# An Application of 16-Valued Logic to Design of Reconfigurable Logic Arrays

Tsutomu Sasao
Department of Computer Science and Electronics,
Kyushu Institute of Technology,
Iizuka 820-8502, Japan

## Abstract

*This paper presents a method to implement a reconfigurable logic array by using FPGA. 16-valued logic is introduced to design circuits with 2-valued 4-input LUTs. Symmetric functions and adders can be efficiently represented, as well as benchmark functions. Comparisons with 2-valued expressions and 4-valued expressions are done. Both sum-of-products expressions and EXOR sum-of-products expressions of 16-valued logic significantly reduces needed FPGA resources.*

## 1. Introduction

With the increase of complexity of the digital systems, the time and cost to develop LSIs have increase tremendously. On the other hand, the life of the products tends to be short because of constantly evolving product lines, especially for consumer appliances such as mobile phones and audiovisual equipments.

Architecture that is dynamically reconfigurable is one way to solve this problem. With FPGAs, we can reconfigure the entire device. However, it may not be necessary to reconfigure the entire device. In most cases, a minor modification is sufficient.

In this paper, we consider dynamic reconfigurable circuits, where only the logic part is reconfigured, but the interconnection part is fixed. Dynamic reconfigurable PLAs and CAMs are examples of such circuits.

In this paper, we present a reconfigurable logic array that is suitable for FPGA implementation. It uses 16-valued logic to design the circuit, and is logically more capable than PLAs or CAMs.

## 2. Realization of 16-Valued Expression on an FPGA

In this paper, we assume Xilinx Virtex FPGAs. Fig. 2.1 shows the architecture of an FPGA: It consists of many configurable logic blocks (*CLBs*). Fig. 2.2 shows the structure of a CLB: It consists of a pair of *Slices*, and each Slice consists of a pair of 4-input look-up tables (*LUTs*) together with carry circuits, control logic, and flip-flops.

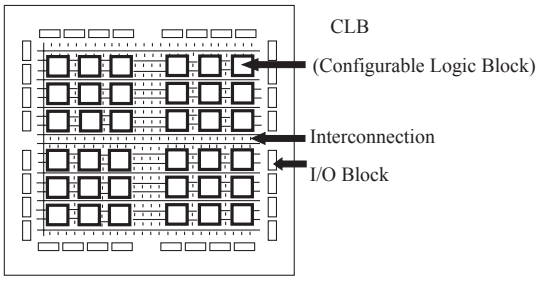


**Figure 2.1. Architecture of FPGA.**

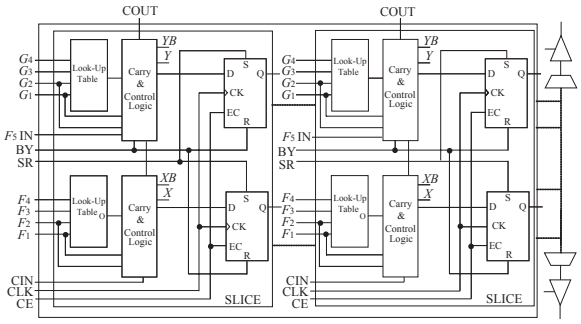


**Figure 2.2. CLB Structure.**

As shown in Fig. 2.3, a 4-input LUT cell has two modes:

