

# GPCle 開発キットユーザガイド

## for GPCle DevKit version 1.4.3

最終更新：2013 年 2 月 8 日

**K&F** Computing Research Co.

株式会社 K & F Computing Research  
E-mail: support@kfcr.jp

# 目次

<b>1</b>	<b>本文書の概要</b>	<b>4</b>
<b>2</b>	<b>パッケージの内容</b>	<b>4</b>
<b>3</b>	<b>機能概要</b>	<b>6</b>
3.1	対応する PCI Express デバイスタイプ	6
3.2	対応する FPGA デバイス	6
3.3	回路構成	7
<b>4</b>	<b>基礎的な使用方法</b>	<b>8</b>
4.1	回路の合成	9
4.2	ホスト計算機からの制御	10
4.2.1	Linux 向けソフトウェアのインストール	10
4.2.2	デバイスドライバの設定	11
4.2.3	動作テスト	12
4.2.4	MTRR の設定	15
4.2.5	ユーザライブラリの使用方法	16
<b>5</b>	<b>より高度な使用方法</b>	<b>18</b>
5.1	ソースコードのアップデート	18
5.2	より高速に動作する回路の生成	19
5.3	PHY チップの使用方法	19
5.4	GPCle エンジンを直接制御する使用方法	19
5.5	割込みの使用方法	22
<b>6</b>	<b>エンティティ詳細</b>	<b>24</b>
6.1	エンティティ hib 詳細	24
6.1.1	ヘッダ部の記述	24
6.1.2	Generic 宣言	24
6.1.3	Port 宣言	26
6.2	エンティティ gpcie 詳細	29
6.2.1	ヘッダ部の記述	29
6.2.2	Generic 宣言	29
6.2.3	Port 宣言	35
6.3	エンティティ phypcs 詳細	44
6.3.1	ヘッダ部の記述	44
6.3.2	Generic 宣言	44

6.3.3	Port 宣言	45
<b>7</b>	<b>利用許諾</b>	<b>50</b>
7.1	GPCle DS, GPCle SP 利用許諾	50
7.2	GPCle 機能限定版 利用許諾	50
<b>8</b>	<b>GPCle 開発キット更新履歴</b>	<b>51</b>

# 1 本文書の概要

この文書では PCI Express IP コア GPCle 開発キットの使用方法を説明します。

## 2 パッケージの内容

GPCle は株式会社 K&F Computing Research 開発の PCI Express IP コアです。ユーザ自身の設計した回路へ GPCle を組み込むことで、PCI Express プロトコルの詳細に立ち入らずに他の PCI Express デバイスとのインタフェースを実現できます。このパッケージには以下の 3 つの回路が含まれます (すべての回路は VHDL ソースコードで提供されます):

1. GPCle 最上位階層に位置し、ユーザ回路に対して簡便なインタフェーシングを行う Host Interface Bridge (HIB)。
2. PCI Express 規格で定められたトランザクション層、データリンク層、PHY MAC 層と、それらの上に位置するアプリケーション層 (PCI Configuration Register や DMA コントローラ) を実現する GPCle エンジン。
3. Altera 社 FPGA の内蔵トランシーバ向け PHY PCS 層、PMA 層 を実現する PHY。

またこのパッケージには、HIB を利用したインタフェース回路のリファレンスデザイン (サンプル回路) と、それをホスト計算機から制御するためのソフトウェアも含まれています。このパッケージの

ファイル構成は以下の通りです:

gpciepkg/	
00readme-j	本ファイル。
00readme	本ファイルの英語版。
doc/	ユーザガイド、その他の文書。
script/	パッケージ管理ユーティリティ。
software/	HIB 制御用ソフトウェア。
driver/	HIB デバイスドライバソースコード (Linux 用)。
win/	HIB デバイスドライバソースコード (Windows 用)。
hibutil/	HIB 制御ライブラリソースコード (Linux 用)。
win/	HIB 制御ライブラリソースコード (Windows 用)。
include/	HIB 制御ライブラリヘッダファイル。
lib/	HIB 制御ライブラリ。
sample/	HIB 制御ライブラリを使用したアプリケーションプログラムの例。
tool/	HIB 制御用ソフトウェアの管理ユーティリティ。
hardware_altera/	GPCIe 本体およびリファレンスデザイン (Altera 社 FPGA 用)。
hib.vhd	GPCIe 最上位階層 Host Interface Bridge (HIB)。
gpcie.vhd	アプリケーション層、トランザクション層、データリンク層、PHY MAC 層。
phy.vhd	Altera 社 FPGA 内蔵トランシーバ用の PHY PCS 層、PHY PMA 層。
topdesign/	HIB を用いたインタフェース回路サンプルの最上位階層。
synth/	リファレンスデザインの合成に使用するファイル (.qpf, .qsf, .sdc)。
Makefile	テンプレートから hib.vhd, gpcie.vhd, phy.vhd を生成するための makefile。
template/	VHDL ソーステンプレート。
hardware_xilinx/	GPCIe 本体およびリファレンスデザイン。 (Xilinx 社 FPGA 用。Xilinx 対応版 GPCIe パッケージのみ)。
hib.vhd	GPCIe 最上位階層 Host Interface Bridge (HIB)。
gpcie.vhd	アプリケーション層、トランザクション層、データリンク層、PHY MAC 層。
phy.vhd	Altera 社 FPGA 内蔵トランシーバ用の PHY PCS 層、PHY PMA 層。
topdesign/	HIB を用いたインタフェース回路サンプルの最上位階層。
synth/	リファレンスデザインの合成に使用するファイル (.qpf, .qsf, .sdc)。
Makefile	テンプレートから hib.vhd, gpcie.vhd, phy.vhd を生成するための makefile。
template/	VHDL ソーステンプレート。

本ユーザガイド第 3 節では GPCIe の機能概要を説明します。第 4 節では GPCIe の基礎的な使用方法について説明します。この節では GPCIe の HIB 階層と Altera 社 FPGA 内蔵トランシーバの利用を前提とした説明を行います。第 5 節では GPCIe のより高度な使用方法について説明します。内蔵トランシーバの代わりに外付けの PHY チップを使用する方法や、HIB 階層を介さずに GPCIe エンジンに直接制御する使用方法の説明を行います。第 6 節では主要な VHDL エンティティの I/O ポートについて説明します。

なお以降では、ファイルのパス名はすべて本パッケージのルートディレクトリ `gpciepkg/` からの相対パスで表記します。

### 3 機能概要

本節では GPCle のサポートしている機能と FPGA デバイス、回路構成を説明します。

#### 3.1 対応する PCI Express デバイスタイプ

エンドポイントとして動作します。スイッチやルートコンプレクス内のルートポートとしては利用できません。

#### 3.2 対応する FPGA デバイス

デバイス	Gen1 (2.5Gbps)			Gen2 (5.0Gbps)		
	x8	x4	x1	x8	x4	x1
	125MHz			250MHz		
PIPE I/F	128b	64b	16b	128b	64b	16b
StratixIV GX	**	**	**	-	-	-
CycloneIV GX	.	**	**	-	-	-
CycloneIII w/PHY chip	.	.	**	-	-	-
StratixII GX	***	***	***	*	*	*
Arria GX	***	***	***	-	-	-
Spartan3 w/PHY chip	.	.	**	-	-	-

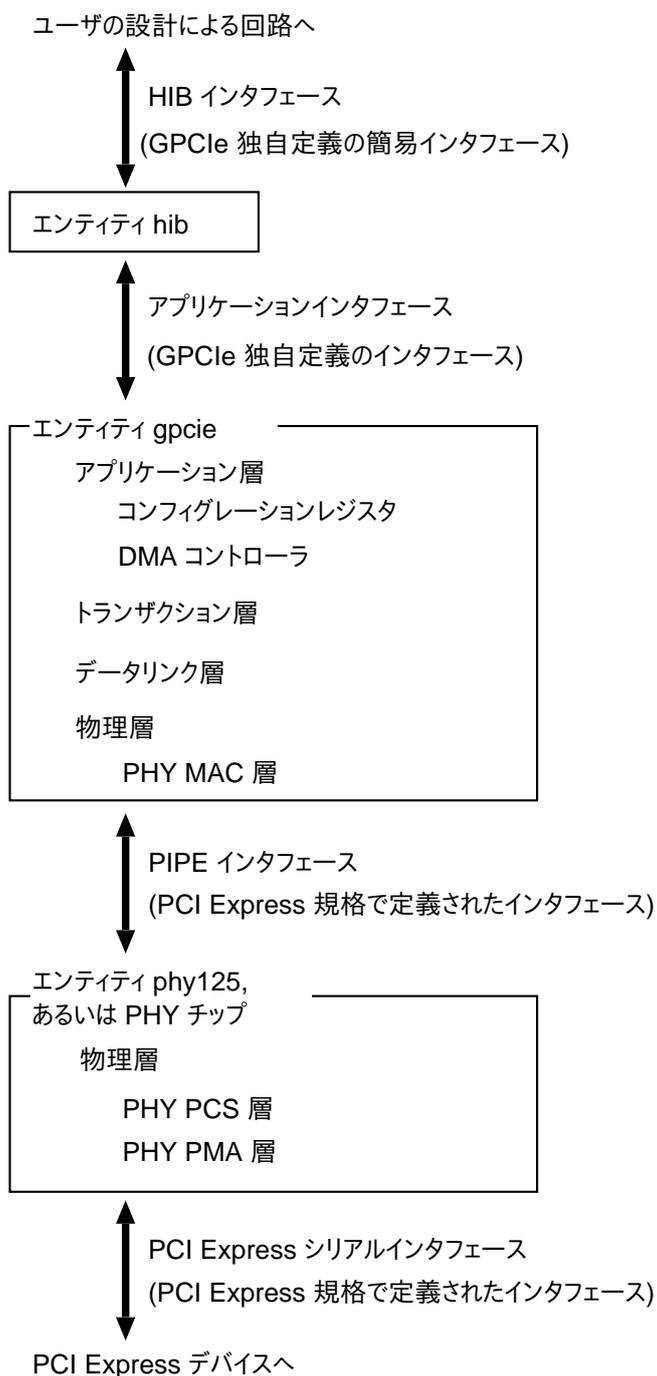
\*\*\* :GPCle のすべてのエディションで対応

\*\* :GPCle SP, GPCle DS のみで対応。GPCle 機能限定版では非対応。

\* :GPCle DS のみで対応。GPCle 機能限定版、GPCle SP では非対応。

. :GPCle DS のみで実装済み。動作未確認。

### 3.3 回路構成



GPCle の回路は hib、gpcie、phypcs の 3 つのエンティティから構成されています。

エンティティ hib は GPCle 最上位階層に位置し、ユーザ回路に対して簡便なインタフェースを提供します。

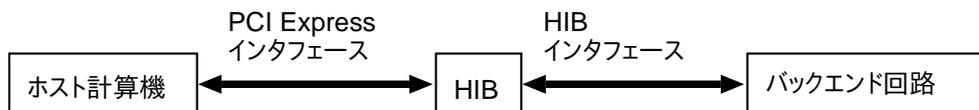
エンティティ `gpcie` は PCI Express 規格で定められた物理層 (PHY MAC 層)、データリンク層、トランザクション層と、その上位に位置するアプリケーション層を実装します。アプリケーション層には PCI Configuration Register と DMA コントローラが含まれます。

