LANX-I16/LANX-I16P
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

このたびは多機能 I/O ユニット『LANX-I16』/『LANX-I16P』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。
- 本製品のネットワークコネクタに、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。

2. 製品概要

□ 特徴

『LANX-I16』/『LANX-I16P』(以下、製品またはデバイス)は多機能 I/O ユニットです。ネットワーク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、シリアル通信などの機能を制御できます。

- **デジタル I/O** – デジタル I/O として絶縁入力端子、出力端子をそれぞれ 16 点ずつ、さらに非絶縁出力端子を 8 点備えています。絶縁出力端子は 1 点あたり 150mA までの電流を駆動できます。入力端子は信号元がシンクでもソースでも駆動できる交流入力タイプです(『LANX-I16』)。
- **32 ビットパルスカウンタ** – 4 チャンネルの 32 ビットパルスカウンタを内蔵しています。デジタル I/O と同じく絶縁入力です。ロータリーエンコーダなどの 2 相パルス出力をカウントするモードも備えています。
- **AD コンバータ** - 非絶縁入力の 10 ビット AD コンバータを 4 チャンネル使用可能です。最大約 370KS/sec[1]で変換可能で、4 チャンネルのうち 2 チャンネルは入力範囲としてユニポーラ(0～+5V)、バイポーラ(-2.5V～+2.5V)を選択できます。
- **DA コンバータ** – 非絶縁出力の 8 ビッド DA コンバータを 2 チャンネル使用可能です。変換時間は 10 μ sec で、出力範囲としてユニポーラ(0～+5V)、バイポーラ(-2.5V～+2.5V)を選択できます。
- **シリアル通信** [2] - RS-232C の信号レベルで通信できるシリアルチャンネルを 1 チャンネル備えています。通信速度は 300bps～38400bps です。
- **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。
- 上記以外にも製品に内蔵されたタイマを利用して、一定サイクルで出力信号を変化させる機能(タイマコピー機能)、パルスカウンタへの入力をトリガとして出力を変化させる機能(コンペアアウト機能)などの独自の機能を備えています。
- 『LANX-I16P』では+12V のインタフェース用電源を内蔵しているため、フォトカプラや出力トランジスタを駆動するための I/O 用電源を必要としません。
- 制御用 API は DLL モジュールで提供され、Visual C++® や Visual Basic® から呼び出すことで、Windows 上のアプリケーションから簡単に制御できます。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 使用 API により変換速度は変化します。

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

3. 製品仕様

□ 仕様

表 2　共通仕様

項目	仕様	備考
寸法	130(W)×135(D)×40(H)[mm]	DIN レール取付具含まず
重量	550[g]	
電源電圧	DC5[V]	
消費電流	最大 800[mA]	
動作温度範囲	0～50[℃]	
	0～45[℃]	『LANX-I16』の 24V 入出力時
フラッシュメモリの プログラム保持年数	10[年]	
絶縁抵抗	50[MΩ]以上	測定条件:500VDC
インタフェース	10BASE-T、100BASE-TX	AUTO-MDIX 対応
対応 OS	Windows XP, Vista, 7, 8, 8.1, 10	

表 3　絶縁入力仕様

項目	仕様	備考
入力点数	16 点	P10～P17, P20～P27
入力方式	シンク、およびソース	『LANX-I16P』はシンクのみ
入力電圧	-25.2～25.2[V]	入力端子,COM 端子間電圧
入力抵抗	4.7kΩ	
フォトカプラ応答速度	最大 100[μsec]	

表 4　絶縁出力仕様

項目	仕様	備考
出力点数	16 点	P40～P47, PA0～PA7
出力方式	オープンドレイン	
出力コモン端子電圧	11.4～25.2[V]	COM+,COM-端子間電圧
出力電圧	最大 25.2[V]	
出力電流	最大 150[mA]	1 ピンあたり
フォトカプラ応答速度	最大 100[μsec]	

表 5　非絶縁出力仕様

項目	仕様	備考
出力点数	8 点	POUT0～POUT7
出力方式	オープンコレクタ	
出力電圧	最大 50[V]	条件:DC 出力時, 25℃
出力電流	最大 30[mA]	1 ピンあたり

表 6　パルスカウンタ仕様

項目	仕様	備考
入力チャンネル	4 チャンネル	PC0～PC3
入力方式	シンク、およびソース	『LANX-I16P』はシンクのみ
入力電圧	-25.2～25.2[V]	
入力抵抗	4.7kΩ	
フォトカプラ応答速度	最大 60[μsec]	
ビット数	32 ビット	
カウント設定	ON→OFF、OFF→ON、両エッジ	
周波数	最大 5[kHz]	

表 7　AD コンバータ仕様

項目		仕様	備考
入力チャンネル		4 チャンネル	AD0～AD3
入力電圧	AD0,AD1	−2.5～2.5[V]、または 0～5.0[V]	
	AD2,AD3	0～5.0[V]	
チャンネル間クロストーク		標準−55[dB]	条件：100[kHz]
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]	
	変換時間	5.36[μsec][3]	
	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
アンプ部[4]	オフセット電圧	最大±28[mV]	条件：全温度範囲
	ゲインエラー	±0.3[%]	条件：全温度範囲

表 8　DA コンバータ仕様

項目		仕様	備考
出力チャンネル		2 チャンネル	DA0～DA1
出力電圧		−2.5～2.5[V]、または 0～5.0[V]	
出力電流		−2.0～+2.0[mA]	チャンネルあたり
チャンネル間クロストーク		標準 20[mV]	条件：他方のチャンネルを最小値から最大値に変化させた時
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	
	変換時間	10[μsec]	
	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±28[mV]	条件：全温度範囲
	ゲインエラー	±0.3[%]	条件：全温度範囲

表 9　シリアルポート仕様

項目	仕様	備考
チャンネル数	1	
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	

[3] 1 チャンネルあたりの時間です。複数チャンネルの変換にはチャンネル数をかけた分だけの時間が必要です。

[4] 入力アンプは AD0 と AD1 チャンネルのみ内蔵しています。AD2、AD3 は直接 AD コンバータに接続されます。

[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。

□　各部の名称

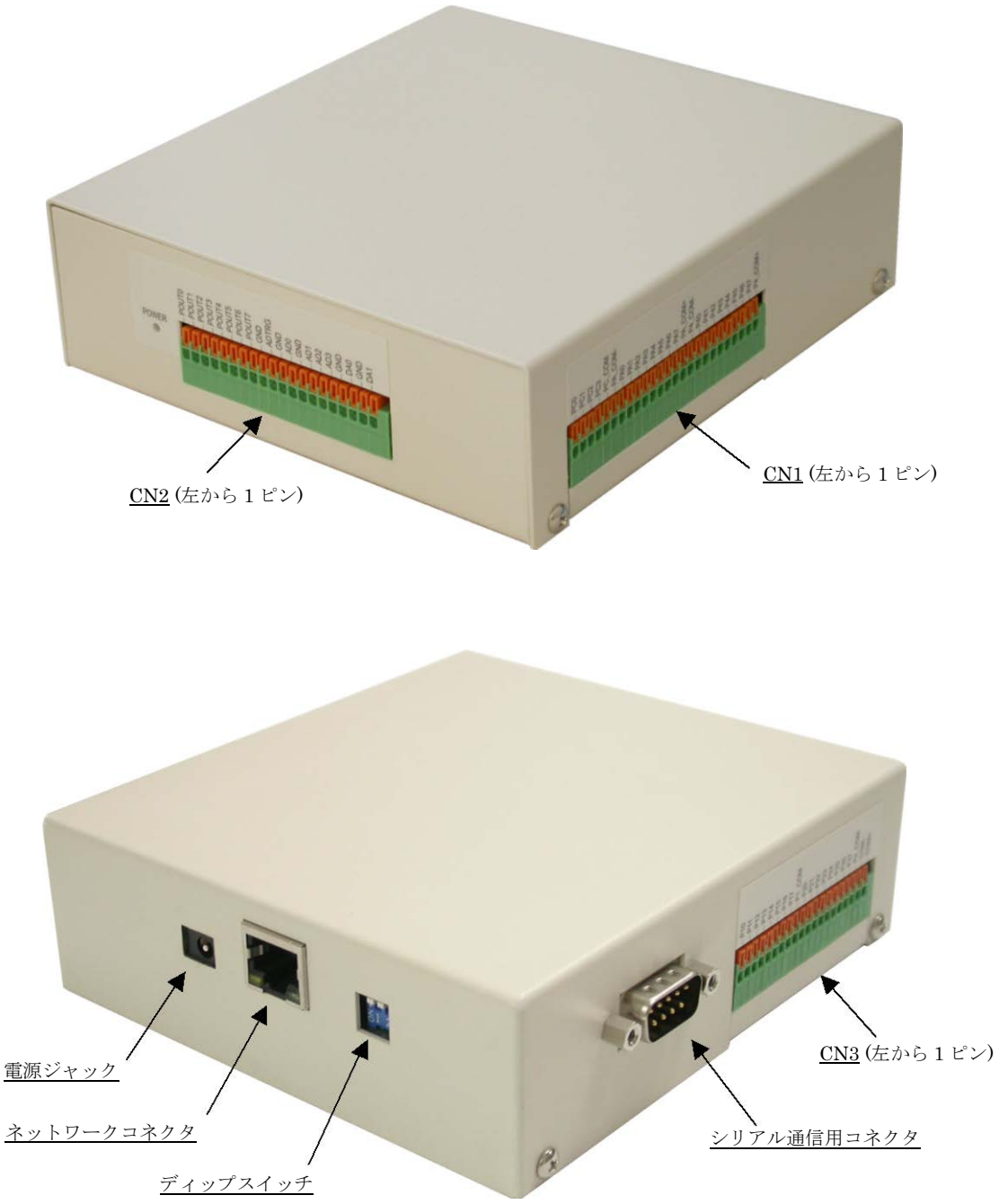


図 1　各部の名称

□ 端子説明

表 10　CN1 端子

コネクタ-ピン番	信号名	説明	方向	絶縁/非絶縁
CN1-1	PC0	パルスカウンタ 0 入力	I	絶縁
CN1-2	PC1	パルスカウンタ 1 入力	I	絶縁
CN1-3	PC2	パルスカウンタ 2 入力	I	絶縁
CN1-4	PC3	パルスカウンタ 3 入力	I	絶縁
CN1-5	PC_COM	パルスカウンタ用コモン	-	絶縁
CN1-6	PA_COM-	PA 用コモン(-)	-	絶縁
CN1-7	PA0	PA デジタル出力	O	絶縁
CN1-8	PA1	PA デジタル出力	O	絶縁
CN1-9	PA2	PA デジタル出力	O	絶縁
CN1-10	PA3	PA デジタル出力	O	絶縁
CN1-11	PA4	PA デジタル出力	O	絶縁
CN1-12	PA5	PA デジタル出力	O	絶縁
CN1-13	PA6	PA デジタル出力	O	絶縁
CN1-14	PA7	PA デジタル出力	O	絶縁
CN1-15	PA_COM+	PA 用コモン(+)	-	絶縁
CN1-16	P4_COM-	P4 用コモン(-)	-	絶縁
CN1-17	P40	P4 デジタル出力	O	絶縁
CN1-18	P41	P4 デジタル出力	O	絶縁
CN1-19	P42	P4 デジタル出力	O	絶縁
CN1-20	P43	P4 デジタル出力	O	絶縁
CN1-21	P44	P4 デジタル出力	O	絶縁
CN1-22	P45	P4 デジタル出力	O	絶縁
CN1-23	P46	P4 デジタル出力	O	絶縁
CN1-24	P47	P4 デジタル出力	O	絶縁
CN1-25	P4_COM+	P4 用コモン(+)	-	絶縁

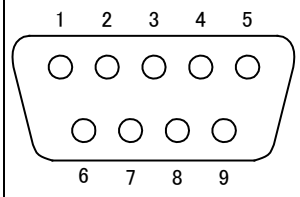
表 11　CN2 端子

コネクタ-ピン番	信号名	説明	方向	絶縁/非絶縁
CN2-1	POUT0	POUT デジタル出力	O	非絶縁
CN2-2	POUT1	POUT デジタル出力	O	非絶縁
CN2-3	POUT2	POUT デジタル出力	O	非絶縁
CN2-4	POUT3	POUT デジタル出力	O	非絶縁
CN2-5	POUT4	POUT デジタル出力	O	非絶縁
CN2-6	POUT5	POUT デジタル出力	O	非絶縁
CN2-7	POUT6	POUT デジタル出力	O	非絶縁
CN2-8	POUT7	POUT デジタル出力	O	非絶縁
CN2-9	GND	シグナル GND	-	非絶縁
CN2-10	ADTRG	AD トリガ入力	I	非絶縁
CN2-11	GND	シグナル GND	-	非絶縁
CN2-12	AD0	AD0 アナログ入力	I	非絶縁
CN2-13	GND	シグナル GND	-	非絶縁
CN2-14	AD1	AD1 アナログ入力	I	非絶縁
CN2-15	AD2	AD2 アナログ入力	I	非絶縁
CN2-16	AD3	AD3 アナログ入力	I	非絶縁
CN2-17	GND	シグナル GND	-	非絶縁
CN2-18	DA0	DA0 アナログ出力	O	非絶縁
CN2-19	GND	シグナル GND	-	非絶縁
CN2-20	DA1	DA0 アナログ出力	O	非絶縁

表 12　CN3 端子

コネクタ-ピン番	信号名	説明	方向	絶縁/非絶縁
CN3-1	P10	P1 デジタル入力	I	絶縁
CN3-2	P11	P1 デジタル入力	I	絶縁
CN3-3	P12	P1 デジタル入力	I	絶縁
CN3-4	P13	P1 デジタル入力	I	絶縁
CN3-5	P14	P1 デジタル入力	I	絶縁
CN3-6	P15	P1 デジタル入力	I	絶縁
CN3-7	P16	P1 デジタル入力	I	絶縁
CN3-8	P17	P1 デジタル入力	I	絶縁
CN3-9	P1_COM	P1 用コモン	-	絶縁
CN3-10	P20	P2 デジタル入力	I	絶縁
CN3-11	P21	P2 デジタル入力	I	絶縁
CN3-12	P22	P2 デジタル入力	I	絶縁
CN3-13	P23	P2 デジタル入力	I	絶縁
CN3-14	P24	P2 デジタル入力	I	絶縁
CN3-15	P25	P2 デジタル入力	I	絶縁
CN3-16	P26	P2 デジタル入力	I	絶縁
CN3-17	P27	P2 デジタル入力	I	絶縁
CN3-18	P2_COM	P2 用コモン	-	絶縁
CN3-19	COM-	内蔵電源端子-[6]	-	絶縁
CN3-20	COM+	内蔵電源端子+[6]	-	絶縁

表 13　シリアル通信用コネクタ(DSUB 9 ピン-オス)



ピン番	信号名	説明	パソコンの信号名(参考)
1	-		DCD
2	RxD	シリアル入力	RxD
3	TxD	シリアル出力	TxD
4	-	6 ピンと接続されています。	DTR
5	GND	シグナル GND	GND
6	-	4 ピンと接続されています。	DSR
7	-	8 ピンと接続されています。	RTS
8	-	7 ピンと接続されています。	CTS
9	-		RI

□　ディップスイッチ

表 14　ディップスイッチ

番号	説明
1	通常は"OFF"で使用します。ユーザーファーム [7]を起動する場合に"ON"にします。
2	機器の番号設定、内蔵ファームウェアのアップデートのとき"ON"にします。通常使用時は必ず"OFF"にしてください。

6 『LANX-I16』では NC 端子です。接続しないでください。

7 カスタム動作させるための追加のファームウェアプログラムをユーザーファームと呼びます。

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する場合は図 2 の向きに取り付けてください。

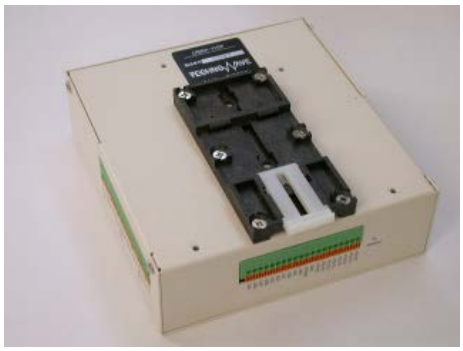


図 2　DIN レール取付具の固定

□ 配線方法

図 3 は製品の端子台と適合する線材です。配線の際はマイナスドライバーなどでスイッチ部分を押し込み、線材を接点部分に挿入します（図 4）。故障の原因となりますので静電気には十分ご注意ください。アースバンドのご利用をお勧めします。

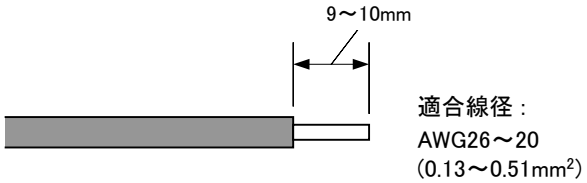


図 3　適合線材

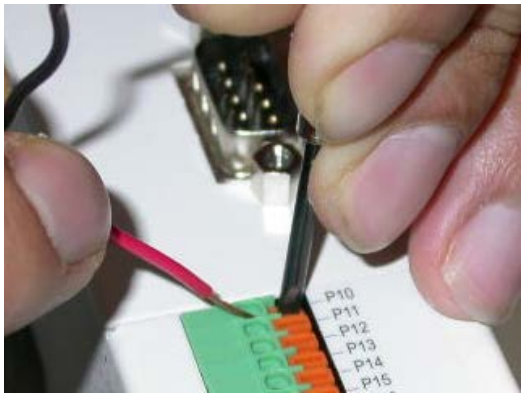


図 4　配線作業

□ ライブラリ、設定ツールのインストール

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

表 15　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「.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)。
	TWX3069.DLL (32bit 版)		
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	
	TWX3069.DLL (64bit 版)		

表 15 は製品の制御に必要なライブラリファイルです。これらのファイルは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。

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

□ LabVIEW ライブラリのインストール

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

VI ライブラリのインストールには、付属 CD の「¥VI¥TWX3069VI」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。

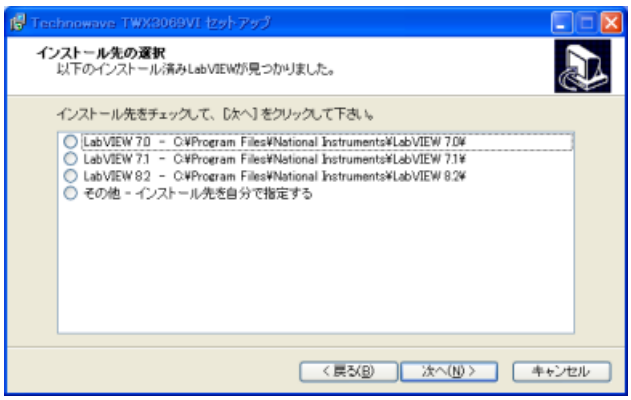


図 5　VI ライブラリのセットアップ画面

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→[テクノウェーブ]→[TWX3069VI]の中に作られます。

□ ネットワーク設定

製品は使用開始前にネットワーク設定を適切に行う必要があります。表 16 はネットワーク設定の設定項目です。

表 16 ネットワーク設定の項目

項目	初期値	説明
MAC アドレス	製品固有値	製品の MAC アドレスです。変更する必要はありません。
ポート	49152	製品をサーバーモードで使用する場合にライブラリから製品に接続するためのポート番号を指定します。通常変更する必要はありません。変更する場合は 49152～65535 の範囲で番号を設定してください。TCP と UDP で同じ番号を使用します。
IP アドレス	DHCP による自動取得	製品が使用する IP アドレス。初期状態では DHCP サーバーから割り当てを受けるように設定されています。
サブネットマスク	DHCP による自動取得	製品が利用するネットワークのマスク。初期状態では DHCP サーバーから割り当てを受けるように設定されています。
ゲートウェイ	DHCP による自動取得	ルーターなどを通じて外部と通信する場合のゲートウェイアドレスを指定します。初期状態では DHCP サーバーから割り当てを受けるように設定されています。ローカルネットだけで使用する場合は空欄でも構いません。
DNS サーバー	DHCP による自動取得	クライアントモードやユーザーファームでドメイン名を解決する必要がある場合に指定します。
NTP サーバー	設定なし	ユーザーファームで日時が必要な場合に、時刻合わせをするためのサーバーを指定します。
パスワード	初期パスワード	パソコンからデバイスにアクセスする場合の認証に使用されるパスワード。設定しない場合はデフォルトの初期パスワードで認証が行われます。
クライアントモード	チェックなし	製品をクライアントモードで動作させる場合にチェックします。クライアントモードにすると一定間隔でサーバーアドレスに指定したホストに接続を試みます。
サーバーアドレス	設定なし	クライアントモード時の接続先サーバーを指定します。IP アドレスまたはドメイン名で指定してください。
サーバーポート	50176	クライアントモード時の接続先サーバーのポート番号を指定します。

- [ポート]を変更した場合は TWXA ライブラリが接続先として使用するポート番号を *TWXA_SetNetworkPort()* 関数で変更するか、*TWXA_OpenByAddress()* 関数でアドレスとポートを指定して接続する必要があります。
- [パスワード]を変更した場合は *TWXA_SetPassword()* 関数を使用し、TWXA ライブラリ側のパスワード設定も変更する必要があります。
- [サーバー]や[NTP サーバー]にドメイン名を指定する場合には、DNS サーバーとの通信が必要になります。

