USBX-I0800 / USBX-I0404 / USBX-I0008
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。

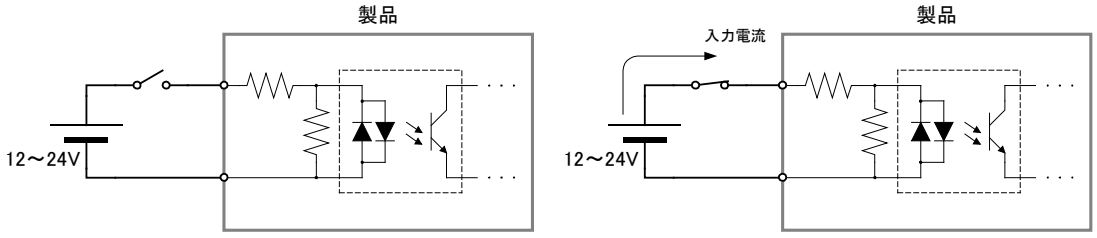
表 1　電気的状態の表記方法

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。

接点入力端子の状態

接点入力端子は十分な入力電圧が印加され電流が流れている状態を″ON″、入力電流が流れていないか十分でない場合を″OFF″とします。

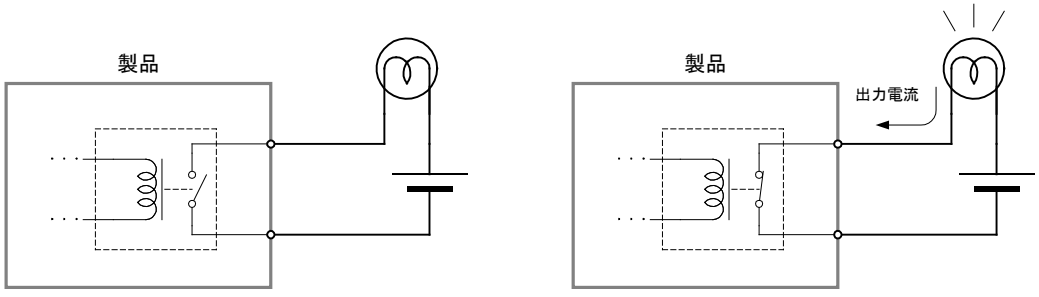


入力端子の″OFF″状態　　入力端子の″ON″状態

図 1　接点入力端子の"OFF"状態と"ON"状態

接点出力端子の状態

接点出力端子はリレー接点が閉じている状態を″ON″、開いている状態を″OFF″とします。



出力端子の″OFF″状態　　出力端子の″ON″状態

図 2　接点出力端子の"OFF"状態と"ON"状態

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。

"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。

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。

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-I0800』/『USBX-I0404』/『USBX-I0008』(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、デジタル I/O、パルスカウンタ、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。

- **接点入出力**[1] – 『USBX-I0800』はフォトカプラ絶縁入力 8 点、『USBX-I0404』はリレー出力 4 点、フォトカプラ絶縁入力 4 点、『USBX-I0008』はリレー出力 8 点を備えています。
- **32 ビットパルスカウンタ**[1] – 『USBX-I0800』と『USBX-I0404』は最大 4 チャンネルの単相 32 ビットソフトウェアカウンタを使用可能です。
- **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。
- **ハードウェアイベントの監視** – 『USBX-I0800』と『USBX-I0404』はパルスカウンタ入力を監視し、指定された条件となった場合に Windows® 上のアプリケーションにメッセージで通知する機能を備えています。
- **パルス出力**[1] – 『USBX-I0404』と『USBX-I0008』はパルスを自動的に出力する機能を備えています。
- 制御用 API は DLL モジュールで提供され、Visual C++® や Visual Basic® などで作成された 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 接点入出力、パルス出力、カウンタは一部の端子、ハードウェア機構を共有しているため、組み合わせにより同時使用できない場合があります。

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。

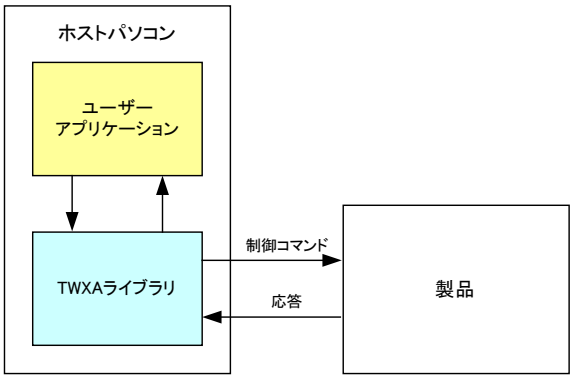


図 3　ホストパソコンからの制御

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。

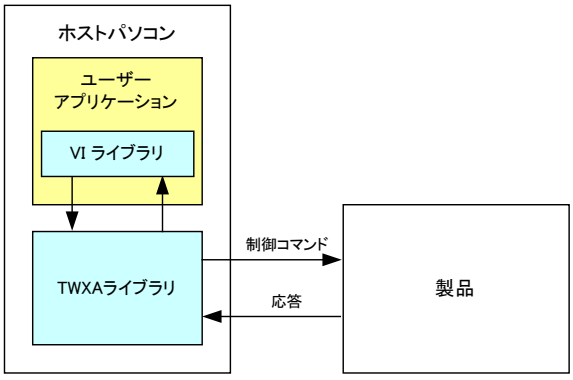


図 4　LabVIEW での利用

ファームウェアの開発

TWXA ライブラリの各関数は図 5 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。

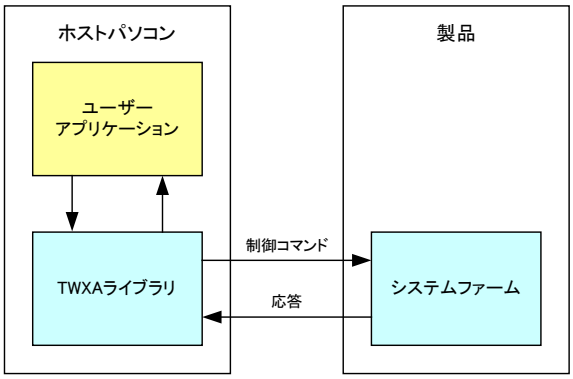


図 5　ホストパソコンからの制御

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 6)。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-I0800/X-I0404/X-I0008 ユーザーファーム開発マニュアル」を参照してください。

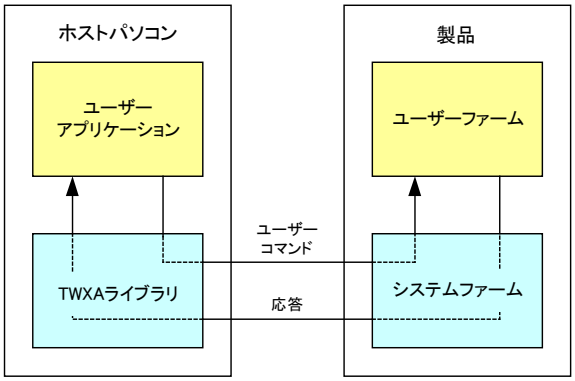


図 6　ユーザーファームの追加

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-I0800/USBX-I0404/USBX-0008 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-I0x0x.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-I0800/X-I0404/X-I0008 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-I0x0xUserFirm.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 バスパワーによる動作
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	Hi-Speed 対応
対応 OS	Windows XP, Vista, 7, 8, 8.1, 10	32bit, 64bit

表 8　接点入力仕様(USBX-I0800/USBX-I0404 のみ)

項目		仕様	備考
入力点数	USBX-I0800	最大 8 点	4 点はパルスカウンタ機能付き
	USBX-I0404	最大 4 点	
入力方式		電圧入力	
絶縁方式		フォトカプラ	PS2801-4 相当品
入力電圧範囲		0～25.2[V]	Ia 端子間電圧、Ic 端子間電圧 無極性
入力抵抗		3.6kΩ	入力回路図参照
入力オン電圧		11.4～25.2[V]	Ia 端子間電圧、Ic 端子間電圧
フォトカプラ応答速度		最大 100[μsec]	
絶縁抵抗	対シャーシ	1000[MΩ]以上	測定条件:500VDC
	対 SG(内部回路)	1000[MΩ]以上	
	対電源端子	1000[MΩ]以上	
絶縁耐圧	対シャーシ	2500[Vrms]	測定条件: カットオフ電流 10mA、1 分間
	対 SG(内部回路)	2500[Vrms]	
	対電源端子	2500[Vrms]	

表 9　接点出力仕様(USBX-I0404/USBX-I0008 のみ)

項目		仕様	備考
出力点数	USBX-I0404	最大 4 点	2 点はパルス出力機能付き
	USBX-I0008	最大 8 点	
出力方式		無電圧接点出力	
絶縁方式		リレー	V23079A1001B301 相当品
最大接点電圧		100[VDC]、100[VAC]	
最大電力		60[VA]	
最大電流		2[A]	
動作時間		最大 4[msec]	
復帰時間		最大 4[msec]	
絶縁抵抗	対シャーシ	1000[MΩ]以上	測定条件:500VDC
	対 SG(内部回路)	1000[MΩ]以上	
	対電源端子	1000[MΩ]以上	
絶縁耐圧	対シャーシ	1500[Vrms]	測定条件: カットオフ電流 10mA、1 分間
	対 SG(内部回路)	1500[Vrms]	
	対電源端子	1500[Vrms]	

表 10　パルス出力仕様(USBX-I0404/USBX-I0008 のみ)

項目	仕様	備考
出力チャンネル数	最大 2 チャンネル	Of0、Of1
出力周期	20[msec]～65[sec]	
周期/デューティ分解能	1[msec]	
パルス幅ひずみ	最大 4[msec]	

表 11　パルスカウンタ(ソフトウェアカウンタ)仕様(USBX-I0800/USBX-I0404 のみ)

項目	仕様	備考
入力チャンネル数	最大 4 チャンネル	Ic0～Ic3
カウンタビット数	32 ビット	
カウントタイミング	入力が OFF から ON に変化時	
周波数	最大 5[kHz]	

表 12　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]	

- パルス出力は接点出力端子の Of0、Of1 を使用します。どちらか片方の機能しか利用できません。
- カウンタ入力は接点入力端子の Ic0～Ic3 を使用します。どちらか片方の機能しか利用できません。

