Graph-Based PLA Minimization

Yi Ma

Rupesh Jain
Final Report

1. Problem Description

The objective of PLA (programmable logic array) minimization is to minimize the area requirements of PLA’s to implement logic functions. The area of a PLA is proportional to the number of product terms in a two level canonical expression of the functions to be implemented. When realizing an m-output function \((f_0, f_1, \ldots, f_{m-1})\), we have the option to implement either \(f_i\) or its complement for each output 0 to \(m-1\). Through assigning different phases for each output, the number of product terms needed to implement the function using the PLA can be reduced. The PLA output phase assignment problem is to find an optimal output phase assignment \((v_0, v_1, \ldots, v_{m-1})\). The PLA minimization problem can be modeled as a graph and the maximum clique of this graph is used to decrease the number of product terms.

1. Method Explanation

Given a logic function \(F\), we can use ESPRESSO (a software tool) to get the double-phase PLA, for example

\[
\begin{array}{cccccccc}
 x_1 & x_2 & x_3 & x_4 & f_0 & f_1 & f_2 & f_0 & f_1 & f_2 \\
 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\end{array}
\]

AND-matrix \hspace{1cm} OR-matrix

The number of rows equals to the number of product terms. And i-th row stands for product term \(T_i\).

In this example, let \(v = (1,1,0)\) be an assignment vector for the function, then we use \((f_0, f_1, f_2)\) to implement the function \(F\). We can see from the following figure that we just need four product terms \(T_1, T_2, T_3, T_4\) to implement the function. It is less than five product terms which are needed if we directly use \((f_0, f_1, f_2)\) to implement \(F\).
Every product term has its assignment condition, consider the OR-matrix of the example we used just now.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & f_0 & f_1 & f_2 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{array}{cccccccc} T & f_0 & f_1 & f_2 & f_0 & f_1 & f_2 \\ T_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ T_2 & 0 & 1 & 0 & 1 & 0 & 0 \\ T_3 & 0 & 1 & 0 & 0 & 0 & 0 \\ T_4 & 1 & 0 & 0 & 0 & 1 & 0 \\ T_5 & 0 & 0 & 1 & 0 & 0 & 0 \\ T_6 & 0 & 0 & 0 & 1 & 0 & 0 \\ T_7 & 0 & 0 & 0 & 0 & 1 & 0 \\ T_8 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

It can be seen that implementing $f_0$ requires product term $T_1$. Hence, $f_0$ has not to be the phase assignment for the first output of the function in order to reduce $T_1$ from DP. Therefore, the phase assignment to the first output is represented as 0, denoting $\overline{f_0}$. Since both $f_1$ and $\overline{f_1}$ do not require product term $T_1$, $T_1$ is reduced from DP whether the phase assignment to the second output of the function is assigned to 0 or 1. In this case, the phase assignment to the output is represented as x (don’t care). Similarly, the phase assignment to the third output of the function is also x. The assignment condition to reduce product term $T_1$ can be represented as vector (0, x, x). Let $V = (v_0, v_1, \ldots, v_{m-1})$ be the assignment condition for $T_1$.

$$v_j = \begin{cases} 0 & \text{if } f_j = 1 \\ 1 & \text{if } \overline{f_j} = 1 \\ x & \text{otherwise} \end{cases}$$

<table>
<thead>
<tr>
<th>No.</th>
<th>$f_0$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$\overline{f_0}$</th>
<th>$\overline{f_1}$</th>
<th>$\overline{f_2}$</th>
<th>$v_0$</th>
<th>$v_1$</th>
<th>$v_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$T_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$T_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$T_4$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$T_5$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$T_6$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$T_7$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$T_8$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
After find the assignment condition for each product term, we explore the relations between these assignment conditions. We intersect (\( \cap \)) two assignment conditions \( \alpha \) and \( \beta \) according to the coordinate intersection and the rules:

\[
\begin{align*}
(1) \quad \alpha \cap \beta &= \phi \quad \text{(empty) if any coordinate intersection is } \phi \ . \\
(2) \quad \alpha \cap \beta &= \text{the vector formed the respective coordinate intersection if (1) does not hold.}
\end{align*}
\]

We say two assignment conditions \( \alpha \) and \( \beta \) are compatible if \( \alpha \cap \beta \neq \phi \). Then the product terms corresponding to the two assignment conditions can be simultaneously reduced from the double phase PLA by using \( \alpha \cap \beta \) as assignment vectors. Now our task is to find the maximum number of compatible assignment conditions since their intersection will give us the optimal phase assignment.

We represent the assignment conditions and their compatibility by a graph \( G = (V, E) \), where \( V \) is the set of assignment conditions for each product term in DP, and a pair of vertices \( (i, j) \in E \) iff condition \( i \) and \( j \) are compatible.

The compatibility graph of our former example is as follow:

![Compatibility Graph](image)

The maximum clique in the compatibility graph will tell us the maximum number of compatible assignment conditions. Therefore, what we need to do to solve the original problem is to find the maximum clique in the compatibility graph.

