

P - P C I ソフト仕様書

99 . 08 . 30	第一版	
99 . 09 . 03	第二版	割り込み説明追加
99 . 11 . 11	第三版	複数枚対応
02 . 06 . 19	第四版	PpciStart()の戻り値変更
05 . 09 . 30	第5版	winNT / 2000 / XP のインストール方法追加

株式会社 **ファード**

目次

1 . P - P C Iボードのドライバインストール方法	3
1 - 1 . Windows 9 5 / 9 8 / Meの場合	3
1 - 2 . Windows NT 4 . 0の場合	3
1 - 3 . Windows 2 0 0 0 / X Pの場合	3
2 . 添付ソフト	4
2 - 1 . Windows 9 5 / 9 8 / Me	4
2 - 2 . Windows NT	4
2 - 3 . Windows 2 0 0 0 / X P	4
3 . 開発について	5
3 - 1 . 開発環境	5
3 - 2 . ライブラリのインストール	5
3 - 3 . ライブラリのリンク	5
3 - 4 . ドライバ転送用のメモリ	5
4 . プログラム手順	6
4 - 1 . ボードのオープン	6
4 - 2 . ボードのクローズ	6
4 - 3 . ターゲットモード	7
4 - 4 . マスタモード (マクロ関数を使用時)	8
4 - 5 . マスタモード (個別関数を使用する場合)	9
4 - 6 . 割り込みについて	13
5 . 関数一覧	15
5 - 1 . 個別制御関数	15
5 - 2 . マクロ関数 (マスタモードのみ)	15
6 . 構造体 (" p p c i a p i . h ")	16
7 . エラーコード (" p p c i a p i . h ")	16
8 . 定数	17
9 . ステータス	18

10	関数詳細 (D L L 形式で提供)		19
関数 1.1	ボードのオープン	有効モード	共通 19
関数 1.2	ドライバの終了	有効モード	共通 20
関数 1.3	ボード動作モード	有効モード	共通 21
関数 1.4	ステータスリード	有効モード	共通 22
関数 2.1	データ方向	有効モード	ターゲット 23
関数 2.2	データ入力	有効モード	ターゲット 24
関数 2.3	データ出力	有効モード	ターゲット 25
関数 2.4	L E D データ出力	有効モード	ターゲット 26
関数 2.5	ディップスイッチ入力	有効モード	ターゲット 27
関数 3.1	ボードの I / O レジスタのリード	有効モード	共通 28
関数 3.2	ボードの I / O レジスタのリード	有効モード	共通 29
関数 4.1	ドライバの転送バッファ取得	有効モード	マスク 30
関数 4.2	ドライバの転送バッファ解放	有効モード	マスク 31
関数 4.3	制御権の設定	有効モード	マスク 32
関数 4.4	データの転送方向	有効モード	マスク 33
関数 4.5	転送データバイト数	有効モード	マスク 34
関数 4.6	転送開始	有効モード	マスク 35
関数 4.7	転送強制終了	有効モード	マスク 36
関数 4.8	割り込みマスクの設定	有効モード	マスク 37
関数 4.9	割り込みクリア	有効モード	マスク 38
関数 4.10	割り込み登録	有効モード	マスク 39
関数 4.11	ドライバのバッファをリード	有効モード	マスク 40
関数 4.12	ドライバのバッファライト	有効モード	マスク 41
関数 5.1	P C I コンフィグレジスタリード	有効モード	共通 42
関数 5.2	I / O ベースアドレスリード	有効モード	共通 43
関数 5.3	メモリベースアドレスリード	有効モード	共通 44
関数 5.4	ボードレビジョンリード	有効モード	共通 45
関数 6.1	ブロック転送初期化 (マクロ関数)	有効モード	マスク 46
関数 6.2	ブロック転送入力 (マクロ)	有効モード	マスク 47
関数 6.3	ブロック転送出力	有効モード	マスク 48

1 . P - P C Iボードのドライバインストール方法

1 - 1 . W i n d o w s 9 5 / 9 8 / M eの場合

パソコンの電源を切ってP - P C IボードをP C Iバスに挿入します。

パソコンの電源を投入しW i n d o w s が起動すると、
新しいデバイスとしてP - P C Iボードが自動検出されます。

ここで付属のドライバディスクからドライバをインストールします。

ドライバをインストールしおわったらパソコンを再起動して下さい。

再起動後、[コントロールパネル][システム]のデバイスマネージャに
” M T D ” としてP - P C Iボードが表示されていればインストール
終了です。

1 - 2 . W i n d o w s N T 4 . 0の場合

添付ソフトのドライバーを H a r d D i s k のディレクトリにコピーし、
E N A B L E . B A T をマウスのダブルクリックで実行します。
(または D O S 窓で実行します)
所定のところにドライバファイルが C o p y されます。

パソコンの電源を切ってP - P C IボードをP C Iバスに挿入します。

この後、P C を再立ち上げして下さい。

立ち上がったら、コントロールパネルの「デバイス」を立ち上げ、
「P _ P C I」を探し、それを「開始」させて下さい。
「開始」できれば、ドライバはインストール完了です。

