REX-USB10

32Bit DIO Data Acquisition System

ユーザーズマニュアル

2002 年 7 月

第 1.1 版

RATOC Systems, Inc.

ラトックシステム株式会社

目　次

I&L サポートセンターへのお問い合わせ

技術的なご質問やご相談の窓口を用意していますのでご利用ください。

ラトックシステム株式会社

I&L サポートセンター

〒556-0012

大阪市浪速区敷津東 1-6-14　朝日なんばビル

TEL:06-6633-6741

FAX:06-6633-3553

<サポート受付時間>

月曜－金曜（祝祭日は除く） 10:00 - 13:00

14:00 - 17:00

また、インターネットのホームページでも受け付けています。

http://www.ratocsystems.com

【ご注意】

☑本書の内容については、将来予告なしに変更することがあります。

☑本書の内容につきましては万全を期して作成しましたが、万一ご不審な点や誤りなどお気づきになられましたら I&L サポートセンターまでご連絡願います。

☑本製品および本製品添付のマニュアルに記載されている会社名および製品名は、各社の商品または登録商標です。

☑本製品の運用を理由とする損失、免失利益などの請求につきましては、いかなる責任も負いかねますので予めご了承願います。

第１章 ご使用になる前に

この章では、本製品の特徴並びに製品仕様について説明します。

1-1. はじめに

このたびは、REX-USB10 32Bit DIO Data Acquisition System をご購入いただきましてありがとうございます。REX-USB10 は USB Specification1.1 の 12Mbps フルスピード仕様準拠の USB データアクイジョンユニットです。8 ポート単位で入出力方向設定可能な TTL レベルのデジタル入出力ポートを 32 ポート用意しています。主に下記のような用途に使用することができます。

■ 32Bit デジタル入出力機能

外部スイッチ等のオンオフ状態の検出､外部機器のオンオフ制御の用途として使用することができます（注.外部機器と接続する場合はアイソレーション等が必要となる場合があります)。

■ 外部イベント監視機能

外部機器のオンオフ状態の変化をイベントとして検出し警報等の出力を行う用途として使用することができます。

■ データ転送機能

8/16/32 ビットいづれかのモードで外部機器とハンドシェークを行い、データの送受信を行う用途として使用することができます。

1-2. 梱包内容のご確認

内 容	個 数	備 考
REX-USB10 32Bit DIO ユニット本体	1	
外部接続用コネクタセット	1	富士通 FCN-361J048-AU ハンダ付けタイプコネクタ FCN-360C048 コネクタカバー
AC アダプタ	1	定格 DC5V,2.3A
USB ケーブル	1	50cm 長
サポートソフトウェアディスク	1	1.44MB 3.5”フロッピーディスク
製品マニュアル	1	
ハードウェア保証書	1	
ご愛用者登録はがき	1	

万一、不足品等ございましたら I&L サポートセンターまでご連絡願います。

1-3. 製品仕様

1.3.1 ハードウェア仕様

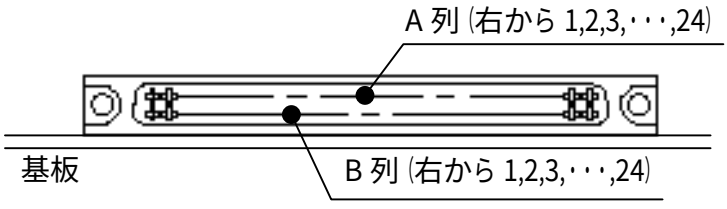
項 目	仕 様	備 考
デジタル入出力数	32 ポート	
方向設定単位	8 ポート単位	
デジタル入力形式	非絶縁型 TTL レベル	74HCT245 相当品
デジタル出力形式	非絶縁型 TTL レベル LSTTL15 個駆動可能	74HCT245 相当品
ストローブ出力端子	TTL レベル 1 ポート	
割り込み入力端子	TTL レベル 1 ポート	ハンドシェーク兼用ポート
内蔵 FIFO	100K バイト	
外部電源供給	5V/2A	セルフパワード動作時
電源供給方式	セルフパワード・バスパワード 自動判別	AC アダプタ接続時セルフパワードモードで動作
消費電流	typ.200mA	全ポート入力設定時
USB 規格	USB Specification Revision1.1 準拠	
USB 接続コネクタ	Series B コネクタ	
デジタル入出力コネクタ	FCN-365P048-AU	富士通製
外形寸法	W155×L103×H26.4	
重量	約 320g	
動作温湿度	0～55℃,10～80%	但し、結露しないこと

1.3.2 ソフトウェア環境

項 目	仕 様	備 考
対応 OS	Windows 98 Windows 98 Second Edition Windows 2000 Windows Millennium Edition Windows XP	
製品付属ドライバ	OHCI 準拠 WDM ドライバ	
製品付属ライブラリ	32 ビット DLL ライブラリ	
対応開発言語	Microsoft Visual C/C++ Microsoft Visual BASIC Borland C/C++	

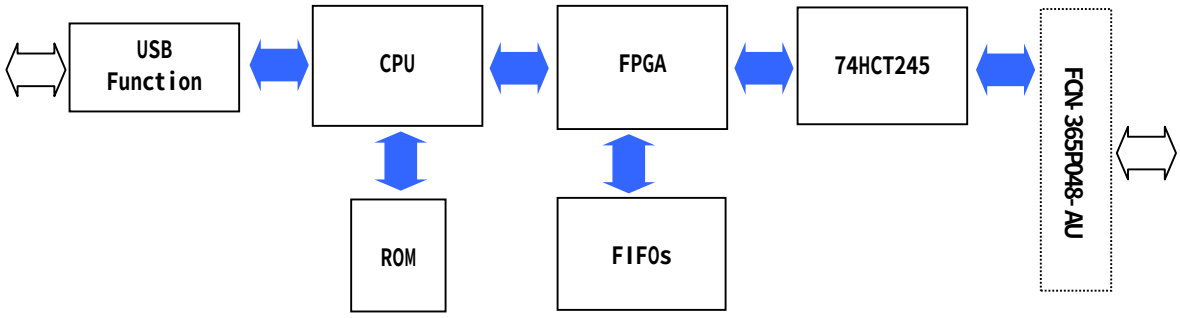
1.3.3 コネクタピンアサイン

A　列		ピン番号	B　列	
説　明	信号名		信号名	説　明
外部出力電源	+5V	1	+5V	外部出力電源
	P0	2	P1	
	P2	3	P3	
	P4	4	P5	
	P6	5	P7	
	P8	6	P9	
	P10	7	P11	
	P12	8	P13	
	P14	9	P15	
信号グランド	GND	10	GND	信号グランド
	P16	11	P17	
	P18	12	P19	
	P20	13	P21	
	P22	14	P23	
	P24	15	P25	
	P26	16	P27	
	P28	17	P29	
	P30	18	P31	
割り込み入力	INT	19	STB	ストローブ出力
-	NC	20	NC	-
-	NC	21	NC	-
-	NC	22	NC	-
-	NC	23	NC	-
信号グランド	GND	24	GND	信号グランド

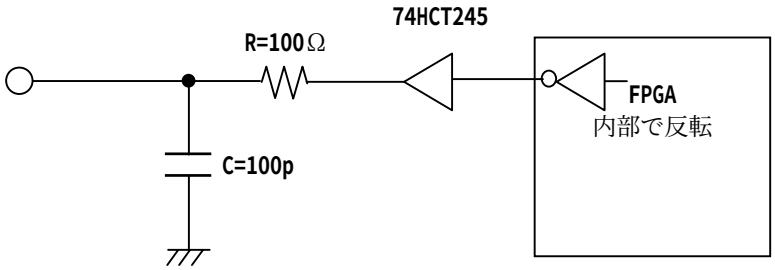


コネクタ挿入側から見た図

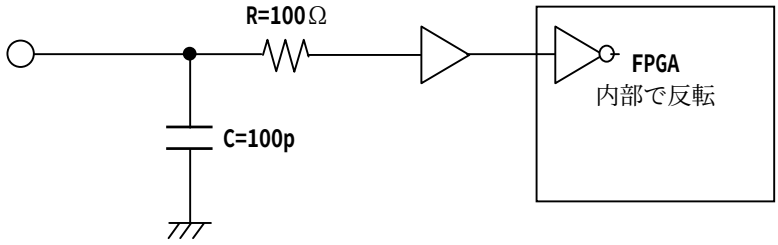
1.3.4 DIO ユニットブロック図



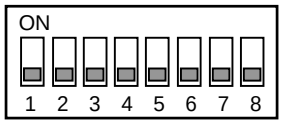
（1）デジタル出力回路(負論理)



（2）デジタル入力回路(負論理)



1-4. 2 台以上のユニットをご使用になる場合



基板上のﾁｯﾌﾟｽｲｯﾁ

基板上のディップスイッチによりユニット ID を設定して下さい。ユニット ID は製品出荷時「0」にセットされています。本ユニットを複数台接続されてご使用になる場合は、個々のユニットに対してユニークな ID を設定しておく必要があります。ユニット ID はカバー上蓋をあけて、下表に示す要領で設定します。尚、8 番のスイッチは ID 設定機能として使用されていません。

ディップスイッチの設定は 1 番のスイッチが 2 進数のビット 0 に相当し、下記要領にて ID 番号を設定します。設定可能範囲は 0 から 127 までです。

ID 番号	ディップスイッチ
0	
1	
2	
3	
4	
5	
6	
7	
8	

1-5. 使用上の注意

1.5.1 セルフパワード(自己電源時)とバスパワード(バスからの給電時)

REX-USB10 は、USB の電源ラインから給電し動作させる場合と、AC アダプタを使用し動作させる場合、両方に対応しています。但し、バスから給電し更に各ポートに給電する Hub を使用し、USB の電源ラインから REX-USB10 に給電する場合、PC 本体から電源容量が不足しているとのメッセージが出力される場合があります。

この場合は、AC アダプタを使用してください。その場合の手順は下記の手順に従ってください。

1) REX-USB10 を PC 本体から切り離します。
2) REX-USB10 に付属の AC アダプタを接続します。
3) PC 本体に再接続します。

REX-USB10 搭載のファームウェアが起動時に、現在 AC アダプタから給電されているかを Check して、PC 本体に REX-USB10 がバスパワーを使用するかどうかを報告します。
その為、必ず REX-USB10 の電源を OFF にする必要があります。

1.5.2 USB の制約について

USB は 12Mbps のバンド幅を持つトークンパケット方式の高速シリアル通信インターフェイスです。しかしながら、実際の運用環境では本製品以外にアイソクロナス転送を使って膨大な量のデータを通信するデバイスの他に USB キーボード、USB マウス等が接続されていることが考えられます。このような複合環境では、アイソクロナス転送を行うデバイスのバンド幅は保証されますが、バルク転送を使ってデータ転送を行う REX-USB10 のバンド幅は保証されていません。これにより、作成した REX-USB10 のアプリケーションのレスポンスが思うように得られなかったり、一時的に転送レートが落ちる場合があります。本問題の有無については、運用される環境により異なりますのでお客様の実環境にて事前に評価する必要があります。

1.5.3 取り扱い上の注意

本製品を取り扱う場合は以下の注意事項を守って下さい

1.本製品を下記のような環境で使用もしくは保管しないで下さい。
直射日光のあたる場所や、高温になる場所
内部に異物（水・ごみ）の入りやすい場所 o
湿気の多い場所や結露しやすい場所
強い磁界・電波を発生する機器の近くや、静電気の影響の多い場所
振動・衝撃の加わる場所
2.外部電源電圧やドライブ電流等の定格を超えないように注意して下さい。
3.AC アダプタは製品付属のもの以外は使用しないで下さい。
4.本製品を改造しないで下さい。改造を行ったものについては一切の責任を負いません。

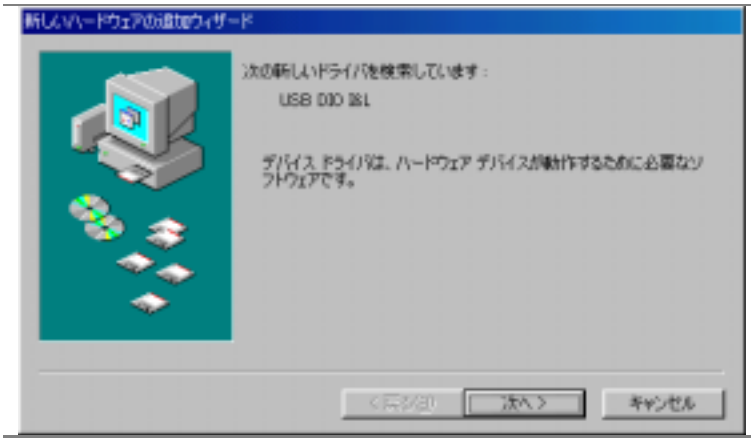
第 2 章 インストレーション

この章では、Windows でのインストール方法について説明します。

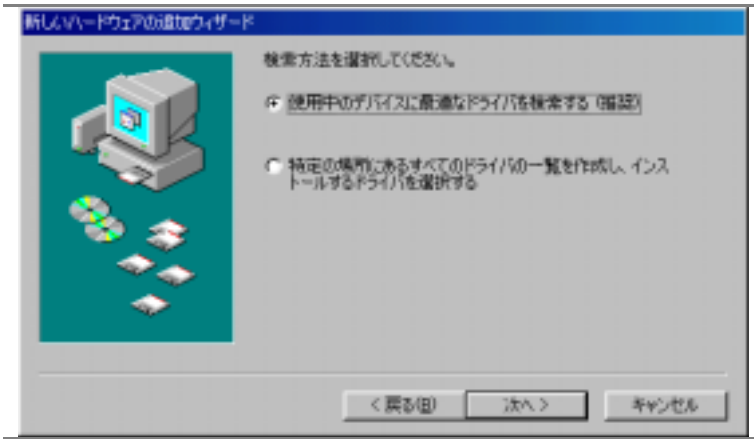
2-1. Windows98/Me 解説

2.1.1 Windows98 インストール

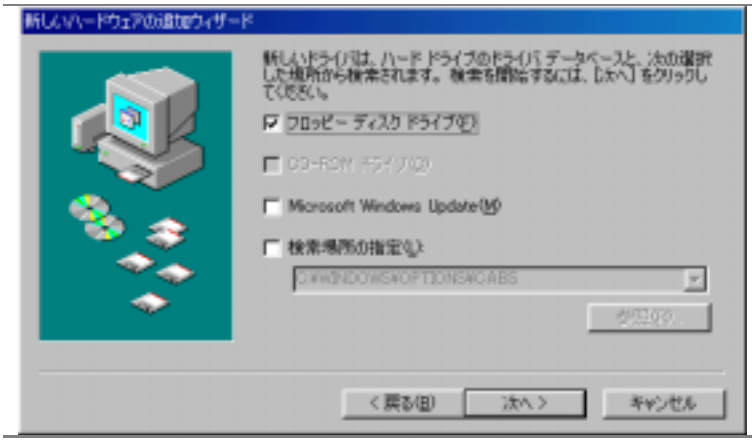
Windows98 環境で本製品をご使用の場合は、下記の手順でインストールを行って下さい。



はじめてパソコンに USB を接続すると、ドライバ情報データベースの作成が開始され、終了すると左図の新しいハードウェア追加ウィザードが起動します。ここでは、「次へ」を選択します。

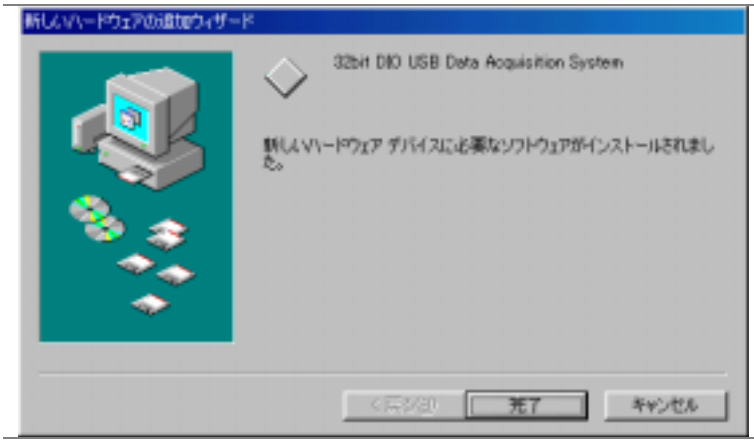


検索方法の選択では、「使用中のデバイスに最適なドライバを検索する」にチェックをいれて「次へ」を選択します。



ドライバの検索では、「フロッピーディスクドライブ」のみチェックを入れます。製品添付の Setup DISK をフロッピーディスクドライブに挿入し、「次へ」を選択します。

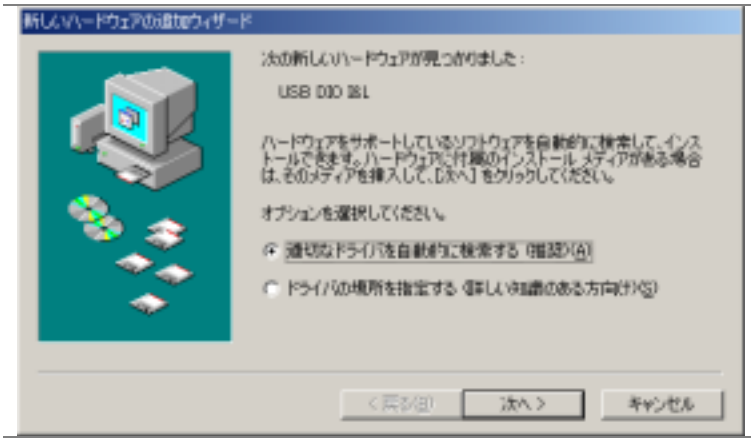
Setup DISK の INF ファイルが正常に認識されると、左図が表示されます。「次へ」を選択すると、ファイルのコピーが開始されます。



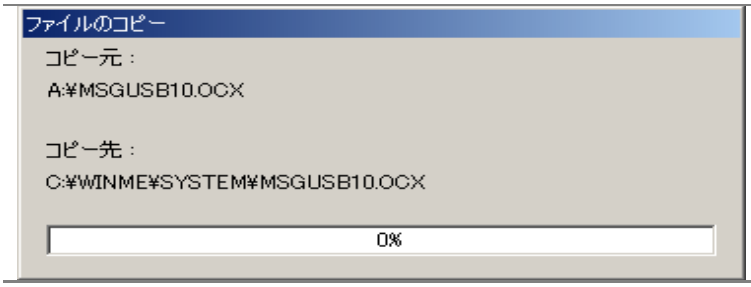
インストールが正常に完了すると左図が表示されます。完了ボタンにより、インストールは終了です。

