ポートに対して同時に出力していませんので、同時出力を行いたい場合は出力同時セット機能を使用してください。

出力同時セット有効設定(OTSTB、命令)、または出力同時セット有効設定(タイマ)を有効にしている場合は、本関数で出力値を設定しても、直ちに出力信号は変化しません。詳細は出力同時セット有効設定を参照してください.。

PD5306P の場合:
入力／出力の設定(Dio_SetIoDir)関数で各信号の入力／出力を出力に設定していないと、本関数で出力値を設定しても出力されません。

■使用例

ポート0に出力値 11h（16進数）を、ポート2に 22h を、ポート4に 33h を、ポート6に 44h を書き込みます。
（SN00, 04, 21, 25, 40, 41, 44, 45, 62, 66 信号に対して出力値1を、それ以外の信号に対して出力値0を設定します。）

[VC]	ULONG PortNoArray[4] = {0, 2, 4, 6}; ULONG WtDataArray[4] = {0x11,0x22,0x33, 0x44}; Ret = Dio_Out_nByte(hBoard, 4, PortNoArray, WtDataArray)
[VB]	Dim PortNoArray(3) As Long Dim WtDataArray(3) As Long PortNoArray(0) = 0: PortNoArray(1) = 2: PortNoArray(2) = 4: PortNoArray(3) = 6 WtDataArray(0) = &H11: WtDataArray(1) = &H22: WtDataArray(2) = &H33 WtDataArray(3) = &H44 Ret = Dio_Out_nByte(hBoard, 4, PortNoArray(0), WtDataArray(0))
[VB.NET]	Dim PortNoArray(3) As Integer Dim WtDataArray(3) As Integer PortNoArray(0) = 0: PortNoArray(1) = 2: PortNoArray(2) = 4: PortNoArray(3) = 6 WtDataArray(0) = &H11: WtDataArray(1) = &H22: WtDataArray(2) = &H33 WtDataArray(3) = &H44 Ret = Dio_Out_nByte(hBoard, 4, PortNoArray(0), WtDataArray(0))
[C#]	uint PortNoArray[4] = {0, 2, 4, 6}; uint WtDataArray[4] = {0x11,0x22,0x33, 0x44}; Ret = PD5000P. _Dio_Out_nByte(hBoard, 4, PortNoArray, WtDataArray)

Dio_ReadIntrpt　割り込み要因読み出し

■機能

割り込みを発生させた要因（入力変化は除く）を読み出します。割り込み要因は、一度読み出すとクリアされます。

■書式

[VC] BOOL Dio_ReadIntrpt(HANDLE hBoard, ULONG* pIntrptData);

[VB] Function Dio_ReadIntrpt(ByVal hBoard As Long, ByRef pIntrptData As Long) As Long

[VB.NET] Function Dio_ReadIntrpt(ByVal hBoard As Integer, ByRef pIntrptData As Integer) As Integer

[C#.NET] bool PD5000P. Dio_ReadIntrpt(int hBoard, out uint pIntrptData);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pIntrptData 読み出すデータを格納する変数のポインタを指定します。
割り込みを発生させた要因（入力変化は除く）を読み出します。

この変数の D1～7 ビットに、割り込み要因を読み出します。（D3 ビットは除く）
割り込みが発生した場合は対応するビットに1が、発生していない場合は0がセットされます。
D0, D3, D8～31 ビットには固定で0が入ります。

D7	D6	D5	D4	D3	D2	D1	D0
TIM3	TIM2	TIM1	TIM0	0	OTS	INS	0

割り込み要因

1：発生した
0：発生していない

● 各ビットの説明

D1	INS	INSTB 信号変化時の割り込みが発生した。	1：発生、 0：未発生
D2	OTS	OTSTB 信号変化時の割り込みが発生した。	1：発生、 0：未発生
D3	―――	このビットは使用しません。値は0固定です。	――――――――
D4	TIM0	タイマ0のタイマ割り込みが発生した。	1：発生、 0：未発生
D5	TIM1	タイマ1のタイマ割り込みが発生した。	1：発生、 0：未発生
D6	TIM2	タイマ2のタイマ割り込みが発生した。	1：発生、 0：未発生
D7	TIM3	タイマ3のタイマ割り込みが発生した。	1：発生、 0：未発生

D1～7 ビットの割り込みを発生させるためには、Dio_SetIntrptMode関数のIntrptMode変数（対応するD1～7 ビット）の割り込みを有効にします。

例えば、TIM0 割り込みが発生した場合、10h（16 進数）がセットされます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

割り込みを発生させるためには、Dio_SetIntrptMode関数で希望する割り込みを有効にしておく必要があります。

割り込み要因は割り込み発生時にドライバ内部に保存され、本関数で読み出したときにドライバ内部の割り込み要因情報がクリアされます。本関数で割り込み要因を読み出す前に複数の割り込みが発生した場合、ドライバ内部の割り込み要因情報には発生した割り込み要因が追加保存されていきます。

●読み出し例
① 本関数で割り込み要因読み出し・・・・・・ ドライバ内部に保存される割り込み要因情報はクリアされます。
② INS 割り込み発生・・・・・・・・・・・・・・・・ 割り込み要因情報 02h(16進数)がドライバ内部に保存されます。
③ OTS 割り込み発生・・・・・・・・・・・・・・・・ 割り込み要因情報 06h(16進数)がドライバ内部に保存されます。
④ TIM1 割り込み発生・・・・・・・・・・・・・・・・ 割り込み要因情報 26h(16進数)がドライバ内部に保存されます。
⑤ 本関数で割り込み要因読み出し・・・・・・ 読み出される割り込み要因は 26h(16進数)です。

■使用例

IntrptData 変数に割り込み要因を読み出します。

```
[ VC ]    Ret = Dio_ReadIntrpt(hBoard, &IntrptData);
[ VB ]    Ret = Dio_ReadIntrpt(hBoard, IntrptData)
[ C# ]    Ret = PD5000P.Dio_ReadIntrpt(hBoard, out IntrptData);
```

Dio_ReadInTransition　入力変化情報読み出し

■機能

入力変化とは、入力信号の変化を捉える機能です。
この関数は、入力変化情報を読み出します。入力変化情報は、一度読み出すとクリアされます。

■書式

[VC] BOOL Dio_ReadInTransition(HANDLE hBoard, ULONG* pInTransiData1,
ULONG* pInTransiData2);

[VB] Function Dio_ReadInTransition(ByVal hBoard As Long, ByRef pInTransiData1 As Long,
ByRef pInTransiData2 As Long) As Long

[VB.NET] Function Dio_ReadInTransition(ByVal hBoard As Integer, ByRef pInTransiData1 As Integer,
ByRef pInTransiData2 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_ReadInTransition(int hBoard, out uint pInTransiData1,
out uint pInTransiData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pInTransiData1 読み出すデータを格納する変数のポインタを指定します。
ポート0～3の各信号の入力変化情報を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力変化なし
1 :入力変化あり

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

例えば、SN00, SN11, SN22, SN33 信号に入力変化があり、その他の信号には入力変化がなかった場合、08040201h（16 進数）がセットされます。

pInTransiData2 読み出すデータを格納する変数のポインタを指定します。
ポート4～7の各信号の入力変化情報を読み出します。
この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。

0 :入力変化なし
1 :入力変化あり

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

例えば、SN44, SN55, SN66, SN77 信号に入力変化があり、その他の信号には入力変化がなかった場合、80402010h（16 進数）がセットされます。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

入力変化有効設定で有効にした入力信号が、入力変化方向設定で設定した方向に変化した場合、入力変化情報がセットされます。

入力変化割り込みを有効にした場合、入力変化情報は割り込み発生時にドライバ内部に保存され、本関数で読み出したときにドライバ内部の入力変化情報がクリアされます。本関数で入力変化情報を読み出す前に複数の入力変化割り込みが発生した場合、ドライバ内部の入力変化情報には発生した入力変化の情報が追加保存されていきます。

入力変化割り込みを有効にしていない状態で入力変化が複数回発生した場合は、IC（PIX132）内部に入力変化情報が追加保存されていきます。本関数実行時に、IC内部の入力変化情報とドライバ内部の入力変化情報（割り込み発生時に保存した情報）を両方合わせた情報が読み出されます。

