Vol.2011-SLDM-150 No.11
2011/5/19

形式的検証を用いた プロセッサエラー回復機構の耐性評価手法の検討

清水修一[†1] 松本剛史[†2] 藤田昌宏[†2,†3]

プロセッサにおいてエラーが発生した場合，そこから正常な状態へ確実に復帰することが求められ，実際に様々なエラー回復機構が実装・提案されている．本稿では形式的検証を用いたプロセッサエラー回復機構の耐性評価手法の提案と評価結果の考察を行う．提案手法では，CLU(the logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions) という論理を用いてエラー回復機構を備えたプロセッサをモデル化し，様々なエラー発生条件に対してエラー回復が正しく実行されるかをモデルチェッカにより検証する．実験を通して，リセットによるエラー回復機構をもつプロセッサモデルに対するエラー耐性の評価を行った．

Evaluation Method for Error Recovery Mechanisms using Formal Verification

Shuichi Shimizu,[†1] Takeshi Matsumoto[†1]
and Masahiro Fujita[†2,†3]

When an error occurred in processor, it is required to recover from the error. Recently, several error recovery mechanisms are implemented. In this paper, We are proposing evaluation method for error recovery mechanisms using formal verification, and discussing about evaluation result. In our proposing method, modeling processor with error tolerance mechanisms using CLU (logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions), and checking whether error recovery was executed correctly by model checker. Through the experimentation, we evaluated an error tolerance of processor architecture models.

1. はじめに

製品として出荷されたプロセッサにバグが見つかり問題となるケースがあり，世代が進むごとに，発見されるバグが 3〜4 倍増大していると報告されている．[1] プロセッサには高速化のための様々な機能が組み込まれているが，それにより設計が複雑化している．そのため，形式的手法によって検証を行う場合，その計算量が非常に大きくなるため，プロセッサの RTL 回路をそのまま検証することはできない．

そのため，より抽象的なアーキテクチャレベルでプロセッサの正しさを形式的に検証するアプローチがとられている．具体的な手法としては，十分に正しさが検証された，単純なプロセッサのアーキテクチャと，高速化のための様々な機能を追加した複雑なプロセッサのアーキテクチャをモデル化し，その 2 つのモデルに対し，命令実行結果の等価性を検証することにより，複雑なアーキテクチャを持つプロセッサの検証を行う．

本稿では，プロセッサアーキテクチャのモデル化と検証の基本的な考え方を紹介し，回路遅延によって生じるタイミングエラーに対する回復機構についてモデル化し，エラーとして誤った値を入れ検証を行うことでエラー耐性の評価を行った結果について考察する．プロセッサアーキテクチャはプロセッサリセットを利用した回復機構[6] をもつ Out-of-Order スーパースカラプロセッサとし，どのようにエラーを入れるべきか検討を行った．

まず 2 節ではプロセッサアーキテクチャの検証手法の概要について説明し，3 節で関連研究について述べる．続いて 4 節でタイミングエラーのモデル化とエラーを代入することによる実験手法の検討と実験結果について考察し，最後に 5 節においてまとめを行う．

2. プロセッサアーキテクチャの検証

プロセッサアーキテクチャの検証において，ALU 等の複雑な設計の詳細部分まで形式的検証手法を適用することは，計算量が増大してしまうためできない．そのため，一度抽象度の高いモデルを作成した上でそのモデルに対し形式的検証手法を適用する．

†1 東京大学大学院工学系研究科電気系工学専攻
Dept. of Electrical Engineering and Information Systems, The University of Tokyo
†2 東京大学 大規模集積システム設計教育研究センター
VLSI Design and Education Center, The University of Tokyo
†3 科学技術振興機構 戦略的創造研究推進事業 CREST
Core Research for Evolutional Science and Technology, Japan Science and Technology Agency

$$
\begin{aligned}
bool-expr ::= & \ \boldsymbol{true} | \boldsymbol{false} | \neg bool-expr \\
& | (bool-expr \wedge bool-expr) \\
& | (int-expr = int-expr) | (int-expr < int-expr) \\
& | \ predicate-expr(int-expr, \cdots, int-expr) \\
int-expr \ ::= & \ int-var | ITE(bool-expr, int-expr, int-expr) \\
& | \boldsymbol{succ}(int-expr) | \boldsymbol{pred}(int-expr) \\
& | function-expr(int-expr, \cdots, int-expr) \\
predicate-expr ::= & \ predicate-symbol \\
& | \lambda int-var, \cdots, int-var.bool-expr \\
function-expr ::= & \ function-symbol \\
& | \lambda int-var, \cdots, int-var.int-expr
\end{aligned}
$$

