USBX-I2424P
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。
- 本製品のシリアルコネクタに、屋外や雷の影響を受けやすい場所に設置されたケーブルを直接接続しないでください。感電や故障の原因となります。
- 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。

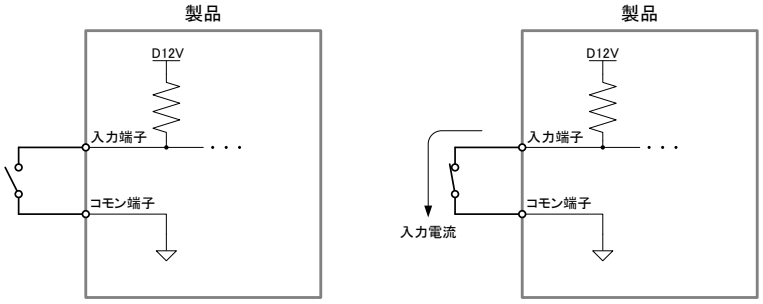
表 1　電気的状態の表記方法

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。

デジタル入力端子の状態

デジタル入力端子は十分な入力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。



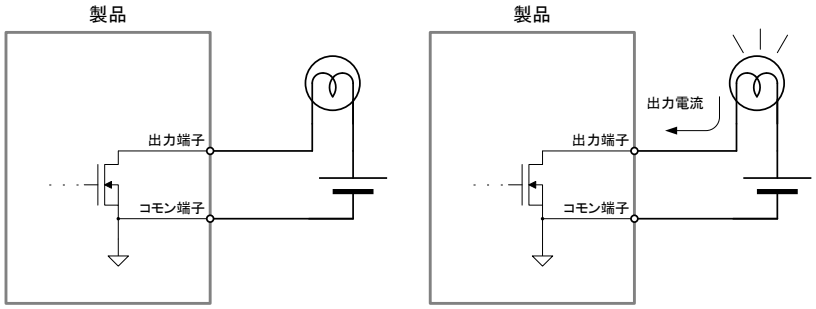
入力端子の″OFF″状態
(ロジックレベルは″Hi″)

入力端子の″ON″状態
(ロジックレベルは″Lo″)

図 1　デジタル入力端子の"OFF"状態と"ON"状態

デジタル出力端子の状態

デジタル出力端子は出力電流が流れている状態を″ON″、流れていない状態を″OFF″と定義します。



出力端子の″OFF″状態

出力端子の″ON″状態

図 2　デジタル出力端子の"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-I2424P』(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、AD コンバータ、DA コンバータ、パルスカウンタ、PWM 出力、シリアル通信などの機能を制御できます。

- **デジタル I/O**[1] – デジタル I/O として絶縁入力最大 24 点、絶縁出力最大 24 点を備えています。絶縁入力端子は無電圧の接点入力が可能で、メカニカルスイッチ、リレー接点、オープンコレクタ信号などを直接接続できます[2]。絶縁出力端子はオープンドレイン出力で、1 点あたり 150mA までの電流を駆動できます。
- **AD コンバータ** – 非絶縁入力の 16 ビット AD コンバータを 4 チャンネル搭載しています。AD コンバータは完全な 4 チャンネル同時サンプリングを、最大 50KS/sec[3]で行うことが可能です。また、オーバーサンプリングレートの設定や、入力レンジとして-5～+5V、-10～+10V を選択できます。
- **DA コンバータ** – 0～5V 出力の非絶縁 10 ビット DA コンバータを 2 チャンネル搭載しています。
- **16 ビットハードウェアカウンタ**[1] – 5MHz まで入力可能な 16 ビットハードウェアカウンタを最大 8 チャンネル使用可能です。ハードウェアカウンタは最大 2 チャンネルの 2.5MHz まで入力可能な 2 相パルスカウンタとしても使用可能でインクリメンタル式ロータリーエンコーダを接続することができます。通常の単相カウンタと 2 相カウンタを組み合わせて同時に使用することもできます。
- **32 ビットソフトウェアカウンタ**[1] – 最大 8 チャンネルの 32 ビットソフトウェアカウンタを使用可能です。ソフトウェアカウンタは最大 4 チャンネルの 2 相パルスカウンタ、最大 2 チャンネルの 3 相パルスカウンタとしても使用可能です。通常の単相カウンタと 2 相カウンタなどを組み合わせて同時に使用することもできます。
- **PWM 出力**[1] – 最大 5 チャンネルの PWM 信号を出力することができます。出力パルス数を指定することができます。
- **シリアル通信**[4] – RS-232C の信号レベルで通信できるシリアルポートを 2 チャンネル備えています。
- **ハードウェアイベントの監視** – ソフトウェアカウンタ、AD コンバータへの入力を監視し、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。
- 制御用 API は DLL モジュールで提供され、Visual C++® や Visual Basic、Visual C#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。
- 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] デジタル I/O、PWM 出力、各カウンタは一部の端子、および、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] ソース出力機器は接続できません。

[3] 使用 API により変換速度は変化します。

[4] 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。32bit 版 LabVIEW のみ対応しています。

□ 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。

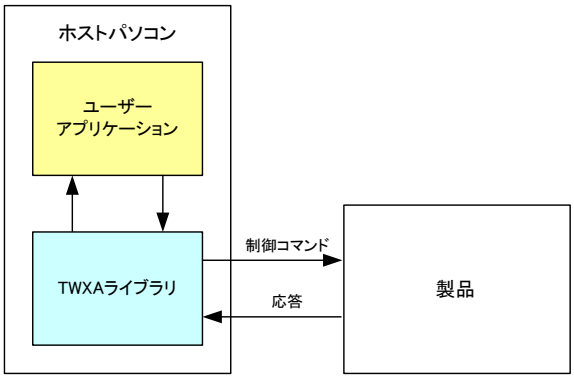


図 3　ホストパソコンからの制御

表 5 のプログラミング言語に対しては、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を提供しています。これらを使用してプログラムから TWXA ライブラリの各関数を呼び出すことで、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

LabVIEW についても TWXA ライブラリの各関数と対応した VI ライブラリを用意していますので、これを使用することで LabVIEW から製品を制御することができます。

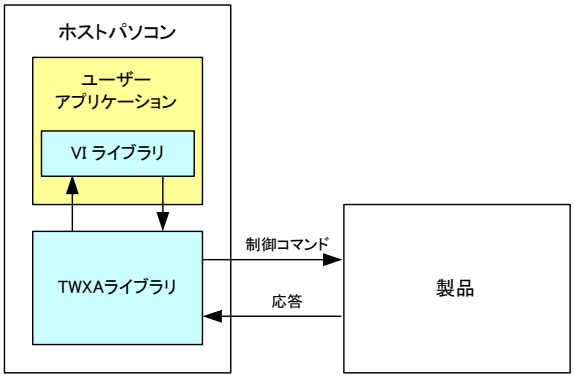


図 4　LabVIEW での利用

□ 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-I2424P 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-I2424.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
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	130(W)×125(D)×30.6(H)[mm]	DIN レール取付具,ゴム足含まず
重量	450[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	30[年]	
絶縁抵抗(GND-COM 間)	10[GΩ]以上	測定条件:500VDC
インタフェース	USB2.0	Hi-Speed 対応
対応 OS	Windows 7, 8, 8.1, 10	32 ビット/64 ビット

表 8　絶縁入力仕様

項目		仕様	備考
入力点数		最大 24 点	
入力方式		無電圧接点入力	
入力電圧		0～25.2[V]	D12V,COM 端子間電圧
入力抵抗	IN00～IN07 IN10～IN17	6.8[kΩ]	入力回路図参照
	IN20～IN27	2.7[kΩ]	入力回路図参照
オン電圧	IN00～IN07 IN10～IN17	DC7.8[V]以上	D12V,入力端子間電圧
	IN20～IN27	DC10.7[V]以上	D12V,入力端子間電圧
フォトカプラ 応答速度	IN00～IN07 IN10～IN17	最大 100[μsec]	
	IN20～IN27	最大 100[nsec]	

表 9　絶縁出力仕様

項目		仕様	備考
出力点数		最大 24 点	
出力方式		オープンドレイン	
出力電圧		最大 25.2[V]	COM,出力端子間電圧
出力電流(シンク電流)		最大 150[mA]	条件：COM,出力端子間電圧 0.4V
フォトカプラ 応答速度	OUT00～OUT07 OUT10～OUT17	最大 200[nsec]	
	OUT20～OUT27	最大 100[μsec]	

表 10　PWM 出力仕様

項目	仕様	備考
出力チャンネル数	最大 5 チャンネル	PWM2,PWM3,PWM4,PWM6,PWM7
出力方式	絶縁出力仕様参照	
出力電圧	絶縁出力仕様参照	
出力電流(シンク電流)	絶縁出力仕様参照	
フォトカプラ応答速度	絶縁出力仕様参照	
出力周波数	最大 1[MHz]	
デューティ分解能	出力周波数に依存	
パルス幅ひずみ	最大 60[nsec]	条件：12V 出力,82Ω負荷

表 11　ハードウェアカウンタ仕様

項目		仕様	備考
入力チャンネル数		最大 8 チャンネル	HC0～HC7
入力方式		絶縁入力仕様参照	
入力電圧		絶縁入力仕様参照	
入力抵抗		絶縁入力仕様参照	
フォトカプラ応答速度		絶縁入力仕様参照	
カウンタビット数		16[bit]	
カウント設定		ON→OFF、OFF→ON、 両エッジ、2 相	
周波数	単相	最大 5[MHz]	
	2 相	最大 2.5[MHz]	

表 12　ソフトウェアカウンタ仕様

項目	仕様	備考
入力チャンネル数	最大 8 チャンネル	SC0～SC7
入力方式	絶縁入力仕様参照	
入力電圧	絶縁入力仕様参照	
入力抵抗	絶縁入力仕様参照	
フォトカプラ応答速度	絶縁入力仕様参照	
カウンタビット数	32[bit]	
カウント設定	ON→OFF、OFF→ON、 両エッジ、2 相、3 相	
周波数	最大 10[kHz]	

表 13　AD コンバータ仕様

項目		仕様	備考
入力チャンネル数		4 チャンネル	AD0～AD3
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
リファレンス温度偏差		標準±10[ppm/℃]	
変換時間		最大 4.2[μsec]	オーバーサンプリング無し
		最大 9.1[μsec]	オーバーサンプリングレート 2
		最大 18.8[μsec]	オーバーサンプリングレート 4
		最大 39[μsec]	オーバーサンプリングレート 8
		最大 78[μsec]	オーバーサンプリングレート 16
		最大 158[μsec]	オーバーサンプリングレート 32
		最大 315[μsec]	オーバーサンプリングレート 64
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 14　DA コンバータ仕様

項目		仕様	備考
出力チャンネル数		2 チャンネル	DA0, DA1
出力電圧		0～5.0[V]	
出力電流		最大 2.0[mA]	チャンネルあたり
変換部	分解能	10[bit]	
	リファレンス精度	最大 0.4[%]	条件：全温度範囲
	リファレンス温度偏差	最大 50[ppm]	
	絶対精度	最大±2[LSB]	条件：全温度範囲
アンプ部	オフセット電圧	最大±7[mV]	条件：全温度範囲

表 15　シリアルポート仕様

項目	仕様	備考
チャンネル数	2	SER0, SER1
方式	調歩同期式(フロー制御なし [5])	
ビットレート	300～256000[bps]	
信号レベル	RS-232C 準拠	

- デジタル出力端子の OUT10、OUT14、OUT06、OUT00、OUT04 と PWM 出力端子 PWM2、PWM3、PWM4、PWM6、PWM7 はそれぞれ同じ端子を使用します。どちらか片方の機能しか利用できません。
- デジタル入力端子 IN00～IN06 とソフトウェアカウンタ入力 SC0～SC6 はそれぞれ同じ端子を使用します。どちらか片方の機能しか利用できません。
- デジタル入力端子 IN07、ソフトウェアカウンタ入力 SC7、AD トリガ入力 ADTRG は同じ端子を使用します。いずれか 1 つの機能しか利用できません。
- デジタル入力端子 IN20～IN26 とハードウェアカウンタ入力 HC0～HC6 はそれぞれ同じ端子を使用します。どちらか片方の機能しか利用できません。
- デジタル入力端子 IN27、ハードウェアカウンタ入力 HC7、AD クロック入力 ADCLK は同じ端子を使用します。いずれか 1 つの機能しか利用できません。
- PWM 出力とハードウェアカウンタは同じハードウェア機能を利用しているため、両機能で合わせて 8 チャンネルまでしか利用できません。

[5] 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

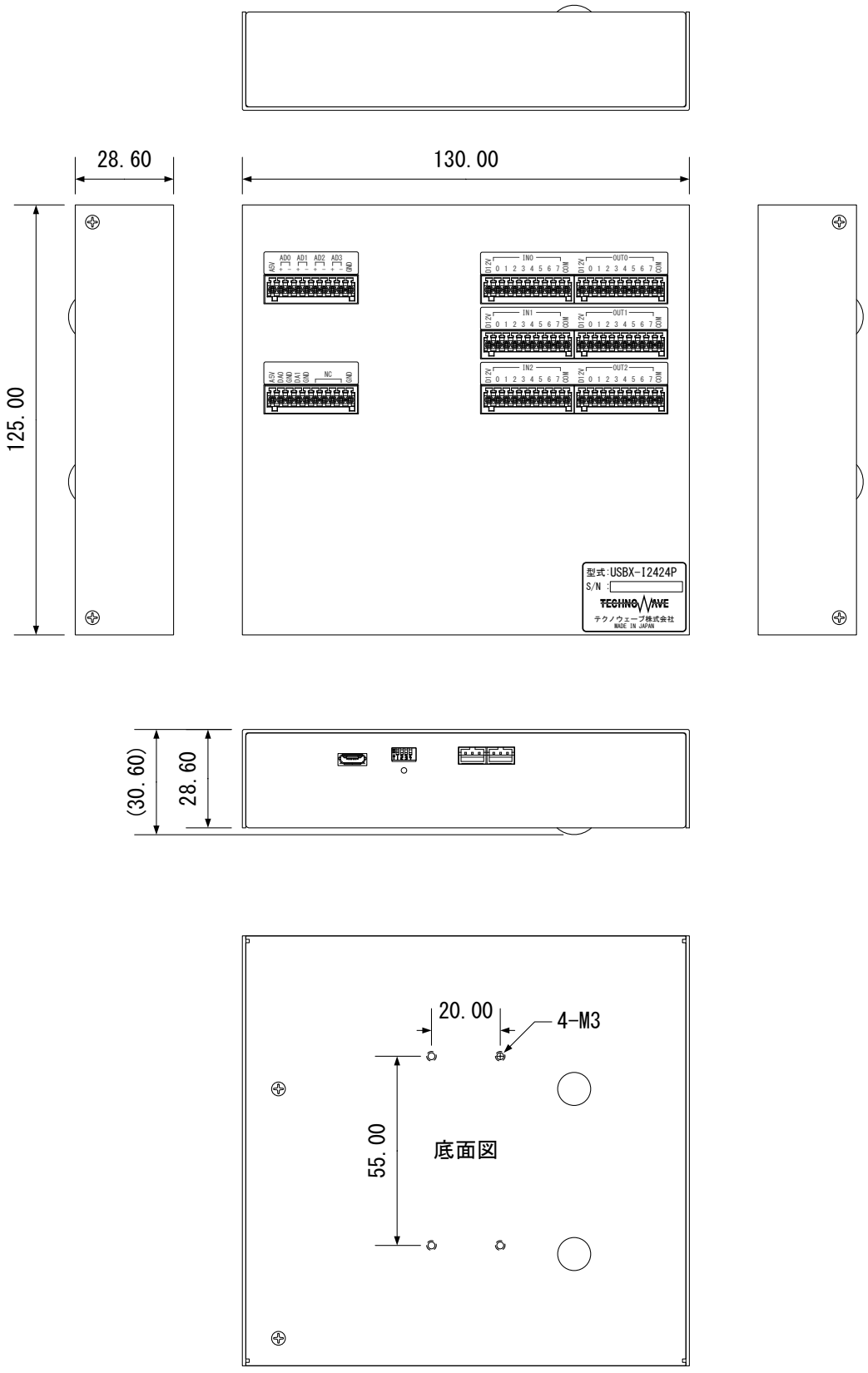
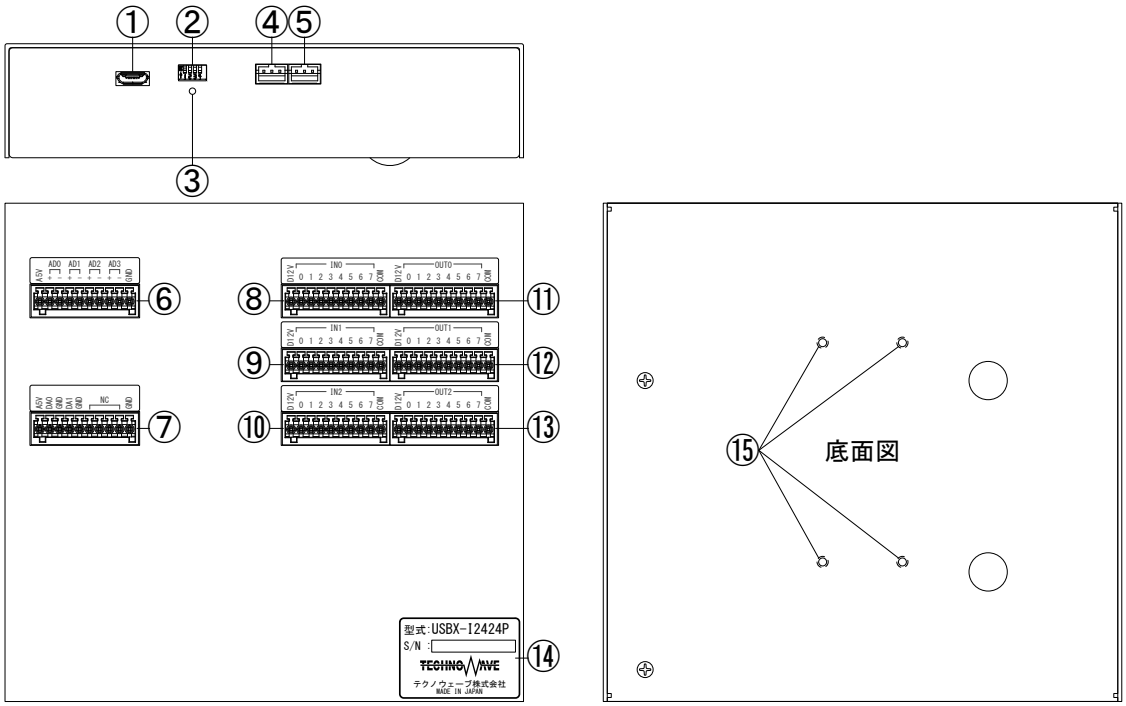


図 5　外形寸法図

□ 各部の名称



No.	名称	説明
①	USB コネクタ (micro USB Type-B)	パソコンの USB ポートに接続します。
②	ディップスイッチ	製品の動作設定を行います。
③	電源表示 LED	電源がオンになると LED が点灯します。
④	SER0 端子	RS-232C による通信に使用します。
⑤	SER1 端子	
⑥	CN1 端子	アナログ信号の入力端子です。
⑦	CN2 端子	アナログ信号の出力端子です。
⑧	CN3 端子	デジタル信号の入力端子です。
⑨	CN4 端子	
⑩	CN5 端子	
⑪	CN6 端子	デジタル信号の出力端子です。
⑫	CN7 端子	
⑬	CN8 端子	
⑭	銘板	製品の型式とシリアル番号が記載されています。
⑮	M3 ネジ穴	DIN レール取付具の固定に使用します。

□ 端子説明

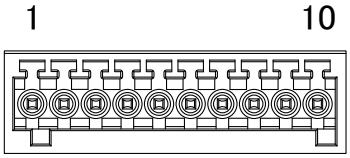


図 6　CN1～CN8 端子配列 [6]

表 16　CN1 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN1-1	A5V	アナログ用 5V 出力	O	非絶縁
CN1-2	AD0+	AD コンバータ 0+	I	非絶縁
CN1-3	AD0-	AD コンバータ 0-	I	非絶縁
CN1-4	AD1+	AD コンバータ 1+	I	非絶縁
CN1-5	AD1-	AD コンバータ 1-	I	非絶縁
CN1-6	AD2+	AD コンバータ 2+	I	非絶縁
CN1-7	AD2-	AD コンバータ 2-	I	非絶縁
CN1-8	AD3+	AD コンバータ 3+	I	非絶縁
CN1-9	AD3-	AD コンバータ 3-	I	非絶縁
CN1-10	GND	シグナルグランド	-	非絶縁

表 17　CN2 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN2-1	A5V	アナログ用 5V 出力	O	非絶縁
CN2-2	DA0	DA コンバータ 0	O	非絶縁
CN2-3	GND	シグナルグランド	-	非絶縁
CN2-4	DA1	DA コンバータ 1	O	非絶縁
CN2-5	GND	シグナルグランド	-	非絶縁
CN2-6	NC	未接続	-	
CN2-7	NC	未接続	-	
CN2-8	NC	未接続	-	
CN2-9	NC	未接続	-	
CN2-10	GND	シグナルグランド	-	非絶縁

表 18　CN3 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN3-1	D12V	デジタル用 12V 出力	O	絶縁
CN3-2	IN00/SC0	デジタル入力/ソフトウェアカウンタ	I	絶縁
CN3-3	IN01/SC1	デジタル入力/ソフトウェアカウンタ	I	絶縁
CN3-4	IN02/SC2	デジタル入力/ソフトウェアカウンタ	I	絶縁
CN3-5	IN03/SC3	デジタル入力/ソフトウェアカウンタ	I	絶縁
CN3-6	IN04/SC4	デジタル入力/ソフトウェアカウンタ	I	絶縁
CN3-7	IN05/SC5	デジタル入力/ソフトウェアカウンタ	I	絶縁
CN3-8	IN06/SC6	デジタル入力/ソフトウェアカウンタ	I	絶縁
CN3-9	IN07/SC7/ ADTRG	デジタル入力/ソフトウェアカウンタ/ AD トリガ入力	I	絶縁
CN3-10	COM	コモン	-	絶縁

