関数分解と依存変数最小化を用いたテーブル参照型 FPGA の論理合成

澤田 宏，須山 敬之，雪下 充輝，名古屋 彰
NTT コミュニケーション科学研究所

あらまし

本稿では，テーブル参照 (LUT: Look-Up Table) 型 FPGA(Field Programmable Gate Array) をターゲットとする論理合成法について述べる．この手法では，関数分解を用いて LUT に割り当てるべき関数を求めている．その際には， disjunctive な分解だけでなく nondisjunctive な分解も用いている．また，本手法では， LUT 回路の合成に特化したブール再代入を用いている．再代入は，既に存在する関数が他の関数を実現するのに有用であるかどうかを調べるのに用いられる技法である．これにより，複数の関数間で部分関数を共有できる．このブール再代入は，不完全指定関数の依存変数最小化を解くことで実行できる．同時に，この技法で LUT 回路の SDC(Satisfiability Don't Care) も扱うことが出来る．

Logic Synthesis for Look-Up Table based FPGAs using Functional Decomposition and Support Minimization

Hiroshi Sawada, Takayuki Suyama, Mitsuteru Yukishita and Akira Nagoya
NTT Communication Science Laboratories

Abstract

This paper presents a logic synthesis method for look-up table (LUT) based field programmable gate arrays (FPGAs). We determine functions to be mapped to LUTs by functional decomposition. We use not only disjunctive decomposition but also nondisjunctive decomposition. Furthermore, we propose a new Boolean resubstitution technique customized for an LUT network synthesis. Resubstitution is used to determine whether an existing function is useful to realize another function; thus, we can share the common function among two or more functions. The Boolean resubstitution is effectively carried out by solving a support minimization problem for an incompletely specified function. We can also handle satisfiability don't cares of an LUT network using the technique.

1 はじめに

テーブル参照 (LUT: Look-Up Table) 型 FPGA(Field Programmable Gate Array) は，プログラム可能な論理ブロックの配列と，それら論理ブロックをつなぐプログラム可能な配線領域から成る．論理ブロックには，LUT とフリップフロップが含まれている．LUT は m(m は 4 か 5 であることが多い) 入力のブール関数を実現できる．

論理合成において，大きな表現から適当な部分表現を抽出することは重要なことである．積和形表現に関しては，カーネル抽出法 [1] などが重要な手法として知られている．LUT 回路の合成に関しては，関数分解 [2, 3] が一つの方法であると考えられ，多くの研究者 [4, 5, 6, 7, 8] がこれを用いている．さらに，何人かの研究者 [5, 6, 7] は，順序つき二分決定グラフ (ordered binary decision diagram: OBDD あるいは単に BDD) [9] で表現された関数に対する関数分解の手法を提案している．本稿でも，BDD 表現に対する関数分解を用いて，LUT に割り当てるべき関数を見つける．多くの研究者は，LUT 回路の合成に対して disjunctive な分解しか用いていないが，本稿では disjunctive な分解だけでなく nondisjunctive な分解も用いる．

論理合成において，共通の部分表現を見つけることもまた，重要なことである．単出力関数それぞれに対する関数分解だけでは，複数の関数の間で LUT を共有することはできない．我々は，再代入の技法を用い，ある関数が他の関数を実現するのに有用であるかどうかを調べている．ある関数をいくつかの他の関数に再代入することにより，その関数がいくつかの関数の間で共通なものであるかどうかを調べることができる．積和形を多段接続した回路に対する再代入の技法は，文献 [1, 10] などに見ることができる．本稿では，LUT 回路の合成に特化したブール再代入の技法を新たに提案する． このブール再代入は，不完全指定関数の依存変数最小化 [11, 12, 13] を解くことで実行できる．また，この技法により，LUT 回路の SDC(Satisfiability Don't Care) も同時に扱うことが出来る．

本稿は以下のように構成されている．2 章では，ブール関数と BDD に関するいくつかの記法を挙げ，関数分解と依存変数最小化について説明する．3 章では，関数分解を用いて LUT に割り当てるべき関数を生成する手法について述べる．4 章では，依存変数最小化に基づく，LUT 回路向けのブール再代入を新しく提案する．5 章では，実験結果およびそれらへの考察を示し，6 章で本稿をしめくくる．

