設計自動化 68－10
(1993. 10. 28)

関数分解を用いたFPGAの設計法

笹尾 勤, 岡村康一郎
九州工業大学 情報工学部 電子情報工学科
〒820 飯塚市大字川津 680-4

あらまし 一般化した分解理論を用いて, テーブル参照型FPGAを設計する方法を述べる. FPGAとしては, 5入力のLUT(Look-up table)を用いたものを対象とする. 主な項目は, 1) 一般化した関数分解の定理, 2)BDDを用いた関数分解の方法, 3)5入力LUTを用いた論理関数の構成方法, 4)MCNCベンチマーク関数を実現した結果を示す.

和文キーワード FPGA, 関数分解, BDD, 対称関数, 論理合成

A Design Method for Field Programmable Gate Arrays using Functional Decomposition.

Tsutomu SASAO and Koichiro OKAMURA
Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu Iizuka,820,Japan

Abstract A design method of Field Programmable Gate Arrays (FPGAs) by functional decomposition is presented, where FPGAs are consist of 5-input LUT, i.e., table look-up type FPGA. Main subjects are 1) Generalized functional decomposition theory, 2) Decomposition using BDDs; 3) Realization of functions by 5-input LUTs; and 4) Experimental results using MCNC benchmark functions.

英文 key words Field programmable gate array, Functional decomposition, BDD, Symmetric function, Logic synthesis

1 はしがき

Ashenhurst の論理関数分解理論 [1] が発表されてから 35 年以上経た. 分解理論はその後多くの研究者によって発展させられ, その研究成果は接点回路網の設計に利用された. しかし, 分解理論を LSI の設計に利用し始めたのは最近のことである [19, 7].

本論文では, 古典的な分解理論を一般化し, FPGA 設計に適用することを考察する. 古典的な分解理論を大規模回路 (ここでは, 入力数 $n = 10 \sim 100$, 出力数 $m = 10 \sim 100$ 程度を考える) の合成に適用する際, 次の二つの問題が生じる.

1) 第一の問題は, 計算時間とメモリである. 入力変数の個数を n とする. 可能な分解の個数は, ほぼ, 2^n個. また, 操作すべき分解表の大きさも 2^n となる. 従って, n が大きくなると, 必要な計算時間やメモリは指数関数的に増大する. そのため, 分解のアルゴリズムを率直にプログラムしたものを計算機で実行するのは, $n = 10 \sim 15$ 程度が限界である.

2) 第二の問題は, 分解の有効性である. 古典的な分解理論では, 図1で示すような回路構造を仮定しており, 部分回路 Hの出力線は1本である. 例えば, 10 入力 1 出力の関数で, 図1の回路で実現可能な関数の個数は 2^{95}であり, これは, 10 変数関数全体の個数 2^{1024}に比べると極めて少ない. つまり, 古典的な分解理論を現在の問題に適用することは非常に困難である.

以上の二つの問題に対して, 本論文では以下のような解決策を用いる: まず, 第一の問題に対しては, 分解表を BDD(Binary Decision Diagram[4]) で表現する. BDD を用いると大規模な関数をコンパクトに表現できる場合が多く, 分解を能率よく実行できる. 第二の問題に対しては, 図2のような, 部分回路 Hの出力線が複数である回路構造を仮定し, 分解理論を一般化する [6, 20]. 例えば, 10 変数関数で, 図2の回路構造を考えると, 実現可能な関数の個数はおよそ 2^{595}となり, 図1の構造に比べ回路能力は飛躍的に増大する.

本論文では, 一般化した分解理論を用いて, テーブル参照型FPGA(以下では単に FPGA と略記) を設計する方法を述べる. 設計対象は, 組合せ論理回路のみを考える. FPGA は, 多段論理回路を直接実現でき, 設計は FPLA に比べ複雑である [3]. 現在, 多くのグループが論理設計法を開発中であり, 設計法として種々の方法が提案されているが, いずれもまだ未熟である.

2 FPGA とそのモデル

FPGA は, 図3に示すように, アレイ状に並べた論理ブロックを縦横に走る配線領域で相互接続できるようにしたものである. 論理素子は, LUT(Look up table) と呼ばれ, 任意の k変数関数を実現する. $k = 4 \sim 6$ のものが多く, LUT と配線はスタティック RAM を用いてプログラムする. 実際の論理ブロックは, 図4のようにかなり複雑で, 多出力関数を実現し, フリップ・フロップを含む. 配線接続にはトランスミッションゲートを用いており, 接続時の ON 抵抗が高いため, 配線の遅延時間が大きい. 繰り返しプログラムでき, システム動作中にもプログラムを変更できるのが大きな特徴である. 配線部の遅延が, 論理部の遅延よりも大きくなることが多い.

