Automated Synthesis of Delay-Reduced Reed-Muller Universal Logic Module Networks

Shahana T. K, Rekha K. James, K. Poulose Jacob
Cochin University of Science and Technology
Kochi, Kerala, India
E-mail:{shahanatk, rekhajames, kpj}@cusat.ac.in

Sreela Sasi
Gannon University
Erie, PA, USA
sasi001@gannon.edu

Abstract

This paper presents a new approach to implement Reed-Muller Universal Logic Module (RM-ULM) networks with reduced delay and hardware for synthesizing logic functions given in Reed-Muller (RM) form. Replication of single control line RM-ULM is used as the only design unit for defining any logic function. An algorithm is proposed that does exhaustive branching to reduce the number of levels and modules required to implement any logic function in RM form. This approach attains a reduction in delay, and power over other implementations of functions having large number of variables.

1. Introduction

Every Boolean function can be expressed in terms of Reed-Muller (RM) expansion. This representation has various advantages such as ease of complementing and testing, and reduction in number of product terms leading to smaller circuits on-chip over the conventional descriptions [1]. Several papers have been published discussing design and minimization techniques for RM logic, derivation of various polarities, as well as conversion between RM and Boolean forms [4, 7, 8].

RM functions can be implemented using discrete components or more conveniently by RM-ULMs. An RM-ULM is a device with c-control inputs, 2^c data inputs and a single output f(c) and is designated as RM-ULM(c). The behavior of this module is described as

$$f(c) = b_0 \oplus b_1x_1 \oplus b_2x_2 \oplus b_3x_2x_1 \oplus \dots \oplus b_{2^c-1}x_cx_{c-1}\dots x_1$$

$$= \oplus \sum_{i=0}^{2^c-1} b_i P_i$$

where b_i = 0 or 1, and the product term (or piterm) P_i is,

$$P_i = x_{i_c} x_{i_{c-1}} \dots x_{i_1} \qquad \text{where } i = \sum_{j=0}^{c-1} 2^j x_j$$

x_{i_K} will be present in P_i if the kth bit of binary representation for i is 1. The logic symbol for RM-ULM(c) is shown in Figure 1.

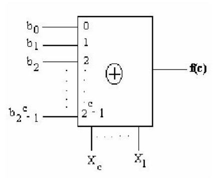

Figure 1. Logic symbol of RM-ULM(c)

VLSI implementations using only one type of modular building blocks can decrease system design and manufacturing cost. For functions in RM form speed and cost can be reduced by using RM-ULMs connected in tree structure. A tree network is very suitable for VLSI realization because of the uniform interconnection structure and the repeated use of identical modules.

The use of RM-ULM for realization of logic functions has already been explored by researchers. A programmed algorithm was developed by L. Xu [2], which is analogous to the algorithm in [6], for optimization of number of modules at sub-system level in a tree network. The algorithm looks for possible cascade networks, and if it is not found a tree structure is implemented. An alternate algorithm was presented by E. C. Tan [3], which performs similar optimization of fixed polarity general Reed-Muller expansions (FPGRM) with a reduced computation time. The above algorithms do not explore all the possible branching options of the tree structure and hence the delay of the circuit synthesized may not be minimal.
In this research, further delay reduction is achieved by using a novel tree-structured exhaustive branching network using RM-ULM(1) for implementing a logic function given in positive polarity Reed-Muller (PPRM) form. A logic function with n-variables can be implemented using 2^n-1 RM-ULM(1)s in n - levels by standard implementation. Any implementation using less than 2^n-1 number of modules and / or lesser number of levels can be considered as an improvement in cost and / or speed.

The organization of this paper is as follows: First, the problem is described and then the proposed exhaustive branching algorithm is demonstrated with examples. Finally, a comparison in terms of delay and number of modules is done for standard implementation and tree implementation [2, 3] for several functions.

2. N-ary exhaustive branching technique

For a given number of input variables n, there is a well-defined number of functions, which is equal to 2^(2^n) [5]. Standard implementation of a tree network requires n-levels to implement these functions. Xu proposed a programmed algorithm to reduce the complexity of the network in terms of number of modules and levels. In his approach 1's, 0's, $x^{\bullet}_i$ (where $x^{\bullet}_i$ is a variable x_i or its complement x_i', $1 \le i \le n$) or functions using any number of variables can be given to any data inputs of the RM-ULM. But the control inputs accept variables only. In this research, the performance is further improved by an exhaustive branching technique with $x^{\bullet}_i$ or functions of 2 variables at control input. Since x_i' or functions are also given to control input, the utilization of all branching options are made possible. This decreases the number of levels, and hence the delay is reduced for any logic function implementation using RM-ULMs.

