第 3 回 ASIC デザインコンテスト 規定課題
「PCI バス インターフェース」

パルテノン研究会[1]

1996/08/16 (Revision 1.4)

[1]文責: NTT 情報通信研究所 永見 康一 (nagami@isl.ntt.jp)

仕様の変更点

Revision 1.2 → Revision 1.3

1. 制御信号 `BusError` を追加した。
2. 双方向信号 `AD`, `nC_BE`, `PAR` に対し、 `ADenb`, `C_BEenb`, `PARenb` 制御信号を付加した。これは、シミュレーション時に、各信号線のドライバが識別できないと、接続回路が記述できないため。

Revision 1.3 → Revision 1.4

1. *PCI* 信号 *RST#* に該当する `input nRST` 信号を加えた。
2. 制御信号 `Ready` について、この信号によってデバイス・コアからの iAccess マスタ要求をイネーブルできるという説明を加えた。
3. `ReadMem`, `ReadMemLine`, `ReadConfig`, `ReadMemReq`, `ReadMemLineReq`, `ReadConfigReq` の各制御信号の第 2 引数として、 `Be` を加えた。
4. 次のように各名称を変更した。

`pci.h`	→	`PCI.h`
`HostBridge`	→	`bridge`

5. コンフィギュレーション・レジスタは、機能回路として記述してもよいこととした。これは、コンフィギュレーション・レジスタの読み取り専用ビットの位置などの特徴が一般にデバイス毎に異なるため、統一的に記述する方法が難しいため。
6. 制御信号 `ReadConfigReq`, `WriteConfigReq` の扱いは実装しなくてもよいこととした。これは、そもそもコンフィギュレーション・レジスタは `PCI` 内部にあるので、デバイス・コアにアクセスする必要はないため。
7. コンフィギュレーション・サイクルのイニシエータ動作は、実装しなくてもよいことにした。すなわち、制御信号 `ReadConfig`, `WriteConfig` の動作の実装は必須ではない。ただし、コンフィギュレーション・サイクルのターゲット動作は実装しなければならない。これに関連して、 `bridge` の PCI インタフェース部分 (図 2 (2) の `PCIb`) には、課題の `PCI` モジュールを用いなくてもよい (この場合、パルテノン研究会が用意した `PCIb` を用います)。

   これは、コンフィギュレーション・サイクルのイニシエータ動作は PCI ブリッジの役目であり、本課題では、少なくとも今回 (第 3 回) は PCI ブリッジの仕様を考慮していないため。
8. コンフィギュレーション・メカニズム 1 の実装を課していたがこれを廃した。これは、 PCI インターフェース回路の動作は、コンフィギュレーション・メカニズムには依存せず、そもそも実装しようがないため。
9. コンフィギュレーション・サイクルのアドレス・フェーズにおける `IDSEL` の連続ステッピングに関して、 `FRAME#` による応答を 1 クロック遅らせる条件を課していたが、この条件を廃した。これは、この `IDSEL` のステッピングについて考慮しなければならないのは、ターゲットではなくイニシエータの方であるため。

はじめに

本稿は、パルテノン研究会が主催する「第 3 回 ASIC デザインコンテスト」の規定課題の一つである、PCI バスインターフェースの仕様を定めるものです。ここでいう PCI とは、 PCI revision 2.0 のうち、パルテノン研究会が定めるサブセットのことを指します。本稿では PCI の仕様の詳細について述べませんので、文献 [1] などを参考にして下さい。

この課題は、ユーザが設計したハードウェア・デバイスを、容易に PCI バスに接続するための汎用インターフェース回路を想定しており、 PCI バス インタフェースを扱う回路をライブラリ・モジュール化するものです (図 1)。

PCI バスのアクセス・インターフェースは正確に定義されたものであり、また他のバス規格と比較して、簡潔にまとめられた理解し易いインターフェースといえるでしょう。しかしながら、周辺デバイスの開発者の立場からすると、バスの根本的な機能であるデータ転送に関係する部分以外に、留意しなければならない繁雑な点があることは否定できません。ユーザ設計デバイスから、このような PCI 独自のインターフェースを直接扱うことは、設計者にとって無用な負担を強いることになります。

そこで、データ転送のみに注目した、単純なアクセス・インターフェース (`iAccess` と名付けました) を定めておき、このインターフェースを PCI インターフェース (`iPCI` と呼ぶことにします) に変換する汎用的な回路があれば、設計者は PCI のインターフェースを実装する必要はなく、負担は大きく軽減されます。本課題は、この変換回路の部分に該当します。

1 課題の概要

図 2 の (1) を見て下さい。 `dlx` という名の DLX [2] 準拠マイクロ・プロセッサと内部キャッシュ回路を含む CPU 、および外部メモリがローカル・バスに接続されています。 CPU とメモリ間のデータ転送などは、本稿によって定義される `iAccess` という名のデータ転送インターフェースによって行なわれます。この、最も簡単な計算機システム `system1` の SFL 記述と、このシステム上で走行する例題プログラムを パルテノン研究会が用意します。例題プログラムに関しては、 `SECONDS` のシミュレーション・スクリプトの形にして `SECONDS` 上で実行シミュレーションができるようにします。

次に、図 2 の (2) を見て下さい。先ほどの CPU と外部メモリが、`PCI` という名の PCI バス・インターフェース回路を介してバス接続されています。 `PCIbus` という名の回路が、バスの接続や、バス使用権のアービトレーションなどを行ないます。さらに、 `bridge` という名の回路が `PCIb` を介してバスに接続されています。この計算機システムでは、 CPU とメモリの間の `iAccess` インターフェースが、`PCI` によって PCI インターフェース (図では `iPCI` と表記されています) に変換され、PCI バスを介してデータ転送が行なわれます。 `bridge` は、 PCI バスのコンフィギュレーションサイクルなどを行ないます。この計算機システム `system2` も パルテノン研究会が用意します。

本課題は、この `PCI` にあたる回路を SFL で記述するというものです。図 2 では、 2 箇所に `PCI` が現れますが、記述するのは 1 個のモジュールです。すなわち、 2 箇所で共通に使える `PCI` を設計して下さい。なお、 `PCI` の満たすべき最低条件として、例題プログラムのシミュレーション実行結果が `system1` と `system2` で同一になることを要求します。

注意

図 2 (1), (2) や、上記の `system1`, `system2` の説明は、あくまで概念的な説明です。すなわち、「設計した PCI インタフェース回路で正しくデータ転送が行なえることを確認するために、 CPU と メモリ間に PCI バスを挿入してプログラムが正しく実行されるかどうかを調べる」ということを主張するものであり、パルテノン研究会が実際に準備するシミュレーション環境 `system1`, `system2` の詳細を定義するものではありません。これらの環境の詳細については、シミュレーション環境と同時に配布される資料 “PCI バス シミュレーション環境” を参照して下さい。

本稿で用いる用語について

バイト、ワード、ダブルワード
データ長の単位です。 1 バイト = 8 ビット、 1 ワード = 2 バイト、 1 ダブルワード = 2 ワード です。

デバイス・コア
CPU や、 memory など、 PCI バスに接続されたデバイスの中で、 PCI インターフェース回路部分をを除いた部分をさします。すなわち、 `iAccess` インターフェースを介して PCI インターフェース部の利用者となる回路です。

