Linux 版 USBM ライブラリ
関数リファレンス

テクノウェーブ株式会社

目次

1. はじめに

□ 『Linux 版 USBM ライブラリ』について

『Linux 版 USBM ライブラリ』は、弊社製品 M3069 マイコンボードシリーズの制御用ライブラリです。現在、表 1 の製品でご利用になれます。本リファレンスは『USBM ライブラリ』の個々の関数について使用方法を解説しています。

表 1　対応するライブラリのバージョン

対応製品	『USBM3069』/『USBM3069F』/『USBM3069-S』/『USBM3069-SL』

『Linux 版 USBM ライブラリ』は製品付属 CD の「¥DLL¥usbm3069.tar.gz」に含まれます。また、製品のサポートページからも最新版をダウンロードしていただけます。ファイルをお使いのシステムで解凍してご利用ください。ライブラリのインストール方法、サンプルプログラムなどについては解凍フォルダの「readme.txt」でご案内していますので、本マニュアルと合わせてご参照ください。

□ 本リファレンスの対応ファイルとバージョンについて

本リファレンスは下記のバージョンのファイル内容を元に記載されています。過去のバージョンについては記載内容と異なる場合がございますのでご注意ください。

表 2　対応するライブラリのバージョン

ファイル名	バージョン番号	soname	対応 OS
libusbm3069.so.1.3.x [1]	1.3.x	libusbm3069.so.1	Linux OS (x86 のみ)

□ 本リファレンス内の表記について

本リファレンス内では対応製品を「デバイス」と表記します。また、ハードウェアの電気的状態について記述する必要がある場合には、下記のように表記します。

表 3　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。
“Z”	端子がハイインピーダンスの状態。

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。同様に「B’」に続く数値は 2 進数であることを表します。例えば“B’01000001”のように表記されます。数値の最初に特別な表記が無い場合は 10 進数です。

[1] x にはより細かなバージョンを示す数値が入ります。

2. 関数リファレンス

各関数の説明は、C 言語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。また、関数で構造体を使用する場合には、その節の先頭で説明を加えています。

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。

□ 関数の戻り値の意味

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」のファイル)中で定義されています。

表 4　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	
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

□ デバイスへの接続／切断に関する関数

USBM_Open()

```
TW_HANDLE USBM_Open(TW_CSTR pSerial)
```

pSerial: [入力]USB のシリアル番号

pSerial が NULL か""の場合、接続できた最初のデバイスのハンドルを返します。それ以外では指定された ID と一致するデバイスと接続します。指定デバイスと接続できなかった場合は 0 を返します。

USBM_OpenByPI()

```
TW_STATUS USBM_OpenByPI(TW_HANDLE *phDev, TW_CSTR pUuid, DWORD Number, long Opt)
```

phDev : [出力]ハンドルの格納先
pUuid : [入力]UUID を指定します。指定が無い場合には NULL または""とします。
Number : Number を指定します。指定が無い場合には 0 とします。
Opt   : インタフェースを指定
　　USBM_IF_USB(0x80000000) : USB デバイスに接続する
　　USBM_IF_ANY(0xff000000) : インタフェースを問わない

PI エリア内の UUID と Number が指定と一致するデバイスと接続します（PI エリアについての詳細は製品マニュアルを参照してください）。
Opt は USBM_IF_USB でインタフェースを指定します。

USBM_Close()

```
TW_STATUS USBM_Close(TW_HANDLE hDev)
```

hDev : デバイスのハンドル

デバイスとの接続を切ります。制御を終了する場合に呼び出します。

USBM_ListDevicesA()

```
TW_STATUS USBM_ListDevicesA(long *pnDev, TW_STR pIID, long *pnID, long Opt)
```

pnDev : [出力]デバイス数の格納先
pIID   : [出力]USB のシリアル番号または IP アドレスの格納先
pnID: [入力]pIID に用意された文字数
　　[出力]pIID に必要な文字数、または実際にコピーした文字数
Opt   : 次のオプション
　　USBM_IF_USB(0x80000000) : USB デバイスをリストアップする

接続されているデバイス数を調べ、USB のシリアル番号の情報を取得します。
pIID には USB デバイスのシリアル番号が格納されます。シリアル番号は文字列で格納されます。複数のデバイスが見つかった場合には、それぞれのデバイスの ID 文字列間に' '(スペース)が挿入され、最後の要素の後には'¥0'が挿入されます。

pnID は入力として pIID に用意された文字数を渡します。出力としては全ての ID 文字列が格納された場合にはその文字数が、格納しきれなかった場合には格納するのに十分な文字数を示します。

Opt は USBM_IF_USB で USB インタフェースを指定します。

USBM_ListDevices()

long USBM_ListDevices(TW_CHAR (*pSerial)[9],long nSerial)

(*pSerial)[9] : [出力]9 文字の文字列へのポインタ、シリアル番号を格納します
nSerial : シリアル番号を格納する配列の数

接続されている USB デバイスの数を返します。pSerial が NULL でなければ nSerial 個までシリアル番号を pSerial に格納します。シリアル番号は製品に予め付けられている英数字 8 文字(ANSI)の NULL 終端文字列で一意な値です。

USBM_ListDevicesByID()

long USBM_ListDevicesByID(WORD VID, WORD PID, TW_CHAR (*pSerial)[9], long nSerial)

VID : ベンダーID
PID : プロダクト ID
(*pSerial)[9] : [出力]9 バイトの文字列へのポインタ、シリアル番号を格納します
nSerial : 上の配列の数

指定されたベンダーID とプロダクト ID に該当するデバイスを検索し、接続されているデバイスの数を返します。pSerial が NULL でなければ nSerial 個までシリアル番号を pSerial に格納します。シリアル番号は製品に予め付けられている英数字 8 文字(ANSI)の文字列で一意な値です。

USBM_OpenByID()

TW_HANDLE USBM_OpenByID(WORD VID, WORD PID, TW_CSTR pSerial)

VID : ベンダーID
PID : プロダクト ID
pSerial : [入力]シリアル番号、NULL で終わる文字列

指定されたベンダーID とプロダクト ID に該当するデバイスを検索しオープンします。pSerial が NULL の場合、または""が渡された場合、オープンできた最初のデバイスのハンドルを返します。それ以外ではシリアル番号が一致したデバイスをオープンします。指定デバイスがオープンできなかった場合は 0 を返します。

USBM_OpenByUA()

TW_HANDLE USBM_OpenByUA(TW_CSTR pUA, long nUA)

pUA : [入力]ユーザーエリアと比較する文字列
nUA : 比較する文字数

ユーザーエリアの値を読み出し、指定文字列と一致したデバイスのハンドルを返します。指定デバイスがオープンできなかった場合は 0 を返します。

ユーザーエリアはデバイス上のコンフィギュレーション情報用 EEPROM 領域に確保された領域で、設定ツールを使用して文字列を書き込むことができます。使用できるユーザーエリアは 8 バイトです。

□ デバイスの状態･初期化に関する関数

USBM_PRODUCT_INFO 構造体

```
typedef struct {
  WORD Empty;           //必ず 0 とする
  WORD wRsv;            //予約
  UUID ID;              //UUID 構造体(UUID の保存)
  DWORD Number;         //任意の番号
  char Description[32]; //製品情報
  char Manufacture[32]; //製造元情報
  BYTE Reserve[40];     //予約
} USBM_PRODUCT_INFO;
```

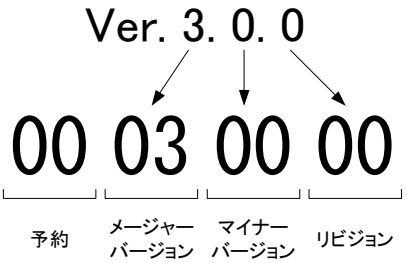
USBM_ReadPI() 関数を使用して製品情報を取得する際に使用します。PI エリアの詳細については製品マニュアルを参照してください。

USBM_ReadVersionA()

```
TW_STATUS USBM_ReadVersionA(TW_HANDLE hDev, DWORD *pVersion)
```

hDev　　: デバイスのハンドル
pVersion : [出力]読み出したバージョンの格納先

USBM_ReadVersion()よりも詳しいファームウェアバージョンを取得します。返される値の最上位 8 ビットは予約です。次の 8 ビットはメジャーバージョンです。大幅な機能変更の際に更新されます。次の 8 ビットはマイナーバージョンです。比較的小さな機能変更の際に更新されます。最下位 8 ビットはリビジョンです。バグフィックスや各インタフェース固有の機能変更の際に更新されます。例えばファームウェアバージョンが Ver.3.0.0 の場合、pVersion には H'00030000 が格納されます。



