Built-In Generation of Weighted Test Sequences for Synchronous Sequential Circuits+

Irith Pomeranz and Sudhakar M. Reddy
Electrical and Computer Engineering Department
University of Iowa
Iowa City, IA 52242, U.S.A.

Abstract

We describe a method for on-chip generation of weighted test sequences for synchronous sequential circuits. For combinational circuits, three weights, 0, 0.5 and 1, are sufficient to achieve complete coverage of stuck-at faults, since these weights are sufficient to reproduce any specific test pattern. For sequential circuits, the weights we use are defined based on subsequences of a deterministic test sequence. Such weights allow us to reproduce parts of the test sequence, and help ensure that complete fault coverage would be obtained by the weighted test sequences generated.

1. Introduction

Methods to generate weighted pseudo-random patterns for combinational circuits were described in [1]-[15]. One of the methods that can be extended to sequential circuits is the one from [10]. This method is based on the use of three weights, 0, 0.5 and 1. A *weight assignment w* associates one of these weights with every primary input of the circuit. A preselected number of patterns N is applied under every weight assignment. A weight of 0.5 assigned to an input i by a weight assignment w implies that pseudo-random patterns are applied to input i while N test patterns are applied to the circuit; a weight of 0 assigned to input i implies that input i is held at 0 constantly for the N test patterns; and a weight of 1 assigned to input i implies that input i is held at 1 constantly for the N test patterns.

The weight assignments are computed in [10] based on a deterministic test set. Each weight assignment is obtained by intersecting a subset of deterministic test patterns. The intersection of identical values, 0 or 1, yields a weight of 0 or 1, respectively. The intersection of different values yields an unspecified value (x), which is translated into a weight of 0.5.

To apply this method to sequential circuits, a set of deterministic test subsequences was used in [10]. The intersection of test subsequences yielded weight assignments that were used in a similar way to the ones for combinational circuits. However, for sequential circuits, the intersection of a subset of test subsequences of length M results in M weight assignments that have to be used consecutively, and changed at every time unit. The need to change the weight assignment at every time unit is undesirable. In addition, this method is not applicable when a single test sequence is given for the circuit.

A more natural extension of the 3-weight approach to synchronous sequential circuits is to use a basic set of weights that are capable of producing test subsequences. Methods with extended sets of weights were described in [11] and [15] for the detection of delay faults, that require two-pattern tests. In addition to the weights {0,0.5,1}, the method of [11] uses two weights denoted by w_{01} and w_{10}. These weights correspond to the case where consecutive patterns are assigned complemented values, i.e., 0 (1) in the first pattern and 1 (0) in the second pattern. In the intersection of a subset of two-pattern tests, a weight of w_{01} (or w_{10}) is used for input i if input i changes from 0 to 1 (or from 1 to 0) in a large number of tests.

In this work, we describe an extension of the 3-weight approach of [10] and the 5-weight approaches of [11] and [15] to synchronous sequential circuits. Under the proposed scheme, a weight is represented by a subsequence α, and implemented by a finite-state machine (FSM) that produces the subsequence α repeatedly. For example, a weight of 001 is implemented by an FSM that produces the subsequence 001 repeatedly, to obtain the sequence $(001001\cdots001)$. The sequence α repeated r times is denoted by α^r. We describe the generation of the basic set of weights and the selection of weight assignments for synchronous sequential circuits based on a single deterministic test sequence given for the circuit.

Due to the use of a deterministic test sequence to guide the selection of weight assignments, the proposed method can achieve complete fault coverage for all the circuits considered. To reduce the number of weight assignments required to achieve complete fault coverage, we explore the possibility of placing observation points to detect some of the faults.

Different approaches to the on-chip generation of test sequences for synchronous sequential circuits were described in [16]-[22]. In the methods of [16]-[19], the on-chip test generator is connected only to the primary inputs of the circuit, and the flip-flops are not affected. The methods of [16] and [17] rely only on on-chip test sequence generation, and do not guarantee that complete fault coverage would be achieved. The methods of [18] and [19] are based on storage of test sequences, and can thus guarantee complete fault coverage. The method of [18] uses encoding of a deterministic test sequence to reduce memory requirements. The method of [19] loads into an on-chip memory subsequences of a deterministic test sequence, and then expands them on-chip into complete test sequences for the circuit. The method proposed here has the advantages of the methods of [16] and [17] in that no storage of test patterns is required. It has the advantages of the methods of [18] and [19] in that it achieves complete fault coverage (or the same fault coverage achieved by a deterministic test sequence). The methods of [20]-[22] modify the circuit flip-flops. In [20], a subset of the flip-flops are incorporated into a partial scan or *BIST* register and the resulting sequential circuit is tested using weighted random patterns. In [21], a hold mode is added to selected flip-flops. While a flip-flop is in the hold mode, its value does not change. This mode is used in order to apply to the combinational logic of the circuit an appropriately biased set of random patterns. In [22], partial reset

+ Research supported in part by NSF Grant No. MIP-9725053.

is used to bring the circuit into states that are required in order to detect hard-to-detect faults. This work is supported by the theory developed in [23], where it was shown that the availability of reset into one or more states is sufficient to detect every sequentially irredundant fault. The method proposed here belongs to the first class of techniques that do not modify the circuit flip-flops. Thus, it avoids the routing overhead for controlling the flip-flops, especially when the number of flip-flops is large.

