修 士 論 文

# ステージ分割によるウェーブパイプライン方式の最適化の研究

指導教官　日比野 靖 教授

審査委員主査　日比野 靖 教授

審査委員　篠田 陽一 助教授

審査委員　堀口 進 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

福家 和久

2001 年 2 月 15 日

## 要 旨

プロセッサ自身のスループット向上を目的として、パイプライン手法が研究されてきた。その中でもウェーブパイプライン方式は、通常のパイプラインよりも短いサイクルで、各ステージが別々のタイミングで動作することにより、より高いスループットが期待できる。ウェーブパイプライン方式によるプロセッサの性能は、それぞれのステージの最大遅延と最小遅延の差の最大値に依存する。よって、この遅延差を短縮してやればプロセッサ性能の向上につながる。

本研究では、このウェーブパイプライン方式をプロセッサに適用するにあたり、その設計において問題となる遅延差短縮の効率のよい手法を提案し、その有効性を評価する。

# 目 次


**1 序論** **1**

**2 ウェーブパイプライン方式** **3**

2.1 ウェーブパイプライン方式の概要 . . . . . 3

2.2 通常のパイプライン方式とウェーブパイプライン方式の違い . . . . . 4

2.2.1 通常のパイプライン方式 . . . . . 4

2.2.2 通常のパイプライン方式とウェーブパイプライン方式の違い . . . . 5

2.3 ウェーブパイプライン方式の問題点 . . . . . 6

**3 ステージ分割による遅延差短縮手法** **8**

3.1 ステージ分割による遅延差短縮手法の目的 . . . . . 8

3.2 遅延バッファ挿入による遅延差短縮手法 . . . . . 9

3.2.1 遅延バッファ挿入による遅延差短縮アルゴリズム . . . . . 9

3.2.2 遅延測定方法 . . . . . 11

3.3 ステージ分割による遅延差短縮手法 . . . . . 12

3.4 ステージ分割による遅延差短縮アルゴリズム . . . . . 13

3.4.1 ステージ分割による遅延差短縮アルゴリズム . . . . . 13

3.4.2 ステージの分割手法 . . . . . 14

3.4.3 最適値の考慮 . . . . . 18

3.5 パスの遅延計算における計算量 . . . . . 19

**4 設計** **21**

4.1 ウェーブパイプライン化する対象のプロセッサ . . . . . 21

4.2 遅延 . . . . . 24

4.2.1 遅延モデル . . . . . 24

4.2.2 遅延パラメータ . . . . . 25

4.3 レイアウト . . . . . 26
4.3.1 レイアウト方法 . . . . . 26

**5 結果** **28**
5.1 実行ステージの遅延差 . . . . . 28
5.2 バッファ挿入のみの遅延差短縮 . . . . . 29
5.3 ステージ分割による遅延差短縮 . . . . . 32
5.3.1 目標値均衡を目的とする場合 . . . . . 34
5.3.2 高性能化を目的とする場合 . . . . . 37

**6 考察** **40**
6.1 性能による考察 . . . . . 40
6.2 手法としての考察 . . . . . 46

**7 結論** **47**

# 第 1章

# 序論

プロセッサの動作速度を向上させる目的で、複数の命令をオーバラップさせて同時実行する技術であるパイプライン手法が研究されてきた。パイプライン手法は n 個のパイプラインステージを用いることで、n 倍の高速化の可能性がある。その性能は、パイプラインステージの最大遅延に依存する。

本研究室では、パイプライン手法の 1 手法であるウェーブパイプライン方式に関する研究を行っている。ウェーブパイプライン方式は、通常のパイプラインよりも速いクロックで動作することにより、高いスループットが期待できる。このウェーブパイプライン方式によるプロセッサ性能は、通常のパイプライン方式が最も遅延の大きなステージの最大遅延によって決定されるのとは異なり、各ステージの遅延差の最大値に依存するため、この遅延差を短縮することにより、プロセッサの性能において通常のパイプライン手法よりも高い向上が可能になる。

本論文では、このウェーブパイプライン方式をプロセッサに導入するに当たって、これまでの研究において問題になっていた素子数の増加によるハードウェア量の増大と、それに伴う面積と消費電力の増加を抑える一方、より高い性能を求めるためにステージ分割を行って遅延差短縮を行う方法を提案する。

本論文の構成は、第 2 章でウェーブパイプライン方式について通常のパイプラインと比較しつつ説明する。そしてこれまでの本研究室でのウェーブパイプライン方式に関する研究においてその結果と問題点を述べる。第 3 章では、ステージ分割によるウェーブパイプラインの遅延差短縮手法を説明し、それにおける問題点も述べる。またもう 1 つの手法である遅延バッファの挿入による遅延差短縮手法についても説明する。第 4 章では、ステージ分割による遅延差短縮手法を導入するに当たって、必要となるプロセッサの構成や、ゲート長 $0.10\mu m$ における素子と配線の遅延モデルを示し、評価する。また、設計に

おいて問題となる素子のレイアウトについての考察も行う。第5章は、実行ステージをステージ分割による遅延差短縮手法とバッファ挿入による遅延差短縮手法を行うことによるバッファ挿入量、面積、遅延差を示す。第6章は、第5章で得られた結果より提案した手法における考察を行う。第7章において、本論文の結論をまとめる。

# 第 2章

# ウェーブパイプライン方式

プロセッサの動作速度を向上させる手法として、パイプライン方式がある。この章では、ウェーブパイプライン方式を説明し、通常のパイプライン方式と比較することによって、ウェーブパイプライン方式を導入する利点を述べる。

## 2.1 ウェーブパイプライン方式の概要

ウェーブパイプライン方式は、クロックサイクルは一定であるが、通常のパイプライン方式よりも短いサイクルで、各ステージが別々のタイミングで動作する。

図 2.1 はウェーブパイプライン方式について時間的、空間的に統合した図であり、これを遅延伝搬グラフと呼ぶ。横軸は時間であり、縦軸はステージないの回路の論理的な深さを表している。影の部分は処理が実行されているであろう不確定な時間間隔を意味する。この影は遅延であるが、その外形はあるステージ内の最長パスと最短パスを伝わった最も速い、そして最も遅い信号の遅延として視覚化されている。

ウェーブパイプライン方式の性能は、ステージの最大遅延時間と最小遅延時間の差の最大値になる。

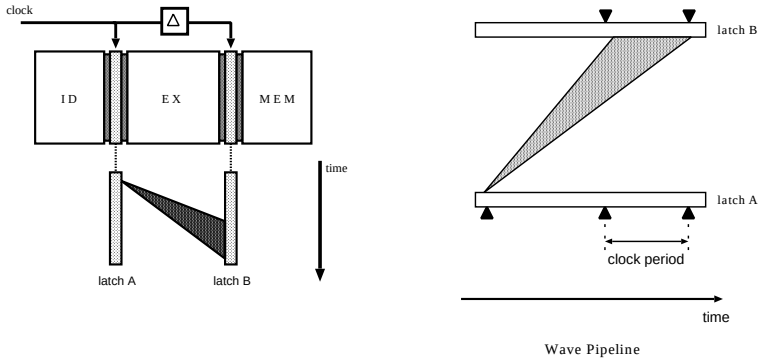


図 2.1: ウェーブパイプライン方式

## 2.2 通常のパイプライン方式とウェーブパイプライン方式の違い

### 2.2.1 通常のパイプライン方式

パイプラインは並列性を利用して高速化を図る手法である。このパイプライン方式は、複数の命令をオーバラップさせて同時実行する手法であり、現在、プロセッサの高速化のための基本技術となっている。

パイプラインは、1 つの命令を連続的に実行される複数の処理単位に分割する。この処理単位をパイプラインステージと言い、各ステージが図 2.2 のように順に接続されている。

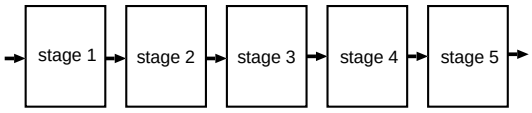


図 2.2: パイプライン

パイプラインのスループットは、命令がパイプラインを出て行く速度で決まる。各ステージは互いに連結しているので、全てのステージが同時に処理を完了しなければならない。1 ステージの処理に要する時間がサイクル時間である。サイクル時間は、最も遅いステージの処理時間によって決まる。本論文では、ステージの処理時間をステージの遅延時間とする。

あるステージでの遅延時間を遅延伝搬グラフ 図 2.3 として示す。斜線部分が遅延時間

である。この図では簡単化のために遅延時間を直線で表しているが、ステージ内の回路構成によって必ず直線になるとは限らない。遅延時間の直線の傾斜が緩やかな方が最大遅延であり、傾斜が急な方が最小遅延である。サイクル時間は latch A からステージに処理が入る時間から latch B に処理が終了して入る時間の差である。

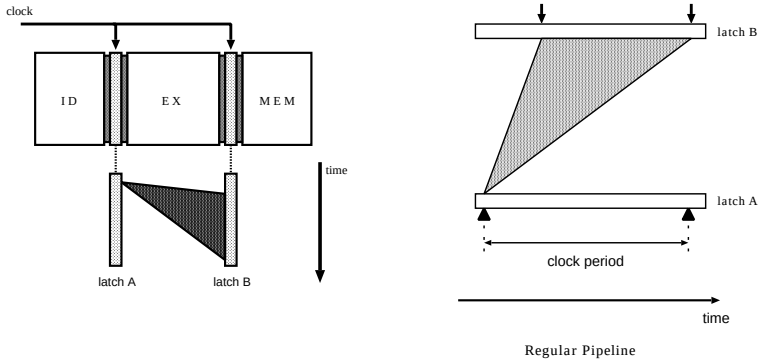


図 2.3: 通常のパイプライン

すなわち、遅延時間の小さいステージが待たされることになり、全体のクロックサイクル時間は遅延の最も大きいステージの時間になってしまう。

### 2.2.2 通常のパイプライン方式とウェーブパイプライン方式の違い

通常のパイプライン方式とウェーブパイプライン方式の動作を図 2.4 と図 2.5 に示す。

この命令でのパイプラインステージにおいて最も遅延時間が大きいのは stage3 である。また、遅延差が最も大きいのは stage4 である。

通常のパイプライン処理の場合、クロックサイクル時間は stage3 の処理時間となる。

ウェーブパイプライン処理の場合、クロックサイクル時間は stage4 の遅延差となる。図 2.4 と図 2.5 から、通常のパイプライン方式に比べてウェーブパイプライン方式の方がクロックサイクル時間が短く、リソースの使用効率が高いことが分かる。

上図とこれまで述べてきた通常のパイプライン方式とウェーブパイプライン方式より違いを示す。

- クロックサイクル時間が通常のパイプライン方式の場合、パイプラインステージの最大遅延時間によって決定されるが、ウェーブパイプライン方式の場合、ステージの最大遅延時間と最小遅延時間の遅延差によって決定される。

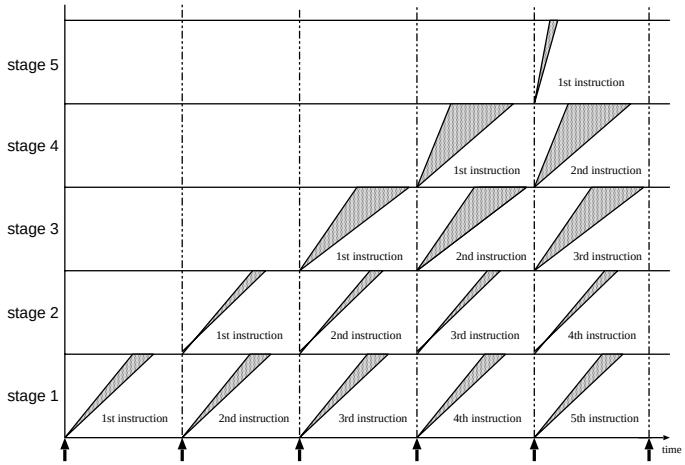


図 2.4: 通常のパイプライン処理

- 各ステージにおいて遅延が生じるのは当然のことである。通常のパイプライン方式の場合、その最大遅延時間をできるだけ小さくし、パイプラインステージの処理時間を均衡化することが重要になる。つまり遅延を無くすことを考える。これは非常に困難である。しかしウェーブパイプライン方式の場合、生じる遅延を利用して遅延差をより小さくすることが重要になる。つまり遅延の問題を回避するために生まれた手法である。

- 通常のパイプライン方式の場合、各ステージに同時にクロックを入れる必要があるが、ウェーブパイプライン方式の場合はそれを考慮しなくてよい為、クロックスキューの問題が解決しやすい。