[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

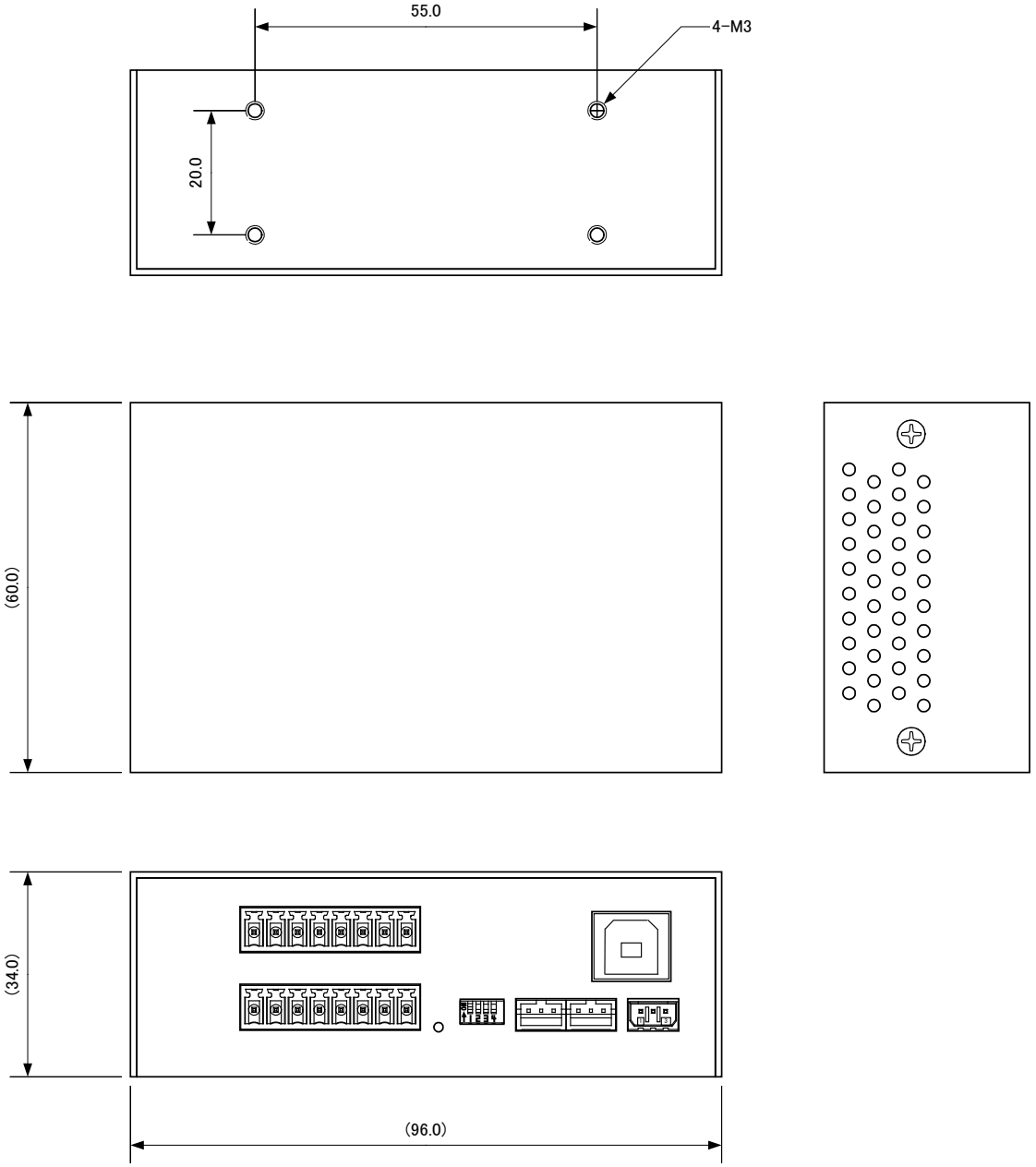


図 7　外形寸法図

□ USBX-I0800 各部の名称と説明

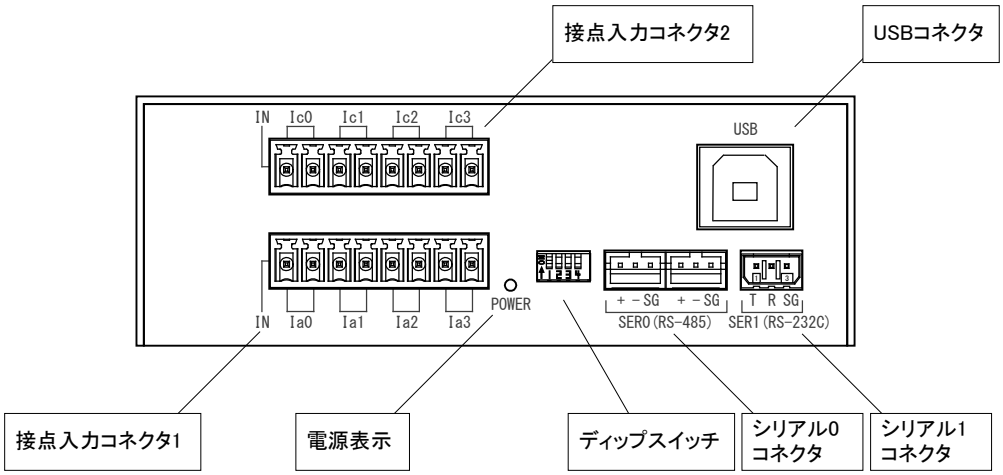


図 8 USBX-I0800 各部の名称

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 19 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。

接点入力コネクタ 1
デジタル信号の入力端子です。適合コネクタは「EC350RL」(DINKLE)です。

接点入力コネクタ 2
デジタル信号の入力端子です。パルスカウンタ入力としても使用可能です。適合コネクタは「EC350R」(DINKLE)です。

□ USBX-I0404 各部の名称と説明

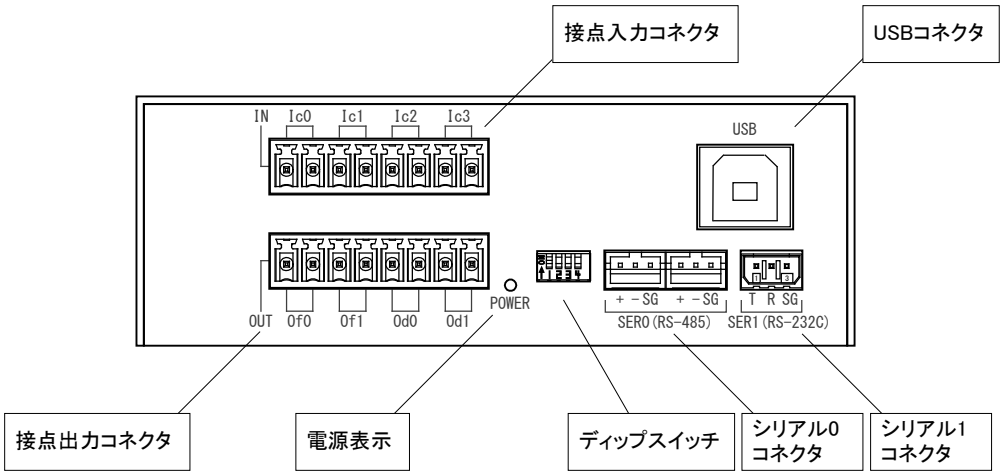


図 9 USBX-I0404 各部の名称

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 19 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。

接点出力コネクタ
デジタル信号の出力端子です。Of0、Of1 は自動でパルス出力を行う機能があります。適合コネクタは「EC350RL」(DINKLE)です。

接点入力コネクタ
デジタル信号の入力端子です。パルスカウンタ入力としても使用可能です。適合コネクタは「EC350R」(DINKLE)です。

□ USBX-I0008 各部の名称と説明

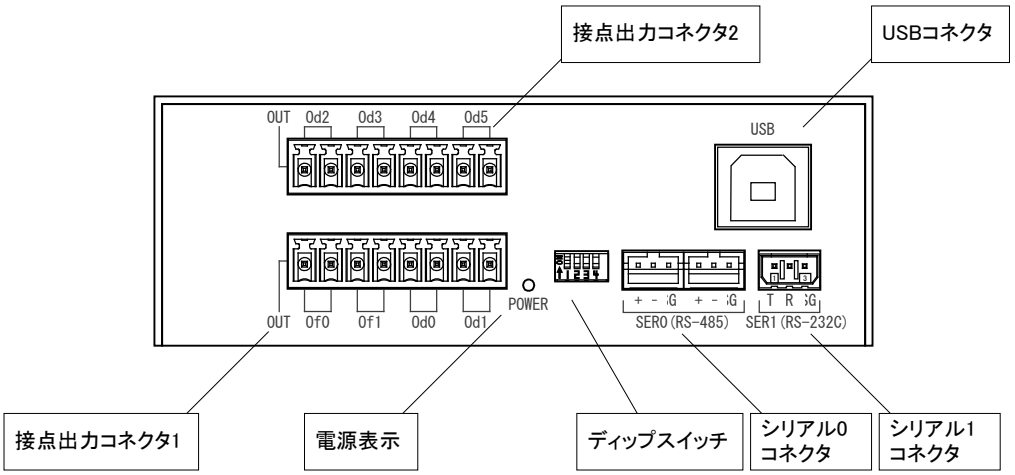


図 10　USBX-I0008 各部の名称

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 19 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。

接点出力コネクタ 1
　デジタル信号の出力端子です。Of0、Of1 は自動でパルス出力を行う機能があります。適合コネクタは「EC350RL」(DINKLE)です。

接点出力コネクタ 2
　デジタル信号の出力端子です。適合コネクタは「EC350R」(DINKLE)です。

□ ディップスイッチ

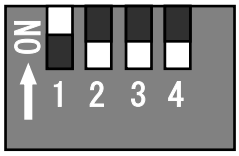


図 11　ディップスイッチ

表 13　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 12 の向きで製品に取り付けます。製品は図 13 の向きになるように固定してください。

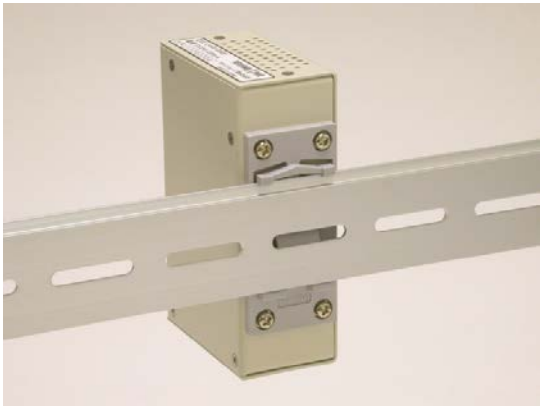


図 12　DIN レール取付具の取付け

図 13　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 14 参照)を挿入し再びスクリューを締めて固定します。