実際の応用では, 論理関数が規定の FPGA に納まればよく, 論理段数は短い方がよい. 設計の良さを示す評価関数としては, 必要なモジュール数と論理段数を用いる. ただし, 遅延時間は, 配置配線によって大きく変化する. モジュール数と論理段数を同時に減らすのは困難である. 実際の FPGA では, 配線部分の遅延時間が, 論理部分の遅延時間より大きくなりがちなので, 配置配線には特に配慮が必要である.

FPGA の論理設計は, 与えられた論理関数を k入力の LUT のみを用いて実現する問題として定式化できる. ここで, 各 LUT は, 任意の k変数論理関数を実現する. FPGA 設計の基本的方法として, 次の三つの方法が知られている.

テクノロジマッピング法 [3]

与えられた論理関数を既存のツールで, AND, OR 等のゲートの多段論理回路に変換し, 次に各ゲートを LUT にマッピングする方法である. 多段論理回路の各ゲートのファンインが k以下の場合には, 容易に FPGA に変換できる. 現在発表されているツールは, このタイプが最も多い. この方法では, 多段論理合成ツールや既存の回路など, 既存の資産を利用できる利点があるが, 必ずしも最適な回路が得られるわけではない.

論理式分解法 [11, 17]

これは, 与えられた論理関数を論理式に変換し, それを, LUT にマッピングする方法である.

関数分解法 [21, 23, 13].

これは, 与えられた論理関数を直接 LUT にマッピングする方法であり, 本論文でも用いている. 論理関数の分解の検出には BDD を利用する. この方法は, 最近発表されたばかりなので, まだ実験データが少ないが, 入力変数の少ない場合には, 他の方法に比べ LUT の少ない回路を生成できる.

3 関数分解による多段論理回路設計 [21]

本節では, 一般化した分解理論について述べる.

定義 1 入力変数を $X = (x_1, x_2, \cdots, x_n)$ とする. Xの変数の集合を $\{X\}$ で表わす. $\{X_1\} \cup \{X_2\} = \{X\}$ かつ $\{X_1\} \cap \{X_2\} = \phi$のとき, $X = (X_1, X_2)$ を Xの分割 (partition) という. Xの変数の個数を $d(X)$ で表わす.

定義 2 完全記述関数 $f(X)$, Xの分割 $X = (X_1, X_2)$ に対して, 2^{n_1}列 2^{n_2}行の表で, 各行, 各列に 2 進符号のラベルをもち, その要素が fの対応する真理値であるような表を, fの分解表 (decomposition chart) という. ここで, $n_1 = d(X_1), n_2 = d(X_2)$ であり, 列, 行はそれぞれ n_1, n_2ビットの全てのパターンを有する.

例 1 表 1は 5 変数関数の分解表の例である. ただし, Xの分割を $X = (X_1, X_2)$ とし, $X_1 = (x_1, x_2, x_3)$, $X_2 = (x_4, x_5)$ である.

定義 3 分解表の異なる列パターンの個数を**列複雑度** (column multiplicity) と呼びμで表わす.

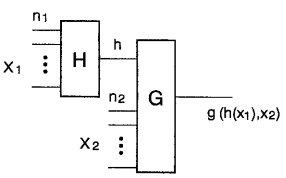

図 1: 単純な関数分解

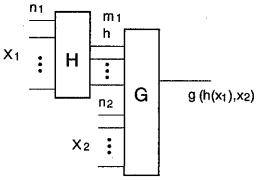

図 2: 一般化した関数分解

表 1: 分解表

		X_1							
		0	0	0	0	1	1	1	1
		0	0	1	1	0	0	1	1
		0	1	0	1	0	1	0	1
X_2	00	0	1	0	0	0	1	1	0
	01	1	1	1	1	1	1	1	1
	10	1	0	1	1	1	0	0	1
	11	0	1	0	0	0	1	1	0

表 2: 分解表

		X_1			
		0	0	1	1
		0	1	0	1
X_2	000	0	0	0	1
	001	1	1	1	1
	010	1	1	1	0
	011	0	0	0	1
	100	1	0	1	0
	101	1	1	1	1
	110	0	1	0	1
	111	1	0	1	0

$\gamma = \min(2^{n_1}, 2^{2^{n_2}})/\mu$を分解の利得という.

例 2 表 1の分解表では, $n_1 = 3$, $n_2 = 2$, $\mu = 2$ である. また, $\gamma = \min(2^3, 2^4)/2 = 8/2 = 4$ である.

列複雑度は, 分解表に対して定義され, 入力変数の分割 $X = (X_1, X_2)$ に依存する. また, 完全記述関数を対象にしているので, 分割が決まれば列複雑度は一意的に定まる.

定義 4 分解表で, 互いに異なる列パターンが表す論理関数を $\Phi_i(X_2)(i = 0, 1, \cdots, \mu - 1)$ とし, 関数$\Phi i(X_2)$ に対する X_1の入力の集合が表す論理関数を$\Psi_i(X_1)$ とする.