The paper is organized as follows. In Section 2, we present an example to demonstrate the proposed approach. In Section 3, we describe the selection of basic weights from which weight assignments will be constructed, and the hardware generation of these weights. In Section 4, we describe the selection of weight assignments, and the corresponding hardware implementation. Experimental results are included in Section 5. Section 6 concludes the paper.

2. Example

A test sequence T for ISCAS-89 benchmark circuit $s27$ is shown in Table 1. For every time unit u, the test pattern included at time unit u of T is denoted by $T(u)$. The test sequence shown in Table 1 detects all the stuck-at faults in the circuit. The faults are denoted by $f_0, f_1, \cdots, f_{31}$. The time unit where fault f is detected by T is denoted by $u_{det}(f)$, and referred to as the *detection time* of f.

Table 1: A test sequence

	$T(u)$			
u	$i=0$	$i=1$	$i=2$	$i=3$
0	0	1	1	1
1	1	0	0	1
2	0	1	1	1
3	1	0	0	1
4	0	1	0	0
5	1	0	1	1
6	1	0	0	1
7	0	0	0	0
8	0	0	0	0
9	1	0	1	1

Table 2: A weighted sequence

	$T_G(u)$			
u	$i=0$	$i=1$	$i=2$	$i=3$
0	0	0	1	1
1	1	0	0	1
2	0	0	0	1
3	1	0	1	1
4	0	0	0	1
5	1	0	0	1
6	0	0	1	1
7	1	0	0	1
8	0	0	0	1
9	1	0	1	1
10	0	0	0	1
11	1	0	0	1

We use T_i to denote the sequence T restricted to input i. Thus, $T_i(u)$ is the value assigned by T to input i at time unit u. In Table 1, $T_0 = (0101011001)$, $T_1 = (1010100000)$, and so on.

Next, we consider weights represented by subsequences of lengths $L_S = 1$, 2 and 3 in order to create weight assignments and test sequences that detect the faults of $s27$. A weight represented by a subsequence α, when assigned to an input i, implies that the sequence α^r is assigned to input i. Therefore, a subsequence α of length L_S selected for input i can be used to generate a sequence α^r that matches the sequence T_i perfectly at least at L_S time units. Our goal is to obtain perfect matches between the sequences α^r and T_i around the detection times of faults, thus maximizing the fault coverage achieved by each test sequence. Let us consider time unit $u = 9$ of T and the input $i = 0$ in Table 1. Two faults are detected at time unit 9, f_{10} and f_{12}. Using $L_S = 1$ and $\alpha = 1$, and repeating α at least ten times, we obtain the sequence $\alpha^r = (1111111111\cdots)$ that matches T_0 perfectly at time unit $u = 9$, where $T_0(u) = 1$. In addition, this sequence matches T_0 at a total of five time units where $T_0(u) = 1$. Using $L_S = 2$, we can repeat the subsequence $\alpha = 01$ five times or more to obtain the sequence $(0101010101\cdots)$.This sequence matches T_0 perfectly at time units 8 and 9. In addition, this sequence matches T_0 at a total of 8 time units. Finally, using $L_S = 3$, we can repeat the subsequence $\alpha = 100$ four times or more to obtain the sequence $(1001001001\cdots)$. This sequence matches T_0 perfectly at time units 7, 8 and 9. In addition, it matches T_0 at a total of 7 time units. Of the three subsequences 1, 01 and 100, the subsequence 01 results in the largest number of matches with T_0. Therefore, we select this subsequence to define the weight for input $i = 0$.

We continue to consider the other inputs in the same way. For input $i = 1$, the subsequences 0, 00 and 000 are the only subsequences of lengths $L_S = 1$, 2 and 3 that allow perfect matching with T_1 at the last L_S time units until time unit 9. All these subsequences produce the same sequence $\alpha^r = (0000000000\cdots)$ that matches T_1 at a total of seven time units. We select the subsequence 0 for input $i = 1$.

For input $i = 2$, we select the subsequence 100 that results in the sequence $(1001001001\cdots)$. This sequence matches T_2 perfectly at time units 7, 8 and 9, and it matches T_2 at a total of six time units.

For input $i = 3$, we select the subsequence 1 that results in the sequence $(1111111111\cdots)$. This sequence matches T_3 perfectly at time unit 9, and it matches T_3 at a total of seven time units.

Using the subsequences selected above to generate a sequence of length 12, we obtain the sequence T_G shown in Table 2. This sequence detects f_{10} as well as eight additional faults.

The weights above were selected based on the subsequences that yielded the best matches with the sequences T_i. It is also possible to use the second-best matches to obtain an additional test sequence. The second-best matches consist of the subsequence 100 for input 0 that results in 7 matches, the subsequence 00 for input 1 that results in 7 matches, the subsequence 01 for input 2 that results in 5 matches, and the subsequence 100 for input 3 that results in 7 matches. Using these subsequences, we obtain a weighted sequence that detects 4 additional faults.

