USBX-A0800
ユーザーズマニュアル

テクノウェーブ株式会社

目次

1. はじめに

このたびは弊社多機能 I/O ユニットをご購入頂き、まことにありがとうございます。以下をよくお読みになり、安全にご使用いただけますようお願い申し上げます。

□ 安全にご使用いただくために

製品を安全にご利用いただくために、以下の事項をお守りください。

危険

これらの注意事項を無視して誤った取り扱いをすると人が死亡または重傷を負う危険が差し迫って生じる可能性があります。

- 引火性のガスがある場所では使用しないでください。爆発、火災、故障の原因となります。

警告

これらの注意事項を無視して誤った取り扱いをすると人が死亡または重傷を負う可能性があります。

- 水や薬品のかかる可能性がある場所では使用しないでください。火災、感電の原因となります。
- 結露の発生する環境では使用しないでください。火災、感電の原因となります。
- 定格の範囲内でご使用ください。火災の原因となります。

注意

これらの注意事項を無視して誤った取り扱いをすると人が傷害を負う可能性があります。また物的損害の発生が想定されます。

- 濡れた手で製品を扱わないでください。故障の原因となります。
- 異臭、過熱、発煙に気がついた場合は、ただちに電源を切断し USB ケーブルを抜いてください。
- 製品を改造しないでください。

□ その他の注意事項

- 本製品は一般民製品です。特別に高い品質・信頼性が要求され、その故障や誤動作が直接人命を脅かしたり、人体に危害を及ぼす恐れのある機器に使用することを前提としていません。本製品をこれらの用途に使用される場合は、お客様の責任においてなされることになります。
- お客様の不注意、誤操作により発生した製品、パソコン、その他の故障、及び事故につきましては弊社は一切の責任を負いませんのでご了承ください。
- 本製品または、付属のソフトウェアの使用による要因で生じた損害、逸失利益または第三者からのいかなる請求についても、当社は一切その責任を負えませんのでご了承ください。

□ マニュアル内の表記について

本マニュアル内ではハードウェアの各電気的状態について下記のように表記いたします。

表 1　電気的状態の表記方法

表記	状態
“ON”	電流が流れている状態、スイッチが閉じている状態、オープンコレクタ(オープンドレイン)出力がシンク出力している状態。
“OFF”	電流が流れていない状態、スイッチが開いている状態、オープンコレクタ(オープンドレイン)出力がハイインピーダンスの状態。
“Hi”	電圧がロジックレベルのハイレベルに相当する状態。
“Lo”	電圧がロジックレベルのローレベルに相当する状態。

また、数値について「0x」、「&H」 、「H’」はいずれもそれに続く数値が 16 進数であることを表します。“0x10”、“&H1F”、“H’20”などはいずれも 16 進数です。

関数・構造体名

本文で関数名を表記する場合、C/C++、Visual Basic®、Visual Basic for Applications の名称に従い″*TWXA_Open()* ″のように表記します。C#の場合、これと対応する関数は *Techw.IO* 名前空間の *TWXA* クラスのスタティックメンバ関数で″*Techw.IO.TWXA.Open()* ″となります。構造体名についても同様です。

関数の宣言を示す場合、C/C++、Visual Basic (.NET 以後)、Visual Basic for Applications (以下 VBA)、C# の順で、それぞれの言語における関数宣言が記載されます(表 2)。C# の場合は、名前空間とクラス名は省略して記述しています。

表 2　関数宣言の表記例

言語	関数宣言
C/C++	TW_STATUS TWXA_Open(TW_HANDLE *phDev, long Number, long Opt)
VB	Function TWXA_Open(ByRef phDev As System.IntPtr, ByVal Number As Integer, ByVal Opt As TWXA_OPEN_OPT) As Integer
VBA	Function TWXA_Open(ByRef phDev As Long, ByVal Number As Long, ByVal Opt As TWXA_OPEN_OPT) As Long
C#	STATUS Open(out System.IntPtr phDev, int Number, OPEN_OPT Opt)

引数の入力候補

各関数の引数の中には、入力できる値が限定されていて、ある定数を入力することが適当なものがあります。そのような場合、各開発環境の入力支援機能(インテリセンス)を十分活用できるよう、言語毎に異なった定数や列挙型を定義しています。

表 3は *TWXA_Open()* 関数の *Opt* 引数の入力候補の一部です。引数の入力候補は表のように各言語別に記述方法が記載されます。

Windows、Visual Studio、Visual C++、Visual Basic、Visual C# は米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

″C/C++″と書かれた行は C および C++で使用できる記述方法です。この値は*#define* で定義された定数です。

″C++″と書かれた行は C++で使用できる記述方法です。定数専用に宣言されたクラスのスタティックメンバになっています。Visual Studio でこの定数を入力する場合、最初に″TWXA::″と入力すると画面に入力候補が表示されますので、定数を選択して入力を行ってください。

″VB/VBA″と書かれた行は Visual Basic と VBA で使用可能な記述方法です。この場合、関数の引数自体が列挙型となっており定数は列挙子です。

″C#″と書かれた行は C#で使用可能な記述方法です。この場合も Visual Basic 同様に関数の引数が列挙型となっています。名前空間は省略して記述しています。

表 3 引数の入力候補の例

言語	値	説明
C/C++	TWXA_ANY_DEVICE	制御できるデバイスであればインタフェースや製品タイプを問わずに接続します。
C++	TWXA::OPEN_OPT::ANY_DEVICE	
VB/VBA	TWXA_OPEN_OPT.ANY_DEVICE	
C#	TWXA.OPEN_OPT.ANY_DEVICE	
C/C++	TWXA_IF_USB	ホストインタフェースが USB のデバイスに接続します。
C++	TWXA::OPEN_OPT::IF_USB	
VB/VBA	TWXA_OPEN_OPT.IF_USB	
C#	TWXA.OPEN_OPT.IF_USB	
C/C++	TWXA_IF_LAN	ホストインタフェースが LAN のデバイスに接続します。
C++	TWXA::OPEN_OPT::IF_LAN	
VB/VBA	TWXA_OPEN_OPT.IF_LAN	
C#	TWXA.OPEN_OPT.IF_LAN	

Null 値

関数の引数の中には Null 値(空値)を要求するものがあります。本文中で Null 値と表記した場合、各言語での対応する記述方法は表 4 のようになります。

表 4 Null 値

言語	記述方法
C/C++	NULL
VB	Nothing
VBA	vbNullString
C#	null

2. 製品概要

□ 特徴

『USBX-A0800』(以下、製品またはデバイス)は多機能 I/O ユニットです。USB を通じてパソコンから、AD コンバータ、シリアル通信などの機能を制御できます。

また、製品に内蔵されたマイコン用のプログラム開発もサポートされていますので、機能をカスタマイズすることにより、高いリアルタイム性が要求される処理にも対応可能です。

- **AD コンバータ** – 非絶縁入力の 16 ビット AD コンバータを 8 チャンネル搭載しています。AD コンバータは完全な 8 チャンネル同時サンプリングを、最大 200KS/sec[1]で行うことが可能です。また、入力範囲として-5～+5V、-10～+10V を選択できます。
- **シリアル通信**[2] – RS-485 用シリアルポートを 1 チャンネル、RS-232C の信号レベルで通信できるシリアルポートを 1 チャンネル備えています。
- 制御用 API は DLL モジュールで提供され、Visual C++® や Visual Basic® 、Visual C#® で作成された Windows 上のアプリケーションプログラムから制御できます。また、ナショナルインスツルメンツ社の LabVIEW™ にも対応していますので、グラフィカルな開発環境でのプログラミングも可能です。
- 内蔵マイコンのプログラミングはエル・アンド・エフ社の Yellow IDE(YCH8)、イエロースコープ(YSH8)に対応し、ソースコードレベルでのデバッグが可能です。
- 製品は付属の取付具を使用することで 35mmDIN レールにワンタッチで着脱できます。

1 使用 API により変換速度は変化します。

2 シリアルポートは OS 上から仮想 COM ポートとして制御することはできません。専用 API でのアクセスとなります。

LabVIEW は、National Instruments Corporation の商標です。

□ 製品の利用方法

パソコンからの制御

製品は専用の制御用 API を通して接続したパソコンから制御することができます。この制御用 API は「TWXA.dll」というファイルで提供され、TWXA ライブラリと呼びます。

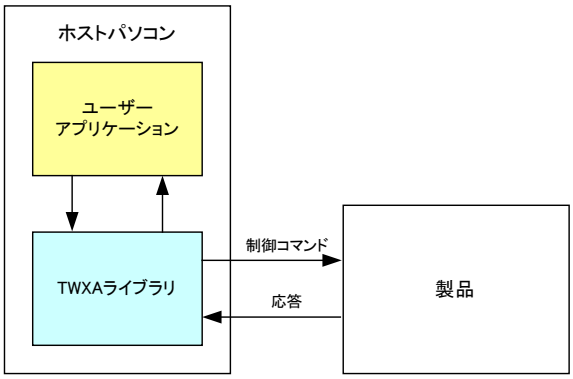

図 1　ホストパソコンからの制御

表 5 のプログラミング言語に対しては、予め開発に必要となるヘッダーファイルやモジュールファイルが提供されており、作成したプログラムから TWXA ライブラリの各関数を呼び出し、製品を制御することができます。また多くの場合、その他のプログラミング言語についても、その言語に合わせた定義ファイルを作成していただくことで製品を利用することが可能になります。

表 5　開発用ファイルが提供される言語

開発言語	開発環境/製品
C	Visual Studio など
C++	Visual Studio など
Visual Basic	Visual Studio など
Visual Basic for Applications	Microsoft Office
C#	Visual Studio など

また、LabVIEW については TWXA ライブラリの各関数と対応した VI ライブラリが用意されており、プログラム内に組み込むことで製品を制御することができます。

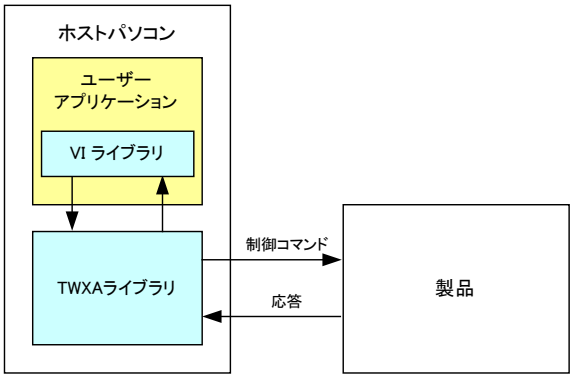

図 2　LabVIEW での利用

ファームウェアの開発

TWXA ライブラリの各関数は図 3 のように製品に組み込まれたファームウェア [3]に独自の制御コマンドを送信することで製品を制御します。最初から製品に組み込まれているこのファームウェアのことをシステムファームと呼びます。

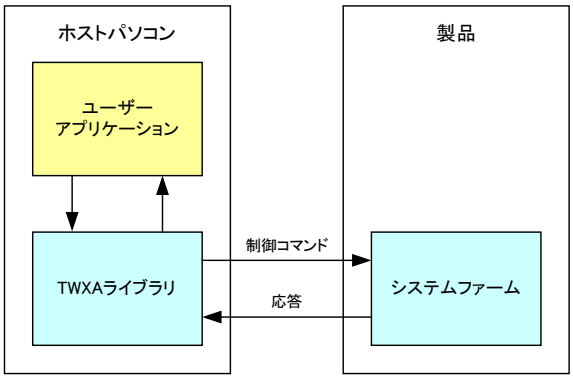

図 3　ホストパソコンからの制御

製品ではファームウェアをユーザーが開発し、動作をカスタマイズする仕組みがサポートされています。これにより、パソコンからのコマンド制御では実現が困難なリアルタイム性が要求される処理や、基本機能では提供されないユーザー独自の機能追加が可能です。このユーザーカスタムのファームウェアのことをユーザーファーム と呼びます(図 4)。ユーザーファームの開発言語は C 言語です。詳細は別紙「X-A0800 ユーザーファーム開発マニュアル」を参照してください。

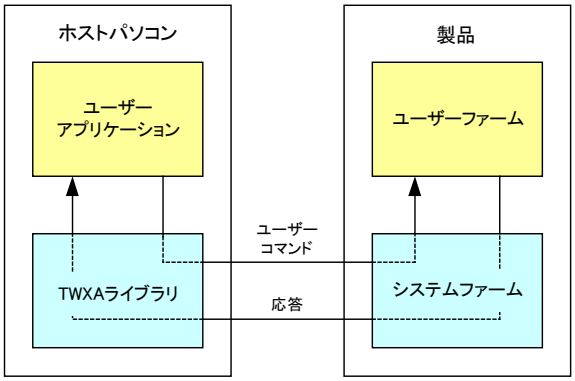

図 4　ユーザーファームの追加

[3] パソコン上で動作するプログラムやソフトウェアと区別するために、製品内蔵のマイコンで動作するプログラムのことをファームウェア、または単にファームと呼びます。

□ 関連ドキュメント

本マニュアルでは製品の設定、ハードウェア、パソコン用プログラムの開発方法を中心に説明しています。TWXA ライブラリ関数の詳細や、VI ライブラリ、ユーザーファームの開発などについては表 6 にあげるドキュメントを参照してください。

表 6　製品関連ドキュメント

ドキュメント名	内容	ファイル名
USBX-A0800 ユーザーズマニュアル（本マニュアル）	基本事項、ハードウェア、専用ライブラリによるホストパソコンからの制御方法など	USBX-A0x0x.pdf
TWXA ライブラリ 関数リファレンス	専用ライブラリの各関数の説明	TWXALibrary.pdf
X-A0800 ユーザーファーム開発マニュアル	ユーザーファーム（製品内蔵マイコン用プログラム）の開発方法	X-A0x0xUserFirm.pdf
VI ライブラリヘルプファイル	LabVIEW 用ライブラリの使用方法	（VI ライブラリをインストールすることで[スタート]メニューに追加されます）

3. 製品仕様

□ 仕様

表 7　共通仕様

項目	仕様	備考
寸法	96(W)×60(D)×34(H)[mm]	ゴム足、端子台、DIN レール取付具含まず
重量	250[g]	付属品含まず
電源電圧	5[VDC]	USB から供給
消費電流	最大 500[mA]	
動作温度範囲	0～50[℃]	
フラッシュメモリの プログラム保持年数	10[年]	
インタフェース	USB2.0	HiSpeed 対応
対応 OS	Windows XP, Vista, 7, 8, 8.1, 10	

表 8　AD コンバータ仕様

項目		仕様	備考
入力チャンネル数		8 チャンネル	
入力方式		シングルエンド入力	チャンネルを 2 つ使用することで差動入力が可能
入力レンジ		±5[V]、または、±10[V]	
分解能		16[bit]	
入力インピーダンス		標準 1M[Ω]	
リファレンス精度		標準-0.04～+0.02[%]	条件：全温度範囲
リファレンス温度係数		標準±10[ppm/℃]	
変換時間		4[μsec]	
非直線性誤差		最大±2[LSB]	条件：全温度範囲
絶対精度	±5[V]レンジ	標準±12[LSB]	条件：全温度範囲
	±10[V]レンジ	標準±6[LSB]	条件：全温度範囲

表 9　シリアルポート仕様

チャンネル	項目	仕様	備考
0	信号レベル	RS-485 準拠	
	適合コネクタ	PHR-3(日本圧着端子製造)	
	通信方式	半二重	
	同期方式	調歩同期式(フロー制御なし)	
	ビットレート	300～38400[bps]	
	内蔵終端抵抗	120[Ω]	ON/OFF 切替え可能
1	信号レベル	RS-232C 準拠	
	適合コネクタ	SBA20-03HG/SB20-03HG (日本オートマチックマシン)	
	通信方式	全二重	
	同期方式	調歩同期式(フロー制御なし)[4]	
	ビットレート	300～38400[bps]	