エンティティ `phypcs` は FPGA 内蔵 Gbit トランシーバ向けの物理層 (PHY PCS 層、PMA 層) を実装します。外付けの PHY チップを用いる場合には GPCle の PIPE インタフェースに直接接続して使用するため、このエンティティは使用しません。

## 4 基礎的な使用方法

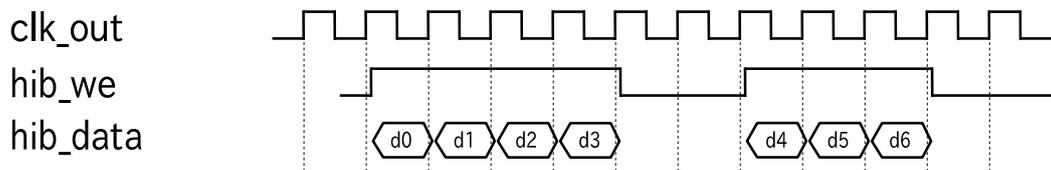
本節では GPCle の基礎的な使用方法について説明します。この節では HIB と FPGA 内蔵トランシーバの利用を前提として説明を行います。

ユーザの設計による回路 (以降バックエンド回路と呼びます) が GPCle を使用するには、エンティティ `hib` をインスタンス化します。バックエンド回路は HIB インタフェースを通じてホスト計算機 (より正確には PCI Express リンクの上流に位置するデバイス) との間でデータ転送を行います。HIB は PCI Express プロトコルによるデータ転送と、HIB インタフェースを介したデータ転送とをブリッジします。

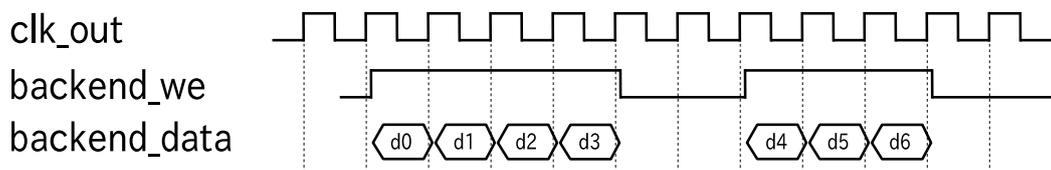


ホスト計算機 から HIB への転送は Programm I/O (PIO) による write によって行います。HIB からのホスト計算機への転送は Direct Memory Access (DMA) による write によって行います。本パッケージにはこれらの転送を行うための Linux 用ソフトウェア (デバイスドライバとユーザライブラリ) が含まれています。ソフトウェアの使用方法については後述します。

バックエンドと HIB との転送は `hib_we`, `hib_data`, `backend_we`, `backend_data` の 4 種の信号と、125MHz のクロック信号 `clk_out` を用いて行います。バックエンドから HIB への転送は、バックエンドのドライブする write 信号 `backend_we` と、それに同期したデータ入力 `backend_data` で行います。HIB からバックエンドへの転送は、HIB のドライブする write 信号 `hib_we` と、それに同期したデータ出力 `hib_data` で行います。



HIB からバックエンドへの書き込み



バックエンドから HIB への書き込み

バックエンドは HIB からの書き込みを待たせることはできません。hib\_we が assert されている間は、常にデータを受け取り続けなくてはなりません。

HIB はバックエンドからの書き込みを受け取るためのバッファを内部に持っています。ホスト計算機から DMA write 要求があると、HIB はこのバッファの内容をホスト計算機へ write します。HIB 自身はバッファのオーバーフローチェックを行いませんので、バックエンドがバッファサイズを超えるデータを write する場合には、バックエンドが書き込みタイミングの制御を行う必要があります。バッファのサイズはデフォルトでは 8k byte に設定されています。

HIB 内部には DMA 制御回路や PIO write 制御回路、それらへホスト計算機からアクセスするためのローカルレジスタ群などが実装されています。ローカルレジスタ群へは BAR0 空間を介してアクセスします。PIO write は、BAR2 空間に対して行います。PIO write は BAR2 を write combined に設定して使用すると高い転送速度を得られるよう設計されています。詳細についてはソースコード `template/hibctl.vhd` を参照してください。

HIB を実装したユーザ回路のサンプルが `topdesign/ifpga_xxx{8,4,1}.vhd` にあります (ここで xxx は agx, s2gx, c3, c4gx などのデバイス名の略号を表します)。次節以降ではこのサンプル回路を例に、回路の合成方法とホスト計算機からの制御方法について説明します。

## 4.1 回路の合成

合成には QuartusII を使用します。それ以外のツールでの合成は確認していません。QuartusII のプロジェクトファイル (.qpf) と設定ファイル (.qsf) のサンプルが

```
synth/ifpga_xxx{8,4,1}.qpf
synth/ifpga_xxx{8,4,1}.qsf
```

にあります。これらを使用して前述のサンプル回路

```
topdesign/ifpga_xxx{8,4,1}.vhd
```

を合成すると、KFCR 社製の評価ボード GPCle-Eval-AGX8 および GPCle2-Eval-S2GX8 で動作する回路が生成されます。なお合成に必要な VHDL ソースファイルは

```
hib.vhd
topdesign/ifpga_xxx{8,4,1}.vhd
```

の 2 ファイルのみです。HIB は内部的に GPCle および FPGA 内蔵トランシーバの制御回路を用いており、それらは gpcie.vhd, phy.vhd で記述されていますが、これらのファイルは hib.vhd に含まれているため、合成には gpcie.vhd, phy.vhd は必要ありません。

## 4.2 ホスト計算機からの制御

本パッケージには、HIB をホスト計算機から制御するためのソフトウェアが含まれています。ソフトウェアはデバイスドライバとユーザライブラリから構成されています。これらのソフトウェアは Linux および Windows 上で動作します (GPCle 機能限定版に含まれるソフトウェアは Linux のみで動作します)。本節ではこれらのソフトウェアのインストール、設定、使用方法について説明します。

Linux, Windows 以外の OS 向けのソフトウェアは現在のところパッケージには含まれていません。ただしこれは、HIB 回路の設計がこれらの OS に依存していることを意味するものではありません。適切なソフトウェアを用意すれば、その他の OS から HIB を制御することも可能です。

### 4.2.1 Linux 向けソフトウェアのインストール

ソフトウェアのインストールは scripts/install.csh を実行し、その指示に従ってください。

```
kawai@localhost[1]>./scripts/install.csh
-----
Host Interface Bridge (HIB) software package
installation program.
-----

-----
Installing HIB device driver...
-----
```

(中略)

```
gcc -O0 -g -I. -I../include -o hibtest hibtest.c hibutil.c -lm
gcc -O0 -g -I. -I../include -o lsgrape lsgrape.c hibutil.c -lm
```

```
done
```

なおソフトウェアのインストールには Linux カーネルの完全なソースツリーが必要です。

#### 4.2.2 デバイスドライバの設定

デバイスドライバを使用するには、デバイスドライバを Linux カーネルへリンクする必要があります。そのためには `driver/` ディレクトリへ移動し、ルート権限下で `make installmodule` を実行します。

```
[root@localhost driver]# make installmodule
./install0.csh

-- install module hibdrv --
hibdrv: 1 HIB(s) found.

rm -f /dev/hibdrv[0-9]

/sbin/insmod -f hibdrv.ko
mknod /dev/hibdrv0 c 253 0

chgrp wheel /dev/hibdrv0

chmod 666 /dev/hibdrv0
crw-rw-rw- 1 root wheel 253, 0 Jul  9 12:59 /dev/hibdrv0
-- done --
```

この操作によって HIB のデバイスドライバ `hibdrv` が Linux カーネルにリンクされます。この操作はホスト計算機を起動するたびに必要です。`/sbin/lsmmod` コマンドを実行し、その出力に `hibdrv` という語を含む行が含まれていれば、デバイスドライバは正常にリンクされています。

```
kawai@localhost[2]>lsmmod
Module          Size  Used by
hibdrv          39608  0
...            ...    ...
```

デバイスドライバの設定を終えると、ホスト計算機上のプログラムから HIB へアクセスできるようになります。

### 4.2.3 動作テスト

hibutil/hibtest コマンドを用いると、HIB の動作テストを行えます。例えば PCI configuration register への read/write, local register への read/write、転送速度の測定などが可能です。以下に使用例を示します。

hibtest を引数無しで実行すると、使用法が表示されます。

```
kawai@localhost[3]>./hibtest
usage: ./hibtest <test_program_ID>
 0) show contents of config & HIB-local registers [devid]
 1) reset DMA and FIFO [devid]
 2) clear HIB-internal FIFO [devid]
 3) show DMA status [devid]
 4) read config register <addr> [devid]
 5) write config register <addr> <val> [devid]
 6) read HIB local registers mapped to BAR0 <addr> [devid]
 7) write HIB local registers mapped to BAR0 <addr> <val> [devid]
 8) read backend memory space mapped to BAR1 <addr> [devid]
 9) write backend memory space mapped to BAR1 <addr> <val> [devid]
10) check DMA read/write function <size> <sendfunc> [devid] (host <-> HIB)
11) measure DMA performance <sendfunc> [devid] (host <-> HIB)
12) measure DMA write performance [devid] (host <- HIB; bypass internal FIFO)
13) measure DMA read performance <sendfunc> [devid] (host -> HIB; bypass internal FIFO)
14) reset backend [devid]
15) raw PIO r/w & DMA r/w [devid]
16) measure DMA performance with multiple HIBs <sendfunc> <# of hibs>
    (host <-> HIBs internal FIFO)
17) measure DMA write performance with multiple HIBs <# of hibs> [devid offset]
    (host <- HIBs; bypass internal FIFO)
18) measure DMA read performance with multiple HIBs <sendfunc> <# of hibs> [devid offset]
    (host -> HIBs; bypass internal FIFO)
19) erase configuration ROM (EPCS64) [devid]
20) write .rpd to configuration ROM (EPCS64) <rpd-file> [devid]
21) read configuration ROM ID (0x10:EPCS1 0x12:EPCS4 0x14:EPCS16 0x16:EPCS64) [devid]
22) set pipeline clock frequency to (PCI-X_bus_freq * N / M) <N> <M> [devid]
```

