社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－ＡＲＣ－168（８）
2006／6／8

タイミング違反を利用するマイクロアーキテクチャの演算器における遅延を考慮した評価

国武 勇次† 千代延 昭宏† 田中 康一郎‡ 佐藤 寿倫¶

† 九州工業大学大学院 情報工学研究科 〒813-0036 飯塚市川津 680-4
‡ 九州工業大学 マイクロ化総合技術センター 〒813-0036 飯塚市川津 680-4
¶ 九州大学 システムLSI研究センター 〒814-0001 福岡市早良区百道浜 3-8-33-3F
E-mail: † {y-kunitake,chiyo}@mickey.ai.kyutech.ac.jp, ‡ tanaka@cms.kyutech.ac.jp, ¶ toshinori.sato@computer.org

あらまし 回路のクリティカルパスが常には活性化されないことに着目し，積極的にタイミング違反を利用してプロセッサの高速化や省電力化を図る方式を，われわれは検討してきた．残念ながら，これまで行ってきた評価は十分であるとは言えない．プロセッサ全体を評価する際には，回路遅延を配慮できていなかった．一方，回路遅延に配慮する場合には，演算器のみの評価に過ぎなかった．本稿では，演算器における遅延に配慮してプロセッサ全体を評価した結果について述べる．

キーワード 性能ばらつき，回路遅延，シミュレーション，典型値指向設計手法

Considering Circuit Delay in Adders on Evaluation of Constructive Timing Violation

Yuji Kunitake† Akihiro Chiyonobu† Koichiro Tanaka‡ and Toshinori Sato¶

† Department of Artificial Intelligence, Kyushu Institute of Technology, Japan
‡ Center for Microelectronic Systems, Kyushu Institute of Technology, Japan
¶ System LSI Research Center, Kyushu University, Japan
E-mail: † {y-kunitake,chiyo}@mickey.ai.kyutech.ac.jp, ‡ tanaka@cms.kyutech.ac.jp, ¶ toshinori.sato@computer.org

Abstract We have investigated a technique for microprocessors, which achieves both high performance and low power. Based on the observation that critical paths in a circuit are not always active, we aggressively exploit timing violations in the circuit, which do not actually occur. We call the technique Constructive Timing Violation (CTV). Unfortunately, until now, we have evaluated the CTV without considering circuit delay. This paper presents evaluation results of a microprocessor utilizing the CTV, with considering circuit delay in adders.

Keyword Performance variations, Circuit delay, Simulations, Typical-case design methodologies

1. はじめに

ディープサブミクロン化（deep submicron: DSM）の進んだ半導体製造技術においては，従来行われてきた最悪値指向設計は不可能になると予想されている．DSMではノイズやプロセスばらつきが増大し，さらに電源電圧を低下させる必要も生じる．その結果，最悪値指向設計で必要となる設計マージンの確保が困難になる．最悪値ではなく典型値を考慮した，マイクロプロセッサの設計手法が求められている．

建設的タイミング違反手法（Constructive Timing Violation: CTV）[1]は上述の状況を考慮した設計手法であり，そこでは設計者は極めて稀な最悪値ではなく典型値に最適化した設計を求められる．CTVは以下の二つの現象を利用している．一つは，回路のクリティカルパスが活性化されることは稀なことである．もう一つは，クリティカルパスを活性化させる入力は極めて限定されることである．言い換えると，設計時のタイミング制約を緩和させたとしても，現実には回路の動作時にタイミングエラーが生じることは稀である．例えば，約80%のパスの遅延はクリティカルパス遅延の半分以下であると報告されている[2]．CTVは回路レベルでの投機実行を利用しているので，タイミングエラーの結果ロジックに誤りを生じる可能性がある．それに対処するためのフォールトトレラント機構が必要である[3, 4, 5, 6]．タイミングエラーが検出された時に

は，マイクロプロセッサに既に備わっている投機失敗からの回復機構を利用して，プロセッサ状態を回復させる．CTV は動作速度を向上させる目的[1]のみならず，省電力の目的[3]でも利用可能である．本方式は故障確率が高いほど性能が低下するため，これまでに履歴を用いることで CTV を適用する際に被る性能低下を抑制する方法を検討してきた[4]．残念ながらこれまでの評価では，CTV を適用した演算器の評価でのみ回路遅延を考慮しており，プロセッサ全体の評価では回路遅延は正確に配慮されていなかった．本稿では，回路遅延を考慮しながらプロセッサ全体での性能評価を行う．

