An Enhanced Local Covering Approach for Minimization of Multiple-Valued Input Binary-Valued Output Functions

MOINUL ISLAM ZABER\textsuperscript{1}, HAFIZ MD. HASAN BABU\textsuperscript{2}

Department of Computer Science and Engineering, BRAC University\textsuperscript{1}, University of Dhaka\textsuperscript{2}, Dhaka BANGLADESH

Abstract: - Local covering technique of Multiple-Valued Input Binary-Valued Output Functions (MVI\textsuperscript{BVO}) is inherently a divide and conquer technique where the minimization process works in ‘expansion’ and ‘selection’ phases. In order to get considerably minimized output proper base minterms should be chosen first. In this paper we propose an enhancement of the existing technique that successfully finds out the essential primes with least number of passes. Result of the experiment shows superior result than the existing techniques.

Key-Words: - MVL, Local Cover, Canonical Cube, SOP

1 Introduction:
Simplification of sum of products expressions is of great importance in logic synthesis. In total computation time for logic synthesis, the ratio of the time spent for simplifications for SOPs is directly related to the simplification of PLAs. In this context the necessity for the use of multiple-valued logic (MVL) is gaining its valued importance day by day. The interconnection complexity of two-valued functions both in chip and between chips is reduced effectively by the adept use of MVL. These functions can be of great use to minimize decoded PLA’s and in the realm of sequential circuits and networks. Among the heuristic methods to find out the minimum cover of the MVITVO functions MINI and ESPRESSO-IIC [4] are very well known. In these methods a near optimum cover of the function \( F \) to be minimized is achieved through iterative improvement by reshaping and reducing the initial cover of it. A slightly different approach to these heuristics is the approach of the local covering technique, where the whole process starts from a properly chosen base minterm. An improved version of this technique due to Caruso, [5] is as follows: First a set of sub functions of the function to be minimized is built (expansion process). Then one or more primes are selected from the ones of each sub-function (selection process). In the end a union of all the selected primes is done which forms a cover of the function \( F \).

In this research we have tried to hold down the racing computational time by emphasizing in finding out the ‘best minterm’ (the minterm which has the fewest number of adjacent minterms) and at the same time enhanced the probability of detecting and selecting the essential primes while expanding. This new algorithm preserves the notion of the previous ones and we are made aware of the lower bound of primes in the minimum cover of the given function. Here we have worked in two phases firstly in finding efficient grouping of the Boolean variables and secondly by finding fast the viable cubes with suitable minterms. In concise our algorithm is fast in computation and prudent in keeping the functions as minimized as possible.

2 Basic Definitions:
A multiple valued input and binary valued output function is a mapping \( F(X_1, X_2, X_3, \ldots, X_n) : P_1 \times P_2 \times P_3 \times \ldots \times P_n \rightarrow B \), where \( X_i \) is a multiple-valued variable, \( B = \{0, 1, *\} \), and \( P_i = \{0, 1, \ldots, p_i - 1\} \), \( (p_i \geq 2) \) is a set of values that this variable may assume. Symbol ‘*’ denotes the value 1 or 0. A product of \( n \) constants \( a_1 \times a_2 \times \ldots \times a_n \) with \( a_i \in P_i \) is a minterm.

The ON set of \( F \) is formed by all the minterms for which the function takes the value 1. Similarly, the OFF set is formed by all the minterms for which the function takes the value 0. And the Don’t Care
A prime implicant, p, is a product term which implies no other cube of the function. An essential prime implicant denotes a prime implicant that covers at least one standard product term (not a Don’t Care term) that cannot be covered by any other prime implicant. The essential prime implicants, therefore must be included to get a minimum cover of the function. A distinguished minterm is a minterm that is covered by only one prime implicants. The essential primes cover one or more distinguished minterms.

If $C_1$ and $C_2$ are two cubes then $C_1$ is said to imply $C_2$ if $C_1 \subseteq C_2$.

A supercube of the cubes $S^i$, $S^2$, . . . , $S^h$ is defined as follows

$$\text{Supercube}(S^i, S^2, . . . , S^h) = (S^i \cup S^2 \cup \ldots \cup S^h) \times \ldots \times (S^i \cup S^2 \cup \ldots \cup S^h)$$

The minimization procedure for MVITVO functions, consists of the following steps:

(a) The determination of all prime implicants of the function.
(b) Finding out the essential prime implicants.
(c) From the remaining prime implicants a minimum set is chosen so that together with the essential prime implicants they cover the function.

