GRMIN: A Heuristic Simplification Algorithm for Generalized Reed-Muller Expressions

Debatosh Debnath and Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820, Japan

Abstract— A generalized Reed-Muller expression (GRM) is a type of AND-EXOR expressions. In a GRM, each variable may appear both complemented and uncomplemented. Networks realized using GRMs are easily tested. This paper presents GRMIN, a heuristic simplification algorithm for GRMs of multiple-output functions. GRMIN uses eight rules. As the primary objective, it reduces the number of products, and as the secondary objective, it reduces the number of literals. Experimental results show that, in most cases, GRMs require fewer products than conventional sum-of-products expressions (SOPs). GRMIN outperforms existing algorithms.

I. Introduction

Logic networks are usually designed by using AND and OR gates. However, networks using exclusive-OR (EXOR) gates have some advantages over conventional AND-OR networks. First, such networks often require fewer gates and interconnections than those designed using AND and OR gates [13]. Examples of such networks include, arithmetic, telecommunication, and error correcting circuits. Second, they can be made easily testable.

Various classes exist in AND-EXOR expressions [6, 12]. Among them, positive polarity Reed-Muller expressions (PPRMs) are well known: a PPRM, an EXOR sum-of-products with positive literals, uniquely represents an arbitrary logic function. Networks based on PPRMs are easily testable [10], but they often require more products than ones based on other expressions. Generalized Reed-Muller expressions (GRMs) [4] are generalization of PPRMs. They never require more products than PPRMs, and often require many fewer products. GRMs were studied many years ago [2], but no practical applications have been reported. Recently, easily testable realizations for GRMs have been developed [14]. Because GRMs often require many fewer products than PPRMs and have very good testability, the GRM based design has practical importance. An exact minimization algorithm for GRMs is available [15], but for the functions with more than six variables it is very time and memory consuming. For heuristic simplification of GRMs, only a few algorithms [3, 9] are available. In this paper, we present GRMIN, a heuristic simplification algorithm for GRMs of multiple-output function. Experimental results show that GRMIN outperforms existing algorithms.

II. Definitions and Basic Properties

2.1. Positional Cube Notation

A positional cube [7] is convenient for manipulation of logic expressions by computers. In positional cube notation, an uncomplemented literal such as x_i is represented by 01, a complemented literal $\bar{x}_i$ is represented by 10, and a don't care (missing variable in a product) is represented by 11. 00 represents no value of the variable, and any cube containing a 00 for any variable position denotes a null cube.

For m-output functions $(f_0, f_1, \ldots, f_{m-1})$ $(m \geq 1)$, an m-bit tag field is concatenated to the cube to denote the output. If the ith bit of the tag field of a cube is 1, the ith output is occupied by the cube. In this case, we treat the outputs as an m-valued variable.

In the positional cube notation, each variable constitutes a **part**. The m-bit tag constitutes the **output part**. Thus, in the representation of an n-variable function there are $n + 1$ parts.

Definition 2.1 *A list of cubes is called the* **OR array** *if it represents the OR of cubes, and called the* **EXOR array** *if it represents the EXOR of cubes.*

Example 2.1 *The three-variable three-output function* $f_0 = x_1\bar{x}_3 \oplus x_2x_3$, $f_1 = x_2x_3$, *and* $f_2 = x_1\bar{x}_3$ *is represented by the four part EXOR array:*

x_1	x_2	x_3	$f_0f_1f_2$
01	11	10	101
11	01	01	110.

(End of Example)

From now on, unless otherwise specified, a *list* of cube represents an EXOR array.

Definition 2.2 *The* **c-distance** *between two cubes is the number of parts in which they differ.*

Example 2.2 *The c-distance between the two cubes in Example 2.1 is four.* *(End of Example)*

2.2. Logical Expression

An alternate representation of a logic function is a logical expression. In this representation, position of 1's in a part of the positional cube notation of a cube are shown as the exponent of the corresponding variable. For example, $x^{\{0\}}$, $x^{\{1\}}$, and $x^{\{0,1\}}$ is used to represent 10, 01, and 11, respectively. In Section 3.3, this representation is used to show the simplification rules for GRMs. From now on, we will consider the binary outputs as a single multiple-valued output and represent by z. All the inputs are two-valued.

