4J-01

回路素子の静的解析を用いた二相化アルゴリズムの改良
Improving Dynamic Time Borrowing Clocking Scheme by Static Timing Analysis

酒井 一憲 † Kazunori Sakai
津坂 章仁 † Akihito Tsusaka
神保 潮 † Ushio Jinbo
五島 正裕 § Masahiro Goshima
坂井 修一 † Shuichi Sakai

1 はじめに

半導体プロセスの微細化が進むことで、素子遅延のランダムばらつきが大きな問題になってきている。ばらつきが増大していくと、最悪値に基づいた設計手法は悲観的になりすぎる。そのため、最悪値に基づくのではなく、実際の遅延に基づいた動作を実現する手法が提案されている。その一種として、**Razor** [1] がある。我々の研究室では、二相ラッチ方式と Razor を組み合わせることで、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、本方式とよぶ）を提案している [2]。本稿では、本方式を既存の回路に適用するためのアルゴリズムを提案する。

2 関連研究

2.1 Razor FF

Razor は動的なタイミング・フォールト検出手法である。タイミング・フォールト（Timing Fault：TF）とは、遅延の動的変化により設計者の意図とは異なる動作が引き起こされる過渡故障である。TF の発生を許容し、検出・回復するのが、動的 TF 検出である。

Fig. 1 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 にくらべて位相の遅れたクロックが供給されており、Main FF と Shadow Latch で 2 回、信号のサンプリングを行う。Main FF と Shadow Latch で信号の値が異なっていれば TF として検出する。

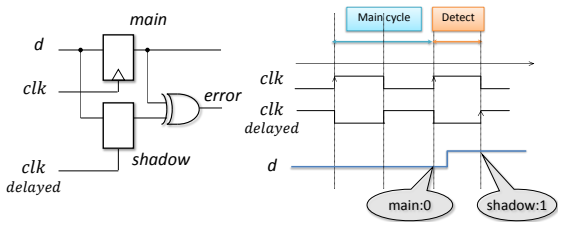

Fig. 1: Razor の回路構成

ショート・パス問題

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 が閉じるよりも早く次の入力が届いてしまうと、前のサイクルの信号と混ざることとなり、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。そのため、Razor FF には特有の最小遅延制約が存在する。ショート・パスに遅延素子を挿入する等の対策が必要となる。

2.2 動的タイム・ボローイング

2.2.1 タイミング・ダイアグラム

本研究では、タイミング制約を主題とするため、各ステージ間の信号の伝わる様子を詳しく見る必要がある。そこで、本稿ではタイム・チャートに変わるものとして、タイミング・ダイアグラム (以下 t-diagram と記述する) を使う。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いく方向を示し、時間の経過に連れて回路を信号が伝わる様子を俯瞰することができる。

2.2.2 動的タイム・ボローイング

マスタースレーブ構造をもつ FF は、正相と逆相で動く 2 つのラッチを組み合わせた構造をしている。二相ラッチは、FF を構成する 2 つのラッチの間にロジックを挿入したものとみなせる。単相 FF 方式の 1 ステージ分のロジックをラッチで 2 分割していることになる。本方式では二相ラッチ方式に加え TF 検出のために Razor を用いることで、本来利用可能であったラッチの開いている期間を、TF 検出を設けることによって利用する。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。これにより、ロジック上の全遅延の存在範囲をラッチの開いている区間にも広げることができるため、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。このことにより実効遅延の分散を吸収し、平均に近いサイクル・タイムでの動作を可能とする。

Fig. 2 の t-diagram において、ラッチの開いている区間に、信号の伝わる様子を表す直線が連続してつながっているのが、動的タイム・ボローイングの効果を表している。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 $1cycle/1stage$ となる。

