Vol.2018-ARC-232 No.23
2018/8/1

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プロセッサへの適用

神保 潮[1,a)]　塩谷 亮太[2,b)]　五島 正裕[3,c)]

概要：ワースト・ケースより平均的ケースにおける遅延に基づいた動作を実現する手法の 1 つとして，我々は動的タイム・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いる．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・フォールト検出と二相ラッチによるクロッキング方式の組み合わせにより実現され，動作時にステージ間で実効的な回路遅延を融通することで，平均的ケースに基づく速度で回路を動作させることが可能になる．本発表では，FPGA をターゲットとし，RISC-V 実装の 1 つである Rocket にこの方式を適用する．

キーワード：ばらつき，タイミング故障検出，タイム・ボローイング，Razor FF，Rocket

1. はじめに

チップ内のランダムな**ばらつき**の増大 [1] により，従来のワースト･ケースに基づいた設計ではチップの性能の向上が見込めなくなりつつある．微細化により，遅延の典型値は短縮されている一方で，ばらつきの増大によって分散は大きくなっている．そのため，歩留まりを一定とすると，ワースト遅延は，ティピカル遅延ほどには短縮されなくなる．こうした傾向が続けば，微細化が進むにつれてティピカル遅延とワースト遅延の差は広がっていき，将来的には，ワースト遅延が短縮され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 (実効遅延) に基づいた動作を実現する手法が提案されている [2]．SSTA のように，設計時に用いられる静的な手法に対し，動作時にタイミング･フォールトを検出し回復する動的な手法 [3–6] がある．

タイミング･フォールト検出

回路遅延の動的な変動によって生じる過渡故障を **タイミング･フォールト** (Timing Fault: **TF**) と呼ぶ．ワースト･ケース設計では，ワースト･ケースにおいてもこの TF が発生しないように，十分なマージンを取った電圧やクロック周波数を設定する．TF が生じるのは，サーマル・センサの故障による熱暴走など，想定外の場合に限られる．

TF を検出・回復する手法を用いることで，ワースト･ケース設計で定められる限界を超えて回路を高い周波数，または低電圧で動作させることができる．TF 発生による IPC の低下が十分小さく，周波数の向上（低電圧化）による恩恵が相殺されない範囲で，回路の実効遅延に応じた周波数や電源電圧で動作させることが可能になる．

本稿の内容

我々は，より効果的な周波数向上や低電圧動作を可能にする手法として，動的タイム･ボローイング（Dynamic Time Borrowing: **DTB**）を可能にするクロッキング方式を提案している [7–9]．現在，提案手法の回路への実装による評価を行っている．今までは，カウンタのような小規模な回路への適用のみが行われていた [8,9]．本稿では，RISC-V ISA [10] に準拠する 64-bit スカラ･プロセッサ Rocket [11] を対象としてこの手法を適用する手順について述べる．

以下，本稿は次のように構成される．2 章では，既存のクロッキング方式についてまとめる．3 章で提案方式について述べる．4 章で TF からの回復手法について述べる．5 章では提案方式の適用手順について述べる．

2. クロッキング方式

本章では，次章で述べる提案方式をよりよく理解するために，まず既存のクロッキング方式を説明する．2.1 節で

[1] 総合研究大学院大学 複合科学研究科
School of Multidisciplinary Sciences, SOKENDAI
[2] 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
[3] 国立情報学研究所 アーキテクチャ科学研究系
Systems Architecture Research Division, NII
[a)] ushio@nii.ac.jp
[b)] shioya@nuee.nagoya-u.ac.jp
[c)] goshima@nii.ac.jp

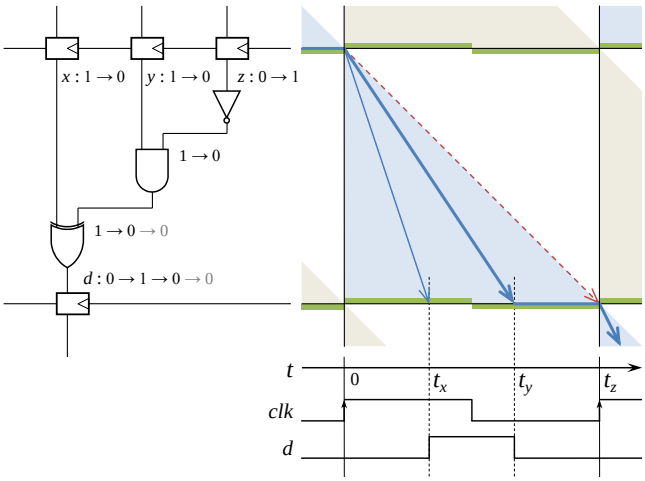

図 1 単相 FF のタイミング･ダイアグラム

は，クロッキング方式の理解に便利な**タイミング･ダイアグラム**を導入する．2.2 節以降で，単相 FF，二相ラッチ，そして，Razor [3] について説明する．

