# TOWARD MINIMAL PLA IMPLEMENTATION OF FINITE STATE MACHINES

***Layla Abou-Hadeed***

Department of Computer Science and Automatic Control, Faculty of Engineering,
Alexandria University, Alexandria-21544, Egypt

## ABSTRACT

This paper considers the state assignment problem of finite state machines (FSMs) implemented using programmable logic arrays (PLAs). Our goal is to reduce the required area of PLA by reducing the number of columns. Rosenkrantz [1] proposed state assignments that make the logic functions unate in state variables. The proposal depends only on the number of states -defined by the FSM- not its transition table. This paper shows that taking the transition table of the machine into consideration may lead to the unateness of the input variables which leads to further reduction in number of columns. The paper also shows that relaxing the constraint on unateness in some state variables may also lead to reduction in number of columns.

***Keywords***: PLA, Finite-state machine, Unate function, State-assignment problem

## INTRODUCTION

A programmable Logic Array (PLA) is a digital integrated circuit that can realize multiple-output function. To realize m-output function of n variables, a PLA of n inputs and m outputs is required. A PLA consists of AND array to realize the selected product terms and OR array to realize the required function [2]. This type of PLA, called two-level PLA, is widely used in the synthesis of combinational and sequential circuits due to its simplicity, flexibility and regularity [3,4]. The PLA area is proportional to the product of the number of columns and the number of rows. The rows represent the product terms required to realize the transition and output functions. The columns represent the input, state and output variables as follows: the AND plane has two columns for each input and state variables and the OR plane has one column for each output and state variable. The increase of PLA area results in reduction in circuit yield, degradation of time (speed)[5], and causes a problem in power [4]. PLA area can be reduced either by finding other solution -for the multiple-output function- that results in minimal area or by PLA folding (topological minimization) [4].

This paper considers the state assignment problem of FSMs implemented using PLAs. The state assignment problem of FSMs has been widely studied [1,6-8]. It is required to find a state encoding for a state table description such that the resulting circuit representing the machine has a minimum area. For an implementation with PLA, minimizing area is obtained by decreasing the number of rows (and/or) the number of columns. Many researches aimed at minimizing the number of rows by minimizing the number of product terms as in References 7 and 10. Finding a state assignment for a machine that reduces number of columns may lead to a reduction of PLA area depending on the changes of number of rows (number of rows depends also on the state assignments). Rosenkrantz [1] proposed choosing assignments that ensures the unateness of the functions in the state variables. This can be realized by selecting assignments that have no relation. This proposal, although may lead to increase in the number of state variables, but it may reduce the total number of columns. This is because only one column (not two columns) is required for each state variable in the AND plane. The disadvantage of this method is that *n state variables are required to realize machines with* $^{n}C_{n/2}$ states. In this paper, we show that using the transition table of the machine may lead to reduction -in number of columns- better than that of Reference 1. The idea is based on trying to find an assignment that makes the function unate in input variable (if possible). The paper also shows *that by relaxing the constraint(s) of unateness in some state variables may lead to designs with less number of columns.*

The rest of the paper is organized as follows: the next section introduces some notations and definitions. Then we present the proposed approach, followed by the introduction of an example. The last section is for the conclusion.

## SOME NOTATIONS AND DEFINITIONS

Let $\delta$ denote a function that maps a present state and input into next state. For example, consider a machine that is currently at state $s_1$. If the next state, under input $i_1$. is $s_2$ then $\delta(s_1, i_1)=s_2$ .

- Let the abbreviation $A(s_i)$ denote the assignment of a state $s_i$.
- The weight of assignment is the number of 1's in this assignment.
- Let $A(s_i) < A(s_j)$ denote that each bit of $A(s_i)$ is less than or equal to the corresponding bit of $A(s_j)$.
- Let $o/p(s_i) < o/p(s_j)$ denote that the output when the current state is $s_i$ is less than or equal to the output when the current state is $s_j$ for all values of input variable.
- Let S denote the set of all states of a FSM.
- "<"-graph: it is a directed graph of nodes and edges where the nodes represent the states of the FSM and an edge from state $s_i$ to $s_j$ represents the condition $A(s_i) < A(s_j)$. The "<"-relation is transitive, i.e. if the condition $A(s_i) < A(s_j)$ exists, then there must exist some path (consists of one edge or more) from node $s_i$ to node $s_j$.
- Degree of FSM: it is the number of states of the machine.