- パイプラインステージ数を増やすと通常のパイプライン方式の場合、レイテンシが増加するが、ウェーブパイプライン方式では増加しない。

よって、プロセッサの性能向上を目的とした場合、ウェーブパイプライン方式を導入することにより、より高い性能向上が図れる。

## 2.3 ウェーブパイプライン方式の問題点

本研究室では、ウェーブパイプライン方式について研究 [6],[8] されてきた。

本研究室では、ウェーブパイプライン方式を導入したプロセッサの設計において、その性能を決定する最大遅延と最小遅延の遅延差を最小にするために、決められたステージの

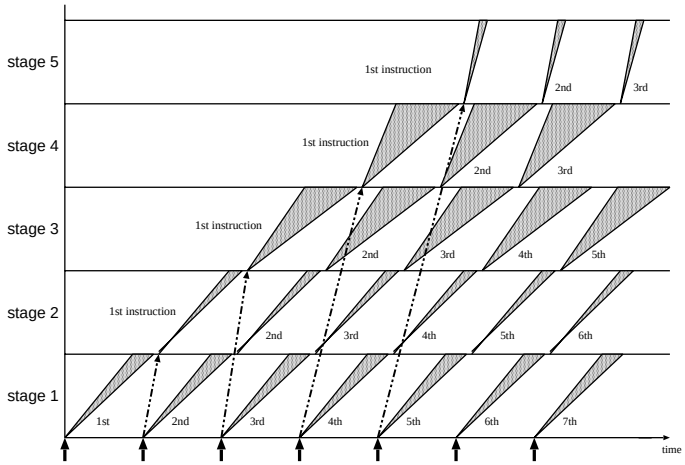


図 2.5: ウェーブパイプライン処理

中で第 3 章で詳しく述べる遅延バッファ挿入による遅延差短縮手法を行っている。

それによって、ウェーブパイプライン方式を導入するに当たっていくつかの問題点が存在している。

- 遅延バッファの挿入による配線遅延とレイアウトを毎回行うことにより設計効率が悪い。
- 上記の問題を解決するために、ステージ内をいくつかのブロックにまとめてそれぞれ別々に遅延バッファ挿入による遅延差短縮を行い、最後に全体のステージとして遅延差短縮を行う。これにより遅延バッファ挿入による設計時間を 45% 短縮できるが、プロセッサの性能、面積共にブロックにまとめた場合の方が劣る。
- プロセッサの性能向上と共に、遅延バッファ挿入による面積の増大も激しい。プロセッサの性能向上約 2.5 倍において面積の増加は約 2.0 倍となる。それにより消費電力の増加も大きくなるために、ウェーブパイプライン方式を導入した場合におけるプロセッサの性能向上の利点が薄い。

# 第 3章

# ステージ分割による遅延差短縮手法

この章では第2章の最後に述べた問題点より、これらの問題を解決するための本論文での提案であるステージ分割による遅延差短縮手法とバッファ挿入による遅延差短縮手法について述べる。

## 3.1 ステージ分割による遅延差短縮手法の目的

ウェーブパイプライン方式を導入することにより、プロセッサの動作速度をより高くすることが一番の目的となるが、第2章で述べたウェーブパイプライン方式を導入することによって生じる問題点である、設計時間の短縮と、性能向上と比べて遅延バッファ挿入による面積と消費電力の増大を抑えることも目的としている。

遅延差短縮を行う場合、次の2つのやり方がある。

- 遅延差が最も小さくなる限界までステージを分割したり論理レベルでパスを再構成するなどして最大遅延時間を短縮したり、遅延バッファの挿入によって最小遅延時間を最大遅延時間に均衡させて遅延差の短縮を図る。
  - 限界までステージを分割することは、ステージ分割がラッチを挿入することによって行われるためラッチを挿入することによるハードウェア量が大きくなる。そしてステージ数が遅延バッファをどんどん挿入するため素子数が激しく増加し、面積や消費電力に大きな影響を与える。
  - 遅延差を限界まで短縮するため、性能が良い。
- あらかじめ目標となる遅延差を設定しておき、それを満足するまで遅延差短縮を行う。

– 目標が決定しているので、遅延差が目標値以下となるとステージ分割もバッファ挿入も行わない。これによりハードウェア量の増加を抑えることができ、それに伴う面積や消費電力も抑えることができる。

– 性能の観点では、上の手法に劣る。

本論文では、後者のやり方で遅延差短縮を行う。

## 3.2 遅延バッファ挿入による遅延差短縮手法

遅延バッファ挿入による遅延差短縮手法は、図 3.1 のように最大遅延時間を増加させないよう、最小遅延のパスに遅延バッファを挿入することによって最小遅延を最大遅延に近づけ、遅延差の短縮を行う手法である。この遅延バッファの挿入による遅延差の均衡によって、各ラッチに入れるクロックの遅延時間が決定される。

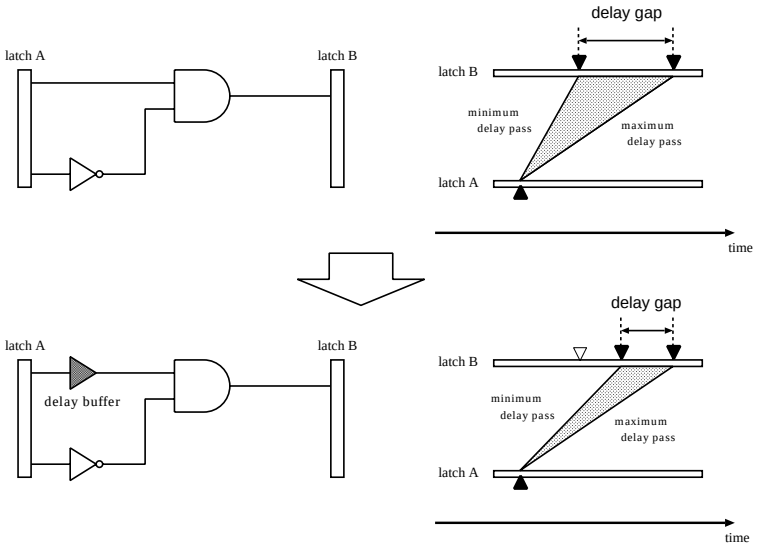


図 3.1: 遅延バッファの挿入による遅延差の均衡

### 3.2.1 遅延バッファ挿入による遅延差短縮アルゴリズム

遅延バッファ挿入による遅延差短縮手法のアルゴリズムは基本的に次のようになっている。

1. ウェーブパイプライン化する対象のステージのネットリストを得る。

2. あらかじめ遅延バッファを挿入するスペースを確保してレイアウトを行う。

3. ステージ内の配線遅延を計算し、そのステージの全てのパスにおける遅延を出力側から入力側に向けて計算する。これより最大遅延差を求める。

4. 最小遅延パスに遅延バッファを挿入して再びレイアウトを行う。

5. 再びステージ内の入力から出力までの全てのパスの遅延を計算する。

6. 終了

このアルゴリズムで遅延差を短縮させる。

1. において対象となるステージのネットリストはPARTHENONにより得られる。また素子と配線の遅延パラメータをSPICEを用いて求めておく。その他に必要となる情報もあらかじめ求めておく。

2. のレイアウトは後述する第1段階レイアウトである。確保するスペースについて、この場合、経験的に挿入される遅延バッファ数は対象となる回路の3倍程度である。

3. では、次に述べる遅延測定によってステージ内の遅延時間を求める。その後、各出力における最大遅延、最小遅延、遅延差を求める。この各遅延差のうち最大となる出力端子を求める。

4. において3. か5. で求めた最大遅延差の出力端子をターゲットとして、その最小遅延パスに遅延バッファを挿入する。これによって遅延差を短縮させる。その後のレイアウトは遅延バッファのレイアウトである。

5. で得られるステージ内の遅延差が目標の値であれば6. へ行く。逆に遅延差が目標の値になっておらず、バススロットにも空きがある場合は4. に戻る。バススロットを使い切った場合、空きスロットが多い場合などは、2. に戻る。

基本的に遅延バッファ挿入による遅延差短縮は4. と5. を繰り返すことによって達成される。

6. 遅延差が目標の値になっており、空きスロットもほとんどない場合、目標が達成されたので終了する。

次に、3. や5. などのステージ内の遅延時間を求めるのに必要な遅延測定方法について述べる。

### 3.2.2 遅延測定方法

遅延バッファ挿入による遅延差短縮や次に述べるステージ分割を行う場合には、まずステージ内の回路の全てにおいてその最大遅延時間と最小遅延時間を知る必要がある。

そのために、ある入力端子から出力端子までの遅延時間を測定しなければならない。遅延は、遅延計算手法によって見積もる。

あるステージにおいてその入力端子からある出力端子までの最大遅延時間と最小遅延時間を測定するとする。その入力端子から出力端子までの全てのパスについて、そのパスが通過するゲート (素子) 遅延と配線遅延の和で遅延時間が求まる。

ここで、あるステージ内の回路が図 3.2 の様であるものを例として用いる。このステージの入力からあるゲートまでの最大遅延時間と最小遅延時間を見積もる。遅延時間の見積りは以下のようにして行う。

1. 入力からあるゲートまでの全てのパスについて、そのパスが通る遅延の和を計算する。遅延は、ゲート遅延に配線遅延も含んでいる。

   図 3.2 において、簡単のために 1 つのゲート遅延を 1 とし、ゲートにおける立ち上がり時間と立ち下がり時間はここでは無視する。また配線遅延についても無視するとする。説明のため、入力 A には遅延 1 が入力 C には遅延 2 があるとする。

2. 入力からあるゲートまでの全てのパスにおける全ての遅延時間から、min(立ち上がり時間、立ち下がり時間) であるものが最小遅延時間、max(立ち上がり時間、立ち下がり時間) であるものが最大遅延時間である。

   図 3.2 で、$maxX - minY$ は、最大遅延時間 $X$、最小遅延時間 $Y$ を意味する。

3. 入力端子 A、B、C、D から最初のゲート (それぞれ gate1、gate2 ) の入力側で遅延を計算する。すなわちそれぞれのゲート遅延そのものは含まない。ここでは、配線遅延はないので gate1 の遅延は $max1 - min0$ となり、gate2 の遅延は $max2 - max0$ となる。

4. 最初のゲート gate1、gate2 の次のゲート gate3 の入力側で遅延を計算する。よって gate3 の遅延は $max3 - max1$ となる。

5. 次は gate4 の入力側での遅延を計算する。遅延は $max4 - min0$ となる。

6. 最後に出力端子 z に入る前の遅延を計算する。遅延は $max5 - min1$ となるので、このステージにおける遅延差は 4 となる。

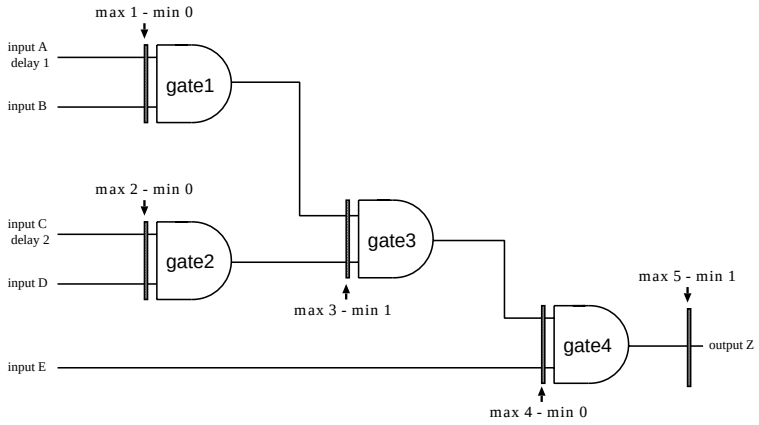


図 3.2: 遅延見積りの例

**遅延計算手法の利点**

この遅延計算手法の利点を述べる。ステージ内の全ゲートが、先ほどの遅延見積り手法で得られる最大遅延時間と最小遅延時間で動作することはほとんどあり得ない。言うなれば、この手法での遅延見積りは正確な値ではない。しかし見積もられた最大遅延時間は実際の最大遅延時間よりも十分に大きく、最小遅延時間も実際の最小遅延時間より小さい。これより、この手法で得られる遅延見積り値における遅延差は実際の遅延差よりも十分大きな値になる。ウェーブパイプライン方式によってこの遅延差より遅延均衡を行い、得られた遅延差より決まるクロックサイクル時間は、実際に動作するのにも十分な時間になる。

この遅延計算手法の利点は、ステージにおける遅延差の最大値が常に出力端子になることである。よって最初に挙げたように、ステージの途中で遅延差が最大になることがない。

## 3.3 ステージ分割による遅延差短縮手法

図 3.3 のように遅延差最大のステージにラッチを置くことでステージを分割する。

ラッチを置くことは、そのステージ中の遅延差を一旦無くすことで遅延差の増加を防ぐことができるので、遅延差を短縮することができる。またこの手法は最大遅延時間そのものを短縮することも特徴の 1 つである。

