JUMP–1
メモリコントローラ仕様説明書

ＪＵＭＰ－１設計グループ

Revision History

0.0	'95 5/1	K. Hiraki	原案作成
0.1	'95 5/15	K. Hiraki	MBP なし版作成
1.0	'95 7/24	K. Hiraki	動作仕様版
1.1	'95 8/12	K. Hiraki	DSP とのインターフェイス仕様
1.2	'95 8/20	K. Hiraki	MCIF とのインターフェイス仕様
2.0	'95 10/8	K. Hiraki	MMC ピン削減のための変更, 当面の最終案

1 概論

JUMP-1拡張メモリコントローラ（以下 MMC と呼ぶ）は並列計算機 JUMP-1 の処理クラスタ内において、Memory-Based Processor が完成するまでのつなぎとして使用するメモリコントローラであり、仮 Memory-Based Processor の一部をなす。

本仕様書では、共通バス、メモリモジュールおよび仮 Memory-Based Processor とのインターフェイス条件を中心に拡張メモリコントローラ設計・製作の仕様を明確化する。

仮ＭＢＰは主な目的がＭＢＰ以外のハードウェアのデバグ、基本ソフトウェアの実装とデバグにあるため、その早期な実現が最も重要な要請事項である。従って、基本的機能だけを備え、JUMP-1 に特徴的な多くの機能、例えば細粒度プログラム実行機構、２階層コンシステンシ維持機能などは接続されるネットワーク・コントローラのソフトウェアで実現される。仮ＭＢＰが直接実現する主な機能は下記の通りである。

メモリモジュールとのインターフェイス キャッシュメモリからの要求とメモリモジュール間のインターフェイスを実現する。メモリモジュールには同期タグを用いたメモリ・ベース同期機構および語毎のカーネル属性指定、コンシステンシ維持のための ack カウント機能等が付随するが、ここでは基本的な read / write / swap およびクラスタ内のコンシステンシ維持だけをハードウェアで実現する。。直接メモリアクセスではサービスできない要求に関しては、バッファに取り込み、ネットワークコントローラのソフトウェアの管理を受ける

ネットワークとのインターフェイス メモリ上に展開されているパケットイメージをネットワークコントローラ経由で RDT ルータに送り、転送するとともに、RDT ルータから流れてくるパケットをメモリ上に格納し、CPU 等へ通知する

STAFF Link とのインターフェイス 高速シリアルリンクである STAFF Link とメモリ上の情報との通信を実現する。

キャッシュコントローラへのインターフェイス ４個のキャッシュコントローラ へ store 操作が実行中であることを示す X_PEND 信号を各々出力する

仮ＭＢＰは、上記機能を実現するために必要最小限なハードウェアとして SDRAM・SRAM モジュール、共通バスと接続するMMC、バッファ群(MCIF, RDTIF)、ネットワークコントローラ（DSP）およびネットワークからのアクセスポートにより構成される。図１はブロックレベルの仮ＭＢＰハードウェア構成を示すものである。

各構成要素は、ゲートアレイであるMMC を除き、FPGA、off the shelf DSP およびメモリにより構成される。各部構成および機能の概要は以下の通りである。

メモリモジュール SDRAM および小量の SRAM で構成されるクラスタメモリ本体。１モジュールあたりアクセス幅６４ビット（タグ４ビットおよび ECC 約８ビットを除く）３２ M Byte であり、アクセス幅１６ビット、２ M Byte の SRAM を高速タグ用として持つ。なお、MMC としての機能では、タグ２ビットおよび高速タグ８ビットをアクセスする

MMC メモリコントローラ。SDRAM ／SRAM のタイミング発生とリフレッシュ管理、ECC 付加と訂正の基本的メモリコントローラ機能に加え、共通バスとのインターフェイス機能、パケットを一時保持するバッファ機能および MCIF への接続機能を持つ。ASIC(ゲートアレイ)で構築される。仮ＭＢＰの実装で最も大きな影響を受ける部分である

DSP ネットワークインターフェイスを簡易に実現するためのプログラマブル・コントローラ。TI 社製 TMS320C40 を使用する

MCIF DSP と MMC およびメモリモジュールとのインターフェイス。DSP 側はメモリバス（LBUS）に接続され、MMC 側では独自のポートに接続される。常に DSP 側がマスタとなってアクセスする。FPGA （１個）を用いて実装する

RDTIF RDT ルータチップと DSP ／メモリとのインターフェイス。RDT では１パケット（長さ１６以下）は１連続のタイミングで送る必要があるため、１組以上のパケットバッファを持つ必要がある。DSP 側は、DSP のローカルメモリが接続されている GBUS に接続されている。FPGA を用いて実装される

SRAM DSP が命令格納／作業用に使用する高速メモリ。0.5 M Byte 実装する

なお、クラスタボード中には仮ＭＢＰは１個だけ実装されることに注意されたい。

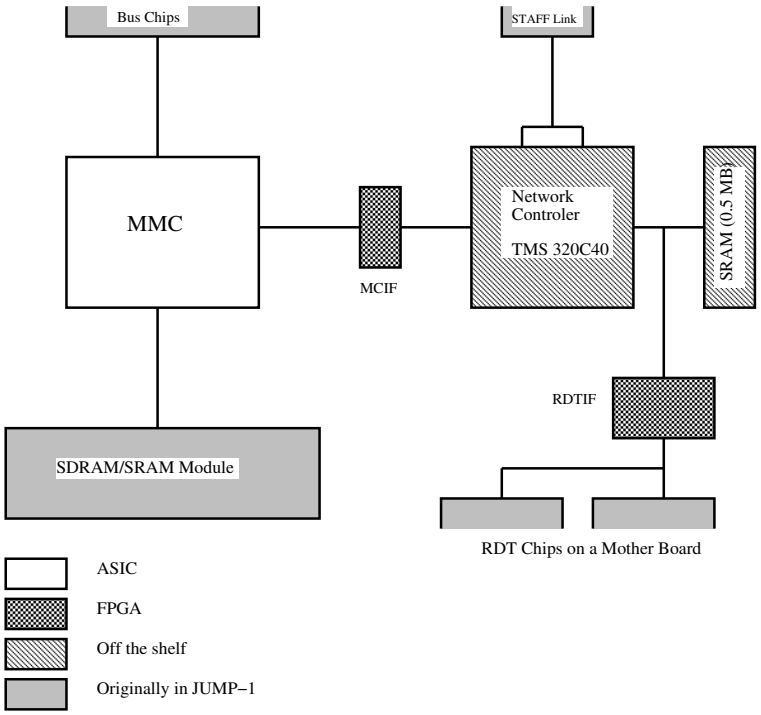

図 1: 暫定的ハードウェア構成

1.1 基本的事項

1.1.1 Endian

JUMP-1 は、CPU、メモリ、バス等全ての項目について、SPARC Architecture に従うと同じ意味の Big Endian アドレス付けを用いる。すなわち、**byte, half word, word については、Big Endian であり、 bit および double word 以上の大きさについては Little Endian である。**また、word 内ビットは常に MSB （場合によっては sign bit）が 最大のビット呼称であり、LSB が最小のビット呼称である。従って、ビットアドレスが要求される記述では、ビット呼称と、ビットアドレスが一致する。図?? は、ＪＵＭＰ－１におけるビット呼称、ビットアドレス、バイトアドレス、half word アドレス、 word アドレス、 double word アドレスを示すものである。

本仕様書説明中では、混乱を避けるため、ビット呼称には ［］で囲み、ビット位置、バイト位置などは囲まれない数字で記述する。また、ビットフィールドの指定は全てビット呼称で統一し、［］内の並び順は Ｘ［Ａ：Ｂ］とＸ［Ｂ：Ａ］を全く混同して使用している。

1.2 デバイス番号

次に、デバイス番号についての convention を定める。キャッシュコントローラの番号は、CPU 番号と同一である。

バスチップは 4 個に分割されるが、番号は ハーフワードのアドレス順と一致する、すなわちデータ double word の MSB 側が バスチップ 0 であり、LSB 側がバスチップ 3 である。

仮ＭＢＰを使用する場合は、１個しかないため、全てのリクエストはＭＢＰ ０が対象である。

1.2.1 信号名

本仕様書で扱う信号線名の記法として、基本機能名を SIG_NAME とすると、論理の極性を接頭辞 （負論理信号で実現する場合 X_SIG_NAME と命名）、信号の方向性を接尾辞（入力信号を SIG_NAME_I 、出力信号を SIG_NAME_O 等と命名）とした慣用を使用する。

2　メモリの構成

本節では、MMCおよび仮ＭＢＰと接続する場合のメモリモジュール構成ついて述べる。なお、仮ＭＢＰ版ではメモリ、特にSRAMの使用方法がＭＢＰ版と異なっているため注意が必要である。

MMMは、メモリモジュール（またはその複数）であり、一枚のモジュールは１組の（２MエントリのSDRAM（読み出し幅８０ビット）、５１２KエントリのSRAM（読み出し幅１６ビット）およびクロック、制御信号線、アドレスをドライブするLVTTLドライバ）で構成される。すなわち、１６MBのDRAM、１MBのSRAMである。メモリモジュールのクロックにはPLLを用いない。なお、設計、特に実装設計の簡略化のために仮ＭＢＰでは最大１組の実装を行なう。[1]

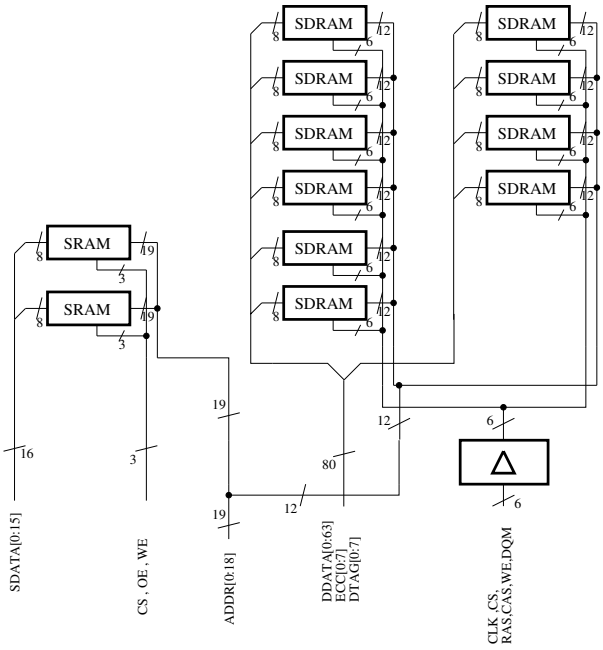

図 2: JUMP-1 メモリ・モジュールの構成

[1]MMMのアクセス幅はＭＢＰ版実装における必要性から決定されているが、MMCによる実現では、SDRAM７４ビットおよびSRAM８ビットのみが使用される。）

クラスタメモリのフィールド

フィールド名	ビット数	記述
DATA[63:0]	64	いわゆるデータ倍語（ダブルワード）。1ブロックは4データ倍語で構成される。ECCによる1ビット誤り訂正、2ビット誤り検出機能が付いている。
ECC[7:0]	8	データ語、タグフィールドに対するECC用パリティビット。ECCパリティビットが全体に掛かっている構成上、ワード以下の変更は、当該部分があるワードのECCフィールド全体を書き直す必要がある。
WAIT[1:0]	2	当該ダブルワードの待ち合わせ状況を保持する同期タグ。MMCのピン数制限からWAITフィールドはパリティによる誤り検出のみ行なう。2ビットのビットパターンは以下の表に従う：
DPARITY	1	DTAG部分（2ビット）のパリティ。DTAG部分はDATA部分と独立な書き込み制御が必要なため、独立したパリティを持たせる
FE[3:0]	4	ブロック内各ダブルワードのFull / Empty を示す。FE[0]はブロック中最も小さいアドレスを持つダブルワード、FE[3]がブロック中最も大きいアドレスを持つダブルワードのFull / Empty 状態を示す。Full / Empty の極性は リクエスト中のFEPOLビットの値で決定される
GLOBAL	1	当該ブロックがクラスタ間で共有されていることを示す。DSPが設定し、DSPまたはMMCが読み出す
VALID	1	当該ブロックが有効であるることを示す。DSPが設定し、DSPまたはMMCが読み出す
RD_TRAP	1	当該ブロックがMMCによりreadアクセスされると無条件でDSPへ割り出しが発生し、制御を渡すことを指定する
WRT_TRAP	1	当該ブロックがMMCによりwriteアクセスされると無条件でDSPへ割り出しが発生し、制御を渡すことを指定する

WAIT[1]	WAIT[0]	State	Comment
0	0	非WAIT状態	当該語の read / write は何も待っていない
0	1	R-wait 状態	当該語の readでＤＳＰによるサービスが必要である
1	0	W-wait 状態	当該語の writeでＤＳＰによるサービスが必要である
1	1	RW-wait 状態	当該語の read および writeでＤＳＰによるサービスが必要である

2.1 メモリ・アドレス

MMC に内蔵または接続されているアクセス対象には全てアドレスが付加されている。また、MMC へは、共通バスおよびDSP からアクセスが行なわれ、特に共通バスからは クラスタアドレス空間とコントロール空間の二種のアドレス体系が用いられ、下記3種のアドレスを使用することとなる。

クラスタアドレス 共通バスからの通常アクセスに用いられるアドレス。バイトアドレスであり、２８ビットアドレスに、 HOME, DUMMY, CH_OW, LARGE の4ビット属性を付加したものである。なお、クラスタアドレスを用いる場合には、DTAG、 STAGアドレスはクラスタアドレスをブロック（３２バイト）単位に変換したものとしてアクセスされる。なお、クラスタアドレスの詳細は「ＪＵＭＰ－１共通バス仕様書」を参照のこと

コントロールアドレス コントロールアドレスは、共通バスから control_read_req および control_write_req によりアクセスされる場合に使用されるアドレスである。コントロールアドレスは、バイトアドレス（実際には下位2ビットは常に0）であること、アクセスモードがないことを除き、下記 DSP アドレスと一致する。

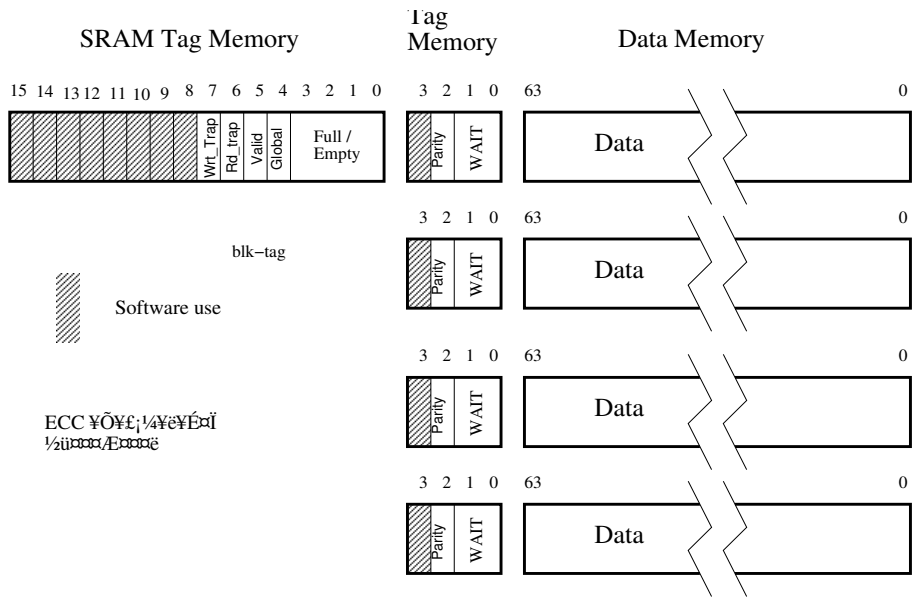

図 3: クラスタメモリのフォーマット

DSP アドレス DSP アドレスは、DSP が MCIF を介してアクセスする場合に用いられるアドレスである。DSP からは、全てのアクセスが word 単位で行なわれるため、３２ビット長の word アドレスを使用する。ただし、最上位ビットは常に 0 であるため、実質３１ビットが使用可能である。DSP からアクセスする場合は、DATA, DTAG, STAG, ECC , MCC 内部資源を別個のアドレスとしてアクセスするため、アドレスの上位で区別する。また、アクセス単位がブロックであるか、ワードであるかを区別するため、アドレス上位にモード・フィールドを持つ。アドレス割り付けの詳細は後述する。

なお、図 4 は上記アドレスの関係を示すものである。

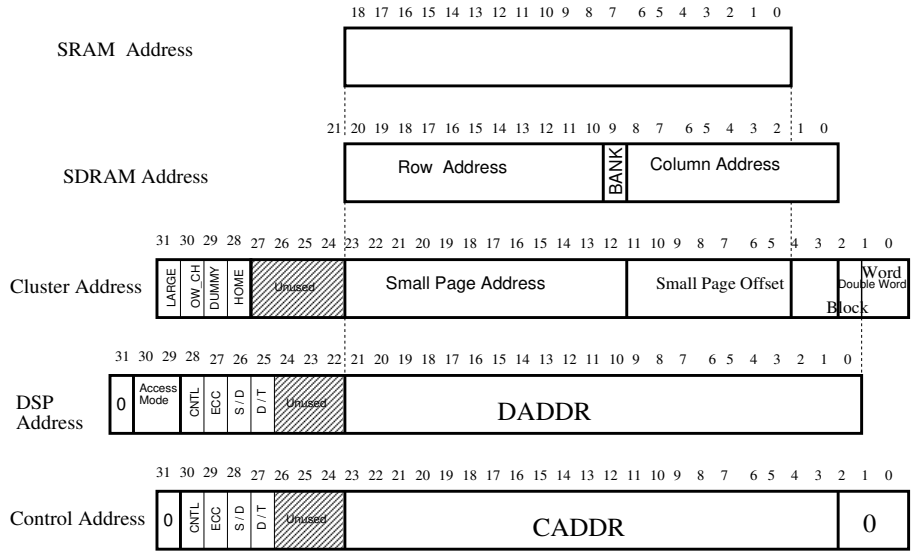

図 4: メモリ・アドレス

2.2 DSP アドレスの詳細

DSP アドレスは、DSP からのアクセスに用いられる３２ビットアドレスである（実際には、最上位ビットは0である必要があるため、３１ビットである）。DSP からは常に word 単位でアクセスされるため、アドレスはバイトアドレスの下位2ビット分シフトし、最下位ビットがワード指定となる。

