SAPPSPC-0010
-10 版: 2010 年 11 月 1 日

スーパーアクセラレータ内蔵メニコア
SAPP64 論理仕様・評価構想

国立大学法人奈良先端科学技術大学院大学情報科学研究科
コンピューティング・アーキテクチャ講座
低電力高性能プロセッサ研究グループ

本書は，国立大学法人奈良先端科学技術大学院大学情報科学研究科コンピューティング・アーキテクチャ講座が発案した「スーパーアクセラレータ内蔵メニコア SAPP64」の論理仕様，および，評価をまとめたものである．前半では SAPP 発案に至る背景，概要，および，論理仕様について詳述する．後半では，SAPP の性能・回路規模・電力について評価するために構築した各モデルについて述べる．性能評価には高速性を有する C-RTL モデルを開発し，実用アプリケーションを走行させた際のサイクル数の測定結果より，アクセラレータを搭載しないシングルコアとの比較を行っている．次に，回路遅延検証のために実用アプリケーションを走行可能な FPGA モデルを開発し，合成配置配線結果から得られる各パイプラインの遅延時間をもとに，C-RTL モデルにおけるパイプライン構成の妥当性を検証し，C-RTL モデルへのフィードバックを行っている．また，ASIC としての回路規模および電力評価には FPGA モデルにおける Verilog 記述を利用した ASIC モデルを開発し，ASIC ライブラリを用いた合成配置配線結果から，演算器レベルの回路規模パラメタおよび電力パラメタの取得を行っている．

最終的に，パイプライン構成の妥当性を検証した C-RTL モデルにより得られる実用アプリケーション実行時のサイクル数，および，ASIC モデルにより得られる回路規模から，回路規模あたりの実効性能について SAPP とメニコア（SAPP 初段のみを SAPP と同一回路規模分並置した構成）を比較することができる．また，C-RTL モデルに電力パラメタを組み込むことにより，実用アプリケーション実行時の電力を評価し，同様に SAPP とメニコアを比較することができる．

おおまかには，単一スレッドどうしの比較では，シングルコアと SAPP アレイモード（36 段構成）の性能比は１：１４，理想的にスレッド分割したメニコアと，単一スレッドにより動作可能な SAPP アレイモードの回路規模あたり性能比は１：１，回路規模あたり消費電力比は，画像処理や一般的な数値計算では８：１，１本の出力配列数が必要とする入力配列数が多い色補正や mgrid では４：１であることが明らかになった．

注意：本仕様書の内容は，予告なしに変更されることがある．

目 次

図 目 次

表 目 次

Chapter 1

背景

1.1 第１の課題は低電力化

最近，動作周波数や命令レベル並列度のいずれもが限界に近付いているために，単一コアの性能向上が鈍化している．代わって注目されているのが，単純コアの並置と並列プログラミングの協調により高性能と低電力を目指すメニコアである．単純コアが低電力であるのは，専ら，高度なスーパスカラに必要となる命令スケジューラ，多ポート大容量レジスタ，および，リザベーションステーションのような電力効率の悪い回路を持たないためである．しかしながら，このような単純コアであっても，最近の実用的プログラムに対してある程度の性能を発揮するためには，相応の容量を有する命令キャッシュ，データキャッシュ，および，多ポートレジスタファイルを備える必要がある．図 1.2 に，当グループが設計した従来型コアの消費電力内訳を示す．このように，命令キャッシュ，データキャッシュ，その他の回路の消費電力が概ね１：１：１となっており，今後，コア数の増加に伴い，メモリやレジスタの消費電力が問題になると考えられる．

ところで，メニコアが適合する並列処理には，機械語命令を用いない専用ハードウェアも適合する．少なくとも命令キャッシュや命令デコータを備える必要がないため，プロセッサよりも明らかに低消費電力である．外部からのデータ流に対して，必要なバッファメモリ，レジスタ，および，演算器のみが動作する状態が，プロセッサにおいて究極の低消費電力化を達成する際の目標であると言える．もちろん，プログラマビリティを損ねてはメニコアとしての存在価値がなくなる．機械語命令を利用しつつ，プロセッサの動作状態をなるべく長期間，専用ハードウェアの動作状態に維持すること，すなわち，図 1.2 に示すように，まず命令キャッシュの動作を停止させ，さらにレジスタおよびデータキャッシュの動作を停止させつつ，従来型プログラムを動作させる仕組みを見つけることが，超低電力メニコアを実現する鍵であると考えられる．

1.2 第２の課題は外部データ供給能力の有効利用

メニコアは，ハードウェアの単純化とコア数の増加により，ILP 追求型単一コアを凌駕する性能を発揮できると考えられているものの，三次元実装等のブレークスルーにより外部データ供給能力が飛躍的に向上するまでは，本質的に，演算ピーク性能と外部データ供給ピーク性能に大きな開きがある．この欠点は，各コアがベクトルレジスタやキャッシュ内ソフトウェア制御可能領域を装備した上で，ソフトウェアが外部から取得したデータを極力コア内部に留保し再利用することにより，補うことが可能であると考えられている．しかし，各コア毎に見ても演算ピーク性能と外部データ供給ピーク性能が一致しない状況では，内部留保したデータが SIMD 型演算機構により処理される時間を利用して，外部データ供給時間の隠蔽を図る従来方法により，理想的な性能を引き出すことは極めて困難である．例えば，SIMD 演算に必要なデータをプリフェッチする場合，演算時間の数倍のプリフェッチ時間は隠蔽することができず，演算効率が極端に低下する．また，プログラムの工夫により，内部留保したデータのみによる SIMD 演算を可能とすることは，アルゴリズムに依存するために極めて困難である．これまでの並列処理は，多数の演算器を遊ばせないために，いかに継続してデータを供給するかについて注力されてきたものの，今後，高性能低電力メニコアを構成する際には，データ供給能力に合わせていかに効率的な演算を行うか，すなわち，コア内部の演算能力に

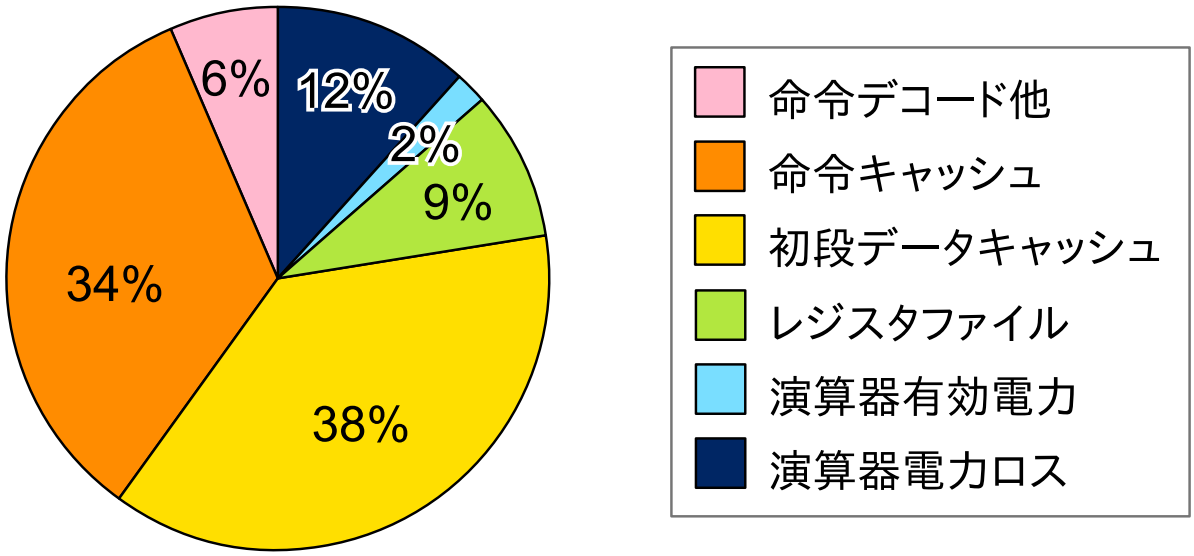


図 1.1: 従来型コアの消費電力内訳

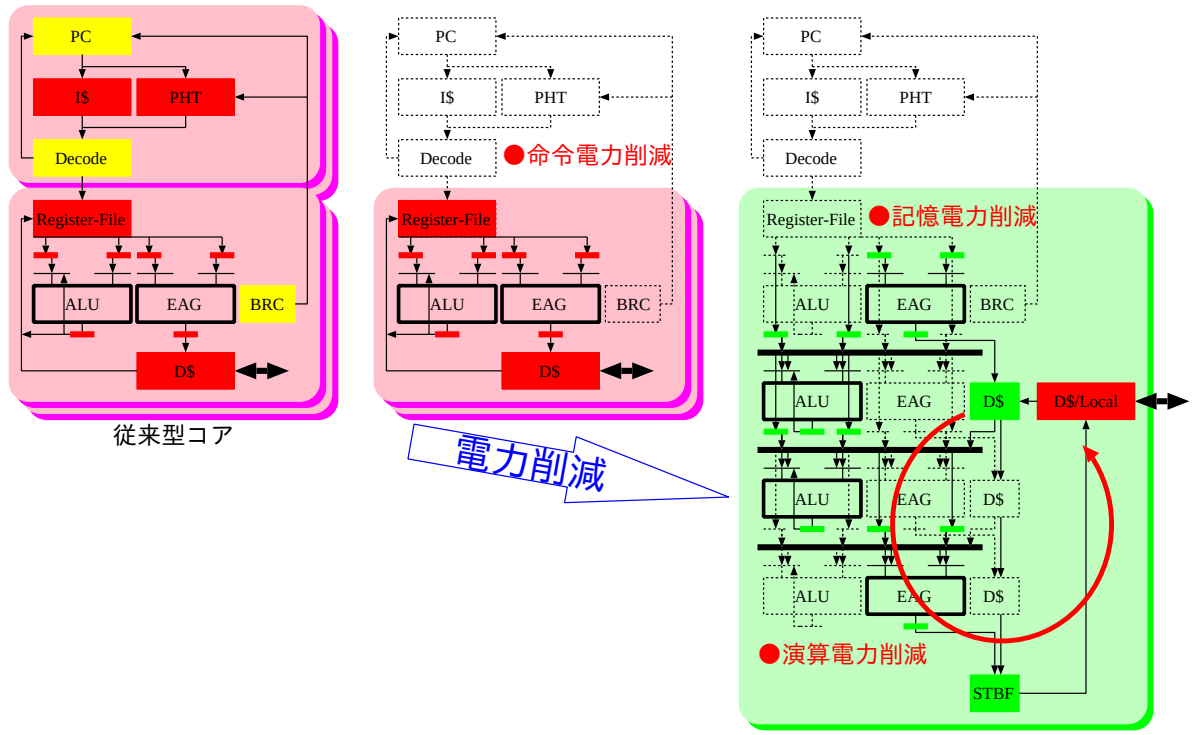


図 1.2: 消費電力削減のアイデア

比べて相対的に貧弱な外部データ供給能力を補うことはもちろん，どのようなプログラムに対してもデータ供給能力を使い切りつつ，必要最小限の内部記憶，演算器および演算器間ネットワークのみを稼働させて低電力化を図ることのできる構造の柔軟さが極めて重要であると言える．

1.3 データ供給能力と演算モデルの適合

図 1.3 に，Load →演算→ Store の一連動作に着目した場合の各実行モデルを示し，メニコア構成時の留意点，および，方向性を示す．まず (1) は，従来型 SIMD 機構の実行モデルである．最小の正方形が 1 要素に対応しており，６４要素からなる８×８の正方形が配列全体を表現している．８×８の正方形のうち左端の黒色部分は先頭の８要素を表現しており，時刻が 1 単位進む毎に８×８の正方形全体が右方向に 1

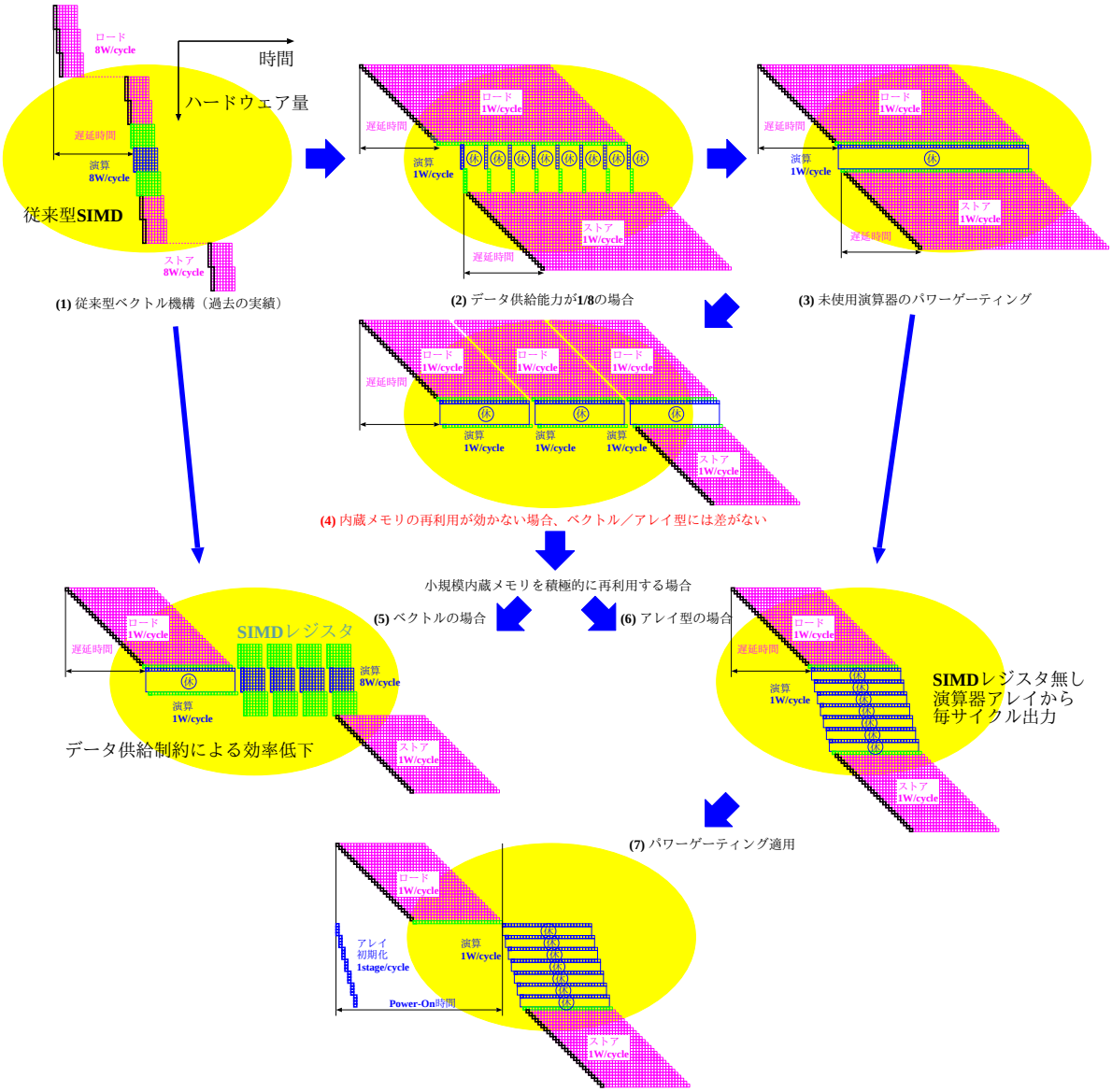


図 1.3: ベクトルと比較した SAPP の実行モデル

単位時間分移動している．先頭の 8 要素は，ロード遅延時間経過後にはレジスタファイル（緑）に到達し，次の時刻には演算器（青）を通過し，出力レジスタファイル（緑）を通過してストアパイプに投入される．すなわち，毎サイクル 8 要素のロード/ストア性能（紫）および演算性能（青）を有する場合，毎サイクル 8 要素の Read/Write が可能なレジスタファイル（緑）を装備することにより最大演算性能を発揮できる．一方，(2) のように，毎サイクル 1 要素のロード/ストア性能しか確保できない場合，最大演算性能の 1 ／ 8 しか発揮できない．8 サイクル毎にしか演算器が動作しないのであれば，(3) のように，稼働させる演算器を絞ることにより，長期間非稼働状態となる演算器に対するパワーゲーティングの適用が可能となる．次に (4) は，3 組の配列要素を入力とし 1 組の配列要素を出力する，より現実的な実行モデル（一般的な積和演算に相当）である．このように常に外部からのデータ供給を必要とするプログラムでは，毎サイクル 8 要素の演算性能は常に過剰である．演算性能を高めるためには，演算に必要なデータを内部留保し，外部からのデータ供給を削減することが極めて重要であると言える．

さて，演算に必要なデータを確実に内部留保する方法として，これまで大きく 2 つの方法が用いられてきた．1 つは，各配列を別々の大容量レジスタファイルに格納しておく方法であり，ベクトルレジスタを用いる従来型 SIMD 機構の実行モデル (5) が典型例である．ただし，ベクトルレジスタの各要素を本来の主記憶アドレスにより個別に参照することはできず，レジスタ間演算が同一要素番号同士に限られることから，配列間演算には強い制約が生じる．もう 1 つは，局所メモリ空間またはキャッシュ内ソフトウェア制御可能

領域を装備し明示的にプリフェッチする方法であり，各要素を主記憶アドレスにより参照できるため要素間演算の自由度は高い．ただし，要素間演算のためには，一旦レジスタにロードする必要があり，４要素同時演算を超える高度な並列演算の実現は，４ウェイを超えるスーパスカラの実現と同様に困難である．すなわち，要素間演算における自由度の高さと，並列演算性能とは，トレードオフの関係にあると言える．並列演算性能の向上を優先するのであれば，ベクトルレジスタの採用が有利であると考えられる．

ところが，ベクトルレジスタ上のデータを再利用することにより，外部データ供給能力を補っているように見える実行モデル (5) は，SIMD 演算中に効率良くデータ供給を行うことが難しい．具体的には，後続ストアに干渉しないロードを SIMD 演算にオーバラップさせようにも，演算性能と外部データ供給性能に差がある場合，スケジューリングは容易ではない．外部データ供給性能を最大限利用する観点からは，SIMD 演算の時間を短縮して隙間をなくすことが重要であるが，SIMD 演算の並列度向上には限界がある．すなわち，演算の多寡に依存することなくデータ供給能力を使い切る観点からは，従来型 SIMD 機構が良い性質を有するとは言えない．

1.4 低電力化とデータ供給能力の有効利用を両立させる方法

従来型 SIMD 型機構が細いデータ供給能力に適合しないのは，演算に要する時間がデータ供給に要する時間と一致しないためである．同一演算を並列実行して高速化する固定観念を捨て，演算に要する時間が常にデータ供給に要する時間と一致するような演算器構成が存在すれは，細いデータ供給能力を常に最大限利用できる実行モデルが構築できるはずである．この考えに基づく実行モデルが (6) である．演算器を直列に配置し，ループイタレーション全体を写像した上で，一方向にデータを流し込むことにより，演算時間とデータ供給時間を一致させている．ところで，この方式は，機械語命令をあらかじめ演算器アレイに写像しておくために，演算中は命令キャッシュや命令デコーダを停止することができる．すなわち，冒頭に述べた「プロセッサの動作状態をなるべく長期間，専用ハードウェアの動作状態に維持する」ことも同時に実現している．

Chapter 2

概要

2.1 命令の写像

図 2.1 は，一般的な輪郭抽出プログラムを VLIW により記述した例である．この命令列は図 2.2 に示すように写像される．命令列が順に実行され演算結果が伝搬していく 1 次元方向に演算器ネットワークを構築すればよいため，演算器数増加に伴うハードウェア量増加は線形に抑えることができる．データを内部留保する手段には，ベクトルレジスタではなく，キャッシュ内ソフトウェア制御可能領域を採用している．要素間演算の自由度を損なうことなく，8 要素同時演算を超える高度な並列演算を実現できるのは，従来型キャッシュがひとまとまりの記憶構造として 1 箇所に存在し，なるべく多くのロード命令を同時に受け付けて性能向上を図る（このこと自体が性能向上の阻害要因となっていることは前述の通りである）のに対し，本構成では，演算器の直列配置と関連付けて，キャッシュ内領域をロード/ストアユニットと共に分散配置するためである．各ロード命令が，どのキャッシュ内ソフトウェア制御可能領域を参照するかは，コンパイル時に静的に決定できるため，各ロード命令が，対応するキャッシュ領域に付随するロード/ストアユニットに発行されるよう静的に命令スケジュールすることは容易である．なお，演算器の段数を超えるループイタレーションは予めイタレーションを分割して写像するか，または，後述する演算器時分割多重化機構により 1/N（N は整数）に圧縮して収容することを想定している．ループ内命令を写像できない場合には，図 2.3 に示すように，一般的な VLIW プロセッサとして初段のみを用いた演算を行う．

以上に述べた構成には，演算時間とデータ供給時間を一致できるだけでなく，SIMD 命令の導入が不要であり，かつ，要素間演算の自由度を維持でき，また，SIMD 演算の多重度を超える高い並列度を達成できる利点がある．さらに，演算結果を一旦ベクトルレジスタ等に書き戻すことなく直接次演算器に供給することにより，消費電力を削減できる効果も期待できる．一方，欠点は，ループイタレーション間にデータ依存関係がある場合に，1 次元方向の演算器ネットワークに命令を写像できないことである．しかし，このような性質のループ構造は従来型 SIMD 機構にも写像できないため，特段の制約条件とは言えない．また，演算器の数が従来型 SIMD 機構に対して圧倒的に多いことは，高い並列度の代償としての欠点であるように見えるものの，一般的なメニコア構成により同数の演算器を確保しようとすると，コア毎に命令キャッシュやデコーダ等が必要となることから，同一性能を達成するために必要な面積は小さい．また，実行モデル (7) に示すように，演算器ネットワークを初期化する期間を利用して，普段は長期間非稼働状態にある演算器のうち，使用するものだけを稼働状態とするパワーゲーティング機構を導入することができるため，同数の演算器を備えたメニコア構成よりも電力効率は高いと考えられる．なお (7) では演算中にデータ供給が行われていないが，演算時間とデータ供給時間が一致することを利用して，次に必要なデータをこの期間にプリフェッチすることは容易である．例えば，直近の 2 つのプリフェッチ先頭アドレスの差分を用いたストライド予測により，演算と同時に行うプリフェッチの先頭アドレスを求めることができる．

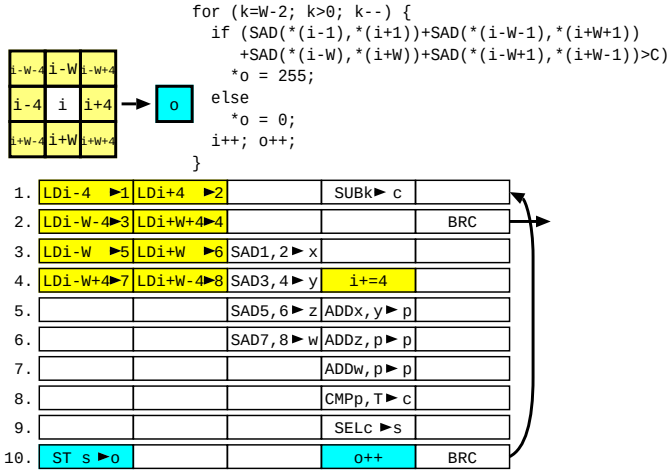


図 2.1: 輪郭抽出を行う VLIW 命令列

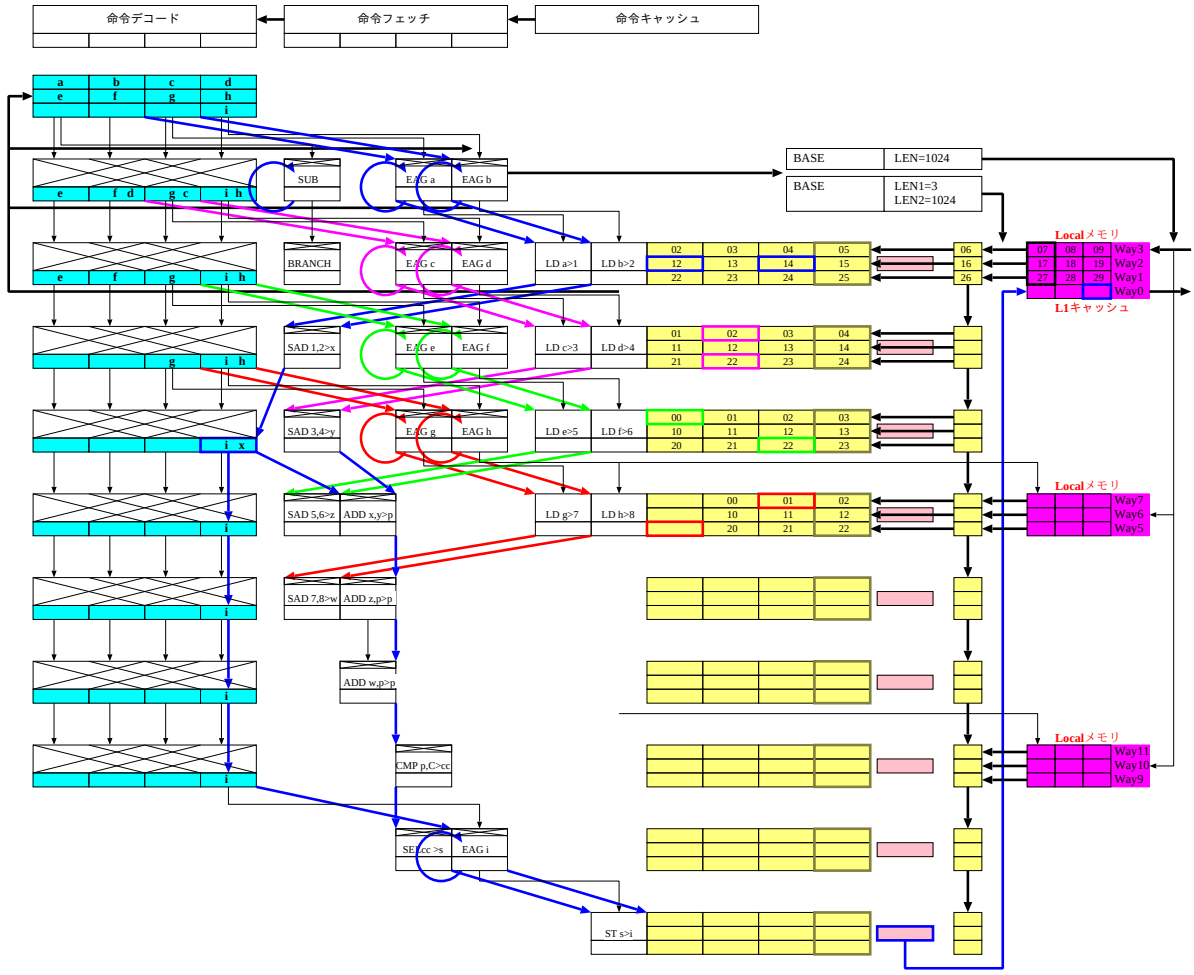


図 2.2: VLIW 命令の写像

2.2 多段演算器構成

図 2.4 に，実行モデル (7) に対応する演算器構成を示す．初段（黒）は，一般的な VLIW プロセッサとして動作する部分であり，命令キャッシュや命令デコーダを備えている．第 2 段（紫），第 3 段（青），第 4 段（緑）が VLIW 構成を 1 次元方向に拡張した部分であり，この段数が，写像可能なループ内 VLIW 命令数を決定する．各段はロード/ストアユニットを備え，初段に属する L1 キャッシュ（way0 は R/W 可能

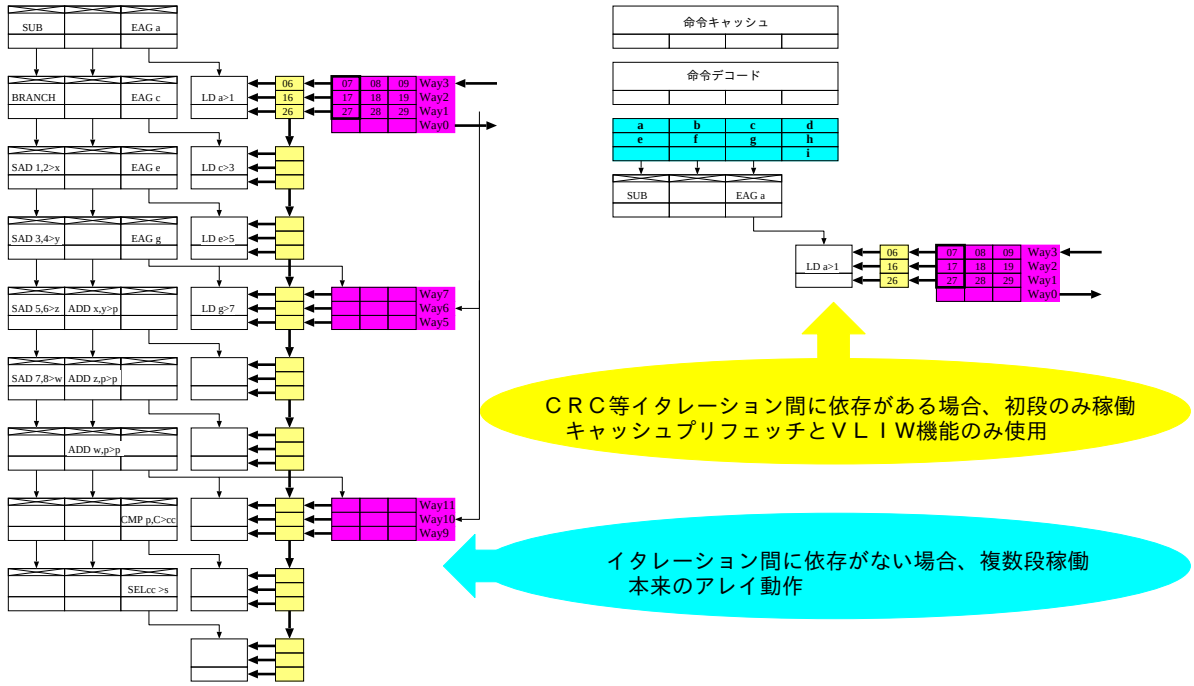


図 2.3: 実行モードの切替え

な一般的なキャッシュ，way1-3 は R/O のキャッシュ内ソフトウェア制御可能領域）の内容が oop4 を経由して順次 L0 キャッシュに伝搬する．この例では，第 3 段に way5-7 として R/O のキャッシュ内ソフトウェア制御可能領域が接続されており，第 3 段のロード/ストアユニットが参照することができる．各段の演算結果やストアバッファの内容は次段へ伝搬され，ストアバッファの内容は，最終的に初段のキャッシュに書き込まれる[1]．

L1 キャッシュの総容量は従来型 SIMD 機構におけるベクトルレジスタの総容量と同程度であり，改めて，L1 キャッシュを均等分割したものがベクトルレジスタであると考えると，L1 キャッシュにプリフェッチした各要素はアドレスにより参照し，ベクトルレジスタにプリフェッチした各要素はベクトルレジスタ番号と要素番号の組により参照していることがわかる．すなわち，SAPP と SIMD 機構の本質的な違いは，(１) 演算時間とデータ供給時間が一致するか不一致か，(２) アドレス情報を残して多様な再利用を可能とするか，アドレス情報を簡略化して比較的単純な演算に特化するかの 2 点に集約されると言える．

また，システムとして見た場合，SAPP と SIMD 機構には，各々スカラ機能が必要である．SAPP の初段はスカラ機能であるため特段のハードウェアは必要ない．一方，SIMD 機構には別途スカラ機能が必要であり，ベクトルレジスタと L1 キャッシュの両方を備える必要があることからメモリ部分の利用効率は低い．L1 キャッシュを除く部分を比較した場合，(１) SAPP はハードウェア資源を演算器および演算器間ネットワークとして利用し，(２) SIMD 機構は演算器間ネットワークの代わりを果たすベクトルレジスタとして利用していると言うこともできる．同一面積あたりのベクトルレジスタ等内部記憶から演算器に対するデータ読み出し能力は同等であり，演算性能は SAPP のほうが高いと言える．

なお一般に，段数が多いほど写像可能な命令数も増加する．しかし，命令数の多寡はプログラムに依存し，多数段に固定された構造では後段の演算器使用率が低下する．図 2.5 に示すように，少数段の構造を基本（サブコア）とし，必要に応じて複数のサブコアを連結する柔軟な構造とすることにより，後段の演算器を有効に利用することができる．４つのサブコアの連結により way0,4,8,12 をストア先として確保し，さらに，ストアバッファを４レーンに増加させることにより，最大４つの配列に対してストアする命令列を写像することもできる[2]．

[1] 図 2.4 では L0 キャッシュ参照のために各段にアドレス計算用の EAG が装備されているが，LD 命令のアドレス情報を L0 キャッシュ内相対位置に置き換えることにより，L0 キャッシュを単一 way かつシフトレジスタにより実装することが可能となる．また，EAG を削減できれば LD-USE ペナルティも 1 段減少するため効率のよい命令写像が可能となる．

[2] 最終段から way0,4,8,12 へ書き戻すための物理的なデータパスの追加が必要である．

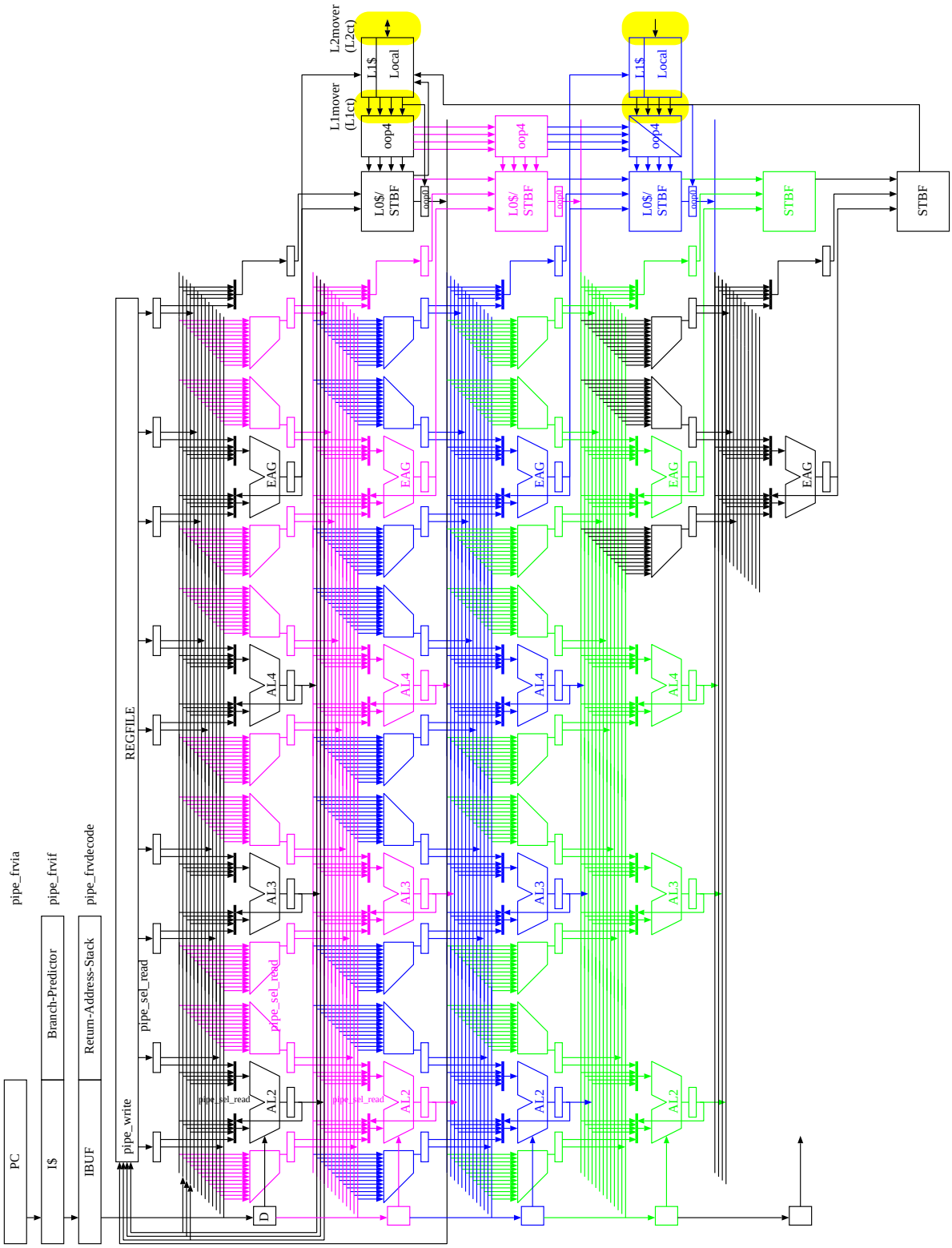


図 2.4: 多段演算器構成

2.3 演算器の時分割多重化

演算器の段数を超えるループイタレーションを 1/N（N は整数）に圧縮して写像し，1/N のスループットに下げて実行する機構が演算器時分割多重化機構である．ただし対応可能な N はモデル依存であり，時分割多重化によっても収容できない場合には，一般的な VLIW プロセッサとして初段のみを用いた演算を行う．図 2.6 に演算器時分割多重化機構を備える構成を示す．図 2.4 に示した基本構成に対して，上下段の

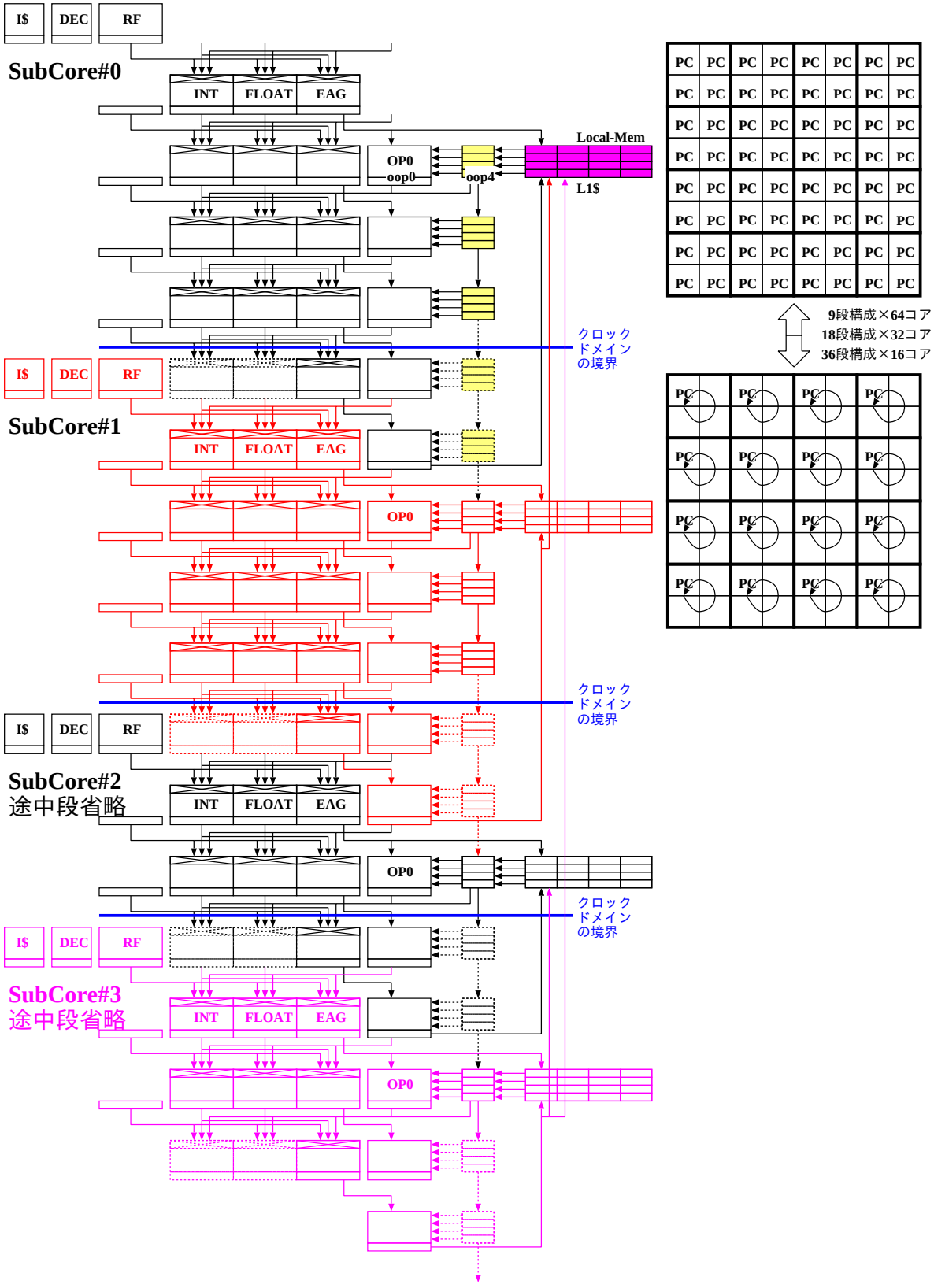


図 2.5: 複数コアの連結による多数段構成

ラッチのいずれかを選択するセレクタ（黒丸部分）が追加されている．本セレクタにより，各演算器に対して連続する N 命令の時分割写像が可能となり，演算器段数の N 倍の命令を写像することができる．すなわち，命令写像ポリシーを変更することなく，命令数と性能のトレードオフのみによる高速実行が可能である．

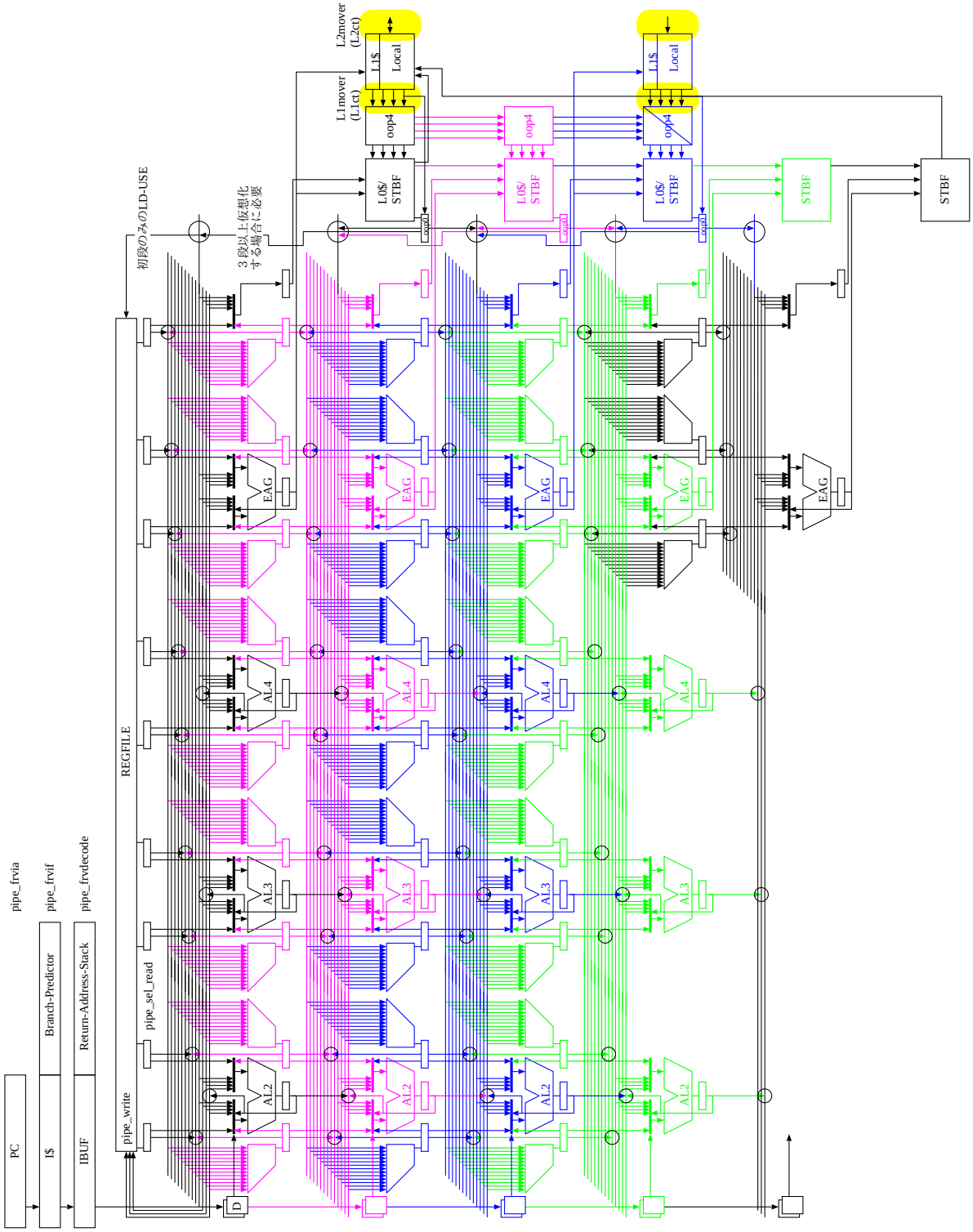


図 2.6: 演算器の時分割多重化

2.4 ベクトルアクセラレータ化

プログラムによっては，従来型ベクトル演算モデルのほうが効率が良いことも考えられる[3]．これまでに説明した演算器アレイ構成では，演算器時分割多重化機構が備えるバイパスを利用して，各段を 1 エレメントの処理に対応付けるベクトルアクセラレータを構成することもできる．図 2.7 にベクトルアクセラレータ化のアイデアを示す．図 2.6 に示した演算器時分割多重化機構のうち，太線部分を使用してベクトル

[3]例えば LD-ADD-ST 等，演算のチェイニングが存在しないケースが該当する．ただし，冒頭に述べたように，十分なデータ供給能力が確保できる場合に限る．

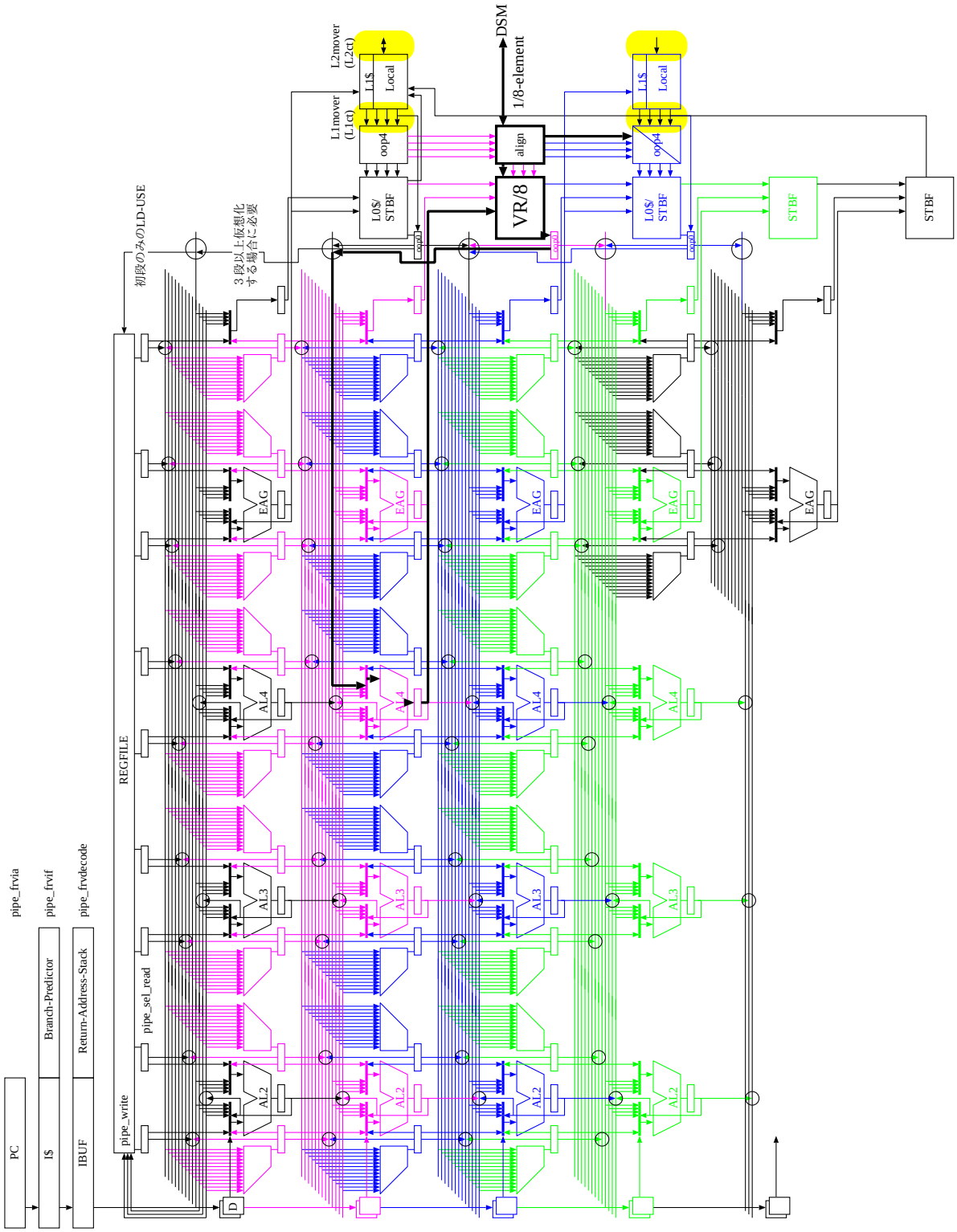


図 2.7: ベクトルアクセラレータ化

ロードパイプおよび演算パイプを構成する．また，L0 キャッシュにはベクトルレジスタ（VR）を収容する．128KB のベクトルレジスタ（8 バイト× 64 要素× 256 本）を備える 8 エレメント構成のベクトルアクセラレータとする場合，1 つの L0 キャッシュには 8 による剰余が同じ要素番号に対応する 16KB の部分ベクトルレジスタを収容する．各 L0 キャッシュには外部メモリインタフェースが接続され，毎サイクル 1 要素のスループットを確保することにより高性能を発揮できる．段間伝搬機構（oop4）には，ベクトルロードとベクトルレジスタの対応付けに必要なアライナとしての機能を持たせる．図 2.8 は，図 2.5 に関連付けたベ

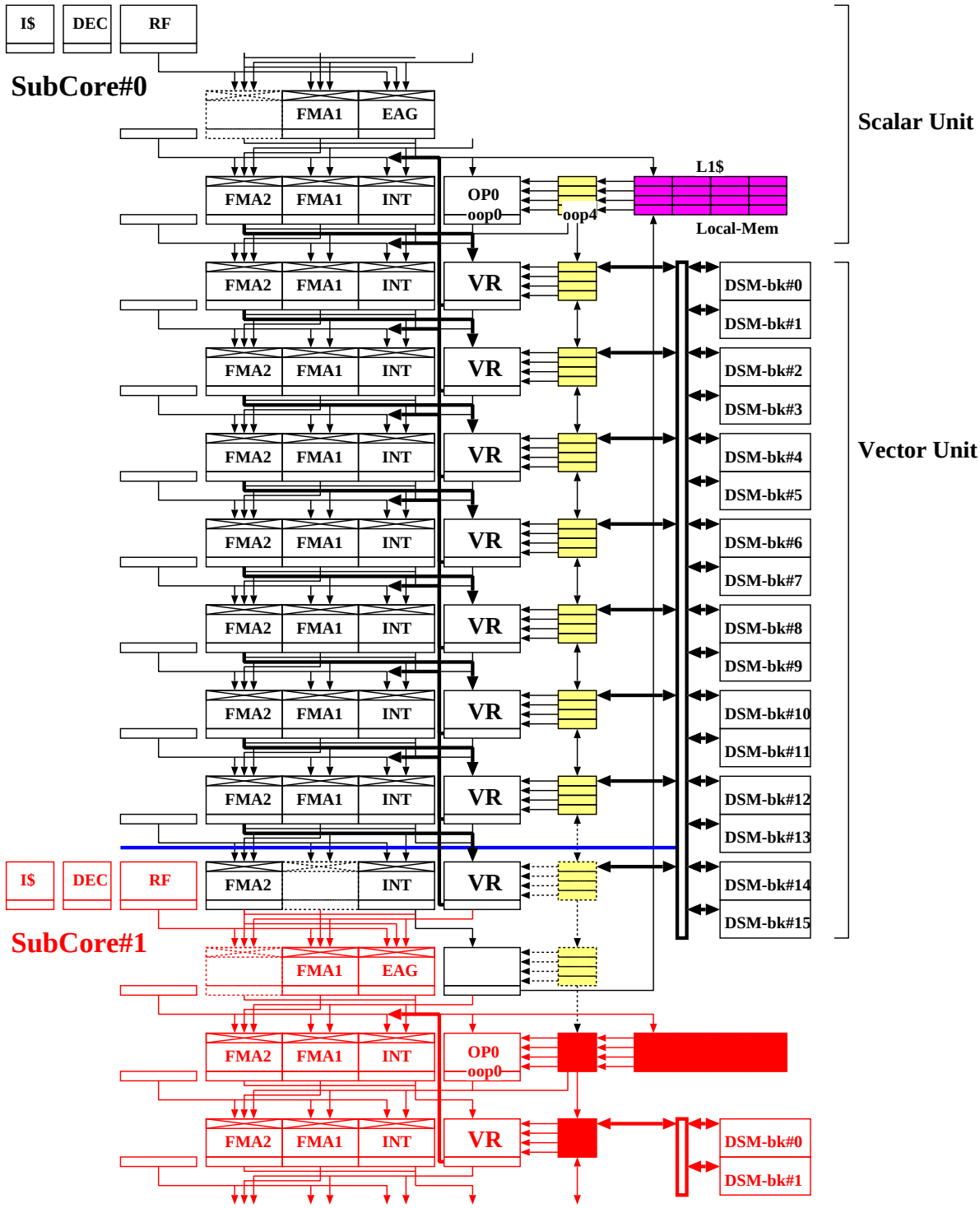


図 2.8: ベクトルアクセラレータモデル

クトルアクセラレータ（８エレメント同時実行）の構成である．初段がスカラユニットに対応し，次段以降がベクトルユニットの各エレメントに対応する．演算器アレイ構成をベクトルアクセラレータとして使用する場合，一般にはベクトル命令列を必要とするが，本構成では，命令写像機構を利用して，一般命令列をベクトル命令に読み替えて実行するモデルを想定している．アドレスが規則的に変化する LD 命令を起点として，関連するレジスタをベクトルレジスタに読み替えて写像する等が考えられる．条件コードレジスタおよび条件付き実行命令をマスクレジスタに読み替えてマスク付き演算とすることも可能である．なお，ベクトルアクセラレータとして動作する場合，各段に同一の演算命令が写像され，各段は独立に動作する．

