Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Reed-Muller function optimization techniques with onset table*

Lun-yao WANG[†1,2], Yin-shui XIA[2], Xie-xiong CHEN[1], A. E. A. ALMAINI[3]
([1]Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China)
([2]Faculty of Information Science and Engineering, Ningbo University, Ningbo 315211, China)
([3]School of Engineering, Napier University, Edinburgh EH10 5DT, UK)
[†]E-mail: wanglunyao@nbu.edu.cn

Received June 12, 2010; Revision accepted Nov. 30, 2010; Crosschecked Mar. 1, 2011

Abstract: By mapping a fixed polarity Reed-Muller (RM) expression into an onset table and studying the properties of the onset table, an algorithm is proposed to obtain a compact multi-level single-output mixed-polarity RM function by searching for and extracting the common variables using the onset table. Furthermore, by employing the multiplexer model, the algorithm is extended to optimize multi-level multi-output mixed-polarity RM forms. The proposed algorithm is implemented in C language and tested using some MCNC benchmarks. Experimental results show that the proposed algorithm can obtain a more compact RM form than that under fixed polarity. Compared with published results, the proposed algorithm makes a significant speed improvement, with a small increase in the number of literals.

Key words: Logic optimization, Reed-Muller functions, Multi-level, Mixed polarity, Onset table
doi:10.1631/jzus.C1000193 **Document code:** A **CLC number:** TP391

1 Introduction

Switching functions can be represented as either AND/OR/NOT based traditional Boolean forms or AND/XOR based Reed-Muller (RM) forms. Methods for RM logic minimization are still attracting researchers' attention (Gupta *et al*., 2006; Balasubramanian *et al*., 2007; Pradhan and Chattopadhyay, 2008; Jankovic *et al*., 2009). Reasons for this include the fact that the logic functions that cannot be minimized well in sum of products (SOP) forms can often be implemented in the RM domain with fewer product terms, leading to reduced size and power consumption.

The minimization of switching functions is one of the most important goals in logic optimization. In general, the goal of minimization is to reduce the number of products and make each product term as simple as possible, namely reducing the literals of the function. Compared to the fixed polarity RM (FPRM) forms, the mixed polarity RM (MPRM) forms are usually more compact. The MPRM expansions have also been found to be a useful tool in Boolean function classification (Tsai and Marek-Sadowska, 1997).

There are several interesting techniques and algorithms for MPRM optimization in the public domain. Green (1990) described the set of 3^n consistent MPRM canonical forms of an n-variable switching function and investigated the structure of various fixed and mixed polarity transforms. Tran and Lee (1993) used a tri-state map to represent a function in a predefined polarization state, and to minimize the function as an RM polynomial in mixed polarity. They then developed a minimization method for RM polynomials in mixed polarity using a decomposition method (Tran and Wang, 1993). By implementing the tabular technique (Almaini *et al*., 1991) and genetic algorithm (GA), Al Jassani *et al*. (2008) proposed a different approach to finding the optimal mixed polarity RM form among 3^n different polarities for large functions. Based on the concept of 3/4-majority cube

* Project supported by the National Natural Science Foundation of China (Nos. 60871022 and 61041001), the Natural Science Foundation of Zhejiang Province (Nos. Z1090622 and Y1080654), and the Ningbo Natural Science Foundation, China (No. 2010A610183)

(Tran and Wang, 1993) and b_j-map, Wang and Almaini (2002) proposed an algorithm which can produce a simplified two-level mixed polarity RM format from the conventional SOP format.

To obtain a more compact RM expression, Xia *et al*. (2005) proposed a truth vector based method for multi-level mixed polarity RM (MMPRM) optimization. The method uses a truth vector with length 2^n to represent an *n*-variable FPRM expansion. By elimination and decomposition of the truth vector, the compact representation of MMPRM forms can be obtained. Wang *et al*. (2006) proposed another approach to obtaining MMPRM by employing an onset table. Unlike the method based on the truth vector which stores all 2^n elements, the onset table just 'records' the positions of those elements whose values are '1'. Therefore, less memory is required in the onset table based algorithm. However, the algorithm (Wang *et al*., 2006) is limited to single-output MMPRM function optimization. Furthermore, the technique of searching for and extracting the common variables, especially the common sub-tables, is complicated, which makes the algorithm work slowly. In this paper, the technique of the common variables searching and extraction is developed. By computing and analyzing the frequency of each variable appearing in the onset table, the common variables are identified and extracted. A multiplexer model is further used for multi-output RM function optimization.

2 Properties of the onset table

A two-level FPRM expansion can be represented as

$$f(\dot{x}_{n-1}\cdots\dot{x}_1\dot{x}_0)=\oplus\sum_{i=0}^{2^n-1}b_i\pi_i. \tag{1}$$

Here $\oplus\sum$ is the XOR sum of products. The sub-index i is in binary form as $i=(i_{n-1}...i_1i_0)_2$, and the products π_i can be expressed as $\pi_i=\prod_{j=0}^{n-1}\dot{x}_j^{\,i_j}$, where

$$\dot{x}_j^{\,i_j}=\begin{cases}1, & i_j=0,\\ \dot{x}_j, & i_j=1,\end{cases}\quad j\in\{0,1,\cdots,n-1\}. \tag{2}$$

In the FPRM, $\dot{x}$ can take either x or its complement, but not both.