The first level (output stage) will have a single RM-ULM, the second level will have a maximum of $(2^c + c)$ RM-ULMs, where c is the number of control inputs (c = 1 in this case) and so on. In general, the maximum number of RM-ULMs in a level can be expressed as $(2^c + c)^{(L-1)}$ where L indicates the number of levels. The maximum number of RM-ULMs in the complete network having L levels will be

$$\sum_{x=1}^{L} (2^c + c)^{(x-1)}$$

A network with 1-level can realize functions up to 3 variables, since there are 3 inputs. By connecting x_i to the control input, the remaining $x^{\bullet}_i$ variables ($j \neq i$) or constants (0 or 1) can be connected to each of the 2 data input lines. So there are 6 possible values for each data input line, resulting in 6^2 distinct functions. Selecting 1 variable as control input from the total of 3 variables and its complements, can take $6C_1$ combinations. Out of 62 distinct functions implemented at level 1, 24 are 3 variable functions which require 3 levels in standard implementation.

Level 2 allows the implementation of functions having maximum of 9 variables, using 3 control lines and 6 data lines. Selecting 3 variables from the total of 9 variables and its complements, results in $18C_3$ x 3! combinations. The remaining 6 variables and its complements or constants (0 or 1) at 6 data lines give rise to 14^6 distinct functions with one combination at control input. In the tree structure given by Xu [3], at level 2, maximum number of variables possible is only 7, which result in 10^4 distinct functions with one combination at control input. The proposed approach increases the number of variables and functions that can be implemented in level 2. As the number of levels increases this difference becomes more and more significant, and more delay reduction can be achieved for functions with large number of variables. In general with L levels, the number of functions that can be implemented using RM-ULMs in the exhaustive branching method is $\{ [2 (y + 1)]^y \}$ for one combination at control inputs. The number of combinations possible at the control inputs is $\{ 2 z^L \mathbf{C} z^{L-1} \} \times \{ z^{L-1}! \}$ where $y = [z^L - z]$ and $z = 2^c + c$ Maximum number of variables at level L is $n_{max} = z^L$ For a given function if there are n dependent variables, the levels L required for implementation in this approach is given as $\lceil \log_{(2^c + c)} n \rceil \le L \le (n - 1)$, whereas in the tree structure, L can be in the range $\lceil \log_{(2^c)} n \rceil \le L \le (n - 1)$

This clearly demonstrates a reduction in delay attained by the proposed exhaustive branching technique over the implementation using tree structure.

A. Exhaustive Branching Algorithm

Behavior of an RM-ULM(1) can be expressed as $F_j \oplus F_s F_k$, where F_s, Fj, F_k are functions of t variables ($1 \le t \le n$). The number of variables of F_s, Fj and F_k varies according to the complexity of the function to be realized. The maximum number of variables in F_s, Fj or Fk determines the delay of the network. The network terminates when F_s, Fj and F_k are 1's, 0's or $x^{\bullet}_i$ ($1 \le i \le n$). If all inputs except one terminate with a variable $x^{\bullet}_i$ or a logical constant and only one input continues into the next level, a cascade is generated where a single module is used in each level. The proposed algorithm aims to identify $x^{\bullet}_i$ or functions of 2 variables at each control input, that eliminate as many branches as possible and reduce the number of levels and modules required for implementation. The algorithm for any function given piterms (P_i), is as follows:

Exhaustive Branching Algorithm:

Step 1: Get the piterms in decimal, and the number of variables, n. Set level, L = 1, number of modules, M = 1 Step 2: List the piterms in n-bit binary as a piterm table. Step 3: Check whether any column in the table is all zeros. Eliminate the variable corresponding to that column and get the reduced piterm table.

Step 4: Get the reduced piterm tables for each variable x_i (one table for $x_i = 1$ and another table for $x_i = 0$) and find the x_i for which the reduced piterm tables correspond to constants (0 or 1) or $x^{\bullet}_i$ ($j \neq i$) by checking the number of ones in each piterm table, c_1. If $c_1 \le 1$, terminate. Step 5: For each x_i check the following conditions:

(i) Number of zeros ≥ number of ones
(ii) For each (1, 0) pair, the remaining bits are constants
(iii) Number of such pairs is equal to 2
(iv) One pair has remaining bits as all zeros and the other has ones in one column only

Terminate if all the above conditions are satisfied. Step 6: L = L + 1, M = M + 1. Get the reduced piterm tables for each variable and find the x_i for which the following conditions are satisfied.

