Sum-of-Generalized Products Expressions Applications and Minimization

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

Abstract

This paper defines the sum-of-generalized-products expression (SOGP), a generalization of a sum-of-products expression (SOP), which is suitable for FPGA implementation. 2^k-valued logic is introduced to design circuits with k-input LUTs. Minimization methods for SOGPs are developed. Experimental results show that SOGPs require many fewer products than SOPs, especially for symmetric functions and adders. The method is promising for FPGA design.

1 Introduction

1.1 Previous Approach

Most logic synthesis systems use sum-of-products expressions (SOPs) to represent logic functions. When AND and OR gates are the basic logic elements, SOPs are quite useful.

However, for a field programmable gate array (FPGA), basic logic elements are look-up tables (LUTs). In such a case, logic design using LUTs is desirable.

In the past, several logic design methods using LUTs as basic elements have been developed. One method is based on multi-valued decision diagrams, where each node of a decision diagram is replaced by an LUT that implements multiplexers or their extensions [8, 9, 13]. Another method is based on binary decision diagrams (BDDs), and uses LUTs to implement cascades [10]. It is useful for functions whose BDDs are relatively small.

1.2 New Approach

In this paper, we consider a generalization of multi-valued SOPs that is suitable for FPGA implementations.

An SOP is realized by a two-level AND-OR circuit such as shown in Fig. 1.1. By replacing a set of inverters with multi-valued literal generators, we have a circuit that realizes a product with multi-valued literals shown in Fig. 1.2. By combining products by an OR gate, we have a circuit that

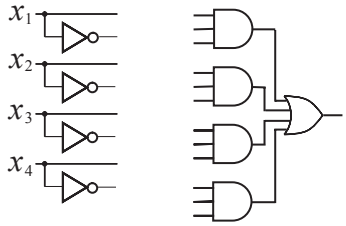

Figure 1.1: AND-OR two-level circuit.

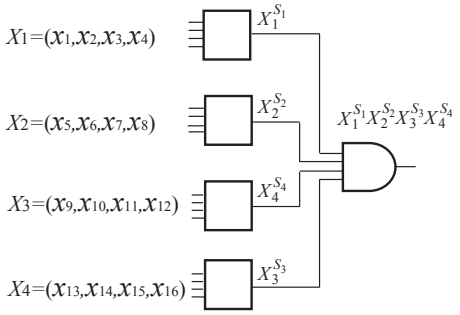

Figure 1.2: Product with multi-valued literals.

realizes a sum-of-products (SOP) with multi-valued literals [11]. In Fig. 1.2, by replacing the AND gate with an LUT, we have a circuit that realizes a generalized product (GP) $g(h_1(X_1), h_2(X_2), h_3(X_3), h_4(X_4))$ as shown in Fig. 1.3. The LUT for h_i is a **literal generator**, while the LUT for g is a **GP generator**. By combining the generalized products with an OR gate, we have a circuit that realizes a **sum-of-generalized-products (SOGP)** as shown in Fig. 1.4.

2 Realization of SOGP on an FPGA

In this paper, we use LUT type FPGAs. Such an FPGA consists of many LUTs, as well as block RAMs (BRAMs). To illustrate the idea, we assume that each FPGA has four inputs. In this case, we introduce 16-valued logic

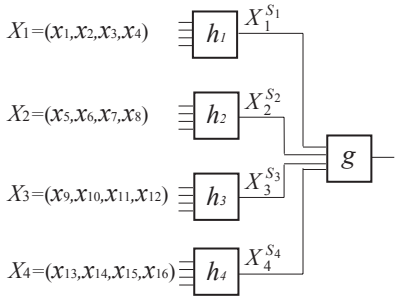

Figure 1.3: Generalized product.

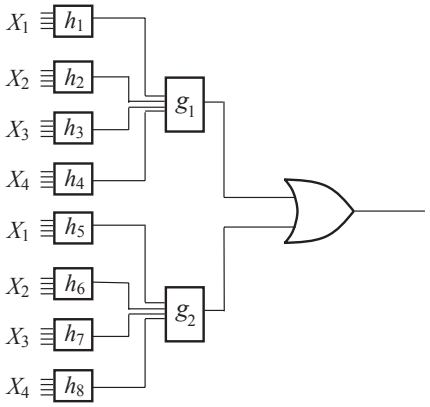

Figure 1.4: Sum-of-generalized-products.

to design 4-input LUT circuits. A 4-input LUT realizes an arbitrary function of four variables. Fig. 2.1 (a) shows a map of a 4-variable function. An arbitrary 4-variable logic function can be represented by a subset of 16 minterms $\{m_0, m_1, \ldots, m_{15}\}$. For example, the function in Fig. 2.1 (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. 2.1 (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

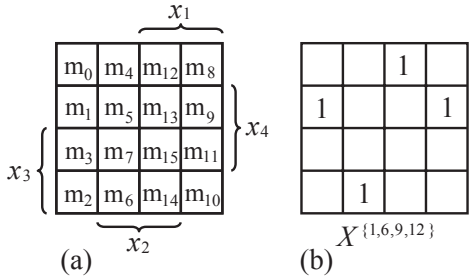

Figure 2.1: Map for 4-variable Function.