イニシエータ, ターゲット
PCI バスのアクセスにおいて、アクセスの主体となる PCI デバイスをイニシエータと呼び、イニシエータによるアクセスの対象となる PCI デバイスのことをターゲットと呼びます。

`iAccess` マスタ, `iAccess` スレーブ
`iAccess` インターフェースにおいて、アクセスの主体となる 側を `iAccess` マスタと呼び、 `iAccess` マスタによるアクセスの対象となる側のことを `iAccess` スレーブと呼びます。

(バス) アービタ
PCI バスにおいて、バスの使用権の調停をする回路のことです。

システム・メモリ (**I/O**, レジスタ) 空間、デバイス・メモリ (**I/O**, レジスタ) 空間
PCI バスでは、 32 ビットのメモリアドレス空間と 32 ビットの I/O アドレス空間が独立して存在していることを前提としています。これら二つのアドレス空間を、本稿ではそれぞれシステム・メモリ空間およびシステム・**I/O** 空間と呼びます。一方で、各 PCI ターゲット・デバイスは、独自のメモリアドレス空間や I/O アドレス空間を持ち得ます。これら二つのアドレス空間を、本稿ではそれぞれデバイス・メモリ空間およびデバイス・**I/O** 空間と呼びます。各デバイスのデバイス・メモリ空間およびデバイス・I/O 空間は、コンフィギュレーションによってシステム・メモリ空間およびシステム・ I/O 空間の然るべき位置に、重ならないようにマップされます。そしてマップされた位置の先頭アドレスが、コンフィギュレーション・レジスタのベース・アドレス・レジスタに格納されます。
　また、異なる PCI デバイス間で、相手側のコンフィギュレーション・レジスタを指定する場合、

ビット位置	31...24	23 ... 16	15 ... 11	10 ... 8	7 ... 2	10
データ	0...0	バス番号	デバイス番号	機能番号	レジスタアドレス	00

というフォーマットの値で指定します。本稿ではこの値をシステム・レジスタアドレスと呼びます。これに対し、単一デバイス内でレジスタを指定する場合には、バス番号とデバイス番号は不要なので、

ビット位置	31...11	10 ... 8	7 ... 2	10
データ	0...0	機能番号	レジスタアドレス	00

というフォーマットの値で指定します。本稿ではこの値をデバイス・レジスタアドレスと呼びます。

2　外部端子仕様

2.1　モジュール宣言

　モジュール `PCI` の外部端子の宣言は、ファイル `PCI.h` の中に記述してあります (本稿の末尾にリストがあります)。また、図 3 も参照して下さい。これらの信号線の型や名前を変更したり、信号を追加・削除したりすることは原則としてできません。オリジナルな実装のためにどうしても変更の必要がある場合は、変更内容とその根拠をレポート内に明記し、動作試験に用いた `SECONDS` スクリプトを提出して下さい。

各信号線の名前について、正論理の信号は大文字から始まる名前にし、負論理の信号は先頭の文字を 'n' にしてあります (例: PCIrdy, nFRAMEout)。ただし、モジュール PCI の記述に際して、内部信号などの名前にこの規則を強制はしません。

2.2 双方向信号の扱い

また、 *FRAME#* などの *s/t/s* (sustained tri-state) タイプの信号については、 SFL の中にこの物理動作をもった信号線の型がないため、次のように 3 本の信号の組を用います (* には、 PCI バス仕様で定められた信号線名から # を除いた文字列が入ります)。

input n*in 信号の値を内部で参照するための入力信号です。

output n*out 信号の値を外部に出力するための出力信号です。

instrout *enb: instr_arg *enb(n*out) 信号の値を外部に出力するタイミングを規定する制御信号です。この制御信号が起動されているときには、 output n*out がドライブされています。

例えば、 *FRAME#* の場合 input nFRAMEin, output nFRAMEout, instrout FRAMEenb となります。回路としてのイメージは、図 4 のようになります。
そして例えば *FRAME#* のアサート / ディアサートおよび信号線の値の参照は、以下のように記述します。

```
module PCI {
    ...
    FRAMEenb(0b0); /* アサート */
    ...
    FRAMEenb(0b1); /* ディアサート */
    ...
    R := nFRAMEin; /* 値の参照 */
}
```

これら *s/t/s* 信号の接続について、シミュレーション上では、図 2 の PCIbus 回路の中で、各 PCI からの *enb 信号を監視し、いずれも起動されていない時には PCIbus が n*in 信号をドライブするという仕組みを採用します。
さらに、 *AD, C/BE#, PAR* といった、通常の t/s (tri-state) タイプの信号については、シミュレーションの都合上、次の用に出力契機信号を付加します (* には、 AD, (n)C_BE, PAR のいずれかが入ります)。

bidirect * PCI の信号そのものを表す双方向信号です。

instrout *enb: instr_arg *enb(*) 信号の値を外部に出力するタイミングを規定する制御信号です。この制御信号が起動されているときには、 bidirect * が、 output としてドライブされています。

そして、これら AD, nC_BE, PAR の信号を出力として用いる時には、必ず次のように *enb 信号を用いて下さい。

```
module PCI {
    ...
    C_BEenb(0b1101); /* O.K.                */
    ...
    nC_BE = 0b1101;  /* N.G. 直接代入は禁止 */
    ...
}
```

2.3 各信号の意味

2.3.1 iAccess 信号群

まず、 PCI の外部信号のうち、 iAccess インターフェースに関するものを説明します。

<u>instrout Ready</u>
バスのセットアップが完了し、バスサイクルを発生することができる状態である間、レベル H を保ちます。イニシエータ・デバイスのデバイス・コアは、この信号が起動されている時のみ、 ReadMem, ReadMemLine, WriteMem, ReadIO, WriteIO, ReadConfig, WriteConfig 信号を起動できます。

<u>instrout Reset</u>
デバイスコアに対するリセット動作を要求する制御信号です。

<u>instrout BusError</u>
PCI バス側で回復不能なエラーが生じたことを、デバイスコアに伝達する制御信号です。

<u>bidirect Adr<32></u>
バスアクセスのアドレスを示します。 iAccess マスタとしての動作の時には、デバイスコアから PCI へ、システム・メモリ (I/O, レジスタ) アドレスが入力され、 iAccess スレーブとしての動作の時には PCI からデバイスコアへデバイス・メモリ (I/O, レジスタ) アドレスが出力されます。

<u>bidirect Be<4></u>
バスアクセスの、バイトイネーブルマスクを示します。

(Be<3> → Data<31:24>)
(Be<2> → Data<23:16>)
(Be<1> → Data<15:8>)
(Be<0> → Data<7:0>)

という対応関係を持ち、 Be のそれぞれのビットが 1 のとき、対応するバイトアクセスが有効であることを示します。 iAccess マスタとしての動作の時には、デバイスコアから PCI へ入力され、 iAccess スレーブとしての動作の時には PCI からデバイスコアへ出力されます。