Definition 1 Any *n*-variable FPRM function can be expressed with a set ON, which is composed of the decimal equivalent of the coefficients of π terms. The set ON is also called onset.

For example, given a four-variable function $f(\dot{x}_3\dot{x}_2\dot{x}_1\dot{x}_0)=\dot{x}_0\oplus\dot{x}_3\dot{x}_1\dot{x}_0\oplus\dot{x}_3\dot{x}_2\dot{x}_0$ under any fixed polarity, if $\pi_1=\pi_{(0001)_2}=\dot{x}_3^0\dot{x}_2^0\dot{x}_1^0\dot{x}_0^1=\dot{x}_0$, $\pi_{11}=\pi_{(1011)_2}=\dot{x}_3^1\dot{x}_2^0\dot{x}_1^1\dot{x}_0^1=\dot{x}_3\dot{x}_1\dot{x}_0$, then the function could be represented by π-terms as $f(\dot{x}_3\dot{x}_2\dot{x}_1\dot{x}_0)=\pi_1\oplus\pi_{11}\oplus\pi_{13}$. It can also be expressed by an ON set as ON={1, 11, 13}, in which the decimal is the sub-index of π. But there exists a special π term in which none of the variables appears. Therefore, all digits of the term are 0's. The sub-index of the π term is 0, namely π_0. Considering π_0 including no variables, a constant '0' or '1' can be expressed as π_0. In this paper, π_0 is used to express a constant '1' because what we consider here is the 'onset', and any π term that equals a constant '0' will be removed because $0\oplus\pi_i=\pi_i$.

Definition 2 The onset table, also called the *T* table (briefly *T*), is to describe the presence of each variable in each π term of an RM function. The onset table consists of the elements of set ON in binary form.

The structure of *T* is as follows: the decimal corresponding to the binary digit in each row of *T* represents each element of the set ON; each column of *T* represents an input variable of the RM function; $k_{ij}\in\{0, 1\}$ is a binary digit on the *i*th row and *j*th column of *T*. k_{ij}=1 means that the variable $\dot{x}_j$ on the *j*th column appears in term π_i while k_{ij}=0 means it does not. Fig. 1a is the onset table *T* of the function $f(\dot{x}_3\dot{x}_2\dot{x}_1\dot{x}_0)=\pi_1\oplus\pi_{11}\oplus\pi_{13}$.

In this paper, the symbol '→' is introduced to describe the conversion between *f* and its onset table *T*. The conversion *f*→*T* means a mapping from *f* to *T*, and *T*→*f* is the inverse. From Definition 2, we can see that a row in *T* can be mapped to a product term in *f*; therefore, the logic 'XOR' can be implemented between rows. Similarly, the logic 'AND' can be implemented between columns. Considering the commutativity of the 'AND' and 'XOR' operations, it is possible to obtain Lemma 1 as follows. In the following, '•' is used to denote the 'AND' operation.

Lemma 1 If an onset table *T*′ is generated by exchanging any two rows or any two columns of *T*, *T*→*f*, *T*′→*f*′, then *f*=*f*′.

Definition 3 (Sub-tables of T) An onset table T can be divided into m sub-tables, ST_i, $i\in\{1, 2, ..., m\}$, $m\in\mathbb{N}$, in either horizontal or vertical direction. Each ST_i covers some rows (π terms) or columns (variables) of T.

Figs. 1b and 1c show the sub-tables, ST_1 and ST_2, resulting from the vertical division and horizontal division of T, respectively.

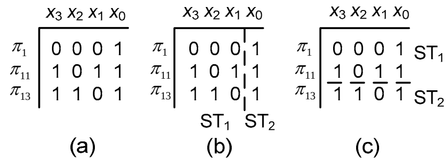

Fig. 1 Onset table and its division

(a) Onset table T; (b) Division in the vertical direction; (c) Division in the horizontal direction

Definition 4 (Combination operators 'Θ' and '^' of sub-tables) If an onset table T is divided into two or more sub-tables, ST_i, $i\in\{1, 2, ..., m\}$, $m\in\mathbb{N}$, then these sub-tables can be combined together to cover T again. The operator 'Θ' denotes the combination of two sub-tables ST_1 and ST_2 into a new T in the horizontal direction that includes every column of ST_1 and ST_2. The operator '^' denotes the combination of two sub-tables ST_1 and ST_2 into a new T in the vertical direction that includes every row of ST_1 and ST_2.

Based on Definition 4, the T in Figs. 1b and 1c can be expressed as $T=\mathrm{ST}_1\Theta\mathrm{ST}_2$ and $T=\mathrm{ST}_1$^ST_2, respectively.

Lemma 2 Given an onset table T, if T can be divided into several sub-tables $\{\mathrm{ST}_1, \mathrm{ST}_2, \ldots, \mathrm{ST}_m\}$ in the horizontal direction, then $f = f_1 \oplus f_2 \oplus ... \oplus f_m$, where $T\to f$, $\mathrm{ST}_i\to f_i$, $1\le i\le m$.