Additional weight assignments can be used to detect the remaining faults. We point out that certain subsequences result in the same sequence after repetition. For example, $\alpha_1 = 0$ and $\alpha_2 = 00$ result in the same sequence after repetition, $(000\cdots)$. Similarly, $\alpha_1 = 01$ and $\alpha_2 = 0101$ result in the same sequence, $(010101\cdots)$. However, for the purpose of selecting weight assignments, it is more convenient to keep these different subsequences even though they produce identical sequences when repeated.

3. Selection of weights

In this section, we first describe the hardware implementation of weights. We then describe a procedure for selecting weights. The construction of weight assignments is considered in the next section.

A given weight represented by a subsequence α is implemented by an FSM that produces the sequence α. Two or more subsequences $\alpha_1, \alpha_2, \cdots, \alpha_m$ of the same length can be implemented by the same FSM. For illustration, we show in Table 3 an FSM the produces the subsequences 00010, 01011 and 11001. After resetting the machine to state A, it will produce the sequences $(00010)^r$ on z_1, $(01011)^r$ on z_2 and $(11001)^r$ on z_3, until it is reset again.

To implement m subsequences of length L_S, we use an FSM with L_S states and m outputs. In the implementation of the

Table 3: An FSM for three weights

PS	NS	z_1	z_2	z_3
A	B	0	0	1
B	C	0	1	1
C	D	0	0	0
D	E	1	1	0
E	A	0	1	1

machine, we have $\lceil \log_2 L_S \rceil$ state variables and therefore $2^{\lceil \log_2 L_S \rceil}$ states, only L_S of them reachable from the initial state. We use one FSM for every set of subsequences of the same length. Thus, in general, comparing output functions corresponding to subsequences of lengths L_{S_1} and L_{S_2} such that $L_{S_1} < L_{S_2}$, the following observations can be made. (1) If $\lceil \log_2 L_{S_1} \rceil < \lceil \log_2 L_{S_2} \rceil$, then output functions corresponding to subsequences of length L_{S_1} require fewer state variables than output functions corresponding to subsequences of length L_{S_2}. (2) If $\lceil \log_2 L_{S_1} \rceil = \lceil \log_2 L_{S_2} \rceil$, then output functions corresponding to subsequences of length L_{S_1} have more unspecified values because of unreachable states than output functions corresponding to subsequences of length L_{S_2}.

Based on the observations above, we prefer to use the shortest possible subsequences in defining weight assignments for a given circuit. In addition, we would like to use the smallest possible number of different subsequences. We achieve these goals by considering the time units of T, at which faults are detected, one at a time, until all the faults are detected. For every time unit u, we select subsequences of increasing length that allow us to reproduce T as closely as possible, with a perfect match around time unit u. We increase the subsequence length until the subsequences we already selected allow us to detect all the faults detected by T at time unit u. The details of the procedure are given next.

We maintain a set of subsequences S which is initially empty. We also maintain a set of faults F that contains all the target faults which have not been detected yet. Initially, F contains all the target faults. At every iteration, we select a detection time u and a length L_S, and we extend S by adding to it subsequences of length L_S. We then construct weight assignments and test sequences based on the new weights in S, and drop the detected faults. The selection of u and the extension of S are done as follows.

We consider the largest time unit u for which there exists a yet-undetected fault $f \in F$ such that $u_{det}(f) = u$. The considerations behind this choice are the following. Faults with higher detection times tend to be more difficult to detect, and their test sequences tend to detect higher numbers of other faults. Once we define new subsequences and include them in S, we also define weight assignments, and generate test sequences based on them. Faults detected by these test sequences are dropped from F. Thus, starting from the detection times of the more difficult to detect faults is likely to minimize the total number of subsequences and the total number of weight assignments required to detect all the target faults.

After selecting a detection time u, we generate subsequences of lengths $L_S = 1, 2, \cdots$ that can be used to reproduce subsequences of T of length L_S that end at time unit u. The generation of subsequences of a given length L_S is demonstrated by the following example. We consider $s27$ under the test sequence given in Table 1. Let $u = 8$ and $L_S = 4$. For input $i = 0$, the subsequence of length 4 ending at time unit 8 is 1100. Based on our approach, we can obtain this subsequence at time units 5 to 8 by applying a subsequence α starting at time unit 0, and repeating α as many times as necessary. Our goal is to obtain $\alpha^r(u') = T_0(u')$ for $5 \le u' \le 8$ (i.e., the sequence α^r produces the value of T_0 at every time unit u' such that $5 \le u' \le 8$). At an arbitrary time unit u', the sequence α^r has the vector included in α at time unit $u' \% L_S$, where % is the modulo operation. For example, $\alpha(0)$ appears at time units 0, L_S, $2L_S$, $\cdots$ of the sequence α^r; $\alpha(1)$ appears at time units 1, $L_S + 1$, $2L_S + 1$, $\cdots$ of the sequence α^r; and so on. Thus, we determine α based on the equation $T_0(u') = \alpha^r(u') = \alpha(u' \% L_S)$ for $5 \le u' \le 8$. In the example of the subsequence 1100 at time units 5 to 8, we obtain $\alpha(5\%4) = \alpha(1) = T_0(5) = 1$, $\alpha(6\%4) = \alpha(2) = T_0(6) = 1$, $\alpha(7\%4) = \alpha(3) = T_0(7) = 0$ and $\alpha(8\%4) = \alpha(0) = T_0(8) = 0$, or $\alpha = 0110$. Repeating α, we obtain the sequence $(011001100\cdots)$ which matches T_0 perfectly at time units 5 to 8. For input $i = 1$, we use $\alpha = 0000$; for input $i = 2$, we use $\alpha = 0100$; for input $i = 3$, we use the same subsequence α used for $i = 0$. Each one of the subsequences 0110, 0000 and 0100 is added to S.

