INT. J. ELECTRONICS, 1994, VOL. 76, NO. 1, 99–106

BENTEN MST, SADIQ M. SAIT•GAP - A GENETIC ALGORITHM APPROACH TO OPTIMIZE 2-BIT DECODER PLAS •INTERNATIONAL JOURNAL OF ELECTRONICS 76 (1): 99-106 JAN 1994

GAP: a genetic algorithm approach to optimize two-bit decoder PLAs

MUHAMMAD S. T. BENTEN† and SADIQ M. SAIT†

PLAs with two-bit decoders at the inputs require a smaller area compared with standard two-level PLAs (Sasao 1984). The number of product rows required for such PLAs is a function of the assignment of pairs of variables to the decoders. This paper describes a minimization procedure that uses a genetic algorithm approach to reduce the size of two-bit decoder PLAs. Results are compared with those obtained by other approaches such as the Tomczuk and Miller heuristic (TMA) (1992) and the simulated annealing technique (Abd-El-Barr and Choy 1993). For large randomly generated test cases and bench-marks, our results are optimal or very near optimal.

1. Introduction

One very general way to implement a combinational logic function of n-inputs and m-outputs is to use a ROM of size $2^n \times m$ bits. The n inputs form the address of the memory and the m outputs are the data contained in that address. Since it is often the case that only a small fraction of the 2^n product minterms are required for a canonical sum-of-products (SOP) implementation, a large area is wasted by using a ROM. A PLA is another alternative that has all the generalities of a memory for the implementation of a combinational logic function. It contains a row of circuit elements *only* for those product terms that are actually required to implement a given logic function. Since PLAs do not contain entries for all possible minterms, they are usually more compact than ROMs having the same function.

A PLA consists of an AND plane and an OR plane and realizes a system of m functions $(f_1(x_1, x_2, \ldots, x_{n-1}, x_n), f_2(x_1, x_2, \ldots, x_{n-1}, x_n), \ldots, f_m(x_1, x_2, \ldots, x_{n-1}, x_n))$ of n variables $(x_1, x_2, \ldots, x_{n-1}, x_n)$. The function of the AND plane is to produce the product terms, and the OR-plane sums the product terms. A matrix notation can be used to represent a PLA. As an example, given below are three functions and their equivalent PLA matrix.

$$F_1(a,b,c) = abc + ab\bar{c} + a\bar{b}c + \bar{a}bc = p_1 + p_2 + p_3 + p_4$$

$$F_2(a,b,c) = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + a\bar{b}c + abc = p_5 + p_6 + p_3 + p_1$$

$$F_3(a,b,c) = ab\bar{c} + abc = p_1 + p_2$$

Received 4 August 1993; accepted 23 August 1993.

†Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran-31261, Saudi Arabia.

p_1	1	1	1	1	1	1
p_2	1	1	0	1	0	1
p_3	1	0	1	1	1	0
p_4	0	1	1	1	0	0
p_5	0	0	0	0	1	0
p_6	0	0	1	0	1	0
	a	b	c	F_1	F_2	F_3

1.1. *Optimization of PLAs*

The area of the VLSI layout of a PLA is directly proportional to the size of the PLA matrix which depends on (*a*) the number of product terms, (*b*) the number of inputs and (*c*) the number of outputs (Mead and Conway 1980). The area of the layout required to implement a given set of boolean functions can be reduced in several ways:

(1) by reducing the number of rows. This is possible by deleting any redundant products terms (Hachtell *et al.* 1980);

(2) allowing the area used by one column to share the circuit of two columns. This technique is known as 'folding'. In a folded PLA, AND plane inputs enter from either top or bottom. If two different inputs arriving from different directions, i.e. top or bottom, can share a column, then the circuitry of the two columns is placed in the area of one, thus reducing the width of the PLA layout (Hachtell *et al.* 1980);

(3) a PLA may be partitioned into two PLAs whose area sum is less than the area of the original single PLA (Ullman 1983);

(4) using 2-to-4 decoders whose output is fed to the AND plane of the PLA. This causes the width of the PLA to remain unchanged, but the number of product terms is considerably reduced (Sasao 1984).

