計算機科学実験及演習 3 HW

SIMPLE/*KITE*

バス・インターフェイス回路 仕様書

Revision History

-1.0	'99 6/02	M.Goshima	第 1 稿
1.0	'00 4/12	M.Goshima	公開
1.1	'00 5/19	M.Goshima	バグ・フィクス
1.2	'00 6/30	M.Goshima	バグ・フィクス
1.3	'01 4/19	M.Goshima	バグ・フィクス

変 更 履 歴

Ver. 1.0 から Ver. 1.1 への変更点

三木 武 氏（平成 12 年度，3 回生）の指摘により，以下の 2 点の修正を施した．

ACK# に関する誤解

問 題 点

Ver. 1.0 以前の実装では ack は，*KITE* ボードからの ACK# 信号を同期化しただけのものであった．にも関わらず，ちょうど 1 サイクルだけアサートされるかのように設計してある．

対 処 法

この問題点への対処法としては，全体の修正量を最小化するため，3 章で述べる実装に加えて，2 章で述べる SIMPLE I/F にも変更を加えることにする．

Ver 1.0 以前では，

> in を指示してから，ack がアサートされるまでの間，(nop 以外の）コマンドの入力は無視される

としているが，これを

> in を指示してから，ack がアサートされるまでの間，in を指示し続けなければならない

と変更する．

SIMPLE コア側で IN 命令の実行によってフェイズ p4 で実行を中断するようにしているならば，この仕様の変更によって SIMPLE の設計を変更する必要はないと思われる[1]．

この変更によって，状態の維持を SIMPLE コアに押し付けることができるので，バス・インターフェイス回路 BIF の実装はむしろ簡略化することができた．

MREQ/IOREQ#/RW# の連続アサート

問 題 点

Ver. 1.0 以前のタイミング設計の記述では，SIMPLE をパイプライン化した場合に，引き続く 2 つのバス・アクセスに対応して，MREQ/IOREQ#/RW# が連続してアサートされることがある．

対 処 法

内容自体が advanced であるので，本文中に問題点を指摘するに留めた．

Ver. 1.1 から Ver. 1.2 への変更点

小澤 正幸 氏（平成 12 年度，3 回生）の指摘により，修正を施した．

[1] 実際問題，『無視される』という仕様は，当初考えられた実装が無視するようなものであったためである．

Ver. 1.2 から Ver. 1.3 への変更点

以下の 2 点を修正した：

1. 図 1 に，ack，およびそれ以外のスイッチを追加.
2. 図 5 中の，入力の一方がバス，もう一方が 1 本の信号線になっている AND ゲートの説明を追加.

本稿の内容

本稿では，*KITE*マイクロプロセッサボード[3] 上に実装する，SIMPLE [1] マイクロ・プロセッサの **バス・インターフェイス回路 BIF** について述べる．

本稿の内容は以下のとおりである：

1 章　BIF の概要

BIF の，*KITE* ボードと SIMPLE コアに対する位置づけ，および，その機能について概説する．

2 章　BIF の SIMPLE I/F

BIF の SIMPLE コアに対する I/F について詳述する．BIF を用いて SIMPLE コアを担当する設計者は，主に 1 章と本章を読めばよい．

3 章　BIF の実装

BIF バス・インターフェイス回路 BIF 内部の実装例について詳述する．

4 章　動作試験の方法

BIF の実機上での動作試験の方法について簡単に述べる．

目 次

図 目 次

表 目 次

1 BIF の概要

BIF は，*KITE* ボード上の LCA[4] において，LCA 内部の SIMPLE のコア部分と，LCA 外部のメモリ，および，I/O ポートとの I/F（インターフェイス）を司る．