hibtest 0 を実行すると、HIB の PCI configuration register の内容が出力されます。

```
kawai@localhost[4]>./hibtest 0
## hib0:
protocol : PCIe
link width negotiated : x8
           supported  : x8
link speed negotiated : 2.5 Gb/s
           supported  : 2.5 Gb/s
max payload size negotiated : 128 byte
           supported  : 256 byte
```

max read request size : 256 byte

configuration register:

```
0x00000000: 0x0e701b1a
0x00000004: 0x00100007
0x00000008: 0xff000001
0x0000000c: 0x00000008
0x00000010: 0xdf608008 0xdf608000
0x00000014: 0xdf610008 0xdf610000
0x00000018: 0xdf600008 0xdf600000
0x0000001c: 0x00000000 0x00000000
0x00000020: 0x00000000
0x00000024: 0x00000000
0x00000028: 0x00000000
0x0000002c: 0x0e701b1a
0x00000030: 0x00000000
0x00000034: 0x00000080
0x00000038: 0x00000000
0x0000003c: 0x000000ff
PCI Express Capability Register:
0x00000080: 0x00110010
0x00000084: 0x00000001
0x00000088: 0x00001000
0x0000008c: 0x00000481
0x00000090: 0x00810000
```

hibtest 10 10 1 を実行すると、ループバック転送のテストを行えます。10 \* 8 byte のデータを PIO write によってホスト計算機から HIB へ送信し、そのデータをそのまま DMA write によって回収します。送信したデータと回収したデータが完全に一致した場合には OK を、一致しない場合には NG を出力します。

```
kawai@localhost[5]>./hibtest 10 10 1
```

```
# check hib[0] DMA read/write (host <-> HIB internal FIFO)
```

```
size 10
```

```
# hib[0] PIO write, and then DMA write (host <-> HIB internal FIFO)
```

```
clear DMA buf...
```

```
DMA read size: 10 words (80 bytes)
```

```
will dmar...
```

```

rbuf[0000]: 0x1111111111111111  wbuf[0000]: 0x1111111111111111
rbuf[0001]: 0x2222222222222222  wbuf[0001]: 0x2222222222222222
rbuf[0002]: 0x3333333333333333  wbuf[0002]: 0x3333333333333333
rbuf[0003]: 0x4444444444444444  wbuf[0003]: 0x4444444444444444
rbuf[0004]: 0x5555555555555555  wbuf[0004]: 0x5555555555555555
rbuf[0005]: 0x6666666666666666  wbuf[0005]: 0x6666666666666666
rbuf[0006]: 0x123456789abc0006  wbuf[0006]: 0x123456789abc0006
rbuf[0007]: 0x123456789abc0007  wbuf[0007]: 0x123456789abc0007
rbuf[0008]: 0x123456789abc0008  wbuf[0008]: 0x123456789abc0008
rbuf[0009]: 0x123456789abc0009  wbuf[0009]: 0x123456789abc0009
---- transfer size reached ----
rbuf[0010]: 0x123456789abc000a  wbuf[0010]: 0xfedcba987654000a
rbuf[0011]: 0x123456789abc000b  wbuf[0011]: 0xfedcba987654000b
done
  10 words (80 bytes).
OK

```

hibtest 12 を実行すると DMA write 転送 (HIB からホスト計算機への write) の速度を測定できます。

```

kawai@localhost[6]>./hibtest 12

# hib[0] DMA write (host <- HIB)
size: 1024 DMA write: 1.562367 sec  512.043597 MB/s
size: 2048 DMA write: 1.101087 sec  726.554697 MB/s
size: 4096 DMA write: 0.857353 sec  933.104598 MB/s
size: 8192 DMA write: 0.739353 sec  1082.027209 MB/s
size: 16384 DMA write: 0.680854 sec  1174.995203 MB/s
size: 32768 DMA write: 0.651100 sec  1228.690060 MB/s

```

hibtest 13 1 を実行すると PIO write 転送 (ホスト計算機から HIB への write) の速度を測定できます。

```

kawai@localhost[7]>./hibtest 13 1

# hib[0] PIO write (host -> HIB)
size: 64 PIO write: 2.037641 sec  392.610858 MB/s
size: 128 PIO write: 1.233335 sec  648.647763 MB/s
size: 256 PIO write: 0.822831 sec  972.253211 MB/s
size: 512 PIO write: 0.639186 sec  1251.591587 MB/s
size: 1024 PIO write: 0.620417 sec  1289.455073 MB/s
size: 2048 PIO write: 0.620460 sec  1289.365885 MB/s
size: 4096 PIO write: 0.620398 sec  1289.495211 MB/s

```

```
size: 8192 PIO write: 0.620425 sec 1289.438721 MB/s
size: 16384 PIO write: 0.620416 sec 1289.457550 MB/s
```

hibtest を用いたその他のテストについてはソースコード `hibutil/hibtest.c` を参照してください。

#### 4.2.4 MTRR の設定

ホスト計算機から HIB へのデータ転送は、PCI アドレス空間 BAR2 に対する PIO write によって行われます。ピーク性能に近い転送速度を得るには、BAR2 空間を write combining モードに設定する必要があります。設定しない場合でも転送は行えますが、ピーク性能の 20% あるいはそれ以下の転送速度しか得られません。

BAR2 空間を write combining モードに設定するには、ルート権限下でスクリプト `scripts/setmtrr.csh` を実行します。

```
[root@localhost driver]# ./setmtrr.csh

Searching for HIB(s)... Found 0 PCI-X HIB(s). Found 1 PCIe HIB(s).
Found 1 HIB(s) in total.

Trying to set 1 MTRR(s)...
  echo "base=0xdf600000 size=0x1000 type=write-combining" > /proc/mtrr
Done.

current setting of MTRRs:
reg00: base=0x00000000 ( 0MB), size=2048MB: write-back, count=1
reg01: base=0x80000000 (2048MB), size=1024MB: write-back, count=1
reg02: base=0x100000000 (4096MB), size=200704MB: write-back, count=1
reg03: base=0x200000000 (8192MB), size=1024MB: write-back, count=1
reg04: base=0xdf600000 (3574MB), size= 4KB: write-combining, count=1
```

スクリプトの出力に "base=0xAAAAAAAA (XXXXMB), size = 4kB: write-combining" という文字列が含まれていれば、正しく設定を行えています。ただし AAAAAAAAA は HIB の BAR2 空間の先頭アドレスです。この値は `hibtest` コマンドを 2 つの引数 4 18 を与えて実行することで確認できます。

```
kawai@localhost[8]>../hibutil/hibtest 4 18
hib[0] config 0x00000018: 0xdf600008
```

注意 1: ホスト計算機の設定によっては MTRR を設定できないことがあります (メモリを 4GB 以上搭載している場合や、CPU の持つ 8 個すべての MTRR が他の PCI デバイスによって既に使用されている場合など)。BIOS の設定変更によってこの問題を回避できる場合があります。設定方法はチップセットやマザーボードに依存しますので、それらに関する資料を参照してください。

注意 2: Linux kernel バージョン 2.6.26 以降で、page attribute table (PAT) サポートが有効になっている場合には、MTRR の設定は不要です。PAT サポートの有無は、Linux kernel のヘッダファイル `autoconf.h` (e.g. `/usr/src/linux/include/linux/autoconf.h`) で確認できます。ファイル内で定数 `CONFIG_X86_PAT` が定義されていれば PAT サポートは有効になっています。

write combining モード設定の前後で `hibtest 13 1` を用いた PIO write 転送速度の測定を行うと、速度の向上を確認できます。

write combining モード設定前 (8 レーン) :

```
kawai@localhost[9]>./hibtest 13 1
# hib[0] PIO write (host -> HIB)
size: 64 PIO write: 7.319836 sec  109.292068 MB/s
size: 128 PIO write: 6.857664 sec  116.657799 MB/s
size: 256 PIO write: 6.597888 sec  121.250922 MB/s
size: 512 PIO write: 6.458101 sec  123.875423 MB/s
size: 1024 PIO write: 6.404411 sec  124.913905 MB/s
size: 2048 PIO write: 6.397210 sec  125.054514 MB/s
size: 4096 PIO write: 6.387041 sec  125.253617 MB/s
size: 8192 PIO write: 6.390173 sec  125.192230 MB/s
size: 16384 PIO write: 6.384816 sec  125.297269 MB/s
```

write combining モード設定後 (8 レーン) :

```
kawai@localhost[10]>./hibtest 13 1
# hib[0] PIO write (host -> HIB)
size: 64 PIO write: 2.037641 sec  392.610858 MB/s
size: 128 PIO write: 1.233335 sec  648.647763 MB/s
size: 256 PIO write: 0.822831 sec  972.253211 MB/s
size: 512 PIO write: 0.639186 sec  1251.591587 MB/s
size: 1024 PIO write: 0.620417 sec  1289.455073 MB/s
size: 2048 PIO write: 0.620460 sec  1289.365885 MB/s
size: 4096 PIO write: 0.620398 sec  1289.495211 MB/s
size: 8192 PIO write: 0.620425 sec  1289.438721 MB/s
size: 16384 PIO write: 0.620416 sec  1289.457550 MB/s
```

#### 4.2.5 ユーザライブラリの使用方法

ユーザライブラリはユーザプログラムに対して HIB 制御用の API を提供します。ユーザライブラリを使用するには、ユーザプログラムに `include/hibutil.h` をインクルードし、実効ファイル生成時にライブラリ `lib/libhib.a` をリンクしてください。

ユーザライブラリの提供する API のうち主要なものを以下で説明します。説明の無い API については `hibutil/hibutil.[hc]` を参照してください。

**Hib\* hib\_openMC(int devid)** は識別子 `devid` を持つ HIB の使用権限を取得します。使用権限が他のプロセスによって取得されている場合には、権限を得られるまでブロックします。`devid` は各 HIB 回路ごとにユニークに与えられる小さな整数です。ホスト計算機に  $n$  個の HIB がインストールされている場合には、それぞれの HIB に識別子  $0$  から  $n - 1$  が割り当てられます。

