卒　業　論　文

題　　目

パイプライン型CGRAにおける電力効率改善手法

年　　度

平成28年度

所　　属

慶應義塾大学
理工学部　情報工学科

指導教員

天野 英晴 教授

氏　　名

61308121
小島　拓也

卒 業 論 文 要 旨

学 科	情報工学	学 籍 番 号	61308121	フリガナ 氏 名	コジマ タクヤ 小島 拓也

(論 文 題 名)

パイプライン型 CGRA における電力効率改善手法

(内 容 の 要 旨)

IoT 時代においてウェアラブルデバイスなどのバッテリー駆動の高機能で小型なデバイスが求められている。こうしたデバイスは一定以上の性能を低消費電力で実現することが要求される。そのためエネルギー効率の高い粗粒度再構成可能アーキテクチャ(CGRA: Coarse-Grained Reconfigurable Architecture) が注目されている。CGRA は PE(Processing Element) と呼ばれる小さな演算処理部をアレイ状に多数持っていて、PE の構成をチップ製造後に変えることで高い柔軟性を持っている。これまで CGRA の一種である CMA(Cool Mega-Array) アーキテクチャが提案されている。CMA は動的な再構成を行わないため PE アレイへのクロック分配、中間データのレジスタを廃し高い電力効率を実現している。さらに CMA の 1 種である VPCMA(Valiable Pipelined CMA) と呼ばれる可変パイプラインの CMA が提案されている。VPCMA ではパイプライン分割のための小さなオーバーヘッドに対して高い性能向上とエネルギー効率が報告されている。しかし、これらの評価はパイプライン段数を変化させるだけに留まり、ボディバイアス制御によるリーク電力の制御を行っていなかった。

そこで本研究では VPCMA に対してパイプライン段数とボディバイアス制御を同時に行うことでさらなる電力効率を改善する手法を検討した。また、この手法では PE アレイに対して一律のボディバイアス電圧を与えるのではなく、さらに小さな粒度で行単位でボディバイアス電圧を制御することによる効果を検討した。まず初めに実行するアプリケーションと要求性能に対して電力を最小化するパイプライン段数とボディバイアス電圧を決定する手法を提案した。4 つの 24bit 画像処理アプリケーション (af,sf,sepia,gray) の実行をシミュレーションし評価を行った。この手法を適用した結果、電力を最小化するパイプライン段数は PE アレイで使用している行の数と同数であることがわかり、アプリケーションのマッピングに依存することがわかった。また PE アレイに対して行毎のボディバイアス制御を行うと、ゼロバイアス時と比べて平均約 46%の電力削減率を得た。一方で PE アレイ全体で一律のボディバイアス制御をする場合と比べると電力削減率は平均約 0.80%であった。

(内容の要旨は約25行程度で記入のこと)

慶 應 義 塾 大 学 理 工 学 部

目 次

図 目 次

表 目 次

第1章

序論

1.1 本研究の背景

ウェアラブルデバイスを始めとする高機能で小型なデバイスが求められている。これらのデバイスはバッテリーを電力源としていることが多く、一定以上の性能を低消費電力で実現することが要求されている。この要求を満たすためにエネルギー効率の高いアクセラレータを利用することが検討されている。特にそのエネルギー効率の高さから粗粒度再構成可能アーキテクチャ(CGRA: Coarse-Grained Reconfigurable Architecture) の一種である動的再構成可能プロセッサ (DRP: Dynamic Reconfiguable Processor) が注目されている。この DRP は PE(Processing Element) と呼ばれる小さな演算処理部分をアレイ状に多数持っており、PE 間の相互接続や演算の種類などをクロックに同期して切り替えることで動的に構成を変化させることができる。汎用的なプロセッサとは異なり命令のフェッチなどが不要であるため低い周波数、小さい電力で高い性能を得ることができる。

一方で DRP は PE アレイへのクロック分配や中間データを保持するレジスタ、動的な再構成による電力が大きな割合を占めている問題があった。我々の研究室で開発した動的再構成可能プロセッサ MuCCRA-3 において動的再構成が全体の 25%、クロックツリーでは全体の 15%の消費電力を占めていた。[1]

これらの電力を削減するためこれまでに高性能で低電力なアクセラレータのためのアーキテクチャとして CMA(Cool Mega-Array) が提案されている。[2]CMA は PE アレイからクロックツリーを取り除き、動的な再構成を行わないことで消費電力の削減を図っている。したがって PE アレイにおける柔軟性が低下してしまうが、マイクロコントローラによるメモリ入出力を工夫することで対応することができる。さらに CMA を低電力化するために Fully Depleted Sicilon On Insulator(FD-SOI) の一種である Silicon On Thin Buried oxide(SOTB) が利用されている。これまでに CMA-SOTB[3]、CMA-SOTB2[4]、CMA-CUBE-SOTB(CC-SOTB)[5] が開発された。SOTB プロセスの特徴はボディバイアス電圧を印加することで遅延とリーク電力のトレードオフさせることができる点であり、CMA の性能と電力の最適化を行うことが可能となった。

CMA-SOTB2 以前の CMA では PE アレイにデータを入力すると演算結果が出力されるまで新たにデータを入力することができなかった。しかし、演算の終盤においては序盤に使用された PE は使用されないことがほとんどであった。そこで CC-SOTB では複数の入力データに対して演算をオーバーラップして実行可能にするためのウェーブパイプラインモードが実装された。ところが、ウェーブパイプラインモードでは前段のデータが後段の

データと衝突する可能性があった。また、PE_ARRAY が大きな組み合わせ回路で構成されているためグリッチによる消費電力が問題となっていた。こうした課題に対処するために、安全に動作可能なパイプラインの実装とその評価が行われた。パイプラインの段数は可変的でありアプリケーションに応じてパイプライン段数を変化させることが可能な可変パイプラインである。CC-SOTB と比べ平均 77%の性能向上と最大で 1461MOPS[1]/mW という非常に高いエネルギー効率を達成できることが報告されている。[6] しかし、これらの評価には SOTB プロセス特有の利点であるボディバイアス制御による電力削減を行っていなかった。そこで本研究では提案された可変パイプラインの CMA(VPCMA: Variable Pipelined CMA) に対してパイプライン段数とボディバイアス制御を同時に行うことでさらなる電力効率の改善を検討する。

1.2 研究目的

本研究の目的は低電力アクセラレータ VPCMA において、電力を最小化するパイプラインの段数の変化とボディバイアス電圧の制御を同時に行うことによる電力削減の効果を明らかにすることである。さらにパイプライン段数とボディバイアス電圧を決定する手法を確立し、類似するアーキテクチャへ適用可能かを検討する。

1.3 本論文の構成

2 章では CMOS プロセスを利用したディジタル回路における消費電力の見積もり方法と SOTB 技術の概要を説明する。3 章では本研究で検討するパイプライン段数選択とボディバイアス電圧制御が適用できるパイプライン型の CGRA を紹介する。4 章では本研究で検討する手法を適用するアーキテクチャの VPCMA の概要を説明する。5 章では本研究の電力最小化の手法に関して説明する。6 章では本研究の予備評価と電力最小化の評価を行う。7 では本論文の結論を述べる。

[1]Million Operation Per Second

第2章

CMOS VLSIにおける電力とSOTB技術

2.1 消費電力

半導体技術の向上により集積度の高い集積化路が広く用いられるようになった。集積度の高い集積化路を VLSI(Very Large Scale Integration) と呼ぶ。VLSI では何十万以上ものトランジスタが 1 チップ上に搭載されている。集積度を向上させることにより高い性能を実現する一方で、その主な問題は消費電力の増加であった。そのような背景から現在では nMOS トランジスタと pMOS トランジスタを相補的に組み合わせて構成する CMOS プロセスが広く使われている。一般に CMOS を用いた VLSI の電力消費は 2 つの要素に分けることができる。

- ダイナミック消費電力 $P_{dynamic}$
- スタティック消費電力 P_{static}

つまり、全体の消費電力 P_{total} との関係は

$$P_{total} = P_{dynamic} + P_{static} \tag{2.1}$$

である。

2.1.1 ダイナミック電力

ダイナミック電力の要因は以下の 2 種類に分類できる。

- ゲートのスイッチングにより発生する負荷容量への充放電
- pMOS と nMOS が同時にオンとなることにより発生する貫通電流

ただし大部分のダイナミック電力は前者のスイッチング電力である。このスイッチング電力を $P_{switching}$ とすると以下の式で表される。[7]

$$P_{switching} = \alpha C V_{DD}^2 f \tag{2.2}$$

ここで α は活性化率と呼ばれ回路中で 0 から 1 へ遷移する確率、C は負荷容量、V_{DD} は電源電圧である。

2.1.2 グリッチ

直列に回路を接続する構造の場合、グリッチと呼ばれる信号変化により発生する電力を考慮する必要がある。グリッチによる信号変化は本来不要なものであるがビット間の遅延時間の差により発生してしまう。図 2.1 は CMA の PE において積算を行った時の波形である。左の赤い線が演算が開始した時刻であり、右の赤い線が演算の完了時刻を表している。最終的な値に確定するまでの間に不要なスイッチングが発生しているのが確認できる。組み合わせ回路の規模が大きくなり、複雑化すれほどグリッチの影響は大きくなる。このグリッチによる消費電力は全体の消費電力の 40%以上を占めることもあり、グリッチを取り除くことが消費電力削減に有効である。[8]

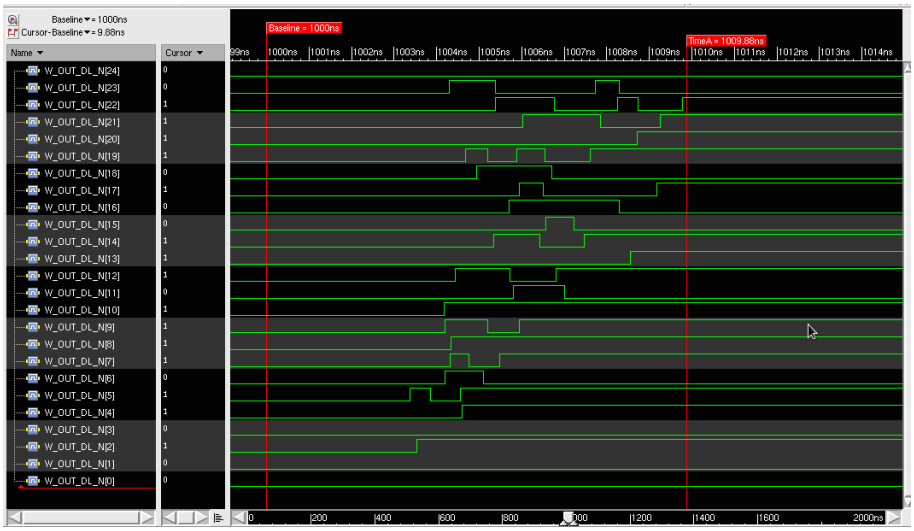


図 2.1: グリッチの様子

2.1.3 スタティック電力

スタティック電力はスイッチングしていないときにも消費される電力である。スイッチングしていないときに流れる電流をリーク電流と呼び。その原因はゲート絶縁膜をキャリアが通過することによるゲートリーク電流や、サブスレッショルドリーク電流など様々である。リーク電流を I_{leak} とするとスタティック電力 P_{static} は以下のように表される。

$$P_{static} = I_{leak} V_{DD} \tag{2.3}$$

2.2 SOTB 技術

2.2.1 SOTB の特徴

SOTB(Silicon on Thin Buried Oxide) は超低電力あるデバイス技術研究組合 (LEAP) によっ

て開発された SOI[1] の一種である。10nm 程度の極薄の SOI 層と BOX 層[2] がウェル基盤の上に積層されている。標準的なバルク CMOS テクノロジでは微細化に伴いスレッショルド電圧のばらつきなどの特性のばらつきが問題であった。SOTB は特性ばらつきが小さくバルクと比べて半分程度のばらつきに抑制されている。[9] これによりスレッショルド電圧を下げることが可能となり、低電力で動作させることが可能となる。さらに、BOX 層の下のウェル基盤に電圧を印加することによりリーク電流を制御することが可能である。この印加する電圧をボディバイアス電圧と呼び、ボディバイアス電圧を制御することによりリーク電力を最適化することができる。