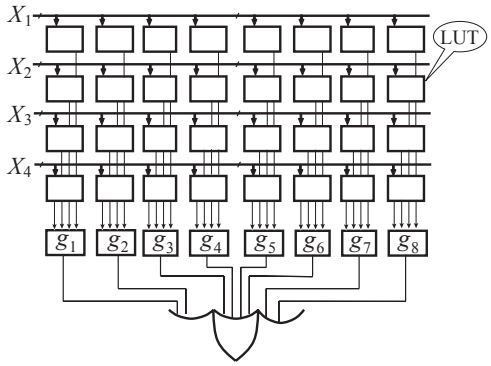

Figure 2.2: Realization of SOGP by LUTs.

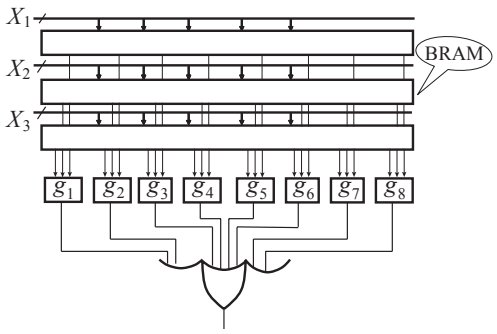

Figure 2.3: Realization of SOGP by BRAMs.

together as $X = (x_1, x_2, x_3, x_4)$, and X is considered as a 16-valued variable. Thus, Fig. 1.2 implements a product of 16-valued literals of the form $X_1^{S_1} X_2^{S_2} X_3^{S_3} X_4^{S_4}$, where $S_i \subseteq P$ and $P = \{0, 1, 2, \ldots, 15\}$.

Fig. 2.2 shows a realization of an SOGP by LUTs. Note that each column realizes a generalized product having a form $g_i(h_1(X_1), h_2(X_2), h_3(X_3), h_4(X_4))$. Since each LUT has four inputs, Fig. 2.2 implements a logic function with 16 binary inputs. Fig. 2.3 shows a realization of an SOGP by BRAMs. Note that FPGAs often have BRAMs that can be configured as memories with 7 to 9 inputs and 8 to 36 outputs. In these figures, the horizontal lines denote bundles of binary lines. Each horizontal bundle denotes a multi-valued variable, and each column realizes a generalized product of multi-valued literals. When the BRAM has k inputs, an SOGP with 2^k-valued literals is realized. When BRAMs (synchronous RAMs) are used as logic elements, a clock pulse is necessary.

We can formulate the optimization problem of an SOGP as follows:

Problem 1 *Given a function f and a partition of the input variables $(X_1, X_2, \ldots, X_r)$, represent $f(X_1, X_2, \ldots, X_r)$ as*

$$\bigvee g_j(h_{1j}(X_1), h_{2j}(X_2), \ldots, h_{rj}(X_r)),$$

using the minimum number of g_k's.

Interesting questions include

1. How to partition the variables?
2. How to minimize the SOGP?
3. How many GPs are necessary to represent a function by using an SOGP?

When FPGAs with 4-input LUTs are used, each variable X_i consists of 4 binary variables.

3 SOP with Multi-Valued Literals

Before showing the design method, we present some definitions. Optimization of SOPs with multi-valued variables is well known [6, 5, 9].

Definition 3.1 *A mapping $f : P^n \rightarrow B$ is a* **p-valued input two-valued output function**, *where $P = \{0,1,\ldots,p-1\}$ and $B = \{0,1\}$. Let X be a variable that takes a value in $P = \{0,1,\ldots,p-1\}$. 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 $X_1{}^{S_1} X_2{}^{S_2} \cdots X_n{}^{S_n}$, the AND of n literals 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* **product term**. *When $|S_i| = 1$ for all i, $(i = 1,2,\ldots,n)$, the product is a* **minterm**. *When $S_i = P$ for all i, the logical product corresponds to the constant 1. When $p = 2$, f is a two-valued logic function. When we consider two-valued logic functions only, we often represent the literal $X^{\{0\}}$ by $\overline{X}$, and $X^{\{1\}}$ by X.*

An arbitrary multi-valued input two-valued output function is represented by an SOP. Many SOPs exist that represent the same function. Among them, one with the minimum number of products is a **minimum SOP**. MINI [2] and ESPRESSO-MV [5] can minimize SOPs with multi-valued inputs.

4 Simplification of SOGPs

In this part, we consider a method to generate a simple SOGP for a given function. The method first generates an SOP with multiple-valued literals. Then, we simplify it using the properties of SOGPs. To show the method, consider the following:

Example 4.1 *Consider an SOP of a function $f : \{0,1,2,3\}^3 \rightarrow \{0,1\}$.*

$f = X_1^{\{0,3\}} X_2^{\{1,2,3\}} X_3^{\{3\}} \vee X_1^{\{1,2\}} X_2^{\{0\}}$.
By rewriting the second product, we have
$f = X_1^{\{0,3\}} X_2^{\{1,2,3\}} X_3^{\{3\}} \vee X_1^{\overline{\{0,3\}}} X_2^{\overline{\{1,2,3\}}}$.
Note that the latter expression requires only one kind of literal generator for each variable. Let
$y_1 = X_1^{\{0,3\}}, y_2 = X_2^{\{1,2,3\}}$ *and* $y_3 = X_3^{\{3\}}$.
Then, the given function can be represented as a single GP:
$f = y_1 y_2 y_3 \vee \overline{y_1}\,\overline{y_2} = g(y_1, y_2, y_3)$. *(End of Example)*

