# 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予備評価

広畑 壮一郎† 神原 太郎† 吉田 宗史†
倉田 成己† 五島 正裕† 坂井 修一†

## 1. はじめに

近年では, 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増大している. そのため, 遅延の平均値とワースト値の差が広がっていき, 従来のワースト値に基づく設計手法は悲観的になりすぎている. この対策として, ワースト・ケースよりも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. このうち, 動作時にタイミング・フォールトを検出し, 回復する手法がある. タイミング・フォールト（以下 TF）とは, 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き起こされる過渡故障である. Razor[1] は, TF を動的に検出することができる. Razor を用いた回路に, DVFS(Dynamic Voltage and Frequency Scaling) を組み合わせると, 実際の遅延に応じた動作を実現することができる.

## 2. 提案手法

提案手法は, 端的に言えば, TF 検出と二相ラッチ[3] を組み合わせたクロッキング方式である（**図 1**）. 提案手法では Razor のショート・パス問題を解決するために 2 相ラッチのクリティカル・パスとショート・パスが合流するゲートを 2 重化して, ショート・パスに遅延を挿入するため, Main ラッチのパスの遅延が増加しない. これにより, ステージ間で実効遅延を融通する動的タイム・ボローイングが可能となり, 単相フリップフロップの最大で 2 倍の動作周波数が達成できる[2]（**図 2**）.

## 3. 変換ツール

我々は EDIF で記述された回路を提案手法を適用した回路に自動的に変換するツールを作成した. このプログラムは主に 3 つのステップから構成される. 最初にステージ分割. 次に 2 相ラッチ化. 最後に Razor の挿入である.

### 3.1 ステージ分割

回路の 2 相ラッチ化を行うにあたって, まず回路を

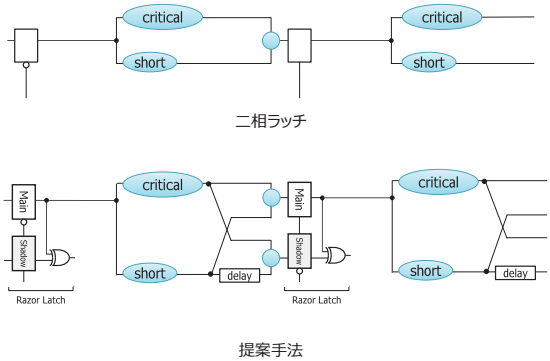


**図 1** 提案手法の回路

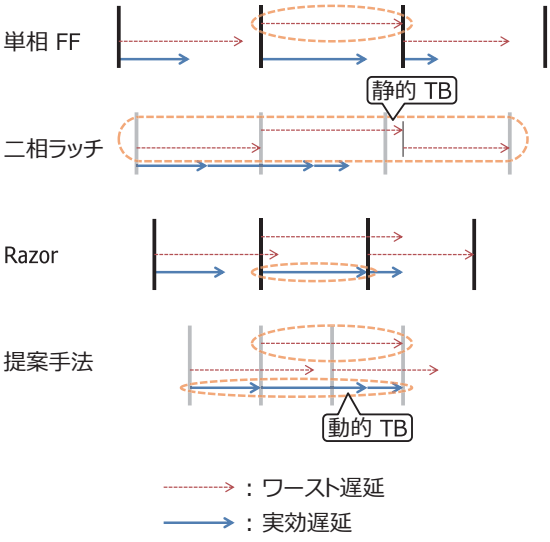


**図 2** 従来のクロッキング方式との比較

ステージという単位に分割する必要がある.

#### 3.1.1 ステージとは

我々がステージと呼んでいるものは, 回路の中でのフリップフロップからフリップフロップまでの間のロジックである. 回路はステージがいくつも繋がって構成されたものであると考えることができる.

#### 3.1.2 分割アルゴリズム

まず任意のゲートを 1 つ選ぶ. このゲートと同じ

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

ステージにあるゲート全てを探索するためには単純にゲートにつながっている入力, 出力両方のエッジをたどっていき, たどり着いたゲートすべてを同じステージとしてマークする. もしたどり着いたゲートが FF であったときはそれ以上先の探索を打ち切る. ここで検出されなかったゲートは別のステージに属するので, 同様に探索を行い, すべてのゲートの属するステージが決まった所で探索終了となる.

