LOGICAL SYNTHESIS FOR EFFICIENT CMOS TRANSISTOR NETWORK

[1]Dionatan de S. Moura, [1]Caio G. P. Alegretti, [1]Leomar S. Rosa Jr.
[2]André I. Reis, [1]Renato P. Ribas

[1]Instituto de Informática – UFRGS, Porto Alegre, Brazil
[2]Nangate Inc., Menlo Park, CA, Herlev, Denmark
{dsmoura, caio, leomarjr, rpribas@inf.ufrgs.br}, are@nangate.com

ABSTRACT

This paper presents the Boolean function synthesis in different approaches, including the Quine-McCluskey sum-of-products and product-of-sums, as well as mux-based approach. A CAD tool has been developed in Java platform to provide such logic equations, useful for transistor network generation at cell level.

1. INTRODUCTION

Chip area, delay time, and power dissipation in digital VLSI design depend on the number of transistors and on the topological structure that is used for the logic gates involved. Several state-of-the-art works investigate how circuit optimizations at transistor level can lead to the best network implementation in order to deliver the most optimized and efficient logic cells structures. Some of them are based on Boolean equation treatment, where algebraic and Boolean optimizations are performed in a given logic expression in order to get a reduced and efficient derived transistor network [1-2]. Other ones are based on graph optimizations, where the graph structure presents a close relation between the logic function description and the derived network [3-6].

Transistor networks can be implemented in different logic styles. In this context, statis CMOS and PTL (Pass Transistor Logic) styles are the most used for digital VLSI designs. Examples of CMOS-like logic style, with PMOs pull-up and NMOS pull-down planes, are CSP (Complementary Series-Parallel, also called as CMOS in the literature) [7], NCSP (Non-Complementary Series-Parallel) [2], and LBBDD (Lower Bound Network derived from BDDs) [8].

In this work, the problem of achieving a minimal logic equation to implement CMOS-like transistor networks is discussed. The Quine-McCluskey method [9] is reviewed and some optimization points are investigated and considered. Also, a MUX-based transistor network topology and a methodology to generate it are presented.

The main contributions of this work are: (1) to investigate and to review the Quine-McCluskey method with the purpose of pointing some criteria that impact on the derived transistor networks; (2) to present an alternative CMOS-like network and a methodology to automatically derive that from logic equations.

The remaining of this paper is organized as follows: Section 2 discusses the Quine-McCluskey method and the impact of the final solution on the transistor network, while Section 3 presents the MUX-based topology and the methodology to generate it. Section 4 presents a brief discussion about factorization vs. branch-based implementations. Finally, Section 5 presents the conclusions.

2. QUINE-MCCLUSKEY OPTIMIZATION

The algorithm of Quine-McCluskey is a method that minimizes a binary function completely with any number of variables. The binary function should be in canonical form so that it is represented completely. The canonical form of a binary function is the sum of all of the minterms, in the case of a sum-of-products (SOP), or it is the product of all of the maxterms, in the case of a product-of-sums (POS). Minterms and maxterms are generically called terms, because the algorithm minimizes POS or SOP.

2.1. Definitions

A literal is an instance of a variable of the function. A product of literal is an AND operation among literal, or a literal alone. A sum of literal is an operation OR among literals or a literal alone. By default, a minterm is a product of literals that belongs to the on-set of the function. In the same way, a maxterm is a sum of literals that belong to the off-set of the function. A POS_0 is a product of the maxterms in 0, a POS_1 is a product of all of the maxterms in 1. A SOP_0 is a sum of all of the minterms in 0. A SOP_1 is a sum of the minterms in 1. A cube represents one or more terms.

2.1. Prime implicants calculation

A prime implicant represents one or more terms. The prime implicant will form the covering table, which will further on be described.