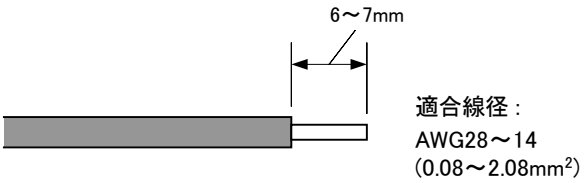


図 14　電線の加工

□ 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 14　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 15　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。

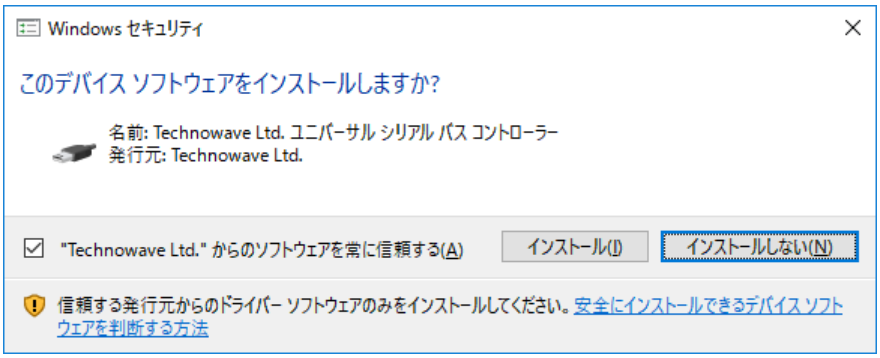


図 16　Windows 10 のドライバインストール画面（2）

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 17　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。

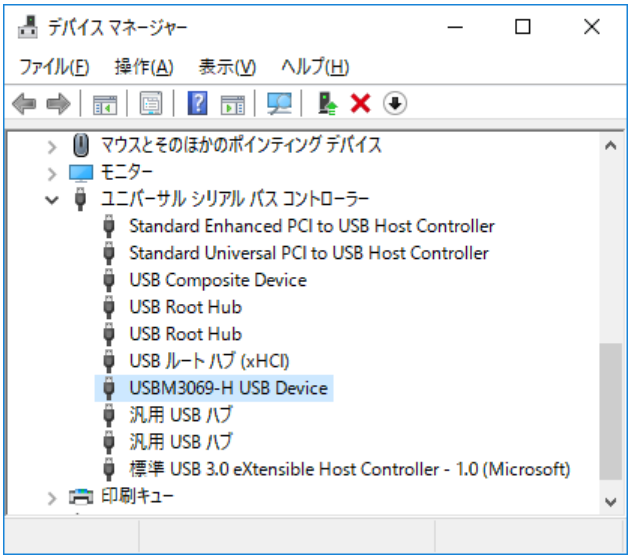


図 18　Windows 10 のドライバインストール確認

- 「デバイスマネージャ」を表示するには[スタート]を右クリックし、表示されたリストの中から［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。

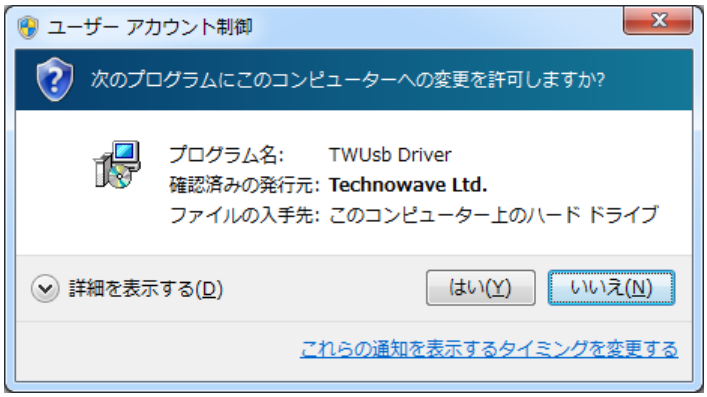


図 19　Windows 7 のドライバインストール画面（1）

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。

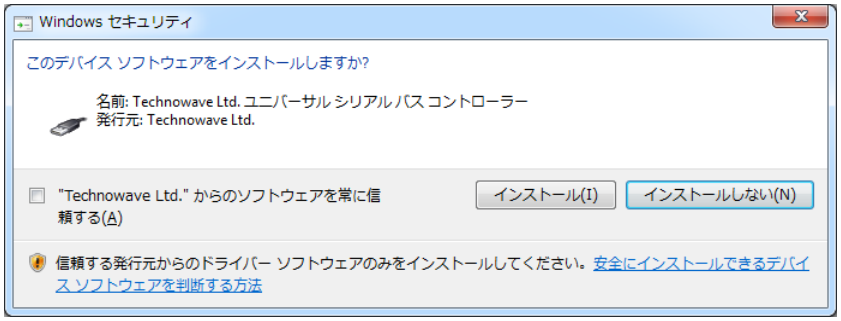


図 20　Windows 7 のドライバインストール画面（2）

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

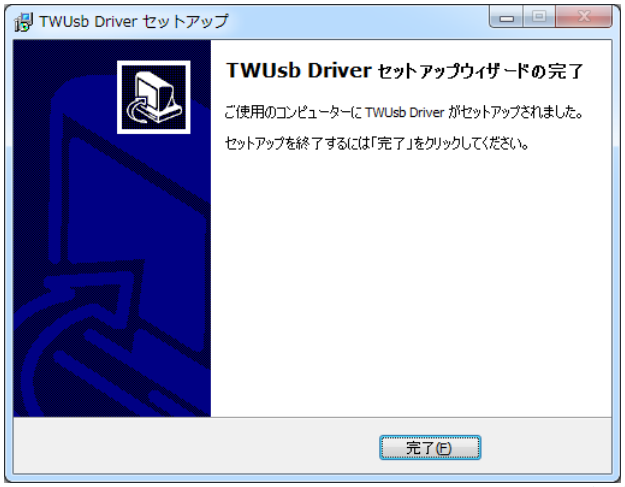


図 21　Windows 7 のドライバインストール画面（3）

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。

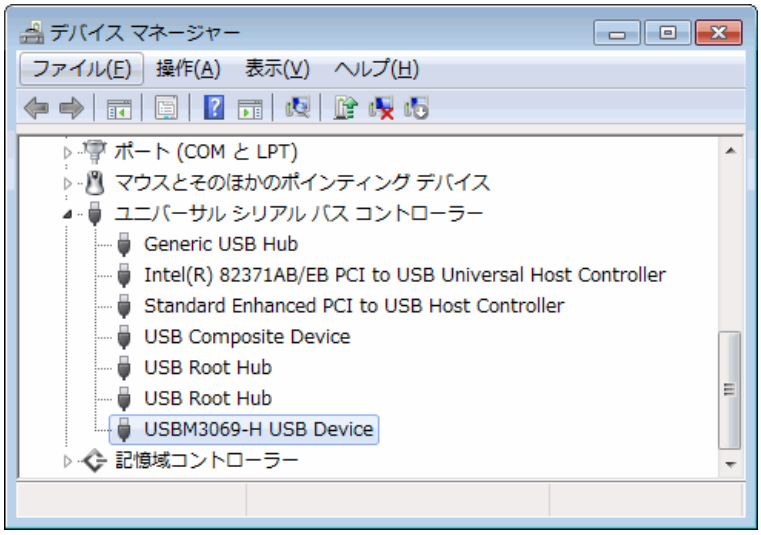


図 22　Windows 7 のドライバインストール確認

- 「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。

Windows XP の場合

① 「setup.exe」を起動し、画面の指示に従ってインストールを行います。

② 下のような画面が表示されたら[続行]ボタンを押してインストールを続けてください。

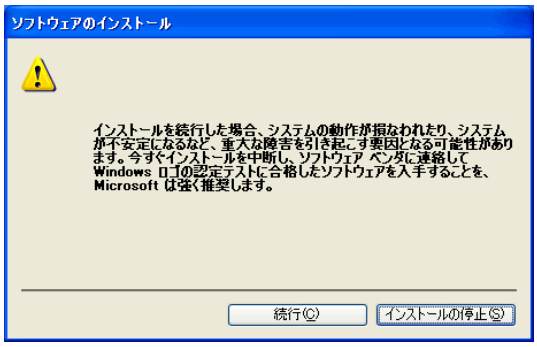


図 23　Windows XP のドライバインストール画面（1）

③ インストールが終了すると、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 24　Windows XP のドライバインストール画面（2）

④ デバイスを USB ケーブルでパソコンに接続すると、図のような画面が表示されますので、[いいえ、今回は接続しません]を選択し、[次へ]のボタンを押します。

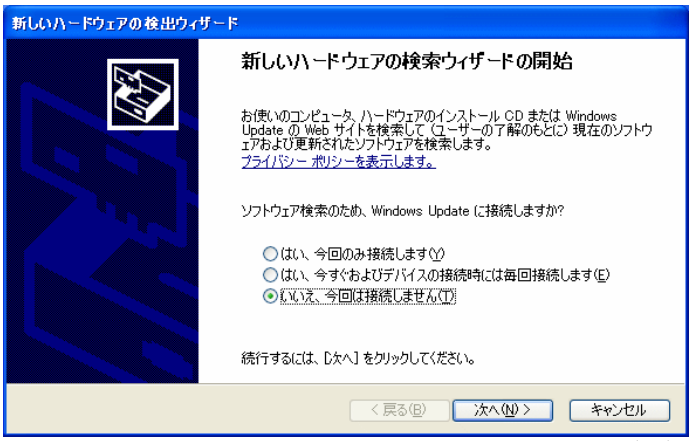


図 25　Windows XP のドライバインストール画面（3）

⑤ 図のような画面が表示されますので、[ソフトウェアを自動的にインストールする]を選択し、[次へ]のボタンを押します。

図 26　Windows XP のドライバインストール画面（4）

⑥ 下のような画面が表示されたら[続行]ボタンを押してインストールを続けてください。

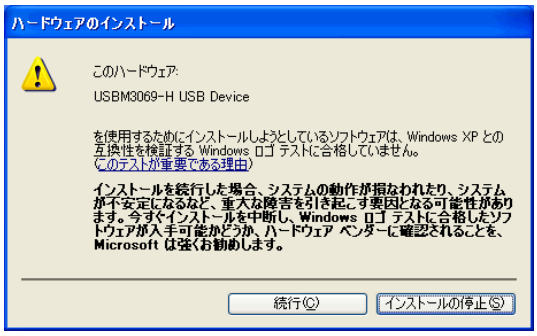


図 27　Windows XP のドライバインストール画面（5）

⑦　図のような画面が表示されますので、[完了]ボタンを押します。

図 28　Windows XP のドライバインストール画面（6）

⑧　図 29 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。

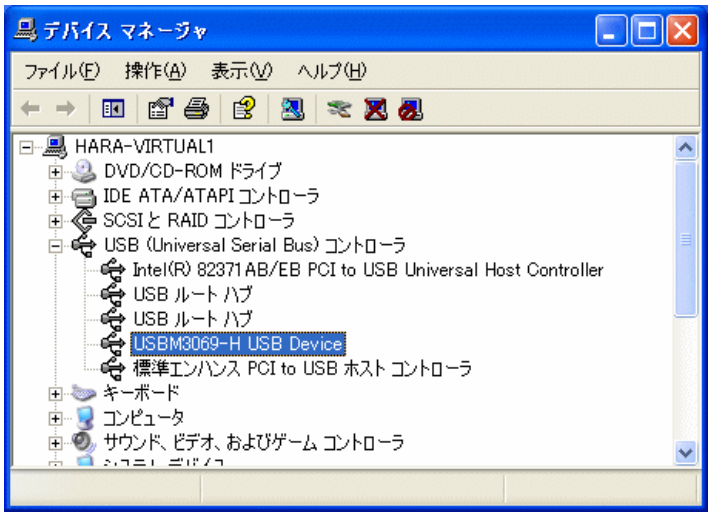


図 29　Windows XP のドライバインストール確認

- 「デバイスマネージャ」を表示するには[マイ コンピュータ]を右クリックし、[プロパティ]を選択します。[システムのプロパティ]画面が表示されますので、[ハードウェア]タブから[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-I0x0xTools」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 15 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 15　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。