[6] 適合コネクタは「FK-MC 0,5/10-ST-2,5(1881406)」(フエニックス・コンタクト)。

表 19　CN4 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN4-1	D12V	デジタル用 12V 出力	O	絶縁
CN4-2	IN10	デジタル入力	I	絶縁
CN4-3	IN11	デジタル入力	I	絶縁
CN4-4	IN12	デジタル入力	I	絶縁
CN4-5	IN13	デジタル入力	I	絶縁
CN4-6	IN14	デジタル入力	I	絶縁
CN4-7	IN15	デジタル入力	I	絶縁
CN4-8	IN16	デジタル入力	I	絶縁
CN4-9	IN17	デジタル入力	I	絶縁
CN4-10	COM	コモン	-	絶縁

表 20　CN5 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN5-1	D12V	デジタル用 12V 出力	O	絶縁
CN5-2	IN20/HC0	デジタル入力/ハードウェアカウンタ	I	絶縁
CN5-3	IN21/HC1	デジタル入力/ハードウェアカウンタ	I	絶縁
CN5-4	IN22/HC2	デジタル入力/ハードウェアカウンタ	I	絶縁
CN5-5	IN23/HC3	デジタル入力/ハードウェアカウンタ	I	絶縁
CN5-6	IN24/HC4	デジタル入力/ハードウェアカウンタ	I	絶縁
CN5-7	IN25/HC5	デジタル入力/ハードウェアカウンタ	I	絶縁
CN5-8	IN26/HC6	デジタル入力/ハードウェアカウンタ	I	絶縁
CN5-9	IN27/HC7/ ADCLK	デジタル入力/ハードウェアカウンタ/ AD クロック入力	I	絶縁
CN5-10	COM	コモン	-	絶縁

表 21　CN6 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN6-1	D12V	デジタル用 12V 出力	O	絶縁
CN6-2	OUT00/PWM6	デジタル出力/PWM 出力	O	絶縁
CN6-3	OUT01	デジタル出力	O	絶縁
CN6-4	OUT02	デジタル出力	O	絶縁
CN6-5	OUT03	デジタル出力	O	絶縁
CN6-6	OUT04/PWM7	デジタル出力/PWM 出力	O	絶縁
CN6-7	OUT05	デジタル出力	O	絶縁
CN6-8	OUT06/PWM4	デジタル出力/PWM 出力	O	絶縁
CN6-9	OUT07	デジタル出力	O	絶縁
CN6-10	COM	コモン	-	絶縁

表 22　CN7 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN7-1	D12V	デジタル用 12V 出力	O	絶縁
CN7-2	OUT10/PWM2	デジタル出力/PWM 出力	O	絶縁
CN7-3	OUT11	デジタル出力	O	絶縁
CN7-4	OUT12	デジタル出力	O	絶縁
CN7-5	OUT13	デジタル出力	O	絶縁
CN7-6	OUT14/PWM3	デジタル出力/PWM 出力	O	絶縁
CN7-7	OUT15	デジタル出力	O	絶縁
CN7-8	OUT16	デジタル出力	O	絶縁
CN7-9	OUT17	デジタル出力	O	絶縁
CN7-10	COM	コモン	-	絶縁

表 23　CN8 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
CN8-1	D12V	デジタル用 12V 出力	O	絶縁
CN8-2	OUT20	デジタル出力	O	絶縁
CN8-3	OUT21	デジタル出力	O	絶縁
CN8-4	OUT22	デジタル出力	O	絶縁
CN8-5	OUT23	デジタル出力	O	絶縁
CN8-6	OUT24	デジタル出力	O	絶縁
CN8-7	OUT25	デジタル出力	O	絶縁
CN8-8	OUT26	デジタル出力	O	絶縁
CN8-9	OUT27	デジタル出力	O	絶縁
CN8-10	COM	コモン	-	絶縁

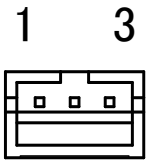


図 7　SER0, SER1 端子配列 [7]

表 24　SER0 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
SER0-1	TxD0	シリアル出力	O	非絶縁
SER0-2	RxD0	シリアル入力	I	非絶縁
SER0-3	GND	シグナルグランド	-	非絶縁

表 25　SER1 端子

ピン番	信号名	説明	方向	絶縁/非絶縁
SER1-1	TxD1	シリアル出力	O	非絶縁
SER1-2	RxD1	シリアル入力	I	非絶縁
SER1-3	GND	シグナルグランド	-	非絶縁

□　ディップスイッチ

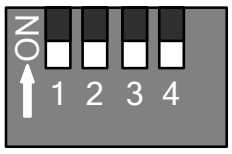


図 8　ディップスイッチ

表 26　ディップスイッチ

番号	説明
1	予約
2	予約
3	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。*
4	予約

*ディップスイッチ 3 番の操作は電源を切った状態で行ってください。

[7] 適合コネクタは「PHR-3」（日本圧着端子製造)。

4. 使用準備

□ DIN レール取付具の固定

DIN レール取付具は図 9 の向きで製品に取り付けます。製品は図 10 の向きになるように固定してください。

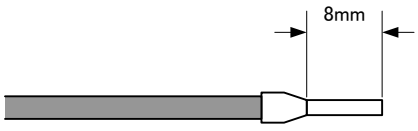
図 9　DIN レール取付具の取付け

図 10　DIN レールへの固定

□ 端子台への配線

付属するコネクタ端子台のスイッチ部分を押し込み、電線、または、棒端子（図 11 参照）を挿入してください。

【適合電線サイズ】
AWG26～20
【適合線断面積】
0.14～0.5mm^2



【適合フェルール端子断面積】
0.25～0.5*mm^2
(＊：プラスチックスリーブなし)

図 11　適合電線径と棒端子断面積

□ ドライバのインストール

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おこなってください。ドライバは付属 CD に納められています。

表 27　ドライバファイルの格納フォルダ

使用 OS	CD 内のドライバファイルの格納フォルダ
Windows 7, 8, 8.1, 10	¥TWUSB_DRIVER

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。

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 12　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。

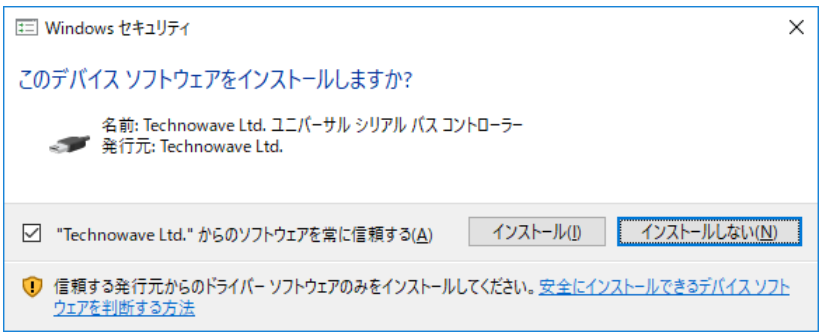


図 13　Windows 10 のドライバインストール画面（2）

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 14　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 19 のように「デバイス マネージャ」の画面に「USBX-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。

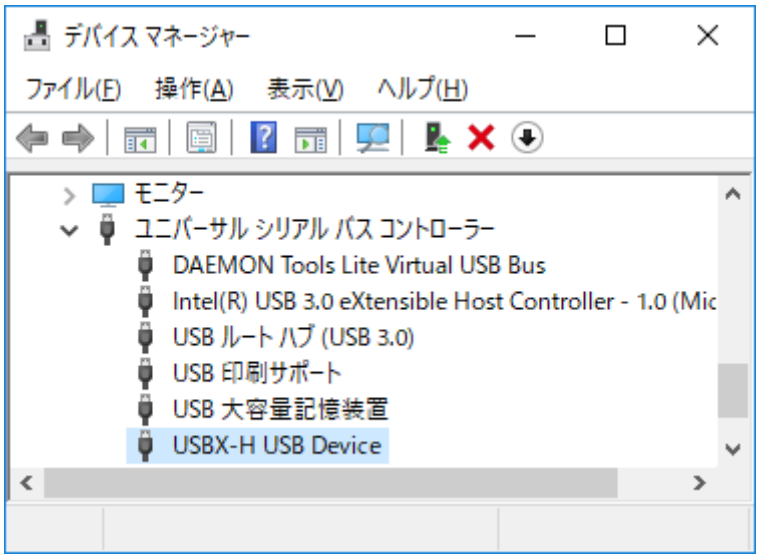


図 15　Windows 10 のドライバインストール確認

- 「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、[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。

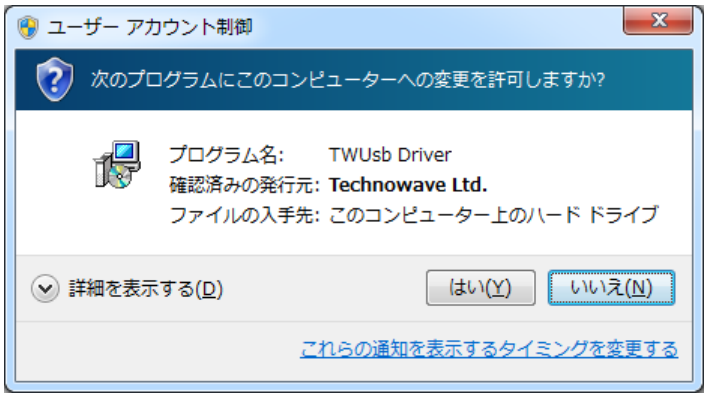


図 16　Windows 7 のドライバインストール画面（1）

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。

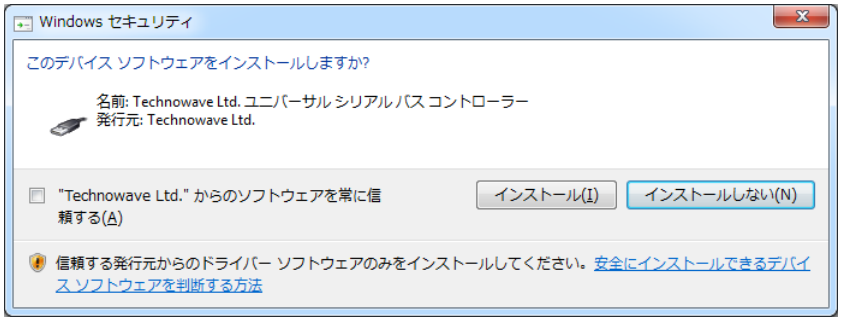


図 17　Windows 7 のドライバインストール画面（2）

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

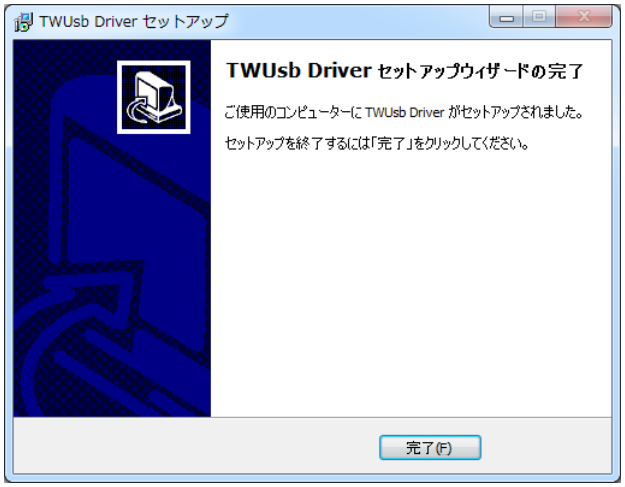


図 18　Windows 7 のドライバインストール画面（3）

⑤ デバイスを USB ケーブルでパソコンに接続します。図 19 のように「デバイス マネージャ」の画面に「USBX-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。

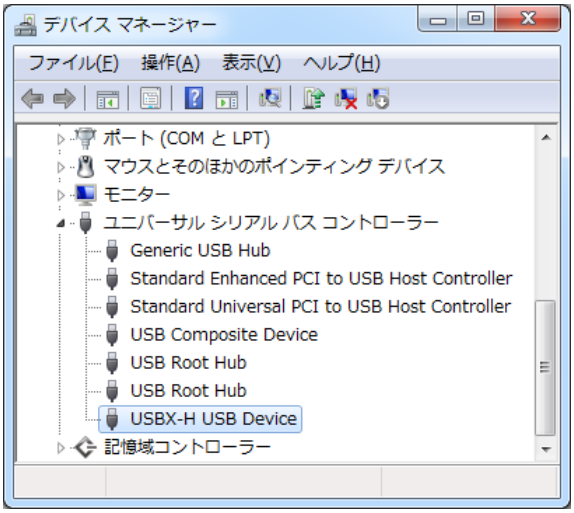


図 19　Windows 7 のドライバインストール確認

- 「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-I2424Tools」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 28 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した場合は、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 [8]。

表 28　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)	¥DLL	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、または、システムフォルダ(「C:¥Windows¥System32」など)
	TW_RX.DLL(32bit 版)		
	TWXA.DLL(32bit 版)		
64bit プログラムから制御する場合	USBM3069.DLL(64bit 版)	¥DLL¥x64	
	TW_RX.DLL(64bit 版)		
	TWXA.DLL(64bit 版)		

- 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。

8 ドライバのインストールは必要です。

□ LabVIEW 用 VI ライブラリのインストール

LabVIEW から製品を制御されたい場合、専用の LabVIEW 用 VI ライブラリをインストールしてください。LabVIEW 用 VI ライブラリは付属 CD に納められています。

表 29　インストールファイルの格納フォルダ

対応バージョン	CD 内の格納フォルダ
日本語版 2010 SP1 以降 [9]	¥VI_LIB¥TWXA-VI

VI ライブラリのインストール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。また、LabVIEW が起動中であれば終了してください。次に表 29 のフォルダから「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。

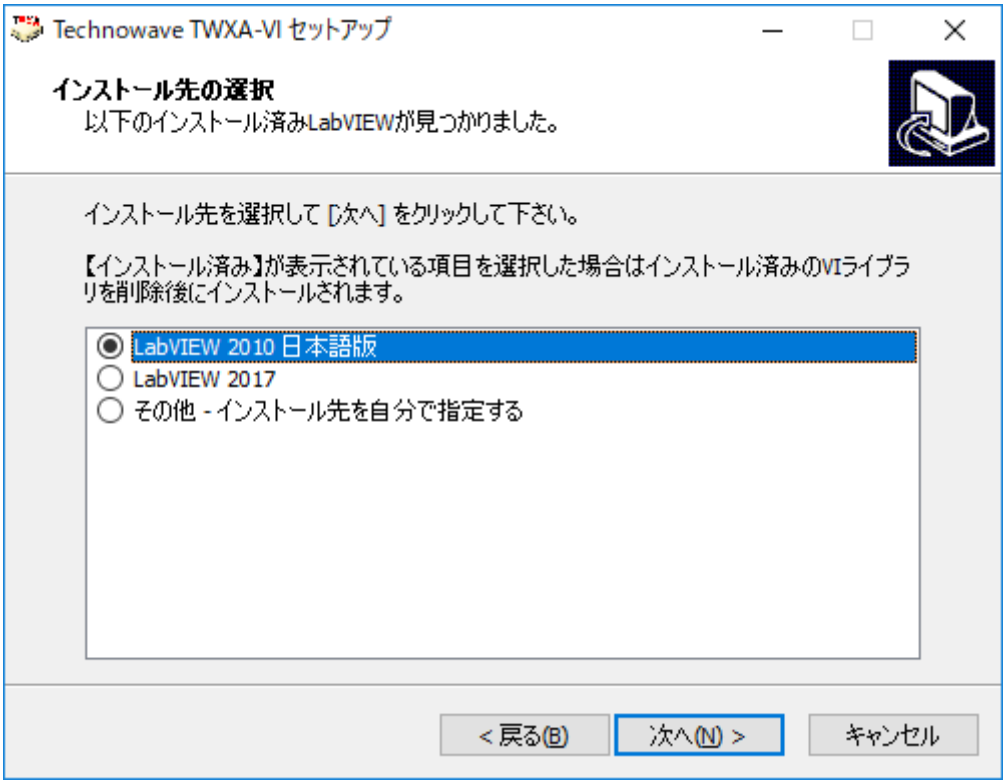


図 20　VI ライブラリのセットアップ画面

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、Windows 10 の場合[スタート]メニュー→[アプリの一覧]→[テクノウェーブ]の中に、Windows 7 の場合[スタート]メニュー→[すべてのプログラム]→[テクノウェーブ]→[TWXA-VI]の中に作られます。

[9] 32 ビット版のみ対応しています。

□ 設定ツールについて

24 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、Windows 10 の場合[スタート]メニュー→[アプリの一覧]→[テクノウェーブ]→[USBX-I2424Tools]から、Windows 7 の場合[スタート]メニュー→[すべてのプログラム]→[テクノウェーブ]→[USBX-I2424Tools]から起動することができます。

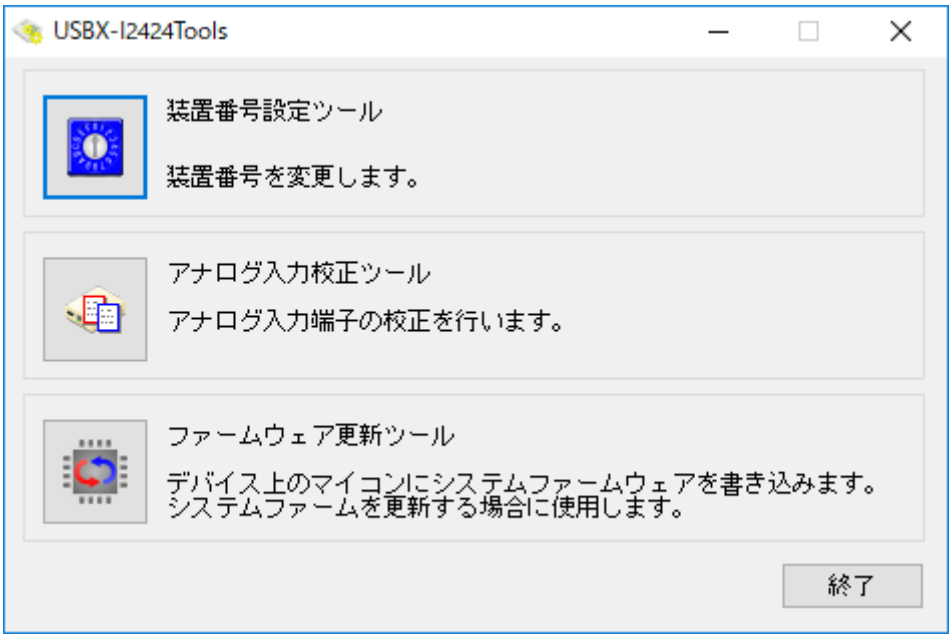


図 21　設定ツールのメニュー画面

表 30　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	装置番号を設定します。装置番号によって複数の製品を識別します。
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。

- 製品には制御用として RX62N マイコン(ルネサスエレクトロニクス)が搭載されています。マイコンにはホストパソコンからの命令を実行するための基本的なプログラムが組み込まれており、このプログラムのことを**システムファーム**と呼びます。
- 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 [10]。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、システムファームを更新する場合、まず最新の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。

[10] 弊社ホームページにて随時公開します。

□ 装置番号設定

製品に識別のための装置番号を付与します。

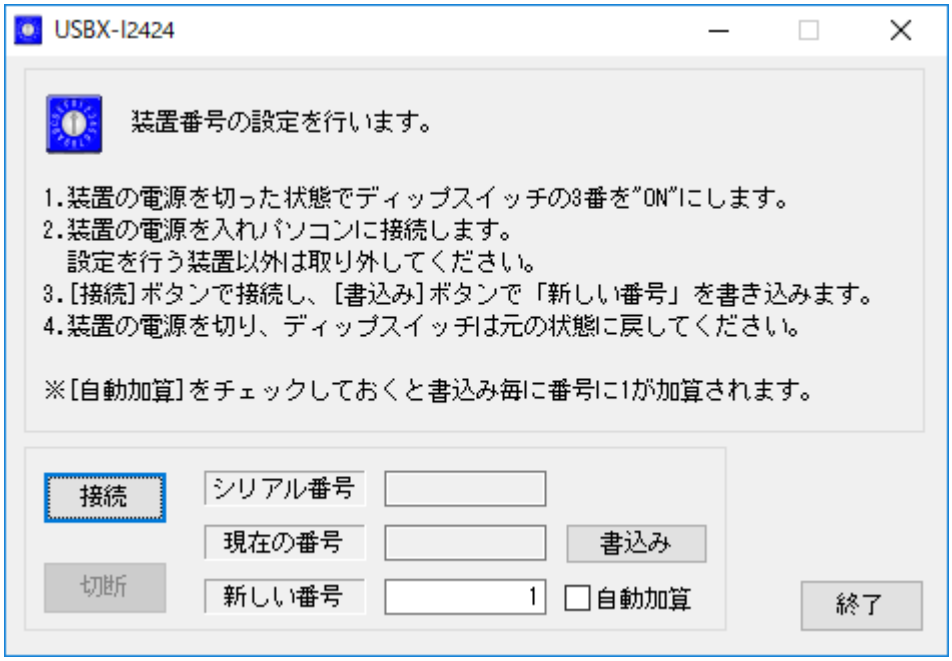


図 22　番号設定ツールの画面

1. 設定する製品のディップスイッチ 3 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(図 21)から[装置番号設定ツール]ボタンを押します。図 22 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。接続に成功すると[シリアル番号]の欄に接続した製品のシリアル番号が表示されます。設定する製品のシリアル番号であることを確認してください。
4. [新しい番号]に 1～65535 の範囲の数値を入力します。
5. [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。またこのとき、[自動加算]にチェックを入れておくと、書込みを行うたびに[新しい番号]が 1 ずつ増加します。TWXA ライブラリの関数からは設定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(*TWXA_Open()* 関数を参照してください)。
6. パソコンから製品を取り外しディップスイッチの 3 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 960,000 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、ご利用環境に適した構成を設定することができます。

