名古屋大学大学院工学研究科博士課程（前期課程）
修士学位論文

ランダム並びの発行キューにおける命令再配置によるプロセッサの性能向上

平成 29 年 3 月

電気工学・電子工学・電子情報学専攻
酒井信二

概要

発行キューはプロセッサの構成要素の中でも性能と電力消費に大きな影響を与えるものの 1 つである．発行キューの代表的な構成方法として，シフトキューとサーキュラキューの 2 つがある．2 つのキューは共通して命令をプログラム順に保持する．しかし，これら 2 つのキューには問題がある．シフトキューには電力消費が大きい．サーキュラキューには容量の無駄があるという欠点が存在する．これらとは異なり，プログラム順に命令を格納しないランダムキューがある．ランダムキューは空きエントリに命令を挿入していくので，キューの容量は無駄にならない．しかし，命令がプログラム順に並ばないため，古い命令に高い発行優先度を与えることができず，プロセッサの IPC が低下するという欠点がある．

現在のプロセッサでは，この IPC 低下を抑制するために，エイジ論理と呼ばれる回路が用いられている．これにより最も古いレディな命令を同定することができるので，IPC 低下が抑制される．しかし，エイジ論理には発行論理の遅延を増加させるという欠点がある．

本研究では，遅延を増加させずにランダムキューの IPC を向上させる再配置ランダムキュー（RRQ:rearranging random queue）を提案する．ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）と呼ぶ 2 つのキューに分割する．フロントエンドからは，MQ にのみ命令を挿入し，OQ へは MQ の最も古い数命令を移動させる．2 つのキューで選択論理を共有し，OQ の命令を MQ の命令よりも優先して発行する．命令移動は命令の選択及び発行を並列に行うので，遅延を増加させずに IPC 低下を抑制することができる．

HSPICE を用いて回路シミュレーションを行い，SPEC CPU 2006 を用いて IPC を評価した．その結果，ランダムキューにエイジ論理を用いた場合，ランダムキューの性能は 7.2%低下したが，RRQ を用いた場合，6.2%増加した．

目 次

第 1 章　はじめに

現在のプロセッサには高性能と低電力が同時に求められている．従来は携帯機器では低電力が，PC やスーパーコンピュータでは高性能が特に求められていた．しかし近年では，高性能の携帯機器の需要が高まっており，また PC やスーパーコンピュータの性能が電力によって制限される状況になっている．そのため，使用される分野を問わず，高性能と低電力を高い次元で両立する必要性が出てきている．

プロセッサの性能に大きな影響を与えるものの 1 つとして発行キュー（IQ:issue queue) が挙げられる．アウトオブオーダ実行では IQ 内の命令の順序にかかわらず，レディになった命令を発行する．一般に，レディな命令のうち，より古い命令を優先的に発行すれば性能はより高くなる．IQ 内で命令をプログラム順に並べておけば，レディな命令から実際に発行する命令を選ぶ選択論理としては，単純な回路で，古い命令に高い優先度を与えるよう実現できる [1] [2] [3].

そのような IQ としては主に，シフトキューとサーキュラキューがある．シフトキュー [4] は，命令を発行したエントリの空きを詰めるコンパクションを行うことで，高い容量効率を達成できるが，コンパクションには大きな電力を要し，回路が複雑になる．一方，サーキュラキューはコンパクションをしないので電力消費が小さくなる．しかし，空いたエントリの分だけ実効的な容量が低下する欠点がある．

コンパクションをせず，実効容量を低下させない第 3 の方式としてランダムキューがある．ランダムキューは単に空いているエントリに新しい命令を挿入する．具体的には，空いているエントリの番号を保持するフリーリストを用意し，命令を挿入するときはそこからエントリ番号を得て，そのエントリに挿入する．ランダムキューには容量を無駄にすることがないという特徴がある．このためランダムキューは，例えば，Alpha 21464 [5], AMD

Bulldozer [6]，IBM POWER8 [7] といったプロセッサに用いられている．しかし，命令がプログラム順に並んでいないため，誤った優先度を選択論理に与えることになる．これにより，レディな命令数が発行幅や機能ユニットの数よりも多く，実際に発行する命令を選択しなければならない状況で，より重要でない命令を選択することがある．このようにして，誤った優先度はプロセッサの性能を低下させる．

この性能低下を抑制するために，上に挙げたプロセッサではエイジ論理が併用されている．エイジ論理は，レディな命令の中で最も古い命令 1 つを同定することができるので，この命令に最も高い優先度を与えることでプロセッサの性能低下を抑制する．しかし，発行論理が複雑になるため，その遅延が増加するという欠点がある．

エイジ論理に対して，本研究では，遅延を増加させずにランダムキューの IPC を向上させる**再配置ランダムキュー (RRQ:rearranging random queue)** を提案する．ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）と呼ぶ 2 つのキューに分割する．MQ と OQ はランダムキューであり，どちらからでも命令を発行できる．一方で，命令ディスパッチは MQ にのみ行い，OQ へは MQ 内の最も古い数命令を移動させる．MQ と OQ で選択論理を共有し，OQ の命令を MQ の命令よりも優先して発行する．命令移動は発行論理とは独立して行われるため，遅延を増加させずに IPC 低下を抑制することができる．

以下本論文の構成について述べる．2 章では関連研究について述べる．3 章では発行論理，及びエイジ論理について説明する．4 章で提案手法について述べ，5 章で評価する回路の構成について述べる．既存手法や提案手法について 6 章と 7 章でそれぞれ IPC と性能の評価を行う．最後に 8 章で本論文をまとめる．

第 2章　関連研究

発行キューに関連する研究について述べる．

2.1　シフトキュー

シフトキューを実装したプロセッサとして Alpha 21264 がある．文献 [4] にその回路の詳細が述べられている．1 章で述べたように，シフトキューは，十分に容量を有効利用できる他，コンパクションにより命令がプログラム順に並んでいるので，選択論理に正しい発行優先度を与えることができるという利点がある．しかし，コンパクションを行うために，複雑な回路が必要になり，電力を大きく消費する．

2.2　サーキュラキュー

Henry らは，サーキュラキューのラップアラウンドに対応する回路を提案した [8]．キューの最後のエントリと最初のエントリをつなぎ，論理的にループさせる．これにより，キューの先頭と末尾の位置がどの位置にあっても，命令の発行優先度が正しくなる．しかし，クリティカルパス上にキューの全体を横断するような配線が加わるため，選択回路の遅延が非常に大きくなる．

2.3　発行論理の最適化

Palacharla らは IQ を複数の FIFO で構成する手法を提案した [1]．命令をディスパッチするときは依存する命令を末尾に保持する FIFO に挿入する．そのような FIFO がなければ空の FIFO に挿入する．空の FIFO もなければ命令ディスパッチを停止する．発行する命令の選択は，FIFO の先頭にある命令のみを対象に行えばよいので，実装が単純になり，

複雑さが減る．しかし，FIFO の先頭の命令のソースオペランドの使用可否をチェックするために，多ポートのスコアボードが必要になるという欠点がある．

Stark らは IQ をパイプライン化する手法を提案した [9]．通常 1 サイクルで行われるウェイクアップと選択を 2 ステージにわけて行う．これにより複雑さが減り，大きな IQ の実装が可能になる．しかし，依存する命令を連続して発行するには，データフローグラフにおいて 2 つ前の命令が選択された時点で投機的にウェイクアップする必要があり，投機失敗からの回復を行う複雑な回路が必要となる．

五島らはウェイクアップ論理を CAM ではなく，依存行列と呼ぶ RAM で構成する手法を提案した [10]．依存行列は IQ 内の命令の依存関係を表す．これを用いることで比較器を使わずに依存する命令をウェイクアップできる．しかし，CAM の IQ と同じく，高い容量効率を達成しつつ，正しい発行優先度を命令に与える簡単な方法は提案されていない．

Brown らは選択論理を省略した IQ を提案した [11]．ウェイクアップと選択からなるクリティカルループから選択を排除することでループを短くでき，IQ の遅延を短くできる．しかし，この手法では，発行要求した命令は常に発行が許可されるとし，投機発行を行うので，投機失敗からの回復を行う複雑な回路が必要になる．

Michaud らは命令を IQ に挿入する前に，実行レイテンシを予測し，あらかじめスケジューリングすることで，発行キューの複雑さを軽減する手法を提案した [12]．しかし，キャッシュミスが起きると，IQ がミスした命令に依存する命令で満たされ，他の命令が発行できなくなる．また，動的にレイテンシが変動する命令については，レイテンシを正確に予測できず，その結果正しいスケジューリングを行えず，命令の発行が遅れるという欠点がある．

Sassone らは，行列を用いて選択論理を構成する手法を提案した [13]．各命令がどの命令より古いのかを表す行列を用いることで，最も古い命令を単純な回路で選択できる．行列を用いる選択論理は Alpha 21464 [5] と IBM POWER8 [7] で使用されているが，最も古い 1 命令しか同定できないので，完全に正しい優先度を与えられない．

第 3章　発行論理とエイジ論理

本章では提案手法の前提となる発行論理，及び比較対象となるエイジ論理について述べる．

3.1　発行論理

発行論理は，リネームされた命令を保持し，発行すべき命令を決定する．図 3.1 に示すように，発行論理はウェイクアップ論理，選択論理，タグ RAM，ペイロード RAM で構成される．

以下発行論理の動作について述べる．ウェイクアップ論理の各エントリは，対応する命令の 2 つのソースオペランドに対応するタグと，それらの状態を表すレディフラグを保持する．各ソースオペランドが利用できる状態ならば，対応するレディフラグがセットされる．2 つのフラグがセットされ，依存が解決したならば，発行要求信号を選択論理へ送る．選択論理は，資源制約などを考慮して，要求された命令の中から発行を許可する命令を選択し，発行許可信号を出力する．発行許可信号はペイロード RAM に送られ，機能ユニットに発行する命令の情報を送信する．発行許可信号はタグ RAM にも出力される．タグ RAM は各命令のデスティネーションタグを保持する．発行許可された命令のタグを読み出し，それをウェイクアップ論理へ放送して，レディフラグを更新する．

IQ がランダムキューの場合は，これらとは別にフリーリストを用意する．フリーリストは IQ の空きエントリの番号を保存するバッファであり，FIFO で構成される．命令をディスパッチするときは，フリーリストから空きエントリの番号を得て，そのエントリに命令を挿入する．一方，命令を発行したら，自身のエントリ番号をフリーリストに返す．

ウェイクアップ→選択→タグ RAM 読み出しからなるループは，1 サイクルで完結しなければならない．そうでなければ，依存する命令を連続したサイクルで発行できなくなり，

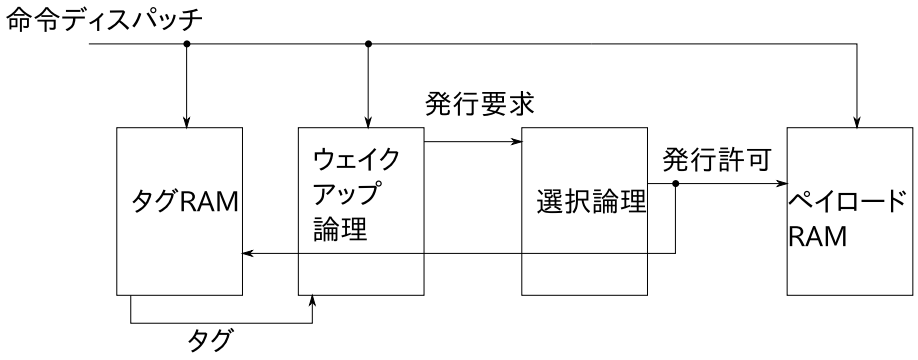

図 3.1: 発行論理の概略図

IPC が低下する．この 1 サイクルのループはプロセッサにおけるクリティカルパスの 1 つとして知られている [1].

3.2 エイジ論理

データフローのクリティカルパス上に命令があるならば，後続の多くの命令が直接，あるいは間接的にこれに依存している．このため，このような命令は IQ 内に長時間残っている古い命令となる確率が高い．すなわち，古い命令はデータフローのクリティカルパス上にあると言えるので，性能への影響が大きく，優先的に発行する必要がある．逆に新しい命令は，依存する命令の数が少ないので，発行の順番がプログラム順でなくとも，性能はあまり低下しないと言える．この推測に基づいてエイジ論理では，データフローのクリティカルパス上にある可能性が最も高い命令，つまり最も古い命令を同定する．これにより，最も古い命令が発行されることによってランダムキューの IPC を改善することができる．

エイジ論理を用いる場合，図 3.2 に示すようにウェイクアップ論理と選択論理の間にエイジ論理が挿入され，選択論理とは並列に動作する．エイジ論理は，発行要求信号を受け取り，その中で最も古い命令を同定する．そして，エイジ論理のウィナー出力と選択論理の先行発行許可が重複した場合に，同一命令の二重発行を防ぐための調停を行い，これの出力を新たな発行許可信号とする．

本節で示すエイジ論理は，Alpha 21464 [5] で用いられたものである．その回路構成を図

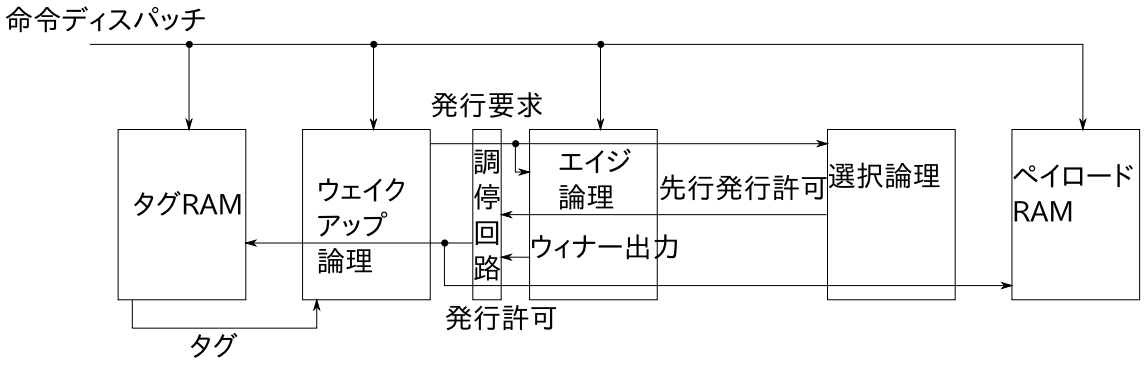

図 3.2: エイジ論理を用いる発行論理

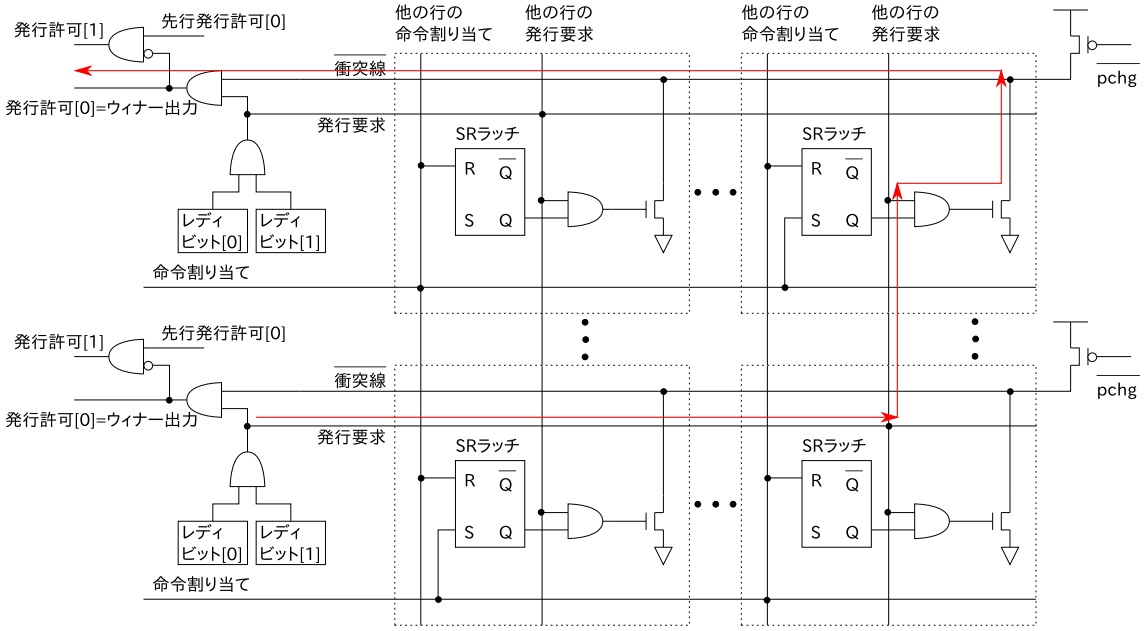

図 3.3: エイジ論理の回路構成

3.3 に示す．エイジ論理は正方形の行列の形で構成され，行および列の数は IQ のエントリ数に等しく，行，列共に IQ のエントリに対応している．図中で点線で囲まれた部分は，行列を構成するセルを表す．各セルは SRFF を含み，これによって命令の古さを表す情報を保持する．セル $[i,j]$ が 1 を保持する場合，行 i の命令は行 j の命令よりも新しいことを表す．この情報を参照することで，ある命令が他の命令よりも新しいか否かを知ることができる．

エイジ論理のセルが示す情報の更新方法について述べる．情報の更新は IQ に命令がディ

スパッチされるときに行われる．ある行に命令がディスパッチされる時，図中で命令割り当てと表された配線がアサートされる．これにより，命令自身が対応する列を除いて，その行の全てのセルの情報をセットする．この動作により，この命令は，他のどの命令よりも新しいことを示すようになる．同時に，ディスパッチされた命令が対応する列の他の行のセルの情報をリセットする．この動作により，各命令はディスパッチされた行にある命令よりも古いことが分かるようになる．

セルの情報を用いて，最も古い命令を同定する方法を説明する．自身よりも古い命令がレディならば，発行要求を打ち消す．この動作を全ての命令について行えば，レディな命令の中で最も古い命令だけが残る．具体的な動作について述べる．図中で $\overline{\text{衝突線}}$ と示された配線は，命令選択が行われる前にプリチャージされる．2つあるレディビットが共にセットされた命令は，エイジ論理に発行要求信号を送出する．この発行要求信号は，自身の対応する列を通して他の行の命令にも伝えられる．各セルにおいて，他の行の発行要求信号とSRFF が保持する値が共に 1 ならば，つまり自身よりも古い命令がレディならば，$\overline{\text{衝突線}}$ がディスチャージされる．これによって新しい命令の発行要求信号が打ち消され，レディな命令の内最も古い命令だけが残り，結果的にそれを同定することができる．

エイジ論理クリティカルパスについて述べる．図 3.3 において赤い矢印で示したパスが，最も長い時間を要するクリティカルパスである．発行要求信号がエイジ論理の一番下の行を横断し，一番右の列を縦断する．そして，プルダウントランジスタを通して $\overline{\text{衝突線}}$ をディスチャージし，一番上の $\overline{\text{衝突線}}$ を横断する．そして，ウィナー出力が得られたら，選択論理から得られた先行発行許可と調停を行う．選択論理よりもエイジ論理を優先するので，ウィナー出力が 1 の時は先行発行許可を打ち消す．クリティカルパスは，エントリ数の 3 倍のセルを通過するので非常に長い．そのため，エイジ論理による遅延は大きく，選択論理の遅延を上回る可能性がある．

