An Algorithm for Minimising the Number of Test Cycles

A.A.Diwan
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay 400076.

Abstract

The scan-path method for testing a VLSI circuit uses a shift register to store the test vectors [1], and a sequence of test patterns is applied by shifting in new patterns one bit at a time. In this paper we present an algorithm to find the order in which the test patterns should be applied in order to minimise the number of shift operations required. The algorithm can be shown to be optimal under certain conditions.

1 Introduction

Testing of circuits is one of the most important steps in the production of VLSI circuits. The usual approach of solving this problem is to apply a set of test vectors to the inputs of the circuit under test and verify the outputs. The problem of finding an appropriate set of test vectors, which will cover as many faults as possible, is itself difficult. In this paper, we consider the problem of applying the test vectors to the inputs of the circuit under test.

The scan-path technique for testing a circuit uses a shift register for storing the test patterns. A new pattern is applied to the inputs of the circuit by shifting in n bits in the shift register. Successively shifting in the appropriate bits, all the test patterns can be applied to the circuit under test. This scheme is widely used, irrespective of the number of inputs to the circuit [1].

One drawback of this approach, is that for applying a new test vector, as many as n shifts may be required, where n is the number of inputs to the circuit. This would make the testing process very slow and expensive. However, by applying the test vectors in a proper order, the number of shifts required can be reduced considerably, by a factor of \( \log n \) in some cases. In this paper we present an algorithm to find the order in which the test vectors must be applied to the circuit, in order to minimise the number of shifts required. The algorithm can be shown to be optimal under certain conditions.

In the next section we formulate the problem in terms of strings and graphs. In section 3, we present the algorithm and prove the optimality of the algorithm. In section 4 we present some computational results and consider some extensions of the algorithm.

2 Formulation of the Problem

We assume that the circuit under test has n inputs, and the test vectors to be applied are n-bit completely specified bit vectors. The problem of minimising the number of shifts required is then equivalent to the problem of finding the shortest bit string which includes every test vector as a substring. Such a string is called a superstring of the set of test vectors. This problem, called the shortest common superstring or SCS problem, can be proved to be NP-Hard in general [2]. Several approximation algorithms are known for this problem [3]. However, by exploiting the fact that all test vectors are of the same length, we can find a better algorithm for this problem.

This is obtained by reformulating the problem in terms of graphs.

Let \( S = \{v_1, v_2, ..., v_m \} \) be the set of test vectors. We construct a directed graph corresponding to this set, denoted by GS as follows. The vertices in GS are the bit strings of length \( n-1 \) which are substrings of some test vector. There is an arc in GS corresponding to each test vector \( v_i \), directed from vertex \( u \) to vertex \( v \), where \( u \) and \( v \) are the prefix and suffix of \( v_i \) of length \( n-1 \), respectively.

Let \( Gc \) be the complete graph with the same vertex set as GS. A superstring of \( S \) is then just a walk in \( Gc \) containing all the arcs in GS. If we replace multiple occurrences of arcs in the walk by multiple arcs in parallel, the graph formed by the walk in \( Gc \) has an eulerian path. We call a graph with an eulerian path eulerian.

We define the cost of an arc directed from a vertex \( u \) to a vertex \( v \) in \( Gc \) to be the length of the shortest string \( w \) such that \( v \) is a suffix of \( uw \). The cost of every arc in GS is 1. The cost of a graph is the sum of the costs of all arcs in the graph. The length of the shortest superstring of \( S \) is then just the cost of the least cost eulerian graph which contains all arcs in GS \( + n - 1 \), allowing multiple arcs from one vertex to another. The superstring itself is defined by the eulerian path in the graph.

ASSUMPTION : The underlying undirected graph of GS is connected.

In this case we give a simple linear time algorithm to compute the shortest superstring of \( S \).

Define the deficiency of a vertex \( u \), Def(\( u \)) in any graph to be the indegree of \( u \) - the outdegree of \( u \). The total deficiency of a graph is the sum of the deficiencies of all vertices which have positive deficiency. It follows that a
graph is eulerian if the underlying undirected graph is connected and its total deficiency is \( \leq 1 \). Therefore, by our assumption on connectivity, it is sufficient to add arcs to \( G^* \) so that the total deficiency is \( \leq 1 \).

