Indian J.Sci.Res. 17(2): 550 - 555, 2018
ISSN: 0976-2876 (Print)
ISSN: 2250-0138(Online)

COMPUTING SEEDS FOR LFSR-BASED TEST GENERATION FROM NONTEST CUBES

Vigneshwar Mohan
Department of Communication and Networking, Trichy Engineering College, Trichy

Abstract- In test data compression methods that are based on the use of a linear-feedback shift register, a seed that produces a test for a target fault is computed based on a test cube for the fault. With a given LFSR, a seed may not exist for a given test cube, even though a seed may exist for a different test cube that detects the same fault. This issue is addressed in this brief by computing seeds for LFSR-based test generation without using test cubes. Instead, the procedure described in this brief is based on the use of nontest cubes. A nontest cube for a fault must be avoided in any test or test cube for the fault in order to allow the fault to be detected. Therefore, nontest cubes do not limit the ability of the procedure to compute seeds with a given LFSR. Experimental results demonstrate the advantages that the use of nontest cubes provides, and the associated computational cost.

Keywords—Linear-feedback shift register (LFSR)-based test generation, nontest cubes, scan circuits, test cubes, test data compression.

I. Introduction

When test data compression is based on the use of a linear-feedback shift register (LFSR), test cubes are used for com- puting seeds for the LFSR [1]–[10]. Given a test cube ci for a target fault fi, a seed si for the LFSR is obtained by solving a set of linear equations that relate si with the specified values of ci [1]. When si is loaded into the LFSR, and the LFSR is clocked for the appropriate number of clock cycles, the scan chains of the circuit are filled with a test ti. The test ti contains all the specified values of ci. Therefore, ti is guaranteed to detect fi.

When an LFSR is used with a given set of test cubes, a seed may not exist for one or more of the test cubes [2], [9]. However, even if a seed does not exist for a test cube ci0 that detects a fault fi, it is possible that a seed exists for a different test cube ci1 for fi.

To address this issue it is possible to compute different test cubes to replace ones for which seeds do not exist. Alternatively, a procedure developed earlier uses a test cube ci for a fault fi only as guidance for the computation of a seed si. The procedure allows the seed si to produce a test ti that conflicts with ci as long as ti detects fi. However, this procedure still relies on the use of specific test cubes. Therefore, even with a partial match, it may not be able to find a seed si for a fault fi based on a test cube ci.

The procedure described in [11] adds to the circuit an XOR network that models the constraints of the test data decompression logic. By performing test generation for the extended circuit, the procedure from [11] finds seeds for an LFSR directly, without first computing test cubes.

The goal of this brief is to show that it is possible to compute seeds for LFSR-based test generation without using test cubes and without extending the circuit. This alleviates the constraints that the use of test cubes places on the ability to detect target faults without the need to perform test generation for a more complex circuit. Instead of test cubes, the procedure described in this brief uses what are called nontest cubes [12]. A nontest cube ui for a fault fi prevents fi from being detected. In order to detect the fault, it is necessary to prevent ui from appearing in a test. This applies to every test and test cube for the fault. Therefore, the use of nontest cubes for computing seeds does not limit the ability of the procedure to find seeds when they exist for a given LFSR.

The procedure for computing seeds based on nontest cubes uses a low-complexity procedure that is based on logic simulation of the LFSR to compute the test ti that a given seed si produces. Fault simulation of the fault fi under ti is used for determining whether ti detects fi. To compute a seed si for a given target fault fi, the procedure uses a set of nontest cubes Ui for fi. It starts from a random assignment to si. It modifies si so as to avoid the appearance of nontest cubes from Ui in ti. The modification of si is expected to lead to the detection of fi when a seed for fi exists.