Definition 4.1 *A* **traditional implicant** *of a function f is a product term that implies f. A* **generalized implicant** *is a function of the form $g(h_1(X_1), h_2(X_2), \ldots, h_r(X_r))$ that implies f. A* **non-traditional implicant** *is a generalized implicant that cannot be represented by a single product.*

Note that in Example 4.1, $X_1^{\{0,3\}} X_2^{\{1,2,3\}} X_3^{\{3\}}$ is a traditional implicant, while $g(y_1,y_2,y_3) = X_1^{\{0,3\}} X_2^{\{1,2,3\}} X_3^{\{3\}} \vee X_1^{\overline{\{0,3\}}} X_2^{\overline{\{1,2,3\}}}$ is a non-traditional implicant. Example 4.1 shows that two different prime implicants can be merged into one GP. To find such pair of products, we need the following:

Definition 4.2 *Consider two products of a function: $c_1 = X_1^{S_1} X_2^{S_2} \ldots X_r^{S_r}$ and $c_2 = X_1^{T_1} X_2^{T_2} \ldots X_r^{T_r}$, where $S_i \subseteq P_i$ and $T_i \subseteq P_i$. Then, c_1 and c_2 are* **compatible** *iff for all i, ($S_i = T_i$ or $S_i = P_i$ or $T_i = P_i$ or $S_i = \overline{T_i}$).*

Compatible implicants can be merged to a single generalized implicant.

Example 4.2 *Consider the function $f : \{0,1,2\}^3 \rightarrow \{0,1\}$. In this case, $P_1 = P_2 = P_3 = \{0,1,2\}$. Let*

$$\begin{aligned} f \;=\; & X_1^{\{0\}} X_2^{\{0\}} X_3^{\{0\}} \vee X_1^{\{0\}} X_2^{\{1,2\}} X_3^{\{1,2\}} \\ & \vee X_1^{\{1,2\}} X_2^{\{0\}} X_3^{\{1,2\}} \vee X_1^{\{1,2\}} X_2^{\{1,2\}} X_3^{\{0\}}. \end{aligned}$$

Note that all the products are compatible each other, since f can be written as follows:

$$\begin{aligned} f \;=\; & X_1^{\{0\}} X_2^{\{0\}} X_3^{\{0\}} \vee X_1^{\{0\}} X_2^{\overline{\{0\}}} X_3^{\overline{\{0\}}} \vee X_1^{\overline{\{0\}}} X_2^{\{0\}} X_3^{\overline{\{0\}}} \\ & \vee X_1^{\overline{\{0\}}} X_2^{\overline{\{0\}}} X_3^{\{0\}}. \end{aligned}$$

Thus, f is represented by a single GP. (End of Example)

To represent a GP compactly, we use the notion of a base product.

Definition 4.3 *Let $g(h_1(X_1), h_2(X_2), \ldots, h_r(X_r))$, be a generalized product. Then, a* **base product** *is $c_0 = X_1^{S_1} X_2^{S_2} \ldots X_r^{S_r}$, where $h_i(X_i) = X_i^{S_i}$.*

Example 4.3 *Consider the function in Example 4.2. Let the base product be one where each literal contains the 0 element. Then, $X_1^{\{0\}} X_2^{\{0\}} X_3^{\{0\}}$ is the base product. (End of Example)*

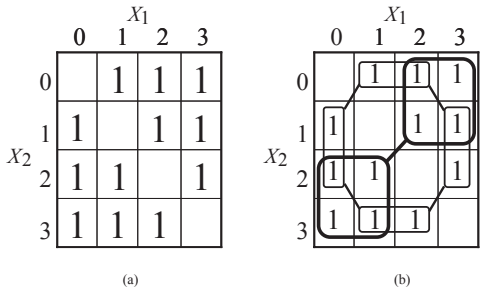

Figure 4.1: Map for 2-variable Function.

Lemma 4.1 *For a given function f and a base product c_0, there exists a unique GP.*

Algorithm 4.1 *(Generation of the GP from a base product).*

1. *Let the base product be $c_0 = X_1^{S_1} X_2^{S_2} \dots X_r^{S_r}$.*
2. *Let $c_1 = X_1^{T_1} X_2^{T_2} \dots X_r^{T_r}$ be a product, where none or some literals of c_0 are complemented. Note that there are 2^r different c_1's to consider.*
3. *The GP is obtained as the sum of all possible products c_1 that does not intersect $\overline{f}$.*

