Synthesis of Multi-Level Dual Reed-Muller Forms

KHALID FARAJ
Computer Science
Wajdi Institute of Technology
Jerusalem, Mount of Olives. P.O. Box 19014
Palestine
kfaraj@wit.edu.ps http://www.wit.edu.ps

Abstract: - This paper introduces the fundamental theory, algorithms, and terminology regarding synthesis of multi-level Dual Reed-Muller expressions. The increasing interest in using Dual Reed-Muller expressions as a way of representing and manipulating switching functions, and as a mean of designing circuits based on OR/XNOR gates has led to this research. Up to present there are only two-level Dual Reed-Muller minimization algorithms in use, although the need for multi-level minimization algorithms has been recognized. A new theory and algorithms for multi-level Dual Reed-Muller minimization have been developed. It introduces a Dual Reed-Muller factored form and uses algebraic algorithms for factorization decomposition, re-substitution, and extraction of common cubes and sub-expressions.

Key-Words: - Multi-level Dual Reed-Muller forms, factorization, decomposition.

1 Introduction

The increasing complexity of chip designs and the continuous development of smaller size fabrication processes present new challenges to the existing tools. Future synthesis tools are required to handle millions of gates in a realistic time. Computer-Aided Design (CAD) tools became critical for design and verification of Very Large Scale Integrated (VLSI) digital circuits. Since most of the research has focused on developing algorithms for AND/OR or NAND/NOR circuits. An alternative description of a Boolean function is Reed-Muller expansion [1, 2]. It employs modulo-2 arithmetic and is also unique and canonical for a given Boolean function. The application of XOR/AND and XNOR/OR gates has some advantages over other implementations. In practice, it is well known that many useful circuits such as arithmetic units and parity checkers are heavily XOR oriented and it is more economical to implement their modulo-2 expressions [3-6]. Some authors [7, 8] even conjecture that it is generally more economical to base logic design on modulo-2 expressions rather than conventional OR expressions. Recent progress in circuit technology makes the use of OR/XNOR gates feasible, especially with the development of the new technologies and the arrival of various programmable gate array (FPGA) devices. A major other characteristic of the XNOR logic is the numerous possible canonical representations of switching functions it provides.

There are several kinds of OR/XNOR circuits. The FPDRM is one of the canonical OR/ XNOR expressions. FPDRMs are a generalization of Positive Polarity Reed-Muller expressions (PPDRM). A PPDRM is unique for a completely specified function, is an OR/XNOR expressions with only un-complemented (positive) literals. Each variable in the FPDRM can appear either in un-complemented or complemented form but not both. For an n-variable completely specified Boolean function there are 2^n distinct FPDRMs. There are techniques for converting from POSs to PPDRM or FPDRM [9-13].

In the domain of combinational logic synthesis, logic minimization plays a vital role in determining the area and performance of the synthesized circuit. Logic minimization based on product of sum (POS) using OR-AND gates is a well studied area. However, minimization based on fixed polarity Dual Reed-Muller (FPDRM) using OR-XNOR gates has received relatively lesser attention. The problem of finding a FPDRM of the given Boolean function with the minimum number of cubes has both theoretical and practical value. From the theoretical point of view OR-XNOR is the most general Dual Reed-Muller form. From the practical point of view, XNOR gates and OR-XNOR have numerous applications in logic synthesis. In particular, it has been shown that the OR-XNOR representation of Boolean functions is typically more compact than the POS representation. Multilevel realizations are the preferred means of implementing combinational circuits in very large scale integrated (VLSI) systems today. Because of the increased potential for reusing subcircuits, there are more degrees of freedom in implementing boolean function than in two level programmable logic array (PLA) case. The goal of mutlilevel logic optimization is to obtain a representation of the Boolean function that is optimal with respect to area, speed, testability, and power dissipation. There has been recent interest in using Dual Reed-Muller expressions as a way of

representing and manipulating switching functions, and as a means of designing circuits based on XNOR gates. Analysis of existing algorithms suggests the need for work in two areas: improving two-level minimization algorithms; and developing algorithms for multi-level minimization. A new algorithm for the two-level minimization of Dual Reed-Muller representations of switching functions has been developed [14].
A procedure for multi-level Dual Reed-Muller minimization has been developed which introduces a Dual Reed-Muller factored form, and uses algebraic algorithms for factorization, decomposition, re-substitution, and extraction of common cubes and sub-expressions.