[4] RTS,DTR は出力されませんので接続する機器の仕様によっては通信できない場合があります。

□　外形寸法

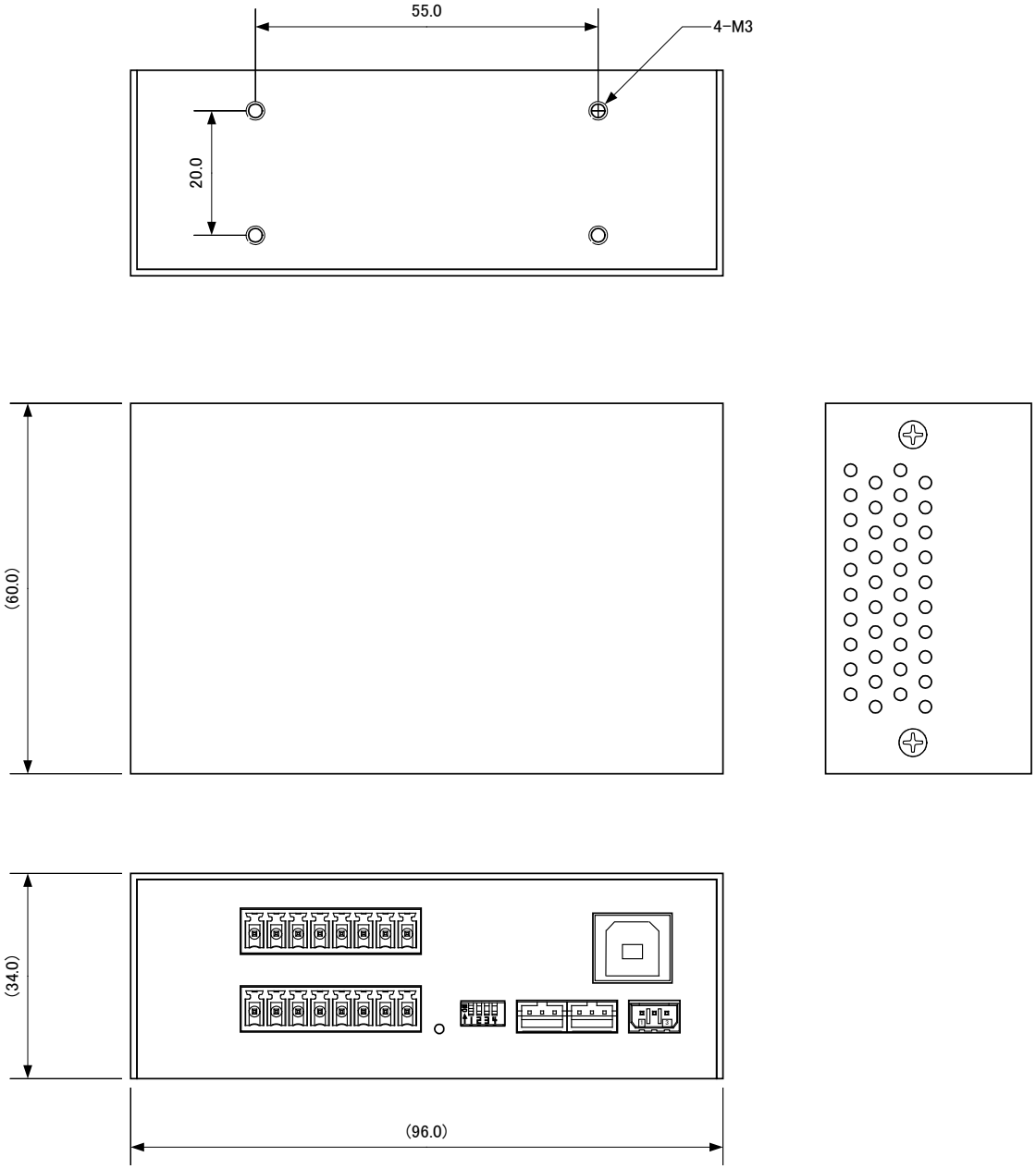

図 5　外形寸法図

□ USBX-A0800 各部の名称と説明

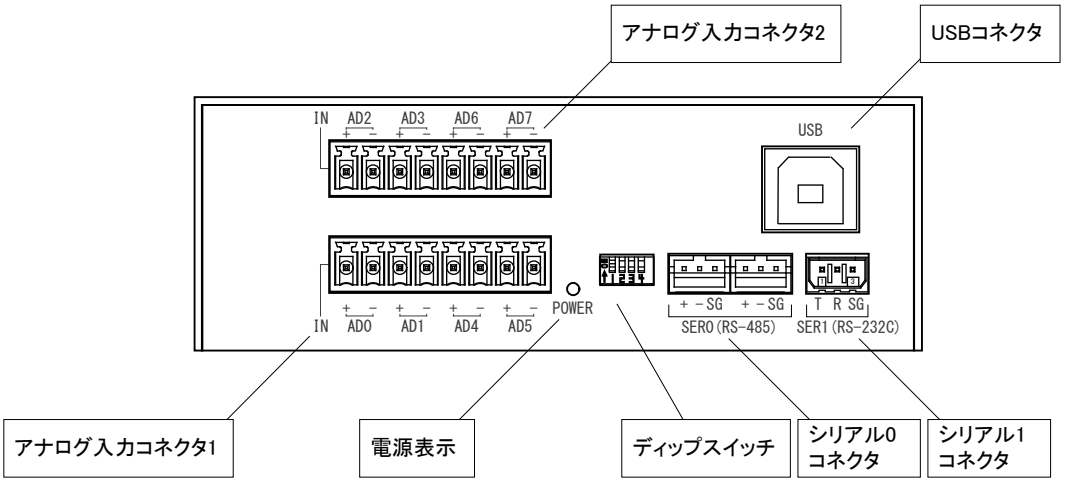

図 6 USBX-A0800 各部の名称

電源表示
電源がオンになると LED が点灯します。

USB コネクタ
パソコンの USB ポートに接続します。

ディップスイッチ
製品の動作設定を行います。詳細は 14 ページを参照してください。

シリアル 0 コネクタ
RS-485 による通信に使用します。2 つのコネクタは内部で並列に接続されています。適合コネクタは「PHR-3」(日本圧着端子製造)です。

シリアル 1 コネクタ
RS-232C による通信に使用します。また、ファームウェアの開発の際にはデバッガとの通信ポートとして使用します。適合コネクタは「SBA20-03HG」または「SB20-03HG」(日本オートマチックマシン)です。

アナログ入力コネクタ 1
アナログ信号の入力端子です。適合コネクタは「EC350RL」(DINKLE)です。

アナログ入力コネクタ 2
アナログ信号の入力端子です。適合コネクタは「EC350R」(DINKLE)です。

□ ディップスイッチ

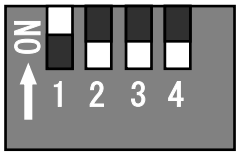

図 7 ディップスイッチ

表 10 ディップスイッチ

番号	説明
1	常に"ON"で使用します。
2	通常は"OFF"で使用します。製品をフラッシュ書換えモードで起動するとき"ON"にします。
3	ライブラリ関数からフラッシュメモリへの書込みを許可する場合に"ON"にします。
4	シリアル 0(RS-485)の終端抵抗を有効にする場合"ON"にします。

4. 使用準備

□ DIN レール取付具の固定

DINレール取付具は図 8の向きで製品に取り付けます。製品は図 9の向きになるように固定してください。

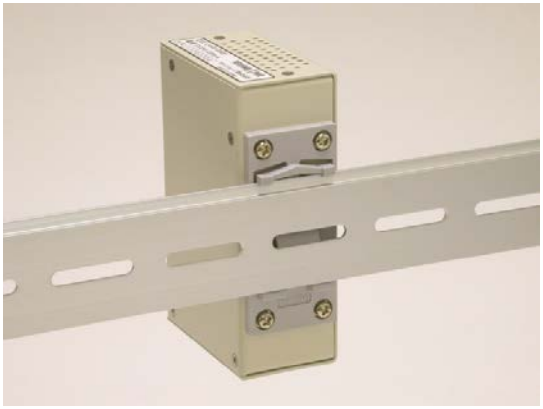

図 8　DIN レール取付具の取付け

図 9　DIN レールへの固定

- 設置時は側面の換気孔をふさがないように注意してください。

□ 端子台への配線

付属するコネクタ端子台のスクリューを緩め、電線(図 10 参照)を挿入し再びスクリューを締めて固定します。

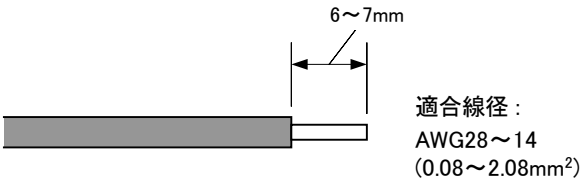

図 10　電線の加工

□ ドライバのインストール

ドライバは付属 CD-ROM に納められています。

表 11　ドライバファイルの格納フォルダ

使用 OS	ドライバファイルの格納フォルダ
Windows XP, Vista	CD の「¥HS_DRIVER¥XP-Vista」フォルダ
Windows 7, 8, 8.1, 10	CD の「¥HS_DRIVER¥7-10」フォルダ

管理者のアカウントでログオンし、上記のフォルダから「setup.exe」を起動してください。

- 製品をパソコンに接続する前にドライバのセットアップを行ってください。

Windows 10　の場合

① 「setup.exe」を起動すると、次のようなウィンドウが表示されますので[はい](または[許可])を選択します。

図 11　Windows 10 のドライバインストール画面（1）

② インストールプログラムが起動しますので、画面の指示に従ってインストールを行います。

③ 下のような画面が表示されたら[インストール]ボタンを押してインストールを続行します。

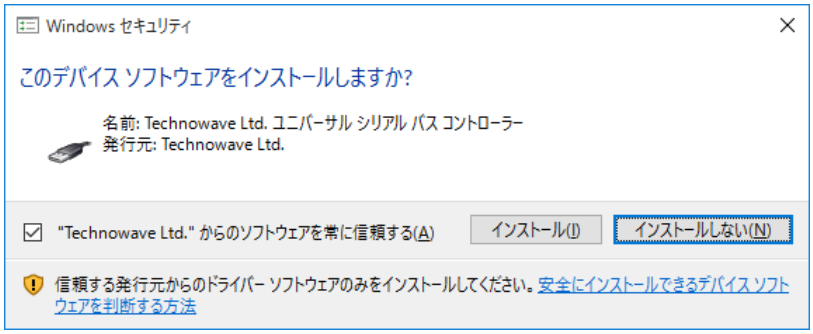

図 12　Windows 10 のドライバインストール画面（2）

④ 次のような画面が表示されますので[完了]ボタンを押してください

図 13　Windows 10 のドライバインストール画面（3）

⑤ デバイスを USB ケーブルでパソコンに接続します。図 18 のように「デバイス マネージャ」の画面に「USBM3069-H USB Device」と表示されれば、ドライバが正しくインストールされています。

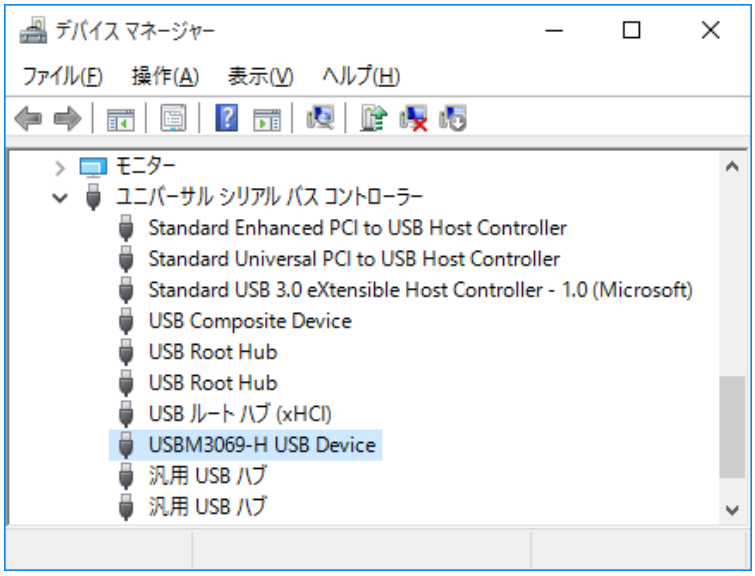

図 14　Windows 10 のドライバインストール確認

- 「デバイスマネージャ」を表示するには[スタート]メニューを右クリックし、表示されたリストから［デバイス マネージャ]をクリックしてください。

Windows 7 の場合

① 「setup.exe」を起動すると、次のようなウィンドウが表示されますので[はい](または[許可])を選択します。

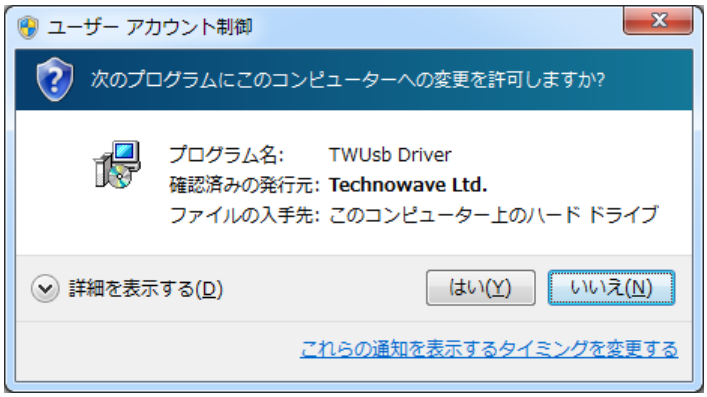

図 15　Windows 7 のドライバインストール画面（1）

② インストールプログラムが起動しますので、画面の指示に従ってインストールを行います。

③ 下のような画面が表示されたら[インストール]ボタンを押してインストールを続行します。

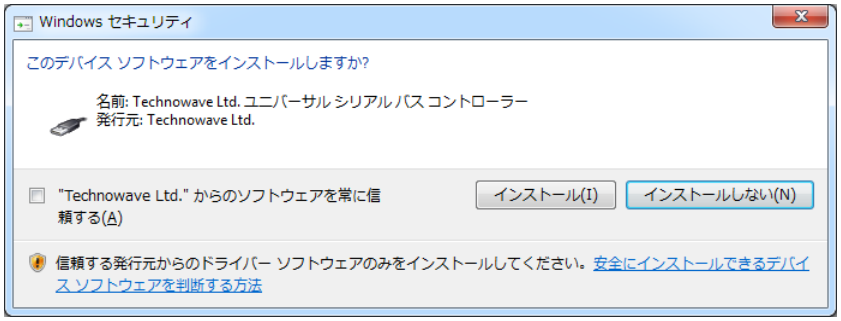

図 16　Windows 7 のドライバインストール画面（2）

④ 次のような画面が表示されますので[完了]ボタンを押してください

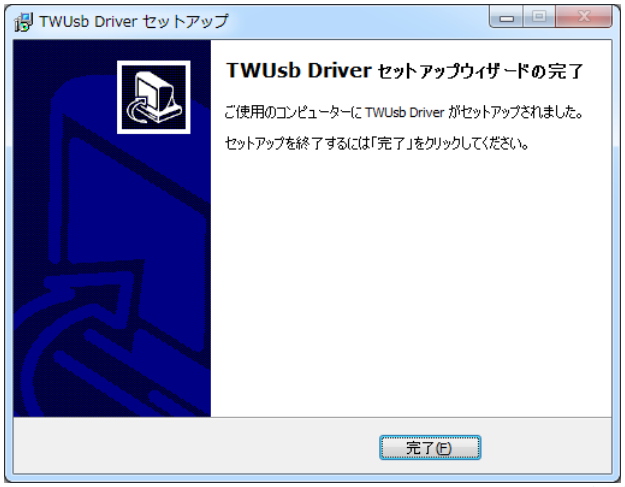

図 17　Windows 7 のドライバインストール画面（3）

⑤ デバイスを USB ケーブルでパソコンに接続します。図 18 のように「デバイス マネージャ」の画面に「USBM3069-H USB Device」と表示されれば、ドライバが正しくインストールされています。