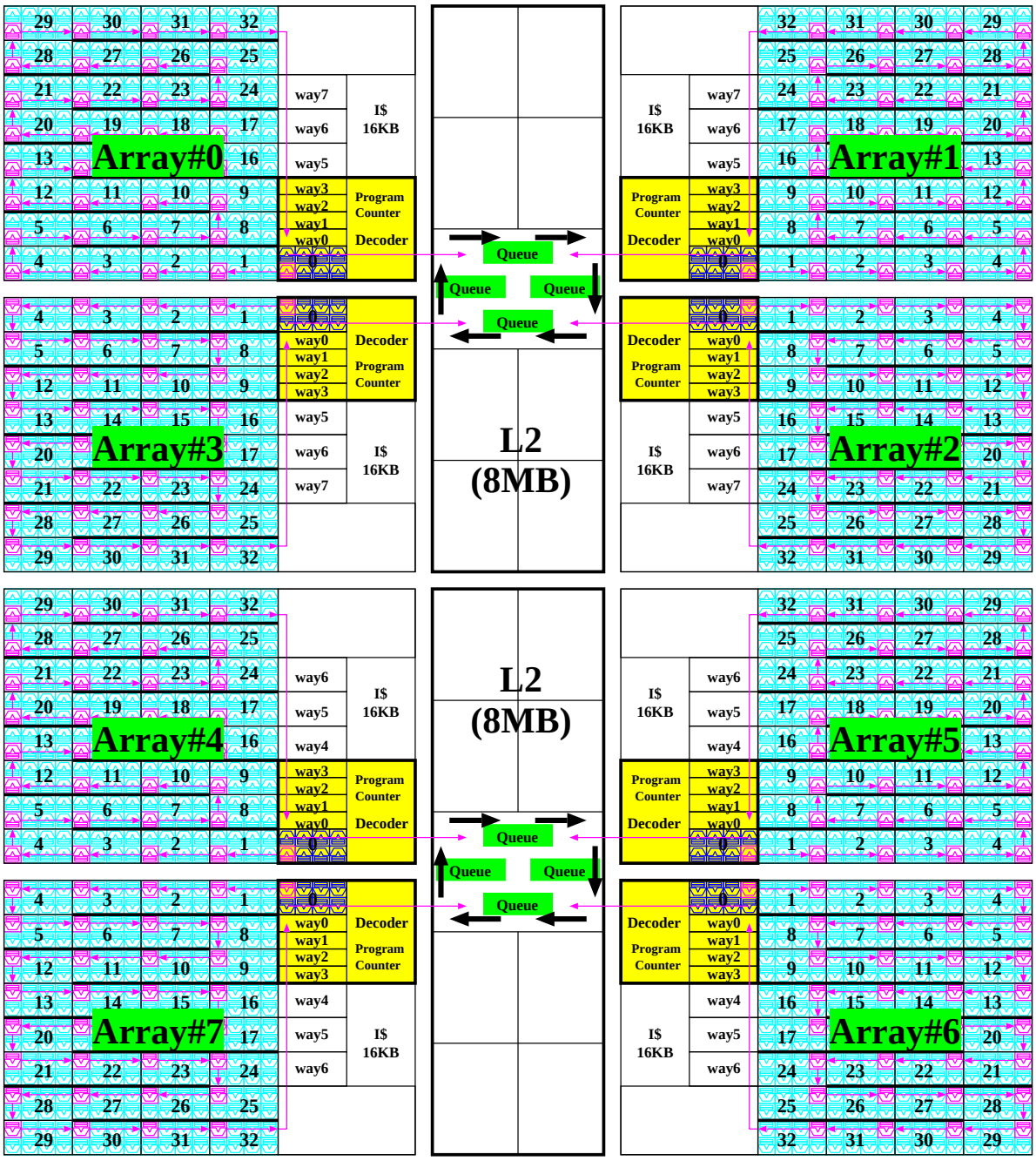


図 2.9: コア間連携との適合

2.5 コア間連携との適合

図 2.9 は，図 2.4 の構造をさらに複数個接続したマルチコア構成を示している．各コアが，写像するループ内 VLIW 命令数に依らず，一定のスループットにより初段の L1 キャッシュに演算結果を書き込む特性を有することから，異なるコアに対して構造の異なるループを対応付けた場合でも均等に負荷分散できると考えられる．この特性を利用しつつ，共有 L2 キャッシュに対するアクセス競合を削減するために，あるコア（例えば Array#1）の演算結果を直ちに次のコア（例えば Array#2）が使用する場合には，L2 への書き込みを行わずに，コア間専用キューを用いた直接データ送受を行う機構を設ける．先頭のコアのみが L2 から読み出しを行い，最後尾のコアのみが L2 へ書き込む状態が理想的な動作状態である．理想的に実行できるアプリケーションプログラムの例を図 2.10 に示す．A[HT][WD] を入力とし，途中結果である B[i]，C[i]，D[i] をコア間で伝搬させつつ，shift(A[i], B[i])，shift(B[i], C[i])，shift(C[i], D[i])，shift(D[i], E[i]) を Array#0 から Array#3 により並列実行し，最終的に E[HT][WD] を出力する．DCPL 命令を演算器ネットワークを

```
#define HT 64
#define WD 64
int   A[HT][WD] __attribute__((aligned(64)));
int   B[HT][WD] __attribute__((aligned(64)));
int   C[HT][WD] __attribute__((aligned(64)));
int   D[HT][WD] __attribute__((aligned(64)));
int   E[HT][WD] __attribute__((aligned(64)));
main(argc, argv) int argc; char **argv;
{
  int i, j;
  int sizeB = sizeof(B);
  int sizeC = sizeof(C);
  int sizeD = sizeof(D);
  int sizeE = sizeof(E);
  for (i=0; i<HT; i++) {
    for (j=0; j<WD; j++)
      A[i][j] = (i<<8)+j;
  }
  if (A-B) { /* Array#0 が担当 */
    __asm__ __volatile__ ("icpl.p %0,%1,#1\n" "bno entry1\n" :: "r"(B), "r"(sizeB));
    for (i=0; i<HT; i++) shift(A[i], B[i]);
    __asm__ __volatile__ ("exit0:" "icpl.p gr0,gr0,#0\n" "bno exit1\n" "entry1:" ::);
  }
  if (B-C) { /* Array#1 が担当 */
    __asm__ __volatile__ ("icpl.p %0,%1,#1\n" "bno entry2\n" :: "r"(C), "r"(sizeC));
    for (i=0; i<HT; i++) shift(B[i], C[i]);
    __asm__ __volatile__ ("exit1:" "icpl.p gr0,gr0,#0\n" "bno exit2\n" "entry2:" ::);
  }
  if (C-D) { /* Array#2 が担当 */
    __asm__ __volatile__ ("icpl.p %0,%1,#1\n" "bno entry3\n" :: "r"(D), "r"(sizeD));
    for (i=0; i<HT; i++) shift(C[i], D[i]);
    __asm__ __volatile__ ("exit2:" "icpl.p gr0,gr0,#0\n" "bno exit3\n" "entry3:" ::);
  }
  if (D-E) { /* Array#3 が担当 */
    __asm__ __volatile__ ("icpl gr0,gr0,#1\n" ::);
    for (i=0; i<HT; i++) shift(D[i], E[i]);
    __asm__ __volatile__ ("exit3:" "icpl gr0,gr0,#0\n" ::);
  }
}
```

```
shift(in:gr4, out:gr5) { /* 各要素を左 1 ビットシフトする */
        addi sp,#-48,sp
        sti.p fp,@(sp,32);    addi sp,#32,fp
        movsg lr,gr11
        sti gr11,@(fp,8)
        addi gr0,65,gr7
        addi gr5,0,gr12
        sethi.p #hi((0<<30)|(0<<29)|(0<<28)|(0<<24)|(64<<12)|64),gr6; setlo #lo((64<<12)|64),gr6
        dcpl.p  gr5,gr6,#1; addi gr4,0,gr0
shift_loop:
        ldi.p @(gr4,0),gr10;  addi.p gr4,#4,gr4; subicc gr7,#1,gr7,icc0
                                                 beq icc0,0x0,shift_exit
        slli gr10,#1,gr11
        st.p gr11,@(gr5,gr0); addi.p gr5,#4,gr5; bra shift_loop
shift_exit:
        addi gr12,0,gr5
        sethi.p #hi((0<<30)|(0<<29)|(0<<28)|(0<<24)|(64<<12)|0),gr6; setlo #lo((64<<12)|0),gr6
        dcpl gr5,gr6,#0
        ldi @(fp,8),gr11
        ldi @(fp,0),fp
        jmpl.p @(gr11,gr0);   addi sp,#48,sp
}
```

図 2.10: 理想的に実行できるプログラム例

利用したコア内高並列処理用ヒント情報として使用し，[j] ループのコア内パイプライン動作を制御する．また，ICPL 命令をコア間連携機能用ヒント情報として使用し，[i] ループのコア間パイプライン動作を制御する．DCPL や ICPL を NOP 命令と見なした場合，通常の単一コア用プログラムと何ら変わらない．一般的

には複雑な並列プログラミングが必要となるこのような処理を単純なヒント命令の挿入のみにより実現できることから，高いプログラマビリティを有する機構であると言える．

Chapter 3

基本方針

本章では，SAPP64 論理仕様検討の初期段階として，いくつかの並列プログラム実行モデルを実現するために必要と考えられるハード・ソフトインタフェースを網羅的に記述している．次章以降に規定される最終的な論理仕様は，本章に記述された仕様候補のフルセットまたはサブセットである．

3.1 実コア番号と論理メモリ空間

SAPP64 では，ユーザプログラムに対して，1 つまたは複数の実コア番号からなるコアグループ，および，論理メモリ空間が提供される．論理メモリ空間は，アドレス範囲によって主記憶装置（DDR メモリ），制御レジスタ，コア分散共有メモリ（DSM），または，クラスタ分散共有メモリ（CSM）に対応付けられている．

DSM 機能および CSM 機能が無効化されている場合は，論理的に透過な L2 キャッシュ用途の内蔵メモリの全てが L2 キャッシュとして主記憶参照の高速化に寄与する．DSM 機能が有効化されている場合は，L2 キャッシュ用途の内蔵メモリの一部（容量はモデルに依存して固定）が論理メモリ空間上の実 DSM 予約空間に実コア番号順に実体化される．DSM 機能利用の有無はプロセス起動時に選択しなければならず，実行途中に変更することはできない．同様に，CSM 機能が有効化されている場合は，L2 キャッシュ用途の内蔵メモリの一部（容量はモデルに依存して固定）が論理メモリ空間上の実 CSM 予約空間に実体化される．CSM 機能利用の有無はプロセス起動時に選択しなければならず，実行途中に変更することはできない．DSM 機能と CSM 機能は独立であり，1 つのプロセスが両機能を同時に利用することができるが，同時利用の場合の L2 キャッシュ容量はシステムが規定する最低容量となる．

DSM/CSM 機能を利用するプロセスの実行は，OS により DSM/CSM 機能が有効化され，かつ，当該プロセス実行に必要な DSM/CSM 資源が確保されるまで，OS により保留される．DSM/CSM 機能を利用するためには，クラスタ単位にハードウェア構成を切り替える必要があり，OS のプロセススケジューラには，ハードウェア構成切り替え機能およびプロセス仕分け機能が要求される．

コア番号変換機構が無効である場合，実コア番号はそのまま物理コア番号（物理位置に直接対応）として使用される．DDR アドレス変換機構（DDR-SAT）が無効である場合，DDR 空間に対応する論理メモリアドレスはそのまま物理 DDR アドレスとして使用される．DSM 機能が有効化されている場合は物理 DSM が実体化され，さらに DSM アドレス変換機構（DSM-SAT）が無効である場合，実 DSM 予約空間に対応する論理メモリアドレスはそのまま物理 DSM アドレスとして使用される．CSM 機能が有効化されている場合は物理 CSM が実体化され，さらに CSM アドレス変換機構（CSM-SAT）が無効である場合，実 CSM 予約空間に対応する論理メモリアドレスはそのまま物理 CSM アドレスとして使用される．

コア番号変換機構と DSM-SAT は独立であり，コア番号変換機構が無効の場合でも，DSM-SAT が有効であれば，実 DSM 予約空間に物理 DSM 領域が物理コア番号順に配置された状態で，各領域内のページ配置を変更することができる（物理 DSM 内の部分故障回避を想定している）．DSM-SAT が無効の場合は，実 DSM 予約空間に物理 DSM 領域が物理コア番号順にそのまま配置される．

コア番号変換機構が有効である場合，実コア番号は，OS が設定するコア番号変換表に基づき物理コア

番号に変換される．実 DSM 予約空間中の実 DSM の並びは常に実コア番号順であり，コア番号変換機構が無効である場合は物理コア番号順に等しい．コア番号変換機構は，使用可能な物理コア番号を意識することなく記述したプログラムを単独または複数同時に実行するための機構である．OS には，障害が発生した物理コア番号を避けたり，複数プログラム間で競合する実コア番号を異なる物理コア番号に割り当てて同時実行を可能とする等の物理コアスケジューリング機能が要求される．

DDR-SAT が有効である場合，DDR 空間に対応する論理メモリアドレスは，OS が設定する DDR アドレス変換表に基づき物理 DDR アドレスに変換される．DSM 機構および DSM-SAT が有効化されている場合，実 DSM 予約空間に対応する論理メモリアドレスは，OS が設定する DSM-SAT の内容に基づき，物理 DSM アドレスに変換される．ただし，物理 DSM 境界をまたぐアドレス変換を行うことはできない．同様に，CSM 機構および CSM-SAT が有効化されている場合，実 CSM 予約空間に対応する論理メモリアドレスは，OS が設定する CSM-SAT の内容に基づき，物理 CSM アドレスに変換される．なお，物理 CSM 境界をまたぐアドレス変換が可能である．

DDR-SAT，DSM-SAT および CSM-SAT は，DDR 空間，実 DSM 予約空間の各 DSM 領域，実 CSM 予約空間よりも狭い各物理メモリ領域を用いて各論理メモリ空間を提供したり，複数プロセスを時分割多重に走行させる仕組みであり，各領域境界を越えて論理メモリ空間を自由に再構成させるものではない（境界をまたぐアドレス変換を前提とするプログラムはアドレス変換機構を無効化した場合に正常動作しない）．

ある論理メモリ空間を参照できる実コア（コアグループのメンバ）は，プロセス起動時にプログラムヘッダ情報を元に OS により使用を許可された実コア数の範囲内においてユーザプログラムが動的に変更可能（主目的は並列度の一時低減による電力削減）である．コアグループに複数の実コアが含まれる状態では，1 つのプロセス中に複数の実コアの内部状態が存在する．個々の実コアの内部状態（プログラムカウンタや汎用レジスタ等）はスレッドと呼びプロセスとは区別する．

3.2 論理メモリ空間の構成

各プロセスに対応付けられる論理メモリ空間は図 3.1 に示すように機能毎に領域分割されている．Firmware 領域，OS 領域，I/O 制御領域は，全ての論理メモリ空間により共有されており物理 DDR アドレス空間および連続する物理メモリ空間上に 1 組の実体が配置される．

`00000001_00000000-0000000f_ffffffff` の範囲はユーザ DDR 領域を実体とするプログラム空間（DDR 空間）であり，MOESI プロトコルによる複数コア間のキャッシュ一貫性制御の対象となる．また，性能保証を目的として L1 キャッシュに Line-Lock 機構が装備される．ただし Lock 可能な範囲は高々 1 つの Text 範囲および 1 つの Stack 範囲であり，非 Lock 範囲のキャッシュ先が確保できない等ハードウェア資源の不足により Lock 指示が無視される場合がある．なお，本空間はアレイモードのアクセラレータ，および，コア間転送機構（DTU）が参照できる．ベクトルモードのアクセラレータは本空間を参照できない．

`00000011_00000000-00000011_0000ffff`，の範囲には自コアの制御レジスタ群が写像されている．L1 キャッシュは無効である．他コアの制御レジスタ群を参照することはできない．

`00000012_00000000-00000012_0001ffff` の範囲は DSM-SAT を指示する制御レジスタ群領域であり，L1 キャッシュは無効である．制御レジスタ群の内容により，物理 DSM 領域の各ページ（ページサイズは DSM-SAT 固有）が実 DSM 予約空間の実 DSM 領域毎に写像される．ページサイズが 4KB の場合，各 1MB の物理 DSM を 64 コア分写像するためには，16K エントリ（各エントリのサイズは 4B）の制御レジスタ群が装備される．さらに，複数プロセスに対して実 DSM 領域を同時提供するために，制御レジスタ群が複数セット設けられる（セット数はモデル依存である）．制御レジスタ群の実体はコア毎に独立であり，コア毎に異なる写像が可能である．なお，実 DSM 予約空間と L1 キャッシュラインの対応は論理アドレスに基づくため，DSM-SAT の内容を変更する場合，実 DSM 予約空間と L1 キャッシュの整合性はソフトウェア（L1 キャッシュフラッシュ命令）により保証しなければならない．

`00000013_00000000-00000013_0001ffff` の範囲は CSM-SAT を指示する制御レジスタ群領域であり，L1 キャッシュは無効である．制御レジスタ群の内容により，物理 CSM 領域の各ページ（ページサイズは CSM-SAT 固有）が実 CSM 予約空間に写像される．ページサイズが 4KB の場合，最大各 16MB の物理 CSM を 4 クラスタ分写像するためには，16K エントリ（各エントリのサイズは 4B）の制御レジスタ群が装

図 3.1: 論理メモリ空間の構成

関連制御レジスタ	保護	値の意味
PEV.DDRSAT	特権	0:DDR-SAT無効, 1:DDR-SAT有効
PEV.DDRPID	特権	DDR-SAT連想検索に使用するプロセスID

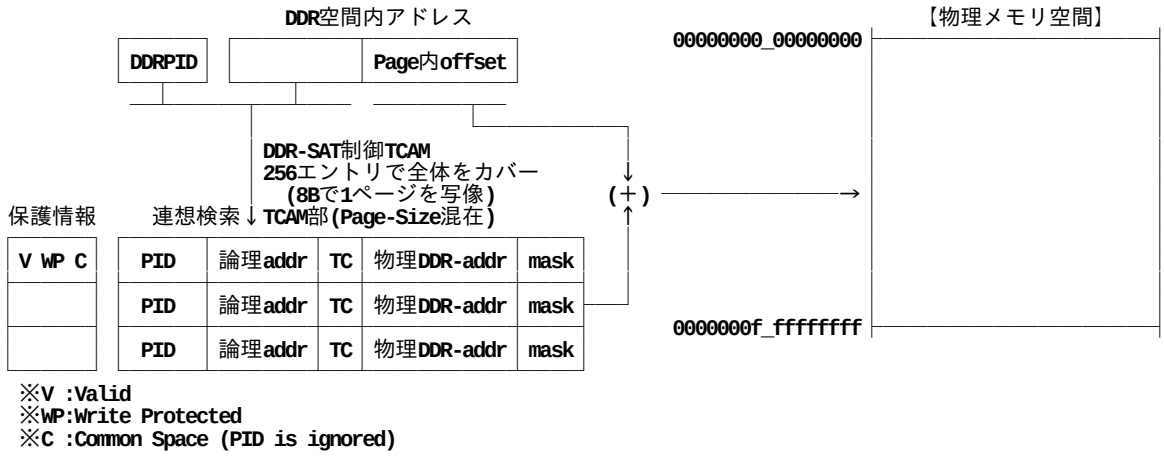


図 3.2: DDR-SAT

備される．さらに，複数プロセスに対して実 CSM 領域を同時提供するために，制御レジスタ群が複数セット設けられる（セット数はモデル依存である）．制御レジスタ群の実体はコア毎に独立であり，コア毎に異なる写像が可能である．

00000020_00000000-00000020_3fffffff の範囲は実 DSM 予約空間である．コア番号変換機構無効かつ DSM-SAT 無効の場合，物理 DSM 領域が物理コア番号順にそのまま配置される．コア番号変換機構無効かつ DSM-SAT 有効の場合，物理 DSM 領域が物理コア番号順に配置された状態で各領域内のページが DSM-SAT の内容に従い配置される．コア番号変換機構有効かつ DSM-SAT 無効の場合，物理 DSM 領域が実コア番号順にそのまま配置される．コア番号変換機構有効かつ DSM-SAT 有効の場合，物理 DSM 領域が実コア番号順に配置された状態で各領域内のページが DSM-SAT の内容に従い配置される．自コアの DSM 領域が写像されているアドレスに対しては通常のロード/ストアが可能（L1 キャッシュはライトスルー）であるものの，他コアの DSM 領域が写像されているアドレスに対しては L1 キャッシュは無効となる．なお，本領域はアレイモードのアクセラレータ（自コアのみ），ベクトルモードのアクセラレータ（自コアのみ），および，コア間転送機構（DTU）が参照できる．

00000030_00000000-00000030_03ffffff の範囲は実 CSM 予約空間であり，L1 キャッシュは無効である．CSM-SAT が無効の場合，物理 CSM 領域がそのまま配置される．各物理 CSM 領域の全アドレス空間にメモリの実体が存在するとは限らないため，CSM-SAT が無効の場合，実 CSM 予約空間においてメモリの実体が存在する領域は不連続となる．CSM-SAT 有効の場合，物理 CSM 領域の各ページが CSM-SAT の内容に従い配置される．なお，本空間はコア間転送（DTU）が参照できる（アレイモードのアクセラレータおよびベクトルモードのアクセラレータは参照できない）[1]．

3.3 DDR アドレス変換機構（DDR-SAT）

DDR-SAT は，各コア内制御レジスタ PEV.DDRSAT の値が 1 である間，有効化される．各コアには DDR-SAT を利用するプロセスを 1 つまたは複数走行させるための 1 組のアドレス変換 TCAM 機構が装備されており，OS は，DDR-SAT を有効化する際，TCAM 機構を適切に初期化しなければならない．DSM-SAT の仕組みを図 3.2 に示す．DDR 空間内アドレスのうち Page 内オフセットを除く上位ビットパターン

[1] アレイモードのアクセラレータは L1 キャッシュを使用するため L1 キャッシュを使用しない他コア DSM 領域および CSM 領域は参照できない．同様にベクトルモードのアクセラレータは自コアの DSM を使用するため DDR 領域，他コア DSM 領域および CSM 領域は参照できない．すなわち，アレイモードとベクトルモードには DDR 領域を参照できるか否かの違いがある．

関連制御レジスタ	保護	値の意味
PEV.DSMSAT	特権	0:DSM-SAT無効, 1:DSM-SAT有効
PEV.DSMSPC	特権	DSM-SATに使用するDSM-SAT制御レジスタ群番号 0:space#0, 1:space#1
PEV.DSMPSZ	特権	DSM-SATのページサイズ 0:4KB, 1:16KB, 2:64KB, 3:256KB, 4:1MB
PEV.CORMAP	特権	0:コア番号変換無効, 1:コア番号変換有効
PEV.CORGRP[0..63]	特権	実コア番号に対応させる物理コア番号 0-63:実コア有効 255:実コア無効

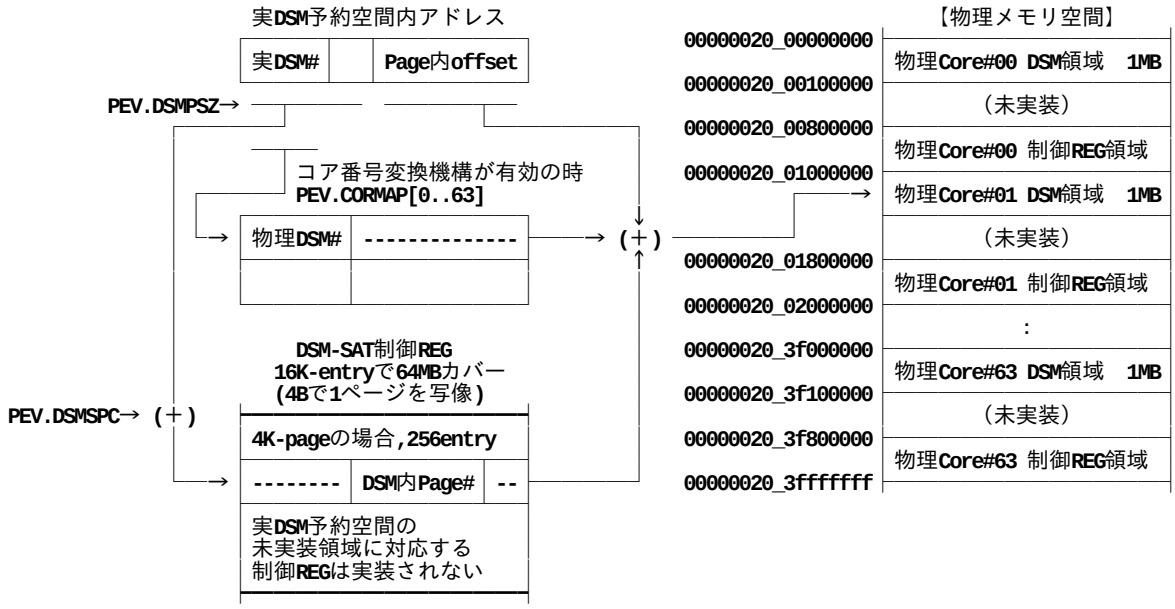


図 3.3: DSM-SAT

に PEV.DDRPID を単純連結した値により，アドレス変換 TCAM 機構が連想検索される．連想検索対象となる TCAM のビット位置の一部（位置はモデル依存）にはマスク機構が装備されており，複数の異なるページサイズを同時に登録可能である．また，保護情報中の C（Common Space）ビットにより，PID を無視するアドレス変換も可能である．連想検索の結果，一致するエントリが検出された場合，当該エントリに付随する領域から物理 DDR アドレスおよびマスク情報が読み出され，Page 内オフセットと連結されて物理 DDR アドレスとなる．一致するエントリがない場合，OS による TCAM 機構の内容更新および当該メモリ参照命令の再実行が求められる[2]．

3.4 DSMアドレス変換機構（DSM-SAT）

DSM-SAT は，各コア内制御レジスタ PEV.DSMSAT の値が 1 である間，有効化される．各コアには DSM-SAT を利用するプロセスを 1 つまたは複数走行させるために，1 組または複数組の DSM-SAT 制御レジスタ群が装備されており，OS は，DSM-SAT を有効化する際，当該プロセスに使用させる DSM-SAT 制御レジスタ群を PEV.DSMSPC により指定しなければならない．また，DSM-SAT のページサイズを PEV.DSMPSZ により指定しなければならない．なお，PEV.DSMSPC および PEV.DSMPSZ に指定可能な値はモデル依存である．DSM-SAT の仕組みを図 3.3 に示す．PEV.DSMSAT の値が 1 の時に実 DSM 予約空間を参照した場合，物理 DSM 番号（当該論理メモリアドレスのうち，実 DSM 番号に該当するビット位置の内容を元にコア番号変換機構から得られる）と，DSM 内ページ番号（当該論理メモリアドレスのうち，実 DSM 番号に該当するビット位置および PEV.DSMPSZ の値により決定されるビット位置の内容を

[2]DSM-SAT および CSM-SAT では写像対象とする DSM/CSM が小容量であるため複数プロセスの同時実行を可能とする各制御レジスタ群が高々数セットしか装備されないのに対し，DDR-SAT は大容量 DDR を写像対象とするため，プロセス ID（PID）により論理メモリ空間を区別する機構，および，CAM によるフルアソシアティブアドレス変換機構が装備される．

関連制御レジスタ	保護	値の意味
PEV.CSMSAT	特権	0:CSM-SAT無効, 1:CSM-SAT有効
PEV.CSMSPC	特権	CSM-SATに使用するCSM-SAT制御レジスタ群番号 0:space#0, 1:space#1
PEV.CSMPSZ	特権	CSM-SATのページサイズ 0:4KB, 1:16KB, 2:64KB, 3:256KB, 4:1MB

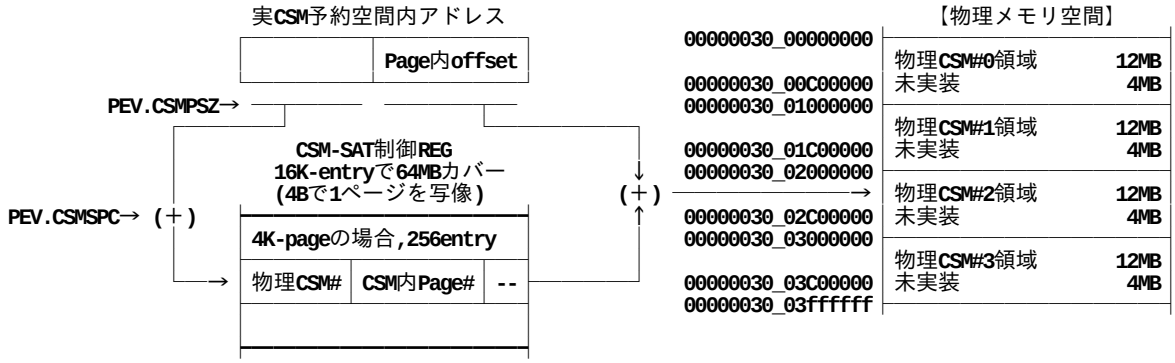


図 3.4: CSM-SAT

インデックスとして，PEV.DSMSPC により選択された DSM-SAT 制御レジスタの 1 つから得られる）と，ページ内オフセット（当該論理メモリアドレスの下位ビットから得られる）の単純連結により得られるアドレスにより物理 DSM 領域が参照される．コア番号変換機構が無効である場合は，実 DSM 番号に該当するビット位置の内容がそのまま物理 DSM 番号として使用される．なお，実 DSM 予約空間と L1 キャッシュラインの対応は論理アドレスに基づくため，DSM-SAT の内容を変更する場合，実 DSM 予約空間と L1 キャッシュの整合性はソフトウェア（L1 キャッシュフラッシュ命令）により保証しなければならない．

3.5 CSM アドレス変換機構（CSM-SAT）

CSM-SAT は，各コア内制御レジスタ PEV.CSMSAT の値が 1 である間，有効化される．各コアには CSM-SAT を利用するプロセスを 1 つまたは複数走行させるために，1 組または複数組の CSM-SAT 制御レジスタ群が装備されており，OS は，CSM-SAT を有効化する際，当該プロセスに使用させる CSM-SAT 制御レジスタ群を PEV.CSMSPC により指定しなければならない．また，CSM-SAT のページサイズを PEV.CSMPSZ により指定しなければならない．なお，PEV.CSMSPC および PEV.CSMPSZ に指定可能な値はモデル依存である．CSM-SAT の仕組みを図 3.4 に示す．PEV.CSMSAT の値が 1 の時に実 CSM 予約空間を参照した場合，物理 CSM 番号および CSM 内ページ番号（当該論理メモリアドレスのうち，PEV.CSMPSZ の値により決定されるビット位置の内容をインデックスとして，PEV.CSMSPC により選択された CSM-SAT 制御レジスタの 1 つから得られる）と，ページ内オフセット（当該論理メモリアドレスの下位ビットから得られる）の単純連結により得られるアドレスにより物理 CSM 領域が参照される．

3.6 キャッシュの構成

SAPP64 のキャッシュ構成を説明するために，図 3.5 にコア数増加とキャッシュ構成の関係を示す．(1) は DDR 空間を参照するコアが 1 個の場合の単純な構成である．(2) は L2 キャッシュの構成を変えずにコア数を増加させる構成，(3) は L2 キャッシュをバンク分割し，参照要求が異なるバンクに分散する限りにおいて複数コアからの要求を同時受け付け可能とする構成である．さらにコア数を増加させる場合，バンク間クロスバネットワークが実現困難になるため，(3) の構成を 1 クラスタとし (4) に示すように DIRECTORY を追加して複数クラスタ構成とする．DDR は領域分割され，各クラスタに接続される．

図 3.6 に想定するキャッシュ構成を示す．図 3.6 は，16 コアから構成されるクラスタを 4 つ相互接続した 64 コア構成を示している．DDR メモリは物理 DDR アドレス空間を等分割した連続空間に各々対応付

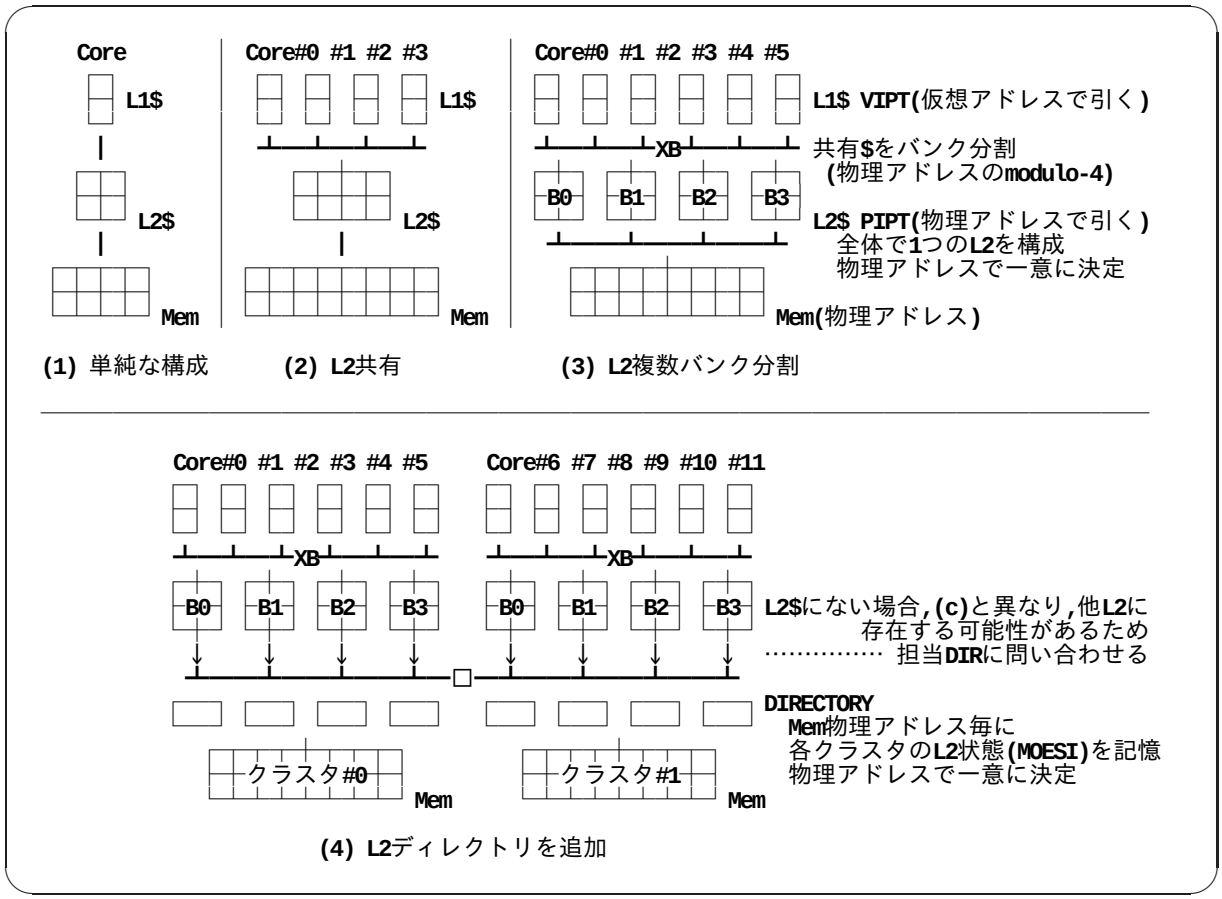


図 3.5: コア数増加とキャッシュ構成の関係

けられており各クラスタに接続されている．各クラスタには 1 組の L2 キャッシュが装備されており，クラスタ内の各コアからはアドレス変換後の物理アドレスにより参照される．また，各 DDR メモリに対応する L2-DIRECTORY（物理 DDR アドレスに対応する有効な L2 キャッシュラインのシステム内物理位置を保持）が同一クラスタ内に配置されている．L2 キャッシュは物理 DDR アドレスによりインタリーブされており，各コアからの参照要求と L2 内バンクの対応は一意に決まる．各 L2-DIRECTORY も各バンクに対応している．

さて，当該バンクにおいて L2 キャッシュミスを検出した場合，参照に使用した物理 DDR アドレスから担当クラスタを求める．自クラスタである場合，L2 直下の L2-DIR に問い合わせを行い，他クラスタの L2 にも存在しなければ（自 L2-DIR に何も登録されていなければ），自クラスタ内 DDR に対して参照要求を発行し自クラスタの L2-TAG を更新する（自 L2-DIR は更新しない）．直下の L2-DIR に問い合わせた結果，他クラスタに存在する場合（自 L2-DIR に登録がある場合），他クラスタ内 L2 に対して参照要求を発行する．読み出しの場合，L2-TAG は Shared 状態 (元 L2 が Clean の場合) または Owned+Shared 状態 (元 L2 が Modified の場合) へ移行し，書き込みの場合は Modified+Invalid 状態 (元 L2 を無効化) へ移行する．

参照に使用した物理 DDR アドレスが他クラスタである場合，参照要求を他クラスタ内の担当 L2 バンクに送出する．担当 L2 バンクは，自身のキャッシュラインが有効である場合は要求元に返信し，無効である場合は直下の L2-DIR に問い合わせを行い，他クラスタになければ，自クラスタの DDR メモリに対して参照要求を発行し要求元に返信した後，L2-DIR を更新する．もし，直下の L2-DIR に問い合わせを行った結果，他クラスタに存在する場合，目的のクラスタ内 L2 に対して参照要求を転送する．目的のクラスタ内 L2 は参照要求に対して同様の処理を行い，最終的には担当 L2-DIR を更新し，キャッシュ内容は要求元に返信する．

図 3.7 に論理メモリ空間とキャッシュの関係を示す.「アクセス先」列は論理メモリ空間の各領域および注目コアの L1 キャッシュの状態により分類されている．L1 キャッシュの状態は，N.C. が L1 キャッシュ常

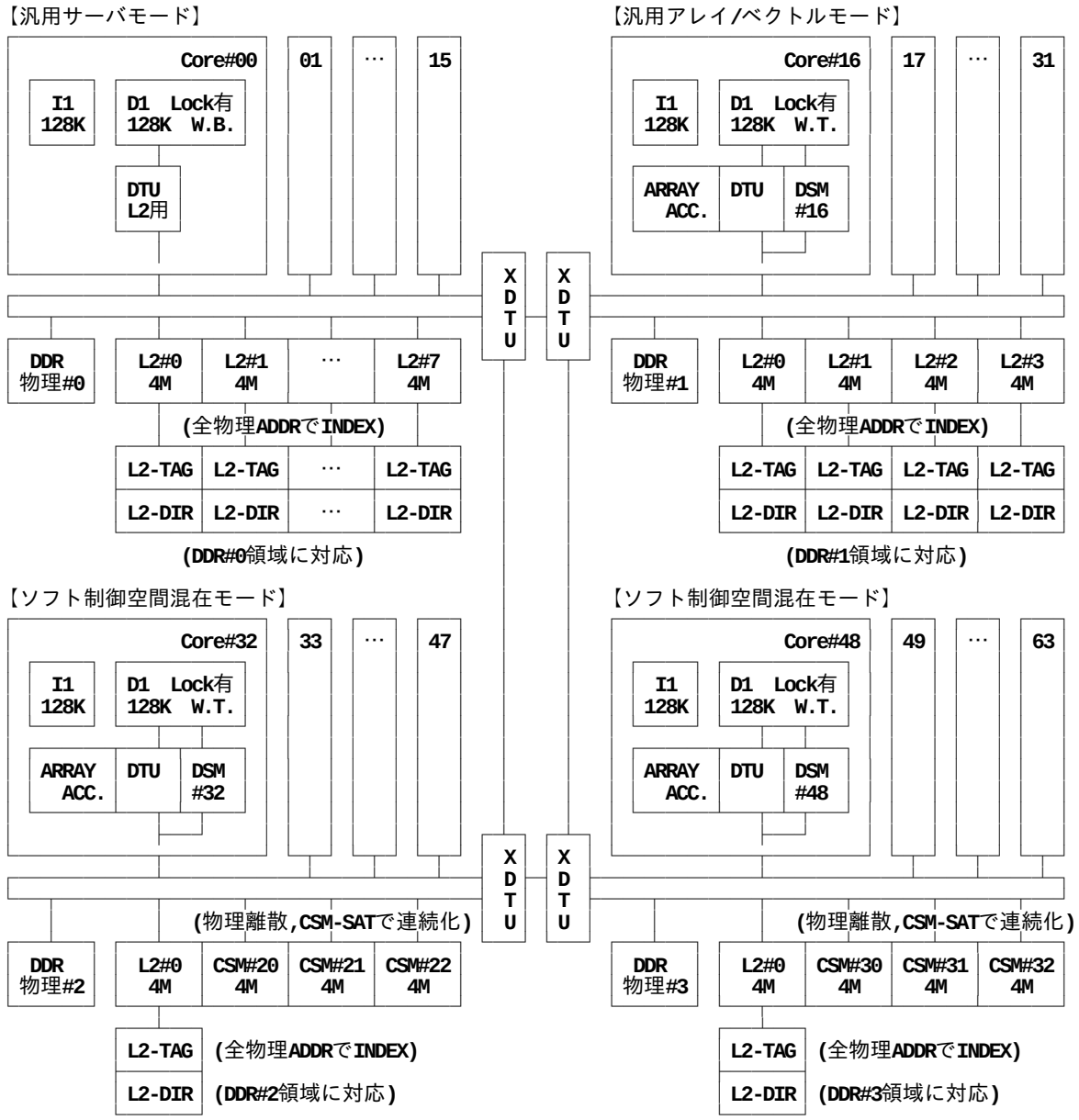


図 3.6: SAPP64 のキャッシュ構成

時無効，L1-hit が注目コアの L1 が有効かつ L2 以下の記憶階層との矛盾がない状態，L1-dirty が注目コアの L1 が有効かつ最新状態が L1 にしかない状態（L2 以下の記憶階層に書き戻されていない状態）を示している.「注目コア」列は注目コアに属するスカラ，ベクトル，DTU，I/O からの Read/Write アクセス,「他コア」列は注目コア以外のコアに属するスカラ，ベクトル，DTU，I/O からの Read/Write を示している．

Firmware 領域および I/O 制御領域は SAPP64 のリセットや IPL 処理のための特権プログラム領域（全コア共通）であり，スカラおよび I/O のみがノンキャッシャブルで参照できる．

OS 領域およびユーザ DDR 領域は L1 キャッシュが有効であり，注目コアの L1 が有効 (L1-hit) である時に他コアからの Read 要求が発生した場合，注目コア他コアとも L1 の当該ラインは Shared 状態となる（○）．注目コアの L1 が有効 (L1-dirty) である時に他コアからの Read 要求が発生した場合，注目コア L1 の当該ラインは一旦 L2 に書き戻されて Shared，他コア L1 の当該ラインは L2 から読み出した後に Shared（●）となる．他コアからの Write 要求が発生した場合，注目コア L1 の当該ラインは Invalid，他コア L1 の当該ラインは Modified（★）となる．注目コアの L1 が有効 (L1-hit) である時に DTU による書き込みが発生した場合，当該 L1 の内容は無効化される（□）．また，注目コアの L1 が有効 (L1-dirty) である時に DTU による読み出しまたは書き込みが発生した場合，当該 L1 の内容は L2 階層以下に追い出された後

アクセス元		注目コア								他コア							
		スカラ		ベクトル		DTU		I／O		スカラ		ベクトル		DTU		I／O	
アクセス先		R	W	R	W	R	W	R	W	R	W	R	W	R	W	R	W
Firmware領域	N.C.	−	−	×	×	×	×	−	−	同左（同一空間）							
OS領域 (L1/L2有効)	L1-hit L1-dirty	◎ ◎	◎ ◎	× ×	× ×	− ■	□ ■	− −	− −	○ ●	★ ★	× ×	× ×	− ■	□ ■	− −	− −
USER-DDR領域 (L1/L2有効)	L1-hit L1-dirty	◎ ◎	◎ ◎	× ×	× ×	− ■	□ ■	− −	− −	○ ●	★ ★	× ×	× ×	− ■	□ ■	− −	− −
I/O制御領域	N.C.	−	−	×	×	×	×	−	−	同左（同一空間）							
物理DSM領域 (L1のみWT有効)	L1-hit	◎	◎	−	□	−	□	×	×	−	−	×	×	−	□	×	×
DSM-SAT領域	N.C.	−	−	×	×	×	×	×	×	×(自コア内のみ参照可)							
物理CSM領域	N.C.	−	−	×	×	−	□	×	×	同左（同一空間）							
CSM-SAT領域	N.C.	−	−	×	×	×	×	×	×	×(自コア内のみ参照可)							

× 参照不可 …(例外検出または別空間)
− L*-考慮不要 …(スカラはキャッシュオフ，その他はDMA)
□ L*-無効化 …(ハードウェアによる整合保証)
■ L*-排出 …(ハードウェアによる整合保証)
★ L*-移転 …(ハードウェアによる整合保証)->自Invalid+他Modify
● L*-供給 …(ハードウェアによる整合保証)->自Shared +他Shared
○ L*-共存 …(ハードウェアによる整合保証)->自Shared +他Shared
◎ L*-ＨＩＴ …(高速)

図 3.7: 論理メモリ空間とキャッシュの関係

に無効化され，DTU により L2 階層以下に対する読み出しまたは書き込みが行われる（■）．

DSM 領域の L1 キャッシュの扱いは，OS 領域およびユーザ DDR 領域と同様である．ただし，注目コアからのアクセスはライトスルーとなり，ベクトルモードのアクセラレータによる書き込み（注目コアからのみ可能）との整合がハードウェアにより保証（L1 の内容無効化）される．また，他コアからの書き込み時は他コアの L1 はノンキャッシャブル，注目コアの L1 は有効（ライトスルー）であるため，同様にハードウェアによる L1 の内容無効化により整合性が維持される．

DSM-SAT 領域，CSM 領域，CSM-SAT 領域はノンキャッシャブルである．

3.7 DDR/DSM 領域の内部動作モデル

論理メモリ空間上の特定アドレスをコア間通信手段に用いる場合の内部動作モデルは DDR 領域と DSM 領域とで異なる．以後，書き込み側コアをＡ，読み出し側コアをＢ（別のクラスタ）とし，Ｂが所属するクラスタに対応する DDR アドレスを用いてＡとＢが交互に動作する場合を想定する．DDR 領域では L1/L2 キャッシュが共に有効であり，Ａの書き込み先は自身の L1 キャッシュとなる（この時点ではまだ書き込みは行われない）．後述するＢの読み出し後は L1-TAG に Shared が表示されている．この情報を元に，Ａから直下の L2 バンクに対して他コアへの該当キャッシュライン無効化要求が送信される．Ｂが所属するクラスタの L2 バンクでは，該当 L2-TAG に Shared および L1 有効（各 L2-TAG は同一クラスタに所属する全コアの L1-TAG 有効状態を保持）と表示されている[3]．Ｂの該当 L1-TAG にも Shared が表示されている．キャッシュライン無効化要求を受信した L2 バンクでは，該当 L2-TAG の状態に基づき，L2-TAG が Invalid に変更されるとともに，Ｂの該当 L1-TAG も Invalid に変更される（Shared の場合 Dirty ではないため内容は破棄してよい）．この後，Ａの該当 L1 への書き込みが行われ L1-TAG が Modified に変更される．この状態では，Ａ直下の L2 バンクの該当 L2-TAG は Modified である．

[3]この構成を実現するために，必然的に L1 キャッシュは L2 キャッシュに対してインクルーシブでなければならない．

次に，Bが同一アドレスから読み出す場合，BのL1ではキャッシュミスが検出され同一クラスタ内のL2バンクへ参照要求が発行され，さらにL2バンクでもキャッシュミスが検出される．該当する物理アドレスはBが所属するクラスタに括り付けられているため，直下のL2-DIRに問い合わせを行った結果，有効なL2キャッシュラインがA側のL2バンクに存在することが判明する．さらにA側のL2バンクに対して参照要求を発行し，読み出し結果を受信後，B側のL2-TAGはSharedに変更される．なお，A側のL2バンクが参照要求を受信した時，前述のようにL2-TAGはModifiedであるが，有効な内容はAのL1(Modified)に存在する．このため，L1に対して常に問い合わせを行い，L1がModifiedである場合にはL2に対して書き戻しを行わせ，L2をOwned状態へ遷移する必要がある．

さて，Bに所属する実DSMアドレスを用いてAとBが交互に動作する場合を想定する．実DSM領域では各コアに属する領域のみL1キャッシュが有効であり，Aの書き込みは自身のL1キャッシュを経由せず，直接Bの実DSMに対して行われる．この際，Bの該当L1-TAGが検査され，有効である場合は該当L1が無効化され，Aからの書き込みデータがDSMに上書きされる．

次に，Bが同一アドレスから読み出す場合，BのL1ではキャッシュミスが検出され自身の実DSM領域から読み出しが行われて該当L1が有効化される[4]．

3.8 アクセラレータの動作モード

SAPP64の各コアに搭載されるアクセラレータでは，モデルに依存して，アレイモードおよびベクトルモード，または，いずれか1つの動作モードがサポートされる．アクセラレータを使用せず一般的なスカラプロセッサとして機械語命令を順次実行するモードを非アレイモードと呼ぶ．プログラムは非アレイモードにより実行が開始され，アレイモード遷移命令またはベクトルモード遷移命令により各モードへ遷移する．また，各遷移命令を実行後，最初に検出した無条件後方分岐により，非アレイモードへ復帰する．すなわち，プログラム中で両モードを切替えて使用することが可能である．最内ループを形成する同一の機械語命令列に対して，アレイモード遷移命令を前置した場合はアレイ動作による高速実行，また，ベクトルモード遷移命令を前置した場合はベクトル動作による高速実行を試みる．ただし，最内ループを動的に解析した結果，アレイ動作またはベクトル動作が困難であることが判明した場合には，当該モード遷移命令は無視され，非アレイモードのまま命令実行が続行される．なお，モード遷移が可能であるための条件は，各モードにより異なる[5]．

3.9 アレイモードアクセラレータのDDR/DSM参照

アレイモードにあるアクセラレータはL1キャッシュの一部を本来のL1キャッシュとして使用し，残りをR/Oの局所メモリとして使用する．前者の動作はアドレスによって異なっており，DDR領域に対してはライトバック，DSM領域に対してはライトスルーとなる（CSM領域はL1キャッシュおよびアレイモードアクセラレータの対象外）．図3.8の第1列および第2列は2つを比較したものである．アレイモードアクセラレータを起動する際に使用するDCPL命令のうちFLUSH機能は，ライトバックの場合にのみ有効であり，ライトスルーの場合には無視される．すなわち，DSM領域を使用することによりオーバヘッドを減らすことができる．ただし，L1キャッシュとDSMに対する書き込み速度に差がある場合，ライトスルー実現のためにはDSMとの間にFIFOを設ける必要があり，2つの性能差は縮まる．

[4]前述のようなワード単位のDSM領域制御プロトコルが実現可能であれば，DDR領域における一般的な分散共有メモリプロトコルに比べてトラフィックおよびレイテンシを削減することができる．ただし，DDR領域でのキャッシュライン単位の送受信プロトコルとは別に，ワード単位の送受信プロトコルが必要となり，組み合わせの増加によりハードウェア設計が複雑化する．もし，ワード単位にAの状態をBに送信することがDSMの主要な用途であるならば，バリア同期機構として別空間を専用に設け，DSM領域のワード単位転送機構を省略，すなわち，DDR領域と同じ制御によりインプリメントすることにより，トラフィックとレイテンシは犠牲になるがハード設計を楽にする方向も考えられる．

[5]ハードウェアが自動的に最適なモードを選択する方法も考えられる．現状では，コンパイラがいずれかのモード遷移命令を選択して前置しなければならない．

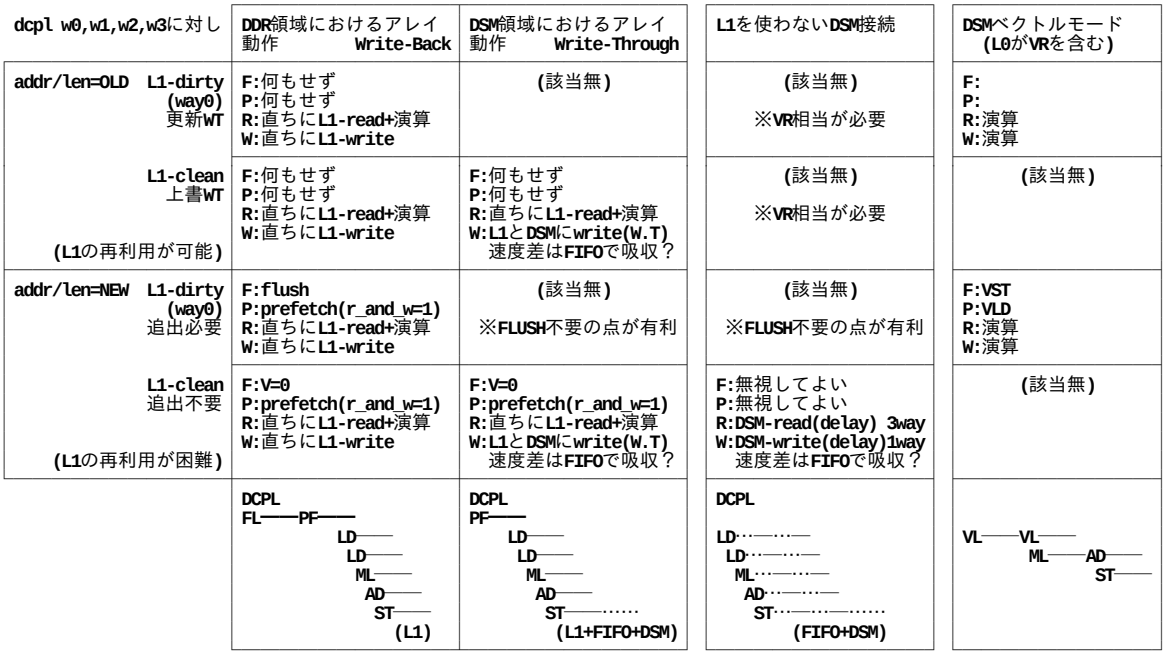


図 3.8: アレイモードアクセラレータの DDR/DSM 参照

3.10 アレイモードアクセラレータと DTU の連結

アレイモードアクセラレータは単独で L1（ライトバック）と DDR 領域のデータ転送（ストライドアクセスを含む）を伴う演算を担当できる（10.4 節「複数 PE 連携機能」に対応）．各アレイ動作全体の実行時間は 1.，2.，3. の合計により律速される．

1. DCPL 命令発行 (L1 ⇒ DDR-X へ FLUSH) … 低速
2. DCPL 命令続き (DDR-A 領域⇒ L1(LDM)) … 低速
3. Array 実行 (L1(LDM) ⇒ L1) … 高速
4. DCPL 命令発行 (L1 ⇒ DDR-A へ FLUSH) … 低速
5. DCPL 命令続き (DSM-B 領域⇒ L1(LDM)) … 低速
6. Array 実行 (L1(LDM) ⇒ L1) … 高速

また，アレイモードアクセラレータが L1（ライトスルー）と DSM の間のデータ転送（ストライドアクセスも可能であるが推奨しない）を伴う演算を担当し，DTU に L2/CSM と DSM の間のデータ転送（多様なストライドアクセスが可能）を分離できる（10.5 節「明示的並列プログラムの実行」に対応）．ソフトウェアは，次の手順によりデータ転送および演算を関連づけてオーバラップ動作させることができる．各アレイ動作全体の実行時間は 1. と 4. の合計により律速され，演算とデータ転送をオーバラップ実行する分だけ，DDR 領域を対象とする前述の手順よりもスループットは高い．

1. DTU-READ 命令発行 (DDR/CSM-B 領域⇒ DSM-B) … 低速，ただし 2.–3. とオーバラップ動作
2. DCPL 命令発行 (DSM-A ⇒ L1(LDM)) … 高速 … 以前の CSM-A ⇒ DSM-A が終っているはず
3. Array 実行 (W.T. ベースの L1(LDM) ⇒ L1&DSM-A) … 高速
4. DTU-WRITE 命令発行 (DSM-A ⇒ DDR/CSM-A 領域) … 低速
5. DTU-READ 命令発行 (DDR/CSM-C 領域⇒ DSM-C) … 低速，ただし 6.–7. とオーバラップ動作
6. DCPL 命令発行 (DSM-B ⇒ L1(LDM)) … 高速 … 1. が終っているはず
7. Array 実行 (W.T. ベースの L1(LDM) ⇒ L1&DSM-B) … 高速
8. DTU-WRITE 命令発行 (DSM-B ⇒ DDR/CSM-B 領域) … 低速

関連制御レジスタ	保護	値の意味
DTU.CONTRL	非特	0:転送開始 1:停止指示
DTU.BLKTOP	非特	DSM内相対制御ブロック先頭アドレス
DTU.CBLKAD	非特	処理中制御ブロックアドレス（表示）
DTU.CBLKST	非特	処理中制御ブロック状況（表示）

→

DTB.NEXTAD	次のDSM内相対制御ブロック先頭アドレス
DTB.TRIGTY	当該ブロック起動トリガ 0:無し 1:有り
:	当該制御ブロックの送受信指示

→

DTB.NEXTAD	00000000 （NULLにより終端）
DTB.TRIGTY	当該ブロック起動トリガ 0:無し 1:有り
:	

図 3.9: DTU 制御レジスタと制御ブロック

3.11 コア間転送機構（DTU）

コア間転送機構（DTU）は，DDR，DSM，CSM 間のデータ転送を行うことができる．転送の際に使用する制御レジスタおよび制御ブロックの内容を図 3.9 に示す．DTU には，送信および受信の際に各々使用するアドレシング機能として，直接，間接，ストライド，２重ストライドが装備されている．

論理アドレスの解釈は，DTU を起動する実コアのアドレス変換機構に従う．複数の制御ブロックが連結されている場合でも，各々に記載される論理アドレスの解釈は DTU 起動時の DDR-SAT，DSM-SAT，CSM-SAT の内容に従うことに注意が必要である．なお，アドレス変換の保持は，DTU 起動時の PEV.DDRSAT，PEV.DDRPID，PEV.DSMSAT，PEV.DSMSPC，PEV.DSMPSZ，PEV.CSMSAT，PEV.CSMSPC，PEV.CSMPSZ の内部保持により行われるため，DTU の動作が終結するまでの間，DDRPID，DSMSPC，CSMSPC に対応する各アドレス変換対を変更してはならない．DDR については DDR 領域内アドレスから得られる物理 DDR アドレスを使用する．物理 DDR アドレス空間は論理的に透過な L2 キャッシュにより高速化されており，DDR の内容と L2 キャッシュの内容はハードウェアにより整合性が維持される（L1 キャッシュの内容も同様）．DSM については実 DSM 予約空間内アドレスから得られる物理 DSM アドレスを使用する．物理 DSM は各物理コアに括り付けられており，L2 キャッシュは経由しないが各コアの L1 キャッシュは有効である．各コア毎の DSM の内容と L1 キャッシュの内容はハードウェアにより整合性が維持される．

3.12 DTU の内部動作モデル

DTU は各コアに装備されており，3.11 節に述べた DDR，DSM，CSM 間のデータ転送は，各メモリを直接参照するのではなく，3.7 節に述べた各コアからの DDR，DSM，CSM 参照と同じ手順に従う．すなわち，キャッシュ階層を含む論理的に正しい Read/Write 動作がハードウェアにより保証される．このため，DTU 転送の効率的利用のためには，キャッシュ整合性維持に関わる内部動作の理解が必要である．

3.13 バリア同期機構（BAR）

プログラム実行開始後，あらかじめ，コアグループに属する 1 つまたは複数の実コアを指定することによりサブコアグループを定義した上で，プログラム実行中の任意の時点で同一プロセス ID に属するサブコアグループ全体の同期を確保する機構である．同期状態は 0 同期状態と 1 同期状態からなり，各実コアが現

関連制御レジスタ	保護	値の意味
PEV.CORMAP	特権	**0:**コア番号変換無効, **1:**コア番号変換有効
PEV.CORGRP[0..63]	特権	実コア番号に対応させる物理コア番号 **0-63:**実コア有効 **255:**実コア無効
BAR.CORSPC	非特	バリア空間 **0:space#0, 1:space#1**
BAR.CORMSK	非特	実コア番号バリアマスク（**64bit**） **0:**同期対象外，**1:**同期対象
BAR.CONTRL	非特	同期要求放送 **0:**０同期要求 **1:**１同期要求
BAR.STATUS	非特	同期状態表示 **00:**非同期状態 **01:**未定義 **10:**０同期状態 **11:**１同期状態

