Vol.2014-ARC-211 No.9
2014/7/28

動的タイム・ボローイングを可能にするクロッキング方式のための二相ラッチ生成アルゴリズム

津坂 章仁[1,a)]　谷川 祐一[1]　広畑 壮一郎[1]　五島 正裕[2]　坂井 修一[1]

概要：半導体プロセスの微細化に伴う回路遅延のばらつきの増加が，回路設計における大きな問題となりつつある．ばらつきが増大していくと，従来のワースト・ケースに基づいた設計手法は悲観的になりすぎる．そのため，ワースト・ケースより実際に近い遅延に基づいた動作を実現する手法が提案されている．そのような手法の 1 つとして，我々の研究室では動的タイム・ボローイングを可能にするクロッキング方式を提案してきた．このクロッキング方式は，動的なばらつき対策手法である動的タイミング・フォールト検出を二相ラッチのクロッキング方式に組み合わせることによって実現される．これにより，動作時にステージ間で回路遅延を融通し，実効遅延に近い速度で動作させることが可能になる．
本稿では，従来の回路に対してこのクロッキング方式を適用する手法の中でも特に，二相ラッチ化のアルゴリズムについて詳しく述べ，実際に適用した場合の評価を示す．

キーワード：ばらつき，TF 検出，Razor，2 相ラッチ，タイムボローイング

1. はじめに

半導体プロセスの微細化に伴って，素子遅延の**ばらつき**が大きな問題となりつつある．ここで特に問題とされているのは，チップ間に跨るシステマティックなばらつきではなく，チップ内のランダムなばらつきである．これは，トランジスタや配線のサイズが原子のサイズに近づくために生ずる本質的な問題であり，原理的に避けえない．

ばらつきが増大していくと，従来の**ワースト値**に基づいた設計手法は悲観的になりすぎる．微細化が進むにつれて，ばらつきの増大により，平均値とワースト値の差は広がっていく．その結果，LSI の設計上の動作速度が向上しなくなってしまうことも考えられる．

そのため，ワースト・ケースより実際に近い遅延に基づいた動作を実現する手法が提案されている．設計段階において遅延のばらつきを統計的に扱う **SSTA**（Statistic Static Timing Analysis : 統計的静的タイミング解析）などもその一例である．SSTA によれば，ワースト・ケースほど悲観的ではない遅延見積もりを行うことができる．

動的タイミング・フォールト検出・回復

SSTA のように設計時に用いられる手法は静的な方法ということができるが，それに対して，動作時にタイミング・フォールトを検出し回復する手法は動的な方法ということができる．

タイミング・フォールト（以下，**TF** と略す）は，遅延の動的な変化によって設計者の意図とは異なる動作が引

[1] 東京大学
The University of Tokyo
[2] 国立情報学研究所
National Institute of Informatics
[a)] tsusaka@mtl.t.u-tokyo.ac.jp

き起こされる過渡故障である．想定した動作条件内のワースト・ケースでも動作するように設計するのがワースト・ケース設計であるので，そのように設計・製造された LSI では，原則 TF は発生しない．実際に TF が起こるのは，想定した動作条件を外れた場合，例えば，冷却ファンや温度センサの故障による熱暴走などに限られる．

Razor [1] は，TF を検出する機能を持つ．このような回路に **DVFS** (Dynamic Voltage and Frequency Scaling)[2] を組み合わせると，見積もりではなく実際の遅延に応じた動作を実現することができる．**V**（Voltage : 電源電圧）を下げる，または，**F**（Frequency : 動作周波数）を上げると，回路はいずれ TF を生じ，検出される．検出直前の V–F が，そのチップのその時の動作環境における実際の遅延に応じた V–F である．後は，TF が頻発しないように V–F を調整すればよい．

既存手法の限界

このような TF 検出手法は実際には，プロセスばらつきに対する直接的な解法にはなっていない．クリティカル・パスが活性化される確率が 1/100 程度である [3] とすると，クリティカル・パスの遅延より V–F を改善することは現実的ではない．クリティカル・パスの遅延以上に V–F を改善すると，100 サイクルに 1 回は TF を生じ，回復のペナルティを被るからである．

ここで，クリティカル・パスの遅延にはプロセスばらつきの影響が含まれていることに注意されたい．チップ内の各クリティカル・パスの遅延はランダムばらつきにより増減するが，チップの V–F は最も増大した遅延によって決まるのである．

結局，TF 検出手法の効果とは，DVFS のマージンを削減するに過ぎないということができる．

動的タイム・ボローイング

そこで我々の研究室では，実効遅延の平均に近いサイクル・タイムでの動作を可能とするクロッキング方式を提案した [4]．提案方式は，端的に言えば，TF 検出と二相ラッチを組み合わせたもので，このことにより**動的タイム・ボローイング**が可能になる．3.2 節で詳しく述べるが，従来からある二相ラッチ方式で可能になるタイム・ボローイングは，言わば**静的タイム・ボローイング**と呼べるもので，設計時にステージ間で遅延を融通するものである．動的タイム・ボローイングとは，実行時に実効遅延がサイクル・タイム以上に伸びてしまった場合，この超過分を次のステージに持ち越すというものである．次のステージの実効遅延が短ければ，この超過分は相殺される．このことにより実効遅延の分散を吸収し，実効遅延の平均に近いサイクル・タイムでの動作が可能となるのである．

本稿の内容

我々の研究室では，動的タイム・ボローイングを可能にするクロッキング方式を既存回路に適用するための手法を提案するとともに，適用を自動で行うツールの開発を進めている [5] [6]．しかしこれまでに提案した適用手法における二相ラッチ化は，要件やアルゴリズムなどに不十分な点がある．

そこで本稿では，二相ラッチ化の要件と目標を改めて明確に定義するとともに，二相ラッチ化を行うアルゴリズムを提案する．

構成は以下の通りである．まず 2 において，従来のクロッキング方式について説明する．次に 3 章では，動的タイム・ボローイングを可能にするクロッキング方式の回路構成・動作と，従来の回路に対する適用手法について概要を説明する．そして 4 章は，適用手法における二相ラッチ化の要件・目標を定義し，二相ラッチ化のアルゴリズムについて詳しく述べる．5 章では，提案するアルゴリズムによって実際の回路を二相ラッチ化した時の評価を示す．最後に 6 章では，まとめと今後の課題について述べる．