<u>bidirect Data<32></u>
バスアクセスのデータを示します。 iAccess マスタ、 iAccess スレーブ動作の両方で共通に用いられます。

<u>instrin ReadMem(Adr, Be)</u>
iAccess マスタによる、 1 ダブルワードのメモリリード要求を示す制御信号です。システム・メモリ空間内のダブルワードアドレス (下位 2 ビットが 00) を引数として起動されます。読み出された各メモリ値は、制御信号 ReadDone とともに、 Data 信号に現れます。何らかの原因によりアクセスが失敗した場合は、 AccessError の起動によってデバイスコアにその旨を伝えます。

<u>instrin ReadMemLine(Adr, Be)</u>
iAccess マスタによる、 1 キャッシュ・ラインのメモリ・リード要求を示す制御信号です。キャッシュラインのサイズについては、何らかの方法 (例えば PCI のコンフィギュレーション) で、合意がとれていることを前提とします。 Adr はキャッシュライン中の任意のダブルワードの、メモリ上の位置を表すシステム・メモリアドレスであり、キャッシュラインのリードアクセスはいわゆるキャッシュライン・ラップモードで行なわれます。すなわち、 Adr, Adr + 1, ..., (キャッシュラインの末尾アドレス), (キャッシュラインの先頭アドレス), ..., Adr - 1 という順序で、各アドレスのメモリ値が読み出されます。読み出された各メモリ値は、制

Adr<1:0>	Be<3:0>
00	---1
01	--10
10	-100
11	1000

-：0，1 のどちらも許される

表 1: I/O アクセス時に Be がとり得る値

御信号 ReadDone とともに、 Data 信号に現れます。すなわち、 ReadMemLine の応答は、キャッシュラインサイズ (単位はダブルワード) と等しい回数の ReadDone の起動です。何らかの原因によりアクセスが失敗した場合は、 AccessError の起動によってデバイスコアにその旨を伝えます。

instrin WriteMem(Adr, Be, Data)

iAccess マスタによる、 1 ダブルワードのメモリライト要求を示す制御信号です。システム・メモリ空間内のダブルワードアドレス (下位 2 ビットが 00)、バイトイネーブル信号、および書き込みデータを引数として起動されます。制御信号 WriteDone が起動されることによって、書き込みが成功したことが応答されます。何らかの原因によりアクセスが失敗した場合は、 AccessError の起動によってデバイスコアにその旨を伝えます。

instrin ReadIO(Adr, Be)

iAccess マスタによる、最大 1 ダブルワードの I/O リード要求を示す制御信号です。 Adr は、アクセスする I/O ポートのシステム・I/O アドレスです。 Be はアクセスするバイトを指定するバイトイネーブルマスクですが、 Adr はバイト単位で指定可能なので、 Adr<1:0> の値によって、 Be のとり得る値は表 1 の通りです。実際にアクセスされるのは、 Adr を先頭アドレスとする 4 バイト ではなく、 Adr の値の下位 2 ビットを 00 としたアドレスを先頭とするダブルワードのうち、 Be の対応するビットが 1 であるバイトであることに注意して下さい。アクセスが成功した場合、 ReadDone の起動と共に読み出された値が Data 上に載ります。このとき、 Be の対応するビットが 0 であるバイトにどのようなデータを載せるかは規定されません。何らかの原因によりアクセスが失敗した場合は、 AccessError の起動によってデバイスコアにその旨を伝えます。

instrin WriteIO(Adr, Be, Data)

iAccess マスタによる、 1 ダブルワードの I/O ライト要求を示す制御信号です。 Adr は、アクセスする I/O ポートのシステム・I/O アドレスで、 Be はアクセスするバイトを指定するバイトイネーブルマスク、 Data はライトデータです。アクセスポジションに関する Adr, Be の注意点は、 ReadIO の場合と同様です。アクセスの成功は WriteDone によって、失敗は AccessError によってデバイスコア伝えます。

instrin ReadConfig(Adr, Be)

デバイス・コアが、 PCI デバイスのコンフィギュレーション・レジスタのリードを要求するための信号です。 Adr は、システム・レジスタアドレスです。読み出されたコンフィギュレーション・データは、制御信号 ReadDone とともに、 Data 信号に現れます。何らかの原因によりアクセスが失敗した場合は、 Access Error の起動によってデバイスコアにその旨を伝えます。

この制御信号の動作を実装する必要はありませんが、もし実装するならば、次のことに注意して下さい。

1. デバイス番号 n $(0 \le n \le 20)$ のデバイスの *IDSEL* 信号は、 *AD* 信号の $(11+n)$ 番ビットでドライブするという接続形態を前提として下さい。したがってコンフィギュレーション・サイクルのアドレス・フェーズ時に AD 信号に載せる値のために、 Adr 内のデバイス番号をデコードしてやる必要があります。

2. *IDSEL* 信号のドライブには、2 clock かかることを前提として下さい。したがって、コンフィギュレーション・サイクルのアドレス・フェーズを開始する 1 clock 前から AD 信号を連続的にドライブしてやる必要があります。

instrin WriteConfig(Adr, Be, Data)
デバイス・コアが、PCI デバイスのコンフィギュレーション・レジスタへのライトを要求するための信号です。Adr の値は、システム・レジスタアドレスです。アクセスの成功は WriteDone によって、失敗は AccessError によってデバイスコア伝えます。

この制御信号の動作を実装する必要はありませんが、もし実装するならば、ReadConfig 信号の説明で述べた注意点を守って下さい。

instrout ReadDone(Data)
制御信号 ReadMem, ReadMemLine, ReadIO, ReadConfig によるリード要求に対する、成功応答となる制御信号です。Data には読み出した値を出力します。

instrout WriteDone()
制御信号 WriteMem, WriteIO, WriteConfig による各ライト要求に対する、成功応答となる制御信号です。

instrout AccessError()
制御信号 ReadMem, ReadMemLine, ReadIO, ReadConfig, WriteMem, WriteIO, WriteConfig によるアクセス要求に対する、失敗応答となる制御信号です。

instrout ReadMemReq(Adr, Be)
~~iAccessスレーブにメモリリード要求を伝えるための信号です。Adrの値は、デバイス・メモリアドレスです。iAccessスレーブはこのリード要求に対し、ReadReqDoneによってリード・データをData信号に載せます。リードの失敗は、ReqErrorの起動によって通知されます。~~