パソコンと 1 対 1 で接続する場合は IP アドレスなどを自動取得することができません。そのような場合はプライベートアドレス [8]を使用するのが一般的です。表 17 にプライベートアドレスを利用した設定例をあげます。

[8] 組織内などの閉じた範囲で自由に割り当てができるアドレス。インターネット上では使用できません。

表 17　プライベートアドレスの設定例

項目	製品側の設定値	パソコン側の設定値
IP アドレス	192.168.0.2	192.168.0.1
サブネットマスク	255.255.255.0	255.255.255.0
ゲートウェイアドレス	空欄	空欄

ネットワーク設定ユーティリティの使用方法

① 製品のディップスイッチ 2 番を"ON"にし、電源を入れ、ネットワークに接続します。

② 「LANX-I16　ネットワーク設定ユーティリティ」を起動します（デフォルトでは[スタート]メニューの[テクノウェーブ]の中にショートカットが作られます）。

③ [デバイスと接続]ボタンを押すと左側の[現在の設定]欄に接続されている製品の情報が表示されます。接続に失敗する場合はオンラインヘルプを参照し、接続設定を行ってください。

④ 右側の「新しい設定」欄を編集します。[IP アドレス]、[サブネットマスク]、[ゲートウェイ]を編集する場合は[自動取得(DHCP)]のチェックを外します。

⑤ クライアントモードやユーザーファームで DNS サーバーへのアクセスが必要な場合は、[DNS サーバー]の欄にサーバーアドレスを設定します。DHCP を使用する場合には[自動取得]とすることもできます。

⑥ デバイスをクライアントモードで動作させる場合には[クライアントモード]をチェックし、接続先となる[サーバー]と[サーバーポート]を指定します。

⑦ [ファイルに保存]ボタンを押すと編集内容をファイルに保存することができます。保存したデータは[ファイルを開く]ボタンで読み出すことができます。**ただし、ファイルから読み出した際はパスワードが表示されませんので、別途控えておく必要があります。**

⑧ 編集が終了したら[デバイスへ書込み]ボタンを押して製品に設定値を書き込みます。製品の電源を切り、ディップスイッチ 2 番を"OFF"に戻してください。ネットワーク設定の書換えは 3200 回まで可能です。

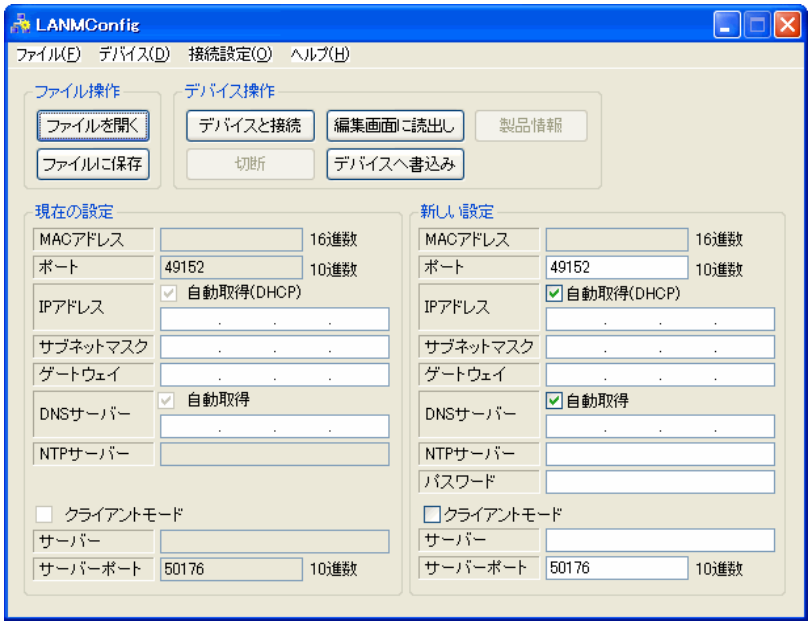


図 6　ネットワーク設定の画面

□ 複数の製品を同時に使用する場合の設定

複数の製品を同時に使用する場合、あらかじめ装置の番号設定を行います。

① 番号を設定する製品のディップスイッチの 2 番を”ON”にしてネットワークに接続し、電源を入れます。

② 設定する製品以外がネットワークに接続されている場合は取り外してください。

③ 「LANX-I16 番号設定ユーティリティ」を起動します（デフォルトでは[スタート]メニューの[テクノウェーブ]の中にショートカットが作られます）。

④ [接続]ボタンを押してください。

⑤ [新しい番号]に 1～65535 の範囲の数値を入力します。[自動加算]にチェックしておくと書込みの度に[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。API 関数からは入力した番号を指定することで、操作する製品を指定することができます。

⑦ [切断]ボタンを押して製品を取り外し、ディップスイッチの 2 番を”OFF”に戻してください。番号の書換えは 3200 回まで可能です。

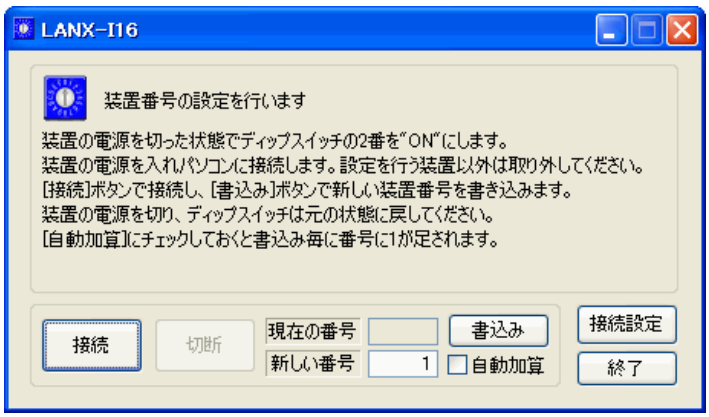


図 7 「LANX-I16 番号設定ユーティリティ」の操作画面

□ 追加ファーム(ユーザーファーム)の利用について

製品には基本機能に加えて利用できる追加のファームウェアが用意されています。表 18 は現在公開されている追加ファームの一覧です。使用方法については、それぞれの格納フォルダ内のドキュメントを参照してください。

表 18　追加ファーム

名称	機能	格納フォルダ
LANX-I16(P)用 Telnet ファームウェア	製品を Telnet で制御できるようになるファームウェアです。 ネットワークを通じた接点交換にも利用可能です。	添付 CD の「TOOL¥LANXTools¥LanxTelnet」
LANX-I16(P)用バイナリコマンド・ファームウェア	TCP による通信で製品を直接制御するためのファームウェアです。 Windows 以外の OS を利用する場合に有効です。	添付 CD の「¥TOOL¥LANXTools¥Lxi16BinCmd」
LANX-I16(P)用 HTTP ファームウェア	製品を WEB ブラウザから制御できるようになるファームウェアです。	添付 CD の「TOOL¥LANXTools¥Lxi16Http」

- 複数の追加ファームを同時に使用することはできません。

□ プログラミングの準備

Visual C++ の場合

●CD-ROM から必要なファイルをコピーします。コピーするファイルとコピー先を以下に示します。

表 19　C 言語用ファイルのインストール

作成するプログラム	ファイル名	コピー元	コピー先
32bit プログラム	TWX3069.H	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ
	TWX3069.LIB		
	USBM3069.H	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ(USBM で始まる関数を使用する場合のみ必要)
	USBM3069.LIB		
64bit プログラム	TWX3069.H	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ
	TWX3069.LIB	CD の「¥DLL¥x64」フォルダ	お客様のプロジェクトフォルダ
	USBM3069.H	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ(USBM で始まる関数を使用する場合のみ必要)
	USBM3069.LIB	CD の「¥DLL¥x64」フォルダ	

●「TWX3069.LIB」、「USBM3069.LIB」をお客様のプロジェクトに追加します。

●API 関数を呼び出す必要がある場合、適宜「TWX3069.H」、「USBM3069.H」をインクルードしてください。

Visual Basic 6.0 の場合

●CD-ROM から必要なファイルをコピーします。コピーするファイルとコピー先を以下に示します。

表 20　Visual Basic 6.0 用ファイルのインストール

ファイル名	コピー元	コピー先
USBM3069.BAS	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ
TWX3069.BAS		

●「USBM3069.BAS」、「TWX3069.BAS」をお客様のプロジェクトに追加します。

Visual Basic .NET 以降の場合

●CD-ROM から必要なファイルをコピーします。コピーするファイルとコピー先を以下に示します。

表 21　Visual Basic .NET 用ファイルのインストール

ファイル名	コピー元	コピー先
USBM3069.VB	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ
TWX3069.VB		

●「USBM3069.VB」、「TWX3069.VB」をお客様のプロジェクトに追加します。

- 上記の開発用ファイルは「LANXTools」をインストールすることで、インストール先のドライブにコピーが作成されます。コピー先のフォルダは通常、[スタート]メニュー→[すべてのプログラム]([プログラム])→[テクノウェーブ]→[ライブラリ]で開くことができます。

□ Visual Basic 6.0 と Visual Basic .NET の相違点

Visual Basic 6.0 と Visual Basic .NET 以降では変数名と、プロシージャの呼び出しの記述方法に相違があります。まず、変数名についてですが、整数を表す変数の名称が以下のように変更になっています。

表 22　Visual Basic 6.0 と Visual Bsic .NET の変数名

ビット数	Visual Basic 6.0	Visual Basic .NET
16 ビット	Integer	Short
32 ビット	Long	Integer
64 ビット	なし	Long

次に、プロシージャを呼び出す場合ですが、戻り値を必要としない場合も引数を"()"で囲む必要があります。本マニュアルには Visual Basic 6.0 を対象としたサンプルを記載していますが、Visual Basic .NET をご使用の場合には、以上の点にご注意ください。以下に例を示します。

Visual Basic 6.0 の場合

```
Dim ADData(3) As Integer

USBM_ADRead hDev, ADData, 3, 1
```

Visual Basic .NET の場合

```
Dim ADData(3) As Short

USBM_ADRead(hDev, ADData, 3, 1)
```

