疑似クロネッカ決定グラフを用いたFPGA設計手法

笹尾 勤[1,2]　栗元 憲一[1]　松浦 宗寛[1]

1. 九州工業大学情報工学部電子情報工学科

2. マイクロ化総合技術センター

〒820-8502 飯塚市大字川津680-4

あらまし: 本論文では疑似クロネッカ決定グラフ (PKDD) を用いたLUT型FPGAの設計法を提案する．提案された手法は関数分解と疑似クロネッカ決定グラフによるLUTマッピングという二つの方法を用いている．プロトタイプの設計システムを開発した．ベンチマーク関数を用いた実験結果は提案された手法の有効性を示している．

和文キーワード: 疑似クロネッカ決定グラフ, 関数分解, フィールド・プログラマブル・ゲート・アレイ

FPGA Design using Pseudo-Kronecker Decision Diagrams

Tsutomu SASAO[1,2], Ken-ichi KURIMOTO [1] and Munehiro MATSUURA[1]

1. Department of Computer Science and Electronics

2. Center for Microelectornic Systems

Kyushu Institute of Technology

680-4 Kawazu, Iizuka 820-8502, Japan

Abstract: This paper shows a design method of LUT type FPGAs by using pseudo-Kronecker decision diagrams(PKDDs). It uses both functional decomposition and a mapping to LUT networks by using PKDDs. A prototype of system has been developed. Experimental results using benchmark functions show the encouraging results.

Key words: Pseudo-Kronecker decision diagrams, Functional decomposition, Field programmable gate array.

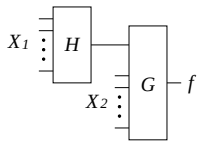


図 1: 関数分解

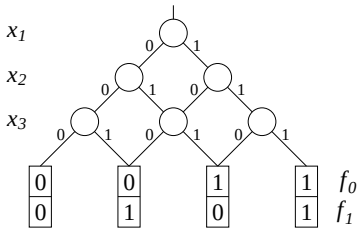


図 2: 例 2.1の MTBDD

1 まえがき

FPGA の設計において, 関数分解 [1] は非常に有効である. 与えられた論理関数が $f = g(h(X_1), X_2)$ の形で表現できるとき, f は図 1の回路構造で実現できる.

H の入力数が k 以下の場合, H は一つの LUT で実現できる. 繰り返し分解することにより, 与えられた論理関数を入力数が高々 k 個のブロックに分解できれば, LUT 回路として実現できることになる.

本論文では, デマルチプレクサ法とマルチプレクサ法の二つの設計手法について述べた後, それぞれの手法の利点と欠点について考察する. そして, 両方の利点を用いた新しい LUT 型 FPGA 設計手法を提案する.

2 決定グラフを用いた論理関数実現法

2.1 デマルチプレクサ法

多出力関数を表現する MTBDD を求め, 各終端節点をデマルチプレクサ (DMUX) で置き換える. また BDD の根の部分に定数 1 を加える. また, 入線度が 2 以上の節点に OR ゲートを付加する. 各終端節点で, 第 i ビットが 1 であるものを OR ゲートで結合し, 出力 f_i $(i = 0, 1, \ldots, m-1)$ とする.

例 **2.1** 図 *2*に *2* 出力関数

$$f_0 = x_1x_2 \vee x_2x_3 \vee x_3x_1,\ f_1 = x_1 \oplus x_2 \oplus x_3$$

の *MTBDD* を示す. 各終端節点を *DMUX* で置き換え, 入線度が *2* 以上の節点に *OR* ゲートを付加すると, 図 *3*の回路が得られる. 各終端節点と f_0 と f_1 の出力の部分にも, *OR* ゲートが必要な事に注意. ∎