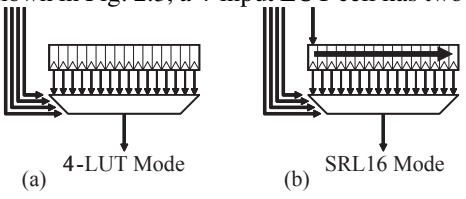


**Figure 2.3. Two operation modes of an LUT cell.**

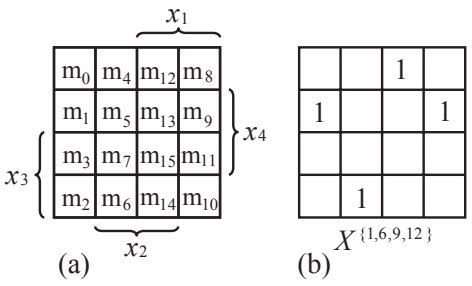


**Figure 2.4. Map for 4-variable Function.**

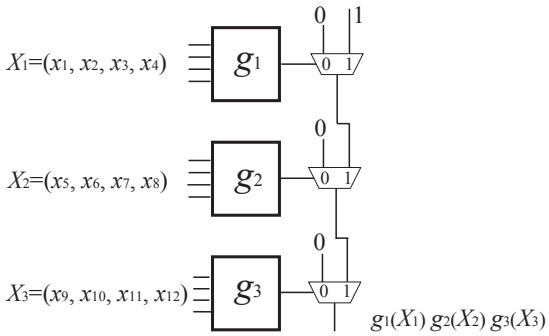


**Figure 2.5. Realization of AND with MUX.**

*4-LUT mode* and *SRL16 mode*. To reconfigure the logic of an LUT, at least 16 clocks are required. This is much faster than reconfiguring the entire FPGA, including interconnections.

Next, we will introduce 16-valued logic to design 4-input LUT circuits. A 4-input LUT realizes an arbitrary function of four variables. Fig. 2.4 (a) shows a map of a 4-variable function. An arbitrary 4-variable logic function can be viewed as a subset of 16 minterms $\{m_0, m_1, \ldots, m_{15}\}$. For example, the function in Fig. 2.4 (b) can be represented by a set of four minterms $\{m_1, m_6, m_9, m_{12}\}$. Instead of using a set of minterms, we can use a 16-valued literal. Let $X = (x_1, x_2, x_3, x_4)$. Then, the function in Fig. 2.4 (b) can be represented by the literal $X^{\{1,6,9,12\}}$. This literal specifies that the function is 1 if and only if the input combination $X = (x_1, x_2, x_3, x_4)$ represents either 1, 6, 9, or 12. In this case, four variables are treated together as $X = (x_1, x_2, x_3, x_4)$, and X is considered as a 16-valued variable.

By using multiplexers in the carry and control circuit of an FPGA, the logical AND of 4-variable functions can be implemented. For example, as shown in Fig. 2.5, three LUTs are connected by the MUX chain to realize the logical product $g_1(X_1)g_2(X_2)g_3(X_3)$, where $X_1 = (x_1, x_2, x_3, x_4), X_2 = (x_5, x_6, x_7, x_8)$, and $X_3 = (x_9, x_{10}, x_{11}, x_{12})$. This realization of the AND requires no LUT and is much faster than one using an LUT [13].

Thus, Fig. 2.5 implements a product of 16-valued literals of the form $X_1^{S_1} X_2^{S_2} X_3^{S_3}$, where $S_i \subseteq P$ and $P = \{0, 1, 2, \ldots, 15\}$. Fig. 2.6 shows a circuit for a sum-of-products expression (SOP), where the OR gates are implemented by LUTs. Alternatively, Fig. 2.7 shows a circuit for an EXOR sum-of-products expression (ESOP), where the EXOR gates are implemented by LUTs. In these figures, the top horizontal line denotes the constant 1 line. Thick horizontal lines denote bundles of four binary lines. Each horizontal bundle can be viewed as carrying a 16-valued variable, and each column realizes a product of up to four 16-valued literals.

