LUTMIN: FPGA Logic Synthesis with MUX-Based and Cascade Realizations

Tsutomu Sasao [1] and Alan Mishchenko [2]

[1] Dept. of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka 820-8502, Japan
[2] Dept. of EECS, University California Berkeley, CA, USA

Abstract

First, this paper considers the number of LUTs to implement logic functions based on MUX-based realization and cascade realization. This is useful to quickly estimate the number of LUTs to implement the functions on a FPGA. Second, this paper shows an algorithm to realize logic functions by 6-LUTs using cascade and MUX-based realizations. It often produces smaller circuits than previous methods when the number of the input variables is smaller than 16.

1 Introduction

A K-LUT (look-up table) is a module that realizes an arbitrary K-variable function. This paper considers the number of K-LUTs to realize given function. FPGAs with $K = 4$ or 5 input LUTs are believed to be the most efficient [15, 16]. However, in the current technology, FPGAs with $K = 6$ are standard [24, 1, 2]. Thus, we have the following:

Problem 1.1 *Given an n-variable (multiple-output) function f, find the minimum number of 6-LUTs needed to implement f, where $n \leq 20$.*

Unfortunately, it is a very hard problem, in general. So, we try to obtain an upper bound by using properties or measures of the function. The property of the function should be quickly detected, such as symmetry, and measures should be quickly calculated. Such measures include

- the number of variables (*i.e.*, the support size),
- the C-measure of a function,
- the number of products in the SOP [11],
- the number of literals in the factored expression [11],
- the number of nodes in the decision diagram [18].

In this paper, we present two types of realizations: MUX-based realization and cascade realization. Also we show various theorems on the number of LUTs. After that, we we show an algorithm to realize logic functions by using 6-input LUTs (6-LUTs). The algorithm is suitable for the functions with up to 16 inputs, and often produces better solutions than previous methods.

Due to the page limitation, all proofs of theorems are omitted.

2 MUX-based Realization

In this part, we show a universal method to implement an n-variable function, and obtain upper bounds on the number of LUTs. Such bounds are useful when we only know the number of the input variables n.

Definition 2.1 *A multiplexer with a single control input (1-MUX) is the selection circuit shown in Fig. 2.1. It performs the logical operation:*

$$g(x, y_0, y_1) = \bar{x}y_0 \vee xy_1.$$

t-MUX is shown in Fig. 2.2. It is a multiplexer with t control inputs $(x_1, \ldots, x_t)$, and 2^t data inputs $(y_0, y_1, \ldots, y_{2^t-1})$. Let $g(x_1, \ldots, x_t, y_0, y_1, \ldots, y_{2^t-1})$ be the output function. Then, $g = y_a$ when the decimal representation of the control input $(x_1, \ldots, x_t)$ is a. That is, when the control input is $(0, 0, \ldots, 0)$, the top data input y_0 is selected. When the control input is $(0, 0, \ldots, 1)$, the second data input y_1 is selected. Also, when the control input is $(1, 1, \ldots, 1)$, the last data input y_{2^t-1} is selected.

Lemma 2.1 *An n-MUX is realized by using $2^n - 1$ copies of 1-MUXs.*

Example 2.1 *A 3-MUX is realized with $2^3 - 1 = 7$ copies of 1-MUXs as shown in Fig. 2.3.* *(End of Example)*

Definition 2.2 *Let $B = \{0, 1\}$. Let $X = (x_1, x_2, \ldots, x_t)$. Then, we can consider that X takes its value from $P =$*

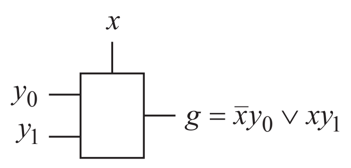

Figure 2.1. 1-MUX.

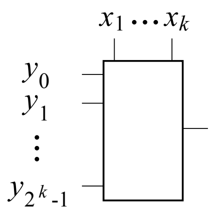

Figure 2.2. K-MUX.