第 4章　再配置ランダムキュー

本章では提案手法である再配置ランダムキューについて述べる．4.1 節では提案手法の概要について述べる．4.2 節では提案手法の発行論理の構成について述べる．4.3 節ではメインキュー内の命令のプログラム順を管理するプログラムオーダーキューについて述べる．4.4 節では命令をメインキューからオールドキューへ移動させるタイミングについて述べる．4.5 節では命令移動のために，MQ からデータを読み出すための回路について述べる．

4.1　概要

3.2 節で，古い命令はデータフローのクリティカルパス上にある可能性が高く，優先的に発行する必要があると述べた．エイジ論理では最も古い命令のみを優先的発行の対象としたが，必ずしもそれだけがクリティカルパス上にあるとは限らない．実際，文献 [17] により，IQ の中の最も古い 4〜8 命令がクリティカルパスに含まれ，性能に影響を与える確率が非常に高いことが示されている．そのため，ランダムキューを用いたプロセッサがより高い性能を発揮するには，最も古い命令だけでなく，その次に古い数命令も優先的に発行する必要がある．

そこで，ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）に分割する再配置ランダムキュー (RRQ:rearranging random queue) を提案する．MQ と OQ はどちらもランダムキューであり，どちらからでも命令を発行できる．一方で，命令のディスパッチは MQ にのみ行い，OQ へは MQ の最も古い数命令を移動させる．選択論理は MQ と OQ とで共有し，OQ の命令を MQ の命令よりも優先的に発行する．以上のような構成により，エイジ論理を用いるランダムキューと比較して，元々のプログラム順においてより多くの古い命令に対して高い優先度を与えるので，プロセッサの性能低下

を抑制できる．

4.2 再配置ランダムキューの構成

RRQ を実装した発行論理全体の概略図を図 4.1 に示す．命令の情報を保持するウェイクアップ論理，タグ RAM，ペイロード RAM は MQ と OQ に分割される．図中の PQ はプログラムオーダーキュー（program order queue）である．これは MQ 内の命令のプログラム順を記憶しており，OQ に移動する MQ 内の命令を決定する．MQ，OQ 共にランダムキューなので，命令を挿入するエントリを決定するためのフリーリストをそれぞれ個別に備えている．

MQ に命令をディスパッチするときは，フリーリストから空きエントリの番号を取得し，そのエントリに命令を挿入する．同時に PQ の末尾にそのエントリ番号を挿入する．

MQ の古い命令を OQ へ移動させるときは，移動させる命令のエントリ番号を PQ の先頭から読み出す．OQ は自身のフリーリストから空きエントリの番号を得て，該当するエントリに MQ から読みだした命令を書き込む．

MQ と OQ は通常の IQ と同じように命令の発行要求を選択論理に出し，発行許可を受ける．選択論理は両方のキューで共有している．選択論理は，図 4.1 における下の命令ほど高い優先度で選択するよう構成し，OQ 内の命令からの要求を MQ 内の命令のそれよりも優先する．この結果，エイジ論理を用いるランダムキューと比べて，クリティカルパス上にある確率が高い複数の命令に高い優先度を与えることができる．

4.3 プログラムオーダーキュー

PQ は MQ 内の命令のプログラム順を保存するキューである．PQ はサーキュラバッファであり，ヘッドポインタとテールポインタでデータの先頭と末尾を管理する．PQ の各エントリは MQ のエントリを指すポインタを持ち，これをプログラム順に保持する．

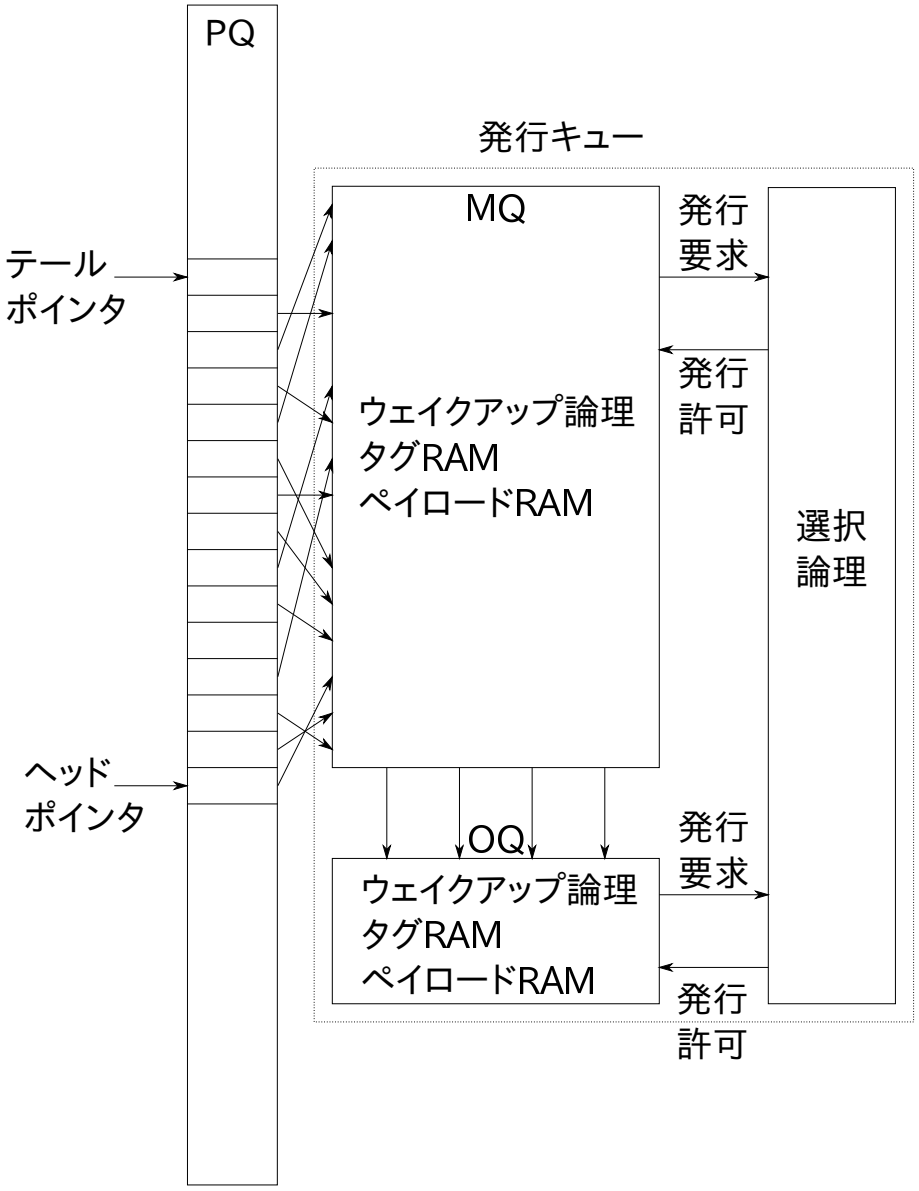

図 4.1: RRQ を実装した発行論理の概略図

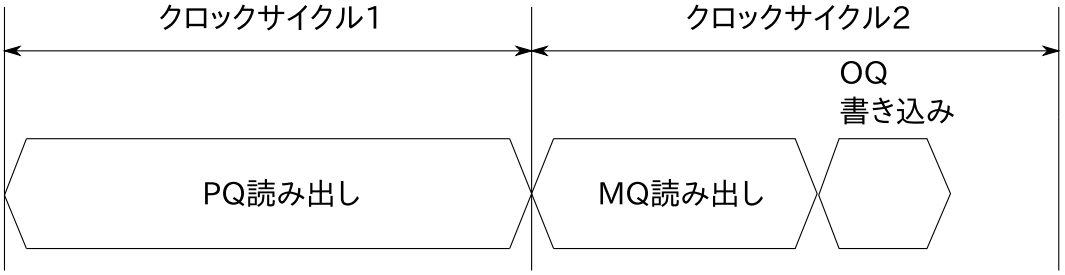

図 4.2: MQ から OQ へ命令移動

PQ の動作について述べる．MQ へ命令をディスパッチすると同時に，ディスパッチされた MQ のエントリ番号を PQ の末尾に挿入する．命令移動時の PQ の動作のタイミングを図 4.2 に示す．まず最初のサイクルで，PQ の先頭から MQ へのポインタを読み出す．次のサイクルでは，MQ の命令を読み出し，OQ へ書き込み，MQ の命令を無効化する．以上のように，PQ 読み出しから，OQ への移動までに合計 2 サイクルを要する．

PQ のサイズについて述べる．MQ は空きエントリのいずれかに命令を挿入できるが，PQ は，命令が挿入された MQ のエントリ番号を末尾にのみ挿入できる．このように，PQ は容量効率が悪いため，MQ より十分大きい必要がある．PQ はプロセッサ中の一部の命令のプログラム順を保持するので，PQ のサイズはインフライト命令数，つまりリオーダバッファ（ROB:reorder buffer）のサイズだけあれば十分である．PQ のサイズをそれ未満にする場合は，エントリ不足でストールし，性能は低下する可能性がある．

4.4 二重発行抑止

MQ から OQ への命令移動においては問題が発生する可能性がある．MQ のあるエントリにおいて，命令移動と発行許可が同一サイクルで起きると，MQ から命令が発行された後に，OQ から同一の命令が再び発行される二重発行という問題が生じる．連続したサイクルで同一命令が発行される場合のタイミングを図 4.3 に示す．図の 1 段目，2 段目はそれぞれ，MQ での命令発行，OQ に関連する動作の各タイミングを表す．ウェイクアップ，選択，タグ RAM 読み出しには，いずれも約 3 分の 1 サイクル時間を要する [3] ので，発行論

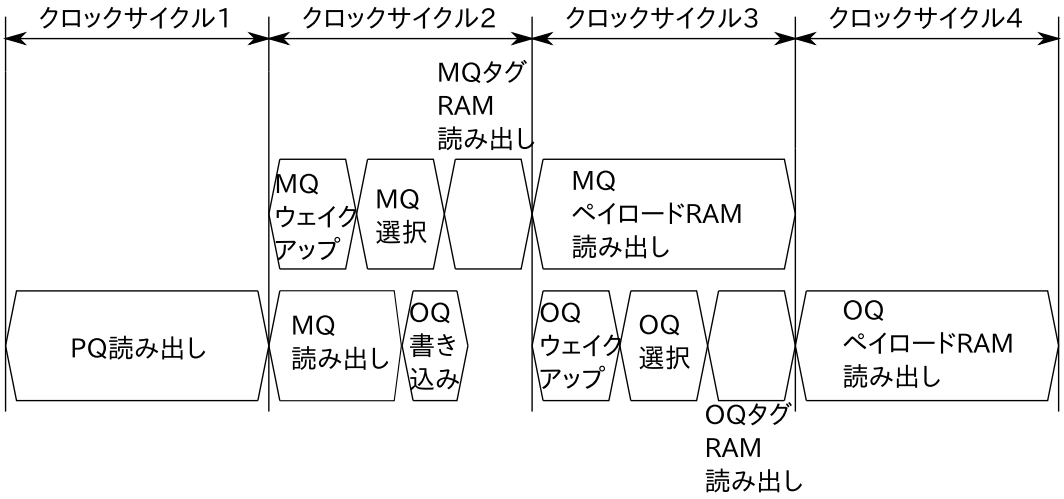

図 4.3: 二重発行が発生する場合のタイミング図

理のタイミングは図のように描ける．

二重発行によって，ROB やロードストアキュー（LSQ:load/store queue），レジスタなどに矛盾が生じ，多くの問題が引き起こされるため，二重発行が発生しない構成にする必要がある．そのような構成として，発行要求信号をゲーティングする方法を提案する．

エントリに有効ビットを設け，ウェイクアップ論理からの発行要求信号をそれによりゲーティングすることで二重発行を防ぐ．これを実現する回路を図 4.4 に示す．あるエントリに命令がディスパッチにより書き込まれたら有効ビットをセットする．一方，命令移動のための読み出しワード線がアサートされたら，対応するエントリの有効ビットをクリアする．これにより，有効ビットによって，選択論理へ送られる発行要求信号の出力は抑止され，二重発行を防ぐことができる．この時のタイミングを図 4.5 に示す．発行要求抑止により図の赤いハッチの部分の動作が行われない．その結果，ウェイクアップは行われるが，選択以降の動作は行われず，MQ からの発行を抑止できる．

この方法には，有効ビットの無効化は，レディビットがセットされるウェイクアップの終了時点までに行われなければならない，というタイミング制約がある．6.4 節の図 6.7 に示すように，命令移動のための読み出しワード線は，ウェイクアップ論理の動作の終了よりも十分早く立ち上がる．そのため，レディビットがセットされるよりも前に有効ビット

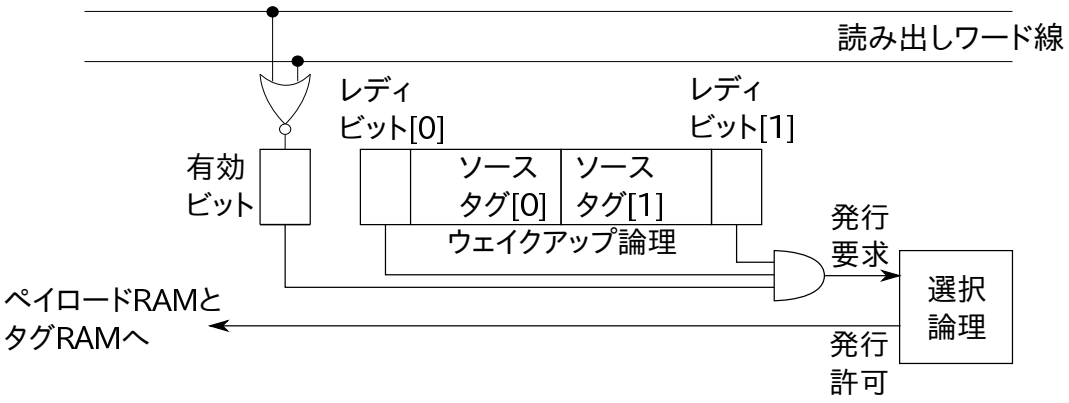

図 4.4: 発行要求信号をゲーティングする回路

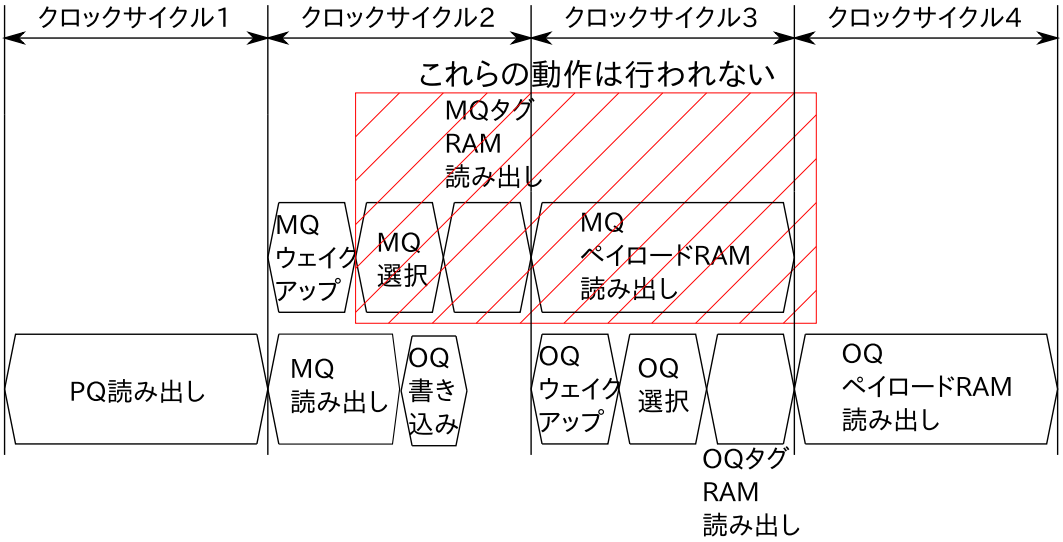

図 4.5: 二重発行を抑止するタイミング

をリセットすることは可能である．

4.5 MQ 読み出し回路

発行論理の遅延を増加させないために，命令移動は，命令のウェイクアップ，選択，タグ RAM 読み出しとは並列に実行しなければならない．また，命令移動のために発行キューのポート数を増やすと，回路面積が大きくなり，遅延が増加してしまう．そのため，命令移動は，命令ディスパッチに用いるポートを時分割で利用して行う．命令ディスパッチは発行論理とは並列に動作しているため，既存のポートを用いることにより，遅延を増加さ

せることなく，命令移動が可能となる．

しかし，遅延を増加させることなく，ポートを時分割で利用するためには，命令移動と命令ディスパッチの遅延の和が発行論理の遅延以下である必要がある．既存の SRAM の回路ではこの制約を満たすことができないため，MQ をバンクに分割しビット線を階層化することによって，命令移動と命令ディスパッチの遅延を減少させる．その回路を図 4.6 に示す．

図中で赤い点線で囲まれた部分が分割された MQ の 1 つのバンクを表す．隣接する 2 つのバンクはそのセンスアンプの出力を NAND ゲートにつなぐ．この NAND ゲートは PMOS のゲート幅が大きく，立ち上がり優先の NAND ゲートであり，入力の電圧降下に素早く反応する．いずれかのバンクから 0 が出力されると，NAND ゲートとプルダウントランジスタを通してグローバルビット線がディスチャージされる．このディスチャージを立ち上がり優先の NOT ゲートで検出し，その NOT をとることで MQ から 0 を読み出すことができる．メモリに 1 が保持されている場合，グローバルビット線はディスチャージされず，1 がそのまま読み出される．

MQ の分割によって生じる影響について述べる．全体としてグローバルビット線が追加される．これは既存の SRAM の上層を走らせることができるので，回路面積に与える影響はない．また，バンク毎に，センスアンプ，ライトドライバ，立ち上がり優先 NAND，そしてプルダウントランジスタが追加される．これらは数エントリに 1 つずつ追加されるので，回路全体の面積に対して与える影響は小さいと考えられる．以上のように，命令移動を可能にするための実装によって回路面積に与える影響は小さく，命令移動を発行論理と並列に動作させることができるため，発行論理の遅延を増加させることなく，RRQ を実装することが可能である．