Proof Let $T\to f$, $f=\oplus\sum_{i=0}^{k-1}\pi_i$, where $k\ge m$, m is the number of sub-tables of T, and k is the number of π_i in T. From Definition 2, it is clear that a row in T is equal to a π_i term in f. Hence, if a T is divided into some sub-tables $\{\mathrm{ST}_1, \mathrm{ST}_2, \ldots, \mathrm{ST}_m\}$ in the horizontal direction, then each sub-table covers at least one row of T. Let $\mathrm{ST}_i\to f_i$, $\mathrm{ST}_i=\{\pi_1, \pi_2, ..., \pi_{u-1}\}$, $l<u$. Then $f_i = \pi_l \oplus \pi_{l+1} \oplus \cdots \oplus \pi_{u-1}$, and

$$f=\oplus\sum_{i=0}^{k-1}\pi_i=\left(\oplus\sum_{i=0}^{h-1}\pi_i\right)\oplus\cdots\oplus\left(\oplus\sum_{i=l}^{u-1}\pi_i\right)\oplus\cdots\oplus\left(\oplus\sum_{i=j}^{m-1}\pi_i\right)$$
$$=f_1\oplus\cdots\oplus f_i\oplus\cdots\oplus f_m,$$

where $0<h<l<u<j<m$.

From Lemma 2 and Fig. 1c, it can be seen that if $\mathrm{ST}_1\to f_1$, $\mathrm{ST}_2\to f_2$, then $(T=\mathrm{ST}_1$^$\mathrm{ST}_2)\to(f=f_1\oplus f_2)$.

Lemma 3 Given an onset table T with n variables, if T can be divided into m sub-tables $\{\mathrm{ST}_1, \mathrm{ST}_2, \ldots, \mathrm{ST}_m\}$ in the vertical direction, and all elements in a sub-table ST_i that cover the variables $\{x_i, x_{i+1}, \ldots, x_{i+k-1}\}$ are 1's, then the RM function f can be expressed as

$$f=\left(\prod_{j=0}^{k-1}x_{i+j}\right)\bullet f_{\mathrm{r}},$$

where $\mathrm{ST}_{\mathrm{r}}\to f_{\mathrm{r}}$, $\mathrm{ST}_{\mathrm{r}}=\{\mathrm{ST}_{x_1},\cdots,\mathrm{ST}_{x_{i-1}},\mathrm{ST}_{x_{i+k}},\cdots,\mathrm{ST}_{x_n}\}$. Here, ST_{x_i} stands for a sub-table that contains column x_i only.

Proof Let $T\to f$, $f=\oplus\sum_{i=0}^{u-1}\pi_i$, where n is the number of variables of T, and u is the number of π terms. Based on the definition of onset table T, a row in T is equal to a π term in f. Therefore, when a sub-table ST_i is generated by dividing a T in the vertical direction whose elements are all 1's, the variables $\{x_i, x_{i+1}, \ldots, x_{i+k-1}\}$ covered by ST_i exist in every π term of f. Then every term π_i can be expressed as $\pi_i=(x_ix_{i+1}\cdots x_{i+k})\bullet\pi_i'$, where π_i' is a remainder after the factor $\{x_i, x_{i+1}, \ldots, x_{i+k-1}\}$ has been extracted from π_i. Therefore, f can be expressed as

$$f=\oplus\sum_{i=0}^{m}\pi_i=(x_ix_{i+1}\cdots x_{i+k-1})\bullet\left(\oplus\sum_{\mathrm{i}=0}^{m}\pi_i'\right)$$
$$=(x_ix_{i+1}\cdots x_{i+k-1})\bullet f_{\mathrm{r}},$$

where $f_{\mathrm{r}}=\oplus\sum_{i=0}^{m}\pi'$.

Note that a '0' in T means that the corresponding variable does not exist in a π term. Hence, there are two ways to express the sub-table ST_{r}, which is a remainder when a sub-table ST_i is removed from T. One is to remove ST_i from T directly and the remainder is ST_{r}. That is, $\mathrm{ST}_{\mathrm{r}}=\{\mathrm{ST}_1, ..., \mathrm{ST}_{i-1}, \mathrm{ST}_{i+1}, ..., \mathrm{ST}_m\}$. The other is to replace the ST_i with 0's in T. That is, $\mathrm{ST}_{\mathrm{r}}=\{\mathrm{ST}_1, ..., \mathrm{ST}_{i-1}, 0, \mathrm{ST}_{i+1}, ..., \mathrm{ST}_n\}$. The two methods are equivalent. In this study, the second method is adopted to express the sub-table ST_{r}. For example, in Fig. 1b all the elements of ST_2 are 1's, which can be extracted from T. We have $(T=\mathrm{ST}_2\Theta\mathrm{ST}_1)\to f=x_0\bullet f_1$, where $\{\mathrm{ST}_1, 0\}\to f$.

3 Extraction of common variables

In general, the goal of minimization of a switching function is to reduce the numbers of products and literals in each product term. For the NOT/AND/OR based switching function, there exists 1+1=1. Here the symbol '+' means the 'OR' operator. Therefore, we can minimize the numbers of the literals and product terms by expanding the cover of each product to be as large as possible and removing the redundant covers. However, for the AND/XOR based RM logic, the overlap of different covers may change the correctness of the function, because for the 'XOR' operation there exists $1\oplus1=0$. This implies that the techniques suitable for NOT/AND/OR based switching function minimization will be limited in RM logic minimization.

