A Modified Approach to Test Plan Generation for Combinational Logic Blocks

ANUPAM BASU, DILIP K. BANERJI*, AMIT BASU, T. C. WILSON and JAY C. MAJITHIA

VLSI-CAD Group, Department of Computing and Information Science, University of Guelph, Guelph, Ontario, CANADA N1G 2W1

Generation of test plans is a crucial step for testing VLSI circuits. This paper presents a modified approach to test plan generation for the BILBO test methodology. A few limitations of the existing approaches have been identified and methods to address these have been suggested. The proposed approach has been implemented for the general case of n-port combinational logic blocks (CLBs). However, due to limitations of space and for clarity, only 2-port CLBs are considered in this paper. For this case, the problem is modelled as a Step Scheduling Matrix and an algorithm is presented for the solution. The algorithm has been tested on a number of benchmark circuits and the results are compared with those obtained through existing methods. The effectiveness of the proposed approach is clear from the results, as it contributes to the reduction in total testing time as well as generates a larger number of test plans.

Keywords: VLSI, test, combinational logic, test plan

1. INTRODUCTION

With the increasing complexity of VLSI circuits, the tasks of gate-level fault modeling and test generation have become excessively time-consuming and expensive. To alleviate this problem, testable design methodologies (TDMs) are being increasingly adopted. One such TDM is the Built-In-Logic-Block-Observer (BILBO) technique, proposed by Koenemann et al [1]. The BILBO approach views a VLSI circuit as consisting of combinational logic blocks (CLBs), interspersed with interconnections and transporting elements like registers, buses, and multiplexers. Although, all of these elements need to be tested, this paper concentrates on testing the CLBs only.

In order to test a CLB, test patterns are applied to the input ports of the CLB and its output responses are collected and compacted. A set of registers serves as *Pseudo Random Pattern Generators* (PRPGs) and another set of registers serves as *Multiple Input Signature Registers* (MISRs). The test patterns propagate from PRPGs to the input ports of a CLB, and the output responses propagate from the output port(s) of the CLB to the MISRs. Such propagation takes place through paths consisting of registers, multiplexers, and buses. These paths merely transfer data and do not modify them in any way. Such paths are, therefore, called identity transfer paths or I-paths [2].

After identifying the I-paths for a CLB, a *test plan*

*Corresponding author. Tel. (519) 824-4120 Ext.3005. Fax (519) 837-0323. E-mail:dbanerji@snowwhite.cis.uoguelph.ca

is generated for transferring test data to the input ports of a CLB and the CLB output responses to MISRs. The generation of a test plan consists of the following steps: i) Combination of the input I-paths, ii) Combination of the output I-paths, and iii) Concatenation of the input and output I-paths.

An I-path consists of a number of steps, where a *step* designates one clock cycle in which the test data propagates from one storage element to another. Combination of I-paths for the different ports of a CLB essentially implies combining the steps of more than one path such that the combined steps can be executed *concurrently*. This is possible only if there is no resource conflict within the combined steps. A resource conflict occurs when two steps require the same circuit element simultaneously. If such conflicts exist between the steps to be combined, then these steps are skewed by inserting No-op steps in one of the I-paths in order to avoid the conflicts. A No-op step in an I-path holds the test pattern in a register, used in the previous step, for one clock cycle. For the input I-paths, it is required that the test vectors arrive simultaneously at all the input ports of a CLB. To achieve this, the task of I-path combination must not only generate a conflict-free combination of steps, but also equalize the lengths of the component I-paths.

After the input and output I-paths have been combined individually, they are concatenated to generate the test plan. Although there are no intra-step conflicts remaining at this stage, the different steps of the test plan may still use common resources. Normally, this factor should not cause any problems since different steps are executed in different clock cycles. However, in order to reduce the total test time, the test patterns are *pipelined* through a test plan. Due to this pipelining, two distinct steps sharing a common resource may overlap in the same clock cycle and thus result in a resource conflict. To alleviate this problem, the initiation of the consecutive iterations of test vectors through the pipeline is delayed by a suitable amount of time. This inter-iteration delay is called the initiation delay (D).

For a test requiring T iterations through a test plan of S steps, the total test time is given by $t = S + (T$

$- 1)D$ [2]. Since T is usually large compared to S , the total test time is much more sensitive to D than to S . Hence, for the reduction of total test time, it is very important to minimize D .

Abadir [2] has proposed an algorithm to combine n I-paths for a multiport CLB. In this algorithm, the number of No-ops inserted at every step has been minimized. Abadir and Breuer [3] have also considered the problem of pipelining the test vectors through a test plan, and have suggested a lower bound on D . An algorithm for minimizing D by inserting No-ops in the combined test plan has also been presented by them. Bhawmik [4] has formulated these tasks as a consistent labeling problem [5], and has solved it using a backtracking algorithm. However, in these approaches, the problems of I-path combination and D -minimization have been treated separately.