Finding the maximum clique in a graph \( G \) is equivalent to finding the maximum
independence set in $G$’s complement graph $\overline{G}$. For this problem, it is better to find the maximum independence set in the compatibility graph’s complement graph, which is called incompatibility graph. Because experimentally, it is found that compatibility graphs of double phase PLA’s often have more edges than their complementary graphs. Moreover, given two assignment conditions $\alpha$ and $\beta$, deciding whether they are incompatible, $\alpha \cap \beta = \phi$, is usually faster than deciding whether they are compatible, $\alpha \cap \beta \neq \phi$.

As a result, we need to construct incompatibility graph rather than compatibility graph for a given double phase PLA (DP). The incompatibility graph of DP is defined as $G = (V, E)$, where $V$ is the set of assignment conditions for each product term in DP, and a pair of vertices $(i, j) \in E$ iff the conditions $i$ and $j$ are incompatible, $i \cap j = \phi$.

The incompatibility graph of the example is:

![Diagram of incompatibility graph](image)

Finding the maximum cliques or independent sets in random graphs is shown to be NP-complete. However, various polynomial time algorithms have been proposed. One of them is Johnson’s simple greedy algorithm. It proceeds as follows. First the vertex with the largest degree is selected and added to a clique. This vertex and any vertices which are adjacent to it are deleted from the graph. This process is repeated until every vertex in the graph has been deleted. The set of selected vertices is guaranteed to be a clique.

In order to find a maximum independent set in incompatibility graph, we use the dual of Johnson’s algorithm. Pick the vertex with the smallest degree and add it to a independent set. This vertex and any vertices which are adjacent to it are deleted from the graph. This process is repeated until every vertex in the graph has been deleted. The set of selected vertices is guaranteed to be an independent set.
2. Implementation Details

We use Visual C++ 6.0.

- **Step 1**: Read input from the file “input.txt”.
  The first line of the input file is the number of product terms in the DP (OR-matrix of a double phase PLA). It is at most 50 because we limited our incompatibility graph size to 50*50. We read it into a global variable \( T \).
  The second line is the number of outputs of the logic function represented by the DP. It is at most 50 too. We read it into a global variable \( f \).
  From the third line, it is the DP. We read it into a global array \( \text{dpmatrix}[50][100] \).
  In a real problem, we are supposed to input a logic function. However, since there is an existed tool ESPRESSO, which can be used to obtain DP from a logic function, we input the DP directly assuming that we can get it from ESPRESSO.

- **Step 2**: Compute the assignment condition for every product term according to the method explained in section 2. We store the conditions in a global array \( \text{assigncondition}[50][50] \).
  The function we use for this step is \( \text{int assign(int, int)} \).

- **Step 3**: Construct the incompatibility graph basing on the assignment conditions we get from step 2. We store the adjacent matrix of the incompatibility graph in \( \text{incompatiblegraph}[50][50] \).
  The function we use for this step is \( \text{int incompatible(int, int)} \).

- **Step 4**: Find a maximum independent set in the incompatibility graph and obtain the assignment vector according to the independent set. This is the main part of your project.
  We have mentioned the algorithm in section 2. now let’s get more specific through pseudo code.

```plaintext
Procedure GOPA;
    Input: the incompatibility graph \( G \) of a double phase PLA;
    Output: a near optimal assignment vector \( v \);
BEGIN
1. Let \( v = (x, \cdots, x) \), where \( x \) represent don’t care;
2. Select a vertex \( V_i \) which has the minimum degree in \( G \);
3. Let \( v = v \setminus V_i \);
4. Delete \( V_i \) and its adjacent vertices;
5. If \( G \) becomes an empty graph, Return(\( v \));
END GOPA.
```

The corresponding function in our code is \( \text{void gopa(int graph[50][50], int s[], int nodes[])} \).
When selecting the node with smallest degree, if several vertices have the
same minimum degree, the vertex which corresponds to the product term with maximum number of “--” (don’t care) is arbitrarily selected in the paper. However, this information is contained in the AND-matrix of the double phase PLA. Since we just input the OR-matrix of the double phase PLA, we can’t get this. Thus, we pick the one which has the smallest label in the adjacent matrix.

According to the paper, GOPA algorithm performs not very well if the vertex of minimum degree is not included in a large independent set. A simple improvement can be effected by selecting each of the vertices in the graph in turn as the initial vertex in the independent set. This modification adds one level of backtracking to GOPA, and is referred to herein as the modified graph-based output phase assignment algorithm (MGOPA).

The corresponding function in our code is `int mgopa(int graph[50][50], int s[])`.

When implementing GOPA and MGOPA, we encountered a small problem at first. We represented deleting a node just by writing the entries of that node in adjacent matrix into 0s. But it can not distinguish that a node has degree 0 or it has been deleted. So we add an array `nodes[50]` to record whether a node has been deleted or not.

### 3. Result

We apply our program to the example in section 2 to test it. It gives us the assignment vector (0, 1, 0). Four product terms are needed based on this assignment vector, which is consistent with the result of the paper.

We are not able to find more suitable inputs to test our program. However, we trace the program step by step through execution. We believe our program works very well.

### 4. References