Vol.2021-SLDM-195 No.5
2021/10/15

近似コンピューティング回路の品質検証を高速化するファジングテスト法

本多 佑成[1]　増田 豊[2]　石原 亨[2]

概要：近似コンピューティング (Approximate Computing; AC) は，計算品質の制約 (例. 機械学習の推論精度) 下で，消費電力や回路面積を削減できる技術として有望視されている．一方，AC 回路では，重要でない計算実行時において，論理故障やタイミング故障の発生を許容し得るため，従来回路と比較して「計算品質が制約を満たしているかどうか検証すること」（以下，品質検証）が困難である．本稿は，「計算品質の制約を違反させるテストパターンをできるだけ多く発見すること」を品質検証における網羅率最大化と定義し，AC 回路を対象とした，品質検証の網羅率最大化手法を研究する．近年，ファジングを用いた回路検証に注目が集まっている．ファジングは，ソフトウェアセキュリティの研究領域で最も有望なテスト手法の一つであり，コード網羅性を高めるテストパターンを軽量かつ高速に生成出来ることに大きな利点がある．本研究ではまず，ファジングを利用した既存の品質検証手法を対象に，(回路規模，検証時間，検証網羅性) 組の関係を定量的に評価する．次に，大規模集積回路への適用を見据えて，ファジングを用いた品質検証における高速な網羅率最大化法を検討する．

1. はじめに

近年，集積システムの省電力化と高性能化を推進可能な設計パラダイムとして，近似コンピューティング (Approximate Computing; AC) に注目が集まっている [1], [2], [3]. AC は，「重要な計算を正確に実行し，他の演算を近似的に実行する」設計指針に基づくものであり，機械学習，ディジタル信号処理，エッジコンピューティング，IoT などの多様な分野において高く期待されている．

AC 回路は，設計後，信頼性を保証するために検証される．検証では，多様なテストパターン (例. ニューラルネットにおけるデータセット) を与えて，期待した結果を出力するか確認する．ここで，AC 回路では，従来回路と異なり，計算結果の品質を満足する範囲であれば，論理故障やタイミング故障の発生を許容する．この観点から，AC 回路の検証では，「計算品質の制約を脅かすテストパターン」を発見する必要がある．本稿では，この検証を品質検証と呼称する．

AC 回路の検証技術は，静的な手法と動的な手法に分類される．静的な検証技術として，形式的検証を利用した手法 [4], [5], [6] が提案されている．これらの手法では，計算品質の制約を違反するテストパターンが存在し得るかどうかを求解する．形式的手法は，網羅的な探索を行うことから，信頼性保証の観点で極めて優れている．一方，複雑な計算品質の制約を考慮することが困難であり，現実的な計算時間での求解に失敗するリスクも存在する．

他方, 動的な技術として, ファジングを用いた手法 [7] が提案されている．ファジングは, ソフトウェアのセキュリティテストにおいて多くの実績を持つ手法であり [8], [9], [10], 未知のバグに対する高い発見能力が数多く報告されている．このようなファジングの探索能力に着想を得て，近年では，集積回路の検証 [7], [11] とセキュリティ評価 [12] への応用法に対する研究が開拓されている．[7] は，Coverage-based Greybox Fuzzing (CGF) と Design Under Test (DUT) 機構を組み合わせた検証手法を提案し，AC 回路の品質検証を 3 倍高速化する成果を示している．本研究では，[7] の手法に着眼し，AC 回路の品質検証における網羅率最大化手法に焦点を絞る．本稿では，「計算品質の制約を違反させるテストパターンをできるだけ多く発見すること」を品質検証における網羅率最大化と呼称する．

検証対象回路の探索空間は，入力のビット幅や状態遷移数など，様々な要素に依存する．しかし，[7] では，3 ビットの近似積和演算器と近似加算器のみを検証対象に有効性を示しており，より大型の演算器に対する有効性は示していない．ファジングの有効性を明らかにするためには，異なる規模の検証対象回路に対して，どのような検証時間と