この制御信号を使う必要はありません。

instrout ReadMemLineReq(Adr, Be)
iAccess スレーブに、1 キャッシュ・ラインのメモリ・リード要求を伝えるための信号です。Adr の値は、キャッシュライン中の任意のダブルワードの、メモリ上の位置を表すデバイス・メモリアドレスです。キャッシュラインのリードアクセスはいわゆるキャッシュライン・ラップモードで行なわれます。このリード要求に対し、ReadReqDone によって、1 キャッシュライン分のリード・データが順に Data 信号に出力されます。リードの失敗は、ReqError の起動によって通知されます。

instrout WriteMemReq(Adr, Be, Data)
iAccess スレーブに、メモリ・ライト要求を伝えるための信号です。Adr の値はデバイス・メモリアドレスです。Be, Data の値については WriteMem 信号を参照して下さい。このライト要求に対して、アクセスの成功は WriteReqDone によって、失敗は ReqError によって通知されます。

instrout ReadIOReq(Adr, Be)
iAccess スレーブに I/O リード要求を伝えるための信号です。Adr の値はデバイス I/O アドレスです。Be の値については、ReadIO を参照して下さい。このリード要求に対し、ReadReqDone によってリード・データを Data 信号に載せます。リードの失敗は、ReqError の起動によって通知します。

instrout WriteIOReq(Adr, Be, Data)
iAccess スレーブに、I/O ライト要求を伝えるための信号です。Adr の値はデバイス・I/O アドレスです。Be, Data の値については WriteIO 信号を参照して下さい。このライト要求に対して、アクセスの成功は WriteReqDone によって、失敗は ReqError によって通知されます。

instrout ReadConfigReq(Adr, Be)
~~iAccessスレーブに、コンフィギュレーション・レジスタのリード要求を伝えるための信号です。Adrの値は、デバイス・レジスタアドレスです。このリード要求に対しiAccessスレーブは、ReadReqDoneによってリード・データをData信号に載せます。リードの失敗は、ReqErrorの起動によって通知します。~~
この制御信号を使う必要はありません。

instrout WriteConfigReq(Adr, Be, Data)
~~iAccessスレーブに、コンフィギュレーション・レジスタのライト要求を伝えるための信号です。Adrの値は、デバイス・レジスタアドレスです。Be,**Data**の値についてはWriteConfig信号を参照して下さい。このライト要求に対して、アクセスの成功はWriteReqDoneによって、失敗はReqErrorによって通知されます。~~
この制御信号を使う必要はありません。

instrin ReadReqDone(Data)
制御信号 ReadMemReq, ReadMemLineReq, ReadIOReq, ReadConfigReq によるリード要求に対する、成功応答となる制御信号です。 Data には読み出した値を出力します。

instrin WriteReqDone()
制御信号 WriteMemReq, WriteIOReq, WriteConfigReq によるライト要求に対する、成功応答となる制御信号です。

instrin ReqError()
制御信号 ReadMemReq, ReadMemLineReq, ReadIOReq, ReadConfigReq, WriteMemReq, WriteIOReq, WriteConfigReq によるアクセス要求に対する、失敗応答となる制御信号です。

アクセス・タイミング・チャート　以上の iAccess インターフェースについて、リード・ライトアクセスのタイミングチャートを図 5, 6, 7に示します。リードライン・アクセスについは、キャッシュラインサイズ = 4 ダブルワードの場合の例を示しています。

ただし、これらの図は、各要求と応答の順序関係と、制御信号と転送データの組合せ関係を示しているに過ぎません。すなわち、図中では要求と応答が 1 クロックずつ順に行なわれていますが、極端な場合にはこれらすべての要求・応答が 1 クロック内で同時に行なわれるかも知れませんし、要求と応答との間に任意クロックのウェイトが挿入されることもあり得ます。

また、 AccessError, ReqError の例を省略してありますが、 AccessError, ReqError は、リードまたはライトの要求が発生した以降の任意のクロックで起動することができます。起動の期間は 1 クロックです。

なお、 ReadMem, ReadMemLine, WriteMem, ReadIO, WriteIO, ReadConfig, WriteConfig は、互いに排他的に起動されることを前提とします。これらのうち複数が同時に起動された時の動作は規定しません。さらに、 iAccess スレーブとして動作している間は、これらの制御信号を起動することはできません。すなわち、 iAccess マスタ動作と iAccess スレーブ動作は排他的です。

2.3.2　PCI インターフェース信号群

次に、 PCI バスインターフェースに関する信号を説明します。前述のように、 $s/t/s$ 属性の信号については、 3 本の信号の組によって実現し、 t/s 属性の信号については、 2 本の信号の組によって実現していることに注意して下さい。また、信号の極性については原則として PCI の仕様に準じていますが、都合により $SERR\#$ のみ正極性にしています。

input nRST<32>
PCI の $nRST\#$ 信号が常に入力されます。

bidirect AD<32>
PCI の $AD\#$ 信号となります。PCI のバスサイクルにおいて、アドレスフェーズでは、アクセスのアドレスを表し、データフェーズではアクセスデータを表します。この信号に値を出力する場合、必ず制御信号 ADenb を起動して下さい。

instrout ADenb
双方向信号 AD<32> が出力方向にドライブされていることを示す制御信号です。

bidirect nC_BE<4>
PCI の $C/BE\#$ 信号となります。アドレスフェーズではバスコマンドを表し、データフェーズではアクセスデータのバイトイネーブルマスクを表します。この信号に値を出力する場合、必ず制御信号 C_BEenb を起動して下さい。

instrout C_BEenb
双方向信号 nC_BE<4> が出力方向にドライブされていることを示す制御信号です。

bidirect PAR
PCI の PAR 信号となります。バスサイクルの各クロックにおいて、AD<32>, nC_BE<4> の合計 36 本の信号に対し、偶数パリティのパリティ信号を、1 クロック遅れて転送します。この信号に値を出力する場合、必ず制御信号 PARenb を起動して下さい。

instrout PARenb
双方向信号 PAR が出力方向にドライブされていることを示す制御信号です。

input nFRAMEin, output nFRAMEout, instrout FRAMEenb
PCI の $FRAME\#$ 信号となります。有効なバスサイクルが発生中であることを表します。この信号がアサートされたクロックよりバスサイクルが発生し、ディアサートされた次のクロックでバスサイクルは終了します。

input nIRDYin, output nIRDYout, instrout IRDYenb
PCI の $IRDY\#$ 信号となります。バスサイクルのイニシエータがデータ転送可能状態であることを表します。

input nTRDYin, output nTRDYout, instrout TRDYenb
PCI の $TRDY\#$ 信号となります。バスサイクルのターゲットがデータ転送可能状態であることを表します。

input nSTOPin, output nSTOPout, instrout STOPenb
PCI の $STOP\#$ 信号となります。現在実行中のデータ転送の中止要求を、ターゲットからイニシエータへ伝えます。

instrin IDSEL
PCI の $IDSEL$ 信号となります。コンフィギュレーションアクセス時のターゲットデバイスを指定する信号です。

input nDEVSELin, output nDEVSELout, instrout DEVSELenb
PCI の $DEVSEL\#$ 信号となります。イニシエータのバスアクセスに対し、ターゲットの応答を伝える信号です。

output nREQ
PCI の $REQ\#$ 信号となります。イニシエータが、バスの使用権を要求する時に用います。

output nGNT
PCI の $GNT\#$ 信号となります。バスアービタが、イニシエータにバスの使用権を与える時に用います。

input nPERRin, output nPERRout, instrout PERRenb
PCI の $PERR\#$ 信号となります。 AD, nC_BE に対するパリティエラーが検出されたことを伝えます。

instrout SERRenb
PCI の $SERR\#$ 信号となります。致命的なエラーが発生したことを伝えます。

3 課題モジュールで実装すべき PCI 動作

PCI の仕様のうち、規定課題 PCI モジュールが実装しなくてはならない項目と、実装しなくてもよい項目を挙げます。

3.1 データ転送

バスコマンド PCI のバスコマンドのうち、

コマンド名	$C/BE\#$ の値
メモリ・リード	0110
メモリ・リード・ライン	1110
メモリ・ライト	0111
I/O リード	0010
I/O ライト	0011
コンフィギュレーション・リード	1010
コンフィギュレーション・ライト	1011