2.1 タイミング･ダイアグラム

図 1 に示すグラフを，我々はタイミング･ダイアグラムと呼んでいる．通常のタイミング･チャートが論理値―時間の関係を表すのに対して，タイミング･ダイアグラムは時間―空間の関係を表す．同図中，右方向が時間を，下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信号が伝わっていく様子を俯瞰することができる．

実際のロジックには，それぞれ遅延が異なるパスが数多く存在する．ダイアグラムでは，入力の変化によって出力が変化した時，その信号伝達を，入力が変化した点から出力が変化した点までを（右下がりの）直線矢印で結んで表す．

実効遅延

ロジック中の信号の伝達の仕方は，ロジックの入力の変化の仕方によって異なる．一部の信号の遷移はマスクされるため，一般にすべてのパスが出力の変化に関与する訳ではない．ロジック中のあるパスを通った信号によってロジックの出力が変化したとき，そのパスは活性化されたと言う．

ダイアグラムでは，あるサイクルにおいて最後の出力の変化をもたらした信号の伝達を実線矢印で表す．この実線矢印の遅延（図上で縦方向の距離）を，そのサイクルの**実効遅延**と呼ぶ．

ダイアグラム上で実線矢印が存在可能な範囲は，ロジック内の最小遅延とクリティカル･パス遅延を表す直線に挟まれた三角形の領域となる．ダイアグラムではこの領域を網掛けにより示す．図中の網掛けの二色については後述する．

なおダイアグラムでは，各ステージのクリティカル･パスに対応する直線矢印の角度を 45° としている．こうすることによって，各ステージの遅延は，ダイアグラム上のステージの幅によって表現することができる．

入力ばらつき

実効遅延という言葉を用いるなら，**入力ばらつき**は，ロジックの入力の変化の仕方に応じて生じる実効遅延のばらつきと定義することができる．

ロジックの出力が一度も変化しなかった時，実効遅延は 0 と考えられる．すなわち入力ばらつきによって，ロジックの実効遅延は 0 からクリティカル･パス遅延まで変化することになる．他の要因によってはロジックの（クリティカル･パス）遅延は数割程度しかばらつかないことを考えると，入力ばらつきは非常に大きいと言える．

2.2 クロッキング方式の表現

次に，図 1 でのクロッキング方式の表現を説明する．

エッジ･トリガ動作

同図はマスタ–スレーブ構造を持つ FF を念頭に描かれている．同図において，FF の下にある縦実線はラッチが閉じている状態を，縦実線と次の縦実線の間の空白は，ラッチが開いている (transparent) 状態を，それぞれ表している．信号の矢印が実線にぶつかった場合，ラッチが開くまで信号は下流側に伝わらない．エッジ･トリガ動作は，マスタ–スレーブ･ラッチを互い違いに記述することで生じる隙間から信号が「漏れる」様子で直感的に表すことができる．

フェーズ

パイプライン動作を行う際には，FF と次の FF に挟まれたロジックがパイプライン･ステージとなり，各クロック･サイクルごとに各ステージが並列に動作を行うことになる．

パイプライン動作においては，一連の処理 —— 典型的には，パイプライン型プロセッサにおける 1 つの命令の処理 —— は，あるサイクルにおいてあるステージで処理された後，次のサイクルにおいて次のステージの処理へと次々引き継がれていく．この一連の処理のことをある**フェーズ**の処理と呼ぶ．

ダイアグラムでは，あるフェーズの処理と次のフェーズの処理を，矢印が存在し得る領域の網掛けの色を分けることで区別している．

2.3 クロッキング方式の要諦

クロッキング方式の要諦は，あるフェーズの信号が前後のフェーズの信号と「混ざる」ことがないように分離した上で，処理を次のサイクルに次のステージへと引き継いでいくことである．

ダイアグラム上では，以下の 2 つの条件が満たされていればよい：

(1) 実線矢印をたどって，次のサイクルに次のステージへと至ることができる．
(2) 矢印が存在し得る範囲を表す網掛けの領域が，前後のフェーズの, すなわち，色の異なる網掛けの領域と重

ならない．

クロッキング方式のタイミング制約は，この 2 条件から導かれる．

次章からは，ダイアグラムを用いてそれぞれのクロッキング方式について説明する．

2.4 単相 FF 方式

単相 FF 方式が上記の条件を満たして正しく動作するためには，各ステージにおいて，あるクロック・エッジで入力側の FF の出力が変化してから，次のクロック・エッジまでに出力側の FF の入力に信号が到着しなければならない．すなわち，サイクル・タイムを τ とすると，各ステージのロジックのクリティカル・パスの遅延が τ 未満であればよいということになる．このことを，最大遅延制約は $1\tau/1$ ステージと表現することとする．