2 Preliminaries

In this section, essential definitions and notations are presented. These are important for the understanding of the paper.
Definitions: A rectangle (R,C) of a matrix B, $B_{ij} \in \{0,1,*\}$ is a subset of rows R and subset of columns C such that $B_{ij} \in \{1,*\}$ for all I ∈ R and I ∈ C. A rectangle (R, C) of B is said to be a prime rectangle if it is not strictly contained in any other rectangle of B. The co-rectangle of a rectangle (R,C) is the pair (R,C′) where C′ is the set of columns not in C.
For example:
f = (a + b + e) ⊗ (a + c + d) ⊗ (b + c + d)

	A 1	B 2	C 3	D 4	E 5
a + b + e 1	1	1	0	0	1
a + c + d 2	1	0	1	1	0
b + c + d 3	0	1	1	1	0

({2,3},{3,4}) is a prime rectangle, because it is not contained in any other rectangle.
A **literal** is a Boolean variable in positive or negative polarity.
A **cube** *C* is a sum term composed of literals using Boolean OR operation.
A Dual Reed-Muller is minimum if it contains the minimum number of cubes among all the OR- XNOR of the given Boolean function.
A **variable** in the cube can have three forms: (1) positive polarity; (2) negative polarity; (3) don't-care.
The fundamental properties of the XNOR operation are given as follows:

$$x \otimes \bar{x} = 0 \quad (1)$$
$$x \otimes x = 1 \quad (2)$$
$$x \otimes 0 = \bar{x} \quad (3)$$
$$x \otimes 1 = x \quad (4)$$

The Distributive Law: This law shows that combinations of OR and XNOR can be written in an expanded form [14].

$$(A + (B \otimes C) = (A + B) \otimes (A + C) \quad (5)$$

Definition 2.1 An *n*-variable Boolean function can be expressed as

$$f(x_{n-1}, x_{n-2}, ..., x_0) = \prod_{i=0}^{2^n-1} (d_i + M_i) \quad (6)$$

Where '∏' represents logical products (AND), the '+' is an OR operation and *I* is a binary *n*-tuple $I = [i_0, i_1, \ldots, i_{n-1}]_2$, $[d_0, d_1, \ldots, d_{2^n-1}]$ is the truth vector of the function f, $d_i \in \{0,1\}$ [12], M_i is a sum term

$$M_i = \prod_{k=n-1}^{0} x_k^{\bullet i_k} = x_{n-1}^{\bullet i_{n-1}} + x_{n-2}^{\bullet i_{n-2}} + \ldots + x_0^{\bullet i_0}$$

and

$$x_k^{\bullet i_k} = \begin{cases} x_k & i_k = 0 \\ \bar{x}_k & i_k = 1 \end{cases}$$

Alternatively, any Boolean function can be represented by a FPDRM expression as:

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) = \bigotimes_{i=0}^{2^n-1} (c_i + S_i) \quad (7)$$

Where '⊗' is XNOR operator, $[c_{2^n-1}, c_{2^n-2}, \ldots, c_0]$ is the truth vector of the function f, $c_i \in \{0,1\}$, $i = [i_0, i_1, \ldots, i_{n-1}]_2$, S_i represents a Sum term as

$$S_i = \prod_{k=n-1}^{0} x_k^{\sim i_k} = x_{n-1}^{\sim i_{n-1}} + x_{n-2}^{\sim i_{n-2}} + \ldots + x_0^{\sim i_0}, \quad (8)$$