3 Algorithms for Minimizing the Multiple-Valued Functions:

In the local cover approach, the minimization process has two steps namely expansion and selection. The expansion process creates a set of sub functions of the function $F$ to be minimized. Each sub function consists of the minterms of $F$ covered by all the primes that cover a base minterm properly chosen. In fact the objective of this step is to expand the base minterm so to result in the SOP where no cubes can be expanded any more. The cubes obtained in this way represent a prime implicant. In the second step, namely the selection step prime implicants are chosen from each sub-function so that the union of all the chosen primes form an irredundant cover of $F$.

The success of the technique depends largely on the number of sub-functions. The larger the set of sub-functions the closer the found cover is to the minimum. Moreover the detection of essential primes as early as possible plays a vital part in decreasing the computational time. Both of these two essential criteria can be achieved by choosing base minterms from the minterms with the smallest number of adjacent minterms. In this process we can...
easily detect the essential primes from the sub-functions containing only one prime as each essential prime is unique and base minterms with smallest number of adjacent are distinguished minterms.

Our proposal to the improvement of the local cover technique is addition of another procedure called ‘cube-rearrangement’ which helps to find the next best minterm to be chosen as a base minterm from the potential canonical cubes (PCC) so that the set of sub functions will increase and essential primes are detected in the earliest phase of the computation. In order to realize this procedure the given expression is preferred to be in canonical form. The expansion process in this algorithm is done by circular shifting the cubes. In case of canonical cubes it generates a set of minterms adjacent to the original cube. Our motivation to the new procedure lies in the fact that the minterm with the smallest number of adjacents would reside in the canonical cubes with smallest number of adjacents. So if we can arrange the given canonical cubes with respect to their adjacency then it would be less time consuming when searching for the minterms with the smallest number of adjacents. Our algorithm uses a table of indices for all the canonical cubes and rearranges it according to their number of adjacents (fig. 4).

The procedure first checks out the $X_i^i$’s (1<i<m, m = number of different multiple values) and counts the number of distinct values (i.e. 0, 1, 2, 3’s) for each $X_i^i$’s (i ∈ [0, n-1], n is the number of literals) of all the canonical cubes. Then we update the index table of the canonical cubes according to the weight of distinct values( a weight here is the number of occurrences of that value). We perform the same procedure for each $X_j^j$’s (1<j<n-1, n =number of literals). In this way we come to a point when the table cannot be updated anymore and this is when we have succeeded in rearranging properly the cubes such that we can get the minterms with the smallest adjacents just by consulting this table sequentially.

**Algorithm 1:**

Rearrange canonical cubes() {
    table={has the initial arrangement of the canonical cubes};
    pass =1, start =1, end =|table|
    Rearrange (pass, start, end)
    {
        a[] be an array, It is initialized to zero in every pass.
        For (start=1 to end)
        {
            Count the number of distinct values $X_{pass}^i$ and put them in their corresponding positions in ‘a’.
            If (for all elements k of a[], a[k] is not 0 )
            {
                Rearrange corresponding indices of table with respect to the count values according to the least occurred values of a[ ].
                end = start + K
                rearrange (pass+1, start, end)
            }
        }
    }
}

**Example 1:**

Let, 5 canonical cubes of $F(X_1, X_2, X_3, ..., X_n)$: $A_1 \times A_2 \times A_3 \times \cdots \times A_n \rightarrow B$ ,

- $A_1 = X_1^{[1]} X_2^{[0]} X_3^{[1]} X_4^{[1,2]}$
- $A_2 = X_1^{[0]} X_2^{[1]} X_3^{[0]} X_4^{[1,2]}$
- $A_3 = X_1^{[0]} X_2^{[0]} X_3^{[3]} X_4^{[1,3]}$
- $A_4 = X_1^{[2]} X_2^{[2]} X_3^{[1]} X_4^{[1]}$
- $A_5 = X_1^{[1]} X_2^{[0]} X_3^{[0]} X_4^{[1,2]}$

If we choose base minterm a= $X_1^{[1]} X_2^{[0]} X_3^{[1]} X_4^{[1]}$ from the canonical cube $A_1$ we get an Adjacent cube $A_5$ but if we choose $A_4$ and a = $X_1^{[2]} X_2^{[2]} X_3^{[1]} X_4^{[1]}$ then we get no adjacent. In order to find the minterms with the smallest adjacent we have to rearrange the canonical cubes.