There are some ways to minimize the RM logic. The techniques for polarity conversion are usually used in two-level RM logic minimization. In multilevel RM logic minimization, the common variables searching and extraction is an important technique. There are two kinds of common variable in FPRM. One is a global common variable, which appears in each product term of the RM function, and the other is the local common variable, which appears only in some product terms of the RM function. In most cases, however, the common variables are referred to the local ones because the global common variables are usually unavailable. Therefore, in this study we focus on the local common variables searching and extraction. The identification of a common variable is based on the fact that, if a variable appears in many product terms (or π terms) of an RM function, namely the appearance frequency (frequency for short) of a variable is high, it is suitable for being extracted as a common variable. The algorithm of the common variables searching and extraction is based on the method we proposed in Wang and Xia (2008), and the main steps are shown below.

Step 1: The variables' frequencies computed in the onset table *T* using the operator frequency(*T*).

The process is as illustrated in Fig. 2a. For example, there are two 1's in the column of the variable x_3 in the onset table. Therefore, the frequency of x_3 is 2. The frequency of a variable equal to the number of π terms means that, the variable appears in every term of the *T* and can be extracted as a global common variable.

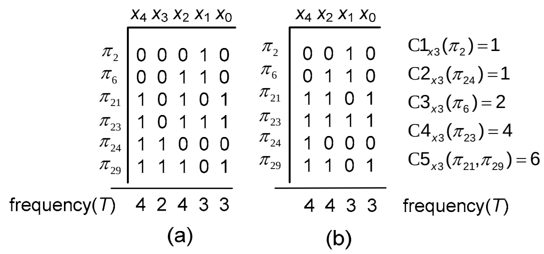

Fig. 2 frequency(*T*) (a) and the weights of the classes when x_3 is removed (b)

Step 2: The classification of π terms using the operator classification(*T*).

First, the frequency of each variable is sorted in increasing order. Then *k* variables that have lower frequencies will be removed from *T*. Here, *k* is the floor of $n/2$, $k=\lfloor n/2 \rfloor$, and *n* is the number of variables. After some variables are removed, the value of each π term in onset table *T* will be updated. According to the values, the updated π terms will be classified. Those π terms with the same values will be put into the same class.

The process of classification is shown in Fig. 2b after x_3 is removed. In Fig. 2b, the expression $C5_{x3}(\pi_{21}, \pi_{29})=6$ means when x_3 is removed, π_{21} and π_{29} have the same value. Therefore, π_{21} and π_{29} can be put into the same class named $C5_{x3}(\pi_{21}, \pi_{29})$ (C5 for short). The result of the expression is 6, which means there are six 1's in C5. The result is also called the weight of C5.

Because there are five variables in the onset table *T* in Fig. 2a, the two variables with lower frequency can be removed. According to the frequency(*T*), two kinds of combinations, (x_3, x_1) and (x_3, x_0), whose frequencies are lower, can be removed. The corresponding weights of the classes will be obtained respectively. Comparing their weights, class $C_{x3,1}(\pi_{21}, \pi_{23}, \pi_{29})$ (obtained by removing x_3 and x_1 from *T*) is the best. To accelerate the evaluation, if many variables have the same frequency, the algorithm will select arbitrarily one or more variables to be removed.

In our algorithm, a variable is removed by bitwise ANDing every π term in *T* with a decimal number named filter, and the filter is created based on frequency(*T*). For example, if we want to remove x_3, we just clear the bit corresponding to x_3 and set other bits in the filter. That is,

$$\text{filter}=(x_4x_3x_2x_1x_0)_2=(10111)_2=23_{10}. \quad (3)$$

Then bitwise ANDing every π term in T with the filter being 23, gives T':

$$T'=\text{filter}\ \&\ T=23\ \&\ \{2, 6, 21, 23, 24, 29\} \\ =\{2, 6, 21, 23, 16, 21\}. \quad (4)$$

Based on Eq. (4), T' can be classified based on the values. This example illustrates that the π terms can be stored as decimal numbers in an array. Unlike the operation in Wang *et al*. (2006), our operation of removing variables can be realized by operating on the corresponding bit of the decimal number directly, which avoids the conversion between binary and decimal and makes the algorithm work fast and need less memory.

To make the algorithm work fast, the number of removed variables, k, will be changed when one of the following conditions is met. Suppose the number of input variables is n.

Condition 1 Suppose S_{FQ} is the total sum of the frequencies, and the variable x_i has the maximum frequency, FQ_i, in frequency(T). If there exists $FQ_i > S_{FQ} - FQ_i$, then the variable x_i will be kept and the other $n-1$ variables are removed, namely $k=n-1$.

Condition 2 If there are m 0's in frequency(T), $m<n$, then $k = m + \lfloor (n-m)/2 \rfloor$. This constraint will stop the algorithm from running into a dead end when $m \geq \lfloor n/2 \rfloor$.

Step 3: The onset table division and common variables extraction using Div_extr(T) operators.

T is divided into two sub-tables, ST_1 and ST_r, in the horizontal direction first. ST_1 contains only the class whose weight is the heaviest, and $ST_r=T-ST_1$. For example, in Fig. 2a, when two variables are removed from T, the weight of class $C_{x3,1}(\pi_{21}, \pi_{23}, \pi_{29})$ is the heaviest, equal to 9. Hence, the onset table T can be divided into $ST_1=\{\pi_{21}, \pi_{23}, \pi_{29}\}$ and $ST_r=\{\pi_2, \pi_6, \pi_{24}\}$. Fig. 3 shows the result of division by Div_extr(T). Namely, $T=ST_r \wedge ST_1$. Let $T \rightarrow f$, $ST_r \rightarrow f_r$, $ST_1 \rightarrow f_1$, $ST_1' \rightarrow f_1'$. Then $(T=ST_r \wedge ST_1=ST_r \wedge (ST_{x1} \Theta ST_{x3} \Theta ST_{x5} \Theta ST_1)) \rightarrow (f = f_r \oplus f_1 = f_r \oplus ((x_4x_2x_0) \bullet f_1'))$.