and

$$x_k^{\sim i_k} = \begin{cases} 0 & i_k = 0 \\ x_k & i_k = 1 \end{cases}$$

Definition 2.2 A polarity vector $(p_{n-1}, p_{n-2}, \ldots, p_0)$ for a FPDRM of an *n*-variable Boolean function is a binary vector with *n* elements, where $p_i = 0$ indicates the variable x_i in an un-complemented form (x_i), while $p_i = 1$ indicates the variable x_i in the complemented form.
Property 1 For an *n*-variable Boolean function, there are 2^n FPDRM expansions corresponding to 2^n different polarity numbers. Each of such expansions is a canonical representation of a completely specified Boolean function.
Maxterms can be identified by expanding a Kronecker sum of *n* basis vectors of the form $[0\ \bar{x}_i]$ for '0' polarity

and [0 x_i] for '1' polarity .
The FPDRM can be deduced by substituting the coefficient vector **c** as shown in equation (9).

$$f(x_{n-1},x_{n-2},.,x_0)=\{[0\ x_{n-1}]*[0\ x_{n-2}]*\ldots*[0\ x_0]\}\square\mathbf{c} \qquad (9)$$

Where '□' represents matrix multiplication based on OR and XNOR [9-13].
For example for a zero polarity (p = 0), with n = 2 the basis vector is generated as follows:

$$[0\ x_1]*[0\ x_0] = [0+0\ \ 0+x_0\ \ x_1+0\ \ x_1+x_0]$$

In order to restructure the Dual Reed Muller expressions we introduce the following operations and algorithms to minimize the multi level Dual reed Muller expressions

3 Multi-level minimization operations

The objective of multi-level optimization is to find a good multi-level representation of a switching function that minimize a cost function with respect to area, speed, testability, and power dissipation. In order to reorganize a switching function the operations described in the sequel are used.

3.1 Decomposition

Decomposition of a switching function is the process of re-expressing a single function as a collection of new functions [15]. For example,

F = (a + b + c) ⊗ (a + b +d) ⊗ (a′ + c′ + d′) ⊗ (b′ + c′ + d′)
To the following set of expressions:
F = (c ⊗ d) + X ⊗ (a′ ⊗ b′) + Y
X = (a + b)
Y = (c′ + d′)

3.2 Extraction

Extraction operation is similar to decomposition. It is the process of identifying, creating some intermediate functions and variables, and re-expressing the original switching functions in terms of the original as well as the intermediate variables. This creates new multiple fan-out nodes. The extraction process thus identifies common sub-functions among different switching functions forming a network. For example,

F = (a + c + d) ⊗ (b + c + d) ⊗ e
G = (a + e′) ⊗ (b + e′)
H = (c + d +e)
yields

F = (X + Y) ⊗ e
G = X + e′
H = Y + e
X = (a ⊗ b)
Y = (c + d)

3.3 Factorization

Factorization is the process of deriving the Dual Reed-Muller factored form the two level form of the expression. For example,

F = (a + c) ⊗ (a + d) ⊗ (b + c) ⊗ (b + d) ⊗ e
F = [(a ⊗ b) + c] ⊗ [(a ⊗ b) + d] ⊗ e
F = (a ⊗ b) + (c ⊗ d) ⊗ e

4 Algebraic division

A key idea behind multilevel minimization process is the concept of division. This is the process that, given Dual Reed Muller expressions of functions f and p, finds quotient q and reminder r so that $f = (p+q) \otimes r$. If p and q are disjoint then the operation is algebraic.
A division operation for sum of products representation was described in [16]. Because this algorithm is algebraic it can be applied to Dual reed Muller expressions. The following is a sketch of the algorithm for carrying out algebraic division: given f and p it returns the quotient q and remainder r:

Algebraic Division (f,p)

U = restriction of f to the literals in p
V = restriction of f to the literals not in p
/* not that $u_j v_j$ is the jth term of f */
Vi = { $v_j \in V \mid u_j = p_i$}
q = $\cap V_i$
$r = f - (p+q)$
Return (q,r)
Figure 1: Algebraic division algorithm

