Built-In Test Generation for Synchronous Sequential Circuits

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

Abstract

We consider the problem of built-in test generation for synchronous sequential circuits. The proposed scheme leaves the circuit flip-flops unmodified, and thus allows at-speed test application. We introduce a uniform, parametrized structure for test pattern generation. By matching the parameters of the test pattern generator to the circuit-under-test, high fault coverage is achieved. In many cases, the fault coverage is equal to the fault coverage that can be achieved by deterministic test sequences. We also investigate a method to minimize the size of the test pattern generator, and study its effectiveness alone and in conjunction with the insertion of test-points.

1. Introduction

A large number of techniques for built-in self-test (*BIST*) were developed for combinational circuits and fully-scanned sequential circuits [1], [2]. *BIST* for synchronous sequential circuits has received limited attention [3]-[5]. The need to develop *BIST* techniques for synchronous sequential circuits is motivated by the fact that if the combinational logic of the circuit is not tested separately, then there is no need to configure the circuit flip-flops into a test pattern generator during test mode, and the circuit flip-flops can be left unmodified. This allows at-speed testing of the circuit under its normal operation conditions.

Built-in test generation methods for synchronous sequential circuits that modify some of the circuit flip-flops were described in [3] and [4]. In [3], 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 [4], 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. For example, if a state variable y assumes the value 1 only rarely, then by holding a state that assigns the value 1 to y, the average frequency with which a 1 is obtained on y can be increased. A similar method was used in [6], except that in [6] all the flip-flops are controlled together, whereas in [4] the hold mode of every modified flip-flop is controlled independently. The methods of [3] and [4] modify the circuit flip-flops to increase the set of reachable states, thus increasing the fault coverage that can be obtained. In this work, we are interested in maximizing the fault coverage by selecting a built-in test pattern generator for primary input sequences without modifying the circuit flip-flops. Techniques that modify the sequential behavior of the circuit can be used to further increase the fault coverage if necessary.

+ Research supported in part by NSF Grant No. MIP-9220549, and in part by NSF Grant No. MIP-9357581

A method to generate input sequences to test a synchronous sequential circuit using built-in test pattern generators without modifying the circuit flip-flops was proposed in [5]. In [5], a sequence of pseudo-random primary input patterns produced by an *LFSR* is modified by holding selected input values fixed for several time units. Experimental results for benchmark circuits are reported in [5], showing fault coverages which are close to the fault coverages achieved by deterministic test sequences. Although the fault coverages reported in [5] are very high, they are not complete. In this work, we propose a test generation scheme suitable for *BIST* that allows higher fault coverages to be obtained. In many cases, the fault coverage is as high as that achieved by deterministic test generation. Under the proposed scheme, a test pattern is applied to the circuit every clock cycle, and allows at-speed testing of the circuit.

An effective test pattern generator (*TPG*) for *BIST* must be a circuit having an area efficient implementation, capable of generating test sequences that achieve high fault coverages. To satisfy these goals we use a preselected parametrized *TPG* structure. We determine the parameters based on circuit properties so as to maximize the fault coverage achieved; however, the overall structure of the *TPG* remains the same. We use in this work a randomized search for the appropriate parameters. We expect that a deterministic search using appropriate heuristics or a directed search such as genetic optimization would allow even higher fault coverages to be obtained at lower area overheads.

The basic structure of the *TPG* is as follows. A k-bit counter is allowed to cycle through its 2^k states a number of times denoted by the parameter R. The state of the counter at time unit u is equal to the binary representation of u, and it is denoted by $Q[u]$. The sequence of counter states is denoted by Q. We have $Q = (0\ 1 \cdots 2^k-1\ 0\ 1 \cdots 2^k-1 \cdots)$. Input i of the circuit-under-test (*CUT*) is determined by a decoding logic driven by the counter and characterized by two parameters L_i and U_i such that $0 \le L_i \le U_i \le 2^k-1$. Input i is set to 1 when $L_i \le Q[u] \le U_i$; otherwise, input i is set to 0. Thus, the sequence assigned to input i consists of L_i 0s followed by $U_i - L_i + 1$ 1s and $2^k - 1 - U_i$ 0s. This sequence is repeated R times. For example, for $k = 3$, $L_i = 2$ and $U_i = 6$, the sequence (00111110), repeated R times, is applied to input i of the *CUT*. The logic block used for producing the sequence on input i has an implementation called a *comparison unit* in [7].