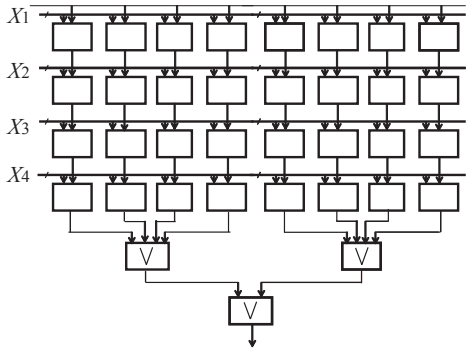


**Figure 2.6. Realization of SOP.**

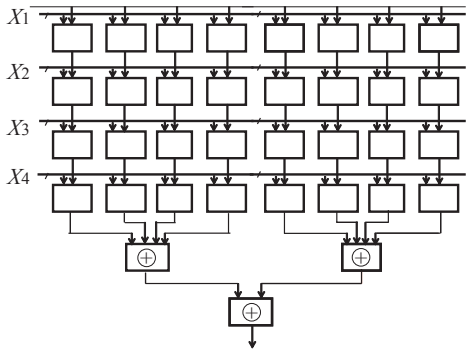


**Figure 2.7. Realization of ESOP.**

To realize a multiple-output function, a programmable OR (EXOR) gate can be used for each output. For example, a two-output function can be implemented by the circuit shown in Fig. 2.8. In this figure, multiple-output LUTs implemented by embedded RAMs are used [1]. In this way, 16-valued SOPs or ESOPs for multiple-output function can be easily realized. In this paper, the architectures shown in Figs. 2.6, 2.7 and 2.8 are called **16-valued reconfigurable logic arrays**.

Interesting questions are

1. How many products are necessary to represent functions by using 16-valued expressions?

[1] When embedded RAMs are used as logic elements, a clock pulse is necessary.

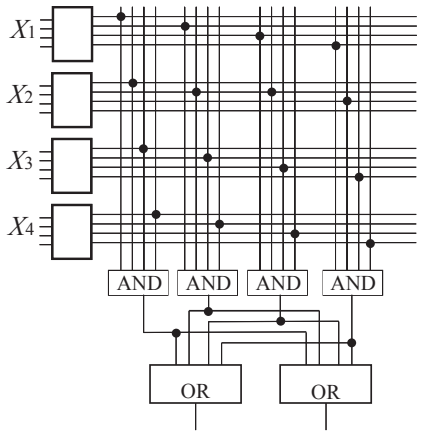


**Figure 2.8. Realization of Multiple-output Function.**

2. Are the reconfigurable logic array implementations of benchmark functions realistic?

## 3. Definition and Basic Properties

In preparation for a discussion of a design method, we need some definitions. They are extensions of 2-valued SOPs to multi-valued ones [6, 9, 5].

**Definition 3.1** *A mapping $f : P^n \rightarrow B$ is a* **p-valued input 2-valued output function***, where $P = \{0, 1, \ldots, p-1\}$ and $B = \{0, 1\}$. Let $X$ be a variable that takes its value from $P = \{0, 1, \ldots, p-1\}$. 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 $S_i \subseteq P (i = 1, 2, \ldots, n)$, 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 $S_i = P$, ${X_i}^{S_i} = 1$ and the logical product is independent of $X_i$. In this case, literal ${X_i}^P$ is redundant and can be deleted. A logical product is also called a* **term***, or a* **product term**. *When $|S_i| = 1$ for $(i = 1, 2, \ldots, n)$, a logical product corresponds to an element of the domain. This product is a* **minterm**. *When $S_i = P$ $(i = 1, 2, \ldots, n)$, the logical product corresponds to the constant 1. When $p = 2$, a function is a 2-valued logic function. When we consider 2-valued logic functions only, we often represent the literal $X^{\{0\}}$ by $\overline{X}$, and $X^{\{1\}}$ by $X$. In an SOP, replacing the OR operators with the EXOR operators will produce an* **EXOR sum-of products expression** *(**ESOP**).*

An arbitrary multi-valued input 2-valued output function is represented by an SOP (ESOP). Many SOPs (ESOPs) exist that represent the same function. Among them, the one with the minimum number of products is the **minimum SOP** (**minimum ESOP**). MINI [2] and ESPRESSO-MV [5] are SOP minimizers, while EXMIN2 [7] and EXORCISM-MV [11] are ESOP minimizers.

## 4. Multiple-Output Function

For multiple-output functions, independent minimization of each output does not always produce exact minimum solutions. To minimize (or simplify) expressions for multi-output functions, we have to consider all the outputs at the same time. We can minimize the total number of products by minimizing the expressions of the characteristic function of the multiple output function.