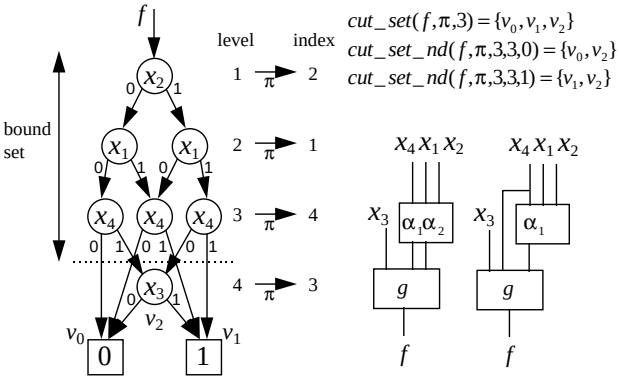

図 1: BDD とその関数分解

2 準備

2.1 ブール関数と BDD

$f : \{0,1\}^n \rightarrow \{0,1\}$ を変数 $(x_1, \ldots, x_n)$ 上のブール関数とする．また，$f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$，$f_{\overline{x_i}} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ とする．関数 f の**依存変数集合 (support)** $sup(f)$ は，その関数が真に依存する変数の集合である．すなわち，$\forall x \in sup(f), f_{\overline{x}} \neq f_x$ かつ $\forall x \notin sup(f), f_{\overline{x}} = f_x$ である．関数 f は，もし $|sup(f)| \leq m$ であれば **m-feasible**，さもなければ **m-infeasible** という．

順序つき二分決定グラフ (BDD) [9] は，ブール関数を表現する非巡回有向グラフである (図 1)．BDD は 2 種類の節点，変数節点と定数節点を持つ．定数節点は，ブール定数 0 または 1 をあらわす．変数節点は，0 と 1 にラベル付けされた 2 つの枝をもち，対応するブール変数への割り当てにより，どちらかの枝が選ばれる．ブール変数への割当に従って任意の変数節点から定数節点までたどる際に，ブール変数はただ一度だけ，しかも定められた順序に従って現れなければならない．ここで，変数節点の**レベル**というものを以下のように定める．ある変数節点 v_i から他の変数節点 v_j への枝があれば，v_i のレベルは v_j のレベルより小さい．レベルからブール変数の添字への 1 対 1 写像を**変数順序** π とする．BDD の形は，ブール関数だけでなく，変数順序にも依存する．

2.2 関数分解

ある関数 $f(x_1, \ldots, x_n)$ の関数分解は以下の形となる．

$$f = g(\alpha_1(X^B), \ldots, \alpha_t(X^B), X^F) = g(\vec{\alpha}(X^B), X^F), \quad (1)$$

ここで X^B と X^F は，$X^B \cup X^F = \{x_1, \ldots, x_n\}$ を満たす変数の集合である．X^B，X^F はそれぞれ，**bound set**, **free set** と呼ばれる．もし $X^B \cap X^F = \emptyset$ であれば，その分解は **disjunctive な分解**，さもなければ，**nondisjunctive な分解**と呼ばれる．また g は

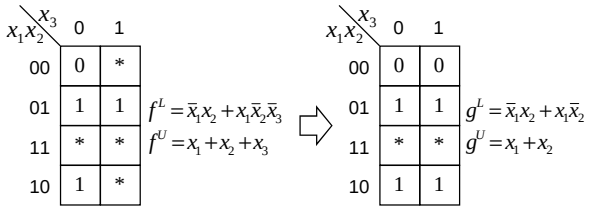

図 2: 依存変数最小化

分解の**像**と呼ばれ，本稿では $\alpha_1(X^B), \ldots, \alpha_t(X^B)$ を分解の**部分関数**と呼ぶことにする．

関数分解の基本的な概念は Ashenhurst [2] や Roth and Karp [3] によって研究された．最近，何人かの研究者 [5, 6, 7] が，BDD 表現に基づく関数分解のアルゴリズムを提案した．本稿では，Lai, Pedram and Vrudhula [7] らの以下の定義と命題を用いることにする．

定義 1 変数順序を π とする関数 f の BDD において，レベルが l よりも大きく，レベルが l 以下の節点からの枝を持つ節点の集合を $cut\_set(f, \pi, l)$ と表記する． □

命題 1 (disjunctive な分解) BDD の変数順序が π である n 変数関数に対して，$|cut\_set(f, \pi, l)| \leq 2^t$ であれば，(1)の形の分解が存在する．ここで，$X^B = \{x_{\pi(1)}, \ldots, x_{\pi(l)}\}$，$X^F = \{x_{\pi(l+1)}, \ldots, x_{\pi(n)}\}$ である． □

定義 2 $s \leq l$, $i \in \{0,1\}^{l-s+1}$ とする．変数順序を π とする関数 f の BDD において，レベル s からレベル l までの変数に値 i を代入した結果の関数を f_i としたとき，$cut\_set\_nd(f, \pi, l, s, i)$ は $cut\_set(f_i, \pi, l)$ を意味するものとする． □