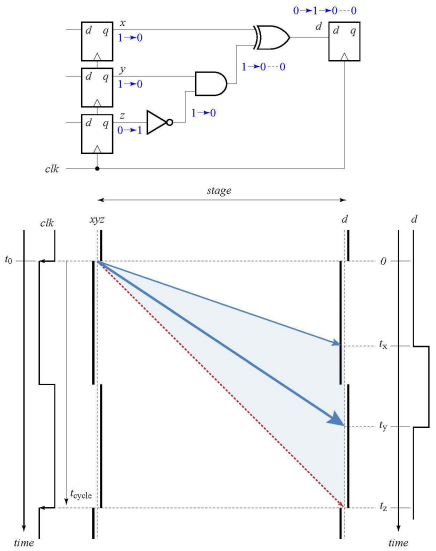

図 1 タイミング・ダイアグラム (t-diagram) と実効遅延

2. 従来のクロッキング方式

本章では 2.1 でクロッキング方式の説明に必要なタイミング・ダイアグラムについて説明し，2.2 で本方式でも用いている Razor の回路構成と動作を述べる．

2.1 タイミング・ダイアグラム

クロッキング方式の動作を説明するにあたって，**タイミング・ダイアグラム** (**t-diagram**) と我々が呼んでいる図を使って説明する．**図 1**（上）の回路において，信号が伝わる様子を図 1（下）に示す．この下の図が t-diagram である．通常のタイム・チャートが論理値-時間の 2 次元を持つのに対して，t-diagram は時間-空間の 2 次元を持つ．

通常のタイム・チャートでは，右方向が時間を，上下方向が論理値を表す．タイム・チャートは，論理値の時間的

変化を表現するが，1 本の波形で表すことができるのは回路の特定の 1 点の振る舞いに限られる．複数の点にまたがる動きを把握するためには，複数の波形を並べなければならない．

それに対して t-diagram は，下方向が時間を，右方向が回路中を信号が伝わって行く方向を表し，時間の経過につれて回路を信号が伝わっていく様子を俯瞰することができる．

図 1（上）に示す回路で，時刻 $t = 0$ に 3 つの FF の出力 (x, y, z) が $(1, 1, 0)$ から $(0, 0, 1)$ に遷移したとする．x，y，z から d に至るパスの遅延をそれぞれ t_x，t_y，t_z とすると，ロジックの出力 d は，時刻 $t = t_x$，t_y において $0 \to 1 \to 0$ と遷移する．z から d に至るパス上を伝わる信号は，y から d に至るパスの信号によってマスクされるため，時刻 $t = t_z$ には出力は変化しないことに注意されたい．同図の右端にある波形が，d における通常のタイム・チャート（を右に 90° 回転したもの）である．

パスの活性化

図 1 のように t-diagram では，ロジックの入力において入力が変化した時刻から，同ロジックの出力において出力が変化した時刻までを直線矢印で結ぶことによって，信号の伝わる様子を表す．

図 1 に示した例では，前述したように，z から d に至るパスを通る信号は途中でマスクされるため，時刻 $t = t_z$ においては出力 d は変化しない．パスを通った信号によって実際にロジックの出力が変化したとき，その信号によってそのパスが**活性化**したと言う．

t-diagram では，パスを活性化した信号の伝達を実線矢印で表す．活性化しなかった場合には，途中でマスクされた段階で信号は物理的には消失しているが，仮想的に点線矢印で表すことにする．

特に，t-diagram ではロジックのクリティカル・パスが活性化した時の信号に対応する矢印を赤で描くこととし，その角度を 45° と決める．この約束により，あるロジックのクリティカル・パスの遅延は，t-diagram 上のロジックに対応する領域の横幅によって表現することができる．

実際のロジックでは，ばらつきのため，遅延は連続的に変化する．そのため，矢印の存在範囲は，ロジックの最小遅延の矢印とクリティカル・パスの遅延の赤矢印に上下を挟まれた領域となる．

t-diagram では，網掛けを施してこの領域を示す．

実効遅延

あるロジックにおいて最後に活性化されたパスの遅延を，このロジックの**実効遅延**と呼ぶことにする．図 1 の場合，時刻 $t = t_z$ においてクリティカル・パスを通った信号が到着するはずだが，マスクされたため，ロジックの出力 d は変化しない．この場合，実効遅延は t_y となる．

時刻 $t = t_y$ において出力 d が変化した時には実効遅延

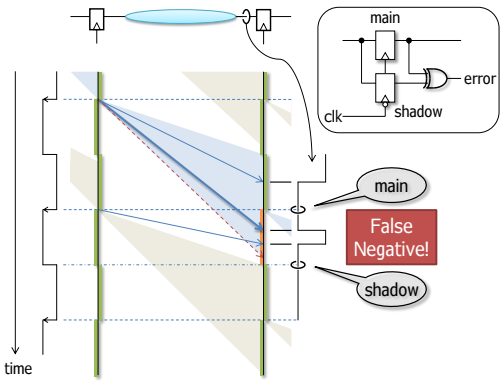

図 2 Razor FF の回路構成とショート・パス問題

が t_y であることは分からない．時刻 $t = t_z$ において d が変化しなかったことを見て初めて t_y であったことが分かる．このように，実効遅延は事後的に分かることに注意されたい．

t-diagram では実効遅延に対応する矢印を太実線で表す．クリティカル・パスが活性化した場合には，45° の赤矢印を更に太くして表す．

入力ばらつきと実効遅延

ロジックへの入力の変化の仕方によって出力の変化の仕方も様々であり，どのパスが最後に活性化されるかは毎サイクル異なる．つまり実効遅延は，入力の変化の仕方によって大きくばらつく．このことを**入力ばらつき**と呼ぶ．

入力ばらつきは，他のばらつきに比べて非常に大きい [3]．出力が直前のサイクルから変化しなかった場合には，実効遅延は実質 0 となる．すなわち，入力ばらつきは，0 からクリティカル・パス遅延まで変化するのである．他のばらつきによる遅延の変化が高々数十 % 程度であることを考えると，この変化の度合いは非常に大きいと言える．また，ロジックの出力の変化率は 1/2 程度であることが知られている．すなわち，1/2 程度の高い確率で実効遅延は 0 となることにも注意する必要がある．