The advantage of this procedure is that it is not constrained by a given test cube. Therefore, a seed for a given LFSR may be found even if one cannot be found based on a test cube. Its disadvantage is that the search for a seed can be more time-consuming, since it is guided only by values that need to be avoided. To address this issue, it is possible to use nontest cubes only for faults that cannot be detected based on test cubes. Experimental results presented in this brief demonstrate this point. Since only hard-to-detect faults are targeted, test compaction with nontest cubes is not considered.

Considering the computation of nontest cubes, a partial set of nontest cubes for a fault fi can be obtained in a preprocessing step. The set can be extended during the computation of a seed si for fi.

[1]Corresponding Author

In particular, every test ti that the seed produces and does not detect fi can be used for computing a nontest cube for fi [12]. In this brief, only nontest cubes with single specified values are used, and they are computed in a preprocessing step. This is based on the exper- imental observations that nontest cubes with single specified values are the most effective in guiding the generation of a seed. In addition, hard-to-detect faults in benchmark circuits have such nontest cubes. Single stuck-at faults are used as target faults. A single stuck-at fault where line gi is stuck at the value ai is denoted by fi = gi /ai. The procedure can be extended to other fault models. For example, to consider transition faults, two-cycle nontest cubes can be used. This brief is organized as follows. Section II describes the computation of nontest cubes. Section III describes the use of nontest cubes for the computation of a seed for a target fault. Section IV describes the generation of seeds for a given set of target faults. Section V presents the experimental results.

II. Computation Of Non Test Cubes

A set of nontest cubes Ui for a target fault fi = gi /ai is computed in a preprocessing step as described in this section.

A nontest cube for fi prevents fi from being activated and/or propagated to an output. Therefore, the nontest cube must be avoided by every test and test cube for fi.

Table I

Test Cubes

j	b	$2j+b$	u_{2j+b}
0	0	0	0xxxx
0	1	1	1xxxx
1	0	2	x0xxx
1	1	3	x1xxx
2	0	4	xx0xx
2	1	5	xx1xx
3	0	6	xxx0x
3	1	7	xxx1x
4	0	8	xxxx0
4	1	9	xxxx1

Only nontest cubes with single specified values are considered.

For a circuit whose combinational logic has n inputs (primary inputs and present-state variables), the test cube u2 j +b , where 0 ≤ j < n and b ∈ {0, 1}, assigns the value b to inputj , and undefined values to the remaining inputs. The test cube u2 j +b is represented as u2 j +b (0) u2 j +b (1) ··· u2 j +b (n − 1), where u2 j +b (k) is the value of input k under u2 j +b. We have that u2 j +b (j) = b and u2 j +b (k) = x fork=j. For illustration, the test cubes with single specified values for a circuit with n = 5 inputs are shown in Table I. In general, for a circuit with n inputs, the number of test cubes with single specified values is 2n.

The number of nontest cubes that will be obtained for a fault is bounded by 2n.

The procedure described in this section determines the set of nontest cubes Ui for a fault fi = gi /ai as follows.

The procedure traces the circuit forward from gi in order to find all the outputs to which fi can potentially be propagated. It then traces the circuit backward from these outputs to find all the inputs that can potentially affect the detection of fi. This set of inputs is referred to as the input cone of fi, and it is denoted by J (fi).

For an input j ∉ J (fi), assigning a value cannot prevent fi from being detected. Therefore, u2 j and u2 j +1 are excluded from Ui without further computations. The procedure evaluates the test cube u2 j +b for every j ∈ J (fi) and b ∈ {0, 1}, as follows.

To evaluate u2 j +b, the procedure first initializes all the circuit lines to unspecified values. It then implies the value b on input j. This yields the values in the fault-free circuit under the test cube u2 j +b. If the fault-free value of line gi is equal to ai, the test cube u2 j +b prevent fi from being activated. The procedure adds u2 j +b to Ui as a nontest cube for fi, and it does not consider u2 j +b further. Otherwise, the procedure computes the values obtained under u2 j +b in the faulty circuit by implying the value ai on line gi. The fault fi can potentially be activated and propagated to an output if it is possible to find a path from gi to an output such that all the lines along the path carry fault-free/faulty values from the set {0/1, 0/x , 1/0, 1/x , x /0, x /1, x /x }.

