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

国武, 勇次
九州工業大学情報工学部知能情報工学科

千代延, 昭宏
九州工業大学情報工学部知能情報工学科

田中, 康一郎
九州工業大学マイクロ化総合技術センター

佐藤, 寿倫
九州大学システムＬＳＩ研究センター

出版情報：DAシンポジウム, pp.169-174, 2006-07. DAシンポジウム
バージョン：
権利関係：

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

国武勇次† 千代延昭宏† 田中康一郎†† 佐藤寿倫‡

† 九州工業大学 情報工学部 知能情報工学科
†† 九州工業大学 マイクロ化総合技術センター
‡ 九州大学 システムＬＳＩ研究センター

我々は，タイミング制約を緩和することで，高性能かつ低電力なマイクロプロセッサの実現を目指している．タイミング違反が生じる可能性があるが，フォールトトレラント機構により動作が保証されている．この方式を評価するためには回路遅延を考慮できる必要がある．しかし，アーキテクチャの評価にはゲートレベルによる詳細な検証は必要ないうえ，評価時間が膨大になる問題がある．そこでアーキテクチャレベルのシミュレーションにゲートレベルの回路遅延を考慮したシミュレーションを組み合わせることを試みる．アーキテクチャレベルで回路遅延を評価できる環境が構築できたが，この環境を検証したところ，いくつかの制約があることが判明した．本稿では，構築された評価環境とその検証における考察について詳細に述べる．

Building an Architectural-level Simulator for Evaluating Timing-speculative Processors with Considering Circuit Delay

Yuji KUNITAKE† Akihiro CHIYONOBU† Koichiro TANAKA†† Toshinori SATO‡

† Department of Artificial Intelligence, Kyushu Institute of Technology
††Center for Microelectronic Systems, Kyushu Institute of Technology
‡System LSI Research Center, Kyushu University

We are investigating high-performance and low-power processors by speculating timing constraints in circuits. While timing violations might occur, a fault tolerant mechanism guarantees every instruction is correctly operated. To evaluate this method, circuit delay must be considered at the architectural level. However, architectural-level evaluations do not require detailed verifications that gate-level evaluations provide. Moreover, gate-level evaluations require huge amount of simulation time, and hence they are inappropriate for system designs. Considering the above, we propose to combine gate-level simulations with architectural-level simulations. We build a prototype architectural-level simulator, which can consider circuit delay within tolerable simulation time, and find it has several constraints in the estimate of accurate circuit delay. This paper describes how the simulator is constructed and what should be considered in its evaluations.

1 はじめに

半導体製造技術はディープサブミクロン化が進展している．これに伴い，あらゆる最悪条件を考慮した保守的な設計が困難になってきた．これはノイズやプロセスばらつきの増大，また電源電圧が低下の傾向を示しているため，最悪条件を考慮するための設計マージンが小さくなっているためである．

このような状況の中，高性能かつ低消費電力なマイクロプロセッサが求められている．しかし，高性能化と低消費電力化はトレードオフの関係にあり，性能の向上に伴い消費電力も増加してしまう．

我々は，高性能，省電力，高信頼なマイクロプロセッサを実現するための技術として建設的タイミング違反方式 (Constructive Timing Violation:CTV)[1, 2, 3] を検討している．LSI

を正常に動作させるための設計制約を緩和することで生じる故障状態を，フォールトトレラント機構を備えることで動作を保証する方式である．CTV を評価するには，回路遅延を考慮する必要がある．それにはゲートレベルでの評価が一般的である．しかし，アーキテクチャの評価にはゲートレベルでの詳細な検証は必要ないうえ，マイクロプロセッサ全体をゲートレベルで評価すると評価時間が膨大になる問題がある．そこで，本稿では CTV を評価するためにアーキテクチャレベル・シミュレーションにゲートレベル・シミュレーションの結果を反映させることを試みる．