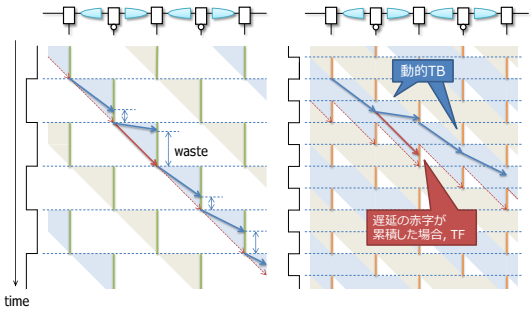

Fig. 2: 動的タイム・ボローイングの様子

2.2.3 適用手法

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 [3]。本節ではそのツールのアルゴリズムを説明する。

3 二相ラッチ化アルゴリズム

3.1 二相ラッチ化の要件

二相ラッチ化は、ステージ内のパスに逆相ラッチを挿入し、単相 FF が置かれていたところに正相ラッチを置くことで完了する。逆相ラッチを挿入する際、各パスに対して 1 つだけ挿入する必要がある。

3.2 二相ラッチ化の目標

動作速度向上

本方式では、サイクル・タイムは 2 分されたステージのクリティカル・パス遅延によって定まる。すなわち、ステージの遅延が大きい方にサイクル・タイムを合わせる必要が有るため、本方式の効果を最大限にするには、逆相ラッチの挿入する際にステージ全体の遅延をできるだけ均等に 2 分する場所に挿入する必要がある。

回路面積の増加の抑制

回路面積の増加を抑えるためには、挿入する逆相ラッチの数を少なくする必要がある。多くのパスが通る配線に対して

†東京大学, The University of Tokyo
§国立情報学研究所, National Institute of Informatics

挿入することによって逆相ラッチの数を少なくすることができる。

3.3 二相ラッチ化の課題

二相ラッチ化の要件を満たすように逆相ラッチの挿入位置を探索する際に問題となるのは、探索空間の大きさである。これは、各パスに対し、素子の段数だけ挿入位置があるためである。そのため、効率的に探索する必要がある。

3.4 アルゴリズムの問題点

先行研究の適用手法 (以下、既存手法と呼ぶ。) は、FPGA 上の回路に適用することを目的としているため、遅延の評価を素子の数で行っている。そのため、ASIC のような素子によって遅延が異なるような回路に対しては、うまく適用できない。次節では、この点について検討し、ASIC でも適用できる手法を提案する。

4 静的遅延解析を用いた遅延評価

本節では、静的遅延解析を用いた逆相ラッチの挿入場所の探索手法を提案する。本提案によって、より正確な逆相ラッチの挿入位置の決定を可能とし、ASIC 等の回路に対しても適用できるようになる。

4.1 静的遅延解析

ロジックは、基本的な素子が配線によってつながった構造を持っている。素子の遅延は主に、出力先のキャパシタンスと、入力の変化にかかる過渡時間によって変化し、また入力が 0 から 1 に上がる場合と 1 から 0 に下がる場合ではかかる時間も異なっている。また、入力の過渡時間についても、素子を通過することによって過渡時間が変化するため、素子の遅延は前後の素子に依存しているといえる。そのため、あるパスの遅延を計算する場合は、各素子の状況を考慮に入れる必要がある。このような素子の特性を考慮し、素子の特性情報を用いて遅延計算をおこなうことを、遅延解析と呼ぶ。

本提案では、パスごとの遅延解析を行うため、静的遅延解析を用いる。

4.2 提案手法

既存の適用手法では素子の遅延を一定として扱っていたのに対し、提案手法では、静的遅延解析を用いて素子毎の遅延を計算することで、ASIC 等の回路への適用を可能とする。既存の適用手法を利用するために、大きな探索アルゴリズムの変更は行わない。そのため、既存の適用手法における素子の遅延の値を変化させることを考える。静的遅延解析の結果を用いて、既存の適用ツールにおける各素子の遅延の値を更新させた後、既存の適用手法を用いることで、ASIC 等の回路への適用が可能となる。

そのため、要件としては静的遅延解析を行うことで、素子毎の遅延の値を更新することとなる。今回用いる静的遅延解析ツールは、Synopsys 社の PrimeTime [4] である。