5. ハードウェア

□ 絶縁入出力端子

LANX-I16（電源なしタイプ）

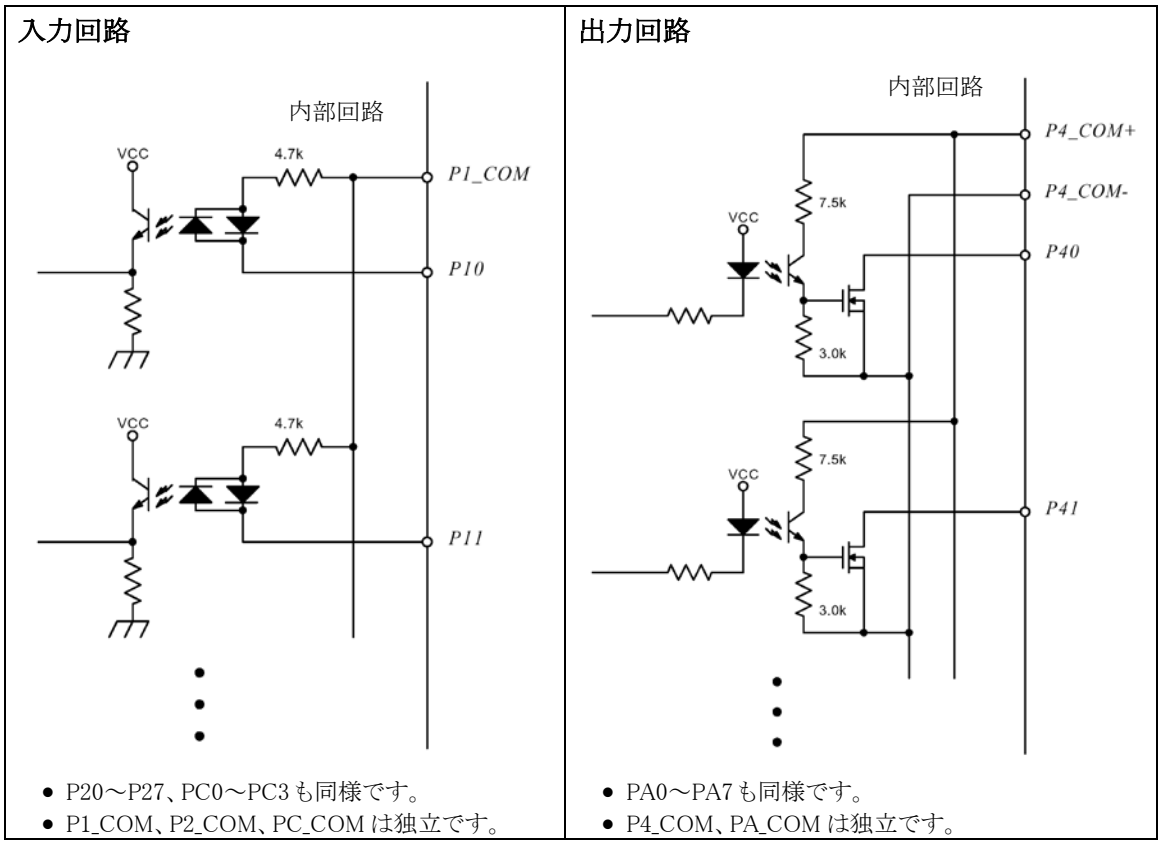


図 8　LANX-I16 の絶縁入出力回路

絶縁入力端子の接続例

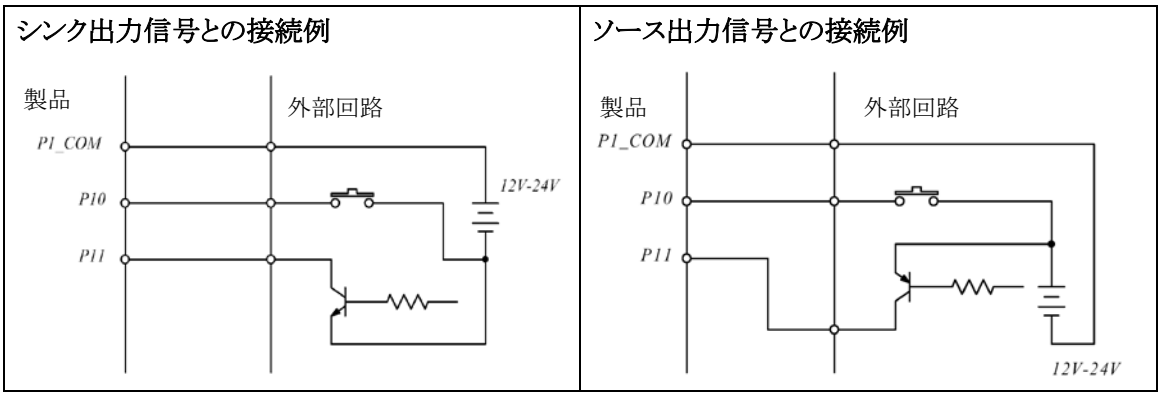


図 9　LANX-I16 絶縁入力端子の接続例

絶縁出力端子の接続例

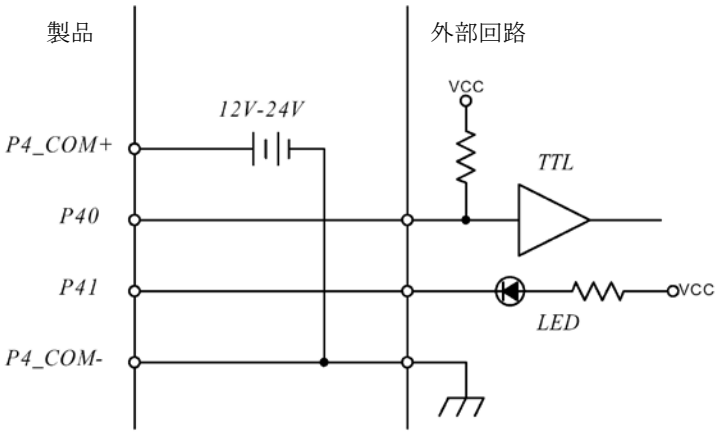


図 10　LANX-I16 絶縁出力端子の接続例

LANX-I16P（電源内蔵タイプ）

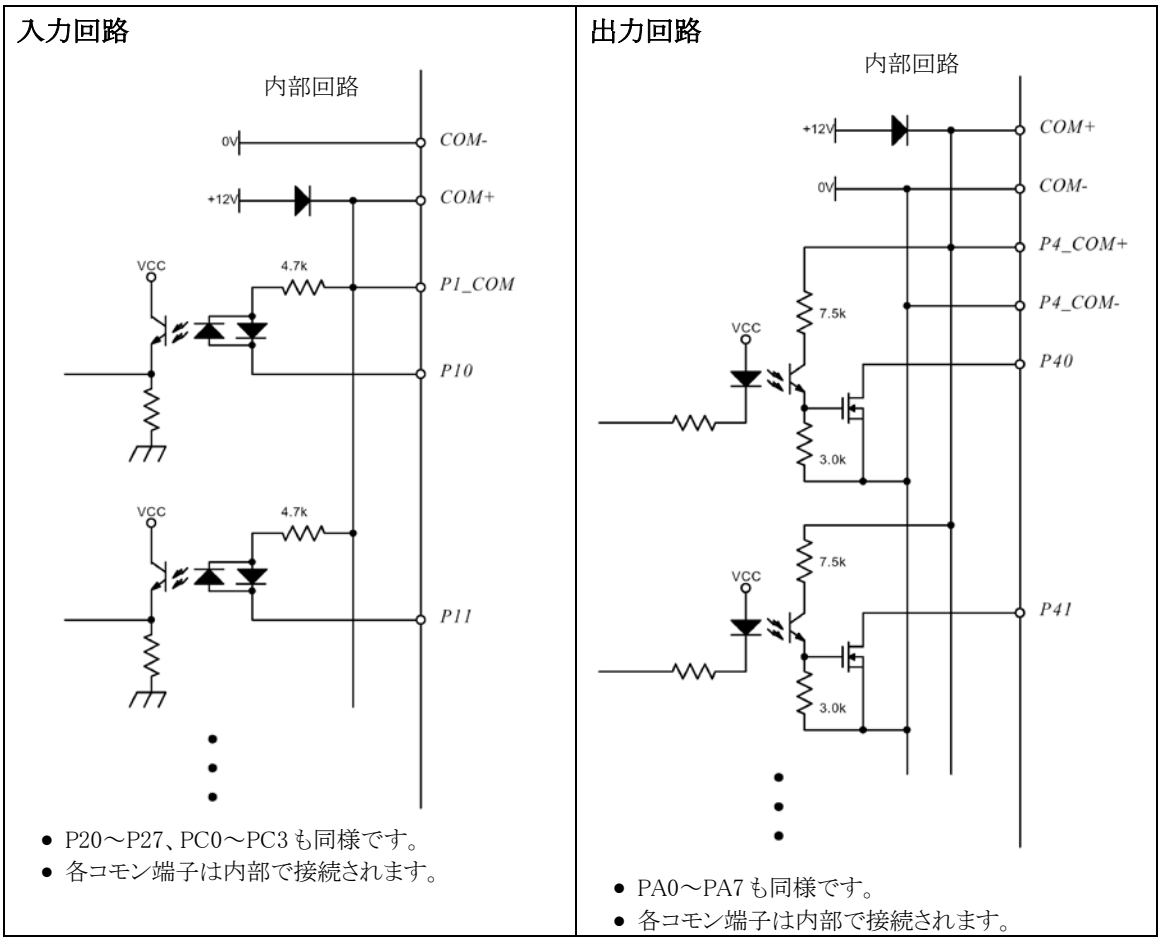


図 11　LANX-I16P の絶縁入出力回路

絶縁入力端子の接続例

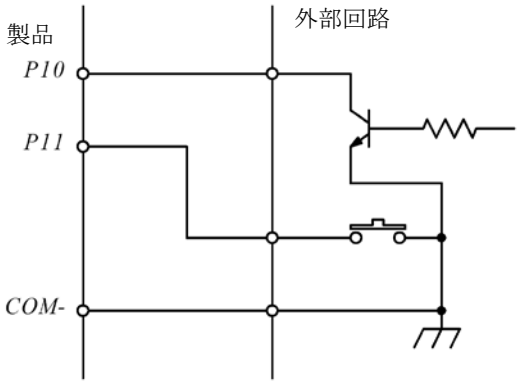


図 12　LANX-I16P 絶縁入力端子の接続例

悪い接続例

外部電源と内蔵電源を下記のような接続にしないでください。

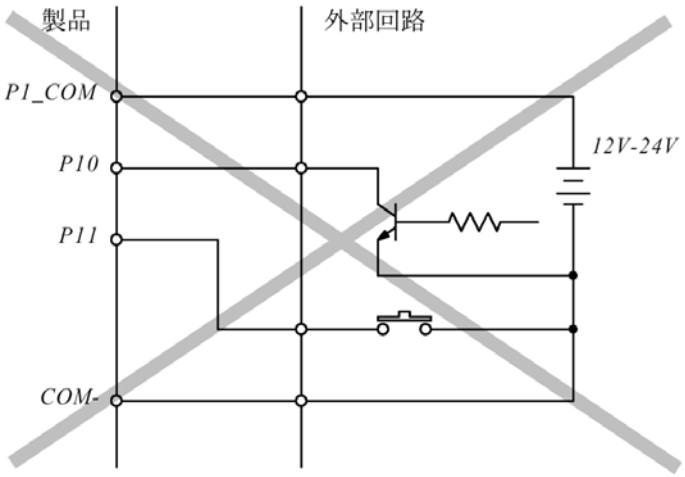


図 13 悪い接続例

絶縁出力端子の接続例

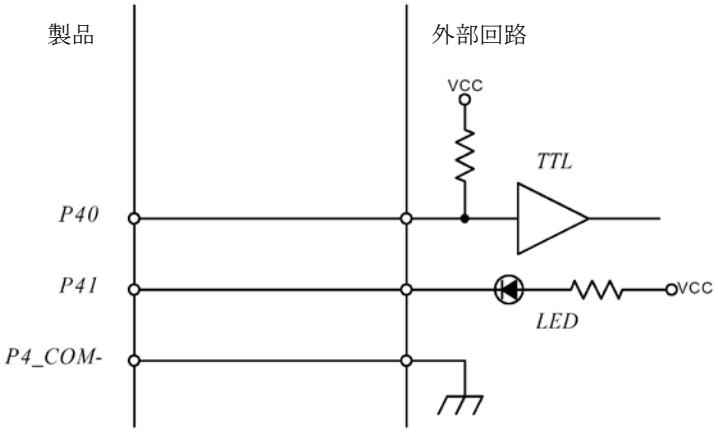


図 14　LANX-I16P 絶縁出力端子の接続例

内蔵電源の容量

内蔵の+12V 電源は外部回路で 60mA までご使用いただけます。ただし 40℃を超える場合はディレーティングが必要になります。

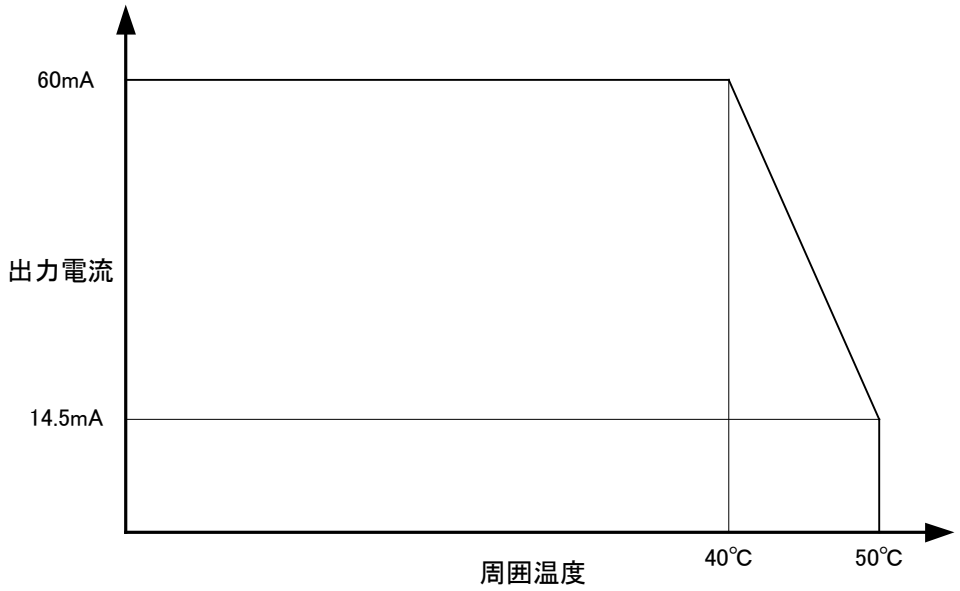


図 15　内蔵電源のディレーティング

□　非絶縁出力端子

出力回路と接続例

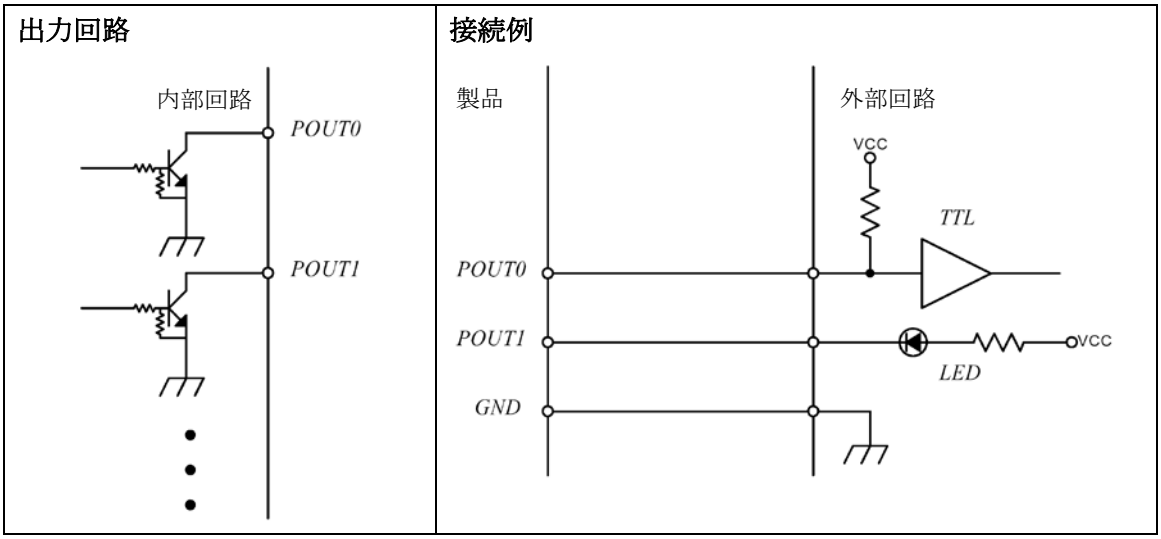


図 16　非絶縁出力回路と接続例

□ アナログ入力端子

アナログ入力端子のうち AD0、AD1 については入力バッファを介して AD コンバータに接続されています。これらの端子は入力インピーダンスが 10MΩ以上あり、入力範囲も 0～5V と-2.5～2.5V のどちらかを選択することができます。そのため、負電圧を測定する必要がある場合や、交流信号を測定する場合に向いています。

AD2、AD3 は直接 AD コンバータに接続されています。これらの端子には負電圧を入力することはできません。直流信号の測定に向いています。

接続例

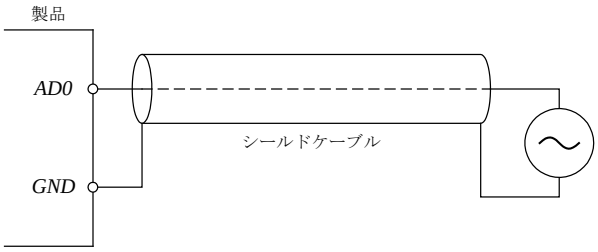


図 17　アナログ入力接続例

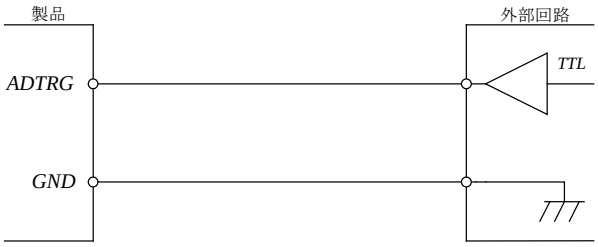


図 18　トリガ入力接続例

- アナログ信号の接続にはシールドケーブルの使用を推奨します。
- トリガ入力は“立ち上がり”検出、“立ち下がり”検出のどちらにも設定可能です。

電源オフ時に入力電圧が加わる場合

製品の電源が入っていないときに、アナログ入力、トリガ入力に電圧が加わる場合、図 19、図 20 のように保護素子を追加してください。特に AD2、AD3、ADTRG 端子は信号源のインピーダンスが低いと、大きな電流が流れ込む恐れがあります。

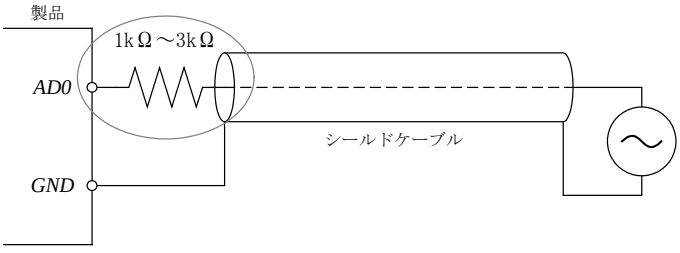


図 19　アナログ入力端子の保護

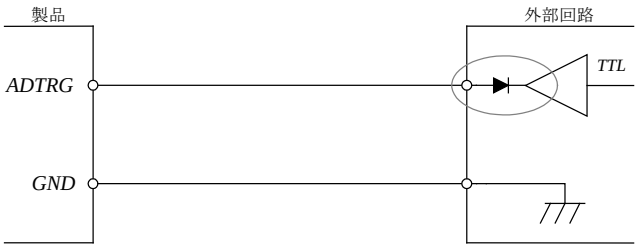


図 20　トリガ入力端子の保護

□ アナログ出力端子

アナログ出力部には出力バッファを内蔵しています。出力範囲は 0～5V と-2.5～2.5V のどちらかを選択することができます。

接続例

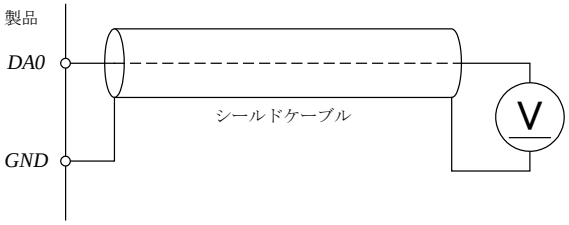


図 21　アナログ出力接続例

- アナログ信号の接続にはシールドケーブルの使用を推奨します。

6. 基本プログラミング

この章では、製品を使用する上で基礎的なプログラミング方法について説明しています。さらに高度な使用方法につきましては次章で説明しますが、製品をご使用になる上で重要な情報を記載しておりますので、まずは本章の内容をご一読ください。

尚、製品の制御に使用するライブラリ関数で名称が"TWX_"で始まる関数は「TWX ライブラリ関数リファレンス」(66 ページ)に詳しい説明があります。また"USBM_"で始まる関数の詳細については別紙「USBM ライブラリ関数リファレンス」を参照してください。

□ 接続と初期化

製品の各機能を使用するには、まず接続処理を行い、ハンドルを取得する必要があります。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、ハンドルの値は終了まで記憶しておく必要があります。また、製品の操作を終えてアプリケーションを終了する場合などは、ハンドルをクローズするようにしてください。

表 23　接続、初期化、終了に使用する関数

関数名	説明
TWX_Open()	デバイスに接続します。
TWX_OpenByAddress()	IP アドレスを指定してデバイスに接続します。
TWX_Close()	デバイスの操作を終了します。
TWX_CloseAll()	プロセスが接続中のデバイス全てを切断します。
TWX_InitializeA()	デバイスの初期化を行います。初期化項目を指定できます。
TWX_Initialize()	デバイスの初期化を行います。
TWX_Listen()	クライアントモードのデバイスの接続待ちを行います。
TWX_Accept()	クライアントモードのデバイスから接続要求があれば接続しハンドルを返します。
TWX_CloseListenSocket()	接続待ちを終了します。
TWX_GetNumber()	接続したデバイスの装置番号を調べます。

デバイスに接続する

パソコンと同一ネットワーク内のデバイスと接続する場合は *TWX_Open()* 関数を使用します。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 18 ページを参照してください。

デバイスへの接続に成功した場合は *TWX_InitializeA()* 関数または *TWX_Initialize()* 関数を呼び出して初期化を行ってください。*TWX_InitializeA()* 関数を使用すると機能を選択して初期化を行うことができ、以前の端子状態などを保存したまま初期化作業を行うことができます。

デバイスの操作を終了する

TWX_Close() 関数を呼び出してください。クローズしたハンドルは無効になりますので、再接続したい場合はもう一度 *TWX_Open()* 関数で新しいハンドルを取得してください。

C 言語の例

```
TW_HANDLE hDev;

TWX_Open(&hDev, 1, TWX_ANY_DEVICE); /*装置番号 1 に接続*/
if(hDev){
  TWX_Initialize(hDev, TWX_INIT_ALL); /*デバイスの初期化*/

  /*...制御の中身*/

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

VisualBasic6.0 の例

```
Dim hDev As Long

hDev = TWX_Open(hDev, 1, TWX_ANY_DEVICE) '装置番号 1 に接続
If hDev Then
  TWX_InitializeA hDev 'デバイスの初期化

  '...制御の中身

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

IP アドレスを指定してデバイスをオープンする

ルーターなどを介して異なるネットワークにあるデバイスと接続する場合には、必ず IP アドレスやドメイン名を指定する必要があります。この場合、*TWX_OpenByAddress()* 関数を使用します。

C 言語の例

```
TW_HANDLE hDev;

/*IP アドレスを指定してオープン*/
TWX_OpenByAddress(&hDev, "192.168.0.2", TWX_LANX_I16);

if(hDev){
  TWX_InitializeA(hDev, TWX_INIT_ALL);  /*デバイスの初期化*/

  /*...*/

  TWX_Close(hDev);  /*デバイスを閉じる*/
}
```

VisualBasic6.0 の例

```
Dim hDev As Long

'IP アドレスを指定してオープン
TWX_OpenByAddress hDev, "192.168.0.2", TWX_LANX_I16

If hDev <> 0 Then
  TWX_InitializeA hDev 'デバイスの初期化

  '...

  TWX_Close hDev 'デバイスを閉じる
End If
```

- DDNSなどのサービスにより、デバイスにドメイン名が割り当てられている場合は、IPアドレスの代わりにドメイン名を指定することもできます。
- ポート番号を指定したい場合は、アドレスの後に":"(コロン)とポート番号を続けてください。

クライアントモードに設定したデバイスと接続する

デバイスをクライアントモードに設定すると、デバイス側からサーバーとなるパソコンに対してネットワーク接続を行います。クライアントモードを利用すると、インターネットなどを通じて複数のデバイスを制御したい場合に、パソコン側のポートだけを外部から接続可能な状態にすれば良いのでネットワーク設定が容易になります。

クライアントモードのデバイスと接続するためには、まず *TWX_Listen()* 関数を呼び出して、ネットワーク接続を受け入れるポートを準備します。処理が成功すると *TWX_Listen()* 関数はソケットと呼ばれる一種の識別子を返します。

次に、取得したソケットを引数として *TWX_Accept()* 関数を呼び出します。*TWX_Accept()* 関数は接続要求を行っているデバイスがあれば、そのデバイスと接続して制御用のハンドルを返します。接続が完了しても、最初に *TWX_Listen()* 関数で準備したポートとソケットは引き続き有効ですので、再度 *TWX_Accept()* 関数に渡して他のデバイスの接続要求を受け入れることができます。

TWX_Accept() 関数は接続要求が無ければ、すぐに終了し *TW_DEVICE_NOT_FOUND*を返しますので、定期的に呼び出してデバイスからの接続をチェックするようにします。

□ デジタル入出力

以下では入力用端子を入力ポート、出力用端子を出力ポートと呼びます。入力ポートと出力ポートを合わせて入出力ポートと表現します。入出力ポートは 8 つの端子を 1 組としてそれぞれに名前が付けられています。表 24 に使用できる入出力ポートの一覧を示します。

さらに入出力ポートの 8 つの端子は 0～7 までの番号で表されます。例えば P1 ポートには P10～P17 までの 8 つの端子があります。その他のポートについても同様です。

表 24　入出力ポート

ポート名	絶縁/非絶縁	方向
P1	絶縁	入力
P2	絶縁	入力
P4	絶縁	出力
PA	絶縁	出力
POUT	非絶縁	出力

出力ポートの制御、入力ポートの読み出しには、表 25 の関数を使用します。

表 25　デジタル入出力で使用する関数

関数名	説明
TWX_PortWrite()	出力ポートの状態を変更します。
TWX_PortRead()	入力ポートから読み出しを行います。

ポートから入力する

TWX_PortRead() 関数を使用することでポートからデータを読むことができます。読み出しは8ビット単位で行います。例えば P1 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 26　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	P17	P16	P15	P14	P13	P12	P11	P10

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”になります。

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。

ポートに出力する

TWX_PortWrite() 関数を使用することで出力ポートの状態を変更できます。入力と同様に 8 ビット単位でデータを書き込むことができます。データビットと端子との関係も入力の場合と同様です。

P4 と PA ポートに関しては“0”を書き込んだビットが“ON”になり、“1”を書き込んだビットが“OFF”になります。POUT ポートは逆に“0”を書き込んだビットが“OFF”になり、“1”を書き込んだビットが“ON”となりますのでご注意ください。

TWX_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と対応する端子は影響を受けません。図 22 は H’55 というデータを、*Mask* を H’0F として出力した例です。

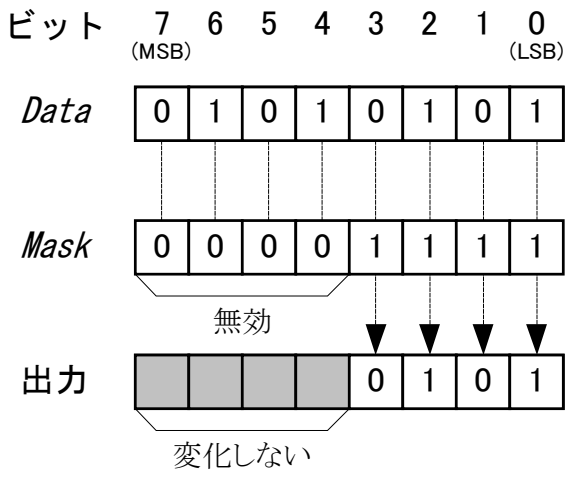


図 22　出力のマスク

- POUT に書き込みを行った場合、ビット毎にデータが反映されるまでの時間に数μsec から 10 数μsec の差が生じますのでご注意ください。

C 言語とVisualBasic6.0 の場合について例を示します。ハンドルの取得と初期化については省略されていますので「接続と初期化」(28 ページ)を参照してください(以下のページのサンプルでも同様です)。

C 言語の例

```
BYTE Data;

TWX_PortRead(hDev, TWX_P1, &Data);        /*P1 からリード*/
TWX_PortWrite(hDev, TWX_P4, Data, 0x0f); /*P40-P43 出力を変更*/
```

VisualBasic6.0 の例

```
Dim Data As Byte

TWX_PortRead hDev, USBM_P1, Data         'P1 からリード
TWX_PortWrite hDev, USBM_P4, Data, &HF   'P40-P43 出力を変更
```

□ アナログ入出力

製品にはアナログ入力用に AD0～AD3、アナログ出力用に DA0～DA1 端子があります。AD2、AD3 以外の端子はアナログ入出力の設定レジスタにより、ユニポーラ(0～5V)とバイポーラ(-2.5V～2.5V)入出力を切り替えることができます。AD2、AD3 はユニポーラ入力のみになります。

表 27 にアナログ入出力で使用する関数を示します。

表 27　アナログ入出力で使用する関数

関数名	説明
TWX_ADRead()	アナログ入力から変換結果を読み出します。
TWX_DAWrite()	アナログ出力値を設定します。
TWX_AnalogReadReg()	アナログ入出力の設定用レジスタの値を読み出します。
TWX_AnalogWriteReg()	アナログ入出力の設定用レジスタの値を設定します。

アナログ入出力端子の電圧範囲を変更する

アナログ入出力端子の電圧範囲は *TWX_AnalogWriteReg()* 関数でアナログ入出力設定レジスタに値を書き込むことで行います。表 28 にレジスタの各ビットの意味と初期値を示します。

例えば DA0 と DA1 をバイポーラ出力、AD0、AD1 をユニポーラ入力とする場合、設定するレジスタ値は H'03 となります。

表 28　アナログ入出力設定レジスタ

ビット	7(MSB)	6	5	4	3	2	1	0(LSB)
意味	-	-	-	TRG	AD1	AD0	DA1	DA0
初期値	1	1	1	0	0	0	0	0

DA0：0のときDA0出力がユニポーラ(0～+5 [V])、1のときバイポーラ(-2.5～+2.5 [V])となります。

DA1：0のときDA1出力がユニポーラ(0～+5 [V])、1のときバイポーラ(-2.5～+2.5 [V])となります。

AD0：0のときAD0入力がユニポーラ(0～+5 [V])、1のときバイポーラ(-2.5～+2.5 [V])となります。

AD1：0のときAD1入力がユニポーラ(0～+5 [V])、1のときバイポーラ(-2.5～+2.5 [V])となります。

TRG：0のときADTRG入力の立ち下がりでトリガ入力となり、1のとき立ち上がりでトリガ入力となります。

レジスタの 4 ビット目は ADTRG 端子の極性を指定します。ADTRG 端子は外部トリガを使用して AD 変換のタイミングを指定する場合に使用します。外部トリガの使用方法に関しては「プログラミング応用編」を参照してください。

アナログ入力値を読み出す

アナログ入力端子の AD 変換結果を読み出すには *TWX_ADRead()* 関数を使用します。入力電圧値と読み出される値の関係を表 29 に示します。

表 29　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
ユニポーラの場合	バイポーラの場合	
5-LSB	2.5-LSB	1023
2.5	0	512
0	-2.5	0

・LSB = 5 / 1024 [V]

・表は理論値を示しています。誤差は含まれません。

C 言語の例

```
long ADData;

TWX_ADRead(hDev, 0, &ADData); /*AD0 の読み出し*/
```

VisualBasic6.0 の例

```
Dim ADData As Long

TWX_ADRead hDev, 0, ADData  'AD0 の読み出し
```

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 *TWX_DAWrite()* 関数を使用します。設定値と出力電圧の関係を表 30 に示します。