クラスタメモリのフィールド

フィールド名	ビット数	記述
DADDR[22:0]	23	ワード単位の DSP アドレス本体。下記 CNTL ビット が 0 の場合、D/T, S/D, ECC ビットの指定により、DATA、DTAG、STAG、ECC メモリを指定する。これらの場合、アドレスは全てワードに対応した位置に置かれる（すなわち、DTAG、ECC では下位１ビットが0であり、STAG では下位３ビットが0である）。CNTL ビットが１の場合、MMC 内部レジスタを指定する
D/T	1	DATA と TAG の切替え。 0 の場合、DATA であり、１の場合 TAG である。D/T が１の場合、 DADDR[0]は don't care である。
S/D	1	SRAM と SDRAM の切替え。 0 の場合、SDRAM であり、１の場合 SRAM である。S/D が１である場合、上記 D/T は１である必要がある。S/D が１である場合、 DADDR[2:0]は don't care である
ECC	1	SDRAM の DATA と ECC の切替え。 0 の場合、DATA であり、１の場合 ECC である。ECC が１である場合、上記 D/T は 0 、S/D は 0 である必要がある。ECC が１である場合、 DADDR[0]は don't care である。
CNTL	1	メモリ空間と MCC 内部レジスタ空間の切替え。0 の場合、上記 D/T, S/D, ECC で指定されるメモリ空間であり、１の場合 MCC 内部レジスタ空間である。内部レジスタ空間は、入力バッファ、出力バッファ、read/modify/write 用レジスタ、 エラーレジスタを含む

なお、内部レジスタに関するアドレスの割り付けは、MMC チップ設計後に決定される。

2.3 メモリタイミング

本節では、SDRAM、SRAM 関係のアクセスタイミングを述べる。まず最初に、設計に際しての仮定をのべる：

1. システムは５０ MHz 単一で動作し、全て同期している
2. SDRAM は CAS Latency=2 で動作する
3. SDRAM のパワーダウンモードは使用しない
4. SDRAM では次のモードを使う：

Mode	Selection
Latency Mode	2
Wrap Type	Sequential
Burst Length	4

5. リフレッシュは MMC チップが行なう。従って、リフレッシュ後の不感時間における共通バスまたは DSP からのアクセスは遅延する。
6. 単純な一定時間毎に、リフレッシュを行なう
7. DSP からの要求は常に受理される。ただし、 write busy 中の write リクエストは、完了するまで待つ必要がある
8. ECC の発生／検査には１クロックを要する
9. SDRAM において、同一 row アクセスによる read/modify/write を使用しない
10. SRAM において Read と Write とのオーバーラップは考えない
11. SRAM において read/modify/write は使用しない
12. SDRAM とのデータ信号線の方向性を切替える場合、１クロックスペースを空ける
13. MCIF との方向性を切替える場合、十分な間隔を空ける

残念ながら、ブロック長が 4 であるため、A ／ B バンクを切替えながら連続使用することが不可能である。従って、パイプライン動作による高速化を図るためには、アクセスの前後関係を見、そのバンク／ページ配置によりタイミングを決定する必要がある。

タイミングの最適化として、ページモードを活用して連続する同一 row address に対してプリチャージなしでアクセスすることが可能であるが、JUMP-1 における実現では使用しない。

次に、データとタグの関係について述べる。JUMP-1 では DRAM に各 double word 毎２ビット＋パリティとして付くタグ(DTAG) と、SRAM に８ビットとして付く高速タグ(STAG)の２種が存在することは既に述べた。SRAM 部分については、SDRAM と別個にアクセス可能である。高速タグを変更する場合は、DSP から書き込むことで実現し、 read/modify/write に対応するアクセスは、ブロックにロックをかけ、DSP から読み出し、次いで書き込むことで実現される

SDRAM が実現する２ビットタグは、バスからのアクセスについては常に read only として扱われ、書き込みは DSP からのアクセスで行なわれる。

2.4 共通バスからのアクセスプリミティブ

MMC が共通バスから受理し、直接メモリを操作するアクセスプリミティブは後節で詳述するが、概要は次の通りである：

read_req, n_c_read_req, allread_req, rdinv_req SDRAM の DATA、DTAG と STAG の FE フィールドを読み出し、リプライパケットとして返送する。STAG の TRAP ビットが立っている場合、リプライパケットを送らず DSP 処理を割り込みで呼び出す

writeback_reply、（blockの）n_c_write_req SDRAM のDATA、STAG のFE に書き込む。STAG のTRAP ビットが立っている場合、DSP 処理を割り込みで呼び出す。

write_req, n_c_write_req SDRAM のDATA に書き込む。STAG のTRAP ビットが立っている場合、DSP 処理を割り込みで呼び出す。書き込み幅がダブルワード未満の場合、read/modify/write で実現する

control_read_req コントロールアドレスに従い、DATA、DTAG, STAG, ECC, MCC 内部情報を取りだし、リプライパケットで返送する。決してDSP 割り込みは発生しない

control_write_req コントロールアドレスに従い、DATA、DTAG, STAG, ECC, MCC 内部情報のいずれかに、パケット上のデータを書き込む。決してDSP 割り込みは発生しない

2.5 DSP からのアクセスプリミティブ

クラスタメモリが受理するDSP からの基本プリミティブは以下の通りである。なお、これら基本プリミティブを組み合わせることにより、更に拡大した範囲の動作を実現することも可能である。

なお、この項目の仕様は大幅に変更される予定である

2.5.1 基本プリミティブ

read アドレスで指定された1ダブルワード を読み出す。ダブルワード以下の長さのデータの読み出しに用いられる。タイミングとして、まず当該 row が アクティベートされ、1ワード分の読み出しが行なわれる。ただちにプリチャージすることにより読み出しを中止する。

wrt_word アドレスで指定されたsingle word に対するwrite 操作。まず当該 word が アクティベートされ、読み出され、次に書き込みを行ない、書き込みが終了した時点でプリチャージが行なわれる。

refresh SDRAM に対してリフレッシュ開始を指示する。

init SDRAM およびECC 部分のイニシャライズを行なう。メモリコントローラが実施するイニシャライズの内容は次の通りである。

1. Power On 時に １００μ秒の間、CKE とDQM をH に保持し、クロックを印加する
2. 両バンクを A10=H のアドレスでプリチャージする。3クロック待つ
3. モード・レジスタ・セット・コマンドを用いてSDRAM のモードを設定する。モードはアドレス A11 — A0 が 0x22 として設定する
4. モード・レジスタ・セット・コマンドを発行した2クロック後、CBR リフレッシュコマンドを7クロック間隔で2回発行する

rdblk 1ブロック（３２バイト）の連続呼び出し。通常の場合、当該row がactivate され、与えられたアドレスを含む1ブロックが読み出される。出力順は与えられたアドレスから巡回する順番である。

各読み出しデータは、MCC チップによりECC チェックと必要な場合の訂正が行なわれ、常に正しいデータとしてDSP に送られる。rdblk コマンドにはオートプリチャージ機能が付随し、2番目のデータをアクセスする時点でプリチャージが自動的に開始する。

同時に、SRAM のデータが読み出され、DSP に送られる

wrtblk 1ブロックの連続書き込み。まず当該row がアクティベートされ、与えられたアドレスを含む1ブロックが書き込まれる。書き込まれる順は、与えられたアドレスから巡回する順番である。

各書き込みデータに対しては、必ずECC が付けられる。なお、ブロック最後のデータが出力されて2クロック後に自動的にプリチャージが開始される。

2.6 動作の詳細

クラスタメモリのアドレス配置を図4に示す。JUMP-1の仕様上、２８ビットクラスタ内物理アドレスは次のように分割される：

Block内アドレス １ブロックは３２バイト、すなわち4倍語である。キャッシュメモリ、RDTルータを通るパケットは１ブロック単位であることが通常である。

カラム・アドレス アドレスの3から１１ビット目までが1個のバンクのカラムアドレスである。すなわち、4Kバイトのページ内では新たなプリチャージ／アクティベートなしで連続アクセスが可能である。これはSuperSPRAC+のsmall pageの大きさに対応する

バンク・アドレス アドレスの１２ビット目。1個のSDRAMが持つ2個のバンクを識別する。リフレッシュ以外では異なるバンクのプリチャージ／アクティベートは別個独自に行なうことが可能である。

ロー・アドレス アドレスの１３ビットから２３ビット目までが、バンク毎のプリチャージ／アクティベートの単位であるロー・アドレスである。

SDRAMを使用するためには、目的物のあるROWを activateする必要があり、またactivateするためには、そのバンクが リフレッシュ間隔以内の時間にプリチャージされている必要がある。SDRAMでは、その特徴としてバンクが異なるページに対しては独立にプリチャージおよび activate 操作を行なうことが可能であり、また auto-precharge 機能として 読み出しまたは書き込み終了時に自動的にプリチャージを行なうことが可能であり、高速化に貢献する。

自動プリチャージおよびprecharge と activate のタイミングは以下の条件に従う（理解を助けるためリフレッシュのことは別に考慮するものとする）。

1. 引き続く2個のアクセスのbank addressと row addressが一致する場合は、プリチャージが必要でなく、直接readまたはwriteコマンドを発することが可能である。このためには、直前のコマンドがPR_REQ = 0で次のコマンドがACT_REQ = 0 の手順でコマンド列を発行することが必要である

2. 引き続く2個のアクセスのbank addressまたはrow addressが異なる場合、bank addressが異なれば他方のバンクがreadやwriteアクセス後のプリチャージと並行してactivateからread/writeへ行くことが可能である。この高速化はＭＢＰ側で早期に（activate）コマンドを発行することで実現する

3. bank addressが同じでrow addressが異なる2個の引き続くアクセスは、残念ながら直列的にプリチャージからactivateへ行く必要がある

従って、MMCから見たタイミングは、同じバンクが連続する場合と、異なるバンクが隣接する場合に分けて考える必要がある。（しかしながら、仮ＭＢＰにおけるMMCでは、同じバンクが連続することによる最適化は行なわない）

2.7 アクセスのタイミング

本節では各アクセス種について、投入およびデータ授受のタイミングについて述べる。全てのタイミングはアクセス種の与え方に対して決定的である。

2.7.1 rdblk アクセス

アドレスとactivateコマンドを同時に与える。＋6クロックで最初のデータが出力され、＋9クロックに最後のデータが出力される。全てのデータはECCにより検査／訂正されたものである。また、付属するSRAMタグは＋2クロックに出力され、SRAMタグ用書き込みデータは＋3クロックの途中(クロック開始から4 ns後程度から)から＋4クロックの頭にかけて出力する。詳細を図5に示す。

rdblkではプリチャージは＋7クロックに行なわれるため、バンクが同じ場合次のrdblkアクセスを＋8クロック以降で次のrdblkアクセスが出力可能である。また、バンクが異なる場合には＋4クロック以降で次のrdblkアクセスが出力可能である。その他の場合については、表参照のこと。

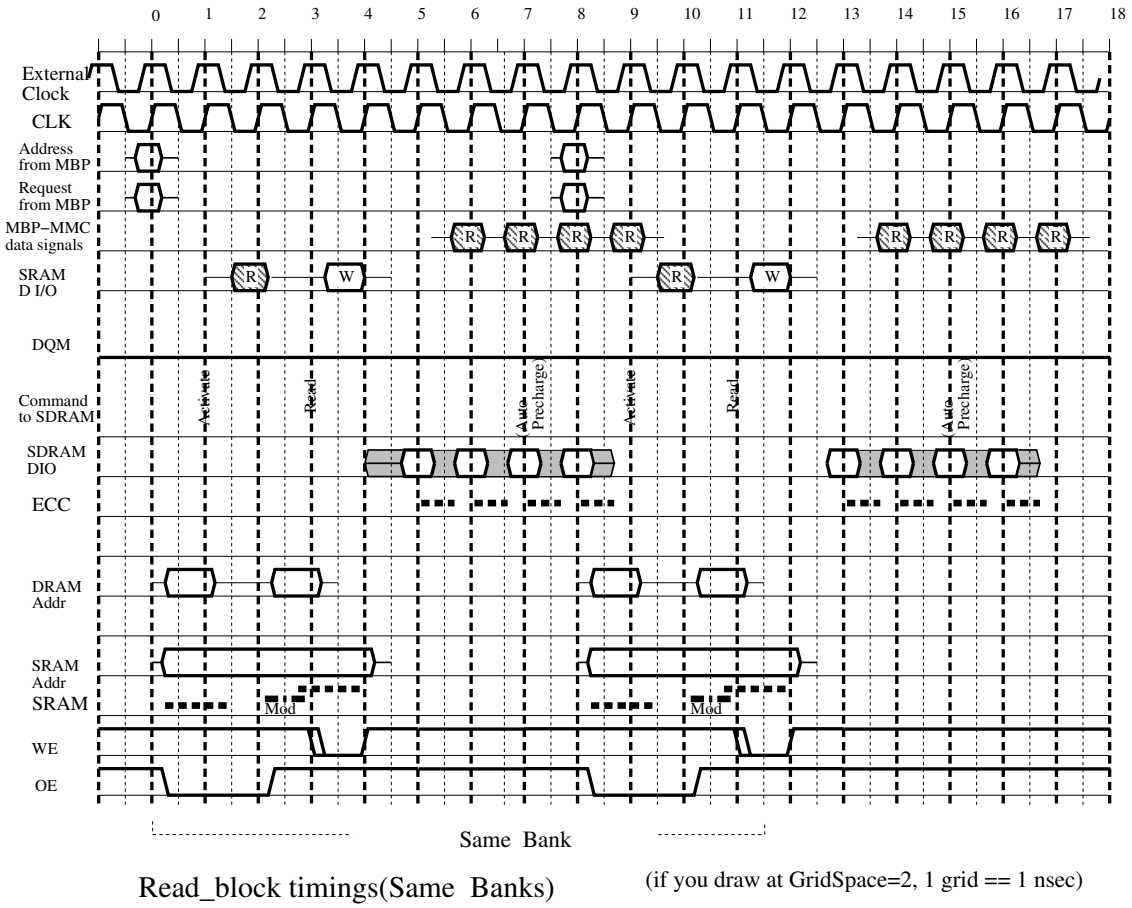

図 5: rdblk アクセス（同じバンク）のタイミング

2.7.2 wrtblk アクセス

wrtblk アクセスは、activate コマンドおよびアドレスを同時に与えることにより起動され、書き込みデータは＋1、＋2、＋3、＋4 クロックで与える。書き込みが行なわれた後、＋7 クロックで自動プリチャージが開始される。一旦書き込みが開始されると中断は困難であるが、例えば同じバンク、同じ row address に対して read または rdblk を開始すると以降の書き込みが中止される。

また、付属する SRAM タグは＋2 クロックに出力され、SRAM タグ用書き込みデータは＋3 クロックの途中(クロック開始から 4 ns 後程度から）から＋4 クロックの頭にかけて出力する。詳細を図 7に示す。

wrtblk ではプリチャージは＋7 クロックに行なわれるため、バンクが同じ場合次の wrtblk アクセスを＋8 クロック以降で次の wrtblk アクセスが出力可能である。また、バンクが異なる場合には＋4 クロック以降で次の wrtblk アクセスが出力可能である。また、直前のアクセス種が read または rdblk であった場合は、接続バスの方向を切替える必要があり、その間コマンド投入を待つ必要がある。その他の場合については、表参照のこと。

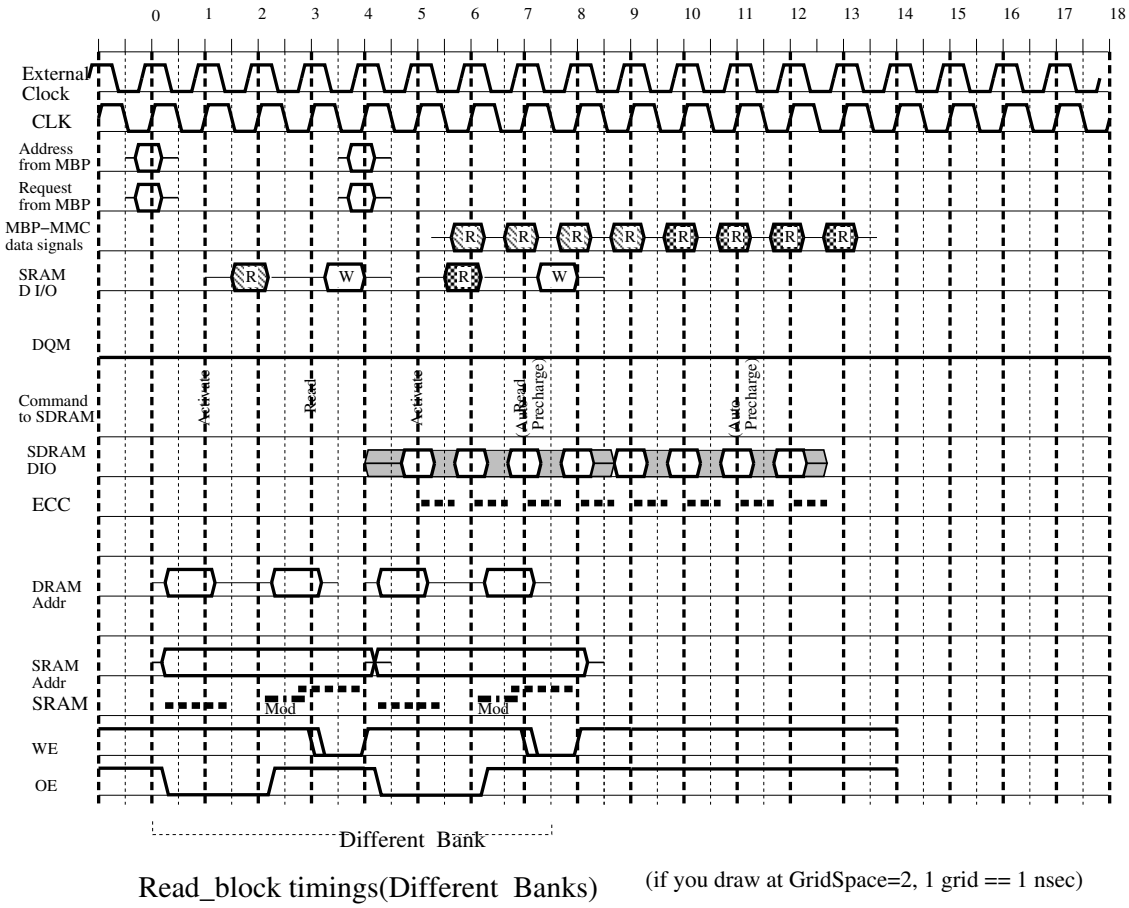

図 6: rdblk アクセス（異なるバンク）のタイミング

2.7.3 read アクセス

read アクセスは 1 ダブルワードだけの読み出しを行なう。タイミングは基本的に rdblk と同じであるが、次のアクセスを早期に開始出来る点が異なっている。 read では activate コマンドとアドレスを同時に与え、＋ 6 クロックで読み出しデータが利用可能である。また、次の read または rdblk が同じバンクが対象の場合、＋ 5 クロックで次の activate コマンドを発行可能であり、異なるバンクの場合には＋ 4 クロックで次の activate コマンドを発行可能である。

read アクセスの処理においては、自動プリチャージを用いず、明示的なプリチャージコマンド （PR_REQ = 1 の場合）または Burst Stop コマンド（PR_REQ = 0 の場合）で読み出しを終了させる点が rdblk と異なっている。従って、連続してメモリ要求を出す場合に余分な制約が入ることに注意する必要がある。

また、付属する SRAM タグは＋ 2 クロックに出力される。詳細を図 9に示す。

2.7.4 write アクセス

write コマンドは 1 ワードの書き込みを行なうコマンドである。タイミングは wrtblk と類似であるが、書き込みデータを 1 個しか与えない点が異なっている。特に、SDRAM に付属する同期タグは 0 側のワードにしか存在しないため、1 側のワード書き込みでは値を与えても無意味である。