図 1 に，*KITE* ボードのバス周りのブロック図を示す．

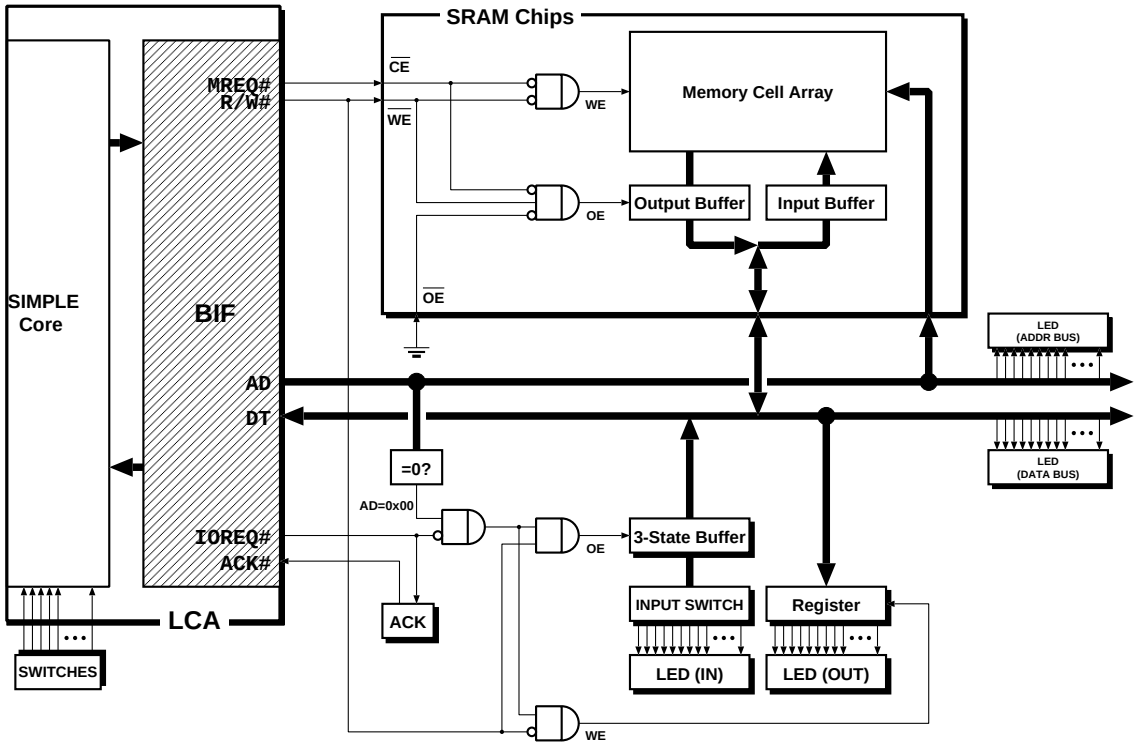

図 1: *KITE* ボードのバス周りのブロック図

BIF の主な機能は，*KITE* ボード上の SRAM，および，I/O ポートの非同期的な I/F を隠蔽し，SIMPLE コアに対して同期的 I/F を提供することである．

2 BIF の SIMPLE I/F

BIF は，SIMPLE コアに対して同期的 I/F を提供する．

2.1 SIMPLE I/F の概要

BIF の入出力端子を，表 1 に示す．

端子名	I/O	幅	説明
cmd	I	3	コマンド（後述）．
ad	I	16	アドレス．in/out 命令に対して，0 にする必要はない．
rd	O	16	読み出しデータ（load/in 命令）
wd	I	16	書き込みデータ（store/out 命令）
ack	O	1	読み出しデータ・レディ（後述）．
clk	I	1	クロック．
rst	I	1	（システム）リセット．クロックに同期している必要がある．