The paper is organized as follows. The *TPG* structure is described in Section 2. The selection of the *TPG* parameters is described in Section 3, including the bounds $\{L_i, U_i\}$ that determine the sequences of 1s on the *CUT* inputs. We describe a basic scheme where the values of the bounds $\{L_i, U_i\}$ are unrestricted, and then introduce a scheme with restricted bound values. The latter *TPG*s have lower hardware overheads. Experi-

0-89791-993-9/97 $10.00

mental results are presented in Section 4, including results of the basic scheme, results of the extended scheme with restricted bound values, and results of the latter scheme in conjunction with test-point insertion. Section 5 concludes the paper.

2. The *TPG* structure

The overall structure of the *TPG* for an n-input circuit is shown in Figure 1. The boxes containing L_i, U_i for $1 \le i \le n$ are comparison units whose structure is described below. A k-bit counter is used to drive the comparison units. At time unit u, the ith comparison unit accepts the k-bit state $Q[u]$ of the counter, and produces the value 1 if and only if $Q[u]$ satisfies $L_i \le Q[u] \le U_i$.

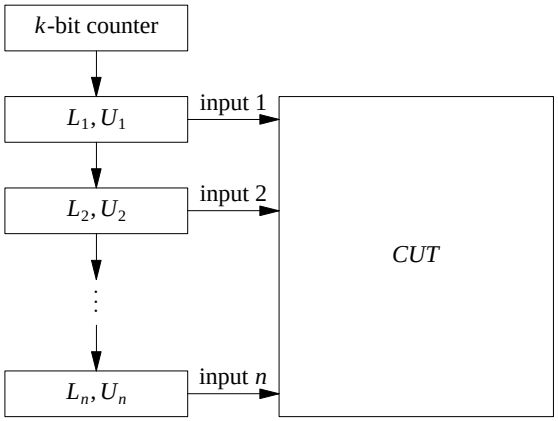

Figure 1: The *TPG* structure

The structure of a comparison unit [7] is shown in Figure 2. The $\ge L$ block produces the value 1 if and only if the input vector represents a binary number whose value is L or more; and the $\le U$ block produces the value 1 if and only if the input vector represents a binary number whose value is U or less. The comparison block inputs are denoted $x_1, x_2, \cdots, x_k$. The counter outputs are denoted $q_1, q_2, \cdots, q_k$ where q_1 is the most significant counter bit and q_k is the least significant counter bit. In Figure 1, we connect x_i to q_i for $1 \le i \le k$. The implementation of a $\ge L$ or $\le U$ block with k inputs requires at most $k-1$ gates [7].

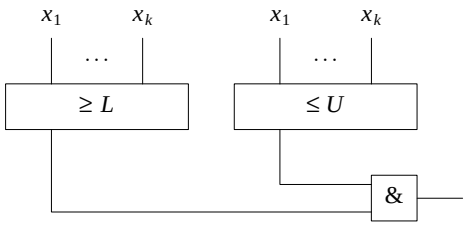

Figure 2: A comparison unit

To minimize the number of comparison blocks needed to implement a given *TPG*, we observe that a $\le U$ block can be implemented using the complemented output of a $\ge L$ block, where $L = U + 1$. This is because the function $\le U$ is the complement of the function $> U$, or $\ge U + 1$. In the implementation of the *TPG*, each bound (original lower bound L or one obtained through complementation of an upper bound U) needs to be implemented only once. For example, if a three input circuit has $L_1 = U_2 + 1 = L_3$, and $U_1 + 1 = U_3 + 1$, then the implementation of Figure 3 can be used. Note that only $\ge L$ blocks are used in this implementation, and their outputs are complemented when they replace $\le U$ blocks.

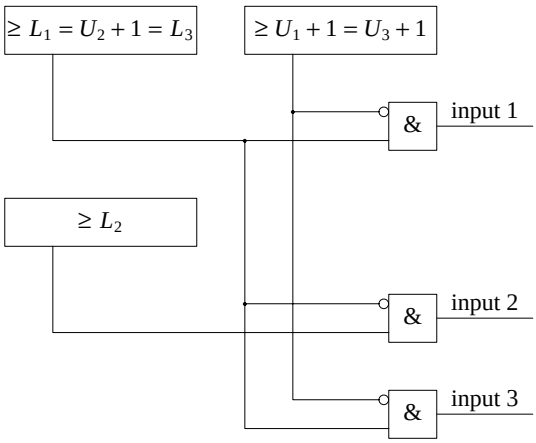

Figure 3: An example of a *TPG*

In some cases, a single *TPG* may not be sufficient to achieve the desired fault coverage for the circuit. In this case, multiple *TPG*s may be used. If more than one *TPG* is needed to achieve the desired fault coverage for a circuit, a multiplexer is required on each input of the *CUT* to select the logic corresponding to the appropriate *TPG*. The control input of the multiplexers can be driven from a counter whose frequency is $R2^k$ lower than the frequency of the counter that drives the comparison units. This will allow each *TPG* to apply a sequence of length $R2^k$ before it is replaced by the next *TPG*. As in the case of a single *TPG*, if multiple *TPG*s use the same bound, then the bound need only be implemented once.