Example 2.3 *The three-variable multiple-output function shown in Example 2.1 can be represented by* $x_1^{\{1\}}x_2^{\{01\}}x_3^{\{0\}}z^{\{02\}} \oplus x_1^{\{01\}}x_2^{\{1\}}x_3^{\{1\}}z^{\{01\}}$. *Because it is an EXOR array, the* $\oplus$ *operator is used. (End of Example)*

2.3. PPRM, FPRM and GRM

In this part, we will define three classes of AND-EXOR expressions. The following lemma is the basis of the EXOR-based expansion [12]:

Lemma 2.1 *An arbitrary logic function* $f(x_1, x_2, \ldots, x_n)$ *can be expanded as*

$$f = \bar{x}_1 f_0 \oplus x_1 f_1, \tag{2.1}$$

$$f = f_0 \oplus x_1 f_2, \tag{2.2}$$

$$f = f_1 \oplus \bar{x}_1 f_2, \tag{2.3}$$

where $f_0 = f(0, x_2, \ldots, x_n)$, $f_1 = f(1, x_2, \ldots, x_n)$, *and* $f_2 = f_0 \oplus f_1$.

(2.1), (2.2), and (2.3) are called the **Shannon expansion**, the **positive Davio expansion**, and the **negative Davio expansion**, respectively. If we use (2.2) recursively to a function f, then we have the following:

Lemma 2.2 *An arbitrary n-variable function* $f(x_1, x_2, \ldots, x_n)$ *can be represented as*

$$\begin{aligned} f = {} & a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n \\ & \oplus a_{12} x_1 x_2 \oplus a_{13} x_1 x_3 \oplus \cdots \oplus a_{n-1\,n} x_{n-1} x_n \oplus \\ & \cdots\cdots\cdots\cdots\cdots\cdots \\ & \oplus a_{12\cdots n} x_1 x_2 x_3 \cdots x_n, \end{aligned} \tag{2.4}$$

where a's are either 0 or 1.

(2.4) is called a **positive polarity Reed-Muller expression** (PPRM). For a given function f, the coefficients $a_0, a_1, a_2, \ldots, a_{12\cdots n}$ are uniquely determined. Thus, the PPRM is a canonical representation. The number of products in (2.4) is at most 2^n, and all the literals are positive (uncomplemented).

In (2.4), for each variable x_i $(i = 1, 2, \ldots, n)$, if we use either a positive literal (x_i) throughout or a negative literal $(\bar{x}_i)$ throughout, then we have a **fixed polarity Reed-Muller expression** (FPRM). For each variable x_i, there are two ways of choosing the polarities: positive (x_i) or negative $(\bar{x}_i)$. Thus, 2^n different set of polarities exist for an n-variable function. For a given function and a given set of polarities, a unique set of coefficients $(a_0, a_1, \ldots, a_{12\cdots n})$ exists. Thus, an FPRM is a canonical representation.

In (2.4), if we can freely choose the polarity for each literal, then we have a **generalized Reed-Muller expression** (GRM). Unlike FPRMs, both x_i and $\bar{x}_i$ can appear in a GRM. Some authors use GRMs to represent another class of expressions [6], thus the terminology is not unified. There are $n2^{n-1}$ literals in (2.4), so $2^{n2^{n-1}}$ different set of polarities exist for an n-variable function. For a given set of polarities, a unique set of coefficients $(a_0, a_1, \ldots, a_{12\cdots n})$ exists. Thus, a GRM is a canonical representation for a logic function.

A GRM for a multiple-output function is defined as follows:

Definition 2.3 *An array represents a* **multiple-output GRM**, *if for each output, the corresponding cubes represent a GRM.*

Example 2.4 *Consider an array representing a three-variable two-output function:*

x_1	x_2	x_3	$f_0 f_1$
10	10	10	10
10	01	01	01
01	01	11	11.

Both $f_0 = \bar{x}_1\bar{x}_2\bar{x}_3 \oplus x_1x_2$, *and* $f_1 = \bar{x}_1x_2x_3 \oplus x_1x_2$ *are GRMs. Thus, the array represents a multiple-output GRM. (End of Example)*

2.4. PSDRM and ESOP

Before studying the simplification method for GRMs, it is convenient to define other classes of expressions.