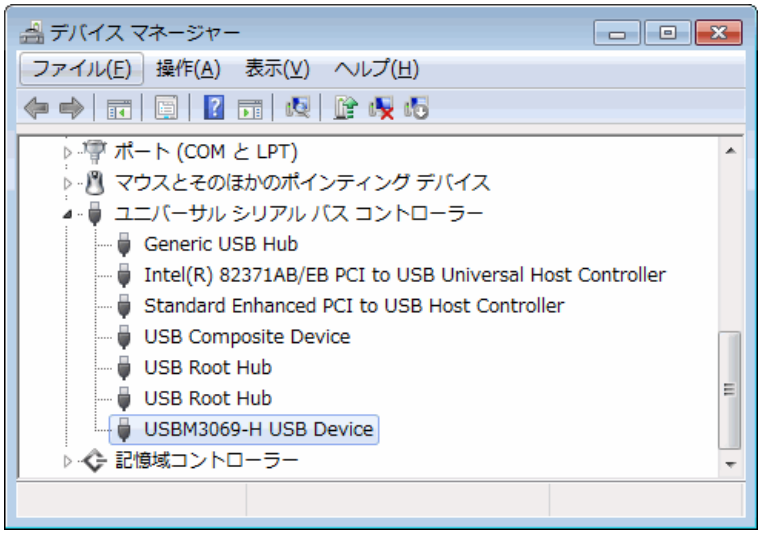

図 18　Windows 7 のドライバインストール確認

- 「デバイスマネージャ」を表示するには[コンピュータ]を右クリックし、[プロパティ]を選択します。[システム]画面が表示されますので、[タスク]中の[デバイスマネージャ]をクリックしてください。

□ ライブラリ、設定ツールのインストール

付属 CD の「¥TOOL¥USBX-A0x0xTools」フォルダから「setup.exe」を実行し、画面の指示に従ってインストールを行ってください。

表 12 は製品の制御に必要なライブラリファイルです。これらのファイルは、設定ツールをインストールすると自動的にシステムフォルダ(「C:¥Windows¥System32」など)にコピーされます。設定ツールをインストールしていないパソコンで製品を利用する際には表の「コピー先」フォルダにファイルをコピーするようにしてください [5]。

表 12　製品の制御に必要なファイル

32bit/64bit	ファイル名	CD 内の格納フォルダ	コピー先
32bit プログラムから制御する場合	USBM3069.DLL(32bit 版)	CD の「¥DLL」フォルダ	お客様で作成された実行ファイル(.EXE ファイル)と同一フォルダかシステムフォルダ(「C:¥Windows¥System32」など)
	TWXA.DLL(32bit 版)		
	M3069FlashWriter.atf		
64bit プログラムから制御する場合	USBM3069.DLL(64bit 版)	CD の「¥DLL¥x64」フォルダ	
	TWXA.DLL(64bit 版)		
	M3069FlashWriter.atf		

- 64bit 版 OS のシステムフォルダに 32bit 版の DLL ファイルをコピーする場合は、「System32」ではなく、「SysWOW64」フォルダにコピーしてください。
- Visual Basic for Applications および LabVIEW で開発したプログラムは 64bit 版 OS で使用する場合でも 32bit 版の DLL が必要です。

5 ドライバのインストールは必要です。

□ LabVIEW ライブラリのインストール

LabVIEW をご利用になる場合には、VI ライブラリのインストールを行います。インストールの前にご利用になるバージョンの LabVIEW がパソコンにインストールされていることをご確認ください。

VI ライブラリのインストール前に起動中の LabVIEW があれば終了してください。次に付属 CD の「¥VI¥TWXA-VI」フォルダから「setup.exe」を実行します。以下のような画面が表示され、現在パソコンにインストールされている LabVIEW のバージョンが表示されます。ご利用になるバージョンを選択して[次へ]ボタンを押してください。以降、画面に従ってインストールを完了します。

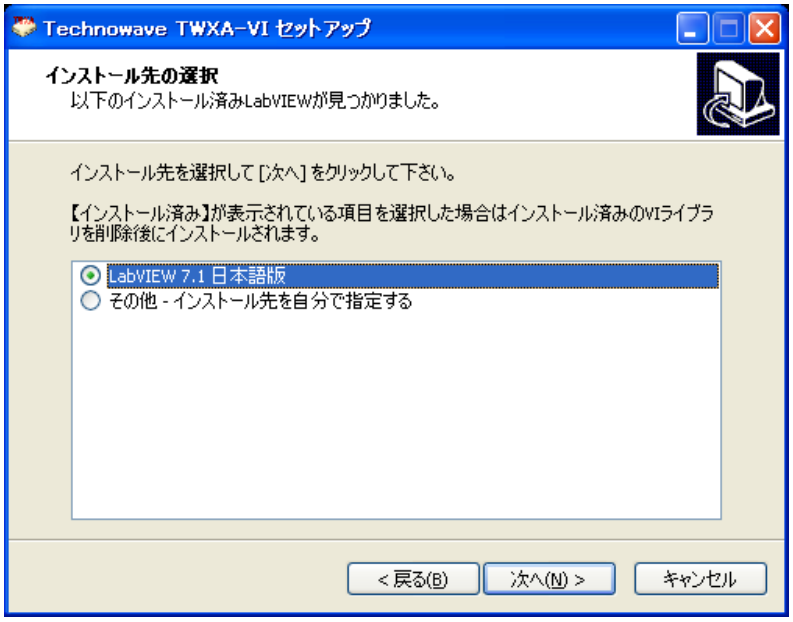

図 19　VI ライブラリのセットアップ画面

VI ライブラリの使用方法に関してはオンラインヘルプを参照してください。ヘルプファイルへのショートカットは[スタート]メニュー→[すべてのプログラム](または、[プログラム])→[テクノウェーブ]→[TWXA-VI」の中に作られます。

□ 設定ツールについて

19 ページの内容に従って設定ツールをインストールすると、[スタート]メニューの中に設定ツールの起動メニューが追加されます。デフォルトのインストールオプションでは[スタート]→[すべてのプログラム](または、[プログラム])→[テクノウェーブ]→[USBX-A0x0xTools]から起動することができます。

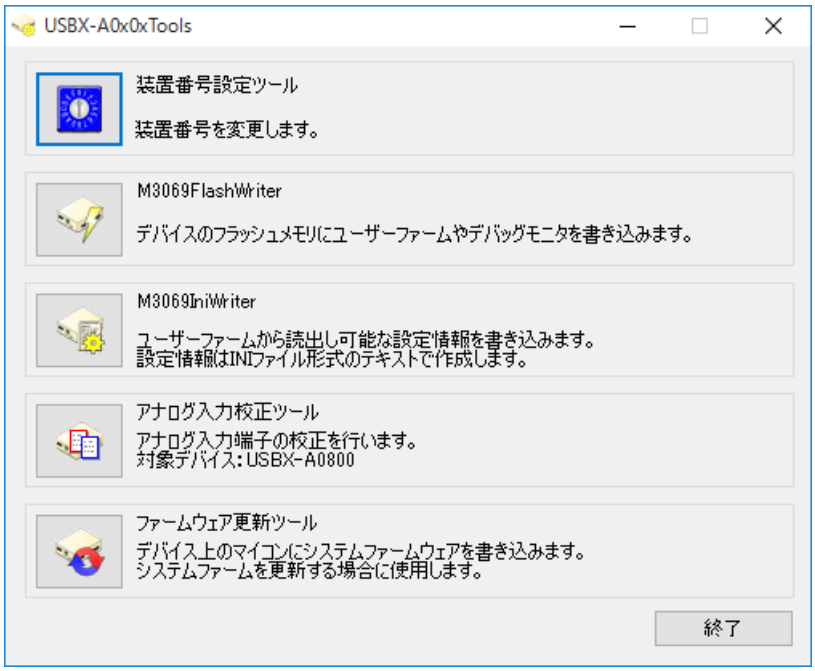

図 20　設定ツールのメニュー画面

表 13　設定ツールの機能説明

プログラム名	機能説明
装置番号設定ツール	装置番号を変更します。装置番号によって複数の製品を識別します。
M3069FlashWriter	主に製品のフラッシュメモリにユーザーファームウェアをダウンロードする場合に使用します。
M3069IniWriter	ユーザーファームに動作パラメータを与えたい場合に使用します。
アナログ入力校正ツール	アナログ入力の校正を行う場合に使用します。
ファームウェア更新ツール	製品のシステムファームを更新します。

各設定ツールの使用方法については、オンラインヘルプまたは画面の説明を参照してください。

- システムファームはバグの修正や、機能追加のために不定期に新しいバージョンのものが公開されます [6]。システムファームの更新ファイルは設定ツールの中に含まれていますので、更新する場合には、まず新しい設定ツールをご利用のパソコンにインストールしてください。

6 弊社ホームページにて随時公開します。

□ 装置番号設定

複数の製品を同時に制御する場合、それぞれの製品に識別のための装置番号を付与します。

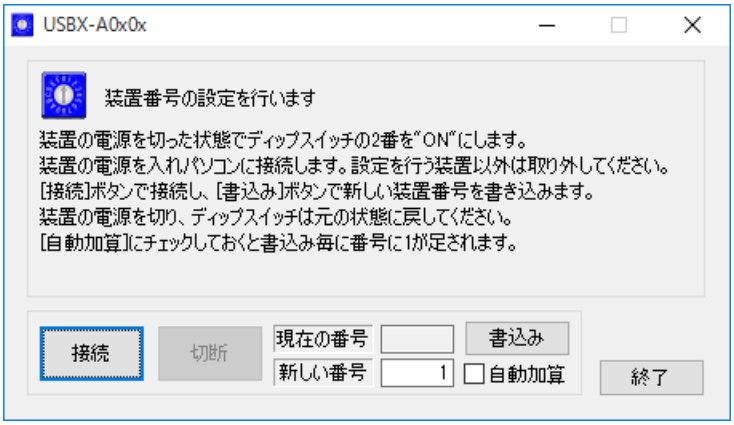

図 21　番号設定ツールの画面

1. 設定する製品のディップスイッチ 2 番を"ON"にしてフラッシュ書換えモードとし、パソコンに接続します。設定ツールは最初に見つかった製品に接続しますので、設定対象以外の製品は取り外してください。
2. 設定ツールのメニュー画面(21 ページ)から[装置番号設定ツール]ボタンを押します。図 21 のような画面が表示されます。
3. [接続]ボタンを押して製品に接続します。
4. [新しい番号]に 1～65535 の範囲の数値を入力します。
5. [自動加算]にチェックを入れておくと、書込みを行う度に[新しい番号]が 1 ずつ増加します。
6. [書込み]ボタンを押すと入力した装置番号が製品に設定されます。TWXA ライブラリの関数からは入力した番号を指定して接続を行うことができるようになります。
7. 製品を取り外しディップスイッチの 2 番を”OFF”に戻してください。番号の書換え可能回数の目安は 3200 回です。