以下本稿の構成について説明する．2 章で CTV について述べる．3 章で CTV を評価するためのシミュレーション環境の構築方法について述べ，4 章で構築した評価環境の検証を行う．5 章で今後の課題を述べ，6 章で本稿をまとめる．

2　建設的タイミング違反方式

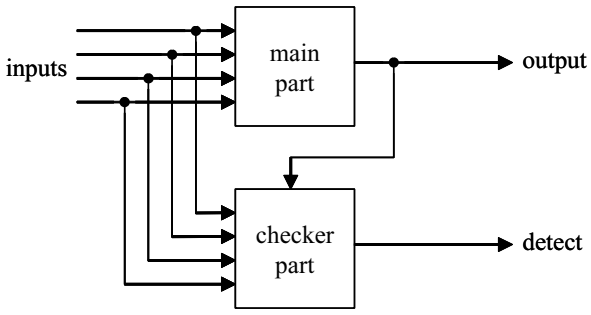

図 1: 建設的タイミング違反方式

CTV の基本的な考え方を図 1 を用いて説明する．CTV は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回路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 1 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 1 の `main part`) とチェック部 (図 1 の `checker part`) から構成される．メイン部はもとの回路であり，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．一方，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．

回復機構の実現は，例えばマイクロプロセッサでは容易である．ある命令でタイミング違反が検出された場合，当該命令に依存する命令は正しい実行結果を得るために再実行されなければならない．このプロセスはマイクロプロセッサに既に備わっている投機実行失敗からの回復に類似する．したがってタイミング違反を起こした命令を投機実行に失敗した命令に読み替えることでマイクロプロセッサを正しい状態に回復できる．

例えば以下のように，CTV を消費電力削減の目的に利用できる．CMOS 回路の消費電力は，充放電電力，リーク電流電力，短絡電流電力に分けられる．近年その問題が注視されているのはリーク電流電力であるが，本稿では依然として消費電力全体の大きな部分を占める充放電電力に注目する．充放電電力 P_{active} とゲート遅延 t_{pd} は以下の式により与えられる．

$$P_{active} \propto f C_{load} V_{dd}^2 \tag{1}$$

$$t_{pd} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \tag{2}$$

f はクロック周波数，C_{load} は負荷容量，V_{dd} は電源電圧で，V_{th} はデバイスの閾値電圧である．α はキャリアの速度飽和を与えるパラメタで，近年の MOSFET では 1.3–1.5 の値を取る．式 (1) から判るように，電源電圧を下げることが電力消費を小さくする最も有効な方法である．しかし，式 (2) から判るように同時にゲート遅延が増加してしまう．マイクロプロセッサの性能を維持するためには，電源電圧を下げてもクロック周波数を変えないことが必要となる．しかしそれはタイミング違反を発生させ，正常な動作を保証できない．そこで我々は上述の CTV を採用することで動作を保証する．

3　シミュレーション環境の構築

3.1　アーキテクチャレベルとゲートレベルのシミュレーション

CTV ではタイミング違反を緩和することで動作周波数の速度向上を計り，それに伴い生じる故障状態をフォールトトレラント機構を備えることで動作を保証している．このため CTV を評価するためには回路遅延を考慮したシミュレーションが必要となる．それにはゲートレベル・シミュレーションを行うことが一般的である．しかし，アーキテクチャの評価にはゲート