●読み出し例

① 本関数で入力変化情報読み出し・・・・・ ドライバ内部に保存される入力変化情報はクリアされます。
IC内部に保存される入力変化情報はクリアされます。

② SN00 信号の入力変化発生
③ SN22 信号の入力変化発生
④ SN44 信号の入力変化発生
⑤ SN66 信号の入力変化発生

⑥ 本関数で入力変化情報読み出し・・・・・ 読み出される入力変化情報は、ポート0～3が 040001h（16進数）、
ポート4～7が 400010h（16進数）です。

■使用例

入力変化情報を読み出します。Data1 変数にポート0～3の値を, Data2 変数にポート4～7の値を読み出します。

```
[ VC ]   Ret = Dio_ReadInTransition(hBoard, &Data1, &Data2);
[ VB ]   Ret = Dio_ReadInTransition(hBoard, Data1, Data2)
[ C# ]   Ret = PD5000P.Dio_ReadInTransition(hBoard, out Data1, out Data2);
```

Dio_ReadLatch　入力同時ラッチデータ読み出し

■機能

入力同時ラッチ機能でラッチされた値（入力同時ラッチデータ）を読み出します。
本関数で読み出す値は、積分フィルタ通過後のラッチされた入力の値を、本関数実行時点（注1）の入力論理設定に従って変換した値です。また、本関数では出力値の読み出しは行いません。（注1:ラッチ時点の入力論理設定ではありません。）

■書式

[VC] BOOL Dio_ReadLatch(HANDLE hBoard, ULONG* pLatchData1, ULONG* pLatchData2);

[VB] Function Dio_ReadLatch(ByVal hBoard As Long, ByRef pLatchData1 As Long,
ByRef pLatchData2 As Long) As Long

[VB.NET] Function Dio_ReadLatch(ByVal hBoard As Integer, ByRef pLatchData1 As Integer,
ByRef pLatchData2 As Integer) As Integer

[C#.NET] bool PD5000P. Dio_ReadLatch(int hBoard, out uint pLatchData1, out uint pLatchData2);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

pLatchData1 読み出すデータを格納する変数のポインタを指定します。
ポート0～3の信号の入力同時ラッチデータを読み出します。

この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。
出力信号（または出力に設定している信号）のビットには0がセットされます。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート0	SN07	SN06	SN05	SN04	SN03	SN02	SN01	SN00

	D15	D14	D13	D12	D11	D10	D9	D8
ポート1	SN17	SN16	SN15	SN14	SN13	SN12	SN11	SN10

	D23	D22	D21	D20	D19	D18	D17	D16
ポート2	SN27	SN26	SN25	SN24	SN23	SN22	SN21	SN20

	D31	D30	D29	D28	D27	D26	D25	D24
ポート3	SN37	SN36	SN35	SN34	SN33	SN32	SN31	SN30

pLatchData2 読み出すデータを格納する変数のポインタを指定します。
ポート4～7の信号の入力同時ラッチデータを読み出します。

この変数の D0～31 ビットには、下図の信号名の所に、その信号に対する値がセットされます。
出力信号（または出力に設定している信号）のビットには0がセットされます。

	D7	D6	D5	D4	D3	D2	D1	D0
ポート4	SN47	SN46	SN45	SN44	SN43	SN42	SN41	SN40

	D15	D14	D13	D12	D11	D10	D9	D8
ポート5	SN57	SN56	SN55	SN54	SN53	SN52	SN51	SN50

	D23	D22	D21	D20	D19	D18	D17	D16
ポート6	SN67	SN66	SN65	SN64	SN63	SN62	SN61	SN60

	D31	D30	D29	D28	D27	D26	D25	D24
ポート7	SN77	SN76	SN75	SN74	SN73	SN72	SN71	SN70

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

入力同時ラッチデータは、入力同時ラッチした時点では入力論理前の値がラッチされ、本関数で値を読み出すときに読み出し時点の入力論理に従い読み出されます。ラッチ時点の入力論理ではありません。このため、ラッチしてからラッチデータを読み出すまでの間は入力論理を変更してはいけません。また、ラッチされた入力論理前の値については、次の入力同時ラッチの前まで同じ値が保持されます。

■注意点

入力同時ラッチを実行してから本関数で入力同時ラッチデータを読み出すまでの間は、入力論理設定を変更してはいけません。

PD5306P の場合：
入力同時ラッチを実行してから本関数で入力同時ラッチデータを読み出すまでの間は、入力／出力の設定を変更してはいけません。
本関数で値を読み出すときに入力信号に設定されている信号のラッチデータのみを読み出し、その時点で出力信号に設定されている信号のビットには0が設定されます。入力／出力の設定を頻繁に変更する場合は、注意してください。

■使用例

入力同時ラッチデータを読み出します。Data1 変数にポート0～3の値を, Data2 変数にポート4～7の値を読み出します。

```
[ VC ]    Ret = Dio_ReadLatch(hBoard, &Data1, &Data2);
[ VB ]    Ret = Dio_ReadLatch(hBoard, Data1, Data2)
[ C# ]    Ret = PD5000P.Dio_ReadLatch(hBoard, out Data1, out Data2);
```

Dio_ReadCurrentTimer　動作タイマ値読み出し

■機能

指定したタイマの動作タイマ値(現在動作中のタイマの経過時間)を読み出します。

■書式

[VC] BOOL Dio_ReadCurrentTimer(HANDLE hBoard, ULONG TimerNo, ULONG* pTimerValue, ULONG* pUnit);

[VB] Function Dio_ReadCurrentTimer(ByVal hBoard As Long, ByVal TimerNo As Long, ByRef pTimerValue As Long, ByRef pUnit As Long) As Long

[VB.NET] Function Dio_ReadCurrentTimer(ByVal hBoard As Integer, ByVal TimerNo As Integer, ByRef pTimerValue As Integer, ByRef pUnit As Integer) As Integer

[C#.NET] bool PD5000P. Dio_ReadCurrentTimer(int hBoard, TIMER TimerNo, out uint pTimerValue, out uint pUnit);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

TimerNo タイマ番号を指定します。

PD5306P の場合:
[VC] [VB] タイマ番号として0～3のいずれかを指定します。(4個のタイマを使用できます)
[C#] タイマ番号として0～3のいずれかを指定します。(4個のタイマを使用できます)
TIMER.TIMER0 :0 TIMER.TIMER1 :1 TIMER.TIMER2 :2

PD5006P, PD5106P, PD5206P の場合:
[VC] [VB] タイマ番号として0、または1を指定します。(2個のタイマを使用できます)
[C#] タイマ番号として0、または1を指定します。(2個のタイマを使用できます)
TIMER.TIMER0 :0 TIMER.TIMER1 :1

pTimerValue 読み出すデータを格納する変数のポインタを指定します。
動作タイマ値(現在動作中のタイマの経過時間)を読み出します。
動作タイマ値は 0～32,767 の範囲です。単位は pUnit 変数で読み出します。
タイマが停止しているときは、0がセットされます。

pUnit 読み出すデータを格納する変数のポインタを指定します。
動作タイマ値の単位を読み出します。この単位は、Dio_SetTimer 関数で設定した単位になります。

0 :μsec 単位、 1 :msec 単位

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

本関数で読み出す動作タイマ値は、本関数内でボードの動作タイマ値を読み出した時点の動作タイマ値であり、動作タイマ値を読み出してからアプリケーションに制御を返すまでの時間は含まれません。

■使用例

タイマ0の動作タイマ値を読み出します。TimerValue 変数にタイマ値を, Unit 変数にタイマ値の単位を読み出します。

[VC] Ret = Dio_ReadCurrentTimer(hBoard, 0, &TimerValue, &Unit);
[VB] Ret = Dio_ReadCurrentTimer(hBoard, 0, TimerValue, Unit)
[C#] Ret = PD5000P.Dio_ReadCurrentTimer(hBoard, TIMER.TIMER0, out TimerValue, out Unit);

Dio_StartTimer　タイマ起動

■機能

指定したタイマを起動します。
起動方法は、タイマ単一起動（タイマ 1 回動作）とタイマ連続起動（タイマ繰り返し動作）の2種類です。

タイマ単一起動は、タイマカウンタが0からカウントアップを開始し、カウントがタイマ設定値（Dio_SetTimer 関数で設定した値）になるとタイマ動作を終了（タイムアウト）します。

タイマ連続起動は、タイマカウンタが0からカウントアップを開始し、カウントがタイマ設定値に到達しタイムアウトするとカウンタは0にクリアされ、再度カウントアップするという動作を繰り返します。

■書式

[VC] BOOL Dio_StartTimer(HANDLE hBoard, ULONG TimerNo, ULONG Mode);