2.2.2 SOTB トランジスタの構造

図 2.2 に SOTB トランジスタの構造を示す。[10]

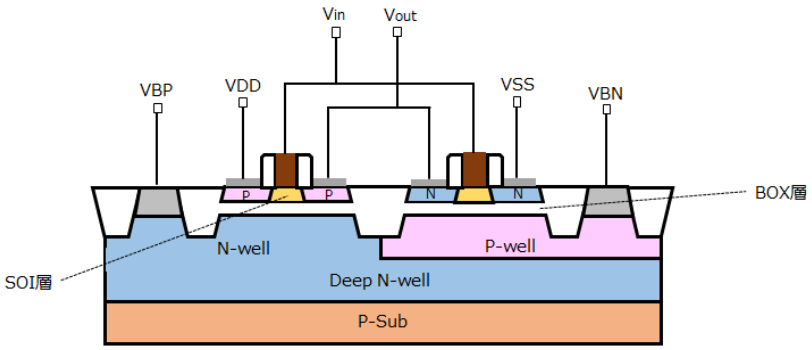


図 2.2: SOTB トランジスタの構造

2.2.3 ボディバイアス制御

ボディバイアス電圧のうち nMOS 側の電圧を V_{BN}、pMOS 側の電圧を V_{PN}、全体のボディバイアス電圧を V_{BB} と呼ぶこととする。nMOS、pMOS のバランスを揃える場合以下の関係が成り立つ。

$$VBN = VBB \tag{2.4}$$

$$VBP = VDD - VBB \tag{2.5}$$

V_{BB} の値が負のときをリバースボディバイアス、0 のときゼロバイアス、正のときをフォワードボディバイアスと呼び以下のトレードオフがある。

- リバースボディバイアス
  - リーク電流が減少

[1] SOI: Sillicon on Insulator
[2] BOX:Buried Oxide

  - 遅延時間が増加

- フォワードボディバイアス
  - リーク電流が増加
  - 遅延時間の減少

ボディバイアスの制御とは上記のような性能と電力のバランスを制御することである。

第3章

パイプライン型CGRA

本章では本研究の電力改善手法が適用できるパイプライン型の CGRA アーキテクチャを説明する。本論文でパイプライン型 CGRA と呼ぶアーキテクチャは以下のような特徴を持つアーキテクチャである。

- 単純なデータフローのみを扱う
- パイプライン的な動作が可能である

PE 間の相互接続のパターンを制限することにより、データフローが特定の方向に流れるように単純化されている。そのため PE アレイと外部にあるデータメモリとの間の入出力も単純化されている。このことによる入出力の柔軟性低下に対処するためにデータメモリと PE アレイとの間にクロスバのような機構が備わっていることが多い。明示的なパイプラインレジスタが必ずしも挿入されているわけではなく PE の内部に中間データを保持するレジスタが備えられていることもある。その場合でもデータフローが PE アレイの列方向となるように単純化されている場合、PE アレイの行をパイプラインステージとみなすことができる。マッピングされる演算によって各パイプラインステージには遅延のばらつきが発生する。パイプラインステージ単位でボディバイアス制御を行うことでこのばらつきを調整することができる。さらに、要求される性能が低い場合はパイプラインステージでの遅延が増加するような構成も許される。ゆえに、演算のマッピングや相互接続などの構成情報を変化させてパイプラインステージにおける遅延を調節することと、同時ににボディバイアス電圧を考慮することが可能である。本研究で対象とした VPCMA ではパイプラインステージの遅延を調節する手法としてパイプライン段数を変化させている。VPCMA 以外のアーキテクチャでパイプライン型 CGRA に分類されるアーキテクチャは PipeRench[11], S5 Engine[13],XPP[14]、DT-CGRA[15] が提案されている。これらのアーキテクチャの概要を説明する。

3.1 PipeRench

PipeRench アーキテクチャは PE を 16 個水平方向に並べて 1stripe と呼んでいる。16stripe のチップが実装されている。データは stripe に対して垂直に流れていくため stripe がパイプラインの 1 ステージと考えることができる。PipeRench は物理的には 16stripe しか存在しないが仮想的に 256stripe まで利用できるのが特徴である。演算が終了した stripe を次の

ステージの stripe として再構成することで仮想化を実現している。本手法を適用すれば使用する stripe の数と stripe 単位のボディバイアス制御によって電力を最小化することができる。

3.2 s5 Engine

S5 Engine はと呼ばれるアーキテクチャは主に 5 つの機能ブロックから構成されている。

1. 命令フェッチとデコードを行うユニット
2. ロード、ストアを行うユニット
3. 整数演算ユニット
4. 浮動小数点演算ユニット
5. 拡張ユニット
    - アレイ状の ALU
    - アレイ状の乗算器

上の 4 つは一般的なプロセッサと同様の動作をする。拡張ユニットはアレイ状の ALU と乗算器を持つ。拡張ユニットでの動作はチップ製造後にプログラマ及びコンパイラが指定可能である。上記ユニットは固定的な 5 段のパイプライン化がされているのに対して、拡張ユニットにおけるパイプラインの段数は可変的である。拡張ユニットの動作周波数はプログラム可能であり要求性能に応じてプログラマが指定することが可能である。拡張ユニットにおいてボディバイアス制御をできるようにすることで本手法を適用し電力を最小化するボディバイアス電圧とパイプライン段数を決定することができる。

3.3 XPP

マルチメディア処理、通信、グラフィックス処理において見られる高い並列度のストリーム処理に適したアーキテクチャである。構成要素は以下の 4 つに分けることができる。構成要素の一つである PAE とは Processing Array Element の略である。さらに ALU-PAE は 3 つに細分化できる。

1. ALU-PAE
    - ALU
    - FREG
    - BREG
2. RAM-PAE
3. IO-Element

4. Configuration manager

ALU-PAE に含まれる ALU では乗算や加算、比較、ソート、シフト演算などの DSP 処理で見られる典型的な演算を行う。FREG では垂直方向下向きのデータフローを管理する。さらにマルチプレクサやスワップなどのデータフローの制御用の演算が可能である。BREG では垂直方向上向きのデータフローを管理する。さらに、加算やバレルシフト、正規化を行うことが可能である。RAM-PAE は中間データを保持するための要素であり、構成情報を変化させることで FIFO として動作させることも可能である。IO-Element は PAE のアレイと外部の RAM とのやりとりを行う。Configuration Manager は各 PAE の構成を管理する。

XPP のアレイはデータ幅や各 PAE の個数などがパラメータ化されたソフトコアとして提供されている。そのため様々な設計が試されている。その一例は ALU-PAE が 64 個、RAM-PAE が 16 個、IO-Element が 4 個の設計がある。これは 8x8 の ALU-PAE のアレイの左右に RAM-PAE が 8 個ずつ配置されている。IO-Element はこのアレイの上側と下側に配置されている。つまり、XPP のアレイへのデータは上側または下側から入力され、上側または下側から出力する単純なデータフローとなる。各 PAE にはアレイ全体で同期を取るために入力部と出力部にレジスタがある。したがって、アレイの 1 行を 1 ステージとしたパイプラインとして動作していると考えることができる。

3.4 DT-CGRA

DT － CGRA アーキテクチャは画像処理や機械学習向けのアクセラレータとして提案されている。構成要素を以下に示す。

1. CA(Computing Array)
    - RC(Reconfigurable Cell)
    - PRC
    - IRC
2. SBU(Stream buffer units)
3. クロスバ

CA は複数列の RC と特別な計算を行う RC が 1 列で構成されている。特別な演算のできる RC は PRC と IRC があり、PRC では平方根や逆数の計算、IRC では補間法で用いられる計算を行うことができる。RC 間で転送するデータ量を減らすために MapReduce の手法が用いられている。各 RC には 5 つの PE を持っている。このうち 3 つを map 処理に、2 つを reduce 処理に用いる。ある処理を RC にマッピングするとき、map 処理で必要な PE の数が 3 つ以上の場合隣接した RC を用いることで対応することが可能である。一方で、map 処理に用いる PE の数が 3 つ未満の場合隣接する RC に使用させることが可能である。SBU では外部のデータメモリから転送された CA への入力データや CA からの出力データを保持する。クロスバは SBU と CA との間に配置されていて、データ転送の柔軟性を高める。

画像処理や機械学習などのアルゴリズムではデータの局所性が強く、異なるデータに対して同じ計算をすることが多い。このことに着目し DT-CGRA では CA の構成は静的に決定され、動的な再構成が行われない。一方で SBU から CA へ入力されるデータ数、入力する CA の行などは動的に再構成される。固定的な計算のマッピングに対してデータフローを動的に制御することで柔軟性を得ている。

演算処理部のマッピングを静的な構成にする点は VPCMA のコンセプトに近い。また、データ転送をクロスバを用いることで工夫している点は VPCMA において 4.2.2 節で説明するデータマニピュレータを使用している点に類似している。RC 間の相互接続は他方向に伸びており単純なデータフローとは呼べない。しかし、RC 内部が複数の PE で構成されること演算部の構成が静的であることから RC がパイプラインの 1 ステージとみなすことができる。同じアルゴリズムをマッピングするとしても map,reduce 処理の組をいくつの RC を用いるかによってパイプラインステージにおける遅延を調節することができる。RC 単位のボディバイアス制御を行えば本手法を適用し電力最小化をすることが可能である。

第4章

Valiable Pipelined CMAの概要

本章では、本論文で対象としているアーキテクチャである Valiable Pipelined CMA の概要を説明する。

4.1 CMA の概要

CMA は主にバッテリー駆動の組み込みシステムにおけるマルチメディア処理を行うアクセラレータとして提案された。[2] マルチメディア処理の特性として一定時間内に一定量のデータ処理を行うことが要求される。一方でこれらの処理を要求されている時間よりも早く完了させることによって得られる利益は少ない。したがって、処理時間を調整して不必要な消費電力を削減することが可能であり、バッテリー使用時間を長くすることにつながる。CMA アーキテクチャでは主に以下のコンセプトによって消費電力の削減を図っている。