**Definition 4.1** *When a multiple-output function has $m$ binary functions, each can be represented as $f_j(X_1, X_2, \ldots, X_n)$ $(j = 0, 1, \ldots, m-1)$. Then, the 2-valued output function $F(X_1, X_2, \ldots, X_n, X_{n+1})$, where $F(X_1, X_2, \ldots, X_n, j) = f_j(X_1, X_2, \ldots, X_n)$, is the* **characteristic function for multi-output function**. *Here, $X_{n+1}$ is the $m$ valued variable representing outputs. In other words, $F(a_1, a_2, \ldots, a_n, j) = 1 \Leftrightarrow (x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n, f_j = 1)$.*

**Example 4.1** *Let us obtain 16-valued representations of the 4-bit adder (ADR4) shown below:*

$$\begin{array}{cccccc} & x_3 & x_2 & x_1 & x_0 \\ +) & y_3 & y_2 & y_1 & y_0 \\ \hline z_4 & z_3 & z_2 & z_1 & z_0 \end{array}$$

*$z_0$, $z_1$, $z_2$, $z_3$ and $z_4$ are represented by logical expressions:*

$$\begin{aligned} z_0 &= (x_0 \oplus y_0), \\ z_1 &= (x_1 \oplus y_1) \oplus (x_0 y_0), \\ c_1 &= x_1 y_1 \vee (x_1 \vee y_1)(x_0 y_0), \\ z_2 &= (x_2 \oplus y_2) \oplus c_1 \\ z_3 &= (x_3 \oplus y_3) \oplus (x_2 y_2 \vee (x_2 \vee y_2) c_1), \text{ and} \\ z_4 &= (x_3 y_3) \vee (x_3 \vee y_3)(x_2 y_2 \vee (x_2 \vee y_2) c_1). \end{aligned}$$

*Let $X_1 = (x_1, y_1, x_0, y_0)$ and $X_2 = (x_3, y_3, x_2, y_2)$. Then, we have the 16-valued expression of the 4-bit adder:*
**SOP**

$$\begin{aligned} z_0 &= X_1^{\{1,2,5,6,9,10,13,14\}}, \\ z_1 &= X_1^{\{3,4,5,6,8,9,10,15\}}, \\ c_1 &= X_1^{\{7,11,12,13,14,15\}}, \\ z_2 &= X_2^{\{1,2,5,6,9,10,13,14\}} \oplus X_1^{\{7,11,12,13,14,15\}} \\ &= X_1^{\{7,11,12,13,14,15\}} X_2^{\{0,3,4,7,8,11,12,15\}} \vee \\ &\quad X_1^{\{0,1,2,3,4,5,6,8,9,10\}} X_2^{\{1,2,5,6,9,10,13,14\}}, \\ z_3 &= X_1^{\{7,11,12,13,14,15\}} X_2^{\{1,2,3,4,8,13,14,15\}} \vee \\ &\quad X_1^{\{0,1,2,3,4,5,6,8,9,10\}} X_2^{\{3,4,5,6,8,9,10,15\}}, \text{ and} \\ z_4 &= X_2^{\{7,11,12,13,14,15\}} \vee \\ &\quad X_1^{\{7,11,12,13,14,15\}} X_2^{\{5,6,7,9,10,11,12,13,14,15\}}. \end{aligned}$$

*Let $X_3$ be a five-valued variable that represents the output part, then the positional cubes[9] of the characteristic function for the multiple-output function are:*

$$\begin{array}{cccc} X_1 & & X_2 & X_3 \\ 0000000000111111 & & 0000000000011111 & \texttt{zzzzz} \\ 0123456789012345 & & 0123456789012345 & 43210 \\ \left[\begin{array}{cccc} 0110011001100110 & - & 1111111111111111 & - & 00001 \\ 0001111011100001 & - & 1111111111111111 & - & 00010 \\ 0000000100011111 & - & 1001100110011001 & - & 00100 \\ 1111111011100000 & - & 0110011001100110 & - & 00100 \\ 0000000100011111 & - & 0111100010000111 & - & 01000 \\ 1111111011100000 & - & 0001111011100001 & - & 01000 \\ 1111111111111111 & - & 0000000100011111 & - & 10000 \\ 0000000100011111 & - & 0000011101111111 & - & 10000 \end{array}\right] \end{array}$$

*Note that this is a minimum SOP. It has 8 products.*

**ESOP** *When products are mutually disjoint, the OR operator in an SOP can be replaced by the EXOR operator. Also, by using the relation [7]:*

$$X_1^A X_2^B \oplus X_1^C X_2^D = X_1^{(A \oplus C)} X_2^B \oplus X_1^C X_2^{(B \oplus D)},$$

