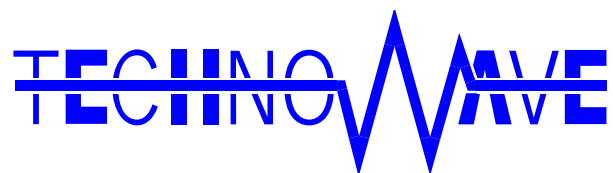


USBX-I2219 ユーザーズマニュアル



テクノウェーブ株式会社

目次

1. はじめに	5
<input type="checkbox"/> 安全にご使用いただくために.....	5
<input type="checkbox"/> その他の注意事項.....	5
<input type="checkbox"/> マニュアル内の表記について.....	6
デジタル入力端子の状態	6
デジタル出力端子の状態	6
関数・構造体名	7
引数の入力候補.....	7
Null 値	8
2. 製品概要	9
<input type="checkbox"/> 特徴.....	9
<input type="checkbox"/> 製品の利用方法	10
パソコンからの制御.....	10
ファームウェアの開発	11
<input type="checkbox"/> 関連ドキュメント	12
3. 製品仕様	13
<input type="checkbox"/> 仕様.....	13
<input type="checkbox"/> 外形寸法	16
<input type="checkbox"/> 各部の名称.....	17
<input type="checkbox"/> 端子説明	18
<input type="checkbox"/> ディップスイッチ.....	20
4. 使用準備	21
<input type="checkbox"/> DIN レール取付具の固定.....	21
<input type="checkbox"/> ドライバのインストール.....	21
Windows 10 の場合	21
Windows 7 の場合	23
Windows XP の場合	25
<input type="checkbox"/> ライブラリ、設定ツールのインストール	27
<input type="checkbox"/> LabVIEW ライブラリのインストール	28
<input type="checkbox"/> 設定ツールについて	29
<input type="checkbox"/> 装置番号設定.....	30
5. ハードウェア	31
<input type="checkbox"/> デジタル入力端子.....	31

入力回路.....	31
接続例	31
□ デジタル出力端子.....	32
出力回路.....	32
接続例	32
□ 電源出力	33
□ アナログ入力端子.....	34
入力回路.....	34
接続例	34
□ アナログ出力端子.....	35
出力回路.....	35
接続例	35
□ アナログ電源出力.....	35
6. プログラミング	36
□ プログラミングの準備	36
C/C++での開発に必要なファイル.....	36
Visual Basic、C# での開発に必要なファイル.....	37
Visual Basic for Applications での開発に必要なファイル.....	37
□ 接続.....	38
デバイスに接続する.....	38
デバイスの操作を終了する.....	39
□ デジタル入出力	41
入力端子の状態を読み取る.....	42
出力端子の状態を変更する.....	43
□ アナログ入出力	45
アナログ入力値を読み取る.....	45
アナログ出力値を変更する.....	47
□ パルスをカウントする	49
ハードウェアカウンタによる単相パルスカウント.....	50
ハードウェアカウンタによる 2 相パルスカウント	50
ハードウェアカウンタの使用法.....	51
パルスカウンタ(ソフトウェアカウンタ)による単相パルスカウント.....	54
パルスカウンタ(ソフトウェアカウンタ)による 2 相パルスカウント.....	54
パルスカウンタ(ソフトウェアカウンタ)による 3 相パルスカウント.....	55
パルスカウンタ(ソフトウェアカウンタ)の使用法	55
□ PWM 出力	59


パルスの設定方法	60
PWM 出力の手順.....	61
□ シリアルポート	64
シリアルポートの設定	65
シリアルポートの使用手順.....	66
□ ハードウェアイベントの監視.....	70
パルスカウンタ入力を監視する	73
アナログ入力を監視する	74
□ ユーザステータスレジスタ/ユーザーメモリの利用	76
ユーザステータスレジスタの操作方法	76
ユーザーメモリの操作方法.....	77
□ フラッシュメモリの利用	78
フラッシュメモリの消去方法	79
フラッシュメモリへの書き込み方法	79
□ エラー処理.....	82
APPENDIX.....	84
□ 製品の応答時間	84
保証期間.....	85
サポート情報.....	85


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。
<ul style="list-style-type: none">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。
<ul style="list-style-type: none">水や薬品のかかる可能性がある場所では使用しないでください。火災、感電の原因となります。結露の発生する環境では使用しないでください。火災、感電の原因となります。定格の範囲内でご使用ください。火災の原因となります。		

	注意	これらの注意事項を無視して誤った取り扱いをすると人が傷害を負う可能性があります。また物的損害の発生が想定されます。
<ul style="list-style-type: none">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

<ul style="list-style-type: none">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となります。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。

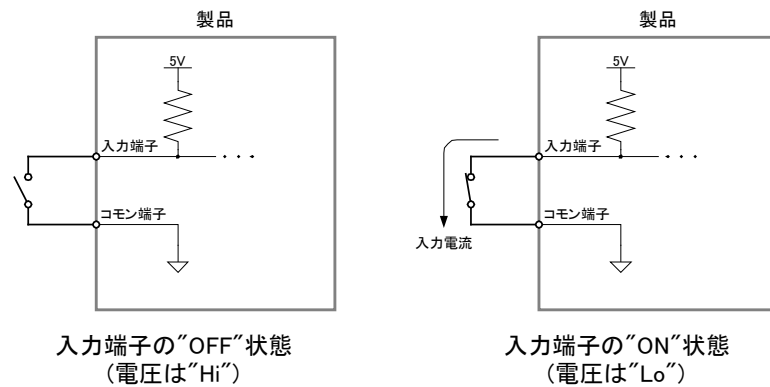


図 1 デジタル入力端子の“OFF”状態と“ON”状態

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。

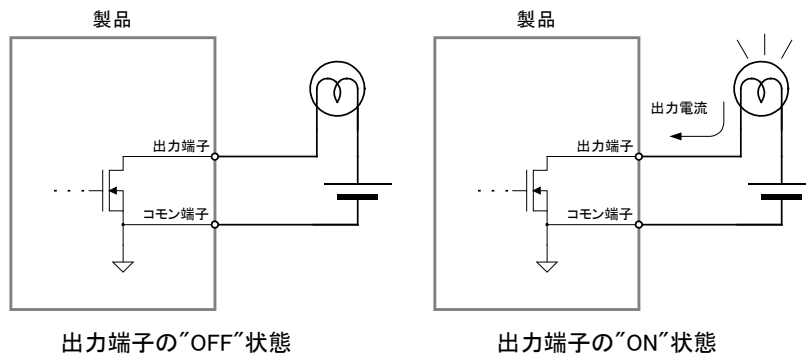


図 2 デジタル出力端子の“OFF”状態と“ON”状態

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

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++で利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++で利用できる記述方法です。定数専用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#で利用可能な記述方法です。この場合も Visual Basic 同様に関数の引数が列挙型となっています。名前空間は省略して記述しています。

表 3 引数の入力候補の例

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

Null 値

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

表 4 Null 値

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

2. 製品概要

□ 特徴

『USBX-I2219』(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。

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

- **デジタル I/O¹** - デジタル I/O として絶縁入力最大 22 点、絶縁出力最大 19 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** - 0~5V 入力の非絶縁 10 ビット AD コンバータを 4 チャンネル搭載しています。
- **DA コンバータ** - 0~5V 出力の非絶縁 8 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 2 チャンネル使用可能です。ハードウェアカウンタは 1 チャンネルの 2.5MHz 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。
- **32 ビットパルスカウンタ¹** - 最大 4 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 2 チャンネルの 2 相パルスカウンタとしても使用可能です。
- **PWM 出力¹** - 出力パルス数を指定可能な PWM 信号を最大 3 チャンネル出力することができます。
- **シリアル通信³** - 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** - パルスカウンタ、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 などで作成された Windows 上のアプリケーションプログラムから制御できます。また、ナショナルインスツルメンツ社の LabVIEW[™] にも対応していますので、グラフィカルな開発環境でのプログラミングも可能です。
- 内蔵マイコンのプログラミングはエル・アンド・エフ社の Yellow IDE(YCH8)、イエロースコープ(YSH8)に対応し、ソースコードレベルでのデバッグが可能です。
- 製品は付属の取付具を使用することで 35mmDIN レールにワンタッチで着脱できます。

¹ デジタル I/O、PWM 出力、各カウンタは一部の端子、ハードウェア機構を共有しているため、組み合わせにより同時使用できない場合があります。

² ソース出力機器は接続できません。

³ 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 ライブラリと呼ばれます。

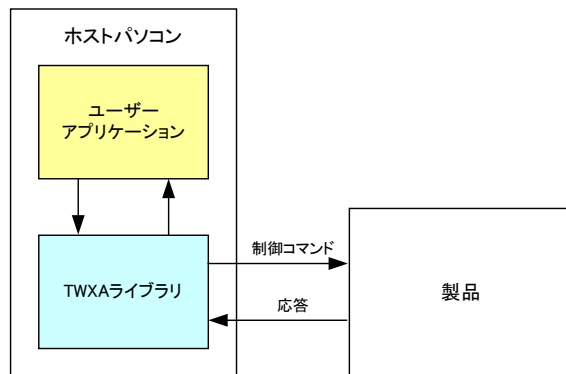


図 3 ホストパソコンからの制御

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

表 5 開発用ファイルが提供される言語

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

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

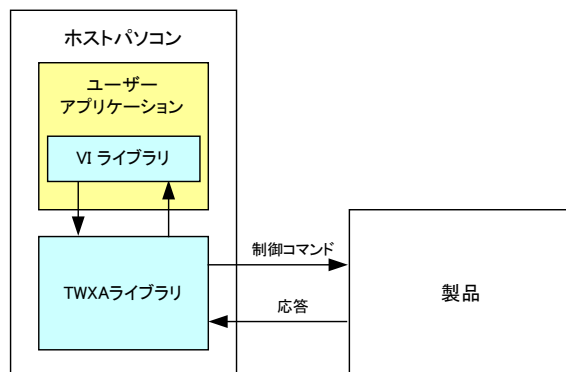


図 4 LabVIEW での利用

ファームウェアの開発

TWXA ライブラリの各関数は図 5 のように製品に組み込まれたファームウェア⁴に独自の制御コマンドを送信することで製品を制御します。最初から製品に組み込まれているこのファームウェアのことをシステムファームと呼びます。

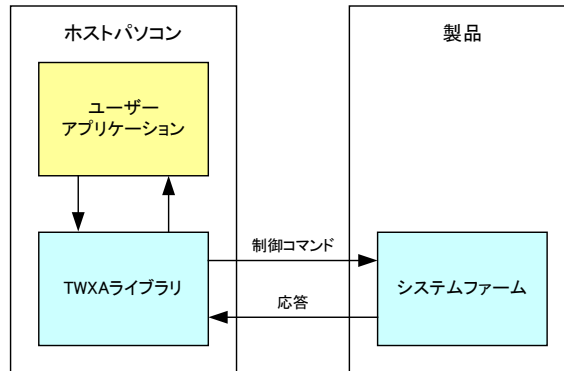


図 5 ホストパソコンからの制御

製品ではファームウェアをユーザーが開発し、動作をカスタマイズする仕組みがサポートされています。これにより、パソコンからのコマンド制御では実現が困難なりリアルタイム性が要求される処理や、基本機能では提供されないユーザー独自の機能追加が可能です。このユーザーカスタムのファームウェアのことをユーザーファームと呼びます(図 6)。

ユーザーファームの開発言語は C 言語です。詳細は別紙「USBX-I2219/LANX-I2219 ユーザーファーム開発マニュアル」を参照してください。

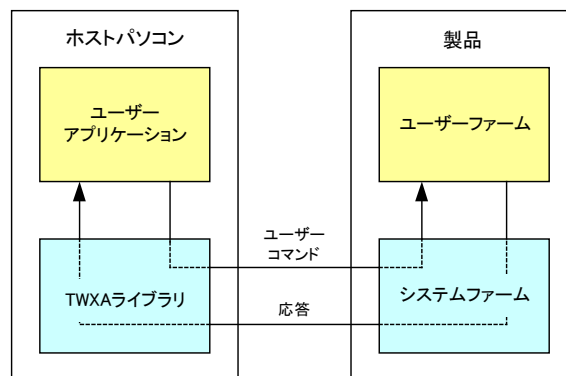


図 6 ユーザーファームの追加

⁴ パソコン上で動作するプログラムやソフトウェアと区別するために、製品内蔵のマイコンで動作するプログラムのことをファームウェア、または単にファームと呼びます。

□ 関連ドキュメント

本マニュアルでは製品の設定、ハードウェア、パソコン用プログラムの開発方法を中心に説明しています。TWXA ライブラリ関数の詳細や、VI ライブラリ、ユーザーファームの開発などについては表 6 にあが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-I2219 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-I2219.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
USBX-I2219/LANX-I2219 ユーザーファーム開発マニュアル	ユーザーファーム(製品内蔵マイコン用プログラム)の開発方法	X2219UserFirm.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	105(W)×113(D)×39.5(H)[mm]	DIN レール取付具、ゴム足含まず
重量	370[g]	付属品含まず
電源電圧	5[VDC]	
消費電流	最大 500[mA]	
動作温度範囲	0～50[°C]	
フラッシュメモリのプログラム保持年数	10[年]	
絶縁抵抗(GND-COM 間)	2000[MΩ]以上	測定条件:500VDC, GND-COM 間
インタフェース	USB2.0	HiSpeed 対応
対応 OS	Windows XP, Vista, 7, 8, 8.1, 10	

表 8 絶縁入力仕様

項目	仕様	備考	
入力点数	最大 22 点	Ia0～Ia7, Ib0～Ib7, Ic0～Ic5 Ic0～Ic3 は PC0～PC3 と共用 Ic4, Ic5 は CLK1, CLK2 と共用	
入力方式	無電圧接点入力		
入力電圧	0～5[V]	入力-COM 端子間電圧	
入力抵抗	Ic4, Ic5 以外	1kΩ	入力回路図参照
	Ic4, Ic5	300Ω	入力回路図参照
入力オン電圧	最大 1.0[V]		
フォトカプラ 応答速度	Ib6, Ib7, Ic0～Ic5 以外	最大 100[μsec]	
	Ib6, Ib7, Ic0～Ic3	最大 60[μsec]	
	Ic4, Ic5	最大 100[nsec]	

表 9 絶縁出力仕様

項目	仕様	備考
出力点数	最大 19 点	Od0～Od7, Oe0～Oe7, Of0～Of2 Of0～Of2 は PWM0～PWM2 と共用
出力方式	オープンドレイン	
出力電圧	最大 25.2[V]	出力-COM 端子間電圧
出力電流(シンク電流)	最大 150[mA]	条件:出力-COM 端子間電圧 0.8V
フォトカプラ応答速度 (Of0～Of2 以外)	最大 100[μsec]	
絶縁素子応答速度(Of0～Of2)	最大 150[nsec]	

表 10 PWM 出力仕様

項目	仕様	備考
出力チャンネル数	最大 3 チャンネル	Of0/PWM0～Of2/PWM2
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	
絶縁素子応答速度	絶縁出力仕様(Of0～Of2)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:5V 出力、36Ω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 2 チャンネル	Ic4/CLK1, Ic5/CLK2
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(Ic4, Ic5)参照	
フォトブラ応答速度		絶縁入力仕様(Ic4, Ic5)参照	
カウンタビット数		16 ビット	
カウント設定		ON→OFF、OFF→ON、両エッジ	
周波数	単相	最大 5[MHz]	
	2 相	最大 2.5[MHz]	

