CIRCUITS SYSTEMS SIGNAL PROCESS
VOL. 19, NO. 6, 2000, PP. 535–548

# OPTIMIZATION OF FIXED-POLARITY REED-MULLER CIRCUITS USING DUAL-POLARITY PROPERTY*

*E. C. Tan*[1] *and H. Yang*[1]

**Abstract.** In the optimization of canonical Reed-Muller (RM) circuits, RM polynomials with different polarities are usually derived directly from Boolean expressions. Time efficiency is thus not fully achieved because the information in finding RM expansion of one polarity is not utilized by others. We show in this paper that two fixed-polarity RM expansions that have the same number of variables and whose polarities are dual can be derived from each other without resorting to Boolean expressions. By repeated operations, RM expansions of all polarities can be derived. We consequently apply the result in conjunction with a hypercube traversal strategy to optimize RM expansions (i.e., to find the best polarity RM expansion). A recursive route is found among all possible polarities to derive RM expansion one by one. Simulation results are given to show that our optimization process, which is simpler, can perform exhaustive search as efficiently as other good exhaustive-search methods in the field.
**Key words:** Reed-Muller circuits, dual-polarity, optimization.

## 1. Introduction

Logic functions may be expressed either in Boolean or in canonical Reed-Muller (RM) forms [2], [12]. The advantages of RM circuits (i.e., AND/EX-OR circuits) over traditional Boolean circuits (i.e., AND/OR circuits) in certain applications have been well reported [2], [12], [26], [28]. Generally, an RM expansion is derived from a given Boolean expression by applying a conversion method such as coefficient-map [14], [32], [33], [39], coefficient-matrix [13], [16], [19], or tabular techniques [3], [4]. However, for an $n$-variable logic function, $2^n$ fixed-polarity RM expansions are possible. The problem has been to find the best expansion in terms of a certain predefined condition. Thus, recent research work in the field of RM logic has been focused, to a greater extent, on the optimization

* Received June 9, 1999; revised June 20, 1999.
[1] School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore. E-mail for Tan: asectan@ntu.edu.sg

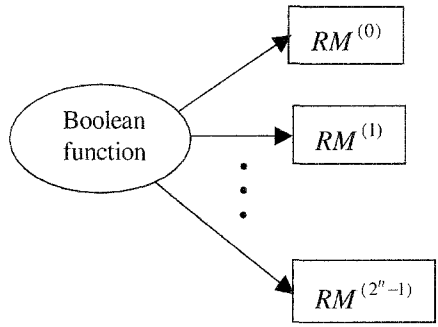


**Figure 1.** Paradigm 1—Conventional optimization approach.

of RM functions, where the search space is among $2^n$ possible polarities and the cost function is the number of EX-OR gates in most cases [20], [24]. In an attempt to find the global best polarity, all of the $2^n$ fixed-polarity RM expansions are first derived from a given Boolean expression using one of the conversion techniques mentioned previously. Then a brute-force, exhaustive-search method is applied to find the most optimum expansion. The optimization process has the general paradigm shown in Figure 1. Unfortunately, most conversion methods are inefficient and perform extremely poorly when $n$ is large. Even with the more efficient tabular technique [4], the conversion from a Boolean function to any fixed-polarity RM expansion has a time complexity of $O(2^n)$, and when $n$ is large, the time taken is still rather substantial. Although Besslich [5] developed an efficient computer method based on coefficient maps by Wu et al. [39] to generate all $2^n$ sets of generalized RM coefficients of an $n$-variable Boolean function, its algorithm and efficiency for a general case (especially when $n$ is large) are not clear. Miller and Thomson [24] have also presented an optimization technique to find out all $b$-vectors from a given "good polarity" $b$-vector, but a good polarity predictor is needed to achieve the potential efficiency of the method. The optimization methods mentioned above, and those in [11], [17], [21], [34]–[37], [40], belong to the exhaustive-search type, which can always guarantee an optimum RM expression in terms of the number of EX-OR gates, but the searching time may become prohibitively large when the number of variables is greater than about 20. However, with the computing power of today's personal computers this has become less of a problem. In addition, there are many circuit realizations that do not require a large number of variables (say, $n \leqslant 20$) [2], [12].

In recent years, graph-based algorithms [6] such as ordered binary decision diagram (OBDD) and ordered functional decision diagram (OFDD) methods have been proposed for the optimization of RM functions [9], [10], [25], [29], [30], [31]. Despite their efficient operations, there are some undesirable characteristics. One of the problems is variable ordering, which is a heuristic process. Initially, one must choose some ordering of the system inputs as arguments to all of the

functions to be represented in OBDDs or OFDDs. For some functions, the size of the graph representing the function is highly sensitive to this ordering. In addition, there are some functions that can be represented by Boolean expressions or logic circuits of reasonable size, but for all input ordering the representation as a function graph is too large to be practical. The problem of computing an ordering that minimizes the size of the graph is itself an NP-complete problem; thus the use of OBDDs and OFDDs to represent fixed-polarity RM forms remains an area for further research. There are also optimization techniques that are based on genetic algorithms [7], [23], [27], [30] and rule-based approaches [8], [27], [30]. These graph-based, rule-based, and genetic-algorithm methods belong to the nonexhaustive-search type, which generally has less searching time (even for a large number of variables), but because of their heuristic nature, the resulting RM expression may not be optimum. In certain cases, for example, in using a genetic algorithm, a wrong choice of an initial state may not lead to a definite answer at all [27].