The rearranging procedure of canonical cubes in different passes of the Algorithm 2 is shown in the figure 2.

Each sub-function consists of the canonical cubes which are adjacent to the base minterm properly chosen from the function F.

Let, $F=\{ A_1, A_2, A_3 \}$ where, $A_1 = X_1^{[0]} X_2^{[0]} X_3^{[1]} X_4^{[0,1,2]}$, $A_2 = X_1^{[3]} X_2^{[1]} X_3^{[2]} X_4^{[1,2]}$, $A_3 = X_1^{[0]} X_2^{[3]} X_3^{[0]} X_4^{[0,1,2]}$ and base minterm a= $X_1^{[0]} X_2^{[0]} X_3^{[1]} X_4^{[0]}$ then after performing the expansion procedure, we get two sub-functions P and Q of F, where P has { $A_1, A_3$ } and Q has $A_2$.

In this technique we first generate a supercube S by computing from the base minterm a and all the
minterms of \( F \) that are adjacent to \( a \), using circular shift operation. Now the using base minterm \( a \) and the supercube \( S \), we generate the canonical cubes for the sub-functions to be built. The selection procedure processes one at a time the sub-functions of an irredundant set. Hence the Algorithm for the Local cover technique can be described as follows:

**Algorithm 2:**

Local cover ()

\[
\begin{align*}
F \text{ is the Multiple-Valued function to be minimized.} \\
\text{rearrange_canonical_cubes () /* Rearrange} \\
\text{indices for the canonical cubes of} \ F \text{ in the} \\
\text{index table (algorithm 2). */} \\
\text{Expansion () /* creates sub- functions with} \\
\text{primes covering the base minterm}\ (a) \text{ (algorithm 3). */} \\
\text{Selection () /* selects primes from each sub-} \\
\text{function and forms an irredundant cover of} \ F \text{ (algorithm 7). */} \\
\text{*/perform the union of all the chosen primes from} \\
\text{each sub-function so that an irredundant set of primes is found. */}
\end{align*}
\]

**Algorithm 3:**

Expansion ()

\[
\begin{align*}
a \text{ is the base minterm.} \\
\text{do{} If (there is more than one minterms left, which} \\
\text{has not yet been included in the sub-} \\
\text{function) } \\
\text{Look up the index table to get the cube for} \\
\text{the base minterm}\ a, \text{ which has the smallest} \\
\text{number of adjacent minterms.} \\
\text{}} \text{generate_subfunction(a, F)/*Create sub-} \\
\text{function associated with the base minterm}\ a, \text{ chosen (algorithm 4)*/}
\end{align*}
\]

**Algorithm 4:**

generate_subfunction(a, F) {
\[
\begin{align*}
\text{/*P={P_1, P_2, ..., P_n} is a subfunction of} \ F \text{ and}\ P_i \\
\text{0<i<n are canonical cubes of}\ P.} \\
\text{S is a supercube of}\ P. \text{ */} \\
\text{P = A /*A is the canonical cube of}\ F \text{ covering}\ a */ \\
\text{K=1; R= Φ , I={0};} \\
\text{S=generate_supercube (a,F). /* (algorithm 5) */} \\
\text{While(k<=|P|) {}} \\
\text{for i=i_1+1to n-1 {}} \\
\text{for j=1 to |S_i|-1 {}} \\
\text{B' , B" =Check_B(B,a,R,F);} \\
\text{if (B' ≠ Φ ) P=P+{B'} , I={i};} \\
\text{if (B" ≠ Φ ) P=P+{B"}; } \\
\text{k=k+1;} \\
\text{return (P, R);} \\
\end{align*}
\]

**Algorithm 5:**

generate_supercube(a, F) {
\[
\begin{align*}
\text{for i=1 to n-1 { } S_i = \{a_i\} \\
\text{for } j= 1 \text{ to } |U_i| -1 \text{ { /* } U_i = \bigcup_{k=1}^{n} A_k \text{ */} \\
\text{a → b /* produce b by circular shifting a */} \\
\text{if (b ∈ F) } S_i = S_i + \{b_i\} \} \\
\text{}} \\
\text{S_n = A_n /* where A is the canonical cube of}\ F \text{ covering}\ a */. \} \\
\end{align*}
\]

**Example 2:** Construction of Supercube.:
If we select the base minterm \( a={0} \times {0} \times {1} \times {0} \) from the cube \( A'= \{0\} \times \{0\} \times \{1\} \times \{0,1,2\} \) of Fig 1; then the supercube we get from the algorithm 7 and the iteration processes are described in the table in Fig 2.

