Synthesis of MVL Functions – Part I: The Genetic Algorithm Approach

Bambang A. B. Sarif
Department of Computer Engineering
KFUPM, KSA
+966-3-8604920
sarif@ccse.kfupm.edu.sa

Mostafa Abd-El-Barr
Department of Information Science CFW, Kuwait University
965-254-9408
mostafa@cfw.kuniv.edu

Abstract—**Multiple-Valued Logic (MVL) has been used in the design of a number of logic systems, including memory, multi-level data communication coding, and a number of special purpose digital processors. Several algorithms have been proposed in the literature for synthesis of multiple valued logic functions. None of these algorithms provides absolute optimum results for synthesis of these functions. The search space is too large to be explored by deterministic algorithms. In this paper, a Genetic Algorithm based algorithm for synthesis of MVL functions is proposed. The algorithm is tested using 200 randomly generated 2-variable 4-valued functions. The results obtained show that the introduced algorithm outperforms the deterministic technique based on the direct cover approach [3] in terms of the average number of product terms required to realize a given MVL function.**

Keywords—**GA, synthesis, MVL, and multi-valued logic.**

I. INTRODUCTION

TWO main categories of algorithms have been reported in the literature for synthesis of multiple-valued logic (MVL) functions. These are the deterministic and the iterative heuristic-based techniques. The deterministic algorithms are based on the use of the direct cover approach. The main difficulty with these techniques is the need for prohibitively high computational time [1,3,4,11]. Iterative heuristics offer the possibility of exploring larger solution space in arriving at near optimal solutions. A number of these techniques have been reported in the literature [2,7,8,10].

In this paper, we introduce a heuristic technique based on genetic algorithms for synthesis of multiple-valued logic (MVL) functions. The paper is organized as follows. In Section 2, we provide some background material. In Section 3, we introduce the proposed approach. In Sections 4 and 5, the experimental work, together with the results obtained, is presented. Some concluding remarks are drawn in Section 6.

II. BACKGROUND MATERIAL

An n-variable MVL function of radix r, f(X), is defined as a mapping f : Rn→R, where R={0,1,…,r-1} is a set of r logic values with r ≥ 2 and X={x1,x2,…,xn} is a set of n variables. The MVL operators reported in the literature include the minimum operator, min, e.g. min(5,2,3) = 2, the maximum operator, max, e.g. max(5,2,3) = 5, the truncated sum operator, tsum, defined as tsum(a1, a2, …, an) = a1 ⊕ a2 ⊕ … ⊕ an = min(a1 + a2 + … + an, r – 1), where ⊕ is the symbol used to indicate tsum, and the complement of a logic level l, defined as l' = (r – 1) – l. These are used, together with the literal operator, to synthesize MVL functions. The literal operator is defined as follows.

$$ {}^{a}X^{b} = \begin{cases} r-1 & if\,(a \le X \le b) \\ 0 & otherwise \end{cases} $$

where a, b ∈ R, and a ≤ b.

The set consisting of {Literal, MIN, TSUM, Constant} is used in this paper.

Direct cover approaches for synthesis of MVL functions consist of the following main steps.

1. Choose a Minterm,
2. Identify a suitable implicant that covers that minterm,
3. Obtain a reduced function by removing the identified implicant,
4. Repeat Steps 1 to 3 until no more minterms remain uncovered.

The selection of minterms and the implicants covering them play an important role in obtaining less expensive solutions (in terms of the number of product terms required to cover a given function). The direct cover approaches reported in the literature differ in the way minterms are chosen and the way according to which implicants are identified. The algorithm due to Armstrong [4] selects minterms randomly and selects the implicant resulting in the largest number of zero minterms (LRZ). The algorithm due to Besslich [1] uses what is called the isolation weight (IW) for choosing minterms and selects the minimum cost implicant to cover each minterm. The algorithm due to Dueck & Miller [3] introduces what is called the isolation factor (IF) for choosing minterms and selects implicant having minimum Relative Break Count (RBC). The last two techniques choose minterms in increasing order of values, i.e., they start with lower minterm values and proceeds to higher minterm values.

Consider the example shown in Figure 1. Dueck & Miller algorithm works by selecting the most isolated lowest value minterm and then selects literal having the minimum RBC covering that minterm iteratively. The list of minterm and implicant selected by the algorithm is shown inTable 1. Final representation of the function is then F=1•0X12•2X23 ⊕ 1•1X11•3X23 ⊕ 2•2X12•3X23 ⊕ 2•0X11•1X22.

