AN APPLICATION OF MICROBIAL GENETIC ALGORITHM FOR VLSI CAD

Sumanth Jagannathan
IV Year B.Tech,
Dept. of Electrical Engineering
Indian Institute of Technology-Madras
Madras, Tamil Nadu, India
email: sumj@hotmail.com

Jay Kumar Sundararajan
IV Year B.Tech,
Dept. of Electrical Engineering
Indian Institute of Technology-Madras
Madras, Tamil Nadu, India
email: jay_kumar_s@hotmail.com

Abstract

This paper is a step towards the application of evolutionary algorithms for digital circuit design. Here we apply the microbial genetic algorithm to the problem of two-level Boolean logic minimization. Genetic algorithms have proved to be an efficient and efficacious answer to problems of optimization, especially when common methods are too computationally intensive. Microbial genetic algorithms are a minimalist version of GAs which take less processing time. We test the algorithm on various two-level problems chosen from the MCNC benchmarks and compare its performance with the ESPRESSO algorithm.

1 INTRODUCTION

In VLSI design, the two main aspects to be taken care of, are the functionality of the IC, and the area occupied by the chip. Careful design and testing procedures ensure the functionality of the chip. On the other hand, optimizing the area on the chip involves reducing the number of gates required in the design as much as possible. Only after this step, layout optimization is done to further minimize the area required per gate. Thus, two-level logic minimization is an important step in VLSI design.

Various optimization techniques have been proposed in the past for two-level logic optimization such as the ESPRESSO algorithm and [5] (Coudert et al). In this paper, we investigate whether genetic algorithms can be used for this problem.

In two-level minimization, as the number of inputs and outputs increase, the CPU time taken to get the optimal solution becomes an important constraint. So we require fast algorithms. This led us to use microbial GAs for this problem. The microbial GA ([3] Harvey, 1996) is a minimal form of the genetic algorithm.

In conventional GA, a generational system is used. From one generation a parental pool is selected with probabilities based on their fitness. Offspring are generated from pairs of parents using one-point, two-point, uniform or some other techniques for crossover. The new generation is then obtained by performing mutation.

On the contrary, in microbial GA there are no generations. We instead work with a single population of strings. First, two strings are chosen at random using methods such as roulette wheel selection. From these parents an offspring is generated using crossover and mutation. The less fit of the pair is then eliminated and replaced by the offspring. Microbial GA's retain all the important features of a conventional GA and still prove to be faster.

While applying genetic algorithms to constrained optimization problems, we have to take care of the fact that a string may not always represent a valid solution to the problem. Even a simple operation such as crossover of two valid strings may result in an invalid solution. The answer to this is to have **a problem-specific genotype representation** [4] (Schnecke,1996). We therefore chose a unique encoding scheme, which helps us to ensure validity of all strings. This has been explained in section 2.3.

This paper is a continuation of our earlier work [6] where we had applied the microbial genetic algorithm to simple test cases that we had generated on our own. The effects of different crossover and mutation rates on the time taken for the GA to converge and on the fitness of the final solution were studied. In this paper, we have tested the algorithm on the MCNC two-level logic benchmarks. We have compared the performance of the GA with that of the ESPRESSO algorithm.

This paper is divided into four sections. In section 2, we describe the methodology we have employed. Section 3

contains the algorithm which has been implemented. In section 4 we give the results obtained for the benchmarks for the GA and the ESPRESSO method.

2 METHODOLOGY

2.1 THE PROBLEM

In the MCNC benchmarks, the test cases are described in the following format. The number of inputs and outputs are first specified. Then the product terms required by the various functions are given in a tabular form. It is also specified which functions must cover which product terms. The outputs may have don't care entries also. The aim is to find the smallest set of product terms using which all outputs can be implemented in sum-of-products form.

2.2 CONSTRAINTS OF THE PROBLEM

The following constraint has to be satisfied. All the 1's and none of the 0's of the truth table corresponding to the outputs should be covered by the solution. The don't-cares may or may not be covered.

This is an equality constraint. Equality constraints may be subsumed into the system model ([1] Goldberg,1989). We have achieved this by keeping track of the 1's that were not covered as explained below.

2.3 ENCODING

The product terms are divided into 2 groups – minterms and non-minterms (by non-minterms we refer to those product terms that are not minterms). Consider the truth table to be implemented. If all the outputs are 0 for a particular input combination, then those non-minterms covering this input combination are unnecessary (i.e. they need not be included in the solution). The encoding is done only with the remaining non-minterms. We call these remaining non-minterms as "useful product terms".

The solutions are represented by strings in the following manner. Each string consists of a number of bits indicating the presence or absence of useful product terms. The size of the string depends on the number of useful product terms.

The minterms are excluded because they will be used to implement the constraint of covering all 1's of all the outputs. Given a string, if the non-minterms present in the solution do not cover a particular 1 of some output, then to cover that 1, the corresponding minterm is included in the solution. Thus we have used a problem specific genotype to ensure that all strings represent valid solutions.

2.3.1 Example

If the functions to be minimized are

$f1(x,y,z,w)=\Sigma m(4,6,12,14)$

$f2(x,y,z,w)=\Sigma m(1,6,12,13) + \Sigma d(4,14)$

m-minterms d-don't cares

the valid product terms(excluding the minterms) are :

yw', yz'w', yzw', xyw', x'yw' and xyz'.

So the string has 6 genes, i.e. 6 bits.

If the string is as follows,

1	0	0	0	0	1

the corresponding non-minterms included in the solution are yw' and xyz'.