分解法 **[7, 3, 5]**: MTBDD において図 4のように変数を X_1 と X_2 の二つに分割する. 決定グラフ A と B の部分

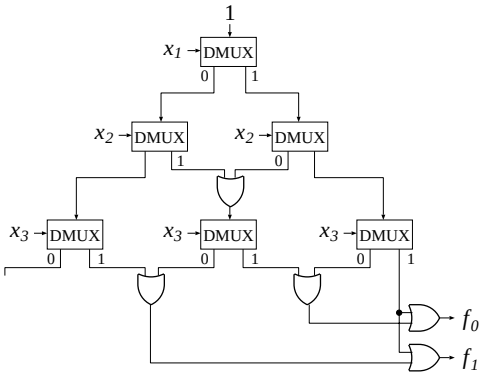


図 3: 例 2.1のデマルチプレクサ回路

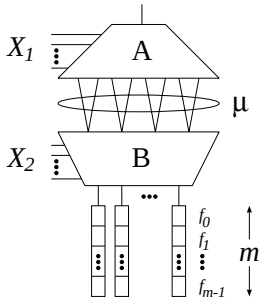


図 4: MTBDD の分解 (1)

の間のカットの数 μ が, 関数分解 $f(X_1, X_2) = g(h(X_1), X_2)$ の列複雑度 μ を与える. 分解後の DD は図 5 のようになり, A ' の出力数 (中間変数の個数) は $\lceil \log_2 \mu \rceil$ になる. A ' の複雑度は A の複雑度とは大きく異なる可能性がある. また, 新たに付加される決定グラフ A" は, 基本的には, デコーダを実現する. OR ゲートが必要な為, もとの DD の節点数から回路の素子数を正確に見積もるのは困難である. また, より簡単な回路を実現するためには符号化を考慮する必要がある [6].

本手法の特徴: 出力数 m が大きい場合, MTBDD の節点数は対応する SBDD の節点数よりはるかに大きくなる傾

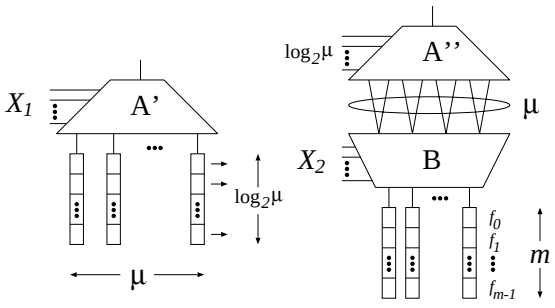


図 5: MTBDD の分解 (2)

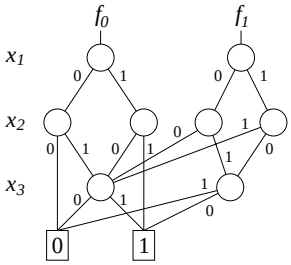


図 6: 2.1の SBDD

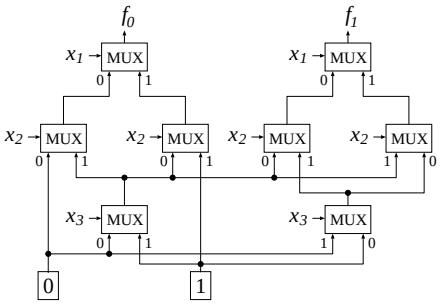


図 7: 例 2.1のマルチプレクサ回路

向がある．その場合, 出力を分割することによって DD を小さく保つ必要がある．多出力関数において, 複数の出力に共通な分解 (中間変数) を見つけるためには特別な工夫が必要である [13]． 分解の存在しない関数 (数学的には, ほとんど全ての関数) に対しては, 分解を行っても効果はなく, 素子数は増えるだけである．そのような場合, いくつかの FPGA 論理合成システムはシャノン展開を再帰的に適用している [10]．しかしながら, そのような構成法は $O(2^n)$ の素子数が必要になる．

2.2 マルチプレクサ法

多出力関数を実現する SBDD を求め, 各非終端節点をマルチプレクサ (MUX) で置き換える．このとき, 根の部分に関数 f_i $(i = 0, 1, \ldots, m-1)$ を生成する．

例 **2.2** 図 *6*に例 *2.1*の *2* 出力関数の *SBDD* を示す．非終端節点をマルチプレクサに置き換える事により, 図 *7*の回路が得られる． ∎

分解 (分割) 法 **[12]**: 図 8のように入力変数を X_1 と X_2 に分割する．図 9に分解 (分割) 後の SBDD を示す．B の各出力に対して A の BDD に補助変数を追加する．但し, B の出力が定数, あるいは, 1 変数関数の場合は省略できる．

