

LUT Cascades and Emulators for Realization of Logic Functions

Tsutomu SASAO¹ Yukihiro IGUCHI² Munehiro MATSUURA¹

¹ Kyushu Institute of Technology, Dept. of Comput. Science & Electronics,
Iizuka 820-8502, Japan

² Meiji University, Dept. of Computer Science, Kawasaki 214-8571, Japan

July 20, 2005

Abstract

Two types of programmable logic devices using LUTs (Look-Up Tables) are presented. An LUT cascade directly implements logic functions by a series connection of LUTs, while an emulator emulates an LUT cascade by sequentially accessing LUTs. The LUT cascade is faster, but has a limited logic capability, while the emulator is slower, but has a higher logic capability. LUT cascades and emulators can be directly generated from the BDDs of target functions. Their performances are easy to estimate. The C-measure that show the complexity of LUT cascades are also presented. Functions with small C-measures have efficient LUT cascade and emulator realizations. Classes of functions that are suitable for LUT cascade and emulator realizations are also presented.

1 Introduction

Memories and PLAs (Programmable Logic Arrays) are popular devices that realize multiple-output combinational logic functions. However, when the number of inputs and/or outputs in the target function is large, these devices often require excessive amount of hardware. Thus, FPGAs (Field Programmable Gate Arrays) are often used [3].

In FPGAs, propagation delays for interconnections among logic cells are much larger than that for logic elements, so the prediction of the performance is difficult without complete physical design.

To solve these problems, an LUT cascade has been developed [12]. As shown in Fig. 1.1, an LUT cascade is a serial connection of Look-Up Tables (LUTs). LUTs are often called **cells**. The LUT cascade uses relatively large cells (10 to 15 inputs and 8 to 16 outputs), and the interconnections between cells are limited to the ad-

jacent cells in the cascade. Thus, the prediction of the performance is easy. The architecture is quite different from FPGAs, which have the two-dimensional architecture of smaller (4 to 6 inputs) LUTs. In the conventional FPGAs, the area for interconnection is fairly large, while in the LUT cascades, the area for interconnection is very small. The large area for the interconnections in an FPGA is replaced with the larger cells in the cascade.

In this paper, we first survey the architecture of LUT cascades and their emulators. Then, we show functions that are suitable for such devices.

2 LUT Cascade

2.1 Features of LUT Cascades

An LUT cascade realizes a given multiple-output function F by the structure shown in Fig. 1.1. An LUT ring is an LUT cascade where outputs of the rightmost LUT drives inputs of the left most LUT, as shown in Fig. 2.1.

The merits of LUT cascades include:

- (1) Logic synthesis is relatively easy.
- (2) Layout and wiring are very easy.
- (3) Delay estimation is easy and accurate.

The demerits of LUT cascades include:

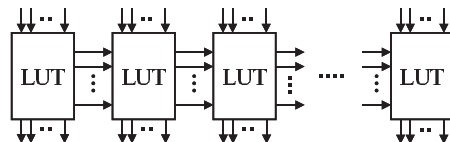


Figure 1.1: LUT cascade.

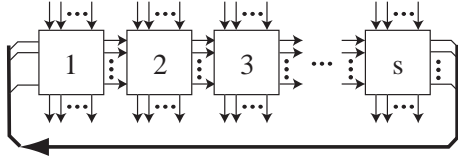


Figure 2.1: LUT ring with multiple units.

- (4) Delay can be larger than random logic networks.
- (5) Logic capability is limited once the number of inputs and outputs for each cell are fixed.

2.2 Design Method for LUT Cascades

The LUT cascade is obtained from a BDD (Binary Decision Diagram) by iterative functional decompositions [12]. The logic synthesis algorithm reduces the necessary amount of memory to represent f by detecting repeated patterns in the logic function. The wires connecting adjacent cells are called **rails**. In the design of an LUT cascade, the reduction of the number of rails is very important, since it is directly related to the size of the subsequent cell. These problem are discussed in Sec. 4.

3 Emulators for LUT Cascades and LUT Rings

3.1 Emulator with a Single Unit

In an LUT cascade, once the number of inputs and outputs for each cell are fixed, only a limited class of functions can be realized. Thus, the **LUT cascade emulator**¹ having the architecture shown in Fig. 3.1 has been developed [8]. It consists of a large memory that stores the data for cells, a programmable interconnection network, and a control circuit. It emulates an LUT cascade sequentially. Although the emulator is slower than the LUT cascade, it has higher logic capabilities than the LUT cascade.