Such a path is referred to as a propagation path. A propagation path can be found in time that is linear in the number of circuit lines.If no propagation path exists for fi , the test cube u2 j +b prevents fi from being detected. In this case, the procedure adds u2 j +b to Ui as a nontest cube for fi.

To compute Ui, the procedure considers at most 2n test cubes. For every test cube that it considers, it performs logic simulation of the fault-free circuit. In addition, it may perform logic simulation of the faulty circuit, and a traversal of the circuit to find a propagation path. For a circuit with G lines, this requires O (n · G) operations.

III. Computation Of A Seed Based On A Set Of Non Test Cubes

Let Ui be a set of nontest cubes for a fault fi. The procedure described in this section uses Ui as it attempts to compute a seed si such that the test ti it produces detects fi.

The procedure initializes si randomly, and computes the test ti that si produces. A nontest cube u2 j +b ∈ Ui indicates that a test ti for fi must have ti (j) = b, where ti (j) is the value of input j under ti. The test ti that si produces is said to avoid a nontest cube u2 j +b ∈ Ui if ti (j) = b. The number of nontest cubes from Ui that ti avoids is denoted by na.

If na < |Ui |, at least one of the nontest cubes in Ui prevents ti from detecting fi. If na = |Ui |, ti avoids all the nontest cubes in Ui, and ti may detect fi. Detection is not guaranteed, since the fault may have other nontest cubes that are not included in Ui . To check whether ti detects fi, the procedure simulates fi under ti. If the fault is detected, the procedure returns si as the required seed. This may occur accidentally for the initial random seed. Otherwise, the procedure modifies si by complementing its bits one at a time in an attempt to detect the fault. The modification is guided by Ui as follows.

Using the random initialization of si , the procedure assigns na,best = na . The procedure considers the bits of si one at a time in a random order. When bit k is considered, the procedure complements the bit by assigning si (k) = si (k).

It then computes ti and na. If na ≥ na, best, the procedure accepts the complementation of bit k, and assigns na, best = na. Otherwise (na < na, best), it complements si (k) again in order to undo the complementation.

If bit k is complemented and na = |Ui |, the procedure simulates fi under ti. If the fault is detected, the procedure returns si as the required seed.

The procedure considers all the bits of si repeatedly NMOD times, where NMOD is a parameter of the procedure. As na, best is increased, the procedure avoids more of the nontest cubes of fi. After na, best reaches |Ui |, ti may detect fi. As long as ti does not detect fi, the procedure continues to modify si while ensuring that na = |Ui | for every bit that it accepts to complement. This increases the likelihood that fi will be detected.

This process is different from a random search in that it avoids the nontest cubes from Ui, thus increasing the likelihood of detecting fi. As shown in [12], avoiding nontest cubes is sufficient for detecting hard-to-detect faults in benchmark circuits. The procedure for generating a seed si for a fault fi is provided in Procedure 1.

The worst case computational complexity of Procedure 1 is deter-mined by its fault simulation effort in the case where it does not find a seed. In this case, it attempts to complement every bit of the seed NMOD times. For an LFSR with B bits, the number of attempts that the procedure makes is NMOD · B. For every attempt, it computes the test ti, and simulates fi under ti if na = |Ui |. Thus, in the worstcase, the procedure simulates fi under NMOD · B tests.

Procedure 1 Generating a Seed s_i for a Fault f_i

1) Initialize s_i randomly.
2) Find the test t_i that s_i produces.
3) Compute n_a and assign $n_{a,best} = n_a$.
4) If $n_a = |U_i|$, simulate f_i under t_i. If the fault is detected, return s_i.
5) For $n_{mod} = 0, 1, \ldots, N_{MOD} - 1$:
   a) For every bit $s_i(k)$ of s_i:
      i) Complement $s_i(k)$.
      ii) Find the test t_i that s_i produces.
      iii) Compute n_a. If $n_a \geq n_{a,best}$, assign $n_{a,best} = n_a$. Else, complement $s_i(k)$ again.
      iv) If $n_a = |U_i|$, simulate f_i under t_i. If the fault is detected, return s_i.
6) Return an indication that the fault is not detected.