$\{0, 1, \ldots, 2^t - 1\}$. *Let S be a subset ($S \subseteq P$) of P. Then, X^S is a* **literal** *of X. When $X \in S$, $X^S = 1$, and when $X \notin S$, $X^S = 0$. Let $S_i \subseteq P_i$ ($i = 1, 2, \ldots, n$), then ${X_1}^{S_1}{X_2}^{S_2}\cdots{X_n}^{S_n}$ is a* **logical product**. $\bigvee_{(S_1,S_2,\ldots,S_n)} {X_1}^{S_1}{X_2}^{S_2}\cdots{X_n}^{S_n}$ *is a* **sum-of-products expression** *(***SOP***). When $S_i = P_i$, ${X_i}^{S_i} = 1$ and the logical product is independent of X_i. In this case, literal ${X_i}^{P_i}$ is redundant and can be deleted. A logical product is also called a* **term**, *or a* **product term**.

Theorem 2.1 *An arbitrary n-variable function is expanded as follows:*

$$f(X_1, X_2) = \bigvee_{i \in P} g_i(X_1)X_2^i,$$

where $X_1 = (x_1, x_2, \ldots, x_k)$ and $X_2 = (x_{k+1}, x_{k+2}, \ldots, x_n)$, $P_2 = \{0, 1, \ldots, 2^{n-k}\}$, and the OR is performed with respect to 2^{n-k} elements.

Example 2.2 *Consider the realization of a 7-variable function $f(X_1, X_2)$, where $X = (x_1, x_2, \ldots, x_5)$, and $X_2 = (x_6, x_7)$. f is expanded into a sum of four products:*
$f(X_1, X_2) = \bigvee_{i=0}^{3} g_i(X_1)X_2^i$
$= g_0(X_1)X_2^0 \vee g_1(X_1)X_2^1 \vee g_2(X_1)X_2^2 \vee g_3(X_1)X_2^3$.
As shown in Fig. 2.4, 2-MUX can be implemented by using three copies of 1-MUXs. The top LUT in the leftmost column realizes g_0, which is selected when $(x_6, x_7) = (0, 0)$. The second LUT in the leftmost column realizes g_1, which is selected when $(x_6, x_7) = (0, 1)$. Other LUTs are derived similarly. (End of Example)

Theorem 2.2 *When $3 \leq K \leq n$, an arbitrary n-variable function is realized by using at most $2^{n-K} - 1$ copies of 1-MUXs, and 2^{n-K} copies of K-LUTs.*

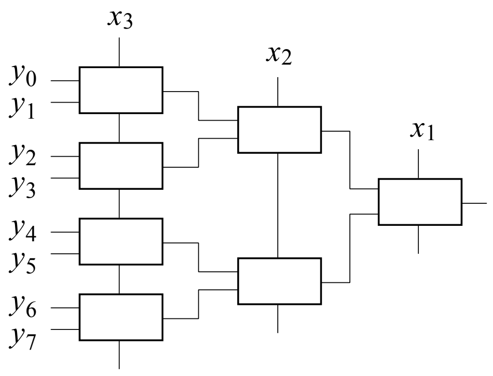

Figure 2.3. Realization of 3-MUX by using 1-MUXs.

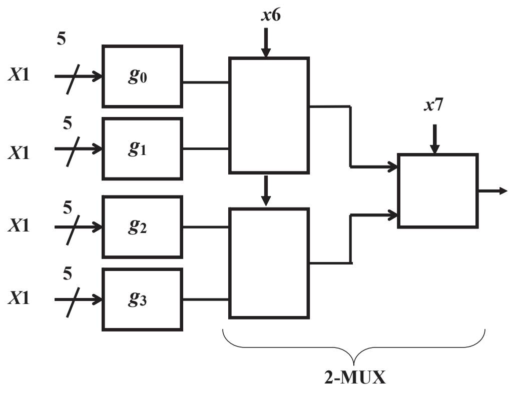

Figure 2.4. Realization of an arbitrary 7-variable function using 5-LUTs.

To implement 2-MUX, we have more efficient realization than Fig.2.4. When $K = 6$, a 2-MUX can be realized by a single 6-LUT instead of using three copies of 1-MUXs.