1 - 3 . W i n d o w s 2 0 0 0 / X Pの場合

パソコンの電源を切ってP - P C IボードをP C Iバスに挿入します。

立ち上げの途中で、P Cボードのドライバーのインストールを要求して
きますので、P p c i . i n fを指定してください。
後は、自動的にインストールをしてくれます。

インストールが、終わりましたら<コントロールパネル> <システム>
<ハードウェア> <デバイスマネージャ>に以下の表示があれば終了です。

```
F i r d P - P C I
└── P - P C I
```

2. 添付ソフト

2-1. Windows 95 / 98 / Me

(1) デバイスドライバー・ファイル

¥ driver p_pci.inf (インストール情報)
p_pci.vxd (ドライバ本体)

(2) テストプロ実行ファイル

¥ exe ¥ ppcisample.exe (テストプロ)
Ppcisb3.dll (関数ライブラリ)

(3) 関数ファイル (テストプロ・ソースファイル内)

Ppcisb3.dll (ライブラリ本体)
Ppcisb3.lib (コンパイル時の参照)
Ppciapi.h (関数定義)

(4) テストプロ・ソースファイル

¥ ppcisamp3 ¥ (ソース一式)
Visual C++で開発したものです。

2-2. Windows NT

(1) デバイスドライバー・ファイル

¥ Driver ¥ Ppci.ini (インストール情報)
p_pci.sys (ドライバ本体)
ENABLE.bat (インストール実行バッチファイル)
REGINI.EXE (インストール実行ファイル)

(2) テストプロ実行ファイル

¥ exe ¥ ppcisample.exe (テストプロ)
Ppcisb3ntc.dll (関数ライブラリ)

(3) 関数ファイル (テストプロ・ソースファイル内)

Ppcisb3ntc.dll (ライブラリ本体)
Ppcisb3ntc.lib (コンパイル時の参照)
Ppciapi.h (関数定義)

(4) テストプロ・ソースファイル

¥ Ppcisample ¥ (ソース一式)
Visual C++で開発したものです。

2-3. Windows 2000 / XP

(1) デバイスドライバー

¥ Driver 2k-xx ¥ Ppci.inf (インストール情報)
p_pci.sys (ドライバ本体)

(2) テストプロ実行ファイル

Windows NTのファイルを使用してください。

(3) 関数ファイル

Windows NTのファイルを使用してください。

(4) テストプロ・ソースファイル

Windows NTのファイルを使用してください。

3 . 開発について

3 - 1 . 開発環境

ウインドウズ95 / 98 / Me / NT4 / 2000 / XPで、
Visual C / C ++を使用して開発できます。

3 - 2 . ライブラリのインストール

ライブラリは、DLL形式で、添付ソフトの¥ppcisamle3¥
ppcisb3.dll (95/98/Me)、ppcisb3ntc.dll (NT/2000/XP) を

WindowsのSystemディレクトリか、
実行ファイルのあるディレクトリにコピーします。

(テストプログラム実行は、既に実行ファイルと共に同じディレクトリに
存在していますので、コピーし直す必要は、ありません。)

3 - 3 . ライブラリのリンク

メニューの [ビルド] [設定] の "リンク" インデックスを選択して、
その中のオブジェクト/ライブラリ モジュールに
" ppcisb3.lib " (95/98/Me)、" ppcisb3ntc.lib " (NT/2000/XP) を入力して
設定します。

" xxx.lib " ファイルは、Visual C/C++のプロジェクトのあるディレクトリに
コピーします。

また、定義用ファイルとして " ppciapi.h " を作成プログラムに
インクルードしてください。

両ファイルは、添付のテストプロ・ソースファイル・ディレクトリにあります。

3 - 4 . ドライバ転送用のメモリ

マスタモード時、転送データ容量分のパソコンの物理メモリ (連続したエリア) を
確保しますので、大容量のデータを転送しようとする、他のアプリケーションの
動作が遅くなったり、また確保、出来なかつたりします。

大容量のデータを扱う場合、それなりのメモリを増設してください。

4 . プログラム手順

実際に関数を使用してアプリケーションを作成する手順を説明します。

関数の説明も参照しながら読んでください。

一部変数の定義は、省略します。

また、複数枚使用時は、ハンドルを枚数分確保し、それぞれそのハンドルで実行してください。

4 - 1 . ボードのオープン

(1) 個別関数を使用する時

```
HANDLE hVxD ; //ハンドル定義 (複数枚使用時は、枚数分定義)
hVxd=PpciOpen(1); // 引数(1)は、基板 ID でディップスイッチ下位 4
// ビットに対応するが、1枚搭載の場合
// 無視する。
```

(注) ボードのオープンは、開始時に一度行えばよい。

(2) マクロ関数を使用する時

```
HANDLE hVxD; //ハンドル定義
hVxD=PpciBlockInit(1); //引数、戻り値は、PpciOpen と同じ。
```

以下の関数についてのハンドルは、この"hVxD"を使用する。

4 - 2 . ボードのクローズ

```
int ret;
ret=PpciClose(hVxD);
```

(注) アプリケーション終了時には、必ず実行する事。

4 - 3 . ターゲットモード

(共通) ターゲットモード設定 (電源立ち上げ時は、このモードになっている)

```
PpciRunMode ( hVxD,QTARGET);
```

(1) L E D の点灯 / 消灯を行う

```
PpciLEDOut(hVxD,LED_data) //LED_data に LED 点灯/消灯データを入力  
// "1":点灯、"0"で消灯
```