図 3.10: BAR 制御レジスタ

在の同期状態を認識し，他の全コアに対して 0 同期または 1 同期のいずれかを指定して次の同期点到達を放送することにより全体が同期する．使用する制御レジスタの内容を図 3.10 に示す．BAR.CORSPC および BAR.CORMSK によりバリア空間およびバリアマスク（サブコアグループ）を定義した上で BAR.CONTRL により同期要求を発行する．同期要求の送信先がサブコアグループのみか全コアとなるかはモデル依存である．BAR.STATUS には，BAR.CORSPC および BAR.CORMSK に基づく同期状態が表示される[6] ．

3.14 電力制御機構

DDR 単位，L2 バンク単位，実コア単位，および，実コアの各機能ユニット単位（浮動小数点演算機構，実 DSM の各電源制御単位，各キャッシュ階層の各電源制御単位など）に電力制御機構が設けられる．制御可能な動作状態には，電源遮断状態，記憶保持状態，低消費電力状態（複数段階），通常状態がある．各状態間の遷移は，自コアまたは他コアが発行する命令の実行による（自コアの電源遮断状態からの復帰は他コアからのみ制御可能）．また，コヒーレントキャッシュ制御機構も電力制御の対象となる．

3.15 段階的評価プラットホーム

以上に述べた仕様の有効性を検証するためには，各レベルのシミュレータを開発することが有効である．図 3.11 に，検証レベルとシミュレータの関係を示す．

[6] バリア同期機構は，後述する監視条件の指定の有無に関連する．DSM 機構を用いた監視条件の指定により BAR 機構と同様の目的を達成できるのであれば，BAR 機構は不要である．あるいは，前脚注のように BAR 機構のみで十分に目的を達成できる場合には，DSM 機構を用いた監視機構は不要である．両者に各々得失があり，両機能が必要である場合には，10.5 節のように両機能を併存させる仕様となる．一般に BAR 機構は監視条件を用いる場合に比べ，複数コアを一度に待ち合わせるオーバヘッドが小さいと考えられる．

	Inst.Level-Siml. 目的：機能検証	Inst.Level-Siml. +cycle数計算 目的：粗い性能見積	RTL-Level-Siml (Verilog化レベル) 目的：詳細性能見積 ミニアプリ性能検証	Parallel-Siml. (高精度・高速) 目的：実機性能見積 実アプリ性能検証
1コア構成（アドレス変換機構なし）	第1段階		第3段階	
CC・ASM・DISASM・LD（Scalar命令） 論理メモリ空間DDR 疑似I/O(Host-Emulation) 一般命令 テストプロ	GNU既存 NAIST NAIST NAIST 後藤さんに提供	−	→ → → → →ここのTMPに使用	（研究実績有）
1クラスタ構成（モデル上は64コアも可能・コア番号変換機構・アドレス変換機構なし）	第2段階 (H22年度末目標？)		第4段階 L2-DIR詳細性能評価	
コアグループ L2キャッシュ(L2-DIR) 並列/単一コア実行制御情報 DSM CSM DTU BAR テストプロ	NAIST HCU NAIST NAIST NAIST ??? ??? 後藤さんに提供	−	→ → → → → → → →ここのTMPに使用	
4クラスタ構成（64コア忠実版・コア番号変換機構・アドレス変換機構なし			第5段階 (H23年度中)	第6段階 (H23年度末に必要)
L2-DIRECTORY 電力制御機構	−	−	HCU ???	→ →
実機構成（コア番号変換機構・アドレス変換機構あり）				
特権命令 空間保護機構 コア番号変換機構 DDRアドレス変換 DSM-SAT CSM-SAT I/O Firmware OS	製品化レベル （高速版）	−	−	製品化レベル （当面は対象外）

図 3.11: 検証レベルとシミュレータの関係

Chapter 4

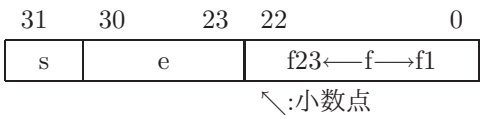
構成

4.1 汎用レジスタ

32bit 汎用レジスタ（GR0–31），4bit 条件コードレジスタ 2 本（ICC0，ICC1）を備えている．

4.2 メディア/浮動小数点レジスタ

32bit メディア/浮動小数点レジスタ（MR/FR0–31）を備えている．単精度浮動小数点数の形式（IEEE754 規格に準拠）は以下の通りである．ただし，単精度浮動小数点演算機能は，後述する C-RTL モデルにのみ実装され，FPGA モデルおよび ASIC モデルには実装されない．



符号	…s	0:正数、	1:負数
バイアス指数	…e	最小指数値:	0
		バイアス指数:	1–254
		最大指数値:	255
		バイアス値:	127
最上位整数	…L	e=0:	L=0
		0 ＜ e ＜ 255:	L=1
		e=255:	L=0
仮数	…f		

不正規化数	
s	0:正数, 1:負数
e	e=0(最小指数値)
L	(0)
f	0 以外
表現式	$(-1)^s * 2^{-126} * (0.f)$

符号付無限大	
s	0:正数, 1:負数
e	e=255(最大指数値)
L	(0)
f	0
表現式	$(-1)^s * \infty$

符号付ゼロ	
s	0:正数, 1:負数
e	e=0(最小指数値)
L	(0)
f	0
表現式	$(-1)^s * 0$

正規化数	
s	0:正数, 1:負数
e	0 ＜ e ＜ 255
L	(1)
f	0 または 0 以外
指数範囲	$2^{-126} - -2^{+127}$
仮数範囲	$1 - -(2 - 2^{-23})$
表現式	$(-1)^s * 2^{e-127} * (1.f)$

非数値	
s	無意味
e	e=255(最大指数値)
L	(0)
f	0 以外
割出し型	f=0XXX…XXX
非割出し型	f=1XXX…XXX

Chapter 5

記憶

5.1 論理メモリ空間

5.2 DDR アドレス変換機構（DDR-SAT）

5.3 DSM アドレス変換機構（DSM-SAT）

5.4 CSM アドレス変換機構（CSM-SAT）

5.5 キャッシュ

Chapter 6

制御

6.1 CPU の動作状態

6.2 制御レジスタの割り付け

各コアの制御レジスタは，物理メモリ空間および論理メモリ空間に共通のアドレスに配置される．制御レジスタの実体はコア毎に独立であり，他コアの制御レジスタを直接参照することはできない．

6.3 リセット

アドレス（下位）	制御レジスタ名	保護	値の意味
0000	CCT.CORMSK	非特	実コア番号制御マスク（64bit） 0:制御対象外，1:制御対象
0008	CCT.RQTYPE	非特	0:起動要求 1:停止要求
0010	CCT.SPINIT	非特	%sp初期値（DSM内相対共通アドレス）
0018	CCT.SPLIMT	非特	%sp下限値（DSM内相対共通アドレス）
0020	CCT.PSTART	非特	PC初期値（DDR3内共通開始アドレス）
0028	CCT.TRIGAD	非特	sleep解除トリガ（自DSM内相対アドレス）
0030	CCT.TRIMSK	非特	sleep解除トリガ（マスク値）
0038	CCT.TRIGDT	非特	sleep解除トリガ（期待値または任意値）
0100	DTU.CONTRL	非特	0:転送開始 1:停止指示
0108	DTU.BLKTOP	非特	DSM内相対制御ブロック先頭アドレス
0110	DTU.CBLKAD	非特	処理中制御ブロックアドレス（表示）
0118	DTU.CBLKST	非特	処理中制御ブロック状況（表示）
0200	BAR.CORSPC	非特	バリア空間 0:space#0, 1:space#1
0208	BAR.CORMSK	非特	実コア番号バリアマスク（64bit） 0:同期対象外，1:同期対象
0210	BAR.CONTRL	非特	同期要求放送 0:０同期要求 1:１同期要求
0218	BAR.STATUS	非特	同期状態表示 00:非同期状態 01:未定義 10:０同期状態 11:１同期状態
0300	PEV.CORMAP	特権	0:コア番号変換無効，1:コア番号変換有効
0308	PEV.DDRSAT .DDRPID	特権	0:DDR-SAT無効，1:DDR-SAT有効 DDR-SAT連想検索に使用するプロセスID
0310	PEV.DSMSAT .DSMSPC .DSMPSZ	特権	0:DSM-SAT無効，1:DSM-SAT有効 DSM-SATに使用するDSM-SAT制御レジスタ群番号 0:space#0, 1:space#1 DSM-SATのページサイズ 0:4KB, 1:16KB, 2:64KB, 3:256KB, 4:1MB
0318	PEV.CSMSAT .CSMSPC .CSMPSZ	特権	0:CSM-SAT無効，1:CSM-SAT有効 CSM-SATに使用するCSM-SAT制御レジスタ群番号 0:space#0, 1:space#1 CSM-SATのページサイズ 0:4KB, 1:16KB, 2:64KB, 3:256KB, 4:1MB
1000 ： 1FF0	PEV.TCAM00 ： PEV.TCAMFF	特権	V,WP,C,PID,論理ADDR,TC,物理DDR-ADDR,MASK
2000 ： 21F8	PEV.CORGRP00 ： PEV.CORGRP63	特権	実コア番号に対応させる物理コア番号 0-63:実コア有効 255:実コア無効

図 6.1: 制御レジスタ一覧

Chapter 7

割り込み

Chapter 8

一般命令

8.1 FRV 命令形式

FRV 命令形式のうち，SAPP64 において使用する命令を列挙する．

8.2 SPARC 命令形式

SPARC-V9 命令形式のうち，SAPP64 において使用する命令を列挙する．

8.2.1 命令グループ０

illtrap，bpcc，bicc，bpr，sethi，fbp，bfcc

8.2.2 命令グループ１

call

8.2.3 命令グループ２

add，and，or，xor，sub，andn，orn，xnor，addc，mulx，umul，smul，subc，udivx，udiv，sdiv，addcc，andcc，orcc，xorcc，subcc，andncc，orncc，xnorcc，addccc，umulcc，smulcc，subccc，udivcc，sdivcc，sll，srl，sra，rdy，movcc，sdivx，popc，movr，wry，float，fcmp，vis，jmpl，return，ticc，flush，save，restore

8.2.4 命令グループ３

lduw，ldub，lduh，ldd，stw，stb，sth，std，ldsw，ldsb，ldsh，ldx，ldstub，stx，lduwa，ldf，lddf，stf，stdf，cas，casx

8.2.5 その他の命令

fmadd, fnmadd, fmsub, fnmsub

表 8.1: 一般命令

	31	30 29 28 27 26 25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10	9 8 7 6	5 4 3 2 1 0	備考
fbse		Rk	opcd	Ri	icc	sop	Rj	
		ADD Ri+Rj->Rk	0 0 0 0 0 0 0		- -	0 0 0 0		★アレイ時自己更新可
		ADDcc Ri+Rj->Rk,cc	0 0 0 0 0 0 0		0 0/1	0 0 0 1		★アレイ時自己更新可
		ADDX Ri+Rj->Rk	0 0 0 0 0 0 0		0 0/1	0 0 1 0		
		ADDXcc Ri+Rj->Rk,cc	0 0 0 0 0 0 0		0 0/1	0 0 1 1		
		SUB Ri-Rj->Rk	0 0 0 0 0 0 0		- -	0 1 0 0		★アレイ時自己更新可
		SUBcc Ri-Rj->Rk,cc	0 0 0 0 0 0 0		0 0/1	0 1 0 1		★アレイ時自己更新可
		SUBX Ri-Rj->Rk	0 0 0 0 0 0 0		0 0/1	0 1 1 0		
		SUBXcc Ri-Rj->Rk,cc	0 0 0 0 0 0 0		0 0/1	0 1 1 1		
		AND Ri&Rj->Rk	0 0 0 0 0 0 1		- -	0 0 0 0		
		ANDcc Ri&Rj->Rk	0 0 0 0 0 0 1		0 0/1	0 0 0 1		
		OR Ri|Rj->Rk	0 0 0 0 0 0 1		- -	0 0 1 0		
		ORcc Ri|Rj->Rk	0 0 0 0 0 0 1		0 0/1	0 0 1 1		
		XOR Ri^Rj->Rk	0 0 0 0 0 0 1		- -	0 1 0 0		
		XORcc Ri^Rj->Rk	0 0 0 0 0 0 1		0 0/1	0 1 0 1		
		NOT ~Rj->Rk	0 0 0 0 0 0 1		- -	0 1 1 0		
		SLL Ri<Rj->Rk	0 0 0 0 0 0 1		- -	1 0 0 0		
		SLLcc Ri<Rj->Rk	0 0 0 0 0 0 1		0 0/1	1 0 0 1		
		SRL Ri>Rj->Rk	0 0 0 0 0 0 1		- -	1 0 1 0		
		SRLcc Ri>Rj->Rk	0 0 0 0 0 0 1		0 0/1	1 0 1 1		
		SRA Ri>Rj->Rk	0 0 0 0 0 0 1		- -	1 1 0 0		
		SRAcc Ri>Rj->Rk	0 0 0 0 0 0 1		0 0/1	1 1 0 1		
fimm10		Rk	opcd	Ri	icc	#s10		
		ADDIcc Ri+#s10->Rk,cc	0 0 1 0 0 0 1		0 0/1			
		ADDXI Ri+#s10->Rk	0 0 1 0 0 1 0		0 0/1			
		ADDXIcc Ri+#s10->Rk,cc	0 0 1 0 0 1 1		0 0/1			
		SUBIcc Ri-#s10->Rk,cc	0 0 1 0 1 0 1		0 0/1			
		SUBXI Ri-#s10->Rk	0 0 1 0 1 1 0		0 0/1			
		SUBXIcc Ri-#s10->Rk,cc	0 0 1 0 1 1 1		0 0/1			
		ANDIcc Ri&#s10->Rk,cc	0 1 0 0 0 0 1		0 0/1			
		ORIcc Ri|#s10->Rk,cc	0 1 0 0 0 1 1		0 0/1			
		XORIcc Ri^#s10->Rk,cc	0 1 0 0 1 0 1		0 0/1			
		SLLIcc Ri<#s10->Rk,cc	0 1 0 1 0 0 1		0 0/1			
		SRLIcc Ri>#s10->Rk,cc	0 1 0 1 0 1 1		0 0/1			
		SRAIcc Ri>#s10->Rk,cc	0 1 0 1 1 0 1		0 0/1			
fimm12		Rk	opcd	Ri	#s12			
		ADDI Ri+#s12->Rk	0 0 1 0 0 0 0					
		SUBI Ri-#s12->Rk	0 0 1 0 1 0 0					
		ANDI Ri&#s12->Rk	0 1 0 0 0 0 0					
		ORI Ri|#s12->Rk	0 1 0 0 0 1 0					
		XORI Ri^#s12->Rk	0 1 0 0 1 0 0					
		SLLI Ri<#s12->Rk	0 1 0 1 0 0 0					
		SRLI Ri>#s12->Rk	0 1 0 1 0 1 0					
		SRAI Ri>#s12->Rk	0 1 0 1 1 0 0					
		LDSBI [Ri+#s12]->Rk	0 1 1 0 0 0 0					
		LDSHI [Ri+#s12]->Rk	0 1 1 0 0 0 1					
		LDI [Ri+#s12]->Rk	0 1 1 0 0 1 0					
		LDUBI [Ri+#s12]->Rk	0 1 1 0 1 0 1					
		LDUHI [Ri+#s12]->Rk	0 1 1 0 1 1 0					
		LDBFI [Ri+#s12]->Rk	0 1 1 1 0 0 0					
		LDFI [Ri+#s12]->Rk	0 1 1 1 0 1 0					
		STBI Rk->[Ri+#s12]	1 0 1 0 0 0 0					
		STHI Rk->[Ri+#s12]	1 0 1 0 0 0 1					
		STI Rk->[Ri+#s12]	1 0 1 0 0 1 0					
		STBFI Rk->[Ri+#s12]	1 0 0 1 1 1 0					
		STFI Rk->[Ri+#s12]	1 0 1 0 1 0 1					
fsop6		Rk	opcd	Ri	sop		Rj	
		LDSB [Ri+Rj]->Rk	0 0 0 0 0 1 0		0 0 0 0 0 0			
		LDUB [Ri+Rj]->Rk	0 0 0 0 0 1 0		0 0 0 0 0 1			
		LDSH [Ri+Rj]->Rk	0 0 0 0 0 1 0		0 0 0 0 1 0			
		LDUH [Ri+Rj]->Rk	0 0 0 0 0 1 0		0 0 0 0 1 1			
		LD [Ri+Rj]->Rk	0 0 0 0 0 1 0		0 0 0 1 0 0			
		LDBF [Ri+Rj]->Rk	0 0 0 0 0 1 0		0 0 1 0 0 0			
		LDF [Ri+Rj]->Rk	0 0 0 0 0 1 0		0 0 1 0 1 0			
		STB Rk->[Ri+Rj]	0 0 0 0 0 1 1		0 0 0 0 0 0			
		STH Rk->[Ri+Rj]	0 0 0 0 0 1 1		0 0 0 0 0 1			
		ST Rk->[Ri+Rj]	0 0 0 0 0 1 1		0 0 0 0 1 0			
		STBF Rk->[Ri+Rj]	0 0 0 0 0 1 1		0 0 1 0 0 0			
		STF Rk->[Ri+Rj]	0 0 0 0 0 1 1		0 0 1 0 1 0			
		MUL Ri*Rj[0]->Rk	1 1 1 1 1 1 1		0 0 0 0 0 0			★独自部分積
		MUL Ri*Rj[1]->Rk	1 1 1 1 1 1 1		0 0 0 0 0 1			★独自部分積
		MUL Ri*Rj[2]->Rk	1 1 1 1 1 1 1		0 0 0 0 1 0			★独自部分積
		MUL Ri*Rj[3]->Rk	1 1 1 1 1 1 1		0 0 0 0 1 1			★独自部分積
		MUL Ri*Rj[0]>>32->	1 1 1 1 1 1 1		0 0 0 1 0 0			★独自部分積
		MUL Ri*Rj[1]>>32->	1 1 1 1 1 1 1		0 0 0 1 0 1			★独自部分積
		MUL Ri*Rj[2]>>32->	1 1 1 1 1 1 1		0 0 0 1 1 0			★独自部分積
		MUL Ri*Rj[3]>>32->	1 1 1 1 1 1 1		0 0 0 1 1 1			★独自部分積
		MIN2 Ri<>Rj->Rk	1 1 1 1 1 1 1		0 1 0 0 0 0			★アレイ時自己更新可
		MAX2 Ri<>Rj->Rk	1 1 1 1 1 1 1		0 1 0 0 0 1			★アレイ時自己更新可
		MAD Ri+Rj->Rk	1 1 1 1 1 1 1		0 1 0 0 1 0			★独自乗算補助
fmov		Rk	opcd	Sr	sop		Rj	
		MOVGS Rj->Rk.Sr	0 0 0 0 0 1 1		0 0 0 1 1 0			★SPR256:ICC0
		MOVSG Rk.Sr->Rj	0 0 0 0 0 1 1		0 0 0 1 1 1			★SPR257:ICC1 ★SPR272:LR
fset		Rk	opcd	--	imm16			
		SETLO #u16,Rk	0 1 1 1 1 0 1					
		SETHI #u16,Rk	0 1 1 1 1 1 0					
		SETLOS #s16,Rk	0 1 1 1 1 1 1					
fbrc		cond / icc	opcd	hint	imm16			
		Bicc icc,label / 0 0/1	0 0 0 0 1 1 0					
fblr		cond / icc	opcd	hint	sop / --	imm12		
		BiccLR icc / 0 0/1	0 0 0 1 1 1 0		0 1 0			
fjmp		-- / li	opcd	Ri	--		Rj	
		JMPL Ri+Rj / 0	0 0 0 1 1 0 0					
		CALLL Ri+Rj / 1	0 0 0 1 1 0 0					
fcall		immhi	opcd	immlo				
		CALL label	0 0 0 1 1 1 1					
ftrp		cond / icc	opcd	Ri	imm12			
		TIicc Ri+imm / 0 0/1	0 0 1 1 1 0 0					(SWI命令)
fcst		Rk	opcd	Ri	cond / ope		Rj	
		CMOVE !cond?Ri:Rj->Rk	1 1 0 0 1 0 1		(ICC0固定) 1 0			★非互換(元はCSWAP)
		CST !cond?Rk->Ri+Rj	1 1 0 0 1 0 0		(ICC0固定) 1 0			★非互換(ICC0のみ)

表 8.2: メディア命令

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	備考
fmedia		Mk						opcd							Mi(s1)						sop						Mj/imm(s2)						
		MNOP0						1	1	1	1	1	1	0							0	0	0	0	0	0							
		MSLL Mi,imm->Mk						1	1	1	1	1	1	0							0	0	0	0	0	1							UL16bitシフト
		drv = ((s1&0xffff0000)<<s2)|((s1<<s2)&0x0000ffff); /* immediate */																														
		MSRL Mi,imm->Mk						1	1	1	1	1	1	0							0	0	0	0	1	0							UL16bitシフト
		drv = ((s1>>s2)&0xffff0000)|((s1&0x0000ffff)>>s2); /* immediate */																														
		MSRA Mi,imm->Mk						1	1	1	1	1	1	0							0	0	0	0	1	1							UL16bitシフト
		drv = (((int)s1>>s2)&0xffff0000)|(((int)(s1<<16)>>(16+s2))&0x0000ffff); /* immediate */																														
		MB2H Mi->Mk						1	1	1	1	1	1	0							0	0	0	1	0	0							
		drv = s2==0?((s1<<8)&0x00ff0000)|(s1&0x000000ff):((s1>>8)&0x00ff0000)|((s1>>16)&0x000000ff);																														
		MH2B Mi->Mk						1	1	1	1	1	1	0							0	0	0	1	0	1							
		drv = s2==0?(((s1>>16)<0x100?(s1>>16)&0xff:255)<< 8)|(((s1&0xffff)<0x100?(s1&0xffff)&0xff:255)) :(((s1>>16)<0x100?(s1>>16)&0xff:255)<<24)|(((s1&0xffff)<0x100?(s1&0xffff)&0xff:255)<<16);																														
		MSUML Mi->MkL						1	1	1	1	1	1	0							0	0	0	1	1	0							U16+L16->L16
		drv = ((s1>>16)+(s1&0xffff))&0xffff;																														
		MSUMH Mi->MkH						1	1	1	1	1	1	0							0	0	0	1	1	1							U16+L16->U16
		drv = (s1&0xffff0000)+(s1<<16);																														
		MH2BD Mi,Mj->Mk						1	1	1	1	1	1	0							0	0	1	0	0	0							
		drv = (((s1>>16)<0x100?(s1>>16)&0xff:255)<<24)|(((s1&0xffff)<0x100?(s1&0xffff)&0xff:255)<<16) |(((s2>>16)<0x100?(s2>>16)&0xff:255)<< 8)|(((s2&0xffff)<0x100?(s2&0xffff)&0xff:255));																														
		MSAD Mi,Mj->Mk						1	1	1	1	1	1	0							0	0	1	0	0	1							SAD(8/8/8/8)->L16
		drv = ((ad(s1&0xff000000,s2&0xff000000)>>8)+ad(s1&0xff0000,s2&0xff0000))|((ad(s1&0xff00,s2&0xff00)>>8)+ad(s1&0xff,s2&0xff));																														
		MSADD Mi,Mj->Mk						1	1	1	1	1	1	0							0	0	1	0	1	0							UL16bit符号付加算
		t1 = ((int) s1 >>16)+((int) s2 >>16); if ((int)t1>(int)0x00007fff) t1 = 0x7fff; if ((int)t1<(int)0xffff8000) t1 = 0x8000;																														
		t2 = ((int)(s1<<16)>>16)+((int)(s2<<16)>>16); if ((int)t2>(int)0x00007fff) t2 = 0x7fff; if ((int)t2<(int)0xffff8000) t2 = 0x8000;																														
		drv = (t1<<16)|(t2&0xffff);																														
		MSSUB Mi,Mj->Mk						1	1	1	1	1	1	0							0	0	1	0	1	1							UL16bit符号無減算
		t1 = ((int) s1 >>16)-((int) s2 >>16); if ((int)t1>(int)0x00007fff) t1 = 0x7fff; if ((int)t1<(int)0xffff8000) t1 = 0x8000;																														
		t2 = ((int)(s1<<16)>>16)-((int)(s2<<16)>>16); if ((int)t2>(int)0x00007fff) t2 = 0x7fff; if ((int)t2<(int)0xffff8000) t2 = 0x8000;																														
		drv = (t1<<16)|(t2&0xffff);																														
		MUADD Mi,Mj->Mk						1	1	1	1	1	1	0							0	0	1	1	0	0							UL16bit符号付加算
		t1 = (s1 >>16)+(s2 >>16); if (t1>0x0000ffff) t1 = 0xffff;																														
		t2 = ((s1<<16)>>16)+((s2<<16)>>16); if (t2>0x0000ffff) t2 = 0xffff;																														
		drv = (t1<<16)|t2;																														
		MUSUB Mi,Mj->Mk						1	1	1	1	1	1	0							0	0	1	1	0	1							UL16bit符号無減算
		t1 = (s1 >>16)-(s2 >>16); if (t1>0x0000ffff) t1 = 0x0000;																														
		t2 = ((s1<<16)>>16)-((s2<<16)>>16); if (t2>0x0000ffff) t2 = 0x0000;																														
		drv = (t1<<16)|t2;																														
		MMUL Mi,Mj->Mk						1	1	1	1	1	1	0							0	0	1	1	1	0							UL8bit*9bit->UL16bit
		t1 = ((s1 >>16)&0x3ff)*(s2&0x1ff); if (t1>0x0000ffff) t1 = 0xffff;																														
		t2 = (((s1<<16)>>16)&0x3ff)*(s2&0x1ff); if (t2>0x0000ffff) t2 = 0xffff;																														
		drv = (t1<<16)|t2;																														
		MOR Mi|Mj->Mk						1	1	1	1	1	1	0							0	1	0	0	0	0							
		MAND Mi&Mj->Mk						1	1	1	1	1	1	0							0	1	0	0	0	1							
		MXOR Mi^Mj->Mk						1	1	1	1	1	1	0							0	1	0	0	1	0							
		BCNT Mi,Mj->Mk						1	1	1	1	1	1	0							0	1	0	0	1	1							Bytewise Count Non0
		drv = ((s1&0xff000000)+(((s2&0xff000000)!=0)<<24))|((s1&0x00ff0000)+(((s2&0x00ff0000)!=0)<<16)) |((s1&0x0000ff00)+(((s2&0x0000ff00)!=0)<< 8))|((s1&0x000000ff)+(((s2&0x000000ff)!=0)));																														
		CSET Mi,Mj->Mk						1	1	1	1	1	1	0							0	1	0	1	0	0							Compare and Set
		drv = (s1<s2)?0:0xff;																														
		PMIN2 Mi,Mj->Mk						1	1	1	1	1	1	0							0	1	1	1	1	0							Bytewise Select MIN
		drv = ((s1&0xff000000)<(s2&0xff000000)?(s1&0xff000000):(s2&0xff000000))|((s1&0x00ff0000)<(s2&0x00ff0000)?(s1&0x00ff0000):(s2&0x00ff0000)) |((s1&0x0000ff00)<(s2&0x0000ff00)?(s1&0x0000ff00):(s2&0x0000ff00))|((s1&0x000000ff)<(s2&0x000000ff)?(s1&0x000000ff):(s2&0x000000ff));																														
		PMAX2 Mi,Mj->Mk						1	1	1	1	1	1	0							0	1	1	1	1	1							Bytewise Select MAX
		drv = ((s1&0xff000000)>(s2&0xff000000)?(s1&0xff000000):(s2&0xff000000))|((s1&0x00ff0000)>(s2&0x00ff0000)?(s1&0x00ff0000):(s2&0x00ff0000)) |((s1&0x0000ff00)>(s2&0x0000ff00)?(s1&0x0000ff00):(s2&0x0000ff00))|((s1&0x000000ff)>(s2&0x000000ff)?(s1&0x000000ff):(s2&0x000000ff));																														
		MMLA Mi,Mj,Mk->Mk						1	1	1	1	1	1	0							1	0	0	0	0	0							MUL+ADD
		t1 = ((s1 >>16)&0x3ff)*(s2&0x1ff); if (t1>0x0000ffff) t1 = 0xffff;																														
		t2 = (((s1<<16)>>16)&0x3ff)*(s2&0x1ff); if (t2>0x0000ffff) t2 = 0xffff;																														
		s2 = (t1<<16)|t2;																														
		t1 = (s2 >>16)+(s3 >>16); if (t1>0x0000ffff) t1 = 0xffff;																														
		t2 = ((s2<<16)>>16)+((s3<<16)>>16); if (t2>0x0000ffff) t2 = 0xffff;																														
		drv = (t1<<16)|t2;																														
		PMIN3 Mi,Mj,Mk->Mk						1	1	1	1	1	1	0							1	0	0	0	0	0							Bytewise Select MIN
		t1 = ((s1&0xff000000)<(s2&0xff000000)?(s1&0xff000000):(s2&0xff000000))|((s1&0x00ff0000)<(s2&0x00ff0000)?(s1&0x00ff0000):(s2&0x00ff0000)) |((s1&0x0000ff00)<(s2&0x0000ff00)?(s1&0x0000ff00):(s2&0x0000ff00))|((s1&0x000000ff)<(s2&0x000000ff)?(s1&0x000000ff):(s2&0x000000ff));																														
		drv = ((t1&0xff000000)<(s3&0xff000000)?(t1&0xff000000):(s3&0xff000000))|((t1&0x00ff0000)<(s3&0x00ff0000)?(t1&0x00ff0000):(s3&0x00ff0000)) |((t1&0x0000ff00)<(s3&0x0000ff00)?(t1&0x0000ff00):(s3&0x0000ff00))|((t1&0x000000ff)<(s3&0x000000ff)?(t1&0x000000ff):(s3&0x000000ff));																														
		PMID3 Mi,Mj,Mk->Mk						1	1	1	1	1	1	0							1	0	0	0	0	0							Bytewise Select MID
		t1 = ((s1&0xff000000)<(s2&0xff000000)?(s1&0xff000000):(s2&0xff000000))|((s1&0x00ff0000)<(s2&0x00ff0000)?(s1&0x00ff0000):(s2&0x00ff0000)) |((s1&0x0000ff00)<(s2&0x0000ff00)?(s1&0x0000ff00):(s2&0x0000ff00))|((s1&0x000000ff)<(s2&0x000000ff)?(s1&0x000000ff):(s2&0x000000ff));																														
		t2 = ((s1&0xff000000)>(s2&0xff000000)?(s1&0xff000000):(s2&0xff000000))|((s1&0x00ff0000)>(s2&0x00ff0000)?(s1&0x00ff0000):(s2&0x00ff0000)) |((s1&0x0000ff00)>(s2&0x0000ff00)?(s1&0x0000ff00):(s2&0x0000ff00))|((s1&0x000000ff)>(s2&0x000000ff)?(s1&0x000000ff):(s2&0x000000ff));																														
		drv = ((s3&0xff000000)<(t1&0xff000000)?(t1&0xff000000):((s3&0xff000000)<(t2&0xff000000)?(s3&0xff000000):(t2&0xff000000))) |((s3&0x00ff0000)<(t1&0x00ff0000)?(t1&0x00ff0000):((s3&0x00ff0000)<(t2&0x00ff0000)?(s3&0x00ff0000):(t2&0x00ff0000))) |((s3&0x0000ff00)<(t1&0x0000ff00)?(t1&0x0000ff00):((s3&0x0000ff00)<(t2&0x0000ff00)?(s3&0x0000ff00):(t2&0x0000ff00))) |((s3&0x000000ff)<(t1&0x000000ff)?(t1&0x000000ff):((s3&0x000000ff)<(t2&0x000000ff)?(s3&0x000000ff):(t2&0x000000ff)));																														
		PMAX3 Mi,Mj,Mk->Mk						1	1	1	1	1	1	0							1	0	0	0	0	0							Bytewise Select MAX
		t1 = ((s1&0xff000000)>(s2&0xff000000)?(s1&0xff000000):(s2&0xff000000))|((s1&0x00ff0000)>(s2&0x00ff0000)?(s1&0x00ff0000):(s2&0x00ff0000)) |((s1&0x0000ff00)>(s2&0x0000ff00)?(s1&0x0000ff00):(s2&0x0000ff00))|((s1&0x000000ff)>(s2&0x000000ff)?(s1&0x000000ff):(s2&0x000000ff));																														
		drv = ((t1&0xff000000)>(s3&0xff000000)?(t1&0xff000000):(s3&0xff000000))|((t1&0x00ff0000)>(s3&0x00ff0000)?(t1&0x00ff0000):(s3&0x00ff0000)) |((t1&0x0000ff00)>(s3&0x0000ff00)?(t1&0x0000ff00):(s3&0x0000ff00))|((t1&0x000000ff)>(s3&0x000000ff)?(t1&0x000000ff):(s3&0x000000ff));																														

表 8.3: 浮動小数点命令

	31	30 29 28 27 26 25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10 9 8 7 6	5 4 3 2 1 0	備考
fmedia		Mk	opcd	Mi	sop	Mj/imm	
		FNOP	1 1 1 1 0 0 1		0 0 0 0 0 0		
		FNEG Mj->Mk	1 1 1 1 0 0 1		0 0 0 0 1 1		
		FABS Mj->Mk	1 1 1 1 0 0 1		0 0 0 1 0 0		
		FADD Mi,Mj->Mk	1 1 1 1 0 0 1		0 0 0 1 1 0		
		FSUB Mi,Mj->Mk	1 1 1 1 0 0 1		0 0 0 1 1 1		
		FMUL Mi,Mj->Mk	1 1 1 1 0 0 1		0 0 1 0 0 0		
		FDIV Mi,Mj->Mk	1 1 1 1 0 0 1		0 0 1 0 0 1		
fcst		Mk	opcd	Mi	cond / ope	Mj	
		CFMOV !cond?Mi:Mj->Mk	1 1 0 1 1 1 0		（ICC0固定） 0 1		★非互換(元はCFDIV)

表 8.4: 複合命令

	31	30 29 28 27 26 25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10 9 8 7 6	5 4 3 2 1 0	備考
複合		Mk	opcd	Mi	sop	Mj/imm	
		FMADD					
		FMSUB					
		FNMADD					
		FNMSUB					

Chapter 9

制御命令

Chapter 10

プログラムの実行

10.1　命令の実行（非アレイ動作）

命令列を通常の方法により実行する動作状態を非アレイ動作と呼び，後述するアレイ動作およびベクトル動作とは区別する．非アレイ動作において SAPP は，前述した一般命令，メディア命令，浮動小数点命令等の全てを実行することができる．

10.2　命令の実行（アレイ動作）

本書の冒頭に述べたように，SAPP の目的は，既存命令を用いてベクトル演算の多重度を超える高い並列度を達成することである．このために，VLIW に合わせた演算器構成を 1 次元方向に拡張した演算器ネットワークを稼働させて並列処理を行う．アレイ動作とは，拡張部分が動作している状態の総称であり，DCPL 命令の検出を契機とする，L1 キャッシュの way0 に対する FLUSH（DSM 領域では NOP），way0 を含む各 way に対する PREFETCH，演算器ネットワーク設定のための命令デコード，L1 キャッシュの連続読み出し動作と演算結果の格納，非アレイ動作への復帰の各フェーズからなる．本節では各フェーズの動作について詳細に説明する．

なお，VLIW が規定されていない SPARC アーキテクチャの場合，アレイ動作が期待される命令区間については，LD 命令が VLIW を形成する先頭命令と解釈する．LD 命令を使用しない VLIW を記述可能とするために，NOP を表現可能な LD 命令が別途定義される．1 つの VLIW 命令にはモデルに依存して最大 8 命令を指定することができる．ただし，命令グループ毎の上限および配置可能な VLIW 中の位置は以下の通りであり，複数グループに属する命令を混在させる場合には，以下の優先順位により先頭から記述しなければならない．また，メディア＋メディア，または，浮動小数点演算＋メディアの組において，第 1 命令では 3 個までの任意のソースレジスタを指定できるものの，第 2 命令は第 1 命令において指定済みのソースレジスタしか指定できない制限がある．

L/S グループから 1 命令 VLIW の先頭位置（第 0 命令位置）にのみ配置可能
　ロード／ストア，SWI，DCPL，ICPL

整数グループから 3 命令 VLIW の第 0 命令位置から第 3 命令位置に配置可能
　整数加減乗算，論理演算，シフト演算，比較演算，選択演算，MOV/SET 演算

メディアグループから 2 組 VLIW の第 0 命令位置から第 7 命令位置に配置可能
　各組はメディア＋メディア，または，浮動小数点演算＋メディアから構成

分岐グループから 1 命令 VLIW の第 0 命令位置から第 7 命令位置に配置可能
　条件分岐命令，無条件分岐命令，CALL 命令，JMP 命令

ところで，VLIW では一般に，同一 VLIW 命令に属する全命令が一斉にソースレジスタの内容を取り出し，結果を一斉に書き込むことを前提に，同一 VLIW 命令内の異なる命令において同一レジスタに対する読み書きを指定することが可能である．しかし，SAPP64 では，そもそも非アレイ動作が VLIW 実行をサポートしている場合を除き，前述した VLIW 解釈に基づいて同一 VLIW に属すると見なされる各命令を

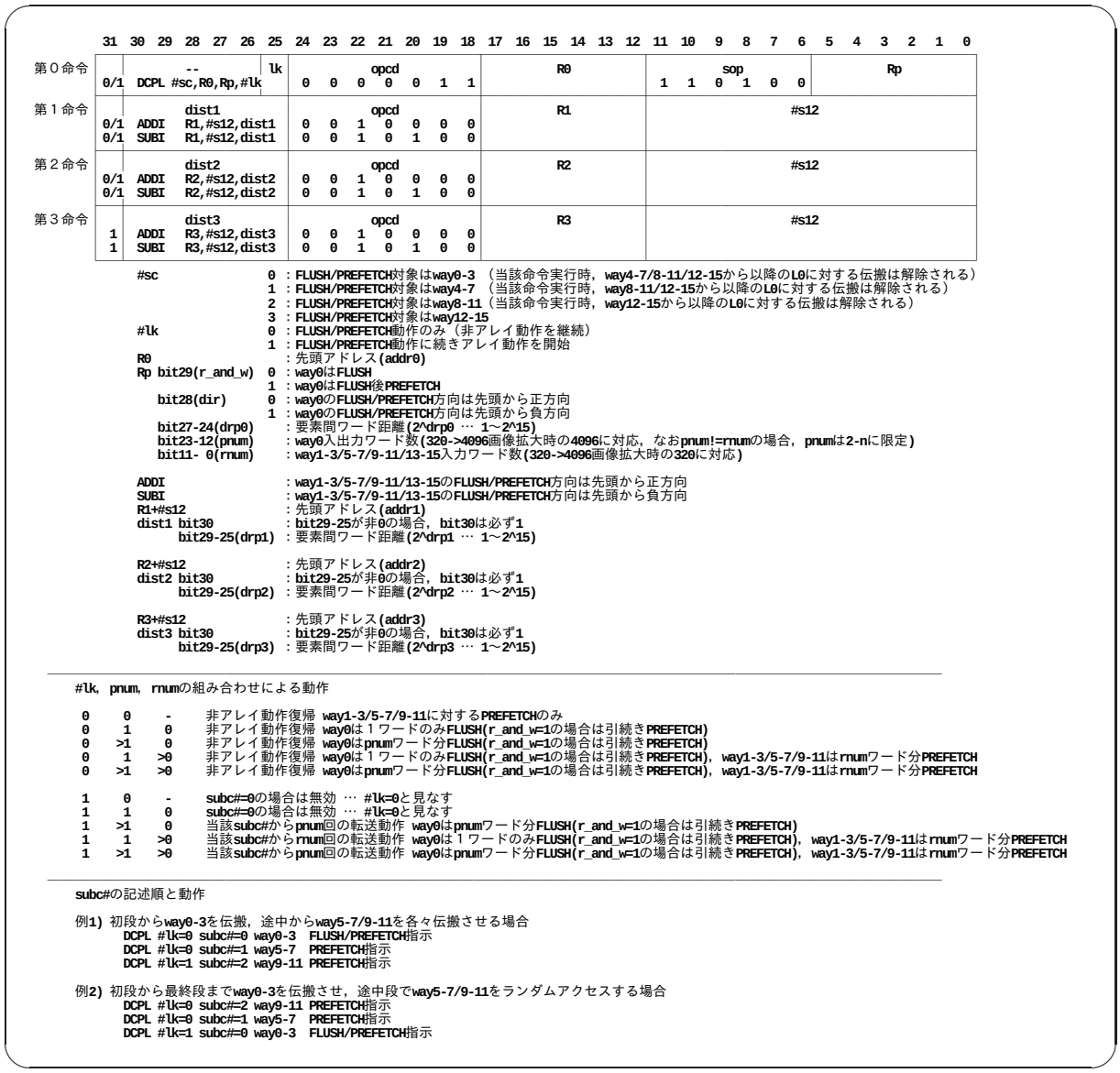


図 10.1: DCPL 命令

記述順に 1 命令ずつ実行しても同一の結果が得られるよう，レジスタの依存関係を考慮して記述しなければならない（SPARC アーキテクチャの場合が該当する）．この制限により，VLIW 解釈およびアレイ動作を前提に記述された命令列をハードウェア資源の不足やデバッグを理由に逐次実行の非アレイ動作により実行した場合でも同じ演算結果を得ることができる[1]．

10.2.1 DCPL 命令の検出

SAPP は，DCPL 命令を含む VLIW 命令の検出を契機として，非アレイ動作からアレイ動作へ遷移する．DCPL 命令を含む VLIW 命令内において，DCPL 命令と同時に指定可能な一般命令は ADD または SUB に制限されており，VLIW 全体としてアレイ動作を指示する．図 10.1 に DCPL 命令を含む VLIW 命令の詳細を示す．なお，DCPL 命令を NOP と見なし非アレイ動作を継続しても正しい結果が得られる．

第 0 命令位置の DCPL 命令には#subc，#lk，R0，Rp の 4 つのオペランドがある．subc#に 0 を指定し

[1]元々VLIW である命令列をアレイ実行するのに対し，VLIW でない命令列をまず VLIW 解釈した後にアレイ実行する方法は，逐次実行時の効率が低下する．例えば (LD[A] ⇒ X,A=A+4); (LD[B] ⇒ Y,B=B+4);(LD[C] ⇒ Z,C=C+4,P=X*J); という命令列（VLIW × 3）は，先頭から 1 命令ずつ逐次実行しても演算結果と命令数は同じであるが，(LD[A] ⇒ X);(LD-nop);(LD-nop,P=X*J); という命令列（VLIW × 3）は，VLIW 解釈の先頭を指示するために LD-nop 命令を追加しており，逐次実行の際に 2 命令の無駄が生じる．LD-nop を省略した場合，逐次実行時の演算結果は同じであるが，VLIW 解釈では X*J に使用される X の値が LD[A] 以前の値となるため演算結果が不正となる．

た場合，第 0 命令は way0，第 1 命令から第 3 命令は way1-3 のいずれかに対応する PREFETCH 情報として使用される．具体的には，各命令は way1-3 に直接対応せず，既に way1-3 のいずれかに PREFETCH 済みの場合，当該命令は無視され，PREFETCH が必要な場合，way1-3 のうち最も古く PREFETCH を行った way が格納先として選択される．subc#に 1，2，3 を指定した場合，第 1 命令から第 3 命令はモデル依存の段（標準では先頭を第 1 段とした場合の第１０段/第１９段/第２８段）に属する way5-7，9-11，13-15 のいずれかに対応する PREFETCH 情報として使用される．具体的には，各命令は way5-7，9-11，13-15 に直接対応せず，既に way5-7，9-11，13-15 のいずれかに PREFETCH 済みの場合，当該命令は無視され，PREFETCH が必要な場合，way5-7，9-11，13-15 のうち最も古く PREFETCH を行った way が格納先として選択される．#lk に 0 が指定されている場合，FLUSH/PREFETCH 動作のみを行い，次命令から直ちに非アレイ動作に復帰する．#lk に 1 が指定されている場合，FLUSH/PREFETCH 動作の完了時に演算器ネットワークを利用した演算動作へ遷移する．R0 には，way0 に対する FLUSH/PREFETCH の先頭アドレス (addr0) を保持するレジスタ番号を指定し，Rp には，FLUSH/PREFETCH 動作の詳細を保持するレジスタ番号を指定する．Rp の内容は 5 個のフィールド（r_and_w，dir，drp0，pnum，rnum）から構成されている．r_and_w に 0 を指定した場合，addr0 から pnum により指定されたワード数の範囲（方向は dir により指定）に対応する way0 の内容が FLUSH される（DSM 領域では NOP）．r_and_w に 1 を指定した場合，FLUSH 動作に続き addr0 から pnum により指定されたワード数が way0 に PREFETCH される．drp0 は way0 において FLUSH/PREFETCH の対象となる要素間ワード距離（2^{drp0}）である．rnum は，way1-3，5-7，9-11，13-15 に対する PREFETCH ワード数である．

第 1 命令から第 3 命令の ADD/SUB 命令には各々R，#s12，dist の 3 つのオペランドがあり，$R+\#s12$ により PREFETCH 先頭アドレス（addr1-3）を指定する．同様に dist により要素間ワード距離を指定する．ADD/SUB 命令の違いは PREFETCH 方向（way0 における dir と同じ）であり，SUB 命令の場合でも先頭アドレスは $R+\#s12$ により計算される．

モデルによっては，各 9 段を有するコアを複数連結した３６段構成が可能である．この場合，各コアに属する L1 キャッシュ（way0 および way1，2，3 からなる構成）が連結され，各コアの way0 に対応する領域を way0，4，8，12 とする合計 16way の L1 キャッシュとなる．前述の#subc に 1，2，3 を指定した場合，第 0 命令位置の DCPL 命令は，way4，8，12 に対する FLUSH/PREFETCH 指示と解釈される．

10.2.2 L1 キャッシュの追い出しおよびプリフェッチ

前述のように，DCPL 命令を含む VLIW 命令により，L1 キャッシュの各 way に対する FLUSH/PREFETCH 動作を指定することができる．ただし，way 毎に，最後に行った FLUSH/PREFETCH 動作に関する先頭アドレスおよびワード数の情報が記録されており，ライン毎の TAG 情報と合わせて，必要最小限の FLUSH/PREFETCH 動作が行われる．具体的には，（１）way0 の PREFETCH 済範囲においてキャッシュミスが発生してラインが置き換わらない限り，次回の同一アドレス範囲に対する PREFETCH 全体を省略し，（２）way1-3 の PREFETCH 済範囲に対し way0 へのストアが発生しない限り，次回の同一アドレス範囲に対する way1-3 への PREFETCH 全体を省略する．PREFETCH 全体を省略できない場合であっても，個々のライン毎の TAG 情報と比較して，PREFETCH が不要なラインに対しては省略して高速化する．また，モデルに依存して，互いに異なる subc#を指定した DCPL 命令は，FLUSH/PREFETCH の同時実行により高速化される．

10.2.3 演算器ネットワーク設定のための命令デコード

VLIW に合わせた演算器構成を 1 次元方向に拡張した演算器ネットワークは，DCPL 命令を含む VLIW 命令の次の命令（ループイタレーションの先頭 VLIW 命令）から，最初に検出した無条件後方分岐命令（最終 VLIW 命令）までの各命令をデコードして得られるレジスタ依存関係を元に設定される．すなわち，先頭 VLIW 命令から順に演算情報が取り出されて各段の演算器が設定されると同時に，新たにデコードされた VLIW 命令の演算情報が対応段の演算器に設定される際には，先行命令の写像結果を参考に必要とするレジスタ値が対応段に伝搬されるよう，対応段に至るまでの演算器ネットワークが動的に構築される．なお，演算器ネットワークが正しく設定されるためには，ループイタレーションに含まれる命令が少なくとも

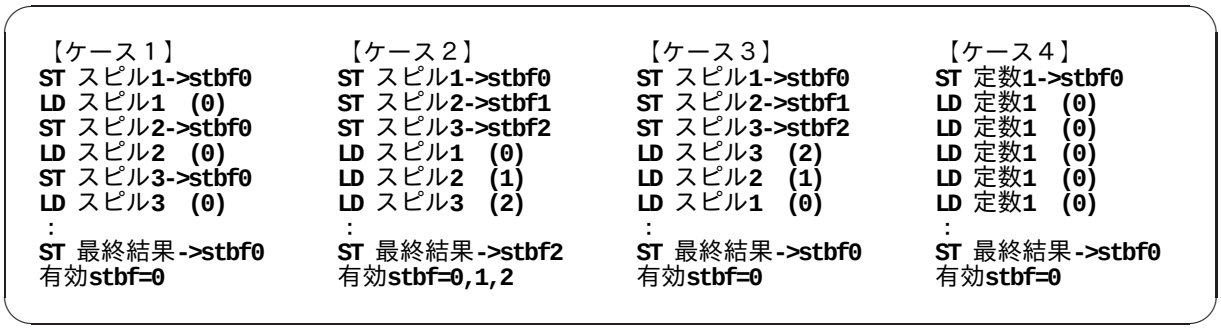


図 10.2: ストア先候補レーン番号の挙動

以下の条件を満たしていなければならない．詳細については Chapter 11 を参照のこと．

- 無条件後方分岐または条件付き前方分岐（ループ脱出用のみ）以外の分岐命令，SWI，DCPL，ICPL を含まないこと
- LD-USE 間は 1VLIW 命令以上の間隔をあけること
- $a = a + b$ のような自己更新型を除きイタレーション間にデータ依存関係がないこと
- 自己更新命令は第 1 ソースとディスティネーションレジスタ番号が同一であること
- 先行 LD/ST 命令のベースレジスタを更新する後続（同一 VLIW 内を含む）自己更新命令は ADDI/SUBI 命令であること
- レジスタ値の伝搬に必要なレジスタ数が演算器ネットワークに装備されるレジスタ数を超えないこと
- VLIW 命令数が演算器ネットワークの段数を超えないこと

新たにデコードされた VLIW 命令の演算情報が初段から対応段まで順に転送される際，各段における演算器ネットワーク設定が順次更新される．具体的には，以下の手順に従う．

- 新たにデコードされた命令のディスティネーションレジスタは，後続命令が使用する同一ソースレジスタの起点となるため，当該命令が各段を通過する際，当該ディスティネーションレジスタは各段において伝搬不要レジスタとして記録される．
- 当該命令が各段を通過する際，伝搬不要レジスタとして記録されていないソースレジスタのうち，直前段の演算器出力から直接供給できないものについて，実体が空き演算器入力レジスタに割り付けられる．
- 同様に，当該命令の演算情報が対応段に設定される際，伝搬不要レジスタとして記録されていないソースレジスタは，実体が対応演算器の入力レジスタに割り付けられる．

10.2.4 L1 キャッシュの連続読み出し動作と演算結果の格納

前述した「L1 キャッシュの追い出しおよびプリフェッチ」と「演算器ネットワーク設定のための命令デコード」は同時動作してよい．ただし一般的に，処理対象となるデータ数のほうが，イタレーション内命令数よりも多いため，通常は前者によって律速される．いずれにせよ両動作が完了した時点で，演算に必要なデータおよび dirty でない結果格納場所が L1 キャッシュに揃い，毎サイクル実行結果を出力するための演算器ネットワークの構築が完了する．

準備が完了次第，L1 キャッシュを構成する way0-3 のうち，DCPL 命令により PREFETCH 対象とされた way の各 PREFETCH 先頭アドレスから，毎サイクル 1 ワードが同時に読み出され，初段から順次，最終段に向かって伝搬していく．ただし，各段において保存される期間は各段が備える L0 キャッシュの容量によって決まるため，一定サイクル経過後に参照可能なアドレス範囲は制限されており，範囲外アドレスに対するロード命令は正しい内容を取得できない．一方，DCPL 命令により way4-7，8-11，12-15 に対する PREFETCH が指定されている場合，当該 way が接続されている段（標準では第１０段/第１９段/第２８段）以降は，way0-3 の内容に代わり，way4-7，8-11，12-15 から読み出された内容が伝搬される．なお，way4-7，8-11，12-15 については，直接接続されたロードストアユニット（標準では第１０段/第１９段/第

２８段）からのランダムロードアクセスが可能である．また，way0-3 からの読み出し回数は，厳密には図 10.1 に示した演算動作回数の制約を受けず，次節に説明する非アレイ動作への復帰条件が満たされた時点で読み出しが停止する．

ストア命令が生成するストアデータは，各段が備えるストアバッファに格納され，毎サイクル次段へ伝搬される．各ストアバッファの容量は 1 ワードであり，前段のストアバッファとワードアドレスが同じストアデータはマージされる．また，ストア→ロード→ストアのように，最終的に L1 キャッシュに書き込むデータを生成するストア命令以前に，一時的に主記憶アドレスを利用してレジスタ間転送を行うことも可能である．ただし，この場合，一時的なストアデータがキャッシュや主記憶へ書き込まれることはない．

前述のように，モデルによっては，各 9 段を有するコアを複数連結した３６段構成が可能である．この場合，way0，4，8，12 が書き込み可能である．さらにストアバッファを 4 レーンに増加させることにより，最大 4 つの配列に対してストアする命令列を写像することができる．way0，1，2，3 には各々レーン番号 0，1，2，3 が対応づけられる．また，初段からストア先候補レーン番号が順次伝搬される．ストア先候補レーン番号の初期値は 0 であり，各段のストア命令がストアデータを格納すべきストアバッファのレーン番号を表示している．ストア命令が写像されている場合，すでにストアバッファに同一アドレスが登録されている場合を除き，次段以降にはストア先候補レーン番号に 1 を加えた値が伝搬される．ただし，後続ロード命令のロードアドレスがストアバッファのいずれかに一致した場合，次段以降には，一致したレーン番号に戻される（スピルアウト/イン後，ストア先候補レーン番号が元に戻ることに対応する）．最終的に way0，4，8，12 に書き込まれるのは，ストア先候補レーン番号から 1 を減じたレーンまでである．図 10.2 に，ストア先候補レーン番号の挙動を示す．ケース 1 では，最終的にストア先候補レーン番号が 1 となり，レーン 0 の内容のみが way0 に書き込まれる．ケース 2 では，最終的にストア先候補レーン番号が 3 となり，不要なレーン 0 および 1 の内容が way0，4 に書き込まれてしまうため，ケース 3 への変形が必要である．ケース 4 のように同一アドレスから複数回ロードしても問題は生じない．

アレイ動作中，way0，4，8，12 のうち複数 way が書き込み先として有効，かつ，ストア先アドレスが重複する場合の挙動は，ストア先アドレスによって異なる．DDR 領域の場合，L1 キャッシュへの書き込み時にはハードウェアによるキャッシュ内容の整合性維持は行われない．すなわち，モデルによっては，DCPL 命令発行時にアドレス重複が検査される．アドレス重複が検査されないモデルの場合，L1 キャッシュの MOESI 状態は予測不能となるため，重複が発生しないようソフトウェアが保証しなければならない．一方，DSM 領域に対してはストアスルーであるため，整合性は維持される（DSM へ書き込む際に，競合する相手方の L1 キャッシュラインが無効化される）．

10.2.5 非アレイ動作への復帰

命令デコード時に，条件付き前方分岐命令はループ脱出命令と見なされ，脱出先アドレスが保存されている．当該前方分岐命令において分岐条件が成立した場合，当該段から最終段に向けて非アレイ動作への復帰指令が伝搬される．最終段においてこの指令を検出した場合，保存されていた脱出先アドレスから非アレイ動作が再開される．なお，複数の条件付き前方分岐命令が存在してよいが，脱出先アドレスは同一でなければならない．同一でない場合，最後にデコードした分岐命令の脱出先アドレスのみが使用される．

10.3 命令の実行（ベクトル動作）

SAPP64 では，ベクトル命令は明示的には規定されない．ただし，ベクトル動作の説明のために，以下のベクトル操作を用いる．ベクトル操作は SPARC-V9 命令形式のうち命令グループ 3 の空き空間を使用し、連続して配置される前置命令 (32 ビット) と後置命令 (32 ビット) を組み合わせた 64 ビットで定義される。なお、以下に定義されるベクトル操作は，VPP5000 ベクトル演算操作のサブセットに新規操作 (16bit/8bit 単位の整数演算等) を追加している。

10.3.1 操作形式

各操作において、

- R1 で読み出し用に vr、mr、gr、fr、vcr、vt を指定するときは
  それぞれ vr1、mr1、gr1、fr1、vcr1、vt1

- R2 で読み出し用に vr、mr、gr、fr を指定するときは
  それぞれ vr2、mr2、gr2、fr2

- R3 で書き込み用に vr、mr、gr、fr、vcr、vt を指定するときは
  それぞれ vr3、mr3、gr3、fr3、vcr3、vt3

- R4 は読み出し用に mr を指定するのみで mr4

- R22 で読み出し用に vr、mr、gr、fr を指定するときは
  それぞれ vr22、mr22、gr22、fr22

- R22 で書き込み用に gr、fr を指定するときは
  それぞれ gr33,fr33

と表記する。
ただし、adclu の mr22 は読み出しと書き込みの両方が行われる。

前置操作の形式

<31:30> '11' 固定
<29:27> 算術演算の符号、アクセス id、比較 id、丸めモードの指定。
特に指定のない操作では <29:27> は'000' に設定し、使用しないこと。
<26:25> op0
op0<26>='0' のとき、
算術演算、比較、およびビット操作. 論理演算の R1 は vr、
op0<26>='1' のとき、
算術演算、比較、およびビット操作. 論理演算の R1 は gr または fr
op0<25> の割り当ては表 10.1 操作コード一覧
<24:23> '01' 固定 ベクトル前置操作の識別
<22:19> op1 割り当ては表 10.1 操作コード一覧
<18:16> op2 割り当ては表 10.1 操作コード一覧
<15:08> R1
vr、mr を指定するときは <15:08>、
gr、fr を指定するときは <13:08> が gr、fr 番号で <15:14> は'00' リザーブ
<07:00> R2
vr、mr を指定するときはは <07:00>、
gr、fr を指定するときは <05:00> が gf、fr 番号で <07:06> は'00' リザーブ