<table>
<thead>
<tr>
<th>Iteration of i</th>
<th>S_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{0,1}</td>
</tr>
<tr>
<td>2</td>
<td>{0,2,3}</td>
</tr>
<tr>
<td>3</td>
<td>{1,3}</td>
</tr>
</tbody>
</table>

Here \( S_P=A_4={0,1,2} \), therefore the supercube is \( S= {0,1} \times {0,2,3} \times {1,3} \times {0,1,2} \)

**Fig. 3.** Construction of Supercube

**Example 3:** Generation of sub-function:
The Algorithm sub_function() builds the sub-function associated with a base minterm \( a = \{0\} \times \{0\} \times \{1\} \times \{0\} \) and therefore we get the on multiple-valued ON set cube \( P \) and OFF set cube \( R \) after running this algorithm we obtain \( P \) and \( R \) as follows

**Algorithm 6:**

Check_B (B, a R, F) {
  /* B is a canonical cube of F */
  D = SuperCube (a, B);
  For each \( r \in R \) such that \( rD \neq \emptyset \)
  \( D = D \# r; \)
  \( B_n = D_n; B' = \emptyset; B'' = B; \)
  If \( a_n \in B \)
  \( B' = C \cap B; B'' = B \# a; \)
  return \( B', B''; \)
}

The routine Check_B reduces \( D \) so that \( D \cap P_{OFF} = \emptyset \). Here \( D = \text{SuperCube} \) (a, B) =P and \( P_{OFF} \) is the offset of P. Since \( P_{OFF} \) is not available, it uses a subset of it \( R \) built during the generation of \( P \).

\[ p_1^1 = \{0\} \times \{0\} \times \{1\} \times \{1\} \]
\[ p_2^1 = \{1\} \times \{0\} \times \{1\} \times \{1\} \]
\[ p_3^1 = \{0\} \times \{1\} \times \{1\} \times \{1\} \]
\[ p_4^1 = \{0\} \times \{0\} \times \{1\} \times \{0\} \]
\[ p_5^1 = \{1\} \times \{1\} \times \{1\} \times \{0\} \]
\[ p_6^1 = \{1\} \times \{1\} \times \{1\} \times \{0\} \]
\[ p_7^1 = \{0\} \times \{0\} \times \{0\} \times \{1\} \]
\[ p_8^1 = \{1\} \times \{1\} \times \{1\} \times \{1\} \]
\[ p_9^1 = \{0\} \times \{0\} \times \{1\} \times \{0\} \]
\[ p_{10}^1 = \{1\} \times \{1\} \times \{1\} \times \{0\} \]

**Algorithm 7:**

Selection (S, a, R) {
  /* Select primes from each sub-function. */
  C = \{S\};
  For \( h = 1 \) to \( |R| \) {
    \( C'' = C'' + \{ C^k \}; \)
    Else {
      For \( i = 1 \) to \( n \)
      if \( (a_i \notin r_i^h) \)
        \( C'' = C'' + \{ C^k \# r^i \}; \)
      Delete each cube
      of \( C' \) implying a cube of \( C'' \);
      \( C = C'' + \{ C^k \}; \)
      return \( C; \)
    }
}