(2) ディップスイッチをリードする。

```
BYTE sw;  
sw=PpciDipSwitchIn(hVxD); //sw にスイッチ情報  
//"1":ON、"0":OFF
```

(3) 外部コネクタと入出力する。

データの入力 / 出力の設定 (バイト単位)

```
PpciTargetDIR(hVxD,dir1,dir2,dir3,dir4);  
//dir1 ~ 4 に入出力方向を入力  
// 入力 : QBYTEIN  
// 出力 : QBYTEOUT
```

アクセス

・入力

```
DWORD data ;  
data=PpciIOInput(hVxD); // 出力モードになっているデータは、  
// ソフトでマスクして使用する。
```

・出力

```
PpciIOOutput(hVxd,out_data); // out_data に出力データを入力
```


4 - 4 . マスタモード (マクロ関数を使用時)

(1) 外部からの入力

アプリケーションデータバッファ確保

```
char *buf;  
buf= (char*)malloc(data_size); // c の関数
```

実行

```
int ret;  
ret=PpciBlockRead(hVxD,buf,data_size,Timeout);  
//Timeout にタイマ監視 ( 1mS 単位 ) を入力
```

アプリケーション開始時は、ボードをオープン (4 - 1 参照) し、
アプリケーション終了時は、ボードをクローズ (4 - 2 参照) する。

(2) 外部へ出力

アプリケーションデータエリア確保 (データ作成)

```
char *buf;  
buf= (char*)malloc(data_size); // c の関数  
このエリアにデータを作成する。
```

実行

```
int ret;  
ret=PpciBlockWrite(hVxD,buf,data_size,Timeout);  
//Timeout にタイマ監視 ( 1mS 単位 ) を入力
```

アプリケーション開始時は、ボードをオープン (4 - 1 参照) し、
アプリケーション終了時は、ボードをクローズ (4 - 2 参照) する。

4 - 5 . マスタモード (個別関数を使用する場合)

(共通) マスタモード設定

```
PpciRunMode ( hVxD,QBUSMASTER);
```

(1) 外部から入力

所有権の有無

・制御権をとる場合

```
int ret;
```

```
ret=PpciSetPRV(hVxD,QPRVON);
```

//もし ret が QPRVOFF の場合制御権は、相手機器にあります。

・制御権をとらない場合 (制御権を放棄する)

```
int ret;
```

```
ret=PpciSetPRV(hVxD,QPRVOFF);
```

//ret は無条件に QPRVOFF になっています。

転送方向を設定

・制御権がある場合

```
PpciSetDIR(hVxD,QMSTIN);
```

・制御権がない場合

```
DWORD status;
```

```
status=PpciStatusRead(hVxD);
```

//status のビット 6 が "1" の場合入力が可能です。

ドライバのバッファを確保

```
int ret;
```

```
ret=PpciMemAlloc(hVxD , data_size ) ;
```

//data_size にバイト数を入力

//ret が NULL の時、エラーで確保できません。

アプリケーションバッファを確保

```
char *buf;
```

```
buf=(char*)malloc(data_size);
```

転送長を設定

```
PpciSetDataLENG(hVxD,data_size-1);
```

//転送長は、data_size-1 を設定

スタート

```
int ret;
```

```
ret=PpciStart(hVxD); // ret=正(0~)の時正常
```

// エラーは、ライブラリ説明を参照。

ステータスリードし終了をチェックする。

```
int status;
```

```
status=PpciStatusRead(hVxD);
```

// status のビット 31="0"の時終了。

ドライバのバッファをリードする。

```
PpciReadBuf(hVxD,buf,data_size);
```

以上で終わりですが、繰り返し同じ容量でしたら 繰り返し
容量が変わるときは、

ドライバ、アプリケーションバッファを解放 () して 繰り返し。

バッファの解放 (アプリケーション終了)

```
free(buf); //アプリケーションバッファ解放
```

```
PpciMemFree(hVxD); //ドライババッファ解放
```

(2) 外部へ出力

所有権の有無

- ・制御権をとる場合

```
int ret;
ret=PpciSetPRV(hVxD,QPRVON);
//もし ret が QPRVOFF の場合制御権は、相手機器にあります。
```

- ・制御権をとらない場合 (制御権を放棄する)

```
int ret;
ret=PpciSetPRV(hVxD,QPRVOFF);
//ret は無条件に QPRVOFF になっています。
```

転送方向を設定

- ・制御権がある場合

```
PpciSetDIR(hVxd,QMSTOUT);
```

- ・制御権がない場合

```
DWORD status;
status=PpciStatusRead(hVxD);
//status のビット 6 が "0" の場合入力が可能です。
```

ドライバのバッファを確保

```
int ret;
ret=PpciMemAlloc(hVxd , data_size ) ;
//data_size にバイト数を入力
//ret が NULL の時、エラーで確保できません。
```

アプリケーションバッファを確保

```
char *buf;
buf=(char*)malloc(data_size);
転送長を設定
PpciSetDataLENG(hVxD,data_size-1);
//転送長は、data_size-1 を設定
```

