

A Frontier Algorithm for Optimization of Multiple-Valued Logic functions

Mostafa I. Abd-El-Barr
Department of Computer Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31261
Saudi Arabia
mostaf@ccse.kfupm.edu.sa

Muhammad M. Abd-El-Barr
Division of Engineering Science
University of Toronto
Toronto, Ontario M5S 1A4
Canada
abdelba@ecf.utoronto.ca

Abstract

A new algorithm, called the Frontiers algorithm, for optimizing the number of product terms required for the implementation of monotonic and permuted monotonic MVL functions is proposed. An experimental system restricted to the case of 2-variable 4-valued set of logic functions has been programmed using the C language and was interfaced to the HAMLET CAD tool to implement the proposed algorithm. The system was tested using 2231 randomly generated monotonic and permuted monotonic functions. The results obtained indicate that the frontiers-based algorithm compares favorably to existing heuristic minimization techniques with the added advantage that it requires less number of implicants to represent the target functions.

1 Introduction

Consider an r -valued logic system where the radix $R = \{0, 1, 2, \dots, r-1\}$. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of n variables, where each x_i can take on values from the set R . An n -variable r -valued function $f(X)$ is the mapping: $R^n \rightarrow R$. A minterm of a function is defined as a function value corresponding to a specific assignment of values a_1, a_2, \dots, a_n to variables in X such that $0 < f(a_1, a_2, \dots, a_n) < r$.

There exists a number of functionally complete sets of operators each of which can be used to represent MVL functions [6]. The possible functionally complete sets include:

1. The set consisting of the operators Maximum (max), Minimum (min), and window literal.
2. The set consisting of the operators truncated-sum (tsum), Minimum (min), and window literal.
3. The set consisting of the operators Maximum (max), Minimum (min), and the cycle.

The first two sets of operators have been used extensively in the literature and therefore will be used in this paper. For definitions of the operators used on these two sets, see [6].

PLAs are of special interest for the implementation of MVL circuits in VLSI. The structure of a MVL PLA consists of a min plane and a max (tsum) plane. Inputs to the min plane consist of the constants 1, 2, ..., $r-1$ and the window literals of all input variables x_1, x_2, \dots, x_n . The outputs of the min plane consist of the product terms needed for the implementation of the given function(s). These product terms are combined in the max (tsum) plane in order to produce the target function(s) [3]-[7].

Absolute minimization of SOP expressions of MVL functions is prohibitively expensive. For example, absolute minimization of only 2-variable 4-valued functions can require in the order of days to be performed [11]. Therefore, heuristic-based algorithms and minimization of special MVL functions, e.g. monotonic functions, are becoming increasingly attractive [5]-[10]. A number of heuristic algorithms has been developed for producing a near minimal realization for MVL functions [2]-[4], [10]. Most of these algorithms are based on an extension of the direct cover approach used in binary systems.

This paper is an attempt in the direction of optimizing MVL functions for implementation using PLAs. The paper proposes extending existing binary Frontiers algorithms [8]-[9] for use in the MVL domain. We also propose a MVL frontiers algorithm to optimize the SOP expressions of MVL monotonic and permuted monotonic functions.

2 The Binary Frontiers Algorithm

The frontiers algorithm has been used to detect the extensional equality of two monotonic functions [9]. It is based on the idea of representing all the arguments of a given function in a finite lattice $\{0, 1\}$ with $0 < 1$. Fig. 1 shows the lattice representation of three variables x, y , and z .

Consider the binary function $f(x, y, z) = x + yz$. The lattice argument values for such function is shown in 2.

