Vol. 34 No. 5　　情　報　処　理　　May 1993

解　説

IPS

BDD（二分決定グラフ）

4. BDD の CAD への応用†

藤田昌宏††　Edmund M. Clarke†††

1. はじめに

BDD (Binary Decision Diagram, 二分決定グラフ) を利用することにより，デジタルシステムの設計でよく使用される論理関数の多くを他の論理関数の表現手法(たとえば，積和形論理式など)と比べ，きわめてコンパクトに表現できる．BDD を利用すれば，論理関数の操作も計算機上で効率的に行えるため，最近のハードウェア設計 CAD 技術のうち，論理関数処理を必要とするものでは，ほとんどのプログラムでなんらかの形で利用されている．

表-1 に，BDD の論理設計 CAD への応用の一覧を示す．表から分かるように，1980 年代の終わりごろから，きわめて多くの応用研究が活発に進められている．本稿では，論理 CAD の基本問題である，論理検証，論理合成，テスト生成が，BDD を利用することにより，いかに効率化されるかについて，いくつかの例を用いて説明する．いずれの場合も，基本は，「BDD を利用することで，より複雑な論理関数をより効率的に操作できる」という点を活かす形で応用されている．なお，本稿で述べたもの以外にも，BDD はさまざまな形で応用されている．詳細については，表-1 に示した文献を参照されたい．

2. 論理検証への応用

シミュレーションでは，与えたテストパターンについて設計の正しさが保証されるだけであるが，論理検証（シミュレーションと明確に区別するため，形式的検証と呼ばれることもある）では，与えられた設計が与えられた仕様を満たすか否かを形式的に（数学的に）証明する．設計対象が複雑になるにしたがい，シミュレーションで誤りを完全に取り除くことが困難になっており，論理検証の重要性は増大している．

デジタルシステムを設計する際には，二つの回路が同じ動作をするかを検証する場合と，なんらかの形で与えられた仕様（設計が満たすべき性質）を与えられた順序回路が満たすかを検証する場合がある．

2.1 二つの回路の比較

二つの組合せ回路を比較するには，回路から論理関数を抽出し，それらの等価性を調べればよい．BDD を利用する場合には，その正規性から二つの回路に対応する BDD を作り，それらが一致するかを調べるだけでよく，BDD の直接の応用であると言える．この場合，実用上重要なのは，どれだけ大きな回路が扱えるかであり，使用する変数順によって BDD の大きさが大きく変化することから，BDD で使用する変数順をいかに決めるかが主な研究対象となっており，現在までに，さまざまな手法が開発されている．BDD が利用されるまでは，与えられた回路を積和形論理式に変換する手法が主流であり，数百ゲート程度の回路しか扱えなかった．ところが，1988 年に初めて実用的な BDD の変数順決定手法が提案され[8),16)]，数千ゲートの規模の回路も実用時間で処理できるようになった．組合せ回路の検証が論理検証の基本であるため，論理検証自体も実用技術として急速に注目を集めるようになったのは，主に BDD が利用されるようになってからである．現在，さらに大きな回路や BDD が不得手な回路(かけ算器など)を取り扱えるように BDD の拡張についても活発に研究されている[12)]．また，二つの順序回路の比較でも，BDD を使用して初めて実

† Application of BDD to CAD for Digital Systems by Masahiro FUJITA (Processor Laboratory, Fujitsu Laboratories Ltd.) and Edmund M. Clarke (School of Computer Science, Carnegie Mellon University).

†† (株)富士通研究所プロセサ研究部
††† カーネギーメロン大学計算機科学科

表-1 BDD の論理 CAD への応用

検　証	記号シミュレーション	組合せ回路の論理，順序回路の論理，遅延シミュレーション[10]，など
	組合せ回路	最適変数順決定[8),16)]，確率的検証[11)]，BDD の拡張[12)]，など
	順序回路	二つの順序回路の等価性[14)]，モデルチェッキング[2)]，など
	タイミング検証	false path 解析[18)]，遅延故障テスト生成，など
合　成	積和形論理式簡単化	簡単化[25)]，全主項の集合の生成[6)]，Boolean Relation の簡単化[23)]，など
	組合せ回路最適化	回路変換による最適化[3),17)]，回路最適化のためのドントケア算出[21)]，タイミング最適化，など
	順序回路最適化	状態遷移表最小化，状態符合最適化，順序回路の初期化系列，など
	テクノロジマッピング	論理を考慮したセルの割当[7)]，など
	FPGA 合成	マルチプレクサタイプ FPGA 最適化[13)]，万能セルタイプ FPGA 最適化，など
	再設計	既存の回路の修正／追加[9),15),23)]，など
テスト	組合せ回路テスト生成	記号シミュレーションの応用，パターン数最小化[26)]，など
	順序回路テスト生成	検証手法の応用[4)]，スイッチレベル回路[5)]，など
	故障シミュレーション	多重故障[22)]，など