After division, the local common variables of ST_1 will be found and extracted. In this example (Fig. 3), the local common variables are x_4, x_2, and x_0.

Step 4: Deleting the π_0 term of the onset table using the Deletterm0(T) operator.

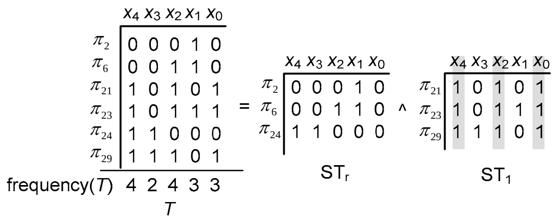

Fig. 3 *T* and its sub-tables after Div_extr(*T*)

After the extraction of the common variables, a row in the onset table may contain 0's only; namely, the onset table contains a π_0 term, which means a constant '1'. The row can then be deleted after the result is stored. For example, in Fig. 3, when variables (x_4, x_2, x_0) are removed from ST_1, the first row of ST_1 contains 0's only and can be deleted.

Step 5: Store the sub-table ST_r, and let ST_1 be T. Then repeat Steps 1–4 to search for other common variables. Sub-tables ST_1 and ST_r can be further divided by repeating the above steps. Figs. 4 and 5 show the results of the division.

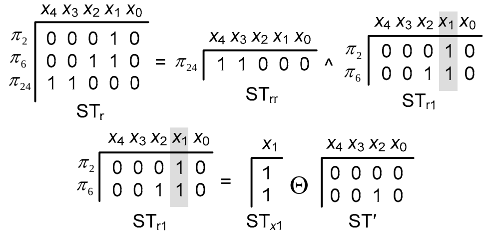

Fig. 4 Further division of ST_r and ST_{r1}

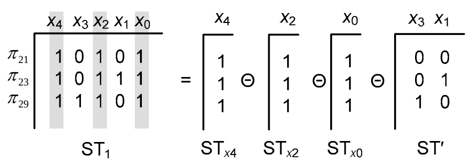

Fig. 5 Further division of ST_1

Therefore, if $T \rightarrow f$, then from Figs. 3–5, a mixed polarity multi-level function f can be obtained:

$$\begin{aligned} f &= (x_4x_3 \oplus x_1(1 \oplus x_2)) \oplus (x_4x_2x_0(1 \oplus x_3 \oplus x_1)) \\ &= x_4x_3 \oplus x_1\overline{x_2} \oplus x_4x_2x_0(\overline{x_3} \oplus x_1). \end{aligned} \quad (5)$$

Considering the commutativity of the '$\oplus$' operations, the sub-expression $(1 \oplus x_3 \oplus x_4)$ in Eq. (5) can be written as

$$(1\oplus x_3 \oplus x_4) = \{x_3 \oplus (1\oplus x_4)\} = (x_3 \oplus \overline{x}_4). \quad (6)$$

In our algorithm, the order of the '⊕' operation between the variables is random, which means that the method cannot lead to a unique RM expression under mixed polarity.

4 Multi-output Reed-Muller function optimization

In Section 3, only single output RM functions are considered, but in real designs, most circuits have many outputs. Therefore, it is essential to develop a method for multi-output RM function optimization. To solve this problem, the model of multiplexer is used. Eq. (7) is the logic function of a multiplexer:

$$f = a_0 i_0 + a_1 i_1 + \cdots + a_{n-1} i_{n-1}, \quad (7)$$

where the variables $\{i_0, i_1, \ldots, i_{n-1}\}$ are the data inputs. The variables $\{a_0, a_1, \ldots, a_{n-1}\}$ are the addresses. The output is f. The a_i can be encoded as a_i=(add$_0$, add$_1$, add$_2$, …, add$_{m-1}$), add$_j \in \{0, 1\}$, $m \geq \log_2 n$.

Considering that for any two address variables a_i and a_j ($i \neq j$), $a_i \bullet a_j = 0$, Eq. (7) can be expressed as

$$f = (a_0 i_0) \oplus (a_1 i_1) \oplus \cdots \oplus (a_{n-1} i_{n-1}). \quad (8)$$

Now if we treat the data inputs in Eq. (8) as the outputs of the multi-output RM function and keep the addresses $\{a_0, a_1, \ldots, a_{n-1}\}$ unchanged during optimization, we can then use the single output RM function optimization algorithm to optimize multi-output RM functions.

For example, an RM function f with two outputs can be described as

$$f_0 = (x_2 x_1) \oplus (x_3 x_0), \;\; f_1 = (x_2 x_1) \oplus (x_3 x_2 x_0). \quad (9)$$

There are two outputs in Eq. (9). Hence, one bit address variable is needed. By using Eq. (8), the multi-output RM function f can be expressed as

$$f = (a_0 f_0) \oplus (a_1 f_1). \quad (10)$$

The onset table of Eq. (10) is shown in Fig. 6. The left part is the onset table of the function $f(f_0, f_1)$. The variable a in T is the address variable, and a_0=0, a_1=1. During optimization, the address variables in T are kept unchanged. Therefore, it is not necessary for their frequencies to be computed by frequency(T).