All the terms of the truth table of the binary function should be obtained, including the terms with 'don't care' values. Each term is a cube. All of the cubes (terms) are separated in sets in agreement with the amount of literal in '1'. Each cube with k literals of value '1' will belong to the set K. Each pair of cubes k_n (belonging to the set T_K) and k_{n+1} (belonging to the set T_{K+1}) that differ from each other by just one literal value should be combined, replacing this literal by a 'don't care'. The cubes that do not combine will be prime implicants and they should be added to the set P of prime implicants. The generated cubes should be separated again in sets and combined, repeating such steps until no more cube can be combined therefore obtaining all prime implicants.

2.1.1 Example of prime implicants calculation

Consider the function(x_1, x_2, x_3) represented by SOP_1, where the '!' notation represents the complemented value of the variable

$$!x_1.!x_2.!x_3 + !x_1.x_2.x_3 + x_1.x_2.!x_3 + x_1.x_2.x_3$$

The terms 000, 011, 110 and 111 are obtained from the truth table (minterms, because it is a SOP_1). Separating the cubes (terms) in each set, it will be T_0 = {000}, T_1 = { }, T_2 = {011,110} and T_3 = {111}.

T_0 and T_1 do not have combinations, as the same for T_1 and T_2. T_2 and T_3 have two pairs of combinations. The pair (011,111) generates the cube X11 and the pair (110,111) generates the cube 11X. The cube 000 did not combine, then it is a prime essential: P = {000}. Because the cubes X11 and 11X will not combine in the next steps, they are essential primes. Then the set P = {000, 11X, 11X} is obtained.

2.2. Minimal covering table

The objective of the minimal covering table is to obtain to smallest amount of essential primes that represent the terms of the function. Another option is to obtain the smallest amount of literal for cube, called here as the lower-bound method [1]. In the case of obtaining the lower-bound version, the removal of all cubes with greater or equal amount of literal should be tried, based on the condition that all those cubes are covered by cubes with smaller amount of literal. All cubes of largest amount can be removed recursively until that it is no longer possible to minimize them, and later to proceed with the objective of obtaining the smallest amount of essential primes.

2.2.1. Stage 1

The covering table is set up with all of the elements of the set P of prime implicants and their covered terms. Each line of the table represents an essential prime. The prime implicants are put in growing order of literal (decreasing of 'don't cares'). The columns represent all the prime implicants, except the terms with 'don't care' value. For each line, it is marked the columns when the prime implicant cover the term of the column.

2.2.2. Stage 2

When a term is covered only by a single prime implicant, the prime implicant becomes a prime implicant essential for the function. After setting up the covering table, all essential prime implicants should be removed from the covering table, as well as remove all terms covered by the essential prime implicants. The essential prime implicants should be added to the set E of essential prime implicants. If the table is empty, the minimized function will be the elements of E.

2.2.3. Stage 3

The dominant terms should be removed. A term t_1 is dominant when exists at least another term t_2, in which all the prime implicants that cover t_2 also cover t_1.

2.2.4. Stage 4

The prime implicants dominated should be removed. A prime implicant i_1 is dominated when exists at least another prime implicant i_2 that cover all the terms of i_1. In case i_1 and i_2 are equivalent, in other words, they dominate each other, the prime implicant with larger amount of literals should be removed.

After reducing the table, one should return to the stage 2. In case that there are no more modifications, in other words, the cycle of the stages 2, 3 and 4 no longer alter the covering table, this table is in a cyclical case, where there are no dominant and dominated prime implicants. In that case, Petrick's method [10] is applied.

2.2.5. Petrick's Method

A POS is built, in which each sum of the POS will represent a term of the cyclical covering table. The elements of each sum of the POS are the prime implicants to cover such term. The objective is to obtain the smallest amount of prime implicants that satisfies the POS. To do so, the law of the distribution is used. The result should be the product with the smallest amount of prime implicants. The prime implicants are removed from the product and added to the set P of prime implicants. The result of the minimized function binary is the union of the set E of essential prime implicants and the set P.