表 12 パルスカウンタ(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 4 チャンネル	Ic0/PC0~Ic3/PC3
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(Ic0~Ic3)参照	
フォトブラ応答速度		絶縁入力仕様(Ic0~Ic3)参照	
カウンタビット数		32 ビット	
カウント設定		ON→OFF、OFF→ON	
周波数		最大 5[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
入力電圧		0~5.0[V]	
変換部	分解能	10[bit]	
	リファレンス精度	最大±0.3[%]	条件:全温度範囲
	リファレンス温度偏差	最大±25[ppm]	
	非直線性誤差	最大±3.5[LSB]	条件:全温度範囲
	オフセット誤差	最大±3.5[LSB]	条件:全温度範囲
	フルスケール誤差	最大±3.5[LSB]	条件:全温度範囲
	量子化誤差	最大±0.5[LSB]	条件:全温度範囲
	絶対精度	最大±4.0[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	8[bit]	
	リファレンス精度	AD コンバータ仕様を参照	
	リファレンス温度偏差	AD コンバータ仕様を参照	
	絶対精度	最大±1.5[LSB]	条件:全温度範囲
アンプ部	オフセット電圧	最大±5[mV]	条件:全温度範囲

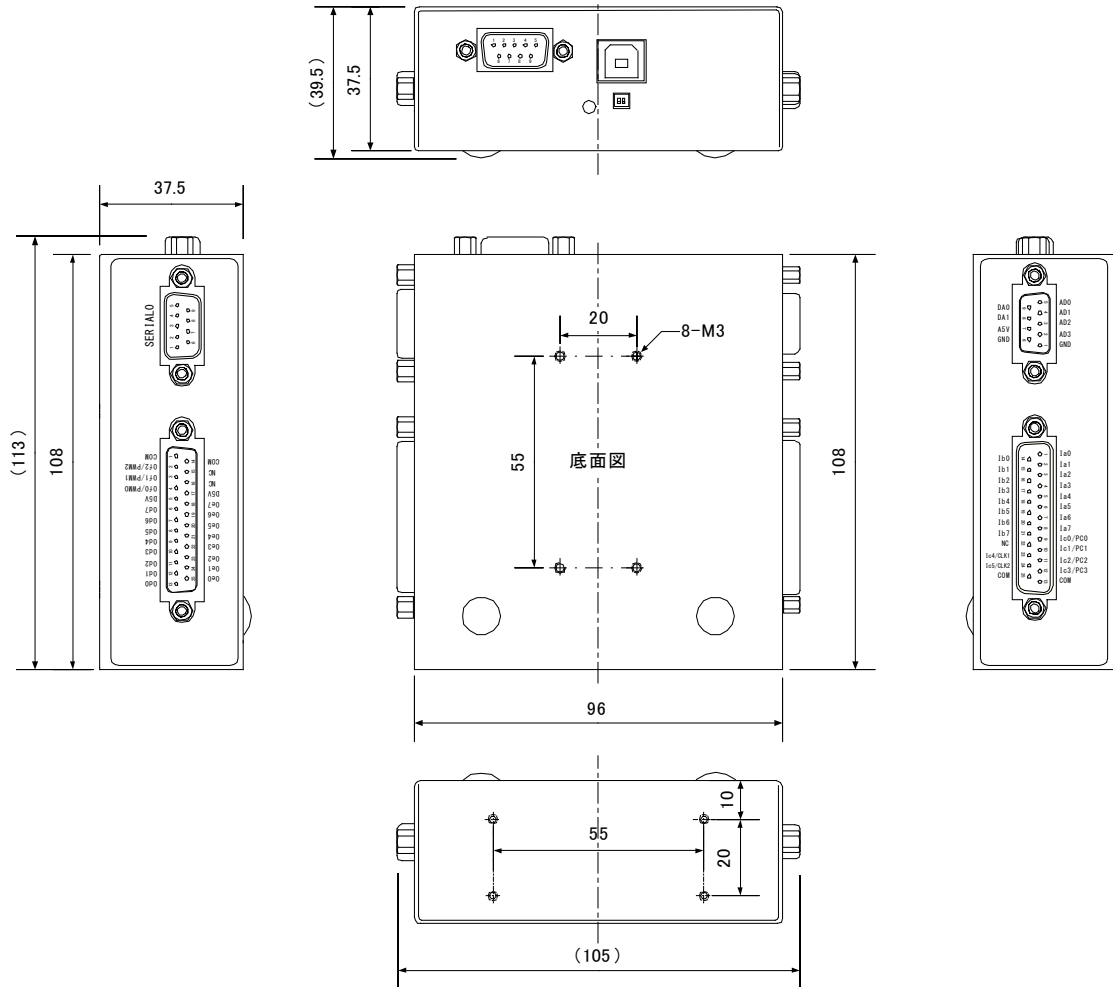
表 15 シリアルポート仕様

項目	仕様	備考
チャンネル数	2	
方式	調歩同期式(フロー制御なし) ⁵	
ビットレート	300~38400bps	
信号レベル	RS-232C 準拠	

- デジタル出力端子の Of0~Of2 と PWM 出力端子 PWM0~PWM2 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 Ic0~Ic3 とパルスカウンタ入力 PC0~PC3 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 Ic4、Ic5 とハードウェアカウンタ入力 CLK1、CLK2 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 Ib6、Ib7 はパルスカウンタで 2 相入力する場合に B 相信号の入力端子として使用します。パルスカウンタを 2 相信号のカウントモードで使用する場合、デジタル入力端子としては使用できません。
- 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 3 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。コネクタ上の RTSは CTS と、DTR は DSR と機器内部で接続されています。

□ 外形寸法



□ 各部の名称

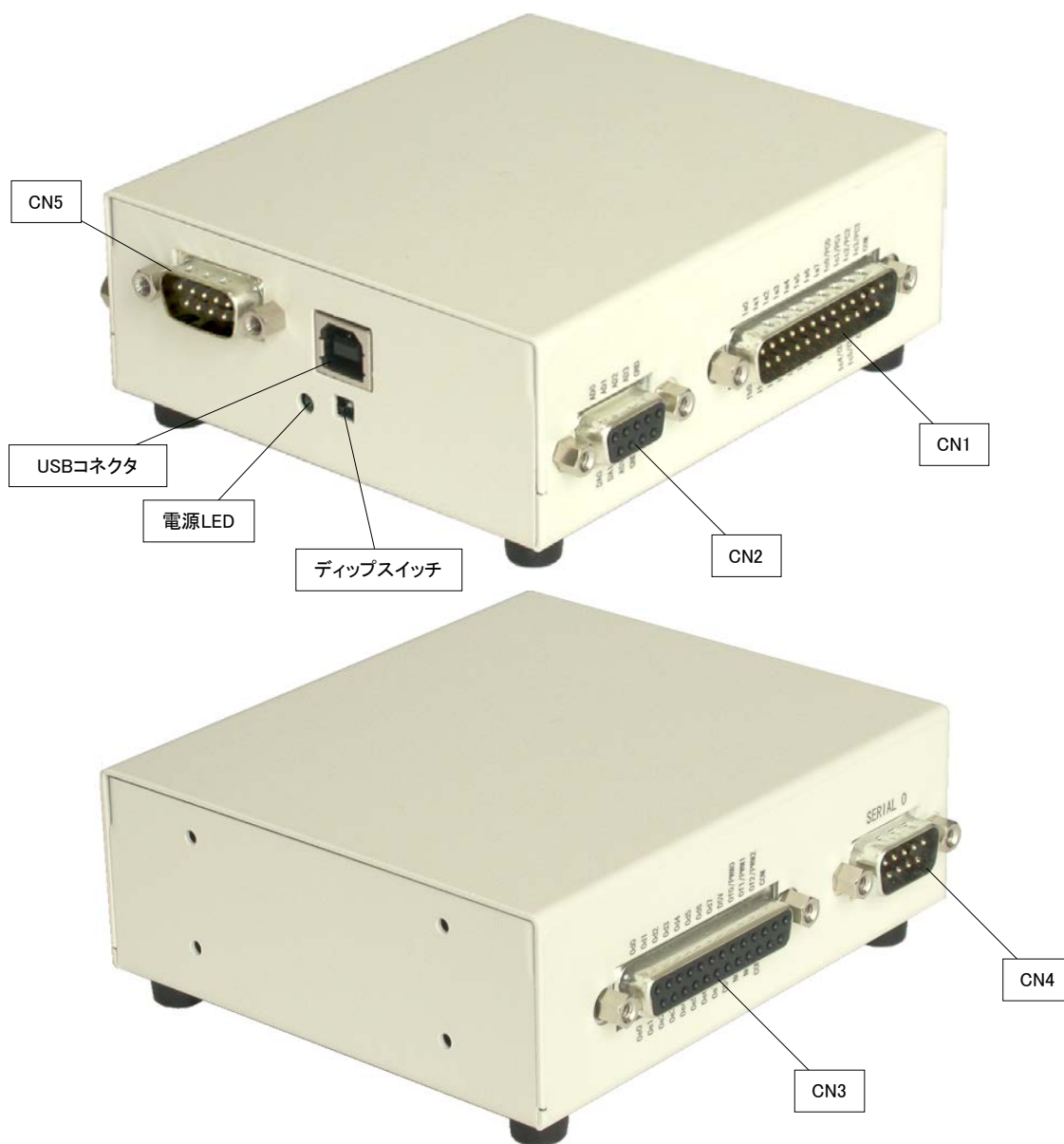


図 8 各部の名称

□ 端子説明

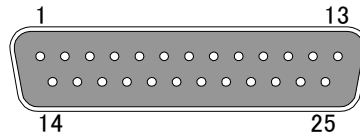


図 9 CN1 端子配列

表 16 CN1 端子(D-sub25 ピン-オス)

コネクタ-ピン番	信号名	説明	方向	絶縁/非絶縁
CN1-1	Ia0	デジタル入力	I	絶縁
CN1-2	Ia1	デジタル入力	I	絶縁
CN1-3	Ia2	デジタル入力	I	絶縁
CN1-4	Ia3	デジタル入力	I	絶縁
CN1-5	Ia4	デジタル入力	I	絶縁
CN1-6	Ia5	デジタル入力	I	絶縁
CN1-7	Ia6	デジタル入力	I	絶縁
CN1-8	Ia7	デジタル入力	I	絶縁
CN1-9	Ic0/PC0	デジタル入力/パルスカウンタ 0	I	絶縁
CN1-10	Ic1/PC1	デジタル入力/パルスカウンタ 1	I	絶縁
CN1-11	Ic2/PC2	デジタル入力/パルスカウンタ 2	I	絶縁
CN1-12	Ic3/PC3	デジタル入力/パルスカウンタ 3	I	絶縁
CN1-13	COM	コモン	-	絶縁
CN1-14	Ib0	デジタル入力	I	絶縁
CN1-15	Ib1	デジタル入力	I	絶縁
CN1-16	Ib2	デジタル入力	I	絶縁
CN1-17	Ib3	デジタル入力	I	絶縁
CN1-18	Ib4	デジタル入力	I	絶縁
CN1-19	Ib5	デジタル入力	I	絶縁
CN1-20	Ib6	デジタル入力	I	絶縁
CN1-21	Ib7	デジタル入力	I	絶縁
CN1-22	RSV0	接続しないでください	-	
CN1-23	Ic4/GLK1	デジタル入力/ハードウェアカウンタ 1	I	絶縁
CN1-24	Ic5/GLK2	デジタル入力/ハードウェアカウンタ 2	I	絶縁
CN1-25	COM	コモン	-	絶縁

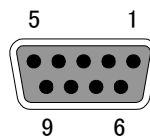


図 10 CN2 端子配列

表 17 CN2 端子(D-sub 9 ピン-メス)

コネクタ-ピン番	信号名	説明	方向	絶縁/非絶縁
CN2-1	GND	シグナルグランド	-	非絶縁
CN2-2	AD3	AD コンバータ 3	I	非絶縁
CN2-3	AD2	AD コンバータ 2	I	非絶縁
CN2-4	AD1	AD コンバータ 1	I	非絶縁
CN2-5	AD0	AD コンバータ 0	I	非絶縁
CN2-6	GND	シグナルグランド	-	非絶縁
CN2-7	A5V	アナログ用 5V 出力	0	非絶縁
CN2-8	DA1	DA コンバータ 1	0	非絶縁
CN2-9	DA0	DA コンバータ 0	0	非絶縁

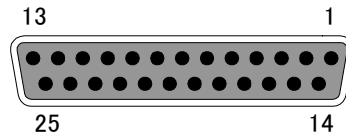


図 11 CN3 端子配列

表 18 CN3 端子(D-sub25 ピン-メス)

コネクタ-ピン番	信号名	説明	方向	絶縁/非絶縁
CN3-1	COM	コモン	-	絶縁
CN3-2	Of2/PWM2	デジタル出力/PWM 出力 2	0	絶縁
CN3-3	Of1/PWM1	デジタル出力/PWM 出力 1	0	絶縁
CN3-4	Of0/PWM0	デジタル出力/PWM 出力 0	0	絶縁
CN3-5	D5V	デジタル用 5V 出力	0	絶縁
CN3-6	Od7	デジタル出力	0	絶縁
CN3-7	Od6	デジタル出力	0	絶縁
CN3-8	Od5	デジタル出力	0	絶縁
CN3-9	Od4	デジタル出力	0	絶縁
CN3-10	Od3	デジタル出力	0	絶縁
CN3-11	Od2	デジタル出力	0	絶縁
CN3-12	Od1	デジタル出力	0	絶縁
CN3-13	Od0	デジタル出力	0	絶縁
CN3-14	COM	コモン	-	絶縁
CN3-15	NC	未接続	-	
CN3-16	NC	未接続	-	
CN3-17	D5V	デジタル用 5V 出力	0	絶縁
CN3-18	Oe7	デジタル出力	0	絶縁
CN3-19	Oe6	デジタル出力	0	絶縁
CN3-20	Oe5	デジタル出力	0	絶縁
CN3-21	Oe4	デジタル出力	0	絶縁
CN3-22	Oe3	デジタル出力	0	絶縁
CN3-23	Oe2	デジタル出力	0	絶縁
CN3-24	Oe1	デジタル出力	0	絶縁
CN3-25	Oe0	デジタル出力	0	絶縁

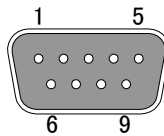


図 12 CN4,CN5 の端子配列

表 19 CN4 端子(D-sub 9 ピン-オス)

コネクタ-ピン番	信号名	説明	方向	絶縁/非絶縁	パソコンのシリアルポートでの信号名(参考)
CN4-1	NC	未接続	-	-	DCD
CN4-2	RxD0	シリアル入力 0	1	非絶縁	RxD
CN4-3	TxD0	シリアル出力 0	0	非絶縁	TxD
CN4-4	RSV1	CN4-6 と接続されています。	-	-	DTR
CN4-5	GND	シグナルグランド	-	非絶縁	GND
CN4-6	RSV1	CN4-4 と接続されています。	-	-	DSR
CN4-7	RSV2	CN4-8 と接続されています。	-	-	RTS
CN4-8	RSV2	CN4-7 と接続されています。	-	-	CTS
CN4-9	NC	未接続	-	-	RI

表 20 CN5 端子(D-sub 9 ピン-オス)

コネクタ-ピン番	信号名	説明	方向	絶縁 /非絶縁	パソコンのシリアルポ ートでの信号名(参考)
CN5-1	NC	未接続	-	-	DCD
CN5-2	RxD1	シリアル入力 1	I	非絶縁	RxD
CN5-3	TxD1	シリアル出力 1	O	非絶縁	TxD
CN5-4	RSV1	CN5-6 と接続されています。	-	-	DTR
CN5-5	GND	シグナルグランド	-	非絶縁	GND
CN5-6	RSV1	CN5-4 と接続されています。	-	-	DSR
CN5-7	RSV2	CN5-8 と接続されています。	-	-	RTS
CN5-8	RSV2	CN5-7 と接続されています。	-	-	CTS
CN5-9	NC	未接続	-	-	RI