Suppose that we are given a three-variable function $f(x_1, x_2, x_3)$. When we expand f using the positive Davio expansion with respect to x_1, we have

$$f = f_0 \oplus x_1 f_2.$$

Next, when we expand f_0 and f_2 in a similar way with respect to x_2, we have

$$f_0 = f_{00} \oplus x_2 f_{02}, \;\; f_2 = f_{20} \oplus x_2 f_{22}.$$

Furthermore, when we use similar expansions with respect to x_3, we have

$$f_{00} = f_{000} \oplus x_3 f_{002}, \;\; f_{02} = f_{020} \oplus x_3 f_{022},$$
$$f_{20} = f_{200} \oplus x_3 f_{202}, \;\; f_{22} = f_{220} \oplus x_3 f_{222}.$$

The expansion tree in Fig. 2.1 illustrates this process. A path from the root node to a terminal node represents a product of an expression, where a label of an edge shows the literal for the corresponding variable. For example, the path from the root node to f_{000} represents the product $1 \cdot 1 \cdot 1 \cdot f_{000} = f_{000}$, and the path to f_{222} represents $x_1x_2x_3f_{222}$. Thus, the tree in Fig. 2.1 shows the PPRM:

$$\begin{aligned} f = {} & f_{000} \oplus x_3 f_{002} \oplus x_2 f_{020} \oplus x_2x_3 f_{022} \oplus x_1 f_{200} \\ & \oplus x_1x_3 f_{202} \oplus x_1x_2 f_{220} \oplus x_1x_2x_3 f_{222}. \end{aligned}$$

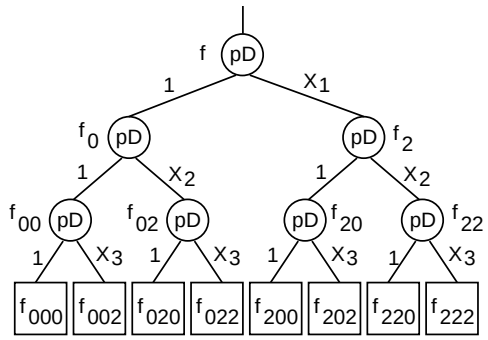

Fig. 2.1. Representation of a logic function using positive Davio expansions.

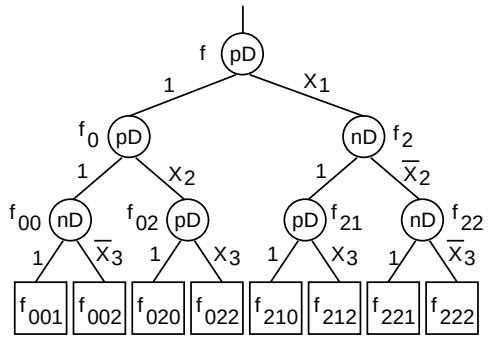

Fig. 2.2. Representation of a logic function using pseudo Reed-Muller expansions.

Each node has a label pD, which shows the positive Davio expansion. In Fig. 2.1, only the positive Davio expansions are used. However, if we use either the positive or the negative Davio expansion for each variable, then we have a more general tree. Such a tree represents an FPRM. If we use either the positive or the negative Davio expansion for each node, then we have a more general tree. Such a tree represents a **pseudo Reed-Muller expression** (PSDRM). For example, in Fig. 2.2, f, f_0, f_{02}, and f_{21} use the positive Davio expansions, while f_2, f_{00}, and f_{22} use the negative Davio expansions. Nodes with label nD denotes the negative Davio expansion. Note that the tree in Fig. 2.2 shows the PSDRM:

$$f = 1 \cdot 1 \cdot 1 \cdot f_{001} \oplus 1 \cdot 1 \cdot \bar{x}_3 f_{002} \oplus 1 \cdot x_2 \cdot 1 \cdot f_{020} \oplus 1 \cdot x_2 x_3 f_{022} \oplus x_1 \cdot 1 \cdot 1 \cdot f_{210} \oplus x_1 \cdot 1 \cdot x_3 f_{212} \oplus x_1 \bar{x}_2 \cdot 1 \cdot f_{221} \oplus x_1 \bar{x}_2 \bar{x}_3 f_{222}.$$

Arbitrary set of product terms combined by EXORs is called an **Exclusive-or Sum-of-Products Expression** (ESOP). The ESOP is the most general AND-EXOR expression. Arbitrary set of product terms combined by ORs is called a **Sum-of-Products Expression** (SOP).