In the discussion above, we ordered the inputs to the comparison units in the same way as the counter outputs. Thus, for a comparison unit with inputs $x_1, x_2, \cdots, x_k$, x_i was connected to the counter output q_i for $1 \le i \le k$. With this configuration, we obtain a single run of 1s on every input of the *CUT* every time the counter goes through a complete cycle. For example, consider a 2-input circuit driven by a 3-bit counter. Suppose that $L_1 = 2$, $U_1 = 4$, $L_2 = 4$ and $U_2 = 5$. Then the sequence shown in Table 1(a) is applied to the *CUT* every time the counter goes through a complete cycle. The 1s on each *CUT* input appear consecutively in this case. It is possible to achieve more varied test sequences by permuting the inputs of the comparison units compared to the counter outputs. For example, let the inputs of the first comparison unit in the example above be x_{11}, x_{12}, x_{13}. Suppose that x_{11} is connected to q_2, x_{12} is connected to q_3, and x_{13} is connected to q_1. Let us use the same bounds as above, $L_1 = 2$ and $U_1 = 4$. The sequence applied to the first *CUT* input is determined as follows.
$(q_1q_2q_3) = (000)$ implies $(x_{11}x_{12}x_{13}) = (000)$. 0 is not within the bounds $L_1 = 2$ and $U_1 = 4$. The *CUT* input is 0.
$(q_1q_2q_3) = (001)$ implies $(x_{11}x_{12}x_{13}) = (010)$. 2 is within the bounds $L_1 = 2$ and $U_1 = 4$. The *CUT* input is 1.
The complete sequence on the first *CUT* input is shown in Table 1(b) under column *CUT input 1*. We also show the values of x_{11}, x_{12}, x_{13} for every counter state. For the second comparison unit with bounds $L_2 = 4$ and $U_2 = 5$, we connect its inputs such that $(x_{21}x_{22}x_{23}) = (q_3q_1q_2)$. The complete sequence on the second *CUT* input is also shown in Table 1(b).

By permuting the inputs of the comparison units as illustrated above, we add another degree of freedom in determining

Table 1: An example of input permutations
(a) Unpermuted inputs

q_1	q_2	q_3	CUT input 1	CUT input 2
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

(b) Permuted inputs

q_1	q_2	q_3	x_{11} q_2	x_{12} q_3	x_{13} q_1	CUT input 1	x_{21} q_3	x_{22} q_1	x_{23} q_2	CUT input 2
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1	0	0	1
0	1	0	1	0	0	1	0	0	1	0
0	1	1	1	1	0	0	1	0	1	1
1	0	0	0	0	1	0	0	1	0	0
1	0	1	0	1	1	1	1	1	0	0
1	1	0	1	0	1	0	0	1	1	0
1	1	1	1	1	1	0	1	1	1	0

the *TPG* output sequences (or the *CUT* test sequences), and a higher fault coverage may potentially be obtained by the resulting test sequence.

3. Selecting the *TPG* parameters

Our approach to determining the *TPG* parameters is based on simulation of randomly selected *TPG*s. We first describe the basic *TPG* selection procedure, and then extend it to reduce the total number of bounds required.

3.1 Basic procedure

The procedure used for selecting a single *TPG* is given below as Procedure 1. The procedure is described for a given counter length k and a given number of repetitions of the counter sequence R_1. The procedure selects the *TPG* out of a set of N_{try} randomly determined *TPG*s, as follows. The parameters L_i and U_i $(1 \le i \le n)$ for each *TPG* and its input permutation are randomly selected in Step 2 of Procedure 1. The output sequence T produced by each *TPG* is computed in Step 3, and the circuit is simulated under T in Step 4. The length of the sequence T is $2^k R_1$. The number of faults detected by the j-th *TPG* is recorded in variable N_{det}^j. This process is repeated for N_{try} *TPG*s (cf. Steps 1-5 of Procedure 1). The *TPG* that results in the highest fault coverage is selected in Step 6. This *TPG* is allowed to generate a sequence of length $2^k R_2$ $(R_2 > R_1)$ in Steps 7 and 8.