1-4244-0765-6/06/$20.00 ©2006 IEEE

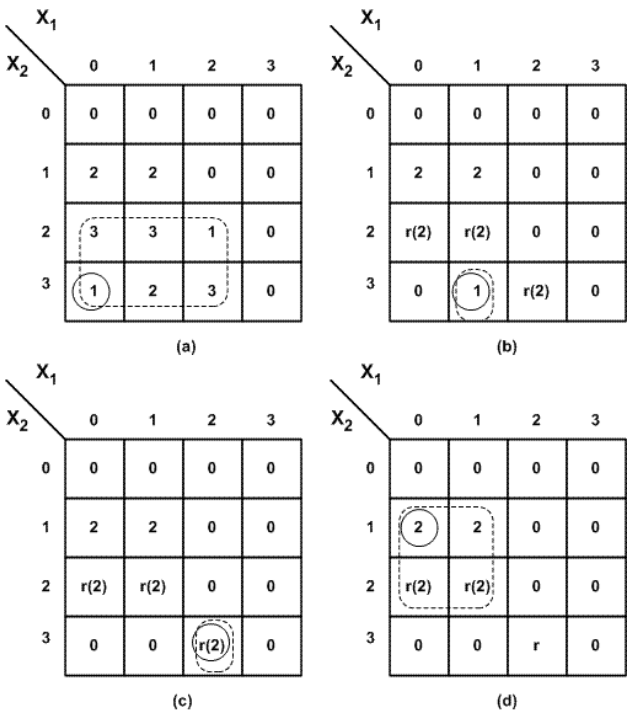

Figure 1. Example of Dueck & Miller algorithm [3].

TABLE 1. MINTERM AND LITERAL SELECTED BY ALGORITHM [3].

Step	Minterm	Literal
1	$1\bullet^0X_1{}^0\bullet^3X_2{}^3$	$1\bullet^0X_1{}^2\bullet^2X_2{}^3$
2	$1\bullet^1X_1{}^1\bullet^3X_2{}^3$	$1\bullet^1X_1{}^1\bullet^3X_2{}^3$
3	$2\bullet^2X_1{}^2\bullet^3X_2{}^3$	$2\bullet^2X_1{}^2\bullet^3X_2{}^3$
4	$2\bullet^0X_1{}^0\bullet^1X_2{}^1$	$2\bullet^0X_1{}^1\bullet^1X_2{}^2$

III. PROPOSED APPROACH

The solutions obtained using GAs are represented as strings of chromosomes. Each chromosome consists of several Genes, which represent a product term. Each gene consists of five integer attributes. The first attribute represents the value of the constant of that corresponding product term. The 2nd and 3rd attributes are for the window's boundary of product term for the first variable X1. The 4th and 5th attributes are for the window's boundary for the second variable, X2. For example, the first product term of function F shown in Figure 2(c) (1•0X10•0X22) is represented by 10002. It is also shown that function F can be represented by two different chromosomes (see Figure 2(c) and Figure 2(d)).

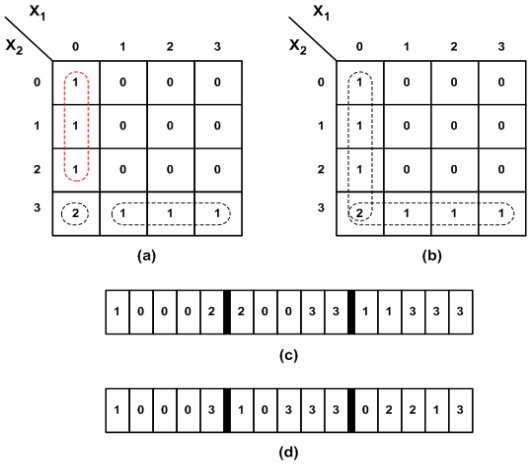

Figure 2. Two different chromosome representations.

The length of the chromosome is an important design parameter. If the chromosome is too small, the GA will unlikely find the best solution. On the other hand, if the chromosome is too large, the GA may waste a lot of time finding solution in the no-solution space. A straightforward approach is to use the length of truth table as the length of chromosome. But this does not guarantee that the GA will find the best solution in an effective way. In this paper, different lengths of chromosome are tested. These are explained below.

Definition: The length of truth table (LTT) is the number of all possible minterms of an MVL function.