Lemma 2.2 *An arbitrary function of $n = K + 1$ variables can be implemented with at most three K-LUTs, where $K \geq 3$.*

Lemma 2.3 *An arbitrary function of $n = K + 2$ variables can be implemented with at most five K-LUTs, where $K \geq 6$.*

Theorem 2.3 *[19, 23] The number of 6-LUTs to implement an arbitrary n-variable function f is*

- $(2^{n-4} - 1)/3$ *or less, when n is even, and*
- $(2^{n-4} + 1)/3$ *or less, when n is odd.*

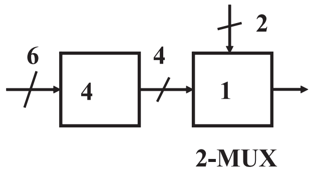

Figure 2.5. Realization of an arbitrary 8-variable function using 6-LUTs.

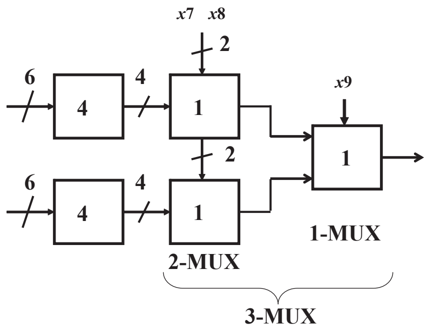

Figure 2.6. Realization of an arbitrary 9-variable function using 6-LUTs.

Example 2.3 *The number of 6-LUTs to implement an n-variable function is*

- *5 or less, when $n = 8$. In this case, $g_i(X_1)$ $(i = 0, 1, 2, 3, 4)$ are implemented by four copies of 6-LUTs, while the 2-MUX is implemented by a single 6-LUT, as shown in Fig. 2.5. In the figure, the numbers in squares denote the numbers of necessary LUTs.*
- *11 or less, when $n = 9$. In this case, $g_i(X_1)$ $(i = 0, 1, 2, \ldots, 7)$ are implemented by 8 copies of 6-LUTs, while the 3-MUX is implemented by three 6-LUTs as shown in Fig. 2.6.*
- *21 or less, when $n = 10$. In this case, $g_i(X_1)$ $(i = 0, 1, 2, \ldots, 15)$ are implemented by 16 copies of 6-LUTs, while the 4-MUX is implemented by using five 6-LUTs as shown in Fig. 2.7.*

(End of Example)

3 Cascade Realization

In this part, we show a cascade realization of a logic function. This gives more compact circuits than the MUX-based realization, but is only applicable to function with special properties.

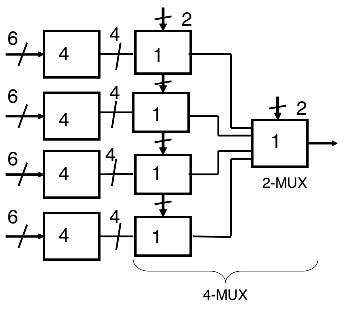

Figure 2.7. Realization of an arbitrary 10-variable function using 6-LUTs.

$$X_1 = (x_1, x_2)$$

		00	01	10	11
$X_2 = (x_3, x_4)$	00	0	0	0	1
	01	1	1	0	0
	10	0	1	0	0
	11	0	0	0	0

Figure 3.1. Decomposition table of an logic function.

3.1 Functional Decomposition

Definition 3.1 *[3] Let $f(X)$ be a logic function, and (X_1, X_2) be a partition of the input variables, where $X_1 = (x_1, x_2, \ldots, x_k)$ and $X_2 = (x_{k+1}, x_{k+2}, \ldots, x_n)$. The* **decomposition table** *for f is a two-dimensional matrix with 2^k columns and 2^{n-k} rows, where each column and row is labeled by a unique binary code, and each element corresponds to the truth value of f. The function represented by a column is a* **column function***. Variables in X_1 are* **bound variables***, while variables in X_2 are* **free variables***. In the decomposition table, the* **column multiplicity** *denoted by μ_k is the number of different column patterns.*

Example 3.1 *Fig. 3.1 shows a decomposition table of a 4-variable function. Since all the column patterns are different, the column multiplicity is $\mu_2 = 4$. (End of Example)*

Theorem 3.1 *[6] For a given function f, let X_1 be the bound variables, let X_2 be the free variables, and let μ_k be the column multiplicity of the decomposition table. Then, the function f can be realized with the network shown in Fig. 3.2. In this case, the number of signal lines connecting two blocks H and G is $\lceil \log_2 \mu_k \rceil$.*

When the number of signal lines connecting two blocks is smaller than the number of input variables in X_1, we can often reduce the total amount of memory by the realization in

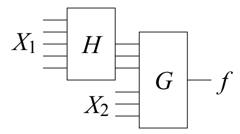

Figure 3.2. Realization of a logic function by decomposition.