4.2.1 アルゴリズムの概要

PrimeTime を利用した手順は以下のようになる。

1. 適用ツールに回路を読み込ませたのち、PrimeTime で遅延を測定するための命令を生成する。
2. 生成した命令を PrimeTime で実行し、遅延の出力をする。
3. 遅延の出力結果を解析し、素子毎の遅延の値を取得する。
4. 取得した遅延の値を適用ツールに代入し、二相ラッチ化と Razor 化を行う。

Fig. 3 は、適用ツールと PrimeTime のデータのやり取りの様子を表したものである。上で述べた概要の内、1 と 4 は Windows 上で、2 と 3 は Linux 上で行うことになる。

4.2.2 アルゴリズムの詳細

命令の生成

適用ツールは、ハードウェア記述言語で書かれた回路を読み込み、回路をグラフ化している。また、PrimeTime では、ある FF を起点としたパスの遅延を調べることができる。これらを組み合わせると、グラフ化した回路情報から FF を参照し、その FF を起点としたパスの遅延を調べることができる。つまり、すべての FF を指定した命令を生成すれば、回路中のすべてのパスの遅延を調べる事ができる。その際に、素子の遅延のばらつきを考慮にいれ、素子が最小遅延の場合と、最大遅延の場合をそれぞれ出力する命令を生成する必要がある。PrimeTime では Tcl を実行することができるため、Tcl 形式の命令を生成する。

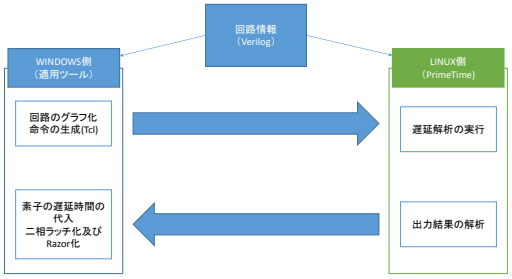

Fig. 3: 適用ツールと PrimeTime の連携

命令の実行と結果の解析

生成された命令を、PrimeTime で実行する。出力結果には、各素子でどれだけ遅延があるのかと、そのパスの遅延の合計が記述されている。出力結果から各素子の遅延を抽出するが、その際素子それぞれの遅延の最大値と最小値を取得する。その後、既存の適用ツールの素子の遅延の最大値、最小値を代入する。既存の適用ツールでは、素子遅延の情報は 1 つだったが、提案手法では 2 つ保持するように変更する必要がある。

二相ラッチ化と Razor 化

代入された素子の遅延情報を用いて、二相ラッチ化と Razor 化を行う。素子の遅延情報は、二相ラッチ化の際の逆相ラッチの挿入と、Razor 化の際のショート・パス問題の解決の 2 つで利用する。しかし、Razor 化において最大値を用いてしまうと、ショート・パス問題が発生する可能性がある。そのため、逆相ラッチの挿入の際は最大値を利用し、クリティカル・パスの検出をできるようにする。一方、ショート・パス問題は最小値を利用することで、ショート・パス問題が起こらないようにする。

5 まとめと今後の課題

本稿では、静的遅延解析を用いる動的タイム・ボローイングを可能と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の改良を検討した。提案手法では、各素子の遅延の最大値と最小値のみを利用したが、素子の遅延が入力によって変化するため、正確な適用ができていない問題点が存在する。また PrimeTime の出力としてパスごとの合計時間があるので、これを利用したアルゴリズムに変更することでより正確な結果を導出することができる可能性があり、その点について今後考察を行う。また現在の手法では，同じ素子に対して何度も解析を行うように命令を生成しているため，これを最適化することも考えられる．

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] 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), 第 6 巻, pp. 1–16, jan. 2013.

[3] 津坂章仁, 谷川祐一, 広畑壮一郎, 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ための二相ラッチ生成アルゴリズム. 情報処理学会研究報告, No. 9 in 2014-ARC-211, pp. 1–10, jul. 2014.

[4] Synopsys. Primetime. `http://www.synopsys.com/JP2/Tools/Implementation/SignOff/Pages/PrimeTime.aspx`.