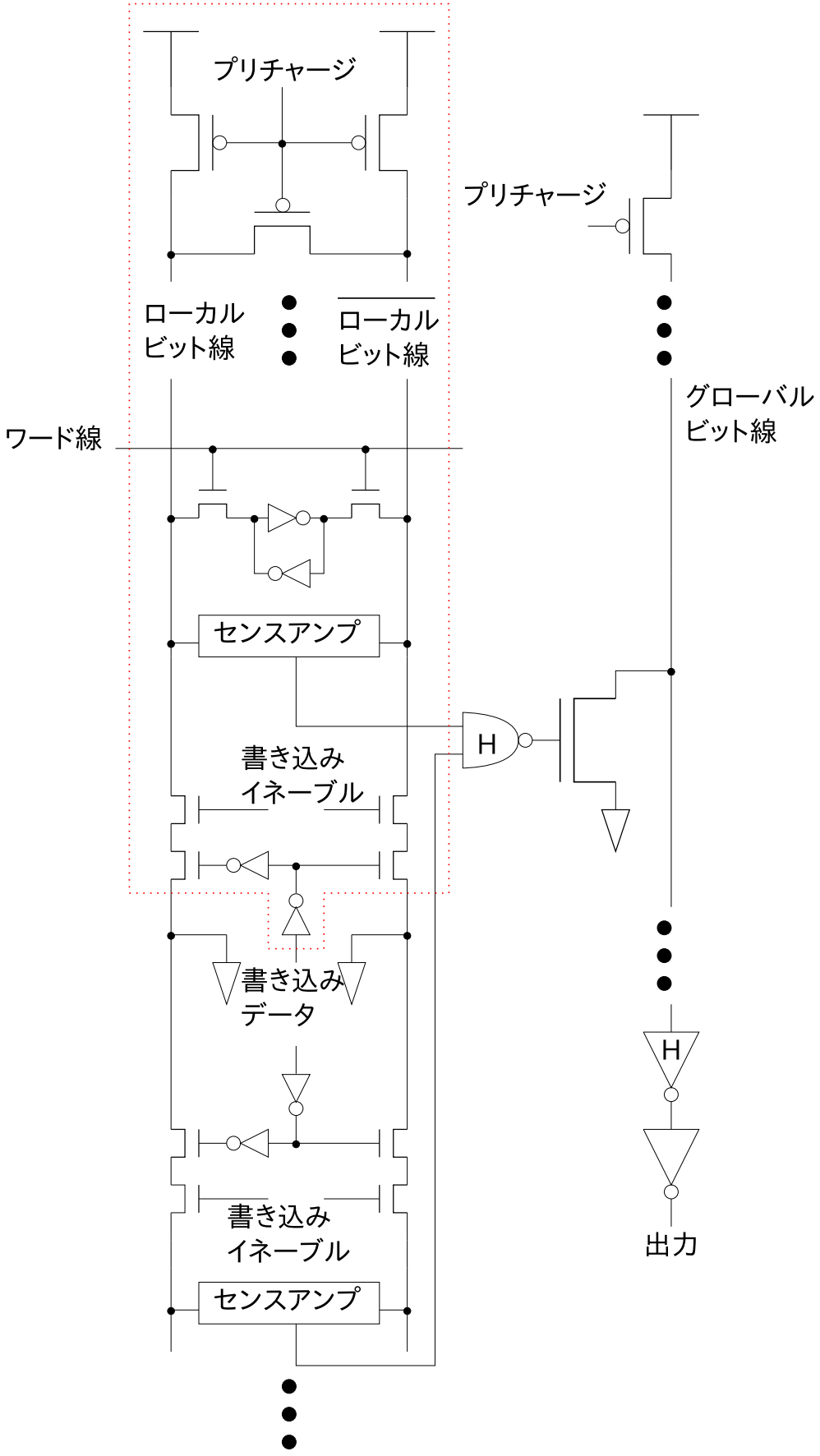
プリチャージ
プリチャージ
ローカル
ビット線
ローカル
ビット線
グローバル
ビット線
ワード線
センスアンプ
H
書き込み
イネーブル
書き込み
データ
H
書き込み
イネーブル
出力
センスアンプ

図 4.6: 分割された MQ

第 5 章　回路構成

この章では，発行論理を構成するウェイクアップ論理，選択論理，タグ RAM，ペイロード RAM，及びエイジ論理と RRQ を構成する回路についてそれぞれ順番に詳しく述べる．

5.1　ウェイクアップ論理

5.1.1　レイアウト図

ウェイクアップ論理では，タグ RAM から送られてきたデスティネーションタグと，ウェイクアップ論理が保持するソースタグとを比較する．そのため，ウェイクアップ論理は CAM で構成される．CAM を構成する 6T SRAM と比較器のレイアウトをそれぞれ図 5.1 と図 5.2 に示す．図は発行幅が 2 の場合を示す．

発行幅を iw とすると，SRAM の縦幅は $(28 + 14.5 * iw)$，比較器の縦幅は $37\lambda/2 * iw = (18.5 * iw)\lambda$ となる．発行幅が 7 未満の場合は SRAM のほうが大きくなり，8 以上の場合は比較器のほうが大きくなるので，ウェイクアップ論理のセルの縦幅はどちらか大きい方を採用する．

SRAM の横幅は $39\lambda + \{3\lambda + (4\lambda + 3\lambda) * (iw - 1)\} * 2 = (31 + 14 * iw)\lambda$，比較器の横幅は $30\lambda + 6\lambda * iw * 2 = (30 + 12 * iw)\lambda$ である．両者の間隔を 3λ 開けなければならないことを考慮すると，ウェイクアップ論理のセルの横幅は $(31\lambda + 14\lambda * iw) + (30\lambda + 12\lambda * iw) + 3\lambda + 3\lambda = (67 + 26 * iw)\lambda$ である．

5.1.2　回路構成

ウェイクアップ論理を構成する回路を図 5.3 に示す．図は発行幅が 2 の時の回路を示している．トランジスタの側にある数字はそのゲート幅を表し，論理ゲートの側に付いている

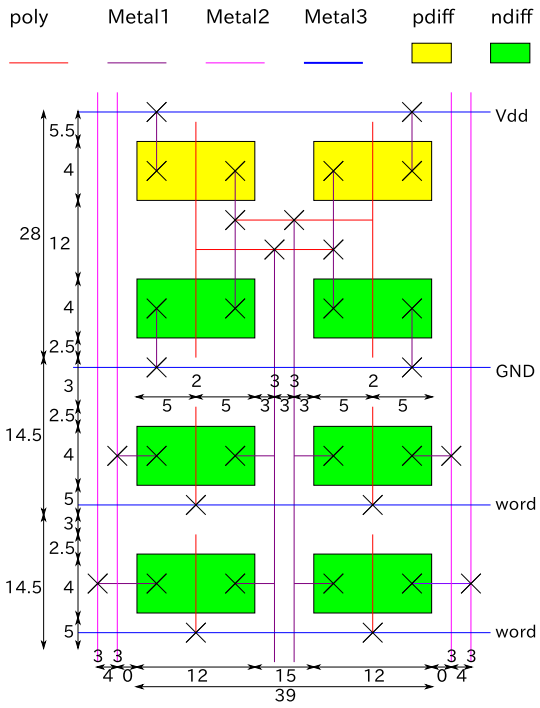

図 5.1: SRAM のレイアウト

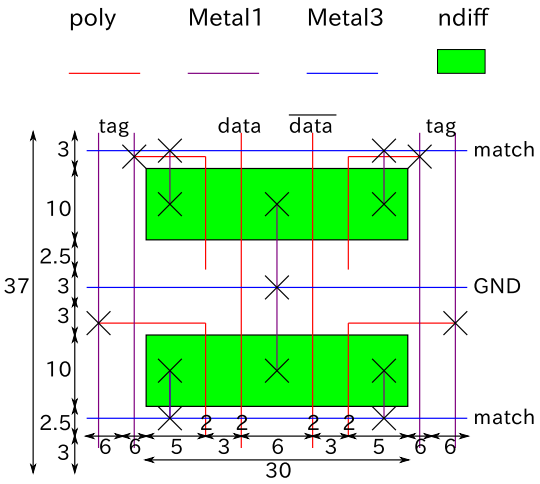

図 5.2: 比較器のレイアウト

数字は (NMOS のゲート幅/PMOS のゲート幅) を表す．図左上にあるタグ線のドライバのゲート幅を表す α や，図下部にあるマッチ線のリピータの NMOS トランジスタのゲート幅を表す β は，最適な遅延になるように調整すべき値である．

遅延測定の流れについて述べる．この回路の信号は左上のパルス波から始まる．このパルス波はタグ RAM から送られてきたタグを表し，シミュレーションではソースタグと不一致となるタグを送る．信号はドライバを通じてタグ RAM を横断し，ウェイクアップ論理の左右のソースにタグを伝える．左のソースを比較するパスの場合，比較結果が得られた後にウェイクアップ論理の右半分を発行要求信号が横断する必要がある．その結果，右のソースと比較するパスよりも左のソースを比較するパスの方が遅延が大きくなるので，後者の遅延をウェイクアップ論理の遅延とする．クリティカルパスは一番下のエントリを通るパスである．タグ線を通じてタグが一番下のエントリまで届き，マッチ線がディスチャージされる．マッチ線の中央には高速化のためのリピータが挿入されている．ダイナミック NAND を用いて全てのタグと一致していないことを調べる．タグが全て不一致なので，こ

のエントリの命令はレディにはならず，レディビットを保持するSRラッチのS入力は0となる．しかし，それではSRラッチの出力が変化せず，遅延を測定できないので，インバータを挿入してS入力を1にする．これにより，SRラッチからレディビットが得られるので，ウェイクアップ論理の最後まで遅延を測定することができる．

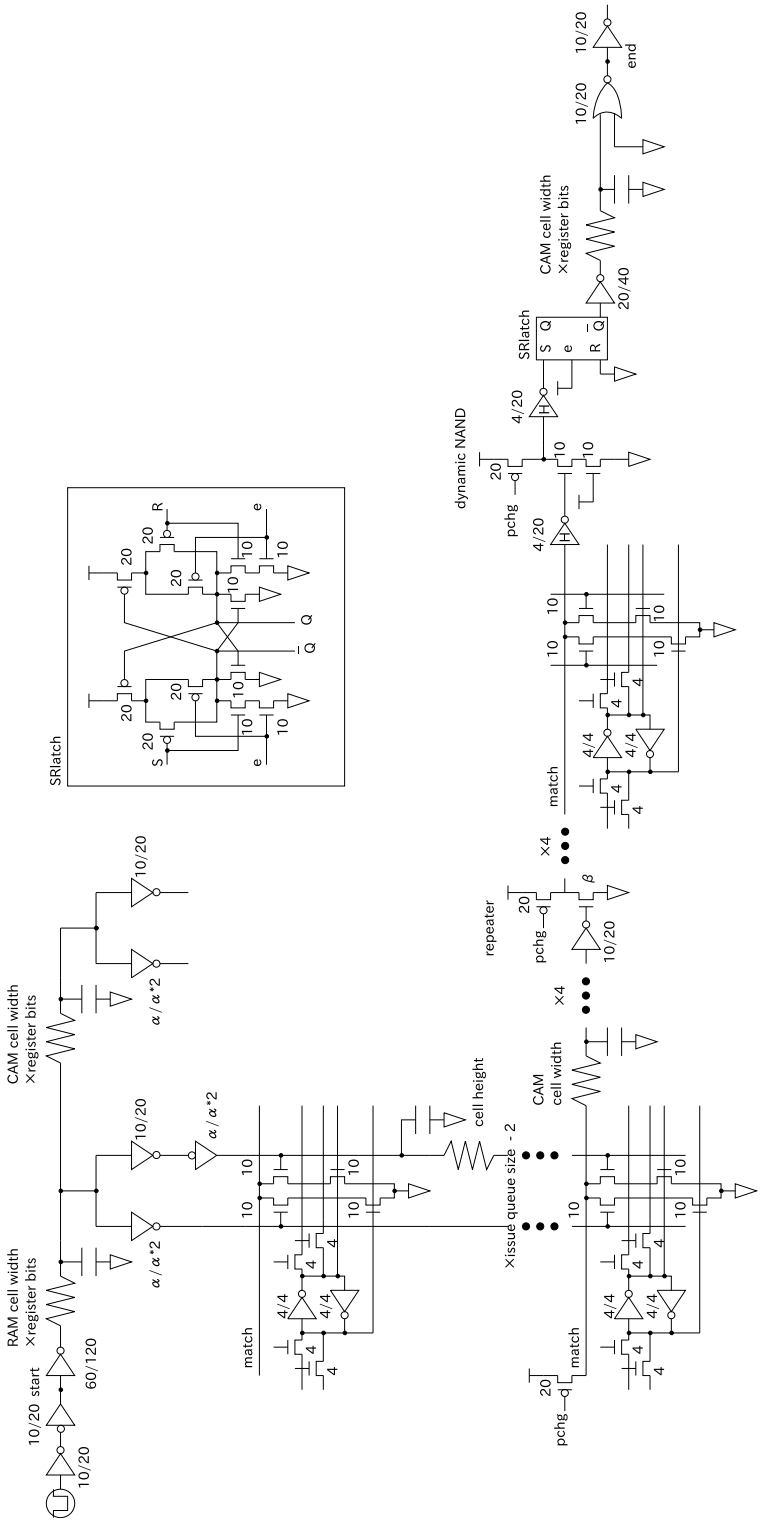

図 5.3: 測定に用いたウェイクアップ論理の回路図

5.2 選択論理

選択論理のセルのレイアウトを設計する上で，発行要求を数えるプレフィックスサム論理，及びそれを構成する加算器が重要となる．そのため，これらについて説明した後に選択論理のセルのレイアウト図や回路図について述べる．

5.2.1 プレフィックスサム

プレフィックスサム論理は入力信号と出力信号の個数が共に N であり，i 番目の出力の値は 0 番目から i 番目の入力の総和である．$N = 16$ の場合の回路図を図 5.4 に示す．図中の四角形は加算器を表す．

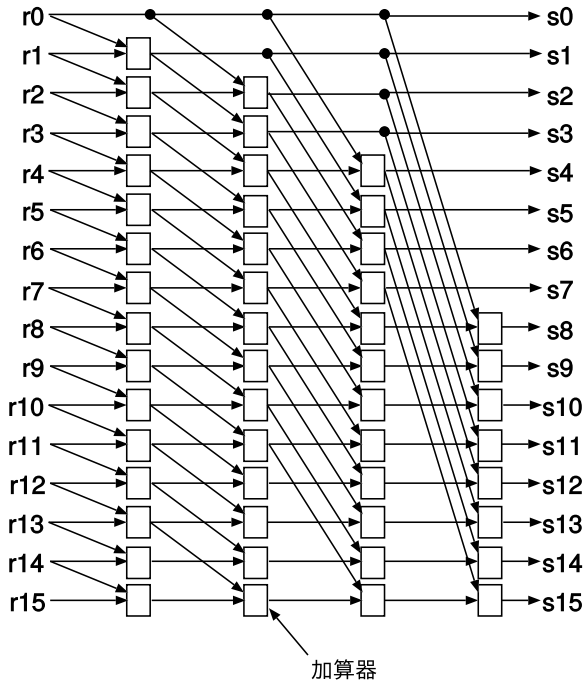

図 5.4: プレフィックスサム回路

この論理の入力を発行要求信号とし，$i-1$ 番目の出力が発行幅未満であり，かつ i 番目の入力が 1 ならば，i 番目の命令の発行を許可することで，プレフィックスサム論理を選択論理として利用することができる．このような選択論理において，遅延を短くした加算器の回路が文献 [2] で提案されている．プレフィックスサム論理を選択論理に採用するならば，

発行幅が 4 の場合，入力及び出力には 5 つの値，すなわち ”0”，”1”，”2”，”3”，”≥4” だけあれば十分である．この 5 つの値を示すように，加算器の入出力を表 5.1 のようにワンホット符号化することで，加算器を簡単にすることができる．その加算器の回路を図 5.5 に示す．この加算器を用いる場合，発行許可信号を出力する最後の加算器周りの回路は図 5.6 に示すようになる．図 5.5 と図 5.6 はどちらも発行幅が 4 の場合を示す．

value	0	1	2	3	≥4
decoded	1000	0100	0010	0001	0000

表 5.1: 発行幅が 4 の場合の加算器の入出力のワンホット符号化

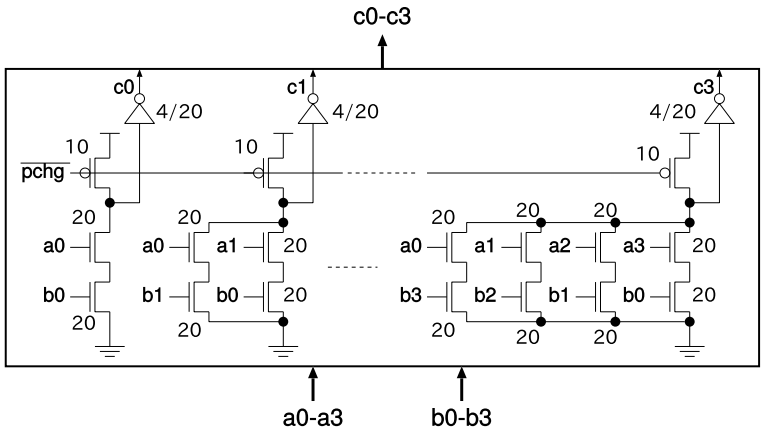

図 5.5: 選択論理で用いる加算器

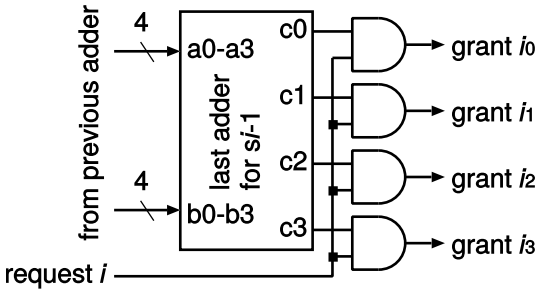

図 5.6: グラント信号を出力する加算器

5.2.2 セルを構成する加算器

選択論理の 1 セルは図 5.5 で示したような加算器で構成される．ここでは，図に示された加算器を構成する個々のダイナミック回路を左から順に，adder_1，adder_2，adder_4 と呼び，これらをまとめて adder とする．

異なる種類の adder を発行幅の数だけ並べることで選択論理のセルを構成できるが，全てのセルにおいて全種類の adder を並べる必要があるわけではない．これについて図 5.7 に示した adder_3 を用いて説明する．図 5.4 において一番左にある 1 段目のセルの場合，発行要求の総和は 0 また 1 なので，表 5.1 に従ってワンホット符号に変換すると，4 ビットある入力の内，3 ビット目と 4 ビット目は必ず 0 になる．これにより，図 5.7 において 3 ビット目に当たる入力 a2 と入力 b2 が必ず 0 になるので，3 つある NMOS の並びの内，2 つは機能しない．よって，この adder_3 を adder_1 に置き換えることができる．adder_4 の場合，入力 a2，a3，b2，b3 が必ず 0 になるので，adder_4 は全く機能せず，不要となる．よって選択論理の最初のセルは，adder_1，adder_2，adder_3，adder_4 の 4 つではなく，adder_1，adder_2，adder_1 の 3 つで構成することが可能である．

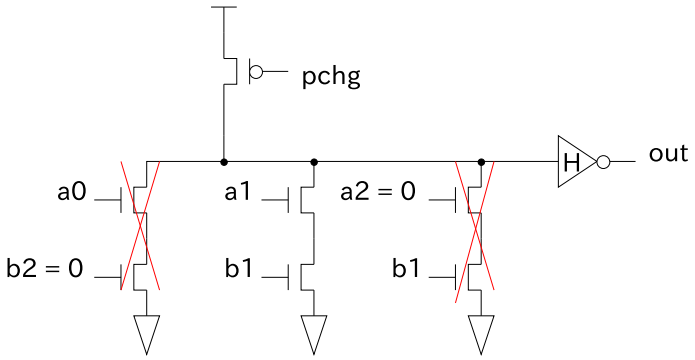

図 5.7: adder_3 の回路構成

2 段目以降の加算器についても同様のことが言える．2 段目ならばそれまでの発行要求の総和はたかだか 2 なので，3 ビットあれば表現できる．よって，adder_4 を adder_2 に置き換えることができ，2 段目のセルは adder_1，adder_2，adder_3，adder_2 で構成できる．3