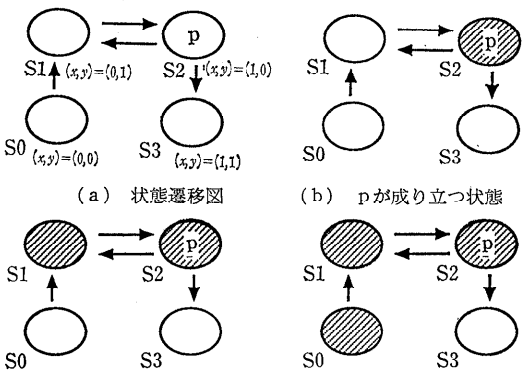
用規模の回路が検証できるようになっている[14)].

2.2 設計が仕様を満たすかの検証

与えられた順序回路が特定の仕様（性質）を満たすかを調べる論理検証については，仕様を時相論理 (Temporal Logic) で記述し，モデルチェッキングと呼ばれる手法で効率的に検証する手法が現在広く利用されている．以下では，BDD を利用したモデルチェッキングについて，例題も含め，少し詳しく説明する．

モデルチェッキングは，1980 年代の初めに提案され，非同期回路の検証などで実設計の誤りを指摘し，その有用性は広く認識されていたが，扱える規模が小さく問題であった．1980 年代の終わりに，BDD により回路の状態値として記号値を使用し，すべての状態を列挙することなく，検証する手法が提案され，後に示すような実回路の検証が数多く試みられるようになった．

ここでは，**図-1**（a）に示す状態遷移図で記述された設計を例としてアルゴリズムを簡単に説明する．詳細については，文献 2) を参照されたい．図-1(a)では，状態 S0 を初期状態とし，動作は S0 から始めるとする．矢印は状態遷移を示し，次の時刻で矢印の先に遷移する．一つの状態から複数の矢印が出ている場合には，そのいずれにも状態遷移できると解釈する．また，状態 S2 でのみ p が成り立ち，それ以外では，p は成り立たないとする．与えられた順序回路から図-1（a）のような状態遷移表現へは簡単に変換することができる．



（a） 状態遷移図　（b） p が成り立つ状態

（c） 1回状態遷移したもの　（d） 2回以上状態遷移したもの

図-1 モデルチェッキングの実行例

CTL (Computation Tree Logic) と呼ばれる時相論理で仕様を記述し，モデルチェッキングを行う手法が，その効率の良さから広く利用されている．CTL は，通常の論理に時間に関する演算子を追加したものである．たとえば，EF という演算子は，「いつかは p が成り立つような動作シーケンスが存在する」ことを要求する．図-1（a）に示す設計に対しては，初期状態 S0 で，EF p という仕様を考えることができる．これは，初期状態 S0 では，ある動作シーケンスでは，いつかは p が成り立つということを表している☆.

モデルチェッキングでは，与えられた時相論理

☆ CTL には，このほかにも時相演算子があり，「常に p が成り立つ」や「q になるまで p が成り立ち続ける」などの性質を記述できるが，基本的に同じ流れで検証できるので，ここでは省略する．

式を現在に関する条件と次の時刻以降に関する条件に展開する変換式を利用して，検証していく．EF p の場合，「現在 p が成り立つか，あるいは，次の時刻で EF p が成り立てばよい」．この条件を使って，設計の各状態に対して，EF p が成り立つか否かを以下のように決定していくことができる．まず，今現在 p が成り立っている状態は，即座に EF p が成り立つと言える．したがって，**図-1**（b）のように斜線を付けた状態で EF p が成り立つ．次に，「現在 p が成り立つか，あるいは，次の時刻で EF p が成り立てばよい」という性質から，斜線の付いた状態に遷移している状態も斜線を付けることができる．これを1回適用することで，**図-1**（c）のように斜線を付けることができる．もう1回適用すると**図-1**（d）のように斜線を付けることができる．しかし，これ以上適用しても斜線を付けられる状態は増えないので，図-1（d）が最終的な結果となり，初期状態 S0 にも斜線がついているので，設計は仕様を満たすということができる．