A PLA implementing functions of n variables will have $2 \times n$ inputs. The number of inputs to a standard PLA is the same as that of a 2-bit decoder PLA ($2 \times n$); where n is the number of variables in the functions to be implemented. The number of product rows in a PLA can be far less if it receives inputs from decoders. To illustrate the point, consider the coincidence function given below:

$$F_{x_1,y_1,x_2,y_2} = (x_1 \odot y_1) \cdot (x_2 \odot y_2)$$

The above function requires four product terms in a standard PLA, while a single product row is sufficient if the inputs are fed through 2-bit decoders. This is illustrated in Fig. 1

The number of product rows required for a decoded-PLA is a function of the assignment of pairs of variables to the decoders. Several attempts have been made to

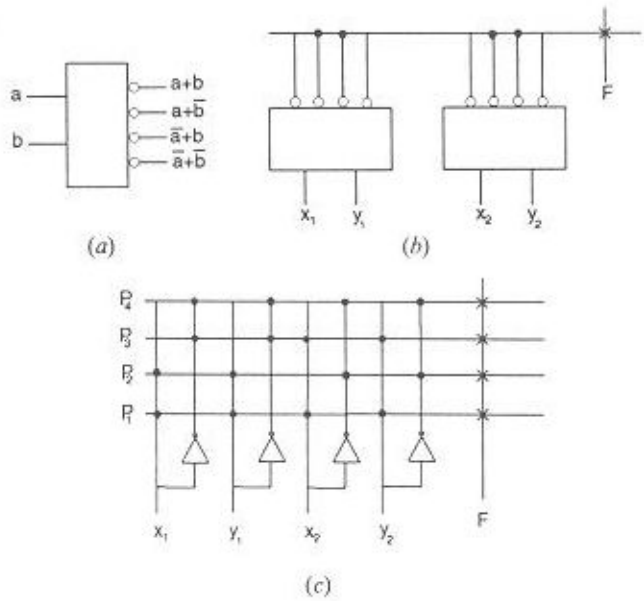

Figure 1. (*a*) 2-to-4 decoder; (*b*) PLA with decoder inputs for coincidence function; (*c*) standard PLA for coincidence function.

find an assignment that will require the smallest number of rows. Sasao addressed the problem of assigning variables to two-bit decoder PLAs (Sasao 1984). In his approach, a complete weighted graph of n nodes is constructed first (called the assignment graph). Each node of the assignment graph represents an input variable x_i and the weight of edge (i,j) represents the complexity of the minimized function if x_i and x_j are paired. The estimate used is the number of unique product terms that remain in the near-minimal two-level canonical expression after the variables x_i and x_j (and their inverses) are removed. The problem then reduces to assigning to a decoder the variables x_i and x_j for each edge (i,j) in a minimum-weight maximum-cardinality matching of the assignment graph. Chen and Muroga (1988) considered a more general grouping where multi-bit decoders are used to reduce the area of decoded-PLAs. Tomczuk and Miller (1992) proposed a heuristic algorithm to find an efficient pairing. Their heuristic is based on an autocorrelation function of truth tables of the function $f_1, f_2, \ldots, f_m$. Abd-El-Barr and Choy started from the assignment produced by Tomczuk and Miller (TMA) and found a better assignment using simulated annealing. In this paper, we present a genetic algorithm that finds an optimal assignment to the inputs of the PLA.

In the next section we present the genetic algorithm preliminaries. The various functions of the genetic algorithm are then elaborated. Results obtained by our algorithm are compared with those produced by the algorithms of Tomczuk and Miller (1992) and Abd-El-Barr and Choy (1993).

2. Genetic algorithm for PLA reduction

Genetic algorithms (GAs) are search algorithms that emulate the natural process of evolution as a means of progressing toward the optimum (Holland 1975). They

have been applied in solving various optimization problems including those in VLSI physical design (Cohoon and Paris 1987, Shahookar and Mazumder 1990).

In the GA approach, at any given instant a number of possible solutions, called the *population*, exists. Their number, denoted by N_p, depends on the problem instant, the size of the problem, and the available memory. Each individual in the population is a string of symbols. These symbols are known as *genes* and the string made up of genes is termed a *chromosome*. The chromosome represents a possible solution to the optimization problem. During each iteration (*generation*), the individuals in the current population are evaluated using some measure of fitness. Based on the *fitness* values, two individuals at a time (called *parents*) are selected from the population. The fitter individuals have a higher probability of being selected. Then genetic operators are applied on the selected parents to generate new individual solutions called *offsprings*. These genetic operators combine the features of both parents. Common operators are *crossover* and *mutation*. They are derived by analogy from the biological process of evolution. The structure of the genetic algorithm is given in Fig. 2. First, we see how a solution can be mapped to a string. Then we discuss the basic operators applied in searching for an optimum assignment to PLA inputs.

```
Procedure (Genetic_Algorithm_For_PLA_Reduction)
  N_p= Population Size.          (* # of possible solutions at any instance. *)
  N_g= Number of Generations.    (* # of iterations. *)
  N_o= Number of Offsprings.     (* To be generated by crossover. *)
  P_μ= Mutation Probability.
  Construct_Population(N_p)      (* Randomly generate initial population *)
    For j = 1 to N_p
      Evaluate Fitness(Population[N_p])
                        (* Reciprocal of # of rows given by espresso *)
    EndFor
    For i = 1 to N_g
      For j = 1 to N_o
          (x, y) ← Choose_parents
          (* Probability of parent selected is proportional to its fitness *)
          offspring[j] ← Generate_offspring(x, y) (* By crossover (PMX) *)

          For j = 1 to N_p
          With probability P_μ Apply Mutation(Population[j])
          EndFor

      Evaluate Fitness(offspring[j])
      EndFor
      Population ← Select(Population, offspring, N_p)
      (* Select best N_g solutions from parents and offsprings *)
    EndFor
    Return highest scoring configuration in Population
    End
```

