

Altera 社、大規模 FPGA : APEX シリーズ対応

FPGA 開発用 PCI インターフェイスボード

## G-PCI4 ソフト(下位 DLL)仕様書

---

第 3 版

### 株式会社 ファード

〒183-0006 府中市緑町 3 - 8 - 2 新東邦ビル 4 F

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	版数	改 版	内 容
2002/7/19	初 版		
2003/4/15	第二版	ミス訂正	2 部分
2005.10/4	第 3 版	Win2000/XP のインストール説明追加	

---

## 目 次

<b>1 . G - P C I 4 ボードのドライバインストール方法.....</b>	<b>3</b>
1 - 1 . W I N D O W S N T の場合.....	3
1 - 2 . W I N D O W S 2 0 0 0 / X P の場合.....	3
<b>2 . 添付ソフト.....</b>	<b>4</b>
2 - 1 . ドライバ.....	4
2 - 2 . 関数 ( DLL で提供し、W I N D O W S N T / 2 0 0 0 / X P は、同じファイルを使用します。).....	4
2 - 3 . サンプルプログラム.....	4
<b>3 . 開発手順.....</b>	<b>5</b>
4 . ドライバ転送用のメモリ.....	5
<b>5 . プログラム手順.....</b>	<b>6</b>
5 - 1 . ボードのオープン.....	6
5 - 2 . ボードのクローズ.....	6
5 - 3 . P C I I / O アクセス.....	6
5 - 4 . ターゲットモード.....	6
5 - 5 . マスタモード.....	7
5 - 6 . 割り込みについて.....	9
<b>6 . 定義定数.....</b>	<b>11</b>
6 - 1 . 構造体.....	11
6 - 2 . エラーコード.....	11
6 - 3 . I / O アドレスオフセット.....	11
6 - 4 . 動作モード用定数.....	12
6 - 5 . 割り込み関連コード.....	12
<b>6 - 6 . ステータス.....</b>	<b>12</b>

---

<b>7 . 関数一覧.....</b>	<b>13</b>
<b>7 - 1 . 個別関数説明.....</b>	<b>14</b>
関数 1 ボードのオープン.....	14
関数 2 ボードのクローズ.....	15
関数 3 割り込みの登録.....	16
関数 4 割り込みステータスクリア.....	17
関数 5 割り込みマスクの設定.....	18
関数 6 割り込みマスクのクリア.....	19
関数 7 アクセスモードを設定する.....	20
関数 8 マスタ転送方向の設定.....	21
関数 9 マスタ転送バイト数の設定.....	22
関数 10 マスタ転送開始.....	23
関数 11 マスタ転送強制終了.....	24
関数 12 割り込みステータスリード.....	25
関数 13 マスタ転送ステータスリード.....	26
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).....	27
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).....	28
関数 16 ドライバのバッファのリード.....	29
関数 17 ドライバのバッファのライト.....	30
関数 18 PCI の I / O 空間リード.....	31
関数 19 PCI の I / O 空間ライト.....	32
関数 20 PCI のメモリ空間リード.....	33
関数 21 PCI のメモリ空間ライト.....	34
関数 22 PCI コンフィグレジスタリード.....	35

## 1. G - P C I 4 ボードのドライバインストール方法

### 1 - 1 . WindowsNT の場合

- ( 1 ) Window NT(Ver 4.0)で、上のドライバ部分をハードディスクのディレクトリにコピーし、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、「GPCI4」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 P C I バスに挿入します。
- ( 2 ) 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、Gpci4 . 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。

```
F i r d G - P C I 4
  |
  └─ G - P C I 4
```

## 2 . 添付ソフト

### 2 - 1 . ドライバ

#### ( 1 ) WindowsNT

¥NT\_Driver¥ GPCI4.SYS (ドライバー本体)  
GPCI4.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¥ GPCI4.SYS (ドライバー本体)  
GPCI4.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

GPCI4DLL.LIB (コンパイル時の参照ファイル)  
GPCI4.DLL (関数本体)  
GPCI4.H (関数定義)

#### ( 2 ) 上位 DLL

GPCI4.LIB (コンパイル時の参照ファイル)  
GPCI4API.DLL (関数本体)  
GPCI4API.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 ( ¥G4TestSaml¥ ) に含まれています

### 2 - 3 . サンプルプログラム

¥G4TestSampl¥ G4TESTSAMPLE.EXE (実行ファイル)  
およびその project file 一式  
およびその project file 一式

### 3 . 開発手順

- 1 ) 下位 DLL、上位 DLL を開発するプロジェクトのディレクトリにコピーしておきます。
- 2 ) Visual C++の「プロジェクト」 > 「設定」で「リンク」のタグの「オブジェクト / ライブラリ」の中に、GPCI4DLL.LIB と GPCI4API.LIB を書き込み指定します。
- 3 ) これで以下の標準関数を使用したプログラムの開発ができます。