3 The Algorithm.

**Algorithm Greedy**

\[ \text{G} := \text{GS}; \]

while Total Deficiency of \( G > 1 \) do

begin

Select an arc with minimum cost directed from a vertex \( u \) with positive deficiency to a vertex \( v \) with negative deficiency.

\[ \text{G} := \text{G union} \ (u, v); \]

(* If \( G \) already contains arcs directed from \( u \) to \( v \) add one more arc *)

end;

We claim that the graph \( G \) obtained by the greedy algorithm is the least cost eulerian graph containing \( G^* \) and the eulerian path in \( G \) defines the shortest super-string of \( S \). We first note that the cost function satisfies the triangle inequality, that is, \( \text{cost}(u, v) \leq \text{cost}(u, w) + \text{cost}(w, v) \).

**Lemma 3.1** Let \( G' \) be the least cost eulerian graph with the fewest number of arcs, containing \( G^* \) as a subgraph. Then every arc in \( G^* \) which is not in \( G \) is directed from a vertex with positive deficiency to a vertex with negative deficiency in \( G \).

**Proof:** Consider any arc \( (u, v) \) in \( G' \) which is not in \( G \). Suppose \( \text{Def}(v) > 0 \) in \( G \). Then there must be an arc \((v, w)\) in \( G' \) which is not in \( G \). Replacing arcs \((u, v)\) and \((v, w)\) by the arc \((u, w)\) gives a graph with the same cost and fewer arcs than \( G' \). The only other possibility is that \( \text{Def}(v) = 0 \) in \( G \) and \( +1 \) in \( G' \), in which case deleting the arc \((u, v)\) from \( G^* \) gives an eulerian graph with fewer arcs. Similarly, we can show that \( \text{Def}(u) > 0 \) in \( G \).

**Lemma 3.1** implies that we can construct an optimal eulerian graph containing \( G^* \) by adding arcs to \( G \) directed from vertices with positive deficiency to vertices with negative deficiency.

**Lemma 3.2** Let \( (u, v), (u, t), \) and \( (s, v) \) be arcs such that \( \text{cost}(u, v) \) is the minimum of the costs of the three arcs. Then \( \text{cost}(u, v) + \text{cost}(s, t) \leq \text{cost}(u, t) + \text{cost}(s, v) \).

**Proof:** Let \( \text{cost}(u, v) = c \), \( \text{cost}(u, t) = c_1 \) and \( \text{cost}(s, v) = c_2 \). Then there exists a string \( x \) with \( |x| = n-1-c \) such that \( x \) is a suffix of \( u \) and a prefix of \( v \). Since \( c_1 > c \), there is a string \( x_1 \) which is a suffix of \( x \) and a prefix of \( t \). Therefore \( x = x_2x_1 \) with \( |x_1| = n-1-c_1 \) and \( |x_2| = c_1-c > 0 \). There exists a string \( x_3 \) with \( |x_3| = n-1-c_2 \), which is a suffix of \( s \) and a prefix of \( v \).

Case (i) \( x_3 \) is a prefix of \( x_2 \), ie \( n-1-c_2 < c_1-c \) or \( (n-1)+c_2 \leq c_1 \) and therefore \( \text{cost}(u, v) + \text{cost}(s, t) \leq \text{cost}(u, t) + \text{cost}(s, v) \).

Case (ii) \( n-1-c_2 > c_1-c \). In this case there is a suffix \( x_4 \) of \( x_3 \), which is also a suffix of \( x_1 \) with \( |x_4| = n-1-c_2-c_1+c \). Therefore \( x_4 \) is a suffix of \( s \) and a prefix of \( t \), hence \( \text{cost}(s, t) < c_2+c_1-c \) or \( \text{cost}(u, v) + \text{cost}(s, t) < \text{cost}(u, t) + \text{cost}(s, v) \).

This proves the lemma.

**Lemma 3.3** Let \( (u, v) \) be an arc with minimum cost such that \( \text{Def}(u) > 0 \) and \( \text{Def}(v) < 0 \) in \( G \). Then there is a minimum cost eulerian graph with \( G^* \) as a subgraph which includes the arc \((u, v)\), assuming that \( G \) itself is not eulerian.