後置操作の形式

<31:30>	’11’	固定
<29:27>		R22<29:27>、比較 cc(op0,op1,op2 により識別、CPU は don’t care)
		特に指定のない操作では <29:27>=’000’ にリザーブされる。使用しないこと。
<26:25>		R22<26:25>
		vmullu のとき、
		<26>=0 で vr3 へ積の下位 64 ビットを格納、
		<26>=1 で vr3 へ積の上位 64 ビットを格納。
<24:23>	’11’	固定 ベクトル後置操作の識別
<22:19>	op3	ベクトル操作では’0000’-’1011’ の範囲で使用
		後置操作の R22,R3 フィールドの gr、fr アクセスを CPU が認識するためのコード。
		=’0000’ R22=gr/fr 以外、R3=gr/fr 以外
		=’0001’ R22=gr/fr 以外、R3=gr-wr(gr への書込み)
		=’0010’ R22=gr/fr 以外、R3=fr-wr(fr への書込み)
		=’0011’ R22=gr-rd(gr からの読み出し)、R3=gr/fr 以外
		=’0100’ R22=fr-rd(fr からの読み出し)、R3=gr/fr 以外
		=’0101’ R22=gr-wr(gr への書込み)、R3=gr-wr(gr への書込み)
		=’0110’ R22=fr-wr(fr への書込み)、R3=gr-wr(gr への書込み)
		=’0111’-’1011’ リザーブ
<18:16>	R22	<18:16>
<15:08>	R3	vr、mr は <15:08>
		gr、fr のときは <13:08> が gr、fr 番号で <15:14> は’00’ リザーブ
<07:00>	R4	mr<07:00> のみ

CPU によるベクトル操作での gr/fr アクセスの認識

1. CPU は操作コードの <31:30>=’11’ かつ <24:23>=’01’ でベクトル前置操作を認識する。

2. CPU は操作コードの <31:30>=’11’かつ<24:23>=’11’ でベクトル後置操作を認識する。

3. ベクトル前置操作では、R1、R2 フィールドで gr または fr の読み出しを指定することがあるが、CPU は前置操作の op0,op1,op2 をデコードして認識する。
   ただし、算術演算操作、比較操作、およびビット操作. 論理演算操作では一つのニーモニックで op0<26>=’0’ のとき R1 フィールドで vr1 を指定、op0<26>=’1’ のとき gr1 または fr1 を指定している。

4. ベクトル後置操作では、R22 フィールドで gr または fr の読み出しと書き込み、および、R3 フィールドで gr または fr への書き込みを指定することがあるが、CPU は後置操作の op3 をデコードして認識する。

10.3.2 ベクトル制御操作

```
vplcr   gr1,vcr3
        op0=’00’ op1=’0100’ op2=’111’ op3=’0000’
vpscr   vcr1,gr3
        op0=’00’ op1=’1000’ op2=’111’ op3=’0001’
vlvt    gr1,VT3
        op0=’01’ op1=’0100’ op2=’111’ op3=’0000’
vsvt    VT1,gr3
        op0=’01’ op1=’1000’ op2=’111’ op3=’0001’
vltr    gr1
```

```
        op0='00' op1='0001' op2='111' op3='0000'
vlcr    gr1,vcr3
        op0='00' op1='0110' op2='111' op3='0000'
vscr    vcr1,gr3
        op0='00' op1='1010' op2='111' op3='0001'
vlvl    gr1
        op0='00' op1='0101' op2='111' op3='0000'
vlvlf   gr1
        op0='01' op1='0101' op2='111' op3='0000'
vpt
        op0='01' op1='0111' op2='111' op3='0000'
```

10.3.3 ベクトルアクセス操作

ベクトルロード

ロード操作ではニーモニックの"(id)"に 3 ビットのアクセス id を指定して、同一のアクセス id を持つアクセス操作 (ロード/ストア) 間でプログラムに記述された実行順序が保証されるように実行させることができる。アクセス id は前置 <29:27> に設定させる。だだし、インダイレクトアクセス操作は先行アクセス操作との間で、常にプログラムに記述された実行順序が保証されるように実行されるため、アクセス id は指定しない。前置 <29:27> は'000' リザーブとする。

```
vldi(id)   gr1,gr2,vr3,mr4
           op0='00' op1='0001' op2='000' op3='0000'
vldiu(id)  gr1,gr2,vr3,mr4
           op0='00' op1='0001' op2='001' op3='0000'
vldf(id)   gr1,gr2,vr3,mr4
           op0='00' op1='0001' op2='010' op3='0000'
vldd(id)   gr1,gr2,vr3,mr4
           op0='00' op1='0001' op2='011' op3='0000'
vldm(id)   gr1,gr2,mr3
           op0='00' op1='0001' op2='110' op3='0000'
vildi      gr1,vr2,vr3,mr4
           op0='00' op1='0011' op2='000' op3='0000'
vildiu     gr1,vr2,vr3,mr4
           op0='00' op1='0011' op2='001' op3='0000'
vildf      gr1,vr2,vr3,mr4
           op0='00' op1='0011' op2='010' op3='0000'
vildd      gr1,vr2,vr3,mr4
           op0='00' op1='0011' op2='011' op3='0000'
vldcpi(id)  gr1,gr2,vr3,mr4
            op0='00' op1='0010' op2='000' op3='0000'
vldcpiu(id) gr1,gr2,vr3,mr4
            op0='00' op1='0010' op2='001' op3='0000'
vldcpf(id)  gr1,gr2,vr3,mr4
            op0='00' op1='0010' op2='010' op3='0000'
vldcpd(id)  gr1,gr2,vr3,mr4
            op0='00' op1='0010' op2='011' op3='0000'
```

ベクトルストア

ストア操作ではニーモニックの”(id)”に 3 ビットのアクセス id を指定して、同一のアクセス id を持つアクセス操作 (ロード/ストア) 間でプログラムに記述された実行順序が保証されるように実行させることができる。アクセス id は前置 <29:27> に設定させる。だだし、インダイレクトアクセス操作は先行アクセス操作との間で、常にプログラムに記述された実行順序が保証されるように実行されるため、アクセス id は指定しない。前置 <29:27> は’000’ リザーブとする。

```
vsti(id)    vr22,gr1,gr2,mr4
            op0=’01’ op1=’0001’ op2=’000’ op3=’0000’
vstf(id)    vr22,gr1,gr2,mr4
            op0=’01’ op1=’0001’ op2=’010’ op3=’0000’
vstd(id)    vr22,gr1,gr2,mr4
            op0=’01’ op1=’0001’ op2=’011’ op3=’0000’
vstm(id)    mr22,gr1,gr2,mr4
            op0=’01’ op1=’0001’ op2=’110’ op3=’0000’
visti       vr22,gr1,vr2,mr4
            op0=’01’ op1=’0011’ op2=’000’ op3=’0000’
vistf       vr22,gr1,vr2,mr4
            op0=’01’ op1=’0011’ op2=’010’ op3=’0000’
vistd       vr22,gr1,vr2,mr4
            op0=’01’ op1=’0011’ op2=’011’ op3=’0000’
vstexi(id)  vr22,gr1,gr2,mr4
            op0=’01’ op1=’0010’ op2=’000’ op3=’0000’
vstexf(id)  vr22,gr1,gr2,mr4
            op0=’01’ op1=’0010’ op2=’010’ op3=’0000’
vstexd(id)  vr22,gr1,gr2,mr4
            op0=’01’ op1=’0010’ op2=’011’ op3=’0000’
```

10.3.4 算術演算

加算/乗算/除算

加算/乗算/除算操作では入力オペランドに記号”-”を付加して、符号付きの入力オペランドの場合、符号を反転させた値を演算対象とし、また、符号無し入力オペランドの場合、1 の補数を演算対象とすることができる。記号”-”を vr1,gr1,fr1 に付加するとき前置 <29>=’1’、記号”-”を vr2 に付加するとき前置 <28>=’1’ を設定すること。前置 <27>=’0’ リザーブとする。記号”-”を付加しない場合、前置 <29>,<28> はそれぞれ’0’ を設定すること。

```
vaddi       (-)vr1,(-)vr2,vr3,mr4
            op0=’01’,op1=’0100’,op2=’000’,op3=’0000’
vaddi       (-)gr1,(-)vr2,vr3,mr4
            op0=’10’,op1=’0100’,op2=’000’,op3=’0000’
vaddl       (-)vr1,(-)vr2,vr3,mr4
            op0=’01’,op1=’0100’,op2=’001’,op3=’0000’
vaddl       (-)gr1,(-)vr2,vr3,mr4
            op0=’10’,op1=’0100’,op2=’001’,op3=’0000’
vadclu      (-)vr1,(-)vr2,(-)mr22,vr3,mr4
            op0=’01’,op1=’0100’,op2=’101’,op3=’0000’
vadclu      (-)gr1,(-)vr2,(-)mr22,vr3,mr4
            op0=’10’,op1=’0100’,op2=’101’,op3=’0000’
vaddf       (-)vr1,(-)vr2,vr3,mr4
```

```
          op0='01',op1='0100',op2='010',op3='0000'
vaddf     (-)fr1,(-)vr2,vr3,mr4
          op0='10',op1='0100',op2='010',op3='0000'
vaddd     (-)vr1,(-)vr2,vr3,mr4
          op0='01',op1='0100',op2='011',op3='0000'
vaddd     (-)fr1,(-)vr2,vr3,mr4
          op0='10',op1='0100',op2='011',op3='0000'
vaddq     (-)vr1,(-)vr2,vr3,mr4
          op0='01',op1='0100',op2='100',op3='0000'
vmuli     (-)vr1,(-)vr2,vr3,mr4
          op0='00',op1='0101',op2='000',op3='0000'
vmuli     (-)gr1,(-)vr2,vr3,mr4
          op0='10',op1='0101',op2='000',op3='0000'
vmull     (-)vr1,(-)vr2,vr3,mr4
          op0='00',op1='0101',op2='001',op3='0000'
vmull     (-)gr1,(-)vr2,vr3,mr4
          op0='10',op1='0101',op2='001',op3='0000'
vmullu    (-)vr1,(-)vr2,vr3,mr4
          op0='00',op1='0101',op2='101',op3='0000'
vmullu    (-)gr1,(-)vr2,vr3,mr4
          op0='10',op1='0101',op2='101',op3='0000'
vmulf     (-)vr1,(-)vr2,vr3,mr4
          op0='00',op1='0101',op2='010',op3='0000'
vmulf     (-)fr1,(-)vr2,vr3,mr4
          op0='10',op1='0101',op2='010',op3='0000'
vmuld     (-)vr1,(-)vr2,vr3,mr4
          op0='00',op1='0101',op2='011',op3='0000'
vmuld     (-)fr1,(-)vr2,vr3,mr4
          op0='10',op1='0101',op2='011',op3='0000'
vmulq     (-)vr1,(-)vr2,vr3,mr4
          op0='00',op1='0101',op2='100',op3='0000'
vdivl     (-)vr1,(-)vr2,vr3,mr4
          op0='01',op1='0111',op2='001',op3='0000'
vdivl     (-)gr1,(-)vr2,vr3,mr4
          op0='10',op1='0111',op2='001',op3='0000'
vdivlu    (-)vr1,(-)vr2,vr3,mr4
          op0='01',op1='0111',op2='101',op3='0000'
vdivlu    (-)gr1,(-)vr2,vr3,mr4
          op0='10',op1='0111',op2='101',op3='0000'
vdivf     (-)vr1,(-)vr2,vr3,mr4
          op0='01',op1='0111',op2='010',op3='0000'
vdivf     (-)fr1,(-)vr2,vr3,mr4
          op0='10',op1='0111',op2='010',op3='0000'
vdivd     (-)vr1,(-)vr2,vr3,mr4
          op0='01',op1='0111',op2='011',op3='0000'
vdivd     (-)fr1,(-)vr2,vr3,mr4
          op0='10',op1='0111',op2='011',op3='0000'
```

乗算加算複合

乗算加算複合操作では入力オペランドに記号"-"を付加して、符号付きの入力オペランドの場合、符号を反転させた値を演算対象とし、また、符号無し入力オペランドの場合、1 の補数を演算対象とすることができる。記号"-"を vr1,gr1,fr1 に付加したとき前置 <29>='1'、記号"-"を vr2 に付加したとき前置 <28>='1'、記号"-"を vr22,gr22,fr22 に付加したとき前置 <27>='1' をそれぞれ設定すること。記号"-"を付加しない場合、前置 <29>,<28>,<27> はそれぞれ'0' を設定すること。

```
vmulai     (-)vr1,(-)vr2,(-)vr22,vr3,mr4
           op0='00',op1='0100',op2='000',op3='0000'
vmulai     (-)gr1,(-)vr2,(-)vr22,vr3,mr4
           op0='10',op1='0100',op2='000',op3='0000'
vmulai     (-)vr1,(-)vr2,(-)gr22,vr3,mr4
           op0='00',op1='0100',op2='000',op3='0011'
vmulal     (-)vr1,(-)vr2,(-)vr22,vr3,mr4
           op0='00',op1='0100',op2='001',op3='0000'
vmulal     (-)gr1,(-)vr2,(-)vr22,vr3,mr4
           op0='10',op1='0100',op2='001',op3='0000'
vmulal     (-)vr1,(-)vr2,(-)gr22,vr3,mr4
           op0='00',op1='0100',op2='001',op3='0011'
vmulaf     (-)vr1,(-)vr2,(-)vr22,vr3,mr4
           op0='00',op1='0100',op2='010',op3='0000'
vmulaf     (-)fr1,(-)vr2,(-)vr22,vr3,mr4
           op0='10',op1='0100',op2='010',op3='0000'
vmulaf     (-)vr1,(-)vr2,(-)fr22,vr3,mr4
           op0='00',op1='0100',op2='010',op3='0100'
vmulad     (-)vr1,(-)vr2,(-)vr22,vr3,mr4
           op0='00',op1='0100',op2='011',op3='0000'
vmulad     (-)fr1,(-)vr2,(-)vr22,vr3,mr4
           op0='10',op1='0100',op2='011',op3='0000'
vmulad     (-)vr1,(-)vr2,(-)fr22,vr3,mr4
           op0='00',op1='0100',op2='011',op3='0100'
vmulaq     (-)vr1,(-)vr2,(-)vr22,vr3,mr4
           op0='00',op1='0100',op2='100',op3='0000'
```

開平

```
vsqrtf     vr1,vr3,mr4
           op0='00',op1='0111',op2='010',op3='0000'
vsqrtd     vr1,vr3,mr4
           op0='00',op1='0111',op2='011',op3='0000'
```

符号制御

```
vabsi      vr1,vr3,mr4
           op0='01',op1='1100',op2='000',op3='0000'
vabsl      vr1,vr3,mr4
           op0='01',op1='1100',op2='001',op3='0000'
vabsf      vr1,vr3,mr4
           op0='01',op1='1100',op2='010',op3='0000'
vabsd      vr1,vr3,mr4
```

```
          op0='01',op1='1100',op2='011',op3='0000'
vsgni     vr1,vr2,vr3,mr4
          op0='01',op1='0101',op2='000',op3='0000'
vsgni     gr1,vr2,vr3,mr4
          op0='10',op1='0101',op2='000',op3='0000'
vsgnl     vr1,vr2,vr3,mr4
          op0='01',op1='0101',op2='001',op3='0000'
vsgnl     gr1,vr2,vr3,mr4
          op0='10',op1='0101',op2='001',op3='0000'
vsgnf     vr1,vr2,vr3,mr4
          op0='01',op1='0101',op2='010',op3='0000'
vsgnf     fr1,vr2,vr3,mr4
          op0='10',op1='0101',op2='010',op3='0000'
vsgnd     vr1,vr2,vr3,mr4
          op0='01',op1='0101',op2='011',op3='0000'
vsgnd     fr1,vr2,vr3,mr4
          op0='10',op1='0101',op2='011',op3='0000'
```

10.3.5 比較

大小選択

```
vmaxi     vr1,vr2,vr3,mr4
          op0='00',op1='1101',op2='000',op3='0000'
vmaxi     gr1,vr2,vr3,mr4
          op0='10',op1='1101',op2='000',op3='0000'
vmaxl     vr1,vr2,vr3,mr4
          op0='00',op1='1101',op2='001',op3='0000'
vmaxl     gr1,vr2,vr3,mr4
          op0='10',op1='1101',op2='001',op3='0000'
vmaxf     vr1,vr2,vr3,mr4
          op0='00',op1='1101',op2='010',op3='0000'
vmaxf     fr1,vr2,vr3,mr4
          op0='10',op1='1101',op2='010',op3='0000'
vmaxd     vr1,vr2,vr3,mr4
          op0='00',op1='1101',op2='011',op3='0000'
vmaxd     fr1,vr2,vr3,mr4
          op0='10',op1='1101',op2='011',op3='0000'
vmini     vr1,vr2,vr3,mr4
          op0='01',op1='1101',op2='000',op3='0000'
vmini     gr1,vr2,vr3,mr4
          op0='10',op1='1101',op2='000',op3='0000'
vminl     vr1,vr2,vr3,mr4
          op0='01',op1='1101',op2='001',op3='0000'
vminl     gr1,vr2,vr3,mr4
          op0='10',op1='1101',op2='001',op3='0000'
vminf     vr1,vr2,vr3,mr4
          op0='01',op1='1101',op2='010',op3='0000'
vminf     fr1,vr2,vr3,mr4
```

```
           op0='10',op1='1101',op2='010',op3='0000'
vmind      vr1,vr2,vr3,mr4
           op0='01',op1='1101',op2='011',op3='0000'
vmind      fr1,vr2,vr3,mr4
           op0='10',op1='1101',op2='011',op3='0000'
```

比較

比較操作ではニーモニックの"(id)"は比較結果の割り込み機能で使用する 3 ビットの比較 id を前置 <29:27> に設定する。比較操作では 3 ビットの比較条件 cc を後置 <29:27> に設定する。比較条件 cc の意味は各操作ごとに定義される。

```
vcmpi(id)   cc,vr1,vr2,mr3,mr4
           op0='01',op1='0110',op2='000',op3='0000'
vcmpi(id)   cc,gr1,vr2,mr3,mr4
           op0='10',op1='0110',op2='000',op3='0000'
vcmpl(id)   cc,vr1,vr2,mr3,mr4
           op0='01',op1='0110',op2='001',op3='0000'
vcmpl(id)   cc,gr1,vr2,mr3,mr4
           op0='10',op1='0110',op2='001',op3='0000'
vcmplu(id)  cc,vr1,vr2,mr3,mr4
           op0='01',op1='0110',op2='101',op3='0000'
vcmplu(id)  cc,gr1,vr2,mr3,mr4
           op0='10',op1='0110',op2='101',op3='0000'
vcmpf(id)   cc,vr1,vr2,mr3,mr4
           op0='01',op1='0110',op2='010',op3='0000'
vcmpf(id)   cc,fr1,vr2,mr3,mr4
           op0='10',op1='0110',op2='010',op3='0000'
vcmpd(id)   cc,vr1,vr2,mr3,mr4
           op0='01',op1='0110',op2='011',op3='0000'
vcmpd(id)   cc,fr1,vr2,mr3,mr4
           op0='10',op1='0110',op2='011',op3='0000'
```

10.3.6 ビット操作

論理演算

論理演算操作では入力オペランドに記号"-"を付加して、入力オペランドのビットごとに否定した値を演算対象とすることができる。記号"-"を vr1,mr1,gr1,fr1 に付加したとき前置 <29>='1'、記号"-"を vr2,mr2 に付加したとき前置 <28>='1' を設定すること。前置 <27>='0' リザーブとする。記号"-"を付加しない場合、前置 <29>,<28> はそれぞれ'0' を設定すること。

```
vand        (-)vr1,(-)vr2,vr3,mr4
           op0='00',op1='1101',op2='101',op3='0000'
vand        (-)gr1,(-)vr2,vr3,mr4
           op0='10',op1='1101',op2='101',op3='0000'
vandm       (-)mr1,(-)mr2,mr3
           op0='00',op1='1101',op2='110',op3='0000'
vor         (-)vr1,(-)vr2,vr3,mr4
           op0='01',op1='1101',op2='101',op3='0000'
vor         (-)gr1,(-)vr2,vr3,mr4
           op0='10',op1='1101',op2='101',op3='0000'
```

```
vorm        (-)mr1,(-)mr2,mr3
           op0='01',op1='1101',op2='110',op3='0000'
vxor        (-)vr1,(-)vr2,vr3,mr4
           op0='01',op1='1100',op2='101',op3='0000'
vxor        (-)gr1,(-)vr2,vr3,mr4
           op0='10',op1='1100',op2='101',op3='0000'
vxorm       (-)mr1,(-)mr2,mr3
           op0='01',op1='1100',op2='110',op3='0000'
```

シフト

```
vsll        gr1,vr2,vr3,mr4
           op0='10',op1='1111',op2='101',op3='0000'
vsrl        gr1,vr2,vr3,mr4
           op0='10',op1='1111',op2='101',op3='0000'
vesl        vr1,vr2,vr3,mr4
           op0='01',op1='1110',op2='101',op3='0000'
vesl32      vr1,vr2,vr3,mr4
           op0='00',op1='1110',op2='101',op3='0000'
vclz        vr1,    vr3,mr4
           op0='00',op1='1100',op2='101',op3='0000'
```

10.3.7 マクロ

ベクトル要素の検索

```
vfnnm      mr1,gr3
           op0='00',op1='1010',op2='110',op3='0001'
vfnxm      mr1,gr3
           op0='01',op1='1010',op2='110',op3='0001'
vfxnm      mr1,gr3
           op0='00',op1='1011',op2='110',op3='0001'
vfxxm      mr1,gr3
           op0='01',op1='1011',op2='110',op3='0001'
vfnni      vr1,gr3,gr33,mr4
           op0='00',op1='1010',op2='000',op3='0101'
vfnxi      vr1,gr3,gr33,mr4
           op0='01',op1='1010',op2='000',op3='0101'
vfxni      vr1,gr3,gr33,mr4
           op0='00',op1='1011',op2='000',op3='0101'
vfxxi      vr1,gr3,gr33,mr4
           op0='01',op1='1011',op2='000',op3='0101'
vfnnl      vr1,gr3,gr33,mr4
           op0='00',op1='1010',op2='001',op3='0101'
vfnxl      vr1,gr3,gr33,mr4
           op0='01',op1='1010',op2='001',op3='0101'
vfxnl      vr1,gr3,gr33,mr4
           op0='00',op1='1011',op2='001',op3='0101'
vfxxl      vr1,gr3,gr33,mr4
           op0='01',op1='1011',op2='001',op3='0101'
```

```
vfnnlu      vr1,gr3,gr33,mr4
            op0='00',op1='1010',op2='101',op3='0101'
vfnxlu      vr1,gr3,gr33,mr4
            op0='01',op1='1010',op2='101',op3='0101'
vfxnlu      vr1,gr3,gr33,mr4
            op0='00',op1='1011',op2='101',op3='0101'
vfxxlu      vr1,gr3,gr33,mr4
            op0='01',op1='1011',op2='101',op3='0101'
vfnnf       vr1,gr3,fr33,mr4
            op0='00',op1='1010',op2='010',op3='0110'
vfnxf       vr1,gr3,fr33,mr4
            op0='01',op1='1010',op2='010',op3='0110'
vfxnf       vr1,gr3,fr33,mr4
            op0='00',op1='1011',op2='010',op3='0110'
vfxxf       vr1,gr3,fr33,mr4
            op0='01',op1='1011',op2='010',op3='0110'
vfnnd       vr1,gr3,fr33,mr4
            op0='00',op1='1010',op2='011',op3='0110'
vfnxd       vr1,gr3,fr33,mr4
            op0='01',op1='1010',op2='011',op3='0110'
vfxnd       vr1,gr3,fr33,mr4
            op0='00',op1='1011',op2='011',op3='0110'
vfxxd       vr1,gr3,fr33,mr4
            op0='01',op1='1011',op2='011',op3='0110'
```

ベクトル要素の抽出

```
vexti       gr1,vr2,gr3
            op0='01',op1='1001',op2='000',op3='0001'
vextl       gr1,vr2,gr3
            op0='01',op1='1001',op2='001',op3='0001'
vextf       gr1,vr2,fr3
            op0='01',op1='1001',op2='010',op3='0010'
vextd       gr1,vr2,fr3
            op0='01',op1='1001',op2='011',op3='0010'
```

ベクトル要素の総和

```
vsumm       (-)mr1,gr3
            op0='01',op1='1000',op2='110',op3='0001'
vsumi       vr1,gr3,mr4
            op0='01',op1='1000',op2='000',op3='0001'
vsuml       vr1,gr3,mr4
            op0='01',op1='1000',op2='001',op3='0001'
vsumf       vr1,fr3,mr4
            op0='01',op1='1000',op2='010',op3='0010'
vsumd       vr1,fr3,mr4
            op0='01',op1='1000',op2='011',op3='0010'
vipi        (-)vr1,(-)vr2,gr3,mr4
```

```
          op0='00',op1='1000',op2='000',op3='0001'
vipl      (-)vr1,(-)vr2,gr3,mr4
          op0='00',op1='1000',op2='001',op3='0001'
vipf      (-)vr1,(-)vr2,fr3,mr4
          op0='00',op1='1000',op2='010',op3='0010'
vipd      (-)vr1,(-)vr2,fr3,mr4
          op0='00',op1='1000',op2='011',op3='0010'
```

10.3.8 型変換

整数から浮動小数点数への変換

```
vcnvif    rm,vr2,vr3,mr4
          op0='00',op1='1111',op2='000',op3='0000'
vcnvid    rm,vr2,vr3,mr4
          op0='01',op1='1111',op2='000',op3='0000'
vcnvlf    rm,vr2,vr3,mr4
          op0='00',op1='1111',op2='001',op3='0000'
vcnvld    rm,vr2,vr3,mr4
          op0='01',op1='1111',op2='001',op3='0000'
```

浮動小数点数から整数への変換

```
vcnvfi    rm,vr2,vr3,mr4
          op0='00',op1='1111',op2='010',op3='0000'
vcnvdi    rm,vr2,vr3,mr4
          op0='00',op1='1111',op2='011',op3='0000'
vcnvfl    rm,vr2,vr3,mr4
          op0='01',op1='1111',op2='010',op3='0000'
vcnvdl    rm,vr2,vr3,mr4
          op0='01',op1='1111',op2='011',op3='0000'
```

浮動小数点数から浮動小数点数への変換

```
vcnvfd    rm,vr2,vr3,mr4
          op0='01',op1='1110',op2='010',op3='0000'
vcnvdf    rm,vr2,vr3,mr4
          op0='01',op1='1110',op2='011',op3='0000'
```

10.3.9 ベクトル編集

ベクトルの圧縮/伸張/複写

```
vcp       vr2,vr3,mr4
          op0='00',op1='0010',op2='101',op3='0000'
vex       vr2,vr3,mr4
          op0='01',op1='0010',op2='101',op3='0000'
vslid     vr2,vr3,mr4
          op0='00',op1='0110',op2='101',op3='0000'
vmovv     vr2,vr3,mr4
          op0='01',op1='0101',op2='101',op3='0000'
```

```
vmovm       mr2,mr3,mr4
            op0='01',op1='0101',op2='110',op3='0000'
```

ベクトルの生成

```
vgfi        gr1,vr3,mr4
            op0='00',op1='1100',op2='000',op3='0000'
vgfl        gr1,vr3,mr4
            op0='00',op1='1100',op2='001',op3='0000'
vgff        fr1,vr3,mr4
            op0='00',op1='1100',op2='010',op3='0000'
vgfd        fr1,vr3,mr4
            op0='00',op1='1100',op2='011',op3='0000'
vgsi        gr1,vr3,mr4
            op0='00',op1='0110',op2='000',op3='0000'
vgsl        gr1,vr3,mr4
            op0='00',op1='0110',op2='001',op3='0000'
vgsm        gr1,mr3
            op0='00',op1='0110',op2='110',op3='0000'
vgm         inv,gr1.mr
            op0='00',op1='1100',op2='110',op3='0000'
```

表 10.1: 操作コード一覧

op1	op0	op2<18:16>							
<22:19>	<25>	000	001	010	011	100	101	110	111
0000	0	ここは使用しない							
	1	ここは使用しない							
0001	0	vldi	vldiu	vldf	vldd			vldm	vltr
	1	vsti		vstf	vstd			vstm	
0010	0	vldcpi	vldcpiu	vldcpf	vldcpd		vcp		
	1	vstexi		vstexl	vstexd		vex		
0011	0	vildi	vildiu	vildf	vildd				
	1	visti		vistf	vistd				
0100	0	vmulai	vmulal	vmulaf	vmulad	vmulaq			vplcr
	1	vaddi	vaddl	vaddf	vaddd	vaddq	vadclu		vlvt
0101	0	vmuli	vmull	vmulf	vmuld	vmulq	vmullu		vlvl
	1	vsgni	vsgnl	vsgnf	vsgnd		vmovv	vmovm	vlvlf
0110	0	vgsi	vgsl				vslid	vgsm	vlcr
	1	vcmpi	vcmpl	vcmpf	vcmpd		vcmplu		vsetcf
0111	0			vsqrtf	vsqrtd				vnlfy
	1		vdivl	vdivf	vdivd		vdivlu		vpt
1000	0	vipi	vipl	vipf	vipd				vpscr
	1	vsumi	vsuml	vsumf	vsumd			vsumm	vsvt
1001	0								
	1	vexti	vextl	vextf	vextd				
1010	0	vfnni	vfnnl	vfnnf	vfnnd		vfnnlu	vfnnm	vscr
	1	vfnxi	vfnxl	vfnxf	vfnxd		vfnxlu	vfnxm	
1011	0	vfxni	vfxnl	vfxnf	vfxnd		vfxnlu	vfxnm	
	1	vfxxi	vfxxl	vfxxf	vfxxd		vfxxlu	vfxxm	
1100	0	vgfi	vgfl	vgff	vgfd		vclz	vgm	
	1	vabsi	vabsl	vabsf	vabsd		vxor	vxorm	
1101	0	vmaxi	vmaxl	vmaxf	vmaxd		vand	vandm	
	1	vmini	vminl	vminf	vmind		vor	vorm	
1110	0						vesl32		
	1			vcnvfd	vcnvdf		vesl		
1111	0	vcnvif	vcnvlf	vcnvfi	vcnvdi		vsll		
	1	vcnvid	vcnvld	vcnvfl	vcnvdll		vsrl		

10.4 複数 PE 連携機能

本節では，複数 PE を連携させる機能について説明する．ハードウェアによる PE 間キャッシュ一貫性保証機構はないものの，後述する PE 間専用キューは相互待ち合わせ機能を有する．

10.4.1 ICPL 命令

ICPL 命令は，複数 PE 連携のためのヒント情報である．ICPL 命令を含む VLIW 命令内において，ICPL 命令と同時に指定可能な一般命令は BNO に制限されており，VLIW 全体として複数 PE 連携を指示する．図 10.3 に ICPL 命令を含む VLIW 命令の詳細を示す．なお，ICPL 命令を NOP と見なし単独 PE により継続実行しても正しい結果が得られる．

図 10.3: ICPL 命令

10.4.2 複数 PE による並列実行

複数 PE により並列実行したい命令範囲の開始点を ICPL(#lk=1)，終了点を ICPL(#lk=0) により挟み，さらに前者において PE 間でデータを待ち合わせるアドレス範囲を指定することにより，L2 キャッシュを経由しない相互待ち合わせと並列実行が可能である．図 10.3 の下段に示すように，並列実行開始点では形式 (A) の ICPL 命令，境界点では形式 (B) の ICPL 命令を指定し，並列実行対象命令区間の最終ブロックは，形式 (C) および (D) の ICPL 命令により挟む．この場合，最初の (A) が PE0 により実行され，引き続くブロックが PE0 により実行されると同時に，最初の (A) に指定された target(2 番目の A) が PE0 から PE1 に送信される．同様に，2 番目の (A) が PE1 により実行され，引き続くブロックが PE1 により実行されると同時に，2 番目の (A) に指定された target(3 番目の A) が PE1 から PE2 に送信される．さらに，3 番目の (A) が PE2 により実行され，引き続くブロックが PE2 により実行されると同時に，3 番目の (A) に指定された target(C) が PE2 から PE3 に送信される．以上の結果，PE0-3 により各担当ブロックが並列実行される．なお，各 PE が (A) に到達しても，次に起動可能な空き PE が存在しない場合は，PE を起動せず，引き続くブロックを自身が実行するのみである．

各担当ブロックの実行が終了し (B) または (D) に到達した時点で，PE1-3 は待機状態となる．一方，PE0 が (B) に到達すると，次のブロックを実行した PE が存在するかどうかを検査し，存在する場合は (B) に指定された target (次の B) へ分岐することを繰り返す．一方，存在しない場合は PE0 を起点として起動した全 PE が待機状態となるまで待機し，その後，次命令の実行を開始する．このように理想的に実行できるアプリケーションプログラム例は，図 2.10 に示した通りである．

10.4.3 L2 キャッシュを経由しない相互待ち合わせ

SAPP は，複数 PE による並列実行機能に加えて，PE 間のデータ待ち合わせ機能を有する．形式 (A) の ICPL 命令において指定された連携領域の先頭アドレスおよび長さは，各 PE の外部メモリインタフェースに接続された共通外部メモリ制御部内に記録される．共通外部メモリ制御部は，さらに L1 キャッシュの 1-way 相当の容量を有する PE 間専用キューを備えており，連携領域に含まれるデータは外部キャッシュではなく PE 間専用キューを経由して PE 間で直接送受される．後続ブロックを実行中の PE（後続 PE と呼ぶ）が PREFETCH に伴う L1 キャッシュミスを検出した場合，Chapter17.3 において説明した外部メモリインタフェースに参照要求がキューイングされる．この時，共通外部メモリ制御部は，要求されたアドレスが連携領域に含まれるか否かを検査し，含まれる場合，先行ブロックを実行中の PE（先行 PE と呼ぶ）が連携領域に対応するアドレスの有効データを外部メモリインタフェースに FLUSH するまで，後続 PE に対

CCT.CORMSK	非特	実コア番号制御マスク（**64bit**） **0:**制御対象外，**1:**制御対象
CCT.RQTYPE	非特	**0:**起動要求 **1:**停止要求
CCT.SPINIT	非特	**%sp**初期値（**DSM**内相対共通アドレス）
CCT.SPLIMT	非特	**%sp**下限値（**DSM**内相対共通アドレス）
CCT.PSTART	非特	**PC**初期値（**DDR3**内共通開始アドレス）

図 10.4: 並列実行制御情報

CCT.TRIGAD	非特	**sleep**解除トリガ（自**DSM**内相対アドレス）
CCT.TRIMSK	非特	**sleep**解除トリガ（マスク値）
CCT.TRIGDT	非特	**sleep**解除トリガ（期待値または任意値）

図 10.5: 監視条件制御情報

する応答を保留する．要求されたアドレスが連携領域に含まれない場合，共通外部メモリ制御部は，外部キャッシュを参照し，後続 PE に対して応答を返す．このような機構を積極的に利用することにより，PE 数が増加した場合でも，共通外部メモリ制御部と外部キャッシュ間のバンド幅が不足することなく，リニアな性能向上が期待できる．

10.5 明示的並列プログラムの実行

ユーザプログラムは，複数コアによる並列実行を行う場合においても 1 つの実行形式ファイルとして構成される．OS 機能に含まれるローダは，ファイル格納装置に存在する実行形式ファイル中のヘッダ情報に基づき，ユーザプロセス起動のために必要なアドレス変換機構の初期化を行い，DDR 上にテキスト領域およびデータ領域を配置する．次に，OS が使用を許可した実コアの中から選択された 1 つのコア（通常は実コア番号 0）が，ヘッダ情報中の実行開始アドレスよりプログラムの実行を開始する[2]．

プログラム実行開始後に，あるコアが複数コアによる並列実行を開始する際には，図 10.4 に示す並列実行制御情報（実コア番号制御マスク，各コアのスタックポインタ初期値，下限値，実行開始アドレス）をハードウェアに対して発行する．本発行により，ハードウェアは，当該各コアに対して並列実行制御情報を通知する．各コアは，各制御情報に従いスタックフレームを初期化し，指定された実行開始アドレスより命令の実行を開始する[3]．その後，一般的には，並列実行に必要となるデータを同一論理メモリ空間内において転送開始した後（高速化のためには先行する演算とオーバラップさせることが望ましい），図 10.5 に示す監視条件制御情報（監視アドレス，マスク値および期待値）をハードウェアに対して発行し，データ転送の終結まで sleep する．sleep 命令発行時に監視条件がすでに充足している場合は，sleep することなく次命令以降の実行を開始する．また，sleep からの復帰は，ハードウェアによる自 DSM への書き込みの検出，値のマスクおよび比較，割り込みの通知，OS による判断，ユーザプログラムのコンテクスト再開という手順により行われる．すなわち，OS は，sleep 命令の実行を契機として，当該コアをコンテクストスイッチさせ，別のプロセスを実行することが可能である[4]．ただし，当初の sleep から復帰するまで，監視条件制御情報の内容を変更してはならず，別のプロセスを実行可能とはいえ，新たに並列実行制御情報や監視条件制御情報の設定を必要とするプロセスに切り替えることはできない．このような切り替えは OS がプロセス種

[2]明示的並列プログラムのプログラムヘッダには，使用する実コア数（実コア番号上限+1），使用する DDR/DSM/CSM 量，プログラムの特性に合わせた物理 DDR から DDR 空間への望ましい写像に関するヒント情報，同様に物理 CSM から実 CSM 予約空間への望ましい写像に関するヒント情報，および，プログラム実行開始アドレスが含まれるはずである．

[3]コア数に関わらず CCT.SPINIT および CCT.SPLIMT が 1 組で済むのは，DSM 内相対アドレスに限定することにより，物理的に分散することが保証されるからである．また，CCT.PSTART が 1 組で済むのは，DDR3 上に配置された同一命令列を用いることを前提とするためである．すなわち，CCT.PSTART に始まる並列処理用共通命令列は，自身のコア番号を使用して分担範囲を認識する構成でなければならない．

[4]OS を搭載しないシミュレータに対しては，当該コアのシミュレーションをスキップして高速化を図ることが期待される．

別を認識することにより回避しなければならない．すなわち，切り替え先のプロセスは，アドレス変換が多重化されている DDR，DSM，CSM（およびモデルによってはこれらの空間を参照する DTU）を使用してよいが，複数コアによる並列実行および終結監視機能を使用してはならない．

あるコアが，複数コアによる並列実行の終了を待ち合わせる際には，前述の監視条件制御情報（監視アドレス，マスク値および期待値）をハードウェアに対して発行し，他コアがストア命令により実行完了を通知するまで sleep する．CCT.TRIGAD には，自コアの DSM 内相対アドレスのみを指定可能であり，範囲外の値を設定した場合の動作は保証されない．なお，CCT.TRIGAD は 8 バイト境界から始まる 8 バイト領域に対応しており，同じく 8 バイトの CCT.TRIMSK と組み合わせることにより，最大 8 個のコアからの各 1 バイトストアを同一 8 バイト領域内の異なる 1 バイトに対応付け，一度に 8 コアの実行完了を待ち合わせることができる．すなわち，本機構により 64 コアの実行完了を待ち合わせる場合には，監視条件制御情報の設定および sleep を 8 回繰り返せば良い．なお，sleep から復帰直後，同一 8 バイトの一部が未完了であるために再度 8 バイト全体を監視対象とする場合を除き，監視対象から外れる CCT.TRIGAD の領域に対しては，次に監視対象として設定する期待値とは異なる値を書き込んでおく必要がある．

図 10.6 にプログラムの実行例を示す．起動直後，DDR 上に配列Ｘ，Ａ，Ｂ，Ｃの領域が確保され，内容が初期化される．次に並列実行制御情報の発行により，4 つのコアが起動起動状態となり，各コアが DDR から各 DSM に対するデータ転送（ＡとＣは一部，Ｂは全部）を開始する．データ転送の間，各コアは各々監視条件を設定し sleep する．DTU から DSM への最後の書き込み完了を監視条件とすることにより，データ転送完了と同時に各コアの演算が開始される．各 DSM に生成された演算結果をさらに全コアで使用する場合には，個別に DSM 間転送を開始し，データ転送の間，他の 3 コアからのデータ転送を待ち合わせるための監視条件を設定し sleep する．最終的に，各 DSM に生成された途中結果（Ｘ）や最終結果（Ｙ）を DSM から DDR に個別に転送開始し，データ転送の間，プログラムの実行を開始したコアにおいて監視条件を設定し sleep する．なお，並列実行が終了したコアは，新たに並列実行制御情報を受信するまでの間，sleep 状態を維持する．

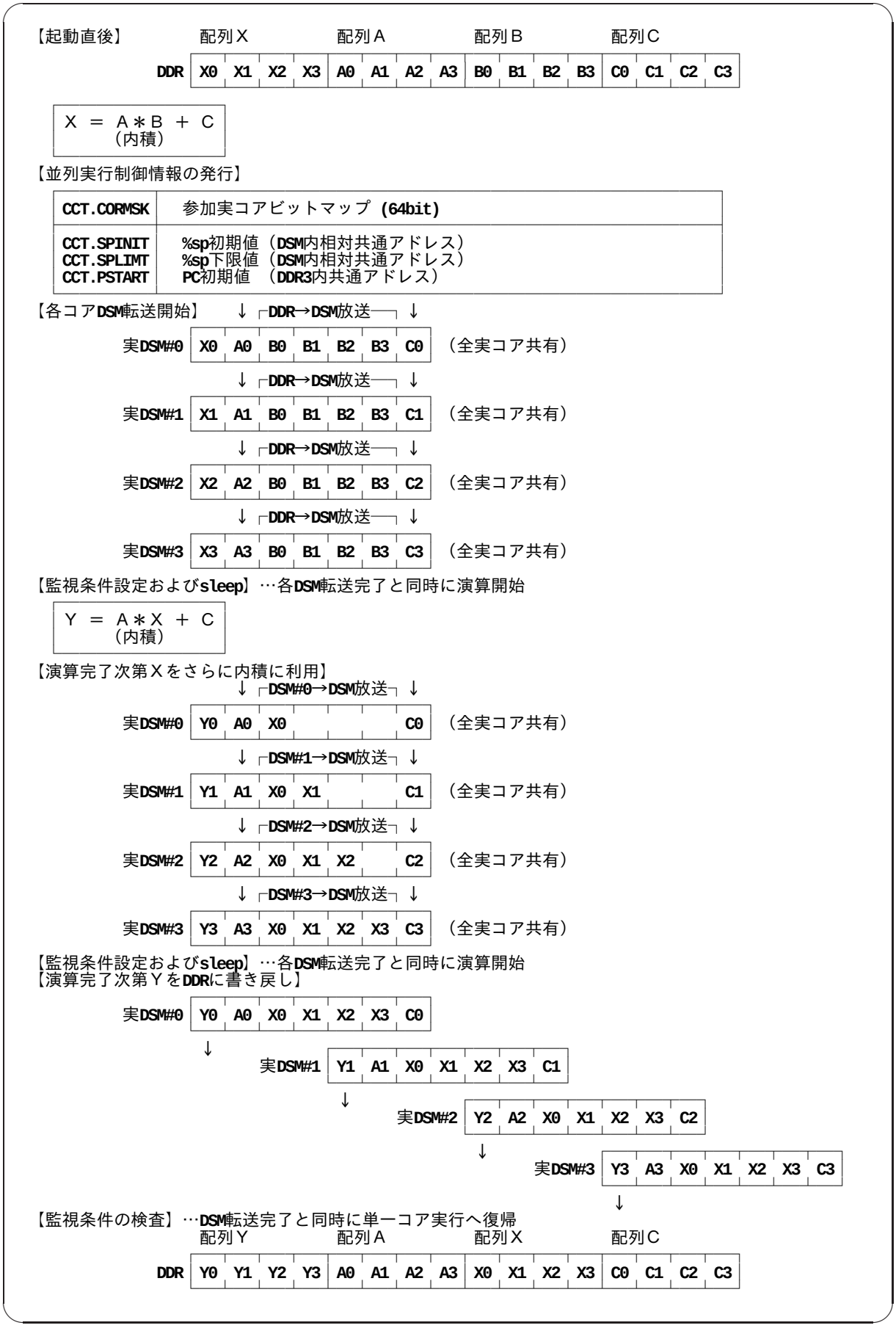


図 10.6: 明示的並列プログラムの実行

Chapter 11

プログラミングガイド

11.1 はじめに

アレイ実行を行うために対象ループに最低限必要な条件は以下のとおりである．

- アレイ化可能なのは最内ループのみ．
- イタレーション間で依存関係があってはいけない．ループ変数や最大値，最小値，総和演算には対応．
- if 文は sel（選択命令）で記述可能なもののみ．
- ループ中で break は不可だが，ループ終了条件に含めることは可能（未検証）．ただし，プリフェッチ幅はループ開始時点で決定することに注意．
- continue は条件付き STORE で実装可能．

以降では，詳細について順に述べる．

11.2 入出力データの確認

次に，プログラム中の変数を以下の 4 種類に分類する．

- 定数
- 入力配列
- 出力配列（入出力配列を含む）
- 変数

定数と変数については基本的に既存プログラムと同様に扱うことができるため，ここでは入力および出力配列に注目する．以降ではアクセス単位を**データストリーム**と呼ぶ．データストリームは 1 次元配列に相当し，ベクトル型プロセッサにおけるベクトルレジスタに対応する．アレイ実行で扱えるデータには以下の制限がある．

- 1 つのデータストリームの範囲が 4KB まで
- 最大 4B × 1024
- ストライドを使うと要素数は反比例で減少（アクセスしない領域も 4KB に含まれる）
- ストライド長は 2^n のみ
- 出力は 4 データストリームのみ (WAY0,4,8,12)．WAY0,4,8,12 は入出力ストリームとすることも可能
- 入力は 12 ストリーム．アクセス可能な要素はイタレーション毎に順次指定された方向へ移動していく．例外として，特定の場所に配置されたロード命令からはランダムアクセスが可能．
- 以上よりアクセス可能なデータの最大は 4KB*(1+12)=52KB．出力は 4KB まで．これ以上はループ長を分割する必要がある．

あるイタレーションに注目すると，アクセス可能なデータは各ストリーム中の連続する 8 要素であり，このアクセス可能要素はイタレーションの実行に伴って順次指定された方向に移動する．また，同一のロー

ドストアユニットに接続されたストリーム群毎に 1 回のみ任意の要素にアクセスすることができる．ハードウェア構成との対応により，各 WAY に格納されたデータにアクセス可能なロードストアユニットにはいくつかの制限がある．WAY0-3 は初段に接続されているため，すべてのロードストアユニットからアクセスが可能であるが，WAY4-7/8-11/12-15 はそれぞれ第 10 段，第 19 段，第 28 段に接続されているため，接続されている段以降のロードストアユニットからのみアクセスが可能である．この制約により，極端な例では 10 回以上アクセスする必要があるデータは WAY13-15 に格納することはできない制限が発生する．実際には，より多くのデータにアクセスするために，WAY1-3 は初段から第 9 段の 9 回，WAY5-7 は第 10 段から第 18 段の 9 回アクセスが可能（以下同様）という構成で利用する事が多い．この場合，各 WAY に均等にアクセスした場合，各 3 回ずつのアクセスが可能である．

11.3 プリフェッチのスケジューリング

11.3.1 2 次元データ

画像処理のように 2 次元データを扱う場合には，1 回のアレイ実行が X 軸方向のループに対応する．このとき，WAY1，2，3 をそれぞれ Y-1,Y,Y+1 に対応付け，自動ローテーション機能を利用する事により，1 回のアレイ実行が終了する度に新しい WAY を 1 つ分のみプリフェッチするだけでよい．このとき，最も古い WAY が自動的に追い出される．このように 2 次元データを扱う場合には WAY0 に出力データ，WAY1-3 に入力データを対応させる事が望ましい．他のデータが必要なければ，全段ですべてのデータにアクセスする事が可能である．

11.3.2 3 次元データ

また，3 次元データの場合には，WAY1-3，5-7，9-11 をそれぞれ Z-1，Z，Z+1 に対応付ける事により，1 回のアレイ実行が終了する度に新しい WAY を 3 つ分プリフェッチすればよい．このように 3 次元データを扱う場合には WAY0 に出力データ，WAY1-3/5-7/9-11 に入力データを対応させる事が望ましい．この場合，WAY1-3/5-7/9-11 にアクセスできるのはそれぞれ 9 段であるので，3×3×3 要素にアクセスするためには，初段から 27 段目までもれなくロード命令を配置する事が必須となる．

11.3.3 複数の 1 次元データ

ベクトル演算のように，複数の 1 次元データから 1 つの 1 次元データを計算する場合には，入力データを WAY1-3/5-7/9-11/13-15 に割り当てる．演算内容にも依存するが，一般的には演算の依存関係を考慮し，演算のクリティカルパスの上流にあるデータを WAY1-3 に割り当てることが望ましい．複数の 1 次元データを扱う場合には，入力データにおけるアレイ実行間の再利用性が無いことが多いため，WAY への対応が性能に影響を与えることは少ない．

11.3.4 複数の 2 次元データ

差分法のシミュレーションのように複数の 2 次元データを扱う場合について考える．アレイ実行間の再利用性がある場合には，再利用性のあるデータは同じ段に接続された WAY に割り当てることが望ましい．例えば再利用性のあるデータが 2 つ (P, V とする) あり，それぞれのデータが 2×2 要素（すなわち X[j][i], X[j][i-1], X[j-1][i], X[j-1][i-1]）にアクセスする場合には，WAY1, 2 に P を，WAY5, 6 に V を割り当てることが望ましい．ここで，WAY3 や way7 には再利用性のないデータを割り当てることで WAY を有効に利用することができる．これにより，再利用性がハードウェアに正しく認識され，必要最低限のプリフェッチが行われるようになる．

11.4 アセンブリプログラミング

本節ではアレイ実行のためのアセンブラプログラムを記述する場合において注意すべき点について述べる．

ループ アレイ実行は最内ループを対象にするが，対象ループの終了条件は条件付きの前方分岐命令，ループの終端はループ先頭に戻る無条件後方分岐でなければならない．また，終了条件はループ中で 1 回のみ記述できる．

定数 アレイ実行中の定数はあらかじめループ開始前にレジスタに保存されていることが望ましい．メモリからロードすることも原理的には可能であるが，事前にプリフェッチが必要となり，（貴重な資源である）WAY を消費してしまうとともに，ロード命令の配置場所に注意が必要である．

変数 変数は通常のレジスタと同様に扱うことができるが，**アレイ実行終了後は利用したレジスタの値は不定**となる．

ロード 前節で述べたとおり，ロード命令を正しく実行するには事前にプリフェッチが必要であり，プリフェッチの方法によってロード命令を配置できる場所に制限がある．

ストア ストア命令は 1 イタレーションあたり 1 ワードまで可能である．1 ワードを超えるストアが発生した場合は最後の 1 ワードのみがメモリに書き込まれ，その他の値はどこにも書き込まれないが，同一イタレーションの後続のロード命令で読み込むことができる．

ロードストアアドレス ロードストアアドレスのベースレジスタの更新には `addi`，`subi` のみを用いることができるという制約があるため，基本的には `ldi`，`ldfi`，`sti`，`stfi` 命令を用いることが望ましい．`ld` 命令を使うことも可能であるがその利用には注意が必要である．

演算 イタレーション間に依存関係がない限り基本的に自由に記述することができるが，アレイ長が有限であるためそれを超えることはできない．アレイ長を超えるループを記述した場合には強制的に通常実行で行われる．

gcc を用いる場合には、特にループ条件と、ロードストアアドレスの更新方法に不都合が発生することが多い。また、アレイ実行後のレジスタの値がアレイ実行を行った場合と行わなかった場合で異なるので、両者で正しく動作するコードを記述するには注意が必要である。この問題は一つの関数内で複数のアレイ実行を行う場合に発生しやすい。gcc は利用する直前に定数をロードすることも多いがこれは容易に対応できる。

11.5 命令スケジューラ

SAPP ではアレイ段数や伝搬レジスタといったこれまでのプロセッサとは異なる評価パラメータが存在する．また，1 回のアレイ実行に含まれる命令数が実行時間に影響を与えないという特徴や，データの格納 WAY に対応して，対応するロード命令の配置可能な場所が限定されるという制約がある．したがって，アレイ実行対象のプログラムの命令スケジューリングにもこれまでとは異なる方針が必要である．

現在の命令スケジューラでは以下の方針で最適な命令配置を求める．

1. 論理的に正しく動作するすべての命令配置を列挙する．
2. 以下の資源について上限や制約があらかじめ与えられている場合には，違反するものを取り除く．
   - 演算器段数
   - 各段の演算器数
   - 各段の伝搬レジスタ数
   - ロード可能位置（未実装）
3. 個々の配置パターンに対して各段の必要伝搬レジスタ数の合計を求め，もっとも少ないものが最適な命令配置である．

11.6 case study

11.6.1 単純な画像処理（サンプルコード，unsharp，blur，edge，bblur に対応）

本節では出力画像の 1 画素が，入力画像中の 3×3 画素を用いた演算で求められる画像処理を SAPP で実行する場合について述べる．画像処理では，カウンタ，中心座標アドレス，書き込みアドレスの各パラメータが入力引数として与えられる．実行の流れは以下のようになる．

プリフェッチ アレイ実行開始前にあらかじめ入力画素データを読み込む．
アレイ実行

load 中心座標からの相対アドレスで必要な画素を読み込み（ldi または ldfi）．プリフェッチした WAY とロード命令の配置場所の対応に注意．
演算 イタレーション間に依存関係はないので，VLIW 命令列がそのまま利用可能．
STORE

11.6.2 複雑な画像処理

テーブルの利用（**tone_curve**） ガンマ補正のようなテーブルが必要な場合は，アレイ実行の前に読み込んでおく．
拡張（**expand**） パラメータを適切に設定することにより，読み書きのインクリメント幅を変えることが可能．（中心座標を単純なインクリメントではなく演算で求める）

11.6.3 数値解析（サンプルコード，daxpy，dscal_r，idamax，fft に対応）

ベクトルをデータストリームに対応 daxpy のような更新型は R/W
idamax MAX は対応可能だが工夫が必要．

11.7 その他

- ストライド情報はオペランドで指定するので，動的な変更は不可．
- freg のスコープは関数内を仮定している．（soft-float を付けているので gcc は freg を使わない．）

11.7.1 FRV からの逸脱

※ここに書いてあることがすべてとは限らないので注意．

- cpr の実体は存在せず，gr を指したり fr を指したりする．

11.8 プログラム例

11.8.1 フレーム補間

```
void hokan1(c, p, s)
     unsigned int *c, *p;
     unsigned short *s; /*[WD/4][8];*/
     /*hokan1(&W[i*WD], &R[(i+j)*WD], &SAD1[i/4][j+4]);*/
{
  int j;
  for (j=0; j<WD; j++) {
    int j2 = j/4*4;
    int k = j%4*2; /* j2+k:0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
    * s    += df(c[j2],p[j2+k-4]) + df(c[j2+1],p[j2+k-3]) + df(c[j2+2],p[j2+k-2]) + df(c[j2+3],p[j2+k-1]);
                  /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
    *(s+1) += df(c[j2],p[j2+k-3]) + df(c[j2+1],p[j2+k-2]) + df(c[j2+2],p[j2+k-1]) + df(c[j2+3],p[j2+k  ]);
                  /* p[-3],p[-2],p[-1],p[ 0] -> p[-1],p[ 0],p[1],p[2] */
    s += 2;
  }
}
```

```
hokan1: addi   sp,#-48,sp
        sti.p  fp,@(sp,#32); addi sp,#32,fp
        movsg  lr,gr11
        sti    gr11,@(fp,#8)
        addi.p gr0,321,gr7;  addi gr0,-1,gr10
        sethi.p #hi(0x80000000|(320<<16)|320),gr13; setlo #lo(0x80000000|(320<<16)|320),gr13
        dcpl.p gr6,gr13,#0; addi.p gr4,#8,gr0; addi gr5,#8,gr0
hokan1_loop:
        addi.p gr10,1,gr10;                              subicc gr7,#1,gr7,icc0
        srli.p gr10,2,gr11;     andi.p gr10,3,gr12;      beq icc0,0x0,hokan1_exit
        slli.p gr11,4,gr12;     slli   gr12,3,gr13       /* j2*4,k*4 */
        add    gr12,gr13,gr13                            /* (j2+k)*4 */
        add.p  gr4,gr12,gr12;   add    gr5,gr13,gr13     /* &c[j2],&p[j2+k] */
        ldfi   @(gr13,-16),fr4
        ldfi   @(gr12,0),fr0
        ldfi   @(gr13,-12),fr5
        ldfi.p @(gr12,4),fr1;   msad   cpr0,cpr4,cpr10
        ldfi.p @(gr13,-8),fr6;  msad   cpr0,cpr5,cpr11
        ldfi.p @(gr12,8),fr2;   msad   cpr1,cpr5,cpr12
        ldfi.p @(gr13,-4),fr7;  msad.p cpr1,cpr6,cpr13; mnop1.p; maduh cpr10,cpr12,cpr10
        ldfi.p @(gr12,12),fr3;  msad.p cpr2,cpr6,cpr12; mnop1.p; maduh cpr11,cpr13,cpr11
        ldfi.p @(gr13,0),fr8;   msad.p cpr2,cpr7,cpr13; mnop1.p; maduh cpr10,cpr12,cpr10
                                msad.p cpr3,cpr7,cpr12; mnop1.p; maduh cpr11,cpr13,cpr11
                                msad.p cpr3,cpr8,cpr13; mnop1.p; maduh cpr10,cpr12,cpr10
                                msml.p cpr10,cpr10;     mnop1.p; maduh cpr11,cpr13,cpr11
        ldfi.p @(gr6,0),fr0;    msmh   cpr11,cpr11
                                mor    cpr11,cpr10,cpr10
                                maduh  cpr0,cpr10,cpr0
        stfi.p fr0,@(gr6,0);    addi.p gr6,4,gr6;        bra  hokan1_loop
```