In this paper, an exhaustive-search method for the optimization of RM polynomials is proposed. We prove that two fixed-polarity RM expansions whose polarities are dual can be derived from each other without resorting to Boolean expressions. This means that from a known fixed-polarity RM expansion, all other fixed-polarity RM expansions with the same number of variables can be derived. Using the property of a hypercube [22], this new conversion method leads to a recursive way of finding all the RM expansions efficiently. Two important features of our method are that the conversion from one RM expansion to another is carried out using one-bit checking and a good polarity predictor is not required. The method enables us to propose a new optimization paradigm, as shown in Figure 2. Some comparison results with known exhaustive-search algorithms [4], [24], [37], [40] are also presented to illustrate the performance of our method. The proposed method should not be compared with nonexhaustive-search methods, as the latter are likely to perform better in terms of speed because of the different searching space and technique.

## 2. RM expansions

### 2.1. *Canonical form*

In a canonical RM expansion, each variable appears either in complemented or uncomplemented form, but not both. Thus, in logic circuit literature, a canonical RM expansion is an alternative name for a fixed-polarity RM expansion. We know that any $n$-variable Boolean function can be represented in a sum-of-product form as

$$f(x_{n-1}, \ldots, x_0) = \sum_{i=0}^{2^n-1} a_i m_i, \tag{1}$$

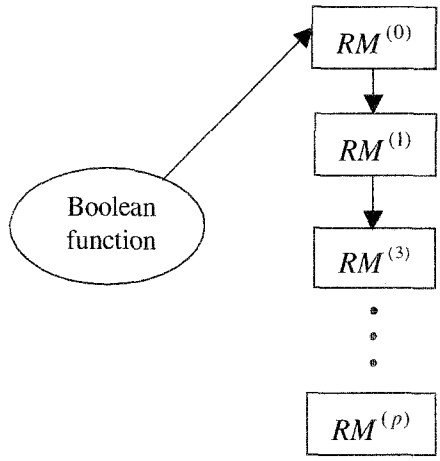


**Figure 2.** Paradigm 2—Our optimization approach.

where $f : \{0, 1\}^n \rightarrow \{0, 1\}$, $m_i$ are the minterms, and $a_i \in \{0, 1\}$. It has been proved directly [1] or indirectly [2], [8] that (1) can also be expressed in canonical RM form as

$$f^{(p)}(\dot{x}_{n-1}, \ldots, \dot{x}_0) = \oplus \sum_{i=0}^{2^n-1} b_i^{(p)} \phi_i^{(p)}, \tag{2}$$

where

$$\phi_i^{(p)} = \prod_{k=0}^{n-1} (\dot{x}_k)^{i_k}, \tag{3}$$

where $p (= 0, \ldots, 2^n - 1)$ refers to a given RM polarity, $\dot{x}_k = x_k$ or $\bar{x}_k$, $x_k \in \{0, 1\}$, $\phi_i^{(p)}$ are known as product terms for the particular $p$, and $b_i^{(p)} \in \{0, 1\}$ determine whether a product term is present or not. In addition, $i_k \in \{0, 1\}$ is the power of $\dot{x}_k$, which indicate the presence or absence of a variable. We define $0^0 = 0$. The symbol $\oplus$ represents the EX-OR operation, and multiplication is assumed to be the AND operation.

### 2.2. *Derivation*

**Definition 1.** Two polarities are defined to be *dual polarity* if they exhibit the following property: the $n$-bit binary strings of these two polarities have $n - 1$ bits in common, where $n$ is the number of variables ($n \geqslant 1$).

**Definition 2.** The only different bit between dual polarities is called the *permuting bit*.

Examples of dual polarities can be found in polarities 0 and 1 ($n = 1$), polarities (0 0) and (0 1) ($n = 2$), and polarities (0 0 0) and (0 0 1) ($n = 3$), etc. In all these examples, the permuting bits are bit 1. For convenience, $(p_j, q_j)$ is used to denote a polarity dual with permuting bit $j$.

**Corollary 1.** The fixed-polarity RM with polarity $q_j$ can be derived from the fixed-polarity RM with polarity $p_j$, where $(p_j, q_j)$ is dual polarity.

