高速、大容量入力ボード（LVDS インターフェース）

ＭＳ-ＰＣＩ(下位 DLL)ソフト仕様書

第２版

株式会社 ファード

〒183-0006 府中市緑町３-８-２新東邦ビル４Ｆ
TEL 042-362-5072　FAX 042-369-8026
fird@coral.ocn.ne.jp　www2.ocn.ne.jp/~fird

【改版履歴】

年月日	改　　版　　内　　容
2002．7/22	初 版
2005．9/30	第２版　Windows2000/XP のインストール説明追加

目　　次

１．ＭＳ－ＰＣＩボードのドライバインストール方法

１－１．WindowsNT の場合

（１）Windown NT(Ver 4.0)で、上のドライバ部分を HardDisk のディレクトリにし、
ENABLE.BAT をマウスのダブルクリックで実行します。（または DOS 窓で実行します）。
所定のところにドライバファイルがコピーされます。
（２）この後、PC を再立ち上げして下さい。
（３）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、「MS_PCI」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。

１－２，Windows2000/XP の場合

（１）パソコンの電源を切って、本ボードをＰＣＩバスに挿入します。

（２） 立ち上げの途中で、ＰＣ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、MSpci. inf 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。

（３）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＞→
＜ハードウエア＞→＜デバイスマネージャ＞に以下の表示があれば終了です。

Ｆｉｒｄ ＭＳ－ＰＣＩ
└ＭＳ－ＰＣＩ

２．添付ソフト

２－１．ドライバ

（１）WindowsNT

￥NT_Driver￥　MSPCI.SYS　（ドライバー本体）
MSPCI.INI　（インストール情報）
ENABLE.BAT（インストール実行バッチ）
REGINI.EXE　（Windows 側のレジストリ登録用）

（２）Windows2000/XP

￥2k_Driver￥　ms_pci.sys　（ドライバー本体）
MSpci.inf　（インストール情報）

２－２．関数（DLL で提供し、WindowsNT/2000/XP は、同じファイルを使用します。）

（１）下位 DLL

MSPCIDLL.LIB　（コンパイル時の参照ファイル）
MSPCIDLL.DLL　（関数本体）
MSPCIDLL.H　　（関数定義）

（２）上位 DLL

MSPCIAPI.LIB　（コンパイル時の参照ファイル）
MSPCIAPI.DLL　（関数本体）
MSPCIAPI.H　　（関数定義）

・以上のファイルは、サンプルプログラムのソースファイル（￥MsTestSamle¥）に含まれています

２－３．サンプルプログラム

¥MsTestSample¥　MSPCITEST.EXE　（実行ファイル）
およびその project file 一式

３．開発について

３－１．開発環境

WindowsNT/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￥MsTestSampl￥にある
MspciDll.Dll と MspciApi.Dll 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。

３－３．ライブラリィのリンク

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／ライブラリィ　モジュールに
“MspciDll.lib MspciApi.lib ”を入力して設定します。
“MspciDll.lib ”と“MspciApi.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。
ファイルは、添付の￥MsTestSampl￥ディレクトリにあります。

また、定義用ファイルとして”MspciDll.h ”と”MspciApi.h ”を作成プログラム
に include してください。
ファイルは、添付の￥MsTestSample￥ ディレクトリィにあります。

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と、他のアプリケーションの
動作が遅くなったり、また確保、出来なかったりします。
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。

４－１．ボードのオープン

```
HANDLE   hVxD；          //ハンドル定義
hVxd=MSpciOpen( 0 );  // 引数(0)は、基板番号で０～になります。
                      // １枚搭載の場合は、無条件に０となります。
```

（注）ボードのオープンは、開始時に一度行えばよい。

４－２．ボードのクローズ

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

（注）アプリケーション終了時には、必ず実行する事。

４－３．ＰＣＩ　Ｉ／Ｏアクセス

ＰＣＩのＩ／Ｏは、ＰＣＩ－ＦＰＧＡに割り当てられていて、マスタ、
ターゲット両モード状態でアクセスできます。

（１）ライト

```
MspciIoWrite( hVxD,offset,data ); //　３２ビットアクセスのみ。
```

（２）リード

```
data=MspciIoRead( hVxD,offset,data ); //３２ビットアクセスのみ。
```

ＰＣＩの各レジスタに関しては、個別関数を用意していますので、
そちらをお使いください。

４－４．ターゲットモード

（１）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）
このモードは、ＳＤＲＡＭ ＦＰＧＡへのメモリレジスタにアクセスするモードです。
ＰＣＩのメモリエリアは、ＳＤＲＡＭ ＦＰＧＡに割り当てられています。

```
MspciSetRunMode(hVxD,QTARGMODE);
```