*we have the following ESOPs:*

$$\begin{aligned} z_0 &= X_1^{\{1,2,5,6,9,10,13,14\}}, \\ z_1 &= X_1^{\{3,4,5,6,8,9,10,15\}}, \\ c_1 &= X_1^{\{7,11,12,13,14,15\}}, \\ z_2 &= X_2^{\{1,2,5,6,9,10,13,14\}} \oplus X_1^{\{7,11,12,13,14,15\}} \\ &= X_1^{\{7,11,12,13,14,15\}} X_2^{\{0,3,4,7,8,11,12,15\}} \oplus \\ &\quad X_1^{\{0,1,2,3,4,5,6,8,9,10\}} X_2^{\{1,2,5,6,9,10,13,14\}}, \\ z_3 &= X_1^{\{7,11,12,13,14,15\}} X_2^{\{1,2,3,4,8,13,14,15\}} \oplus \\ &\quad X_1^{\{0,1,2,3,4,5,6,8,9,10\}} X_2^{\{3,4,5,6,8,9,10,15\}}, \\ &= X_2^{\{1,3,4,8,13,14,15\}} \oplus \\ &\quad X_1^{\{0,1,2,3,4,5,6,8,9,10\}} X_2^{\{1,2,5,6,8,9,10,13,14\}}, \text{ and} \\ z_4 &= X_1^{\{0,1,2,3,4,5,6,8,9,10\}} X_2^{\{7,11,12,13,14,15\}} \oplus \\ &\quad X_1^{\{7,11,12,13,14,15\}} X_2^{\{5,6,7,9,10,11,12,13,14,15\}} \\ &= X_2^{\{7,11,12,13,14,15\}} \oplus X_1^{\{7,11,12,13,14,15\}} X_2^{\{5,6,9,10\}}. \end{aligned}$$

*Note that $z_2$ and $z_3$ share a product. Thus, we need 7 different products. The positional cubes of minimum ESOP are:*

$$\begin{array}{cccc} X_1 & & X_2 & X_3 \\ 0000000000111111 & & 0000000000011111 & \texttt{zzzzz} \\ 0123456789012345 & & 0123456789012345 & 43210 \\ \left[\begin{array}{cccc} 0110011001100110 & - & 1111111111111111 & - & 00001 \\ 0001111011100001 & - & 1111111111111111 & - & 00010 \\ 0000000100011111 & - & 1001100110011001 & - & 00100 \\ 1111111011100000 & - & 0110011001100110 & - & 01100 \\ 1111111111111111 & - & 0111100010000111 & - & 01000 \\ 1111111111111111 & - & 0000000100011111 & - & 10000 \\ 0000000100011111 & - & 0000011001100000 & - & 10000 \end{array}\right] \end{array}$$

*(End of Example)*

## 5. Complexity of 16-Valued Expressions

**Theorem 5.1** *An arbitrary function of $n = 4r$ variables can be represented by a 16-valued SOP (ESOP) with at most $2^{n-4}$ products.*

(Proof) An arbitrary function can be represented by

$$f(X_1, X_2, \ldots, X_r) = \bigvee_{(a_2,\ldots,a_r)} X_1^{S_1} \cdot X_2^{a_2} \cdots X_r^{a_r}. \quad (5.1)$$

The sum is taken for all possible $\vec{a} = (a_2, a_3, \ldots, a_r)$, where $a_i \in \{0, 1, \ldots, 15\}$. Thus, the total number of products in (5.1) is $16^{r-1} = 2^{n-4}$. Since the products in (5.1) are mutually disjoint, the inclusive OR operation can be replaced with the exclusive OR operation without changing the function. So, the theorem holds for both an SOP and an ESOP. $(Q.E.D.)$

**Theorem 5.2** *Consider a function $f(X_1, X_2, \ldots, X_r)$, where $X_i$ consists of 4 binary variables. Let $f(X_1, X_2, \ldots, X_r)$ be partially symmetric with respect to $X_i$ for $i = 1, 2, \ldots, r$. That is, $f$ is invariant under the permutation of variables in $X_i$. Then, $f$ can be represented by a 16-valued SOP (ESOP) with at most $5^{r-1}$ products.*