**Proof.** From (2) with $p = p_j$, we have

$$f^{(p_j)} = \oplus \sum_{i=0}^{2^n-1} b_i^{(p_j)} \phi_i^{(p_j)}, \tag{4}$$

where

$$\phi_i^{(p_j)} = \prod_{k=0}^{n-1} (\dot{x}_k)^{i_k}. \tag{5}$$

Equation (5) can be written as

$$\phi_i^{(p_j)} = \left( \prod_{k=0}^{j-1} (\dot{x}_k)^{i_k} \right) (\dot{x}_j)^{i_j} \left( \prod_{k=j+1}^{n-1} (\dot{x}_k)^{i_k} \right). \tag{6}$$

Replacing $(\dot{x}_j)^{i_j}$ with $1 \oplus (\bar{\dot{x}}_j)^{i_j}$, and $p_j$ with $q_j$, we obtain

$$\phi_i^{(q_j)} = \left( \prod_{k=0}^{j-1} (\dot{x}_k)^{i_k} \right) (1 \oplus (\bar{\dot{x}}_j)^{i_j}) \left( \prod_{k=j+1}^{n-1} (\dot{x}_k)^{i_k} \right). \tag{7}$$

Equation (7) is not yet in product-term form. Because $A(B \oplus C) = AB \oplus AC$, it is easy to show that

$$A(B \oplus C)D = ABD \oplus ACD \tag{8}$$

for some arbitrary binary variables $A$, $B$, $C$, and $D$. Applying (8), (7) becomes

$$\phi_i^{(q_j)} = \left( \prod_{k=0}^{j-1} (\dot{x}_k)^{i_k} \prod_{k=j+1}^{n-1} (\dot{x}_k)^{i_k} \right) \oplus \left( \left( \prod_{k=0}^{j-1} (\dot{x}_k)^{i_k} \right) (\bar{\dot{x}}_j)^{i_j} \left( \prod_{k=j+1}^{n-1} (\dot{x}_k)^{i_k} \right) \right), \tag{9}$$

which is now in the form of product terms. In (9), the variable $\dot{x}_j$ has been replaced by $\bar{\dot{x}}_j$. Thus, by substituting (9) in (4), we obtain the RM expansion $f^{(q_j)}$. □

In the derivation, we need to check the RM expansion at one particular bit only to derive the RM expansion of other polarity. Comparing (6) and (9), we notice that a new term is created when a product term in polarity $p_j$ is expressed in polarity $q_j$. If we write (9) as $\phi_i^{(q_j)} = \mu \oplus \nu$, then if $(\dot{x}_j)^{i_j} = 1$, the new term is $\nu$; if $(\dot{x}_j)^{i_j}$ is a variable, the new term is $\mu$. The new term can be generated as follows. Convert the product term into a binary string by replacing a variable by 1 or 0 depending on whether the variable is present or not. Copy all bits of

this string except bit $j$, which is changed from 1 to 0. Keep in mind that the final RM expansion with polarity $q_j$ is given by (2) with $p = q_j$. The new term may be coincident with other terms and should be canceled according to the rule $A \oplus A = 0$. This leads to the following algorithm for deriving $f^{(q_j)}$ from $f^{(p_j)}$.

### 2.3. *Algorithm*

*Step 1.* Convert product terms of $f^{(p_j)}$ into binary strings. A variable that is present in a product term is replaced by 1, and an absent variable is replaced by 0.
*Step 2.* Generate a new term if bit $j$ of a binary string is 1. Replace bit $j$ with 0 and copy all other bits to generate the new term.
*Step 3.* Cancel pairs between original strings and newly generated strings.
*Step 4.* The uncanceled strings are the product terms of $f^{(q_j)}$.

### 2.4. *Example*

Consider a three-variable Boolean logic function:

$$f(A, B, C) = \sum(0, 2, 4, 7) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC. \tag{10}$$

Its RM expansion in polarity 6 is given by

$$f^{(6)} = \oplus \sum(1, 2, 4, 6) = C \oplus \overline{B} \oplus \overline{A} \oplus \overline{A}\,\overline{B}. \tag{11}$$

Polarity 4 (100) and polarity 6 (110) are single-bit different at bit 2. Thus the RM expansion in polarity 4 can be derived directly from (11) instead of transforming it from the Boolean form in (10). Table 1 illustrates the process, and we obtain

$$f^{(4)} = \oplus \sum(0, 1, 2, 6) = 1 \oplus C \oplus B \oplus \overline{A}B.$$

**Table 1.** Derivation of RM expansion

| Polarity = 6 | | | New terms | | | Polarity = 4 | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | | | | A | B | C |
| 0 | 0 | 1 | | | | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ~~1~~ | ~~0~~ | ~~0~~ | | | | 0 | 1 | 0 |
| 1 | 1 | 0 | ~~1~~ | ~~0~~ | ~~0~~ | 1 | 1 | 0 |