この分割を行ったステージのそれぞれにおいて遅延バッファ挿入による遅延差短縮も

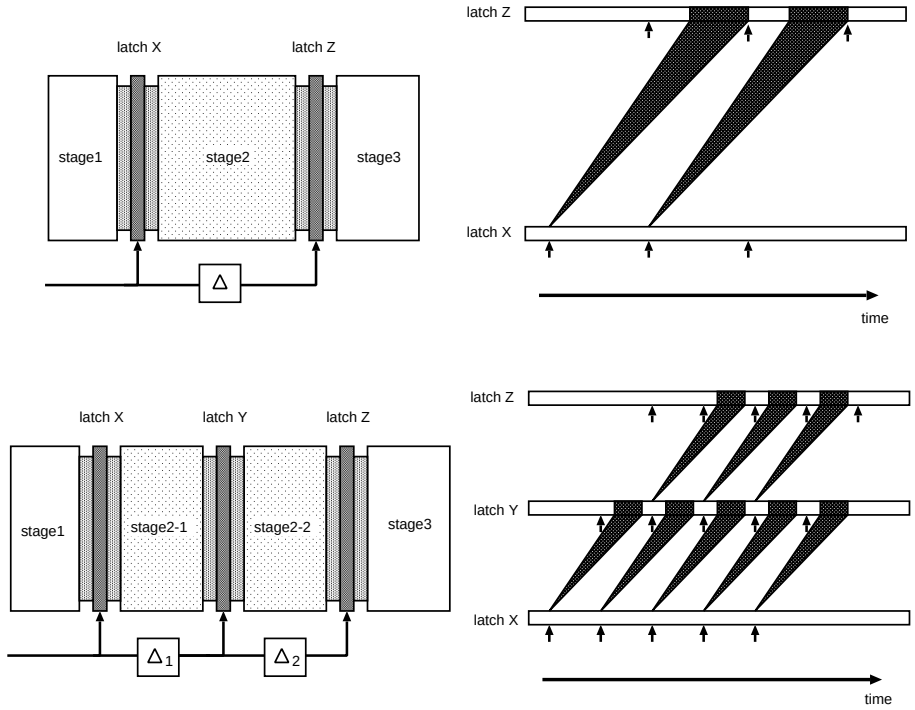


図 3.3: ウェーブパイプライン方式におけるステージ分割

行う。

まずステージの分割を行い、その後で遅延バッファを挿入することによって、先の分割によってステージ内の遅延差が小さくなっているため、挿入する遅延素子数を少量にすることができる。

## 3.4 ステージ分割による遅延差短縮アルゴリズム

ここでは、本研究で提案したステージ分割による遅延差短縮手法のアルゴリズムを示す。ステージ分割による遅延差短縮手法には、先に述べた遅延バッファ挿入による遅延差短縮手法も含んでいる。

### 3.4.1 ステージ分割による遅延差短縮アルゴリズム

ステージ分割による遅延差短縮手法を行うアルゴリズムを次に示す。

1. ウェーブパイプライン化する対象のステージのネットリストを得る。

2. あらかじめ遅延バッファを挿入するスペースを確保してレイアウトを行う。この場合、経験的に挿入される遅延バッファ数は対象となる回路の 3 倍程度である。

3. ステージ内の配線遅延を計算し、 そのステージの全てのパスにおける遅延を出力側から入力側に向けて計算する。

4. 計算した全てのパス遅延の中の最大遅延差が目標の $n$ 倍以上の時は、 ステージ分割を行い、 2. からの処理を繰り返す。最大遅延差が $n$ 倍より小さい時は、遅延バッファの挿入による遅延差短縮手法である 5. の処理へ行く。

5. 4. で得られたものの最小遅延パスに遅延バッファを挿入して再びレイアウトを行う。

6. 再びステージ内の入力から出力までの全てのパスの遅延を計算する。

7. 終了

このアルゴリズムにおいて、1. 2. 3. 5. は遅延バッファ挿入による遅延差短縮アルゴリズムですでに述べた。

4. において、ステージを分割する方法は次に示す。最大遅延差が目標の値であれば遅延バッファを挿入せずに 7. へ行く。$n$ の値は、設計目的によって変化する設定値である。詳しくは後述する。

6. で、得られた遅延差が目標の値であれば 7. へ行く。遅延差が目標の値ではないが、バススロットに空きがある場合は 5. に戻る。遅延差が目標の値でなく、バススロットを使い切った場合やまた遅延差は目標の値になったが空きスロットが多い場合などは、2. に戻る。

4. まではパイプラインステージの分割を行い、5. 6. は遅延バッファ挿入によってそれぞれ遅延差を短縮させている。

### 3.4.2 ステージの分割手法

ここではステージを分割する方法について述べる。

ステージ分割による遅延差短縮アルゴリズムにおいて、最大遅延差が目標の $n$ 倍である場合、ステージの分割を行う。

ステージの分割手法としては次の 2 つが考えられる。

- まず対象となるステージ内の回路素子段数を測定し、段数が半分となるところで分割する手法。

この手法では、ステージは 1 つのステージが常に 2 つに分割される。それ以上の分割は行われない。

遅延差はステージ分割を行うか否かの決定において、その条件として用いるのみである。よって、この手法では分割時に遅延差などはいっさい考慮しない。回路素子段数のみを考慮にいれて、分割した 2 つのステージ内の回路素子段数が同じになるように分割を行う。この手法の特徴を以下に示す。

- 分割されたステージは常に 2 つであるので、挿入するラッチ数が少なくてすむ。
- 分割後の各ステージの遅延差を考慮にいれないので、分割されたステージの遅延差が必ずしも短縮されない可能性がある。

- 測定している遅延差を調べながら、遅延差が目標の値となるところで分割を行う手法。

  この手法は回路素子段数そのものはほとんど考慮しない。遅延差を考慮にいれて、分割されて作られたステージは常に同じ遅延差であるように分割を行う。遅延差を調べながら分割を行うため、1 つのステージが複数個のステージに分割される可能性がある。この手法の特徴を以下に示す。

  - 遅延差を調べながら分割を行うため、分割されたステージは遅延バッファ挿入による遅延差短縮を行う必要がなくなる。
  - 分割後のステージ数が複数個である可能性があるため、挿入するラッチ数が多い。

本論文では、前者の方法でステージ分割を行う。

### ステージ分割アルゴリズム

ステージ分割手法は、対象となるステージ内の回路素子段数が半分となるところで分割する手法により行う。

ここで、基本的なステージ分割アルゴリズムを説明するために、分割する対象のステージ内の回路が図 3.4 であるものを例として用いる。

このステージを分割する。分割のやり方は以下のようにして行う。

1. まず出力端子 Z と Y からの全パスにおける回路素子段数を測定する。

   その結果を図 3.5 に示す。回路素子段数の測定は、出力端子から調べていく。

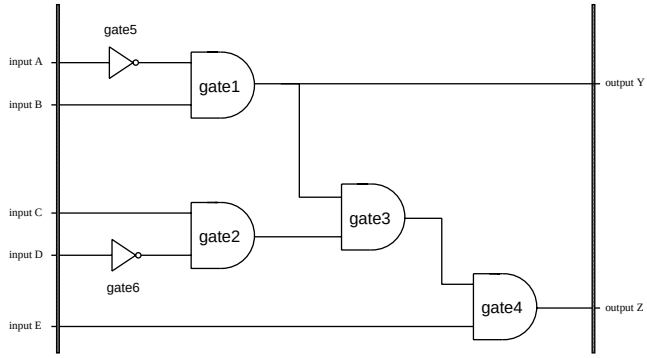


図 3.4: 分割を行う対象のステージの例

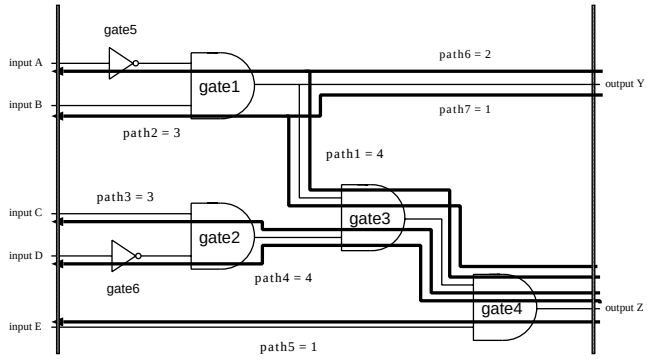


図 3.5: 全パスにおける回路素子段数の測定

2. 測定した結果より、ステージ内の回路素子段数が半分となるところに仮ラッチを挿入する。

   その結果を図 3.6 に示す。これより path1 と path7 での仮ラッチは latch a であり、同様に path2 と path3 での仮ラッチは latch c、path4 での仮ラッチは latch b、path5 での仮ラッチは latch d、そして path6 での仮ラッチは latch e となる。

   次に全パスにおいて最も回路素子段数の多いパスを見つけ、そのパスでの仮ラッチを分割基準ラッチとする。このステージにおいては、path1 と path4 の回路素子段数 4 が最も多いため、この 2 つのパスにおける仮ラッチ latch a と latch b を分割基準ラッチとする。

3. 分割基準ラッチを含んでいるパスにおいては、分割基準ラッチ以外の仮ラッチを省いていくことにより、余分な仮ラッチを整理する。

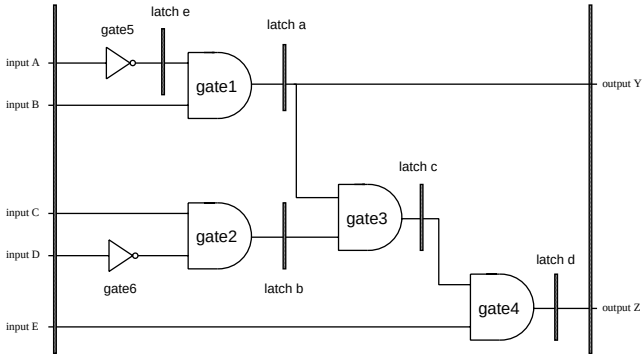


図 3.6: 全パスにおける仮ラッチの挿入

その結果を図3.7 に示す。これより、仮ラッチ latch c と latch e が整理される。latch d は path5 が分割基準ラッチを含んでいないので、ここでは整理されない。

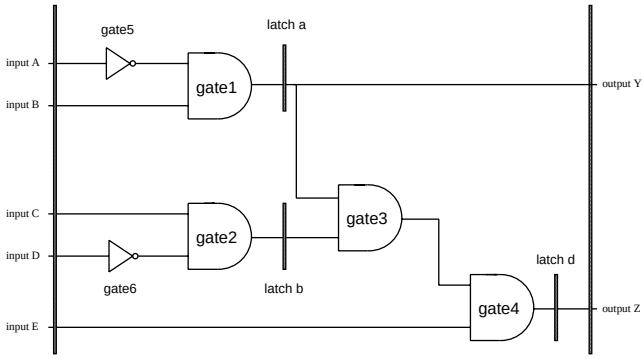


図 3.7: 仮ラッチの整理

4. 回路素子段数が最大でなく、かつ分割基準ラッチを含んでいないパスにおいては回路素子段数が最も多いパス中に含まれている素子が分割基準ラッチを含んでいないパス中に含まれていないかを調べる。

   含まれている場合は、出力端子から見て最後に含まれている素子の前に仮ラッチを挿入してやり、これまであった仮ラッチを整理する。この時、回路素子段数も考慮にいれる。

   含まれていない場合は、含まれていないパスをにおいて 1. からの処理を繰り返していく。

その結果を図 3.8 に示す。これより latch d は、gate4 の後ろであったものが gate4 の前に移動する。これで分割する点 latch a、latch b、latch d が決まる。

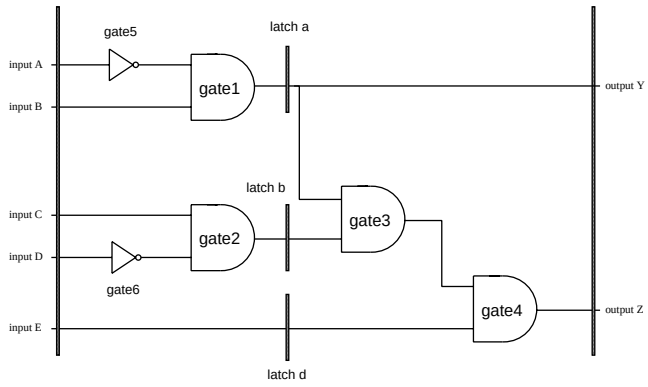


図 3.8: 分割基準ラッチを含まないパスの仮ラッチの整理

このようにして、ラッチの挿入場所を決める。

ステージ分割を行う (ラッチを挿入する) 場所については、ファンインやファンアウトを考慮して、より詳細な場所を決めてやる必要がある。

### 3.4.3 最適値の考慮

ステージ分割による遅延差短縮アルゴリズムにおける、最適値 $n$ の値について検討する。

ステージ分割による遅延差短縮アルゴリズムでの遅延バッファ挿入によるステージの最小遅延と最大遅延の均衡の処理において、遅延差は最初の $\frac{1}{2}$〜$\frac{1}{4}$ になる [8] ことより、 最大遅延差が目標の 2〜4 倍であるならば、 目標の値に均衡させることが可能である。