以上のようにモデルチェッキングは，以下の二つの基本操作により実現できる．

1. 現在斜線の付いた状態へ遷移している状態にも斜線を付ける
2. 斜線の付いた状態が増加したかどうか判定する

従来の BDD を使わない方法では，各状態一つずつに対し，上の二つの基本操作を施していた．順序回路の場合，フリップフロップの数が n 個あると状態数は 2^n 個まであるので，回路が少し大きくなると扱えなくなっていた．

BDD を使う手法では，上記の基本操作をいずれも論理演算に置き換えて処理する．状態数の LOG の数だけ変数を用意し，それらの変数の値の組合せで，各状態を表現する（図-1 では，用意する変数を x,y とし，状態 S0, S1, S2, S3 を x,y の値として，それぞれ，00, 01, 10, 11 と表すことにする）．このようにすると，状態の集合をこれらの変数の論理式で表すことができる．たとえば，図-1（b）で斜線が付いた状態の集合（S2 のみ）は，論理式 $x\bar{y}$ で表現できる．同様に図-1（c）（S1 と S2）は，$\bar{x}y+x\bar{y}$ となる．これは，状態遷移関係を x,y の論理式で表現することにより，以下のように，論理演算として計算することができる．

まず，x',y' をそれぞれ x,y に対する次の時刻の値を示す変数とする．すると，図-1 の状態遷移から次式が成立する☆．

$$x'=x\bar{y}+\bar{x}y,$$
$$y'=\bar{y}$$

図-1（b）から（c）を求めるには，次の時刻に状態 S2 つまり，$x\bar{y}$ となる x,y を上の式から求めればよい．$y'=0$ から $y=1$ と決まり，その結果，$x=0$ と分かり，求める状態（の集合）は，状態 S1 であることが分かる．以上の計算は，論理関数の操作として一般的に実行できる（つまり，上の基本操作 1. を論理演算で実現できる）．

このように状態の集合を論理式で表しておくと，上の基本操作 2. も，状態の集合を表す論理式が等価か否かで判定することができる．したがって，論理式を BDD で表現しておけば，複雑な論理式が扱える（したがって，より複雑な設計が扱える）とともに，BDD の正規性により，基本操作 2. も容易に実行できる．この状態の表現手法では，状態数に対し LOG の数しか変数が必要なく，順序回路の場合でもフリップフロップの数だけ変数を用意すればよい．

ここで実用システムの検証例として，並列計算機 (Encore Gigamax multiprocessor) の分散キャッシュシステムのプロトコルの検証について説明する[19]．Gigamax は，共有メモリで結ばれた複数のプロセッサを一つのクラスタとし，それらクラスタがさらに分散システムとして結合された，**図-2** のような構成の並列計算機システムである．

一つのクラスタ内のバスにおいては，スヌープキャッシュに基づいて管理されており，それらクラスタ間は，UIC というインタフェースモジュールにより，ネットワークとして結合されている．この UIC が，クラスタ間のキャッシュの管理を行う．検証を行うために，図-2 の構成に基づくアーキテクチャレベルの動作を複数の互いに通信する有限状態機械で記述し，「キャッシュへの書き込みが止まれば，すべてのキャッシュの内容はメモリのものといつかは必ず一致する」，「キ

☆ 図-1 の状態遷移から，x が1となるのは，状態 S2 か S3 にいる場合であり，それらの状態には，S1 と S2 から遷移していくことができることが分かる．したがって，次の時刻に x が1となる（つまり $x'=1$）のは，現在，状態 S1 か S2 にいる場合であり，論理式では，$x\bar{y}+xy$ となる．y' についても同様．