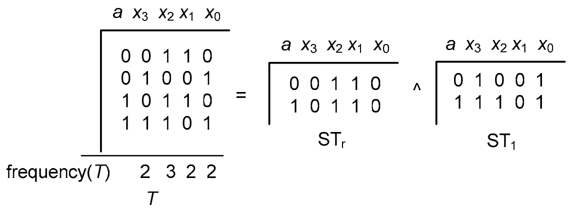

Fig. 6 Onset table T with address variables and its division

The right part of Fig. 6 shows the division of T. The sub-tables ST$_r$ and ST$_1$ can be expressed as $(a_0 \oplus a_1)x_2x_1$ and $(a_0 \oplus a_1 x_2)x_3x_0$, respectively. The multi-output RM function f can then be expressed as

$$f = (a_0 f_0) \oplus (a_1 f_1) = (a_0 \oplus a_1 x_2) x_3 x_0 \oplus (a_0 \oplus a_1) x_2 x_1. \quad (11)$$

It is clear from Eq. (11) that x_3x_0 and x_2x_1 are common factors of f_0 and f_1, respectively. Now let $y_1 = x_3x_0$, $y_2 = x_2x_1$. Then Eq. (9) can be expressed as

$$f_0 = y_1 \oplus y_2, \;\; f_1 = (x_2 y_1) \oplus y_2. \quad (12)$$

Fig. 7 is the logic diagram of Eq. (12).

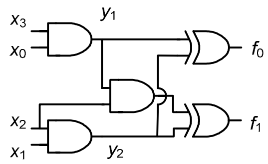

Fig. 7 The logic diagram of $f_0 = y_1 \oplus y_2$, $f_1 = (x_2 y_1) \oplus y_2$

To make the algorithm work effectively, three criteria are introduced. When the onset table T meets one of these criteria, the T will no longer be divided.

Criterion 1 The RM function contains only one π term; that is, there is only one row in the onset table. The algorithm will stop and output the results directly. Under this condition, there is no common variable to be extracted, and the function is in its compact form.

Criterion 2 The RM function contains some π terms, but the variables in each π term are different, such as $f = ac \oplus b \oplus de$. Under this condition, there are no common variables to be extracted, and the function is in its compact form.

Criterion 3 All elements (except the address variable) are 0's. Under this condition, the function is a constant.

When these three stop criteria are described by an onset table, it will be discovered that no variable frequency in frequency(T) is greater than 1. Therefore, by checking the result of frequency(T), we can decide whether to divide the onset table or not. The corresponding operation to check stop criteria is called rulescheck(T).

The pseudo code in Fig. 8 illustrates how the algorithm optimizes a multi-output RM function under a fixed polarity to obtain its corresponding compact form under mixed polarity.

```
file convert and generate T;
store T into a List;
copylist=0;
rulescheck(T);
if (meeting the criteria)
  output results;
else
{
  while(!(List==NULL &&
    copylist==0))
  {
   if (copylist)
      store T=STr in List;
   else
   {
    comn_var_extr(T);
    Deleterm0(T);
    rulescheck(T);
    if (meeting the critera)
     {
      copylist=1;
      store the result of T;
     }
    else
     {
     frequency(T);
     classification(T);
     {STr, ST1}=Div_extr(T);
      rulescheck(STr);
      if (meeting the critera)
       {
        store the results of STr;
        free(STr);
       }
      else
          put STr into List;
          rulescheck(ST1);
          if (meeting the critera)
           {
            store the result of ST1;
            free(ST1);
            copylist=1;
           }
          else
             T=ST1;
        }
      }
    }
  }
output the results;
```

Fig. 8 Pseudo code of the algorithm to obtain a multi-output multi-level mixed polarity Reed-Muller (MMPRM) form from a fixed polarity RM (FPRM) form

5 Experimental results

The proposed algorithm was implemented in C language and applied to MCNC benchmarks and some randomly generated circuits. The results were obtained using a PC with Intel 2.4 GHz CPU and 1 GB memory under Windows XP. Table 1 shows the results of the single output benchmarks, while Tables 2 and 3 present the results of the multi-output benchmarks. The number of literals is used to estimate the performance of the proposed algorithm. In Table 1, the results from Wang *et al*. (2006)'s method and the proposed algorithm under polarity 0 are both presented. The literals improvement is defined as Imp_1 in Table 1:

$$\mathrm{Imp}_1=\frac{\text{literals\_other}-\text{literals\_proposed}}{\text{literals\_other}}\times 100\%, \quad (13)$$

where literals_proposed is the number of the literals obtained using our proposed algorithm, and literals_other means the published results. Compared with the method in Wang *et al.* (2006), there is on average 8.6% literals increase using the proposed algorithm. Compared with the number of literals under polarity 0, the average literal saving of the proposed algorithm is great, up to 54.1%.