分解表の定義から, Ψ_iは, 次のような性質を持つ.

$$\Psi_i \cdot \Psi_j = 0 \ \ (i \neq j), \ \ \bigvee_{i=0}^{p-1} \Psi_i = 1$$

また, 関数$\Phi i = f(\mid X_1 = \Psi_i)$ と表現できる. ここで, $f(\mid X_1 = \Psi_i)$ は, 関数fにおいてX_1の値をΨ_iのいずれかの最小項に固定したものである.

例 3 表2の分解表では, $X_1 = (x_1, x_2), X_2 = (x_3, x_4, x_5)$ である. また $n_1 = 2$, $n_2 = 3$, $\mu = 3$, $\gamma = \min(2^2, 2^8)/3 = 4/3 = 1.33$ である.

$$\Phi_0(X_2) = \bar{x}_3\bar{x}_4x_5 \vee \bar{x}_3x_4\bar{x}_5 \vee x_3\bar{x}_4\bar{x}_5 \vee x_3\bar{x}_4x_5 \vee x_3x_4x_5$$
$$\Phi_1(X_2) = \bar{x}_3\bar{x}_4x_5 \vee \bar{x}_3x_4\bar{x}_5 \vee x_3\bar{x}_4x_5 \vee x_3x_4x_5$$
$$\Phi_2(X_2) = \bar{x}_3\bar{x}_4\bar{x}_5 \vee \bar{x}_3\bar{x}_4x_5 \vee \bar{x}_3x_4x_5 \vee x_3\bar{x}_4x_5 \vee x_3x_4\bar{x}_5$$
$$\Psi_0(X_1) = \bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2$$
$$\Psi_1(X_1) = \bar{x}_1x_2$$
$$\Psi_2(X_1) = x_1x_2$$

定義 5 p 値出力関数 h を, 以下のように定義する.
$h(X_1) = i \rightleftharpoons \Psi_i(X_1) = 1 \ (i = 0, 1, \cdots, p-1)$

定義 6 関数 $g(Y) : P \times B^{n_2} \rightarrow B, P = \{0, 1, \cdots, p-1\}$, ただし, $f(X) = g(h(X_1), X_2)$ とする.

定義 7 定義4~6の関数 f, g, h に対して, $f(X) = g(h(X_1), X_2)$ を関数 fの分解という. $\gamma > 1$のとき, この分解は自明でない (non-trivial) という. 自明でない分解を持つとき, fは分解可能 (decomposable) であるという.

古典的な分解では図1の回路実現を対象にしており, $\gamma = 2^{n_1-1}$のときのみ分解可能である. ここでは, 図2に対応する分解を考えており, $\gamma > 1$ ならば分解可能である. また, 関数 h を実現する回路 Hの出力線は一般に複数である. $\gamma \geq 2$ のとき, 回路 Hの出力線は, 入力線より少なくてよい. また, $1 < \gamma < 2$ のときには, 回路 Hの出力線数は減らないが, Hの出力には, 決して生じない組合せが $2^{n_1} - \mu$個存在するようにできる. 従って, 回路 G を設計する際, それらの組合せはドントケアとして利用できる. そのため, 利得が最大の分割を求めることが重要である.

定理 1 分解 $f(X) = g(h(X_1), X_2)$ の列複雑度がμのとき, fは図2の回路で実現可能である. ただし, m_1は$\mu \leq 2^{m_1}$となる最小の整数である [6].
(証明) 定義4~6より, $f(X) = g(h(X_1), X_2)$ となる関数
$g : M \times B^{n_2} \rightarrow B$, および $h : B^{n_1} \rightarrow B$,
が存在する. ここで, $B = \{0, 1\}$, $M = \{0, 1, \cdots, \mu - 1\}$, $n_1 = d(X_1)$, $n_2 = d(X_2)$ であり, μは列複雑度を示す. 回路 Hで m_1個の関数 $h_1, h_2, \cdots, h_{m_1}$ を実現可能である. ただし, 二進ベクトル $(h_1, h_2, \cdots, h_{m_1})$ が Mの値を表現するものとする. また, 回路 G で関数 gを実現可能である. よって, 図2の回路で関数 fを実現可能である. (証明終)

例 4 表2の分解表について考える.
Hと G に対する関数を表3と図5に示す. 列複雑度が3なので, Hの出力数は2となる. Hは(1,1)となる出力を生じないので, G を設計する際, この入力はドントケアとして利用できる. 図5では, ×印を示した部分がドントケアで, これに対する回路を図6に示す.