[VB] Function Dio_StartTimer(ByVal hBoard As Long, ByVal TimerNo As Long, ByVal Mode As Long) As Long

[VB.NET] Function Dio_StartTimer(ByVal hBoard As Integer, ByVal TimerNo As Integer, ByVal Mode As Integer) As Integer

[C#.NET] bool PD5000P. Dio_StartTimer(int hBoard, TIMER TimerNo, TMODE Mode);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

TimerNo 起動するタイマのタイマ番号を指定します。

PD5306P の場合：
[VC] [VB] タイマ番号として0～3のいずれかを指定します。（4個のタイマを使用できます）
[C#] タイマ番号として0～3のいずれかを指定します。（4個のタイマを使用できます）
TIMER.TIMER0 ：0 TIMER.TIMER1 ：1 TIMER.TIMER2 ：2

PD5006P, PD5106P, PD5206P の場合：
[VC] [VB] タイマ番号として0、または1を指定します。（2個のタイマを使用できます）
[C#] タイマ番号として0、または1を指定します。（2個のタイマを使用できます）
TIMER.TIMER0 ：0 TIMER.TIMER1 ：1

Mode タイマ起動モード（単一起動、または連続起動）を指定します。

[VC] [VB] 0 ：タイマ単一起動、 1 ：タイマ連続起動
[C#] TMODE.RUN_SINGLE ：タイマ単一起動
TMODE.RUN_CYCLE ：タイマ連続起動

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

動作タイマ値(現在動作中のタイマの経過時間)はDio_ReadCurrentTimer関数で読み出すことができます。
タイマ単一起動を停止させるには、Dio_StopTimer関数のタイマ即停止を実行します。
タイマ連続起動を停止させるには、Dio_StopTimer関数のタイマ即停止、またはタイマサイクル停止を実行します。

Dio_SetIntrptMode関数でタイマ割り込みを有効にしておくと、タイマがタイムアウトしたとき(タイマ連続起動ではタイムアウトごと)に割り込みが発生します。

PD5206P の場合:
入力同時ラッチや出力同時セットの動作を、対応したタイマのタイムアウトで実行することができます。これらの動作を行わせる場合には、Dio_SetSmlInTimer や Dio_SetSmlOutTimer 関数でそれらの動作を有効にしてください。

■使用例

タイマ1を単一起動します。

```
[ VC ]    Ret = Dio_StartTimer(hBoard, 1, 0);
[ VB ]    Ret = Dio_StartTimer(hBoard, 1, 0)
[ C# ]    Ret = PD5000P. Dio_StartTimer(hBoard, TIMER.TIMER1, TMODE.RUN_SINGLE);
```

Dio_StopTimer　タイマ停止

■機能

動作中の指定タイマを停止します。
停止方法は、タイマ即停止（タイマを停止させる）とタイマサイクル停止（タイマを周期完了で停止させる）の2種類です。

タイマ即停止は、動作中のタイマが停止します。タイマ即停止で停止させ、再度タイマを起動した場合、タイマカウンタは0から開始します。

タイマサイクル停止は、タイマ連続起動で動作させたタイマをタイマ設定値に到達させタイムアウトで停止させたい場合に実行します。タイマ単一起動の場合にタイマサイクル停止を実行しても意味がありません。

■書式

[VC] BOOL Dio_StopTimer(HANDLE hBoard, ULONG TimerNo, ULONG Mode);

[VB] Function Dio_StopTimer(ByVal hBoard As Long, ByVal TimerNo As Long, ByVal Mode As Long) As Long

[VB.NET] Function Dio_StopTimer(ByVal hBoard As Integer, ByVal TimerNo As Integer, ByVal Mode As Integer) As Integer

[C#.NET] bool PD5000P. Dio_StopTimer(int hBoard, TIMER TimerNo, TMODE Mode);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

TimerNo 停止するタイマのタイマ番号を指定します。

PD5306P の場合：
[VC] [VB] タイマ番号として0～3のいずれかを指定します。（4個のタイマを使用できます）
[C#] タイマ番号として0～3のいずれかを指定します。（4個のタイマを使用できます）
TIMER.TIMER0 ：0 TIMER.TIMER1 ：1 TIMER.TIMER2 ：2

PD5006P, PD5106P, PD5206P の場合：
[VC] [VB] タイマ番号として0、または1を指定します。（2個のタイマを使用できます）
[C#] タイマ番号として0、または1を指定します。（2個のタイマを使用できます）
TIMER.TIMER0 ：0 TIMER.TIMER1 ：1

Mode タイマ停止モード（タイマ即停止、またはタイマサイクル停止）を指定します。

[VC] [VB] 0 ：タイマ即停止、 1 ：タイマサイクル停止
[C#] TMODE.STOP ：タイマ即停止、 TMODE.STOP_CYCLE ：タイマサイクル停止

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

タイマ即停止の場合、Windows の性質上、実際にタイマが停止するまでの時間を保証することはできません。

■注意点

タイマが停止している状態でタイマサイクル停止を実行すると、次にそのタイマのタイマ連続起動を実行した場合、タイマサイクル停止が有効となりタイマは1回で終了します。以降のタイマ連続起動については通常通りの動作になります。

■使用例

タイマ1を即停止します。

```
[VC]    Ret = Dio_StopTimer(hBoard, 1, 0);
[VB]    Ret = Dio_StopTimer(hBoard, 1, 0)
[C#]    Ret = PD5000P.Dio_StopTimer(hBoard, TIMER.TIMER1, TMODE.STOP);
```

Dio_ClearInTransition　入力変化情報クリア

■機能

全信号の入力変化情報をクリアします。

■書式 TMODE.STOP

[VC] BOOL Dio_ClearInTransition(HANDLE hBoard);
[VB] Function Dio_ClearInTransition(ByVal hBoard As Long) As Long
[VB.NET] Function Dio_ClearInTransition(ByVal hBoard As Integer) As Integer
[C#.NET] bool PD5000P. Dio_ClearInTransition(int hBoard);

■パラメータ

hBoard Dio_Open 関数で取得したボードハンドルを指定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

Dio_ReadInTransition関数で入力変化情報を読み出した場合も、全信号の入力変化情報はクリアされます。

■使用例

[VC] Ret = Dio_ClearInTransition(hBoard);
[VB] Ret = Dio_ClearInTransition(hBoard)
[C#] Ret = PD5000P.Dio_ClearInTransition(hBoard);

Dio_ExecSmlInLatch　入力同時ラッチ命令の実行

■機能

すべての入力信号の入力値を同時にラッチします。

本関数実行時に、すべての入力信号について、積分フィルタ通過後の入力論理設定前の値が同時にラッチされます。
ラッチされた値をDio_ReadLatch関数で読み出すと、読み出し時点の入力論理設定に従って変換した入力値が読み出されます。ただし、本関数を実行する前にDio_SetSmlInStbCmd関数で、入力同時ラッチ有効設定(INSTB、命令)を有効にしておく必要があります。

■書式

[VC]　BOOL Dio_ExecSmlInLatch(HANDLE hBoard);
[VB]　Function Dio_ExecSmlInLatch(ByVal hBoard As Long) As Long
[VB.NET]　Function Dio_ExecSmlInLatch(ByVal hBoard As Integer) As Integer
[C#.NET]　bool PD5000P. Dio_ExecSmlInLatch(int hBoard);

■パラメータ

hBoard　Dio_Open 関数で取得したボードハンドルを指定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

PD5006P, PD5306P の場合:
本関数を使用する場合、ボードの JP3 ジャンパーで命令を使用する設定にしてください。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。
Dio_SetIntrptMode 関数でINSTB信号変化時の割り込みを有効にしていると、本関数実行時にINSTB割り込みが発生します。

■注意点

入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力論理設定を変更してはいけません。

PD5306P の場合:
入力同時ラッチを実行してから入力同時ラッチデータを読み出すまでの間は、入力／出力の設定を変更してはいけません。

■使用例

[VC]　Ret = Dio_ExecSmlInLatch(hBoard);
[VB]　Ret = Dio_ExecSmlInLatch(hBoard)
[C#]　Ret = PD5000P.Dio_ExecSmlInLatch(hBoard);

Dio_ExecSmlOut　出力同時セット命令の実行

■機能

設定した出力値を、すべての出力信号に対して同時に出力します。

あらかじめDio_Out_xxByte関数（注1）で設定した出力値を、本関数実行時に、すべての出力信号に対して同時に出力します。ただし、本関数を実行する前にDio_SetSmlOutStbCmd関数で、出力同時セット有効設定（OTSTB、命令）を有効にしておく必要があります。