Fig. 3.2 [7]. Such technique is call a **Curtis decomposition** [6], in this paper.

3.2 C-Measure

Definition 3.2 *Let $f(x_1, x_2, \dots, x_n)$ be a logic function. Let μ_k be the column multiplicity of the decomposition table for $f(X_1, X_2)$, where $X_1 = (x_1, x_2, \dots, x_k)$ and $X_2 = (x_{k+1}, \dots, x_n)$. The* **C-Measure** *of the function f is $\max(\mu_1, \mu_2, \dots, \mu_n)$, and denoted by $\mu(f)$.*

By repeatedly applying functional decompositions to a given function, we have an **LUT cascade** [20] shown in Fig. 3.3. An LUT cascade consists of **cells**, and the signal lines connecting adjacent cell are **rails**. A logic function with a small C-measure can be realized by a compact LUT cascade. To obtain the C-measure, a decomposition table is not necessary. A **quasi-reduced binary decision diagram** (QRBDD) that represents the logic function directly shows the C-measure: The C-measure is equal to the maximum width of the QRBDD, where the variable ordering is $(x_1, x_2, \dots, x_n)$ [17].

3.3 LUT Cascade

Lemma 3.1 *[20] Let $\mu(f)$ be the C-measure of a function f. Then, f can be implemented by an LUT cascade, whose cells have at most $\lceil \log_2 \mu(f) \rceil + 1$ inputs, and $\lceil \log_2 \mu(f) \rceil$ outputs.*

Lemma 3.2 *[21] In an LUT cascade that realizes the function f, let n be the number of the input variables; s be the number of cells; $w = \lceil \log_2 \mu(f) \rceil$ be the maximum number of rails; K be the number of inputs for a cell; $n \geq K + 1$; and $K \geq \lceil \log_2 \mu(f) \rceil + 1$. Then, an LUT cascade satisfying the following condition exists:*

$$s \leq \left\lceil \frac{n - w}{K - w} \right\rceil .$$

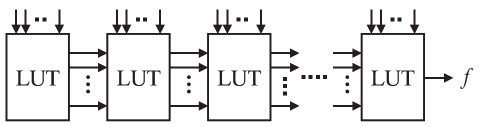

Figure 3.3. LUT cascade.

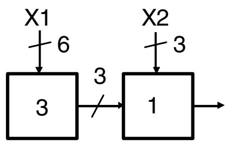

Figure 3.4. Realization of a 9-variable function with C-measure at most 8.

3.4 Bounds for Functions with Small C-Measure

From Lemmas 3.1 and 3.2, we can derive the followings:

Theorem 3.2 *[23] The number of 6-LUTs to implement an arbitrary n-variable function, where $n \geq 8$ is:*

1. *$5n - 35$ or less, when $\mu(f) \leq 32$.*
2. *$2n - 11$ or less, when $\mu(f) \leq 16$.*
3. *$n - 5$ or less, when $\mu(f) \leq 8$ $(n = 3r)$.*
4. *$n - 4$ or less, when $\mu(f) \leq 8$ $(n \neq 3r)$.*

Example 3.2 *Let $f(X_1, X_2)$ be a 9-variable function, where $X_1 = (x_1, x_2, \dots, x_6)$ and $X_2 = (x_7, x_8, x_9)$. Let μ_6 be the column multiplicity of the decomposition of f with respect to (X_1, X_2). If $\mu_6 \leq 8$, then $f(X_1, X_2)$ can be implemented with four copies of 6-LUTs, as shown in Fig. 3.4. To check that a given 9-variable function can be realized as Fig. 3.4, we need to check the column multiplicities for only $\binom{9}{6} = 42$ combinations.*

3.5 Bound for Symmetric Functions

When the given function is symmetric, it can be implemented more efficiently than a general function [17, 22]. Efficient algorithms to detect symmetric functions exist, e.g., [13].

