Available online at www.sciencedirect.com
ScienceDirect
Progress in Natural Science 19 (2009) 1311–1315
Progress in Natural Science
www.elsevier.com/locate/pnsc

A detection method for logic functions suitable for dual-logic synthesis

Yinshui Xia *, Fei Sun, Keyi Mao

The Faculty of Information Science & Engineering, Ningbo University, Ningbo 315211, China

Received 5 September 2008; received in revised form 23 October 2008; accepted 21 November 2008

Abstract

Logic functions can be implemented in either AND/OR/NOT-based traditional Boolean (TB) logic or AND/XOR-based Reed–Muller (RM) logic. To the majority of logic functions, it will be beneficial to be partially implemented in both TB logic and RM logic, called dual-logic. In this paper, a detection condition favoring dual-logic synthesis is proposed. A corresponding detection algorithm is developed and implemented in C. The algorithm is applied to test a set of MCNC91 benchmarks for verifying the algorithm. The results show that the proposed algorithm is more efficient than published ones.

Keywords: Logic synthesis; Boolean logic; Reed–Muller logic; Detection method

1. Introduction

Logic functions can be implemented by either using AND/OR/NOT-based traditional Boolean (TB) logic or using AND/XOR-based Reed–Muller (RM) logic. Statistically, 50% of applications can obtain better design under RM logic domain than that under TB logic domain [1,2]. For example, for an n variable parity function, using TB logic to implement takes 2^{n-1} product terms or $2^{n-1}n$ literals because each product term contains n literals. However, if RM logic is used to implement, it only takes n product terms or n literals because each product term consists of only one literal [3–6]. Furthermore, with the increase in n, potential savings of a chip area and power dissipation are significant. In addition, RM logic-based design is of good testability, which offers an efficient solution to overcome verification difficulty for today's IC industry. Therefore, for digital designs, both TB logic and RM logic are of equal importance. If two logic domains are combined to synthesize logic functions, then better solutions could be obtained. To do so, the first task is to efficiently detect whether a logic function is favorable to implement under TB logic or RM logic. A traditional synthesis flow is as follows: converting a logic function in both TB logic and RM forms; synthesizing them under two logic domains, respectively; comparing the synthesis results and determining implementation of the logic domain. However, there are two drawbacks for this flow. First, it is time-consuming. Second, only a small percentage of logic functions are favorable to implement under TB logic or RM logic. In fact, the majority of logic functions are favorable to partial implementation in both TB logic and RM logic, which is called dual-logic synthesis.

In terms of dual-logic synthesis, to our best knowledge, the only work was done by Dubrova and Bengtsson [7], which is an XOR logic detection algorithm. The authors proposed a sufficient condition. If a function meets the sufficient condition, then the function can be represented in a form as $f = (g \oplus h) + r$ and the number of product terms is fewer than that in the Sum of Products (SOP) form. However, the condition is based on counting the number of minimum product terms in the TB logic form compared

* Corresponding author. Tel.: +86 15901227121; fax: +86 574 87600491.
E-mail address: xiayinshui@nbu.edu.cn (Y. Xia).

1002-0071/$ - see front matter © 2009 National Natural Science Foundation of China and Chinese Academy of Sciences. Published by Elsevier Limited and Science in China Press. All rights reserved.
doi:10.1016/j.pnsc.2008.11.014

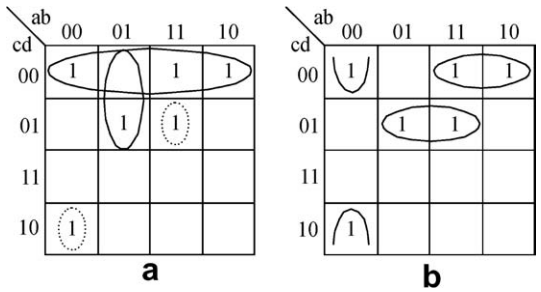

Fig. 1. Karnaugh map of a four variable function: (a) Karnaugh loop based on the condition in [7] and (b) Karnaugh loop in TB logic.

with the number of non-minimum product terms in $f = (g \oplus h) + r$. Furthermore, the condition does not take the structure of the remainder function r into account. Thus, the application of the condition has its limitations. Take the function in Fig. 1, for example, the function meets the sufficient condition proposed in [7], which means the function synthesizing in Karnaugh loops as in Fig. 1(a) can obtain better results than that in Fig. 1(b). However, from Fig. 1(a), it takes four product terms while from Fig. 1(b) it only takes three product terms.