The speed improvement, Imp_2, is defined as

$$\mathrm{Imp}_2=\frac{\text{CPU\_time\_other}-\text{CPU\_time\_proposed}}{\text{CPU\_time\_other}}\times 100\%, \quad (14)$$

where CPU_time_other is the time needed by Wang *et al.* (2006)'s method, and CPU_time_proposed is the

Table 1 Experimental comparison for single output circuits

Circuit	Number of I/Os	Number of literals			CPU time (s)		Imp_2 (%)
		Proposed	Under polarity 0**	Wang *et al.* (2006)**	Wang *et al.* (2006)	Proposed	
9sym	9/1	291	756 (61.5)	304 (4.3)	0.11	<0.001	99.1
Sym10	10/1	528	1300 (59.4)	528 (0)	0.23	0.02	91.3
T481	16/1	56	108 (48.1)	55 (−1.8)	0.03	<0.001	>96.7
Newtag	8/1	43	88 (51.1)	27 (−59.2)	0.02	<0.001	>95.0
Newwill	8/1	79	237 (66.7)	70 (−12.9)	0.03	<0.001	>96.7
Life	9/1	343	792 (56.7)	321 (−6.8)	0.11	<0.001	>99.1
T1*	15/1	2355	4677 (49.6)	2477 (4.9)	1.88	0.20	89.4
T2*	19/1	8145	21503 (62.1)	8097 (−0.6)	13.83	6.47	53.2
T3*	28/1	17577	25682 (31.6)	16712 (−5.2)	71.80	56.72	21.0
Avg. imp. (%)			54.1	−8.6			>82.4

* Randomly generated benchmarks under polarity 0. ** Numbers in brackets represent the literals improvement, Imp_1 (%), as defined in Eq. (13)

time needed for our proposed algorithm. The proposed algorithm shows a significant speed improvement. Compared with Wang *et al.* (2006)'s method, when there are fewer than 15 input variables, the proposed algorithm can save approximately 95% CPU time, and the average CPU time saving is up to 82%.

Table 1 shows that the proposed algorithm makes a significant speed improvement, with a small increase in the number of literals for most cases. For some cases such as 'Newtag', however, Wang *et al.* (2006)'s method obtains more compact results but takes much more time. This is caused by the way of the common sub-table searching and extraction. Take Fig. 9 for example.

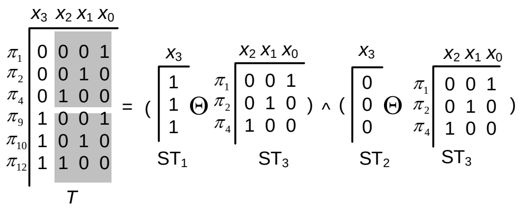

Fig. 9 Common sub-tables and its extraction

In the left part of the equation in Fig. 9, the two shadow blocks in table T are the same. Hence, the T can be split as the right part of the equation in Fig. 9. Therefore, ST_3 is the common sub-table and can be extracted. The final expression of T can be written as

$$f=(x_3\oplus 1)(x_2\oplus x_1\oplus x_0)=\overline{x_3}(x_2\oplus x_1\oplus x_0). \quad (15)$$

But if the T is split based on common variables, the final expression can be expressed as

$$f=x_3(x_2\oplus x_1\oplus x_0)\oplus x_2\oplus x_1\oplus x_0. \quad (16)$$

Compared with Eq. (15), there are more literals in Eq. (16).

For the specific benchmark 'Newtag', it leaves common sub-tables under polarity 0, which is very beneficial for the function expression compacting by common sub-table extraction. This explains why our proposed method needs 1.6 times as many literals as Wang *et al.* (2006)'s method. Conversely, the common sub-tables searching takes much CPU time and it is one of the reasons why the algorithm in Wang *et al.* (2006) cannot work very fast.

The results of the MMPRM expansions are shown in Table 2. The maximum improvement could be as much as 71.7% and the average improvement is 43.2%.

Table 2   Experimental comparison between a multi-output fixed polarity Reed-Muller (FPRM) expansion under polarity 0 and a multi-level mixed polarity RM (MMPRM) form

Circuit	Number of I/Os	Number of literals		Imp_1 (%)	CPU time (s)
		Under polarity 0	Proposed		
Alu4	14/8	33270	9390	71.7	5.69
Apex4	9/19	5140	3625	29.5	2.11
Inc	7/9	480	281	41.5	<0.001
Misex1	8/7	294	164	44.2	<0.001
Misex3	14/14	53389	19959	62.6	11.97
Bw	5/28	452	412	8.8	<0.001
B12	15/9	907	464	48.8	<0.001
Clip	9/5	1286	570	55.6	<0.001
Con1	7/2	50	36	28.0	<0.001
Rd73	7/3	189	106	43.9	<0.001
Rd84	8/4	352	170	51.7	<0.001
Sao2	10/4	6493	2736	57.8	0.19
Table3	14/14	63500	28797	54.6	6.90
MT1*	20/28	78053	47204	39.5	83.20
MT2*	25/20	58244	38416	34.0	147.56
MT3*	30/30	76522	62406	18.4	352.30
Avg. imp. (%)				43.2	

* Randomly generated benchmarks under polarity 0

Table 3 shows the comparison results from the multi-output FPRM expansion under the best polarity to an MMPRM expansion. The third column lists the number of literals under the best polarities. The other columns have the same meaning as in Table 2. The maximum and average improvements are up to 71.4% and 49.9%, respectively.

In terms of spatial complexity, the proposed algorithm needs only to store a T table. Therefore, it can be estimated that the spatial complexity is $O(M\times n)$, where M is the number of elements in the onset table and n is the number of input variables for a specific function. The CPU time needed to solve a function with 30 input variables and 30 outputs is 352 s. This should not be a major problem as modern computer performance is rapidly improving.