**hib\_openMC()** は成功すると `Hib` 型構造体へのポインタを返します。この構造体には HIB に関する情報がまとめられており、必要に応じて他の API が利用します (*cf.* `hib_dmawMC`)。

**void hib\_closeMC(int devid)** は識別子 `devid` を持つ HIB の使用権限を破棄します。使用権限を破棄した HIB は、他のプロセスから使用できるようになります。

**void hib\_piowMC(int devid, int size, UINT64 \*buf)** は識別子 `devid` を持つ HIB に対して、`buf` でポイントされるアドレスから始まる ( $\text{size} * 8$ ) byte に格納されているデータを書き込みます。`buf` にはユーザ空間上で静的に確保した配列や、`malloc()` を用いて動的に確保した領域など、通常のメモリ領域を指定できます。

**void hib\_start\_dmawMC(int devid, int size, UINT64 \*buf)** は識別子 `devid` を持つ HIB に対して、`buf` でポイントされるアドレスから始まる ( $\text{size} * 8$ ) byte へデータを書き込むよう要求します。

ここで、`buf` には任意のメモリ領域を指定することはできない点に注意してください。`buf` として指定できるのは `h->dmaw_buf`、あるいは `h->dmaw_buf` にオフセットを加えた `h->dmaw_buf + offset` のみです。また `offset + size` の値は 32k byte を超えることはできません。ただし `h` は `hib_openMC()` が返す `Hib` 型構造体へのポインタです。`h->dmaw_buf` は Linux のカーネル空間内に確保された 32k byte の連続領域を、ユーザ空間へマップしたものです。

通常のメモリ領域 (ユーザ空間上で静的に確保した配列や、`malloc()` など用いて動的に確保した領域) へ HIB から受け取ったデータを格納するためには、いったん HIB から `h->dmaw_buf` へデータを受け取り、それをコピーしてください。

**int hib\_finish\_dmawMC(int devid)** は `hib_start_dmawMC()` で発行した書き込み要求の終了を待ちます。

**UINT32 hib\_config\_readMC(int devid, UINT32 addr)** は識別子 `devid` を持つ HIB の、PCI Configuration Register アドレス `addr` 番地の値を読み出します。

**void hib\_config\_writeMC(int devid, UINT32 addr, UINT32 value)** は識別子 `devid` を持つ HIB の、PCI Configuration Register アドレス `addr` 番地へ値 `value` を書き込みます。

UINT32 hib\_mem\_readMC(int devid, UINT32 addr) は識別子 devid を持つ HIB の、Local Register アドレス addr 番地の値を読み出します。Local Register のアドレスマップについては template/hibctl.vhd を参照してください。

void hib\_mem\_writeMC(int devid, UINT32 addr, UINT32 value) は識別子 devid を持つ HIB の、Local Register アドレス addr 番地へ値 value を書き込みます。

### ユーザライブラリの使用例

ユーザライブラリを使用したアプリケーションプログラムの例が sample/loopback.c にあります。このプログラムはループバック転送のテストを行います。10 \* 8 byte のデータを PIO write によってホスト計算機から HIB へ送信し、そのデータをそのまま DMA write によって回収します。送信したデータと回収したデータが完全に一致した場合には OK を、一致しない場合には NG を出力します。

```
kawai@localhost[9]> ./loopback
0x0000 sent : 0x123456789abc0000 received : 0x123456789abc0000 OK
0x0001 sent : 0x123456789abc0001 received : 0x123456789abc0001 OK
0x0002 sent : 0x123456789abc0002 received : 0x123456789abc0002 OK
0x0003 sent : 0x123456789abc0003 received : 0x123456789abc0003 OK
0x0004 sent : 0x123456789abc0004 received : 0x123456789abc0004 OK
0x0005 sent : 0x123456789abc0005 received : 0x123456789abc0005 OK
0x0006 sent : 0x123456789abc0006 received : 0x123456789abc0006 OK
0x0007 sent : 0x123456789abc0007 received : 0x123456789abc0007 OK
0x0008 sent : 0x123456789abc0008 received : 0x123456789abc0008 OK
0x0009 sent : 0x123456789abc0009 received : 0x123456789abc0009 OK
```

## 5 より高度な使用方法

本節では GPCle のより高度な使用方法について説明します。5.1 節では回路の変更方法について説明します。5.2 節ではより高速に動作する回路の生成方法を説明します。5.3 節では内蔵トランシーバの代わりに外付けの PHY チップを使用する方法を説明します。5.4 節では HIB を介さずに GPCle エンジンを直接制御する方法について説明します。5.5 節では割込みの使用方法を説明します。

### 5.1 ソースコードのアップデート

GPCle のソースコードは template/ ディレクトリ内の多数のファイルに分割されて記述されています。これらのうちエンティティ hib の依存しているファイルを使用上の便宜からひとつのファイルにまとめたものが hib.vhd です。同様に、エンティティ gpcie と phy pcs の依存しているファイルをそれぞれひとつにまとめたものが、gpcie.vhd と phy.vhd です。

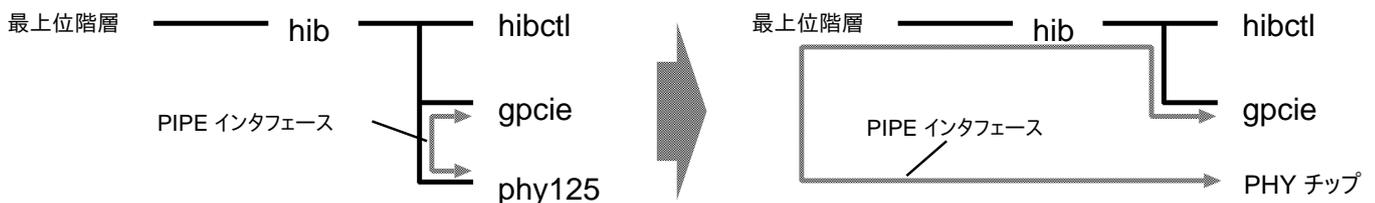
ソースコードに変更を加えた場合には、gpciepkg のルートディレクトリで make を実行してください。ソースコードの変更が hib.vhd、gpcie.vhd、phy.vhd へ反映されます。

## 5.2 より高速に動作する回路の生成

開発キットのソースコード中には、より高速に動作する回路が用意されています。この回路は、特に Arria GX を用いる場合には、タイミング制約を満たすことが困難なので、初期設定では無効化されています。この回路を生成するには gpciepkg のルートディレクトリで make fast を実行します。hib0.vhd、gpcie0.vhd が生成されるので、これらを従来の hib.vhd、gpcie.vhd と置き換えて使用してください。この回路を用いると、DMA write 転送の実効性能が従来版に比べて 20% 程度向上する可能性があります。PIO write 転送や DMA read 転送の性能は変わりません。

## 5.3 PHY チップの使用方法

FPGA に内蔵のトランシーバの代わりに外付けの PHY チップを使用するには、エンティティ hib を以下に示すように変更する必要があります (エンティティ hib は template/hibtop.vhd 内で定義されています)。なお GPCle DS および GPCle SP パッケージには、この変更を行った回路が含まれています。



エンティティ hib はその内部で 3 つのエンティティ hibctl、gpcie、phypcs のインスタンスを使用しています。これらに対して 2 つの変更が必要です。まずインスタンス phypcs を削除します。インスタンス phypcs は内蔵トランシーバの PHY PCS 層と PHY PMA 層を実装しています。これらの層は PHY チップによって実装されるため、phypcs は不要になります。この変更にともない、phypcs と gpcie との間の PIPE インタフェースによる接続も削除されます。

次にインスタンス gpcie の PIPE インタフェースを PHY チップの PIPE インタフェースと接続します。このためには PHY チップの PIPE インタフェースを FPGA の I/O ピンから回路の最上位階層 (サンプル回路を使用する場合には ifpga\_xxx{8,4,1}.vhd 内で定義されているエンティティ ifpga) の port に接続し、さらにそれをエンティティ hib へと引き入れる必要があります。

## 5.4 GPCle エンジンを直接制御する使用方法

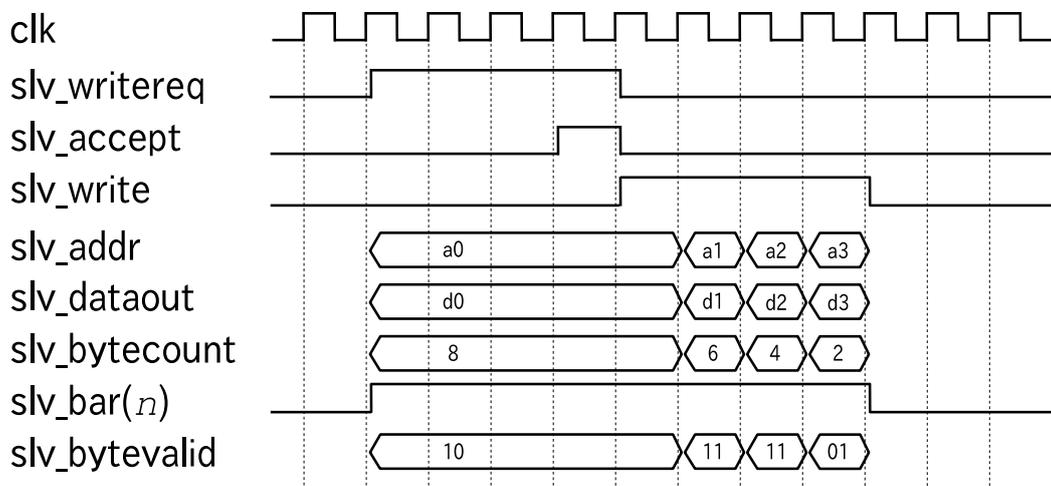
HIB はユーザ回路に対して簡便なインタフェースを提供しますが、HIB インタフェースから GPCle のサポートする機能すべてを利用することはできません。例えば以下の機能を利用するためには、