2.1.2 WindowsMe インストール

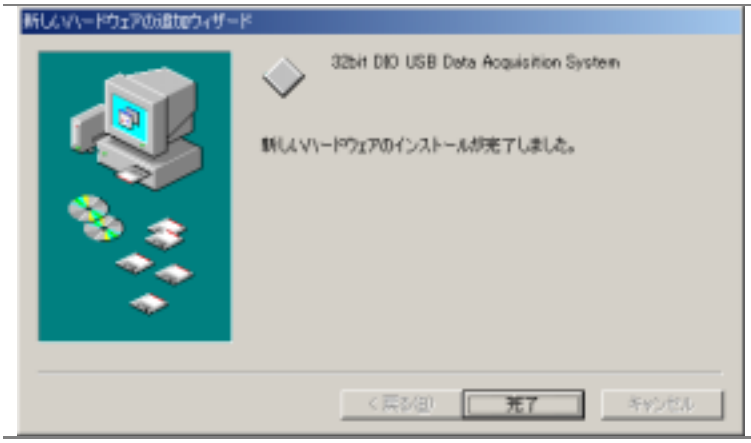
WindowsMe 環境で本製品をご使用の場合は、下記の手順でインストールを行って下さい。



本製品をパソコンに接続すると左図の新しいハードウェア追加ウィザードが起動します。
製品添付のドライバ FD をフロッピーディスクドライブに挿入した後、「適切なドライバを自動的に検索する(推奨)」を選択し、「次へ」を押します。



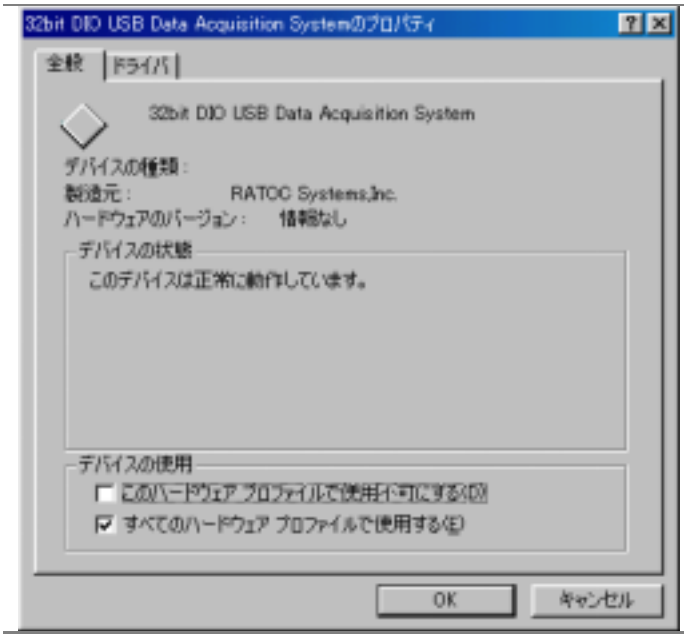
フロッピーディスク内のセットアップファイルを自動検索し、左図のようにファイルのコピーが始まります。



インストールが正常に完了すると左図が表示されます。完了ボタンによりインストールは完了です。

2.1.3 インストール後の確認

コントロールパネルのシステムを起動します。左図のデバイスマネージャ」のタブを開いて DataAcquisition クラスに 32bit DIO USB Data Acquisition System が登録されているか確認します。



上記より、32bit DIO USB Data Acquisition System のプロパティをオープンします。左図のようにデバイスの状態に正常動作が表示されていることを確認します。

2.1.4 アンインストール方法について

インストールした内容を削除する方法について説明します。削除は、(1) デバイスの削除、(2) INF ファイルの削除、(3) ドライバの削除の手順で行います。

(1) デバイスの削除

コントロールパネルのシステムを起動します。下図のデバイスマネージャのタブを開いて DataAcquisition クラスの 32bit DIO USB Data Acquisition System を選択し、削除ボタンを押して削除します。

(2) INF ファイルの削除

エクスプローラからフォルダ「C:¥WINDOWS¥INF¥OTHER」を開き、INF ファイル「RATOC Systems,Inc.REXUSB10.INF」を削除します。

エクスプローラの設定が「全てのファイルを表示」になっていないとフォルダ「C:¥WINDOWS¥INF」は表示されません。設定の変更は、エクスプローラメニューの「表示」から「フォルダオプション」を選択して変更します。

(3) ドライバの削除

「C:¥WINDOWS¥SYSTEM32¥DRIVERS」にコピーされた WDM ドライバ「REXUSB10.SYS」と、「C:¥WINDOWS¥SYSTEM」にコピーされた DLL ライブラリ「USBDIO32.DLL」、OCX「MSGUSB10.OCX」を削除します。

2.2.1 Windows2000 インストール

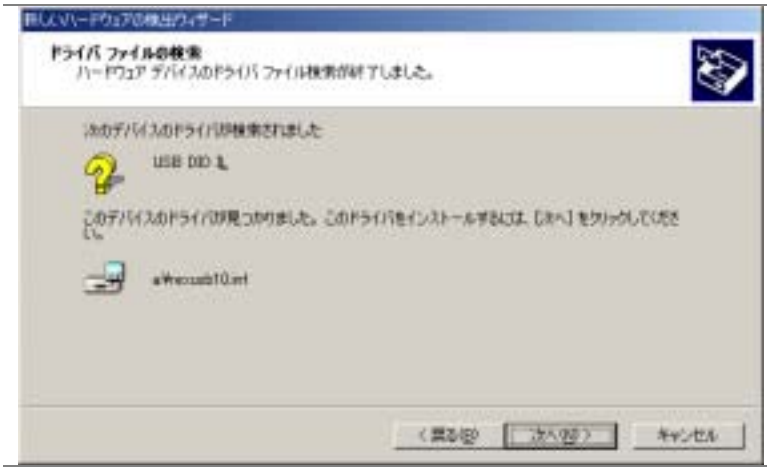
Windows2000 環境で本製品をご使用の場合は、下記の手順でインストールを行って下さい。

本製品をパソコンに接続すると左図の新しいハードウェア追加ウィザードが起動します。

「新しいハードウェアの検索」に表示が切り替わります。ここでは、「次へ」を選択します。

インストールするドライバの検索方法を指定します。ここでは、「デバイスに最適なドライバを検索する」を選択し、次へ進みます。

ドライバ検索場所は「フロッピーディスクドライブ」を指定し、製品添付のインストールディスクをフロッピーディスクドライブにセットした後、「次へ」のボタンを選択します。



フロッピーディスク内のセットアップファイルが見つかります。「次へ」を押します。

インストールが正常に完了すると左図が表示されます。完了ボタンによりインストールは完了です。

2.2.2 WindowsXP インストール

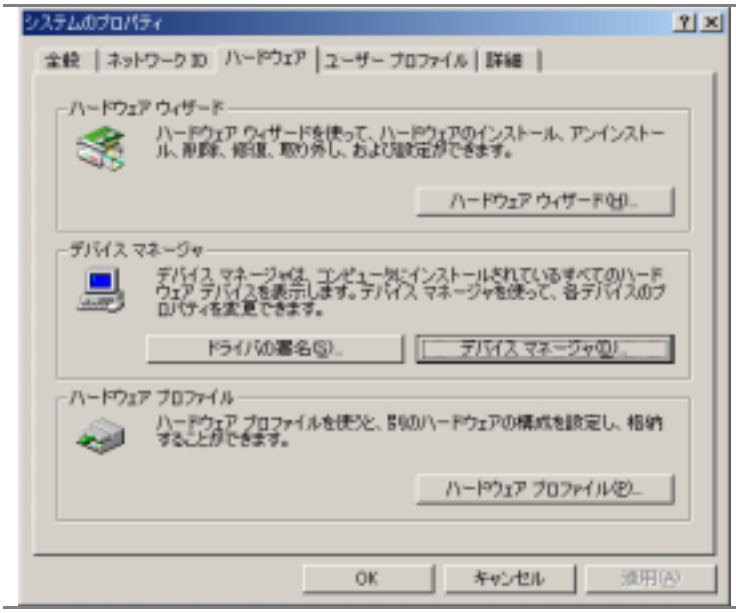
WindowsXP 環境で本製品をご使用の場合は、下記の手順でインストールを行って下さい。

本製品をパソコンに接続すると左図の新しいハードウェアの検出ウィザードが起動します。
製品添付のドライバ FD をフロッピーディスクドライブに挿入した後、インストール方法の選択で「ソフトウェアを自動的にインストールする(推奨)」を選択し「次へ」を押します。

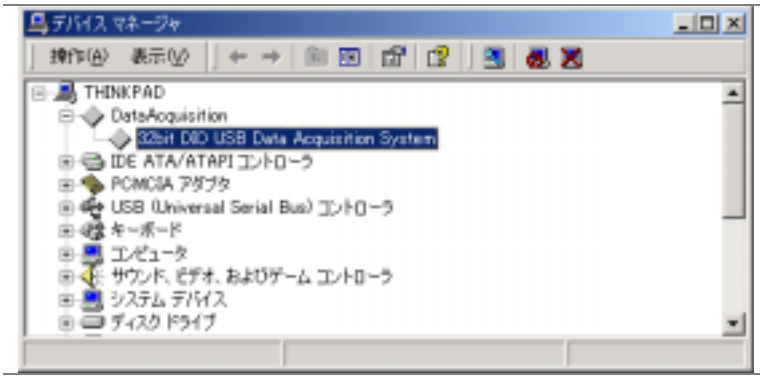
フロッピーディスク内のセットアップファイルを自動検索し、左図のようにファイルのコピーが始まります。

インストールが正常に完了すると左図が表示されます。完了ボタンによりインストールは完了です。

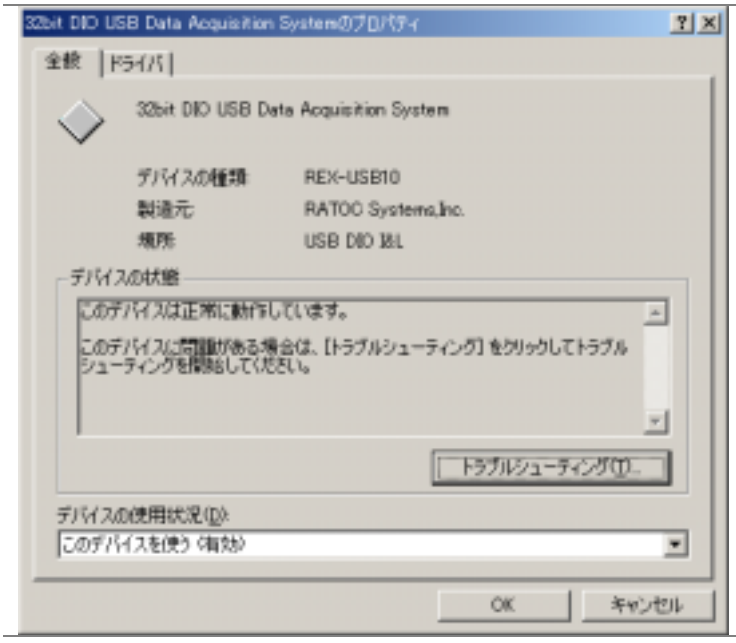
2.2.3 インストール後の確認



コントロールパネルのシステムを起動します。システムのプロパティの「ハードウェア」のタブから「デバイスマネージャ」のボタンを押します。



左図のようにデバイス一覧からDataAcquisition クラスに 32bit DIO USB Data Acquisition System が登録されているか確認します。



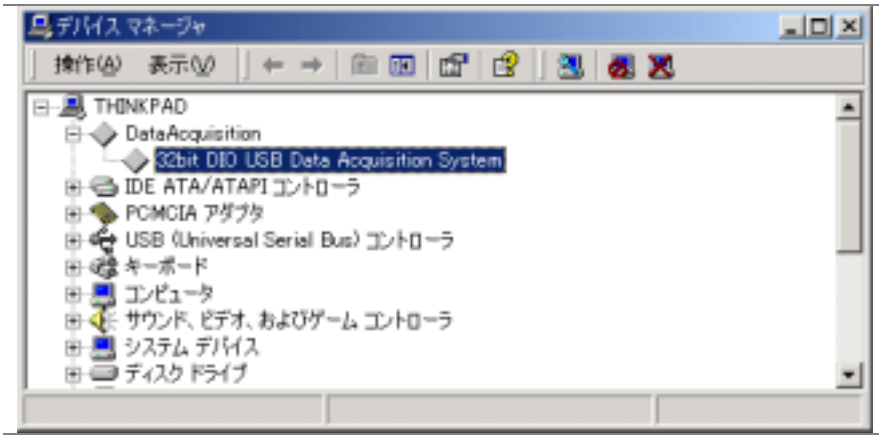
上記より、32bit DIO USB Data Acquisition System のプロパティをオープンします。左図のようにデバイスの状態に正常動作が表示されていることを確認します。

2.2.4 アンインストール方法について

インストールした内容を削除する方法について説明します。削除は、(1) デバイスの削除、(2) INF ファイルの削除、(3) ドライバの削除の手順で行います。

(1) デバイスの削除

コントロールパネルのシステムを起動します。システムのプロパティの「ハードウェア」のタブから「デバイスマネージャ」のボタンを押します。下図のようにデバイス一覧から DataAcquisition クラスの 32bit DIO USB Data Acquisition System を選択し、メニューバーの「操作」→「削除」をクリックします。デバイスの削除の確認メッセージが表示されますので、「OK」ボタンを押して削除します。



(2) INF ファイルの削除

エクスプローラからフォルダ「C:¥WINDOWS¥INF」を開き、INF ファイル「OEMx.INF」を削除します。 x には数字が入ります。例えば OEM0.INF が 1 つだけの場合は、OEM0.INF と拡張子のみ異なる OEM0.PNF を削除してください。OEMx.INF が複数ある場合(OEM0.INF, OEM1.INF,・・・)は、NotePad.exe 等でそれぞれの INF ファイルを開いて、その内容の [Manufacturer]セクションが %RexUsb10.Manufacturer%=RexUsb10 となっているファイルと拡張子のみ異なる PNF ファイルを削除してください。

エクスプローラの設定が「全てのファイルを表示」になっていないとフォルダ「C:¥WINDOWS¥INF」は表示されません。設定の変更は、エクスプローラメニューの「ツール」から「フォルダオプション」を選択して変更します。

(3) ドライバの削除

「C:¥WINDOWS¥SYSTEM32¥DRIVERS」にコピーされた WDM ドライバ「REXUSB10.SYS」と、「C:¥WINDOWS¥SYSTEM」にコピーされた DLL ライブラリ「USBDIO32.DLL」、OCX「MSGUSB10.OCX」を削除します。

第 3 章 Windows アプリケーション作成

この章では、ライブラリ関数仕様とサンプルプログラムについて説明します。

3-1. ライブラリの呼び出し方法

3.1.1 C/C++呼び出し方法

Visual C/C++のアプリケーションから製品に添付された DLL ライブラリ「UsbDio32.DLL」の API を呼び出すには以下の二つの作業が必要です。

(1)製品付属 FD の VC フォルダから「UsbDio32.h」をプロジェクトにコピーし、プログラムにインクルード

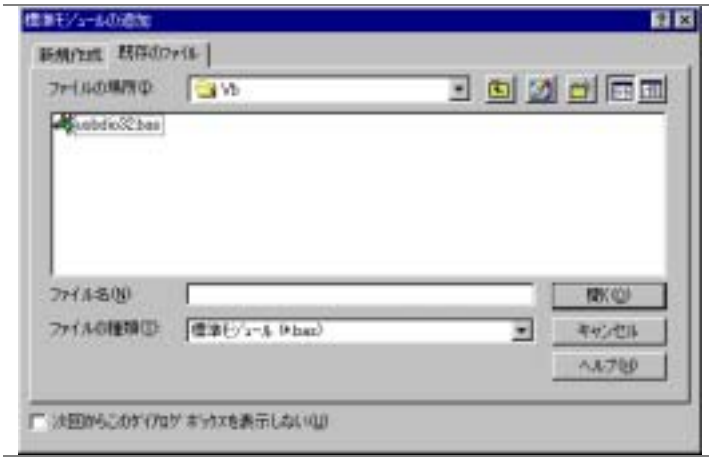
(2)プロジェクト作成後、プロジェクトメニューの「プロジェクトへ追加」->「ファイル」を選択し、ファイルの種類「ライブラリファイル(*.lib)」指定後、製品付属 FD の VC フォルダにある「UsbDio32.lib」をコピーしプロジェクトファイルに追加

以上で、DLL の API 呼び出しが可能になります。

3.1.2 VB 呼び出し方法

Visual BASIC のアプリケーションから製品に添付された DLL ライブラリ「UsbDio32.DLL」の API を呼び出すには以下の二つの作業が必要です。

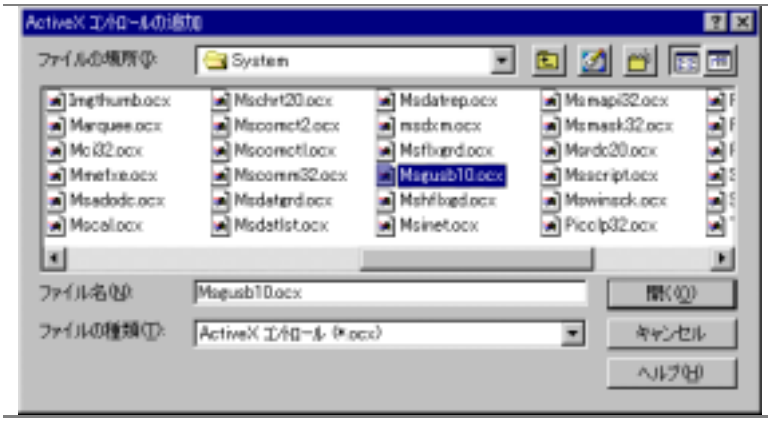
(1) 製品付属 FD の「UsbDio32.Bas」をプロジェクトに追加



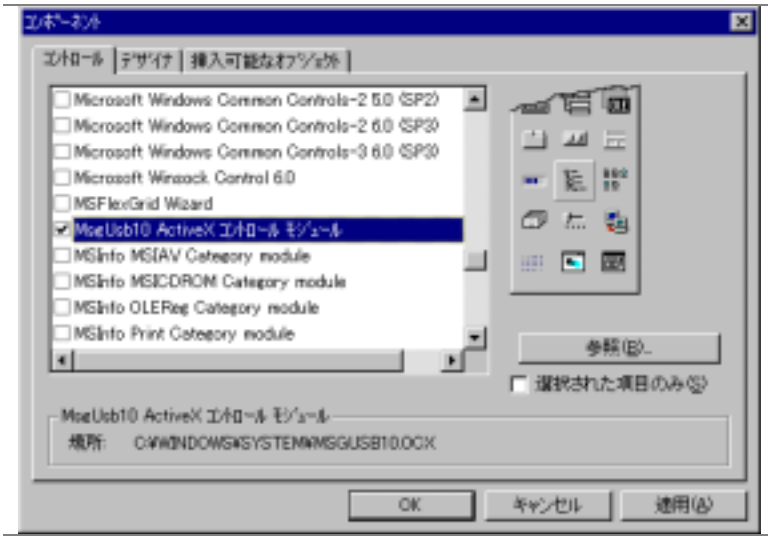
VB プロジェクトメニューから「標準モジュールの追加」を選択します。右図の「既存のファイル」タグを選択し、製品付属の FD の「VB」フォルダから「UsbDio32.Bas」を開きます。