ここで n=2 の場合は、 細かくステージを分割することになるのでラッチ数が多くなってしまうが、分割したステージの遅延差は小さいために挿入するバッファ数は少量ですむ。逆に n=4 のときは、 ステージの分割数を抑えることができるためにラッチ数は少なくてすむが、 ウェーブパイプライン化により遅延バッファ数が膨大になり、 それによってチップ面積と消費電力が増大してしまう。これらのトレードオフを考慮し、 最適な n の値を求めることも重要になる。

この最適値 $n$ の値については、設計目標やそれぞれのステージにおける最大遅延差の結果によって最適値が変化する。それぞれの条件において、あてはまる最適値 $n$ の値を設定することになる。

## 3.5　パスの遅延計算における計算量

ステージ分割による遅延差短縮手法の目的でも述べたように、ステージ分割による遅延差短縮手法を用いることでプロセッサの動作速度の向上と共に設計時間の短縮も目的に含まれている。そのため、ここで遅延バッファ挿入のみによる遅延差短縮手法とステージ分割による遅延差短縮手法のそれぞれにおける設計時間に直接関係する計算量について見積もる。

ステージ内のパスの遅延計算は、出力側から入力側に向けて1つ1つ全てを計算しなければならない。 よってステージ内の回路が大きくなると計算量も大きくなり、計算時間の増大につながる。

パイプラインステージを分割した場合の計算量と分割しない場合の計算量の違いを考える。

図 3.9 のようなステージの回路において、入出力が $m$ ビット、回路段数が $k$ 段とする。

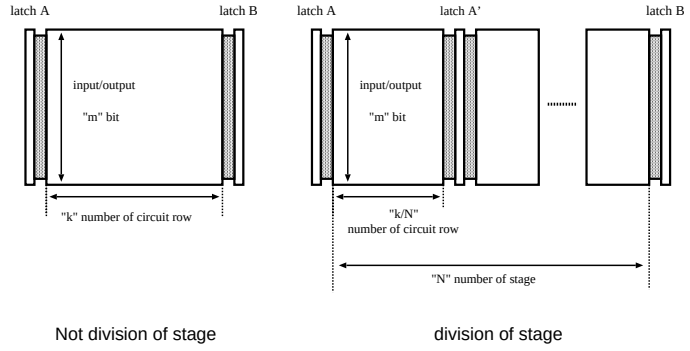


図 3.9: パスの遅延計算における非分割と分割のモデル

この時、ステージを分割しない場合の総遅延計算量 $S_0$ は、

$$S_0 = m^k \tag{3.1}$$

となる。

次にステージを $N$ 個に分割した場合の総遅延計算量 $S_d$ は、

$$S_d = Nm^{\frac{k}{N}} \tag{3.2}$$

となる。

3.1、 3.2 より、ステージ分割による遅延差短縮手法の遅延計算量の方が、ステージ分割を行わない場合に比べ遥かに少なくなることが分かる。これよりパイプラインステージ

を分割することになっても、パスの遅延計算量を抑えることができるので、ステージ分割における設計時間の増大を防ぐことができる。

# 第 4章

# 設計

## 4.1 ウェーブパイプライン化する対象のプロセッサ

ウェーブパイプライン化を行う対象として以下のような簡単なパイプラインプロセッサを用いる。

- プロセッサのパイプラインステージ数は、IF(Instruction Fetch:命令フェッチ)、ID(Instruction Decode：命令デコード)、EXE(EXEcution：実行)、MEM(MEMory Access：メモリアクセス)、WB(Write Back：ライトバック) の 5 段である。
- アドレスバスとデータバスは 16 bit である。
- キャッシュメモリは命令キャッシュとデータキャッシュに分けない (ハーバードアーキテクチャではない)。
- RISC アーキテクチャの中のロード-ストアアーキテクチャである。
- 例外処理機構はない。

このプロセッサ機構を図 4.1 に示す。

あらかじめウェーブパイプライン化を行わない (通常のパイプライン) 場合の各ステージの遅延時間や面積を測定した。その結果を表 4.1 に示し、各ステージの遅延伝搬グラフを図 4.2 に示す。

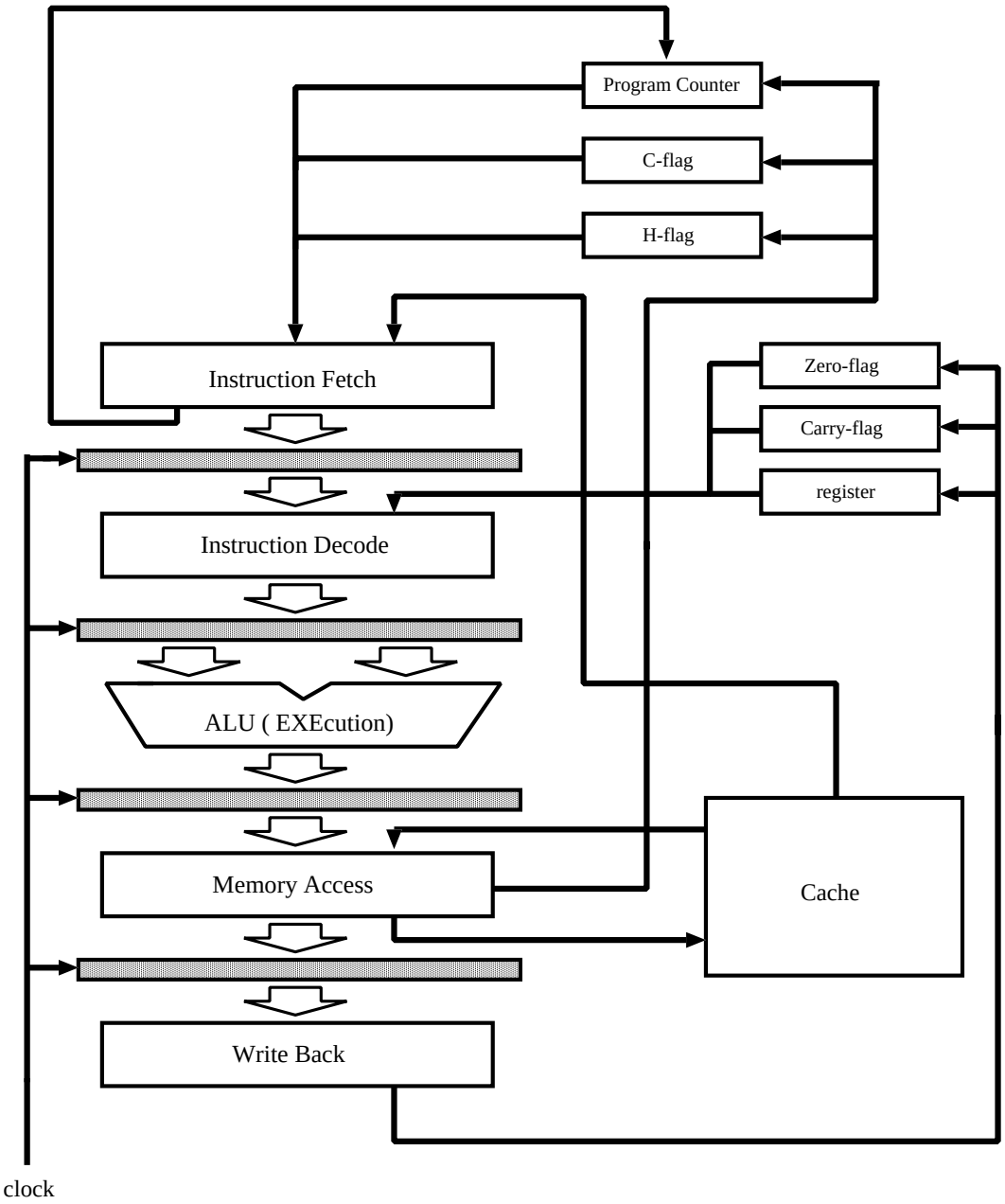


図 4.1: ウェーブパイプライン化する対象のプロセッサ機構

表 4.1: プロセッサの各ステージにおける遅延 (プロセスルール 0.10 $\mu m$)

| ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|
| IF | 129.36 | 13.72 | 115.64 | 112.72 |
| ID | 26.63 | 22.65 | 3.98 | 111.36 |
| EXE | 418.93 | 50.04 | 368.90 | 535.70 |
| MEM | 151.25 | 29.43 | 121.82 | 245.13 |
| WB | 26.63 | 22.65 | 3.98 | 78.88 |

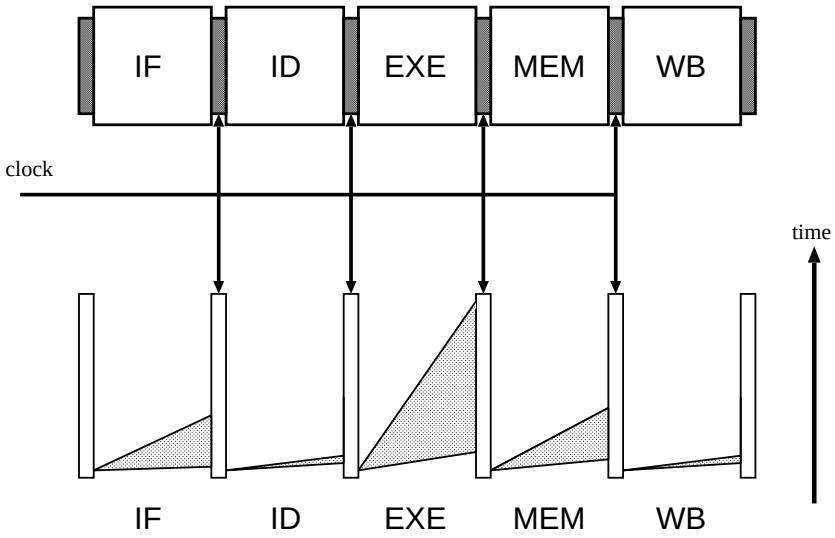


図 4.2: 対象となるプロセッサの構成と遅延図

この結果より、このプロセッサでの性能を決めるのは、最大遅延の最も大きな EXE ステージであることがわかる。また、ウェーブパイプライン化を行う場合、その性能を決定する遅延差においても EXE ステージが最も大きい。よって、まず最初の対象はこの EXE ステージとなるので、この EXE ステージに遅延バッファ挿入による遅延差短縮手法でウェーブパイプライン化を行い、次に本研究で提案したステージ分割による遅延差短縮手法でウェーブパイプライン化を行う。目標の遅延差は、次に遅延差が大きい MEM ステージの遅延差となる。その後は短縮された遅延差により対象を決めていく。

## 4.2 遅延

本論文では、プロセスルール $0.10\mu m$ を視野にいれているため、遅延時間の計算において必要となる遅延モデルについて述べる。

### 4.2.1 遅延モデル

#### 素子モデルと配線モデル

ステージ内にウェーブ化を行うに当たって、遅延計算等で必要となるステージ内の素子と配線を含めた遅延モデルを図 4.3 に示す。

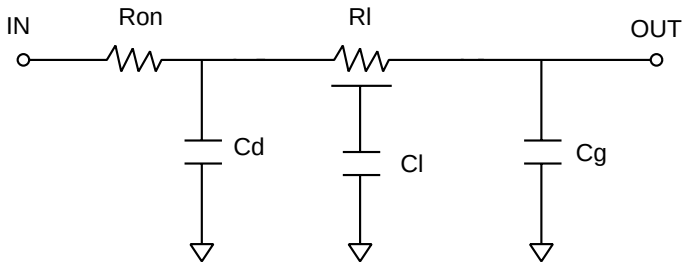


図 4.3: 遅延モデル

ここで、$Ron$ はトランジスタのオン抵抗、$Cd$ は拡散容量、$Cg$ は入力端子容量を表す。また、$Rl$ は配線抵抗、$Cl$ は配線容量を表す。

先の遅延モデルの等価回路を図 4.4 に示す。

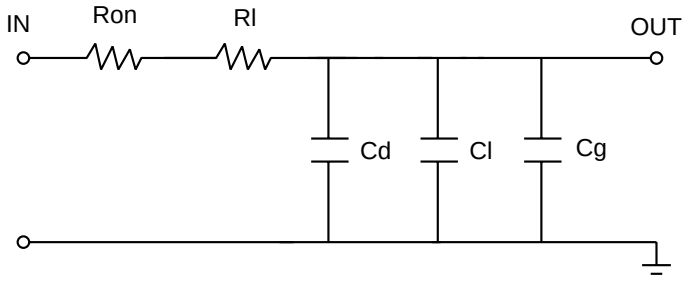


図 4.4: 簡単化した等価回路

図 4.4 の等価回路より、遅延情報の測定に必要となる素子と配線の遅延パラメータを求めた。

### 4.2.2 遅延パラメータ

プロセスルール 0.10 $\mu m$ の時の素子と配線の遅延パラメータは表4.2のように求まった。

#### 素子の遅延パラメータ

表 4.2: 素子の遅延パラメータ (プロセスルール 0.10 $\mu m$)

| 素子名 | 固定成分 ($t_{int}$)[ps] | 入力端子容量 (Cg)[fF] | 拡散容量 (Cd)[fF] | オン抵抗 (Ron)[kΩ] | 面積 [$\mu m^2$] |
|---|---|---|---|---|---|
| inverter | 0.88 | 0.78 | 0.095 | 4.88 | 2.09 |
| delay buffer | 1.75 | 0.78 | 0.190 | 9.75 | 4.18 |
| NAND | 1.79 | 0.70 | 0.103 | 5.30 | 2.55 |
| NOR | 3.87 | 0.70 | 0.105 | 5.39 | 2.55 |
| EX-NOR | 1.79 | 1.20 | 0.103 | 5.30 | 7.42 |
| EX-OR | 1.79 | 1.20 | 0.103 | 5.30 | 7.42 |