図 1（および，**図 2** (a)）では，クリティカル・パスの遅延を表す赤い 45° の線がちょうど次のクロック・エッジに到着しており，最大遅延制約の限界を達成した場合を表している．なお，簡単のため，FF やラッチのセットアップ/ホールド時間やスキューなどは省略しているが，これらを議論に組み込むことは容易である．

通常，クリティカル・パスが活性化される確率は高くない．図 1 のように，実効遅延とクリティカル・パス遅延の差の分だけ，無駄な待ち時間が生じることになる．

2.5 二相ラッチ方式

図 2 (b) に，二相ラッチ方式のダイアグラムを示す．二相ラッチ方式は，単相 FF 方式における FF を構成するマスタ，スレーブの 2 つのラッチのうちの 1 つをロジックの中間へと移したものと理解することができる．移されたラッチによって分割された後のステージを特に**半ステージ**と呼ぶ．

単にラッチの位置を動かしただけなので，二相ラッチ方式の最大遅延制約は，基本的には，$0.5\tau/1$ 半ステージ となり，単相 FF の $1\tau/1$ ステージ と変わらない．

2.6 Razor

本節では，TF 検出技術の代表として Razor FF [3] について述べる．

回路構成と動作

図 3 左に，Razor FF の回路構成を示す．1 つの Razor FF は，**メイン FF** と**シャドウ・ラッチ**によって構成される．シャドウ・ラッチには，メイン FF へのクロック clk より Δ だけ位相の遅れたクロック clk_d が供給されている．その結果，メイン FF とシャドウ・ラッチで 2 回，入力 d のサンプリングを行うことになる．それらの値が異なっていれば，TF が検出され，エラー e がアサートされる．