(2)MsgUsb10.ocx をセットアップ

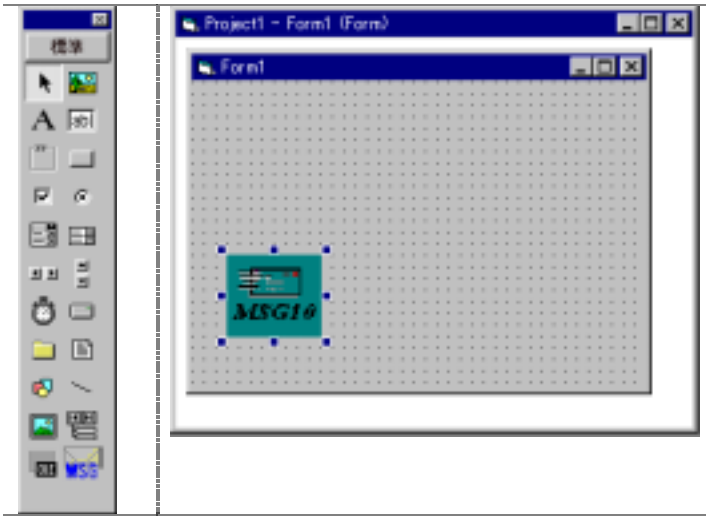
ユーザ定義メッセージを処理するプログラムを作成する場合は、下記の手順で「MsgUsb10.ocx」を登録します。



VB プロジェクトメニューから「コンポーネントの追加」を選択します。次ページのコンポーネントダイアログより、「参照」ボタンを押します。ファイルの場所を「%Windows%\System」フォルダとし「MsgUsb10.ocx」を開きます。



「MsgUsb10 ActiveXｺﾝﾄﾛｰﾙﾓｼﾞｭｰﾙ」にチェックを入れてOKボタンをクリックします。



コントロールバーに「MSG」という名前のコントロールが追加されます。これを選択し、フォームの上に貼り付けます。

上図の「MSG10」をクリックして展開される部分に処理を記述します。

3.2.1 C/C++関数仕様

書 式	INT **dioEnumUnit**(HWND hWnd, PUSHORT pUnitId, USHORT MaxUnit)	
機 能	接続されているユニット ID を MaxUnit に達するまで列挙して、pUnitId に格納します。 呼び出し元は dioOpenUnit()を行っていない状態、全てクローズされている状態で呼び出します。本関数は、複数台のユニットを同一 USB バスに接続して利用する場合、それぞれのユニットに異なる ID が設定されていることを確認する目的で用意されています。1 台のユニットしか利用されない場合は、本関数を呼び出す必要はありません。 dioOpenUnit()で現在のユニット ID を指定してユニットをオープンして下さい。	
引 数	hWnd	ウィンドウハンドル
	pUnitId	列挙したユニット ID の格納先アドレス
	MaxUnit	列挙する最大ユニット台数
戻 値	ユニットの列挙正常終了時、ユニット接続台数を返します。	
	エラーコード	エラーの内容
	-1	オープンできたのにユニットから ID が取得できない内部エラーです。
	-2	同じ ID に設定されたユニットが見つかりました。この時呼び出し元の pUnitId[0]には重複ユニット番号がセットされます。

書 式	INT **dioOpenUnit**(HWND hWnd, USHORT UnitId)	
機 能	指定の ID のユニットをオープンします。正常にオープンされた場合に返されるユニット ID は他の関数呼び出し時に必要となります。 アプリケーション終了時、dioCloseUnit()によりユニットをクローズするようにして下さい。	
引 数	hWnd	ウィンドウハンドル
	UnitId	オープンするユニット ID
戻 値	正常に指定のユニットをオープンした場合はオープンしたユニット ID を返します （オープンした状態ですので呼出し元は終了時ユニットのクローズ処理が必要です）	
	エラーコード	エラーの内容
	-1	ユニット ID 番号指定エラー
	-2	オープンできたユニットから ID 番号が取得できない
	-3	指定ユニット ID のデバイスは接続されていない

書 式	VOID **dioCloseUnit**(USHORT UnitId)	
機 能	ユニットをクローズします。	
引 数	UnitId	クローズするユニット ID
戻 値	なし	

<table>
<tr><td>書 式</td><td colspan="2">INT dioSetDirection
(HWND hwnd, USHORT UnitId, USHORT PortDef, USHORT Direction)</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>機 能</td><td colspan="2">PORT0・PORT1・PORT2・PORT3 の入出力方向を設定します。
PIO0-PIO7:PORT0、PIO8-PIO15:PORT1、PIO16-PIO23:PORT2、PIO24-PIO31:PORT3 の単位で方向を設定することができます。</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>引 数</td><td>hwnd</td><td>ウィンドウハンドル</td></tr>
<tr><td></td><td>UnitId</td><td>ユニット ID</td></tr>
<tr><td></td><td>PortDef</td><td>どのポートの入出力方向を設定するか指定します。
PORT1 と PORT3 の方向を設定する場合：PORT1|PORT3 となります。</td></tr>
<tr><td></td><td>Direction</td><td>各ポートの入出力方向を下記ビットマップで指定します。PortDef で定義されていないポートの方向は変化しません。
<table>
<tr><td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td></tr>
<tr><td>―</td><td>―</td><td>―</td><td>―</td><td>PORT3</td><td>PORT2</td><td>PORT1</td><td>PORT0</td></tr>
<tr><td colspan="8">0:入力方向 1:出力方向</td></tr>
</table></td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>戻 値</td><td colspan="2">正常に方向設定が行われると 0 が返されます。</td></tr>
<tr><td></td><td>エラーコード</td><td>エラーの内容</td></tr>
<tr><td></td><td>-1</td><td>ユニット ID 番号取得エラー</td></tr>
<tr><td></td><td>-2</td><td>ドライバー呼び出しエラー</td></tr>
<tr><td></td><td>-3</td><td>ドライバーシグナル状態応答エラー</td></tr>
<tr><td></td><td>-4</td><td>ユニットがオープンされていない</td></tr>
</table>

書 式	INT **dioOutPort**(HWND hwnd, USHORT UnitId, USHORT PortId, UCHAR Value)	
機 能	ポートに 1 バイトライトします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	出力するポート番号 PORT0,PORT1,PORT2,PORT3 の中からどれかを指定
	Value	出力する値
戻 値	正常に出力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	出力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	INT **dioInPort** (HWND hwnd, USHORT UnitId, USHORT PortId, PUCHAR pValToRead)	
機 能	指定のポートから 1 バイトリードします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	入力するポート番号 PORT0,PORT1,PORT2,PORT3 の中からどれかを指定
	pValToRead	入力した値の格納先アドレス
戻 値	正常に入力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	入力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	INT **dioWOutPort** (HWND hwnd, USHORT UnitId, USHORT PortId, USHORT wOutVal)	
機 能	2 ポートに対し、一度に 2 バイトライトします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	出力するポート番号 PORT0,PORT1,PORT2 の中からどれかを指定します(PORT3 指定不可)。 PORT0 指定時 PORT0/PORT1、PORT1 指定時 PORT1/PORT2、PORT2 指定時 PORT2/PORT3 に対し出力されます。
	wOutVal	出力するワードの値 （各バイトと出力ポートの対応は下記のようになります） 上位バイト 下位バイト PortId で PORT0 指定時 PORT1 PORT0 の対応で出力
戻 値	正常に出力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	出力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	INT **dioWInPort** (HWND hwnd, USHORT UnitId, USHORT PortId, PUSHORT pwInVal)	
機 能	2 ポート同時にリードします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	PORT0,PORT1,PORT2 の中からどれかを指定します(PORT3 指定不可)。 PORT0 指定時 PORT0/PORT1、PORT1 指定時 PORT1/PORT2、PORT2 指定時 PORT2/PORT3 にから入力を行います。
	pwInVal	入力値を格納する USHORT 型変数へのアドレス。 USHORT 型変数の各バイトと入力ポートの対応は下記のようになります。 上位バイト 下位バイト PortId で PORT0 指定時 PORT1 PORT0 の対応で格納
戻 値	正常に入力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	入力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	INT **dioDWInPort**(HWND hwnd, USHORT UnitId, PDWORD pdwInVal)	
機 能	全てのポート（PORT0,PORT1,PORT2,PORT3）に対し、一度に 4 バイトリードします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	pdwInVal	入力値を格納する DWORD 型変数へのアドレス 格納先変数の各バイト位置と入力ポートとの対応は下記のようになります。 最上位バイト 最下位バイト PORT3 | PORT2 | PORT1 | PORT0
戻 値	正常に入力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	INT **dioDWOutPort**(HWND hwnd, USHORT UnitId, DWORD dwOutVal)	
機 能	全てのポート(PORT0,PORT1,PORT2,PORT3)に対し、一度に 4 バイトライトします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	dwOutVal	出力する値（各バイトと出力ポートの対応は下記のようになります） 最上位バイト 最下位バイト PORT3 | PORT2 | PORT1 | PORT0
戻 値	正常に出力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式

INT **dioStartInterrupt**(HWND hwnd, USHORT UnitId)

機 能

本関数は dioSetEventMode()で設定されたイベント検出モードに従って、ユニットの INT 入力端子に接続された TTL レベル信号のオンオフ状態を監視し外部イベントの検出を行います。指定のイベントを検出すると、ユーザ定義メッセージを呼び出し元ウィンドウにポストします。

dioStartInterrupt()呼び出しによりイベント検出を開始し、dioStopInterrupt()により検出動作を停止するまで繰り返し検出動作を行います。

○ポストされるユーザ定義メッセージは下記のものがあります。

1. 正常に外部イベント監視動作に入ると、イベント検出開始を示すユーザ定義メッセージ WM_USBDIOEVENT を呼び出し元ウィンドウにポストします。この時、32 ビットウィンドウメッセージ追加情報 1(WPARAM) の下位ワードにステータス：USBDIO_INTERRUPT_START、上位ワードにメッセージをポストしたユニットの ID がセットされます。
2. 指定の外部イベント検出時、追加情報 1 の下位ワードにステータス：USBDIO_INTERRUPT_END がセットされ、上位ワードにメッセージをポストしたユニットの ID がセットされます。追加情報 2(LPARAM)にはイベント検出時にリードした全てのデジタル I/O ポートの状態が下記レイアウトでセットされます。出力方向にセットされたポートの値は無効となります。

最上位バイト			最下位バイト
PORT3	PORT2	PORT1	PORT0

3. dioStopInterrupt()で実行中の外部イベント検出を中断すると、追加情報 1 の下位ワードにステータス：USBDIO_INTERRUPT_ABORT、上位ワードにイベント検出を停止したユニットの ID がセットされたメッセージがポストされます。
4. 何らかのエラーが発生した場合は、追加情報 1 の下位バイトにステータス:USBDIO_INTERRUPT_ERROR がセットされ、上位バイトにはエラーが発生したユニットの ID がセットされたメッセージがポストされます。この時、追加情報 2 には内部エラーコードがセットされます。

○ユーザ定義メッセージの処理方法

アプリケーション側では下記のようにメッセージを処理して下さい。

```
OnUserDefineMessage (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    WORD EventCode = LOWORD(wParam); // ポストされたメッセージの内容を示すコード
    WORD MsgPostUnitId = HIWORD(wParam); // メッセージをポストしたユニットの ID
    switch ( EventCode )
    {
     case USBDIO_INTERRUPT_END: // イベント発生
    }
```

引 数

hwnd	ウィンドウハンドル
UnitId	ユニット ID

戻 値

イベント検出が正常に実行されると 0 が返されます。

エラーコード	エラーの内容
-1	ユニット ID 番号取得エラー
-2	ドライバー呼び出しエラー
-3	ドライバーシグナル状態応答エラー
-4	イベント検出開始エラー
-5	イベント検出スレッド起動エラー
-6	既にイベント検出は実行されています
-7	ユニットがオープンされていない

書 式	INT **dioStopInterrupt**(USHORT UnitId)	
機 能	dioStartInterrupt()により実行中の外部イベント検出動作を中止します。中止が正常に行われると、ユーザ定義メッセージ WM_USBDIOEVENT がポストされます。この時、32 ビットウィンドウメッセージ追加情報 1 の下位ワードにステータス:USBDIO_INTERRUPT_ABORT がセットされ、上位ワードにイベント検出を停止したユニットの ID がセットされています。	
引 数	UnitId	ユニット ID
戻 値	関数が正常終了すると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ユニットがオープンされていない

書 式	INT **dioSetEventMode**(HWND hwnd, USHORT UnitId, USHORT EventMode)							
機 能	INT 信号入力端子の動作モードを設定します。							
引 数	hwnd	ウィンドウハンドル						
	UnitId	ユニット ID						
	EventMode	INT 信号入力端子の動作モードを下記ビットマップ指定						
		b7	b6	b5	b4	b3	b2	b1	b0
								Mode	Edge
		Mode - 0:INT 端子を外部イベント検出モードで使用 1:INT 端子をデータ転送時のハンドシェークモードで使用 Edge - 0:エッジの立ち上がりでトリガー 1:エッジの立ち下がりでトリガー							
戻 値	正常にモード設定が行われれば 0 が返されます。							
	エラーコード	エラーの内容						
	-1	ユニット ID 番号取得エラー						
	-2	ドライバー呼び出しエラー						
	-3	ドライバーシグナル状態応答エラー						
	-4	ユニットがオープンされていない						

書 式	INT **dioStartWrite** (HWND hwnd, USHORT UnitId, LPVOID pBufToWrite, ULONG nBytesToWrite)	

機 能	指定されたユニットの内部 FIFO バッファにデータを転送します。外部機器への送信は行われません。外部機器への送信は dioStartHandShake ()で開始されます。
	dioStartWrite()で一度に送信可能な最大バイト数は 100K バイトです。それ以上のサイズのデータを送信する場合は、一度 dioStartHandShake ()により外部機器へデータ送信し内部 FIFO バッファを空の状態にする必要があります。
	
	○ポストされるユーザ定義メッセージは下記のものがあります。
	1. 4K バイト単位でユニットへの転送が完了するとユーザ定義メッセージ WM_USBDIOEVENT を呼び出し元ウィンドウにポストします。この時、32 ビットウィンドウメッセージ追加情報 1(WPARAM)の下位ワードにステータス:USBDIO_WRITE_START、上位ワードにメッセージをポストしたユニットの ID がセットされます。このメッセージは必要がなければ特に処理する必要はありません。
	2. 全ての転送が完了すると、追加情報 1 の下位ワードにステータス:USBDIO_WRITE_END、上位ワードにユニットの ID がセットされます。追加情報 2(LPARAM)に転送完了バイト数がセットされます。
	3. エラーが発生すると、追加情報 1 の下位ワードにステータス:USBDIO_WRITE_ERROR が、上位ワードにユニットの ID がセットされます。追加情報 2 に内部エラーコードがセットされます。
	
	○ユーザ定義メッセージの処理方法
	OnUserDefineMessage (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
	{
	WORD EventCode = LOWORD(wParam); // ポストされたメッセージの内容を示すコード
	WORD MsgPostUnitId = HIWORD(wParam); // メッセージをポストしたユニットの ID
	・・・
	}	

引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	pBufToWrite	ユニットに転送するデータが格納されているバッファのアドレス
	nBytesToWrite	ユニットに転送するバイト数
戻 値	データ転送が正常に開始されると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	データ転送開始エラー
	-5	データ転送スレッド起動エラー
	-6	ユニットがオープンされていない

書 式	INT **dioStopWrite**(USHORT UnitId)	
機 能	dioStartWrite()により実行中のデータ転送動作を中止します。中止が正常に行われると、ユーザ定義メッセージ WM_USBDIOEVENT がポストされます。この時、32 ビットウィンドウメッセージ追加情報 1 の下位ワードにステータス：USBDIO_WRITE_STOP がセットされ、上位ワードにイベント検出を停止したユニットの ID がセットされています。	
引 数	UnitId	ユニット ID
戻 値	正常に完了通知が終了すると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	INT **dioStopRead**(USHORT UnitId)	
機 能	dioStartRead()により実行中のデータ転送動作を中止します。中止が正常に行われると、ユーザ定義メッセージ WM_USBDIOEVENT がポストされます。この時、32 ビットウィンドウメッセージ追加情報 1 の下位ワードにステータス：USBDIO_READ_STOP がセットされ、上位ワードにイベント検出を停止したユニットの ID がセットされています。	
引 数	UnitId	ユニット ID
戻 値	正常に完了通知が終了すると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	INT **dioStartRead** (HWND hwnd, USHORT UnitId, LPVOID pBufToRead, ULONG nBytesToRead)	

機 能	指定されたユニットの内部 FIFO バッファに格納されているデータをパソコンに転送します。従って dioStartRead()を実行する前に、予め外部機器からのデータ受信を完了しておく必要があります。外部機器からの受信は dioStartHandShake ()で行われます。
	本関数を呼び出す前に、アプリケーション側は予めデータを格納するためのバッファをアロケーションしておく必要があります。現在ユニットのバッファに格納されているデータのサイズは dioGetNumInFifo()で知ることができます。この情報を元に、アプリケーションはデータ格納用バッファをアロケーションし pBufToRead にアドレスを、nBytesToRead にアロケーションしたバッファのサイズを渡してください。
	
	○ポストされるユーザ定義メッセージは下記のものがあります。
	1. 4K バイト単位でユニットへの転送が完了するとユーザ定義メッセージ WM_USBDIOEVENT を呼び出し元ウィンドウにポストします。この時、32 ビットウィンドウメッセージ追加情報 1(WPARAM)の下位ワードにステータス：USBDIO_READ_START、上位ワードにメッセージをポストしたユニットの ID がセットされます。
	2. 全ての転送が完了すると、追加情報 1 の下位ワードにステータス：USBDIO_READ_END、上位ワードにユニットの ID がセットされます。追加情報 2(LPARAM)に転送完了バイト数がセットされます。
	3. エラーが発生すると、追加情報 1 の下位ワードにステータス：USBDIO_READ_ERROR が、上位ワードにユニットの ID がセットされます。追加情報 2 に内部エラーコードがセットされます。
	
	○ユーザ定義メッセージの処理方法
	OnUserDefineMessage (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
	{
	WORD EventCode = LOWORD(wParam); // ポストされたメッセージの内容を示すコード
	WORD MsgPostUnitId = HIWORD(wParam); // メッセージをポストしたユニットの ID
	・・・
	}	

引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	pBufToRead	ユニットから転送したデータを格納するバッファのアドレス
	nBytesToRead	ユニットから転送するバイト数
戻 値	データ転送が正常に開始されると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	データ転送開始エラー
	-5	データ転送スレッド起動エラー
	-6	ユニットがオープンされていない

書 式	INT **dioGetNumInFifo**(HWND hwnd, USHORT UnitId, PULONG pByteInFifo)	
機 能	現在ユニットの内部 FIFO バッファに存在する有効データバイト数を取得します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	pByteInFifo	取得した FIFO データバイト数格納先アドレス
戻 値	正常に取得できれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	FIFO オーバーフローによりユニットはデータ転送を停止しています。この時、取得したデータ個数は正しい値が返されます。
	-5	FIFO が空になったためにユニットはデータ出力を停止しました。このエラーは、クロック同期で外部にデータを出力している時にのみ発生します。
	-6	その他のユニット内部エラー
	-7	ユニットがオープンされていない