Figure 2. Genetic algorithm for 2-to-4 decoder PLA reduction.

2.1. *String encoding* ζ

Any string of length n containing unique input variables $[x_1, x_2, \ldots, x_n]$ represents a possible solution. Since the decoder function is symmetric, the string can be thought of as containing $n/2$ pairs. A permutation of these $n/2$ pairs or a permutation of the pair itself (interchange of the two elements of the pair) does not produce a different solution. Thus a chromosome can be thought of as a set of $n/2$ elements, each element itself being a set of two unique input variables. The number of such unique sets of $n/2$ elements is very large ($n!/2^k \cdot k!$, where $k = \lfloor n/2 \rfloor$). Therefore, for large values of n (greater than 10) a systematic procedure must be applied to search for the solution space to obtain the best assignment. For example, for $n=6$ a possible solution represented as a set of sets is $\{\{x_1, x_4\}, \{x_2, x_6\}, \{x_5, x_3\}\}$. The corresponding string is the sequence $[x_1, x_4, x_2, x_6, x_5, x_3]$.

2.2. *Fitness function* σ

The fitness function, also known as the scoring function, depends on the number of product rows required by each assignment to implement the given function. This is obtained with the help of *espresso* (Brayton *et al.* 1984). *Espresso* takes (1) a PLA matrix and (2) the pairings (or assignment), as input and returns a reduced PLA. The reciprocal of the number of rows of the reduced PLA is the fitness of an individual in the population.

2.3. *Population constructor* Ξ

The initial population is randomly chosen from all possible unique assignments. The size of the population depends on the size of the PLA matrix, i.e. the number of inputs, outputs, and product terms. For the examples presented, typical values of $N_p = 10$ and $N_g = 10$ are used.

2.4. *Crossover operator* ψ

Crossover is the main genetic operator. It operates on two individuals and generates an offspring. It is an inheritance mechanism whereby the offspring inherits the characteristics of the parents. A simple crossover operation is as follows: it chooses a *random* cut point and generates the offspring by combining the segment of one parent (string) to the left of the cut point with the segment of the other parent to the right of the cut point.

In our case the elements of the sets are treated as an ordered sequence. Two such sequences of two parents are selected. The above simple technique is not directly applicable here because some variables to the left of the cut point in one parent may also exist in the right substring of the second parent. Several other crossover techniques have been reported in the literature that can be applied. One commonly used technique in such situations is the partially mapped crossover (PMX) technique (Shahookar and Mazumder 1990).

The PMX crossover is implemented as follows: two parents are selected (say 1 and 2) and a random cut point is chosen. The entire right substring of parent 2 is copied to the offspring. Next, the left substring of parent 1 is scanned from the left, gene by gene, to the point of cut. If a gene does not exist in the offspring, then it is copied to the offspring. However if it already exists in the offspring, then its position in parent 2 is determined and the gene from parent 1 in the determined position is copied.

As an example, consider the two parents $[x_2x_4x_5x_6x_7|x_3x_8x_1]$ and $[x_1x_7x_8x_3x_2|x_4x_5x_6]$. Let the crossover position be after 5. Then the offspring due to PMX is $[x_2x_3x_8x_1x_7|x_4x_5x_6]$.

2.5. *Mutation function* μ

Mutation produces incremental random changes in the offspring generated by the crossover. In our case, a simple mutation mechanism is the pairwise swap. Mutation is important because crossover alone will not help to obtain a good solution. Crossover is only an inheritance mechanism. The mutation operator generates new characteristics. If the new offsprings perform well, then the configurations containing them are retained and these spread throughout the population. The mutation rate controls the rate at which new genes are introduced into the population for trial. If mutation rate is low then many genes that would have been good are never tried out. If mutation rate is high then there will be too much random disturbance, causing the offsprings to lose resemblance to their parents and the algorithm will lose its ability to learn from the history of the search.