[1] 名古屋大学情報学部
[2] 名古屋大学大学院情報学研究科

検証網羅性を達成するか定量的に議論することが必要不可欠である．本稿ではまず，[7] の手法を用いて，多様なビット幅の演算器を対象に品質検証を行い，検証に要する計算時間と得られた網羅性の関係を定量的に議論する．次に，大規模な AC 回路の品質検証に対してファジングを適用することを見据えて，品質検証の網羅率を素早く高めるファジングテスト手法を提案する．

本研究の主な貢献は，既存のファジングテスト法 [7] の検証時間と網羅性の定量的な評価，および，ファジングテストの高速化法の提案にある．提案手法の肝は，DUT 機構の効率化，および，階層的テストの活用による探索空間の削減にある．

本稿の以降の構成は以下の通りである．2 章では，関連研究としてファジングと AC 回路の品質検証手法について説明する．3 章では，[7] の手法を用いた予備実験を実施し，検証時間と網羅性の関係を定量的に評価する．4 章では，大規模な AC 回路の品質検証における網羅率最大化に向けた，高速化手法を提案する．最後に，5 章で結論と今後の展望を述べる．

2. 関連研究

本章では，関連研究としてファジングと AC 回路の品質検証手法について説明する．まず，2.1 節で，ファジングについて述べる．次に，2.2 節で，ファジングの一種である CGF を AC 回路の品質検証に応用する手法を紹介する．

2.1 ファジング

ファジングとは，テストパターンの変異と実行を繰り返し行う手法を指す．ファジングの手法は，ブラックボックス，グレイボックス，ホワイトボックスの三種類に大別される．ブラックボックス・ファジングはランダムテストの亜種であり，テストパターンの生成が容易である利点を持つ．ホワイトボックス・ファジングは，既知のソースコードに対して解析を行い，探索を効率的に行うテストパターンを生成する手法である．グレイボックス・ファジングは，上述した二つのテスト法の中間的な存在であり，両者をハイブリッドに組み合わせることで，短時間で効率的なテストパターンを生成することを狙った手法である．グレイボックス・ファジングの中で，特に有望な手法の一つとして，Coverage-based Greybox Fuzzing (CGF) が盛んに研究されている [8], [13], [14].

図 1 を用いて，CGF の動作について説明する．CGF では，条件分岐に着目し，「テストパターンの変異，プログラム実行 (Program Under Test; PUT)，テスト網羅性の集計とテストパターンへのフィードバック」を繰り返し行うことで，テスト対象空間を自動探索する．PUT 時に新たな実行経路パスが発見された際には，その入力パターンをもとに次の変異を生成し，利用する．変異方法は，ビット反転，四則演算，トリミング，ランダム入力など，軽量かつ多様な処理で構成されている．CGF は，局所的な探索と大域的な探索を高速かつ自動的に実施できることから，ソフトウェア・システムの品質テストにおいて多くの実績を挙げており，近年ではハードウェアの研究領域においても注目が集まっている [7], [11], [12].

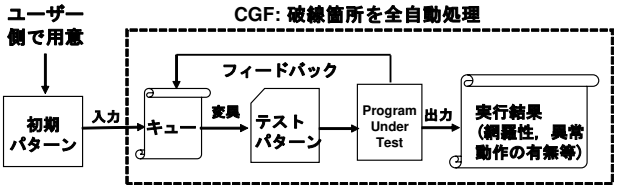

図 1: CGF (Coverage-based Grey-box Fuzzing)

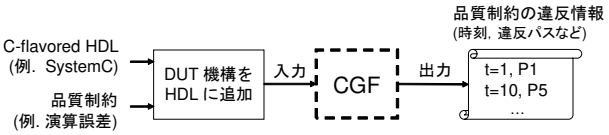

図 2: CGF を用いた品質検証手法 [7]