- ・ドライバーのインストール、添付ファイル、開発手順は「上位 DLL 説明書」を参照してください。

### 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。

## 5 . 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。

ここでの説明は、下位関数のみです。

### 5 - 1 . ボードのオープン

```
HANDLE hVxD ;           //ハンドル定義
hVxd=Gpci4Open(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。

### 5 - 2 . ボードのクローズ

```
int ret;
ret=Gpci4Close( hVxD );
```

(注) アプリケーション終了時には、必ず実行する事。

### 5 - 3 . P C I I / Oアクセス

P C IのI / Oは、P C I - F P G A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

```
Gpci4IoWrite( hVxD,offset,data ); // 32ビットアクセスのみ。
```

(2) リード

```
data=Gpci4IoRead( hVxD,offset,data ); // 32ビットアクセスのみ。
```

P C I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。

### 5 - 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、S D R A M F P G Aへのメモリレジスタにアクセスするモードです。

P C Iのメモリエリアは、S D R A M F P G Aに割り当てられています。

```
Gpci4SetRunMode(hVxD,QTARGMODE);
```

(2) メモリ (レジスタ) ライト

```
Gpci4MemWrite( 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=Gpci4MemRead( hVxD,offset ); // 32ビットアクセスのみ
```

## 5 - 5 . マスタモード

( 共通 1 ) S D R A M F P G A 側のマスタスタートを実行し、待ち状態にする。  
S D R A M F P G A 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

```
Gpci4SetRunMode ( hVxD,QMSTMODE );
```

( 1 ) マスタリード ( S D R A M F P G A P C I - F P G A )

転送方向を設定

```
Gpci4MasterDIR( hVxd,QDIRIN );
```

ドライバのバッファを確保

```
int ret;
ret=Gpci4MemAlloc( 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

```
Gpci4DataLENG( hVxD,data_size );
//転送長は、data_size を設定
```

スタート

```
int ret;
ret=Gpci4Start( hVxD ); // ret=正の値の時正常
// 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。

```
BYTE status;
status=Gpci4StatusRead( 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。
```

ドライバのバッファをリードする。

```
Gpci4ReadBuff( hVxD,buf,data_size );
```

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
Gpci4MemFree( hVxD ); //ドライババッファ解放
```



## ( 2 ) マスタライト ( P C I - F P G A S D R A M 側 F P G A )

転送方向を設定

```
Gpci4MasterDIR( hVxd,QDIROUT );
```

ドライバのバッファを確保

```
int ret;
ret=Gpci4MemAlloc( 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

```
Gpci4DataLENG( hVxD,data_size );
//転送長は、data_size 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。

```
Gpci4WriteBuff( hVxD,buf,data_size );
```

スタート

```
int ret;
ret=Gpci4Start( hVxD ); // ret=正の値の時正常
// 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。

```
BYTE status;
status=Gpci4StatusRead( 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。
```

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
Gpci4MemFree( hVxD ); //ドライババッファ解放
```

## 5 - 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

```
Gpci4INTEntry( 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 CGpci4sampView::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。
        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 Gpci4sampView）、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。

## 6 . 定義定数

定義定数については、“Gpci4Dll.h”のファイルに入っています。

### 6 - 1 . 構造体

- ・ 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;
};
```

Gpci4ConfigRead 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していない。
#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 // P C I 転送アドレスレジスタ
#define QMSTLENIO  0x08 // P C I 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 ( Gpci4IoDADIR、 Gpci4MasterDIR 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 ( Gpci4AccessCnt 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

Gpci4StatusRead 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
   Gpci4Start 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

“Gpci4Dll.dll” で D L L 形式で供給

No.	関数名	概略内容
1	Gpci4Open	ボードのオープン（使用開始）
2	Gpci4Close	ボードのクローズ（使用終了）
3	Gpci4INTEntry	割り込み登録
4	Gpci4INTClear	割り込みステータス・クリア
5	Gpci4INTMaskSet	割り込みマスクのセット
6	Gpci4INTMaskClear	割り込みマスクのクリア
7	Gpci4SetRunMode	マスタ/ターゲットのアクセス選択
8	Gpci4MasterDIR	マスタ転送方向設定
9	Gpci4SetDataLENG	マスタ転送バイト数の設定
10	Gpci4Start	マスタ転送スタート
11	Gpci4Stop	マスタ転送ストップ（強制終了）
12	Gpci4MemAlloc	マスタ転送時のドライババッファの確保
13	Gpci4MemFree	” ” の解放
14	Gpci4ReadBuff	マスタバッファからのリード
15	Gpci4WriteBuff	マスタバッファへのライト
16	Gpci4INTStatusRead	割り込みステータス・リード
17	Gpci4CNTStatusRead	マスタ制御レジスタ・リード
18	Gpci4IoRead	I/O 空間のレジスタリード（PCI-FPGA）
19	Gpci4IoWrite	” ” ライト（ ” ）
20	Gpci4MemRead	メモリ空間のレジスタリード（SDRAM FPGA）
21	Gpci4MemWrite	” ” ライト（ ” ）
22	Gpci4ConfigRead	PCI コンフィグレジ・リード