2. 建設的タイミング違反方式

ディープサブミクロン化により，従来の保守的な設計手法である最悪ケースを考慮した方法では，近い将来 LSI の設計は不可能になると予想されている．ディープサブミクロン化したプロセス技術では，ノイズやプロセスばらつきが増大している．加えて電源電圧を下げる必要があり，さらにノイズ耐性の悪化に拍車をかけている．このような条件化では，最悪ケースを考慮するための設計マージンを確保することが非常に困難である．以上を踏まえ我々は，LSI の設計制約を楽観的に考えてタイミング違反による動作異常の状態を許容し，そのかわりにタイミング違反に対するフォールトトレランス機構を備えることを提案している．これが CTV と呼んでいる手法である[1]．

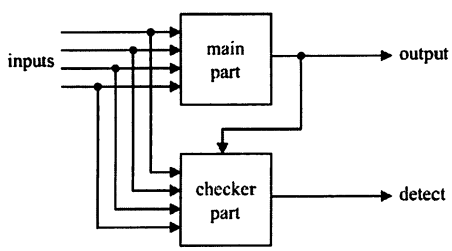

図 1：建設的タイミング違反方式

CTV の基本的な考え方を，図 1 で説明する．CTV では，設計時に検出されたタイミング違反が実行時には発生しないと仮定している．回路レベルで投機的実行を行っており，タイミング違反を検出する機構と違反検出時にプロセッサの状態を正常に回復させる機構とが必要となる．図 1 では回復機構は省略されている．図に示されている通り，CTV 方式で設計された回路はメイン部（図の main part）とチェック部（図の checker part）から構成される．メイン部は元の回路であり，低レイテンシかつ高スループットという高性能を維持できるように設計されるが，タイミング違反が発生する可能性を秘めている．チェック部はタイミング違反を生じないように設計されており，メイン部の入出力が入力されメイン部のタイミング違反を検出する．

これまでに，回路を多重化する方式[1, 3, 5]と，パイプライン処理を利用する方式[6]，そして加算比較器を応用する方式[7]を検討してきた．図 2 に多重化方式の CTV を施された加算器を示す．高周波数を供給されるメイン部加算器と，低周波数を供給される二つのチェック部加算器から構成されている．メイン部とチェック部の加算器の演算結果を比較することで，タイミングエラーを検出できる．高いスループットを維持するために，二つのチェック部加算器が交互にメイン部加算器の結果をチェックしている．

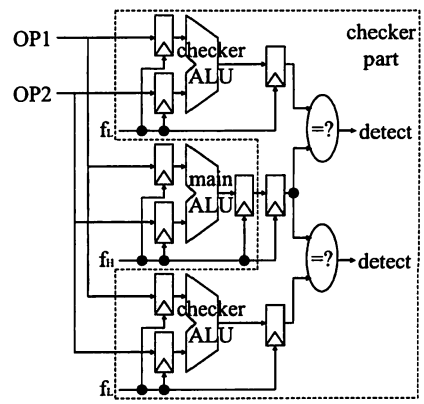

図 2：CTV を施された加算器

タイミング違反が検出されると，何らかの方法でプロセッサの状態を正常に回復する必要がある．そのためには，現在のマイクロプロセッサに既に備わっている投機的実行方式のための機構を利用できる．プログラムの実行中に分岐命令が現われると，分岐の方向が決定される前に予測に基づいて将来実行されると期待できる命令を実行する．分岐予測に失敗した場合には，プロセッサの状態を投機開始前の状態に回復させなければならない．この場合，プロセッサは予測に失敗した命令以降を全て破棄し，命令フェッチからやり直す．容易に判るように，これらはいま必要としているタイミング違反からの回復と非常に類似している．つまり，タイミング違反を生じた命令を投機に失敗した命令であると読み替えることで，回復機構を実現できる．タイミング違反を起こした命令は再実行されないことに注意されたい．正しい結果はチェック部で求められており，それを用いて以降の処理を進める．したがって

処理がデッドロックに陥る心配は無い．この方法を用いれば，タイミング違反からの回復を実現するハードウエアを改めて用意する必要はなく，ハードウエアのオーバヘッドは極めて小さい．

3. 性能低下の抑制手法

信号遅延で生じるタイミングエラーは，実行される命令や CTV を適用した回路の特性によってエラーを生じる命令や演算に偏りがある．この偏りを把握し利用することで，タイミングエラーがあらかじめ起こることを予測する．エラーを生じさせない対策をとることで，プロセッサ状態を回復させるペナルティを被ることを避ける．これまでに，同じ命令が何度も故障を起こす可能性が高い，あるいは，同じ演算が何度も故障を起こす可能性が高い，という特徴を利用した方式を提案している[4].