図 1 CLU Syntax[2)]

2.1 CLU を用いたモデル化

プロセッサ等の無限に状態遷移をするシステムをモデル化し検証するために，CLU(Counter arithmetic with Lambda expressions and Uninterpreted functions)[2)] という論理が提案されている．Uninterpreted function は，変数の値や演算の内容を解釈せずに対象を表現する方法である．例えば，$ALU(instr)$ という関数を定義した場合，この関数は ALU と $instr$ という記号のみにより評価される．従って uninterpreted function を用いると，プロセッサの ALU を関数として抽象化することができる．図 1 に CLU 構文を示す．CLU には true/false の 2 値をもつ bool 型，integer 型，それから integer 型から bool 型への関数である predicate 型，integer 型から bool 型への関数である function 型の 4 種類のデータタイプが定義されている．ITE(if-then-else) 演算子は $ITE($ 評価式 $, x1, x2)$ により評価式の真偽によって $x1, x2$ のどちらかの値を返す．succ() と pred() はそれぞれ，括弧内の記号値の前後の値を表わし，実際の数値を用いずに値の大小について比較できる．

2.1.1 例:CLU を用いたメモリモデル

CLU を用いたシステムのモデル化の例としてメモリを例に挙げる．ここでのメモリとはアドレスが与えられたときに，そのアドレスに対応したデータの対応を表現した単純なものである．ここではラムダ式を用いてメモリをモデル化している．ラムダ式は関数を簡易に書き表わす方法であり，ラムダ式 $\lambda x.(x+2)$ は関数 $f(x) = x + 2$ を示す．よってメモリのモデル M は

$$M' = \lambda addr.ITE(addr = A, D, M(addr)) \quad (1)$$

と表現できる．A はアドレス，D はデータを指し，M' は与えられたアドレス addr に新たなデータ D が書き込まれたときのメモリの状態を表わしている．

2.2 Out-of-Order プロセッサの CLU を用いた抽象化

文献[3)] で CLU による Out-of-Order プロセッサの検証が提案されている．検証には UCLID という検証システムを利用しているが，本稿でも UCLID による検証を行っている．UCLID については後述する．

モデルを記述する際に抽象化された Out-of-Order プロセッサアーキテクチャを図 2 に示す．Out-of-Order とはフェッチ・デコードをした命令を逐次実行するのではなく，実行速度を上げるため命令順序を入れ替えて実行するアーキテクチャを指す．

ここでは抽象化によるモデルの簡略化のため，リザベーションステーションと ROB(Re-Order Buffer) を統合している．命令実行の流れとしては，まずプログラムメモリからフェッチ/デコードされた命令が，ROB にプログラムカウンタや ROB エントリの有効性を示す制御ビットなどとともにディスパッチされる．次にオペランドの有効性が確認できた命令から逐次実行される．この実行は Out-of-Order 実行であるが，ROB がキューであるため，レジスタへのライトバックは In-Order で行われる．

2.3 UCLID による検証

UCLID[7)] は CLU をベースとしてモデル記述ができる検証システムである．ユーザは UCLID モデル記述言語で検証対象のプロセッサモデルや，プロパティなど検証における条件を記述した.ucl という拡張子のファイルを作成し，UCLID の入力として与える．すると UCLID の内部では Front End が字句解析・構文解析を行い，続けて記号シミュレーションが実行される．外部の BDD ツールまたは SAT チェッカにより判定され，検証結果と反例が生成される．

[3)] ではプロセッサの検証内容として以下の 3 つが挙げられている．

- Bounded Model Checking of Processor
- Correspondence Checking
- Inductive Invariant Checking

モデルは 1 サイクルごとの状態遷移モデルとして記述されているので，指定したサイクル数で記号シミュレーションを行うことができる．Bounded Model Checking による方法では，これを利用し毎サイクルごとプロパティのチェックを行う．