書 式	INT **dioSetHandShakeMode** (HWND hwnd, USHORT UnitId, USHORT TxDirection, USHORT TxWidth)	
機 能	dioStartHandShake()での転送動作を設定します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	TxDirection	データ転送方向指定 DIOUNIT_TO_EXTERNAL ： ユニットから外部機器に送信 EXTERNAL_TO_DIOUNIT ： 外部機器からユニットに受信
	TxWidth	データ転送ビット幅指定 TXWIDTH_32PORT ： 全ポートを使った 32 ビットデータ転送 TXWIDTH_16PORT ： PORT0/PORT1 を使った 16 ビットデータ転送 TXWIDTH_8PORT ： PORT0 による 8 ビットデータ転送
戻 値		
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	データ転送方向指定エラー
	-5	データ転送ビット幅指定エラー
	-6	ユニットがオープンされていない

書 式	INT **dioStartHandShake**(HWND hwnd, USHORT UnitId, ULONG nBytes)	
機 能	dioSetHandShakeMode()で設定されたモードに従って外部機器とのデータ送受信を開始します。本関数を呼び出す場合は、dioSetHandShakeMode()でハンドシェークモードを指定して下さい。 外部機器にデータを送信する場合は、予め dioStartWrite()によりユニットの内部 FIFO バッファに外部機器に送信するデータ格納しておく必要があります。 外部機器からデータを受信する場合はユニットは内部 FIFO バッファにデータを受信します。受信完了後、アプリケーションは dioGetNumInFifo()により FIFO に格納されたデータ個数を取得し、dioStartRead()により FIFO からデータをパソコン側に転送します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	nBytes	転送バイト数
戻 値	転送が正常に開始されれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ハンドシェークデータ転送開始エラー
	-5	ユニットがオープンされていない

書 式	INT **dioStopHandShake**(USHORT UnitId)	
機 能	dioStartHandShake()で開始したデータ転送を終了します。	
引 数	UnitId	ユニット ID
戻 値	停止が正常に行われれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	INT **dioSetClockMode** (HWND hwnd, USHORT UnitId, USHORT TxDirection, USHORT TxWidth)	
機 能	ユニットの内部クロックに同期してデータ入出力を行うモードを設定します。	
引 数	Hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	TxDirection	入出力方向指定 DIOUNIT_TO_EXTERNAL ： ユニットから外部機器に送信 EXTERNAL_TO_DIOUNIT ： 外部機器からユニットに受信
	TxWidth	入出力ビット幅指定 TXWIDTH_32PORT ： 全ポートを使った 32 ビット入出力 TXWIDTH_16PORT ： PORT0/PORT1 を使った 16 ビット入出力 TXWIDTH_8PORT ： PORT0 による 8 ビット入出力
戻 値	モード設定が正常に行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	データ転送方向指定エラー
	-5	データ転送ビット幅指定エラー
	-6	ユニットがオープンされていない

書 式	INT **dioSetClock** (HWND hwnd, USHORT UnitId, USHORT ClockTimer, USHORT TimeUnit)	
機 能	ユニットの内部クロックに同期した入出力を行う場合のクロック周期を設定します。 クロック周期は ClockTimer×1.0049 秒の値になります。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	ClockTimer	時間を指定
	TimeUnit	0 を指定してください
戻 値	設定が正常に行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	クロック設定エラー
	-5	単位指定エラー
	-6	設定可能範囲エラー
	-7	ユニットがオープンされていない

書 式	INT **dioStartClock**(HWND hwnd, USHORT UnitId, ULONG nBytes)	
機 能	dioSetClockMode()で設定されたモードと、dioSetClock()で設定された内部クロックに同期したデータ送受信を開始します。本関数を呼び出す場合は、dioSetClockMode()/dioSetClock()でクロック転送のパラメータを設定して下さい。 外部機器にデータを送信する場合は、予め dioStartWrite()によりユニットの内部 FIFO バッファに外部機器に送信するデータ格納しておく必要があります。 外部機器からデータを受信する場合はユニットは内部 FIFO バッファにデータを受信します。受信完了後、アプリケーションは dioGetNumInFifo()により FIFO に格納されたデータ個数を取得し、dioStartRead()により FIFO からデータをパソコン側に転送します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	nBytes	転送バイト数
戻 値	転送が正常に開始されれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	INT **dioStopClock**(HWND hwnd, USHORT UnitId)	
機 能	dioStartClock()で開始したクロック同期転送を停止します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
戻 値	正常に停止されれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

3.2.2 VB 関数仕様

書 式	Declare Function **dioEnumUnit** Lib "UsbDio32.dll" (ByVal hwnd As Long, pUnitId As Any, ByVal MaxUnit As Integer) As Long	
機 能	接続されているユニット ID を MaxUnit に達するまで列挙して、pUnitId に格納します。 呼び出し元は dioOpenUnit()を行っていない状態、全てクローズされている状態で呼び出します。本関数は、複数台のユニットを同一 USB バスに接続して利用する場合、それぞれのユニットに異なる ID が設定されていることを確認する目的で用意されています。1 台のユニットしか利用されない場合は、本関数を呼び出す必要はありません。 dioOpenUnit()で現在のユニット ID を指定してユニットをオープンして下さい。	
引 数	hwnd	ウィンドウハンドル
	pUnitId	列挙したユニット ID の格納先アドレス
	MaxUnit	列挙する最大ユニット台数
戻 値	ユニットの列挙正常終了時、ユニット接続台数を返します。	
	エラーコード	エラーの内容
	-1	オープンできたのにユニットから ID が取得できない内部エラーです。
	-2	同じ ID に設定されたユニットが見つかりました。この時呼び出し元の pUnitId(0)には重複ユニット番号がセットされます。

書 式	Declare Function **dioOpenUnit** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer) As Long	
機 能	指定の ID のユニットをオープンします。正常にオープンされた場合に返されるユニット ID は他の関数呼び出し時に必要となります。 アプリケーション終了時、dioCloseUnit()によりユニットをクローズするようにして下さい。	
引 数	hwnd	ウィンドウハンドル
	UnitId	オープンするユニット ID
戻 値	正常に指定のユニットをオープンした場合はオープンしたユニット ID を返します （オープンした状態ですので呼出し元は終了時ユニットのクローズ処理が必要です）	
	エラーコード	エラーの内容
	-1	ユニット ID 番号指定エラー
	-2	オープンできたユニットから ID 番号が取得できない
	-3	指定ユニット ID のデバイスは接続されていない

書 式	Declare Sub **dioCloseUnit** Lib "UsbDio32.dll" (ByVal UnitId As Integer)	
機 能	ユニットをクローズします。	
引 数	UnitId	クローズするユニット ID
戻 値	なし	

書 式	Declare Function **dioSetDirection** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal PortDef As Integer, ByVal Direction As Integer) As Long	
機 能	PORT0・PORT1・PORT2・PORT3 の入出力方向を設定します。 PIO0-PIO7:PORT0、PIO8-PIO15:PORT1、PIO16-PIO23:PORT2、PIO24-PIO31:PORT3 の単位で方向を設定することができます。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortDef	どのポートの入出力方向を設定するか指定します。 PORT1 と PORT3 の方向を設定する場合：PORT1|PORT3 となります。
	Direction	各ポートの入出力方向を下記ビットマップで指定します。PortDef で定義されていないポートの方向は変化しません。

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	PORT3	PORT2	PORT1	PORT0

0:入力方向 1:出力方向

戻 値	正常に方向設定が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	Declare Function **dioOutPort** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal PortId As Integer, ByVal bOutVal As Byte) As Long	
機 能	指定のポートに 1 バイトライトします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	出力するポート番号 PORT0,PORT1,PORT2,PORT3 の中からどれかを指定
	bOutVal	出力する値
戻 値	正常に出力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	出力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	Declare Function **dioInPort** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal PortId As Integer, pValToRead As Any) As Long	
機 能	指定のポートから 1 バイトリードします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	出力するポート番号 PORT0,PORT1,PORT2,PORT3 の中からどれかを指定
	pValToRead	入力した値の格納先アドレス
戻 値	正常に入力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	入力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	Declare Function **dioWOutPort** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal PortId As Integer, ByVal wOutVal As Long) As Long	
機 能	2 ポートに対し、一度に 2 バイトライトします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	出力するポート番号 PORT0,PORT1,PORT2 の中からどれかを指定します(PORT3 指定不可)。 PORT0 指定時 PORT0/PORT1、PORT1 指定時 PORT1/PORT2、PORT2 指定時 PORT2/PORT3 に対し出力されます。
	wOutVal	出力するワードの値 （各バイトと出力ポートの対応は下記のようになります） 上位バイト　下位バイト PortId で PORT0 指定時 | PORT1 | PORT0 | の対応で出力
戻 値	正常に出力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	出力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	Declare Function **dioWInPort** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal PortId As Integer, pwInVal As Any) As Long	
機 能	2 ポート同時にリードします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	PortId	PORT0,PORT1,PORT2 の中からどれかを指定します(PORT3 指定不可)。 PORT0 指定時 PORT0/PORT1、PORT1 指定時 PORT1/PORT2、PORT2 指定時 PORT2/PORT3 にから入力を行います。
	pwInVal	入力値を格納する USHORT 型変数へのアドレス。 USHORT 型変数の各バイトと入力ポートの対応は下記のようになります。 上位バイト　下位バイト PortId で PORT0 指定時 PORT1 | PORT0 の対応で出力
戻 値	正常に入力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	入力ポート番号の指定誤り
	-5	ユニットがオープンされていない

書 式	Declare Function **dioDWOutPort** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal dwOutVal As Long) As Long	
機 能	全てのポート(PORT0,PORT1,PORT2,PORT3)に対し、一度に 4 バイトライトします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
		出力する値（各バイトと出力ポートの対応は下記のようになります） 最上位バイト　最下位バイト PORT3 | PORT2 | PORT1 | PORT0
戻 値	正常に出力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	Declare Function **dioDWInPort** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, pdwInVal As Any) As Long	
機 能	全てのポート（PORT0,PORT1,PORT2,PORT3）に対し、一度に 4 バイトリードします。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	pdwInVal	入力値を格納する DWORD 型変数へのアドレス 格納先変数の各バイト位置と入力ポートとの対応は下記のようになります。 最上位バイト　　最下位バイト PORT3 | PORT2 | PORT1 | PORT0
戻 値	正常に入力が行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	Declare Function **dioSetEventMode** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal EventMode As Integer) As Long	
機 能	INT 信号入力端子の動作モードを設定します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	EventMode	INT 信号入力端子の動作モードを下記ビットマップ指定 b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 — | — | — | — | — | — | Mode | Edge Mode - 0:INT 端子を外部イベント検出モードで使用 1:INT 端子をデータ転送時のハンドシェークモードで使用 Edge - 0:エッジの立ち上がりでトリガー 1:エッジの立ち下がりでトリガー
戻 値	正常にモード設定が行われれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式

```
Declare Function dioStartInterrupt Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer) As Long
```

機 能

本関数は dioSetEventMode()で設定されたイベント検出モードに従って、ユニットの INT 入力端子に接続された TTL レベル信号のオンオフ状態を監視し外部イベントの検出を行います。指定のイベントを検出すると、ユーザ定義メッセージを呼び出し元ウィンドウにポストします。
dioStartInterrupt()呼び出しによりイベント検出を開始し、dioStopInterrupt()により検出動作を停止するまで繰り返し検出動作を行います。
○ポストされるユーザ定義メッセージは下記のものがあります。

1. 正常に外部イベント監視動作に入ると、イベント検出開始を示すユーザ定義メッセージ WM_USBDIOEVENT を呼び出し元ウィンドウにポストします。この時、32 ビットウィンドウメッセージ追加情報 1(WPARAM) の下位ワードにステータス：USBDIO_INTERRUPT_START、上位ワードにメッセージをポストしたユニットの ID がセットされます。
2. 指定の外部イベント検出時、追加情報 1 の下位ワードにステータス：USBDIO_INTERRUPT_END がセットされ、上位ワードにメッセージをポストしたユニットの ID がセットされます。追加情報 2(LPARAM)にはイベント検出時にリードした全てのデジタル I/O ポートの状態が下記レイアウトでセットされます。出力方向にセットされたポートの値は無効となります。

最上位バイト　　最下位バイト

PORT3	PORT2	PORT1	PORT0

3. dioStopInterrupt()で実行中の外部イベント検出を中断すると、追加情報 1 の下位ワードにステータス：USBDIO_INTERRUPT_ABORT、上位ワードにイベント検出を停止したユニットの ID がセットされたメッセージがポストされます。
4. 何らかのエラーが発生した場合は、追加情報 1 の下位バイトにステータス:USBDIO_INTERRUPT_ERROR がセットされ、上位バイトにはエラーが発生したユニットの ID がセットされたメッセージがポストされます。この時、追加情報 2 には内部エラーコードがセットされます。

○ユーザ定義メッセージの処理方法
Visual BASIC ではこのメッセージを受け取るために製品添付の MsgUsb10（OCX）の登録が必要です。呼び出しもとでのメッセージ処理は下記のように行います。

```
Private Sub MsgUsb_OnMsgPost(ByVal wParam As Long, ByVal lParam As Long)
    Dim EventCode As Long, MsgPostUnitId As Long
    EventCode = wParam Mod &HFFFF
    MsgPostUnitId = wParam \ &HFFFF
    Select Case EventCode
        Case USBDIO_INTERRUPT_START; IDS_STATUS.Caption = "イベント待ち..."
        Case USBDIO_INTERRUPT_ERROR; IDS_STATUS.Caption = "イベント検出エラー"
    End Select
End Sub
```

引 数

hwnd	ウィンドウハンドル
UnitId	ユニット ID

戻 値

イベント検出が正常に実行されると 0 が返されます。

エラーコード	エラーの内容
-1	ユニット ID 番号取得エラー
-2	ドライバー呼び出しエラー
-3	ドライバーシグナル状態応答エラー
-4	イベント検出開始エラー
-5	イベント検出スレッド起動エラー
-6	既にイベント検出は実行されています
-7	ユニットがオープンされていない

書 式	Declare Function **dioStopInterrupt** Lib "UsbDio32.dll" (ByVal UnitId As Integer) As Long	
機 能	dioStartInterrupt()により実行中の外部イベント検出動作を中止します。中止が正常に行われると、ユーザ定義メッセージ WM_USBDIOEVENT がポストされます。この時、32 ビットウィンドウメッセージ追加情報 1 の下位ワードにステータス:USBDIO_INTERRUPT_ABORT がセットされ、上位ワードにイベント検出を停止したユニットの ID がセットされています。	
引 数	UnitId	ユニット ID
戻 値	関数が正常終了すると 0 が返されます。実際の外部イベント監視動作の中止は上記ユーザ定義メッセージを処理して知ることができます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ユニットがオープンされていない

書 式	Declare Function **dioStartWrite** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, pBufToWrite As Any, ByVal nBytesToWrite As Long) As Long
機 能	指定されたユニットの内部 FIFO バッファにデータを転送します。外部機器への送信は行われません。外部機器への送信は dioStartHandShake で開始されます。 dioStartWrite()で一度に送信可能な最大バイト数は 100K バイトです。それ以上のサイズのデータを送信する場合は、一度 dioStartHandShake により外部機器へデータ送信し内部 FIFO バッファを空の状態にする必要があります。 ○ポストされるユーザ定義メッセージは下記のものがあります。 1. 4K バイト単位でユニットへの転送が完了するとユーザ定義メッセージ WM_USBDIOEVENT を呼び出し元ウィンドウにポストします。この時、32 ビットウィンドウメッセージ追加情報 1(WPARAM)の下位ワードにステータス:USBDIO_WRITE_START、上位ワードにメッセージをポストしたユニットの ID がセットされます。このメッセージは必要がなければ特に処理する必要はありません。 2. 全ての転送が完了すると、追加情報 1 の下位ワードにステータス:USBDIO_WRITE_END、上位ワードにユニットの ID がセットされます。追加情報 2(LPARAM)に転送完了バイト数がセットされます。 3. エラーが発生すると、追加情報 1 の下位ワードにステータス：USBDIO_WRITE_ERROR が、上位ワードにユニットの ID がセットされます。追加情報 2 に内部エラーコードがセットされます。 ○ユーザ定義メッセージの処理方法 Visual BASIC ではこれらのメッセージを受け取るために製品添付の MsgUsb10（OCX）の登録が必要です。呼び出しもとでのメッセージ処理は下記のように行います。 Private Sub MsgUsb_OnMsgPost(ByVal wParam As Long, ByVal lParam As Long) Dim EventCode As Long, MsgPostUnitId As Long EventCode = wParam Mod &HFFFF MsgPostUnitId = wParam \ &HFFFF Select Case EventCode Case USBDIO_WRITE_START; IDS_STATUS.Caption = "ユニットへの転送開始" Case USBDIO_WRITE_ END; ・・・ Case USBDIO_ WRITE _ERROR; ・・・ End Select End Sub

引 数		
	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	pBufToWrite	ユニットに転送するデータが格納されているバッファのアドレス
	nBytesToWrite	ユニットに転送するバイト数

戻 値		
	データ転送が正常に開始されると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	データ転送開始エラー
	-5	データ転送スレッド起動エラー
	-6	ユニットがオープンされていない

書 式	Declare Function **dioStopWrite** Lib "UsbDio32.dll" (ByVal UnitId As Integer) As Long	
機 能	dioStartWrite()により実行中のデータ転送動作を中止します。中止が正常に行われると、ユーザ定義メッセージ WM_USBDIOEVENT がポストされます。この時、32 ビットウィンドウメッセージ追加情報 1 の下位ワードにステータス：USBDIO_WRITE_STOP がセットされ、上位ワードにイベント検出を停止したユニットの ID がセットされています。	
引 数	UnitId	ユニット ID
戻 値	正常に完了通知が終了すると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	Declare Function **dioStopRead** Lib "UsbDio32.dll" (ByVal UnitId As Integer) As Long	
機 能	dioStartRead()により実行中のデータ転送動作を中止します。中止が正常に行われると、ユーザ定義メッセージ WM_USBDIOEVENT がポストされます。この時、32 ビットウィンドウメッセージ追加情報 1 の下位ワードにステータス：USBDIO_READ_STOP がセットされ、上位ワードにイベント検出を停止したユニットの ID がセットされています。	
引 数	UnitId	ユニット ID
戻 値	正常に完了通知が終了すると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式

```
Declare Function dioStartRead Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, pBufToRead As Any, ByVal nBytesToRead As Long) As Long
```

機 能

指定されたユニットの内部 FIFO バッファに格納されているデータをパソコンに転送します。従って dioStartRead()を実行する前に、予め外部機器からのデータ受信を完了しておく必要があります。外部機器からの受信は dioStartHandShake ()で行われます。
本関数を呼び出す前に、アプリケーション側は予めデータを格納するためのバッファをアロケーションしておく必要があります。現在ユニットのバッファに格納されているデータのサイズは dioGetNumInFifo()で知ることができます。この情報を元に、アプリケーションはデータ格納用バッファをアロケーションし pBufToRead にアドレスを、nBytesToRead にアロケーションしたバッファのサイズを渡してください。