（２）メモリ（レジスタ）ライト

```
MspciMemWrite( hVxD,offset,data ); //　３２ビットアクセスのみ
```

（３）メモリ（レジスタ）リード

```
data=MspciMemRead( hVxD,offset ); //　３２ビットアクセスのみ
```

４－５．マスタモード

（共通１）ＳＤＲＡＭ ＦＰＧＡ側のマスタースタートを実行し、待ち状態にする。
ＳＤＲＡＭ ＦＰＧＡ側のスタート時の操作は、その設計によるので
ここでは、省略します。

（共通２）マスタモード設定

```
MspciSetRunMode（ hVxD,QMSTMODE );
```

（１）マスタリード（ ＳＤＲＡＭ ＦＰＧＡ→ＰＣＩ－ＦＰＧＡ）

① 転送方向を設定

```
MSpciMasterDIR( hVxd,QDIRIN );
```

②ドライバのバッファを確保

```
int   ret;
ret=MSpciMemAlloc( 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。
```

③アプリケーションバッファを確保

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

④ 転送長を設定

```
MSpciDataLENG( hVxD,data_size );
        //転送長は、data_size を設定
```

⑤スタート

```
int   ret;
ret=MSpciStart( hVxD );   //   ret=正の値の時正常
                          //   エラーは、ライブラリィ説明を参照。
```

⑥ステータスリードし終了をチェックする。

```
BYTE   status;
status=MSpciStatusRead( hVxD );
// status のビット 7="0"の時終了。
// 割り込みでも終了を検知する事ができます。
// 割り込みについては、割り込みの説明の項を参照のこと。
```

⑦ドライバのバッファをリードする。

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

⑧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
MSpciMemFree( hVxD )；     //ドライババッファ解放
```

（2）マスタライト（ＰＣＩ－ＦＰＧＡ→ＳＤＲＡＭ側 ＦＰＧＡ）

① 転送方向を設定

```
MSpciMasterDIR( hVxd,QDIROUT );
```

②ドライバのバッファを確保

```
int   ret;
ret=MSpciMemAlloc( 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。
```

③アプリケーションバッファを確保

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

④転送長を設定

```
MSpciDataLENG( hVxD,data_size );
     //転送長は、data_size を設定
```

⑤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。

⑥ドライバのバッファにライトする。

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

⑦スタート

```
int   ret;
ret=MSpciStart( hVxD );   //   ret=正の値の時正常
                          //   エラーは、ライブラリィ説明を参照。
```

⑧ステータスリードし終了をチェックする。

```
BYTE   status;
status=MSpciStatusRead( hVxD );
// status のビット 7="0"の時終了。
// 割り込みにて、終了が検知できます。
```

⑨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
MSpciMemFree( hVxD )；     //ドライババッファ解放
```

５．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。

①割り込みの登録

```
MSpciINTEntry( 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。
```

②メッセージ処理

```
LRESULT CMSpcisampView::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
                                      //ｵｰﾌﾟﾝ時のハンドルを使用する。
      if(message==WM_USER_INT){
        if((DWORD)lParam==(DWORD)myhnd){ //複数ボード使用時
          if(wParam & 0x0001){              //End
            m_List.AddString("-----> Transfer Interrupt occured !");
          }
          if(wParam & 0x0002){//INT 1
            m_List.AddString("-----> INT 1 Interrupt occured !");
          }
          if(wParam & 0x0004){//INT 2
            m_List.AddString("-----> INT 2 Interrupt occured !");
          }
          m_List.SetCurSel( m_List.GetCount()-1) ;//最終行へ
        }
      }
    }
    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);
}
```

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 MSpcisampView)、メッセージの WindowProc を選択し、ダブルクリックに
より、上記青色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。
割り込み処理（ドライバ）で、割り込みステータスは、マスクがかかっていない
ビットについてはクリアされます。

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。

定義定数

定義定数については、“ MSpciDll.h **”のファイルに入っています。**

６．構造体

６－１．ＰＣＩ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;
};
```

MSpciConfigRead **関数で使用する。**

６－２．エラーコード