段目も発行要求の総和はたかだか 4 なので発行幅が 6 以上であれば，一部の adder をより小さい adder に置き換えることができる．より小さい adder を用いることで選択論理の横幅を短くすることができる．

5.2.3 レイアウト図

上述したように選択論理のセルは adder によって構成される．よって adder のレイアウトを設計すれば，選択論理のセルのレイアウト設計が可能になる．adder_1 と adder_2，それぞれのレイアウトを図 5.8 と図 5.9 に示す．図は発行幅が 4 の場合を示している．

adder は縦方向には 3 つしかトランジスタが並んでいないので，セルの縦幅は配線に依存する．入力となる Metal2 は，幅が 3λ であり，via を用いる時の増加分を考慮すると，間隔は 5λ となる．pchg を含めて 9 本あるので，これらが占める大きさは $(3\lambda+5\lambda)*(9-1)+3\lambda = 67\lambda$ である．VDD と GND はエレクトロ・マイグレーションを防ぐために，配線の幅を 9λ としている．VDD と GND の配線は，入力の Metal2 や出力の Metal3 とも異なる Metal4 にし，他の配線と一部を重ねている．これにより，adder の縦幅は Metal4 により，1 本辺り 9λ ではなく 4λ 増える．また，VDD と GND の間に 4λ の間隔が必要である．以上より選択論理のセルの縦幅は，2 つの図の右側に示す通り，$67\lambda + 4\lambda * 2 + 4\lambda = 79\lambda$ となる．これは adder の種類には依存しない．

横幅は adder の種類によって異なる．adder_1 の場合は，横幅はプリチャージトランジスタとインバータの PMOS によって決定される．図 5.8 の上側に示すように，その横幅は $(6 + 10 + 5.5 + 5 + 20 + 2.5 + 3)\lambda = 51.5\lambda$ である．

adder_2 の場合は，横幅は NMOS によって決定される．図 5.9 の左下に示すように，NMOS1 つ分の横幅が $(6 + 20 + 5.5)\lambda = 31.5\lambda$ である．加算器に用いるインバータには立ち上がり優先 NOT を採用したので，NMOS が小さくなっている．これらをあわせると adder_2 の横幅は，図 5.9 の下側に示すように，$(6 + 20 + 5.5 + 6 + 20 + 4 + 4 + 2.5 + 3)\lambda = 71\lambda$ となる．

adder_3 以降の場合は，adder_2 の場合と同様に，横幅は NMOS によって決定される．

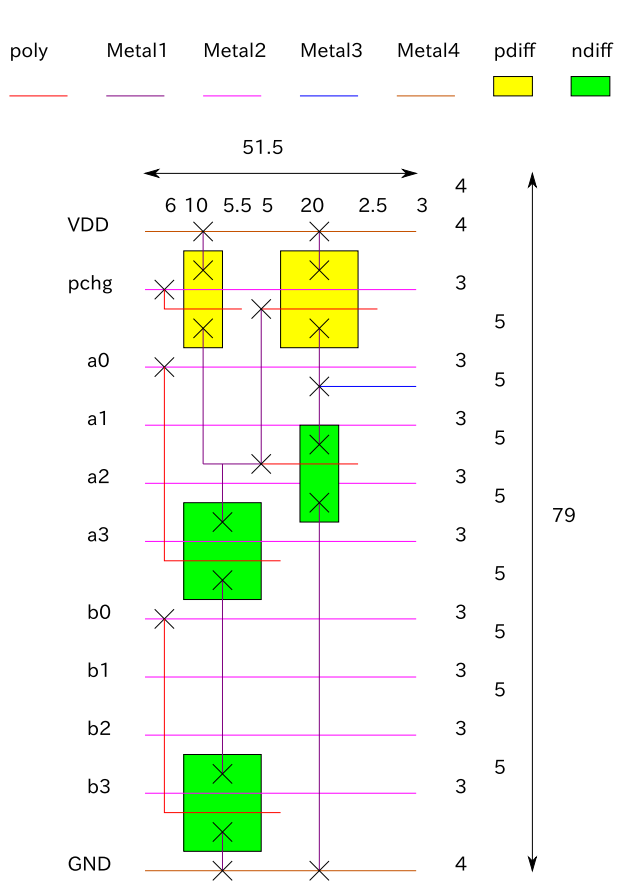

図 5.8: adder_1 のレイアウト

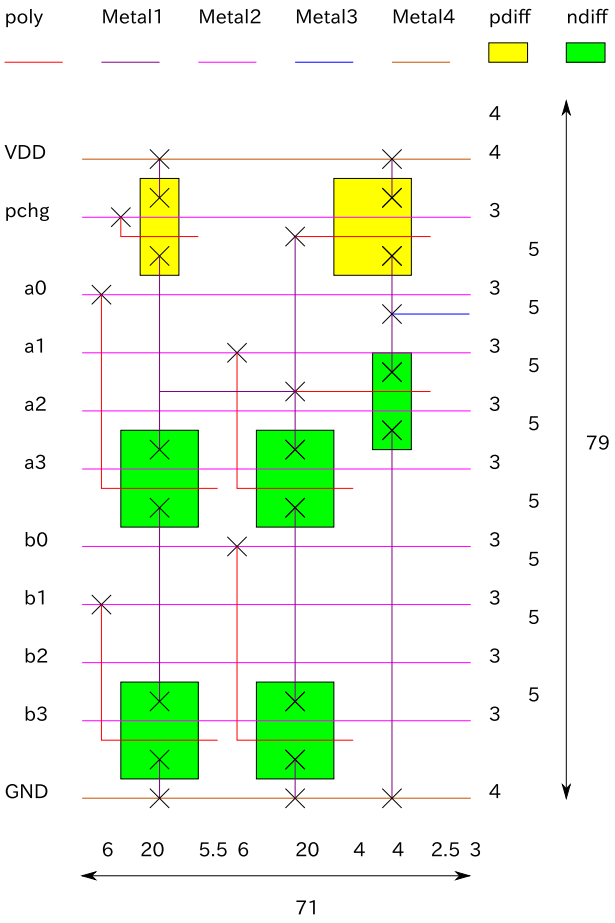

図 5.9: adder_2 のレイアウト

adder_2 との相違は NMOS が横に並ぶ個数なので，adder の入力が 1 つ増えるたびに，NMOS1 つ分の横幅である 31.5λ 増える．よって adder_n の横幅は $(8 + 31.5 * n)\lambda$ である．

以上のように任意の adder の横幅を求めることができたので，選択論理のセルの横幅を計算することができる．例えば，1 つ目のセルは adder_1，adder_2，adder_1 で構成されるので，その横幅は $(51.5+71+51.5)\lambda = 174\lambda$ である．発行幅が 4 の場合，3 つ目以降のセルは adder1，adder2，adder3，adder4 で構成されるので．その横幅は $(51.5 + 71 + 102.5 + 134)\lambda = 359\lambda$ である．

5.2.4 回路構成

選択論理の回路図を図 5.10 に示す．図は発行幅が 4，エントリ数が 64 の場合を示している．

選択論理のクリティカルパスは 2 番目のエントリから始まり，63 番目までの発行要求の個数を数え，64 番目のエントリの発行許可を決定するパスである．図中の四角形は選択論理のセルである．各セルは四角形内で示された数字の adder で構成される．セルの横にあるインバータは，発行許可の決定に必要な，それまでの計算結果を伝えるためのドライバの 1 段目である．セルを通過する毎に縦断するエントリ数が増えるので，配線が長くなる．その配線に長さによっては，ドライバやリピータが必要になる場合がある．最後まで計算したら，発行要求信号とのダイナミック AND で発行許可信号を生成する．そして最後に生成した発行許可信号を選択論理の左端まで伝える．select logic width は選択論理全体の横幅である．これは発行幅 4，エントリ数 64 の場合，$(174 + 296 + 359 + 359 + 359 + 359)\lambda = 1906\lambda$ である．図では最後の配線の抵抗と容量を 1 つに集中させているが，SPICE のネットリスト上では adder1 個分の大きさの配線抵抗と配線容量に分散させている．

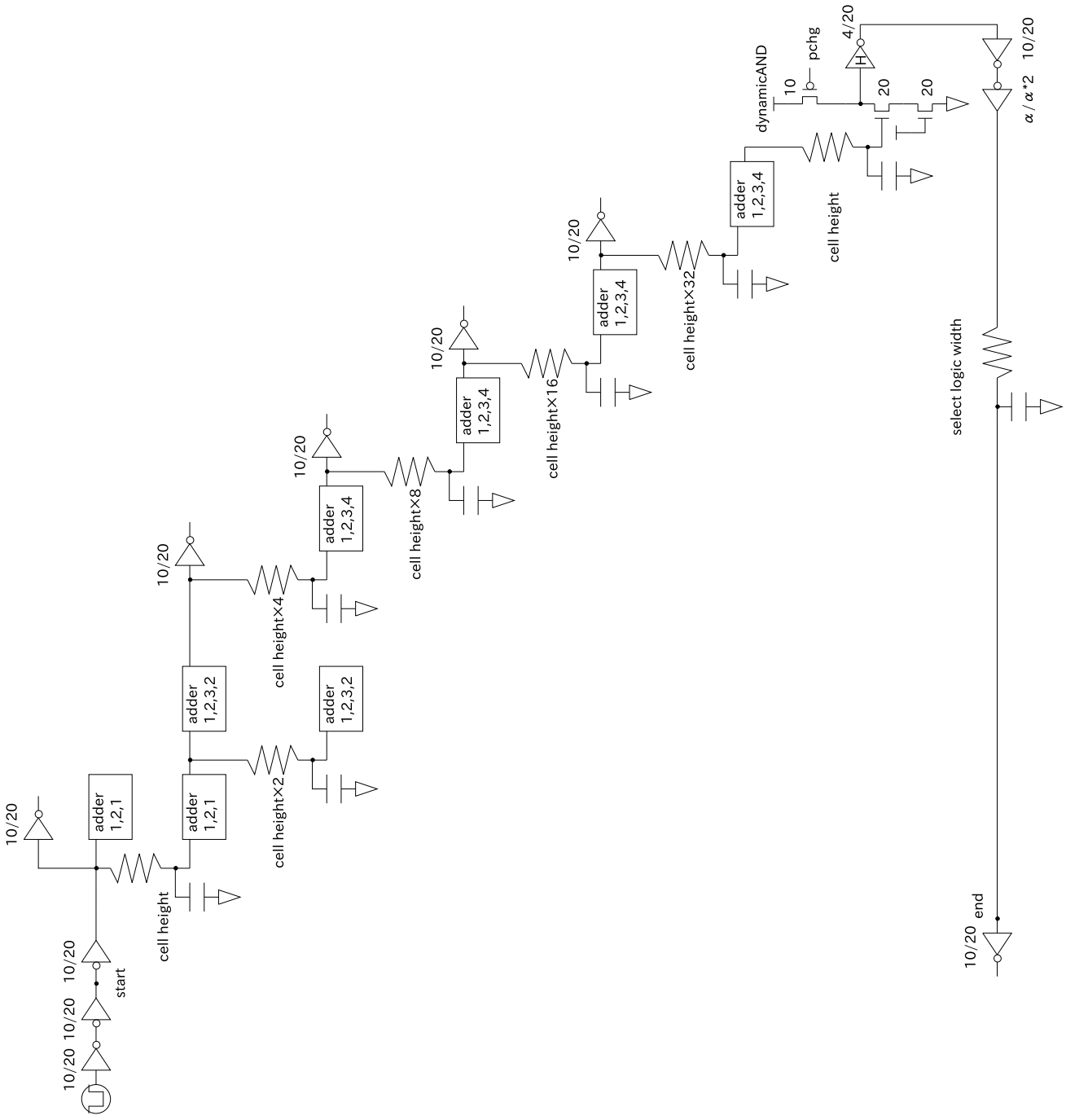

図 5.10: 測定に用いる選択論理の回路図

5.3 タグ RAM

5.3.1 レイアウト図

タグ RAM のセルは図 5.1 に示す 6T SRAM に別途読み出しビット線を設けたものである．そのレイアウトを図 5.11 に示す．この回路の縦幅は図 5.2 に示した比較器と同じく $(18.5 * iw)\lambda$ なので，ウェイクアップ論理と同じになる．一方で横幅は，$(17 + 6 * iw)\lambda$ なので，タグ RAM セルの横幅はウェイクアップ論理のセルとは異なる．5.1.1 節で示したように，SRAM の横幅は $(31 + 14 * iw)\lambda$ なので，SRAM と図 5.11 に示した回路との間隔を考慮すると，タグ RAM セルの横幅は $(31 + 14 * iw)\lambda + (17 + 6 * iw)\lambda + 3\lambda + 3\lambda = (54 + 20 * iw)\lambda$ となる．

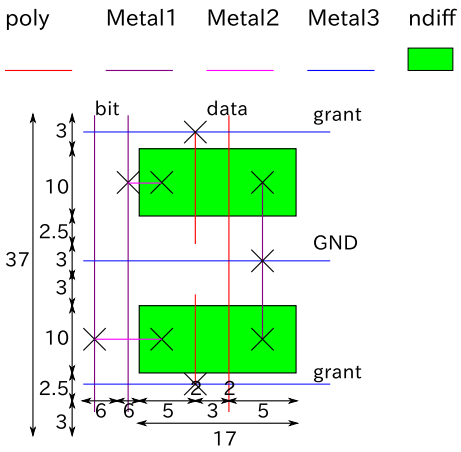

図 5.11: RAM の読み出しのための回路

5.3.2 回路構成

タグ RAM を構成する回路を図 5.12 に示す．図に示されたタグ RAM は高速化のためにバンク化されている．

右上のパルス波によってワード線がアサートされ，アクセストランジスタを駆動する．メモリには 1 が保持されており，ローカルビット線がディスチャージされる．これを立ち上がり優先 NOT によって素早く検知し，グローバルビット線をディスチャージする．グロー

バルビット線のディスチャージも立ち上がり優先 NOT によって検知し，これの出力をタグとする．

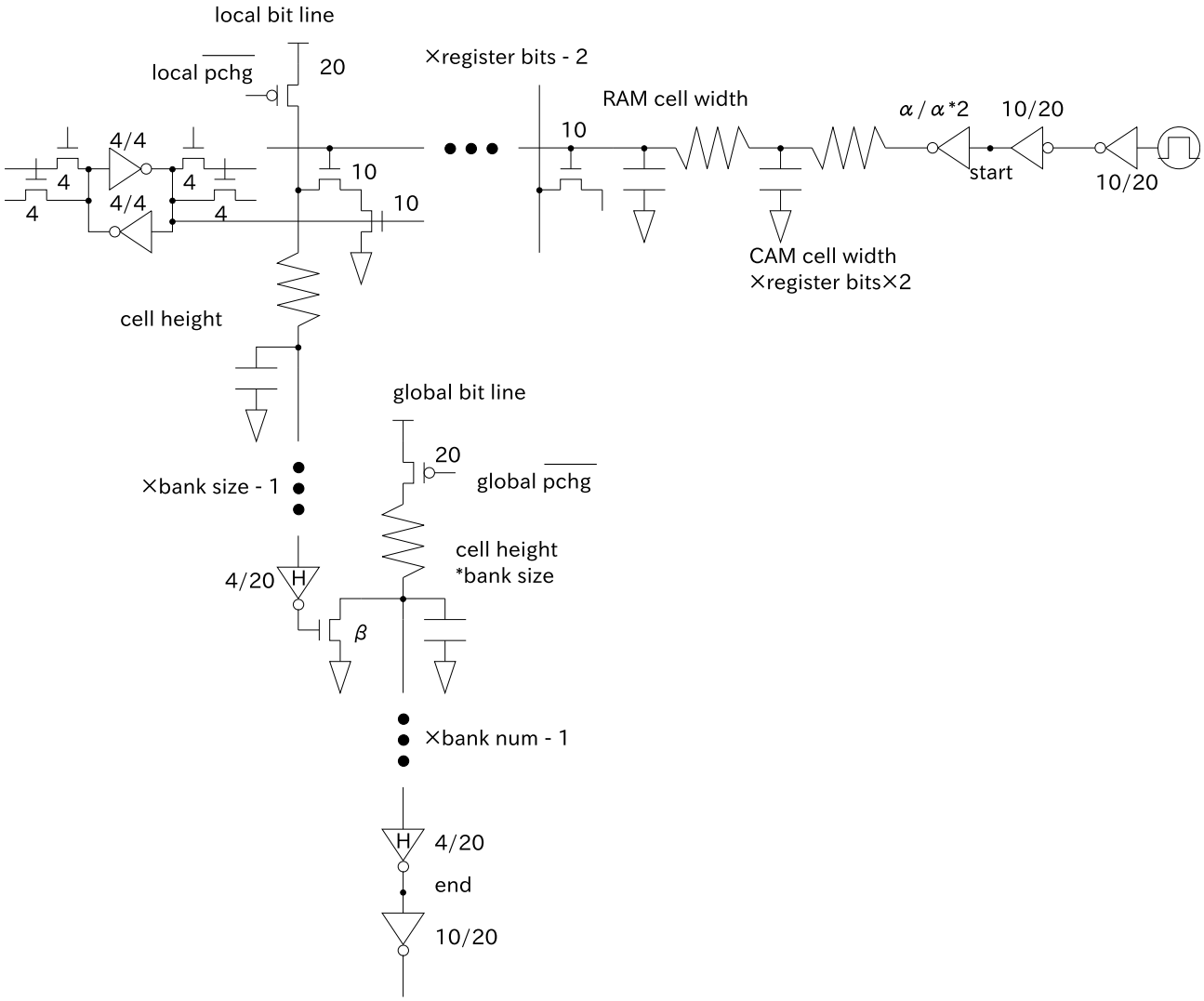

図 5.12: 測定に用いるタグ RAM の回路図

5.4 ペイロード RAM

ペイロード RAM はタグ RAM と同じく RAM で構成されるので，基本的にタグ RAM と同様である．違いは RAM に保持されるビット数と，ワード線の長さである．

ペイロード RAM は機能ユニットに送る命令の情報を保持する．本研究では，命令長を

32 ビット，論理レジスタのビット数を 5 ビットと想定している．また．7 章の表 7.1 に示す基本構成のように，物理レジスタを int,fp 合わせて 256 個用意するので，その時の物理レジスタ番号を表すビット数は 8 ビットである．ペイロード RAM では，命令に含まれる 3 つの論理レジスタ番号は全て物理レジスタ番号に変更され，命令全体を保持する．そのため，本研究におけるペイロード RAM のビット数は 32-5*3+8*3=41 ビットである．このためペイロード RAM のワード線は，タグ RAM やウェイクアップ論理よりも長くなっている．

また，図 5.13 に示すように，タグ RAM，ウェイクアップ論理，ペイロード RAM では命令ディスパッチ時に同時にワード線がアサートされる．ワード線の駆動の遅延を削減するため，ペイロード RAM は他の論理とは独立したワード線を持つ．一方で，タグ RAM とウェイクアップ論理はワード線を共有している．