□ 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。

[5] ドライバのインストールは必要です。

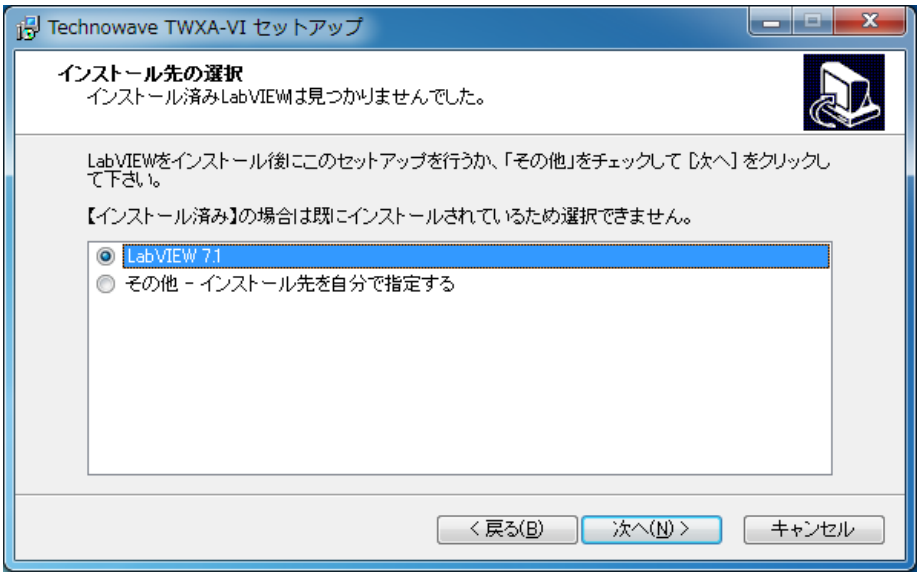


図 30　VI ライブラリのセットアップ画面

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

27 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-I0x0xTools]から起動することができます。

図 31　設定ツールのメニュー画面

表 16　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。
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。

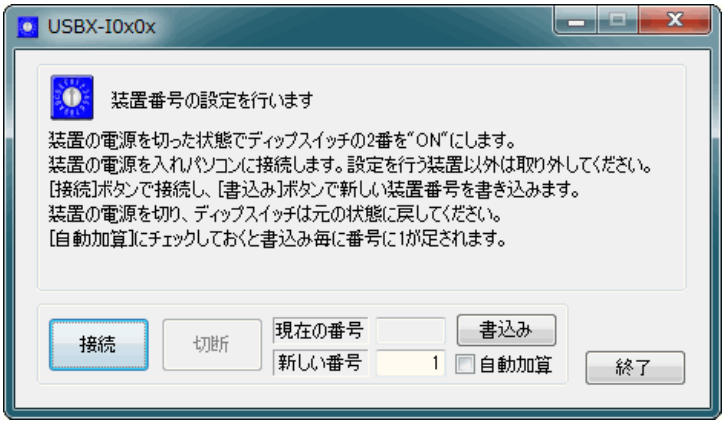


図 32　番号設定ツールの画面

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(29 ページ)から[装置番号設定ツール]ボタンを押します。図 32 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。

5. ハードウェア

□ 接点入力(USBX-I0800/USBX-I0404 のみ)

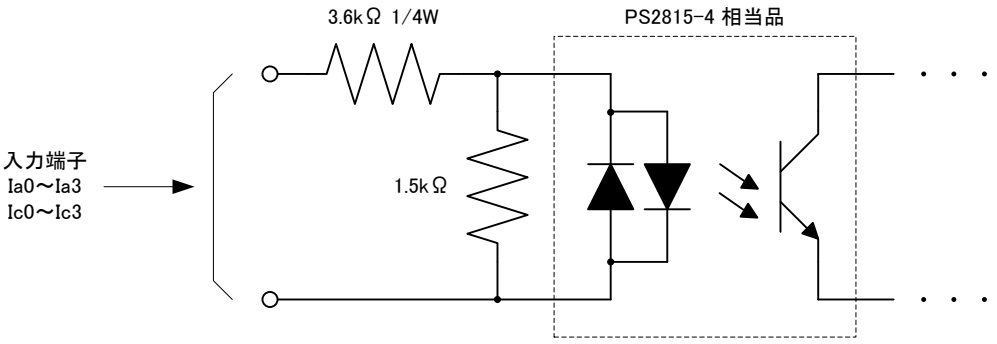


図 33　接点入力回路

図 34 のように各入力の 2 つの端子間に 12～24V の電圧をかけると"ON"となります。極性はありませんので、どちらの端子が＋側でも構いません。

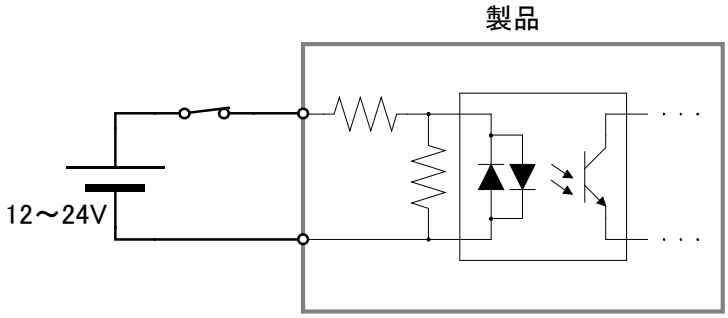


図 34　接点入力の接続例

□ 接点出力(USBX-I0404/USBX-I0008 のみ)

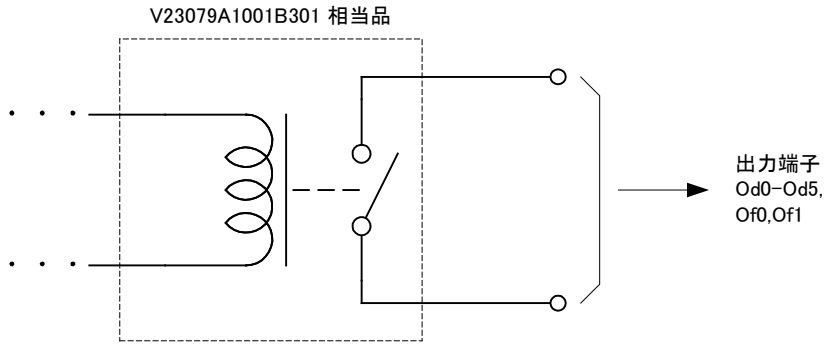


図 35　接点出力回路

□ シリアル 0 (RS-485)

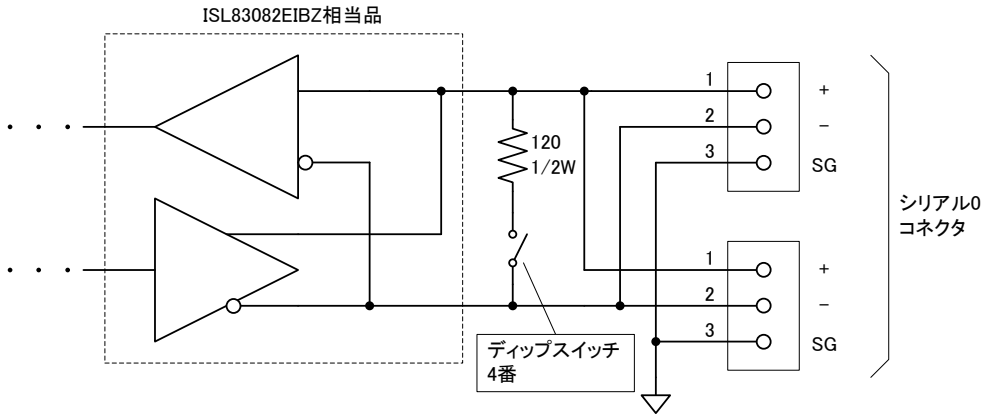


図 36　シリアル 0 の入出力回路

図 37、図 38 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。

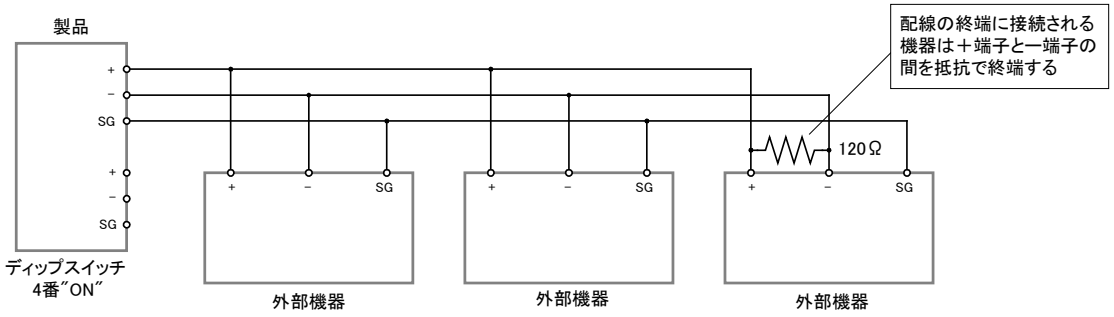


図 37　RS-485 の接続例(製品が配線の終端にある場合)