For 2-variable 4-valued function, the LTT is equal to 16. For 3-variable 4-valued function, LTT is equal to 43 = 64.

For 2-variable 4-valued function, the size of LTT(=16) can be used as the length of chromosome since each minterm (rather than an implicant) can be represented as a gene. However, in most cases, the number of product terms will likely to be less than the number of minterms. It has been shown in [10] that the maximum number of implicants to cover any 2-variable 4-valued function is 12, which is 75% of LTT.

The number of non-zero minterms of a MVL's truth table, NM, for function shown in part (a) of Figure 3 is 7, while it is 8 for the function shown in part (b) of the figure. The function shown in part (a) of the figure has 7 minterms and can be expressed as F=1•0X13•0X20 ⊕ 1•0X10•0X23, which contains 2 implicants only. There is no need to use 12 or even 16 genes to represent this function. Even if the length of chromosome is 7, the search space for the GA to explore is still large. Thus, the length of the chromosome shouldn't be kept static. It should depend on the number of minterms.

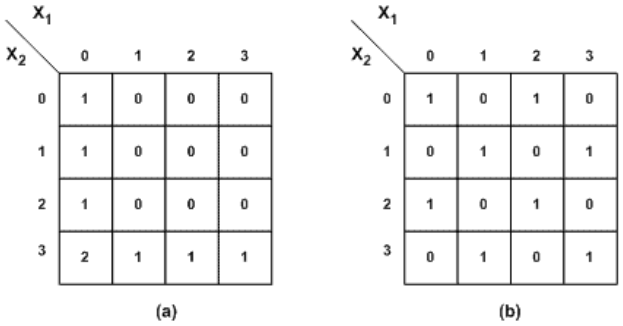

Figure 3. Two examples of a 2-variable 4-valued function.

The following two approaches for choosing the length of chromosome (LC):

5. Static (SGA-MVL): LC is equal to LTT (Length of Truth Table).
6. Reduced Static (RSGA-MVL): LC is equal to 75% of LTT.

A. *Fitness Function Calculation*

Two goals must be reflected in the fitness function calculation. These are correctness and optimality. Correctness deals with the functionality of the representation. A correct expression will cover all minterms and only those minterms of the function. Optimality deals with the quality of solution, i.e., how many implicants are required to implement the function.

In this paper, the fitness function is divided into two parts. The first part is called Functional Fitness, Ff. The second part of the fitness function is called Objective Fitness, Fo. These are explained below.

Functional fitness

Functional fitness is obtained by comparing the truth table of the given function with that of the obtained one. The value of the functional fitness is equal to the number of minterm matching (hits) between the two truth tables less the number of mismatch between the two truth tables. This can be formulated as follows:

$$F_f = N_h - N_m$$

Equation 1. Functional fitness.

where

N_h = Number of hits

N_m = Number of misses

The truth table of a given function is obtained by scanning the tabular format of the function row-wise and transforms it into a string of integer. For example, the function shown in Figure 2 will be represented by the string "1000100010002111".

The truth table of the obtained function is synthesized by adding the truth table of all of its product terms. Consider, for example, the chromosome X= "101111002310233" as the one obtained for the function shown in Figure 2. The chromosome consists of three genes (three product terms). Thus, the truth table of the obtained function is:

7. G_1 = "10111". Truth table of G_1 = "0000110000000000". Truth table of obtained function so far is "0000110000000000".
8. G_2 = "10023". Truth table of G_2 = "0000000010001000". Truth table of obtained function will become "0000110000000000" ⊕ "0000000010001000" = "0000110010001000".
9. G_3 = "10233". Truth table of G_3 = "0000000000001110". Truth table of obtained function will become "0000110010001000" ⊕ "0000000000001110" = "0000110010002110".

Using the formulation shown in Equation 1, the following are obtained Nh= 13, Nm= 3 and Ff F_f = 10.

Objective fitness

This is obtained by calculating the number of product terms needed in a given solution. The less the number of product terms the better the Objective Fitness is. There exists a number of ways to measure the Objective Fitness. In this paper, the following formulation is used:

$$F_o = \frac{100 - N_p}{100}$$

Equation 2. Objective fitness.

where

N_p = Number of product terms needed

Using the above formulation, the objective fitness of chromosome X given above is 0.97. The justification for the use of the above formulation is twofold. First, it is easy to interpret the value of the Objective Fitness to obtain the number of product terms. Second, there is a benefit in calculating the overall fitness, as explained below.