□ デイップスイッチ

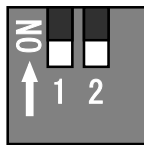


図 13 デイップスイッチ

表 21 デイップスイッチ

番号	説明
1	通常は“OFF”で使います。ユーザーファームを起動する場合に“ON”にします。
2	通常は“OFF”で使います。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. 使用準備

□ DIN レール取付具の固定

DIN レール取付具を使用し、垂直面に取り付ける場合は図 14、図 15 の向きで使用することを推奨します。

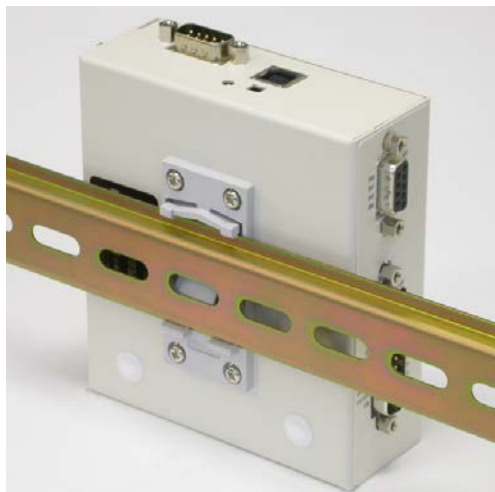


図 14 DIN レール取付具の向き(1)

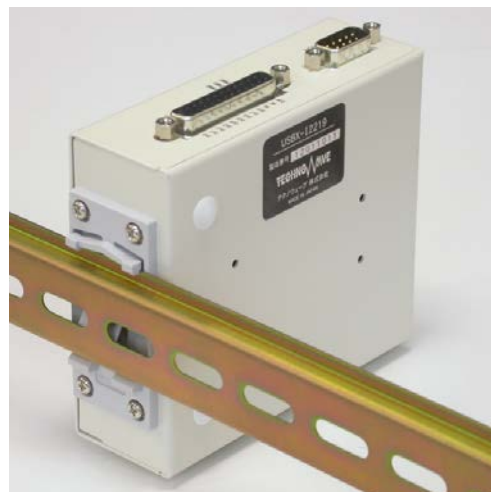


図 15 DIN レール取付具の向き(2)

□ ドライバのインストール

ドライバは付属 CD-ROM に納められています。

表 22 ドライバファイルの格納フォルダ

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

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

- 製品をパソコンに接続する前にドライバのセットアップを行ってください。

Windows 10 の場合

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



図 16 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。

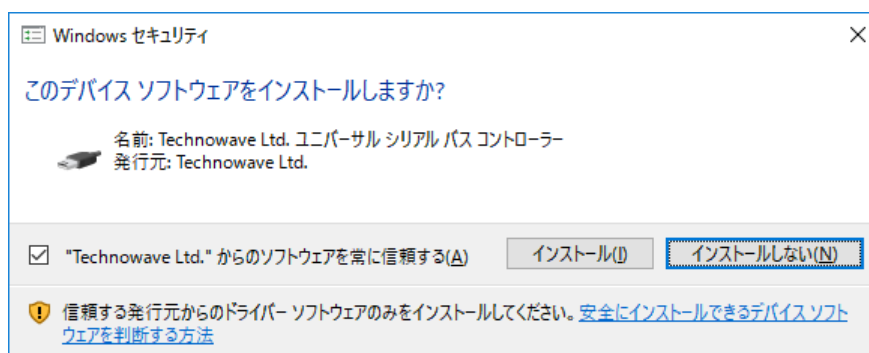


図 17 Windows 10 のドライバインストール画面 (2)

- 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 18 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「USBM3069-H USB Device」と表示されれば、ドライバが正しくインストールされています。

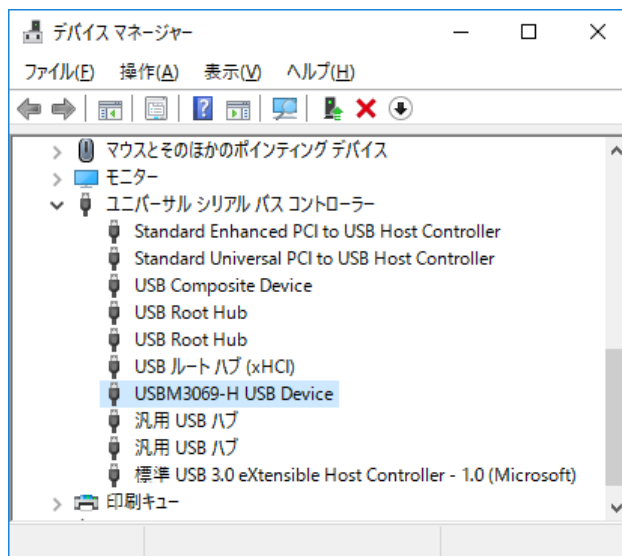


図 19 Windows 10 のドライバインストール確認

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

Windows 7 の場合

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

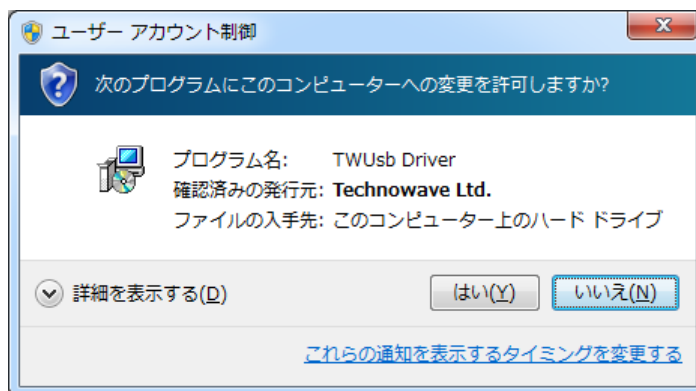


図 20 Windows 7 のドライバインストール画面 (1)

- 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。

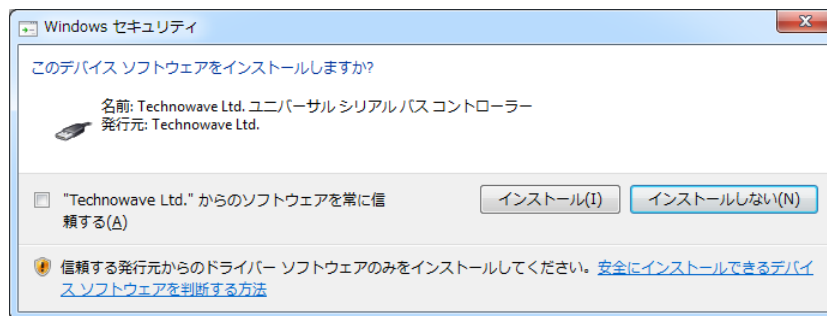


図 21 Windows 7 のドライバインストール画面 (2)

- 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 22 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「USBM3069-H USB Device」と表示されれば、ドライバが正しくインストールされています。

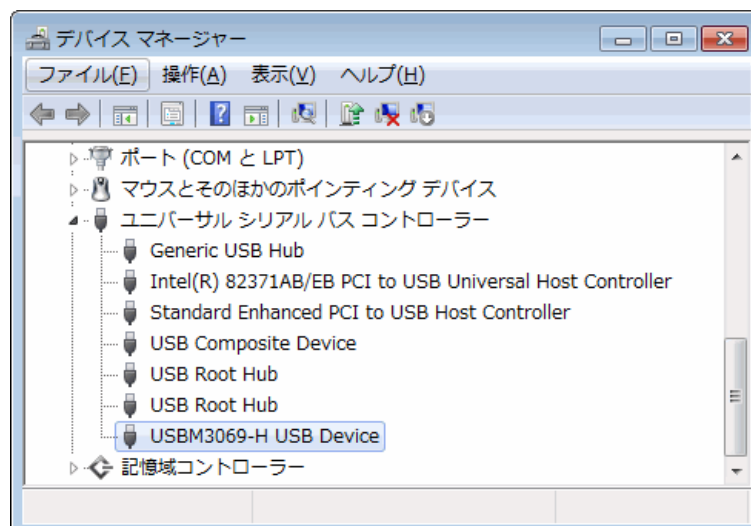


図 23 Windows 7 のドライバインストール確認

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

Windows XP の場合

- ① 「setup.exe」を起動し、画面の指示に従ってインストールを行います。
- ② 下のような画面が表示されたら[続行]ボタンを押してインストールを続けてください。

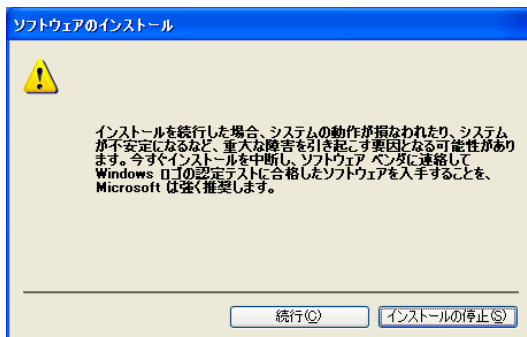


図 24 Windows XP のドライバインストール画面 (1)

- ③ インストールが終了すると、次のような画面が表示されますので[完了]ボタンを押してください。



図 25 Windows XP のドライバインストール画面 (2)

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

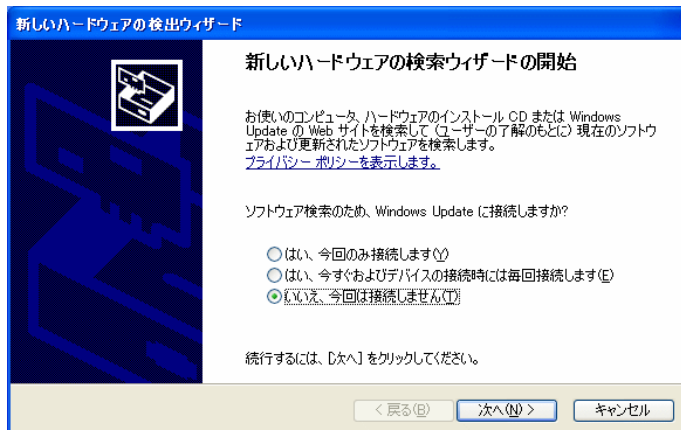


図 26 Windows XP のドライバインストール画面 (3)

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

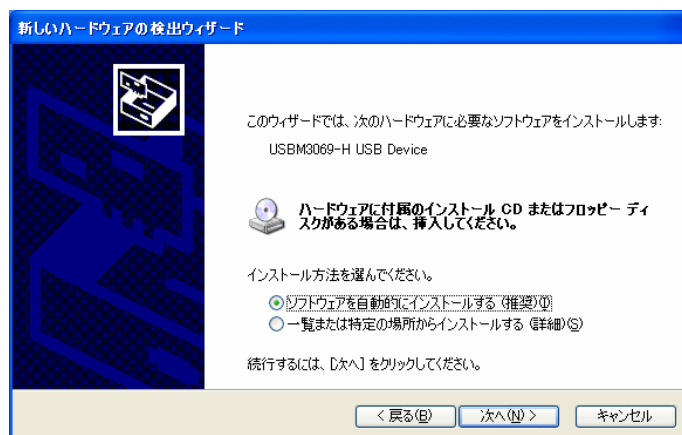


図 27 Windows XP のドライバインストール画面 (4)

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

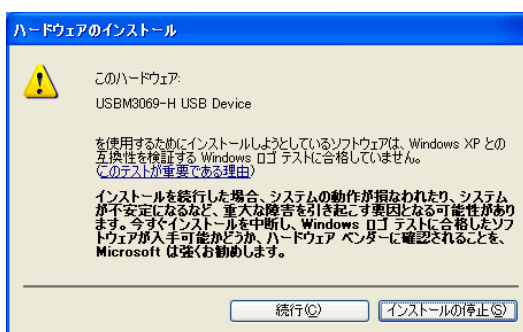


図 28 Windows XP のドライバインストール画面 (5)

- ⑦ 図のような画面が表示されますので、[完了]ボタンを押します。



図 29 Windows XP のドライバインストール画面 (6)

- ⑧ 図 30 のように「デバイス マネージャ」の画面に「USBM3069-H USB Device」と表示されれば、ドライバが正しくインストールされています。



図 30 Windows XP のドライバインストール確認

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

□ ライブラリ、設定ツールのインストール

付属 CD の「¥TOOL¥USBX2219Tools」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 23 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⁶。

表 23 製品の制御に必要なファイル

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

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

⁶ ドライバのインストールは必要です。

□ LabVIEW ライブラリのインストール

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

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

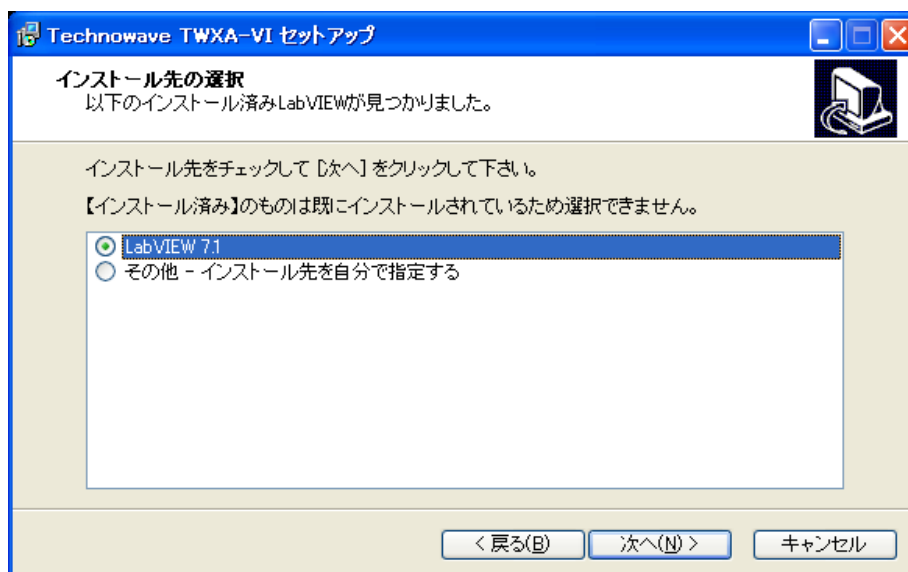


図 31 VI ライブラリのセットアップ画面

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

□ 設定ツールについて

27 ページの内容に従って設定ツールをインストールすると、[スタート]メニューの中に設定ツールの起動メニューが追加されます。デフォルトのインストールオプションでは[スタート]→[すべてのプログラム](または、[プログラム])→[テクノウェーブ]→[USBX2219Tools]から起動することができます。

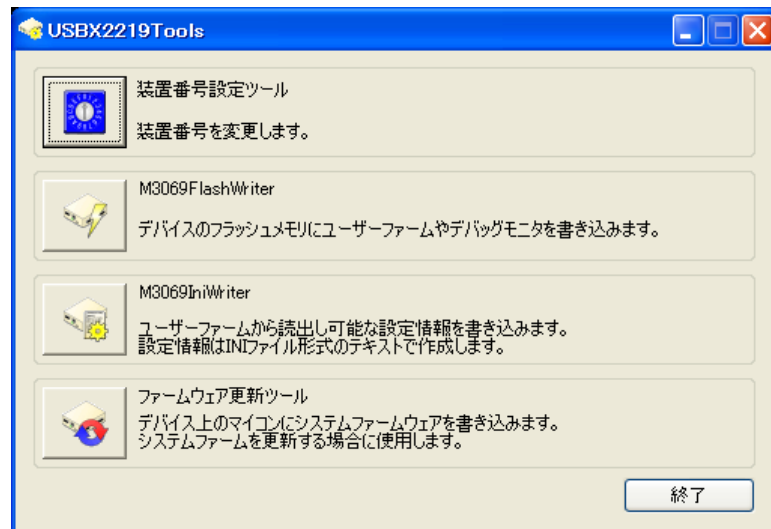


図 32 設定ツールのメニュー画面

表 24 設定ツールの機能説明

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

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

- システムファームはバグの修正や、機能追加のために不定期に新しいバージョンのものが公開されます⁷。システムファームの更新ファイルは設定ツールの中に含まれていますので、更新する場合には、まず新しい設定ツールをご利用のパソコンにインストールしてください。