Example 2.5

1. *$x_1 \oplus x_2 \oplus x_1 x_2$ is a PPRM (all literals are uncomplemented).*
2. *$x_1 \oplus \bar{x}_2 \oplus x_1 \bar{x}_2$ is an FPRM, but not a PPRM (x_2 have complemented literals).*
3. *$x_2 \oplus x_1 \bar{x}_2$ is a PSDRM, but not an FPRM (x_2 have both complemented and uncomplemented literals).*
4. *$x_1 \oplus x_2 \oplus \bar{x}_1 \bar{x}_2$ is a GRM, but not a PSDRM (it cannot be generated by an expansion tree for a PSDRM).*

2.5. Properties of GRMs

In this part, we consider some properties of GRMs, useful for the simplification of expressions.

Definition 2.4 *The* **variable set of a product** *p is denoted by $V(p) = \{x_i \mid x_i$ or $\bar{x}_i$ appears in $p\}$. The* **variable set of a cube** *is the variable set of the product represented by the input parts of the cube.*

Example 2.6 *$V(x_1 \bar{x}_2 \bar{x}_4) = \{x_1, x_2, x_4\}$. The variable sets of the first and second cubes in Example 2.1 are $\{x_1, x_3\}$, and $\{x_2, x_3\}$, respectively. (End of Example)*

From the definition of GRMs, we have the following lemma:

Lemma 2.3 *An AND-EXOR expression (for single-output function) is a GRM, iff no two products have the same set of variables.*

Example 2.7 *Let $f = x_4 \oplus x_1 x_2 x_3 \oplus \bar{x}_1 \bar{x}_2 \bar{x}_3$. Then $V(x_1 x_2 x_3) = \{x_1, x_2, x_3\}$, and $V(\bar{x}_1 \bar{x}_2 \bar{x}_3) = \{x_1, x_2, x_3\}$. Thus, f is not a GRM because two products have the same set of variables. (End of Example)*

In a GRM for multiple-output function, more than one cube may have the same set of variables.

Lemma 2.4 *An array represents a multiple-output GRM, if for any output, no two cubes have the same set of variables.*

(Proof) Let the array represent m single output functions, $f_i\,(i = 0, \ldots, m-1)$. If the ith bit of the output part of a cube is 1, f_i contains the corresponding cube. From the hypothesis of the lemma, no two cubes of f_i have the same set of variables. So, by Lemma 2.3, f_i represents a GRM of single output function. Thus, from the Definition 2.3, we have the lemma. (Q.E.D.)

Corollary 2.1 *In an array for a multiple-output function, if cubes having the same set of variables have non-disjoint output parts, then the array does not represent a GRM.*

Example 2.8 *Consider the array in Example 2.4. The first and the second cubes have the same set of variables, but their output parts are disjoint. Thus, the array represents a multiple-output GRM. (End of Example)*

III. Simplification Algorithm

3.1. Outline of GRMIN

GRMIN has the following features:

1. As an input, it accepts a PSDRM, or quickly generates good initial solutions from the given SOP.
2. It simplifies multiple-output functions.
3. It uses eight rules iteratively to reduce the number of products.
4. It modifies the cubes repeatedly by replacing a pair of cubes with another one, while keeping the array to represent a GRM.
5. When reduction of the number of products become impossible in the iterative improvement mode, it temporarily increases the number of products, and simplifies again. An increase in the number of products, increases the computation time, but often reduce the number of products in the final solution [1].

3.2. Initial Solution

As an initial solution, GRMIN accepts any GRMs. Alternatively, one can minimize two GRMs from two different initial solutions and take the one with fewer products.

PSDRMs: PPRMs, FPRMs and PSDRMs are special classes of GRMs, and any of them can be used as an initial solution for the GRM. We use a PSDRM as an initial solution, because minimal PSDRMs are easy to derive and usually require fewer products than PPRMs and FPRMs. We used the algorithm in [12] to obtain PSDRMs. For functions with up to 14-variables the algorithm quickly produce good initial solutions on an ordinary workstation.

Modified DSOPs: If each pair of products in an SOP are disjoint, then it is called a **disjoint SOP** (DSOP). In a DSOP, the OR operators can be replaced by the EXOR operators without changing the function represented by the expression. To obtain an initial solution, from the given SOP, first we obtain a DSOP [13]. Then, we use some heuristic to remove the cubes having the same set of variables for a output (Lemma 2.4).