The Shifter that drives the programmable interconnection network is used for memory packing, while the Shifter that drives the output register is used to accumulate the outputs.

Example 3.1 *Let us emulate the LUT cascade with four cells shown in Fig. 3.2 by the emulator with a single unit.*

¹ In the previous publication [12], the emulator was called the LUT cascade. However, in this paper, the sequential circuit that emulates an LUT cascade is called an emulator.

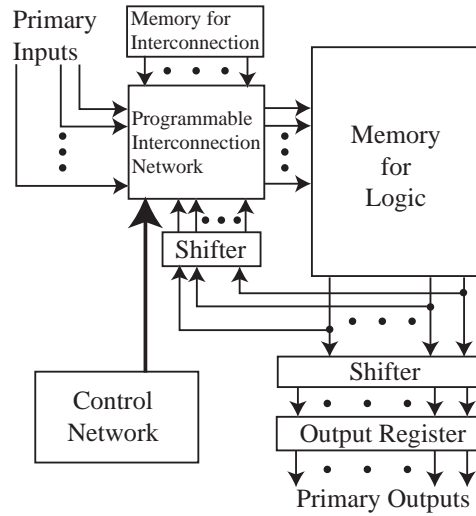


Figure 3.1: Emulator with a single unit.

Step 1 To emulate Cell₁, the two most significant bits of address are set to (0, 0) to specify the 1st page. Also, the values of X_1 are set to the lower address bits through the programmable interconnection network, as shown in Fig. 3.2(a). By reading the content of the 1st page, we obtain the outputs of Cell₁.

Step 2 To emulate Cell₂, the two most significant bits of address are set to (0, 1) to specify the 2nd page. Also, the values of X_2 are set to the middle address bits, and the outputs of Cell₁ are connected to the least significant bits through the programmable interconnection network, as shown in Fig. 3.2(b). By reading the content of the 2nd page, we obtain the outputs of Cell₂.

Step 3 To emulate Cell₃, the two most significant bits of address are set to (1, 0) to specify the 3rd page. Also, the values of X_3 are set to the middle address bits, and the outputs of Cell₂ are connected to the least significant bits through the programmable interconnection network, as shown in Fig. 3.2(c). By reading the content of the 3rd page, we obtain the outputs of Cell₃.

Step 4 To emulate Cell₄, the two most significant bits of address are set to (1, 1) to specify the last page. Also, the values of X_4 are set to the middle address bits, and the outputs of Cell₃ are connected to the least significant bits through the programmable interconnection network, as shown in Fig. 3.2(d). By reading the content of the 4th page, we obtain the outputs of Cell₄. (End of Example)

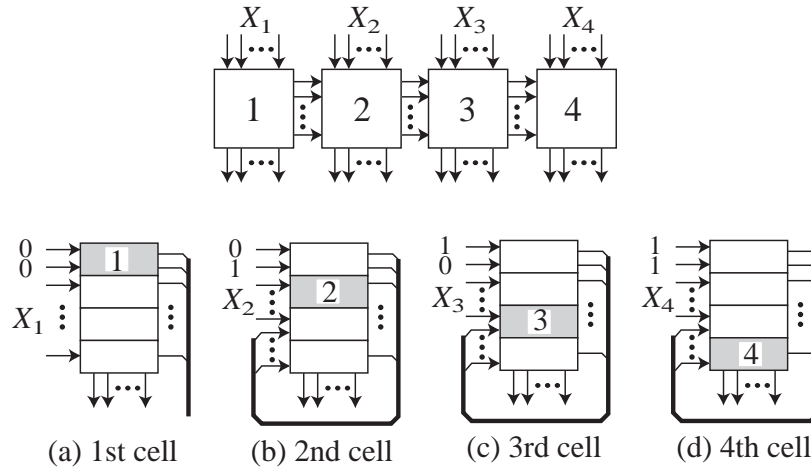


Figure 3.2: Operation of an emulator with a single unit.

