修 士 論 文

フィードバックのあるパイプライン回路のウェーブパイプライン化に関する研究

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

大石 亮介

2002 年 3 月

修 士 論 文

フィードバックのあるパイプライン回路のウェーブパイプライン化に関する研究

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　田中清史 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

010016 大石 亮介

提出年月: 2002 年 2 月

概 要

プロセッサの設計においてクロック周期の短縮は最重要課題のひとつである. ラッチへのクロックの同時入力を前提としないウェーブパイプラインでは, クロック周期を大幅に短縮することが可能になるがフィードバックがあると実現が困難である. 一方で, 配地・配線段階の設計手法として準同期式回路が提案されている. 準同期式回路とウェーブパイプラインはクロック入力の同期制約がないという点で共通している. 本研究ではウェーブパイプライン回路を準同期式回路とみなして回路を高速化する, 準同期パイプラインを提案する.

目 次

第1章　はじめに

プロセッサの設計においてクロック周期の短縮は最重要課題のひとつである．通常のパイプラインプロセッサでは, クロックを全てのラッチに同時に入力することを前提としており, プロセッサが動作するクロック周期はラッチ間の最大信号遅延によって制限される．ラッチへのクロックの同時入力を前提としないウェーブパイプライン [1, 5, 10] では, 各々のラッチにクロックを入力するタイミングを制御し, またラッチ間に適切に遅延を挿入することでクロック周期を大幅に短縮することが可能になる．しかしフォワーディング機構のようなフィードバックを持つパイプライン回路の場合, ラッチへのクロック入力のタイミング制御が難しくなる．そのため従来研究ではウェーブパイプラインはフォワーディングの必要のないマルチスレッドアーキテクチャに限定されている場合が多かった．一方で, 配地・配線段階の設計手法として準同期式回路 [2, 4, 6] が提案されている．準同期式回路は回路中のフリップフロップへのクロック信号線に対し意図的に遅延を挿入することによってクロック周期を小さくすることができる．準同期式回路とウェーブパイプラインはクロック入力の同期制約がないという点で共通している．

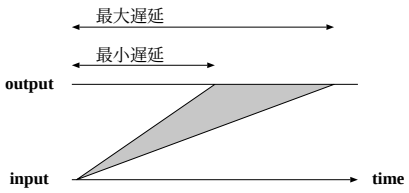
本研究ではウェーブパイプライン回路を準同期式回路とみなして回路を高速化する, 準同期パイプラインを提案する．準同期式回路のクロック周期の下限はレジスタ間の各信号線の遅延制約から得られる制約グラフのクリティカルサイクルによって求められる．そこでパイプライン回路に対しラッチ位置の変更やステージ分割を行うことで, 制約グラフのクリティカルサイクルを修正することによりクロック周期を小さくすることを試みる．準同期パイプラインの構成アルゴリズムを利用することで, フィードバックがありながらも高速に動作するパイプラインを持つアーキテクチャを設計することができる．

第2章　準同期パイプライン

本研究ではウェーブパイプラインに準同期式回路の制約グラフの概念を導入した, 準同期パイプラインを提案する.

2.1　ウェーブパイプライン

ウェーブパイプラインは, パイプラインの一種である. 図 2.1 に示されるような, 通常のプロセッサにおける同期パイプラインでは, 全てのラッチに対し同時にクロックを入力することが前提となっている. このとき, 最小クロック周期は各ラッチ間の最小遅延と最大遅延の差で制限される. 一方, 図 2.2 のようなウェーブパイプラインでは, クロックの同時入力の制約はない. そのため, 最小クロック周期はラッチ間の最大遅延と最小遅延の差で制限される. このため回路に冗長な遅延を挿入し, 最小遅延を大きくさせる事で最大遅延と最小遅延の差を小さくし, クロック周期を短縮させることが可能になる.



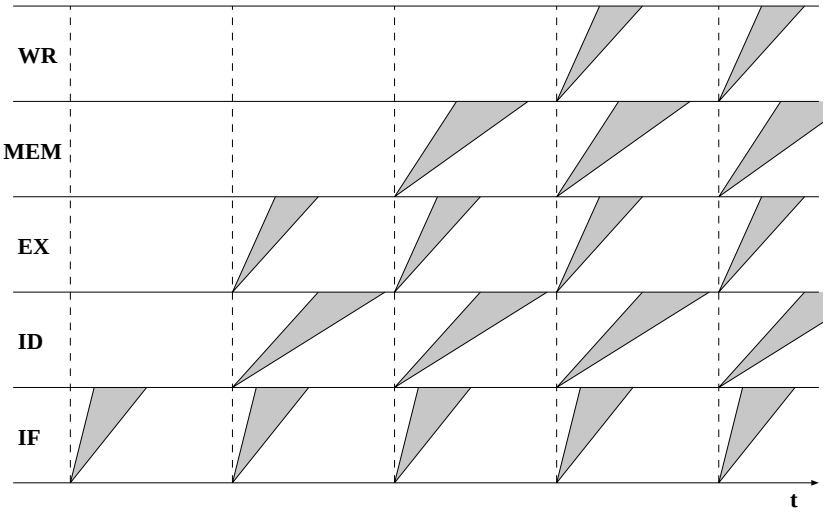


図 2.1: 同期パイプライン回路

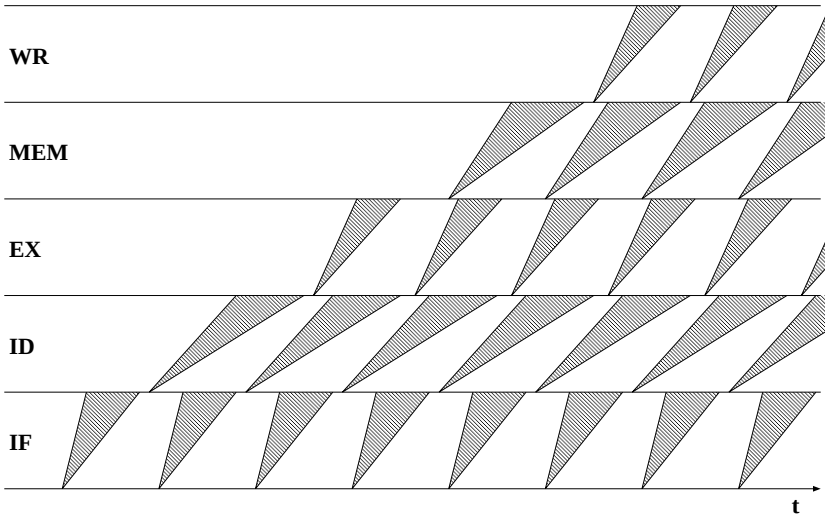


図 2.2: ウェーブパイプライン回路

2.2 準同期式回路

図 2.3 のような同期式回路を考える．レジスタ b からレジスタ a への信号遅延は 12 であり，これはこの回路のレジスタ間の最大信号遅延である．したがって，この回路の完全同期式における最小クロック周期は 12 となる．しかし，各レジスタにクロックは異なるタイミングで入力されている．クロックはレジスタ b を基準として a， c， d にそれぞれ 3， 7， 6 だけ遅れて入力されおり，この回路はクロック周期 9 で動作する．このように，各レジスタにクロックを異なるタイミングで入力することを許容する回路を**準同期式回路** と呼ぶ．

各レジスタに入力するクロック時刻の基準時刻からの遅れを**クロックタイミング**と呼び，レジスタ v のクロックタイミングを $s(v)$ で表す．図 2.3 では $s(a) = 3$， $s(b) = 0$ 等となる．また，すべてのレジスタのクロックタイミングをまとめてクロックスケジュールと呼ぶ．