2.2 Razor FF

回路構成とタイミング・ダイアグラム

図 2（右上）に，Razor FF の回路構成を示す [1]．Razor FF は，通常の FF（Main FF）と，Shadow Latch によって構成される．Shadow Latch には，Main FF へのそれより位相の遅れたクロックが供給されており，Main FF と Shadow Latch でそれぞれ 1 回ずつ，信号のサンプリングを行う．図 2 では，0.5cycle 遅らせたクロックが Shadow Latch に供給されている．このため，TF が発生して Main FF のサンプリング・タイミングまでにクリティカル・パスの信号が到達しなくても，Shadow Latch はクリティカル・パスの値をサンプリングすることができる．Main FF と Shadow Latch の値を比較して，異なっていれば TF として検出する．

Razor FF では，クリティカル・パスの遅延に対応する 45° の赤線が検出ウィンドウの下端までに到着すれば，TF として処理することができる．したがって，サイクル・タイムに対する検出ウィンドウの割合を α とすると，最大遅延制約は $(1+\alpha)$cycle/1stage となる．ここでは，Shadow Latch に 0.5cycle 遅れたクロックが供給されているため α は 0.5 となり，最大遅延制約は 1.5cycle/1stage である．

また，実効遅延に対応する太矢印が検出ウィンドウの上端より先に到着していれば，TF とならない．すなわち，TF とならない遅延制約は通常の FF を用いた回路と同じく最大 1cycle/1stage となる．

Razor を本方式に適用するにあたり，ラッチに逆相で動作する Shadow Latch とサンプリングされた値を比較する XOR ゲートを追加する．これは Razor で用いられる Razor FF の Main FF をラッチに置き換えた構造となる．ここでは便宜上，Razor Latch と呼ぶことにする．

ショート・パス問題

クリティカル・パスのおおむね半分以下の遅延を持つパスを**ショート・パス**と呼ぶ．セットアップ/ホールド・タイム違反など，ショート・パスの活性化が原因で遅延制約が満たされない問題を**ショート・パス問題**と呼ぶ．

図 2 は Razor FF のショート・パス問題を図示した t-diagram である．Razor FF は，Main FF と Shadow Latch の値を比較することで TF を検出するが，Shadow Latch が正しい値をサンプリングするためには，ロジックのショート・パスを通った信号が Shadow Latch のサンプリング・タイミングよりも後に到達しなければならない．さもないと，あるサイクルにおいてショート・パスを通った信号が，前のサイクルの遅れた信号と混ざり，その結果 Shadow Latch が本来の値を正しくサンプリングできない可能性がある．

このため，Razor FF には特有の最小遅延制約が存在する．サイクル・タイムに対する検出ウィンドウの割合を α とすると，最小遅延制約は αcycle/1stage となる．ショート・パスに遅延素子を挿入するなどして，ロジックの最小遅延を αcycle 以上にする必要がある．

3. 動的タイム・ボローイングを可能にするクロッキング方式

本章では，我々の研究室が提案している，動的タイム・ボローイングを可能にするクロッキング方式（以下，本方式と呼ぶ）の説明をする．これは，二相ラッチ方式と TF 検出を組み合わせたものであり，**動的タイム・ボローイング**を可能にしている．3.1 ではこの手法の回路構成を説明する．次に 3.2 で動的タイム・ボローイングを説明する．最後に 3.3 にてこの手法を従来の回路に適用する手法について述べる．

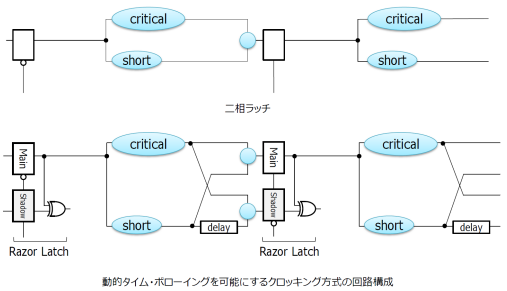

図 3 二相ラッチ方式（上）と動的タイム・ボローイングを可能にするクロッキング方式（下）の回路構成

3.1 回路構成

図 3 は二相ラッチ方式と本方式の回路構成を比較したものである．

図 3（上）は二相ラッチの回路の概略図であり，ロジックのショート・パスとクリティカル・パスとが，あるゲート（図中〇印）で合流した後，ラッチに接続されている．二相ラッチは，マスタ–スレーブ構造を持つ FF を構成する 2 つのラッチのうちの 1 つを，ロジックの中ほどに移動したものと理解することができる．単相 FF 方式の 1 ステージに相当するロジックを，このラッチが二分する形になる．

図 3（下）は本方式を適用した回路の概略図である．本方式では TF 検出のために，2.2 で示した Razor[1] を用いている．動的タイム・ボローイング方式においても 2.2 で示したような Razor FF のショート・パス問題が起きないよう，ロジックのショート・パスに遅延を挿入する必要がある．遅延を挿入する必要がある箇所は，Shadow Latch に至るショート・パスだけである．これは，TF 検出時は Shadow Latch が開き，Main Latch が閉じている状態であり，Main Latch は前サイクルの値を保持しているため，次サイクルのショート・パスの活性化の影響を受けないからである．このため，ショート・パスとクリティカル・パスの合流するゲートを二重化し，Shadow Latch に至るショート・パスにのみ遅延を挿入する．Main Latch に至るショート・パスには遅延が挿入されていないので，ロジックの遅延分布がクリティカル・パスの遅延の方に偏ることはない．

3.2 動的タイム・ボローイング

本方式では二相ラッチ方式によって本来的には利用可能であったこのラッチの開いている期間を，TF 検出を設けることにより利用できるようにしている．ラッチが開いてから 2 ステージ連続してロジックのクリティカル・パスが活性化した場合が TF 検出限界となるようにサイクル・タイムを定める．このようにすると，サイクル・タイムの遅延によって定められるワースト遅延の境界が t-diagram 上で階段状となり，サイクル・タイムを詰めるように短縮することが可能となる．これにより，ロジック上の全遅延の存在領域をラッチの開いている区間にも広げることがで

