論理関数の自由度の新しい表現方法とその FPGA 向け論理設計への応用

山下 茂　澤田 宏　名古屋 彰

NTT コミュニケーション科学研究所
〒 619-02 京都府相楽郡精華町光台 2
Tel: 0774-95-1867　Fax: 0774-95-1876
E-mail: {ger, sawada, nagoya}@cslab.kecl.ntt.co.jp

あらまし　本稿では，LUT (look-up table) 型 FPGA (field programmable gate array) 向け論理合成のための論理関数の自由度を表現する新しい手法を紹介する．紹介する手法は，**SPFD(Sets of Pairs of Functions to be Distinguished**) と呼ばれる関数の対の集合を用いることを特徴とする．不完全指定論理関数とは違い，SPFD は LUT の内部論理が自由に変更できという性質を利用することができる．

本稿では，SPFD の応用として LUT の回路を簡単化する手法についても言及する．また，その実験結果も示す．

キーワード　論理関数, 自由度，ルックアップテーブル, FPGA, 簡単化

A New Method to Express Functional Permissibilities and Its Application to FPGA Synthesis

Shigeru Yamashita,　Hiroshi Sawada,　Akira Nagoya

NTT Communication Science Laboratories
2 Hikaridai, Soraku-gun, Seika-cho, Kyoto, 619-02 Japan
Tel: +81-774-95-1867　Fax: +81-774-95-1876
E-mail: {ger, sawada, nagoya}@cslab.kecl.ntt.co.jp

Abstract　This paper presents a new method to express functional permissibilities for look-up table (LUT) based field programmable gate array (FPGA) logic synthesis. The method represents functional permissibilities by using sets of pairs of functions called **SPFD (Sets of Pairs of Functions to be Distinguished)**. Unlike incompletely specified functions, SPFDs make good use of the properties of LUTs such that their internal logics can be freely changed.

As an application of SPFDs, a method to minimize LUT circuits is presented. An experimental result of minimization is also given to show the effectiveness of SPFDs.

key words　logic function, permissibility, look-up table, FPGA, minimization

1 はじめに

FPGA (Field Programmable Gate Array[1]) は，従来の ASIC では考えられなっかった柔軟性を設計者に提供してくれる素子である，近年，その性能が十分実用に耐えうるものとなってきたため，再構成可能なハードウェアへの適用などさまざまな応用面で注目を集めている．

所望の論理回路を FPGA で実現するために，従来の ASIC 向けの設計手法を流用することが可能である．しかし，FPGA のアーキテクチャは（製品によってかなり違いはあるが）従来の ASIC で用いられてきた素子とは大きく異なるため，設計手法も FPGA に特化した手法を開発するのが望ましいと考えられる．そのため，筆者らのグループでは FPGA に特化した論理設計手法を開発し [2]，既に発表している論理合成ツール PARTHENON[3] に組み入れることを検討している．現在のところ，我々は特に Xilinx 等に代表される LUT(Look-Up Table) 型の FPGA を対象として検討している．LUT とはある一定数 k(3 から 5 程度が多い) 入力以下の任意の関数が実現できる万能セルである．そのため，LUT 型 FPGA 向けの論理設計では，まず与えられた論理を全て k 入力以下の関数の多段接続に変換しなければならない．以下では，この変換後の回路を LUT 回路と呼ぶことにする．

一方，論理設計では一旦生成した初期回路を内在する冗長度を利用して簡単化することも重要である．従来，この簡単化の際には回路内の冗長度を不完全指定論理関数を用いて表現していた [4]．LUT 回路の簡単化にも不完全指定論理関数を利用することは可能であるが，それでは LUT が万能セルであることを利用できない．そこで，筆者らは LUT が万能セルであることを利用可能な論理関数の自由度の表現方法を考案した [5]．