To reduce the amount of steps and/or memory in the emulator, we have two methods:

- (1) Use cells with different sizes. The straightforward way to design an emulator is to use cells with the same number of inputs. However, we can often reduce memory by using cells with different number of inputs. Also, by using cells with a different number of inputs, we can often reduce the number of cells to make the emulator faster without increasing the total amount of memory.
- (2) **Memory packing** [14]. In an emulator, all outputs of a cell must be read simultaneously. Thus, they must be stored in the same page of the memory. However, if there is any vacancy, the data for multiple cells can be stored in the same page. By using this property, we can reduce the required amount of memory.

Example 3.2 Fig. 3.3 shows an example of an LUT cascade with four cells: $Cell_1$ has 4 inputs and 2 outputs, $Cell_2$ has 3 inputs and 4 outputs, $Cell_3$ has 3 inputs and 3 outputs, and $Cell_4$ has 4 inputs and 2 outputs.

Let us realize the function by using an emulator. Assume that page sizes of the memory for logic are $2^4 = 16$ words. Fig. 3.4(a) shows the memory map, where shaded areas are unused.

We can reduce the size of the memory for logic by memory packing. Fig. 3.4(b) shows the packed memory map. Note that in this map, only five bits are used in the address. To use this memory map, we operate the emulator as follows:

Step 1 To emulate $Cell_1$, the most significant bit of address is set to (0). Also, the values of $X_1 =$

(x_1, x_2, x_3, x_4) are set to the lower address bits. By reading the memory, we obtain the outputs of $Cell_1$.

Step 2 To emulate $Cell_2$, two most significant bits of address are set to (1, 0). Also, the value of $X_2 = (x_5)$ is set to the middle address bit, and outputs of $Cell_1$ are connected to the least significant bits. By reading the memory, we obtain the outputs of $Cell_2$.

Step 3 To emulate $Cell_3$, two most significant bits of address are set to (1, 1). Also, the value of $X_3 = (x_6)$ is set to the middle address bit, and the outputs of $Cell_2$ are connected to the least significant bits. By reading the memory, we obtain the outputs of $Cell_3$.

Step 4 To emulate $Cell_4$, the most significant bit of address is set to (0). Also, the values of $X_4 = (x_7, x_8)$ are set to the middle address bits, and the outputs of $Cell_3$ are connected to the least significant bits. By reading the 4th page, we obtain the outputs of $Cell_4$. (End of Example)

3.2 Emulator with Multiple Units

Although the emulator with a single unit is simple, it has several drawbacks. First, it dissipates high power. To emulate an LUT cascade with s cells, we have to access the large memory unit s times. Also, it is slow, since both the memory unit and the programmable interconnection network are large.

We can avoid such problems by using a multiple units in the emulator.

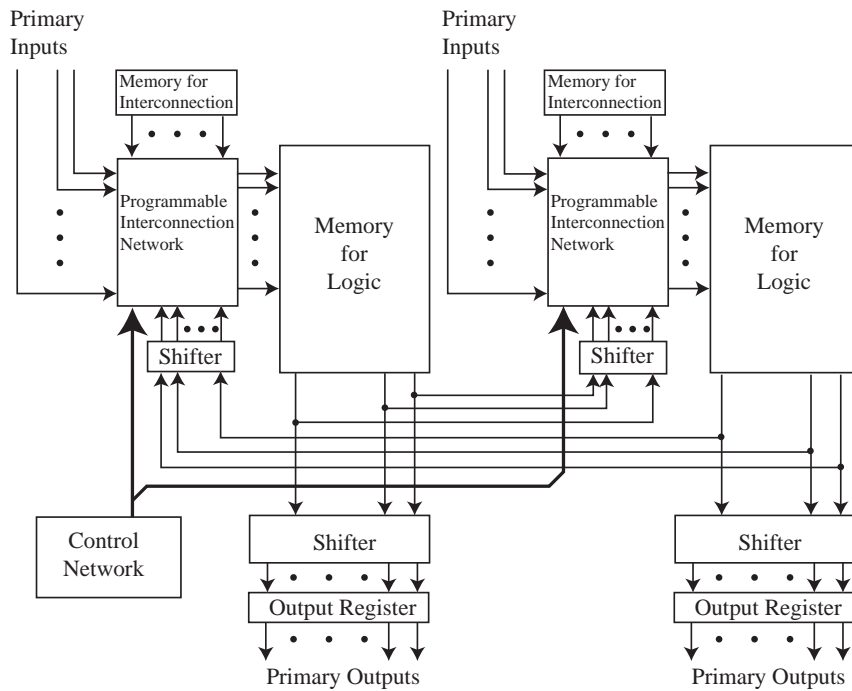


Figure 3.5: Emulator with two units.