If the result of the minimized binary function is a SOP_1, the resulting cubes (prime implicants of E union P) will be the products of the sum. In case it is a POS_0, they will be the sums of the product.

2.2.6. Petrick's Method example

Given a cyclical covering table with the terms t_1, t_2, t_3 and t_4, with the prime implicants i_1, i_2, i_3 e i_4.

$POS_{Petrick} = (i_1 + i_2).(i_1 + i_3).(i_2 + i_4).(i_3 + i_4)$
$= (i_1 + i_2.i_3).(i_2.i_3 + i_4)$
$= i_2.i_3 + i_1.i_4$

The law of the distribution is used. So much i_2 and i_3 as i_1 and i_4 cover the cyclical covering table. The result of the Petrick's method will be the prime implicants i_2 and i_3, or i_1 and i_4.

Table 1. Cyclic covering table to $POS_{Petrick}$.

	t_1	t_2	t_3	t_4
i_1	C	C	-	-
i_2	C	-	C	-
i_3	-	C	-	C
i_4	-	-	C	C

2.3. Criteria in the generation of covering table

For pattern, Quine-McCluskey algorithm minimizes a binary function with the objective of obtaining the smallest amount of cubes (prime implicants) that represent the function entirely, resulting in a smaller amount of sums in SOP or products in the POS. One can also be tried a smaller amount of literal for cubes, called lower-bound [1]. A smaller amount of literal for cubes reduces the amount of transistors in series in CMOS-like networks, bringing several advantages described in [1].

2.4. Results and analysis

Consider the $SOP_1 = \sum m(0, 1, 4, 5, 8, 9, 10, 11, 13, 16, 20, 22, 23, 24, 28, 29, 30, 31)$. Below they follow the results of Quine-McCluskey applied in that SOP:

Table 2. Several representations of the same function.

	'lower-bound'	Function
POS_0	No	(A+!B+!C+E).(A+!C+!D). .(!A+B+D+!E).(!A+C+!E). .(!A+C+!D)
POS_1	No	(A+!B+C).(!A+!C+!D). .(!A+D+E).(A+B+D). .(!B+!C+D+!E)
SOP_0	No	!ABC!E + !ACD + A!B!DE+ + A!CE + A!CD
SOP_1	No	!AB!C + ACD + A!D!E+ +!A!B!D + BC!DE
POS_0	Yes	(A+!B+!C+E).(A+!C+!D). .(!A+B+D+!E).(!A+C+!E). .(!A+C+!D)
POS_1	Yes	(A+!B+C).(!A+!C+!D). .(!A+D+E).(!A+!B+!C). .(B+D+E).(A+D+!E)
SOP_0	Yes	!ABC!E + !ACD + A!B!DE+ + A!CE + A!CD
SOP_1	Yes	!AB!C + ACD + A!D!E+ +!A!DE + ABC + !B!D!E

The POS_1 and the SOP_1 with and without lower-bound are syntactically different. In the SOP_1, the products '!A!B!D' and 'BC!DE' are represented by the products '!A!DE', 'ABC' and '!B!D!E'. By doing so, two minterms (with 3 and 4 literals) are replaced by three minterms all of them with 3 literals, which is advantageous in terms of implementation. The case is similar for POS_1.

3. MUX-BASED

The principle of Mux-based style is to obtain the implementation of a binary function starting from a multiplexer, or simply mux, where one of the variables of the function is the input of the mux and the other variables are the control.

3.1. Mux-based example

Consider the SOP_1 S = !A.!B.C + A.!B.!C + A.B.!C + A.B.C represented by mux-based, with input C and control A e B, generating S_{MUX} = !A.!B.C + !A.B.0 + A.!B.!C + A.B.1 = !A.!B.C + A.!B.!C + A.B. Note that, !A.B.0 was removed from S_{MUX} and that A.B.1 was optimized to A.B.