If S is extended using time unit u and $L_S = u + 1$, S contains subsequences that allow us to reproduce T completely up to time unit u. This guarantees the detection of all the faults detected by T at time unit u or earlier. When $u = L - 1$, where L is the length of T, and $L_S = u + 1 = L$, the complete test sequence T can be reproduced. In practice, complete fault coverage can be achieved with subsequences that are much shorter than T.

4. Selection of weight assignments

Given a set of weights (or subsequences) S and a maximum subsequence length L_S, we describe in this section the selection of weight assignments based on S.

4.1 Selecting weight assignments

Weight assignments are constructed around the detection time u of a yet-undetected fault f (the detection time is the time unit at which f is detected by the given test sequence T). Next, we describe the selection of weight assignments for a given detection time u of a yet-undetected fault f.

For every input i and every subsequence length $L'_S \le L_S$, we find every subsequence $\alpha \in S$ of length L'_S that results in a perfect match with the last L'_S time units of T_i until time unit u. A perfect match is obtained if $T_i(u') = \alpha(u' \% L'_S)$ for $u - L'_S + 1 \le u' \le u$. We denote by A_i the set of subsequences out of S yielding perfect matches with the last L'_S time units of T_i until time unit u. For example, we consider $s27$ with the deterministic test sequence shown in Table 1, the set of weights S shown in Table 4, and $L_S = 3$. Considering time unit 9 where f_{10} is detected, we obtain the sets A_i shown in Table 5. To the left of each subsequence, we show its index in S. For every subsequence $\alpha_{i,j} \in A_i$, we compute the number of time units where $\alpha^r_{i,j}$ matches T_i, i.e., the number of time units u' where $\alpha^r_{i,j}(u') = T_i(u')$. We denote this number by n_m, and include it in Table 5. We order the subsequences in A_i by decreasing value of n_m.

Table 4: A set of weights for $s27$

j	0	1	2	3	4	5	6	7	8	9
α_j	0	1	00	10	01	11	000	100	010	110

j	10	11	12	13
α_j	001	101	011	111

Table 5: The sets A_i for $s27$

	A_0		A_1		A_2		A_3	
j	$\alpha_{0,j}$	n_m	$\alpha_{1,j}$	n_m	$\alpha_{2,j}$	n_m	$\alpha_{3,j}$	n_m
0	(4)01	8	(0)0	7	(7)100	6	(1)1	7
1	(7)100	7	(2)00	7	(4)01	5	(7)100	7
2	(1)1	5	(6)000	7	(1)1	4	(4)01	6

It is interesting to note that in Table 5, there is no apparent relationship between the number of matches n_m and the lengths of the subsequences. In general, sorting the subsequences according to decreasing value of n_m may place subsequences of smaller lengths higher in the list. This has an advantage in ensuring that the test sequences generated have larger periods than the lengths of the individual subsequences they are made up of.

Our goal in selecting the weight assignments is to maximize the number of matches, since this is likely to maximize the number of faults detected. Therefore, we start with a weight assignment based on $\alpha_{i,0}$ for every input i. Since we keep the subsequences in A_i sorted by decreasing value of n_m, $\alpha_{i,0}$ has the largest value of n_m of all the subsequences in A_i. By selecting $\alpha_{i,0}$ for every input i, we select the weight assignment with the largest number of matches for all the inputs. For $s27$, we select the weight assignment based on the subsequences 01, 0, 100 and 1. We generate a sequence based on this weight assignment, fault simulate it, and drop the faults detected. For $s27$, the weight assignment above results in the sequence of Table 2. Recall that the sets A_i were computed based on the detection time u of a yet-undetected fault f. If any fault f' with the same detection time as f is left undetected, we consider next the weight assignment based on $\alpha_{i,1}$ for every input i. For $s27$, this implies the weight assignment based on the subsequences 100, 00, 01 and 100. Again, we generate a sequence based on this weight assignment, fault simulate it, and drop the faults detected. We repeat this process using weight assignments based on u as long as an undetected fault f' exists with $u_{det}(f') = u$. With every weight assignment we increase the value of j. This ensures that new weight assignments are considered at every iteration.