ユーザ回路が HIB を介さずに GPCle エンジンに直接制御する必要があります。

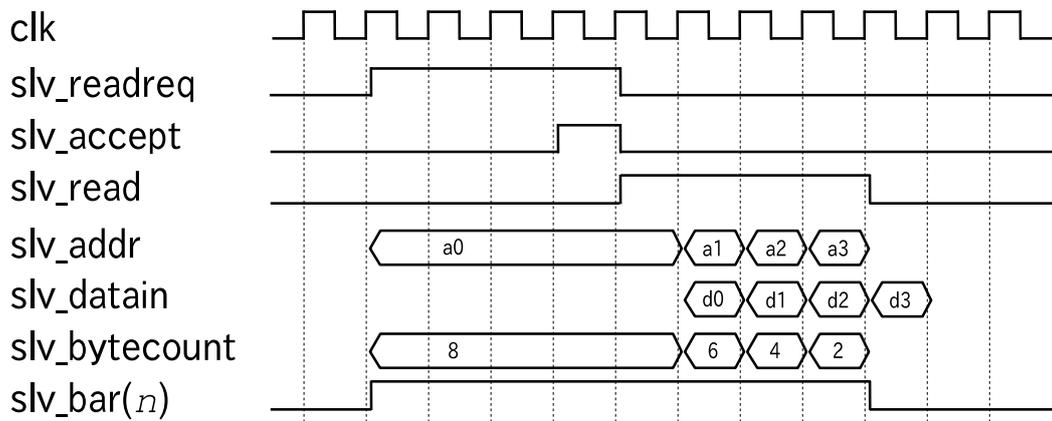
- アドレスやバイトイネーブル、ウェイトつきの PIO read/write 転送。
- ユーザ独自の BAR0~BAR5 空間定義。
- DMA チャンネルの追加・削除。
- 割込みの発行 (節 5.5)。

ユーザ回路から GPCle エンジンに直接制御するには、`gpci.vhd` 内で定義されているエンティティ `gpcie` をユーザの回路内にインスタンス化します。FPGA 内蔵 Gbit トランシーバを用いる場合には、`phy.vhd` 内で定義されているエンティティ `phypcs` もインスタンス化し、両者の PIPE インタフェースを接続します。

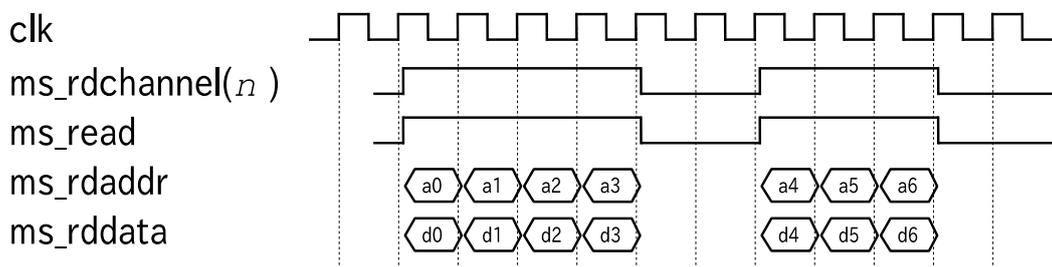
GPCle エンジンとバックエンド回路と間の転送には、slave read/write および master read/write の 2 通りの方法が用意されています。slave read/write 転送は、ホスト計算機からの Programm I/O read/write 要求に従って、GPCle エンジンがバックエンドに対して行う read/write 転送です。master read/write 転送は、DMA コントローラからの要求に従って、バックエンドが GPCle に対して行う read/write 転送です。これら 4 通りの転送と、DMA コントローラ制御のタイミングチャートを以下に示します。各信号の意味については第 6 章を参照してください。



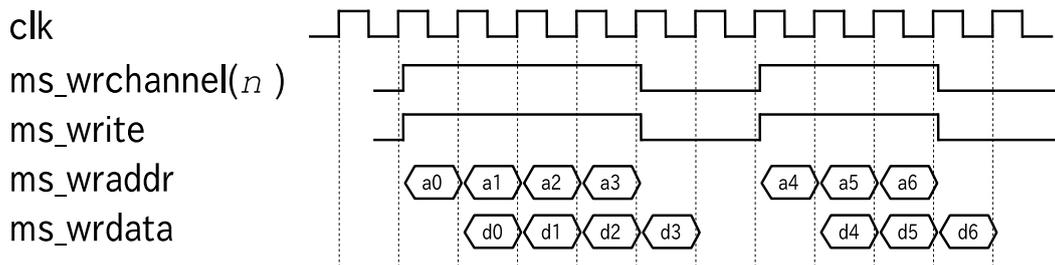
Slave write (GPCle エンジンによるバックエンド回路への書き込み): GPCle エンジンからの書き込み要求 (`slv_writereq`) に対してバックエンド回路が `slv_accept` をアサートすると、書き込みが開始されます。データの移動方向はホスト計算機 → PCIe デバイス。



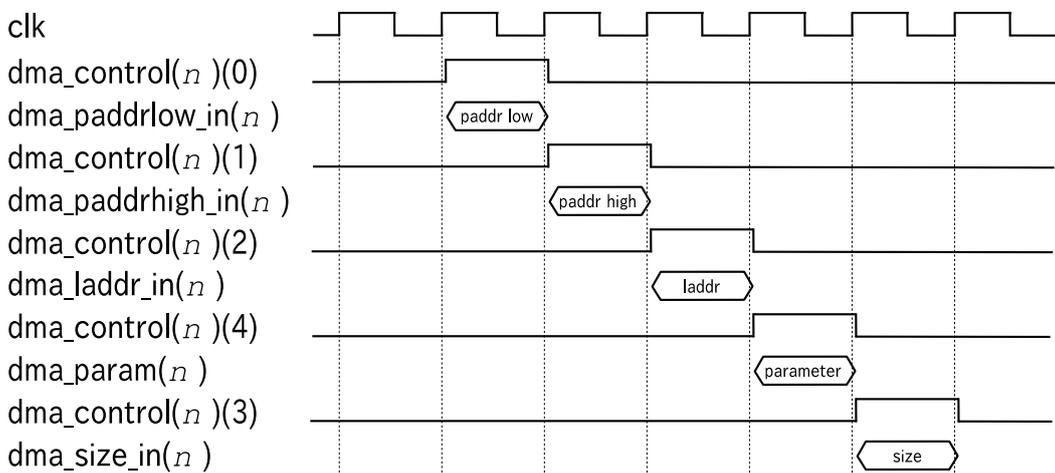
Slave read (GPCle エンジンによるバックエンド回路からの読み出し): GPCle エンジンからの読み出し要求 (slv\_readreq) に対してバックエンド回路が slv\_accept をアサートすると、読み出しが開始されます。データの移動方向は PCIe デバイス → ホスト計算機。



Master read (バックエンド回路による GPCle エンジンからの読み出し): データの移動方向はホスト計算機 → PCIe デバイス。



Master write (バックエンド回路による GPCle エンジンへの書き込み): データの移動方向は PCIe デバイス → ホスト計算機。



DMA コントローラの制御: DMA コントローラの各レジスタに値を設定します。dma\_control(n)(3) をアサートして転送長を設定すると同時に、DMA 転送が開始されます ( $n$  は DMA チャンネル番号)。

## 5.5 割込みの使用方法

GPCle は INT<sub>x</sub> 方式および MSI 方式による割込みをサポートしています (割り込みは GPCle SP, GPCle DS のみで対応しています。GPCle 機能限定版は非対応です)。

ユーザ回路が割込みを発生させるにはエンティティ `gpcie` の下記の 2 種のポートを使います。

```
int_req      : in  std_logic;
int_ack     : out std_logic;
```

ユーザ回路は int\_req をアサートし、GPCIe に対して割込みの生成を要求してください。GPCIe は準備が整いしだい割り込みを発行し、発行を完了すると int\_ack をアサートします。ユーザ回路は int\_ack がアサートされるまで int\_req をアサートし続け、int\_ack がアサートされたら int\_req をディアサートしてください。GPCIe は int\_req がディアサートされたことを確認して int\_ack をディアサートします。int\_ack がディアサートされるまで、次の割込み要求は発行できません。

割込み方式として INTx、MSI のいずれを使用するかは、エンティティ gpcie の generic 宣言で指定します。以下で指定方法を説明します。

## INTx 方式

- エンティティ gpcie の generic パラメタ CFG\_COMMAND\_INIT の第 10 ビットに 0 を設定します。これにより INTx 方式の割込みが有効になります。
- エンティティ gpcie の generic パラメタ CFG\_INT\_PIN\_INIT により、使用する割り込みピンを指定します。

割込みピン	値
INTA	0000_0001
INTB	0000_0010
INTC	0000_0011
INTD	0000_0100
使用しない	0000_0000

- ブート時に、RootComplex によって割込み番号 (IRQ) が割り当てられます。割り当てられた番号は PCI Configuration Register の該当アドレス (3Ch) に設定されます。

## MSI 方式

MSI は従来の INTx よりも自由度の高い割込み方式です。割込み発生時に RootComplex の任意のアドレスに任意のデータを書き込めます。ただし RootComplex が必ずしもこの方式に対応しているとは限りません。

MSI 方式の割込みを使用するには、エンティティ gpcie の generic パラメタ CFG\_COMMAND\_INIT の第 10 ビットに 1 を設定します。これにより INTx 方式の割込みが無効になり、RootComplex が対応している場合には、MSI 方式を使用するようになります。

RootComplex が MSI 方式に対応している場合には、ブート時に RootComplex によってデバイスの PCI Configuration Register の所定のビットに値 1 が設定されます。「所定のビット」とは MSI Capability Register Structure 内の MSI Control Register の第 0 ビットを指します。GPCIe の場合には、このビットは PCI Configuration Register のバイトアドレス 52h の最下位ビットに配置されています。

割込み発生時に RootComplex に対して発行する書き込みのアドレスとデータは MSI Capability Register Structure 内の MSI Address と MSI Data で指定します。GPCIe の場合には、これらの領域は PCI Configuration Register のバイトアドレス 54h-5Bh に配置されています。

GPCIe の MSI Capability Register Structure の配置は下表の通りです。

PCI Conf. Reg. byte address	MSI Capability Structure			
	31:24	23:16	15:08	07:00
50	MSI Control Register		next pointer(00h)	capability ID(05h)
54	MSI Address (32 bit)			
58	0000h		MSI Data (16 bit)	

## 6 エンティティ詳細

VHDL エンティティ `hib`、`gpcie`、`phypcs` は多くの generic パラメタと port を持ちます。以下では特に重要な generic パラメタと、すべての port について説明します。

パラメタの中には設定を誤ると PCI Express デバイスとして正しく動作しなくなるものや、転送速度に影響を与えるもの、回路資源の消費量に影響を与えるもの、回路の動作周波数に影響を与えるものがあります。パラメタの意味を理解せずに変更することはお薦めできません。