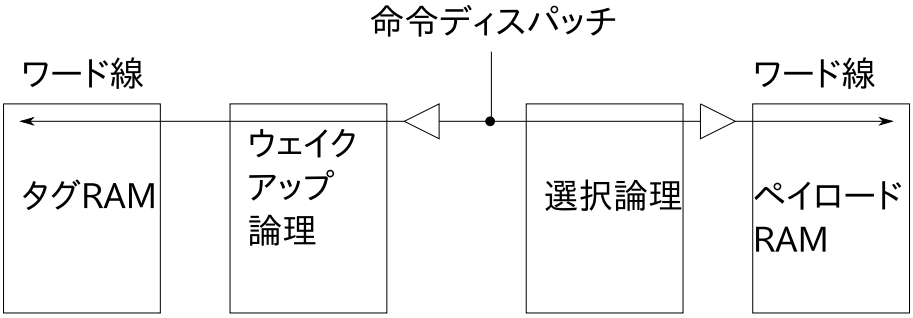

図 5.13: 発行論理のワード線

5.5 エイジ論理

実際のエイジ論理では，図 3.3 で示した回路とは異なり，ゲート数を少なくするために負論理で動作する．そのため，発行要求信号そのものではなく，反転した負の発行要求信号がエイジ論理の中を伝わる．加えて各セルの SR ラッチの Q ではなく $\overline{Q}$ と NOR をとり，その出力によってマッチ線をプルダウンする．これに基づいて，エイジ論理のレイアウトと回路構成について詳しく述べる．

5.5.1 レイアウト図

エイジ論理を構成するセルのレイアウトを図 5.14 に示す．複雑な構成となっているため，スティック図ではなく詳細なレイアウト図を示す．

図の下半分の拡散層 6 つは，SR ラッチを構成する NOR ゲートを 2 つを表している．その上の拡散層 3 つは NOR ゲートを表し，一番上の拡散層はプルダウントランジスタを表す．このようにレイアウトを設計した結果，エイジ論理のセルの縦幅は 86λ，横幅は 56λ となった．この範囲に収めるためには，プルダウントランジスタのサイズは 48λ 以下でなければならない．

5.5.2 回路構成

SPICE シミュレーションに用いたエイジ論理の回路構成を図 5.15 に示す．図は発行幅が 2 の場合を表す．点線で囲まれた部分はエイジ論理を構成する 1 つのセルを表す．シミュレーションにおいては命令ディスパッチ時の動作について調査せず，発行許可時の動作についてのみ調査する．そのため，命令ディスパッチ時に変動する SR ラッチが保持する値はセルごとに任意に決定してもよい．このことを図中のセルでは VDD と GND を切り替えるスイッチで表している．この回路では，あるエントリ j から出力された発行要求信号によって，あるエントリ i の発行許可を打ち消すまでの遅延を測定する．この場合，スイッチが示す値はセル $[i,\ j]$ のみ 1 であり，他のセルは全て 0 である．

図の左下のように，発行要求を送るエントリ j にパルス波を出力する電源を置く．反転した発行要求が行を伝わり，対角線上のセルから他の行にも伝わる．そして，発行許可を打ち消したいエントリ i において NOR ゲートの出力が立ち上がり，プルダウントランジスタが駆動する．その後，NOT 及び NOR を通じて発行許可信号が打ち消される．

エイジ論理で得られた結果と選択論理で選択した結果が重複した時，同一命令の複数発行が生じる．これを防ぐために，各論理から得られた結果を調停する必要がある．そのための回路が図の左上に備えられている．エイジ論理の出力 $winner$ が 0 の時，選択論理の発

行許可が有効になる．この調停の結果を得られるまでの遅延がエイジ論理，及びエイジ論理を用いる場合の選択論理の遅延となる．

5.6 RRQ

命令移動とディスパッチの遅延を測定するための回路を図 5.16 に示す．これらの遅延は，ワード線の長いペイロード RAM において最も長くなるため，ペイロード RAM においてこれらの遅延を測定する．図はペイロード RAM において発行幅が 2 の場合を表す．左上の SRAM が MQ を，下側の SRAM が OQ をそれぞれ表す．1 つの回路で異なる動作を行うため，パルス波の入力が多い．命令移動とディスパッチにわけて回路の動作を説明する．

5.6.1 命令移動

右上の main read word のパルス波によって，MQ のワード線がアサートされる．メモリに保持された値によって一方のローカルビット線がディスチャージされ，センスアンプで 2 つのビット線の電位差を検出する．他のバンクのセンスアンプの出力と NAND をとり，グローバルビット線をプルダウンする．これを立ち上がり優先 NOT で検出し，その出力を OQ の書き込みデータとする．old write enable がアサートされれば，OQ のビット線がディスチャージされる．その後，old word によって OQ のワード線がアサートされるので，OQ に対して書き込みが行われる．

この回路では，ローカルビット線がプリチャージされた状態におけるセンスアンプの不定な出力を 0 に近づけるために，インバータを 3 個つなげて出力を安定させている．本来は，センスアンプの出力をゲーティングし，安定した出力が得られるタイミングでイネーブルをオンにして，グローバルビット線をプルダウンする回路を構成するべきである．しかし，イネーブルを入れるべきタイミングは回路のパラメータによって変化し，あらかじめ求めることはできない．そこで本研究ではシミュレーションを容易にするために，インバータを複数つなげることによってセンスアンプの出力を常に安定させる回路を構成して

いる．

5.6.2 ディスパッチ

命令移動のための読み出しが行われた後，main write pchg によってビット線をプリチャージする．このとき，グローバルビット線を通じて OQ の書き込むデータが変化してしまう．これを防ぐために，MQ のビット線がプリチャージされるとき，及びディスパッチするときは，pass gate enable をオンにしパスゲートによって MQ と OQ を分断する．write data と main write enable がアサートされ，ビット線がディスチャージされる．その後，main write word によって MQ のワード線がアサートされ，MQ に値が書き込まれる．

5.6.3 発行論理を構築する各論理の大きさ

最後に書く論理の大きさについて述べる．エイジ論理を用いる発行論理は図 5.17 に示すような大きさの回路で構成される．この図はエントリ数が 64，発行幅が 4 の場合を示している．

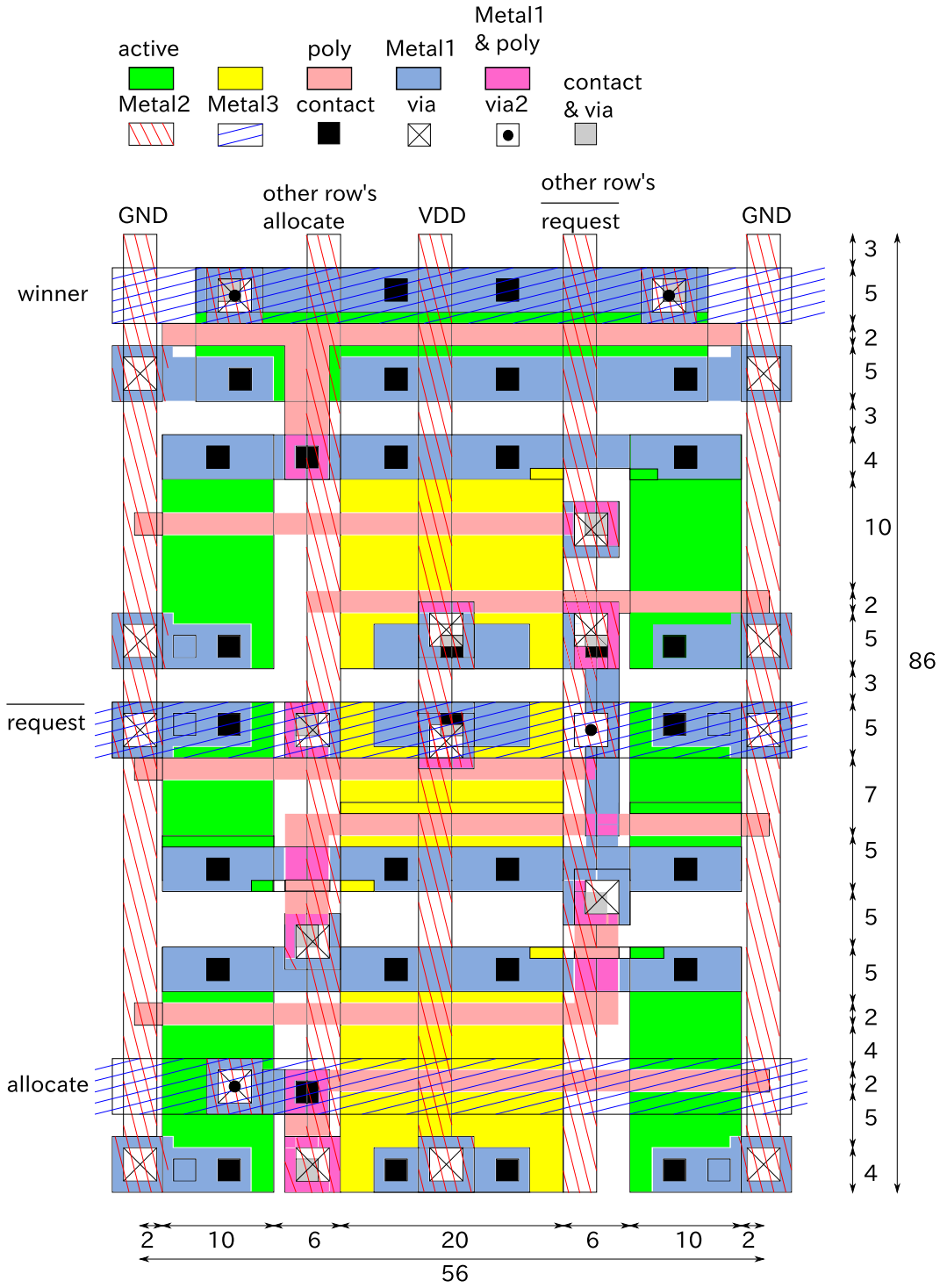

図 5.14: エイジ論理のセルのレイアウト

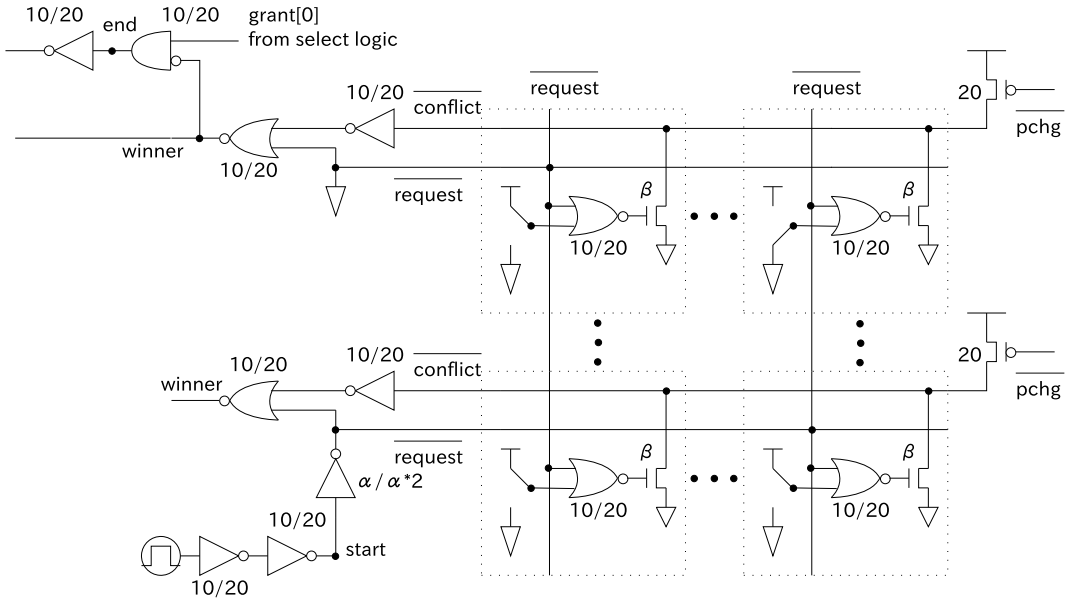

図 5.15: エイジ論理の回路構成

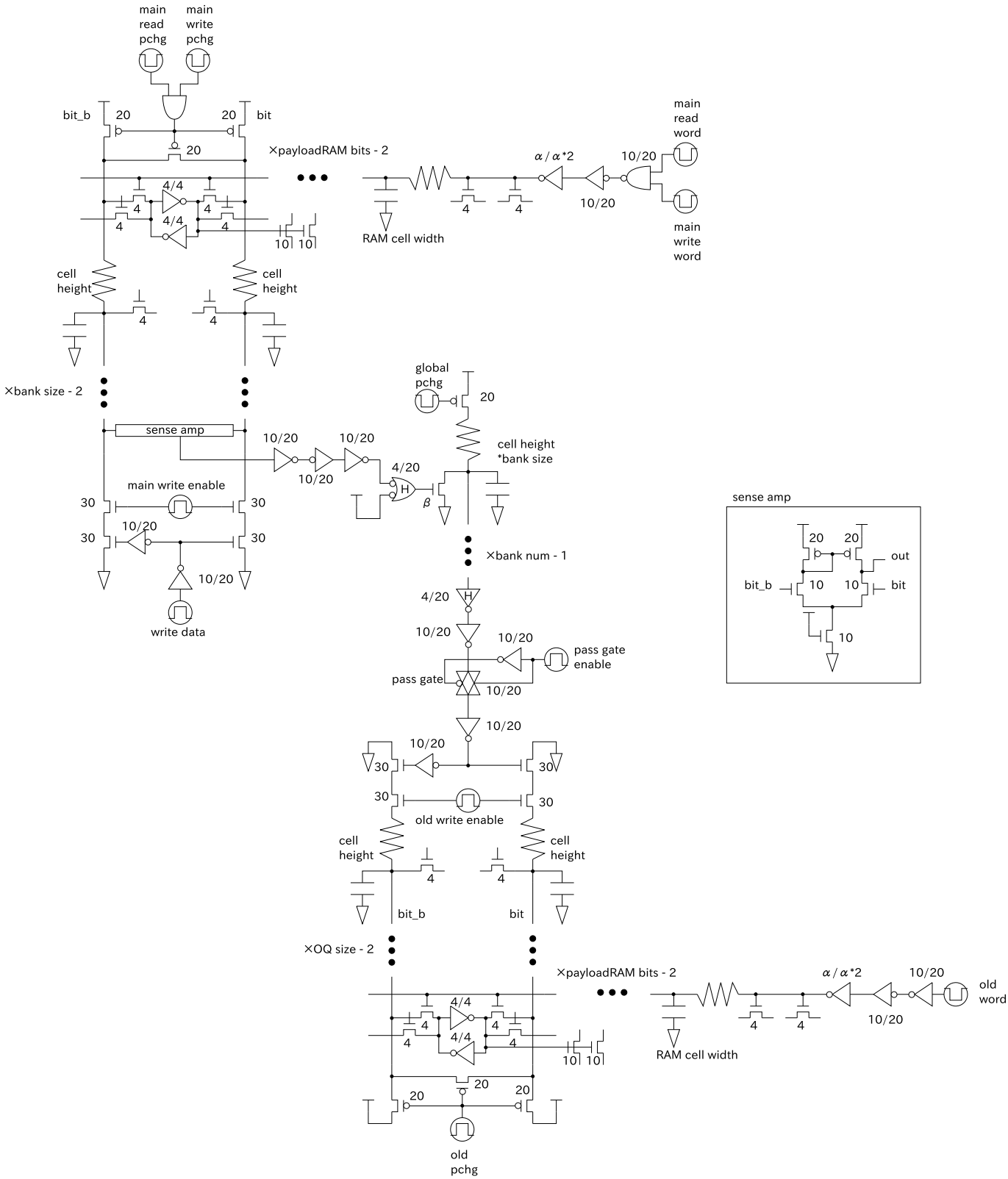

図 5.16: 命令移動とディスパッチの遅延を測定する回路図

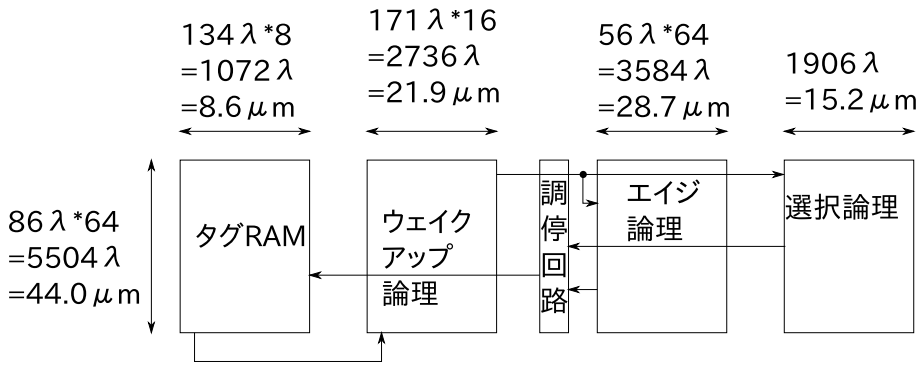

図 5.17: 選択論理で用いる加算器

第 6 章　回路の遅延評価

SPICE による回路シミュレーションによって発行論理の遅延を測定した．回路のレイアウトは MOSIS デザインルール [14] に則って設計した．16nm プロセスを仮定し，SPICE のトランジスタモデルとして，アリゾナ州立大学の Predictive Technology Model(PTM) [15] を使用した．International Technology Roadmap for Semconductors(ITRS) [16] により公開されているデータより，単位長さあたりの配線抵抗 $46.96M\Omega/m$ と配線容量 $0.165nF/m$ を求め，これを用いた．発行論理，エイジ論理，RRQ の遅延についてそれぞれ順に述べる．

6.1　発行論理の遅延

ウェイクアップ論理，選択論理，タグ RAM に分けて，様々な発行幅とエントリ数に応じて，ドライバやリピータのサイズについて適切な値を求めた．

6.1.1　ウェイクアップ論理の遅延

ウェイクアップ論理において測定した結果を表 6.1 に示す．iw は発行幅，IQS はエントリ数を表す．これ以降，論理ゲートのサイズとは，その論理ゲートを構成する NMOS のゲート幅 [λ] を表す．tag_driver(α) はタグ線のドライバのサイズである．tag_repeater_num はタグ線に挿入するリピータの個数である．これらのリピータのサイズは tag_driver のサイズと同一とした．match_repeater(β) はマッチ線のリピータのサイズである．マッチ線のリピータの個数は 1 個で固定である．これらの値の内，α と β は図 5.3 で示した α，β を指している．

	tag_driver(α)[λ]	tag_repeater_num	match_repeater(β)[λ]	delay[ps]
iw=4, IQS=64	40	0	60	93.6
iw=8, IQS=64	35	1	65	129.0
iw=4, IQS=128	45	1	65	123.1
iw=8, IQS=128	35	3	80	175.8
iw=4, IQS=192	45	2	75	153.0
iw=8, IQS=192	40	5	75	223.0
iw=4, IQS=256	40	3	75	181.4
iw=8, IQS=256	40	7	80	267.3