#### 配線の遅延パラメータ

配線における抵抗と容量を求める式は、配線長 $L[\mu m]$、配線幅 $w[\mu m]$、配線の厚さ $t[\mu m]$ とすると

- 配線抵抗 ($Rl$)

  まず配線の体積抵抗率 $\rho[\Omega\mu m]$ よりシート抵抗値 $\rho_s[\Omega]$ を求める。

  $$\rho_s = \frac{\rho}{t} \tag{4.1}$$

  配線長 L[$\mu m$] の表記は次のようにする。

  $$L = \text{プロセスルール} \times \text{配線長}\,[grid] \tag{4.2}$$

  1grid は 0.10[$\mu m$] である。よって配線抵抗 Rl[kΩ] は

  $$Rl = \rho_s \frac{L}{w} \tag{4.3}$$

  となる。

- 配線容量 ($Cl$)

  まず酸化膜厚 $T_{ox}$[m]、真空の誘電率 $\varepsilon_o[F/m^2]$、$SiO_2$ の比誘電率 $\varepsilon_{ox}[F/m^2]$ より単位面積あたりの酸化膜容量 Co$[F/m^2]$ を求める。

$$Co = \frac{\varepsilon_{ox}\varepsilon_o}{T_{ox}} \tag{4.4}$$

  配線の面積 S$[\mu m^2]$ は、

$$S = L \times w \tag{4.5}$$

  である。以上より配線容量 Cl[fF] は

$$Cl = CoS \tag{4.6}$$

となる。

配線の遅延パラメータは表 4.3 となる。

表 4.3: 配線の遅延パラメータ (プロセスルール 0.10 $\mu m$)

| | |
|---|---|
| 配線抵抗 $Rl$[kΩ] | 0.0009 × grid 数 |
| 配線容量 $Cl$[fF] | 0.00276 × grid 数 |

## 4.3 レイアウト

ステージ分割による遅延差短縮手法において、レイアウトによる遅延時間の影響は大きくなる。ここでは、レイアウト方法について述べる。

### 4.3.1 レイアウト方法

レイアウトは、初期レイアウトとレイアウトの改善の 2 段階で行われる。レイアウトの評価としては、仮想配線長の和とする。仮想配線長とは、配置された素子間の配線の長さを仮想的に評価したものである。この仮想配線長の総和を最小にする配置を求める方法である。

初期レイアウトには、構成的方法であるペアリンキング法 (pair-linking method) を用いる。ペアリンキング法は、すでに配置済みの素子と接続の最も多い素子をその近くに配置する方法である。

レイアウトの改善は、初期レイアウトで与えられた配置をさらに改善する方法である。これにはペア交換法 (pairwise interchange method) を用いる。この方法は、2 つの素子の位置を交換し、総仮想配線長が改良されていれば、古い配置と置き換える方法である。

レイアウト方法は以下の手順により行う。

1. 第 1 段階レイアウト

   第 1 段階レイアウトを行う前に仮想配線長からバッファ容量を求める。そして出力端子からつながる最初の素子を配列化し、次につながる素子を配列化していく。この処理を繰り返すことにより素子の接続順に配列化される。

   この配列に仮想配線長で求めたバッファスロットをバランス良く予約配置していく。そして各素子の配線長が長い場合、ペア交換法によりレイアウトの改善を行う。ここで仮想配線長は、配線長が回路の一辺の $\frac{1}{3}$ とした。

2. 遅延バッファを挿入する。

3. 遅延バッファのレイアウト

   第一段階レイアウトから配線長が、遅延バッファ挿入による遅延差短縮手法からバッファを挿入する場所が決定される。素子の後列のバッファスロットで最も近いものにバッファをレイアウトする。ここでもペア交換法によりレイアウトの改善を行う。

遅延バッファの挿入によるレイアウトは、2. と 3. を繰り返す。

# 第 5 章

# 結果

プロセスルール 0.10 $\mu m$ における対象プロセッサの遅延を表 5.1 に示す。

表 5.1: プロセッサの各ステージにおける遅延 (プロセスルール 0.10 $\mu m$)

| ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|
| IF | 129.36 | 13.72 | 115.64 | 112.72 |
| ID | 26.63 | 22.65 | 3.98 | 111.36 |
| EXE | 418.93 | 50.04 | 368.90 | 535.70 |
| MEM | 151.25 | 29.43 | 121.82 | 245.13 |
| WB | 26.63 | 22.65 | 3.98 | 78.88 |

このプロセッサにウェーブパイプライン化を行う。

対象となるステージは、遅延時間と遅延差が最大となる EXE ステージであり、目標遅延差は MEM ステージの 121.82 ピコ秒である。

## 5.1 実行ステージの遅延差

各遅延差短縮手法を行う前の出力ピンの遅延差を図 5.1 に示す。

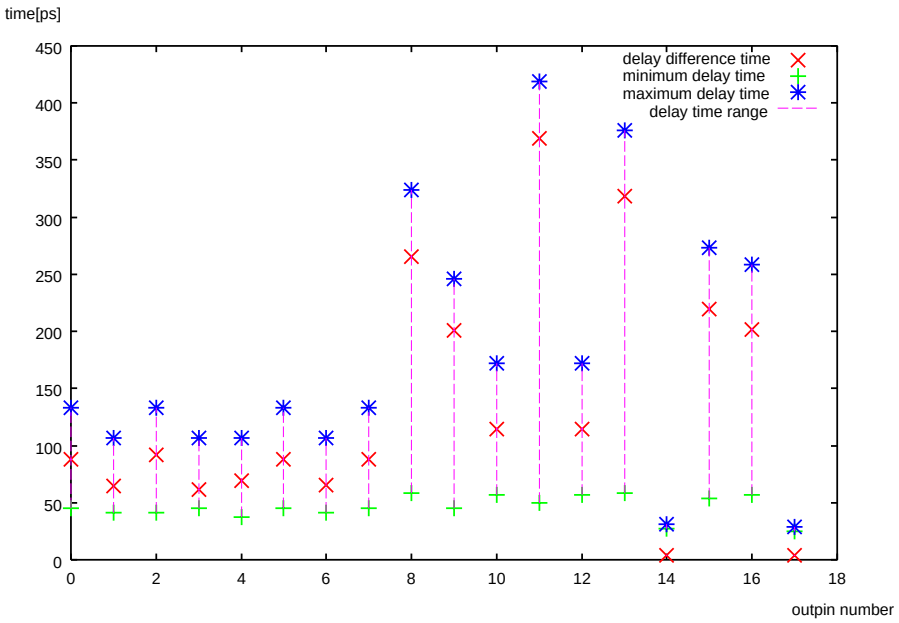


図 5.1: ウェーブパイプライン化を行う前の各出力ピンにおける遅延差

## 5.2 バッファ挿入のみの遅延差短縮

対象となる EXE ステージの遅延差は 368.9 ピコ秒であり、目標となる遅延差は MEM ステージの遅延差である 121.82 ピコ秒である。現在の遅延差が目標の 3 倍程度であるため、最適値 $n$ が 4 である場合は、遅延バッファ挿入のみによる遅延差短縮手法を用いることになる。

この EXE ステージに遅延バッファ挿入のみによる遅延差短縮手法でウェーブパイプライン化を行った。

その結果を表 5.2 に示す。挿入する遅延バッファの数と遅延差との関係を図 5.2 に示す。この遅延差短縮を行った後の各出力ピンにおける遅延差を図 5.3 に示す。

表 5.2: 実行ステージにおけるバッファ挿入による遅延 (プロセスルール 0.10 $\mu m$)

| | ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|---|
| 基本 | EXE | 418.93 | 50.04 | 368.90 | 535.70 |
| 遅延バッファ挿入のみ | EXE | 479.55 | 315.02 | 164.53 | 1902.56 |

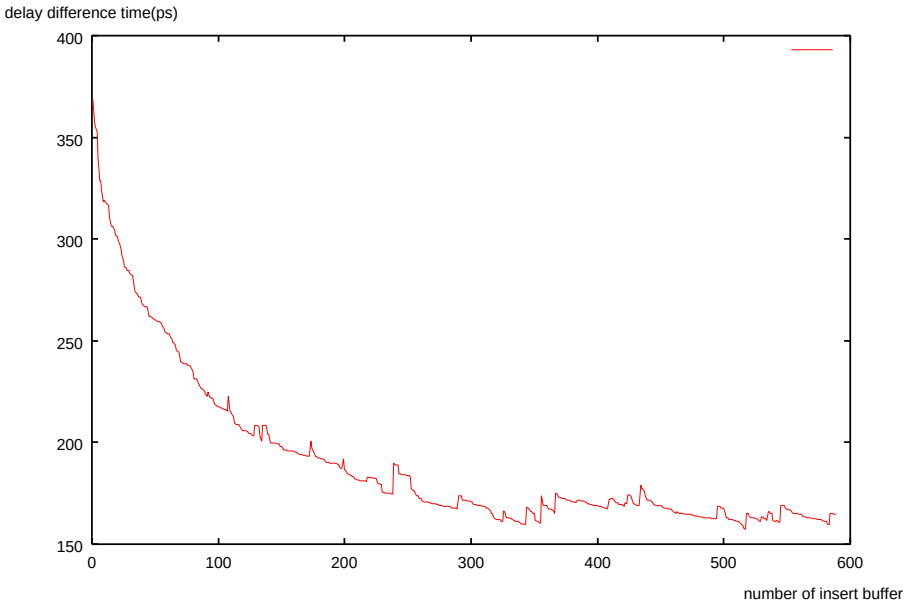


図 5.2: 挿入するバッファ数と遅延差の関係

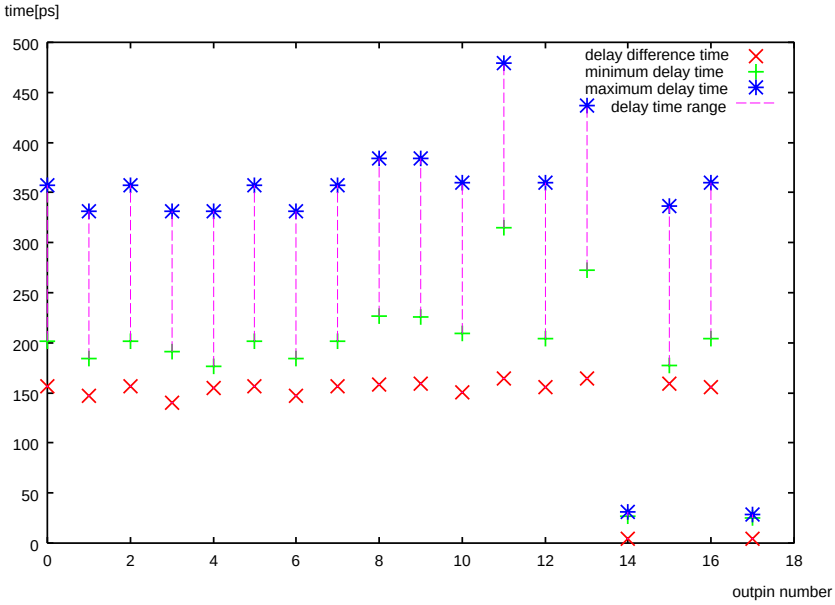


図 5.3: 各出力ピンにおける遅延の関係

## 5.3 ステージ分割による遅延差短縮

対象となる EXE ステージの遅延差は 368.9 ピコ秒であり、目標となる遅延差は MEM ステージの遅延差である 121.82 ピコ秒である。現在の遅延差が目標の 3 倍程度であるため、ステージ分割による遅延差短縮手法を用いる場合、最適値 $n$ は 2 か 3 となる。

ステージ分割による遅延差短縮手法を用いた場合のステージ分割後の結果を表 5.3 に示す。

分割した 2 つのステージにおいて、その前段を分割ステージ 1、後段を分割ステージ 2 とする。

表 5.3: 実行ステージにおける分割遅延 (プロセスルール 0.10 $\mu m$)

| | ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|---|
| 基本 | EXE | 418.93 | 50.04 | 368.90 | 535.70 |
| 遅延バッファ挿入のみ | EXE | 479.55 | 315.02 | 164.53 | 1902.56 |
| ステージ分割のみ | EXE1 | 216.15 | 31.29 | 184.86 | 202.01 |
| | EXE2 | 186.64 | 42.05 | 144.59 | 333.69 |