## THE PROPOSED APPROACH

Our proposed approach allows using *assignments that may have relation* in the *form of "<" to* enlarge the degree of the machines that can be realized using n state variables. The transition table is studied to extract the relation between state assignments that must be satisfied to ensure unateness in input variables (if possible) and in (all/some of) state variables. We will consider machines with one input variable *and one output variable.*

### *Unateness in Input Variables*

*In contrast to the state variables,* the functions may/may not be unate in input variables depending on the specification of the machine. The unateness in input variable I can be realized if the following two conditions are satisfied:

- If there exists a state $s_i$ where the outputs under inputs I=0 and I=1 are 0 and 1 (1 and 0) respectively, then there does not exist a state $s_j$ whose outputs under inputs I=0 and I=1 are 1 and 0 (0 and 1) respectively.
- there exists an assignment that satisfies the following condition:
- $A(\delta(s_i,I=0)) < A(\delta(s_i,I=1))$ $\forall$ $s_i \in S$ or $A(\delta(s_i,I=1)) < A(\delta(s_i,I=0))$ $\forall$ $s_i \in S$.
- The importance of these conditions can be trivially proved from the definition of unateness of a function[9].
- If some conditions -relating to some input variable- contradict others (for example $A(s_i)<A(s_j)$ and $A(s_j)<A(s_i)$) , then there exists no assignment that leads to unateness in that variable and all conditions relating to that input variable are omitted.

### Unateness in State Variables

The state assignments -in addition to satisfy the conditions imposed by input variables- are required to satisfy some conditions to ensure that the logic functions are unate in state variables.

To ensure the unateness in state variables, the following conditions must be satisfied:

- if $o/p(s_i) < o/p(s_j)$ , then $A(s_i) < A(s_j)$.
- if $A(s_i) < A(s_j)$ then $A(\delta(s_i,I)) < A(\delta(s_j,I))$.

- It has to be noted that the second condition must be applied recursively until no new condition is added. A "<"-graph can represent the relations that must be satisfied to ensure unateness in state variables and (if possible) in input variables.

The conditions for unateness in both state variables and input variables lead to one of three cases:

(1) There exists an assignment that satisfies these conditions without increase in the number of state variables.

(2) Some conditions contradict others (for example $A(s_i)<A(s_j)$ and $A(s_j)<A(s_i)$), in this case, one or both of them must be relaxed. This leads to state assignments and functions that are not unate in some variables.

It has to be noted that relaxing the condition of unateness in input and state variables may lead to better solutions as the following theorem states.

**Theorem1**

A FSM with n state-variables - implemented using PLA- that is unate in n-1 variables has less number of columns than an n+1- state variable that are unate in all variables.

**Proof**

For the n-state-variables case, the PLA requires n-1 columns for the unate variables, in addition to two columns for the last variable. i.e. n+1 columns are required for input. Also, n columns are required to generate state variables. Consequently, 2n+1 columns are required.

For the n+1-state-variables case, the corresponding PLA requires n+1 columns for input and n+1 columns for output, giving a total of 2n+2; i.e. making the function not unate in some variables may be better than increasing the number of state variables to make the function unate in all state variables.

(3) Drawing "<"-graph that relates the states of the machine according to the conditions may indicate the existence of a path that can not be satisfied with the current number of state variables as the following theorem states:

**Theorem2**

Consider a "<"-graph as mentioned above. To get a suitable state assignment that satisfies the conditions represented in the graph, it is necessary that the length of the maximum path of the graph (in terms of number of edges) is less than or equal to the number of state variables.

**Proof**

The proof is based on the fact that the maximum possible assignments that can be linked in one path is n+1 (for n state variables) . These n+1 assignments are the assignments that have the following weights 0,1,..,n such that the hamming distance between assignments of weights k and k+1 is 1 for $0\leq k<n$.

In this case, we have the possibility of either increasing the number of state variables , or relaxing the constraints of unateness in some variables.

Example of such a path: $A(s_5)<A(s_4)$ , $A(s_4)<A(s_3)$, $A(s_3)<A(s_2)$ and $A(s_2)<A(s_1)$.

No assignment of three variables can satisfy these conditions. This is because 000 must be assigned to $s_1$ , 001 to $s_2$, 011 to $s_3$ and 111 to $s_4$. No assignment can be found for $s_5$ such that $A(s_3)<A(s_5)$ and $A(s_5)<A(s_4)$.

The proposed steps for choosing the state assignment:

1- For the input I, find the conditions that guarantee the unateness of the functions in I.

2- If some condition(s) contradicts other(s), then omit all conditions.

3- Find the conditions that guarantee the unateness of output function in state variables.

4- For the conditions resulting from step 1 and 3, apply the following condition recursively:

if $A(s_i) <A(s_j)$ then $A(\delta(s_i,I)) < A(\delta(s_j,I))\ \forall\ I$.

5- Draw the "<"-graph for the resulting conditions.