(Proof) Since $f$ is symmetric with respect to $X_i$ for $i = 2, \ldots, r$, $f$ can be represented by

$$f(X_1, X_2, \ldots, X_r) = \bigvee_{\vec{b}} g(X_1, \vec{b}) \cdot g_{b_2}(X_2) \cdot g_{b_3}(X_3) \cdots g_{b_r}(X_r). \quad (5.2)$$

The sum is taken for all possible $\vec{b} = (b_2, b_3, \ldots, b_r)$, where $b_i \in \{0, 1, 2, 3, 4\}$. Note that

$$g_j(X_i) = \begin{cases} 1 & (\text{when the number of 1's in } X_i \text{ is } j) \\ 0 & (\text{otherwise}). \end{cases}$$

Thus, the number of products in (5.2) is $5^{r-1}$. Since the products in (5.2) are mutually disjoint, the inclusive OR operation can be replaced with the exclusive OR operation. So, the theorem holds for both an SOP and an ESOP. $(Q.E.D.)$

**Theorem 5.3** *The number of products to implement an n-bit adder where $n = 2r$ is as follows:*

- *2-valued SOP :* $6 \cdot 2^n - 4n - 5$
- *2-valued ESOP :* $2^{n+1} - 1$
- *4-valued SOP:* $n^2 + 1$
- *4-valued ESOP :* $(n^2 + n + 2)/2$
- *16-valued SOP :* $2r^2 - r + 2$
- *16-valued ESOP :* $r^2 + r + 1$

The proof is omitted due to the page limitation.

## 6. Generation of 16-Valued Expressions

The number of products in the multi-valued expressions greatly depends on the method of grouping the variables [6, 9]. An **optimum grouping of the input variables** is one that minimizes the number of products in the expression. To obtain the exact optimum grouping of the input variables, we have to consider all possible groupings. For a function of $n = 4r$ variables, the number of different groupings to consider is

$$\eta(n) = \frac{n!}{(4!)^r \cdot r!}.$$

$\eta(n)$ is 35 when $n = 8$, and is 5775 when $n = 12$, which are the practical upper bounds on the number of variables that we can obtain the optimum solutions by an exhaustive method.

For functions with more variables, the exhaustive method requires too much computation time. The heuristic method shown below obtains good solutions in a short time. The method first pairs 2-valued variables to make 4-valued variables, and then pairs 4-valued variables to make 16-valued variables. Here, we show how to pair the 2-valued variables. The method to pair 4-valued variables is similar.

**Definition 6.1** *Let $I = \{1, 2, \ldots, n\}$ be a set of subscripts for the input variables $X$. Let $\Pi$ be a partition of $I$ (corresponding to the partition of $X$). Let $t(f : \Pi)$ be the number of products in a minimum SOP for $f$, under the partition $\Pi$. Let $F$ be an SOP for the function $f$. Let $q(i, j)$ be the number of different products in the SOP that are obtained from $F$ by deleting everywhere literals of $x_i$ and $x_j$. Let $t(f : \Pi_{ij})$ be the number of products in a minimum SOP for $f$, when $x_i$ and $x_j$ are paired.*

**Example 6.1** *Let $F$ be*

$$F = \bar{x}_1\bar{x}_2x_3x_4 \vee \bar{x}_1x_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1x_2\bar{x}_3\bar{x}_4.$$

*The products that are obtained by deleting the literals of $x_3$ and $x_4$ from $F$ are: $\bar{x}_1\bar{x}_2, \bar{x}_1x_2, x_1\bar{x}_2, x_1\bar{x}_2$ and $x_1x_2$. The number of distinct product terms is 4, so we have $q(3, 4) = 4$. Similarly, we have $q(2, 3) = q(2, 4) = 3$, $q(1, 2) = q(1, 4) = q(1, 3) = 4$. (End of Example)*

**Lemma 6.1** *Let $\Pi_{ij} = \{[1], [2], \ldots, [i, j], \ldots, [n]\}$. Then, $t(f : \Pi_{ij}) \leq q(i, j)$.*

The smaller the value of $t(f : \Pi_{ij})$, the simpler the SOP when the variables $x_i$ and $x_j$ are paired. Thus, $t(f : \Pi_{ij})$ is used as a figure of merit when the variables $x_i$ and $x_j$ are paired. However, to compute the value of $t(f : \Pi_{ij})$ is time consuming. Note that $q(i, j)$ is an upper bound of $t(f : \Pi_{ij})$.