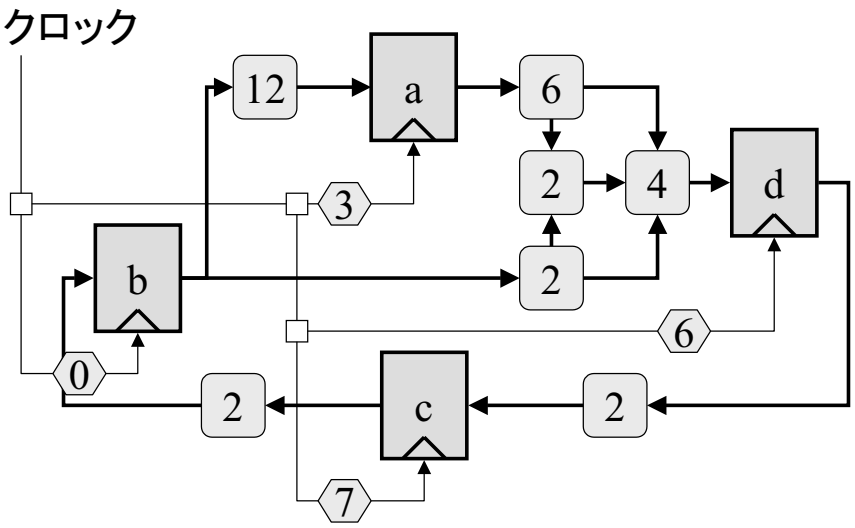


図 2.3: 準同期式回路

レジスタ u からレジスタ v への信号伝搬の最小（最大）遅延を $d_{\min}(u, v)$ （$d_{\max}(u, v)$）で表す．遅延グラフはレジスタを点とし，レジスタ間の信号伝搬を枝としたグラフで，回路のレジスタレベルの遅延情報を表す．各枝は最大遅延，最小遅延の二つのラベルを持つ．図 2.4 は図 2.3 の回路の遅延グラフである．例えば，レジスタ a からレジスタ d への最小遅延 10， 最大遅延 12 の信号伝搬に対応し，遅延グラフは a から d へラベル $10, 12$ の枝を持つ．また，枝 (u, v) は対応する制約式が満足されないとき**違反枝**と呼ばれる．

レジスタ u からレジスタ v へ信号が伝搬するとき，その最小遅延と最大遅延に対しては，準同期式回路が動作するための制約が課せられる．遅延 $d_{\min}(u, v)$， $d_{\max}(u, v)$ とクロックタイミング $s(u)$， $s(v)$ を用いてその制約は以下のように表すことができる．

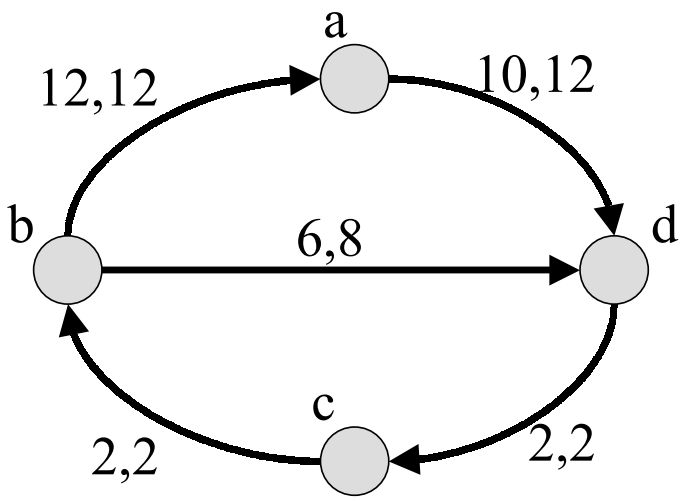


図 2.4: 回路のレジスタレベル表現

No-Zero-Clocking Constraint

$$s(u) - s(v) \leq T - d_{\max}(u, v)$$

No-Double-Clocking Constraint

$$s(v) - s(u) \leq d_{\min}(u, v)$$

この式で，T はクロック周期である（図 2.5 参照）．準同期式回路が動作するためには，すべての信号伝搬の最小遅延，最大遅延に対して上記の式が成り立たなければならない．信号伝搬の最大遅延に対する制約 No-Zero-Clocking Constraint と最小遅延に対する制約 No-Double-Clocking Constraint は $s(v) - s(u) \leq w$ （w は $T - d_{\max}(v, u)$ または $d_{\min}(u, v)$）という形をしている．この制約式を u から v への重み w の枝として表した有向グラフ $G(V, E, w)$ を**制約グラフ**と呼ぶ．ここで，V はレジスタ集合，E は制約式を表す枝 (u, v) の集合，w は枝の重みを表す．制約グラフは，遅延グラフの枝 (u, v) に対して，重み $T - d_{\max}(u, v)$ を持つ枝 (v, u)（Z 枝），重み $d_{\min}(u, v)$ を持つ枝 (u, v)（D 枝）を持つ．

図 2.4 に対応する制約グラフを図 2.6 に示す．例えば，レジスタ a から d への最大遅延 12 , 最小遅延 10 の信号伝搬にはレジスタ d から a への重み $T - 12$ の Z 枝と a から d へ重み 10 の D 枝が対応する．図中の点に書かれた数値は対応するレジスタのクロックタイミングを表す．図 2.6 のスケジュールでは，クロック周期 9 で違反枝が生じない．

ここで，遅延 $d_{\max}(u, v)$, $d_{\min}(u, v)$ を定数として，クロックタイミング $s(v)$, $s(u)$ は任意に設定可能とすると，以下の定理が成り立つ．

定理 1 あるクロック周期 T において，すべての制約式を満たすクロックスケジュールが存在する必要十分条件は，制約グラフで $T = T'$ とおいたとき，負の重みを持つ有向閉路