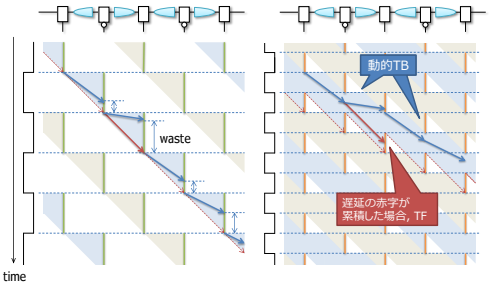

図 4　二相ラッチ方式（左）と動的タイム・ボローイングを可能にするクロッキング方式（右）の t-diagram

き，動作時に各ステージで実効遅延を融通することが可能となる．

動作時に実効遅延を各ステージで融通する，この時間の貸し借りのことを**動的タイム・ボローイング**と呼ぶ．

図 4（右）の t-diagram において，信号が伝わる様子を示す直線矢印が，複数ステージ間に渡ってラッチの開いている区間を通って繋がっているのは，動的タイム・ボローイングの効果を表しているといえる．

本方式のサイクル・タイムは 0.5 ステージ分のロジックのサイクル・タイムの遅延によって決定できるので，最大遅延制約は，1cycle/0.5stage となる．

3.3　適用手法

我々の研究室では以前，従来の回路に対して動的タイム・ボローイングを可能にするクロッキング方式を手作業で適用し，FPGA 上に実装して評価を行った [4]．この時の回路は，リプルキャリー・アダーやキャリールックアヘッド・アダーを用いたアップ・カウンタのような比較的小規模な回路であった．

しかし，プロセッサのように回路規模が大きいものに対して手作業で適用するのは，不可能に近い．なぜならば，このクロッキング方式の適用には Razor の挿入に代表されるように，回路のすべてのパスの遅延を考慮する必要があるからである．

そこで，我々の研究室では動的タイム・ボローイングを可能にするクロッキング方式を既存回路に適用するための手法を提案するとともに，適用を自動で行うツールの開発を進めている [6]．ここではその適用手法について，続く 4 章で述べる二相ラッチ化の理解に必要な部分に焦点を絞って概要を述べる．

適用手法は，グラフ構造への変換，ステージ分割，二相ラッチ化，TF 検出機構の付与という 4 つのステップから構成される．

(1) グラフ構造への変換

適用ツールではまず，単相 FF 方式で設計された Verilog HDL 形式の回路を読み込み，有向グラフ構造へと変換する．変換されたグラフにおけるノードは，元の回路の FF や入力/出力ポート，ゲートといった回路素子を表し，グラフのエッジは回路素子間の配線 (ネット) を表す．グラフにおけるエッジの方向は，元の回路における信号の伝達方向と対応付ける．このグラフに対して本クロッキング方式を適用し，適用後にグラフから Verilog HDL への書き戻しを行う．

(2) ステージ分割

FF や入力/出力ポートに挟まれたロジックのことを，ステージと呼ぶことにする．二相ラッチ化や Razor の挿入といった本クロッキング方式の適用は，各ステージに対して行われる．そのために，回路全体のグラフをステージ単位のグラフに分割する．

(3) 二相ラッチ化

ステージ単位に分割したグラフに対し，ステージ内のロジックに逆相ラッチを挿入し，FF を正相ラッチに置き換える．続く 4 章で，二相ラッチ化について詳しく述べるとともに，そのアルゴリズムを提案する．

(4)TF 検出機構の付与

二相ラッチ化を行ったグラフに対し，Razor への変更，遅延素子の挿入，ゲートの二重化を行うことで TF 検出機構を付与する．

4.　二相ラッチ化アルゴリズム

本章では，動的タイム・ボローイングを可能にするクロッキング方式の適用手法のための二相ラッチ化アルゴリズムを提案する．まず 4.2 節で二相ラッチ化の要件と目標を明確にし，続く 4.2 節でアルゴリズムの内容を，4.3 節で挿入候補パスのオーダリングについて述べる．4.4 において，アルゴリズムで用いるコストについて説明する．

4.1　要件と目標

適用手法における二相ラッチ化として，ステージ単位に分割したグラフに対し，ステージ内のロジックに逆相ラッチを挿入し，FF を正相ラッチに置換する．二相ラッチ化において満たさなければならない要件と達成したい目標を以下に述べる．

4.1.1　要件

ステージ単位のグラフ上の全てのパスにおいて，逆相ラッチがそれぞれ 1 つだけ挿入されている必要がある．

ステージのグラフにおいて，入力側境界ノードから出力側境界ノードまでの経路を**パス**と呼ぶことにする．二相ラッチ化として境界ノードを正相ラッチに変換するので，パス上に逆相ラッチが存在しない場合や逆相ラッチが複数存在する場合は，二相ラッチ方式の回路として正しく動作しない．回路が二相ラッチ方式として正しく動作するためには，全てのパスに逆相ラッチがそれぞれ 1 つだけ挿入されるようにしなければいけない．ただし，境界ノードが入出力ポートであるパスには逆相ラッチを挿入しなくても

よい．

4.1.2 目標

目標は，以下の 2 点である：

(1) 各パスの遅延が二分に近くなるようなステージ分割になること

(2) 挿入する逆相ラッチの数を少なくすること

以下，それぞれについて詳しく述べる．

(1) 各パスの遅延が二分に近くなるようなステージ分割になること

本クロッキング方式の効果を最も発揮させるためには，二分されたステージの各クリティカル・パス遅延ができるだけ均等にならなければいけない．動的タイム・ボローイングを可能にするクロッキング方式では，逆相ラッチの挿入により二分されたステージのクリティカル・パス遅延によってサイクル・タイムが定まる．そのためクリティカル・パス遅延が均等に二分されない場合は，遅延の大きい方によってサイクル・タイムが制限されてしまい，最大削減幅である半減までサイクル・タイムを削減できない．またクリティカル・パス遅延が均等に二分される場合でも，クリティカル・パス以外のパスにおいて二分されたクリティカル・パス遅延の最大値を超えるような遅延分割がなされてしまうと，同様にサイクル・タイムをクリティカル・パス遅延の半分まで削減できない．