表 6.1: ウェイクアップ論理のトランジスタサイズと遅延

6.1.2 選択論理の遅延

選択論理において測定した結果を表 6.2 に示す．選択論理では，加算器を通過する毎に出力の配線が縦断するエントリ数は多くなる．例えば，1 段目の加算器の出力は 2 エントリしか縦断しないが，6 段目の出力は 64 エントリを縦断する．縦断するエントリ数が多いと配線が長くなり，ドライバ及びリピータが必要になる．その時のドライバやリピータを投入する間隔を space，そのサイズを size と表記する．これらが – と記されている場合はドライバやリピータが一切不要ということである．space と縦断するエントリ数が一致する場合，リピータは不要であり，ドライバのみが挿入されていることを示す．grant_driver(α) は選択論理の最後に発行許可信号を増幅するドライバのサイズである．この表の α は図 5.10 で示した α を指す．

	grant_driver(α)[λ]	space	size[λ]	delay[ps]
iw=4, IQS=64	-	-	-	78.9
iw=8, IQS=64	40	32	40	112.1
iw=4, IQS=128	-	64	45	107.1
iw=8, IQS=128	45	32	40	162.4
iw=4, IQS=192	-	54	40	135.3
iw=8, IQS=192	45	64	50	210.9
iw=4, IQS=256	-	128	50	156.5
iw=8, IQS=256	45	64	45	248.8

表 6.2: 選択論理のトランジスタサイズと遅延

6.1.3 タグ RAM の遅延

タグ RAM において測定した結果を表 6.3 に示す．基本的なバンクサイズは 8 とした．driver(α) は発行許可信号のドライバのサイズである．pulldown(β) はグローバルビット線をプルダウンするトランジスタのゲート幅である．repeater_size はグローバルビット線に挿入するリピータのサイズであり，repeater_num はその個数である．表内の α 及び β は図 5.12 で示したものを指す．

	driver(α)[λ]	pulldown(β)[λ]	repeater_size[λ]	repeater_num	delay[ps]
iw=4, IQS=64	45	60	-	0	50.8
iw=8, IQS=64	50	70	-	0	75.6
iw=4, IQS=128	45	60	60	1	70.4
iw=8, IQS=128	55	70	60	3	106.3
iw=4, IQS=192	45	55	60	2	88.2
iw=8, IQS=192	55	70	60	5	138.2
iw=4, IQS=256	45	55	60	3	106.8
iw=8, IQS=256	55	80	60	7	166.9

表 6.3: タグ RAM のトランジスタサイズと遅延

図 6.1 にバンク数を変化させた時のタグ RAM の遅延を示す．

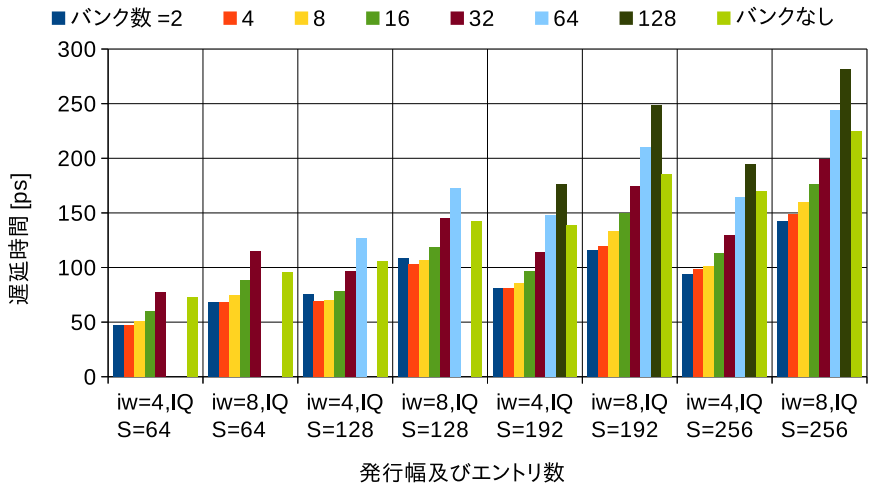

図 6.1: バンクサイズを変化させた時のタグ RAM の遅延

6.2 エイジ論理の遅延

図 6.2 のように，タグ RAM，ウェイクアップ論理，エイジ論理，選択論理と並べた時のエイジ論理の遅延について測定した結果を表 6.4 に示す．driver(α) は発行要求信号を駆動するドライバのサイズである．repeater_position は縦と横に走る配線 $\overline{request}$ 上にリピータを設置する位置を表す．例えば，64 エントリで 50,100 ならば，横に走る配線上の左から 50 番目のセルの次と，そこから 50 個のセルを通過後，つまり縦に走る配線上の上から (64*2-100)=28 番目のセルの次にリピータを挿入することを示す．これらのリピータのサイズは driver に等しい．pulldown(β) は各セル中にあるプルダウントランジスタのゲート幅である．repeater_size，及び repeater_num はそれぞれ $\overline{conflict}$ に挿入するリピータのサイズと個数である．表中の α と β は図 5.15 で示されたものと同一である．

また，64 エントリ，4 発行幅の時，図 6.2 中の赤い矢印で表された選択論理を通るパスにおいて，エイジ論理を横断する配線を信号が往復する時間と，調停に要する時間はそれぞれ 17.7ps，11.5ps であった．その他のエントリ数および発行幅の場合の結果を表 6.5 に示す．左から順に信号が往復する距離，配線遅延，調停の遅延時間を表す．

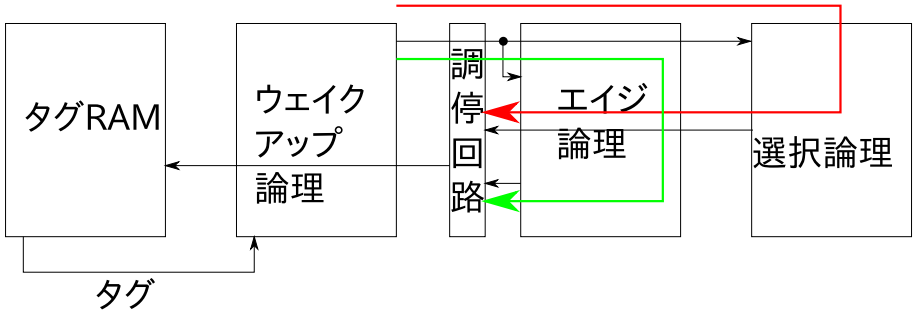

図 6.2: ウェイクアップ論理と選択論理も間にエイジ論理を挿入した発行論理

また，エイジ論理は図 6.3 のように，エイジ論理と選択論理を互いに入れ替えて配置することもできる．この時の選択論理を横断する配線を信号が往復する時間と，調停に要する時間を表 6.6 に示す．選択論理を往復する場合，表 6.5 と比べて調停に要する時間が短くなっている．この理由は，選択論理を距離が短いので，その間を走る配線の電位が素早く

	driver(α) [λ]	repeater position	pulldown(β) [λ]	repeater size[λ]	repeater num	delay[ps]
iw=4, IQS=64	40	79	36	-	0	96.2
iw=8, IQS=64	40	79	36	-	0	98.7
iw=4, IQS=128	60	124,172,212	35	40	1	167.1
iw=8, IQS=128	60	124,172,212	35	40	1	169.4
iw=4, IQS=192	60	176,240,288,336	42	60	3	240.0
iw=8, IQS=192	60	176,240,288,336	42	60	3	242.4
iw=4, IQS=256	60	112,240,304,352,400,464	47	60	3	314.2
iw=8, IQS=256	60	112,240,304,352,400,464	47	60	7	316.9

表 6.4: エイジ論理のトランジスタサイズと遅延

	length[λ]	wiring[ps]	arbitor[ps]
iw=4, IQS=64	3584*2	17.7	11.5
iw=8, IQS=64	3584*2	17.8	12.4
iw=4, IQS=128	7168*2	40.0	13.8
iw=8, IQS=128	7168*2	45.6	14.0
iw=4, IQS=192	10752*2	69.9	16.4
iw=8, IQS=192	10752*2	77.4	17.5
iw=4, IQS=256	14336*2	105.3	19.4
iw=8, IQS=256	14336*2	115.3	20.6

表 6.5: エイジ論理を信号が横断する距離とその遅延と調停に要する遅延

変化し，調停回路の動作も素早く完了するためである．

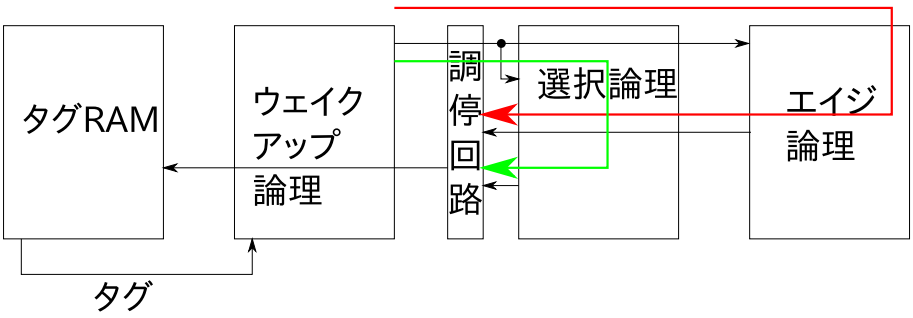

図 6.3: エイジ論理と選択論理を入れ替えた発行論理

6.3 発行論理の遅延

発行論理を構成する各部品のクリティカルパスの遅延をを合計し，得られた発行論理の遅延を図 6.4 に示す．この時のエントリ数は 64，発行幅は 4，タグのビット数は 8 とした．

	length[λ]	wiring[ps]	arbitor[ps]
iw=4, IQS=64	1906*2	6.6	9.7
iw=8, IQS=64	4983.5*2	26.1	10.7
iw=4, IQS=128	2265*2	7.0	9.7
iw=8, IQS=128	6193.5*2	33.8	10.7
iw=4, IQS=192	2624*2	8.3	9.6
iw=8, IQS=192	7403.5*2	41.7	10.7
iw=4, IQS=256	2624*2	8.6	9 .7
iw=8, IQS=256	7403.5*2	42.2	10.8

表 6.6: 選択論理を信号が横断する距離とその遅延と調停に要する遅延

縦軸は遅延時間を表す．5 つある棒グラフの内，一番左の棒グラフは，エイジ論理を用いない発行論理の遅延を表す．残り 4 つはエイジ論理を用いた発行論理の遅延を表し，左側 2 つは図 6.2 のように発行論理の各部品が並んだ場合を表し，右側 2 つは図 6.3 のように並んだ場合を表す．それぞれの 2 つの棒グラフの中で，左側が各図中の赤い矢印で示されたパスを通る遅延を，右側が緑の矢印で示されたパスを通る遅延を表す．

iw=4,IQS=64 で，エイジ論理を用いない通常の発行論理では，ウェイクアップ論理に 93.6ps，選択論理に 78.9ps，タグ RAM 読み出しに 50.8ps の時間を要し，その合計は 223.3ps となった．ここで，エイジ論理を導入すると，途中の論理を横断する遅延と調停による遅延が加わる．その結果，図 6.2 のように並べた方がクリティカルパスは短くなり，その遅延は 252.6ps となった．このようにエイジ論理を導入することによって，13.1%も発行論理の遅延が増加する．

各発行幅，各エントリ数について同様にクリティカルパスの遅延を求めた結果を表 6.7 に示す．各列は左から，発行論理のレイアウト，ウェイクアップ論理，タグ RAM，選択論理，エイジ論理，横断，調停，及びそれらの合計を表し，一番右の列はエイジ論理を用いない通常の発行論理の遅延を表す．単位はいずれも ps である．

表より，iw=4，IQS=64 の場合のみ選択論理を通るパスがクリティカルパスであり，それ以外の場合はエイジ論理を通るパスがクリティカルパスとわかる．これは，エントリ数の増加に対して，選択論理を構成するセルが対数的に増加していくのに対して，エイジ論

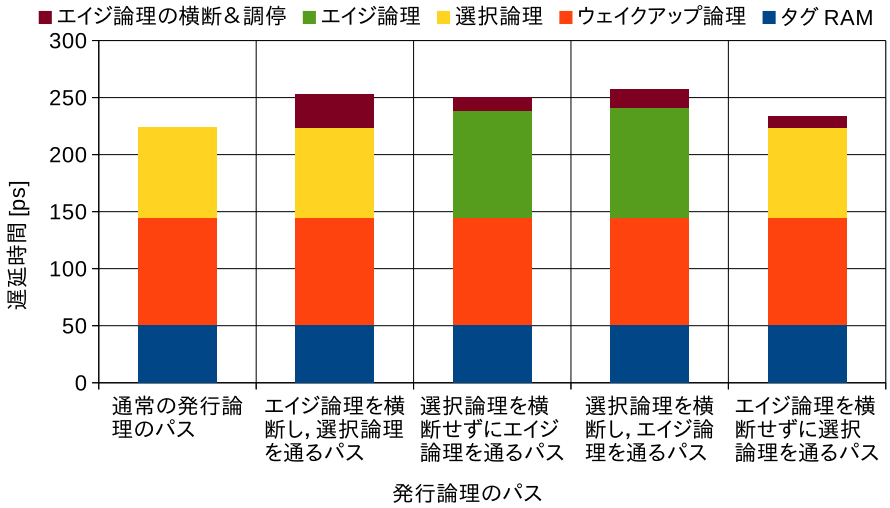

図 6.4: 発行論理の遅延

理を構成するセルは線形に増加していくので，遅延がより大きくなるためである．

また，iw=4 の時は図 6.2 のように，iw=8 の時は図 6.3 のように発行論理を構成したほうが，もう一方のように構成するよりも遅延が短くなる．これは発行幅が増加すると選択論理の遅延が大きく増加するため，発行幅が大きい場合，選択論理をウェイクアップ論理のより近い位置に配置したほうがクリティカルパスの遅延の増加を抑制できるためである．

	layout	wakeup	tagRAM	select	age	wiring	arbitor	sum	normal
iw=4, IQS=64	図 6.2	93.6	50.8	78.9	-	17.7	11.5	252.6	223.3
iw=8, IQS=64	図 6.3	129.0	75.6	-	98.7	26.1	10.7	340.1	316.7
iw=4, IQS=128	図 6.2	123.1	70.4	-	167.1	40.0	13.8	370.0	300.7
iw=8, IQS=128	図 6.3	175.8	106.3	-	169.4	33.8	10.7	496.1	444.5
iw=4, IQS=192	図 6.2	153.0	88.2	-	240.0	69.9	16.4	490.7	376.5
iw=8, IQS=192	図 6.3	223.0	138.2	-	242.4	41.7	10.7	656.0	572.0
iw=4, IQS=256	図 6.2	181.5	106.8	-	314.3	105.3	19.4	612.2	444.8
iw=8, IQS=256	図 6.3	267.3	166.9	-	316.9	42.2	10.8	804.1	683.0

表 6.7: エイジ論理を用いた発行論理のクリティカルパスの遅延 [ps]

以上のようにエイジ論理の導入により，発行幅が 4，エントリ数が 64 の場合は，発行論理の遅延が 223.3ps から 252.6ps まで，13.1%も増加させる．そのため，発行論理の遅延を増加させることなく，ランダムキューの IPC を向上させる手法が必要である．

6.4 命令移動とディスパッチの遅延

命令移動とディスパッチは，タグRAM，ウェイクアップ論理，ペイロードRAMの各論理で行われる．これらの内，ペイロードRAMの保持するビット数が最も多いので，ペイロードRAMの遅延が最も大きくなる．そのため，ペイロードRAMにおける命令移動とディスパッチの遅延を測定した．

ワード線のドライバ及びリピータのサイズを変化させて，MQからの読み出し，及びMQへの書き込みに要する遅延を測定した．その結果を図6.5に示す．図はエントリ数が64の時の結果を示す．横軸はドライバとリピータのサイズ，縦軸はMQの読み書きの遅延の合計を表す．各曲線は発行幅とリピータの挿入数が異なる場合の遅延を表す．

発行幅が8の時，65λのドライバとリピータを1つ挿入した時，遅延が最小となった．一方で発行幅が4の時，リピータを入れないほうが遅延は短くなり，ドライバのサイズが大きくなるほど遅延は減少した．発行幅8の時と同じ65λの場合，遅延は十分小さくなっている．

エントリ数が増加すると，物理レジスタ数が増加するので，それに伴ってペイロードRAMの保持するビット数が増え，ワード線が長くなる．しかし，大きなエントリ数についても図6.5と同じ傾向を示し，サイズが65λの時に発行幅が4の場合，遅延は十分小さくなり，発行幅が8の場合，最小になる．よって，今後の測定でも，ワード線のドライバとリピータのサイズは65λとする．

次に各発行幅，各エントリ数でプルダウントランジスタのサイズを変化させて遅延を測定した．その結果を図6.6に示す．各曲線は異なる発行幅とエントリ数におけるに遅延を表す．それぞれの場合において遅延が最小になるプルダウントランジスタのサイズとその遅延を表6.8に示す．表に示されたようにいずれの場合でも，トランジスタのサイズが60λの場合に遅延は最小となりうる．そして，図を見る限りどの場合においても，サイズが60λ以上の時，遅延はほとんど変化しないので，以降の測定では全ての場合においてプルダウ

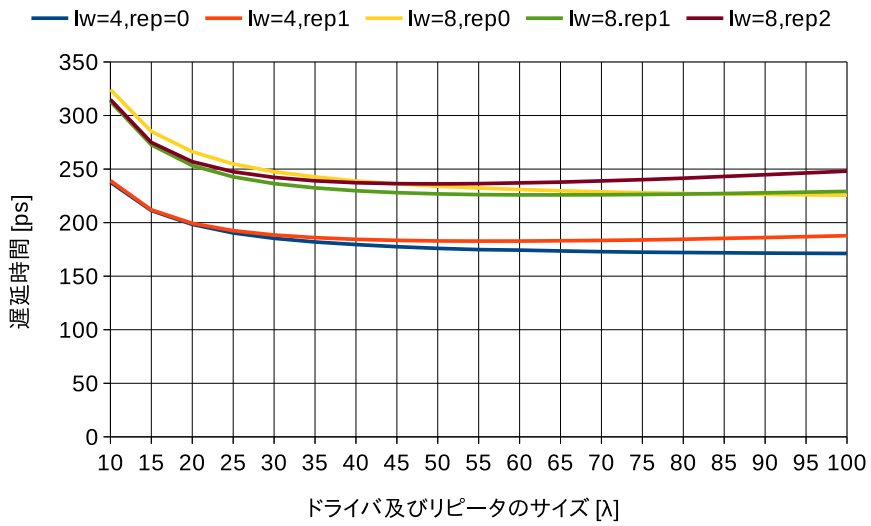

図 6.5: ワード線のドライバとリピータのサイズを変化させた時の MQ の読み書きの遅延の合計

ントランジスタのサイズを 60λ として測定した．

	size[λ]	delay[ps]
iw=4, IQS=64	90	166.1
iw=8, IQS=64	60	223.1
iw=4, IQS=128	85	190.9
iw=8, IQS=128	100	277.3
iw=4, IQS=192	75	224.7
iw=8, IQS=192	95	356.2
iw=4, IQS=256	75	263.2
iw=8, IQS=256	100	451.2

表 6.8: MQ の読み書きの遅延の合計が最小となるプルダウントランジスタとその時の遅延

次にグローバルビット線にリピータを挿入する場合の遅延を測定した．その結果を表 6.9 に示す．この表は様々な発行幅とエントリ数における，遅延が最小となるリピータのサイズと個数を示す．size はリピータのサイズであり，num は挿入するリピータの個数である．delay はその時の遅延である．発行幅が 4，エントリ数が 64 の場合を除いて，リピータを挿入することによって遅延を短縮することができた．

最後に，バンクサイズを変化させて命令移動と命令ディスパッチと発行論理の遅延を並べた様子を図 6.7 に示す．図の一番上はエイジ論理を用いない発行論理の遅延を表す．そ