In this paper, we first show that the process of combination of I-paths can affect the lower bound on D . Accordingly, we propose an approach to I-path combination which refrains from increasing the lower bound on D , as far as possible. The earlier algorithms [2,4] carry out the task of I-path combination in the direction of test pattern propagation. However, as shown in the next section, this approach may sometimes inhibit obtaining a feasible combination even if it exists. In order to resolve this problem, our approach starts the I-path combination process from the port side of a CLB and proceeds *outwards* towards a PRPG or MISR. This approach always finds a feasible combination if it exists. To deal with some typical conflict situations, it may, however, be necessary to resort to backtracking to arrive at a feasible solution.

The following section presents the basis of our approach. It is followed by an algorithm for combining the I-paths for CLBs with *two* input/output ports. This special case has been modeled as a *Step Scheduling Matrix* and the heuristics employed in the two-port algorithm have been explained with respect to this model.

The I-paths for n -port CLBs, where $n \geq 2$, may be combined by repeated application of the two-port algorithm. However, this process of pairwise combination can render the value of D sensitive to the order

of selecting the pairs to be combined. Hence, to ensure a minimum D , all possible pairwise combinations may need to be tried. Since the value of n does not exceed 4 for most practical cases, this approach will not be too expensive from a practical viewpoint. However, we do have a general n -port algorithm of which the 2-port algorithm is a specific case. The n -port algorithm is also based on heuristics that are similar to the 2-port case. However, due to space limitations, we are unable to present the details of the n -port algorithm.

2. BASIS OF THE PROPOSED APPROACH

Before presenting the basis of the proposed approach, we introduce certain notations which have been used in describing the 2-port algorithm.

2.1 Notations

I_k ($1 \leq k \leq n$): An Input I-path for port k of a CLB, where n = number of input ports.

O_k ($1 \leq k \leq n$): An Output I-path for port k of a CLB, where n = number of output ports.

P_k ($1 \leq k \leq n$): An I-path (may be input or output) for port k of a CLB.

s_k : number of steps in the I-path P_k .

I_{k_i} ($1 \leq k \leq s_k$): i th step of the input I-path I_k .

O_{k_i} ($1 \leq k \leq s_k$): i th step of the output I-path O_k .

P_{k_i} : i th step of the I-path P_k .

$I_{k_i'}$: A No-Op step inserted in the input I-path I_k , after step I_{k_i} .

$O_{k_i'}$: A No-Op step inserted in the output I-path O_k , after step O_{k_i} .

$P_{k_i'}$: A No-Op step inserted in the I-path P_k , after step P_{k_i} .

A combination of input I-paths is designated as C^*I , where C^*I is an ordered set of combined steps CI_i . Thus, $C^*I = \{CI_1, CI_2, \dots, CI_j, \dots, CI_s\}$. A combined step CI_i is a set of steps $\{I_{j_i}\}$ (of the different input I-paths), combined together, where $i \in \{1, \dots, s_j\} \cup \{1', \dots, s_j'\}$, and $1 \leq j \leq n$. For example, consider the

combination of the three input I-paths shown in Fig. 1. The combined input I-path can be represented as:

$$C^*I = \{CI_1, CI_2, CI_3\}, \text{ where}$$

$$CI_1 = \{I_{1_1}, I_{2_1}, I_{3_1}\}$$

$$CI_2 = \{I_{1_2}, I_{2_1'}, I_{3_2}\}$$

$$CI_3 = \{I_{1_3}, I_{2_2}, I_{3_3}\}$$

Similarly, a combination of output I-paths is designated as C^*O where: $C^*O = \{CO_1, CO_2, \dots, CO_j, \dots, CO_p\}$.

A test plan is generated by concatenating the combined input I-path and the combined output I-path.

Apart from the notation presented here, the I-paths are often represented graphically. In such representations, the individual steps are represented as nodes, and the conflicts by dotted edges. In the following paragraphs, we explain the basic strategies adopted in the proposed approach to test plan generation and their underlying rationale.

2.2 Approaching from the Port Side

The existing algorithms for test plan generation [2,4] attempt to combine the I-paths, starting from the test pattern generator side (for the input I-paths) and from the port side (for the output I-paths). But this approach may lead to the serious limitation of precluding feasible input I-path combinations even when they exist. Moreover, it is also prone to yield a higher value of D . These aspects are illustrated through examples.

Example 1: Consider the two input I-paths to two ports of a CLB Fig. 2(a)). The nodes denote the steps

FIGURE 1 Three Input I-Paths

FIGURE 2 (a) I-Paths for Example 1. (b) Resulting Schedule

and the dotted edge denotes that there is a common register R shared in the two steps, $I1_3$ and $I2_2$.