Procedure 1: Selecting *TPG* parameters

(1) Set $j = 1$.

(2) For every *CUT* input i:
  (a) Select a pair of numbers (L_i^j, U_i^j) such that $0 \le L_i^j \le 2^k - 1$, $0 \le U_i^j \le 2^k - 1$ and $L_i^j \le U_i^j$.
  (b) Select a permutation of the counter outputs $< q_{i1}^j, q_{i2}^j, \cdots, q_{ik}^j >$.

(3) Generate the test sequence T produced during R_1 cycles of the counter by the *TPG* where (a) *CUT* input i is characterized by bounds L_i^j and U_i^j for $1 \le i \le n$, and (b) input x_{im}^j of the i-th comparison unit is connected to output q_{im}^j of the counter for $1 \le m \le k$ and $1 \le i \le n$.

(4) Fault simulate the circuit under the sequence T produced in Step 3. Let N_{det}^j faults be detected.

(5) Set $j = j + 1$. If $j \le N_{try}$, where N_{try} is a preselected number, go to Step 2.

(6) Select the *TPG*, j_0, for which $N_{det}^{j_0}$ is maximum.

(7) Generate the test sequence T produced by *TPG* j_0 during R_2 cycles.

(8) Fault simulate the circuit under the test sequence T produced in Step 7.

(9) Stop: The faults detected are the ones that will be detected by the selected *TPG*.

To increaes the fault coverage obtained by the *TPG* selected by Procedure 1, we describe next two extensions.

Let a *TPG* selected by Procedure 1 produce a test sequence T. Simple modifications of the *TPG* allow modified versions of T to be applied to the *CUT*. These modified versions of T may detect faults that are not detected by T, thus increasing the fault coverage. We consider the following modifications of T and their combination. First, by adding multiplexers and inverters on the *TPG* outputs (or the *CUT* inputs) and allowing complemented values to be applied to the *CUT*, the sequence T' which is the complement of T can be applied to the *CUT*. Second, by using an up/down counter instead of a counter that can only count up, the sequence T_r which is the reverse of T can be applied to the *CUT*. In addition, by using both the down count and the complemented *TPG* outputs, it is possible to apply to the *CUT* the sequence T_r' which is the complemented sequence T' applied in reverse order. An example of the four versions of T for a three-input circuit is shown in Table 2. Procedure 1E given below is the same as Procedure 1 except that the modified sequences described above are used as well.

Table 2: An example of a modified sequence

T			T'			T_r			T_r'		
0	0	0	1	1	1	1	0	1	0	1	0
0	1	1	1	0	0	1	0	0	0	1	1
1	0	0	0	1	1	0	1	1	1	0	0
1	0	1	0	1	0	0	0	0	1	1	1

Procedure 1E: Selecting *TPG* parameters (extended)

(1) Set $j = 1$.

(2) For every *CUT* input i:
  (a) Select a pair of numbers (L_i^j, U_i^j) such that $0 \le L_i^j \le 2^k - 1$, $0 \le U_i^j \le 2^k - 1$ and $L_i^j \le U_i^j$.
  (b) Select a permutation of the counter outputs $< q_{i1}^j, q_{i2}^j, \cdots, q_{ik}^j >$.

(3) Generate the test sequence T produced during R_1 cycles of the counter by the *TPG* where *CUT* input i is characterized by bounds L_i^j and U_i^j for $1 \le i \le n$, and input x_{im}^j of the ith comparison unit is connected to output q_{im}^j of the counter for $1 \le m \le k$ and $1 \le i \le n$.

(4) Fault simulate the circuit under the test sequence T produced in Step 3 (fault simulation with fault dropping starts from the complete set of target faults). Let $N_{det,1}^j$ faults be detected.

(5) Complement T and fault simulate the *CUT* under the complemented sequence T' (fault simulation with fault dropping is used starting from the set of faults left undetected in Step 4). Let $N_{det,2}^j$ faults be detected.

(6) Reverse the order of the vectors in T' and fault simulate the *CUT* under the reverse test sequence T'_r. Let the number of detected faults be $N^j_{det,3}$.

(7) Complement T'_r and fault simulate the *CUT* under the resulting sequence T_r. Let the number of detected faults be $N^j_{det,4}$.

(8) Let $N^j_{det} = N^j_{det,1} + N^j_{det,2} + N^j_{det,3} + N^j_{det,4}$.

(9) Set $j = j + 1$. If $j \leq N_{try}$, where N_{try} is a preselected number, go to Step 2.