の各コマンドを実装すること。これ以外のバスコマンドは実装しなくてよい。なお、コンフィギュレーション・リード (ライト) を除く上記の各コマンドについては、イニシエータ動作、ターゲット動作の両方を実装すること。コンフィギュレーション・リード (ライト) については、ターゲット動作のみ実装すればよい。

ステッピング 信号 AD, PAR に対する連続ステッピングおよび分散ステッピングは実装しなくてよい。

アドレス・デコーディング ポジティブ・デコーディングを実装すること。サブトラクティブ・デコーディングは実装しなくてよい。

アドレッシング・タイプ メモリ・バースト転送におけるアドレッシングについて、リニア・モード (AD<1:0> == 0b00) およびキャッシュ・ライン・ラップ・モード (AD<1:0> == 0b01) はどちらも実装すること。

3.2 バス・サイクルの終了

バスサイクルの終了パターンは、完了、タイムアウト、マスタ・アボート、ディスコネクト、リトライ、ターゲット・アボート のすべてを実装すること

3.3 アービトレーション

アービトレーション 信号 *REQ#*, *GNT#* を用いた、バス使用権のアービトレーションへの対応を実装すること。アービトレーションを行なう回路は パルテノン研究会が用意します。

パーキング アービトレーション・パーキングへの対応を実装すること。

高速バック・ツー・バック・トランザクション 高速バック・ツー・バック・トランザクションは実装しなくてもよい。

3.4 排他的アクセス

リソース・ロックとバス・ロックは共に実装しなくてよい。

3.5 キャッシュ・メモリのサポート

信号 *SDONE, SBO#* による、キャッシュサポートの動作は実装しなくてよい。

3.6 インタラプトとエラー処理

インタラプト動作 信号 *INTA#*, *INTB#*, *INTC#*, *INTD#* によるインタラプト動作は、実装しなくてよい。

パリティ・エラー検出 信号 *PERR#* によるパリティ・エラー検出を実装すること。また、信号 *SERR#* による致命的エラーの通知は、必要に応じて実装すればよい。

3.7 64 ビット拡張

PCI 仕様の 64 ビット拡張については実装しなくてよい。

3.8 コンフィギュレーション・サイクル

~~コンフィギュレーション・メカニズム1を実装すること。コンフィギュレーション・メカニズム2は実装しなくてよい。~~

コンフィギュレーション・サイクル・タイプ タイプ 0 コンフィギュレーション・サイクルを実装すること。タイプ 1 コンフィギュレーション・サイクルは実装しなくてよい。

IDSEL のステッピング パルテノン研究会が用意する system2 の SFL 記述では、 IDSEL 信号を、 AD のあるビットでドライブする実装法を採用します。この場合、電気的仕様により、コンフィギュレーションサイクルのアドレスフェーズにおいて、 IDSEL 信号のドライブに連続ステッピングが適用されます。~~シミュレーションにおいてはこの動作を、「IDSELのドライブを2クロックに渡って行なう」という動作に置き換えます。したがってこのIDSELに対する応答となるFRAME#のアサートは、2クロック目に行なうという実装にすること。~~コンフィギュレーションのイニシエータ動作を実装する場合には気をつけること (ReadConfig, WriteConfig の説明を参照)。

3.9 コンフィギュレーション・レジスタ

PCI が定める、 256 バイトのコンフィギュレーション・レジスタはすべて実装すること。ただし、設計者の意図により読み出し専用としたレジスタに関して、記憶素子を割り当てない実装にすることは構わない。また、このレジスタ群を機能回路として記述し、合成の対象外としても構わない。ただし、この場合その機能回路のインターフェースや、シミュレーション時の設定方法などを明確に報告すること。

参考文献

[1] 滝 誠一: “PCI バスの詳細と応用へのステップ”, インターフェース増刊 OpenDesign No.7, pp.4 – 140, CQ 出版社, 1995 年 3 月

[2] John L.Hennessy and David A.Patterson; 富田 眞治 ・ 村上 和彰 ・ 新實治男 訳: “コンピュータ・アーキテクチャ: 設計・実現・評価の定量的アプローチ”, 日経 BP 社, 1992 年