Since the existing algorithms start the combination process from the input (pattern generator) side, $I1_1$ will be scheduled with $I2_1$, and $I1_2$ will be scheduled with $I2_2$. Since the test patterns must arrive at the CLB input ports simultaneously, the two input I-paths must be synchronized in time (clock cycles). This is achieved by introducing two No-Op steps after step $I2_2$. The resulting schedule is shown in Fig. 2(b).

The No-Op step $I2_2'$, implies holding the contents of register R , resulting in a conflict between $I1_3$ and $I2_2'$. Since $I2_2$ is the last step in $I2$, this conflict cannot be resolved if the combination process starts from the pattern generator side and, hence, a feasible combination will not be attained.

Example 2: Consider two input I-paths as shown in Fig. 3(a). Note that these I-paths have no resource conflicts between them. Hence, when they are combined from the pattern generator side, we obtain the schedule shown in Fig. 3(b). In the combined schedule, Step 2 of the second I-path ($I2_2$) has to be followed by a number of No-Op steps, in order to equalize the two input I-paths. The $I2_2'$ steps will hold up the resources used in the step $I2_2$ for three more clock cycles. This results in *inter-step conflicts* in the combined schedule. Specifically, the combined steps CI_2 through CI_5 happen to share resources common to $I2_2$ (Fig. 3(b)). If the test patterns are pipelined through this combined schedule, then the inter-iteration delay

FIGURE 3 (a) I-Paths for Example 2. (b) Combined I-Path.

(D) has to be *at least* 4 in order to avoid the inter-step conflicts. However, as will be shown shortly, the value of D could have been 1 in this case.

In order to address the problems highlighted by the preceding two examples, we start the I-path combination process from the CLB port side. Thus, the last steps of the I-paths are combined first—the preceding steps are combined subsequently. The advantage of this approach is illustrated through the same examples as shown in Fig. 2(a) and Fig. 2(b). For the two I-paths shown in Fig. 2(a), the combined schedule resulting from I-path combination, starting from the port side, is shown in Fig. 4(a). Note that a feasible schedule has been obtained in this case, and that the No-Op steps required for synchronizing the paths have been pushed to the pattern generator side. These

FIGURE 4 (a) Port Side Combination for I-Paths of Example 1. (b) Port Side Combination for I-Paths of Example 2.

No-Ops imply initial delay in pattern generation and do not involve holding of any resources, unlike the No-Ops inserted *within* a path.

For the I-paths shown in Fig. 3(a), combination from the CLB port side results in the schedule shown in Fig. 4(b). Note that there is no interstep conflict in this combined schedule. Hence, the minimum value of D is 1 in this case, as contrasted to the value 4 obtained by the existing methods [2,4]. The reason for starting the combination process from the CLB port side is based on the fact that the No-Op steps required for equalizing the paths are pushed towards the pattern generator side, instead of the port side. Since there are no *real* steps preceding these No-Op steps, no additional resource conflicts can arise.

2.3. Minimizing Contiguous No-Ops

Abadir [2] attempted to resolve the resource conflicts (to obtain a feasible schedule) by minimizing the total number of No-Ops inserted at a particular *frontier* step. However, his approach does not exclude the possibility of two contiguous No-Ops being inserted in the same I-path. We observe that the number of contiguous No-Ops inserted in a particular I-path is a very important factor that affects the lower bound on D .

THEOREM 1: *If n contiguous No-Op steps (involving register holds) are inserted in any particular I-path being combined, then the inter-iteration delay D of the combined test plan will be at least $(n + 1)$ clock cycles.*

Proof: Let s_i be a step of the *combined I-path* such that the steps $s_{i+1}, s_{i+2}, \dots, s_{i+n}$, all share a common resource (register). Now, let us assume that the minimum feasible D is $D_{min} < n + 1$. Let $D_{min} = p$ ($1 \leq p \leq n$). Therefore, during test pipelining, Step s_i of one iteration will be in the same pipeline phase as Step s_{i+p} of the previous iteration. That is, Step s_i of the current iteration and Step s_{i+p} of the previous iteration will be executed in the same clock cycle. But, this cannot be allowed since s_i (current) and s_{i+p}

(previous) use the same register. Hence $D_{min} \geq n + 1$.

In the light of Theorem 1, our approach attempts to distribute the No-Op steps in the different I-paths and, thereby, minimizes the number of contiguous No-Ops in any I-path as far as possible. Hence, the salient strategies adopted in our approach are:

- (i) Initiating the I-path combination process from the CLB port side.
- (ii) Attempting to minimize the number of contiguous No-Ops inserted in a particular I-path.

The following section explains how these strategies have been incorporated in our approach.

3. TWO-PORT ALGORITHM

For CLBs with two input/output ports, the problem of I-path combination can be conveniently depicted as a two-dimensional *Step Scheduling Matrix*. Since two-port CLBs are very common in digital circuits, this case is presented to highlight the main features of our approach.

3.1. Step Scheduling Matrix (SSM)

Let $P1$ and $P2$ denote I-paths for two ports, and let m and n be the total number of steps in $P1$ and $P2$, respectively. Also, let $n \geq m$, without any loss of generality.