○ポストされるユーザ定義メッセージは下記のものがあります。

1. 4K バイト単位でユニットへの転送が完了するとユーザ定義メッセージ WM_USBDIOEVENT を呼び出し元ウィンドウにポストします。この時、32 ビットウィンドウメッセージ追加情報 1(WPARAM)の下位ワードにステータス：USBDIO_READ_START、上位ワードにメッセージをポストしたユニットの ID がセットされます。
2. 全ての転送が完了すると、追加情報 1 の下位ワードにステータス：USBDIO_READ_END、上位ワードにユニットの ID がセットされます。追加情報 2(LPARAM)に転送完了バイト数がセットされます。
3. エラーが発生すると、追加情報 1 の下位ワードにステータス：USBDIO_READ_ERROR が、上位ワードにユニットの ID がセットされます。追加情報 2 に内部エラーコードがセットされます。

○ユーザ定義メッセージの処理方法
Visual BASIC ではこれらのメッセージを受け取るために製品添付の MsgUsb10（OCX）の登録が必要です。呼び出しもとでのメッセージ処理は下記のように行います。

```
Private Sub MsgUsb_OnMsgPost(ByVal wParam As Long, ByVal lParam As Long)
    Dim EventCode As Long, MsgPostUnitId As Long
    EventCode = wParam Mod &HFFFF
    MsgPostUnitId = wParam \ &HFFFF
    Select Case EventCode
        Case USBDIO_ READ _START;  IDS_STATUS.Caption = "ユニットへの転送開始"
        Case USBDIO_ READ _ END;    ・・・
        Case USBDIO_ READ _ERROR; ・・・
    End Select
End Sub
```

引 数

hwnd	ウィンドウハンドル
UnitId	ユニット ID
pBufToRead	ユニットから転送したデータを格納するバッファのアドレス
nBytesToRead	ユニットから転送するバイト数

戻 値

データ転送が正常に開始されると 0 が返されます。

エラーコード	エラーの内容
-1	ユニット ID 番号取得エラー
-2	ドライバー呼び出しエラー
-3	ドライバーシグナル状態応答エラー
-4	データ転送開始エラー
-5	データ転送スレッド起動エラー
-6	ユニットがオープンされていない

書 式	Declare Function **dioGetNumInFifo** Lib "UsbDio32.dll" (ByVal hwnd As Any, ByVal UnitId As Integer, pByteInFifo As Any) As Long	
機 能	現在ユニットの内部 FIFO バッファに存在する有効データバイト数を取得します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	pByteInFifo	取得した FIFO データバイト数格納先アドレス
戻 値	正常に取得できれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	FIFO オーバーフローによりユニットはデータ転送を停止しています。この時、取得したデータ個数は正しい値が返されます。
	-5	FIFO が空になったためにユニットはデータ出力を停止しました。このエラーは、クロック同期で外部にデータを出力している時にのみ発生します。
	-6	その他のユニット内部エラー
	-7	ユニットがオープンされていない

書 式	Declare Function **dioSetHandShakeMode** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal TxDirection As Integer, ByVal TxWidth As Integer) As Long	
機 能	dioStartHandShake()での転送動作を設定します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	TxDirection	データ転送方向指定 DIOUNIT_TO_EXTERNAL ： ユニットから外部機器に送信 EXTERNAL_TO_DIOUNIT ： 外部機器からユニットに受信
	TxWidth	データ転送ビット幅指定 TXWIDTH_32PORT ： 全ポートを使った 32 ビットデータ転送 TXWIDTH_16PORT ： PORT0/PORT1 を使った 16 ビットデータ転送 TXWIDTH_8PORT ： PORT0 による 8 ビットデータ転送
戻 値	正常に取得できれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	データ転送方向指定エラー
	-5	データ転送ビット幅指定エラー
	-6	ユニットがオープンされていない

書 式	Declare Function **dioStartHandShake** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer,ByVal nBytes As Long) As Long	
機 能	dioSetHandShakeMode()で設定されたモードに従って外部機器とのデータ送受信を開始します。本関数を呼び出す場合は、dioSetHandShakeMode()でハンドシェークモードを指定して下さい。 外部機器にデータを送信する場合は、予め dioStartWrite()によりユニットの内部 FIFO バッファに外部機器に送信するデータ格納しておく必要があります。 外部機器からデータを受信する場合はユニットは内部 FIFO バッファにデータを受信します。受信完了後、アプリケーションは dioGetNumInFifo()により FIFO に格納されたデータ個数を取得し、dioStartRead()により FIFO からデータをパソコン側に転送します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	nBytes	転送バイト数
戻 値	転送が正常に開始されれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ハンドシェークデータ転送開始エラー
	-5	ユニットがオープンされていない

書 式	Declare Function **dioStopHandShake** Lib "UsbDio32.dll" (ByVal UnitId As Integer) As Long	
機 能	dioStartHandShake()で開始したデータ転送を終了します。	
引 数	UnitId	ユニット ID
戻 値	停止が正常に行われれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	Declare Function **dioSetClockMode** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal TxDirection As Integer, ByVal TxWidth As Integer) As Long	
機 能	クロックに同期してデータ入出力を行うモードを設定します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	TxDirection	入出力方向指定 DIOUNIT_TO_EXTERNAL ： ユニットから外部機器に送信 EXTERNAL_TO_DIOUNIT ： 外部機器からユニットに受信
	TxWidth	入出力ビット幅指定 TXWIDTH_32PORT ： 全ポートを使った 32 ビット入出力 TXWIDTH_16PORT ： PORT0/PORT1 を使った 16 ビット入出力 TXWIDTH_8PORT ： PORT0 による 8 ビット入出力
戻 値	モード設定が正常に行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	データ転送方向指定エラー
	-5	データ転送ビット幅指定エラー
	-6	ユニットがオープンされていない

書 式	Declare Function **dioSetClock** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer, ByVal ClockTimer As Integer, ByVal TimeUnit As Integer) As Long	
機 能	ユニットの内部クロックに同期した入出力を行う場合のクロック周期を設定します。 クロック周期は ClockTimer×1.0049 秒の値になります。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	ClockTimer	時間を指定
	TimeUnit	0 を指定してください
戻 値	設定が正常に行われると 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	クロック設定エラー
	-5	単位指定エラー
	-6	設定可能範囲エラー
	-7	ユニットがオープンされていない

書 式	Declare Function **dioStartClock** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer,ByVal nBytes As Long) As Long	
機 能	dioSetClockMode()で設定されたモードと、dioSetClock()で設定された内部クロックに同期したデータ送受信を開始します。本関数を呼び出す場合は、dioSetClockMode()/dioSetClock()でクロック転送のパラメータを設定して下さい。 外部機器にデータを送信する場合は、予め dioStartWrite()によりユニットの内部 FIFO バッファに外部機器に送信するデータ格納しておく必要があります。 外部機器からデータを受信する場合はユニットは内部 FIFO バッファにデータを受信します。受信完了後、アプリケーションは dioGetNumInFifo()により FIFO に格納されたデータ個数を取得し、dioStartRead()により FIFO からデータをパソコン側に転送します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
	nBytes	転送バイト数
戻 値	転送が正常に開始されれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

書 式	Declare Function **dioStopClock** Lib "UsbDio32.dll" (ByVal hwnd As Long, ByVal UnitId As Integer) As Long	
機 能	dioStartClock()で開始したクロック同期転送を停止します。	
引 数	hwnd	ウィンドウハンドル
	UnitId	ユニット ID
戻 値	正常に停止されれば 0 が返されます。	
	エラーコード	エラーの内容
	-1	ユニット ID 番号取得エラー
	-2	ドライバー呼び出しエラー
	-3	ドライバーシグナル状態応答エラー
	-4	ユニットがオープンされていない

3.3.1 デジタル入出力基本制御サンプルプログラム

【1】Visual C/C++サンプル解説

本サンプルプログラムは 32 ポートを使って外部機器とオンオフ入出力を行う場合のサンプルプログラムになります。サンプルプログラムを起動すると、現在 USB に接続されたユニットの ID が「接続ユニット ID」に欄に列挙されます。都合上、計 4 台までしか表示されません。オンオフ入出力を行う前に、「オープンユニット ID」に ID を入力し「オープン」ボタンから明示的にユニットをオープンする必要があります。本サンプルプログラムは、この手順をわかりやすくするために敢えて上記のインターフェイスを設けています。実際にお客様が作成される場合には、この部分はプログラム内部に隠蔽されるのが一般的かと思います。

□ 最大 4 台までのユニットを列挙するプログラムは下記のように記述します。UnitFound には現在接続されているユニット台数が返されます。また、接続されているユニットのディップスイッチから ID 番号を読み込み pUnitId[4]に各々の ID 番号が格納されています。

```
USHORT        MaxUnit = 4;
USHORT        pUnitId[4];              // 最大４台数分確保しておく
int   UnitFound;
UnitFound = dioEnumUnit( hwnd, pUnitId, MaxUnit );
```

□ユニットオープン時のプログラムは下記のように記述します。正常にオープンできれば OpenUnitId には UnitId で指定した内容と同一のユニット ID が返されます。

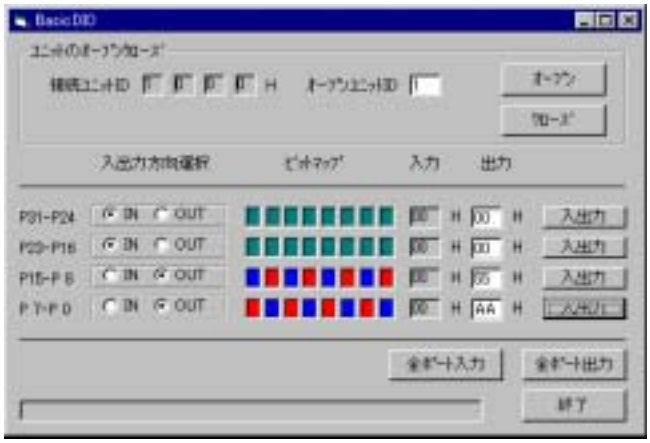
```
int OpenUnitId,UnitId;
UnitId = 0x01;
OpenUnitId = dioOpenUnit( hwnd, UnitId );

// プログラム終了時、オープンしたユニットは必ずクローズして終了します。
dioCloseUnit(OpenUnitId);
```

□ポート 0 に 0xAA を出力するプログラムは下記のように記述します。

```
// 入出力方向を出力に設定
Status = dioSetDirection( hwnd, UnitId, PORT0, 0x01 );
// 0xAA を出力する
Status = dioOutPort( hwnd, UnitId, PORT0, 0xAA );
```

【2】Visual BASIC サンプル解説

本サンプルプログラムは 32 ポートを使って外部機器とオンオフ入出力を行う場合のサンプルプログラムになります。サンプルプログラムを起動すると、現在 USB に接続されたユニットの ID が「接続ユニット ID」に欄に列挙されます。都合上、計 4 台までしか表示されません。オンオフ入出力を行う前に、「オープンユニット ID」に ID を入力し「オープン」ボタンから明示的にユニットをオープンする必要があります。本サンプルプログラムは、この手順をわかりやすくするために敢えて上記のインターフェイスを設けています。実際にお客様が作成される場合には、この部分はプログラムで内部的に処理されるのが一般的かと思います。



□ 最大 4 台までのユニットを列挙するプログラムは下記のように記述します。UnitFound には現在接続されているユニット台数が返されます。また、接続されているユニットのディップスイッチから ID 番号を読み込み pUnitId[4]に各々の ID 番号が格納されています。

```
Dim MaxUnit As Byte
Dim pUnitId(0 To 3) As Integer  'pUnitId の配列を４確保
' 最大 4 台までのﾕﾆｯﾄを列挙する
MaxUnit = 4
UnitFound = dioEnumUnit(hwnd, pUnitId(0), MaxUnit)
```

□ユニットオープン時のプログラムは下記のように記述します。正常にオープンできれば OpenUnitId には UnitId で指定した内容と同一のユニット ID が返されます。

```
Dim OpenUnitId As Long, UnitId As Long
'ｵｰﾌﾟﾝするﾕﾆｯﾄ ID を取得
UnitId = &H1
OpenUnitId = dioOpenUnit(hwnd, UnitId)
```

□ポート 0 に 0xAA を出力するプログラムは下記のように記述します。

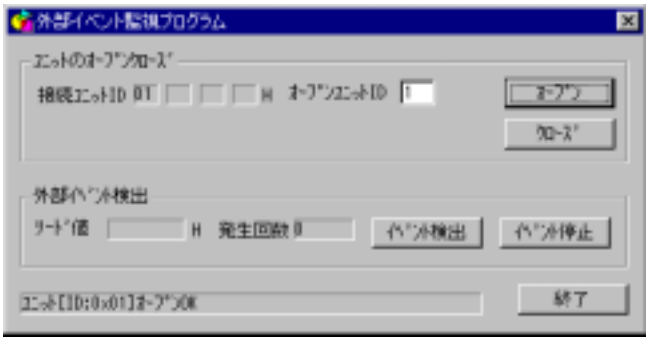
```
'入出力方向設定
Status = dioSetDirection(hwnd, UnitId, PORT0, &H01)
' ﾎﾟｰﾄ 1 にﾗｲﾄ
Status = dioOutPort(hwnd, UnitId, PORT0, &HAA)
```

3.3.2 外部機器イベント監視サンプルプログラム

【1】Visual C/C++サンプル解説

本サンプルプログラムは INT 端子に外部機器からのオンオフ信号を入力し、オンオフ状態変化を監視するサンプルプログラムになります。

接続されているユニットの列挙とユニットオープン処理に関しては、3.3.1 のサンプルプログラム「デジタル入出力基本制御」を参考にしてください。



□ イベント検出ボタンが押されると、最初に全てのポートの入出力方向を入力に設定しています。入力方向に設定されることによりドライバ内部では、イベント検出時全ポートをリードしその値をユーザ定義メッセージの第 4 パラメータとして返すことができます。この方向設定はアプリケーションの目的により、あるポートは出力に設定する場合もあります。方向設定はどのように設定されても問題ありませんが、ユーザ定義メッセージの第 4 パラメータとしてリードした値は出力に設定されたポートについては有効な値ではありません。

```
/* 方向は全て入力に設定しておく */
dioSetDirection( hwnd, UnitId, PORT3|PORT2|PORT1|PORT0, 0 );
/* 外部ｲﾍﾞﾝﾄﾓｰﾄﾞ設定 */
EventMode = 0x0;     // bit0：立ち上がりｴｯｼﾞ、bit1：外部ｲﾍﾞﾝﾄﾓｰﾄﾞｾｯﾄ
dioSetEventMode( hwnd, UnitId, EventMode );
dioStartInterrupt( hwnd, UnitId );
```

□ 外部イベントが発生するとユーザ定義メッセージ USBDIO_INTERRUPT_END がポストされます。ここでは単にイベント発生回数をカウントするだけですが、実際のアプリケーションでは警報出力のためのランプ点灯等の処理が想定されます。

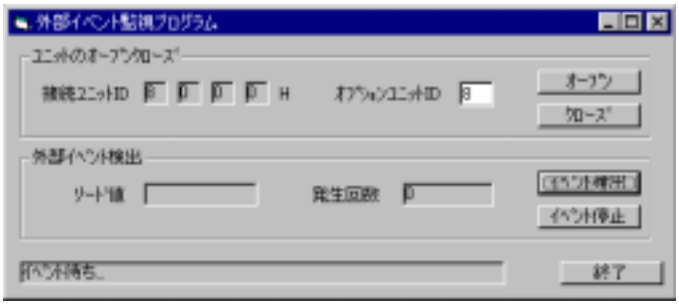
```
void Dlg_OnUserDefineMessage (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
      WORD EventCode = LOWORD(wParam);          // ﾒｯｾｰｼﾞのｺｰﾄﾞ
      WORD MsgPostUnitId = HIWORD(wParam);      // ﾕﾆｯﾄの ID
      switch ( EventCode ) {
      case USBDIO_INTERRUPT_END:
            sprintf( MsgBuf, "ｲﾍﾞﾝﾄ発生回数:%d", EventCount++ );
            SetDlgItemInt( hwnd, IDS_COUNTER, EventCount, FALSE );
            break;
      case USBDIO_INTERRUPT_START:              // ｲﾍﾞﾝﾄ待ち
      case USBDIO_INTERRUPT_UNITTIMEOUT:        // ｲﾍﾞﾝﾄ検出ﾀｲﾑｱｳﾄ
      case USBDIO_INTERRUPT_ABORT:              // ｲﾍﾞﾝﾄ検出ｱﾎﾞｰﾄ
      case USBDIO_INTERRUPT_ERROR:              // ｲﾍﾞﾝﾄ検出ｴﾗｰ終了
            break;
      }
 }
```

□ 外部イベント監視を終了する場合は、dioStopInterrupt()を呼び出します。正常に終了するとユーザ定義メッセージ USBDIO_INTERRUPT_ABORT がポストされます

```
dioStopInterrupt( UnitId );
```

【2】Visual BASIC サンプル解説

本サンプルプログラムは INT 端子に外部機器からのオンオフ信号を入力し、オンオフ状態変化を監視するサンプルプログラムになります。
接続されているユニットの列挙とユニットオープン処理に関しては、3.3.1 のサンプルプログラム「デジタル入出力基本制御」を参考にしてください。



□ イベント検出ボタンが押されると、最初に全てのポートの入出力方向を入力に設定しています。入力方向に設定されることによりドライバ内部では、イベント検出時全ポートをリードしその値をユーザ定義メッセージの第 4 パラメータとして返すことができます。この方向設定はアプリケーションの目的により、あるポートは出力に設定する場合もあります。方向設定はどのように設定されても問題ありませんが、ユーザ定義メッセージの第 4 パラメータとしてリードした値は出力に設定されたポートについては有効な値ではありません。

```
Dim EventMode As Long, OleHandle As Long
'方向は全て入力に設定しておく
dioSetDirection(hwnd, UnitId, PORT3 Or PORT2 Or PORT1 Or PORT0, &H0)
'外部ｲﾍﾞﾝﾄﾓｰﾄﾞ設定
EventMode = &H0;     ' bit0：立ち上がりｴｯｼﾞ、bit1：外部ｲﾍﾞﾝﾄﾓｰﾄﾞｾｯﾄ
dioSetEventMode( hwnd, UnitId, EventMode );
'ﾕｰｻﾞ定義ﾒｯｾｰｼﾞを受け取る MsgUsb10 ActiveX ｺﾝﾄﾛｰﾙのﾊﾝﾄﾞﾙ取得
OleHandle = MsgUsb.GetMboxWnd()
'ｲﾍﾞﾝﾄ検出開始（MsgUsb10 ｺﾝﾄﾛｰﾙのﾊﾝﾄﾞﾙを渡します）
dioStartInterrupt(OleHandle, UnitId );
```