Example 4.4 *Consider the two-variable function $f(X_1, X_2)$ shown in Fig. 4.1(a). Let the base product be $c_0 = X_1^{\{0\}} X_2^{\{0\}}$. Consider the products $c_1 = X_1^{\overline{\{0\}}} X_2^{\{0\}}$, $c_2 = X_1^{\{0\}} X_2^{\overline{\{0\}}}$, and $c_3 = X_1^{\overline{\{0\}}} X_2^{\overline{\{0\}}}$. Since c_1 and c_2 do not intersect $\overline{f}$, the GP generated by c_0 is*
$c_4 = X_1^{\overline{\{0\}}} X_2^{\{0\}} \vee X_1^{\{0\}} X_2^{\overline{\{0\}}} = X_1^{\{0\}} \oplus X_2^{\{0\}}$.
From the base product $c_5 = X_1^{\{1\}} X_2^{\{1\}}$, we have the GP:
$c_6 = X_1^{\overline{\{1\}}} X_2^{\{1\}} \vee X_1^{\{1\}} X_2^{\overline{\{1\}}} = X_1^{\{1\}} \oplus X_2^{\{1\}}$.
From the base product $c_7 = X_1^{\{2\}} X_2^{\{2\}}$, we have the GP:
$c_8 = X_1^{\overline{\{2\}}} X_2^{\{2\}} \vee X_1^{\{2\}} X_2^{\overline{\{2\}}} = X_1^{\{2\}} \oplus X_2^{\{2\}}$.
From the base product $c_9 = X_1^{\{3\}} X_2^{\{3\}}$, we have the GP:
$c_{10} = X_1^{\overline{\{3\}}} X_2^{\{3\}} \vee X_1^{\{3\}} X_2^{\overline{\{3\}}} = X_1^{\{3\}} \oplus X_2^{\{3\}}$.
From the base product $c_{11} = X_1^{\{0,1\}} X_2^{\{0,1\}}$, we have the GP:
$c_{12} = X_1^{\overline{\{0,1\}}} X_2^{\{0,1\}} \vee X_1^{\{0,1\}} X_2^{\overline{\{0,1\}}} = X_1^{\{0,1\}} \oplus X_2^{\{0,1\}}$.
From the base product $c_{13} = X_1^{\{0,2\}} X_2^{\{0,2\}}$, we have the GP:
$c_{14} = X_1^{\overline{\{0,2\}}} X_2^{\{0,2\}} \vee X_1^{\{0,2\}} X_2^{\overline{\{0,2\}}} = X_1^{\{0,2\}} \oplus X_2^{\{0,2\}}$.
From the base product $c_{15} = X_1^{\{0,3\}} X_2^{\{0,3\}}$, we have the GP:
$c_{16} = X_1^{\overline{\{0,3\}}} X_2^{\{0,3\}} \vee X_1^{\{0,3\}} X_2^{\overline{\{0,3\}}} = X_1^{\{0,3\}} \oplus X_2^{\{0,3\}}$.
In this way, we have 7 GPs for f. (End of Example)

A set of compatible products can be uniquely represented by a base product and f. In an SOP of f, a product of f corresponds to an implicant of f, while in an SOGP, a base product $X_1^{S_1} X_2^{S_2} \dots X_r^{S_r}$ may not be an implicant of f. It shows that $X_1^{T_1} X_2^{T_2} \dots X_r^{T_r}$ is an implicant of f, where $T_i = S_i$ or $\overline{S_i}$.

From the set of prime implicants of a multiple-valued SOP, we can generate a GP cover of the function. However, to obtain a minimum SOGP, we need additional GPs. The following example illustrates this:

Example 4.5 *Consider the 3-valued input 4-variable function:*

$$\begin{aligned} f \;=\; & X_4^{\{0,2\}} X_3^{\{1,2\}} X_2^{\{1,2\}} X_1^{\{2\}} \vee X_4^{\{1\}} X_3^{\{0,1\}} X_2^{\{0\}} \\ & \vee X_4^{\{0,2\}} X_3^{\{1,2\}} X_2^{\{2\}} \vee X_4^{\{1\}} X_3^{\{0\}} \\ & \vee X_4^{\{0,2\}} X_3^{\{2\}} \vee X_4^{\{1\}} X_3^{\{0,1\}} X_2^{\{0,1\}} X_1^{\{0,1\}}. \end{aligned}$$

This is the minimum SOP consisting of the essential prime implicants [9] only. The following SOP consisting of prime and non-prime implicants also denotes the same function:

$$\begin{aligned} f \;=\; & X_4^{\{0,2\}} X_3^{\{1\}} X_2^{\{1,2\}} X_1^{\{2\}} \vee X_4^{\{1\}} X_3^{\{1\}} X_2^{\{0\}} \\ & \vee X_4^{\{0,2\}} X_3^{\{1,2\}} X_2^{\{2\}} \vee X_4^{\{1\}} X_3^{\{0\}} \\ & \vee X_4^{\{0,2\}} X_3^{\{2\}} \vee X_4^{\{1\}} X_3^{\{0,1\}} X_2^{\{0,1\}} X_1^{\{0,1\}}. \end{aligned}$$