Overall fitness

Overall fitness of a chromosome is the addition of both functional fitness and objective fitness, i.e.,

$$Fitness = F_f + F_o$$

Equation 3. Overall fitness.

Using Equation 3, a chromosome with the highest functional fitness will be considered best solution. However, if there is more than one chromosome that has the highest functional fitness, then the value of objective fitness will determine which of these to use as the best solution.

IV. EXPERIMENTAL RESULTS

The experiments that have been conducted and the results obtained are presented. There are several merits that are to be measured. These merits are:

10. SR = Successful Rate (the number of successful iterations divided by the number of runs)
11. PT = Number of Product Terms

Unless it is mentioned explicitly, for all experiments, the following settings are used:

12. Number of Populations = 2000
13. Maximum number of Iterations = 500
14. Maximum number of run = 10
15. Crossover Rate = 100%
16. Mutation Rate = 5-10%

V. RESULTS OBTAINED

For the purpose of the experiment, 200 randomly generated 2-variable 4-valued functions are generated. The proposed approaches were able to find solutions for those functions. Table 2 shows a comparison of the average value of SR and PT for all 200 2-variable 4-valued functions. The table shows that RSGA-MVL outperforms the other approach.

TABLE 2. SR AND PT VALUES FOR ALL PROPOSED APPROACHES

Algorithm	*SR*	*PT*
SGA-MVL	0.45	7.24
RSGA-MVL	0.58	7.18

Table 3 shows the average number of implicants required to implement the 200 randomly generated 2-variable 4-valued functions for both the proposed approach and existing techniques [1][3][4]. It can be seen that the proposed algorithms outperforms other techniques.

TABLE 3: COMPARISON WITH OTHER TECHNIQUES

Algorithms	*PT*
ARM [4]	7.955
BS [1]	8.055
DM [3]	7.355
SGA-MVL	7.24
RSGA-MVL	7.18

VI. CONCLUDING REMARKS

A GA-based technique for synthesis of MVL functions is used in this paper. The comparison made with the direct cover approaches [1, 3, 4] using 200 randomly generated 2-variable 4-valued functions shows that the technique introduced produces better results, in terms of the number product terms needed to synthesize a number of functions.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support of Kuwait University for the work reported in this paper through Research Grant # WI 05/04.

REFERENCES

[1] Besslich P. W. "Heuristic Minimization of MVL functions: A Direct Cover Approach." IEEE Transactions on computer, Feb1986, pp.134-144.

[2] C. Yildirim and J. T. Butler and C. Yang, "Multiple-valued PLA minimization by concurrent multiple and mixed simulated annealing," Proceedings of the 23rd International Symposium on Multiple-Valued Logic, May 1993, pp. 17-23.

[3] Dueck, G. W. and Miller, D. M., "A Direct Cover MVL Minimization Using the Truncated Sum." Proceeding of the 17th international symposium on multi-valued logic, May 1987, pp. 221-227.

[4] G. Pomper and J. A. Armstrong, “Representation of Multivalued Functions Using Direct Cover Method,” IEEE Transactions on Computing. Sept. 1981, pp. 674-679.

[5] G. W. Dueck and G. H. J. van Rees, “On the Maximum Number of Implicants Needed to Cover a Multiple-Valued Logic Function Using Window Literals”, International Symposium on Multiple Valued Logic (ISMVL 1991), pp. 280-286.

[6] J. Holland. Adaptation in Natural and Artificial Systems. MIT Press, MA. 197

[7] Kalganova, T., J. Miller and T. Fogarty (1998) Some Aspects of an Evolvable Hardware Approach for Multiple-Valued Combinational Circuit Design. Proc. of Second International Conference on Evolvable System: From Biology to Hardware (ICES'98). Lausanne, Switzerland. pp. 78-89.

[8] Kalganova, T., J. Miller and N. Lipnitskaya (1998) Multiple-Valued Combinational Circuits Synthesized using Evolvable Hardware Approach. Proc. of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems (ULSI'98) in association with ISMVL'98, Fukuoka, Japan.

[9] Takahiro Hozumi, Osamu Kakusho, Kazuharu Yamato: An Evolutionary Computing Approach to Multilevel Logic Synthesis Using Various Logic Operations. ISMVL 2000: pp. 259-264

[10] Tirumalai, P. and Butler, J. “Analysis of Minimization Algorithms for Multiple-Valued Programmable Logic Arrays.” ISMVL 1988, pp. 226-236.