**Definition 6.2** *A* **grouping graph** *$G$ of an $n$-variable function $f(x_1, x_2, \ldots, x_n)$ is the complete graph with weights satisfying the following conditions:*

1. *$G$ has $n$ nodes.*
2. *The weight of the edge $(i, j)$ is $q(i, j)$.*

The following algorithm first obtains a 4-valued SOP by pairing 2-valued variables, and then it obtains a 16-valued SOP by pairing 4-valued variables.

**Algorithm 6.1** *(Grouping 2-valued variables to obtain a 16-valued expression)*

1. *Simplify the 2-valued SOP to obtain $f(x_1, x_2, \ldots, x_n)$.*
2. *Construct a grouping graph $G_1$ for $f$.*
3. *Cover all the nodes of $G_1$ by a set of edges that have no common elements. In this case, find a set such that the sum of the weights is minimum. This is the* **optimum matching** *of the graph $G_1$.*
4. *Partition the 2-valued variables corresponding to the edges, and generate the 4-valued SOP by pairing 2-valued variables.*
5. *Simplify the 4-valued SOP to obtain $F(Y_1, Y_2, \ldots, Y_{\frac{n}{2}})$.*
6. *Construct a grouping graph $G_2$ for $F$.*
7. *Obtain the optimum matching of the graph $G_2$.*
8. *Partition the 4-valued variables corresponding to the edges, and generate the 16-valued SOP by pairing 4-valued variables.*
9. *Simplify the 16-valued SOP to obtain $\hat{F}(Z_1, Z_2, \ldots, Z_{\frac{n}{4}})$.*

Algorithm 6.1 is a heuristic, and it does not always produce the optimal solution.

## 7. Experimental Results

We minimized standard PLA benchmarks [12] as well as adders, symmetric functions, and address generation functions. Table 7.1 shows the results. *IN* denotes the number of inputs; *OU* denotes the number of outputs; *SOP* denotes the number of products in a sum-of-products expression; *ESOP* denotes the number of products in an EXOR sum-of-products expression. *2-valued* denotes the number of products in a 2-valued expression; *4-valued* denotes the number of products in a 4-valued expression; *16-valued* denotes the number of products in a 16-valued expression. Algorithm 6.1 was used to obtain both 4-valued and 16-valued expressions.

Table 7.1 shows that 4-valued expressions require fewer products than 2-valued ones, and 16-valued expressions require fewer products than 4-valued ones. For adders (*adr8*

**Table 7.1. Number of products to represent functions**

| | IN | OU | 2-Valued | | 4-Valued | | 16-Valued | |
|---|---|---|---|---|---|---|---|---|
| | | | SOP | ESOP | SOP | ESOP | SOP | ESOP |
| adr8 | 16 | 17 | 1499 | 511 | 65 | 37 | 30 | 21 |
| adr10 | 20 | 21 | 6099 | 2047 | 101 | 56 | 47 | 31 |
| adr12 | 24 | 25 | 24523 | 8191 | 145 | 79 | 68 | 43 |
| alu4 | 14 | 8 | 577 | 288 | 253 | 124 | 156 | 107 |
| alupla | 25 | 5 | 2144 | 1429 | 1008 | 806 | 429 | 372 |
| apex2 | 39 | 3 | 39 | 60 | 14 | 27 | 14 | 27 |
| chkn | 29 | 7 | 140 | 145 | 106 | 124 | 63 | 78 |
| cordic | 23 | 2 | 914 | 776 | 67 | 104 | 15 | 10 |
| cps | 24 | 109 | 162 | 140 | 146 | 134 | 140 | 151 |
| intb | 15 | 7 | 629 | 261 | 294 | 172 | 200 | 129 |
| misex3 | 14 | 14 | 690 | 507 | 462 | 401 | 192 | 177 |
| mlp6 | 12 | 12 | 1877 | 706 | 1206 | 571 | 570 | 391 |
| rd84 | 8 | 4 | 255 | 56 | 54 | 23 | 12 | 9 |
| rdm16 | 16 | 16 | 404 | 176 | 281 | 140 | 140 | 93 |
| seq | 41 | 35 | 336 | 248 | 216 | 179 | 135 | 121 |
| spla | 16 | 46 | 261 | 264 | 211 | 183 | 105 | 105 |
| sqr12 | 12 | 24 | 2602 | 1219 | 2016 | 1300 | 1111 | 937 |
| sym12 | 12 | 1 | 496 | 245 | 90 | 65 | 15 | 12 |
| sym16 | 16 | 1 | 8009 | 2795 | 784 | 439 | 67 | 62 |
| t481 | 16 | 1 | 481 | 13 | 32 | 8 | 5 | 3 |
| tial | 14 | 6 | 575 | 428 | 280 | 172 | 195 | 137 |
| wgt12 | 12 | 4 | 4095 | 519 | 425 | 146 | 51 | 28 |