The structure of a genetic algorithm is given in Fig. 2. The algorithm starts with an initial set of random configurations N_p, the size of which is always fixed. Following this, a mating pool is established in which pairs of individuals from the population are chosen. The probability of choosing a particular pair for mating is proportional to the individuals fitness value. A roulette wheel technique is used where individuals with higher fitness value have a greater chance of being selected for crossover (Goldberg 1989). N_0 new offsprings are generated by applying crossover. The offsprings generated are next evaluated on the basis of fitness, and a new generation is formed by selecting the best N_p individuals from both the parents and the offsprings. Mutation is then applied with a fairly high probability to the entire new population except the best individual. In our experiments, we observed that only crossover with *no* mutation produces no optimal results. The above procedure is executed N_g times, where N_g is the number of generations. After a fixed number of generations (N_g), the fittest individual, i.e. the one with highest fitness value is returned as the desired solution.

3. Discussion and results

The results obtained by applying GA to randomly generated circuits are tabulated in Table 1. In all cases, the results obtained are better than those obtained

PLA circuit	n	Initial rows	Final rows		
			SA	GA	Optimal
1	10	68	20	19†	19
2	12	427	64	62	60
3	12	150	36	34	33
4	12	803	124	122	121
5	12	276	40	37†	37
6	6	16	8	8†	8
7	8	54	—	19†	19

Table 1. Table comparing the results of GA and SA († indicates GA = optimal).

PLA circuit	n	Initial rows	Final rows		
			TZM	SA	GA
1	4	16	9	8	8
2	4	16	7	5	5
3	6	64	20	19	18
4	6	64	30	28	27
5	8	256	67	67	64
6	8	256	102	97	95

Table 2. Table comparing the results obtained from the TZM heuristic, simulated annealing on TZM results and the genetic algorithm approach.

by applying simulated annealing alone. In most cases they are optimal. Table 2 shows the results obtained by executing (*a*) the TMA heuristic, (*b*) simulated annealing on TMA results, and (*c*) the genetic algorithm.

The TMA approach requires that the PLA be given as a completely specified truth table. The SA approach of Abd-El-Barr and Choy (1993) iterates on the constructive solution provided by TMA and therefore has a similar requirement. Our algorithm does not have any such requirement. It starts from a population of randomly generated possible solutions and systematically searches the solution space for an optimal assignment.

4. Conclusions

In this paper, we have presented a new technique to optimize decoded PLAs. Experiments on large randomly generated circuits and other bench-marks obtained show that the procedure is very effective in reducing the number of product rows required. The results obtained are comparable or better than some earlier approaches.

Acknowledgments

The authors acknowledge King Fahd University of Petroleum and Minerals for all support. They would also like to thank Dr M. Abd-El-Barr of University of Saskatchewan for supplying bench-marks and comparison data.

References

Abd-El-Barr, M., and Choy, H., 1993, Use of simulated annealing to reduce 2-bit decoder PLAs. *International Journal of Electronics*, **74**, 441–450.

Brayton, R. K., Hachtell, G. D., McMullen, C. T., and Sangiovanni-Vincentelli, L. A., 1984, *Logic Minimization Algorithms for VLSI Synthesis* (Boston, Mass: Kluwer Academic).

Chen, K. C., and Muroga, S., 1988, Input assignment algorithm for decoded PLAs with multi-input decoders. *Proceedings of the International Conference on Computer Aided Design*, pp. 474–477.

Cohoon, J. P., and Paris, W. D., 1987, Genetic placement. *IEEE Transactions on Computer Aided Design*, **6**, 956–964.

Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley).

HACHTELL, G. D., SANGIOVANNI-VINCENTELLI, L. A., and NEWTON, R. A., 1980, Some results in optimum PLA folding. *Proceedings of the International Conference on Circuits and Systems*, p. 1040.

HOLLAND, J. H., 1975, *Adaptation in Natural and Artificial Systems* (Ann Arbor, Mich: University of Michigan Press).

MEAD, C., and CONWAY, L., 1980, *Introduction to VLSI Systems* (Addison-Wesley).

SASAO, T., 1984, Input variable assignment and output phase optimization of PLAs. *IEEE Transactions on Computers*, **33**, 879–894.

SHAHOOKAR, K., and MAZUMDER, P., 1990, A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer Aided Design*, **9**, 500–511.

TOMCZUK, R., and MILLER, D. M., 1992, Autocorrelation techniques for multi-bit decoder PLAs. *Proceedings of the 22nd International Symposium on Multivalued Logic*.

ULLMAN, J. D., 1983, *Computational Aspects of VLSI* (Rockville, Maryland, U.S.A.: Computer Science Press).