Vol.2021-SLDM-195 No.9
2021/10/15

近似コンピューティング回路の設計最適化に向けた計算重要度評価技術

陸 佳萱[1]　増田 豊[1]　石原 亨[1]

概要：ポストムーアコンピューティング技術の一つとして，近似コンピューティング (Approximate Computing: AC) に注目が集まっている．AC は重要な計算を正確に，他の計算を近似的に実行することにより，集積回路の省エネルギー化，小面積化，高性能化を推進する技術である．AC 回路を実用化するためには，どの計算がどれ程重要か慎重に決定し，論理最適化およびタイミング最適化に還元する必要がある．本研究は，「計算重要度」を「計算時の故障が実行結果に与える誤差の平均値」と定義し，Flip-Flop (FF) ごとに重要度を評価する技術を提案する．また，重要度の評価時間を効率的に削減するために，複数の FF を活性化情報に基づきグループ化し，グループ毎に重要度を評価する手法を提案する．

1. 序論

ムーアの法則を指針とした微細化の継続は，物理的制約と開発コストの観点から限界が訪れつつある．製造プロセスの微細化に頼らずに，集積回路の省エネルギー化，小面積化，高性能化を推進する新しいパラダイムとして，近年，近似コンピューティング (AC: Approximate Computing) [1], [2] に注目が集まっている．従来設計は全ての計算を正確に実行する方針に基づくが，AC ではこの方針を緩和して，「重要な計算を正確に、他の計算を近似的に」行う．近似処理をソフトウェアとハードウェアの両面に取り入れることによって，計算効率を大幅に向上できるため，機械学習，IoT，画像・音声・信号処理などの様々な分野に応用できると期待されている．

AC 回路の設計最適化には，設計，性能評価，検証まで一貫した設計開発支援 (CAD: Computer Aided Design) 技術が不可欠である．ここで，従来の CAD 技術は全てのパスが機能的，タイミング的に完全に正しく動作することを前提とする．他方，AC 回路では，計算結果の品質 (例. 画質) を満足する範囲で，機能的故障とタイミング故障の発生を許容し得る．すなわち，AC 回路は従来回路と異なり一定量の計算誤差を許容できる特性を持つため，従来の CAD 技術をそのまま適用することは容易ではない．

このような背景から，AC 回路向けの CAD 技術が盛んに研究されている．設計手法としては，プログラミング言語レベル [3] からハードウェアレベル [4], [5] に渡って，数多くの手法が提案されている．一方，性能評価技術と検証技術としては，様々な形式的手法 [6], [7], [8] が提案されているものの，有効性は演算器を対象とした評価実験によってのみ示されており，多様な設計技術に適用出来る水準には到達していない．換言すれば，設計後 AC 回路の性能評価と検証手法は未だ十分に確立されておらず，新たな CAD 技術の開発が強く求められている．

本研究では，AC 回路の「計算重要度」を Flip-Flop (FF) 単位で考慮可能な評価技術を提案する．計算重要度を設計最適化に還元することにより，計算結果の品質を担保しつつ，省エネルギー化，小面積化，高性能化を推進する狙いがある．**図 1** を用いて，計算重要度を説明する．本稿では，計算重要度を，計算時の故障が実行結果に与える誤差の平均値とする．図 1(a) は正常動作の様子，図 1(b) は実行結果の上流に位置する FF において故障が発生した際の動作をそれぞれ表す．図 1(a) より，正常動作における実行結果は 4'b0000 である．一方，図 1(b) より，故障時の結果はそれぞれ 4'b0010，4'b0100，4'b0000 である．従って，この FF の計算重要度は，$\frac{2+4+0}{3} = 2$ と導出される．計算重要度が高いほど，故障時に計算結果の品質を大きく

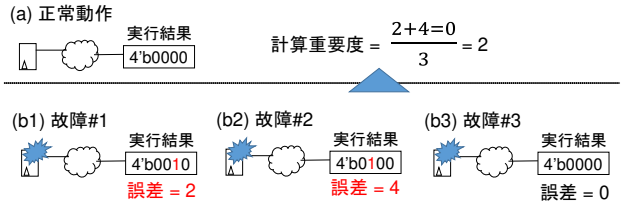

図 1: Flip-Flop (FF) の計算重要度: 計算時の故障が実行結果に与える誤差の平均値

[1] 名古屋大学大学院情報学研究科

低減するため，論理故障と遅延故障の発生をより注意深く防ぐ必要がある．

本研究は，「計算重要度」を「計算時の故障が実行結果に与える誤差の平均値」と定義し，Flip-Flop (FF) ごとに重要度を評価する技術を提案する．さらに，複数の FF を活性化情報に基づきグループ化し，グループ毎に重要度を評価する手法を提案する．グループ化により，重要度の過小評価を防ぎつつ，評価時間を削減する狙いがある．