□ アナログ入力校正

製品は出荷時にアナログ入力の校正が行われていますが、「アナログ入力校正ツール」を使用することで、ご利用環境に応じた設定で校正を行うことが可能です。

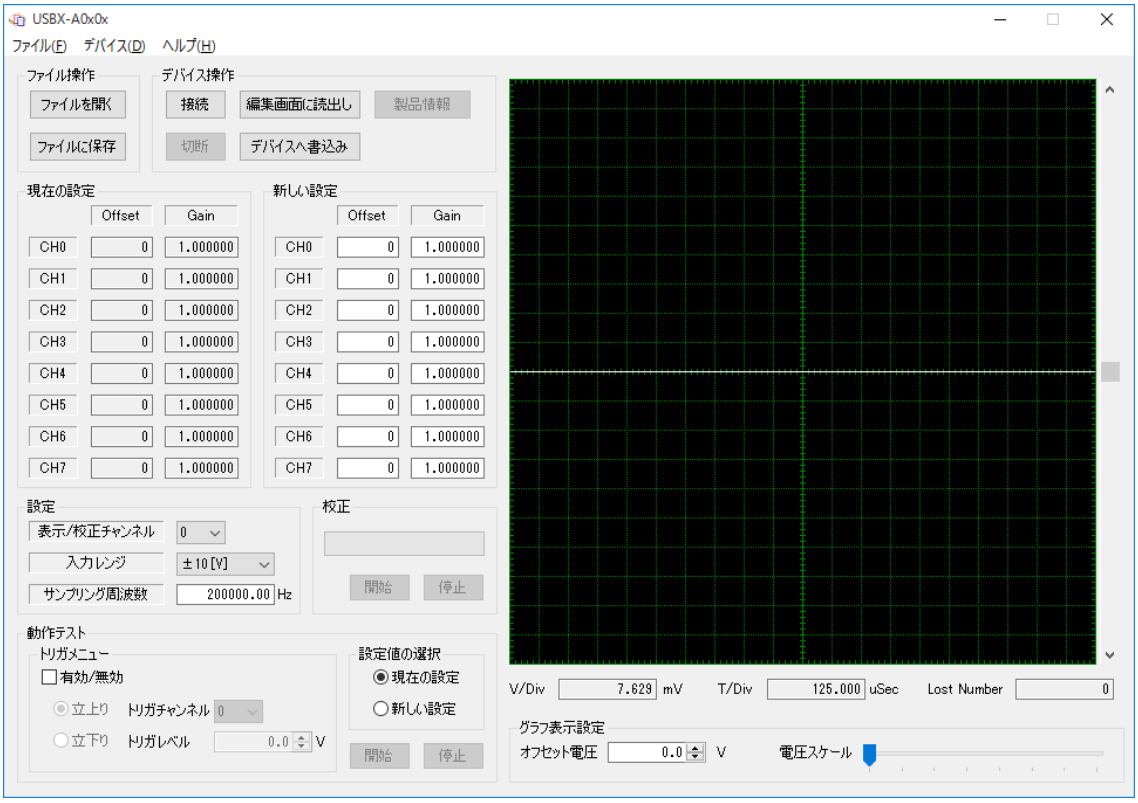

図 22　アナログ入力校正ツールの画面

製品に校正値が登録されている場合、全ての AD 変換結果は校正値が反映された、符号付き整数で返されます。

5. ハードウェア

□ アナログ入力

入力回路

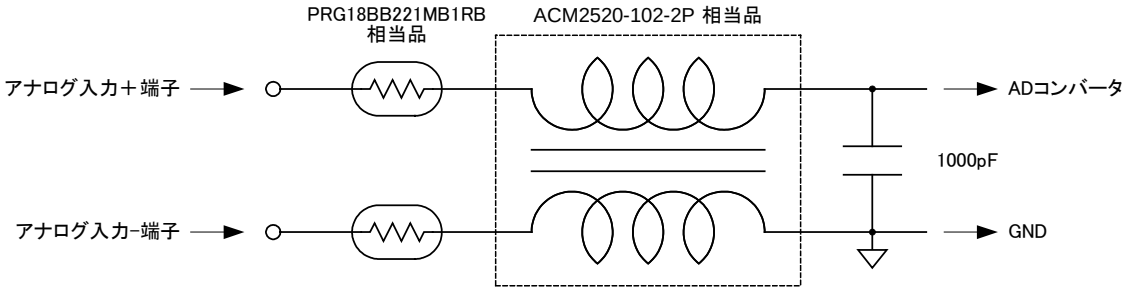

図 23　アナログ入力回路

- 全ての GND は製品内部で接続されています。

接続例（シングルエンド入力）

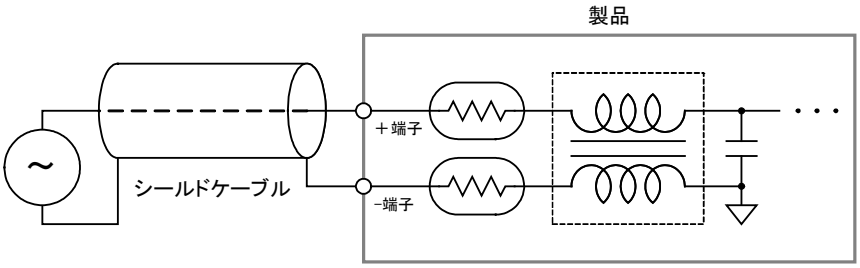

図 24　シングルエンド入力の接続例

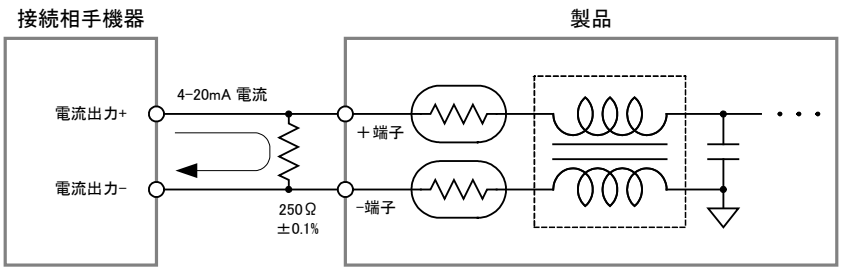

図 25　4-20mA 出力機器との接続例

接続例（差動信号入力）

任意のアナログ入力チャンネルを 2 つ使用することで、差動信号を計測することができます。図 26 に差動信号出力機器との接続例を示します。

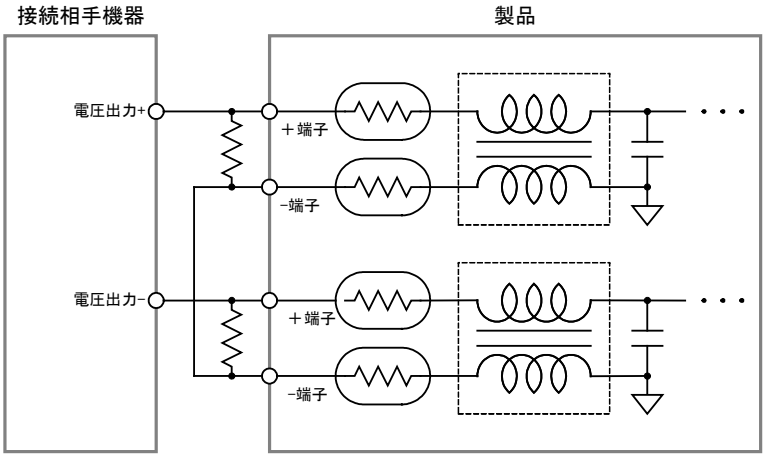

図 26　差動信号入力の接続例

差動信号の AD 変換結果は、使用する 2 チャンネルの AD 変換結果から算出します。

$$V_{diff} = V_{in+} - V_{in-}$$

V_{diff}：差動信号のAD変換結果

V_{in+}：差動信号＋入力チャンネルのAD変換結果

V_{in-}：差動信号 - 入力チャンネルのAD変換結果

式 1　差動信号の AD 変換結果取得方法

□ シリアル 0 (RS-485)

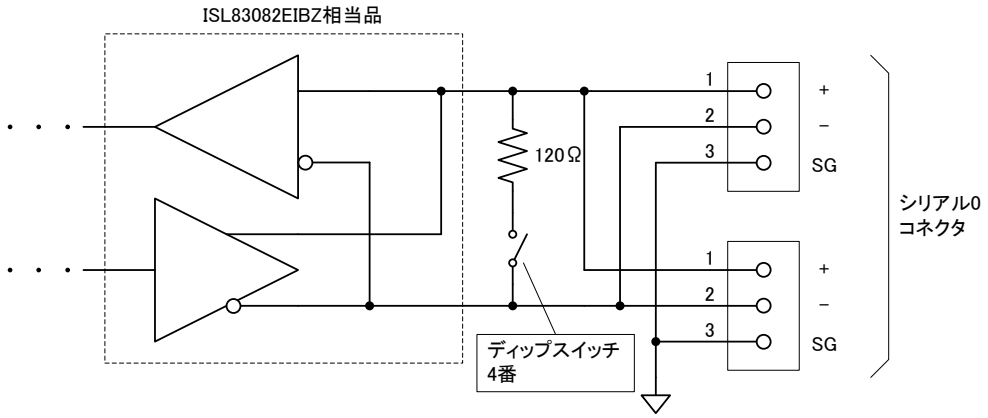

図 27　シリアル 0 の入出力回路

図 28、図 29 に RS-485 機器との接続例を示します。図のように製品が配線の終端位置にある場合にはディップスイッチの 4 番を"ON"にして終端抵抗を接続し、配線の中間にある場合にはディップスイッチの 4 番を"OFF"にします。

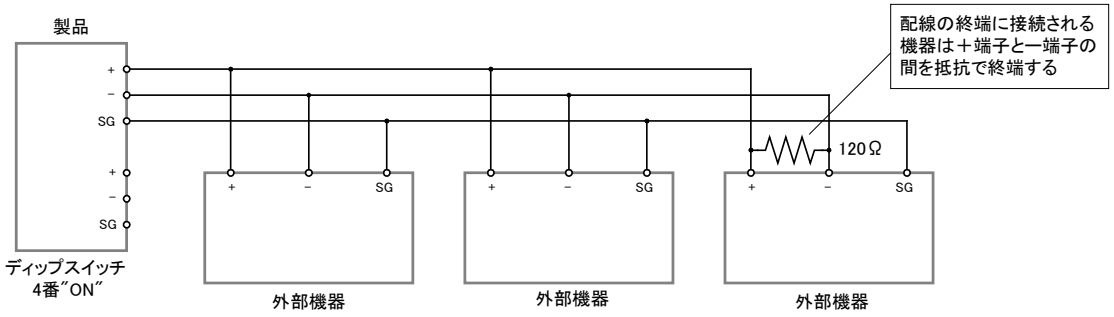

図 28　RS-485 の接続例(製品が配線の終端にある場合)

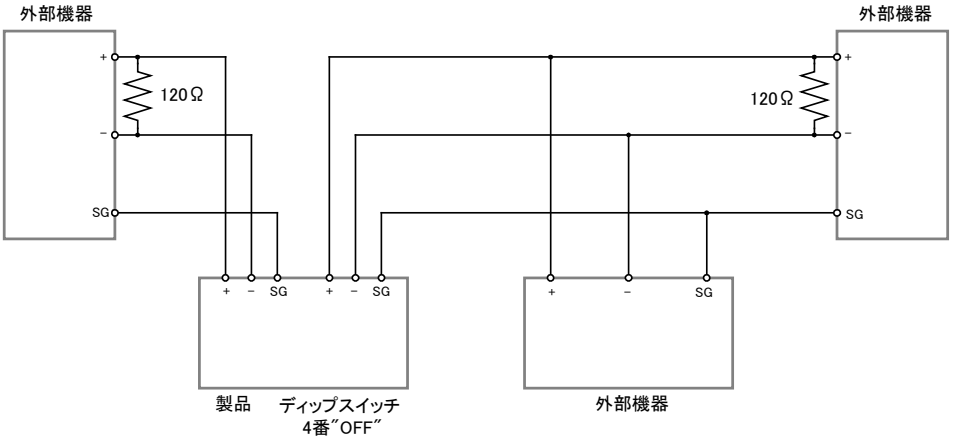

図 29　RS-485 の接続例(製品が配線の中間にある場合)

- 図 28、図 29 は一般的な例を示しています。外部機器の接続方法は使用する機器のマニュアルに従ってください。
- 使用するケーブルは特性インピーダンス 120Ωのシールド付ツイストペアケーブルが推奨されます。＋端子、－端子を芯線に、シールド線を SG に接続してください。
- SG が無い機器と接続する場合は、相手機器のマニュアルに従って接続してください。SG 端子を接続しない場合、通信エラーが起こりやすくなることがあります。

□ シリアル 1(RS-232C)

図 30 はシリアルポートのチャンネル 1 と一般的なパソコンのシリアルポートとの接続例です。ユーザーファームのデバッグ時には図 30 のように接続します。

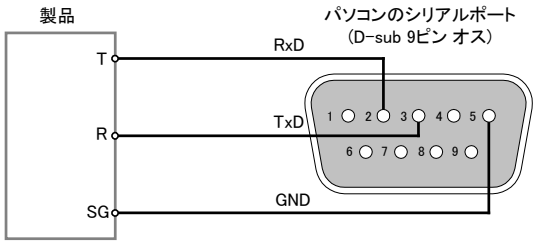

図 30 シリアル 1 の接続例

パソコン以外の機器と接続する場合は、表 14 を参照してください。

表 14　シリアル 1 の接続方法

製品の端子			接続相手機器の端子	
ピン番号	表示	入力/出力	信号名	入力/出力
1	T	出力	RxD(RD)	入力
2	R	入力	TxD(SD)	出力
3	SG	－	GND	－

6. プログラミング

Visual Studio 用のサンプルプログラム[7]は、付属 CD の「¥SAMPLE¥A0x0x_Samples」フォルダ中に収められています。言語別にソリューションファイル(表 15)が準備されていますので、必要に応じてご参照ください。

表 15　言語別のサンプルファイル

言語	ソリューションファイル
Visual C++(MFC)	A0x0xSamplesMFC.sln
Visual Basic	A0x0xSamplesVB.sln
Visual C#	A0x0xSamplesCS.sln

Visual Basic for Applications用のサンプルは付属CDの「¥SAMPLE¥A0x0x_Samples¥VBASamples」フォルダ内に格納されています。

□ プログラミングの準備

C/C++での開発に必要なファイル

表 16 は C/C++で開発を行うために必要なファイルです。製品付属の設定ツール(「USBX-A0x0xTools」)をインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 16　C/C++での開発に必要なファイル

ファイル名	説明	付属 CD 内の格納フォルダ
TWXA.h	TWXA ライブラリを使用するためのヘッダーファイル	「¥DLL」フォルダ
TWXA.lib(32bit 用)	TWXA ライブラリを静的にリンクするための lib ファイル	「¥DLL」フォルダ
TWXA.lib(64bit 用)		「¥DLL¥X64」フォルダ

「TWXA.h」は、TWXA ライブラリの関数や定数を使用するソースファイルでインクルードしてください。

「TWXA.lib」はプロジェクトをビルドする際のリンクファイルに含める必要があります。Visual Studio では、リスト 1 のように *#pragma* を使用してソースファイル中でリンク指定することもできます。

リスト 1　インクルードとリンク指定

```
#include "TWXA.h"
#pragma comment(lib, "TWXA.lib")
```

これらのファイルはコンパイラがビルド時に検索できるフォルダにコピーしておく必要があります。最も簡単な方法は、ビルドするプロジェクトと同一フォルダにコピーすることです。

[7] Visual Studio 2005 で作成されています。ご利用のバージョンによっては変換作業が必要になります(ソリューションファイルを開くと自動的に変換ウィザードが起動します)。

複数のプロジェクトを開発する場合は、これらのファイルを格納したフォルダを、開発環境の標準のインクルードパスや標準のリンクパスに追加すると便利です。追加の方法は開発環境によって異なりますので、それぞれのオンラインヘルプなどを参照してください。

- 「TWXA.h」は WIN32 API 固有の型などを使用しています。「コンソール アプリケーション」や「フォーム アプリケーション」を作成する場合には、「TWXA.h」より前に「Windows.h」のインクルードが必要な場合があります。

Visual Basic、C# での開発に必要なファイル

表 17 は Visual Basic、または、C# で開発を行うために必要なファイルです。製品付属の設定ツール(「USBX-A0x0xTools」)をインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 17　Visual Basic、C#での開発に必要なファイル

開発環境	ファイル名	説明	付属 CD 内の格納フォルダ
Visual Basic	TWXA.vb	TWXA ライブラリを使用するための定義ファイル	「¥DLL」フォルダ
Visual C#	TWXA.cs		

どちらの開発環境の場合も、Visual Studio の「ソリューション エクスプローラ」を開き、対応するファイルを開発プロジェクトの中にドラッグ・アンド・ドロップで追加することで、TWXA ライブラリの呼び出しが可能になります。これらのファイルは 32 ビット、64 ビットのどちらのプログラムを作成する場合にも共通で利用可能です。

Visual Basic for Applications での開発に必要なファイル

表 18 は Microsoft Office 製品の VBA で開発を行うために必要なファイルです。製品付属の設定ツール(「USBX-A0x0xTools」)をインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 18　Visual Basic for Applications での開発に必要なファイル

ファイル名	説明	付属 CD 内の格納フォルダ
TWXA.bas	TWXA ライブラリを使用するための定義ファイル	「¥DLL」フォルダ

開発を行うアプリケーションソフトで [Alt] + [F11]キーを押し、Visual Basic Editor を起動し、上記ファイルをプロジェクトウィンドウにドラッグ・アンド・ドロップで追加することで、TWXA ライブラリの呼び出しが可能になります。

- プロジェクトに追加したファイルは、ドキュメントファイル内にコピーが作成されます。ファイルを更新する場合は、以前に追加したファイルを一度解放し、新しいファイルを追加してください。

□ 接続

デバイスを操作するには、まず接続作業を行いハンドルを取得する必要があります。ハンドルとは接続時に決定される整数値で接続中のデバイスを識別するIDと考えることができます(図 31)。以降の操作は取得したハンドルを使用して行いますので、ハンドルの値は製品の操作を終了するまで記憶しておく必要があります。

また、デバイスの操作を終える場合はハンドルのクローズを行います。製品は 1 つのプログラムとしか接続ができませんので、ハンドルをクローズしていないプログラムが実行中の場合、他のプログラムからその製品に接続することはできません。

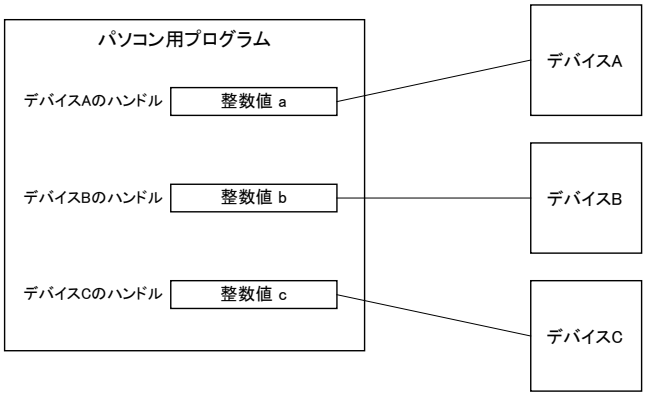

図 31　ハンドル

表 19　接続、初期化、終了に使用する関数

関数名	説明
TWXA_Open()	デバイスに接続します。
TWXA_Close()	ハンドルをクローズし、デバイスの操作を終了します。
TWXA_CloseAll()	プロセスが接続している全てのデバイスの操作を終了します。
TWXA_Initialize()	デバイスの再初期化が必要な場合呼び出します。必須ではありません。

デバイスに接続する

製品に接続する場合は表 20 の *TWXA_Open()* 関数を使用します。

表 20　TWXA_Open() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_Open(TW_HANDLE *phDev, long Number, long Opt)
VB	Function TWXA_Open(ByRef phDev As System.IntPtr, ByVal Number As Integer, ByVal Opt As TWXA_OPEN_OPT) As Integer
VBA	Function TWXA_Open(ByRef phDev As Long, ByVal Number As Long, ByVal Opt As TWXA_OPEN_OPT) As Long
C#	STATUS Open(out System.IntPtr phDev, int Number, OPEN_OPT Opt)

TWXA_Open() 関数では装置番号を指定してデバイスに接続できます。装置番号を指定する場合は引数 *Number* に番号を指定します。*Number* を 0 とした場合は、装置番号と無関係に最初に見つかったデバイスに接続されます。装置番号の設定方法は 22 ページを参照してください。

デバイスの操作を終了する

TWXA_Close() 関数を呼び出します。クローズしたハンドルは無効になります。

リスト 2　接続／切断の例(C 言語)

```
TW_HANDLE hDev;

//装置番号１のデバイスに接続
TWXA_Open(&hDev, 1, TWXA_ANY_DEVICE);

if(hDev){

  //...制御の中身

  TWXA_Close(hDev);  //操作を終了したらハンドルを閉じる
}
```