また，適用手法では二相ラッチ化の後に TF 検出機構の付与を行うので，それを考慮して二相ラッチ化を行う必要がある．考慮すべきことは Razor への変更と，2.2 で示した Razor FF のショート・パス問題を解消するための遅延素子の挿入の 2 つであり，どちらも回路素子を追加することになるので回路面積が増加する．Razor への変更では，クリティカル・パス遅延 d_{cp}，検出ウィンドウ幅 W に対してパスの遅延が $d_{cp} - W$ 以上であれば Razor 化を行う．遅延素子の挿入では，パスの遅延が W 以上になるように遅延挿入を行う．パスの遅延ができるだけ均等になるように二分されていれば，どちらの工程も行われることが少なくなるので，回路面積の増加を抑えられる．

(2) 挿入する逆相ラッチの数を少なくすること

動的タイム・ボローイングを可能にするクロッキング方式を適用することによって生じる回路面積の増加をできるだけ抑えるため，挿入する逆相ラッチの数を少なくする．同じ回路でも，たくさんのパスが通るネットに対して挿入することで，全体として挿入する逆相ラッチの数は少なくなる．

4.2 アルゴリズム

アルゴリズムの概要は以下の通りである．

挿入対象ネットを探索するにあたり，重み付き探索木 T を作成する．T の各ノードは，逆相ラッチの挿入対象ネットを決定していく過程において，既に挿入対象となっているネットと残っている挿入候補パスと挿入候補ネットを保持する．そして，挿入対象ネットを 1 つずつ決定していったときのステージの状態を，T におけるノードの根から子へと順に対応付ける．T のノードについて，根は挿入対象ネットが存在しない状態に対応し，根から子へと 1 つずつノードを経るにつれてそのノードの挿入対象ネットは 1 つずつ増えていき，葉は挿入対象ネットを決定しきった状態に対応する．逆に，根は全てのパスを挿入候補パスとして持ち，根から子へ進むに従って挿入候補パスが減っていき，葉においては挿入候補パスが存在しない状態になる．

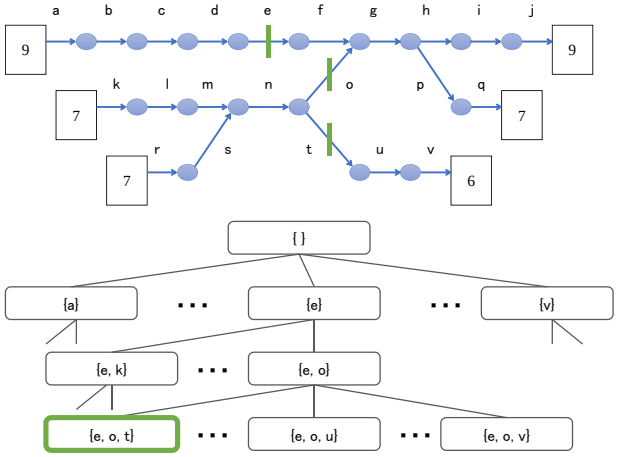

図 5 回路のグラフと重み付き探索木

そして，ネットに逆相ラッチを挿入するのに評価関数を用いて計算したコストが必要だとみなし，各ノードに重みとしてコストを設定する．このコストについては 5.2 にて詳しく述べるが，で挙げた目標に近いほど低くなるように設定する．こうすることで，そのコストの総和を最小化するような逆相ラッチの挿入対象ネットの組み合わせの探索は，重み付き探索木の根から葉への最短経路問題に帰着する．

以上で述べたことをまとめると，探索木の各ノードは挿入対象ネット，挿入候補パス，挿入候補ネット，コストを持つことになる．

具体的な探索木の様子が**図 5** である．図の上が回路で，下がその回路から作成された重み付き探索木である．上について，白い四角が FF を，青い丸が素子を，緑の線が挿入対象となったネットを表している．各 FF にはその FF から伸びる最も長いパスの長さが書いてある．例えば，図 5 だと，上の回路の緑のネットが選ばれた場合に，下の探索木では緑で示したノードが対応する．

この探索木を用いた最短経路問題によって得られる解が要件と目標が満たされることを説明する．

要件

全てのパスについて，各パス上に挿入対象ネットがな

い場合は挿入候補パスとして残っており，挿入されるまで続くので全てのパス上に挿入対象ネットが存在する．また，パス上に挿入対象ネットが加わった時点で，そのパス上のネットは挿入候補から外れるため，各パス上に複数の挿入対象ネットが存在することはない．これらにより要件は満たされる．

目標

逆相ラッチの挿入に必要なコストとして，目標に近いほどコストが低くなるようなコスト関数を設計する．つまり，挿入対象ネットがパスの遅延を均等に二分するほど，挿入対象ネットが含まれるパスの遅延の小さいほど，挿入する逆相ラッチの数が少ないほど，コストが小さくなるように設定する．よって，そのコスト関数の総和を最小化する挿入対象ネットを探索することで，全体として最も目標に近いものを探索できる．

4.3 探索候補パスのオーダリング

子のノードを作成にあたっては，挿入候補パスから最も長いパスを選択し，そのパス上に存在する探索候補ネットを列挙する．挿入候補パスのうち最長距離のパスを選ぶのは，パスの距離の大小によって挿入候補ネットを選んだ時に取り除かれる挿入候補のパス数が違うことに着目して，T の枝数を少なくして探索を効率的に行うためである．

探索で求める逆相ラッチの挿入対象ネットは，ステージにおけるネットの組み合わせであり，その順列は考慮する必要がない．よってコスト関数が，逆相ラッチの挿入対象とするネットの順番に依存しない限り，挿入対象とするネットを選ぶ順番は自由に変えても良い．ゆえに，全てのパスに逆相ラッチを挿入する必要があることを考えると，挿入候補ネットを選んだ時に挿入候補のパスが減りやすい，すなわち距離の長いパス上の挿入候補ネットから順に逆相ラッチの挿入対象ネットとしていくことにする．これにより T の探索が終わるまでの枝数を少なくできるので，比較的効率的に探索を行うことができる．

4.4 コスト関数

コスト関数の基本的な考え方として，4.1.2 項で述べた目標を満たすほど値が小さくなるように設計する．

目標 (1) に関しては，挿入対象ネットがパスの中央から近いほど，挿入対象ネットが存在するパスのうち最長のものの長さが短いほどコストが小さくなるようにする．挿入対象ネットがパスの中央から離れるほどクリティカル・パスの長さに影響を与えやすい．さらにパスの距離が長いほど，クリティカル・パスの長さに影響を与えることが多いことに加え，適用手法の TF 検出機構の付与においては Razor への変更や遅延素子の挿入が行われることが多いことが考えられる．