本研究の主な貢献は，FF ごとに重要度を評価する技術の提案と，FF のグループ化に基づくスケーラビリティの向上手法にある．提案手法は，故障挿入 (Fault Injection; FI) に着目し，FI シミュレーションを用いて FF の「計算重要度」を評価する．FI シミュレーション回数，すなわち重要度の評価時間を削減するために，同時に活性化する FF 組を一つのグループに集約し，グループ単位で故障を挿入する FI シミュレーションを実施する．提案手法に基づき評価された重要度は，AC 回路の設計最適化に還元され，省エネルギー化，小面積化，高性能化のさらなる推進に貢献すると期待される．積和演算器を対象に実施した，予備実験結果をあわせて紹介する．

本稿の以降の構成は以下の通りである．2 章では，FI シミュレーションについて説明する．3 章では，FF の計算重要度評価に向けた，FI シミュレーション手法を提案する．4 章では，提案手法を用いた予備実験について説明する．最後に，5 章で結論と今後の展望を述べる．

2. 故障挿入シミュレーション

故障挿入 (FI) は，回路内の一部に恣意的に誤り情報を混入する操作を指す．挿入した誤り情報が異常動作 (例. 汎用プロセッサにおけるクラッシュ動作) を誘発するかどうかをシミュレーション時に観測することにより，故障に脆弱な回路機構を探索できる．FI シミュレーションでは，想定する故障をモデル化し，その故障モデルに則した挿入時刻 (故障発生時刻) と挿入箇所 (故障発生個所) を決定することが肝要である．この観点から，回路内でランダムに故障発生する仮説に基づき，ソフトエラーを対象とした FI 手法が数多く研究されてきた．一方，近年では，遅延故障などの異なるモデルに対する FI の適用についても注目が集まっている [9]．本稿では，遅延故障を対象とした FI に焦点を絞る．

FI シミュレーションの実装法は，ソフトウェア，命令セット，レジスタ転送レベル (Register Transfer Level; RTL)，論理ゲートレベル，に大別される．ソフトウェアレベルの FI では，C++ プログラムなどのプログラム内の変数値を反転させる方法であり，シミュレーション速度が高速である利点を持つ．命令セットレベルの FI は，命令実行時のオペランドや命令コードに故障を挿入する手法である．RTL レベルおよび論理ゲートレベルの FI は，レジスタや論理ゲートに論理値の反転を挿入する手法であり，ハードウェアのアーキテクチャを考慮したシミュレーションを提供できる点に優位性が存在する．

これまでに，上記の実装法と多様な技術を組み合わせた FI 手法が数多く研究されている．例えば，文献 [10] では，Graphic Processing Unit (GPU) の信頼性を評価するために，ソフトウェアレベルと RTL の FI を組み合わせる手法を提案している．また，[9] では，遅延故障が AC 回路に及ぼす影響を評価するために，統計的タイミング解析 (Static Timing Analysis; STA) と命令セットレベルの FI を用いた手法を提案している．本研究では，FF の計算重要度を AC 回路の設計最適化に還元することを見据えて，RTL シミュレーションを利用した FI 手法を構想する．次章で詳細について説明する．

3. 提案構想: 計算重要度の評価技術

本章では，AC 回路の設計最適化に向けた計算重要度評価技術を提案する．提案手法は，FI を用いて FF の「計算重要度」を評価する．FI シミュレーション回数を削減するために，複数の FF を活性化情報に基づきグループ化し，グループ毎に FI シミュレーションを実行する．まず，3.1 節では，提案構想の全体像を紹介する．次に，3.2 節では，グループ化について述べる．

3.1 提案構想の全体像

図 2 に，提案構想の全体像を示す．提案手法は，評価対象の回路 (DUT: Design Under Test) とテストベンチを入力として受け取る．次に，DUT とテストベンチを用いて RTL シミュレーションを実行し，活性化情報を VCD (Value Change Dump) ファイルとして取得する．活性化情報を元に，活性化 FF 群をグループ化し (3.2 節)，FI 箇所

