Optimum PLA Folding through Boolean Satisfiability

J.M. Quintana[1], M.J. Avedillo[1], M.P. Parra[2] and J.L. Huertas[1]

Dpto. de Diseño Analógico, Centro Nacional de Microelectrónica
Edificio CICA. C/ Tarfia s/n. 41012-Sevilla

[1]also with the Dpto. de Electrónica y Electromagnetismo de la Universidad de Sevilla
[2]also with the Dpto. de Tecnología Electrónica de la Universidad de Sevilla

Tel: +34-5 423 99 23, FAX: +34-5462 45 06
e-mail: josem@cnm.us.es

Abstract - This paper proposes an algorithm for optimum PLA folding based on its formulation as a problem of boolean satisfiability. A logical expression is derived such that the assignment of variables that satisfies it defines a folding with a minimum number of columns. The proposed algorithm uses BDDs to represent boolean functions and incorporates novel reduction techniques, obtaining satisfactory results.

I. INTRODUCTION

The *Programmable Logic Array* (PLA) is a very effective structure for the design of both combinational and sequential logic. They have been extensively used in integrated circuit design, especially in controller implementations. A PLA consists of a group of rows (carrying each one a product term) and columns (corresponding to inputs and outputs). A logic gate can be located at each intersection of a row with a column. The objective of a folding algorithm is to determine permutations of the rows (and columns) which permit a maximal set of columns (rows) to be implemented in the same physical column (row) of the logic array. Since large arrays are usually very sparse, a considerable area reduction can be achieved by folding. If we restrict the folding to pairs of columns, simple folding is obtained; if no restrictions are imposed, the folding is multiple. The problem of optimum PLA folding is discussed extensively in literature [1-3]. Also, design ideas for the problem of multiple folding of PLAs apply to the gate matrix layout problem [3]. The folding problem has been shown to be NP-complete [4]. Consequently most efforts have been made in the search of heuristic algorithms.

Here, we transform the problem of simple or multiple column folding into one of boolean satisfiability in a similar way to [5] for the problem of optimal layout. This transformation constructs a boolean function where the number of variables for which we must determine codification increases linearly with the sum of input/output and product terms of the PLA. This transformation allows us to apply sophisticated logic verification strategies and check if the constructed function presents any combination which makes it "1". Among the various techniques that allow this check, we have used that based on BDDs.

Binary Decision Diagrams (BDDs) [6] form a convenient representation of logic functions that have proved efficient in diverse fields of VLSI design. Thus they have been used extensively in the formal verification of digital circuits, in numerous optimization problems, and in the general context of logic synthesis. A BDD is an acyclic directed graph (DAG) representing a multiple output logic function. Formally, a reduced and ordered BDD (ROBDD) representing a function $f(x)$: $\{0,1\}^n \rightarrow \{0,1\}$ is a DAG $(V \cup \{1\}, E)$, where V is a set of internal nodes. Each node $v \in V$ is associated with one of the variables and has two descendents denoted T (*then* child) and E (*else* child). The terminal node is $\{1\}$ and E is the set of branches of the graph.

The function f represented by a BDD is defined as follows:

1. The function of the terminal node is the constant function 1.

2. The function of a branch is the function of the node it is directed to, unless the branch has the complement attribute, in which case the function of the branch is the complement of the function of the node.

3. The function of a node $v \in V$ associated with the variable x_j is: $f_v = x_j f_{T(v)} + x_j f_{E(v)}$.

4. The function f represented by the BDD is the function represented by the root node.

A BDD is reduced if two distinct nodes under no circumstances represent the same function. A BDD is ordered if the variables appear in any path from the root to the terminal node in a previously-defined, identical order. Given a variable ordering, an ROBDD in which the use of the complementary attribute on the T branches is prohibited, is a canonical representation of logic functions. This is the type of representation used in the folding algorithm developed here; however, for commodity we will use the term BDD.

The rest of the paper is organized as follows: Section 2 presents formulation of the problem of optimum PLA folding as a problem of satisfiability and shows an example of the method.