注1： Dio_Out_xxByte の xx には 1,2,4,8,n のいずれかが入ります。

■書式

[VC]	BOOL Dio_ExecSmlOut(HANDLE hBoard);
[VB]	Function Dio_ExecSmlOut(ByVal hBoard As Long) As Long
[VB.NET]	Function Dio_ExecSmlOut(ByVal hBoard As Integer) As Integer
[C#.NET]	bool PD5000P. Dio_ExecSmlOut(int hBoard);

■パラメータ

hBoard	Dio_Open 関数で取得したボードハンドルを指定します。

■戻り値

関数が成功すると、0以外の値が返ります。
関数が失敗すると、0が返ります。エラーコードを取得する場合は Dio_GetLastError 関数を使用します。

■解説

PD5106P, PD5306P の場合：
本関数を使用する場合、ボードの JP2 ジャンパーで命令を使用する設定にしてください。ジャンパーの設定方法については、ボードの取扱説明書を参照してください。
Dio_SetIntrptMode関数 でOTSTB信号変化時の割り込みを有効にしていると、本関数実行時にOTSTB割り込みが発生します。

■使用例

[VC]	Ret = Dio_ExecSmlOut(hBoard);
[VB]	Ret = Dio_ExecSmlOut(hBoard)
[C#]	Ret = PD5000P.Dio_ExecSmlOut(hBoard);

Dio_GetLastError　エラーコード取得

■機能

呼び出し側のスレッドが持つ最新のエラーコードを取得します。
エラーコードは、スレッドごとに保持されるため、複数のスレッドが互いの最新のエラーコードを上書きすることはありません。

■書式

[VC]	long Dio_GetLastError();
[VB]	Function Dio_GetLastError() As Long
[VB.NET]	Function Dio_GetLastError() As Integer
[C#.NET]	Nmc_Status PD5000P. Dio_GetLastError();

■パラメータ

パラメータはありません。

■戻り値

[VC] [VB]	呼び出し側のスレッドが持つ最新のエラーコードが返ります。 エラーコードの取得に失敗すると、－1が返ります。
[C#]	呼び出し側のスレッドが持つ最新のエラーコードに対応する定義名が返ります。 エラーコードの内容については、「4.4.3 補足説明(3)」を参照してください。

■解説

エラーコードは、各関数が失敗したときに設定されます。新しくエラーが発生すると、古いエラーコードが上書きされます。
Dio_ClearLastError関数でエラーコードをクリアすることができます。エラーコードの初期値は0です。

エラーコードの内容については、「4.4.3 補足説明(3)」を参照してください。

■使用例

Err 変数に最新のエラーコードがセットされます。

[VC]	Err = Dio_GetLastError();
[VB]	Err = Dio_GetLastError()
[C#]	Err = PD5000P.Dio_GetLastError();

Dio_ClearLastError　エラーコードクリア

■機能

呼び出し側スレッドが持つ最新のエラーコードを0クリアします。

■書式

[VC]　　void Dio_ClearLastError();
[VB]　　Sub Dio_ClearLastError()
[VB.NET]　Sub Dio_ClearLastError()
[C#.NET]　void PD5000P.Dio_ClearLastError();

■パラメータ

パラメータはありません。

■戻り値

戻り値はありません。

■使用例

[VC]　　Dio_ClearLastError();
[VB]　　Dio_ClearLastError
[C#]　　PD5000P.Dio_ClearLastError();

4.4.3 補足説明

（1） 入力値／出力値

①入力値

入力値は、入力信号が積分フィルタを通過し、入力論理設定に従って判断された値です。
入力値は、0 または 1 で表現されます。
入力信号 Hi を入力値1とするか、入力信号 Low を入力値1とするかは、入力論理設定で設定します。

②出力値

各信号の対応するビットに出力値 0 をセットすると出力信号 Hi に、出力値 1 をセットすると出力信号 Low になります。

③出力値の読み出し

Dio_In_XXX 関数(注1)で出力値を読み出した場合、現在の出力信号の状態が読み出されます。また、出力同時セット有効設定を有効にした場合、Dio_Out_xxByte 関数(注2)で出力値を書いても直ちに出力信号には反映されません。この時点で出力値を読み出した場合、現在の出力信号の状態が読み出されますのでご注意ください。

注1： Dio_In_XXX 関数の XXX には、1Bit, 1Byte, 2Byte, 4Byte, 8Byte, nBit, nByte のいずれかが入ります。

注2： Dio_Out_xxByte 関数の xx には 1,2,4,8,n のいずれかが入ります。

（2） 信号名と信号番号の対応表

ボードの信号名と関数で指定する信号番号の対応は、下表の通りです。

ポート番号	信号名	信号番号（16 進数表示）	ポート番号	信号名	信号番号（16 進数表示）
ポート0	SN00	00	ポート4	SN40	20
	SN01	01		SN41	21
	SN02	02		SN42	22
	SN03	03		SN43	23
	SN04	04		SN44	24
	SN05	05		SN45	25
	SN06	06		SN46	26
	SN07	07		SN47	27
ポート1	SN10	08	ポート5	SN50	28
	SN11	09		SN51	29
	SN12	0A		SN52	2A
	SN13	0B		SN53	2B
	SN14	0C		SN54	2C
	SN15	0D		SN55	2D
	SN16	0E		SN56	2E
	SN17	0F		SN57	2F
ポート2	SN20	10	ポート6	SN60	30
	SN21	11		SN61	31
	SN22	12		SN62	32
	SN23	13		SN63	33
	SN24	14		SN64	34
	SN25	15		SN65	35
	SN26	16		SN66	36
	SN27	17		SN67	37
ポート3	SN30	18	ポート7	SN70	38
	SN31	19		SN71	39
	SN32	1A		SN72	3A
	SN33	1B		SN73	3B
	SN34	1C		SN74	3C
	SN35	1D		SN75	3D
	SN36	1E		SN76	3E
	SN37	1F		SN77	3F

（3） エラーコード

エラーコードは、下表の通りです。各定義は提供ファイルにて定義しています。

定義名	エラーコード（16 進数）	エラー内容
ERR_NO_SUPPORT	100	指定ボードではサポートしていない関数をコールしました。
ERR_PORT_NO	101	指定したポート番号が間違っています。
EER_INTRPT_OPEN	102	指定ボードは既に別アプリが割込みを使用したオープンを行っています。 1ボードに対して割り込みを使用したオープンができるのは、1つのアプリケーションのみです。（注1）
ERR_ALREADY_OPEN	200	指定したボードは既にオープン中です。
ERR_OPEN_BOARD	201	ボードをオープンできません。（指定したデバイスID、スイッチ番号が間違っている場合など）
ERR_ACCESS_BOARD	202	指定したボードにアクセスできません。（指定したボードハンドルが間違っている場合や オープンしていないボードにアクセスした場合など）
ERR_ALLOC_MEM	203	メモリを確保できません。
ERR_INTRPT_INVALID	204	Dio_Open 時に割り込み処理を使用しない設定にしている為、指定関数を実行できません。
ERR_END_THREAD	205	スレッドを終了できませんでした。（割り込み用ユーザー関数実行中に Dio_Close した場合などに発生します。）（注2）
ERR_PARAM	300	引数の値が間違っています。

その他のエラーコードについては公開しておりません。

注1：
割り込みを使用したオープンを行ったとき、既に別アプリケーションがそのボードに対して割り込み使用オープンを行っていた場合、このエラーが発生します。
また、前回割り込み使用オープン時に正しくクローズしなかった場合は、次回の割り込み使用オープンでこのエラーが発生する場合がありますので、必ずクローズ処理を行ってください。クローズしないためにこのエラーが発生した場合は、パソコンを再起動してください。

注2：
アプリケーションは、割り込みユーザー関数（Dio_SetEventFunc 関数で指定した関数）実行中にクローズ処理（Dio_Close または Dio_CloseAll 関数）を実行しないでください。クローズ処理を行う場合は、必ず、割り込みユーザー関数が終了している状態で行ってください。

（4） 入力／出力ポート

各ボードのポートの入出力は下記の通りです。

ボード型式	ポート0～3	ポート4～7
PD5006P	入力ポート	
PD5106P	出力ポート	
PD5206P	入力ポート	出力ポート
PD5306P	入力／出力ポート（双方向）	