3.1. 演算結果キャッシュ

演算結果キャッシュ(Result cache: RC)は過去に行った演算を保持している[8]．図 3 に RC の構成を示す．キャッシュアクセスには演算オペランドをハッシュ関数にかけた値を用いる．ハッシュ値の下位ビットをインデクスとし，キャッシュラインにアクセスする．RC のタグにはハッシュ値の上位ビットが格納されている．したがって，ハッシュ値の上位ビットとタグを比較して等しければキャッシュヒットとなりキャッシュに格納されている値を出力する．等しくなければミスヒットとなる．

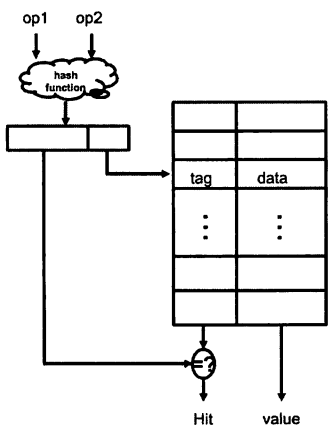

図 3：演算結果キャッシュ

キャッシュアクセスはメイン部 ALU の演算と同時に行われる．RC にヒットした場合，演算結果は RC から獲得された値を採用する．一方，チェック部は RC のヒット／ミスヒットにかかわらず故障検出を行う．キャッシュにヒットした場合，故障が起こるかの検証を行っても故障は検出されない．キャッシュにヒットした演算の検証は無駄のように思える．しかし RC にヒットするか否かで故障検出を行うか否かを決めるのは故障検出機構のパイプラインスケジューリングを複雑にし，故障検出機構の動作速度を低下させてしまう恐れがある．したがって故障の検証を行う必要がない場合でも検証を行うようにする．

本手法で利用する RC は故障を回避する目的で利用する．したがって，キャッシュには過去に故障を起こした演算の正しい結果のみを保持する．もしキャッシュに存在する演算を実行するならば，RC 内の値を実行結果として用いる．これにより正しい結果を得られるため故障を回避できる．ただし性能低下を避けるためには，演算器と同じレイテンシでアクセスできる必要がある．

3.2. 故障履歴バッファ

故障履歴バッファ(Fault History Buffer: FHB)は過去に故障を起こした命令のプログラムカウンタ（PC）を保持するバッファである．図 4 に構成を示す．PC の下位ビットをインデックスとしてバッファにアクセスする．FHB のタグには PC の上位ビットが格納されている．したがって PC の上位ビットとタグを比較して等しければキャッシュヒットとなり，等しくなければミスヒットとなる．ミスヒットの場合は通常の実行を行い故障検出機構で検証を行う．一方 FHB 内に実行される命令の PC が存在する場合には，故障を起こす命令と判断して，安全で低速な故障検出機構に搭載されている ALU で実行する．これにより演算レイテンシは増加するが，故障を回避することができる．FHB へのエントリ追加は故障検出機構で故障が検出された場合に行う．