2.2 CGF を用いた AC 回路の品質検証手法

本節では，CGF を用いた AC 回路の品質検証手法における関連研究として，[7] の手法を紹介する．[7] で提案された品質検証手法の概要を**図 2** に示す．この手法ではまず，検証対象である AC 回路に対して DUT (Design Under Test) 機構を追加する．ここで，CGF が分岐網羅性を指標とする点に着目し，ハードウェアのデータパスの分岐を考慮した DUT 機構を提案した．一部を抜粋した結果を Listing 1 に示す．[7] では，近似回路と正確な回路をインスタンス化し，両者に CGF から受け取った入力を与え，両回路の計算結果を元に近似回路の計算誤差を算出し，品質制約を違反しているか診断する．近似回路への入力データパターンを羅列することにより (8 行目以降)，CGF は，品質制約を違反するテストパターンを発見した際に，そのパターンを次の変異に取り入れることができる．

Listing 1: AC 回路に対する DUT 機構挿入 [7]

```
1  int sc_main (int argc, char *argv[]) {
2      ** CGF のテストパターンを受け取る機構**
3      ** 近似加算器と正確な加算器をインスタンス化**
4
5      // 計算誤差の診断
6      if ( (AC_Y - EX_Y) > ERROR_TH){
7          // 制約を違反するパターンの特定
8          if (input == XX){}
9          else if ...
10     }
11
12     return 0;
13 }
```

以上の仕組みに基づき，[7] は，ランダムテストと比較して，90% の検証網羅性を $\frac{1}{3}$ の検証時間で達成する成果を示している．しかし，[7] では，検証対象回路は 3 ビットの近似積和演算器と近似加算器のみであり，異なる規模の回路に対する有効性は示されていない．本研究では，多様なビット幅の演算器を対象に品質検証を行う．次章では，[7] の手法を用いて，検証に要する計算時間と得られた網羅性の関係を定量的に議論する．

3. 予備実験

本章では，[7] を用いて，二つの予備実験を行う．第一の実験では，DUT 機構の搭載に要する CPU 時間を調査する．第二の実験では，CGF の検証時間と検証網羅性 (計算品質を違反するテストパターンに対する発見網羅性) の関係を評価する．3.1 節では各実験の評価環境を示し，3.2 節において実験結果を示す．

3.1 評価環境

本実験では，対象回路として，文献 [15] で提案されている近似加算器 (図 3) を選択した．本近似加算器のビット幅については，第一の実験では，4 ビットから 16 ビットまで 2 ビット刻みで 7 通り，第二の実験では，4 ビット，6 ビット，8 ビットの 3 通りを用意した．正確な加算結果 Y_{ex} と近似加算結果 Y_{ac} を元に，以下の式の通りに演算誤差 E を算出し，計算品質として，$E = 0.1$ を設定した．

$$E = \frac{|Y_{ex} - Y_{ac}|}{Y_{ex}} \quad (1)$$

以上の (ビット幅，計算品質の制約) 組のそれぞれに対して，DUT 機構を挿入後，ファジングテストを実行し，検証網羅性を実験的に評価した．ここで，検証網羅性は，$E > 0.1$ を満たすテストパターン組の中から，発見できた割合を指す．評価実験では，Ubuntu 16.04.7 LTS と AMD Ryzen Threadripper 3990X 64-Core Processor を搭載した計算機を使用し，ファジングツールとして，AFL2.52b [8] を利用した．また，各ファジングテストの最大実行時間は 5 分とした．

3.2 評価結果

図 4 に，近似加算器のビット数と DUT 機構の挿入時間の関係を示す．図 4 より，ビット数を増加するにつれて，DUT 機構の挿入時間が劇的に増加していることが読

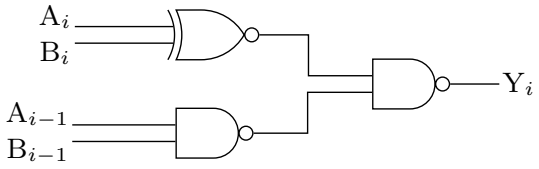

図 3: 評価対象回路の近似加算器 [15]

