# 大規模論理関数用BDDパッケージ：H-BDD

4L－4

内部こなぎ　鈴木 敬　庄内 亨　清水嗣雄
(株)日立製作所　中央研究所

## 1. はじめに

論理生成、論理検証等、論理DAの各分野のプログラムにおいて、論理関数を計算機上で効率良く扱うことは不可欠である。論理関数処理の効率化を実現する論理関数の表現方法として、BDD(Binary Decision Diagram：二分決定グラフ)が近年注目され、現在まで様々な研究発表がなされている。BDDを用いると、論理関数を比較的少ないメモリ使用量で一意に表現でき、また、論理演算を高速に行うことができる。

BDDをプログラム中の論理関数処理に効果的に用いるために、BDDを用いた論理関数処理用プログラムライブラリ、BDDパッケージの開発が多くの機関で行われている。今回、[1][2]の手法に基づき、大規模論理関数への適用を目的として、BDDパッケージ－H-BDD－を開発した。その概要、特に論理演算結果管理に用いるテーブルのハッシュ関数の改良による処理の高速化について述べる。

## 2. H-BDDの取扱規模

H-BDDでは取り扱い規模拡大のため、ノードを指すポインタを図1の構造とした。ポインタは、ノード数削減のために用いた否定エッジと入力反転エッジ(図2, 3)の二つのエッジ属性ビットとインデクスで構成した。この構造により、H-BDDは1Giga個のノードを扱うことができ、大規模回路への適用が可能となる。

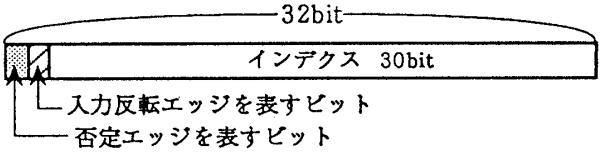

図1　ノードを指すポインタ

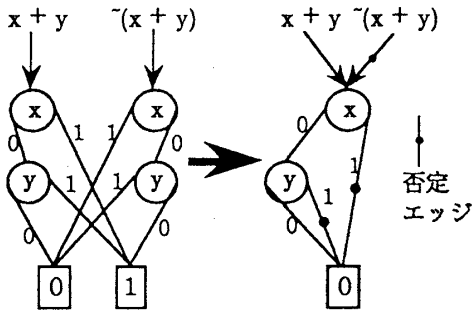

図2　否定エッジ

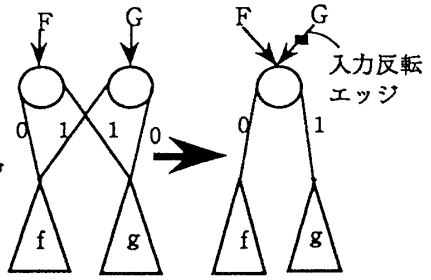

図3　入力反転エッジ

## 3. if-then-else 演算

if-then-else演算(以下ite演算)は ite(f, g, h)=f・g＋(~f)・h (ただし、・:論理積、＋:論理和、~:否定演算)なる計算を行う論理演算であり、全ての二項論理演算をite演算一つで実現できる(例：f・g＝ite(f, g, 0)、f＋g＝ite(f, 1, g))[1]。H-BDDにおいてもite演算によって全ての二項論理演算を実現する。更に、ite演算の演算結果をハッシュテーブルに保持し、再利用を行う。これにより、ite演算の計算回数が減少し、処理時間を削減できる。ハッシュ関数mhashはite演算の引数であるポインタf、g、hを引数(キー)とし、演算結果の格納番地を求める。

## 4 ハッシュ関数mhashの改良

H-BDDの開発では上記のハッシュ関数mhashについて検討を行った。組合せ回路の単一不良修正プログラム[3]にH-BDDを組み込み、mhashを変えて処理時間を測定した。測定データはISCAS'85のベンチマーク回路[4]2種を使用した。測定に用いた不良修正プログラムは、入力回路と不良設定に基づいて不良混入回