In this paper, a detection condition favoring dual-logic domain synthesis is proposed. A corresponding detection algorithm is developed and implemented in C. The algorithm is applied to a set of MCNC91 benchmarks to test the algorithm efficiency.

2. Terminology and definitions

If the function f:$\{0, 1\}^n \rightarrow \{0, 1\}$, then it is called a completely specified function, or CSF in brief. If function f:$\{0, 1\}^n \rightarrow \{0, 1, -\}$, where "–" denotes a "do not care value", then it is called an incompletely specified function, or ISF in brief. For an ISF, F_f, R_f and D_f are used to denote onset, offset and do not care set of the function f, respectively; while for a CSF, only F_f and R_f exist. In this paper, CSF is focused. For convenience, the following definitions are helpful.

Definition 1. One decimal integer i is represented into n-bit binary code $(x_{n-1}x_{n-2}, \ldots, x_0)$, $i = \sum_{k=0}^{n-1} x_k \cdot 2^k$, and $x_k \in \{0, 1\}$; $H_{ij} = \sum_{k=0}^{n-1}(x_k \oplus y_k)$ is called the Hamming Distance (HD) of two binary codes, $(x_{n-1}x_{n-2}, \ldots, x_0)$, $(y_{n-1}y_{n-2}, \ldots, y_0)$, corresponding to two decimal numbers i and j where $i = \sum_{k=0}^{n-1} x_k \cdot 2^k$, $j = \sum_{k=0}^{n-1} y_k \cdot 2^k$, and "$\oplus$" represents exclusive OR.

Given an n variable function $f(x_0, x_1, \ldots, x_{n-1})$, it can be represented as $f(x_0, x_1, \ldots, x_{n-1}) = \sum_{i=0}^{2^n-1} a_i m_i$ where m_i is called minterm, which is a Boolean product of the variables, x_0, x_1, $\cdots$, x_{n-1}, or their complements, and the variables in m_i are also called literals. $a_i \in \{0, 1\}$ is the coefficient of the ith minterm. If $a_i = 1$, the ith minterm exists in the function expression. Otherwise, it does not. $\{a_i = 1\}$ consists of the onset F_f. A product term is also called a cube. Hence, cube and product term are used interchangeably in this paper.

Definition 2. The intersection of two sets A and B, denoted by $A \cap B$, is the union of the pairwise intersection of the cubes from sets A and B. The union of two sets, denoted by $A \cup B$, is the union of the cubes from A and B.

Definition 3. The complement of the A is denoted by $\overline{A}$ and $A \cap \overline{A} = \phi$.

Definition 4. A cube is called the minimum cube if it does not cover any other cubes except itself. A supercube of two cubes a and b is the cube at least covering a and b, denoted by $\sup(a, b)$.

From the above definitions, it is known that any minterm m_i covers one Karnaugh lattice while any supercube covers at least two Karnaugh lattices. It can be deduced that a supercube of two minimum cubes m_i and m_j with $H_{ij} = 2$ covers four Karnaugh lattices.

Definition 5. Let $l_1, l_2, \ldots, l_k$ $(k > 1)$ be cubes from the onset F_f of the Boolean function f:$\{0, 1\}^n \rightarrow \{0, 1\}$ and let the intersection of $\sup(l_1, l_2, \ldots, l_k)$ with the offset R_f be a non-empty set of cubes $\sup(l_1, l_2, \ldots, l_k) \cap R_f = \bigcup_{i=1}^{p} c_i$, $p \geqslant 1$. $\sup(l_1, l_2, \ldots, l_k)$ is called the $\frac{k}{p+k}$-cube.

For example, Fig. 2 is a Karnaugh map of a four variable function $f = \sum(m_0, m_5, m_{11}, m_{14}, m_{15})$. The Karnaugh loop a is a $\frac{1}{2}$-cube while the Karnaugh loop b is a $\frac{3}{4}$-cube.