**Example 4:** Deriving the primes of a sub-functions:

Algorithm 8 [2] generate the set of cubes \( C \) that can be build by computing the set \( S \# R \) and deleting every cube that implies one of the other yielded cubes or does not cover the base minterm. Initially \( C \) contains only \( S \). The inner loop processes one at a time the cubes of \( C \). Let \( C^h \) be the cube under processing. If \( C^h \) does not intersect \( R^h \), it is inserted in \( C'' \), otherwise, each cube \( C^h \# R^h \) that covers the base minterm is inserted in \( C' \). Then each cube of \( C' \) implying a cube of \( C'' \) is deleted and a new \( C \) is formed by the residual \( C' \) and \( C'' \). This process is repeated for such a \( C \) in the next iteration of the outer loop. After running this algorithm we get the primes which are as follows

\[ C^1 = \{0\} \times \{0,2\} \times \{1,3\} \times \{0,1,2\} \]
\[ C^2 = \{0,1\} \times \{0\} \times \{1\} \times \{0,1,2\} \]
\[ C^3 = \{0,1\} \times \{0\} \times \{1,3\} \times \{0,1\} \]
\[ C^4 = \{0\} \times \{0,2,3\} \times \{1,3\} \times \{0\} \]
\[ C^5 = \{0,1\} \times \{0,3\} \times \{1,3\} \times \{0\} \]

Espresso and MINI need the preliminary generation of the OFF set of the given function. Unfortunately, there exist functions for which such a set is exceedingly large. In some cases, it is possible to overcome such a drawback by using a reduced OFF set [8]. The referred work is focused on minimization of binary-valued functions with single output. Our local cover algorithm uses a subset of the OFF set of a subfunction both to build the subfunction itself and extract primes from it. However, such a subset does not coincide with the reduced OFF set introduced in [8]. Consider for instance, the following example drawn from [8]

\[ F_{ON} = a'b'cd + a'b'c'd' \]
\[ F_{OFF} = ab' + a'b + ac + cd' \]
\[ F_{DC} = a'b'c'd + abc'd \]

The reduced OFF set associated with \( a'b'c'd' \) is \( a+b+c \). Whereas the set \( R \) yielded by generate_sub_function() by expanding \( a'b'c'd' \) is empty. In fact, the yielded subfunction holds only
one essential prime of $F$; i.e. $a'b'c'$. $a'b'c'd'$ is a distinguished minterm of such a prime.

4 Experimental Results:

Our proposed algorithms for grouping and minimization processes have been written using the language C and tested extensively on Windows Workstations. The experimental results given below have been received from an Intel® Pentium® III Mobile CPU 1000 MHz under Microsoft Windows XP professional edition.

The table 1 shows the superiority of our grouping scheme using the enhanced assignment graph (EAG). And the results of the comparison of the algorithm proposed by [2] and our proposed one is shown in table 2. It can be understood that in most of the cases our improvement resulted in decrease in the number of products and computation time.

5 Conclusion:

New Boolean variable assignment algorithm and minimization techniques have been proposed, so that both the total computation time and number of products decreases. Our algorithmic extension to [2] has been proven to be efficient in detecting and selecting the essential prime implicants as well as furnishing the lower bound on the number of prime implicants in the first phase of the computation process. The new concepts of ‘enhanced assignment graph’, ‘use of Hamiltonian path’ in finding the best pairs, and the technique of ‘cube rearrangement’ are proven to be efficient in step-by-step minimization process. Along with these some heuristics used in different phases of expansion and selection has nevertheless improved the quality of the whole technique. The comparison of our proposed technique with the algorithm proposed by Caruso [2] has proved its superiority.

References


<table>
<thead>
<tr>
<th>Function</th>
<th>Standard PLA Proposed algorithm</th>
<th>Decoded PLA Proposed algorithm</th>
<th>Algorithm [2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clip</td>
<td>np</td>
<td>CPU time (Sec.)</td>
<td>np</td>
</tr>
<tr>
<td>Con1</td>
<td>9</td>
<td>0.012</td>
<td>9</td>
</tr>
<tr>
<td>9sym</td>
<td>87</td>
<td>4.83</td>
<td>92</td>
</tr>
<tr>
<td>Rd53</td>
<td>31</td>
<td>0.007</td>
<td>31</td>
</tr>
<tr>
<td>Rd73</td>
<td>125</td>
<td>0.031</td>
<td>127</td>
</tr>
<tr>
<td>Bw</td>
<td>20</td>
<td>0.027</td>
<td>22</td>
</tr>
<tr>
<td>Z5xp1</td>
<td>61</td>
<td>0.459</td>
<td>63</td>
</tr>
</tbody>
</table>

np: number of products

Table 1. Comparison of the proposed algorithm with the algorithm [2].