□ 外部イベントが発生するとユーザ定義メッセージ USBDIO_INTERRUPT_END がポストされます。ここでは単にイベント発生回数をカウントするだけですが、実際のアプリケーションでは警報出力のためのランプ点灯等の処理が想定されます。

```
Private Sub MsgUsb_OnMsgPost(ByVal wParam As Long, ByVal lParam As Long)
    Dim EventCode As Long, MsgPostUnitId As Long
    EventCode = wParam Mod &HFFFF              ' 下位ﾜｰﾄﾞにはﾎﾟｽﾄされたﾒｯｾｰｼﾞのｺｰﾄﾞ
    MsgPostUnitId = wParam \ &HFFFF            ' 上位ﾜｰﾄﾞにはﾒｯｾｰｼﾞをﾎﾟｽﾄしたﾕﾆｯﾄの ID
    Select Case EventCode
        Case USBDIO_INTERRUPT_END           '外部ｲﾍﾞﾝﾄ検出
            EventCount = EventCount + 1
            IDS_COUNTER.Caption = Str(EventCount)
        Case USBDIO_INTERRUPT_START         'ｲﾍﾞﾝﾄ待ち
        Case USBDIO_INTERRUPT_UNITTIMEOUT   'ｲﾍﾞﾝﾄ検出ﾀｲﾑｱｳﾄ
        Case USBDIO_INTERRUPT_ABORT         'ｲﾍﾞﾝﾄ検出ｱﾎﾞｰﾄ
        Case USBDIO_INTERRUPT_ERROR         'ｲﾍﾞﾝﾄ検出ｴﾗｰ終了
    End Select
End Sub
```

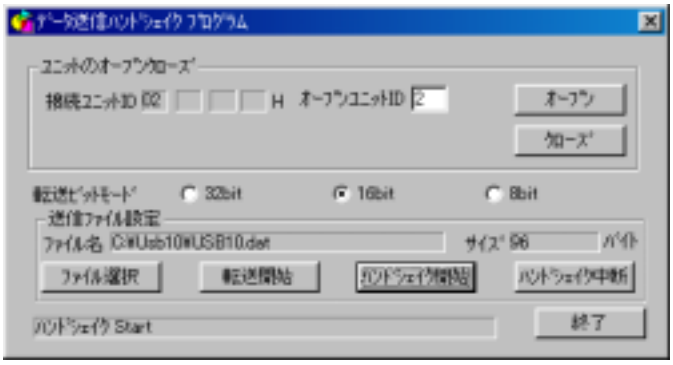
□ 外部イベント監視を終了する場合は、dioStopInterrupt()を呼び出します。正常に終了するとユーザ定義メッセージ USBDIO_INTERRUPT_ABORT がポストされます。

```
dioStopInterrupt(UnitId)
```

3.3.3 ハンドシェークデータ転送サンプルプログラム

【1】Visual C/C++サンプル解説

本サンプルプログラムは INT/STROBE 端子を使用しハンドシェークモードで外部機器にデータ送信を行うサンプルプログラムになります。
接続されているユニットの列挙とユニットオープン処理に関しては、3.3.1 のサンプルプログラム「デジタル入出力基本制御」を参考にしてください。



□ファイル選択ボタンで外部機器に送信するファイルを選択します。ファイルを選択するとパス名とサイズが表示されます。

□転送開始ボタンでファイルをパソコンから REX-USB10 に送信します。この時点では外部機器への送信は行われていません。正常に送信できれば USBDIO_WRITE_END がポストされます。

```
/* pSendData から格納されているファイルデータを 96 バイト転送します。 */
dioStartWrite( hwnd, UnitId, pSendData, 96 );
```

□ハンドシェイク開始ボタンで外部機器にファイルを送信いたします。正常に送信できれば USBDIO_HANDSHAKE_END がポストされます。この時のサンプルプログラムは下記に表記します。

```
/* 方向は全て出力に設定しておく */
dioSetDirection( hwnd, UnitId, PORT3|PORT2|PORT1|PORT0, 0x0f );
/* 外部ｲﾍﾞﾝﾄﾓｰﾄﾞ設定 */
EventMode = 0x2;      // bit0：立ち上がりｴｯｼﾞ、bit1：外部ｲﾍﾞﾝﾄﾓｰﾄﾞｾｯﾄ
dioSetEventMode( hwnd, UnitId, EventMode );
/* ユニットから外部機器に PORT0 による 8 ビットデータ送信 */
dioSetHandShakeMode( hwnd, UnitId, DIOUNIT_TO_EXTERNAL, TXWIDTH_8PORT);
dioStartHandShake( hwnd, UnitId, 96 );  //96ﾊﾞｲﾄ  ﾊﾝﾄﾞｼｪｲｸﾓｰﾄﾞで転送開始
```

□ ハンドシェイク中断ボタンで外部機器への送信を中断いたします。正常に終了すると USBDIO_HANDSHAKE_STOP がポストされます

```
dioStopHandShake ( UnitId );
```

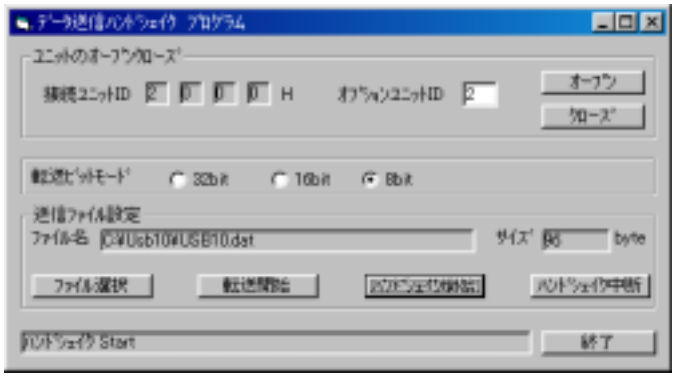
□ユーザー定義メッセージ処理

ハンドシェイクモードでデータ送信開始する前に REX-USB10 への転送が終了している事を確認してください。本プログラムではハンドシェイク開始ボタンを無効にし、USBDIO_ WRITE _END にポストされた時にハンドシェイク開始ボタンを有効にしています。

```
void Dlg_OnUserDefineMessage (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
      WORD EventCode = LOWORD(wParam);          // ﾒｯｾｰｼﾞのｺｰﾄﾞ
      WORD MsgPostUnitId = HIWORD(wParam);      // ﾕﾆｯﾄの ID
      switch ( EventCode ) {
      case USBDIO_ HANDSHAKE _START:            // ﾊﾝﾄﾞｼｪｲｸ送信開始
      case USBDIO_ HANDSHAKE _END:              // ﾊﾝﾄﾞｼｪｲｸ送信終了
      case USBDIO_ HANDSHAKE _STOP:             // ﾊﾝﾄﾞｼｪｲｸ送信途中停止
      case USBDIO_ HANDSHAKE _ERROR:            // ﾊﾝﾄﾞｼｪｲｸ送信ｴﾗｰ
      case USBDIO_ WRITE _START:                // REX-USB10 への送信開始
      case USBDIO_ WRITE _ERROR:                // REX-USB10 への送信ｴﾗｰ
          break;
      case USBDIO_ WRITE _END:                  // REX-USB10 への送信終了
          // ﾊﾝﾄﾞｼｪｲｸ開始ボタンを有効にする
          EnableWindow( GetDlgItem( hwnd, IDB_START ), TRUE );
          break;
      }
 }
```

【2】Visual BASIC サンプル解説

本サンプルプログラムはINT/STROBE端子を使用しハンドシェークモードで外部機器にデータ送信を行うサンプルプログラムになります。
接続されているユニットの列挙とユニットオープン処理に関しては､3.3.1 のサンプルプログラム「デジタル入出力基本制御」を参考にしてください。



□ファイル選択ボタンで外部機器に送信するファイルを選択します｡ファイルを選択するとパス名とサイズが表示されます。

□転送開始ボタンでファイルをパソコンから REX-USB10 に送信します｡この時点では外部機器への送信は行われていません。正常に送信できれば USBDIO_WRITE_END がポストされます。

```
Dim OleHandle As Long
'ﾕｰｻﾞ定義ﾒｯｾｰｼﾞを受け取る MsgUsb10 ActiveX ｺﾝﾄﾛｰﾙのﾊﾝﾄﾞﾙ取得
OleHandle = MsgUsb.GetMboxWnd()
'pSendData(0)から格納されているファイルデータを 96 バイト転送します。
dioStartWrite(OleHandle, UnitId, pSendData(0), 96 )
```

□ハンドシェイク開始ボタンで外部機器にファイルを送信いたします。正常に送信できれば USBDIO_HANDSHAKE_END がポストされます。この時のサンプルプログラムは下記に表記します。

```
Dim EventMode As Long, OleHandle As Long
'方向は全て出力に設定しておく
dioSetDirection(hwnd, UnitId, PORT3 Or PORT2 Or PORT1 Or PORT0, &Hf)
'外部ｲﾍﾞﾝﾄﾓｰﾄﾞ設定
EventMode = &H2;     ' bit0：立ち上がりｴｯｼﾞ、bit1：外部ｲﾍﾞﾝﾄﾓｰﾄﾞｾｯﾄ
dioSetEventMode( hwnd, UnitId, EventMode );
‘ ユニットから外部機器に PORT0 による 8 ビットデータ送信
dioSetHandShakeMode( hwnd, UnitId, DIOUNIT_TO_EXTERNAL, TXWIDTH_8PORT );
'ﾕｰｻﾞ定義ﾒｯｾｰｼﾞを受け取る MsgUsb10 ActiveX ｺﾝﾄﾛｰﾙのﾊﾝﾄﾞﾙ取得
OleHandle = MsgUsb.GetMboxWnd()
dioStartHandShake(OleHandle, UnitId, 96 )  ‘ 96ﾊﾞｲﾄ  ﾊﾝﾄﾞｼｪｲｸﾓｰﾄﾞで送信開始
```

□ ハンドシェイクを終了する場合は、dioStopHandShake ()を呼び出します。正常に終了すると USBDIO_HANDSHAKE_STOP がポストされます

```
dioStopHandShake ( UnitId )
```

□ユーザー定義メッセージ処理

ハンドシェイクモードでデータ送信開始する前に REX-USB10 への転送が終了している事を確認してください。本プログラムではﾊﾝﾄﾞｼｪｲｸ開始ボタンを無効にし、USBDIO_ WRITE _END にポストされた時にハンドシェイク開始ボタンを有効にしています。

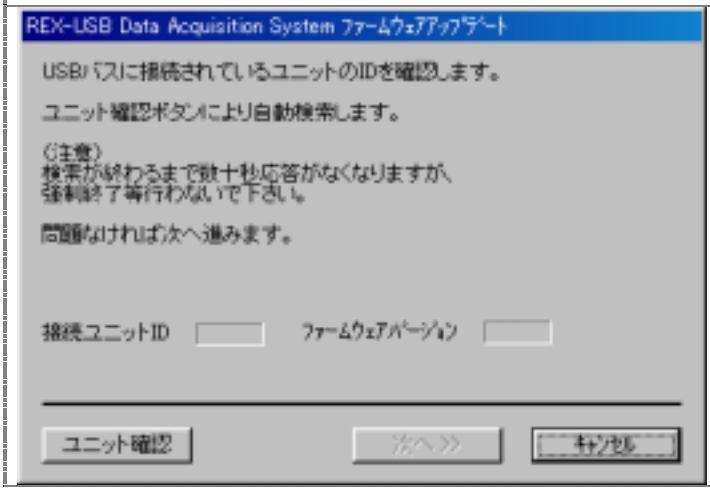
```
Private Sub MsgUsb_OnMsgPost(ByVal wParam As Long, ByVal lParam As Long)
    Dim EventCode As Long, MsgPostUnitId As Long
    EventCode = wParam Mod &HFFFF              ' 下位ﾜｰﾄﾞにはﾎﾟｽﾄされたﾒｯｾｰｼﾞのｺｰﾄﾞ
    MsgPostUnitId = wParam \ &HFFFF            ' 上位ﾜｰﾄﾞにはﾒｯｾｰｼﾞをﾎﾟｽﾄしたﾕﾆｯﾄの ID
    Select Case EventCode
        Case USBDIO_ HANDSHAKE _START          ‘ ﾊﾝﾄﾞｼｪｲｸ送信開始
        Case USBDIO_ HANDSHAKE _STOP           ‘ ﾊﾝﾄﾞｼｪｲｸ送信途中停止
        Case USBDIO_ HANDSHAKE _END            ‘ ﾊﾝﾄﾞｼｪｲｸ送信終了
        Case USBDIO_ HANDSHAKE _ERROR          ‘ ﾊﾝﾄﾞｼｪｲｸ送信ｴﾗｰ
        Case USBDIO_ WRITE _START              ‘ REX-USB10 への送信開始
            ‘ ﾊﾝﾄﾞｼｪｲｸ開始ボタンを有効にする
            IDB_START.Enabled = True
        Case USBDIO_ WRITE _END                ‘ REX-USB10 への送信終了
        Case USBDIO_ WRITE _ERROR              ‘ REX-USB10 への送信ｴﾗｰ
    End Select
End Sub
```

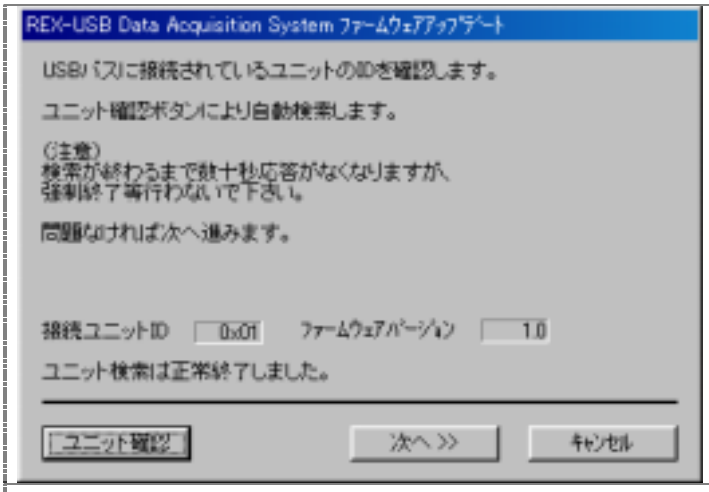
第 4 章 追加情報

この章では、USB の規格とファームウェア更新方法について説明します。

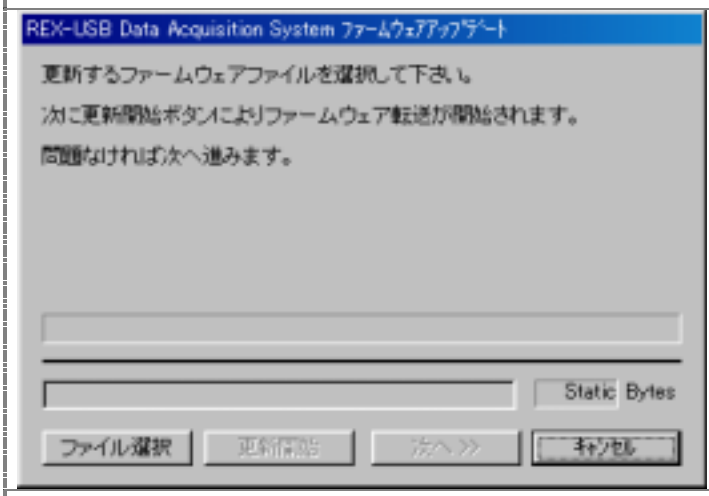
4-1. ユニットファームウェアアップデート

製品添付のファームウェアアップデートプログラム（FwUpWiz.Exe）により、ユニット内のファームウェアを書き換えて本ユニットの機能を拡張したり、不具合動作を修正することができます。弊社ホームページで公開されたファームウェアのバージョンアップ情報に基づいて新しいファームウェア（拡張子ＩＨＸファイル）をダウンロードして下さい。ファームウェアアップデートプログラム起動後、以下の手順で書き換えを行います。

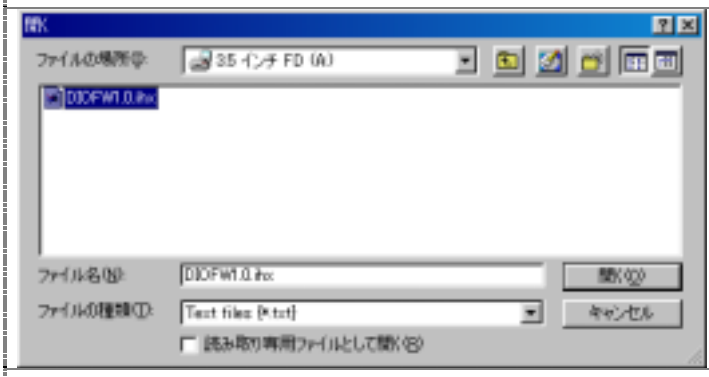
注意 ファームウェアアップデート実行中に USB のケーブルを抜いたり、パソコンをリセットしたりしないで下さい。万一、再起動後ユニットの認識ができなくなった場合は、弊社に送付していただき書き換えを行うことになります。	
	パソコンの USB ポートには本ユニット一台のみ接続して下さい。HUB 等を使用して複数の USB 機器を接続されている場合は、全て切り離しておいて下さい。注意事項をご確認後、問題がなければ次へ進んで下さい。 尚、ファームウェアアップデート完了後はパソコンの再起動が必要になります。起動中のプログラムは全て終了しておいて下さい。
	ユニット確認ボタンを押して、接続ユニット ID および現在ユニットに書き込まれているファームウェアのバージョンが表示されるのを待ちます。この間、多少時間がかかりますが、リセット等行わないようにして下さい。



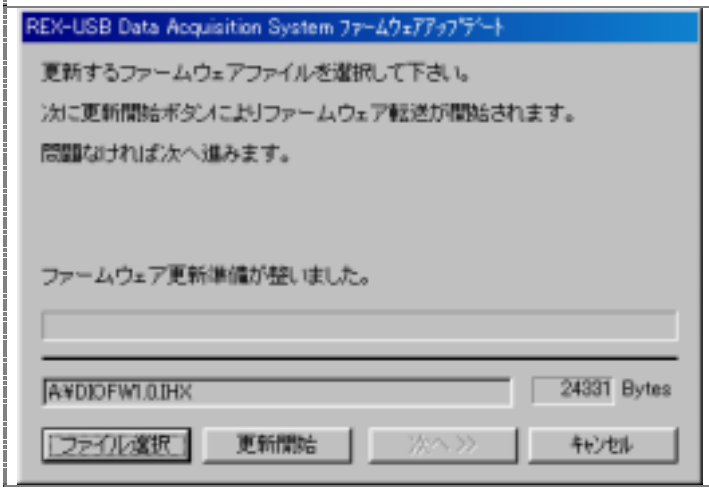
接続ユニット ID および現在ユニットに書き込まれているファームウェアのバージョンが表示されたら次へ進みます。