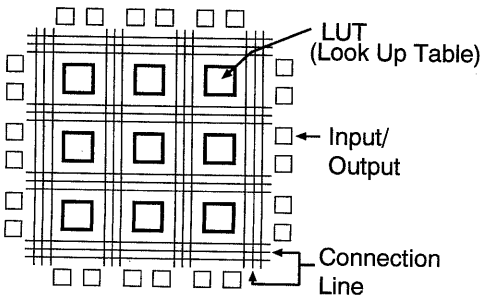

図 3: FPGA

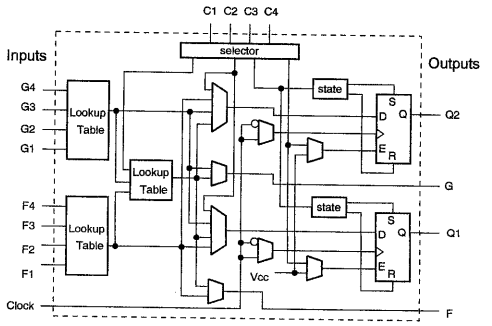

図 4:

表 3: Hの関数

x_1	x_2	h	h_1	h_2
0	0	0	0	0
0	1	1	0	1
1	0	2	1	0
1	1	0	0	0

4 BDD を用いた関数分解法 [21]

n 変数関数の分解を検出するために分解表を用いると 2^nの記憶容量が必要である. また, 分割の方法は 2^n通 り存在する. そのため, n が大きい場合には, 率直な方法で分解を実行するのが困難である. ここでは, BDD を用いて, 分解を検出する方法を示す.

定義 8 ROBDD(Reduced Ordered Binary Decision Diagram) とは完全な二分判定図から, 同形な部分グラフと冗長な節点を除去したものである. QROBDD(Quasi-Reduced BDD) とは, 根から終端節点に至るどのパスも全ての変数を含み, かつ, 同一レベルに同形な部分グラフを含まない BDD である.

定義 9 BDD において, 根からある節点までたど り着 く条件を表す論理関数をその節点のパス関数とい う.

補題 1 変数の分割を $X = (X_1, X_2)$ とし, 関数 fの QROBDD が図 7のよ うに二つのブロ ックに分割されているとする. このとき下部ブロ ック中にある節点の内, 上部ブロ ックと枝で接続している節点のパス関数を, それぞれ, $q_i (i = 1, \cdots, k)$ とするとき, $q_i \cdot q_j = 0, \vee q_i = 1$ が成立する.

(証明) n 変数の完全二分判定木を考え, 図 7の形に書 く. このとき, $q_1, q_2, \cdots, q_k$は, $n_1 = d(X_1)$ 変数の論理最小項を表現する. これは, 補題の条件を満足している. この BDD において, 同じ関数を表現する部分木を併合するとい う操作を繰 り返しても, この性質は保存される. (証明終)

定理 2 変数の分割を $X = (X1, X2)$ とし, 関数 fの ROBDD が図 7のよ うに二つのブロ ックに分割されているとする. このとき, 下部ブロ ックにある節点の内, 上部ブロ ックと枝で接続している節点数を k とすると, kは関数分解 $f = g(h(X_1), X_2)$ の列複雑度μに等しい.

(証明) ROBDD のかわ りに QROBDD を考える.

1) 下部ブロ ック中の節点で上部ブロ ックと枝で接続している節点のパス関数を $q_1(X_1), q_2(X_1), \cdots, q_k(X_1)$ とする. QROBDD の定義よ り, $qi \cdot qj = 0\ (i, j = 1, 2, \cdots, k, i \neq j)$ である. また, $\boldsymbol{a}, \boldsymbol{b} \in qi$ なる入力,$\boldsymbol{a}, \boldsymbol{b}$ に対して, $g(h(\boldsymbol{a}), X_2) = g(h(\boldsymbol{b}), X_2)$ が成立する. これよ り$k \leq r$を得る.

2)n 変数の完全二分判定木を考え, 図 7のよ うに二つに分割する. 下部のブロ ックを縮約して得られる BDD を考える. このとき, 下部のブロ ックのの節点の うち, 上部のブロ ックと枝で接続している節点の数を rとすると, 列複雑度の定義によ り, $r = \mu$である. この BDD をさらに縮約して得られるものが ROBDD であるが, 節点数は増えることはない. 従って, $k \leq \mu$ を得る. ROBDD は QROBDD から, 冗長な節点を除去したものであるが, 二つの BDD の kの値は変わらない. 以上よ り定理を得る. (証明終).

rは BDD の幅とも呼ばれている [14].

5 FPGA 設計アルゴ リズム

与えられた n 変数関数を最小個の LUT を用いて実現する一般的方法は知られていない. ここでは, LUT 回路網を関数分解を利用して構成する手法について述べる. また, 5 入力 LUT を多数集積した FPGA を用いて, 任意の論理関数を実現する方法を考える.