- It has to be noted that the second condition must be applied recursively until no new condition is added. A "<"-graph can represent the relations that must be satisfied to ensure unateness in state variables and (if possible) in input variables.

The conditions for unateness in both state variables and input variables lead to one of three cases:

(1) There exists an assignment that satisfies these conditions without increase in the number of state variables.

(2) Some conditions contradict others (for example $A(s_i)<A(s_j)$ and $A(s_j)<A(s_i)$), in this case, one or both of them must be relaxed. This leads to state assignments and functions that are not unate in some variables.

It has to be noted that relaxing the condition of unateness in input and state variables may lead to better solutions as the following theorem states.

**Theorem1**

A FSM with n state-variables - implemented using PLA- that is unate in n-1 variables has less number of columns than an n+1- state variable that are unate in all variables.

**Proof**

For the n-state-variables case, the PLA requires n-1 columns for the unate variables, in addition to two columns for the last variable. i.e. n+1 columns are required for input. Also, n columns are required to generate state variables. Consequently, 2n+1 columns are required.

For the n+1-state-variables case, the corresponding PLA requires n+1 columns for input and n+1 columns for output, giving a total of 2n+2; i.e. making the function not unate in some variables may be better than increasing the number of state variables to make the function unate in all state variables.

(3) Drawing "<"-graph that relates the states of the machine according to the conditions may indicate the existence of a path that can not be satisfied with the current number of state variables as the following theorem states:

**Theorem2**

Consider a "<"-graph as mentioned above. To get a suitable state assignment that satisfies the conditions represented in the graph, it is necessary that the length of the maximum path of the graph (in terms of number of edges) is less than or equal to the number of state variables.

**Proof**

The proof is based on the fact that the maximum possible assignments that can be linked in one path is n+1 (for n state variables) . These n+1 assignments are the assignments that have the following weights 0,1,..,n such that the hamming distance between assignments of weights k and k+1 is 1 for $0\leq k<n$.

In this case, we have the possibility of either increasing the number of state variables , or relaxing the constraints of unateness in some variables.

Example of such a path: $A(s_5)<A(s_4)$ , $A(s_4)<A(s_3)$, $A(s_3)<A(s_2)$ and $A(s_2)<A(s_1)$.

No assignment of three variables can satisfy these conditions. This is because 000 must be assigned to $s_1$ , 001 to $s_2$, 011 to $s_3$ and 111 to $s_4$. No assignment can be found for $s_5$ such that $A(s_3)<A(s_5)$ and $A(s_5)<A(s_4)$.

The proposed steps for choosing the state assignment:

1- For the input I, find the conditions that guarantee the unateness of the functions in I.

2- If some condition(s) contradicts other(s), then omit all conditions.

3- Find the conditions that guarantee the unateness of output function in state variables.

4- For the conditions resulting from step 1 and 3, apply the following condition recursively:

if $A(s_i) <A(s_j)$ then $A(\delta(s_i,I)) < A(\delta(s_j,I)) \ \forall$ I.

5- Draw the "<"-graph for the resulting conditions.

6- Find a suitable assignment that satisfies all the conditions.
7- If no such assignment, either relax some condition or increase the number of state variables and then find a suitable assignment.

**EXAMPLE**

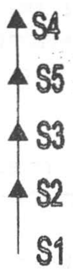
| | I=0 | I=1 |
|---|---|---|
| $s_1$ | $s_1$,0 | $s_2$,0 |
| $s_2$ | $s_1$,0 | $s_3$,0 |
| $s_3$ | $s_1$,0 | $s_4$,1 |
| $s_4$ | $s_5$,1 | $s_4$,1 |
| $s_5$ | $s_3$,1 | $s_4$,1 |

With respect to the input variable, the assignments can lead to unateness in variable I if the following conditions are satisfied
$A(s_1)<A(s_2)$, $A(s_1)<A(s_3)$, $A(s_1)<A(s_4)$, $A(s_5)<A(s_4)$, and $A(s_3)<A(s_4)$.

So, if we find an assignment that satisfies the previous conditions, then the logic to compute the output and next state will be unate in I .

With respect to state variables: the output function imposes the following conditions $A(s_1)<A(s_3)$, $A(s_2)<A(s_3)$, $A(s_3)<A(s_4)$, and $A(s_3)<A(s_5)$. Considering these conditions in additions to the above conditions for unateness in input variable leads to the following conditions: $A(s_1)<A(s_2)$, $A(s_2)<A(s_3)$, $A(s_3)<A(s_4)$, $A(s_1)<A(s_5)$, $A(s_1)<A(s_3)$, $A(s_2)<A(s_4)$, $A(s_3)<A(s_5)$ , $A(s_1)<A(s_4)$,and $A(s_5)<A(s_4)$.