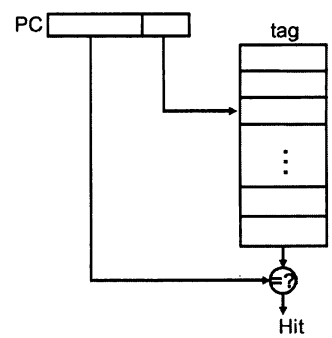

図 4：故障履歴バッファ

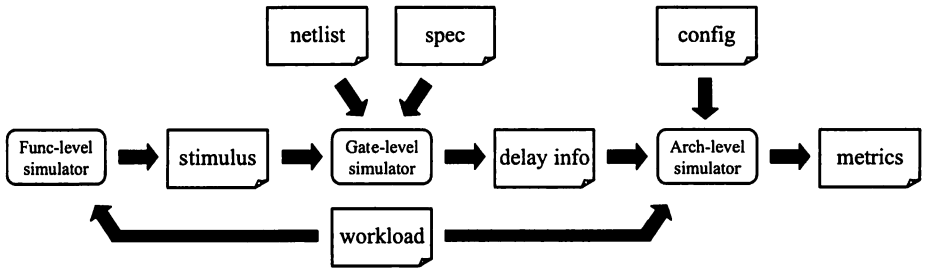

図 6：シミュレーションフロー

4. 評価方法

採用したプロセッサモデルは 4 命令並列のアウトオブオーダ・スーパースカラプロセッサである．CTV を施す対象は，図 2 に示した回路多重化方式に基づく加算器[1, 3, 5]とする．メイン部にはチェック部の 2 倍のクロックを供給する．したがって，タイミングエラーが発生しなければプロセッサ性能は二倍に向上する．タイミングエラーの検出には 2 サイクルのレイテンシを必要とすると仮定する．タイミングエラーが検出されると，分岐予測で用いられる予測失敗からの回復機構を用いてパイプラインをフラッシュする．RC と FHB はそれぞれ 2〜64 エントリおよび 32〜1024 エントリとする．

ベンチマークプログラムには，SPEC2000 から 164.gzip，175.vpr，176.gcc，197.parser，255.vortex そして 256.bzip を選んで使用する．

回路遅延を考慮できるアーキテクチャレベルのシミュレーション環境を構築した[9]．まず最初に SimPoint [10]を用いて，各プログラムの特徴的な 10,000 命令を抽出する．この時には SimpleSclar ツールセット[11]が提供している命令レベルシミュレータを利用する．シミュレーション対象の命令数が非常に少ない理由は，後に述べるようにゲートレベルのシミュレーションを実施する必要があるからである．

続いて，ゲートレベルシミュレーションを実施する．この際には加算器のネットリストが必要である．Verilog-HDL を用いて，図 5 に示す 32 ビットのキャリー選択加算器（Carry Select Adder: CSLA）を設計した．ビット幅の異なる 13 個のキャリー伝播加算器（Ripple Carry Adder: RCA）から構成されている．論理合成時に遅延情報を持つ論理素子ライブラリを使用することにより，回路に遅延を付加させる．今回使用するライブラリは VDEC から提供されている日立製 0.18μm プロセス ASIC ライブラリである．論理合成にはシノプシス社の DesignCompiler を用い，遅延情報を含むネットリストを出力する．得られた遅延情報から最大遅延時間を求め，動作周波数を決定する．シミュレーションにはケイデンス社の Verilog-XL を使用する．

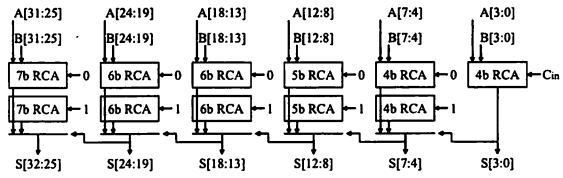

図 5：キャリー選択加算器

最後に，MASE シミュレータ[11]を利用してアーキテクチャレベルのシミュレータを構築する．このシミュレータは，ゲートレベルシミュレーションで獲得された遅延情報を与えられると，内部で遅延情報テーブルを作成する．CSLA への入力オペランドを用いてテーブルを参照すると，回路を考慮した遅延時間が獲得できる．テーブル参照の時間を短縮するために，テーブルにはタイミングエラーを生じるオペランドについてのみ登録する．こうして，回路遅延を考慮できるアーキテクチャレベルのシミュレーションが可能になる．シミュレーションの対象は，SimPoint で抽出された 10,000 命令である．

以上のシミュレーションフローを図 6 に示す．

5. 結果

図 7 にタイミングエラー率を示す．CSLA にのみ CTV が施されているので，加減算命令を対象として調査している．いずれのプログラムでも高いエラー率となっており，平均で 55.9%に至っている．以前に実施した，回路遅延を考慮できないアーキテクチャレベルのシミュレーション結果[4]によると，エラー率が 15%を超える辺りでプロセッサ性能が低下する．このことより，図 6 に示されるエラー率ではプロセッサ性能の改善は望めないと予想できる．また[4]で実施した評価ではタイミングエラー率を 30%と仮定して RC と FHB の評価