(10) Select the *TPG*, j_0, for which $N^{j_0}_{det}$ is maximum.

(11) Generate the sequence T produced by *TPG* j_0 during R_2 cycles. Fault simulate the circuit under T (start from the complete set of target faults).

(12) Complement the test sequence T and fault simulate the circuit under the complemented test sequence T'.

(13) Reverse the test sequence T' and fault simulate the circuit under the reverse test sequence T'_r.

(14) Complement the test sequence T'_r and fault simulate the circuit under the reverse test sequence T_r.

(15) Stop: The faults detected in Steps 11-14 are the ones that will be detected by the selected *TPG*.

If the *TPG* selected by Procedure 1 or Procedure 1E is not sufficient to detect all the faults in the *CUT*, we repeat Procedure 1 or Procedure 1E with the faults left undetected to select additional *TPG*s, until the accumulated fault coverage reaches the desired level or no improvement in fault coverage is possible. This procedure is summarized as Procedure 2 next. Procedure 2E is similar to Procedure 2, except that Procedure 1E is used in Step 2 instead of Procedure 1.

Procedure 2 (2E): Selecting multiple *TPG*s

(1) Let F be the set of target faults.

(2) Apply Procedure 1 (1E) to select a *TPG* that detects the maximum number of faults out of F. Let F_{det} be the set of faults detected by the selected *TPG*.

(3) If $F_{det} = \phi$, stop: No additional *TPG*s are selected.

(4) Set $F = F - F_{det}$. If $F = \phi$, stop: All the target faults are detected by the selected *TPG*s.

(5) Go to Step 2.

In our implementation, Procedure 2 (or Procedure 2E) is applied with counter lengths $k = 10, 11, 12, 13$, and the best result is selected. Theoretically, any *TPG* that can be produced using a counter of length k_1 can also be produced using a counter of length $k_2 > k_1$ by ignoring the most significant bits of the counter or by permuting them such that they drive the least significant bits of the comparison units and selecting appropriate bounds. However, in practice, Procedures 1 and 1E explore only a limited number of *TPG*s, and therefore, may not yield increasingly better *TPG*s as k is increased.

The area of the *TPG* selected for a circuit depends on the number of *different* bounds used. Two bounds are said to be the same if they are computed using the same input permutation, and their values are the same. For a circuit with n inputs, m *TPG*s and a counter of length k, the number of different bounds is not larger than $\min\{2mn, k!2^k\}$. The term $2mn$ results from the fact that each circuit input needs two bounds for each *TPG*. The term $k!2^k$ results from the fact that there are 2^k different bound values for a counter of length k, and there are $k!$ permutations of the comparison unit inputs, each resulting in a different set of bound values. Each bound is implemented using at most $k-1$ 2-input gates. In addition, at most one 2-input gate is required to *AND* the two bounds of each comparison unit. The total number of 2-input gates is thus $\leq (k-1) \cdot \min\{2mn, k!2^k\} + n$. In most practical cases, $2mn < k!2^k$, and the number of 2-input gates is at most $(2(k-1)m+1)n$. This area can be reduced by incorporating techniques such as test-point insertion to allow a smaller number of *TPG*s to be used. They may also make the approach described in the following section more effective.

3.2 *TPG*s with limited sets of bounds

Our goal in this subsection is to replace the term $k!2^k$ in the expression determining the worst-case number of bounds by a smaller value. The new value is a constant, typically smaller than the term $2mn$, thus potentially reducing the number of different bounds. We achieve this goal by imposing two restrictions on the bound values. First, we limit the number of permutations allowed for the comparison unit inputs. In our experiments, we use only the permutation of the comparison unit inputs where $x_i = q_i$ for $1 \leq i \leq k$. Thus, the term $k!$ is removed. In addition, we only allow bounds taken out of a preselected set B containing a preselected number of bounds N_B. For a given number of bounds N_B, we define $L_r = 2^k/N_B$. We allow only bounds that are multiples of L_r, i.e., $B = \{0, L_r, 2L_r, 3L_r, \cdots, (N_B-1)L_r\}$. For example, for $k = 8$ and $N_B = 4$, we obtain $L_r = 256/4 = 64$ and $B = \{0, 64, 128, 192\}$. In Step 2 of Procedure 1 (or Procedure 1E), bounds are randomly selected out of the set B.