The Step Scheduling Matrix is a two-dimensional matrix of m rows and n columns. Each element in the SSM is denoted by (i,j) , $1 \leq i \leq m$, and $1 \leq j \leq n$, where i and j denote distinct steps of $P1$ and $P2$, respectively. The rows and columns are numbered in such a manner that the lower rightmost corner of the SSM represents the steps that are nearest to the ports. Accordingly, for *input* I-path combination, the lower rightmost element is labeled (m,n) , whereas for *output* I-path combination, it is labeled $(1,1)$.

If $P1_i$ and $P2_j$ share a common resource, then there exists a *potential* resource conflict between these steps. Such a conflict is shown as a *cross* (X) in position (i,j) of the SSM. As an example, Fig. 5(a) illustrates two input I-paths of a CLB, with the corresponding SSM shown in Fig. 5(b). Fig. 5(c) illustrates two output I-paths of a CLB, and the corresponding SSM is shown in Fig. 5(d).

It may be noted that the placement of a cross (X) at any position (i,j) of the SSM implies that step i of $P1(P1_i)$ cannot be scheduled with step j of $P2(P2_j)$, since they share a common resource.

3.2. I-path Combination

In terms of the SSM, the task of combining the different steps of I-paths is depicted by placing markers,

FIGURE 5 (a) Two Input I-Paths. (b) The Corresponding SSM. (c) Two Output I-Paths. (d) The Corresponding SSM.

denoted by *circles* (0) in the different positions of the SSM, subject to the following rules:

- (1) A marker cannot be placed in a position (i,j) if (i,j) contains a cross.
- (2) A marker can be placed in a position $(i - 1, j - 1)$ if:
 - a) the position (i,j) already contains a marker, or
 - b) the position $(i,j - 1)$ already contains a marker, or
 - c) the position $(i - 1, j)$ already contains a marker.

The placement of a marker at position (i,j) implies that $P1_i$ is scheduled simultaneously with $P2_j$.

A few notations used in the following discussion are now presented.

$[(i,j),C]$: the position (i,j) in the SSM contains a cross

$[(i,j),\sim C]$: the position (i,j) in the SSM does not contain a cross

$[(i,j),M]$: the position (i,j) in the SSM contains a marker

$[(i,j),\sim M]$: the position (i,j) in the SSM does not contain a marker

Corresponding to the aforesaid rules, three possible *moves* in the SSM may now be defined.

Diagonal move (D-move): If $[(i,j),M]$ and if $[(i - 1, j - 1), \sim C]$, then a marker may be placed in position $(i - 1, j - 1)$. Such a placement constitutes a D-move in the SSM.

Horizontal move (H-move): If $[(i,j),M]$ and if $[(i, j - 1), \sim C]$, then a marker may be placed in position $(i, j - 1)$. Such a placement constitutes an H-move in the SSM.

Vertical move (V-move): If $[(i,j),M]$ and if $[(i - 1, j), \sim C]$, then a marker may be placed in position $(i - 1, j)$. Such a placement constitutes a V-move in the SSM.

Application of these moves is now illustrated with an example. Fig. 6(a) shows two input I-paths of a CLB. As before, the potential conflicts between the I-paths are shown by dotted edges. Figure 6(b) is the

FIGURE 6 (a) Two Input I-Paths with Conflicts. (b) SSM for Fig.6(a). (c) Combined I-Path.

corresponding SSM. Note that crosses have been placed at the positions corresponding to the conflicting steps. As mentioned earlier, the process of I-path combination starts from the port side of a CLB. Accordingly, the task of placing markers in the SSM starts from the lower righthand corner.

First, a marker is placed in position (5,6). Next, by a D-move, another marker is placed in position (4,5). Since both the D-move and H-move from (4,5) are inhibited by crosses, a V-move is made to place a marker in position (3,5). The next marker is placed at (2,4). Since both the D-move and V-move are inhibited at (2,4), an H-move is made to place the next marker at (2,3). The last marker is placed at (1,2) by a diagonal move. The resulting schedule is shown in Fig. 6(c).

It may be noted that the V-move applied at position (4,5) results in placing two markers in column 5. This implies that both the Steps $I1_3$ and $I1_4$ are scheduled

with Step $I2_5$. This, in turn, implies that a No-Op has to be inserted after $I2_5$, so that $I1_3$ is scheduled with $I2_5$ and $I1_4$ is scheduled with $I2_5$.

Similarly, the H-move at position (2,4) places two markers in the second row of the SSM. This implies that a No-Op has to be inserted after $I1_2$ so that $I2_3$ is scheduled with $I1_2$, and $I2_4$ is scheduled with $I2_2$. These observations can be generalized as follows.

- (1) Each vertical move in the SSM introduces a No-Op in the current step of the second I-path.
- (2) Each horizontal move in the SSM introduces a No-Op in the current step of the first I-path.