⁷ 弊社ホームページにて随時公開します。

□ 装置番号設定

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

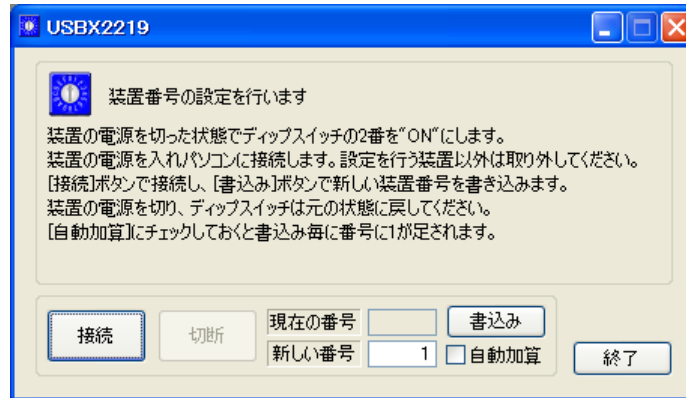


図 33 番号設定ツールの画面

1. 設定する製品のディップスイッチ 2 番を"ON"にしてフラッシュ書換えモードとし、パソコンに接続します。設定ツールは最初に見つかった製品に接続しますので、設定対象以外の製品は取り外してください。
2. 設定ツールのメニュー画面(図 32)から[装置番号設定ツール]ボタンを押します。図 33 のような画面が表示されます。
3. [接続]ボタンを押して製品に接続します。
4. [新しい番号]に 1～65535 の範囲の数値を入力します。
5. [自動加算]にチェックを入れておくと、書込みを行う度に[新しい番号]が 1 ずつ増加します。
6. [書込み]ボタンを押すと入力した装置番号が製品に設定されます。TWXA ライブラリの関数からは入力した番号を指定して接続を行うことができますようになります。
7. 製品を取り外しディップスイッチの 2 番を"OFF"に戻してください。番号の書換え可能回数の目安は 3200 回です。

5. ハードウェア

□ デジタル入力端子 入力回路

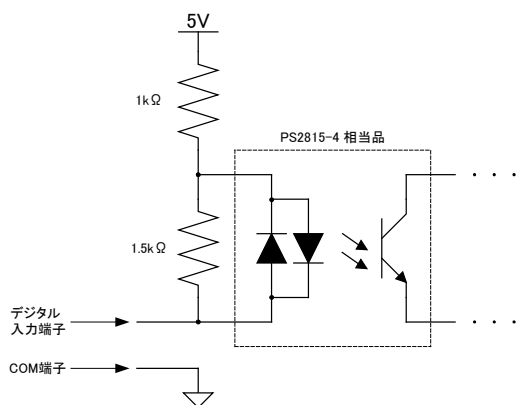


図 34 デジタル入力回路(Ic4,Ic5 以外)

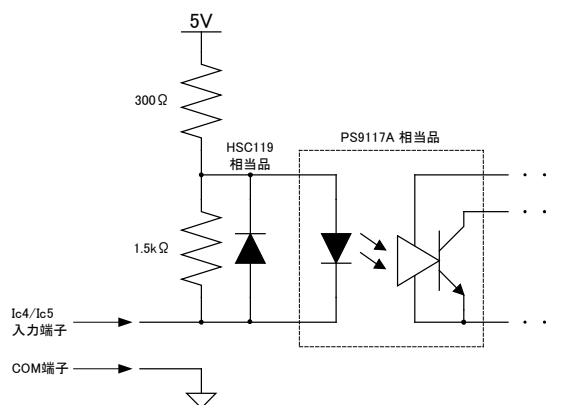


図 35 デジタル入力回路(Ic4, Ic5)

接続例

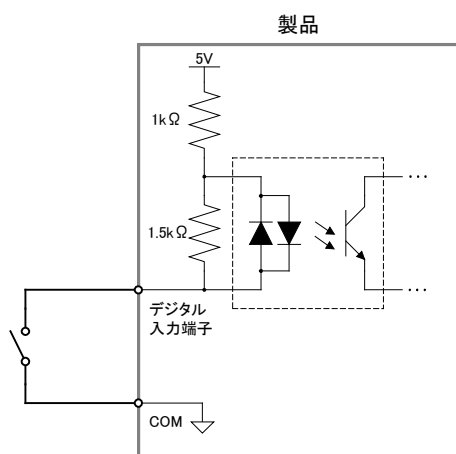


図 36 スイッチの接続例

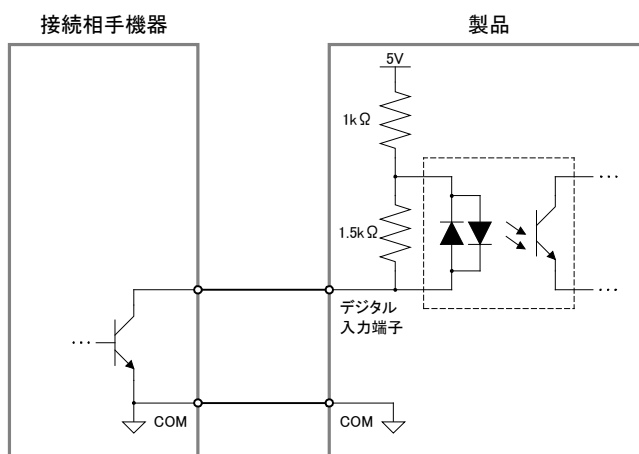


図 37 オープンコレクタ出力の機器との接続例

- 製品や接続相手機器の故障の原因となりますので、入力端子-COM 端子間に 5V を超える電圧を入力しないでください。
- 全ての COM 端子、5V 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

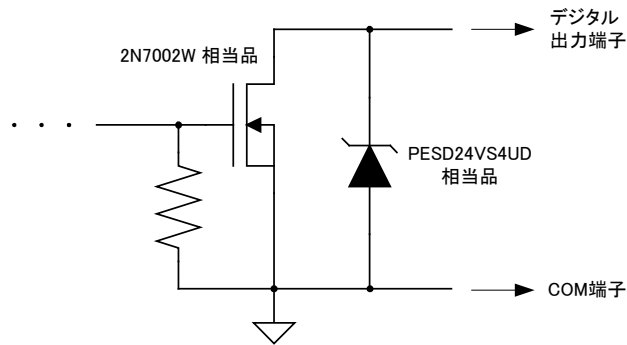


図 38 デジタル出力回路

接続例

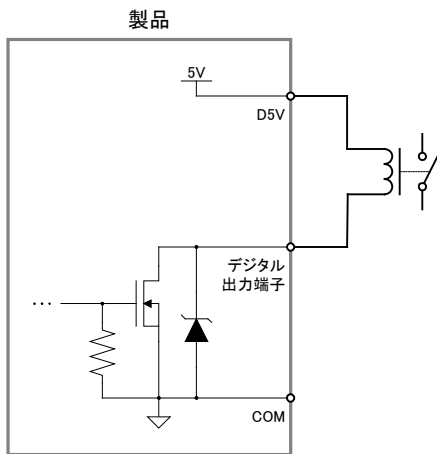


図 39 5V リレーの接続例

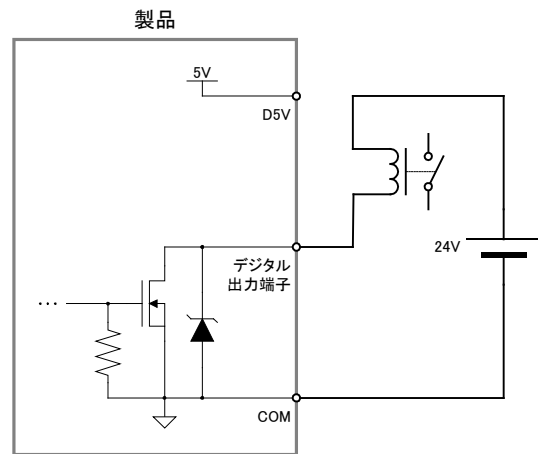


図 40 24V リレーの接続例

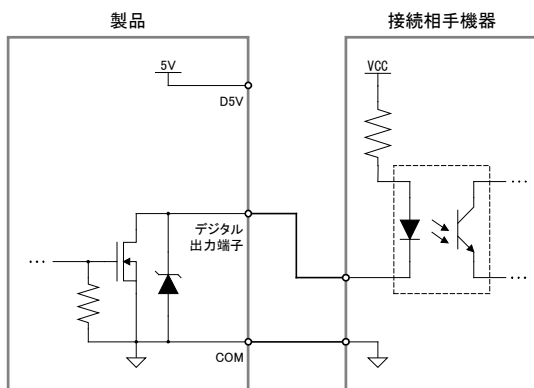


図 41 フォトカプラ入力機器(電源内蔵)との接続例

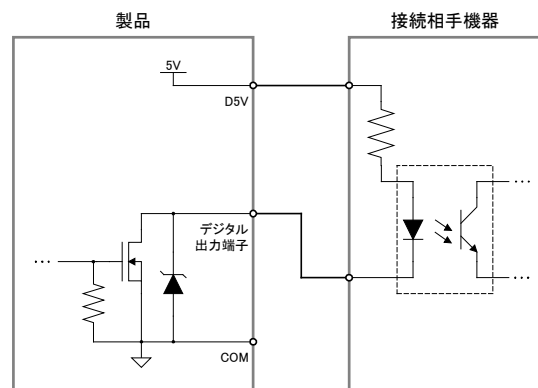


図 42 フォトカプラ入力機器(5V入力)との接続例

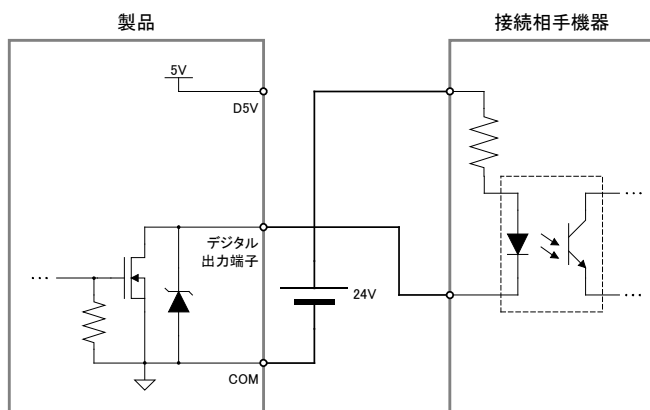


図 43 フォトコプラ入力機器(24V 入力)との接続例

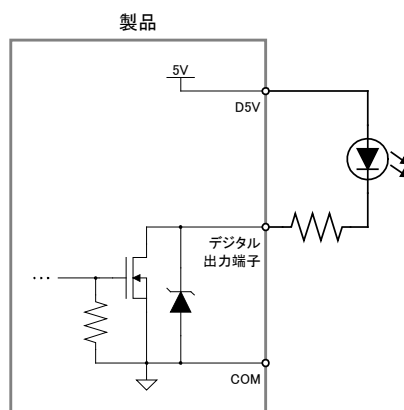


図 44 LED の接続例

□ 電源出力

D5V 端子から出力される 5V 電源は 80mA 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、Ic4/CLK1、Ic5/CLK2 端子については 1 端子あたり 15mA、その他のデジタル入力端子については 5.5mA 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 220mA まで利用可能となります。

$$80[\text{mA}] + (15[\text{mA}] \times 2) + (5.5[\text{mA}] \times 20) = 220[\text{mA}]$$

□ アナログ入力端子
入力回路

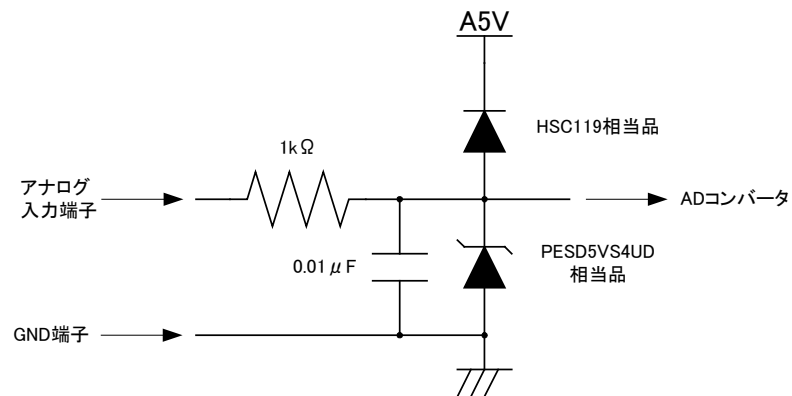


図 45 アナログ入力回路

接続例

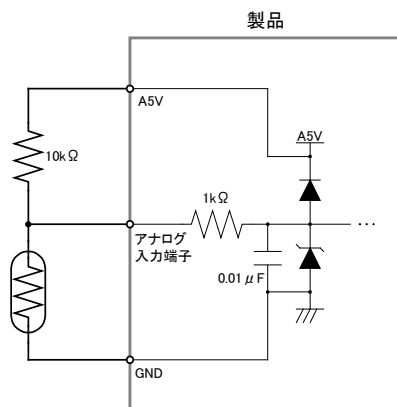


図 46 サーミスタの接続例

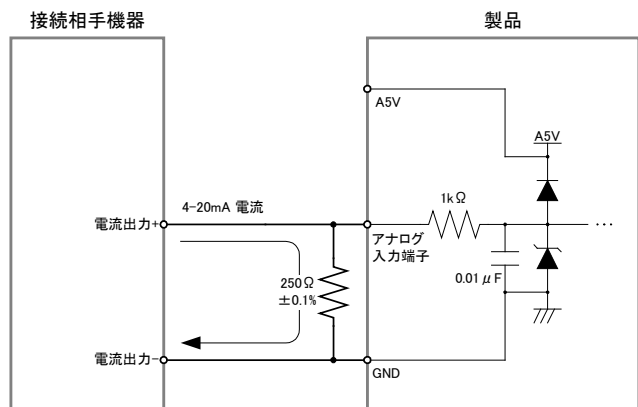


図 47 4-20mA 出力機器との接続例

- 電源が入っていない製品のアナログ入力端子に電圧を入力すると、通常よりも大きな電流(数 mA)が流れます。
- A5V 端子から出力できる電流は最大 4mA です。