As a result of the restrictions above, the number of different bounds required to implement a set of m *TPG*s for an n-input circuit using a counter of length k is bounded by $\min\{2mn, N_B\}$. In our experiments we used $N_B = 4$, ensuring that at most four bounds would have to be implemented (in fact, a bound of 0 does not need to be implemented, since every counter state is larger than or equal to 0; therefore, the number of bounds is at most $\min\{2mn, N_B - 1\}$). With $N_B = 4$, the number of 2-input gates required to realize the *TPG* is $\leq 3(k-1) + n$.

4. Experimental results

In this section, we describe the results obtained by applying Procedure 2 and Procedure 2E to ISCAS-89 benchmark circuits. For comparison purposes, we include in Table 3 the fault coverage reported in [5] using the hold method proposed there. We also report in Table 3 the highest fault coverage reported for several deterministic test generation procedures. Finally, we include in the last column of Table 3 the fault coverage obtained by applying 100,000 random patterns.

Table 3: Comparison

circuit	hold	determ	rand
s208	NA	63.72	43.72
s298	NA	86.04	84.74
s344	NA	96.20	96.20
s382	86.0	90.73	13.28
s386	NA	81.77	71.35
s420	NA	41.63	33.72
s444	80.4	87.76	12.66
s526	NA	80.00	9.37
s641	NA	86.51	86.51
s820	57.8	95.76	49.29
s1196	NA	99.76	98.79
s1423	86.3	90.89	62.97
s1488	NA	97.17	71.27
s5378	74.4	78.47	66.54

The following parameters were used in the implementation of Procedures 2 and 2E. The numbers of repetitions of the counter sequence are $R_1 = 1$ and $R_2 = 2$. The counter lengths considered are $k =$ 10, 11, 12 and 13. We report only the shortest counter length that resulted in the best fault coverage using the smallest number of *TPG*s. The number of *TPG*s tried in Procedure 1 and 1E is $N_{try} = 5$.

When a *TPG* is allowed to produce $R2^k$ test patterns which are then simulated on the *CUT*, we capture the last pattern u_{eff} effective in detecting any new fault. We refer to the length of the test sequence up to u_{eff} as the *effective test length*. If multiple *TPG*s are used, the total effective test length is the sum of the effective test lengths for all the *TPG*s used.

In the following subsections we describe the results obtained with unlimited bound values, and when the bound values are restricted. We also present results of the latter scheme in conjunction with test-point insertion.

4.1 Unlimited bounds

The results obtained for several benchmark circuits by applying Procedure 2 using the parameters above are given in Table 4. After circuit name we show the counter length k, the number of *TPG*s used, the total effective length for all the *TPG*s, and the fault coverage achieved. Following the fault coverage we show how the fault coverage compares to the maximum fault coverage reported for deterministic test generation, and to the fault coverage reported in [5]. We enter <, > or = to indicate that the fault coverage achieved by the proposed method is smaller than, larger than or equal to the fault coverage it is compared to. We also enter a ? when the corresponding fault coverage is not known. It can be seen that the fault coverage is higher than that reported in [5] in all the cases considered; however, the fault coverage is lower in many cases than that achievable by deterministic test generation. Increasing the values of R_1, R_2 and N_{try}, or allowing Procedure 2 to go through additional iterations, did not have significant effects on the results of Table 4.

Table 4: Results of Procedure 2

circuit	k	e.len	*TPG*s	f.c	targ	[5]
s208	11	2428	4	63.26	<	?
s298	11	721	2	86.04	=	?
s344	12	3908	4	96.20	=	?
s382	10	3066	5	88.97	<	>
s386	12	10714	4	79.95	<	?
s420	13	4264	2	41.16	<	?
s444	11	5208	3	87.76	=	>
s526	13	19819	3	78.92	<	?
s641	11	8739	6	86.51	=	?

The results of Procedure 2E for the same benchmark circuits and for additional ones are shown in Table 5. The same parameters used to obtain Table 4 are used in this case. The effective length for Procedure 2E is computed as follows. For each *TPG*, we compute the maximum effective length of its output sequence T, its complement T', its reverse T_r and its complemented reverse T'_r. We then add up all the maximum effective lengths. It can be seen by comparing Tables 4 and 5 that Procedure 2E achieves higher fault coverages than Procedure 2 using fewer *TPG*s. Procedure 2E reaches the target fault coverage set by deterministic test sequences more often than Procedure 2.