IV. Computation of Seeds for a Set of Target Faults

Given a set of detectable target faults F, the procedure described in this section is applied to compute a set of seeds for F. The set of seeds is denoted by SNTC (for nontest cubes)

The procedure considers the faults from F one at a time iteratively. Because of the random decisions made by Procedure 1, including the random selection of an initial seed, and because nontest cubes do not provide complete information about the values that are needed for detecting a fault, it is possible that a fault will be detected only after several iterations.

In iteration I ≥ 1, the procedure considers every fault fi ∈ F. For fi, it computes the set of nontest cubes Ui. It then calls procedure 1 to compute a seed. If a seed si is found, the proce-dure computes the test ti that the seed produces. It performs fault simulation with fault dropping of F under ti. It then adds si to SNTC.

The procedure terminates if all the faults in F are detected.In addition, the procedure has a termination condition based on its run time. This is given by the parameter RT.

The procedure is summarized as procedure 2.The set of nontest cubes Ui for a fault fi is recomputed every time the procedure considers fi .Alternatively, the set can be computed once and stored for future use if fi is not detected.

Although the run time of Procedure 2 is bounded by RT, it is interesting to consider the worst case computational complexity of an iteration of the procedure. This is determined by its fault simulation effort in the case where it does not detect any fault. In this case, the procedure calls Procedure 1 with every fault from F, for a total of |F | calls. Procedure 1 simulates a fault under at most NMOD · B tests. Overall, in an iteration, Procedure 2 simulates a fault under at most NMOD · B · |F | tests.

Procedure 2 Generating a Set of Seeds S_{NTC}

1) Assign $S_{NTC} = \emptyset$.
2) For $I = 1, 2, \ldots$, as long as $F \neq \emptyset$:
   a) For every fault $f_i \in F$:
      i) Compute the set of nontest cubes U_i.
      ii) Call Procedure 1 to generate a seed. If a seed s_i is generated:
         A) Find the test t_i that s_i produces. Perform fault simulation with fault dropping of F under t_i.
         B) Add s_i to S_{NTC}.
      iii) If the run time reached RT, go to Step 3.

V. Experimental Results

The main advantage of Procedure 2 is that it is not restricted by a given set of test cubes. The goal of the experiment described in this section is to show that this flexibility allows it to detect faults that are not detected by a procedure that uses test cubes. To achieve this goal, Procedure 2 is applied to the hard-to-detect faults that remain undetected by a procedure that is guided by test cubes. The experiment proceeds as follows.

A procedure that was developed earlier, and is guided by test cubes, allows partial matches between the tests that the LFSR produces and the test cubes, as long as the tests detect target faults. Thus, the procedure is more flexible than a procedure that solves linear equations in order to find seeds for given test cubes. In an experiment whose goal was to study the effectiveness of this procedure, all the flip-flops of the circuit were included in a single scan chain, and a primitive LFSR from [13] was used for driving the scan chain directly. A binary search process yielded the LFSR with the smallest number of bits for which the procedure achieves the highest fault coverage. Let the number of bits in this LFSR be B0, and let the set of seeds be STC (B0).

In this brief, primitive B -bit LFSRs from [13] are considered for B = B0/2, B0/2 + 1,.., B0 − 1. Only one LFSR is given in [13] for every value of B, and this LFSR is used without any selection. For every value of B, the procedure based on test cubes is used for generating a set of seeds that is denoted by STC (B). With B < B0, there are cases where STC (B) does not detect all the detectable single stuck-at faults. Considering only the faults that remain undetected, Procedure 2 is used for generating a set of seeds that is denoted by SNTC (B).