For example:
F = (a +d) ⊗ (b + c) ⊗ (b + d) ⊗ e
P = (a ⊗ b)

Then
U = a ⊗ b ⊗ b ⊗ 0
V = d ⊗ c ⊗ d ⊗ e
V1 = d
V2 = c ⊗ d
q = $\cap V_i$ = d
p + q = [(a ⊗ b)] + d = (a + d) ⊗ (b + d)
r = f – (p + q)
r = (a +d) ⊗ (b + c) ⊗ (b + d) ⊗ e –[(a + d) ⊗ (b + d)]
r = (b + c) ⊗ e

Therefore,
F = (p + q) ⊗ r

4.1 Kernels and algebraic divisors

Given an efficient method for algebraic division, optimization can be carried out if good algebraic divisors can be found.

Kernels

The idea of the kernel of an algebraic expression is used to find good divisors and sub-expressions common to two or more expressions [16, 17]. An expression t is a common sub-expression of f and g if f and can be written as:
F = (q1 + t) ⊗ r1
And
G = (q2 + t) ⊗ r2
where q1 and q2 are nonzero.

The notation of a kernel of a logic expression was introduced to provide an efficient m*eans* for finding common sub-expressions. We see that kernels are a bridge between algebraic expressions and factored form. A kernel of an expression f is defined by the following two rules:

1) A kernel k of an expression f is the quotient of f and a cube c; k= f/c.
2) A kernel k is "cube-free" (k cannot be written as k = dg, where d is a nontrivial cube and g is an expression.)

For example, suppose that
F = (a + b + c + d + e) ⊗ (a + b + c + d + f) ⊗ (a + b + c + g).

		a 1	b 2	c 3	d 4	e 5	f 6	g 7
a + b + c	1	1	1	1	.	.	.	.
a + b + d	2	1	1	.	1	.	.	.
e + g	3	.	.	.	.	1		1
a + b + f + g	4	1	1	.	.	.	1	1
b + d	5	.	1	.	1	.	.	.
e + f	6	.	.	.	.	1	1	.

Then F/a = (b + c + d + e) ⊗ (b + c + d + f) ⊗ (b + c + g) is a primary divisor, but not a kernel, because b divides each term. The kernels are the quotient of f and the cube a, but it is not cube-free since the cube b is a factor of f/a,

F/a = [(b + c + d + e) ⊗ (b + c + d + f) ⊗ (b + c + g).

However, F/(a+b) = [(c + d + e) ⊗ (c + d + f) ⊗ (c + g) is cube free and hence a kernel.

4.2 Computing the kernels

It is possible to find kernels of an expression by looking at the intersection of the cubes of a function. If a subset of the cubes intersect then the cube corresponding to this intersection is a co-kernel, therefore, the kernel can be found by dividing the co-kernel into F. this intersection can be found by finding the prime rectangles of a matrix formed from the cubes and the literals of an expression [17]. This algorithm can be applied directly to dual Reed-Muller expressions.

		a 1	b 2	c 3	d 4	e 5	f 6	g 7
a + b + c + d + e	1	1	1	1	1	1	.	.
a + b + c + d + f	2	1	1	1	1	.	1	.
a + b + c + g	3	1	1	1	.	.	.	1

Each row corresponds to a cube of the function, while each column corresponds to a literal in the set of cubes. A position (i,j) is set to 1 if cube c_i contains literal j. For instance the prime rectangle {{1,2,3},{1,2,3}} which gives co-kernel (a + b + c).

4.3 Common kernel extraction

Common kernel extraction is the process of finding sub-expressions that appear in two or more expressions, then extracting the sub-expression to simplify those expressions. To optimize the switching functions it is necessary to find the particular kernel-cube, which is common to two or more functions.

4.4 Common cube extraction Kernels

Common cube extraction is the process of finding cubes common to two or more expressions and extracting the common cube to simplify each of the expressions. The process of extracting a cube adds a new node the network with a logic function which is the common cube divisor.