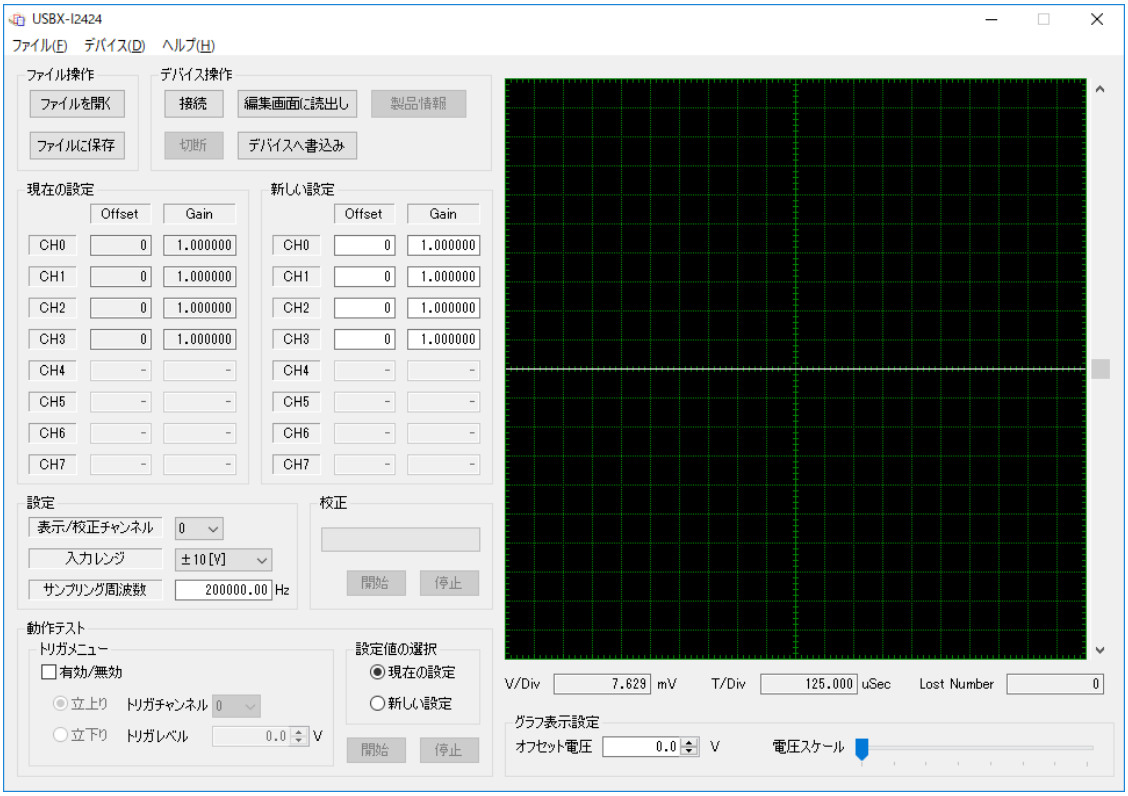


図 23　アナログ入力校正ツールの画面

製品に校正データが登録されている場合、AD 変換結果は校正データが反映された値が返されます。

5. ハードウェア

□ デジタル入力端子

入力回路

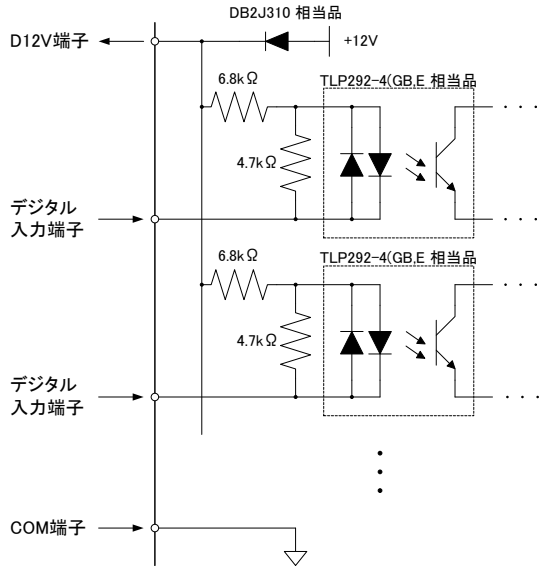


図 24　デジタル入力回路(IN0x,IN1x)

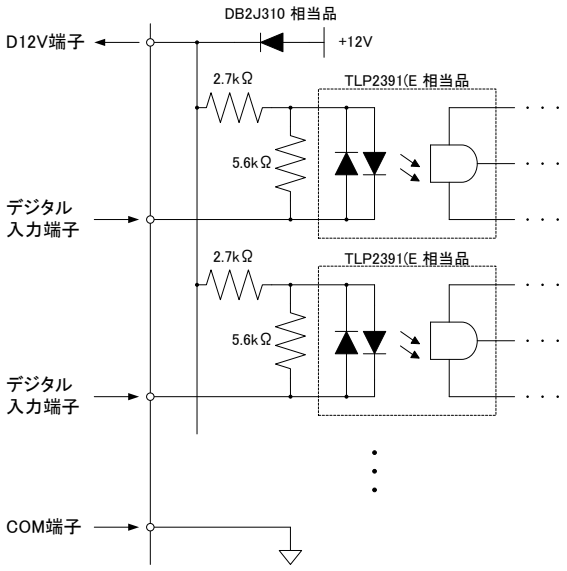


図 25　デジタル入力回路(IN2x)

接続例

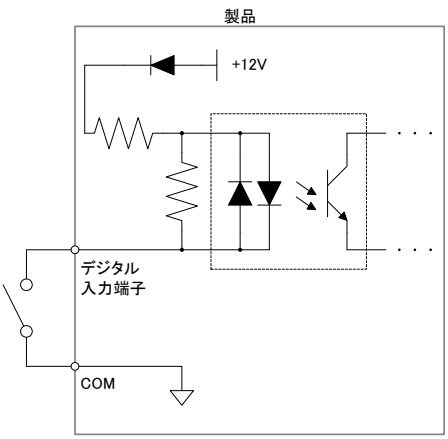


図 26　スイッチの接続例

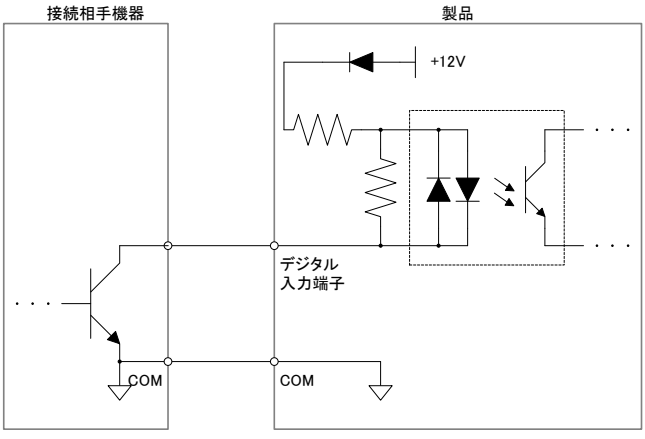


図 27　オープンコレクタ出力の機器との接続例

- 製品や接続相手機器の故障の原因となりますので、デジタル入力端子-COM 端子間に 25.2V を超える電圧を入力しないでください。
- 全ての COM 端子は製品内部で接続されています。

□ デジタル出力端子

出力回路

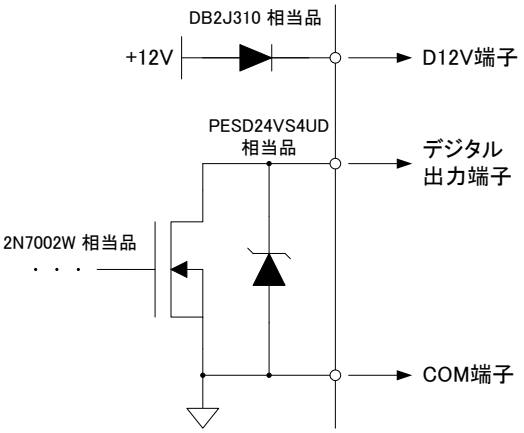


図 28　デジタル出力回路

接続例

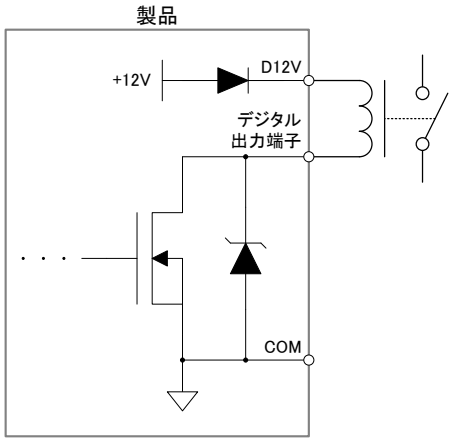


図 29　12V リレーの接続例

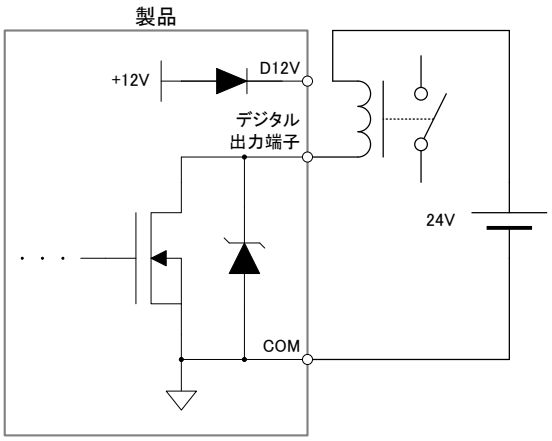


図 30　24V リレーの接続例

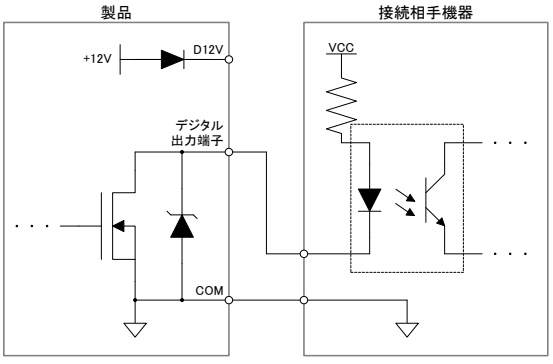


図 31 フォトカプラ入力機器(電源内蔵)との接続例

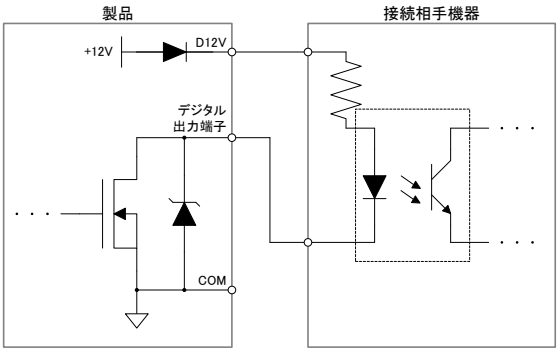


図 32 フォトカプラ入力機器(12V 入力)との接続例

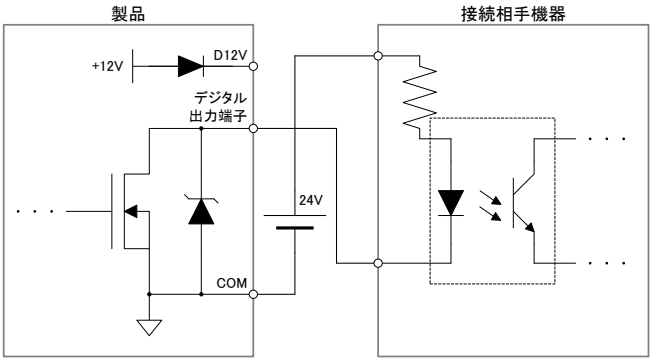


図 33　フォトカプラ入力機器(24V 入力)との接続例

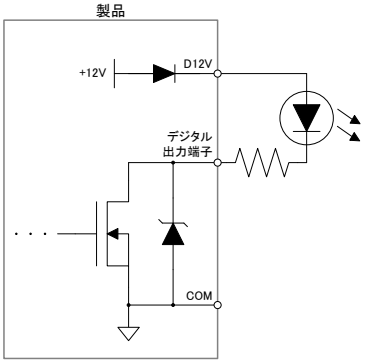


図 34　LED の接続例

□ 電源出力

D12V 端子から出力される 12V 電源は 10mA まで外部回路で使用可能です。また、利用しないデジタル入力端子がある場合には、その分の電流を外部回路に供給することが可能です。その場合、IN0x、IN1x 端子については 1 端子あたり 1.6mA、IN2x 端子については 4.1mA が外部回路で利用可能になります。

デジタル入力端子を全く使用しない場合、下記の計算のように 68.4mA まで利用可能となります。

$$10[mA] + (1.6[mA] \times 16) + (4.1[mA] \times 8) = 68.4[mA]$$

□ アナログ入力端子

入力回路

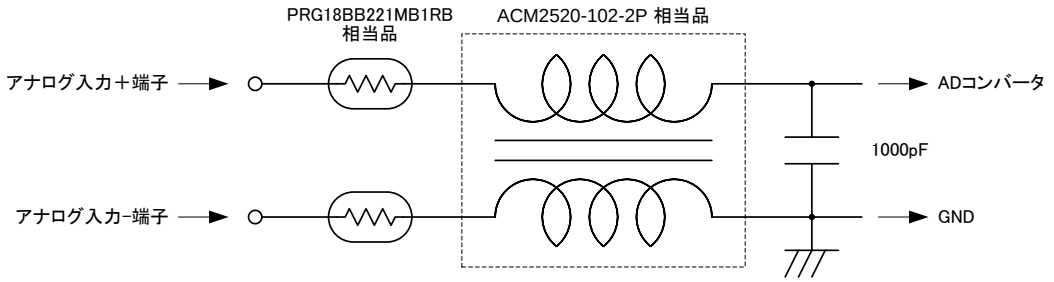


図 35　アナログ入力回路

- 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）

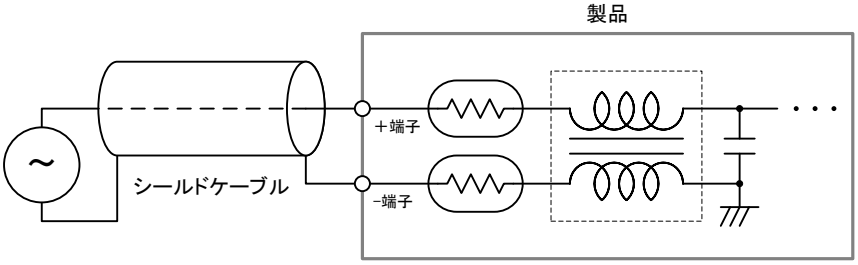


図 36　シングルエンド入力の接続例

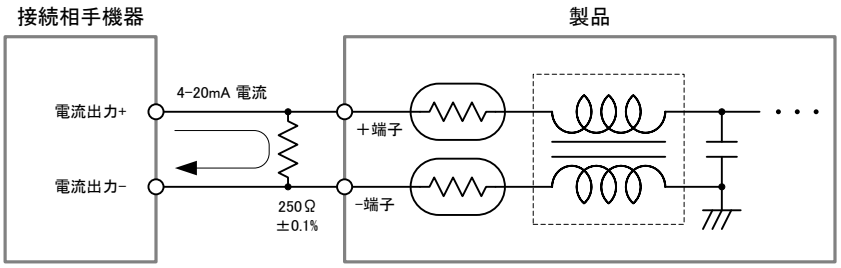


図 37　4-20mA 出力機器との接続例

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 38 に差動信号出力機器との接続例を示します。

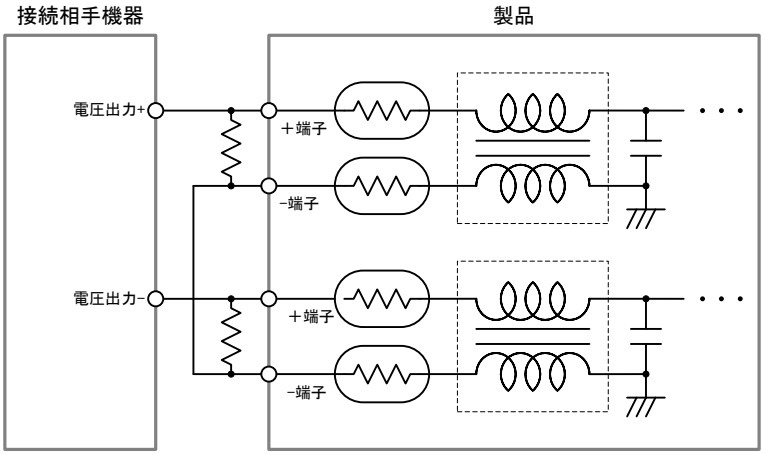


図 38　差動信号入力の接続例

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

□　アナログ出力端子

出力回路

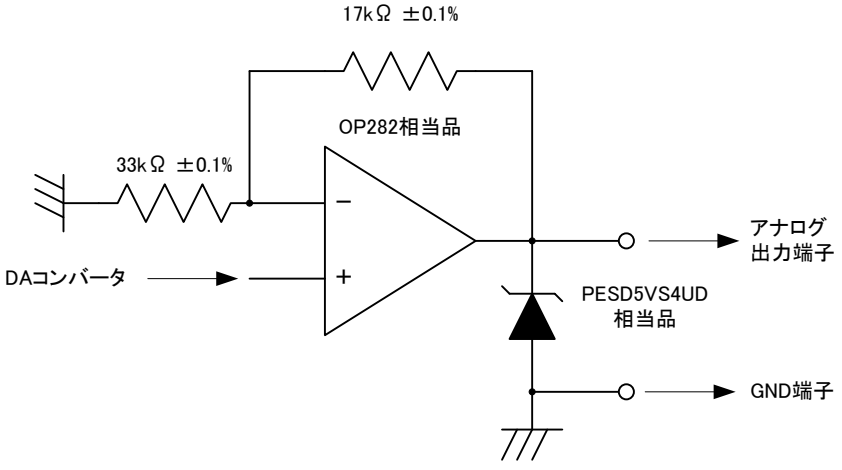


図 39　アナログ出力回路

接続例

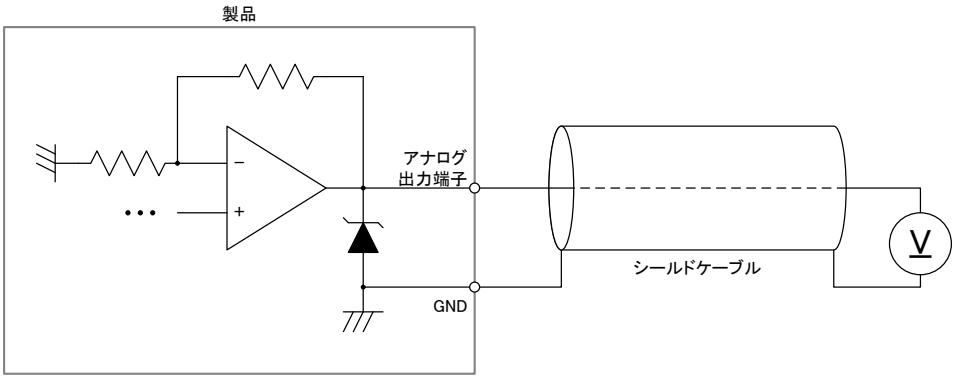


図 40　アナログ出力接続例

- アナログ出力の接続にはシールドケーブルを推奨します。
- アナログ出力端子から出力できる電流は最大 2mA です。

□ アナログ電源出力

A5V 端子は高精度の 5V 電圧を出力します。出力できる電流は最大 4mA ですので、外部機器などを駆動することはできません。サーミスタのリファレンス電圧など、負荷の小さい用途に利用してください。

□ シリアル(RS-232C)

図 41 はシリアルポートと一般的なパソコンのシリアルポートとの接続例です。

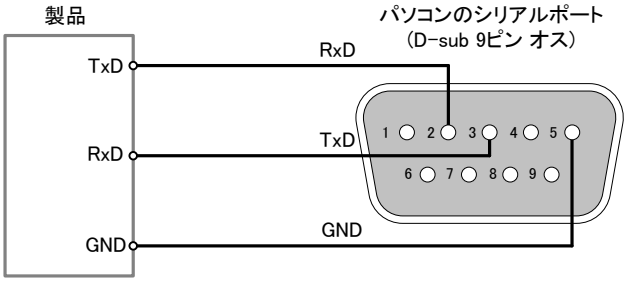


図 41 シリアルの接続例

パソコン以外の機器と接続する場合は、表 31、および、図 7 を参照してください。

表 31 シリアルの接続方法

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	TxD	出力	RxD(RD)	入力
2	RxD	入力	TxD(SD)	出力
3	GND	-	GND	-

6. プログラミング

サンプルプログラム は、付属 CD に収められています(表 32)。言語別に準備されていますので、必要に応じてご参照ください。

表 32　言語別のサンプルファイル

言語	付属 CD 内の格納フォルダ	ソリューションファイル
Visual C++(MFC) [11]	¥SAMPLE¥X2424_Samples	X2424SamplesMFC.sln
Visual Basic[11]		X2424SamplesVB.sln
Visual C#[11]		X2424SamplesCS.sln
Visual Basci for Application	¥SAMPLE¥X2424_Samples¥VBASamples	-
LabVIEW[12]	¥SAMPLE¥X2424_Samples¥LabVIEW_Samples	-