Procedure 2 is applied with the following parameter values. The number of times Procedure 1 considers the bits of a seed for complementation, NMOD, is determined as follows. For I ≤ 100, where I is the iteration of Procedure 2, NMOD = I. For I > 100, NMOD = 100. Thus, the procedure considers all the bits of a seed once in iteration 1, twice in iteration 2, and so on. Beyond iteration 100 (if it is reached), the procedure considers all the bits of a seed 100 times.

The run time limit RT is defined with respect to the normalized run time of Procedure 2. For normalization, the run time is divided by the run time for single stuck-at fault simulation of the tests produced by STC (B0). Normalization provides an indication of the computational effort of Procedure 2, which is based on fault simulation. The value of RT is such that the normalized run time is limited to 1000.

Table II

Benchmark Circuits

circuit		
s1423		
s5378		
s9234		
s13207		
s15850		
s35932		
s38417		
s38584		
b04		
b07		
b14		
b15		
b20		
aes core		
des area		
i2c		
pci spoci ctrl		
sasc		
simple spi		
spi		
systemcaes		
systemcdes		
tv80		
usb phy		
wb dma		

The procedure based on test cubes was run with the same limit on its run time to compute STC (B), for B = B0/2, B0/2+1, ., B0 −1. A lower run time limit was used in the earlier study for computing B0 and STC (B0).

A high limit on the run time was selected in order to allow everyone of the procedures a sufficient number of iterations for every fault. With this limit, the procedure based on test cubes is not likely to find additional seeds even if it is given a higher run time. The results are shown in Tables II–IV. Table II shows all the benchmark circuits that are considered for this experiment. For every circuit, it shows the results of the procedure that is based on test cubes when it uses the B0-bit LFSR. Column in p shows the number of inputs to the combinational logic of the circuit. Column B shows the number of LFSR bits (the value of B0). Column f.c. shows the single stuck-at fault coverage that the procedure achieves. Column seed s shows the number of seeds that the procedure produces.

For most of the circuits in Table II, the procedure based on test cubes achieves the highest possible single stuck-at fault coverage by detecting all the detectable faults. The fault coverage varies with the LFSR when test cubes as well as nontest cubes are used. Tables III and IV report on cases with B = B0/2, B0/2 + 1, ..., B0 − 1, where the use of nontest cubes improves the fault coverage compared with the use of test cubes. As B is increased, Tables III and IV report on caseswhere the fault coverage of STC (B) ∪ SNTC (B) increases as well. The only exception is s1423, where all the values of B are reported.

For every circuit in Tables III and IV, column in p shows the number of inputs. Column B shows the number of LFSR bits, B. Column test cubes shows the results of the procedure that is guided by test cubes. The corresponding set of seeds is STC (B).Column nont est cubes shows the results of Procedure 2. The set of seeds considered in this case is STC (B) ∪ SNTC (B).

For both procedures, subcolumn f.c. shows the single stuck-at fault coverage. Subcolumn seeds shows the number of seeds. Subcolumn nt i me shows the normalized run time of the procedure. In addition, for Procedure 2, subcolumn U shows the average number of nontest cubes in a set Ui based on which a seed was computed. For ISCAS-89 benchmarks in Table III, subcolumn left shows the percentage of detectable faults that are left undetected by Procedure 2. For comparison, subcolumn rand shows the percentage of detected faults that are left undetected when 16K random tests are simulated

