Vol.2015-SLDM-173 No.12
2015/12/2

SATソルバーを援用したカバレッジ駆動設計検証について

浜口 清治[1,a)]

概要：ハードウェアの設計検証は，依然，設計工程におけるボトルネックとなっており，より一層の自動化・高速化が必要となっている．レジスタトランファレベル設計に対する検証に関しては，主としてシミュレーションベースの手法が用いられており，近年では，各種のカバレッジ指標をターゲットとしたカバレッジ駆動検証が利用されている．特定の信号線にトグルを生成する入力パターンを見つけるためにSATソルバを用いるとトグルカバレッジの改善は容易になるが，計算コストが増大する．本稿では，SATソルバーをランダムシミュレーションと組み合わせて使用した場合の効果について述べる．SATソルバを少数回利用すると単純なランダムシミュレーションではカバーすることが難しいトグル条件を効率的にカバーすることができることができることを示す．

キーワード：RTL設計検証，カバレッジ駆動検証，SATソルバ，自動テストベンチ

Abstract: Verification of logic design has been a long-standing bottleneck in the process of hardware design, where its automation and improvement of efficiency has demanding needs. Mainly simulation-based verification has been used for this purpose, and recently, coverage-driven verification has been widely used, of which target is improvement of some metric called coverage. To find input patterns which cause some toggles on each signal, a SAT solver could be used, but this is computationaly costly. In this report, we study the effect of combination of random simulation and usage of a SAT solver. The experimental results show that, in some designs, a small number of calls of a SAT solver can improve entire toggle coverage effectively, compared with simple random simulation.

Keywords: RTL verification, coverage-driven verification, sat solver, auomated testbench

1. はじめに

1.1 背景

ハードウェア設計における検証は，依然，設計工程におけるボトルネックとなっている．レジスタ転送レベルないしはゲートレベル設計記述に対する機能検証については，フォーマル検証とシミュレーションベース検証の２つの方式がある．等価性判定やプロパティ検査などのフォーマル検証手法は，SATソルバの性能改善やSATソルバを利用したアルゴリズムにより，適用可能な規模を拡大してきたが，計算コストが大きいため，適用できる場面は限定的となっている．このため，シミュレーションベース検証が検証方式の主流となっている．

シミュレーションベース検証では，人手または自動でテスト用の入力パタンを準備する必要がある．一般に，人手による入力パタン生成は作成にコストがかかり，コーナーケースを見逃す可能性がある．その意味で，自動的なテストパタンの生成が望ましいが，たとえば単純なランダムパタン生成では，まれにしか生成されない入力パタンに対する動作は検証できないなどの問題がある．

また，シミュレーションベース検証では，フォーマル検証の場合と異なり，網羅性を達成することは基本的には不可能となる．このため，検証の進捗の目安が必要であり，カバレッジ (coverage) とよばれる数値指標が用いられる．カバレッジには設計の記述に着目したコードカバレッジと機能に着目した機能カバレッジがある．コードカバレッジは，さらにステートメントカバレッジやトグルカバレッジなど複数の指標があり，入力パタンを与えればシミュレーションを行うことによって，機械的に計算刷ることができる．設計記述内のステートメントや信号線が，カバーするべきポイント (カバーポイント) として指定される．カバレッジの種類によって，カバーのために必要な論理条件は異なる．機能カバレッジの計測にはアサーションなどなんらかの形での機能の定義が必要となる．

カバレッジ駆動検証 (coverage driven verification) は，

[1] 島根大学総合理工学研究科
Shimane University, Matsue, Simane, 690-8504, Japan
[a)] hama@cis.shimane-u.ac.jp

数値であるカバレッジを指標として，これを改善することを目指す検証方式である．準備した入力パタンによってシミュレーションを行った後，カバレッジ解析を行い，その結果をもとにカバレッジの改善が期待できるような入力パタンの生成を行うという手順を繰り返して適用する．この際問題となるのは入力パタンの生成であり，人手で行うと多くのコストを必要とする．この部分を完全に自動化すると閉ループ型のカバレッジ駆動検証システムとなる．本報告でもカバレッジ駆動検証の自動化を目的としている．