目標 (2) に関しては，(1) で定めたようなコストの総和の最小値を求めることで，おおよそ達成できると期待できる．コストはラッチの挿入ごとに増加するので，ラッチの挿入箇所が増えればコストの総和も増える．つまり，コストの総和が最小になるように求めれば，ラッチの数も少なくなると考えられる．

また先に述べたように，距離の長いパス上から挿入対象ネットを選んで効率良く探索を行うため，ある逆相ラッチの挿入対象ネットの組み合わせにおいて，挿入対象ネットの選び方の順序に関わらずコスト関数の総和は一定でなければならない．そうでないならば，距離の長いパス上のネットから順に逆相ラッチの挿入対象ネットとしていくときに見つかる解が最適解であることを保証できない．

この考えに基づき，あるネットに逆相ラッチを挿入するのに必要なコスト関数の式は，そのネットに逆相ラッチを挿入することによって挿入候補から除外される最長のパスの距離を d_{path}，そのパス上における挿入対象ネットの距離を d_{target}，N を定数と表すとすると，

$$(|d_{target} - d_{path}| + 1)^N * d_{path}$$

と設定した．$|d_{target} - d_{path}|$ がパスの中央からの距離を表し，それに 1 を加えた値を累乗した上でパスの距離 d_{path} を乗じている．これは，中央からの距離が 1 離れた位置でもコストが十分に大きくなるようにするためである．

コストの総和の最小値を求める上では N の値を大きくしていくと，パスの中央からの距離が遠い挿入候補のコストが非常に大きくなっていくので，そのような挿入位置はより選ばれにくくなる．

また，逆相ラッチの挿入によって除外される最長のパスについてコストを考慮しているので，挿入対象ネットの選び方の順序によってコスト関数の値が変わることはない．

5. 二相ラッチ化アルゴリズムの評価

ここでは，4 章で述べた手法を実際の回路に適用した時の評価について述べる．評価にあたって用いた探索アルゴリズムを 5.1 にて示す．5.2 では，回路に適用する場合に探索するノードの数と探索にかかる時間を示す．5.3 にて，現状の問題点について述べる．

5.1 評価に用いた探索アルゴリズムについて

今回，探索木 T の探索アルゴリズムとして最良優先探索を用いた．最良優先探索にするのは，T の探索に必要な計算量を抑えるためである．計算量を抑えるために求められることとして，探索アルゴリズムにおいて最初に見つける解が必ず最適解であるという最適性が挙げられる．先に述べたようにラッチを挿入する箇所の組み合わせを考慮すると，T は非常に多くのノードを持つと考えられる．そのため，最適解を求める場合に最適性のないアルゴリズムを用いると計算量が莫大に増え，現実的には最適解を求めるこ

とが不可能だと考えられる．

最良優先探索とは，ある規則に従って最も望ましいノードを展開していくグラフ探索アルゴリズムである．最良優先探索においては，まず任意のノード n を正の数に対応させる評価関数 $f(n)$ を定義し，スタートノードだけを含むオープンリストを作成する．そして，オープンリストから $f(n)$ が最も望ましいノードを除外し，除外したノードから辿れるノードを作成し，オープンリストに追加するということを繰り返していく (このことを展開と呼ぶ)．オープンリストから最も望ましいノードを選ぶ操作を Visit という．ゴールノードを展開したら探索を終了する．

T に最良優先探索を適用するときには，スタートノードを根，ゴールノードを任意の葉とする．評価関数 $f(n)$ には，スタートノードからノード n までの経路距離 $g(n)$ を設定する（これは均一コスト探索にあたる）．T のノードを展開するとき，そのノードが対応するステージの挿入対象ネットの状態から，次に 1 つ挿入するネットを挿入対象ネットとした状態に対応するノードを作成し，展開するノードの子とする．ネットの重さには，コスト関数を用いて計算した，新たに挿入対象となるネットに実際に逆相ラッチを挿入するのに必要なコストを設定する．

最良優先探索が終了した時に，T において根からの距離が最短となる葉の対応する挿入対象ネットが探索結果のネットとなる．

5.2 探索木のノードの数と探索時間による評価

今回，評価に用いた回路は bit 数が 4 ,5 ,6 のキャリールックアヘッド・アダー (CLA) を用いた．各回路の候補パスの数，候補ネットの数，クリティカル・パスの長さは以下の表 1 のとおりである．CLA の bit 数が増えるに従って回路の規模が大きくなり，候補パス数と候補ネット数とクリティカルパスの長さがいずれも増えている．これらの回路に対して，今回の手法を適用した時の評価を示す．

表 1　評価に用いた回路

bit 数	4	5	6
候補パス数	105	246	459
候補ネット数	109	199	243
CP の長さ	10	14	15

まず，で示した評価関数の定数値を N = 6 で固定した時における，回路と Visit した探索ノード数の関係を示したグラフが**図 6** である．CLA4bit，CLA5bit に比べて，CLA6bit の方がはるかに探索ノード数が増えている．候補パスの数で比較すると，6bit は 4bit の 4.3 倍であるが，探索ノード数は 22 倍となっている．さらに，グラフにはないが CLA8bit，CLA16bit に対しても適用しようと試みたのだが，半日ほど経過しても探索が終わらず，その時点での探索ノード数も相応に大きくなっていた．このことから，