Table III

Fault Coverage Improvement With Non-Test Cubes

(ISCAS-89)

				t t b				t t b			
circuit	inp	B	f	d	t m	f	d	t m	U	l ft	d
s1423	91	9	94.72	57	1001.71						
s1423	91	10	96.96	59	1001.71						
s1423	91	11	98.15	60	1001.71						
s1423	91	12	98.15	61	1001.71	98.28	60	1002.00	9.00	0.80	0.20
s1423	91	13	98.75	65	1001.86						
s1423	91	14	99.01	64	1001.71						
s1423	91	15	98.35	64	1001.86						
s1423	91	16	98.94	63	1001.71						
s1423	91	17	9 6	65	1001 6	99.08	63	108.57	1.00	0.00	0.20
s5378	214	21	98.15	238	1002.31	98.18	239	1002.39	7.00	0.95	0.61
s5378	214	32	99.09	242	1002.10	99.13	243	118.37	11.00	0.00	0.61
s9234	247	37	92.83	333	1002.01	93.24	335	1002.01	9.93	0.23	9.02
s9234	247	40	92.75	345	1002.03	93.36	354	1002.09	7.41	0.11	9.02
s9234	247	46	93.33	339	1002.02	93.47	341	989.69	7.40	0.00	9.02
s13207	700	23	96.94	400	1002.05	97.73	442	1002.27	10.28	0.73	4.75
s13207	700	24	97.37	403	1002.07	98.22	449	1002.29	9.71	0.24	4.75
s13207	700	34	97.02	394	1002.07	98.39	459	1002.19	9.93	0.07	4.75
s15850	611	28	94.55	356	1001.90	95.79	389	1001.84	12.72	0.89	5.00
s15850	611	29	94.86	359	1002.07	95.92	394	1002.24	13.15	0.76	5.00
s15850	611	31	95.89	375	1002.13	96.21	385	1001.91	10.94	0.47	5.00
s15850	611	33	96.36	403	1002.14	96.55	409	1002.36	18.78	0.13	5.00
s15850	611	43	95.85	394	1001.81	96.61	427	1001.88	15.00	0.07	5.00
s15850	611	49	96.665	412	1001.71	96.674	412	923.18	10.00	0.006	5.00
			88.369	45	1002.00						

Table IV

Fault Coverage Improvement With Non-Test Cubes

(ITC-99 and IWLS-05)