Section 3 describes implementation of the proposed solution. Section 4 presents preliminary results obtained with a prototype of the algorithm, demonstrating the power of approximation. Lastly, Section 5 summarizes the conclusions.

II. Optimum Foldig Via Boolean Satisfiability

A PLA is generally described symbolically by its personality matrix. This is a matrix of I×R, with as many columns (I) as inputs and outputs, and as many rows (R) as product terms. In the AND plane, "1" at position (*i*,*j*) means that the *j*-th input is present in the *i*-th product (complemented or not); "0", that the *j*-th input does not appear in this product term. In the OR plane, "1" at position (*r*,*s*) means that the product term *r* forms part of the expression of the *s* function; and not if "0". In this study we consider the folding of columns. A group of rows implicated by the column c_i, $F(c_i)$, refers to the set formed by the rows where 1 appears in the personality matrix for that column. Two columns c_i and c_j can be folded if both belong to the same plane and $F(c_i) \cap F(c_j) = \varnothing$.

Given the personality matrix of a PLA, the following procedure constructs a boolean function such that if the function is satisfiable, then the PLA can be folded into *P* columns.

1) Each column c_i has $N = \lceil \log_2(I) \rceil$ associated boolean variables, $c_{i1}, \ldots c_{iN}$. These variables show us in which physical column c_i is.

2) Each product term r_j has $r = \lceil \log_2(R) \rceil$ associated boolean variables, $r_{j1}, \ldots r_{jr}$.

3) For each pair of columns (c_i, c_j) where c_i corresponds to an input (output) and c_j is an output (input), we will generate the equation $c_{i1} \oplus c_{j1} + c_{i2} \oplus c_{j2} + \ldots + c_{iN} \oplus c_{jN}$, where $\oplus$ is the *exclusive-or* operator. This indicates that an input cannot be folded with an output.

4) For each pair of columns (c_i, c_j), if $F(c_i) \cap F(c_j) \neq \varnothing$, we generate the equation $c_{i1} \oplus c_{j1} + c_{i2} \oplus c_{j2} + \ldots + c_{iN} \oplus c_{jN}$. This indicates that inputs that cannot be folded cannot be located in the same physical column.

5) Each pair of columns (c_i, c_j) with $F(c_i) \cap F(c_j) = \varnothing$ imposes a set of restrictions on the codes of rows belonging to $F(c_i)$ and $F(c_j)$. These restrictions impose that each row in $F(c_i)$ be over each row in $F(c_j)$ or vice versa. If r_i is an element of $F(c_i)$ and r_j of $F(c_j)$, then,

$$(c_{i1} \oplus c_{j1} + c_{i2} \oplus c_{j2} + \ldots + c_{iN} \oplus c_{jN}) + (r_{i1}, r_{i2}, \ldots, r_{ir}) > (r_{j1}, r_{j2}, \ldots, r_{jr}) + (r_{j1}, r_{j2}, \ldots, r_{jr}) > (r_{i1}, r_{i2}, \ldots, r_{ir}) \quad \forall r_i \in F(c_i), \forall r_j \in F(c_j)$$

that is, the inputs do not fold (and consequently have different codes), or fold, and they meet the condition that all the rows in $F(c_i)$ and in $F(c_j)$ are in different partitions.

6) If $P < 2^N$ there are binary vectors not used, hence we may prohibit their appearance. This can be expressed by the equation: $(c_{i1} \oplus c_1 + c_{i2} \oplus c_2 + \ldots + c_{iN} \oplus c_N)$ for each prohibited code $(c_1, \ldots c_N)$ and each column c_i.

7) The codes of each row should be different. This is expressed as $(r_{i1} \oplus r_{j1} + r_{i2} \oplus r_{j2} + \ldots + r_{ir} \oplus r_{jr})$ for every r_i and r_j.