11.8.2 鮮鋭化

```
void unsharp(p, r, i)
     unsigned char *p;
     unsigned char *r;
     int i;
{
  int t0,t1,t2;
  int j, k;
  int p0 = ((i  )*WD+(1  ))*4;  // p1 p5 p2
  int p1 = ((i-1)*WD+(1-1))*4;  // p6 p0 p7
  int p2 = ((i-1)*WD+(1+1))*4;  // p3 p8 p4
  int p3 = ((i+1)*WD+(1-1))*4;
  int p4 = ((i+1)*WD+(1+1))*4;
  int p5 = ((i-1)*WD+(1  ))*4;
  int p6 = ((i  )*WD+(1-1))*4;
  int p7 = ((i  )*WD+(1+1))*4;
  int p8 = ((i+1)*WD+(1  ))*4;
  for (j=0; j<WD; j++) {
    r[p0+0] = 0;

    t0 = p[p0+1];
    t1 = p[p1+1] + p[p2+1] + p[p3+1] + p[p4+1];
    t2 = p[p5+1] + p[p6+1] + p[p7+1] + p[p8+1];
    r[p0+1] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    t0 = p[p0+2];
    t1 = p[p1+2] + p[p2+2] + p[p3+2] + p[p4+2];
    t2 = p[p5+2] + p[p6+2] + p[p7+2] + p[p8+2];
    r[p0+2] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    t0 = p[p0+3];
    t1 = p[p1+3] + p[p2+3] + p[p3+3] + p[p4+3];
    t2 = p[p5+3] + p[p6+3] + p[p7+3] + p[p8+3];
    r[p0+3] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    p0+=4; p1+=4; p2+=4; p3+=4; p4+=4; p5+=4; p6+=4; p7+=4; p8+=4;
  }
}
```

```
unsharp: ...
         sethi.p #hi((320<<16)|320),gr6; setlo #lo((320<<16)|320),gr6
         dcpl.p  gr5,gr6,#0; addi.p gr4,-1280,gr0; addi.p gr4,0,gr0; addi gr4,1280,gr0
unsharp_loop:
         ldfi.p  @(gr4,-1284),fr1;                subicc gr7,#1,gr7,icc0
         ldfi.p  @(gr4,-1276),fr2;                beq icc0,0x0,unsharp_exit
                                                  mb2h.p  cpr1,0,cpr12;               mb2h  cpr1,1,cpr13
         ldfi.p  @(gr4,1276),fr3;                 mb2h.p  cpr2,0,cpr14;               mb2h  cpr2,1,cpr15
                                                  maduh.p cpr12,cpr14,cpr12; mnop1.p; maduh cpr13,cpr15,cpr13
         ldfi.p  @(gr4,1284),fr4;                 mb2h.p  cpr3,0,cpr16;               mb2h  cpr3,1,cpr17
                                                  maduh.p cpr12,cpr16,cpr12; mnop1.p; maduh cpr13,cpr17,cpr13
         ldfi.p  @(gr4,-1280),fr5;                mb2h.p  cpr4,0,cpr18;               mb2h  cpr4,1,cpr19
         ldfi.p  @(gr4,-4),fr6;                   maduh.p cpr12,cpr18,cpr12; mnop1.p; maduh cpr13,cpr19,cpr13
                                                  mb2h.p  cpr5,0,cpr14;               mb2h  cpr5,1,cpr15
         ldfi.p  @(gr4,4),fr7;                    mb2h.p  cpr6,0,cpr16;               mb2h  cpr6,1,cpr17
                                                  maduh.p cpr14,cpr16,cpr14; mnop1.p; maduh cpr15,cpr17,cpr15
         ldfi.p  @(gr4,1280),fr8;                 mb2h.p  cpr7,0,cpr18;               mb2h  cpr7,1,cpr19
                                                  maduh.p cpr14,cpr18,cpr14; mnop1.p; maduh cpr15,cpr19,cpr15
         ldfi.p  @(gr4,0),fr0; addi.p gr4,#4,gr4; mb2h.p  cpr8,0,cpr20;               mb2h  cpr8,1,cpr21
                                                  maduh.p cpr14,cpr20,cpr14; mnop1.p; maduh cpr15,cpr21,cpr15
                                                  mb2h.p  cpr0,0,cpr10;               mb2h  cpr0,1,cpr11
                                                  mmlh.p  cpr10,cpr9,cpr10;  mnop1.p; mmlh  cpr11,cpr9,cpr11
                                                  mmlh.p  cpr12,cpr30,cpr12; mnop1.p; mmlh  cpr13,cpr30,cpr13
                                                  msrl.p  cpr14,2,cpr16;     mnop1.p; msrl  cpr15,2,cpr17
                                                  mmlh.p  cpr14,cpr31,cpr14; mnop1.p; mmlh  cpr15,cpr31,cpr15
                                                  msbuh.p cpr10,cpr12,cpr10; mnop1.p; msbuh cpr11,cpr13,cpr11
                                                  msbuh.p cpr10,cpr14,cpr10; mnop1.p; msbuh cpr11,cpr15,cpr11
                                                  msbuh.p cpr10,cpr16,cpr10; mnop1.p; msbuh cpr11,cpr17,cpr11
                                                  msrl.p  cpr10,7,cpr10;     mnop1.p; msrl  cpr11,7,cpr11
                                                  mh2bw   cpr11,cpr10,cpr0
         stfi.p  fr0,@(gr5,0); addi.p gr5,#4,gr5; bra unsharp_loop
```

Chapter 12

DTU機構

コア間転送機構（DTU）は，DDR，DSM，CSM 間のデータ転送を行うことができる．転送の際に使用する制御レジスタおよび制御ブロックの内容を図 12.1 に示す．DTU には，送信および受信の際に各々使用するアドレシング機能として，直接，間接，ストライド，２重ストライドが装備されている．

論理アドレスの解釈は，DTU を起動する実コアのアドレス変換機構に従う．複数の制御ブロックが連結されている場合でも，各々に記載される論理アドレスの解釈は DTU 起動時の DDR-SAT，DSM-SAT，CSM-SAT の内容に従うことに注意が必要である．なお，アドレス変換の保持は，DTU 起動時の PEV.DDRSAT，PEV.DDRPID，PEV.DSMSAT，PEV.DSMSPC，PEV.DSMPSZ，PEV.CSMSAT，PEV.CSMSPC，PEV.CSMPSZ の内部保持により行われるため，DTU の動作が終結するまでの間，DDRPID，DSMSPC，CSMSPC に対応する各アドレス変換対を変更してはならない．DDR については DDR 領域内アドレスから得られる物理 DDR アドレスを使用する．物理 DDR アドレス空間は論理的に透過な L2 キャッシュにより高速化されており，DDR の内容と L2 キャッシュの内容はハードウェアにより整合性が維持される（L1 キャッシュの内容も同様）．DSM については実 DSM 予約空間内アドレスから得られる物理 DSM アドレスを使用する．物理 DSM は各物理コアに括り付けられており，L2 キャッシュは経由しないが各コアの L1 キャッシュは有効である．各コア毎の DSM の内容と L1 キャッシュの内容はハードウェアにより整合性が維持される．

論理メモリ空間にあらかじめ格納した制御ブロックの先頭アドレスを DTU.BLKTOP に書き込み，DTU.CONTROL に対して動作開始指示を書き込むことにより，制御ブロックの内容に基づくデータ転送動作が開始される．各制御ブロックには，当該ブロックの内容に基づくデータ転送を開始するための起動トリガを指示する領域が設けられており，CPU や DTU による書き込みを検出して動作を開始させることができる．また，各制御ブロックには，データ転送終了時の動作として，割り込み，制御レジスタへの表示，特定アドレスへの書き込みなどの機能を指定することができる．特定アドレスへの書き込みを行うことにより，複数の DTU が CPU とは独立して自律的にデータ転送を行うことが可能である．

関連制御レジスタ	保護	値の意味
DTU.CONTRL	非特	**0:**転送開始 **1:**停止指示
DTU.BLKTOP	非特	**DSM**内相対制御ブロック先頭アドレス
DTU.CBLKAD	非特	処理中制御ブロックアドレス（表示）
DTU.CBLKST	非特	処理中制御ブロック状況（表示）

→

DTB.NEXTAD	次の**DSM**内相対制御ブロック先頭アドレス
DTB.TRIGTY	当該ブロック起動トリガ **0:**無し **1:**有り
DTB.TRIGAD	当該ブロック起動トリガ（監視アドレス）
DTB.TRIMSK	当該ブロック起動トリガ（マスク値）
DTB.TRIVAL	当該ブロック起動トリガ（期待値または任意値）
DTB.TERMTY	当該制御ブロック転送終了時動作指示 **0:**無し **1:**割り込み **2:**制御レジスタ表示 **3:**特定アドレスへの書き込み
DTB.TERMAD	当該ブロック完了時動作（書き込みアドレス）
DTB.TERMSK	当該ブロック完了時動作（マスク値）
DTB.TERVAL	当該ブロック完了時動作（書き込み値）
DTB.SNDTYP	送信アドレス方式（直接/間接/ストライド/2 重など）
DTB.SNDTOP	送信先頭論理アドレス **(DDR/DSM/CSM)**
DTB.SNDIND	インデックスリスト論理アドレス **(DDR/DSM/CSM)**
DTB.SXSIZE	送信要素間距離 1 ＋要素数
DTB.SYSIZE	送信要素間距離 2 ＋要素数（二次元配列を想定）
DTB.RCVTYP	受信アドレス方式（直接/間接/ストライド/2 重など）
DTB.RCVTOP	受信先頭論理アドレス **(DDR/DSM/CSM)**
DTB.RCVIND	インデックスリスト論理アドレス **(DDR/DSM/CSM)**
DTB.RXSIZE	受信要素間距離 1 ＋要素数
DTB.RYSIZE	受信要素間距離 2 ＋要素数（二次元配列を想定）

→

DTB.NEXTAD	**00000000** （**NULL**により終端）
DTB.TERMTY	当該制御ブロック転送終了時動作指示 **0:**無し **1:**割り込み **2:**制御レジスタ表示 **3:**特定アドレスへの書き込み
:	

図 12.1: DTU 制御レジスタと制御ブロック

Chapter 13

バリア同期機構

プログラム実行開始後，あらかじめ，コアグループに属する 1 つまたは複数の実コアを指定することによりサブコアグループを定義した上で，プログラム実行中の任意の時点で同一プロセス ID に属するサブコアグループ全体の同期を確保する機構である．同期状態は 0 同期状態と 1 同期状態からなり，各実コアが現在の同期状態を認識し，他の全コアに対して 0 同期または 1 同期のいずれかを指定して次の同期点到達を放送することにより全体が同期する．使用する制御レジスタの内容を図 13.1 に示す．BAR.CORSPC および BAR.CORMSK によりバリア空間およびバリアマスク（サブコアグループ）を定義した上で BAR.CONTRL により同期要求を発行する．同期要求の送信先がサブコアグループのみか全コアとなるかはモデル依存である．BAR.STATUS には，BAR.CORSPC および BAR.CORMSK に基づく同期状態が表示される．

関連制御レジスタ	保護	値の意味
PEV.CORMAP	特権	**0:**コア番号変換無効, **1:**コア番号変換有効
PEV.CORGRP[0..63]	特権	実コア番号に対応させる物理コア番号 **0-63:**実コア有効 **255:**実コア無効
BAR.CORSPC	非特	バリア空間 **0:space#0, 1:space#1**
BAR.CORMSK	非特	実コア番号バリアマスク（**64bit**） **0:**同期対象外，**1:**同期対象
BAR.CONTRL	非特	同期要求放送 **0:**０同期要求 **1:**１同期要求
BAR.STATUS	非特	同期状態表示 **00:**非同期状態 **01:**未定義 **10:**０同期状態 **11:**１同期状態

図 13.1: BAR 制御レジスタ

Chapter 14

入出力機構

入出力は、すべてメモリにマップされたレジスタにより制御される。従って、入出力のための特別なアーキテクチャは存在しない。

Chapter 15

電力制御機構

Chapter 16

性能評価モデル

本書の最終目的は，性能・回路規模・電力の観点から，演算器アレイアクセラレータSAPP と，アクセラレータを搭載しない一般的なメニコアを定量的に比較することにある．このための準備として，本章では，SAPP の性能評価モデルについて説明する．

16.1 C-RTL モデル

一般に性能評価には，実用アプリケーションを走行させた際のサイクル数を測定する必要がある．厳密な測定には，少なくとも Verilog 等のハードウェア記述言語を用いてマイクロアーキテクチャを実装し，Verilog シミュレータ等によりクロックサイクル数を計測しなければならない．しかし，Verilog シミュレータ等は実行速度が極めて遅く，パラメタを変化させながら実用アプリケーションを何度も走行させることは困難である．一方，C 言語を用いて Verilog と同レベルの RTL モデルを記述することにより，数百倍の速度にてサイクルシミュレーションを行うことができる．以下，本モデルを C-RTL モデルと呼び，C-RTL モデルをコンパイルして得られる実行形式ファイルを RTL シミュレータと呼ぶことにする．

C-RTL モデルは，SAPP 全体を主要な回路ブロックに分割し，各回路ブロックをパイプラインレジスタにより終端させた構成に対してクロックを順次印加するモデルである．Verilog モデルと異なり，パイプラインレジスタへのクロック印加順序は回路ブロックのシミュレーション順になる．前段から後段に向かうデータ流全体の同期動作に対応するため，一般に，パイプラインの後段から前段に向けて順にクロックを印加する．ただし，このような単純な方法では，後段から前段に向かうバイパスを正確に模擬することはできない．バイパスに関しては，1 サイクル前の後段パイプラインレジスタの状態を保存しておき，前段に向かわせる工夫が必要である．このような構成により，Verilog により記述したハードウェアと同等のタイミングモデルを構築しつつ，高速な RTL シミュレータを実現することができる．以下，図 2.4 を用いて各回路ブロックの説明を行う．

pipe_frvia Program Counter(PC) を更新する回路ブロックである．pipe_frvdecode からの分岐予測情報，および，pipe_write からの条件分岐先情報により分岐命令に対応する．

pipe_frvif 命令キャッシュおよび分岐予測表の読み出しを行う回路ブロックである．

pipe_frvdecode 命令デコードを行う回路ブロックである．リターンアドレススタックを有しており，非アレイ動作中に LR が関与する命令を検出した場合，リターンアドレススタックの参照/更新も行う．アレイ動作中は，SAPP 初段におけるデコード結果を用いて，10.2.3 節「演算器ネットワーク設定のための命令デコード」において説明した手順により次段以降の演算器ネットワークを順次設定する．なお，pipe_frvia から pipe_frvdecode までの回路ブロックは，DCPL 命令を起点として無条件後方分岐命令に至る命令列を検出した後，非アレイ動作に復帰するまでの間，動作を停止することが可能である．メモリから構成される命令キャッシュおよび分岐予測表はスリープモード（後述の電力パラメタ POWER_F1_ACTIVE/POWER_F1_INACTIVE に対応）へ，その他の回路は Power-OFF に相当する状態（後述の POWER_IA_IF_DECODE および POWER_SELRD に対応）へ各々移行する．また，次段以降の演算器ネットワーク設定に関わる伝搬情報生成回路は，DCPL 命令を検出してから無

条件後方分岐命令を検出するまでの間以外はPower-OFF に相当する状態（後述の POWER_SELRD に対応）へ移行する．

pipe_sel_read SAPP 初段では，レジスタファイルから演算器入力ラッチに対する読み出しを行う．DCPL 命令を起点として無条件後方分岐命令に至る命令列を検出した後，非アレイ動作に復帰するまでの間，動作を停止することが可能である．すなわち，レジスタファイルはスリープモード（後述の POWER_REGFILE_INACTIVE に対応）へ，その他の回路は Power-OFF に相当する状態（後述の POWER_PREG*に対応）へ各々移行する．一方，次段以降では，前述の演算器ネットワーク設定に従い，多入力セレクタを経由して，前々段の演算器入力ラッチまたは前段の演算器出力ラッチから演算器入力ラッチに対する読み出しを行う．すなわち，演算器入力ラッチ毎に，アレイ動作に関与しないセレクタおよびラッチはPower-OFF に相当する状態（後述の POWER_PREG*に対応）のままとなる．なお，演算器入力ラッチはレジスタ値の段間伝搬機構としても使用するため，演算器入力ラッチと演算器本体の Power-OFF は必ずしも連動しない．

pipe_exec_brc SAPP 初段では，入力ラッチの情報を元に，条件判定および分岐先アドレス計算を行う．次段以降では，アレイ動作終結のための条件判定のみが必要であり，分岐先アドレス計算を行う必要はない．演算中の電力は，後述の POWER_EXEC_BRC に対応する．

pipe_exec_me1-4 入力ラッチの情報を元に，浮動少数点演算またはメディア演算を行う．演算中の電力は，後述の POWER_EXEC_ME1-4 に対応する．

pipe_exec_al2-4 入力ラッチの情報を元に，ALU/MULTIPLY(部分積)/SHIFT 演算を行う．演算中の電力は，後述の POWER_EXEC_AL2-4 に対応する．

pipe_exec_eag 入力ラッチの情報を元に，LD/ST のためのアドレス計算を行う．演算中の電力は，後述の POWER_EXEC_EAG に対応する．

pipe_exec_op1 pipe_exec_eag からのアドレス情報を元に，L1 キャッシュを参照し，また，各段の L0 キャッシュへの書き込みに必要な段間伝搬レジスタ（図 2.4 における oop4）の更新を行う．サブコア毎の L1 キャッシュの電力は，後述の POWER_L1 に対応する．また，LD/ST が写像された最終段まで oop4 が伝搬される．oop4 毎の電力は，後述の POWER_OOP4 が対応する．

pipe_exec_op0 pipe_exec_eag からのアドレス情報を元に，L0 キャッシュから oop1 への読み出しまたはストアバッファへの書き込みを行い，また，oop4 から L0 キャッシュへの書き込みを行う．LD/ST が写像された段のみ動作する L0 キャッシュの電力は，後述の POWER_L0 に対応する．

表 16.1: 性能パラメタ

パラメタ名	設定値 (cycle)	説明
Q_FRET	8	return-address stack
STBF	4	store buffer
F1WAYS	4	F1 cache-ways 4
F1WSIZE	4096	F1 way-size 4KB
F1LINE	64	Line–size 64B
L0WSIZE	64	L0 way-size 64B
L0LINE	16	Line–size 16B
L1WAYS	4	L1 cache-ways 4
L1WSIZE	4096	L1 way-size 4KB
L1LINE	64	Line–size 64B
L2SIZE	16M	L2 total-size 16MB(direct-map)
L2LINE	64	Line–size 64B
F1DELAY	8	F1 miss-penalty 8cycle
L1DELAY	8	L1 miss-penalty 8cycle
L2DELAY	8	L2 miss-penalty 8cycle
SUB_ARRAY	9	Depth of subcore
SUBCORE_NUM	4	Number of subcore per PE
L2BANK_INTERLEAVE	L1WSIZE	L2 bank address interleave size
L2BANK_NUM	256	Number of L2 bank
PENUM	4	Number of PE

16.2 性能パラメタ

C-RTL モデルは，表 16.1 に列挙する性能パラメタを変化させての評価が可能である．

Chapter 17

回路遅延検証モデル

本章では，C-RTL モデルにおけるパイプライン構成の妥当性を検証するための回路遅延検証モデルについて説明する．

17.1 FPGA モデル

C-RTL モデルは，プロセッサ機能，外部キャッシュ機能，主記憶機能，入出力機能を有し，十分な速度で実用アプリケーションを走行させて性能評価を行うことができる．しかし，C-RTL モデルのパイプライン構成および各回路ブロックの構成が，タイミングおよび回路遅延の観点から LSI として実現可能であることを検証する必要がある．モデルの妥当性を裏付けるためには，プロセッサ機能およびその他の機能について C-RTL モデルと同等のハードウェアを実現し，各回路ブロックの動作タイミングを検証しつつ，合成配置配線結果によって明らかになった遅延の偏り等をフィードバックすることが重要である．また，FPGA 化により，C-RTL モデルのさらに 50 倍程度の実用アプリケーション動作速度を得られることから，より多くの試験プログラムを走行させることができる．以下，本モデルを FPGA モデルと呼び，実用アプリケーションを走行可能なシステム全体を GP5V330MF システムと呼ぶことにする．

17.2 GP5V330MF システムの構成

GP5V330MF システムは，PC/AT 互換機（PCI-HOST）の PCI-Express スロットに，FPGA（XC5V330T）および SRAM を搭載した PCI ボード（GP5V330MF）を接続し，プロセッサ機能を FPGA 上に，外部キャッシュ機能をオンボード SRAM 上に，また，主記憶機能および入出力機能を PCI-HOST 上に各々実現し，全体として実用アプリケーションを走行可能とする装置である．また，複数の GP5V330MF システムを相互接続することにより，図 2.9 に示すマルチコア構成の評価モデルを構築できる．このような装置は市販されておらず，独自仕様のインタフェースを用いることから，以下，各部物理インタフェースおよび論理インタフェースを中心に説明を行う．

17.2.1 FPGA 外部の物理構成

GP5V330MF の構成

まず，図 17.1 および図 17.2 に示す GP5V330MF 単体の外部仕様を列挙する．

- PCI-Express2.0 スロットに対応（データ転送速度は 10Gbps）
- PCI-Express2.0 デバイスドライバは，FreeBSD7.2R、32bit 版 Linux カーネル 2.6.18，64bit 版 Linux カーネル 2.6.9 に対応
- 512Mbit の DDR2-SDRAM を搭載
- XC5V330T を 2 基搭載（デイジーチェインによるコンフィグレーション）
- Ethernet 経由またはフラッシュメモリによるコンフィグレーション

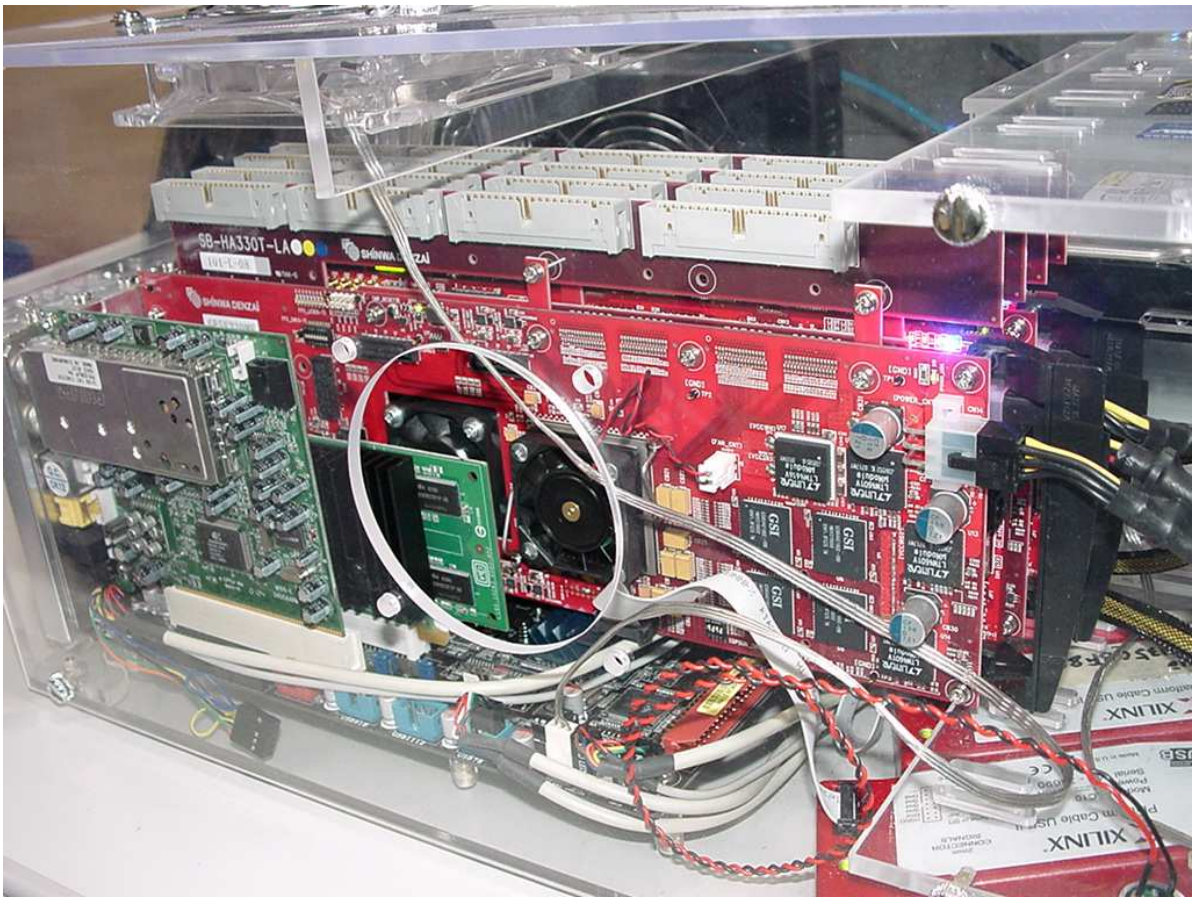


図 17.1: GP5V330MF の外観

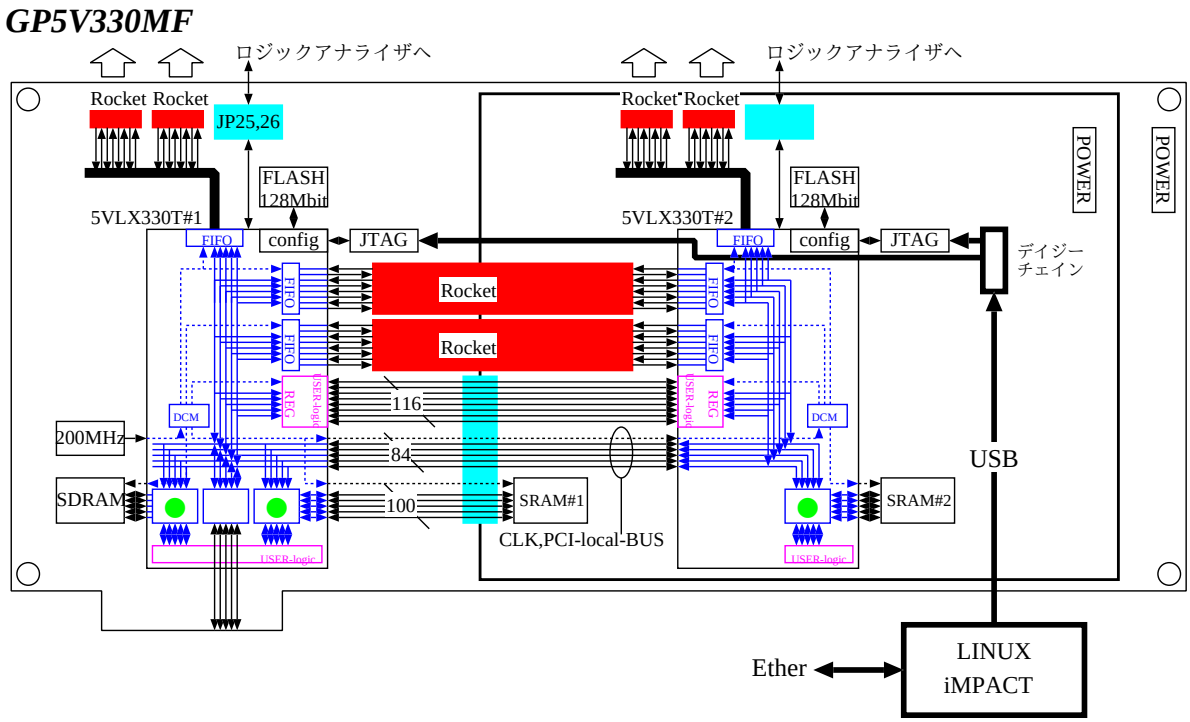


図 17.2: GP5V330MF の構成

- XC5V330T への供給クロックは 200MHz．XC5V330T 内部の DCM モジュールによる逓倍または分周クロックをボード上の各ユニットに対して独立に供給

表 17.1: ZBT-SSRAM 側信号の制御

信号名	説明
A(入力)	SRAMA/B_A19-0 に対応（アドレス信号）
BA-BD(入力)	SRAMA/B_BW3-0 に対応（バイト毎書き込み，Active-LOW）
W(入力)	SRAMA/B_RW に対応（書き込みイネーブル，Active-LOW）
E1(入力)	SRAM_CE7-0 に対応（チップイネーブル，Active-LOW）
G(入力)	SRAMA/B_OE に対応（出力イネーブル，常時 LOW）
ADV(入力)	LOW 固定（常時アドレスセット可）
CKE(入力)	LOW 固定（クロック入力バッファ常時イネーブル）
ZZ(入力)	LOW 固定（パワーダウン未使用）
FT(入力)	N.C.（Pipeline モード）
LBO(入力)	SRAMA/B_LBO に対応（リニアバースト，常時 LOW）
CK(入力)	XC5V330T の XXX に対応（クロック）
DQA-DQD(入出力)	SRAMA/B_D35-0 に対応

- XC5V330T 毎にロジックアナライザによる 64 チャネルの波形観測が可能
- XC5V330T 毎に 32MB の ZBT-SSRAM を搭載
- XC5V330T 相互接続手段として 1Gbps の Rocket-I/O を各 16 組（送受信ペア）装備

各 XC5V330T の外部ピン接続は以下の通りである．

- **ロジックアナライザ用コネクタ (CN5,CN6)**
  AM2（CN5.CLK0）…【未定】
- **デジタルオシロ接続用タップ**
  AM2（TP1）…【未定】
- **DDR2-SDRAM**
  AM2（TP1）…【未定】
- **ZBT-SSRAM(GS8320Z36)**
  AM2（TP1）…【未定】
- **Local-BUS**
  AM2（TP1）…【未定】
- **Rocket-I/O**
  AM2（TP1）…【未定】
- **隣接 XC5V330T**
  AM2（TP1）…【未定】

XC5V330T と DDR2-SDRAM の物理インタフェース

FPGA モデルの実装には使用しない．

XC5V330T と ZBT-SSRAM の物理インタフェース

XC5V330T 毎に 36bit × 1M ワードの ZBT-SSRAM を計 8 個搭載しており，36bit × 4M ワード× 2 組の外部メモリとして使用できる．表 17.1 に ZBT-SSRAM 側信号の概要を示す．

PCI ブリッジ Local-BUS の物理インタフェース

PCI インタフェースと，全ての XC5V330T，DDR2-SDRAM，ZBT-SSRAM を接続する汎用バスである．これらの資源は，PCI-HOST のメモリ空間上にマップされ，直接参照することができる．詳細は GP5V330

表 17.2: PCI ブリッジ Local-BUS インタフェース信号の概要

信号名	説明
L_RSTX(出力)	リセット信号
L_DRQX(出力)	ブリッジからの転送要求信号
L_DACKX(入力)	転送要求に対する応答（LOW の場合 Local 側がスレーブ，HIGH の場合 Local 側がマスタ）
L_ADSX(入出力)	転送開始（バスマスタが出力）
L_WEX(入出力)	LOW は WRITE，HIGH は READ に対応
L_BSTMX(入出力)	LOW はバーストモードに対応
L_RDYOX(出力)	ブリッジ側転送準備 OK
L_RDYIX(入力)	Local 側転送準備 Ok
L_AD(入出力)	ブリッジ側マスタの時はアドレス＋データ，Local 側マスタの時はアドレス＋バイト数＋データ
L_BEX(入出力)	バイトイネーブル
L_INTX(入力)	割り込み信号
L_S64X(入力)	PCI 制御レジスタ選択信号
L_DBSYX(出力)	PCI 書き込み状態の表示（ただし，特に使用しなくてよい）

Local ブリッジ IP コア仕様書を参照のこと．表 17.2 にインタフェース信号の概要，図 17.3 にタイミングチャートを示す．

Local-BUS による ZBT-SSRAM の直接制御

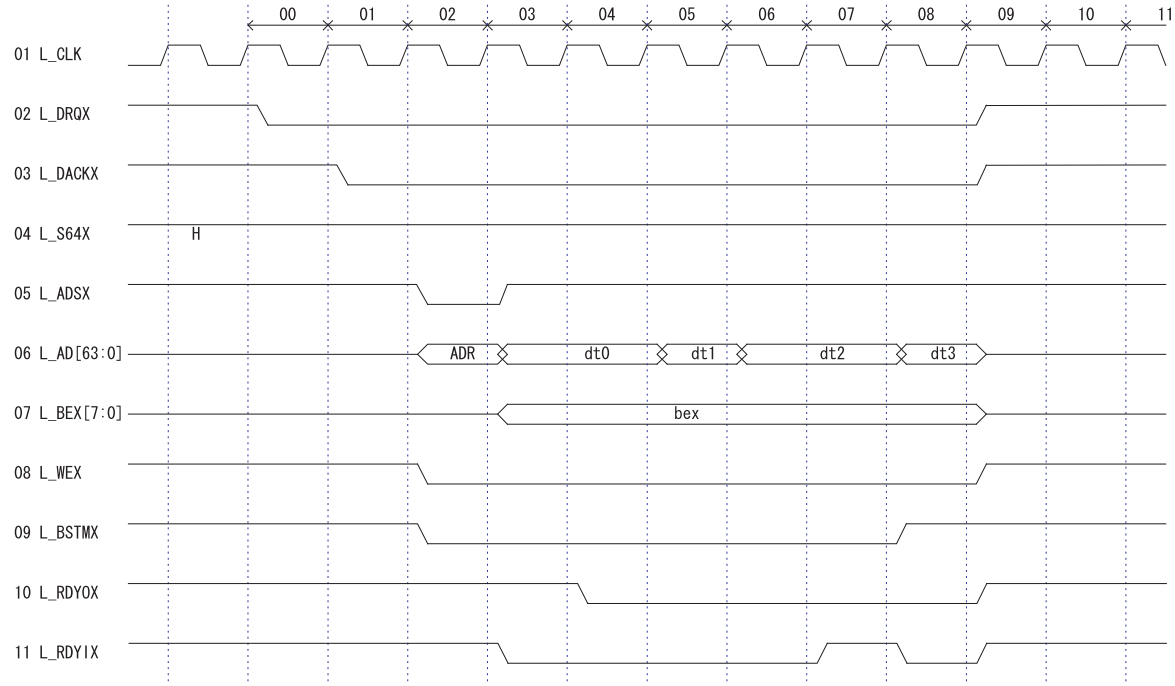
前述の ZBT-SSRAM および Local-BUS インタフェースの組み合わせにより，Local-BUS から SSRAM を直接制御できる．すなわち，PCI-HOST の主記憶空間に ZBT-SSRAM を対応付け，PCI-HOST が ZBT-SSRAM の内容を直接参照/更新することができる．図 17.4 から図 17.7 に，Loca-BUS による ZBT-SSRAM 直接制御のタイミングチャートを示す．

17.2.2 FPGA 内部の物理構成

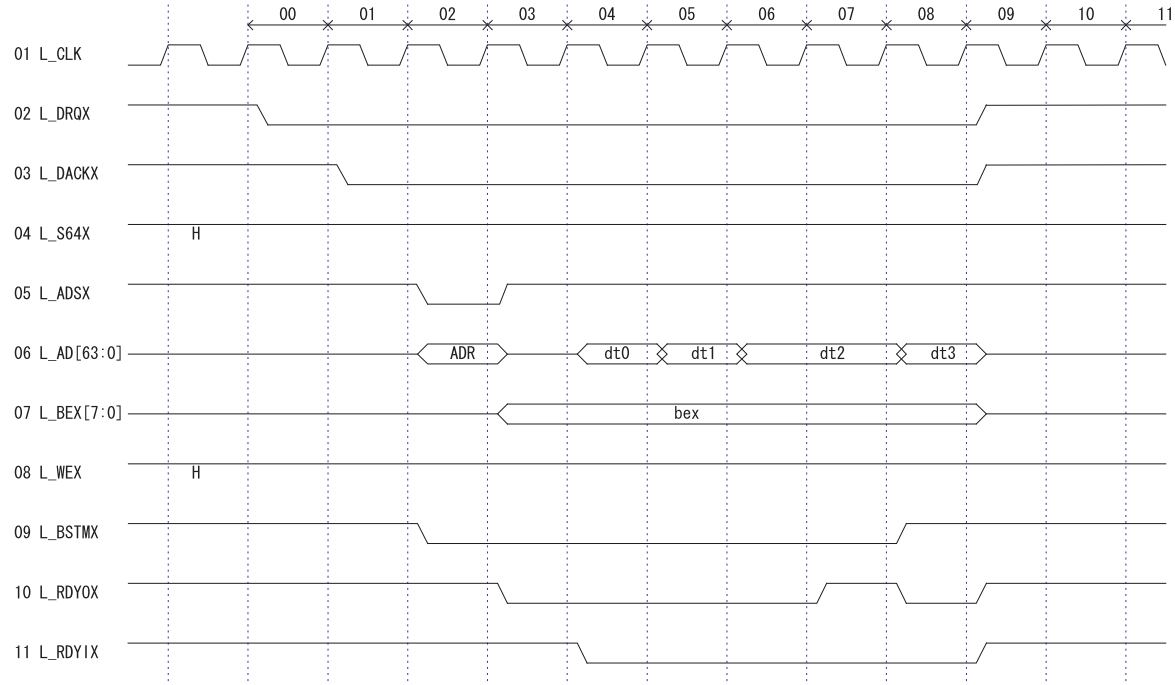
XC5V330T 内に，前述した物理構成と接続する SAPP 機能を搭載することにより，GP5V330MF を SAPP 評価システムとして利用する．SAPP 機能は，大まかに，プロセッサ機能と，XC5V330T 外部の物理構成とのインタフェース機能からなる．各ブロック間インタフェースの概要を図 17.8 に示す．

プロセッサ機能と外部メモリ機能の物理インタフェース

表 17.3 に外部メモリ機能側からみたプロセッサ機能の物理インタフェース，図 17.9 から図 17.12 にタイミングチャートを示す．ビット幅や未使用信号の取扱いに関する差異を除き，表 17.1 に示した ZBT-SSRAM 物理インタフェースと同様である．ただし，ハンドシェイクのための信号 BSYX/XREQ/XGNT/WRDY/RRDY が追加されている．外部メモリへの書き込み時，プロセッサ機能内のメモリ制御部は，BSYX 信号が OFF であることを確認した上で A/B/W/D に有効値を出力し XREQ 信号を ON にする．次に XGNT 信号の ON を観測し XREQ 信号を OFF にする．外部メモリへの書き込み完了と同時に WRDY 信号は ON，BSYX 信号は OFF となる．同様に，外部メモリからの読み出し時，プロセッサ機能内のメモリ制御部は，BSYX 信号が OFF であることを確認した上で A/W に有効値を出力し XREQ 信号を ON にする．次に XGNT 信号の ON を観測し XREQ 信号を OFF にする．読み出し完了と同時に RRDY 信号は ON，BSYX 信号は OFF となり，メモリ制御部は D 上の読み出しデータを取り込む．なお，外部メモリに対する書き込み要求および読み出し要求の単位は，データ信号線幅と同じく 64 ビットである．



(1) PCI ブリッジ⇒ Local-BUS の書き込み



(2) PCI ブリッジ⇒ Local-BUS の読み出し

図 17.3: PCI ブリッジ Local-BUS インタフェースのタイミングチャート

プロセッサ機能と PCI-HOST 機能の物理インタフェース

表 17.4 にプロセッサ機能と PCI-HOST 機能の物理インタフェース，図 17.13 にタイミングチャートを示す．XINT は，PCI-HOST が後述する BAR2 空間先頭ワードへの書き込みを完了し，プロセッサ機能に対して動作を指示する際に ON になる．プロセッサ機能が XINT を検出し XACK を ON にした時点で，XINT は OFF にならなければならない．XREQ はプロセッサ機能が BAR2 空間に対して書き込みまたは読み出しを行う際に ON にする信号線である．

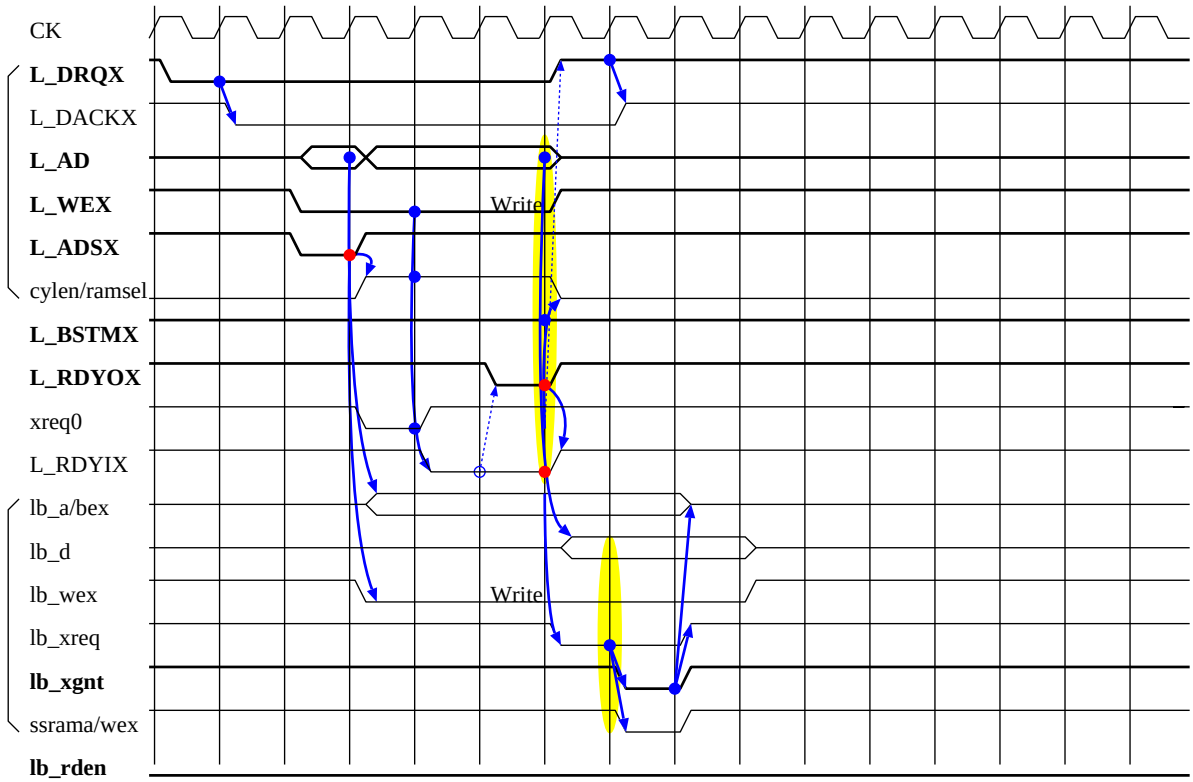


図 17.4: PCI-HOST による SSRAM 直接制御 (WRITE)

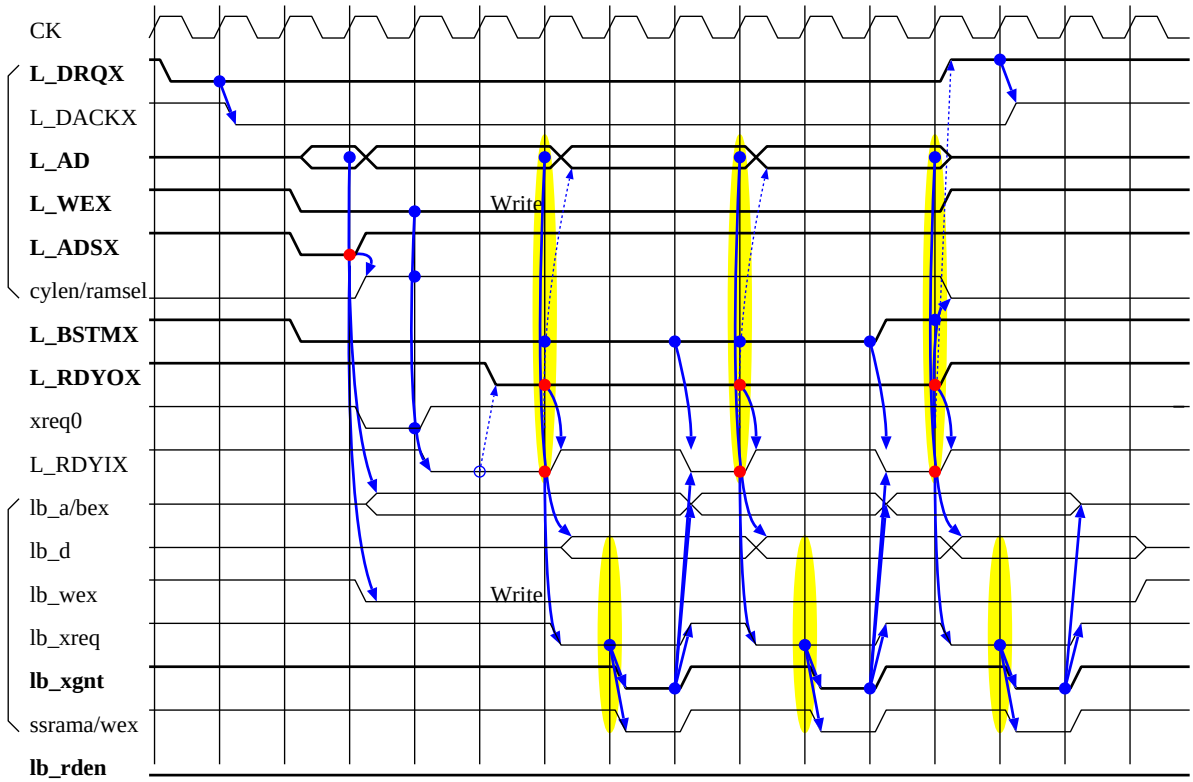


図 17.5: PCI-HOST による SSRAM 直接制御 (WRITE バーストモード)

17.2.3 PCI-HOST に対する論理インタフェース

PCI-HOST と GP5V330MF の論理インタフェースは，PCI バス上のレジスタ空間に写像された GP5V330MF 上の空間群 BAR3,BAR2 である（表 17.5）．ioctl() のオーバヘッドを削減するために，BAR3 の後半アド

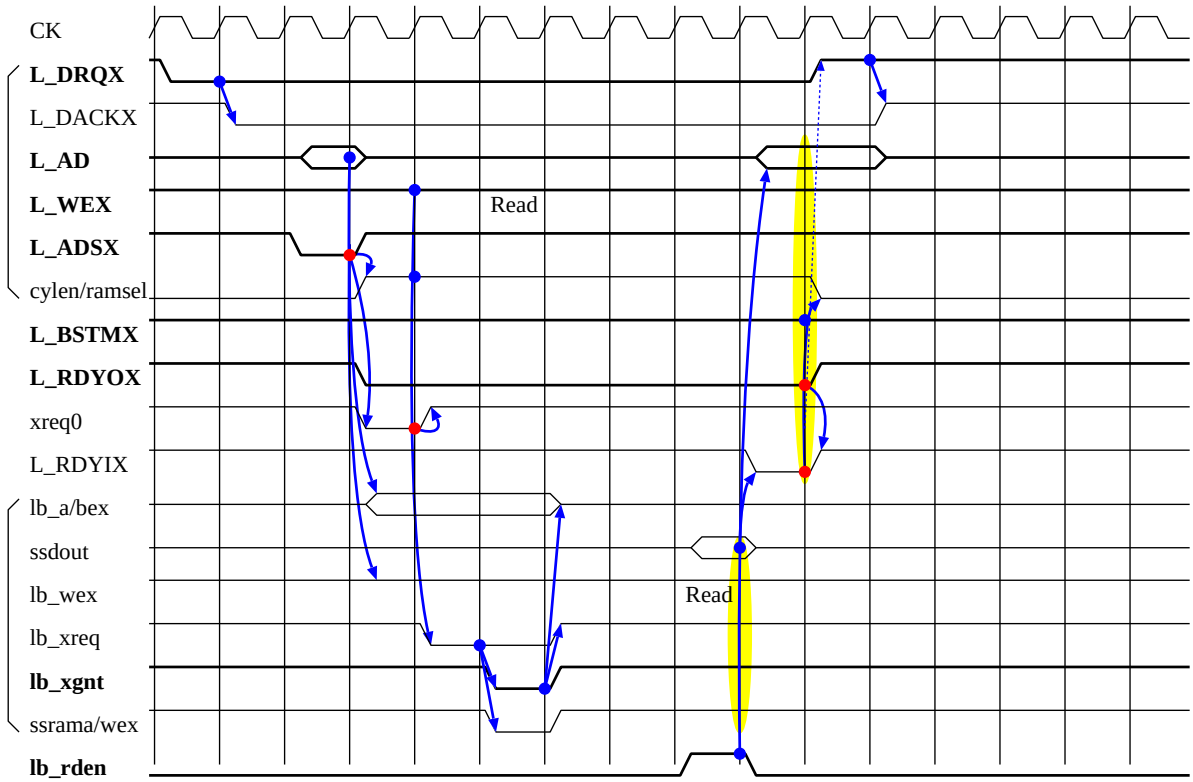


図 17.6: PCI-HOST による SSRAM 直接制御 (READ)

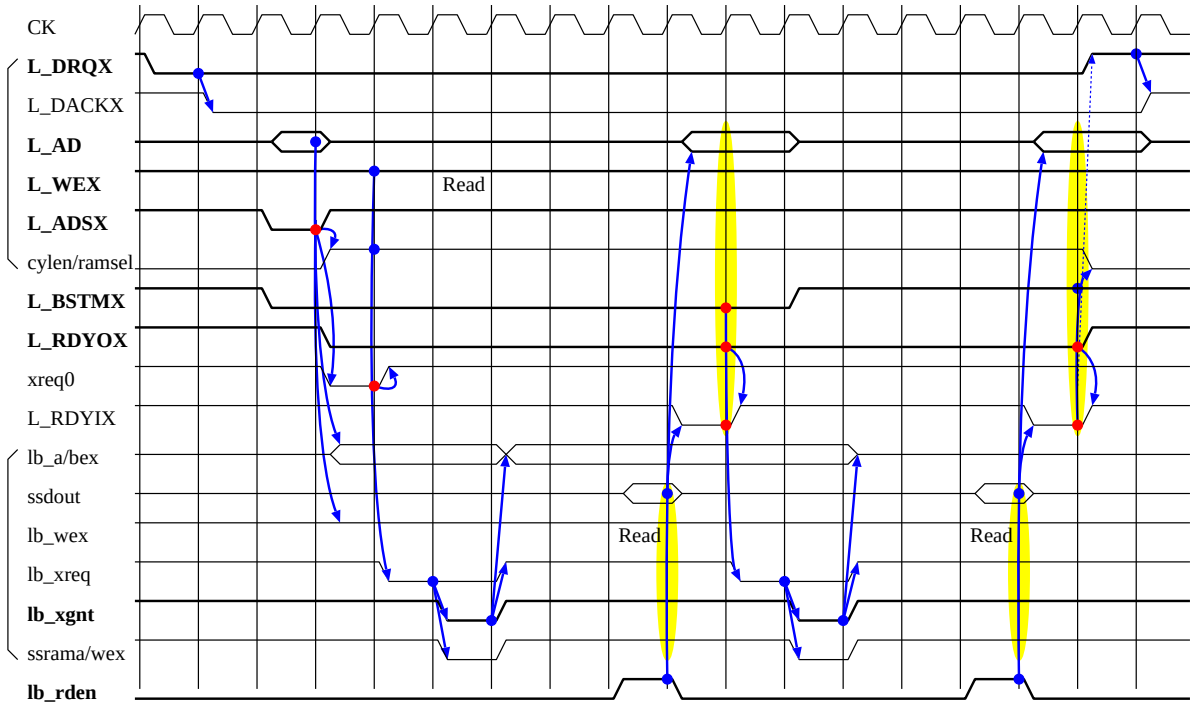


図 17.7: PCI-HOST による SSRAM 直接制御 (READ バーストモード)

レスに BAR2 の機能を割り当て，全てを mmap() により写像して使用してよい．BAR3 空間は外部メモリインタフェース（詳細は第 4 章），BAR2 空間は制御インタフェース（詳細は第 5 章）に各々対応する．PCI-HOST 側デバイスドライバは，付録を参照のこと．

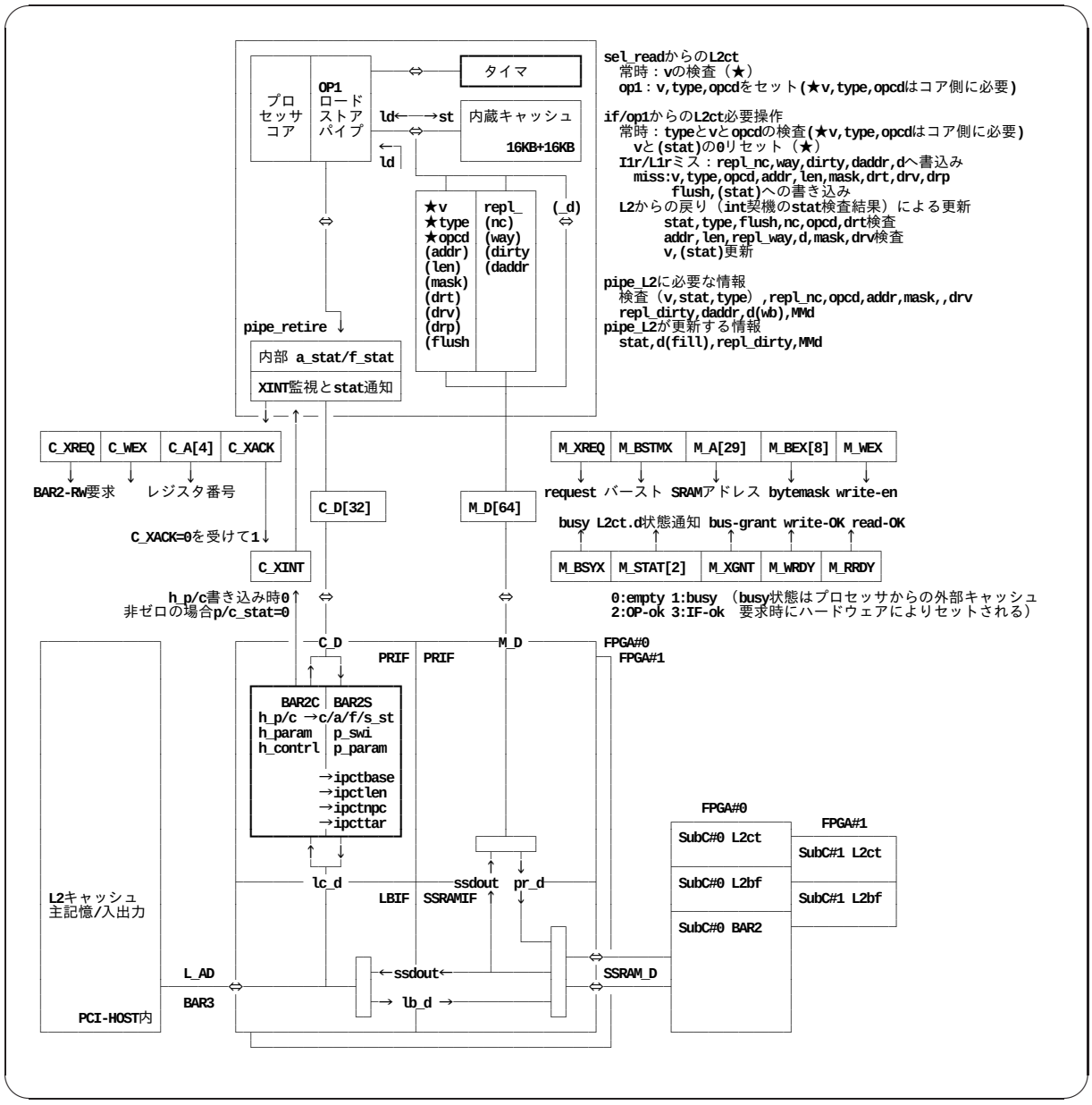


図 17.8: 各ブロック間の物理インタフェース

17.2.4 GP5V330MF システムの全体構造

GP5V330MF システムは単一の FPGA 上に構築することが困難であるため，図 17.14 に示すように，1 台の PC/AT 互換機に，GP5V330MF を 2 基搭載する PCI ボードを 2 枚搭載し，Rocket-I/O により 4 基の FPGA をリング接続した基本構成を利用する．各 PE は 4 基の FPGA を用いて実装される．複数 PE を実現するためには，複数台の PC/AT 互換機を Ethernet により接続する．全体として 1 つのシミュレーション環境を実現するために，主記憶および外部キャッシュを集中管理する SAPP 制御プログラムを PCI-HOST#0 上で動作させ，その他の PCI-HOST にはスレーブプログラムを動作させる．すなわち，実現可能な PE 数は PCI-HOST 数によって決まる．図 17.15 にブロック間接続の詳細を示す．

17.3 GP5V330MF システムの記憶

本章では，主記憶アドレスに関連する記憶に関して説明する．アドレス計算は 32 ビットでラップ・アラウンドする．2,4,8 バイトをそれぞれ半語，語，倍長語と呼ぶ．半語，語，倍長語は任意のアドレスから始