種々のカットを考えることにより, 非分離的分解も見つける事ができる．分解された SBDD において, 非終端節点をマルチプレクサに置き換える事によって, 回路が求

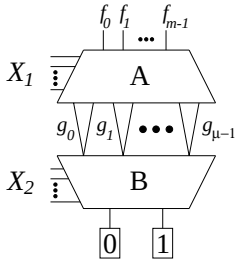


図 8: SBDD の分解 (1)

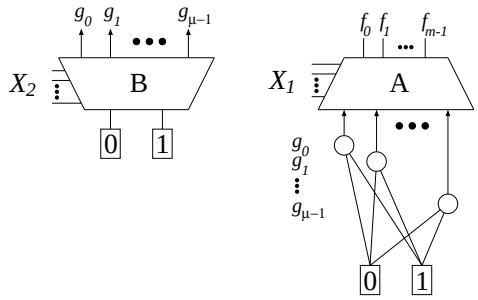


図 9: SBDD の分解 (2)

まる．そのため, 分解後の回路の素子数の見積もりは容易である．

本手法の特徴: SBDD は, 通常, 対応する MTBDD よりも小さい． n 入力 m 出力の SBDD のノード数は高々 $O(2^{n+\log_2 m}/(n+\log_2 m))$ 個である．マルチプレクサ法において, 複数の出力が節点を共有する．また, 中間変数の符号化は自動的に行われる．本手法は, 関数分解が存在しない関数に対しても有効である．しかし, B の入力数が n_2 のとき, B の出力数 (中間変数の個数) は最大 $O(2^{2^{n_2}})$ になる．中間変数が多いことは FPGA の設計において致命的な欠点になりうる．

2.3 二つの手法を組み合わせた設計手法

ここでは, 最初に, デマルチプレクサ法で論理関数を小さくし, 次にマルチプレクサ法で各論理関数を LUT 回路にマッピングし, 最後に, 冗長な LUT を除去するという構成法を提案する．

2.3.1 大域的分解

多出力関数を大域的に分解する方法として, 入力の分割と, 出力の分割の 2 種類が存在する．ここでは, 入力変数集合の分割法について考察する．一般に, 任意の n 入力 m 出力関数は高々

$$A = \frac{2^{n+\log_2 m}}{n+\log_2 m} \tag{1}$$

個の素子を用いて実現可能である.

図 5の MTBDD において, 分解後の回路の素子数の上界は, X_1 の入力変数の個数を n_1, X_2 の入力変数の個数を n_2, この分解の列複雑度を μ とすると,

$$B = \frac{2^{n_1+\log_2\log_2\mu}}{n_1+\log_2\log_2\mu} + \frac{2^{n_2+\log_2\mu+\log_2 m}}{n_2+\log_2\mu+\log_2 m}, \quad (2)$$

となる. 従って, 変数の分割 (X_1, X_2) を用いた際, $\rho = B/A$ が 1 より小さい場合は, 分解が有効である可能性は大きい. 逆に ρ の値が 1 に比べて大きい場合は, 分解が有効である可能性は小さい. 従って, 入力変数集合の分割は, ρ が 1 より小さいものに対してのみ行い, 分解後の SBDD の節点数の総和がもとの SBDD より小さくなる限り分解を続ける.

2.3.2 LUT 回路へのマッピング

BDD の代りに PKDD を用いることにより, 論理素子数を減らすことができる. PKDD は BDD を一般化したものであり, 同一の論理関数を表現する BDD より大きくなることはない [8]. また, ある種の論理関数では, その BDD 表現の大きさが入力変数の個数 n に対して指数関数のオーダで増えるのに対して, PKDD 表現の大きさは n の多項式オーダで十分な場合もある. 特に算術演算関数では, PKDD 表現は, BDD 表現より格段に小さい.

2.3.3 LUT 回路の最適化

LUT 回路が完成した時点で, 次の最適化を行う.

- 二つの隣接する LUT が併合できる場合, 併合して, LUT の個数を減らす.
- 中間変数の個数が大きすぎる場合, 依存変数の最小化問題を解くことにより, 中間変数の除去を行う (詳細は省略).