In this figure, if the application of the function to a node in the lattice results in a 0, then that node is called a 0-node; otherwise the node is called a 1-node. For example, node $(x, y, z) = (1, 0, 0)$ is a 1-node, while

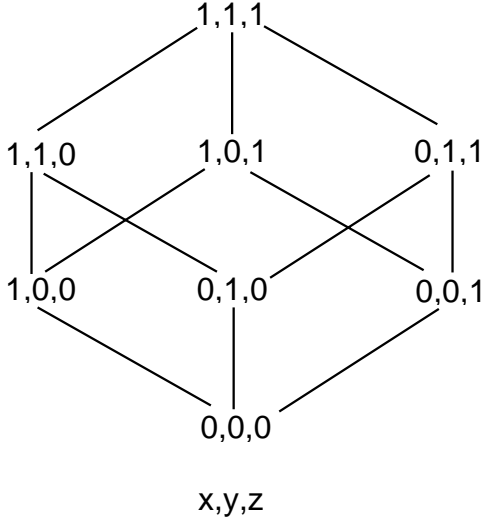


Figure 1: The lattice for a 3-variable function.

node (0,0,1) is a 0-node. In Fig. 2 all 0-nodes are marked with *. There are three such nodes in Fig. 2. It should be noted that a boundary line can be drawn such that all nodes above this boundary line are 1-nodes and all nodes below the boundary line are 0-nodes. This is a property of all monotonic functions. Such property provides a way for efficient representation of monotonic functions. Instead of listing all the 0- and the 1-nodes, a monotonic function can be completely characterized by the set of nodes that constitute the boundary between the 0-nodes and the 1-nodes. This set of nodes are called the Frontiers of the function. The 0-frontier is the set of the 0-nodes on the boundary and the 1-frontier is the set of 1-nodes on the boundary. For example, for the function $f(x, y, z) = x + yz$, the 1-frontiers are (0,1,1), (1,1,0) & (1,0,0) and the 0-frontiers are (0,0,0), (0,0,1) & (0,1,0).

In expressing a given function using the 0-frontiers (1-frontiers), it is possible to use only a subset of the 0-frontiers (1-frontiers) to fully express the function in the SOP form. For example, consider the function whose lattice shown in Fig. 2. It is possible to fully express this function using the subset $\{(0,1,0), (0,0,1)\}$ of the 0-frontiers. Such subset is called a maximal 0-nodes. The set of maximal 0-nodes (1-nodes) is the set of 0-nodes (1-nodes) in which only incomparable nodes are included. Incomparable nodes are those nodes which when a bit-wise XOR is performed on them will result a non-zero value in more than one bit position. A straightforward way for finding the 0-frontiers of a given function is to explore the lattice of the function from top to bottom.

An isomorphism between frontiers and canonical forms for monotonic binary functions has been established in [8]. According to this isomorphism, every monotonic function of the type $T^n \rightarrow T$ has a unique irredundant SOP representation which

can be derived directly based on the 1-frontiers (or the 0-frontiers). For example, assume the 1-frontiers of a monotonic function, f , is $A = \{(a_{1,1}, a_{1,2}, \dots, a_{1,n}), \dots, (a_{m,1}, a_{m,2}, \dots, a_{m,n})\}$ where m is the number of frontiers, and n is the number of variables of the function, then it has been shown that the irredundant SOP expression of the function is given by [8]:

$$f = \sum_{i=1}^m \prod_{j=1}^n a_{i,j} \cdot x_j$$

Consider, for example, a 3-variable function whose 1-frontiers are given by (0, 1, 0), (1, 0, 1), then using the above relation we can derive the irredundant SOP for the function to be $f_1(x, y, z) = y + xz$. Conversely, given the irredundant SOP expression for a function, the 1-frontiers of the function can be obtained without searching the lattice of the function. It should also be noted that using the duality property, an isomorphism between the 0-frontiers and the irredundant product-of-sum (POS) form can be established [8].

3 MVL Frontiers Algorithm

In this section, we extend the binary frontiers concept into the MVL domain. In order to be able to do so, we first introduce the concept of the MVL argument lattice and frontiers into the MVL system.