- PE アレイからクロックツリーとレジスタを除去し、完全な組み合わせ回路として構成する
- 各 PE の演算やデータパスの動的な再構成を行わない

一般的な DRP では、動的な再構成のために各 PE においてレジスタを持ち、クロックが供給される。したがってクロックツリーと動的な再構成のための処理において大きな電力を消費する。一方で CMA では PE にはレジスタやクロックツリーは存在せず組み合わせ回路のみで構成している。このような構成を採用することにより上記のコンセプトを実現している。

初代試作機の CMA-1 から CCSOTB までは PE_ARRAY へのクロック供給とレジスタはなかった。そこで PE_ARRAY をパイプライン分割することによる発生するオーバーヘッドと性能の関係が研究され Valiable Pipelined CMA が提案された。[6] パイプライン分割のために PE_ARRAY に追加されたパイプラインレジスタにおける消費電力は全体の数%であり、クロックツリーでの消費電力は最大で全体の 60%を占めている。このようにパイプライン分割による電力増加がある、一方で高いスループット向上が得られ全体としてはエネルギー効率が改善されている。エネルギー効率は最大で 96%向上したと報告されている。

4.2 CMA の構成

CMA を構成する要素とそれらの動作を説明していく。CMA は主に以下の要素で構成されている。

- PE_ARRAY
- μ コントローラ
- コンフィギュレーションレジスタ
- 定数値レジスタ

4.2.1 PE_ARRAY

図 4.1 に PE_ARRAY の構成を示す。12 列 ×8 行の PE とそれらの相互接続、パイプライン分割のためのレジスタから構成されている。パイプラインレジスタは PE の行と行の間に置かれている。パイプラインの動作に関しては 4.3.2 節で詳しく説明する。すでに述べているように PE に対してはクロックの分配をしない、一方でパイプラインレジスタへはクロックが分配されている。ただし、可変パイプラインであるため使用しないパイプラインレジスタに対してはクロックゲーティングを行うことで不要な消費電力が発生しないようにしている。この PE_ARRAY 全体はおよそ 0.5V~1.2V の電源電圧で動作する。PE_ARRAY の入出力は南方向のみであり、入力には fetch レジスタが、出力には gather レジスタが接続されている。つまり、fetch レジスタ、gatcher レジスタともに PE_ARRAY の 0 行目の 0 番から 11 番までの列と接続されている。そのためデータ幅は 25bit×12 である。fetch レジスタにデータを書き込むことで PE_ARRAY での演算が開始し、適切なタイミングで gather レジスタにデータを取り込む。

PE の構成を図 4.2 に示す。各 PE は以下の要素から構成されている。

- ALU(Arithmetic Logic Unit)
- ALU への入力のセレクタ
- SE(Switch Element)

ALU はデータ幅 25bit の 2 入力 1 出力で演算を行う部分である。ALU の出力は SE への入力、隣接する北方向、北東方向、北西方向の PE へのダイレクトリンクへと接続される。ダイレクトリンクとは SE を経由せずに隣接する PE へデータを送るローカルな接続である。図 4.2 において破線で表される接続がダイレクトリンクである。ALU で使用可能な演算を表 4.1 に示す。ただし、入力される 2 つのデータを *indataA*, *indataB* としている。SE は東西南北方向の隣接する PE との相互接続を行うスイッチである。ALU への入力のセレクタは東西南方向からの入力、南方向、南東方向、南西方向からのダイレクトリンク、および定数値レジスタから演算に使用するデータを選択する。図 4.1 において PE 間の相互接続を見ると行をまたぐ接続ー 3 つのダイレクトリンク、北方向の出力と出力、南方向からの入力ーはパイプラインレジスタを経由することがわかる。ただし、南方向への出力、北方向からの入力はパイプラインレジスタを経由していない。

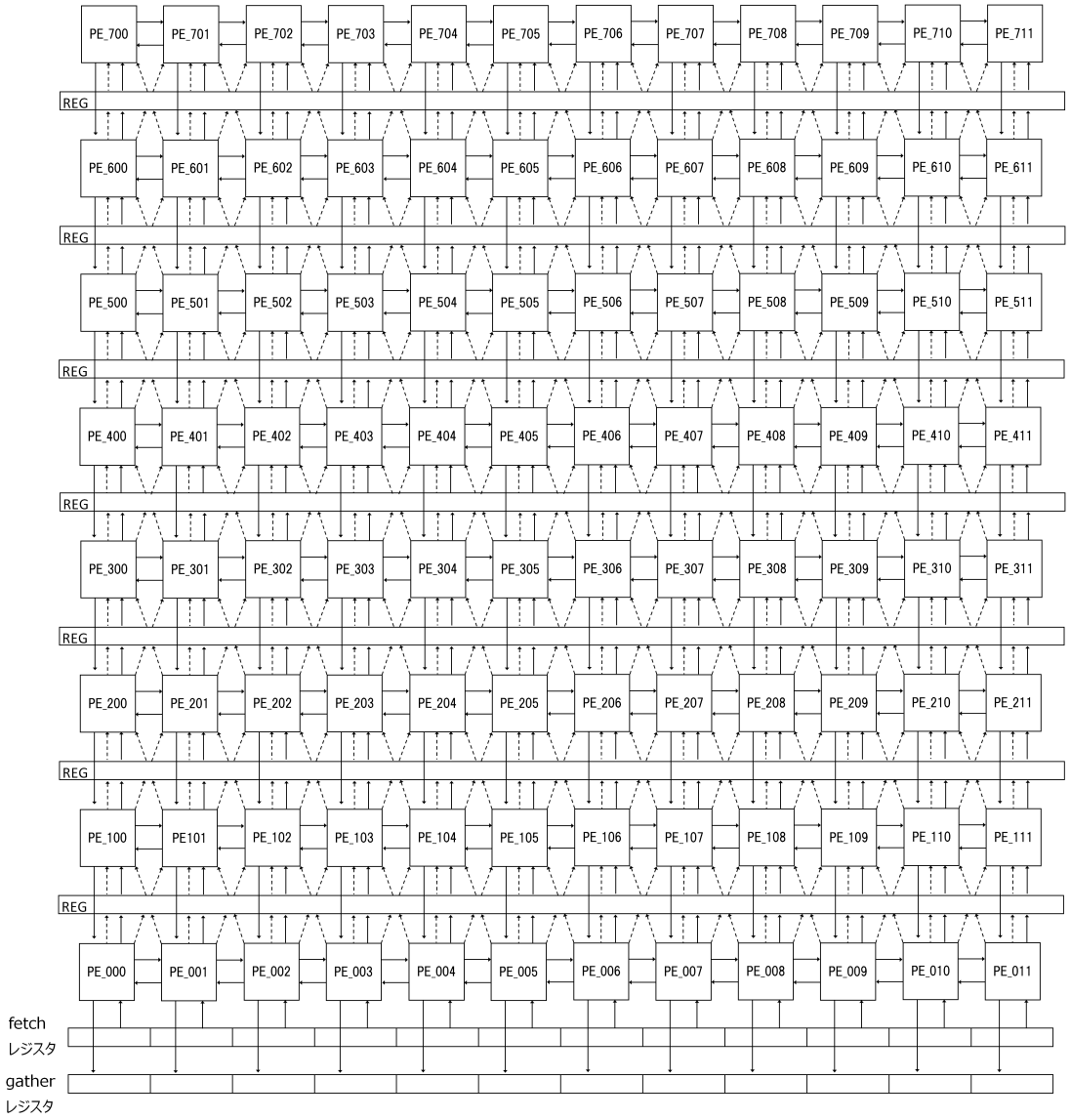
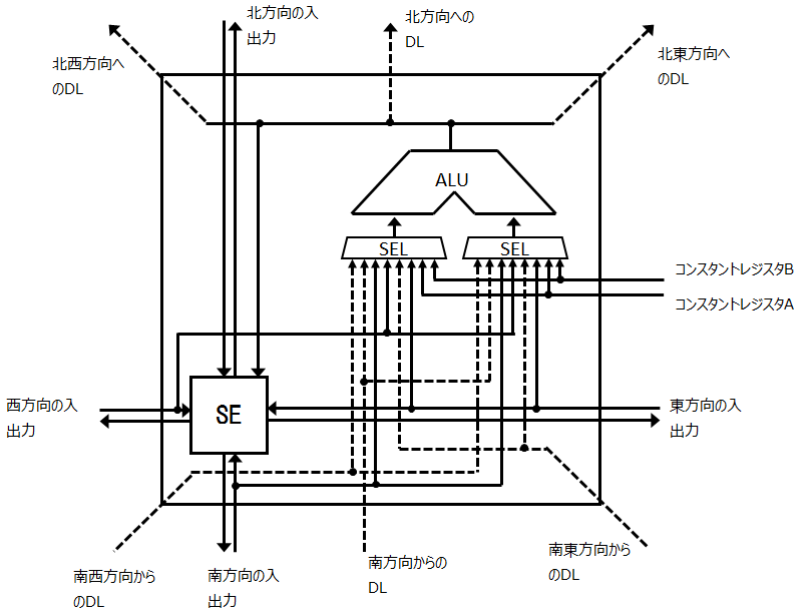


図 4.1: PE_ARRAY の構成



※ DL:ダイレクトリンク

図 4.2: PE の構成

4.2.2 μ コントローラ

μ コントローラは固定長 16bit の命令を実行する小さなプロセッサであり、PE_ARRAY とデータメモリの間のデータ転送などを行う。μ コントローラには表 4.2 に示す命令セットを持つ。μ コントローラは 25bit の汎用レジスタを 16 個持っている。表 4.2 中の rd,rs はレジスタを表す。n,imm,adr, X, X1, X2 は即値であり、左の () で囲まれた数値はビット数である。

データメモリのロードとストアの動作について説明する。μ コントローラにおけるデータメモリは 12 個のバンクに分割されている。そのため同時に 25bit のデータを最大で 12 個読み書きすることが可能である。これは PE_ARRAY が 12 列で構成されているためである。データメモリからロードしたデータを fetch レジスタのどの列へ書き込みを行うかの対応付けを行うのがデータマニピュレータである。同様に gather レジスタのどの位置からデータを読み込み、データメモリへストアするかの対応付けをデータマニピュレータが行う。

表 4.3 をデータマニピュレータのテーブルとして指定した時のロード時の様子を図 4.3 に示す。マスク部分はその列番号についてロードを行うか否かを表している。0 の列の fetch レジスタには書き込みをせず、1 の列の fetch レジスタに対して書き込みを行う。fetch レジスタへ書き込みを行う列はデータメモリのアドレスを指定する必要がある。LD テーブルは各列についてロード開始アドレスに対するオフセットを持っており、これによって読み込むデータのアドレスを指定している。この例ではロードするデータの数は 4 個であるため、1 回のロードが終わるとロード開始アドレスは 4 加算され次のロード処理に備える。

表 4.1: ALU で使用可能な演算の一覧

演算名	出力される演算結果
NOP	0
ADD	*indataA* + *indataB*
SUB	*indataA* - *indataB*
MULT	*indataA* × *indataB*
SL	*indataA* を *indataB* ビット左へシフト
SR	*indataA* を *indataB* ビット右へ論理シフト
SRA	*indataA* を *indataB* ビット右へ算術シフト
SEL	if (*indataA* の MSB==1) *indataA* else *indataB*
CAT	*indataA*
NOT	NOT *indataA*
AND	*indataA* AND *indataB*
OR	*indataA* OR *indataB*
XOR	*indataA* XOR *indataB*
EQL	if (*indataA*==*indataB*) *indataA* else *indataB*
GT	if (*indataA* > *indataB*) *indataA* else *indataB*
LT	if (*indataA* < *indataB*) *indataA* else *indataB*