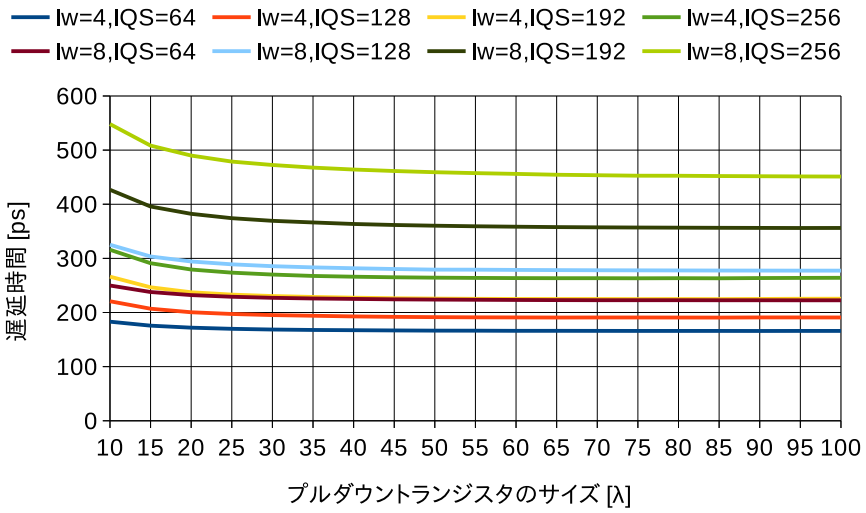

図 6.6: プルダウントランジスタのサイズを変化させた時の MQ の読み書きの遅延の合計

	size[λ]	num	delay[ps]
iw=4, IQS=64	-	-	166.1
iw=8, IQS=64	90	1	221.0
iw=4, IQS=128	70	1	190.7
iw=8, IQS=128	90	3	255.7
iw=4, IQS=192	75	2	217.7
iw=8, IQS=192	80	5	290.7
iw=4, IQS=256	70	1	239.4
iw=8, IQS=256	85	7	320.1

表 6.9: リピータのサイズと個数を変化させた時の MQ の読み書きの遅延の合計

の下はエイジ論理を用いる場合の発行論理の遅延を表す．それより下は，異なるバンクサイズにおける命令移動と命令ディスパッチの遅延を表す．これらの遅延はエントリ数が 64，発行幅が 4，OQ のサイズが 4 の場合を表す．ここから OQ のサイズを変化させると同じような結果が得られた．それらの遅延の合計を表 6.10 に示す．単位は全て ps である．

OQ のサイズが 4 の時の遅延の変化を見てみると，バンクサイズが 4 の時に，MQ への読み書きの遅延の合計が最小の 164.5ps となった．これら MQ の読み書きの遅延の合計は，発行論理の遅延よりも小さければ十分であるため，最小の遅延を追求する必要はない．しかし，バンクサイズを小さくすると，バンク毎にライトドライバやセンスアンプ，プルダウントランジスタなど余計な回路が増え，回路面積が大きく増加してしまうため，条件を

	OQ=4	OQ=8	OQ=12	OQ=16
2 entry / bank	165.1	163.8	162.5	161.4
4 entry / bank	164.5	163.2	162.4	161.1
8 entry / bank	166.1	165.2	164.4	163.2
16 entry / bank	171.7	170.9	170.0	169.3
32 entry / bank	184.2	183.3	182.4	181.6
no bank	192.0	188.6	185.0	181.2

表 6.10: バンクサイズと OQ のサイズを変化させた時の MQ の読み書きの遅延の合計

満たす限りできるだけ大きいバンクであることが望ましい．

エイジ論理を導入したことによって，発行論理の遅延が 13.1%増加したのに対して，RRQ では実装する上で回路面積を大きく増加させることはなく，また命令移動と命令ディスパッチが時分割できないという問題は発生しなかった．よって RRQ ならば，発行論理の遅延を増加させることなくランダムキューの IPC を向上させることができる．

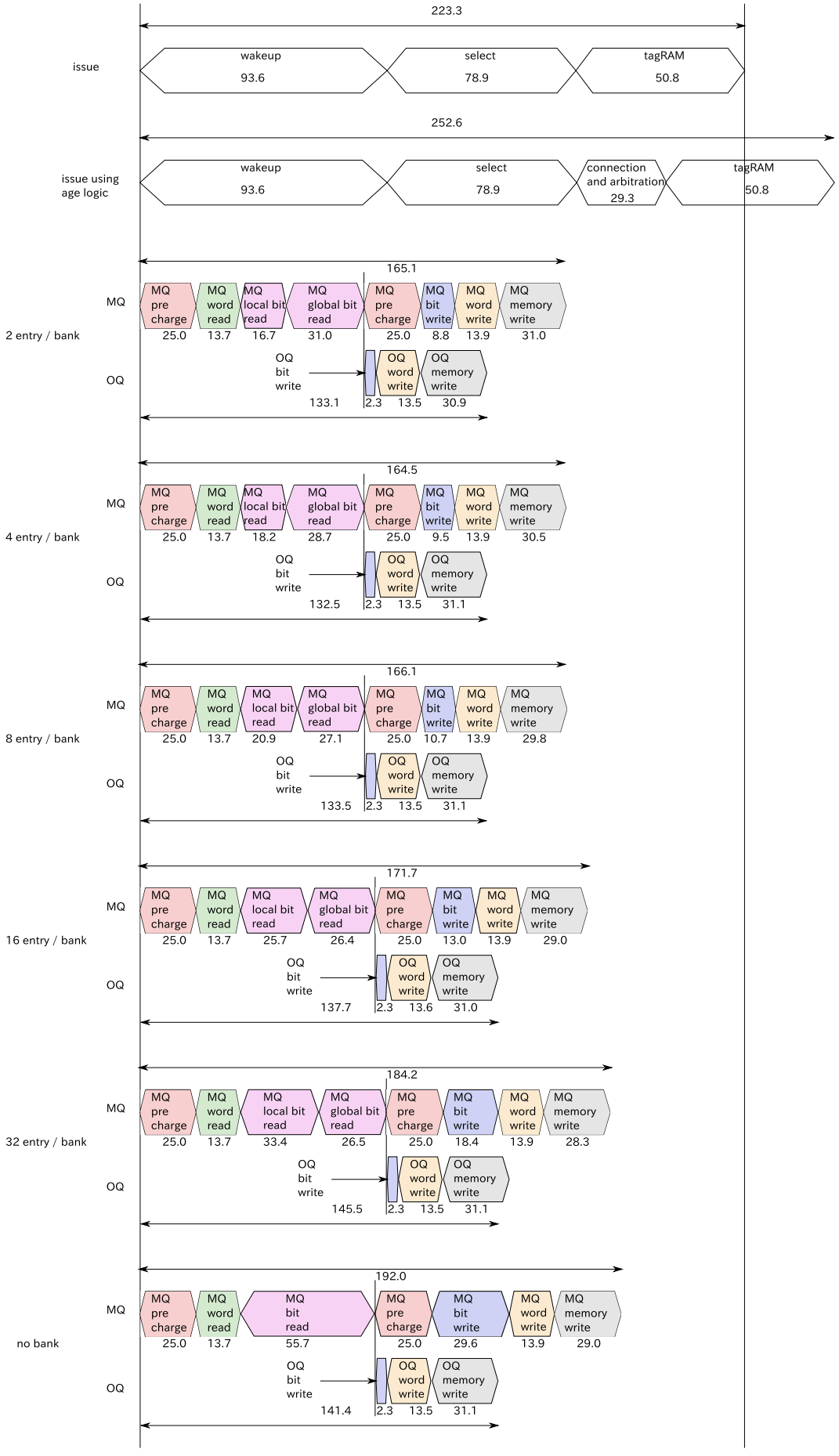

図 6.7: 発行論理及び命令移動と命令ディスパッチのタイミングチャート

第 7章　IPC評価

本章では発行論理を変化させた時のプロセッサのIPCについて評価する．SimpleScalar Tool Set Version 3.0a [18] をベースにRRQを実装したシミュレータを作成し，性能を評価した．命令セットはAlpha ISAである．ベンチマークプログラムには，現在のところシミュレータが正しく動作しないwrfを除いたSPEC CPU2006の28本を採用した．プログラムはgcc ver.4.5.3でオプション-O3を用いコンパイルした．プログラムへの入力にはrefデータ・セットを用いた．各ベンチマークの特徴を反映するために，SimPoint [19] を用いて適切な100M命令区間を求め，その範囲でシミュレーションを行った．

プロセッサの基本構成を表7.1に示す．1章で述べたように，命令がランダムに並ぶことによる性能低下は，レディな命令数が機能ユニットの数を上回った場合に生じる．このため，機能ユニットの数は評価において重要なパラメータとなる．本評価においては，NORMAL，LARGE，VERY_LARGEの3つの機能ユニットセットを用いた．各セットの内容を表7.2に示す．NORMALとLARGEはそれぞれARM Cortex-A72 [20]，Intel Skylake [21] をベースとしている．VERY_LAGEはNORMALの2倍の発行幅と機能ユニットを持つ．また，以下の5つのモデルで評価した．

- **shift**:IQがシフトキューで構成されたモデル
- **circular**:IQがサーキューで構成されたモデル
- **random**:IQが従来のランダムキューで構成されたモデル
- **age**:エイジ論理を導入したrandomモデル
- **RRQ**:IQがRRQで構成されたモデル

Pipeline width	4-instruction wide for each of fetch, decode, issue, and commit
ROB	128 entries
IQ	64 entries
LSQ	64 entries
PQ	128 entries
Physical Registers	128(int) + 128(fp)
Branch prediction	16-bit history 4K-entry PHT gshare, 2K-set 4-way BTB, 10-cycle misprediction penalty
Function unit	2 iALU, 1 iMULT/DIV, 2 Ld/St, 2 fpALU/MULT/DIV/SQRT
L1 I-cache	32KB, 8-way, 64B line
L1 D-cache	32KB, 8-way, 64B line, 2 ports, 2-cycle hit latency, non-blocking
L2 cache	2MB, 16-way, 64B line, 12-cycle hit latency
Main memory	300-cycle min. latency, 8B/cycle bandwidth
Data prefetcher	stride-based, 4K-entry, 4-way table, 16-data prefetch to L2 cache on miss

表 7.1: プロセッサの基本構成

7.1 オールドキューのサイズを変えた場合の評価

NORMAL セットにおいて OQ のサイズを変えて IPC を測定した．得られた結果を図 7.1 に示す．横軸は OQ のサイズ，縦軸は shift に対する IPC 低下率を示している．値が小さいほど IPC が良いことを表している．箱ひげグラフの縦線の上下の末端はそれぞれ IPC 低下率の最大値と最小値を表す．箱の上端と下端はそれぞれ第 3 四分位点と第 1 四分位点を表す．緑の十字は IPC の幾何平均における低下率を表す．MQ と OQ のサイズの和は，常に表 7.1 の IQ のサイズとなっている．OQ のサイズが 0 の時は，OQ が存在しない，つまり従来のランダムキューである random モデルに等価である．

OQ のサイズが 4 の時に，幾何平均の値，最大値共に最小となった．それぞれの値は 1.8%，及び 5.6%となった，OQ のサイズがこれより小さければ，複数の古い命令に高い優先度を

	NORMAL	LARGE	VERY_LARGE
pipeline width	4	8	8
iALU	2	4	4
iMULT/DIV	1	1	2
Ld/St	2	2	4
fpALU/MULT/DIV/SQRT	2	2	4

表 7.2: 機能ユニットの各種セット

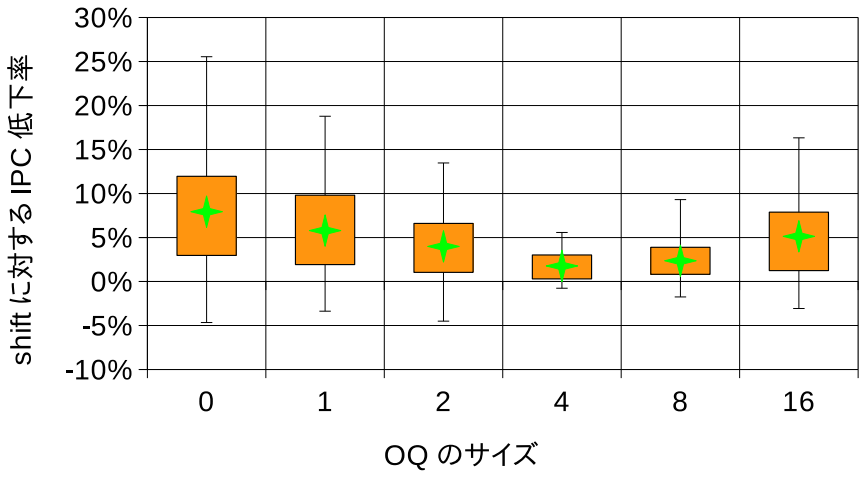

図 7.1: OQ のサイズを変えた場合の IPC 低下率

持たせることができないため，IPC が低下する．逆に OQ のサイズがこれより大きければ，OQ の中で重要な命令とそうでない命令が混在し，重要な命令にのみ高い発行優先度を与えることができないため，IPC が低下する．

NORMAL セットを用いて，エントリ数を 64 から 256 まで 64 エントリずつ増やして，同じように OQ のサイズを変化させて IPC を測定した．その結果を図 7.2 に示す．エントリ数を増やした場合でも，OQ のサイズが 4 の時に最も IPC 低下率が小さくなった．また，OQ のサイズが 8 の場合，図中の箱が小さくなっているので，多くのベンチマークにおいて IPC が改善されていることがわかる．一方で，誤差範囲が上に伸びているので，一部のベンチマークではエントリ数が増えるほど IPC が低下すると判断できる．

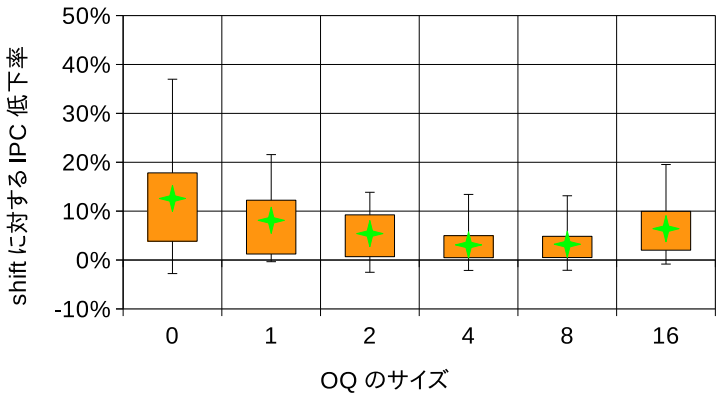

図 7.2: 128 エントリの IQ において OQ のサイズを変えた場合の IPC 低下率

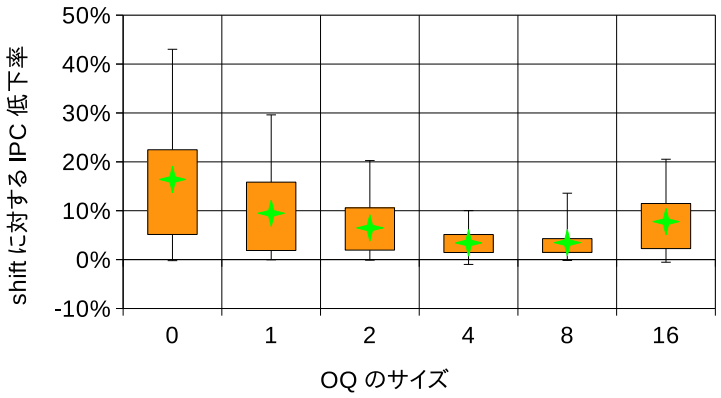

図 7.3: 192 エントリの IQ において OQ のサイズを変えた場合の IPC 低下率

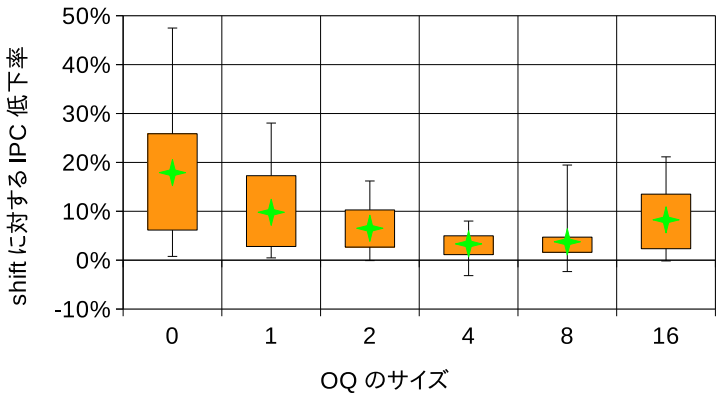

図 7.4: 256 エントリの IQ において OQ のサイズを変えた場合の IPC 低下率

7.2 プリフェッチャを変えた場合の評価

NORMAL セットにおいて，shift，circular，random，age，RRQ の各モデルでプリフェッチしない場合，ストリームプリフェッチを用いる場合，ストライドプリフェッチを用いる場合に分けて IPC を測定した．それぞれの測定結果を図 7.5，図 7.6，図 7.7 に示す．横軸は各モデル，縦軸は shift に対する IPC 低下率を示している．RRQ の OQ のサイズは 4 とした．各プリフェッチの設定を表 7.3 に示す．表の各列は左から順に，1 回あたりのプリフェッチ要求数，プリフェッチするデータアドレスまでの距離，プリフェッチ用の表のエントリ数，その連想度，学習ウィンドウの大きさ，学習期間の閾値を表す．

	degree	distance	entry	way	win_size	threshold
stream	4	32	32	-	16	2
stride	16	1	1024	4	-	-

表 7.3: プリフェッチャの設定

プリフェッチャを導入することによって circular モデルの IPC 低下率が大幅に減少し，ストライドプリフェッチャの場合は最大で 44.4%から，22.5%にまで減少した．一方で，ランダムキューを使う他のモデルでは IPC 低下率はむしろ増加した．これは命令の発行優先度がランダムになっているため，学習がうまくいかないためであると考えられる．

これについて説明する．メモリ命令がプログラム順に従って発行されるならば，これらの命令が指し示すメモリアドレスも順番にアクセスされる．そのため，ストライドプリフェッチの場合，これらのアドレスの間隔が一定ならばストライドの検知が可能になり，プリフェッチを行うことができる．しかし，命令がランダムな順番で発行されると，メモリ命令によってアクセスされるアドレスの順番がランダムになり，ストライドを検知できなってしまう．同様のことがストリームプリフェッチにもあてはまり，学習に長い時間を要したり，誤った学習をする可能性がある．これらの問題は age モデルや RRQ モデルではある程度解決されるため，これらのモデルの IPC 低下率の増加量は random モデルよりも小さくなっている．

7.3 機能ユニットを変えた場合の評価

NORMAL セットだけではなく，LARGE セットと VERY_LARGE セットの 2 つの機能ユニットについても，RRQ モデルの OQ のサイズを変化させて IPC を測定した．それぞれの結果を図 7.8 と図 7.9 に示す．横軸は各モデル，縦軸は shift モデルに対する IPC 低下率である．