The corresponding "<"-graph is as shown in what follows:



No solutions of three state variables can satisfy the unateness in both input variable and state variables. So, one have the choice of either increasing the number of state variables to 4 or relaxing the constraint of unateness in some variable(s). The first case will increase the number of columns by 2 (one for input and one for output) and the other case will increase the number of columns by 1 (for input to get the complement). So, the final solution is preferred and the assignments are as follows: $s_1$=000, $s_2$=001, $s_3$=101,$s_4$=111, and $s_5$=110. The resulting functions are unate in I, x1, x2 but not in x3. This leads to number of columns=9 and number of rows=6 (shown in the following figure), which gives 54 units which is better than that of [1] (66 units).

The notation used in the figure is as follows: 1 denotes connection and 0 denotes no connection.

| | AND | | | | | OR | | | |
|---|---|---|---|---|---|---|---|---|---|
| $X_3$ I | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| $X_2$ $X_3$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| I | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $X_1$ $\overline{X}_3$ | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $X_2$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| $X_1$ I | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| | I | $X_1$ | $X_2$ | $X_3$ | $\overline{X}_3$ | $X_1$ | $X_2$ | $X_3$ | 0 |

## CONCLUSION

This paper shows the possibility of finding better assignment than the general assignment proposed in Reference 1 by considering the transition table of the machine. The idea is to try to ensure the unateness in input variables (if possible). It also tries to relax the condition of unatenss in some state variables if this can lead to less number of columns. The paper also discusses some guide lines that must be considered to get suitable assignments, but finding an efficient algorithm for such searching problem is still an open point.

## REFERENCES

1. D.J. Rosenkrantz, “Half-Hot State Assignments for Finite State Machines”, IEEE Trans. On Comp., Vol. 39, no. 5, pp. 700-702, (1990).
2. C.H. Roth, “Fundamentals of Logic Design”, 4th Edition West Publishing Company, pp. 234, (1992).
3. M.J. Ciesielski and S. Yang, “PLADE: A Two-Stage PLA Decomposition”, IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, Vol. 11, No. 8, pp. 943-954, (1992).
4. C.Y. Liu and K.K. Saluja, "An Efficient Algorithm for Bipartite PLA Folding: IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 12, No. 12, pp. 1839-1847, (1993).
5. G.D. Hachtel, A.R. Newton and A.L. Sangiovarrni-Vincentelli, "An Algorithm for Optimal PLA Folding" IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. CAD-1, No. 2, pp. 63-77, (1982).
6. I.Pomeranz and K.T. Cheng, "STOIC State Assignment Based on Output/ Input Functions"IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 12, No. 8, pp. 1123-1131, (1993).
7. T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-level Logic Implementation", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 9, No. 9, pp. 905-924, (1990).
8. C.R. Mohan and P.P. Chakrabarti, "Earth: Combined State Assignment of PLA-Based FSM's Targeting Area and Testability", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 7, pp. 727-731, (1996).
9. A.D. Friedman, "Logical Design of Digital Systems", Pitman Publishing Limited, pp. 148-149, (1975).
10. G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal State Assignment for Finite State Machines", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. CAD-4, No. 3, pp. 269-284, (1985).

Received October 27. 1998
Accepted September 18. 1999

# نحو تنفيذ –أقل تكلفة– للآلات محدودة الحالات باستخدام مصفوفة منطقية قابلة للبرمجة

ليلى أبو حديد
قسم الالات الحاسبة والتحكم الآلي - جامعة الاسكندرية

## ملخص البحث

يناقش هذا البحث موضوع تخصيص الحالة –للآلات محدودة الحالات– المنفذ باستخدام مصفوفات منطقية قابلة للبرمجة. وهدف هذا البحث هو تخفيض المساحة المطلوبة للمصفوفة المستخدمة وذلك بتخفيض عدد الأعمدة. وقد اقترح أحد الباحثين طريقة لتخصيص الحالة تعتمد على جعل الدوال المنطقية أحادية فى متغيرات الحالة. وهذا الاقتراح يعتمد فقط على عدد الحالات للآلة وليس على جدول الانتقال الخاص بها.

ويبين هذا البحث أنه بمعلومية جدول الانتقال الخاص بالآلة، يمكن فى بعض الأحيان جعل الدوال المنطقية أحادية فى متغيرات الادخال وهذا يؤدى بدوره الى تخفيض أكثر فى عدد الأعمدة. كما يبين البحث كذلك أن التغاضى عن شرط الأحادية فى بعض متغيرات الحالة يمكن كذلك أن يؤدى الى تخفيض فى عدد الأعمدة.