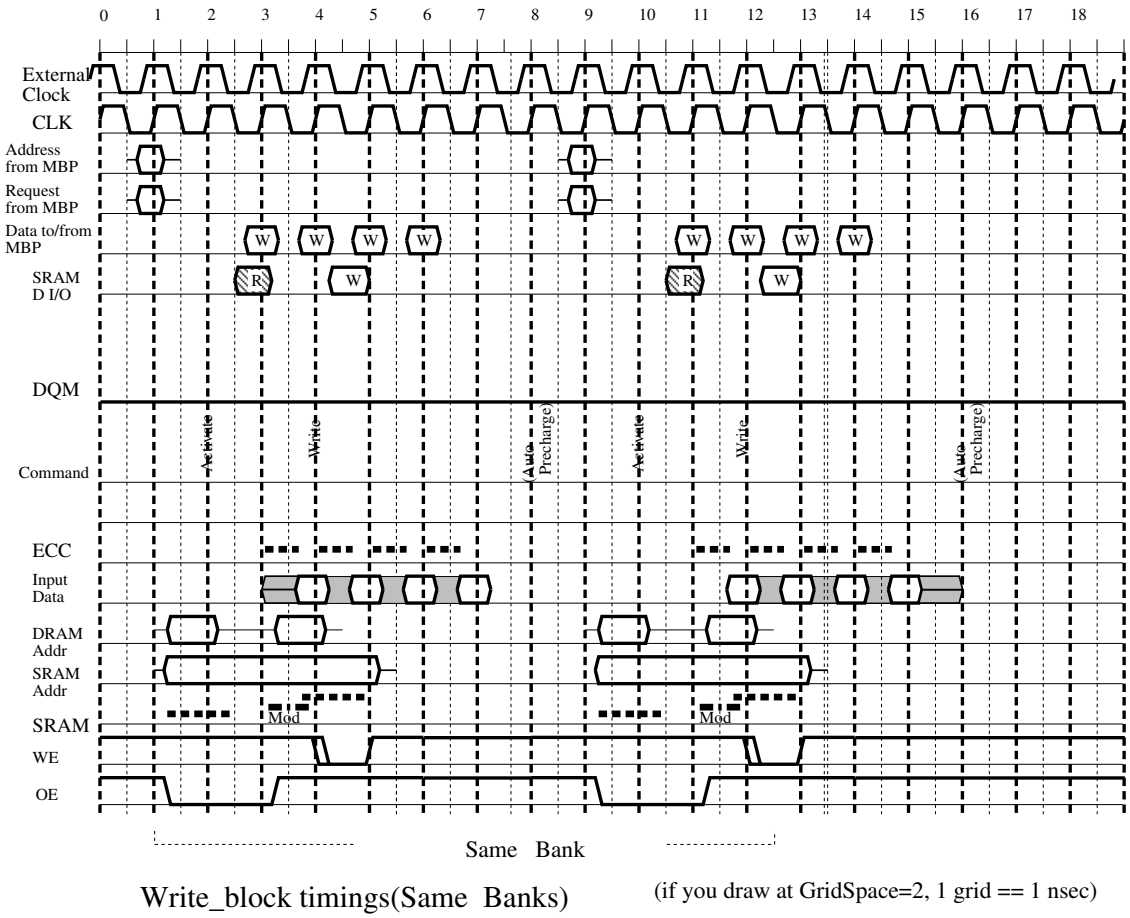

図 7: wrtblk アクセス（同じバンク）のタイミング

2.7.5 read / modify / write について

すでに示したタイミング図から明らかなように、読み出しに引き続く write コマンド発行は、接続の方向を反転するために 1 クロック待つ必要がある。従って、read の次の write は 同じバンクの場合 ＋6、異なるバンクの場合 ＋5 クロック以降に出す必要がある。従って、read / modify / write が一連となったコマンドを作成することは何らタイミング上の利益がない。従って、MBP から read と write コマンドを出すことにより実現する。同様の理由により、read と write の間の precharge / activate を排することも利益をもたらさない。

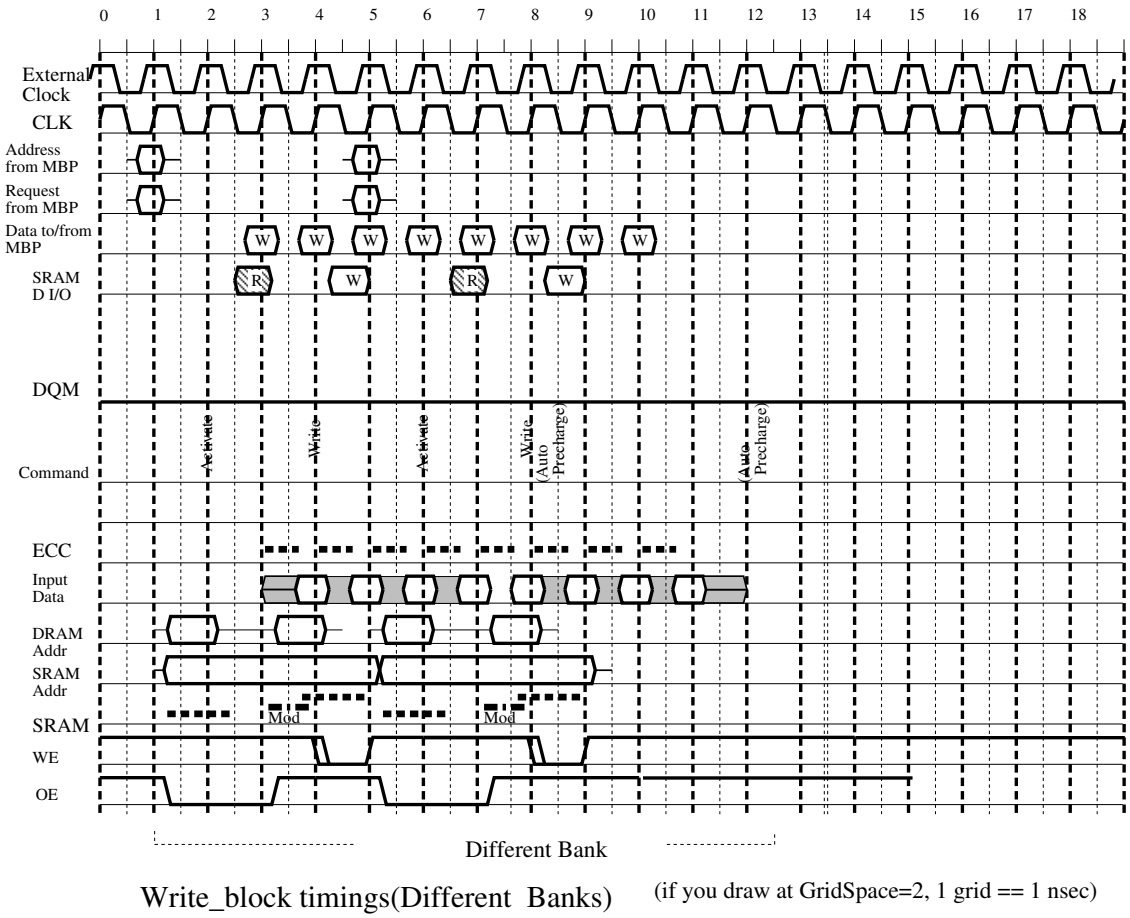

図 8: wrtblk アクセス（異なるバンク）のタイミング

2.8 参考図

ここからは、理解を助ける参考図である。後に説明を加える予定である。

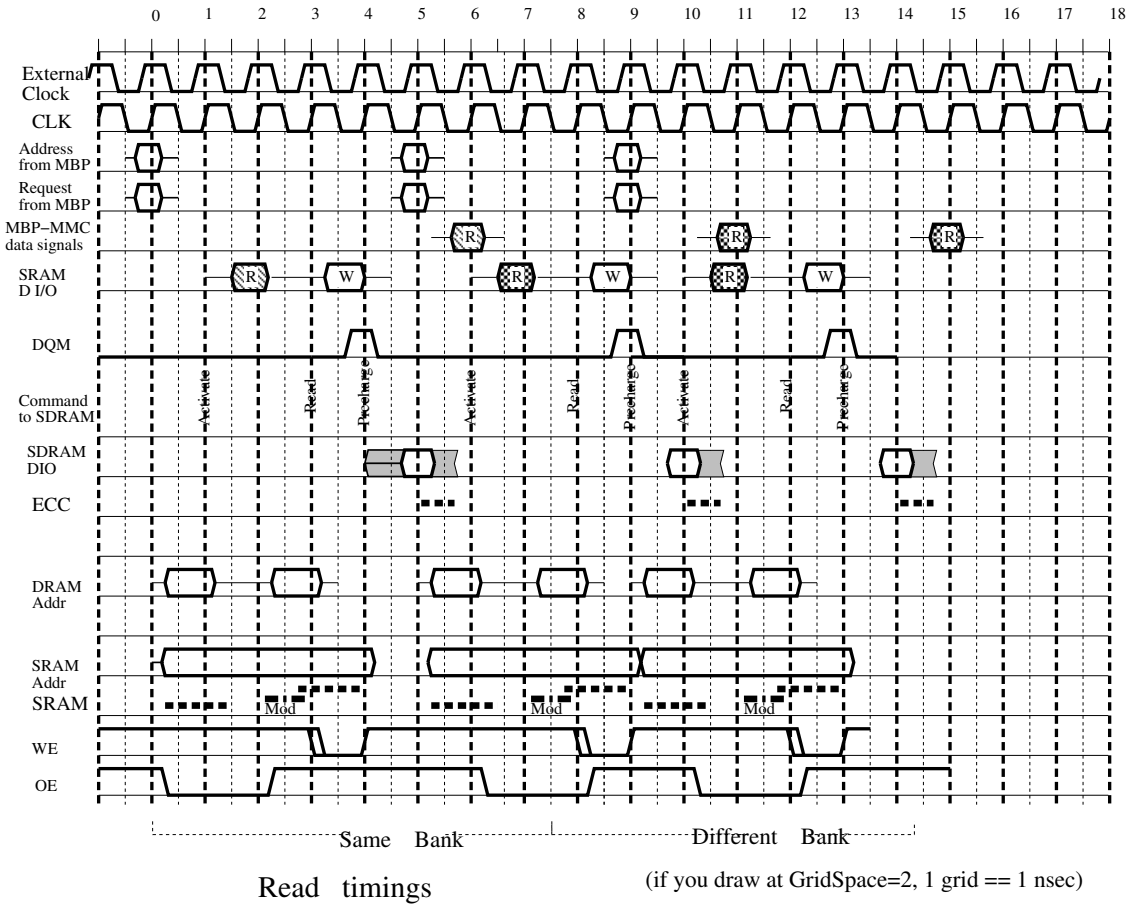

図 9: read アクセスのタイミング

3 MMC チップの構成

本節では、MMC チップの信号接続について述べる。なお物理的実現方式として MMC は 共通バスチップ（BC）、キャッシュバッファチップ（CB）と同一チップを使用するに留意する必要がある。

ここに示す MMC チップ信号線はバウンダリ・スキャンなどデバグ／メンテナンスに用いるものを含まない。これらの信号線はチップ外部仕様書において規定する予定である。**なお、V2.0 から、チップのピン制限により信号数が減少したことに注意すること**

信号の記述は、全て MMC チップを主体として見ている。すなわち、MMC チップへの信号が入力であり、MMC チップからメモリ／DSP などへの信号が出力である。従って、例えば共通バスチップへは、 xxx_O 信号がバスチップの xxx_I 信号へ、 xxx_I 信号は バスチップの xxx_O 信号へと接続される。また、正論理はＨが真であり、負論理はＬが真である。

まず、チップとしての共通な制御信号線を示す。これらは、チップが選択されるどの機能の状態にあっても、共通の目的で使用されるものである。

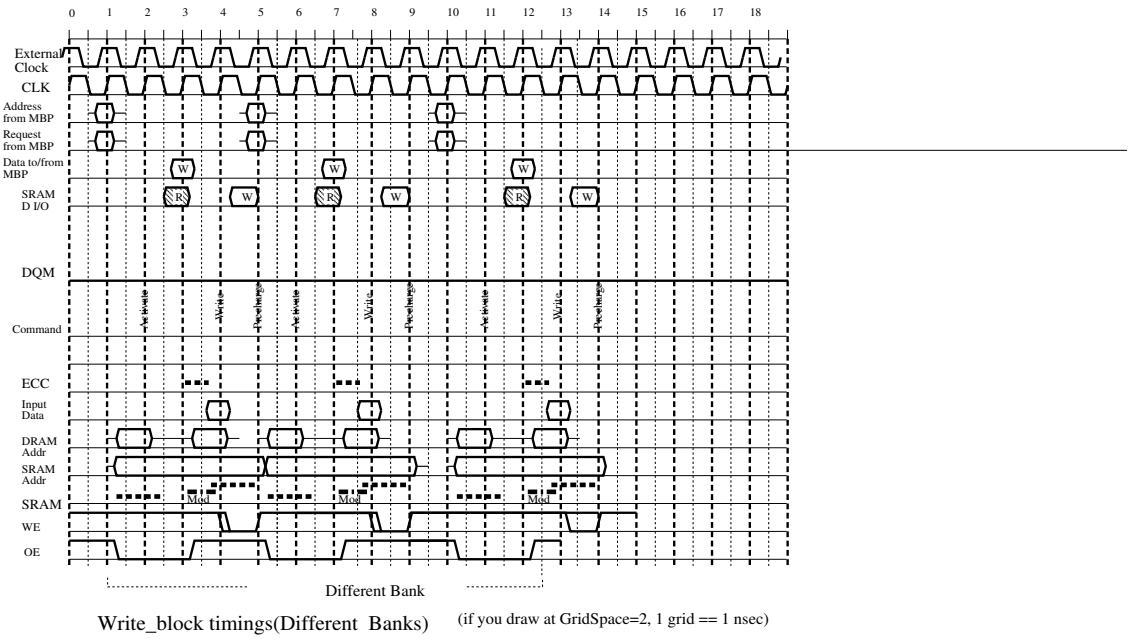

図 10: write アクセスのタイミング

MMC チップ共通制御信号線

信号名	本数	方向	レベル	記述
CLK	1	入力	正論理	MMC に対するクロック。５０MH z クロックを入力する。クロックはL→Hのエッジを用い、H→Lのエッジは利用しない。バスチップに対するクロックと同一周波数、同一位相である
X_RESET	1	入力	負論理	MMC チップに対するリセット信号。X_RESET はバスチップ各部分の初期化に使用され、X_RESET がLである期間に少なくとも２５６個 CLK によるクロックが加えられる必要がある。
FS	2	入力	正論理	バスチップの機能設定を行なう。バスチップは、FS により次の各機能を実現する。FS は、チップが論理動作を行なう期間、レベルが固定している必要があり、FS が変化した場合には X_RESET によりリセットをかける必要がある。 **0.** バスチップ。バスマスター用 **1.** バスチップ。バススレーブ用 **2.** キャッシュコントローラ用データバッファ **3.** MMC
SCLK	1	入力	正論理	バスチップ JTAB バウンダリ・スキャン用クロック入力
SRST	1	入力	正論理	バスチップ JTAB バウンダリ・スキャン回路のリセット信号
SMOD	1	入力	正論理	バスチップ JTAB バウンダリ・スキャン回路の動作モード指定
SI	1	入力	正論理	バスチップ JTAB バウンダリ・スキャンデータ入力
SO	1	出力	正論理	バスチップ JTAB バウンダリ・スキャン出力

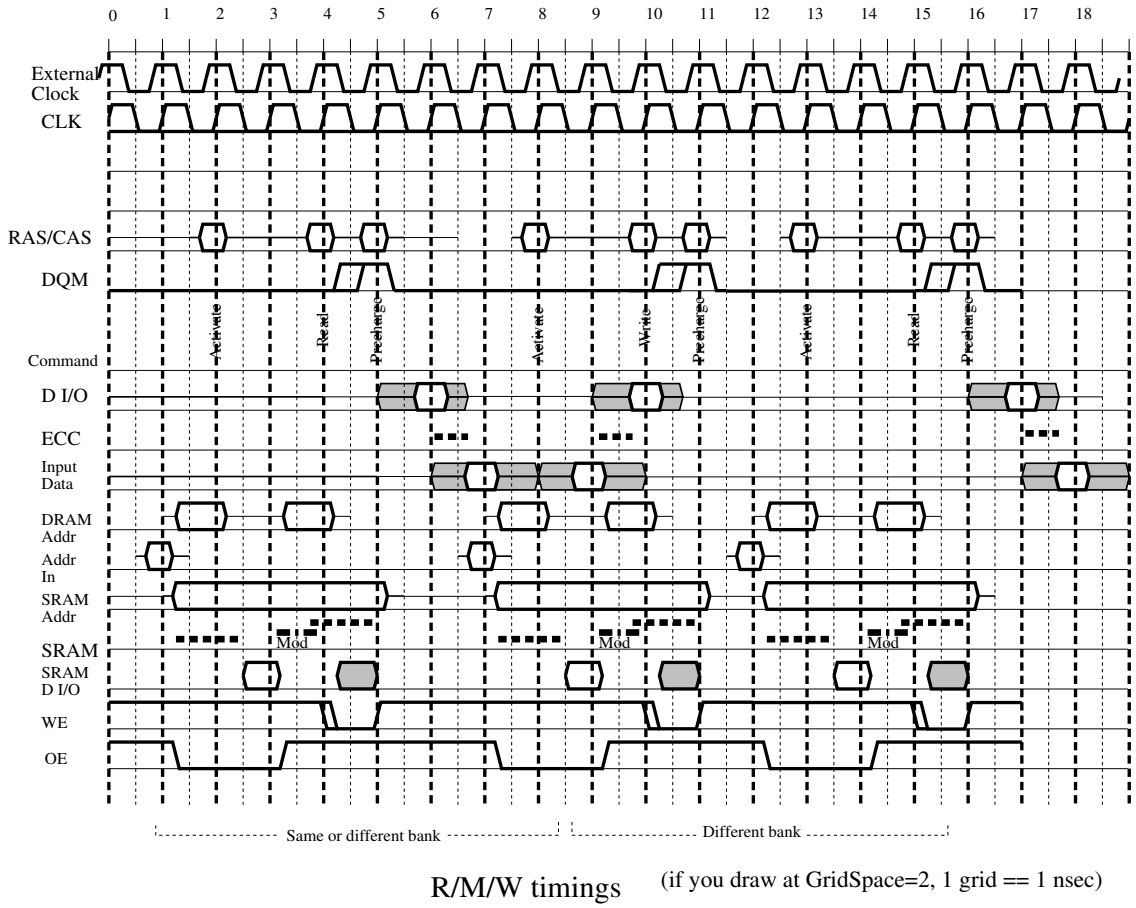

図 11: read / modify / write 操作のタイミング

3.1 バスチップとの接続

MMC チップは、CPU からキャッシュコントローラを介した接続として、バスチップとの接続線を持つ。バスチップとの接続電圧は、バスチップとの間で一貫していればよく、任意性を持つ。