同図右は，d の遷移がメイン FF のクロック・エッジよりも遅れてしまった場合のタイミング・チャートである．メ

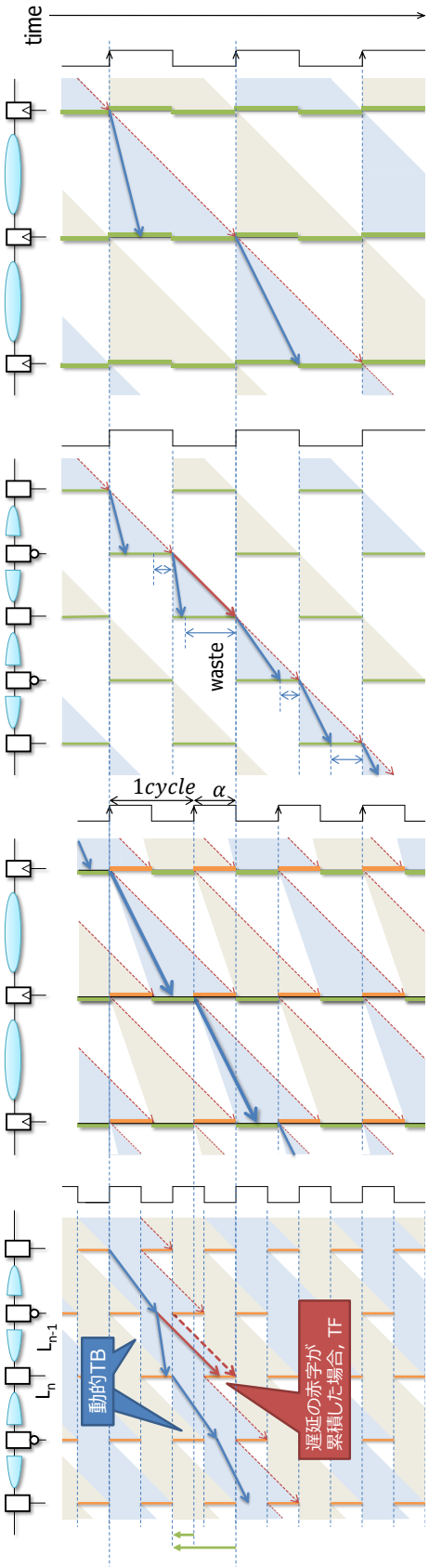

図 2 各クロッキング方式のタイミング・ダイアグラム：
(a) 単相 FF，(b) 二相ラッチ，(c) Razor FF，(d) 提案方式

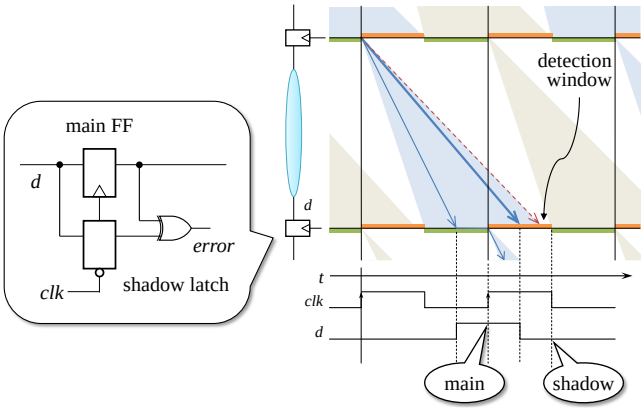

図 3　Razor FF の回路と動作

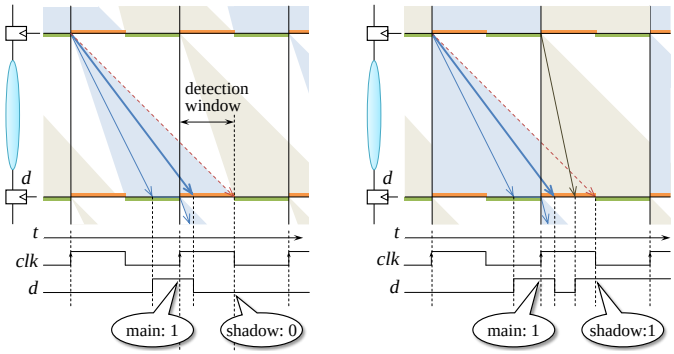

図 4　Razor のショート・パス問題

イン FF は t_1 で 1 をサンプリングするが，シャドウ・ラッチは $t_1 + \Delta$ で 0 をサンプリングする．両者は異なっているため，e は 1 となる．t_1 から $t_1 + \Delta$ の期間を，本稿では TF 検出ウィンドウと呼び，図中では網掛けで示す．

なお，メイン FF がメタステーブルとなった場合，ここで説明したダブル・サンプリングによる方法では対応できない．一方，遷移検出を用いる方式の Razor FF では，メタステーブルを TF として検出することができる [4, 12]．ただし後者は，ダイナミック・ロジックを利用するため，FPGA 上に実現することはできない．そこで以下では前者を前提として説明を行うが，同様の議論は後者についても成り立つ．

タイミング制約

図 2 (c) に，Razor FF のダイアグラムを示す．同図では，$\Delta = 0.5\tau$，すなわち，半周期遅れたクロックをシャドウ・ラッチに供給している．ダイアグラムでは，FF の下の濃さの異なる縦実線（橙色）が，TF 検出ウィンドウを表している．

クリティカル・パスの遅延に対応する 45° の破線が検出ウィンドウの下端までに到着するなら，TF が発生したとしても検出し，回復することができる．そのため，45° の破線矢印はジグザグとなる．一方，TF 検出を行わない単相 FF や二相ラッチでは，45° の破線は一直線となる（同図 (a)，(b)）．

TF 検出を行う方式では，このジグザグの分だけ，クリティカル・パス遅延を超えてサイクル・タイムを短縮することができる．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（図では $\alpha = 0.5$），最大遅延制約は $(1 + \alpha)\tau/1$ ステージ となり，単相 FF 方式より $\alpha\tau$ だけ改善される．

Razor のショート・パス問題

クロック・スキューに起因するホールド・タイム違反など，ショート・パス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．Razor には，Razor 特有のショート・パス問題がある．

図 4 のダイアグラムにおいて，シャドウ・ラッチ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シャドウ・ラッチのサンプリング・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，あるフェーズにおいてショート・パスを通った信号が，前のフェーズの信号と「混ざる」．その結果，シャドウ・ラッチが本来とは異なる値をサンプリングし，TF を検出できない可能性がある．

このため Razor は，最小遅延制約をもつ．図 4 では，シャドウ・ラッチのサンプリングを 0.5τ 遅らせているため，最小遅延制約は $0.5\tau/1$ ステージ となる．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 $\alpha\tau/1$ ステージ となり，単相 FF 方式より $\alpha\tau$ だけ厳しくなる．したがって，同図のようにショート・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 $\alpha\tau$ 以上にする必要がある．

3. DTB を可能にするクロッキング方式

我々は入力ばらつきにおける平均遅延に基づいた動作を可能にする手法として，DTB を可能にするクロッキング方式を提案している [7–9].