表 1: BIF の入出力端子

コマンド cmd の割り当てを表 2 に示す．

cmd[2:0]	operation
0XX	nop
100	load/ifetch
101	store
110	in
111	out

表 2: コマンド cmd の割り当て

SIMPLE コアに対する入出力は，クロック clk に同期して行われる．図 2 に，SIMPLE I/F の信号波形を示す．

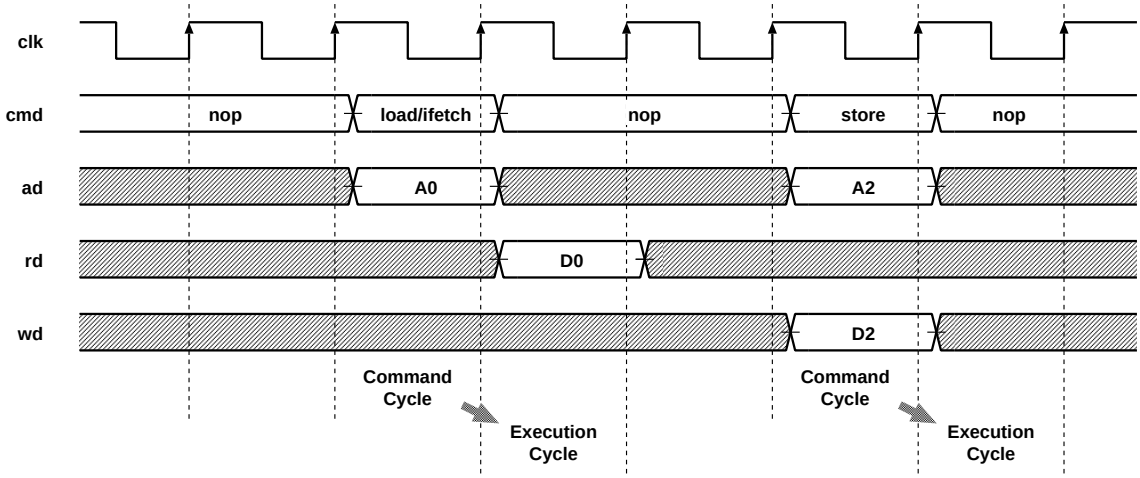

図 2: SIMPLE I/F の信号波形

図 5 に示されているように LCA 外部への出力には必ず FF を介するので，バスへのアクセスは BIF に cmd を与えたサイクルの次のサイクルに行われる．cmd を与えたサイクルをコマンド・サイクル，実際にバスへのアクセスが行われるコマンド・サイクルの次のサイクルを，そのコマンドに対する実行サイクルと呼ぶことにする．BIF への入出力は以下に示すタイミングで行われる：

- バスへのアクセスに ad が必要である load/ifetch，store では，コマンド・サイクルに ad を与える必要がある．
- バスへのアクセスに wd が必要である store，out では，コマンド・サイクルに wd を与える必要がある．
- load/ifetch では，rd に有効なデータが表われるのは，実行サイクルである．
- in では，rd に有効なデーが表われるサイクルは，静的には決まらず，ack によって指示される．

各コマンドに対する動作を以下にまとめる：

load/ifetch コマンド・サイクルにおいて，ad にメモリのアドレスを与える．実行サイクルにおいて，メモリ・アクセスが行われ，rd に読み出しデータが表われる．

store コマンド・サイクルに，ad，および，wd を与える．実行サイクルにおいて，メモリのアドレス ad に対して wd の書き込みが行われる．

in in コマンドを与えると，BIF は自動的に入力ポートからの入力（ACK ボタンの押下）を待つ．rd に読み出しデータが表わるサイクルは，ack によって指示される．

out コマンド・サイクルに wd を与える．出力ポートへの出力は，実行サイクルに行われる．

in，out コマンドでは，I/O ポートのアドレスを指定する必要はなく，コマンド・サイクルの ad は don't care である．

2.2 BIF と SIMPLE/B のフェイズ

本節では，BIF と SIMPLE コアの接続の方法を，SIMPLE/B[1] を例に説明する．

2.2.1 ad, rd, wd の接続

BIF の rd は，SIMPLE/B の，IR の入力と MDR の入力に接続することになる．基本的には，フェイズ p1 で IR の，フェイズ p4 で MDR のライト・イネーブルを，それぞれアサートすればよい．

wd には，AR への 入力 を接続する[2]．

ad には，PC への入力と DR への入力を選択したものを接続する．基本的には，p1 の前のフェイズでは PC への入力を，p3 の前のフェイズでは DR への入力を，それぞれ選択すればよい．p3 の前のフェイズでは常に p2 がアサートされているが，p1 の前のフェイズでは，リセット直後には p5 はアサートされていないであろう．したがってこのセレクタ（マルチプレクサ）の選択入力には p2 を接続し，p2 では DR への入力を，それ以外では PC への入力をそれぞれ選択するようにすればよい．