この機構により PE_ARRAY とデータメモリのやり取りを柔軟に行うことが可能となる。

gather レジスタへ演算結果を取り込みデータメモリへのストアが行われるのは表 4.2 で見たようにデータメモリから fetch レジスタへデータをロードしてから指定したサイクル数遅延してから行われる。これは PE_ARRAY は大規模な組み合わせ回路であり計算結果が出力されるまでに長い時間が必要であり、μ コントローラの 1 クロックサイクル以内に演算を完了できない場合が殆どだからである。ストアの動作に関しても同様にデータマニピュレータの対応付けによって指定した列の出力のみをデータメモリへ書き込み、書き込んだデータ数の分だけストア開始アドレスが加算される。表 4.4 をデータマニピュレータのテーブルとして指定した時のストア時の様子を図 4.4 に示す。この例ではストアするデータの数は 2 個であるため、1 回のストアが終わるとストア開始アドレスは 2 加算される。

4.2.3 コンフィギュレーションデータ

コンフィギュレーションデータとは PE_ARRAY 内にある各 PE とパイプラインレジスタの構成情報を指定するものである。PE_ARRAY の外部にコンフィギュレーションレジスタが配置されており、そこから PE_ARRAY へ送信される。使用するパイプラインレジスタをコンフィギュレーションデータに含めることでアプリケーションごとに可変なパイプラ

表 4.2: μ コントローラの命令セット

オペコード	オペランド	命令の意味
NOP		
ADD	rd, rd	rd ←rd + rs
SUB	rd, rs	rd ←rd - rs
MV	rd, rs	rd ← rs
DONE		プログラムの終了を示す
DELAY	n(4)	G.R にデータを取り込むまでの遅延サイクル数 n を指定する
LDI	rd, imm(8)	rd ← imm
ADDI	rd, imm(8)	rd ← rd + imm
BEZ	rd, imm(8)	if (rd==0) pc ← pc + imm
BEZD	rd, imm(8)	rd ← rd - 1 if (rd==0) pc ← pc + imm
BNZ	rd, imm(8)	if (rd!=0) pc ← pc + imm
BNZD	rd, imm(8)	rd ← rd - 1 if (rd!=0) pc ← pc + imm
LD_SET	adr(8), X(4)	データメモリから F.R へのデータをロードする際の設定を行う adr: データメモリのロード開始アドレスを指定 X:データマニピュレータのテーブル番号を指定
ST_SET	adr(8) , X(4)	G.R からデータメモリへデータをストアする際の設定を行う adr: データメモリのストア開始アドレスを指定 X: データマニピュレータのテーブル番号を指定
LDST_ADD	X1(4), X2(4)	データメモリのロード開始アドレスから X1 個のデータを F.R レジスタへロードする ロード開始アドレスに X1 を加算する n サイクルの遅延の後 G.R からデータメモリのストア開始アドレスから X2 個のデータをストア ストア開始アドレスを X2 加算する

※ F.R: fetch レジスタ　G.R: gather レジスタ pc: プログラムカウンタ

表 4.3: LD テーブルの例

fetch レジスタの列番号	0	1	2	3	4	5	6	7	8	9	10	11
マスク部分	1	0	0	1	0	0	1	0	0	1	0	0
ロード開始アドレスに対するオフセット	0x0	0x0	0x0	0x1	0x0	0x0	0x2	0x0	0x0	0x3	0x0	0x0

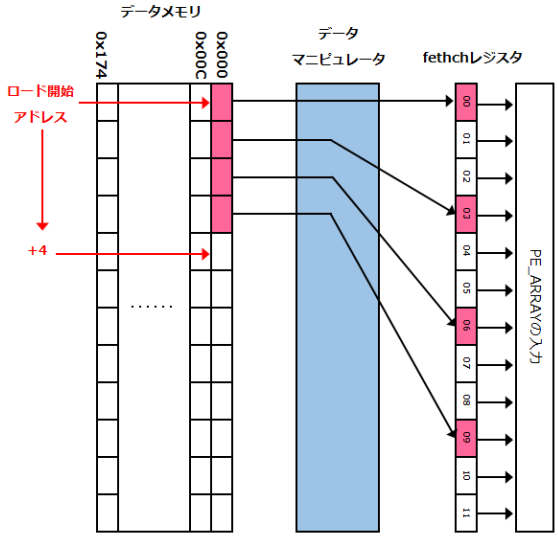


図 4.3: ロード時のデータマニピュレータ様子

イン段数を使用することが可能となる。すでに述べているようにアプリケーション実行を通してコンフィギュレーションデータは動的に変化することはない。

4.3　演算のマッピングとアプリケーション実行の様子

PE_ARRAY にアプリケーションをマッピングして実行する様子を説明する。前半ではパイプライン分割を行わない場合を、後半ではパイプライン分割を行った場合の動作を説明する。例として RGB24bit のグレースケールを行うアプリケーションを用いる。PE_ARRAY は 12 列あるがここでは簡略化のために PE_ARRAY のはじめの 2 列のみを示す。演算のマッピングされた PE とその相互接続の様子を図 4.5 に示す。各 PE にマッピングされた演算が記されている。何も書かれたいない 2 つの PE はスイッチとしてのみ利用されていて、ALU は使用されない。

次に μ コントローラにおけるアプリケーション実行の様子を説明する。比較のためにパイプライン分割しない場合と、パイプライン分割する場合で同じクロックサイクル、つまり同じ周波数で動作する例を考える。

表 4.4: ST テーブルの例

gather レジスタの列番号	0	1	2	3	4	5	6	7	8	9	10	11
マスク部分	1	0	0	0	0	0	1	0	0	0	0	0
ストア開始アドレスに対するオフセット	0x0	0x0	0x0	0x0	0x0	0x0	0x1	0x0	0x0	0x0	0x0	0x0

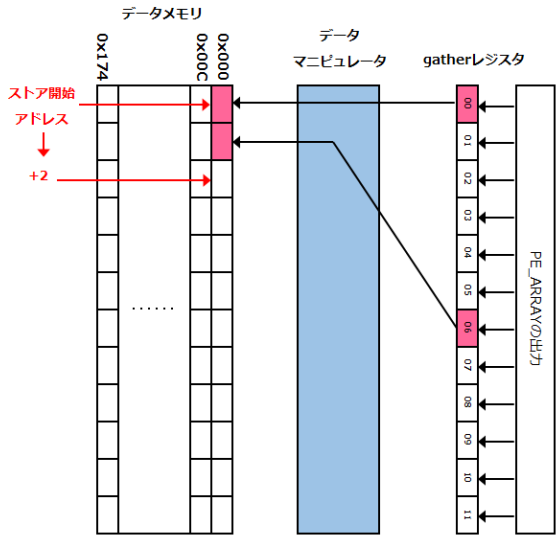


図 4.4: ストア時のデータマニピュレータの様子

4.3.1　パイプライン分割をしない場合

μ コントローラにおける命令コードを以下に示す。この命令コードを実行したときの様子を図 4.6 に示す。

パイプライン分割しない場合の命令コード

```
        DELAY 7
        LDI r3,#8
        SET_LD #0,#6
        SET_ST #0x30,#6
LP:     LDST_ADD #0,#0
        NOP
        NOP
        NOP
        NOP
        NOP
        BNZD r3,LP
        DONE
```

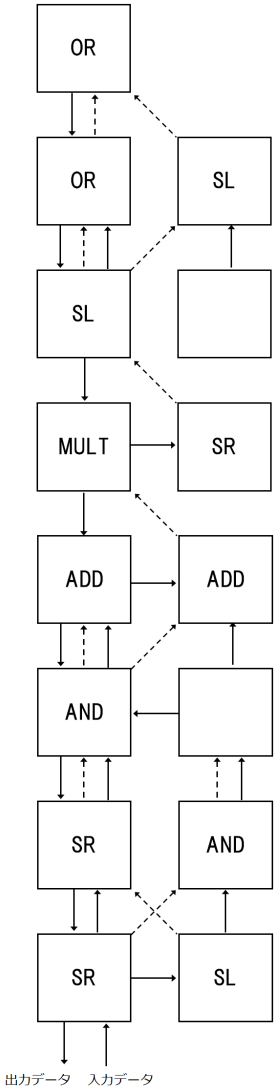


図 4.5: アプリケーションのマッピング例:

データメモリの 0x0 番地からデータを 6 個ロードして、データメモリの 0x30 番地にデータを 6 個ストアすることを 8 回ループすることを示している。演算終了まで 7 クロックサイクル遅延してからストアを行っている。演算結果を gather レジスタに取り込むまでは新しいデータを PE_ARRAY に入力して演算を開始させることができないのでそれまで NOP 命令を挿入している。

図 4.6: パイプライン分割しない場合の実行の様子

4.3.2　パイプライン分割をする場合

ここではすべてのパイプラインレジスタを用いる、すなわちパイプライン段数 8 の場合を考える。μ コントローラにおける命令コードを以下に示す。この命令コードを実行したときの様子を図 4.7 に示す。

パイプライン分割しない場合の命令コード

```
        DELAY 7
        LDI r3,#8
        SET_LD #0,#6
        SET_ST #0x30,#6
LP:     LDST_ADD #0,#0
        BNZD r3,LP
        DONE
```

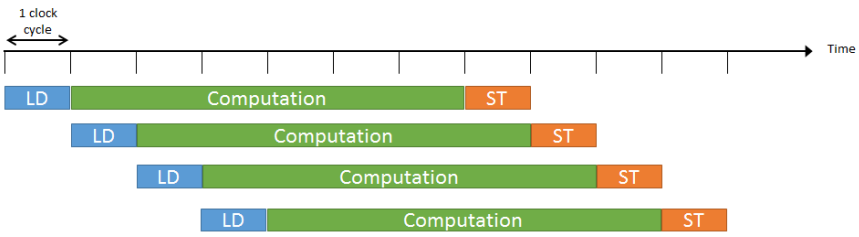


図 4.7: パイプライン分割する場合の実行の様子

パイプライン分割しない場合の図 4.6 と比べて各パイプラインステージで演算をオーバーラップさせて実行できることがわかる。

第5章

パイプライン段数とボディバイアス制御による電力効率改善手法

Valiable Pipelined CMA は CCSOTB と比べて高い性能と高い電力効率が得られると示された。[6] 一方で要求される性能が低い場合パイプライン分割のためのオーバーヘッドのため CCSOTB よりも高い電力となってしまうことが問題であった。この問題を解決するためにパイプライン分割とボディバイアス制御を同時を行うことを検討する。