```
//スタート時エラーステータス
#define   ERROR_START1   -1   //  マスタモードになっていない。
#define   ERROR_START2   -2   //  転送バイト数を設定していない。
#define   ERROR_START3   -3   //  転送バッファを取得あいていない。
#define   ERROR_START4   -4   //  転送バッファより転送バイト数が大きい
```

６－３．Ｉ／Ｏアドレスオフセット

```
// I/O address offset
 #define   QMODEIO      0x00   // 動作モードレジスタ
 #define   QMSTADRIO    0x04   //  ＰＣＩ転送アドレスレジスタ
 #define   QMSTLENIO    0x08   //  ＰＣＩ転送バイト数レジスタ
 #define   QMSTCTLIO    0x0C   //  マスタコントロールレジスタ
 #define   QIODAIO      0x10   //  汎用Ｉ／Ｏレジスタ
 #define   QINTSETIO    0x14   //  割り込みマスクレジスタ
 #define   QINTSTSIO    0x18   //  割り込みステータスレジスタ
 #define   QMSTSTSIO    0x1C   //   マスタ転送ステータスレジスタ
```

６－４．動作モード用定数

```
//データ方向選択（MspciIoDADIR、MSpciMasterDIR で使用）
  #define QDIRIN     0  // 入力
  #define QDIROUT   1  // 出力
//マスタ／ターゲットアクセス選択（MSpciAccessCnt で使用）
  #define QTRGMODE   0  //  ターゲットモードアクセス
  #define QMSTMODE   1  //  マスタモードアクセス
```

６－５．割り込み関連コード

（１）割り込みマスク

```
#define QENDMASK   0x10 //  終了割り込みマスク
#define QINT1MASK   0x20 //   INT1       〃
#define QINT2MASK   0x40 //   INT2       〃
```

（２）割り込みステータスクリアビット

```
#define QENDCLR    0x1 //  終了割り込みステータスクリア
#define QINT1CLR    0x2 //   INT1              〃
#define QINT2CLR    0x4 //   INT2              〃
```

（３）割り込みステータスビット