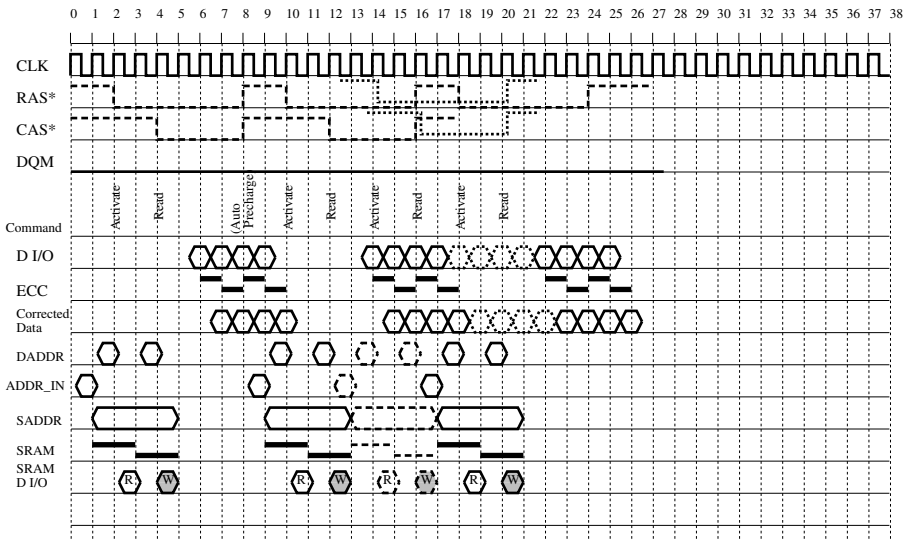

図 12: rdblk 操作の手順

バスチップとの接続信号線(1)

信号名	本数	方向	レベル	記述
DATA[63:0]	64	入出力	正論理	データおよびアドレス転送に用いられる。バス・パケット転送手順の最初では、アドレス／コマンドが送られ、残りの部分ではデータまたは機能を実現するために必要な情報が送られる。DATA の転送方向を切替える場合には、1 クロックの余裕（双方が入力モードとなる）をとることが必要である。
WWE[1:0]	2	入出力	正論理	WWE[1:0]は、パケットのアドレスフリットの場合、request パケットのデータフリットの場合および reply パケットのデータフリットの場合の 3 種類の意味に解釈される。 パケットの第一フリットであるアドレス・フリットにおいては、WWE[1:0]はパケット長を表す。パケット長の解釈は下表の通りである。 WWE[1] / WWE[0] / Packet Length L / L / 1 L / H / 2 H / L / 5 H / H / Reserved データ・フリットにおいては、キャッシュからの request パケットにおいてはバスチップとのデータ（64 ビット）の語（32 ビット）単位の書き込みイネーブル信号。**WWE[0]が DATA[63:32]に対応し、 WWE[1]が DATA[31:0]に対応する。** **キャッシュまたは MBP からの reply パケットでは** WWE[0]**が当該 double word の R-WAIT フラグ（読み出しを待っている書き込まれたデータが存在する）を運び、** WWE[1]**が W-WAIT フラグ（書き込みを待っている読み出しリクエストが存在する）を運ぶ。** なお、 WWE の転送方向は DATA の転送方向と一致する
X_ASTB_O	1	出力	負論理	バスチップへの DATA 信号がアドレスであることを示す。パケット転送の最初に使用される。
X_ASTB_I	1	入力	負論理	バスチップが送り出す DATA 信号がアドレスであることを示す。

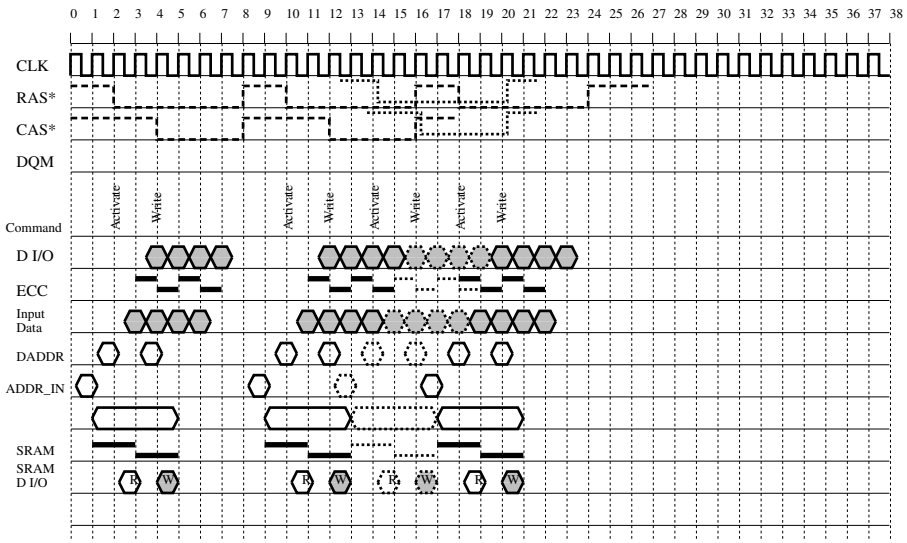

図 13: wrtblk 操作の手順

バスチップとの接続信号線

信号名	本数	方向	レベル	記述
X_BREQ[1:0]	2	出力	負論理	バス要求出力。要求には 2 種の優先度がつけられ、X_BREQ[0]が優先度の低いリクエスト系要求を、X_BREQ[1]が優先度の高いリプライ系要求を表す。 X_BGRT を受けとった 1 クロック後のバスサイクルには、対応する X_BREQ[1:0]の値は解除されている必要がある。さもなければ、新たに要求を行なっているものと解釈される。
X_BBUSY[1:0]	2	出力	負論理	MMC が新たな要求を受けとることが可能か、優先度別に示す。なお、X_BBUSY[1:0]の値は毎クロック独立に与えられるが、アービトレーションを行なった時点と状況がかわり、受けとれなくなった request 系パケットは後述 X_BRETRY_O 信号を出力することにより受けとりを拒否する必要がある。**ただし、** reply **系のパケットは必ず受理する必要がある。** 従って、下記条件に従って X_BBUSY[1:0]を作成する必要がある。 **1. X_BBUSY[1]は assert された時点で arbitration を通過した reply 系パケット全てを取り込めることを前提に assert する必要がある。** **2. X_BBUSY[0]は assert された時点で arbitration を通過したパケットは、リトライする前提で assert しても良い** 3. X_BBUSY[1:0]は、パケットを受け取るデバイスの一時的原因で assert できる。である。ただし， BBUSY の原因は何も共通バスからパケットを受けとらず，また「request 系パケットを送り出さなくても」時間の経過とともに解決する必要がある。 **デッドロックに関する注意：ネットワークを含めた系でのデッドロックを避けるため、リプライキューとリクエストキューは完全に分離している必要がある。**

X_BBUSY[1]	X_BBUSY[0]	Device Status
H	H	全てのパケットを受け取り可
H	L	リクエスト系パケット受け取り不可
L	H	リプライ系パケット受け取り不可
L	L	全てのパケット受け取り不可

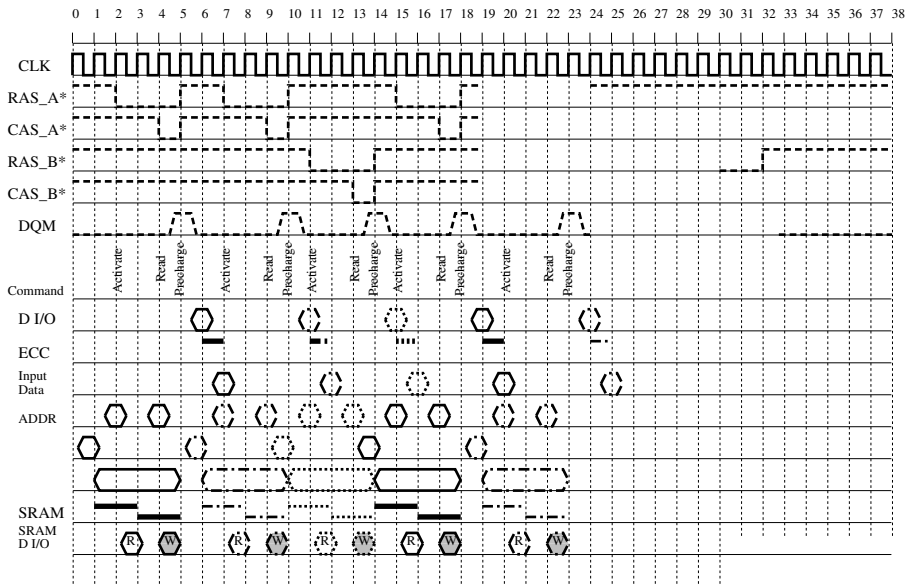

図 14: read 操作の手順

バスチップとの接続信号線 (続き)

信号名	本数	方向	レベル	記述
X_BGRT	1	入力	負論理	X_BGRT が L となると、次のバスサイクルからデータの転送方向が確定し、 MMC がバスを占有する。なお、実際のバス転送は X_ASTB_O 信号が L となるバスサイクルから開始される。 X_BGRT は常に assert された 1 クロック前の要求に対してアービトレーションを行なった結果を示している。（なお、アービタの回路設計上の問題点から、2 クロック前の要求に従う可能性も残っていることに留意されたい） X_BGRT 信号が L となったが、X_ASTB_O 信号が L とならず実際のバス転送が長期間（予め設定されたバスクロック数）開始されない場合には、致命的エラーとなりシステムの動作は保証されない。バスの解放時期は、パケット長で決定される。
X_BRETRY_I	1	入力	負論理	バスチップから来るリトライ要求。パケット送出時では、第 i バスクロックから開始したパケットに関する X_BRETRY_I 信号は第 $i+4$ バスクロックで受理する。 BRETRY **によりバスをリトライする場合はすべてデッドロックを考慮し、絶対に発生しないことを確認する必要がある。**
X_BRETRY_O	1	出力	負論理	MMC が受理したパケットに対するリトライ信号。 request 系パケットにのみ有効である。 MMC 内部で reserve したブロックに対する request 等に対して出力される

バスチップとの接続信号線(3)

信号名	本数	方向	レベル	記述
X_CRPLY_I	1	入力	負論理	現在転送中または直前に転送したパケットに対して、キャッシュメモリがリプライすることを示す信号。

3.2 メモリモジュールとの接続

メモリモジュールは、８０ビット幅のSDRAMと、１６ビット幅のSRAMを搭載している。（実際には、８０ビット中７６ビットが使用される）。メモリモジュールでは、SRAM系は５V、SDRAM系は3.3Vであるため、電圧に関する注意が必要である。

メモリモジュールとの接続信号線(1)

信号名	本数	方向	レベル	記述
SDATA[7:0]	8	入出力	正論理	SRAMで構成される高速タグへの接続。5V信号である
DDATA[63:0]	64	入出力	正論理	SDRAMで構成されるデータ部分への接続。3.3V信号である
BECC[7:0]	8	入出力	正論理	SDRAMのDDATAに関するECCフィールド。3.3V信号である
DTAG[2:0]	3	入出力	正論理	SDRAMのDDATAに付随するタグフィールドとパリティ。3.3V信号である
X_WE	1	出力	負論理	SRAMに対するWE信号。5V信号である
X_OE	1	出力	負論理	SRAMに対するOE信号。5V信号である
X_DCS	1	出力	負論理	SDRAMのCS信号。3.3V信号である
X_RAS	1	出力	負論理	SDRAMのRAS信号。3.3V信号である
X_CAS	1	出力	負論理	SDRAMのCAS信号。3.3V信号である
X_DWE	1	出力	負論理	SDRAMのWE信号。3.3V信号である
DQM	1	出力	正論理	SDRAMのDQM信号。3.3V信号である
SDADR[18:0]	19	出力	正論理	SRAMおよびSDRAMに対するアドレス

3.3 DSPに対する接続信号

MCIFとMMC間は、以下に示す信号線で結合している。なお、MCIFとMMC間の信号線の速度は、主クロックである５０MHｚの１／２である ２５MHｚであることに注意すること。なお、信号線の向きはMMCチップを主体として見ている。すなわち、MMCチップが入力するものを入力と記す。信号は全て ５V TTLレベルである。

MCIF 接続信号線

信号名	本数	方向	レベル	記述
HCLK	1	入力	正論理	MCIF とのインターフェイスに使用するクロック。２５MHｚクロックを入力する。クロックはH→Lのエッジを用い、L→Hのエッジは利用しない。
X_MCREQ	1	入力	負論理	MCIF からの アクセス要求／アクセス制御信号.
X_BLOCK	1	入力	負論理	MCIF からのアクセス要求がブロックアクセス要求であり続けることをしめすレベル信号。ブロック信号が assert されている期間は同一ブロックアクセスであり、アドレス送致が必要ないことを示す
MCRD	1	入力	正論理	MCIF からのアクセス要求が read, block read, であることを示す。 X_MCREQ が assert されている期間に有効である
MCDATA[31:0]	32	入出力	正論理	MCIF とのデータおよびアドレスの転送に用いるデータバス。一般に、 X_MCREQ 信号が assert されている期間はアドレスを、それ以外ではデータを転送する
X_MCRDY	1	出力	負論理	MCIF からのアクセス完了を通知する信号。 read の場合、 X_MCRDY が assert されているクロックでの MCDATA には、読み出し値が入っている。 write の場合には、前のアクセスが完了しているため、新たな write リクエストを出して良いことを示す。リクエストを受理していない場合には、 assert されている場合に新たなアクセスを開始して良いことを示す。 read_block では、最初のデータが ready となったことを通知し、以後転送が終了するまでは ready 状態に保たれる。 write_block では、ブロックモード中は常に assert され続け、ブロックモードから脱して実際の書き込みに入った段階で、 negate される。書き込みが終り、次の要求を受理可能となった時点で再び assert される。
X_MCINT	1	出力	負論理	MMC からの割り込みを通知する信号

3.4 CC に対する直接接続信号

CC と MMC 間は、以下に示す信号線で結合している。信号は全て 5 V TTL レベルである。

MCIF 接続信号線

信号名	本数	方向	レベル	記述
X_PEND[3:0]	4	出力	負論理	各 CC からの要求で MMC で処理中のメモリ処理が存在することを示す。 STBAR 実現で使用される

3.5 MMC 信号ピンのまとめ

以上まとめると、 MMC チップに必要な信号線数は下表の通りである。

バスチップのピン数割り当て

分類	信号名	方向	本数	
共通制御	CLK	I	1	
	X_RESET	I	1	
	FS	I	2	
	SCLK	I	1	スキャンパス用
	SRST	I	1	スキャンパス 用
	SMOD	I	1	スキャンパス用
	SI	I	1	スキャンパス用
	SO	O	1	スキャンパス用
共通バス接続	DATA	IO	64	
	WWE	IO	2	
	X_ASTB_I	I	1	
	X_BGRT	I	1	
	X_BRETRY_I	I	1	
	X_CRPLY_I	I	1	
	X_BREQ	O	2	
	X_BBUSY	O	2	
	X_ASTB_O	O	1	
	X_BRETRY_O	O	1	
メモリ接続	SDATA	IO	8	
	DDATA	IO	64	
	BECC	IO	8	
	DTAG	IO	3	with parity
	X_WE	O	1	
	X_OE	O	1	
	X_DCS	O	1	
	X_RAS	O	1	
	X_CAS	O	1	
	X_DWE	O	1	
	DQM	O	1	
	SDADR	O	19	
DSP との接続	MCDATA	IO	32	
	HCLK	I	1	half clock
	X_MCREQ	I	1	
	X_BLOCK	I	1	
	MCRD	I	1	
	X_MCRDY	O	1	
	X_MCINT	O	1	
CC との接続	X_PEND	O	4	1 for each CC
合計	入力合計	I	16	
	出力合計	O	39	
	入出力合計	IO	181	
	総計		236	

4　共通バスパケットの動作

MMC は共通バスからの情報をパケットおよびそれに付随する信号として授受する。

パケット動作の詳細は、「ＪＵＭＰ－１共通バス仕様書」による指定に準拠するが、ここではMMCチップ固有の問題について述べる。

4.1　パケットのリトライに関する注意

共通バスにおけるリトライは、デッドロックに入る危険性を防止するため下記条件を守る必要がある。

1. **reply 系パケットは、リトライしてはならない**

2. **request 系パケットは、パケットを発信したデバイス自身で常にリトライ可能である。**ただし、MMCがパケットを発信時にMMC自身がリトライを発することはない。すなわち、MMCがパケットを発信する場合は、MMC自身の入力バッファを確保した後に行なう必要がある。

3. request 系パケットは受け取るデバイスの一時的原因でリトライ可能である。ただし，リトライの原因は何も共通バスからパケットを受けとらず，また「request 系パケットを送り出さなくても」時間の経過とともに解決する必要がある。(reply 系パケットを送りだすことが条件でも良い)

4. **request 系パケットを受け取る二次キャッシュおよびMMCは、対象ブロックに関して reply 系パケット出す予定があれば、受け取る request 系パケットをリトライする必要がある。**（つまり、reply しようとしているブロックに対する 他からの request 系パケットはリトライする）

5. **request 系パケットを受け取る二次キャッシュは、対象ブロックに関して request 系パケットを出す予定によって、受け取る request 系パケットをリトライしてはならない。**（つまり、request しようとしているブロックに対する他からの request をリトライすることは駄目。）

6. **request 系パケットを受け取るMMCは、対象ブロックに関して request 系パケットを出す予定がある場合、受け取る request 系パケットをリトライ可能である。**（つまり、request しようとしているブロックに対する他からの request をMMCはリトライして良い。）

7. **二次キャッシュでは request 系パケットの出力（とその結果のリトライ）により reply 系パケットの送出を阻害してはならない。**（つまり、外部からの reply を待っているブロックに対する request を リトライする場合、当該ブロックに対するリプライ出力を阻害してはならない)

8. 上記大原則の実現方法として、MMCによるインプリメントを考慮して下記小原則を設定する：

   (a) **キャッシュからの返答を待つブロックに対する MMCからの request は二次キャッシュがリトライすることが必要である。**（つまり、CRPLY が返るMMCからの read 、 rdinv req の読み出し対象ブロックに対する request のリトライは良い。MMCのインプリメント上二次キャッシュがリトライすることが必要である。）

   (b) **MMCからの返答を待つブロックに対する 二次キャッシュからの request のリトライは二次キャッシュがする必要がある。**（つまり、CRPLY が assert されない request 系パケットで、MMCから返答が返るブロックに対して、二次キャッシュからの新たな request 系パケットはリトライすることが可能である。MMCのインプリメント上このリトライは二次キャッシュで行なうことが必要である。）二次キャッシュにおける本リトライは下記方針で実装される。

   アドレスAに関する MMC からの返答を待っている二次キャッシュは，他の二次キャッシュが送出するパケットの内， ADDR[27:5] が A[27:5] に等しいものを全てリトライさせる。

   なお，返答待ちのトランザクションが特殊なものである場合(e.g. control_read)，リトライされるトランザクションが特殊なものである場合，あるいは返答待ちのトランザクションとリトライされるトランザクションのキャッシュ可否が一致しない場合は「不必要なリトライ」が発生する。

   (規則 3 によって MMC は reply を送出することにより解決する一時的原因によって任意のパケットをリトライさせることができ，これを返答待ち二次キャッシュが代行することも可能である。)

4.2 フリット

フリットは，CBus の基準クロックの 1 サイクルにおける B_DATA[63:0]上およびデータ転送に関連する信号線のデータである。フリットには以下の 2 種のフォーマットがある：

1. アドレス・フリット (Address Flit)
2. データ・フリット (Data Flit)

各パケットの先頭のフリットは必ず アドレス・フリットであり、アドレスおよび引き続くフリットの内容・動作を指定する。引き続く 0 個、1 個または 4 個のフリットは、データ・フリットである。従って、合計 3 種類のパケット形式をなす。

以下では，各フリットに関してMMC固有の点を述べる。

4.2.1 アドレス・フリット

アドレス・フリット は，すべてのパケットの第 1 サイクルに現れる．すなわち、X_BGRT が L となった後 X_ASTB が L であるバスサイクルに出現する。

MMC におけるアドレスフリット解釈では、以下の点が 「ＪＵＭＰ－１共通バス仕様書」に記載されている仕様より制限されている。

のもある。また、パケットの種別と WWE によるパケット長が食い違う指定がされた場合は、パケット長に関して WWE による指定が優先され、パケットの余分な部分は捨てられ、不足の部分は予想できないデータが補充される。図?? に，アドレス・フリット のフォーマットを示す.