データ作成

アプリケーションバッファ (buf) にデータを作成する。

ドライバのバッファにライトする。

```
PpciWriteBuf(hVxD,buf,data_size);
```

スタート

```
int ret;
ret=PpciStart(hVxD); // ret=正(0~)の時正常
// エラーは、ライブラリ説明を参照。
```

ステータスリードし終了をチェックする。

```
int status;
status=PpciStatusRead(hVxD);
// status のビット 31="0"の時終了。
```

以上で終わりですが、繰り返し同じ容量でしたら 繰り返し
容量が変わるときは、ドライバ、アプリケーションバッファを
解放 () して 繰り返し。

バッファの解放 (アプリケーション終了)

```
free(buf); //アプリケーションバッファ解放  
PpciMemFree(hVxD); //ドライババッファ解放
```

4 - 6 . 割り込みについて

割り込みは、直接割り込みが入るのではなく、割り込みが入るとユーザーアプリケーションに対しメッセージを送ります。ユーザーは、そのメッセージを処理します。

割り込みの登録

```
PpciCallbackWND(hVxD,(DWORD)this->m_hWnd);  
// this->m_hWnd でそのクラスのウインドハンドルが得られる。
```

メッセージ処理

```
LRESULT CPpcisampleView::WindowProc  
(UINT message, WPARAM wParam, LPARAM lParam)  
{  
  
if(message==WM_USER_INT){ // 割り込みメッセージ番号  
  
if((DWORD)lParam==(DWORD)hVxD){ //ボード複数枚の識別で  
//オープン時のハンドルを使用する。  
  
CString cs=" ";  
if((wParam&0x1)!=0){//正常終了  
cs+="正常終了、";  
}  
if((wParam&0x2)!=0){//強制終了  
cs+="強制終了、";  
}  
if((wParam&0x4)!=0){//開始要求  
cs+="開始要求";  
}  
cs+="の割り込みが入りました";  
MessageBox(cs);  
}  
  
if(lParam==(DWORD)hVxD1){ //次のボードをチェックする。  
上記同様ステータスチェックをする。  
}  
return CFormView::WindowProc(message, wParam, lParam);  
}
```

上記は、添付テストプログラムより引用しました。
実際には、classWizardで作成するクラス、オブジェクト ID を選択し
(この場合 CPcismpleView)、メッセージを WindowProc を選択し、
ダブルクリックにより、上記下線の関数が生成されます。
[コード編集]でその関数位置にジャンプし、それ以外のコードを作成します。

message をチェックし、WM_USER_INT でしたら、割り込みが入っていて、
wParam に割り込みステータスが、格納されています。
また、複数枚対応として、lParam にボードオープン時に獲得したハンドルが格納
されていますので、それをチェックしどのボードから割り込みが入ったかを判断
します。
割り込みステータスは、マスクが解除されているもののみ入っていて、マスクが
されているビットについては、関知しません。
しかし、割り込み処理(ドライバ)で、割り込みステータスは、クリアされます
ので、マスクされている割り込みステータスは、破棄されます。

割り込みメッセージ番号は、定義 (WM_USER_INT) していますが、
別定義にする場合の値は、" WM_USER+1 " にしてください。

5 . 関数一覧

5 - 1 . 個別制御関数

No.	モード	関数名	概略内容
1	共通	PpciOpen PpciClose PpciSetRunMode PpciStatusRead	ドライバのオープン " クローズ 転送方式 ステータスリード
2	ポート入出力	PpciTargetDIR PpciIOInput PpciIOOutput PpciLEDOut PpciDipSwitchIn	入出力方向 データ入力 データ出力 LED出力 ディップスイッチ・リード
3	I/O 入出力	PpciIORead PpciIOWrite	I/Oリード I/Oライト
4	マスタ転送	PpciMemAlloc PpciMemFree PpciSetPRV PpciMasterDIR PpciSetDataLENG PpciStart PpciStop PpciINTMask PpciINTClear PpciCallbackWND PpciReadBuf PpciWriteBuf	ドライババッファの取得 " 解放 制御権の設定 転送方向 転送バイト数 転送スタート 転送強制終了 割り込みマスク 割り込みクリア 割り込み登録 ドライババッファのリード " ライト
5	PCI コンフィグレジスタ	PpciConfigRead PpciGetIOAddress PpciGetMEMAddress PpciGetRevID	全てのコンフィグレジ・リード I/Oベースアドレス・リード メモリ " バージョン ID リード

5 - 2 . マクロ関数 (マスタモードのみ)

No.	モード	関数名	概略内容
1	マスタ転送	PpciBlockInit PpciBlockRead PpciBlockWrite	ブロックモード初期化 ブロックリード ブロックライト

(注) アプリケーション終了時は、個別関数の PpciClose を使用してください。

6. 構造体 (" p p c i a p i . h ")