リスト 3　接続／切断の例(Visual Basic)

```
Dim hDev As System.IntPtr

'装置番号１番のデバイスに接続
TWXA_Open(hDev, 1, TWXA_OPEN_OPT.ANY_DEVICE)

If hDev <> System.IntPtr.Zero Then

  '...制御の中身

  TWXA_Close(hDev)  '操作を終了したらハンドルを閉じる
End If
```

リスト 4　接続／切断の例(VBA)

```
Dim hDev As Long

'装置番号１番のデバイスに接続
TWXA_Open hDev, 1, TWXA_OPEN_OPT.ANY_DEVICE

If hDev <> 0 Then

  '...制御の中身

  TWXA_Close hDev  '操作を終了したらハンドルを閉じる
End If
```

リスト 5　接続／切断の例(C#)

```
System.IntPtr hDev;

//装置番号１番のデバイスに接続
TWXA.Open(out hDev, 1, TWXA.OPEN_OPT.ANY_DEVICE);

if (hDev != System.IntPtr.Zero)
{
  //...制御の中身

  TWXA.Close(hDev);  //操作を終了したらハンドルを閉じる
}
```

TWXA_CloseAll() による切断

デバイスのハンドルはプロセスが終了した時点で全て解放されます。多くの開発環境ではデバッグを途中で停止すると開発中のプログラムのプロセスが終了しハンドルが解放されます。この場合、デバッグ中のプログラムに接続されていたデバイスは再度接続可能な状態に戻ります。

しかし、Microsoft Office などの一部の開発環境では開発中のプログラムが 1 つのプロセスの中で実行されるケースがあります。このような場合、プログラムのデバッグを途中で停止してもハンドルを所有していたプロセスは終了しないため、デバイスは切断されたことを認識することができません。そのため再度デバイスに接続しようとしてもデバイスは使用中とみなされ接続できない状態となります。

このような場合はプログラムの開始位置で *TWXA_CloseAll()* 関数を使用すると、プロセスが接続していたデバイスが一旦全て解放されるため、デバッグを途中で停止しても再度接続することが可能になります。

□ アナログ入力

製品はアナログ入力として非絶縁 16 ビット AD コンバータを 8 チャンネル搭載しています。アナログ入力に使用する端子は AD0～AD7 端子です。全ての端子はシングルエンドのバイポーラ入力となっており、入力レンジは-5～+5V と-10～+10V のどちらかを選択することができます。

表 21 はアナログ入力を制御するための関数です。表 22 はアナログ入力のサンプルプログラムです。

表 21　アナログ入力で使用する関数

関数名	説明
TWXA_ADRead()	AD 変換を一回行い、結果を読み出します。
TWXA_An16ToVolt()	アナログ入力の取得値を電圧値(ボルト単位)に変換します。
TWXA_ADSetRange()	アナログ入力端子の入力レンジを設定します。
TWXA_ADStartFastSampling()	アナログ入力の高速サンプリングを開始します。
TWXA_ADStartAutoSampling()	アナログ入力の回数を指定したサンプリングを開始します。
TWXA_ADStopSampling()	アナログ入力の連続サンプリングを停止します。
TWXA_ADGetQueueStatus()	バッファ中に蓄えられたサンプリングデータのデータ数を調べます。
TWXA_ADReadBuffer()	バッファ中に蓄えられたサンプリングデータを読み出します。
TWXA_ADPurgeBuffer()	バッファをクリアします。

表 22　アナログ入力のサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	AnalogSample	各アナログ入力端子の入力電圧を表示します。 TWXA_ADRead() を使用したサンプルプログラムです。
Visual Basic	AnalogSampleVB	
Visual C#	AnalogSampleCS	
Visual C++ (MFC)	AnalogAutoSample	一定周期でサンプリングした各アナログ入力端子の入力電圧を表示します。TWXA_ADStartAutoSampling()を使用したサンプルプログラムです。
Visual Basic	AnalogAutoSampleVB	
Visual C#	AnalogAutoSampleCS	
Visual C++ (MFC)	AnalogFastSample	高速でサンプリングされたデータをグラフへ表示する簡易オシロスコープです。TWXA_ADStartFastSampling()を使用したサンプルプログラムです。
Visual Basic	AnalogFastSampleVB	
Visual C#	AnalogFastSampleCS	
VBA（Excel）	AnalogSample.xls	簡易データロガーです。各アナログ入力端子の入力電圧を定期的に記録します。

入力レンジの設定

入力レンジを変更するには表 23 の *TWXA_ADSetRange()* 関数を使用します。*Range* 引数には表 24 の入力レンジを指定します。

入力レンジを変更する際は、連続サンプリングが停止している状態で行ってください。

表 23　TWXA_ADSetRange() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADSetRange(TW_HANDLE hDev, long Range)
VB	Function TWXA_ADSetRange(ByVal hDev As System.IntPtr, ByVal Range As TWXA_AN_OPTION) As Integer
VBA	Function TWXA_ADSetRange(ByVal hDev As Long, ByVal Range As TWXA_AN_OPTION) As Long
C#	STATUS ADSetRange(System.IntPtr hDev, AN_OPTION Range)

表 24　TWXA_ADSetRange() の Range 引数に指定する値

言語	値	説明
C/C++	TWXA_AN_10VPP	入力レンジを 10Vpp(-5～+5V)に設定します。
C++	TWXA::AN_OPTION::RANGE_10VPP	
VB/VBA	TWXA_AN_OPTION.RANGE_10VPP	
C#	TWXA.AN_OPTION.RANGE_10VPP	
C/C++	TWXA_AN_20VPP	入力レンジを 20Vpp(-10～+10V)に設定します。
C++	TWXA::AN_OPTION::RANGE_20VPP	
VB/VBA	TWXA_AN_OPTION.RANGE_20VPP	
C#	TWXA.AN_OPTION.RANGE_20VPP	

入力電圧値と読み出される値の関係は表 25 のようになります。

表 25　アナログ入力電圧と変換結果の関係

入力電圧値([V])		読み出される値
-5～+5V レンジの場合	-10～+10V レンジの場合	
5-LSB (LSB = 10 / 65536)	10-LSB (LSB = 20 / 65536)	32767
2.5	5	16384
0	0	0
-2.5	-5	-16384
-5	-10	-32768

・表は理論値を示しています。

読み出した全ての変換値は表 26 の *TWXA_An16ToVolt()* 関数を使用して電圧値に変換することが可能です。*Opt* 引数には *TWXA_ADSetRange()* 関数で設定した入力レンジと同じ値を指定してください。

表 26　TWXA_An16ToVolt() の関数宣言

言語	関数宣言
C/C++	double TWXA_An16ToVolt(long Data, long Opt)
VB	Function TWXA_An16ToVolt(ByVal Data As Integer, ByVal Opt As Integer) As Double Function TWXA_An16ToVolt(ByVal Data As Integer, ByVal Opt As TWXA_AN_OPTION) As Double
VBA	Function TWXA_An16ToVolt(ByVal Data As Long,ByVal Opt As Long) As Double
C#	double An16ToVolt(int Data) double An16ToVolt(int Data, uint Opt) double An16ToVolt(int Data, AN_OPTION Opt)

表 27　TWXA_An16ToVolt() の Opt 引数に指定する値

言語	値	説明
C/C++	TWXA_AN_10VPP	入力レンジが 10Vpp(-5～+5V)の場合に指定します。
C++	TWXA::AN_OPTION::RANGE_10VPP	
VB/VBA	TWXA_AN_OPTION.RANGE_10VPP	
C#	TWXA.AN_OPTION.RANGE_10VPP	
C/C++	TWXA_AN_20VPP	入力レンジが 20Vpp(-10～+10V)の場合に指定します。
C++	TWXA::AN_OPTION::RANGE_20VPP	
VB/VBA	TWXA_AN_OPTION.RANGE_20VPP	
C#	TWXA.AN_OPTION.RANGE_20VPP	

AD 変換結果の取得

AD 変換結果を得る方法は大きく分けて 2 つの方法があります。

- *TWXA_ADRead()* 関数を使用して、単純に命令発行時のアナログ電圧値を読み出す方法。
- *TWXA_ADStartAutoSampling()* または *TWXA_ADStartFastSampling()* 関数を使用して、タイマに同期した連続サンプリングを行う方法。

表 28 は、それぞれの変換方法の特徴をまとめたものです。

表 28　AD 変換の方法と特徴

代表関数名	サンプリング・レート	特徴
TWXA_ADRead()	-	使い方が簡単ですが、サンプリング・レートが使用環境に依存します。直流向きです。
TWXA_ADStartAutoSampling()	0.02～40,000[Hz]*1	一度製品内部でバッファリングを行うので、データを読み出すことが可能になるまでに最大で 500msec の遅延時間が発生します。サンプリング回数の指定およびサンプリング中のデバイスアクセスが可能です。
TWXA_ADStartFastSampling()	1,000～200,000[Hz]*1	最大レートでのサンプリングが可能ですが、サンプリング回数の指定およびサンプリング中のデバイスアクセスができません。

*1 設定可能なサンプリング・レートでも USB の通信状態や使用環境により、サンプリングデータを全て転送できない場合があります。

命令発行時のアナログ電圧値を読み出す

TWXA_ADRead() 関数(表 29)を使用します。関数を呼び出すと、ホストパソコンからデバイスに変換コマンドが送信され、デバイスは *Ch* 引数で指定されたチャンネルの AD 変換を行い、ホストパソコンに変換結果を返します。

表 29　TWXA_ADRead() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADRead(TW_HANDLE hDev, long Ch, long *pData)
VB	Function TWXA_ADRead(ByVal hDev As System.IntPtr, ByVal Ch As Integer, ByRef pData As Integer) As Integer Function TWXA_ADRead(ByVal hDev As System.IntPtr, ByVal Ch As Integer, ByVal pData() As Integer) As Integer
VBA	Function TWXA_ADRead(ByVal hDev As Long, ByVal Ch As Long, ByRef pData As Long) As Long
C#	STATUS ADRead(System.IntPtr hDev, int Ch, out int pData) STATUS ADRead(System.IntPtr hDev, int Ch, int []pData)

命令を呼び出して実際に AD 変換が行われるまでの時間は不定です(一般に数 msec のオーダーとなります)。繰り返し呼び出した場合の変換間隔も一定とはなりませんので、交流信号の変換には向きません。使い方が単純ですので直流信号を読み取るには適しています。

AD 変換結果は *pData* 引数に格納されます。1 チャンネルずつ読み出すこともできますが、*Ch* 引数に *TWXA_AD_ALL*(相当の値)を指定すると、0～7 チャンネルまで全てのチャンネルを、同時に変換した結果を読み出すことができます。その場合は、*pData* 引数として 8 チャンネル分(32 バイト)の領域を確保するようにしてください。

差動信号を計測する場合は、必ず *Ch* 引数に *TWXA_AD_ALL*(相当の値)を指定してください。

リスト 6　TWXA_ADRead()の使用例 (C 言語)

```
long lData[8];
double dVolt;

//入力レンジを-10～+10V に設定
TWXA_ADSetRange(hDev, TWXA_AN_20VPP);

//AD0-AD7 の AD 変換結果を読出し
TWXA_ADRead(hDev, TWXA_AD_ALL, lData);

//AD0 を電圧値に変換
dVolt = TWXA_An16ToVolt(lData[0], TWXA_AN_20VPP);

//CH0 へ差動信号の[+]、CH1 へ差動信号の[-]を入力した場合
dVolt = TWXA_An16ToVolt(lData[0] - lData[1], TWXA_AN_20VPP);
```

リスト 7　TWXA_ADRead()の使用例（Visual Basic）

```
Dim iAD(7) As Integer
Dim dVolt As Double

'入力レンジを-10～+10V に設定
TWXA_ADSetRange(hDev, TWXA_AN_OPTION.RANGE_20VPP)

'AD0-AD7 の AD 変換結果を読出し
TWXA_ADRead(hDev, TWXA_AD_ALL, iAD)

'AD0 を電圧値に変換
dVolt = TWXA_An16ToVolt(iAD(0), TWXA_AN_OPTION.RANGE_20VPP)

'CH0 へ差動信号の[+]、CH1 へ差動信号の[-]を入力した場合
dVolt = TWXA_An16ToVolt(iAD(0) - iAD(1), TWXA_AN_OPTION.RANGE_20VPP)
```

リスト 8　TWXA_ADRead()の使用例（C#）

```
int[] iAD = new int[8];
double dVolt;

//入力レンジを-10～+10V に設定
TWXA.ADSetRange(hDev, TWXA.AN_OPTION.RANGE_20VPP);

//AD0-AD7 の AD 変換結果を読出し
TWXA.ADRead(hDev, TWXA.AD_ALL, iAD);

//AD0 を電圧値に変換
dVolt = TWXA.An16ToVolt(iAD[0], TWXA.AN_OPTION.RANGE_20VPP);

//CH0 へ差動信号の[+]、CH1 へ差動信号の[-]を入力した場合
dVolt = TWXA.An16ToVolt(iAD[0] - iAD[1], TWXA.AN_OPTION.RANGE_20VPP);
```

- 例ではデバイスへの接続やエラー処理が省略されています。接続方法については 30 ページを、エラー処理については 60 ページを参照してください。以降のページで示す例も同様です。

タイマに同期した連続サンプリングを開始する

サンプリング回数を指定して連続サンプリングを行うには *TWXA_ADStartAutoSampling()* 関数(表 30)、最大レートで連続サンプリングを行うには *TWXA_ADStartFastSampling()* 関数(表 31)を使用します。

表 30 TWXA_ADStartAutoSampling() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADStartAutoSampling(TW_HANDLE hDev, double *pRate, DWORD nSampling)
VB	Function TWXA_ADStartAutoSampling(ByVal hDev As System.IntPtr, ByRef pRate As Double, ByVal nSampling As Integer) As Integer
VBA	Function TWXA_ADStartAutoSampling(ByVal hDev As Long, ByRef pRate As Double, ByVal nSampling As Long) As Long
C#	STATUS ADStartAutoSampling(System.IntPtr hDev, ref double pRate, uint nSampling) STATUS ADStartAutoSampling(System.IntPtr hDev, ref double pRate)

表 31 TWXA_ADStartFastSampling() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADStartFastSampling(TW_HANDLE hDev, double *pRate)
VB	Function TWXA_ADStartFastSampling(ByVal hDev As System.IntPtr, ByRef pRate As Double) As Integer
VBA	Function TWXA_ADStartFastSampling(ByVal hDev As Long, ByRef pRate As Double) As Long
C#	STATUS ADStartFastSampling(System.IntPtr hDev, ref double pRate)

pRate 引数にはサンプリング・レートを Hz単位で入力します。サンプリング・レートは内部クロックを分周して生成されるため、実際に設定できるサンプリング・レートは離散的になります。*TWXA_ADStartAutoSampling()* 、*TWXA_ADStartFastSampling()* 関数は *pRate* 引数の入力値と近い値に調整し、実際に設定できた値を *pRate* 引数に出力して返ります。設定可能なサンプリング・レートは関数により異なります(表 32 参照)。

表 32 ***pRate*** 引数に設定可能なサンプリング・レート

関数名	サンプリング・レート
TWXA_ADStartAutoSampling()	0.02～40,000[Hz]*1
TWXA_ADStartFastSampling()	1,000～200,000[Hz]*1

*1 設定可能なサンプリング・レートでも USB の通信状態や使用環境により、サンプリングデータを全て転送できない場合があります。

TWXA_ADStartAutoSampling() 関数の *nSampling* 引数にはサンプリング回数を入力します。*nSampling* 引数に 0xFFFFFFFF を指定すると *TWXA_ADStopSampling()* 関数を呼び出すまでサンプリングを行います。