(i) One reduced piterm table corresponds to a constant (0 or 1) or $x^{\bullet}_i$ ($j \neq i$).
(ii) The other reduced piterm table is a single module implementation by repeating the steps 4 & 5.

Step 7: Get reduced piterm tables for each possible $1 \oplus x_i$ (by checking conditions (i) & (ii) of step 5), and find

the (1⊕ x_i) for which the conditions (i) & (ii) of step 6 are satisfied.
Step 8: M = M + 1. Get the reduced piterm tables for each variable, and find the x_i for which the reduced piterm tables are single module implementations by repeating the steps 4 & 5.
Step 9: Get reduced piterm tables for each possible 1⊕ x_i (by checking conditions (i) & (ii) of step 5), and find that 1⊕ x_i for which the reduced piterm tables are single module implementations by repeating the steps 4 & 5.
Step 10: Get the reduced piterm tables for each possible x_ix_j, and find the x_ix_j for which the conditions (i) & (ii) of step 6 are satisfied.
Step 11: Get the reduced piterm tables for each possible (x_i ⊕ x_j), and find the (x_i ⊕ x_j) for which the conditions (i) & (ii) of step 6 are satisfied.
Step 12: M = M + 1. Get the reduced piterm tables for each possible x_ix_j, and find the x_ix_j for which both reduced piterm tables are single module implementations by repeating the steps 4 & 5.
Step 13: Get the reduced piterm tables for each possible (x_i ⊕ x_j), and find the (x_i ⊕ x_j) for which both reduced piterm tables are single module implementations by repeating the steps 4 & 5.

The exhaustive branching algorithm is demonstrated in the following examples.

Example 1:
Implementation of 4-variable function *F* = ⊕ ∑ (13, 14)

The delivered network has 3 modules using only 2 levels in the proposed approach, as shown in the Figure 2, while in the tree implementation [2, 3] the synthesized network will have 3 modules in 3 levels as shown in Figure 3.

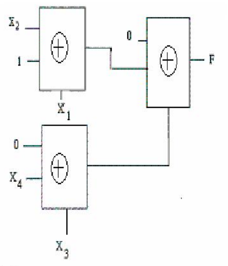

Figure 2.
Exhaustive branched implementation for
F =⊕ ∑ (13, 14)

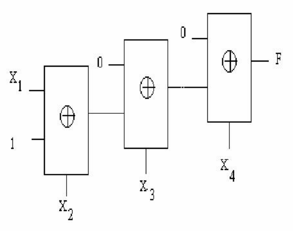

Figure 3.
Tree implementation for function
F =⊕ ∑ (13, 14)

Example 2:
Implementation of the 4-variable function, F = ⊕ ∑ (5, 6, 9, 10) has 3 modules using only 2 levels in this approach, as shown in Figure 4, while the tree implementation [2, 3] will have 4 modules in 3 levels or a minimum of 3 modules in 3 levels as shown in Figure 5. In the above two examples a reduction in delay is found using the proposed approach.

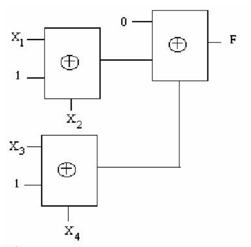

Figure 4.
Exhaustive branched implementation for
F =⊕ ∑ (5, 6, 9, 10)

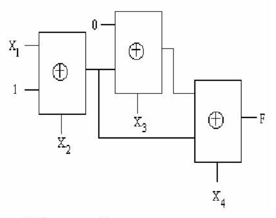

Figure 5.
Tree implementation for function
F =⊕ ∑ (5, 6, 9, 10)

Example 3:
Implementation of the 3-variable function, F = ⊕ ∑ (0, 1, 2, 4, 6) is given in Figure 6.

The delivered network has only 1 module using 1 level in the proposed approach, whereas the tree implementation [2, 3] requires 2 modules in 2 levels. One possible implementation is shown in Figure 7. This example clearly indicates the reduction in delay and number of modules.

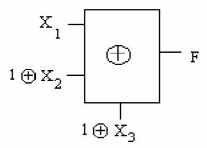

Figure 6.
Exhaustive branched implementation for
F =⊕ ∑ (0, 1, 2 ,4, 6)

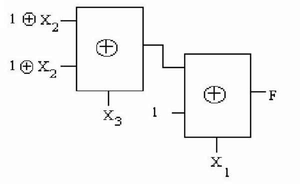

Figure 7.
Tree implementation for function
F =⊕ ∑ (0, 1, 2 ,4, 6)

Example 4:
Implementation of the 3-variable function, F = ⊕ ∑ (0, 2, 3, 4, 5) is given in Figure 8. The delivered network has only 2 modules using 2 levels in the proposed approach, while the tree implementation [2, 3] requires 3 modules in 2 levels as shown in Figure 9.