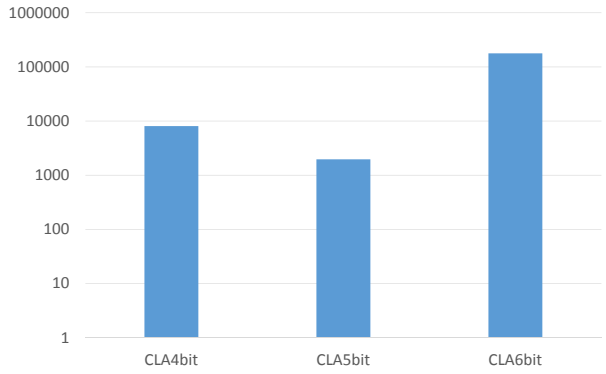

図 6　回路と探索ノード数の関係

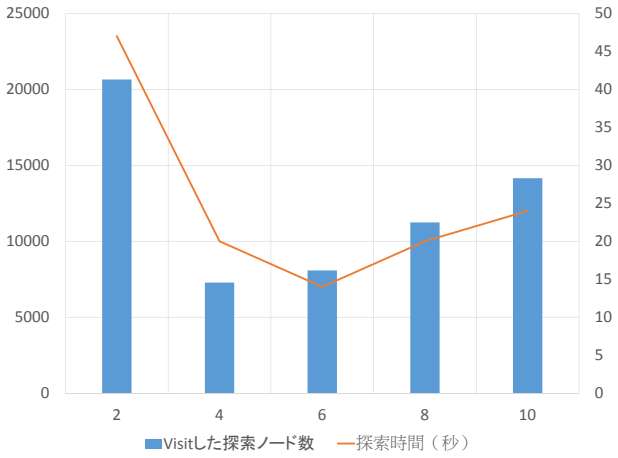

図 7　CLA 4bit における評価関数と探索ノード数と探索時間の関係

回路規模が大きくなっていくと探索ノード数がはるかに増えていき，探索が実用的な時間内に終わらなくなるということがいえる．これは，大きな回路に対して適用したいという適用ツールの目的に適わず，大きな問題となっている．

今回の手法を用いて評価関数の定数値を変えながら，各回路に対して適用した時の Visit したノード数と探索にかかった時間を示したグラフが**図 7 図 8 図 9** である．グラフの横軸が 5.2 で示した評価関数の定数値 N である．棒グラフが Visit した探索ノード数を示しており，左の軸が対応している．折れ線グラフが探索にかかった時間 (秒) を示しており，右の軸が対応している．CLA6bit の N = 2 が空白になっているのは，数時間経過しても探索が終了しなかったためである．これらから探索ノード数と探索時間の関係は概ね比例すると言えるだろう．

また，これらのグラフから，N の値を変化させてコスト関数を適切に設定すれば探索時間が短くなることがわかる．5.2 で示したコスト関数を用いているので，N の値を大き

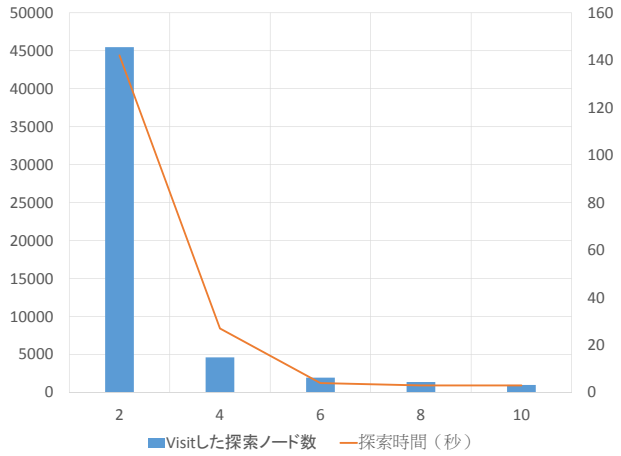

図 8　CLA 5bit における評価関数と探索ノード数と探索時間の関係

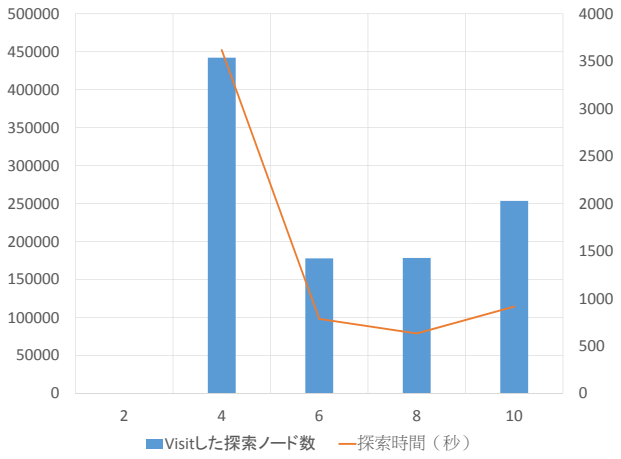

図 9　CLA 6bit における評価関数と探索ノード数と探索時間の関係

くすればパスの中央からの距離が遠い候補は探索されにくくなるので，探索ノード数も少なくなると考えられる．しかしながら，N の値を大きくすれば常に探索時間が短くなるというわけではないようである．CLA4bit と CLA6bit においては N の値を大きくした時に逆に探索ノード数が増えている．

これら N の値を変化させた時における各出力された回路について調べたところ，CLA5bit と CLA6bit では全ての N の値においてラッチの数とクリティカル・パスの長さが一致していた．CLA4bit では N = 2 の時のみ異なるが，他の値の時は一致していた．N = 2 の時は，よりラッチの数が少なくクリティカル・パスが長いものを出力していた．

5.3　探索における課題

探索ノード数が膨大になる問題

今回評価の対象にしたような，比較的小さな回路においても探索ノード数が大きくなり，それに従って探索に要する時間も大きくなってしまうという問題がある．適用ツールの目的が，プロセッサのような大規模な回路に対して適用したいというものなので，これは非常に問題である．

原因の一つとして考えられるのが，似たような組み合わせの挿入対象ネットの存在が大量にあることが挙げられる．2 つの挿入対象ネットの組み合わせの内，1 つの挿入対象ネットの位置が同じパス上で 1 つだけずれた箇所であっても，それぞれ別の組み合わせであり，探索がなされる．

また，探索木の深い方へ探索していくにつれて，パスの中央に近い探索候補ネットがなくなり，また探索木の浅いところから探索し直すことが多いということである．探索し直す時に，上で述べたように似たような組み合わせをまた探索することになるので，探索ノード数が増える．

これらを防ぐためには，似たような組み合わせを同じものとみなして，探索を行わないようにするなどの工夫が必要と考えられる．

逆相ラッチをまとめることで評価値が変化する問題

今の適用ツールでは，探索の結果として挿入対象となったネットに対して逆相ラッチを挿入する時に，同じノードからの出力を 1 つの逆相ラッチに入力し，逆相ラッチの出力を別々のノードに入力するようにしている．これは，不必要に逆相ラッチを挿入しないようにするためである．ステージのグラフ構造においてはノードの出力はネットの途中で分岐することはないが，実際の回路においては配線が分岐しているため，分岐前に逆相ラッチを挿入すれば良い．