この結果より、分割ステージ 1、分割ステージ 2 ともに設計目標である 121.8 ピコ秒と比べて 2 倍以下であるため、更なる分割の必要はない。

このステージ分割後の 2 つのステージでの各出力ピンにおける遅延差を図 5.4 と図 5.5 に示す。

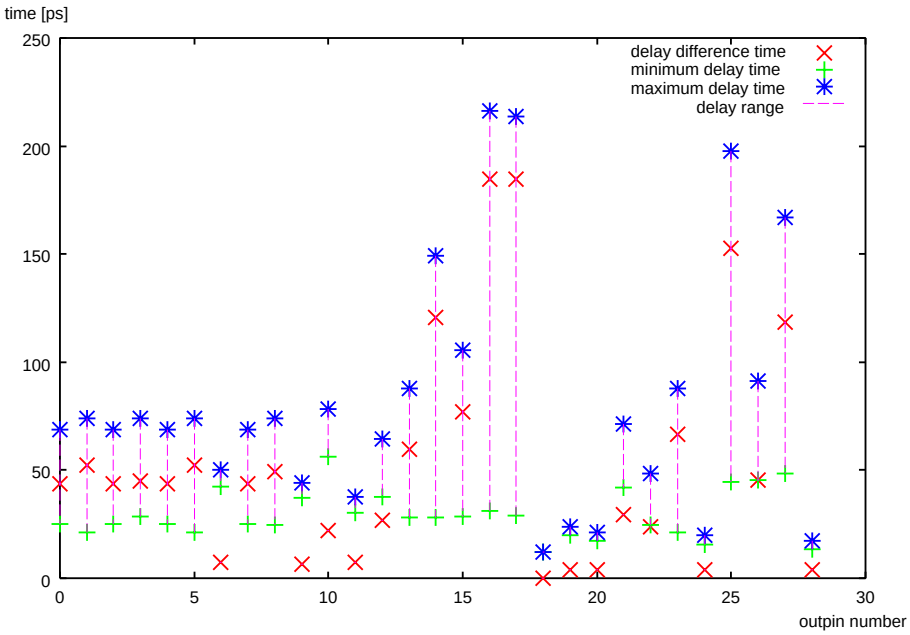


図 5.4: 分割ステージ 1 の各出力ピンにおける遅延の関係

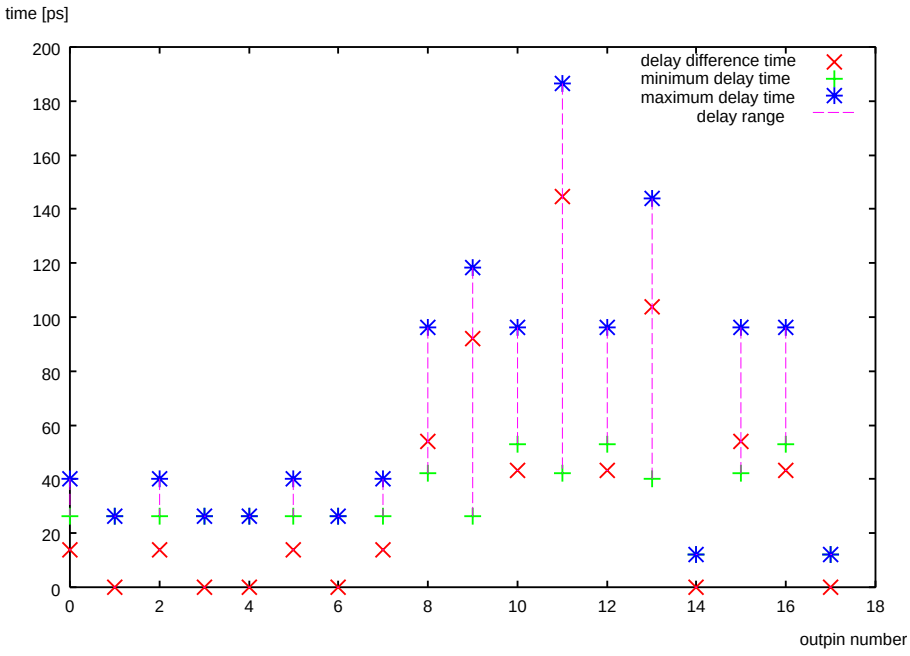


図 5.5: 分割ステージ 2 の各出力ピンにおける遅延の関係

次にステージ分割を行った後の 2 つのステージに遅延バッファ挿入による遅延差短縮を行う。

### 5.3.1 目標値均衡を目的とする場合

ここでは、遅延差があらかじめ設定した目標以下になることを目的として遅延差短縮を行う。目標となる遅延差は MEM ステージの遅延差である 121.8 ピコ秒である。

分割したステージ 1 と 2 において遅延差短縮を行った結果を表 5.4 に示す。

この場合では、遅延差 < 目標遅延差になれば遅延バッファ挿入による遅延差短縮を終了する。

表 5.4: 分割した実行ステージのそれぞれの遅延 (プロセスルール 0.10 $\mu m$)

| | ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|---|
| 分割ステージ 1 | EXE1 | 216.15 | 96.70 | 119.45 | 256.35 |
| 分割ステージ 2 | EXE2 | 186.64 | 67.15 | 119.48 | 354.59 |

分割ステージ 1 において、挿入するバッファ数と遅延差との関係を図 5.6 に示す。

同様にして、分割ステージ 2 における挿入バッファ数と遅延差との関係を図 5.7 に示す。

ここで 2 つのステージでの各出力ピンにおける遅延差を図 5.8 と図 5.9 に示す。

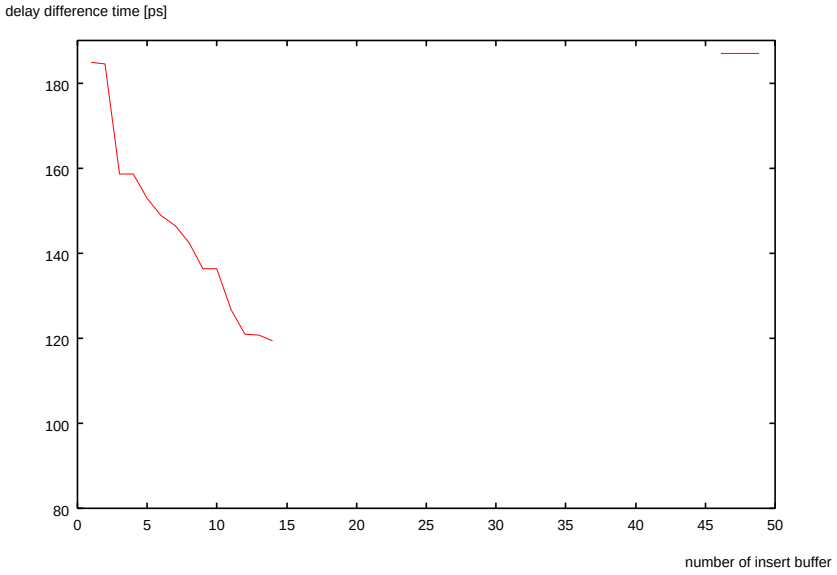


図 5.6: 分割ステージ 1 における挿入バッファ数と遅延差との関係

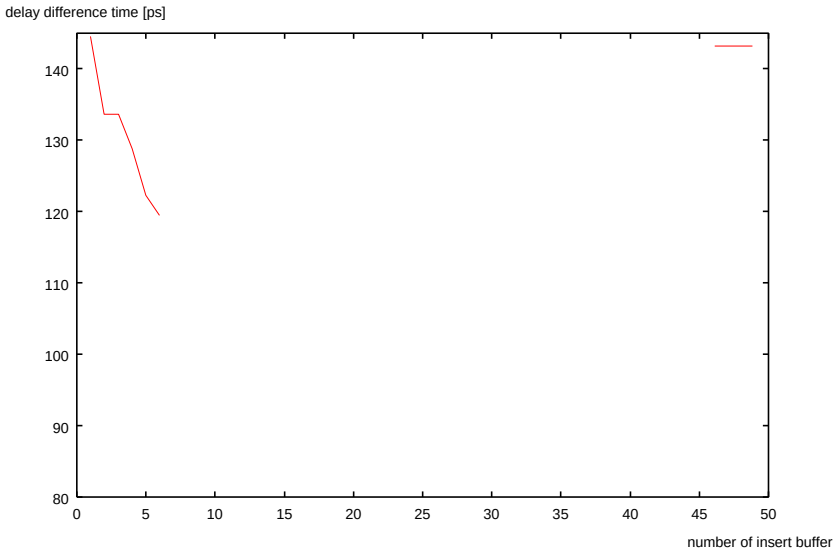


図 5.7: 分割ステージ 2 における挿入バッファ数と遅延差との関係

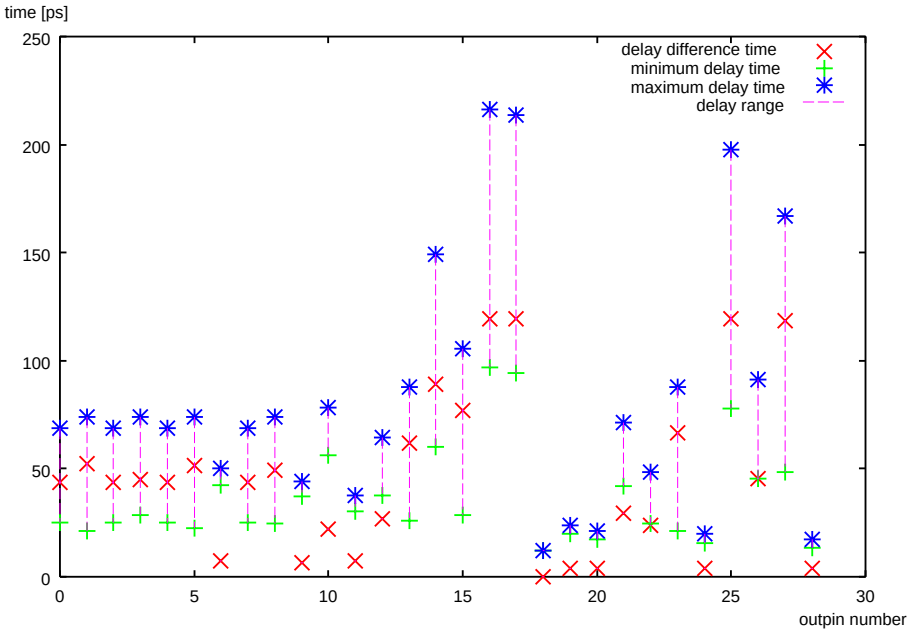


図 5.8: 分割ステージ 1 の各出力ピンにおける遅延の関係

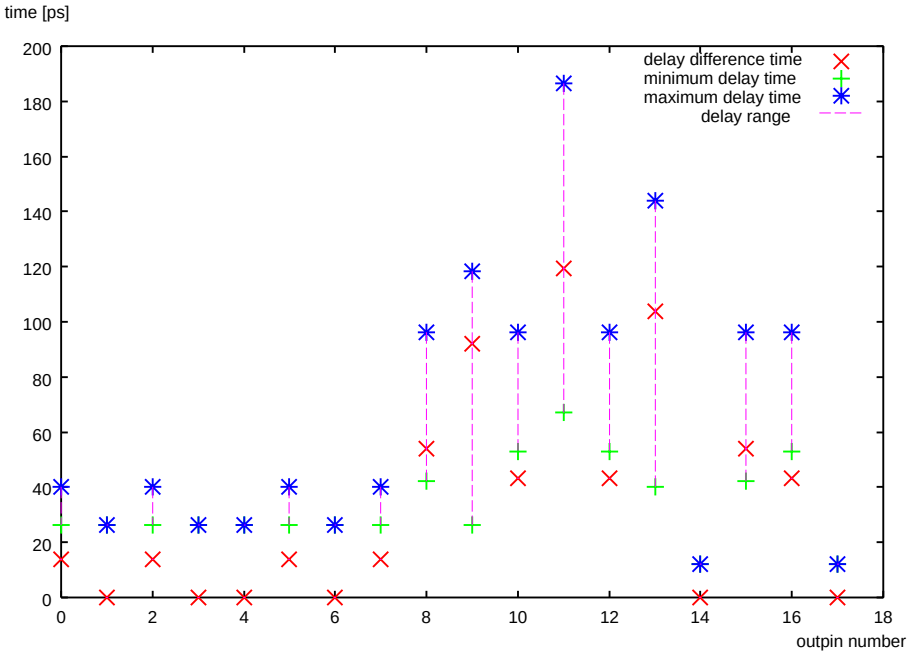


図 5.9: 分割ステージ 2 の各出力ピンにおける遅延の関係

### 5.3.2 高性能化を目的とする場合

ここでは分割したそれぞれのステージがどこまで遅延差短縮できるか (高性能化できるか) を目的として、遅延差短縮を行う。

分割したステージ 1 と 2 において遅延差短縮を行った結果を表 5.5 に示す。

表 5.5: 分割した実行ステージのそれぞれの遅延 (プロセスルール 0.10 $\mu m$)

| | ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|---|
| 分割ステージ 1 | EXE1 | 217.14 | 195.33 | 21.81 | 607.47 |
| 分割ステージ 2 | EXE2 | 186.64 | 157.01 | 29.63 | 651.37 |