### 6.1 エンティティ `hib` 詳細

#### 6.1.1 ヘッダ部の記述

標準的なライブラリの他に、パッケージ `gpciepkg` を use する必要があります。パッケージ `gpciepkg` は `gpciepkg.vhd` 内で定義されています。

記述例：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.gpciepkg.all;
```

#### 6.1.2 Generic 宣言

---

パラメタ名：	DEVICE
型：	string
デフォルト値：	"Arria GX"
機能：	ターゲットとする FPGA デバイスを指定します。"Arria GX" もしくは "Stratix II GX" を指定してください。

---

パラメタ名 : GENERATION  
型 : natural  
デフォルト値 : 1

機能 : コアのサポートする PCI Express の動作周波数を指定します。1 は 2.5GHz (Gen1) を、2 は 5.0GHz (Gen2) を意味します。

---

パラメタ名 : NLANE  
型 : natural  
デフォルト値 : -

機能 : PCI Express のリンク幅を指定します。1, 4, 8 のいずれかの値を指定してください。

---

パラメタ名 : PIOBUF\_DEPTH  
型 : natural  
デフォルト値 : 8

機能 : PIO write バッファの深さ。デフォルト値は 256 ( $= 2^8$ ) word です。リンク幅 1, 4, 8 レーンの場合のサイズはそれぞれ 512, 2k, 4k byte です。この値は PIO write 性能に影響を与えますが、変更の必要はまれでしょう。

---

パラメタ名 : TXBUF\_DEPTH  
型 : natural  
デフォルト値 : 10

機能 : backend\_data 回収用バッファの深さ。デフォルト値は 1024 word ( $= 2^{10}$ ) です。リンク幅 1, 4, 8 レーンの場合のサイズは、それぞれ 2k, 8k, 16k byte です。1 回の DMA write 転送でこのサイズを越えるデータ転送を行う必要がある場合には、バッファのオーバフローを防ぐための機構をバックエンド回路側に実装してください。

---

パラメタ名 : USE\_CLK32

型 : natural

デフォルト値 : 1

機能 : 0 を設定すると clk32 を使わない動作を試みます。0 を設定した場合には動作が不安定になる可能性があるため、通常は 1 (デフォルト値) を設定してください。何らかの制限により clk32 を供給できない場合にのみ 0 を設定してください。

---

---

### 6.1.3 Port 宣言

---

---

ポート名 : phy\_linkup

型 : std\_logic

方向 : out

機能 : PCIe 物理層で link が確立すると assert されます。LED などの外部出力へ接続し、リンクのインジケータとして使用できます。

---

ポート名 : dl\_linkup

型 : std\_logic

方向 : out

機能 : PCIe datalink 層の初期化が完了すると assert されます。LED などの外部出力へ接続し、リンクのインジケータとして使用できます。

---

ポート名 : linkspeed

型 : std\_logic\_vector(3 downto 0)

方向 : out

機能 : 実際に確立された PCI Express の動作周波数を指定します。"0001" は 2.5GHz (Gen1) を、"0010" は 5.0GHz (Gen2) を意味します。

---

ポート名 : clk100\_ext  
型 : std\_logic  
方向 : in  
機能 : Gbit トランシーバ用のリファレンスクロックです。  
100MHz のディファレンシャル信号を入力してください。

---

ポート名 : clk32  
型 : std\_logic  
方向 : in  
機能 : トランシーバの power on reset のタイミングを生成するために使用するクロック信号です。トランシーバのキャリブレーションにも使用します。10MHz-125MHz の任意の周波数の信号を入力してください。

---

ポート名 : mperst  
型 : std\_logic  
方向 : in  
機能 : active low のリセット線です。

---

ポート名 : rx\_in  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : in  
機能 : PCI Express 高速シリアル信号の受信ポートです。

---

ポート名 : tx\_out  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : out  
機能 : PCI Express 高速シリアル信号の送出ポートです。

---

ポート名 : clk\_out  
型 : std\_logic  
方向 : out  
機能 : PHY PCS 層で clk100 から生成される 125MHz のパラレルインタフェースクロックです。HIB のすべてのパラレル信号はこのクロックに同期します。

---

ポート名 : wake  
型 : std\_logic  
方向 : out  
機能 : 未使用。

---

ポート名 : hib\_we  
型 : std\_logic  
方向 : out  
機能 : HIB からバックエンド回路への write 信号です。この信号に同期してデータが hib\_data からバックエンド回路へ出力されます。

---

ポート名 : hib\_data  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : out  
機能 : HIB からバックエンド回路へのデータ出力です。

---

ポート名 : backend\_we  
型 : std\_logic  
方向 : in  
機能 : バックエンド回路から HIB への write 信号です。この信号に同期してデータが backend\_data から HIB へ出力されます。

---

ポート名 : backend\_data  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : in  
機能 : バックエンド回路から HIB へのデータ出力です。

---

ポート名 : reset\_backend  
型 : std\_logic  
方向 : out  
機能 : バックエンド回路へのリセット信号です (active high)。

---

ポート名 : board\_info  
型 : std\_logic\_vector(31 downto 0)  
方向 : in  
機能 : ホスト計算機から読み書き可能なメールアドレスレジスタ board\_info の初期値を与えます。このレジスタはバックエンド回路が任意の目的に使用できます。

---

---

## 6.2 エンティティ gpcie 詳細

### 6.2.1 ヘッダ部の記述

標準的なライブラリの他に、パッケージ gpciepkg を use する必要があります。パッケージ gpciepkg は gpciepkg.vhd 内で定義されています。

記述例 :

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
use work.gpciepkg.all;
```

### 6.2.2 Generic 宣言

エンティティ gpcie の generic パラメタには、デフォルト値として HIB 用に最適化された値が設定されています。用途に応じて設定を変更してください。

---

---

パラメタ名 : NLANE  
型 : natural  
デフォルト値 : 8  
機能 : PCI Express のリンク幅を指定します。1, 4, 8 のいずれかの値を指定してください。

---

パラメタ名 : GENERATION  
型 : natural  
デフォルト値 : 1

機能 : コアのサポートする PCI Express の動作周波数を指定します。1 は 2.5GHz (Gen1) を、2 は 5.0GHz (Gen2) を意味します。

---

パラメタ名 : NDMACH  
型 : natural  
デフォルト値 : 2

機能 : 実装する DMA チャンネル数を指定します。最大 8 チャンネルで動作することを意図して設計されていますが、現時点では 2 チャンネルまででしか動作確認を行っていません。

---

パラメタ名 : MAX\_READ\_REQ\_SIZE  
型 : natural  
デフォルト値 : 256

機能 : max read request size を指定します。単位は byte です。

---

パラメタ名 : MAX\_PAYLOAD  
型 : natural  
デフォルト値 : 256

機能 : この回路でサポートする max payload size の最大値を指定します。単位は byte です。実際の通信に使用される max payload size は上流デバイスとのネゴシエーションによって決定されます。

---

パラメタ名 : CA\_PH\_VC0\_INIT  
型 : natural  
デフォルト値 : 16

機能 : Rx Flow Control バッファ (posted, header) の深さを指定します。単位はパケット数です。

---

パラメタ名 : CA\_PD\_VC0\_INIT  
型 : natural  
デフォルト値 : 64  
機能 : Rx Flow Control バッファ (posted, data) の深さを指定します。単位は 16byte です。

---

パラメタ名 : CA\_NPH\_VC0\_INIT  
型 : natural  
デフォルト値 : 2  
機能 : Rx Flow Control バッファ (non-posted, header) の深さを指定します。単位はパケット数です。

---

パラメタ名 : CA\_NPD\_VC0\_INIT  
型 : natural  
デフォルト値 : 16  
機能 : Rx Flow Control バッファ (non-posted, data) の深さを指定します。単位は 16byte です。

---

パラメタ名 : CA\_CH\_VC0\_INIT  
型 : natural  
デフォルト値 : 2  
機能 : Rx Flow Control バッファ (completion, header) の深さを指定します。単位はパケット数です。

---

パラメタ名 : CA\_CD\_VC0\_INIT  
型 : natural  
デフォルト値 : 16  
機能 : Rx Flow Control バッファ (completion, data) の深さを指定します。単位は 16byte です。

---

パラメタ名 : CL\_PH\_VC0\_INIT  
型 : natural  
デフォルト値 : 16  
機能 : Tx Flow Control バッファ (posted, header) の深さを指定します。単位はパケット数です。

---

---

パラメタ名 : CL\_PD\_VC0\_INIT  
型 : natural  
デフォルト値 : 64  
機能 : Tx Flow Control バッファ (posted, data) の深さを指定します。単位は 16byte です。

---

パラメタ名 : CL\_NPH\_VC0\_INIT  
型 : natural  
デフォルト値 : 2  
機能 : Tx Flow Control バッファ (non-posted, header) の深さを指定します。単位はパケット数です。

---

パラメタ名 : CL\_NPD\_VC0\_INIT  
型 : natural  
デフォルト値 : 16  
機能 : Tx Flow Control バッファ (non-posted, data) の深さを指定します。単位は 16byte です。

---

パラメタ名 : CL\_CH\_VC0\_INIT  
型 : natural  
デフォルト値 : 2  
機能 : Tx Flow Control バッファ (completion, header) の深さを指定します。単位はパケット数です。

---

パラメタ名 : CL\_CD\_VC0\_INIT  
型 : natural  
デフォルト値 : 16  
機能 : Tx Flow Control バッファ (completion, data) の深さを指定します。単位は 16byte です。

---

パラメタ名 : CFG\_VENDOR\_ID\_INIT  
型 : std\_logic\_vector(15 downto 0)  
デフォルト値 : x"1b1a"  
機能 : KFCR 社のベンダ ID です。変更しないでください (利用許諾を参照)。

---

パラメタ名 : CFG\_DEVICE\_ID\_INIT  
型 : std\_logic\_vector(15 downto 0)  
デフォルト値 : x"0e70"  
機能 : デバイス ID です。0e70h は KFCR 社が HIB に対して割り当てた値です。

---

パラメタ名 : CFG\_REVISION\_ID\_INIT  
型 : std\_logic\_vector(7 downto 0)  
デフォルト値 : x"01"  
機能 : リビジョン ID です。

---

パラメタ名 : CFG\_CLASS\_CODE\_INIT  
型 : std\_logic\_vector(23 downto 0)  
デフォルト値 : x"ff0000"  
機能 : PCI クラスコードです。