These non-minterms do not cover the minterm x'y'z'w. Therefore this minterm is also included in the solution.

2.4 FITNESS

To assign fitness we use the method of penalties. We count the non-minterms represented by the string. We also count the minterms, which are necessary to ensure that the string represents a valid solution. For every product term included in the solution, a penalty is given to the fitness. Thus the fittest string represents the optimal solution.

2.4.1 Example

To any string we give an initial fitness of 30,000. For every product term in the solution, we give a penalty of 1. For the 4-input case considered here, these values ensure that no string has a negative fitness.

Consider the string in the previous example.

1	0	0	0	0	1

This solution has 2 non-minterms and 1 minterm. Therefore the total penalty is 3
The string is given a fitness of 29997.

2.5 IMPLEMENTATION

The above algorithm was simulated on the computer using Turbo C++. First the given benchmark test cases is converted into truth table form. Using this truth table, we identify the useful product terms. We form tables for each function, that tell which of the useful product terms can be covered by that function without violating any constraint. All this pre-processing is done in order to make the genetic algorithm faster.
The fitness of a string at any time is calculated using these look-up tables. The strings are stored in a manner that

allows crossover and mutation to be done using bit-wise operators. This helps speed up things.
A population size of 26 strings has been used throughout. The mutation rate used was 5 %. The crossover rate used was 100%. A number called the convergence limit is defined. This is the number of tournaments that the algorithm waits for the solution's fitness to change, before declaring the solution to have converged. This is chosen to be 5000. However for some test cases, it is found that a larger value is needed for getting the optimal solution.

3 THE MICROBIAL GENETIC ALGORITHM

1. Initialize a population of strings randomly, wherein every string is a candidate solution to the given problem.
2. Evaluate fitness for all the strings.
3. while (there is improvement in the max. fitness of the population) do

   begin

   a. Select 2 parent strings from the population randomly.

   b. Evaluate the fitness of both strings.

   c. *Crossover* these 2 parents to produce offspring (genetic exchange takes place) with a probability equal to the crossover rate.

   d. Carry out *mutation* on the offspring with a probability equal to the mutation rate.

   e. Replace the parent having less fitness with the newly formed offspring.

   f. Evaluate fitness for the new offspring.

   end begin

end

After all the iterations, select the string with maximum fitness as the solution for the given problem.

We have used one-point crossover and single mutation where a randomly chosen bit is altered with a certain probability.

4 RESULTS

The ESPRESSO algorithm as well as the microbial genetic algorithm were run for 7 MCNC benchmark test cases on a Pentium 4, 1.5 GHz processor. The results obtained are shown in table 1.

The table gives the time taken and the number of product terms in the final solution for each test case, for both the algorithms.

The results indicate that the performance of the genetic algorithm is comparable to that of the ESPRESSO algorithm in terms of the quality of the solution. In fact, for one of the test cases called squar5.pla, the GA was able to find a solution with fewer product terms than ESPRESSO.

However, in all the cases, GA takes more time to arrive at the solution compared to ESPRESSO. For functions having more inputs and outputs, the GA method runs into problems of large memory requirement and long time to converge.

5 CONCLUSION

In conclusion, this paper has demonstrated that the microbial GA can be applied to the problem of two-level minimization. It has been tested on MCNC benchmarks and its performance is found to be good with respect to how close the solution is to the optimum.

However, the time taken for the algorithm to converge is an important consideration. In this respect, using a more efficient data structure and quicker ways to calculate the fitness of a string can make the algorithm faster. Moreover, different kinds of genetic operators may also help to accelerate the convergence of the algorithm.

Another improvement proposed is with regard to the positioning of the genes in the chromosome. For example, one could arrange the chromosome so that the genes representing product terms that can be used for maximum number of functions are kept close to each other. Such an arrangement could help in the formation of good schema thereby leading to a better solution.

It is hoped that in future these ideas will enable this algorithm to perform on par with other existing algorithms even for larger number of inputs.

References

[1] David E. Goldberg. 1989. Genetic Algorithms in Search, Optimization , and Machine Learning. Addison-Wesley Longman, Inc.

[2] David Beasley, David R. Bull, Ralph R. Martin. 1993. An overview of Genetic algorithms:Part2, Research Topics, University Computing.

[3] Inman Harvey, The Microbial Genetic Algorithm, submitted as a letter to *Evolutionary Computation,* 19th January, 1996

[4] Volker Schnecke, Oliver Vonberger, A Genetic Algorithm for VLSI Physical Design Automation, Proceedings of ACEDC '96.

[5] Olivier Coudert, Jean Christophe Madre, Henri Fraisse, A New Viewpoint on Two-level Logic Minimization, Bull Corporate Research Center, France

[6] Jay Kumar S, Sumanth Jagannthan, Two-level Boolean Logic Mnimization using Microbial Genetic Algorithms, Late-breaking paper, GECCO 2001.

Name of test case	No. of inputs	No. of outputs	Espresso		MicGA	
			Time(s)	No. of product terms	Time(s)	No. of product terms
Test1	3	3	0.02	8	0.02	8
Bw	5	28	0.11	22	0.73	22
rd53	5	3	0.05	31	0.05	31
majority	5	1	0.03	5	0.05	5
xor5	5	1	0.02	16	0.02	16
squar5	5	8	0.05	26	0.27	25
rd73	7	3	0.11	127	1.47	127

Table 1. Results for ESPRESSO versus MicGA for MCNC benchmarks