USBM_ReadVersion()

```
TW_STATUS USBM_ReadVersion(TW_HANDLE hDev, DWORD *pVer)
```

hDev : デバイスのハンドル
pVer : [出力]読み出したバージョンの格納先

ハードウェアのバージョンの上位 2 桁を読み取ります。バージョンは、x.x の形式を 10 倍にして整数として返します。

USBM_ReadPI()

```
TW_STATUS USBM_ReadPI(TW_HANDLE hDev, USBM_PRODUCT_INFO *pInfo)
```

hDev ： デバイスのハンドル
pInfo ： [出力]製品情報の格納先

PI エリアから製品情報を読み出します。
※この関数は Ver.2.1 以降のファームウェアが必要です。

USBM_ReadStatus()

```
TW_STATUS USBM_ReadStatus(TW_HANDLE hDev, DWORD *pStat)
```

hDev ： デバイスのハンドル
pStat ： [出力]読み出したステータスの格納先へのポインタ
USBM_STS_TIMEOUT 処理がタイムアウトした
USBM_STS_ILLEGAL_ACCESS 不正なアドレスへのアクセス

デバイスのステータスを読み取ります。上で定義されるステータスビットの組み合わせが返されます。
一度読み出すとデバイスのステータスはクリアされます。

USBM_InitializeA()

```
TW_STATUS USBM_InitializeA(TW_HANDLE hDev, DWORD InitOption)
```

hDev ： デバイスのハンドル
InitOption ： 初期化する機能を指定

USBM_INIT_PORT_DIR(0x0001)	ポートの方向、プルアップを初期化
USBM_INIT_PORT_DATA(0x0002)	ポートのデータを初期化
USBM_INIT_BUS(0x0004)	外部バス設定を初期化
USBM_INIT_DMA(0x0008)	DMA を初期化
USBM_INIT_TIMER(0x0010)	16 ビットタイマを初期化
USBM_INIT_AD(0x0020)	AD コンバータを初期化
USBM_INIT_SCI(0x0040)	シリアルポートを初期化
USBM_INIT_PC(0x0080)	パルスカウンタを初期化
USBM_INIT_TCPY(0x0100)	タイマコピーを初期化
USBM_INIT_ALL(0xffffffff)	全ての機能を初期化

デバイスを初期化します。InitOption で初期化する機能を選択できます。
※この関数は Ver.3.2.1 以降のファームウェアが必要です。

USBM_Initialize()

```
TW_STATUS USBM_Initialize(TW_HANDLE hDev)
```

hDev ： デバイスのハンドル

デバイスを初期化します。リセットと違いシステムファームで使用されないレジスタなどは初期化されません。

□ バス操作関数

USBM_AddressEnable()

```
TW_STATUS USBM_AddressEnable(TW_HANDLE hDev, long nBits)
```

hDev ： デバイスのハンドル
nBits ： 有効にするビット数(0,8,16,20)

アドレス出力を有効/無効にします。LSB から指定ビット数が有効になります。アドレスバスはポート 1, ポート 2,ポート 5 と共用です。アドレスを出力すると入力ポートとしては使用できませんので注意してください。アドレスは一部だけを出力できますので出力していないポートは入力ポートのまま使用できます。

USBM_BusSetWait()

```
TW_STATUS USBM_BusSetWait(TW_HANDLE hDev, long Area, BYTE Wait)
```

hDev ： デバイスのハンドル
Area ： メモリエリア
　　USBM_AREA0(0x01) エリア 0
　　USBM_AREA2(0x04) エリア 2
　　USBM_AREA3(0x08) エリア 3
　　USBM_AREA5(0x10) エリア 5

Wait ： ウェイト数
　　USBM_BUS_WAIT0(0)　ウェイトなし
　　USBM_BUS_WAIT1(1)　ウェイト 1
　　USBM_BUS_WAIT2(2)　ウェイト 2
　　USBM_BUS_WAIT3(3)　ウェイト 3
　　USBM_BUS_2STATE(4) 2 ステートアクセス

メモリエリアのソフトウェアウェイトを設定します。
Wait を 0-3 の範囲とすると指定エリアが 3 ステートアクセスに設定され、指定のウェイトが挿入されます。 Wait を USBM_BUS_2STATE とすると 2 ステートアクセスに設定されます。この場合ウェイトを指定することはできません。
アクセス速度は USBM_BUS_2STATE とした場合が最も高速で、それ以外は 0,1,2,3 の順となり、3 の場合が最も低速です。初期状態ではウェイト 1 に設定されています。

USBM_CSEnable()

```
TW_STATUS USBM_CSEnable(TW_HANDLE hDev, long Area)
```

hDev ： デバイスのハンドル
Area ： CS 信号を有効にするエリア
　　USBM_AREA2 CS2#を出力
　　USBM_AREA3 CS3#を出力

CS 信号を有効にするエリアを指定します。使用できる CS 信号のうち CS0#,CS5#は常に出力されますので USBM_AREA2 と USBM_AREA3 が指定できます。両方出力する場合は OR で結合して指定してください (USBM_AREA2 | USBM_AREA3)。CS2#を有効にした場合 PC3#が、CS3#を有効にした場合 PC2#は使用できませんので該当のパルスカウンタは必ず無効にしてください。

□ ポート操作関数

USBM_PortWrite8A()

```
TW_STATUS USBM_PortWrite8A(TW_HANDLE hDev, DWORD Port, BYTE Data, BYTE Mask)
```

hDev : デバイスのハンドル
Port : 書き込むアドレス
Data : 書き込むデータ
Mask : マスクデータ。1のビットだけ反映されます。

マイコンのPortで指定されるアドレスに1バイトのデータを書き込みます。マイコンの制御レジスタにも書き込みが行えますので、定数以外のアドレスを指定する場合には注意が必要です。Maskが0となっているビットには影響を与えません。
※この関数にはVer.2.2以降のファームウェアが必要です。

USBM_PortRead8()

```
TW_STATUS USBM_PortRead8(TW_HANDLE hDev, DWORD Port, BYTE *pData)
```

hDev　: デバイスのハンドル
Port　: 読み出すアドレス
pData　: [出力]読み出したデータの格納先

マイコンのPortで指定されるアドレスから1バイトのデータを読み出します。

USBM_PortWrite16()

```
TW_STATUS USBM_PortWrite16(TW_HANDLE hDev, DWORD Port, WORD Data)
```

hDev : デバイスのハンドル
Port : 書き込むアドレス
Data : 書き込むデータ

マイコンのPortで指定されるアドレスに2バイトのデータを書き込みます。16ビットのデータを同時にアクセスしたい場合に使用します。マイコンの制御レジスタにも書き込みが行えますので、定数以外のアドレスを指定する場合には注意が必要です。Portアドレスは偶数番地でなければなりません。バイトアクセス専用の空間では1バイトずつ2回ライトされます。

USBM_PortRead16()

```
TW_STATUS USBM_PortRead16(TW_HANDLE hDev, DWORD Port, WORD *pData)
```

hDev　: デバイスのハンドル
Port　: 読み出すアドレス
pData　: [出力]読み出したデータの格納先

マイコンのPortで指定されるアドレスから2バイトのデータを読み出します。Portアドレスは偶数番地でなければなりません。バイトアクセス専用の空間では1バイトずつ2回リードされます。

USBM_PortBWrite()

```
TW_STATUS USBM_PortBWrite(TW_HANDLE hDev, DWORD Port,
                          void *pData, long nData, long Inc, BYTE Dma)
```

hDev　: デバイスのハンドル
Port　: 書き込むアドレス
pData　: [入力]書き込むデータへのポインタ
nData　: データのバイト数(0～65536)
Inc　: マイコンのアドレスをインクリメントするかどうかを示すフラグ(TRUE,FALSE)
Dma　: DMA を使用するかどうかのフラグ(TURE,FALSE)

マイコンの Port で指定されるアドレスに nData で指定される大きさのデータを転送します。Inc を FALSE とするとマイコン側のアドレスがインクリメントされませんので FIFO のようなデバイスに書き込みを行えます。
Dma を TRUE とするとマイコン内蔵の DMA を使用してデータを転送します。DMA を使用すると転送速度が向上します。ただし、USB デバイスの場合、転送終了時に PA0/TCLKA 端子に TEND#信号を出力するので注意が必要です。PA0/TCLKA 端子に出力が出てはいけない場合には Dma=FALSE として呼び出してください。

USBM_PortBRead()

```
TW_STATUS USBM_PortBRead(TW_HANDLE hDev, DWORD Port,
                         void *pData, long nData, long Inc, BYTE Dma)
```

hDev　: デバイスのハンドル
Port　: 読み出すアドレス
pData　: [出力]読み出したデータの格納先へのポインタ
nData　: データのバイト数(0～65536)
Inc　: マイコンのアドレスをインクリメントするかどうかを示すフラグ(TRUE,FALSE)
Dma　: DMA を使用するかどうかのフラグ(TRUE,FALSE)