```
#define QENDSTS    0x1 //  終了割り込みステータス
#define QINT1STS    0x2 //   INT1             〃
#define QINT2STS    0x4 //   INT2             〃
```

６－６．ステータス

MSpciStatusRead 関数で得られるデータは、Ｈ／Ｗステータスです。

ビット

７　マスタビジー　　"0"：停止（終了）、"1"：実行中
　　MSpciStart 関数でスタート後、"0"になると
　　転送終了。

６　空き（＝０）

５　ＡＣＫ　　：ＲＥＱに対する応答（PCI-FPGA が出力）

４　ＲＥＱ　　：転送要求　　（SDRAM FPGA　〃　）

３　ＥＮＢ＿Ｗ　：マスタライト時のイネーブル（SDRAM FPGA が出力）

２　ＶＬＤ＿Ｗ　：　　〃　　データ有効（PCI-FPGA　〃　）

１　ＥＮＢ＿Ｒ　：マスタリード時のイネーブル（PCI-FPGA　〃　）

０　ＶＬＤ＿Ｒ　：　　〃　　データ有効（SDRAM FPGA　〃　）

ビット０～５は、“１”でアクティブであり、
マスタモード時ブロック転送コントロール信号である。

7．関数一覧

“MspciDll.dll”でＤＬＬ形式で供給

No.	関数名	概略内容
1	MSpciOpen	ボードのオープン（使用開始）
2	MSpciClose	ボードのクローズ（使用終了）
3	MSpciINTEntry	割り込み登録
4	MSpciINTClear	割り込みステータス・クリア
5	MSpciINTMaskSet	割り込みマスクの設定
6	MSpciINTMaskClear	割り込みマスクのクリア
7	MSpciSetRunMode	マスタ／ターゲットのアクセス選択
8	MSpciMasterDIR	マスタ転送方向設定
9	MSpciSetDataLENG	マスタ転送バイト数の設定
10	MSpciStart	マスタ転送スタート
11	MSpciStop	マスタ転送ストップ（強制終了）
12	MSpciMemAlloc	マスタ転送時のドライババッファの確保
13	MSpciMemFree	〃　の解放
14	MSpciReadBuff	マスタバッファからのリード
15	MSpciWriteBuff	マスタバッファへのライト
16	MSpciStatusRead	マスタステータス・リード
17	MSpciCNTStatusRead	マスタ制御レジスタ・リード
18	MSpciIoRead	I/O 空間のレジスタリード（PCI-FPGA）
19	MSpciIoWrite	〃　〃　ライト（　〃　）
20	MSpciMemRead	メモリ空間のレジスタリード（SDRAM FPGA）
21	MSpciMemWrite	〃　〃　ライト（　〃　）
22	MSpciConfigRead	PCI コンフィグレジ・リード

8．個別関数説明

関数　1	ボードのオープン
ﾌﾟﾛﾄﾀｲﾌﾟ	HANDLE　MSpciOpen(　BYTE　p1　)
引　数	BYTE　p1：ボード番号 ボードが複数存在しているとき、０～１５を設定する。 １枚目が０で、次のボードから１，2…と番号付けする。 ボードが、１枚のみの場合、この番号は、無視される。
戻り値	HANDLE　：オープンしたボードのハンドル。 このハンドルで、以後の関数を使用する。 エラー時、以下の値を戻す。 INVALID_HANDLE_VALUE （ボードがない場合か、すでにオープンされている場合）
内　容	ボードをオープンし、使用できる状態にする。
備　考	

関数　2	ボードのクローズ
ﾌﾟﾛﾄﾀｲﾌﾟ	int　MSpciClose(　HANDLE　ph　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル）
戻り値	int：ステータス 0　　：複数枚使用時、全てボードは、クローズされている。 0 以外：　〃　　、他にオープンされているボードが存在する。 エラーは、ありません。 （オープンされていないボードをクローズにしてもエラーにしません。）
内　容	現在使用しているボードをクローズします。 そして、どれかのアプリケーションからのオープンを待つ。
備　考	複数アプリケーションで、同じボードを使用する場合、同時に 同じボードをオープンできません。 その場合、使用中のアプリケーションは、ボードをクローズして、 解放後、他のアプリケーションでオープンして使用してください。

関数　3	割り込みの登録
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciINTEntry(　HANDLE　ph,DWORD　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） DWORD　p1：アプリケーションのウインドウハンドル 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	複数枚、１アプリケーションで使用する場合、ウインドウハンドルは、同じになるので、メッセージを送るときパラメータとして ボードハンドル（ph）も送りますので、それによりどのボードかの判断をする。

関数　4	割り込みステータスクリア
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciINTClear(　HANDLE　ph,BYTE　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） BYTE　p1：ステータスクリアデータ （クリアデータ） QENDCLR　：終了割り込みステータスクリア　（0x01） QINT1CLR　：INT1　　〃　　(0x02） QINT2CLR　：INT2　　〃　　(0x04）
戻り値	なし
内　容	割り込みステータスをクリアする。 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。 クリアされたステータスは、メッセージにより渡される。
備　考	

関数　5	**割り込みマスクのセット**
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciINTMaskSet(　HANDLE　ph,BYTE　p1　)