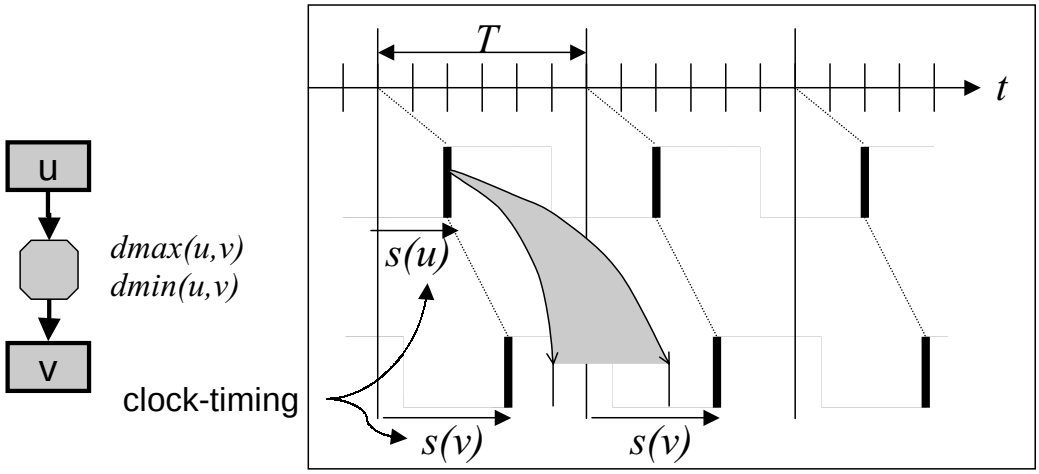


図 2.5: 準同期式回路のタイミング制約

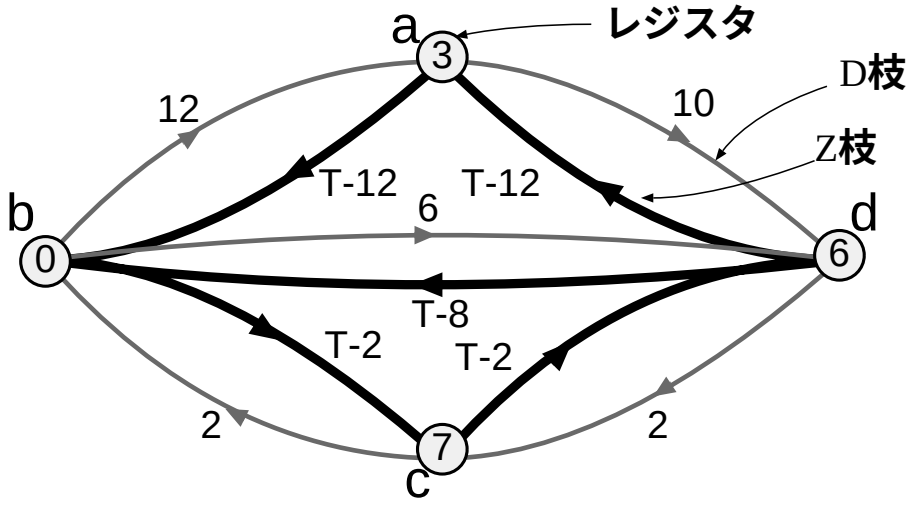


図 2.6: 制約グラフ

（負閉路）が存在しないことである．

ここで，閉路の重みは閉路中のすべての枝重みの和である．

定理 1 より, 準同期式回路が動作可能な最小クロック周期は制約グラフ中に負閉路が存在しないような最小の T であることがわかる．これによりより最小クロック周期は一意に定まり，多項式時間で求めることができる [[6]]．クロック周期を最小クロック周期以下に設定すると回路に何らかの変更を加えなければならない．

2.3 準同期パイプライン

クロックの同時入力制約のないウェーブパイプライン回路は準同期式回路の一種と見ることができる．このとき, パイプラインの各ステージはレジスタを含む順序回路に, パイプラインラッチはレジスタに相当する．このように準同期式回路として捉えたウェーブパイプラインを, 本稿では**準同期パイプライン**と呼ぶ．準同期パイプラインは, 本質的にはウェーブパイプラインと同一である．

第3章　準同期パイプラインの動作

図 3.1 は, 準同期パイプラインの一つのステージを表している. L_1, L_2 はラッチで, ラッチ間には最大遅延 $d_{\max}$, 最小遅延 $d_{\min}$ の遅延素子がある. 各ラッチへのクロック信号の入力のタイミングには, クロックソースに対し $s(L_1)$, $s(L_2)$ だけ遅延がある (図 3.2). このパイプラインが動作する条件は以下のようになる.

$$s(L_1) + d_{\max} \leq s(L_2) + T \tag{3.1}$$

$$s(L_1) + d_{\min} \geq s(L_2) \tag{3.2}$$

この 2 つの式を, $s(u) - s(v) \leq w$ という形に変形し, 「点 u から点 v への重み w の枝」として制約グラフ (図 3.3) を得る. 制約グラフ中の任意の閉路の重み和が非負であるときにのみ, 回路が動作する事が知られている [2, 6]. このとき, 制約グラフは与えられた回路の論理には全く依存しないため, フィードバックがあるパイプラインでも同様に制約グラフを構成することができる. この制約条件を満たすようなクロック周期は, 一般的にラッチ間の最大遅延よりも小さくすることができる.

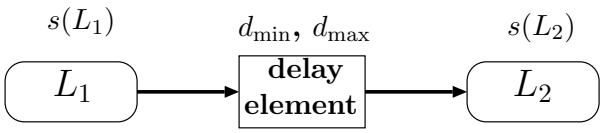


図 3.1: パイプラインと遅延

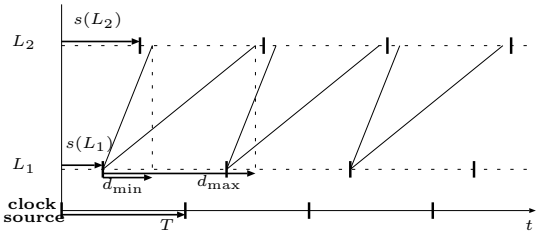


図 3.2: 準同期パイプラインの動作

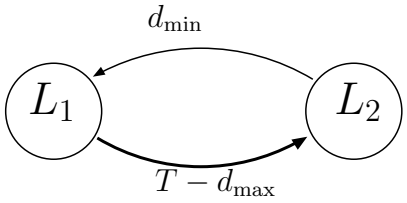


図 3.3: 制約グラフ

第4章　準同期パイプラインの構成法

本研究で提案する, 同期式で設計したパイプラインから準同期パイプラインを構成するアルゴリズムと, このアルゴリズムに必要なバッファ挿入アルゴリズムを示す.

4.1　準同期パイプラインの構成アルゴリズム

準同期パイプラインの構成問題は以下のように定式化される.

準同期パイプライン構成問題

入力 同期パイプライン回路のネットリスト

出力 準同期化されたパイプライン回路のネットリスト

制約 ソフトウェアの変更を伴わない.

目標 クロック周期ならびにプログラムの実行速度 (CPI) の高速化

準同期パイプラインの構成アルゴリズムは以下のように行う.

1. 与えられた回路の制約グラフを構成し, 最小クロック周期とクリティカルサイクルを求める. クリティカルサイクルとは, 枝重みの和が最小になるような閉路のことである.
2. クリティカルサイクルに含まれるステージに対し, ステージ分割または遅延バッファ挿入を行う.
3. 変更後の回路が消費電力やチップ面積等の制約条件を満たしているか調べる.
4. 最小クロック周期が小さくなる限り, 2 と 3 の操作を繰り返す.

以上のようにしてウェーブ化されたパイプライン回路を, さらにプロセッサ全体に対して準同期化を行うことも可能である [7].

4.2　バッファ挿入アルゴリズム

バッファ挿入によるプロセッサの高速化を目標とする. 準同期パイプライン回路に対するバッファ挿入のアルゴリズムを以下に示す.

目標スケジューリング問題は以下のように定式化できる．

目標スケジューリング問題

入力 制約グラフ $G(V, E, w)$，クロック周期 T, 目標クロックスケジュール $o(v)$

出力 クロックタイミング $s(v)$

制約 $s(u) - s(v) \leq w(u, v)$

目標 $\sum(s(v) - o(v))$ の最小化

制約グラフはクロック周期 T に定数を代入すると定数重みの枝を持つ有向グラフになる．制約グラフは枝 (u, v) が存在すれば必ず枝 (v, u) が存在するなどの特徴があるが，以下，制約グラフを定数重みを持つ一般の有向グラフとして扱う．

クロックスケジュール s が準同期パイプラインのすべての制約を満足するとき， s は “feasible” であるという．クロック周期 T 最小クロック周期 T_S 以上であるならば feasible な解が存在する．本章では T は T_S 以上であるとする．

4.3 二段階スケジューリングアルゴリズム

この問題に対し,以下のような二段階アプローチをとるアルゴリズムを利用する．(1) 初期解としてすべてのラッチを目標値にスケジュールし，違反枝が存在する限りその枝の両端点のクロックタイミングを制約を満たすように変更する（違反枝除去）．(2) 制約を満たす範囲で各ラッチのクロックタイミングを目標に近づける（タイミング調整）．

このアルゴリズムはグリーディーなアルゴリズムであるが，最初に各ラッチのタイミングを目標値に設定することにより，目標値から離れた局所最適解に収束することを防いでいる．

違反枝除去の操作を詳しく説明する．枝 (u, v) が違反枝であるとき，ラッチ u, v のクロックタイミングをそれぞれ $s(u), s(v)$ とすると，

$$\Delta = -s(u) + s(v) + w(u, v)$$

は枝 (u, v) の違反量を表し $\Delta > 0$ である．この時，違反枝除去では制約を満たすようにラッチ u, v のクロックタイミングを以下のように $s'(u), s'(v)$ と変更する．

$$s'(u) = s(u) - \frac{\Delta}{2}$$
$$s'(v) = s(v) + \frac{\Delta}{2}$$

この操作を繰り返し，違反枝のないクロックスケジュールを求める違反枝除去アルゴリズムを図 4.1 に示す．図 4.1 で， ϵ は計算誤差である．このアルゴリズムの収束性についてはまだ証明されていないが，実験的に α が単調減少することを確認されている．