5.1 LUT 回路網の構成法

まず, 図 2で, H の出力数が 3 以下の場合の関数分解を試みる. これは回路の出力側からの分解である. この分解を可能な限 り行な う. 残 った部分の回路 G に対しては, 図 8のよ うな樹枝状回路構造で実現する. これは回路の入力側からの分解である. この際, 出力部の LUT の制御変数は 2 個に制限されている. 図 8の回路では配線が容易であ り, 遅延時間もさほど大き くならない.

例えば, 7 変数関数 $f(x_1, x_2, \cdots, x_7)$ が図 8の回路構造で実現可能であるための条件は, $f = g(h_1, h_2, h_3, x_1, x_2)$, $h_1 = h_1(x_3, \cdots, x_7)$, $h_2 = h_2(x_3, \cdots, x_7)$, $h_3 = h_3(x_3, \cdots, x_7)$, と表現できることである. そのための必要十分条件は, 『分解 $f = g(h(x_3, x_4, \cdots, x_7), x_1, x_2)$ の列複雑度が 8 以下』である. 上の条件を満たさない場合でも, 変数を適当に置換することによ って, 列複雑度を 8 以下にできる場合もある. ここでは, 5 入力 LUT および 3 入力 LUT を用いて与えられた論理関数を LUT の樹枝状回路網で実現するために, 以下の問題を考える.

問題 1 5 入力 LUT および 3 入力 LUT を用いて与えられた論理関数を図 2の形で分解した後, 図 8のよ うな回路で実現する. 関数分解が全 く不可能な場合は, 与えられた関数 fを $f = (h_0\bar{x}_2 \vee h_1x_2)\bar{x}_1 \vee h_2x_1\bar{x}_2 \vee h_3x_1x_2$ と変形し, 図 9の回路よ うに 3 入力 LUT と 5 入力 LUT で関数を実現する. ここで, 小さなブロ ック は $g = h_0\bar{x}_2 \vee h_1x_2$ を実現するマルチプレ クサで 3 入力 LUT で実現できる. また, 3 入力 LUT のコス トは 5 入力 LUT のコス トの半分と仮定する. 必要な LUT 数をなるべ く少な くする.

このとき, 次のよ うな設計方策を用いる.

1. $\mu(i, j) = 2^0 = 1$ のとき, 関数は X_1には依存せず, 図 10(a) の回路で実現する.
2. $\mu(i, j) = 2^1 = 2$ のとき, 図 10(b) の回路で実現する.
3. $3 \leq \mu(i, j) \leq 2^2 = 4$ のとき, 図 10(c) の回路で実現する.
4. $5 \leq \mu(i, j) \leq 2^3 = 8$ のとき, 図 10(d) の回路で実現する.
5. $\mu(i, j) \geq 9$ のとき図 10(e) の回路で実現する.

また, 開発当初は図 10に示す通 り, $d(X_2) = 2$ に固定して関数分解を行な った. この段数は, $n/2$ 程必要となる。$d(X_2) = 3$ で$\mu \leq 4$ と分解できるときは制御変数の数を 3 個とする. この方策によ って, 段数削減をねら う.

	(h_1, h_2)			
(x_3, x_4, x_5)	00	01	10	11
000	0	0	×	1
001	1	1	×	1
011	0	0	×	1
010	1	1	×	0
100	0	1	×	1
101	1	0	×	0
111	1	1	×	1
110	1	0	×	0

図 5: G に対するマ ップ

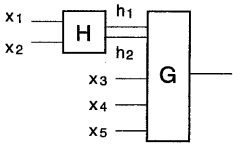

図 6: 表 2の関数の実現

本手法では, 変数の分解順序によって必要なLUT数が大きく変化する. 列複雑度の大きい変数から分解を行なうと, LUT数が増加するので, なるべく列複雑度の小さい変数から分解を行なう. 多出力関数の場合, 各関数によって同じ変数ペアの列複雑度が異なるのだが, BDDを分割した場合の上部ブロックと枝で接続している接点数を目安とした.

これらの問題を考慮に入れたアルゴリズムを以下に示す.

アルゴリズム 1 1. 関数 fが与えられたとき, 全ての対 $X_1 = (x_i, x_j)$ に対して列複雑度を求める. 列複雑度が3以下の対を組み合わせて5組 $(i_1, i_2, i_3, i_4, i_5)$ $(1 \le i_1 < i_2 < i_3 < i_4 < i_5 \le n)$ を作成する. $X_1 = (x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}, x_{i_5})$ に対する列複雑度 $\mu(i_1, i_2, i_3, i_4, i_5)$ を求める. 最小の列複雑度をμとする.