3. Principle of sufficient condition

A logic function can be implemented by AND/OR/NOT-based TB logic, denoted as f_B, or by AND/XOR-based RM logic, denoted as f_{RM}. For a specific logic function, in terms of die size and power dissipation, it is interesting to know which logic is favorable to implement; it is more important to know whether the logic function is beneficial for mono-logic or dual-logic being implemented. This problem can be addressed as follows.

Given a logic function in TB logic form, detect whether a compact form, in which part of the function is implemented by RM logic and the remainder is implemented by TB logic, could be obtained under dual-logic synthesis. In other words, an AND/OR/NOT-based expression f_B can be transformed into $f_B^* + f_{RM}$ called dual-logic, denoted as $f_D = f_B^* + f_{RM}$. Here, f_{RM} denotes that the func-

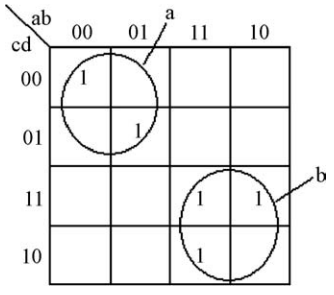

Fig. 2. Karnaugh map of a four variable function f.

tion is expressed into RM form while f_B^* represents that the function is in Boolean form; while f_B and f_D have the same logic function.

To see whether dual-logic synthesis is beneficial to the function f_B, a cost function is needed. As known, in two-level AND/OR expression, the number of cubes is used to evaluate the implementation cost of the function while in multi-level expression the number of literals is used as the cost. Here, both the number of cubes and the number of literals are used to measure the cost. The criteria to tell which form is compact are if the number of cubes is different, then the compact form can be directly determined, that is, the fewer the number of cubes, the more compact the function form; if the number of cubes from f_B and f_D is the same, the number of literals is used to measure the cost.

The following Lemma is useful to lead to the solution.

Lemma 1. *Let $l_1, l_2, \ldots, l_k$, $k > 1$ be cubes from the onset F_f of Boolean function f*: i$\{0,1\}^n \to \{0,1\}$, *n be the number of variables, and let the intersection of* $\sup(l_1, l_2, \ldots, l_k)$ *with the offset R_f be a non-empty set of cubes* $\sup(l_1, l_2, \ldots, l_k) \cap R_f = \bigcup_{i=1}^{p} c_i$, $p \geqslant 1$, $k = 1, 2, \ldots, 2^n$.

(i) if $p < k$, $k = 2, 3, \ldots, 2^n - 1$, for each cube c_i, a cube $b_i \in F_f$ such that $\sup(b_i, c_i) \cap R_f = c_i$ can be found as well as $\sup(l_1, l_2, \ldots, l_k) \cap \sup(b_i, c_i) = c_i$, then

$$\sup(l_1, l_2, \ldots, l_k) \oplus \bigcup_{i=1}^{p} \sup(b_i, c_i) = \bigcup_{i=1}^{k} l_i \cup \bigcup_{j=1}^{p} (\sup(b_j, c_j) - c_j) \quad (1)$$

$\sup(l_1, l_2, \ldots, l_k)$ is called the $\frac{k}{k+p}$-cube.

(ii) if $p = k = 2^r$, and $r = 1, 2, 3, \ldots, n-1$, for each cube l_i, always there is cube l_j, and $l_i, l_j \in F_f$, $i \neq j$, such that $\sup(l_i, c_i) \cap R_f = \sup(l_j, c_i) \cap R_f = c_i$ can be found as well as $\sup(l_j, c_i) \cap \sup(l_i, c_i) = c_i$. The minimum HD between any two cubes is 2. Then

$$\sup\left(l_1, l_2, \ldots, l_{\frac{k}{2}}\right) \oplus \bigcup_{j=\frac{k}{2}+1}^{k} \sup(l_j, c_j) = \bigcup_{h=1}^{k} l_h \quad (2)$$

$\sup(l_1, l_2, \ldots, l_k)$ is called the $\frac{1}{2}$-cube.