□　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 33 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-I2424Tools」をインストールした場合、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、Windows 10 の場合[スタート]メニュー→[アプリの一覧]→[テクノウェーブ]→[ライブラリ]を、Windows 7 の場合[スタート]メニュー→[すべてのプログラム]→[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 33　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
TWXA.lib(32bit 用)	TWXA ライブラリを静的にリンクするためのライブラリファイル	¥DLL
TWXA.lib(64bit 用)		¥DLL¥X64

「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")
```

[11] Visual Studio 2010 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)。

[12] LabVIEW 2010 SP1 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。

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。

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 34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-I2424Tools」をインストールした場合、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、Windows 10 の場合[スタート]メニュー→[アプリの一覧]→[テクノウェーブ]→[ライブラリ]を、Windows 7 の場合[スタート]メニュー→[すべてのプログラム]→[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 34 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
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 35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-I2424Tools」をインストールした場合、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、Windows 10の場合[スタート]メニュー→[アプリの一覧]→[テクノウェーブ]→[ライブラリ]を、Windows 7の場合[スタート]メニュー→[すべてのプログラム]→[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 35 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

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。

LabVIEW での開発に必要なファイル

表 36はLabVIEWで開発を行うために必要なファイルです。25ページの内容に従ってインストールすると、対象の LabVEIW のユーザーライブラリに TWXA-VI ライブラリが追加されます。

表 36　LabVIEW での開発に必要なファイル

ライブラリ名	説明
TWXA-VI ライブラリ	TWXA ライブラリを使用するための VI ライブラリ

□ 接続

製品を操作するには、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、接続中の製品を識別するIDと考えることができます(図 42)。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、ハンドルの値は操作を終了するまで記憶しておく必要があります。

また、製品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。

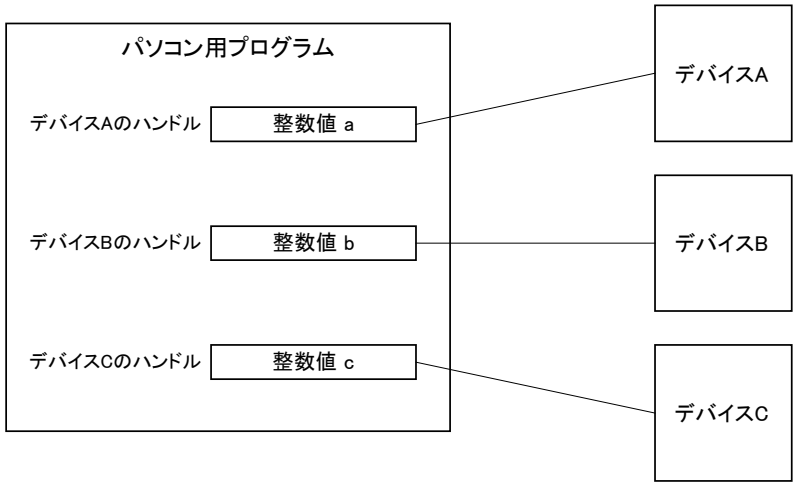


図 42　ハンドル

表 37　接続、初期化、終了に使用する関数

関数名	説明
TWXA_Open()	デバイスに接続してハンドルを取得します。
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 38 の *TWXA_Open()* 関数を使用します。

表 38　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。接続に成功すると引数 *phDev* にハンドルを返します。装置番号の設定方法は 26 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番のデバイスに接続
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。

□ デジタル入出力

デバイスが使用できるデジタル入力端子、デジタル出力端子を表 39 に示します。入力端子／出力端子は 8 つの端子を 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。

表 39　入出力端子

端子名	端子数	方向	ポート名	兼用端子
IN00-IN07	8	入力	PIN0	SC0-SC7, ADTRG
IN10-IN17	8	入力	PIN1	-
IN20-IN27	8	入力	PIN2	HC0-HC7, ADCLK
OUT00-OUT07	8	出力	POUT0	PWM4, PWM6, PWM7
OUT10-OUT17	8	出力	POUT1	PWM2, PWM3
OUT20-OUT27	8	出力	POUT2	-

入力端子、出力端子はそれぞれ、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。入力端子は入力ポートと、出力端子は出力ポートと 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端子の状態の読み取り、出力ポートへの書き込みは出力端子状態の変更と等価です。

入出力ポートの制御には、表 40 の関数を使用します。また、表 41 はデジタル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 40　デジタル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 41　デジタル入出力のサンプルプログラム

開発環境	プロジェクト名/ファイル名	説明
Visual C++ (MFC)	PortSample	入出力端子の状態を表示し、出力端子の状態を操作できます。
Visual Basic	PortSampleVB	
Visual C#	PortSampleCS	
LabVIEW	PortSample.vi	
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。

端子の状態を読み取る

端子の状態を読み取るには *TWXA_PortRead()* 関数を使用します。*Port* 引数で読み出したいポートを指定します。

表 42　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 43　TWXA_PortRead()の Port 引数に指定する値

言語	値	説明
C/C++	TWXA_PIN0	IN00-IN07 の入力状態を読み取ります。
C++	TWXA::RPORT::PIN0	
VB/VBA	TWXA_RPORT.PIN0	
C#	TWXA.RPORT.PIN0	
C/C++	TWXA_PIN1	IN10-IN17 の入力状態を読み取ります。
C++	TWXA::RPORT::PIN1	
VB/VBA	TWXA_RPORT.PIN1	
C#	TWXA.RPORT.PIN1	
C/C++	TWXA_PIN2	IN20-IN27 の入力状態を読み取ります。
C++	TWXA::RPORT::PIN2	
VB/VBA	TWXA_RPORT.PIN2	
C#	TWXA.RPORT.PIN2	
C/C++	TWXA_POUT0	OUT00-OUT07 の出力状態を読み取ります。
C++	TWXA::RPORT::POUT0	
VB/VBA	TWXA_RPORT.POUT0	
C#	TWXA.RPORT.POUT0	
C/C++	TWXA_POUT1	OUT10-OUT17 の出力状態を読み取ります。
C++	TWXA::RPORT::POUT1	
VB/VBA	TWXA_RPORT.POUT1	
C#	TWXA.RPORT.POUT1	
C/C++	TWXA_POUT2	OUT20-OUT27 の出力状態を読み取ります。
C++	TWXA::RPORT::POUT2	
VB/VBA	TWXA_RPORT.POUT2	
C#	TWXA.RPORT.POUT2	

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 PIN0 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端子の入力値と対応しています。

表 44　データビットと端子の関係

ビット	7(MSB)	6	5	4	3	2	1	0(LSB)
対応端子	IN07	IN06	IN05	IN04	IN03	IN02	IN01	IN00

対応する端子が"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。

出力端子の状態を変更する

出力端子の状態を変更するには *TWXA_PortWrite()* 関数を使用します。

表 45　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 46　TWXA_PortWrite()の Port 引数に指定する値

言語	値	説明
C/C++	TWXA_POUT0	OUT00-OUT07 の出力状態を変更します。
C++	TWXA::WPORT::POUT0	
VB/VBA	TWXA_WPORT.POUT0	
C#	TWXA.WPORT.POUT0	
C/C++	TWXA_POUT1	OUT10-OUT17 の出力状態を変更します。
C++	TWXA::WPORT::POUT1	
VB/VBA	TWXA_WPORT.POUT1	
C#	TWXA.WPORT.POUT1	
C/C++	TWXA_POUT2	OUT20-OUT27 の出力状態を変更します。
C++	TWXA::WPORT::POUT2	
VB/VBA	TWXA_WPORT.POUT2	
C#	TWXA.WPORT.POUT2	

読出しと同様に 8 ビット単位でデータを書き込みます。データビットと端子との関係は読出しの場合と同様で、"0"を書き込んだビットと対応する端子は"OFF"となり、"1"を書き込んだビットと対応する端子は"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* のうち"0"となっているビットは影響を受けません。図 43 は H'55 というデータを、*Mask* を H'0F として出力した例です。

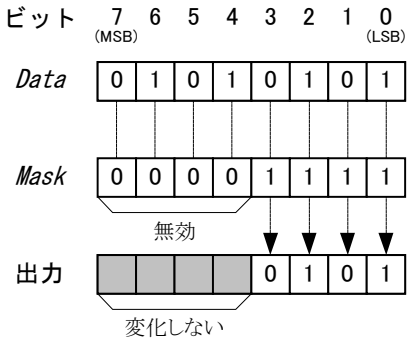


図 43　出力のマスク

リスト 6　デジタル入出力の例(C 言語)

```
BYTE bData;

//IN00-IN07 の読み出し
TWXA_PortRead(hDev, TWXA_PIN0, &bData);

//OUT07 だけを"ON"にし、OUT00-OUT06 は変更しない
TWXA_PortWrite(hDev, TWXA_POUT0, 0xff, 0x80);
```

リスト 7　デジタル入出力の例(Visual Basic)

```
Dim bData As Byte

'IN00-IN07 の読み出し
TWXA_PortRead(hDev, TWXA_RPORT.PIN0, bData)

'OUT07 だけを"ON"にし、OUT00-OUT06 は変更しない
TWXA_PortWrite(hDev, TWXA_WPORT.POUT0, &HFF, &H80)
```

リスト 8　デジタル入出力の例(C#)

```
byte bData;

//IN00-IN07 の読み出し
TWXA.PortRead(hDev, TWXA.RPORT.PIN0, out bData);

//OUT07 だけを"ON"にし、OUT00-OUT06 は変更しない
TWXA.PortWrite(hDev, TWXA.WPORT.POUT0, 0xff, 0x80);
```

- 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 39 ページを、エラー処理については 106 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。

□ 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 4 チャンネル搭載しています。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。アナログ入力に使用する端子は AD0～AD3 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」のどちらかをソフトウェア上から選択します。

表 47 はアナログ入力を制御するための関数です。表 48 はアナログ入力のサンプルプログラムです。

表 47　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() *TWXA_AnToVolt()*	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_ADGetRange()	アナログ入力端子に設定されている入力レンジを取得します。
TWXA_ADSetMode()	16 ビット AD コンバータの動作モードを設定します。
TWXA_ADGetMode()	16 ビット AD コンバータに設定されている動作モードを取得します。
TWXA_ADStartAutoSampling()	アナログ入力の連続サンプリングを開始します。
TWXA_ADStartAutoSamplingEx()	アナログ入力の連続サンプリングを開始します。 AD 変換開始トリガを指定できます。
TWXAADStartExtSyncSampling()	外部クロックを使用した連続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()	パソコンの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()	パソコンの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()	パソコンのバッファをクリアします。

表 48　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/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	
LabVIEW	AnalogSample.vi	
Visual C++ (MFC)	AnalogAutoSample	サンプリングし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_ADStartAutoSampling()* 、および、*TWXA_ADStartExtSyncSampling()* を使用したサンプルプログラムです。
Visual Basic	AnalogAutoSampleVB	
Visual C#	AnalogAutoSampleCS	
LabVIEW	AnalogAutoSample.vi	
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 49 の *TWXA_ADSetRange()* 関数を使用します。*Range* 引数には表 50 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 49　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 50　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 51 のようになります。

表 51　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。

オーバーサンプリングレートの設定

オーバーサンプリングとは、指定の周波数よりも高い周波数で AD 変換を行うことをいいます。

デバイスの起動直後、および、初期化後は、オーバーサンプリング機能を使用しない設定となります。オーバーサンプリング機能の設定を変更するには表 52 の *TWXA_ADSetMode()* 関数を使用します。*Mode* 引数には表 53 のオーバーサンプリングレートを指定します。

オーバーサンプリングレートを変更する際は、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 52　TWXA_ADSetMode() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADSetMode(TW_HANDLE hDev, long Mode)
VB	Function TWXA_ADSetMode(ByVal hDev As System.IntPtr, ByVal Mode As TWXA_AN_MODE) As Integer
VBA	Function TWXA_ADSetMode(ByVal hDev As Long, ByVal Mode As TWXA_AN_MODE) As Long
C#	STATUS ADSetMode(System.IntPtr hDev, AN_MODE Mode)

表 53　TWXA_ADSetMode() の Mode 引数に指定する値

言語	値	説明
C/C++	TWXA_AN_OSR_NON	オーバーサンプリング機能を使用しません。 （デフォルト）
C++	TWXA::AN_MODE::OSR_NON	
VB/VBA	TWXA_AN_MODE.OSR_NON	
C#	TWXA.AN_MODE.OSR_NON	
C/C++	TWXA_AN_OSR2	オーバーサンプリングレートを 2 に設定します。
C++	TWXA::AN_MODE::OSR2	
VB/VBA	TWXA_AN_MODE.OSR2	
C#	TWXA.AN_MODE.OSR2	
C/C++	TWXA_AN_OSR4	オーバーサンプリングレートを 4 に設定します。
C++	TWXA::AN_MODE::OSR4	
VB/VBA	TWXA_AN_MODE.OSR4	
C#	TWXA.AN_MODE.OSR4	
C/C++	TWXA_AN_OSR8	オーバーサンプリングレートを 8 に設定します。
C++	TWXA::AN_MODE::OSR8	
VB/VBA	TWXA_AN_MODE.OSR8	
C#	TWXA.AN_MODE.OSR8	
C/C++	TWXA_AN_OSR16	オーバーサンプリングレートを 16 に設定します。
C++	TWXA::AN_MODE::OSR16	
VB/VBA	TWXA_AN_MODE.OSR16	
C#	TWXA.AN_MODE.OSR16	
C/C++	TWXA_AN_OSR32	オーバーサンプリングレートを 32 に設定します。
C++	TWXA::AN_MODE::OSR32	
VB/VBA	TWXA_AN_MODE.OSR32	
C#	TWXA.AN_MODE.OSR32	
C/C++	TWXA_AN_OSR64	オーバーサンプリングレートを 64 に設定します。
C++	TWXA::AN_MODE::OSR64	
VB/VBA	TWXA_AN_MODE.OSR64	
C#	TWXA.AN_MODE.OSR64	

TWXA_ADSetMode() 関数によりオーバーサンプリングレートを設定すると、以降全ての AD 変換はオーバーサンプリングレートと同じ回数のサンプリングが行われ、その平均値が変換結果として読み出されます（図 44）。

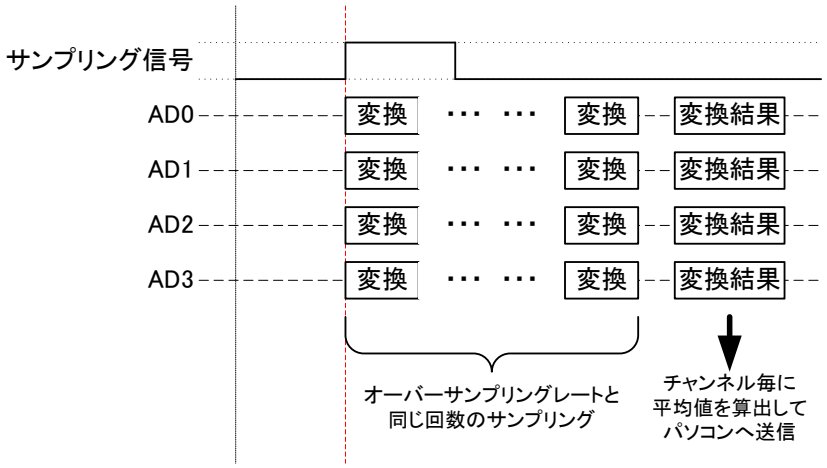


図 44　オーバーサンプリングレートと変換結果の関係

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 3 つの方法があります。

- *TWXA_ADRead()* 関数を使用して、命令発行時のアナログ電圧値を読み出す方法。
- *TWXA_ADStartAutoSampling()* 関数、または、*TWXA_ADStartAutoSamplingEx()* 関数を使用して、内部タイマに同期した連続サンプリングを行う方法。
- *TWXA_ADStartExtSyncSampling()* 関数を使用して、外部クロックに同期した連続サンプリングを行う方法。

表 54 は、それぞれの変換方法の特徴をまとめたものです。

表 54　AD 変換の方法と特徴

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。 変換するチャンネルを指定することが可能です。
TWXA_ADStartAutoSampling() *TWXA_ADStartAutoSamplingEx()*	1～50,000[Hz]*1	内部タイマに同期した連続サンプリングを行います。 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 1sec 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_ADStartExtSyncSampling()	～50,000[Hz]*1	外部クロックに同期した連続サンプリングを行います。 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 1sec 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。

*1 設定可能なサンプリングレートでも USB ポート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。

読み出した全ての変換値は表 55 の *TWXA_An16ToVolt()* 関数、または、表 56 の *TWXA_AnToVolt()* 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。*TWXA_AnToVolt()*

関数の *Bit* 引数には AD コンバータの分解能 16 を指定してください。

表 55　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 56　TWXA_AnToVolt() の関数宣言

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

表 57　TWXA_An16ToVolt()、および、TWXA_AnToVolt() 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	

命令発行時のアナログ電圧値を読み出す

TWXA_ADRead() 関数(表 58)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 58　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)。一定周期で関数を呼び出した場合でも変換間隔が一定になるとは限りませんので、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、0～3 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 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。

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。

TWXA_ADSetMode() 関数によりオーバーサンプリングレートを設定すると、オーバーサンプリングレートと同じ回数のサンプリングが行われ、その平均値が変換結果として *pData* 引数に格納されます(49 ページ、図 44)。

リスト 9　TWXA_ADRead()の使用例 (C 言語)

```
long lData[4];
double dVolt;

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

//オーバーサンプリングレートを 4 に設定
TWXA_ADSetMode(hDev, TWXA_AN_OSR4);

//AD0-AD3 の AD 変換結果を読み出し
TWXA_ADRead(hDev, TWXA_AD_ALL, lData);

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

//AD0 へ差動信号の[+]、AD1 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 10　TWXA_ADRead()の使用例（Visual Basic）

```
Dim iAD(3) 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)

'オーバーサンプリングレートを 4 に設定
TWXA_ADSetMode(hDev, TWXA_AN_MODE.OSR4)

'AD0-AD3 の AD 変換結果を読み出し
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)

'AD0 へ差動信号の[+]、AD1 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 11　TWXA_ADRead()の使用例（C#）

```
int[] iAD = new int[4];
double dVolt;

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

//オーバーサンプリングレートを 4 に設定
TWXA.ADSetMode(hDev, TWXA.AN_MODE.OSR4);

//AD0-AD3 の AD 変換結果を読み出し
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);