図 3 に，最初の命令が LD 命令であった場合の波形の例を示す．

[2]store/out のコマンド・サイクル以外では，AR の意味の無い値が wd に与えられるが，don't care である．

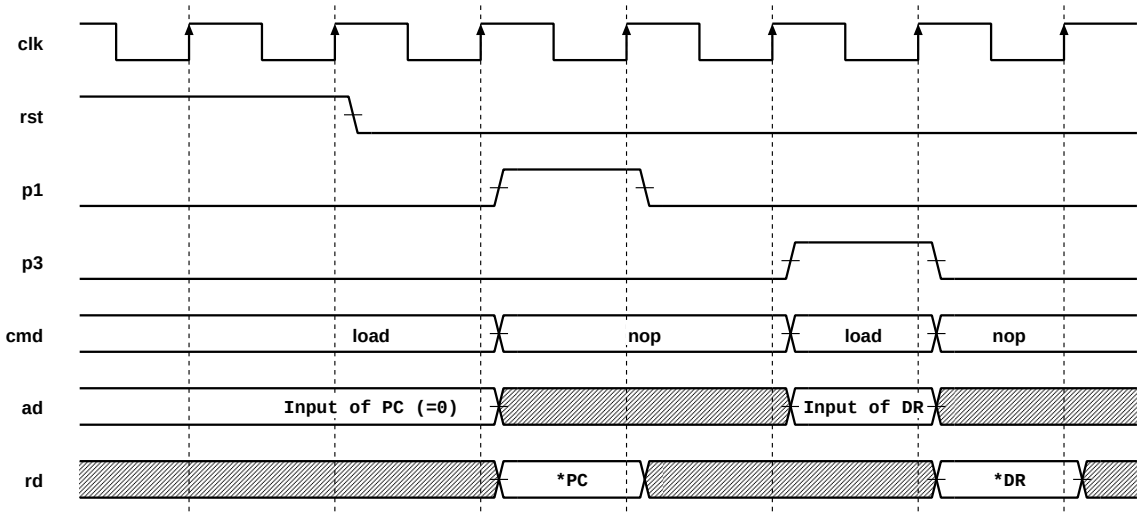

図 3: 最初の命令フェッチ

図では，rst によって cmd は load のパタンに，PC は 0 に初期化されている．rst が切れてから，最初の p1 がアサートされるまでに 1 サイクルの空きが設けてあり，このサイクルを最初の命令フェッチに対するコマンド・サイクルとしている[3].

2.2.2 in 命令

KITE ボードの入力ポートの仕様は，SIMPLE の IN 命令の仕様とは，うまく噛み合わない．*KITE* ボードでは，入力ポートからの読み出しが完了するまで，IOREQ#，R/W#，AD を確定させておく必要がある．一方，SIMPLE の IN 命令の仕様では，実際の入力処理自体が 1 サイクルで終了することを想定している[3].

本稿で述べる BIF では，SIMPLE アーキテクチャを次のように仕様変更することを想定する：すなわち，IN 命令を実行すると，プロセッサはその完了まで動作を停止する．入出力フラグは，仕様から削除する．

このように変更するとフェイズ制御回路は，IN 命令のフェイズ p4 で実行を中断し，ack のアサートによって再開すればよい．図 4 にその様子を示す[4].