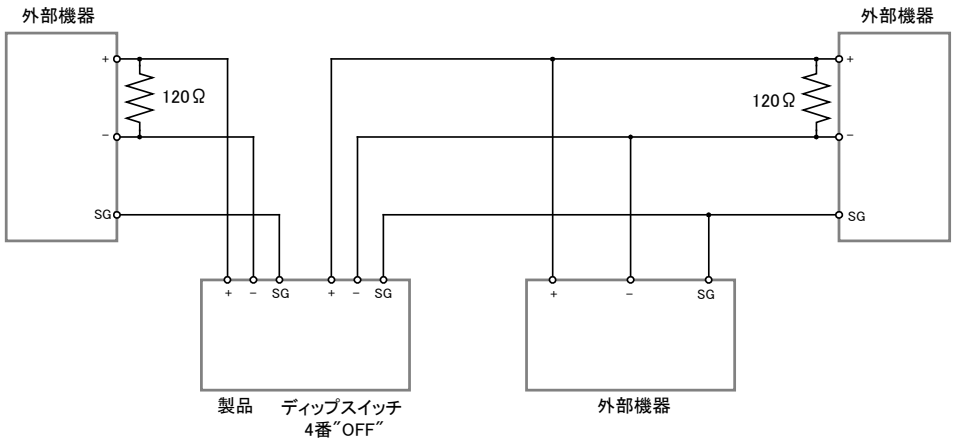


図 38　RS-485 の接続例(製品が配線の中間にある場合)

- 図 37、図 38 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 39 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 39 のように接続します。

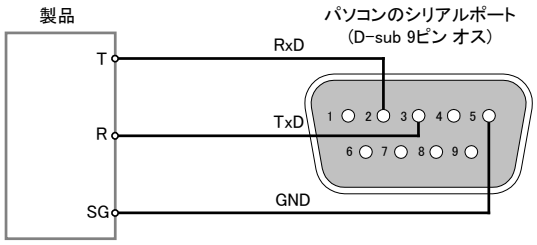


図 39 シリアル 1 の接続例

パソコン以外の機器と接続する場合は、表 17 を参照してください。

表 17　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¥I0x0x_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 18)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 18　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)	I0x0xSamplesMFC.sln
Visual Basic	I0x0xSamplesVB.sln
Visual C#	I0x0xSamplesCS.sln

Visual Basic for Applications 用のサンプルは付属 CD の「¥SAMPLE¥I0x0x_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 19 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-I0x0xTools」)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 19　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 20 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-I0x0xTools」)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 20 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 21 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-I0x0xTools」)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 21 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 40)。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。

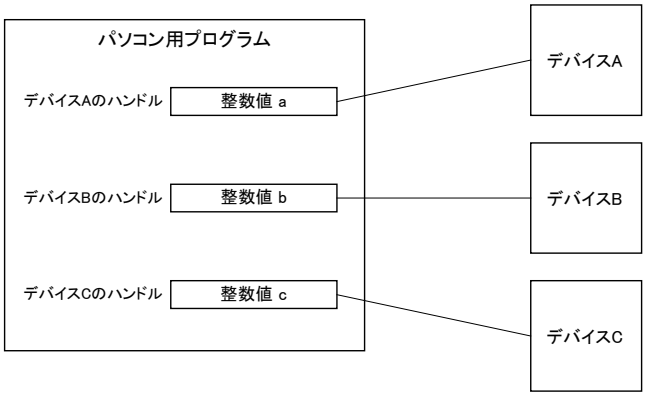


図 40　ハンドル

表 22　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 23 の *TWXA_Open()* 関数を使用します。

表 23　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 30 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;

TWXA_Open(&hDev, 1, TWXA_ANY_DEVICE); //装置番号 1 に接続
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)
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
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。

□ 接点入出力

デバイスが使用できる入出力接点を表 24 に示します。入力接点／出力接点は最大 6 つの接点を 1 つのグループとして、グループ単位で読み出し、書き込みを行います。一部の端子は他の機能と兼用となっています。

尚、本製品では接点入力のことをデジタル入力、接点出力をデジタル出力と表記する場合があります。

表 24　入出力接点

信号名	接点数	方向	ポート名	兼用端子	利用可能な製品
Ia0-Ia3	4	入力	PIa		USBX-I0800
Ic0-Ic3	4	入力	PIc	パルスカウンタと兼用	USBX-I0800/USBX-I0404
Od0-Od1	2	出力	POd		USBX-I0404/USBX-I0008
Od2-Od5	4	出力	POd		USBX-I0008
Of0-Of1	2	出力	POf	パルス出力と兼用	USBX-I0404

入力接点、出力接点は、それぞれ、入力ポート、出力ポートというハードウェアを通じて制御します。入力接点は入力ポートと、出力接点は出力ポートと 1 対 1 に接続されていますので、入力ポートからの読み出しは入力接点の状態の読み取り、出力ポートへの書き込みは出力接点状態の変更と等価です。入出力ポートの制御には、表 25の関数を使用します。また、表 26は絶縁入出力のサンプルとして用意されているプログラムです。

表 25　接点入出力で使用する関数

関数名	説明
TWXA_PortWrite()	出力ポートへ書き込みを行います。
TWXA_PortRead()	入力ポートから読み出しを行います。

表 26　接点入出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)	PortSample	入力接点の状態を表示し、出力接点の状態を操作できます。
Visual Basic	PortSampleVB	
Visual C#	PortSampleCS	
VBA (Excel)	PortSample1.xls	簡易プログラマブルタイマです。テーブルに指定した時刻に出力ポートを操作します。
	PortSample2.xls	簡易データロガーです。入力ポートを監視し、変化があると時刻と状態を記録します。

入力接点の状態を読み取る

TWXA_PortRead() 関数で入力ポートからデータを読み出すことで、入力接点の状態を読むことができます。*Port* 引数で読み出したいポートを指定します。

表 27　TWXA_PortRead() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_PortRead(TW_HANDLE hDev, DWORD Port, BYTE *pData)
VB	Function TWXA_PortRead(ByVal hDev As System.IntPtr, ByVal Port As TWXA_RPORT, ByRef pData As Byte) As Integer
VBA	Function TWXA_PortRead(ByVal hDev As Long, ByVal Port As TWXA_RPORT, ByRef pData As Byte) As Long
C#	STATUS PortRead(System.IntPtr hDev, RPORT Port, out byte pData)

表 28　TWXA_PortRead()の Port 引数に指定する値

言語	値	説明
C/C++	TWXA_PIa	Ia0-Ia3 入力を読み取ります。
C++	TWXA::RPORT::PIa	
VB/VBA	TWXA_RPORT.PIa	
C#	TWXA.RPORT.PIa	
C/C++	TWXA_PIc	Ic0-Ic3 入力を読み取ります。
C++	TWXA::RPORT::PIc	
VB/VBA	TWXA_RPORT.PIc	
C#	TWXA.RPORT.PIc	
C/C++	TWXA_POd	Od0-Od5 の出力値を読み取ります。
C++	TWXA::RPORT::POd	
VB/VBA	TWXA_RPORT.POd	
C#	TWXA.RPORT.POd	
C/C++	TWXA_POf	Of0-Of1 の出力値を読み取ります。
C++	TWXA::RPORT::POf	
VB/VBA	TWXA_RPORT.POf	
C#	TWXA.RPORT.POf	

読み出しは 8 ビット単位で行い、結果は *pData* 引数に格納されます。例えば PIa ポートを読み出した場合、読み取ったデータの各ビットは下の表のように各接点の入力値と対応しています。

表 29　データビットと接点の関係

ビット	7(MSB)	6	5	4	3	2	1	0(LSB)
対応接点	-	-	-	-	Ia3	Ia2	Ia1	Ia0

対応する接点が"OFF"となっているビットは"0"に、"ON"となっているビットは"1"として読み出されます。出力ポートから読み出しを行った場合、現在の出力状態が読み出されます。

出力接点の状態を変更する

TWXA_PortWrite() 関数で出力ポートに書き込みを行うことで、出力接点の状態を変更できます。

表 30　TWXA_PortWrite() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_PortWrite(TW_HANDLE hDev, DWORD Port, BYTE Data, BYTE Mask)
VB	Function TWXA_PortWrite(ByVal hDev As System.IntPtr, ByVal Port As TWXA_WPORT, ByVal Data As Byte, ByVal Mask As Byte) As Integer
VBA	Function TWXA_PortWrite(ByVal hDev As Long, ByVal Port As TWXA_WPORT, ByVal Data As Byte, ByVal Mask As Byte) As Long
C#	STATUS PortWrite(System.IntPtr hDev, WPORT Port, byte Data) STATUS PortWrite(System.IntPtr hDev, WPORT Port, byte Data, byte Mask)

表 31　TWXA_PortWrite()の Port 引数に指定する値

言語	値	説明
C/C++	TWXA_POd	Od0-Od5 の出力値を変更します。
C++	TWXA::WPORT::POd	
VB/VBA	TWXA_WPORT.POd	
C#	TWXA.WPORT.POd	
C/C++	TWXA_POf	Of0-Of1 の出力値を変更します。
C++	TWXA::WPORT::POf	
VB/VBA	TWXA_WPORT.POf	
C#	TWXA.WPORT.POf	

入力と同様に 8 ビット単位でデータを書き込みます。データビットと接点との関係は入力の場合と同様で、"0"を書き込んだビットと対応する接点は"OFF"となり、"1"を書き込んだビットと対応する接点は"ON"になります。

TWXA_PortWrite() 関数の引数 *Mask* に H'FF 以外を指定した場合は、*Mask* バイトのうち"0"となっているビットは影響を受けません。図 41 は H'03 というデータを、*Mask* を H'01 として出力した例です。

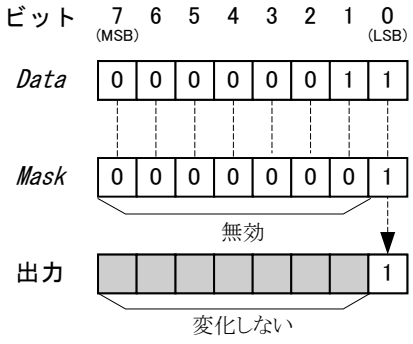


図 41　出力のマスク

リスト 6　デジタル入出力の例(C 言語)

```
BYTE bData;

//Ia0-Ia3 の読出し
TWXA_PortRead(hDev, TWXA_PIa, &bData);

//Od0 だけを"ON"にし、Od1-Od5 は変更しない
TWXA_PortWrite(hDev, TWXA_POd, 0xff, 0x01);
```

リスト 7　デジタル入出力の例(Visual Basic)

```
Dim bData As Byte

'Ia0-Ia3 の読出し
TWXA_PortRead(hDev, TWXA_RPORT.PIa, bData)

'Od0 だけを"ON"にし、Od1-Od5 は変更しない
TWXA_PortWrite(hDev, TWXA_WPORT.POd, &HFF, &H01)
```