Proof. For the proof of Case (i) please refer to Ref. [7], and the proof of Case (ii) is shown as follows. Since $p = k = 2^r$, it is known that $\sup(l_1, l_2, \ldots, l_k)$ covers 2^{r+1} Karnaugh lattices, in which 2^r lattices are occupied by the onset while 2^r lattices are occupied by the offset. Since for each cube l_i, always there is cube l_j, and $l_i, l_j \in F_f$, $i \neq j$, such that $\sup(l_i, c_i) \cap R_f = \sup(l_j, c_i) \cap R_f = c_i$ can be found as well as $\sup(l_j, c_i) \cap \sup(l_i, c_i) = c_i$. Hence, the onset $\{l_i\}$ is evenly distributed among 2^{r+1} Karnaugh lattices. Using the deductive proof, it can be proved that for each l_j, $(j = \frac{k}{2} + 1, \frac{k}{2} + 2, \ldots, k)$, can always find a c_j $(j = \frac{k}{2} + 1, \frac{k}{2} + 2, \ldots, k)$ covered by $\sup(l_1, l_2, \ldots, l_{\frac{k}{2}})$, so that l_j is logically adjacent to c_j. Then, Eq. (2) is satisfied. □

Lemma 1 gives a condition for substituting a subset F_S of the onset F_f of a function f_B by two functions f_B^* and f_{RM} so that $f_B = f_D = f_B^* + f_{RM}$ and the implementation cost in $f_B^* + f_{RM}$ is smaller than that in f_B. The following theorem guarantees that this condition is sufficient.

Theorem 1 (Sufficient condition)**.** *If a Boolean function f_B satisfies Lemma* 1 *for some set of cubes $(l_1, l_2, \ldots, l_k)$, $l_i \in F_f$, $\forall i \in \{1, 2, \ldots, k\}$, then the implementation cost of f_D is smaller than that of f_B.*

Proof. Suppose that a part of the function f_B satisfies Case (i) in *Lemma* 1. Then Eq. (1) is satisfied. From the right part of Eq. (1), $\bigcup_{i=1}^{k} l_i \cup \bigcup_{j=1}^{p} (\sup(b_j, c_j) - c_j)$, which is an AND/OR/NOT-based expression, it costs $p + k$ terms. However, from the left part of Eq. (1), $\sup(l_1, l_2, \ldots, l_k) \oplus \bigcup_{i=1}^{p} \sup(b_i, c_i)$, which is in RM form, and it needs $p + 1$ terms. Since $k > 1$, the left part of the expression is more compact than the right part.

If part of the function f_B satisfies Case (ii) in *Lemma* 1, then Eq. (2) is met. From the right part of Eq. (2), $\bigcup_{h=1}^{k} l_h$, it needs k terms. Similarly, from the left part of Eq. (2), $\sup\left(l_1, l_2, \ldots, l_{\frac{k}{2}}\right) \oplus \bigcup_{j=\frac{k}{2}+1}^{k} \sup(l_j, c_j)$, it costs $\frac{k}{2} + 1$ terms. Again, since $k > 1$, the left part of the expression is equal to or more compact than the right part. □

In the following, two examples are shown to explain how to apply *Theorem* 1 to detect functions.

Example 1. A 4-variable function f_1 is shown in Fig. 3.

On the one hand, let $m_1 = 0001$, $m_2 = 0101$, $m_3 = 1001$, and $m_4 = 1100$. Then we have $\sup(m_1, m_2, m_3) = --01$ as the loop in Fig. 3 and $\sup(m_1, m_2, m_3) \cap R_f = 1101 = c_1$. Since $p = 1$ and $k = 3$, $\sup(m_1, m_2, m_3)$ is a $\frac{3}{4}$-cube. It can be seen that $b_1 = m_4 = 1100$ satisfies $\sup(b_1, c_1) \cap R_f = c_1$. Since $\sup(b_1, c_1) - c_1 = b_1$, we obtain that $\sup(m_1, m_2, m_3) \oplus \sup(b_1, c_1) = m_1 \cup m_2 \cup m_3 \cup b_1$.

Hence, $f_{RM} = \overline{x_1} x_2 \oplus \overline{x_1} x_3 x_4$. On the other hand, let $m_5 = 0110$ and $m_6 = 1110$. Then we have $f_B^* = x_1 \overline{x_2} x_4$. Hence, under the dual-logic synthesis, $f_1 = f_D = f_{RM} + f_B^* = \overline{x_1} x_2 \oplus \overline{x_1} x_3 x_4 + x_1 \overline{x_2} x_4$. However, if Boolean logic is used to synthesize the function, then $f_1 = f_B = \overline{x_1} x_2 \overline{x_3} + \overline{x_1} x_2 \overline{x_4} + \overline{x_2} x_3 x_4 + x_1 \overline{x_2} x_4$. It can be seen that using RM logic can save one cube in terms of the number of cubes but save

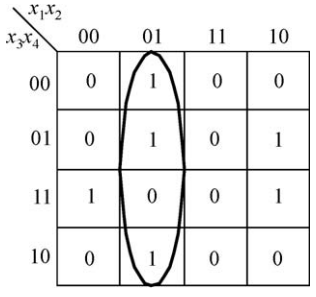

Fig. 3. Part of function f_1 that meets Case (i) in Lemma 1.