なお，ack のアサートされるタイミングに関して，以下の点に注意すること：

- コマンド・サイクルには，ack はアサートされない．
- in を指示してから，ack がアサートされるまでの間，in コマンドを入力し続ける必要がある．

2.2.3 BIF と SIMPLE のパイプライン化

BIF は，コマンド・サイクル部と実行サイクル部自体がパイプライン化されているので，実行サイクル中に次のコマンドを受け付けることができる．したがって，基本的に毎サイクル新しいコマンドを受け付けることができ，SIMPLE のパイプライン化にもほぼそのまま対応可能である．

[3]実際には，rst が切れてから最初の p1 がアサートされるまでの空きサイクルは 1 サイクル以上あってよい．その場合，最初の命令の読み出しが 2 回以上行われるが，問題はない．

[4]図 4 では，入力ポートからの読み出しを実行している間，p4 がアサートされ続けている．p4 を MDR のライト・イネーブルに接続しているとその間 無効なデータが MDR に書き込まれることになるが，最後に有効なデータによって上書きされるので問題ない．

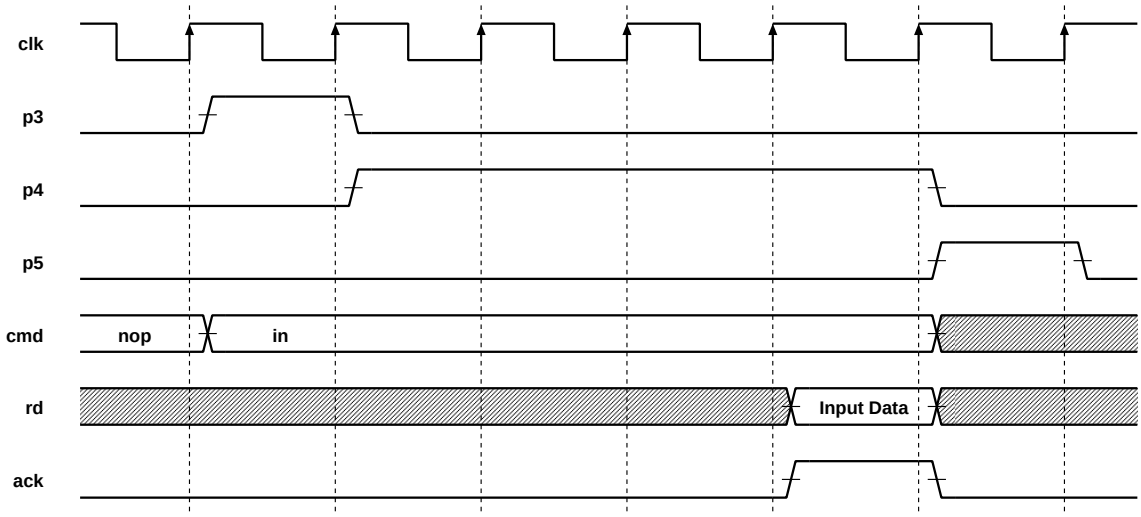

図 4: IN 命令の実行

ただし in コマンドの実行には複数サイクルかかるので，ack がアサートされるまでパイプライン全体をストールさせる必要がある．

3 BIF の実装

本章では，2 章で述べたバス・インターフェイス回路 BIF の仕様に対する実装例を示す.

3.1 BIF の構成

図 5 に，BIF のブロック図を示す．同図中の，bif/ad と $IOREQ$ を入力とする AND ゲートは，bif/ad の各線に対して 1 つづつの 2-input AND ゲートを用意し，そのもう一方の入力にはすべて $IOREQ$ を接続することを意味する.