命題 2 (nondisjunctive な分解) BDD の変数順序が π である n 変数関数に対して，$\forall i \in \{0,1\}^{l-s+1}$, $|cut\_set\_nd(f, \pi, l, s, i)| \leq 2^t$ であれば，(1)の形の分解が存在する．ここで，$X^B = \{x_{\pi(1)}, \ldots, x_{\pi(l)}\}$, $X^F = \{x_{\pi(s)}, \ldots, x_{\pi(n)}\}$ である． □

図 1に，cut_set および cut_set_nd と，それらの分解形との関係を示す．$|cut\_set(f, \pi, 3)| = 3 \leq 2^2$ なので，$f = g(\alpha_1(x_2, x_1, x_4), \alpha_2(x_2, x_1, x_4), x_3)$ となる形の分解が存在する．この形は，3 入力 LUT を 3 つ必要とする．また，$|cut\_set\_nd(f, \pi, 3, 3, 0)| = 2 \leq 2^1$ かつ $|cut\_set\_nd(f, \pi, 3, 3, 1)| = 2 \leq 2^1$ なので，$f = g(\alpha_1(x_2, x_1, x_4), x_4, x_3)$ となる形の分解が存在する．この形は，3 入力 LUT を 2 つしか必要としない．

2.3 依存変数最小化

2 つの関数の $f \cdot \overline{g} = 0$ なる関係を $f \leq g$ と書くことにする．不完全指定関数 $\hat{f} : \{0,1\}^n \to \{0,1,*\}$ ($*$ はドントケアを意味する) は，**interval** $[f^L, f^U]$ を用いてあらわすことができる．ここで，f^L と f^U は，$f^L \leq f^U$ を満たす完全指定関数である．1, 0, $*$ に対応する最小項の集合はそれぞれ，$\{X \mid f^L(X) = 1\}$, $\{X \mid f^U(X) = 0\}$, $\{X \mid f^L(X) = 0, f^U(X) = 1\}$ によって与えられる．本稿では，f の代わりに $\hat{f}$ という記法を用いて，その関数が不完全指定かもしれないということを表すことにする．不完全指定関数 $\hat{f}$ の依存変数集合は $sup(\hat{f}) = sup(f^L) \cup sup(f^U)$ となる．

完全指定関数 f は，もし $f^L \leq f \leq f^U$ であれば，不完全指定関数 $[f^L, f^U]$ に**両立 (compatible)** するといい，$f \prec [f^L, f^U]$ と表記する．同様に，不完全指定関数 $[g^L, g^U]$ は，もし $f^L \leq g^L \leq g^U \leq f^U$ であれば，不完全指定関数 $[f^L, f^U]$ に両立するという．

不完全指定関数の依存変数最小化は，文献 [11, 12, 13] で議論されている．ここでは依存変数最小化を “不完全指定関数 $[f^L, f^U]$ が与えられたとき，その依存変数集合 $sup(\hat{g})$ が最小であるような両立関数 $\hat{g}$ を 1 つ見つけよ” と定める．例えば，図 2にある不完全指定関数 $\hat{f} = [f^L = \overline{x_1}x_2 + x_1\overline{x}_2\overline{x}_3,\ f^U = x_1 + x_2 + x_3]$ を考える．その依存変数集合 $sup(\hat{f})$ は $\{x_1, x_2, x_3\}$ である．$*$ を 1 や 0 に置き換えることで，その依存変数集合 $sup(\hat{g})$ が $\{x_1, x_2\}$ であるような両立関数 $\hat{g} = [g^L = \overline{x_1}x_2 + x_1\overline{x_2},\ g^U = x_1 + x_2]$ を得ることができる．$\hat{g}$ に両立するすべての関数，$\overline{x_1}x_2 + x_1\overline{x_2}$ と $x_1 + x_2$ もまた，$\hat{f}$ に両立する．

本稿では，文献 [11] にある以下の定義と命題を用いる．

定義 3 $f(x_1, \ldots, x_n)$ をブール関数，R と S を $\{x_1, \ldots, x_n\}$ の部分集合とする．ここで，**disjunctive eliminant** $edis(f, R)$ と **conjunctive eliminant** $econ(f, R)$ を以下のように定義する．

$$
\begin{aligned}
&edis(f, \emptyset) = f \\
&edis(f, \{x_i\}) = f_{\overline{x_i}} + f_{x_i},\ i \in \{1, \ldots, n\} \\
&edis(f, R \cup S) = edis(edis(f, R), S) \\
&econ(f, \emptyset) = f \\
&econ(f, \{x_i\}) = f_{\overline{x_i}} \cdot f_{x_i},\ i \in \{1, \ldots, n\} \\
&econ(f, R \cup S) = econ(econ(f, R), S) \quad \square
\end{aligned}
$$