アドレス・フリット（１）

フィールド名	幅	記述
ADDR[27:0]	28	パケットの操作対象を示すクラスタ・アドレス。操作対象のサイズが byte, half word, word にかかわらず、各々定められたバウンダリを守る必要がある。すなわち、ブロックデータではブロック境界、2 語データでは 2 語境界、1 語データでは 1 語境界、半語データでは偶数バイト境界のアドレスである必要がある。 ADDR がクラスタメモリの範囲であり（仮ＭＢＰ版の場合、下位１６ＭBytes）、かつ下記 OW_CH ビットが 1 でなく、DUMMY が 1 でない場合、直接クラスタメモリがアクセスされ、それ以外の場合はＭＢＰ動作が起動される
LARGE	1	LARGE が 1 である場合 ADDR フィールドで指定する対象が ラージページ （２５６Ｋバイト単位）にあることを示す。MMCでは使用されない。
SNOOP_CNTL[2:0]	3	SNOOP_CNTL フィールドは、パケットがスヌープ操作を伴う場合に使用される。下位ビットから： **MSH_RPLY** MSH_RPLY ビット。しかしながら、MMCは本ビットを何らセットしないため、各キャッシュコントローラは当該トランザクションの X_MSH 信号を見張り、適切に状態を設定する必要がある。
DID[2:0]	3	パケットの行き先デバイスの指定。仮ＭＢＰ版での実装では、１個しか実装されないが、ＭＢＰ０およびＭＢＰ１の両方の DID を持つパケット、および all が DID で指定されているパケットがMMCで受理される。

4.2.2　データフリット

データフリット の解釈は、「ＪＵＭＰ－１共通バス仕様書」と一致する。

共通バスコマンド一覧表(MMC 用)

PCMD: SCMD	パケット種	対応するリプライ	動作形態	備考
0000-000	read_req	read_reply	ハードウェア動作	コヒーレント読み出し要求
0000-001	n_c_rd_req	n_c_rd_reply	ハードウェア動作	非コヒーレント読み出し要求
0000-100	read_XA_req	read_XA_reply	DSP 呼び出し	BQ 先頭ブロック読み出し要求
0000-101	read_XB_req	read_XB_reply	DSP 呼び出し	BQ 第二先頭ブロック読み出し／デキュー要求
0000-110	read_XC_req	read_XC_reply	DSP 呼び出し	BQ 先頭ブロック読み出し要求
0000-111	read_XD_req	read_XD_reply	DSP 呼び出し	BQ 第二先頭ブロック読み出し／デキュー要求
0010-000	allread_req	allread_reply	ハードウェア動作	コヒーレント allread 要求
0010-010	injection_req	none	動作なし	インジェクション要求
0000-011	swap_req	swap_reply	DSP 呼び出し	スワップ要求
0010-011	swap_M_req	swap_M_reply	DSP 呼び出し	MBP 呼び出し要求
0100-000	rdinv_req	rdinv_reply	ハードウェア動作	read invalidate 要求
0100-100	rdinv_P_req	rdinv_P_reply	DSP 呼び出し	p 同期付き rdinv 要求
0100-110	rdinv_Q_req	rdinv_Q_reply	DSP 呼び出し	q 同期付き rdinv 要求
0100-111	rdinv_F_req	rdinv_F_reply	DSP 呼び出し	q-check 用 要求
1000-000	update_via _home _req	update_reply	ローカルでは動作せず、グローバルは DSP 呼び出し	ホーム経由の update 要求
1000-100	update_via _home _S_req	update_S_reply	DSP 呼び出し	s 同期つき update 要求
1000-110	update_via _home _Q_req	update_Q_reply	DSP 呼び出し	q 同期つき update 要求
1001-000	update_req	none	動作せず	ホーム経由の update 操作
1001-100	update_S_req	none	動作せず	s 同期つき update 操作
1001-110	update_Q_req	none	動作せず	q 同期つき update の操作
1010-000	update_direct_req	none	ローカルは動作せず、グローバルは DSP 呼び出し	直接アップデート要求
1010-100	update_direct _S _req	none	DSP 呼び出し	s 同期つき update 要求
1000-010	write_req	none	ハードウェア動作	ライト要求
1000-011	cflush_req	none	DSP 呼び出し	インバリデートするライトバック

PCMD: SCMD	パケット種	対応するリプライ	動作形態	備考
1010-001	n_c_write_req	none	ハードウェア動作	非コヒーレント書き込み要求
1010-011	n_c_write_M_req	none	DSP 呼び出し	非コヒーレント M 同期書き込み
1100-000	invalidate _via _home _req	invalidate_ req or none	ローカルは動作せず、グローバルは DSP 呼び出し	ホーム経由インバリデート要求
1100-100	write_XA_req	write_XA_reply	DSP 呼び出し	高機能書き込み要求 A
1100-101	write_XB_req	write_XB_reply	DSP 呼び出し	高機能書き込み要求 B
1100-110	write_XC_req	write_XC_reply	DSP 呼び出し	高機能書き込み要求 C
1100-111	write_XD_req	write_XD_reply	DSP 呼び出し	高機能書き込み要求 D
1101-000	invalidate_req	none	動作せず	ホーム経由インバリデート操作
1110-000	control_read_req	control_read_reply	ハードウェア動作	コントロール読み出し要求
1110-010	control_write_req	none	ハードウェア動作	コントロール書き込み要求
1110-001	demap_req	demap_ack	動作せず	デマップ要求
1110-011	interrupt_req	none	動作せず	割り込み要求

リプライパケット一覧表

PCMD: SCMD	パケット種	等価なリプライ	動作形態	備考
0001-000	read_reply	read_reply	DSP へ渡す	読み出しリプライ
0001-001	n_c_rd_reply	read_reply	(DSP へ渡す)	非コヒーレント読み出しリプライ
0001-100	read_XA_reply		(DSP へ渡す)	read_XA_req へのリプライ
0001-101	read_XB_reply		(DSP へ渡す)	read_XB_req へのリプライ
0001-110	read_XC_reply		(DSP へ渡す)	read_XC_req へのリプライ
0001-111	read_XD_reply		(DSP へ渡す)	read_XD_req へのリプライ
0011-000	allread_reply	allread_reply	(DSP へ渡す)	allread リプライ
0001-011	swap_reply	read_reply	(DSP へ渡す)	スワップのリプライ
0011-011	swap_M_reply	read_reply	(DSP へ渡す)	M 同期つきスワップのリプライ
0111-000	rdinv_reply	rdinv_reply	DSP へ渡す	read invalidate リプライ
0101-000	rdinv_reply	rdinv_reply	DSP へ渡す	上記の alias
0101-100	rdinv_P_reply	rdinv_reply	(DSP へ渡す)	p 同期付き rdinv リプライ
0101-110	rdinv_Q_reply	rdinv_reply	(DSP へ渡す)	q 同期付き rdinv リプライ
0101-111	rdinv_F_reply	rdinv_reply	(DSP へ渡す)	q-check 実現用 リプライ
1001-010	writeback_reply	none	ハードウェア動作	ライトバック要求 (リプライレベル)
1101-100	write_XA_reply		(DSP へ渡す)	高機能書き込み要求 A のリプライ
1101-101	write_XB_reply		(DSP へ渡す)	高機能書き込み要求 B のリプライ
1101-110	write_XC_reply		(DSP へ渡す)	高機能書き込み要求 C のリプライ
1101-111	write_XD_reply		(DSP へ渡す)	高機能書き込み要求 D のリプライ
1111-000	control_read_reply	read_reply	DSP へ渡す	コントロール読み出しリプライ
1111-001	demap_ack	read_reply	DSP へ渡す	デマップリプライ
1111-111	nop		無視する	no operation

コード表

PCMD: SCMD	パケット種	備考
0000-000	**read_req**	コヒーレント読み出し要求
0000-001	**n_c_rd_req**	非コヒーレント読み出し要求
0000-010		Reserved
0000-011	swap_req	スワップ要求
0000-100	read_XA_req	BQ 先頭ブロック読み出し
0000-101	read_XB_req	BQ 第二先頭ブロック読み出し
0000-110	read_XC_req	BQ の先頭ブロック読み出し／デキュー
0000-111	read_XD_req	BQ の第二先頭ブロック読み出し／デキュー
0001-000	read reply	読み出しリプライ
0001-001	n_c_rd_reply	非コヒーレント読み出しリプライ
0001-010		Reserved
0001-011	swap_reply	スワップのリプライ
0001-100	read_XA_reply	read_XA_req へのリプライ
0001-101	read_XB_reply	read_XB_req へのリプライ
0001-110	read_XC_reply	read_XC_req へのリプライ
0001-111	read_XD_reply	read_XD_req へのリプライ
0010-000	**allread_req**	コヒーレント allread 要求
0010-001		Reserved
0010-010	injection_req	インジェクション要求
0010-011	swap_M_req	MBP 呼び出し要求
0010-100		Reserved
0010-101		Reserved
0010-110		Reserved
0010-111		Reserved
0011-000	allread_reply	allread リプライ
0011-001		Reserved
0011-010		Reserved
0011-011	swap_M_reply	M 同期つきスワップのリプライ
0011-100		Reserved
0011-101		Reserved
0011-110		Reserved
0011-111		Reserved

PCMD: SCMD	パケット種	備考
0100-000	**rdinv_req**	rdinv 要求
0100-001		Reserved
0100-010		Reserved
0100-011		Reserved
0100-100	rdinv_P_req	p 同期付き rdinv 要求
0100-101		Reserved
0100-110	rdinv_Q_req	q 同期付き rdinv 要求
0100-111	rdinv_F_req	q-check 用要求
0101-000	rdinv_reply	rdinv への reply
0101-001		Reserved
0101-010		Reserved
0101-011		Reserved
0101-100	rdinv_P_reply	p 同期付き rdinv リプライ
0101-101		Reserved
0101-110	rdinv_Q_reply	q 同期付き rdinv リプライ
0101-111	rdinv_F_reply	q-check 用リプライ
0110-000		Reserved
0110-001		Reserved
0110-010		Reserved
0110-011		Reserved
0110-100		Reserved
0110-101		Reserved
0110-110		Reserved
0110-111		Reserved
0111-000	rdinv_reply	read invalidate リプライ
0111-001		Reserved
0111-010		Reserved
0111-011		Reserved
0111-100		Reserved
0111-101		Reserved
0111-110		Reserved
0111-111		Reserved

PCMD: SCMD	パケット種	備考
1000-000	**update_via _home _req**	ホーム経由の update 要求
1000-001		Reserved
1000-010	**write_req**	ライト要求
1000-011	cflush_req	インバリデートするライトバック
1000-100	update_via _home _S_req	s 同期つき update 要求
1000-101		Reserved
1000-110	update_via _home _Q_req	q 同期つき update 要求
1000-111		Reserved
1001-000	update_req	ホーム経由の update のリプライ
1001-001		Reserved
1001-010	**writeback_reply**	ライトバック要求（リプライレベル）
1001-011		Reserved
1001-100	update_S_req	s 同期つき update のリプライ
1001-101		Reserved
1001-110	update_Q_req	q 同期つき update のリプライ
1001-111		Reserved
1010-000	**update_direct_req**	直接アップデート要求
1010-001	**n_c_write_req**	非コヒーレント書き込み要求
1010-010		Reserved
1010-011	n_c_write_M_req	非コヒーレント M 同期書き込み
1010-100	update_direct _S _req	s 同期つき update 要求
1010-101		Reserved
1010-110		Reserved
1010-111		Reserved
1011-000		Reserved
1011-001		Reserved
1011-010		Reserved
1011-011		Reserved
1011-100		Reserved
1011-101		Reserved
1011-110		Reserved
1011-111		Reserved

PCMD: SCMD	パケット種	備考
1100-000	invalidate _via _home _req	ホーム経由インバリデート要求
1100-001		Reserved
1100-010		Reserved
1100-011		Reserved
1100-100	write_XA_req	高機能書き込み命令 A
1100-101	write_XB_req	高機能書き込み命令 B
1100-110	write_XC_req	高機能書き込み命令 C
1100-111	write_XD_req	高機能書き込み命令 D
1101-000	invalidate_req	ホーム経由インバリデートのリプライ
1101-001		Reserved
1101-010		Reserved
1101-011		Reserved
1101-100	write_XA_reply	高機能書き込み命令 A のリプライ
1101-101	write_XB_reply	高機能書き込み命令 B のリプライ
1101-110	write_XC_reply	高機能書き込み命令 C のリプライ
1101-111	write_XD_reply	高機能書き込み命令 D のリプライ
1110-000	**control_read_req**	コントロール読み出し要求
1110-001	demap_req	デマップ要求
1110-010	**control_write_req**	コントロール書き込み要求
1110-011	interrupt_req	割り込み要求
1110-100		Reserved
1110-101		Reserved
1110-110		Reserved
1110-111		Reserved
1111-000	control_read_reply	コントロール読み出しリプライ
1111-001	demap_ack	デマップリプライ
1111-010		Reserved
1111-011		Reserved
1111-100		Reserved
1111-101		Reserved
1111-110		Reserved
1111-111	**nop**	No operation

4.3 パケットの詳細

パケットの種類は、アドレス・フリットの PCMD および SCMD フィールドの値により定まる。種類は大別してリクエスト系とリプライ系にわけられ、各々異なった要求優先順位が与えられる。なお、リクエスト系かリプライ系かを、PCMD または SCMD の特定ビットで判断することは不可能であり、full bit のデコード結果を用いる。

コマンド表中、reserved となっているものは、全て request 系パケットとして解釈される。

各パケットの説明中、パケット種別は PCMD および SCMD であたえられるものとし、説明から省略されている。

4.3.1 read_req

read_req は、バスに接続されたメモリからのコヒーレント読み出し要求であり、パケット長は 1 である。MMC は、パケットの DID が all または MMC である場合にパケットを受理し、以下の動作を開始する。

MMC はまず、ＤＳＰからセットされる BLK_RSV レジスタの内容とアドレスを比較し、ＤＳＰにより予約されているブロックが対象である場合、RETRY 信号を assert する。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。

次に、キャッシュからの RETRY 信号、 CRPLY 信号を受理する。

受理した RETRY 信号または CRPLY 信号が 真 である場合は、全ての処理を放棄し、初期状態に戻る。

受理したアドレスが実装されているクラスタアドレスの範囲外または、 DUMMY フィールドが 真 である場合 または OW_CH フィールドが真である場合はＤＳＰに割り込み、プログラム処理に移行する。

入力したアドレスに従い、DATA、DTAG および STAG をアクセスする。

STAG 中、 RD_TRAP または VALID が 偽である場合、処理を中断し、ＤＳＰプログラム処理に移行する。

さもなくば、 DATA,DTAG,STAG に従い read_reply パケットを返す。 DATA,DTAG,STAG とも、状態を変更しない。

注 1 ： MMC が発する read_req に対し、キャッシュがリプライしない場合は、 MMC 自身でリプライする必要がある。ただし、 MMC から MMC へのパケットの受渡しにバスを使えないため、 1 クロックの遅延後、直接転送する必要がある。 MMC ではキャッシュがリプライすることが確定した時点、すなわち X_CRPLY 信号が assert された場合に MMC 内バッファに取り込んだパケットを取り消す

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.addr > MAX_ADDR)||(in.dummy == 1)||(in.ow_ch == 1)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.valid == 0)||(stag.rd_trap == 1)) -> TRAP(in);

out.cmd <- READ_REPLY;

out.addr <-   in.addr;
out.home <-   in.home;
out.dummy <-  in.dummy;
out.ow_ch <-  in.ow_ch;
out.large <-  in.large;
out.dirty <-  0;
out.msh <-    0; /* V2.0 */
out.su <-     in.su;
out.fe <-     stag.fe;
out.clsh <-   stag.global;
out.size <-   in.size;
out.sid <-    MMC;
out.did <-    in.sid;
out.ccbl <-   in.ccbl;
out.ack <-    0;
out.inj_mask <- 0;

(in.size == BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
 dtag1 = DTRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 dtag2 = DTRAM[in.addr[5:24] | (in.addr[3:4]+2)mod4];
 dtag3 = DTRAM[in.addr[5:24] | (in.addr[3:4]+3)mod4];
 data0 = DRAM[in.addr[3:24]];
 data1 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data2 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data3 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];

 out.len = 2;
```

```
 out.wwe0 = dtag0[0:1];  /* W-WAIT : R-WAIT */
 out.data0 = data0;
 out.wwe1 = dtag1[0:1];  /* W-WAIT : R-WAIT */
 out.data1 = data1;
 out.wwe2 = dtag2[0:1];  /* W-WAIT : R-WAIT */
 out.data2 = data2;
 out.wwe3 = dtag3[0:1];  /* W-WAIT : R-WAIT */
 out.data3 = data3;
}

(in.size != BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
 data0 = DRAM[in.addr[3:24]];

 out.len = 1;
 out.wwe0 = dtag0[0:1];  /* W-WAIT : R-WAIT */
 out.data0 = data0;
}