・ P C I コンフィグレジスタ情報用構造体

```
struct    PPCI_CONFIG_DATA{
    WORD    DeviceID;
    WORD    BenderID;
    WORD    Status;
    WORD    Command;
    DWORD   ClassCode;
    BYTE    RevisionID;
    BYTE    Bist;
    BYTE    HeaderType;
    BYTE    LatencyTimer;
    BYTE    CacheLineSize;
    DWORD   BaseAddress1;           // I/O
    DWORD   BaseAddress2;         // メモリ
    BYTE    MaxLatency;
    BYTE    MinGrant;
    BYTE    IntPin;
    BYTE    IntLine;
};
```

PpciConfigRead 関数で使用する。

7. エラーコード (" p p c i a p i . h ")

```
#define QERROR_END    0xffffffff
#define QNORMAL_END   0x0
#define QSTOP_END     0x1
#define QERROR_CNN    0x2
#define QERROR_BUF    0x4
#define QTIME_OVER    0x8
```

8 . 定数

```
#define WM_USER_INT WM_USER+1 //割り込み時、アプリケーションに渡す
                                //メッセージ
#define QTARGET      0          // ターゲットモード
#define QBUSMASTER  1          // マスターモード
#define QBYTEIN     0          // ターゲット時のデータ入力方向
#define QBYTEOUT    1          //          "          出力方向
#define QMSTOUT     1          // マスタ時のデータ出力方向
#define QMSTIN      0          //          "          入力方向
//
#define QPRVON      0          // 制御権獲得
#define QPRVOFF    1          // 制御権解放

#define QSTARTMASK  0x0004    // 開始割り込みマスク
#define QSTOPMASK   0x0002    // 強制終了 "
#define QENDMASK    0x0001    // 正常終了 "
```

9. ステータス

PpciStatusRead関数で得られるデータは、H/Wステータスです。

ビット

3 1 マスタビジー "0": 停止 (終了)、"1": 実行中

PpciStart関数でスタート後、"0"になると
転送終了。

3 0

~ 1 9 空き

1 8 開始要求割り込みステータス "1": 開始要求割り込みあり

1 7 強制終了 " " "1": 強制終了 "

1 6 正常終了 " " "1": 正常終了 "

(注) 割り込みステータスは、割り込みマスクをしてもセットされる。

8 動作モード "0": ターゲットモード、"1": マスタモード

7 制御権 "0": 制御権あり、"1": 制御権なし

6 マスタモード時の転送方向

・制御権ありの時 "0": 入力、"1": 出力

・制御権なしの時 "0": 出力、"1": 入力

5 ドータボードの有無 "0": あり、"1": なし

4 空き

3 REQUEST状態 "1": アクティブ

2 READY状態 "

1 ENABLE状態 "

0 VALID状態 "

ビット0 ~ 3は、マスタモード時ブロック転送コントロール信号である。

10. 関数詳細 (DLL形式で提供)

関数 1.1	ボードのオープン	有効モード	共通
プロトタイプ	HANDLE PpciOpen (BYTE p1)		
引数	<p>BYTE p1:ディップスイッチ下位4ビット (Sw-No 4 ~ 1)</p> <p>ボードが複数存在するときに識別用として、ディップスイッチを使用する。 1枚のみ使用時は、無条件に選択する。</p> <p>DIPSW "1": ON "0": OFF</p>		
戻り値	<p>HANDLE : オープンしたボードのハンドル番号 この HANDLE で、以後の関数で使用する。</p>		
内容	<p>ドライバをオープンし、使用できるようにする。</p>		
備考			

関数 1.2	ドライバの終了	有効モード	共通
プロトタイプ	int PpciClose(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	int : エラーステータス 0 : 正常終了 - 1 : オープンされていないのにクローズをした。		
内容	現在使用しているボードのハンドラをクローズする。(クローズ)		
備考			

関数 1.3	ボード動作モード	有効モード	共通
プロトタイプ	void PpciSetRunMode (HANDLE ph, int p1)		
引数	<p>HANDLE ph : ボードオープン時の戻り値</p> <p>int p1 : 基板動作モード (基板動作モード)</p> <p> QTARGET (ターゲットモード : パラレル I / O)</p> <p> QBUSMASTER (マスタモード : ブロック転送)</p>		
戻り値	なし		
内容	<p>ボードの動作モードを設定する。</p> <p>ターゲットモードは、単発アクセスで 32 ビットパラレルアクセスを行うモードです。</p> <p>バスマスタモードは、コントロール線を使用して、32 ビットバーストアクセスを行う。</p> <p>その時、ボードはパソコンのメモリに対しバスマスタアクセスを行う。</p>		
備考	転送方式に関しては、H / W 説明書を参照してください。		

関数 1.4	ステータスリード	有効モード	共通
プロトタイプ	DWORD PpciStatusRead(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	DWORD : ステータス 32 ビット ステータス内容は、" 9 . ステータス " を参照してください。		
内容	ボードの実行状態、設定状態をリードする。		
備考	ビットの内容に対しては、ステータス説明を参照してください。		

関数 2.1	データ方向	有効モード	ターゲット
プロトタイプ	void PpciTargetDIR (HANDLE ph,int p1,int p2,int p3,int p4)		
引数	<p>HANDLE ph : ボードオープン時の戻り値</p> <p>int p1 : 第 1 バイト目の方向 (最下位バイト)</p> <p>int p2 : 第 2 バイト目の方向</p> <p>int p3 : 第 3 バイト目の方向</p> <p>int p4 : 第 4 バイト目の方向 (最上位バイト)</p> <p>(設定値)</p> <p>QBYTEIN : 入力モード</p> <p>QBYTEOUT : 出力モード</p>		
戻り値	なし		
内容	ターゲット (パラレル I / O) モード時の各データバイトの方向を設定する。		
備考			