A 図

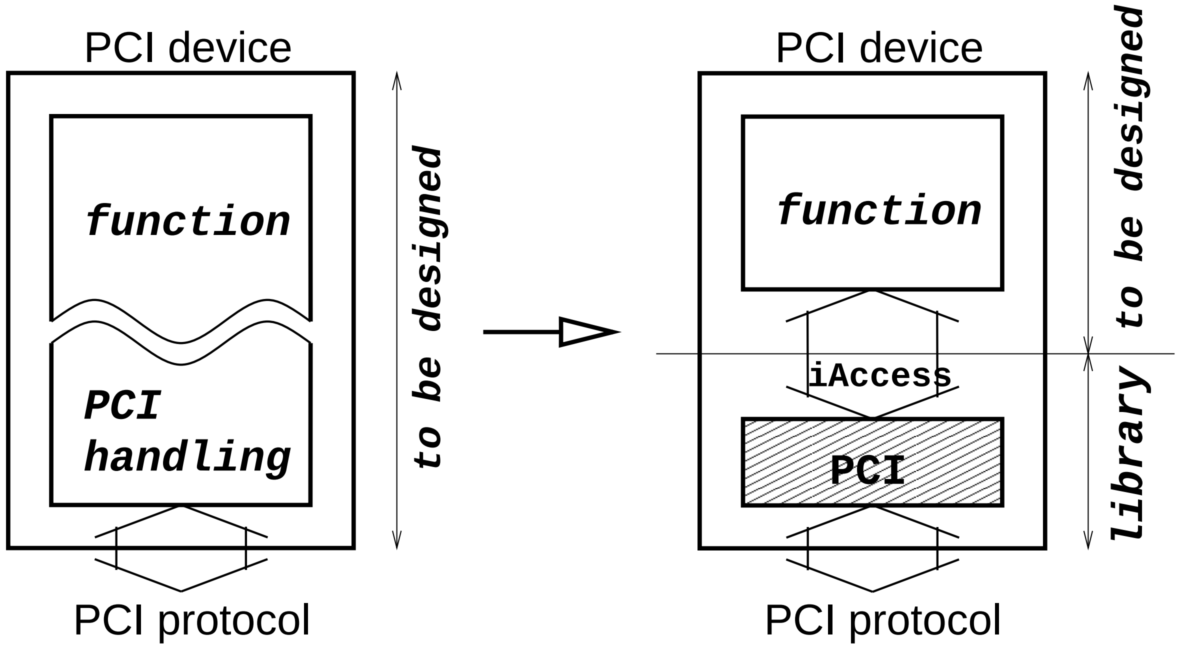

図 1: ライブラリ・モジュール化された PCI バス インターフェース回路

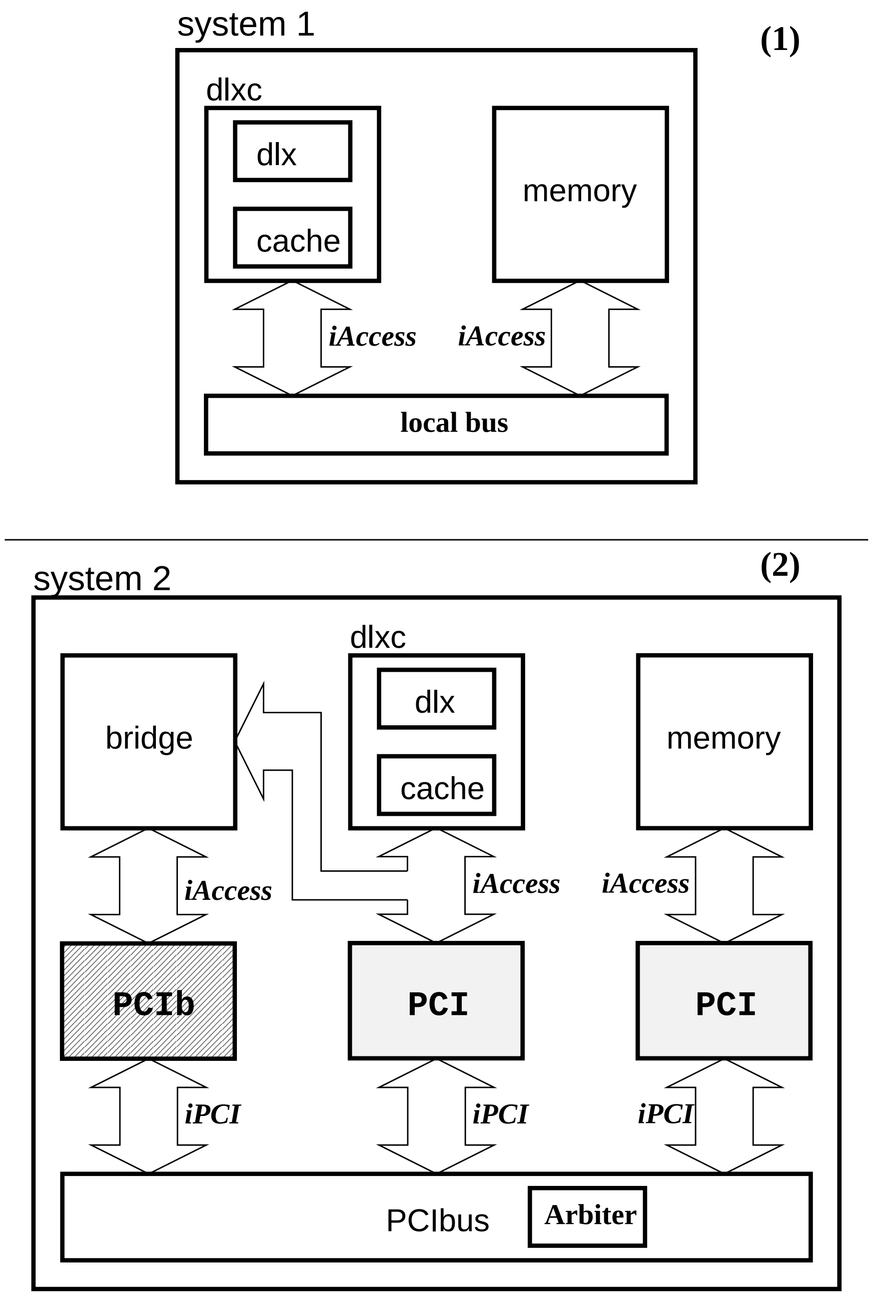
system 1
(1)
dlxc
dlx
cache
memory
iAccess
iAccess
local bus
system 2
(2)
dlxc
bridge
dlx
cache
memory
iAccess
iAccess
iAccess
PCIb
PCI
PCI
iPCI
iPCI
iPCI
PCIbus
Arbiter

図 2: 課題の概要

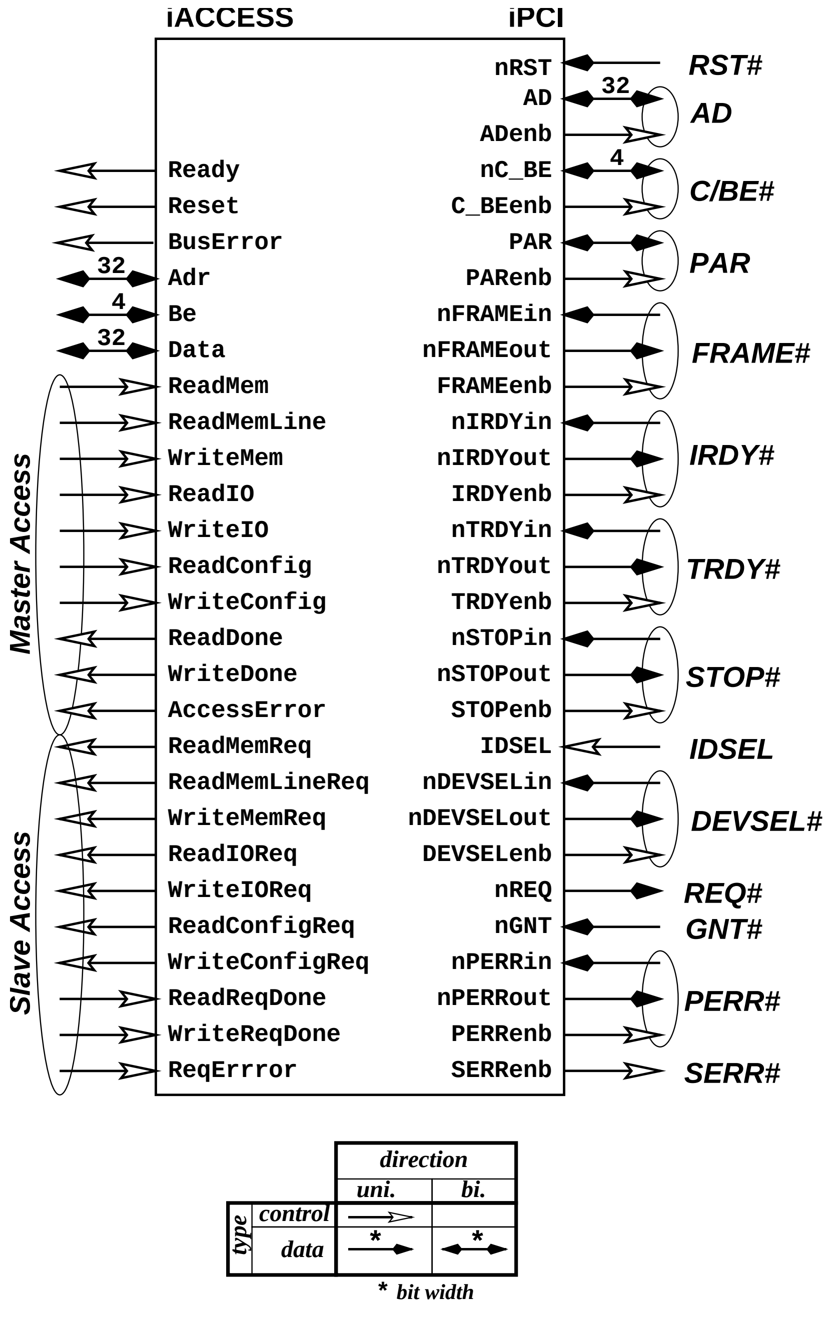
iACCESS
iPCI
nRST
AD
ADenb
nC_BE
C_BEenb
PAR
PARenb
nFRAMEin
nFRAMEout
FRAMEenb
nIRDYin
nIRDYout
IRDYenb
nTRDYin
nTRDYout
TRDYenb
nSTOPin
nSTOPout
STOPenb
IDSEL
nDEVSELin
nDEVSELout
DEVSELenb
nREQ
nGNT
nPERRin
nPERRout
PERRenb
SERRenb
RST#
32
AD
4
C/BE#
PAR
FRAME#
IRDY#
TRDY#
STOP#
IDSEL
DEVSEL#
REQ#
GNT#
PERR#
SERR#
Ready
Reset
BusError
32
Adr
4
Be
32
Data
ReadMem
ReadMemLine
WriteMem
ReadIO
WriteIO
ReadConfig
WriteConfig
ReadDone
WriteDone
AccessError
ReadMemReq
ReadMemLineReq
WriteMemReq
ReadIOReq
WriteIOReq
ReadConfigReq
WriteConfigReq
ReadReqDone
WriteReqDone
ReqErrror
Master Access
Slave Access
direction
uni.
bi.
type
control
data
*
*
* bit width