連続サンプリングの場合、全てのチャンネルの AD 変換は同じタイミングで行われます。サンプリング・レートと各チャンネルの変換タイミングを図 32 に示します。

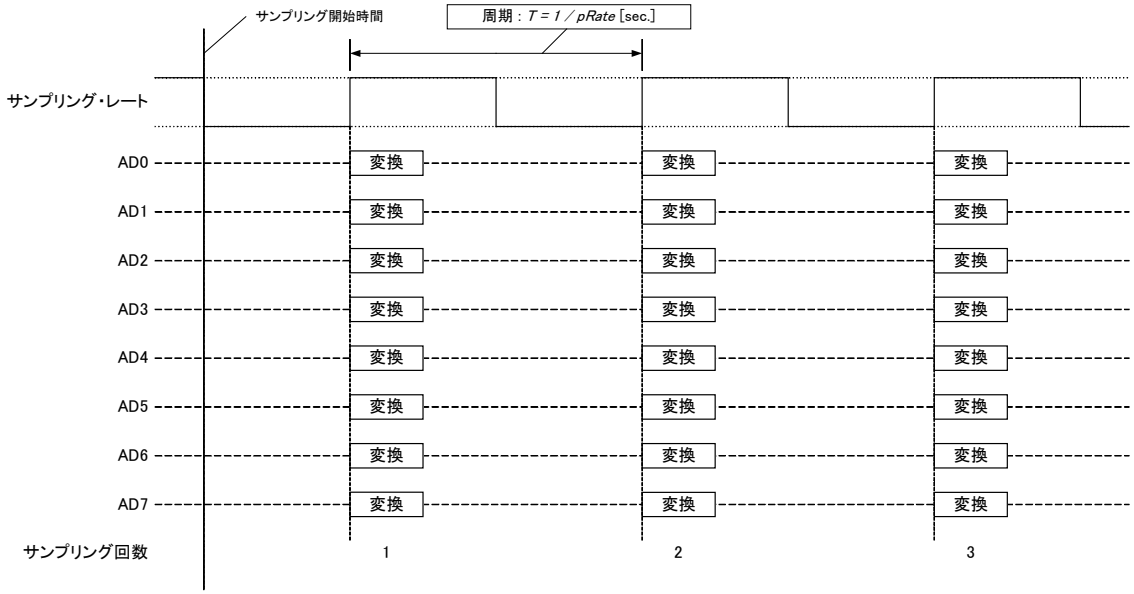

図 32　サンプリング・レートと変換タイミングの関係

TWXA_ADStartAutoSampling() または *TWXA_ADStartFastSampling()* 関数を呼び出すと、デバイスは連続サンプリングを開始しますが、関数自体はすぐにリターンします。

TWXA_ADStartAutoSampling() を使用して連続サンプリングを開始した場合、サンプリングデータは一度製品内部のバッファ [8]に保存され、一定データ数溜まる、または、一定時間経過するとホストパソコンに送信されます。

デバイスから送信されたサンプリングデータはパソコン上のメモリにバッファリング [9]されますが、接続された USB ポートの通信状態や使用環境により、デバイスはサンプリングデータを転送できない場合があります。その場合、サンプリングデータはホストパソコンに転送されるまで製品内部のバッファに蓄積されます。ただし、転送できない状態が続き製品内部のバッファがいっぱいになってしまうと、新たにサンプリングされたデータは破棄されてしまいますのでご注意ください。

サンプリング中はホストパソコンのプログラムはブロッキングされませんので、メッセージ処理や画面描画などを行うことができます。また、サンプリング中にシリアルポートやユーザーステータス等の操作を行うことができます。

8 128 データ分をバッファできます。
9 65536 データ分をバッファできます。

TWXA_ADStartFastSampling() を使用して連続サンプリングを開始した場合、サンプリングデータは逐次ホストパソコンに送信されます。

デバイスから送信されたサンプリングデータはパソコン上のメモリにバッファリング[9]されますが、USBポートの通信状態や使用環境により、デバイスはサンプリングデータを転送できない場合があります。その場合のサンプリングデータはデバイスに蓄積されることなく破棄されてしまいますのでご注意ください。

サンプリング中はホストパソコンのプログラムはブロッキングされませんので、メッセージ処理や画面描画などを行うことができます。ただし、デバイスには *TWXA_ADStopSampling()* による中断コマンド以外のコマンドを送信しないでください。誤ってコマンドを送信すると連続サンプリングは停止してしまいます。

連続サンプリングを停止する

TWXA_ADStopSampling() 関数(表 33)を使用します。連続サンプリングを開始した場合、必ず *TWXA_ADStopSampling()* 関数を呼び出してください。

表 33 TWXA_ADStopSampling() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADStopSampling(TW_HANDLE hDev)
VB	Function TWXA_ADStopSampling(ByVal hDev As System.IntPtr) As Integer
VBA	Function TWXA_ADStopSampling(ByVal hDev As Long) As Long
C#	STATUS ADStopSampling(System.IntPtr hDev)

サンプリングデータを読み出す

連続サンプリングの動作状態および受信バッファに蓄えられたデータ数を取得するには *TWXA_ADGetQueueStatus()* 関数(表 34)、受信バッファからデータを読み出すには *TWXA_ADReadBuffer()* 関数(表 35)を使用します。*TWXA_ADReadBuffer()* 関数の *pData* 引数には *TWXA_A0x0x_DATA* 構造体(表 36)の配列を渡します。

これらの関数はデバイスにコマンドを送信しないので *TWXA_ADStartFastSampling()* を使用して連続サンプリングを開始した場合でも呼び出すことができます。

表 34 TWXA_ADGetQueueStatus() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADGetQueueStatus(TW_HANDLE hDev, int *pStatus, long *pnReceive)
VB	Function TWXA_ADGetQueueStatus(ByVal hDev As System.IntPtr, ByRef pStatus As Integer, ByRef pnReceive As Integer) As Integer
VBA	Function TWXA_ADGetQueueStatus(ByVal hDev As Long, ByRef pStatus As Long, ByRef pnReceive As Long) As Long
C#	STATUS ADGetQueueStatus(System.IntPtr hDev, out int pStatus, out int pnReceive)

表 35　TWXA_ADReadBuffer() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADReadBuffer(TW_HANDLE hDev, void *pData, long nData, long *pnRead)
VB	Function TWXA_ADReadBuffer(ByVal hDev As System.IntPtr, ByVal pData() As TWXA_AOxOx_DATA, ByVal nData As Integer, ByRef pnRead As Integer) As Integer
VBA	Function TWXA_ADReadBuffer(ByVal hDev As Long, ByRef pData As Any, ByVal nData As Long, ByRef pnRead As Long) As Long
C#	STATUS ADReadBuffer(System.IntPtr hDev, AOxOx_DATA []pData, int nData, out int pnRead)

表 36　TWXA_A0x0x_DATA 構造体の宣言

言語	関数宣言
C/C++	typedef struct { DWORD Index; short Data[8]; } TWXA_AOxOx_DATA;
VB	Public Structure TWXA_AOxOx_DATA Public Index As Integer <MarshalAs(UnmanagedType.ByValArray, SizeConst:=8)> _ Public Data() As Short Public Sub Initialize() ReDim Data(7) End Sub End Structure
VBA	Public Type TWXA_ATF_INFO Index As Long Data(7) As Integer End Type

C#	public struct AOxOx_DATA
	{
	public uint Index;
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
	public short[] Data;
	
	public void Initialize()
	{
	Data = new short[8];
	}
	}

Index
データのインデックスが出力されます。データが破棄されていない場合、連番となります。データが破棄されている場合、インデックスに破棄されたデータ数が加算されます。

Data
アナログ入力の各チャンネルの AD 変換値が出力されます。配列のインデックスがアナログ入力のチャンネルと対応しています。

Initialize()
Visual Basic と C#では構造体メンバ *Data* の領域確保用に最初に呼び出します。

リスト 9　TWXA_ADStartAutoSampling()の使用例（C 言語）

```
double dRate;
long n;
int status;
DWORD i;
DWORD dwIndex;
DWORD dwLost;
TWXA_AOxOx_DATA Data[1000];
TCHAR c[256];

dRate = 1000.0; //サンプリング・レート = 1,000S/sec

//入力レンジを-10～+10V に設定
TWXA_ADSetRange(hDev, TWXA_AN_20VPP);

//回数を指定して連続サンプリングを開始
TWXA_ADStartAutoSampling(hDev, &dRate, 1000);

while(1){
  TWXA_ADGetQueueStatus(hDev, &status, &n); //受信データ数を取得
  if(n >= 1000) break; //指定回数のサンプリングが終了したら抜ける
}

//連続サンプリングを停止
TWXA_ADStopSampling(hDev);

//サンプリングデータの読出し
TWXA_ADReadBuffer(hDev, Data, 1000, &n);

//バッファに残っているデータをクリア
TWXA_ADPurgeBuffer(hDev);

//インデックスを確認してデータが破棄されていないか確認
dwIndex = Data[0].Index + 1;
for(i = 1, dwLost = 0; i < (unsigned)n; i++){
  if(Data[i].Index != dwIndex){
    dwLost += (dwIndex - Data[i].Index);
    dwIndex = Data[i].Index;
  }
  dwIndex++;
}

if(dwLost != 0){
  _sprintf_s(c, 256, _T("%d 個のデータが破棄されました。¥n"), dwLost);
  OutputDebugString(c);
}
```

リスト 10　TWXA_ADStartAutoSampling()の使用例 (Visual Basic)

```
Dim dRate As Double
Dim n As Integer
Dim status As Integer
Dim i As Integer
Dim Index As Integer
Dim Lost As Integer
Dim Data(999) As TWXA_AOxOx_DATA

dRate = 1000.0 'サンプリング・レート = 1,000S/sec
For i = 0 To 999
  Data(i).Initialize() '構造体メンバの Data を初期化
Next

'入力レンジを-10～+10V に設定
TWXA_ADSetRange(hDev, TWXA_AN_OPTION.RANGE_20VPP)

'回数を指定して連続サンプリングを開始
TWXA_ADStartAutoSampling(hDev, dRate, 1000)

While True
  TWXA_ADGetQueueStatus(hDev, status, n) '受信データ数を取得
  If n >= 1000 Then Exit While '指定回数のサンプリングが終了したら抜ける
End While

'連続サンプリングを停止
TWXA_ADStopSampling(hDev)

'サンプリングデータの読出し
TWXA_ADReadBuffer(hDev, Data, 1000, n)

'バッファに残っているデータをクリア
TWXA_ADPurgeBuffer(hDev)

'インデックスを確認してデータが破棄されていないか確認
Index = Data(0).Index + 1
Lost = 0
For i = 1 To n - 1
  If Data(i).Index <> Index Then
    Lost = Lost + Index - Data(i).Index
    Index = Data(i).Index
  End If
  Index = Index + 1
Next

If Lost <> 0 Then
  Debug.WriteLine(String.Format("{0}個のデータが破棄されました。", Lost))
End If
```

リスト 11　TWXA_ADStartAutoSampling()の使用例（C#）

```
double dRate;
int n;
int status;
uint i;
uint dwIndex;
uint dwLost;
TWXA.AOxOx_DATA[] Data = new TWXA.AOxOx_DATA[1000];

dRate = 1000.0; //サンプリング・レート = 1,000S/sec
for (i = 0; i < 1000; i++) Data[i].Initialize(); //構造体メンバの Data を初期化

//入力レンジを-10～+10V に設定
TWXA.ADSetRange(hDev, TWXA.AN_OPTION.RANGE_20VPP);

//回数を指定して連続サンプリングを開始
TWXA.ADStartAutoSampling(hDev, ref dRate, 1000);

while (true)
{
  TWXA.ADGetQueueStatus(hDev, out status, out n); //受信データ数を取得
  if (n >= 1000) break; //指定回数のサンプリングが終了したら抜ける
}

//連続サンプリングを停止
TWXA.ADStopSampling(hDev);

//サンプリングデータの読出し
TWXA.ADReadBuffer(hDev, Data, 1000, out n);

//バッファに残っているデータをクリア
TWXA.ADPurgeBuffer(hDev);

//インデックスを確認してデータが破棄されていないか確認
dwIndex = Data[0].Index + 1;
for (i = 1, dwLost = 0; i < n; i++)
{
  if (Data[i].Index != dwIndex)
  {
    dwLost += (dwIndex - Data[i].Index);
    dwIndex = Data[i].Index;
  }
  dwIndex++;
}

if (dwLost != 0)
{
  Debug.WriteLine(string.Format("{0}個のデータが破棄されました。", dwLost));
}
```

リスト 12　TWXA_ADStartFastSampling()の使用例 (C 言語)

```
double dRate;
long n;
int status;
DWORD i;
DWORD dwIndex;
DWORD dwLost;
TWXA_AOxOx_DATA Data[10000];
TCHAR c[256];

dRate = 10000.0; //サンプリング・レート = 10,000S/sec

//入力レンジを-10～+10V に設定
TWXA_ADSetRange(hDev, TWXA_AN_20VPP);

//連続サンプリングを開始
TWXA_ADStartFastSampling(hDev, &dRate);

while(1){
  TWXA_ADGetQueueStatus(hDev, &status, &n); //受信データ数を取得
  if(n >= 10000) break; //必要なデータ数を受信したら抜ける
}

//連続サンプリングを停止
TWXA_ADStopSampling(hDev);

//サンプリングデータの読出し
TWXA_ADReadBuffer(hDev, Data, 10000, &n);

//バッファに残っているデータをクリア
TWXA_ADPurgeBuffer(hDev);

//インデックスを確認してデータが破棄されていないか確認
dwIndex = Data[0].Index + 1;
dwLost = 0;
for(i = 1; i < (unsigned)n; i++){
  if(Data[i].Index != dwIndex){
    dwLost += (dwIndex - Data[i].Index);
    dwIndex = Data[i].Index;
  }
  dwIndex++;
}

if(dwLost != 0){
  _sprintf_s(c, 256, _T("%d 個のデータが破棄されました。¥n"), dwLost);
  OutputDebugString(c);
}
```

リスト 13　TWXA_ADStartFastSampling()の使用例 (Visual Basic)

```
Dim dRate As Double
Dim n As Integer
Dim status As Integer
Dim i As Integer
Dim Index As Integer
Dim Lost As Integer
Dim Data(9999) As TWXA_AOxOx_DATA

dRate = 10000.0 'サンプリング・レート = 10,000S/sec
For i = 0 To 9999
  Data(i).Initialize() '構造体メンバの Data を初期化
Next

'入力レンジを-10～+10V に設定
TWXA_ADSetRange(hDev, TWXA_AN_OPTION.RANGE_20VPP)

'連続サンプリングを開始
TWXA_ADStartFastSampling(hDev, dRate)

While True
  TWXA_ADGetQueueStatus(hDev, status, n) '受信データ数を取得
  If n >= 10000 Then Exit While '必要なデータ数を受信したら抜ける
End While

'連続サンプリングを停止
TWXA_ADStopSampling(hDev)

'サンプリングデータの読出し
TWXA_ADReadBuffer(hDev, Data, 10000, n)

'バッファに残っているデータをクリア
TWXA_ADPurgeBuffer(hDev)

'インデックスを確認してデータが破棄されていないか確認
Index = Data(0).Index + 1
Lost = 0
For i = 1 To n - 1
  If Data(i).Index <> Index Then
    Lost = Lost + Index - Data(i).Index
    Index = Data(i).Index
  End If
  Index = Index + 1
Next

If Lost <> 0 Then
  Debug.WriteLine(String.Format("{0}個のデータが破棄されました。", Lost))
End If
```