### 3.2 二相ラッチ化

ステージを二相ラッチ化するためには, ステージのロジックの間にラッチを挟むことで, ロジックを二分する必要がある. まずステージのクリティカル・パスの遅延を求める. 遅延がクリティカル・パスの遅延の二分の一程度になる場所を探索し, そこにラッチを挟むことで, ロジックを二相ラッチ化する.

### 3.3 Razor の挿入

TF を検出するために, TF を引き起こす可能性のあるパスに Razor を挿入する必要がある. 提案手法では TF はクリティカル・パスの活性化によって遅延が蓄積することで発生する. したがって TF が発生する可能性のあるパスとは, 遅延が蓄積する可能性のあるパスのことである. 遅延の蓄積はサイクルタイムの半分より遅延の大きいパスが活性化することで起こる. 動作させる目標の動作周期の半分より長い遅延を持つパスを検出し, そこに Razor を挿入する.

#### 3.3.1 ２重化と遅延の挿入

Razor の検出ウィンドウは, Razor の TF 検出が Shadow Latch によって行われるために, 動作クロックによって検出ウィンドウの長さが決まる. Razor の検出ウィンドウは Main Latch が閉じている期間である. ショート・パスが活性化することでこの検出ウィンドウに入るとショート・パス問題が起こる. したがってショート・パス問題が発生する遅延は, サイクルタイムの半分となる. Razor のショート・パス問題を回避するために, ショート・パスに遅延を挿入する必要がある.

しかし, 単純にショート・パスに遅延を挿入すると, ステージ間の動的タイムボローイングに於いて遅延の蓄積が大きくなり, TF が発生しやすくなってしまう. これを避けるために, 遅延素子からラッチまでの間にあるゲートを複製し, Main Latch につながるゲートと Shadow Latch につながるゲートを分ける. 我々はこれをゲートの２重化と呼んでいる. そして Shadow Latch 側のゲートにだけ遅延を挿入することで, Main Latch 側に遅延が入らなくなる.

しかしこのようにゲートを２重化すると, 素子数が増えてしまう. 回路面積のオーバーヘッドを抑えるために, できるだけ２重化するゲートを少なくしたい. Main Latch に遅延を入れないためには, ステージ内の遅延より出力側のすべてのゲートを２重化する必要がある. したがって遅延素子をステージの出力側にできるだけ近いところに挿入することで回路面積の増加を軽減する事ができる.

#### 3.3.2 遅延挿入アルゴリズム

このような動作を実現するアルゴリズムを以下に示す. ショート・パス問題を起こさないための最小遅延を $D_{th}$ とする. あらかじめステージのゲートの入力からの最大遅延と最小遅延を求めておく. ステージの出力側のラッチから入力を辿って探索を始める. ゲートの最小遅延 + このゲートから探索開始ラッチまでの遅延 が $D_{th}$ より小さいならばこのゲートはショート・パスを含んでいる. このゲートの出力側に入れることのできる遅延は 探索開始ラッチの最大遅延 - (ゲートの最大遅延 + 探索開始ラッチまでの遅延) である. ショート・パス問題を解消するために必要な遅延がこの上限より小さい時はこのゲートとこのゲートの出力につながっている２重化されたゲートの間に遅延を挿入する. ショート・パス問題を解消するために必要な遅延がこの上限を超えているときは遅延は更に前のゲートに入れる. このゲートを２重化し, このゲートの入力側のゲートに再帰的にこの処理を繰り返す. この再帰を Razor のラッチすべてに対して行うことで回路の２重化と遅延の挿入が完了する.

## 4. おわりに

製造ばらつき対策手法は数多く提案されているが, 提案手法では遅延が大きくばらつく入力ばらつきを利用することで, より実効遅延に近い速度で動作し, 高クロック化や低電圧化を達成できる. 単相フリップフロップ方式に対して, 最大で 2 倍の動作周波数が可能となる. さらに, 変換ツールにより一般的な回路に対して容易に提案手法を適用することができる.

**謝辞** 本論文の研究は, 一部 JST CREST「ディペンダブル VLSI システムの基盤技術」「アーキテクチャと形式的検証による超ディペンダブル VLSI」による.

## 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”, 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, Vol. 6, No. 1, pp. 1-16 (2013).
3) D. Harris, “Skew-tolerant circuit design”, Morgan Kauf-mann Publishers, pp.12-14, (2001).