For comparison, let us use the tabular technique to derive $f^{(4)}$ directly from the given Boolean function (i.e., (10)) as follows:

| Minterms | New terms ($C$) | New terms ($B$) | New terms ($A$) |
|---|---|---|---|
| ~~0 0 0~~ | ~~0 0 1~~ | ~~0 1 0~~ | ~~0 0 0~~ |
| ~~0 1 0~~ | ~~0 1 1~~ | 1 1 0 | ~~0 0 1~~ |
| 1 0 0 | 1 0 1 | ~~0 1 1~~ | 0 1 0 |
| ~~1 1 1~~ | | ~~1 1 1~~ | |

| Remaining terms | | Polarity | | Resulting terms |
|---|---|---|---|---|
| 1 0 0 | $\oplus$ | 1 0 0 | = | 0 0 0 |
| 1 0 1 | $\oplus$ | 1 0 0 | = | 0 0 1 |
| 1 1 0 | $\oplus$ | 1 0 0 | = | 0 1 0 |
| 0 1 0 | $\oplus$ | 1 0 0 | = | 1 1 0 |

Clearly, we have the same results.

### 2.5. *Comments*

It is easy to see the advantage of the algorithm in Section 2.3 over the tabular technique. It needs only one-bit operation. However, to derive an RM expansion directly from minterms using the tabular technique, one must compare each bit of a Boolean function with the desired polarity; thus, $n$ bits must be checked for an $n$-variable function. In addition, the remaining terms in the tabular technique must undergo an EX-OR operation with the desired polarity in order to obtain the corresponding RM expansion. In our method, no EX-OR operation is needed, and no good polarity predictor is required. Substantial time can be saved if we derive all the $2^n$ RM expansions this way.

Notice from the example in Section 2.4 that, apart from $p = 100$, we can easily derive RM expansions for $p = 111$ and 010. Then from $p = 111$, we can go on to $p = 110$, 101, and 011; from $p = 010$, we can go on to $p = 011$, 000, and 110, etc. Thus, by repeated application of the algorithm on newly generated RM expansions, we can derive RM expansions for all polarities. However, when $n$ is large and without elaborate checking steps, this process generates many repetitions of the same polarities and does not know when to stop. All of this implies inefficiency. Because of the single-bit difference in polarity, it motivates us to exploit a particular link among the $2^n$ polarities based on a hypercube structure. By formulating all polarities as nodes of a hypercube, we will show in the next section that there exists a recursive route that covers all the nodes and transverses each node only once. The result enables us to propose an efficient exhaustive-search optimization of RM expansions.

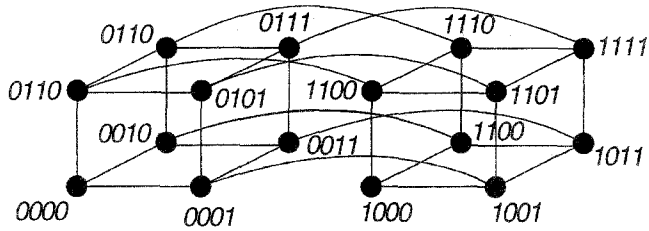


**Figure 3.** Four-dimensional hypercube.

## 3. Hypercube

Although traversing single-bit-difference binary strings can be based on Gray code, we have chosen a hypercube data structure because of its regularity, its small diameter, its many interesting graph-theoretic properties, and its ability to handle many computations quickly and simply. Basically, a hypercube consists of $2^n$ nodes interconnected into an $n$-dimensional Boolean cube that can be defined as follows. Let the binary representation of $i$ be $i_{n-1}i_{n-2}\cdots i_0$, where $0 \leqslant i \leqslant 2^n - 1$. Then node $d_i$ is connected to node $d_j$, where $j = i_{n-1}i_{n-2}\cdots i_k \cdots i_0$, for $0 \leqslant k \leqslant n-1$. In other words, two nodes are connected if and only if their indices differ by only one bit position. Figure 3 shows a four-dimensional hypercube.

Of interest to us is that the hypercube has a recursive structure [22]. We can extend a $d$-dimensional cube to a $(d+1)$-dimensional cube by connecting corresponding nodes of two $d$-dimensional cubes. One cube has the most significant address bit equal to 0; the other cube has the most significant address bit equal to 1.

**Corollary 2.** Let each node of an $n$-dimensional hypercube represent a polarity of an $n$-variable RM function. If there is a direct connection between node $d_i$ and node $d_j$, then $d_i$ and $d_j$ are single-bit different.

**Proof.** In constructing a hypercube, two nodes are connected if and only if their indices differ by only one-bit position. Therefore, from Definition 1, the connection between node $d_i$ and node $d_j$ means that $d_i$ and $d_j$ are single-bit different. □

**Theorem .** *There exists a route in an n-dimensional hypercube that passes all nodes only once.*

**Proof.** The proof is by induction on $n$. For $n = 1$, the 1-dimensional hypercube has two nodes with indices of 0 and 1, respectively. These two nodes are connected, so we have a route that passes all nodes only once.