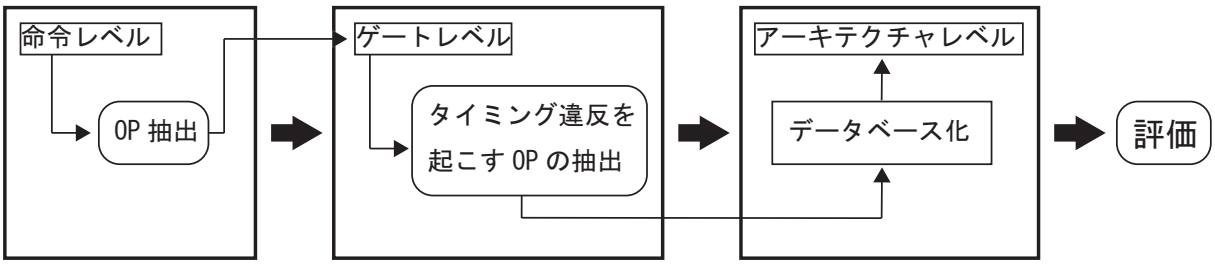

図 2: シミュレーション環境構築の流れ

レベルでの詳細な検証は必要ない上，マイクロプロセッサ全体を評価すると評価時間が膨大になってしまう．そこで，一般にマイクロプロセッサの評価に用いられるアーキテクチャレベル・シミュレーションに，回路遅延を考慮したい CTV を適用した部分のみをゲートレベルでシミュレーションした結果を反映させることを試みる．これによりゲートレベル・シミュレーションの対象となる回路規模を抑えつつ，回路遅延を反映したアーキテクチャ・シミュレーションが可能になる．次節で評価環境の構築手法について説明する．

3.2 評価環境の構築手法

図 2 にシミュレーション環境の構築の流れを示す．シミュレーション環境の構築には，三つの工程がある．第一工程で，アーキテクチャレベル (あるいはそれ相当) のシミュレーション結果を利用して，ゲートレベル・シミュレーションに必要なテストベンチを生成する．続いて第二工程で，ゲートレベル・シミュレーションにより回路遅延情報を収集する．最後に第三工程で回路遅延情報をアーキテクチャシミュレータに取り込み，回路遅延を考慮したアーキテクチャレベル・シミュレーションを実施する．

CTV を ALU に適用する場合の評価環境の構築について述べる．第一に，命令レベル・シミュレーションにより，実行されるオペランドを抽出する．第二に，第一工程で抽出したオペランドをゲートレベルでシミュレーションしタイミング違反を発生するオペランドの組合せを抽出する．第三に，第二工程で抽出したタイミング違反を発生するオペランドの組合せをデータベース化する．

第一工程の命令レベル・シミュレーションと第三工程のアーキテクチャレベル・シミュレーションには，SimpleScalar[4] を用い，命令セットは PISA(Portable Instruction Set Architecture) とする．ベンチマークプログラムには SPEC CINT2000 から表 1 に示す 6 個を用いる．

表 1: ベンチマークプログラム

program	input set
164.gzip	ref/input.program
175.vpr	ref/net.in,ref/arch.in
176.gcc	ref/200.i
197.parser	ref/ref.in
255.vortex	ref/lendian1.raw
256.bzip2	ref/input.source

第一工程で SimPoint[5] を用いて，各ベンチマークでシミュレーションを行うポイントを決定する．SimPoint は，理想的なシミュレーションポイントを選択するためのシミュレーション分析ツールである．インターバル (命令間隔) を指定することでインターバルごとのシミュレーションポイントと，各ポイントの実行頻度を分析できる．今回の評価でインターバルを 10,000 命令として各ベンチマークのシミュレーションポイントを調査し，最も実行頻度の高いポイントから 10,000 命令分のアーキテクチャレベル・シミュレーションを実行し，オペランドの抽出を行う．

第二工程のゲートレベル・シミュレーションでは，32*bit* の桁上げ選択加算器 (Carry Select Adder: CSLA) を用いる．図 3 に示すように遅延情報を含まない理想的な CSLA と遅延情報を含んだ CSLA を並列に動作させ，出力された演算結果を比較する．結果が一致しないものをタイミング違反が発生するオペランドの組合せとして抽出する．CTV を CSLA に適用した際のタイミング違反の発生率は先行研究 [6] によって調査されている．動作周波数が 1.4 倍を越えない範囲ではほとんどタイミング違反は発生せず，2 倍にしても 30%以上のエネルギー削