Algorithm 違反枝修正
{
  $s(v) = o(v)$ for $\forall v$;
  do
  {
    $\alpha = 0$;
    while$((u, v) \in E)$
      if$(s(v) - s(u) > w(u, v))$
        $\Delta = -s(v) + s(u) + w(u, v)$;
        $s(v) = s(v) - \frac{\Delta}{2}$;
        $s(u) = s(u) + \frac{\Delta}{2}$;
        $\alpha = \max(\alpha, \Delta)$;
  }
  while$(\alpha > \epsilon)$
}

図 4.1: 違反枝修正アルゴリズム

違反枝除去を繰り返すと他の枝の修正によりある枝に余裕が生じる場合がある．例えば，図 4.2(a) に示す制約グラフの一部とそれに含まれる各ラッチの目標クロックタイミングが与えられたとする．枝 (a, b)，(c, b) の順に違反枝除去の操作を行うと，図 4.2(c) のスケジュールが得られる．この時，枝 (a, b) に余裕が生じ，ラッチ a のクロックタイミングを目標値に戻すことができる（図 4.2(d)）．タイミング調整では，このように余裕が生じた枝に対して，制約を満たす範囲で両端点のクロックタイミングを目標に近づける操作を繰り返し，feasible でかつより目標により近いスケジュールを得る．

あるラッチのクロックタイミングが制約を満たすか否かは，他のラッチのクロックタイミングに依存する．各ラッチのクロックタイミングに上下限が与えられたとき，ラッチ v のクロックタイミングは，ラッチ v に接続する枝がすべて制約を満たすように隣接点のクロックタイミングを与えられた範囲内で設定可能ならば，feasible と呼ばれる．各ラッチのクロックタイミングは，与えられた上下限内の連続した区間で feasible となる．

ラッチ v の feasible なクロックタイミングの区間を $(s_{\min}(v), s_{\max}(v))$ と表す．ラッチ v の feasible な区間は，隣接するラッチのクロックタイミングの上下限より，以下のように求めることができる．

$$
\begin{aligned}
s_{\max}(v) &= \min_{\forall(u,v)}(w(u, v) + s_{\max}(u)) \\
s_{\min}(u) &= \max_{\forall(u,v)}(s_{\min}(v) - w(u, v))
\end{aligned}
$$

制約グラフ中には負閉路がないので，$s_{\max}(v)$ は最短パスアルゴリズムを用いて簡単に求めることができる．同様に $s_{\min}(u)$ は制約グラフの枝の向きを反転したグラフ上で最短パスアルゴリズムを用いて求めることができる．

与えられた上下限に対し，すべてのラッチが feasible なクロックタイミングを持つとす

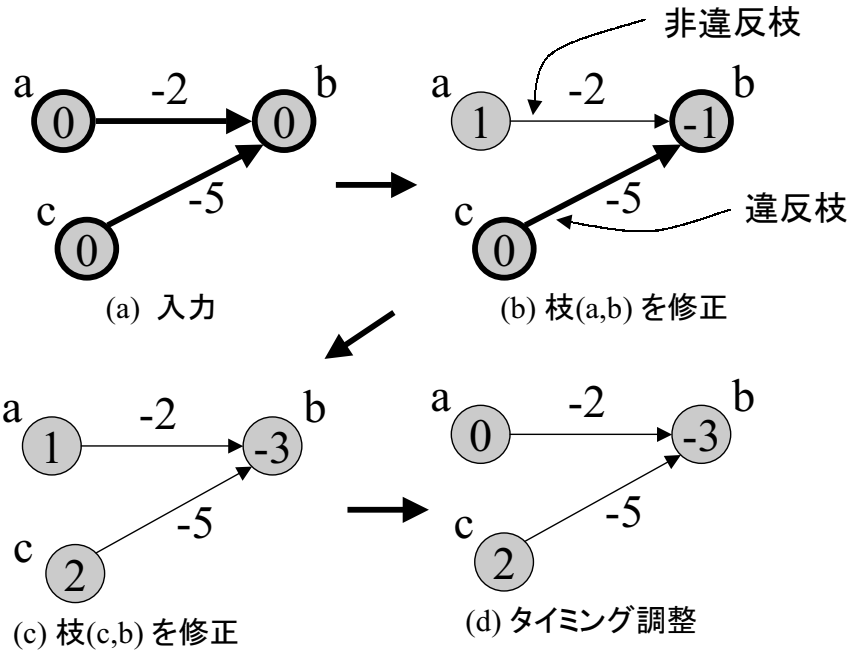


図 4.2: タイミング調整可能な例

る．この時，あるラッチのクロックタイミングを feasible な値に設定すると，他のラッチの feasible なクロックタイミングの範囲は狭くなることがあるが， feasible なクロックタイミングが空となることはない．

違反枝除去で定められた $s(v)$ と目標値 $o(v)$ の間を初期の上下限として設定すると，$s(v)$ がその範囲内に含まれているので，各ラッチには少なくとも feasible なクロックタイミングが存在する．タイミング調整では，任意にラッチを選択し，そのラッチのクロックタイミングを目標値に最も近い feasible なクロックタイミングに設定する，という操作を繰り返して，より目標値に近いクロックスケジュールを得る．

タイミング調整アルゴリズムを図 4.3 に示す．タイミング調整の結果はラッチを選ぶ順番により変化する．

4.4 遅延挿入に伴う回路の冗長化

このバッファ挿入アルゴリズムは, 一つの論理が途中で複数のパスに分岐しているような場合は, それらを全て別々の枝とみなす．このため, ある枝に対してバッファを挿入した場合に, その挿入位置によっては別の枝の制約条件が変化してしまうことがある．このような事態を避けるために, 回路を冗長化してバッファを挿入することがある．図 4.4 は論理パス $(L1, L2)$ に対するバッファ挿入の例である．配線遅延を無視すると, このラッチ

Algorithm タイミング調整
{
  while($v \in V$)
    if($s(v) > o(v)$)
    {
      $s_{\min}(v) = o(v)$; $s_{\max}(v) = s(v)$;
    } **else** {
      $s_{\max}(v) = o(v)$; $s_{\min}(v) = s(v)$;
    }
  while($v \in V$)
  {
    $s_{\min}, s_{\max}$ を再計算 ;
    if($s(v) > 0$)
    {
      $s(v) = s_{\min}(v)$;
    } **else** {
      $s(v) = s_{\max}(v)$;
    }
  }
}

図 4.3: タイミング調整アルゴリズム

間は最大遅延と最小遅延の差が 7 であるが (c, d) 間に遅延 5 のバッファを挿入により, 遅延差を 2 に縮めている. このとき演算素子 d を複製し, パスによって動作のタイミングを変更することにより, 最大遅延が大きくなってしまうのを防いでいる.