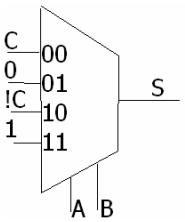

Figure 1 - Mux representing the equation:
SOP_1 = !A.!B.C + A.!B.!C + A.B.!C + A.B.C.

Depending on which variable is chosen to be the input of the mux, it can have dependences of the input variable more or less, in other words, to have more or less constant (0 or 1) as input.

3.2. Algorithm to find the input variable that generates the best mux for a binary function

It is obtained from the truth table of the binary function a list L_0 of the minterms with values in '0', other list L_1 with values in '1' and other list L_{dc} with values in 'don't care'. For each possible input of the mux it is made the combinations of the values of the true table, contained in those lists, analyzing the dependence of the input variable in relation to the output. If the dependence exists, the value of the input can depend on the value of the variable directly or to depend on the denied value of the variable. If the dependence does not exist, the input value will be constant and it will be '0' or '1'. A list with the combination of the variables of control of the mux is created for each dependence type. Then, it is analyzed for which input it is obtained the larger number of independence of the input variable, in other words, a larger number of 0's and 1's as input to the mux.

The result of the analysis is the input variable that results in larger amount of constant input values (0 or 1), i.e. that has the largest list of independent values of input.

3.3. Results and analisys

Consider the sum-of-products Boolean equation:

$$SOP1 = \sum m(0,1,4,5,8,9,10,11,13,16,20,22,23,24,28,29,30,31)$$

The function has five variables: v_1, v_2, v_3, v_4 and v_5. The result of the algorithm of Mux-Based is in Table 3. Both v_1 and v_4 have the largest number of input constants, or independences of the input variable.

Table 3. Relation of the input variable with the amount of independences (constants of input)

Input variable	# Input constants
v_1	12
v_2	8
v_3	8
v_4	12
v_5	6

5. CONCLUSIONS

In this paper was presented a CAD tool for logic equation synthesis, including the Quine-McCluskey sum-of-products and product-of-sums, as well as mux-based approach. This tool is very useful for logical synthesisas well as for CMOS transistor network generation.

6. REFERENCES

[1] F.R.Schneider, R.P.Ribas, S.S.Sapatnekar, A.I.Reis, "Exact lower bound for the number of switches in series to implement a combinational logic cell", ICCD 2005, pp. 357 – 362.

[2] F.R.Schneider, A.I.Reis, "Fast CMOS Logic Style Using Minimum Transistor Stack for Pull-up and Pull-down Networks", IWLS 2006, pp. 134 – 141.

[3] R.E.B. Poli, F.R. Schneider, R.P. Ribas and A.I. Reis. Unified theory to build cell-level transistor networks from BDDs. SBCCI 2003. Pages: 199 – 204.

[4] M. Avci and T. Yildirim. General design method for complementary pass transistor logic circuits. Electronics Letters, Vol.: 39, Number: 1, 9 Jan. 2003. Pages: 46 – 48.

[5] D. Kagaris and T.Haniotakis, "Transistor level Optimization of Supergates", ISQED 2006, pp.1 – 10.

[6] L.S.da Rosa Jr., F.Marques, T.M.G.Cardoso, R.P.Ribas, S.S.Sapatnekar, A.I.Reis, "Fast Transistor Networks from BDDs". SBCCI 2006 proceedings.

[7] N. Weste and K. Eshraghian, "Principles of CMOS VLSI Design". Addison-Wesley, 2nd edition, 1993.

[8] L.S.da Rosa Jr., F.Marques, T.M.G.Cardoso, R.P.Ribas, A.I.Reis. "BDDs and transistor networks with minimum pull-up/pull-down chains", IWLS 2006, pp. 142 – 149.

[9] McCluskey E.J. Minimization of Boolean functions. 1956. Bell System Tech. J., Vol. 35, No. 5, pp. 1417 – 1444.

[10] Petrick S.K. On the minimization of Boolean functions. 1959. Proc. Int. Conf. Information Processing, Paris: Unesco, pp. 422 – 423.