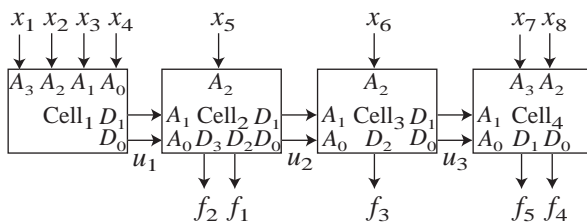


Figure 3.3: Example of LUT cascade.

Fig. 3.5 shows an emulator with two units. Each unit has a programmable interconnection and a memory. The emulator with multiple units can be lower power since at most one unit can be in the high-power (active) mode, and the other units can be in the low-power (stand-by) mode. Also, the emulator with multiple units can have higher throughput than the emulator with single unit, since they can work simultaneously.

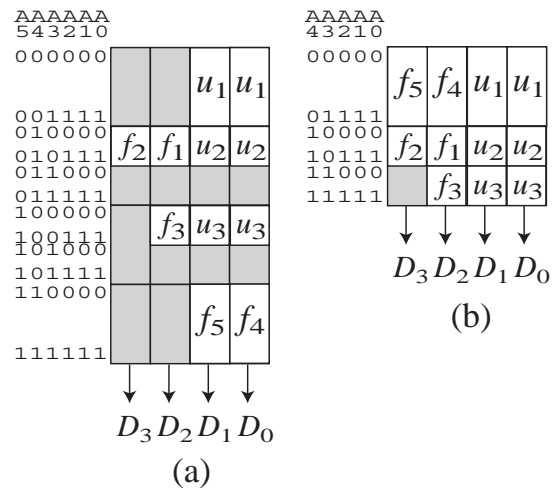


Figure 3.4: Memory map for packing.

4 Complexities of LUT Cascades

4.1 Functional Decomposition and LUT Cascades

Fig. 2.1 shows an LUT ring with s units, where each unit may have two or more pages.

An arbitrary logic function can be implemented by a single memory. However, with the increase of the num-

ber of input variables, the size of the memory increases exponentially.

Definition 4.1 [15] Consider a function $\vec{F}(\vec{X}) : B^n \rightarrow B^q$, where $B = \{0,1\}$ and $\vec{X} = (x_1, x_2, \dots, x_n)$. Let (\vec{X}_L, \vec{X}_H) be a partition of \vec{X} , where $\vec{X}_L = (x_0, x_1, \dots, x_{n_L-1})$ and $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \dots, x_{n-1})$. The **decomposition chart** for f is a two-dimensional matrix, where the column labels have all possible assignments of elements of B to \vec{X}_L , the row labels have all possible assignments of elements of B to \vec{X}_H , and the corresponding matrix value is equal to $\vec{F}(\vec{X}_L, \vec{X}_H)$. Among the decomposition charts for \vec{F} , the one whose column label values and row label values increase when the label moves from the left to the right, and from the top to the bottom, is the **standard decomposition chart**. The number of different column patterns in the decomposition chart is the **column multiplicity of the decomposition chart**. Here, we also consider the case, where $\vec{X}_L = X$.

Note that there are n different standard decomposition charts for an n -variable function $\vec{F}(\vec{X})$.

Definition 4.2 The C-measure of a logic function $f(x_1, x_2, \dots, x_n)$ is the maximum value of the column multiplicities of all possible standard decomposition charts for $f(x_1, x_2, \dots, x_n)$.

The column multiplicity of a decomposition chart is equal to the width of the MTBDD [10]. So, the C-measure of a logic function is equal to the maximum width of the MTBDD for a given ordering of the input variables. For a given logic function $f(x_1, x_2, \dots, x_n)$, the C-measure is easy to obtain and is uniquely defined. Functions having small C-measures have efficient LUT cascade realizations. So, it is a **complexity measure** of the logic function for implementing by an LUT cascade. For example, adders and symmetric functions have small C-measures, while multipliers and random functions have large C-measures for any permutation of the input variables. Fortunately, many practical functions have small C-measures, and thus have efficient LUT cascade realizations.