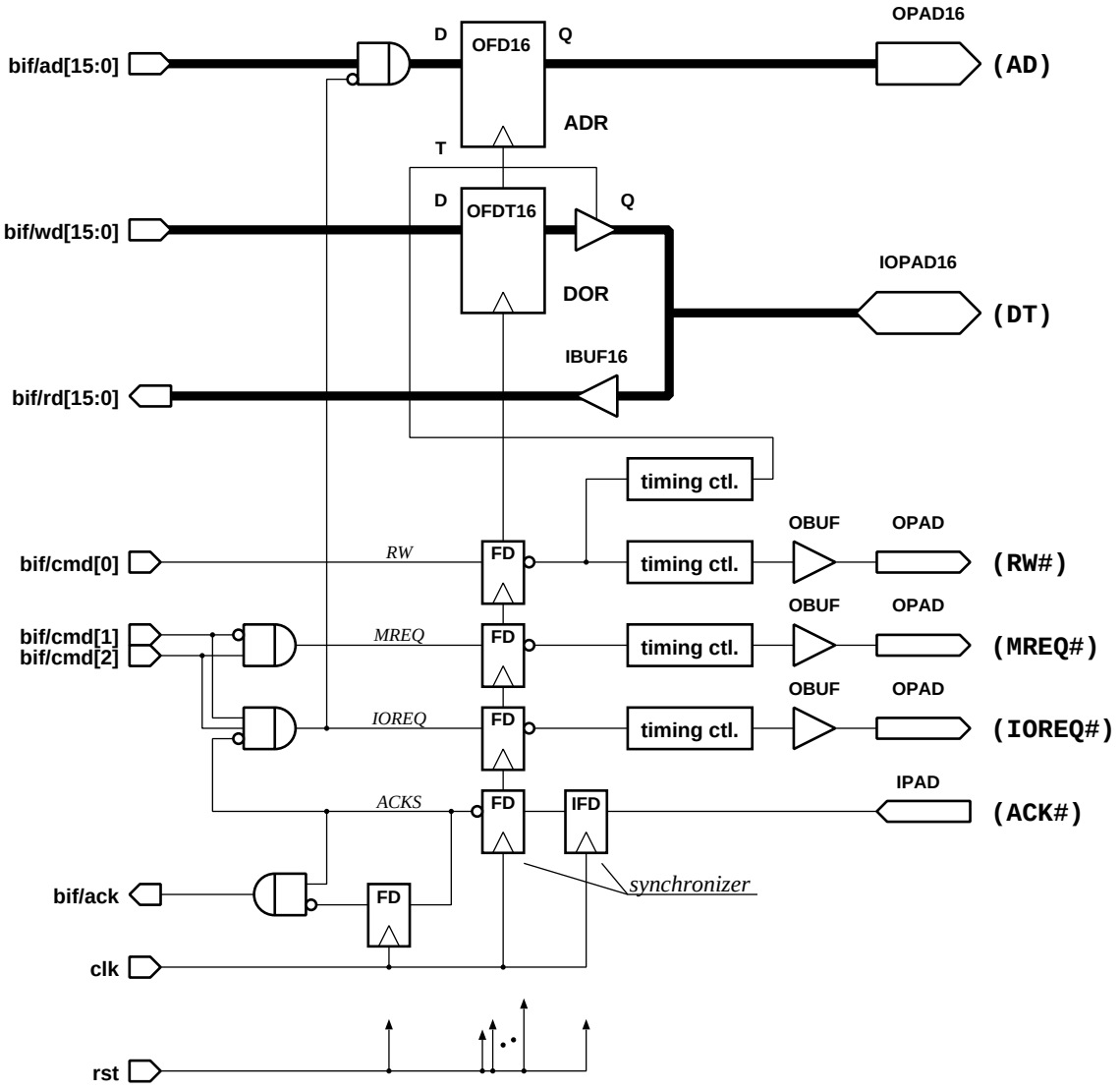

図 5: BIF のブロック図

バス周りの非同期的 I/F に対してハザードの無い信号を供給するため，バスに対する信号はすべて基本的に FF の出力を接続する．SIMPLE コアから BIF への入力は一旦 FF に蓄えられ，次のサイクルにバスに対して出力される.

3.2 制御信号の生成

表 2 などから，各制御信号は，基本的には，以下のようにすればよい：

$$\begin{aligned} RW &= \mathsf{cmd[0]} \\ MREQ &= \mathsf{cmd[2]} \cdot \overline{\mathsf{cmd[1]}} \\ IOREQ' &= \mathsf{cmd[2]} \cdot \mathsf{cmd[1]} \end{aligned}$$

上式に示されるように，$MREQ$ と $IOREQ$ は，基本的には，cmd[1] を入力，cmd[2] を出力イネーブルとする 1b バイナリ・デコーダの出力となっている．すなわち，この部分の論理設計さえ間違えなければ（例えば SIMPLE のコア部分にバグがあっても）**MREQ#**と**IOREQ#**は決して同時にアサートされることはなく，安全である．逆に，表 2 に示した割り当ては，そのような回路になるように決めたものである．