For $n = k$, assume that the theorem is true. This means that there is a route $R_{ST}$ starting from node $d_S$ ($S = S_{k-1}S_{k-2}\cdots S_0$) and stopping at node $d_T$ ($T =$

$T_{k-1}T_{k-2}\cdots T_0$) and it passes all $2^k$ nodes only once. Note that the reverse route, $R_{TS}$ (from $d_T$ to $d_S$), also passes all nodes only once.

For $n = k + 1$, a $(k + 1)$-dimensional cube is constructed by connecting corresponding nodes of two $k$-dimensional cubes. One cube has the most significant address bit equal to 0; the other cube has the most significant address bit equal to 1. We can traverse the first $k$-dimensional cube taking route $R_{ST}$, go to the second cube using the connection between node $(0T_{k-1}T_{k-2}\cdots T_0)$ and node $(1T_{k-1}T_{k-2}\cdots T_0)$, and traverse the second $k$-dimensional cube taking route $R_{TS}$. This route covers all $2^{k+1}$ nodes and passes each of them only one time. □

The proof also gives a method to construct the route.

**Definition 3.** The route in a hypercube possessing the feature as described in the preceding Theorem is called a recursive route.

### *3.1. Examples of recursive routes*

Let the route for a 1-dimensional hypercube be $0 \rightarrow 1$. Then the route for a 2-dimensional hypercube is $00 \rightarrow 01 \rightarrow 11 \rightarrow 10$. The route for a 3-dimensional hypercube is $000 \rightarrow 001 \rightarrow 011 \rightarrow 010 \rightarrow 110 \rightarrow 111 \rightarrow 101 \rightarrow 100$, and the route for a 4-dimensional hypercube is $0000 \rightarrow 0001 \rightarrow 0011 \rightarrow 0010 \rightarrow 0110 \rightarrow 0111 \rightarrow 0101 \rightarrow 0100 \rightarrow 1100 \rightarrow 1101 \rightarrow 1111 \rightarrow 1110 \rightarrow 1010 \rightarrow 1011 \rightarrow 1001 \rightarrow 1000$, and so on.

## 4. Optimization of RM circuits

### *4.1. Exhaustive search based on recursive route*

Now we are in a position to perform optimization of RM expansions using the paradigm in Figure 2. First, a given Boolean function is transformed into a zero-polarity RM function using a known method, say the tabular technique [4]. From there on, our algorithm (given in Section 4.2) will take over to perform an exhaustive search over the $2^n$ polarities to determine the polarity (or polarities) which results (or result) in the minimum number of EX-OR gates in the RM expansion (expansions), assuming that EX-OR gates are the most expensive components. The bracketed quantities mean that there may be more than one RM expansion with different polarities but with the same minimum number of EX-OR gates, which constitutes the defined cost function.

In Section 2, we presented the algorithm for deriving RM expansions with dual polarity. In Section 3, we defined the recursive route that covers all possible $2^n$ polarities. In the recursive route, two neighboring polarities are single-bit different, thus making it possible to derive all the RM expansions. Combining the algorithm in Section 2 and the recursive route in Section 3, an exhaustive-search technique is presented below for the optimization of RM circuits.

**Table 2.** Result of Example 4.3

| Polarity | RM expansion | No. of EX-OR gates | Best polarity |
|---|---|---|---|
| 0 0 0 | $\oplus\sum(0,1,6)$ | 2 | |
| 0 0 1 | $\oplus\sum(1,6)$ | 1 | * |
| 0 1 1 | $\oplus\sum(1,6,4)$ | 2 | |
| 0 1 0 | $\oplus\sum(0,1,6,4)$ | 3 | |
| 1 1 0 | $\oplus\sum(1,2,4,6)$ | 3 | |
| 1 1 1 | $\oplus\sum(0,1,2,4,6)$ | 4 | |
| 1 0 1 | $\oplus\sum(1,2,6)$ | 2 | |
| 1 0 0 | $\oplus\sum(0,1,2,6)$ | 3 | |

### *4.2. Optimisation algorithm*

*Step 1.* Get the number of variables $n$, and get the minterms.
Initialize the polarity, $p' = 0$.

*Step 2.* Obtain the zero-polarity RM expansion (using the tabular technique).
Calculate the total number of EX-OR gates, $N_{eor}$.
Set *best-gates* $= N_{eor}$. Set *best-polarity* $= 0$.

*Step 3.* Determine the next polarity, $p'$, of the RM expansion according to the recursive route.

*Step 4.* Obtain the RM expansion of polarity $p'$ based on the algorithm in Section 2.3.
Calculate the total number of EX-OR gates, $N_{eor}$.
If $N_{eor} <$ *best-gates*, set *best-polarity* $= p'$ and *best-gates* $= N_{eor}$.

*Step 5.* Stop if all polarities have been treated. Otherwise repeat Steps 3 and 4.