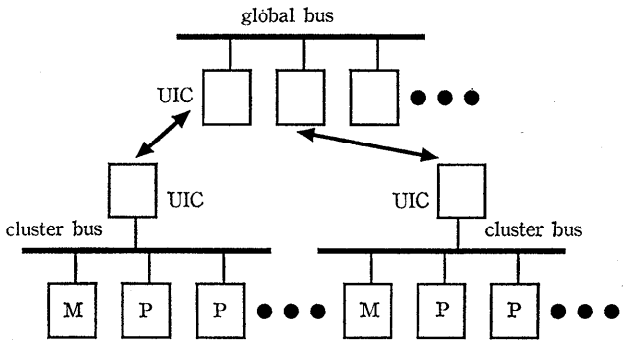


図-2 検証対象計算機システムの構造

ャッシュプロトコルがデッドロック状態にならない」,「システムに組み込まれている診断システムがエラー状態となることはない」という仕様を検証した.

クラスタが二つで,一つのクラスタ内に6プロセッサある場合 SPARC 2 で,約 30 秒で上記の仕様を検証できる.この場合,状態遷移を表す関係を表現する BDD の大きさは 37556 ノードであった.

実際に,いくつかの誤りが発見されたが,そのうちの一つは,その誤状態に到達するのに少なくとも 15 サイクルが必要であった.したがって,ランダムなシミュレーションにより,この誤設計が発見される可能性はほとんどなく,論理検証の有用性は高いと言える.

2.3 タイミング検証

回路の遅延時間を正しく計算することも,デジタルシステムを正しく動作させる上で重要であり,一般にタイミング検証と呼ばれている.

たとえば,**図-3** に示す回路を考える.今,すべてのゲートの遅延が1であるとすると,単純には,この回路の遅延は3であると言える(a, d, f, gや b, d, f, g というパスの長さ).ところが,実際には,これら長さ3のパス上を信号は伝播しない.これは,次のようにして言える.今,パス a, d, f, g 上を信号を伝播させることを考える.信号が a から d に伝わるためには,b は1でなければいけない(0だと即座に d は0と決まってしまう).

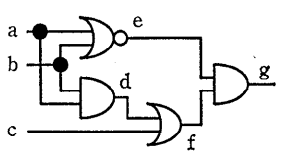


図-3 false path の例

すると e は0となり,その結果 g も0と決まってしまう(この信号の流れは長さ2である).パス b, d, f, g も同様に信号が伝播しないため,この回路の遅延は実際には3ではなく2であると言える☆.パス a, d, f, g や b, d, f, g は false path と呼ばれる.

一つのパス上を信号が伝播するか否かは,以下のように論理関数処理として扱える.あるパス上を信号が伝播するには,そのパス上の各ゲートのそのパス上にない入力(サイド入力と呼ばれる)は,AND, NAND ゲートなら1, OR, NOR ゲートなら0でなければならない(さもなければ,信号はそこから先へは伝播しない).したがって,サイド入力の論理関数を計算し,それらが指定された値になるような外部入力値があれば,そのパス上を信号を伝播させることができる.もし,なければ,決してそのパス上を信号が伝播することはなく,false path であるということができる.図-3 のパス a, d, f, g については,b の論理関数を1にし, c の論理関数を0にし, e の論理関数を1にするような入力を求めることになり,これらの論理関数の積を求めて,それが恒等的に0になる(したがって,この条件を満たす外部入力はない)と,false path であると言える.この手法は,BDD を計算できるような回路規模に対しては効率的であり,プログラム化され,一部実用化されている.

3. 論理合成への応用

イリノイ大学で開発されたトランスダクション法による論理回路最適化を例として, BDD を利用することで,いかに効率化できるかを説明する.

トランスダクション (transduction) とは, transformation & reduction の略で,回路の変形 (transformation)・冗長部分の削除 (reduction) を繰り返し行うことによって,回路の最適化を行うものであり,1970年代にイリノイ大で提案・開発されたものである[20].以下の説明では,論理関数を論理

☆これは,スタティックな解析であり,ハザードまで考慮するダイナミックな解析では,長さ3のパスをハザードが走る場合があることが分かり,遅延は3となる.ダイナミックな解析も以下のスタティックな場合と同様に解析できるため,ここでは省略する.