表 17.3: プロセッサ-外部メモリインタフェース信号（110 本）の概要

信号名	説明
M_XREQ(入力)	外部メモリ参照要求信号，Active-LOW
M_XGNT(出力)	要求受付完了信号，Active-LOW
M_WEX(入力)	書き込みイネーブル，Active-LOW
M_BSTMX(入力)	LOW 時はバーストモード
M_WRDY(出力)	書込完了信号，Active-HIGH．本信号の ON をもってプロセッサ機能は参照要求を消去できる
M_RRDY(出力)	読出データ valid 信号，Active-HIGH．本信号の ON をもってプロセッサ機能は参照要求を消去できる
M_A(入力)	アドレス，A31-3
M_BEX(入力)	バイト毎書き込みフラグ，Active-LOW，B7-0
M_D(入出力)	データ，D63-0
M_BSYX(出力)	動作中を示すビジー信号，Active-LOW
M_STAT(出力)	状態表示信号，STAT1-0（0:empty 2:OP-ok 3:IF-ok）

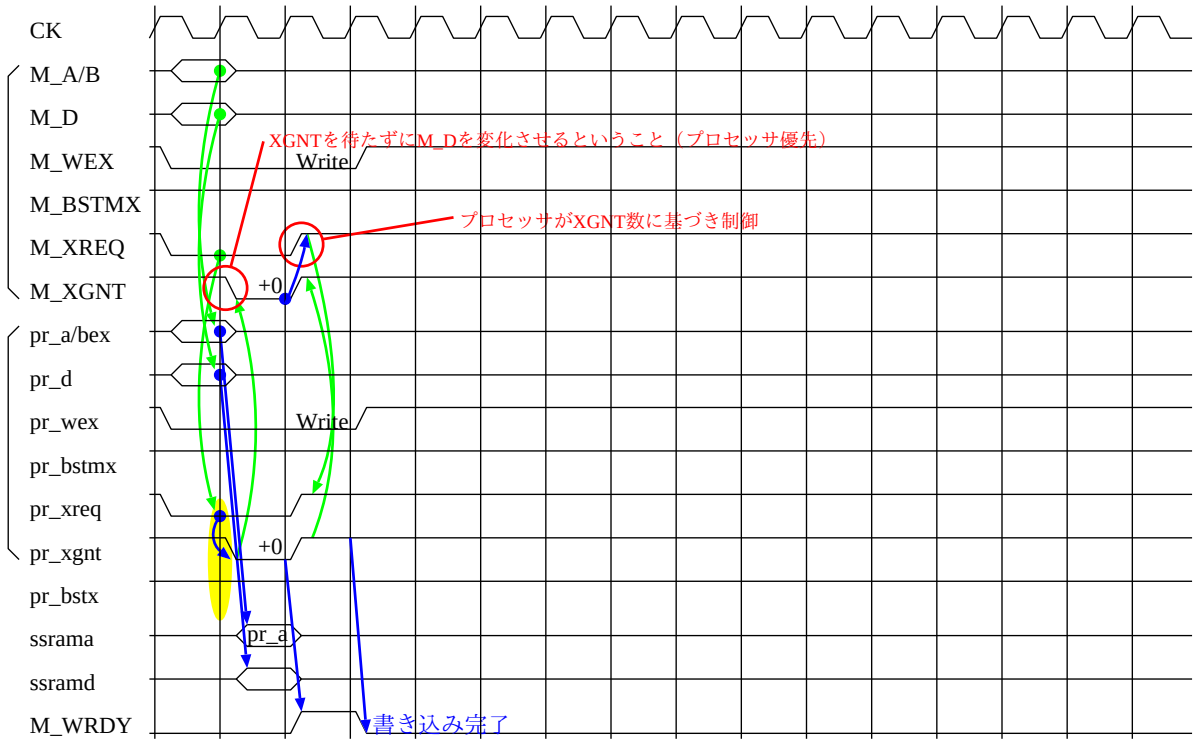


図 17.9: 外部メモリインタフェース信号のタイミングチャート (WRITE)

まるバイト位置に置けるのではなく，決まったアドレスの場所のみに置くことができる．すなわち，0 番地，または，それぞれ 2,4,8 の倍数のアドレスから始まる場所にしか置けない．レジスタの各ビットとの関係はリトルエンディアンに従う．また，アドレスのタイプは実アドレスである．主記憶アドレス空間は，表 17.6 の通りである．このうち，キャッシャブル主記憶アドレス空間の読み出しは，プロセッサ内蔵キャッシュ，外部キャッシュ，PCI-HOST 主記憶装置の優先順に行われる．主記憶アドレス空間への書き込みは，内蔵キャッシュから外部キャッシュへはストアスワップ，外部キャッシュから主記憶へもストアスワップ方式に従う．プロセッサ内蔵キャッシュと外部キャッシュのインタフェースは，表 17.7 に示す ZBT-SSRAM 空間に配置される．なお，この空間は GP5V330MF 毎に存在する．PCI-HOST が複数枚の GP5V330MF を搭載し，各 SubCore が各 FPGA に実装される場合，各空間には SubCore#0 と SubCore#1 に対応する領域のみが存在する．図 17.16 に，記憶階層間のキャッシュライン転送手順の詳細を示す．

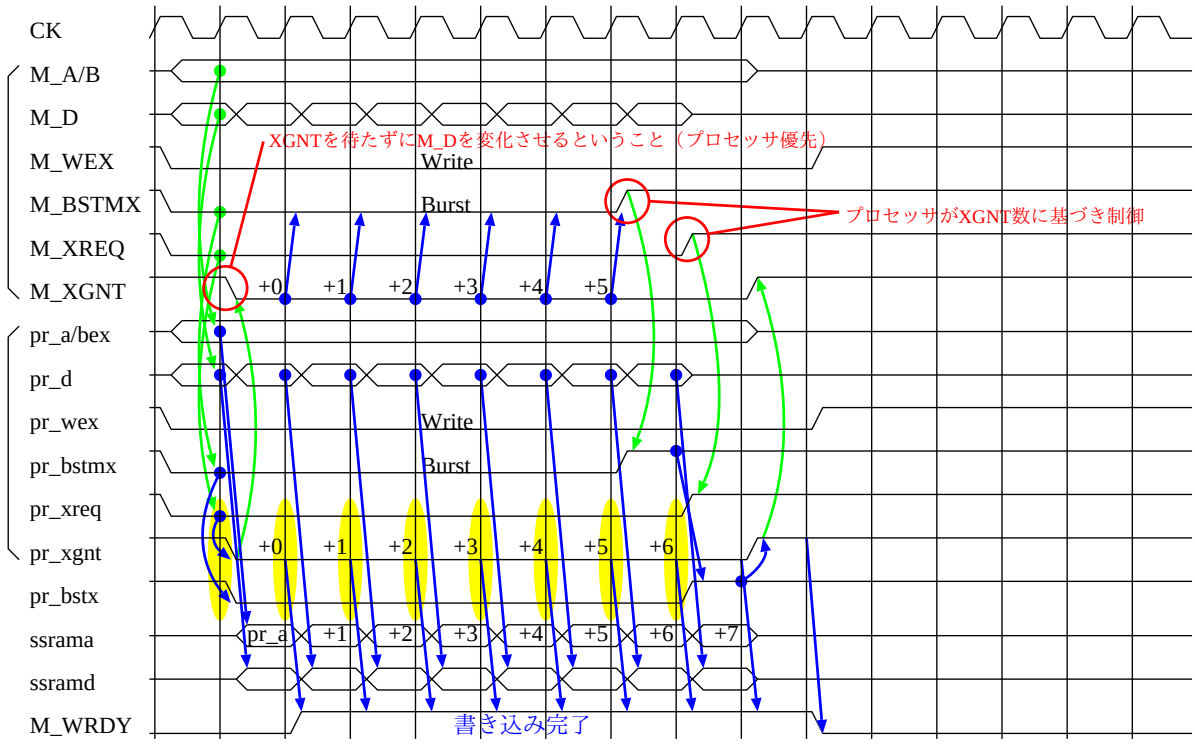


図 17.10: 外部メモリインタフェース信号のタイミングチャート (WRITE バーストモード)

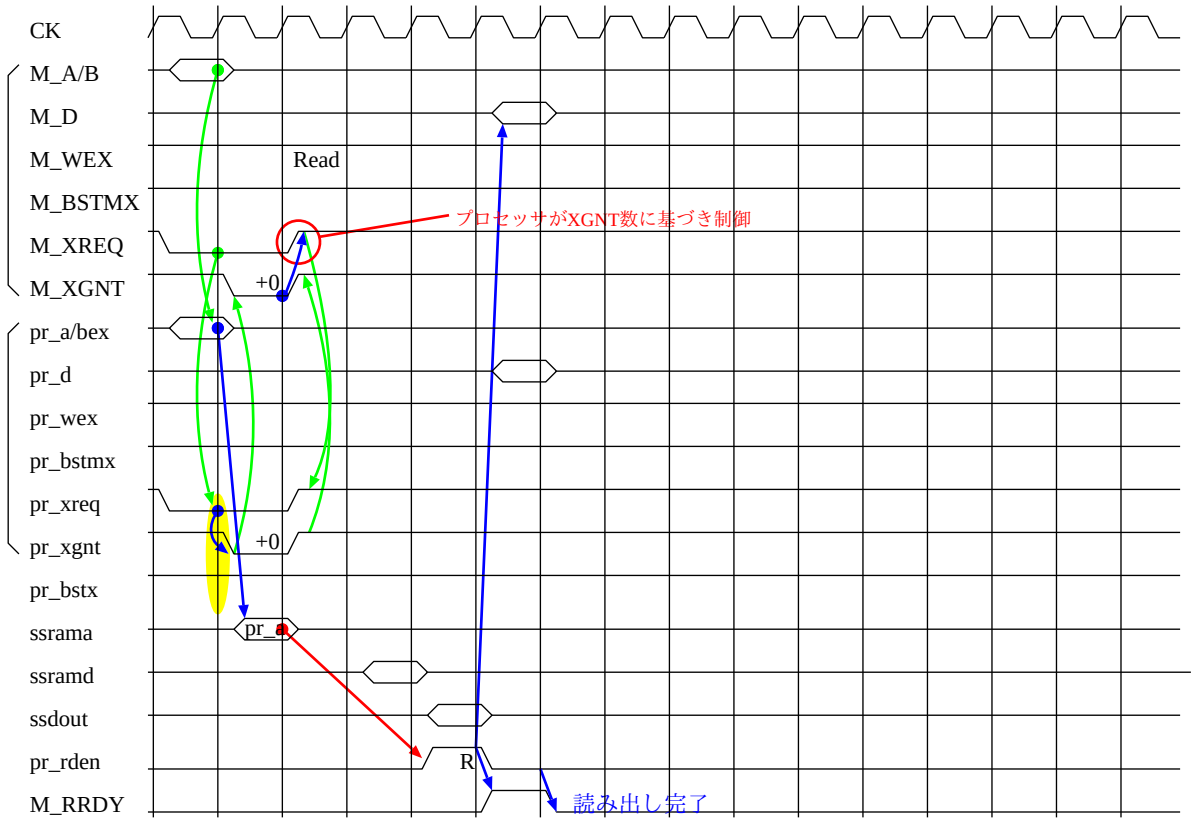


図 17.11: 外部メモリインタフェース信号のタイミングチャート (READ)

17.3.1 内蔵キャッシュ

内蔵命令キャッシュは 4 ウェイ物理アドレスキャッシュ構成，内蔵データキャッシュは，1 ウェイの通常物理アドレスキャッシュにソフトウェア制御可能領域（読み込み専用が 9 ウェイ分）を加えた構成である．

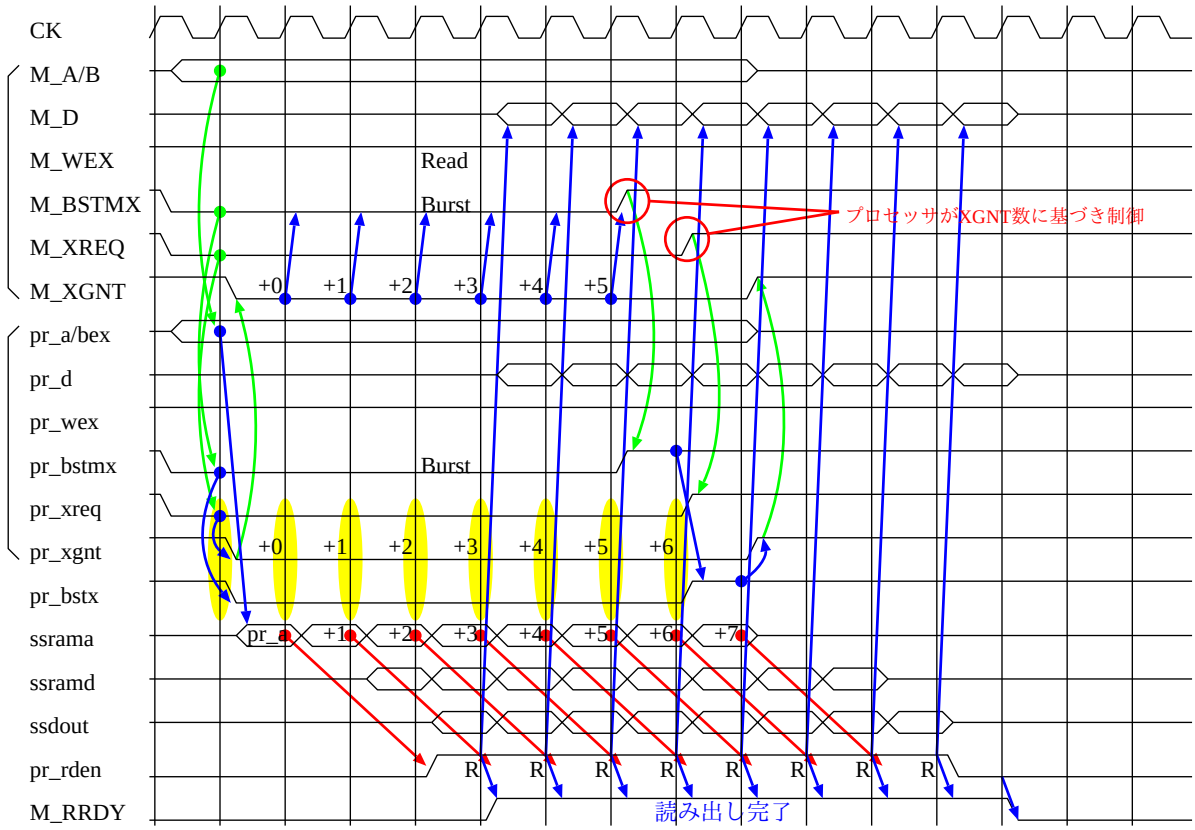


図 17.12: 外部メモリインタフェース信号のタイミングチャート (READ バーストモード)

表 17.4: プロセッサ制御インタフェース信号（40 本）の概要

信号名	説明
C_XINT(出力)	PCI-HOST からのリクエスト信号，Active-LOW
C_XACK(入力)	プロセッサからの INT 受付け信号，Active-LOW
C_XREQ(入力)	プロセッサからのレジスタ参照信号，Active-LOW
C_A(入力)	アドレス，A3-0
C_WEX(入力)	書き込みイネーブル，Active-LOW
C_D(入出力)	データ，D31-0

各ウェイのラインサイズは 64 バイト，容量は 4K バイトである．

内蔵命令キャッシュの読み出しに際してキャッシュミスが検出された場合，最終的に外部キャッシュから転送される有効ラインの格納先となる入れ換え対象ラインが 4 ウェイの中から LRU アルゴリズムに基づいて決定される．有効ラインの読み出し要求が，読み出し主記憶アドレスとともに外部メモリインタフェースにキューイング（L2_ctl.v=3，stat=1，M_STAT=1，type）される．外部キャッシュから到着した有効ラインは一旦外部メモリインタフェース内バッファ（L2_ctl.dline）に格納される．プロセッサ機能は，M_STAT=3 による通知を受けて，内蔵命令キャッシュの入れ換え対象ラインに書き込む．

内蔵データキャッシュは 1 ウェイの通常物理アドレスキャッシュであるため，読み出しに際してキャッシュミスが検出された場合，最終的に外部キャッシュから転送される有効ラインの格納先は一意に決まる．入れ換え対象ラインがダーティラインである場合，入れ換え対象ラインの内容が外部メモリインタフェース内バッファ（L2_ctl.dline）に書き込まれ，外部キャッシュへの追い出しおよび有効ラインの読み出し要求が，追い出し先主記憶アドレス，読み出し主記憶アドレス，書き込み先ウェイ番号（L2_ctl.repl_daddr，L2_ctl.addr，L2_ctl.repl_way）とともに外部メモリインタフェースにキューイング（L2_ctl.v=3，stat=1，M_STAT=1，type）される．一方，入れ換え対象ラインがダーティラインでない場合，有効ラインの読み出し

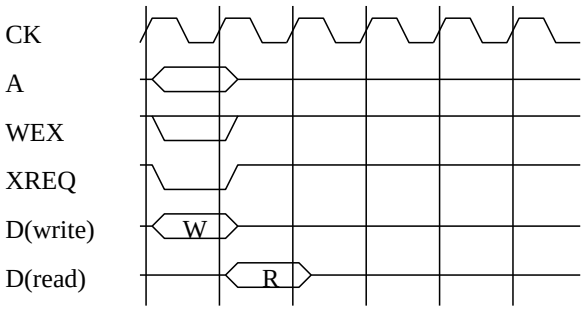


図 17.13: プロセッサ制御インタフェースのタイミングチャート (WRITE/READ)

表 17.5: Local-BUS 上の論理インタフェース

atop(vtophys((vm_offset_t)rman_get_virtual(bus_alloc_resource(dev, SYS_RES_MEMORY, RID#))))		
インタフェース名	RID#	説明
BAR3(0x0-0x3fffff)	PCI_CBMA:0x10	前半 32MB は ZBT-SSRAM 空間，後半 32MB は未使用．プロセッサ機能が外部メモリを参照する際に使用する．
BAR2(0x0-0x1ff)	PCI_CBMA:0x10	PCI-HOST がプロセッサ機能へ動作を指示する際，および，プロセッサ機能から PCI-HOST への例外通知／サービス要求の際に使用する．

要求が，読み出し主記憶アドレスとともに外部メモリインタフェースにキューイング（L2_ctl.v=3，stat=1，M_STAT=1，type）される．外部キャッシュから到着した有効ラインは一旦外部メモリインタフェース内バッファ（L2_ctl.dline）に格納される．プロセッサ機能は，M_STAT=2 による通知を受けて，内蔵キャッシュの入れ換え対象ラインに書き込む．

内蔵データキャッシュへの書き込みに際してキャッシュミスが検出された場合も同様にキューイングされる．外部キャッシュから到着した有効ラインは一旦外部メモリインタフェース内バッファ（L2_ctl.dline）に格納される．プロセッサ機能は，M_STAT=2 による通知を受けて，内蔵キャッシュの入れ換え対象ラインに，ストアデータとともに書き込む．また，入れ換え対象ラインのダーティビットをセットする．

なお，プロセッサ機能が命令フェッチやロード命令を投機実行する場合，外部メモリインタフェースに一旦キューイング（L2_ctl.v=3，stat=1，M_STAT=1，type）された読み出し要求が，プロセッサ機能により消去（L2_ctl.opcd=15 または L2_ctl.type=8 の場合に，L2_ctl.v=0，stat=3 指定により stat=M_STAT）されることがある．この操作によっても，外部キャッシュの動作が停止することは保証されず，後述の動作により M_STAT=2/3 が報告されることがある．プロセッサ機能が，読み出し要求を消去した後に，外部メモリインタフェースに対して新たな要求をキューイングする際には，M_STAT が busy（=1）でないことを確認しなければならない．

17.3.2 外部キャッシュ

命令/データ共用外部キャッシュは，ダイレクトマップの物理アドレスキャッシュ，ラインサイズは 64 バイト，容量は 2M バイトである．

外部メモリインタフェースに読み出し要求がキューイング（L2_ctl.v=3，stat=1，M_STAT=1，type）されている場合，外部キャッシュが検査される．キャッシュヒットの場合，当該ラインの内容が外部メモリインタフェース内バッファ（L2_ctl.dline）に格納され，内蔵キャッシュに対して有効ラインの到着が M_STAT=2/3 により通知される．外部キャッシュミスが検出された場合で，入れ換え対象ライン（ダイレクトマップなので一意に決まる）がさらにダーティラインである場合，入れ換え対象ラインの内容が主記憶インタフェース内バッファ（MM_ctl.dline）に読み出され，主記憶への追い出しおよび有効ラインの読み出し要求が，追い出し先主記憶アドレスおよび読み出し主記憶アドレスとともに主記憶インタフェース（MM_ctl.repl_daddr，MM_ctl.addr）にキューイングされる．一方，入れ換え対象ラインがダーティラインでない場合，有効ライ

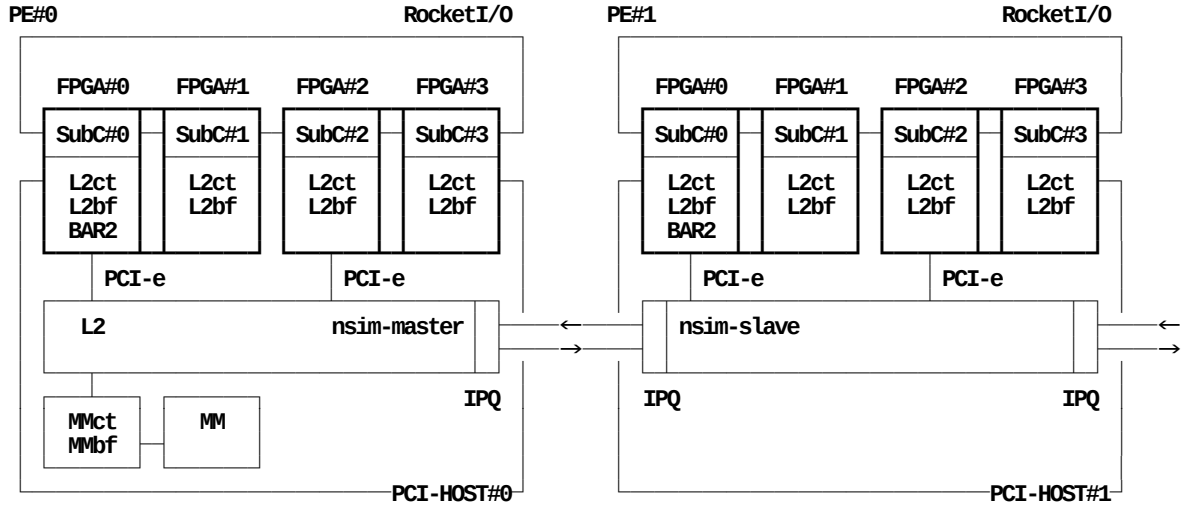


図 17.14: GP5V330MF システムの全体概要

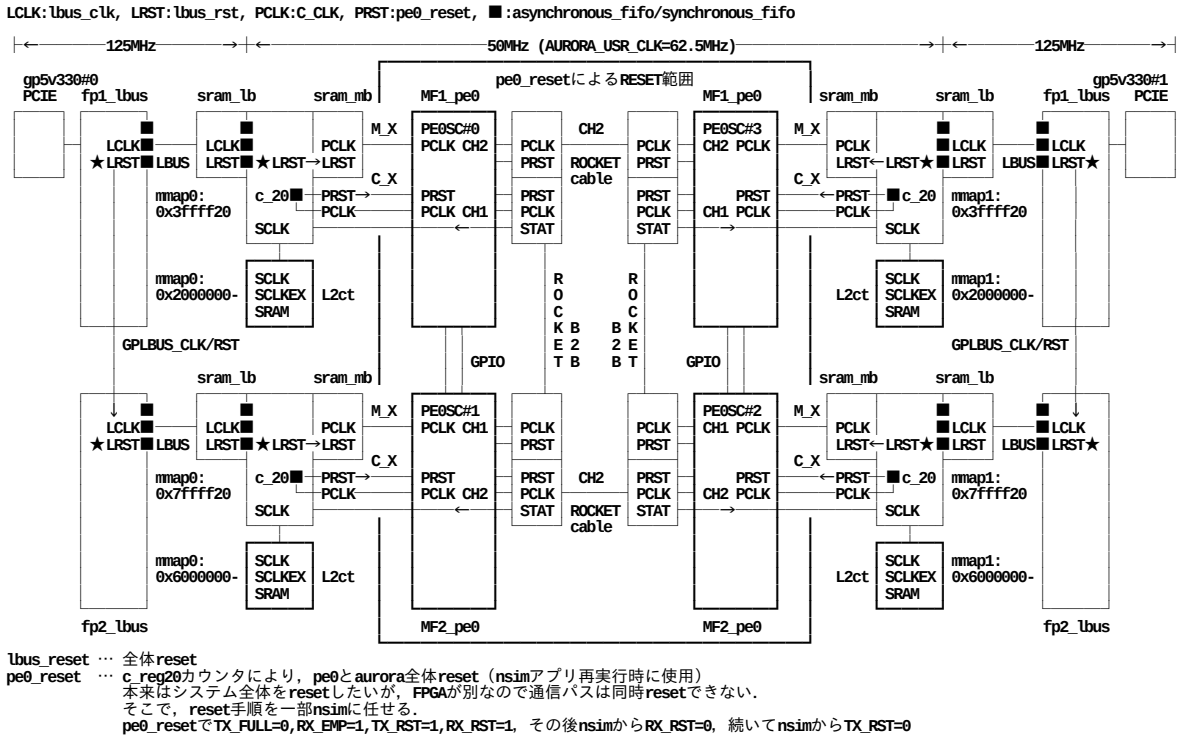


図 17.15: GP5V330MF システムの全体詳細

ンの読み出し要求が，読み出し主記憶アドレスとともに主記憶インタフェースにキューイングされる．

主記憶から到着した有効ラインは一旦主記憶インタフェース内バッファ（MM_ctl.dline）に格納され，外部キャッシュの入れ換え対象ラインに書き込まれるとともに，外部メモリインタフェース内バッファ（L2_ctl.dline）に格納され，内蔵キャッシュに対して有効ラインの到着が M_STAT=2/3 により通知される．

外部メモリインタフェースに追い出し要求がキューイングされている場合，追い出し先となる外部キャッシュが検査される．外部キャッシュミスが検出された場合で，入れ換え対象ライン（ダイレクトマップなので一意に決まる）がさらにダーティラインである場合，入れ換え対象ラインの内容が主記憶インタフェース内バッファ（MM_ctl.dline）に読み出され，主記憶への追い出し要求が，追い出し先主記憶アドレスとともに主記憶インタフェースにキューイングされる．一方，入れ換え対象ラインがダーティラインでない場合，または，外部キャッシュヒットの場合，外部メモリインタフェース内バッファ（L2_ctl.dline）の内容が入れ換え対象ラインに格納され，入れ換え対象ラインのダーティビットがセットされる．

表 17.6: 主記憶アドレス空間

0x00000000-0x0effffff 0x0f000000-0x0fffffff	キャッシャブル主記憶領域 ノンキャッシャブル主記憶領域
0x0f004000	SWI-OPCD
0x0f004010	SWI-GR4
0x0f004014	SWI-GR5
0x0f004018	SWI-GR6
0x0f00401C	SWI-GR7
0x0f004020	SWI-RETVAL
0x0f004100	書き込み時，PCI-HOSTのcam_capt()を呼び出す
0x0f004104	書き込み時，PCI-HOSTのx11_updt()を呼び出す
0x0f100000	CAM_L_BUF（320x240仮想スクリーン＃１[左上]
0x0f180000	CAM_R_BUF（320x240仮想スクリーン＃２[右上]
0x0f200000	CAM_W_BUF（320x240仮想スクリーン＃３[左下]
0x0f280000	CAM_D_BUF（320x240仮想スクリーン＃４[右下]
0x10000000-0xffffffff	未実装領域

表 17.7: ZBT-SSRAM 空間

BAR3内アドレス	説明	各ビットの説明		備考
0x000003f - 00	SubCore#0 L2bf lineバッファ	byte63-32：l2bf	byte31-00：l2bf	
0x0000047 - 40	SubCore#0 L2ct.ADDR,MASK	bit63-32：addr	bit31- 0：mask	メモリ要求バイトマスク
0x000004f - 48	SubCore#0 L2ct.DRV, DRPT	bit63-32：drv	bit 7- 0：drt bit31- 8：drp	ストア値等マージ値　格納先REG種別 格納先REG番号
0x0000057 - 50	SubCore#0 L2ct.DADR,REPL	 bit63-32：repl_daddr	bit 0：repl_nc bit 1：repl_dirty bit31-24：repl_way	non-cacheable 追い出し指示 追い出し先ADDR　追い出しway#
0x000005f - 58	SubCore#0 L2ct.DMKH,DMKL	bit63-32：dmkh	bit31- 0：dmkl	追い出しライン部分有効バイトマスク
0x0000077 - 60	未使用			
0x000007f - 78	SubCore#0 L2ct.0000		bit 1- 0：v bit 5- 4：stat bit 7 ：flush bit13- 8：t bit17-14：type bit23-18：opcd bit25-24：at bit31-26：len	processorから3書込時はstatに1を指定 processorから0書込時， M_STAT無変更時はstatに3を指定 M_STAT=0変更時はstatに0を指定 hostからstatを更新する際にvも変更されるがprocessorはvを参照しない (内部的なvを参照するのみ) processorから3書込時はM_STATをコピー hostからの場合はそのままM_STATに伝搬 FLUSH指示 遅延調整タイマ メモリ要求種別 メモリ要求詳細 メモリ要求アーキテクチャ識別子 メモリ要求長
0x00000bf - 80	SubCore#1 L2ct領域			SubCore#0に同じ
0x00000ff - c0	SubCore#1 L2bf領域			SubCore#0に同じ
0x0000100 ： 0x0000fff	未使用			
0x000107f - 00	SubCore#0 BAR2制御レジスタ			
0x00010ff - 80	SubCore#1 BAR2制御レジスタ			SubCore#0に同じ
0x0001fff ： 0x000ffff	未使用			64KB空間

なお，前章において説明した通り，ZBT-SSRAM 空間は PCI-HOST が直接参照/ 更新することができる．外部メモリインタフェースを含む外部キャッシュの制御は，ソフトウェアとして PCI-HOST 上に実装してもよいし，ハードウェアとして XC5V330T 上に実装してもよい．1 次キャッシュミスに要する実サイクル数を減らす必要がある場合には，後者の構成が望ましい．ただし，後者の構成とする場合には，ZBT-SSRAM

図 17.16: 記憶階層間のキャッシュライン転送手順

空間に主記憶インタフェースを追加する必要がある．

17.3.3 主記憶

主記憶を管理する PCI-HOST は，主記憶インタフェースに読み出し要求がキューイングされている場合，主記憶から該当ラインを読み出し，主記憶インタフェース内バッファ（MM_ctl.dline）に格納した後に外部キャッシュ制御部に対して有効ラインの到着を通知する．また，主記憶インタフェースに追い出し要求がキューイングされている場合，主記憶インタフェース内バッファ（MM_ctl.dline）から主記憶に書き込みを行った後に外部キャッシュ制御部に対して書き込み完了を通知する．

17.3.4 システムコール時の動作

プロセッサ機能が SWI 命令を検出し PCI-HOST にシステムコールを実行させる場合，PCI-HOST に通知する前にプロセッサ機能が内蔵キャッシュの内容をフラッシュし，最新データが外部キャッシュまたは主記憶上に格納されている状態を保証しなければならない．SWI 命令によるプロセッサ機能の停止を検出した PCI-HOST は，必要に応じて外部キャッシュのダーティラインを主記憶装置に書き出し，外部キャッ

表 17.8: 制御レジスタ

BAR2内アドレス	説明		プロセッサR/W	PCI-HOSTR/W
プロセッサ制御レジスタ				
0x0000007 - 00	(h_c)	bit 3- 0	-未使用-	-未使用-
	h_p	bit15- 4	R	W
	h_s	bit31-16	R	W
0x000000f - 08	h_param	下位４バイト	R	W
0x0000017 - 10	h_control	下位４バイト	R	W
0x0000027 - 20	pe0_reset	bit15- 0	ー	W
システムコール制御レジスタ				
0x0000047 - 40	(c_stat)	bit 3- 0	-未使用-	-未使用-
	(a_stat)	bit 9- 4	-未使用-	-未使用-
	f_stat	bit15-10	W	R (h_p非0書込時busy)
	s_stat	bit23-16	W	R (h_s非0書込時busy)
	p_ipctv	bit27-24	W	R
	p_ipctb	bit31-28	W	R
0x000004f - 48	ipctbase	下位４バイト	W	R
0x0000057 - 50	ipctlen	下位４バイト	W	R
0x000005f - 58	ipctnpc	下位４バイト	W	R
0x0000067 - 60	ipcttar	下位４バイト	W	R
0x000006f - 68	-未使用-		-未使用-	-未使用-
0x0000077 - 70	-未使用-		-未使用-	-未使用-
0x000007f - 78	-未使用-		-未使用-	-未使用-

シュの該当ラインを無効化した上でシステムコールを実行し，当該プロセッサ機能に対して継続実行要求またはリスタート要求（longjump 時のみ）を発行しなければならない．

17.4 GP5V330MF システムの制御

本章では，制御インタフェースについて説明する．制御インタフェースは，前述の BAR2 空間上に割り付けられた，プロセッサ制御レジスタ（ハードウェアモニタ制御レジスタ含む）およびシステムコール制御レジスタからなる．プロセッサ機能は，PCI-HOST からの制御情報（h_p，h_s，h_param，h_control）を受取り，また，PCI-HOST に対してプロセッサ機能の状態（f_stat，s_stat，p_ipctv，p_ipctb，その他 PE 間連携用レジスタ）を通知する．表 17.8 に各制御レジスタの詳細を示す．プロセッサ R/W の列は，プロセッサ機能側に許可されるアクセス種別，PCI-HOSTR/W の列は，PCI-HOST 側に許可されるアクセス種別を示す．なお，PCI-HOST が pe0_rstcount に非ゼロの 16 ビット値を書き込んだ場合，プロセッサ機能に至る PE0_RESET 信号がアサートされ，pe0_rstcount がデクリメントされて 0 に達した後に PE0_RESET 信号が解除される．

PCI-HOST が h_p に対して FRV 制御コマンド（0x800 から 0xfff までのいずれか）を書き込んだ場合，f_stat に 1（STATUS_BUSY）がセットされ，その後，各機能毎に規定された値がセットされる．h_s に対してハードウェアモニタ制御コマンドを書き込んだ場合，s_stat に 1（STATUS_BUSY）がセットされ，その後，各機能毎に規定された値がセットされる．

プロセッサ機能は，PCI-HOST からの h_p または h_s への書き込みに伴ってアサートされる前述の C_XINT 信号により，h_p または h_s の更新を検知し，BAR2 空間を直接参照することにより，h_p，h_s，h_param，h_control を受け取る．なお，現時点では，h_control の有効ビットは SINGLE_CYCLE（ビット 1 ）である．SWI 命令を検知したプロセッサ機能は，SWI 命令が PCI-HOST により実行されるよう，FRV の場合は f_stat=4 （STATUS_SWI）により PCI-HOST に対して SWI 情報を通知し，前述の C_XINT 信号により PCI-HOST による SWI 命令の実行完了を待ち合わせる．SINGLE_CYCLE が 1 の場合， 1 サイクル動作後に，s_stat=2 （SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）を通知する．ただし，ZBT-SSRAM のバースト転送を行う外部メモリインタフェースに関しては， 1 サイクルで停止する保

証はなく，引き続く 1 サイクル動作が論理的に矛盾しない範囲内において外部メモリインタフェースが複数サイクル動作後に停止状態に遷移することがある．

17.4.1 プロセッサ制御レジスタ

本インタフェースは，PCI-HOST による，プロセッサ機能の起動/停止，および，システムコール実行時のアーキテクチャレジスタの参照/更新のための論理インタフェースである．

PROC_NOP

h_p	0x000
h_param	未使用

何もしない．f_stat は変化しない．

PROC_FRV_CONTINUE

h_p	0x801
h_param	未使用

プロセッサ機能は停止状態から走行状態に遷移し，プロセッサ内部に保持されている実行開始アドレスから命令の実行を開始する．プロセッサが走行状態である時に本機能を指定した場合，実行結果は予測不能である．本機能の実行完了時に，f_stat が 1 から 0（STATUS_NORMAL）に遷移する．

PROC_FRV_RESTART

h_p	0x802
h_param	実行開始アドレス

プロセッサ機能は停止状態から走行状態に遷移し，h_param により指定される実行開始アドレスから命令の実行を開始する．プロセッサ機能が走行状態である時に本機能を指定した場合，実行結果は予測不能である．本機能の実行完了時に，f_stat が 1 から 0（STATUS_NORMAL）に遷移する．

PROC_FRV_FLUSH_STOP

h_p	0x803
h_param	未使用

本機能は，通常，システムリセット直後に発行される．プロセッサ機能は，内蔵キャッシュのダーティラインを外部キャッシュに書き出した後，全てのキャッシュラインを無効化し，走行状態から停止状態に遷移する．外部キャッシュ機能を PCI-HOST 上のソフトウェアとして実装している場合，Flush_Stop 動作中に，L2_ctl を通じて PCI-HOST に対して外部キャッシュへの追い出し要求が通知されることがある．本機能の実行中，f_stat が 1 から 2（STATUS_FLUSH）に遷移し，実行完了時に 3（STATUS_FLUSH_END）に遷移する．

PROC_FRV_STOP

h_p	0x804
h_param	未使用

プロセッサ機能は，内蔵キャッシュの状態を保持したままで，走行状態から停止状態に遷移する．本機能の実行完了時に，f_stat が 1 から 5 （STATUS_STOP）に遷移する．

Alter-REG

h_p	0x9aX:FRV アーキテクチャレジスタ#0-15
	0x9bX:FRV アーキテクチャレジスタ#16-31
	0x9c0:FRV Link Register
	0x9c1:FRV ICC0
	0x9c2:FRV ICC1
h_param	書き込み値

h_p で指定したプロセッサ内部のレジスタに値を書き込む．ただし，実行開始アドレスの指定には，前述したプロセッサ制御コマンドを使用しなければならない．本コマンドは，f_stat に STATUS_STOP，STATUS_FLUSH_STOP または STATUS_SWI が表示されているときのみ受付可能である．本コマンド受付後，f_stat は STATUS_BUSY に遷移する．本コマンドの実行完了後，f_stat は本コマンドを受け付けた時点の値に遷移する．

Display-REG

h_p	0xaaX:FRV アーキテクチャレジスタ#0-15
	0xabX:FRV アーキテクチャレジスタ#16-31
	0xac0:FRV Link Register
	0xac1:FRV ICC0
	0xac2:FRV ICC1
h_param	未使用

h_p で指定したプロセッサ内部のレジスタから p_param へ値を読み出す．本コマンドは，f_stat に STATUS_STOP，STATUS_FLUSH_STOP または STATUS_SWI が表示されているときのみ受付可能である．本コマンド受付後，f_stat は STATUS_BUSY に遷移する．本コマンドの実行完了後，f_stat は本コマンドを受け付けた時点の値に遷移する．

未定義機能

未定義機能を指定した場合，実行結果は予測不能である．

17.4.2 ハードウェアモニタ制御レジスタ

本インタフェースは，PCI-HOST がプロセッサ機能の内部状態を把握するための論理インタフェースである．PCI-HOST がSINGLE_CYCLE をアサートした状態でプロセッサを起動すると，s_stat=2（SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）が通知され，プロセッサ機能は停止状態となる．この状態において，以後，PCI-HOST が以下の機能を利用することができる．

SCAN_NOP

h_s	0x0000

何もしない．s_stat は変化しない．

SCAN_STEP

h_s	0x0001

s_stat=2（SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）が通知されている状態において，さらに 1 サイクル実行する．s_stat=1 （SCAN_BUSY）を経て，本機能の実行完了時に，s_stat=2（SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）が通知される．

内部ラッチ書き込み

h_s	0x0500-0x05ff: SCAN_FRVIA1 グループ
h_s	0x0600-0x06ff: SCAN_FRVIF1 グループ
h_s	0x0700-0x07ff: SCAN_FRVDECODE グループ
h_s	0x1800-0x1fff: SCAN_SEL_READ グループ
h_s	0x2000-0x20ff: SCAN_EXEC_BRC グループ
h_s	0x2100-0x21ff: SCAN_EXEC_SFM グループ
h_s	0x2200-0x22ff: SCAN_EXEC_ALU グループ
h_s	0x2300-0x23ff: SCAN_EXEC_EAG グループ
h_s	0x2400-0x24ff: SCAN_EXEC_ME1 グループ
h_s	0x2500-0x25ff: SCAN_EXEC_ME2 グループ
h_s	0x2600-0x26ff: SCAN_EXEC_ME3 グループ
h_s	0x2700-0x27ff: SCAN_EXEC_ME4 グループ
h_s	0x2800-0x28ff: SCAN_EXEC_OP1 グループ
h_s	0x2900-0x29ff: SCAN_L2CT グループ
h_s	0x3800-0x3fff: SCAN_FRVIFCACHE_TAG グループ
h_s	0x5000-0x5fff: SCAN_FRVIFCACHE_DAT グループ
h_s	0x6000-0x7fff: SCAN_OPCACHE グループ
h_param	書き込み値

s_stat=2（SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）が通知されている状態において，内部状態に h_param を書き込む．s_stat=1（SCAN_BUSY）を経て，本機能の実行完了時に，s_stat=2 （SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）が通知される．内部状態は h_s により選択される．グループ内の詳細な状態割り当てについては別途規定する．

内部ラッチ読み出し

h_s	0x8000 | 0x0500-0x05ff: SCAN_FRVIA1 グループ
h_s	0x8000 | 0x0600-0x06ff: SCAN_FRVIF1 グループ
h_s	0x8000 | 0x0700-0x07ff: SCAN_FRVDECODE グループ
h_s	0x8000 | 0x1800-0x1fff: SCAN_SEL_READ グループ
h_s	0x8000 | 0x2000-0x20ff: SCAN_EXEC_BRC グループ
h_s	0x8000 | 0x2100-0x21ff: SCAN_EXEC_SFM グループ
h_s	0x8000 | 0x2200-0x22ff: SCAN_EXEC_ALU グループ
h_s	0x8000 | 0x2300-0x23ff: SCAN_EXEC_EAG グループ
h_s	0x8000 | 0x2400-0x24ff: SCAN_EXEC_ME1 グループ
h_s	0x8000 | 0x2500-0x25ff: SCAN_EXEC_ME2 グループ
h_s	0x8000 | 0x2600-0x26ff: SCAN_EXEC_ME3 グループ
h_s	0x8000 | 0x2700-0x27ff: SCAN_EXEC_ME4 グループ
h_s	0x8000 | 0x2800-0x28ff: SCAN_EXEC_OP1 グループ
h_s	0x8000 | 0x2900-0x29ff: SCAN_L2CT グループ
h_s	0x8000 | 0x3800-0x3fff: SCAN_FRVIFCACHE_TAG グループ
h_s	0x8000 | 0x5000-0x5fff: SCAN_FRVIFCACHE_DAT グループ
h_s	0x8000 | 0x6000-0x7fff: SCAN_OPCACHE グループ
h_param	未使用

s_stat=2（SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）が通知されている状態において，内部状態を p_param に読み出す．s_stat=1（SCAN_BUSY）を経て，本機能の実行完了時に，s_stat=2 （SCAN_SINGLE_STOP）または s_stat=3（SCAN_ERROR_STOP）が通知される．内部状態は h_s により選択される．グループ内の詳細な状態割り当てについては別途規定する．

17.4.3 システムコール制御レジスタ

STATUS_SWI が f_stat に表示され，PCI-HOST がプロセッサ機能からシステムコール要求を受けた場合，PCI-HOST は，主記憶アドレス 0x0f004000（ノンキャッシャブル空間）の内容により指定されたシステムコールを実行しなければならない．システムコール検出時には，プロセッサ内部の一次キャッシュは無効化されているものの，外部キャッシュにはダーティラインが残っている可能性がある．このため，PCI-HOST は，システムコールを実行する前に，ダーティラインを主記憶に追い出し，外部キャッシュを全て無効化しなければならない．もちろん，システムコール実行による主記憶内容更新の影響範囲を把握できる場合には，該当範囲のキャッシュラインのみを無効化すればよい．また，Exit を除くシステムコールについては，返り値を主記憶アドレス 0x0f004020 に書き込んだ後，PROC_FRV_CONTINUE によりプロセッサを再スタート（Longjmp の場合は PROC_FRV_RESTART）しなければならない．

Exit

0x0f004000	1
0x0f004010-401f	未使用

プロセッサ機能がプログラムにより指定された停止命令を実行したことを示す．引続きプログラムを実行するためには，PCI-HOST はプロセッサ機能に対し，PROC_FRV_FLUSH_STOP および PROC_FRV_RESTART を発行して内部状態をリセットしなければならない．本手順を行わずに PROC_FRV_CONTINUE 等により実行を再開した場合，実行結果は予測不能である．

Read

0x0f004000	3
0x0f004010	ファイルディスクリプタ
0x0f004014	転送先先頭物理アドレス
0x0f004018	バイト数
0x0f00401c	未使用

入力装置から主記憶へのデータ転送要求である．主記憶への書き込みを伴うため，実行前に，影響を受ける範囲の外部キャッシュのダーティラインについては追い出しおよび無効化が必要である．

Write

0x0f004000	4
0x0f004010	ファイルディスクリプタ
0x0f004014	転送元先頭物理アドレス
0x0f004018	バイト数
0x0f00401c	未使用

主記憶から出力装置へのデータ転送要求である．主記憶からの読み出しを伴うため，実行前に，参照される範囲の外部キャッシュのダーティラインについては追い出しが必要である．

Open

0x0f004000	5
0x0f004010	ファイル名先頭物理アドレス
0x0f004014	第 2 引数
0x0f004018	第 3 引数
0x0f00401c	未使用

ファイルオープン要求である．

Close

0x0f004000	6
0x0f004010	ファイルディスクリプタ
0x0f004014-401f	未使用

ファイルクローズ要求である.

Fstat

0x0f004000	7
0x0f004010	ファイルディスクリプタ
0x0f004014	格納先先頭物理アドレス
0x0f004018-401f	未使用

ファイル状態の取得要求である.

Stat

0x0f004000	8
0x0f004010	ファイル名先頭物理アドレス
0x0f004014	格納先先頭物理アドレス
0x0f004018-401f	未使用

ファイル状態の取得要求である.

Access

0x0f004000	9
0x0f004010	ファイル名先頭物理アドレス
0x0f004014	ファイルモード
0x0f004018-401f	未使用

ファイルのアクセシビリティ検査要求である.

Fopen

0x0f004000	10
0x0f004010	ファイル名先頭物理アドレス
0x0f004014	ファイルモード指定文字列先頭物理アドレス
0x0f004018-401f	未使用

ファイルオープン要求である.

Freopen

0x0f004000	32
0x0f004010	ファイル名先頭物理アドレス
0x0f004014	ファイルモード指定文字列先頭物理アドレス
0x0f004018	オープン済ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f00401c	未使用

ファイル再オープン要求である.

Fread

0x0f004000	11
0x0f004010	格納先先頭物理アドレス
0x0f004014	ブロックサイズ
0x0f004018	ブロック数
0x0f00401c	ファイルポインタ
	0xf0 ⇒ stdin に読み替え．その他⇒無変更

ファイル読み込み要求である．主記憶への書き込みを伴うため，実行前に，影響を受ける範囲の外部キャッシュのダーティラインについては追い出しおよび無効化が必要である．

Fgets

0x0f004000	12
0x0f004010	格納先先頭物理アドレス
0x0f004014	バイト数
0x0f004018	ファイルポインタ
	0xf0 ⇒ stdin に読み替え．その他⇒無変更
0x0f00401c	未使用

ファイル読み込み要求である．主記憶への書き込みを伴うため，実行前に，影響を受ける範囲の外部キャッシュのダーティラインについては追い出しおよび無効化が必要である．また，返り値が 0 以外の場合，PCI ホストにおける fgets() の返り値をプロセッサの主記憶物理アドレスに変換した値を 0x0f004020 に格納しなければならない．

Fgetc

0x0f004000	13
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin に読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイル読み込み要求である．

Ungetc

0x0f004000	14
0x0f004010	取消文字
0x0f004014	ファイルポインタ
	0xf0 ⇒ stdin に読み替え．その他⇒無変更
0x0f004018-401f	未使用

ファイル読み込み取消要求である．

Getchar

0x0f004000	15
0x0f004010-401f	未使用

標準入力読み込み要求である．

Fwrite

0x0f004000	16
0x0f004010	先頭物理アドレス
0x0f004014	ブロックサイズ
0x0f004018	ブロック数
0x0f00401c	ファイルポインタ
	0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更

ファイル書き込み要求である．主記憶からの読み出しを伴うため，実行前に，参照される範囲の外部キャッシュのダーティラインについては追い出しが必要である．

Fputs

0x0f004000	17
0x0f004010	先頭物理アドレス
0x0f004014	ファイルポインタ
	0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004018-401f	未使用

ファイル書き込み要求である．主記憶からの読み出しを伴うため，実行前に，参照される範囲の外部キャッシュのダーティラインについては追い出しが必要である．

Puts

0x0f004000	31
0x0f004010	先頭物理アドレス
0x0f004014-401f	未使用

標準出力書き込み要求である．主記憶からの読み出しを伴うため，実行前に，参照される範囲の外部キャッシュのダーティラインについては追い出しが必要である．

Fputc

0x0f004000	18
0x0f004010	出力文字
0x0f004014	ファイルポインタ
	0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004018-401f	未使用

ファイル書き込み要求である．

Putchar

0x0f004000	19
0x0f004010	出力文字
0x0f004014-401f	未使用

標準出力書き込み要求である．

Fflush

0x0f004000	20
0x0f004010	ファイルポインタ
	0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイルフラッシュ要求である．

Fseek

0x0f004000	21
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014	オフセット
0x0f004018	ポジション指定
0x0f00401c	未使用

ファイルシーク要求である．

Ftell

0x0f004000	22
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイルポジション通知要求である．

Fgetpos

0x0f004000	39
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014	格納先主記憶物理アドレス
0x0f004018-401f	未使用

ファイルポジション通知要求である．

Rewind

0x0f004000	33
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイルリワインド要求である．

Fclose

0x0f004000	23
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイルクローズ要求である．

Clearerr

0x0f004000	24
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイルエラー状態消去要求である．

Ferror

0x0f004000	25
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイルエラー状態通知要求である．

Fileno

0x0f004000	26
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイルディスクリプタ通知要求である．

Feof

0x0f004000	28
0x0f004010	ファイルポインタ
	0xf0 ⇒ stdin，0xf1 ⇒ stdout，0xf2 ⇒ stderr に各々読み替え．その他⇒無変更
0x0f004014-401f	未使用

ファイル終端検査要求である．

Unlink

0x0f004000	27
0x0f004010	ファイル名先頭物理アドレス
0x0f004014-401f	未使用

ファイル削除要求である．

Remove

0x0f004000	38
0x0f004010	ファイル名先頭物理アドレス
0x0f004014-401f	未使用

ファイル削除要求である．

Rename

0x0f004000	29
0x0f004010	旧ファイル名先頭物理アドレス
0x0f004014	新ファイル名先頭物理アドレス
0x0f004018-401f	未使用

ファイル改名要求である．

Time

0x0f004000	30
0x0f004010	格納先先頭物理アドレス
0x0f004014-401f	未使用

時刻取得要求である．0x0f004010 が 0 の場合，格納先物理アドレスも 0 である．

Mktemp

0x0f004000	34
0x0f004010	ファイル名テンプレート先頭物理アドレス
0x0f004014-401f	未使用

一時ファイル作成要求である．返り値が 0 以外の場合，PCI ホストにおける mktemp() の返り値をプロセッサの主記憶物理アドレスに変換した値をレジスタ#0 に格納しなければならない．

Tmpfile

0x0f004000	35
0x0f004010-401f	未使用

ファイルポインタ通知要求である．

Isatty

0x0f004000	36
0x0f004010	ファイルディスクリプタ
0x0f004014-401f	未使用

ファイルディスクリプタ検査要求である．

Ttyname

0x0f004000	37
0x0f004010	ファイルディスクリプタ
0x0f004014-401f	未使用

デバイス名通知要求である．返り値が 0 以外の場合，PCI ホストにおける ttyname() の返り値をプロセッサの主記憶物理アドレスに変換した値を 0x0f004020 に格納しなければならない．

未定義

0x0f004000 が未定義の場合は，ハードウェア異常である．

Chapter 18

回路規模・電力評価モデル

本章では，C-RTL モデルに対して回路規模および電力パラメタを付与するための回路規模・電力評価モデルについて説明する．

18.1 ASIC モデル

FPGA モデルに用いた Verilog 記述を ASIC ライブラリと組み合わせて合成配置配線することにより，LSI として実現した場合の回路規模を評価することができる．また，ネットリストと ASIC ライブラリの電力モデルを組み合わせることにより，ゲートレベルの電力評価を行い，前述した各回路ブロックの電力パラメタを取得できる．以下，本モデルを ASIC モデルと呼ぶことにする．回路規模の見積りと，C-RTL モデルにより得られる実用アプリケーション実行時のサイクル数を組み合わせることにより，回路規模あたりの実効性能について SAPP とメニコア（SAPP 初段のみを SAPP と同一回路規模分並置した構成）を比較できる．また，FPGA モデルにより妥当性を裏付けられた C-RTL モデルに対し，さらに，取得した電力パラメタを組み込むことにより，実用アプリケーション走行時の消費電力を見積もり，同様に SAPP とメニコアを比較できる．図 18.1 にフロントエンドのブロック図，図 18.2 にアレイ演算部のブロック図を示す．

18.2 回路規模パラメタ

Chapter16 において各回路ブロックの説明を行った．各回路ブロックに関連付けられる回路規模パラメタは，表 18.1 の通りである．

18.3 電力パラメタ

各回路ブロックに関連付けられる電力パラメタは，表 18.2 の通りである．なお，各電力パラメタを各回路ブロックの動作サイクル数に乗じた合計が電力量（表 18.3 および表 18.4）である．動作サイクル数は以下の各モードが占める割合により，回路ブロック毎に異なる．以下，動作サイクル数の集計方法について説明する．

非アレイモード (Normal) = SAPP 初段のみが VLIW 命令を実行している状態

演算器ネットワーク設定モード (NW) = DCPL 命令を検出してから無条件後方分岐命令を検出するまでの状態（10.2.3 節に対応）

アレイ演算モード (Array) = 無条件後方分岐命令を検出して非アレイ動作に復帰するまでの状態（10.2.2 節および 10.2.4 節に対応）．ただし，10.2.3 節の動作と 10.2.2 節の動作はオーバラップするため，10.2.2 節の動作時間全体がアレイ演算モードに計上されるのではないことに注意．

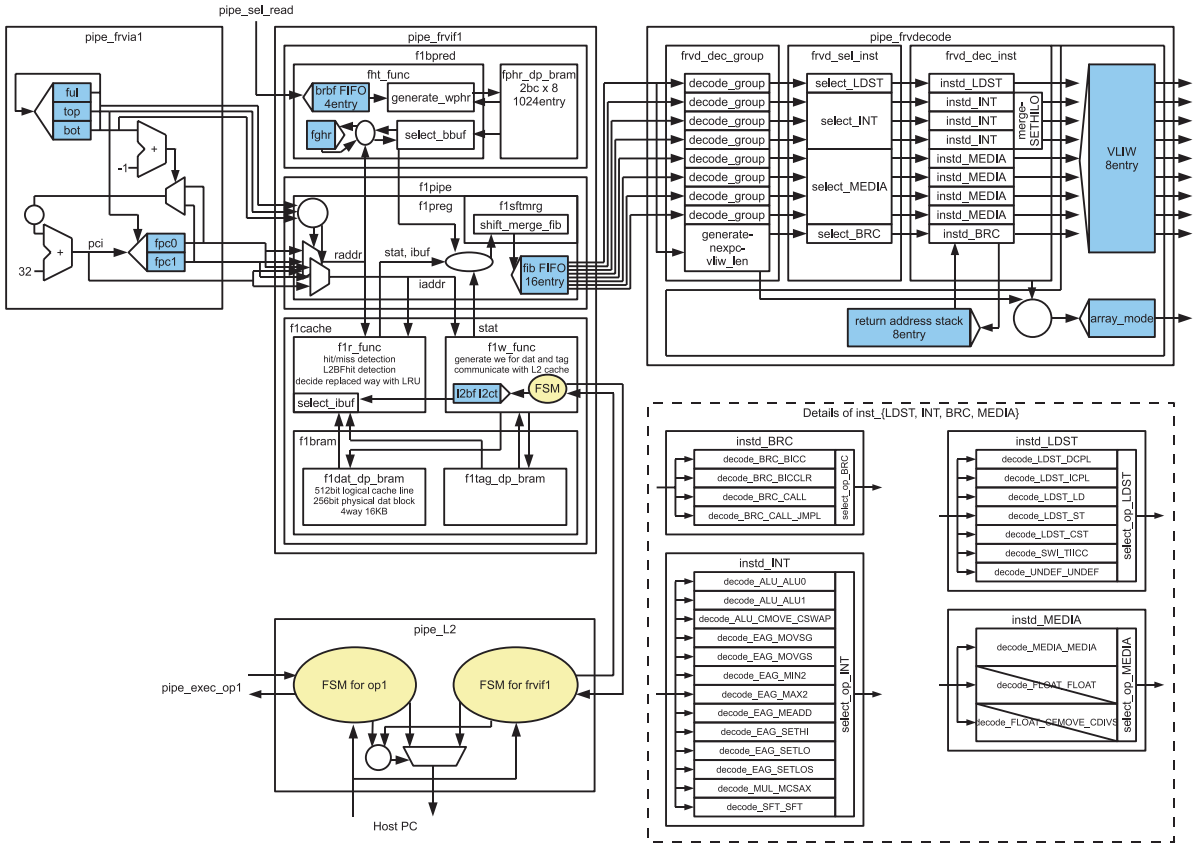


図 18.1: SAPP フロントエンド・ブロック図

表 18.1: 回路規模パラメタ

パラメタ名	設定値 (unit)	説明
AREA_IA_IF_DECODE	608+33613+16718	pipe_frvia，pipe_frvif，pipe_frvdecode におけるメモリ（命令キャッシュおよび分岐予測表）を除く基本回路のゲート数
AREA_F1	36467	命令キャッシュおよび分岐予測表の換算ゲート数
AREA_L1	51723	サブコア毎のオペランドキャッシュの換算ゲート数
AREA_REGFILE	56252	初段レジスタファイルのゲート数
AREA_SELRD	16718	pipe_frvdecode の機能のうち，各段間演算器ネットワークの設定に関する回路のゲート数
AREA_PREG_EAG	740	各 EAG 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_AL2	740	各 AL2 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_AL3	740	各 AL3 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_AL4	740	各 AL4 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_ME1	723	各 ME1 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_ME2	723	各 ME2 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_ME3	723	各 ME3 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_ME4	723	各 ME4 入力レジスタおよび関連セレクタのゲート数
AREA_PREG_BRC	740	各 BRC 入力レジスタおよび関連セレクタのゲート数
AREA_EXEC_EAG	1260	各 EAG 演算器本体のゲート数
AREA_EXEC_AL2	6600	各 AL2 演算器本体のゲート数
AREA_EXEC_AL3	6600	各 AL3 演算器本体のゲート数
AREA_EXEC_AL4	6600	各 AL4 演算器本体のゲート数
AREA_EXEC_ME1	5040	各 ME1 演算器本体のゲート数（Float 演算器も同一と仮定）
AREA_EXEC_ME2	5040	各 ME2 演算器本体のゲート数
AREA_EXEC_ME3	5040	各 ME3 演算器本体のゲート数（Float 演算器も同一と仮定）
AREA_EXEC_ME4	5040	各 ME4 演算器本体のゲート数
AREA_EXEC_BRC	747	各 BRC 演算器本体のゲート数
AREA_L0	14245	各 L0 キャッシュのゲート数
AREA_OOP4	1000	各 oop4 伝搬機能のゲート数
初段合計	284147	
次段以降各段合計	88777	