リスト 14　TWXA_ADStartFastSampling()の使用例（C#）

```
double dRate;
int n;
int status;
uint i;
uint dwIndex;
uint dwLost;
TWXA.AOxOx_DATA[] Data = new TWXA.AOxOx_DATA[10000];

dRate = 10000.0; //サンプリング・レート = 10,000S/sec
for (i = 0; i < 10000; i++) Data[i].Initialize(); //構造体メンバの Data を初期化

//入力レンジを-10～+10V に設定
TWXA.ADSetRange(hDev, TWXA.AN_OPTION.RANGE_20VPP);

//連続サンプリングを開始
TWXA.ADStartFastSampling(hDev, ref dRate);

while (true)
{
  TWXA.ADGetQueueStatus(hDev, out status, out n); //受信データ数を取得
  if (n >= 10000) break; //必要なデータ数を受信したら抜ける
}

//連続サンプリングを停止
TWXA.ADStopSampling(hDev);

//サンプリングデータの読出し
TWXA.ADReadBuffer(hDev, Data, 10000, out n);

//バッファに残っているデータをクリア
TWXA.ADPurgeBuffer(hDev);

//インデックスを確認してデータが破棄されていないか確認
dwIndex = Data[0].Index + 1;
for (i = 1, dwLost = 0; i < n; i++)
{
  if (Data[i].Index != dwIndex)
  {
    dwLost += (dwIndex - Data[i].Index);
    dwIndex = Data[i].Index;
  }
  dwIndex++;
}

if (dwLost != 0)
{
  Debug.WriteLine(string.Format("{0}個のデータが破棄されました。", dwLost));
}
```

□ シリアルポート

シリアルポートは最大 2 チャンネル使用可能です。シリアル 0 は RS-485 の半二重通信用です。通常は受信状態となっており、送信用の関数を呼び出した場合のみ自動的に送信状態に切り替わります。

シリアル 1 は RS-232C に準拠した信号レベルでの通信を行います。デフォルトの状態ではユーザーファームのデバッグ用ポート、または、標準入出力ポートとして機能します。ユーザーファームを利用しない場合は、*TWXA_SCISetMode()* をシリアル 1 に対して呼び出すことで、TWXA ライブラリから制御可能な状態となります。

通信方式は調歩同期のみです。通信速度は 300bps～38400bps でフロー制御はありません。受信バッファは 127 バイトでオーバーフローするとステータスレジスタにエラーを記録し、オーバーフローしたデータは捨てられます。

また、受信データを改行コードなどで分割して読み出したい場合には、デリミタコードを設定しておくことができます。デリミタコードを設定しておくと、*TWXA_SCIRead()* 呼び出し時に受信データがチェックされ、デリミタコード(1 バイトまたは 2 バイト)が現れると、シリアルポートからの読取りを一旦中止し、デリミタコードより後には指定バイトまで 0 をコピーしてデータを返します。

表 37 にシリアルポート制御で使用する関数をあげます。

表 37　シリアルポート制御で使用する関数

関数名	説明
TWXA_SCISetMode()	通信条件の設定を行います。
TWXA_SCIReadStatus()	シリアルポートのエラー、受信バイト数を読み出します。
TWXA_SCIRead()	シリアルポートから指定バイト数のデータを読み出します。
TWXA_SCIWrite()	シリアルポートからデータを送信します。
TWXA_SCISetDelimiter()	デリミタ文字を指定します。

表 38　シリアルポート制御のサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	SerialSample	文字の送受信が可能な簡易なターミナルソフト。
Visual Basic	SerialSampleVB	
Visual C#	SerialSampleCS	

シリアルポートの設定

表 39 は *TWXA_SCISetMode()* 関数の宣言です。*Mode* 引数には表 40 に示す値を OR で結合して指定します。その際、データ長、パリティ、ストップビットの設定から 1 つずつオプションを選択して結合するようにしてください。指定がない設定項目はデフォルトと書かれたオプションが選択されます。また、*Baud* 引数には表 41 のボーレートを入力します。

表 39 TWXA_SCISetMode() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_SCISetMode(TW_HANDLE hDev, long Ch, long Mode, long Baud)
VB	Function TWXA_SCISetMode(ByVal hDev As System.IntPtr, ByVal Ch As Integer, ByVal Mode As TWXA_SCI_MODE, ByVal Baud As TWXA_SCI_BAUD) As Integer
VBA	Function TWXA_SCISetMode(ByVal hDev As Long, ByVal Ch As Long, ByVal Mode As TWXA_SCI_MODE, ByVal Baud As TWXA_SCI_BAUD) As Long
C#	STATUS SCISetMode(System.IntPtr hDev, int Ch, SCI_MODE Mode, SCI_BAUD Baud)

表 40 TWXA_SCISetMode() の Mode 引数に指定する値

	言語	値	説明
データ長	C/C++	TWXA_SCI_DATA8	データ長を 8 ビットにします(デフォルト)。
	C++	TWXA::SCI_MODE::DATA8	
	VB/VBA	TWXA_SCI_MODE.DATA8	
	C#	TWXA.SCI_MODE.DATA8	
	C/C++	TWXA_SCI_DATA7	データ長を 7 ビットにします。
	C++	TWXA::SCI_MODE::DATA7	
	VB/VBA	TWXA_SCI_MODE.DATA7	
	C#	TWXA.SCI_MODE.DATA7	
パリティ	C/C++	TWXA_SCI_NOPARITY	パリティビットを使用しません(デフォルト)。
	C++	TWXA::SCI_MODE::NO_PARITY	
	VB/VBA	TWXA_SCI_MODE.NO_PARITY	
	C#	TWXA.SCI_MODE.NO_PARITY	
	C/C++	TWXA_SCI_EVEN	偶数パリティを使用します。
	C++	TWXA::SCI_MODE::EVEN	
	VB/VBA	TWXA_SCI_MODE.EVEN	
	C#	TWXA.SCI_MODE.EVEN	
	C/C++	TWXA_SCI_ODD	奇数パリティを使用します。
	C++	TWXA::SCI_MODE::ODD	
	VB/VBA	TWXA_SCI_MODE.ODD	
	C#	TWXA.SCI_MODE.ODD	
ストップビット	C/C++	TWXA_SCI_STOP1	ストップビットを 1 ビットとします(デフォルト)。
	C++	TWXA::SCI_MODE::STOP1	
	VB/VBA	TWXA_SCI_MODE.STOP1	
	C#	TWXA.SCI_MODE.STOP1	
	C/C++	TWXA_SCI_STOP2	ストップビットを 2 ビットとします。
	C++	TWXA::SCI_MODE::STOP2	
	VB/VBA	TWXA_SCI_MODE.STOP2	
	C#	TWXA.SCI_MODE.STOP2	

表 41　TWXA_SCISetMode() の Baud 引数に指定する値

言語	値	説明
C/C++	TWXA_SCI_BAUD300	ボーレートを 300bps にします。
C++	TWXA::SCI_BAUD::BAUD300	
VB/VBA	TWXA_SCI_BAUD.BAUD300	
C#	TWXA.SCI_BAUD.BAUD300	
C/C++	TWXA_SCI_BAUD600	ボーレートを 600bps にします。
C++	TWXA::SCI_BAUD::BAUD600	
VB/VBA	TWXA_SCI_BAUD.BAUD600	
C#	TWXA.SCI_BAUD.BAUD600	
C/C++	TWXA_SCI_BAUD1200	ボーレートを 1200bps にします。
C++	TWXA::SCI_BAUD::BAUD1200	
VB/VBA	TWXA_SCI_BAUD.BAUD1200	
C#	TWXA.SCI_BAUD.BAUD1200	
C/C++	TWXA_SCI_BAUD2400	ボーレートを 2400bps にします。
C++	TWXA::SCI_BAUD::BAUD2400	
VB/VBA	TWXA_SCI_BAUD.BAUD2400	
C#	TWXA.SCI_BAUD.BAUD2400	
C/C++	TWXA_SCI_BAUD4800	ボーレートを 4800bps にします。
C++	TWXA::SCI_BAUD::BAUD4800	
VB/VBA	TWXA_SCI_BAUD.BAUD4800	
C#	TWXA.SCI_BAUD.BAUD4800	
C/C++	TWXA_SCI_BAUD9600	ボーレートを 9600bps にします。
C++	TWXA::SCI_BAUD::BAUD9600	
VB/VBA	TWXA_SCI_BAUD.BAUD9600	
C#	TWXA.SCI_BAUD.BAUD9600	
C/C++	TWXA_SCI_BAUD14400	ボーレートを 14400bps にします。
C++	TWXA::SCI_BAUD::BAUD14400	
VB/VBA	TWXA_SCI_BAUD.BAUD14400	
C#	TWXA.SCI_BAUD.BAUD14400	
C/C++	TWXA_SCI_BAUD19200	ボーレートを 19200bps にします。
C++	TWXA::SCI_BAUD::BAUD19200	
VB/VBA	TWXA_SCI_BAUD.BAUD19200	
C#	TWXA.SCI_BAUD.BAUD19200	
C/C++	TWXA_SCI_BAUD38400	ボーレートを 38400bps にします。
C++	TWXA::SCI_BAUD::BAUD38400	
VB/VBA	TWXA_SCI_BAUD.BAUD38400	
C#	TWXA.SCI_BAUD.BAUD38400	

シリアルポートの使用手順

1. *TWXA_SCISetMode()* 関数で通信設定を行います。
2. 必要があれば *TWXA_SCISetDelimiter()* 関数でデリミタコードを設定します。
3. データ送信には *TWXA_SCIWrite()* 関数を使用します。
4. 受信データ数やエラーを調べるには *TWXA_SCIReadStatus()* 関数を使用します。
5. データを受信するには *TWXA_SCIRead()* 関数を使用します。

リスト 15　シリアルポートの使用例(C 言語)

```
char cRecv[255];
char cSend[] = "Hello¥r¥nWorld¥r¥n"; //送信文字列
long L;

//シリアル 0 とシリアル 1 を設定(Mode はデフォルト設定)
TWXA_SCISetMode(hDev, 0, 0, TWXA_SCI_BAUD9600);
TWXA_SCISetMode(hDev, 1, 0, TWXA_SCI_BAUD9600);

//シリアル 1 のデリミタを CR+LF に設定
TWXA_SCISetDelimiter(hDev, 1, "¥r¥n", 2);

//0 チャンネルから文字列を送信
TWXA_SCIWrite(hDev, 0, cSend, (long)strlen(cSend));

while(1){
  //受信数を調べる
  TWXA_SCIReadStatus(hDev, 1, NULL, &L);
  if(L == 0) break;

  //受信データを読み出す
  TWXA_SCIRead(hDev, 1, cRecv, L, NULL);
  OutputDebugStringA(cRecv);
}
```

リスト 16　シリアルポートの使用例(Visual Basic)

```
Dim str As String
Dim bBuff(254) As Byte
Dim i As Integer

'送信文字列
str = "Hello" & vbCrLf & "World" & vbCrLf

'シリアル 0 とシリアル 1 を設定
TWXA_SCISetMode(hDev, 0, 0, TWXA_SCI_BAUD.BAUD9600)
TWXA_SCISetMode(hDev, 1, 0, TWXA_SCI_BAUD.BAUD9600)

'シリアル 1 のデリミタを CR+LF に設定
TWXA_SCISetDelimiter(hDev, 1, vbCrLf, 2)

'0 チャンネルから文字列を送信
TWXA_SCIWrite(hDev, 0, str, str.Length)

Do
  '受信数を調べる
  TWXA_SCIReadStatus(hDev, 1, Nothing, i)
  If i = 0 Then Exit Do

  '受信データを読み出して文字列に変換
  TWXA_SCIRead(hDev, 1, bBuff, i, i)
  str = System.Text.Encoding.GetEncoding(932).GetString(bBuff, 0, i)
  Debug.WriteLine(str)
Loop
```

リスト 17　シリアルポートの使用例(VBA)

```
Dim str As String
Dim bBuff(254) As Byte
Dim bSend() As Byte
Dim L As Long

'送信文字列
str = "Hello" & vbCrLf & "World" & vbCrLf

'シリアル0とシリアル1を設定
TWXA_SCISetMode hDev, 0, 0, TWXA_SCI_BAUD.BAUD9600
TWXA_SCISetMode hDev, 1, 0, TWXA_SCI_BAUD.BAUD9600

'シリアル1のデリミタをCR+LFに設定
bBuff(0) = &HD 'CR
bBuff(1) = &HA 'LF
TWXA_SCISetDelimiter hDev, 1, bBuff(0), 2

'0チャンネルから文字列を送信
bSend = StrConv(str, vbFromUnicode)
TWXA_SCIWrite hDev, 0, bSend(0), Len(str)

Do
  '受信数を調べる
  TWXA_SCIReadStatus hDev, 1, bBuff(0), L

  If L = 0 Then Exit Do

  '受信データを読み出して文字列に変換
  TWXA_SCIRead hDev, 1, bBuff(0), L, L
  bBuff(L) = 0

  Debug.Print StrConv(bBuff(), vbUnicode)
Loop
```

リスト 18　シリアルポートの使用例(C#)

```
byte[] bBuff = new byte[255];
string str = "Hello¥r¥nWorld¥r¥n"; //送信文字列
int i;
byte b;

//シリアル 0 とシリアル 1 を設定(Mode はデフォルト設定)
TWXA.SCISetMode(hDev, 0, 0, TWXA.SCI_BAUD.BAUD9600);
TWXA.SCISetMode(hDev, 1, 0, TWXA.SCI_BAUD.BAUD9600);

//シリアル 1 のデリミタを CR+LF に設定
TWXA.SCISetDelimiter(hDev, 1, "¥r¥n", 2);

//0 チャンネルから文字列を送信
TWXA.SCIWrite(hDev, 0, str, str.Length);

while (true)
{
  //受信数を調べる
  TWXA.SCIReadStatus(hDev, 1, out b, out i);
  if (i == 0) break;

  //受信データを読み出す
  TWXA.SCIRead(hDev, 1, bBuff, i, out i);
  str = System.Text.Encoding.GetEncoding(932).GetString(bBuff, 0, i);
  Debug.WriteLine(str);
}
```

□ ユーザーステータスレジスタ／ユーザーメモリの利用

パソコン上のアプリケーションプログラムを終了させても、デバイスがどのような状態にあるかを記憶しておき、次にアプリケーションプログラムを実行したときに、その続きから制御を行いたい場合があります。このようなときにユーザーステータスレジスタとユーザーメモリが利用できます。

ユーザーステータスレジスタはデバイス内の 1 バイトのメモリで、デバイスの起動時や再初期化のときには必ず 0 にクリアされます。ユーザーステータスレジスタを利用して、デバイスが初期化済みであるか、どのような状態にあるか、といった簡単な情報を保存しておくことができます。

ユーザーメモリはデバイスの RAM に確保された 10K バイトのメモリ空間です。ユーザーステータスレジスタでは保存できない比較的大きな設定情報などを記憶することができます。この領域の値は起動時には不定となり、自動的にクリアされることもありませんのでユーザーステータスレジスタと組み合わせて使用してください。ユーザーメモリのアドレスはデバイス上の H'FFBF20～H'FFE71F の範囲です。

表 42　ユーザーステータスレジスタ／ユーザーメモリの操作に使用する関数

関数名	説明
TWXA_PortWrite()	ユーザーステータスレジスタにデータを書き込みます。
TWXA_PortRead()	ユーザーステータスレジスタからデータを読み出します。
TWXA_PortBWrite()	ユーザーメモリにデータを書き込みます。
TWXA_PortBRead()	ユーザーメモリからデータを読み出します。

ユーザーステータスレジスタの操作方法

入出力ポートなどと同様に *TWXA_PortWrite()*、*TWXA_PortRead()* 関数を使用して、書込み、読出しが行えます。*Port* 引数には表 43 の値を指定してください。

表 43　ユーザーステータスレジスタを指定する定数

言語	値	説明
C/C++	TWXA_USER_STATUS	ユーザーステータスレジスタを変更します。
C++	TWXA::WPORT::USER_STATUS	
VB/VBA	TWXA_WPORT.USER_STATUS	
C#	TWXA.WPORT.USER_STATUS	

ユーザーメモリの操作方法

TWXA_PortBRead()、*TWXA_PortBWrite()* 関数を使用すると、大きなデータを効率良くリード／ライトできます。これらの関数では *Port* 引数にアドレス、*nData* 引数にバイト数を指定してデバイス上の任意のメモリアドレスにアクセスできます。