8) If our interest lies in simple folding, additional equations are necessary to prohibit more than two columns having the same code. This can be expressed as:

$$(c_{i1} \oplus c_{j1} + c_{i2} \oplus c_{j2} + \ldots + c_{iN} \oplus c_{jN}) + (c_{i1} \oplus c_{k1} + c_{i2} \oplus c_{k2} + \ldots + c_{iN} \oplus c_{kN})$$

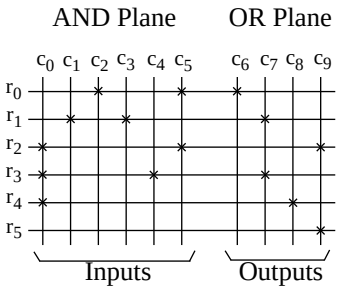
for each c_k which can be folded with c_i; that is, if the input c_i and c_j are folded, it must be avoided that any column c_k, capable of folding with c_i has its same code.

9) We construct the boolean function F_{pleg} from the conjunction of the equations obtained previously.

Obviously, the assignment which satisfies F_{pleg} ensures folding in *P* columns.

A. *Example:*

As an illustration of this formulation procedure, consider the PLA in Fig. 1 with 10 columns and 6 rows. Values of N = 4 and *r*=3 are used. We need three variables to code the rows. We can obtain the foldable and unfoldable pairs from its personality matrix.



001001 1000
010100 0100
100001 0001
100010 0100
100000 0010
000000 0001
Personality matrix

Figure 1: PLA description.

point 3), the unfolding columns due to their location on distinct planes are the pairs (c_i, c_j) where $i \in \{0,...5\}$ and $j \in \{6,...,9\}$; a total of 24 pairs. For example, the equation generated by (c_0, c_1), is:

$$c_{01} \oplus c_{11} + c_{02} \oplus c_{12} + c_{03} \oplus c_{13} + c_{04} \oplus c_{14}$$

point 4) the pairs of unfoldable columns due to non empty intersections are: (c_0, c_4), (c_0, c_5), (c_1, c_3), (c_2, c_5); and the equation corresponding to (c_0, c_4):

$$c_{01} \oplus c_{41} + c_{02} \oplus c_{42} + c_{03} \oplus c_{43} + c_{04} \oplus c_{44}$$

point 5) the pairs of foldable columns are: (c_0, c_1), (c_0, c_2), (c_0, c_3), (c_1, c_2), (c_1, c_4), (c_1, c_5), (c_2, c_3), (c_2, c_4), (c_3, c_4), (c_3, c_5), (c_4,c_5), (c_6, c_7), (c_6, c_8), (c_6, c_9), (c_7, c_8), (c_7, c_9), (c_8, c_9). For example, the equation generated by the pair (c_0, c_1) is:

$$c_{01} \oplus c_{11} + c_{02} \oplus c_{12} + c_{03} \oplus c_{13} + c_{04} \oplus c_{14} +$$
$$\left(r_{21} \cdot r_{22} \cdot \overline{r_{12}} + r_{22} \cdot \overline{r_{11}} \cdot \overline{r_{12}} + r_{21} \cdot \overline{r_{11}}\right) \cdot$$
$$\left(r_{31} \cdot r_{32} \cdot \overline{r_{12}} + r_{32} \cdot \overline{r_{11}} \cdot \overline{r_{12}} + r_{31} \cdot \overline{r_{11}}\right) \cdot$$
$$\left(r_{41} \cdot r_{42} \cdot \overline{r_{12}} + r_{42} \cdot \overline{r_{11}} \cdot \overline{r_{12}} + r_{41} \cdot \overline{r_{11}}\right) +$$
$$\left(r_{11} \cdot r_{12} \cdot \overline{r_{22}} + r_{12} \cdot \overline{r_{21}} \cdot \overline{r_{22}} + r_{11} \cdot \overline{r_{21}}\right) \cdot$$
$$\left(r_{11} \cdot r_{12} \cdot \overline{r_{32}} + r_{12} \cdot \overline{r_{31}} \cdot \overline{r_{32}} + r_{11} \cdot \overline{r_{31}}\right) \cdot$$
$$\left(r_{11} \cdot r_{12} \cdot \overline{r_{42}} + r_{12} \cdot \overline{r_{41}} \cdot \overline{r_{42}} + r_{11} \cdot \overline{r_{41}}\right)$$