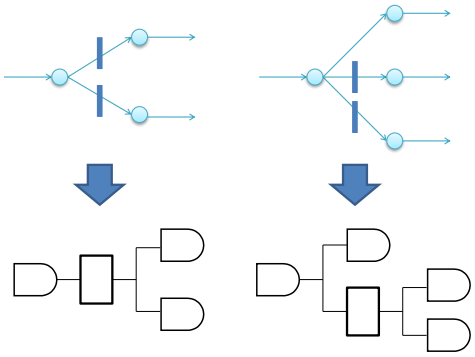

図 10　ソースノードが等しいネットに逆相ラッチを挿入する場合

図 10 において，上図はロジックのグラフ構造を示し，下図は逆相ラッチを挿入したときの実際の回路を簡単化したものを示している．上図のグラフ構造において，青色の縦棒に逆相ラッチを挿入することとする．このとき下図の実際の回路においては，同じノードからの出力が分岐する前に逆相ラッチを挿入している．左側の例のように同じノードの出力全てに逆相ラッチが挿入する場合は，逆相ラッチの後に分岐させれば良い．ただし，右側の例のように同じノードの一部の出力に逆相ラッチを挿入する場合は，逆相ラッチの前後で分岐を分ける必要がある．

以上の操作によって，逆相ラッチを挿入する数は減るが，

探索に用いていたコストの値が変化してしまう．コストは逆相ラッチを挿入する対象それぞれに対して計算しているので，複数の挿入対象を 1 つの逆相ラッチにまとめることにより，出力される回路のコストは探索に用いていたコストと異なってくる．この理由より，よりよいコストを持つ逆相ラッチの挿入位置の組み合わせを見逃している可能性が出てくる．これを解消するためには，探索の途中でまとめることにして，まとめた逆相ラッチの挿入位置に対してコストを計算するといった手法が考えられる．これに関しては今後の課題としたい．

6. おわりに

本稿では，動的タイム・ボローイングを可能にするクロッキング方式の適用手法において，二相ラッチ化の要件・目標を改めて定義するとともに，二相ラッチ化のアルゴリズムを提案した．

二相ラッチ化の要件は，全てのパスに逆相ラッチがそれぞれ 1 つだけ挿入されていることである．二相ラッチ化の目標は，挿入する逆相ラッチの数を少なくすることと，パスの遅延ができるだけ二分されるようにすることである．二相ラッチ化のアルゴリズムとして，全てのパスに逆相ラッチが挿入されるまでパスをイテレートして挿入対象ネットを探索することで二相ラッチ化の要件を満足させる．また，各逆相ラッチを挿入するのにコストが必要だとみなして，コストの総和が最小となるものを探索することで，望ましい目標を達成する．

二相ラッチ化の今後の課題としては，より早く目標に近い挿入位置の組み合わせを探索できるようなコスト関数を設計することと，より効率の良い探索アルゴリズムを採用することが挙げられる．

当初，探索アルゴリズムを決める時にはより効率のよい $A*$ アルゴリズムの採用を考えていた．$A*$ アルゴリズムでは，評価関数 $f(n)$ としてスタートノードから n までの距離 $g(n)$ と，ノード n からゴールノードまでの距離 $h(n)$ の推定値とを加えた $f^*(n) = g^*(n) + h^*(n)$ を用いる．ただし，任意の n に対して，$0 \leq h^*(n) \leq h(n)$ を満たす必要がある．しかし，探索木 T における $h(n)$ は，n の逆相ラッチの挿入対象ネットの状態から，残りの全ての逆相ラッチの挿入対象ネットに逆相ラッチを挿入するのに必要なコストの合計であるので，探索途中で残りどれだけのネットを挿入対象とすれば挿入対象ネットを全て挙げきれるかを予測することが難しく，条件を満たす $h^*(n)$ を推定することができていない．$A*$ アルゴリズムのような，効率のよい探索アルゴリズムの使用は今後の課題としたい．

今後は本稿で行った提案を，動的タイム・ボローイングを可能にするクロッキング方式の適用手法に組み入れ，このクロッキング方式の FPU やプロセッサなどを実装して評価していく予定である．

謝辞

本研究の一部は，科学研究費補助金基盤研究 (B)・26280012「レジリエンス指向コンピュータシステムに関する研究」の支援により行われた．

参考文献

[1] D.Ernst, N.Kim, S.Das, S.Pant, T.Pham, R.Rao, C.Ziesler, D.Blaauw, T.Austin, and T.Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Int'l Symp. on Microarchitecture (MICRO)*, pp. 7–18, 2003.

[2] Arindam Mallik, Jack Cosgrove, Robert P. Dick, Gokhan Memik, and Peter Dinda. PICSEL: Measuring user-perceived performance to control dynamic frequency scaling. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 70–79, 2008.

[3] 喜多貴信, 塩谷亮太, 五島正裕, 坂井修一. タイミング制約を緩和するクロッキング方式. 先進的計算基盤シンポジウム SACSIS, pp. 347–354, 2010.

[4] 吉田宗史, 広畑壮一郎, 倉田成己, 塩谷亮太, 五島正裕, 坂井修一. 動的タイム・ボローイングを可能にするクロッキング方式. 情報処理学会論文誌コンピューティングシステム（ACS）, Vol. 6, No. 1, pp. 1–16, jan 2013.

[5] 広畑壮一郎, 吉田宗史, 倉田成己, 五島正裕, 坂井修一. 動的タイム・ボローイング可能にするクロッキング方式の適用手法. 情報処理学会研究報告, No. 20 in 2012-ARC-201, pp. 1–8, 2012.

[6] 吉田宗史, 広畑壮一郎, 倉田成己, 五島正裕, 坂井修一. 動的タイム・ボローイングを可能にするクロッキング方式の適用手法の評価. 情報処理学会研究報告, No. 6, pp. 1–13, jul 2013.

[7] 五島正裕. ディジタル回路. 数理工学社, 2007.