In spite of the advantages of sorting the subsequences by decreasing number of matches, the longest subsequences have an advantage in that their matches are obtained at the last time units before the detection time u of a yet-undetected target fault f. Duplicating T at the time units before the detection time of f is likely to enable us to detect f. To accommodate this observation, we modify the sets A_i as follows. After arranging the sets A_i as described above, we check whether there exists a weight assignment $w_j = \{\alpha_{i,j}: 1 \le i \le n\}$ such that the length of $\alpha_{i,j}$ is L_S for every input i. If no such weight assignment exists, we redefine A_i for every i by adding at its beginning the subsequence $\alpha_{i,k} \in A_i$ with length L_S.

4.2 Overall procedure

The overall procedure for selecting weight assignments proceeds as described next.

The procedure starts with the set of target faults F consisting of all the faults detected by the deterministic test sequence T. Considering the detection times u in decreasing order, and allowing the subsequence lengths L_S to increase starting from $L_S = 1$, the procedure first extends S based on u and L_S. It then constructs the sets A_i by using subsequences of length at most L_S included in S. Based on the sets A_i, the procedure generates weight assignments of the form $w_j = \{\alpha_{i,j}: 1 \le i \le n\}$. However, not all the weight assignments defined by the sets A_i are considered. For w_j to be considered, at least one subsequence $\alpha_{i,j}$ included in it must be of length L_S. For every weight assignment that satisfies this condition, the procedure generates a test sequence T_G of length L_G, where L_G is a preselected constant. The sequence T_G is simulated, and all the faults detected by T_G are dropped from F. The weight assignments producing test sequences that are useful in detecting any yet-undetected faults are stored in a set Ω.

The worst-case complexity of the proposed procedure is as follows. In the worst case, the procedure performs a number of iterations equal to the number of target faults, N_F. For every fault, we derive at most L subsequences for every one of N_{PI} primary inputs. The derivation of a subsequence has complexity $O(L)$. Thus, we have $O(N_F L^2 N_{PI})$ for the complexity of deriving the subsequences. For every possible subsequence length L_S, we have at most N_{PI} weight assignments (since only weight assignments that have at least one subsequence of length L_S are considered). For each weight assignment, we fault simulate a sequence of length L_G. The total complexity of simulation is therefore equal to the simulation of $O(N_F L N_{PI})$ sequences of length L_G. The simulation effort is the dominant part of this computation. We reduce it by first simulating a sample of faults under every sequence T_G, including the fault for which T_G was generated. If no fault out of the sample is detected, simulation of T_G stops.

4.3 Postprocessing

The set of weight assignments Ω is constructed by first considering weight assignments based on short subsequences, and then extending the subsequence lengths as necessary to detect a yet-undetected fault. This structure of the procedure is advantageous in ensuring that the subsequences used would be as short as possible. However, it may result in the inclusion of *redundant* weight assignments in Ω. A weight assignment $\Omega_j \in \Omega$ is redundant if weight assignments $\Omega_{k_1}, \Omega_{k_2}, \cdots$, generated after Ω_j, produce test sequences that detect all the faults detected by the sequence based on Ω_j. We remove redundant sequences from Ω by performing reverse order simulation of the weight assignments in Ω. During this simulation process, the weight assignments in Ω are considered in reverse order of generation. When a weight assignment Ω_j is considered, the test sequence T_G based on Ω_j is generated and fault simulated using the set of faults F. Initially, F includes all the target faults. When Ω_j is considered, all the faults out of F detected by T_G are dropped from F. If no faults out of F are detected by T_G, then Ω_j is removed from Ω.

4.4 Implementation

To implement a test sequence generator based on the set Ω, we use the structure shown in Figure 1. For the figure, we assume that the circuit-under-test (*CUT*) has three inputs, I_0, I_1 and I_2, and that Ω contains four weight assignments, $\Omega_1, \cdots, \Omega_4$. We assume that Ω_j consists of subsequences $\alpha_{i,j}$. Each subsequence $\alpha_{i,j}$ is generated by an FSM as described in Section 3. The control inputs s_1 and s_2 come from a binary counter that advances every L_G clock cycles, where L_G is the length of the sequence T_G applied based on every weight assignment.

In the implementation above, we do not allow pseudo-random sequences (or *LFSR* sequences) on the circuit inputs. Adding this option is likely to reduce the number of subsequences that need to be generated.

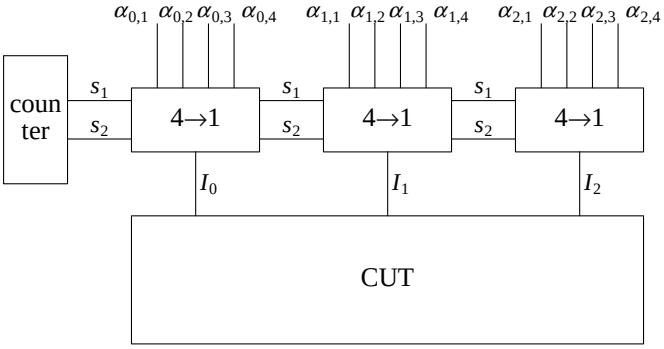

Figure 1: A test sequence generator

5. Experimental results

We applied the proposed procedure to ISCAS-89 benchmark circuits. The results are reported in this section. We report the results of the proposed procedure when it is used to achieve complete fault coverage, and then consider the proposed procedure in conjunction with the insertion of observation points. Observation points are not necessary for achieving complete fault coverage; however, they can allow complete fault coverage to be achieved with fewer weight assignments.