本稿では，上述した新しい論理関数の自由度の表現方法の概念とその計算方法を紹介する．その表現方法は，論理関数の自由度を **SPFD (Sets of Pairs of Functions to be Distinguished**) と呼ばれる関数の対の集合を用いることを特徴とする．LUT 回路では，LUT が万能セルであるということを利用できる SPFD を用いれば，従来の不完全指定論理関数よりも広範囲な自由度を表現することができる．そのため，SPFD を利用すれば LUT 回路の簡単化の際により多くの置き換え可能な関数を検出でき，そのことによってより回路を簡単化できることが期待できる．また，SPFD を用いて LUT 回路の簡単化を行った実験結果も示す．

以下では，2 章で本稿で用いる記号の用法および SPFD の概念について述べる．3 章では，SPFD の計算方法を述べる．4 章では SPFD の回路簡単化手法への応用と実験結果を示し，5 章で結論を述べる．

2 SPFD の概念

本章では，SPFD の概念を簡単に説明する．以下では LUT 回路を対象とし，回路内の LUT は L，結線は c という文字を用いて表す．

まず SPFD の概念の説明のために以下の定義を行う．論理関数 f と g において，$g \cdot \overline{f}$ が恒偽関数となる時 f は g を**包含する**と言う．直感的には，g が 1 となる変数割り当てでは必ず f が 1 となる時，f は g を包含すると言われる．

定義 1 以下の 2 つの条件のいずれかを満たしている場合，関数 f は 2 つの関数 g_1 と g_2 を**区別 (distinguish)** しているという．

条件 1 f が g_1 を包含し，$\overline{f}$ が g_2 を包含する．

条件 2 f が g_2 を包含し，$\overline{f}$ が g_1 を包含する．

これは，f の ON-set および OFF-set の一方に g_1 が，他方に g_2 が包含されるということである．ここで，$(g_1 \cdot g_2)$ が恒偽関数であることが前提となっていることを注意して頂きたい．

SPFD の形式的な定義は以下の通りである．

定義 2 複数の区別すべき関数対からなる集合を **SPFD (Sets of Pairs of Functions to be Distinguished)** と呼ぶ．

SPFD は，$\{(g_{11}, g_{21}), (g_{12}, g_{22}), \cdots, (g_{1n}, g_{2n})\}$ という形で表せる．SPFD は単なる関数対の集合ではなく，(g_{1i}, g_{2i}) が区別されるべきことが前提となっているため，$(g_{1i} \cdot g_{2i})$ が必ず恒偽関数であるという条件をもった集合であることに注意して頂きたい．

次に，LUT 回路内の中間地点で実現されるべき論理関数の自由度が SPFD によって表現可能であることを説明する．

LUT 回路内の論理関数について，以下のような考察が成り立つ．回路の中間地点での論理関数の役割は，「その論理関数の真理値表において，ある部分が 1 であり，別のある部分が 0 であるという情報をその論理関数が入力となっている後段の素子へと伝えること」である．この考えに基づいて，ある論理関数の真理値表において，1 でも 0 でもどちらの情報を後段に伝えても良い部分を $*$(ドントケア) として表現するのが，従来の不完全指定論理関数による自由度の表現方法と言える．もし，素子が LUT であれば入力の否定を自由に取ることができるため，論理関数の働きは「その論理関数の真理値表において，ある部分とある部分の値が違う (一方が 1 なら他方は 0) という情報をその論理関数が入力となっている後段の素子へと伝えること」であると言い換えられる．つまり，中間地点での論理関数の働きは，「その関数によってある関数と別のある関数が**区別**されなければならない」と表現できる．より一般的にいうと，ある関数の働きはその関数が区別しなければならない関数の組を複数指定することによって表現できる．(これは後述する例の計算結果を参照して頂きたい．)

以上の考察から次の定理が導かれる．

定理 1 LUT 回路において，ある LUT の出力 (またはある結線) で実現すべき関数 f の回路内での働きは，次章で述べる方法で計算される「f が区別すべき関数の対の集合」，すなわち **SPFD** で表現可能である．