命題 3 $\hat{f} = [f^L, f^U]$ を不完全指定関数，E を $sup(\hat{f})$ の部分集合とする．もし，$edis(f^L, E) \leq econ(f^U, E)$ ならば，$\hat{f}' = [edis(f^L, E), econ(f^U, E)]$ は $\hat{f}$ に両立し，$sup(\hat{f}') = sup(\hat{f}) - E$ となる． □

命題 3によれば，依存変数最小化は，消去できる変数の最大部分集合 E を見つけることである．図 2では，$g^L = edis(f^L, \{x_3\})$，$g^U = econ(f^U, \{x_3\})$ である．もし，E を $\{x_1\}$ や $\{x_2\}$ としても，不等式 $edis(f^L, E) \leq econ(f^U, E)$ は満たされない．従って，$\{x_3\}$ は消去できる変数の最大部分集合である．

3 関数分解を用いた m-feasible 関数の生成

3.1 分解の形とコスト

回路中のすべての LUT は m ($m \geq 3$) 入力 1 出

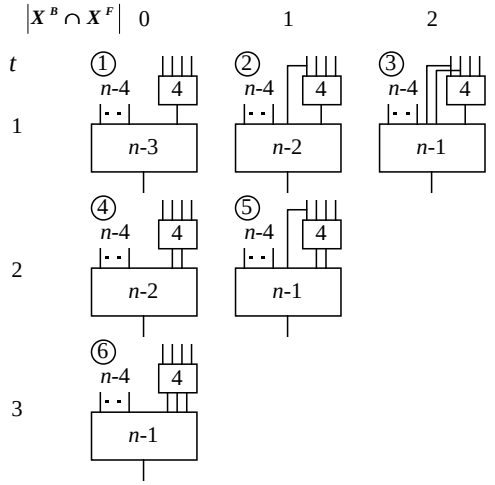

図 3: 分解の形とコスト

力のブール関数を実現できるものとする．我々の手法では，すべての関数の依存変数集合が m 以下になるまで，関数分解を繰り返す．関数分解により，ある関数をより小さな依存変数集合を持つ複数の新しい関数に分けることができる．関数分解は，多出力関数にではなく，単出力関数それぞれに適用する．共通の LUT をいくつかの関数で共有する方法については，4 章で議論する．

我々は，m-infeasible な関数に対し，bound set X^B の大きさが m であるような関数分解を試みる．従って，分解の部分関数 $\alpha_1(X^B), \ldots, \alpha_t(X^B)$ は m-feasible であり，これを LUT に割り当てることができる．もし分解の像 g が m-feasible ならば，これも LUT に割り当てることができる．それ以外の場合，g は新たな m-infeasible な関数となる．

ここでは，その像がもともとの関数よりも小さな依存変数集合を持つ分解のみを扱う．従って，不等式 $t + |X^F| < |X^B| + |X^F| - |X^B \cap X^F|$ が，分解可能性の条件として与えられる．この不等式と $|X^B| = m$ から，$t + |X^B \cap X^F| < m$ が導かれる．$t \geq 1$ かつ $|X^B \cap X^F| \geq 0$ なので，$m(m-1)/2$ 種類の分解の形を考えることができる．例えば，$m = 4$ のとき，図 3に示すような 6 種類の分解が考えられる．分解の形のコストは以下のように定める．すなわち，より少ない t を持つ分解はコストが少なく，同じ t を持つ分解では，より少ない $|X^B \cap X^F|$ を持つ分解の方がコストが少ない．図 3では，円の中の数字が分解のコストを示す．

関数 f が図 3のどのような形にも分解できないときは，ある変数 $x_i \in sup(f)$ を用いて展開 $f = \overline{x_i} \cdot f_{\overline{x_i}} + x_i \cdot f_{x_i}$ を行う．その結果，関数 $\overline{x_1} \cdot x_2 + x_1 \cdot x_3$ が 1 つの LUT で実現でき，$f_{\overline{x_i}}$ と f_{x_i} は新たな m-infeasible な関数となる．

3.2 分解可能性の判定

```
/* 最良解を保存するための大域変数 */
mincost;
minπ;
/* f は BDD、π は f の変数順序を示す */
/* Nin は bound set に含まれた変数の数 */
/* Nout は bound set から除かれた変数の数 */
bound_set(f, π, Nin, Nout) {
    if ( Nin = m or Nout = n−m ) { /* 終端条件 */
        cost = least_cost_decomposition(f, π);
        if ( mincost > cost ) {
            mincost = cost;
            minπ = π;
        }
    }
    else { /* 非終端 */
        /* レベル Nin+1 の変数を bound set に含める */
        bound_set(f, π, Nin+1, Nout);
        /* レベル Nin+1 の変数を bound set から除く */
        (newf, newπ) = jump_down(f, π, Nin+1, n−Nout);
        bound_set(newf, newπ, Nin, Nout+1);
    }
}
```