EXIT;
```

4.3.2 non_coherent_read_req

non_coherent_read_req は、バスに接続されたメモリからの非コヒーレント読み出し要求であり、パケット長は１である。MMCは、パケットの DID が all または MMC である場合にパケットを受理し、read_req と以下の点を除き同一の動作を行なう：

1. read_reply の変わりに、non_coherent_read_reply を返す。

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.addr > MAX_ADDR)||(in.dummy == 1)||(in.ow_ch == 1)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.valid == 0)||(stag.rd_trap == 1)) -> TRAP(in);

out.cmd <- NON_COHERENT_READ_REPLY;

out.addr <-   in.addr;
out.home <-   in.home;
out.dummy <-  in.dummy;
out.ow_ch <-  in.ow_ch;
out.large <-  in.large;
out.dirty <-  0;
out.msh <-    0;   /* V2.0 */
out.su <-     in.su;
out.fe <-     stag.fe;
out.clsh <-   stag.global;
out.size <-   in.size;
out.sid <-    MMC;
out.did <-    in.sid;
out.ccbl <-   in.ccbl;
out.ack <-    0;
out.inj_mask <- 0;

(in.size == BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
 dtag1 = DTRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 dtag2 = DTRAM[in.addr[5:24] | (in.addr[3:4]+2)mod4];
 dtag3 = DTRAM[in.addr[5:24] | (in.addr[3:4]+3)mod4];
 data0 = DRAM[in.addr[3:24]];
 data1 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data2 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data3 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];

 out.len = 2;
 out.wwe0 = dtag0[0:1];
 out.data0 = data0;
 out.wwe1 = dtag1[0:1];
 out.data1 = data1;
 out.wwe2 = dtag2[0:1];
 out.data2 = data2;
 out.wwe3 = dtag3[0:1];
 out.data3 = data3;
}

(in.size != BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
 data0 = DRAM[in.addr[3:24]];
```

```
 out.len = 1;
 out.wwe0 = dtag0[0:1];
 out.data0 = data0;
}

EXIT;
```

4.3.3 allread_req

allread_req は、バスに接続されたメモリからのキャッシュインジェクション付きコヒーレント読み出し要求であり、パケット長は 1 である。MMC は、パケットの DID が all または MMC である場合にパケットを受理し、以下に示す点を除き read_req と同一の動作を行なう：

1. read_reply の代わりに allread_reply を返す
2. INJ_MASK フィールドに all 1 を返す

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.addr > MAX_ADDR)||(in.dummy == 1)||(in.ow_ch == 1)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.valid == 0)||(stag.rd_trap == 1)) -> TRAP(in);

out.cmd <- ALLREAD_REPLY;

out.addr <-   in.addr;
out.home <-   in.home;
out.dummy <-  in.dummy;
out.ow_ch <-  in.ow_ch;
out.large <-  in.large;
out.dirty <-  0;
out.msh <-    0;     /*  V2.0  */
out.su <-     in.su;
out.fe <-     stag.fe;
out.clsh <-   stag.global;
out.size <-   in.size;
out.sid <-    MMC;
out.did <-    in.sid;
out.ccbl <-   in.ccbl;
out.ack <-    0;
out.inj_mask <- 0xf;

(in.size == BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
 dtag1 = DTRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 dtag2 = DTRAM[in.addr[5:24] | (in.addr[3:4]+2)mod4];
 dtag3 = DTRAM[in.addr[5:24] | (in.addr[3:4]+3)mod4];
 data0 = DRAM[in.addr[3:24]];
 data1 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data2 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data3 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];

 out.len = 2;
 out.wwe0 = dtag0[0:1];
 out.data0 = data0;
 out.wwe1 = dtag1[0:1];
 out.data1 = data1;
 out.wwe2 = dtag2[0:1];
 out.data2 = data2;
 out.wwe3 = dtag3[0:1];
 out.data3 = data3;
}

(in.size != BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
```

```
 data0 = DRAM[in.addr[3:24]];

 out.len = 1;
 out.wwe0 = dtag0[0:1];
 out.data0 = data0;
}

EXIT;
```

4.3.4 rdinv_req

rdinv_req は、バスに接続されたメモリからのコヒーレント読み出し兼インバリデート要求であり、パケット長は 1 である。MMC は、パケットの DID が all または MMC である場合にパケットを受理し、以下の動作を開始する。

MMC はまず、ＤＳＰからセットされる BLK_RSV レジスタの内容とアドレスを比較し、ＤＳＰにより予約されているブロックが対象である場合、RETRY 信号を assert する。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。

次に、キャッシュからの RETRY 信号、 CRPLY 信号を受理する。

受理した RETRY 信号または CRPLY 信号が 真 である場合は、全ての処理を放棄し、初期状態に戻る。

受理したアドレスが実装されているクラスタアドレスの範囲外または、 DUMMY フィールドが 真 である場合 または OW_CH フィールドが真である場合はＤＳＰに割り込み、プログラム処理に移行する。

入力したアドレスに従い、DATA、DTAG および STAG をアクセスする。

STAG 中、GLOBAL が真である場合、RD_TRAP または VALID が 偽である場合、処理を中断し、ＤＳＰプログラム処理に移行する。

さもなくば、DATA,DTAG,STAG に従い read_reply パケットを返す。 DATA,DTAG,STAG とも、状態を変更しない。

注 1 ： DID フィールドの値が all または MMC の場合、MMC は rdinv_req パケットを受理する。
注 2 ： MMC が発する read_req に対し、キャッシュがリプライしない場合は、 MMC 自身でリプライする必要がある。ただし、MMC から MMC へのパケットの受渡しにバスを使えないため、 1 クロックの遅延後、直接転送する必要がある。
注 3 ： CCBL = 0 の場合、 non cacheable または 3 次 cacheable として扱うが、global 以外ではそのまま読み出す

注 4 ： PEND 信号を発生するための PEND_CNTR の増減と、それに伴う処理は、全ての reqest 系パケットに共通の処理であるが、 rdinv_req では重要であるため、特に示した。なお、 PEND 信号は、 PEND_CNTR が 0 でない場合に送出される。

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

((in.addr > MAX_ADDR)||(in.dummy == 1)||(in.ow_ch == 1)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.global == 1)|(stag.valid == 0)||
 (stag.rd_trap == 1)) -> TRAP(in);

out.cmd <- RDINV_REPLY;

out.addr <-   in.addr;
out.home <-   in.home;
out.dummy <-  in.dummy;
out.ow_ch <-  in.ow_ch;
out.large <-  in.large;
out.dirty <-  0;
out.msh <-    0;
out.su <-     in.su;
out.fe <-     stag.fe;
out.clsh <-   stag.global; /* <- 0 でも良い*/
out.size <-   in.size;
out.sid <-    MMC;
out.did <-    in.sid;
out.ccbl <-   in.ccbl;
out.ack <-    0;
out.inj_mask <- 0;

(in.size == BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
```

```
 dtag1 = DTRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 dtag2 = DTRAM[in.addr[5:24] | (in.addr[3:4]+2)mod4];
 dtag3 = DTRAM[in.addr[5:24] | (in.addr[3:4]+3)mod4];
 data0 = DRAM[in.addr[3:24]];
 data1 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data2 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];
 data3 = DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4];

 out.len = 2;
 out.wwe0 = dtag0[0:1];
 out.data0 = data0;
 out.wwe1 = dtag1[0:1];
 out.data1 = data1;
 out.wwe2 = dtag2[0:1];
 out.data2 = data2;
 out.wwe3 = dtag3[0:1];
 out.data3 = data3;
}

(in.size != BLOCK) ->{
 dtag0 = DTRAM[in.addr[3:24]];
 data0 = DRAM[in.addr[3:24]];

 out.len = 1;
 out.wwe0 = dtag0[0:1];
 out.data0 = data0;
}

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.5 writeback_reply

writeback_reply は、キャッシュメモリからの ライトバック （CCBL=1 の場合）に用いられる。

writeback_reply のパケット長は、書き込むデータがブロックなので 5 である。データサイズに masked_block が指定される場合は、データフリットに付属する WWE フィールドにより、書き込みを行なう word を指定する。実際にはこの動作は DSP プログラムにより実現される。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。

受理したアドレスが実装されているクラスタアドレスの範囲外または、 DUMMY フィールドが 真 である場合 または OW_CH フィールドが真である場合、または **サイズが masked_block である場合** はＤＳＰに割り込み、プログラム処理に移行する。

入力したアドレスに従い、DATA、DTAG および STAG をアクセスする。

STAG 中、GLOBAL **が真である場合**、 WRT_TRAP または VALID が 偽である場合、処理を中断し、ＤＳＰプログラム処理に移行する。

さもなくば、DATA,STAG の FE フィールドに入力データを書き込み、DTAG は状態を変更しない。

注１ ： DID フィールドの値が all または MMC の場合、MMC は パケットを受理する。

注２ ： PEND 信号を発生するための PEND_CNTR の増減と、それに伴う処理が、reply 系パケットとして例外的に必要である。なお、 PEND 信号は、 PEND_CNTR が 0 でない場合に送出される。

```
((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

((in.addr > MAX_ADDR)||(in.dummy == 1)||
 (in.ow_ch == 1)||
 (in.size != BLOCK)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.global == 1)|(stag.valid == 0)||
 (stag.wrt_trap == 1)) -> TRAP(in);

 DRAM[in.addr[3:24]] = in.data0;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data1;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data2;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data3;

 stagw.fe = in.fe;
 stagw.all_trap = stag.all_trap;
 stagw.rd_trap =  stag.rd_trap;
 stagw.wrt_trap = stag.wrt_trap;
 stagw.up_trap =  stag.up_trap;
 stagw.inv_trap = stag.inv_trap;
 stagw.valid = stag.valid;
 stagw.global = stag.global;

 SRAM[in.addr[5:24]] = stagw;


((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.6 write_req

write_req は、バスに接続されたメモリへのコヒーレント書き込み要求である。write_req は通常キャッシュメモリからの non cacheable write （CCBL=0 の場合）に用いられる。

write_req の動作は、アドレスフリット中、スヌープコントロールフィールドの CLSH ビットの値により大きくかわる。CLSH==1 の場合、ホームまで書き戻す writeback_to_home の意であり、CLSH == 0 の場合、キャッシュが属するクラスタ内のクラスタメモリまで書き戻す。

write_req のパケット長は、書き込むデータがブロックの場合 5 であり、その他の場合 2 である。データサイズに masked_block が指定される場合は、データフリットに付属する WWE フィールドにより、書き込みを行なう word を指定する。実際にはこの動作は DSP プログラムにより実現される。

MMC はまず、ＤＳＰからセットされる BLK_RSV レジスタの内容とアドレスを比較し、ＤＳＰにより予約されているブロックが対象である場合、RETRY 信号を assert する。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。

次に、キャッシュからの RETRY 信号、 CRPLY 信号を受理する。

受理した RETRY 信号または CRPLY 信号が 真 である場合は、全ての処理を放棄し、初期状態に戻る。ただし、この状態は本リクエストでは発生しないことが予定され、発生する場合は何らかの不都合が発生している。

受理したアドレスが実装されているクラスタアドレスの範囲外または、 DUMMY フィールドが 真 である場合 または OW_CH フィールドが真である場合、**または CLSH ビットが 1 である場合**、または **サイズが masked_block である場合** はＤＳＰに割り込み、プログラム処理に移行する。

入力したアドレスに従い、DATA、DTAG および STAG をアクセスする。

STAG 中、GLOBAL **が真である場合**、 WRT_TRAP または VALID が 偽である場合、処理を中断し、ＤＳＰプログラム処理に移行する。

さもなくば、DATA, に入力データを書き込み、 DTAG,STAG は状態を変更しない。

注 1 ： DID フィールドの値が all または MMC の場合、MMC は パケットを受理する。

注 2 ： CCBL = 0 の場合、 non cacheable または 3 次 cacheable として扱うが、global 以外ではそのまま書き込む

注 3 ： PEND 信号を発生するための PEND_CNTR の増減と、それに伴う処理は、全ての reqest 系パケットに共通の処理であるが、 rdinv_req では重要であるため、特に示した。なお、 PEND 信号は、 PEND_CNTR が 0 でない場合に送出される。

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

((in.addr > MAX_ADDR)||(in.dummy == 1)||
 (in.ow_ch == 1)||(in.clsh == 1)||
 (in.size == MASKED_BLOCK)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.global == 1)|(stag.valid == 0)||
 (stag.wrt_trap == 1)) -> TRAP(in);

(in.size == BLOCK) ->{

 DRAM[in.addr[3:24]] = in.data0;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data1;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data2;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data3;

}

(in.size != BLOCK) ->{
 DRAM[in.addr[3:24]] = in.data0;
}
```

```
((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.7 non_coherent_write_req

non_coherent_write_req は、二次キャッシュメモリから、バスに接続されたメモリへの非コヒーレント書き込み要求である。

non_coherent_write_req の動作は 以下の点を含め write_req と全く同一である。

- non_coherent_write_req はスヌープされない。（キャッシュの動作が異なる）
- non_coherent_write_req は MMC におけるコヒーレンシ維持操作を起動しない（実際には、global なブロックに対しての non_coherent_write_req は DSP プログラムを起動するため、DSP プログラムが global な場合の動作を定める）

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

((in.addr > MAX_ADDR)||(in.dummy == 1)||
 (in.ow_ch == 1)||(in.clsh == 1)||
 (in.size == MASKED_BLOCK)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.global == 1)|(stag.valid == 0)||
 (stag.wrt_trap == 1)) -> TRAP(in);

(in.size == BLOCK) ->{

 DRAM[in.addr[3:24]] = in.data0;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data1;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data2;
 DRAM[in.addr[5:24] | (in.addr[3:4]+1)mod4] = in.data3;

}

(in.size != BLOCK) ->{
 DRAM[in.addr[3:24]] = in.data0;
}

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.8 update_direct_req

update_direct_req は、バスに接続されたキャッシュおよびメモリへの直接アップデート要求である。update_direct_req のパケット長は、ブロックを書き込む場合には 5 であり、それ以外の場合は 2 である。

リクエストがキャッシャブルの場合には各キャッシュは、update プロトコルに対応する動作を行なう。リクエストが non cacheable の場合、DSP プログラム呼び出しとなる。

データサイズに masked_block が指定される場合では、データフリットに付属する WWE フィールドにより、書き込みを行なう word を指定する。実際にはこの動作が MMC で必要になる場合は DSP プログラムにより実現される。

MMC はまず、ＤＳＰからセットされる BLK_RSV レジスタの内容とアドレスを比較し、ＤＳＰにより予約されているブロックが対象である場合、RETRY 信号を assert する。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。

次に、キャッシュからの RETRY 信号、CRPLY 信号を受理する。

受理した RETRY 信号または CRPLY 信号が 真 である場合は、全ての処理を放棄し、初期状態に戻る。ただし、CRPLY 信号は本リクエストでは発生しないことが予定され、発生する場合は何らかの不都合が発生している。（なお、update_direct_req では CRPLY 信号が返ることは仕様上あり得ないが、他のパケットと統一性を持たせるため、本仕様となる）

受理したアドレスが実装されているクラスタアドレスの範囲外または、DUMMY フィールドが 真 である場合 または OW_CH フィールドが真である場合、**または CLSH ビットが 1 である場合**、または **サイズが masked_block である場合** はＤＳＰに割り込み、プログラム処理に移行する。

入力したアドレスに従い、STAG をアクセスする。

STAG 中、GLOBAL **が真である場合、** または VALID が 偽である場合、処理を中断し、ＤＳＰプログラム処理に移行する。

さもなくば、何も状態を変化せず、出力パケットも出さずに動作を終了する。

注 1 ： DID フィールドの値が all または MMC の場合、MMC は パケットを受理する。

注 2 ： CCBL = 0 の場合、non cacheable または 3 次 cacheable として扱うが、global 以外ではそのまま書き込む

注 3 ： PEND 信号を発生するための PEND_CNTR の増減と、それに伴う処理は、全ての reqest 系パケットに共通の処理であるが、rdinv_req では重要であるため、特に示した。なお、PEND 信号は、PEND_CNTR が 0 でない場合に送出される。

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

((in.addr > MAX_ADDR)||(in.dummy == 1)||
 (in.ow_ch == 1)||(in.clsh == 1)||
 (in.ccbl == 0)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.global == 1)|(stag.valid == 0)) -> TRAP(in);  /* V2.0 */


((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.9 update_via_home_req

update_via_home_req は、キャッシュに対するホームページを経由したアップデート要求である。update_via_home_req のパケット長は、ブロックが対象の場合は 5 である、その他の場合には 2 である。リクエストがキャッシャブルの場合、ブロックが global でなければ直ちに update がキャッシュにおいて行なわれる。ブロックが global な場合にはホームによりシリアライズを経たのち、MMC から update_req パケットにより各キャッシュが update される。

データサイズに masked_block が指定される場合は、データフリットに付属する WWE フィールドにより、書き込みを行なう word を指定する。実際にはこの動作は DSP プログラムにより実現される。

MMC はまず、ＤＳＰからセットされる BLK_RSV レジスタの内容とアドレスを比較し、ＤＳＰにより予約されているブロックが対象である場合、RETRY 信号を assert する。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。

次に、キャッシュからの RETRY 信号、CRPLY 信号を受理する。

受理した RETRY 信号または CRPLY 信号が 真 である場合は、全ての処理を放棄し、初期状態に戻る。ただし、この状態は本リクエストでは発生しないことが予定され、発生する場合は何らかの不都合が発生している。

受理したアドレスが実装されているクラスタアドレスの範囲外または、DUMMY フィールドが 真 である場合 または OW_CH フィールドが真である場合、**または CLSH ビットが 1 である場合**、または **CCBL==0** である場合 はＤＳＰに割り込み、プログラム処理に移行する。

入力したアドレスに従い、TAG をアクセスする。

STAG 中、GLOBAL **が真である場合**、 または VALID が 偽である場合、処理を中断し、ＤＳＰプログラム処理に移行する。

さもなくば、何もせず動作を終了する。

注 1 ： DID フィールドの値が all または MMC の場合、MMC は パケットを受理する。

注 2 ： CCBL = 0 の場合、 non cacheable または 3 次 cacheable として扱う

注 3 ： PEND 信号を発生するための PEND_CNTR の増減と、それに伴う処理は、全ての reqest 系パケットに共通の処理であるが、 rdinv_req では重要であるため、特に示した。なお、 PEND 信号は、 PEND_CNTR が 0 でない場合に送出される。

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

((in.addr > MAX_ADDR)||(in.dummy == 1)||
 (in.ow_ch == 1)||(in.clsh == 1)||
 (CCBL == 0)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.global == 1)|(stag.valid == 0)) -> TRAP(in);

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.10 invalidate_via_home_req

invalidate_via_home_req は、キャッシュおよびコピーページに対するホームページを経由したインバリデート要求であり、パケット長は 1 である。リクエストが Non-cacheable の場合の動作は未定義であり、リクエストがキャッシャブルの場合にはホームによりシリアライズを経たのち、MBP から invalidate_reply パケットにより各キャッシュが invalidate される。

MMC はまず、ＤＳＰからセットされる BLK_RSV レジスタの内容とアドレスを比較し、ＤＳＰにより予約されているブロックが対象である場合、RETRY 信号を assert する。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。

次に、キャッシュからの RETRY 信号、 CRPLY 信号を受理する。

受理した RETRY 信号または CRPLY 信号が 真 である場合は、全ての処理を放棄し、初期状態に戻る。ただし、この状態は本リクエストでは発生しないことが予定され、発生する場合は何らかの不都合が発生している。