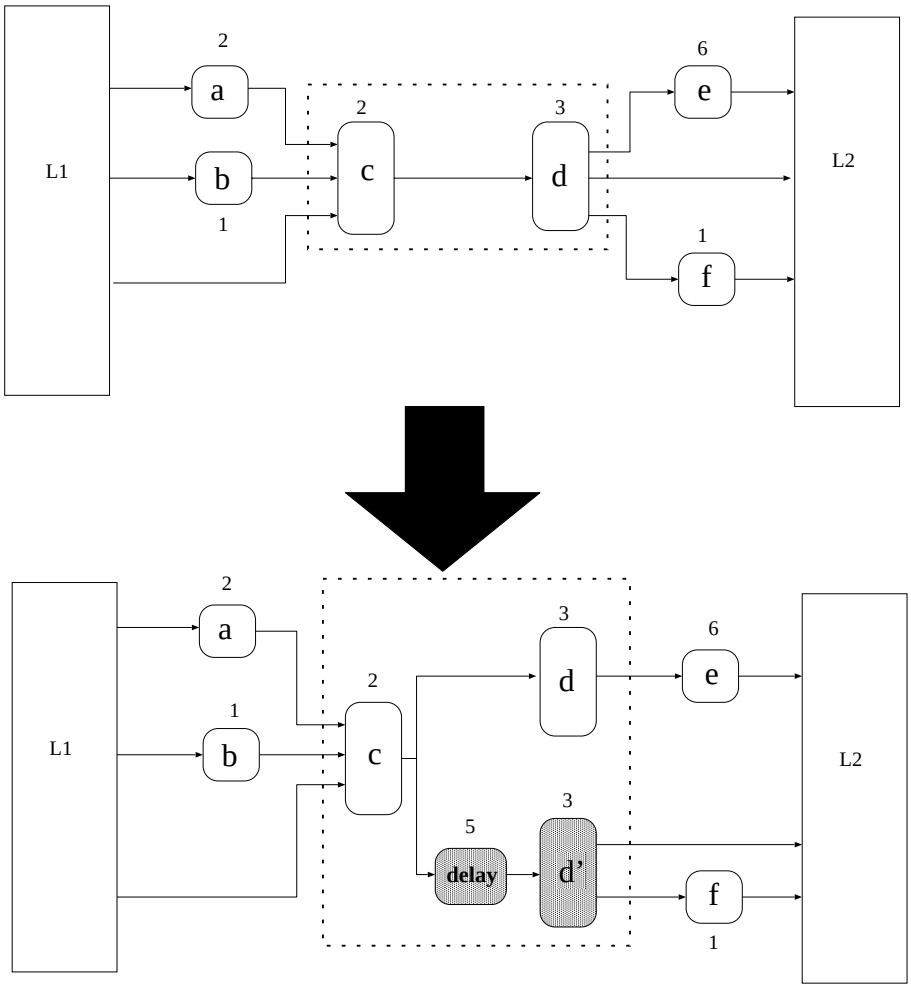


図 4.4: 遅延挿入に伴う回路の冗長化

第5章　準同期パイプラインプロセッサ WAVIST の試作

準同期パイプラインの動作検証を目的とした, 非常に簡単な RISC 型プロセッサ **WAVIST** を試作した.

本試作を行うにあたり, 論理設計ならびにシミュレーションには Cadence 社の Verilog-XL を, 論理合成には Synopsys 社の Design Compiler を, 配置・配線には Avant! 社の Apollo をそれぞれ利用した.

5.1　試作の手順

本試作は, 以下のような手順で行った.

1. 同期式でフィードバックを持たないパイプライン回路を設計した.
2. パイプライン回路に対し, フィードバックとなるフォワーディング回路を追加した.
3. 準同期パイプライン化を行った.

フォワーディング回路の実装方法はいろいろ考えられるが, WAVIST のパス遅延を評価したところ最大遅延パスが EXE ステージにあったため, EXE ステージの遅延が増大しないように実装した. そのため, フォワーディング回路を MEM ステージに置き, EXE ステージへフォワードさせることにした. そのため, 演算命令でフォワーディングを行っても 1 サイクル分のストールが発生する.

このような実装にした理由はもう一つある. EXE ステージの出力をまた EXE ステージへフォワーディングさせると, ストールなしに演算命令の出力をフォワーディングさせることができる. しかし, この場合, 制約グラフに自己閉路ができる. 制約グラフにおけるクリティカルサイクルが自己閉路であった場合, 制約式が

$$T - d_{\max} \leq 0$$

となってしまい最小遅延の影響を受けないので, 準同期化による高速化が不可能になる. これは本実験の主旨に反する.

5.2 プロセス概要

本試作で利用したプロセスの概要を以下に示す.

- PolySi: 2 層
- メタル配線: 3 層
- 使用ライブラリ: EXD 社ライブラリ (東大 VDEC 版) (Verilog-XL, DesignCompiler, ApolloXO)
- 電源電圧: 3.3[V]

5.3 WAVIST 基本仕様

レジスタ 8bit の汎用レジスタを 8 本持つ. そのうちのひとつをゼロレジスタとして実装する. また汎用レジスタの他にプログラムカウンタ (PC) として 8bit のレジスタを 1 本持つ.

命令語長 全ての命令語は, オペランドを含め 16bit である. 命令フェッチも 16bit 単位で行う. 命令には以下の 2 つの形式がある.

- 命令部 (5bit) + **rs**(3bit) + **rt**(3bit) + **rd**(3bit) + function(2bit)
- 命令部 (5bit) + **rs**(3bit) + address/immediate(8bit)

アドレス空間 命令メモリ, データメモリそれぞれ 8bit ずつのアドレス空間を持つ. 命令メモリとデータメモリは完全に独立している.

メモリ 命令メモリとして最大 16bit * 256word, データメモリとして最大 8bit * 256word を **WAVIST** 内部に持つ. **WAVIST** はプログラム実行中に外部メモリへ直接アクセスすることはできないが, 実行を停止している間に, 外部から内部メモリの参照/変更が可能である. またユーザプログラムが命令メモリの内容を更新することはできない.

外部割り込み 外部からの割り込みが発生すると, **WAVIST** は現在の状態を保存したまま一時的に実行を停止する. 割り込み要因が解除された時点で実行を再開する.

図 5.1 は **WAVIST** の概略回路図である.

5.4 命令セット

Load/Store データメモリ・レジスタ間転送命令.

- load: R1 ← [address]
- store: [address] ← R1

Arithmetic 算術演算命令. 演算は全て 2 の補数として行う.

- add: R1 ← R2 + R3
- sub: R1 ← R2 - R3

Logical 論理演算命令

- and: R1 ← R2 and R3
- or: R1 ← R2 or R3
- xor: R1 ← R2 xor R3
- not(1 の補数): R1 ← not R2
- neg(2 の補数): R1 ← neg R2
- shift left arithmetic: R1 ← R2 * R3
- shift right arithmetic: R1 ← R2 / R3

Branch 分岐命令. 分岐予測は常に不成立側へ行う.

- jmp: PC ← R1
- push: R1 ← PC
- blt: PC ← address if R1 < 0
- beq: PC ← address if R1 $= 0$

Other Fuctions その他の命令

- nop
- halt

5.5　命令のインプリメント

	5bit	3bit	3bit	3bit	2bit
nop	00000	000	000	000	00
add	00001	rs	rt	rd	00
sub	00001	rs	rt	rd	01
and	00010	rs	rt	rd	00
or	00010	rs	rt	rd	01
xor	00010	rs	rt	rd	10
not	00100	rs	rt	000	00
neg	00100	rs	rt	000	01
sla	00100	rs	rt	000	10
sra	00100	rs	rt	000	11
jmp	01000	rs	000	000	00
push	01000	rs	000	000	01
blt	10001	rs	address		
beq	10010	rs	address		
load	10100	rs	address		
store	11000	rs	address		
halt	11111	111	111	111	11

5.6 ピン割り当て

端子名	説明	ピン数
VDD	電源	16
VSS	電源	16
A0-A7	アドレスバス	8
D0-D15	データバス	16
PCE	チップ・イネーブル (命令)	1
PWE	ライト・イネーブル (命令)	1
DCE	チップ・イネーブル (データ)	1
DWE	ライト・イネーブル (データ)	1
CLK	クロック信号	1
RST	内部状態のリセット	1
INT	外部割り込み (一時停止)	1
ALC	オールクリア	1
PC0-CP7	プログラムカウンタ	8
SR0-SR7	デバッグ用シフトレジスタ	8
SCLK	シフトレジスタ用クロック	1
LT0-LT23	ラッチの一部	24
NC	予備 or 拡張用	6
ピン数合計		111