When a function has many outputs, the C-measures tend to be large. So, in such cases, we can partition the outputs into several groups, and realize each group by a separate LUT cascade. For example, Fig. 4.1 shows the realization, where the outputs are partitioned into three groups. When these functions are implemented by an emulator, to evaluate all the outputs, we need 9 table look-ups.

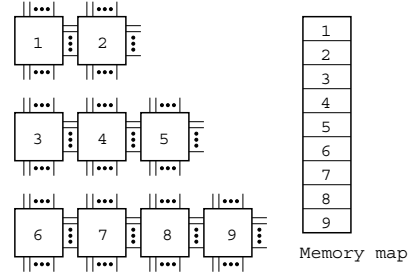


Figure 4.1: Realization of logic functions by separate cascades.

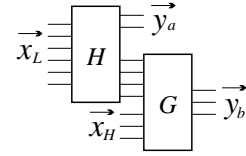


Figure 4.2: Realization of logic functions by decomposition.

Theorem 4.1 For a given function f , let \vec{X}_L be the variables labeling the columns, and let \vec{X}_H be the variables labeling the rows, and let μ be the column multiplicity of the decomposition chart. Then, the function f is realizable with the network shown in Fig. 4.2. In this case, the number of signal lines that connect two blocks H and G is $\lceil \log_2 \mu \rceil$.

When the number of signal lines that connect two blocks is smaller than the number of variables in \vec{X}_L , we can often reduce the size of memory to implement the function. This technique is **functional decomposition** [1].

By applying functional decomposition repeatedly to the given function, we have the **LUT cascade** [12] shown in Fig. 1.1. The cascade consists of **cells**, and the wires connecting adjacent cells are **rails**. Functions with small C-measures have compact LUT cascade realizations. To derive C-measures, we need not use decomposition charts. We can efficiently obtain the C-measure by the **binary decision diagram (BDD_for_CF)** that represents the characteristic function for the multiple-output function [13, 17]. The C-measure is equal to the maximum width of the BDD [10].

Theorem 4.2 [12] Let μ be the C-measure of the function f . Then, f can be implemented by the LUT cascade consisting of cells with at most $\lceil \log_2 \mu \rceil + 1$ inputs and $\lceil \log_2 \mu \rceil$ outputs.

Theorem 4.3 Consider an LUT cascade for a function f . Let n be the number of primary inputs, s be the num-

ber of cells, r be the maximum number of rails (i.e., the number of lines between cells), k be the maximum number of inputs of a cell, μ be the C-measure of the BDD for f , and $k \geq \lceil \log_2 \mu \rceil + 1$. Then, there is an LUT cascade for f that satisfies the relation:

$$s \leq \left\lceil \frac{n-r}{k-r} \right\rceil$$

4.2 Functions with Small C-Measures

Mathematically, for the most functions, C-measures increase exponentially with the number of input variables n . However, many practical functions have small C-measures. The C-measures of the following functions are small, and these functions can be implemented by LUT cascades efficiently.

4.2.1 Segment Index Logic Function

Definition 4.3 A **Segment Index Encoder function** [15] is a mapping: $SIE : I \rightarrow I$, where I is a set of integers, and $a \geq b$ implies $SIE(a) \geq SIE(b)$. **Segment index logic function** is the SIE function represented by binary variables.

Theorem 4.4 Let p be the number of segments in an SIE function. Then, the C-measure of the SIE function is at most $p + 1$ [15].

4.2.2 WS Function

A WS function [18] is a mathematical model of bit counting circuits [11], code converters [16], and distributed arithmetic [19], etc .