カバレッジ駆動検証の自動化では，1.2 節で述べるように機械学習を用いる手法や SAT を援用する方法がある．機械学習を用いる方法ではなんらかの意味で学習が必要であるが，カバーポイントをカバーするような入力パタンが学習データに含まれていない場合，学習が不可能となり，入力パタンの生成もできない．

本報告では，カバレッジ駆動検証システムのカバレッジ解析後の入力パタン生成に SAT ソルバを用いる．各信号線のトグルに着目したトグルカバレッジを指標としている．入力の生成には [1] で提案されたダイバース SAT ソルバを用いて，トグルカバレッジの改善について評価する．

1.2 関連研究

カバレッジ駆動検証システムでは機械学習を利用するものが多い [5], [6], [7], [8]．機械学習によって入力パタンの持つ性質とカバレッジポイントとの関係を学んでおき，これを利用してカバレッジを改善する．機械学習の枠組みとしては，文献 [7], [8] ではベイジアンネットワークが [6] では帰納的論理プログラミングがそれぞれ用いられている．あらかじめ入力パタンの持つ性質と個別のカバレッジポイントとの性質を学んでおき，これを使ってクロスカバレッジ (複数のカバレッジポイントを組み合わせて評価するカバレッジ) の改善を行う．文献 [5] では回路のアクティビティを観測することによりマルコフモデルの学習を行い，入力パタンの生成方法を動的に調節してカバレッジを改善する．これらはいずれも閉ループ型であるが，対象がマイクロプロセッサまたはマイクロアーキテクチャに限られており，学習の際に注目する入力の性質 (命令の種類や出現確率など) の抽出はマイクロプロセッサに特化した形で人手で行われている．

SAT ソルバを用いてカバレッジを改善する方式 [10], [11] も提案されている．文献 [10] では，ターゲットとなるカバレッジポイントのカバー条件について，その解空間を分割することにより，効率的な解の生成を行っているが，実験の対象が組合せ回路に限定されている．文献 [11] では抽象化を利用して入力パタンの生成を行っているが，少数のカバーポイントをターゲットにしており，対象がマイクロプロセッサに限られている．

文献 [2], [3] でも SAT ソルバーを用いているが，これらでは複数のシナリオ (入力に対する論理制約) について，満足する解の個数がシナリオ間で不均等にならないよう入力生成を行うことにより，カバレッジの改善をはかっているが，ターゲットなるカバーポイントに対するカバレッジ解析に基づいて，入力パタン生成を行っているわけではない．

入力パタンの生成に関しては，与えれた論理制約に対して，できるだけ一様なパタンを生成する方法が提唱されている [1], [4], [9]．これらは与えられた入力制約を満足する解を求めるために利用されている．

2. 準備

2.1 カバレッジとカバレッジ駆動検証

カバレッジ (coverage) には種々の評価指標がある．ステートメントカバレッジ (statement coverage) は，シミュレーション時に設計記述中のステートメントの集合あるいは部分集合の内，実行されたことがあるステートメントの割合として評価される．ここでは，カバーポイント (cover point) は注目している設計記述中または設計の部分と，カバーの条件の対であるとする．

本報告ではカバレッジとして，トグルカバレッジ (toggle coverage) に注目する．トグルカバレッジのカバーポイントは，設計内の信号線 (あるいは変数)，および，2 クロックサイクルの間に信号線の値が 0 から 1 へ変化するという論理条件，または 1 から 0 へ変化するという論理条件の対となる．本報告では 0 から 1 への変化と 1 から 0 への変化は異なるものとみなす．0 から 1 または 1 から 0 への値の変化が起こったとき，トグルしたという．

トグルカバレッジはシミュレーションによってどの程度，設計が活性化されたかの目安として用いられる．100% になった場合でも設計の正しさが保証されるわけではないが，0 から 1 または 1 から 0 への変化が全くない信号線あるいは回路の一部分は，十分テストされていない可能性が大きく，さらに入力パタンを印加するか，あるいは値が変化しない理由を調べる必要があることを示唆している．

カバーポイントの集合 C と (複数サイクルに渡る) 入力パタン P に対して，トグルカバレッジは，シミュレーションを行った際に 1 度以上カバーされたカバーポイントの集合 C' に対して，$|C'|/|C|$ と定義する．入力パタンが複数ある場合も同様に定義される．

