Functional Test Generation for Full Scan Circuits[+]

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

Abstract

We study the effectiveness of functional tests for full scan circuits. Functional tests are important for design validation, and they potentially have a high defect coverage independent of the circuit implementation. The functional fault model we consider consists of single state-transition faults. The test generation procedure we describe uses one of two approaches at any given time in order to minimize the number of tests while minimizing the test application time. (1) It may use scan to set the state of the circuit, and observe fault effects propagated to the next-state variables. (2) It may use transfer sequences to set the circuit state, or unique input-output sequences to propagate fault effects to the primary outputs. We present experimental results to demonstrate the effectiveness of scan-based functional tests.

1. Introduction

We study the effectiveness of functional testing of full scan circuits. Functional test generation is useful for several reasons. (1) It can be used for design verification and validation. In these applications, full scan can be simulated if it is not already part of the circuit description. (2) It results in a test set that is independent of the circuit implementation, and is expected to be effective for any implementation. The test set can be generated at early design stages, before an implementation is selected, and it remains valid as the implementation evolves throughout the design process. (3) The test set is effective in detecting defects of various types, more than a test set generated for a specific gate-level fault model, and more than if scan is not available. We discuss this point with respect to the specific functional fault model we use later on.

The circuits we consider are described by state tables, and the functional fault model we target is the single state-transition fault model [1]-[3]. Under this model, any single state-transition may result in a faulty next-state or output combination. However, the proposed methodology can be extended to other functional descriptions.

Given a functional description of a circuit in the form of a state table, and given that full-scan will be used for the circuit, it is possible to test every state-transition as follows. Consider a state-transition $s_i \xrightarrow[\beta_{ij}]{\alpha_j} s_{ij}$ from a state s_i under an input combination α_j, with next state s_{ij} and output combination β_{ij}. It is possible to test this state-transition by scanning-in the initial state s_i, applying the primary input combination α_j, observing the primary output combination, and scanning-out the next-state. If the output combination is different from β_{ij} or the next-state is different from s_{ij}, a fault is detected. Note that it is not necessary to specify a faulty next-state or output combination as part of the fault model, since any faulty value will be detected.

For a circuit with N_{ST} states and N_{PIC} primary input combinations, the number of state-transitions is $N_{ST} N_{PIC}$. Therefore, a test set that tests each state-transition separately contains $N_{ST} N_{PIC}$ tests, and requires $N_{ST} N_{PIC} + 1$ scan-in and scan-out operations. Here, we use the following terminology. A test starts and ends with a scan operation, and consists of one or more primary input combinations applied between the scan operations. The length of a test is the number of primary input combinations it applies between the scan operations. When each state-transition is tested by a separate test, the length of each test is one.

In the procedure proposed here, we attempt to reduce the number of tests and the number of scan operations by testing several state-transitions by the same test. In this way, we achieve several goals. (1) The circuit is tested at-speed during the application of test sequences whose length is larger than one. This may contribute to the detection of delay defects that are not detected if each state-transition is tested separately. (2) Longer test sequences also help reduce the number of scan operations required, and thus the test application time. To achieve these goals, we use a functional counterpart of the techniques described next.

Test generation procedures for gate-level scan designs that attempt to minimize the number of scan operations were described in [4]-[6]. These test generation procedures decide whether to continue applying input combinations in order to activate and propagate fault effects, or scan-out the state and scan-in a new state, based on the number of clock cycles required to detect target faults. In [7], a static compaction procedure was proposed for scan designs described at the gate-level. The procedure of [7] starts from a given set of tests, and reduces the number of scan operations by *combining* as many tests as possible. Combining two tests τ_i and τ_j results in the removal of the scan-out operation at the end of τ_i, and of the scan-in operation at the beginning of τ_j. The procedure of [7] attempts to combine every pair of tests, and accepts all the combinations that do not reduce the fault coverage.

For the functional description considered in this work, our goal is to develop a test generation procedure that detects as many state-transition faults as possible by the same test. When we do not scan-out the state following a state transition $s_i \xrightarrow[\beta_{ij}]{\alpha_j} s_{ij}$, we need to guarantee that any change in the next-state s_{ij} due to a fault will be detected. We achieve this by using unique input-output sequences [1] (to the extent that the selected unique input-output sequences are not aliased due to faults). A unique input-output sequence for a state s distinguishes s from every other state in the circuit. More accurately, let the output sequence pro-

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

duced by the circuit in response to input sequence A when the circuit starts in state s be $B(A, s)$. The sequence D_s is a unique input-output sequence for state s if $B(D_s, s) \neq B(D_s, \hat{s})$ for every state $\hat{s} \neq s$.