4 literals in terms of the number of literals. The function meets Case (i) in *Lemma* 1.

Example 2. A 4-variable function f_2 is shown in Fig. 4.

Let $m_1 = 0001$, $m_2 = 0100$, $m_3 = 1101$ and $m_4 = 1000$. Then we have $\sup(m_1, m_2, m_3, m_4) = --0-$ as the loop in Fig. 4 and $\sup(m_1, m_2, m_3, m_4) \cap R_f = (0000, 0101, 1001, 1100) = (c_1, c_2, c_3, c_4)$. Since $k = p = 4$, $\sup(m_1, m_2, m_3, m_4)$ is a $\frac{1}{2}$-cube. It can be seen that $\sup(l_i, c_i) \cap R_f = \sup(l_j, c_i) \cap R_f = c_i$ can be found as well as $\sup(l_j, c_i) \cap \sup(l_i, c_i) = c_i$, $l_i, l_j \in F_f$, $i \neq j$ and $i, j \in \{1, 2, 3, 4\}$. The minimum HD among two cubes is 2. Then, the loop in Fig. 4 can be expressed in the form $\sup(m_1, m_2) \oplus \sup(m_3, c_1) \oplus \sup(m_4, c_2) = \overline{x_1}x_3 \oplus \overline{x_1}x_2\overline{x_4} \oplus \overline{x_1}x_2x_4$ while the remainder can be expressed as $f_B^* = x_3x_2x_1$. Hence, $f_1 = f_D = f_{RM} + f_B^* = \overline{x_1}x_3 \oplus \overline{x_1}x_2\overline{x_4} \oplus \overline{x_1}x_2x_4 + x_3x_2x_1$ while under Boolean logic domain, $f_1 = f_B = \overline{x_1}x_2\overline{x_3}\overline{x_4} + \overline{x_1}x_2x_3x_4 + x_2x_3x_4 + \overline{x_1}\overline{x_2}x_3\overline{x_4} + x_3x_2x_1$. It can be seen that using dual-logic synthesis can save one cube. This example is the case $\frac{1}{2}$-cube, which is the Case (ii) in *Lemma* 1.

4. Dual-logic detection algorithm

Based on *Theorem* 1, one is able to detect whether a logic function is beneficial for using dual-logic implementation. The larger the subset of the onset of a function f satisfying *Lemma* 1, the more the f can benefit from dual-logic synthesis. Exhaustively searching such a subset may not be necessary since what we need is to quickly evaluate whether a given function is likely to benefit from dual-logic implementation, but not to optimize the function. The proposed algorithm is used to lead to the solution. The input is the onset F_f and offset of R_f. The output is "YES" if a function is likely to benefit from dual-logic minimization, "NO" otherwise. The algorithm repeats the following steps:

Step 1. Choose a pair of cubes, m_j, m_{j+1}, and compute its supercube $\sup(m_j, m_{j+1})$.
Step 2. Compute the intersection $\sup(m_j, m_{j+1}) \cap R_f = \bigcup_{i=1}^{p} c_i$.
Step 3. Check whether $\sup(m_j, m_{j+1})$ is a kind of $\frac{k}{k+p}$-cube. If yes, let $N_{RM1} = N_{RM1} + 1$, and delete m_j, m_{j+1} and go to Step 1. If no, go to Step 4.

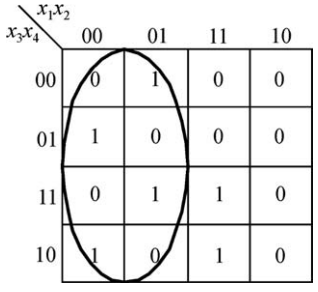

Fig. 4. Part of function f_2 that meets case (ii) in Lemma 1.