また以下の用語を定義しておく．

定義 3 論理関数 f がある SPFD に含まれる関数の対を全て区別する場合，f はその **SPFD の条件を満たしている**という．

上述したように，SPFD はある関数の区別すべき関数対を指定することによって，その関数が実現すべき働きを表現する．このことは，その関数が該当する SPFD の条件を満たす限り，どのように変更されたとしても回路の出力が変わらない（その際にいくつかの LUT の内部論理変更を要する場合があるが）ということを意味している．従って，SPFD は回路内のある LUT や結線で実現されている論理関数 f の自由度を表現するものであるとも考えられるため，以下ではそのように扱うことにする．

また，以下では次のような記号を用いることにする．

$f(L_i)$**:** LUT L_i の出力で実現されている（回路の外部入力による) 論理関数.

$SPFD(L_i)$**:** LUT L_i の出力で実現すべき関数の自由度を表す SPFD.

$SPFD(c_j)$**:** 結線 c_j で実現すべき関数の自由度を表す SPFD.

ある LUT L_i の出力が複数の LUT への入力となっているとき，それらの出力結線 $c_{i_1}, c_{i_2}, \cdots$ が伝えている関数の自由度は一般には全て異なる．そのため，上記のように $SPFD(L_i)$ と $SPFD(c_j)$ を区別することにする．

定理 1は，「ある LUT L_i の入力結線 c_j を，次章で述べる方法で計算された $SPFD(c_j)$ の条件を満たす関数を実現している LUT の出力で置き換えても，L_i の内部論理を適切に変更すれば回路の出力が変わらない」ということを意味している．この定理の正当性は，次章の計算方法および，結線のつなぎ換えの際の LUT の内部論理の変更の仕方から明らかとなる．ただし，LUT の内部論理の変更の仕方については紙面の都合上省略するため，文献 [5] を参照されたい．

3 SPFD の計算手法

SPFD は回路の出力側から計算される．まず，回路の出力となっている LUT L_i に対しては，$SPFD(L_i) = \{(f(L_i), \overline{f(L_i)})\}$ とする．($f(L_i)$ 以外に $\overline{f(L_i)}$ もこの SPFD の条件を満たすが，LUT の出力関数の否定をとることは自由にできるため，便宜上このように定める．)

次に回路の出力側から SPFD を計算していくが，以下の 2 つの計算が必要となる．

1. $SPFD(L_i)$ が計算されている時に，L_i の入力結線 c_j に対して，$SPFD(c_j)$ を計算する．

2. L_i の出力となっている全ての結線 c_j に対して $SPFD(c_j)$ が計算されている時に，$SPFD(L_i)$ を計算する．

2 つ目の計算方法は複数の SPFD による条件を 1 つの SPFD の条件にまとめることである．これは，(多少の工夫は必要だが) 単に集合の和集合を求めることであるため，本稿では説明を省略する．

以下では，1 つ目の計算方法について例を用いて説明する．今，回路内のある LUT L が 3 つの入力結線 c_1, c_2, c_3 を持ち，それらで実現されている関数の真理値表がそれぞれ図 1の $f(c_1)$，$f(c_2)$，$f(c_3)$ で示されるものとする．ここでは説明の簡略化のため，回路の外部入力として x_1, x_2, x_3 の 3 変数のみを扱うことにする．また，L の内部論理は $(c_1 \cdot c_2 \cdot c_3 + \overline{c_1} \cdot \overline{c_2} \cdot c_3)$ となっているとする．このとき，$f(L)$ は図 1に示す通りとなる．ここで，$SPFD(L)$ は既に計算されていて，図 1の f_0, f_1 に対して $\{(f_0, f_1)\}$ となっているとする．この時，以下の手順で $SPFD(c_1)$，$SPFD(c_2)$，
$SPFD(c_3)$ を計算することができる．

ステップ 1 $f(c_1), f(c_2), f(c_3)$ それぞれのの否定と肯定の全ての組み合わせ (2^3=8 通り) の論理積を求める．これらの論理積を $b_{000}, \cdots, b_{111}$ とする．ここで，b_k の添え字は以下の規則に基づいた 2 進数とする．