ただし in コマンドでは，他のコマンドとは異なり，**IOREQ#**と **AD** を複数サイクルに渡って維持する必要がある．次節では，in コマンドについて詳しく述べる．

3.3 in コマンド

図 6 に，in コマンド実行中の波形を示す．

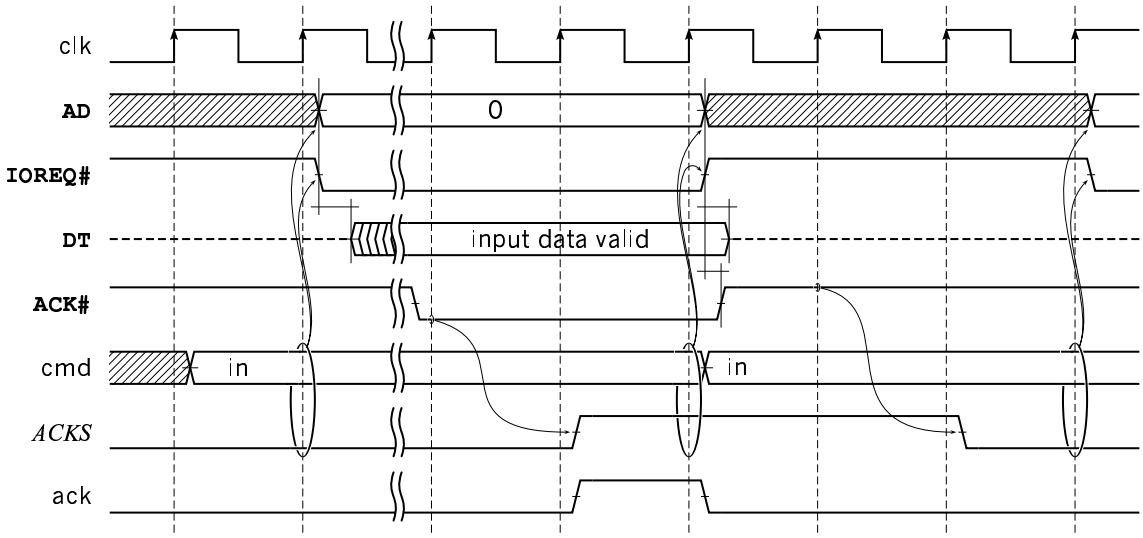

図 6: in コマンドの実行

$IOREQ$ は，**ACK#**がアサートされるまで high に保つ必要がある．前章で示した仕様では，ack がアサートされるまで cmd には in を維持することになっているので，$IOREQ$ は，単に，以下のようにすればよい：

$$IOREQ = \mathsf{cmd[2]} \cdot \mathsf{cmd[1]} \cdot \overline{ACKS}$$

図 6 に示すように，ack をアサートした次のサイクルで再び in コマンドが入力されても，**ACK#** がディアサートされるまで **IOREQ#** はアサートされない．このようにしてハンド・シェイクが成立する．

ただしこの実装では，out コマンドの直後に in コマンドを投入する場合に **IOREQ#** が連続してアサートされてしまうので，若干の修正が必要である．

図 5 に示した ack の生成回路は，$ACKS$ の最初の 1 サイクルだけアサートされるようにしたものである．この形の回路は，波頭微分回路と呼ばれ，よく使用される．

3.4 タイミング設計

MREQ#，IOREQ#，R/W#などの非同期の制御信号には，微妙なタイミングの調整が必要である．例えばライト・パルスは,「遅く出て，早く消える」必要がある[3].

3.4.1 変化タイミングの異なる信号の生成

図 7 に，そのような波形の生成の方法の 1 例を示す.