3.3. Simplification Rules

Definition 3.1 *$\cap$ and $\cup$ denote the intersection and union operation between two sets, respectively. Also '$-$' (overline) denotes the complement of a set. If A and B are sets, $A \oplus B = (A \cap \overline{B}) \cup (\overline{A} \cap B)$. The symbol $\oplus$ is also used to denote the EXOR of two logic functions. A set is* null *(ϕ) if it contains no element.*

For simplification of GRMs, we use eight rules. Let $A, B, C, D \subseteq P$, where $P = \{0, 1\}$. Then, the simplification rules for GRMs are as follows:

1. X-MERGE
$$X^A \oplus X^B \Rightarrow X^{(A \oplus B)}$$
2. RESHAPE
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^A Y^{(B \cap \overline{D})} \oplus X^{(A \cup C)} Y^D \quad \text{if } (A \cap C = \phi,\ B \supset D)$$
3. DUAL-COMPLEMENT
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(\overline{A} \cap C)} Y^B \oplus X^C Y^{(B \cap \overline{D})} \quad \text{if } (A \subset C,\ B \supset D)$$
4. X-EXPAND-1
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^A Y^{(B \cup D)} \oplus X^{(A \cup C)} Y^D \Rightarrow X^{(A \cup C)} Y^B \oplus X^C Y^{(B \cup D)} \quad \text{if } (A \cap C = \phi,\ B \cap D = \phi)$$
5. X-EXPAND-2
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(A \cup C)} Y^B \oplus X^C Y^{(B \cap \overline{D})} \quad \text{if } (A \cap C = \phi,\ B \supset D)$$
6. X-REDUCE-1
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(A \cap \overline{C})} Y^B \oplus X^C Y^{(D \cap \overline{B})} \quad \text{if } (A \supset C,\ B \subset D)$$
7. X-REDUCE-2
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(A \cap \overline{C})} Y^B \oplus X^C Y^{(B \cap \overline{D})} \quad \text{if } (A \supset C,\ B \supset D)$$
8. COMPLEMENT
$$X^A \Rightarrow X^{\overline{A}} \oplus X^P$$

The representation, analysis, and proof of these rules for multiple-valued input functions can be found in [13]. All the rules except for COMPLEMENT are also used in EXMIN2 [13] to simplify ESOPs.

3.4. Examples of Simplification

Example 3.1 *Consider the following array:*

x_1	x_2	$f_0 f_1$
10	– 11	– 01
01	– 10	– 01
11	– 10	– 10.

X-MERGE is inapplicable to this array, but RESHAPE is applicable to the first two cubes, and we have the following array:

x_1	x_2	$f_0 f_1$
10	– 01	– 01
11	– 10	– 01
11	– 10	– 10.

By merging the last two cubes, we have a GRM with two cubes:

x_1	x_2	$f_0 f_1$
10	– 01	– 01
11	– 10	– 11.

(End of Example)

3.5. Properties of the Simplification Rules[1]

To simplify GRMs we use the eight rules of Section 3.3. Among them, X-MERGE is the only rule that reduces the number of cubes. When X-MERGE becomes inapplicable, other rules are used to modify the shape of the cubes so that X-MERGE becomes applicable. For a pair

[1] The proof of the lemmas can be found from the authors.

of cubes, the rules (1-7) are applicable only when the c-distance between them is one or two. The COMPLEMENT operation is used to increase the number of products. During simplification, the array is kept to represent a GRM.

Lemma 3.1 *In an array for a GRM, if the output parts of two cubes differ, then RESHAPE is inapplicable.*

Lemma 3.2 *For a pair of cubes in an array for GRM, if two input parts differ, then X-EXPAND-1 is inapplicable.*

Lemma 3.3 *For a pair of cubes in an array for GRM, if they are disjoint in an input part, and differ in the output part, then X-EXPAND-2 is inapplicable.*

Lemma 3.4 *In an array for GRM, if a pair of cubes differs in two input parts, then X-REDUCE-1 is inapplicable.*

Rules 2-8 of Section 3.3 may produce cubes with new variable sets and modified output parts. Therefore, before applying these rules, we check if the resultant array represents a GRM or not. If it is non-GRM, we discard the operation to keep the array to represent a GRM.

Example 3.2 *Consider an array for a GRM:*

x_1	x_2	x_3	$f_0 f_1 f_2$
10	− 10	− 11	− 110
10	− 01	− 01	− 110
01	− 11	− 10	− 011.

By applying RESHAPE to the first two cubes, we have

x_1	x_2	x_3	$f_0 f_1 f_2$
10	− 10	− 10	− 110
10	− 11	− 01	− 110
01	− 11	− 10	− 011.