2. $\mu \le 8$ならば, (h_0, h_1, h_2) を新たな変数として, 元の関数 fを表現する. $3 \le \mu \le 4$ ならば, (h_0, h_1) を新たな変数として, 元の関数 fを表現する. $\mu = 2$ ならば, (h_0) を新たな変数として, 元の関数 fを表現する. $\mu = 1$ ならば, fは X_1には依存しない.
3. fに同じ操作を繰り返す. ただし, 分解の利得が4.0以下の時は分解しない.
4. 上の分解を行い, 残った関数を新たな fとする.
5. 変数の全ての対 (i, j) $(1 \le i < j \le n)$ に対する列複雑度$\mu(i, j)$を求める.
6. fの変数の個数を n とする. 節点数が n で列複雑度が8以下の節点対 (i, j) 間のみに枝があるようなグラフを G とする. それぞれの枝に列複雑度を重みとして持たせる.
7. Gの最大マッチングを求める.
8. 分解する変数の順序を決める. 列複雑度が最小となる変数対 (i, j) が出力部に来るようにする.
9. 列複雑度に従って, 図10(a)~ (e) の実現法を用いる. ただし, (b)~(d) に対しては, 2変数以上の分解を試みる. 各列のパターンに対して, kビットの二進ベクトルを割り当てる (これを符号割当という). $h_0, h_1, \cdots, h_{k-1}$を分解により得られた部分関数とする.
10. k個の出力関数 $(h_0, h_1, \cdots, h_{k-1})$ をステップ5からステップ9まで実行し, 各部分関数において, 依存する変数の個数が5以下になれば, その関数に対する分解を停止する.

5.2 部分対称関数の構成法

定義 10 論理関数の変数の部分集合 X_1において, その内部で変数を置換しても関数が不変のとき, その関数は変数の集合 X_1において部分対称 (partially symmetric) であるという.

補題 2 関数 fが X_1において部分対称であれば, 分解 $f(X) = g(h(X_1), X_2)$ の列複雑度は高々$n_1 + 1$である. ここで, $n_1 = d(X_1)$である.

実用上重要な演算回路は, 部分対称なものが多く, 一般化した分解理論が適用できる.

補題 3 関数 fが $f(Y, x_1, x_2) = f(Y, x_2, x_1)$ を満たすとき, fは図11の回路網で実現可能である.

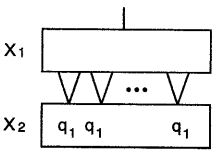

図 7: BDD の分割

定理 3 任意の n 変数対称関数 $(n = 2k + 5)$ は図8の樹枝状回路網で実現できる. また, LUTは, 高々$\frac{1}{2}(3^{k+1} - 1)$ 個あれば十分である.
(証明) 補題2より, 対称関数が図8の回路網で実現できることは, 明かである. LUTがk段の樹枝状回路の素子数は $1 + 3^1 + 3^2 + \cdots + 3^k$ である. (証明終)

例 5 9変数完全対称関数 $\mathrm{SYM9}(x_1, x_2, \cdots, x_9) = S^9_{\{3,4,5,6\}}$を5入力LUTを用いて構成してみよう. 完全対称関数なので, $X_1 = (x_1, x_2, x_3, x_4, x_5), X_2 = (x_6, x_7, x_8, x_9)$ と分割した場合の分解表の列複雑度は高々6である. 従って, 図2の回路構造で実現できる. たとえば, 図12の初段で, $h_1 = S^5_{\{1,3,5\}}, h_2 = S^5_{\{2,3\}}, h_3 = S^5_{\{4,5\}}$ の三つの関数を実現すると,

$$\begin{aligned}\mathrm{SYM9} \;=\; & \bar{h}_1 \cdot \bar{h}_2 \cdot \bar{h}_3 \cdot S^4_{\{3,4\}} \vee \bar{h}_1 \cdot \bar{h}_2 \cdot h_3 \cdot S^4_{\{2,3,4\}} \\ & \vee \bar{h}_1 \cdot h_2 \cdot \bar{h}_3 \cdot S^4_{\{1,2,3,4\}} \vee \bar{h}_1 \cdot h_2 \cdot h_3 \cdot S^4_{\{0,1,2,3\}} \\ & \vee h_1 \cdot \bar{h}_2 \cdot \bar{h}_3 \cdot S^4_{\{0,1,2\}} \vee h_1 \cdot \bar{h}_2 \cdot h_3 \cdot S^4_{\{0,1\}}\end{aligned}$$

と表現できる. ここで, S^4_Aは, 変数 (x_6, x_7, x_8, x_9) の対称関数である. 従って, SYM9は図12に示すように, 7個のLUTで実現できる. (例題終)

5.3 多出力関数の実現

多出力関数の場合, 各出力関数を独立に実現すると, 同じ部分回路が2回以上できる場合がある. これを避けるため, 以下の手法を用いる. m 出力関数の出力部を表す m 値の変数を zとする.

$$F(i, x_1, x_2, \cdots, x_n) = f_i(x_1, x_2, \cdots, x_n) \;\; (i = 0, 1, \cdots, m-1)$$

なる特性関数 $F(Z, x_1, x_2, \cdots, x_n)$ のBDD(図13) を考える. このようにすると, 各出力で, 同一の部分関数を共有できる.