point 6), the equations generated depend on the value of *P*, for example, the equation that excludes the code (1,0,1,1) (since there are only 10 columns) for the column c_0 is:

$$c_{01} \oplus 1 + c_{02} \oplus 0 + c_{03} \oplus 1 + c_{04} \oplus 1 = \overline{c_{01}} + c_{02} + \overline{c_{03}} + \overline{c_{04}}$$

point 7) we generate equations for each pair of rows. For the pair (r_0,r_1): $r_{01} \oplus r_{11} + r_{02} \oplus r_{12} + r_{03} \oplus r_{13}$

point 8) we generate equations to avoid multiple folding. For the pair of folding columns, (c_0, c_1):

$$(c_{01} \oplus c_{11} + c_{02} \oplus c_{12} + c_{03} \oplus c_{13} + c_{04} \oplus c_{14}) +$$
$$(c_{01} \oplus c_{21} + c_{02} \oplus c_{22} + c_{03} \oplus c_{23} + c_{04} \oplus c_{24}) \cdot$$
$$(c_{01} \oplus c_{31} + c_{02} \oplus c_{32} + c_{03} \oplus c_{33} + c_{04} \oplus c_{34})$$

If we wish to find a simple fold, the value of *P* cannot be less than $\lceil 6/2 \rceil + \lceil 4/2 \rceil$, which corresponds to a maximum fold. This value is 5 in the example PLA and in effect, a 5-column fold is possible (Fig. 2a). For a multiple folding the minimum value of *P* is 2, that is, a column of input and another of output. For the example PLA, it is impossible to find any assignment that leads us to a fold of these characteristics; increasing the value of *P* to 3, leads us to the solution shown in Fig. 2b. ■

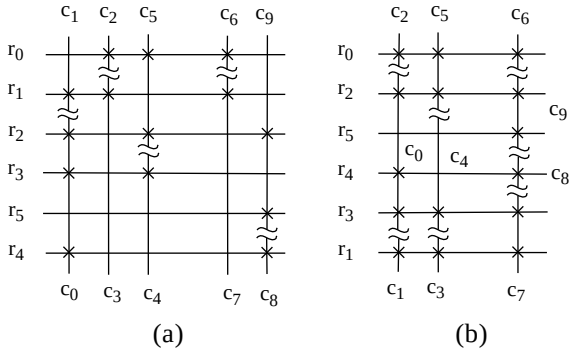


Figure 2: (a) Simple folding. (b) Multiple folding

III. Implementation of the New Folding Algorithm

The use of a technique based on the representation of the function derived from the previous Section through BDD, is attractive in its correspondence between solutions to our problem (assignments that satisfy the function) and paths from the root to the terminal nodes with an even number of branches affected by the complementary attribute. The complexity of determining a path with this characteristic is $O(n)$, where n is the number of variables of the function. However, difficulties frequently arise in the construction of the BDD itself, due to the excessive memory requirements. This situation can be encountered in practice, even for relatively small PLAs. To solve this difficulty, the boolean function F_{pleg} is expressed as: $f(x_1, x_2, ..., x_n) = \prod_{i=1}^{top} f_i$ where each f_i is represented by a BDD F_i. Figure 3a shows the pseudocode of the recursive algorithm proposed to obtain assignments that satisfy all the subfunctions, and Fig. 3b, the pseudocode of the folding algorithm.

The algorithm takes the PLA's personality matrix as input and preprocesses it. This task is performed by the function *Preprocess()* covered later, and is intended to simplify the boolean function. The function *Init_bound()* evaluates a lower bound for *P*[1]. Then, as long as a solution is not found, the BDD set whose conjunction represents the function F_{pleg} is constructed and a satisfying assignment of variables is searched. If it is found, we can fold the PLA in *P* physical columns. Else, we need to increase *P*.

SATISFY is called with three arguments. The first (*Node*) is an array and each element $Node_i$ is the upper node from F_i. The second (*Parity_index*) also has an element for each subfunction and is used to represent the parity of the complementary attribute number in a path. Lastly, *Var* is the first of the variables in the order that is used in the construction of the BDDs. The description of the function *SATISFY* uses the *Assignment* array with *n* elements (*n*, the number of variables in the function) that stores the assigned value of each variable. Initially none of the positions of *Assignment* are specified.