図 3: 外部信号

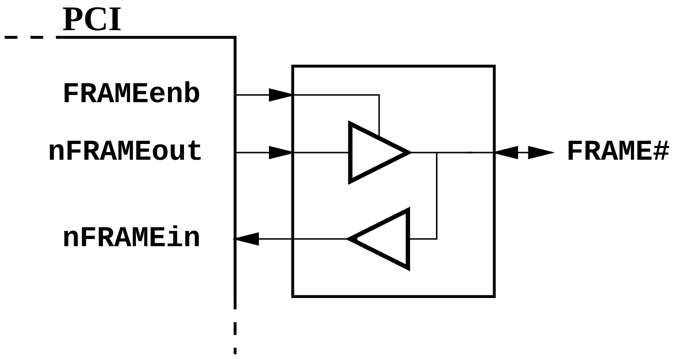

図 4: $s/t/s$ 信号の分離

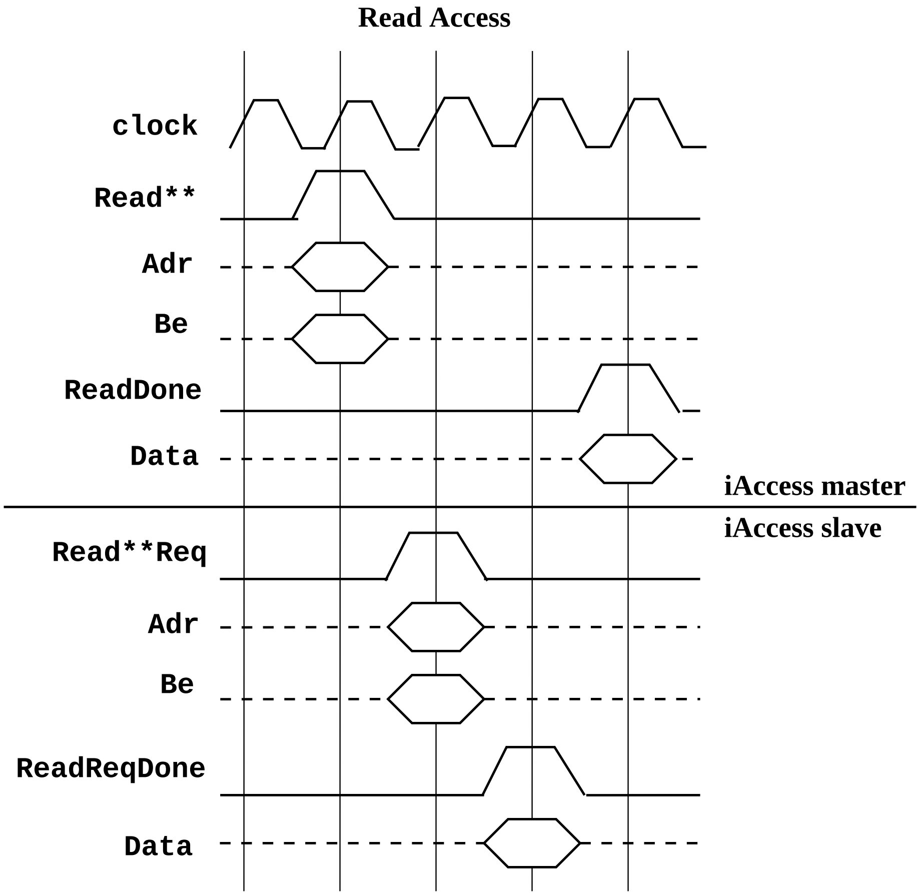

図 5: iAccess インターフェース: リード・アクセス

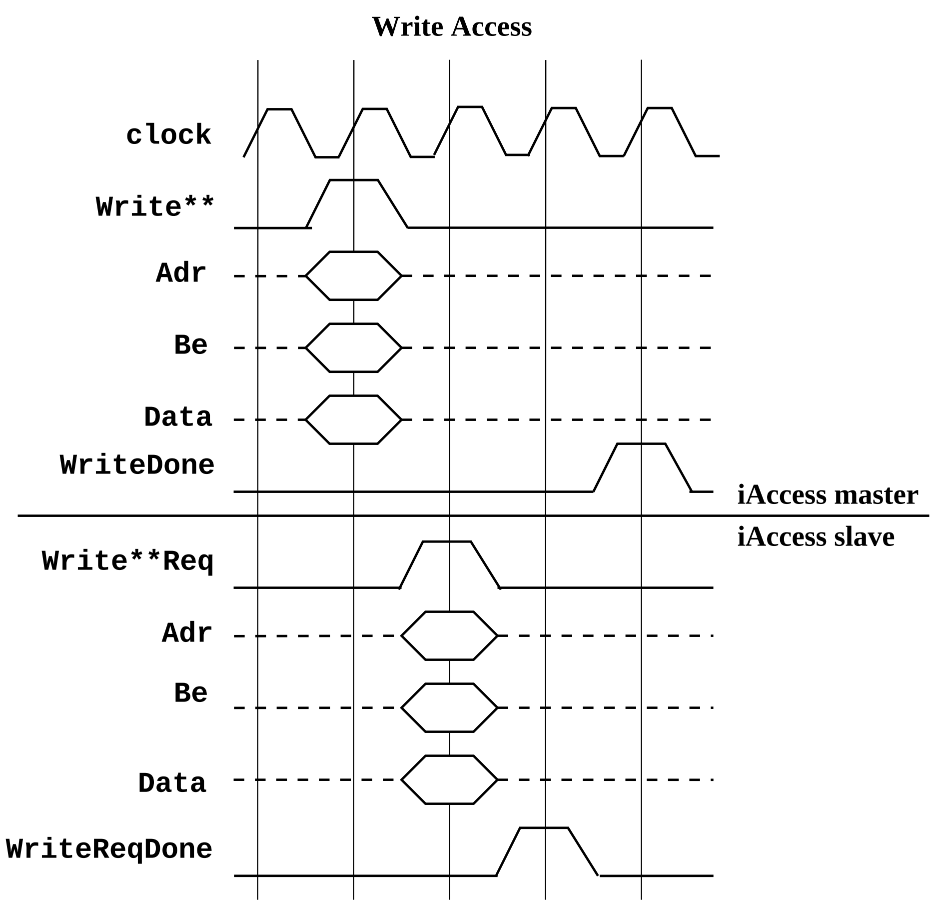

図 6: iAccess インターフェース: ライト・アクセス

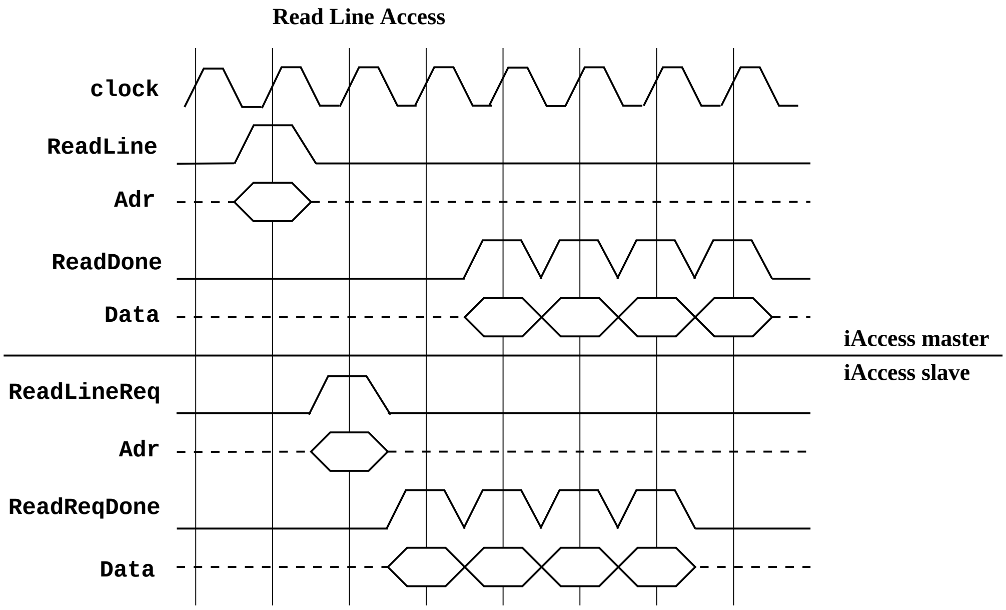

図 7: iAccess インターフェース: メモリ・リード・ライン・アクセス

B SFL ソースリスト (pci.h)

```
/* /home/nagami/src/master/dlx/PCI.h,v
** nagami
**
** 1996/08/15 17:59:42
** 1.2
** Exp */
declare PCI {
/****************************************************************/
/*  Access Interface                                            */
/****************************************************************/
    /* Control signals */
    instrout Ready;
    instrout Reset;
    instrout BusError;

    /* Data access signals */
    bidirect Adr<32>;
    bidirect Be<4>;
    bidirect Data<32>;
    instrin ReadMem;
    instrin ReadMemLine;
    instrin WriteMem;
    instrin ReadIO;
    instrin WriteIO;
    instrin ReadConfig;
    instrin WriteConfig;
    instrout ReadDone;
    instrout WriteDone;
    instrout AccessError;

    instrout ReadMemReq;
    instrout ReadMemLineReq;
    instrout WriteMemReq;
    instrout ReadIOReq;
    instrout WriteIOReq;
    instrout ReadConfigReq;
    instrout WriteConfigReq;
    instrin ReadReqDone;
    instrin WriteReqDone;
    instrin ReqError;

/****************************************************************/
/*  PCI interface                                               */
/****************************************************************/
    /* System pins */
    input nRST;

    /* Address and data pins */
    bidirect AD<32>; instrout ADenb;
    bidirect nC_BE<4>; instrout C_BEenb;
    bidirect PAR; instrout PARenb;

    /* Interface control pins */
    input nFRAMEin; output nFRAMEout; instrout FRAMEenb;
