Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays

Li-Fei Wu, and Marek A. Perkowski
Department of Electrical Engineering, Portland State University
P.0. Box 751, Portland, Oregon 97207, tel. (503) 725-5411

Abstract

The new family of Field Programmable Gate Arrays, CLI6000 from Concurrent Logic Inc realizes the truly Cellular Logic. It has been mainly designed for the realization of data path architectures. However, introduced by it new universal logic cell calls also for new logic synthesis methods based on regularity of connections. In this paper we present two programs, exact and approximate, for the minimization of Permuted Reed-Muller Trees that are obtained by repetitive application of Davio expansions (Shannon expansions for EXOR gates) in all possible orders of variables in subtrees. Such trees are particularly well matched to both the realization of logic cell and connection structure of the CLI6000 device. It is shown on several standard benchmarks that the heuristic algorithm gives good quality results in much less time than the exact algorithm.

1 Introduction

There is recently an increased interest in logic synthesis using EXOR gates [1], [3] - [14]. New technologies, PLD and FPGA, either include EXOR gates, or allow to realize them in "universal logic modules". Exor circuits have smaller cost than inclusive (AND/OR) circuits. They are always very easily testable and have universal tests [5, 13]. One way to realize EXOR-based circuits is through GRMs. The *Generalized Reed-Muller (GRM)* forms, which also include Reed-Muller (RM) form [9], are canonical. All literals in the RM form are positive. In GRM forms all variables are in a fixed form, i.e. each occurrence of a variable in products of the form is either consistently positive or consistently negative. The problem of finding the minimal GRM form of optimal polarity [13] (called also fixed-polarity Reed-Muller), as well as the problem of finding the minimal Exclusive-OR Sum of Products (ESOP) of a Boolean function [1, 7, 10], are the classical ones in logic synthesis theory. Recently efficient solutions have been proposed: to ESOPs in [1], and to GRMs in [13]. However, to our knowledge, with an exception of [14] and [12] there are no any programs to minimize multi-level (more than three levels) circuits that use EXOR gates.

Recently several companies, including Concurrent Logic [2] and Algotronix brought to market a new generation of FPGAs that can be called "cellular", since the number of global connections for a cell is very limited and most of the connections are only to the abutting cells. CLI chip is a rectangular array of cells, such as one from Fig. 1. The internal structure of the cell is shown in Fig. 2. The basic cell of CLI6000 can be programmed to the *two-input multiplexer* ($A * B \oplus C * \overline{B}$) and the *AND/EXOR cell* ($A * B \oplus C$) as two of its most efficient combinational logic uses. This suggests using these cells for tree-like expansions. The method outlined in this paper is particularly suitable for these new FPGAs for which no special design methods have been yet proposed. These "cellular logic" devices require regular connection patterns in the netlists resulting from logic synthesis.

The AND/OR factorization (MIS II) creates irregular circuits that are hard to map to CLI6000 cells. ESOP factorization approach from [12] creates multi-level circuits that better utilize AND/EXOR cells but are still irregular. This makes their routing to cellular logic devices (specifically CLI6000) extremely difficult and many cells are wasted while programmed as connections only. Several concepts of logic synthesis for cellular logic using EXORs have appeared in past in the literature which may be used to minimal implementations of binary and multiple-valued logic functions [3, 8, 10]. This paper presents a new tree searching program which generates AND/EXOR tree circuits. The Permuted- Reed-Muller-Tree (PRMT) searching program presented here, *REMIT (REed-Muller Ideally permuted Trees)*, accepts a completely specified Boolean function in the form of an array of ON disjoint cubes as the input [4]. After decomposing the function with respect to all input variables according to *Davio Expansion* the program returns the exact PRM tree (the tree structure that minimizes the cost function) – in version REMIT-EXACT, or the quasi-minimum PRM tree – in version REMIT.