Both the variable ordering chosen as well as the breakdown of F_{pleg} into subfunctions are critical for the efficiency of the algorithm. The importance of the first lies in that the number of nodes in the BDD for a given function depends on the order selected for the variables.The order used here was chosen on the basis of reducing the size of the BDDs that represent the constraints described in points 4 and 5 of the previous Section. Decomposition of F_{pleg} into subfunctions should be a compromise between *top* (number of subfunctions) and the size of the individual BDDs.

Diverse techniques are incorporated in the procedure to reduce its complexity both in memory requirements as in execution time. Including these simplification strategies enables

1.*P* = ⌈# input columns / 2⌉ + ⌈# output columns /2⌉ for simple fold, and *P* = 2 for multiple fold.

```
SATISFY(Node, Parity_Index, Var)
{
  for ( i = 1; i ≤ top ; i++)
  {
     if ((Node_i == 1)&&(Parity_index_i == 0)
     return /* variable assignment does not satisfy f_i */
  }

  if (∀i ((Node_i == 1)&& (Parity_index_i == 1)))
     { The variable assignment is a solution } /* End */

  else
  {
     Assignment[Var] = 0
     Node = {Node_i = else child of Node_i if it has Var
                 as its asssociated variable
           Node_i = Node_i if not }
     Parity_index = {Parity_index_i = Parity_index_i ⊕
                 1 if the else branch has its comple-
                 mentary attribute
                 Parity_index_i = Parity_index_i if
                 not}

     SATISFY(Node, Parity_index, next Var)

     Assignment[Var] = 1
     Node = {Node_i = then child of Node_i if it has Var
                 as the associated variable
           Node_i = Node_i if not }

     SATISFY(Node, Parity_index, next Var)
  }
}
```

(a)

```
PLEPLAS(PLA)
{
  Read_PLA(PLA)
  Preprocess()
  P=Init_bound()
  While (a satisfying assignment is not found)
  {
     F = {F_i} = Build_BDD(P)
     SATISFY(Upper_Nodes_F_i, Parity_Index, first
                 variable)
     P += 1
  }
}
```

(b)

Figure 3: (a)*Pidgin_C* description of *SATISFY.*
(b) *Pidgin_C* description of the new folding algorithm.

increasing the size of the PLA to be treated with our algorithm without exceeding the practical limit of memory resources. The number of variables in the problem can be reduced, in some cases considerably, and construction of the F_{pleg} simplified by detecting columns and/or rows whose actual position is irrelevant to folding. A clear example of this is a column c_i such that $F(c_i) \cap F(c_j) \neq \emptyset \ , \forall j$. Since this column cannot be folded with any other, it need not be considered in the folding process. On one hand, the elimination of columns may lead to personality matrices with rows in which all the input are 0, which are obviously eliminated. In addition, the remaining rows may be partitioned into groups (macrorows) with the characteristic that the relative order of the rows of one same group is irrelevant for folding. Two identical rows or two rows with a dominance relation are examples of macrorows. In general, two rows r_m, r_k can not belong to the same macrorow (they are incompatible) if it exists a folding pair (c_i, c_j), such that $r_m \in F(c_i)$ and $r_k \in F(c_j)$. Two rows are compatible it they are not incompatible. Macrorows are determined generating maximum sets of pairwise compatible rows and selecting a minimum cover; that is, a subset that covers all rows. In this way, the problem of obtaining row permutation to enable a maximum fold is simplified to obtaining permutation of macrorows, whose number is in many cases inferior.

The function *Preprocess()* (Fig. 3b) analyzes the PLA matrix eliminating the irrelevant columns and partitioning the rows in a minimum number of disjointed macrorows. The above sketched algorithm developed for this last task is novel, and more general than that devised in previous approximations [3], that only contemplate equality or row dominance. This generality results in a further reduction in the number of rows involved to obtain folding.