The last two cubes of this array have identical variable sets, i.e., $\{x_1, x_3\}$, and the outputs are non-disjoint. Thus, the array no longer represents a GRM, and we discard this operation and keep the array to represent a GRM. (End of Example)

Lemma 3.5 *In an array for a GRM, if a rule does not change the variable set of a cube, and produces outputs which are a subset of the original outputs, then the cube can be modified without checking the array.*

Example 3.3 *Consider an array of k ($k > 3$) cubes for a GRM. Let three of the cubes in the array be:*

x_1	x_2	x_3	$f_0 f_1 f_2$
01	− 10	− 01	− 101
01	− 10	− 11	− 001
10	− 01	− 11	− 110.

By applying DUAL-COMPLEMENT to the first two cubes, we have

x_1	x_2	x_3	$f_0 f_1 f_2$
01	− 10	− 10	− 101
01	− 10	− 11	− 100
10	− 01	− 11	− 110.

In the first cube, the variable set remains the same and the outputs are the subset of the original outputs (equal here). Thus, we can modify the first cube without checking the array. Note that the polarity of the literal x_3 has been changed.
In the second cube, the variable set remains the same but the outputs are not a subset of the original outputs. So, we have to check if the resultant array represents a GRM or not. The second and third cube have the same set of variables and the outputs are non-disjoint. Thus, the array no longer represent a GRM, and we have to discard this operation to keep the array represent a GRM. (End of Example)

Lemma 3.6 *In an array for GRM, if a pair of cubes differs in two input parts, and DUAL-COMPLEMENT is applied, then the cubes can be modified without checking other cubes in the array.*

3.6. Algorithm

For many functions, the order of the simplification rules influences the final solution. Currently, we are using the following heuristic algorithm.

Algorithm 3.1 *(GRMIN: Simplification of GRMs)*

1. *Obtain an initial solution from the given SOP (Section 3.2).*
2. *For each pair of cubes, check if X-MERGE is applicable. If so, X-MERGE them. Continue this step while reductions of the number of cubes are possible.*
3. *For each pair of cubes, check if RESHAPE, DUAL-COMPLEMENT, X-REDUCE-1, or X-REDUCE-2 are applicable. If so, apply that.*
4. *For each pair of cubes, check if X-MERGE is applicable. If so, X-MERGE them. Continue this step while reductions of the number of cubes are possible.*
5. *If the number of cubes is reduced in step 4, then go to step 3.*
6. *For each pair of cubes, check if X-EXPAND-1, X-EXPAND-2, RESHAPE, or DUAL-COMPLEMENT are applicable. If so, apply that.*
7. *For each pair of cubes, check if X-MERGE is applicable. If so, X-MERGE them. Continue this step while reductions of the number of cubes are possible.*
8. *If the number of cubes is reduced in step 7, then go to step 6.*
9. *If the number of cubes is reduced between steps 3-8, then go to step 3.*
10. *Temporarily increase the number of products by COMPLEMENT, and simplify using steps 2-9. Continue this step while reductions of the number of cubes are possible.*

TABLE 4.1
COMPARISON WITH CANNES [3]

Data	In	Out	Cannes	GRMIN	Improvement
5xp1	7	10	60	35	42%
con1	7	2	12	9	25%
misex1	8	7	20	13	35%
rd53	5	3	20	20 *	0%
rd73	7	3	63	63 *	0%
sao2	10	4	52	28	46%
squar5	5	8	22	19	14%
sym9	9	1	131	126 *	4%
xor5	5	1	5	5 *	0%

* Exact minimum GRM.

11. *For each pair of cubes, check if X-REDUCE-1, X-EXPAND-2 (when two input parts differ), RESHAPE, or DUAL-COMPLEMENT (when two input parts differ), are applicable. If so, apply that. Continue this step while reductions of the number of connections are possible.*
12. *Check that the simplified cubes represent a GRM, and verify that it is functionally equivalent to the given SOP.*

In steps 2, 4, and 7, more than one merging passes are often required. In our implementation, we used additional fields for each cube. Using these fields, we can avoid many redundant computations. One of these fields stores when the cube was modified. For example, during the first merging pass in step 4 of Algorithm 3.1, it checks two cubes if at least one of them was modified in step 3. As stated in Section 3.5, before applying a rule, the algorithm checks the array to determine if it represents a GRM. Another field is used to make this checking easier by storing variable set information.