Lemma 3.3 *Let f be a symmetric function of n-variables. Then, $\mu(f) \leq n + 1$.*

Theorem 3.3 *The number of K-LUTs to implement an n-variable symmetric function is*

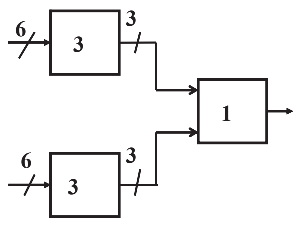

Figure 3.5. Realization of a symmetric function of 12 variables by 6-LUTs.

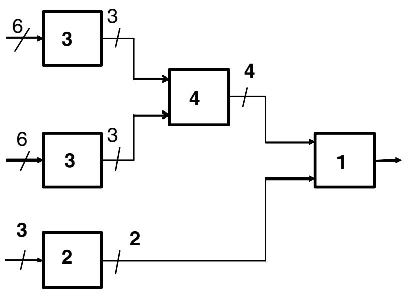

Figure 3.6. Realization of a symmetric function of 15 variables by 6-LUTs.

1. *4 or less, when $n = 9$ and $K = 6$. Fig. 3.4 shows the realization.*

2. *7 or less, when $n = 12$ and $K = 6$. Fig. 3.5 shows the realization.*

3. *13 or less, when $n = 15$ and $K = 6$. Fig. 3.6 shows the realization.*

4 Logic Synthesis with 6-LUTs

For some class of functions, the LUT cascade realizations require many fewer LUTs than other methods. Current version of the program works for only small problems. The algorithm obtains the column multiplicity μ and calculate the number of cells in the LUT cascade. When $\lceil log_2(\mu)\rceil \geq K$, the algorithm uses a MUX-based realization. The outline of the method is as follows:

Algorithm 4.1 *(LUTMIN:Logic synthesis using 6-LUTs)*

1. *For a multi-output function, decompose it into single-output functions.*

2. *For each single-output function F:*
   - *(a) Minimize support of F by removing redundant variables.*
   - *(b) If the support size of F is less than or equal to K (LUT size), implement it using one K-LUT.*
   - *(c) If the support size of F is exactly $K+1$, the best we can do is to realize with by two K-LUTs. In this case, we check cofactors of F w.r.t. each variable. If the support size of one cofactor is $K-2$ or less, implement F using two K-LUTs composed of the larger cofactor in one LUT, and smaller cofactor and MUX in another LUT.*
   - *(d) If F is not decomposed in step (2.b) and (2.c), reorder BDD of F to minimize the number of nodes.*
   - *(e) Consider $M = \lceil log_2(\mu)\rceil$, where μ is column multiplicity of F w.r.t. the bound-set composed of K variables on top of the BDD.*
   - *(f) If $M < K$, use Curtis decomposition w.r.t. K topmost variables as the bound set. Implement M bound set nodes using K-LUTs and perform recursive decomposition of the composition function whose support size is equal to: $Support\_size(F) - K + M$.*
   - *(g) If $M \geq K$, create 2-MUX w.r.t. to the two topmost variables in the BDD, and recursively decompose four cofactors.*

3. *As a last post-processing step, detect and merge functionally equivalent nodes in the resulting LUT network. (Such nodes could be created in step (2.g) when, for example, two of the four cofactors of F are equal.)*

Example 4.1 *Consider sym12 [19], a symmetric function of 12 variables. sym12 is 1 iff the number of 1's in the inputs is between 4 and 8.*

1. *The column multiplicity of the function with respect to the bound set composed of $K = 6$ variables is $\mu_6 = 7$. Thus, $M = \lceil log_2 7\rceil = 3$.*

2. *Since $M < K$, we use Curtis decomposition with respect to the topmost variables in the bound set: $x_1, x_2, \ldots, x_6$. Realize the first cell using 6-LUTs, which corresponds to the leftmost cell in Fig.4.1.*

3. *The composition function has 12-6+3=9 variables. The top of the variables are three outputs of the leftmost cell, and x_7, x_8, x_9. In this case, the column multiplicity is $\mu_9 = 8$. Thus, $M = \lceil log_2 8\rceil = 3$.*

4. *Since $M < K$, we use Curtis decomposition with respect to the topmost variables in the bound set: three outputs of the leftmost cell, and x_7, x_8, x_9. Realize the second cell using 6-LUTs, which corresponds to the middle cell in Fig.4.1.*

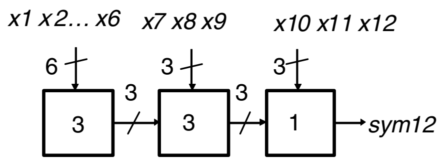

Figure 4.1. sym12 implemented by 6-LUTs.