### *4.3. Example*

Consider the same three-variable Boolean logic function in Section 2.4, i.e.,

$$f(A, B, C) = \sum(0, 2, 4, 7) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$$

The result of the exhaustive-search method taking the recursive route is shown in Table 2.

## 5. Simulations and comparisons

The actual timing of exhaustive search, taking the conventional way (Figure 1) or using the hypercube recursive route (Figure 2), will vary with the Boolean functions. In addition, the number of minterms in a Boolean function also affects the optimization time. Therefore, we are not able to give an exact formula to

**Table 3.** Some optimization examples

| Number of variables | Boolean function | Best polarity | Reed-Muller form |
|---|---|---|---|
| 4 | $f = \sum(1,4,5,13,14)$ | 7, 8 | $f^{(7)} = \oplus\sum(2,7,9,13,14,15)$<br>$f^{(8)} = \oplus\sum(5,6,9,11,12,15)$ |
| 4 | $f = \sum(7,3,2,10,5,4,13,14,8)$ | 9 | $f = \oplus\sum(9,10,15,1,4,6)$ |
| 5 | $f = \sum(20,11,15,6,21,0,26,5,22,31,$<br>$30,2,1,28,14,24,9,23,29)$ | 30 | $f = \oplus\sum(31,25,22,20,18,$<br>$12,13,5,0)$ |
| 6 | $f = \sum(24,45,23,62,8,35,13,28,34,61,$<br>$39,11,30,31,16,33,40,5,32,22,$<br>$14,7,19,36,26,3,1,29,48,10,$<br>$12,43,37,38,59,50,27,44)$ | 37 | $f = \oplus\sum(33,41,43,49,50,$<br>$62,59,0,13,12,10,$<br>$21,16,29,31,25,24)$ |
| 7 | $f = \sum(114,72,18,50,22,65,20,15,101,$<br>$75,111,110,37,62,51,82,53,100,$<br>$2,99,68,3,119,34,0,69,59,106,$<br>$47,23,80,13,32,112,6,126,38,$<br>$31,45,123,109,94,93,108,8,4,$<br>$83,49,73,117,66,52,21,92,19,$<br>$115,12,27,36,107,8,10,86,120,$<br>$74,103,42,98,91,11,39,46,29,$<br>$25,5,127)$ | 19 | $f = \oplus\sum(19,22,21,20,26,25,$<br>$28,2,0,6,10,9,14,49,$<br>$48,55,52,59,63,60,$<br>$35,34,39,36,47,46,$<br>$81,80,85,89,88,95,$<br>$94,93,67,69,68,74,$<br>$77,119,118,122,$<br>$121,120,99,98,$<br>$107,110,109)$ |

calculate the timing of our method in the general case. But by comparing the performance with other exhaustive-search methods [4], [24], [37], [40], we can still show the effectiveness of our method experimentally based on randomly generated Boolean functions. All methods were implemented on a Pentium II-300MHz PC with 64MB RAM using C++ programs.

The first experiment was to verify the correctness of our method compared to the conventional method. The objective was to find the RM expansion with the least number of EX-OR operations. Numerous randomly generated logic functions whose number of variables ranges from 3 to 10 were tested. Table 3 lists some of the tested results. For each test function, both methods gave the same result, thus verifying the correctness of our approach. In cases where more than one optimum RM expansion was found for a given Boolean function, extra cost functions [24], like AND-gate = 2 cost units and NOT-gate = 1 cost unit, could be included to refine selections in actual circuit implementation. Notice that the first 4-variable Boolean function in Table 3 was optimized at polarities 7 and 8. Applying the mentioned cost function, $f^{(7)}$ and $f^{(8)}$ would incur 32 cost units and 22 cost units, respectively. Hence, the best choice was $f^{(8)}$.

The second experiment was to compare the time taken to find the optimum RM expansion. The number of minterms in a Boolean function affects the optimization time, so we carried out the comparison in two cases. In the first case, the number of minterms in a randomly generated Boolean function was set at 80% of all possible numbers of minterms. Table 4 lists the comparison results. In the second case, the number of minterms was set at 20% of all possible numbers of

**Table 4.** Comparison of algorithms: CPU time in sec. (80% minterms)

| $n$ | Tabular method [4] | Algo. from [24] | Algo. from [37] | FRMT* [40] | 2nd algo. from [40] | Proposed method |
|---|---|---|---|---|---|---|
| 15 | 3870.63 | 1414.60 | 1368.98 | 1436.18 | 1292.16 | 1309.56 |
| 16 | 19965.19 | 5263.94 | 5319.79 | 5224.56 | 4985.64 | 5089.60 |
| 17 | 87593.81 | 25210.64 | 19645.56 | 19877.61 | 18935.73 | 19060.92 |
| 18 | 384926.35 | 88732.05 | 72256.94 | 76133.58 | 72018.85 | 72301.76 |
| 19 | 1746064.86 | 394666.23 | 273264.61 | 283992.34 | 271326.22 | 27106.61 |
| 20 | 5472792.91 | 1738318.24 | 1302215.01 | 1134070.11 | 112849.84 | 1129165.25 |

(*FRMT = Fast Reed-Muller transform.)