図 1 にカバレッジ駆動検証のフローを示す．

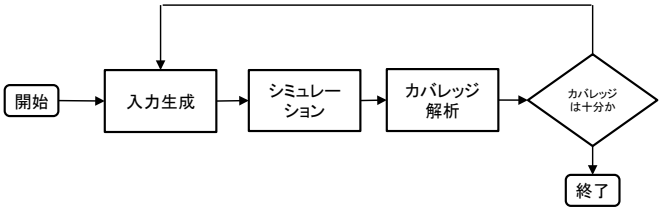

図 1 カバレッジ駆動検証のフロー

また，本報告で想定する適用場面では，例えば検証対象となる設計 (design under verification, 以下 DUV) には，入力に対して正しい出力を与える参照モデル (reference model) が準備されており (図 2)．設計誤りの有無は同じ入力に対する比較によって行う．この枠組みでは DUV に対するカバレッジはできるかぎり大きい方がよい．以下，カバレッジに言及する際は，DUV のみについて考え，テストベンチや参照モデルに対するカバレッジは考えないこととする．

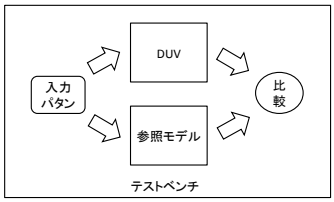

図 2 テストベンチの構成

2.2 ダイバース SAT ソルバ

充足可能性判定問題 (Satisfiability, 以下 SAT) は与えられた命題論理式を 1 とするような変数割当があるかどうか (充足可能であるかどうか) を判定問題である．そのような変数割当を解といい，SAT ソルバは充足可能な場合はその解を見つける．

トグルカバレッジの改善を目指して，カバレッジ駆動検証の枠組みで SAT ソルバーを用いることを考える．トグルの発生条件を DUV の記述とともに SAT ソルバに与えて，その解を入力パタンとするというのが自然な適用方式であるが，計算コストが一般に大きいため，簡単にトグルが発生しないカバーポイントに対して適用し，それ以外は別のもっと計算コストの小さい手法 (ランダムシミュレーション) でカバーすることが望ましい．SAT ソルバを何度も繰り返して起動すると計算コストが増大するため，1 度に多数の解を生成した方が効率が良いと考えられる．

実装上は，1 つの解が得られる度に，その否定を表現する論理式 (通常の SAT ソルバでは節の形式となる) を SAT ソルバの保持している論理式に付け加えた上で，あらたに解を探索するという手順が考えられる．しかしながら，通常の SAT ソルバは連続的に解の生成を行うと，変数に対する 0 と 1 の割り当て方がほとんど同じような解をみつけてしまうことが多い．カバレッジ駆動検証が，可能なかぎり多数のカバーポイントを効率的にカバーすることを考えると，これは望ましいとは言えない．

解集合が与えられたとき，そこに含まれる解のちらばり具合を表す指標が必要となる．図 3 は変数 vi $(i = 1, 2, \ldots, 8)$ に対する 2 つの解の集合 $M = \{\mu_1, \mu_2, \mu_3, \mu_4\}$ と $M' = \{\mu'_1, \mu'_2, \mu'_3, \mu'_4\}$ を表している．変数 $v5, v6, v7, v8$ への割当をみると，M は M' より似通った解の集合になっていることがわかる．

	μ_1	μ_2	μ_3	μ_4
$v1$	1	0	0	0
$v2$	0	1	0	0
$v3$	0	0	1	0
$v4$	0	0	0	1
$v5$	1	1	1	1
$v6$	1	1	1	1
$v7$	1	1	1	1
$v8$	1	1	1	1

(a) 解集合 M

	μ'_1	μ'_2	μ'_3	μ'_4
$v1$	1	0	0	1
$v2$	0	1	1	0
$v3$	1	0	1	0
$v4$	0	1	0	1
$v5$	0	0	1	1
$v6$	0	1	1	0
$v7$	1	0	0	1
$v8$	1	0	1	0

(b) 解集合 M'