## 7 - 1 . 個別関数説明

関数 1	ボードのオープン
プロトタイプ	HANDLE Gpci4Open( BYTE p1 )
引 数	<p>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。</p> <p>ボードが、1 枚のみの場合、この番号は、無視される。</p>
戻り値	<p>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 )</p>
内 容	<p>ボードをオープンし、使用できる状態にする。</p>
備 考	

関数 2	ボードのクローズ
プロトタイプ	int Gpci4Close( 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
プロトタイプ	voidGpci4INTEntry( 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  ウインドウハンドルは、(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
プロトタイプ	void Gpci4INTClear( 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	
プロトタイプ	void Gpci4INTMaskSet( 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	設定されたビットの割り込みマスクをセット (禁止) する。 他のビットには影響を与えない。  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
プロトタイプ	void Gpci4INTMaskClear( 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 (マスク転送) (0x10) QINT1MASK : INT1 " (0x20) QINT2MASK : INT2 " (0x40)
戻り値	なし
内 容	設定されたビットの割り込みマスクをクリア (許可) する。 他のビットには影響を与えない。 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
プロトタイプ	void Gpci4SetRunMode( HANDLE ph, BYTE p1 )
引 数	<p>HANDLE ph : ボードオープン時の戻り値 (ハンドル)</p> <p>BYTE p1 : アクセスモード (アクセスモード)</p> <p>QMSTMODE : マスタモード</p> <p>QTRGMODE : ターゲットモード</p> <p>マスタ : SDRAM FPGA とバーストモードでデータ転送を行う。</p> <p>ターゲット : Gpci4MMRead、Gpci4MmemWrite 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。</p>
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	<p>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。</p> <p>(注) マスタモード時は、Gpci4MemRead、Gpci4MemWrite 関数は、 使用できません。</p>

関数 8	マスタ転送方向の設定
プロトタイプ	void Gpci4MasterDIR( 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
プロトタイプ	void Gpci4SetDataLENG( 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 ~ 16 M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、Gpci4MemAlloc 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
プロトタイプ	int Gpci4Start( 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
プロトタイプ	void Gpci4Stop( 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	<p>Gpci4Start 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)</p> <ul style="list-style-type: none"><li>・ nACK</li><li>・ nVLD_W / nENB_R</li></ul>
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
プロトタイプ	BYTE Gpci4INTStatusRead( 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 (0x1) QINT1STS : INT1 " (0x2) QINT2STS : INT2 " (0x4)
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
プロトタイプ	BYTE Gpci4CNTStatusRead( 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	<p>BYTE : ステータスデータ</p> <p>QMSTBUSY : マスタビジーデータ (0x80)</p> <p>QACK : ACK データ (0x20)</p> <p>QREQ : REQ データ (0x10)</p> <p>QENBW : ENB_W データ (0x08)</p> <p>QVLDW : VLD_W " (0x04)</p> <p>QENBR : ENB_R " (0x02)</p> <p>QVLDR : VLD_R " (0x01)</p> <p>上記いづれも " 1 " でアクティブ。</p>
内 容	<p>マスタ転送時のコントロール信号をリードする。</p> <p>(注) QMSTBUSY は、動作中のステータスで終了すると " 0 " になる。</p>
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)
プロトタイプ	int Gpci4MemAlloc( 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 16 M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)
プロトタイプ	void Gpci4MemFree( 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 Gpci4MemAlloc 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
プロトタイプ	void Gpci4ReadBUFF( 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 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
プロトタイプ	void Gpci4WriteBUFF( HANDLE p h , 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 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
プロトタイプ	DWORD Gpci4IoRead( 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
プロトタイプ	viod Gpci4IoWrite( 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
プロトタイプ	DWORD Gpci4MemRead( 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。 <u>(注) ターゲットモードでのみ有効で、</u> <u>マスタモード時は、不定である。</u>
備考	PCI-FPGA はI/Oマッピングレジスタを、SDRAM FPGA はメモリマッピングレジスタを割り当てている。

関数 21	P C Iのメモリ空間ライト
プロトタイプ	void Gpci4MemWrite( 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 ( 3 2 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	P C I コンフィグレジスタリード
プロトタイプ	void Gpci4ConfigRead( 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 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	