を行っているが，それでは正確な評価となっていなかったことが確認できる．

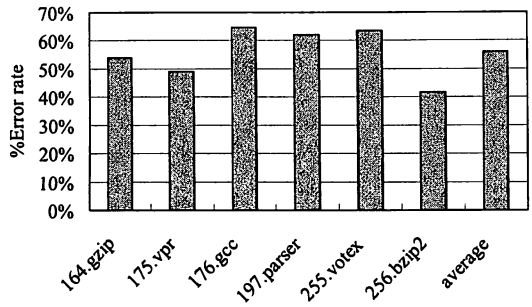

図 7：タイミングエラー率

図 8 にプロセッサ性能をまとめた．CTV を用いない，つまり動作周波数を二倍にしていない，ベースラインのプロセッサモデルの性能からの相対値である．上の予想どおり，ほとんどのプログラムでプロセッサ性能が低下している．平均で 32.5%の低下となっている．性能改善を達成できたのは，比較的タイミングエラー率の小さかった 256.bzip2 のみである．容易に予想されることだが，255.vortex を除いて，タイミングエラー率の大きなプログラムほど，性能低下が著しいということが観察される．

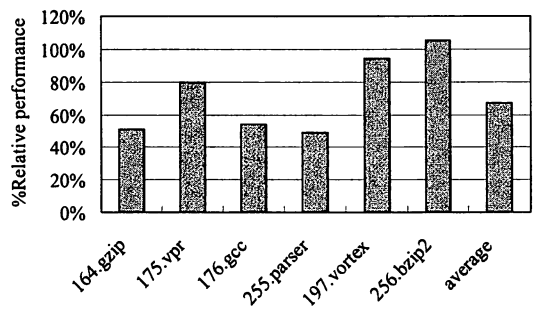

図 8：プロセッサ性能

つづいて RC の効果を評価する．図 9 はシミュレーション結果である．以降の評価ではサイクルあたりのコミット命令数（committed Instructions Per Cycle: IPC）を用いる．縦軸は，ベースラインモデルの IPC からの相対値を示している．したがって，この値が 50%以下になるとプロセッサ性能が低下していることに相当する．RC のエントリ数を 2 から 64 まで変化させて評価した．各プログラムの 7 本のグラフは，各 RC エントリ数に対応した IPC の相対値である．

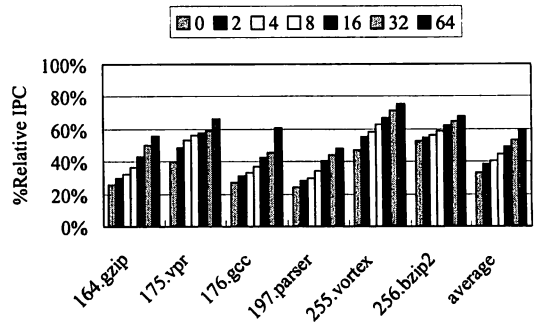

図 9：RC の効果

容易にわかるように，エントリ数の増加にしたがって，単調に IPC が回復されている．しかし，64 エントリでも平均で 60.0%までしか回復できず，特に 255.parser では依然としてベースラインよりも性能が低くなっている．

図 10 は FHB の効果を調査した結果である．同様に IPC の相対値を用いて評価している．FHB のエントリ数は 32～1024 の間で変化させた．RC よりもエントリ数が大きいのは，FHB は RC よりもハードウエアが単純なため，同じアクセス時間で容量を増加できると期待されることが理由である．

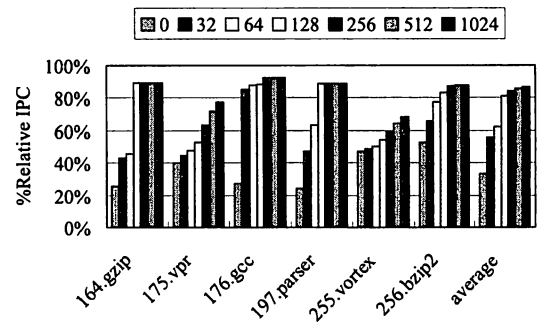

図 10：FHB の効果

175.vpr と 197.vortex を除いて，小さなエントリ数でも効果的に IPC を回復できていることが観察できる．128 エントリあれば，平均で 81.0%まで IPC が回復されている．上記の二つのプログラムについては，128 エントリではようやくプロセッサ性能を回復できるに過ぎない．

図 11 では RC と FHB を組み合わせる効果を調査している．各プログラムの 4 本のグラフは，左からそれぞれ，CTV のみを採用した場合，CTV に 64 エントリの RC のみを追加した場合，CTV に 512 エントリの FHB