As shown in Fig. 6(c), the combined schedule of the I-paths can be obtained by aligning the row and column indices of the marked positions in the SSM. It may be further noted that $I2_2$ has been scheduled with $I1_1$, and, consequently, $I2_1$ is scheduled alone. This may be viewed as delaying the first step of $I1$ by one clock cycle. However, since such delays precede the *first* step, it does not result in holding up any resources.

In the context of the preceding discussion, the process of I-path combination can be viewed as obtaining a continuous sequence of marked positions in the SSM. Such a sequence may be obtained using the following steps.

1. Form the SSM and place crosses at the conflict positions.
2. Mark the lower rightmost corner (LRC) of the SSM. Current position $\leftarrow$ LRC
3. **while** (No position in the first row is marked **or** no position in the first column is marked) **do**
begin
 - 3a. Select a *proper* move with respect to the current position.
 - 3b. Place a marker at the position arrived at by the move.
 - 3c. Current position $\leftarrow$ Newly marked position**end {while}**

The key step in this algorithm is the selection of the *proper* move. It is explained in the next subsection.

3.3. Selecting a Move

As shown in Theorem 1, placement of contiguous No-Ops in a path adversely affects the lower bound on the inter-iteration delay D . Hence, the heuristics adopted for deciding on the moves in the SSM aim to avoid the placement of contiguous No-Ops in the same path, as far as possible. The heuristics are explained in the following paragraphs.

Lookahead Conflict (LC): With respect to a marker at position (i, j) in the SSM, if there is a cross (X) either in the position $(i - 2, j - 1)$ or $(i - 1, j - 2)$, then these are called *Lookahead Conflicts* with respect to the position (i, j) . The conflict in position $(i - 1, j - 2)$ is termed as a horizontal lookahead conflict (HLC) and that in the position $(i - 2, j - 1)$ is termed as a vertical lookahead conflict (VLC).

As an example, consider the SSM shown in Fig. 7. Let the currently marked position be $(5, 6)$. No D-move is possible from this position. As shown earlier, a V-move or an H-move implies the insertion of a No-Op in either path. Moreover, with respect to position $(5, 6)$, the cross at position $(3, 5)$ is a VLC. Hence, even if a V-move is applied at position $(5, 6)$ marking the position $(4, 6)$, no D-move can still be applied from $(4, 6)$. This, in turn, implies that *at least* two contiguous No-Ops have to be placed in I_2 if a

FIGURE 7 SSM Showing Lookahead Conflict

V-move is selected. Similarly, the cross at position $(4, 4)$ is an HLC with respect to position $(5, 6)$, implying that at least two No-Ops have to be placed in I_1 if an H-move is selected.

It may be further noted from Fig. 7 that, for V-moves, the contiguous crosses at positions $(4, 5)$, $(3, 5)$ and $(2, 5)$ necessitate the insertion of three contiguous No-Ops in I_2 for making a D-move possible. On the other hand, for H-moves, the contiguous crosses at positions $(4, 5)$ and $(4, 4)$ require insertion of two contiguous No-Ops in I_1 before a D-move can be applied. Accordingly, the costs of an H-move or a V-move are defined as follows.

H-cost: If the presently marked position is (i, j) and if there are k contiguous crosses at positions $(i - 1, j - p)$, $p = 1, 2, \dots, k$, then the H-cost (cost of a horizontal move from (i, j)) is k .

V-cost: If the presently marked position is (i, j) and if there are k contiguous crosses at positions $(i - p, j - 1)$, $p = 1, 2, \dots, k$, then the V-cost (cost of a vertical move from (i, j)) is k .

It is obvious that H-costs and V-costs signify the number of *contiguous* No-Ops placed in the first and second path, respectively. In this context, the rules for selecting the moves are outlined as follows.

Let the current position be (i, j) .

Let $(HC)_{ij}$ and $(VC)_{ij}$ denote, respectively, the H-cost and V-cost with respect to position (i, j) .

Rule 1: if $[(i - 1, j - 1), \sim C]$ then select D-move

Rule 2: if $[(i - 1, j - 1), C]$ and $[(i, j - 1), C]$ and $[(i - 1, j), \sim C]$ then select V-move

Rule 3: if $[(i - 1, j - 1), C]$ and $[(i - 1, j), C]$ and $[(i, j - 1), \sim C]$ then select H-move

Rule 4: if $[(i - 1, j - 1), C]$ and $[(i - 1, j), \sim C]$ and $[(i, j - 1), \sim C]$ then

begin

Compute $(HC)_{ij}$ and $(VC)_{ij}$.

if $(HC)_{ij} > (VC)_{ij}$

then

Select V-move

else

Select H-move

end

Rule 5: if $[(i - 1, j - 1), C]$ and $[(i, j - 1), C]$ and $[(i - 1, j), C]$ then Backtrack

It may be noted that while computing the costs $(HC)_{ij}$ and $(VC)_{ij}$, more than one level of lookahead for conflict is performed, and also that such computations are required only when both the H-move and V-move are the only possible candidates.