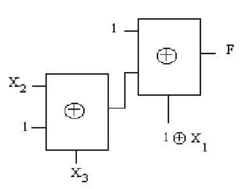

Figure 8.
Exhaustive branched implementation for
F =⊕ ∑ (0, 2, 3, 4, 5)

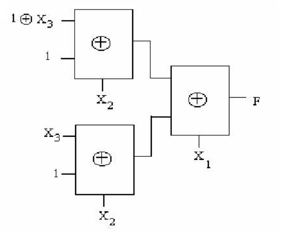

Figure 9.
Tree implementation for function
F =⊕ ∑ (0, 2, 3, 4, 5)

Simulation is done for 2, 3 and 4-variable functions up to 2 levels. Table 1 shows the reduction in delay and / or hardware for certain functions. The reduction in number of modules required will lead to reduced power consumption.

Table 1. Comparison in terms of delay and hardware for standard implementation, tree implementation, and exhaustive branched implementation

Functions	Standard Implementation D / M	Tree Implementation D / M	Exhaustive Branched Implementation D / M
F= ⊕Σ(13, 14)	4 / 15	3 / 3	2 / 3
F= ⊕Σ(5, 6, 9, 10)	4 / 15	3 / 3	2 / 3
F= ⊕Σ(0, 1, 2, 4, 6)	3 / 7	2 / 2	1 / 1
F= ⊕Σ(0, 2, 3, 4, 5)	3 / 7	2 / 3	2 / 2

D / M – Delay (number of levels) / Number of modules

3. Conclusion and future work

An algorithm for the synthesis of RM-ULM network with reduced delay is presented. The delivered network has reduction in delay and complexity in terms of number of modules, compared to the existing implementations. By suitable selection of variables, its complements or functions as control inputs, the number of modules and delay are reduced. The reduction in number of modules results in reduced power consumption of the synthesized network. Theoretically, the algorithm can handle any number of variables for any completely specified logic function. The computation time is not always directly proportional to the number of variables, but this increases with the complexity of the function to be realized. Since the topology of the delivered network is that of a tree, VLSI implementation of this network requires very few extra work in routing algorithms to redesign or for circuit layout.

This algorithm can be explored for the synthesis of incompletely specified functions in future. Network complexity can be reduced if the RM-ULM considered have normal and complemented outputs. Research may be done to consider different size RM-ULM for an alternative implementation.

REFERENCES

[1] B. Harking, "Efficient algorithm for canonical Reed-Muller expansions of Boolean functions", IEE Proceedings-E, Vol. 137, No. 5, September 1990.

[2] L. Xu, A.E.A. Almaini and J.F. Miller, L. McKenzie, "Reed-Muller universal logic module networks", IEE Proceedings-E, Vol. 140, No. 2, March 1993, pp. 105-108.

[3] E.C. Tan and C.Y. Chia, "Alternative algorithm for optimization of Reed-Muller universal logic module networks", IEE Proceedings Computers and digital techniques, Vol. 143, No. 6, November 1996, pp. 385-390

[4] T. Sasao, "Logic Synthesis with EXOR gates", *"Logic Synthesis and Optimization"*, edited by T. Sasao, Kluwer Academic Publishers, London, ISBN: 0-7923-9308-2, 1993.

[5] V.P. Correia and A. I. Reis, "Classifying n - input Boolean functions", VII Workshop IBERCHIP 2001, Montevideo, IWS 2001, pp. 58.

[6] A.E.A. Almaini, J.F. Miller and L Xu, "Automated synthesis of digital multiplexer networks", IEE Proceedings-E, Vol. 139, No. 4, July 1992, pp. 329-334.

[7] J.F. Miller and P. Thomson, "Combinational and sequential logic optimization using Genetic Algorithms" Genetic Algorithms in Engineering Systems: Innovations and Applications 12-14 September 1995, Conference Publication No. 414, 1995 pp. 34-38.

[8] D.Varma, E.A.Trachtenberg, "Computation of Reed Muller expansions of incompletely specified Boolean functions from reduced representations", IEE Proceedings-E, Vol. 138, No. 2, March 1991, pp. 85-92.