**Proof:** Let \( G' \) be a least cost eulerian graph with the fewest number of arcs which contains \( G^* \) as a subgraph, and suppose arc \((u, v)\) does not belong to \( G' \).

Case (i) There are arcs \((u, t)\) and \((s, v)\) in \( G' \) which are not in \( G \). By \( \text{Lemma 3.1, Def}(t) < 0 \) and \( \text{Def}(s) > 0 \) in \( G \) which implies that \( \text{cost}(u, v) < \text{cost}(u, t) + \text{cost}(s, v) \). By \( \text{Lemma 3.2} \) cost \((u, v) + \text{cost}(s, t) < \text{cost}(u, t) + \text{cost}(s, v) \). Therefore by replacing the arcs \((u, t)\) and \((s, v)\) in \( G' \) by the arcs \((u, v)\) and \((s, t)\) we get the required graph.

Case (ii) There is an arc \((u, t)\) but no arc \((s, v)\) in \( G' \) which is not in \( G \). Then \( \text{Def}(v) = -1 \) in both \( GS \) and \( G' \), and therefore \( \text{Def}(t) = 0 \) in \( G' \), but \( \text{Def}(s) < 0 \) in \( G \).

Replacing the arc \((u, t)\) by the arc \((u, v)\) in \( G' \) gives the required graph. The case when there is no arc \((u, t)\) but there is an arc \((s, v)\) in \( G' \) which is not in \( G \) is similar.

Case (iii) There is no arc \((u, t)\) or \((s, v)\) in \( G' \) which is not in \( G \). Then \( \text{Def}(u) = +1 \) and \( \text{Def}(v) = -1 \) in both \( GS \) and \( G' \). Since \( GS \) is itself not eulerian, there is at least one arc in \( G' \) which is not in \( GS \). Deleting that arc and adding the arc \((u, v)\) gives the required graph.

**Theorem 3.1** Algorithm Greedy produces an optimal solution.

**Proof:** The proof follows from \( \text{Lemma 3.3} \), since an optimal solution for \( GS \) which includes the arc \((u, v)\) is an optimal solution for the graph \( GS \) union \((u, v)\) and vice versa.

4 Computational Results.

In this section we present some computational results to show the effectiveness of the algorithm. We note first that the algorithm greedy is still be used even if the Assumption mentioned in section 2 is not satisfied. In this case however, there is no guarantee that the solution obtained is optimal. A modification in the algorithm is required in order to ensure that the graph \( G \) obtained is connected. At every stage, if there is a choice amongst arcs which can be selected, an arc which connects two different components should be selected. Finally, if the graph is still disconnected, more arcs have to be added in order to connect the graph and obtain an eulerian path.

We first consider the example circuit given in [1]. The The minimal set of test vectors required for testing this circuit are, \( v_1 = 01100110, v_2 = 01111111, v_3 = 01101101, v_4 = 01110111, v_5 = 01101111, v_6 = 155 \).
01100011, v7 = 00111000, v8 = 01011000 and v9 = 11111000. If each test vector is shifted in independently in the shift register, 72 shift operations would be required. However by shifting in the vectors in the order v1, v3, v5, v2, v9, v8, v6, v7, v4 only 41 shift operations are required. The superstring of these test vectors can also be used for reducing the storage space required for storing the test patterns.

We next show the results obtained using random sets of test vectors. The number of test patterns used is twice the number of inputs. This is a small number compared to the total number of patterns. Even in this case, a reduction of about 40% is obtained in the number of shifts required, as shown in Table 1. When the number of test patterns is large, the improvement is even more significant.

Table 1.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>No. of shifts</th>
<th>n*m</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>13</td>
<td>32</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>42</td>
<td>72</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>67</td>
<td>128</td>
<td>48</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>115</td>
<td>200</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>167</td>
<td>288</td>
<td>42</td>
</tr>
<tr>
<td>14</td>
<td>28</td>
<td>255</td>
<td>392</td>
<td>35</td>
</tr>
</tbody>
</table>

References