3.4. An Extension

The I-path combination algorithm, as discussed in the previous section, fails to generate a solution if a bus or multiplexer is shared between the *nearest-to-port* steps of the two I-paths. Figure 8(a) shows part of a VLSI circuit, where there is a bus conflict between the last steps of input I-paths $I1$ and $I2$. Fig. 8(b) specifies the steps in each I-path, and the corresponding SSM is shown in Fig. 8(c). Since there is a conflict (X) in the lower rightmost position of the SSM,

FIGURE 8 (a) Part of a VLSI Circuit. (b) I-Paths $I1$ and $I2$. (c) The SSM. (d) Modified SSM. (e) Combined I-Path.

the algorithm, as presented, will not be able to initiate the combination process. In order to resolve this problem, the SSM is extended by an additional row, if there exists a conflict in the last step. This implies the addition of a new step $I1_2'$, after Step 2 of I-path $I1$, as shown in Fig. 8(d). The two-port algorithm is then applied on this modified SSM. If position (2,1) in the SSM had a conflict, then a No-Op step $I2_2'$ is added after Step 2 of I-path $I2$. This implies the addition of a new column to the SSM, instead of a row. The function of such No-Op steps is to hold the contents of the *nearest-to-port* register of the concerned I-path for one clock cycle, so that the conflicting steps are scheduled at different times. The combined input I-path, denoted CI , is shown in Fig. 8(e).

Note that if the *nearest-to-port* steps had a *register* conflict, then the I-paths could not be combined at all since the same register has to latch or hold the data for two different ports at the same control step, which is impossible.

3.5. Backtracking

If, from a position (i, j) in the SSM, none of the moves is applicable, then the algorithm resorts to backtracking. This is accomplished through the following steps.

1. **If** (Current position is LRC)
then
No feasible combination; Exit.
else
Current position $\leftarrow$ Previous position
2. Place a cross at (i, j) .
3. Select the proper move with respect to the current position.

To illustrate the process of backtracking, consider the potential conflicts shown in Fig. 9(a). The corresponding SSM is shown in Fig. 9(b). After placing a marker at position (4,4) through a D-move, no further moves can be applied. Hence, the algorithm backtracks to position (5,5) and assumes a cross to be placed at position (4,4). Next, using lookahead heu-

- $CI_1 : \{ I2_1 \}$
 $CI_2 : \{ I2_2 \}$
 $CI_3 : \{ I1_1, I2_3 \}$
 $CI_4 : \{ I1_2, I2_4 \}$
 $CI_5 : \{ I1_3, I2_5 \}$
 $CI_6 : \{ I1_4, I2_5' \}$
 $CI_7 : \{ I1_5, I2_5' \}$
- (c)

FIGURE 9 (a) A Conflict Situation. (b) SSM Illustrating Back-track. (c) The Combined I-Path.

ristics, two vertical moves are selected to generate the schedule shown in Fig. 9(c).

4. DISCUSSION OF RESULTS

The two-port algorithm presented in the previous section has been applied to generate test plans for a number of VLSI circuits. A few CLBs from such circuits are selected here to illustrate the effectiveness of the algorithm.

Three different circuits are shown in Figs. 10(a), (b), and (c). The input and output I-paths of these circuits are shown in Tables I(a), I(b) and I(c), respectively. Table II presents the combined test plans generated by the proposed approach (B) and the existing approach (A) [2]. The pipelined schedules and the

FIGURE 10 The Different Circuits

values of D obtained are also presented for comparison. It is clear from Table II, that the values of D obtained by the proposed approach are less than those obtained by method A. Moreover, in some cases (Example 1), method A fails to arrive at a feasible combination, while method B succeeds in achieving a feasible schedule. In Example 3 of Table II, the proposed approach succeeds because of the facility of backtracking, whereas the existing approach fails to achieve a feasible combination.

The 2-port algorithm has been generalized to the case of n -ports. However, due to space limitations, the extension is not presented here.

4.1. Further Results and Discussion

The proposed test plan generation scheme has been applied to a number of well-known synthesized data-