図 3 2 種類の解集合

解集合内の解のちらばり具合を評価する指標として，次の分散指標 (diversification quality) [1] を用いる．これは解集合内の全ての解の対についてのハミング距離の総和を，最大値が 1.0 になるように正規化した値に等しい．

$$Q_m = \frac{\sum_{u \in V} p_m^u \times n_m^u}{q \binom{m}{2}} \tag{1}$$

ここで，m は解の個数，p_m^u (n_m^u) は変数 u に 1 (0) 割り当てられている解の個数，q は変数の個数である．

ダイバース k セット問題 (Diverse$k$Set)[1] は，解の個数が k であるような解集合のうち，分散指標の大きな集合を求める問題である．この問題に特化した SAT ソルバをダイバース SAT ソルバと呼ぶことにする．

ダイバース k セット問題を解くためのヒューリスティックはいくつか提案されている [1] が，4 節で説明する実装では，最も簡単な PGUIDE を用いている．式 (1) を見ると，各変数について $p_m^u \times n_m^u$ を最大化すれば良い．$p_m^u + n_m^u = m$ であることから，p_m^u と n_m^u をできるだけ同じ値に近づけるようにすれば良いことがわかる．これには，m 個の解集合に対する $p_m^u - n_m^u$ を各変数 u ごとに記憶しておいて，$m+1$ 個目の解を探索する際に，変数の極性選択の手続きで $p_{m+1}^u - n_{m+1}^u$ が 0 に近づくように変数 u の極性を選ぶことによって制御する．

3. SAT ソルバを用いたカバレッジ駆動検証

3.1 シミュレーションの構成

以下では，単相クロック同期式順序回路としての設計を仮定する．また，リセットのための操作がテストベンチなどの形式で与えられているとする．

シミュレーションの構成の概略は次のようになる．まず，単純なランダムパタンを生成して入力に印加し，カバレッジの上昇がほとんど見られなくなった状態でシミュレーションをいったん停止する．カバーされていないカバーポイントを選択したのち，停止した状態を初期状態として，

Algorithm 1: SATRandom-CDV

Data: D:テストベンチおよび DUV，L:ランダムシミュレーションのサイクル数，N:ダイバース SAT ソルバが生成する解の個数の上限値，E:展開サイクル数 (SAT ソルバ起動時)，$SHalt1$: 停止条件の判定 (ランダム)，$SHalt2$: 停止条件の判定 (SAT 起動後)
Result: Tc:トグルカバレッジ

```
1  begin
2      s = ResetState(D);
3      s = RandSim(D, s, L, SHalt1);
4      U = {s};
5      while SHalt2()==False do
           // SAT ベースシミュレーション
6          S = {};
7          for s ∈ U do
8              S = S∪ SATSim (D, N, s, E);
           // ランダムシミュレーション
9          U = {};
10         for s ∈ S do
11             U = U ∪ { RandSim(D, s, L, SHalt1) };
12     Tc = ToggleAnalysis();
```

Algorithm 2: RandSim

Data: $D, s, L, SHalt$(Algorithm 1 を参照)
Result: s:終了状態

```
1  begin
2      while RSHalt() == False do
3          t = GenRandPattern(D, L);
4          s = RunSim(D, t, s);
```

Algorithm 3: SATSim

Data: D, N, s, E(Algorithm 1 を参照)
Result: S：終了状態の集合

```
1  begin
2      T = GenSATPattern(D, N, s, E);
3      S = {};
4      for t ∈ T do
5          s' = RunSim(D, t, s);
6          S = S ∪ s';
```

これらのカバーポイントのいずれかをカバーする条件を表す論理式を構成する．その充足可能性判定を SAT ソルバで行い，複数の入力パタンを見つけて印加する．到達した状態からさらにランダムシミュレーションを行って，カバレッジの上昇が認められなくなった状態から再び SAT ソルバによる入力パタンの生成を行う．

3.2 手続きの詳細

以上を定式化したものがアルゴリズム 1,2,3 である．これらの手続き内では，カバレッジの解析を同時に行う必要があるが，記述の煩雑化を避けるためここでは記載していない．シミュレーションの切り換え/終了の判定は，２つの条件判定手続き $SHalt1$ と $SHalt2$ によっており，ここにカバレッジ解析も含まれている．