Note that the 1st and 2nd products are compatible. Similarly, the 3rd and the 4th products are compatible. Also, the 5rd and 6th products are compatible. From this, we have the following SOGP consisting of only three GPs:

$$\begin{aligned} f \;=\; & (X_4^{\overline{\{1\}}} X_3^{\{1\}} X_2^{\overline{\{0\}}} X_1^{\{2\}} \vee X_4^{\{1\}} X_3^{\{1\}} X_2^{\{0\}}) \\ & \vee (X_4^{\overline{\{1\}}} X_3^{\overline{\{0\}}} X_2^{\{2\}} \vee X_4^{\{1\}} X_3^{\{0\}}) \\ & \vee (X_4^{\overline{\{1\}}} X_3^{\{2\}} \vee X_4^{\{1\}} X_3^{\overline{\{2\}}} X_2^{\overline{\{1\}}} X_1^{\overline{\{2\}}}). \end{aligned}$$

In the first SOP, the 1st and the 2nd products are not compatible. So, they cannot be merged. However, in the second SOP, they are compatible and can be merged into one. Thus, to derive a minimum SOGP, we have to consider non-prime implicants, as well as prime implicants. (End of Example)

The above example shows that the minimization of SOGPs is more complicated than that of SOPs.

Definition 4.4 *A* **prime generalized implicant** *(PGI) of a function f is a generalized implicant of f that cannot be covered by other generalized implicant of f.*

As for the number of PGIs, we have the following:

Theorem 4.1 *Consider a function $f(X_1, X_2, \dots, X_r)$ of n variables, where $n = k \cdot r$ and each X_i consists of k variables. Suppose that each literal generator has k binary inputs. Then, the number of distinct PGIs of the function f is at most $2^{r(2^k-1)}$.*

Corollary 4.1 *Consider the cases of $k = 1, 2, 3,$ and 4. The numbers of PGIs of an n-variable function is at most*

1. $2^r = 2^n$, *when* $k = 1$.
2. $(2^3)^r = \sqrt{8}^n = (2.828\ldots)^n$, *when* $k = 2$.
3. $(2^7)^r = 128^{\frac{n}{3}} = (5.03\ldots)^n$, *when* $k = 3$.
4. $(2^{15})^r = 32768^{\frac{n}{4}} = (13.45\ldots)^n$, *when* $k = 4$.

In the case of $k = 1$, only one base product $\bar{x_1}\bar{x_2}\cdots\bar{x_n}$ is necessary to represent an n-variable function. Note that the number of variables for g in Fig. 1.3 is n. This corresponds to a single-memory realization of the logic function, and is not interesting.
As for the number of products in an SOGP, we have the following:

Theorem 4.2 *An arbitrary function of $n = kr$ variables can be represented by a 2^k-valued SOP with at most 2^{n-k} products.*

If we can find the set of all the PGIs, then we can find an exact minimum SOGP, in a similar way to the case of SOPs. Unfortunately, too many different PGIs exist for a function. So, generating all the PGIs is not practical in many cases. With this observation, we have developed two types of SOGP minimization algorithms. Algorithm 4.2 is similar to the Quine-McCluskey method, and obtains good solutions, but requires a large amount of memory.

Algorithm 4.2 *(GQM: Generation of a good SOGP).*

1. *Obtain the set of all the prime implicants of a multiple-valued SOP.*
2. *Merge the compatible products.*
3. *Find the set of base products.*
4. *For each base product, generate the GP by Algorithm 4.1.*
5. *Append additional GPs (Optional).*
6. *Find a minimum cover of the minterms using GPs.*

Example 4.6 *Consider the two-variable function with 4-valued input shown in Fig. 4.1(a).*
Represent it as an SOP with 4-valued inputs.
There exist 14 prime implicants:
$X_1^{\{0\}}X_2^{\{1,2,3\}}$, $X_1^{\{1,2,3\}}X_2^{\{0\}}$; $X_1^{\{1\}}X_2^{\{0,2,3\}}$, $X_1^{\{0,2,3\}}X_2^{\{1\}}$; $X_1^{\{2\}}X_2^{\{0,1,3\}}$, $X_1^{\{0,1,3\}}X_2^{\{2\}}$; $X_1^{\{3\}}X_2^{\{0,1,2\}}$, $X_1^{\{0,1,2\}}X_2^{\{3\}}$; $X_1^{\{0,1\}}X_2^{\{2,3\}}$, $X_1^{\{2,3\}}X_2^{\{0,1\}}$; $X_1^{\{0,2\}}X_2^{\{1,3\}}$, $X_1^{\{1,3\}}X_2^{\{0,2\}}$; $X_1^{\{0,3\}}X_2^{\{1,2\}}$, $X_1^{\{1,2\}}X_2^{\{0,3\}}$.
A minimum SOP has consists of four products: $f(X_1,X_2) = X_1^{\{0\}}X_2^{\{1,2,3\}} \vee X_1^{\{1\}}X_2^{\{0,2,3\}} \vee X_1^{\{2\}}X_2^{\{0,1,3\}} \vee X_1^{\{3\}}X_2^{\{0,1,2\}}$.