The length of the test sequences generated based on each weight assignment is $L_G = 2000$. As a deterministic test sequence T, we use test sequences generated by the test generation procedures *STRATEGATE* [24] and *SEQCOM* [25], and compacted by static compaction.

The results of the proposed procedure are reported in Table 6. After the circuit name, we show the length of the test sequence T, and the number of faults it detects. Under column *proposed*, we show the results obtained by the proposed procedure after reverse order simulation. We show the number of test sequences that detected yet-undetected faults, which is also the number of weight assignments included in Ω after reverse order simulation. Next, we show the number of subsequences that define these weight assignments, and the maximum length of any of these subsequences. The fault coverage achieved by the weight assignments computed by the proposed procedure is the same as the fault coverage of the deterministic test sequence T in every case. From Table 6, it can be seen that the maximum length of any subsequence is significantly shorter than the length of the deterministic test sequence. For *s*1196, subsequences of length at most three are sufficient to produce weighted test sequences that achieve complete fault coverage.

Next, we provide information about the FSMs required to implement the weight assignments selected after reverse order simulation. As discussed above, we implement all the subsequences of the same length by a single FSM. The different subsequences of the same length are implemented by different outputs of the FSM. Thus, the number of FSMs is equal to the number of different subsequence lengths, and the number of outputs for all the FSMs is equal to the number of subsequences. In some cases, we obtain two different subsequences α_1 and α_2 that produce identical sequences when they are repeated. For example, $\alpha_1 = (01)$ and $\alpha_2 = (0101)$ produce the same sequence, $(010101\cdots)$, when they are repeated. In such a case, we eliminate α_2 and use α_1 instead. Under column *FSMs* of Table 6, we show the number of FSMs required to implement all the weight assignments, and the total number of outputs for all the FSMs. It can be seen that the number of different FSMs is in most cases

Table 6: Experimental results

	given seq		proposed			FSMs	
circuit	len	det	seq	subs	len	num	out
s208	105	137	10	39	18	14	38
s298	117	265	3	9	44	7	9
s344	57	329	9	60	8	8	56
s382	516	364	5	15	211	9	15
s386	121	314	20	94	14	13	80
s400	611	380	4	12	154	8	12
s420	108	179	5	90	18	11	90
s444	608	424	4	12	231	8	12
s526	1006	454	11	32	161	28	32
s641	101	404	10	145	10	10	127
s820	491	814	14	244	86	28	236
s1196	238	1239	151	14	3	3	10
s1423	1024	1414	15	223	201	46	219
s1488	455	1444	6	46	225	16	46
s5378	646	3639	27	701	25	25	679
s35932	150	35100	14	445	53	23	436

significantly smaller than the number of subsequences. Nevertheless, the number of FSMs may be large in some cases. For such cases, we explore the use of observation points to reduce the number of weight assignments, and thus, the number of subsequences and FSMs required.

The observation point insertion experiment proceeds as follows. We apply the proposed procedure to generate the set of weight assignments Ω (before reverse order simulation). We then select weight assignments out of Ω one at a time. Initially, we have an empty set of selected weight assignments Ω_{lim}, and a set of faults F that contains all the target faults. We select the weight assignment $\Omega_j \in \Omega$ that detects the largest number of faults out of F. We add Ω_j to Ω_{lim}, simulate the test sequence defined by Ω_j, and drop from F all the faults it detects. This is repeated until all the faults in F are detected.

For every set Ω_{lim}, we use observation points to increase the fault coverage achieved by Ω_{lim}. To select the lines on which observation points will be inserted, we perform the following computation. For every fault $f \in F$ (F contains the faults not detected by Ω_{lim}), we compute a set of lines $OP(f)$ such that if an observation point is added on any line $g \in OP(f)$, f will be detected by one of the sequences defined by Ω_{lim}. We then use a covering procedure to select a minimal number of lines OP such that for every $f \in F$, if $OP(f) \neq \phi$, OP contains at least one line out of $OP(f)$. The set OP defines the set of observation points for the circuit. If OP contains a line out of $OP(f)$, then f is detected on one of the selected observation points.

The results of the experiment above are reported in Tables 7-16 as follows. For every Ω_{lim}, we report the following information. We show the number of test sequences used (which is also the number of weight assignments in Ω_{lim}), the number of subsequences defining these weight assignment, the length of the longest subsequence, and the fault efficiency achieved. The fault efficiency is defined as the number of faults detected by Ω_{lim} divided by the number of faults detected by Ω. Next, we show the number of observation points added to the circuit based on Ω_{lim}, and the fault efficiency achieved using these observation points. We only report the results when the final fault efficiency is 99% or higher. For space considerations, we only report on some of the circuits considered in Table 6.

As may be expected, there is a tradeoff between the number of weight assignments selected, and the number of observation points required to improve the fault efficiency. In some

cases, there is a minimum number of weight assignments necessary to allow 100% fault efficiency to be achieved by insertion of observation points.