Correspondence Checking は検証対象のプロセッサと，比較のための ISA モデルに同じ

Vol.2011-SLDM-150 No.11
2011/5/19

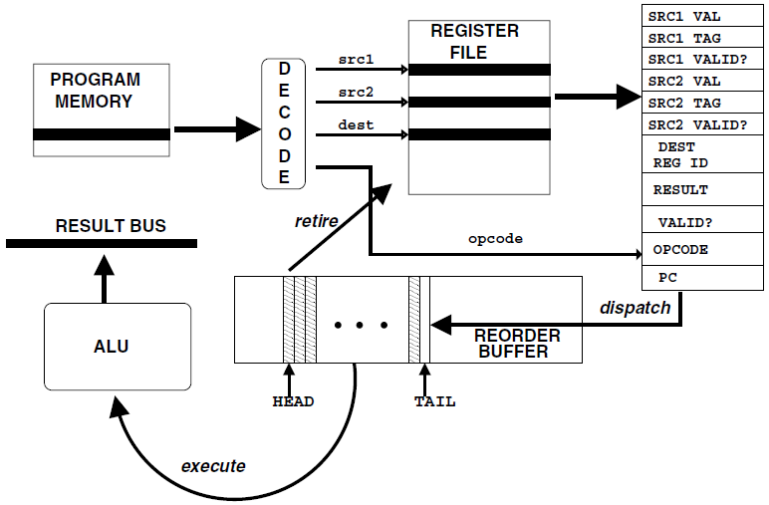

図 2 Out-of-Order Superscalar Processor[3)]

命令を実行させ，最終的なプログラムカウンタとレジスタファイルが等価であるかを見る方法である．

最後に Inductive Invariant Checking は検証対象が常に満足していなければならない条件を検証する方法であり，任意状態から 1 サイクル記号シミュレーションをした後の，前後の状態において共にプロパティが満たされているかを検査する．

3. 関 連 研 究

文献[4)] ではエラー回復機構に RazorII[8)] を用いた Out-of-order プロセッサの検証を行っている．パイプラインの各ステージごとにエラーを任意値により発生させ，エラーの伝搬をモデル化している．本稿と同様に UCLID を用いて bounded model checking を行い，加えてマルチプレクサによるモデルの最適化手法についても提案している．

5) ではエラー回復機構に RazorII を用いた Out-of-Order スーパースカラプロセッサの検証を行っている．エラー発生はコミットステージで，ステップ毎に任意にエラーを発生させ，誤ったデータを生成してレジスタファイルに伝えている．例外処理や分岐予測も考慮したモデルを作成し，correspondence checking による safety property と liveness property を検証している．

3.1 Razor

本稿で扱うプロセッサはエラー回復を行うアーキテクチャなので，既存のエラー回復機構について触れておく．

Razor[8)] はタイミングエラー等のエラーからの回復を，回路レベルで行う手法である．エラーの検知にはクロックのネガティブエッジでラッチする，シャドウラッチと呼ばれる Flip-Flop を用いる．メインの FF とシャドウラッチの値が異なっていた場合，タイミングエラーが発生し不正な値をラッチしたと検知し，シャドウラッチの正しい値をメイン FF に入力することで回復させる．

Razor の改良版として RazorII[8)] がある．これは回路レベルではエラー検知のみを行い，プロセッサがパイプラインで動作していることを利用し，アーキテクチャレベルでの回復を実現させている．エラーがパイプラインのライトバックステージまで伝搬すると，フェッチステージの PC をライトバックステージの PC で上書きしエラーが発生した命令を再実行する．

4. エラーリカバリ機構とエラー発生のモデリング

4.1 プロセッサリセットによるエラー回復

データパス系においてタイミングエラー等が発生した場合には，前述の Razor[8)] 等を用いたエラー回復が有効である．しかし，制御系においてもエラーが起こる可能性はあり，その場合プロセッサは予期していないアーキテクチャステートへ遷移し，異常動作を引き起こしてしまう．そこで文献[6)] においてプロセッサリセットを利用したエラーリカバリ機構が提案されている．図 3 に概念図を示す．

検知されたエラーはクリティカルパスとならないよう RazorII を利用しパイプライン化されたフォルト通知パスにより，コミットステージまで伝わる． コミットステージにエラー検知信号が届くとレジスタファイルへの書き込みは禁止され，プロセッサリセットが実行される． リセットはプログラムカウンタやレジスタファイル等を除くレジスタをリセットするために部分的に行われる． 最終的にコミットステージの命令のプログラムカウンタをロールバックし，正しく実行されなかった命令を再度実行することでエラー回復を実現している．