Represent it as an SOGP with 4-valued inputs.
Note that each pair of prime implicants separated by semicolons are compatible. So they can be combined into 7 PGIs:
$X_1^{\{0\}}X_2^{\{1,2,3\}} \vee X_1^{\{1,2,3\}}X_2^{\{0\}} = X_1^{\{0\}} \oplus X_2^{\{0\}}$,
$X_1^{\{1\}}X_2^{\{0,2,3\}} \vee X_1^{\{0,2,3\}}X_2^{\{1\}} = X_1^{\{1\}} \oplus X_2^{\{1\}}$,
$X_1^{\{2\}}X_2^{\{0,1,3\}} \vee X_1^{\{0,1,3\}}X_2^{\{2\}} = X_1^{\{2\}} \oplus X_2^{\{2\}}$,
$X_1^{\{3\}}X_2^{\{0,1,2\}} \vee X_1^{\{0,1,2\}}X_2^{\{3\}} = X_1^{\{3\}} \oplus X_2^{\{3\}}$,
$X_1^{\{0,1\}}X_2^{\{2,3\}} \vee X_1^{\{2,3\}}X_2^{\{0,1\}} = X_1^{\{0,1\}} \oplus X_2^{\{0,1\}}$,
$X_1^{\{0,2\}}X_2^{\{1,3\}} \vee X_1^{\{1,3\}}X_2^{\{0,2\}} = X_1^{\{0,2\}} \oplus X_2^{\{0,2\}}$,
$X_1^{\{0,3\}}X_2^{\{1,2\}} \vee X_1^{\{1,2\}}X_2^{\{0,3\}} = X_1^{\{0,3\}} \oplus X_2^{\{0,3\}}$.
In this case, the base products of them are: $X_1^{\{0\}}X_2^{\{0\}}$, $X_1^{\{1\}}X_2^{\{1\}}$, $X_1^{\{2\}}X_2^{\{2\}}$, $X_1^{\{3\}}X_2^{\{3\}}$, $X_1^{\{0,1\}}X_2^{\{0,1\}}$, $X_1^{\{0,2\}}X_2^{\{0,2\}}$, *and* $X_1^{\{0,3\}}X_2^{\{0,3\}}$.
The corresponding GPs are shown above. A minimum SOGP consists of two products
$f(X_1,X_2) = (X_1^{\{0,1\}} \oplus X_2^{\{0,1\}}) \vee (X_1^{\{0,3\}} \oplus X_2^{\{0,3\}})$. *Fig. 4.1(b) shows the map of the SOGP. (End of Example)*

Algorithm 4.3 is a heuristic one and requires less memory than Algorithm 4.2: It derives an SOGP quickly.

Algorithm 4.3 *(GMINI: A fast generation of an SOGP).*

1. $g \leftarrow f$, $sol \leftarrow \phi$.
2. *Obtain a near minimum multiple-valued SOP for g.*
3. *For each product c_i in the SOP, obtain the generalized implicant $GI(c_i)$ by Algorithm 4.1.*
4. *Let $vol(GI(c_i))$ be the number of minterms of g that is covered by $GI(c_i)$. Find the $GI(c_i)$ whose $vol(GI(c_i))$ is the maximum.*
5. $sol \leftarrow sol \cup c_i$, $g \leftarrow g \cdot \overline{GI(c_i)}$, $DC \leftarrow DC \cup GI(c_i)$, *and go to Step 2.*

5 SOGPs for Symmetric Functions and Adders

5.1 Symmetric Functions

Regarding symmetric functions, we have the following:

Theorem 5.1 *Consider a function $f(X_1, X_2, \ldots, X_r)$, where X_i consists of k 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 an SOP with 2^k-valued variables with at most $(k+1)^{r-1}$ products.*

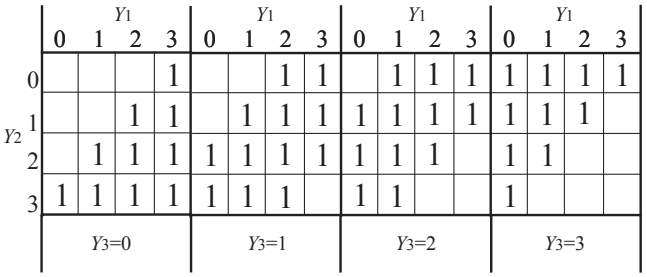

Figure 5.1: Map for 3-variable function.

Example 5.1
Sym9 is the symmetric function $S^9_{\{3,4,5,6\}}(X_1,X_2,X_3)$*, where* $X_1=(x_1,x_2,x_3), X_2=(x_4,x_5,x_6)$ *and* $X_3=(x_7,x_8,x_9)$ *[7]. The function is 1 iff the weight of the input vector is 3,4,5, or 6. From Theorem 5.1, Sym9 can be represented by an SOGP with at most* $(3+1)^{3-1}=4^2=16$ *products, since* $k=r=3$*. In fact, the 8-valued SOGP requires only 6 GPs:*

$$\begin{aligned} f \;=\; & X_1^{\{1,2,3,4,5,6\}}(X_2^{\{0,1,2,4\}} \oplus X_3^{\{0,1,2,4\}}) \\ & \vee X_2^{\{1,2,3,4,5,6\}}(X_1^{\{0,1,2,4\}} \oplus X_3^{\{0,1,2,4\}}) \\ & \vee X_3^{\{1,2,3,4,5,6\}}(X_1^{\{0,1,2,4\}} \oplus X_2^{\{0,1,2,4\}}) \\ & \vee X_1^{\{1,2,3,4,5,6\}} X_2^{\{1,2,3,4,5,6\}} X_3^{\{1,2,3,4,5,6\}} \\ & \vee X_2^{\{0\}}(X_1^{\{7\}} \vee X_3^{\{7\}}) \vee X_2^{\{7\}}(X_1^{\{0\}} \vee X_3^{\{0\}}). \end{aligned}$$