図 4: すべての bound set に対する分解可能性の判定

分解すべき関数 f に対し，我々は，大きさ m のすべての bound set X^B に対して分解を試み，最小コストの分解形を見つける．$|sup(f)|$ が n の場合，大きさ m のすべての bound set の数は ${}_nC_m$ である．命題 1と 2から，bound set の変数は，BDD 表現においてレベル 1 から m までに置かれなければならない．従って，異なった変数順序の ${}_nC_m$ 個の BDD を構築する必要がある．BDD の変数順序の変更は，$jump\_down$ という操作によって行う．$jump\_down(i, j)$ は，レベル i の変数をレベル $j (i < j)$ に移動し，レベル $i+1$ からレベル j までの変数のレベルを 1 ずつ減少させる．図 4に，大きさ m のすべての bound set に対して分解形とコストを調べるための再帰的アルゴリズムを示す．その計算は，$bound\_set(f, \pi, 0, 0)$ を呼び出すことから始まる．

3.3 エンコードとドントケア

最小コストの分解を与える bound set と free set が見つかったとしても，像 g と部分関数 $\alpha_1, \ldots, \alpha_t$ は一意には定まらない．cut_set や cut_set_nd のエンコードが異なると，g や $\alpha_1, \ldots, \alpha_t$ も異なるものとなる．エンコードに関する問題は，文献 [5, 8] で議論されている．しかし，これまでの我々の実装では，i 番目の要素に i の 2 進表現を割り当てるという単純な方法で cut_set や cut_set_nd をエンコードしている．例えば図 1において，$cut\_set(f, \pi, 3)$ の要素は $\alpha_2\alpha_1 = \{v_0 : 00, v_1 : 01, v_2 : 10\}$ とエンコードされる．

ここで，$\alpha_2\alpha_1$ は決して 11 にならないので，像 g の最小項のうち $g(1,1,0)$ と $g(1,1,1)$ は，ドントケアと

して扱うことができる．$|cut\_set(f,\pi,l)| = 2^t$ または $\forall i \in \{0,1\}^{l-s+1}$, $|cut\_set\_nd(f,\pi,l,s,i)| = 2^t$ でない限り，像 g がドントケアを持つように cut_set や cut_set_nd をエンコードすることができる．次の章で議論するブール再代入の技法は，SDC を用いているので，このようなドントケアを利用することができる．

4 依存変数最小化に基づくブール再代入

4.1 問題の定式化

3 章に示した方法だけでは，いくつかの関数の間で LUT を共有することはできない．文献 [1, 10] で議論されている再代入は，既に存在する関数が他の関数を実現するのに有用であるかどうかを調べるための技法である．例えば，$y_1 = x_1x_2 + x_1x_3 + x_4$ かつ $y_2 = x_2 + x_3$ としたとき，もし y_2 を y_1 に再代入すれば，y_1 は $y_1 = x_1(x_2 + x_3) + x_4 = x_1y_2 + x_4$ と表現でき，これは元の表現よりコストが少ない．

3.3 節で示したように，関数分解の像 g は不完全指定になることがある．その結果，他の関数を利用しようとする関数が，不完全指定関数になっている場合がある．このような場合は，他の関数を代入した結果の関数が，元の不完全指定関数に両立するようであれば，その再代入は成功したものとする．また，LUT 回路の合成では，依存変数集合の大きさがブール関数のコストの 1 つとみなすことができる．従って，ここでは，ブール再代入を以下のように定式化する．

問題 1 $\hat{f}$ をその依存変数集合が X であるような不完全指定関数とし，$h_1, \ldots, h_s$ をその依存変数集合が X', $(X' \subseteq X)$ であるような完全指定関数とする．$\hat{g}(h_1(X'), \ldots, h_s(X'), X'')$, $(X'' \subset X)$ が $\hat{f}$ に両立し，$sup(\hat{g})$ $(sup(\hat{g}) < sup(\hat{f}))$ が最小となるような関数 $\hat{g}$ を見つけよ． □

4.2 依存変数最小化に基づくアルゴリズム

y を $y = h(X')$ なる変数とする．もし y が他の関数に利用されるならば，$y \neq h(X')$ によって表される最小項の集合はドントケアとなる．このようなドントケアは，SDC (Satisfiability Don't Care) [14] と呼ばれる．ブール再代入は，SDC を考慮することで生じる不完全指定関数に対し，依存変数最小化を施すことにより行われる．