TABLE I I-paths for the circuits of Fig.10

(a)			
I1:	I1 ₁ :R1 (PRPG) I1 ₂ :R2 (LATCH) I1 ₃ :MUX (R2), R4 (LATCH) I1 ₄ :R5 (LATCH)	I2:	I2 ₁ :R3 (PRPG) I2 ₂ :MUX (R3), R4 (LATCH)
O1:	O1 ₁ :R6 (MISR)		
(b)			
I1:	I1 ₁ :R1 (PRPG) I1 ₂ :R2 (LATCH)	I2:	I2 ₁ :R3 (PRPG)
O1:	O1 ₁ :R4 (MISR)		
(c)			
I1:	I1 ₁ :R1 (PRPG)	I2:	I2 ₁ :R7 (PRPG)
O1:	O1 ₁ :R2 (LATCH) O1 ₂ :R4 (LATCH) O1 ₃ :BUS (R4),R5 (LATCH) O1 ₄ :R8 (MISR)	O2:	O2 ₁ :R3 (LATCH) O2 ₂ :R5 (LATCH) O2 ₃ :BUS (R5),R4 (LATCH) O2 ₄ :R6 (MISR)

paths taken from the literature [6–9]. Table III summarizes the performance of the proposed algorithm (Method B) compared to the algorithm proposed by Abadir (Method A). The table presents the frequency distribution of the generated test plans over the different values of D . The number of test plans generated for the different CLBs has been shown sepa-

ately. In the table, f_i denotes the number of test plans generated with the value of $D = i$. $\sum f_i$ denotes the total number of test plans generated for a particular CLB. The average value of D is designated by D_{mean} .

It can be observed from the table that in certain cases Method B can generate a larger number of test plans than Method A (for example, in elliptical wave

TABLE II A comparative study of results obtained for circuits of Fig. 10

Example number	Method* used	Execution schedule			Value of D obtained
		After combining the port I-paths	After pipelining		
			Phase #	Combined step	
1	A	fails		—	—
		CI ₁ :I1 ₁	1:	CI ₁ CI ₃ CO ₁	
		CI ₂ :I1 ₂	2:	CI ₂ CI ₄	
	B	CI ₃ :I1 ₃ ,I2 ₁ CI ₄ :I1 ₄ ,I2 ₂ CO ₁ :O1 ₁			2
2	A	CI ₁ :I1 ₁ ,I2 ₁ CI ₂ :I1 ₂ ,I2 ₁ CO ₁ :O1 ₁	1: 2:	CI ₁ CO ₁ CI ₂	2
	B	CI ₁ :I1 ₁ CI ₂ :I1 ₂ ,I2 ₁ CO ₁ :O1 ₁	1:	CI ₁ CI ₂ CO ₁	1
	A	fails		—	—
		CI ₁ :I1 ₁ ,I2 ₁ CO ₁ :O1 ₁ ,O2 ₁ CO ₂ :O1 ₁ ,O2 ₂ CO ₃ :O1 ₁ ,O2 ₃ CO ₄ :O1 ₂ ,O2 ₄ CO ₅ :O1 ₃ CO ₆ :O1 ₄	1: 2: 3: 4:	CI ₁ CO ₄ CO ₁ CO ₅ CO ₂ CO ₆ CO ₃	4

*Method A: Abadir [2]; Method B: Proposed algorithm

TABLE III A comparative study of D obtained from generated test plans

Datapath	CLB #	Method	f_1	f_2	f_3	f_4	f_5	f_6	Σf_i	D_{mean}
elliptical wave filter [Ref. [6], pp. 227]	0	A*	12	9					21	1.43
		B	21						21	1.00
	1	A	242	588	717	315	9		1871	2.61
		B	399	1097	310	110			1916	2.07
	2	A	62	270	276	157	25	1	791	2.77
		B	154	476	132	34			796	2.06
greatest common divisor [Ref. [7], pp. 101]	0	A		4	1	1			6	2.50
		B		6					6	2.00
	1	A		5	6	3			14	2.86
		B		14					14	2.00
	2	A	2	4	3	1			10	2.30
		B	8	2					10	1.20
	3	A	2	4	3	1			10	2.30
		B	8	2					10	1.20
	4	A	4						4	1.00
		B	4						4	1.00
	5	A	1						1	1.00
		B	1						1	1.00
diffeqn [Ref. [8], pp. 165]	0	A	4	4					8	1.50
		B	8						8	1.00
	1	A	8		8				16	2.00
		B	8	4	4				16	1.75
	2	A	1						1	1.00
		B	1						1	1.00
	3	A	1						1	1.00
		B	1						1	1.00
	4	A	1						1	1.00
		B	1						1	1.00
	0	A	8	121	526	452	29		1136	3.33
		B	15	200	480	474	19		1188	3.24
mcs6502 [Ref. [9], pp. 137]	1	A	8	121	526	452	29		1136	3.33
		B	15	200	480	474	19		1188	3.24
	2	A	8	121	526	452	29		1136	3.33
		B	15	200	480	474	19		1188	3.24
	3	A	11	73	93	35	3		215	2.75
		B	19	167	29				215	2.05
pdp8 micro processor [Ref. [7], pp. 131]	0	A	5						5	1.00
		B	5						5	1.00
	1	A			9	8			17	3.47
		B			17				17	3.00
	2	A		2					2	2.00
		B		2					2	2.00
	3	A			9	8			17	3.47
		B			17				17	3.00
	4	A		2					2	2.00
		B		2					2	2.00
	5	A	1	5	4				10	2.30
		B	10						10	1.00