5.1 ボディバイアス電圧とパイプライン段数のトレードオフ

ボディバイアス電圧を変化させると、遅延時間とリーク電力との間にトレードオフがあることは 2.2 節において述べた。つまり、動作周波数とリーク電力との間にトレードオフがある。パイプライン分割の段数を変化させることによるトレードオフについて説明する。パイプライン段数の増加により得られる利益は主に 3 つある。1 つ目は PE_ARRAY の長いデータパスを分割することで動作可能な周波数が高くなることである。2 つ目はパイプラインレジスタを経由することにより 2.1.2 節で述べたグリッチが次段の PE に伝搬するのを防ぐことが可能となることである。3 つ目はオーバーラップして演算を進めることが可能となりスループットが向上することである。一方パイプライン段数の増加により発生する負の影響はクロックツリー、パイプラインレジスタでの消費電力が大きくなることである。

以上のトレードオフを表 5.1 にまとめる。

5.2 電力効率改善手法

要求性能が与えられた時に、それを満たし電力を最小化するパイプライン段数とボディバイアス電圧を決定する手法を提案する。今回前提とする条件を以下に挙げる。

- ボディバイアス制御のドメインは PE_ARRAY の行単位とする
- パイプライン段数は 1 段,2 段,4 段, 全段の 4 パターン
- 探索はブルートフォース探索で行う

表 5.1: トレードオフのまとめ

ボディバイアス電圧		
	VBB > 0	VBB < 0
動作周波数	向上	低下
リーク電力	増加	減少
パイプライン段数		
	大	小
動作周波数	向上	低下
スループット	向上	低下
グリッチの影響	低下	向上
レジスタ、クロックツリーでの消費電力	増加	減少

- 要求性能はパイプライン分割しない時の周波数で与えられる

ボディバイアス制御のドメインとは同一のボディバイアス電圧を印加する領域のことである。つまり、PE_ARRAY において同一行の PE は同じボディバイアス電圧になるが、行ごとに異なるボディバイアスを印加することができる。パイプラインレジスタは 7 本あるので分割のパターンは 128 通りある。しかしながら、計算量の関係から上記の 4 つのパターンのみを検討することにした。全段は 8 段とは限らない。なぜならアプリケーションによって使用している行数が違うからである。例えばアプリケーションに sepia を選んだ場合、これは 6 行しか使用していないため全段とは 6 段を意味する。

5.2.1 要求性能の換算

CMA におけるパイプライン分割では汎用的なプロセッサのそれと異なり、ハザードが発生することによるスループットの低下が起こることはない。よって同一クロックサイクルであれば、n 段に分割するとスループットは n 倍となる。ゆえに、要求性能を換算するとすると 1 段パイプラインで要求される周波数が f である場合 n 段パイプラインでは f/n で動作することが要求されることになる。

5.2.2 対象のモデル化

2.1 に基づき、ダイナミック電力 $P_{dynamic}$ とスタティック電力 P_{static} のモデル式を以下のように定義する。

$$P_{dynamic} = \Big(E_{comb}(N_p) + (E_{reg} + E_{clk}) \times (N_p - 1)\Big) \times f \tag{5.1}$$

$$P_{static} = P_{leak,clk} + P_{leak,reg} + \sum_{i=0}^{7} P_{leak,row}(VBB_i) \tag{5.2}$$

ダイナミック電力のモデル式における各変数の内容を以下に示す。

- N_p: パイプライン段数
- $E_{comb}(N_p)$: 組み合わせ回路部分における 1MHz あたりの消費電力
- E_{reg}: パイプラインレジスタにおける 1MHz、１本あたりの消費電力
- E_{clk}: パイプラインレジスタへのクロックツリーにおける 1MHz,1 本あたりの消費電力
- f: 動作周波数

$E_{comb}(N_p)$ は全 PE での消費エネルギーを表している。段数 N_p に依存することを表している。なぜなら、5.1 節で述べたように段数に応じてグリッチの影響が変化し、発生するスイッチングが変化するからである。

スタティック電力のモデル式における各変数の内容を以下に示す。

- $P_{leak,clk}$: クロックツリーのリーク電力
- $P_{leak,reg}$: パイプラインレジスタのリーク電力
- VBB_i: PE_ARRAY の i 行目に対するボディバイアス電圧
- $P_{leak,row}(VBB_i)$: VBB_i 印加時の行全体のリーク電力

クロックツリー、パイプラインレジスタはボディバイアス制御を行わないため一定値となる。PE_ARRAY の行単位のリーク電力はボディバイアス電圧によって制御されるため、VBB_i に依存する。

動作周波数 f を決定するには PE_ARRAY 内の全データフローの中で最も大きい遅延時間 D_{max} を計算し、$f = 1/D_{max}$ とすれば良い。最大遅延時間 D_{max} の求め方の手順を以下に示す。

1. PE_ARRAY 中のデータパスを抽出
2. 各データパスの遅延時間を求める
3. 求めた遅延時間のなかで最大のものを決定する

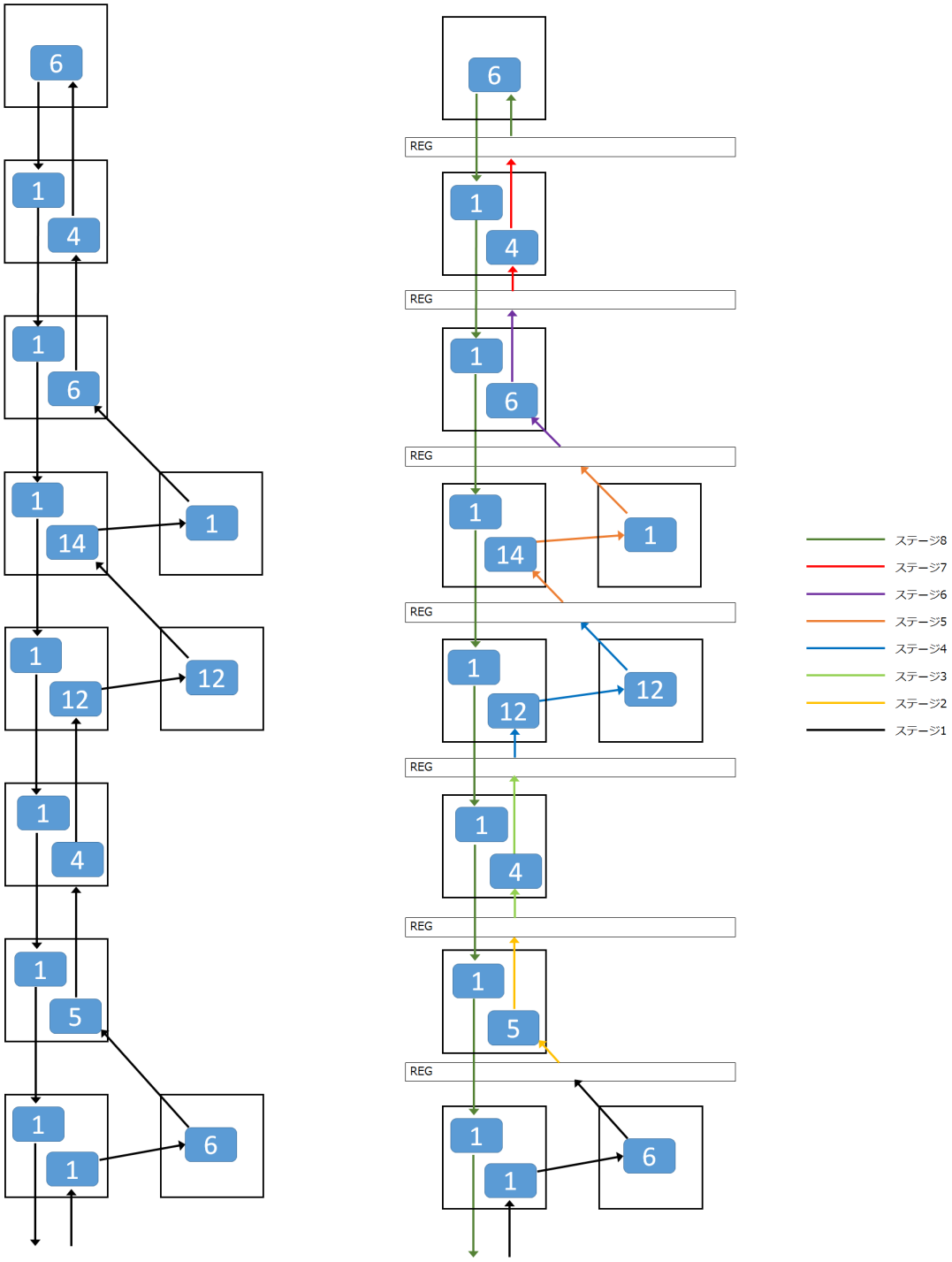
例えば、図 5.1 のようなデータパスと各 PE で図中に示される数値の遅延時間を持つとする。図中の矢印はデータの流れを示している。この矢印と矢印の間に書かれている数字がその PE で生じる遅延を示す。図 5.1(a) は 1 段パイプラインの例であり、(b) は 8 段パイプラインの例である。いずれもマッピングされた演算とデータパスは同一である。

パイプライン分割しない場合の遅延時間を D_1 とし、8 段パイプラインの n 番目のステージでの遅延を $D_8(n)$ とする。これらは以下のように計算される。

$$\begin{aligned} D_1 &= 1+6+5+4+12+12+14+1+6+4+6+1+1+1+1+1+1+1 \\ &= 78 \end{aligned} \tag{5.3}$$

$$D_8(1) = 1+6=7 \tag{5.4}$$
$$D_8(2) = 5 \tag{5.5}$$
$$D_8(3) = 4 \tag{5.6}$$
$$D_8(4) = 12+12=24 \tag{5.7}$$
$$D_8(5) = 14+1=15 \tag{5.8}$$
$$D_8(6) = 6 \tag{5.9}$$
$$D_8(7) = 4 \tag{5.10}$$
$$D_8(8) = 6+1+1+1+1+1+1+1=13 \tag{5.11}$$

つまり、パイプライン分割しない 1 ステージの場合遅延が 78 であり、8 ステージの場合最大遅延が 24 であると計算できる。



(a)1 ステージの場合　　(b)8 ステージの場合

図 5.1: データパスと遅延の例

第6章

評価

6.1 予備評価とその環境

5.2.2 節で述べたモデル式を用いて電力を最小化するパイプライン段数とボディバイアス電圧を探索するために予備評価を実施した。表 6.1 は全ての測定に共通する環境である。予備評価で測定した各データについて測定手法と測定結果を示す。

表 6.1: 測定環境

設計	verilogHDL
プロセス	LEAP65nm/LPT-8
論理合成	Synopsys Design Compiler 2016.03-SP4
配置配線	Synopsys IC Compiler 2016.03-SP4

6.1.1 遅延の見積もり

PE 単位で割り当てる演算とボディバイアスを変化させてシミュレーションを行い遅延時間を取得した。シミュレーションした演算の種類は表 4.1 に示したものである。さらに、ALU を経由せず、SE のみを経由する場合の遅延時間を取得した。シミュレーション環境を以下に示す。

表 6.2: 遅延時間のシミュレーション環境

対象	PE
シミュレーション	Synopsys HSIM 2012.06-SP2
電源電圧	0.55V
ボディバイアス電圧の範囲	-2.0V~ 0.4V で 0.2V 毎
温度	25°C