□ **アナログ出力端子**
出力回路

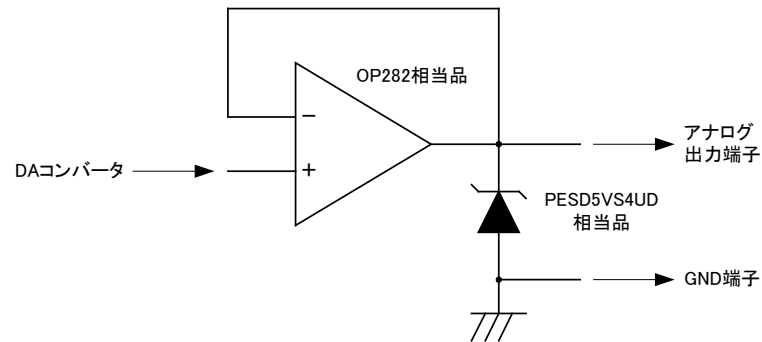


図 48 アナログ出力回路

接続例

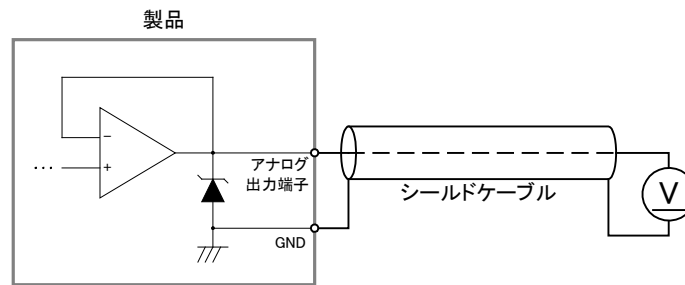


図 49 アナログ出力接続例

- 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。外部機器などを駆動することはできません。図 46(34 ページ)に示した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。

6. プログラミング

Visual Studio 用のサンプルプログラム⁸は、付属 CD の「¥SAMPLE¥I2219_Samples」フォルダ中に収められています。言語別にソリューションファイル(表 25)が準備されていますので、必要に応じてご参照ください。

表 25 言語別のサンプルファイル

言語	ソリューションファイル
Visual C++(MFC)	X2219SamplesMFC.sln
Visual Basic	X2219SamplesVB.sln
Visual C#	X2219SamplesCS.sln

Visual Basic for Applications用のサンプルは付属CDの「¥SAMPLE¥X2219_Samples¥VBASamples」フォルダ内に格納されています。

□ プログラミングの準備

C/C++での開発に必要なファイル

表 26 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(「USBX2219Tools」)をインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 26 C/C++での開発に必要なファイル

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

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

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

リスト 1 インクルードとリンク指定

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

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

⁸ Visual Studio 2005 で作成されています。ご利用のバージョンによっては変換作業が必要になります(ソリューションファイルを開くと自動的に変換ウィザードが起動します)。

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

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

Visual Basic、C# での開発に必要なファイル

表 27 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(「USBX2219Tools」)をインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 27 Visual Basic、C#での開発に必要なファイル

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

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

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

表 28 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(「USBX2219Tools」)をインストールした場合は、ローカルドライブにコピーが作られ、デフォルトの設定では[スタート]メニュー→[すべてのプログラム](または[プログラム])→[テクノウェーブ]→[ライブラリ]を選択して表示することができます。

表 28 Visual Basic for Applications での開発に必要なファイル

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

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

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

□ **接続**

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 50)。以降の操作は取得したハンドルを使用して行いますので、ハンドルの値は製品の操作を終了するまで記憶しておく必要があります。

また、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。

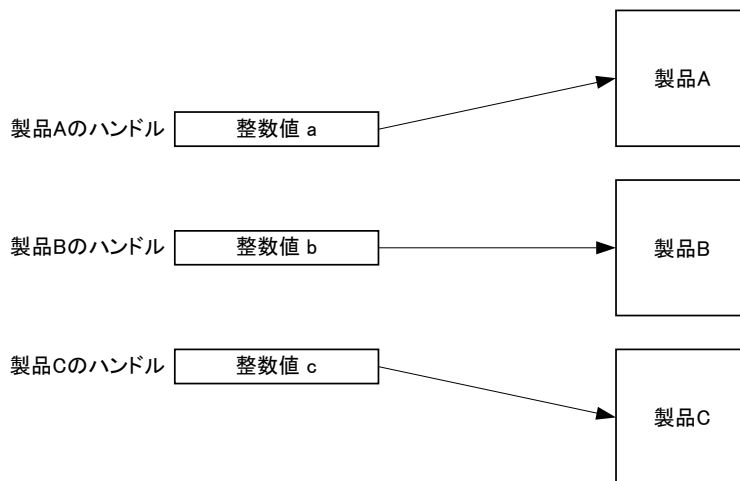


図 50 ハンドル

表 29 接続、初期化、終了に使用する関数

関数名	説明
<i>TWXA_Open()</i>	デバイスに接続します。
<i>TWXA_Close()</i>	ハンドルをクローズし、デバイスの操作を終了します。
<i>TWXA_CloseAll()</i>	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。
<i>TWXA_Initialize()</i>	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 30 の *TWXA_Open()* 関数を使用します。

表 30 *TWXA_Open()* の関数宣言

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

`TWXA_Open()` 関数では装置番号を指定してデバイスに接続できます。装置番号を指定する場合は引数 `Number` に番号を指定します。`Number` を 0 とした場合は、装置番号と無関係に最初に見つかったデバイスに接続されます。装置番号の設定方法は 29 ページを参照してください。

デバイスの操作を終了する

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

リスト 2 接続/切断の例(C言語)

```
TW_HANDLE hDev;

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

    //... 制御の中身

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

リスト 3 接続/切断の例(Visual Basic)

```
Dim hDev As System.IntPtr

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

If hDev <> System.IntPtr.Zero Then

    '... 制御の中身

    TWXA_Close (hDev)
End If
```

リスト 4 接続/切断の例(VBA)

```
Dim hDev As Long

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

If hDev <> 0 Then

    '... 制御の中身

    TWXA_Close hDev
End If
```

リスト 5 接続／切断の例(C#)

```
System.IntPtr hDev;

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

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 31 に示します。入力端子／出力端子は最大 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 31 入出力端子

端子名	端子数	方向	ポート名	兼用端子
Ia0-Ia7	8	入力	PIa	-
Ib0-Ib7	8	入力	PIb	-
Ic0-Ic5	6	入力	PIc	PC0-PC3,CLK1,CLK2 と兼用
Od0-Od7	8	出力	POd	-
Oe0-Oe7	8	出力	POe	-
Of0-Of2	3	出力	POf	PWM0-PWM2 と兼用

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 32 の関数を使用します。また、表 33 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 32 デジタル入出力で使用する関数

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

表 33 デジタル入出力のサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	PortSample	入力端子の状態を表示し、出力端子状態を操作できます。
Visual Basic	PortSampleVB	
Visual C#	PortSampleCS	
VBA (Excel)	PortSample1.xls	簡易プログラマブルタイマです。テーブルに指定した時刻に出力ポートを操作します。
	PortSample2.xls	簡易データロガーです。入力ポートを監視し、変化があると時刻と状態を記録します。

入力端子の状態を読み取る

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

表 34 `TWXA_PortRead()` の関数宣言

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

表 35 `TWXA_PortRead()` の `Port` 引数に指定する値

言語	値	説明
C/C++	<code>TWXA_PiA</code>	Ia0-Ia7 入力を読み取ります。
C++	<code>TWXA::RPORT::PiA</code>	
VB/VBA	<code>TWXA_RPORT.PiA</code>	
C#	<code>TWXA.RPORT.PiA</code>	
C/C++	<code>TWXA_PiB</code>	Ib0-Ib7 入力を読み取ります。
C++	<code>TWXA::RPORT::PiB</code>	
VB/VBA	<code>TWXA_RPORT.PiB</code>	
C#	<code>TWXA.RPORT.PiB</code>	
C/C++	<code>TWXA_PiC</code>	Ic0-Ic5 入力を読み取ります。
C++	<code>TWXA::RPORT::PiC</code>	
VB/VBA	<code>TWXA_RPORT.PiC</code>	
C#	<code>TWXA.RPORT.PiC</code>	
C/C++	<code>TWXA_PoD</code>	Od0-Od7 の出力値を読み取ります。
C++	<code>TWXA::RPORT::PoD</code>	
VB/VBA	<code>TWXA_RPORT.PoD</code>	
C#	<code>TWXA.RPORT.PoD</code>	
C/C++	<code>TWXA_PoE</code>	Oe0-Oe7 の出力値を読み取ります。
C++	<code>TWXA::RPORT::PoE</code>	
VB/VBA	<code>TWXA_RPORT.PoE</code>	
C#	<code>TWXA.RPORT.PoE</code>	
C/C++	<code>TWXA_PoF</code>	Of0-Of2 の出力値を読み取ります。
C++	<code>TWXA::RPORT::PoF</code>	
VB/VBA	<code>TWXA_RPORT.PoF</code>	
C#	<code>TWXA.RPORT.PoF</code>	

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

表 36 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	Ia7	Ia6	Ia5	Ia4	Ia3	Ia2	Ia1	Ia0

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

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

表 37 `TWXA_PortWrite()` の関数宣言

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

表 38 `TWXA_PortWrite()` の Port 引数に指定する値

言語	値	説明
C/C++	<code>TWXA_POd</code>	Od0-Od7 の出力値を変更します。
C++	<code>TWXA::WPORT::POd</code>	
VB/VBA	<code>TWXA_WPORT.POd</code>	
C#	<code>TWXA.WPORT.POd</code>	
C/C++	<code>TWXA_POe</code>	Oe0-Oe7 の出力値を変更します。
C++	<code>TWXA::WPORT::POe</code>	
VB/VBA	<code>TWXA_WPORT.POe</code>	
C#	<code>TWXA.WPORT.POe</code>	
C/C++	<code>TWXA_POf</code>	Of0-Of2 の出力値を変更します。
C++	<code>TWXA::WPORT::POf</code>	
VB/VBA	<code>TWXA_WPORT.POf</code>	
C#	<code>TWXA.WPORT.POf</code>	

入力と同様に 8 ビット単位でデータを書き込みます。データビットと端子との関係は入力の場合と同様で、“0”を書き込んだビットと対応する端子は“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` バイトのうち“0”となっているビットは影響を受けません。図 51 は H'55 というデータを、`Mask` を H'0F として出力した例です。

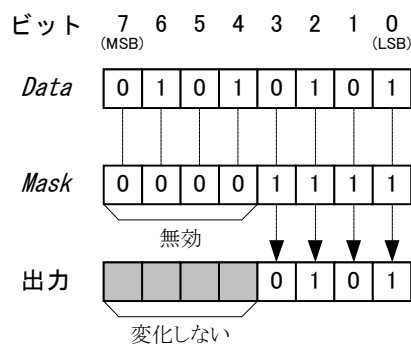


図 51 出力のマスク

リスト 6 デジタル入出力の例(C 言語)

```
BYTE bData;  
  
//Ia0-Ia7 の読み出し  
TWXA_PortRead(hDev, TWXA_P1a, &bData);  
  
//Od7 だけを"ON"にし、Od6-Od0 は変更しない  
TWXA_PortWrite(hDev, TWXA_P0d, 0xff, 0x80);
```

リスト 7 デジタル入出力の例(Visual Basic)

```
Dim bData As Byte  
  
' Ia0-Ia7 の読み出し  
TWXA_PortRead(hDev, TWXA_RPORT.P1a, bData)  
  
' Od7 だけを"ON"にし、Od6-Od0 は変更しない  
TWXA_PortWrite(hDev, TWXA_WPORT.P0d, &HFF, &H80)
```

リスト 8 デジタル入出力の例(C#)

```
byte bData;  
  
//Ia0-Ia7 の読み出し  
TWXA.PortRead(hDev, TWXA.RPORT.P1a, out bData);  
  
//Od7 だけを"ON"にし、Od6-Od0 は変更しない  
TWXA.PortWrite(hDev, TWXA.WPORT.P0d, 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 38 ページを、エラー処理については 82 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。

□ アナログ入出力

製品はアナログ入力用に AD0~AD3、アナログ出力用に DA0~DA1 端子を備えています。

表 39 はアナログ入出力を制御するための関数です。表 40 はアナログ入出力のサンプルプログラムです。

表 39 アナログ入出力で使用する関数

関数名	説明
<i>TWXA_ADRead()</i>	アナログ入力から変換結果を読み出します。
<i>TWXA_PortWrite()</i>	アナログ出力値を設定します。
<i>TWXA_An16ToVolt()</i>	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。
<i>TWXA_An8FromVolt()</i>	電圧値(ボルト単位)から DA コンバータに書き込む値を計算します。

表 40 アナログ入出力のサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	AnalogSample	各アナログ入力端子の入力電圧を表示し、アナログ出力電圧を変更できるプログラムです。
Visual Basic	AnalogSampleVB	
Visual C#	AnalogSampleCS	
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。

アナログ入力値を読み取る

アナログ入力端子の AD 変換結果を読み出すには *TWXA_ADRead()* 関数を使用します。

表 41 *TWXA_ADRead()* の関数宣言

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

AD 変換結果は引数 *pData* に図 52 のように格納されます。入力電圧値と読み出される値の関係は表 42 のようになります。*pData* の値は *TWXA_An16ToVolt()* 関数を使用して電圧値に変換することが可能です。

ビット	31-16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
値	常に 0	AD 変換結果										常に 0					

図 52 AD 変換結果の格納

表 42 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-LSB	65472
2.5	32768
0	0

・LSB = 5 / 1024 [V]
 ・表は理論値を示しています。

リスト 9 アナログ入力の例(C言語)

```
long LData;
double dVolt;

//AD0 の AD 変換結果を読み出し
TWXA_ADRead(hDev, 0, &LData);

//取得値を電圧値に変換
dVolt = TWXA_An16ToVolt(LData, 0);
```

リスト 10 アナログ入力の例(Visual Basic)

```
Dim iData As Integer
Dim dVolt As Double

' AD0 の AD 変換結果を読み出し
TWXA_ADRead(hDev, 0, iData)

' 取得値を電圧値に変換
dVolt = TWXA_An16ToVolt(iData)
```

リスト 11 アナログ入力の例(C#)

```
int iData;
double dVolt;

//AD0 の AD 変換結果を読み出し
TWXA.ADRead(hDev, 0, out iData);

//取得値を電圧値に変換
dVolt = TWXA.An16ToVolt(iData);
```

アナログ出力値を変更する

アナログ出力端子の出力電圧を変更するには、デジタル出力の場合と同じ `TWXA_PortWrite()` 関数(43 ページ、表 37)を使用します。`Port` 引数には DA のチャンネルを示す定数(表 43)を指定します。`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 44 に示します。

`TWXA_An8FromVolt()` 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 43 DA コンバータを示す定数値

言語	値	説明
C/C++	TWXA_DAO	DAO 出力を変更します。
C++	TWXA::WPORT::DAO	
VB/VBA	TWXA.WPORT.DAO	
C#	TWXA.WPORT.DAO	
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 44 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]
255	5-LSB
128	2.5
0	0

•LSB = 5 / 256 [V]

•表は理論値を示しています。

リスト 12 アナログ出力の例(C 言語)

```
double dVolt;  
  
//DAO 出力を約 3.5V に設定  
dVolt = 3.5;  
TWXA_PortWrite(hDev, TWXA_DAO, TWXA_An8FromVolt(&dVolt, 0), 0xff);
```

リスト 13 アナログ出力の例(Visual Basic)

```
Dim dVolt As Double  
  
' DAO 出力を約 3.5V に設定  
dVolt = 3.5  
TWXA_PortWrite(hDev, TWXA_WPORT.DAO, TWXA_An8FromVolt(dVolt))
```

リスト 14 アナログ出力の例(C#)

```
double dVolt;  
  