- 論理積 b_k に $f(c_i)$ が否定 (または肯定) で現れれば，k の左から i 番目のビットが 0 (または 1) となる．

例えば，$b_{011} = \overline{f(c_1)} \cdot f(c_2) \cdot f(c_3)$ である．ここで，例えば $(f(c_1), f(c_2), f(c_3)) = (0, 1, 0)$ という組合せはありえないので，b_{010} は恒偽関数となる．

ステップ 2 $b_{000}, \cdots, b_{111}$ に対して $a_i = b_i \cdot (f_1 + f_0)$ を計算する．これは，真理値表上で b_i の f_0 および f_1 が 0 である部分，すなわち $f(L)$ がドントケアである部分を無視することに相当する．参考のため $a_{000}, \cdots, a_{111}$ を図 2に示す．

ステップ 3 $a_{000}, \cdots, a_{111}$ から f_1 に包含される a_{001} と a_{111} を選び，その集合を $F1$ とする．f_0 に包含される a_{000}，a_{011}，a_{101} および a_{110} を選んで，その集合を $F0$ とする．a_{010} と a_{100} は恒偽関数のため考慮しない．
なお，$a_{000}, \cdots, a_{111}$ は f_0 または f_1 のどちらかには必ず包含されることに注意して頂きたい．

ステップ 4 $F1$ と $F0$ の要素の考えられる全ての組合せの集合 (集合の直積) を作り．その集合を F とする．

参考のために，ステップ 3 とステップ 4 の計算結果を図 3に示す．

ここまでで計算された F は $SPFD(L)$ の条件を $f(c_1), f(c_2), f(c_3)$ を使ってより細かく表現しなおしたものと言える．つまり，「F の全ての要素 (a_i, a_j) に対して $f(L)$ は a_i と a_j を区別しなければならない．」ということが言える．L の内部論理が自由に変更できることを考えると，このことは次のように言い換えられる．F の全ての要素 (a_i, a_j) に対して，$f(c_1), f(c_2), f(c_3)$ の少なくとも 1 つの関数が a_i と a_j を区別しなければならない．この条件を満たすような関数群なら，どのようなものでも (3 入力でなくても) L の入力群と置き換えられる．つまり，F は L の入力となる関数群の自由度をまとめて表現していることになる．この条件を L の再合成等に用いることが可能だが，ここでは 1 つ 1 つの結線の関数の自由度を計算して回路の入力側に伝搬させていくことを考えているため，とりあえず $SPFD(c_1)$，$SPFD(c_2)$，$SPFD(c_3)$ を計算しなければならない．そのため，以下のステップで F の要素 (区別すべき論理関数の対) を $SPFD(c_1)$，$SPFD(c_2)$，$SPFD(c_3)$ のいずれかに割り当てる．ここでは，少なくとも $f(c_1)$，$f(c_2)$，$f(c_3)$ がそれぞれ $SPFD(c_1)$，$SPFD(c_2)$，$SPFD(c_3)$ の条件を満たすように割り当てる．

割り当てステップ F の 1 つめの要素 (a_{001}, a_{000}) を以下のように割り当てる．(001) と (000) は，左から 3 ビット目だけが異なる．これは，a_{001} と a_{000} を区別できるのは $f(c_3)$ だ

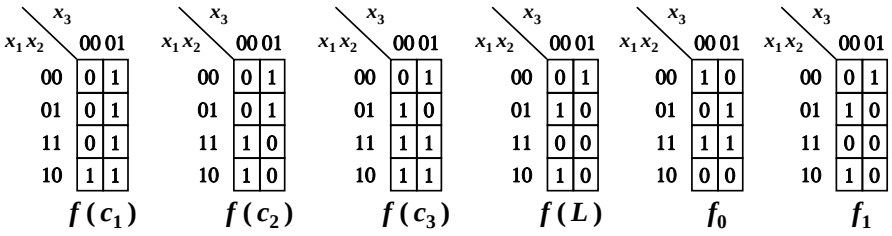

図 1: SPFD の計算例 (1)