HSIM によるシミュレーションで得られる波形データから遅延時間を求めた。図 6.1 に得られたシミュレーション結果の例を示す。ボディバイアスが VBB=-0.6, ADD を演算として割り当てた PE での波形である。入力方向は南方向、出力方向は北方向であり、入力側の 25bit のうち 1bit 目が変化してから出力側の 23bit 目が変化するまでの時間を得た。入力側の変化は立ち上がり、立ち下がりの両方を測定し大きい方を最大遅延とした。この例では 22.901nsec を最大遅延として取得した。遅延を取得するビット位置の決定方法は Synopsis IC Compiler のタイミング情報を元に決定した。

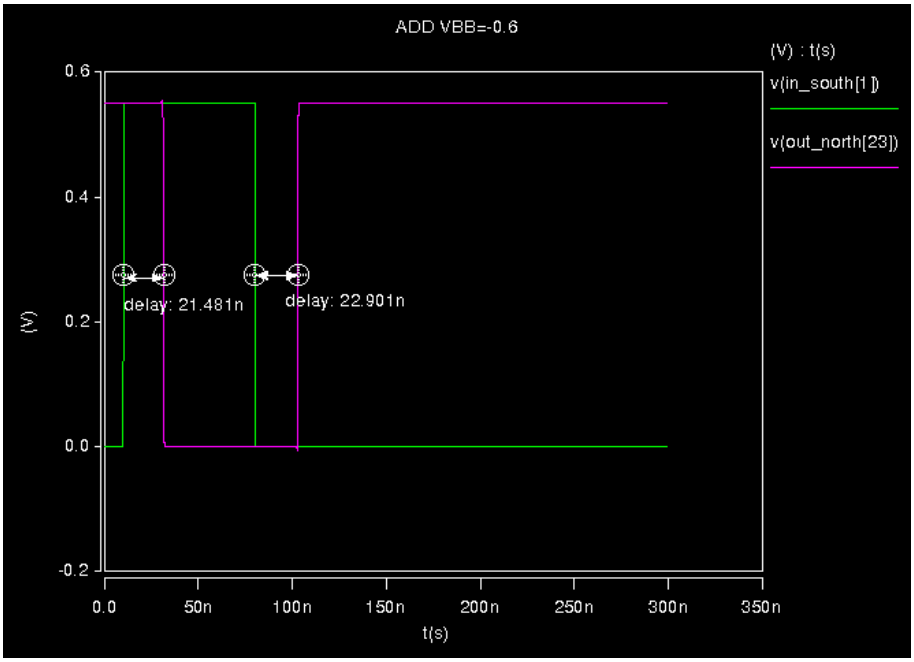


図 6.1: 遅延時間測定のための波形

図 6.2 に ADD をマッピングした PE における測定結果を示す。VBB が強いリバースバイアスになるにつれて指数関数に近い変化で遅延時間が増加していることが確認できる。電源電圧を VDD、スレッショルド電圧を V_{th} とした時 $VDD > V_{th}$ の領域では遅延時間はボディバイアス電圧に対して多項式的に変化する。一方で $VDD < V_{th}$ の領域では遅延時間はボディバイアス電圧に対して指数関数的に変化する。[7] 今回のシミュレーションでは VDD を 0.55V としており、V_{th} に近い値である。よって、多項式的変化と指数関数的変化の中間的な変化になるため図 6.2 のような変化となる。

6.1.2 リーク電力の測定

PE_ARRAY の行単位ー以降 ROW と呼ぶーでボディバイアスを変化させてシミュレーションを行いリーク電力を測定した。測定環境を以下に示す。

2.1.3 節で述べたようにリーク電力はスイッチングしていないにもかかわらず消費される電力であるため、入力値を変化させなかった時に消費される電力をリーク電力とした。ただし、入力値による違いを考慮し全ビットに 0 を入力した場合と全ビットに 1 を入力した場合の両方を測定し、その平均値を求めた。測定した結果を図 6.3 に示す。

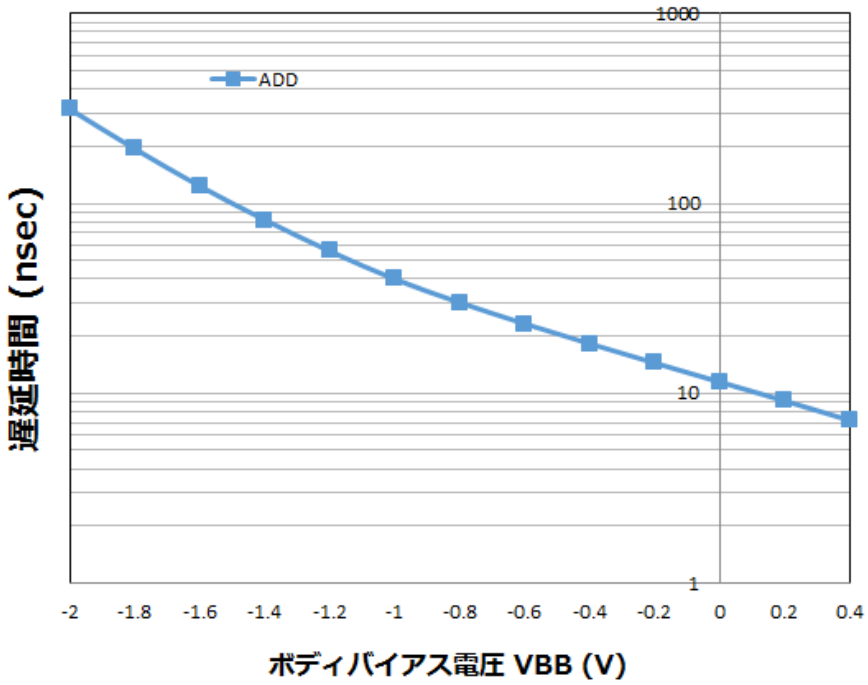


図 6.2: 遅延時間の測定結果

6.1.3 ダイナミック電力の測定

アプリケーション実行時のダイナミック電力を測定し、モデル式 (5.2) で用いるデータを測定した。測定したアプリケーションは表 6.4 に示す 4 種類である。測定環境を表 6.5 に示す。

Cadence NC-Verilog を用いてアプリケーション実行時のスイッチング率を求め、Synopsis Prime Time を用いてダイナミック電力を測定した。測定した結果を表 6.6 に示す。

6.2 評価結果

6.2.1 パイプライン分割の段数と電力最小化の関係

4 つのパイプライン分割パターン (1 段, 2 段, 4 段, 全段) でボディバイアスを制御し電力を最小化した結果を図 6.4~6.7 に示す。

どのアプリケーションにも共通して全段のパイプラインが最も小さい電力となっている。これはパイプライン段数を増やすことによる電力増加が小さい、加えてグリッチの削減とスループット向上のため動作周波数を低くできることによる動的電力の削減効果の方が大きいからである。VPCMA においてはパイプライン段数を増やす、つまり各ステージの遅延を小さくすることが有効である。この結果は VPCMA に特有である。他のパイプライン型 CGRA においてはステージの遅延を小さくするためには使用する PE の数を増やす必要がある。多くの CGRA では各 PE にクロック分配が行われているためステージの遅延を小さくすることによる電力増加は VPCMA に比べて大きい。また、PE 内に中間データ保持

表 6.3: リーク電力の測定環境

対象	ROW
設計	verilogHDL
シミュレーション	Synopsys HSIM 2012.06-SP2
電源電圧	0.55V
ボディバイアス電圧の範囲	-2.0V~ 0.4V で 0.2V 毎
温度	25°C
入力値	全ビットに 0 全ビットに 1

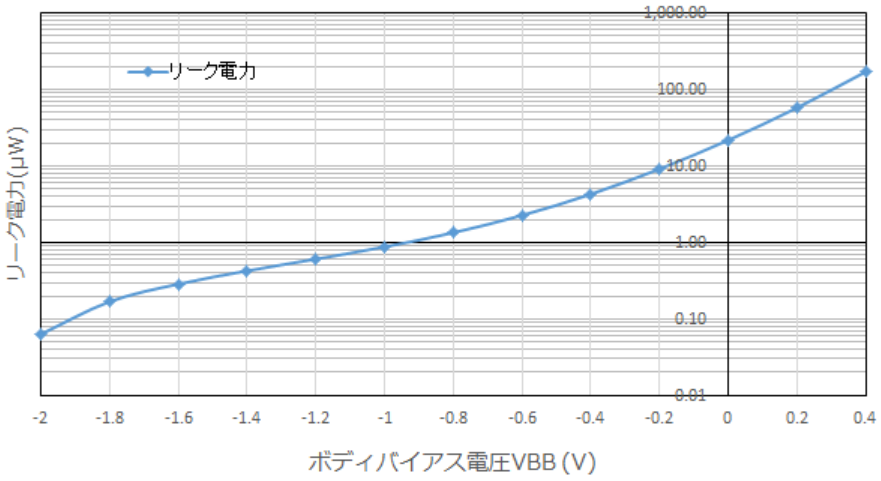


図 6.3: ROW におけるリーク電力の測定結果

のためのレジスタを持っていることが多いのでグリッチ削減による電力削減は期待できない。本研究ではパイプライン分割のパターンに制限を設けブルートフォース探索を行った。このことによってアーキテクチャによる違いに左右されずに電力を最小化するパイプライン段数を決定することができるとわかる。しかし、必ずしも全段が電力を最小化するとは限らない。例えば gray において本研究では考慮しなかった 7 段で電力が最小になる可能性がある。ゆえに、本手法で粗粒度にパイプライン段数を変化させてブルートフォース探索を行い、その段数の近傍で細粒度に探索を行うことが必要である。

次にアプリケーション間の関係を議論する。gray では他のアプリケーションと比べて全段にすることによる効果が小さい。この結果から gray には 4 段と 8 段との間にさらに電力を最小化する段数がある可能性を示している。全段にパイプライン分割する有効性と使用する PE_ARRARY の行数には関係がない。表 6.4PE の使用率を見ると gray は使用率、使用行数は大きい。一方で同じ行数を使用する sf では全段の分割は 4 段に比べて電力の削減効果は大きい。単純に使用率からではこうした効果は判断ができず本手法はこのようなアプリケーションの依存性に対しても対応できる。

表 6.4: シミュレーションしたアプリケーション

アプリケーション名	内容	PE 使用率 (%)	使用している行数
af	24bit(RGB) アルファブレンダ	75.00%	7 行
sf	8bit セピアフィルタ	83.33%	8 行
sepia	24bit(RGB) セピアフィルタ	50%	6 行
gray	24bit(RGB) グレースケール	81.25%	8 行

表 6.5: ダイナミック電力の測定環境

対象	PE_ARRAY
電力シミュレーション	Synopsys Prime Time 2012.12-SP3
動作シミュレーション	Cadence NC-Verilog 10.20-s131
電源電圧	0.55V
パイプライン段数	1,2,4, 全段

6.2.2 ボディバイアス制御の違いによる効果

次にボディバイアス制御を行うことの効果と行毎の粒度でボディバイアス電圧を制御することの効果について議論する。sepia においてボディバイアス制御を行わずにすべてゼロバイアスとする場合、PE_ARRAY に一律のボディバイアス電圧を与える場合、行単位でボディバイアス制御をする場合の 3 パターンでの消費電力を図 6.8 に示す。また、各アプリケーション、各段数でのボディバイアス制御をしない場合に対する行単位の制御による平均電力削減率を表 6.7 と図 6.9 に、一律制御に対する平均電力削減率を表 6.8 と図 6.10 に示す。