IV. EXPERIMENTAL RESULTS

We coded GRMIN in C. As an initial solution, it accepts a PSDRM, or generates an initial GRM from the given SOP. Table 4.1 compares the number of products generated by GRMIN with that of Cannes [3], another heuristic simplification program for GRMs. It shows that GRMIN outperforms Cannes, and the improvement is up to 46%. For *rd53*, *rd73*, *sym9*, and *xor5*, GRMIN produced exact minimum solutions (denoted by *). To show the minimality, we obtained lower bounds on the number of products in GRM [15].

Table 4.2 compares the number of products required to realize different arithmetic functions by various AND-EXOR expressions. In this experiment, PPRMs, FPRMs, and PSDRMs were minimized by a program in [12]; ESOPs were simplified by EXMIN2 [13]; and GRMs were simplified by GRMIN. For *adr4*, *inc8*, *sym9*, and *wgt8*, GRMIN produced exact minimum solutions. This table also shows that GRMs for arithmetic functions often require many fewer products than SOPs.

TABLE 4.2
NUMBER OF PRODUCTS TO REALIZE ARITHMETIC FUNCTIONS

Data	PPRM	FPRM	PSDRM	GRM	ESOP	SOP
adr4	34	34	34	34 *	31	75
inc8	16	16	16	15 *	15	37
log8	253	193	163	105	96	123
mlp4	97	97	90	71	61	121
nrm4	216	185	150	96	69	120
rdm8	56	56	46	31	31	76
rot8	225	118	81	51	35	57
sqr8	168	168	164	121	112	178
sym9	210	173	127	126 *	51	84
wgt8	107	107	107	107 *	58	255

*Exact minimum GRM.

TABLE 4.3
NUMBER OF PRODUCTS TO REALIZE RANDOMLY GENERATED FUNCTIONS

n	$\|f\|$	PPRM	FPRM	PSDRM	GRM	ESOP	SOP
4	8	6	5	4	4 *	3	4
5	16	16	10	7	5 *	5	6
6	32	36	17	13	10 *	10	13
7	64	64	54	30	19	19	24
8	128	122	101	56	36	36	46
9	256	236	226	112	66	64	86
10	512	528	459	235	142	142	167
11	1024	1021	956	458	275	274	331
12	2048	1996	1925	909	542	539	611
13	4096	4136	3923	1813	1098	1045	1157
14	8192	8210	7924	3617	2205	2150	2234

*Exact minimum GRM.

Table 4.3 compares the number of products to realize randomly generated functions. For each value of n, an n-variable pseudo-random function with 2^{n-1} minterms was generated and minimized. Here $|f|$ denotes the number of true minterms of the function. For up to 6-variable functions, we verified that GRMIN produced exact minimum solutions by using an exact minimization program [15]. In this experiment, SOPs were simplified by MINI II [12], and other data were obtained by the same programs as mentioned above. This table shows that for randomly generated functions, GRMs require fewer products than SOPs.

Table 4.4 compares the number of products to realize other benchmark functions.[2] This table shows that, in many cases, GRMs require fewer products than SOPs. For adders and some other functions, GRMIN produced exact minimum GRMs. By solving a recurrence relation generated from the experimental results, we found that the minimum number of products required by GRM for **adr**n is $2^{n+1} + n - 2$.

[2] In this table **adr**n is an n-bit adder without carry input, **inc**n increments an n-bit binary number, and **wgt**n counts the number of 1-bits in an n-bit binary number.