In the above expression, the weight of $X^{\{1,2,3,4,5,6\}}$ *is either 1 or 2; the weight of* $X^{\{0,1,2,4\}}$ *is either 0 or 1; the weight of* $X^{\{0\}}$ *is 0; and the weight of* $X^{\{7\}}$ *is 3. The first three GPs in the above SOGP denote the combinations that the sums of the weights for* X_1*,* X_2*, and* X_3 *are (1 or 2) +(0 or 1)+ (2 or 3). The fourth GP denotes the combination that the sum of the weights is (1 or 2) +(1 or 2)+ (1 or 2). The last two GPs denote the combinations that the sums of the weighs are (0) +(3)+ (0 or 1 or 2). Note that a two-valued SOP for Sym9 requires 84 products. (End of Example)*

Example 5.2 *Consider the function* $g(Y_1,Y_2,Y_3)$ *shown in Fig. 5.1. In this function, the value of* Y_i *denotes the number of one's of* X_i *in Sym9 in the previous example. So, the number of PGIs for g is the same as the number of PGIs for Sym9. The minimum SOGP is*
$f(Y_1,Y_2,Y_3) = Y_1^{\{1,2\}}(Y_2^{\{0,1\}} \oplus Y_3^{\{0,1\}}) \vee$
$Y_2^{\{1,2\}}(Y_1^{\{0,1\}} \oplus Y_3^{\{0,1\}}) \vee Y_3^{\{1,2\}}(Y_1^{\{0,1\}} \oplus Y_2^{\{0,1\}}) \vee$
$Y_1^{\{1,2\}}Y_2^{\{1,2\}}Y_3^{\{1,2\}} \vee Y_2^{\{0\}}(Y_1^{\{3\}} \vee Y_3^{\{3\}}) \vee Y_2^{\{3\}}(Y_1^{\{0\}} \vee Y_3^{\{0\}})$.
This also illustrates that Sym 9 can be represented by only 6 PGIs. (End of Example)

5.2 Adders

Regarding adders, we have the following:

Lemma 5.1 *The i-th bit of an n-bit adder,* s_i*, can be represented by a 4-valued input SOGP with i products, where the least significant output bit is* s_0*.*

From the above Lemma, we have the following:

Theorem 5.2 *An n-bit adder can be represented by a 4-valued input SOGP with* $\frac{n^2+n+2}{2}$ *products.*

Note that the numbers of products to implement *n*-bit adders where $n=2r$ are as follows [9, 11]:

- 2-valued SOP : $6\cdot 2^n - 4n - 5$
- 4-valued SOP: n^2+1
- 16-valued SOP : $2r^2 - r + 2$

6 Experimental Results

We have developed two types of minimization programs GQM and GMINI, and minimized SOGPs for standard PLA benchmarks [12] as well as adders, randomly generated functions and symmetric functions.

Benchmark Functions Table 6.1 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; *SOGP* denotes the number of products in a sum-of-generalized 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. *P* denotes the program used to obtain the result: Q denotes GQM, while M denotes GMINI. Algorithm 6.1 in [11] was used to generate both 4-valued and 16-valued expressions.
In general, 4-valued expressions require fewer products than 2-valued ones, and 16-valued expressions require fewer products than 4-valued ones. Note that the SOGP for the benchmark function *t481* require only one product, while the minimum 2-valued SOP requires 481 products. Fig. 6.1 shows the gate-level realization of *t481* [4].

Randomly Generated Functions We generated random functions of *n* variables, where the number of true minterms is 2^{n-1}. For example, 8_128 is an 8 input random function that has $2^{8-1}=128$ true minterms. In this case, 4-valued SOGPs require fewer products than 4-valued SOPs. However, for 16-valued cases, SOPs and SOGP require the same number of products.

Symmetric Functions Table 6.2 shows the number of products to represent symmetric functions. These functions are defined in [9]. In this case, three bits are grouped to

Table 6.1: Number of products to represent benchmark functions

	IN	OU	2-valued	4-valued		16-valued		P
			SOP	SOP	SOGP	SOP	SOGP	
adr4	8	5	75	17	11	8	7	Q
adr6	12	7	355	37	22	17	14	Q
adr8	16	9	1499	65	37	30	25	M
alu4	14	8	577	253	148	156	135	Q
alupla	25	5	2144	1008	561	429	381	M
apex2	39	3	39	14	12	14	14	M
apex4	9	19	427	412	412	383	382	M
chkn	29	7	140	106	92	63	59	M
cordic	23	2	914	67	20	15	10	M
ex1010	10	10	356	351	350	345	344	M
intb	15	7	629	294	185	200	176	M
misex3	14	14	658	424	361	187	184	Q
mlp4	8	8	119	83	68	47	47	Q
mlp5	10	10	486	325	234	184	163	Q
mlp6	12	12	1877	1192	833	553	518	Q
rdm16	16	16	404	281	192	140	129	M
t481	16	1	481	32	1	5	1	Q
tial	14	6	575	280	209	195	169	Q
8_128	8	1	46	32	28	14	14	Q
12_2048	12	1	572	429	373	214	214	Q