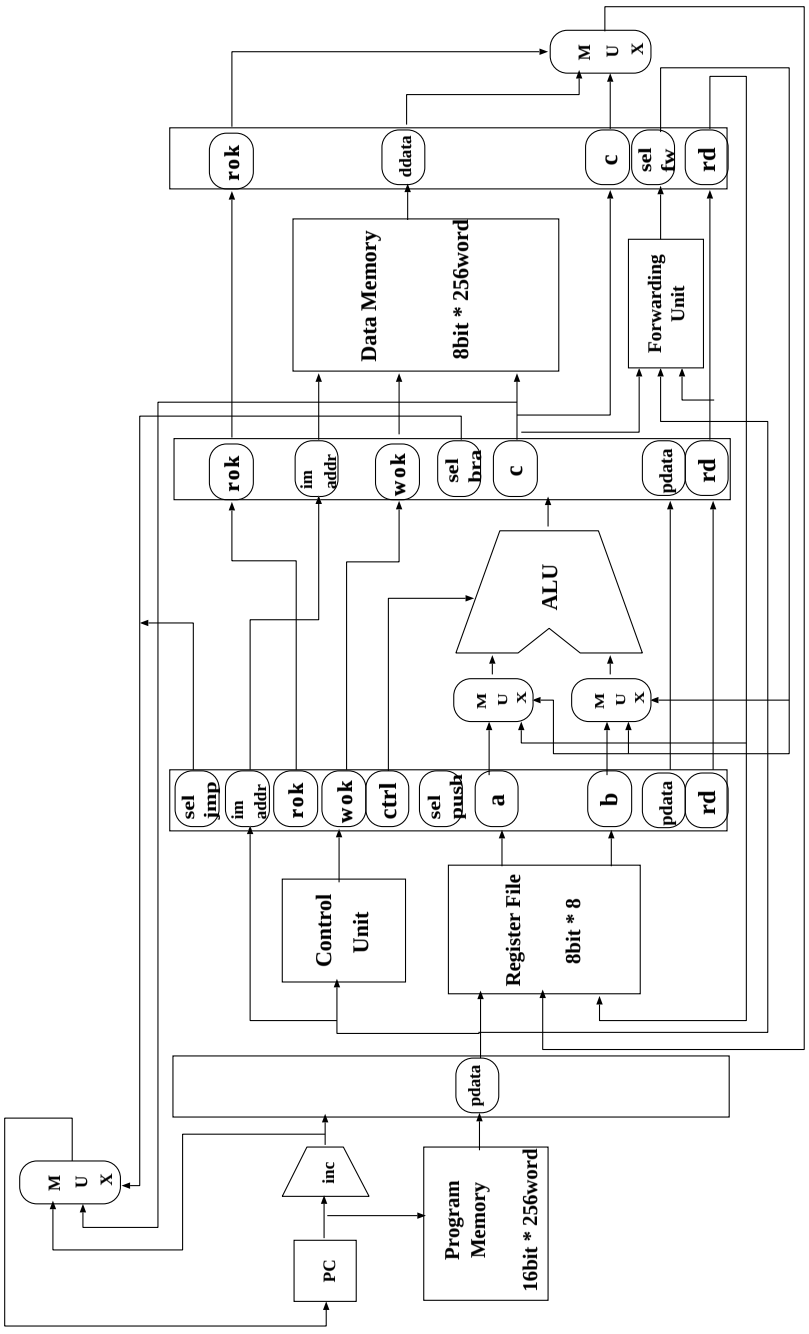


図 5.1: **WAVIST** 概略回路図

5.7 実験

試作したプロセッサ **WAVIST** に対し、本研究で提案するアルゴリズムによる準同期化の実験を行い、性能を評価した。**WAVIST** を Design Compiler 論理合成した結果, メモリとレジスタファイルを除いてラッチの総ビット数が 159, 総論理パス数が 1029 の Verilog 形式のネットリストを得た. 次にを Apollo で初期配置・配線を行い, 配線遅延を付加した SDF 形式の遅延情報ファイルを得た.

このネットリストと遅延情報ファイルを元に最大遅延, 最小遅延を計算し, 制約グラフを構成した上で準同期化に必要なバッファ挿入を行った. 今回の実験ではステージ分割は行わなかった. 挿入する遅延バッファの遅延を 0.25[ns] とした[1]. このとき, 初期解として与えるスケジュールの目標値は 0.00[ns] とした. またバッファ挿入による配線遅延の影響は考慮しないものとした. 4.4 節で説明した回路の冗長化は行わなかったため, バッファの挿入位置によっては最大遅延が大きくなる可能性がある. クロックツリーに対するバッファ挿入は, 各ラッチごとにスケジュールされたクロックタイミングの値に最も近くなるように行った. プログラムメモリ, データメモリ内部のパスでは遅延の挿入を行わななかった.

この条件で準同期化アルゴリズムを実行したところ, 412 個のバッファを挿入できる事がわかった. この時パイプラインの性能は図 5.2 のようになり, 高速化できる事が示された. このとき, 各ステージでの遅延は図 5.3 のようになった. 各ラッチのスケジュールは, 図 5.4 のようになった. また, 準同期化の前後でクリティカルサイクルは図 5.5 のように変化した. 実際の各パスの遅延の例として, EXE ステージについて図 5.6–5.10 にバッファ挿入後の遅延データを示した.

	同期パイプライン	準同期パイプライン	増減比
最大遅延 [ns]	11.95	11.95	100.0%
最大遅延差 [ns]	9.29	6.18	66.5%
最小クロック周期 [ns]	11.80	9.03	76.5%
メモリを除く論理回路の素子数	932	1344	144.2%
クロックツリーのバッファ数	283	652	230.3%
チップ全体の素子数	26569	29350	110.4%

図 5.2: **WAVIST** 準同期化による性能改善

MEM ステージに関しては, データメモリ部の最大遅延が 11.95[ns], フォワーディング回路の最大遅延が 2.61[ns] であった. このため, フォワーディング回路の挿入による MEM

[1] この値は実際の回路で挿入した場合の遅延の値とは異なる.

ステージ	最大遅延 [ns]	最小遅延 [ns]	準同期化後の最大遅延 [ns]	準同期化後の最小遅延 [ns]
IF	10.67	1.10	10.67	5.88
ID	4.08	0.24	4.08	2.64
EX	11.80	0.71	11.80	6.28
MEM	11.95	0.25	11.95	5.77
WB	3.29	0.13	3.29	1.30

図 5.3: **WAVIST** 準同期化による各ステージの性能改善

スケジュール [ns]	ラッチ数
0.00	73
$0.25 \sim 1.00$	0
$1.25 \sim 2.00$	43
$2.25 \sim 3.00$	32
$3.25 \sim 4.00$	11