//AD0 へ差動信号の[+]、AD1 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);
```

内部タイマに同期した連続サンプリングを行う

TWXA_ADStartAutoSampling() 関数(表 59)、または、*TWXA_ADStartAutoSamplingEx()* 関数(表 60)を使用します。

表 59　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 60　TWXA_ADStartAutoSamplingEx() の関数宣言

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

表 61　TWXA_ ADStartAutoSamplingEx() の Opt 引数に指定する値

言語	値	説明
C/C++	TWXA_AN_TRIG_NON	本関数の呼び出しで、直ちに連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_NON	
VB/VBA	TWXA_AN_MODE.TRIG_NON	
C#	TWXA.AN_MODE.TRIG_NON	
C/C++	TWXA_AN_TRIG_OFF_TO_ON	AD トリガ入力端子[ADTRG]の状態が[OFF]から[ON]になると連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_OFF_TO_ON	
VB/VBA	TWXA_AN_MODE.TRIG_OFF_TO_ON	
C#	TWXA.AN_MODE.TRIG_OFF_TO_ON	
C/C++	TWXA_AN_TRIG_ON_TO_OFF	AD トリガ入力端子[ADTRG]の状態が[ON]から[OFF]になると連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_ON_TO_OFF	
VB/VBA	TWXA_AN_MODE.TRIG_ON_TO_OFF	
C#	TWXA.AN_MODE.TRIG_ON_TO_OFF	
C/C++	TWXA_AN_TRIG_BOTH	AD トリガ入力端子[ADTRG]の状態が変化すると連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_BOTH	
VB/VBA	TWXA_AN_MODE.TRIG_BOTH	
C#	TWXA.AN_MODE.TRIG_BOTH	

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()* 関数、および、*TWXA_ADStartAutoSamplingEx()* 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オーバーサンプリングレートの設定により異なります(設定可能なサンプリングレートでも、USB ポートの通信状態や使用環境により、サンプリングデータをすべて転送できない場合があります)。オーバーサンプリングレートの設定は *TWXA_ADSetMode()* 関数で行います。

表 62　オーバーサンプリングレートと設定可能なサンプリングレートの関係

オーバーサンプリングレート	設定可能周波数範囲[Hz]
未使用	1-50,000
2	
4	
8	1-25,000
16	1-12,500
32	1-6,250
64	1-3,125

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。

Opt 引数には、連続サンプリングの開始タイミングを指定することができます。

オーバーサンプリングレートを設定すると、オーバーサンプリングレートと同じ回数のサンプリングが行われ、その平均値が変換結果としてパソコンへ送信されます。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 45 と図 46 に示します。

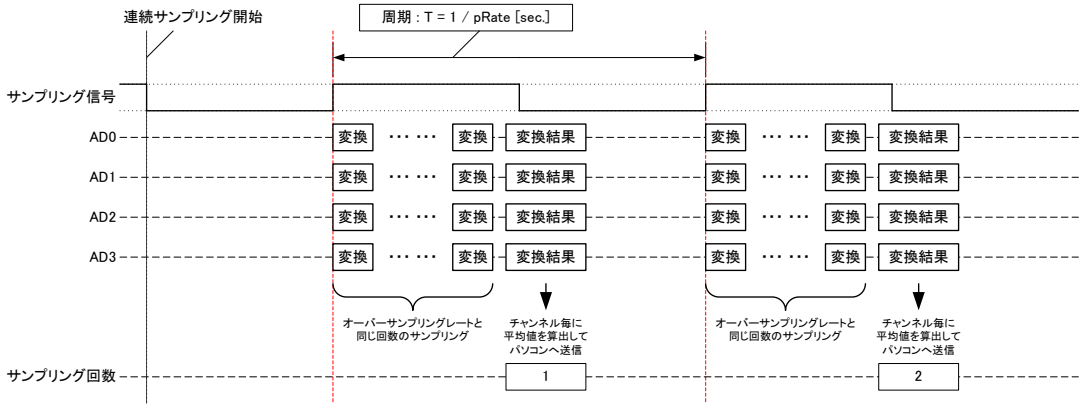


図 45　サンプリングレートと変換タイミングの関係（関数呼び出しによるサンプリング開始）

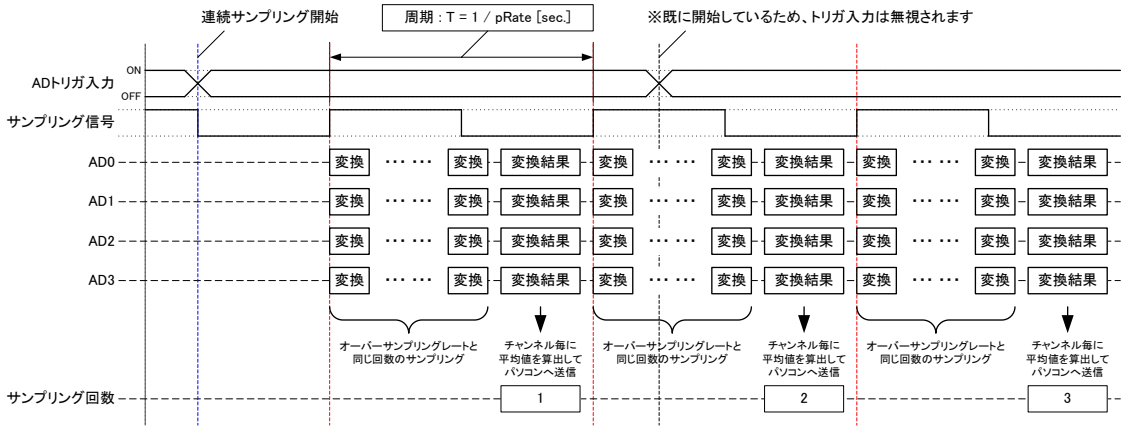


図 46　サンプリングレートと変換タイミングの関係（トリガ入力によるサンプリング開始）

TWXA_ADStartAutoSampling() 関数、*TWXA_ADStartAutoSamplingEx()* 関数を呼び出すと、デバイスは連続サンプリングを直ちに開始するか、開始トリガの入力待ち状態となりますが、関数自体はすぐに返ります。

連続サンプリングの開始後、サンプリングデータは一度製品内部のバッファ [13]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 [14]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等、他の機能の操作を行うことができます。

- ソフトウェアカウンタ CH7 が 2 相パルスカウントモード、または、3 相パルスカウントモードに設定されている場合、[IN07/SC7/ADTRG]を AD トリガ入力端子として使用できません。端子を AD トリガ入力として使用する場合、ソフトウェアカウンタ CH7 を単相カウントモードに変更してください（76 ページを参照）。

[13] 300 データ分をバッファできます。

[14] 65536 データ分をバッファできます。

外部クロックに同期した連続サンプリングを行う

TWXA_ADStartExtSyncSampling() 関数(表 63)を使用します。

表 63　TWXA_ADStartExtSyncSampling() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_ADStartExtSyncSampling(TW_HANDLE hDev, DWORD nSampling, long Opt)
VB	Function TWXA_ADStartExtSyncSampling(ByVal hDev As System.IntPtr, ByVal nSampling As Integer, ByVal Opt As TWXA_AN_MODE) As Integer
VBA	Function TWXA_ADStartExtSyncSampling(ByVal hDev As Long, ByVal nSampling As Long, ByVal Opt As TWXA_AN_MODE) As Long
C#	STATUS ADStartExtSyncSampling(System.IntPtr hDev, uint nSampling, AN_MODE Opt) STATUS ADStartExtSyncSampling(System.IntPtr hDev, uint nSampling) STATUS ADStartExtSyncSampling(System.IntPtr hDev)

表 64　TWXA_ADStartExtSyncSampling() の Opt 引数に指定する値

言語	値	説明
C/C++	TWXA_AN_SYNC_OFF_TO_ON	AD クロック入力端子[ADCLK]が[OFF]から[ON]になるとサンプリングを 1 回行います。
C++	TWXA::AN_MODE::SYNC_OFF_TO_ON	
VB/VBA	TWXA_AN_MODE.SYNC_OFF_TO_ON	
C#	TWXA.AN_MODE.SYNC_OFF_TO_ON	
C/C++	TWXA_AN_SYNC_ON_TO_OFF	AD クロック入力端子[ADCLK]が[ON]から[OFF]になるとサンプリングを 1 回行います。
C++	TWXA::AN_MODE::SYNC_ON_TO_OFF	
VB/VBA	TWXA_AN_MODE.SYNC_ON_TO_OFF	
C#	TWXA.AN_MODE.SYNC_ON_TO_OFF	
C/C++	TWXA_AN_TRIG_NON	本関数の呼び出しで、直ちに連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_NON	
VB/VBA	TWXA_AN_MODE.TRIG_NON	
C#	TWXA.AN_MODE.TRIG_NON	
C/C++	TWXA_AN_TRIG_OFF_TO_ON	AD トリガ入力端子[ADTRG]の状態が[OFF]から[ON]になると連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_OFF_TO_ON	
VB/VBA	TWXA_AN_MODE.TRIG_OFF_TO_ON	
C#	TWXA.AN_MODE.TRIG_OFF_TO_ON	
C/C++	TWXA_AN_TRIG_ON_TO_OFF	AD トリガ入力端子[ADTRG]の状態が[ON]から[OFF]になると連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_ON_TO_OFF	
VB/VBA	TWXA_AN_MODE.TRIG_ON_TO_OFF	
C#	TWXA.AN_MODE.TRIG_ON_TO_OFF	
C/C++	TWXA_AN_TRIG_BOTH	AD トリガ入力端子[ADTRG]の状態が変化すると連続サンプリングを開始します。
C++	TWXA::AN_MODE::TRIG_BOTH	
VB/VBA	TWXA_AN_MODE.TRIG_BOTH	
C#	TWXA.AN_MODE.TRIG_BOTH	

入力可能な外部クロックの周波数はオーバーサンプリングレートの設定により異なります(入力可能な周波数でも、USB ポートの通信状態や使用環境により、サンプリングデータをすべて転送できない場合があります)。オーバーサンプリングレートの設定は *TWXA_ADSetMode()* 関数で行います。

表 65　オーバーサンプリングレートと入力可能な外部クロックの関係

オーバーサンプリングレート	入力可能最大周波数[Hz]
未使用	50, 000
2	
4	
8	25, 000
16	12, 500
32	6, 250
64	3, 125

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。

Opt 引数には、連続サンプリングの開始タイミング、および、外部クロック信号による変換タイミングを指定することができます。

オーバーサンプリングレートを設定すると、オーバーサンプリングレートと同じ回数のサンプリングが行われ、その平均値が変換結果としてパソコンへ送信されます。外部クロックを使用した場合の各チャンネルの変換タイミングを図 47 に示します。

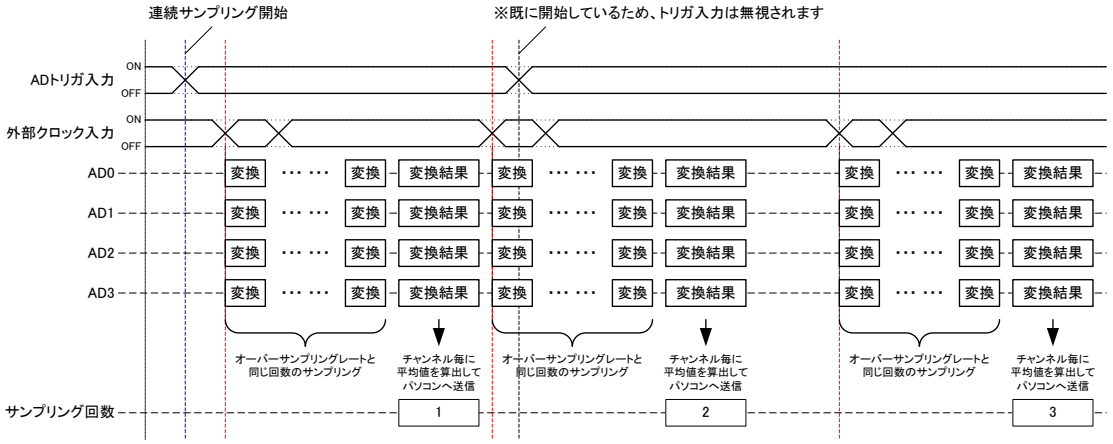


図 47　外部クロックを使用した変換タイミング（トリガ入力によるサンプリング開始）

TWXA_ADStartExtSyncSampling() 関数を呼び出すと、デバイスは連続サンプリングを直ちに開始するか、開始トリガの入力待ち状態となりますが、関数自体はすぐに返ります。

TWXA_ADStartAutoSampling() 関数や *TWXA_ADStartAutoSamplingEx()* 関数と同様に、連続サンプリングの開始後、サンプリングデータは一度製品内部のバッファ[13]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[14]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等、他の機能の操作を行うことができます。

- ソフトウェアカウンタ CH7 が 2 相パルスカウントモード、または、3 相パルスカウントモードに設定されている場合、[IN07/SC7/ADTRG]を AD トリガ入力端子として使用できません。端子を AD トリガ入力として使用する場合、ソフトウェアカウンタ CH7 を単相カウントモードに変更してください(76 ページを参照)。
- ハードウェアカウンタ CH7 が設定されている場合、または、PWM 出力 CH7 が外部クロック入力動作に設定されている場合、 [IN27/HC7/ADCLK]を AD クロック入力端子として使用できません。端子を AD クロック入力として使用する場合、ハードウェアカウンタ CH7 のモードを解除する(71 ページを参照)、または、PWM 出力 CH7 を内部タイマで動作するように変更してください(82 ページを参照)。

連続サンプリングを停止する

TWXA_ADStopSampling() 関数(表 66)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 66 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パソコンの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 67)、受信バッファからデータを読み出すには *TWXA_ADReadBuffer()* 関数(表 68)を使用します。*TWXA_ADReadBuffer()* 関数の *pData* 引数には *TWXA_A0x0x_DATA* 構造体(表 69)の配列を渡します。

表 67 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 68　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 69　TWXA_A0x0x_DATA 構造体の宣言

言語	関数宣言

C/C++	typedef struct tagTwxaAOxOxData
	{
	  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_AOxOx_DATA
	  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、インデックスは破棄されたデータ数だけ欠落します。

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。配列のインデックス4～7 は常に "0" が格納されます。

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 12　TWXA_ADStartAutoSampling()の使用例（C 言語）

```
double dRate = 1000.0; //サンプリングレート = 1,000S/sec
long nData;
int iStatus;
DWORD i, nIndex, nLost;
TWXA_AOxOx_DATA Data[1000];
TCHAR c[256];

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

//オーバーサンプリングレートを 4 に設定
TWXA_ADSetMode(hDev, TWXA_AN_OSR4);

//連続サンプリングを開始
TWXA_ADStartAutoSampling(hDev, &dRate, 0xFFFFFFFFUL);

while (1)
{
  TWXA_ADGetQueueStatus(hDev, & iStatus, &nData); //受信データ数を取得
  if (1000UL <= nData) break; //必要な回数のサンプリングが終了したら抜ける
}

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

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

//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
nIndex = Data[0].Index + 1;
for (i = 1, nLost = 0; i < nData; i++)
{
  if (Data[i].Index != nIndex)
  {
    nLost += (nIndex - Data[i].Index);
    nIndex = Data[i].Index;
  }
  nIndex++;
}

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

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

```
Dim dRate As Double = 1000.0 'サンプリングレート = 1,000S/sec
Dim nData, iStatus, i, nIndex, nLost As Integer
Dim Data(999) As TWXA_AOxOx_DATA

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)

'オーバーサンプリングレートを 4 に設定
TWXA_ADSetMode(hDev, TWXA_AN_MODE.OSR4)

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

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

'サンプリングデータの読み出し
TWXA_ADReadBuffer(hDev, Data, 1000, nData)

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

'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
nIndex = Data(0).Index + 1
nLost = 0
For i = 1 To nData - 1
  If Data(i).Index <> nIndex Then
    nLost = nLost + nIndex - Data(i).Index
    nIndex = Data(i).Index
  End If
  nIndex = Index + 1
Next

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

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

```
double dRate = 1000.0; //サンプリングレート = 1,000S/sec
int nData, iStatus;
uint i, nIndex, nLost;
TWXA.AOxOx_DATA[] Data = new TWXA.AOxOx_DATA[1000];

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);

//オーバーサンプリングレートを 4 に設定
TWXA.ADSetMode(hDev, TWXA.AN_MODE.OSR4);

//連続サンプリングを開始
TWXA.ADStartAutoSampling(hDev, ref dRate, 0xFFFFFFFF);

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

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

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

//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
nIndex = Data[0].Index + 1;
for (i = 1, nLost = 0; i < nData; i++)
{
  if (Data[i].Index != nIndex)
  {
    nLost += (nIndex - Data[i].Index);
    dwIndex = Data[i].Index;
  }
  nIndex++;
}

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

□ アナログ出力

製品はアナログ出力用に DA0、DA1 端子を備えています。

表 70 はアナログ出力を制御するための関数です。表 71 はアナログ出力のサンプルプログラムです。

表 70　アナログ出力で使用する関数

関数名	説明
TWXA_DARead()	現在のアナログ出力値を読み出します。
TWXA_DAWrite()	アナログ出力値を設定します。
TWXA_AnFromVolt()	電圧値(ボルト単位)から DA コンバータに書き込む値に変換します。
TWXA_DASetConvertMode()	アナログ出力の周期出力設定を行います。
TWXA_DAStart()	アナログ出力の周期出力動作を開始します。
TWXA_DAStop()	アナログ出力の周期出力動作を停止します。
TWXA_PortBWrite()	アナログ出力の周期出力するデータをデバイス上のメモリへ転送します。

表 71　アナログ出力のサンプルプログラム

開発環境	プロジェクト名/ファイル名	説明
Visual C++ (MFC)	AnalogSample	各アナログ出力電圧を変更します。 *TWXA_DAWrite()* を使用したサンプルプログラムです。
Visual Basic	AnalogSampleVB	
Visual C#	AnalogSampleCS	
LabVIEW	AnalogSample.vi	
Visual C++ (MFC)	AnalogAutoSample	アナログ出力の周期動作を利用して、正弦波、矩形波、三角波、のこぎり波を出力します。 *TWXA_DASetConvertMode()* を使用したサンプルプログラムです。
Visual Basic	AnalogAutoSampleVB	
Visual C#	AnalogAutoSampleCS	
LabVIEW	AnalogAutoSample.vi	

アナログ出力値の変更

アナログ出力端子の出力電圧を変更する方法は大きく分けて 2 つの方法があります。

- *TWXA_DAWrite()* 関数を使用した、命令発行毎にアナログ出力値を変更する方法。
- *TWXA_DASetConvertMode()* 関数を使用した、内部タイマに同期して出力値を変更する方法。

表 72 は、それぞれの出力方法の特徴をまとめたものです。

表 72　アナログ出力値の変更方法と特徴

代表関数名	変換レート	特徴
TWXA_DAWrite()	-	アナログ出力端子から出力する電圧値を一度だけ更新します。連続して使用できますが、変換間隔が一定となるとは限りません。
TWXA_DASetConvertMode()	1～20,000[Hz]	デバイスのメモリに転送されたデータを、指定の変換レートで 1 データずつ順に出力します。データを繰り返し出力することができるので、決まった波形パターンを出力することができます。

命令発行毎にアナログ出力値を変更する

TWXA_DAWrite() 関数(表 73)を使用します。*Ch* 引数には DA のチャンネルを示す定数(表 74)を、Data 引数には DA コンバータへの設定値を入力します。DA 設定値と出力電圧の関係を表 75 に示します。

TWXA_AnFromVolt() 関数を使用すると、電圧値から DA コンバータへの設定値を計算することができます。

表 73　TWXA_DAWrite() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_DAWrite(TW_HANDLE hDev, long Ch, WORD Data)
VB	Function TWXA_DAWrite(ByVal hDev As System.IntPtr, ByVal Ch As Integer, ByVal Data As Short) As Integer
VBA	Function TWXA_DAWrite(ByVal hDev As Long, ByVal Ch As Long, ByVal Data As Integer) As Long
C#	STATUS DAWrite(System.IntPtr hDev, int Ch, ushort Data)

表 74　TWXA_DAWrite() の Ch 引数に指定する値

言語	値	説明
C/C++	TWXA_DA0	DA0 出力を変更します。
C++	TWXA::WPORT::DA0	
VB/VBA	TWXA_WPORT.DA0	
C#	TWXA.WPORT.DA0	
C/C++	TWXA_DA1	DA1 出力を変更します。
C++	TWXA::WPORT::DA1	
VB/VBA	TWXA_WPORT.DA1	
C#	TWXA.WPORT.DA1	

表 75　DA 設定値とアナログ出力電圧の関係

DA 設定値	出力電圧([V])
1023	5-LSB
512	2.5
0	0

・LSB = 5 / 1024 [V]
・表は理論値を示しています。

関数を呼び出して実際にアナログ出力に反映さ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)。一定周期で呼び出した場合でも更新間隔が一定になるとは限りませんので、決まった波形パターンを出力するような用途には向きません。

リスト 15　TWXA_DAWrite()の使用例(C 言語)

```
double dVolt;

//DA0 出力を約 3.5V に設定
dVolt = 3.5;
TWXA_DAWrite(hDev, TWXA_DA0, TWXA_AnFromVolt(&dVolt, 10, 0));
```

リスト 16　TWXA_DAWrite()の使用例(Visual Basic)

```
Dim dVolt As Double

'DA0 出力を約 3.5V に設定
dVolt = 3.5
TWXA_DAWrite(hDev, TWXA_WPORT.DA0, TWXA_AnFromVolt(dVolt, 10))
```

リスト 17　TWXA_DAWrite()の使用例(C#)

```
double dVolt;

//DA0 出力を約 3.5V に設定
dVolt = 3.5;
TWXA.DAWrite(hDev, TWXA.WPORT.DA0, TWXA.AnFromVolt(ref dVolt, 10));
```

内部タイマに同期してアナログ出力値を変更する

アナログ出力を周期的に変更する場合は、*TWXA_DASetConvertMode()* 関数(表 76)を使用します。*Ch* 引数には DA のチャンネルを示す定数(表 74)を入力します。

表 76 TWXA_DASetConvertMode() の関数宣言

言語	関数宣言
C/C++	long TWXA_DASetConvertMode(TW_HANDLE hDev, long Ch, double *pFreq, DWORD Addr, DWORD Cnt, BOOL Flg)
VB	Function TWXA_DASetConvertMode(ByVal hDev As System.IntPtr, ByVal Ch As TWXA_DA_CH, ByRef pFreq As Double, ByVal Addr As Integer, ByVal Cnt As Integer, ByVal Flg As Integer) As Integer
VBA	Function TWXA_DASetConvertMode(ByVal hDev As Long, ByVal Ch As TWXA_DA_CH, ByRef pFreq As Double, ByVal Addr As Long, ByVal Cnt As Long, ByVal Flg As Long) As Long
C#	STATUS DASetConvertMode(System.IntPtr hDev, DA_CH Ch, ref double pFreq, Uint Addr, uint Cnt, int Flg)

pFreq 引数にはアナログ出力の更新レートを Hz 単位で入力します。更新レートは内部クロックを分周して生成されるため、実際に設定できる更新レートは離散的になります。*TWXA_DASetConvertMode()* 関数は *pFreq* 引数の入力値と近い値に調整し、実際に設定できた値を pFreq 引数に出力して返ります。設定可能な更新レートは 1～20,000Hz です。

Cnt 引数にはメモリに書き込んだ出力データの数を入力します。*Flg* 引数に 0 以外を入力すると、*TWXA_DAStop()* 関数が呼び出されるまで、繰り返しアナログ出力の更新を行います。

Addr 引数には、実際に出力するデータの格納先先頭アドレスを指定します。

実際に出力するデータは *TWXA_PortBWrite()* 関数でユーザーメモリへ書き込みを行います。データを書き込む際は、1 データを 2 バイトの符号付き整数とし、ビッグエンディアンで構成してください。*TWXA_PortBWrite()* 関数の使用方法、および、ユーザーメモリのアドレス範囲は 100 ページを参照してください。

アナログ出力の周期出力を開始する

アナログ出力の周期出力を開始するには *TWXA_DAStart()* 関数(表 77)を使用します。*Ch* 引数には DA のチャンネルを示す定数(表 74)を入力します。

表 77 TWXA_DAStart() の関数宣言

言語	関数宣言
C/C++	long TWXA_DAStart(TW_HANDLE hDev, long Ch)
VB	Function TWXA_DAStart(ByVal hDev As System.IntPtr, ByVal Ch As TWXA_DA_CH) As Integer
VBA	Function TWXA_DAStart(ByVal hDev As Long, ByVal Ch As TWXA_DA_CH) As Long
C#	STATUS DAStart(System.IntPtr hDev, DA_CH Ch)

アナログ出力の周期出力を停止する

アナログ出力の周期出力を停止するには *TWXA_DAStop()* 関数(表 78)を使用します。*Ch*引数にはDAのチャンネルを示す定数(表 74)を入力します。

表 78　TWXA_DAStop() の関数宣言

言語	関数宣言
C/C++	long TWXA_DAStart(TW_HANDLE hDev, long Ch)
VB	Function TWXA_DAStart(ByVal hDev As System.IntPtr, ByVal Ch As TWXA_DA_CH) As Integer
VBA	Function TWXA_DAStart(ByVal hDev As Long, ByVal Ch As TWXA_DA_CH) As Long
C#	STATUS DAStart(System.IntPtr hDev, DA_CH Ch)