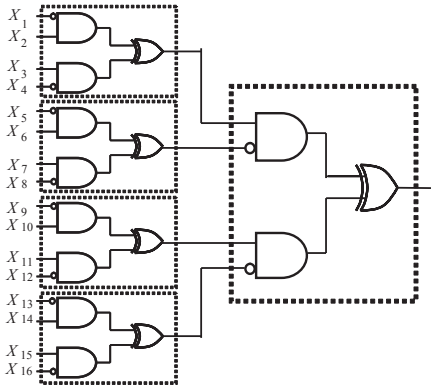

Figure 6.1: Gate-level realization of t481.

derive 8-valued variables. In the case of symmetric functions, 8-valued SOGPs required many fewer products than 8-valued SOPs.

Comparison with Other FPGA Synthesis System

To show the potential of the approach, we compared the number of LUTs and depth with IMap[3]. Table 6.3 compares the number of 4-input LUTs for IMap and SOGP-base synthesis tool. Note that IMap shows a result for a particular function, while the SOGP shows a result for a programmable circuits that implements a given function (*i.e.*,

Table 6.2: Number of products to represent symmetric functions

	In	Out	2-valued	8-valued		P
			SOP	SOP	SOGP	
sym6	6	1	15	3	2	Q
sym9	9	1	84	10	6	Q
sym12	12	1	495	31	10	Q
sym15	15	1	3003	101	30	Q
sym18	18	1	18673	346	145	M
wgt9	9	4	511	35	27	Q
wgt12	12	4	4095	135	104	Q
wgt15	15	4	32767	530	391	Q

Table 6.3: Number of LUTs and depth to implement functions

	In	Out	IMap		SOGP	
			LUT	depth	LUT	depth
alu4	14	8	1035	8	1019	6
ex1010	10	10	2524	9	2456	7
misex3	14	14	1086	8	1732	6

redundant inputs of the OR gates are also implemented). Thus, it may not be a fair comparison. For the mapping of SOGP, we used Xilinx Spartan III(SC3S4000F), and a Xilinx tool.

7 Conclusion and Comments

In this paper, we defined an SOGP, a generalization of an SOP. An SOGP can be efficiently realized by a network of LUTs. We also showed methods to simplify SOGPs. Experimental results show that SOGPs require many fewer products than corresponding SOPs, especially for symmetric functions and adders. Note that an SOGP corresponds to a three-level network, i.e., literal generators, GP generators, and the OR gate. In this case, the literal generators and GP generators are implemented by LUTs. A more general problem is to realize a logic function by a three-level network of LUTs.

Scalability

Let $n = k \cdot r$ be the number of the inputs, k be the number of inputs for LUTs for literal generators, and r be the number of inputs for GP generators. When the maximal number of inputs for an LUT in an FPGA is 6, the functions with at most 36 inputs are directly implemented by an SOGP.

For the functions with more inputs, GP generators can be realized by a trees of LUTs instead of the r-input LUTs. In this case, we have to modify the algorithm. Even in such a case, the number of products does not exceeds that of the SOP, since the architecture implements an SOP as a special

case. A function whose SOP has a reasonable size can be implemented by our method.

Applications

Since the circuit has a regular structure, an SOGP is suitable for a programmable device. Since the interconnections are fixed, dynamic reconfiguration is easy. For functions with many inputs, LUTs with more inputs (5 or 6) or BRAMs with more inputs can be used [11]. We are currently improving an algorithm to group the input variables, as well as a mapping system for FPGAs.

Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS. Discussion with Prof. Jon T. Butler improved English presentation. Mr. Hiroki Nakahara worked for FPGA implementation.

References

[1] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Boston, MA. Kluwer, 1984.

[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] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristic for area minimization in LUT-based FPGA technology mapping," *International Workshop on Logic and Synthesis (IWLS04)*, Temecula, CA, June 2-4, 2004. pp. 14-21.

[4] A. Mishchenko and T. Sasao, "Large-scale SOP minimization using decomposition and functional properties," *40th Design Automation Conference*, Ahaneim, CA, USA, June 2-6, 2003, pp. 149-154.

[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 (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[8] T. Sasao and J. T. Butler, "A design method for look-up table type FPGA by pseudo-Kronecker expansion," *ISMVL-1994*, Boston, MA, U.S.A., May 1994, pp. 97-106.

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

[10] 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.

[11] T. Sasao, "An application of 16-valued logic to design of reconfigurable logic arrays," *ISMVL-2007*, Oslo, Norway, May 13-16, 2007, (to be published).

[12] S. Yang, *Logic Synthesis and Optimization Benchmark User Guide, Version 3.0*, MCNC, Jan. 1991.

[13] S. S. Yau and C. K. Tang, "Universal logic modules and their applications," *IEEE Trans. Computers*, Vol. C-10, pp. 141-149, 1970.