図 5.4: **WAVIST** 準同期化によるスケジューリング

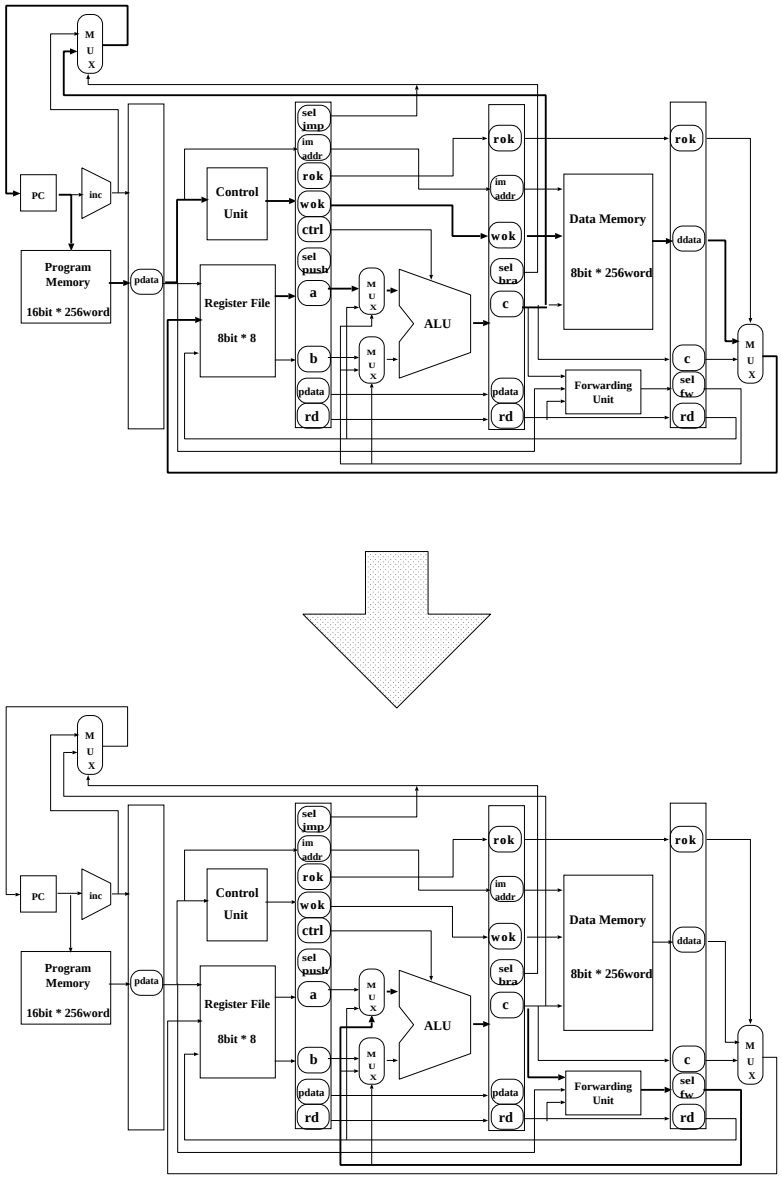


図 5.5: 準同期化によるクリティカルサイクルの変化

ステージへの遅延の影響はなかったと言える.

入力端子名	出力端子名	遅延差	最大遅延	最小遅延
IE1	EM1	2.91	11.34	8.43
IE2	EM1	2.77	10.85	8.08
IE3	EM1	2.55	10.69	8.14
IE1	EM2	2.50	10.89	8.39
IE2	EM2	2.36	10.40	8.04
IE3	EM2	2.21	10.24	8.03
IE4	EM2	1.89	10.78	8.89
IE4	EM1	1.88	11.23	9.35
IE1	EM3	1.33	9.50	8.18
IE5	EM2	1.29	9.37	8.09
IE6	EM1	1.41	10.04	8.63
IE5	EM1	1.31	9.82	8.51
IE2	EM3	1.07	9.02	7.95
IE3	EM3	1.00	8.86	7.86
IE1	EM4	1.27	9.89	8.62
IE6	EM2	0.97	9.59	8.63
IE2	EM4	0.86	9.40	8.54
IE7	EM1	0.82	9.30	8.48
IE2	EM5	0.57	8.39	7.81
IE8	EM2	0.69	9.03	8.35
IE3	EM4	0.73	9.24	8.51
IE1	EM5	0.60	8.87	8.28
IE4	EM3	0.72	9.39	8.67
IE9	EM1	0.71	9.41	8.70
IE3	EM5	0.39	8.23	7.83
IE4	EM5	0.45	8.76	8.31
IE7	EM2	0.41	8.81	8.40
IE9	EM2	0.42	8.96	8.55
IE5	EM3	0.06	7.99	7.93
IE8	EM1	0.39	9.50	9.11
IE10	EM2	1.30	9.36	8.06
IE5	EM4	0.06	8.38	8.32
IE10	EM1	0.06	8.52	8.46
IE11	EM1	0.10	8.66	8.56
IE6	EM3	1.34	9.64	8.29

図 5.6: EXE ステージの準同期化によるスケジュールと遅延の変化 (1)

入力端子名	出力端子名	遅延差	最大遅延	最小遅延
IE11	EM2	1.34	9.73	8.38
IE8	EM4	1.30	9.60	8.30
IE12	EM2	1.17	9.05	7.88
IE5	EM5	1.16	9.05	7.88
IE13	EM1	1.20	9.31	8.10
IE4	EM4	0.04	9.78	9.74
IE7	EM3	1.20	9.33	8.13
IE6	EM5	1.21	9.37	8.16
IE1	EM6	1.18	9.24	8.06
IE8	EM5	1.10	8.89	7.79
IE9	EM3	1.21	9.44	8.23
IE1	EM7	1.27	9.72	8.45
IE14	EM1	1.08	8.88	7.80
IE6	EM4	1.42	10.45	9.03
IE2	EM7	1.17	9.43	8.25
IE2	EM6	1.07	9.03	7.96
IE15	EM4	1.14	9.37	8.22
IE3	EM7	1.13	9.33	8.20
IE3	EM6	1.04	8.92	7.88
IE11	EM3	1.05	8.99	7.94
IE8	EM3	1.23	9.82	8.59
IE16	EM1	1.10	9.25	8.15
IE9	EM5	1.08	9.13	8.05
IE4	EM6	1.15	9.58	8.43
IE17	EM3	0.86	8.36	7.50
IE16	EM2	1.04	9.21	8.17
IE4	EM8	0.99	9.02	8.03
IE9	EM4	1.29	10.38	9.09
IE6	EM7	1.00	9.10	8.10
IE7	EM5	1.02	9.20	8.17
IE4	EM7	1.29	10.49	9.20
IE15	EM2	1.40	11.04	9.65
IE1	EM8	0.98	9.19	8.21
IE18	EM2	1.37	10.95	9.58
IE2	EM8	0.87	8.72	7.86

図 5.7: EXE ステージの準同期化によるスケジュールと遅延の変化 (2)

入力端子名	出力端子名	遅延差	最大遅延	最小遅延
IE16	EM4	1.02	9.50	8.47
IE17	EM2	1.18	10.21	9.03
IE15	EM1	1.50	11.69	10.19
IE16	EM8	0.85	8.79	7.94
IE18	EM1	1.47	11.60	10.13
IE5	EM7	0.95	9.28	8.33
IE12	EM1	1.29	10.86	9.57
IE5	EM6	0.80	8.69	7.88
IE9	EM6	0.79	8.64	7.85
IE16	EM3	0.77	8.56	7.79
IE17	EM1	1.27	10.85	9.58
IE3	EM8	0.88	9.08	8.20
IE16	EM5	0.73	8.49	7.76
IE16	EM7	0.86	9.10	8.24
IE16	EM6	0.73	8.59	7.85
IE19	EM4	0.77	8.76	7.99
IE6	EM6	0.91	9.38	8.48
IE1	EM9	0.67	8.36	7.69
IE18	EM3	1.08	10.21	9.13
IE5	EM8	0.82	9.05	8.23
IE19	EM1	0.72	8.62	7.90
IE15	EM3	1.12	10.43	9.31
IE19	EM7	0.69	8.57	7.88
IE13	EM4	0.89	9.47	8.59
IE20	EM10	0.69	8.60	7.91
IE19	EM2	0.65	8.42	7.77
IE15	EM5	0.99	10.02	9.04
IE11	EM4	0.86	9.52	8.66
IE10	EM5	0.71	8.85	8.14
IE16	EM9	0.58	8.28	7.70
IE6	EM9	0.68	8.76	8.08
IE19	EM6	0.49	7.95	7.45
IE19	EM3	0.51	8.04	7.53
IE19	EM8	0.49	7.91	7.43
IE20	EM11	1.24	11.32	10.08

図 5.8: EXE ステージの準同期化によるスケジュールと遅延の変化 (3)