3.1 MVL Argument Lattice

The arguments lattice of an n-variable r-valued function, $f(x_1, x_2, \dots, x_n)$, form an n-tuple (a_1, a_2, \dots, a_n) , where $a_i \in R = \{0, 1, \dots, r-1\}$. In the domain of the argument tuple, we can define the following partial order (\leq).

Definition 1

For two arguments $A = (a_1, a_2, \dots, a_n)$ and $B = (b_1, b_2, \dots, b_n)$ if $a_i \leq b_i$ for all $i = 1, 2, \dots, n$, then $A \leq B$.

Definition 2

The argument lattice for an r-valued n-variable function is a finite lattice which is formed by representing all possible argument tuples of the function as a directed graph based on the partial order, \leq , given in definition 1. Each node in the lattice stands for an argument tuple.

Definition 3

For an r-valued n-variable function, f , if the result of applying f to an argument tuple denoted by the node A in the argument lattice is $k \in R = \{0, 1, 2, \dots, r-1\}$, then the node A is called a k-node of function f .

The argument lattice for 2-variable 4-valued functions is shown in Fig. 3.

Definition 4

For an r-valued n-variable function, f , a node A is said to be a maximal k-node of f if there is no other k-node B such that $A \leq B$.

Definition 5

For an r-valued n-variable function, f , a node A is a minimal k-node of f if there is no other k-node B such that $B \leq A$.

An r-valued n-variable function, $f(x_1, x_2, \dots, x_n)$, is said to be monotonically increasing (decreasing) over one of its variables, x_i , if for each valuation of the other

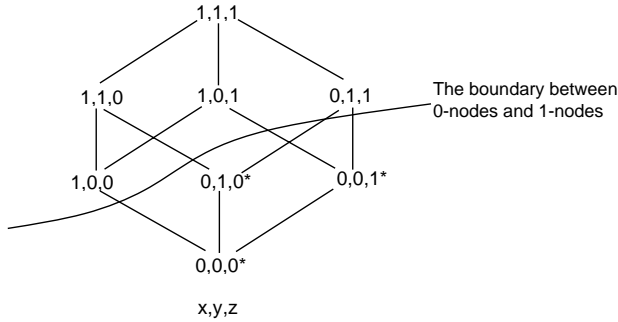


Figure 2: The 0-nodes and the 1-nodes for a three-variable function.

remaining variables the function value remains constant or increases (decreases) when increases. A monotonic function is defined as one which increases (decreases) over all of its variables [1]. If the given MVL function is monotonic, then all the k -nodes are above the $(k-1)$ -nodes and below the $(k+1)$ -nodes in the argument lattice. This will mean that the k -nodes represent the boundary between $(k-1)$ -nodes and $(k+1)$ -nodes in the lattice.

Definition 6

The k -frontiers of a monotonic r -valued n -variable function f are the minimal k -nodes of f , which constitute the boundary between $(k-1)$ -nodes and k -nodes in the argument lattice of f .

In an r -valued system, there are $r-1$ possible frontiers. These are 1-frontiers, 2-frontiers, ..., $(r-1)$ -frontiers. Consider, for example, the 4-valued 2-variable function shown in Fig. 3. The frontiers are: 1-frontiers $(1,1)$, $(2,0)$, 2-frontiers $(1,2)$, $(3,1)$, and 3-frontiers $(1,3)$, $(3,2)$.

It is possible to extend the algorithm for finding the frontiers of a given binary function to the case of MVL functions. This can be done by searching the lattice of the MVL function upward from the bottom $r-1$ times, each time finding a specific set of frontiers, say k -frontier, for $k \in \{1, 2, \dots, r-1\}$. An algorithm that can be used to find the frontiers of a given MVL function has been developed (see Fig. 4).

4 Permuted Monotonic Functions

In this section, a method that can be used in order to explore the idea of permuting the logic values of a given function for the purpose of determining whether the function can be transformed into a monotonic function [12]. A function for which there exists a logic permutation that can transform it into a monotonic function is called a permuted monotonic function. Consider, for example, the 2-variable 4-valued function whose map is shown in Fig. 5.