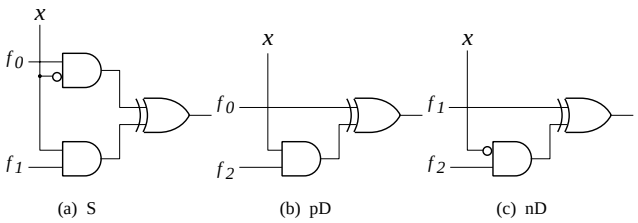
3 PKDD に基づく FPGA 設計法

3.1 回路構成法

多出力関数を SPKDD(Shared Pseudo-Kronecker Decision Diagram) で表現し, 各節点を図 10の各モジュールで置き換えれば, 多出力関数を実現できる. SPKDD が大きい場合には, それを分割し, 各部分を k 入力 LUT にマッピングする.

3.2 DD の分割法

定義 **3.1** 決定グラフを $G = (V, E)$ とし, V_1 を V の非終端節点の部分集合とする. G から V_1 を取り除くと, グラフが二つの要素に分離するとき,V_1 をカットという. ただし, 分離後一方の要素は根に接続しており, もう一方の要素は全ての終端節点に接続しているものとする.



(a) S (b) pD (c) nD

図 10: PKDD 節点に対応するモジュール

定義 **3.2** カット V_1 のうちで, V_1 からどの節点を取り除いても, カットにならないものを極小カットという.

定義 **3.3** 決定グラフ G の非終端節点の集合を $V = \{v_1, v_2, \ldots, v_N\}$ とする. このとき, 論理関数 ξ を

$$\xi(u_1, u_2, \ldots, u_N) = 1 \Leftrightarrow \{v_i | u_i = 1\} \text{ は } G \text{ のカット.}$$

と定義する. このとき, ξ を G の特性関数という.

アルゴリズム **3.1** *(*決定グラフ G の特性関数の導出*)*

1. G の非終端節点が一つのとき, G の特性関数は $\xi(u_1) = u_1$ である.
2. $G(V, E)$ が一つのソースを持つとき, それを s とする. s のファンアウトを $FO(s) = \{v_1, v_2, \ldots, v_m\}$ とし, G_j を節点 v_j を根に持つ部分 DAG とする. ξ_{G_j} を G_j の特性関数とすると, G の特性関数は

$$\xi_G = u_s \vee \bigwedge_{j=1}^{m} \xi_{g_j},$$

で与えられる. ここで, u_s は節点 s に対応する変数である.

3.3 FPGA 合成アルゴリズム

アルゴリズム **3.2** *(PKmap)*

1. 各出力の依存変数を求める. 依存変数が k 以下の場合, 一つの LUT で実現する.
2. *1* で実現されなかった関数の $SBDD$ を求める. $SBDD$ が大きい場合, 依存変数を考慮して, 出力を幾つかのグループに分割し, 複数の $SBDD$ を作成する.
3. 各 $SBDD$ が分解可能な場合は分解する.
4. 各 $SBDD$ から $PKDD$ を作成し, 最適化する.
5. $|X_2| = k$ となるようなカットを求め, 回路のコストの上界を計算する.
6. $PKDD$ の特性関数を求める.
7. 極小カットに対するコストを計算し, コスト最小の極小カットを求める. この際, *5* で求めたコストを上界とする.
8. カットに対して回路を分割する.
9. それぞれの回路に対して, 本アルゴリズムを再帰的に適用する.

表 1: PKDD と BDD のノード数

Name	In	Out	PKDD	BDD
ran_7	7	1	28	32
ran_8	8	1	50	59
ran_9	9	1	92	113
adr_8	16	9	31	38
adr_10	20	11	39	48
adr_12	24	13	47	58
wgt_12	12	4	61	89
wgt_14	14	4	77	119
wgt_16	16	5	101	157
clique_5	10	1	19	77
clique_6	15	1	36	453

4 実験結果

与えられた関数をいくつかの SBDD に分解し, それぞれの SBDD を PKDD に変換するプログラム, 及び, PKDD を FPGA に変換するプログラムを開発した. 2章での考察を確認するため以下の実験を行った.