引　数	HANDLE　ph：**ボードオープン時の戻り値（ハンドル）** BYTE　p1：**マスク設定データ** **（マスクデータ）** QENDMASK　：**終了割り込みマスク（マスタ転送）**(0x10) QINT1MASK　：INT1　〃　(0x20) QINT2MASK　：INT2　〃　(0x40)
戻り値	**なし**
内　容	**割り込みのマスクをセット（禁止）する。** **割り込みのマスクは、割り込みステータスに影響を与えない。**
備　考	

関数　6	割り込みマスクのクリア
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciINTMaskClear(　HANDLE　ph,BYTE　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） BYTE　p1：マスク設定データ （マスクデータ） QENDMASK　：終了割り込みマスク（マスタ転送）（0x10） QINT1MASK　：INT1　〃　（0x20） QINT2MASK　：INT2　〃　（0x40）
戻り値	なし
内　容	割り込みをマスクをクリア（許可）する。 割り込みのマスクは、割り込みステータスに影響を与えない。
備　考	

関数　7	アクセスモードを設定する
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MspciSetRunMode(　HANDLE　ph,BYTE　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） BYTE　p1：アクセスモード （アクセスモード） QMSTMODE：マスタモード QTRGMODE：ターゲットモード マスタ　　：SDRAM FPGA とバーストモードでデータ転送を行う。 ターゲット：MSpciMMRead、MspciMmemWrite 関数を使用して SDRAM FPGA のレジスタ（メモリマップ）を アクセスする。
戻り値	なし
内　容	マスタ／ターゲット・モードを選択する。
備　考	マスタアクセスもターゲットアクセスもデータバスを同じ（DAT0～31） バスを使用しているため、同時アクセスはできない。 （注）マスタモード時は、MspciMemRead、MspciMemWrite 関数は、 使用できません。

関数　8	マスタ転送方向の設定
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciMasterDIR(　HANDLE　ph,BYTE　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） BYTE　p1：転送方向 （転送方向） QDIRIN　：入力 QDIROUT：出力
戻り値	なし
内　容	マスタ転送時の方向を設定する。 出力：PCI-FPGA　→　SDRAM FPGA 入力：SDRAM FPGA　→　PCI-FPGA
備　考	マスタ転送中は、設定を変えないこと。

関数　9	マスタ転送バイト数の設定
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciDataLENG(　HANDLE　ph,DWORD　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） DWORD　p1：転送バイト数 （転送バイト数） 設定範囲は、８～１６Mバイト。 設定数は、４の倍数で端数は、切り捨てられる。
戻り値	なし
内　容	マスタ転送バイト数で、MSpciMemAllocで取得したバイト数 以内に設定する。
備　考	

関数　10	マスタ転送開始
ﾌﾟﾛﾄﾀｲﾌﾟ	int　MSpciStart(　HANDLE　ph　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル）
戻り値	int　：エラーステータス 正の値：正常 負の値： ERROR_START1：マスタモードになっていない。 ERROR_START2：転送バイト数を設定していない。 ERROR_START3：転送バッファを取得していない。 ERROR_START4：転送バッファより転送バイト数が大きい。
内　容	マスタ転送を開始する。 しかし、SDRAM FPGA がスタートしていないと待ち状態となる。
備　考	上記エラー時、 スタートは、無視される。（　ERROR_START1～4　）

関数　11	マスタ転送強制終了
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciStop(　HANDLE　ph　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル）
戻り値	なし
内　容	MSpciStart でスタート後、この関数を実行すると終了する。 ただし、この終了では、PCI-FPGA が出力しているコントロール線を 非アクティブにします。 それにより、SDRAM FPGA もその信号で終了させないと マスタ転送のフェーズが合わなくなることがあります。 （非アクティブにするコントロール線） ・ nACK ・ nVLD_W／nENB_R
備　考	この関数の実行による PCI-FPGA の終了の割り込みは、 起こりません。

関数　12	割り込みステータスリード
ﾌﾟﾛﾄﾀｲﾌﾟ	BYTE　MSpciINTStatusRead(　HANDLE　ph　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル）
戻り値	BYTE：割り込みステータスデータ （ステータスビットデータ） QENDSTS　：終了割り込みステータスビットデータ（0x01） QINT1STS　：INT1　〃　（0x02） QINT2STS　：INT2　〃　（0x04）
内　容	割り込みステータスデータをリードする。
備　考	

関数　13	マスタ転送ステータスリード
ﾌﾟﾛﾄﾀｲﾌﾟ	BYTE　MSpciCNTStatusRead(　HANDLE　ph　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル）
戻り値	BYTE：ステータスデータ QMSTBUSY　：マスタビジーデータ（0x80） QACK　　：ACK データ　　（0x20） QREQ　　：REQ データ　　（0x10） QENBW　　：ENB_W データ　　（0x08） QVLDW　　：VLD_W　〃　　（0x04） QENBR　　：ENB_R　〃　　（0x02） QVLDR　　：VLD_R　〃　　（0x01） 上記いづれも“１”でアクティブ。