3.1 回路構成と動作

図 5 に，提案方式の回路構成を模式的に示す．提案方式は，基本的には，二相ラッチと TF 検出との組み合わせである．すなわち，同図上に示すような二相ラッチの回路のラッチ部分を，Razor の TF 検出回路に置き換えたものと考えてよい．なお，2.6 節で述べたように，本稿では TF 検出にダブル・サンプリングを用いた場合の説明を行うが，実用的な設計では遷移検出を想定する．

2.6 節で述べた Razor 特有のショート・パス問題を回避するため，ショート・パスに遅延を挿入する必要があるが，以下の工夫を行う：同図上の二相ラッチの回路では，ロジックのショート・パスとクリティカル・パスとが，図中〇印で示すゲートで合流した後，ラッチに接続されている．この場合，合流するゲート〇を二重化し，それぞれをメインとシャドウに接続する．その上で，シャドウに至るショート・パスにのみ遅延を挿入する．これにより，以下の 2 つを両立す

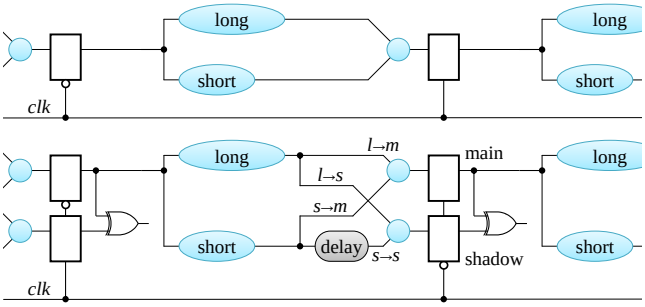

図 5　二相ラッチ（上）と提案方式（下）の回路の模式図

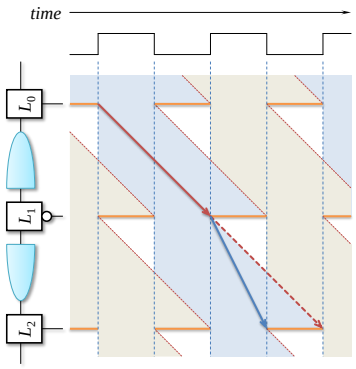

図 6　動的タイム・ボローイング (DTB)

ることができる：

- Razor 特有のショート・パス問題は，ショート・パスによりシャドウが正しい値をサンプリングできない問題であるから，シャドウに至るショート・パスに遅延を挿入すれば解消される．逆に，
- メインに至るパスに遅延を挿入しないことによって，ショート・パスが活性化した場合の実効遅延が伸びることが避けられる．3.2 節で詳述するように，これにより DTB の効果が最大化される．

実際の回路は，同図のようにショート・パスとクリティカル・パスがきれいに二分されている訳ではない．実際の遅延の挿入方法は [13] に詳しい．

3.2　動的タイム・ボローイング

2.5 節で述べた二相ラッチ方式においてはラッチの開いている期間を利用することができない．開いている期間を利用すべく，クリティカル・パス遅延よりサイクル・タイムを短くすると，クリティカル・パスが連続で活性化した場合に TF が発生するためである．提案方式では，TF 検出・回復を組み合わせることにより，ラッチの開いている期間を積極的に利用することが可能となる．

そしてこの結果，動作時に各ステージ間での**実効遅延の融通**が可能になる．**図 6** に，提案方式のダイアグラムを示す．同図では，最初の半ステージでクリティカル・パスが活性化しているが，直後の半ステージで実効遅延が 0.5τ のパスが活性化したため，ぎりぎり TF を起こすことなく動作した場合を表している．逆に，直後の半ステージで再びクリティカル・パスが活性化した場合には，TF として検出されることになる．

遅延の「借金」

このように提案方式では，ラッチの開いている期間を利用することによって，遅延の累積を解消することができる．ダイアグラム上における，直線矢印がつながってステージ間を伝播する様子は DTB の効果を表している．

このように，遅延の累積を解消するためには実効遅延が短いことが望ましい．3.1 節で述べたように，ショート・パス問題のための遅延の挿入はメインに至るパスには行わないが，それは実効遅延をできる限り短縮するためである．

遅延の「貯金」

同図では，網掛けの領域が上下にオーバーラップしているが，これは図 5 に示す二重化されたパスの上で起こっている．すなわち，前のフェーズのシャドウに至るクリティカル・パスと，次のフェーズのメインに至るショート・パスにおける信号の伝達が同時に起こり得るため，ダイアグラム上でオーバーラップして見えるのである．したがって，別のフェーズが「混ざる」ことはない．

ショート・パスが連続で活性化した場合には，（同図ではオーバーラップの裏で）信号はラッチの閉じている期間に到着する．そこで，ラッチが開くまで待たされることになる．

したがって提案方式では，遅延の「借金」を持ち越して解消することができるが，遅延の「貯金」を持ち越すことは残念ながらできない．

タイミング制約

提案方式の最大遅延制約は，Razor と同様，TF 検出の検出限界によって決まる．図 6 のように，クリティカル・パスの遅延に対応する 45° の破線が検出ウィンドウの下端までに到着するなら，TF を検出することができる．