Step 4. Check whether $\sup(m_j, m_{j+1})$ is a kind of $\frac{1}{2}$-cube. If yes, let $N_{RM2} = N_{RM2} + 1$, and delete m_j, m_{j+1} and go to Step 1. If no, go to Step 5.
Step 5. Check whether $j < n - 1$. If yes, let $j = j + 1$, go to Step 1. If no, go to Step 6.
Step 6. If $N_{RM1} + N_{RM2} > 1$, return "YES" with the number of pairs, N_{RM1}, N_{RM2}. Otherwise, return "NO".

Its pseudocode is shown as follows:

```
Check_Dual_Logic(F_f, R_f)
Input: F_f, R_f
Output: "YES" if function is likely to benefit from dual-logic minimization, "NO" otherwise.
  N_RM1 = 0, N_RM2 = 0;
  for (each pair of cubes, (m_j, m_{j+1}) ∈ (F_f × F_f)){
    Flag_ lemma_ satisfied = 0;
    Compute_ supercube sup(m_j, m_{j+1});
    Compute sup(m_j, m_{j+1}) ∩ R_f = ∪_{i=1}^{p} c_i;
    for (each cube c_i ∈ {c_1, ..., c_p}) {
    flag_found_a = 0;
    for (each cube b_l ∈ F_f) {
      Compute_ supercube sup(b_l, c_i);
      if(sup(b_l, c_i) ∩ R_f = c_i){
        if(sup(m_j, m_{j+1}) ∩ sup(b_l, c_i) = c_i) {
        flag_ found_a = 1; /* found b_l for the c_i */
          N_RM1 = N_RM1 + 1;
          break; /* break for-loop */
        }
      }
  }
  if(flag_ found_a == 0) {/* failed to find b_l for some c_i */
    flag_found_b = 0;
      if(sup(l_i, c_i) ∩ R_f = c_i&&sup(l_j, c_i) ∩ R_f = c_i){
        if(sup(l_j, c_i) ∩ sup(l_i, c_i) = c_i){
        flag_found_b = 1;
        N_RM2 = N_RM2 + 1;
          break; /* break for-loop */
          }
    }
  }
  if(flag_found_a == 1||flag_found_b == 1) {
    flag_ lemma_ satisfied = 1;
    }
  }
  if(flag_ lemma_ satisfied = 1)/* Lemma is satisfied for (m_j, m_{j+1})*/
    N_RM = N_RM1 + N_RM2;
  }
  if(N_RM > 0)
    return (YES; N_RM1, N_RM2, N_RM);
  else
    return (NO)
```

The main purpose of the proposed algorithm is to determine whether the expression of a function is more compact using dual-logic synthesis than using Boolean logic by

counting the number of pairs (m_j, m_{j+1}) that satisfies *Lemma* 1. Obviously, the more the number of pairs (m_j, m_{j+1}), the more compact the function expression using dual-logic.

5. Experimental results

The proposed algorithm was implemented in C and run under Pentium 486/2.4 GHz/512 M. A set of experiments was performed on 18 benchmarks from MCNC91. Table 1 summarizes the results. Column 1 shows the circuit name. Column 2 gives the number of inputs n and outputs. Column 3 refers to the cube number of F_f in the cover computed by Espresso [5]. Column 4 gives the number, N_{RM1}, satisfying *Lemma* 1 (i) while Column 5 shows the number, N_{RM2}, satisfying *Lemma* 1 (ii). Column 6 gives the number, N_{RM}, that shows the satisfaction of *Lemma* 1. Column 7 shows the result "Yes" or "No" to indicate whether the function is suitable to the dual-logic synthesis. The final column presents the CPU time to detect the function.

From *Theorem* 1, the larger the function with N_{RM}, the less the cost to implement under the dual-logic synthesis. From the table, it can be seen that 16 out of 18 circuits are marked with "Yes", which indicates that those functions are beneficial from the dual-logic synthesis since they have larger N_{RM}. Functions with large N_{RM}, like 9sym, Newill, Rd84, Sym10, Xor5, T481, Cordic are known to have a smaller cost using dual-logic than using Boolean logic, which is confirmed by the published results [7–9]. However, functions like Ryy6, Cm150a are not beneficial under dual-logic synthesis. It should be pointed out that the algorithm in [7] is unable to detect T481 properly. But the detection result based on the proposed algorithm shows that it is beneficial for T481 to synthesize under dual-logic that is confirmed in [7] based on its structure analysis. From the CPU time in the table, it is known that the proposed algorithm has reasonable time complexity.