（5） 定義内容

各定義は、それぞれ下記のファイルで行っています。

VC++	:	Nv_PiDio_DLL.h
VB6.0	:	Nv_PiDio_DLL.bas
VB. NET2003、VB2005、VB2008	:	Nv_PiDio_DLL.vb
C#. NET	:	Pd5000pWrap.dll

VCの定義内容を以下に示します。

①デバイスID

```
#define ID_PD5006P      0xA0C6      // PD5006P  入力64点
#define ID_PD5106P      0xA0C7      // PD5106P  出力64点
#define ID_PD5206P      0xA0C8      // PD5206P  入力32点／出力32点
#define ID_PD5306P      0xA0C9      // PD5306P  双方向64点
```

②メッセージ

```
#define WM_INTRPT       (WM_USER + 0x100)       // 割り込み通知メッセージ
```

③エラーコード

「4.4.3 補足説明(3)エラーコード」の項目を参照してください。

C#. NETの定義内容を以下に示します。

1） 配列変数のポインタ使用した関数のパラメータ

①PD5000P._Dio_In_nBit 関数
関数のパラメータに配列変数のポインタを使用する為
unsafe コンテキストの fixed ステートメントを使用して配列変数を固定

```
public static bool _Dio_In_nBit(int hBoard, uint Cnt, ref SIGNAL[] SignalNoArray, ref uint[] RdDataArray)
{
    fixed (SIGNAL* _SignalNoArray = SignalNoArray)
    {
        fixed (uint* _RdDataArray = RdDataArray)
        {
            return(PD5000P.Dio_In_nBit(hBoard, Cnt, _SignalNoArray, _RdDataArray));
        }
    }
}
```

②PD5000P._Dio_In_nByte 関数
関数のパラメータに配列変数のポインタを使用する為
unsafe コンテキストの fixed ステートメントを使用して配列変数を固定

```
public static bool _Dio_In_nByte(int hBoard, uint Cnt, ref PORT[] PortNoArray, ref uint[] RdDataArray)
{
	fixed (PORT* _PortNoArray = PortNoArray)
	{
		fixed (uint* _RdDataArray = RdDataArray)
		{
			return(PD5000P.Dio_In_nByte(hBoard, Cnt, _PortNoArray, _RdDataArray));
		}
	}
}
```

③PD5000P. _Dio_Out_nBit 関数
関数のパラメータに配列変数のポインタを使用する為
unsafe コンテキストの fixed ステートメントを使用して配列変数を固定

```
public static bool _Dio_Out_nBit(int hBoard, uint Cnt, ref SIGNAL[] SignalNoArray, ref uint[] WtDataArray)
{
	fixed (SIGNAL* _SignalNoArray = SignalNoArray)
	{
		fixed (uint* _WtDataArray = WtDataArray)
		{
			return(PD5000P.Dio_Out_nBit(hBoard, Cnt, _SignalNoArray, _WtDataArray));
		}
	}
}
```

④PD5000P._Dio_Out_nByte 関数
関数のパラメータに配列変数のポインタを使用する為
unsafe コンテキストの fixed ステートメントを使用して配列変数を固定

```
public static bool _Dio_Out_nByte(int hBoard, uint Cnt, ref PORT[] PortNoArray, ref uint[] WtDataArray)
{
	fixed (PORT* _PortNoArray = PortNoArray)
	{
		fixed (uint* _WtDataArray = WtDataArray)
		{
			return(PD5000P.Dio_Out_nByte(hBoard, Cnt, _PortNoArray, _WtDataArray));
		}
	}
}
```

4.4.4 プログラミング上の注意点

（1） 割り込みについて

割り込みが短時間に連続して複数回発生した場合、アプリ(アプリケーション)への割り込み通知はまとめて 1 回になる場合があります。また、2 回の割り込みが短時間に発生した場合で、1 回目の割り込み通知でアプリが割り込み要因を読んでいる間に 2 回目の割り込みが発生した場合は、1 回目のアプリの割り込み要因読み出しで 2 回目の割り込み要因も読み出される可能性があります。この場合、2 回目の割り込み通知でアプリが割り込み要因を読み出しても割り込み要因は既に 1 回目のときに読み出されていてクリアされている場合があります。

割り込み通知を使用する場合、Windows の性質上、割り込み発生からアプリケーションが通知を受けるまでの時間を保証することはできません。(割り込み通知方法は関数呼び出しとメッセージ送信の2種類です。)

タイマ割り込み有効のタイマを連続起動する場合、極端に小さいタイマ値にすると、短い時間に割り込みが多発し、アプリケーションの動作が遅くなる場合があるので注意してください。

（2） 出力同時セット関連

出力同時セット有効状態のときに出力値を書き、OTSTB 信号、命令、またはタイマで出力する前に出力同時セットを無効状態に変更した場合、このときの出力は不定になりますので、このような操作は行わないでください。
また、出力同時セット有効設定を有効にしている場合、Dio_Out_1Bit と Dio_Out_nBit 関数は使用できません。

（3） パソコンのスタンバイ、休止動作

本ドライバは、パソコンのスタンバイ動作、または休止動作を行った後のドライバの動作を保証していません。
パソコンのスタンバイ動作、または休止動作を行った後に、ボードにアクセスしたい場合は、一度パソコンを再起動させてから行ってください。

（4） その他

その他の注意点については、各章の説明を参照してください。

5. 評価ツール

この章では、PD5000P シリーズのボードの評価ツールについて説明します。

本評価ツールプログラムは、弊社ホームページよりダウンロードすることができます。
実行プログラムは PD_Tool.exe です。評価ツールを実行する前に、PiDio デバイスドライバをインストールして下さい。

5.1 概要

評価ツールを起動すると、ボード選択画面が表示されます。その画面で、ボード名称とスイッチ番号(ボードのロータリスイッチ番号(0～F))を選択し、OKボタンを押すと、メイン画面が表示されます。

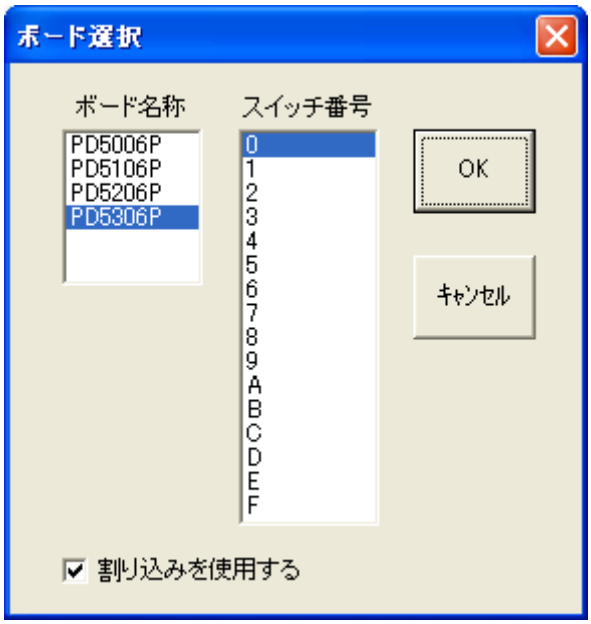
メイン画面では、入力値・出力値の読み出し、出力動作、タイマ起動・停止、入力変化情報読み出し、入力同時ラッチデータ読み出し、命令実行などを行います。また、モード・パラメータ設定のボタンを押すと、各画面が開き、設定を行う事ができます。
また、評価ツール起動時に、現在設定されているモード・パラメータ設定値や出力値をボードから読み出し、表示します。

■ 全画面共通の表示内容について

Port0～7 はポート番号、SNx0～7 は信号番号です。SNx0～7 の x にはポート番号が入ります。
ボードの型式によっては使用できない機能があり、その場合はその機能のボタンを無効表示にしています。
画面の右上の×ボタンを押すと、画面を閉じます。