4.2 エラー発生と回復機構のモデル化

前述のプロセッサリセットによるエラー回復機構をもつプロセッサのモデルを記述し，UCLID による検証を行った．

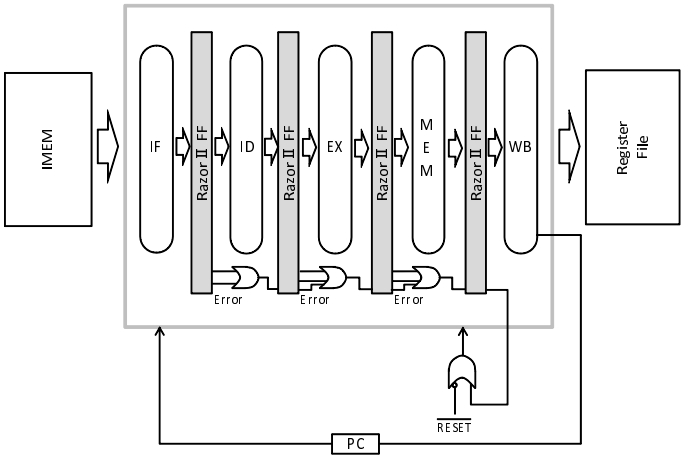

図 3 Error Recovery Architecture using Processor Reset

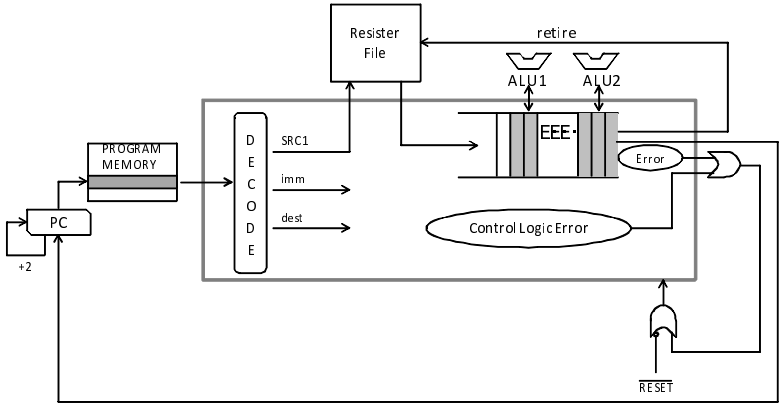

図 4 dual-issue OoO Superscalar Processor Model with Error Tolerance

図 4 に今回作成したモデルの概略図を示す．

2-way の Out-of-Order スーパースカラプロセッサであり，文献[3)4)] を参考にした．2 命令を同時にフェッチし，演算結果の依存関係をみて命令順に関係なく Out-of-Order で演算を実行する．得られた結果は命令順に In-Order でレジスタファイルに書き込まれる．なお

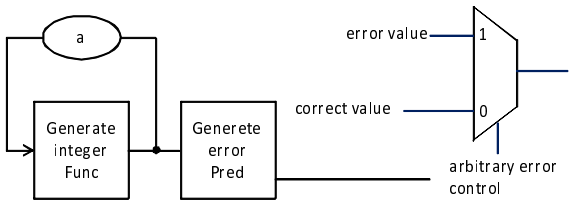

図 5 Error Insertion

Out-of-Order での実行の正しさが検証出来ればよいので，条件分岐等の命令は考慮せず，命令セットはレジスタファイルの値，即値，書き込み先アドレスだけを考えたものとした．

次にエラー発生と回復機構のモデル化について述べる．エラーが発生する箇所は演算結果やリオーダバッファのエントリが正しいかどうかを示す制御ビット等の制御系と，デコードした命令や演算結果などのデータパス系の両方で発生するとした．ただし以下については回復する際に必要となるため，長い書き込み時間または十分な冗長性をもたせてあるとしてエラーは発生しないとしている．

- プログラムカウンタ
- レジスタファイル
- リオーダバッファ中のプログラムカウンタの値

エラー回復機構が正常動作することを確かめるには，モデルにエラーを挿入する必要がある。そこで，エラー発生についても図 5 に示す方法でエラーが発生する可能性のある全ての変数を対象にエラー発生のモデリングをする．

図 5 において a は integer 変数であり，毎ステップ値を更新し，Predicate Function へ入力することで任意のブール値を生成する．エラーは状態遷移の 1 ステップごとに非決定的に発生するとし，実際にエラー値をいれて回復が正常に実行されるかをみるため，エラー発生信号が 1 となったときにエラー値として自由変数を強制的に代入する．自由変数なので正しい値も誤った値もすべての値を考慮出来るため，誤った値が代入されない問題は起こらないと考えられる．