				b				b	
circuit	inp	B	f	d	m	f	d	m	U
b04	78	14	99.18	47	1002.00	99.26	48	1005.50	6.00
b04	78	17	99.26	48	1002.00	99.33	49	1003.00	4.00
b04	78	20	99.63	50	1002.00	99.78	52	1002.00	4.00
b07	53	29	89.94	46	1002.50	90.11	46	1003.00	2.00
b07	53	32	96.96	52	1002.00	99.07	59	1002.50	8.09
b14	280	64	89.73	252	1002.13	89.78	253	1002.09	5.25
b14	280	65	90.51	276	1002.42	90.53	276	1001.85	7.00
b14	280	67	90.53	256	1002.29	90.58	257	1002.31	6.00
b14	280	75	90.53	269	1002.34	90.72	273	1002.41	5.78
b14	280	79	92.60	300	1002.35	92.61	300	1002.40	7.00
b14	280	87	94.17	330	1001.93	94.19	331	1002.04	7.00
b14	280	95	94.76	341	1001.92	94.77	341	1002.49	7.00
b14	280	110	94.95	360	1002.54	94.96	361	1002.65	7.00
b14	280	115	94.99	358	1004.17	95.00	358	1004.21	7.00
b15	483	56	95.89	412	1002.03	97.74	476	1002.11	3.80
b15	483	57	98.37	512	1002.16	98.39	513	1002.03	3.00
b15	483	61	98.47	517	1002.12	98.50	517	1001.88	8.40
b15	483	64	98.56	538	495.38	98.57	539	1002.23	3.00
b15	483	74	98.57	518	462.68	98.58	518	655.71	3.00
b20	527	59	88.03	270	1001.12	88.50	283	1001.20	4.97
b20	527	60	90.71	349	1001.73	90.77	352	1001.73	13.80
b20	527	63	91 0(	359	1001 (5	91.22	364	1001.95	6.10
b20	527	71	91.71	403	1002.98	91.82	408	1004.34	6.50
b20	527	79	92.47	427	1004.18	92.54	430	1004.22	3.43
aes core	788	14	99.57	574	1001.67	99.58	573	1002.71	8.00
aes core	788	18	99.598	574	1004.22	99.600	575	1002.88	7.00
i2c	145	24	99.27	99	1000.60	99.32	99	1000.67	21.00
pci spoci ctrl	3	3	95.34	153	1001.13	95.75	157	1001.16	14.60
pci spoci ctrl	83	40	98.19	171	1003.40	98.31	171	1000.49	3.00
simple spi	146	19	95.72	57	1005.11	97.91	61	1008.33	9.80
simple spi	146	20	99.52	67	1010.11	99.71	68	1009.44	16.50
simple spi	146	26	99.86	65	1001.22	99 90	(5	1001 5(	19 00
simple spi	146	27	99.14	62	1010.33	99.95	68	1010.22	10.27
simple spi	146	30	99 90	((	1010 00	100 00	(	313 00	11 00
spi	274	32	99.94	472	1001.94	99.97	472	235.07	17.00
y m	928	16	99 959	1 (	1001 (	99 9(3	1 (	1001 5	00
y m d	320	7	96.27	89	1001.57	96.29	87	1002.61	0.00
tv80	372	54	99.20	656	1002.95	99.35	659	809.51	8.71
tv80	372	56	99.33	665	625.33	99.35	666	32.77	3.00
wb dma	738	23	99.02	182	1002.25	99.23	192	1002.33	10.62
wb dma	738	25	99.37	198	1002.27	99.49	204	1002.29	11.60
wb dma	738	31	99.51	192	1002.16	99.53	193	1002.13	7.00
wb dma	738	34	99.90	213	1002.40	99.97	216	1002.28	17.33
wb dma	738	37	99.89	216	1002.16	99.99	218	1002.19	13.71
wb dma	738	38	99.96	210	1002.22	100.00	211	420.80	15.00

The information for Procedure 2 is omitted in the case of s1423 if the use of nontest cubes does not increase the fault coverage.

The following points can be seen from Tables III and IV. There are cases where the use of nontest cubes increases the fault coverage compared with the use of test cubes alone. The existence of such cases is significant given that the procedure based on test cubes already allows partial matches between the tests that the LFSR produces and the test cubes. Thus, it is not as constrained by the given test cubes as a procedure that solves linear equations for finding seeds. Even with this flexibility, the use of nontest cubes increases the fault coverage in a significant number of cases.

Procedure 2 finds nontrivial numbers of nontest cubes for target faults. These nontest cubes are effective in guiding the generation of seeds.

The number of seeds may be lower after nontest cubes are generated because Procedure 2 applies forward-looking reverse order fault simulation to remove seeds that become unnecessary. For this experiment, forward-looking reverse order fault simulation is applied to STC (B) ∪ SNTC (B).

Detailed consideration of the normalized run times indicates that the procedures typically reach the final fault coverage with a normalized run time that is significantly lower than 1000. Thus, they can be run with a lower run time limit. This can also be seen in Tables III and IV, for example, from the case of s1423 with B = 17, where Procedure 2 terminates after detecting all the detectable faults. It is also interesting to note that seeds are computed for faults that are not detected by random tests.

Table V

Using Non-Test Cubes Alone.

	t t b			nontest cubes only			
	f	d	t m	f	d	t m	U
	99.13	243	14.99	99.13	247	39.43	7.35
	93.47	348	42.97	93.47	330	591.63	7.83
	98.46	454	88.85	98.46	459	315.91	9.23
	96.68	418	36.65	96.68	420	252.53	10.71
	99.47	812	53.44	99.47	821	6247.57	5.82
aes core	100.00	571	29.23	100.00	579	75.22	6.31
des area	100.00	162	34.23	100.00	158	100.62	3.08
spi	99.98	475	79.23	99.98	459	284.33	3.38
systemcaes	100.00	182	19.67	100.00	182	87.98	4.43
tv80	99.33	660	34.23	99.36	667	1367.93	8.41
wb dma	100.00	202	45.67	100.00	209	203.43	9.22

Finally, Table V demonstrates that it is possible to use Procedure 2 for all the target faults, without first using test cubes to compute seeds. For Table V, the procedure based on test cubes and Procedure 2 are applied independently to all the target faults using the B0 bit LFSR.