表 30　アナログ出力設定値と出力電圧の関係

設定値	出力電圧値([V])	
	ユニポーラの場合	バイポーラの場合
255	5-LSB	2.5-LSB
128	2.5	0
0	0	-2.5

・LSB = 5 / 256 [V]
・表は理論値を示しています。誤差は含まれません。

C 言語の例

```
TWX_AnalogWriteReg(hDev, 0x03); /*DA0 と DA1 をバイポーラ出力に設定*/
TWX_DAWrite(hDev, 1, 128);      /*DA1 を中間電圧に設定*/
```

VisualBasic6.0 の例

```
TWX_AnalogWriteReg hDev, &H3  'DA0 と DA1 をバイポーラ出力に設定
TWX_DAWrite hDev, 1, 128      'DA1 を中間電圧に設定
```

□ パルスカウンタ

製品は32ビットのパルスカウンタを4チャンネル内蔵しています。それぞれのカウンタはPC0～PC3の入力端子に接続され、信号の立ち上がり(“OFF”→“ON”)、立ち下り(“ON”→“OFF”)、または両方のエッジをカウントすることができます。

それぞれのチャンネルが信号のどのエッジをカウントするかは、パルスカウンタの設定用レジスタで自由に設定できます。

また、PC0とPC1、またはPC2とPC3の組み合わせで、エンコーダなどの2相出力をアップ/ダウンカウントすることもできます。

表 31　パルスカウンタで使用する関数

関数名	説明
TWX_PCSetMode()	パルスカウンタのカウントモード設定を行います。
TWX_PCSetCnt()	カウンタの値を設定します。
TWX_PCReadCnt()	カウンタの値を読み出します。
TWX_PCStart()	指定チャンネルのカウントをスタートします。
TWX_PCStop()	指定チャンネルのカウントをストップします。
TWX_PCWriteReg()	パルスカウンタの設定用レジスタに書き込みます。
TWX_PCReadReg()	パルスカウンタの設定用レジスタを読み出します。

カウントするエッジを設定する

それぞれのチャンネルが信号のどのエッジをカウントするかは、*TWX_PCWriteReg()* 関数でパルスカウンタの設定用レジスタに書き込みを行うことで設定できます。

表 32 にレジスタの各ビットの意味と初期値を示します。例として全てのチャンネルの立ち上がりエッジのみをカウントする場合は、H’55を設定します。

表 32　パルスカウンタ設定用レジスタ

ビット	7	6	5	4	3	2	1	0
意味	PC3F	PC3R	PC2F	PC2R	PC1F	PC1R	PC0F	PC0R
初期値	0	0	0	0	0	0	0	0

PC0R：1のときPC0の立ち上がり(“OFF”→“ON”)でカウントを行ないます。

PC0F：1のときPC0の立ち下がり(“ON”→“OFF”)でカウントを行ないます。

PC1R：1のときPC1の立ち上がり(“OFF”→“ON”)でカウントを行ないます。

PC1F：1のときPC1の立ち下がり(“ON”→“OFF”)でカウントを行ないます。

PC2R：1のときPC2の立ち上がり(“OFF”→“ON”)でカウントを行ないます。

PC2F：1のときPC2の立ち下がり(“ON”→“OFF”)でカウントを行ないます。

PC3R：1のときPC3の立ち上がり(“OFF”→“ON”)でカウントを行ないます。

PC3F：1のときPC3の立ち下がり(“ON”→“OFF”)でカウントを行ないます。

単相のパルスをカウントする

チャンネル毎に独立して単相のパルスをカウントする手順を示します。

① *TWX_PCWriteReg()* 関数を呼び出し、使用するチャンネルでカウントするエッジを選択します。

② *TWX_PCSetCnt()* 関数を使用して、使用するチャンネルのカウンタをクリアします。

③ *TWX_PCStart()* 関数を使用して、使用するチャンネルのカウントを開始します。

④ カウンタの値を調べるために *TWX_PCReadCnt()* 関数を呼び出します。

⑤ カウンタを停止するには *TWX_PCStop()* 関数を呼び出します。

C 言語の例

```
long Count;

/*チャンネル 0 の単相カウント*/
TWX_PCWriteReg(hDev, 0x01);      /*PC0 の立ち上がりをカウント*/
TWX_PCSetCnt(hDev, TWX_PC0, 0); /*カウンタの値をクリア*/
TWX_PCStart(hDev, TWX_PC0);      /*0 チャンネルのカウントを開始*/

/*...*/

TWX_PCReadCnt(hDev, TWX_PC0, &Count); /*現在のカウント値を取得*/

/*...*/

TWX_PCStop(hDev, TWX_PC0); /*カウント終了*/
```

VisualBasic6.0 の例

```
Dim Count As Long

'チャンネル 0 の単相カウント
TWX_PCWriteReg hDev, &H1        'PC0 の立ち上がりをカウント
TWX_PCSetCnt hDev, TWX_PC0, 0  'カウンタの値をクリア
TWX_PCStart hDev, TWX_PC0       '0 チャンネルのカウントを開始

'...

TWX_PCReadCnt hDev, TWX_PC0, Count '現在のカウント値を取得

'...

TWX_PCStop hDev, TWX_PC0 'カウント終了
```

2 相パルスをカウントする

市販のロータリーエンコーダなどの 2 相出力をカウントする方法を示します。このモードでは 2 つのチャンネルを 1 組として使用します。PC0 と PC1、または PC2 と PC3 の組み合わせを指定することができます。PC0 と PC1 を使用して 2 相のパルス入力をカウントするときの様子を図 23 に示します。

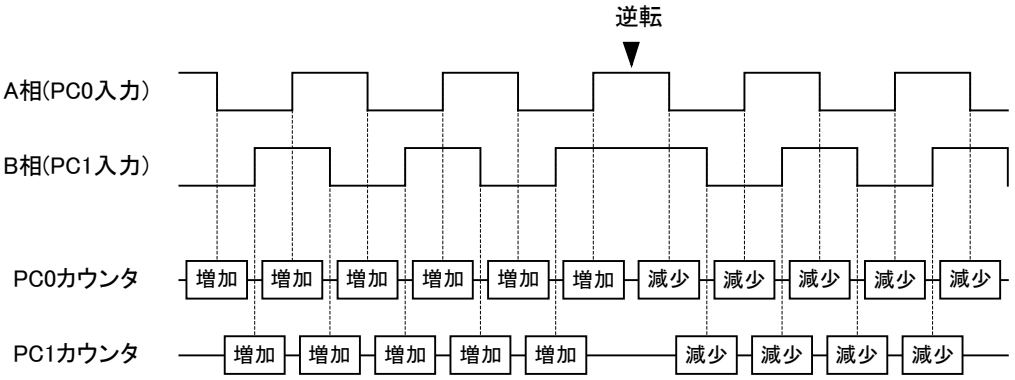


図 23　2 相パルスのカウント

① *TWX_PCSetMode()* 関数を呼び出します。引数 *Mode* には *TWX_PC_2PHASE* を指定します。引数 *CHBits* で使用するチャンネルを指定できます。カウンタのクリアと、エッジの選択は自動的に行われますので設定する必要はありません。

② *TWX_PCStart()* 関数を使用して、使用するチャンネルのカウントを開始します。

③ カウンタの値を調べるために *TWX_PCReadCnt()* 関数を呼び出します。

④ カウンタを停止するには *TWX_PCStop()* 関数を呼び出します。

C 言語の例

```
long Count;

/*チャンネル 0,1 の 2 相カウント*/
TWX_PCSetMode(hDev, TWX_PC_2PHASE, TWX_PC0_PC1); /*PC0,PC1 を 2 相カウントに設定*/
TWX_PCStart(hDev, TWX_PC0_PC1); /*PC0,PC1 のカウントを開始*/

/*...*/

TWX_PCReadCnt(hDev, TWX_PC0_PC1, &Count); /*現在のカウント値(PC0 と PC1 の合計値)を取得*/

/*...*/

TWX_PCStop(hDev, TWX_PC0_PC1); /*カウント終了*/
```

VisualBasic6.0 の例

```
Dim Count As Long

'チャンネル 0,1 の 2 相カウント
TWX_PCSetMode hDev, TWX_PC_2PHASE, TWX_PC0_PC1  ' PC0,PC1 を 2 相カウントに設定
TWX_PCStart hDev, TWX_PC0_PC1 'PC0,PC1 のカウントを開始

'...

TWX_PCReadCnt hDev, TWX_PC0_PC1, Count '現在のカウント値(PC0 と PC1 の合計値)を取得

'...

TWX_PCStop hDev, TWX_PC0_PC1 'カウント終了
```

□ シリアルポート

製品では RS-232C 準拠のシリアル通信チャンネルを 1 つ利用可能です。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 SCI 用のステータスレジスタにエラーをセットし、オーバーフローしたデータは捨てられます。表 33 に SCI で使用する関数をあげます。

ここで使用する「USBM_」で始まる関数の詳細については別紙「USBM ライブラリ関数リファレンス」を参照してください。

表 33　SCI で使用する関数

関数名	説明
USBM_SCISetMode()	通信条件の設定を行います。
USBM_SCIReadStatus()	SCI のエラー、受信バイト数を読み出します。
USBM_SCIRead()	SCI から指定バイト数のデータを読み出します。
USBM_SCIWrite()	SCI からデータを送信します。
USBM_SCISetDelimiter()	デリミタ文字を指定します。

デリミタ文字を指定しておくと、*USBM_SCIRead()* 呼び出したときにデバイス側でデリミタ文字をチェックし、発見した場合は受信データが指定バイト数に達していなくても、残りのデータを 0 で埋めて処理を戻します。

データを送信する

① *USBM_SCISetMode()* 関数でボーレート、データビット数、パリティ、ストップビットなどを設定します。

② *USBM_SCIWrite()* 関数で送信したいデータを送ります。

データを受信する

① *USBM_SCISetMode()* 関数でボーレート、データビット数、パリティ、ストップビットなどを設定します。

② 受信データ長が不定で、デリミタ文字によってパケットの区切りを識別する場合には、*USBM_SCISetDelimiter()* 関数を使用します。

③ *USBM_SCIRead()* 関数でデータを受信します。

C 言語の例

```
char Data[6] = “Hello”;

USBM_SCISetMode(hDev,0,USBM_SCI_DATA8 | USBM_SCI_NOPARITY | USBM_SCI_STOP1,
                USBM_SCI_BAUD38400);  /*モードを設定*/

USBM_SCIWrite(hDev,0,Data,6); /*シリアルポートから出力*/
USBM_SCIRead(hDev,0,Data,6,NULL);  /*シリアルポートから読み出し*/
```

VisualBasic6.0 の例

```
Dim Data(0 To 5) As Byte

Data(0) = Asc("H")
Data(1) = Asc("e")
Data(2) = Asc("l")
Data(3) = Asc("l")
Data(4) = Asc("o")
Data(5) = Asc(vbNullChar)

USBM_SCISetMode hDev, 0, (USBM_SCI_DATA8 Or USBM_SCI_NOPARITY Or USBM_SCI_STOP1), _
USBM_SCI_BAUD38400  'モードを設定

USBM_SCIWrite hDev, 0, Data, 6  'シリアルポートから出力
USBM_SCIRead hDev, 0, Data, 6, 0   'シリアルポートから読み出し
```

7. 応用プログラミング

□ 付属ライブラリについて

製品には「TWX3069.dll」、「USBM3069.dll」の 2 つのライブラリファイルが付属しています。それぞれのライブラリの関係は右の図のようになっています。

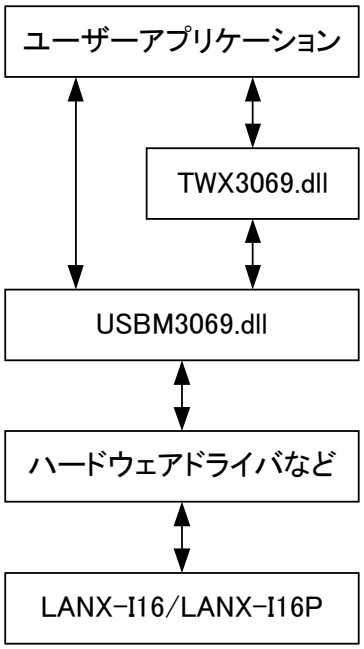


図 24　ライブラリの階層図

「TWX3069.dll」（以下、TWX ライブラリ）には「TWX_」で始まる関数が含まれます。このライブラリは製品の基本的な機能だけを抽出し、比較的簡単なプログラミングでご使用いただけるように構成されています。

「USBM3069.dll」（以下、USBM ライブラリ）には「USBM_」で始まる関数が含まれます。このライブラリは TWX ライブラリと比較して低水準な関数で構成され、より多くの機能をサポートしています。

単純な I/O のみのプログラムの場合には TWX ライブラリのみでプログラミング可能ですが、より高機能なアプリケーションを作成するためには USBM ライブラリが必要になります。本章では主に USBM ライブラリの使用方法について説明していきます。

TWX ライブラリの詳細は「TWX ライブラリ関数リファレンス」（66 ページ）を参照してください。USBM ライブラリの詳細については別紙「USBM ライブラリ関数リファレンス」を参照してください。

□ ハードウェアについて

本製品は内蔵するワンチップマイコンにより制御されています（図 25 参照）。提供される機能の多くはマイコンに集積された I/O ポート、メモリ、タイマ、DMA コントローラなどのハードウェアを利用することで実現されています。デジタル回路だけでなく、AD コンバータや DA コンバータもマイコンに搭載された機能の一部です。CPU を搭載しているため、様々な機能を比較的容易に提供でき、しかもそれぞれの機能を連携させて動作することも可能となっています。さらに、ある程度自立的に動作できることからホストパソコンとの通信によるオーバーヘッドを減らすことができます。

しかし反面、複数の機能でハードウェアリソースをシェアするために、「ある機能を使用している場合は、別のある機能は一部しか使用できない」というような制限があります。また、マイコン上のソフトウェア動作による速度的な限界もあります。このような特徴を把握してご利用いただくことで、製品の持つ機能を一層有効にご活用いただけます。

以下に本製品で使用するマイコン機能の概略を説明します。

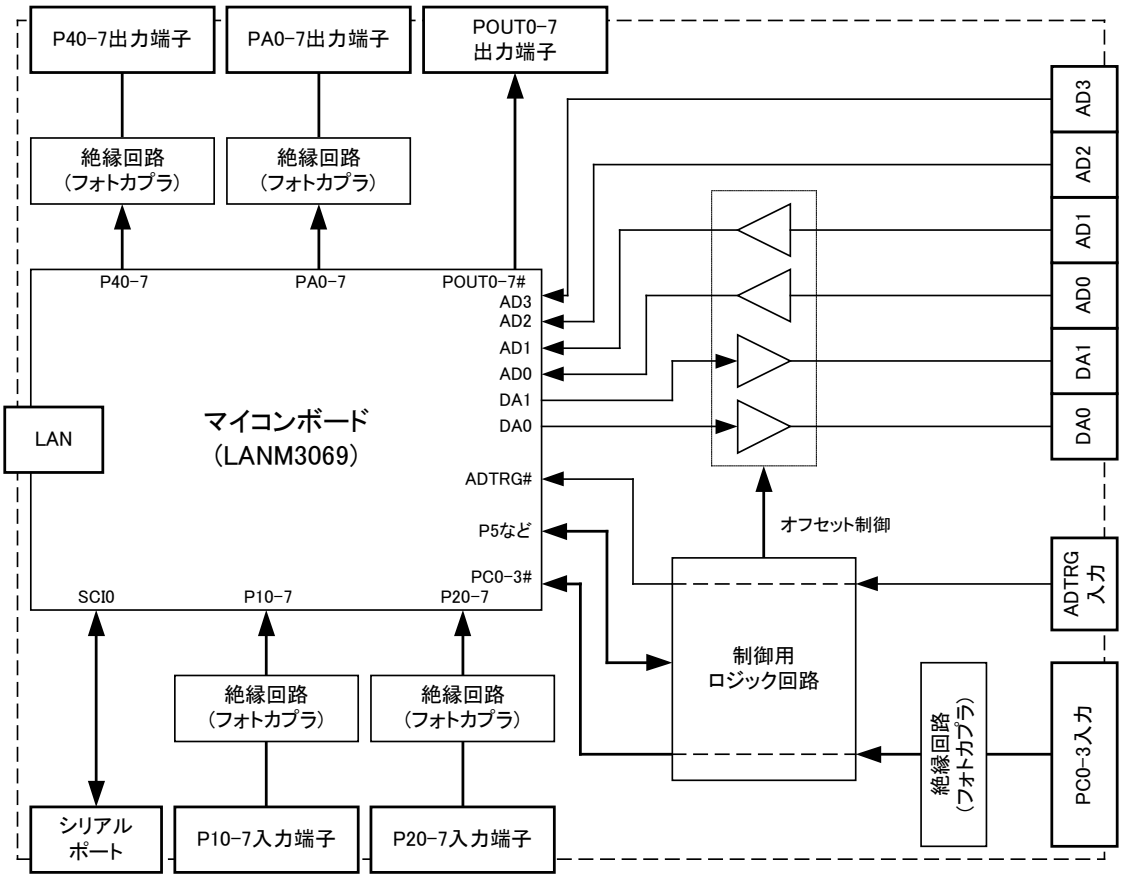


図 25　内部ブロック図

ユーザーメモリ

マイコンの内部メモリのうちユーザーに開放されているエリアのことです。以下のような用途に使用することができます。

- AD コンバータの変換結果の格納
- DA コンバータに転送するデータの格納
- タイマコピーで出力ポートに転送するデータの格納

マイコンのメモリ番地で H'FFBF20～H'FFE71F までの 10Kbyte が利用できます。ユーザーメモリへのアクセスは *USBMPortBRead()*、*USBM_PortBWrite()* などの関数で行うことができます。関数呼び出し時には *Port* 引数に読み書きを行うアドレスを指定してください。

8 ビットタイマ

8 ビットカウンタと比較用のコンペアレジスタを備えたタイマです。8 ビットカウンタはクロックが入力される毎に 1 ずつインクリメントされます。予め設定されたコンペアレジスタの値と 8 ビットカウンタの値が一致すると、“コンペアマッチ”と呼ばれるイベント（割り込み）が発生します。製品では周期的な動作が必要な場合にこの機能を利用しています。8 ビットタイマは以下の用途で使用されます。

- AD コンバータの変換周期の生成
- タイマコピーの転送周期の生成

上のそれぞれの機能には独立したチャンネルが割り当てられているので、並列して動作させることができます。また、入力クロックは“3125kHz”、“390.635kHz”、約“3052Hz”から選択できます。

16 ビットタイマ

16 ビットカウンタと比較用のコンペアレジスタを備えたタイマです。8 ビットタイマと同様に動作し、やはりコンペアマッチを発生します。16 ビットタイマは以下の用途で使用されます。

- DA コンバータへのデータ転送周期の生成

2 チャンネルの DA コンバータに独立のチャンネルが割り当てられているので、並列して動作させることができます。また、入力クロックは“25MHz”、“12.5MHz”、“6250kHz”、“3125kHz”から選択できます。

DMA コントローラ

マイコン内のメモリやレジスタ間でのデータ転送を自動的に行うハードウェアです。DMA コントローラは以下の用途で使用されます。

- AD コンバータの変換結果の転送（*USBM_ADCopy()* 関数を使用する場合）
- DA コンバータへのデータ転送

DMA コントローラは 2 チャンネル搭載されますが、AD コンバータで 1 チャンネル、DA コンバータで最大 2 チャンネル使用しますので全て同時には使用できません。

割り込み

一部の機能はマイコンの割り込みを利用しています。割り込みはマイコン内のプログラムで 1 つずつ処理されるので、複数の機能で同時に割り込みが発生した場合には期待どおりのタイミングで動作しない場合や、誤動作する場合があります。割り込みは以下の機能で使用しています。

- パルスカウンタへのパルス入力
- タイマコピーの転送時
- シリアルポートへのデータ入力

□ AD コンバータ

アナログ入力信号をある周期で定期的にサンプリングする方法や、外部トリガ入力を使用する方法、高速にサンプリングする方法などを説明します。アナログ入力の基本的な使用方法に関しては「アナログ入出力」(33 ページ)を参照してください。

表 34　AD コンバータで使用する関数

関数名	説明
USBM_ADRead()	AD 変換を 1 回行い、結果を返します。
USBM_ADSetCycle()	連続して AD 変換を行う場合の変換周期を設定します。
USBM_ADBRead()	指定回数の AD 変換を連続して行い、結果を返します。
USBM_ADStart()	指定回数の AD 変換を連続して行います。この関数では変換と読み出しを非同期に行えます。
USBM_GetQueueStatus()	*USBM_ADStart()* で変換した結果が USB のリードバッファに何バイトあるか調べます。
USBM_Read()	*USBM_ADStart()* で変換した結果を USB のリードバッファから読み出します。
USBM_Abort()	*USBM_ADStart()* での変換を中止する場合や、*USBM_ADBRead()* がタイムアウトした場合に使用します。
USBM_Purge()	リードバッファをクリアするのに使用します。
USBM_ReadStatus()	*USBM_ADBRead()*、または *USBM_ADStart()* による AD 変換中に、変換データが正しく転送されたかどうかを知るのに使用します。
USBM_ADCopy()	指定回数の AD 変換を連続して行い、デバイス上のメモリに結果を保存します。変換速度が他の関数よりも高速です。
USBM_ADReadCopyStatus()	*USBM_ADCopy()* の進行状況を読み出します。
USBM_ADReadBuffer()	*USBM_ADCopy()* で変換した結果を、デバイス上のメモリから読み出します。
USBM_ADStopCopy()	*USBM_ADCopy()* による変換を終了します。