関数 2.2	データ入力	有効モード	ターゲット
プロトタイプ	DWORD PpciIOInput (HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	DWORD : 32ビットデータ入力。		
内容	<p>ターゲットモード時のデータ入力。 (注) データを入力にしているバイトのみ有効。 出力にしているバイトに対しては、出力データが入力される。</p>		
備考			

関数 2.3	データ出力	有効モード	ターゲット
プロトタイプ	void PpciIOOutput (HANDLE ph,DWORD p1)		
引数	HANDLE ph : ボードオープン時の戻り値 DWORD p1 : 出力データ 32 ビット。		
戻り値	なし		
内容	ターゲット (P I / O) モード時のデータ出力をする。 入力モードに設定されているバイトのデータは、任意である。		
備考			

関数 2.4	L E Dデータ出力	有効モード	ターゲット
プロトタイプ	void PpciLEDOut(HANDLE ph, BYTE p1)		
引数	HANDLE ph : ボードオープン時の戻り値 BYTE p1 : L E Dデータ8ビット		
戻り値	なし		
内容	L E Dを点灯 / 消灯する。 "1"で点灯。 "0"で消灯。		
備考	ターゲット (P I / O) モード時のみ有効で、バスマスタ (ブロック転送) 時には、無効である。		

関数 2.5	ディップスイッチ入力	有効モード	ターゲット
プロトタイプ	BYTE PpciDipSwitchIn(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	BYTE : ディップスイッチデータ 8 ビット		
内容	ディップスイッチをリードする。 スイッチ ON で "1"、OFF で "0"。		
備考	ターゲット (P I / O) モード時のみ有効で、バスマスタ (ブロック転送) 時には、無効である。 (注) 電源立ち上げ時、下位 4 ビットをボード識別番号として 使用する。 それ以降では、使用は任意である。		

関数 3.1	ボードのI/Oレジスタのリード	有効モード	共通
プロトタイプ	DWORD PpciIORead(HANDLE ph, DWORD p1)		
引数	HANDLE ph : ボードオープン時の戻り値 DWORD p1 : I/Oアドレスのオフセット (+0 ~)		
戻り値	DWORD 該当オフセットのレジスタデータ 32ビット		
内容	ボードのレジスタをリードする。 ないレジスタをリードしたとき、データは保証されない。		
備考	レジスタのオフセットについては、ハード仕様書を参照してください。		

関数 3.2	ボードのI/Oレジスタのリード	有効モード	共通
プロトタイプ	void PpciIOWrite (HANDLE ph,DOWRD p1,DWORD p2)		
引数	<p>HANDLE ph : ボードオープン時の戻り値</p> <p>DWORD p1 : I/Oアドレスのオフセット (+0 ~)</p> <p>DWORD p2 : ライトデータ</p>		
戻り値	なし		
内容	<p>データをレジスタにライトする。 ないレジスタのオフセットにライトしないでください、予期しない動作が、おきることがあります。</p>		
備考	レジスタのオフセットについては、ハード仕様書を参照してください。		

関数 4.1	ドライバの転送バッファ取得	有効モード	マスタ
プロトタイプ	int PpciMemAlloc(HANDLE ph, DWORD p1)		
引数	<p>HANDLE ph : ボードオープン時の戻り値 DWORD p1 : 取得するバイト数</p> <p>バイト数は、4の倍数で最大256Mバイト設定できるが、パソコンのメモリ容量により、取得出来ないことがある。 (バイト数が4の倍数でないとき、端数は切り捨てられる。)</p>		
戻り値	<p>int : 物理アドレス NULL の時取得失敗。</p>		
内容	<p>ドライバが、バスマスタ転送用に使用する物理メモリを取得する。</p>		
備考			

関数 4.2	ドライバの転送バッファ解放	有効モード	マスク
プロトタイプ	void PpciMemFree(HANDLE p1)		
引数	HANDLE p1 : 初期化時の戻り値		
戻り値	なし		
内容	関数 PpciMemAlloc で取得したドライバのバッファを解放する。		
備考			

関数 4.3	制御権の設定	有効モード	マスク
プロトタイプ	int PpciSetPRV(HANDLE ph,int p1,int p2)		
引数	<p>HANDLE ph : ボードオープン時の戻り値</p> <p>int p1 : 制御権値</p> <p>int p2 : リトライ回数 (制御権値)</p> <p>QPRVON : 制御権あり</p> <p>QPRVOFF : 制御権なし</p> <p>(リトライ回数)</p> <p>QMASTER で制御権をとれなかった場合のリトライ回数。</p>		
戻り値	<p>int : QPRVON/QPRVOFF</p> <p>QPRVON で制御権をとれなかった場合、QPRVOFF を戻す。</p>		
内容	<p>制御権を取得/解放する。</p> <p>制御権とは、転送方向を決めることができる権利である。</p>		
備考			