**Table 5.** Comparison of algorithms: CPU time in sec. (20% minterms)

| $n$ | Tabular method [4] | Algo. from [24] | Algo. from [37] | FRMT* [40] | 2nd algo. from [40] | Proposed method |
|---|---|---|---|---|---|---|
| 15 | 2703.24 | 983.82 | 893.72 | 807.39 | 788.03 | 790.08 |
| 16 | 9871.47 | 3908.21 | 3862.65 | 3874.53 | 3698.65 | 3716.97 |
| 17 | 37712.03 | 13129.57 | 12152.98 | 12207.02 | 12070.29 | 12014.46 |
| 18 | 124990.59 | 45506.29 | 43557.57 | 43672.24 | 42942.59 | 43644.62 |
| 19 | 788671.99 | 155095.75 | 153028.63 | 152270.98 | 131882.80 | 131078.66 |
| 20 | 2162801.76 | 505351.84 | 468345.46 | 472309.69 | 471615.12 | 472032.77 |

minterms, with other aspects of the experiment being the same as the first case. The comparison results are given in Table 5. It can be observed from Tables 4 and 5 that the method presented in this paper generally performs more efficiently than the algorithms in [4], [24], [37], and the fast Reed-Muller transform (FRMT) described in [40]. However, the proposed method is not as time efficient as the second algorithm in [40] because the latter has extra steps for adjacent polarity mapping which may speed up execution. However, it should be noted that this speedup execution is achieved at the expense of a more complicated algorithm, and in addition, the second algorithm in [40] has a memory requirement of at least $O(2^n)$, whereas our algorithm requires storage of at most $O(n^2)$.

## 6. Conclusions

We have shown how to derive canonical RM expansions of all polarities from a known fixed-polarity RM expansion. As a result, a simple algorithm has been developed to facilitate efficient derivation of RM expansions based on a single-bit difference in polarity. To avoid repetitions of polarities and to provide a proper stopping criterion for the algorithm, we have formulated all polarities as nodes of a hypercube and used a recursive route to traverse all nodes without repetitions. An exhaustive-search optimization method has thus been proposed, and test results have shown that it generally works as efficiently as, if not better than, other

good exhaustive-search methods of Boolean-to-RM conversions. Because of the simplicity of the proposed algorithm, it can also be implemented more readily.

## References


[1] S. B. Akers, On a theory of Boolean functions, *J. SIAM*, vol. 7, 487–498, 1959.

[2] A. E. A. Almaini, *Electronic Logic Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[3] A. E. A. Almaini and L. McKenzie, Tabular techniques for generating Kronecker expansions, *Proc. IEE—Computer Digital Techs.*, vol. 143, 205–212, 1996.

[4] A. E. A. Almaini, P. Thomson, and D. Hanson, Tabular techniques for Reed-Muller logic, *Internat. J. Electron.*, vol. 70, 23–34, 1991.

[5] P. W. Besslich, Efficient computer method for EX-OR logic design, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 130, 203–206, 1983.

[6] R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Computers*, vol. 35, 677–691, 1986.

[7] S. Chattopadhyay, S. Roy, and P. P. Chaudhuri, Synthesis of highly testable fixed-polarity AND-XOR canonical networks-A genetic algorithm-based approach, *IEEE Trans. Computers*, vol. 45, 487–490, 1996.

[8] D. Debnath and T. Sasao, GRMIN2: A heuristic simplification algorithm for generalised Reed-Muller expressions, *Proc. IEE—Computer Digital Techs.*, vol. 143, 376–384, 1996.

[9] R. Drechsler and B. Becker, Relation between OFDDs and FPRMs, *Electron. Lett.*, vol. 32, 1975–1976, 1996.

[10] R. Drechsler, M. Theobald, and B. Becker, Fast OFDD-based minimisation of fixed polarity Reed-Muller expressions, *IEEE Trans. Computers*, vol. 45, 1294–1299, 1996.

[11] B. Fei, Q. Hong, H. Wu, M. A. Perkowski, and N. Zhuang, Efficient computation for ternary Reed-Muller expansions under fixed polarities, *Internat. J. Electron.*, vol. 75, 685–688, 1993.

[12] D. H. Green, *Modern logic design*, Addison-Wesley, Wokingham, U.K., 1986.

[13] D. H. Green, Reed-Muller expansions of incompletely specified functions, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 134, 228–236, 1987.

[14] D. H. Green, Simplification of switching functions using variable-entered maps, *Internat. J. Electron.*, vol. 75, 877–886, 1993.