The function shown in Fig. 5(a) is not monotonic and therefore the frontier algorithm cannot be applied to such function. However, if the following permutation

$0 \Rightarrow 1, 1 \Rightarrow 2, 2 \Rightarrow 3, 3 \Rightarrow 0$, is applied, then the resulting function, f_1 , is monotonic (see Fig. 4(b)). The frontiers algorithm can then be applied to the function to obtain a reduced expression for such function. The original function can then be restored by applying the reverse permutation $0 \Rightarrow 3, 1 \Rightarrow 0, 2 \Rightarrow 1, 3 \Rightarrow 2$.

5 Implementation and Results

Based on the MVL frontiers algorithm and the functional transformation using permutation, the proposed MVL optimization algorithm can be summarized as a procedure as shown in Fig. 4. The proposed optimization algorithm is applicable for any n -variable r -valued function. However, we have restricted our implementation of the algorithm using C programming language to an experimental system for the 2-variable 4-valued functions case. This is done due to a number of practical reasons. We have applied the algorithm to 2231 randomly generated monotonic or permuted monotonic functions. The results obtained using our experimental system together with those obtained using Dueck and Miller (D&M) using HAMLET [11] are summarized below. The following specific example

Table 1: Summary of the results obtained.

Attribute	Frontiers	D&M
#functions tested	2231	2231
Total # product terms used in all functions	8320	8672
Average # product terms per function	3.729	3.887
Max. # product terms in a given function	6	8
Min. # product terms in a given function	1	1
# functions with fewer terms using Frontiers	709	
92 functions with 3 terms fewer		
139 functions with 2 terms fewer		
478 functions with 1 term fewer		
# functions with fewer terms using D & M		608
18 functions with 3 terms fewer		
107 functions with 2 terms fewer		
483 functions with 1 term fewer		
Number of functions having equal terms	914	914

functions were optimized using both the Frontiers algorithm and the D&M algorithm.

Table 2: Optimization of sample functions.

The function	# terms using Frontiers	# terms using D&M
0 2 2 2	5	6
0 2 1 3		
1 1 3 3		
1 1 3 3		
1 1 2 2	3	
1 1 2 2		
1 0 0 0		
2 0 0 0		
0 0 1 1	6	6
1 1 1 3		
1 2 2 3		
2 3 3 3		
2 2 3 3	4	5
2 2 3 3		
2 1 1 1		
3 1 1 1		
1 1 1 1	3	4
1 1 0 0		
1 1 2 2		
1 2 2 2		

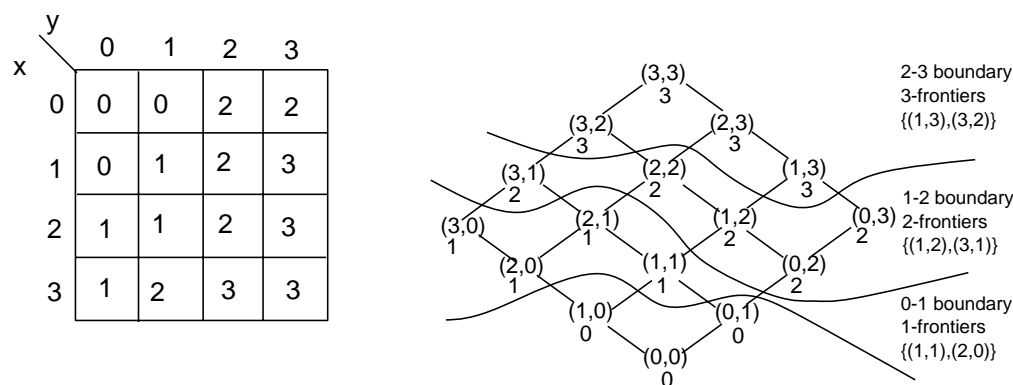


Figure 3: The MVL Frontiers of a 2-variable 4-valued monotonic function.