マイコンの Port で指定されるアドレスから nData で指定される大きさのデータを読み出します。Inc を FALSE とするとマイコン側のアドレスがインクリメントされませんので FIFO のようなデバイスから読み出しを行えます。
Dma を TRUE とするとマイコン内蔵の DMA を使用してデータを転送します。DMA を使用すると転送速度が向上します。ただし、USB デバイスの場合、転送終了時に PA1/TCLKB 端子に TEND#信号を出力するので注意が必要です。PA1/TCLKB 端子に出力が出てはいけない場合には Dma=FALSE として呼び出してください。

USBM_PortCopy8()

```
TW_STATUS USBM_PortCopy8(TW_HANDLE hDev, DWORD SrcPort, DWORD DstPort)
```

hDev　: デバイスのハンドル
SrcPort : コピー元
DstPort : コピー先

SrcPort で示されるアドレスから 1 バイトを読み取り DstPort で示されるアドレスに書込みます。

USBM_PortCopy16()

```
TW_STATUS USBM_PortCopy16(TW_HANDLE hDev, DWORD SrcPort, DWORD DstPort)
```

hDev　　: デバイスのハンドル
SrcPort : コピー元
DstPort : コピー先

SrcPort で示されるアドレスから 16 ビットのデータを読み取り DstPort で示されるアドレスに書込みます。アドレスはどちらも偶数番地でなければなりません。バイトアクセス専用の空間では 1 バイトずつ 2 回アクセスされます。

USBM_PortBCopy()

```
TW_STATUS USBM_PortBCopy(TW_HANDLE hDev, DWORD SrcPort, DWORD DstPort,
                    long nData,long SrcInc,long DstInc)
```

hDev　　: デバイスのハンドル
SrcPort : コピー元
DstPort : コピー先
nData　 : バイト数(0～65536)
SrcInc　: コピー元アドレスのインクリメント(1,0,-1)
DstInc　: コピー先アドレスのインクリメント(1,0,-1)

SrcPort から DstPort へ指定バイト数だけコピーを行います。DMA のバーストモードで転送しますので、転送が終わるまでマイコンは割込みも含めて他の処理を行えません。

USBM_PortSetDir()

```
TW_STATUS USBM_PortSetDir(TW_HANDLE hDev, DWORD Port, BYTE Bits)
```

hDev : デバイスのハンドル
Port : 方向を指定するポート
　　USBM_P1 P10～P17 端子の方向を設定
　　USBM_P2 P20～P27 端子の方向を設定
　　USBM_P4 P40～P47 端子の方向を設定
　　USBM_P5 P50～P53 端子の方向を設定
　　USBM_PA PA0～PA7 端子の方向を設定

Bits : 方向(1 のビット:出力,0 のビット:入力)

ポートの入力/出力を切替えます。Bits の 0 のビットは入力、1 のビットは出力となります。
注:P1,P2,P5 は入力専用で出力にするとバスのアドレス出力になります。

USBM_PortWrite8()

TW_STATUS USBM_PortWrite8(TW_HANDLE hDev, DWORD Port, BYTE Data)

hDev : デバイスのハンドル
Port : 書き込むアドレス
Data : 書き込むデータ

マイコンの Port で指定されるアドレスに 1 バイトのデータを書き込みます。マイコンの制御レジスタにも書き込みが行えますので、定数以外のアドレスを指定する場合には注意が必要です

□ 16 ビットタイマ操作関数

USBM_TimerStartA()

TW_STATUS USBM_TimerStartA(TW_HANDLE hDev, BYTE Bits, long TrigPC, long Repeat, long Cmp)

hDev ： デバイスのハンドル
Bits ： スタートするタイマチャンネル
ビット 0(LSB) 1 の場合、チャンネル 0 をスタート
ビット 1 1 の場合、チャンネル 1 をスタート
ビット 2 1 の場合、チャンネル 2 をスタート
TrigPC ： 起動のトリガとなるパルスカウンタチャンネル
-1 直ちにタイマを起動します
0～3 指定された PCx#信号でタイマを起動します
Repeat ： 起動トリガを繰り返し機能させるかどうかのフラグ(TRUE,FALSE)
Cmp ： パルスカウンタのコンペア値(1～2147483647)

16 ビットタイマの指定チャンネルをスタートします。USBM_TimerStart()との違いはスタートするチャンネル以外に影響を与えないことです。
また、TrigPC にパルスカウンタのチャンネルを指定すると、指定したチャンネルはすぐには起動せず、パルスカウンタへの入力待ちになります。パルスカウンタにパルスが入力されると 8us～20us 以内にタイマが起動されます。この場合、Repeat が真ならパルスカウンタ入力は繰り返し受付られ、その度にタイマが起動されます。
Cmp は通常 1 ですが、パルスカウンタへ複数のパルス入力によりタイマを起動する場合は回数を設定してください。
パルスカウンタを起動要因とすると、指定のパルスカウンタがカウントを開始しますので不要になった場合は USBM_PCStop()関数を呼び出して停止してください。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_TimerStop()

TW_STATUS USBM_TimerStop(TW_HANDLE hDev, BYTE Bits, long TrigPC, long Repeat, long Cmp)

hDev　: デバイスのハンドル
Bits　: ストップするタイマチャンネル
　　ビット0(LSB) 1の場合、チャンネル0を停止
　　ビット1　　1の場合、チャンネル1を停止
　　ビット2　　1の場合、チャンネル2を停止
TrigPC : 終了のトリガとなるパルスカウンタチャンネル
　　-1　直ちにタイマを停止します
　　0～3　指定されたPCx#信号でタイマを停止します
Repeat : 終了トリガを繰り返し機能させるかどうかのフラグ(TRUE,FALSE)
Cmp　: パルスカウンタのコンペア値(1～2147483647)

16ビットタイマの指定チャンネルを停止します。USBM_TimerStart()との違いは、停止するチャンネル以外に影響を与えないことです。
また、TrigPCにパルスカウンタのチャンネルを指定すると、指定したチャンネルはすぐには停止せず、パルスカウンタへの入力待ちになります。パルスカウンタにパルスが入力されると 8us～20us 以内にタイマが停止されます。この場合、Repeatが真ならパルスカウンタ入力は繰り返し受付られ、その度にタイマが停止されます。
Cmpは通常1ですが、パルスカウンタへ複数のパルス入力によりタイマを停止する場合は回数を設定してください。
パルスカウンタを停止要因とすると、指定のパルスカウンタがカウントを開始しますので不要になった場合はUSBM_PCStop()関数を呼び出して停止してください。

※この関数はVer.2.0以降のファームウェアが必要です。

USBM_TimerSetCnt()

TW_STATUS USBM_TimerSetCnt(TW_HANDLE hDev, long CH, short Cnt)

hDev : デバイスのハンドル
CH　: タイマチャンネル(0,1,2)
Cnt　: 設定する値

タイマのカウンタ値を設定します。チャンネル間でPWMの位相を調整する場合などに使用できます。

USBM_TimerReadCnt()

TW_STATUS USBM_TimerReadCnt(TW_HANDLE hDev, long CH, short *pCnt)

hDev　: デバイスのハンドル
CH　　: タイマチャンネル(0,1,2)
pCnt　: [出力]カウント値の格納先

タイマのカウンタ値を読み出します。

USBM_TimerSetClk()

TW_STATUS USBM_TimerSetClk(TW_HANDLE hDev, long CH, BYTE Bits)

hDev : デバイスのハンドル
CH : タイマチャンネル(0,1,2)
Bits : USBM_TCLK25000 25MHz
USBM_TCLK12500 12.5MHz
USBM_TCLK6250 6250kHz
USBM_TCLK3125 3125kHz
USBM_TCLKA TCLKA からの外部入力
USBM_TCLKB TCLKB からの外部入力

16 ビットタイマに使用するクロックを選択します。TCLKA または TCLKB を選択すると PA0 または PA1 ピンをクロック入力として使用します。USB デバイスの場合、TCLKA,TCLKB は USBM_PortBWrite()または USBM_PortBRead()関数で DMA を使用すると一時的に TEND#信号として出力ピンに設定されてしまいますので同時に使用することはできません。

USBM_TimerSetPulse()

TW_STATUS USBM_TimerSetPulse(TW_HANDLE hDev, long CH, WORD LtoH, WORD HtoL)

hDev : デバイスのハンドル
CH : タイマチャンネル(0,1,2)
LtoH : タイマ出力(TIOCA ピン)が 1 になるカウント値を指定(1～65535)
HtoL : タイマ出力(TIOCA ピン)が 0 になるカウント値を指定(1～65535)