6 実験結果

与えられた関数を, 第5節のアルゴリズム1を用いて実現するプログラムをC言語で開発した.

種々のベンチマーク関数を実現した結果を表4に示す. in/outは, それぞれのベンチマークの入出力数を, Lは LUT回路網の段数を, LUTsは5入力LUTの数を表している. 計算機は HP9000/720(64MB: メインメモリ) を使用した. また, MIS-PGA2[16], Chortle-d[10] dagmap[25] の結果を示す.

入力数が小さい場合には, 他手法と比べて良い結果が得られた. しかし, 入力数が大きい場合には, 段数と LUT数のどちらも増える結果となった. この原因は, 図8のように同一段の入力を統一したことであろう. こうすると配線が規則的であるという利点がある. また, 多出力関数を同時に分解するため, アルゴリズム1のステップ1~3の部分がうまく適用できなかったことも理由の一つである.

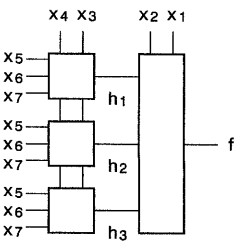

図 8: LUT 回路網

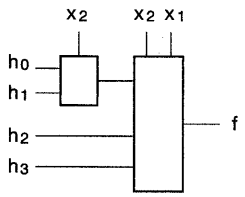

図 9: 関数分解不可能な場合の実現法

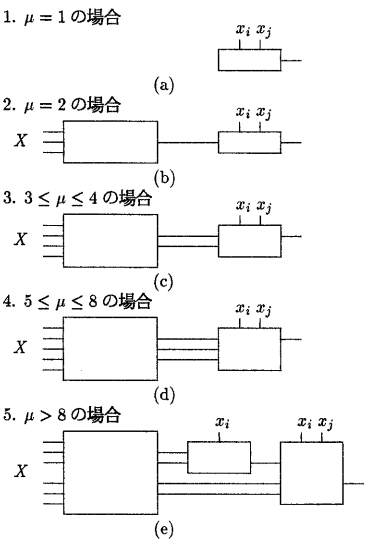

図 10: 列複雑度に対するそれぞれの実現法

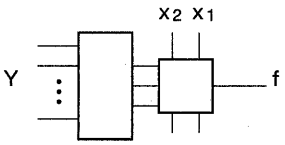

図 11: 部分対称関数の実現

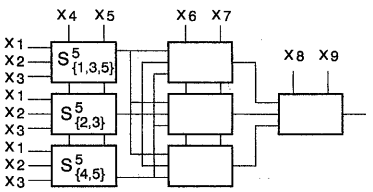

図 12: 5 入力 LUT による SYM9 の実現

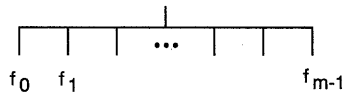

図 13: m 出力関数の BDD

表 4: 実験結果

関数	in/out	BDDEC		MIS-PGA2		Chortel-d		dag-map	
		L	LUTs	L	LUTs	L	LUT	L	LUT
z4ml	7/ 4	2	5	2	10	3	20	3	17
misex1	8/ 7	3	24	2	17	3	25	2	17
vg2	25/ 8	10	81	4	39	3	54	3	42
5xp1	7/10	2	16	2	21	4	29	3	28
9sym	9/ 1	3	7	3	7	5	130	5	63
rd84	8/ 4	3	12	3	13	4	69	4	48
e64	65/65	16	597	5	212	4	356	3	167
apex2	39/ 3	13	287	6	116	5	165	5	164
alu2	10/ 8	4	42	6	122	8	316	9	199
duke2	22/29	9	423	6	164	4	248	4	195
sao2	10/ 4	4	51	5	45	4	58		
rd73	7/ 3	2	7	2	8	4	52		
misex2	25/18	8	58	3	37	2	52		
f51m	8/ 8	3	16	4	23	5	65		
clip	9/ 5	3	32	4	54	4	83		
bw	5/28	1	28	1	28				
b9	16/ 5	7	68	3	47	3	62		

L: LUT 回路網の段数
LUTs: LUT の数

本手法は, 比較的簡単なアルゴリズムで設計でき, 変数の数が少ない場合には有効であると言える.

7 まとめ

FPGA の設計は, 基本回路の構造や評価関数が複雑であり, 現在のところ, 他の設計法に比べ圧倒的に優れているという方法は見つかっていない. 従って, 多くの設計アルゴリズムで用いられている方法は, 種々の手法を組み合わせて回路を構成し, その中から最も適切な回路を選ぶ』方法である. また,

1. 従来から研究されている AND-OR の多段回路の合成でさえも, それほど容易ではない.
2. FPGA の論理ブロックは, AND や OR などに比べると大幅に複雑である.
3. 多数の会社が毎年種々の論理構造の FPGA を開発している.