路を生成し、不良混入前の回路を仕様として不良の指摘、修正を行う。今回は、2種の回路でそれぞれ2種類の不良設定を行い、処理時間を測定した(表2参照)。開発初期の関数mhash1、mhash2は以下の2式

mhash1(f, g, h) = (f ^ g ^ h) & MASK　(^: ビット演算EXOR、&: ビット演算AND、MASK: テーブルサイズ)

mhash2(f, g, h) = ((f << a) ^ (g << b) ^ (h << c)) & MASK　(A<< B: 数Aを左にBだけシフトした数)

で表され、C499に対して多大の処理時間を要した。調査の結果、以下の二つの場合にハッシュ値の重複が非常に多いことが分かった。

表2 ハッシュ関数と処理時間 (相対値)

| データ回路 | mhash1 | mhash2 | mhash3 | mhash4 |
|---|---|---|---|---|
| C432-(1) | 2 | 1 | 1 | 1 |
| C432-(2) | 15 | 1 | 1 | 1 |
| C499-(1) | >3000 | >3000 | 1 | 1 |
| C499-(2) | >3000 | >3000 | >30 | 1 |

(1) エッジ属性を表すビットのみが異なり、インデクスが全く同じ場合

(2) 引数g、hが恒偽関数又は恒真関数を指す場合

そこで、これらの場合に対処するため、対策(a)(b)を行った。

(a) ハッシュ値計算の際、属性エッジビットも用いる。

(b) gが恒偽関数又は恒真関数の場合、hが恒偽関数又は恒真関数の場合、その他の場合でハッシュ値が異なるようにする。

mhash3はmhash2に対策(a)を行った関数、mhash4はmhash3に対策(b)を行った関数である。mhash3の使用により、C499-(1)では処理時間を大幅に削減できた。また、mhash4を使用すると、C499-(2)で処理時間をmhash3の1／30に短縮できた。以上より、ハッシュ関数mhashにより、処理時間が大きく異なることがわかった。

## 5. H-BDDの処理時間

上記不良修正プログラムを用い、ISCAS'85ベンチマーク回路7種に対し処理時間を測定した。測定はHP9000/730(64MB)上で、各々の回路に対して最大20種の不良設定を与えて行った。各回路の処理時間の平均値を表3に示す。測定の結果、文献[2]のBDDパッケージと同等の処理速度が得られた。

表3 H-BDDの処理時間(秒)

| 回路名 | ゲート数 | 処理時間 | |
|---|---|---|---|
| | | 文献[2] | H-BDD |
| C432 | 160 | 21.44 | 21.68 |
| C499 | 202 | 136.39 | 126.21 |
| C880 | 357 | 16.66 | 13.71 |
| C1355 | 514 | 2285.26 | 2243.35 |
| C1908 | 718 | 709.32 | 652.75 |
| C2670 | 998 | 27.69 | 23.35 |
| C7552 | 2979 | 138.21 | 142.62 |

## 6. 結論

H-BDDは1Giga個のノードを扱うことのできる、大規模回路対応のBDDパッケージである。処理効率向上のため、汎用論理演算であるif-then-else演算の結果をハッシュテーブルに保持し、再利用する方法を用いている。そのハッシュテーブルに関するハッシュ関数mhashについて検討した結果、mhashの良し悪しが、BDDパッケージの処理時間に大きく影響することが分かった。ハッシュ値の重複が多い場合の引数の特徴を考慮し、ハッシュ関数の改良を行ったところ、改良前に比べ、数十倍～数千倍の高速化が達成された。

謝辞：SBDD Packageを提供して頂いたNTTの湊氏に感謝致します。

[1]Brace, K. S., et al.: Efficient Implementation of BDD Package;27th DAC, Paper3.1,pp.40-45 (June 1990).

[2]湊真一, 他: 論理関数の共有二分決定グラフによる表現とその効率的手法 情報処理学会論文誌, Vol.32, No.1,pp77-85 (Jan. 1991)

[3]伊藤雅樹, 他: 不良組合せ回路の単一不良修正方法; 信学技法 VLD92-71,pp9-16 (Jan 1993)

[4]F. Brglez, et al: A neutral netlist of 10 combinational bench-mark circuits and a targettranslatorin FORTRAN ; presented at ISCAS85 (1985).