タイマ出力ピンが 1 になるタイミングと 0 になるタイミングを指定します。デフォルトの設定ではタイマは PWM モードに設定され TIOCA ピンだけが出力となります。すなわち 16 ビットタイマのカウンタが LtoH の値と等しくなると TIOCA ピンに 1 が出力され、HtoL と等しくなると 0 が出力されます。またデフォルトでは 0 になるタイミングでカウンタがリセットされるように設定されます。つまり、HtoL の値でカウンタ出力の周期を LtoH の値でデューティ比を決定できます。チャンネル間の位相差はタイマスタート前に USBM_TimerSetCnt()を呼び出すことで調整できます。パルスの周期は (HtoL+1)/fclk[sec] 、デューティは (1-(LtoH+1)/(HtoL+1))*100[%]となります。

USBM_TimerSetLevel()

TW_STATUS USBM_TimerSetLevel(TW_HANDLE hDev, BYTE Bits)

hDev : デバイスのハンドル
Bits : 各端子に対応するビット
0 ビット(LSB) TIOCA0
1 ビット TIOCB0
2 ビット TIOCA1
3 ビット TIOCB1
4 ビット TIOCA2
5 ビット TIOCB2

タイマ出力に設定されているピンのレベルを変更します。1 にセットすると対応する出力ピンが 1 出力になります。タイマ出力に設定されていないピンは影響を受けません。デフォルトでは有効にしたチャンネルの TIOCA ピンだけが出力になります。

USBM_TimerEnable()

TW_STATUS USBM_TimerEnable(TW_HANDLE hDev, BYTE Bits, long MDF)

hDev : デバイスのハンドル
Bits : PWM 出力を行うチャンネルを示すビット
　　0 ビット(LSB) タイマ 0
　　1 ビット　　タイマ 1
　　2 ビット　　タイマ 2
MDF　: 0 以外を指定するとチャンネル 2 を位相計数モードに設定します。

指定チャンネルを PWM モードに設定します。PWM モードに設定したチャンネルの TIOCA ピンはタイマ出力となります。MDF を真にするとチャンネル 2 を位相計数モードに設定します(位相計数モードの詳細は H8 マイコンのドキュメントを参照してください。)。位相計数モードになると自動的に TCLKA,TCLKB ピンがクロック入力となります。

USBM_TimerSetCmp()

TW_STATUS USBM_TimerSetCmp(TW_HANDLE hDev, short CmpA, short CmpB)

hDev　: デバイスのハンドル
CmpA　: コンペアレジスタ A
CmpB　: コンペアレジスタ B

チャンネル 2 を位相計数モードで使用する場合のコンペア値を設定します。コンペア値とチャンネル 2 のカウンタ値が一致したときの動作は USBM_TimerSetCmpOut()関数で設定します。

USBM_TimerSetCmpOut()

TW_STATUS USBM_TimerSetCmpOut(TW_HANDLE hDev, long CmpReg, DWORD Port, BYTE Data)

hDev　: デバイスのハンドル
CmpReg : 設定するコンペアレジスタ
　　0 コンペアレジスタ A
　　1 コンペアレジスタ B
Port　: 値を書き込むアドレス
Data　: 書き込むデータ

タイマチャンネル 2 のコンペアアウトを設定します。コンペアアウトを設定したコンペアレジスタの値とタイマチャンネル 2 のカウント値が一致すると Port で指定されたアドレスに Data がライトされます。出力が行われないようにするには Port を 0 にして呼び出してください。タイマチャンネル 2 が PWM モードでも位相計数モードでも機能します。PWM モードでは 1 になるタイミングがコンペアレジスタ A とのマッチングで、0 になるタイミング(同時にカウンタが 0 にクリアされるタイミング)がコンペアレジスタ B とのマッチングです。

USBM_TimerSetCmpClr()

TW_STATUS USBM_TimerSetCmpClr(TW_HANDLE hDev, BYTE Bits)

hDev : デバイスのハンドル
Bits : コンペアマッチでカウンタをクリアするコンペアレジスタを指定
0 ビット(LSB)が 1 のときコンペアレジスタ A とのマッチングでカウンタがクリアされます
1 ビットが 1 のときコンペアレジスタ B とのマッチングでカウンタがクリアされます

指定されたコンペアレジスタとコンペアマッチが発生したときタイマチャンネル 2 のカウンタがクリアされます。

USBM_TimerSetCapture()

TW_STATUS USBM_TimerSetCapture(TW_HANDLE hDev, long CH, BYTE EdgeA, BYTE EdgeB)

hDev : デバイスのハンドル
CH : インプットキャプチャ設定を行うタイマチャンネル(0,1,2)
EdgeA : TIOCA 端子でキャプチャする信号エッジ
0 キャプチャは行わない
USBM_CAPT_RISE 立ち上がりをキャプチャ
USBM_CAPT_FALL 立ち下がりをキャプチャ
USBM_CAPT_BOTH 立ち上がり、立ち下がり両方でキャプチャ
EdgeB : TIOCB 端子でキャプチャする信号エッジ(指定できる値は EdgeA と同じ)

16 ビットタイマの指定チャンネルをインプットキャプチャモードに設定します。インプットキャプチャに設定すると該当チャンネルの TIOCA,TIOCB 端子は入力端子となり、引数で指定された信号のエッジを検出すると、そのときのカウンタの値がマイコンの GRA,GRB レジスタに転送されます。
この機能を利用することで、TIOCA 端子で信号を検出してから、TIOCB で信号が検出するまでの時間差をタイマクロックのカウント値で測定することができます。クロックの設定には USBM_TimerSetClk()関数を使用してください。カウンタは TIOCB 端子が信号を検出したときにクリアされます。
動作中は信号が入力される度に、何度でもキャプチャされます。

USBM_TimerReadCaptureCnt()

TW_STATUS USBM_TimerReadCaptureCnt(TW_HANDLE hDev, long CH, long *pCntA, long *pCntB)

hDev : デバイスのハンドル
CH : タイマチャンネル(0,1,2)
pCntA : [出力]GRA レジスタ値の格納先へのポインタ
pCntB : [出力]GRB レジスタ値の格納先へのポインタ

インプットキャプチャした結果を読み出します。

USBM_TimerSetCaptureCnt()

TW_STATUS USBM_TimerSetCaptureCnt(TW_HANDLE hDev, long CH, long CntA, long CntB)

hDev : デバイスのハンドル
CH : タイマチャンネル(0,1,2)
CntA : GRA レジスタにセットする値(0～65535)
CntB : GRB レジスタにセットする値(0～65535)

GRA、GRB レジスタに値をセットします。主にインプットキャプチャを行う前にレジスタをクリアするのに使用します。

USBM_TimerSetSinglePulse()

TW_STATUS USBM_TimerSetSinglePulse(TW_HANDLE hDev, long CH, long Setup, long Width, long Pos)

hDev : デバイスのハンドル
CH : 使用する 16 ビットタイマチャンネル(0,1)
Setup : タイマ起動からパルス出力までの時間をクロック数で指定(1～65534)
Width : パルス幅をクロック数で指定(1～65534)
Pos : TRUE の場合、正のパルス、FALSE の場合、負のパルスが出力されます

16 ビットタイマを利用して単発のパルスを発生させます。Setup と Width を足した値が 65535 を超えてはいけません。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_TimerStart()

TW_STATUS USBM_TimerStart(TW_HANDLE hDev, BYTE Bits)

hDev : デバイスのハンドル
Bits : 0 ビット(LSB) タイマ 0
1 ビット タイマ 1
2 ビット タイマ 2

タイマのスタート/ストップを制御します。1 にセットしたビットに対応するタイマがスタートし、0 にセットしたビットに対応するタイマはストップします。

□ AD コンバータ操作関数

USBM_ADRead()

TW_STATUS USBM_ADRead(TW_HANDLE hDev, WORD *pData, long CH, long Scan)

hDev : デバイスのハンドル
pData : [出力]読み出したデータの格納先へのポインタ
CH : AD コンバータのチャンネル(0,1,2,3)
Scan : 複数チャンネルをスキャンするかどうかのフラグ(TRUE,FALSE)

Scan が FALSE のとき指定されたチャンネルの AD 変換結果を pData 位置に格納します。Scan が TRUE のときは 0 から指定されたチャンネルまでを AD 変換し pData の位置から格納します。例えば Scan=TRUE で CH=2 の場合、0～2 チャンネルの 3 チャンネル分の変換結果を pData 位置から 3 ワード分格納します。pData 領域の大きさはチェックされませんのでチャンネル数に合わせて呼び出し側で確保してください。