6 Concluding Remarks

In this paper, we have extended the concepts of frontiers and argument lattice to the case of MVL systems. A MVL frontiers algorithm to determine all frontiers of a monotonic MVL function is introduced. In addition, a frontiers-based algorithm for optimization of MVL functions is proposed. An experimental system restricted to the 2-variable 4-valued functions has been implemented using the C language. The system was tested using 2231 randomly generated monotonic or permuted monotonic functions. The results obtained using the D&M algorithm and the frontiers-based algorithms using the HAMLET CAD tool have been presented and compared. As an extension to the proposed system, we are currently working on extending the restricted system to include a larger set of MVL functions.

Acknowledgement

The authors would like to acknowledge the continued support of King Fahd University of Petroleum and Minerals.

References

- [1] M. Abd-El-Barr. On the design of mvl ccd logic circuits. *Ph.D. Thesis, University of Toronto, Canada*, 1986.
- [2] P. Besslich. Heuristic minimization of mvl functions. *IEEE Transactions on Computers*, c-35(2):134–144, 2 1986.
- [3] G. Dueck, P. Earle, P. Tirmualai, and J. Butler. Mvl pla minimization by simulated annealing. *ISMVL-22*, pages 66–74, 5 1992.
- [4] G. Dueck and D. Miller. A direct cover mvl minimization using the truncated sum. *ISMVL-86*, pages 232–240, 5 1986.
- [5] G. Dueck and D. Miller. A direct cover mvl minimization using the truncated-sum. *ISMVL-17*, pages 221–227, 5 1987.

- [6] A. Jain. Mvl design of current-mode cmos. *Ph.D. Thesis, Department of EE, University of Saskatchewan, Saskatoon*, 1994.
- [7] H. Kou. Mvl plas and its application in modular design. *International Symposium on Multiple-Valued Logic (ISMVL)*, pages 10–18, 5 1984.
- [8] C. McCrosky and Y. Wang. Finding frontiers without searching. *Technical Report, University of saskatchewan, Saskatoon*, 1992.
- [9] S. Peyton and C. Clack. Finding fixpoints in abstract interpretation. *Abstract Interpretation of Declarative Languages*, pages 246–265, 1987.
- [10] G. Promper and J. Armstrong. Representation of multivalued functions using the direct cover method. *IEEE Transactions on Computers*, c-30(9):674–679, 9 1981.
- [11] P. Tiruamalai and J. Butler. Analysis of minimization heuristics for mvl plas. *ISMVL-18*, pages 226–236, 5 1988.
- [12] Z. Vranesic and K. Waliazzaman. Functional transformation in simplification of multi-valued switching functions. *IEEE Transactions on Computers*, (1), 1 1972.

Procedure Optimization-using-Frontiers(f)
begin

- Test the function f for being monotonic
 - if f is not monotonic, then
 - find a permutation ρ which can transform f to a monotonic function;
 - * if ρ exists, then
 - * perform the permutation on f ;
 - * save the permutation results in f'' ;
 - * else f is not a permuted monotonic function;
 - * exist;
 - else;
 - $f'' \leftarrow f$;
 - save the permutation to ρ ;

- find all the frontiers of f'' using the MVL Frontiers algorithm;
- generate a SOP expression for f'' using Frontiers;
- output the optimized expression and the reverse permutation ρ^{-1} ;

end.

Figure 4: The proposed algorithm.

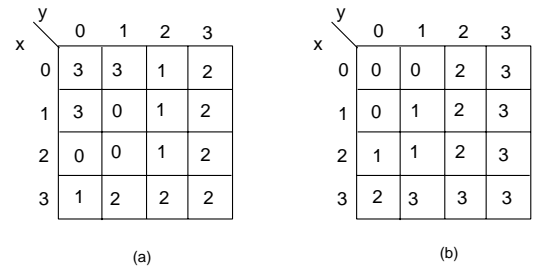


Figure 5: Example of a permuted monotonic function.