The number of different bounds obtained for the *TPG*s in Tables 4 and 5 is typically equal to the worst-case of $\min\{2mn, k!2^k\}$. In the following subsection we report the results obtained by restricting the possible bounds to a subset B of size N_B, as described in Section 3.2

Table 5: Results of Procedure 2E

circuit	k	e.len	*TPG*s	f.c	targ	[5]
s208	11	2599	3	63.26	<	?
s298	10	899	1	86.04	=	?
s344	12	3407	1	96.20	=	?
s382	12	4694	2	89.47	<	>
s386	12	4840	2	81.77	=	?
s420	12	15670	5	41.63	=	?
s444	12	4603	2	87.76	=	>
s526	13	19864	4	79.28	<	?
s641	13	6352	1	86.51	=	?
s820	13	46973	8	82.82	<	>
s1196	13	43574	8	99.11	<	?
s1423	11	17629	8	90.89	=	>
s1488	12	12194	3	97.17	=	?
s5378	12	39141	12	75.34	<	>

4.2 Restricted bounds

In the experiments reported in this section we use Procedure 2E with the value of k used in Table 5. We restrict the number of different bounds to $N_B = 4$. The results are reported in Table 6. The results of Table 5 are repeated in column $N_B = unl$ for ease of comparison. It can be seen that the fault coverage for restricted bounds is lower than the fault coverage when the bounds are unrestricted. However, even this lower fault coverage is higher than that of [5] for most of the circuits where a comparison is possible. In addition, the loss of fault coverage may be justified by the reduced number of bounds and thus the reduced hardware complexity of the *TPG*s. Techniques such as test point insertion may be used to compensate for this effect.

Table 6: Results using restricted bounds

	N_B=4		$N_B = unl$	
circuit	TPGs	f.c	TPGs	f.c
s208/11	3	61.40	3	63.26
s298/10	4	85.06	1	86.04
s344/12	4	92.98	1	96.20
s382/12	2	87.72	2	89.47
s386/12	8	73.44	2	81.77
s420/12	3	40.23	5	41.63
s444/12	1	86.08	2	87.76
s526/13	1	78.38	4	79.28
s641/13	10	83.51	1	86.51
s1423/11	19	77.49	8	90.89

4.3 Restricted bounds with test-point insertion

In this subsection, we consider the *TPG*s selected in Table 6 for $N_B = 4$. These *TPG*s have the lowest area overhead of all the *TPG*s considered in the previous subsections; however, they usually result in the lowest fault coverage. To increase the fault coverage, we consider two schemes for test point insertion as described next.

We describe the test point insertion schemes for a *TPG* made of m *TPG*s, $TPG_1, TPG_2, \cdots, TPG_m$, designed for a circuit with a set of faults F. In both schemes, we simulate the sequences produced by the *TPG*s in the order $TPG_1, TPG_2, \cdots, TPG_m$, and drop all the faults detected by each *TPG* before the next one is considered. In the first scheme, test-points are inserted after every *TPG* is considered. As a result, when TPG_{j_1} is considered, test-points may be inserted to detect faults that are also detected by TPG_{j_2} where $j_2 > j_1$. In the sec-

ond scheme, we first drop all the faults detected by all the *TPG*s, and then reconsider each *TPG* for test-point insertion. In the first scheme, it may be possible to omit a *TPG*, say TPG_j, if all the faults it detects are already detected by $TPG_1, \cdots, TPG_{j-1}$, simulated earlier. In the second scheme, all the *TPG*s will remain necessary. However, the first scheme may require more test-points, since test-points are inserted even for faults that may be detected by a *TPG* simulated later. The two schemes are described in more detail next. For simplicity of presentation, we refer to the sequences T, T', T_r and T'_r produced by a *TPG* as the *TPG* output sequence $\tilde{T}$.

The first test-point insertion scheme proceeds as follows. For TPG_1, we generate the *TPG* output sequence $\tilde{T}_1$, and simulate the set of target faults F under $\tilde{T}_1$. Let $\tilde{T}_1$ detect a subset of faults F_1. We simulate every fault $f \in F - F_1$ under the test sequence $\tilde{T}_1$ again, this time capturing every line v to which 0/1 or 1/0 values are propagated in the presence of f. Each such line, if selected as a test-point, will ensure that f can be detected by $\tilde{T}_1$. We use a covering procedure to select a minimal number of test-points to cover all the faults in $F - F_1$ that are detectable through test-points. Let the set of faults detected through test-points be F'_1. We drop from F all the faults in $F_1 \bigcup F'_1$. Next, we consider TPG_2. We generate the *TPG* output sequence $\tilde{T}_2$, and simulate F under $\tilde{T}_2$. Let $\tilde{T}_2$ detect a subset of faults F_2. We simulate every fault $f \in F - F_2$, and capture potential test-point locations for the fault. We then check whether the fault is detected by any one of the test-points already selected for TPG_1. For the remaining faults in $F - F_2$, we select additional test-point locations using a covering procedure. We denote the set of faults detected by TPG_2 through test-points by F'_2. We drop from F all the faults in $F_2 \bigcup F'_2$. The same process is repeated for all the *TPG*s. If $F_j \bigcup F'_j = \phi$ for *TPG* j, then *TPG* j can be omitted without reducing the fault coverage.