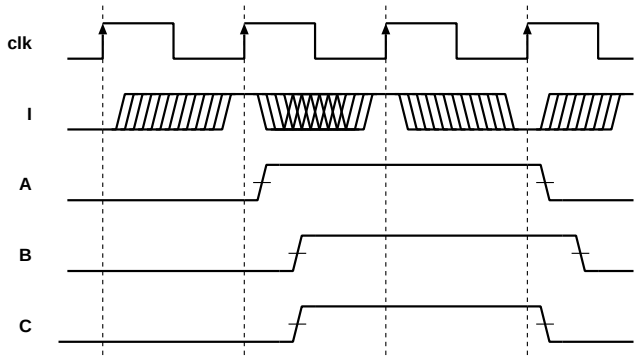

図 7: タイミングの調整

図 7 中，A は I を入力とする FF の出力であり，B は A を組み合わせ回路的に遅らせたものである．C は A と B の AND をとったもので,「遅く出て，早く消え」ている．C は組み合わせ回路の出力ではあるが，ハザードが生じないのは明白であろう.

Xilinx Foundation シリーズにおいて A から B を生成するには，内部バッファ BUF を用いる．ただし，BUF を設けただけでは論理最適化の段階で削除されてしまうので，BUF の入出力両方のネットに “X” (explicit) 属性を付ける必要がある.

なお上図では，2 サイクル連続でアサートされる I に対して，2 サイクルで 1 つのパルスを生成している．1 サイクルのパルスを 2 つ生成するためには，別の方法を用いる必要がある.

3.4.2 具体的なタイミング設計

具体的なタイミング設計に関しては，本稿では詳しくは述べない．ただし基本的には，MREQ#と IOREQ#が,「十分遅く出て，できるだけ早く消える」ようにすれば，他の信号の変化タイミングは，与えられた制約から自動的に決定できるであろう.

なお，OBUF に対して “FAST” 属性を付与すれば，遅延を更に小さくすることができる.

4 動作試験の方法

本章では，BIF の実機における動作試験（テスト）の方法について簡単に触れる．

テストのために BIF の回路に変更を加えては，何をやっているのかよく分からない．外部に適当な回路（glue logic と言う）を加えることによって，できるだけ実際の稼働状況に近いテスト環境を実現することが肝要である．

もっとも簡単なテストの方法は，BIF の SIMPLE コア I/F のすべての信号を *KITE* ボードの適当なトグル・スイッチ，LED に接続することである．つまり，人間が SIMPLE コアの代わりをすればよい．トグル・スイッチで BIF への入力を適当に設定した後に clk を投入すれば，動作周波数以外の点で実際の稼働状況と同じ環境を再現することができる．

ただし，以下の点に注意する必要がある：

- clk にはグリッチがあってはならないので，*KITE* ボードのステップ実行機能を用いるとよい[3].
- その他の入力にはグリッチがあっても構わない．メタ・ステーブルも起こり得ないので，同期化回路も必要なく，トグル・スイッチの出力をそのまま接続すればよい．ただし，ACK#の入力には注意が必要である．

参考文献

[1] 計算機科学実験及演習3 SIMPLE設計資料，情報工学教室SIMPLE担当グループ，1995.

[2] *KITE* マイクロプロセッサボード PLUS$^+$ 取扱説明書 Version 1.00.

[3] 計算機科学実験及演習3 ハードウェア *KITE* マイクロプロセッサボード PLUS$^+$ 取扱説明書 Version 1.00（補足），情報学科 計算機科学コース 計算機科学実験及演習3 ハードウェア担当, 1999.

[4] XC4000E and XC4000X Series Field Programmable Gate Arrays, http://www.xilinx.co.jp/partinfo/4000.pdf, j_4000_1_4.pdf, Xilinx Corp., 1999.

[5] Xilinx Foundation シリーズ オペレーション・マニュアル, 大倉エレクトロニクス.

[6] 計算機科学実験及演習3 ハードウェア Xilinx Foundation シリーズ オペレーション・マニュアル（補足），情報学科 計算機科学コース 計算機科学実験及演習3 ハードウェア担当, 1998.

[7] 計算機科学実験及演習3 ハードウェア 中間報告書問題, 情報学科 計算機科学コース 計算機科学実験及演習3 HW 担当, 1999.