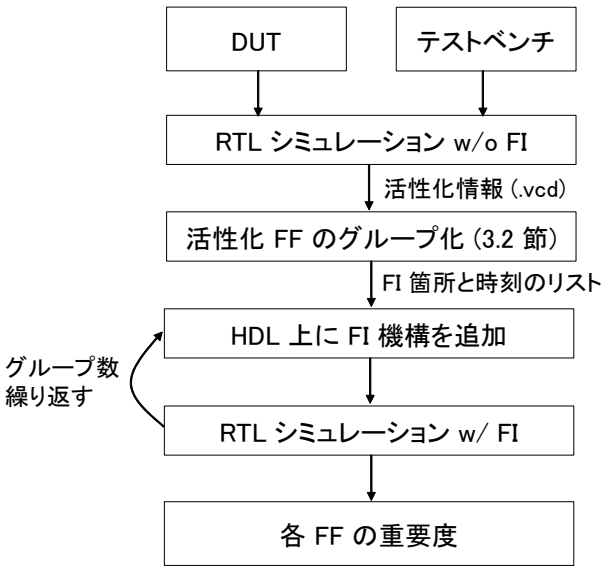

図 2: 提案構想の全体像

と時刻のリストとして整理する．その後，各グループに対して，ハードウェア記述言語 (HDL: Hardware Description Language) 上に FI 機構を追加し，RTL シミュレーションを実行する．実行結果を観測することで，故障を挿入した FF 群の重要度を評価する．

3.2 活性化 FF のグループ化

提案構想は，RTL シミュレーション結果から抽出した活性化 FF 群に基づき，FI 対象 FF をグループ化する．本稿では遅延故障を対象とし，活性化しない FF については，遅延故障確率は 0 と想定して重要度の算出対象から除外する．次に，活性化 FF 群をグループ化する動機について，**図 3** を用いて説明する．N_{FF} 個の活性化 FF を N_G 個のグループに分類すると仮定する．また，各活性化 FF はいずれか 1 つのグループのみに属するものとする．グループ毎と FF 毎の FI において，重要度算出時間を比較すると，$N_G \leq N_{FF}$ より，グループ毎の FI 手法が短時間で重要度を算出できると期待される．

他方，グループ毎の手法では複数の故障をまとめて挿入しているため，一見すると，FF 毎の手法と比べて重要度

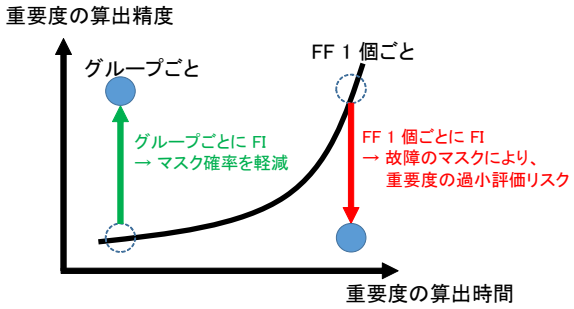

図 3: グループ毎の FI : マスク確率の削減に期待

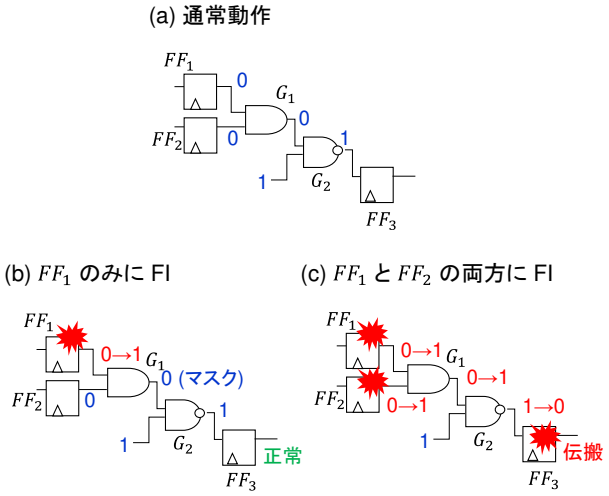

図 4: 故障のマスク例: (a) 通常動作，(b)FI 後マスク，(c)FI 後，後段の FF に故障伝搬

の算出精度が低いように思われる．しかし，FF 毎の FI では，挿入された故障がマスクされ，重要度の過小評価に繋がるリスクが存在する．**図 4** にマスクの例を示す．図 4 は，3 つの FF (FF_1，FF_2，FF_3)，2 つの論理ゲート (G_1，G_2) から構成される回路の例である．図 4(a) は FI 前の回路，図 4(b) は FF_1 にのみ FI して故障がマスクされた動作，図 4(c) は FF_1 と FF_2 の両方に FI して故障が FF_3 に伝搬する様子をそれぞれ示す．一般的に，多くの故障を挿入するほど，故障がマスクされる確率は低減する．この観点から，FF 毎のグループ化は，重要度の算出時間を削減しつつ，マスクの見逃しを防止することにより算出精度を向上できる余地がある．以上の考えに基づき，提案構想では，FF のグループ化に着目する．グループ化の方法としては，集合分割問題 (コスト関数を定義することにより，0-1 整数線形計画問題に帰着可能)，貪欲法，クラスタリングなどの多様な方法が考えられる．適切なグループ化方法の選定は，今後の課題の一つである．