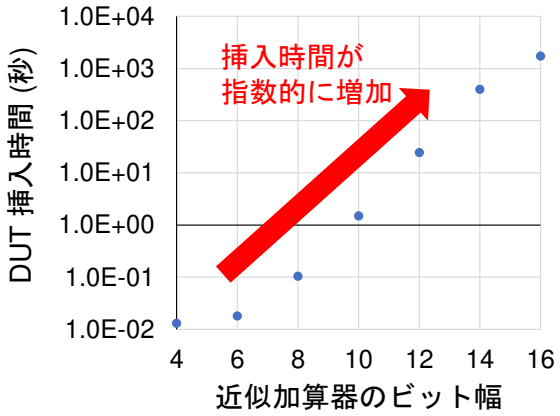

図 4: DUT 機構の挿入に要する計算時間: ビット幅に対して，挿入時間が指数的に増加

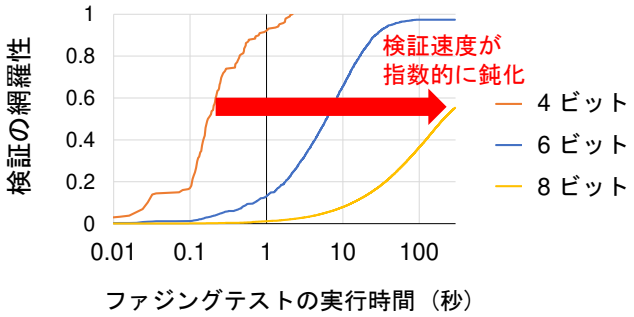

図 5: 検証速度とテスト網羅性:ビット幅に対して，検証速度が指数的に鈍化

み取れる．例えば，4 ビットの加算器では，$2^{2\times4+1} = 2^9$ 通りの条件分岐を挿入し，0.013 秒を要している．一方，16 ビットの加算器では，$2^{2\times16+1} = 2^{33}$ 通りの分岐挿入を行い，1710.13 秒を消費している．ビット幅の増加に応じて，データパス数が指数的に増加するため，DUT 機構 のコードオーバヘッド及び追加時間が劇的に増加していると考えられる．この観点から，大規模な集積回路に対して，全データパスに対応するテストパターン (Listing 1 の 8 行目以降) を網羅的に列挙することは現実的ではなく，DUT 機構のオーバヘッドを削減することは，極めて重要である．4.1 節において，コードオーバヘッドと検証時間の削減に向けた，DUT 機構のスパース化手法を提案する．

図 5 に，CGF の実行時間と検証網羅性の関係を示す．図 5 より，ビット幅が大きくなるにつれて，検証網羅性の増加速度が急激に鈍化する傾向が読み取れる．例えば，CGF 実行開始後 100 秒に注目すると，入力ビットが 6 ビットの際は網羅性は 0.95 であるのに対して，8 ビットでは 0.35 である．以上より，[7] の手法を用いた検証速度が，回路規模の増加に応じて劇的に低下することを実験的に確認した．4.2 節では，検証速度の鈍化を緩和し，より大規模な AC 回路の検証を行うために，階層的テスト手法を提案する．

以上の予備実験により，ビット幅の増大に応じて，DUT

機構の挿入時間が指数的に増加すること，CGF が検証網羅性を向上する速度が劇的に鈍化することを実験的に確認した．従って，より大規模な集積回路に対しては，[7] の手法をそのまま適用することは困難であり，DUT 機構のオーバヘッドを削減しつつ，CGF の速度を向上する方策が不可欠であると考えられる．次章では，これらの課題の解決に向けて，ファジングテストの高速化手法を提案する．