Definition 4.4 An n -input **WS function** $\vec{F}(\vec{X})$ computes

$$WS(\vec{X}) = \sum_{i=1}^n w_i \cdot x_i, \quad (4.1)$$

where $\vec{X} = (x_1, x_2, \dots, x_n)$ is the **input vector**, $\vec{W} = (w_1, w_2, \dots, w_n)$ is the **weight vector**, and w_i ($i = 1, 2, \dots, n$) is an integer. Let $\vec{F} = (f_{q-1}, f_{q-2}, \dots, f_0)$ be the binary representation of the WS function. Then,

$$WS(\vec{X}) = \sum_{i=0}^{q-1} f_i(\vec{X}) \cdot 2^i. \quad (4.2)$$

Theorem 4.5 [18] The C-measure of a WS function with the weight vector $\vec{W} = (w_1, w_2, \dots, w_n)$, is at most $UB1 = 1 + \sum_{j=1}^n |w_j|$.

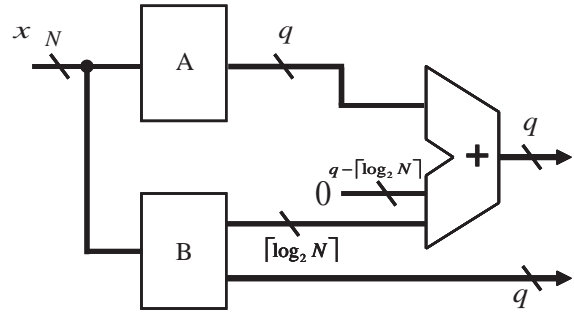


Figure 4.3: Arithmetic Decomposition of $2q$ -output WS Function.

When the sum of the weights is large, a monolithic cascade implementation of a WS function can be large, since the C-measure of a WS function can increase exponentially with n . In this case, we can partition the outputs into groups, and implement each group separately [18].

Theorem 4.6 Let $\vec{F}_{LSB}(\vec{X})$ be the logic function that represents the least significant q bits of a WS function. Then, $\vec{F}_{LSB}(\vec{X})$ can be realized with the LUT cascade consisting of cells with $q + 1$ inputs and q outputs.

A $2q$ -output WS function can be decomposed into a pair of WS functions as follows: Let, w_i be a weight of a $2q$ -output WS function. Then, w_i can be written as

$$w_i = 2^q w_{Ai} + w_{Bi},$$

where w_{Ai} denotes most significant q bits, and w_{Bi} denotes least significant q bits. In this case, we can implement the $2q$ -output WS function by using a pair of WS functions and an adder, as shown in Fig. 4.3.

Theorem 4.7 [19] A $2q$ -output WS function $\vec{F}(\vec{X})$ that represents

$$\sum_{i=0}^{n-1} w_i x_i$$

can be decomposed into a pair of WS functions $\vec{F}_A(\vec{X})$ and $\vec{F}_B(\vec{X})$, where $\vec{F}_A(\vec{X})$ is a q -output WS function representing

$$\sum_{i=0}^{n-1} w_{Ai} x_i,$$

and $\vec{F}_B(\vec{X})$ is a $q + \lceil \log_2 n \rceil$ -output WS function representing

$$\sum_{i=0}^{n-1} w_{Bi} x_i,$$

and

$$w_i = 2^q w_{Ai} + w_{Bi}.$$

This is an **arithmetic decomposition of a WS function**.

4.2.3 Threshold Function

Definition 4.5 A **threshold function** $f(x_1, x_2, \dots, x_n)$ satisfies the relation:

$$f = 1 \text{ if } \sum_{i=1}^n w_i x_i \geq T, \text{ and}$$

$$f = 0 \text{ otherwise,}$$

where (w_1, w_2, \dots, w_n) are **weights** and T is the **threshold**.

Theorem 4.8 The C-measure of a threshold function with weights (w_1, w_2, \dots, w_n) is at most $1 + \sum_{i=1}^n w_i$.

(Proof) It is clear that the C-measure of f is not greater than that of the WS function having the same weights. By Theorem 4.5, the C-measure of the WS function is at most $1 + \sum_{i=1}^n w_i$. Hence, we have the theorem. (Q.E.D.)

4.2.4 Symmetric Function

Definition 4.6 A function $f(x_1, x_2, \dots, x_n)$ is **symmetric** if any permutation of the variables does not change the function.

Theorem 4.9 The C-measure of an n -variable symmetric function is at most $n + 1$.

(Proof) Consider the WS function f , where $w_i = 1$ for all i . It is clear that the C-measure of a symmetric function is not greater than that of the WS function. By Theorem 4.5, we can show that the C-measure is at most $1 + \sum_{i=1}^n w_i = 1 + n$. Hence, we have the theorem. (Q.E.D.)