4. 予備実験

本章では，3 章で構想したグループ化に向けて，同時に活性化する FF 組の抽出を予備実験として行う．本実験では，対象回路として，8 ビット 2 入力の積和演算器を選択した．積和演算器は Verilog HDL で実装した．上記の回路に対して，RTL シミュレーションを実行して活性化 FF を抽出し (図 2)，各 FF 組に対して，同時に活性化したクロックサイクル数を調査した．本実験では，このクロックサイクル数を同時活性化回数と呼ぶ．ワークロードとして，ランダムテスト (100 クロックサイクル) を実行した．本予備実験では，CentOS release 6.10 と 18-Core Processor を搭載した計算機マシン，および ModelSim version 10.5b を用いた．

同時活性化回数の評価結果を**図 5** に示す．図 5 より，FF 組によって，同時に活性化する回数に大きな差が存在することが読み取れる．また，同じクロックサイクルには同時に活性化しない FF 組も多数観測した．これらの FF

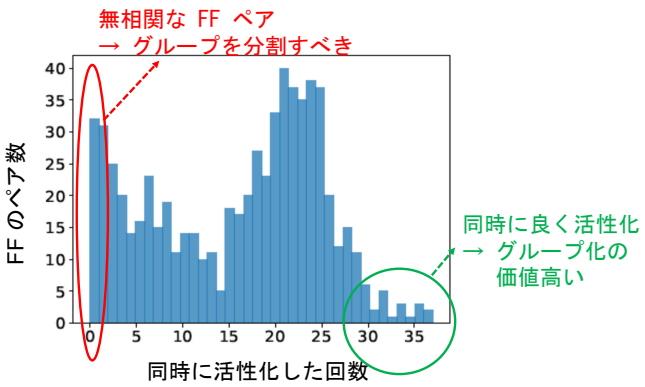

図 5: FF の同時活性化回数のヒストグラム: FF 組によって，同時活性化回数は大きく異なる．

組については，最適化関数にペナルティ関数を設けるなど，同じ集合に振り分けない方策が必要であると考えられる．FF の同時活性化回数については，入力するワークロードと回路構造に大きく依存することが予想される．異なる回路とワークロードを用いた評価実験は，今後の課題の一つである．

5. 結論と今後の課題

本研究は，AC 回路の設計最適化に向けた計算重要度評価技術を提案した．提案手法の肝は，複数の Flip-Flop (FF) を活性化情報に基づきグループ化し，グループ毎に故障挿入シミュレーションを実行する点にある．グループ化により，重要度の過小評価を防ぎつつシミュレーション回数を削減する狙いがある．

積和演算器を対象に，活性化 FF 組を抽出し，FI 機構の動作を確認する予備実験を行った．評価結果より，FF 組によって， 同時活性化回数は大きくばらつきことを実験的に確認した．

今後の課題は，異なる回路とワークロードを用いた評価実験，同時活性化回数の分布を元に FI 箇所と時刻を決定するアルゴリズムの構築，シミュレータでの重要度算出および妥当性検証，タイミングクリティカルな FF を対象としたグループ化法の検討である．

謝辞 本研究の一部は JSPS 科研費 (20K19767) の助成を受けたものである．

参考文献

[1] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” *Proc. ETS*, pp. 1-6, 2013.
[2] Q. Xu, T. Mytkowicz, and N. S. Kim, “Approximate computing: A survey,” *IEEE Design & Test*, vol. 33, no. 1, pp. 8-22, 2016.
[3] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Architecture support for disciplined approximate programming,” *Proc. ASPLOS*, pp. 301-312, 2012.
[4] R. Hegde and N. R. Shanbhag, “Soft digital signal processing,” *IEEE TVLSI*, vol. 9, no. 6, pp. 813-823, 2001.
[5] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, “Slack redistribution for graceful degradation under voltage overscaling,” *Proc. ASP-DAC*, pp. 825-831, 2010.
[6] S. Froehlich, D. Große, and R. Drechsler, “One method–all error-metrics: A three-stage approach for error-metric evaluation in approximate computing,” *Proc. DATE*, pp. 284–287, 2019.
[7] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, “Precise error determination of approximated components in sequential circuits with model checking,” *Proc. DAC*, pp. 1–6, 2016.
[8] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, “MACACO: Modeling and analysis of circuits for approximate computing,” *Proc. ICCAD*, pp. 667–673, 2011.
[9] J. Constantin, A. Burg, Z. Wang, A. Chattopadhyay, and G. Karakonstantis, “Statistical fault injection for impact-evaluation of timing errors on application performance,” *Proc. DAC*, pp. 1-6, 2016.
[10] F. F. d. Santos, J. E. R. Condia, L. Carro, M. S. Reorda, and P. Rech, “Revealing GPUs vulnerabilities by combining register-transfer and software-level fault injection,” *Proc. DSN*, pp. 292-304, 2021.