値のベクタの形で表し，論理値としては，0，1，ならびに * を用いる．* はドントケアを表す．たとえば，**図-4** に示すように，入力変数 a, b の積である c は，

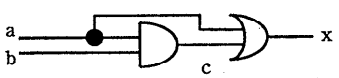
$$c=ab=[0001]$$

となる．以下では，真理値表を用いて説明していくが，BDD は，真理値表のきわめてコンパクトな表現ということができるため，すべて容易に BDD で効率的に実装できる．

トランスダクション法は許容関数（permissible function）と呼ばれる論理関数をもとに最適化を行う．回路中の変数 v において実現されている論理関数 f を他の論理 f' に置き換えても，回路のどの出力変数の論理関数も変化しないとき，この f' を変数 v に対する許容関数と呼ぶ．図-4 において，c の実現している論理関数は，[0001] であるが，これをたとえば，[0011] に変えても，x の論理関数は変化しないので，[0011] は変数 c の許容関数であると言える．同様に [0010]，[0000] も許容関数となるため，変数 c の許容関数はまとめて [00**] と表される．これは，定数を 0, 1, * の 3 値に拡張した BDD を用いて容易に表現でき，図-4 に示すようになる．真理値表は，入力数 n に対し，2^n の大きさとなるが，BDD で表現することにより，かなり大きな n でも取り扱うことができる．

許容関数は以下のように計算できる．

1. 入力変数から出力に向かって，各変数の実現している論理関数を求める．
2. 出力変数の許容関数を求める．明示的なドントケア条件がある場合は，そのドントケア条件を考慮する．そうでない場合は，出力変数の論理そのものが許容関数となる．
3. 許容関数の求められた変数（ゲート）の入力となっているネットの許容関数を求める．



a	b	c	x
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

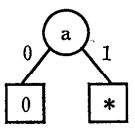


図-4 論理回路と許容関数の例

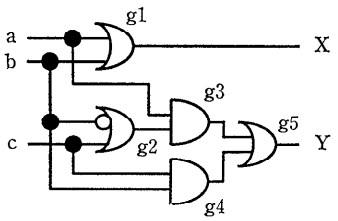
4. ネットの許容関数から変数の許容関数を求める．
5. まだ，許容関数の計算されていない変数があれば，3. へ．

上の 3.，4. の処理が許容関数の計算の中心である．これらでは，真理値表の各値を順に求めていくことで，許容関数を計算する．ここでは，3. について説明する．3. では，ゲート g の許容関数からそのゲートの入力 g_i の許容関数を計算する．これは，ゲートの種類を考慮して，真理値表の各値ごとに計算できる．今，ゲートが AND ゲートであるとする．g の許容関数の値が * であるとすると，g_i の許容関数も * となる（ゲートの出力がドントケアなので，入力は何でもよい）．g の許容関数が 1 であると，g_i の許容関数も 1 でなければならない．ところが，g の許容関数が 0 であると，もし，ゲート g の g_i 以外の入力に 0 があれば，それで AND ゲートの出力も 0 となるため，g_i の許容関数は * とすることができる．このような処理を各ゲートの種類ごと，真理値表の値ごとに行えばよい．以上の処理は，g の許容関数と，g_i 以外のゲート g の入力の論理関数の間の論理演算として計算できる．このため，BDD に関する論理演算である apply を利用して容易に実現できる．

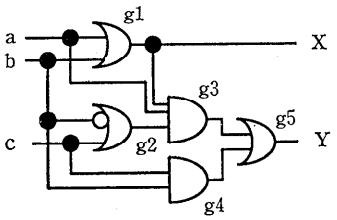
トランスダクション法では，許容関数を利用して回路変換可能かどうかを判定している．回路変換とは，ネットの追加と除去である．たとえば，**図-5**(a) の回路は，これ自体には，冗長なネットは存在しないが，**図-5**(b) のように冗長なネットを追加すると，結果的に**図-5**(c) のように，他の部分で多くの冗長性が発生しそれを除去することで，もとよりも回路を簡単化できる．このような回路最適化では，現在の回路への冗長なネットの新たな追加（追加しても外部出力の論理は変化しないネットの追加）と結果的に生じた冗長なネットの除去が基本であり，いずれも許容関数を利用して，効率的に実現できる[1]．図-5(a) の場合，g3 の論理関数は $a(\bar{b}+c)$ である．g1 から g3 に新たにネットを追加して図-5(b) のような回路にしても，g1 の論理関数は $a+b$ であるから，g3 の論理関数は，