USBM_ADBRead()

TW_STATUS USBM_ADBRead(TW_HANDLE hDev, WORD *pData, DWORD nData, long CH, DWORD *pnRead)

hDev : デバイスのハンドル
pData : [出力]読み出したデータの格納先へのポインタ
nData : 読み出すデータ数(バイト単位ではなくワードデータの数、1～4294967295)
CH : AD コンバータのチャンネル(0,1,2,3)
pnRead : 実際に読み出したデータ数の格納先(バイト単位ではなくワードデータの数)

指定チャンネルの AD コンバータで定期的に AD 変換し、そのデータを読み出します。変換タイミングはマイコン内蔵の 8 ビットタイマで作り出す方法と、外部トリガを入力する方法があります。デフォルトでは外部トリガの入力で変換が開始されますので、タイマを利用する場合には、予め USBM_ADSetCycle() 関数でクロックとコンペア値を設定しておいてください。
関数自体は USBM_SetTimeouts()で指定された時間でタイムアウトしますが、デバイスは指定されたデータ数を読み出すまで処理を続けます。中止させるためには USBM_Abort()関数を使用してください。

USBM_ADStart()

TW_STATUS USBM_ADStart(TW_HANDLE hDev, long nCnv, long CH, long Scan, long lByte, long Trig)

hDev　: デバイスのハンドル
nCnv　: 変換回数。複数チャンネルをスキャンする場合は、それぞれ nCnv 回変換されます。
　　-1 の場合中止されるまで繰り返します。(-1,1～2147483647)
CH　　: AD コンバータのチャンネル(0,1,2,3)
Scan　: 複数チャンネルをスキャンするかどうかのフラグ(TRUE,FALSE)
lByte : バイト読み出しするかどうかのフラグ(TRUE,FALSE)
Trig　: 内蔵タイマの起動を ADTRG#で行うかどうかのフラグ(TRUE,FALSE)

AD コンバータを起動し、断続的に変換を行います。変換結果は USBM_Read()関数で取り出してください。変換タイミングはマイコン内蔵の 8 ビットタイマで作り出す方法と、ADTRG#端子に信号を入力する方法があります。ADTRG#を選択した場合、ADC はトリガ入力待ちとなり、1 回のトリガにつき 1 回の変換を行います。タイマを選択した場合、タイマを起動し予め設定したサイクル毎に変換を行います。デフォルトでは外部トリガの入力で 変換が開始されますので、タイマを利用する場合には、予め USBM_ADSetCycle()関数でクロックとコンペア値を設定します。

Scan が FALSE のときは指定チャンネルの変換を指定された回数行います。
Scan が TRUE のときは 1 回のトリガにつき、0 から指定されたチャンネルまでを 1 回ずつ AD 変換します。例えば Scan=TRUE で CH=2 の場合、0～2 チャンネルの 3 チャンネル分の変換をトリガ毎に行いホストに転送します。

lByte が TRUE の場合は AD 変換結果の上位 8 ビットのみをホストに転送します。16 ビット転送を行う場合よりも、変換レートを上げることが可能となります。詳しくは「AD コンバータ」を参照してください。

変換タイミングをマイコンの内蔵タイマで作る場合でも、Trig を TRUE に設定すると外部トリガに変換を同期させることができます。この場合、USBM_ADSetCycle()で外部トリガを指定した場合と違い、ADTRG#入力で内蔵タイマを起動し、AD のトリガ信号自体はタイマから供給されますので、ADTRG#への入力は 1 度でよくなります。

ADBRead()関数との違いは、ADBRead()関数は指定したデータ数の変換が全て終了するまでデータが取り出せないのに対して、終了したデータだけを逐次取り出せることです。また、複数チャンネルの連続変換も可能です。

デバイスはこの関数を呼び出した後は、指定の変換回数が終了するまで、USBM_Abort()以外の関数を受付ませんのでご注意ください。また、変換終了後はバッファ内のデータを確実に取り出してください。データが残っている場合、後の関数が誤動作します。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_ADSetCycle()

TW_STATUS USBM_ADSetCycle(TW_HANDLE hDev, BYTE Cmp, long CLK)

hDev : デバイスのハンドル
Cmp　: コンペアレジスタの値(1～255)
CLK　: クロックの選択
　　USBM_TCLK3125 3125kHz
　　USBM_TCLK390　390.625kHz
　　USBM_TCLK3　　約 3052Hz

USBM_ADBRead()、USBM_ADStart()で連続して AD 変換を行う場合の周期を設定します。CLK でクロックの周波数を選択し、Cmp の値でそのクロックを何回カウントしたときに変換を開始するかを決定します。CLK を 0 にして呼び出すとタイマではなく外部トリガ入力で変換を開始する設定となります。

USBM_ADCopy()

```
TW_STATUS USBM_ADCopy(TW_HANDLE hDev, DWORD DstPort, long nCnv, long CH, long CKS,
                      long DMA_CH, long DstInc)
```

hDev　　：デバイスのハンドル
DstPort ：変換結果を格納するマイコン内のアドレス
nCnv　　：変換回数。複数チャンネルをスキャンする場合は、それぞれ nCnv 回変換されます。(1～65536)
CH　　：終了チャンネル(0,1,2,3)
CKS　　：変換ステート(TRUE,FALSE)
DMA_CH　：使用する DMA チャンネル(0,1)
DstInc　：転送先アドレスをインクリメントするかどうかのフラグ(TRUE,FALSE)

AD 変換を連続で行い、変換結果をマイコン内のメモリに転送します。呼び出すと、USBM_ADSetCycle()で指定した変換周期はクリアされ、起動要因は外部トリガに固定されます。設定終了後、変換が終了したかどうかにかかわらず、関数はリターンします。その後、デバイスが ADTRG#の立ち下がりを検出すると、指定回数の変換を自動で行います。このとき変換は常に最大レートで行われます。また、変換は必ず 0 チャンネルから開始され、CH で指定したチャンネルまで変換されます。例えば、CH=2 のとき 0～2 までの 3 チャンネルの変換を nCnv 回行います。
指定回数の変換が終了したかどうかを調べるには USBM_ADReadCopyStatus()を使用してください。中断して終了する場合、及び変換終了後は USBM_ADStopCopy()を呼び出してください。

CKS が FALSE のときは最初の 1 サイクルの変換時間が最大 5.36us で、以降、連続変換中は 5.12us/チャンネルです。CKS を TRUE とすることで精度を犠牲にして、変換レートを上げることができます。この場合、最初の 1 サイクルの変換時間が最大で 2.8us で、以降、連続変換中は 2.64us/チャンネルとなります。変換精度については製品マニュアルを参照してください。

DstPort と nCnv の設定は慎重に行ってください。変換結果を格納する領域が、ユーザーメモリの範囲を超えてもチェックは行われません。使用できる範囲を超えて DMA 転送が行われると、マイコンの内部メモリの情報を破壊して正常に動作しなくなります。

DMA チャンネルを 1 つ利用しますので、該当チャンネルの DMA 転送は行えなくなります DMA0 を使用する場合には USBM_PortBWrite()の DMA 転送と USBM_PortBCopy()が使用できなくなり、DMA1 を使用した場合には USBM_PortBRead()の DMA 転送が使えません。また、DA コンバータへの自動転送も同じ番号のチャンネルが使用できなくなります。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_ADReadCopyBuffer()

TW_STATUS USBM_ADReadCopyBuffer(TW_HANDLE hDev, DWORD Port, WORD *pData, long nData, long Inc, BYTE Dma)

hDev　: デバイスのハンドル
Port　: 読み出すアドレス
pData　: [出力]読み出したデータの格納先へのポインタ
nData　: データの数(0～32768、ワード単位)
Inc　: マイコンのアドレスをインクリメントするかを示すフラグ(TRUE,FALSE)
Dma　: DMA を使用するかどうかのフラグ(TRUE,FALSE)

USBM_ADCopy()関数で変換されたデータをデバイス上のバッファから読み出すのに使用します。USBM_PortBRead()関数でも読み出せますが、マイコン上のデータは Windows パソコンとバイトオーダーが違うため変換が必要になります。この関数では読み出したデータを、パソコンと同じバイトオーダー(リトルエンディアン)に並び替えて返します。
Inc を FALSE とするとマイコン側のアドレスがインクリメントされませんので FIFO のようなデバイスから読み出しを行えます。
Dma を TRUE とするとマイコン内蔵の DMA を使用してデータを転送します。DMA を使用すると転送速度が向上します。ただし、USB デバイスの場合、転送終了時に PA1/TCLKB 端子に TEND#信号を出力するので注意が必要です。PA1/TCLKB 端子に出力が出てはいけない場合には Dma=FALSE として呼び出してください。