*Method A is the algorithm used in [2,3], while Method B is the algorithm proposed in this paper.

filter CLB #1 and in mcs6502 datapath). This is because of the fact that the proposed algorithm succeeds in certain cases where Method A fails. Even in the cases where the same number of test plans has been generated, the frequency distribution shows that the test plans generated by the proposed approach are clustered more in the lower values of D . This fact is evident from Fig. 11 which shows the distribution of the test plans in the case of the elliptical wave filter. For CLB #0, it can be observed that all the 21 test plans generated by Method B have $D = 1$, whereas the test plans generated by Method A contain 12 test plans with $D = 1$, and the remaining with $D = 2$. This phenomenon is demonstrated in almost all the test cases shown in Table III, even when the same number of test plans is generated by both methods. The mean value of D , computed as $D_{mean} = \sum i \cdot f_i / \sum f_i$,

FIGURE 11 Test Plan Distribution for Elliptical Wave Filter

has been presented to provide a quick reference about the relative performance of the two approaches as far as the the inter-iteration delay is concerned.

The results clearly demonstrate the effectiveness of the proposed approach in yielding a lower value of D . Since the value of D is a very important criterion for determining the total test time, it is logically expected that the test plans generated by the proposed algorithm will help in reducing the total testing time.

References

- [1] Koenemann B., J. Mucha, and G. Zwielfhoff, "Built-in logic block observation techniques", *Proc. Int. Test Conf.*, 1979, pp. 37-44.
- [2] Abadir, M.S., "A Knowledge Based System for Designing Testable VLSI Circuits", *Ph.D. dissertation*, Univ. of Southern California, May 1986.
- [3] Abadir, M.S. and Breuer, M.A., "Constructing optimal test schedules for VLSI circuits having built in test hardware", *15th International Conference on Fault Tolerant Computing*, June 1985.
- [4] Bhawmik, S., and P. Palchoudhuri, "DFT EXPERT: Designing Testable VLSI Circuits", *IEEE Design & Test of Computers*, Oct. 1989, pp. 8-17.
- [5] Haralick, R.M., "The Consistent Labeling Problem: Part I", *IEEE Trans. on PAMI*, Vol. PAMI-I, No. 2, Apr. 1979.
- [6] Paulin, P.G., "High-Level Synthesis of Digital Circuits Using Global Scheduling and Binding Algorithms", *Ph.D. dissertation*, Dep. of Electronics, Carleton Univ., Jan. 1988.
- [7] Sutarwala, S., "Automated Synthesis of Digital Systems from a Behavioral Description", *M.Sc. dissertation*, Dep. Comp. & Info. Sc., Univ. of Guelph, Canada, May 1988.
- [8] Midwinter, J., "Improving Interconnect and Register Allocation for the Behavioral Synthesis of Digital Circuits", *M. Eng. dissertation*, Dept. of Systems and Computer Engg., Carleton Univ., March 1988.
- [9] Tseng, C.-J., "Automated Synthesis of Data paths in Digital Systems", *Ph.D. dissertation*, Dep. Elec. Engg., Carnegie-Mellon University, Pittsburgh, Apr. 1984.

Authors' Biographies

Anupam Basu is an Associate Professor in the Department of Computer Science & Engineering at the Indian Institute of Technology, Kharagpur, India. He received his Ph.D degree from the same institute in 1989. His research interests include VLSI design automation, testing, and AI/expert systems.

Dilip K. Banerji is a Professor in the Department of Computing & Information Science at the University of Guelph, Guelph, Ontario. He holds a B. Tech

degree in Electrical Engineering from the Indian Institute of Technology, Kanpur, M.Sc. in Electrical Engineering from the University of Ottawa, Canada, and Ph.D in Computer Science from the University of Waterloo. His primary research interest is VLSI design automation, particularly high-level synthesis.

Amit Basu received his B. Tech degree in Computer Science & Engineering from the Indian Institute of Technology, Kharagpur in 1988, and M.Sc in Computer Science from the University of Guelph, Guelph, Ontario in 1990. Presently, he is with HAL Computer Corp., in California. His primary interests are VLSI synthesis and testing.

Thomas C. Wilson is an Associate Professor in the Department of Computing & Information Science at the University of Guelph, Guelph, Ontario. He holds

a BA degree from the University of Iowa, M.Sc from the University of Chicago, and Ph.D from the University of Waterloo. His research interests lie in the areas of distributed systems and VLSI-CAD, particularly high-level synthesis and retargetable code generation.

Jay C. Majithia is a Professor in the Department of Computing & Information Science at the University of Guelph, Guelph, Ontario. He holds a B.Sc degree in Electrical Engineering from the University of London, and M.Sc as well as Ph.D degrees in Electrical Engineering from McMaster University, Hamilton, Ontario. His main research areas are computer networks and VLSI design automation.