ボディバイアス制御を行うことで、行わない場合と比べて電力を大きく削減できていることがわかる。図 6.9 からわかるように電力を最小化と明らかになった全段のときに 45%を超える最も高い削減率を示している。段数を増やすと削減率が大きくなっているのは段数を増やせば遅延時間に余裕が生まれリバースバイアスを与えてリーク電力を削減する機会が増えるためである。一方で行単位の制御は一律制御と比べて電力の削減効果が小さく sepia 以外 1%に満たない。これは 6.2.1 節で述べたように最小化するときの段数が全段であるからである。本研究でシミュレーションした 5MHz~120MHz の領域では全段の場合、電力を最小化するときのボディバイアスがどの行に対しても-1.6V などの強いリバースバイアスを一定に与える場合が多かった。このため、PE_ARRAY に対して一律のボディバイアス電圧を与える場合と比べて大差がなかった。sepia が他のアプリケーションと比べて若干削減率が高い理由は行単位の制御にすると使用していない行に強いリバースバイアスを与えることができるからである。6 行した使用していない speia はこの点で他のアプリケーションと比べ効果が高い。図 6.10 を見ると電力が最小とはならなかった 2 段、4 段では全段に比べ高い電力削減率となっている。特に 2 段のときでは 18%を超える削減率を示している。アプリケーションを sf, 段数を 2 段、要求性能を 45MHz としたときには行単位でボディバイアス制御をすると一律制御と比べて 51.14%の電力削減率が得らた。同様に

表 6.6: ダイナミック電力の測定結果

		af (μW/MHz)	sf (μW/MHz)	sepia (μW/MHz)	gray (μW/MHz)
E_{comb}	$N_p = 1$	50.1430	116.5400	128.6133	46.7077
	$N_p = 2$	17.8915	24.2268	30.1860	9.6325
	$N_p = 4$	6.2765	8.8185	9.2963	3.5283
	$N_p =$ 全段	2.68138	3.49525	2.74738	1.44004
E_{clk}		3.73297	3.69753	3.74962	3.76229
E_{reg}		0.26836	0.35357	0.10738	0.08125

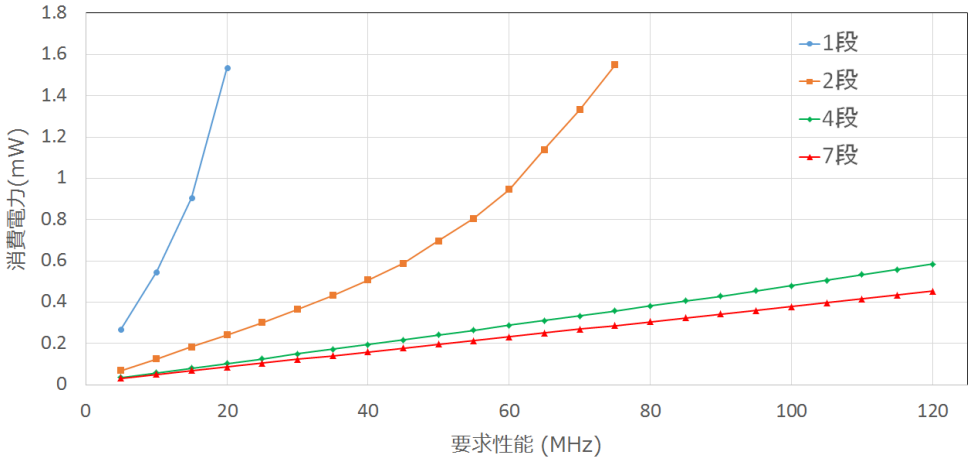


図 6.4: af: 電力最小化の結果

sf で 4 段、要求性能 110MHz とした場合 34.56%の削減率が得られた。要求性能が高くなると遅延時間を小さくするために 0V に近いリバースバイアスまたはフォワードバイアスを与える必要が出てくる。遅延のボトルネックとなる行は限られているが一律制御の場合は全体に対してフォワードバイアスを与えなくてはならなくなりリーク電力が増大する。一方で行単位の制御ではボトルネックとなる行のみフォワードバイアスを与え、そうではない行には異なるボディバイアス電圧を与えることができる。したがって、要求性能が高い場合には行単位の制御は効果的である。今回は 120MHz までの要求性能しか考慮していないがより高い性能であれば全段でも高い削減率を得られる。ただし、これに関してもアプリケーション依存性がある。なぜならば行単位で遅延のばらつきが発生していないようなマッピングであれば要求性能が高くなっても一律制御で十分だからである。しかし、そういったアプリケーションは稀であるため高い性能が要求される場合は行単位の制御を行うべきである。

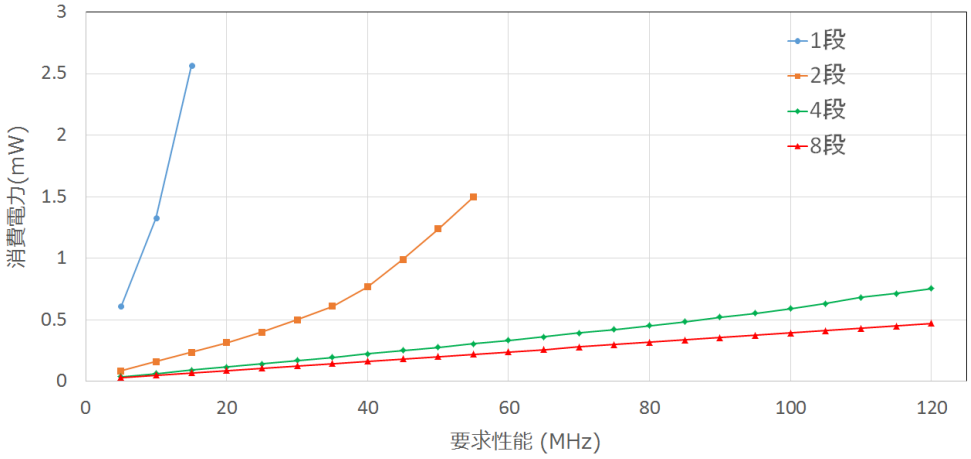


図 6.5: sf: 電力最小化の結果

表 6.7: ゼロバイアス時に対する平均電力削減率

アプリケーション	1 段	2 段	4 段	全段
af	20.88%	37.73%	40.60%	46.28%
sf	11.25%	35.54%	38.71%	45.34%
sepia	14.86%	29.81%	37.01%	46.40%
gray	27.02%	44.02%	43.48%	47.15%

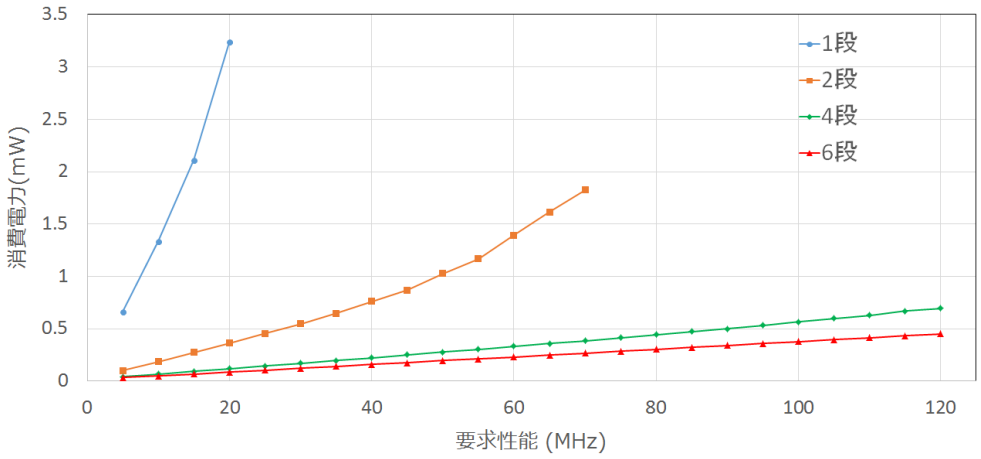


図 6.6: sepia: 電力最小化の結果

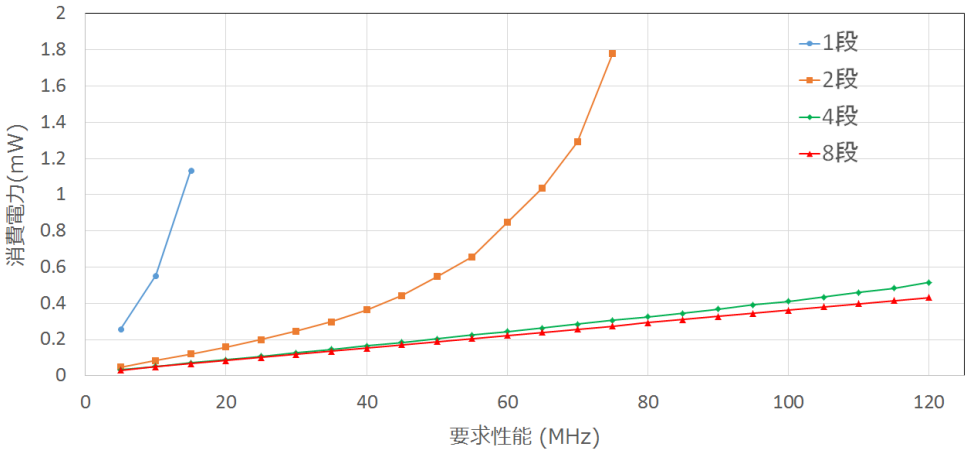


図 6.7: gray: 電力最小化の結果

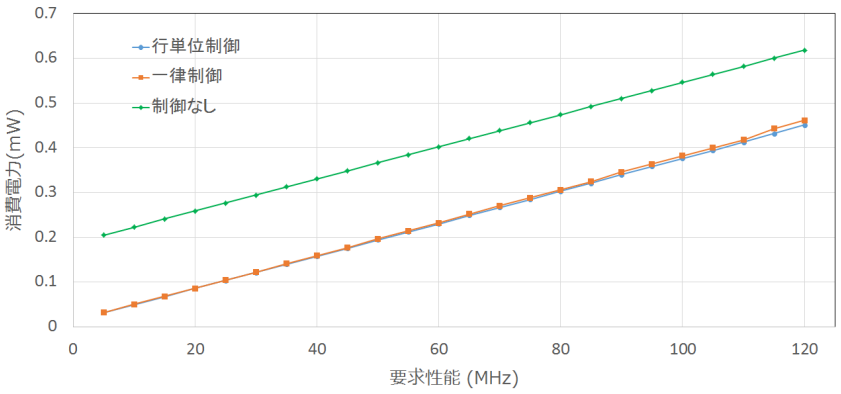


図 6.8: sepia における電力最小化結果の比較

表 6.8: 一律ボディバイアス制御に対する平均電力削減率

アプリケーション	1 段	2 段	4 段	全段
af	11.91%	16.46%	3.25%	0.43%
sf	6.77%	18.17%	9.90%	0.80%
sepia	8.47%	13.76%	6.82%	1.19%
gray	7.14%	13.04%	4.49%	0.76%