4.1 予備実験

4.1.1 PKDD と BDD

表 1 は各種関数を表現する BDD および PKDD の非終端節点数を比較したものである (否定エッジを使用). 各グラフに対して, 節点数がなるべく少なくなるような入力変数の順序を求めた. BDD の節点数を最小化する変数順序は, PKDD では, 必ずしも節点数を最小化しない.

ran_n: n 入力の乱数関数. 真理値表濃度は 1/2.

adr_n: n ビット加算回路. $2n$ 入力 $(n+1)$ 出力.

wgt_n: n ビット重み計数回路 [9].

clique_n: 3 クリーク判定回路. $N = n(n-1)/2$ 入力 1 出力関数. 接点数 n の完全グラフ G を考え, G の枝 $e_i (i = 1, 2, \ldots, N)$ に変数 x_i を対応させる. $x_i = 1$ のとき, G の部分グラフ G_S が枝 e_i を含むとする. G_s が次数 3 の完全グラフを丁度奇数個含むとき, $g(X) = 1$ とする. $g(X)$ を Reed-Muller 変換をした関数.

4.1.2 デマルチプレクサ法とマルチプレクサ法の比較

2章での考察を確認するため, プログラムが公開されている FlowSYN[2] と SIS_1.2[11] とアルゴリズム 3.2に示した方法 (PKmap) との比較を行う.

FlowSYN は LUT の個数よりも段数を減らすことを優先するように設計されている. 初期回路の影響を除くため全てのデータは, 一旦, 二段回路に展開している. その後, SIS(script.rugged) を用いて, 多段回路に合成し, そのデータを FlowSYN に与えている. 同様に, SIS_1.2 では同様の入力データを用いて以下のスクリプトを使用した.

表 2: 乱数関数とクリーク関数を実現する LUT の個数

Name	In	Out	PKmap	FlowSYN	SIS_1.2
ran_7	7	1	6	7	16
ran_8	8	1	15	90	47
ran_9	9	1	58	174	92
clique_5	10	1	8	180	over
clique_6	15	1	17	over	over

over: SIS time over (more than 1 hour)

表 3: ビット計数関数と加算器を実現する LUT の個数

Name	In	Out	PKmap	FlowSYN	SIS_1.2
adr_8	16	9	13	34	25
adr_10	20	11	18	34	31
adr_12	24	13	21	over	over
wgt_10	10	4	12	60	92
wgt_12	12	4	15	76	81
wgt_14	14	4	28	over	over
wgt_16	16	5	31	over	over

over: SIS time over (more than 1 hour)

xl_partition –tm
xl_k_decomp –n 5
xl_partition –tm
xl_cover

乱数関数: 表 2から明らかなように, PKmap は FlowSYN や SIS_1.2 よりも少ない LUT 数の回路を生成している. このことは, マルチプレクサ法が乱数関数に対して効果的であることを示している. また, 計算時間も PKmap は圧倒的に速い. これは PKmap が単にマッピングだけをしているのに対し, FlowSYN や SIS_1.2 が分解を検出しているためである.

クリーク関数: この関数は, 分解不能であり、 BDD のサイズは入力変数に対し, 指数関数的に増加するが, PKDD のサイズは入力変数に対し, 多項式オーダの増加をする. よって, PKmap は, この関数に対して非常に効果的である. 表 2に示すように, PKmap は非常に良い結果を示している.

加算回路とビット計数回路: 表 1に示すように, 加算回路とビット計数回路に対して, PKDD は BDD よりノード数が少ない. また, これらの関数は分解可能である. 表 3に示すように, これらの関数に対して, PKmap は, FlowSYN や SIS_1.2 よりも LUT の個数が少ない回路を生成している. また, 計算時間も PKmap は, 非常に短い.

4.1.3 考察

種々のベンチマーク関数に対する実験より, 以下の方策が有効であるとの結論に達した.