分割ステージ 1 において、挿入するバッファ数と遅延差との関係を図 5.10 に、分割ステージ 2 における挿入バッファ数と遅延差との関係を図 5.11 にそれぞれ示す。

また、2 つのステージでの各出力ピンにおける遅延差を図 5.12 と図 5.13 に示す。

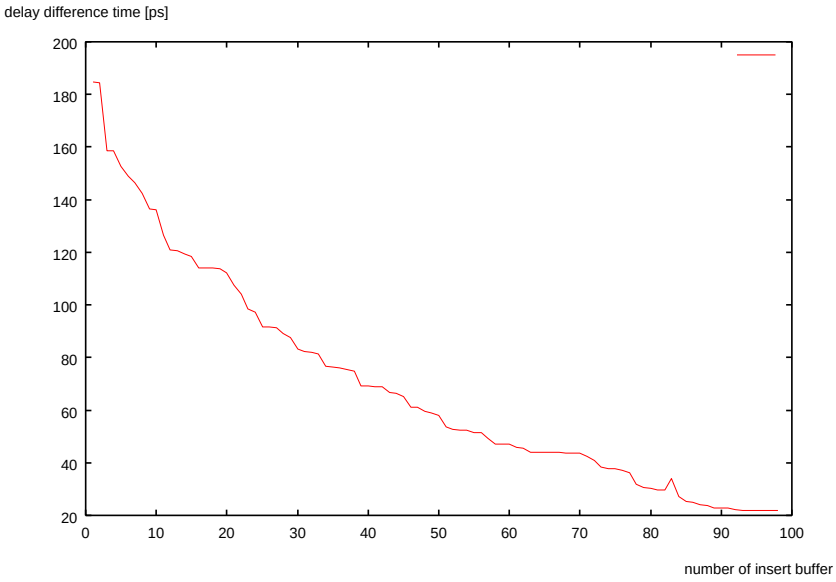


図 5.10: 分割ステージ 1 における挿入バッファ数と遅延差との関係

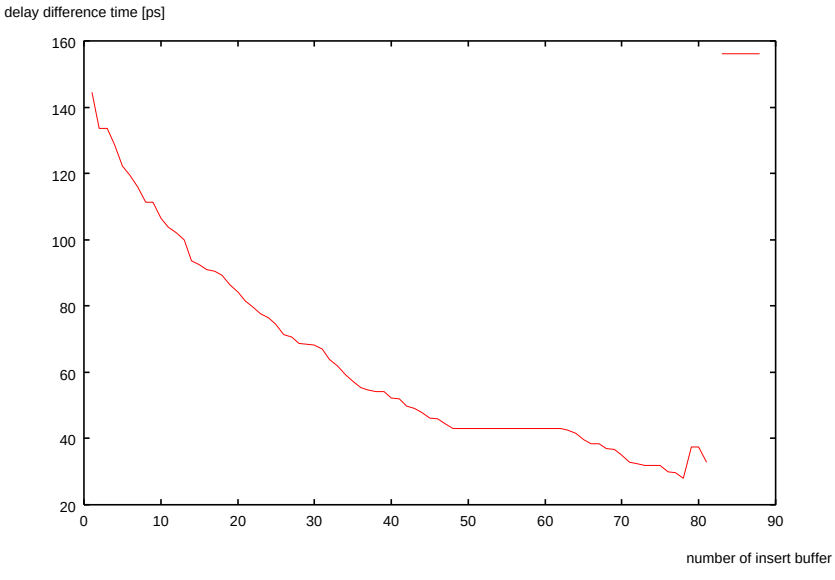


図 5.11: 分割ステージ 2 における挿入バッファ数と遅延差との関係

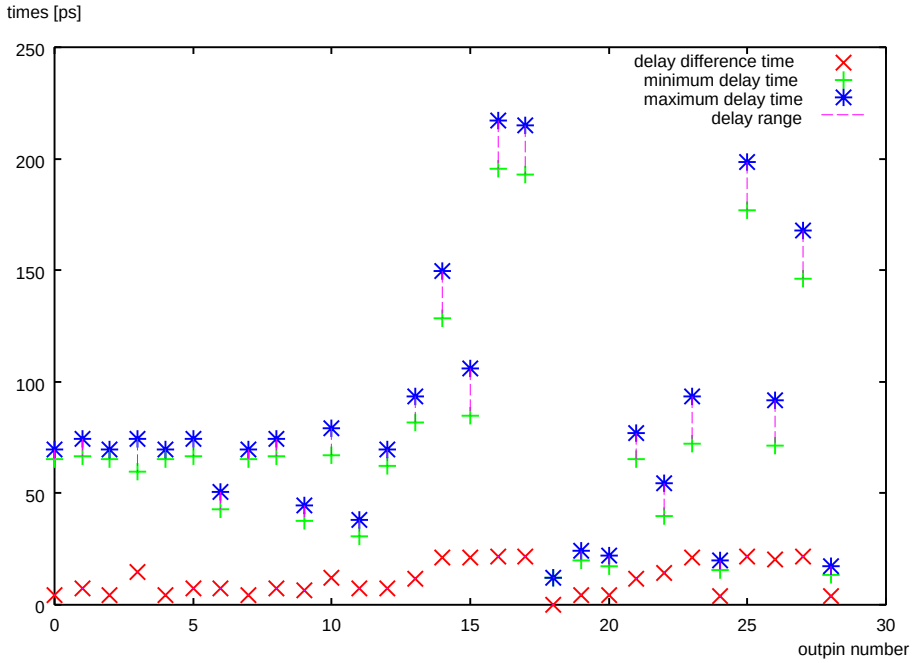


図 5.12: 分割ステージ 1 の各出力ピンにおける遅延の関係

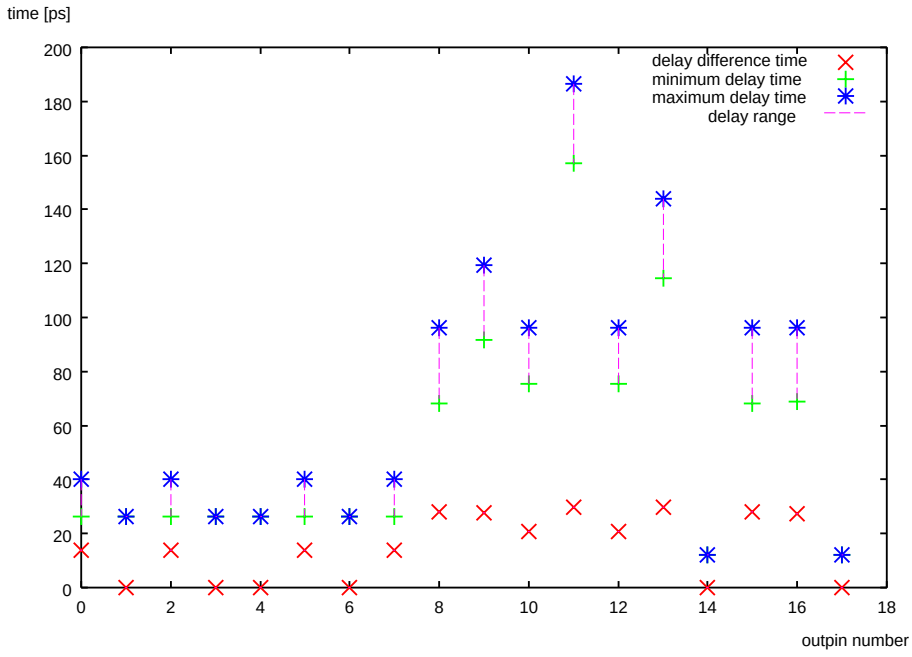


図 5.13: 分割ステージ 2 の各出力ピンにおける遅延の関係

# 第 6 章

# 考察

第 5 章で得られた結果より、提案したステージ分割による遅延差短縮手法の有効性を評価しながら考察を行う。

## 6.1 性能による考察

結果を実行ステージのみでなく、プロセッサ全体からまとめる。まず何も行わない基本状態における遅延と面積の結果を表 6.1、最適値 $n = 4$ である場合の遅延バッファ挿入のみによる遅延差短縮手法を表 6.2、最適値 $n = 2, 3$ である場合のステージ分割による遅延差短縮手法を表 6.3 にまとめる。

そしてそれぞれの手法における計算量をまとめたものを表 6.4、消費電力をまとめたものを表 6.5 に示す。

この結果より、それぞれの手法における各性能の比較を行ったものを表 6.6、消費電力についての比較を行ったものを表 6.7 に示す。

表 6.1: プロセッサの各ステージにおける遅延 (基本状態)

| ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|
| IF | 129.36 | 13.72 | 115.64 | 112.72 |
| ID | 26.63 | 22.65 | 3.98 | 111.36 |
| EXE | 418.93 | 50.04 | 368.90 | 535.70 |
| MEM | 151.25 | 29.43 | 121.82 | 245.13 |
| WB | 26.63 | 22.65 | 3.98 | 78.88 |
| その他 | | | | 1158 |
| 総合 | | | | 2241.79 |

表 6.2: プロセッサの各ステージにおける遅延 (遅延バッファ挿入による遅延差短縮手法)

| ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|
| IF | 129.36 | 13.72 | 115.64 | 112.72 |
| ID | 26.63 | 22.65 | 3.98 | 111.36 |
| EXE | 479.55 | 315.02 | 164.53 | 1902.56 |
| MEM | 151.25 | 29.43 | 121.82 | 245.13 |
| WB | 26.63 | 22.65 | 3.98 | 78.88 |
| その他 | | | | 1158 |
| 総合 | | | | 3608.65 |

表 6.3: プロセッサの各ステージにおける遅延 (目標値均衡でのステージ分割による遅延差短縮手法)

| ステージ名 | 最大遅延時間 [ps] | 最小遅延時間 [ps] | 遅延差 [ps] | 面積 [$\mu m^2$] |
|---|---|---|---|---|
| IF | 129.36 | 13.72 | 115.64 | 112.72 |
| ID | 26.63 | 22.65 | 3.98 | 111.36 |
| EXE1 | 216.15 | 96.70 | 119.45 | 256.35 |
| EXE2 | 186.64 | 67.15 | 119.48 | 354.59 |
| MEM | 151.25 | 29.43 | 121.82 | 245.13 |
| WB | 26.63 | 22.65 | 3.98 | 78.88 |
| その他 | | | | 1391.68 |
| 総合 | | | | 2550.71 |

表 6.4: それぞれの手法におけるパス計算量

| | 総合ステージ数 | 実行ステージ数 | 実行ステージの素子数 | 全パスの計算量 |
|---|---|---|---|---|
| 基本状態 | 5 | 1 | 188 | 2512 |
| 遅延バッファ挿入のみ | 5 | 1 | 508 | 4460 |
| ステージ分割のみ | 6 | 2 | 188 | 1322 |
| ステージ分割後にバッファ挿入 | 6 | 2 | 208 | 1382 |

表 6.5: それぞれの手法における消費電力

| | 総合ステージ数 | 実行ステージの素子数 | 実行ステージのみの消費電力 $[\mu W]$ | 総合消費電力 $[\mu W]$ | 1 MHz 当りの消費電力 $[nW]$ |
|---|---|---|---|---|---|
| 基本状態 | 5 | 188 | 387.80 | 2064.42 | 867.40 |
| 遅延バッファ挿入のみ | 5 | 508 | 4386.35 | 8655.12 | 1428.24 |
| ステージ分割のみ | 6 | 188 | 754.41 | 4502.79 | 972.52 |
| ステージ分割後にバッファ挿入 | 6 | 208 | 1625.52 | 8272.20 | 1007.58 |

表 6.6: 遅延結果からの比較

| | 基本状態との性能比 | 実行ステージのみの面積比 | 総合面積比 | 計算量比 |
|---|---|---|---|---|
| 基本状態 | 1.00 | 1.00 | 1.00 | 1.00 |
| バッファ挿入のみ | 2.55 | 3.56 | 1.61 | 1.78 |
| ステージ分割のみ | 1.95 | 1.00 | 1.10 | 0.53 |
| 分割後のバッファ挿入 | 3.45 | 1.14 | 1.14 | 0.55 |