2 Davio Expansions

Decomposition is commonly used in logic minimization. Using Shannon and Davio Expansions, the decomposition is achieved step-by-step with respect to all input variables. This kind of decomposition can be applied always, in contrast to other types of Boolean decompositions that are applicable to only

[0] †This research was partially supported by the NSF grant MIP-9110772

some functions, and for which checking the decomposition possibility is very time consuming. The well-known *Shannon Expansion* can be applied to decomposition with an EXOR gate [3]. It is called *Davio expansion*, and is defined as follows:

$$f \;=\; x_i \cdot f_{x_i} \;\oplus\; \overline{x_i} \;\cdot\; f_{\overline{x_i}} \tag{1}$$

$$f \;=\; f_{\overline{x_i}} \;\oplus\; x_i \;\cdot\; [f_{x_i} \;\oplus\; f_{\overline{x_i}}] \tag{2}$$

$$f \;=\; f_{x_i} \;\oplus\; \overline{x_i} \;\cdot\; [f_{x_i} \;\oplus\; f_{\overline{x_i}}] \tag{3}$$

where $f_{x_i} = f(x_1, ..., x_i = 1, ..., x_n)$ and $f_{\overline{x_i}} = f(x_1, ..., x_i = 0, ..., x_n)$. Let us observe that these expansion formulas have been applied by several authors for the synthesis of GRM forms for completely specified functions [11]. Davio [3] and Green [6] use them as a base of *Kronecker Reed-Muller (KRM)*, *Pseudo-Kronecker Reed-Muller (PKRM)*, and *Quasi-Kronecker Reed-Muller (QKRM)* forms (Green uses also trees for better explanation). If only rule 2 is used repeatedly for some fixed order of expansion variables, the RM Trees are created, which correspond to RM forms after their flattening. If for every variable one uses either rule 2 or rule 3, the GRM Trees are created, from which GRMs are obtained by flattening (which proves in other way why there is 2^n of such forms). If for every variable one uses either rule 1, rule 2, or rule 3, the KRM Trees are created, from which KRMs are obtained by flattening (which proves in other way why there is 3^n of such forms).

If rules 1, 2 and 3 are used, but in each subtree there is a choice of a rule, the PKRM Trees are generated from which PKRM forms are obtained by flattening. Now, if additionally we allow the expansion variables to have various orders (but the same in the entire tree), one obtains the QKRM Trees, and QKRM flattened forms, respectively. Allowing various orders of variables in subtrees creates an even wider family of trees [10,11], called *Permuted Trees*. The trees based on expansion (2) but with different orders of variables in subtrees will be called *Permuted Reed-Muller Trees*. They include RM Trees as special cases when the same variable is used for expansion on the same level of the tree. This paper presents an efficient PRM tree searching method.

3 Tree Search to find PRM Trees

For a given Boolean function, after using Davio Expansion to decompose with respect to variable x_i, the function will be divided into two parts: $f_{\overline{x_i}}$ and $(f_{x_i} \oplus f_{\overline{x_i}})$. Both parts don't contain variable x_i any more. The parts are connected by an EXOR-gate and an AND-gate. The AND-gate takes (x_i) and $(f_{x_i} \oplus f_{\overline{x_i}})$ as its two inputs; the EXOR-gate takes $(f_{\overline{x_i}})$ and the output of the AND-gate as its two inputs. For each part $f_{\overline{x_i}}$ and $(f_{x_i} \oplus f_{\overline{x_i}})$, we use again Davio Expansion to decompose for another variable, say x_j. Both parts will be divided into another two parts. At each level of the tree the process is repeated, until all variables are decomposed. The output of the original function f is the root of the tree. At each level there can be different input variables. This kind of decomposition is ideally suited to regular array realization of combinational logic, such as one offered by "cellular logic" FPGAs. In many experiments with multi-output functions we found that the shape of a tree is not a trapezium, as it might be expected, but is closer to a rectangle. There are thus not many cells wasted for routing.