4. 提案手法

本章では，AC 回路の品質検証における網羅率最大化を目的とした，ファジングテストの高速化手法を提案する．提案手法の肝は，DUT 機構の効率化に基づくファジング速度向上，および，階層的テストの活用による探索空間の削減にある．各手法の概要を**図 6** と **図 7** に示す．本章の以降で，それぞれについて詳述する．

4.1 DUT 機構の効率化

図 4 で示した通り，[7] において，DUT 機構のコードオーバヘッドは回路規模に対して指数的に増加する．従って，データパスの全ての組み合わせを DUT 機構により追加することは，コードサイズの過度な増加に繋がる．これは，プログラム実行時間を増大し，検証速度そのものを大きく低下させるリスクを併せ持つ．一方，CGF ではコード網羅性，特に，if-else 文などの分岐網羅性を考慮してテストパターンを変異する．すなわち，DUT 機構により挿入されるデータパスの分岐記述は，CGF の変異元に利用される．この観点から，DUT 機構の分岐記述量を過度に削減すると，CGF の変異回数が大きく低下するため，探索効率が劣化する恐れがある．以上より，DUT 機構の実装では，コードオーバヘッドを抑えつつ探索効率を高めるように，記述するデータパスの分岐を選定することが肝要である．

また，2.1 節で前述した通り，ファジングは大域的な探索 (例. ランダム変異) と局所的な探索 (例. 数値加算) を併用する．ここで，一度到達した箇所の近傍箇所は，局所的な探索により発見される可能性が高いと考えられる．この点に着眼し，本研究では，「DUT 機構におけるデータパスの分岐記述量をどのようにスパース化するか」という最適化問題を，テストパターン間の局所性を考慮した検証探索空間の被覆問題として解くことを提案する (図 6(右))．本問題を正確に解くためには，検証対象空間の局所性を適切に考える必要がある．そこで，「CGF から入力される文字列を，近似回路の入力論理値にどのようにマッピングするか」という問題についても合わせて研究する (図 6(左))．マッピングを行う写像関数を改良することで，CGF の変異による局所探索の範囲を制御し，DUT 機構の正確なスパース化に応用する狙いがある．

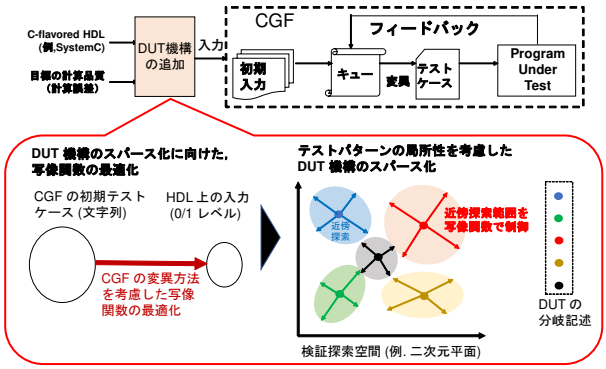

図 6: DUT 機構の効率化

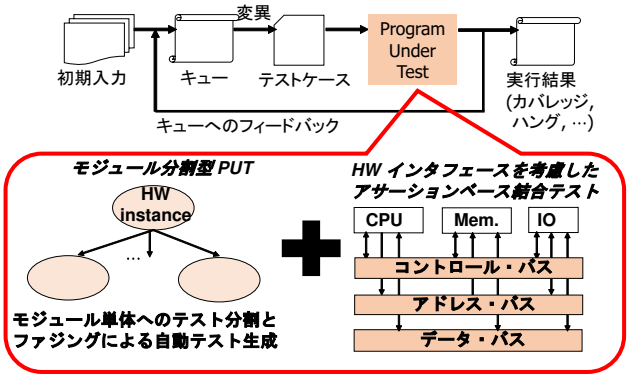

図 7: 階層的テストを用いた探索空間の削減

4.2 ユニットテスト

4.1 節で提案した手法は，比較的小規模な単一モジュールに対しては効果的であると考えられるが，大規模回路への適用を見据えると，「回路規模に対する，テスト工数の増加速度」を鈍化させるアルゴリズムの導入は，依然として必須である．ここで，ソフトウェアテストの研究領域に目を向けると，テスト対象を関数やライブラリなどに細かく分離する単体テストとテスト対象の連結によりテスト網羅性を補強する結合テストが強力な手法として知られている．