For example,
F = (a + b + c) ⊗ (a + b + d) ⊗ (e + g)
G = a + b + f + g
H = (b + d) ⊗ (e + f)
Thus the cube literal matrix is
The rectangle ({1,2,4},{1,2}) corresponds to a common cube (a + b). If this common cube is extracted a new node is added with a function representing the common cube.

$F = X + (c \otimes d) \otimes (e + g)$
$G = X + f + g$
$H = (b + d)\ V\otimes\ (e + f)$

5 Conclusion

We introduced the fundamental foundations, and theory for minimization of the multi-level Dual Reed-Muller expressions. The process uses algebraic algorithms for factorization, decomposition, re-substitution, and extraction of common cubes and sub-expressions.
This algorithms have been tested on some random functions.

References:

[1] Reed, I.S., A claa of multiple-error-correction codes and their decoding schem. IRE trans. Inform. Theory, IT-4, 1954, pp. 38-49.
[2] Muller, D.E. Sept., Application of Boolean algebra to switching circuit design for error detection. IEEE trans. Comput, EC-3, 1954, pp. 6-12.
[3] Jeong, B.k., Sung, J.H., and Jong, K., New circuit for XOR and XNOR functions. International Journal of Electronics, 82, 1997, pp. 131-143.
[4] McCluskey, E., Logic Design Principles with Emphasis on Testable Semicustom Circuits. (Prentice Hall). 1986.
[5] Wang, J.M., Fang, S. C., and Feng, W.S., New efficient design for XOR and XNOR functions on the transistor level. IEEE Journal of solid-state Circuits, 29, 1994, pp. 780-786.
[6] Sasao, T.,Easily Testable Realizations for Generalized Reed-Muller Expressions. IEEE Transaction on computers, 46, 1997, pp. 709-716.
[7] Robinson, J.P. and Yeh, C.L. Aug., A method for modulo-2 minimization. IEEE Trans. Comput, C-31, 1982, pp. 800-801.
[8] Sassao, T and Besslich, P. Feb., On the complexity of Mod-2 sum PLA's. IEEE Trans. Comput. C-39, 1990, pp. 262-266.
[9] Cheng, J., Chen, X., Faraj, K.M., and Almaini, A.E.A., Expansion of logical functions in the OR-coincidence system and the transform between it and maxterm. IEE Proc.-comput. Digit. Tech, 150, 2003, pp. 397-402.
[10] Green, D.H., Dual forms of Reed-Muller expansions. IEE Proc.-Comput. Digit. Tech, 141, 1994, pp. 184-192.
[11] Faraj, K., MacCallum, M., and Almaini, A.E.A., Fast computation of Conjunctive Canonical Reed-Muller functions. PREP Proceeding, University of Hertfordshire, 2004, 144.
[12] Faraj, K., Almaini, A.E.A.,Minimization of Dual Reed-Muller Forms using Dual Property. WSEAS TRANSACTIONS on CIRCUITS and SYSTEMS, Issue 1, Vol. 6, January 2007, pp. 9-15.
[13] Faraj, K., Almaini, A.E.A., Optimal Expression for Fixed Polarity Dual Reed-Muller Forms. WSEAS TRANSACTIONS on CIRCUITS and SYSTEMS, Issue 3, Vol. 6, March 2007.
[14] Faraj, K., Minimization of OR-XNOR Expressions Using Four New Linking Rules. Advances on artificial intelligence, knowledge Engineering and data bases.rack, February 2008, pp. 489-494
[15] S. Devadas, A. Ghosh, and K. Keutzer, Logic Synthesis,McGraw-Hill, Inc.1994.
[16] Brayton, R. K., Factoring logic functions., IBM J. RES. DEVELOP. VOL. 31 NO. 2, March 1987, pp 187-198.
[17] Jonathan Saul, An algorithm for the Multi-level minimization of Reed-Muller representations, IEEE, 1991, pp. 634-637.