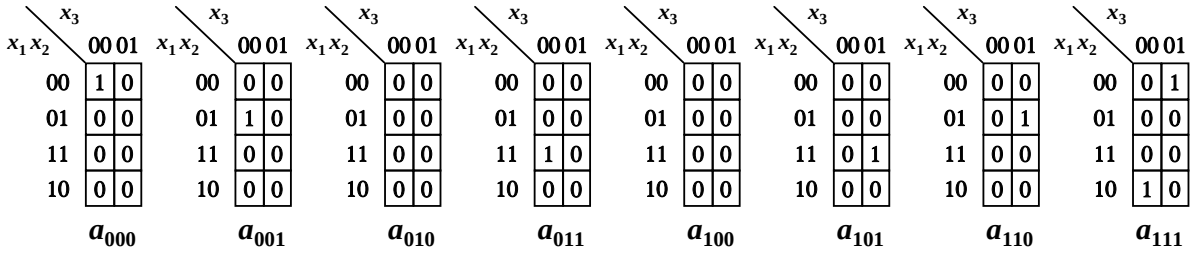

図 2: SPFD の計算例 (2)

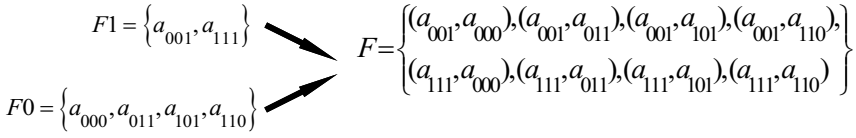

図 3: SPFD の計算例 (3)

けであることを示す．従って，$SPFD(c_3)$ に (a_{001}, a_{000}) を加える．F の次の要素に対して同様に処理を続ける．処理すべき F の要素が無くなれば終了．

a_{001} と a_{000} はそれぞれ，$(\overline{f(c_1)}\cdot\overline{f(c_2)}\cdot f(c_3)$ と $(\overline{f(c_1)}\cdot\overline{f(c_2)}\cdot\overline{f(c_3)}$ であるため，$f(c_3)$ が肯定または否定となっていることのみが違う．そのため，これらは $f(c_3)$ でのみ区別可能だとわかる．このことは，ステップ 1 の規則で添え字をつけておけば，(a_i, a_j) の添え字を見るだけでわかる．

以下同様に計算を進め，F の各要素を $SPFD(c_1)$，$SPFD(c_2)$，$SPFD(c_3)$ のいずれかに割り当てていく．例えば，(111) と (000) は全てのビットが異なるので，a_{111} と a_{000} は $f(c_1)$，$f(c_2)$，$f(c_3)$ のどの関数でも区別可能である．そのため，
$SPFD(c_1)$，$SPFD(c_2)$，$SPFD(c_3)$ のいずれにも (a_{111}, a_{000}) を割り当て可能である．このような場合には c_1 に優先して割り当てるようにすると，計算結果は以下のようになる．

$$SPFD(c_1) = \{(a_{001}, a_{101}), (a_{001}, a_{110}), (a_{111}, a_{000}), (a_{111}, a_{011})\}.$$

$$SPFD(c_2) = \{(a_{001}, a_{011}), (a_{111}, a_{101})\}.$$

$$SPFD(c_3) = \{(a_{001}, a_{000}), (a_{111}, a_{110})\}.$$

例えば，ある LUT L_j に対して $f(L_j)$ がこの計算結果の $SPFD(c_1)$ の条件を満たしていれば，c_1 を L_j の出力に置き換えても，L の内部論理を表す論理式中の c_1 の極性 (否定または肯定) を適当に変更するだけで回路の出力は変化しない．(詳細は文献 [5] 参照．)