---

パラメタ名 : CFG\_BAR0\_INIT  
型 : std\_logic\_vector(31 downto 0)  
デフォルト値 : x"ffff8008"  
機能 : PCI Base Address Register0 (BAR0) の初期値を指定します。デフォルト値 x"ffff8008" は 32kbyte, prefetchable, 32-bit address, memory space を意味します。

---

パラメタ名 : CFG\_BAR1\_INIT  
型 : std\_logic\_vector(31 downto 0)  
デフォルト値 : x"fffff008"  
機能 : PCI Base Address Register1 (BAR1) の初期値を指定します。デフォルト値 x"fffff008" は 4kbyte, prefetchable, 32-bit address, memory space を意味します。

---

パラメタ名 : CFG\_BAR2\_INIT  
型 : std\_logic\_vector(31 downto 0)  
デフォルト値 : x'ffff8008"  
機能 : PCI Base Address Register2 (BAR2) の初期値を指定します。デフォルト値 x'ffff8008" は 32kbyte, prefetchable, 32-bit address, memory space を意味します。

---

パラメタ名 : CFG\_BAR3\_INIT  
型 : std\_logic\_vector(31 downto 0)  
デフォルト値 : x'00000000"  
機能 : PCI Base Address Register3 (BAR3) の初期値を指定します。デフォルト値 x'00000000" は「未使用」 を意味します。

---

パラメタ名 : CFG\_BAR4\_INIT  
型 : std\_logic\_vector(31 downto 0)  
デフォルト値 : x'00000000"  
機能 : PCI Base Address Register4 (BAR4) の初期値を指定します。デフォルト値 x'00000000" は「未使用」 を意味します。

---

パラメタ名 : CFG\_BAR5\_INIT  
型 : std\_logic\_vector(31 downto 0)  
デフォルト値 : x'00000000"  
機能 : PCI Base Address Register5 (BAR5) の初期値を指定します。デフォルト値 x'00000000" は「未使用」 を意味します。

---

パラメタ名 : CFG\_BAR\_ROM\_INIT  
型 : std\_logic\_vector(31 downto 0)  
デフォルト値 : x'00000000"  
機能 : PCI Expansion ROM Base Address の初期値を指定します。デフォルト値 x'00000000" は「未使用」 を意味します。

---

パラメタ名 : CFG.SUB.VENDOR.ID.INIT  
型 : std\_logic\_vector(15 downto 0)  
デフォルト値 : x"1b1a"  
機能 : サブベンダ ID です。

---

パラメタ名 : CFG.SUB.DEVICE.ID.INIT  
型 : std\_logic\_vector(15 downto 0)  
デフォルト値 : x"0e70"  
機能 : サブデバイス ID です。

---

パラメタ名 : CFG.INT.PIN.INIT  
型 : std\_logic\_vector(7 downto 0)  
デフォルト値 : x"00"  
機能 : INTx 方式の割込みに使用する割り込みピンを指定します。使用法は節 5.5 を参照してください。

---

---

### 6.2.3 Port 宣言

---

ポート名 : phy\_linkup  
型 : std\_logic  
方向 : out  
機能 : 物理層でリンクトレーニングが終了し、リンクが確立されると assert されます。

---

ポート名 : dl\_linkup  
型 : std\_logic  
方向 : out  
機能 : データリンク層の初期化が終了し、通常の通信を行える状態に遷移すると assert されます。

---

ポート名 : linkspeed  
型 : std\_logic\_vector(3 downto 0)  
方向 : out  
機能 : 実際に確立された PCI Express の動作周波数を指定します。"0001" は 2.5GHz (Gen1) を、"0010" は 5.0GHz (Gen2) を意味します。

---

ポート名 : clk  
型 : std\_logic  
方向 : in  
機能 : PHY PCS 層から供給される 125MHz のパラレルインタフェースクロックを入力します。PIPE インタフェースを含むすべてのパラレル信号はこのクロックに同期します。

---

ポート名 : rstn  
型 : std\_logic  
方向 : in  
機能 : リセット線です (active low)。

---

---

## PIPE インタフェース

---

---

ポート名 : phystatus  
型 : std\_logic  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : powerdown  
型 : std\_logic\_vector(1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txdetectrx  
型 : std\_logic  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

---

ポート名 : txdata  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txdatak  
型 : std\_logic\_vector(NLANE\*2-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txelecidle  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txcompl  
型 : std\_logic\_vector(NLANE-1 downto 0);  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxpolarity  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxdata  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxdatak  
型 : std\_logic\_vector(NLANE\*2-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxvalid  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxelecidle  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxstatus  
型 : std\_logic\_vector(NLANE\*3-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

---

#### アプリケーションインタフェース (スレーブとしてのインタフェース)

---

---

ポート名 : slv\_readreq  
型 : std\_logic  
方向 : out  
機能 : read 要求信号です。要求に対して slv\_accept を assert すると、assert 直後のクロックから read 転送が開始されます。

---

ポート名 : slv\_writereq  
型 : std\_logic  
方向 : out  
機能 : write 要求信号です。要求に対して slv\_accept を assert すると、assert 直後のクロックから write 転送が開始されます。

---

ポート名 : slv\_accept  
型 : std\_logic\_vector(downto 0)  
方向 : in  
機能 : アクセス許可信号です。

---

---

ポート名 : slv\_read  
型 : std\_logic  
方向 : out  
機能 : read 信号です。ユーザ回路はこの信号の次のクロックで slv\_datain へデータを供給してください。

---

ポート名 : slv\_write  
型 : std\_logic  
方向 : out  
機能 : write 信号です。この信号に同期してデータが slv\_dataout からユーザ回路へ出力されます。

---

ポート名 : slv\_bar  
型 : std\_logic\_vector(6 downto 0)  
方向 : out  
機能 : 現在アクセスを受けている base address 空間を示します。

---

ポート名 : slv\_addr  
型 : std\_logic\_vector(63 downto 0)  
方向 : out  
機能 : 現在アクセスを受けているローカルアドレスを示します。

---

ポート名 : slv\_bytevalid  
型 : std\_logic\_vector(NLANE\*2-1 downto 0)  
方向 : out  
機能 : slv\_dataout の byte enable 信号です。write アクセス時のみ使用します。

---

ポート名 : slv\_bytecount  
型 : std\_logic\_vector(11 downto 0)  
方向 : out  
機能 : 転送の残りバイト数です。現在のクロックサイクルにおける転送バイト数は残りバイト数には含まれません。従って転送の最後のクロックサイクルでは必ず 0 が出力されます。

---

ポート名 : slv\_dataout  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : out  
機能 : GPCle からのデータ出力です。

---

ポート名 : slv\_datain  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : in  
機能 : GPCle へのデータ入力です。

---

---

#### アプリケーションインタフェース (マスタとしてのインタフェース)

---

---

ポート名 : ms\_wrchannel  
型 : std\_logic\_vector(NDMACH-1 downto 0)  
方向 : out  
機能 : 現在 DMA write アクセスを受けている DMA チャンネルを示します。

---

ポート名 : ms\_write  
型 : std\_logic  
方向 : out  
機能 : DMA write 信号です。ユーザ回路はこの信号の次のクロックで ms\_wrdata へデータを供給してください。

---

ポート名 : ms\_wraddr  
型 : std\_logic\_vector(31 downto 0)  
方向 : out  
機能 : 現在 DMA write アクセスを受けているローカルアドレスを示します。

---

ポート名 : ms\_wrdata  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : in  
機能 : GPCle への DMA write データ入力です。

---

ポート名 : ms\_rdchannel  
型 : std\_logic\_vector(NDMACH-1 downto 0)  
方向 : out  
機能 : 現在 DMA read アクセスを受けている DMA チャンネルを示します。

---

ポート名 : ms\_read  
型 : std\_logic  
方向 : out  
機能 : DMA read 信号です。この信号に同期してデータが ms\_rddata からユーザ回路へ出力されます。

---

ポート名 : ms\_rdaddr  
型 : std\_logic\_vector(31 downto 0)  
方向 : out  
機能 : 現在 DMA read アクセスを受けているローカルアドレスを示します。

---

ポート名 : ms\_rddata  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : out  
機能 : GPCle からの DMA read データ出力です。

---

ポート名 : int\_req  
型 : std\_logic  
方向 : in  
機能 : 割込み要求信号です。使用法は節 5.5 を参照してください。

---

ポート名 : int\_ack  
型 : std\_logic  
方向 : out  
機能 : 割込みの発行完了を通知する信号です。使用法は節 5.5 を参照してください。

---

---

## アプリケーションインタフェース (DMA コントロールインタフェース)

第 0 チャンネルから第 NDMACH-1 チャンネルまでの各 DMA チャンネルに、以下で説明するインタフェースが一組ずつ提供されます。例えば第  $n$  チャンネルの `dma_control` 信号は `dma_control(n)` (6 downto 0) です。なお 2 次元配列型 `each7b`, `each16b`, `each4b` 等はパッケージ `gpciepkg` 内で定義されています。

---

ポート名 : `dma_control`  
型 : `each7b(NDMACH-1 downto 0)`  
方向 : `in`  
機能 : DMA レジスタ の制御信号です。  
`dma_control(n)(0)` : `dma_paddr_low_in(n)` の write 信号です。  
`dma_control(n)(1)` : `dma_paddr_high_in(n)` の write 信号です。  
`dma_control(n)(2)` : `dma_laddr_in(n)` の write 信号です。  
`dma_control(n)(3)` : `dma_size_in(n)` の write 信号です。  
write と同時に DMA 転送を開始します。  
`dma_control(n)(4)` : `dma_param_in(n)` の write 信号です。  
`dma_control(n)(6)` : 1 クロックのパルスを入力すると DMA 転送を強制終了します。

---

ポート名 : `dma_param`  
型 : `each16b(NDMACH-1 downto 0)`  
方向 : `in`  
機能 : DMA 転送パラメタを設定します。  
`dma_param(n)(7 downto 0)` : 未使用  
`dma_param(n)(8)` : 転送方向を設定します。  
0 : read (ホスト計算機からの読み出し)  
1 : write (ホスト計算機への書き込み)  
`dma_param(n)(15 downto 9)` : 未使用

---

ポート名 : `dma_status`  
型 : `each4b(NDMACH-1 downto 0)`  
方向 : `out`  
機能 : DMA 転送の状態を示します。  
`dma_status(n)(2 downto 0)` : 未使用  
`dma_status(n)(3)` : DMA 転送の完了を示すフラグです。  
0 : 転送中  
1 : 転送終了