機能ユニットが多くなると，レディな命令の大部分を発行できるようになるので，OQ の存在しないランダムキューの IPC 低下率は減少する．また，同様のことが OQ についてもあてはまる．OQ に多くの命令を移動させても，OQ 内の古い命令が発行されないという現象が起こる可能性が小さくなるため，最適な OQ のサイズは大きくなる．図より，LARGE モデルと VERY_LARGE モデルの最適な OQ のサイズはそれぞれ 8 と 16 であると分かる．

R

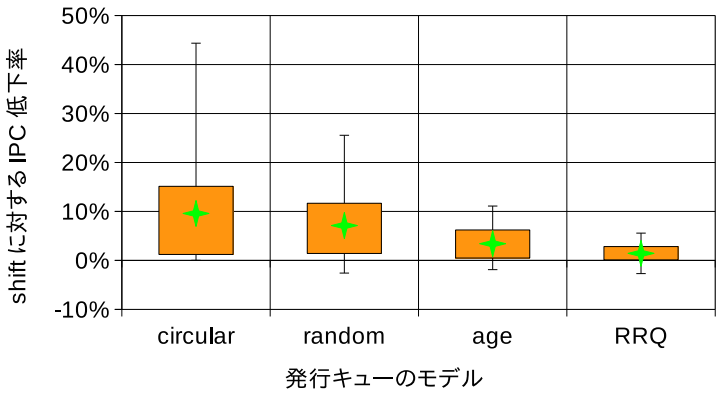

図 7.5: プリフェッチしない場合の IPC 低下率

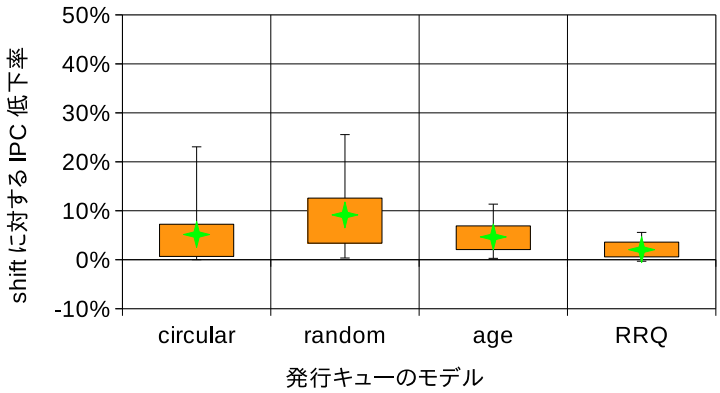

図 7.6: ストリームプリフェッチを使用した場合の IPC 低下率

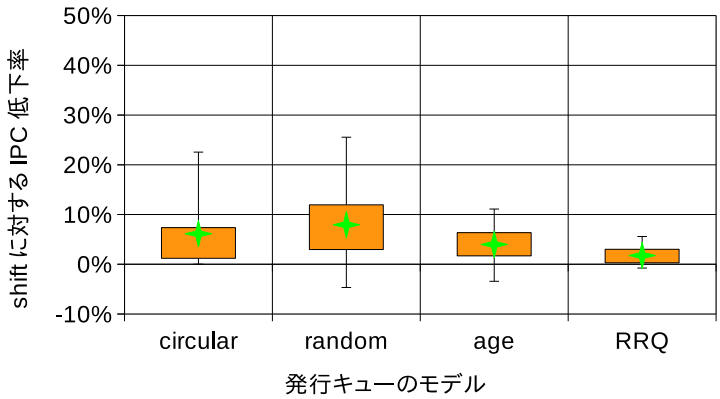

図 7.7: ストライドプリフェッチを使用した場合の IPC 低下率

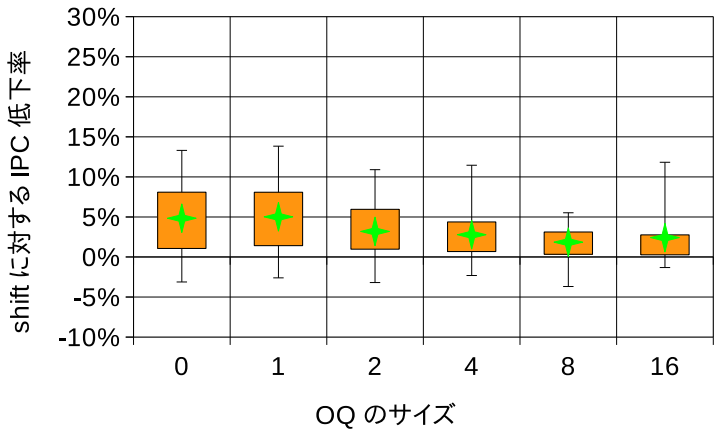

図 7.8: LARGE セットにおいて OQ のサイズを変えた場合の IPC 低下率

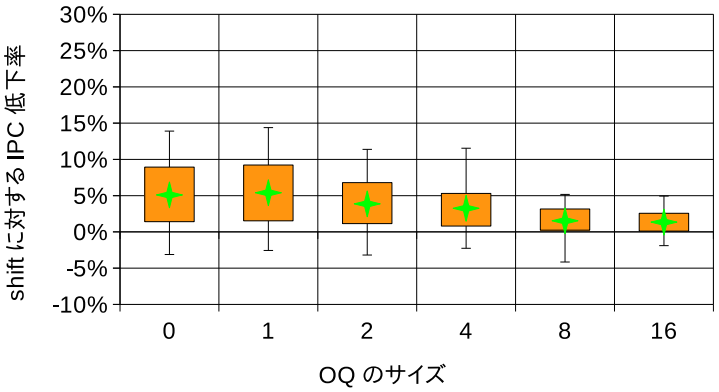

図 7.9: VERY_LARGE セットにおいて OQ のサイズを変えた場合の IPC 低下率

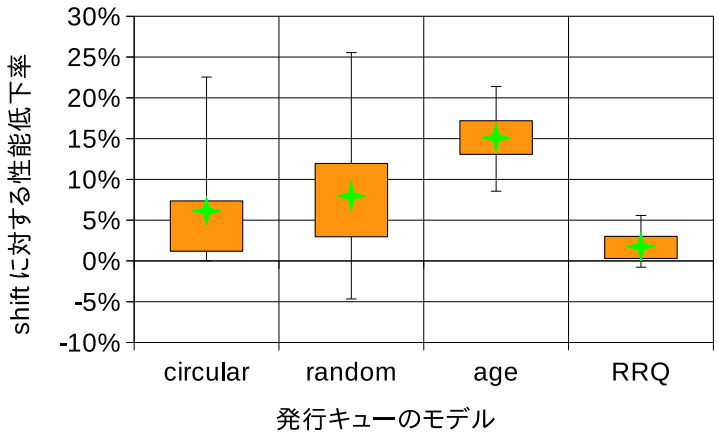

図 7.10: shift モデルに対する性能低下率

7.4 遅延と IPC を考慮した性能評価

IQ の遅延がサイクル時間を決定しているとし，その結果を反映させた各モデルの性能を図 7.10 に示す．

横軸は各モデル，縦軸は shift モデルに対する性能低下率を表す．サイクル時間が変化するのは，エイジ論理を用いている age モデルのみなので，それ以外のモデルの性能低下率は図 7.7 で示した IPC 低下率と一致する．

age モデルでは IPC が 4.3%増加したが，遅延は 13%増加したので，性能低下率の幾何平均値は 15.1%であった．random モデルと比較すると，性能低下率の最大値は 25.5%から 21.4%まで改善しているものの，幾何平均値は 7.9%から 15.1%まで 7.2%増加し，最小値は-4.7%から 8.5%まで 15.2%増加した．以上の結果からエイジ論理は，ランダムキューの IPC を向上させるものの，サイクル時間がそれ以上に増加するので，性能を低下させることがわかった．

逆に RRQ モデルではサイクル時間の増加がないので，IPC の向上はそのまま性能向上につながる．性能低下率の最小値は増加したものの，幾何平均値は 7.9%から 1.8%まで 6.1%低下させ，最大値は 25.5%から 5.6%まで改善することができた．

第 8章　まとめ

発行キューはプロセッサの構成要素の中でも性能に大きな影響を与えるものの 1 つである．発行キューの実装方法の 1 つとして，プログラム順に命令を格納しないランダムキューがある．これは空きエントリに命令を挿入していくので，命令の並びはランダムとなる．キューの容量効率は良いが，古い命令に高い優先度を与えることができず，プロセッサの性能が低下する．ランダムキューによる性能低下を解決する既存手法としてエイジ論理があるが，これは発行論理の遅延を増加させる．

遅延を増加させることなくランダムキューの性能を引き上げるために，ランダムキューをメインキュー（MQ:main queue）とオールドキュー（OQ:old queue）と呼ぶ 2 つのキューに分割して発行を行う手法を提案した．フロントエンドからは MQ にのみ命令を挿入するが，OQ へは MQ の最も古い数命令を移動させる．OQ の命令を MQ の命令よりも優先して発行することで，データフローのクリティカルパス上の命令に高い優先度を与え，プロセッサの性能低下を抑制する．

SPICE を用いて回路シミュレーションを行った結果，エイジ論理は発行論理の遅延を 13.1%増加させた．SPEC CPU 2006 を用い評価した結果，エイジ論理を用いてもランダムキューの IPC は 3.9%しか増加しなかった．一方で提案手法の RRQ では発行論理の遅延を増加させることなく，IPC が 6.2%向上した．その結果，クロックサイクル時間が発行論理の遅延で決定されるとした場合，エイジ論理はランダムキューの性能を 7.2%低下させ，RRQ は 6.2%向上させた．

付 録 A　プロセステクノロジー

6.3 節において，トランジスタのプロセスは 16nm を想定していると述べた．正確には 16nm のプレナー型トランジスタであり，現在の主流のトランジスタとは形状が異なる．この付録 A では，それらの特徴や回路設計の違いについて述べる．

今現在，トランジスタは大きく 2 種類に分けられる．平面的な形をした従来通りのプレナー型トランジスタと，立体的な形をした 3 次元構造のトランジスタである．後者のトランジスタにはいくつかの種類が存在し，その中でも FinFET [22] が特に有名である．プレナー型トランジスタと FinFET のおおまかな形状を，それぞれ図 A.1 と図 A.2 に示す．

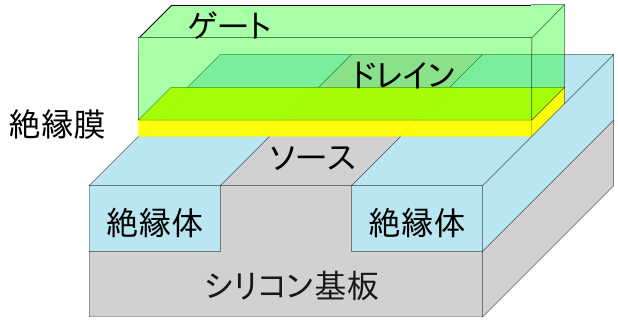

図 A.1: プレナー型トランジスタの概略

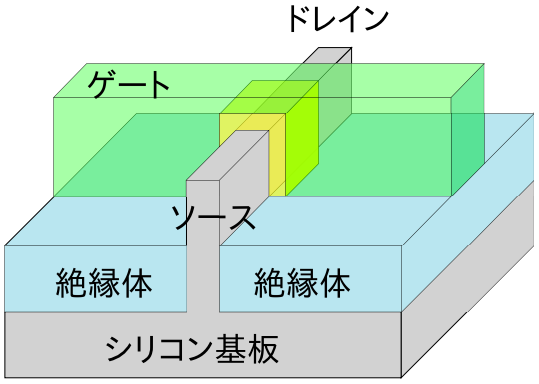

図 A.2: FinFET の概略

プレナー型トランジスタでは絶縁体とソース，ドレインが同じ高さにあるのに対して，FinFET ではソースとドレインが絶縁体から突き出ており，それを絶縁膜とゲートが覆っている．この図 A.2 のように，基板から突き出た，ソースとドレインをつなぐ電流の細い通り道をフィンと呼ぶ．FinFET はプレナー型トランジスタよりもリーク電流が小さいので，Intel の Haswell [23] をはじめ，現在のプロセッサではプレナー型トランジスタではなく FinFET が採用されている．

両者のレイアウト設計の違いについて述べる．それぞれの場合における単独のトランジスタのレイアウト例を図 A.3 と図 A.4 に示す．FinFET において電流の通り道であるフィンの数を増やすことは，プレナー型トランジスタにおいてゲート幅を増やすことと等価である．一方，プレナー型トランジスタではゲート幅を連続的に変更できるのに対して，FinFET では固定長のフィンの個数を変更するので，トランジスタの幅を離散的にしか変更できないという違いがある．

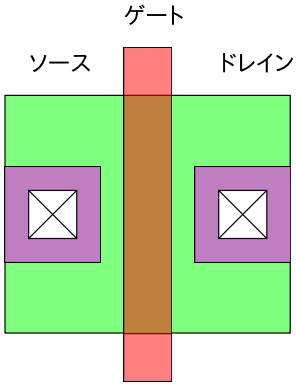

図 A.3: プレナー型トランジスタにおけるレイアウト例

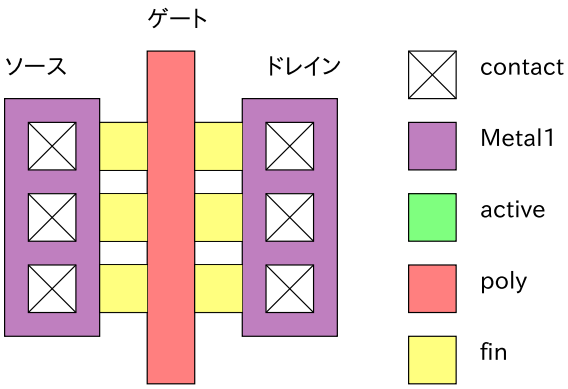

図 A.4: FinFET におけるレイアウト例

以上のように，FinFET のレイアウト設計ルールはプレナー型トランジスタの場合とは異なる部分がある．そして FinFET の画一的なレイアウト設計ルールが公開されていない．PTM [15] には，プレナー型の 16nm プロセスのトランジスタモデルだけではなく，より小さなプロセスの FinFET モデルも公開されているが，上記の理由で FinFET のレイアウト設計ができない．そのため，本研究では従来通りのプレナー型トランジスタの 16nm プロセスを用いて，MOSIS デザインデール [14] に則って回路設計を行った．

発表実績

- 酒井信二，塩谷亮太，安藤秀樹，“ランダムバッファの発行キューにより生じる性能低下の抑制，’’　情報処理学会研究報告，Vol.2015-ARC-216，No.16，2015 年 8 月

受賞歴

- 情報処理学会　システム・アーキテクチャ研究会　若手奨励賞
- 情報処理学会　2016 年度山下記念研究賞

謝辞

本研究をすすめるにあたり, 多大なる御指導と御鞭撻を賜わりました名古屋大学大学院工学研究科電子情報システム専攻 安藤秀樹教授, 塩谷亮太准教授に心より感謝いたします. 本研究の遂行を支えてくださいました, 名古屋大学大学院工学研究科電子情報システム専攻安藤研究室の諸氏に深く感謝します.

本研究の一部は，日本学術振興会 科学研究費補助金基盤研究 (C)（課題番号 25330057 及び 1K00070），及び科学研究費補助金 若手研究 (A)（課題番号 24680005）による補助のもとで行われたものである．

また，本研究は東京大学大規模集積システム設計教育研究センターを通し，シノプシス株式会社の協力で行われたものである．

参 考 文 献

[1] S. Palacharla, N. P. Jouppi, and J. E. Smith, Complexity-Effective Superscalar Processors, In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pp. 206–218, June 1997.

[2] M. Goshima, *Reserch on High-speed Instruction Scheduling Logic for Out-of-Order ILP Processors*, PhD thesis, Kyoto University, March 2004.

[3] K. Yamaguchi, Y. Kora, and H. Ando, Evaluation of Issue Queue Delay: Banking Tag RAM and Identifying Correct Critical Path, In *Proceedings of the 29th International Conference on Computer Design*, pp. 313–319, October 2011.

[4] J. A. Farrell and T. C. Fischer, Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor, *Journal of Solid-State Circuits*, Vol. 33, No. 5, pp. 707–712, May 1998.

[5] R. P. Preston, R. W. Badeau, D. W. Bailey, S. L. Bell, L. L. Biro, W. J. Bowhill, D. E. Dever, S. Felix, R. Gammack, V. Germini, M. K. Gowan, P. Gronowski, D. B. Jackson, S. Mehta, S. V. Morton, J. D. Pickholtz, M. H. Reilly, M. J. Smith, Design of an 8-wide Superscalar RISC Microprocessor with Simultaneous Multithreading, *2002 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 334–472, February 2002.

[6] J. Vinh M. Golden, S. Arekapudi, 40-entry Unified Out-of-Order Scheduler and Integer Execution Unit for the AMD Bulldozer x86-64 Core, *2011 IEEE International Solid-*

State Circuits Conference, Digest of Technical Papers, pp. 80–82, February 2011.

[7] R. J. Eickemeyer H. Q. Le J. Leenstra D. Q. Nguyen B. Konigsburg K. Ward M. D. Brown J. E. Moreira D. Levitan S. Tung D. Hrusecky J. W. Bishop M. Gschwind M. Boersma M. Kroener M. Kaltenbacha T. Karkhanis K. M. Fernsler B. Sinharoy, J. A. Van Norstrand, IBM Power8 Processor Core Microarchitecture, *IBM Journal of Research and Development*, pp. 2:1 – 2:21, January - February 2015.

[8] D. S. Henry, B. C. Kuszmaul, G. H. Loh, and R. Sami, Circuits for Wide-Window Superscalar Processors, In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 236–247, June 2000.

[9] J. Stark, M. D. Brown, and Y. N. Patt, On Pipelining Dynamic Instruction Scheduling Logic, In *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, pp. 57–66, December 2000.

[10] M. Goshima, K. Nishino, T. Kitamura, Y. Nakashima, S. Tomita, and S. Mori, A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors, In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp. 225–236, December 2001.

[11] M. D. Brown, J. Stark, and Y. N. Patt, Select-Free Instruction Scheduling Logic, In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp. 204–213, December 2001.

[12] P. Michaud and A. Seznec, Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors, In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pp. 27–36, January 2001.

[13] P. G. Sassone, J. Rupley II, E. Brekelbaum, G. H. Loh, and B. Black, Matrix Scheduler Reloaded, In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pp. 335–346, June 2007.

[14] https://www.mosis.com/.

[15] http://ptm.asu.edu/.

[16] International Technology Roadmap for Semiconductors, http://www.itrs2.net/.

[17] 小林誠弥, 塩谷亮太, 安藤秀樹, タグの 2 段階比較による発行キューの消費エネルギー削減, 2013 年先進的計算基盤システムシンポジウム, pp. 2–9, 2014 年 5 月.

[18] http://www.simplescalar.com/.

[19] G. Hamerly, E. Perelman, J. Lau, and B. Calder, SimPoint 3.0: Faster and More Flexible Program Analysis, *Journal of Instruction-Level Parallelism*, Vol. 7, pp. 1–28, September 2005.

[20] L. Gwennap, ARM Optimizes Cortex-a72 for Phones, *Microprocessor Report*, May 2015.

[21] L. Gwennap, Skylake Speedshifts to Nest Gear, *Microprocessor Report*, September 2015.

[22] T. R. Halfhill, Intel Sprouts Fins at 22nm, *Microprocessor Report*, May 2011.

[23] M. Demler, Intel's Haswell Boosts Graphics, *Microprocessor Report*, June 2013.