Even though every variable's decomposition is according to the same form (2), the variable decomposing order still makes the results different. The *minimum PRM Tree* is one that has the minimum regular layout-related cost (the weighted cost function takes many factors such as the total number of gates, connections, shape, etc.). The PRM tree searching program introduced here can find the best variable decomposing order and obtain the exact solution for functions of not more than 9 variables. The diagram from Fig. 3 illustrates the search tree created to find the exact PRM Tree. This tree describes all applications of Davio Expansion with respect to all possible variables in all subtrees. (1) Suppose the initial function has n variables. At the first level of the tree (the root of the tree), there is only one node, which stores the cube set of the initial function. (2) At the second level, there are $2n$ nodes. Nodes of each pair correspond to the two parts of the function from Davio Expansion. They are stored as sets of disjoint cubes. (3) To each node (cube set) at the second level Davio Expansion is further applied to decompose other variables as their parent_node did, but the number of decomposing choices is less by one than in the parent_node (because the number of choices is determined by the number of remaining variables in the cube set). (4) At the third level, there are $2n * (n-1)$ nodes. (5) Each node continues the decomposing process until all variables are decomposed (no more variable remains in the cube set). (6) In the worst case, the tree has the maximum of n levels. At the last level there is a total of $n! * 2^n$ nodes.

4 Exact PRM Tree Searching Algorithm

The goal of PRM tree searching in REMIT-EXACT is to find the *best variable decomposing order* for PRM circuits. The *best* variable decomposing order will lead to the circuit having the smallest cost. For achieving the best solution, we use full_tree searching. *Full_tree_searching* searches the whole tree to look at all possible combinations of different variable orders in all subtrees independently. Various cost functions can be declared, corresponding to costs of AND, OR, AND/EXOR gates, the numbers of connections, etc. The memory-efficient *backtracking full-tree-searching algorithm* with cut-offs based on costs is used. It generates only the solutions that are not more expensive than those previously generated. The *Decomposing_Tree* searching starts at the root, and the PRM_Tree creating at the leafs.

Although backtracking allows to discard some obviously worse solution candidates, still a large tree has to be searched.

5 Heuristic Tree Searching

The heuristic algorithm, REMIT, uses certain heuristics to select the best expansion variable for every subtree of the Decomposing tree. However, at certain levels of the tree all possible variable decomposing orders for each branch are also checked. The variable selection rules are close to those presented in [13, 10], for instance one of the heuristics states that the variable that occurs most often in disjoint cubes should be selected. The user has a number of parameters to control the depth and the type of a tree at which the full_searching is executed. For instance, he can request that when the sub-function in the Decomposing tree has 8 variables, the full_searching is executed for it.

6 Evaluation of Results

The test results shows that, regardless how many input variables or how many input cubes, the full_tree searching always returns the best solution (Table 1). I/N denotes the number of inputs, O/N – the number of outputs, #AND_gate – the number of AND gates, #EXOR_gate – the number of EXOR gates. One can see that this method is practical for less than nine input variables.

The heuristic tree searching is designed to solve larger functions. When input function has more than nine variables, the full_tree searching takes extremely long time, because the size of the tree increases exponentially with the number of inputs. The tree searching time increases then also exponentially. For example, for an eight-input function, the full_tree searching takes 200 seconds but a nine-input function may take up to eight hours. Thus, using full_tree searching on high number input functions (more than nine variables) is not practical.

Our heuristic tree searching has rationally solved this problem (Fig. 4). The result quality varies with the different initial functions. For some functions the heuristic tree searching will find the best solutions, while for others the results from the heuristic tree searching are quite different from the results of the full_tree searching. But all results fall obviously in an acceptable region of the best result (Table 2).

A comparison graph from Fig. 4 shows the searching time percentage when using the heuristic tree searching compared with using the full_tree searching. The decrease in searching time is very considerable. The graph from Fig. 5 compares results (cost for the AND-EXOR circuit). It shows the cost increasing percentage of the heuristic tree searching comparing with the full_tree searching. In can be seen from Fig. 5 that the costs given by the heuristic tree search are only slightly higher than the costs given by the full_tree searching.