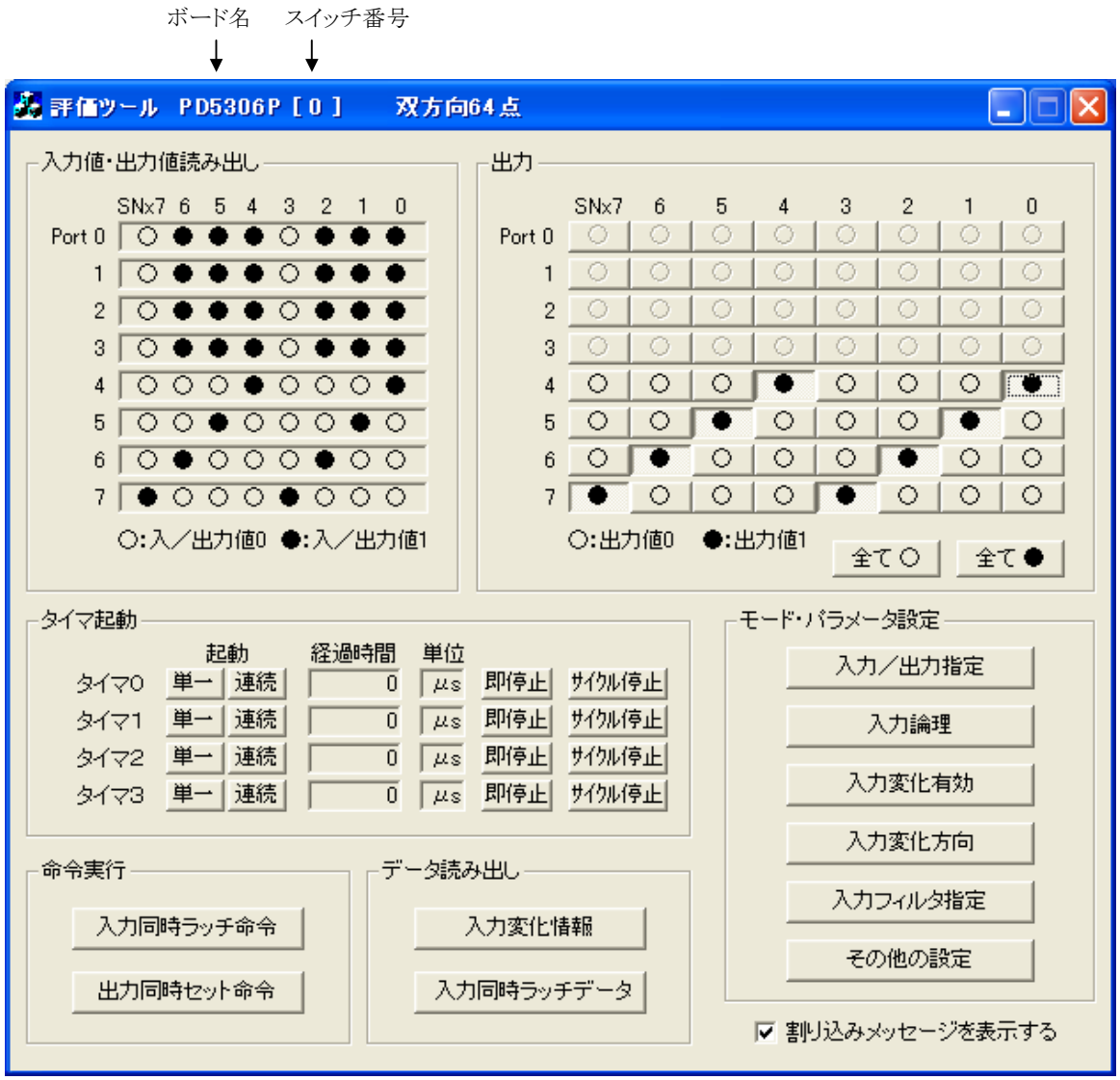
5.2 ボード選択画面

ボード名称とスイッチ番号(ボードのロータリスイッチ番号(0～F))を選択し、OKボタンを押すと、メイン画面が表示されます。
割り込みを使用する場合は、「割り込みを使用する」チェックボックスをチェックしたままにします。
割り込みを使用しない場合は、「割り込みを使用する」チェックボックスのチェックを外します。
キャンセルボタンを押すと、評価ツールは終了します。



5.3 メイン画面

入力値・出力値の読み出し、出力動作、タイマ起動・停止、入力変化情報読み出し、入力同時ラッチデータ読み出し、命令実行などを行います。また、モード・パラメータ設定のボタンを押すと、各画面が開き、設定を行う事ができます。
画面の右上の×ボタンを押すと、評価ツールを終了します。



■ 入力値・出力値の読み出し

全信号の入力値・出力値を 100ms 間隔で読み出し、画面に表示しています。
○は入／出力値0、●は入／出力値1です。入力値 0,1 の定義については、入力論理設定画面の説明を参照してください。
入力信号の場合は入力値を、出力信号の場合は出力値を読み出します。

■ 出力

ボタンを押して表示を○にした場合、対応する信号に対して出力値0を書き込みます。
ボタンを押して表示を●にした場合、対応する信号に対して出力値1を書き込みます。
出力値 0 をセットすると出力信号 Hi に、出力値 1 をセットすると出力信号 Low になります。
「全て○」ボタンを押すとすべての表示が○に、「全て●」ボタンを押すとすべての表示が●になります。

出力同時セット無効の場合は、ボタンを押すと直ちに出力されます。出力同時セット有効の場合は、ボタンを押しても直ちに出力されません。その場合、出力同時セットの動作を行ったときに出力されます。
PD5306P の場合は、入力／出力指定画面で出力信号にした信号のみ出力することができます。

■ モード・パラメータ設定

モード・パラメータ設定を行います。入力論理や入力変化有効ボタンなどを押すと、各設定画面が開き、それぞれの設定を行うことができます。「その他の設定画面」の場合は、OKボタンか適用ボタンを押した時点で値がボードに設定されます。それ以外の画面では、各ボタンやチェックボックスなどを押した時点で値がボードに設定されます。

■ タイマ起動

各タイマ(タイマ0～3)のボタンを押したときに、タイマ起動(単一、連続)、タイマ停止(即停止、サイクル停止)を行います。また、タイマ実行中の動作タイマ値を 100ms 間隔で読み出し、画面の経過時間と単位の枠に表示します。

■ データ読み出し

入力変化情報ボタン ・・・・・・・・・ 入力変化情報読み出し画面を開きます。
入力同時ラッチデータボタン ・・・ 入力同時ラッチデータ読み出し画面を開きます。

■ 命令実行

入力同時ラッチ命令ボタン ・・・ 入力同時ラッチ命令を実行します。「その他の設定画面」で、命令による入力同時ラッチを有効にしてから実行してください。PD5206P 以外の場合は、ボードの JP3 ジャンパーで命令を使用する設定にしてください。

出力同時セット命令ボタン ・・・ 出力同時セット命令を実行します。「その他の設定画面」で、命令による出力同時セットを有効にし、あらかじめメイン画面の出力ボタンで出力値を書いてから実行してください。PD5206P 以外の場合は、ボードの JP2 ジャンパーで命令を使用する設定にしてください。

■ 割り込みメッセージを表示するチェックボックス

「割り込みメッセージを表示する」チェックボックスをチェックすると、割り込みが発生したときに割り込み画面を表示し、割り込みが発生したことを知らせます。チェックを外すと、割り込みが発生しても割り込み画面を表示しません。また、ボード選択画面で「割り込みを使用する」チェックボックスのチェックを外した場合、本チェックボックスは使用できません。

5.4 入力／出力指定画面（PD5306P のみ）

すべての信号（全64点）について、入力信号として使用するか、出力信号として使用するかを1点ごとに設定します。
ボタンを押して表示を○にした場合は入力信号です。ボタンを押して表示を●にした場合は出力信号です。
「全て入力信号」ボタンを押すとすべて入力信号に、「全て出力信号」ボタンを押すとすべて出力信号になります。

注意:本画面で出力信号に変更する時に、該当信号の入力論理と入力変化有効が●になっている場合は、本プログラムでそれらを○の状態に変更しています。また、入力信号に変更する時に、該当信号の出力値が●になっている場合は、本プログラムで出力値を○の状態に変更しています。

5.5 入力論理設定画面

すべての入力信号について、論理レベルを1点ごとに設定します。
ボタンを押して表示を○にした場合は、入力信号 Hi レベルが入力値1です。
ボタンを押して表示を●にした場合は、入力信号 Low レベルが入力値1です。
「全て○」ボタンを押すとすべての表示が○に、「全て●」ボタンを押すとすべての表示が●になります。
PD5306P の場合は、入力／出力指定画面で入力信号にした信号のみ設定することができます。

5.6 入力変化有効設定画面

すべての入力信号について、入力変化（入力信号の変化を捉える機能）有効／無効の設定を1点ごとに行います。
ボタンを押して表示を○にした場合は入力変化無効、ボタンを押して表示を●にした場合は入力変化有効です。
「全て○」ボタンを押すとすべての表示が○に、「全て●」ボタンを押すとすべての表示が●になります。
PD5306P の場合は、入力／出力指定画面で入力信号にした信号のみ設定することができます。