表 6.7: 消費電力の比較

| | 基本状態との性能比 | 実行ステージのみの消費電力比 | 総合消費電力比 | 1 MHz 当りの消費電力比 |
|---|---|---|---|---|
| 基本状態 | 1.00 | 1.00 | 1.00 | 1.00 |
| バッファ挿入のみ | 2.55 | 11.31 | 4.19 | 1.65 |
| ステージ分割のみ | 1.95 | 1.95 | 2.18 | 1.12 |
| 分割後のバッファ挿入 | 3.45 | 4.19 | 4.01 | 1.16 |

遅延バッファ挿入のみによる遅延差短縮手法における性能は、基本状態と比べ 2.55 倍の向上が図れた。面積については、挿入した遅延バッファ量が 320 個あったために実行ステージのみで 3.56 倍となっており、全体では 1.61 倍となっている。消費電力では、総合消費電力比が 4.19 倍、1 MHz 当りの消費電力比は 1.65 倍となった。総合ステージ数は基本状態と同じである。また、遅延バッファ挿入のみによる遅延差短縮手法では、遅延差が目標遅延差である 121.82 ピコ秒以下にすることができなかった。

次に、提案したステージ分割による遅延差短縮手法における性能は、基本状態と比べ 3.45 倍の向上が図れた。挿入した遅延バッファ量は分割ステージ 1 で 14 個、分割ステージ 2 で 6 個、合計で 20 個の遅延バッファを挿入した。この挿入した遅延バッファによって面積は、実行ステージのみで 1.14 倍、これにステージ分割による 1 つのラッチ面積を含めた全体でも 1.14 倍となっている。消費電力では、総合消費電力比 4.01 倍、1 MHz 当りの消費電力比 1.16 倍となっている。総合ステージ数はステージ分割により実行ステージが 1 つ増え、6 ステージである。これより遅延バッファ挿入のみによる遅延差短縮手法よりも性能に対する面積向上率は低く、消費電力増加率も低いことがわかる。

計算時間については 5.3.1 の目標均衡を目的にした場合の結果より、遅延バッファの挿入量が少ないため明らかに計算時間が少ないことが分かる。実際に分割した各ステージにおいての計算時間は非常に小さかった。よって分割したステージ両方の計算時間の和においても非常に小さい。

計算量でもそれは明らかである。基本状態で全パスの遅延計算を行った場合と比較する

と、遅延バッファ挿入のみによる遅延差短縮手法では 1.78 倍になっているが、ステージ分割による遅延差短縮手法では、0.55 倍と計算量が少なくなっている。

ステージ分割の時間を含めればやや長くなるが、それでも遅延バッファ挿入のみによる遅延差短縮手法に比べれば遥かに短い時間で設計が行える。

遅延バッファの挿入量が少ないことは、これまでのウェーブパイプライン化手法において問題であった消費電力の増加を抑えることにも繋がる。

次に遅延バッファ挿入のみによる遅延差短縮手法と提案したステージ分割による遅延差短縮手法を比較する。性能においては、遅延バッファ挿入のみによる遅延差短縮手法よりステージ分割による遅延差短縮手法の方が 1.35 倍高い性能である。面積においては、実行ステージのみで 0.32 倍、全体では 0.71 倍である。消費電力においては、実行ステージのみの消費電力比が 0.37 倍であり、総合消費電力比は 0.96 倍、$1MHz$ 当りの消費電力比は 0.71 倍である。これよりステージ分割による遅延差短縮手法は遅延バッファ挿入による遅延差短縮手法に比べて、ステージ数は 1 つ増加するが、挿入する遅延バッファ数は 0.06 倍と少なくなる。そして性能は向上し、面積、消費電力は低下することが分かる。

ステージ分割のみによる結果とステージ分割による遅延差短縮手法を比較する。性能においては、ステージ分割による遅延差短縮手法の方がステージ分割のみより 1.77 倍高い性能となる。面積においては、実行ステージのみで 1.14 倍、全体では 1.03 倍の増加である。消費電力においては、実行ステージのみの消費電力比が 2.15 倍、総合消費電力比は 1.84 倍、$1MHz$ 当りの消費電力比は 1.04 倍の増加である。性能が高くなれば、消費電力も同率で増加するので、ステージ分割のみとステージ分割による遅延差短縮手法を比較した場合、ステージ数は同じであり、挿入した遅延バッファ量は 20 個と少数である。消費電力、面積はほぼ等しいが性能が向上することが分かる。

以上よりステージ分割による遅延差短縮手法は、ステージ数を少数増加させることによって、高い性能向上を得ることができ、しかも挿入する遅延バッファ量を非常に少なくできるので消費電力も抑えることができる。このため、本論文で提案したステージ分割による遅延差短縮手法は非常に有効な手法であると言える。

しかし対象となるプロセッサが複雑になると、ステージ分割の時間を含めた計算量が増えるために設計時間の増大が起こる。ステージ分割の時間によっては設計時間が大幅に増大する可能性がある。

## 6.2　手法としての考察

6.1 の表よりも分かるように、目的とした遅延差は MEM ステージの遅延差 121.8 ピコ秒である。遅延バッファ挿入のみによる遅延差短縮手法では、遅延差が 164.5 ピコ秒と目標値以下にすることはできなかった。しかしステージ分割による遅延差短縮手法では、ステージ分割のみで、ステージの遅延差が 184.8 ピコ秒、144.6 ピコ秒になる。これらのステージに遅延バッファ挿入による遅延差短縮手法を行うと、遅延差が分割する前よりも短いために挿入するバッファ数が少なくなった。

これよりステージ分割による遅延差短縮手法では、目標となる遅延差を設定し、その目標を達成するような設計の場合に非常に有効である。

目標値となるものは、メモリやキャッシュなどの記憶素子へのアクセスがあるステージにおける遅延時間である。これらの記憶素子へのアクセスがあるステージにおいては、ウェーブパイプライン化はうまく適用しない。提案したステージ分割による遅延差短縮手法は、ラッチを挿入してステージを分割する。そのため、パイプラインステージの役割を理解し、ウェーブ化が適用できるかどうかを判断する必要がある。

またステージを分割することは非常に困難である。今回は対象としたプロセッサが簡単なものであったためにうまく分割が行えたが、実際のプロセッサにおいて、最適な場所でステージを分割することは非常に難しい。

# 第 7章

# 結論

本論文では、このウェーブパイプライン方式をプロセッサに導入するに当たって、プロセッサの動作速度をより高くする目的とともに、ウェーブパイプライン方式を導入することによって生じる問題点である、設計時間の短縮と、性能向上と比べて遅延バッファ挿入による面積と消費電力の増大を抑えることも目的として、ステージ分割による遅延差短縮手法を提案し、その有効性を評価した。

第 2 章でウェーブパイプライン方式の説明を通常のパイプライン方式と比較しながら行った。様々な違いがあるが、ウェーブパイプライン方式と通常のパイプライン方式の最も異なる点は、通常のパイプライン方式がステージの最大遅延の最大値によって性能が決まるのと違い、ウェーブパイプライン方式はステージの最大遅延と最小遅延の差の最大値によって性能が決定することである。またこれまでの本研究室における結果より、ウェーブパイプライン方式の問題点も述べた。ウェーブパイプライン方式を導入する場合の問題点は、遅延差短縮を行うのに遅延バッファを挿入するため、性能の向上とともに面積も増大する。この面積増加が大きいために性能向上の利点が薄くなってしまう。

第 3 章では、第 2 章で述べたウェーブパイプライン方式の問題点を改善するため、これまでの遅延バッファ挿入による遅延差短縮手法を含んだステージ分割による遅延差短縮手法を新しく提案した。この手法は、各ステージの最小遅延と最大遅延を測定しつつ、ステージ内の回路素子段数も調べ、遅延差が目標遅延差の $n$ 倍以上であるかによって分割するか否かを決定し、ステージ内の回路素子段数が半分となるところで分割を行う。その後、これまでの手法である遅延バッファ挿入による遅延差短縮方法を行うものである。

この手法の利点を以下に示す。

- パイプラインのステージ数は増加するが、最大遅延がそのものが小さくなり遅延差を短縮させることができる。

- 分割することで遅延差が短縮されているので、遅延バッファ挿入による遅延差短縮手法を行っても挿入する遅延バッファ数を抑えることができる。
- 分割することで 1 つのステージ内の素子数が減るため、設計時間に関係する計算量が減る。

第 4 章でウェーブパイプライン方式を用いる対象となるプロセッサを決定した。それと共にプロセスルール 0.1 $\mu m$ における遅延モデルを決定し素子遅延と配線遅延の遅延パラメータを決定した。

第 5 章と第 6 章で、第 4 章で決定したプロセッサにステージ分割による遅延差短縮と遅延バッファ挿入による遅延差短縮手法のそれぞれにおけるウェーブパイプライン化を行い、評価した。結果として、ステージを分割することによる効果が大きいことを示し、それについて考察を行った。

最後に今後の課題について述べる。ステージ分割を行う条件となる最適値 $n$ の設定は注意が必要である。$n$ の値を小さくして何度もステージ分割を行うと、細かくステージを分割することになるのでラッチ数が多くなってしまうが、分割したステージの遅延差は小さいために挿入するバッファ数は少量ですむ。逆に $n$ の値を大きく設定すれば、ステージの分割数を抑えることができるためラッチ数は少なくてすむが、ウェーブパイプライン化により遅延バッファ数が膨大になり、 それによってチップ面積と消費電力が増大してしまう。これらのトレードオフを考慮し、設計目標によって最適値を設定することが重要になる。

ステージの分割手法において、本論文では遅延差が目標遅延差の $n$ 倍以上であるならば、1 つのステージを 2 つに分割する手法を用いている。ここでもし、分割したステージのそれぞれの遅延差も目標遅延差の $n$ 倍以上であるならば、分割したステージをさらに 2 つに分割して遅延差を調べることになる。しかし、ここで元の 1 つのステージを 3 つに分割することによって遅延差が目標遅延差の $n$ 倍以下になる可能性がある。この場合、本論文で提案したステージ分割手法ではうまく分割できず、ステージ数の増加に繋がる。よって 1 つのステージを 3 つのステージに分割する手法も考える必要がある。1 つのステージを 2 つか 3 つに分割する手法ができれば、後はこの組合わせによって 4 つ、5 つ、$\cdots$ とステージを分割することが可能となる。

また、本論文においては簡単なパイプラインプロセッサによってその有効性を確かめたが、我々の研究室では、マルチスレッド型プロセッサ・アーキテクチャの 1 つである Multithreaded Ultrapipeline Processor(MUP) の研究も行っている [4],[7]。この MUP はマルチスレッド型プロセッサを高性能化したものであり、それに用いるパイプライン化され

たキャッシュメモリの研究も行われている [5]。これによりプロセスルール $0.1\mu m$ において 6.5GHz まで動作周波数を上げることが可能となっている。

このMUPはウェーブパイプライン方式を導入するのに適している。そのためこの手法をMUPに適用することで、キャッシュメモリにおける限界の性能まで、さらなる性能向上を図ることができると考えられる。

# 謝辞

本研究を進めるにあたり、終始熱心かつ寛容な御指導を賜わりました 日比野 靖 教授に心から感謝いたします。

その他、貴重な御意見、御討論を頂きました日比野研究室の皆様をはじめ、多くの方々の御助言に対し厚く御礼申し上げます。

# 参考文献


[1] C.Thomas Gray, Wentai Liu, Ralph K.Cavin, “Wave Pipelining : Theory and CMOS Implementation”, Kluwer Academic Publishers, 1994

[2] Fabian Klass and Michael J.Flynn, “COMPARATIVE STUDIES OF PIPELINED CIRCUITS”, Technical Report No.CSL-TR-93-579, 1993

[3] Wayne P.Burleson, Maciej Ciesielski, Fabian Klass, Wentai Liu, “Wave-Pipelining : A Tutorial and Research Survey”, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION(VLSI) SYSTEMS, VOL.6, NO.3, 1998

[4] 伊藤 英治, “関数型プログラムの実行に適したマルチスレッド型プロセッサ・アーキテクチャに関する研究”, 北陸先端科学技術大学院大学修士論文, 1997

[5] 鵜飼 和歳, “パイプライン化によるキャッシュの高周波動作の可能性に関する研究”, 北陸先端科学技術大学院大学修士論文, 1999

[6] 池田 吉朗, “ウェーブパイプラインを用いたマルチスレッド型プロセッサアーキテクチャに関する研究”, 北陸先端科学技術大学院大学修士論文, 1999

[7] 永田 真也, “可変機構を備えたマルチスレッド型プロセッサに関する研究”, 北陸先端科学技術大学院大学修士論文, 2000

[8] 永谷 充孝, “ウェーブパイプラインを用いたプロセッサの設計に関する研究”, 北陸先端科学技術大学院大学修士論文, 2000

[9] 堀口勝治, “ULSI 設計技術”, 電子情報通信学会, 1993

[10] 榎本忠儀, “CMOS 集積回路”, 培風館, 1996