ただし提案方式では，前述したオーバーラップによって，サイクル・タイムを更に短縮することが可能となる．提案方式の最大遅延制約は $1\tau/0.5$ ステージとなり，単相 FF 方式や二相ラッチ方式に比べ，最大 2 倍の動作周波数の向上を見込むことができる．

大数の法則 と 入力ばらつき

開いている期間においては，ラッチはバッファとして機能する．すなわち，開いている期間を信号が通過する限りにおいては，各半ステージのロジックは，長大な 1 つの組み合わせ回路として動作することになる．このため，大数の法則により，入力ばらつきの平均値に基づく動作が可能となるのである．

3.3　クロッキング方式ごとの最小サイクル・タイムの比較

本章の最後に，各クロッキング方式における 1 ステージのクリティカル・パス遅延 c と，シャドウ FF/シャドウ・ラッチへのショート・パス遅延 s に対する最小・最大サイクル・

タイムについてまとめる．各クロッキング方式の最小/最大遅延制約を満たすように最小/最大サイクル･タイム τ は，**表 1** のようにまとめられる．Razor は，提案方式と同じく，$\alpha = 0.5$ とした．

TF 検出を行う方式では，最大のサイクル･タイムがシャドウに至るショート･パスの遅延に応じて決まる．提案においては $1/2 \times c$ から s までのサイクル･タイムを取り得るため，c を所与とすると，s は $1/2 \times d$ 以上である必要がある．

4. TF からの回復機構

TF 検出・回復手法を適用するうえで，プロセッサは，他の一般のハードウェアより対応が容易である．それは，プロセッサにはアーキテクチャ・ステートが定義されているからである．

4.1 アーキテクチャ・ステート

アーキテクチャ・ステート（Architecture State : AS）（以下では AS とする）は通常，命令セット・アーキテクチャにおいて定義され，PC（を含む PSW）と（論理）レジスタ・ファイル，および，その他の制御レジスタからなる．AS は，主に OS とのインタフェースの一部をなし，たとえば，コンテクスト・スイッチ時にセーブ/レストアされる対象となる．

一方，マイクロアーキテクチャにおいては，PC や（論理）レジスタ・ファイルに加えて，主記憶も AS に含めると都合がよい．そのようにすると，命令の**コミット**を，命令（の実行結果）による AS の不可逆的な更新と定義できるからである．以下，本稿では，この拡張された定義を採用する．

4.2 アーキテクチャ・レベル回復手法の概略

我々はスーパスカラ･プロセッサに適したアーキテクチャ・レベル回復手法を提案している [14,15]．プロセッサにおける回復は，AS を利用して，次のようにすればよい：

(1) コミットの停止　：コミットを停止することによって，AS を TF から保護する．

(2) TF の影響の除去　：パイプラインから TF の影響を取り除く．その後，保護された AS から実行を再開する．

この方式ではエラー回復のオーバーヘッドは例外からの回復と同程度であり，数〜数十サイクルとなる．TF の発生が数千サイクルに一回程度になるように電圧・サイクル･タイムを制御すれば，オーバーヘッドは 1% 程度に抑えることができる．

例外ではパイプライン・フラッシュを用いて間違った命令を除去するが，それとは異なり，TF の影響の除去は**初期化**によって行う．これは制御系のレジスタが TF の影響を受けた場合，パイプライン・フラッシュではその影響を取り除くことができないためである．制御系のレジスタの初期化は，ポインタ・カウンタの値であれば 0 にすることであり，Valid ビットであればそれを落とすことである．

表 1　クロッキング方式の最小/最大サイクル･タイム

方式	最小	最大
単相 FF	c	N/A
二相ラッチ	c	N/A
Razor	$2/3 \times c$	$2 \times s$
提案方式	$1/2 \times c$	s

5. DTB を可能にするクロッキング方式の適用

本章では，プロセッサへの提案方式の適用に関して詳述する．

5.1 プロセッサへの適用手順

本稿はイン・オーダの 5-ステージ・スカラ・プロセッサである Rocket [11] を適用の対象とする．Nexys4 DDR FPGA ボードをターゲットとした開発環境として lowRISC [16] を用いる．

一般的なプロセッサを対象とした提案方式の適用は以下の手順で行う．

(1) FPGA 上に構成する TF の発生を想定する回路部分 (A) を明確にし，AS を明確化し，スタビライズ・ステージを付加する．
(2) A に対して論理合成を行い，ネットリストを得る．
(3) 得たネットリストに対して，二相ラッチ化を行う．
(4) TF 検出が必要なパスの終端を Razor に置き換える．
(5) エラーを伝搬するネットワークを付加し，AS の更新を制御する機構を設ける．
(6) 初期化のための信号を分配する．
(7) 回路変換後のネットリストを元の回路部分 A と置き換え，デザイン全体に対して再度論理合成，配置配線等を行う．