アルゴリズム 1 は，設計記述 DUV とテストベンチ記述を入力としている．テストベンチ記述は，リセットシークエンスのみを実行する記述と与えられた入力パターンを実行する記述の 2 種類を仮定している．また，いずれも最終状態を出力することができる．

アルゴリズム 1 では，まず関数 ResetState によってリセット後の状態に遷移させる (2 行目)．その状態を初期状態としてランダムシミュレーションを行う (3 行目)．得られた状態を起点として SAT ソルバを使い (8 行目)，入力パタンの生成を行ってシミュレーションを実行する．その結果，到達した状態を集合 S に格納する．この S に含まれる状態を起点として再びランダムシミュレーションを行う (11 行目)．なお，SAT ソルバが解をみつけることができない場合は，S が空集合となり，10-11 行目が実行されない．この場合，条件判定手続き SHalt2 がシミュレーションを停止する．

ランダムシミュレーションの手続きはアルゴリズム 2 に示している．L サイクル数分のランダムパタンを生成 (関数 GenRandPattern による) し，実行 (関数 RunSim による) して，判定手続き RSHalt() 内でカバレッジ解析を行い，シミュレーションを停止させるかどうか判定する．

ダイバース SAT ソルバを利用して入力パターンを生成する手続きがアルゴリズム 3 である．関数 GenSATPattern が，入力パターンを生成する．この関数は，まずカバレッジ解析を行って，まだカバーされていないカバーポイントを抽出する．0 から 1(1 から 0) への遷移に関するカバーポイントの信号線の集合を $C_{01}(C_{10})$ とすると，生成される論理式は次のようになる．有界モデル検査 [12] と同様，指定されたサイクル数分，記述の時間展開を行って，その間にいずれかのトグルがいずれかのサイクルで発生する条件を記述する．

$$s(v^0) \wedge \bigwedge_{i=0,\ldots,E-1} R(v^i, v^{i+1}) \wedge T(v) \qquad (2)$$

ここで，v^i $(i = 0, \ldots, E)$ は i サイクル目における状態変数と入力変数の集合を表している．$v = \cup_{i=0,\ldots,E} v^i$ とする．$s(v^0)$ は初期状態を表現する論理条件，R は (1 サイクル分の) 状態遷移関係である．また，T はトグルの発生条件を表す論理式であり，次のように定義される．ここで，変数 u^i は変数の集合 v の要素である．

$$T(v) = \bigvee_{u \in C_{01}} (\neg u^i \wedge u^{i+1}) \vee \bigvee_{u \in C_{10}} (u^i \wedge \neg u^{i+1})$$

3.3 システムの構成

3.2 節で示したカバレッジ駆動検証の手順は，いくつかのツールを組み合わせて実装している．以下に，ツールの利用について述べる．ツール間は Python によるプログラムを作成して結合しており，これについても述べる．

- シミュレーション
  Verilog HDL による設計記述を用いることを前提に，Icarus Verilog シミュレータ [15] を利用する．リセット系列のみを含むテストベンチ記述を準備しておく．この記述から，リセット後の状態を出力するテストベンチ，および開始状態とテストパターンを与えると終了状態を出力するテストベンチを生成するプログラムを作成している．また，ランダムパタンを生成するプログラムも準備している．
- トグルカバレッジ解析
  上記で述べたテストベンチは，VCD (value change dump) 形式のシミュレーション・トレースを出力する．これを解析するプログラムを作成し，各カバーポイントのカバー状態を計算し，全体のトグルカバレッジを計算する．
- 論理式への変換
  Verilog 記述を論理式に変換するためには，レジスタ転送レベルで書かれた記述については，ゲートレベルへ変換する必要がある．この変換には Yosys[14] を利用している．Yosys はゲートレベルの Verilog 記述を生成することができる．出力された記述から論理式 (CNF 式) を生成するプログラムを作成して，SAT ソルバへの入力としている．
- ダイバース SAT ソルバ
  ダイバース SAT ソルバは minisat[13] を改造して作成した．解をみつけるごとに，その否定となる節を付け加えて，次の解を探索する．変数の極性選択の際に，2.2 節で述べた方式を用いる．得られた解は再度入力パタンに変換する．