//DA0 出力を約 3.5V に設定  
dVolt = 3.5;  
TWXA.PortWrite(hDev, TWXA.WPORT.DA0, TWXA.An8FromVolt(ref dVolt), 0xff);
```


□ 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 2 チャンネル、2 相パルスカウントする場合は 1 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 相カウントのどちらにも利用でき、単相パルスカウントの場合最大 4 チャンネル、2 相パルスカウントの場合は最大 2 チャンネル利用可能です。また、2 相信号でカウンタをクリアする 3 相動作も設定可能です。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。本マニュアルでパルスカウンタと表記した場合は、ソフトウェアカウンタのことを指します。

表 45 ハードウェアカウンタとパルスカウンタ(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			
ハードウェアカウンタ	2	1	16	高速、2 相カウント時の分解能が高い	マイコンのハードウェア機能(16 ビットタイマ)を利用
パルスカウンタ	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 46 パルスカウントのサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	PulseCountSample	ハードウェアカウンタとパルスカウンタのサンプルです。各カウンタの設定とカウント値の表示を行います。
Visual Basic	PulseCountSampleVB	
Visual C#	PulseCountSampleCS	
VBA (Excel)	PulseCountSample.xls	簡易データロガーです。定期的に関数値と、前回の値との差分値を記録します。

- PWM 出力も 16 ビットタイマの機能を使用します。ハードウェアカウンタと PWM 出力の使用は、合わせて 3 チャンネルまでです。

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、CLK1 入力を 16 ビットタイマのチャンネル 1 で、CLK2 入力をチャンネル 2 でそれぞれカウントします。カウントエッジは"OFF→ON"、"ON→OFF"、または、両エッジから選択可能です。

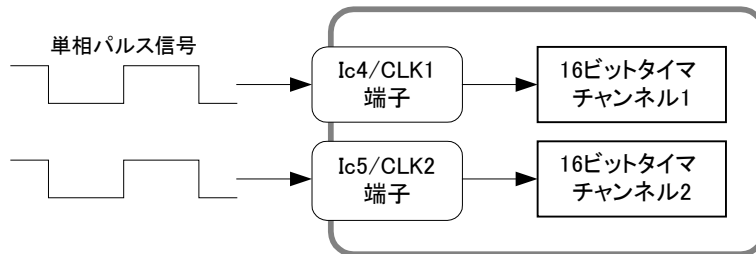


図 53 ハードウェアカウンタによる単相パルスカウント

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、16 ビットタイマのチャンネル 2 を使用します。

接続は CLK1 に B 相信号を CLK2 に A 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 55)。

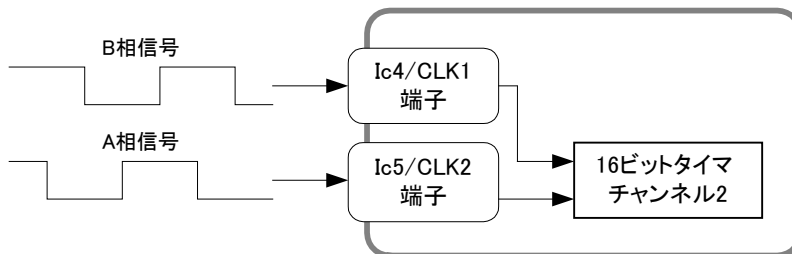


図 54 ハードウェアカウンタによる 2 相パルスカウント

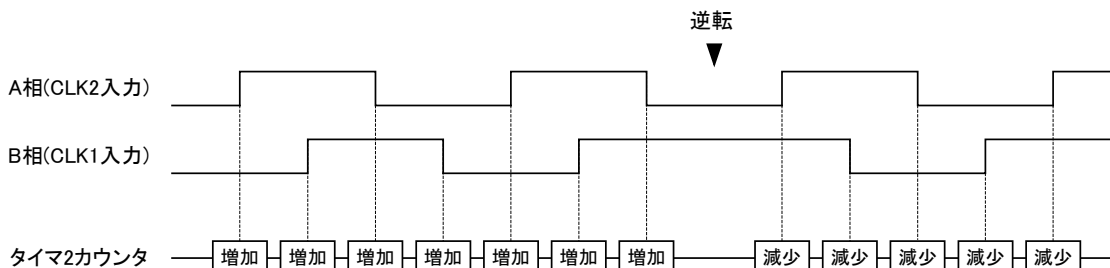


図 55 2 相パルス入力とハードウェアカウンタの増減

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 49 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 47 ハードウェアカウンタで使用する関数

関数名	説明
<code>TWXA_TimerSetMode()</code>	カウントモードを設定します。
<code>TWXA_TimerStart()</code>	カウントを開始します。
<code>TWXA_TimerStop()</code>	カウントを停止します。
<code>TWXA_TimerReadCnt()</code>	カウンタ値を読み出します。
<code>TWXA_TimerSetCnt()</code>	カウンタ値をセットします。主にカウンタクリアに使用します。

表 48 `TWXA_TimerSetMode()` の関数宣言

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

表 49 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++	<code>TWXA_TIMER_OFF_TO_ON</code>	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。1、2 チャンネルで指定可能です。
C++	<code>TWXA::TIMER_MODE::COUNT_OFF_TO_ON</code>	
VB/VBA	<code>TWXA_TIMER_MODE.COUNT_OFF_TO_ON</code>	
C#	<code>TWXA.TIMER_MODE.COUNT_OFF_TO_ON</code>	
C/C++	<code>TWXA_TIMER_ON_TO_OFF</code>	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。1、2 チャンネルで指定可能です。
C++	<code>TWXA::TIMER_MODE::COUNT_ON_TO_OFF</code>	
VB/VBA	<code>TWXA_TIMER_MODE.COUNT_ON_TO_OFF</code>	
C#	<code>TWXA.TIMER_MODE.COUNT_ON_TO_OFF</code>	
C/C++	<code>TWXA_TIMER_BOTH</code>	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に変化したときにカウントします。1、2 チャンネルで指定可能です。
C++	<code>TWXA::TIMER_MODE::COUNT_BOTH</code>	
VB/VBA	<code>TWXA_TIMER_MODE.COUNT_BOTH</code>	
C#	<code>TWXA.TIMER_MODE.COUNT_BOTH</code>	
C/C++	<code>TWXA_TIMER_2PHASE</code>	90° 位相差の A 相、B 相の 2 相信号をカウントします。2 チャンネルのみ指定可能です。
C++	<code>TWXA::TIMER_MODE::COUNT_2PHASE</code>	
VB/VBA	<code>TWXA_TIMER_MODE.COUNT_2PHASE</code>	
C#	<code>TWXA.TIMER_MODE.COUNT_2PHASE</code>	

リスト 15 ハードウェアカウンタの使用例(C 言語)

```
WORD wCnt;

//チャンネル1でOFFからON時にカウント
TWXA_TimerSetMode(hDev, 1, TWXA_TIMER_OFF_TO_ON);

//タイマ1のカウントをスタート
TWXA_TimerStart(hDev, TWXA_TIMER_BIT1);

//タイマ1のカウント値を符号なし整数として読み出し
TWXA_TimerReadCnt(hDev, 1, (short*)&wCnt);
```

リスト 16 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1でOFFからON時にカウント
TWXA_TimerSetMode(hDev, 1, TWXA_TIMER_MODE.COUNT_OFF_TO_ON)

'タイマ1のカウントをスタート
TWXA_TimerStart(hDev, TWXA_TIMER_BITS.TIMER1)

'タイマ1のカウント値を符号なし整数として読み出し
TWXA_TimerReadCnt(hDev, 1, wCnt)
```

リスト 17 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1でOFFからON時にカウント
TWXA_TimerSetMode hDev, 1, TWXA_TIMER_MODE.COUNT_OFF_TO_ON

'タイマ1のカウントをスタート
TWXA_TimerStart hDev, TWXA_TIMER_BITS.TIMER1

'タイマ1のカウント値を読み出し、0~65535の範囲の値に変換
TWXA_TimerReadCnt hDev, 1, iCnt
LCnt = TWXA_ToINT32(iCnt)
```

リスト 18 ハードウェアカウンタの使用例(C#)

```
ushort usCnt;

//チャンネル1でOFFからON時にカウント
TWXA.TimerSetMode(hDev, 1, TWXA.TIMER_MODE.COUNT_OFF_TO_ON);

//タイマ1のカウントをスタート
TWXA.TimerStart(hDev, TWXA.TIMER_BITS.TIMER1);

//タイマ1のカウント値を符号なし整数として読み出し
TWXA.TimerReadCnt(hDev, 1, out usCnt);
```

パルスカウンタ(ソフトウェアカウンタ)による単相パルスカウント

パルスカウンタによる単相カウントの場合、PC0~PC3 入力をそれぞれチャンネル 0~3 でカウントします。カウントエッジは"OFF→ON"、"ON→OFF"から選択可能です。

パルスカウンタ(ソフトウェアカウンタ)による 2 相パルスカウント

パルスカウンタにより 90° 位相差 2 相パルスをカウントする場合、チャンネル 0 と 1 の組み合わせ、または、チャンネル 2 と 3 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、PC0、PC1 の両方の端子に A 相信号を接続し、Ib6 端子に B 相信号を接続します。カウント値はチャンネル 0 と 1 の合計値になります。

チャンネル 2 と 3 の組み合わせでカウントする場合には、PC2、PC3 の両方の端子に A 相信号を接続し、Ib7 端子に B 相信号を接続します。カウント値はチャンネル 2 と 3 の合計値になります。

どちらの組み合わせを使用した場合も、インクリメンタル方式のロータリーエンコーダを接続した場合、1 回転あたりのカウント数は出力パルス数の 2 倍になります(図 57)。

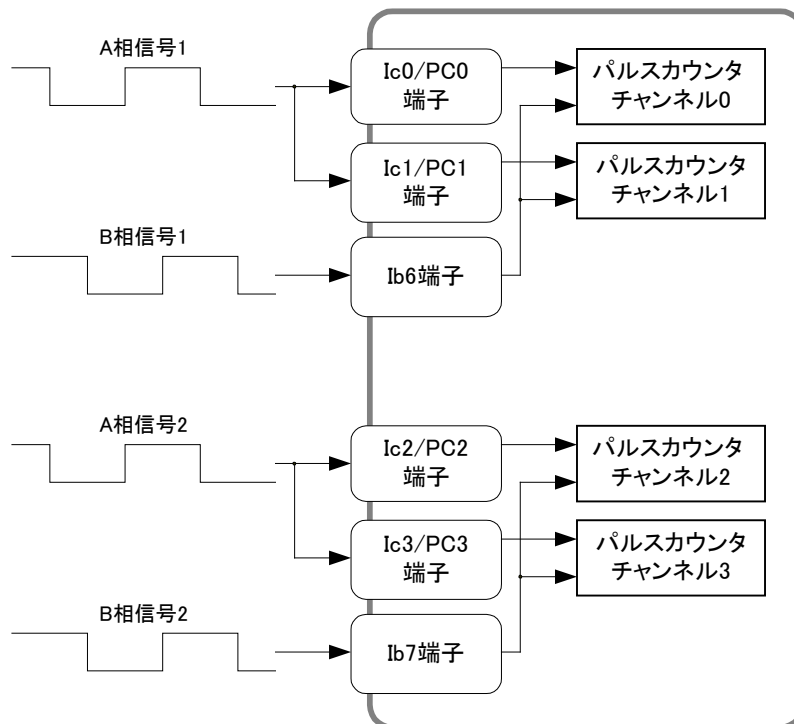


図 56 ソフトウェアカウンタによる 2 相パルスカウント

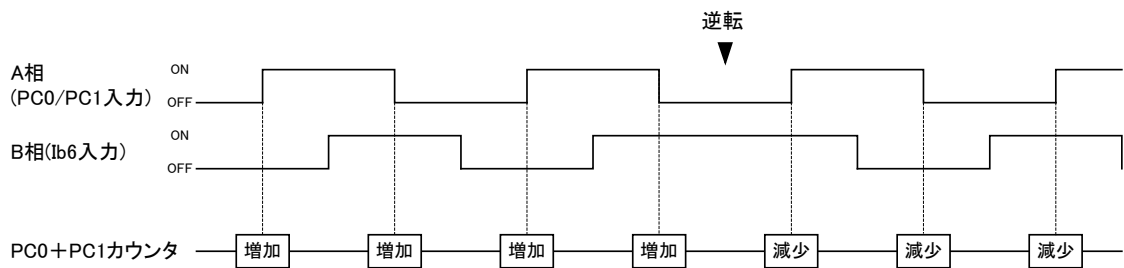


図 57 2相パルス入力とソフトウェアカウンタの増減

パルスカウンタ(ソフトウェアカウンタ)による3相パルスカウント

基本的にはパルスカウンタ2、3チャンネルを使用した2相パルスカウントと同様の動作をしますが、パルスカウンタ0への入力で、パルスカウンタ2、3のカウンタ値がクリアされます。PC0にZ相信号を接続すると1回転毎にカウンタ値がクリアされ、パルスカウンタ0の値が増加します。

パルスカウンタ(ソフトウェアカウンタ)の使用方法

ソフトウェアカウンタを使用するには、まず `TWXA_PCSetMode()` 関数(表 51)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 52 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 53 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 54)を使用します。1チャンネルずつ読み出すこともできますが、`ChBits` 引数に `TWXA_PC_ALL` などの全てのチャンネルを示す定数を指定すると0~3チャンネルまで全てのカウンタ値を読み出すことができます。その場合は、`pCnt` 引数として4チャンネル分(16バイト)の領域を確保するようにしてください。

表 50 ソフトウェアカウンタで使用する関数

関数名	説明
<code>TWXA_PCSetMode()</code>	カウントモードを設定します。
<code>TWXA_PCStart()</code>	カウントを開始します。
<code>TWXA_PCStop()</code>	カウントを停止します。
<code>TWXA_PCReadCnt()</code>	カウンタ値を読み出します。
<code>TWXA_PCSetCnt()</code>	カウンタ値をセットします。主にカウンタクリアに使用します。

表 51 `TWXA_PCSetMode()` の関数宣言

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

表 52 ソフトウェアカウンタ操作関数の ChBits 引数に指定する値

言語	値	説明
C/C++	TWXA_PC0	パルスカウンタ 0 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC0	
VB/VBA	TWXA_PC_BITS.PC0	
C#	TWXA.PC_BITS.PC0	
C/C++	TWXA_PC1	パルスカウンタ 1 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC1	
VB/VBA	TWXA_PC_BITS.PC1	
C#	TWXA.PC_BITS.PC1	
C/C++	TWXA_PC2	パルスカウンタ 2 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC2	
VB/VBA	TWXA_PC_BITS.PC2	
C#	TWXA.PC_BITS.PC2	
C/C++	TWXA_PC3	パルスカウンタ 3 の設定や読み出しなどで指定します。
C++	TWXA::PC_BITS::PC3	
VB/VBA	TWXA_PC_BITS.PC3	
C#	TWXA.PC_BITS.PC3	
C/C++	TWXA_PC0_PC1	パルスカウンタ 0 と 1 の設定や読み出しなどで指定します。 読み出しの場合はカウンタ 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	パルスカウンタ 2 と 3 の設定や読み出しなどで指定します。 読み出しの場合はカウンタ 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_PC_ALL	全てのチャンネルを同じ動作設定にする場合や、全てのカウンタ値を読み出す場合に指定します。
C++	TWXA::PC_BITS::PC_ALL	
VB/VBA	TWXA_PC_BITS.PC_ALL	
C#	TWXA.PC_BITS.PC_ALL	

表 53 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_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	PC2 と PC3 で 2 相信号のカウントを行い、PC0 でカウンタをクリアします。ChBits の値は無視されます。
C++	TWXA::PC_MODE::COUNT_3PHASE	
VB/VBA	TWXA_PC_MODE.COUNT_3PHASE	
C#	TWXA.PC_MODE.COUNT_3PHASE	

表 54 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 19 ソフトウェアカウンタによるパルスカウンタの例(C言語)

```

long LCnt[4];
long L2Phase;

//パルスカウンタ 0,1 を単相カウントに設定
TWXA_PCSetMode(hDev, TWXA_PC0_PC1, TWXA_PC_OFF_TO_ON);