表 44　TWXA_PortBWrite() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_PortBWrite(TW_HANDLE hDev, DWORD Port, void *pData, long nData)
VB	Function TWXA_PortBWrite(ByVal hDev As System.IntPtr, ByVal Port As Integer, ByVal pData As Object, ByVal nData As Integer) As Integer
VBA	Function TWXA_PortBWrite(ByVal hDev As Long, ByVal Port As Long, ByRef pData As Any, ByVal nData As Long) As Long
C#	STATUS PortBWrite(System.IntPtr hDev, uint Port, object pData, int nData)

表 45　TWXA_PortBRead() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_PortBRead(TW_HANDLE hDev, DWORD Port, void *pData, long nData)
VB	Function TWXA_PortBRead(ByVal hDev As System.IntPtr, ByVal Port As Integer, ByVal pData As Object, ByVal nData As Integer) As Integer
VBA	Function TWXA_PortBRead(ByVal hDev As Long, ByVal Port As Long, ByRef pData As Any, ByVal nData As Long) As Long
C#	STATUS PortBRead(System.IntPtr hDev, uint Port, object pData, int nData)

- ユーザーメモリ以外の領域に対して読み書きを行うと、誤動作する場合があります。
- ユーザーメモリはユーザーファームの動作にも使用します。ユーザーファーム利用時には自由に使用できる領域が変化しますので誤って操作しないように特に注意が必要です。

□ フラッシュメモリの利用

製品にはフラッシュメモリが内蔵されています。フラッシュメモリは電源を切っても記録した情報が保存される不揮発性のメモリ空間で、製品が動作するためのファームウェアもこの領域に書き込まれています。図 33 はフラッシュメモリ領域を詳しく示した図です。

フラッシュメモリは消去単位毎に EB0～EB15 の 16 ブロックに分けて管理されます。このうち、EB1～EB3 の 12K バイトの領域がユーザーに開放されています。電源を切っても内容が消えないため、アプリケーション固有の設定情報やキャリブレーションデータの保存などに利用可能です。

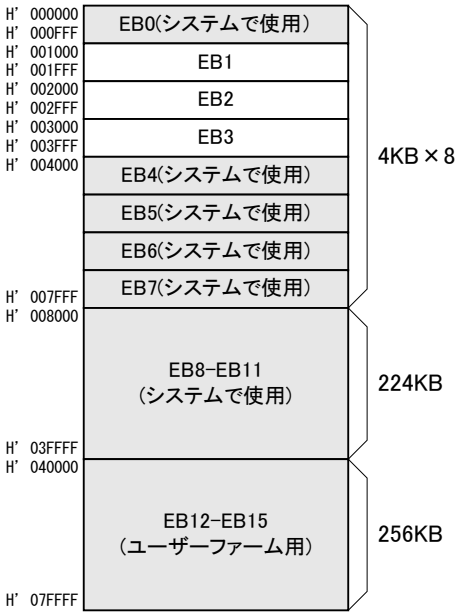

図 33　フラッシュメモリマップ

表 46　フラッシュメモリの操作に使用する関数

関数名	説明
TWXA_FlashAttachWriter()	フラッシュメモリの消去／書込みのためのファームウェアをデバイスにダウンロードします。
TWXA_FlashEraseBlk()	フラッシュメモリの指定ブロックを消去します。
TWXA_FlashWrite()	フラッシュメモリに書込みを行います。
TWXA_PortBRead()	フラッシュメモリからデータを読み出します。

表 47　フラッシュメモリ操作のサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	FlashSample	フラッシュメモリの状態表示、ファイルデータのフラッシュメモリへの書込みを行います。
Visual Basic	FlashSampleVB	
Visual C#	FlashSampleCS	
VBA (Excel)	FlashSample.xls	セルを利用した簡易バイナリエディタです。編集内容をフラッシュメモリに書き込むことができます。

フラッシュメモリへの書込み操作は特殊で、通常のメモリのように 1 バイト単位でデータを書き込むことはできません。書込みを行う領域には、まず消去の操作を行います。消去の単位は図 33 に示し

た EB1～EB3 のブロック単位で、消去対象のブロックは全ビットが"1"となります。

続いて、実際に保存するデータの書込みを行います。書込みは 128 バイト毎のブロック単位で行います。そのため、書込みの先頭アドレスは常に 128 バイト境界(アドレスの下位 7 ビットが 0)となります。

また、フラッシュメモリの消去／書込みを行う際には、予めフラッシュメモリを制御するためのファームウェアをデバイスにダウンロードする必要があります。このファームウェアはユーザーメモリ(54 ページ参照)にダウンロードされますので、ユーザーメモリは一時的に使用できなくなり、内部のデータも破壊されてしまいますので注意してください。

フラッシュメモリからの読出しは、ユーザーメモリなどと同様に行うことができます。

- TWXA ライブラリによるフラッシュメモリ操作を行うにはディップスイッチの 3 番を"ON"にする必要があります(通常モードを使用しますのでディップスイッチの 2 番は"OFF"のままにします)。
- フラッシュメモリの操作を行うとユーザーファームは停止します。再度動作させるには製品を再起動する必要があります。
- フラッシュメモリの書換え可能回数の目安は 100 回、データ保持年数は 10 年です。

フラッシュメモリの消去方法

1. *TWXA_FlashAttachWriter()* 関数を呼びます。
1. *TWXA_FlashEraseBlk()* 関数を呼び出します。*Blk* 引数に消去したいブロック番号(1～3)を指定します。

表 48　TWXA_FlashEraseBlk() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_FlashEraseBlk(TW_HANDLE hDev, long Blk)
VB	Function TWXA_FlashEraseBlk(ByVal hDev As System.IntPtr, ByVal Blk As Integer) As Integer
VBA	Function TWXA_FlashEraseBlk(ByVal hDev As Long, ByVal Blk As Long) As Long
C#	STATUS FlashEraseBlk(System.IntPtr hDev, int Blk)

フラッシュメモリへの書込み方法

1. *TWXA_FlashAttachWriter()* 関数を呼びます。
2. *TWXA_FlashWrite()* 関数(表 49)を呼び出します。*Address* 引数には書込み先のアドレスとして 0x1000～0x3f80 の値が指定できますが、128 バイト境界に合わせる必要がありますので、下位 7 ビットは常に 0 になります。また、*nData* 引数に指定する書込みバイト数も 128 の倍数としてください。

表 49　TWXA_FlashWrite() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_FlashWrite(TW_HANDLE hDev, DWORD Address, void *pData, DWORD nData)
VB	Function TWXA_FlashWrite(ByVal hDev As System.IntPtr, ByVal Address As Integer, ByVal pData As Object, ByVal nData As Integer) As Integer
VBA	Function TWXA_FlashWrite(ByVal hDev As Long, ByVal Address As Long, ByRef pData As Any, ByVal nData As Long) As Long
C#	STATUS FlashWrite(System.IntPtr hDev, uint Address, object pData, int nData)

リスト 19　フラッシュメモリの使用例(C 言語)

```
char cWrite[128] = "Hello World";
char cRead[128];

//ファームウェアのダウンロード
TWXA_FlashAttachWriter(hDev);

//ブロック１を消去
TWXA_FlashEraseBlk(hDev, 1);

//書込み
TWXA_FlashWrite(hDev, 0x1000, cWrite, 128);

//読出し
TWXA_PortBRead(hDev, 0x1000, cRead, 128);
OutputDebugStringA(cRead);
```

リスト 20　フラッシュメモリの使用例(Visual Basic)

```
Dim strWrite As New System.Text.StringBuilder("Hello World")
Dim bBuff(127) As Byte

strWrite.Length = 128

'ファームウェアのダウンロード
TWXA_FlashAttachWriter(hDev)

'ブロック１を消去
TWXA_FlashEraseBlk(hDev, 1)

'書込み
TWXA_FlashWrite(hDev, &H1000, strWrite, 128)

'読出し
TWXA_PortBRead(hDev, &H1000, bBuff, 128)
Debug.WriteLine(System.Text.Encoding.GetEncoding(932).GetString(bBuff, 0, 128))
```

リスト 21　フラッシュメモリの使用例(VBA)

```
Dim bBuff(127) As Byte
Dim bSend() As Byte

bSend = StrConv("Hello World", vbFromUnicode)
ReDim Preserve bSend(127)

'ファームウェアのダウンロード
TWXA_FlashAttachWriter hDev

'ブロック１を消去
TWXA_FlashEraseBlk hDev, 1

'書込み
TWXA_FlashWrite hDev, &H1000, bSend(0), 128

'読出し
TWXA_PortBRead hDev, &H1000, bBuff(0), 128
Debug.Print StrConv(bBuff(), vbUnicode)
```

リスト 22　フラッシュメモリの使用例(C#)

```
StringBuilder strWrite = new System.Text.StringBuilder("Hello World");
byte []bBuff = new byte[128];

strWrite.Length = 128;

//ファームウェアのダウンロード
TWXA.FlashAttachWriter(hDev);

//ブロック１を消去
TWXA.FlashEraseBlk(hDev, 1);

//書込み
TWXA.FlashWrite(hDev, 0x1000, strWrite, 128);

//読出し
TWXA.PortBRead(hDev, 0x1000, bBuff, 128);
Debug.WriteLine(System.Text.Encoding.GetEncoding(932).GetString(bBuff, 0, 128));
```

□ エラー処理

TWXAライブラリの関数のほとんどは戻り値で関数の実行結果を返します。本マニュアルのプログラム例は要点を分かりやすくするために、関数の戻り値チェックを省略していますが、実際のプログラムでは関数が正しく実行されたかどうかチェックすることを推奨します。

関数の戻り値についての詳細は「TWXA 関数リファレンス」を参照してください。

リスト 23　エラー処理の例(C 言語)

```
TW_STATUS ret;

ret = TWXA_PortWrite(hDev, TWXA_USER_STATUS, 0x00, 0xff);
if(ret){
  TWXA_Close(hDev);
  hDev = 0;
  printf("エラーが発生しました。TW_STATUS = %08X(HEX)", ret);
  return ret;
}
```

リスト 24　エラー処理の例(C++/MFC)

```
CString str;
TW_STATUS ret;

ret = TWXA_PortWrite(hDev, TWXA::WPORT::USER_STATUS, 0x00);
if(ret){
  TWXA_Close(hDev);
  hDev = 0;
  str.Format(_T("エラーが発生しました。TW_STATUS = %08X(HEX)"), ret);
  AfxMessageBox(str);
  return ret;
}
```

リスト 25　エラー処理の例(Visual Basic)

```
Dim ret As Integer

ret = TWXA_PortWrite(hDev, TWXA_WPORT.USER_STATUS, &H0)
If ret <> TW_STATUS.TW_OK Then
  TWXA_Close(hDev)
  hDev = System.IntPtr.Zero
  MsgBox(String.Format("エラーが発生しました。TW_STATUS = {0:X8}(HEX)", ret))
  Exit Sub
End If
```

リスト 26　エラー処理の例(VBA)

```
Dim ret As Long

ret = TWXA_PortWrite(hDev, TWXA_WPORT.USER_STATUS, &H0)
If ret <> TW_STATUS.TW_OK Then
  TWXA_Close hDev
  hDev = 0
  MsgBox "エラーが発生しました。TW_STATUS = " & Hex(ret) & "(HEX)"
  Exit Sub
End If
```

リスト 27　エラー処理の例(C#)

```
TWXA.STATUS ret;

ret = TWXA.PortWrite(hDev, TWXA.WPORT.USER_STATUS, 0);
if (ret != TWXA.STATUS.TW_OK)
{
  TWXA.Close(hDev);
  hDev = System.IntPtr.Zero;
  MessageBox.Show(string.Format("エラーが発生しました。TW_STATUS = {0:X8}(HEX)", ret));
  return ret;
}
```

Appendix

□ 製品の応答時間

ライブラリ関数の呼び出しに対する応答時間は使用環境によって影響を受けますので一定ではありません。特に実行プロセスやスレッドの切り替えが起こった場合には、関数の実行に 10msec 以上の時間がかかる場合もありますのでご注意ください。

図 34 は参考として異なる規格の USB ポートとの接続に対して *TWXA_ADRead()* 関数呼び出しを 1000 回ずつ行い、関数実行に要した時間をプロットしたものです。

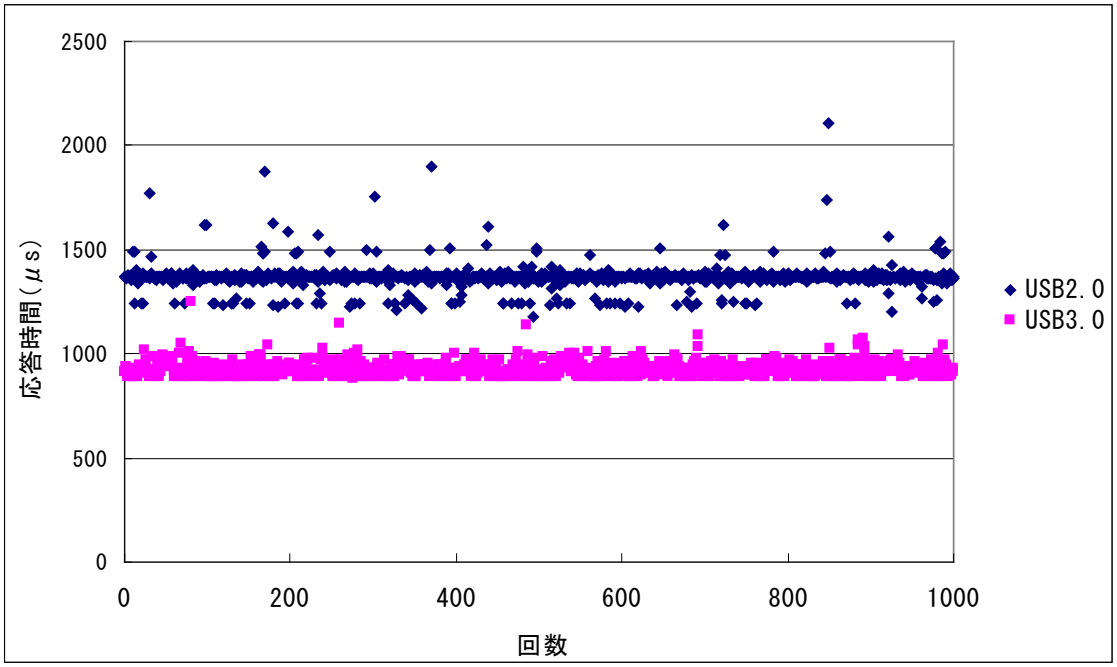

図 34　TWXA_ADRead() 関数の応答時間

保証期間

本製品の保証期間は、お買い上げ日より 1 年間です。保証期間中の故障につきましては、無償修理または代品との交換で対応させていただきます。ただし、以下の場合は保証期間内であっても有償での対応とさせていただきますのでご了承ください。

1) 本マニュアルに記載外の誤った使用方法による故障。
2) 火災、震災、風水害、落雷などの天災地変および公害、塩害、ガス害などによる故障。
3) お買い上げ後の輸送、落下などによる故障。

サポート情報

製品に関する情報、最新のファームウェア、ユーティリティなどは弊社ホームページにてご案内しております。また、お問い合わせ、ご質問などは下記までご連絡ください。

テクノウェーブ(株)
URL : http://www.techw.co.jp
E-mail : support@techw.co.jp

(1) 本書、および本製品のホームページに掲載されている応用回路、プログラム、使用方法などは、製品の代表的動作・応用例を説明するための参考資料です。これらに起因する第三者の権利(工業所有権を含む)侵害、損害に対し、弊社はいかなる責任も負いません。

(2) 本書の内容の一部または全部を無断転載することをお断りします。

(3) 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。

(4) 本書の内容については、万全を期して作成いたしましたが、万一ご不審な点や誤り、記載もれなど、お気づきの点がございましたらご連絡ください。

改訂記録

年月	版	改訂内容
2016 年 3 月	初	
2018 年 4 月	2	・ドライバファイルの更新に伴いインストール手順を修正

Ⓒ テクノウェーブ株式会社 2016-2018