The second test-point insertion scheme proceeds as follows. We first generate the *TPG* sequence $\tilde{T}_j$ for every TPG_j, and drop the faults detected from F. We then reconsider each *TPG* separately. When TPG_j is considered, we simulate every fault $f \in F$ under the test sequence $\tilde{T}_j$ again, this time capturing every line v to which 0/1 or 1/0 values are propagated in the presence of f. Every fault detectable through an already selected test-point is dropped from F. We then use a covering procedure to select a minimal number of test-points to cover all the faults in F that are detectable through test-points when $\tilde{T}_j$ is applied. The same process is repeated for all the *TPG*s.

The results of test-point insertion using the *TPG*s produced for $N_B = 4$ in Table 6 are shown in Table 7. For each scheme, we show the number of *TPG*s that were useful for detecting any new faults (for Scheme 2, this number is equal to the number of *TPG*s in Table 6). We then show the total number of test-points inserted, and the final fault coverage obtained. Under column *targ* we mark whether the fault coverage is lower than or equal to the fault coverage achievable by deterministic test generation. For this comparison to be meaningful, we only consider faults that can be detected by deterministic test generation without test-points. It can be seen that Scheme 1 places more test-points, but requires fewer *TPG*s. Scheme 2 places fewer test-points but uses all the *TPG*s of Table 6. Compared to Table 6, the fault coverage is always increased. The fault coverage is not always equal to the target fault coverage achievable by deterministic test generation procedures. This is due to the fact that some detectable faults are never activated by the selected *TPG*s. The fault coverage can be increased by performing test-point insertion together with *TPG* selection. We did not implement this option. We point out that the area overhead for comparison units in this case includes the implementation of at most three bounds, requiring at most $(3(k-1)+n)$ 2-input gates.

Table 7: Results using restricted bounds and test-points

	scheme 1				scheme 2			
circuit	TPGs	t.p	f.c	targ	TPGs	t.p	f.c	targ
s208/11	2	10	63.72	=	3	5	63.72	=
s298/10	2	9	86.04	=	4	2	86.04	=
s344/12	4	21	95.61	<	4	8	95.61	<
s382/12	1	8	89.47	<	2	7	89.47	<
s386/12	4	65	81.51	<	8	22	81.51	<
s420/12	2	12	41.63	=	3	5	41.63	=
s444/12	1	8	87.76	=	1	8	87.76	=
s526/13	1	9	80.00	=	1	8	87.76	=
s641/13	4	64	86.30	<	10	13	86.30	<

5. Concluding remarks

We proposed a structure for a built-in test pattern generator for synchronous sequential circuits. The motivation for considering built-in test of synchronous sequential circuits is that the circuit flip-flops may be left unmodified, and test application may be done at-speed. The proposed test pattern generator had a uniform, parametrized structure. By matching the parameters of the test pattern generator to the circuit-under-test, high fault coverage was achieved for benchmark circuits. In many cases, the fault coverage was equal to the fault coverage that can be achieved by deterministic test sequences. Reduced area overheads were achieved by restricting the parameter values and inserting test-points to compensate for the lower fault coverage that may result.

Several extensions of the proposed approach may be useful in reducing the hardware overhead and increasing the fault coverage. More careful selection of *TPG* parameters may yield improved results at lower hardware overheads. Incorporation of test-points during the *TPG* selection process may allow fewer *TPG*s and fewer bounds to be used.

References

[1] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A Tutorial on Built-In Self-Test Part 1: Principles", IEEE Design and Test of Computers, March 1993, pp. 73-82.

[2] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A Tutorial on Built-In Self-Test Part 2: Applications", IEEE Design and Test of Computers, June 1993, pp. 69-77.

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

[4] 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.

[5] 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.

[6] M. Abramovici, K. Rajan and D. Miller, "FREEZE: A New Approach for Testing Sequential Circuits", in Proc. 29th Design Autom. Conf., June 1992, pp. 22-25.

[7] I. Pomeranz and S. M. Reddy, "On Synthesis-for-Testability of Combinational Logic Circuits", in Proc. 32nd Design Automation Conf., June 1995, pp. 126-132.