USBM ライブラリには AD コンバータを制御する関数が多数用意されています。表 34 はそれら関数の一覧です。ここであげた関数では変換結果は図 26 のように 16 ビット変数の上位 10 ビットに納められ、下位 6 ビットは常に 0 となります。*TWX_ADRead()* 関数とデータの格納方法が異なりますのでご注意ください。

変換結果は、符号無し整数として返されますので、Visual Basic で開発される場合は、ヘルパー関数を使用して適宜 *Long* 型変数に変換してください。

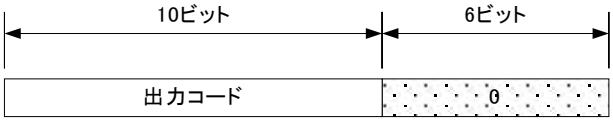


図 26　変換結果の格納方法

AD 変換結果を得る方法は大きく分けて 4 つの方法があります。

- 単純に命令発行時のアナログ電圧値を読み出す、*USBM_ADRead()* 関数を使用する方法。
- 8 ビットタイマまたは外部トリガに同期して連続で変換結果を得る *USBM_ADBRead()* 関数を用いる方法。
- 8 ビットタイマまたは外部トリガに同期して連続変換した結果をホストパソコンでバッファリングし、逐次データを取り出せる *USBM_ADStart()* 関数を使用する方法。
- 最高速度で連続変換した結果を、DMA を用いてデバイス内のメモリにバッファリングする *USBM_ADCopy()* 関数を使用する方法。

表 35 は、それぞれの変換方法の特徴をまとめたものです。

表 35　AD 変換の方法と特徴

代表関数名	変換レート	プログラム	複数チャンネル*1	逐次読出し	特徴
USBM_ADRead()	低(数 msec)	容易	可	–	使い方が簡単ですが、変換レートが使用環境に依存します。直流向き。
USBM_ADBRead()	中(60 μ sec)	容易	不可	不可	使い方が簡単ですが、複数のチャンネルのスキャンができません。
USBM_ADStart()	中(60 μ sec)	やや複雑	可	可	複数チャンネルのスキャンもでき、変換結果を逐次取り出せます。
USBM_ADCopy()	高(2.8 μ sec/ch)*2	やや複雑	可	不可	変換レートが最高で、変換中のデバイスアクセス可能です。

*1 全く同時に変換できるのは 1 チャンネルです。複数チャンネルの場合、スキャンモードを使用した順次スキャンになります。
*2 変換レートの設定はできません。常に最大のレートで変換します。

USBM_ADRead() を使用する(命令毎に変換)

USBM_ADRead() 関数を使用します。複数のチャンネルを同時に読み出すこともできます。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、デバイスは指定チャンネルの AD 変換を行い、ホストパソコンに変換結果を返します。

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

全ての変換方法に共通してマイコンの AD 変換回路は、完全に同時に複数のアナログ信号をサンプリングすることはできません。同時にサンプリングおよび変換が可能になるのは常に 1 チャンネルのみです。図 27 は 3 チャンネルの変換を行ったときの様子を示します。図中の変換時間 t_c は最初の 1 チャンネルが最大 5.36 μ sec、残りのチャンネルは 5.12 μ sec となります。

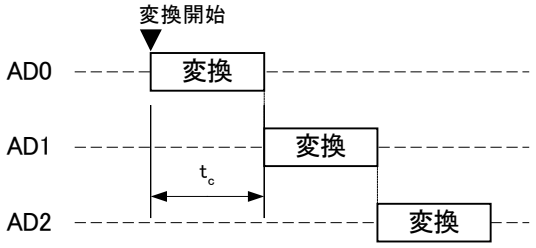


図 27　複数チャンネルの AD 変換の様子

C 言語の例

```
WORD ADData[4];

USBM_ADRead(hDev,ADData,3,TRUE);          /*0-3 チャンネル全て読み出し*/
```

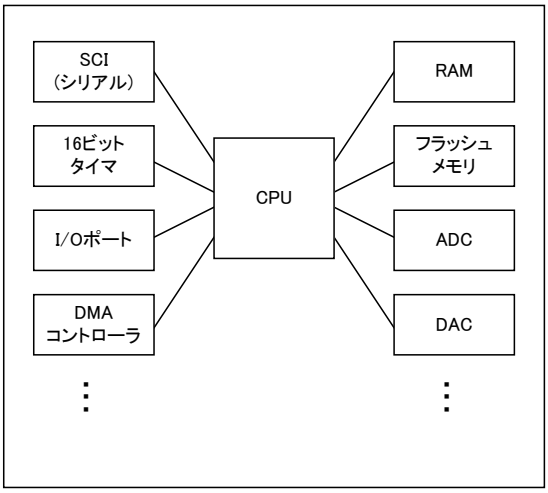
VisualBasic6.0 の例

```
Dim ADData(3) As Integer
Dim i As Integer

USBM_ADRead hDev, ADData, 3, 1  '0-3 チャンネル全て読み出し

Dim ADData32(3) As Long

For i = 0 To 3
  ADData32(i) = USBM_ToINT32(ADData(i)) '符号無し整数を正しく読むために変換
Next i
```

搭載マイコンについて

製品には「H8/3069RF」(株式会社ルネサステクノロジ)というマイクロコントローラが搭載されています。このマイコンチップ内部には、CPU コアの他に、非常に豊富な周辺回路が内蔵されています。そのため、わずかな外付け回路と組み合わせることで、様々な製品に応用可能となっています。

H8/3069RF マイコン



付属のライブラリ関数とファームウェアは、マイコンが持つ多くの機能をできる限り簡単に使用できることを目的としています。単純な I/O 製品と比較してライブラリ関数が多いため、最初は戸惑われるかもしれませんが、必要な機能を絞り込んで動作をご理解いただければ、プログラミングは決して難しいものではありません。是非、お客様のアプリケーション開発にお役立てください。

付属の CD-ROM には「H8/3069R F-ZTAT™ ハードウェアマニュアル」も収録しております。本マニュアルと合わせてご参照ください。

USBM_ADBRead() を使用する(連続で変換)

USBM_ADBRead() 関数を使用すると、予め設定した変換レートで連続サンプリングを行うことができます。変換タイミングはマイコンの 8 ビットタイマを利用して作られます(外部から入力することも可能)。変換周期の最小値は 60 μ sec です。

タイマコピー、パルスカウンタなどの割込みを利用する機能と同時に使用すると、変換が正しいタイミングで行われない可能性がありますのでご注意ください。

変換周期の設定値が短すぎる場合や、割り込みなどの処理により変換データに抜けが生じた可能性がある場合には、デバイスのステータスに *USBM_STS_TIMEOUT* のビットが立ちます。ステータスは *USBM_ReadStatus()* 関数で取得することができます。

USBM_ADBRead() 関数では複数チャンネルをサンプリングすることはできません。

① H8/3069RF の 8 ビットタイマを使用して AD 変換のタイミングを作るには、*USBM_ADSetCycle()* 関数を使用します。変換周期は以下のようになります。

$$Tc = (Cmp+1) / f_{clk}\ [\mathrm{sec}]$$ (f_{clk} : CLK で選択した周波数)

初期状態では外部トリガ信号(ADTRG#)によって変換を開始する設定となっています。

② *USBM_ADBRead()* 関数を呼び出すと 8 ビットタイマのコンペアマッチまたは ADTRG#の立下りの度に 1 回の変換を行い、指定回数終了した時点で関数からリターンします。連続変換で読み出すことのできるのは 0～3 チャンネルの中から指定した 1 チャンネルのみです。

C 言語の例

```
WORD Data[100];

USBM_ADSetCycle(hDev,99,USBM_TCLK390); /*約 3.9kHz で変換*/
USBM_ADBRead(hDev,Data,100,0,NULL);    /*チャンネル 0 を 100 回サンプリング*/
```

VisualBasic6.0 の例

```
Dim Data(99) As Integer
Dim i As Integer

USBM_ADSetCycle hDev, 99, USBM_TCLK390 '約 3.9kHz で変換
USBM_ADBRead hDev, Data, 100, 0, 0     'チャンネル 0 を 100 回サンプリング

Dim Data32(99) As Long

For i = 0 To 99
  Data32(i) = USBM_ToINT32(Data(i))  '符号無し整数を正しく読むために変換
Next i
```

USBM_ADStart() を使用する(変換しながらデータを取り出す)

USBM_ADStart() 関数も *USBM_ADBRead()* 関数の場合と同様に、予め設定した変換レートで連続してサンプリングを行うことができます。最小変換時間は 60μsec です(複数チャンネル使用時も同じ)。*USBM_ADStart()* を呼び出す場合に *lByte* 引数を *TRUE* とすると変換結果の上位8ビットのみをホストパソコンに送信します。この場合データ量は半分になりますが変換時間には影響しません。

USBM_ADStart() を呼び出すとデバイスは、AD 変換を開始しますが、関数自体はすぐにリターンします。変換結果はデバイスからホストパソコンへ送られ、パソコン上のメモリにバッファリング [9]されます。変換中ホストパソコンのプログラムはブロックされませんので、他の非同期関数を呼び出した場合と同様に、メッセージ処理や画面描画などを行うことができます。ただし、デバイス自身は *USBM_Abort()* による中断コマンド以外を受け付けませんのでご注意ください。

バッファに溜まったデータのバイト数を知るには *USBM_GetQueueStatus()* 関数を、データを取り出すには *USBM_Read()* 関数を呼び出してください(これらの関数はデバイスにコマンドを送らないので呼び出し可能です)。

8ビットタイマで変換周期を作る場合で、*USBM_ADStart()* 呼び出し時に *Trig* 引数に *TRUE* を指定すると、ADTRG#入力により、タイマが起動される設定となります(図 28 参照)。この場合、ADTRIG#信号は1回だけで良く、AD コンバータ自体はタイマから起動されます。図中の t_d は ADTRG#が入力されてからタイマが起動されるまでの遅延時間で 1.8μsec～3μsec の間で変化します。T_C は *USBM_ADSetCycle()* で設定した変換周期です。

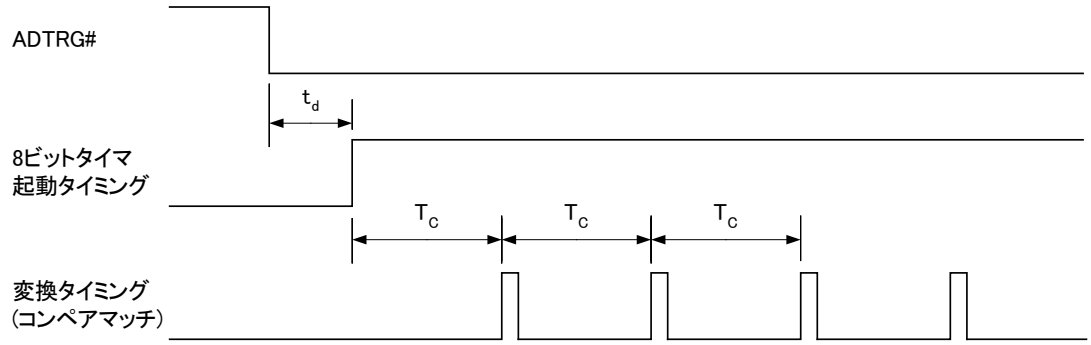


図 28 *USBM_ADStart()* の *Trig* 指定時の変換タイミング

USBM_ADBRead() の場合と同様、タイマコピー、パルスカウンタなどの割り込みの影響を受けます。また、ステータスも同じように *USBM_ReadStatus()* 関数で取得することができます。

① H8/3069RF の8ビットタイマを使用して AD 変換のタイミングを作るには、*USBM_ADSetCycle()* 関数を使用します。変換周期は以下のようになります。

$$Tc = (Cmp+1) / f_{clk}\ [\mathrm{sec}]\quad (f_{clk}: CLK\text{で選択した周波数})$$

初期状態では外部トリガ信号(ADTRG#)によって変換を開始する設定となっています。

② *USBM_ADStart()* 関数を呼び出すと、デバイスは 8 ビットタイマのコンペアマッチまたは ADTRG#の立下りの度に1回の変換を行い、結果をホストパソコンに送信します。*Trig* を *TRUE* とした場合は、ADTRG#が入力されるまでタイマの起動を待機します。

[9] 8K バイトまでバッファできます。

③ *USBM_GetQueueStatus()* 関数を使用して受信したデータのバイト数を取得します。

④ 必要な量のデータがバッファリングされたら、*USBM_Read()* 関数で読み出します。複数チャンネルの変換の場合には、データは AD0、AD1、AD2、AD3、AD0、・・・のように順番に読み出されます。各データのサイズは *USBM_ADStart()* の *lByte* 引数により、1 バイト、または、2 バイトとなります。

⑤ 中断する場合には *USBM_Abort()* 関数を呼び出してください。受信バッファにデータが残っている場合は、*USBM_Read()* 関数で全て読み出すか、*USBM_Purge()* 関数で受信バッファをクリアしてください。

- 受信バッファ内に AD 変換データが残っていると、以降のデバイス制御が不能になりますので、必ずクリアするようにしてください。

C 言語の例

```
WORD wBuff[1024];
DWORD n;

USBM_ADSetCycle(hDev,38,USBM_TCLK390);     /* 約 10kHz で変換 */
USBM_ADStart(hDev,-1,3,TRUE,FALSE,FALSE); /* 停止するまで全チャンネル変換 */
while(1){
  USBM_GetQueueStatus(hDev,&n);  /* 変換されたデータ数を読み出す */
  if(n >= 2048) break;           /* 2048 バイト(1024 ワード)変換されたら抜ける*/
}
USBM_Read(hDev,wBuff,2048,&n); /* 変換結果の読み出し */
USBM_Abort(hDev);              /* 変換の終了 */
USBM_Purge(hDev,USBM_PURGE_RX);  /* 受信バッファをクリア */
```

VisualBasic6.0 の例

```
Dim wBuff(1023) As Integer
Dim n As Integer
Dim i As Integer

USBM_ADSetCycle hDev, 38, USBM_TCLK390    '約 10kHz で変換
USBM_ADStart hDev, -1 , 3, 1, 0, 0 '停止するまで全チャンネル変換

Do
  USBM_GetQueueStatus hDev, n  '変換されたデータ数を読み出す */
  If n >= 2048 Then Exit Do    '2048 バイト(1024 ワード)変換されたら抜ける
Loop

USBM_Read hDev, wBuff, 2048, n '変換結果の読み出し

USBM_Abort hDev               '変換の終了
USBM_Purge hDev, USBM_PURGE_RX  '受信バッファをクリア

Dim lBuff(0 To 1023) As Long

For i = 0 To 1023
  lBuff(i) = USBM_ToINT32(wBuff(i)) '符号無し整数を正しく読むための変換
Next i
```

USBM_ADCopy() を使用する(最大レートで変換する)

USBM_ADCopy() 関数は AD コンバータの最大の変換レートでサンプリングを行うことができます。サンプリングしたデータはユーザーメモリに蓄えられます。

常に最大レートで変換を行うため、*USBM_ADSetCycle()* 関数は無効です。変換開始のトリガ信号は ADTRG 端子より入力します。図 29 に *USBM_ADCopy()* を使用した場合の、変換の様子を示します。厳密には ADTRG 信号はマイコン内部でサンプリングされ、マイコンの内部クロックに同期して変換が開始されます。

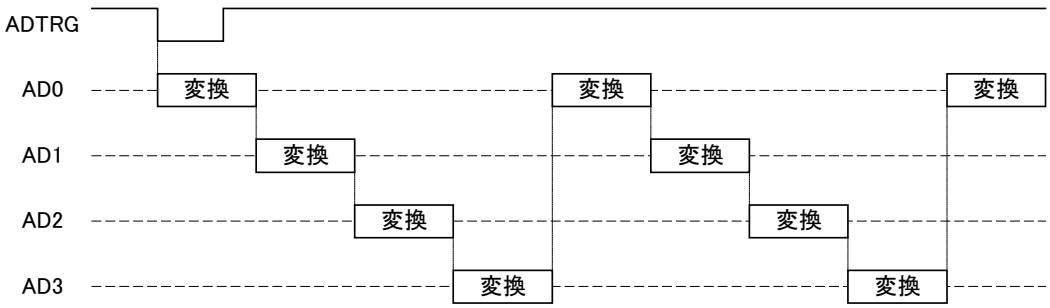


図 29　*USBM_ADCopy()* の変換の様子

また、変換は常に AD0 から開始され、指定の終了チャンネルまで行われます。1 チャンネルのみの変換を行う場合には、常に AD0 を使用する必要がありますのでご注意ください。

USBM_ADCopy() 関数を呼び出す際に *CKS* に *TRUE* を指定すると、変換ステート数を減らして、高速に変換ができるようになります。変換精度は低下しますが(表 36 参照)、精度よりも速度を優先する場合に使用します。

表 36　CKS の設定による変換特性の相違

項目		min	max	単位
変換時間: 134 ステート(CKS=0)	変換時間(単一モード)		5.36	μsec
	変換時間(連続変換中)	5.12	5.12	μsec
	許容信号源インピーダンス[10]		5	kΩ
	非直線性誤差		±3.5	LSB
	オフセット誤差		±3.5	LSB
	フルスケール誤差		±3.5	LSB
	量子化誤差		±0.5	LSB
	絶対精度		±4.0	LSB
変換時間: 70 ステート(CKS=1)	変換時間(単一モード)		2.8	μsec
	変換時間(連続変換中)	2.64	2.64	μsec
	許容信号源インピーダンス		3	kΩ
	非直線性誤差		±7.5	LSB
	オフセット誤差		±7.5	LSB
	フルスケール誤差		±7.5	LSB
	量子化誤差		±0.5	LSB
	絶対精度		±8.0	LSB

[10] AD2、AD3 のみ適用。AD0、AD1 の入力インピーダンスは 10MΩ以上です。

変換データのメモリへの転送は DMA が使用されます。その為、変換動作中であっても、他の関数を呼び出して操作を行うことができます。ただし、他の用途で同一の DMA チャンネルが使用されないように注意が必要です。

指定された回数の変換が終了したかどうかを調べるためには *USBM_ADReadCopyStatus()* 関数を使用します。また、変換終了時、もしくは変換を中断する場合には *USBM_ADStopCopy()* 関数を呼び出してください。

① *USBM_ADCopy()* 関数を呼び出します。

② ADTRG 端子に信号が入力されると変換が開始されます。変換が終了したかどうかを調べるためには、*USBM_ADReadCopyStatus()* 関数を呼び出してください。

③ 指定回数の変換が終了したら *USBM_ADReadBuffer()* 関数を使用して結果を呼び出します。

④ *USBM_ADStopCopy()* 関数で終了処理をします。中断する場合にも、*USBM_ADStopCopy()* 関数を使用します。

C 言語の例

```
WORD wBuff[1024];
long n;

USBM_ADCopy(hDev,USBM_USER_AREA,256,3,FALSE,0,TRUE); /* 0-3 チャンネルを 256 回変換 */
while(1){ /* 変換が終わるまでループ(ADTRG が入力されるまで変換開始されません) */
  USBM_ADReadCopyStatus(hDev,&n,0); /* 残りの変換数を調べる */
  if(n == 0) break; /* 変換が終わっていたら抜ける */
}
USBM_ADReadCopyBuffer(hDev,USBM_USER_AREA,wBuff,1024,TRUE,FALSE);
USBM_ADStopCopy(hDev,0);
```

VisualBasic6.0 の例

```
Dim wBuff(1023) As Integer
Dim n As Long
Dim i As Integer

USBM_ADCopy hDev, USBM_USER_AREA, 256, 3, 0, 0, 1 '0-3 チャンネルを 256 回変換
Do '変換が終わるまでループ(ADTRG が入力されるまで変換開始されません)
  USBM_ADReadCopyStatus hDev, n, 0 '残りの変換数を調べる
  If n = 0 Then Exit Do '変換が終わっていたら抜ける
Loop
USBM_ADReadCopyBuffer hDev, USBM_USER_AREA, wBuff, 1024, 1, 0
USBM_ADStopCopy hDev, 0

Dim lBuff(1023) As Long

For i = 0 To 1023
  lBuff(i) = USBM_ToINT32(wBuff(i)) '符号無し整数を正しく読むために変換
Next i
```

□ DA コンバータ

内蔵 16 ビットタイマと DMA を利用して、ハードウェアのみで高速に DA コンバータ出力を変化させる方法を説明します。この方法は、予め設定した波形パターンを再生するような用途に向いています。データは DMA で自動的に転送されるため、デバイスは変換中であっても、他の命令を処理することが可能です。この機能を利用した場合、DA コンバータ 1 チャンネルにつき、16 ビットタイマと DMA を 1 チャンネルずつ使用します。16 ビットタイマと DMA は使用する DA と同一のチャンネルが使用されます。例えば、DA0 を利用する際は、タイマと DMA も 0 チャンネルが使用できなくなります。

表 37　DA コンバータで使用する関数

関数名	説明
USBM_PortWrite8()	DA コンバータに出力値を設定します。
USBM_PortBWrite()	連続して DA 変換する場合の変換データをデバイス上のメモリに転送します。
USBM_DASetCycle()	DA コンバータの変換周期を設定します。
USBM_DASetParm()	連続して DA 変換する場合のパラメータを設定します。
USBM_DAReadStatus()	連続して DA 変換する場合の未変換のデータ数を調べます。
USBM_DAStart()	連続 DA 変換を開始します。
USBM_DAStop()	連続 DA 変換を終了します。