4 SPFD を用いた回路簡単化手法および実験結果

4.1 SPFD を用いた回路簡単化手法

SPFD に関する考察が，文献 [6] で述べられている．これによると，SPFD は LUT 回路の簡単化以外にも応用があることが示唆されている．例えば，基本ゲートの回路において従来の技術では簡単化できないが SPFD を用いれば簡単化が可能である例を紹介している．また，SPFD は従来技術では見つけられない boolean divisor を見つけるのに利用できるかもしれないとも言及されている．これらの応用とは別に本稿では，SPFD で LUT 回路を簡単化する手法についてで述べる．SPFD を用いた LUT 回路の簡単化手法は以下の通りである．

ステップ 1 回路内の全ての LUT および結線で実現されている関数の SPFD を計算する．

ステップ 2 回路内から一つずつ結線 c_i を選び，ステップ 3 へ．全ての結線を選んだら終了．

ステップ 3 もし $SPFD(c_i)$ が空であれば，c_i を削除してステップ 5 へ．そうでなければステップ 4 へ．

ステップ 4 回路内のある LUT L_j に対して，もし $f(L_j)$ が $SPFD(c_i)$ の条件を満たしていれば，c_i を L_j の出力で置き換えてステップ 5 へ．そのような L_j が存在しなければステップ 2 へ．

ステップ 5 回路の変形により $f(L_i)$ が変化した場合，L_i の内部論理を $f(L_i)$ が $SPFD(L_i)$ の条件を満たすように変化させる．ステップ 2 へ．

ある結線 c_i が実現している関数 $f(c_i)$ を $SPFD(c_i)$ の条件を満たす別の関数と置き換えても，c_i が入力となっている LUT L_j の内部論理を変更すれば，$f(L_j)$ を $SPFD(L_j)$ を満たすように必ず変更できる．この証明は [6] を参照されたい．また，どのように L_j の内部論理を変更するかは [5] を参照されたい．

この手法では，ステップ 3 およびステップ 4 での c_i，L_j の選ぶ順序で結果が変わる．次節で述べる実験ではこれらの順序は特に考慮をせずに行った．

4.2 実験結果および考察

SPFD が従来の不完全指定論理関数よりも大きな表現能力を持つことは明らかであるが，計算時間もより多くかかることが予想される．そこで，

実際にSPFDを計算するプログラムを実装し，4.1章で述べた回路簡単化手法を適用する実験を行った．実験で回路簡単化手法を適用した初期回路には，MCNCベンチマーク回路[7]からUCBで開発された論理合成ツールSISの推奨スクリプトにより5入力のLUTにマッピングを行ったものを用いた．この回路の簡単化結果を表1の「面積優先」の欄に示す．4.1章で述べた回路簡単化手法のステップ5の結線のつなぎ換えにより回路の段数が増える場合 (例えばdes等) があった．そのため，そのような場合には変形を行わないようにする簡単化手法の実験も行った．この結果を「段数優先」の欄に示す．CPUはSparc Station 20上で測定した計算時間 (秒) を示す．

n入力のLUTの入力結線で実現されている関数のSPFDを求める時には，3章で述べたSPFDの計算のステップ1において2^n通りの論理積を求めている．このことにより，入力関数の任意な組み合わせにより実現可能な関数を全て考慮していることになる．そのためLUTの内部論理が自由に変更できることを考慮にいれていることになり，このことがSPFDが不完全指定論理関数より表現能力が大きいことの原因となっている．しかし，このことから逆にSPFDの計算時間はLUTの入力数に対して指数関数的に大きくなることがわかる．ただ，実際にはLUTの入力は高々5程度であるため，このことはそれほど問題にならないことが実験によりわかった．実験では，内部で論理関数を表現するのには，BDD(Binary Decision Diagram)[8]を用いたが，このBDDが指数爆発を起こさないような回路では計算時間はそれほど問題にならなかった．

また，今回の実験で用いた簡単化手法は4.1章で述べた通り非常に単純なものであるが，初期回路の冗長性をある程度簡単化できることがわかった．ヒューリスティックの改良により，さらに簡単化できることが期待できる．

5　まとめ及び今後の課題

本稿では，論理関数の自由度を**SPFD**と呼ぶ関数の対の集合で表現する手法について述べた．また，SPFDの計算手法とLUT回路の簡単化の実験結果についても述べた．SPFDは従来の不完全指定論理関数よりも広範囲な自由度を表現可能である．また，ベンチマーク実験によれば現実的な時間で計算できることがわかった．

今後は，本稿で紹介した簡単化手法を改良していきたい．また，SPFDの論理合成での他の応用についても考えていきたい．

参考文献

[1] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *FIELD-PROGRAMMABLE GATE ARRAYS*. Kluwer Academic Publishers, 1992.