- *adr12*), symmetric functions (*sym12, sym16, rd84, wgt12*) and some arithmetic circuits (e.g., *alu4, alupla, cordic, tial, wgt12*), ESOPs require fewer products than SOPs, in many cases. However, for some functions (e.g., *apex2, chkn,*), ESOPs require more products than SOPs. For minimization of ESOPs, we used EXMIN3, an improved version of EXMIN2[7].

## 8. Conclusion and Comments

In this paper, we presented a method to design reconfigurable logic array on an FPGA. We showed that an FPGA with 4-input LUTs directly implements 16-valued expressions.

Experimental results show that 16-valued expressions require fewer products than corresponding 2-valued ones. For some functions ESOPs require fewer products than SOPs, and vice versa. Since we can implement both expressions, we can select smaller ones. We also showed that a 16-valued expression represent symmetric functions and adders quite efficiently.

In this paper, we only presented the design method for 16-valued logic. An extension to $2^k$-valued logic by using $k$-input LUTs is straightforward. Also, we can use LUTs with different number of inputs. LUTs with $k = 3$ to 6 inputs are available in modern FPGAs.

The logic design method is similar to that of PLAs with $k$-bit input decoders [6, 9]. The PLAs with $k$-bit input decoders use $2^k$ literal lines for each group, while the $2^k$-valued reconfigurable logic array use only $k$ horizontal lines. Also, in the PLAs, each input decoder implements all the $2^k$ literals, while in the $2^k$-valued reconfigurable logic array, each LUT implements only one literal.

## 9. Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, and the grant of Kitakyushu Innovative Cluster Project. Discussion with Prof. Jon T. Butler improved English presentation.

## References

[1] S. A. Guccione, D. Levi, and D. Downs, "A reconfigurable content addressable memory," In Jose Rolim et al. editors, *Parallel and Distributed Processing*, pp. 882-889, Springer-Verlag, Berlin, May 2000. *Proceedings of the 15th International Parallel and Distributed Processing Workshops*, IPDPS 2000. *Lecture Notes in Computer Science* 1800.

[2] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. & Develop.* pp. 443-458, Sept. 1974.

[3] P. B. James-Roxby and D.J. Downs, "An efficient content-addressable memory implementation using dynamic routing," *FCCM'01 2001*, pp.81- 90, 2001.

[4] G. Nilsen, J. Torresen, O. Sorasen, "A variable word-width content addressable memory for fast string matching," *Norchip Conference*, 2004

[5] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization", *IEEE Trans. CAD*, Vol. 6(5), pp. 727-750, Sep. 1987.

[6] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. Comput.*, Vol. C-33, No. 10, pp. 879-894, Oct. 1984.

[7] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 5, May 1993, pp. 621-632.

[8] T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

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

[10] T. Sasao, "Design methods for multiple-valued input address generators," *ISMVL-2006* (invited paper), Singapore, May 17-20, 2006.

[11] N. Song and M. A. Perkowski, "Minimization of exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions," *IEEE Trans.Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-15, No. 4, pp. 385-395, April 1996.

[12] S. Yang, "Logic synthesis and optimization benchmark user guide, version 3.0," MCNC, Jan. 1991.

[13] Xilinx,http://www.xilinx.com/