DMA を使用して高速に変換する

① DA 変換するデータをユーザーメモリの任意の位置に *USBM_PortBWrite()* 関数で書き込んでおきます。

② *USBM_DASetCycle()* 関数を使用して DA コンバータの変換サイクルを決定します。

③ *USBM_DASetParm()* 関数を使用してパラメータを設定します。ソースアドレスには①でデータを書き込んだアドレスを指定してください。また、*lLoop* を *TRUE* とすることで中断を指示するまで、繰り返し変換を行うことが可能ですが、その場合、*nData* に指定できるデータ数が 255 に制限されますのでご注意ください。

④ *USBM_DAStart()* 関数を呼び出して変換を開始します。*USBM_DAReadStatus()* で残りのデータ数を読み出すことができます。

⑤ 終了処理のために *USBM_DAStop()* 関数を呼び出してください。また、中断する場合にもこの関数を用います。

C 言語の例

```
int i;
BYTE buff[255];

/* のこぎり波を出力します */
for(i=0;i<255;i++) buff[i] = (BYTE)i; /* 変数を初期化 */

USBM_PortBWrite(hDev,USBM_USER_AREA,buff,255,TRUE,FALSE); /*ユーザーメモリに書き込み*/

USBM_DASetCycle(hDev,0,249,USBM_TCLK25000); /* 変換周期を 10us に設定 */
USBM_DASetParm(hDev,0,USBM_USER_AREA,255,TRUE); /* 255 バイトのデータを繰り返し変換 */
USBM_DAStart(hDev,0x01); /* 開始 */

/* ... */

USBM_DAStop(hDev,0x01); /* 終了 */
```

VisualBasic6.0 の例

```
Dim i As Integer
Dim buff(254) As Byte

'のこぎり波を出力します
For i = 0 To 254
  buff(i) = i '変数を初期化
Next i

USBM_PortBWrite hDev, USBM_USER_AREA, buff, 255, 1, 0 'ユーザーメモリに書き込み

USBM_DASetCycle hDev, 0, 249, USBM_TCLK25000 '変換周期を 10us に設定
USBM_DASetParm hDev, 0, USBM_USER_AREA, 255, 1 '255 バイトのデータを繰り返し変換
USBM_DAStart hDev, &H1 '開始

'...

USBM_DAStop hDev, &H1 '終了
```

□ パルスカウンタ

パルスカウンタは、そのカウント方法を他の入力ポートの状態で制御する、ある値になったときにカウント値を自動的にクリアする、などといった細かな設定が可能です。また、単にパルス数をカウントするという使い方の他に、カウンタ値がある値に達したときに、出力ポートの値を変化させる“コンペアアウト”という機能も備えています。ここでは、それらの設定方法について説明します。

表 38　パルスカウンタで使用する関数

関数名	説明
USBM_PCSetCmp()	コンペアレジスタの値を設定します。
USBM_PCSetCnt()	カウンタの値を設定します。*TWX_PCSetCnt()* 関数と機能は同じですが、チャンネルを 0～3 の番号で指定する点にご注意ください。
USBM_PCReadCnt()	カウンタの値を読み出します。*TWX_PCReadCnt()* 関数とほぼ同様の機能です。全チャンネルのカウント数を 1 度で読むことができます。チャンネルを 0～3 の番号で指定する点にご注意ください。
USBM_PCSetControl()	パルスカウンタのカウントアップ/ダウンの方法、クリア条件を設定します。
USBM_PCSetCondBit()	パルスカウンタのアップ/ダウンを決定する条件ビットを指定します。
USBM_PCSetCmpOut()	コンペアマッチ時に出力を行うポートとデータを設定します。
USBM_PCStartA()	指定チャンネルのカウントをスタートします。*TWX_PCStart()* 関数と同じです。
USBM_PCStop()	指定チャンネルのカウントをストップします。*TWX_PCStop()* 関数と同じです。

パルスカウンタをスタートすると、チャンネルに対応する端子入力に従い、カウンタが 1 ずつ増加、または減少します。カウンタの増減は、*USBM_PCSetControl()* 関数で設定しますが、その際、*USBM_PCSetCondBit()* で指定された“コンディションビット”が参照されます。一般にコンディションビットには、入力ポートの任意のビットを指定します。このビットの状態によって、カウンタが入力パルスにより、増加するのか、減少するのか、パルスを無視するのかを決定することができます。例えば、ポート 1 の P10 端子をコンディションビットとして指定した場合、この端子が“Lo(OFF)”の場合はパルス入力でカウントアップ、“Hi(ON)”の場合は逆にカウントダウン、というような設定が可能になります。図 30 に *USBM_PCSetControl()* 関数の *Bits* 引数の意味を、表 39 に *Bits* 引数の設定値とカウンタの増減の関係を示します。

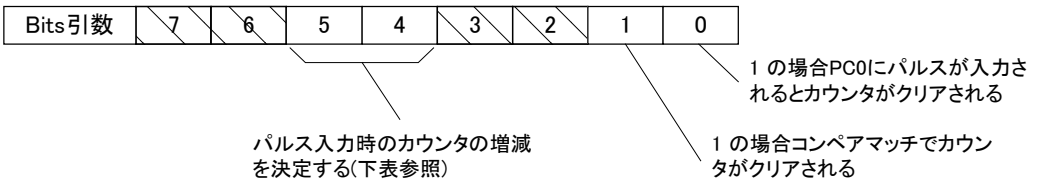


図 30　*USBM_PCSetControl()* の *Bits* 引数

表 39　*USBM_PCSetControl()* によるカウント方法の設定

USBM_PCSetControl() の引数(Bits)		コンディションビットによる増減	
ビット 5	ビット 4	コンディションビット = 0 のとき	コンディションビット = 1 のとき
0	0	カウントアップ	
0	1	カウントアップ	カウントダウン
1	0	カウントダウン	カウントアップ
1	1	カウントしない	カウントアップ

全てのチャンネルには 32 ビットのコンペアレジスタが用意されており、カウンタ値とコンペアレジスタの値が一致した場合に、1 つのポートに書き込み操作を行うことができます。この機能を“コンペアアウト”と呼びます。この機能を利用すると、外部からデバイスへの信号入力に対するフィードバックを素早く行うことができます。図 31 にチャンネル 0 へのパルス入力に応答して、ある出力ポートの値を変化させる例を示します。図中の t_d はパルス入力から実際にポート操作が行われるまでの遅延時間で、10 μ sec～80 μ sec(出力ポートの反応時間は含みません)となります。同じ処理を、ホストパソコンを介して行った場合、カウンタ値の読み出しに数百 μ sec、ポート出力に数百 μ sec の合わせて 1msec 以上の時間を要します。また、ネットワーク環境などの影響により時間は変化します。

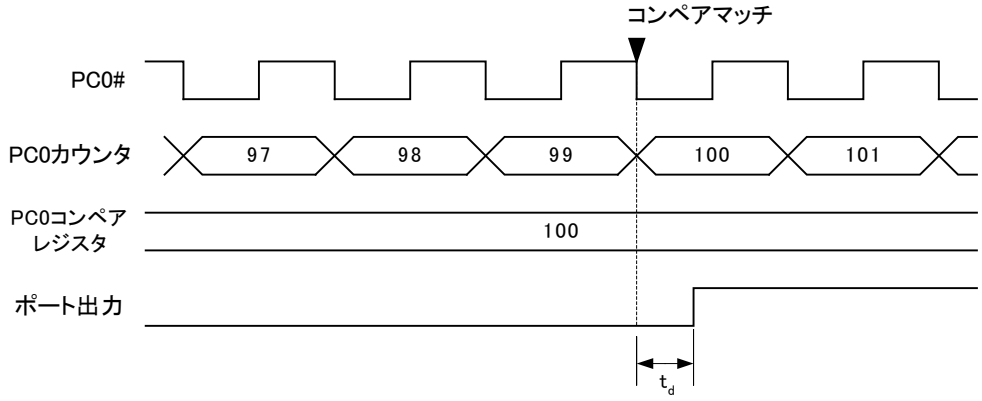


図 31　パルスカウンタのコンペアアウト

コンペアアウトを使用する

① *TWX_PCWriteReg()* 関数を呼び出し、使用するチャンネルでカウントするエッジを選択します。

② カウンタに初期値を設定する場合、またはリセットする場合には *TWX_PCSetCnt()* 関数を使用します。

③ *USBM_PCSetCmp()* 関数でコンペアレジスタを設定します。

④ *USBM_PCSetCmpOut()* 関数でコンペアマッチ時に出力するポートと値を設定します。

⑤ 必要な場合には *USBM_PCSetControl()* 関数でクリアの条件を設定します。初期設定ではクリアは自動的に行われない設定になっています。

⑥ *TWX_PCStart()* 関数でカウントを開始します。

⑦ 終了するには *TWX_PCStop()* 関数を呼び出します。

C 言語の例

```
/* PC0 入力で POUT を全て ON、PC1 入力で全て OFF にします */
TWX_PCWriteReg(hDev, 0x05); /* PC0 と PC1 の立ち上がりをカウント */
USBM_PCSetCmp(hDev, 0, 1);  /* PC0 と PC1 のコンペアレジスタを 1 に設定 */
USBM_PCSetCmp(hDev, 1, 1);
USBM_PCSetCmpOut(hDev, 0, USBM_POUT, 0xff); /* PC0 入力で POUT を全て ON */
USBM_PCSetCmpOut(hDev, 1, USBM_POUT, 0x00); /* PC1 入力で POUT を全て OFF */
USBM_PCSetControl(hDev, 0, 0x02); /*コンペアマッチでクリア(繰り返し入力可能にする) */
USBM_PCSetControl(hDev, 1, 0x02);
USBM_PCStartA(hDev, USBM_PC0 | USBM_PC1); /* カウント開始 */

/* ... */

USBM_PCStop(hDev, USBM_PC0 | USBM_PC1); /* 終了 */
```

VisualBasic6.0 の例

```
'PC0 入力で POUT を全て ON、PC1 入力で全て OFF にします
TWX_PCWriteReg hDev, &H5 'PC0 と PC1 の立ち上がりをカウント
USBM_PCSetCmp hDev, 0, 1 'PC0 と PC1 のコンペアレジスタを 1 に設定
USBM_PCSetCmp hDev, 1, 1
USBM_PCSetCmpOut hDev, 0, USBM_POUT, &HFF 'PC0 入力で POUT を全て ON
USBM_PCSetCmpOut hDev, 1, USBM_POUT, &H0  'PC1 入力で POUT を全て OFF
USBM_PCSetControl hDev, 0, &H2 'コンペアマッチでクリア(繰り返し入力可能にする)
USBM_PCSetControl hDev, 1, &H2
USBM_PCStartA hDev, USBM_PC0 Or USBM_PC1 'カウント開始

' ...

USBM_PCStop hDev, USBM_PC0 Or USBM_PC1 '終了
```

□ タイマコピー

製品には内蔵 8 ビットタイマを利用したタイマコピーという機能を実装しています。8 ビットタイマのコンペアマッチにより発生する割り込みを利用してポート間で(あるアドレスからあるアドレスへ)1 バイトずつデータをコピーすることができます。ユーザーメモリに予めデータを転送しておき、タイマコピーを起動することで、出力ポートの値を一定周期で更新する、といった使い方ができます。図 32 にパルスモーター用のパターンを POUT に出力する様子を示します。

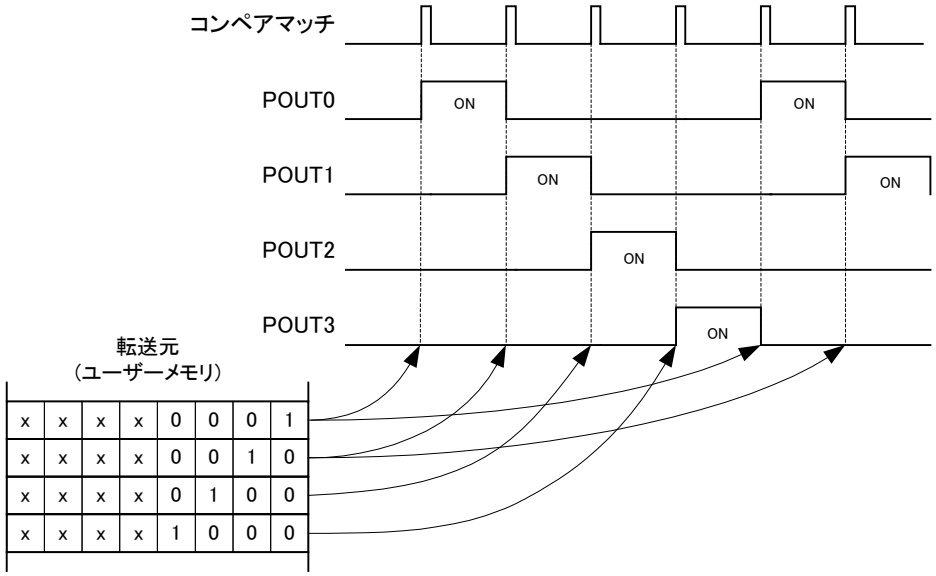


図 32　タイマコピーの動作の様子

タイマコピーでは 2 チャンネルが同時に使用できます。また、それぞれのチャンネルの起動、停止トリガとしてパルスカウンタの入力を指定することができます（図 33 参照）。

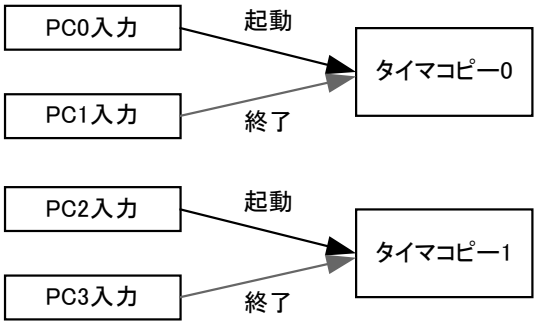


図 33　パルスカウンタ入力によるタイマコピーの操作

表 40　タイマコピーで使用する関数

関数名	説明
USBM_TCPYSetParm()	転送の設定を行います。
USBM_TCPYSetPatternCtrl()	出力ポートにパルスパターンを順次出力するよう設定します。
USBM_TCPYSetCycle()	転送周期を設定します。
USBM_TCPYSetTrig()	タイマコピーのトリガ信号を選択します。
USBM_TCPYReadStatus()	転送状況を読み出します。
USBM_TCPYStart()	タイマコピーを開始、終了します。

表 40 はタイマコピーで使用する関数です。*USBM_TCPYSetPatternCtrl()* 関数を使用すると、用意した転送パターンの途中から転送を開始したり、停止させたりということが可能となっています。*USBM_TCPYSetPatternCtrl()* 関数の *SrcPort* には転送パターンの先頭アドレス、*nSrc* は用意された転送パターンのバイト数です。*nCopy* は何回転送を行うかを設定します。*nCopy* 回の転送途中に転送元アドレスが転送パターンの数を超えると、自動的に転送元をパターンの先頭に戻して転送を続けます。逆に転送パターンの下限を超えた場合も同様に最後尾から転送を続けます。表 41 に *USBM_TCPYSetPatternCtrl()* 関数の引数の設定例、図 34 に、その場合の動作の様子を示します。転送先である POUT の出力値はコンペアマッチの発生とともに、図 34 の右のように変化します。

表 41　***USBM_TCPYSetPatternCtrl()* の設定例**

hDev	*CH*	*SrcPort*	*DstPort*	*nCopy*	*nSrc*	*SrcInc*	*Start*	*Mask*
xxx	xxx	0xffbf20	USBM_POUT	5	4	1	2	0xff

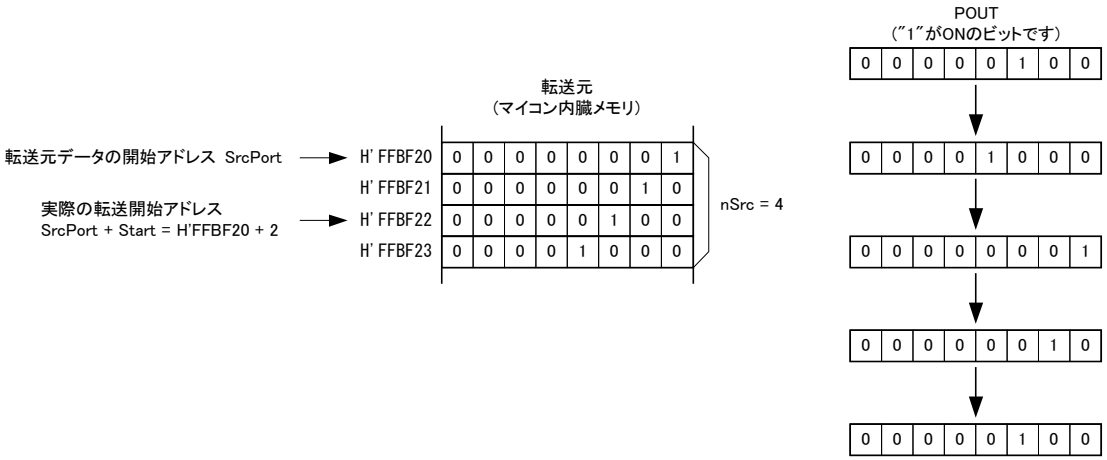


図 34　*USBM_TCPYSetPatternCtrl()* の設定例

タイマコピー機能を使用する

① *USBM_TCPYSetCycle()* 関数でコピーの周期を指定します。

② *USBM_TCPYSetParm()* 関数、または *USBM_TCPYSetPatternCtrl()* 関数で転送に関するパラメータを設定します。

③ パルスカウンタを起動トリガ、終了トリガとして利用する場合は、*USBM_TCPYSetTrig()* 関数を使用します。直ぐに転送を開始する場合には *USBM_TCPYStart()* 関数で該当チャンネルをスタートさせます。

④ コピーを終了するには *USBM_TCPYStart()* 関数を、該当するビットを 0 にして呼び出します。

⑤ パルスカウンタをトリガとして使用した場合には、*USBM_PCStop()* 関数で使用したチャンネルを停止します。

- タイマコピーで1回の転送に要する時間は最大 25μsec です。ただし、タイマコピーは(製品内蔵マイコンへの)割り込みを利用しソフトウェアで実装されていますので、他の割り込みなどで直ちにコピーできない場合があります。そのため、複数のチャンネルを使用する場合、また他に割り込みを利用する機能を使用している場合にはコピーのタイミングがタイマの設定値どおりに行われないことがあります。また、最悪の場合には割り込みが無視され出力が更新されない可能性もあります。

C 言語の例

```
BYTE Data[4] = { 0x01, 0x02, 0x04, 0x08 };

USBM_PortBWrite(hDev, USBM_USER_AREA, Data, 4, TRUE, FALSE); /*ユーザーメモリにコピー*/
USBM_TCPYSetCycle(hDev, 0, 99, USBM_TCLK3); /*転送サイクルを約 30Hz に設定*/

/* Data のパターンを POUT に 100 回転送 */
USBM_TCPYSetPatternCtrl(hDev, 0, USBM_USER_AREA, USBM_POUT, 100, 4, 1, 1, 0xff);
USBM_TCPYStart(hDev,1); /*チャンネル 0 をスタート*/

/*...*/

USBM_TCPYStart(hDev,0); /*コピー終了*/
```

VisualBasic6.0 の例

```
Dim Data(3) As Byte
Data(0) = &H1
Data(1) = &H2
Data(2) = &H4
Data(3) = &H8

USBM_PortBWrite hDev, USBM_USER_AREA, Data, 4, TRUE, FALSE 'ユーザーメモリにコピー
USBM_TCPYSetCycle hDev, 0, 99, USBM_TCLK3 '転送サイクルを約 30Hz に設定

'Data のパターンを POUT に 100 回転送
USBM_TCPYSetPatternCtrl hDev, 0, USBM_USER_AREA, USBM_POUT, 100, 4, 1, 1, 0xff
USBM_TCPYStart hDev,1 'チャンネル 0 をスタート

/*...*/

USBM_TCPYStart hDev,0 'コピー終了
```

□ タイムアウト設定

専用 API 関数は、そのほとんどが同期動作です。そのため、用途によってはタイムアウト時間の設定が必要になる場合があります。その場合、*USBM_SetTimeouts()* 関数を使用してください。初期状態ではデバイスへの書き込み、デバイスからの読み出し、共に約 5 秒間でタイムアウトするように設定されています。

また、*USBM_ADBRead()* 関数、*USBM_SCIRead()* 関数を呼び出すと、マイコンは指定数のデータが読み込まれるまで待ち状態となります。そのため、何らかの理由で、前記関数がタイムアウトして戻った場合に、デバイス側は後の命令を受け付けなくなります。その場合、*USBM_Abort()* 関数を呼び出し、通常のコマンドループに戻るよう指示する必要があります。

関数がタイムアウトした場合の復帰処理

① 関数の戻り値をチェックし、タイムアウトが発生したかどうかを調べます。

② タイムアウトした場合、*USBM_Abort()* 関数を呼び出し、マイコン内の読み出しループを中止します。

③ *USBM_Purge()* 関数を呼び出し、USB のリードバッファに溜まったデータ(ループを中止させる前にデバイスから送られたデータ)を破棄します。

C 言語の例

```
long nRead;
char Data[100];
TW_STATUS ret;

USBM_SetTimeouts(hDev, 2000, 1000); //リードタイムアウト 2 秒,ライトタイムアウト 1 秒

ret = USBM_SCIRead(hDev, 0, Data, 100, &nRead);
if(ret == TW_TIMEIOUT)){  //タイムアウトした場合
  USBM_Abort(hDev);
  USBM_Purge(hDev, USBM_PURGE_RX);  //リードバッファをクリア
  return;
}

/*...*/
```