リスト 18　TWXA_DASetConvertMode()の使用例(C 言語)

```
short sData[100]; //波形データ
double dFreq;     //更新レート

//更新レートを設定
dFreq = 20000.0;

//波形データをユーザーメモリへ書き込み
TWXA_PortBWrite(hDev, 0x10000, sData, 200);

//DA0 出力の周期出力を設定(繰り返しは行わない)
TWXA_DASetConvertMode(hDev, TWXA_DA0, &dFreq, 0x10000, 100, 0);

//周期出力を開始
TWXA_DAStart(hDev, TWXA_DA0);

...

//周期出力を停止
TWXA_DAStop(hDev, TWXA_DA0);

//DA0 出力を 0V に初期化
TWXA_DAWrite(hDev, TWXA_DA0, 0);
```

リスト 19　TWXA_DASetConvertMode()の使用例(Visual Basic)

```
Dim sData(99) As Short '波形データ
Dim dFreq As Double    '更新レート

'更新レートを設定
dFreq = 20000.0

'波形データをユーザーメモリへ書き込み
TWXA_PortBWrite(hDev, &H10000, sData, 200);

'DA0出力の周期出力を設定(繰り返しは行わない)
TWXA_DASetConvertMode(hDev, TWXA_DA_CH.DA0, dFreq, &H10000, 100, 0)

'周期出力を開始
TWXA_DAStart(hDev, TWXA_DA_CH.DA0)

...

'周期出力を停止
TWXA_DAStop(hDev, TWXA_DA_CH.DA0)

'DA0出力を0Vに初期化
TWXA_DAWrite(hDev, TWXA_DA_CH.DA0, 0)
```

リスト 20　TWXA_DASetConvertMode()の使用例(C#)

```
short[] sData = new short[100]; //波形データ
double dFreq;                   //更新レート

//更新レートを設定
dFreq = 20000.0;

//波形データをユーザーメモリへ書き込み
TWXA.PortBWrite(hDev, 0x10000, sData, 200);

//DA0出力の周期出力を設定(繰り返しは行わない)
TWXA.DASetConvertMode(hDev, TWXA.DA_CH.DA0, ref dFreq, 0x10000, 100, 0);

//周期出力を開始
TWXA.DAStart(hDev, TWXA.DA_CH.DA0);

...

//周期出力を停止
TWXA.DAStop(hDev, TWXA.DA_CH.DA0);

//DA0出力を0Vに初期化
TWXA.DAWrite(hDev, TWXA.DA_CH.DA0, 0);
```

□ パルスをカウントする

製品ではハードウェアカウンタとソフトウェアカウンタの 2 種類の方法でパルスをカウントすることができます。

ハードウェアカウンタは製品に搭載されるマイコンの 16 ビットタイマというハードウェア機能を利用したもので、名前の通り 16 ビットのカウンタレジスタで入力パルスをカウントすることができます。単相カウント、2 相カウントのどちらにも利用でき、単相パルスカウントの場合最大 8 チャンネル、2 相パルスカウントの場合は最大 2 チャンネル利用できます。ハードウェアを利用するため高速なパルス信号に対応できる特徴があります。

ソフトウェア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。単相カウント、2 相カウント、3 相カウントを利用でき、単相パルスカウントの場合最大 8 チャンネル、2 相パルスカウントの場合は最大 4 チャンネル、3 相パルスカウントの場合は最大 2 チャンネル利用可能です。ソフトウェアでカウントを行い、接続されるフォトカプラの反応速度も遅いため、高速な信号には対応できませんが、反面ノイズに強いという特徴があります。

表 79　ハードウェアカウンタとソフトウェアカウンタの特徴

カウンタ種類	チャンネル数(最大)			カウンタビット数	特徴	備考
	単相	2 相	3 相			
ハードウェアカウンタ	8	2	-	16	高速	マイコンのハードウェア機能(16 ビットタイマ)を利用
ソフトウェアカウンタ	8	4	2	32	ノイズに強い、オーバーフローしにくい	マイコンの外部割り込みをソフトウェアでカウント

表 80　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/ファイル名	説明
Visual C++ (MFC)	SoftwareCounterSample	ソフトウェア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	SoftwareCounterSampleVB	
Visual C#	SoftwareCounterSampleCS	
LabVIEW	SoftwareCounterSample.vi	
VBA (Excel)	SoftwareCounterSample.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。
Visual C++ (MFC)	HardwareCounterSample	ハードウェア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	HardwareCounterSampleVB	
Visual C#	HardwareCounterSampleCS	
LabVIEW	HardwareCounterSample.vi	
VBA (Excel)	HardwareCounterSample.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。

- AD トリガ入力端子として[IN07/SC7/ADTRG]を使用している場合、ソフトウェアカウンタ CH7 による単相カウント/2 相カウント/3 相カウントとして使用できません。
- AD クロック入力端子として[IN27/HC7/ADCLK]を使用している場合、ハードウェアカウンタ CH7 による単相カウント/2 相カウントとして使用できません。
- PWM 出力として使用しているチャンネルはハードウェアカウンタとして使用できません。

ハードウェアカウンタによる単相パルスカウント

ハードウェアカウンタによる単相カウントの場合、HC0～HC7 端子への入力を 16 ビットタイマのチャンネル 0～7 でそれぞれカウントします。カウントエッジは"OFF→ON"、"ON→OFF"、"両エッジ"から選択可能です。

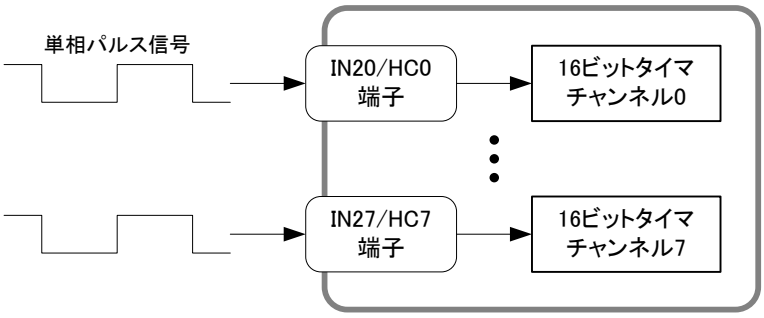


図 48　ハードウェアカウンタによる単相パルスカウント

ハードウェアカウンタによる 2 相パルスカウント

ハードウェアカウンタにより 90° 位相差 2 相パルスをカウントする場合、デジタル入力端子の HC4 と HC5、HC6 と HC7 の組み合わせで使用します。

接続は、チャンネル 4 を使用する場合、HC4 に A 相信号を HC5 に B 相信号を入力します。またチャンネル 6 を使用する場合、HC6 に A 相信号 HC7 に B 相信号を入力します。一般的なインクリメンタル方式のロータリーエンコーダをこのように接続すると CW 回転でカウンタが増加、CCW 回転でカウンタが減少します。また、1 回転あたりのカウント数はロータリーエンコーダの出力パルス数の 4 倍となります(図 50)。

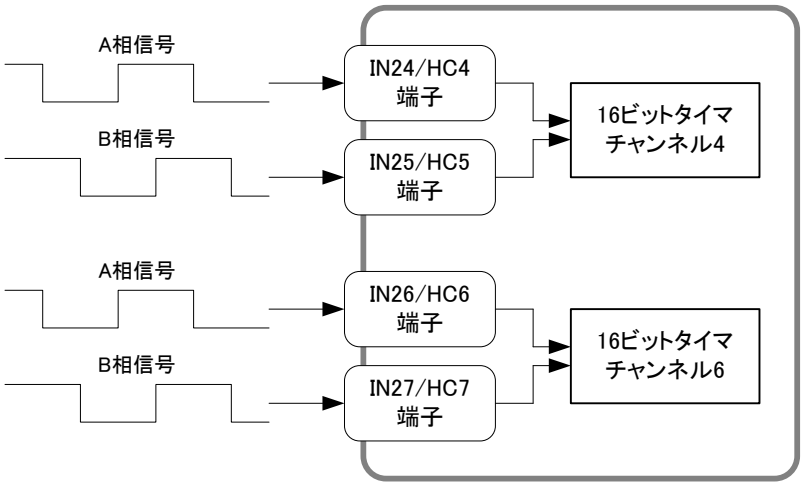


図 49　ハードウェアカウンタによる 2 相パルスカウント

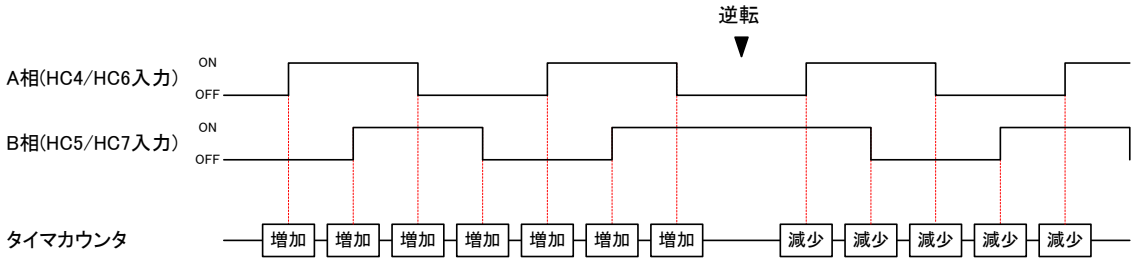


図 50　2 相パルス入力とハードウェアカウンタの増減

ハードウェアカウンタの使用方法

ハードウェアカウンタを使用するには、まず *TWXA_TimerSetMode()* 関数を呼び出し、*Mode* 引数(表 84 参照)によって使用するチャンネルのカウントモードを設定します。

TWXA_TimerStart() 関数でカウントを開始し、*TWXA_TimerReadCnt()* 関数でカウント値を読み出します。

表 81　ハードウェアカウンタで使用する関数

関数名	説明
TWXA_TimerSetMode()	カウントモードを設定します。
TWXA_TimerStart()	カウントを開始します。
TWXA_TimerStop()	カウントを停止します。
TWXA_TimerReadCnt()	カウント値を読み出します。
TWXA_TimerSetCnt()	カウント値をセットします。カウント値のクリアなどに使用します。

表 82　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 83　ハードウェアカウンタ操作関数の Ch 引数に指定する値

言語	値	説明
C/C++	TWXA_TIMER_BIT0	チャンネル 0 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER0	
VB/VBA	TWXA_TIMER_BITS.TIMER0	
C#	TWXA.TIMER_BITS.TIMER0	
C/C++	TWXA_TIMER_BIT1	チャンネル 1 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER1	
VB/VBA	TWXA_TIMER_BITS.TIMER1	
C#	TWXA.TIMER_BITS.TIMER1	
C/C++	TWXA_TIMER_BIT2	チャンネル 2 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER2	
VB/VBA	TWXA_TIMER_BITS.TIMER2	
C#	TWXA.TIMER_BITS.TIMER2	
C/C++	TWXA_TIMER_BIT3	チャンネル 3 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER3	
VB/VBA	TWXA_TIMER_BITS.TIMER3	
C#	TWXA.TIMER_BITS.TIMER3	
C/C++	TWXA_TIMER_BIT4	チャンネル 4 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER4	
VB/VBA	TWXA_TIMER_BITS.TIMER4	
C#	TWXA.TIMER_BITS.TIMER4	
C/C++	TWXA_TIMER_BIT5	チャンネル 5 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER5	
VB/VBA	TWXA_TIMER_BITS.TIMER5	
C#	TWXA.TIMER_BITS.TIMER5	
C/C++	TWXA_TIMER_BIT6	チャンネル 6 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER6	
VB/VBA	TWXA_TIMER_BITS.TIMER6	
C#	TWXA.TIMER_BITS.TIMER6	
C/C++	TWXA_TIMER_BIT7	チャンネル 7 の設定や読み出しなどで指定します。
C++	TWXA::TIMER_BITS::TIMER7	
VB/VBA	TWXA_TIMER_BITS.TIMER7	
C#	TWXA.TIMER_BITS.TIMER7	
C/C++	TWXA_TIMER_BITS_ALL	全てのチャンネルを同じ動作設定にする場合や、全てのカウント値を読み出す場合に指定します。
C++	TWXA::TIMER_BITS::TIMER_ALL	
VB/VBA	TWXA_TIMER_BITS.TIMER_ALL	
C#	TWXA.TIMER_BITS.TIMER_ALL	

表 84　ハードウェアカウンタ使用時に Mode 引数に指定する値

言語	値	説明
C/C++	TWXA_TIMER_DISABLE	カウンタ機能を解除します。指定チャンネルのハードウェアカウンタに使用されている端子は、デジタル入力端子として使用可能になります。全てのチャンネルで指定可能です。
C++	TWXA::TIMER_MODE::DISABLE	
VB/VBA	TWXA_TIMER_MODE.DISABLE	
C#	TWXA.TIMER_MODE.DISABLE	
C/C++	TWXA_TIMER_OFF_TO_ON	指定チャンネルをパルスカウントモードとし、対応する入力が OFF から ON に変化したときカウントします。全てのチャンネルで指定可能です。
C++	TWXA::TIMER_MODE::COUNT_OFF_TO_ON	
VB/VBA	TWXA_TIMER_MODE.COUNT_OFF_TO_ON	
C#	TWXA.TIMER_MODE.COUNT_OFF_TO_ON	
C/C++	TWXA_TIMER_ON_TO_OFF	指定チャンネルをパルスカウントモードとし、対応する入力が ON から OFF に変化したときカウントします。全てのチャンネルで指定可能です。
C++	TWXA::TIMER_MODE::COUNT_ON_TO_OFF	
VB/VBA	TWXA_TIMER_MODE.COUNT_ON_TO_OFF	
C#	TWXA.TIMER_MODE.COUNT_ON_TO_OFF	
C/C++	TWXA_TIMER_BOTH	指定チャンネルをパルスカウントモードとし、極性によらず対応する入力が変化したときにカウントします。全てのチャンネルで指定可能です。
C++	TWXA::TIMER_MODE::COUNT_BOTH	
VB/VBA	TWXA_TIMER_MODE.COUNT_BOTH	
C#	TWXA.TIMER_MODE.COUNT_BOTH	
C/C++	TWXA_TIMER_2PHASE	90° 位相差の A 相、B 相の 2 相信号をカウントします。チャンネル 4、および、チャンネル 6 で指定可能です。
C++	TWXA::TIMER_MODE::COUNT_2PHASE	
VB/VBA	TWXA_TIMER_MODE.COUNT_2PHASE	
C#	TWXA.TIMER_MODE.COUNT_2PHASE	

リスト 21　ハードウェアカウンタの使用例(C 言語)

```
WORD wCnt;

//チャンネル 0 で OFF から ON 時にカウント(カウンタは 0 に初期化されます)
TWXA_TimerSetMode(hDev, TWXA_TIMER_BIT0, TWXA_TIMER_OFF_TO_ON);

//タイマ 0 カウントをスタート
TWXA_TimerStart(hDev, TWXA_TIMER_BIT0);

...

//タイマ 0 のカウント値を符号なし整数として読み出し
TWXA_TimerReadCnt(hDev, TWXA_TIMER_BIT0, (short*)&wCnt);
```

リスト 22　ハードウェアカウンタの使用例(Visual Basic)

```
Dim wCnt As System.UInt16

'チャンネル 0 で OFF から ON 時にカウント(カウンタは 0 に初期化されます)
TWXA_TimerSetMode(hDev, TWXA_TIMER_BITS.TIMER0, TWXA_TIMER_MODE.COUNT_OFF_TO_ON)

'タイマ 0 のカウントをスタート
TWXA_TimerStart(hDev, TWXA_TIMER_BITS.TIMER0)

...

'タイマ 0 のカウント値を符号なし整数として読み出し
TWXA_TimerReadCnt(hDev, TWXA_TIMER_BITS.TIMER0, wCnt)
```

リスト 23　ハードウェアカウンタの使用例(VBA)

```
Dim iCnt As Integer
Dim LCnt As Long

'チャンネル 0 で OFF から ON 時にカウント(カウンタは 0 に初期化されます)
TWXA_TimerSetMode hDev, TWXA_TIMER_BITS.TIMER0, TWXA_TIMER_MODE.COUNT_OFF_TO_ON

'タイマ 0 のカウントをスタート
TWXA_TimerStart hDev, TWXA_TIMER_BITS.TIMER0

...

'タイマ 0 のカウント値を読み出し、0～65535 の範囲の値に変換
TWXA_TimerReadCnt hDev, TWXA_TIMER_BITS.TIMER0, iCnt
LCnt = TWXA_ToINT32(iCnt)
```

リスト 24　ハードウェアカウンタの使用例(C#)

```
ushort usCnt;

//チャンネル 0 で OFF から ON 時にカウント(カウンタは 0 に初期化されます)
TWXA.TimerSetMode(hDev, TWXA.TIMER_BITS.TIMER0, TWXA.TIMER_MODE.COUNT_OFF_TO_ON);

//タイマ 0 のカウントをスタート
TWXA.TimerStart(hDev, TWXA.TIMER_BITS.TIMER0);

...

//タイマ 0 のカウント値を符号なし整数として読み出し
TWXA.TimerReadCnt(hDev, TWXA.TIMER_BITS.TIMER0, out usCnt);
```

ソフトウェアカウンタによる単相パルスカウント

ソフトウェアカウンタによる単相カウントの場合、SC0～SC7 端子への入力をそれぞれチャンネル 0～7 でカウントします。カウントエッジは″OFF→ON″、″ON→OFF″、″両エッジ″から選択可能です。

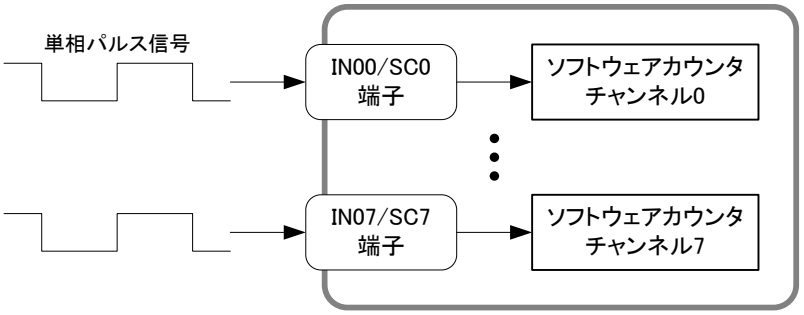


図 51　ソフトウェアカウンタによる単相パルスカウント

ソフトウェアカウンタによる 2 相パルスカウント

ソフトウェアカウンタにより 90° 位相差 2 相パルスをカウントする場合、それぞれチャンネル 0 と 1、チャンネル 2 と 3、チャンネル 4 と 5、および、チャンネル 6 と 7 の組み合わせでカウントします。

チャンネル 0 と 1 の組み合わせでカウントする場合には、SC0 端子に A 相信号を接続し、SC1 端子に B 相信号を接続します。一般的なインクリメンタル方式のロータリーエンコーダをこのように接続すると CW 回転でカウンタが増加、CCW 回転でカウンタが減少します。どちらの組み合わせを使用した場合も、1 回転あたりのカウント数はロータリーエンコーダの出力パルス数の 4 倍になります(図 53)。

2 相パルスカウントの場合、カウント値は組み合わせとなるチャンネルの合計値になります。

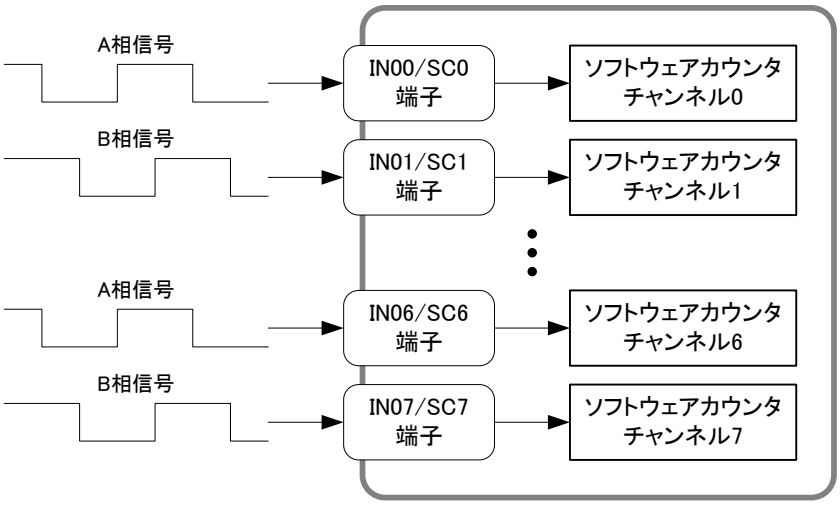


図 52 ソフトウェアカウンタによる 2 相パルスカウント

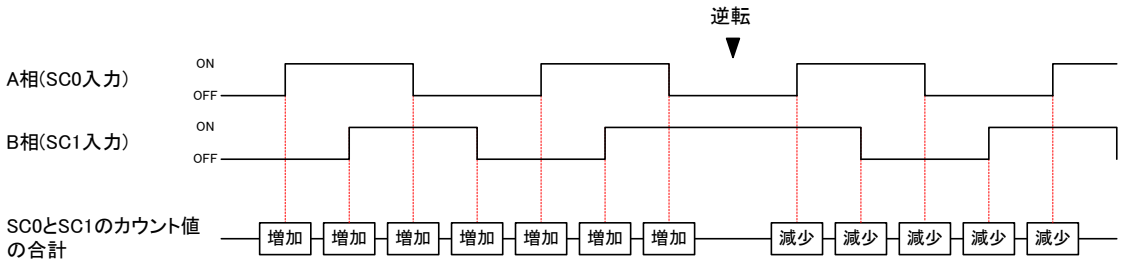


図 53　2 相パルス入力とソフトウェアカウンタの増減

ソフトウェアカウンタによる 3 相パルスカウント

ソフトウェアカウンタにより 3 相パルスをカウントする場合、それぞれチャンネル 0 と 2 と 3、チャンネル 4 と 6 と 7 の組み合わせでカウントします。基本的にはソフトウェアカウンタの 2 相パルスカウントと同様の動作をします。チャンネル 0、2、3 を使用する場合、SC0 端子への入力で、チャンネル 2、3 のカウント値がクリアされ、チャンネル 0 の値が増加します。チャンネル 4、6、7 を使用する場合、SC4 端子への入力で、チャンネル 6、7 のカウント値がクリアされ、チャンネル 4 の値が増加します。