入力変化有効に設定した入力信号について、設定した方向に入力信号が変化した場合、変化を捉えます。方向は、入力変化方向設定で設定します。

5.7 入力変化方向設定画面

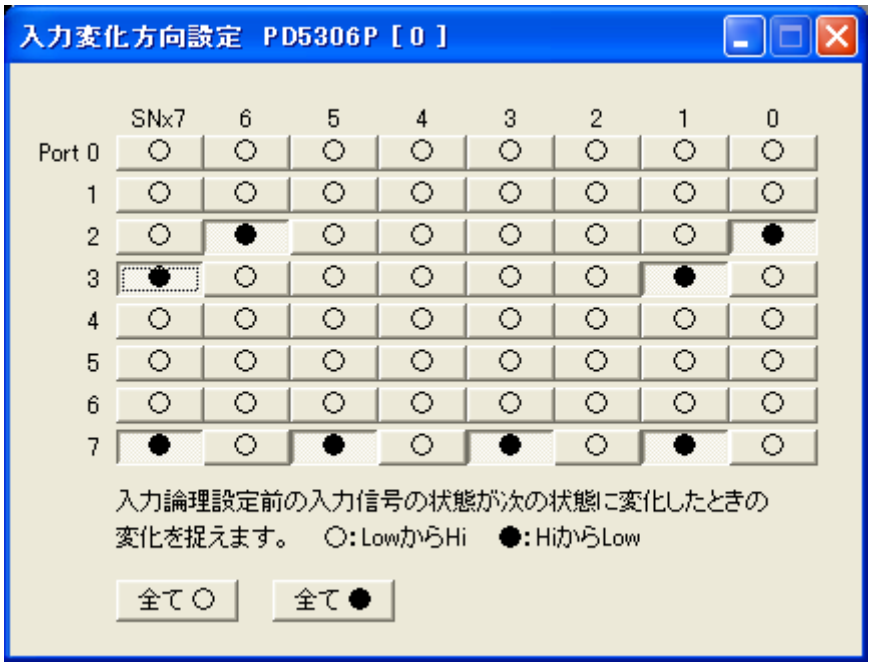
すべての入力信号について、入力変化方向(入力信号 Low から Hi の変化を捉える、または入力信号 Hi から Low の変化を捉える)の設定を1点ごとに行います。

設定する変化方向は、入力論理設定前の入力信号の状態について設定します。

ボタンを押して表示を○にした場合は Low から Hi、ボタンを押して表示を●にした場合は Hi から Low です。

「全て○」ボタンを押すとすべての表示が○に、「全て●」ボタンを押すとすべての表示が●になります。

入力変化有効設定で入力変化有効にした入力信号について、入力変化方向設定で設定した方向に入力信号が変化した場合、変化を捉えます。



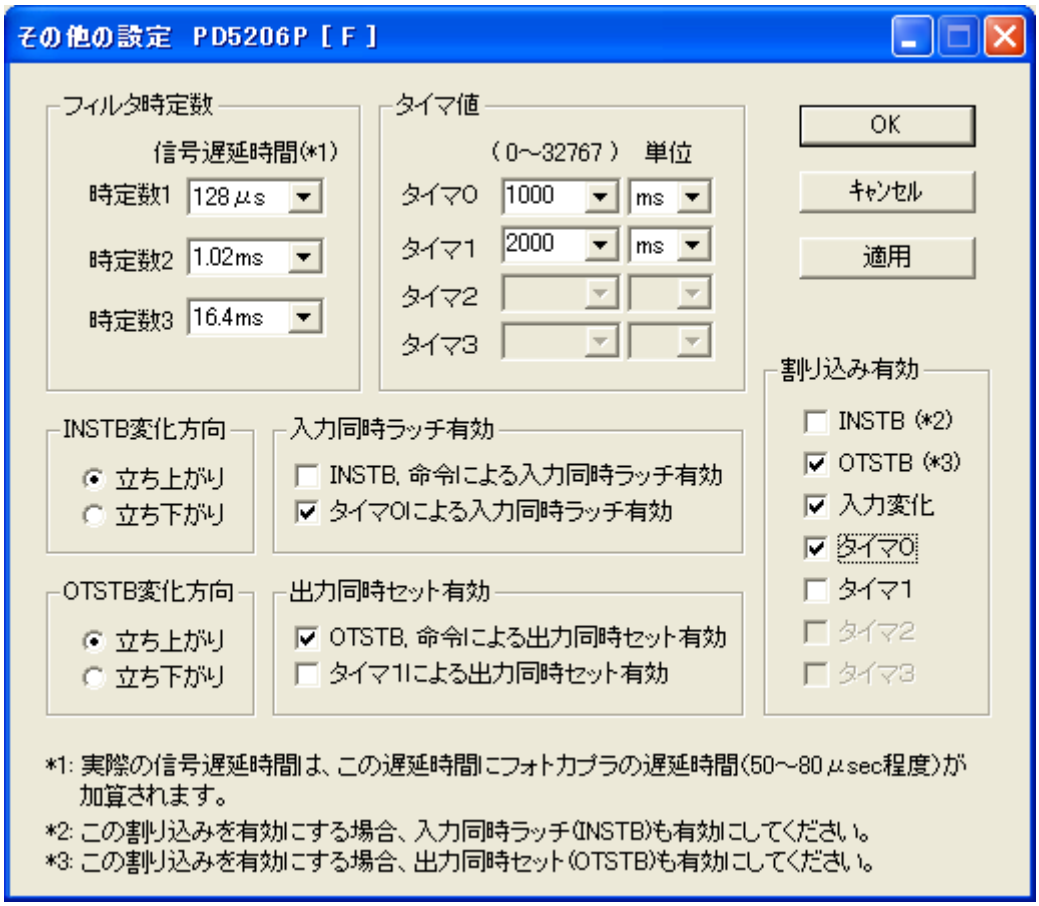
5.8 入力フィルタ指定画面

すべての入力信号に対して、信号4点ごとに積分フィルタの時定数を指定します。
フィルタは、3種類のフィルタ時定数(1，2，3)とフィルタなしの中から選択でき、4点(各ポートの上位4点／下位4点)ごとに設定します。(フィルタ時定数は、「その他の設定画面」で、16個の信号遅延時間から3個を選択しフィルタ時定数1，2，3に設定します。)

オプションボタンをクリックすると、指定した4点に下記のフィルタが設定されます。
なし：フィルタなし、 1：時定数1、 2：時定数2、 3：時定数3

5.9 その他の設定画面

その他のモード・パラメータ設定を行います。各設定を選択後、OKボタンまたは適用ボタンを押すと設定内容をボードに設定します。キャンセルボタンを押した場合は設定しません。
OKボタンは、設定後画面を閉じます。適用ボタンは、設定後画面を閉じません。



■ フィルタ時定数設定

フィルタ時定数1, 2, 3の信号遅延時間を設定します。本ボードは3種類のフィルタ時定数(1, 2, 3)を持っています。各々のフィルタ時定数は、16個の信号遅延時間の中から1つを選択することができます。コンボボックスから信号遅延時間を選択してください。

■ タイマ値設定

タイマの値を 0～32,767 の範囲で、μ sec 単位、または msec 単位で設定します。
PD5306P は4個のタイマを、それ以外のボードは2個のタイマを使用でき、それぞれのタイマにタイマ値を設定します。
タイマ値は、コンボボックスから選択するか、または値を直接入力することができます。単位は、コンボボックスから選択してください。μ sec 単位は μ s、msec 単位は ms です。

■ INSTB 変化方向設定

INSTB(外部ストローブ信号)の変化方向として、信号の立ち上がりを使用するか立ち下がりを使用するかを設定します。
INSTB 信号は入力同時ラッチと割り込みで使用します。オプションボタンで立ち上がり、または立ち下がりを選択してください。

■ OTSTB 変化方向設定

OTSTB(外部ストローブ信号)の変化方向として、信号の立ち上がりを使用するか立ち下がりを使用するかを設定します。
OTSTB 信号は出力同時セットと割り込みで使用します。オプションボタンで立ち上がり、または立ち下がりを選択してください。

■ 入力同時ラッチ有効設定

①INSTB、命令による入力同時ラッチ

INSTB、命令による入力同時ラッチ有効チェックボックスは、INSTB 信号、または命令による入力同時ラッチ機能の有効／無効を設定します。チェックすると有効、チェックを外すと無効になります。
この機能を有効にすると、INSTB 信号の変化時（立ち上がりまたは立ち下がりを設定）、または入力同時ラッチ命令実行時に、全入力が内部に取り込まれます。