VisualBasic6.0 の例

```
Dim nRead As Long
Dim Data(99) As Byte
Dim ret As Long

USBM_SetTimeouts hDev, 2000, 1000 '//リードタイムアウト 2 秒,ライトタイムアウト 1 秒

ret = USBM_SCIRead(hDev, 0, Data, 100, nRead)
If ret = TW_TIMEOUT Then 'タイムアウトした場合
  USBM_Abort hDev
  USBM_Purge hDev, USBM_PURGE_RX 'リードバッファをクリア
  Exit Sub
End If

'...
```

□ ハードウェアイベントの監視

『LANX-I16(P)』では、パルス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、Windows のメッセージ機構を通じてアプリケーションに通知することができます。

- ハードウェアイベントの監視機能は LabVIEW では使用できません。
- システムファーム Ver.4.0.1 以降が必要です。

表 42　ハードウェアイベントの監視に使用する関数

関数名	説明
USBM_SetHwEvent()	ハードウェアイベントの監視を開始/終了します。
USBM_PCSetCnt()	カウンタの値を設定します。
USBM_PCSetControl()	パルスカウンタのカウントアップ/ダウンの方法、クリア条件を設定します。
USBM_PCSetCondBit()	パルスカウンタのアップ/ダウンを決定する条件ビットを指定します。
USBM_PCSetCmp()	カウンタを自動的にクリアする場合などの設定を行います。
USBM_PCStartA()	指定チャンネルのカウントをスタートします。
USBM_PCStop()	指定チャンネルのカウントをストップします。

ハードウェアイベントの監視を開始するには、*USBM_SetHwEvent()* 関数を呼び出します。この関数には引数として *USBM_HW_EVENT* 構造体を渡します。*USBM_HW_EVENT* 構造体の C 言語でのプロトタイプと使用する定数を図 35 に示します。

```
typedef struct tagHwEvent{
  HWND hRecvWindow;   //メッセージを受け取るウィンドウのハンドル
  DWORD idRecvThread; //メッセージを受け取るスレッドの ID
  LPVOID lpRsv;       //予約
  UINT Message;       //受け取るメッセージの番号(0x8000-0xbfff)
  DWORD EventBits;    //監視するイベントをビットで指定
  long PCCnt[4];      //PC0-PC3 のカウント値と比較する閾値
  long PCCmp[4];      //PC0-PC3 の比較方法/PCCnt の自動インクリメント
  long ADVal[4];      //AD0-AD3 の入力値と比較する閾値
  short ADCmp[4];     //AD0-AD3 の比較方法/ヒステリシス
} USBM_HW_EVENT;

#define USBM_EVENT_PC0  0x00000001  //パルスカウンタ 0(PC0)を監視
#define USBM_EVENT_PC1  0x00000002  //パルスカウンタ 1(PC1)を監視
#define USBM_EVENT_PC2  0x00000004  //パルスカウンタ 2(PC2)を監視
#define USBM_EVENT_PC3  0x00000008  //パルスカウンタ 3(PC3)を監視
#define USBM_EVENT_AD0  0x00000010  //アナログ入力 0(AD0)を監視
#define USBM_EVENT_AD1  0x00000020  //アナログ入力 1(AD1)を監視
#define USBM_EVENT_AD2  0x00000040  //アナログ入力 2(AD2)を監視
#define USBM_EVENT_AD3  0x00000080  //アナログ入力 3(AD3)を監視
#define USBM_EVENT_USER 0x80000000  //ユーザー定義
#define USBM_EVENT_PC   0x0000000f  //パルスカウンタ全て
#define USBM_EVENT_AD   0x000000f0  //アナログ入力全て

#define USBM_CMP_GE     0x7fffffff  //カウンタ値が≧PCCnt[]で通知
#define USBM_CMP_LE     0x80000000  //カウンタ値が≦PCCnt[]で通知
```

図 35　USBM_HW_EVENT 構造体と使用する定数

ハードウェアイベントの通知先がウィンドウの場合、*hRecvWindow* にウィンドウのハンドルを指定します。通知先がスレッドの場合には *idRecvThread* にスレッド ID を指定します。これらは必要の無い場合 0 としてください。

EventBits には監視するハードウェアイベントをビットで指定します。例えば、AD0 のアナログ入力を監視する場合、*EventBits* には *USBM_EVENT_AD0* を指定します。複数の監視を行う場合、ビットを OR で結合します。

Message にはメッセージの番号を指定します。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。その際、メッセージのパラメータとして渡される *wParam* は *EventBits* に指定したビットのうち 1 つが"1"となり、メッセージの原因となったハードウェアイベントを示します。また、*lParam* にはパルスカウンタに関するイベントの場合はそのカウント値が、アナログ入力に関するイベントの場合は AD 変換の結果がセットされます。

- アプリケーションで自由に使用できるメッセージ番号は 0x8000(*WM_APP*)～0xbfff の範囲です。

監視を終了するには *USBM_HW_EVENT* 構造体のポインタとして *NULL* を渡すか、*EventBits* の値を 0 として *USBM_SetHwEvent()* 関数を呼び出します。

パルスカウンタ入力を監視する

パルスカウンタの入力を監視する場合、*USBM_HW_EVENT* 構造体の *PCCnt[]* と *PCCmp[]* に値を設定します。*PCCnt[0]* ～ *PCCnt[3]* には、それぞれ 0～3 チャンネルのカウンタ値と比較する閾値を設定します。*PCCmp[0]* ～ *PCCmp[3]* には閾値との比較方法、閾値の自動インクリメントを設定します。*PCCmp[]* の設定値と具体的動作を表 43 に示します。

表 43 PCCmp[]の設定値と動作の関係

PCCmp[x]の設定	ハードウェアイベントの検出条件	イベント検出後の PCCnt[x]の値
USBM_CMP_GE (0x7fffffff)	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	変化なし
USBM_CMP_GE 以外の正の値	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	PCCnt[x] = PCCnt[x] + PCCmp[x]
0	指定チャンネル(x)のカウンタ値が変化した場合	変化なし
USBM_CMP_LE 以外の負の値	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	PCCnt[x] = PCCnt[x] + PCCmp[x]
USBM_CMP_LE (0x80000000)	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	変化なし

PCCmp[] が 0 の場合、カウンタ値が変化する毎にアプリケーションに通知されます。*PCCnt[]* の値は無視されます。

PCCmp[] を正の値とすると、指定チャンネルのカウンタ値が *PCCnt[]* 以上となった場合にハードウェアイベントとして検出しアプリケーションに通知されます。負の値とすると、逆にカウンタ値が *PCCnt[]* 以下になった場合に通知されます。

また、*USBM_CMP_GE* (0x7fffffff)、*USBM_CMP_LE* (0x80000000)以外の値を指定した場合には、

ハードウェアイベントとして検出された後に、*PCCmp[]* の値が *PCCnt[]* に自動的に加算されます。これによって、一定カウント毎にハードウェアイベントとしてアプリケーションに通知を行うことができます。例えば、*PCCnt[0]* の初期値が 100、*PCCmp[0]* の値が 100 の場合、0 チャンネルのカウンタが 100 になったとき、最初のメッセージが送信されます。このとき *PCCnt[0]* は *PCCmp[0]* の値が加算され 200 となります。その後、カウンタ値が 200 になると、2 番目のメッセージが送信されます。以下、同様に 100 カウント毎にメッセージが送信されます。

PCCmp[] が *USBM_CMP_GE* または *USBM_CMP_LE* の場合、一度ハードウェアイベントが発生し、メッセージが送信されると、その条件が解除されるまで再度通知されることはありません。例えば、*PCCnt[0]* が 100 で、*PCCmp[0]* が *USBM_CMP_GE* の場合、0 チャンネルのカウンタ値が 100 以上になった場合、メッセージが送られますが、以降カウンタ値が変化してもメッセージは送信されません。この場合、*USBM_PCSetCnt()* 関数の呼び出しなどで、カウンタ値が 100 未満になると、再びメッセージ送信可能な状態になります。

USBM_SetHwEvent() を呼び出しただけでは、パルスカウンタのカウント動作は開始されませんので、別途制御関数を呼び出す必要があります。

① *USBM_SetHwEvent()* 関数を使用し、ハードウェアイベントの監視を開始します。

② 必要であれば、*USBM_PCSetControl()* 関数などを使用し、パルスカウンタのカウント条件を設定します。カウント条件等の詳細は「パルスカウンタ」(54 ページ)を参照してください。

③ *USBM_PCStartA()* 関数を使用し、パルスカウンタの計数を開始します。

- 自動インクリメントでは *PCCnt[]* 値のオーバーフローは考慮されません。例えば *PCCnt[]* の値が正の最大値を超えた場合、負の値となってしまいますのでご注意ください。

アナログ入力を監視する

アナログ入力を監視する場合、前記の *USBM_HW_EVENT* 構造体の *ADVal[]* と *ADCmp[]* に値を設定します。*ADVal[0]* ～ *ADVal[3]* には、それぞれ 0～3 チャンネルのアナログ入力値と比較する閾値を設定します。*ADCmp[0]* ～ *ADCmp[3]* には閾値との比較方法とヒステリシスを設定します。

ADVal[] は 32 ビットの変数ですが、格納する値は図 26 (44 ページ)と同様の 16 ビット値です(上位 16 ビットは常に 0 としてください)。設定したい閾値電圧を V_{TH} とすると、*ADVal[]* への設定値 C_{th} は、以下の式で求めることができます。

$$C_{TH} \fallingdotseq (V_{TH} / VREF) \times 65536$$

ADCmp[] は 0 以上の場合、指定チャンネルの AD 変換結果が *ADVal[]* 以上となる場合に、ハードウェアイベントとして検出しアプリケーションに通知されます。*ADCmp[]* が負の場合は、AD 変換結果が *ADVal[]* 以下となる場合に通知されます。

また、*ADCmp[]* の大きさはヒステリシスの大きさを表します。*ADCmp[]* が正の場合、対応するアナ

ログ入力電圧が V_{TH}[V] 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、同じイベントが何度も通知されるのを避けるために、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} [V] 以下となったときに解除されます(図 36)。このときの V_{HYST} を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。

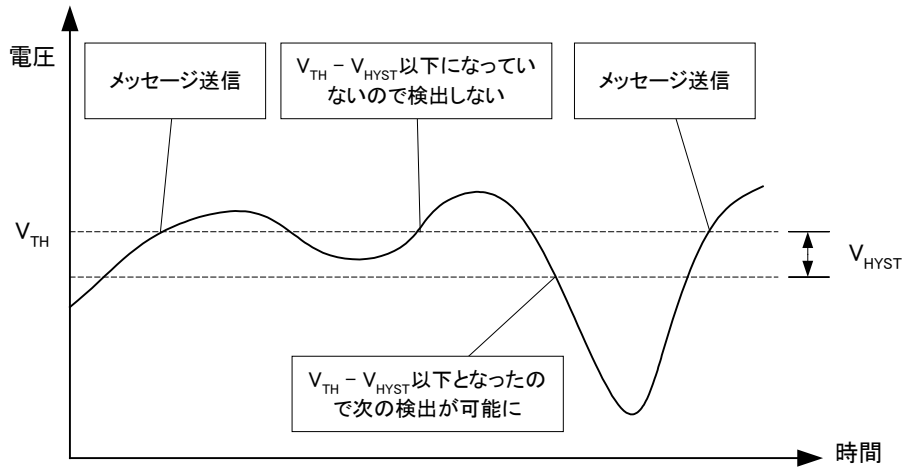


図 36　ヒステリシスが設定されている場合の動作

ADCmp[] の設定値 C_{CMP} も V_{HYST} 電圧から以下の式で求めることができます。

$$C_{CMP} \fallingdotseq (V_{HYST} / VREF) \times 65536$$

例えば、AD0 の V_{TH} を VREF/2 [V]、V_{HYST} を約 VREF/100 [V] とし、入力電圧 ≧ V_{TH} のときにメッセージを受け取るには、以下の計算から、*ADVal[0]*= 32768、*ADCmp[0]*= 655 とします。

$$C_{TH} = (VREF / 2) / VREF \times 65536 = 32768$$
$$C_{CMP} = (VREF / 100) / VREF \times 65536 = 655.36 \fallingdotseq 655$$

逆に入力電圧 ≦ V_{TH} のときにメッセージを受け取るには、*ADCmp[0]*= −655 とします。

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

8. トラブルシューティング

□ デバイスに接続できない場合

① ご利用のパソコンで、インターネット通信などを監視するセキュリティソフトを使用されている場合は、セキュリティソフトを一時的に停止し、再度接続してみてください。セキュリティソフトの設定によっては、製品との通信がブロックされる場合があります。

② 「Windows ファイアウォール」が有効になっている場合、一時的に無効に設定し、再度接続してみてください。「Windows ファイアウォール」によって通信がブロックされてしまう場合は、「コントロールパネル→セキュリティセンター→Windows ファイアウォール」を開き、「例外」にアプリケーションを登録することで通信可能になります。

③ 製品に固定の IP アドレスを割り当ててみてください(16 ページ参照)。

④ パソコンの LAN アダプタが複数ある場合は、製品に固定 IP を設定した後、製品と繋がっているアダプタ以外を無効にしてみてください(接続名を右クリックし、「無効にする」を選びます)。

⑤ ネットワーク内に、製品、および、ホストパソコンと同一の IP アドレスを使用しているノードが無いこと確認してください。また、制御に使用するポート番号(TCP、および、UDP の 49152 番)を利用しているアプリケーションが無いことを確認してください。

Appendix

□ マルチスレッドプログラムからの呼び出しについて

ライブラリでは複数のスレッドからの関数呼び出しをサポートしていますが、デバイスとの通信仕様により、1 つのデバイスを複数のスレッドから同時に制御することはできません。何らかの理由により、複数のスレッドから 1 つのデバイスにアクセスする必要がある場合には、クリティカルセクションなどを使用することにより、ライブラリ関数の呼び出しをシリアル化し、複数のスレッドが同時に 1 つのデバイスにアクセスしないようにプログラムしてください。

1 つのスレッドが 1 つのデバイスのみを制御する場合は、複数のスレッドから同時にライブラリ関数を呼び出しても問題ありません。

□ TWX ライブラリ関数リファレンス

各関数の説明は、C 言語、VisualBasic、VisualBasic.NET それぞれにおけるプロトタイプ、変数の説明、動作説明の順になっています。

ほとんどの関数の戻り値は 32 ビットの整数で関数の実行結果を表します（以下参照）。関数がそれ以外の特別な戻り値を返す場合は、各関数の動作説明の欄で内容を示します。

以下に主な戻り値の意味を示します。尚、戻り値を示す各定数は各言語用の定義ファイル（拡張子が「.h」、「.bas」、「.vb」のファイル）中で定義されています。

表 44　関数の戻り値

定数	値	意味
TW_OK	0x00000000	正常終了
TW_INVALID_HANDLE	0x00000001	デバイスのハンドルが無効
TW_DEVICE_NOT_FOUND	0x00000002	デバイスが見つからない
TW_IO_ERROR	0x00000004	送受信中にエラーが発生した
TW_INSUFFICIENT_RESOURCES	0x00000005	リソースエラー(デバイスの最大接続数を超えた場合など)
TW_INVALID_ARGS	0x00000010	関数に渡された引数が無効
TW_NOT_SUPPORTED	0x00000011	サポートされない機能
TW_OTHER_ERROR	0x00000012	その他のエラー
TW_TIMEOUT	0xffff0001	送信または受信処理がタイムアウトした
TW_FILE_ERROR	0xffff0002	ファイル操作に関するエラーが発生した
TW_MEMORY_ERROR	0xffff0003	メモリの確保に失敗した
TW_DATA_NOT_FOUND	0xffff0004	有効なデータが見つからなかった
TW_SOCKET_ERROR	0xffff0005	Winsock のエラー（多くの場合 WSAGetLastError() を呼び出すとさらに詳しい情報を得ることができます）
TW_ACCESS_DENIED	0xffff0006	デバイスとの認証作業に失敗した
TW_NOT_SUPPORTED_MODE	0xffff0007	関数をサポートしないモードでデバイスに接続している
TW_FLASH_MODE_DEVICE	0xffff0008	フラッシュ書換えモードのデバイスのため制御できない

TWX_Open()

```
TW_STATUS TWX_Open(TW_HANDLE *phDev, long Number, long ProductsType)
Function TWX_Open(ByRef phDev As Long, ByVal Number As Long, ByVal ProductsType As Long) As Long
Function TWX_Open(ByRef phDev As System.IntPtr,
                  ByVal Number As Integer, ByVal ProductsType As Integer) As Integer
```

phDev : 取得したハンドルの格納先
Number : 接続する製品の番号
ProductsType : 接続する製品のタイプを以下の値で指定します
TWX_USBX_I16 : USBX-I16 または USBX-I16P に接続する
TWX_LANX_I16 : LANX-I16 または LANX-I16P に接続する
TWX_ANY_DEVICE : ライブラリが対応する製品であれば接続する

デバイスに接続します。成功すると phDev にデバイスへのハンドルが格納されます。

TWX_Close()

```
TW_STATUS TWX_Close(TW_HANDLE hDev)
Function TWX_Close(ByVal hDev As Long) As Long
Function TWX_Close(ByVal hDev As System.IntPtr) As Integer
```

hDev : デバイスのハンドル

ハンドルをクローズし、デバイスへのアクセスを終了します。

TWX_CloseAll()

```
TW_STATUS TWX_Close()
Function TWX_Close() As Long
Function TWX_Close() As Integer
```

プロセスが接続中の全てのハンドルをクローズします。

TWX_OpenByAddress()

```
TW_STATUS TWX_OpenByAddress(TW_HANDLE *phDev, LPCTSTR Address, long ProductType)
Function TWX_OpenByAddress(ByRef phDev As Long,
                           ByVal Address As String, ByVal ProductType As Long) As Long
Function TWX_OpenByAddress(ByRef phDev As System.IntPtr,
                           ByVal Address As String, ByVal ProductType As Integer) As Integer
```

phDev : 取得したハンドルの格納先
Address : IP アドレスまたはドメイン名
ProductType : 接続する製品のタイプ。TWX_LANX_I16 としてください。

IP アドレスやドメイン名を指定してデバイスと接続します。
Address には IP アドレスまたはドメイン名の後に":"(コロン)とポート番号を続けることで、ポート番号を指定することができます(例:"lanx01.mydomain.co.jp:49153")。

TWX_Listen()

```
TW_STATUS TWX_Listen(UINT_PTR *pListenSocket, LPCTSTR pLocalIP, DWORD PortNumber)
Function TWX_Listen(ByRef pListenSocket As Long, ByVal pLocalIP As String,
                    ByVal PortNumber As Long ) As Long
Function TWX_Listen(ByRef pListenSocket As System.IntPtr, ByVal pLocalIP As String,
                    ByVal PortNumber As Integer) As Integer
```

pListenSocket : 接続待ちソケットの格納先
pLocalIP : 接続待ちを行うローカル(パソコン側)の IP アドレス
PortNumber : 接続待ちを行うポート番号

指定のポート番号をオープンし、クライアントモードのデバイスの接続を待ちます。
pListenSocket に返された値を TWX_Accept() に渡すことで、ここで設定したポートへの接続を受け入れることができます。
pLocalIP はパソコンのネットワークカードが複数ある場合に、接続待ちを行う IP アドレスを指定します。
※この関数は Ver.5.0.1 以降のファームウェアが必要です。

TWX_Accept()

```
TW_STATUS TWX_Accept(UINT_PTR ListenSocket, TW_HANDLE *phDev)
Function TWX_Accept(ByVal ListenSocket As Long, ByRef phDev As Long) As Long
Function TWX_Accept (ByVal ListenSocket As System.IntPtr, ByRef phDev As System.IntPtr) As Integer
```

ListenSocket : 接続待ちソケット
phDev : デバイスへのハンドルの格納先

TWX_Listen()でオープンした接続待ちソケットに対して、接続要求があれば受け入れてデバイスへのハンドルを返します。
接続要求が無い場合は TW_DEVICE_NOT_FOUND を返します。接続待ちを行う間はこの関数を繰り返し呼び出して、接続要求の有無を確認してください。
※この関数は Ver.5.0.1 以降のファームウェアが必要です。

TWX_CloseListenSocket()

```
TW_STATUS TWX_CloseListenSocket(UINT_PTR ListenSocket)
Function TWX_CloseListenSocket(ByVal ListenSocket As Long) As Long
Function TWX_CloseListenSocket(ByVal ListenSocket As System.IntPtr) As Integer
```

ListenSocket : 接続待ちソケット

TWX_Listen()でオープンした接続待ちのソケットをクローズします。
※この関数は Ver.5.0.1 以降のファームウェアが必要です。

TWX_InitializeA()

TW_STATUS TWX_Initialize(TW_HANDLE hDev, DWORD InitOption)

Function TWX_InitializeA(ByVal hDev As Long, ByVal InitOption As Long) As Long

Function TWX_InitializeA(ByVal hDev As System.IntPtr, ByVal InitOption As Integer) As Integer

hDev : デバイスのハンドル
InitOption : 初期化する機能を指定。以下を OR で結合します。

TWX_INIT_PORT_DATA(0x0002)	ポートのデータを初期化
TWX_INIT_BUS(0x0004)	外部バス設定を初期化
TWX_INIT_DMA(0x0008)	DMA を初期化
TWX_INIT_TIMER(0x0010)	16 ビットタイマを初期化
TWX_INIT_AD(0x0020)	AD コンバータを初期化
TWX_INIT_SCI(0x0040)	シリアルポートを初期化
TWX_INIT_PC(0x0080)	パルスカウンタを初期化
TWX_INIT_TCPY(0x0100)	タイマコピーを初期化
TWX_INIT_ALL(0xffffffff)	全ての機能を初期化