[2] H. Sawada, T. Suyama, and A. Nagoya, "Logic Synthesis for Look-up Table Based FPGAs Using Functional Decomposition and Support Minimization," in *Proc. ICCAD*, pp. 353–358, Nov. 1995.

[3] Y. Nakamura, K. Oguri, A. Nagoya, M. Yukishita, and R. Nomura, "High-Level Synthesis Design at NTT Systems Labs," in *Proc. of the Synthesis and Simulation Meeting and International Interchange*, pp. 344–353, 1992.

[4] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The Transduction Method-Design of Logic Networks Based on Permissible Functions," *IEEE Trans. Computers*, vol. 38, pp. 1404–1424, Oct. 1989.

[5] S. Yamashita, H. Sawada, and A. Nagoya, "A New Method to Express Functional Permissibilities for LUT based FPGAs and Its Applications," in *Proc. ICCAD*, pp. 254–261, Nov. 1996.

[6] R. K. Brayton, "Understanding SPFDs: A New Method for Specifying Flexibility," in *Notes of International Workshop on Logic Synthesis (IWLS'97)*, May 1997.

[7] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," *MCNC*, Jan. 1991.

表 1: SPFD による簡単化の結果

回路名	初期回路			面積優先				段数優先			
	LUT	結線数	段数	LUT	結線数	段数	CPU	LUT	結線数	段数	CPU
C1908	103	429	13	98	389	13	48.54	98	393	11	32.95
C432	66	275	17	63	257	16	12.25	63	257	16	10.03
alu2	109	482	19	97	378	17	1.75	97	378	17	1.45
alu4	208	862	24	192	723	26	21.23	198	745	22	5.5
apex6	194	894	10	181	803	10	5.23	181	803	10	4.72
apex7	73	292	6	67	247	11	1.19	68	255	6	0.78
cordic	17	76	8	12	52	8	0.17	12	52	8	0.17
dalu	331	1393	16	286	1115	10	15.79	287	1125	9	10.97
des	1118	4663	11	1104	4407	22	3585.08	1111	4508	11	681.79
example2	105	451	5	100	396	8	4.14	101	415	5	2.04
frg2	339	1307	8	278	1019	9	22.08	278	1039	8	15.21
i9	138	679	5	137	675	5	6.88	137	675	5	4.75
k2	536	2325	9	528	2144	13	156.04	533	2267	9	19.66
lal	36	142	4	30	102	8	0.46	31	121	3	0.17
rot	192	753	14	187	707	13	502.3	187	707	13	409.9
t481	404	1738	21	379	1505	21	13.75	379	1505	21	11.75
term1	69	303	7	45	186	6	0.68	45	186	6	0.68
too_large	188	882	12	179	805	12	36.65	179	805	12	36.65
ttt2	53	237	4	46	189	5	0.28	47	196	4	0.36
vda	246	1043	8	239	941	25	280.19	246	992	8	3.42
x1	111	455	6	96	383	7	2.4	99	393	6	3.19
x2	13	56	3	12	48	3	0.04	12	48	3	0.04
x3	205	938	6	189	825	6	7.47	189	830	5	3.43
x4	140	598	4	110	441	4	1.2	110	441	4	1.2
合計	4994	21273	240	4655	18737	278	4723.79	4688	19136	222	1260.8
比率	1.00	1.00	1.0	0.93	0.88	1.1		0.94	0.90	0.9	

[8] R. E. Bryant, "Graph-based algorithm for Boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 667–691, Aug. 1986.