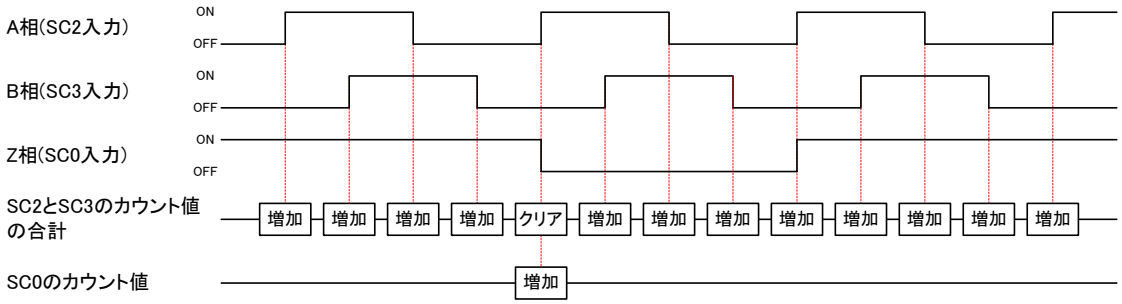


図 54　3 相パルス入力とソフトウェアカウンタのクリア

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を使用するには、まず *TWXA_PCSetMode()* 関数(表 86)を呼び出し、使用するチャンネルの動作モードを設定します。*ChBits* 引数には表 87 に示すチャンネルを示す定数を指定します。*Mode* 引数には表 88 に示すカウントモードを指定します。

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 89)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 0～7 チャンネルまで全てのカウント値を読み出すことができます。その場合は、*pCnt* 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。

表 85　ソフトウェアカウンタで使用する関数

関数名	説明
TWXA_PCSetMode()	カウントモードを設定します。
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()	カウント値を読み出します。
TWXA_PCSetCnt()	カウント値をセットします。主にカウンタクリアに使用します。

表 86　TWXA_PCSetMode() の関数宣言

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

表 87　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	チャンネル 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	チャンネル 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	チャンネル 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	チャンネル 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_PC4	チャンネル 4 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC4	
VB/VBA	TWXA_PC_BITS.PC4	
C#	TWXA.PC_BITS.PC4	
C/C++	TWXA_PC5	チャンネル 5 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC5	
VB/VBA	TWXA_PC_BITS.PC5	
C#	TWXA.PC_BITS.PC5	
C/C++	TWXA_PC6	チャンネル 6 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC6	
VB/VBA	TWXA_PC_BITS.PC6	
C#	TWXA.PC_BITS.PC6	
C/C++	TWXA_PC7	チャンネル 7 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC7	
VB/VBA	TWXA_PC_BITS.PC7	
C#	TWXA.PC_BITS.PC7	

言語	値	説明
C/C++	TWXA_PC0_PC1	チャンネル 0 と 1 の設定や読み出しなどで指定します。 読み出しの場合はチャンネル0と1の合計値が返ります。
C++	TWXA::PC_BITS::PC0_PC1	
VB/VBA	TWXA_PC_BITS.PC0_PC1	
C#	TWXA.PC_BITS.PC0_PC1	
C/C++	TWXA_PC2_PC3	チャンネル 2 と 3 の設定や読み出しなどで指定します。 読み出しの場合はチャンネル2と3の合計値が返ります。
C++	TWXA::PC_BITS::PC2_PC3	
VB/VBA	TWXA_PC_BITS.PC2_PC3	
C#	TWXA.PC_BITS.PC2_PC3	
C/C++	TWXA_PC4_PC5	チャンネル 4 と 5 の設定や読み出しなどで指定します。 読み出しの場合はチャンネル4と5の合計値が返ります。
C++	TWXA::PC_BITS::PC4_PC5	
VB/VBA	TWXA_PC_BITS.PC4_PC5	
C#	TWXA.PC_BITS.PC4_PC5	
C/C++	TWXA_PC6_PC7	チャンネル 6 と 7 の設定や読み出しなどで指定します。 読み出しの場合はチャンネル6と7の合計値が返ります。
C++	TWXA::PC_BITS::PC6_PC7	
VB/VBA	TWXA_PC_BITS.PC6_PC7	
C#	TWXA.PC_BITS.PC6_PC7	
C/C++	TWXA_PC0_PC2_PC3	チャンネル0と2と3の設定で指定します。
C++	TWXA::PC_BITS::PC0_PC2_PC3	
VB/VBA	TWXA_PC_BITS.PC0_PC2_PC3	
C#	TWXA.PC_BITS.PC0_PC2_PC3	
C/C++	TWXA_PC4_PC6_PC7	チャンネル4と6と7の設定で指定します。
C++	TWXA::PC_BITS::PC4_PC6_PC7	
VB/VBA	TWXA_PC_BITS.PC4_PC6_PC7	
C#	TWXA.PC_BITS.PC4_PC6_PC7	
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、全てのカウント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	

表 88　TWXA_PCSetMode() の Mode 引数に指定する値

言語	値	説明
C/C++	TWXA_PC_OFF_TO_ON	入力がOFFからONに変化したときにカウントします。
C++	TWXA::PC_MODE::COUNT_OFF_TO_ON	
VB/VBA	TWXA_PC_MODE.COUNT_OFF_TO_ON	
C#	TWXA.PC_MODE.COUNT_OFF_TO_ON	
C/C++	TWXA_PC_ON_TO_OFF	入力がONからOFFに変化したときにカウントします。
C++	TWXA::PC_MODE::COUNT_ON_TO_OFF	
VB/VBA	TWXA_PC_MODE.COUNT_ON_TO_OFF	
C#	TWXA.PC_MODE.COUNT_ON_TO_OFF	
C/C++	TWXA_PC_BOTH	入力が変化したときにカウントします。
C++	TWXA::PC_MODE::COUNT_BOTH	
VB/VBA	TWXA_PC_MODE.COUNT_BOTH	
C#	TWXA.PC_MODE.COUNT_BOTH	
C/C++	TWXA_PC_2PHASE	90° 位相差のA相、B相の2相信号をカウントするモードです。
C++	TWXA::PC_MODE::COUNT_2PHASE	
VB/VBA	TWXA_PC_MODE.COUNT_2PHASE	
C#	TWXA.PC_MODE.COUNT_2PHASE	
C/C++	TWXA_PC_3PHASE	A相とB相で2相信号のカウントを行い、Z相への入力でカウンタをクリアします。
C++	TWXA::PC_MODE::COUNT_3PHASE	
VB/VBA	TWXA_PC_MODE.COUNT_3PHASE	
C#	TWXA.PC_MODE.COUNT_3PHASE	

表 89　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)

リスト 25　ソフトウェアカウンタによるパルスカウントの例(C 言語)

```
long LCnt[8];
long L2Phase23;
long L2Phase67;

//ソフトウェアカウンタ 0,1 を単相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC0_PC1, TWXA_PC_OFF_TO_ON);

//ソフトウェアカウンタ 2,3 を 2 相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC2_PC3, TWXA_PC_2PHASE);

//ソフトウェアカウンタ 4,5 を単相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC4_PC5, TWXA_PC_BOTH);

//ソフトウェアカウンタ 6,7 を 2 相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC6_PC7, TWXA_PC_2PHASE);

//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ト値を読み出し
TWXA_PCReadCnt(hDev, TWXA_PC_ALL, LCnt);

//2 相カウントの結果は 2 チャンネルのカウント値の和
L2Phase23 = LCnt[2] + LCnt[3];
L2Phase67 = LCnt[6] + LCnt[7];
```

リスト 26　ソフトウェアカウンタによるパルスカウントの例(Visual Basic)

```
Dim LCnt(7) As Integer
Dim L2Phase23 As Integer
Dim L2Phase67 As Integer

'ソフトウェアカウンタ 0,1 を単相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC_BITS.PC0_PC1, TWXA_PC_MODE.COUNT_OFF_TO_ON)

'ソフトウェアカウンタ 2,3 を 2 相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC_BITS.PC2_PC3, TWXA_PC_MODE.COUNT_2PHASE)

'ソフトウェアカウンタ 4,5 を単相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC_BITS.PC4_PC5, TWXA_PC_MODE.COUNT_BOTH)

'ソフトウェアカウンタ 6,7 を 2 相カウントに設定(カウンタは 0 に初期化されます)
TWXA_PCSetMode(hDev, TWXA_PC_BITS.PC6_PC7, TWXA_PC_MODE.COUNT_2PHASE)

'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ト値を読み出し
TWXA_PCReadCnt(hDev, TWXA_PC_BITS.PC_ALL, LCnt(0))

'2 相カウントの結果は 2 チャンネルのカウント値の和
L2Phase23 = LCnt(2) + LCnt(3)
L2Phase67 = LCnt(6) + LCnt(7)
```

リスト 27　ソフトウェアカウンタによるパルスカウントの例(C#)

```
int[] iCnt = new int[4];
int i2Phase23;
int i2Phase67;

//ソフトウェアカウンタ 0,1 を単相カウントに設定(カウンタは 0 に初期化されます)
TWXA.PCSetMode(hDev, TWXA.PC_BITS.PC0_PC1, TWXA.PC_MODE.COUNT_OFF_TO_ON);

//ソフトウェアカウンタ 2,3 を 2 相カウントに設定(カウンタは 0 に初期化されます)
TWXA.PCSetMode(hDev, TWXA.PC_BITS.PC2_PC3, TWXA.PC_MODE.COUNT_2PHASE);

//ソフトウェアカウンタ 4,5 を単相カウントに設定(カウンタは 0 に初期化されます)
TWXA.PCSetMode(hDev, TWXA.PC_BITS.PC4_PC5, TWXA.PC_MODE.COUNT_BOTH);

//ソフトウェアカウンタ 6,7 を 2 相カウントに設定(カウンタは 0 に初期化されます)
TWXA.PCSetMode(hDev, TWXA.PC_BITS.PC6_PC7, TWXA.PC_MODE.COUNT_2PHASE);

//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ト値を読み出し
TWXA.PCReadCnt(hDev, TWXA.PC_BITS.PC_ALL, iCnt);

//2 相カウントの結果は 2 チャンネルのカウント値の和
i2Phase23 = iCnt[2] + iCnt[3];
i2Phase67 = iCnt[6] + iCnt[7];
```

□ PWM 出力

製品では最大 5 チャンネルの PWM 出力が可能です。PWM の基準クロックは 48MHz の内部クロックをプリスケーラ [15]で分周した後、そのクロックのカウントに 16 ビットタイマを使用することで生成することができます。この場合、出力できる周波数範囲はチャンネル 2、3、4、7 では 1Hz～1MHz、チャンネル 6 では 3Hz～1MHz までの範囲です。

PWM の基準クロックは外部から HC 端子に入力することもできます。より周期の長い信号が必要な場合や、外部クロックとの同期が必要な場合に利用します。外部クロックとして別チャンネルの PWM 出力を利用することもできます(図 55)。外部クロックを使用する場合、外部クロックの入力として使用する HC 端子の番号と、PWM 出力するチャンネル番号を一致させてください。例えば、PWM7 の外部クロックは HC7 端子へ入力します(表 90)。

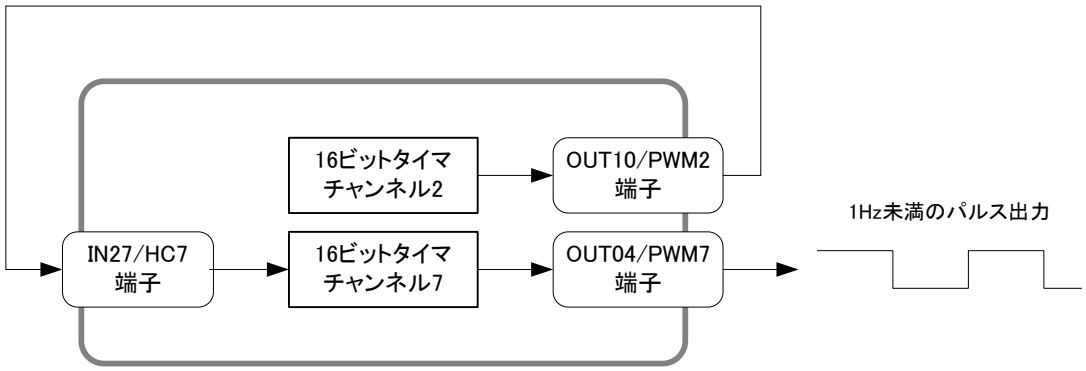


図 55　他チャンネルの出力を基準クロックとして利用

表 90　PWM 出力端子と外部クロック入力端子の対応

チャンネル	PWM 出力端子	外部クロック入力端子
2	PWM2	HC2
3	PWM3	HC3
4	PWM4	HC4
6	PWM6	HC6
7	PWM7	HC7

- AD クロック入力端子として[IN27/HC7/ADCLK]を使用している場合、CH7 は外部クロック入力による PWM 出力は使用できません。
- ハードウェアカウンタとして使用しているチャンネルは PWM 出力として使用できません。

[15] 48MHz のクロックを 1/1、1/4、1/16、1/64 などに分周します。

表 91　PWM 出力で使用する関数

関数名	説明
TWXA_TimerSetMode()	16 ビットタイマに対して PWM モードの設定、または、解除を行います。
TWXA_TimerSetPwm()	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_TimerSetPwmExt()	外部クロックを用いた場合のパルス設定を行います。
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。

表 92　PWM 出力のサンプルプログラム

開発環境	プロジェクト名/ファイル名	説明
Visual C++ (MFC)	PwmSample	5 チャンネルの周波数、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	
LabVIEW	PwmSample.vi	
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

内部クロックを使用したパルスの設定には *TWXA_TimerSetPwm()* 関数(表 93)、外部クロックを使用したパルスの設定には *TWXA_TimerSetPwmExt()* 関数(表 94)を使用します。

表 93　TWXA_TimerSetPwm() の関数宣言

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

表 94　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)

*pFrequency*引数にはパルスの周波数をHz単位で入力します。*pDuty*引数にはONデューティを0～1.0 の範囲で入力します。*pPhase* 引数には出力開始時の位相を 0～1.0 の範囲で入力します。*TWXA_TimerSetPwmExt()* 関数の *dClkFreq* 引数は入力する外部クロックの周波数を Hz 単位で設定してください。

各引数と出力パルスの関係を図 56 に示します。

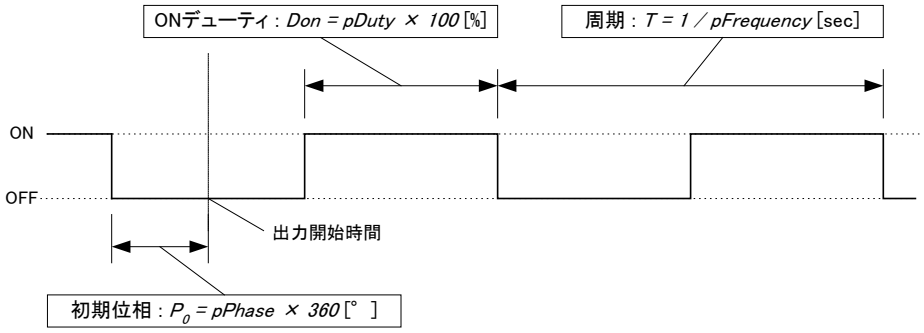


図 56　パラメータと出力パルスの関係

パルスのタイミングは基準クロック(48MHz の内部クロック、または、外部クロック)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_TimerSetPwm()* 、*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。

PWM 出力の手順

1. *TWXA_TimerSetMode()* 関数(72 ページ、表 82)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 95 を参照してください。

表 95 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。PWM 端子がデジタル出力端子として使用可能になります。
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。対応する端子は PWM 出力用となります。
C++	TWXA::TIMER_MODE::PWM	
VB/VBA	TWXA_TIMER_MODE.PWM	
C#	TWXA.TIMER_MODE.PWM	

2. *TWXA_TimerSetPwm()* または *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_TimerSetNumOfPulse()* 関数で出力パルス数を設定します。
4. *TWXA_TimerStart()* 関数でパルス出力を開始します。
5. パルス出力中も *TWXA_TimerSetPwm()* 関数等で周波数とデューティを変更することが可能です。
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 OUT0、または、OUT1 の該当する端子に 0 を書き込みます(44 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_TimerStop()* 関数で出力動作を停止した後に、*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。

- 動作中に出力周波数の変更を行うと、パルスタイミングの誤差が生じる場合があります。

リスト 28　PWM 出力の例(C 言語)

```
double dFreq = 9500; //周波数 = 9.5kHz
double dDuty = 0.6;  //デューティ = 60%
double dPhase = 0.0;
TCHAR c[256];

//タイマ 2 を PWM に設定
TWXA_TimerSetMode(hDev, TWXA_TIMER_BIT2, TWXA_TIMER_PWM);

//パルス設定
TWXA_TimerSetPwm(hDev, TWXA_TIMER_BIT2, &dFreq, &dDuty, &dPhase);

//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, TWXA_TIMER_BIT2, 100);

//出力開始
TWXA_TimerStart(hDev, TWXA_TIMER_BIT2);
```

リスト 29　PWM 出力の例(Visual Basic)

```
Dim dFreq As Double = 9500 '周波数 = 9.5kHz
Dim dDuty As Double = 0.6  'デューティ = 60%
Dim dPhase As Double = 0.0

'タイマ 2 を PWM に設定
TWXA_TimerSetMode(hDev, TWXA_TIMER_BITS.TIMER2, TWXA_TIMER_MODE.PWM)

'パルス設定
TWXA_TimerSetPwm(hDev, TWXA_TIMER_BITS.TIMER2, dFreq, dDuty, 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, TWXA_TIMER_BITS.TIMER2, 100)

'出力開始
TWXA_TimerStart(hDev, TWXA_TIMER_BITS.TIMER2)
```

リスト 30　PWM 出力の例(C#)

```
double dFreq = 9500; //周波数 = 9.5kHz
double dDuty = 0.6;  //デューティ = 60%
double dPhase = 0;

//タイマ 2 を PWM に設定
TWXA.TimerSetMode(hDev, TWXA.TIMER_MODE.TIMER2, TWXA.TIMER_MODE.PWM);

//パルス設定
TWXA.TimerSetPwm(hDev, TWXA.TIMER_MODE.TIMER2, 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, TWXA.TIMER_MODE.TIMER2, 100);

//出力開始
TWXA.TimerStart(hDev, TWXA.TIMER_BITS.TIMER2);
```

□ 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。通信方式は調歩同期のみです。通信速度は300bps～256000bps でフロー制御はありません。受信バッファは511 バイトで、オーバーフローすると古いデータから破棄されます。

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 96 にシリアルポート制御で使用する関数をあげます。

表 96　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 97　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/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	
LabVIEW	SerialSample.vi	

シリアルポートの設定

表 98 は *TWXA_SCISetMode()* 関数の宣言です。*Mode* 引数には表 99 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 100 のボーレートを入力します。

表 98　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 99　TWXA_SCISetMode() の Mode 引数に指定する値

設定項目	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 100　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	

言語	値	説明
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	
C/C++	TWXA_SCI_BAUD56000	ボーレートを 56000bps にします。
C++	TWXA::SCI_BAUD::BAUD56000	
VB/VBA	TWXA_SCI_BAUD.BAUD56000	
C#	TWXA.SCI_BAUD.BAUD56000	
C/C++	TWXA_SCI_BAUD57600	ボーレートを 57600bps にします。
C++	TWXA::SCI_BAUD::BAUD57600	
VB/VBA	TWXA_SCI_BAUD.BAUD57600	
C#	TWXA.SCI_BAUD.BAUD57600	
C/C++	TWXA_SCI_BAUD115200	ボーレートを 115200bps にします。
C++	TWXA::SCI_BAUD::BAUD115200	
VB/VBA	TWXA_SCI_BAUD.BAUD115200	
C#	TWXA.SCI_BAUD.BAUD115200	
C/C++	TWXA_SCI_BAUD128000	ボーレートを 128000bps にします。
C++	TWXA::SCI_BAUD::BAUD128000	
VB/VBA	TWXA_SCI_BAUD.BAUD128000	
C#	TWXA.SCI_BAUD.BAUD128000	
C/C++	TWXA_SCI_BAUD256000	ボーレートを 256000bps にします。
C++	TWXA::SCI_BAUD::BAUD256000	
VB/VBA	TWXA_SCI_BAUD.BAUD256000	
C#	TWXA.SCI_BAUD.BAUD256000	

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. デバイスの受信バッファ内の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. 受信バッファ内のデータを読み出すには *TWXA_SCIRead()* 関数を使用します。

リスト 31　シリアルポートの使用例(C 言語)

```
char cRecv[511];
char cSend[] = "Hello¥r¥nWorld¥r¥n"; //送信文字列
long L;

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

//シリアル 0 のデリミタを CR+LF に設定
TWXA_SCISetDelimiter(hDev, 0, "¥r¥n", 2);

//シリアル 0 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, 0, NULL, &L);
  if (L == 0) break;

  //受信データを読み出す
  TWXA_SCIRead(hDev, 0, cRecv, L, NULL);
  OutputDebugStringA(cRecv);
}
```