Table 5.1. Number of 6-LUTs realize functions.

			MBC		$LUTMIN$		SAS	
$Name$	PI	PO	LUT	lev	LUT	lev	LUT	lev
$alu4$	14	8	341	5	248	5	**208**	8
amd	14	24	**86**	3	109	4	93	4
$apex4$	9	19	536	4	187	3	**164**	4
$chkn$	29	7	**74**	5	117	10	118	11
$cordic$	23	2	**9**	3	22	6	22	6
cps	24	109	**347**	5	720	9	556	6
$ex1010$	10	10	498	5	**194**	3	210	4
$exep$	30	63	**155**	3	202	5	203	8
$in2$	20	10	102	4	140	6	**98**	6
$intb$	15	7	319	5	**174**	6	274	9
$misex3$	14	14	303	5	161	5	**154**	6
$misj$	35	14	17	2	19	2	**16**	3
pdc	16	40	243	5	327	6	**160**	5
$spla$	16	46	282	5	301	6	**222**	5
ti	48	72	**274**	4	649	8	497	8
$tial$	14	8	332	5	**208**	5	289	8

5. *The composition function has 9-6+3=6 variables. Since the support size is equal to K=6, implement the function by a 6-LUT, which corresponds to the right-most cell in Fig.4.1.*

6. *In this way, sym12 is realized by* $3+3+1=7$ *LUTs of 6-inputs. Note that this realization is different from Fig. 3.5.*

(End of Example)

5 Experimental Results

Algorithm 4.1 was implemented in ABC [4] as command *lutmin*, applicable to logic networks in ABC. The implementation was tested on 16 benchmarks from Espresso/MCNC benchmark suites. The experiments were run on the laptop with Intel Core 2 Duo U9400 1.4GHz CPU with 4Gb RAM (only a few Mb of RAM was used). The total runtime for 16 benchmarks was 8 sec. The resulting LUT networks were verified using SAT-based combinational equivalence checking command *cec* in ABC.

Table 5.1 shows the numbers of LUTs and logic levels when $K = 6$. The columns headed with *MBC* show the results of improved version of Mishchenko *et al.* [10]. The columns headed with *LUTMIN* show the results obtained by *lutmin* (Algorithm 4.1). When the number of inputs is less than 16, Algorithm 4.1 often produced better solutions than Mishchenko *et al.*[10]. The columns headed with *SAS* shows the upper bounds obtained by using additional idea. (It obtains only the number of LUTs to show the possibility of further improvement.)

In Manohararajah *et al.* [8] and Mishchenko *et al.* [10], numbers of 6-LUTs to implement some benchmark functions are shown. For example, *ex1010*, a 10-input 10-output function. Example 2.3 shows that to implement any 10 variable function, 21 LUTs are sufficient. Thus, to implement *ex1010*, which is a 10-output function, requires at most $21 \times 10 = 210$ LUTs. Note that 1519 LUTs are used in [8] and 1059 LUTs are used in [10].

apex4 is a 9 input 19-output function. Example 2.3 shows that to implement any 9 variable function, 11 LUTs is sufficient. Thus, *apex4*, which is 19-output function, requires at most $11 \times 19 = 209$ LUTs. Note that 738 LUTs are used in [8] and 732 LUTs are used in [10]. These results show that the even the state-of-the-art logic synthesis algorithms still have room for improvements, which is also noted by Cong and Minkovich [5].

The reason why [10] failed to obtain the optimum solutions are as follows:

1. [10] uses AND inverter graphs as an initial solutions, which can lead to local minimal solutions.

2. [10] uses heuristic method to design larger circuits, and some optimization is omitted to save computation time.

It should be noted that [10] is applicable to large circuits, while Algorithm 4.1 is applicable to only small problems.