//パルスカウンタ 2,3 を2相カウントに設定
TWXA_PCSetMode(hDev, TWXA_PC2_PC3, 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
L2Phase = LCnt[2] + LCnt[3];

```

リスト 20 ソフトウェアカウンタによるパルスカウンタの例(Visual Basic)

```
Dim LCnt(3) As Integer
Dim L2Phase As Integer

'パルスカウンタ 0,1 を単相カウンタに設定
TWXA_PCSetMode(hDev, TWXA_PC_BITS.PC0_PC1, TWXA_PC_MODE.COUNT_OFF_TO_ON)

'パルスカウンタ 2,3 を2相カウンタに設定
TWXA_PCSetMode(hDev, TWXA_PC_BITS.PC2_PC3, TWXA_PC_MODE.COUNT_2PHASE)

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

'全てのチャンネルのカウンタ値を読み出し
TWXA_PCReadCnt(hDev, TWXA_PC_BITS.PC_ALL, LCnt(0))

'2相カウンタの結果は2チャンネルのカウンタ値の和
L2Phase = LCnt(2) + LCnt(3)
```

リスト 21 ソフトウェアカウンタによるパルスカウンタの例(C#)

```
int[] iCnt = new int[4];
int i2Phase;

//パルスカウンタ 0,1 を単相カウンタに設定
TWXA.PCSetMode(hDev, TWXA.PC_BITS.PC0_PC1, TWXA.PC_MODE.COUNT_OFF_TO_ON);

//パルスカウンタ 2,3 を2相カウンタに設定
TWXA.PCSetMode(hDev, TWXA.PC_BITS.PC2_PC3, TWXA.PC_MODE.COUNT_2PHASE);

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

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

//2相カウンタの結果は2チャンネルのカウンタ値の和
i2Phase = iCnt[2] + iCnt[3];
```

□ PWM 出力

製品では最大 3 チャンネルの PWM 出力が可能です。通常、PWM のパルスタイミングは 25MHz の内部クロックをプリスケアラ⁹と 16 ビットタイマという内蔵カウンタで分周することで生成されます。この場合、出力できる周波数範囲は約 48Hz～1MHz までの範囲です。

PWM 信号生成のための基準クロックは外部から CLK1 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 58)。

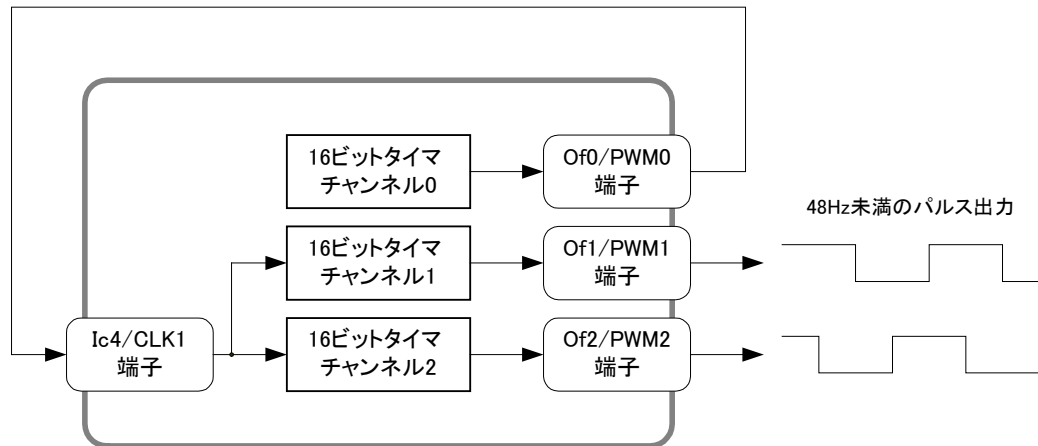


図 58 他チャンネルの出力を基準クロックとして利用

- ハードウェアカウンタも 16ビットタイマの機能を使用します。ハードウェアカウンタとPWM出力の使用は、合わせて 3 チャンネルまでです。

表 55 PWM 出力で使用する関数

関数名	説明
<i>TWXA_TimerSetMode()</i>	16ビットタイマを PWM モードへの設定、PWM モードの解除を行います。
<i>TWXA_TimerSetPwm()</i>	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。
<i>TWXA_TimerSetPwmExt()</i>	外部クロックを用いた場合のパルス設定を行います。
<i>TWXA_TimerStart()</i>	パルス出力動作を開始します。
<i>TWXA_TimerStop()</i>	パルス出力動作を停止します。
<i>TWXA_SetNumOfPulse()</i>	出力パルス数を設定します。
<i>TWXA_ReadNumOfPulse()</i>	残りの出力パルス数を読み出します。
<i>TWXA_TimerReadStatus()</i>	パルス出力中かどうか調べます。
<i>TWXA_TimerSetLevel()</i>	停止中に PWM 出力の状態を設定します。

表 56 PWM 出力のサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	PwmSample	3 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	
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。

⁹ 25MHz のクロックを 1/2、1/4、または、1/8 に分周します。

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 57)、外部クロックを使用したパルスの設定には `TWXA_TimerSetPwmExt()` 関数(表 58)を使用します。

表 57 `TWXA_TimerSetPwm()` の関数宣言

言語	関数宣言
C/C++	<code>TW_STATUS TWXA_TimerSetPwm(TW_HANDLE hDev, long Ch, double *pFrequency, double *pDuty, double *pPhase)</code>
VB	<code>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</code>
VBA	<code>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</code>
C#	<code>STATUS TimerSetPwm(System.IntPtr hDev, int Ch, ref double pFrequency, ref double pDuty, ref double pPhase)</code>

表 58 `TWXA_TimerSetPwmExt()` の関数宣言

言語	関数宣言
C/C++	<code>TW_STATUS TWXA_TimerSetPwmExt(TW_HANDLE hDev, long Ch, double dClkFreq, double *pFrequency, double *pDuty, double *pPhase)</code>
VB	<code>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</code>
VBA	<code>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</code>
C#	<code>STATUS TimerSetPwmExt(System.IntPtr hDev, int Ch, double dClkFreq, ref double pFrequency, ref double pDuty, ref double pPhase)</code>

`pFrequency` 引数はパルスの繰り返し周波数を Hz 単位で入力します。`pDuty` 引数は ON デューティを 0~1.0 の範囲で入力します。`pPhase` 引数は出力開始時の位相を 0~1.0 の範囲で入力します。`TWXA_TimerSetPwmExt()` 関数の `dClkFreq` 引数は CLK1 に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 59 に示します。

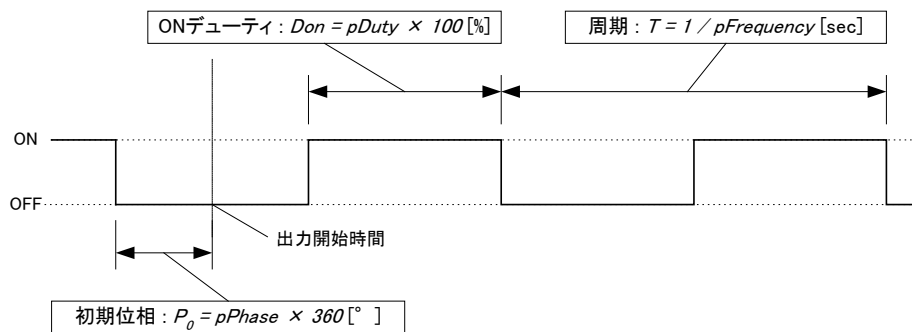


図 59 パラメータと出力パルスの関係

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(51 ページ、表 48)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 59 を参照してください。

表 59 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_TimerSetLevel()* 関数で PWM 端子の初期状態を変更することが可能です(16 ビットタイマチャンネルを PWM モードに設定すると対応する PWM 出力端子は、デジタル出力としてコントロールすることができなくなります)。
4. 必要であれば *TWXA_TimerSetNumOfPulse()* 関数で出力パルス数を設定します。
5. *TWXA_TimerStart()* 関数でパルス出力を開始します。
6. 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。
7. *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。
8. パルス出力を停止する場合は *TWXA_TimerStop()* 関数を使用します。

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

1. *TWXA_PortWrite()* 関数で *POf* に 0 を書き込みます(43 ページを参照)。これにより PWM 端子の機能をデジタル出力に戻したときに出力値が自動的に"OFF"になります。
2. *TWXA_TimerSetMode()* 関数で PWM モードを解除します。この時点で端子の機能が PWM からデジタル出力に切り替わり、出力が"OFF"になります。また、タイマの動作も停止します。
3. *TWXA_TimerSetLevel()* 関数で停止したタイマ出力を"OFF"にします。これを行わないと次回の PWM 出力時に意図しないパルスが出力される場合があります。

- 内部クロックによる動作中に 400Hz 以下の範囲の出力周波数変更を行うと、パルスタイミングの誤差を生じやすくなります。
- 出力周波数が 100kHz 以上の範囲でのデューティや周波数の変更を行うと、パルスタイミングの誤差や、パルスの抜け¹⁰を生じやすくなります。
- `TWXA_TimerSetNumOfPulse()` 関数によってパルス数を指定する場合は、“OFF”期間が 50 μ sec 以上になるようにしてください。“OFF”期間が短すぎると、停止処理にかかる時間により指定のパルス数を越えてしまったり、“ON”期間に停止したりする場合があります。

リスト 22 PWM 出力の例(C 言語)

```

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

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

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

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

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

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

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

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

```

¹⁰ 出力パルスに抜けが生じると、1 サイクル以上"ON"状態または"OFF"状態となります。

リスト 23 PWM 出力の例(Visual Basic)

```
Dim dFreq As Double
Dim dDuty As Double

dFreq = 9500 ' 周波数 = 9.5kHz
dDuty = 0.6 ' デューティ = 60%

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

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

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

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

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

リスト 24 PWM 出力の例(C#)

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

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

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

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

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

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

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

□ シリアルポート

シリアルポートは最大 2 チャンネル使用可能です。シリアル 0 は自由に使用することができます。

シリアル 1 はデフォルトの状態ではユーザーファームのデバッグ用ポート、または、標準入出力ポートとして機能します。ユーザーファームを利用しない場合は、`TWXA_SCISetMode()` をシリアル 1 に対して呼び出すことで、使用可能な状態となります。

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

また、受信データを改行コードなどで分割して読み出したい場合には、デリミタコードを設定しておくことができます。デリミタコードを設定しておく、`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 60 にシリアルポート制御で使用する関数をあげます。

表 60 シリアルポート制御で使用する関数

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

表 61 シリアルポート制御のサンプルプログラム

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

シリアルポートの設定

表 62 は `TWXA_SCISetMode()` 関数の宣言です。`Mode` 引数には表 63 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 64 のボーレートを入力します。

表 62 `TWXA_SCISetMode()` の関数宣言

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

表 63 `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++	<code>TWXA_SCI_DATA8</code>	データ長を 8 ビットにします(デフォルト)。
	C++	<code>TWXA::SCI_MODE::DATA8</code>	
	VB/VBA	<code>TWXA_SCI_MODE.DATA8</code>	
	C#	<code>TWXA.SCI_MODE.DATA8</code>	
	C/C++	<code>TWXA_SCI_DATA7</code>	データ長を 7 ビットにします。
	C++	<code>TWXA::SCI_MODE::DATA7</code>	
	VB/VBA	<code>TWXA_SCI_MODE.DATA7</code>	
	C#	<code>TWXA.SCI_MODE.DATA7</code>	
パリティ	C/C++	<code>TWXA_SCI_NOPARITY</code>	パリティビットを使用しません(デフォルト)。
	C++	<code>TWXA::SCI_MODE::NO_PARITY</code>	
	VB/VBA	<code>TWXA_SCI_MODE.NO_PARITY</code>	
	C#	<code>TWXA.SCI_MODE.NO_PARITY</code>	
	C/C++	<code>TWXA_SCI_EVEN</code>	偶数パリティを使用します。
	C++	<code>TWXA::SCI_MODE::EVEN</code>	
	VB/VBA	<code>TWXA_SCI_MODE.EVEN</code>	
	C#	<code>TWXA.SCI_MODE.EVEN</code>	
	C/C++	<code>TWXA_SCI_ODD</code>	奇数パリティを使用します。
	C++	<code>TWXA::SCI_MODE::ODD</code>	
	VB/VBA	<code>TWXA_SCI_MODE.ODD</code>	
	C#	<code>TWXA.SCI_MODE.ODD</code>	
ストップビット	C/C++	<code>TWXA_SCI_STOP1</code>	ストップビットを 1 ビットとします(デフォルト)。
	C++	<code>TWXA::SCI_MODE::STOP1</code>	
	VB/VBA	<code>TWXA_SCI_MODE.STOP1</code>	
	C#	<code>TWXA.SCI_MODE.STOP1</code>	
	C/C++	<code>TWXA_SCI_STOP2</code>	ストップビットを 2 ビットとします。
	C++	<code>TWXA::SCI_MODE::STOP2</code>	
	VB/VBA	<code>TWXA_SCI_MODE.STOP2</code>	
	C#	<code>TWXA.SCI_MODE.STOP2</code>	

表 64 TWXA_SCISetMode() の Baud 引数に指定する値

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

シリアルポートの使用手順

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

リスト 25 シリアルポートの使用例(C言語)

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

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

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

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

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

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

リスト 26 シリアルポートの使用例(Visual Basic)

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

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

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

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

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

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

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

リスト 27 シリアルポートの使用例(VBA)

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

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

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

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

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

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

If L = 0 Then Exit Do

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

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

リスト 28 シリアルポートの使用例(C#)

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

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

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

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

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

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

□ ハードウェアイベントの監視

製品ではパルスカウンタ(ソフトウェアカウンタ)のカウント値や 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。また、ユーザーファームを作成した場合は独自のイベントをアプリケーションに通知することもできます。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 65 はハードウェアイベントのサンプルプログラムです。

表 65 ハードウェアイベントのサンプルプログラム

開発環境	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	EventSample	パルスカウンタの値が変化した場合、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	

- 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 66 の `TWXA_SetHwEvent()` 関数を使用します。この関数には引数として `TWXA_HW_EVENT` 構造体(表 67)を渡します。

表 66 `TWXA_SetHwEvent()` の関数宣言

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

表 67 TWXA_HW_EVENT 構造体の宣言

言語	宣言
C/C++	<pre> typedef struct tagHwEvent{ HWND hRecvWindow; DWORD idRecvThread; LPVOID lpRsv; UINT Message; DWORD EventBits; long PCCnt[4]; long PCCmp[4]; long ADVal[4]; short ADCmp[4]; } TWXA_HW_EVENT; </pre>
VB	<pre> <StructLayout(LayoutKind.Sequential)> _ Public Structure TWXA_HW_EVENT Public hRecvWindow As System.IntPtr Public idRecvThread As Integer Public lpRsv As System.IntPtr Public Message As Integer Public EventBits As Integer <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _ Public PCCnt() As Integer <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _ Public PCCmp() As Integer <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _ Public ADVal() As Integer <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> _ Public ADCmp() As Short Public Sub Initialize() ReDim PCCnt(3) ReDim PCCmp(3) ReDim ADVal(3) ReDim ADCmp(3) End Sub End Structure </pre>

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

hRecvWindow

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

idRecvThread

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

lpRsv

予約領域です。0 にしてください。

Message

指定のイベントが発生したときに通知されるメッセージ番号です。通常は H'8000~H' BFFF の範囲の任意の値を指定します。パルスカウンタ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