リスト 8　デジタル入出力の例(C#)

```
byte bData;

//Ia0-Ia3 の読出し
TWXA.PortRead(hDev, TWXA.RPORT.PIa, out bData);

//Od0 だけを"ON"にし、Od1-Od5 は変更しない
TWXA.PortWrite(hDev, TWXA.WPORT.POd, 0xff, 0x01);
```

- 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 36 ページを、エラー処理については 67 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。

□ パルスをカウントする(USBX-I0800/USBX-I0404 のみ)

パルスカウンタは製品に搭載されるマイコンの外部割り込みを利用したソフトウェアによるカウンタ機能で、割り込み発生回数を 32 ビットのカウンタ変数に記録するものです。入力が"OFF"から"ON"に変化した際にカウントアップする単相カウントのみ可能です。カウンタ入力に使用できる端子は Ic0～Ic3 で、それぞれカウンタチャンネルの 0～3 に対応しています。

表 32　パルスカウント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)	PulseCountSample	パルスカウンタのサンプルです。各カウンタの設定とカウント値の表示を行います。
Visual Basic	PulseCountSampleVB	
Visual C#	PulseCountSampleCS	
VBA (Excel)	PulseCountSample.xls	簡易データロガーです。定期的に各カウンタの値と、前回の値との差分値を記録します。

パルスカウンタ(ソフトウェアカウンタ)の使用方法

ソフトウェアカウンタは *TWXA_PCStart()* 関数でカウントを開始します。カウント値の読み出しには *TWXA_PCReadCnt()* 関数(表 35)を使用します。1 チャンネルずつ読み出すこともできますが、*ChBits* 引数に *TWXA_PC_ALL* などの全てのチャンネルを示す定数を指定すると 0～3 チャンネルまで全てのカウンタ値を読み出すことができます。その場合は、*pCnt* 引数として 4 チャンネル分(16 バイト)の領域を確保するようにしてください。

表 33　ソフトウェアカウンタで使用する関数

関数名	説明
TWXA_PCStart()	カウントを開始します。
TWXA_PCStop()	カウントを停止します。
TWXA_PCReadCnt()	カウンタ値を読み出します。
TWXA_PCSetCnt()	カウンタ値をセットします。主にカウンタクリアに使用します。

表 34　TWXA_PCStart() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_PCStart(TW_HANDLE hDev, long ChBits)
VB	Function TWXA_PCStart(ByVal hDev As System.IntPtr, ByVal ChBits As TWXA_PC_BITS) As Integer
VBA	Function TWXA_PCStart(ByVal hDev As Long, ByVal ChBits As TWXA_PC_BITS) As Long
C#	STATUS PCStart(System.IntPtr hDev, PC_BITS ChBits)

表 35　TWXA_PCReadCnt() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_PCReadCnt(TW_HANDLE hDev, long ChBits, long *pCnt)
VB	Function TWXA_PCReadCnt(ByVal hDev As System.IntPtr, ByVal ChBits As TWXA_PC_BITS, ByRef pCnt As Integer) As Integer Function TWXA_PCReadCnt(ByVal hDev As System.IntPtr, ByVal ChBits As TWXA_PC_BITS, ByVal pCnt() As Integer) As Integer
VBA	Function TWXA_PCReadCnt(ByVal hDev As Long, ByVal ChBits As TWXA_PC_BITS, ByRef pCnt As Long) As Long
C#	STATUS PCReadCnt(System.IntPtr hDev, PC_BITS ChBits, out int pCnt) STATUS PCReadCnt(System.IntPtr hDev, PC_BITS ChBits, out uint pCnt) STATUS PCReadCnt(System.IntPtr hDev, PC_BITS ChBits, int []pCnt) STATUS PCReadCnt(System.IntPtr hDev, PC_BITS ChBits, uint []pCnt)

表 36　ソフトウェアカウンタ操作関数の ChBits 引数に指定する値

言語	値	説明
C/C++	TWXA_PC0	パルスカウンタ 0 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC0	
VB/VBA	TWXA_PC_BITS.PC0	
C#	TWXA.PC_BITS.PC0	
C/C++	TWXA_PC1	パルスカウンタ 1 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC1	
VB/VBA	TWXA_PC_BITS.PC1	
C#	TWXA.PC_BITS.PC1	
C/C++	TWXA_PC2	パルスカウンタ 2 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC2	
VB/VBA	TWXA_PC_BITS.PC2	
C#	TWXA.PC_BITS.PC2	
C/C++	TWXA_PC3	パルスカウンタ 3 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC3	
VB/VBA	TWXA_PC_BITS.PC3	
C#	TWXA.PC_BITS.PC3	
C/C++	TWXA_PC_ALL	全てのチャンネルを同じ動作設定にする場合や、全てのカウンタ値を読み出す場合に指定します。
C++	TWXA::PC_BITS::PC_ALL	
VB/VBA	TWXA_PC_BITS.PC_ALL	
C#	TWXA.PC_BITS.PC_ALL	

リスト 9　パルスカウントの例(C 言語)

```
long LCnt[4];

//全てのチャンネルのカウントを開始
TWXA_PCStart(hDev, TWXA_PC_ALL);

//全てのチャンネルのカウンタ値を読み出し
TWXA_PCReadCnt(hDev, TWXA_PC_ALL, LCnt);
```

リスト 10　パルスカウントの例(Visual Basic)

```
Dim iCnt(3) As Integer

'全てのチャンネルのカウントを開始
TWXA_PCStart(hDev, TWXA_PC_BITS.PC_ALL)

'全てのチャンネルのカウンタ値を読み出し
TWXA_PCReadCnt(hDev, TWXA_PC_BITS.PC_ALL, iCnt)
```

リスト 11　パルスカウントの例(C#)

```
int []iCnt = new int[4];

//全てのチャンネルのカウントを開始
TWXA.PCStart(hDev, TWXA.PC_BITS.PC_ALL);

//全てのチャンネルのカウンタ値を読出し
TWXA.PCReadCnt(hDev, TWXA.PC_BITS.PC_ALL, iCnt);
```

□ パルス出力(USBX-I0404/USBX-I0008 のみ)

製品には内蔵のタイマ機能を使用して、一定周期のパルスを自動的に出力する機能があります。この機能のことを PWM 出力と呼びます。PWM 出力に利用できる端子は Of0、Of1 端子で、それぞれがタイマのチャンネル 0、タイマチャンネル 1 出力に対応しています。

出力のタイミングは 1kHz の内部クロックを 16 ビットの内蔵カウンタで分周することで生成されます。出力できるパルスの周期は約 20msec～65sec までの範囲です。

表 37　PWM 出力で使用する関数

関数名	説明
TWXA_TimerSetMode()	PWM モードの設定/解除を行います。
TWXA_TimerSetPwmExt()	出力パルスの周波数、デューティ、初期位相の設定を行います。
TWXA_TimerStart()	パルス出力動作を開始します。
TWXA_TimerStop()	パルス出力動作を停止します。
TWXA_SetNumOfPulse()	出力パルス数を設定します。
TWXA_ReadNumOfPulse()	残りの出力パルス数を読み出します。
TWXA_TimerReadStatus()	パルス出力中かどうか調べます。
TWXA_TimerSetLevel()	停止中に接点出力の状態を設定します。

表 38　PWM 出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)	PwmSample	各チャンネルの周期、デューティ、初期位相を設定し、指定のパルス数で出力を行います。
Visual Basic	PwmSampleVB	
Visual C#	PwmSampleCS	
VBA (Excel)	PwmSample.xls	予めテーブルに入力した周波数とパルス設定を順次出力します。

パルスの設定方法

パルスの設定には *TWXA_TimerSetPwmExt()* 関数(表 39)を使用します。

表 39　TWXA_TimerSetPwmExt() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_TimerSetPwmExt(TW_HANDLE hDev, long Ch, double dClkFreq, double *pFrequency, double *pDuty, double *pPhase)
VB	Function TWXA_TimerSetPwmExt(ByVal hDev As System.IntPtr, ByVal Ch As Integer, ByVal dClkFreq As Double, ByRef pFrequency As Double, ByRef pDuty As Double, ByRef pPhase As Double) As Integer
VBA	Function TWXA_TimerSetPwmExt(ByVal hDev As Long, ByVal Ch As Long, ByVal dClkFreq As Double, ByRef pFrequency As Double, ByRef pDuty As Double, ByRef pPhase As Double) As Long
C#	STATUS TimerSetPwmExt(System.IntPtr hDev, int Ch, double dClkFreq, ref double pFrequency, ref double pDuty, ref double pPhase)

dClkFreq 引数は必ず 1000 を指定してください。*pFrequency* 引数はパルスの繰り返し周波数を Hz 単位で入力します。*pDuty* 引数は ON デューティを 0～1.0 の範囲で入力します。*pPhase* 引数は出力開始時の位相を 0～1.0 の範囲で入力します。

各引数と出力パルスの関係を図 42 に示します。

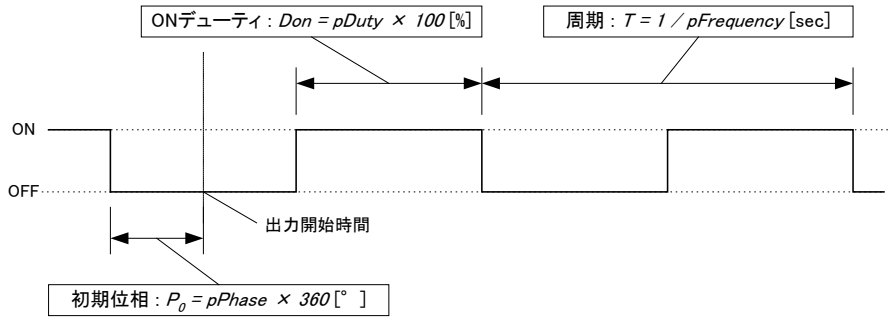


図 42　パラメータと出力パルスの関係

パルスのタイミングは 1kHz の基準クロックを分周して生成されるため、実際に設定できる周波数、デューティ、初期位相の各値は離散的になります。*TWXA_TimerSetPwmExt()* 関数は各パラメータを引数の入力値と近い値に調整し、*pFrequency*、*pDuty*、*pPhase* の各引数に実際に設定できた値を出力して返ります。

パルス出力の手順

1. *TWXA_TimerSetMode()* 関数(表 40)を呼び出し、使用するタイマチャンネルを PWM モードに設定します。*Mode* 引数の値は表 41 を参照してください。