$$(a+b)a(\bar{b}+c)=(a+ab)(\bar{b}+c)=a(\bar{b}+c)$$

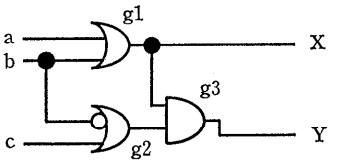
となり，変化しない．このようにネットが追加で



（a） 簡単化対象回路



（b） 冗長なネットの追加



（c） 発生した冗長ネットの削除後

図-5 ネットの付け換えによる論理回路の最適化

きるか否かや，削除できるか否かの判定は論理関数を毎回作成し直せばよい．トランスダクション法では，許容関数を利用することにより，これを効率的に実現している．

まず，ネットが追加できるかどうかは，実際にネットを追加して追加されたゲートの出力の論理関数を計算し直し，それがそのゲートの許容関数に含まれるかどうかを調べればよい．含まれていれば，追加可能であり，含まれていなければ追加できない．より効率的には，以下のように判定する．対象のゲートが AND ゲートの場合，1 の値を追加しても論理に変化はないので，0 の値のみに注目する．接続を試みる変数の論理が 0 になるときのゲートの許容関数が 1 でなければ（すなわち，0 か * であれば）追加可能である．この処理も真理値表の値ごとにできるため，BDD の apply を利用して容易に実現できる．

逆に，あるネットが冗長かどうかは，そのネットの許容関数が 0 と * のみか，1 と * のみからなっているかどうかを調べればよい．このような場合には，そのネットを 0 や 1 の定数関数と置き換えてもよいので，ネットを除去できる．この判定は，BDD では，定数ノードが 0 と * のみか，1 と * のみかを調べればよく，容易に行える．

以上のようにして，BDD を使ってトランスダクション法を効率的に BDD で実現することができる．結果的に得られる性能は，入力変数の数がある程度（だいたい 15）以上になると，真理値表による実現はもちろん，積和形を用いた実現と比較しても，数倍以上の高速化を達成でき，また，他の実現手法では扱えないような大きな回路を扱うことができる[24]．

4. テスト生成への応用

論理回路のテスト生成に関する研究の歴史は長く，各種の効率的なアルゴリズムが開発されてきている．特に，実用上重要な組合せ回路の縮退故障に対しては，冗長故障の検出も含め，実用規模の回路を実用時間で扱えるようになってきている．縮退故障とは，回路中のある信号線の値が，入力値にかかわらず，0 または 1 に固定される故障である．これを検出するテストパターンとは，0 縮退故障の場合，その信号線の値を 1 に設定し，かつ，その信号線の値が 0 か 1 かで外部出力の値が変化する入力値として求められる．この条件は，回路の論理が分かっている場合には，そのまま論理関数の条件として表現できるため，回路の BDD が計算できる場合には，容易にその故障に対するすべてのテストパターンを求めることができる．

しかし，実際には，テスト生成で取り扱わなければならない回路規模は，1 万ゲート以上のことが多く，BDD を生成できる回路規模（数千ゲートくらい）より大きい．このため，BDD を直接，縮退故障のテスト生成に応用することは，効果的でなく，従来のテスト生成手法のほうが強力である☆．しかし，全故障を検出するテストパターン数をできるだけ減らしたい場合には，各故障に対するすべてのテストパターンを生成できるという BDD を利用したテスト生成手法の特徴は有益であり，その方向の研究も進められている[26]．

スキャンを利用しない順序回路のテスト生成に関しては，従来のテスト生成手法でも効率的には

☆ 通常のテスト生成手法は，回路の構造（接続のされ方）を効率よく辿ることで，論理回路の性質をうまく利用して，信号が伝播されるか否かを調べており，BDD を直接使う手法より，特に大規模回路で効率的である．

取り扱えていないため，BDD による二つの順序回路の等価性判定手法の応用が効果的であると言える．これは，故障がある回路と正常な回路の二つの回路の等価性を **2.** の手法で調べ，もし両者が一致すれば，その故障は冗長であり，また，一致しなければ，反例がテストパターンとなる．この手法は検証可能な回路規模程度までの回路に対してテスト生成でき，だいたい数百ゲートまで扱える．これは，一見小さいようにみえるが，スキャンなしの順序回路のテスト生成はたいへん難しく，これでもある程度実用であると言える．