ここで，問題 1 を解くための我々の手順を示す．

1. $y_1, \ldots, y_s$ を $y_i = h_i(X')$, $i \in \{1, \ldots, s\}$ なる変数とする．X' と $y_1, \ldots, y_s$ に関する SDC として，$D = \sum_{i \in \{1,\ldots,s\}} y_i \neq h_i(X')$ を考える．
2. $\hat{f} = [f^L, f^U]$ とする．ここで，不完全指定関数 $\hat{g}_{INIT} = [f^L \cdot \overline{D}, f^U + D]$ を考える．$\hat{g}_{INIT}$ の依存変数集合は $\{y_1, \ldots, y_s\} \cup X$ である．D は，$h_i(X')$ を y_i $(\forall i \in \{1, \ldots, s\})$ に代入することで 0 になるので，$\hat{f} = \hat{g}_{INIT}(h_1(X'), \ldots, h_s(X'), X)$ である．
3. $\hat{g}_{INIT}$ に対して依存変数最小化を適用し，最小の依存変数集合を持つ両立関数 $\hat{g}$ を見つける．もし $sup(\hat{g}) < |X|$ であれば，その再代入は成功であり，それ以外の場合は失敗となる．E を依存変数最小化で消去した変数の集合とすると，$\hat{f} \succ \hat{g}(h_1(X'), \ldots, h_s(X'), X'')$, $X'' = X - E$ となる．

```
/* 最良解を保存する大域変数 */
minNsup;
minf^L;
minf^U;
/* 対象となる関数は [f^L, f^U] で与えられる */
/* Nsup はその関数の依存変数集合の大きさ */
/* elim は消去される変数の添字 */
support_min(f^L, f^U, Nsup, elim) {
    if ( elim ≤ 0 ) return; /* 終端条件 */
    if ( Nsup−elim ≥ minNsup ) return;
    newf^L = econ(f^L, x_elim);
    newf^U = edis(f^U, x_elim);
    if ( newf^L ≤ newf^U ) { /* x_elim を消去 */
        if ( Nsup−1 < minNsup ) {
            minNsup = Nsup−1;
            minf^L = newf^L;
            minf^U = newf^U;
        }
        support_min(newf^L, newf^U, Nsup−1, elim−1);
    }
    support_min(f^L, f^U, Nsup, elim−1); /* x_elim を含める */
}
```

図 5: 依存変数最小化

Lin [13] は，BDD 表現に基づき，すべての依存変数集合を見つけるアルゴリズムを示した．我々の手法も BDD 表現に基づいているが，ただ 1 つの最小の依存変数集合を見つけるだけである．図 5に，その再帰的アルゴリズムを示す．$sup(\hat{f}) = \{x_1, \ldots, x_n\}$ なる $\hat{f} = [f^L, f^U]$ に対し，その計算は $support\_min(f^L, f^U, n, n)$ を呼び出すことから始まる．このアルゴリズムでは，以下の 2 つの場合に探索空間を枝刈りすることができる．すなわち，もし $newf^L \leq newf^U$ を満たさなければ，この探索状態から両立関数を見つけることはできない．また，もし $Nsup{-}elim \geq minNsup$ であれば，依存変数集合の大きさが $minNsup$ より小さい両立関数をこの探索状態から見つけることはできない．

4.3 m-feasible 関数の再代入

ここで，関数分解だけでなくブール再代入も用いた LUT 回路の合成手順を示す．これは，すべての関数が m-feasible になるまで，以下のステップを繰り返す．

1. $\hat{f}_1, \ldots, \hat{f}_k$ を m-infeasible な関数，f_i を $\hat{f}_i$ に両立するある完全指定関数とする ($i \in \{1, \ldots, k\}$).
2. 3 章で述べた手順を用いて，$f_1, \ldots, f_k$ のそれぞれに対し最小コストの分解形を見つける．f_i の最小コスト分解で生じた部分関数を $\vec{\alpha}_i(X_i^B)$ とする．$\vec{\alpha}_i(X_i^B)$ は m-feasible 関数である．
3. すべての $i, j \in \{1, \ldots, k\}$ に対し，$\vec{\alpha}_i(X_i^B)$ を $\hat{f}_j$ に再代入してみる．ここで，$\vec{\alpha}_i(X_i^B)$ の $\hat{f}_j$ への再代入が成功するとき，$j \in success_i$ となるような $\{1, \ldots, k\}$ の部分集合 $success_i$ を定める．また，$\vec{\alpha}_i(X_i^B)$ を $\hat{f}_j$ に再代入したときに生じる関数を $\hat{g}_{ij}$ とする．
4. $gain_i = \sum_{j \in success_i} |sup(\hat{f}_j)| - |sup(\hat{g}_{ij})|$ を計算する．これは，もし $\vec{\alpha}_i(X_i^B)$ を用いたら，全体としてどれだけのファンイン数を減らせるかということを示すものである．そして，$\vec{\alpha}_1, \ldots, \vec{\alpha}_k$ から，$gain_b$ が最大となるような部分関数 $\vec{\alpha}_b(X_b^B)$ を見つける．
5. $\vec{\alpha}_b(X_b^B)$ に LUT を割り当て，すべての $j \in success_b$ に対し $\hat{f}_j$ を $\hat{g}_{bj}$ で置き換える．そのあと，もし $\hat{f}_1, \ldots, \hat{f}_k$ の中に m-feasible 関数があれば，それらに対しても LUT を割り当てる．