表 2: シミュレーションモデル

命令フェッチ幅	4 命令
分岐予測機構	bimodal, 1024 エントリダイレクトマップ BTB, コミット時更新 ミスペナルティ3 サイクル, リターンアドレス・スタック 8 エントリ
命令ウィンドウ	リザベーションステーション 16 エントリ リオーダバッファ16 エントリ, ロード・ストアキュー 8 エントリ
命令ディスパッチ幅	4 命令
命令コミット幅	4 命令
機能ユニット数	4 iALU's, 1 iMUL/DIV, 2 Ld/St, 4 fpALU's, 2 fpMUL/DIVs
レイテンシ	iALU 1/1, iMUL 7/1, iDIV 12/9, Ld/St 1/1, fADD 4/1
(全/投入間隔)	fCMP 4/1, fCVT 3/1, fMUL 4/1, fDIV 12/9, fSQRT 18/15
レジスタファイル	32 ビット整数レジスタ 32 本, 32 ビット浮動小数点レジスタ 32 本
データキャッシュ	16K 4 ウェイ・セットアソシアティブ, ライン幅 32 バイト
命令キャッシュ	16K ダイレクトマップ, ライン幅 32 バイト
2 次キャッシュ	共用, 256K 4 ウェイ・セットアソシアティブ, ライン幅 64 バイト

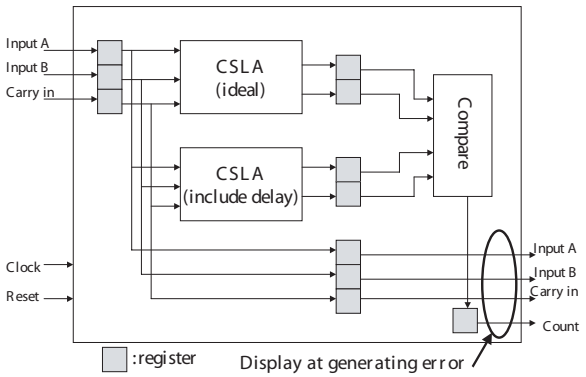

図 3: 検証に作成した回路

減効果が期待できる．よって，ALU のタイミング制約を満足する動作周波数の 2 倍の動作周波数を供給するように設定し第一工程で抽出したオペランドを入力としてゲートレベル・シミュレーションを行う．対象命令は，整数 ALU で実行される ADD 命令と SUB 命令とする．

また，回路の設計およびゲートレベル・シミュレーションの環境は以下の通りである．論理素子ライブラリには日立製 $0.18\mu m$ プロセス ASIC ライブラリを使用する．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と回路解析結果を出力する．シミュレーションにはケイデンス社の Verilog-XL を用いる．

第三工程では，まず第二工程で得られた情報を用いて，オペランドとタイミング違反を結びつけるデータベースを構築する．このデータベースをアーキテクチャレベル・シミュレーションでは以下のように使用する．実行されるオペランドがデータベース内に存在する場合は，タイミング違反が起こったとして投機失敗からの回復を行わせる．一方，オペランドがデータベース内から見つからない場合は，回路レベルの投機実行に成功したと見なす．アーキテクチャレベル・シミュレータにはオペランドでデータベースを検索，データベースから当該オペランドが見つかった場合に投機失敗の機構を呼び出す機能を付け加える．このようにすることで，シミュレーション時間をアーキテクチャレベル・シミュレーション程度に抑えつつ，ゲートレベル・シミュレーションに匹敵する精度で CTV の評価が可能になると考えられる．

アーキテクチャレベル・シミュレーションで用いた構成を表 2 に示す．故障回復には分岐予測失敗からの回復機構を利用し，故障した命令以降のパイプライン中にある命令を全て破棄する．

4 シミュレーション環境の検証

構築したシミュレーション環境を評価するために，ゲートレベルでのタイミング違反発生率とアーキテクチャレベルでのタイミング違反発生率を比較する．図 4 にゲートレベルとアーキテクチャレベルのタイミング違反発生率を示す．縦軸はそれぞれのシミュレーションでのタイミング違反発生率，横軸は各ベンチマークを示す．gate1 は第二工程のゲートレベルでのタイミング違反発生率，arch は第三工程のアーキテクチャレベルでのタイミング違反発生率を表す．