関数 4.4	データの転送方向	有効モード	マスク
プロトタイプ	void PpciMasterDIR(HANDLE ph,int p1)		
引数	<p>HANDLE ph : ボードオープン時の戻り値 int p1 : 転送方向</p> <p>(転送方向) QMSTIN : 入力 QMSTOUT : 出力</p>		
戻り値	なし		
内容	<p>ブロック転送の方向を設定する。 ただし、制御権を取得 (QPRVON) していなければならない。 (QPRVOFF の場合、この関数は無効である。)</p>		
備考			

関数 4.5	転送データバイト数	有効モード	マスク
プロトタイプ	void PpciSetDataLENG(HANDLE ph,DWORD p1)		
引数	<p>HANDLE ph : ボードオープン時の戻り値 DWORD p1 : 転送データ数 (バイト数) の設定。 (転送データ数) 設定範囲は 4 ~ 256M バイト。 設定は 4 の倍数で、端数は切り捨てられる</p>		
戻り値	なし		
内容	<p>転送データバイト数で、PpciMemAlloc で取得したバイト数以下にする。</p>		
備考			

関数 4.6	転送開始	有効モード	マスク
プロトタイプ	int PpciStart(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	int : -1=転送長をセットしていない。 -2=転送バッファを確保していない。 -3=転送バッファより転送長の方が長い。 0=正常終了		
内容	転送を開始する。 しかし、対抗機器が、スタートしていない時は、開始待ち状態になる。		
備考			

関数 4.7	転送強制終了	有効モード	マスタ
プロトタイプ	void PpciStop(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	なし		
内容	PpciStart でスタート後、この関数を実行すると終了する。 割り込み（強制終了、正常終了）は、起こらない。		
備考			

関数 4.8	割り込みマスクの設定	有効モード	マスク
プロトタイプ	void PpciINTMask(HANDLE ph, BYTE p1)		
引数	<p>HANDLE ph : ボードオープン時の戻り値 BYTE p1 : 割り込みマスクデータ (マスクデータ)</p> <p>QENDMASK : 正常終了マスク QSTOPMASK : 強制終了マスク QSTARTMASK : 開始マスク</p> <p>マスクをセット(割り込み禁止)するときは、上記値を またクリア(割り込み許可)する時は、上記値を入れない。</p>		
戻り値	なし		
内容	<p>割り込みのマスク(禁止)する。値がないときは、マスクを クリア(許可)する。 割り込みのマスクをしても、ステータスには、影響を与えない。</p>		
備考			

関数 4.9	割り込みクリア	有効モード	マスク
プロトタイプ	void PpciINTClear(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	なし		
内容	<p>割り込みステータスをクリアする。</p> <p>割り込みを使用している場合、割り込みが発生した時点でドライバにより自動的にクリアされる。</p> <p>割り込み要因は、メッセージにより渡される。</p>		
備考			

関数 4.10	割り込み登録	有効モード	マスク
プロトタイプ	void PpciCallbackWND(HANDLE ph,DWORD p1)		
引数	HANDLE ph : ボードオープン時の戻り値 DWORD p1 : アプリケーションウインドハンドル ウインドハンドルは、(DWORD) this->m_hWnd で得られる。		
戻り値	なし		
内容	割り込み時、メッセージを送るウインドを登録する。		
備考			

関数 4.11	ドライバのバッファをリード	有効モード	マスタ
プロトタイプ	void PpciReadBuf(HANDLE ph,void *p1,DWORD p2)		
引数	HANDLE ph : ボードオープン時の戻り値 void *p1 : アプリケーションバッファポインター DWORD p2 : リードバイト数 (4 の倍数)		
戻り値	なし		
内容	ドライバのバッファからアプリケーションバッファにデータをリードする。		
備考			

関数 4.12	ドライバのバッファライト	有効モード	マスタ
プロトタイプ	void PpciWriteBuf(HANDLE ph,void *p1,DWORD p2)		
引数	HANDLE ph : ボードオープン時の戻り値 void *p1 : アプリケーションバッファポインター DWORD p2 : ライトバイト数 (4 の倍数)		
戻り値	なし		
内容	アプリケーションバッファからドライバのバッファにライトする。		
備考			

関数 5.1	P C I コンフィグレジスタリード	有効モード	共通
プロトタイプ	void PpciConfigRead(HANDLE ph, struct PPCI_CONFIG_DATA *p1)		
引数	HANDLE ph : ボードオープン時の戻り値 PPCI_CONFIG_DATA *p1 : P C I コンフィグレジスタ構造体の ポインター 構造体の内容は、構造体の項を参照。		
戻り値	なし		
内容	現在オープンされているボードの P C I コンフィグレジスタを全て リードし、PPCI_CONFIG_DATA 構造体内に格納する。		
備考			

関数 5.2	I / Oベースアドレスリード	有効モード	共通
プロトタイプ	DWORD PpciGetIOAddress(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	<p>DWORD : I / Oアドレス値32ビット 有効アドレスは、下位16ビットである。 (INTEL系の32ビットCPUのI / Oアドレスが、 16ビットの為)</p>		
内容	<p>ボードのI / Oを直接アクセスするためのアドレスで、このアドレスにオフセットを加算し、C言語または、アセンブラ言語のI / O命令で実行する。</p> <p>(注)直接 I/O 命令でアクセスできるのはWindows95/98で、 WindowsNT/2000/XP の場合は、できません。</p> <p>I/O のアクセスは、できるならIopciIORead、IopciIOWrite の関数を使用してください。</p>		
備考	このベースアドレスは、PpciConfigRead を実行し、PPCI_CONFIG_DATA 構造体からも取得できる。		