表 40　TWXA_TimerSetMode() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_TimerSetMode(TW_HANDLE hDev, long Ch, long Mode)
VB	Function TWXA_TimerSetMode(ByVal hDev As System.IntPtr, ByVal Ch As Integer, ByVal Mode As TWXA_TIMER_MODE) As Integer
VBA	Function TWXA_TimerSetMode(ByVal hDev As Long, ByVal Ch As Long, ByVal Mode As TWXA_TIMER_MODE) As Long
C#	STATUS TimerSetMode(System.IntPtr hDev, int Ch, TIMER_MODE Mode)

表 41　PWM 出力で Mode 引数に指定する値

言語	値	説明
C/C++	TWXA_TIMER_DISABLE	PWM モードを解除する場合に指定します。対応する端子が接点出力として制御可能になります。
C++	TWXA::TIMER_MODE::DISABLE	
VB/VBA	TWXA_TIMER_MODE.DISABLE	
C#	TWXA.TIMER_MODE.DISABLE	
C/C++	TWXA_TIMER_PWM	指定チャンネルを PWM モードに設定します。チャンネル 0、チャンネル 1 はそれぞれ Of0、Of1 端子と対応し、指定端子はパルス出力専用となります。
C++	TWXA::TIMER_MODE::PWM	
VB/VBA	TWXA_TIMER_MODE.PWM	
C#	TWXA.TIMER_MODE.PWM	

2. *TWXA_TimerSetPwmExt()* 関数を使用し、出力パルスの設定を行います。
3. 必要であれば *TWXA_TimerSetLevel()* 関数で PWM 出力端子の初期状態を変更することが可能です(タイマチャンネルを PWM モードに設定すると、対応する出力端子は *TWXA_PortWrite()* 関数で操作できなくなります)。
4. 必要であれば *TWXA_TimerSetNumOfPulse()* 関数で出力パルス数を設定します。
5. *TWXA_TimerStart()* 関数でパルス出力を開始します。

6. *TWXA_TimerSetNumOfPulse()* 関数で出力パルス数を設定した場合は、指定のパルス数を出力するとタイマが自動的に停止します。残りの出力パルス数を調べたい場合には、*TWXA_TimerReadNumOfPulse()* 関数を使用します。タイマが動作中か停止中かを調べるには *TWXA_TimerReadStatus()* 関数を使用します。
7. パルス出力を停止する場合は *TWXA_TimerStop()* 関数を使用します。

TWXA_TimerStop() 関数でタイマの動作と非同期に停止を行うと、パルス出力が"ON"状態で停止する場合があります。これを避けたい場合には以下の手順で停止を行ってください。

1. *TWXA_PortWrite()* 関数で POf に 0 を書き込みます(41 ページ参照)。これにより PWM 端子の機能をデジタル出力に戻したときに出力値が自動的に"OFF"になります。
2. *TWXA_TimerSetMode()* 関数で PWM モードを解除します。この時点で端子の機能が PWM からデジタル出力に切り替わり、出力が"OFF"になります。また、タイマの動作も停止します。
3. *TWXA_TimerSetLevel()* 関数で停止したタイマ出力を"OFF"にします。これを行わないと次回の PWM 出力時に意図しないパルスが出力される場合があります。

- パルス出力を開始した後に *TWXA_TimerSetPwmExt()* 関数で周期やデューティを変更することができますが、変更は出力中のパルスと非同期に実行されるため、最初に設定した位相とズレを生じたり、1 周期以上"ON"(または"OFF")出力が続いてしまう場合があります。

リスト 12　PWM 出力の例(C 言語)

```
double dFreq;
double dDuty;
TCHAR c[256];

dFreq = 10;   //周波数 = 10Hz
dDuty = 0.6;  //デューティ = 60%

//タイマ 0 を PWM に設定
TWXA_TimerSetMode(hDev, 0, TWXA_TIMER_PWM);

//パルス設定
TWXA_TimerSetPwmExt(hDev, 0, 1000, &dFreq, &dDuty);

//実際の設定値を表示
_stprintf_s(c, 256, _T("周波数 : %.2f Hz  "), dFreq);
OutputDebugString(c);

_stprintf_s(c, 256, _T("デューティ : %.2f %% に設定しました。¥n"), dDuty * 100);
OutputDebugString(c);

//出力パルス数を 100 に設定
TWXA_TimerSetNumOfPulse(hDev, 0, 100);

//出力開始
TWXA_TimerStart(hDev, TWXA_TIMER_BIT0);
```

リスト 13　PWM 出力の例(Visual Basic)

```
Dim dFreq As Double
Dim dDuty As Double

dFreq = 10 '周波数 = 10Hz
dDuty = 0.6  'デューティ = 60%

'タイマ 0 を PWM に設定
TWXA_TimerSetMode(hDev, 0, TWXA_TIMER_MODE.PWM)

'パルス設定
TWXA_TimerSetPwmExt(hDev, 0, 1000, dFreq, dDuty)

'実際の設定値を表示
Debug.WriteLine(String.Format("周波数 : {0:#.#0} Hz", dFreq))
Debug.WriteLine(String.Format("デューティ : {0:##0.#0} % に設定しました。", dDuty * 100))

'出力パルス数を 100 に設定
TWXA_TimerSetNumOfPulse(hDev, 0, 100)

'出力開始
TWXA_TimerStart(hDev, TWXA_TIMER_BITS.TIMER0)
```

リスト 14　PWM 出力の例(C#)

```
double dFreq;
double dDuty;
double dPhase;

dFreq = 10;   //周波数 = 10Hz
dDuty = 0.6;  //デューティ = 60%
dPhase = 0;

//タイマ 0 を PWM に設定
TWXA.TimerSetMode(hDev, 0, TWXA.TIMER_MODE.PWM);

//パルス設定
TWXA.TimerSetPwmExt(hDev, 0, 1000, ref dFreq, ref dDuty, ref dPhase);

//実際の設定値を表示
Debug.WriteLine(string.Format("周波数 : {0:#.#0} Hz", dFreq));
Debug.WriteLine(string.Format("デューティ : {0:##0.#0} % に設定しました。", dDuty * 100));

//出力パルス数を 100 に設定
TWXA.TimerSetNumOfPulse(hDev, 0, 100);

//出力開始
TWXA.TimerStart(hDev, TWXA.TIMER_BITS.TIMER0);
```

□ 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み取りを一旦中止し、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 42 にシリアルポート制御で使用する関数をあげます。

表 42　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 43　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 44 は *TWXA_SCISetMode()* 関数の宣言です。*Mode* 引数には表 45 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 46 のボーレートを入力します。