Thus, a symmetric function is quite easy to implement by an LUT cascade.

5 Conclusions and Comments

In this paper, we presented two types of programmable logic devices that use LUTs: The LUT cascade and the emulator. Both devices use multiple-output LUTs with larger number of inputs than conventional FPGAs [9].

The LUT cascade is a combinational circuit, while the emulator is a sequential circuit. They have regular structures and quite promising for future LSIs [4]. We also defined C-measure that shows the complexity of LUT cascades. We showed classes of functions with small C-measures. They include SIE functions, WS functions, threshold functions, and symmetric functions. All are efficiently implemented by LUT cascades. Logic design system of LUT cascades and emulator are now under development [14, 13, 17, 6]. Also, a prototype of LUT cascade chip has been fabricated by using 0.35 micron CMOS process [7].

Acknowledgment

This work was supported by a grant from the Japanese Ministry of MEXT via Kitakyushu innovative cluster project, and the Aid for Scientific Research of the Japan Society for the Promotion of science (JSPS). Discussion with Jon T. Butler improved English presentation.

References

- [1] R. L. Ashenurst, "The decomposition of switching functions," *In Proceedings of an International Symposium on the Theory of Switching*, pp. 74-116, April 1957.
- [2] H. A. Curtis, *A New Approach to The Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.
- [3] S. D. Brown, R. J. Fancis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.
- [4] S. Hassoun and T. Sasao (eds.), *Logic Synthesis and Verification*, Kluwer Publishers, 2001.
- [5] K. Hosaka, Y. Takenaga, T. Kaneda, and S. Yajima, "Size of ordered binary decision diagrams representing threshold functions," *Theoretical Computer Science*, Vol. 180, pp. 47-60, 1997.
- [6] H. Nakahara, T. Sasao, and M. Matsuura, "A design algorithm for sequential circuits using LUT rings," SASIMI2004, pp.430-437.
- [7] K. Nakamura, T. Sasao, M. Matsuura, K. Tanaka, K. Yoshizumi, H. Qin, and Y. Iguchi, "Programmable logic device with an 8-stage cascade of 64K-bit asynchronous SRAMs," *Cool Chips VIII*, IEEE Symposium on Low-Power and High-Speed Chips, April 20-22, 2005, Yokohama, Japan.
- [8] H. Qin, T. Sasao, M. Matsuura, S. Nagayama, K. Nakamura, and Y. Iguchi, "A realization of multiple-output functions by look-up table rings", *IEICE Transactions on Fundamentals*, Vol.E87-A, Dec. 2004, pp. 3141-3150.

- [9] 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 Journal of Solid-State Circuits*, Vol. 25, No. 5, pp. 1217-225, Oct. 1990.
- [10] T. Sasao, "FPGA design by generalized functional decomposition," In *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publisher, pp. 233-258, 1993.
- [11] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.
- [12] 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.
- [13] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *41st Design Automation Conference*, San Diego, CA, USA, June 2-6, 2004, pp. 428-433.
- [14] T. Sasao, M. Kusano, and M. Matsuura, "Optimization methods in look-up table rings," *International Workshop on Logic and Synthesis (IWLS-2004)*, Temecula, California, U.S.A., June 2-4, 2004.
- [15] T. Sasao, J. T. Butler, and M. Riedel, "Application of LUT cascades to numerical function generators," *12th SASIMI Workshop*, Kanazawa, Japan, Oct 18-19, 2004, pp. 422-429.
- [16] T. Sasao, "Radix converters: Complexity and implementation by LUT cascades," *International Symposium on Multiple-Valued Logic*, Calgary, Canada, May 18-21, 2005, pp.265-263.
- [17] T. Sasao and M. Matsuura, "BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition," *Design Automation Conference*, June 2005, pp.373-378.
- [18] T. Sasao, "Analysis and synthesis of weighted-sum functions," *International Workshop on Logic and Synthesis*, Lake Arrowhead, CA, USA, June 8-10, 2005, pp.455-462.
- [19] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT cascade realizations of FIR filters," *DSD'2005, 8th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, Porto, Portugal, Aug. 30 - Sept. 3, 2005. (To be published).