リスト 32　シリアルポートの使用例(Visual Basic)

```
Dim str As String
Dim bBuff(510) As Byte
Dim i As Integer

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

'シリアル 0 を設定(Mode はデフォルト設定)
TWXA_SCISetMode(hDev, 0, 0, TWXA_SCI_BAUD.BAUD9600)

'シリアル 0 のデリミタを CR+LF に設定
TWXA_SCISetDelimiter(hDev, 0, 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, 0, 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, 0, bBuff, i, i)
  str = System.Text.Encoding.GetEncoding(932).GetString(bBuff, 0, i)
  Debug.WriteLine(str)
Loop
```

リスト 33　シリアルポートの使用例(VBA)

```
Dim str As String
Dim bBuff(510) 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を設定(Modeはデフォルト設定)
TWXA_SCISetMode hDev, 0, 0, TWXA_SCI_BAUD.BAUD9600

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

'シリアル0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, 0, 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, 0, bBuff(0), L, L
  bBuff(L) = 0

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

リスト 34　シリアルポートの使用例(C#)

```
byte[] bBuff = new byte[511];
string str = "Hello¥r¥nWorld¥r¥n"; //送信文字列
int i;
byte b;

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

//シリアル 0 のデリミタを CR+LF に設定
TWXA.SCISetDelimiter(hDev, 0, "¥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, 0, out b, out i);
  if (i == 0) break;

  //受信データを読み出す
  TWXA.SCIRead(hDev, 0, 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

製品ではソフトウェアカウンタのカウント値やAD変換結果を閾値と比較し、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。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 101 はハードウェアイベントのサンプルプログラムです。

表 101　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/ファイル名	説明
Visual C++ (MFC)	EventSample	ソフトウェアカウンタの値が変化した場合、または、アナログ入力が閾値を超えた場合に画面に表示します。
Visual Basic	EventSampleVB	
Visual C#	EventSampleCS	
LabVIEW	EventSample.vi	

- 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 102 の *TWXA_SetHwEventEx()* 関数を使用します。この関数には引数として *TWXA_HW_EVENT_EXA* 構造体(表 103)を渡します。

表 102　TWXA_SetHwEventEx() の関数宣言

言語	関数宣言
C/C++	TW_STATUS TWXA_SetHwEventEx(TW_HANDLE hDev, void *pHwEventEx)
VB	Function TWXA_SetHwEventEx(ByVal hDev As System.IntPtr, ByVal pHwEventEx As Object) As Integer
C#	STATUS SetHwEventEx(System.IntPtr hDev, object pHwEventEx)

表 103　TWXA_HW_EVENT_EXA 構造体の宣言

言語	宣言

C/C++	typedef struct tagHwEventExA
	{
	  HWND hRecvWindow;
	  DWORD idRecvThread;
	  LPVOID lpRsv;
	  UINT Message;
	  DWORD EventBits;
	  long PCCnt[8];
	  long PCCmp[8];
	  long ADVal[4];
	  long ADCmp[4];
	} TWXA_HW_EVENT_EXA;

言語	宣言

VB	<StructLayout(LayoutKind.Sequential)> _
	Public Structure TWXA_HW_EVENT_EXA
	  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:=8)> _
	  Public PCCnt() As Integer
	  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=8)> _
	  Public PCCmp() As Integer
	  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _
	  Public ADVal() As Integer
	  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _
	  Public ADCmp() As Integer
	
	  Public Sub Initialize()
	  ReDim PCCnt(7)
	  ReDim PCCmp(7)
	  ReDim ADVal(3)
	  ReDim ADCmp(3)
	  End Sub
	End Structure

C#	public struct HW_EVENT_EXA
	{
	  public System.IntPtr hRecvWindow;
	  public uint idRecvThread;
	  public System.IntPtr lpRsv;
	  public int Message;
	  public uint EventBits;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
	  public int[] PCCnt;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
	  public int[] PCCmp;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
	  public int[] ADVal;
	  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
	  public int[] ADCmp;
	
	  public void Initialize()
	  {
	  PCCnt = new int[8];
	  PCCmp = new int[8];
	  ADVal = new int[4];
	  ADCmp = new int[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(WM_APP)～H'BFFF の範囲の任意の値を指定します。ソフトウェアカウンタやアナログ入力について条件が成立すると、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
監視するイベントをビットで指定します(表 104)。複数のビットを指定することができます。

PCCnt
ソフトウェア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ソフトウェアカウンタのチャンネルと対応しています。

PCCmp
ソフトウェアカウンタの値と閾値の比較方法を指定します(表 105)。配列のインデックスがソフトウェアカウンタのチャンネルと対応しています。

ADVal
アナログ入力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アナログ入力のチャンネルと対応しています。

ADCmp
アナログ入力の値と閾値の比較方法、ヒステリシスの大きさを指定します。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#では配列の領域確保用に最初に呼び出します。

表 104　EventBits の指定

ビット	監視するイベント
TWXA_EVENT_PC0(H'00000001)	ソフトウェアカウンタ 0 を監視
TWXA_EVENT_PC1(H'00000002)	ソフトウェアカウンタ 1 を監視
TWXA_EVENT_PC2(H'00000004)	ソフトウェアカウンタ 2 を監視
TWXA_EVENT_PC3(H'00000008)	ソフトウェアカウンタ 3 を監視
TWXA_EVENT_PC4(H'00000100)	ソフトウェアカウンタ 4 を監視
TWXA_EVENT_PC5(H'00000200)	ソフトウェアカウンタ 5 を監視
TWXA_EVENT_PC6(H'00000400)	ソフトウェアカウンタ 6 を監視
TWXA_EVENT_PC7(H'00000800)	ソフトウェアカウンタ 7 を監視
TWXA_EVENT_PC_ALL(H'00000F0F)	ソフトウェアカウンタ全チャンネルを監視
TWXA_EVENT_AD0(H'00000010)	アナログ入力 0 を監視
TWXA_EVENT_AD1(H'00000020)	アナログ入力 1 を監視
TWXA_EVENT_AD2(H'00000040)	アナログ入力 2 を監視
TWXA_EVENT_AD3(H'00000080)	アナログ入力 3 を監視
TWXA_EVENT_AD_ALL(H'000000F0)	アナログ入力全チャンネルを監視

ソフトウェアカウンタ入力を監視する

ソフトウェアカウンタによるイベントは、カウント値が予め設定した閾値以上となった場合、閾値以下となった場合、または、カウント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_EXA* 構造体の *Initialize()* メソッドを呼びます。
2. *TWXA_HW_EVENT_EXA* 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_EXA* 構造体の *EventBits* に監視するソフトウェアカウンタチャンネルを指定します(表 104 参照)。
4. *TWXA_HW_EVENT_EXA* 構造体の *PCCnt* にカウント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。例えばソフトウェアカウンタのチャンネル 2 を監視する場合は、*PCCmp[2]* に閾値を設定します。
5. *TWXA_HW_EVENT_EXA* 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 105 に示します。

表 105　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(2,147,483,647)	指定チャンネル(x)のカウント値が PCCnt[x]以上になった場合 (1 度だけ発生)
1～2,147,483,646	指定チャンネル(x)のカウント値が PCCnt[x]以上になった場合 (イベント発生後に PCCnt[x]は PCCnt[x]+PCCmp[x]に更新され、イベントの監視を継続)
TWXA_CMP_NO(0)	指定チャンネル(x)のカウント値が変化した場合
-1～-2,147,483,647	指定チャンネル(x)のカウント値が PCCnt[x]以下になった場合 (イベント発生後に PCCnt[x]は PCCnt[x]+PCCmp[x]に更新され、イベントの監視を継続)
TWXA_CMP_LE(-2,147,483,648)	指定チャンネル(x)のカウント値が PCCnt[x]以下となった場合

6. パラメータを設定した構造体を引数として *TWXA_SetHwEventEx()* 関数を呼び出します。
7. 使用するソフトウェアカウンタの設定を行い、カウント動作を開始します(69 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 106　ソフトウェア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_EXA 構造体の Message に指定した値
wParam(WParam)	イベントが発生したカウンタチャンネルを示すビット(表 104)
lParam(LParam)	イベント発生時のカウンタの値

9. *pHwEventEx* 引数を Null 値とするか、*EventBits* を 0 として *TWXA_SetHwEventEx()* 関数を呼び出すとイベントの監視を終了します。

アナログ入力を監視する

アナログ入力によるイベントは、アナログ入力値が予め設定した閾値以上となった場合、または、閾値以下となった場合に発生させることができます。

また、アナログ入力値が閾値付近にあるとき、不要なイベントが何度も発生するのを防ぐために適当なヒステリシス(V_{HYST})を持たせることができます。例えば、閾値電圧を V_{TH}、ヒステリシス電圧を V_{HYSY} (> 0)に設定した場合、アナログ入力電圧が V_{TH} 以上になることでハードウェアイベントが検出されますが、この時点で該当チャンネルの次のイベント検出は一旦禁止されます。この禁止状態は入力電圧が(V_{TH} 以下ではなく) V_{TH} - V_{HYST} 以下となったときに解除されます(図 57)。

ヒステリシス電圧が適切な大きさに設定されていないと、入力電圧が V_{TH} 付近のとき、ノイズなどによる微小な電圧変化でもハードウェアイベントが検出されてしまい、不要なメッセージが何度も通知される場合があります。

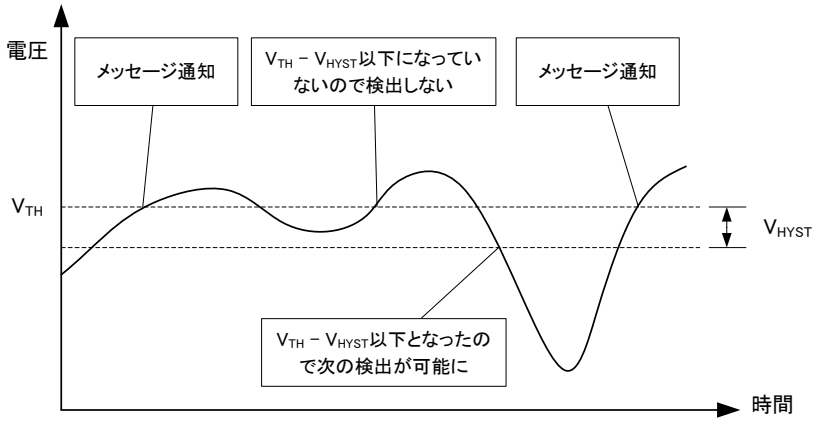


図 57　ヒステリシスが設定されている場合の動作

1. アナログ入力の入力レンジを設定します(46 ページ参照)。
2. Visual Basic または C# を利用する場合、*TWXA_HW_EVENT_EXA* 構造体の *Initialize()* メソッドを呼びます。
3. *TWXA_HW_EVENT_EXA* 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。
4. *TWXA_HW_EVENT_EXA* 構造体の *EventBits* に監視するアナログ入力チャンネルを指定します(表 104 参照)。
5. *TWXA_HW_EVENT_EXA* 構造体の *ADVal* にアナログ入力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。例えばアナログ入力のチャンネル 2 を監視する場合は、*ADVal[2]* に閾値を設定します。閾値電圧 V_{TH} から *ADVal* への設定値 C_{TH} を求めるには *TWXA_AnFromVolt()* 関数を使用するか、下の式を使用します。

   C_{TH} ≒ (V_{TH} [V] / V_{pp} [V]) × 65536

   V_{pp} : 入力レンジが±5[V]の場合 10、入力レンジが±10[V]の場合 20

6. *TWXA_HW_EVENT_EXA* 構造体の *ADCmp* に比較方法とヒステリシス電圧を指定します。表 107 に *ADCmp* に指定する値と、イベント発生条件、再度イベント検出が可能になる条件を示します。ヒステリシス電圧 V_{HYST} から *ADCmp* への設定値 C_{HYST} を求めるには *TWXA_AnFromVolt()* 関数を使用するか、下の式を使用します。

表 107　ADCmp の設定値とハードウェアイベントの発生条件

ADCmp[x]の設定	ハードウェアイベントの発生条件	再度イベント検出可能となる条件
正の場合	指定チャンネル(x)の AD 変換値が ADVal[x]以上になった場合	指定チャンネル(x)の AD 変換値が ADVal[x] - ADCmp[x]以下になった場合
負の場合	指定チャンネル(x)の AD 変換値が ADVal[x]以下になった場合	指定チャンネル(x)の AD 変換値が ADVal[x] - ADCmp[x]以上になった場合

C_{HYST} ≒ (V_{HYST} [V] / V_{pp} [V]) × 65536

V_{HYST}：イベントの発生条件を *ADVal* 以上とする場合、正の値

イベントの発生条件を *ADVal* 以下とする場合、負の値

V_{pp}：入力レンジが±5[V]の場合 10、±10[V]の場合 20

7. パラメータを設定した構造体を引数として *TWXA_SetHwEventEx()* 関数を呼び出すと、指定のアナログ入力チャンネルの監視が開始されます。
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 108　アナログ入力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_EXA 構造体の Message に指定した値
wParam(WParam)	イベントが発生したアナログ入力を示すビット(表 104)
lParam(LParam)	イベントが発生したアナログ入力チャンネルの AD 変換結果

9. *pHwEventEx* 引数を Null 値とするか、*EventBits* を 0 として *TWXA_SetHwEventEx()* 関数を呼び出すとイベントの監視を終了します。

- -32768 ＜ *ADVal[x]* － *ADCmp[x]* ＜ 32767 となるようにしてください。

□ 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 16 バイトの、1 バイトずつ読み書きできるメモリで、R0～R9 および RA～RF という 16 の領域によって構成されています。これらはすべてデバイスの起動時、リセット時、および *TWXA_Initialize()* 関数が呼び出された時には必ず 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。

利用例として、ソフトウェアカウンタやハードウェアカウンタを使ったアプリケーションプログラムで、「プログラム終了後もカウント動作を継続し、再度プログラムを起動した場合にはそのときのカウント値を表示したい」といった場合を考えます。このようなとき、最初にアプリケーションプログラムがカウント動作を設定した時点で、ユーザーステータスレジスタに初期化済みであるフラグを記録しておきます。2 回目以降のアプリケーションプログラムの実行では、フラグを調べてカウンタの初期化が必要無く、単にカウント値を読み出せば良いことがわかります。何らかの理由でデバイスの電源が切れた場合には、ユーザーステータスレジスタ上のフラグがクリアされるので初期化が必要なことがわかります。

ユーザーメモリはデバイスの RAM に確保された 32K 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データを記憶することができます。この領域の値は起動時に不定となり、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。ユーザーメモリの領域はデバイス上の H'0001 0000～H'0001 7FFF の範囲になります。

表 109　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。

表 110　ユーザーステータスレジスタを利用したサンプルプログラム

開発環境	プロジェクト名/ファイル名	説明
Visual C++ (MFC)	SoftwareCounterSample	カウントモードの記録にユーザーステータスレジスタを利用しています。
Visual Basic	SoftwareCounterSampleVB	
Visual C#	SoftwareCounterSampleCS	
LabVIEW	SoftwareCounterSample.vi	
Visual C++ (MFC)	HardwareCounterSample	
Visual Basic	HardwareCounterSampleVB	
Visual C#	HardwareCounterSampleCS	
LabVIEW	HardwareCounterSample.vi	

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 111 の値を指定してください。

表 111　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_Rx	ユーザー用ステータスレジスタ Rx を変更します。 (x = 0～9, A～F)
C++	TWXA::WPORT::USER_STATUS_Rx	
VB/VBA	TWXA_WPORT.USER_STATUS_Rx	
C#	TWXA.WPORT.USER_STATUS_Rx	

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 112　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 113　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。

□ フラッシュメモリの利用

製品にはデータ用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す。図 58 はデータ用フラッシュメモリ領域のユーザーに開放された部分を詳しく示した図です。

データ用フラッシュメモリは消去単位毎にEB1～EB3の3ブロックに分けて管理されます。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校正データの保存などに利用可能です。

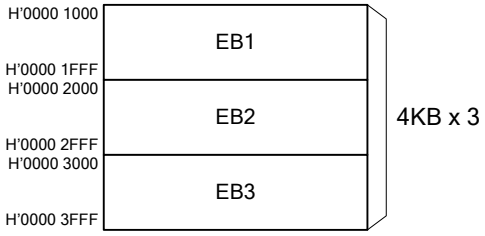


図 58　フラッシュメモリマップ(ユーザー領域)

表 114　フラッシュメモリの操作に使用する関数

関数名	説明
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データを書き込みます。
TWXA_FlashRead()	フラッシュメモリからデータを読み出します。

表 115　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/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	
LabVIEW	FlashSample.vi	
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。

フラッシュメモリの読み出し操作、および、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。

書き込みを行う場合は、対象の領域を予め消去しておく必要があります。消去の単位は図 58 に示したEB1～EB3のブロック単位です。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。

- フラッシュメモリの書換え可能回数の目安は 30,000 回、データ保持年数は 30 年です。
- TWXA ライブラリによるフラッシュメモリ操作は通常モードで行います。

フラッシュメモリからの読み出し方法

TWXA_FlashRead() 関数を呼び出します。*Address* 引数には読み出し先のアドレスとして H'0000 1000～H'0000 3F80 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* 引数に指定する読み出しバイト数も 128 の倍数としてください。

表 116　TWXA_FlashRead() の関数宣言

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

フラッシュメモリの消去方法

TWXA_FlashBlk() 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 117　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

TWXA_FlashWrite() 関数を呼び出します。*Address* 引数には書き込み先のアドレスとして H'0000 1000～H'0000 3F80 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 118　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, long 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 35　フラッシュメモリの使用例(C 言語)

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

//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_FlashRead(hDev, 0x1000, cRead, 128);
OutputDebugStringA(cRead);
```

リスト 36　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

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

リスト 37　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)

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

リスト 38　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;

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

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

//読み出し
TWXA.FlashRead(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 ライブラリ 関数リファレンス」を参照してください。

リスト 39　エラー処理の例(C 言語)

```
TW_STATUS ret;

ret = TWXA_PortWrite(hDev, TWXA_POUT0, 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 40　エラー処理の例(C++/MFC)

```
CString str;
TW_STATUS ret;

ret = TWXA_PortWrite(hDev, TWXA::WPORT::POUT0, 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 41　エラー処理の例(Visual Basic)

```
Dim ret As Integer

ret = TWXA_PortWrite(hDev, TWXA_WPORT.POUT0, &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 42　エラー処理の例(VBA)

```
Dim ret As Long

ret = TWXA_PortWrite(hDev, TWXA_WPORT.POUT0, &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 43　エラー処理の例(C#)

```
TWXA.STATUS ret;

ret = TWXA.PortWrite(hDev, TWXA.WPORT.POUT0, 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 59 は参考として *TWXA_ADRead()* 関数呼び出しを 1000 回行い、関数実行に要した時間をプロットしたものです。

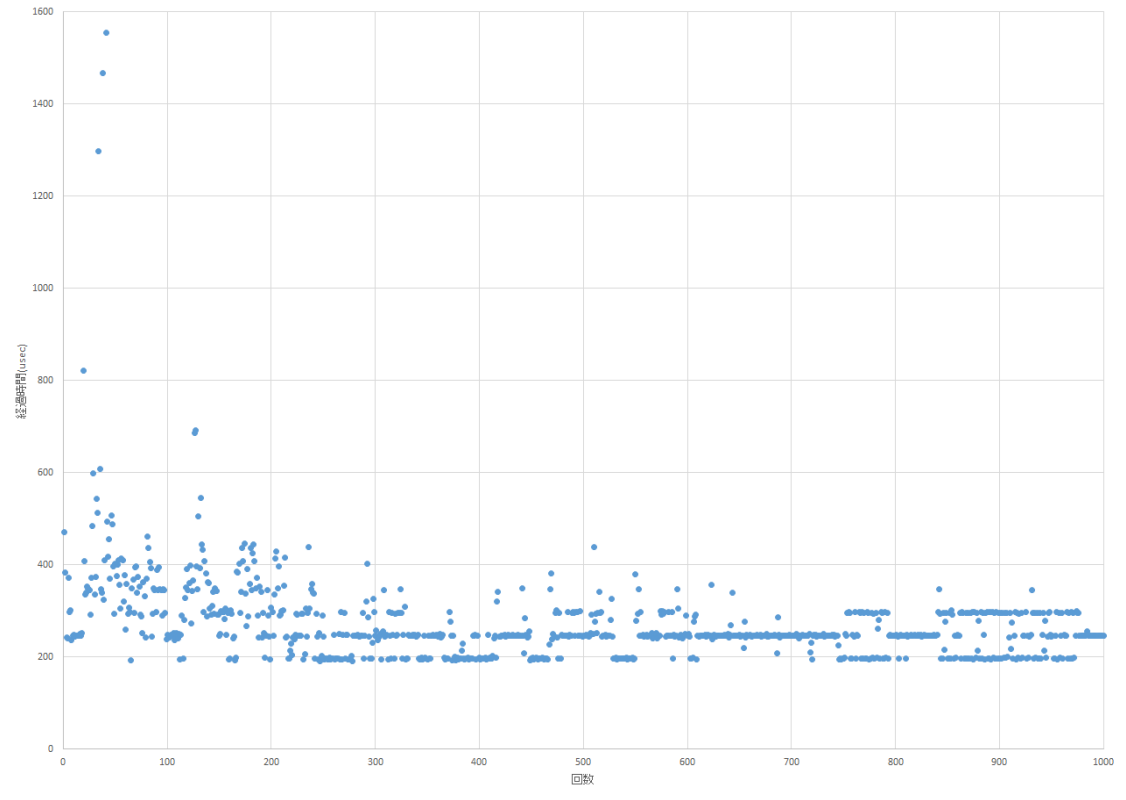


図 59　TWXA_ADRead() 関数の応答時間

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
2018 年 11 月	初	
2020 年 8 月	2	・AD コンバータの追加機能に対応 ・誤記の修正

Ⓒ テクノウェーブ株式会社 2018-2020