Table V demonstrates that Procedure 2 can compute a complete set of seeds. Its run time is higher as discussed earlier, supporting its use only for hard-to-detect faults.

VI. Conclusion

This brief described a procedure for computing seeds for LFSR-based test generation without using test cubes. Instead, the procedure uses nontest cubes. This was motivated by the fact that a seed may not exist for a given test cube even though a seed may exist for a different test cube that detects the same fault. Thus, the use of test cubes limits the flexibility of a procedure to compute seeds for target faults. A nontest cube for a fault must be avoided in every test for the fault in order to allow the fault to be detected. Therefore, a nontest cube does not limit the ability of the procedure to compute seeds with a given LFSR. The cost of using nontest cubes is an increased computational effort for computing a seed. Experimental results demonstrated that, in spite of this cost, the procedure can compute seeds for some faults that cannot be detected by a procedure that uses test cubes.

References

[1] B. Koenemann, “LFSR-coded test patterns for scan designs,” in Proc.Eur. Test Conf., 1991, pp. 237–242.

[2] S.Hellebrand,S.Tarnick,J.Rajski, and B. Courtois,“Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers,” in Proc. Int. Test Conf., 1992, pp. 120–129.

[3] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, “OPMISR: The foundation for compressed ATPG vec- tors,” in Proc. Int. Test Conf., Oct. 2001, pp. 748–757.

[4] J. Rajski et al., “Embedded deterministic test for low cost manufacturing test” in Proc. Int. Test Conf., 2002, pp. 301–310.

[5] N. A. Touba, “Survey of test vector compression techniques,” IEEE Des.Test Comput., vol. 23, no. 4, pp. 294–303, Apr. 2006.

[6] S. Alampally, R. T. Venkatesh, P. Shanmugasundaram, R. A. Parekhji, and V. D. Agrawal, “An efficient test data reduction technique through dynamic pattern mixing across multiple fault models,” in Proc. IEEE 29th VLSI Test Symp., May 2011, pp. 285–290.

[7] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, P. Szczerbicki, and J. Tyszer, “Deterministic clustering of incompatible test cubes for higher power-aware EDT compression,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 30, no. 8, pp. 1225–1238, Aug. 2011.A. Chandra, J. Saikia, and R. Kapur, “Breaking the test application time barriers in compression: Adaptive scan-cyclical (AS-C),” in Proc. Asian Test Symp., Nov. 2011, pp. 432–437.

[8] O. Acevedo and D. Kagaris, “Using the Berlekamp–Massey algorithm to obtain LFSR characteristic polynomials for TPG,” in Proc. Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst., Oct. 2012, pp. 233–238.

[9] X. Lin and J. Rajski, “On utilizing test cube properties to reduce test data volume further,” in Proc. IEEE 21st Asian Test Symp., Nov. 2012, pp. 83–88.

[10] T. Moriyasu and S. Ohtake, “A method of one-pass seed generation for LFSR-based deterministic/pseudo-random testing of static faults,” in Proc. Latin-Amer. Test Symp., Mar. 2015, pp. 1–6.

[11] Pomeranz, “Non-test cubes for test generation targeting hard-to-detect faults,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 32, no. 12, pp. 1957–1965, Dec. 2013.

[12] P. H. Bardell, W. H. McAnney, and J. Savir, Built in Test for VLSI: Pseudorandom Techniques. New York, NY, USA: Wiley.