Table 1
Number of pairs satisfying Lemma 1.

Benchmarks	I/O	Products No.	N_{RM1}	N_{RM2}	N_{RM}	Result	CPU time(ms)
Newill	8/1	22	3	4	7	Yes	0
Newtag	8/1	14	3	2	5	Yes	0
Rd73	7/3	141	4	48	52	Yes	16
Rd84	8/4	256	10	118	128	Yes	188
Ryy6	16/1	112	0	0	0	No	46
9Sym	9/1	87	0	41	41	Yes	47
Sym10	10/1	837	0	412	412	Yes	10,500
T481	16/1	481	86	53	139	Yes	4157
Xor5	5/1	16	0	4	4	Yes	0
5Xp1	7/10	75	15	3	18	Yes	0
Cm150a	21/1	17	0	0	0	No	0
Cordic	23/2	1206	365	0	365	Yes	19,610
Con1	7/2	9	3	0	3	Yes	0
Alu4	14/8	1028	118	137	255	Yes	1375
Duke2	22/29	87	44	12	56	Yes	31
I5	133/66	369	78	6	84	Yes	250
Misex2	25/18	29	3	0	3	Yes	16
Rd53	5/3	32	2	8	10	Yes	0

6. Conclusions

In this paper a sufficient condition is formulated for a function $f(x_0, x_1, \ldots, x_{n-1})$ to be synthesized under dual-logic domain with less cost than that based on Boolean logic. Based on this condition, an algorithm for detecting whether a function is likely to benefit from dual-logic synthesis is developed and implemented in C. The experimental results tested on MCNC91 benchmarks confirm that the proposed algorithm can properly detect whether the functions are beneficial under the dual-logic domain synthesis. Future work will cover how to optimize a function under the dual-logic synthesis.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. 60676017), Zhejiang Natural Science Foundation (Grant No. R105614) and the Science & Technology Program from the Department of Zhejiang Sci. & Tech. (Grant No. 2007C24017). The work is also sponsored by the K.C. Wong Magna Fund in Ningbo University.

References

[1] Xia Y, Wu X, Almaini AEA. Power minimization of FPRM functions based on polarity conversion. J Comput Sci Technol 2003;18(3):325–31.
[2] Xia Y, Ye X, Wang L, et al. Novel synthesis method of mixed polarity Reed–Muller functions. In: Proceedings of third IASTED conference on circuits, signals, and systems, Marina del Rey, CA, USA, October 24–26; 2005. p. 148–53.
[3] Dautovic S, Novak L. A comment on "Boolean functions classification via fixed polarity Reed–Muller form". IEEE Trans Comput 2006:1067–9.
[4] Sasao T. A design method for AND–OR–EXOR three-level networks. In: Proceedings of the international workshop on logic synthesis, Lake Tahoe, California, May 3; 1995. p. 811–20.
[5] Jabir A, Saul J. A heuristic decomposition algorithm for AND–OR–EXOR three-level minimization of Boolean functions. In: Proceedings of the 4th international workshop on the applications of the Reed–Muller expansion in circuit design, Victoria, BC, Canada, August 20–21; 1999. p. 55–74.
[6] Shafiqul Khalid ATM, Awwal AAS. XOR realization using KH-map. In: IEEE proceedings of the national aerospace and electronics conference. Ohio, USA, May 20–23; 1996. p. 280–5.
[7] Dubrova E, Bengtsson T. An algorithm for detecting XOR-type logic. In: Proceeding of the 5th international workshop of applications of the Reed–Muller expansion in circuit design, Mississippi, USA, August 10–11; 2001. p. 271–6.
[8] Maxfield M. A Reed–Muller extraction utility. EDN access for design [1996-06-24].
[9] Debnath D, Sasao T. GRMIN: a heuristic minimization algorithm for generalized Reed–Muller expression. In: Proceedings of the Asia and South Pacific design automation conference (ASPDAC'95), Chiba, Japan, August 29–September 1; 1995. p. 341–7.