Note that the results in Tables 7-16 for 100% fault efficiency without observation points may be different from the results of Table 6. This is because in Table 6, we used reverse order simulation on the set Ω produced by the proposed procedure, whereas observation point insertion is done with a different selection procedure for including weight assignments in Ω_{lim}.

6. Concluding remarks

We described a method for on-chip generation of weighted test sequences for synchronous sequential circuits. The weights we used were defined based on subsequences of a deterministic test sequence. A weight represented by a subsequence α assigned to input i implies that input i assumes the sequence α^r obtained by repeating α r times. The use of a deterministic test sequence to define the weights allowed us to reproduce parts of the test sequence, and helped ensure that complete fault coverage would be obtained. We described a procedure for defining a set of weights from which weight assignments can be constructed, a procedure for selecting weight assignments so as to detect target faults, and presented experimental results to demonstrate that complete fault coverage can be achieved by this method. We also investigated the tradeoff between the number of weight assignments and the number of observation points required to achieve complete fault coverage. The use of pure-random sequences as part of the weight scheme, followed by the synthesis of the on-chip test generation hardware, are the subject of future work.

References

[1] H. D. Shnurmann, E. Lindbloom and R.G. Carpenter, "The weighted Random Test-Pattern Generator", IEEE Trans. on Computers, pp. 695-700, July 1975.

[2] R. Lisanke, F. Brglez, A. De Geus and D. Gregory, "Testability-Driven Random Pattern Generation", in Proc. Intl. Conf. on Computer-Aided Design, 1986, pp. 144-147.

[3] J. A. Waicukauski and E. Lindbloom, "Fault Detection Effectiveness of Weighted Random Patterns", in Proc. Intl. Test Conf., pp. 245-250, 1988.

[4] H.-J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns", in Proc. Intl. Test Conf., 1988 pp. 236-244.

[5] F. Siavoshi, "WTPGA: A Novel Weighted Test Pattern Generation Approach for VLSI Built-In Self-Test", in Proc. Intl. Test Conf., 1988, pp. 256-262.

[6] R. W. Bassett et. al., "Low Cost Testing of High Density Logic Components", in Proc. Intl. Test Conf., 1989, pp. 550-557.

[7] F. Brglez, C. Gloster and G. Kedem, "Hardware-Based Weighted Random Pattern Generation for Boundary Scan", in Proc. Intl. Test Conf., 1989, pp. 264-273.

[8] F. Muradali, V. K. Agarwal, B. Nadeau-Drostie, "A New Procedure for Weighted Random Built-In Self-Test", in Proc. Intl. Test Conf., pp. 660-669, 1990.

[9] S. Pateras and J. Rajski, "Generation of Correlated Random Patterns for the Complete Testing of Synthesized Multi-Level Circuits", in Proc. Design Autom. Conf., June 1991.

[10] I. Pomeranz and S. M. Reddy, "3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set", IEEE Trans. on Computer-Aided Design, July 1993, pp. 1050-1058.

[11] I. Pomeranz and S. M. Reddy, "On the Generation of Weights for Weighted Pseudo Random Testing", in Proc. 1993 VLSI Design Conf., Jan. 1993, pp. 69-72.

[12] J. Hartmann and G. Kemnitz, "How To Do Weighted Random Testing for BIST?", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 1993, pp. 568-571.

[13] R. Kapur, S. Patil, T. J. Snethen and T. W. Williams, "Design of an Efficient Weighted Random Pattern Generation System", in Proc. Intl Test Conf., Oct. 1994, pp. 491-500.

[14] B. Reeb and H.-J. Wunderlich, "Deterministic Pattern Generation for Weighted Random Pattern Testing", in Proc. Europ. Design and Test Conf., 1996, pp. 30-36.

[15] S. Cremoux, C. Fagot, P. Girard, C. Landrault and S. Pravossoudovich, "A New Test Pattern Generation Method for Delay Fault Testing," in Proc. VLSI Test Symp., 1996, pp. 296-301.

[16] L. Nachman, K. K. Saluja, S. Upadhyaya and R. Reuse, "Random Pattern Testing for Sequential Circuits Revisited", in Proc. 26th Fault-Tolerant Computing Symp., June 1996, pp. 44-52.

[17] I. Pomeranz and S. M. Reddy, "Built-In Test Generation for Synchronous Sequential Circuits", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 1997, pp. 421-426.

[18] V. Iyengar, K. Chakrabarty, and B. T. Murray "Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets," in Proc. VLSI Test Symp., April 1998, pp. 418-422.

[19] I. Pomeranz and S. M. Reddy, "Built-In Test Sequence Generation for Synchronous Sequential Circuits Based on Loading and Expansion of Test Subsequences", in Proc. 36th Design Autom. Conf., June 1999.

[20] H. Wunderlich, "The Design of Random-Testable Sequential Circuits", in Proc. 19th Fault-Tolerant Computing Symp., June 1989, pp. 110-117.

[21] F. Muradali, T. Nishida and T. Shimizu, "A Structure and Technique for Pseudorandom-Based Testing of Sequential Circuits", Journal of Electronic Testing: Theory and Applications, 1995, pp. 107-115.