ことを考えると, 高性能な FPGA 設計システムを完成するには, 相当な投資と年月が必要であると予測できる.

最後に, 本手法のこれからの改善点をあげる. 段数を削減するために, 多出力関数を個々に分解することを考える. これにより, 図2のような分解がより多く検出できると推測できる. また, 変数の分解順序やコード割り当ての異なる回路を設計し, その中より選択するといった考えも盛り込む. 本プログラムでは, ドントケアを活用していない. ドントケアを活用することにより, さらに性能を改善できる.

謝辞

本研究は一部文部省科学研究費による. 卒業研究生の北島, 井上, 波多野君にはプログラムの開発で協力頂いた. また, 有益な資料を提供頂いた, カリフォルニア大学 Murgai 氏に深謝する.

参考文献

[1] R. L. Ashenhurst, "The decomposition of switching functions, " in Proceedings of an international symposium on the theory of switching, pp. 74-116, April 1957.

[2] B. Babba and M. Crastes, "Automatic synthesis on table lookup- based PGAs," Euro ASIC'92.

[3] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays*, Kluwer Academic Publishers, Boston 1992.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput. Vol. C-35, No. 8, Aug. 1986, pp. 677-691.

[5] J. Cong and Y. Ding, "An optimal technology mapping algorithm for delay optimization in look-up table based FPGA designs," Proc. ICCAD, pp.48-53, 1992.

[6] H. A. Curtis, Design of Switching Circuits, Van Nostand, Princton, N.J. 1962.

[7] S. Devadas, A. Wang, A. R. Newton, and A. Sangiovanni- Vincentelli,"Boolean decomposition in multilevel logic optimization," IEEE Journal of Solid-State Circuits, Vol. 24, April 1989, pp. 399-408.

[8] S. Ercolani and G. De Micheli, "Technology mapping for electrically programmable gate arrays, "DAC-28, pp. 234-239, June 1991.

[9] R. J. Francis, J. Roze, and K. Chung,"Chordtle: A technology mapping for lookup table-based field programmable gate arrays," DAC-27, pp. 613-619, June 1990.

[10] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping of lookup table-based FPGAs for performance," ICCAD-91, pp. 568-571, Nov. 1991.

[11] R.J. Francis, J. Roze, and Z. Vranesic,"Chortle-crf: Fast technology mapping for lookup table-based FPGAs," DAC-28, pp. 227-233, June 1991.

[12] K. Karplus, "Xmap: a technology mapper for table-lookup field-programmable gate arrays," DAC-91, pp. 240-243, June 1991.

[13] Y-T. Lai, M. Pedram and S. B. K. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," 30th ACM/IEEE Design Automation Conference, June 1993.

[14] S. Minato, "Minimum-width method of variable ordering for binary decision diagrams," IEICE Trans. Fundamentals Vol. E-75-A. No. 3, pp. 392-399, March 1992

[15] R. Murgai, Y. Nishizaki, N. Shenooy, R. Brayton, and A. Sangiovanni-Vincentelli,"Logic synthesis for programmable gate arrays,"Proc. DAC27, pp. 620-625, June 1990.

[16] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures," ICCAD-91, pp. 564-567, Nov. 1991.

[17] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance directed synthesis for table look up programmable gate arrays," ICCAD-91, pp. 564-567, Nov. 1991.

[18] 笹尾 勤,"ＰＬＡの作り方使い方," 日刊工業新聞社 (1986-05).

[19] T. Sasao, "Functional decomposition of PLA's," The International Workshop on Logic Synthesis, Research Triangle Park, North Carolina, May. 1987.

[20] T. Sasao, "Application of multiple-valued logic to a serial decomposition of PLA's," International Symposium on Multiple-Valued Logic, Zangzou, China, pp. 264-271, May 1989.

[21] T. Sasao, "FPGA design by generalized functional decompositioin," in (Sasao e.d.)*Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[22] M-H Tsai, T. T. Hwang, and Y-L. Lin, "Technology mapping for field programmable gate arrays using binary decision diagram," *Synthesis And Simulation Meeting and International Interchange*, April 6-8,1992.

[23] W. Wan and M.A. Perkowski, "A new approach to the decomposition of incompletely specified multi-output functions based on graph coloring and local transformations and its applications to FPGA mapping, " Proc. of Euro DAC'92, 1992.

[24] N-S Woo, "A heuristic method for FPGA technology mapping based on the edge visibility, "DAC-91, pp. 248-251, June 1991.

[25] J. Cong, A. Kahng, and P. Traj mar, "Graph based FPGA technology mapping for delay optimization", In 1st International ACM/SIGDA Workshop on FPGAs, 1992.