表 44 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 45 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 46　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);
}
```

□ ハードウェアイベントの監視

『USBX-I0800』/『USBX-I0404』ではパルスカウンタ(ソフトウェアカウンタ)のカウント値を閾値と比較し、指定の値になったときに、アプリケーションに通知することができます。ユーザーファームを作成した場合は、どの製品でも独自のイベントをアプリケーションに通知することができます。この通知機能をハードウェアイベントと呼びます。

ハードウェアイベントは通常のアプリケーションプログラムには Windows のメッセージとして、LabVIEW を用いたプログラムにはユーザーイベントとして通知されます。

表 47 はハードウェアイベントのプログラムです。

表 47 ハードウェアイベント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)	EventSample	パルスカウンタの値が変化した場合に画面に表示します。
Visual Basic	EventSampleVB	
Visual C#	EventSampleCS	

- Visual Basic for Applications ではサポートされません。
- Windows のメッセージによる通知は、割り込みのように瞬時に行われるものではありませんので、リアルタイム制御には利用できません。

ハードウェアイベントの監視を開始するには、表 48 の *TWXA_SetHwEvent()* 関数を使用します。この関数には引数として *TWXA_HW_EVENT* 構造体(表 49)を渡します。

表 48 TWXA_SetHwEvent() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_SetHwEvent(TW_HANDLE hDev, TWXA_HW_EVENT *pHwEvent)
VB	Function TWXA_SetHwEvent(ByVal hDev As System.IntPtr, ByRef pHwEvent As TWXA_HW_EVENT) As Integer
VBA	Function TWXA_SetHwEvent(ByVal hDev As Long, ByRef pHwEvent As TWXA_HW_EVENT) As Long
C#	STATUS SetHwEvent(System.IntPtr hDev, ref HW_EVENT pHwEvent)

表 49　TWXA_HW_EVENT 構造体の宣言

言語	宣言

C/C++	typedef struct tagHwEvent{
	  HWND hRecvWindow;
	  DWORD idRecvThread;
	  LPVOID lpRsv;
	  UINT Message;
	  DWORD EventBits;
	  long PCCnt[4];
	  long PCCmp[4];
	  long ADVal[4];
	  short ADCmp[4];
	} TWXA_HW_EVENT;

VB	Public Structure TWXA_HW_EVENT
	  Public hRecvWindow As System.IntPtr
	  Public idRecvThread As Integer
	  Public lpRsv As System.IntPtr
	  Public Message As Integer
	  Public EventBits As Integer
	  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _
	  Public PCCnt() As Integer
	  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _
	  Public PCCmp() As Integer
	  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _
	  Public ADVal() As Integer
	  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _
	  Public ADCmp() As Short
	
	  Public Sub Initialize()
	  ReDim PCCnt(3)
	  ReDim PCCmp(3)
	  ReDim ADVal(3)
	  ReDim ADCmp(3)
	  End Sub
	End Structure

VBA	Public Type TWXA_HW_EVENT hRecvWindow As Long idRecvThread As Long lpRsv As Long Message As Long EventBits As Long PCCnt(3) As Long PCCmp(3) As Long ADVal(3) As Long ADCmp(3) As Integer End Type

言語	宣言

C#	public struct HW_EVENT
	{
	  public System.IntPtr hRecvWindow;
	  public uint idRecvThread;
	  public System.IntPtr lpRsv;
	  public int Message;
	  public uint EventBits;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
	  public int[] PCCnt;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
	  public int[] PCCmp;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
	  public int[] ADVal;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
	  public short[] ADCmp;
	
	  public void Initialize()
	  {
	  PCCnt = new int[4];
	  PCCmp = new int[4];
	  ADVal = new int[4];
	  ADCmp = new short[4];
	  }
	}

hRecvWindow
ウィンドウでメッセージを受け取る場合は、ウィンドウハンドルを入力します。必要が無い場合は0にしてください。

idRecvThread
スレッドでメッセージを受け取る場合は、スレッド ID を指定します。必要が無い場合は 0 にしてください。

lpRsv
予約領域です。0 にしてください。

Message
指定のイベントが発生したときに通知されるメッセージ番号です。通常は H'8000～H'BFFF の範囲の任意の値を指定します。パルスカウンタの通知条件が成立すると、ここで設定した番号のメッセージが、ウィンドウ、または、スレッドにポストされます。

EventBits
通知を受けたいイベントをビットで指定します(表 50)。複数のビットを指定することもできます。

PCCnt
パルスカウンタの値と比較する閾値を指定します。配列のインデックスがパルスカウンタのチャンネルと対応しています。

PCCmp
パルスカウンタの値と閾値の比較方法を指定します(表 51)。配列のインデックスがパルスカウンタのチャンネルと対応しています。

ADVal
本製品では使用しません。0 にしてください。

ADCmp
本製品では使用しません。0 にしてください。

Initialize()
Visual Basic と C#では配列の領域確保用に最初に呼び出します。

表 50　EventBits の指定

ビット	監視するイベント
TWXA_EVENT_PC0(H'00000001)	パルスカウンタ 0 を監視
TWXA_EVENT_PC1(H'00000002)	パルスカウンタ 1 を監視
TWXA_EVENT_PC2(H'00000004)	パルスカウンタ 2 を監視
TWXA_EVENT_PC3(H'00000008)	パルスカウンタ 3 を監視
TWXA_EVENT_USER(H'80000000)	ユーザー定義(ユーザーファームから通信を行う場合)
TWXA_EVENT_PC_ALL(H'0000000F)	パルスカウンタ全チャンネルを監視

パルスカウンタ入力を監視する

パルスカウンタによるイベントはカウンタ値が、予め設定した閾値以上となった場合、または、カウンタ値が変化した場合に発生させることができます。

1. Visual Basic または C# を利用する場合、*TWXA_HW_EVENT* 構造体の *Initialize()* メソッドを呼びます。
2. *TWXA_HW_EVENT* 構造体の *hRecvWindow* にウィンドウのハンドルを指定します。ウィンドウを持たないアプリケーションの場合、*idRecvThread* にスレッド ID を指定します。また、イベント発生時に受け取るメッセージ番号を *Message* に指定します。
3. *TWXA_HW_EVENT* 構造体の *EventBits* に監視するパルスカウンタチャンネルを指定します(表 50 参照)。
4. *TWXA_HW_EVENT* 構造体の *PCCnt* にカウンタ値と比較する閾値を指定します。配列のインデックスはチャンネルを示します。例えばパルスカウンタ 2 を監視する場合は、*PCCmp[2]* に閾値を設定します。
5. *TWXA_HW_EVENT* 構造体の *PCCmp* に比較方法を指定します。*PCCmp* に指定する値と、イベント発生条件を 表 51 に示します。

表 51　PCCmp の設定値とハードウェアイベントの発生条件

PCCmp[x]の設定	ハードウェアイベントの発生条件
TWXA_CMP_GE(H'7FFFFFFF)	指定チャンネル(x)のカウンタ値が PCCnt[x]以上になった場合
0	指定チャンネル(x)のカウンタ値が変化した場合

6. パラメータを設定した構造体を引数として *TWXA_SetHwEvent()* 関数を呼び出します。
7. 使用するパルスカウンタの設定を行い、カウント動作を開始します(43 ページ参照)。
8. 設定した条件が成立すると、指定したウィンドウ(または、スレッド)にメッセージがポストされます。メッセージの各パラメータは以下の値となります。

表 52　パルスカウンタイベントによるメッセージのパラメータ

項目	ハードウェアイベントの発生条件
Msg	TWXA_HW_EVENT 構造体の Message に指定した値
wParam(WParam)	イベントを発生したカウンタチャンネルを示すビット(表 50)
lParam(LParam)	イベント発生時のカウンタの値

9. *TWXA_HW_EVENT* 構造体への参照を Null 値とするか、*EventBits* を 0 として *TWXA_SetHwEvent()* 関数を呼び出すとイベントの監視を終了します。

□ 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 53　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、書き込み、読み出しが行えます。*Port* 引数には表 54 の値を指定してください。

表 54　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 55　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 56　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 43 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。

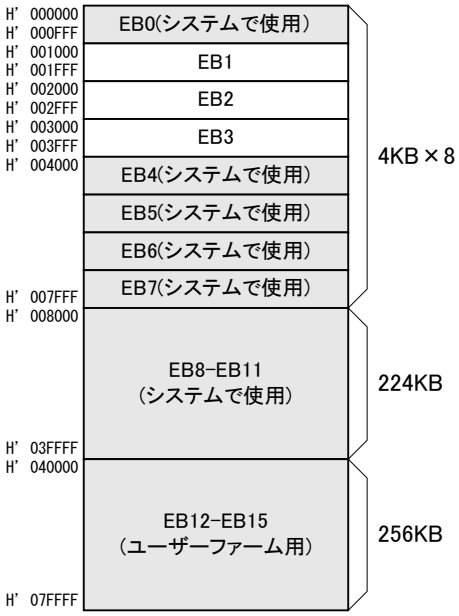


図 43　フラッシュメモリマップ

表 57　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／書き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き込みを行います。
TWXA_PortBRead()	フラッシュメモリからデータを読み出します。

表 58　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き込みを行います。
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き込み操作は特殊で、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。書き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 43 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き込みを行います。書き込みは 128 バイト毎のブロック単位で行います。そのため、書き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／書き込みを行う際には、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(60 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み出しは、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。
2. *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 59　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き込み方法

1. *TWXA_FlashAttachWriter()* 関数を呼びます。
2. *TWXA_FlashWrite()* 関数(表 60)を呼び出します。*Address* 引数には書き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* 引数に指定する書き込みバイト数も 128 の倍数としてください。

表 60　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);

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

//読み出し
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)

'書き込み
TWXA_FlashWrite(hDev, &H1000, strWrite, 128)

'読み出し
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

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

'読み出し
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));
```

□ EEPROM の利用

製品には 8K バイトの EEPROM が内蔵されています。EEPROM はフラッシュメモリと同様に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です。1 バイト単位での書込みも可能で、100 万回の書込み耐久性能がありますので、フラッシュメモリよりも手軽に利用することができます。

EEPROM はフラッシュメモリやユーザーメモリとは別のアドレス空間に配置され、0～8191 のアドレスを指定してアクセスします。EEPROM への書込みには *TWXA_EEWrite()* 関数(表 63)、EEPROM からのデータの読出しには *TWXA_EERead()* 関数(表 64)を使用します。

表 61　EEPROM の操作に使用する関数

関数名	説明
TWXA_EEWrite()	EEPROM に書き込みを行います。
TWXA_EERead()	EEPROM からデータを読み出します。

表 62　EEPROM 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)	EEPROMSample	EEPROM の状態表示、ファイルデータの EEPROM への書き込みを行います。
Visual Basic	EEPROMSampleVB	
Visual C#	EEPROMSampleCS	
VBA (Excel)	EEPROMSample.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 EEPROM に書き込むことができます。

表 63　TWXA_EEWrite() の関数宣言

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

表 64　TWXA_EERead() の関数宣言

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

□ 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_PIc, 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::POd, 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.POd, &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.POd, &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.POd, 0);

if (ret != 0)
{
  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 44 は参考として各入力ポートに対する *TWXA_PortRead()* 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。

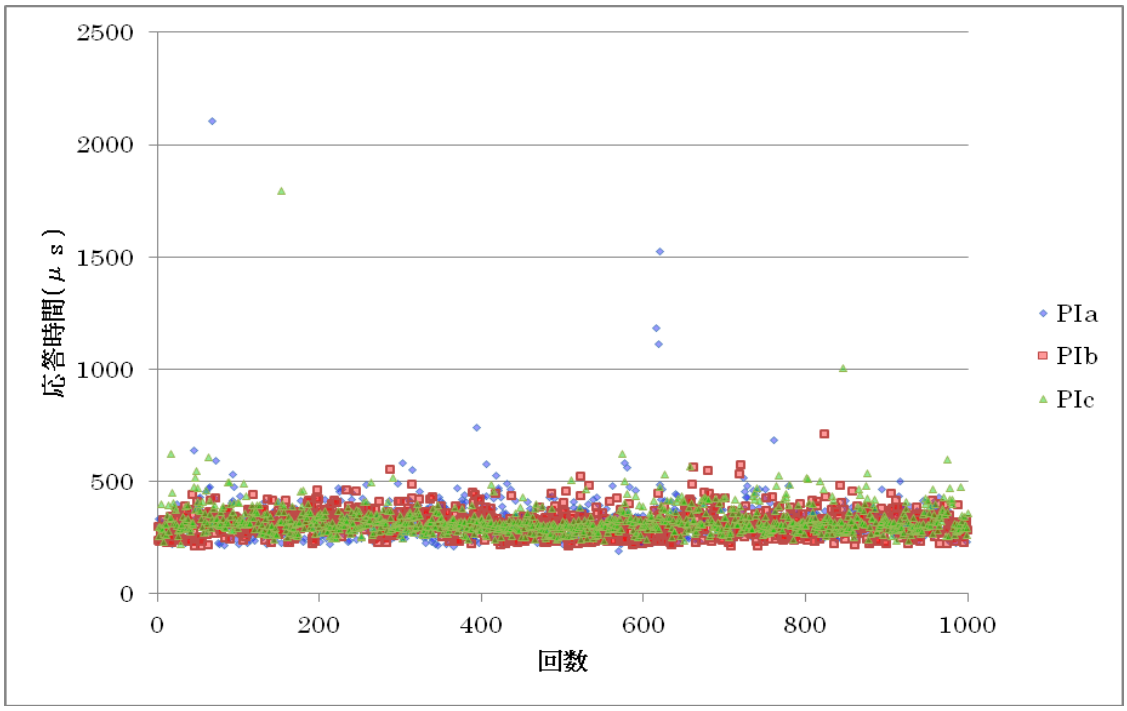


図 44　TWXA_PortRead() 関数の応答時間

リアルタイム処理

パソコン上のプログラムではアプリケーションが必要とするリアルタイム性能を満足できないことも考えられます。例えば、10msec 幅のパルスを検出するために、繰り返し *TWXA_PortRead()* 関数の呼び出しを行うプログラムを作成したとします。このプログラムは、同時に動作している他のプロセス動作のために *TWXA_PortRead()* 関数の呼び出し間隔が 10msec 以上となってしまうと、必要なパルスを検出できない場合が発生します。

パソコンからの命令だけでは十分な性能が得られない場合には、ユーザーファームの導入を検討してください。ユーザーファームでは割り込みを利用するなどして、よりリアルタイム性の高いプログラムを記述することができます。

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
2013 年 6 月	初	
2018 年 4 月	2	・対応 OS に Windows 10 を追加 ・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
2020 年 12 月	3	・仕様に関する、誤解を招く恐れのある表現を修正

© テクノウェーブ株式会社 2013-2020