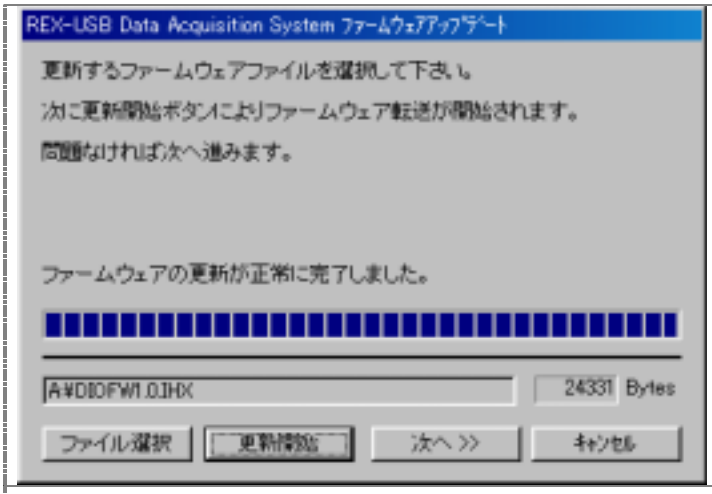
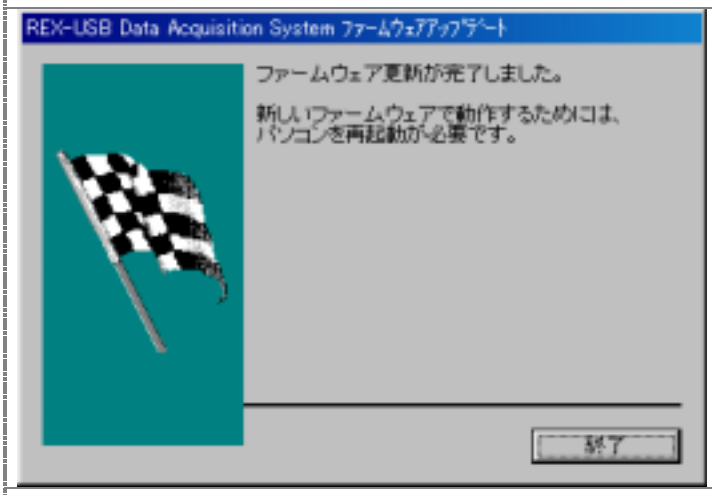
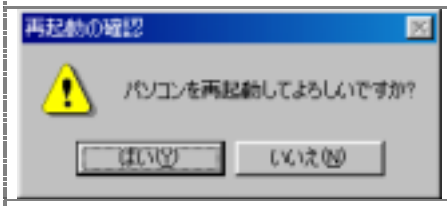
この段階では新しくアップデートするファームウェアファイル名が指定されていません。ファイル選択ボタンによりダウンロードしたファームウェアファイル名を指定します。



ファームウェアファイルを指定します。



更新開始ボタンにより書き込みを開始します。

	書き込みが完了したら次に進みます。
	以上でファームウェアアップデートが完了しました。終了ボタンを押してプログラムを終了します。
	ファームウェアアップデート後は一度再起動が必要になります。

4-2. USB 規格解説

本製品を使った計測システム構築を考える際、必ずしも USB 規格の詳細について理解する必要はありません。しかしながら、USB の得手不得手を理解しシステム構築における問題の有無を事前に検討しておくことは重要なことです。ここでは、Sep.23,1998 にリリースされた USB 規格 Revision1.1 に基づいて、要点をかいつまんで説明いたします。詳細かつ正確な情報が必要な方は、下記 URL から規格ドキュメントをダウンロードすることができますので、そちらを参照願います。

URL>>http://www.usb.org

4.2.1 USB インターフェイス概要

USB には、12Mbps フルスピード転送レートデバイスと 1.5Mbps ロースピード転送レートデバイス仕様のものがあります。本製品は転送レート 12Mbps 仕様になっています。

1 台の USB ホスト（パソコン）に最大 127 台までの USB デバイスを接続することができます。USB ケーブルは 2 本の電源ラインと 2 本のデータ転送ラインの計 4 本から構成されています。

USB ホストがトークンパケット方式によるバスの管理を行います。ホストからブロードキャストされたトークンパケットに記述されたアドレスに一致したデバイスが応答し、１対１の通信を行います。12Mbps のバンド幅は 1msec 間隔のタイムスロットに分割され、時分割方式でバスに接続された全てのデバイスの間で共有されます。

4.2.2 USB バストポロジー

USB の物理的な接続形態は、図 4.1 に示す HUB を使用したスター型に構成になります。USB バスには必ず 1 台のみの USB ホスト（通常パソコン）が存在します。ROOT HUB は USB ホスト内部に存在し、複数のポートを提供しています。

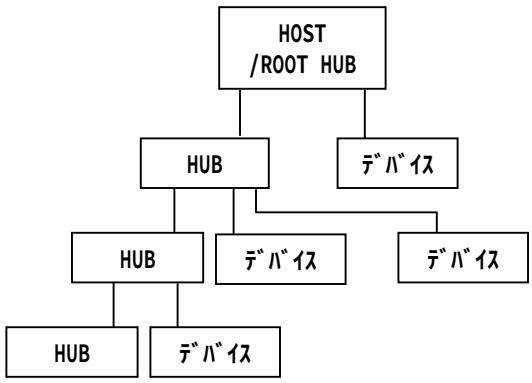


図 4.1 USB バストポロジー

4.2.3 USB 機械的･電気的仕様

USB のコネクタには、図 4.2 に示す series A と series B の二種類があります。series A コネクタは ROOT HUB である上位側のパソコンに接続され、series B は下位側の HUB もしくはデバイス側に接続されます。

Full Speed 仕様のケーブルは、20-28AWG の 2 本の電源ラインとツイストされた 28AWG の 2 本の信号線を外部シールドされた図 4.3 の仕様のものになります。V_{BUS} /GND が電源ライン、D+/D-が信号ラインです。

図 4.4 に示すように Full Speed 仕様の場合、D+ラインが 1.5KΩでプルアップされています。Low Speed 仕様の場合、D-ラインがプルアップされます。28-44Ωの直列抵抗によりインピーダンスマッチングが行われます。

USB ROOT HUB もしくはセルフパワード HUB からデバイスに供給可能な電源は限られています。V_{BUS} は 4.75V-5.25V、GND は 0V になります。USB バスから電源供給を受けて動作するデバイスをバスパワード、自分自身の電源で動作するデバイスをセルフパワードデバイスと呼びます。ROOT HUB もしくはセルフパワード HUB に接続されたバスパワードデバイスは、最大 500mA の電源供給を受けることができます。

デバイスを USB バスに装着した直後、ホストからデバイスが要求する電流が与えられるまで、デバイスは 100mA 以上の電流を消費することは許されません。

バスパワード HUB に装着されたデバイスは最大 100mA 以上の電流を消費することは許されません。

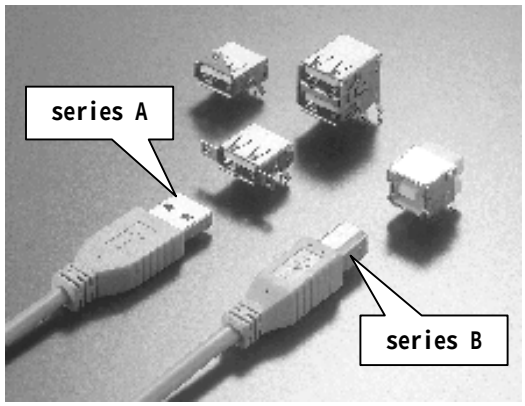


図 4.2 USB コネクタ

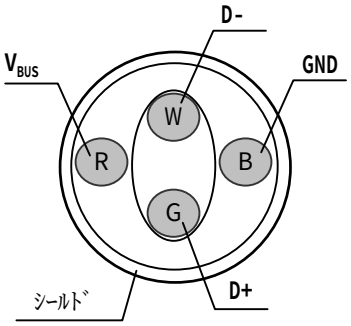


図 4.3 USB ケーブル

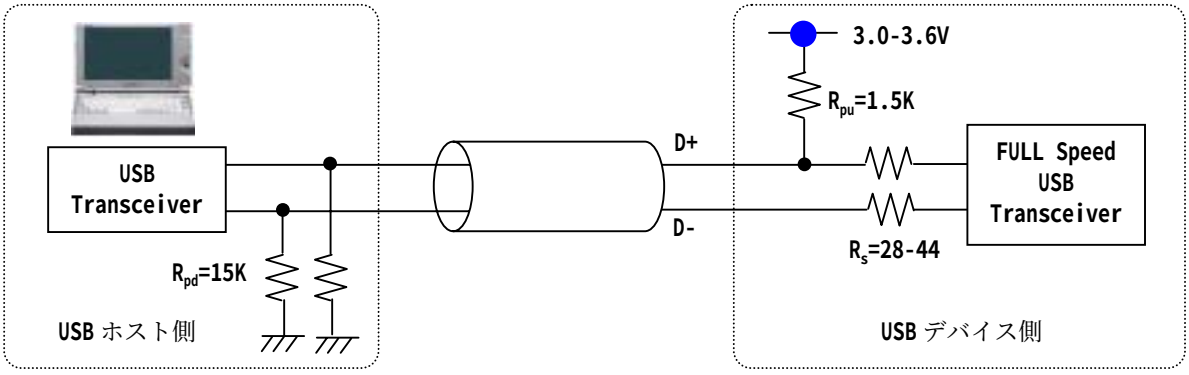


図 4.4 フルスピードデバイス接続回路

4.2.4 USB データフローモデル

図 4.5 に示す USB 通信階層モデルは、クライアントレイヤー・USB システムレイヤーと USB バスインターフェイスレイヤーの三層に分けられます。ホストとデバイス間の全ての通信データは実際には USB バスインターフェイスレイヤーを通過することになりますが、ホストとデバイス間の各層には論理的なコネクションが張られます。図 4.5 に示す階層モデルは、OSI の 7 層モデルをベースにしています。

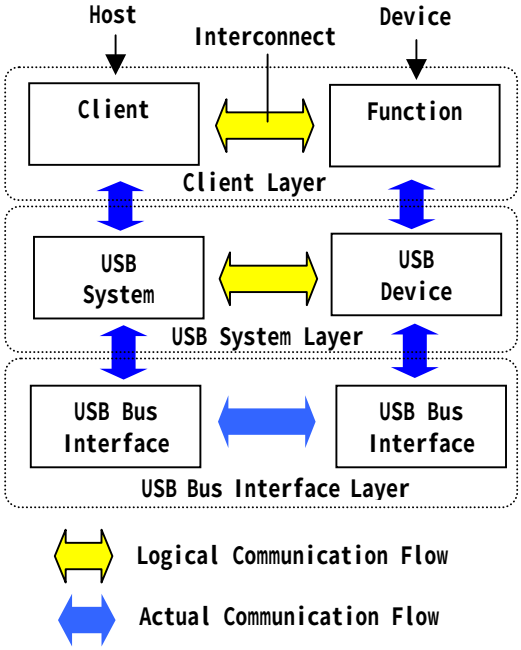


図 4.5 USB 通信階層モデル

このように階層化されることにより、パソコン上のアプリケーションソフトを作成する場合、デバイスの物理層を制御する部分を作成する必要はありません。USB バスインターフェイスレイヤーは、ホストとデバイスの物理的コネクションを制御します。次の層は、USB バスインターフェイスレイヤーにより提供されるサービスを使って、ホストとデバイス間のデータ転送の管理を行います。

ホスト側の USB システムレイヤーは、基本的に下記三つのドライバから構成されます。

- Host Software
- USB Driver(USBD)
- Host Controller Driver(HCD)

HCD はホストコントローラのハードウェア部に関するインターフェイスを USBD に提供しています。USBD と HCD 間のインターフェイスが Host Controller Driver Interface と呼ばれています。Microsoft Windows98 は Open Host Controller Interface(OHCI)と Universal Host Controller Interface(UHCI)をサポートしています。

クライアントレイヤーは、USB システムレイヤーから提供されるサービスを呼び出し、USB デバイスと直接やり取りを行う部分が記述されます。USBD の特定のクラスは Windows に提供されていますが、通常カスタムドライバを作成する必要があります。

4.2.5 USB デバイスアドレスとエンドポイント

USB のデバイスアドレスとエンドポイントは図 4.6 に示すパイプの概念により説明することができます。USB は一本の Big パイプで表され、127 本までの Small パイプで構成されます。各 Small パイプは USB デバイスに接続されます。USB トークンには 128 個のデバイスをアドレス可能な、7 ビットのアドレスフィールドがあります。その内、アドレス 000 0000B はデバイスが最初にバスに接続されたときに割り当てられるアドレスに使用されます。従って、USB デバイスは最大 127 台まで同一のバスに接続することができます。USB デバイスに接続された各々の Small パイプは、更に小さい Tiny パイプから構成されます。1 本の Small パイプは、最大 16 本の Tiny パイプで構成することが可能です。USB トークンの 4 ビットのエンドポイントフィールドで Tiny パイプ番号を指定することができます。USB でのデータ転送は、特定の Tiny パイプ（エンドポイント）を指定して行われます。

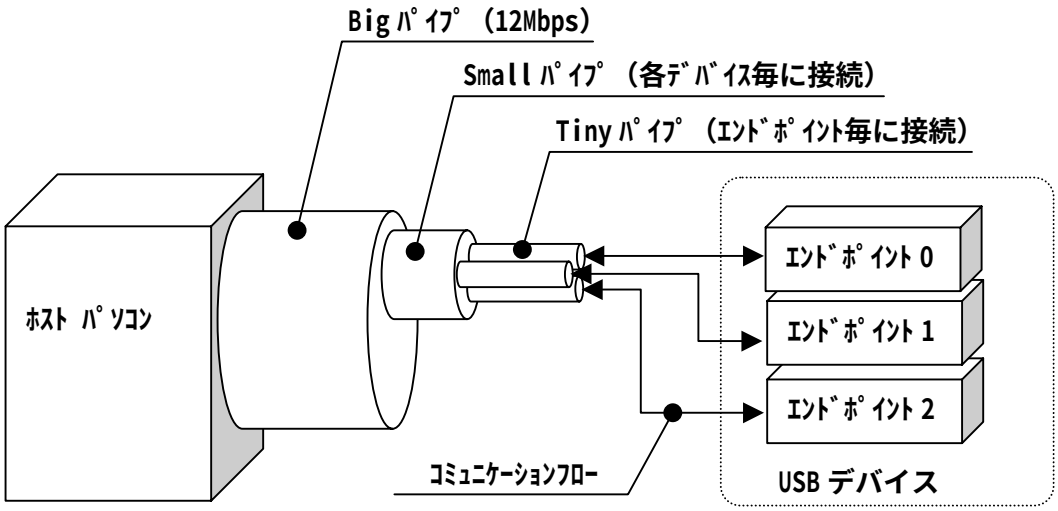


図 4.6 パイプの概念

4.2.6 USB 転送タイプとプロトコル

USB の転送タイプには、(1)コントロール転送、(2)インターラプト転送、 (3)バルク転送、(4)アイソクロナス転送の 4 種類があります。

（1）コントロール転送

デバイスに対するコンフィギュレーション、コマンド送信及びステータス取得を目的に使用されます。双方向のデータ転送を行うことができます。コントロール転送は図 4.7 に示すように、セットアップステージ・データステージ・ステータスステージの 3 ステージにより構成されます。このうち、データステージはオプションです。

セットアップステージでは、ホストはデバイスに対しリクエストを送信します。データステージでは、セットアップステージで示された方向のデータ転送が行われます。ステータスステージでは、デバイスはホストに対しハンドシェークを行います。

全ての USB デバイスは、コントロール転送のためのエンドポイント 0 を実装する必要があります。エンドポイント 0 はデバイスが USB バスに接続されたときの通信に使用されます。

コントロール転送では､CRC によりエラーチェックとエラー発生した場合はデータの再送が行われますので、正しいデータ転送が保証されます。

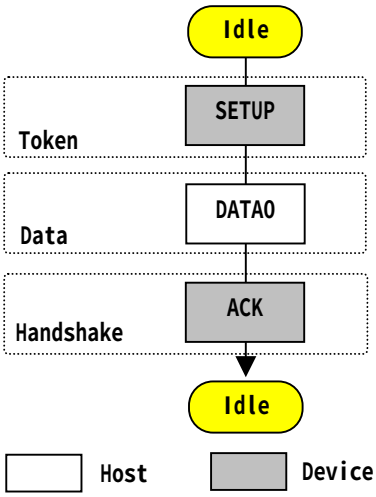


図 4.7 ｺﾝﾄﾛｰﾙｾｯﾄｱｯﾌﾟﾄﾗﾝｻﾞｸｼｮﾝ

（2）インターラプト転送

インターラプト転送はデバイスからホストに対する単一方向のデータ通信になります｡データ転送サイズが小さく、低頻度・勃発的なデータ転送に使用されます。インターラプト転送は、ホストが周期的にデバイスのエンドポイントに対し転送したいデータの有無を問い合わせます｡フルスピードデバイスの場合、1msec～255msec の範囲でポーリング間隔を指定することができます。エラー訂正は行われますので、転送データは保証されます。

インターラプト転送はトークンフェーズで IN トークンしか存在しないことを除いて、バルク転送と同じようなトランザクションが行われます。デバイスはホストから IN トークンを受け取ると、データもしくは NAK・STALL パケットを返すことができます。デバイス側でイベントがペンディングされている場合は、データを返信します。送信すべきデータがない場合は、NAK ハンドシェークパケットを返します。ホストは返信されたデータに対しエラーがない場合は、ACK ハンドシェークパケットを返します。エラーが検出された場合は、何も返信しません。

（3）バルク転送

バルク転送は単一方向、双方向のどちらの転送も行うことができます。データ転送量が大きく、必ず正しいデータ転送が必要な場合に使用されますが、アイソクロナス転送と異なり転送レートは保証されません。転送レートは、USB バスの未使用バンド幅となります。

図 4.8 に示すように、バルク転送はアイソクロナス転送と同じようなトランザクション仕様ですが、データフェーズの後にハンドシェークフェーズが続きます。ハンドシェークフェーズにより正しくデータ転送が行われたどうかの確認が行われます。間違いなくデータ転送が行われた場合は、データを受け取った側のホストまたはデバイスが ACK ハンドシェークパケットを送信します。デバイスは ACK 以外に NAK ハンドシェークパケットもしくは STALL ハンドシェークパケットを返すことができます。NAK ハンドシェークパケットは、ホストからのリクエストに対し現在デバイスが応答できないことを示します。STALL ハンドシェークパケットは、デバイス側でエラー発生しているためにホスト側で何らかの調整が必要であることを示します。NAK・STALL ハンドシェークパケットは必ずデバイス側から送信されることに注意して下さい。

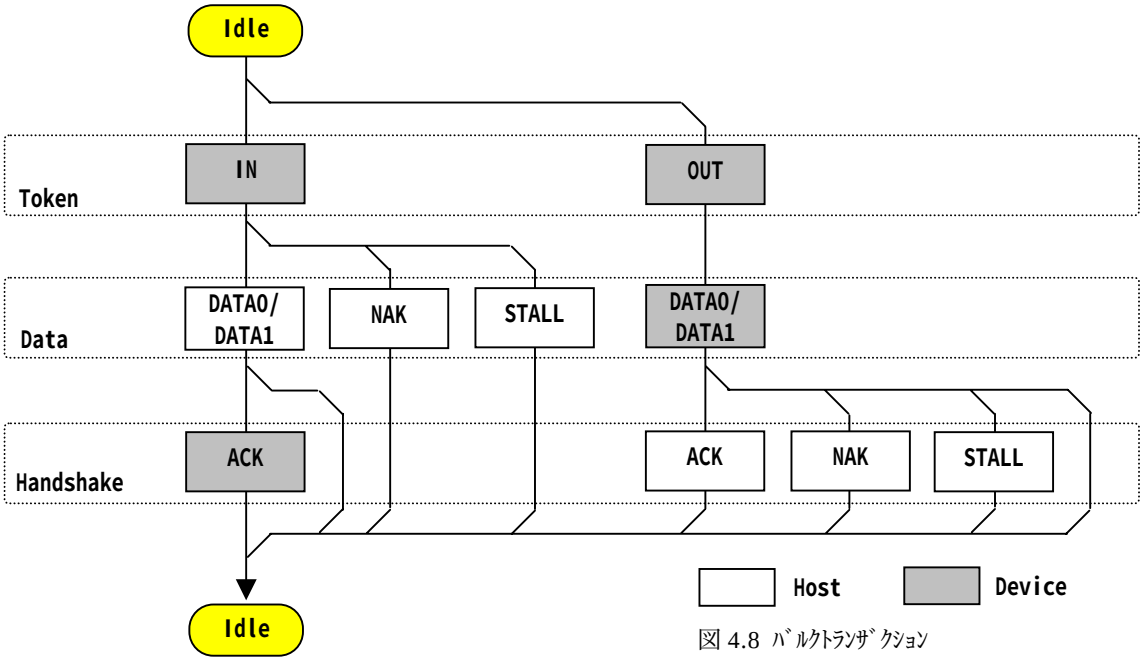


図 4.8 ﾊﾞﾙｸﾄﾗﾝｻﾞｸｼｮﾝ

（4）アイソクロナス転送

アイソクロナス転送は単一方向、双方向のどちらの転送も行うことができます。一定の転送レートが保証されますが、データ転送エラーチェックは行われません。

1msec の USB フレーム単位で送信可能な最大パケットサイズは、1023 バイトです。従って、アイソクロナス転送での最大通信レートは 8.184Mbps となります。