4. 評価実験

実験環境は OS : ubuntu 14.04 LTS(32 ビット版), メモリ : 2GB, プロセッサ : Intel Corei7-4850HQ @2GHz × 2 である．Icarus Verilog, Yosys, minisat 以外の必要なツールについては Python 2.7.6 により実装した．

4.1 実験対象と諸パラメータ

IWLS2005 のベンチマークから表 1 に示す 6 つの DUV に対して実験を行った．フリップフロップ数 (#FF) と論理素子数 (#Logic) は，Yosys によって得られたゲートレベル記述における値である．それぞれリセット系列を与えるテストベンチを作成して，3.3 節で説明した通り，シミュレーションに利用している．現状のシステムでは SAT ソルバで不定値を処理することができないため，全ての状態変数がリセット時に定数値 (0) を持つように記述を改変している．

DUV	#FF	#Logic	#CovPnt
uart	28	181	56
usb_phy	98	503	196
simple_spi	132	895	264
s5378.v	163	1832	326
s13207.v	275	2901	550
tv80	347	9278	694

表 1 設計記述

ターゲットとしたカバーポイントは，Yosys によりゲートレベル記述の生成を行った後，残った状態変数 (フリップフロップの出力) すべてとした．

実験は 800 秒を上限として行って，最終的なカバレッジを評価した．1 回のランダムシミュレーションは 100 サイクル分に設定している．最初のランダムシミュレーションの繰り返しでは，カバレッジが続けて 10 回変化しなくなった時点で SAT ソルバを起動している．その後のランダムシミュレーションではカバレッジが 3 回以上変化しなくなった時点で打ち切っている．SATSim における論理式の生成では，その時点でカバー回数が 0 となっているカバーポイント全てをターゲットとした．ダイバース SAT の解は 50 個までとした．SAT ソルバを呼び出す際の展開サイクル数は，制限時間内に最初の SAT ソルバの呼び出しが終了する範囲内で，できるかぎり大きな値に設定している．

4.2 結果

実験結果を表 2 に示す．random は単純なランダムパターンを用いた場合 (SAT を利用しなかった場合), simple-SAT は変数の極性選択で分散指標を考慮しなかった場合, divSAT は 3 節での手法を適用した場合の結果を示している．SATLength は時間展開したサイクル数 (式 (2) の E) である．cov. は最終のトグルカバレッジ，sat(秒) は実行時間のうち，SAT ソルバの実行時間を示している．#SAT は SAT ソルバが呼び出された回数の平均値である．

これらの例では，最も高いカバレッジはダイバース SAT ソルバを用いた場合に得られている．一方，ランダムシミュレーションとほぼ同じ程度のカバレッジしか達成できていない場合もある．十分にランダムシミュレーションを行った後に，SAT ソルバを用いることにより，カバレッジは改善されるはずであるが，実際にはランダムシミュレーションによるカバレッジの上昇がとまったと判定する時点が早すぎ，また SAT ソルバの計算時間が占める割合が多いため，十分にランダムシミュレーションが行われていないことが原因であると考えられる．

単純なランダムシミュレーションとダイバース SAT ソルバを利用した場合を比較すると，カバレッジ結果に差が

DUV	random	SAT	simpleSAT			divSAT		
	cov.	Length	cov.	sat(秒)	#SAT	cov.	sat(秒)	#SAT
uart	0.29	50	0.73	395.1	11.0	0.99	465.9	4.5
usb_phy	0.78	200	0.99	677.6	4.5	(∗) 1.00	285.6	1.0
simple_spi	0.63	50	0.60	339.3	10.0	0.68	469.9	15.5
s5378.v	0.90	50	0.94	212.9	6.0	0.95	266.7	4.0
s13207.v	0.56	50	0.56	62.8	1.0	0.56	77.0	1.0
tv80	0.69	20	0.64	631.6	1.0	0.68	735.9	1.0

(*) divSAT は 312.5 秒でカバレッジが 1.0 となった.