この手法では，回路の外部入力側から外部出力側に向かって LUT が割り当てられていく．従って，まだ LUT に割り当てられていない関数を簡単化するために，回路の SDC を用いることができる．この再代入の技法は，結果的にこのような SDC を扱っていることになる．ステップ 3 では，もし $i = j$ であれば，再代入が成功することは明らかである．しかし，ここでは実際に $\vec{\alpha}_i(X_i^B)$ を $\hat{f}_i$ に再代入して $\hat{g}_{ii}$ をつくり出す．これは，*cut_set* や *cut_set_nd* のエンコードによって生じたドントケアを利用するためである．

4.4 他の外部出力の再代入

他の外部出力関数を再代入することで，ある外部出力関数が簡単に実現できてしまうことがある．このような状況は，これまでに述べた方法では検出できないかもしれない．

我々は，4.3 節で述べた手順を行う前に，以下のような方法で，他の外部出力の再代入を行う．$f_1, \ldots, f_k$ を外部出力関数とする．すべての $i, j \in \{1, \ldots, k\}$ に対し，f_i を f_j に再代入してみる．実際には，以下の優先順位に従って再代入を行っている．これは，回路の段数がなるべく増えないようにするためである．

1. 再代入によって生じる関数が m-feasible である場合．このとき，その関数を実現するためには，ただ 1 つの LUT を加えるだけでよい．
2. f_i が m-feasible である場合．
3. その他の場合．

5 実験結果

我々は，これまでに述べてきた LUT 回路合成のプログラムを実現した．そのプログラムへの入力は，組合せ論理回路 (多段でも 2 段でもよい) である．入力された回路記述は，外部入力を変数とする外部出力の BDD 表現に変換され，m 入力 LUT の回路を合成する手続きへと移る．

表 1は，“回路” の列に列挙した MCNC ベンチマーク回路 [16] に対する結果を示している．‘LUT 数” と “段数” はそれぞれ，5 入力 1 出力 LUT の数と回路の段数を示している．“時間” は，SPARCstation 10/51 での CPU 時間を秒単位で表している．使用できる BDD 節点の数は 1,000,000 とした．

それぞれのベンチマーク回路に対し，3 種類の実験を行った．“再代入なし” にはブール再代入を実行していないときの結果を示す．この場合，複数の外部出力間で LUT を共有することはない．“再代入あり” には m-feasible 関数の再代入を行ったときの結果を示す．“外部出力の再代入” には他の外部出力の再代入も行った時の結果を示す．

それぞれの実験で，大きさ m のすべての bound set に対する関数分解を行っている．従って，大きな依存変数集合を持つ外部出力を含む回路の実行時間は，莫大になる傾向にある．いくつかの大きい回路，例えば C880 などに対しては，メモリ不足のため実行できなかった．

“再代入なし” と “再代入あり” を比べると，ブール再代入はいくつかの関数で LUT を共有することで LUT の数を減らし，多くの場合段数はそれほど増加していないので，とても有効であることがわかる．さらに，ブール再代入の実行時間 (そのほとんどは依存変数最小化に費やされている) は，それほど大きくない．“再代入あり” と “外部出力の再代入” を比べると，外部出力の再代入は LUT の数と実行時間を減少させるが，回路の段数を増やす傾向にあるということがわかる．我々の結果と他の LUT 回路合成プログラムの結果を比較するため，文献 [4, 15, 5] に見られる結果を表 1に掲載した．我々の手法は多くの回路に対して良い結果を出していることがわかる．

6 おわりに

関数分解と依存変数最小化に基づくブール再代入を用いた，LUT 回路の論理合成法について述べてきた．関数分解は LUT に割り当てる m-feasible 関数の候補を列挙するために用いられている．そのあと，それらの候補をすべての m-infeasible 関数に再代入してみることで，最もよい m-feasible 関数が決定される．

合成の各過程では，1 つの関数から作る m-feasible な関数の候補はただ 1 つである．より良質な LUT 回

表 1: 実験結果 (5 入力 1 出力の LUT)