After testing many functions of a high number of inputs on the heuristic tree searching (with number of inputs varying from 10 to 20), it was found that the speed of finding the results is no longer a problem. Since there are no results comparison with the full_tree searching for functions of many inputs, it is difficult to evaluate exactly how close are the results provided by the heuristic tree searching to the exact minimum results. We have, however, reasons to believe that the results of the heuristic tree searching are good, because it includes many partial full_tree searches.

7 Conclusion.

A practical program with exact and quasi-minimum options for the minimization of PRM Trees to be realized in cellular-logic FPGAs has been introduced. It is included in a comprehensive design automation system for the CLI 6000 series, that includes special state-machine and physical design programs as well. REMIT gives good results on completely specified functions. Contrary to most papers from the literature, the algorithm uses cubes, not minterms [10, 11, 13]. Other programs of the system generalize this approach to GRM Trees for incompletely specified multi-output functions [10], and to Kronecker Trees [6, 10, 11] (which use also multiplexer cells of CLI6000). Similarly as the Shannon expansion is a base of the Binary Decision Diagrams (BDDs), which are a very efficient representation for Boolean functions manipulation, the Davio Expansions are the base of our new *EXOR Decision Diagrams* which exhibit similar properties to BDDs, but reflect the AND/EXOR structure of CLI physical resources. This will be published elsewhere.

The methods shown here open a wide area of interesting and new applications, especially to cellular FPGAs, in which high regularity of connections is more important than the number of logic blocks – a requirement gives priority to structures such as trees, DAGs, and levelized decompositions with EXOR pre- and post- processors. While for other FPGAs the logic synthesis and the physical design stages are performed separately, for CLi 6000 we advocate a new design approach, which we propose to call *geometrical logic synthesis.* This approach combines the stages of logic synthesis and rough routing/placement into a *single*, comprehensive, and cellular-medium-related, stage.

References

[1] Ph.V. Besslich, *Proc. Euro ASIC'91,* Paris, 1991.

[2] Concurrent Logic Inc., "CLI6000 Series Field Programmable Gate Arrays", *Preliminary Information,* Dec. 1, 1991, Rev. 1.3.

[3] P. Davio, et al, *Discrete and Switching Functions.* George and McGraw-Hill, New York, 1978.

[4] B.J. Falkowski, I. Schaefer, and M.A. Perkowski, "Effective Computer Methods for the Calculation of Hadamard-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions," *Accepted to IEEE Trans. on CAD.*, 1991.

[5] H. Fujiwara, *Logic Testing and Design for Testability*, Computer System Series, The MIT Press, 1986.

[6] D. Green, *Int. J. Electr.*, Vol. 70, No. 2, pp. 259-280, Jan. 91.

[7] M. Helliwell, and M.A. Perkowski, *Proc. of 25th DAC* 1988, pp. 427 - 432.

[8] S.L. Hurst, *The Logical Processing of Digital Signals*, *Crane-Russak,* New York, 1978.

[9] D.E. Muller, *IRE Trans. Electron. Comp.*, Vol EC-3, pp. 6-12, Sept. 1954.

[10] M.A. Perkowski, P. Dysko, and B.J. Falkowski, *Proc. IEEE Int. Phoenix Conf. on Comp. and Comm.*, Scottsdale, AZ, pp. 606-613, March 1990.

[11] M.A. Perkowski, *Proc. of 22-nd ISMVL*, pp. 442-450, Sendai, Japan, May 27-29, 1992.

[12] M.A. Perkowski, and A. Shahjahan, "Efficient Rectangle Factorization Algorithm for ESOP Circuits", *PSU Report,* Dec. 1991.

[13] A. Sarabi, and M.A. Perkowski, pp. 30-35, *Proc. DAC'92*, June 1992.

[14] J.M. Saul, *Proc. IEEE ICCD'91*, Sept. 17-19, 191, pp. 634-637.