デバイスの初期化を行います。InitOption で初期化する機能を指定できます。 ただし、アナログ入出力の設定用レジスタとパルスカウンタの設定用レジスタは影響を受けません。未対応の製品の場合 TW_NOT_SUPPORTED が返ります。
※この関数は Ver.3.2.1 以降のファームウェアが必要です。

TWX_Initialize()

TW_STATUS TWX_Initialize(TW_HANDLE hDev)

Function TWX_Initialize(ByVal hDev As Long) As Long

Function TWX_Initialize(ByVal hDev As System.IntPtr) As Integer

hDev : デバイスのハンドル

デバイスの初期化を行います。未対応の製品の場合 TW_NOT_SUPPORTED が返ります。

TWX_GetNumber()

long TWX_GetNumber(TW_HANDLE hDev)

Function TWX_GetNumber(ByVal hDev As Long) As Long

Function TWX_GetNumber(ByVal hDev As System.IntPtr) As Integer

hDev : デバイスのハンドル

戻り値として接続中のデバイスの装置番号を返します。ハンドルが無効な場合には 0 を返します。

TWX_PortWrite()

```
TW_STATUS TWX_PortWrite(TW_HANDLE hDev, DWORD Port, BYTE Data, BYTE Mask)

Function TWX_PortWrite(ByVal hDev As Long, ByVal Port As Long,
                       ByVal Data As Byte, ByVal Mask As Byte) As Long

Function TWX_PortWrite(ByVal hDev As System.IntPtr, ByVal Port As Integer,
                       ByVal Data As Byte, ByVal Mask As Byte) As Integer
```

hDev : デバイスのハンドル
Port : 出力するポート
TWX_P4 : P40-P47 を操作
TWX_PA : PA0-PA7 を操作
TWX_POUT : POUT0-POUT7 を操作
Data : 出力データ
Mask : 操作するビットを指定するマスク

出力ポートの出力値を変更します。特定のビットだけを操作する場合には Mask に値を設定してください。
例えば P47 だけを操作する場合、Mask = 0x80 とします。
P4、PA ポートは"0"を書いたビットと対応する端子が"ON"となり、"1"のビットと対応する端子が"OFF"となります。POUT ポートは逆に"1"のビットと対応する端子が"ON"となりますのでご注意ください。

TWX_PortRead()

```
TW_STATUS TWX_PortRead(TW_HANDLE hDev, DWORD Port, BYTE *pData)

Function TWX_PortRead(ByVal hDev As Long, ByVal Port As Long, ByRef pData As Byte) As Long

Function TWX_PortRead(ByVal hDev As System.IntPtr,
                      ByVal Port As Integer, ByRef pData As Byte) As Integer
```

hDev : デバイスのハンドル
Port : 入力するポート
TWX_P1 : P10-P17 から入力
TWX_P2 : P20-P27 から入力
pData : 入力データの格納先

入力ポートの入力値を読みます。"ON"の端子と対応するビットが"1"になります。

TWX_AnalogWriteReg()

```
TW_STATUS TWX_AnalogWriteReg(TW_HANDLE hDev, BYTE Reg)

Function TWX_AnalogWriteReg(ByVal hDev As Long, ByVal Reg As Byte) As Long

Function TWX_AnalogWriteReg(ByVal hDev As System.IntPtr, ByVal Reg As Byte) As Integer
```

hDev : デバイスのハンドル
pReg : レジスタ値。ビットの意味は以下。
ビット0 : 0のときDA0がユニポーラ出力。1のときバイポーラ出力
ビット1 : 0のときDA1がユニポーラ出力。1のときバイポーラ出力
ビット2 : 0のときAD0がユニポーラ入力。1のときバイポーラ入力
ビット3 : 0のときAD1がユニポーラ入力。1のときバイポーラ入力
ビット4 : 0のときADTRIGの立ち下がり､1のとき立ち上がりがトリガ入力
ビット5-7 : 予約。無視されます。

アナログ入出力の設定用レジスタに値を書き込みます。このレジスタでアナログチャンネルのユニポーラ／バイポーラ設定、ADTRIG 信号の極性設定を行います。ユニポーラ設定では 0～5V、バイポーラ設定では-2.5～2.5Vの範囲の入出力となります。AD2,AD3は設定に無関係にユニポーラ入力です。

TWX_AnalogReadReg()

```
TW_STATUS TWX_AnalogReadReg(TW_HANDLE hDev, BYTE *pReg)

Function TWX_AnalogReadReg(ByVal hDev As Long, ByRef pReg As Byte) As Long

Function TWX_AnalogReadReg(ByVal hDev As System.IntPtr, ByRef pReg As Byte) As Integer
```

hDev : デバイスのハンドル
pReg : レジスタ値の格納先

アナログ入出力の設定用レジスタの値を読み出します。レジスタ値の意味は TWX_AnalogWriteReg()の説明を参照してください。

TWX_ADRead()

```
TW_STATUS TWX_ADRead(TW_HANDLE hDev, long CH, long *pData)

Function TWX_ADRead(ByVal hDev As Long, ByVal CH As Long, ByRef pData As Long) As Long

Function TWX_ADRead(ByVal hDev As System.IntPtr,
                    ByVal CH As Integer, ByRef pData As Integer) As Integer
```

hDev : デバイスのハンドル
CH : 入力するチャンネル(0,1,2,3)
pData : 入力データの格納先

指定チャンネルをAD変換した結果を読み出します。変換結果は0～1023までの値となります。

TWX_DAWrite()

```
TW_STATUS TWX_DAWrite(TW_HANDLE hDev, long CH, BYTE Data)

Function TWX_DAWrite(ByVal hDev As Long, ByVal CH As Long, ByRef Data As Byte) As Long

Function TWX_DAWrite(ByVal hDev As System.IntPtr, ByVal CH As Integer, ByVal Data As Byte) As Integer
```

hDev : デバイスのハンドル
CH : 出力するチャンネル(0,1)
Data : 出力するデータ

指定チャンネルをDAコンバータの出力値を変更します。

TWX_PCWriteReg()

```
TW_STATUS TWX_PCWriteReg(TW_HANDLE hDev, BYTE Reg)
Function TWX_PCWriteReg(ByVal hDev As Long, ByVal Reg As Byte) As Long
Function TWX_PCWriteReg(ByVal hDev As System.IntPtr, ByVal Reg As Byte) As Integer
```

hDev : デバイスのハンドル
Reg : レジスタ値。ビットの意味は以下。
ビット 0 : PC0 は立ち上がり("OFF"→"ON")をカウントします。
ビット 1 : PC0 は立ち下がり("ON"→"OFF")をカウントします。
ビット 2 : PC1 は立ち上がり("OFF"→"ON")をカウントします。
ビット 3 : PC1 は立ち下がり("ON"→"OFF")をカウントします。
ビット 4 : PC2 は立ち上がり("OFF"→"ON")をカウントします。
ビット 5 : PC2 は立ち下がり("ON"→"OFF")をカウントします。
ビット 6 : PC3 は立ち上がり("OFF"→"ON")をカウントします。
ビット 7 : PC3 は立ち下がり("ON"→"OFF")をカウントします。

パルスカウンタの設定用レジスタに値を書き込みます。このレジスタはパルスカウンタのカウントエッジを指定します。初期設定では全て 0 になっています。

TWX_PCReadReg()

```
TW_STATUS TWX_PCReadReg(TW_HANDLE hDev, BYTE *pReg)
Function TWX_PCReadReg(ByVal hDev As Long, ByRef pReg As Byte) As Long
Function TWX_PCReadReg(ByVal hDev As System.IntPtr, ByRef pReg As Byte) As Integer
```

hDev : デバイスのハンドル
pReg : レジスタ値の格納先

パルスカウンタの設定用レジスタの値を読み出します。レジスタ値の意味は TWX_PCWriteReg()の説明を参照してください。

TWX_PCSetMode()

```
TW_STATUS TWX_PCSetMode(TW_HANDLE hDev, long Mode, long CHBits)
Function TWX_PCSetMode(ByVal hDev As Long, ByVal Mode As Long, ByVal CHBits As Long) As Long
Function TWX_PCSetMode(ByVal hDev As System.IntPtr,
                       ByVal Mode As Integer, ByVal CHBits As Integer) As Integer
```

hDev : デバイスのハンドル
Mode : カウントモード
TWX_PC_DEFAULT : 初期状態に戻します。各チャンネルは独立です
TWX_PC_2PHASE : 2 相のエンコーダ信号をカウントします
TWX_PC_3PHASE : 2 相のエンコーダ信号をカウントし、Z 信号でクリア
CHBits : 使用チャンネル(Mode = TWX_PC_2PHASE のときのみ有効)
TWX_PC0_PC1 : PC0 と PC1 を使用してカウントします。
TWX_PC2_PC3 : PC2 と PC3 を使用してカウントします。

パルスカウンタを 2 相のエンコーダ信号(A 相、B 相)をカウントできるように設定します。CHBits で指定したそれぞれの端子に A 相、B 相信号を入力してください。
Mode に TWX_PC_3PHASE を指定すると、PC0 がカウンタクリア、PC2 と PC3 で 2 相カウントする設定になります。PC2、PC3 に A 相、B 相信号を入力し、PC0 に Z 相信号を入力すると 1 毎に PC2 と PC3 のカウンタがクリアされ、PC0 のカウンタがカウントアップします。

TWX_PCStart()

```
TW_STATUS TWX_PCStart(TW_HANDLE hDev, BYTE CHBits)
Function TWX_PCStart(ByVal hDev As Long, ByVal CHBits As Byte) As Long
Function TWX_PCStart(ByVal hDev As System.IntPtr, ByVal CHBits As Byte) As Integer
```

hDev　: デバイスのハンドル
CHBits : パルスカウンタのチャンネル
　　TWX_PC0_PC1 : PC0 と PC1 をスタート
　　TWX_PC2_PC3 : PC2 と PC3 をスタート
　　TWX_PC0　　: PC0 をスタート
　　TWX_PC1　　: PC1 をスタート
　　TWX_PC2　　: PC2 をスタート
　　TWX_PC3　　: PC3 をスタート

指定チャンネルのパルスカウンタの計数をスタートさせます。CHBits に指定する定数は OR で結合することができます。

TWX_PCStop()

```
TW_STATUS TWX_PCStop(TW_HANDLE hDev, BYTE CHBits)
Function TWX_PCStop(ByVal hDev As Long, ByVal CHBits As Byte) As Long
Function TWX_PCStop(ByVal hDev As System.IntPtr, ByVal CHBits As Byte) As Integer
```

hDev　: デバイスのハンドル
CHBits : パルスカウンタのチャンネル
　　TWX_PC0_PC1 : PC0 と PC1 をストップ
　　TWX_PC2_PC3 : PC2 と PC3 をストップ
　　TWX_PC0　　: PC0 をストップ
　　TWX_PC1　　: PC1 をストップ
　　TWX_PC2　　: PC2 をストップ
　　TWX_PC3　　: PC3 をストップ

指定チャンネルのパルスカウンタの計数をストップします。CHBits に指定する定数は OR で結合することができます。

TWX_PCReadCnt()

```
TW_STATUS TWX_PCReadCnt(TW_HANDLE hDev, long CHBits, long *pCnt)
Function TWX_PCReadCnt(ByVal hDev As Long, ByVal CHBits As Long, ByRef pCnt As Long) As Long
Function TWX_PCReadCnt(ByVal hDev As System.IntPtr,
                       ByVal CHBits As Integer, ByRef pCnt As Integer) As Integer
```

hDev　: デバイスのハンドル
CHBits : パルスカウンタのチャンネル
　　TWX_PC0_PC1 : PC0 と PC1 で計数した値
　　TWX_PC2_PC3 : PC2 と PC3 で計数した値
　　TWX_PC0　　: PC0 のカウンタ値
　　TWX_PC1　　: PC1 のカウンタ値
　　TWX_PC2　　: PC2 のカウンタ値
　　TWX_PC3　　: PC3 のカウンタ値
pCnt　: カウント値の格納先

指定チャンネルのパルスカウンタの値を読み出します。チャンネルは番号ではなく定数で指定しますので、ご注意してください。

TWX_PCSetCnt()

```
TW_STATUS TWX_PCSetCnt(TW_HANDLE hDev, long CHBits, long *pCnt)
Function TWX_PCSetCnt(ByVal hDev As Long, ByVal CHBits As Long, ByVal Cnt As Long) As Long
Function TWX_PCSetCnt(ByVal hDev As System.IntPtr,
                      ByVal CHBits As Integer, ByVal Cnt As Integer) As Integer
```

hDev　: デバイスのハンドル
CHBits : パルスカウンタのチャンネル
　　TWX_PC0_PC1 : PC0 と PC1 で計数した値
　　TWX_PC2_PC3 : PC2 と PC3 で計数した値
　　TWX_PC0　　: PC0 のカウンタ値
　　TWX_PC1　　: PC1 のカウンタ値
　　TWX_PC2　　: PC2 のカウンタ値
　　TWX_PC3　　: PC3 のカウンタ値
Cnt　: セットする値

指定チャンネルのパルスカウンタに値をセットします。クリアする場合は Cnt を 0 として呼び出します。チャンネルは番号ではなく定数で指定しますのでご注意ください。

□ 命令実行までのレイテンシ

ネットワークを使用したデジタル/アナログ入出力製品で、しばしば問題となるのは命令実行までのレイテンシ(遅延時間)です。図 37 は *USBM_PortRead8()* 関数を 1000 回呼び出したときの実行時間の分布です。

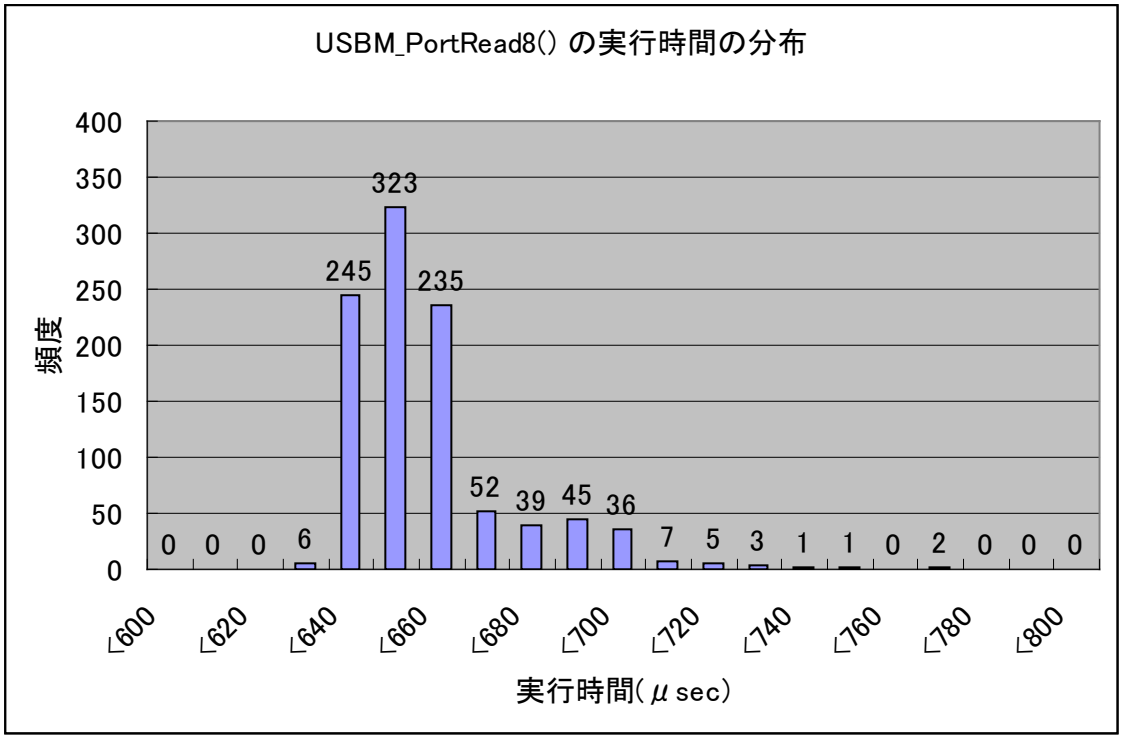


図 37　*USBM_PortRead8()* 1000 回の実行時間の分布

図 37 はホストパソコンとデバイスを直接接続して測定した参考データです。実際には通過するネットワークハブやルーターの数、経路、ホストパソコンの処理能力など様々な要因の影響を受けるため、命令を発行してから実際に実行されるまでの時間を一定に保つのは困難です。

製品ではハードウェアからの入力に対する簡単なフィードバックを、ホストパソコンを介さず、デバイス上で処理できるようにパルスカウンタにコンペアアウトという機能を設けています(詳しくは「パルスカウンタ」の節を参照してください)。また、DA コンバータやタイマコピーなどのように搭載マイコンに用意したデータテーブルを利用して出力を一定周期で変化させる機能も用意されています。このような機能を上手く利用することで、リアルタイム性を確保し応用分野を広げることが可能です。

□ ネットワーク用語集

AUTO-MDIX（Automatic medium-dependent interface crossover）

通常、ネットワーク機器同士を接続する場合は、接続する機器の種類に応じてストレートケーブルとクロスケーブルを使いわける必要があります。AUTO-MDIX に対応した機器では、相手機器との接続状態を自動判別して通信を行いますので、ケーブル種別を意識すること無く接続することができます。

DHCP（Dynamic Host Configuration Protocol）

一時的にネットワークに接続する機器に対して、使用可能な IP アドレスを割り当て、通信に必要な情報を提供するためのプロトコルです。割り当てを行う側の機器やプログラムを DHCP サーバー、割り当てを受ける側の機器やプログラムを DHCP クライアントと呼びます。一般的なブロードバンドルーターには DHCP サーバーとしての機能が備わっています。

DNS（Domain Name System）

ドメイン名と IP アドレスの対応を管理するために使用されるシステムです。DNS サーバーに問い合わせを行うことで、ドメイン名から IP アドレスを検索し、通信が可能になります。一般的なブロードバンドルーターには DNS サーバーとしての機能が備わっています。

一般に DDNS(Dynamic DNS)と呼ばれるサービスを利用することで、DNS データベースを適宜更新することが可能になり、プロバイダから一時的に割り当てられた IP アドレスでもサーバーを公開することができます。

MAC アドレス（Media Access Control address）

ネットワーク機器を識別するために、1 つ1つの機器に割り当てられた個別の番号です。

NTP（Network Time Protocol）

ネットワーク機器の時計を正しい時刻に同期するためのプロトコルです。通常は SNTP(Simple Network Time Protocol)という簡易版が使用されます。

ゲートウェイアドレス

異なるネットワーク上の機器と通信する場合に、窓口の役割を果たす機器のアドレスです。

サブネットマスク

IP アドレスとのアンド（論理積）をとることでネットワークアドレスを計算できるマスク値です。

ネットワークアドレスは管理上の理由などで分割されたネットワークそれぞれを識別するための番号で、ネットワークアドレスが違う機器同士は直接通信することができません。そのため、異なるネットワークへのデータを届ける場合には予め設定されたゲートウェイアドレスに対してデータを送信します。

ドメイン名

「www.techw.co.jp」のような形式で表記されるホスト名です。ドメイン名で与えられたホストと通信を行うためには、まず DNS などの仕組みを使って、そのホストの IP アドレスを調べることが必要になります。

ブロードキャスト

送信相手を特定せずにパケットを送信することです。同一ネットワークの全ての機器が受信可能です。

ポート番号

ネットワーク上のサービスやアプリケーションを識別するのに使用される番号です。TCPプロトコルとUDPプロトコルそれぞれで番号が管理されています。1～65535 までの番号が使用可能ですが、1～49151 までは FTP や HTTP といった特定のプロトコルやアプリケーションに使用されることになっています。『LANX-I16/LANX-I16P』では TCP、UDP 両方のプロトコルを使用しますが、どちらも同じポート番号（デフォルトでは 49152）を使用します。

保証期間

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

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

サポート情報

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

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

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

(2) 本書の内容の一部または全部を無断転載することをお断りします。

(3) 本書の内容については、将来予告なしに変更することがあります。

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

改訂記録

年月	版	改訂内容
2007 年 5 月	初	
2007 年 6 月	2	・「製品の制御に必要なファイル」の表を修正
2007 年 6 月	3	・「製品の制御に必要なファイル」の表を修正
2008 年 11 月	4	・図 15 のタイトルを修正 ・設定ツールのバージョンアップに対応 ・トラブルシューティングを追加 ・関数リファレンスに TWX_InitializeA()を追加
2009 年 5 月	5	・ハードウェアイベント監視の説明を追加
2009 年 5 月	6	・アナログ入力端子の説明に「電源オフ時に入力電圧が加わる場合」を追加
2009 年 11 月	7	・64bit 版に対応した記述に変更
2011 年 9 月	8	・ファームウェア 5.0.1 に対応 ・対応 OS を修正 ・誤記の修正
2012 年 2 月	9	・追加ファームに関する記述を追加 ・設定ツールのバージョンアップに対応 ・誤記の修正 ・その他
2013 年 3 月	10	・「安全にご使用いただくために」に注意事項を追加 ・対応 OS に Windows 8 を追加
2014 年 11 月	11	・マルチスレッドプログラミングに関する記述を追加
2018 年 4 月	12	・対応 OS に Windows 10 を追加

管理番号:TW-RE-MB23-12