USBM_ADStopCopy()

TW_STATUS USBM_ADStopCopy(TW_HANDLE hDev, long DMA_CH)

hDev　: デバイスのハンドル
DMA_CH : USBM_ADCopy()で指定した DMA チャンネル(0,1)

USBM_ADCopy()の終了処理をします。中断して終了する場合にもこの関数を呼び出してください。
※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_ADReadCopyStatus()

TW_STATUS USBM_ADReadCopyStatus(TW_HANDLE hDev, long *pnCnv, long DMA_CH)

hDev　: デバイスのハンドル
pnCnv　: [出力]残りの変換数の格納先
DMA_CH : USBM_ADCopy()で指定した DMA チャンネル(0,1)

USBM_ADCopy()の残りの変換回数を調べます。
※この関数は Ver.2.0 以降のファームウェアが必要です。

□ DA コンバータ操作関数

USBM_DASetParm()

TW_STATUS USBM_DASetParm(TW_HANDLE hDev,long CH,DWORD SrcPort,long nData,long lLoop)

hDev : デバイスのハンドル
CH : DA チャンネル(0,1)
SrcPort : 転送元アドレス
nData : データ数
繰り返さない場合、1～65536
繰り返し転送する場合、1～255
lLoop : 繰り返し転送するかのフラグ(TRUE,FALSE)

DA コンバータへの DMA 転送設定を行います。転送(変換)の周期は USBM_DASetCycle()で指定してください。転送元アドレスは一般にユーザーメモリ内のアドレスを指定し、予めデジタルデータを設定しておく必要があります。lLoop が真の場合、nData の変換終了後に、再びデータの先頭から変換を繰り返します。
DA0 の転送には DMA0 チャンネル、DA1 の転送には DMA1 チャンネルを利用しますので、該当チャンネルの DMA 転送は行えなくなります。具体的には DA0 の DMA 転送時には USBM_PortBWrite()の DMA 転送と USBM_PortBCopy()が使用できなくなり、DA1 の DMA 転送時には USBM_PortBRead()の DMA 転送が使えません。また、DA チャンネルと同じチャンネルの 16 ビットタイマも使用できなくなります。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_DAStart()

TW_STATUS USBM_DAStart(TW_HANDLE hDev, BYTE Bits)

hDev : デバイスのハンドル
Bits : スタートするチャンネル
ビット 0(LSB) 1 の場合 DA0 への転送(変換)を開始します。
ビット 1 1 の場合 DA1 への転送(変換)を開始します。

DA コンバータへのデータ転送を開始します。転送を開始する前に毎回 USBM_DASetParm()を呼び出してください。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_DAStop()

TW_STATUS _stdcall USBM_DAStop(TW_HANDLE hDev, BYTE Bits)

hDev : デバイスのハンドル
Bits : 終了するチャンネル
ビット 0(LSB) 1 の場合 DA0 への転送(変換)を終了します。
ビット 1 1 の場合 DA1 への転送(変換)を終了します。

DA コンバータへのデータ転送を終了します。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_DAReadStatus()

TW_STATUS USBM_DAReadStatus(TW_HANDLE hDev, long CH, long *pnData)

hDev　　: デバイスのハンドル
CH　　: DA のチャンネル(0,1)
pnData　: [出力]残りの変換データ数の格納先

DA コンバータの残り変換データ数を調べます。繰り返し変換中は常に 0 以外の値が返ります。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_DASetCycle()

TW_STATUS USBM_DASetCycle(TW_HANDLE hDev,long CH,long Cmp,long CLK)

hDev : デバイスのハンドル
CH　: 周期を設定する DA チャンネル
Cmp　: クロックをカウントする回数
CLK　: クロックを選択
　　USBM_TCLK25000 25MHz
　　USBM_TCLK12500 12.5MHz
　　USBM_TCLK6250　6250kHz
　　USBM_TCLK3125　3125kHz

DA コンバータの変換周期を設定します。変換周期は下のようになります。

Tc = (Cmp + 1) / fclk [sec] (fclk:CLK で選択した周波数)

DA0 のタイミング生成には 0 チャンネルの、DA1 のタイミングには 1 チャンネルの 16 ビットタイマを利用しますので該当チャンネルの 16 ビットタイマは他の用途には使用できなくなります。

※この関数は Ver.2.0 以降のファームウェアが必要です。

□ SCI(シリアルインタフェース)操作関数

USBM_SCIRead()

```
TW_STATUS USBM_SCIRead(TW_HANDLE hDev, long CH, void *pData, long nData, long *pnRead)
```

hDev : デバイスのハンドル
CH : チャンネル(0 のみ)
pData : [出力]格納先へのポインタ
nData : データのバイト数(0～255)
pnRead : 実際に読み出したバイト数の格納先

マイコンボード上のシリアルポートからデータを読み出します。関数自体は USBM_SetTimeouts()で指定された時間でタイムアウトしますが、デバイスは指定されたデータ数を読み出すまで処理を続けます。中止させるためには USBM_Abort()関数を使用してください。

USBM_SCIWrite()

```
TW_STATUS USBM_SCIWrite(TW_HANDLE hDev, long CH, void *pData, long nData)
```

hDev : デバイスのハンドル
CH : チャンネル(0 のみ)
pData : [入力]データへのポインタ
nData : データのバイト数(0～255)

マイコンボード上のシリアルポートからデータを出力します。

USBM_SCISetMode()

```
TW_STATUS USBM_SCISetMode(TW_HANDLE hDev, long CH, long Mode, long Baud)
```

hDev : デバイスのハンドル
CH : SCI チャンネル(0 のみ)
Mode : キャラクタのビット数,パリティ,ストップビットの設定の OR で結合します。
- USBM_SCI_DATA8 : 8 ビットデータ
- USBM_SCI_DATA7 : 7 ビットデータ
- USBM_SCI_NOPARITY : パリティなし
- USBM_SCI_EVEN : 偶数パリティ
- USBM_SCI_ODD : 奇数パリティ
- USBM_SCI_STOP1 : 1 ストップビット
- USBM_SCI_STOP2 : 2 ストップビット

Baud : ボーレート
- USBM_SCI_BAUD300 : 300bps
- USBM_SCI_BAUD600 : 600bps
- USBM_SCI_BAUD1200 : 1200bps
- USBM_SCI_BAUD2400 : 2400bps
- USBM_SCI_BAUD4800 : 4800bps
- USBM_SCI_BAUD9600 : 9600bps
- USBM_SCI_BAUD14400 : 14400bps
- USBM_SCI_BAUD19200 : 19200bps
- USBM_SCI_BAUD38400 : 38400bps

SCI チャンネルの設定を行います。使用できるのは調歩同期のみです。

USBM_SCISetDelimiter()

```
TW_STATUS USBM_SCISetDelimiter(TW_HANDLE hDev, long CH, char *pDelimiter, long nDelimiter)
```

hDev : デバイスのハンドル
CH : SCI チャンネル(0 のみ)
pDelimiter : [入力]デリミタ文字へのポインタ
nDelimiter : デリミタ文字の数(0～2)

SCI チャンネルのデリミタ文字を指定します。デリミタ文字(1 文字または 2 文字)が現れると、USBM_SCIRead()関数はシリアルポートからの読み取りを中止し、読み取り指定バイトまで 0 を埋めてデータを返します。

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_SCIReadStatus()

```
TW_STATUS USBM_SCIReadStatus(TW_HANDLE hDev, long CH, BYTE *pStatus, long *pnReceive)
```

hDev : デバイスのハンドル
CH : チャンネル(0 のみ)
pStatus : [出力]ステータスの格納先へのポインタ
0(LSB)ビット～2 ビット 0 です
3 ビット パリティエラーが起こった場合に 1 になります
4 ビット フレーミングエラーが起こった場合に 1 になります
5 ビット オーバーランエラーが起こった場合に 1 になります
6 ビット～7 ビット(MSB) 0 です
pnReceive : [出力]受信データ数の格納先

SCI のステータス情報と受信バッファに格納されているデータ数を読み出します。

□ パルスカウンタ操作関数

USBM_PCSetCnt()

```
TW_STATUS USBM_PCSetCnt(TW_HANDLE hDev, long CH, long Cnt)
```

hDev : デバイスのハンドル
CH　: パルスカウンタのチャンネル(0～3)
Cnt　: カウンタの値(-2147483648～2147483647)

パルスカウンタに値を設定します。

USBM_PCSetCmp()

```
TW_STATUS USBM_PCSetCmp(TW_HANDLE hDev, long CH, long Cmp)
```

hDev : デバイスのハンドル
CH　: パルスカウンタのチャンネル(0～3)
Cmp　: コンペア値(-2147483648～2147483647)