また，故障シミュレーションへ BDD を応用する研究も行われている．上では，すべて故障は回路中に一つのみとしているが，回路が大規模化するにつれ，複数の故障が同時に存在する（多重故障と呼ばれる）ことも考えられる．したがって，多重故障のためのテスト生成技術，特に，多重故障をいかに故障シミュレーションするかも重要である．回路中にゲートが n 個あると故障が同時に1個しかなければ，全故障数は n 程度であるが，k 重故障を仮定すると，故障は，n^k 程度ある．したがって，故障シミュレーション時に多重の故障を考慮する必要があり，メモリ消費量が急増してしまう．ところが，故障の集合を論理式で表して☆，故障シミュレーションすると，メモリ消費量をかなり押さえられることが分かっている[22]．これは，BDD のうまい応用であると言える．

5. おわりに

BDD の論理 CAD への応用として，論理検証，論理合成，テスト生成のいくつかの手法について説明した．これらはいずれも BDD を使用して初めて実用的規模の回路を扱うことが可能となっている．

CAD の国際会議では，ここ2，3年，論理 CAD 関係の投稿論文の半数以上で BDD がなんらかの形で利用されており，BDD は研究レベルではすでに浸透している．今後も BDD の拡張も含めて，研究・開発は活発に続けられると思われる．

一方，実用ツールに関しても，論理検証，論理合成では，すでに内部に取り込まれており，知らず知らずのうちに BDD を使っている設計者も多いのではないかと思われる．ツールの外に BDD がみえてくることはほとんどないと考えられるが，本稿で述べたように，BDD によってツールの性能が大幅に向上していることを理解していただければ幸いである．

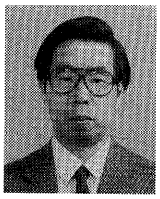
☆ 2. のモデルチェッキングで，状態の集合を論理式で表したのと同じ方法で故障の集合も論理式で表現できる．

参考文献

1) Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. Computer*, Vol. C-35, No. 8, pp. 667-691 (Aug. 1986).
2) Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L. and Hwang, J.: Symbolic Model Checking: 10^{20} States and Beyond, In *Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science* (June 1990).
3) Chen, K. C. and Fujita, M.: Efficient Sum-To-One Subsets Algorithm for Logic Optimization, In *Proc. of 29th DAC* (June 1992).
4) Cho, H., Hachtel, G., Jeong, S-W. Plessier, B., Schwarz, E. and Somenzi, F.: ATPG Aspects of FSM Verification, In *Proc. IEEE Int. Conf. on Computer-Aided Design* (*ICCAD-90*), pp. 134-137 (Nov. 1990).
5) Cho, K. and Bryant, R. E.: Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation, In *Proc. of 26th DAC*, pp. 418-423 (June 1989).
6) Coudert, O. and Madre, J. C.: Implicit and Incremental Computation of Primes and Essential Primes of Boolean functions, In *Proc. 29th ACM/IEEE Design Automation Conf.*, pp. 36-39 (June 1992).
7) Ercolani, S. and De Micheli, G.: Technology Mapping for Electrically Programmable Gate Arrays, In *Proc. of 28th DAC*, pp. 234-239 (June 1991).
8) Fujita, M., Fujisawa, H. and Kawato, N.: Evaluation and Implementation of Boolean Comparison Method Based on Binary Decision Diagrams, In *Proc. of ICCAD-88*, pp. 6-9 (Nov. 1988).
9) Fujita, M., Tamiya, Y., Kukimoto, Y. and Chen, K. C.: Application of Boolean Unification to Combinational Logic Synthesis, In *Proc. of IEEE Int. Conf. on Computer-Aided Design*, pp. 510-513 (Nov. 1991).
10) Ishiura, N., Deguchi, Y. and Yajima, S.: Coded Time-Symbolic Simulation Using Shared Binary Decision Diagram, In *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 130-135 (June 1990).
11) Jain, J., Bitner, J., Fussell, D. S. and Abraham, J. A.: Probabilistic Design Verification, In *Proc. of ICCAD-91*, pp. 468-471 (Nov. 1991).
12) Jeong, S. W., Plessier, B., Hachtel, G. and