表 2 実験結果

みられる．特に，simple_spi, uart, usb-phy はいずれも内部にバッファまたはキューに相当するレジスタを備えている．特定の入力系列が連続的に入力されなければ，バッファやキューに値が貯えられず，このため，特定の内部動作を引き起こすに至らない (関連するカバーポイントがカバーされない)．このような例に対しては SAT ソルバによる直接的な入力パタン生成が有効であると考えられる．

単純に複数の解を生成する SAT ソルバ (simpleSAT) とダイバース SAT ソルバ (divSAT) を利用した場合を比較すると，simpleSAT ではランダムシミュレーションよりも結果が悪化する場合があることがわかる．これは一度に生成される解集合の分散指標が低いため，カバレッジを上昇させる効果が限定的になっている一方で計算コストが増大しているためと考えられる．実際，これらの実験で生成された解集合の分散指標は，ダイバース SAT ソルバの場合は常に 0.1 を越えているが，通常の SAT ソルバの場合は 10^{-3} ないし 10^{-4} 以下であった．一方，ダイバース SAT ソルバ 1 回の計算コストは通常の SAT ソルバの場合よりも大きく，検証時間に占める割合が大きい．このため，実験結果でも呼び出された回数が少なくなっている．

5. おわりに

本報告では，カバレッジ駆動検証において SAT ソルバを用いた場合の効果について閉ループ型の検証システムを構築して評価を行った．これらの結果から，特にダイバース SAT ソルバの利用は，カバレッジの改善に有効であると考えられる．

一方で，SAT ソルバに比べてダイバース SAT ソルバは計算コストが大きいため，適切な適用タイミングを見つけ出すしくみが必要である．また，設計規模が大きくなると，SAT ソルバが解をみつけられない場合がある．これについては，文献 [11] などの手法により，計算コストを緩和することが考えられる．また，SAT ソルバによる直接的な入力パタンの発見が難しい場合には，文献 [5] や文献 [7],[8] のように機械学習を援用する手法と組み合わせることも考えられる．

参考文献

[1] A. Nadel, “Generating Diverse Solutions in SAT”, Theory and Applications of Satisfiability Testing-SAT, pp.287-301, 2011.
[2] S. Yang, R. Wille, and R. Drechsler, “Determining cases of scenarios to improve coverage in simulation-based verification,” Symp. on Integrated Circuits and System Design, 2014.
[3] S. Yang, R. Wille, D. Große, and R. Drechsler,“Coverage-driven stimuli generation,” EUROMICRO Symp. on Digital System Design, pp. 525-528, 2012.
[4] Supratik Chakraborty, Kuldeep S. Meel, Moshe Y. Vardi, A scalable and nearly uniform generator of SAT witnesses, Computer-Aided Verification, 2013.
[5] I. Wagner, V. Bertacco, T. Austin, “Microprocessor Verification via Feedback-Adjusted Markov Models”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.26, no.6, pp.1126-1138, 2007.
[6] C. Ioannides, K. Eder, “Coverage directed test generation automated by machine learning - a review,”, ACM Trans. on Design Automation of Electronic Systems, vol. 17, no. 1, pp. 7:1-7:21, 2012.
[7] S. Fine and A. Ziv, “Coverage directed test generation for functional verification using bayesian networks,” Design Automation Conference, pp. 286-291, 2003.
[8] Y. Katz, M. Rimon, A. Ziv, and G. Shaked, “Learning microarchitectural behavious to improve stimuli generation quality,” Design Automation Conference, pp. 848-853, 2011.
[9] Nathan Kitchen, Andreas Kuehlmann, “Stimulus Generation for Constrained Random Simulation” International conference on Computer-aided design, pp. 258-265, 2007.
[10] S. M. Plaza, I. L. Markov, and V. Bertacco, “Toggle: A coverage-guided random stimulus generator,” Int’ l Workshop on Logic Synthesis, pp. 35-1357, 2007.
[11] S. Shyam, V Bertacco, “Distance-Guided Hybrid Verification with GUIDO”, Proceedings of the conference on Design, automation and test in Europe, p. 1211-1216, 2006.
[12] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, “Symbolic model checking without BDDs,” Tools and Algorithms for the Construction and Analysis of Systems, pp. 193-207, 1999.
[13] N. Eén and N. Sörensson,”An extensible SAT solver” Conference on Theory and Applications of Satisfiability Testing, LNCS, vol. 2919, pp. 502-518, 2004.
[14] Yosys, http://www.clifford.at/yosys/about.html.
[15] Icarus Verilog, http://iverilog.icarus.com/.