パルスカウンタのコンペア値を設定します。設定を行うことでパルスカウンタのカウント値がこの値と一致したときにカウンタ値がリセットされるようにしたり、指定ポートにデータを出力したりするようにできます。

USBM_PCReadCnt()

```
TW_STATUS USBM_PCReadCnt(TW_HANDLE hDev, long CH, long *pCnt)
```

hDev : デバイスのハンドル
CH　: パルスカウンタのチャンネル(0～3)
　　USBM_PC_ALL を指定すると 0～3 チャンネル全てを読み出します
pCnt : [出力]カウンタの値の格納先

パルスカウンタの値を読み出します。全部のチャンネルを呼び出す場合は 4 チャンネル分の領域を pCnt に指定してください。

USBM_PCSetControl()

```
TW_STATUS USBM_PCSetControl(TW_HANDLE hDev, long CH, BYTE Bits)
```

hDev : デバイスのハンドル
CH　: チャンネル(0,1,2,3)
Bits : 0 ビット(LSB) '1'にすると PC0 の入力でカウンタが 0 にクリアされます
　　1 ビット　　'1'にするとコンペアマッチでカウンタが 0 にクリアされます
　　4,5 ビット　'00'にするとパルス入力で無条件にカウントアップします
　　'01'にすると条件ビットが真のときダウン、偽のときアップ
　　'10'にすると条件ビットが真のときアップ、偽のときダウン
　　'11'にすると条件ビットが真のときアップ、偽のときカウントしません
　　上記以外のビットは予約です。0 にしてください。

パルスカウンタの制御方法を指定します。Bits 下位 2 ビットでリセット条件を 4,5 ビットでカウントの方法を指定します。下位 2 ビットがどちらも 0 の場合、カウンタはクリアされません。条件ビットとは USBM_PCSetCondBit()関数で指定されるビットです。

USBM_PCSetCondBit()

```
TW_STATUS USBM_PCSetCondBit(TW_HANDLE hDev, long CH, DWORD Port, BYTE Mask)
```

hDev : デバイスのハンドル
CH　: チャンネル(0,1,2,3)
Port : コンディションを読み取るアドレス
Mask : Port アドレスの値とアンドを取るマスク

パルスが入力されたときにカウンタの増減を決める条件ビットを指定します。条件ビットは Port アドレスから読み取られた値と Mask 値の AND(論理積)をとって決定されます。例えば 2 相のロータリーエンコーダー入力の位相関係でカウンタの増減を切替えるような場合に利用できます。

USBM_PCSetCmpOut()

```
TW_STATUS USBM_PCSetCmpOut(TW_HANDLE hDev, long CH, DWORD Port, BYTE Data)
```

hDev : デバイスのハンドル
CH　: チャンネル(0,1,2,3)
Port : コンペアマッチ時に書き込むアドレス
Data : コンペアマッチ時に書き込むデータ

コンペアマッチ時に行うポート出力を設定します。パルスカウンタの値がコンペア値と一致すると Port アドレスに Data を出力します。ポート操作が必要ない場合は Port=0 としてください。

USBM_PCStartA()

```
TW_STATUS USBM_PCStartA(TW_HANDLE hDev, BYTE Bits)
```

hDev : デバイスのハンドル
Bits : スタートするチャンネルをビットで指定
　　USBM_PC0(0x10) : チャンネル 0 を許可
　　USBM_PC1(0x20) : チャンネル 1 を許可
　　USBM_PC2(0x02) : チャンネル 2 を許可
　　USBM_PC3(0x04) : チャンネル 3 を許可

パルスカウントを許可します。許可するチャンネルが複数ある場合には上記の定数を OR で結合して Bits に指定してください。USBM_PCStart()関数との違いは、スタートするように指定したチャンネル以外に影響を与えないことです。

USBM_PCStop()

```
TW_STATUS USBM_PCStop(TW_HANDLE hDev, BYTE Bits)
```

hDev : デバイスのハンドル
Bits : ストップするチャンネルをビットで指定
　　USBM_PC0(0x10) : チャンネル 0 のカウントを終了
　　USBM_PC1(0x20) : チャンネル 1 のカウントを終了
　　USBM_PC2(0x02) : チャンネル 2 のカウントを終了
　　USBM_PC3(0x04) : チャンネル 3 のカウントを終了

指定チャンネルのパルスカウントを終了します。終了するチャンネルが複数ある場合には上記の定数を OR で結合して Bits に指定してください。USBM_PCStart()関数との違いは、終了するように指定したチャンネル以外に影響を与えないことです。

USBM_PCStart()

```
TW_STATUS USBM_PCStart(TW_HANDLE hDev, BYTE Bits)
```

hDev : デバイスのハンドル
Bits : スタートするチャンネルをビットで指定
　　USBM_PC0(0x10) : チャンネル 0 を許可
　　USBM_PC1(0x20) : チャンネル 1 を許可
　　USBM_PC2(0x02) : チャンネル 2 を許可
　　USBM_PC3(0x04) : チャンネル 3 を許可

パルスカウントを許可します。許可するチャンネルが複数ある場合には上記の定数を OR で結合して Bits に指定してください。指定されたチャンネル以外はカウントを終了します。

□ タイマコピー操作関数

USBM_TCPYSetClk()

```
TW_STATUS USBM_TCPYSetClk(TW_HANDLE hDev, long CH, long CLK)
```

```
hDev : デバイスのハンドル
CH   : クロックを設定するチャンネル(0,1)
CLK  : USBM_TCLK3125  3125kHz
       USBM_TCLK390   390.625kHz
       USBM_TCLK3     3052Hz
       USBM_TCLKUP    外部クロックの立ち上がり
       USBM_TCLKDOWN  外部クロックの立ち下がり
       USBM_TCLKBOTH  外部クロックの両エッジ
```

タイマコピーに使用するクロックを選択します。外部クロックはチャネル 0 では TCLKA がチャンネル 1 では TCLKB が使用されます。タイマコピーが動作している間は呼び出さないで下さい。

USBM_TCPYSetCmp()

```
TW_STATUS USBM_TCPYSetCmp(TW_HANDLE hDev, long CH, BYTE Cmp)
```

```
hDev : デバイスのハンドル
CH   : 設定するチャンネル(0,1)
Cmp  : コンペア値(1～255)
```

8ビットタイマがコンペア値と一致するとUSBM_TCPYSetParm()で指定した転送元から転送先アドレスに1バイトのデータがコピーされます。またタイマのカウント値はリセットされ最初からカウントされますので転送の周期をこの関数で決定します。
転送の周期は T = (Cmp + 1) / CLK となります。ただし CLK は USBM_TCPYSetClk()で指定したクロック周波数です。

USBM_TCPYSetCycle()

```
TW_STATUS USBM_TCPYSetCycle(TW_HANDLE hDev, long CH, BYTE Cmp, long CLK)
```

```
hDev : デバイスのハンドル
CH   : クロックを設定するチャンネル(0,1)
Cmp  : コンペア値(1～255)
CLK  : USBM_TCLK3125  3125kHz
       USBM_TCLK390   390.625kHz
       USBM_TCLK3     約 3052Hz
       USBM_TCLKUP    外部クロックの立ち上がり
       USBM_TCLKDOWN  外部クロックの立ち下がり
       USBM_TCLKBOTH  外部クロックの両エッジ
```

タイマコピーのコピーサイクルを設定します。8 ビットタイマのカウンタは CLK で選択されたクロックでカウントされカウンタの値がコンペア値と一致すると USBM_TCPYSetParm()で指定した転送元から転送先アドレスに 1 バイトのデータがコピーされます。またタイマのカウント値はリセットされ最初からカウントされますので転送の周期をこの関数で決定します。
転送の周期は T = (Cmp + 1) / CLK となります。外部クロックはチャネル 0 では TCLKA がチャンネル 1 では TCLKB が使用されます。タイマコピーが動作している間は呼び出さないで下さい。

USBM_TCPYSetParm()

```
TW_STATUS USBM_TCPYSetParm(TW_HANDLE hDev, long CH, DWORD SrcPort, DWORD DstPort,
                           long nCopy, long SrcInc, long DstInc, long lLoop)
```

hDev : デバイスのハンドル
CH : 設定するチャンネル(0,1)
SrcPort : 転送元アドレス
DstPort : 転送先アドレス
nCopy : コピーするバイト数(1～65535)
SrcInc : 転送毎に転送元アドレスを増加するバイト数(-128～127)
DstInc : 転送毎に転送先アドレスを増加するバイト数(-128～127)
lLoop : 真の場合転送終了後に再び最初の状態に戻って転送を再開します(TRUE,FALSE)