6. Conclusion and Comments

In this paper, we consider the number of 6-LUTs to implement logic functions. Also presented a synthesis method with 6-LUTs. The method is MUX-based realizations and cascade-based realizations. Although, this method produces circuits with more levels and is practical for the functions with at most 16 inputs, it often produces circuits with fewer LUTs than existing methods [10, 8].

A possible extension is to optimize encodings of the rail outputs [9]. By considering the encodings, we can often replace one or more outputs functions of LUTs by primary input variables. In such a case, the number of LUTs can be further reduced. Another extension is to extract decomposable component whose number of inputs is at most 6.

In this paper, to implement the multiple-output functions, functions are decomposed into single-output ones. However, decomposition into groups of a few functions also produces very good circuits [12]. Yet another possible extension is to use such partitions of the outputs.

7. Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, and the Grant of Knowledge Cluster Project of MEXT. Discussion with Prof. R. K. Brayton was quite useful.

References

[1] Altera, "Stratix IV FPGA Core Fabric Architecture," http://www.altera.com/

[2] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 12, No. 3, March 2004, pp.288-298.

[3] R. L. Ashenhurst, "The decomposition of switching functions," *International Symposium on the Theory of Switching*,pp. 74-116, April 1957.

[4] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 70911. http://www.eecs.berkeley.edu/ alanmi/abc/

[5] J. Cong and K. Minkovich, "Optimality study of logic synthesis for LUT-Based FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26, Issue 2, Feb. 2007, pp.230-239.

[6] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[7] V. N. Kravets, K. A. Sakallah,"Constructive library-aware synthesis using symmetries," *DATE* 2000, pp.208-213, March 2000.

[8] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 11, November 2006, pp. 2331-2340.

[9] A. Mishchenko and T. Sasao, "Encoding of Boolean functions and its application to LUT cascade synthesis, " *International Workshop on Logic and Synthesis* (IWLS2002), New Orleans, Louisiana, June 4-7, 2002, pp.115-120.

[10] A. Mishchenko, R. K. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks", *Proc. ICCAD'08*, Nov. 2008, pp.38-45.

[11] R. Murgai, R. Brayton, and A. Sangiovanni Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays*, Springer, July 1995.

[12] H. Nakahara and T. Sasao, "A PC-based logic simulator using a look-up table cascade emulator," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No.12, Dec. 2006, pp. 3471-3481.

[13] S. Panda, F. Somenzi, and B. F. Plessier,"Symmetry detection and dynamic variable ordering of decision diagrams," *ICCAD1994*, Nov. 1994, pp.628-631.

[14] C. A. Papachristou, "Characteristic measures of switching functions," *Information Science*, Vol.13,No.1, pp.51-75, 1977.

[15] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field programmable gate arrays: The effect of logic block functionality on area efficiency,"*IEEE J. Solid State Circ. 25,5*, pp. 1217-1225, Oct. 1990.

[16] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proc. IEEE*,Vol. 81, No. 7, pp.1013-1029, July 1993.

[17] T. Sasao, "FPGA design by generalized functional decomposition," In *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publisher, pp. 233-258, 1993.

[18] T. Sasao and J. T. Butler, "A Design method for look-up table type FPGA by pseudo-Kronecker expansion," *IEEE International Symposium on Multiple-Valued Logic*, Boston, May 1994, pp. 97-106.

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

[20] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *International Workshop on Logic and Synthesis (IWLS01)*, Lake Tahoe, CA, June 12-15, 2001, pp. 225-230.

[21] T. Sasao, "Analysis and synthesis of weighted-sum functions," *IEEE TCAD, Special issue on International Workshop on Logic and Synthesis*, Vol. 25, No. 5, May 2006, pp. 789 - 796.

[22] T. Sasao," A New expansion of symmetric functions and their application to non-disjoint functional decompositions for LUT-type FPGAs," *International Workshop on Logic Synthesis*, Dana Point, California, U.S.A., May 31 -June 2, 2000, pp.105-110.

[23] T. Sasao, "On the number of LUTs to realize sparse logic functions," International Workshop on Logic and Synthesis *IWLS-2009*, July 31-Aug.2, 2009, Berkeley, U.S.A..

[24] Xilinx, "Advantages of the Virtex-5 FPGA, 6-Input LUT Architecture," WP284 (v1.0), Dec. 19, 2007, http://www.xilinx.com/