受理したアドレスが実装されているクラスタアドレスの範囲外または、 DUMMY フィールドが 真 である場合 または OW_CH フィールドが真である場合、**または CLSH ビットが 1 である場合** はＤＳＰに割り込み、プログラム処理に移行する。

入力したアドレスに従い、STAG をアクセスする。

STAG 中、GLOBAL **が真である場合、** または VALID が 偽である場合、処理を中断し、ＤＳＰプログラム処理に移行する。

さもなくば、何もせず、動作を終了する。

注 1 ： DID フィールドの値が all または MMC の場合、MMC は パケットを受理する。

注 2 ： CCBL = 0 の場合、 non cacheable または 3 次 cacheable として扱う

注 3 ： PEND 信号を発生するための PEND_CNTR の増減と、それに伴う処理は、全ての reqest 系パケットに共通の処理であるが、 rdinv_req では重要であるため、特に示した。なお、 PEND 信号は、 PEND_CNTR が 0 でない場合に送出される。

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

((in.addr > MAX_ADDR)||(in.dummy == 1)||
 (in.ow_ch == 1)||(in.clsh == 1)) -> TRAP(in);

stag = SRAM[in.addr[5:24]];

((stag.global == 1)|(stag.valid == 0)) -> TRAP(in);

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.11 rdinv_P_req

rdinv_P_req は、バスに接続されたメモリからの読みだし要求兼インバリデート要求であり、 Memory-Based FIFO を構成するアクセス法のための nonblocking 読みだしである。 rdinv_P_req では当該ロケーションの同期ビットの full/empty 情報と、 full である場合はデータを要求元キャッシュへ exclusive な状態として読み出し、 FE ビットをクリアする。 empty であることは rdinv_P_reply パケットのアドレス・フリットに含まれる FEBIT フィールドで示され、実際に CPU 返り値としての 0 (null) 作成は二次キャッシュにおいて処理される。

rdinv_P_req パケットは、後述 rdinv_Q_req とほぼ同じ動作をするが、 CPU から見た動作、ＭＢＰにおける待ち合わせ動作が異なるため、異なるコマンドとする必要がある。

rdinv_P_req のパケット長は 1 である。MMC は、パケットの DID が all または MMC である場合にパケットを受理し、 retry または CRPLY の場合を除き、必ず DSP プログラムを呼び出す。

DSP プログラム呼び出しにあたって、 ACK フィールドが 1 の場合 MCC 内部の ACK カウンタをカウントアップし、 PEND 信号を assert する。 ACK カウンタのカウントダウンと、それに伴う（重複していない場合の） PEND の deassert は DSP プログラムにより実現される。

なお、rdinv_Q_req、 は全ての処理がDSP プログラムで実現され、rdinv_P_req と同じ動作を行なう。

注 1 ： DID フィールドの値が all または MMC の場合、MMC は パケットを受理する。

注 2 ： MMC が発する request に対し、キャッシュがリプライしない場合は、 MMC 自身でリプライする必要がある。ただし、 MMC から MMC へのパケットの受渡しにバスを使えないため、 1 クロックの遅延後、直接転送する必要がある。

注 4 ： PEND 信号を発生するための PEND_CNTR の増減と、それに伴う処理は、全ての reqest 系パケットに共通の処理であるが、 rdinv_req では重要であるため、特に示した。なお、 PEND 信号は、 PEND_CNTR が 0 でない場合に送出される。

```
((in.retry == 1)||(in.crply == 1)) -> EXIT;

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] + 1;

(1) -> TRAP(in);

((in.ack ==1)||(in.sid==CC0)||(in.sid==CC1)||
 (in.sid==CC2)||(in.sid==CC3)) -> pend_cntr[sid] <- pend_cntr[sid] - 1;

EXIT;
```

4.3.12 non_coherent_write_M_req

non_coherent_write_M_req は、メモリへの m 同期つき（ＭＢＰコア呼び出しに使用される）非コヒーレント書き込み要求である。

4.3.13 swap_req

swap_req は、バスに接続されたメモリへのスワップ要求であり、パケット長は２である。リクエストがNon-cacheable の場合には、自明の スワップ要求であり、動作の結果 swap_reply パケットとして返る。リクエストがキャッシャブルの場合には各キャッシュは、 update プロトコルに対応するスワップ動作を行なうため、 swap_req を使用しない。従って CCBL==1 の swap_req パケットの動作は定義されない。

MMC における実現で、 non cacheable な swap 時にだけ用いられることから、全ての動作を DSP プログラムで実現する。

4.3.14 swap_M_req

swap_M_req は、バスに接続された返値を必要とするDSP プログラム呼び出し要求であり、パケット長は２である。リクエストはNon-cacheable であり、動作の結果 swap_M_reply パケットとして返る。

MMC では、無条件のDSP プログラム呼び出しとして扱う。

4.3.15 update_direct_S_req

update_direct_S_req は、生産者－消費者同期を実現するs-write 要求である。同期以外の点では、 update_direct_req に準拠するが、ＭＭＣにおける実現では無条件でＤＳＰプログラム呼び出しとなる。

4.3.16 update_via_home_S_req

update_via_home_S_req は、ホームページを経由した s-write 同期アクセス要求である。無条件で DSP プログラムを呼び出す。

4.3.17 update_via_home_Q_req

update_via_home_Q_req は、ホームページを経由した q-write 同期アクセス要求である。無条件で DSP プログラムを呼び出す。

4.3.18 cflush_req

cflush_req は、キャッシュメモリのブロックをホームまでフラッシュする命令であり、無条件 DSP プログラム呼び出しである。

4.3.19 control_read_req

control_read_req は、バスに接続されたメモリからの内部情報読み出し要求であり、パケット長は 1 である。MMC は、パケットの DID が all または MMC である場合にパケットを受理し、以下の動作を開始する。

MMC はまず、ＤＳＰからセットされる BLK_RSV レジスタの内容とアドレスを比較し、ＤＳＰにより予約されているブロックが対象である場合、 RETRY 信号を assert する。なお、 control_read_req の場合、アドレス体系がクラスタアドレスでないことに注意する必要がある。

さもなくば、パケットを入力バッファに取り込み、処理を開始する。 control_read_req は無条件 DSP プログラム呼び出しとして処理される。

4.3.20 control_write_req

control_write_req は、バスに接続された各デバイスの持つレジスタ、メモリ等への書き込み命令である。書き込みが完了するとアクノレッジ (write_reply) を返す場合がある。MMC における実現では、 control_write_req は全て DSP プログラム呼び出しとして処理される。

4.3.21 demap_req

demap_req は、バスに接続された各キャッシュおよび CPU にたいする demap 要求である。MMC における実現では、MMC は demap_req を受理しない。

4.3.22 interrupt_req

interrupt_req は、バスに接続された各キャッシュおよび CPU にたいする 割り込み要求である。MMC における実現では、MMC は demap_req を受理しない。

4.3.23 read_reply

read_reply は read_req の読み出し要求に対してのリプライパケットである。MMCにおける実現では、MMCは DID が MBP(MCC) である read_reply パケットだけを受理し、全て DSP への入力バッファへ取り込み、DSP プログラムから処理される。

4.3.24 rdinv_reply

rdinv_reply は rdinv_req 等の読み出し要求に対してのリプライパケットである。各キャッシュメモリはスヌープを行ない、DID で示すもの以外のキャッシュに存在するブロックは全てインバリデートされる。

MMCにおける実現では、MMCは DID が MBP(MCC) であるリプライ パケットだけを受理し、全て DSP への入力バッファへ取り込み、DSP プログラムから処理される。

4.3.25 update_req

update_req は update_via_home_req 等の読み出し要求に対してのリプライパケットであるが、パケットの優先度としてはリクエスト パケットである。各キャッシュメモリはスヌープを行ない、対象以外のキャッシュに存在するブロックを全て update する。

update_req は常にＭＢＰが生成し、クラスタ内の各キャッシュに存在するデータを update する。従ってMMCは、update_req を受理しない。

4.3.26 invalidate_req

invalidate_req は invalidate_req 等の読み出し要求に対しての返答であるが、処理の都合上 request レベルである。invalidate_req に対応するリプライパケットはない。

従って、MMC は invalidate_req を受理しない。

4.3.27 demap_ack

demap_ack は demap_req の要求に対してのリプライパケットであり、動作が終了したことを通知する。

MMCは DID=all または MMC である demap_ack パケットを受理し、無条件で DSP プログラムに制御を渡す。

4.3.28 nop

nop は 何もしない動作のパケットであり、当然何のリプライも引き起こさない。nop パケットは出しかけのリクエストを取り消す時などに使われる。

MMC は nop パケットを受理しない。

5 DSP へのインターフェイス

5.1 概説

MCIF は MMC と DSP とのインターフェイスである。MCIF の果たす機能は次の通りである：

1. DSP からのメモリアクセスを MMC にインターフェイスする

   - 32 ビット語単位の read/write
   - 32 バイトブロック単位のブロック read/write
   - MMC 内部情報の read/write

2. MMC から DSP への割り込みをインターフェイスする

   - DSP 呼び出しパケット割り込み
   - メモリアクセス条件割り込み(STAG の TRAP ビットによる指定)
   - メモリエラー割り込み(single error, double error, parity error)
   - ハードウェア・エラー割り込み (time out、非許可ステートなど)

3. MMC を経由してパケットを共通バスと授受する

   - パケット送信
   - パケット受信

また、論理的目的ではないが、DSP と MMC のタイミングにおける差を吸収することも MCIF の目的である。

本仕様書では、MCIF と MMC 間のインターフェイス仕様および、動作の概要を規定する。

5.2 MCIF との接続

MCIF と MMC 間は、以下に示す信号線で結合している。なお、MCIF と MMC 間の信号線の速度は、主クロックである５０ＭＨｚの１／２である ２５ＭＨｚであることに注意すること。なお、信号線の向きは MMC チップを主体として見ている。すなわち、MMC チップが入力するものを入力と記す。信号は全て ５Ｖ TTL レベルである。

MCIF 接続信号線

<table>
<tr><th>信号名</th><th>本数</th><th>方向</th><th>レベル</th><th>記述</th></tr>
<tr><td>HCLK</td><td>1</td><td>入力</td><td>正論理</td><td>MCIF とのインターフェイスに使用するクロック。２５MHｚクロックを入力する。クロックはH→Lのエッジを用い、L→Hのエッジは利用しない。</td></tr>
<tr><td>X_MCREQ</td><td>1</td><td>入力</td><td>負論理</td><td>MCIF からの アクセス要求／アクセス制御信号.
MCIF は MMC へアクセス要求する場合， X_MCREQ を assert し， MCRD, X_BLOCK により，アクセス・モードを伝える． X_MCREQ が assert された最初のクロック・サイクル（アクセス開始サイクル）では， MCDATA 上にアクセス・アドレスが出力される． MCRD および MCDATA 上のアドレスは当該クロックでサンプリングされる。
MCIF からのアクセスが read である場合（すなわち、 read_block, write, write_block でない）は、次クロックで X_MCREQ は negate される。 MCIF からのアクセスが write である場合は、２クロック間（すなわち、書き込みデータの転送が終了するまで）保持され、３クロック目で negate される。
ブロックモードの read / write の場合は、２クロック目以降で X_MCREQ が assert されていることは、次のデータ要求（read_block）または、次の有効データの存在を示している。
従って、アクセス開始後， MMC が X_MCRDY を出してアクセスが終了するまでの間は， X_MCREQ は，アクセス・モードにより，以下の意味を持つ.
<table>
<tr><th rowspan="2">Access Mode</th><th colspan="2">X_MCREQ</th></tr>
<tr><th>L</th><th>H</th></tr>
<tr><td>read 時
block read 時
write 時
block write 時</td><td>-
(Next) Data Request
２クロック目
MCDATA is Valid</td><td>必ず１である
Current Data Hold Req.
３クロック目以降
MCDATA is Invalid</td></tr>
</table>
</td></tr>
<tr><td>X_BLOCK</td><td>1</td><td>入力</td><td>負論理</td><td>MCIF からのアクセス要求がブロックアクセス要求であり続けることをしめすレベル信号。ブロック信号が assert されている期間は同一ブロックアクセスであり、アドレス送致が必要ないことを示す</td></tr>
<tr><td>MCRD</td><td>1</td><td>入力</td><td>正論理</td><td>MCIF からのアクセス要求が read, block read, であることを示す。 X_MCREQ が assert されている期間に有効である</td></tr>
<tr><td>MCDATA</td><td>32</td><td>入出力</td><td>正論理</td><td>MCIF とのデータおよびアドレスの転送に用いるデータバス。一般に、 X_MCREQ 信号が assert されている期間はアドレスを、それ以外ではデータを転送する</td></tr>
<tr><td>X_MCRDY</td><td>1</td><td>出力</td><td>負論理</td><td>MCIF からのアクセス完了を通知する信号。 read の場合、 X_MCRDY が assert されているクロックでの MCDATA には、読み出し値が入っている。 write の場合には、前のアクセスが完了しているため、新たな write リクエストを出して良いことを示す。リクエストを受理していない場合には、 assert されている場合に新たなアクセスを開始して良いことを示す。 read_block では、最初のデータが ready となったことを通知し、以後転送が終了するまでは ready 状態に保たれる。 write_block では、ブロックモード中は常に assert され続け、ブロックモードから脱して実際の書き込みに入った段階で、 negate される。書き込みが終り、次の要求を受理可能となった時点で再び assert される。</td></tr>
<tr><td>X_MCINT</td><td>1</td><td>出力</td><td>負論理</td><td>MMC からの割り込みを通知する信号</td></tr>
</table>

5.3 MCIF からのアクセス

MCIF から MMC へは、read, write, block_read, block_write アクセスを行なう。アクセス対象は、SDRAM の DATA、DTAG、SRAM の STAG、および MMC 内部レジスタである。これらアクセス対象は MMC から送られるアドレス語の値により識別される。

5.3.1 read アクセス

MCIF が X_MCREQ, MCRD = 1 とアクセス対象のアドレスを同時に送り、アドレス中の Access_Mode フィールドが NON_BLOCK であることにより、read アクセスが開始される。ただし、MCIF が X_MCREQ を送ったクロックに X_MCRDY が assert されていない場合、送ったリクエストは MMC により受理されない。従って、再送が必要である。(X_MCRDY が assert されていない場合に X_MCREQ を送ることにより、X_MCRDY が assert されてから最小のレイテンシでアクセスを行なうことが可能となる)

MMC が read 要求を受理すると、直ちに X_MCRDY をクリアする。同時にアドレスに従って、DRAM、SRAM または内部レジスタをアクセスする。

アクセスが終了すると、X_MCRDY を assert するとともに、読み出したデータを MCDATA を用いて転送し、アクセスを終了する。MCIF は X_MCRDY を常に監視し、assert された時点で直ちにデータを取り込む必要がある。

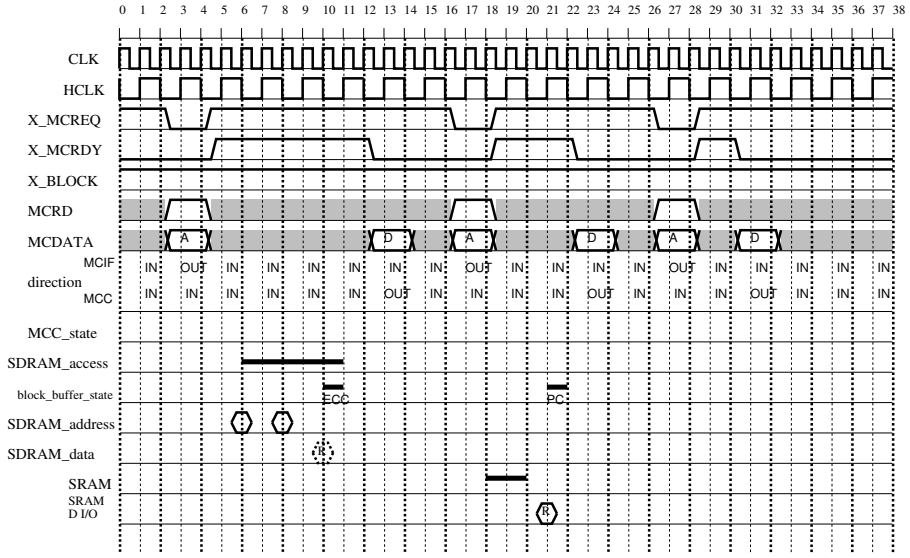

図 15: read アクセスのタイミング

5.3.2 write アクセス

MCIF が X_MCREQ, ～MCRD とアクセス対象のアドレスを同時に送り、アドレスの Access_Mode フィールドが NON_BLOCK であることにより、write アクセスが開始される。ただし、MCIF が X_MCREQ を送ったクロックに X_MCRDY が assert されていない場合、送ったリクエストは MMC により受理されない。従って、再送が必要である。(X_MCRDY が assert されていない場合に X_MCREQ を送ることにより、X_MCRDY が assert されてから最小のレイテンシでアクセスを行なうことが可能となる)

MCIF は、write 要求が受理されると引き続くクロックで書き込まれるデータを送る。write アクセスでは、アドレスに続き 1 個の 3 2 ビットデータを送りアクセスが終了する。MCIF はデータ転送を行なう 2 クロック目まで、X_MCREQ を assert し続ける。

MMC が write 要求を受理すると、直ちに X_MCRDY をクリアする。同時にアドレスに従って、受理したデータを DRAM、SRAM または内部レジスタに書き込む。

アクセスが終了すると、X_MCRDY を assert する。MCIF は X_MCRDY を常に監視し、assert された時点以降で次の X_MCREQ を発行する

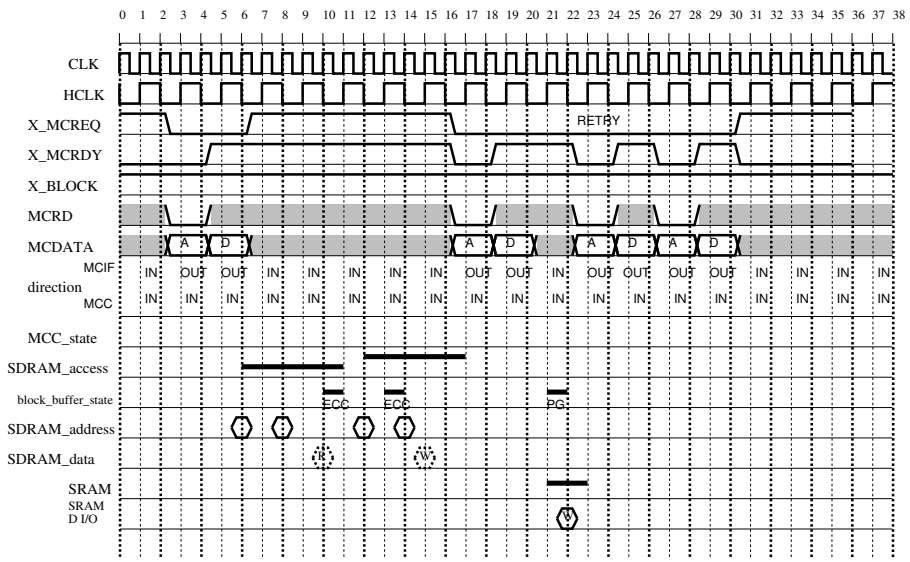

図 16: write アクセスのタイミング

5.3.3 read_block アクセス

read_block アクセスは、 MCIF が MCC に 1 回アドレスを送ることにより、 1 ブロック長のデータを連続して受理するアクセスモードである。 MCIF における設計上の問題点は、 DSP から見るとブロックアクセスでも要素毎の独立したメモリアクセスであり、ブロック途中で割り込み等により中断される可能性があることである。