回路			再代入なし			再代入あり			外部出力の再代入			[4]	[15]	[5]
名前	入	出	LUT 数	段数	時間	LUT 数	段数	時間	LUT 数	段数	時間	LUT 数		
5xp1	7	10	15	2	0.2	11	2	0.2	10	3	0.2	18	27	12
9sym	9	1	7	3	0.7	7	3	0.8	7	3	0.7	7	59	6
alu2	10	6	48	6	13.8	48	6	16.7	48	6	17.0	109	116	54
alu4	14	8	172	7	291.4	90	7	234.7	56	9	38.4	55	195	
apex4	9	19	374	5	76.7	374	5	230.8	374	5	240.5	412	558	
apex6	135	99	192	6	623.9	161	4	755.0	155	8	208.7	182	212	204
apex7	49	37	120	5	164.1	61	5	126.9	54	5	12.3	60	64	56
b12	15	9	16	3	0.3	16	3	0.3	16	3	0.4			
b9	41	21	53	4	11.3	39	5	12.9	37	5	6.5	39		35
clip	9	5	18	3	3.7	11	3	3.8	14	8	2.6	28		
cordic	23	2	15	5	45.9	9	4	48.0	10	6	27.6			
count	35	16	52	4	5.3	31	6	11.3	31	6	11.0	31	31	32
duke2	22	29	175	7	432.2	155	7	489.4	150	8	451.8	110	120	
f51m	8	8	12	3	0.3	10	3	0.3	8	4	0.1	17		12
misex1	8	7	12	2	0.2	10	2	0.2	10	4	0.2	11	19	11
misex2	25	18	40	3	1.5	36	3	1.8	36	4	2.3	28		29
misex3	14	14	195	9	503.4	213	9	670.4	120	13	136.6			
misex3c	14	14	107	9	132.4	99	9	131.1	92	11	101.6			
rd73	7	3	8	2	0.1	6	3	0.2	6	3	0.2	6		7
rd84	8	4	12	3	0.4	7	3	0.5	8	5	0.3	10	73	12
sao2	10	4	23	4	7.4	21	4	8.2	21	4	7.6	28		46
t481	16	1	5	3	4.4	5	3	4.5	5	3	4.5			
vg2	25	8	44	5	217.5	21	5	253.0	17	4	4.2	20	21	
z4ml	7	4	6	2	0.2	5	2	0.2	4	2	0.1	5	6	5
合計			1721	105	2537.3	1446	106	3001.2	1289	132	1275.4			

路を合成するためには，より多くの候補を列挙するための手法が必要となるであろう．本稿で提案したブール再代入の技法は，それほど時間のかかるものではなく，多くの候補の中から効率良く共通の LUT を見つけ出すことができると思われる．

大きな依存変数集合を持たない関数に対しては，それほど時間をかけずにすべての bound set に対して関数分解を行うことができ，LUT に割り当てるのにふさわしい m-feasible 関数を見つけ出すことができる．しかし，大きな依存変数集合を持つ関数に対しては，その関数分解の実行に時間がかかり，時にはメモリ不足で実行できないこともある．従って，大規模回路に対しては，全解探索を避けるようなヒューリスティックが必要である．

謝辞

この研究を始めるきっかけを与えて下さった NTT 情報通信研究所の小栗清さんに感謝致します．

参考文献

[1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. CAD*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[2] R. L. Ashenhurst, "The Decomposition of Switching Functions," in *Proc. of an International Symposium on the Theory of Switching*, Apr. 1957.

[3] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM journal*, pp. 227–238, Apr. 1962.

[4] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," in *ICCAD*, pp. 564–567, Nov. 1991.

[5] S. Chang and M. Marek-Sadowska, "Technology Mapping via Transformations of Function Graphs," in *ICCD*, pp. 159–162, Oct. 1992.

[6] T. Sasao, "FPGA design by generalized functional decomposition," in *Logic Synthesis and Optimization* (T. Sasao, ed.), pp. 233–258, Kluwer Academic Publishers, 1993.

[7] Y.-T. Lai, M. Pedram, and S. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *30th DAC*, pp. 642–647, June 1993.

[8] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum Functional Decomposition Using Encoding," in *31st DAC*, pp. 408–414, June 1994.

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

[10] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita, "Boolean Resubstitution With Permissible Functions and Binary Decision Diagrams," in *27th DAC*, pp. 284–289, June 1990.

[11] F. M. Brown, *Boolean Reasoning: The Logic of Boolean Equations.* Kluwer Academic Publishers, 1990.

[12] M. Fujita and Y. Matsunaga, "Multi-level Logic Minimization based on Minimal Support and its Application to the Minimization of Look-up Table Type FPGAs," in *ICCAD*, pp. 560–563, Nov. 1991.

[13] B. Lin, "Efficient Symbolic Support Manipulation," in *ICCD*, pp. 513–516, Oct. 1993.

[14] H. Savoj, R. K. Brayton, and H. J. Touati, "Extracting Local Don't Cares for Network Optimization," in *ICCAD*, pp. 514–517, Nov. 1991.

[15] R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," in *28th DAC*, pp. 227–232, June 1991.

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