アイソクロナス転送はトークンフェーズとデータフェーズから構成され、1msec 間隔の 1 フレームの間に完了します。ホストからの IN トークンに対し、デバイスはデータをホストに送信します。OUT トークンが送られた場合は、その直後にデバイスに対しデータが送信されます。ハンドシェークフェーズは存在しません。

4.2.7 USB パケットフォーマット仕様

USB で使用されるパケットは、トークン・SOF・データ・ハンドシェークパケットの 4 種類に分類することができます。

（1）トークンパケット

SYNC	PID	ADDR	ENDP	CRC

USB のトランザクションは常にホストがイニシエータとなり、上記のトークンパケットが最初に送信されます。

SYNC　全てのパケットはシンクロナイゼーションフィールドで始まります。シンクロナイゼーションフィールドは 8 ビットの長さで、データ受信回路でローカルクロックと、送信データとの同期を確立するために使用されます。

PID　SYNC フィールドの直後に、8 ビット長の PID (Packet Identifier)フィールドが続きます。上位 4 ビットが PID で下位 4 ビットは PID をビット反転したもので、チェック用に使用されます。

PID_0	PID_1	PID_2	PID_3	$\wedge PID_0$	$\wedge PID_1$	$\wedge PID_2$	$\wedge PID_3$

下表に示すように、[PID_1:PID_0]により転送パケットタイプを区分します。Token パケットは[0:1]、Data パケットは[1:1]、Hnadshake パケットは[1:0]、Special パケットは[0:0]となります。[PID_3: PID_2]により更に細分化されたパケット識別名を表します。

PID ﾀｲﾌﾟ	PID 名	PID[3:0]	説明
Token	OUT	0001B	Address + endpoint number in H-to-D transaction
	IN	1001B	Address + endpoint number in D-to-H transaction
	SOF	0101B	Start-of-frame marker and frame number
	SETUP	1101B	Address + endpoint number in H-to-D transaction for SETUP to a control pipe
Data	DATA0	0011B	Data packet PID even
	DATA1	1011B	Data packet PID odd
Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Rx device cannot accept data or Tx device cannot send data
	STALL	1110B	Endpoint is halted or a cintrol pie request is not supported
Special	PRE	1100B	Host-issued preamble. Enables downstream bus traffic to low-speed devices.

ADDR　7 ビット長のアドレスフィールドは、パケット送信先デバイスのアドレスを表します。ホストは USB バスに初めてデバイスが接続された時のエニュメレーションプロセスで、各デバイスにユニークなアドレスを割り当てます。

ENDP　4 ビットのエンドポイントフィードは、デバイスのどの Tiny パイプと通信するかを表すエンドポイント番号が指定されます。

CRC　CRCs(Cyclic Redundancy Checks)フィールドにはエラー訂正コードが記載されます。

（2）SOF パケット

SYNC	PID	Frame Number	CRC

Start of Frame(SOF)パケットは、ホストから 1.00±0.05msec 間隔でブロードキャストされます。SYNC フィールド・PID フィールドの後に、11 ビット長の Frame Number フィールドが続きます。

（3）データパケット

SYNC	PID	DATA	CRC

USB ではホストがバスの管理を行い、デバイスはホストからのリクエストに応答する形態を取ります。ホストはデバイスからの情報が必要な場合、ホストはデバイスのアドレスとエンドポイントを指定して IN トークンパケットを送信します。指定されたデバイスは上記のデータパケットで、PID に DATA0 もしくは DATA1 を指定してホストにデータを送信します。
ホストからデータ送信する必要のある場合は、OUT トークンが送信されます。その後すぐに、ホストからデータパケットが送信されます。

（4）ハンドシェークパケット

SYNC	PID

ハンドシェークパケットは PID のみになります。PID には ACK,NACK もしくは STALL がセットされます。

（5）スペシャルパケット

Special Preamble(PRE)パケットはホストが Low Speed デバイスと通信する際に送信されます。

4.2.8 USB データ転送の実例

4 種類の USB 転送タイプの中でコントロール転送を例にして、実際に USB バス上でどのようにデータの受け渡しが行われるかアナライザーでパケットをキャプチャーして解説します。

最初にコントロール転送で使用されるパケットの仕様について多少理解しておく必要があります。コントロール転送は、セットアップ・データ（オプション）・ステータスのステージから構成されます。セットアップステージでは、下表 4-1 のデバイスリクエストがホストから送られ、次に続くデータステージのフォーマットが定義されます。デバイスリクエストには規格で定められたフォーマットの標準デバイスリクエストと、ベンダが独自に定めるベンダリクエストがあります。

表.4-1 デバイスリクエスト

ｵﾌｾｯﾄ	ﾌｨｰﾙﾄﾞ	ｻｲｽﾞ	値	記述	
0	bmRequestType	1	Bitmap	ﾘｸｴｽﾄのﾀｲﾌﾟを示す	
				b7	ﾃﾞｰﾀ方向 0:ﾎｽﾄ←ﾃﾞﾊﾞｲｽ 1: ﾃﾞﾊﾞｲｽ→ﾎｽﾄ
				b6,b5	ﾘｸｴｽﾄﾀｲﾌﾟ 0:標準ﾘｸｴｽﾄ 1:ｸﾗｽﾘｸｴｽﾄ 2:ﾍﾞﾝﾀﾞｰﾘｸｴｽﾄ 3:予約
				b4…b0	受信側ｲﾝﾀｰﾌｪｲｽ 0:ﾃﾞﾊﾞｲｽ 1:ｲﾝﾀｰﾌｪｲｽ 2:ｴﾝﾄﾞﾎﾟｲﾝﾄ 3:その他 4 以降:予約
1	bRequest	1	Value	特定のﾘｸｴｽﾄ	
2	wValue	2	Value	bRequest に従った値	
4	wIndex	2	Index/Offset	bRequest に従ったｲﾝﾃﾞｯｸｽまたはｵﾌｾｯﾄ	
6	wLength	2	Count	ﾃﾞｰﾀｽﾃｰｼﾞのﾊﾞｲﾄ数	

表.4-2 標準デバイスリクエスト

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B 10000001B 10000010B	0:GET_STASUS	0	Zero Interface Endpoint	2	Device,Interface, or Endpoint Status
00000000B 00000001B 00000010B	1:CLEAR	Feature Selector	Zero Interface Endpoint	0	None
00000000B 00000001B 00000010B	3:SET_FEATURE	Feature Selector	Zero Interface Endpoint	0	None
00000000B	5:SET_ADDRESS	Device Address	0	0	None
10000000B	6:GET_DESCRIPTOR	Descriptor Type and Descriptor Index	0 or Language ID	Descriptor Length	Descriptor
00000000B	7:SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Language ID	Descriptor Length	Descriptor
10000000B	8:GET_CONFIGURATION	0	0	1	Configuration Value
00000000B	9:SET_CONFIGURATION	Configuration Value	0	0	None
10000001B	10:GET_INTERFACE	0	Interface	1	Alternate Setting
00000001B	11:SET_INTERFACE	Alternative Setting	Interface	0	None
10000010B	12:SYNCH_FRAME	0	Endpoint	2	Frame Number

USB バスにデバイスを接続すると、ホストはバスに接続されたデバイスの列挙を行います。最初に標準デバイスリクエスト GET_DESCRIPTOR により、デバイスデスクリプタ情報を取得します。この後、バスリセットが行われ、標準デバイスリクエスト SET_ADDRESS によりデバイスにアドレスを割り当てます。新しいアドレスを割り当てた後、ホストは GET_DESCRIPTOR コマンドによりデスクリプタ情報を取得します。引き続き、同じ GET_DESCRIPTOR コマンドによりデバイスのコンフィグレーションデスクリプタ情報を取得します。この情報をもとに、最後にホストはデバイスに SET_CONFIGURATION コマンドによりデバイスのコンフィグレーションを行います。

下記キャプチャーデータは、上記デバイス列挙プロセスの最初の GET_DESCRIPTOR 及び SET_ADDRESS コマンド部分になります。ホストは 1msec インターバルで SOF パケットを送信しています。これにより、パケット№21 と№25 の間は 1msec であることが分かります。パケット№22 はホストから最初に送られるセットアップパケットで、デバイスアドレスは 0x00 でエンドポイント 0 に送られていることが分かります。パケット№23 の 2 バイト目 bRequest:0x06 及び wValue:0x0001 より GET_DESCRIPTOR コマンドでデバイスデスクリプタの取得が要求されています。デバイスはこの要求を受けて、パケット№24 で ACK 応答を行っています。ここまでが、セットアップステージになります。

№21	Sync(00000001) SOF(0xA5) Frame #(0x320) CRC5(0x07)
№22	Sync(00000001) SETUP(0xB4) ADDR(0x00) ENDP(0x0) CRC5(0x08)
№23	Sync(00000001) DATA0(0xC3) DATA(80 06 00 01 00 00 40 00) CRC16(0xBB29)
№24	Sync(00000001) ACK(0x4B)
№25	Sync(00000001) SOF(0xA5) Frame #(0x321) CRC5(0x18)

パケット№27 の In トークンに応答して、デバイスはパケット№28 で 18 バイトのクリプタ情報を送信しています。ホストは、パケット№29 で正常に受信できたことを示す ACK 応答を行っています。ここまでが、データステージです。

№26	Sync(__000001) SOF(0xA5) Frame #(0x322) CRC5(0x1A)
№27	Sync(00000001) IN(0x96) ADDR(0x00) ENDP(0x0) CRC5(0x08)
№28	Sync(00000001) DATA1(0xD2) DATA(12 01 00 01 FF 00 FF 40 84 05 10 A0 01 00 01 02 03 01) CRC16(0x7FD6)
№29	Sync(00000001) ACK(0x4B)
№30	Sync(00000001) SOF(0xA5) Frame #(0x323) CRC5(0x05)

パケット№32 からコントロール転送の最後のステージであるステータスステージになります。ホストはデバイスに対し、パケット№32 の OUT トークンとパケット№33 の NULL パケットを送信しています。デバイスはパケット№34 で ACK 応答を行っています。

№31	Sync(00000001) SOF(0xA5) Frame #(0x324) CRC5(0x1B)
№32	Sync(00000001) OUT(0x87) ADDR(0x00) ENDP(0x0) CRC5(0x08)
№33	Sync(00000001) DATA1(0xD2) DATA() CRC16(0x0000)
№34	Sync(00000001) ACK(0x4B)
№35	Sync(00000001) SOF(0xA5) Frame #(0x325) CRC5(0x04)
№36	RESET(Start of Reset)
№37	RESET(13.92 milliseconds)

引き続きホストは、パケット№127 で示す SET_ADDRESS コマンドを送信します。2 バイト目の 0x05 は SET_ADDRESS コマンドを示し、wValue：0x02 はホストがデバイスに割り当てた新しいアドレスになります。標準デバイスリクエスト SET_ADDRESS にはデータステージはありませんので、パケット№130 から№132 のステータスステージで完了します。これまでホストはデバイスアドレスを 0x00 としていましたが、今後は新しく割り当てた 0x02 を使用します。

№125	Sync(00000001) SOF(0xA5) Frame #(0x389) CRC5(0x0A)
№126	Sync(00000001) SETUP(0xB4) ADDR(0x00) ENDP(0x0) CRC5(0x08)
№127	Sync(00000001) DATA0(0xC3) DATA(00 05 02 00 00 00 00 00) CRC16(0xD768)
№128	Sync(00000001) ACK(0x4B)

№129	Sync(00000001) SOF(0xA5) Frame #(0x38A) CRC5(0x08)
№130	Sync(00000001) IN(0x96) ADDR(0x00) ENDP(0x0) CRC5(0x08)
№131	Sync(00000001) DATA1(0xD2) DATA() CRC16(0x0000)
№132	Sync(00000001) ACK(0x4B)

4-3. REX-USB10 デスクリプタ仕様

本製品のデスクリプタ仕様を参考までに記載します。尚、実際の製品とは一部異なることがあります。

(1) Device Descriptor

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	18	ディスクリプタのサイズ
bDescriptorType	1	01H	ディスクリプタのタイプ
bCdUSB	2	0001H	USB Spec. 1.0
bDeviceClass	1	FFH	クラスコード(ベンダ)
bDeviceSubClass	1	00H	サブクラスコード
bDeviceProtocol	1	FFH	プロトコルコード(ベンダ)
wMaxPacketSize	1	64	エンドポイント 0 の最大パケット数
idVender	2	0584H	ベンダ ID
idProduct	2	A010H	プロダクト ID
bcdDevice	2	0100H	デバイスリリース番号 Ver 1.00
iManufacturer	1	1	製造者を表すストリングディスクリプタへのインデックス
iPuroduct	1	2	製品を表すストリングディスクリプタへのインデックス
iSerialNumber	1	3	デバイスの製造番号を表すストリングディスクリプタへのインデックス
bNumConfiguration	1	1	構成可能な数

(2) Configuration Descriptor

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	9	ディスクリプタのサイズ
bDescriptorType	1	02H	ディスクリプタのタイプ
wTotalLength	2	0027H	構成全体の長さ（Table3.2～Table3.5）
bNumInterface	1	1	構成の持つインタフェイスの数
bConfigurationValue	1	1	SetConfiguration でこの構成を選択するための引数値
bConfiguration	1	4	構成を表すストリングディスクリプタへのインデックス
bAttribute	1	A0H	構成の特性
bMaxPower	1	4BH	最大バス電力消費量を 2mA 単位で指定:300mA（注 1）

(3) Interface Descriptor

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	9	ディスクリプタのサイズ
bDescriptorType	1	04H	ディスクリプタのタイプ
bInterfaceNumber	1	0	このインタフェイスを表すインデックス(ORG. 0)
bAlternateSetting	1	0	SetInterface で代替え設定を選択するための引数値
bNumEndpoint	1	3	インタフェイスの持つエンドポイント数
bInterfaceClass	1	FFH	ベンダ
bInterfaceSubClass	1	FFH	サブクラスコード
bInterfaceProtocol	1	FFH	ベンダ固有
iInterface	1	4	インタフェイスを表すストリングデータへのインデックス

(4) Endpoint Descriptor

Endpoint１ディスクリプタ構造体

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	7	ディスクリプタのサイズ
bDescriptorType	1	05H	ディスクリプタのタイプ
bEndpointAddress	1	01H	
bmAttribute	1	02H	属性 バルク
wMaxPacketSize	2	0040H	最大パケットサイズ
bInterval	1	0	ポーリング間隔を mS 単位で指定

Endpoint2 ディスクリプタ構造体

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	7	ディスクリプタのサイズ
bDescriptorType	1	05H	ディスクリプタのタイプ
bEndpointAddress	1	82H	
bmAttribute	1	02H	属性 バルク
wMaxPacketSize	2	0040H	最大パケットサイズ
bInterval	1	0	ポーリング間隔を mS 単位で指定

Endpoint3 ディスクリプタ構造体

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	7	ディスクリプタのサイズ
bDescriptorType	1	05H	ディスクリプタのタイプ
bEndpointAddress	1	83H	
bmAttribute	1	03H	属性:割り込み
wMaxPacketSize	2	0040H	最大パケットサイズ
bInterval	1	255	ポーリング間隔を mS 単位で指定

(5) String Descriptor

INDEX 0 (LANG ID)

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	4	ディスクリプタのサイズ
bDescriptorType	1	03H	ディスクリプタのタイプ
wLANGID[0]	2	0904H	LANGID

INDEX 1 (製造者)

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	36	ディスクリプタのサイズ
bDescriptorType	1	03H	ディスクリプタのタイプ
bString	34		'R',0,'A',0,'T',0,'O',0,'C',0,'S',0,'y',0,'s',0, 't',0,'e',0,'m',0,'s',0,',',0,'I',0,'n',0,'c',0, '.',0,0

INDEX 2 (製品名)

項目	ﾊﾞｲﾄ数	値	説明
bLength	1	24	ディスクリプタのサイズ
bDescriptorType	1	03H	ディスクリプタのタイプ
bString	22		'U',0,'S',0,'B',0,'_',0,'D',0,'I',0,'O',0,'_',0, 'I',0,'&',0,'L',0,

第 5 章 オプションアイソレーションユニット

この章では、REX-USB10 のオプションの説明します。

5-1. REX-IPI16 製品概要

REX-IPI16 は外部機器からの無電圧接点入力をフォトカプラによって電気的にアイソレートして TTL レベル信号に変換するアイソレーション入力ユニットです。

リードスイッチや近接センサーからの信号入力インターフェイスとして使用できます。各ポートの信号状態を LED で確認できます。

入力電圧を 5V，12V，24V に変更可能です。パソコンの電源電圧（5V）以上の外部機器と接続される場合には、使用する必要があります。

入力ポート数 16．ワンタッチスクリューの螺子端子を使用し、また外形も REX-USB10 とスタックするために、同一サイズとしてあり非常に使いやすくなっております。

5-2. REX-IPO16 製品概要

REX-IPO16 はパソコン側からの 16 ポートの TTL レベル信号をフォトカプラによって電気的にアイソレートして外部出力を行うアイソレーション出力ユニットです。

出力は、最大 300V、165mA の駆動能力があり、電磁弁やパワーリレーの駆動制御、スイッチの ON/OFF 制御のインターフェイスとして使用できます。各ポートの信号状態を LED で確認できます。L レベル信号出力時に LED が点灯します。

出力ポート数 16. ワンタッチスクリューの螺子端子を使用し、また外形も REX-USB10 とスタックするために、同一サイズとしてあり非常に使いやすくなっております。

5-3. RCL-TRM48 製品概要

製品添付のオス側コネクタと外部機器を結線し、そのコネクタを本体メス側コネクタに挿入して信号の入出力を行うワンタッチスクリューターミナルユニットです。REX-USB10 の信号を全てワンタッチスクリューの螺子端子に出すためのユニットです。

また外形も REX-USB10 とスタックするために、床面積は同一サイズとしてあり非常に使いやすくなっております。