IV. Experimental Results

A first version of the new PLA folding algorithm has been coded in C, using the BDD packet of SIS [7] to implement the construction and manipulation operations of these structures. The prototype program is called PLEPLAS. We have applied it to a group of PLAs taken from different sources. Table I shows the results obtained for some of them. Columns have been included for the number of rows (R) and columns (I) in the original PLA, as well as for its sparsity (SP). The number of columns in the folded PLA, I', both for simple and multiple folding are also shown. Time, in seconds, corresponds to a SparcStation10. Results from PLEASURE [8], an standard heuristic folding algorithm, have also been included in Table I. Half of the single foldings an two multiple ones obtained with PLEASURE are non optimum. These results show how far heuristic solutions may be from the optimum even for relatively small machines, and indicate that practical instances of the folding problem can be exactly solved without requiring much computation resources.

				PLEPLAS S		PLEPLAS M		PLEASURE S	PLEASURE M
PLA	*R*	*I*	*SP %*	I'	t	I'	t	I'	I
Hatch82	6	6	74	3	1.3	3	0.6	3	3
Hwang1	6	8	75	5	2.5	5	0.7	5	5
Con1	9	9	78	5	1.4	5	14.3	6	5
Hwang2	7	10	77	7	2.6	7	0.8	9	9
Demi84	6	10	85	5	0.9	3	1.5	6	6
adr4	75	13	74	11	0.2	9	0.1	11	9
Biswas86	16	14	85	7	66	7	500	10	7
dc2	39	15	72	11	6.6	10	19.2	11	10
alu2	68	18	82	14	4.5	13	22.2	15	13
alu3	66	18	82	14	2.8	12	1.9	14	12

Table I: Experimental Results.
S: Single Folding; M: Multiple Folding

V. CONCLUSIONS

The problem of optimum PLA folding (simple and multiple) has been formulated as a problem of boolean satisfiability, deriving a boolean function such that an assignment of variables that satisfy it defines a fold with a minimum number of columns. A fold algorithm has been developed using sophisticated logic verification techniques based on the representation of functions by BDDs. Novel reduction techniques are incorporated to treat PLAs with a higher number of rows and/or columns. Our results show that although the problem is NP and so heuristic method are required in general, optimum single and multiple folding is feasible for practical instances of the folding problem.

REFERENCES

[1] G.D. Hachtel, A.R. Newton y A.L. Sangiovanni-Vincentelli, “An Algorithm for Optimal PLA Folding”, *IEEE Trans. Computer-Aided Design*, vol. CAD-1, Jan. 1982, pp. 63-76.

[2] J.E. Lecky, J.O. Murphy y R.G. Absher, “Graph-Theoretical Algorithms for the PLA Folding Problem”, *IEEE Trans. Computer-Aided Design*, vol. CAD-8, Sept. 1989, pp. 1014-1021.

[3] A.G. Ferreira, y S.W. Song, “Achieving Optimality for Gate Matrix Layout and PLA folding: A Graph theoretic Approach”, *Integration, the VLSI Journal*, 14 (1992), pp. 173-195

[4] M. Luby, U. Vazirani y A.L. Sangiovanni-Vincentelli, “Some Theoretical Results on the Optimal PLA Folding Problem”, *Proc. IEEE Conf. on Circuits and Computers*, pp. 165-170, 1982.

[5] S. Devadas, “Optimal Layout Via Boolean Satisfiability", *Proc. IEEE ICCAD 89*, pp. 294-298, 1989.

[6] R. E. Bryant, “Graph Based Algorithms for Boolean Function Manipulation”, *IEEE Trans. on Computers*, C-35(8):677-691, August 1986.

[7] E.M. Sentovich et al., “SIS: A System for Sequential Circuit Synthesis”, *Memo UCB/ERL M92/41*, Univ. California, Berkeley, May, 1992.

[8] G. De Micheli and A.L. Sangiovanni-Vincentelli, “Multiple Constrained Folding of Programmable Logic Arrays: Theory and Applications”, *IEEE Trans. Computer-Aided Design*, vol. CAD-2, July 1983, pp. 151-167.