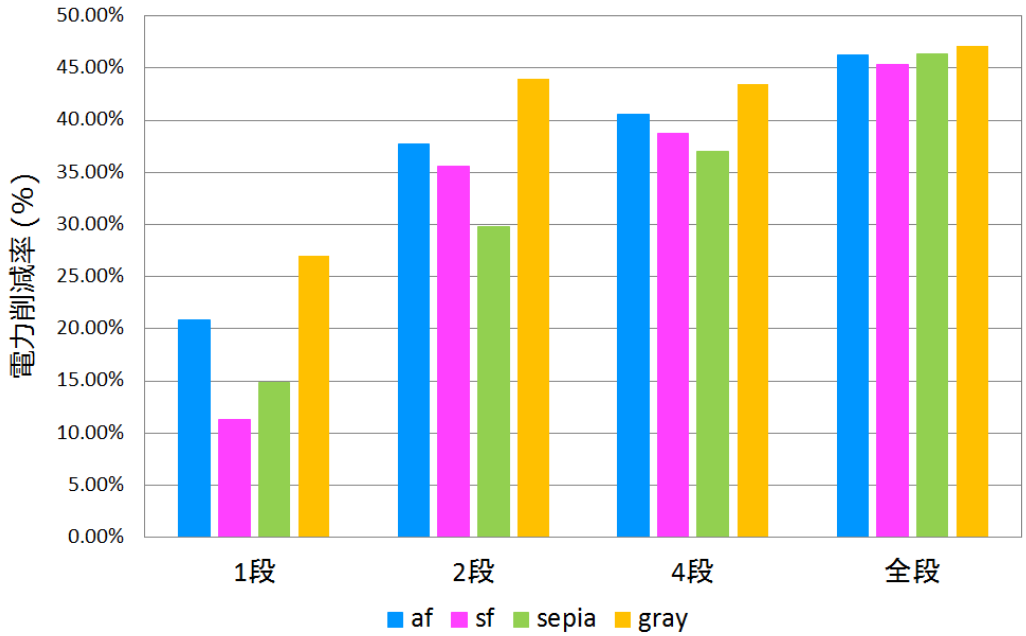


図 6.9: ゼロバイアス時に対する平均電力削減率

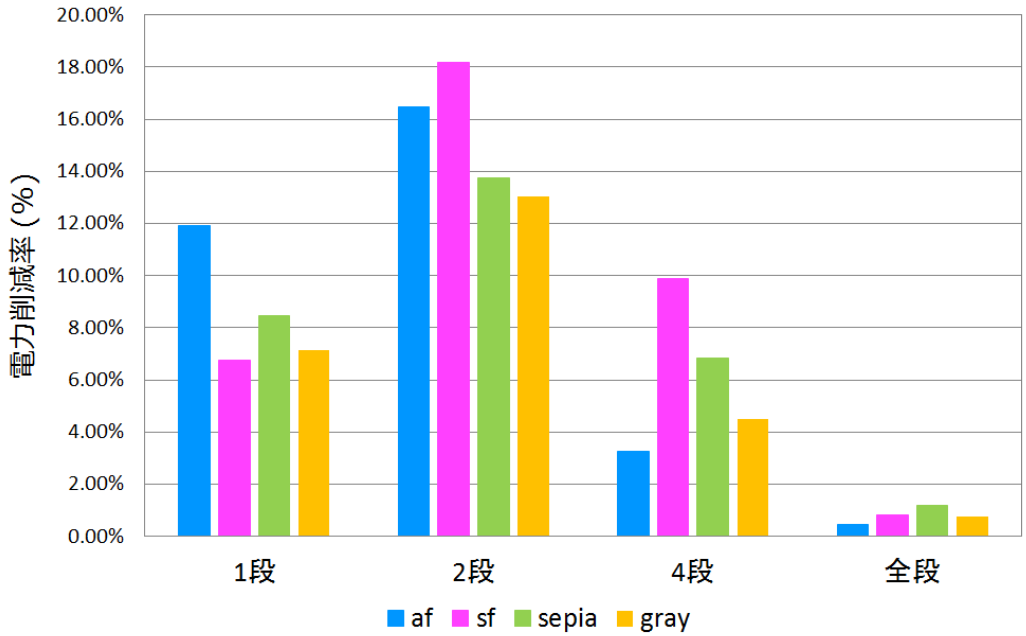


図 6.10: 一律ボディバイアス制御に対する平均電力削減率

第7章

まとめと今後の課題

7.1 結論

VPCMA において、実行するアプリケーションと要求性能に応じて電力を最小化するパイプライン段数とボディバイアス電圧を決定する手法を検討した。トレードオフの複雑さから単純に計算することが困難であったため探索はブルートフォースで行った。4 つの 24bit 画像処理アプリケーションを実行するシミュレーションを行い評価を行った。

パイプライン段数に着目すると VPCMA おいてはアプリケーションや要求性能によらず全段にパイプライン分割した場合がもっとも電力が小さいとわかった。この結果は VPCMA 特有のものであり、ブルートフォース探索によってによって明らかとなったアーキテクチャの特徴とも言える。このように本手法ではアーキテクチャに依存せずに電力を最小化するパイプライン段数を求めることができた。

ボディバイアス制御の粒度を行単位とすることによる効果をボディバイアス制御をしない場合、制御の粒度を PE_ARRAY 全体とした場合との比較を行うことにより検討した。ボディバイアス制御をしない場合と比べて平均約 46%の電力削減率が得られた。一方で一律制御と比べると電力削減率は平均 0.80%であった。しかし、要求性能が高くなるにつれて行単位の制御では一律制御に対して高い電力削減率を示し、その効果はアプリケーションと要求性能に依存することがわかった。

7.2 今後の課題

本論文で検討した手法にはブルートフォース探索を採用していることはすでに述べた。そのため、パイプライン分割のパターンを制限してもパイプライン段数とボディバイアス電圧を決定するのに時間がかかっていた。このようにパイプライン段数は粗粒度に探索を行っているため本研究では想定していないパイプライン分割のパターンの方がより小さい電力を示す可能性がある。ただし、本研究で示されたように電力を最小化する段数は使用している行数の付近にあるはずであり、探索はその周辺を行えばよい。よって、本論文で提案した手法のアルゴリズムには改良の余地がある。例えば遺伝的アルゴリズムなどを代表とする進化的計算アルゴリズムを採用し得られる結果と本研究のブルートフォース探索の結果を比較することにより、この手法に適するアルゴリズムを検討する必要があると考えられる。

アプリケーション依存性があるということは、どの PE にどの演算をマッピングするかに依存するということである。本研究ではアプリケーションのマッピングは固定したままでパイプライン段数とボディバイアス電圧を変化させて電力を見積もり最小のものを探索していた。しかし、各々のパイプライン段数、ボディバイアス電圧に対する適切な演算のマッピングは異なる可能性がある。したがって、演算のマッピングを変化させた場合の影響についても評価を行う必要があると考えられる。

第8章

謝辞

本研究に取り組むにあたりご指導ご鞭撻を賜りました天野英晴教授、奥原颯氏に深く感謝いたします。本研究で対象としたVPCMAやその他のCMAアーキテクチャについての研究を行ってきた増山滉一朗氏、安藤尚輝氏、CMAにおけるボディバイアス制御の研究を行ってきた松下悠亮氏、及び様々な面で貴重な助言と励ましを下さった研究室の皆様に深く感謝致します.

参考文献

[1] Yoshiki Saito, T. Sano, M. Kato, V. Tunbunheng, Yoshihiro Yasuda, Masayuki Kimura, and H. Amano. Muccra-3: A low power dynamically reconfigurable processor array. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 377–378, Jan 2010.

[2] N. Ozaki, Y. Yasuda, M. Izawa, Y. Saito, D. Ikebuchi, H. Amano, H. Nakamura, K. Usami, M. Namiki, and M. Kondo. Cool mega-arrays: Ultralow-power reconfigurable accelerator chips. *IEEE Micro*, Vol. 31, No. 6, pp. 6–18, Nov 2011.

[3] H. Su, Y. Fujita, and H. Amano. Body bias control for a coarse grained reconfigurable accelerator implemented with silicon on thin box technology. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–6, Sept 2014.

[4] K. Masuyama, Y. Fujita, H. Okuhara, and H. Amano. A 297mops/0.4mw ultra low power coarse-grained reconfigurable accelerator cma-sotb-2. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–6, Dec 2015.

[5] Y. Matsushita, H. Okuhara, K. Masuyama, Y. Fujita, R. Kawano, and H. Amano. Body bias grain size exploration for a coarse grained reconfigurable accelerator. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, Aug 2016.

[6] Variable Pipeline Structure for Coarse Grained Reconfigurable Array CMA. Naoki ando, koichiro masuyama, hayate okuhara, hideharu amano. In *2016 INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE TECHNOLOGY*, pp. 231–238, 2016.

[7] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, 4th edition, 2010.

[8] Tezaswi Raja, Vishwani D Agrawal, and Michael L Bushnell. Variable input delay cmos logic for low power design. *IEEE transactions on very large scale integration (VLSI) systems*, Vol. 17, No. 10, pp. 1534–1545, 2009.

[9] Y. Morita, R. Tsuchiya, T. Ishigaki, N. Sugii, T. Iwamatsu, T. Ipposhi, H. Oda, Y. Inoue, K. Torii, and S. Kimura. Smallest vth variability achieved by intrinsic silicon on thin box (sotb) cmos with single metal gate. In *2008 Symposium on VLSI Technology*, pp. 166–167, June 2008.

[10] T. Ishigaki, N. Sugii, R. Tsuchiya, S. Kimura, and Y. Morita. *Ultralow-power LSI Technology with Silicon on Thin Buried Oxide (SOTB) CMOSFET.*, chapter 7, pp. 145–156. INTECH Open Access Publisher, 2010.

[11] Herman Schmit, David Whelihan, Andrew Tsai, Matthew Moe, Benjamin Levine, and R Reed Taylor. Piperench: A virtualized programmable datapath in 0.18 micron technology. In *Custom Integrated Circuits Conference, 2002. Proceedings of the IEEE 2002*, pp. 63–66. IEEE, 2002.

[12] Mario Konijnenburg, Yeongojn Cho, Maryam Ashouei, Tobias Gemmeke, Changmoo Kim, Jos Hulzink, Jan Stuyt, Mookyung Jung, Jos Huisken, Soojung Ryu, et al. Reliable and energy-efficient 1mhz 0.4 v dynamically reconfigurable soc for exg applications in 40nm lp cmos. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 430–431. IEEE, 2013.

[13] Jeffrey M Arnold. S5: the architecture and development flow of a software configurable processor. In *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, pp. 121–128. IEEE, 2005.

[14] Mihail Petrov, Tudor Murgan, Frank May, Martin Vorbach, Peter Zipf, and Manfred Glesner. The xpp architecture and its co-simulation within the simulink environment. In *International Conference on Field Programmable Logic and Applications*, pp. 761–770. Springer, 2004.

[15] Xitian Fan, Huimin Li, Wei Cao, and Lingli Wang. Dt-cgra: Dual-track coarse-grained reconfigurable architecture for stream applications. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pp. 1–9. IEEE, 2016.