- 分解可能な関数に対しては, デマルチプレクサ法が有効であり, 分解不能な関数に対しては, マルチプレク

表 4: ベンチマーク関数を実現する LUT の個数

Name	In	Out	PK map	Flow SYN	SIS 1.2	[10]	[6]	[4]	[12]
5xp1	7	10	15	20	42	9		13	26
9sym	9	1	8	7	20	6	8	6	31
alu2	10	6	49	165	91	47	52	50	117
b9	16	5	28	44	32	33	41	36	37
clip	9	5	21	24	67	11	22	14	32
count	35	16	34	43	31	31		31	31
duke2	22	29	197	211	281		213	116	134
f51m	8	8	12	16	42	7		12	
misex1	8	7	12	16	20	10		13	13
misex2	25	18	33	36	34	36	40	29	30
rd73	7	3	7	8	27	6		6	17
rd84	8	4	11	13	72	7	12	9	24
sao2	10	4	31	43	73	20	22	22	39
t481	16	1	5	5	5	5			14

サ法が有効である.

- BDD が大きい場合, 対応する PKDD の最適化には, 時間がかかる. その場合, 入力変数集合や, 出力変数集合を分割することにより, BDD を取り扱い可能な大きさまで分割する.
- 入力数と出力数が小さい場合, また, 分解不能な関数に対しては, マルチプレクサ法は他の方法に比べ良い回路を生成する.

4.2 標準ベンチマーク関数を用いた実験

現在, システム全体は完成しておらず, 各ステップが独立に動いている. 最後のステップ (LUT 回路の簡単化) は, まだ動作していないが, PKmap は既存の手法と遜色のない結果を得ている. 表 4にベンチマーク関数に対する LUT 数 ($k = 5$) を示している. [10] の列は, 再代入を用いない場合を示している.

5 結論

本稿では, 最初に, 多出力関数を表現する方法として, デマルチプレクサ法と, マルチプレクサ法の二つの方法を定式化し, 各々の手法の得失を比較した. デマルチプレクサ法は, 分解可能な関数に適しており, マルチプレクサ法は, 分解不能な関数の実現に適している. 次に, 二つの方法の長所を生かした LUT 回路の設計法を提案した. さらに, 設計システムのプロトタイプを作成し, 実験を行い, 本手法が有望であることを示した.

謝辞

本研究は, 一部, 文部省科学研究費補助金による.

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] J. Cong and Y. Ding, "Beyond the combinational limit in depth minimization for LUT-based FPGA design," *ICCAD '93*, pp. 110-114, Nov. 1993.

[3] Ting-Ting Hwang, R. M. Owens, M. J. Irwin, and Kuo Hua Wang, "Logic synthesis for field-programmable gate arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 13, No. 10, pp. 1280-1287, Oct. 1994.

[4] J.-H. R. Jian, J.-Y. Jou, and J.-D. Huang, "Compatible class encoding in hyper-function decomposition for FPGA synthesis," *Design Automation Conference*, pp. 712-717, June 1998.

[5] Y-T. Lai, M. Pedram, and S. B. K. Vrudhula, "EVBDD-based algorithm for integer linear programming, spectral transformation, and functional decomposition," *IEEE Trans. CAD*, Vol. 13, No. 8, pp. 959-975, Aug. 1994.

[6] C. Legl, B. Wurth, and K. EcKl, "Computing support-minimal subfunctions during functional decomposition," *IEEE Trans. VLSI*, Vol. 6, No. 3, pp. 354-363, Sept. 1998.

[7] T. Sasao, "FPGA design by generalized functional decomposition," *(Sasao ed.) Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[8] T. Sasao and M. Fujita (ed.), *Representation of Discrete Functions*, Kluwer Academic Publishers, 1996.

[9] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[10] H. Sawada, T. Suyama, and A. Nagoya, "Logic synthesis for look-up table based FPGAs using functional decomposition and Boolean resubstitution," *IEICE Trans. Inf. & Sys.*, Vol. E80-D, No. 10, pp. 1017-1023, Oct. 1997.

[11] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoy, P. Ptephen R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," *U.C. Berkeley Technical Report*, UCE/ERL M92/41, May 1992.

[12] T. Stanion and C. Sechen, "A method for finding good Ashenhurst decompositions and its applications to FPGA synthesis," *32nd Design Automation Conference*, pp. 60-64, June 1995.

[13] B. Wurth, K. Eckl, and K. Anterich, "Functional multiple-output decomposition: Theory and implicit algorithm," *Design Automation Conf.*, pp. 54-59, June 1995.