[15] D. H. Green, Dual forms of Reed-Muller expansions, *Proc. IEE—Computer Digital Techs.*, vol. 141, 184–192, 1994.

[16] B. Harking, Efficient algorithm for canonical Reed-Muller expansions of Boolean functions, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 137, 366–370, 1990.

[17] T. L. Lin and A. Tran, Minimization of multiple-output exclusive-OR switching functions, *Internat. J. Electron.*, vol. 75, 665–674, 1993.

[18] T. C. Lin, A. Tran, and C. Ouyang, Reed-Muller universal logic module and its application to function realization, *Internat. J. Electron.*, vol. 75, 675–684, 1993.

[19] P. K. Lui and J. C. Muzio, Boolean matrix transforms for the parity spectrum and minimisation of modulo-2 canonical expansions, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 138, 411–417, 1991.

[20] L. McKenzie, A. E. A. Almaini, J. F. Miller, and P. Thomson, Optimisation of Reed-Muller logic functions, *Internat. J. Electron.*, vol. 75, 451–466, 1993.

[21] L. McKenzie and A. E. A. Almaini, Generating Kronecker expansions from reduced Boolean forms using tabular methods, *Internat. J. Electron.*, vol. 82, 313–325, 1997.

[22] H. C. Michael, *Parallel programming—A new approach*, Silicon Press, Summit, NJ, 1992.

[23] J. F. Miller, H. Luchian, P. V. G. Bradbeer, and P. J. Barclay, Using a genetic algorithm for optimizing fixed polarity Reed-Muller expansions of Boolean functions, *Internat. J. Electron.*, vol. 76, 601–609, 1994.

[24] J. F. Miller and P. Thomson, Highly efficient exhaustive search algorithm for optimizing canonical Reed-Muller expansions of Boolean functions, *Internat. J. Electron.*, vol. 76, 37–56, 1994.

[25] S. Purwar, An efficient method of computing generalised Reed-Muller expansions from binary decision diagram, *IEEE Trans. Computers*, vol. 40, 1298–1301, 1991.

[26] S. M. Reddy, Easily testable realisations for logic functions, *IEEE Trans. Computers*, vol. 21, 1183–1188, 1972.

[27] G. I. Robertson, J. F. Miller, and P. Thomson, Non-exhaustive search methods and their use in the minimization of Reed-Muller canonical expansions, *Internat. J. Electron.*, vol. 80, 1–12, 1996.

[28] T. Sasao and P. Besslich, On the complexity of mod-2 sum PLAs, *IEEE Trans. Computers*, vol. 39, 262–266, 1990.

[29] T. Sasao and F. Izuhara, Exact minimization of AND-EXOR ternary decision diagrams, in *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 213–220, 1995.

[30] P. Thomson and J. F. Miller, Symbolic method for simplifying AND-EXOR representations of Boolean functions using a binary-decision technique and a genetic algorithm, *Proc. IEE—Computer Digital Techs.*, vol. 143, 151–155, 1996.

[31] M. A. Thornton and V. S. S. Nair, BDD-based spectral approach for Reed-Muller circuit realisation, *Proc. IEE—Computer Digital Techs.*, vol. 143, 145–150, 1996.

[32] A. Tran, Graphical method for the conversion of minterms to Reed-Muller coefficients and the minimisation of exclusive-OR switching functions, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 134, 93–99, 1987.

[33] A. Tran, Tri-state map for the minimisation of exclusive-OR switching functions, *Proc. IEE, Pt E, Computer Digital Techs.*, vol. 136, 16–21, 1989.

[34] C.-C. Tsai and M. Marek-Sadowska, Minimisation of fixed-polarity AND/XOR canonical networks, *Proc. IEE—Computer Digital Techs.*, vol. 141, 369–374, 1994.

[35] C.-C. Tsai and M. Marek-Sadowska, Boolean functions classification via fixed-polarity Reed-Muller forms, *IEEE Trans. Computers*, vol. 46, 173–186, 1997.

[36] B. Vinnakota and V. V. B. Rao, Generation of all Reed-Muller expansions of a switching function, *IEEE Trans. Computers*, vol. 43, 122–124, 1994.

[37] L. Wang, A. E. A. Almaini, and A. Bystrov, Efficient polarity conversion for large Boolean functions, *Proc. IEE—Computer Digital Techs.*, vol. 146, 197–204, 1999.

[38] X. Wu, A. E. A. Almaini, J. F. Miller, and L. McKenzie, Reed-Muller universal logic module networks, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 140, 105–108, 1993.

[39] X. Wu, X. Chen, and S. L. Hurst, Mapping of Reed-Muller coefficients and the minimisation of EX-OR switching functions, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 129, 15–20, 1982.

[40] Y. Z. Zhang and P. J. W. Rayner, Minimisation of Reed-Muller polynomials with fixed polarity, *Proc. IEE, Pt. E, Computer Digital Techs.*, vol. 131, 177–186, 1984.