エラー検知の抽象化のため，error control 変数をそのまま用いたエラー検知信号は，パイプラインを伝わりコミットステージにて検知される．しかし，モデルを記述するにあたってエラー検知パスの詳細なモデル化は行わず，同じ命令のデータパス系ついては、検知信号をまとめ，その命令のパイプラインがコミットステージに到達した場合にプロセッサリセッ

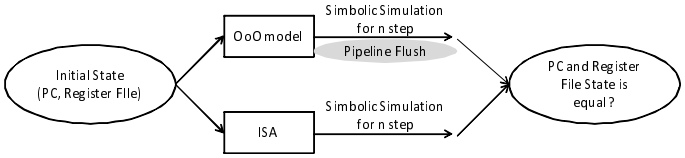

図 6 Flow of Bounded Model Checking

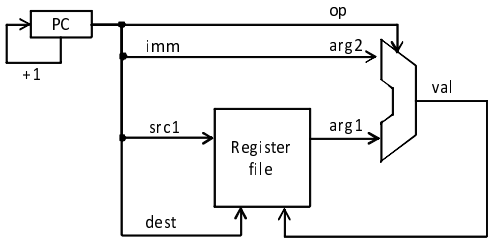

図 7 ISA Model

トとプログラムカウンタのロールバックが行われるように抽象化した．

4.3 Bounded model checking によるプロセッサ検証

プロセッサの検証には bounded model checking を用いた．検証フローを図 6 に示す．

前述のプロセッサアーキテクチャのモデルとは別に命令を In-order で実行しレジスタへ書き込む，図 7 に示した ISA(Instruction Set Architecture) モデルを用意する．初期状態として，OoO モデルと ISA モデルのプログラムカウンタとレジスタファイルを等しい状態にし，任意のステップで記号シミュレーションを行う。ここで Out-of-Order モデルは一度命令をフェッチした後，パイプラインフラッシュにより命令フェッチは禁止され，最初にフェッチされた命令だけが実行される．検証するプロパティとして下記のように最終的なプログラムカウンタの値とレジスタファイルの状態が等価であるかチェックする．

- $\forall regid.[OoO.RFvalid(regid) \Rightarrow (ISA.RF(regid) = OoO.RF(regid))]$
- $OoO.PC = ISA.PC$

5. 実　　験

前述のプロセッサリセットによるエラーリカバリ機構をもつ Out-of-Order スーパースカラプロセッサアーキテクチャのエラー耐性を評価するため，複数モデルを記述し UCLID による検証を行った．実験に際して作成したモデルの way 数，行数，挿入するエラーの回数を**表 1** に示す．

2，4，5-way のプロセッサについて，エラーをそれぞれ 1〜4 回入れ実験を行った．ここでエラー回数は状態遷移の 1 ステップの内にエラーが発生したときに 1 とカウントする．エラーは前述のように一部の状態変数を除き全てにおいて発生する場合としない場合を考慮するため，1 ステップにおいて複数の状態変数でエラーが発生する可能性も扱っている．

また，あまり大きい bound 数を設定するとメモリ不足により検証が中止されたため，bound は 20 とした．

5.1 実 験 環 境

検証には UCLID ver.3.0，SAT:minisat-2.2.0，OS:Linux，CPU:Intel Core-i7 3.0GHz，Memory:8GB のマシンを利用した．

5.2 実 験 結 果

実験結果を**表 2** に示す．

結果よりプロパティを満たすという結果になったものについては，データパス系，制御系のどちらのエラーに対しても回復が正常に行われ，正しい値がレジスタファイルに書き込まれたと考えられる．一方でプロパティが invalid となったものについて，生成された反例を解析した結果，エラーリカバリが何度も行われた結果，指定した命令数の実行が 20 ステップ以内に完了しなかったためであることが分かった．

5-way でエラー発生回数が 4 回のモデルについて，bound を変えて実行したところ，bound

表 1 作成したモデル

	way	#lines	#errors
2way_err1	2	838	1
2way_err2	2	838	2
2way_err3	2	838	3
2way_err4	2	838	4
4way_err1	4	1435	1
4way_err2	4	1435	2
4way_err3	4	1435	3
4way_err4	4	1435	4
5way_err1	5	1741	1
5way_err2	5	1741	2
5way_err3	5	1741	3
5way_err4	5	1741	4
ISAmodel	1	40	0