手順の 3 以降は自動的に行うことができる．この自動変換について詳細を述べる．

5.2 二相ラッチ化と TF 検出・回復のための回路変換

自動変換の入力となるのは，回路のネットリストと，その AS を保持する FF や RAM の指定である．Rocket ではレジスタ・ファイル，データ・キャッシュ，CSR を AS として指定する．

二相ラッチ化と TF 検出や遅延挿入の位置探索には，回路の最大遅延や最小遅延を得る必要がある．FPGA の場合，LUT の遅延が均質であるから，パスの遅延はパス上の LUT の個数によって計算する．また，遅延素子には 1 入力の LUT を用いる．

まず，二相ラッチ方式への変換を行う．我々は二相ラッチ

化のためのアルゴリズムを開発しており [17,18]，これを用いる．

次に，タイミング制約違反を起こすパスの終端となるラッチを Razor latch へ置き換える．3.2 節で述べたように，DTB を可能にするクロッキング方式では半ステージのクリティカル･パス遅延によって最小サイクル･タイムが決まり，サイクル･タイムの 1/2 を超える遅延のパスが検出対象である．また，ショート･パス問題を起こさないように，Razor latch に至るショート･パスの一部の回路素子を複製し，遅延素子を挿入する．

次に，パイプライン・ラッチに対して，そのラッチから AS に信号が到達するまでの最短サイクル数による順序付けを行う．これは，AS に指定された FF や RAM から，入力方向に幅優先探索で素子を辿り，ラッチを通過した数をラッチごとに記憶すればよい．そして，Razor latch が出力するエラー信号をこの順序が同じものごとに OR ゲート（FPGA なので実体は LUT）を用いて集約し，順序が降順になるように伝搬する．最も AS に近いラッチ群のエラー信号をまとめた信号をコミット・ステージのライト・イネーブルに用いることで，エラー信号は TF の影響を受けた信号よりも早くコミット・ステージまで到達する．伝搬されたエラー信号は回復機構への入力としても使用し，回復処理を駆動する．

次に初期化のための信号を分配する．初期化対象となるのは，AS を除いたパワーオン・リセットの対象である．この時，PC については，再開すべき PC の値にセットされるようにする．再開すべき PC の値は，最後にコミットを行った命令の PC の次の命令の PC であり，これも AS として加えておく必要がある．これはコミットした命令が分岐命令であれば分岐先の PC であり，命令が例外を起こしていた場合は例外に応じたトラップハンドラの先頭 PC である．

5.3 イン・オーダ完了の必要性

基本的には上記のように適用を行うが，Rocket についてはあらかじめ更なるアーキテクチャの変更を要する．これは Rocket が必ずしもイン・オーダ完了するわけではないプロセッサであるためである．

TF 検出・回復手法は，4 章で述べたように，例外処理と同様の機構によって回復を行う．そのため，正確な例外の保証と同様の理由から，イン・オーダ完了を徹底することが望ましい [19].

しかしながら，Rocket はキャッシュ・ミス時や乗算，除算，FPU 除算等のレイテンシの長い命令について，イン・オーダ完了が徹底されていない．例えば，除算命令の直後の命令 i が除算の結果に依存しない場合，命令 i は実行され，レジスタ・ファイルへの書き込みやメモリへのストアを行ってしまう．仮にこのまま TF の発生を想定するならば，除算の途中であり，かつ命令 i がレジスタ・ファイルへの書き込みを完了した後のタイミングにおいて TF が発生した場合，命令 i によって AS が更新されてしまっているため，プロセッサは正しく再開することができない．

したがって，Rocket についてはイン・オーダ完了が徹底されるようにマイクロ・アーキテクチャを変更する必要がある．キャッシュ・ミス時は上流の命令をフラッシュし，除算の場合は後続の命令はデコード・ステージでストールするように変更する．

6. おわりに

我々はティピカル･ケースの遅延に基づいた動作を実現するための手法として，DTB を可能にするクロッキング方式を提案している．これまでは小規模なカウンタに対する適用のみが行われていた．本稿では開発した自動適用ツールを用いてプロセッサへの一部に対して DTB を可能にするクロッキング方式を適用する手順について述べた．

今後は，このプロセッサへの回復機構の実装によって，動作周波数向上を考慮した性能の評価を行う予定である．また，我々は現在，NORCS [20] など様々な技術を取り入れた高効率な out-of-order スーパスカラ･プロセッサの開発を行っており，このプロセッサに DTB を可能にするクロッキング方式を適用し，より詳細な評価を行う予定である．

謝辞 本研究の一部は，文部科学省科学研究費補助金 No. 16H02797 による．

参考文献

[1] 平本俊郎，竹内 潔，西田彰男: 1. MOS トランジスタのスケーリングに伴う特性ばらつき (小特集, CMOS デバイスの微細化に伴う特性ばらつきの増大とその対策)，電子情報通信学会誌，Vol. 92, No. 6, pp. 416–426（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110007227367/⟩ (2009).