[22] M.-L. Flottes, C. Landrault and A. Petitqueux, "Partial Set for Flip-Flops based on State Requirement for Non-scan BIST Scheme", in Proc. Europ. Test Workshop, pp. 104-109, May 1999.

[23] I. Pomeranz and S. M. Reddy, "On Full Reset as a Design-for-Testability Technique", in Proc. 1997 VLSI Design Conf., Jan. 1997, pp. 534-536

[24] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal", in Proc. 1996 Europ. Design & Test Conf., March 1996, pp. 22-28.

[25] I. Pomeranz and S. M. Reddy, "On Generating Compact Test Sequences for Synchronous Sequential Circuits", in Proc. EURO-DAC '95, Sept. 1995.

Table 7: Observation point insertion for *s*208

circuit	seq	sub	len	f.e.	obs	f.e.
s208	2	15	18	93.4	7	100.0
	3	20	18	95.6	4	100.0
	4	23	18	97.8	3	100.0
	5	26	18	98.5	2	100.0
	6	28	18	99.3	1	100.0
	7	30	18	100.0	0	100.0

Table 8: Observation point insertion for *s*298

circuit	seq	sub	len	f.e.	obs	f.e.
s298	1	3	17	98.1	4	100.0
	2	6	44	99.6	1	100.0
	3	9	44	100.0	0	100.0

Table 9: Observation point insertion for *s*344

circuit	seq	sub	len	f.e.	obs	f.e.
s344	4	30	8	97.0	9	100.0
	5	35	8	98.2	6	100.0
	6	44	8	99.1	3	100.0
	7	53	8	99.7	1	100.0
	8	57	8	100.0	0	100.0

Table 10: Observation point insertion for *s*386

circuit	seq	sub	len	f.e.	obs	f.e.
s386	2	12	10	74.2	33	99.0
	3	19	14	79.9	26	99.4
	4	21	14	85.7	20	99.4
	5	24	14	88.8	17	99.7
	6	25	14	91.1	13	99.7
	7	31	14	93.0	12	100.0
	8	38	14	94.6	10	100.0
	9	41	14	95.5	8	100.0
	10	47	14	96.5	7	100.0
	12	56	14	97.8	6	100.0
	13	63	14	98.1	5	100.0
	14	66	14	98.4	4	100.0
	16	75	14	99.0	3	100.0
	17	82	14	99.4	2	100.0
	18	88	14	99.7	1	100.0
	19	91	14	100.0	0	100.0

Table 11: Observation point insertion for *s*400

circuit	seq	sub	len	f.e.	obs	f.e.
s400	1	3	116	96.3	12	99.7
	2	6	154	98.2	7	100.0
	3	9	154	99.7	1	100.0
	4	12	154	100.0	0	100.0

Table 12: Observation point insertion for *s*420

circuit	seq	sub	len	f.e.	obs	f.e.
s420	2	37	18	97.2	3	100.0
	3	56	18	98.9	2	100.0
	4	75	18	99.4	1	100.0
	5	91	18	100.0	0	100.0

Table 13: Observation point insertion for *s*526

circuit	seq	sub	len	f.e.	obs	f.e.
s526	1	3	138	95.4	18	100.0
	2	6	161	96.9	14	100.0
	3	9	161	98.0	9	100.0
	4	12	161	98.5	7	100.0
	5	15	161	98.9	5	100.0
	6	18	161	99.3	3	100.0
	7	21	161	99.6	2	100.0
	8	24	161	99.8	1	100.0
	9	27	161	100.0	0	100.0

Table 14: Observation point insertion for *s*641

circuit	seq	sub	len	f.e.	obs	f.e.
s641	3	81	10	96.5	12	100.0
	4	94	10	98.3	6	100.0
	5	106	10	99.3	3	100.0
	6	118	10	99.7	1	100.0
	7	133	10	100.0	0	100.0

Table 15: Observation point insertion for *s*1423

circuit	seq	sub	len	f.e.	obs	f.e.
s1423	4	68	201	98.80	9	100.00
	5	85	201	99.43	7	100.00
	6	102	201	99.72	4	100.00
	7	118	201	99.86	2	100.00
	8	135	201	99.93	1	100.00
	9	150	201	100.00	0	100.00

Table 16: Observation point insertion for *s*5378

circuit	seq	sub	len	f.e.	obs	f.e.
s5378	2	64	24	94.17	78	99.20
	3	89	24	96.70	39	99.81
	5	155	24	97.86	31	100.00
	6	189	24	98.13	30	100.00
	7	223	24	98.38	27	100.00
	8	253	24	98.60	20	100.00
	9	286	24	98.82	19	100.00
	10	320	24	99.04	17	100.00
	11	338	24	99.23	12	100.00
	12	372	24	99.37	11	100.00
	13	388	24	99.48	8	100.00
	14	423	24	99.59	7	100.00
	17	504	24	99.81	5	100.00
	18	534	24	99.86	4	100.00
	20	572	25	99.92	3	100.00
	21	574	25	99.95	2	100.00
	22	594	25	99.97	1	100.00
	23	629	25	100.00	0	100.00