通知を受けたいイベントをビットで指定します(表 68)。複数のビットを指定することもできます。

PCCnt

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

PCCmp

パルスカウンタの値と閾値の比較方法を指定します(表 69)。配列のインデックスがパルスカウンタのチャンネルと対応しています。

ADVal

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 68 **EventBits** の指定

ビット	監視するイベント
TWXA_EVENT_PC0 (H' 00000001)	パルスカウンタ 0 を監視
TWXA_EVENT_PC1 (H' 00000002)	パルスカウンタ 1 を監視
TWXA_EVENT_PC2 (H' 00000004)	パルスカウンタ 2 を監視
TWXA_EVENT_PC3 (H' 00000008)	パルスカウンタ 3 を監視
TWXA_EVENT_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_USER (H' 80000000)	ユーザー定義(ユーザーファームから通信を行う場合)
TWXA_EVENT_PC_ALL (H' 0000000F)	パルスカウンタ全チャンネルを監視
TWXA_EVENT_AD_ALL (H' 000000F0)	アナログ入力全チャンネルを監視

パルスカウンタ入力を監視する

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

1. Visual Basic または C# を利用する場合、*TWXA_HW_EVENT* 構造体の *Initialize()* メソッドを呼びます。
2. *TWXA_HW_EVENT* 構造体の *hRecvWindow* にウィンドウのハンドルを指定します。ウィンドウを持たないアプリケーションの場合、*idRecvThread* にスレッド ID を指定します。また、イベント発生時に受け取るメッセージ番号を *Message* に指定します。
3. *TWXA_HW_EVENT* 構造体の *EventBits* に監視するパルスカウンタチャンネルを指定します(表 68 参照)。
4. *TWXA_HW_EVENT* 構造体の *PCCnt* にカウンタ値と比較する閾値を指定します。配列のインデックスはチャンネルを示します。例えばパルスカウンタ 2 を監視する場合は、*PCCmp[2]* に閾値を設定します。
5. *TWXA_HW_EVENT* 構造体の *PCCmp* に比較方法を指定します。*PCCmp* に指定する値と、イベント発生条件を 表 69 に示します。

表 69 **PCCmp** の設定値とハードウェアイベントの発生条件

PCCmp[x]の設定	ハードウェアイベントの発生条件
TWXA_CMP_GE (H' 7FFFFFFF)	指定チャンネル(x)のカウンタ値が PCCnt[x]以上になった場合
0	指定チャンネル(x)のカウンタ値が変化した場合
TWXA_CMP_LE (H' 80000000)	指定チャンネル(x)のカウンタ値が 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_SetHwEvent()` 関数を呼び出します。
7. 使用するパルスカウンタの設定を行い、カウント動作を開始します(49 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 70 パルスカウンタイベントによるメッセージのパラメータ

項目	ハードウェアイベントの発生条件
Msg	TWXA_HW_EVENT 構造体の Message に指定した値
wParam (WParam)	イベントが発生したカウンタチャンネルを示すビット(表 68)
lParam (LParam)	イベント発生時のカウンタの値

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

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を持たせることができます。例えば、アナログ入力値がある閾値電圧 V_{TH} 以上になった場合にイベントが発生するように設定した場合、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 60)。ここでは V_{HYST} をヒステリシス、または、ヒステリシス電圧と呼びます。

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送信される場合があります。

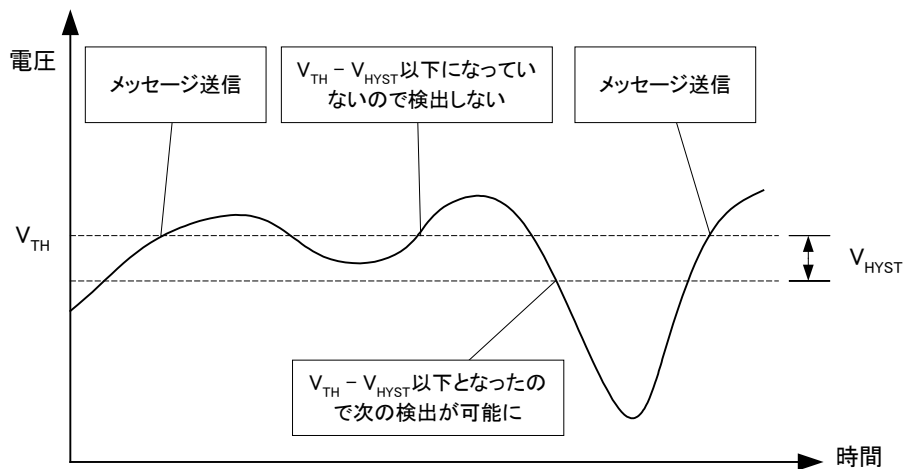


図 60 ヒステリシスが設定されている場合の動作

1. Visual Basic または C# を利用する場合、*TWXA_HW_EVENT* 構造体の *Initialize()* メソッドを呼びます。
2. *TWXA_HW_EVENT* 構造体の *hRecvWindow* にウィンドウのハンドルを指定します。ウィンドウを持たないアプリケーションの場合、*idRecvThread* にスレッド ID を指定します。また、イベント発生時に受け取るメッセージ番号を *Message* に指定します。
3. *TWXA_HW_EVENT* 構造体の *EventBits* に監視するアナログ入力チャンネルを指定します(表 68 参照)。
4. *TWXA_HW_EVENT* 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。*ADVal* の値は 45 ページ 図 52 の形式の変換値と比較されます。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 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} を求めるには下の式を使用します。

$$C_{TH} \doteq (V_{TH} [V] / 5 [V]) \times 65536 \quad \dots \text{式 1}$$

5. *TWXA_HW_EVENT* 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 71 に *ADCmp* に指定する値と、イベント発生条件、再度イベント発生が可能になる条件を示します。

表 71 *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	再度イベント発生可能となる条件
0 以上の場合	指定チャンネル(x)の AD 変換値が <i>ADVal[x]</i> 以上になった場合	指定チャンネル(x)の AD 変換値が <i>ADVal[x] - ADCmp[x]</i> 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 <i>ADVal[x]</i> 以下になった場合	指定チャンネル(x)の AD 変換値が <i>ADVal[x] - ADCmp[x]</i> 以上になった場合

6. パラメータを設定した構造体を引数として *TWXA_SetHwEvent()* 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。
7. 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 72 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	<i>TWXA_HW_EVENT</i> 構造体の <i>Message</i> 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 68)
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

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

- $0 < ADVal[x] - ADCmp[x] < 65535$ となるようにしてください。

□ ユーザ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。

ユーザステータスレジスタはデバイス内の 1 バイトのメモリで、デバイスの起動時や再初期化のときには必ず 0 にクリアされます。ユーザ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、パルス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タ値を表示したい」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パルスカウンタを設定した時点で、ユーザ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、単にカウンタ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 10K バイトのメモリ空間です。ユーザステータスレジスタでは保存できない比較的大きな設定情報などを記憶することができます。この領域の値は起動時に不定となり、自動的にクリアされることもありませんのでユーザステータスレジスタと組み合わせ使用してください。ユーザーメモリのアドレスはデバイス上の H'FFBF20～H'FFE71F の範囲です。

表 73 ユーザ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
<code>TWXA_PortWrite()</code>	ユーザステータスレジスタにデータを書き込みます。
<code>TWXA_PortRead()</code>	ユーザステータスレジスタからデータを読み出します。
<code>TWXA_PortBWrite()</code>	ユーザーメモリにデータを書き込みます。
<code>TWXA_PortBRead()</code>	ユーザーメモリからデータを読み出します。

表 74 ユーザ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	プロジェクト名またはファイル名	説明
Visual C++ (MFC)	PulseCountSample	カウントモードの記録にユーザステータスレジスタを利用しています。
Visual Basic	PulseCountSampleVB	
Visual C#	PulseCountSampleCS	

ユーザ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 75 の値を指定してください。

表 75 ユーザーステータスレジスタを指定する定数

言語	値	説明
C/C++	TWXA_USER_STATUS	ユーザーステータスレジスタの読み書き時に指定します。
C++	TWXA::WPORT::USER_STATUS	
VB/VBA	TWXA.WPORT.USER_STATUS	
C#	TWXA.WPORT.USER_STATUS	

ユーザーメモリの操作方法

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

表 76 *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 77 *TWXA_PortBRead()* の関数宣言

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

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

□ フラッシュメモリの利用

製品にはフラッシュメモリが内蔵されています。フラッシュメモリは電源を切っても記録した情報が保存される不揮発性のメモリ空間で、製品が動作するためのファームウェアもこの領域に書き込まれています。図 61 はフラッシュメモリ領域を詳しく示した図です。

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

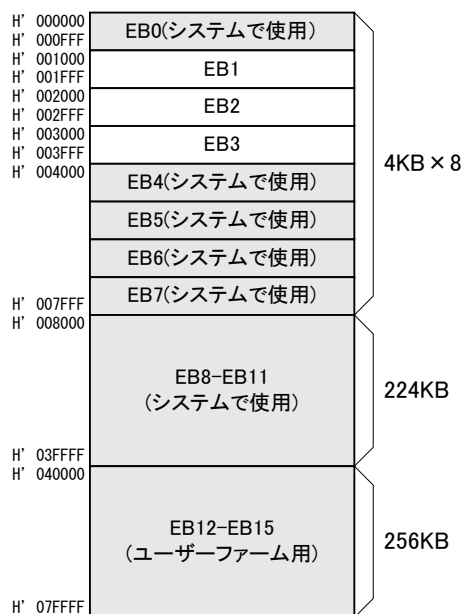


図 61 フラッシュメモリマップ

表 78 フラッシュメモリの操作に使用する関数

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

表 79 フラッシュメモリ操作のサンプルプログラム

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

フラッシュメモリへの書き込み操作は特殊で、通常のメモリのように1バイト単位でデータを書き込むことはできません。書き込みを行う領域には、まず消去の操作を行います。消去の単位は図 61 に示

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

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

また、フラッシュメモリの消去／書き込みを行う際には、予めフラッシュメモリを制御するためのファームウェアをデバイスにダウンロードする必要があります。このファームウェアはユーザーメモリ(76 ページ参照)にダウンロードされますので、ユーザーメモリは一時的に使用できなくなり、内部のデータも破壊されてしまいますので注意してください。

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

- フラッシュメモリの書換え可能回数の目安は 100 回、データ保持年数は 10 年です。
- 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。また、フラッシュメモリの消去／書き込みを行う場合はユーザーファームを動作させることはできませんので、ディップスイッチは 1 番、2 番とも“OFF”の状態デバイス起動し、操作を行ってください。

フラッシュメモリの消去方法

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

表 80 `TWXA_FlashEraseBlk()` の関数宣言

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

フラッシュメモリへの書き込み方法

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

表 81 TWXA_FlashWrite() の関数宣言

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

リスト 29 フラッシュメモリの使用例(C 言語)

```

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

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

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

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

//読み出し
TWXA_PortBRead(hDev, 0x1000, cRead, 128);
OutputDebugStringA(cRead);

```

リスト 30 フラッシュメモリの使用例(Visual Basic)

```

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

strWrite.Length = 128

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

'ブロック 1 を消去
TWXA_FlashEraseBlk(hDev, 1)

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

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

```

リスト 31 フラッシュメモリの使用例(VBA)

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

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

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

'ブロック1を消去
TWXA_FlashEraseBlk hDev, 1

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

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

リスト 32 フラッシュメモリの使用例(C#)

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

strWrite.Length = 128;

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

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

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

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

□ エラー処理

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

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

リスト 33 エラー処理の例(C言語)

```
TW_STATUS ret;

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

リスト 34 エラー処理の例(C++/MFC)

```
CString str;
TW_STATUS ret;

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

リスト 35 エラー処理の例(Visual Basic)

```
Dim ret As Integer

ret = TWXA_PortWrite(hDev, TWXA_WPORT.POd, &H0)

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

リスト 36 エラー処理の例(VBA)

```
Dim ret As Long

ret = TWXA_PortWrite(hDev, TWXA_WPORT.P0d, &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 37 エラー処理の例(C#)

```
TWXA.STATUS ret;

ret = TWXA.PortWrite(hDev, TWXA.WPORT.P0d, 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 62 は参考として各入力ポートに対する `TWXA_PortRead()` 関数呼び出しを 1000 回ずつ行い、関数実行に要した時間をプロットしたものです。

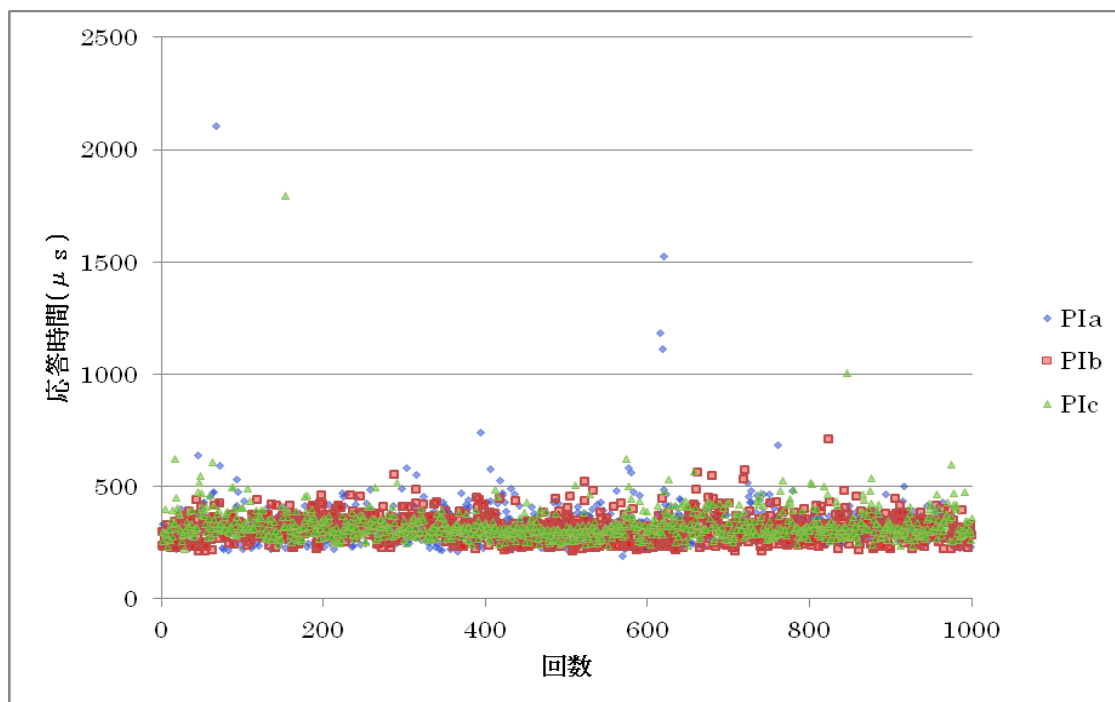


図 62 `TWXA_PortRead()` 関数の応答時間

リアルタイム処理

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

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

保証期間

本製品の保証期間は、お買い上げ日より1年間です。保証期間中の故障につきましては、無償修理または代品との交換で対応させていただきます。ただし、以下の場合に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でしたが、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
2011年12月	初	
2012年2月	2	<ul style="list-style-type: none"> ・寸法、DA コンバータ仕様の誤りを修正 ・設定ツールのバージョンアップに対応 ・誤記を修正
2013年3月	3	<ul style="list-style-type: none"> ・『YCH8/YSH8』の発売元変更に対応 ・絶縁入力の入力オン電圧仕様を変更 ・対応 OS に Windows 8 を追加 ・誤記を修正
2014年11月	4	<ul style="list-style-type: none"> ・開発言語 C#に関する記述を追加 ・対応 OS に Windows 8.1 を追加 ・誤記を修正
2018年4月	5	<ul style="list-style-type: none"> ・対応 OS に Windows 10 を追加 ・ドライバファイルの更新に伴いインストール手順を修正
2019年8月	6	<ul style="list-style-type: none"> ・誤記を修正