---

ポート名 : dma\_fifocnt  
型 : each13b(NDMACH-1 downto 0)  
方向 : in  
機能 : DMA 転送の状態を示します。  
DMA write 時 : 送信用バッファに溜ったデータの byte 数を入力します。  
DMA read 時 : 受信用バッファの空き byte 数を入力します。  
フロー制御が不要 (つまり常時データを送受信可能) な場合には "1 0000 0000 0000" を入力します

---

ポート名 : dma\_paddr\_low\_in  
型 : each32b(NDMACH-1 downto 0)  
方向 : in  
機能 : 転送開始アドレスの下位 32-bit (PCI アドレス空間) を入力します。

---

ポート名 : dma\_paddr\_high\_in  
型 : each32b(NDMACH-1 downto 0)  
方向 : in  
機能 : 転送開始アドレスの上位 32-bit (PCI アドレス空間) を入力します。

---

ポート名 : dma\_laddr\_in  
型 : each32b(NDMACH-1 downto 0)  
方向 : in  
機能 : 転送開始アドレス (ローカルアドレス空間) を入力します。

---

ポート名 : dma\_size\_in  
型 : each32b(NDMACH-1 downto 0)  
方向 : in  
機能 : 転送データサイズを入力します。単位は byte です。

---

ポート名 : dma\_paddr\_low\_out  
型 : each32b(NDMACH-1 downto 0)  
方向 : out  
機能 : 現在転送中のアドレスを示します (PCI アドレス空間、下位 32-bit)。

---

ポート名 : dma\_paddrhigh\_out  
型 : each32b(NDMACH-1 downto 0)  
方向 : out  
機能 : 現在転送中のアドレスを示します (PCI アドレス空間、上位 32-bit)。

---

ポート名 : dma\_laddr\_out  
型 : each32b(NDMACH-1 downto 0)  
方向 : out  
機能 : 現在転送中のアドレスを示します (ローカルアドレス空間)。

---

ポート名 : dma\_size\_out  
型 : each32b(NDMACH-1 downto 0)  
方向 : out  
機能 : 残り転送データサイズを示します。単位は byte です。

---

---

## 6.3 エンティティ phy pcs 詳細

### 6.3.1 ヘッダ部の記述

記述例 :

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;
```

### 6.3.2 Generic 宣言

---

---

パラメタ名 : DEVICE  
型 : string  
デフォルト値 : "Arria GX"  
機能 : ターゲットとする FPGA デバイスを指定します。"Arria GX" もしくは "Stratix II GX" を指定してください。

---

パラメタ名 : GENERATION  
型 : natural  
デフォルト値 : 1

機能 : コアのサポートする PCI Express の動作周波数を指定します。1 は 2.5GHz (Gen1) を、2 は 5.0GHz (Gen2) を意味します。

---

パラメタ名 : NLANE  
型 : natural  
デフォルト値 : -

機能 : PCI Express のリンク幅を指定します。1, 4, 8 のいずれかの値を指定してください。

---

パラメタ名 : USE\_CLK32  
型 : natural  
デフォルト値 : 1

機能 : 0 を設定すると clk32 を使わない動作を試みます。0 を設定した場合には動作が不安定になる可能性があるため、通常は 1 (デフォルト値) を設定してください。何らかの制限により clk32 を供給できない場合にのみ 0 を設定してください。

---

---

### 6.3.3 Port 宣言

---

---

ポート名 : cal\_blk\_clk  
型 : std\_logic  
方向 : in  
機能 : トランシーバのキャリブレーションに使用するクロック信号です。10MHz-125MHz の任意の周波数の信号を入力してください。

---

ポート名 : clk32  
型 : std\_logic  
方向 : in  
機能 : トランシーバの power on reset のタイミングを生成するために使用するクロック信号です。10MHz-125MHz の任意の周波数の信号を入力してください。

---

ポート名 : clk100  
型 : std\_logic  
方向 : in  
機能 : Gbit トランシーバ用のリファレンスクロックです。100MHz のディファレンシャル信号を入力してください。

---

ポート名 : clk125out  
型 : std\_logic  
方向 : out  
機能 : clk100 から生成される 125MHz のパラレルインタフェースクロックです。PHY のすべてのパラレル信号はこのクロックに同期します。

---

ポート名 : clk125plllock  
型 : std\_logic  
方向 : out  
機能 : PLL がロックされ、clk125out が安定して出力されている状態で assert されます。

---

ポート名 : rstn  
型 : std\_logic  
方向 : int  
機能 : リセット線です (active low)。

---

ポート名 : rx.in  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : in  
機能 : PCI Express 高速シリアル信号の受信ポートです。

---

ポート名 : tx\_out  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : out  
機能 : PCI Express 高速シリアル信号の送出ポートです。

---

ポート名 : wake  
型 : std\_logic  
方向 : out  
機能 : 未使用。

---

---

## PIPE インタフェース

---

---

ポート名 : phystatus  
型 : std\_logic  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : powerdown  
型 : std\_logic\_vector(1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txdetectrx  
型 : std\_logic  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txdata  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txdatak  
型 : std\_logic\_vector(NLANE\*2-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txelecidle  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : txcompl  
型 : std\_logic\_vector(NLANE-1 downto 0);  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxpolarity  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : in  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxdata  
型 : std\_logic\_vector(NLANE\*16-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxdatak  
型 : std\_logic\_vector(NLANE\*2-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxvalid  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxelecidle  
型 : std\_logic\_vector(NLANE-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

ポート名 : rxstatus  
型 : std\_logic\_vector(NLANE\*3-1 downto 0)  
方向 : out  
機能 : PIPE インタフェースの仕様を参照してください。

---

---

## 7 利用許諾

### 7.1 GPCle DS, GPCle SP 利用許諾

本ソフトウェアおよび関連文書のファイルの複製 (以下「ソフトウェア」) を取得した者に対し、以下の利用条件のもとでの利用を許可します。

利用条件:

- 本ソフトウェアに起因するいかなる損害に関しても、著作権者に対してその責任を問わないこと。
- 本ソフトウェア利用地域を原則として日本国内に限ること。日本国外での利用を希望する場合には個別に著作権者の許可を得ること。
- 本ソフトウェアおよびその改変物の頒布および販売を行わないこと。ただし、頒布および販売を伴わない範囲での本ソフトウェアの複写および改変と、本ソフトウェアを使用して生成した成果物 (回路のネットリスト、ビットデータ等) の頒布や販売は許可します。
- 本ソフトウェアの複製および改変物もまた本利用許諾に従うこと。
- 上記の著作権表示および本許諾表示を、本ソフトウェアのすべての複製または重要な部分に記載すること。

### 7.2 GPCle 機能限定版 利用許諾

本ソフトウェアおよび関連文書のファイルの複製 (以下「ソフトウェア」) を取得した者に対し、以下の利用条件のもとでの利用を無償で許可します。これには、ソフトウェアの複製を使用、複写、改変、頒布、サブライセンス、または販売する権利、およびソフトウェアを提供する相手に同じことを許可する権利も無制限に含まれます。

利用条件:

- 本ソフトウェアを用いて生成された PCI Express インタフェース回路は、その PCI コンフィグレーションレジスタのアドレス 00h 番地に、初期値として当社のベンダ ID (1B1Ah) が書き込まれている。この値を変更しないこと。
- 本ソフトウェアを用いて得られた成果物 (本ソフトウェアを用いて生成した PCI Express インタフェース回路をその内部で使用している回路データ、またはその回路データを搭載したハードウェア、それらを利用して得られた研究成果) の掲載、頒布、サブライセンス、販売、または公表おこなう際には、成果物が株式会社 K&F Computing Research 製 PCI Express IP コア GPCle の利用によって得られた旨を、成果物本体または取扱説明書、論文等の、成果物の利用者が容易に確認し得る箇所に明記すること。

- 本ソフトウェアに起因するいかなる損害に関しても、著作権者に対してその責任を問わないこと。
- 本ソフトウェア利用地域を原則として日本国内に限ること。日本国外での利用を希望する場合には個別に著作権者の許可を得ること。
- 本ソフトウェアを再頒布 (機能の改変、拡張をおこなったものの頒布も含む) する際には、再頒布物もまた本利用許諾に従うこと。
- 上記の著作権表示および本許諾表示を、本ソフトウェアのすべての複製または重要な部分に記載すること。

日本国外での利用、ベンダ ID を変更しての利用、未実装機能の実装、Linux, Windows 以外のプラットフォームへの対応などについては support@kfcr.jp までご相談ください。

## 8 GPCle 開発キット更新履歴

version	date	description	author(s)
1.4.3	08-Feb-2013	ユーザガイドに割込みの使用法を追記。	AK
1.4.2	07-Apr-2012	Cyclone IV GX 向けに remote update 機能を追加。	AK
1.4.1	14-Jan-2012	Cyclone IV GX 向けに backend_clk 周波数の動的変更機能を追加。	AK
1.4.0	05-Dec-2011	Stratix II GX 向け Gen2 IP コア (これまで別個のパッケージで提供していたもの) を統合。	AK
1.3.3	21-Nov-2011	Altera Cyclone IV GX Dev Kit (soft IP Gen1 x1/x4) をサポート。	AK
1.3.2	02-Mar-2011	Altera Cyclone IV GX Starter Kit (soft IP Gen1 x1) をサポート。	AK
1.3.1	10-Aug-2010	backend_clk 周波数の動的変更機能を追加 (Stratix IV のみ)。	AK
1.3.0	04-Jul-2010	Stratix IV (soft IP Gen1 x1/x4/x8) をサポート。 トレーニングシーケンスに L0s ステートを実装。	AK
1.2.0	06-Mar-2010	GPCle 機能限定版と DS のパッケージを統合。 ディレクトリ構成を変更。	AK
1.1.1	28-Nov-2009	ユーザガイドにタイミングチャートを追記。	AK
1.0	03-Aug-2009	回路、デバイスドライバ等の更新に対応。	AK
0.8.1	31-Dec-2008	ユーザガイドの体裁を変更。	AK
0.8	17-Nov-2008	回路を最適化。ユーザガイド PDF 版を作成。	AK
0.7	09-Oct-2008	x1 をサポート。	AK
0.6	28-Sep-2008	DMA read 機能を追加。	AK
0.5	28-Jul-2008	DMA write 性能を向上。英文書類を追加。	AK
0.1	10-Jul-2008	パッケージ構成を変更。ユーザガイドを改訂。	AK
0.0	09-Jul-2008	初版作成。	A. Kawai

お問い合わせおよびバグレポートは下記まで:  
株式会社 K&F Computing Research (support@kfcr.jp)