が 25 のときにメモリ不足で検証が止り，bound が 24 でも 5 命令実行されるはずが 4 命令の実行までで検証が終了した．従ってさらに bound を増やし実行を進めると，リカバリできる可能性がる．今後検討していきたい．実行時間については，反例が見つかった時点で検証はストップし，反例が出力されるため，プロパティが valid となった場合の方が実行時間は長くなる傾向にある．

結果的にそれぞれの状態変数に任意のエラー発生を設定したため，モデルのサイズは大きくなったが自由変数によるエラー値を代入することにより，実際にエラーが起こったときの挙動を検査できた．またエラーの発生回数を制御することで，複数回エラーが発生した場合でも回復機構が正しく動作することを確認できた．

6. ま と め

プロセッサリセットを用いたエラー回復機構をもつ Out-of-Order スーパースカラプロセッサアーキテクチャの検証を行った．エラー回復機構が正しく動作しているかをみるため，それぞれの状態変数にエラーを発生させ，エラー値を代入する評価手法の検討し，エラー発生回数を変えて実験を行った．データパス系，制御系それぞれの値で個々にエラーを入力したため，反例からどの部分でエラーが発生し，プロパティを満たさなかったのか確認することが可能となった．

今後は，例外処理や条件分岐等の命令をもつ，より複雑なプロセッサアーキテクチャについても対応できるよう，手法の改良を行う予定である．

表 2 実験結果

	#vers	#truth	#term	CNF-var	CNF-cls	CNF-lit	Property	time(sec)
2way_err1	720	4786	1965	73665	1706907	3738339	valid	52
2way_err2	720	4786	1965	573630	1706802	3738094	valid	68
2way_err3	720	4786	1965	573634	1706814	3738122	valid	112
2way_err4	720	4786	1966	573576	1706640	3737716	invalid	46
4way_err1	1354	12425	4583	2013167	6010800	13444262	valid	593
4way_err2	1354	12425	4583	2013148	6010743	13444129	valid	1557
4way_err3	1354	12425	4584	2013152	6010755	13444157	invalid	226
4way_err4	1354	12425	4583	2013092	6010575	13443737	invalid	236
5way_err1	1670	17718	6307	3001698	8969727	20130083	valid	1700
5way_err2	1670	17718	6307	3001679	8969670	20129950	invalid	939
5way_err3	1670	17718	6307	3001679	8969670	20129950	invalid	916
5way_err4	1670	17718	6308	3001614	8969475	20129495	invalid	408

参 考 文 献

1) Tom Schubert. “High Level Formal Verification of Next-Generation Microprocessors”. *in Proc. Design Atomation Conference(DAC 2003)*, pp. 1-6, 2003.
2) Randel E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia. “Modeling and Verifying Systems using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions”. *in Computer-Aided Verification(CAV 2002)*, LNCS 2404, pp. 106-122, 2002.
3) Shuvendu K. Lahiri, Sanjit A. Seshia, and Randal E. Bryant. “Modeling and Verification of Out-of-Order Microprocessors in UCLID”. *in Formal Methods in Computer-Aided Design(FMCAD 2002)*, LNCS 2517, pp. 142-159, 2002.
4) Bijan Alizadeh, Amir Masoud Gharehbaghi, and Masahiro Fujita. “Pipelined Microprocessors Optimization and Debugging”. *in International Symposium on Applied Reconfigurable Computing(ARC 2010)*, LNCS 5992, pp. 435-444, 2010.
5) Miroslav N. Velev and Ping Gao. “Method for Formal Verification of Soft-Error Tolerance Mechanisms in Pipelined Microprocessors”. *in International Conference on Formal Engineering Methods(ICFEM 2010)*, LNCS 6447, pp. 355-370, 2010.
6) 杉本 健, 入江 英嗣, 五島 正裕, 坂井 修一. “タイミング・エラー耐性を持つスーパースカラ・プロセッサ”. 電子情報通信学会研究報告 CPSY2008-14, pp. 19-24, 2008
7) Bryan A. Brady, Sanjit A. Seshia, Shuvendu K. Lahiri, Randal E Bryant. “A User’s Guide to UCLID Version 3.0”. Oct. 2008.
8) S. Das, C. Tokunaga, S. Pant, W-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. “RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance”. *in IEEE Journal of Solid-State Circuits*, Vol. 44, No. 1, pp. 32-48, 2009.