TABLE 4.4
NUMBER OF PRODUCTS TO REALIZE OTHER BENCHMARK FUNCTIONS

Data	In	Out	GRM	ESOP	SOP
5xp1	7	10	35	32	63
adr4	8	5	34 *	31	75
adr5	10	6	67 *	63	167
adr6	12	7	132 *	127	355
adr7	14	8	261 *	255	535
adr8	16	9	518 *	511	1499
adr9	18	10	1031 *	1023	3031
adr10	20	11	2056 *	2047	6099
al2	16	47	69	68	66
alu1	12	8	16	16	19
amd	14	24	71	58	66
b12	15	9	28	28	44
b9	16	5	81	81	119
chkn	29	7	151	145	141
clip	9	5	117	67	117
cps	24	109	175	137	172
duke2	22	29	87	81	86
gary	15	11	116	102	107
in2	19	10	122	108	134
inc12	12	13	23 *	23	79
inc13	13	14	25 *	25	92
intb	15	7	375	327	629
life	9	1	51	49	84
log8mod	8	5	30	30	38
luc	8	27	34	28	26
m181	15	9	29	29	41
m3	8	16	57	51	62
m4	8	16	95	84	101
max1024	10	6	324	165	262
max128	7	24	86	63	78
max46	9	1	47	41	46
max512	9	6	151	84	133
misex2	25	18	27	27	28
misex3	14	14	764	553	696
mlp6	12	12	1277	862	1870
newbyte	5	8	8	8	8
newcpla1	9	16	34	33	38
newxcpla	9	23	34	30	41
radd	8	5	34 *	31	75
rckl	32	7	32	32	32
rd53	5	3	20 *	14	31
rd73	7	3	63 *	35	127
rd84	8	4	107 *	58	255
risc	8	31	27	26	27
sao2	10	4	28	28	58
sqr6	6	12	35	34	47
sym10	10	1	110	82	210
sym9	9	1	126 *	51	84
t1	21	23	94	90	103
t481	16	1	13	13	481
ts10	22	16	128	128	128
wgt10	10	4	310 *	143	1023
wgt12	12	4	1068 *	445	4095
z4	7	4	32 *	29	59

*Exact minimum GRM.

V. CONCLUSION

In this paper, we presented GRMIN, a heuristic simplification algorithm for GRMs of multiple-output functions. GRMs have easily testable realizations. Experimental results show that, in most cases, GRMs require fewer products than SOPs. Also, we showed that GRMIN outperforms existing algorithms. For adders and some other functions, GRMIN produced exact minimum solutions.

ACKNOWLEDGEMENT

This work was supported in part by a Grant in Aid for the Scientific Research of the Ministry of Education, Science and Culture of Japan.

REFERENCES

[1] D. Brand, and T. Sasao, "Minimization of AND-EXOR expressions using rewrite rules," *IEEE Trans. Comput.*, vol. 42, No. 5, pp. 568-576, May 1993.

[2] M. Cohn, "Inconsistent canonical forms of switching functions," *IRE Trans.*, EC-11, pp. 284-285, Apr. 1962.

[3] L. Csanky, M. A. Perkowski, and I. Schäfer, "Canonical restricted mixed-polarity exclusive-OR sums of products and the efficient algorithm for their minimisation," *IEE Proceedings-E*, vol. 140, No. 1, pp. 69-77, Jan. 1993.

[4] M. Davio, J-P Deschamps, and A. Thayse, *Discrete and Switching Functions*, McGraw-Hill Int., New York, 1978.

[5] H. Fujiwara, *Logic Testing and Design for Testability*, The MIT Press, Cambridge, 1985.

[6] D. Green, *Modern Logic Design*, Addison-Wesley Publishing Company, Wokingham, England, 1986.

[7] 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.

[8] A. Mukhopadhyay, and G. Schmitz, "Minimization of EXCLUSIVE OR and LOGICAL EQUIVALENCE switching circuits," *IEEE Trans. Comput.*, vol. C-19, pp. 132-140, Feb. 1970.

[9] M. A. Perkowski, L. Csanky, A. Sarabi, and I. Schäfer, "Fast minimization of mixed-polarity AND-XOR canonical networks," *Proc. International Conference on Computer Design 1992*, pp. 33-36, Oct. 1992.

[10] S. M. Reddy, "Easily testable realizations for logic functions," *IEEE Trans. Comput.*, vol. C-21, No. 11, pp. 1183-1188, Nov. 1972.

[11] T. Sasao, and P. Besslich, "On the complexity of MOD-2 sum PLA's," *IEEE Trans. Comput.*, vol. 39, No. 2, pp. 262-266, Feb. 1990.

[12] T. Sasao, "AND-EXOR expressions and their optimization," in (Sasao e.d.) *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[13] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Trans. CAD.*, vol. 12, No. 5, pp. 621-632, May 1993.

[14] T. Sasao, "Easily testable realizations for generalized Reed-Muller expressions," *Proc. IEEE The Third Asian Test Symposium*, pp. 157-162, Nov. 1994.

[15] T. Sasao, and D. Debnath, "An exact minimization algorithm for generalized Reed-Muller expressions," *Proc. IEEE Asia-Pacific Conference on Circuits & Systems*, pp. 460-465, Dec. 1994.