内　容	マスタ転送時のコントロール信号をリードする。 （注）QMSTBUSY は、動作中のステータスで終了すると “０”になる。
備　考	

関数　14	ドライバのマスタ転送用バッファ取得（パソコンのメモリ）
ﾌﾟﾛﾄﾀｲﾌﾟ	int　MSpciMemAlloc(　HANDLE　ph,DWORD　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） DWORD　p1：取得するバイト数 バイト数は、4 の倍数で最大 16Mバイト設定できる。 設定数が、4 の倍数でない時、端数は切り捨てられる。
戻り値	int　：物理メモリアドレス NULL の時、取得失敗。
内　容	マスタ転送時、ドライバが使用する物理メモリを取得する。
備　考	取得バイト数は、転送バイト数以上設定してください。 転送バイト数以下だと、違うメモリを壊す恐れがあり、それにより パソコンが、異常動作する事もあります。

関数　15	**ドライバのマスタ転送用バッファ解放（パソコンのメモリ）**
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciMemFree(　HANDLE　ph　)
引　数	HANDLE　ph：**ボードオープン時の戻り値（ハンドル）**
戻り値	**なし**
内　容	**関数** MSpciMemAlloc **で取得したメモリを解放します。**
備　考	**アプリケーション終了時には、必ずこの関数を実行する必要がある。**

関数　16	**ドライバのバッファのリード**
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciReadBUFF(　HANDLE　ph,void　*p1, DWORD　p2,DWORD　p3　)
引　数	HANDLE　ph：**ボードオープン時の戻り値（ハンドル）** void　*p1：**アプリケーションバッファのポインター** DWORD　p2：**ドライババッファのバイトオフセット（４の倍数）** DWORD　p3：**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リードする。**
備　考	

関数　17	ドライバのバッファのライト
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciWriteBUFF(　HANDLE　ｐｈ, void　*p1, DWORD　p2,DWORD　p3)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） void　*p1：アプリケーションバッファのポインター DWORD　p2：ドライババッファのバイトオフセット（４の倍数） DWORD　p3：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へライトする。
備　考	

関数　18	ＰＣＩのＩ／Ｏ空間リード
ﾌﾟﾛﾄﾀｲﾌﾟ	DWORD　MspciIoRead(　HANDLE　ph,DWORD　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） DWORD　p1：I/O アドレスオフセット オフセット値は、DWORD アクセスなので４の倍数で設定。
戻り値	Ｉ／Ｏリードデータ（３２ビット）
内　容	PCI-FPGA のＩ／Ｏレジスタをリードする。
備　考	PCI-FPGA はＩ／Ｏマップレジスタを、SDRAM-FPGA は メモリマップレジスタを割り当てている。

関数　19	ＰＣＩのＩ／Ｏ空間ライト
ﾌﾟﾛﾄﾀｲﾌﾟ	viod MspciIoWrite(　HANDLE　ph,DWORD　p1 ,DWORD　p2　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） DWORD　p1：I/O アドレスオフセット DWORD　p2：ライトデータ（３２ビット） オフセット値は、DWORD アクセスなので４の倍数で設定。
戻り値	なし
内　容	PCI-FPGA のＩ／Ｏレジスタにライトする。
備　考	PCI-FPGA はＩ／Ｏマップレジスタを、SDRAM FPGA は メモリマップレジスタを割り当てている。

関数　20	ＰＣＩのメモリ空間リード
ﾌﾟﾛﾄﾀｲﾌﾟ	DWORD　MspciMemRead(　HANDLE　ph,DWORD　p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） DWORD　p1：メモリアドレスオフセット 　オフセット値は、DWORD アクセスなので４の倍数で設定。
戻り値	リードデータ（３２ビット）
内　容	SDRAM FPGA のメモリレジスタからリードする。 （注）ターゲットモードでのみ有効で、 　マスタモード時は、不定である。
備　考	PCI-FPGA はＩ／Ｏマップレジスタを、SDRAM FPGA は メモリマップレジスタを割り当てている。

関数　21	ＰＣＩのメモリ空間ライト
ﾌﾟﾛﾄﾀｲﾌﾟ	void　MSpciMemWrite(　HANDLE　ph,DWORD　p1 ,DWORD　p2　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） DWORD　p1：メモリアドレスオフセット DWORD　p2：ライトデータ（３２ビット） 　オフセット値は、DWORD アクセスなので４の倍数で設定。
戻り値	なし
内　容	SDRAM FPGA のメモリレジスタにデータをライトする。 （注）ターゲットモード時のみ有効で、 　　マスタモード時は、なにもしない。
備　考	PCI-FPGA はＩ／Ｏマップレジスタを、SDRAM FPGA は メモリマップレジスタを割り当てている。

関数　22	ＰＣＩコンフィグレジスタリード
ﾌﾟﾛﾄﾀｲﾌﾟ	voidMSpciConfigRead(HANDLE　ph,struct　PPCI_CONFIG_DATA　*p1　)
引　数	HANDLE　ph：ボードオープン時の戻り値（ハンドル） PPCI_CONFIG_DATA　*p1：PCI コンフィグレジスタへの 構造体ポインター 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 PCI コンフィグレジスタの内容を PPCI_CONFIG_DATA 構造体内に格納する。
備　考	