②タイマ0による入力同時ラッチ（PD5206P のみ）

タイマ0による入力同時ラッチ有効チェックボックスは、タイマ0による入力同時ラッチ機能の有効／無効を設定します。チェックすると有効、チェックを外すと無効になります。
この機能を有効にすると、タイマ0を起動させ、タイマがタイムアウトしたとき（タイマ連続起動ではタイムアウトごと）に、全入力が内部に取り込まれます。タイマはタイマ0を使用してください。

■ 出力同時セット有効設定

①OTSTB、命令による出力同時セット

OTSTB、命令による出力同時セット有効チェックボックスは、OTSTB 信号、または命令による出力同時セット機能の有効／無効を設定します。チェックすると有効、チェックを外すと無効になります。
この機能を有効にすると、メイン画面の出力ボタンを押しても、出力信号は変化しません。OTSTB 信号の変化時（立ち上がりまたは立ち下がりを設定）、または出力同時セット命令実行時に、あらかじめ出力ボタンで設定した出力値が出力信号に反映されます。

②タイマ1による出力同時セット（PD5206P のみ）

タイマ1による出力同時セット有効チェックボックスは、タイマ1による出力同時セット機能の有効／無効を設定します。
チェックすると有効、チェックを外すと無効になります。
この機能を有効にすると、メイン画面の出力ボタンを押しても、出力信号は変化しません。タイマ1を起動させ、タイマがタイムアウトしたとき（タイマ連続起動ではタイムアウトごと）に、あらかじめ出力ボタンで設定した出力値が出力信号に反映されます。
タイマはタイマ1を使用してください。

■ 割り込み設定

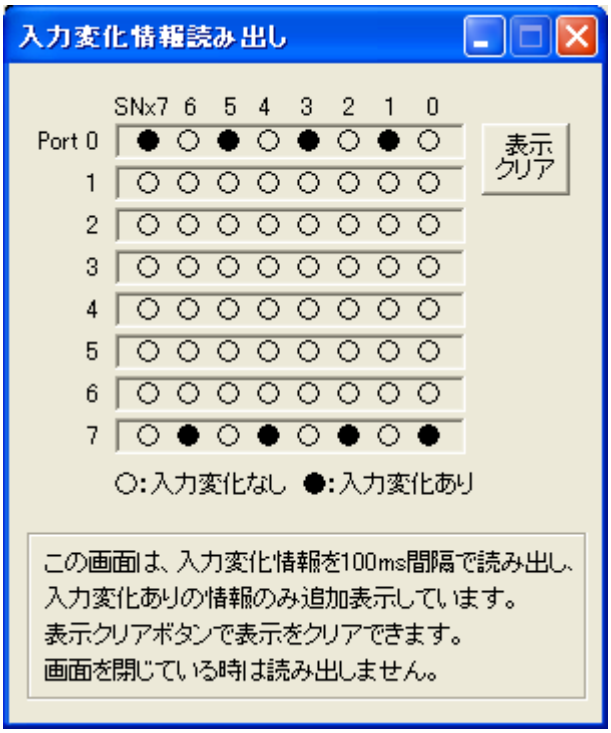
各割り込みの有効／無効を設定します。各チェックボックスをチェックすると有効、チェックを外すと無効になります。
有効にすると、指定した条件のとき、ボード内で割り込みが発生します。割り込みが発生したときに割り込みメッセージ画面を表示したいときは、メイン画面の「割り込みメッセージを表示する」チェックボックスをチェックしてください。

項目	説明
INSTB	INSTB 信号が指定した変化方向（立ち上がり／立ち下がり）に変化したときに、割り込みが発生します。（注1）
OTSTB	OTSTB 信号が指定した変化方向（立ち上がり／立ち下がり）に変化したときに、割り込みが発生します。（注2）
入力変化	入力変化有効設定で有効にした入力信号のうち、いずれかの信号が変化したときに割り込みが発生します。入力変化有効にした信号が変化（変化方向は入力変化方向設定で設定）すると、割り込みが発生し、入力変化情報がセットされます。割り込み発生時の入力変化情報は、入力変化情報画面で確認してください。
タイマ0	タイマ0を起動させ、タイマがタイムアウトすると割り込みが発生します。 タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。
タイマ1	タイマ1を起動させ、タイマがタイムアウトすると割り込みが発生します。 タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。
タイマ2	タイマ2を起動させ、タイマがタイムアウトすると割り込みが発生します。 タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。
タイマ3	タイマ3を起動させ、タイマがタイムアウトすると割り込みが発生します。 タイマを連続起動させた場合には、タイムアウトごとに割り込みが発生します。

注1： 入力同時ラッチ有効設定（INSTB、命令）を有効にしないと、この割り込みは有効になりません。
注2： 出力同時セット有効設定（OTSTB、命令）を有効にしないと、この割り込みは有効になりません。

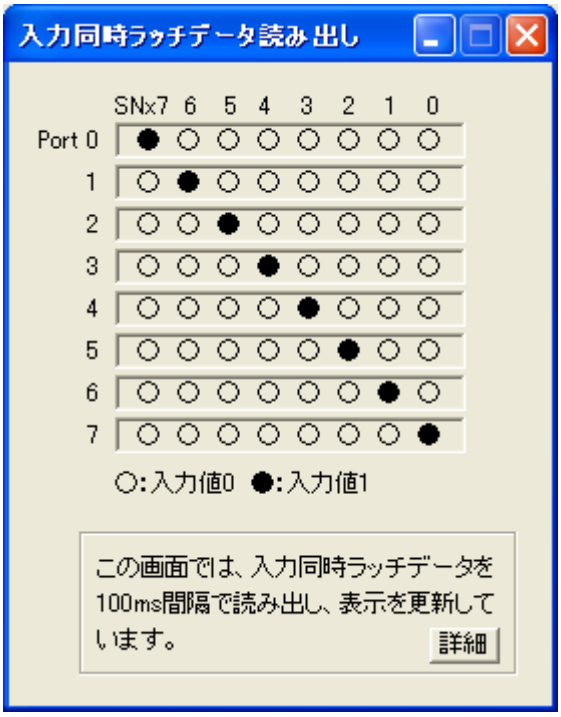
5.10 入力変化情報読み出し画面

入力変化情報を 100ms 間隔で読み出し、入力変化ありの情報のみ保持し続け、画面に追加表示しています。
○は入力変化なし、●は入力変化ありです。
ボードから入力変化情報を読み出すと、ボード内の入力変化情報はクリアされますが、本プログラムではプログラム内に変化の情報を保持し、表示クリアボタンでクリアするまでは、変化ありの情報を表示し続けます。本画面を閉じているときは、情報を読み出しません。



5.11 入力同時ラッチデータ読み出し画面

入力同時ラッチデータを 100ms 間隔で読み出し、画面に表示しています。○は入力値0、●は入力値1です。
入力値 0,1 の定義については、入力論理設定画面の説明を参照してください。



注意: 入力同時ラッチデータは、入力同時ラッチした時点では、入力論理前の値がラッチされ、値を読み出すとき、読み出し時点の入力論理に従い読み出されます。ラッチ時点の入力論理ではありません。このため、ラッチしてからラッチデータを読み出すまでの間は入力論理を変更してはいけません。入力論理を変更した場合は本画面の表示も変更されます。
また、PD5306P の場合、本関数で値を読み出すときに入力信号に設定されている信号のラッチデータのみを読み出すので、入力／出力指定を変更した場合は本画面の表示が変更される場合もあります。

5.12 割り込みメッセージ画面

割り込みが発生したときに、本画面を表示し、割り込みが発生したことを知らせます。
割り込み発生要因(入力変化以外)を読み出し、画面に表示します。○が発生、×が未発生です。
割り込み発生要因の詳細については、「その他の設定画面」の割り込み設定の説明を参照してください。入力変化情報の割り込みが発生したときも本画面を表示し、割り込みが発生したことを知らせますが、本画面には入力変化情報を表示しません。
割り込み発生時の入力変化情報については、入力変化情報読み出し画面で確認してください。

割り込みが発生したときに本画面を表示したいときは、メイン画面の「割り込みメッセージを表示する」チェックボックスをチェックしてください。チェックを外すと本画面は表示されません。

注: 本画面表示中に発生した割り込みは、本画面を閉じた後に読み出し処理を行います。