のみを追加した場合，そして，CTV に両者を追加した場合のシミュレーション結果を表している．

RC と FHB へのアクセス方法は以下のとおりである．まず RC へアクセスし，ヒットしない場合のみ FHB へアクセスする．RC ヒット時には，RC から獲得された値を演算結果とする．FHB ヒット時には，タイミングエラー検出機構で演算を行った結果を採用する．RC にヒットしない場合には，メイン部の演算結果をタイミングエラー検出機構で検証する．この時にタイミングエラーが検出されれば，RC と FHB の両方にエントリを追加する．

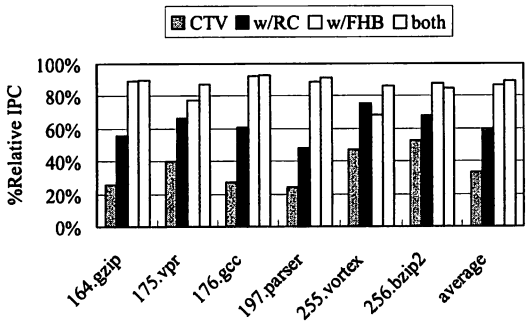

図 11：組合せの効果

興味深いのは，255.vortex では FHB よりも RC の方が効果的であることである．このような場合でも，両者を組み合わせることで相乗効果が得られることがわかる．平均で 89.2%まで IPC を回復できている．ただし，256.bzip2 では両者を組み合わせると，FHB 単独よりも効果が低下している．

6. まとめ

CTV を適用したマイクロプロセッサにおいて，タイミングエラー検出のために生じる性能低下を抑制する気候を評価した．以前にアーキテクチャレベルの評価を実施した際には，回路レベルの遅延を考慮できていなかった．今回，回路遅延を考慮できるアーキテクチャレベルのシミュレータを構築し，提案機構の評価を実施した．シミュレーションの結果，タイミングエラー率は以前に想定した 30%よりも大きく，平均で 55.9%であった．このためプロセッサ性能の低下も著しいものとなったが，64 エントリの RC を用いた場合で平均 60%まで，128 エントリの FHB を用いた場合で平均 81.0%まで，64 エントリの RC と 512 エントリの FHB の組み合わせの場合で平均 89.2%まで，IPC を回復できることを確認した．

今後の課題は，CTV を消費電力削減の目的で利用する時の RC と FHB の効果，特に両機構が消費する電力の影響を調査することである．

謝辞

本研究の一部は，科学研究費補助金（No.16300019, No.176549）の援助によるものです．なお，東京大学 VDEC を通じて提供していただいた株式会社日立製作所製の LSI 設計用ライブラリを使用しています．

文　献

[1] T. Sato and I. Arita, "Give up Meeting Timing Constraints, but Tolerate Violations", COOL Chips IV, 2001.

[2] K. Usami, M. Igarashi, F. Minami, T. Ishikawa, M. Kanazawa, M. Ichida, and K. Nogami, "Automated Low-power Technique Exploiting Multiple Supply Voltages Applied to a Media Processor", IEEE Journal of Solid State Circuit, Vol.33, No.3, 1998.

[3] 佐藤，有田，"遅延故障を考慮したフォールトトレランス技術に基づく低消費電力方式"，信学技報 VLD2001-5, 2001.

[4] 千代延，美馬，佐藤，"タイミング違反を利用した省電力プロセッサにおける履歴を用いた性能低下抑制手法"，情処研報 2005-ARC-167, 2006.

[5] 谷野，佐藤，"建設的タイミング違反方式に基づく ALU の HDL 設計とその評価"，信学技報 ICD2002-212, 2003.

[6] 美馬，佐藤，"建設的タイミング違反方式を適用した ALU の改良とその評価"，情処研報 2004-ARC-159, 2004.

[7] 山原，美馬，佐藤，"タイミング違反を利用した省電力 ALU における違反検出回路の高速化手法とその評価"，火の国情報シンポジウム，2005.

[8] S. E. Richardson, "Exploiting Trivial and Redundant Computation", 11th International Symposium on Computer Arithmetic, 1993.

[9] 国武，千代延，田中，佐藤，"タイミング違反を積極的に利用するプロセッサの評価のための回路遅延を考慮するアーキテクチャレベル評価環境の構築"，DA シンポジウム，2006.

[10] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: faster and more flexible program analysis", Workshop on Modeling, Benchmarking and Simulation, 2005.

[11] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling", IEEE Computer, Vol. 35, No. 2, 2002.