    input nIRDYin; output nIRDYout; instrout IRDYenb;
    input nTRDYin; output nTRDYout; instrout TRDYenb;
    input nSTOPin; output nSTOPout; instrout STOPenb;
```

```
    instrin IDSEL;
    input nDEVSELin; output nDEVSELout; instrout DEVSELenb;
    output nREQ;
    input nGNT;
    input nPERRin; output nPERRout; instrout PERRenb;
    instrout SERRenb;

/***********************************+**********************************/
/*  Instruct Arguments                                                */
/***********************************+**********************************/
    instr_arg ReadMem    (Adr, Be);
    instr_arg ReadMemLine(Adr, Be);
    instr_arg WriteMem   (Adr, Be, Data);
    instr_arg ReadIO     (Adr, Be);
    instr_arg WriteIO    (Adr, Be, Data);
    instr_arg ReadConfig (Adr, Be);
    instr_arg WriteConfig(Adr, Be, Data);

    instr_arg ReadReqDone(Data);
    instr_arg WriteReqDone();
    instr_arg ReqError();

    instr_arg IDSEL();
    /* In your 'PCI' module, you should declare these instr_arg's. */
    /*
    instr_arg Ready();
    instr_arg Reset();
    instr_arg BusError();
    instr_arg ReadDone(Data);
    instr_arg WriteDone();
    instr_arg AccessError();

    instr_arg ReadMemReq    (Adr, Be);
    instr_arg ReadMemLineReq(Adr, Be);
    instr_arg WriteMemReq   (Adr, Be, Data);
    instr_arg ReadIOReq     (Adr, Be);
    instr_arg WriteIOReq    (Adr, Be, Data);
    instr_arg ReadConfigReq (Adr, Be);
    instr_arg WriteConfigReq(Adr, Be, Data);

    instr_arg ADenb(AD);
    instr_arg C_BEenb(nC_BE);
    instr_arg PARenb(PAR);
    instr_arg FRAMEenb(nFRAMEout);
    instr_arg IRDYenb(nIRDYout);
    instr_arg TRDYenb(nTRDYout);
    instr_arg STOPenb(nSTOPout);
    instr_arg DEVSELenb(nDEVSELout);
    instr_arg PERRenb(nPERRout);
    instr_arg SERRenb();
     */
}
```