タイマコピーのパラメータを設定します。タイマコピーでは 8 ビットタイマがコンペア値と一致する度に転送元に指定したアドレスから転送先に指定したアドレスに 1 バイトずつデータをコピーします。タイマコピーが動作している間は呼び出さないで下さい。

USBM_TCPYSetPatternCtrl()

```
TW_STATUS USBM_TCPYSetPatternCtrl(TW_HANDLE hDev, long CH, DWORD SrcPort, DWORD DstPort,
                                  long nCopy, long nSrc, long SrcInc, long Start, BYTE Mask);
```

hDev : デバイスのハンドル
CH : 設定するチャンネル(0,1)
SrcPort : パターンデータの先頭アドレス
DstPort : 転送先アドレス
nCopy : 転送回数
  1～65535　指定回数だけコピーを行います
  -1　　停止されるまでコピーを繰り返します
nSrc : パターンデータのバイト数(1～65535)
SrcInc : 転送毎に転送元アドレスを増減させるバイト数(-128～127)
Start : 開始時の転送元を示す 0 から始まるインデックス値(0～65534)
Mask : 転送先に反映するビットを指定

タイマコピーの機能を利用して､任意のポートへのパルスパターン出力を設定します｡タイマコピーでは 8 ビットタイマがコンペア値と一致する度に転送元に指定したアドレスから転送先に指定したアドレスに 1 バイトずつデータをコピーされます。
USBM_TCPYSetParm()との違いは転送元に転送回数とは別にパターン数を指定できる点です。転送元アドレスはパターンデータの最後尾を超えると、自動的に先頭に移動します。逆に先頭アドレスより小さくなると、自動的に最後尾に移動します。また、転送先アドレスは常に固定です。
Start は転送開始時の転送元の位置を指定します。転送元として SrcPort+Start から開始されます。
Mask は転送先に反映するビットを指定します。0 のビットは書換えられません。Mask は出力専用ポート(USBM_POUT)に対しては無効です。
この関数は該当チャンネルのタイマコピーが動作している間は呼び出さないで下さい

※この関数は Ver.2.0 以降のファームウェアが必要です。

USBM_TCPYSetTrig()

TW_STATUS USBM_TCPYSetTrig(TW_HANDLE hDev, long CH, long StartPC, long StopPC,long Repeat)

hDev : デバイスのハンドル
CH : 設定するタイマコピーのチャンネル(0,1)
StartPC : 起動要因となるパルスカウンタチャンネルを指定します。
-1 パルスカウンタでは起動しません。
0～3 指定された PCx#信号の立ち下がりで起動します。
StopPC : 終了要因となるパルスカウンタチャンネルを指定します。
-1 パルスカウンタ入力では終了しません。
0～3 指定された PCx#信号の立ち下がりで終了します。
Repeat : パルスカウンタ入力を繰り返し受け付けるかのフラグ(TRUE,FALSE)

タイマコピーの起動要因、終了要因に PC0#～PC3#のパルスカウンタ入力を指定します。起動要因と終了要因は、別々のチャンネルを指定してください。
この関数を呼び出すと、指定のパルスカウンタがカウントを開始しますので、不要になった場合には USBM_PCStop()関数を呼び出してパルスカウンタを停止してください。

USBM_TCPYStart()

TW_STATUS USBM_TCPYStart(TW_HANDLE hDev, BYTE Bits)

hDev : デバイスのハンドル
Bits : ビット 0(LSB) 1 にするとチャンネル 0 をスタートします。
ビット 1 1 にするとチャンネル 1 をスタートします。

Bits で指定されたチャンネルのタイマコピーをスタートします。ストップする場合は対応するビットを 0 にして呼び出します。Ver.1.0 のファームウェアでは呼び出し時に初期化が行われましたが、Ver.2.0 以降では中断した位置から再開可能なように、初期化されない仕様に変更されました。新しく転送を開始する場合は USBM_TCPYSetParm()、または USBM_TCPYSetPatternCtrl()関数を呼び出してください。

USBM_TCPYReadStatus()

TW_STATUS USBM_TCPYReadStatus(TW_HANDLE hDev, long CH, long *pnCopy, long *pSrcPos);

hDev : デバイスのハンドル
CH : 設定するタイマコピーのチャンネル(0,1)
pnCopy : [出力]現在の転送回数の格納先
pSrcPos : [出力]現在の転送元位置を示す 0 から始まるインデックス

指定チャンネルのタイマコピーの終了した転送回数と転送元の位置を返します。pSrcPos には設定時に指定した転送元アドレスから何バイトの位置を転送中であるかが返ります。
転送中に呼び出すと、pnCopy の値と pSrcPos の値が同期していない場合があります。

□ その他の関数

USBM_Abort()

TW_STATUS USBM_Abort(TW_HANDLE hDev)

hDev : デバイスのハンドル

USBM_ADBRead(),USBM_SCIRead()関数がタイムアウトした場合に、マイコン側の処理を中止させる場合に使用します。また、USBM_ADStart()関数での AD 変換を中止する場合にも使用します。

USBM_GetFTHandle()

FT_HANDLE USBM_GetFTHandle(TW_HANDLE hDev)

hDev : デバイスのハンドル

FTDI 社の D2XX API を呼び出すためのハンドルを取得します。

USBM_GetQueueStatus()

TW_STATUS USBM_GetQueueStatus(TW_HANDLE hDev, DWORD *pnQueue)

hDev : デバイスのハンドル
pnQueue : [出力]受信バイト数の格納先

デバイスから受信したデータのバイト数を取得します。USBM_ADStart()を使用時に受信した変換結果のバイト数を取得するのに使用します。

USBM_Purge()

TW_STATUS USBM_Purge(TW_HANDLE hDev, DWORD dwMask)

hDev : デバイスのハンドル
dwMask : クリアするバッファを指定
USBM_PURGE_RX 受信バッファをクリア
USBM_PURGE_TX 送信バッファをクリア

デバイスとの通信に使用するバッファをクリアします。特にデバイスからデータを読み出す関数がタイムアウトした場合、関数から復帰した後にデバイスから送られたデータが受信バッファに溜まっている場合があるので、それらをクリアするために必要となります。
LAN デバイスでは USBM_PURGE_TX は無効です。

USBM_SetTimeouts()

```
TW_STATUS USBM_SetTimeouts(TW_HANDLE hDev, DWORD dwReadTimeout, DWORD dwWriteTimeout)
```

hDev : デバイスのハンドル
dwReadTimeout : 読み出しのタイムアウト(msec 単位)
dwWriteTimeout : 書き込みのタイムアウト(msec 単位)

デバイスとの送信及び受信の際に適用されるタイムアウト時間を設定します。デフォルトの設定では、送信、受信とも 5 秒です。

USBM_Read()

```
TW_STATUS USBM_Read(TW_HANDLE hDev, void *pData, DWORD nData, DWORD *pRead)
```

hDev : デバイスのハンドル
pData : [出力]読み出したデータの格納先へのポインタ
nData : 読み出すバイト数(0～65536)
pRead : [出力]読み出したデータのバイト数の格納先へのポインタ

デバイスから送られたデータを読み出します。指定バイト数のデータを読み出すかタイムアウトするまで関数から戻りません。

USBM_Write()

```
TW_STATUS USBM_Write(TW_HANDLE hDev, void *pData, DWORD nData, DWORD *pWritten)
```

hDev : デバイスのハンドル
pData : [入力]送信するデータへのポインタ
nData : 送信するバイト数(0～65536)
pWritten : [出力]実際に送信したバイト数の格納先

デバイスにデータを送信します。

Appendix

定数、変数型の名称変更について

以前のバージョンで「FT_～」と表記していた定数、変数の型名の一部は「TW_～」または「USBM_～」という名称に改められました。ただし、意味は同様となっていますので以前の名称でも使用可能です。

D2XX 関数の呼び出しについて

FTDI 社標準の D2XX API 関数(*FT_Purge()*など)の使用について、以前のマニュアルでは使用方法の一部としてご案内していましたが、本リファレンスに記載のライブラリバージョンでは、デバイスのハンドルに互換が無く、これらの関数を直接呼び出すことはできなくなっています。D2XX API 関数の呼び出しが必要な場合は、下記のようにハンドル変換用の関数を使用してください。

```
FT_Purge(USBM_GetFTHandle(hDev), USBM_PURGE_RX)
```

引数の初期値変更について

USBM_PortBRead() 、*USBM_PortBWrite()* 関数の *Dma* 引数の初期値が以前のバージョン(2.2.x.x 以前)では TRUE(DMA を使用する)でしたが、FALSE(DMA を使用しない)に変更になっています。引数を省略して呼び出していた場合、動作が変わってしまいますのでご注意ください。

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

改訂記録

年月	版	改訂内容
2009 年 12 月	初	