図 4 から分かるようにタイミング違反の発生率に大きな違いが見られる．この理由を，まず

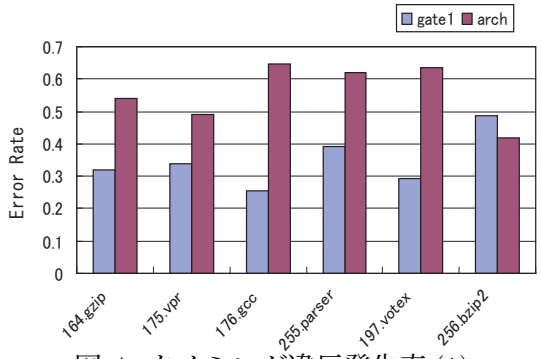

図 4: タイミング違反発生率 (1)

以下のように推測した．アーキテクチャレベル・シミュレーションでは，故障回復機構を用いてタイミング違反を起こした命令の後続命令を再実行する．これにより実行される命令が増加してオペランド列が変化するために，両者の故障率が大きく異なると予想される．

この予想を確認する目的で，ゲートレベルとアーキテクチャレベルを同じオペランド列でシミュレーションすることを試みる．そのために，第三工程のアーキテクチャレベル・シミュレーションから実行されるオペランド列を抽出する．抽出したオペランド列を用いてゲートレベルでシミュレーションする．図 5 に第三工程のアーキテクチャレベル・シミュレーションから抽出したオペランド列を用いて，ゲートレベルでタイミング違反発生率を測定した結果を示す．arch は第三工程のアーキテクチャレベルでのタイミング違反発生率を表す．gate2 は第三工程で実行されたオペランド列を使って測定したゲートレベルでのタイミング違反発生率を表す．

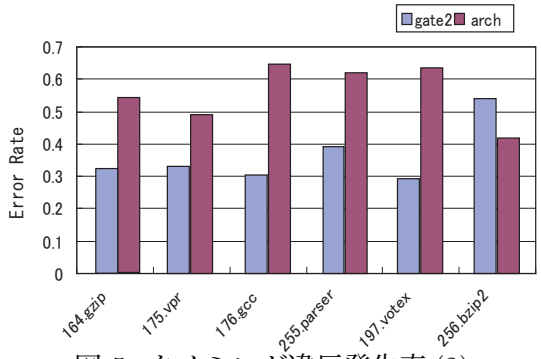

図 5: タイミング違反発生率 (2)

図 5 から分かるように，図 4 とほぼ同じような結果が得られた．つまり，故障回復による命令の再実行による実行命令数の増加とオペランド列の変化が直接の原因ではないことが判る．

この結果を受けて，続いて以下のように理由を考察した．ゲートレベル・シミュレーションでは，同一オペランド同士の演算でも，その直前に演算されたオペランドの組合せによって回路遅延に違いが生じる．このため，同一オペランドの演算でもタイミング違反が発生する場合と発生しない場合が存在する．一方アーキテクチャレベル・シミュレーションでは，同一オペランドで常にタイミング違反を発生させている．これがアーキテクチャレベル (arch) とゲートレベル (gate2) のタイミング違反発生率が異なる要因と考えられる．

そこで，ゲートレベルとアーキテクチャレベルのオペランドの組み合わせとその組み合わせがタイミング違反を起こすかどうかを比較した．その結果を表 3 に示す．表の arch total error はアーキテクチャレベルでタイミング違反を起こした命令数を表し，difference はアーキテクチャレベルでタイミング違反を起こしているオペランドの組合せのうちゲートレベルでタイミング違反を発生していないオペランドの組合せの総数を表す．