入力端子名	出力端子名	遅延差	最大遅延	最小遅延
IE20	EM12	1.20	11.15	9.95
IE9	EM9	0.59	8.41	7.81
IE13	EM7	0.83	9.48	8.65
IE13	EM9	0.65	8.67	8.03
IE19	EM5	0.48	7.95	7.47
IE10	EM9	0.67	8.81	8.14
IE20	EM6	0.63	8.61	7.98
IE18	EM5	0.88	9.77	8.88
IE9	EM7	0.80	9.40	8.60
IE7	EM9	0.68	8.87	8.19
IE8	EM7	0.90	9.86	8.96
IE11	EM5	0.69	8.92	8.23
IE7	EM7	0.78	9.37	8.59
IE21	EM13	0.33	7.35	7.01
IE11	EM9	0.64	8.74	8.10
IE13	EM2	0.74	9.16	8.42
IE8	EM9	0.71	9.06	8.35
IE11	EM7	0.87	9.78	8.91
IE10	EM7	0.81	9.50	8.69
IE11	EM6	0.64	8.76	8.12
IE20	EM14	1.35	11.80	10.45
IE20	EM15	1.30	11.75	10.45
IE10	EM4	0.88	9.84	8.96
IE10	EM6	0.66	8.84	8.19
IE20	EM16	1.10	10.83	9.73
IE7	EM8	0.75	9.25	8.50
IE11	EM8	0.81	9.52	8.71
IE9	EM8	0.66	8.86	8.20
IE7	EM4	0.89	9.89	9.01
IE7	EM6	0.63	8.73	8.10
IE10	EM8	0.73	9.19	8.46
IE8	EM6	0.73	9.18	8.45
IE13	EM5	0.62	8.68	8.07
IE20	EM17	1.17	11.18	10.01
IE8	EM8	0.82	9.62	8.80

図 5.9: EXE ステージの準同期化によるスケジュールと遅延の変化 (4)

入力端子名	出力端子名	遅延差	最大遅延	最小遅延
IE22	EM18	0.27	7.13	6.87
IE20	EM5	0.38	7.65	7.27
IE20	EM19	1.04	10.66	9.61
IE20	EM20	1.18	11.29	10.10
IE10	EM3	0.66	8.92	8.27
IE23	EM21	0.23	7.03	6.79
IE13	EM8	0.73	9.29	8.55
IE4	EM9	0.80	9.60	8.80
IE20	EM7	0.95	10.29	9.34
IE20	EM22	1.00	10.50	9.50
IE20	EM21	0.99	10.46	9.47
IE24	EM19	0.20	6.92	6.72
IE20	EM9	1.06	10.79	9.73
IE25	EM14	0.19	6.87	6.69
IE13	EM3	0.63	8.85	8.23
IE26	EM22	0.19	6.87	6.69
IE27	EM15	0.18	6.87	6.68
IE28	EM10	0.18	6.87	6.68
IE20	EM8	0.48	8.21	7.73
IE29	EM12	0.18	6.86	6.68
IE13	EM6	0.62	8.83	8.21
IE30	EM11	0.18	6.85	6.67
IE31	EM17	0.17	6.84	6.66
IE32	EM20	0.17	6.84	6.66
IE33	EM16	0.17	6.82	6.65
IE6	EM8	0.72	9.34	8.62

図 5.10: EXE ステージの準同期化によるスケジュールと遅延の変化 (5))

第6章　まとめ

本研究では準同期式パイプラインの構成アルゴリズムを提案した.

第 1 章では, ウェーブパイプラインならびに準同期式回路と本研究で提案する準同期パイプラインの関係を説明した. 第 2 章では, ウェーブパイプラインならびに準同期式回路, 準同期パイプラインの概念ついて説明した. 第 3 章では, 準同期パイプラインの動作と, 制約グラフとクロック周期の関係について説明した. 第 4 章では, 本研究で提案する準同期式パイプラインを構成するアルゴリズムならびに遅延バッファ挿入アルゴリズムについて説明した. 第 5 章では, 準同期パイプラインプロセッサ WAVIST の試作について説明した. またバッファ挿入実験によりパイプラインが高速化できることを示した.

謝辞

本研究を進めるにあたり, 終始熱心な御指導を頂きました日比野靖教授に深く感謝いたします.

また, 研究を進める上で御助言を頂きました本学の田中清史助教授, 堀口進教授, 宮崎純助手, ならびに日比野・田中研究室の皆様に感謝いたします. 特に, ウェーブパイプラインプロセッサ **WAVIST** の設計・製作については, 松居昭宏君の協力がなければ, これだけの短期間で完成させる事は到底不可能でした. この場を借りてお礼申し上げます.

また, 本チップ試作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ローム (株) および凸版印刷 (株) の協力で行われたものです.

さらに, 以下の方々に深くお礼を申し上げます.
東京工業大学の高橋篤司助教授からは, クロックスケジューリングとバッファ挿入アルゴリズムについて, 多くの御助言を頂きました. 同研究室の石島誠一郎君と内海哲章君からは, チップ試作に関する資料を多数御提供頂きました.
金沢大学の深山正幸助手からは, CAD ツールの利用に関して貴重な御助言を頂きました.
東芝株式会社の依田友幸氏からは, クロックスケジューリング計算プログラムを御提供頂きました.
本学の金子峰雄教授ならびに高島康裕助手, 北九州市立大学の梶谷洋司教授, マイクロアーク株式会社の村田洋氏, 株式会社富士通研究所の河村薫氏ならびに畔上謙吾氏, 熊本大学の高岡裕氏からは, 研究を進める上で直接または間接に多くの御助言を頂きました.
椎間板ヘルニアに苦しめられた際にも, 下崎整形外科病院の皆様の御協力のおかげで研究を進めることができました.

参考文献

[1] O. Hauck, M. Garg, and S. A. Huss. Efficient and Safe Asynchronous Wave-Pipeline Architectures for Datapath and Control Unit Applications. Proceedings 0th Great Lakes Symposium on VLSI, pp. 38–41, 1999.

[2] A. Takahashi and Y. Kajitani. Performance and reliability driven clock scheduling of sequential logic circuits. Proc. ASP–DAC '97, pp. 37–42, 1997.

[3] 依田友幸, 佐々木哲雄, 高橋篤司. 準同期式回路の高速化のための修正コストを考慮したクロックスケジューリング. 電子情報通信学会技術報告 (VLD99–36), Vol. 99, No. 108, pp. 45–53, 1999.

[4] T. Yoda, and A. Takahashi. Clock Period Minimization of Semi-Synchronous Circuits by Gate-Level Delay Insertion. IEICE Transactions on Fundamentals, Vol. E82-A, No. 11, pp. 2383–2389, 1999.

[5] 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.

[6] 大石亮介, 高橋篤司. 準同期式回路における最小クロック周期を求めるアルゴリズムの高速化. 電子情報通信学会技術研究報告 (VLD99–125), Vol. 99, pp. 63–68, 2000

[7] 大戸友博, 石島誠一郎, 内海哲章, 畔上謙吾, 高橋篤司. 準同期式設計法を用いたプロセッサ設計. 電子情報通信学会技術報告書 (VLD2000–101), Vol. 100, No. 437, pp. 45–53, 2000.

[8] 大石亮介, 松居昭宏, 日比野靖. 厳密な遅延評価による準同期パイプラインプロセッサの設計. 情報処理学会研究報告 (2001–ARC–145), Vol. 2001, No. 116, pp. 63–66, 2001.

[9] VDEC 監修, 浅田邦博編. ディジタル集積回路の設計と試作. 培風館, 2000.

[10] 内藤 郁之. 低消費電力プロセッサ設計に関する研究. 北陸先端科学技術大学院大学修士論文, 2001.

[11] R. M. Karp. A Characterization of the minimum cycle mean in a digraph. Discrete Mathematics, Vol. 23, pp. 309–311, 1978.