Table 3 Experimental comparison between a multi-output fixed polarity Reed-Muller (FPRM) expansion under the best polarity and a multi-level mixed polarity RM (MMPRM) form

Circuit	Number of I/Os	Number of literals		Imp_1 (%)	CPU time (s)
		Under polarity 0	Proposed		
Alu4	14/8	27183	7770	71.4	4.66
Apex4	9/19	5140	3625	29.5	0.03
Inc	7/9	279	165	40.8	<0.001
Misex1	8/7	115	68	40.8	<0.001
Misex3	14/14	33356	10403	68.8	2.18
Bw	5/28	260	225	13.5	<0.001
B12	15/9	288	127	55.9	<0.001
Clip	9/5	1214	526	56.7	<0.001
Con1	7/2	51	25	51.0	<0.001
Rd73	7/3	189	106	43.9	<0.001
Rd84	8/4	352	170	51.7	<0.001
Sao2	10/4	805	255	68.3	<0.001
Table3	14/14	23954	10569	55.8	0.45
Avg. imp. (%)				49.9	

6 Conclusions

We present an algorithm to optimize multi-level mixed polarity RM (MMPRM) using an onset table for single- and multi-output fixed polarity RM (FPRM) functions. The algorithm has the advantages of being able to process functions with a large number of variables and needs less memory and time.

The mapping relationship between RM functions and the onset table is established. Based on the onset table a new strategy for the common variables searching is presented. The variables with high frequencies are usually extracted as the common variables. Some criteria and conditions are introduced to make the algorithm work effectively. Furthermore, by using a multiplexer model, a multi-output function can be treated as a single output function with address variables. The corresponding onset table and optimization method for multi-output RM functions is also proposed. The average saving in literals is approximately 54% when the RM functions are single output and under polarity 0. In comparison with the published results of single output MMPRM, the proposed algorithm can obtain more than 80% average time saving, with a small increase in the number of literals. For the multi-output RM function optimization, the average savings of literals is more than 40%.

References

Al Jassani, B.A., Urquhart, N., Almaini, A.E.A., 2008. Optimization of MPRM functions using tabular techniques and genetic algorithms. *Mediterr. J. Electron. Commun.*, **4**(4):115-125.

Almaini, A.E.A., Thomson, P., Hanson, D., 1991. Tabular techniques for Reed-Muller logic. *Int. J. Electron.*, **70**(1):23-24. [doi:10.1080/00207219108921253]

Balasubramanian, P., Edwards, D.A., Narayanan, C.H., 2007. Low Power Synthesis of XOR-XNOR Intensive Combinational Logic. IEEE Conf. on Electrical and Computer Engineering, p.243-246. [doi:10.1109/CCECE.2007.66]

Green, D.H., 1990. Reed-Muller canonical forms with mixed polarity and their manipulations. *IEE Proc. E: Comput. Dig. Techn.*, **137**(1):103-113. [doi:10.1049/ip-e.1990.0010]

Gupta, P., Agrawal, A., Jha, N.K., 2006. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.*, **25**(11):2317-2330. [doi:10.1109/TCAD.2006.871622]

Jankovic, D., Stankovic, R.S., Moraga, C., 2009. Optimization of polynomial expressions by using the extended dual polarity. *IEEE Trans. Comput.*, **58**(12):1710-1725. [doi:10.1109/TC.2009.113]

Pradhan, S.N., Chattopadhyay, S., 2008. Two-level AND-XOR networks synthesis with area-power trade-off. *Int. J. Comput. Sci. Network Secur.*, **8**(9):365-375.

Tran, A., Lee, E., 1993. Generalization of tri-state map and a composition method for minimization of Reed-Muller polynomials in mixed polarity. *IEE Proc. E*, **140**(1): 59-64.

Tran, A., Wang, J., 1993. Decomposition method for minimization of Reed-Muller polynomials in mixed polarity. *IEE Proc. E*, **140**(1):65-68.

Tsai, C.C., Marek-Sadowska, M., 1997. Boolean functions classifications via fixed polarity Reed-Muller forms. *IEEE Trans. Comput.*, **46**(2):173-186. [doi:10.1109/12.565592]

Wang, L., Almaini, A.E.A., 2002. Optimization of Reed-Muller PLA implementations. *IEE Proc.-Circ. Dev. Syst.*, **149**(2):119-128. [doi:10.1049/ip-cds:20020354]

Wang, L., Xia, Y., 2008. A Fast Algorithm for Multi-level Mixed-Polarity Reed-Muller Functions Optimization. 11th IEEE Int. Conf. on Communication and Technology, p.509-512. [doi:10.1109/ICCT.2008.4716096]

Wang, L., Xia, Y., Yang, M., Almaini, A.E.A., 2006. An approach to obtain compacted multi-level mixed polarity Reed-Muller functions. *WSEAS Trans. Circ. Syst.*, **5**(5): 625-632.

Xia, Y., Wang, L., Zhou, Z., Ye, X., Hu, J., Almaini, A.E.A., 2005. Novel synthesis and optimization of multi-level mixed polarity Reed-Muller functions. *J. Comput. Sci. Technol.*, **20**(6):895-900. [doi:10.1007/s11390-005-0895-2]