表 3: タイミング違反発生率の異なる要因

program	arch total error	difference
164.gzip	8,058	3,707
175.vpr	10,215	4,706
176.gcc	19,047	11,547
197.parser	15,331	7,999
255.vortex	19,638	11,482
256.bzip2	2,992	1,047

表 3 から分かるように，アーキテクチャレベルでタイミング違反を起こしたケースのおよそ半分でゲートレベルではタイミング違反を起こしていない．この結果から，直前の演算結果が現在の遅延に影響を与えていることが確認できた．これがアーキテクチャレベルとゲートレベルのシミュレーションでタイミング違反発生率が大きく異なっている理由である．

例外として，bzip2 ではアーキテクチャレベルよりゲートレベルでタイミング違反が多い．この理由の考察は，今後の課題である．

これらの結果から構築した評価環境の制約をまとめる．

1. 構築した環境のアーキテクチャレベル・シミュレーションでは，タイミング違反が先行の演算結果に依存することを再現できない．

2. 第二工程のゲートレベルと第三工程のアーキテクチャレベルでは，オペランドが現れる順序が同じとは限らない．

3. アーキテクチャレベルで現れるオペランドを，ゲートレベルで網羅できている保証はない．例えば，タイミング違反によって得られた間違った結果を，入力オペランドとする演算を実行できない．

5 今後の課題

本研究では，回路遅延を考慮したアーキテクチャレベルの評価環境を構築することを目指した．ゲートレベルでタイミング違反を発生するオペランドの組み合わせをデータベース化し，アーキテクチャレベル・シミュレーションで利用することを考えた．しかし，そのような評価環境を構築するに当たって，いくつかの制約が判明した．

まず，タイミング違反が先行の演算結果に依存することを，アーキテクチャレベルでは再現できない．この制約は，先行する演算結果までを考慮してデータベースを構築することによって解決できるが，先行の演算結果を考慮するとデータ量が膨大になってしまうため現実的ではないかもしれない．

次に，アーキテクチャレベルで現れるオペランドをゲートレベルで網羅できる保証がないという制約がある．ゲートレベルとアーキテクチャレベルでは，オペランドが現れる順序が同じとは限らないし，また故障回復による命令の再実行を正確に把握することが難しいためである．

これらを解決するには，ゲートレベルでタイミング違反を起こしたオペランドをアーキテクチャレベルのオペランドに一対一で対応付けできる手法の確立が必要である．その手法の実現方法として，ゲートレベルとアーキテクチャレベルを協調させることでシミュレーションを行うことを検討している．これは，アーキテクチャレベルでシミュレーションを行う時，遅延の考慮が必要な部分だけをゲートレベルで実行しその結果をアーキテクチャレベルで利用する手法である．

6 まとめ

CTV は，タイミング違反を正確に発生させるモデルでの評価が必要である．このため回路遅延を考慮してシミュレーションを行う必要がある．しかし，ゲートレベルでのシミュレーションは低速なため，アーキテクチャレベルで回路遅延を考慮し評価を行うことが望ましい．本研究では，回路遅延を考慮する部分のみゲートレベルでシミュレーションし，遅延情報をデータベースとして獲得した．このデータベースを利用することで回路遅延をアーキテクチャレベルで考慮した．しかし，データベースにより全てのタイミング違反を考慮することが困難であることが判明した．この問題を解決すればゲートレベルにおける評価時間を短縮し回路遅延を考慮したアーキテクチャの評価が可能となる．

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．

参考文献

[1] 谷野, 佐藤, 有田, “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.

[2] 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.

[3] 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.

[4] E.Larson, S.Chatterjee, and T.Austin, “MASE: A Novel Infrastructure for Detailed Microarchitectural Modeling,” International Symposium on Performance Analysis of Systems and Software, 2001.

[5] 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.

[6] 田中，川原，国武，永尾，重松，増永，千代延，佐藤，有田，“リコンフィギャラブルシステム RICE と SystemC による設計環境,” Electronic Design and Solution Fair ユニバーシティプラザ予稿集，2006.