MCIF は、ブロックアクセスの最初である load 要求として、 Access_Mode が BLOCK_START であるアドレスを受けとると、ブロックモードに入る。 MMC は Access_Mode が BLOCK_START かつ MCRD を X_MCREQ と同時に受理すると、ブロックモードに入る。

MMC はブロックモードに入ると、アクセスする 1 ブロック（ 8 語）を DRAM/ SRAM/ 内部レジスタから読み出す。ブロックモードにおける転送の順序は、 read_block 要求における語アドレスを A[27:0]とすると、以下の順序である。

順序	アドレス
0	A[27:3]— (A[2:0]& 6)
1	A[27:3]— (A[2:0]— 1)
2	A[27:3]— ((A[2:0]+ 2) & 6)
3	A[27:3]— ((A[2:0]+ 2) — 1)
4	A[27:3]— ((A[2:0]+ 4) & 6)
5	A[27:3]— ((A[2:0]+ 4) — 1)
6	A[27:3]— ((A[2:0]+ 6) & 6)
7	A[27:3]— ((A[2:0]+ 6) — 1)

まず最初にブロック先頭語を MCDATA を用いて転送し、 X_MCRDY でデータ送信を通知する。 MCIF は、ブロックモードに入ると、 MMC から X_MCRDY が assert され、データが送られると DSP に最初の語の値を返す。 MCIF は X_MCRDY が assert されると、直ちにデータを受理する必要がある。 ブロックモードでは、 MCIF が X_MCREQ を出している場合には、 X_MCRDY が assert された次のクロックでは、 2 語目のデータを直ちに出力し MCIF からの受理を待つ。

2 語目以降のアクセスにおいては、 X_MCREQ, X_MCRDY の意味が通常の語アクセスの場合と変わる。 X_MCREQ は、当該クロックで MCDATA を受理し、次の語の要求を示す。従って、 MMC は、 X_MCREQ が assert されていると、次のクロックで、次の語を送致する。 X_MCREQ が assert されていない場合には、そのクロックと同一語を、次クロックで再送する。

DSP は 2 語目以降では Access_Mode は BLOCK_CONT を持ったアドレスで行なう。 MCIF は 1 語目の受理が終ると、 DSP に返送すると共に、 MMC に X_MCRDY を送り、 2 語目を MMC から受理する。 ブロックモードにおいて BLOCK_CONT を持った 読み出し要求を受理すると、直ちに既に受理している 2 語目を DSP に返送す

るとともに、MMC にアドレス転送を伴わない X_MCREQ を X_BLOCK 信号と共に送り次の語を準備する。

MMC は X_BLOCK 信号と共に_MCREQ を受理すると、ブロックモードアクセスと認定し、アドレスを受理せず直ちに次の語のデータを MCDATA 上にのせる。X_MCRDY は read_block の手順が終了するまで assert し続けられる。

MCIF のブロックモードは、MCIF が異なる種類のアクセスを受理した場合、Access_Mode が BLOCK_CONT でないアドレスのアクセスを受理した場合解除される。ブロックモードが解除された次のアクセスは、アドレスから転送される。

MMC は、X_BLOCK が negate されると直ちにブロックモードが解除される。従って、途中で中断された block_read の残りの部分は、通常アクセスとして処理される。

なお、double word の読み出しも block_read を用いることが効率上望ましい。SDRAM の制御上、ブロックリードを行なうオーバーヘッドは高々 2 MMC クロックであるため、DSP からはオーバーヘッドとして見えない。

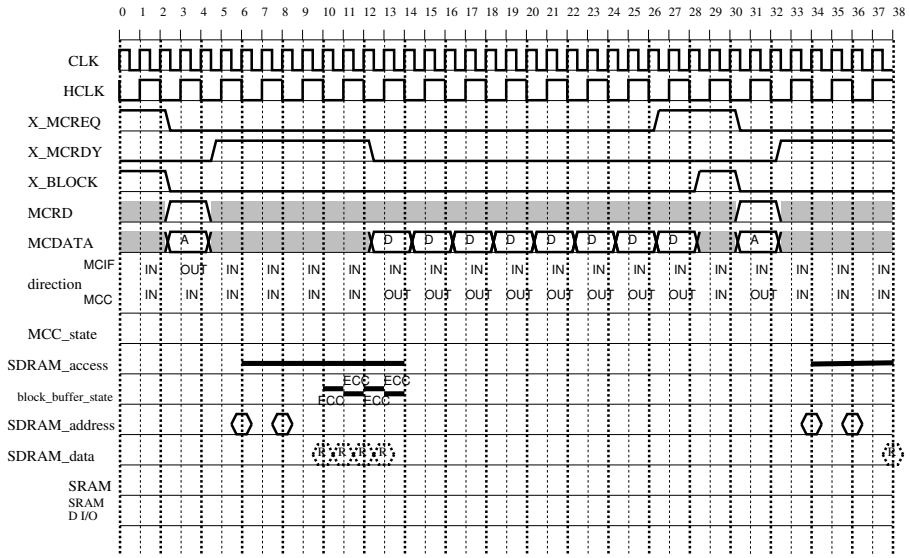

図 17: read_block アクセスのタイミング

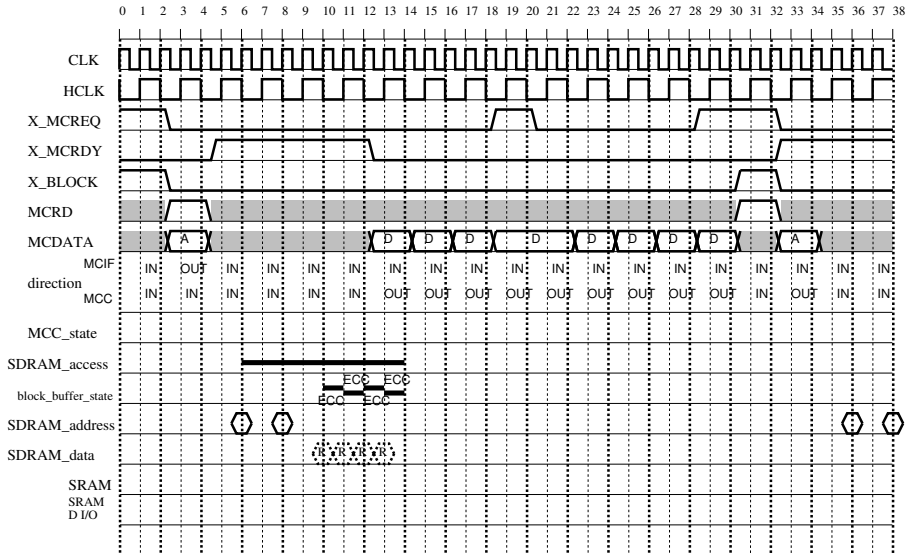

図 18: read_block アクセスのタイミング（要求ウェイトあり）

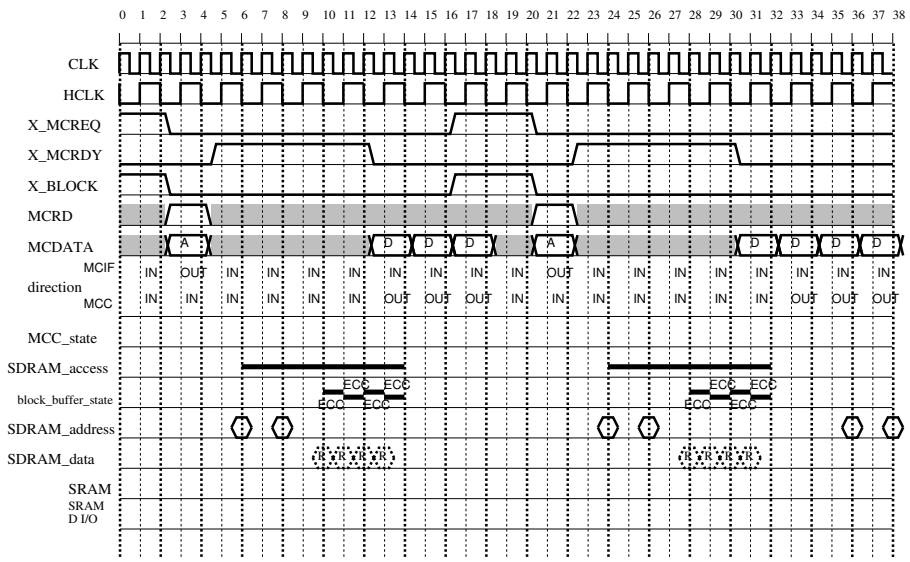

図 19: read_block アクセスのタイミング（モード中断）

5.3.4 write_block

write_block アクセスは、MCIF が MCC に 1 回アドレスを送ることにより、1 ブロック長のデータを連続して書き込むアクセスモードである。MCIF における設計上の問題点は、DSP から見るとブロックアクセスでも要素毎の独立したメモリアクセスであり、ブロック途中で割り込み等により中断される可能性があることである。

MCIF では、アドレス上位の Access_Mode フィールドの設定と、X_BLOCK 信号により、現在の X_MCREQ がブロックモードにおけるデータ要求か、通常の要求かを区別する。MCIF は、ブロックアクセスの最初である store 要求として、Access_Mode が BLOCK_START を受けとると、ブロックモードに入る。MMC は Access_Mode が BLOCK_START である write アクセスを受理すると、ブロックモードに入る。

MMC はブロックモードに入ると、アクセスする 1 ブロック（8 語）を MMC 内ブロックバッファに取り込み、ブロックモードが解除される時点でブロックバッファの内容で、データが存在する部分について書き込みを行なう。まず最初にブロック先頭語を MCDATA を用いてブロックバッファに転送する。MMC は write_block 要求を受理すると、ブロックモードから脱するまで X_MCRDY を assert し続ける。

DSP はブロックの 2 語目からのアクセスは、Access_Mode として BLOCK_CONT を持ったアドレスで行なう。MCIF は ブロックモードにおいて BLOCK_CONT を持った 書き込み要求を受理すると、MMC にデータ転送と共に X_MCREQ 信号を送る。MMC は X_BLOCK 信号と共に X_MCREQ を受理すると、ブロックモードアクセスと認定し、アドレスでなく、直ちにデータを MCDATA 上から受理しブロックバッファに書き込む。

MCIF のブロックモードは、MCIF が異なる種類のアクセスを受理した場合、Access_Mode が BLOCK_CONT でないアドレスのアクセスを受理した場合解除される。ブロックモードが解除された次のアクセスは、アドレスから転送される。

MMC は、X_BLOCK が negate されるとブロックモードを解除する。ブロックモードが解除されると、ブロックバッファ上に存在するデータがメモリに書き込まれる。従って、X_BLOCK 信号が negate されると X_MCRDY 信号が、書き込みが完了し次の要求が受理可能となるまで negate される。途中で中断された block_read の残りの部分は、通常アクセスとして処理される。

なお、double word の書き込みも write_block を用いることが効率上望ましい。SDRAM の制御上、ブロックリードを行なうオーバーヘッドは高々 2 MMC クロックであるため、DSP からはオーバーヘッドとして見えない。

5.4 MCIF への割り込み

すでに述べたように、MMC は下記要因で MCIF に割り込みを通知する。割り込みは全て DSP の同一レベルに対して行なわれる。

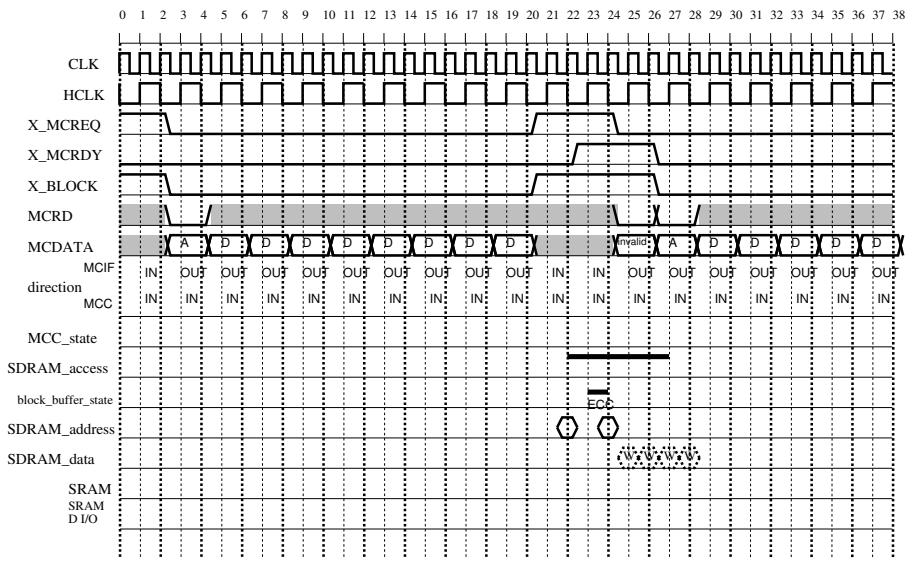

図 20: write_block アクセスのタイミング

DSP 呼び出しパケット割り込み JUMP-1 共通バスからのパケットが到着し、パケットがハードウェアで直接動作可能なもの以外は全て DSP 呼び出しとして割り出す。なお、MMC 中の Status Word 中の設定により、全ての到着した共通パケットが DSP を呼び出すことが指定可能である。

メモリアクセス条件割り込み STAG の TRAP ビットによる指定で、割り込みを発生する。例えば、STAG の RD_TRAP ビットがセットされているブロックに対し、read_req パケットが到着すると、DSP が呼び出される

メモリエラー割り込み JUMP-1 メモリを読み出した結果、single error, double error, parity error を通知する。

ハードウェア・エラー割り込み time out、非許可ステートなど検出されたハードウェアエラー（多くの場合、fatal である）を通知する

DSP 割り込み要因が MCC 中で検出されると、対応する割り込み許可レジスタビットが立っている場合、割り込みを発生する。割り込みが発生すると同時に、割り込み要因レジスタの対応するビットを立て、割り込み原因を DSP から取り込み可能とする。

DSP は、MMC 内部レジスタである割り込み要因レジスタを見ることにより、割り込まれた原因を特定し、必要な場合その他の MMC 内部レジスタを読み出す。更に、割り込み処理が終った時点などで、割り込み要因の解除を行なう。割り込み要因の解除は、要因レジスタの対応するビットをクリアすることにより、得られる。

割り込み処理上問題となる点は、同じ要因または異なる要因による多重割り込みの処理である。要因の異なる多重割り込みは、現在サービスを受けている割り込み処理終了後に処理を行ない、MMC 内部レジスタ上で、割り込みマスク、割り込み要因ともに別ビットを使用することにより実現される。従って、異なる要因間の優先度は DSP ソフトウェアにより設定される。

同じ要因における多重割り込みは以下のように処理される：

共通バスからのパケットが原因の割り込みは、パケットの処理順が保たれることが要請されるため到着順で行なうことを保証する必要がある。従って、割り込み要因が発生した後のパケットは全て リトライするか（request 系パケットの場合）、X_BBUSY を assert する（reply 系パケットの場合）ことが必要である。

エラーが原因の多重割り込みは、最初に発生した割り込みをサービスし、2 個目以下の要因は多重発生を通知する Status Word のビットで記録を残すだけとする。

これらの処理の詳細を以下の節で示す：

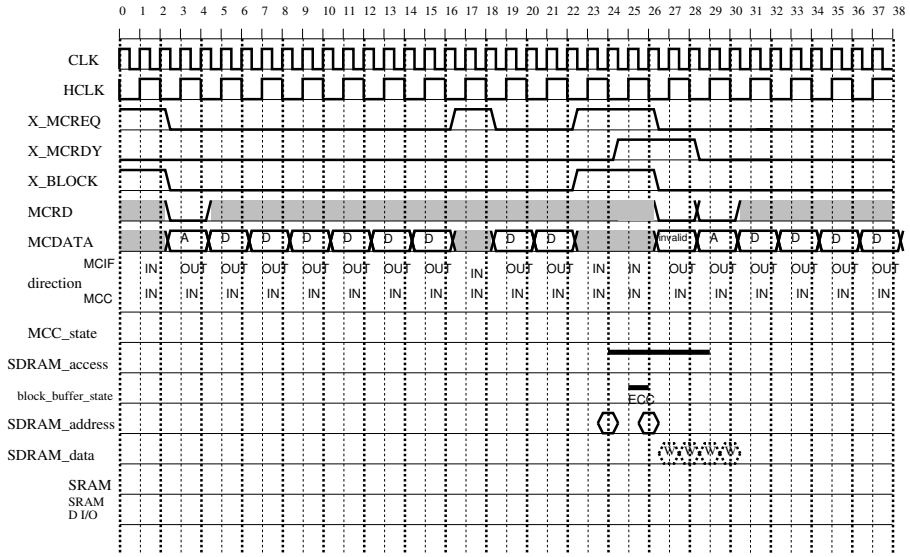

図 21: write_block アクセスのタイミング（出力ウェイトあり）

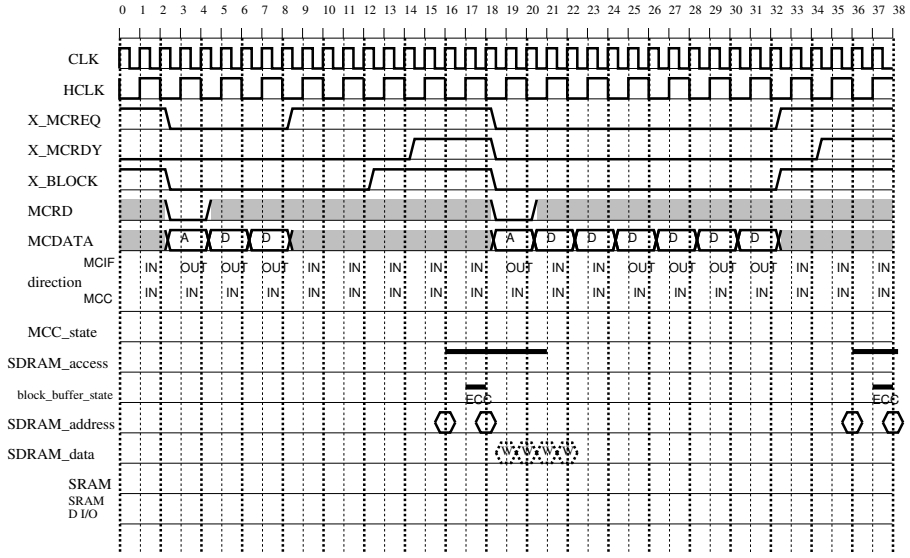

図 22: write_block アクセスのタイミング（モード中断）