そこで本研究では，単体テストと結合テストの階層性に着目し，ハードウェアのモジュール単位に検証対象を分割したモジュール分割型テスト法，バスインタフェースを経由した通信動作を，アサーションチェック機構を利用して確認する結合テスト法を提案する．提案手法により，テスト対象の回路規模を，ハードウェア全体から，規模最大のモジュールにまで削減し，検証時間を大幅に削減する狙いがある．モジュール単体の検証については，4.1 節のスパース化，重点的サンプリングやコンコリックテストを活用したハイブリッドファジングの導入により，検証精度を高めつつ検証時間を高速化できると期待される．以上の技術を統合して， AC 回路に対して，現実的な時間で，精度良く，高網羅率な品質検証を遂行可能なファジングテスト法の実現を目指す．

5. 結論と今後の課題

本研究では，既存の AC 回路向けファジングテスト法を用いて，検証時間と検証網羅性を定量的に評価した．予備実験により，回路の大規模化につれ，DUT 機構の挿入に要する時間が指数的に増加すること，テスト網羅性の増加速度が大幅に鈍化することを実験的に確認した．従って大規模な AC 回路にファジングテストを適用するためには，DUT 機構の工夫が不可欠である．

また，大規模集積回路への適用を見据えて，ファジングテストの高速化法を提案した．提案手法の肝は，DUT 機構の効率化と階層的テストの活用にある．今後の課題は，DUT 機構に記述するデータパスの分岐情報の最適化，階層的テスト法の考案，重点的サンプリングやコンコリックテストの活用による更なる高速化法の検討である．

謝辞 本研究の一部は JSPS 科研費 (20K19767) および JST，さきがけ，JPMJPR20M9 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] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, “A survey of testing techniques for approximate integrated circuits,” *Proc. IEEE*, pp. 1-17, 2020.
[4] 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.
[5] 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.
[6] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, “MACACO: Modeling and analysis of circuits for approximate computing,” *Proc. ICCAD*, pp. 667–673, 2011.
[7] K. Yoshisue, Y. Masuda and T. Ishihara, “Dynamic verification of approximate computing circuits using coverage-based grey-box fuzzing,” *Proc. IOLTS*, 2021.
[8] M. Zalewski, “American Fuzzy Lop,” http://lcamtuf.coredump.cx/afl/.
[9] P. Godefroid, M. Y. Levin, and D. A. Molnar, “Automated whitebox fuzz testing,” *Proc. NDSS*, pp. 151–166, 2008.
[10] V. J. M. Manès et al., “The art, science, and engineering of fuzzing: A survey,” *IEEE TSE*, Oct. 2019.
[11] K. Laeufer, J. Koenig, D. Kim, J. Bachrach, and K. Sen, “RFUZZ: Coverage-directed fuzz testing of RTL on FPGAs,” *Proc. ICCAD*, pp. 1-8, 2018.
[12] H. M. Le, D. Groβe, N. Bruns, and R. Drechsler, “Detection of hardware trojans in SystemC HLS designs via coverage-guided fuzzing,” *Proc. DATE*, pp. 602-605, 2019.
[13] C. Lemieux and K. Sen, “FairFuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage,” *Proc. ASE*, pp. 475-485, 2018.
[14] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, “VUzzer: Application-aware evolutionary fuzzing,” *Proc. NDSS*, 2017.
[15] C. Liu, J. Han, and F. Lombardi, “A low-power, high-performance approximate multiplier with configurable partial error recovery,” *Proc. DATE*, pp. 1-4, 2014.