[2] Srivastava, A., Sylvester, D. and Blaauw, D.: *Statistical Analysis and Optimization for VLSI: Timing and Power*, Springer Science & Business Media (2006).

[3] Ernst, D., Kim, N. S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K. and Mudge, T.: Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation, *Proc. 36th Annual IEEE/ACM Int'l Symp. Microarchitecture*, pp. 7–18 (online), DOI: 10.1109/MICRO.2003.1253179 (2003).

[4] Bull, D., Das, S., Shivshankar, K., Dasika, G., Flautner, K. and Blaauw, D.: A power-efficient 32b ARM ISA processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation, *ISSCC DIGTECHPAPERS*, pp. 284 –285 (online), DOI: 10.1109/ISSCC.2010.5433919 (2010).

[5] Bowman, K. A., Tschanz, J. W., Kim, N. S., Lee, J. C., Wilkerson, C. B., Lu, S. L., Karnik, T. and De, V. K.: Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance, *IEEE J. Solid-State Circuits*, Vol. 44, No. 1, pp. 49–63 (online), DOI: 10.1109/JSSC.2008.2007148 (2009).

[6] Choudhury, M., Chandra, V., Mohanram, K. and Aitken, R.: TIMBER: Time borrowing and error relaying for online tim-

ing error resilience, *Design, Automation and Test in Europe CONF Exhibition (DATE)*, pp. 1554 –1559 (2010).
[7] 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).
[8] 神保　潮，山田淳二，五島正裕: 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 (2017). *cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming* (*xSIG 2017*) に採択.
[9] 神保　潮，山田淳二，五島正裕: 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，情報処理学会論文誌：コンピューティングシステム， Vol. 10, No. 2, pp. 1 – 12（オンライン），入手先 ⟨http://id.nii.ac.jp/1001/00183237/⟩ (2017).
[10] RISC-V Foundation: RISC-V Foundation | Instruction Set Architecture (ISA) (online), available from ⟨http://riscv.org/⟩.
[11] Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D. A., Richards, B., Schmidt, C., Twigg, S., Vo, H. and Waterman, A.: The Rocket Chip Generator, Technical Report UCB/EECS-2016-17, EECS Dept., UCB (online), available from ⟨http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html⟩ (2016).
[12] Das, S., Tokunaga, C., Pant, S., Ma, W.-H., Kalaiselvan, S., Lai, K., Bull, D. M. and Blaauw, D. T.: RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance, *IEEE J. Solid-State Circuits*, Vol. 44, No. 1, pp. 32–48 (online), DOI: 10.1109/JSSC.2008.2007145 (2009).
[13] 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二相ラッチ生成アルゴリズム，情報処理学会研究報告，Vol. 2014-ARC-211, No. 9, pp. 1–10（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009808089/⟩ (2014).
[14] 五島正裕，倉田成己，塩谷亮太，坂井修一: タイミング・フォールト耐性を持つ Out-of-Order プロセッサ，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. 17–30（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009527308/⟩ (2013).
[15] 吉田宗史，倉田成己，塩谷亮太，五島正裕，坂井修一: タイミング・フォールト耐性を持つ Out-of-Order プロセッサの検出/回復方式，先進的計算基盤システムシンポジウム SACSIS，pp. 10–19（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/170000076897/⟩ (2013).
[16] Bradbury, A., Ferris, G. and Mullins, R.: Tagged memory and minion cores in the lowRISC SoC, *Memo, University of Cambridge* (2014).
[17] 神保　潮，五島正裕: 逆方向カット・エッジのない最小カットを求めるアルゴリズム，情報処理学会論文誌：コンピューティングシステム， Vol. 11, No. 1, pp. 1–11 (2018).
[18] 神保　潮，五島正裕: 逆方向カット・エッジのない最小カットを求めるアルゴリズムの改良，情報処理学会研究報告，Vol. 2018-ARC-230, No. 35, pp. 1 – 6 (2018).
[19] 安藤秀樹: 命令レベル並列処理，Vol. 10 (2005).
[20] Shioya, R., Horio, K., Goshima, M. and Sakai, S.: Register Cache System not for Latency Reduction Purpose, *International Symposium on Microarchitecture (MICRO) (MICRO-43)*, pp. 301–312 (online), DOI: 10.1109/MICRO.2010.43 (2010).