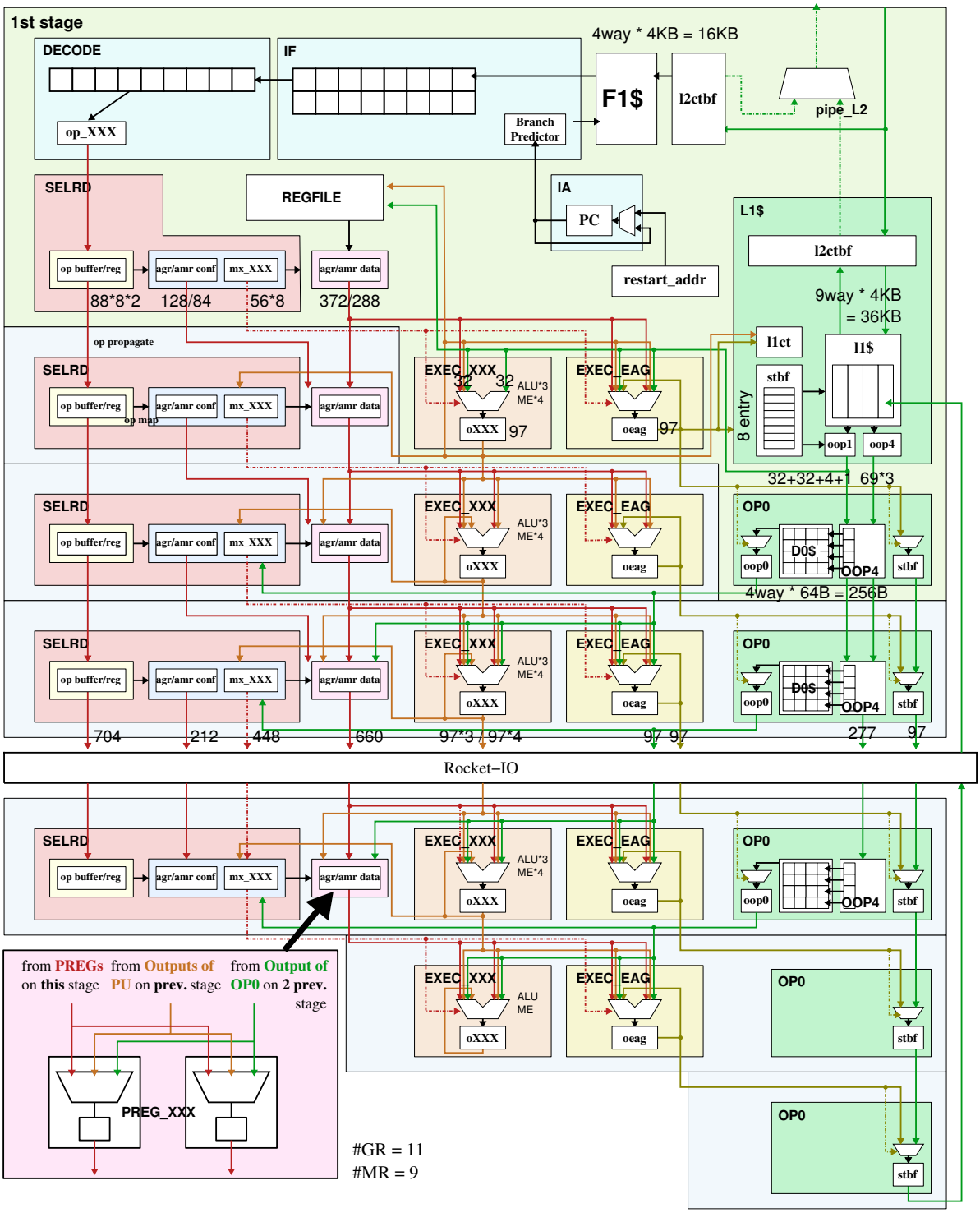


図 18.2: SAPP アレイ演算部・ブロック図

表 18.2: 電力パラメタ

パラメタ名	設定値 (unit/cycle)	説明
POWER_IA_IF_DECODE	36+1125+654	pipe_frvia，pipe_frvif，pipe_frvdecode におけるメモリ（命令キャッシュおよび分岐予測表）を除く基本回路の電力
POWER_F1_ACTIVE	9440	命令キャッシュおよび分岐予測表の通常時電力
POWER_F1_INACTIVE	3147	命令キャッシュおよび分岐予測表のスリープモード時電力
POWER_L1	10532	サブコア毎のオペランドキャッシュの通常時電力
POWER_REGFILE_ACTIVE	1900	初段レジスタファイルの通常時電力
POWER_REGFILE_INACTIVE	633	初段レジスタファイルのスリープモード時電力
POWER_SELRD	1840	pipe_frvdecode の機能のうち，各段間演算器ネットワークの設定に関する回路の電力
POWER_PREG_EAG	30	各 EAG 入力レジスタおよび関連セレクタの電力
POWER_PREG_AL2	30	各 AL2 入力レジスタおよび関連セレクタの電力
POWER_PREG_AL3	30	各 AL3 入力レジスタおよび関連セレクタの電力
POWER_PREG_AL4	30	各 AL4 入力レジスタおよび関連セレクタの電力
POWER_PREG_ME1	30	各 ME1 入力レジスタおよび関連セレクタの電力
POWER_PREG_ME2	30	各 ME2 入力レジスタおよび関連セレクタの電力
POWER_PREG_ME3	30	各 ME3 入力レジスタおよび関連セレクタの電力
POWER_PREG_ME4	30	各 ME4 入力レジスタおよび関連セレクタの電力
POWER_PREG_BRC	30	各 BRC 入力レジスタおよび関連セレクタの電力
POWER_EXEC_EAG	80	各 EAG 演算器本体の電力
POWER_EXEC_AL2	650	各 AL2 演算器本体の電力
POWER_EXEC_AL3	650	各 AL3 演算器本体の電力
POWER_EXEC_AL4	650	各 AL4 演算器本体の電力
POWER_EXEC_ME1	436	各 ME1 演算器本体の電力（Float 演算電力も同一と仮定）
POWER_EXEC_ME2	436	各 ME2 演算器本体の電力
POWER_EXEC_ME3	436	各 ME3 演算器本体の電力（Float 演算電力も同一と仮定）
POWER_EXEC_ME4	436	各 ME4 演算器本体の電力
POWER_EXEC_BRC	36	各 BRC 演算器本体の電力
POWER_L0	1420	各 L0 キャッシュの電力
POWER_OOP4	122	各 oop4 伝搬機能の電力

表 18.3: 初段の電力量集計

命令関連	= POWER_IA_IF_DECODE	* (Normal + NW)
	+ POWER_SELRD	* (Normal + NW)
命令 cache 関連	= POWER_F1_ACTIVE	* (Normal + NW)
	+ POWER_F1_INACTIVE	* (Array)
データ cache 関連	= POWER_L1	* (Normal + NW + Array)
	+ POWER_L0	* (L0 キャッシュ動作サイクル数)
	+ POWER_OOP4	* (oop4 伝搬機能動作サイクル数)
レジスタ関連	= POWER_REGFILE_ACTIVE	* (Normal + NW)
	+ POWER_REGFILE_INACTIVE	* (Array)
	+ POWER_PREG_EAG	* (EAG 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数 + #3)
	+ POWER_PREG_AL2	* (AL2 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数)
	+ POWER_PREG_AL3	* (AL3 入力レジスタ#1 動作サイクル数 + #3 動作サイクル数)
	+ POWER_PREG_AL4	* (AL4 入力レジスタ#1 動作サイクル数 + #4 動作サイクル数)
	+ POWER_PREG_ME1	* (ME1 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数 + #3)
	+ POWER_PREG_ME2	* (ME2 入力レジスタ#1 動作サイクル数)
	+ POWER_PREG_ME3	* (ME3 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数 + #3)
	+ POWER_PREG_ME4	* (ME4 入力レジスタ#1 動作サイクル数)
	+ POWER_PREG_BRC	* (BRC 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数)
演算器関連	= POWER_EXEC_EAG	* (EAG 演算器動作サイクル数)
	+ POWER_EXEC_AL2	* (AL2 演算器動作サイクル数)
	+ POWER_EXEC_AL3	* (AL3 演算器動作サイクル数)
	+ POWER_EXEC_AL4	* (AL4 演算器動作サイクル数)
	+ POWER_EXEC_ME1	* (ME1 演算器動作サイクル数)
	+ POWER_EXEC_ME2	* (ME2 演算器動作サイクル数)
	+ POWER_EXEC_ME3	* (ME3 演算器動作サイクル数)
	+ POWER_EXEC_ME4	* (ME4 演算器動作サイクル数)
	+ POWER_EXEC_BRC	* (BRC 演算器動作サイクル数)

表 18.4: 次段以降各段の電力量集計

命令関連	= POWER_SELRD	* (NW)
命令 cache 関連	= 0	
データ cache 関連	= POWER_L1	* (サブコア L1 キャッシュ使用サイクル数)
	+ POWER_L0	* (L0 キャッシュ動作サイクル数)
	+ POWER_OOP4	* (oop4 伝搬機能動作サイクル数)
レジスタ関連	= POWER_PREG_EAG	* (EAG 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数 + #3)
	+ POWER_PREG_AL2	* (AL2 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数)
	+ POWER_PREG_AL3	* (AL3 入力レジスタ#1 動作サイクル数 + #3 動作サイクル数)
	+ POWER_PREG_AL4	* (AL4 入力レジスタ#1 動作サイクル数 + #4 動作サイクル数)
	+ POWER_PREG_ME1	* (ME1 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数 + #3)
	+ POWER_PREG_ME2	* (ME2 入力レジスタ#1 動作サイクル数)
	+ POWER_PREG_ME3	* (ME3 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数 + #3)
	+ POWER_PREG_ME4	* (ME4 入力レジスタ#1 動作サイクル数)
	+ POWER_PREG_BRC	* (BRC 入力レジスタ#1 動作サイクル数 + #2 動作サイクル数)
演算器関連	= POWER_EXEC_EAG	* (EAG 演算器動作サイクル数)
	+ POWER_EXEC_AL2	* (AL2 演算器動作サイクル数)
	+ POWER_EXEC_AL3	* (AL3 演算器動作サイクル数)
	+ POWER_EXEC_AL4	* (AL4 演算器動作サイクル数)
	+ POWER_EXEC_ME1	* (ME1 演算器動作サイクル数)
	+ POWER_EXEC_ME2	* (ME2 演算器動作サイクル数)
	+ POWER_EXEC_ME3	* (ME3 演算器動作サイクル数)
	+ POWER_EXEC_ME4	* (ME4 演算器動作サイクル数)
	+ POWER_EXEC_BRC	* (BRC 演算器動作サイクル数)

Chapter 19

性能・電力評価RTLシミュレータ使用手引

RTL シミュレータ（ssim9）は，C-RTL モデルに対応する実行形式ファイルであると同時に，FPGA モデルに対応する GP5V330MF システムの制御プログラムである．PCI-HOST 上で動作し，主記憶機能および入出力機能の提供，また，システム全体の制御を行う．

19.1 動作環境

以下に説明するツールチェインとともに，i386-FreeBSD-7.2R において動作する．

19.2 ツールチェインの導入

1. ツール格納用ディレクトリを作成

   % mkdirhier /usr/home/nakashim/proj-sap/

2. CD-ROM 内のファイルを配置

   % cd /CDROM/GNU-tools
   % tar cf – * | (cd /usr/home/nakashim/proj-sap/; tar xf -)
   % cd /CDROM
   % tar cf – sample | (cd /usr/home/nakashim/proj-sap/; tar xf -)
   % tar cf – src | (cd /usr/home/nakashim/proj-sap/; tar xf -)

3. ツール使用のためのコマンドパス設定

   % set path = (/usr/home/nakashim/proj-sap/bin /usr/home/nakashim/proj-sap/src/ssim9-xxxxxxxx $path)
   % rehash

19.3 ツールチェインの再構築

上記以外の環境の場合，RTL シミュレータおよび GNU-tools の再コンパイルが必要である．以下の手順を参考に再構築できる場合がある．（以下は参考手順につき，利用環境に応じて各自対応のこと）

1. RTL シミュレータの再コンパイル

   % cd /usr/home/nakashim/proj-sap/src/ssim9-xxxxxxxx/
   % gmake clean
   % gmake

2. アセンブラ・リンカ等の再コンパイル

```
% cd /usr/home/nakashim/proj-sap/src/binutils-2.18
% ./MAKE_INSTALL.sparc
% less Make.log （ログ確認）
% gmake install
```

3. C コンパイラの再コンパイル（参考手順）

```
% cd /usr/home/nakashim/proj-sap/src/gcc-4.1.2/
% ./MAKE_INSTALL.sparc
% less Make.log （ログ確認：[configure-target-libstdc++-v3] error で停止してよい）
% gmake install
```

19.4 アプリケーションプログラムの実行

前述の RTL シミュレータ，アセンブラ・リンカ・コンパイラ等が使用可能であることを前提に，アプリケーションプログラムを RTL シミュレータ上で実行し，性能評価を行う手順を順に説明する．説明には，前述の「sample/」を用いる．

1. Hello.c のコンパイルと実行

```
% cd sample/stanford/
% rm Hello64
% gmake Hello64
% ssim9 Hello64 | less
```

2. Queens のコンパイルと実行

```
% cd sample/stanford/
% rm Queens64
% gmake Queens64
% ssim9 Queens64 | less
```

3. Towers のコンパイルと実行

```
% cd sample/stanford/
% rm Towers64
% gmake Towers64
% ssim9 Towers64 | less
```

4. 画像処理サンプルプログラム（メディア命令なし）のコンパイルと実行

```
% Xwindow を起動
% cd sample/filter/
% gmake clean -f Makefile.ssim9-c
% gmake -f Makefile.ssim9-c
% gmake -f Makefile.ssim9-c run
```

5. 画像処理サンプルプログラム（メディア命令使用）のコンパイルと実行

```
% Xwindow を起動
% cd sample/filter/
% gmake clean -f Makefile.ssim9-media
% gmake -f Makefile.ssim9-media
% gmake -f Makefile.ssim9-media run （非アレイ動作）
% ssim9 -x frontend.ssim9-media -f /tmp/1 81.ppm 82.ppm （同じく非アレイ動作）
```

% ssim9 -x -a frontend.ssim9-media -f /tmp/1 81.ppm 82.ppm （アレイ動作）

% ssim9 -x -a -b479664 frontend.ssim9-media -f /tmp/1 81.ppm 82.ppm | less -S -R -j10 -c -y1 -B -b128 -p'≪ PE0SG00 ≫'

19.5 起動時オプション

-i 10.4 節において説明した複数 PE 連携機能を有効にする．

-l 10.2 節において説明したアレイ動作機能のうち PREFETCH 機能のみを有効にする．

-a 10.2 節において説明したアレイ動作機能全体を有効にする．

-g GP5V330MF システムの制御プログラムとして動作する．本オプションを指定しない場合，C-RTL モデルとして動作する．

-t 命令実行は行わず，表 17.6 の cam_capt() 機能および x11_updt() 機能の動作検証のみを行う．

-x 命令実行中，表 17.6 の cam_capt() 機能および x11_updt() 機能を有効にする．

-bxxx 命令実行トレースを xxx ステップ経過後に開始する．

-exxx 命令実行トレースを xxx ステップ経過後に停止する．

-f 命令実行状態を表示する．

-p パイプラインの状態を表示する．

-dxxx 全体動作概略を xxx ステップ経過後に開始する．

19.6 シミュレーション動作中の表示

```
=PE0 STAT0 INTERPE0 ARRAY_MODE8 SET_STAGE0 RESTART_PENDING0==========                                  ⇒プロセッサ状態
       <FPC  1|00021920 1|00021940                                                                     ⇒プログラムカウンタ
        F1TG 808888000000888cccc8ec808cce8880888cc888888000ceeeecccc888000000                          ⇒命令キャッシュTAG 状態
       <FIB  1|1|000218ec|00|96c82008 1|1|000218f0|00|84c82000 1|1|000218f4|00|0030b000 1|1|000218f8|00|82401030 1|1|000218fc|00|80880000
        RTAS 00020564 00021534 0002142c 0002141c 000217bc 00018e38 0002170c 00018ccc                  ⇒復帰アドレススタック ↑命令バッファ
<<PE0SG00>> |cf:ta:o0e:9|ce:t4:o02:5|ce:t6:o0d:1|c0:t0:o00:1|c0:t0:o00:1|c0:t0:o00:1|c0:t0:o00:1|c0:t0:o00:1|ce:tc:o0b:1|pc:00020564
 ss2 sp1 s00|x2:05:z0:00|x2:07:z0:00|x2:14:z0:00|x0:00:z0:00|x0:00:z0:00|x0:00:z0:00|x0:00:z0:00|x0:00:z0:00|x0:00:z0:00|x1:00:z0:00 ⇒初段命令デコーダ EAG,AL2,AL3,AL4,ME1,ME2,ME3,ME4,BRC
            |y2:0000000e|y1:00000001|y1:80000040|y0:00000000|y0:00000000|y0:00000000|y0:00000000|y0:00000000|y1:00021800
            |ss1:01::sc0|ss0:00::sc0|ss0:00::sc0|ss0:00::sc0|ss0:00::sc0|ss0:00::sc0|ss0:00::sc0|ss0:00::sc0|ss1:01::sc0
            |dr0:00::dc0|dr2:07::dc2|dr2:14::dc0|dr0:00::dc0|dr0:00::dc0|dr0:00::dc0|dr0:00::dc0|dr0:00::dc0|dr2:32::dc0
          GR|00:00000000|01:0efffbb8|02:0efffbd8|03:00000000|04:007be080|05:003d1080|06:0efffc30|07:00000141|08:00027f30|09:00000000|10:00000000|11:00020564|12:0efffd30|13:0efffe30|
            |16:00000000|17:0004b000|18:00000000|19:000277a8|20:00027794|21:000277ac|22:00000000|23:00000000|24:00000000|25:00000000|26:003d1080|27:00683080|28:007be080|29:005a2080|
          MR|00:00000000|01:00000000|02:00000000|03:00000000|04:00000000|05:00000000|06:00000000|07:00000000|08:00000000|09:00000000|10:00000000|11:00000000|12:00000000|13:00000000|
            |16:00000000|17:00000000|18:00000000|19:00000000|20:00000000|21:00000000|22:00000000|23:00000000|24:00000000|25:00000000|26:00000000|27:00000000|28:00000000|29:00000000|
         PSKG_00002202_00220020_00000000_00000000_0 PSKM_00000000_00000000_00000000_00000000          ⇒レジスタ伝搬情報 ↑汎用レジスタ ↑メディアレジスタ
    <RPRT|00|04|06|12|00|07|63|  |00|13|05|  |00|00|00|  |00|00|00|00|  |  |00|  |  |00|00|00|  |  |00|  |  |00|00|  |00 ⇒演算器毎レジスタ READ ポート（ラッチ）
            |g1:007be080|g1:00000141|g1:0efffe30|g1:00000000|m1:00000000|m1:0|m2:0  |m1:00000000|m1:0|m2:0  |g1:00020564 ⇒演算器入力レジスタ
            |g2:0efffc30|g2:00000001|g2:003d1080|g2:00000000|m2:00000000|m2:00000000|m2:00000000|m2:00000000|g2:00021800 ⇒演算器入力レジスタ
 m3:0efffd30|g3:0efffd30|sc:0       |sc:0       |sc:0       |m3:00000000|m3:0       |m3:00000000|m3:0       |sc:0        ⇒演算器入力レジスタ
       <INMX|0:f:ta:op0e|2:e:t4:op02|0:e:t3:op03|0:0:t0:op00|0:0:t0:op00|0:0:t0:op00|0:0:t0:op00|0:0:t0:op00|0:0:t0:op00|0:e:tc:op09 ⇒演算器入力セレクタ（バイパス切替え）
            |x0:y5:z0   |x4y1s0:00:0|x2y2s0:00:0|x0y0s0:00:0|x0:y0:z0   |x0:y0:z0   |x0:y0:z0   |x0:y0:z0   |x2y1s1:01:0
            |dr0:00::dc0|dr2:07::dc2|dr2:10::dc0|dr0:00::dc0|dr0:00::dc0|dr0:00::dc0|dr0:00::dc0|dr0:00::dc0|dr0:00::dc0
       <OTWB|2:s0:dr0:00|2:s0:dr2:07|2:s0:dr2:10|0:s0:dr0:00|0:s0:dr0:00|0:s0:dr0:00|0:s0:dr0:00|0:s0:dr0:00|2:s0:dr0:00 ⇒演算器出力ラッチ（EAG は OP0/OP1 の入力ラッチ兼用）
            |dv:0efffbd8|dv:0000013e|dv:00000000|dv:00000000|dv:00000000|dv:00000000|dv:00000000|dv:00000000|rst:00000000
            |st:0efffd30|dct2:mf:v0 |dct0:m0:v0 |dct0:m0:v0 |dct0:m0:v0 |dct0:m0:v0 |dct0:m0:v0 |dct0:m0:v0 |dct0:m3:v1
            |eag:0.00000|al2:0.02344|al3:0.00000|al4:0.00000|me1:0.00000|me2:0.00000|me3:0.00000|me4:0.00000|brc:0.00000|ipc:0.02344(ave. of 128cycles)IPC:0.04688 DECODER:0.00000
            ↑ EAG 動作率↑ AL2 動作率↑ AL3 動作率↑ AL4 動作率↑ ME1 動作率↑ ME2 動作率↑ ME3 動作率↑ ME4 動作率↑ BRC 動作率 ↑過去 128 サイクルの平均 IPC    ↑全体 IPC  ↑デコーダ動作率
  STBF                                                                                                ⇒ストアバッファ
 <OP0[0] 2|s0|dc0:0:0|dr0:00:0efffbd8 <OP0[1] 2|s0|dc0:0:0|dr0:00:0efffbe0 <OP4[0] d0:0:00000000|d1:1:8c8e4400|d2:0:00000000|d3:0:00000000 <OP4[1] d0:0:00000000|d1:1:4f502500|d2:0:0
  L0(0)[0]|0:00000000 00000000 00000000 00000000 00000000|0:00000010 00000000 00000000 00000000 00000000|0:00000020 00000000 00000000 00000000 00000000|0:00000030 00000000 00000000
  L0(0)[1]|0:00000000 00000000 00000000 00000000 00000000|0:00000010 00000000 00000000 00000000 00000000|0:00000020 00000000 00000000 00000000 00000000|0:00000030 00000000 00000000
  L0(0)[2]|0:00000000 00000000 00000000 00000000 00000000|0:00000010 00000000 00000000 00000000 00000000|0:00000020 00000000 00000000 00000000 00000000|0:00000030 00000000 00000000
  L0(0)[3]|0:00000000 00000000 00000000 00000000 00000000|0:00000010 00000000 00000000 00000000 00000000|0:00000020 00000000 00000000 00000000 00000000|0:00000030 00000000 00000000
 <L1CT_DLINE 00000000_0000001f                                                                        ↑ L0 キャッシュ内容
 <L1CT[0] 0|st2|rw0|pn140|rn140|0:0-0-1-3d1080|1:0-0-1-7be080|2:0-0-0-0|3:0-0-0-0|cd0:00000005|cd1:00000640   ⇒初段 L1 制御
  L1TG0|.....................................................D.............. 1|st3|ct1|q0|A003d1080|L0500      ⇒初段 L1 内容および転送状態
  L1TG1|..VVVVVVVVVVVVVVVVVVV..................................... 1|st2|ct1|q7|A007be080|L0500
  L1TG2|.......................................................... 0|st0|ct1|q0|A00000000|L0000
  L1TG3|.......................................................... 0|st0|ct1|q0|A00000000|L0000
 <L1CT[1] 0|st0|rw0|pn000|rn040|0:0-0-1-0|1:0-0-1-efffc30|2:0-0-0-0|3:0-0-0-0|cd0:00000000|cd1:00000000      ⇒第 10 段 L1 制御
  L1TG0|.......................................................... 0|st0|ct0|q0|A00000000|L0000      ⇒第 10 段 L1 内容および転送状態
  L1TG1|.....................................................VVVVV.......... 1|st2|ct0|q7|A0efffc30|L0110
  L1TG2|.......................................................... 0|st0|ct0|q0|A00000000|L0000
  L1TG3|.......................................................... 0|st0|ct0|q0|A00000000|L0000
 <L1CT[2] 0|st0|rw0|pn000|rn040|0:0-0-1-0|1:0-0-1-efffd30|2:0-0-0-0|3:0-0-0-0|cd0:00000000|cd1:00000000      ⇒第 19 段 L1 制御
  L1TG0|.......................................................... 0|st0|ct0|q0|A00000000|L0000      ⇒第 19 段 L1 内容および転送状態
  L1TG1|.....................................................VVVVV..... 1|st2|ct0|q7|A0efffd30|L0110
  L1TG2|.......................................................... 0|st0|ct0|q0|A00000000|L0000
  L1TG3|.......................................................... 0|st0|ct0|q0|A00000000|L0000
 <L1CT[3] 0|st0|rw0|pn000|rn040|0:0-0-1-0|1:0-0-1-efffe30|2:0-0-0-0|3:0-0-0-0|cd0:00000000|cd1:00000000      ⇒第 28 段 L1 制御
  L1TG0|.......................................................... 0|st0|ct0|q0|A00000000|L0000      ⇒第 28 段 L1 内容および転送状態
  L1TG1|.....................................................VVVVV... 1|st2|ct0|q7|A0efffe30|L0110
  L1TG2|.......................................................... 0|st0|ct0|q0|A00000000|L0000
  L1TG3|.......................................................... 0|st0|ct0|q0|A00000000|L0000
 <L2 0|t00|ty8|ope|at2|A007be540|L01|Mffffffff|dr00:000000:00000000|s0|f10|nc0|rw1|rd0|ra00000000|mh00000000|ml00000000|e1e4f400 e7e5f600 e7e5f600 e7e5f600 e7eaf600 e7eaf600 e7eaf600
 <L2 0|t00|ty8|ope|at2|A0efffd00|L01|Mffffffff|dr00:000000:00000000|s0|f10|nc0|rw1|rd0|ra00000000|mh00000000|ml00000000|28292a2b 24252627 20212223 1c1d1e1f 18191a1b 14151617 10111213
 <L2 0|t00|ty8|ope|at2|A0efffe00|L01|Mffffffff|dr00:000000:00000000|s0|f10|nc0|rw1|rd0|ra00000000|mh00000000|ml00000000|28292a2b 24252627 20212223 1c1d1e1f 18191a1b 14151617 10111213
 <L2 0|t00|ty8|ope|at2|A0efffF00|L01|Mffffffff|dr00:000000:00000000|s0|f10|nc0|rw1|rd0|ra00000000|mh00000000|ml00000000|28292a2b 24252627 20212223 1c1d1e1f 18191a1b 14151617 10111213
 <L2BANK -.-:9 -.-:9 -.-:9 -.-:9 -.-:9 -.-:9 -.-:9 -.-:9                                              ⇒ L2 バンク占有状態
 <MM 0|t00|ty8|ope|at2|A007be540|L16|Mffffffff|pid0|sub0|s0|rd0|bf0|ra00000000|e1e4f400 e1e4f400 e7e5f600 e7e5f600 e7e5f600 e7eaf600 e7eaf600 e7eaf600 e7eaf600 e1e4f400
 <IP0>lk0|wq0|A00000000|L00000000.....                                                                ⇒ PE0-1 間連携機能状態
 <IP1>lk0|wq0|A00000000|L00000000.....                                                                ⇒ PE1-2 間連携機能状態
 <IP2>lk0|wq0|A00000000|L00000000.....                                                                ⇒ PE2-3 間連携機能状態
 <IP3>lk0|wq0|A00000000|L00000000.....                                                                ⇒ PE3-4 間連携機能状態
 :  ↓以下，第２段の内部状態（初段との違いは，命令$，命令バッファ，レジスタファイル，L1$がないこと）
<<PE0SG01>> |c0:t0:o00:0|c0:t0:o00:0|c0:t0:o00:0|c0:t0:o00:0|c0:t0:o00:0|c0:t0:o00:0|c0:t0:o00:0|c0:tc:o08:0|pc:00021868
```

19.7 SVC 実行時の表示

システムコール実行時に，直前のシステムコール以降の IPC，L2 キャッシュ参照数，デコーダ動作率（全サイクル数から 10.2.4 節のアレイ演算動作サイクル数を除いた割合），および，各段の電力量を表示する．電力量の算出方法は，18.3 節にて説明した通りである．

```
svc31:cycle=00000000_001a31e4 ipc=14.248 l2req=0001aecb l2mis=00002447 decode/c=0.069
 pw.tot=0000000b_cff2f1e4 (00050733: 00002840 00006133 00030546 00002071 00009141)  ⇒各段の総計
                                  5.599   12.090   60.210    4.082   18.018(演算器稼働率:3.25\%)  ⇒全体に対する比率
                        各段総電力量 命令関連 命令キャッシュ関連
                                    |        |        |        データキャッシュ関連
                                    |        |        |        |        レジスタ関連
                                    ↓        ↓        ↓        ↓        ↓        ↓演算器関連
                                       inst   icache   dcache      reg     exec
 pw[00]=00000006_04a5cdea (00025847: 00000424 00006133 00018080 00000408 00000800)
 pw[01]=00000000_4784c29a (00001199: 00000069 00000000 00000195 00000139 00000795)
 pw[02]=00000000_43dc521a (00001138: 00000069 00000000 00000195 00000099 00000774)
 pw[03]=00000000_2a69841a (00000711: 00000069 00000000 00000195 00000060 00000387)
 pw[04]=00000000_42a7e99a (00001118: 00000069 00000000 00000195 00000079 00000774)
 pw[05]=00000000_4c47369a (00001279: 00000069 00000000 00001122 00000040 00000047)
 :
 pw[35]=00000000_042e3de0 (00000070: 00000069 00000000 00000000 00000001 00000000)
```

19.8 シミュレーション終了時の表示

```
==== Program normal end. ====                                                    ⇒ 停止状態表示
sparc_malloc_top = 0x003fe3b0                                                    ⇒ malloc() ポインタの最終値表示
 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000        ⇒ 先頭６４バイト領域（NULL ポインタ付近）表示
 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
====
0:cycle    =0x00000000_001b9bce                                                  ⇒ 実行サイクル数　↓実行命令数/IPC
0:sparc_step =0x00000000_001c41df(FRV_IPC=1.023 w.o.swi=1.024 w.o.swi+oop0=1.024 w.o.swi+oop0+stbf=1.024)
0:sparc_b_hit=0x00000000_0002025f(hit:0.643)                                     ⇒ 初段分岐予測 HIT 回数/率
0:sparc_b_mis=0x00000000_00011e23(mis:0.357)                                     ⇒ 初段分岐予測 MIS 回数/率
0:sparc_r_hit=0x00000000_00004003(hit:1.000)                                     ⇒ 初段復帰アドレススタック HIT 回数/率
0:sparc_r_mis=0x00000000_00000000(mis:0.000)                                     ⇒ 初段復帰アドレススタック MIS 回数/率
0:F1_hit   =0x00000000_0008ed35(hit:1.000)                                       ⇒ 命令キャッシュHIT 回数/率
0:F1_mis   =0x00000000_00000093(mis:0.000)                                       ⇒ 命令キャッシュMIS 回数/率
0:F1_rpl   =0x00000000_00000000(rpl:0.000)                                       ⇒ 命令キャッシュ無効化回数/率
0:sparc_reif =0x00000000_0001a263(0.059)                                         ⇒ 命令再フェッチ回数/率（主に分岐 TAKEN 時）
0:sparc_rest =0x00000000_0008f138(0.324)                                         ⇒ 命令再スタート回数/率（主に分岐予測 MIS 時）
0:w_swi    =0x00000000_000001a6(0.000)                                           ⇒ システムコール回数/率
0:w_op0b   =0x00000000_0002c038(0.100)                                           ⇒ OP0 に関する LD-USE 待ち回数/率
0:w_exeb   =0x00000000_00000000(0.000)                                           ⇒ レジスタ更新型 BRC に関する WRITE-READ 待ち回数/率
0:w_oeag   =0x00000000_00000048(0.000)                                           ⇒ OP0-INBUF-FULL に関する OP1 発行待ち回数/率
0:w_oop0   =0x00000000_00000000(0.000)                                           ⇒ OP0-OUTBUF-FULL に関する OP1 実行待ち回数/率
0:w_stbf   =0x00000000_0000007f(0.000)                                           ⇒ OP0-STBF-FULL に関する OP1 実行待ち回数/率
0:w_l2ct_op=0x00000000_00000053(0.000)                                           ⇒ L2$-REQ-QUEUE-FULL に関する OP0 実行待ち回数/率
0:w_l2ct_rt=0x00000000_00000266(0.000)                                           ⇒ L2$-REQ-QUEUE-FULL に関する STBF 追出待ち回数/率
0:SB_hit   =0x00000000_00000000(hit:0.000)                                       ⇒ LD 先が STBF にヒットした回数/率
0:L1_hit   =0x00000000_000c006d(hit:1.000)                                       ⇒ L1$HIT 回数/率
0:L1_mis   =0x00000000_00000052(mis:0.000)                                       ⇒ L1$MIS 回数/率
0:L1_rpl   =0x00000000_00000000(rpl:0.000)                                       ⇒ L1$無効化回数/率
0:decoder  =0x00000000_001b9bcd(1.000)                                           ⇒ 命令デコーダ動作回数/率
                                                                                 ↓ 各段の各ユニット動作回数/率
0.00:ea=000c008a(0.43)a2=000aa07c(0.38)a3=00022056(0.08)a4=00002000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.01:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.02:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.03:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.04:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.05:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.06:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.07:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.08:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.09:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.10:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.11:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.12:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.13:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.14:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.15:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.16:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.17:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.18:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.19:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.20:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.21:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.22:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.23:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.24:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.25:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.26:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.27:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.28:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.29:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
```

```
0.30:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.31:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.32:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.33:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.34:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
0.35:ea=00000000(0.00)a2=00000000(0.00)a3=00000000(0.00)a4=00000000(0.00)m1=00000000(0.00)m2=00000000(0.00)m3=00000000(0.00)m4=00000
A:L2_hit   =0x00000000_00000000(hit:0.000)                              ⇒ L2$HIT 回数/率
A:L2_mis   =0x00000000_00000009(mis:1.000)                              ⇒ L2$MIS 回数/率
A:L2_rpl   =0x00000000_00000000(rpl:0.000)                              ⇒ L2$無効化回数/率
```

Chapter 20

総合評価

評価モデルを図 20.1 に示す．VLIW プロセッサとして動作する SAPP 初段の回路規模は，アクセラレータを搭載しない一般的なメニコアの 1 コアと同じであるとする．また，ASIC モデルにより得られた回路規模パラメタより，SAPP 次段以降の各段の回路規模は初段の 33%である．すなわち，9 段構成のサブコアを 4 個連結した 36 段構成の SAPP 全体の回路規模は，14 コアからなるメニコアの回路規模と同等である．本章では，この比を基準として，性能・回路規模・電力の比較を行う．

20.1 画像処理による評価

電力モデルを追加した RTL シミュレータにより，画像フィルタを実行した結果を表 20.1 に示す．評価時の性能パラメタは表 16.1 の通りである．なお,「非アレイ実行」は SAPP 初段および L1 キャッシュ(way0) による実行,「PREFETCH のみ」は DCPL 命令の指定を用いた Local メモリ (way1-3) に対する PREFETCH を組み合わせた初段および L1 キャッシュ(way0) による実行,「アレイ実行」は PREFETCH に加えて全段を使用する SAPP 本来の実行形態である．非アレイ実行とアレイ実行の IPC 比は 1 ： 1 4 であることから，おおまかには，単一スレッドどうしの比較では，シングルコアと SAPP（36 段構成）の性能比は 1 ： 1 4 であると言える．

また,「PREFETCH のみ」の IPC を 14 倍した値を SAPP と同等の回路規模を有する 14 コアからなるメニコア (PREFETCH 適用) の IPC であると仮定した場合の比較を表 20.2 に示す．14 コアの場合，実行時間は理想的に 1/14 となり，14 コアが 1/14 の時間だけ動作すると仮定していることから電力量は同じである．14 コアと SAPP の回路規模は同等，また，IPC も同等であることから，理想的にスレッド分割したメニコア（L2 のバンク競合も発生しない）と，単一スレッドにより動作可能な SAPP の回路規模あたり性能比は 1 ： 1，回路規模あたり消費電力比は 8 ： 1（色補正除く）であると言える．なお，色補正の電力量比率が他に比べて高いのは，複数のサブコアに属する L1 を使用するためであり，この性質は，後述する数値計算にも見られる．

図 20.2 に，フレーム補間（SAD 計算）と鮮鋭化における電力量内訳（18.3 節参照）を示す．命令関連 (inst)，命令キャッシュ(icache)，データキャッシュ(dcache)，レジスタ (reg)，演算器 (exec) について比較すると，14 コア通常メニコアの場合，命令キャッシュ，データキャッシュ，その他の合計がほぼ均等であるのに対し，4 コア連結 SAPP では，データキャッシュと演算器の消費電力が大部分を占める理想状態となっている．特に命令関連および命令キャッシュの電力量削減が著しく，次いでデータキャッシュおよびレジスタの電力量削減も大きいことがわかる．14 コア通常メニコアでは，14 コア分の命令関連および命令キャッシュが動作するのに対し，4 コア連結 SAPP では，各段の演算器ネットワーク (POWER_SELRD) において電力を消費するものの，1 コア分の命令キャッシュとデコーダのみが動作し，かつ，アレイ動作中はパワーダウン（命令キャッシュの電力が 1 ／ 3 ）が可能であることによる削減効果が大きい．データキャッシュについては，各段において L0 キャッシュ(POWER_L0) および伝搬機能 (POWER_OOP4) が電力を消費するものの，1 コア分のデータキャッシュのみが稼働することによる削減効果が大きい．さらに，レジスタについても，各段において物理レジスタおよび入力セレクタ (POWER_PREG_XXX) が電力を消費するものの，初

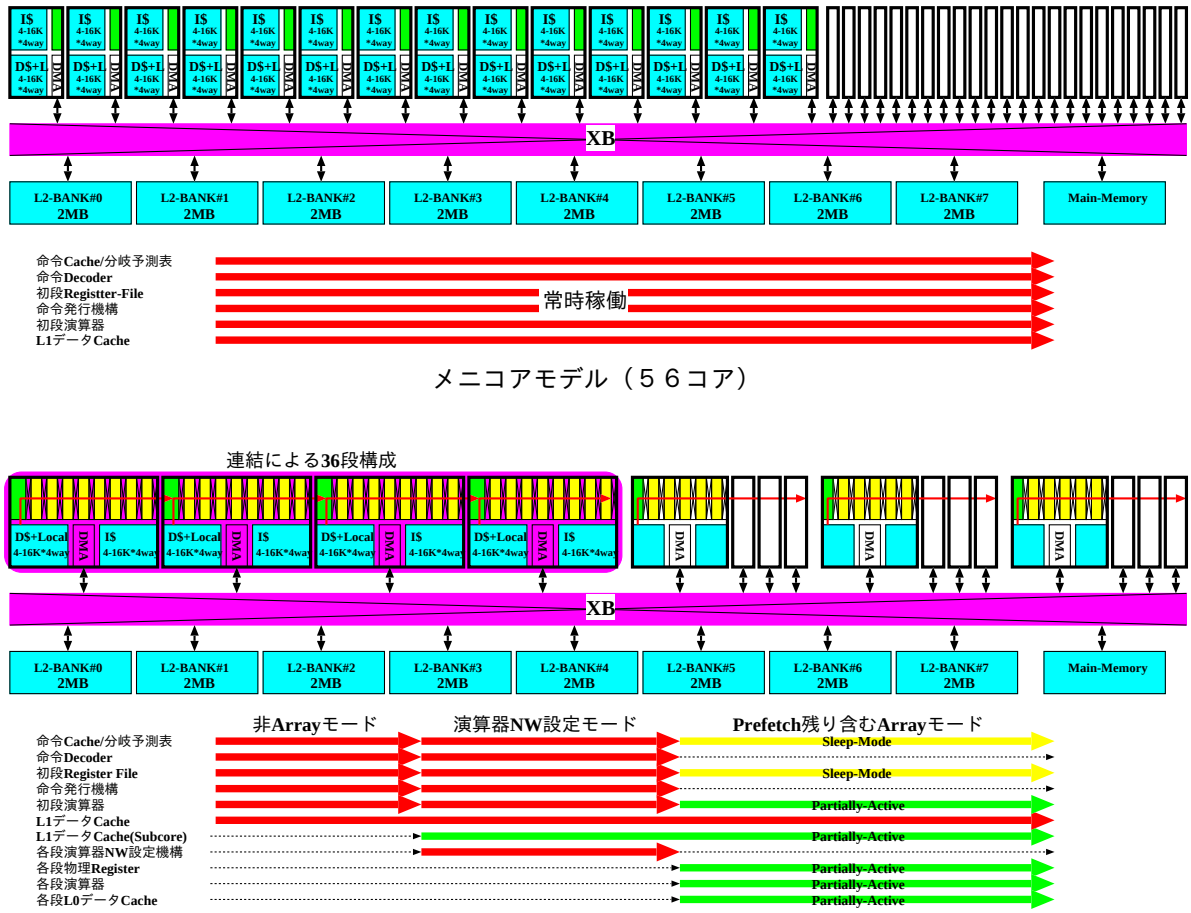


図 20.1: 評価モデル

表 20.1: 画像フィルタの結果 (非アレイ実行とアレイ実行の比較)

	非アレイ実行		PREFETCH のみ		アレイ実行	
フィルタ種別	IPC	電力量	IPC	電力量	IPC	電力量
色補正	0.513	63	1.172	27	15.574	12
フレーム補間 (SAD 計算)	1.190	614	1.452	503	14.248	68
〃 (SAD 最小位置探索)	1.159	206	1.258	190	28.084	17
〃 (補間画像生成)	0.936	160	1.324	113	9.002	14
拡大補間	1.752	309	1.859	291	35.445	25
鮮鋭化	1.179	120	1.694	83	27.336	8
メディアンフィルタ	1.981	57	2.030	56	22.315	7
輪郭抽出	1.132	46	1.194	44	11.951	5
輪郭ノイズ除去	2.013	20	2.038	19	24.043	3
ステレオマッチング (SAD)	1.125	1,850	1.240	1,679	25.416	184
平均	1.308		1.511		21.341	

段のレジスタファイルをパワーダウン（内容保持のために電力は１／３を仮定）できることの効果が大きいと言える．

表 20.2: 画像フィルタの結果 (14 コア通常メニコアと 4 コア連結 SAPP アレイ実行の比較)

	14 コア通常メニコア (1 コア性能の 14 倍を仮定)		4 コア連結 SAPP アレイ実行		
フィルタ種別	IPC	電力量	IPC	電力量	電力量比率
色補正	16.40	27	15.574	12	0.444
フレーム補間 (SAD 計算)	20.32	503	14.248	68	0.135
〃 (SAD 最小位置探索)	17.61	190	28.084	17	0.089
〃 (補間画像生成)	18.53	113	9.002	14	0.123
拡大補間	26.02	291	35.445	25	0.085
鮮鋭化	23.71	83	27.336	8	0.096
メディアンフィルタ	28.42	56	22.315	7	0.125
輪郭抽出	16.71	44	11.951	5	0.113
輪郭ノイズ除去	28.53	19	24.043	3	0.157
ステレオマッチング (SAD)	17.36	1,679	25.416	184	0.109
平均	21.36		21.341		0.147

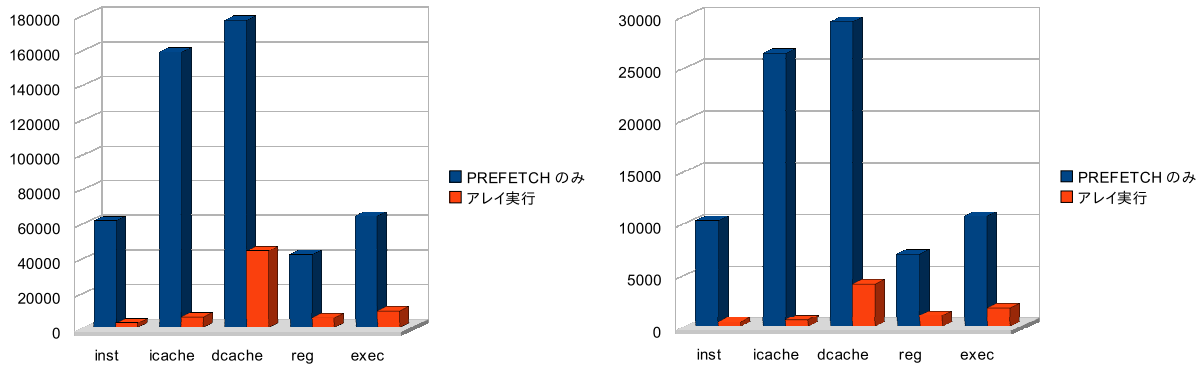


図 20.2: フレーム補間と鮮鋭化の電力量比較 (14 コア通常メニコア対 4 コア連結 SAPP)

20.2 数値計算による評価

20.2.1 各プログラムの最適化手法

tomcatv

```
[tomcatv.c]
  for (I=1; I<N-1; I++) {
    XX = X[J][I+1]-X[J][I-1];
    YX = Y[J][I+1]-Y[J][I-1];
    XY = X[J+1][I]-X[J-1][I];
    YY = Y[J+1][I]-Y[J-1][I];
    A  = 0.25  * (XY*XY+YY*YY);
    B  = 0.25  * (XX*XX+YX*YX);
    C  = 0.125 * (XX*XY+YX*YY);
    PXX = X[J][I+1]-2.0*X[J][I]+X[J][I-1];
    PYY = X[J+1][I]-2.0*X[J][I]+X[J-1][I];
    PXY = X[J+1][I+1]-X[J-1][I+1]-X[J+1][I-1]+X[J-1][I-1];
    RX[J][I]   = A*PXX+B*PYY-C*PXY;
  }
```

```
[tomcatv.s]
    addi sp,#-48,sp
    sti.p fp,@(sp,32);    addi sp,#32,fp
    movsg lr,gr11
    sti gr11,@(fp,8)
    sethi.p #hi(-513*4),gr12; setlo #lo(-513*4),gr12
    sethi.p #hi( 513*4),gr13; setlo #lo( 513*4),gr13
    add.p   gr6,gr12,gr12;    add   gr6,gr13,gr13
    sethi.p #hi((0<<30)|(1<<29)|(0<<28)|(0<<24)|(256<<12)|256),gr11; setlo #lo((256<<12)|256),gr11
    dcpl.p  gr4,gr11,#0; addi.p gr12,0,gr0; addi.p gr6,0,gr0; addi gr13,0,gr0
    sethi.p #hi(-513*4),gr12; setlo #lo(-513*4),gr12
    sethi.p #hi( 513*4),gr13; setlo #lo( 513*4),gr13
    add.p   gr5,gr12,gr12;    add   gr5,gr13,gr13
    sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|(  0<<12)|256),gr11; setlo #lo((  0<<12)|256),gr11
    dcpl.p  gr0,gr11,#1; addi.p gr12,0,gr0; addi.p gr5,0,gr0; addi gr13,0,gr0
sub_loop:
    sethi.p #hi(-513*4),gr12; setlo.p #lo(-513*4),gr12; subicc gr7,#1,gr7,icc0
    sethi.p #hi( 513*4),gr13; setlo.p #lo( 513*4),gr13; beq icc0,0x0,sub_exit
    ldfi   @(gr6,+4),fr0
    ldfi   @(gr6,-4),fr1
    ldf    @(gr6,gr13),fr2
    ldf.p  @(gr6,gr12),fr3;   addi.p gr6,6,gr6;   fsubs   fr0,fr1,fr8
                                                  fmuls   fr8,fr8,fr16 /* YX*YX */
                                                  fsubs   fr2,fr3,fr9
                                                  fmuls.p fr9,fr9,fr17;/* YY*YY */   fmuls fr8,fr9,fr18 /* YX*YY */
    ldfi   @(gr5,+4),fr0
    ldfi   @(gr5,-4),fr1
    ldf.p  @(gr5,gr13),fr2;   addi gr13,4,gr13
    ldf.p  @(gr5,gr12),fr3;   addi.p gr12,4,gr12; fsubs   fr0,fr1,fr10
                                                  fmuls   fr10,fr10,fr19 /* XX*XX */
                              addi.p gr13,-8,gr13;fsubs.p fr2,fr3,fr11;              fadds fr16,fr19,fr16
                              addi.p gr12,-8,gr12;fmuls.p fr11,fr11,fr20;/* XY*XY */ fmuls fr10,fr11,fr21 /* XX*XY */
                                                  fadds.p fr17,fr20,fr17;            fmuls fr16,fr22,fr16 /* A */
                              addi.p gr5,4,gr5;   fadds.p fr18,fr21,fr18;            fmuls fr17,fr22,fr17 /* B */
    ldfi.p   @(gr5, 0),fr4;                                                         fmuls fr18,fr22,fr18 /* C */
    ldf    @(gr5,gr13),fr5
    ldf.p  @(gr5,gr12),fr6;                       fmuls   fr4,fr22,fr4;  /* 2*X[0][I] */
    ldf.p  @(gr5,gr13),fr7;                       fsubs   fr0,fr4,fr19   /* PXX */
    ldf.p  @(gr5,gr12),fr8;                       fadds   fr19,fr1,fr19  /* PXX */
                                                  fsubs.p fr2,fr4,fr20;  /* PYY */   fmuls fr16,fr19,fr16
                                                  fadds   fr20,fr3,fr20  /* PYY */
                                                  fsubs.p fr5,fr6,fr5;   /* PXY */   fmuls fr17,fr20,fr17
                                                  fsubs.p fr5,fr7,fr5;   /* PXY */   fadds fr16,fr17,fr16
                                                  fadds   fr5,fr8,fr5    /* PXY */
                                                                                      fmuls fr18,fr5,fr18
                                                                                      fsubs fr16,fr18,fr0
    stfi.p fr0,@(gr4,0);      addi.p gr4,4,gr4;   bra sub_loop
sub_exit:
    ldi @(fp,8),gr11
    ldi @(fp,0),fp
    jmpl.p @(gr11,gr0);     addi sp,#48,sp
```

tomcatv.cより，X，Yが2次元の入力配列，XX，XY，YX，YY，A，B，C，PXX，PYY，PXYが変数，0.25，0.125が定数，RXが2次元の出力配列である．1回のアレイ実行，すなわちIのループに注目すると，X[J-1]，X[J]，X[J+1]，Y[J-1]，Y[J]，Y[J+1]のプリフェッチが必要である．I-1，I，I+1は同一のWAYからロード可能である．tomcatv.sでは，gr5がX[J]の先頭アドレス，gr6がY[J]の先頭アドレスを示し，gr4

が RX[J] の先頭アドレスを示す．1 回目の dcpl 命令で WAY0 に RX[J] を保存する場所を確保し，WAY1-3 に Y[J-1]，Y[J]，Y[J+1] をプリフェッチし，2 回目の dcpl 命令で WAY5-7 に X[J-1]，X[J]，X[J+1] をプリフェッチする．アレイ実行が終了すると，J が 1 だけインクリメントされ再び同一のコードが実行される．このとき，X[J-1]，X[J]，Y[J-1]，Y[J] に相当するデータはプリフェッチ済みであるので，X[J+1]，Y[J+1] の示すデータのみが新たにプリフェッチされる．（他のアレイ実行等によってキャッシュが変更されていた場合は，すべてプリフェッチし直さなければならない．）

swim-calc1

```
[calc1.c]
CALC1()
  for (J=0; J<N; J++) {
    for (I=0; I<M; I++) {
      CU[J][I+1]  = .5*(P[J][I+1]+P[J][I])*U[J][I+1];
      CV[J+1][I]  = .5*(P[J+1][I]+P[J][I])*V[J+1][I];
      Z[J+1][I+1] = (FSDX*(V[J+1][I+1]-V[J+1][I])-FSDY*(U[J+1][I+1]-U[J][I+1]))
                    /(P[J][I]+P[J][I+1]+P[J+1][I+1]+P[J+1][I]);
      H[J][I]     = P[J][I]+.25
                    *(U[J][I+1]*U[J][I+1]+U[J][I]*U[J][I]+V[J+1][I]*V[J+1][I]+V[J][I]*V[J][I]);
    }
  }
```

```
[calc1.s]
/*                  way,    no raw, dir + , stride, pnum , dnum [words] */
        sethi.p #hi((0<<30)|(1<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr27,gr30
        dcpl.p  gr26,gr30,#0;  addi.p gr6, 0,gr0; addi gr4,0,gr0
        sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr14,gr30
        dcpl.p  gr0, gr30,#0;  addi.p gr5, 0,gr0; addi gr12,0,gr0
        sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr14,gr30
        dcpl.p  gr0, gr30,#1;  addi.p gr11,0,gr0; addi gr10,0,gr0
LOOP1a:
#fr5:P0[0]      fr6:P0[1]       fr7:U0[1]
/*1*/   subicc gr14,#1,gr14,icc0
        beq icc0,0,LOOP1a_exit
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        ldfi @(gr6,4), fr6
        ldfi @(gr6,0), fr5
/*10*/  ldfi @(gr5,4), fr7
        addi.p gr6,#4,gr6       ; fadds fr5,fr6,fr17
        addi.p gr5,#4,gr5       ; fmuls fr17,fr1,fr17
        fmuls fr17,fr7,fr17
        stfi.p fr17, @(gr26,4)  ; addi.p gr26,#4,gr26   ; bra LOOP1a
LOOP1a_exit:
        ...
/*                  way,    no raw, dir + , stride, pnum , dnum [words] */
        sethi.p #hi((0<<30)|(1<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr27,gr30
        dcpl.p  gr26,gr30,#0;  addi.p gr6, 0,gr0; addi gr4,0,gr0
        sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr13,gr30
        dcpl.p  gr0, gr30,#0;  addi.p gr5, 0,gr0; addi gr12,0,gr0
        sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr13,gr30
        dcpl.p  gr0, gr30,#1;  addi.p gr11,0,gr0; addi gr10,0,gr0
LOOP2a:
#fr5:P0[0]      #fr8:V1[0]      fr9:P1[0]
/*1*/   subicc gr14,#1,gr14,icc0
        beq icc0,0,LOOP2a_exit
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        ldfi.p @(gr6,0), fr5    ; addi gr6,#4,gr6
/*9*/   ldfi @(gr4,0), fr9
        addi gr4,#4,gr4
        fadds fr9,fr5,fr17
        fmuls fr17,fr1,fr17
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
/*18*/  mnop0
        ldfi @(gr10,0), fr8
        addi gr10,#4,gr10
        fmuls fr17,fr8,fr17
        stfi.p fr17, @(gr26,0)  ; addi.p gr26,#4,gr26           ; bra LOOP2a
LOOP2a_exit:
```

```
/*                  way,    no raw, dir + , stride, pnum , dnum [words] */
        sethi.p #hi((0<<30)|(1<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr27,gr30
        dcpl.p  gr26,gr30,#0;  addi.p gr6, 0,gr0; addi gr4,0,gr0
        sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr13,gr30
        dcpl.p  gr0, gr30,#0;  addi.p gr5, 0,gr0; addi gr12,0,gr0
        sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr13,gr30
        dcpl.p  gr0, gr30,#1;  addi.p gr11,0,gr0; addi gr10,0,gr0
LOOP3a:
#fr10:V1[1]     fr11:U1[1]      fr12:P1[1]
/*1*/   subicc gr14,#1,gr14,icc0
        beq icc0,0,LOOP3a_exit
        mnop0
        mnop0
        mnop0
        ldfi @(gr6,4), fr6
        ldfi @(gr6,0), fr5
        ldfi.p @(gr4,4), fr12   ; addi gr6,#4,gr6
/*9*/   ldfi.p @(gr4,0), fr9    ; fadds fr6,fr5,fr18
        addi.p gr4,#4,gr4       ; fadds fr18,fr12,fr18
        fadds fr18,fr9,fr18
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        ldfi.p @(gr5,4), fr7    ; addi gr5,#4,gr5
/*18*/  ldfi.p @(gr12,4), fr11  ; addi gr12,#4,gr12
        ldfi @(gr10,4), fr10
        ldfi.p @(gr10,0), fr8   ; fsubs fr11,fr7,fr19
        addi.p gr10,#4,gr10     ; fmuls fr4,fr19,fr19
        fsubs fr10,fr8,fr17
        fmuls fr3,fr17,fr17
        fsubs fr17,fr19,fr17
        fdivs fr17,fr18,fr17
        stfi.p fr17, @(gr26,4)  ; addi.p gr26,#4,gr26   ; bra LOOP3a
LOOP3a_exit:
        ...
/*                  way,    no raw, dir + , stride, pnum , dnum [words] */
        sethi.p #hi((0<<30)|(1<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr27,gr30
        dcpl.p  gr26,gr30,#0;  addi.p gr6, 0,gr0; addi gr4,0,gr0
        sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr13,gr30
        dcpl.p  gr0, gr30,#0;  addi.p gr5, 0,gr0; addi gr12,0,gr0
        sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr30;  setlo #lo(( 0<<12)|0),gr30
        or gr30,gr13,gr30
        dcpl.p  gr0, gr30,#1;  addi.p gr11,0,gr0; addi gr10,0,gr0
LOOP4a:
#fr13:U0[0]     fr14:V0[0]
        subicc gr14,#1,gr14,icc0
        beq icc0,0,LOOP4a_exit
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
/*9*/   ldfi.p @(gr6,0), fr5    ; addi gr6,#4,gr6
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        ldfi @(gr5,4), fr7
        ldfi.p @(gr5,0), fr13   ; addi gr5,#4,gr5
/*18*/                          fmuls fr7,fr7,fr17
        ldfi.p @(gr10,0), fr8   ; fmuls fr13,fr13,fr18
        addi.p gr10,#4,gr10     ; fadds fr17,fr18,fr17
        ldfi.p @(gr11,0), fr14  ; fmuls fr8,fr8,fr18
        addi.p gr11,#4,gr11     ; fadds fr17,fr18,fr17
                                  fmuls fr14,fr14,fr18
                                  fadds fr17,fr18,fr17
                                  fmuls fr17,fr2,fr17
                                  fadds fr5,fr17,fr17
        stfi.p fr17, @(gr26,0)  ; addi.p gr26,#4,gr26   ; bra LOOP4a
LOOP4a_exit:
```

calc1.c より，P，U，V が 2 次元の入力配列，FSDX，FSDY，0.25 が定数であり，CU，CV，Z，H が 2 次元の出力配列である．SAPP では出力配列は 1 つに制限されているため，calc1 のように 4 つの出力配列が必要となる場合には，4 回のアレイ実行が必要である．このときの分割方法には，以下の calc1_a と calc1_b の 2 通りが考えられる．ここで各アレイ実行の入力配列に注目すると大部分が共通しており，さらに 4 回の