関数 5.3	メモリベースアドレスリード	有効モード	共通
プロトタイプ	DWORD PpciGetMEMAddress(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	DWORD : メモリベースアドレス値 32 ビット		
内容	メモリベース物理アドレスで、参照用です。 使用できません。		
備考			

関数 5.4	ボードレビジョンリード	有効モード	共通
プロトタイプ	BYTE PpciGetRevID(HANDLE ph)		
引数	HANDLE ph : ボードオープン時の戻り値		
戻り値	BYTE : オープンされているボードのレビジョン番号		
内容	ボードのレビジョン番号リードで、管理用である。		
備考	PpciConfigRead でも得られる。		

関数 6.1	ブロック転送初期化 (マクロ関数)	有効モード	マスク
プロトタイプ	HANDLE PpciBlockInit(BYTE p1)		
引数	<p>BYTE p1:ディップスイッチ下位 4 ビット (Sw-No 4 ~ 1)</p> <p>ボードが複数存在するときに識別用として、ディップスイッチを使用する。 1 枚のみ使用時は、無条件に選択する。</p> <p style="text-align: center;">DIPSW "1" : O N "0" : O F F</p>		
戻り値	HANDLE : オープンしたボードのハンドル番号		
内容	PpciOpen と PpciRUNMode、PpciSetPRV 関数で構成され、マクロ関数 PpciBlockRead、PpciBlockWrite のみ使用時は、初期化として実行する。		
備考	アプリケーション開始時に、初期化として実行する。		

関数 6.2	ブロック転送入力 (マクロ)	有効モード	マスク
プロトタイプ	int PpciBlockRead(HANDLE ph,void *p1,DWORD p2,DWORD p3)		
引数	<p>HANDLE ph : ボードオープン時の戻り値 void *p1 : データ入力バッファポインター DWORD p2 : 入力データバイト数 DWORD p3 : タイムオーバー 入力バッファポインター : アプリ側の入力格納バッファのポインター 入力データバイト数 : 入力データバイト数で、4 の倍数設定。 4 ~ 2 5 6 M バイト設定出来る。 タイムオーバー : 1 m S 単位で、スタート (待ちも含む) からの 時間管理タイマ。 "0" 設定で、時間管理を行わない。</p>		
戻り値	<p>int : エラー情報 QERROR_END : 制御権がなくて、データ転送方向が逆である。 QERROR_BUF : ドライバのバッファが確保出来ない。 QTIME_OVER : タイムオーバー</p>		
内容	<p>ブロック転送入力で、マクロ関数である。 内部で実行している関数は、 転送方向を設定 (制御権が取得している時) 。 PpciMasterDIR ドライバのバッファを獲得。 PpciMemAlloc 転送長セット。 PpciSetDataLENG スタート。 PpciStart タイムアウトチェック及び ステータスで転送終了待ち ドライバのバッファ解放。 PpciMemFree</p>		
備考	<p>制御権が、取得出来なかった場合 (PRVOFF) で、転送方向が、 出力になっていた場合、エラーになり実行されない。</p>		

関数 6.3	ブロック転送出力	有効モード	マスク
プロトタイプ	int PpciBlockWrite(HANDLE ph,void *p1,DWORD p2,int p3)		
引数	<p>HANDLE ph : ボードオープン時の戻り値</p> <p>DWORD *p1 : データ出力バッファポインター</p> <p>DWORD p2 : 出力データバイト数</p> <p>DWORD p3 : タイムオーバー</p> <p>出力バッファポインター : アプリ側の出力格納バッファのポインター</p> <p>出力データバイト数 : 入力データバイト数で、4 の倍数設定。 8 ~ 256 Mバイト設定出来る。</p> <p>タイムオーバー : 1 mS 単位で、スタート (待ちも含む) からの 時間管理タイマ。 "0"設定で、時間管理を行わない。</p>		
戻り値	<p>int : エラー情報</p> <p>QERROR_END : 制御権がなくて、データ転送方向が逆である。</p> <p>QERROR_BUF : ドライバのバッファが確保出来ない。</p> <p>QTIME_OVER : タイムオーバー</p>		
内容	<p>ブロック転送出力で、マクロ関数である。</p> <p>内部で実行している関数は、</p> <p>転送方向を設定 (制御権が取得している時)。PpciMasterDIR ドライバのバッファを獲得。PpciMemAlloc 転送長セット。PpciSetDataLENG スタート。PpciStart タイムアウトチェック及び ステータスで転送終了待ち ドライバのバッファ解放。PpciMemFree</p>		
備考	<p>制御権が、取得出来なかった場合 (PRVOFF) で、転送方向が、 入力になっていた場合エラーになり、実行されない。</p> <p>また、この関数は終了ステータス待ちをしているので、終了かエラーに ならないと抜けて来ない。</p>		