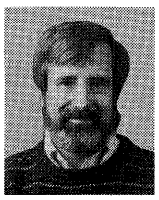
Somenzi, F.: Extended BDDs: Trading off Canonicity for Structure in Verification Algorithms, In *Proc. of ICCAD-91*, pp. 464-467 (Nov. 1991).
13) Karplus, K.: Xmap: A Technology Mapper for Table-Lookup Field-Programmable Gate Arrays, In *Proc. of 26th DAC*, pp. 240-243 (June 1991).
14) Madre, J. C. and Billon, J. P.: Proving Circuit Correctness Using Formal Comparison Between Expected and Extracted Behavior, In *Proc. 25th ACM/IEEE Design Automation Conf.*, pp. 205-210 (June 1988).
15) Madre, J. C., Coudert, O. and Billon, J. P.: Automating the Diagnosis and the Rectification of Design Errors with PRIAM, In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-89)*, pp. 30-33 (Nov. 1989).
16) Malik, S., Wang, A. R., Brayton, R. K. and Vincentelli, A. S.: Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-88)*, pp. 6-9 (June 1988).
17) Matsunaga, Y. and Fujita, M.: Multi Level Logic Optimization Using Binary Decision Diagrams, In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-89)*, pp. 556-559 (Nov. 1989).
18) McGeer, P. C., Saldanha, A., Stephan, P. R., Brayton, R. K. and Sangiovanni-Vinceltelli, A. S.: Timing Analysis and Delay-Fault Test Generation Using Path-Recursive Functions, In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-91)*, pp. 180-183 (Nov. 1991).
19) McMillan, K. L.: Symbolic Model Checking: An Approach to the State Explosion Problem, Technical Report CMU-CS-92-131, Carnegie Mellon University (May 1992).
20) Muroga, S., Kambayashi, Y., Lai, H. C. and Culliney, J. N.: The Transduction Method—Design of Logic Networks based on Permissible Functions, *IEEE Trans. Comput.*, Vol. C-38, No. 10, pp. 1404-1424 (Oct. 1989).
21) Savoj, H. and Brayton, R. K.: Observability Relations and Observability Don't Cares, In *Proc. of IEEE Int. Conf. on Computer-Aided Design*, pp. 518-521 (Nov. 1991).
22) Takahashi, N., Ishiura, N. and Yajima, S.: Fault Simulation for Multiple Faults Using Shared BDD Representation of Fault Sets, In *Proc. IEEE Int. Conf. on Computer-Aided Design (ICCAD-91)*, pp. 550-553 (Nov. 1991).
23) Watanabe, Y.: Minimization of Multiple-Valued Relations, Technical Report UCB/ERL M 91/48, Electronics Research Laboratory, University of California, Berkeley (May 1991).
24) 松永裕介，藤田昌宏：順序付き2分決定グラフと許容関数を用いた多段論理回路簡単化手法，電子情報通信学会論文誌，Vol. J 74-A, No. 2, pp. 196-205 (1991).
25) 湊 真一：二分決定グラフからの非冗長積和形の高速生成法，電子情報通信学会技術報告，VLD 91-107 (Dec. 1991).
26) 樋口博之，石浦菜岐佐，矢島脩三：記号故障シミュレーションに基づくコンパクトなテスト集合の生成，電子情報通信学会技術研究報告，FTS 91-28 (1991).

(平成5年1月18日受付)



藤田 昌宏 (正会員)

1980年東京大学電気工学科卒業. 1982年同大学院情報工学専門課程修士修了. 1985年同大学院情報工学専門課程博士修了. 工学博士. 同年(株)富士通研究所入社. 以来, 主に論理設計用CADの研究・開発に従事. 1988～1989年イリノイ大学客員研究員. ハードウェア設計支援技術全般に興味をもっている. Formal Methods in Systems Design 誌編集委員. IEEE 会員.



Edmund M. Clarke

1967年 Virginia 大学数学科卒業. 1968年 Duke 大学数学科修士. 1976年 Cornell 大学計算機科学科博士. Duke 大学, Harvard 大学を経て, 現在, Carnegie-Mellon 大学計算機科学科正教授. ソフトウェア, ハードウェアの検証, 及び, 自動定理証明に興味をもっている. Distributed Computing 誌, Logic and Computation 誌, Formal Methods in Systems Design 誌各編集委員. ACM, IEEE, Sigma Xi, Phi Beta Kappa 各会員.