アレイ実行すべてに必要な入力配列がL1 キャッシュサイズよりも小さいためCALC1_a の方が適している．つまり，CU に関するアレイ実行を行う前にP，U，V をプリフェッチを完了すれば，CV，Z，H に関するアレイ実行の前に入力データに対するプリフェッチを行う必要はない．

```
CALC1_a()
for (J=0; J<N; J++) {
  for (I=0; I<M; I++) {
    CU[J][I+1] = .5*(P[J][I+1]+P[J][I])*U[J][I+1];
  }
  for (I=0; I<M; I++) {
    CV[J+1][I] = .5*(P[J+1][I]+P[J][I])*V[J+1][I];
  }
  for (I=0; I<M; I++) {
    Z[J+1][I+1] = (FSDX*(V[J+1][I+1]-V[J+1][I])-FSDY*(U[J+1][I+1]-U[J][I+1]))
                  /(P[J][I]+P[J][I+1]+P[J+1][I+1]+P[J+1][I]);
  }
  for (I=0; I<M; I++) {
    H[J][I] = P[J][I]+.25
              *(U[J][I+1]*U[J][I+1]+U[J][I]*U[J][I]+V[J+1][I]*V[J+1][I]+V[J][I]*V[J][I]);
  }
}
```

```
CALC1_b()
for (J=0; J<N; J++) {
  for (I=0; I<M; I++) {
    CU[J][I+1] = .5*(P[J][I+1]+P[J][I])*U[J][I+1];
  }
}
for (J=0; J<N; J++) {
  for (I=0; I<M; I++) {
    CV[J+1][I] = .5*(P[J+1][I]+P[J][I])*V[J+1][I];
  }
}
for (J=0; J<N; J++) {
  for (I=0; I<M; I++) {
    Z[J+1][I+1] = (FSDX*(V[J+1][I+1]-V[J+1][I])-FSDY*(U[J+1][I+1]-U[J][I+1]))
                  /(P[J][I]+P[J][I+1]+P[J+1][I+1]+P[J+1][I]);
  }
}
for (J=0; J<N; J++) {
  for (I=0; I<M; I++) {
    H[J][I] = P[J][I]+.25
     *(U[J][I+1]*U[J][I+1]+U[J][I]*U[J][I]+V[J+1][I]*V[J+1][I]+V[J][I]*V[J][I]);
  }
}
```

calc1.s では，gr6，gr4 にP[J]，P[J+1]，gr5，gr12 にU[J]，U[J+1]，gr11，gr10 にV[J]，V[J+1] のアドレスが，gr26 にはCU，CV，Z，H のいずれかのアドレスが格納されている．4 回のアレイ実行，すなわちLOOP1a からLOOP4a の計 4 個のI のループに注目すると，P[J]，P[J+1]，U[J]，U[J+1]，V[J]，V[J+1] のプリフェッチが必要である．4 回のアレイ実行が記述されているが，プリフェッチ内容はすべてのアレイ実行で共通で，1 回目のdcpl 命令でWAY0 に出力を保存する場所を確保し，WAY1-2 にP[J]，P[J+1] をプリフェッチ，2 回目のdcpl 命令でWAY5-6 にU[J]，U[J+1]，3 回目でWAY9-10 にV[J]，V[J+1] をプリフェッチする．前述のように 2 番目以降のアレイ実行ではプリフェッチ済みであることを自動的に検出し，即座にプリフェッチは完了する．ここで，WAY1-2 の内容 (P) は 1-9 段目，WAY5-6 の内容 (U) は 10-18 段目，WAY9-10 の内容 (V) は 19 段目以降からしか読み込むことができない点に注意が必要である．また，ロードと演算の間が離れているとその間を伝搬するためのレジスタを確保しなければならないため，nop を入れる場所にも注意を要する．アレイ実行が終了すると，J が 1 だけインクリメントされ再び同一のコードが実行される．このとき，P[J]，U[J]，V[J] に相当するデータはプリフェッチ済みであるので，P[J+1]，U[J+1]，V[J+1] の示すデータのみが新たにプリフェッチされる．

swim-calc2

```
[calc2.c]
CALC2()
  for (J=0; J<N; J++) {
    for (I=0; I<M; I++) {
      UNEW[J][I+1] = UOLD[J][I+1]
                     +TDTS8*(Z[J+1][I+1]+Z[J][I+1])*(CV[J+1][I+1]+CV[J+1][I]+CV[J][I]+CV[J][I+1])
                     -TDTSDX*(H[J][I+1]-H[J][I]);
      VNEW[J+1][I] = VOLD[J+1][I]
                     -TDTS8*(Z[J+1][I+1]+Z[J+1][I])*(CU[J+1][I+1]+CU[J+1][I]+CU[J][I]+CU[J][I+1])
                     -TDTSDY*(H[J+1][I]-H[J][I]);
      PNEW[J][I]   = POLD[J][I]
                     -TDTSDX*(CU[J][I+1]-CU[J][I])
                     -TDTSDY*(CV[J+1][I]-CV[J][I]);
    }
  }
```

```
[calc2.s]
/*                  way     no raw, dir + , stride, pnum,   dnum [words] */
        sethi.p #hi((0<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr12,gr15
        dcpl.p  gr27,gr15,#0    ;  addi.p gr5,0,gr0;    addi gr4,0,gr0
        sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#0    ;  addi.p gr30,0,gr0;   addi gr13,0,gr0
        sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#0    ;  addi.p gr10,0,gr0;   addi.p gr6,0,gr0; addi gr28,0,gr0;
        sethi.p #hi((3<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#1    ;  addi.p gr11,0,gr0;   addi gr16,0,gr0
LOOP21a:
/*1*/   subicc gr7,#1,gr7,icc0
        beq icc0,0,LOOP21a_exit
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        ldfi.p @(gr4,4), fr4    ; addi gr4,#4,gr4
/*9*/   ldfi.p @(gr5,4), fr5    ; addi gr5,#4,gr5
        mnop0
        fadds fr4,fr5,fr17
        fmuls fr17,fr1,fr17
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
/*18*/  mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        ldfi @(gr6,0), fr4      ;
        ldfi.p @(gr6,4), fr5    ; addi gr6,#4,gr6
        ldfi @(gr10,0), fr6     ;
        ldfi.p @(gr10,4), fr7   ; addi.p gr10,#4,gr10   ; fadds fr4,fr5,fr19
/*27*/  ldfi.p @(gr28,4), fr4   ; addi.p gr28,#4,gr28   ; fadds fr19,fr6,fr19
        ldfi.p @(gr11,0), fr8   ;                         fadds fr19,fr7,fr19
        ldfi.p @(gr11,4), fr9   ;                         fmuls fr17,fr19,fr17
        addi gr11,#4,gr11       ;                         fadds fr4,fr17,fr17
        fsubs fr9,fr8,fr18
        fmuls fr18,fr2,fr18
        fsubs fr17,fr18,fr17
        stfi.p fr17, @(gr27,4)  ; addi.p gr27,#4,gr27   ; bra LOOP21a
LOOP21a_exit:
```

```
/*                  way     no raw, dir + , stride, pnum,   dnum [words] */
        sethi.p #hi((0<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr12,gr15
        dcpl.p  gr27,gr15,#0    ;  addi.p gr5,0,gr0;    addi gr4,0,gr0
        sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#0    ;  addi.p gr30,0,gr0;   addi gr13,0,gr0
        sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#0    ;  addi.p gr10,0,gr0;   addi.p gr6,0,gr0; addi gr28,0,gr0;
        sethi.p #hi((3<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#1    ;  addi.p gr11,0,gr0;   addi gr16,0,gr0
LOOP22a:
/*1*/   subicc gr7,#1,gr7,icc0
        beq icc0,0,LOOP22a_exit
        mnop0
...
/*9*/   mnop0
...
        mnop0
        ldfi @(gr30,4), fr4
/*18*/  ldfi.p @(gr30,0), fr5   ; addi gr30,#4,gr30
        ldfi.p @(gr6,0), fr7    ; addi gr6,#4,gr6
        ldfi.p @(gr10,0), fr6   ; addi gr10,#4,gr10     ; fsubs fr4,fr5,fr17
                                                          fmuls fr17,fr2,fr17
        ldfi.p @(gr28,0), fr8   ; addi gr28,#4,gr28     ; fsubs fr7,fr6,fr18
        fmuls fr18,fr3,fr18
        fsubs fr8,fr17,fr17
        fsubs fr17,fr18,fr17
        stfi.p fr17, @(gr27,0)  ; addi.p gr27,#4,gr27   ; bra LOOP22a
LOOP22a_exit:
...
/*                  way     no raw, dir + , stride, pnum,   dnum [words] */
        sethi.p #hi((0<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr12,gr15
        addi gr15,1,gr15
        dcpl.p  gr27,gr15,#0    ;  addi.p gr5,0,gr0;    addi gr4,0,gr0
        sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#0    ;  addi.p gr30,0,gr0;   addi gr13,0,gr0
        sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#0    ;  addi.p gr10,0,gr0;   addi.p gr6,0,gr0; addi gr28,0,gr0;
        sethi.p #hi((3<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|0),gr15    ;  setlo #lo(( 0<<12)|0),gr15
        or gr15,gr7,gr15
        addi gr15,1,gr15
        dcpl.p  gr0, gr15,#1    ;  addi.p gr11,0,gr0;   addi gr16,0,gr0
LOOP23a:
/*1*/   subicc gr7,#1,gr7,icc0
        beq icc0,0,LOOP23a_exit
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
        ldfi @(gr4,4), fr4
/*9*/   ldfi @(gr4,0), fr5
        addi gr4,#4,gr4
        fadds fr4,fr5,fr17
        fmuls fr17,fr1,fr17
        mnop0
        mnop0
        ldfi @(gr13,4), fr4
        ldfi.p @(gr13,0), fr5   ; addi gr13,#4,gr13
        ldfi @(gr30,0), fr6
/*18*/  ldfi.p @(gr30,4), fr7   ; addi.p gr30,#4,gr30   ; fadds fr4,fr5,fr18
        fadds fr18,fr6,fr18
        fadds fr18,fr7,fr18
        fmuls fr17,fr18,fr17
        mnop0
        mnop0
        mnop0
        mnop0
        mnop0
/*27*/  ldfi.p @(gr28,0), fr4   ; addi gr28,#4,gr28
        ldfi.p @(gr16,0), fr5   ; addi gr16,#4,gr16
        ldfi.p @(gr11,0), fr8   ; addi gr11,#4,gr11
        fsubs fr4,fr17,fr17
        fsubs fr5,fr8,fr18
        fmuls fr18,fr3,fr18
        fsubs fr17,fr18,fr17
        stfi.p fr17, @(gr27,0)  ; addi.p gr27,#4,gr27   ; bra LOOP23a
LOOP23a_exit:
```

calc2.c より，UOLD，VOLD，PLOD，CU，CV，Z，H が 2 次元の入力配列，TDTS8，TDTSDX，TDTSDY が定数であり，UNEW，VNEW，PNEW が 2 次元の出力配列である．出力が 3 配列であるので，swim-calc2 と同様に 1 イタレーションあたり 3 回のアレイ実行が必要である．

UOLD，VOLD，POLD の再利用性が無いため，これらのプリフェッチには同一の WAY を割り当てる．残りの入力データには再利用性があり，3 回のアレイ実行で必要となる配列は CU[J], CU[J+1], CV[J], CV[J+1], Z[J], Z[J+1], H[J], H[J+1] である．プリフェッチ済みかどうかの判定は同一段に接続された WAY 間でしか行われないため，再利用性を最大限生かすためには同一の配列は同一段に接続された WAY に割り当てることが望ましい．したがって，この場合は 4 つの入力配列を各 WAY に割り当てる必要があるが，特に WAY13-15 については，第 28 段目に接続されているため，変数の割り当てによっては演算器段数が不足する可能性がある．ここでは，WAY1-2 に Z[J]，Z[J+1]，WAY5-6 に CU[J]，CU[J+1]，WAY9-10 に CV[J]，CV[J+1]，WAY13-14 に H[J]，H[J+1] を割り当てた．UOLD，VOLD，POLD は空き WAY に割り当てることになるが，伝搬コストと残り演算器段数のバランスを考慮し WAY11 に割り当てた．

calc2.s では，gr5，gr4 に Z[J]，Z[J+1]，gr30，gr13 に CU[J]，CU[J+1]，gr10，gr6 に CV[J]，CV[J+1]，gr11，gr16 に H[J]，H[J+1] のアドレス，gr28 には UOLD，POLD，VOLD のいずれか，gr27 に UNEW，PNEW，VNEW のいずれかのアドレスが格納されている．以降は calc1 と同様である．

mgrid

```
[mgrid.c]
  for (I1=1; I1<=n-2; I1++) {
    /* 金太郎飴の断面図（奥行き方向は I1）      */
    /* [I3-1][I2-1] [I3-1][I2  ] [I3-1][I2+1] */
    /* [I3  ][I2-1] [I3  ][I2  ] [I3  ][I2+1] */
    /* [I3+1][I2-1] [I3+1][I2  ] [I3+1][I2+1] */
    /* A[0]*(中央)                                                                    */
    /* A[1]*(中央+中央+ 左 + 右 + 上 + 下)                                 上下左右各 1 方向          */
    /* A[2]*( 左 + 左 + 右 + 右 +左上+右上+左下+右下+ 上 + 下 + 上 + 下) 上下左右各 2 方向 斜め各 1 方向 */
    /* A[3]*(左上+左上+右上+右上+左下+左下+右下+右下)                       斜め各 2 方向          */

    /* 全部で 9way（ルービックキューブの全コマ 27 要素）が同時に必要 */
    /* I2+1 により，新たに，[I3-1][I2+1]，[I3][I2+1]，[I3+1][I2+1] の 3 ラインのプリフェッチが必要 */

    R[I3][I2][I1] = V[I3][I2][I1]
      - A[0]*(U[I3  ][I2  ][I1  ])
      - A[1]*(U[I3  ][I2  ][I1-1]+U[I3  ][I2  ][I1+1]+U[I3  ][I2-1][I1  ]
             +U[I3  ][I2+1][I1  ]+U[I3-1][I2  ][I1  ]+U[I3+1][I2  ][I1  ])
      - A[2]*(U[I3  ][I2-1][I1-1]+U[I3  ][I2-1][I1+1]+U[I3  ][I2+1][I1-1]+U[I3  ][I2+1][I1+1]
             +U[I3-1][I2-1][I1  ]+U[I3-1][I2+1][I1  ]+U[I3+1][I2-1][I1  ]+U[I3+1][I2+1][I1  ]
             +U[I3-1][I2  ][I1-1]+U[I3+1][I2  ][I1-1]+U[I3-1][I2  ][I1+1]+U[I3+1][I2  ][I1+1])
      - A[3]*(U[I3-1][I2-1][I1-1]+U[I3-1][I2-1][I1+1]+U[I3-1][I2+1][I1-1]+U[I3-1][I2+1][I1+1]
             +U[I3+1][I2-1][I1-1]+U[I3+1][I2-1][I1+1]+U[I3+1][I2+1][I1-1]+U[I3+1][I2+1][I1+1]);
  }
```

```
[mgrid.s]
    addi sp,#-48,sp
    sti.p fp,@(sp,32);    addi sp,#32,fp
    movsg lr,gr11
    sti gr11,@(fp,8)
    addi gr0,65,gr7
    sethi.p #hi(-4356*4),gr12; setlo #lo(-4356*4),gr12
    sethi.p #hi( 4356*4),gr13; setlo #lo( 4356*4),gr13
    add.p   gr5,gr12,gr12;     add   gr5,gr13,gr13
    sethi.p #hi((0<<30)|(1<<29)|(0<<28)|(0<<24)|(64<<12)|64),gr6; setlo #lo((64<<12)|64),gr6
    dcpl.p  gr4,gr6,#0; addi.p gr12,-66*4,gr0; addi.p gr12,0,gr0; addi gr12,+66*4,gr0
    sethi.p #hi((1<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|64),gr6; setlo #lo(( 0<<12)|64),gr6
    dcpl.p  gr0,gr6,#0; addi.p gr13,-66*4,gr0; addi.p gr13,0,gr0; addi gr13,+66*4,gr0
    sethi.p #hi((2<<30)|(0<<29)|(0<<28)|(0<<24)|( 0<<12)|64),gr6; setlo #lo(( 0<<12)|64),gr6
    dcpl.p  gr0,gr6,#1; addi.p gr5, -66*4,gr0; addi.p gr5, 0,gr0; addi gr5, +66*4,gr0
resid_loop:
    ldfi.p @(gr12,-66*4-4),fr0;     subicc gr7,#1,gr7,icc0
    ldfi.p @(gr12,-66*4+4),fr1;     beq icc0,0x0,resid_exit
    ldfi.p @(gr12,+66*4-4),fr2;     nop
    ldfi.p @(gr12,+66*4+4),fr3;                         fadds   fr0,fr1,fr13
    ldfi.p @(gr12,-66*4  ),fr4;                         fadds   fr2,fr13,fr13
    ldfi.p @(gr12,+66*4  ),fr5;                         fadds   fr3,fr13,fr13
    ldfi.p @(gr12,     -4),fr6;     nop
    ldfi.p @(gr12,     +4),fr7;                         fadds   fr4,fr5,fr12
    ldfi.p @(gr12,      0),fr11;    addi.p gr12,4,gr12; fadds   fr6,fr12,fr12
    ldfi.p @(gr13,-66*4-4),fr0;                         fadds   fr7,fr12,fr12
    ldfi.p @(gr13,-66*4+4),fr1;     nop
    ldfi.p @(gr13,+66*4-4),fr2;                         fadds   fr0,fr13,fr13
    ldfi.p @(gr13,+66*4+4),fr3;                         fadds   fr1,fr13,fr13
    ldfi.p @(gr13,-66*4  ),fr4;                         fadds   fr2,fr13,fr13
    ldfi.p @(gr13,+66*4  ),fr5;                         fadds   fr3,fr13,fr13
    ldfi.p @(gr13,     -4),fr6;                         fadds.p fr4,fr12,fr12; fmuls fr13,fr19,fr13
    ldfi.p @(gr13,     +4),fr7;                         fadds   fr5,fr12,fr12
    ldfi.p @(gr13,      0),fr8;     addi.p gr13,4,gr13; fadds   fr6,fr12,fr12
    ldfi.p @(gr5, -66*4-4),fr0;                         fadds   fr7,fr12,fr12
    ldfi.p @(gr5, -66*4+4),fr1;                         fadds   fr8,fr11,fr11
    ldfi.p @(gr5, +66*4-4),fr2;                         fadds   fr0,fr12,fr12
    ldfi.p @(gr5, +66*4+4),fr3;                         fadds   fr1,fr12,fr12
    ldfi.p @(gr5, -66*4  ),fr4;                         fadds   fr2,fr12,fr12
    ldfi.p @(gr5, +66*4  ),fr5;                         fadds   fr3,fr12,fr12
    ldfi.p @(gr5,      -4),fr6;                         fadds.p fr4,fr11,fr11; fmuls fr12,fr18,fr12
    ldfi.p @(gr5,      +4),fr7;                         fadds   fr5,fr11,fr11
    ldfi.p @(gr5,       0),fr10;    addi.p gr5,4,gr5;   fadds   fr6,fr11,fr11
                                                        fadds   fr7,fr11,fr11
                                                        fadds.p fr12,fr13,fr13; fmuls fr11,fr17,fr11
    ldfi.p @(gr4,       0),fr0;                         fadds.p fr11,fr13,fr13; fmuls fr10,fr16,fr10
                                                        fadds   fr10,fr13,fr13
                                                        fsubs   fr0,fr13,fr0
    stfi.p fr0,@(gr4,0);            addi.p gr4,4,gr4;   bra resid_loop
resid_exit:
    ldi @(fp,8),gr11
    ldi @(fp,0),fp
    jmpl.p @(gr11,gr0);     addi sp,#48,sp
```

表 20.3: 数値計算の結果 (非アレイ実行とアレイ実行の比較)

	非アレイ実行		PREFETCH のみ		アレイ実行	
プログラム種別	IPC	電力量	IPC	電力量	IPC	電力量
tomcatv	0.946	114	1.160	93	18.405	11
swim-calc1	0.417	1,464	0.501	1,218	7.807	128
swim-calc2	0.381	1,652	0.558	1,128	12.750	110
mgrid-resid64	1.336	6.4	1.349	6.4	14.683	1.6
mgrid-resid128	0.700	48	0.695	48	17.839	5
平均	0.756		0.852		14.296	

表 20.4: 数値計算の結果 (14 コア通常メニコアと 4 コア連結 SAPP アレイ実行の比較)

	14 コア通常メニコア（1 コア性能の 14 倍を仮定）		4 コア連結 SAPP アレイ実行		
プログラム種別	IPC	電力量	IPC	電力量	電力量比率
tomcatv	16.24	93	18.405	11	0.118
swim-calc1	7.01	1,218	7.807	128	0.105
swim-calc2	7.81	1,128	12.750	110	0.097
mgrid-resid64	18.88	6.4	14.683	1.6	0.250
mgrid-resid128	13.12	48	17.839	5	0.104
平均	11.92		14.296		0.134

mgrid.c より，`V`，`U` が 3 次元の入力配列，`A[0-3]` が定数であり，`R` が 3 次元の出力配列である．1 回のアレイ実行に注目すると，入力配列が 3 次元であるため，プリフェッチは `V[I3][I2]` と `U[I3-1][I2-1]`，`U[I3-1][I2]`，`U[I3-1][I2+1]`，`U[I3][I2-1]`，`U[I3][I2]`，`U[I3][I2+1]`，`U[I3+1][I2-1]`，`U[I3+1][I2]`，`U[I3+1][I2+1]` の合計 10WAY 必要である．ここでは WAY1-3 に `U[I3-1][I2-1]`，`U[I3-1][I2]`，`U[I3-1][I2+1]`，WAY5-7 に `U[I3][I2-1]`，`U[I3][I2]`，`U[I3][I2+1]`，WAY9-11 に `U[I3+1][I2-1]`，`U[I3+1][I2]`，`U[I3+1][I2+1]` を割り当てる．以降は他のプログラムと同様である．

20.2.2 評価結果

tomcatv，swim，mgrid の実行結果を表 20.3 および表 20.4 に示す．評価時の性能パラメタは表 16.1 の通りである．画像処理と同様に，単一スレッドどうしの比較では，シングルコアと SAPP（36 段構成）の性能比は 1 ： 1 6，単一スレッドにより動作可能な SAPP の回路規模あたり性能比は 1 ： 1，回路規模あたり消費電力比は 8 ： 1 であると言える．なお，mgrid において「非アレイ実行」と「PREFETCH のみ」の性能に大差がないのは,「アレイ実行」では way5-7,9-11 に対応する配列要素が,「PREFETCH のみ」では PREFETCH 対象外であり，way0 において発生するキャッシュミスが効果を薄めるためと考えられる．

図 20.3 に，tomcatv と mgrid128 における電力量内訳（18.3 節参照）を示す．画像処理の場合と同様に数値計算においても，14 コア通常メニコアの場合，命令キャッシュ，データキャッシュ，その他の合計がほぼ均等であるのに対し，4 コア連結 SAPP では，データキャッシュと演算器の消費電力が大部分を占める理想状態となっていることがわかる．

また，表 20.5 に，4 および 64 スレッド分割による実行結果を示す．mgrid-resid128x4PE は，PE 間連携機能（ICPL 命令）により mgrid-resid 全体を 4 スレッドにて実行した場合の測定結果（非アレイ実行は各 PE の初段のみによる 4 スレッド実行，アレイ実行は 36 段構成の PE を 4 台投入した 4 スレッド実行）である．mgrid-resid128x64PE は，同様に 64 スレッド（アレイ実行は，サブコア数 256，一般的メニコアの 1024 コア構成と同規模）にて実行した．各々，L1way 長，L2 バンク数，インタリーブ間隔を変化させている．

まず，L1way 長が 4KB の場合，4 スレッド実行，64 スレッド実行のいずれにおいても「非アレイ実行」

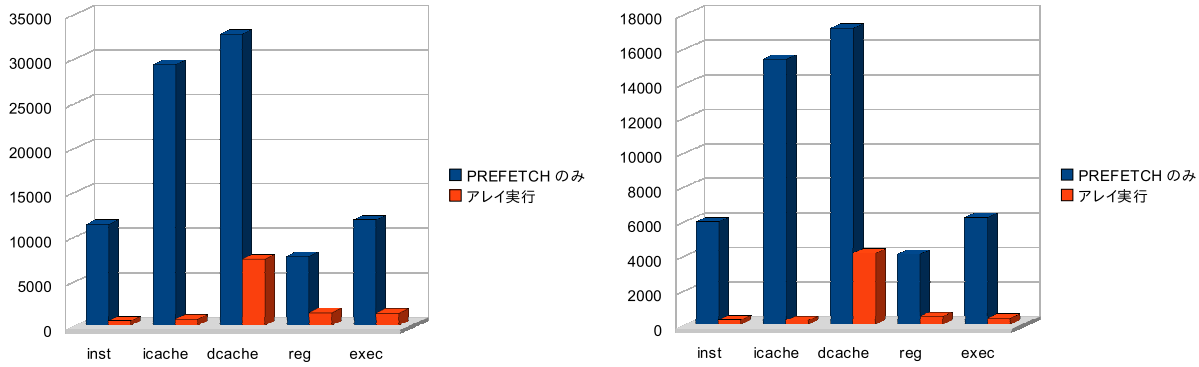


図 20.3: tomcatv と mgrid128 の電力量比較 (14 コア通常メニコア対 4 コア連結 SAPP)

表 20.5: 数値計算の結果 (4-64 スレッド分割における非アレイ実行とアレイ実行の比較)

	非アレイ実行		PREFETCH のみ		アレイ実行	
プログラム種別	IPC	電力量	IPC	電力量	IPC	電力量
mgrid-resid128x4PE						
(L1way 長:16K,L2bank:8way/64B)	1.337*4	812*4	1.333*4	814*4	15.822*4	198*4
(L1way 長: 8K,L2bank:8way/64B)	1.335*4	813*4	1.332*4	814*4	15.887*4	198*4
(L1way 長: 4K,L2bank:8way/64B)	0.805*4	1,349*4	0.826*4	1,313*4	15.862*4	198*4
(L1way 長:16K,L2bank:8way/4K)	1.337*4	812*4	1.333*4	814*4	15.617*4	201*4
(L1way 長: 8K,L2bank:8way/4K)	1.335*4	813*4	1.331*4	815*4	15.202*4	207*4
(L1way 長: 4K,L2bank:8way/4K)	0.813*4	1,336*4	0.817*4	1,328*4	15.129*4	207*4
(L1way 長: 4K,L2bank:256way/4K)	0.843*4	1,289*4	0.839*4	1,293*4	16.616*4	187*4
mgrid-resid128x64PE						
(L1way 長:16K,L2bank:8way/64B)	0.897*64	151*64	0.891*64	152*64	1.914*64	88*64
(L1way 長: 8K,L2bank:8way/64B)	0.895*64	151*64	0.891*64	152*64	1.915*64	88*64
(L1way 長: 4K,L2bank:8way/64B)	0.211*64	643*64	0.203*64	668*64	1.913*64	88*64
(L1way 長:16K,L2bank:8way/4K)	0.899*64	151*64	0.855*64	158*64	1.899*64	89*64
(L1way 長: 8K,L2bank:8way/4K)	0.897*64	151*64	0.841*64	161*64	1.890*64	90*64
(L1way 長: 4K,L2bank:8way/4K)	0.212*64	641*64	0.200*64	679*64	1.902*64	89*64
(L1way 長: 4K,L2bank:256way/4K)	0.668*64	203*64	0.666*64	203*64	1.935*64	86*64

および「PREFETCH のみ」では，深刻な性能低下が見られる．一方,「アレイ実行」では，ほとんど差はない．L1way 長の不足に対しては，複数サブコアの Local メモリに対する PREFETCH 機能が極めて有効であることがわかる．次に，L1way 長を 4KB として L2 バンク数を 8 と 256 に変化させたところ，64 スレッド実行かつバンク数 8 では,「非アレイ実行」および「PREFETCH のみ」に深刻な性能低下が見られた．64 スレッドの場合，L2 リクエストが多数競合していると推測される．また,「アレイ実行」では，L2bank のインタリーブ単位を 64B と 4KB に変化させた際の性能に差があり，64B のほうが若干高性能である（L2bank=256way に匹敵）．これはバンク参照が均一化するためと考えられるが，キャッシュライン長を超えてバースト転送できる機会の有効利用を考えた場合，性能が逆転する可能性がある．

さて，同一回路規模での比較を行うために，表 20.6 に，64 スレッド分割の「PREFETCH のみ」と 4 スレッド分割の「アレイ実行」を再掲した．なお，スレッドあたりの回路規模は厳密には１：１４なので，56 スレッド分割の「PREFETCH のみ」と比較すべきであるが，データ分割を簡単化するために 64 スレッド分割の「PREFETCH のみ」と比較している．L1way 長が 8KB 以上の場合，性能比では１０：１１，電力比では１２：１となっている．14%面積増の 64 スレッドを投入しても「PREFETCH のみ」が 4 スレッド分割の「アレイ実行」に性能で勝てないのは，L2 におけるバンク競合のためと推測される．また，このために電力比も大きくなっていると考えられる．

表 20.6: 数値計算の結果 (4-64 スレッド分割における非アレイ実行とアレイ実行の比較)

	64 スレッド PREFETCH のみ		4 スレッドアレイ実行		
プログラム種別	IPC	電力量	IPC	電力量	電力量比率
(L1way 長:16K,L2bank:8way/64B)	0.891*64	152*64	15.822*4	198*4	0.081
(L1way 長: 8K,L2bank:8way/64B)	0.891*64	152*64	15.887*4	198*4	0.081
(L1way 長: 4K,L2bank:8way/64B)	0.203*64	668*64	15.862*4	198*4	0.018
(L1way 長:16K,L2bank:8way/4K)	0.855*64	158*64	15.617*4	201*4	0.079
(L1way 長: 8K,L2bank:8way/4K)	0.841*64	161*64	15.202*4	207*4	0.080
(L1way 長: 4K,L2bank:8way/4K)	0.200*64	679*64	15.129*4	207*4	0.019

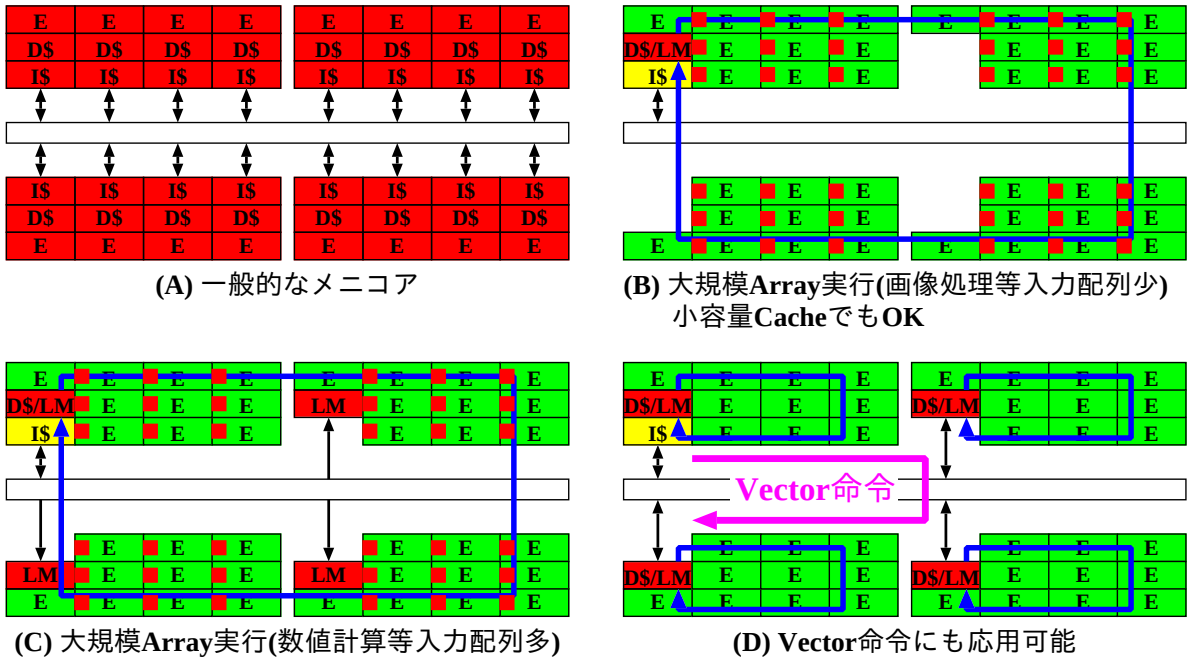


図 20.4: 電力消費モデル総覧

20.3 結論

図 20.4 に電力消費モデルを総覧する．本研究では，今後，プロセッサの主流となることが予想されているメニコア構成に関し，既存命令セット，Local メモリと外部データ・プリフェッチ機構，さらに，パワーゲーティングとの親和性を兼ね備えるアクセラレータ構成方式について詳細に検討し，性能評価モデル，回路遅延検証モデル，および，回路規模・電力評価モデルを用いた定量的評価を行った．特に，複数コアに分散配置される演算器，物理レジスタ，L1 キャッシュおよび Local メモリを線形接続して既存命令セットを動作させる演算方式，プリフェッチ時間に隠蔽可能な命令写像方式，小規模物理レジスタおよび小規模キャッシュを用いて本来のレジスタ/キャッシュ空間を仮想的に提供できる伝搬方式を新たに考案し，有効性を実証した．単一スレッドの比較では，単一コア構成に比べて性能で１４倍，また，同一回路規模の比較では，アクセラレータを搭載しない場合（図 20.4(A) に対応）に比べて，性能は同等，消費電力は，画像処理や一般的な数値計算では１／８（図 20.4(B) に対応），１本の出力配列数が必要とする入力配列数が多い色補正や mgrid では１／４（図 20.4(C) に対応）であることを明らかにした．なお，Vector 命令を利用する場合は，各段の L0 キャッシュに必要な回路を縮小することができるため，後者のプログラムも１／８（図 20.4(D) に対応）に改善できると考えられる．全体としては，図 20.5 に示すように，命令キャッシュのパワーダウンによる電力削減効果が大きく，次いでデータキャッシュの電力削減効果，さらに，レジスタの電力削減効

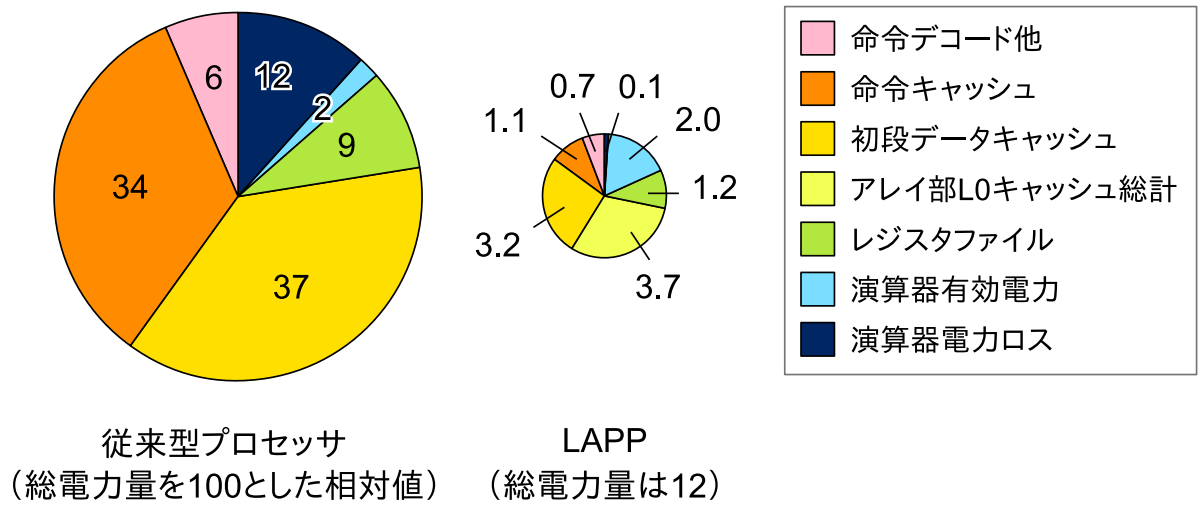


図 20.5: 電力削減の内訳

果も大きく寄与することがわかった．本構成手法が，現状のメニコアが抱える課題，すなわち，コア全体の演算性能に対して外部データ供給能力が脆弱である課題を解決する有効な手段として，今後，広く利用されることを期待する．本構成手法の特長をまとめると，以下の通りである．

- 複数サブコアのレジスタ・演算器・Local メモリをスケーラブルに連結し，既存命令列のシングルスレッド実行にて 50 超の IPC を達成する命令写像およびストリーム処理
- アレイ動作時フロントエンド停止機構および細粒度パワーゲーティングによる低消費電力化
- 多機能 PREFETCH 命令，自動 way ローテーション機構，冗長 PREFETCH 抑止による複数サブコアの L1 キャッシュと Local メモリの効果的制御
- L2 キャッシュを経由しないコア間連携機構および自動マルチスレッド化機構によるスケーラブルな自動並列化

20.4 関連発表

1. 森浩大, 岩上拓矢, 吉村和浩, 中田尚, 中島康彦: ”演算器アレイ型アクセラレータのための命令変換手法の検討”, SWoPP2010(Vol.2010-ARC-190 No.26 2010/8/4), pp.1-6, Aug. (2010)
2. 岩上拓矢, 吉村和浩, 上利宗久, 中田尚, 中島康彦: ”プログラマビリティを備える低電力アクセラレータの提案と評価”, 先進的計算基盤システムシンポジウム SACSIS2010 論文集 (poseter), May. (2010)
3. Takuya Iwakami, Munehisa Agari, Kazuhiro Yoshimura, Takashi Nakada, Yasuhiko Nakashima: ”Area Optimization of FU Array in Low-Power Accelerators”, IEEE Symposium on Low-Power and High-Speed Chips 2010 (poster), Apr. (2010)
4. 吉村和浩, 上利宗久, 中田尚, 中島康彦: ”演算器アレイ型プロセッサのための命令スケジューラの設計と評価”, 信学技報, Vol.109, No.474, pp.511-516, Mar. (2010)
5. 中田尚, 中島康彦: ”線形アレイ VLIW プロセッサにおける適応性検討”, 情報処理学会研究報告, Vol.2009-ARC-186, No.10, HOKKE-17, pp.1-9, Nov. (2009)
6. 上利宗久, 中田尚, 中島康彦: ”線形アレイ型 VLIW プロセッサの面積効率評価”, 平成 21 年度情報処理学会関西支部大会講演論文集, A-03, Sep. (2009) 【情報処理学会関西支部大会学生奨励賞】
7. 中島康彦: ”グリーンコンピューターへの道～計算の低消費電力化～”, 関西学研都市 6 大学市民講座, Oct. (2009) 【招待講演】
8. 中田尚, 上利宗久, 中島康彦: ”画像処理向け線形アレイ VLIW プロセッサ”, 先進的計算基盤システムシンポジウム SACSIS2009 論文集, pp.293-300, May. (2009)
9. Munehisa Agari, Takashi Nakada, Yasuhiko Nakashima: ”A Linear Array VLIW Processor for Image Processing”, IEEE Symposium on Low-Power and High-Speed Chips 2009 (poster), p.153,

Apr. (2009)
10. 中島康彦: ”脱マルチコアの試み －ヘテロ SMT 型 VLIW とリニアアレイ型 VLIW －”, 情報処理学会ものづくり基盤コンピューティングシステム研究会招待講演, Mar. (2009)
11. 中島康彦: ”３－ｗａｙから９Ｎ－ｗａｙに至る最近のＶＬＩＷ研究紹介”, 電子情報通信学会コンピュータシステム研究会招待講演, CPSY, Vol.2008 No.43-52, pp.31-36, Dec. (2008)
12. 上利宗久, 中田尚, 中島康彦: ”N倍速を目指すＶＬＩＷプロセッサの構想”, IPSJ SIG Technical Report, 2008-ARC-180, pp21-24, Oct. (2008)
13. 市來亮人: ”プロセッサ評価のためのハイブリッドプラットホーム”, STARC フォーラム/シンポジウム 学生ポスター, Jul. (2008)
14. Akihito Ichiki, Takashi Nakada, Yasuhiko Nakashima: ”A Hybrid Platform for Practical Evaluation of Processors”, IEEE Symposium on Low-Power and High-Speed Chips 2008 (poster), Apr. (2008)

Appendix A

命令一覧

Appendix B

モデル依存情報

Appendix C

用語集

プロセス 1 つのプログラムの実行状態であり，計算資源（実行に関わるソフトウェアから観測可能な全てのコア内外の論理的アーキテクチャレジスタおよび論理的メモリアドレス空間）の内容の総体である．

論理メモリ空間 1 つのプロセスに対応して 1 つ割り当てられる論理的なメモリアドレス空間である．当該プログラムの実行に参画するコアは，OS 機能を実行する場合も含めて当該論理メモリ空間のみを参照することができる．

スレッド 1 つのプロセスの実行に参画するコア毎に論理的に認識される計算資源の内容の総体である．1 つのプロセスを単一コアにより実行する場合，プロセスとスレッドは同一となる．1 つのプロセスを複数コアにより並列実行する場合，1 つのプロセスに複数のスレッドが属する．命令語単位のステップ実行はスレッド毎に行うことができる．

コアグループ 1 つのプロセスの実行に参画する実コアの集合である．プログラムヘッダ情報に基づいて OS がコアグループを構成し，コアグループに属する 1 つの実コアによりプログラムの実行が開始される．以後，当該プロセスの実行は，コアグループに属する実コアのみにより行われる．なお，プロセス実行中に動的にコアグループを再構成することも考えられるが，このようなプロセスが複数同時に存在する場合，OS による物理コア割り当て作業が複雑化し，場合によってはプロセスの停止など予測不能な著しい性能低下を引き起こす．このため，本仕様ではコアグループの動的再構成を規定しない．

物理コア番号 ハードウェアの物理的構成に依存し，各コアの物理位置によって一意に決まるコア番号である．使用可能状態や使用不可能状態（ハードウェア障害や未実装等による）は物理コア番号により OS に通知される．

実コア番号 プログラムから認識される論理的なコア番号である．OS による物理コア割り当て作業により，物理コア番号が実コア番号に変換された後，各プロセスに対して提供される．OS が設定するコア番号変換表に基づき，コア番号の変換を行う機構がコア番号変換機構である．

DSM 機能 1 つの物理コアに 1 つ付随して物理アドレス空間の一部を専有する局所メモリ（物理 DSM）および DSM 参照のための各機構の総称である．

実 DSM 番号 物理コア番号と物理 DSM 番号が 1 対 1 に対応することから，実コア番号に連動する番号である．各プロセスにおいて実 DSM 領域を一意に特定する際に使用する．

DSM アドレス変換機構（DSM-SAT） プロセス毎に，各実 DSM 領域の各ページ（ページサイズは DSM-SAT 固有）に対応付ける物理 DSM 領域内のページ位置を指定する制御レジスタ群である．関連して，物理 DSM 領域をまたぐアドレス境界を物理 DSM 境界と呼ぶ．

実 DSM 予約空間 論理メモリ空間中，実 DSM 番号順に実 DSM 領域が写像される空間である．コアグループの構成によっては DSM の実体がないアドレス範囲も存在するため，予約空間という表現を使用する．実コア番号と物理コア番号の対応はコア番号変換表に従うため，異なる物理コアに属する DSM 領域内ページを 1 つの実 DSM 領域内に混在させることはできない．すなわち，物理 DSM 境界を越えてアドレス変換を行うことはできない．

CSM 機能 各クラスタに付随して物理アドレス空間の一部を専有する局所メモリ（物理 CSM）および CSM 参照のための各機構の総称である．

CSM アドレス変換機構（CSM-SAT） プロセス毎に，実 CSM 予約空間の各ページ（ページサイズは CSM-SAT 固有）に対応付ける物理 CSM 領域内のページ位置を指定する制御レジスタ群である．関連して，クラスタをまたぐアドレス境界を物理 CSM 境界と呼ぶ．

実 CSM 予約空間 論理メモリ空間中，物理 CSM がページを単位として連続的に写像される空間である．CSM の構成によっては CSM の実体がないアドレス範囲も存在するため，予約空間という表現を使用する．実 DSM 予約空間と異なり，異なるクラスタに属する物理 CSM の各ページを混在させることができる．すなわち，物理 CSM 境界を越えてアドレス変換を行うことができる．

物理 DDR アドレス システムに接続される物理的な DDR メモリ内の位置を特定するために使用される．物理 DDR アドレスは DDR が接続されるクラスタ毎に連続であり，全クラスタで 1 つの物理 DDR アドレス空間を構成する．

Firmware 領域 物理 DDR アドレス空間上に存在し，リセットや IPL の際に特権モードで動作するプログラムおよびデータが配置される領域である．ユーザモードにあるプロセスは本領域を直接参照することはできない（OS が設定する DDR アドレス変換機構によりアクセス例外が報告される）．

OS 領域 物理 DDR アドレス空間上に存在し，Firmware の IPL 動作により OS が格納される領域である．ユーザモードにあるプロセスは本領域を直接参照することはできない（OS が設定する DDR アドレス変換機構によりアクセス例外が報告される）．ユーザモードにあるプロセスは，システムコール命令（ticc）を発行することにより特権モードへ移行し，本領域の機能を利用することができる．

I/O 制御領域 物理 DDR アドレス空間に連続する領域であり，I/O 機器の制御レジスタや物理入出力バッファが配置される．実体は物理 DDR アドレス空間上には存在しないが，OS 制御下の DDR アドレス変換機構により DDR 空間への写像が可能である．

DDR 空間 物理 DDR アドレス空間に写像される Firmware 領域，OS 領域，ユーザ DDR 領域の総称である．ユーザモードのプロセスから観測可能な論理メモリ空間はユーザ DDR 領域，一部の I/O 制御領域，実 DSM 予約空間，実 CSM 予約空間に大別されることから，ユーザモードのプロセスの視点から説明を行う場合には，ユーザ DDR 領域を DDR 空間と呼ぶ場合がある．

DDR アドレス変換機構（DDR-SAT） DSM-SAT および CSM-SAT を除く，DDR 空間を物理 DDR アドレスに変換するためのアドレス変換表およびアドレス変換過程から構成される論理的な機構である．アドレス変換に使用する変換表は OS による更新される．モデルによってはアドレス変換過程の一部もソフトウェア（OS）により実現される．

特権命令 特権モードでのみ使用が許可される命令の総称である．一般に Firmware や OS が使用する．

L2-TAG L2 キャッシュの各バンクに設けられる TAG の集合体である．L1 キャッシュミスを検出した場合，DDR アドレス変換後の物理 DDR アドレスを用いて同一クラスタの L2 バンクから 1 つが選択され，さらに物理 DDR アドレスにより決まる L2-TAG の内容により L2 キャッシュの HIT/MISS が判定される．L2-TAG は物理 DDR アドレスの下位ビットによりバンク分割され，バンク内位置も物理 DDR アドレスの一部によりインデックスされているため，参照すべき L2-TAG 位置は，物理 DDR アドレスから一意に決定できる．また，L2-TAG は有効な L1 キャッシュラインを保持するクラスタ内物理コア番号も保持している．

L2-DIRECTORY 自クラスタにおいて L2 キャッシュミスが発生した場合，かつ，物理 DDR アドレスが自クラスタ内 DDR を指す場合に，自クラスタ内 DDR に対応する有効なキャッシュラインが存在し得る他クラスタ番号を保持する機構である．

Shared 状態 異なるキャッシュに属するキャッシュラインが下位の記憶階層も含めて同一内容を保持している状態．Shared 状態にある各キャッシュラインはいつでも内容を無効化できる．

Modified+Invalid 状態 1 つのキャッシュラインが最新の内容を保持し，下位の記憶階層には内容が反映されておらず，他のキャッシュには同一アドレスに対応する有効なキャッシュラインが存在しない状態．

Owned+Shared 状態 Modified 状態にあるキャッシュラインに対応する他のキャッシュラインに対して READ 要求が発行され，Modified 状態にあるキャッシュラインから最新の内容が転送され，キャッシュラインが有効になった状態．

コア間転送機構（DTU） コアの命令実行とは非同期に，DDR，DSM，CSM 間のデータ転送を行うことができる機構．

バリア同期機構（BAR） 同期状態は 0 同期状態と 1 同期状態からなり，各実コアが現在の同期状態を認識し，他コアに対して 0 同期または 1 同期のいずれかを指定して次の同期点到達を放送することによりコアグループ全体が同期する．

並列実行制御情報 各コアのスタックポインタ初期値，下限値，各コアの命令開始アドレスである．複数コアによる並列実行を開始する際の各コアの初期化に使用する．

監視条件 論理メモリアドレス，マスク値および値の組である．コア間同期に使用する．

Appendix D

関連資料一覧

本章では，関連仕様書・規格，参考文献，関連ソースプログラム，および，SAPP ツールチェインを列挙する．

D.1 関連仕様書・規格

- GP5V330MF ボード設計仕様書第 0.4 版, 新和電材株式会社 (2010/3/11)
  … proj-gp5v330/doc/GP5V330MF 設計仕様書_V0_4.pdf
- GP5V330M コアジェネレータ解説書第 0.2 版, 新和電材株式会社 (2010/3/2)
  … proj-gp5v330/doc/GP5V330M_コアジェネレータ解説書 V0_2.pdf
- GP5V330M FPGA ソース解説書第 0.3 版，新和電材株式会社 (2010/3/2)
  … proj-gp5v330/doc/GP5V330M_FPGA ソース解説書 V0_3.pdf
- GP5V330M タイミングチャート第 0.2 版, 新和電材株式会社 (2010/3/30)
  … proj-gp5v330/doc/GP5V330M_タイミングチャート V0_2.pdf
- GP5V330M 合成配置配線手順書第 0.3 版, 新和電材株式会社 (2010/3/2)
  … proj-gp5v330/doc/GP5V330M_合成配置配線手順書 V0_3.pdf
- GP5V330 コンフィグレーション説明書暫定版, 三精システム株式会社 (2010/1/9)
  … proj-gp5v330/doc/コンフィグレーション説明書_暫定.pdf
- SYPCIE IP コア，SYSTEC Corporation (2009/6/29)
  … proj-gp5v330/doc/SYPCIE 機能仕様書_v100.pdf
- TB-5V-xx-PCIE-EX ハードユーザマニュアル Rev1.02, 東京エレクトロンデバイス
  … proj-gp5v330/doc/TB-5V-xx-PCIE-EX_HWUserManual_1.02_summary.pdf
- 36Mb Pipelined and Flow Through Synchronous NBT SRAMs Rev: 1.03b, GSI Technology (2005/4)
  … proj-gp5v330/doc/GS8320Z36GT.pdf
- FR550 Series Instruction Set Manual Ver.1.1, 富士通株式会社 (2002/2)
  … proj-lap/doc/frv/FRV550-1.pdf
- FR550 シリーズ MB93551 LSI 仕様書 第 1.1a 版, 富士通株式会社 (2002/3)
  … proj-lap/doc/frv/FRV550-2.pdf
- The SPARC Architecture Manual Version 9 SAV09R1459912
  … proj-sap/doc/sparc-v9/V9.pdf

D.2 参考文献

- 3.3V ZERO DELAY CLOCK BUFFER IDT2309, IDT (2002/11)
  … proj-gp5v330/doc/xilinx/IDT2309.pdf
- Digital Clock Manager (DCM) Module (2004/8/13)

⋯ proj-gp5v330/doc/xilinx/dcm_module.pdf
- LogiCORE FIFO Generator v2.3 User Guide (2006/1/11)
  ⋯ proj-gp5v330/doc/xilinx/fifo_generator_ug175.pdf
- FIFO Generator v2.3 (2006/1/11)
  ⋯ proj-gp5v330/doc/xilinx/fifo_generator_ds317.pdf
- Block Memory Generator v1.1 (2006/1/18)
  ⋯ proj-gp5v330/doc/xilinx/blk_mem_gen_ds512.pdf
- Dual-Port Block Memory Core v6.3 (2005/8/31)
  ⋯ proj-gp5v330/doc/xilinx/dp_block_mem.pdf
- Single-Port Block Memory Core v6.2 (2005/4/28)
  ⋯ proj-gp5v330/doc/xilinx/sp_block_mem.pdf
- Content-Addressable Memory v5.1 (2004/11/11)
  ⋯ proj-gp5v330/doc/xilinx/cam.pdf

D.3 関連ソースプログラム

- GP5V330-SAPP FreeBSD-7R 用ドライバ ⋯ proj-lap/src/nsim/GP5V330-Driver.c
- GP5V330-SAPP FreeBSD-7R 用ドライバ組み込み手順書 ⋯ proj-lap/src/nsim/GP5V330-Driver.hlp
- GP5V330-SAPP FreeBSD-7R 用ユーザ関数およびヘッダ ⋯ proj-lap/src/nsim/gp5v330.c
- GP5V330-SAPP Verilog 雛型 ⋯ proj-lap/fpga/step0013/*
- SAPP 性能・電力評価 RTL シミュレータ ⋯ proj-sap/src/ssim9/*

D.4 SAPP ツールチェイン（主要ツールのみ列挙）

- GP5V330-SAPP MCS ⋯ proj-lap/fpga/step1013/MF1,MF2/top/top.mcs
- GP5V330-SAPP コンフィグレーションツール ⋯ proj-lap/src/nsim/Config-gp5v330
- SAPP C プリプロセッサ ⋯ proj-sap/bin/sparc64-elf-cpp
- SAPP C コンパイラ ⋯ proj-sap/bin/sparc64-elf-gcc
- SAPP クロスアセンブラ ⋯ proj-sap/bin/sparc64-elf-as
- SAPP クロスリンカ ⋯ proj-sap/bin/sparc64-elf-ld
- SAPP 逆アセンブラ ⋯ proj-sap/bin/sparc64-elf-objdump
- SAPP 性能･電力評価 RTL シミュレータ（ssim9）用ライブラリ ⋯ proj-sap/lib/ssim64-lib/_start.o,_sub.o
- SAPP 性能・電力評価 RTL シミュレータ（ssim9） ⋯ proj-sap/src/ssim9/ssim964