The functional tests produced by the proposed procedure have the following form. An initial state s_{i_0} is scanned-in. An input combination α_{j_0} is applied to test the state-transition from s_{i_0} under α_{j_0}. Let the next-state of this state-transition be $s_{i_0 j_0}$. If $s_{i_0 j_0}$ does not have a unique input-output sequence, then the generation of the test must stop, and the final state must be scanned-out. Otherwise, if $s_{i_0 j_0}$ has a unique input-output sequence $D_{s_{i_0 j_0}}$, we may apply this sequence and avoid the scan operation. We discuss the considerations behind the decision to apply $D_{s_{i_0 j_0}}$ below. For now, suppose that this sequence is applied. Let the final state reached after applying $D_{s_{i_0 j_0}}$ be s_{i_1}. We can now test a state-transition starting from s_{i_1}, if one exists that has not been tested yet. For this purpose, we apply an input combination α_{j_1}, followed by the unique input-output sequence for the next-state $s_{i_1 j_1}$, if it exists. Consequently, the test sequence has the following form.

$$s_{i_0} \overset{\alpha_{j_0}}{\rightarrow} s_{i_0 j_0} \overset{D_{s_{i_0 j_0}}}{\rightarrow} s_{i_1} \overset{\alpha_{j_1}}{\rightarrow} s_{i_1 j_1} \overset{D_{s_{i_1 j_1}}}{\rightarrow} s_{i_2} \overset{\alpha_{j_2}}{\rightarrow} s_{i_2 j_2} \cdots .$$

During the test generation process, the k-th state-transition tested ends in state $s_{i_k j_k}$. Assuming that a unique input-output sequence $D_{s_{i_k j_k}}$ exists, we need to decide whether or not to apply it. Suppose that $D_{s_{i_k j_k}}$ takes the circuit into state $s_{i_{k+1}}$. If there exists a yet-untested state-transition out of $s_{i_{k+1}}$, $D_{s_{i_k j_k}}$ is applied. If all the state-transitions out of $s_{i_{k+1}}$ have been tested, one of two options is taken. We look for a transfer sequence from $s_{i_{k+1}}$ to a state $\hat{s}_{i_{k+1}}$ that still has untested state-transitions. If a transfer sequence can be found, $D_{s_{i_k j_k}}$ is applied and test generation continues with the transfer sequence, and an input combination $\alpha_{j_{k+1}}$ that takes the circuit through a yet-untested state transition. If a transfer sequence cannot be found, test generation stops at state $s_{i_k j_k}$, and the final state is scanned out.

We control the overall number of clock cycles required for test application (including the time for scan-in/scan-out and the time for application of primary input combinations) by restricting the length of the unique input-output sequences we allow, and the lengths of the transfer sequences. For a state that does not have a unique input-output sequence, it is possible to use a subset of sequences, with each sequence distinguishing the state from a different subset of states. We do not explore this option here. The number of clock cycles required for test application is discussed in more detail later.

Functional test generation for state-transition faults in non-scan designs was shown in [2] and [3] to result in high coverage of gate-level stuck-at faults. Full-scan is expected to allow complete fault coverage to be achieved by functional tests. Experimental results reported below support this claim.

In the following sections, we present an example to demonstrate the proposed procedure. We then provide experimental results to show the number of tests obtained, and the coverage of gate-level stuck-at faults and bridging faults.

2. The procedure

In Table 1, we show the state table of MCNC finite-state machine benchmark *lion*. The machine has four states labeled 0, 1, 2 and 3, two inputs and one output. The input and output values are given as binary values.

Table 1: State table of *lion*

	NS, z for $x_1 x_2 =$			
PS	00	01	10	11
0	0,0	1,1	0,0	0,0
1	1,1	1,1	3,1	0,0
2	2,1	2,1	3,1	3,1
3	1,1	2,1	3,1	3,1

Unique input-output sequences for *lion* are shown in Table 2. For state 0, the input sequence (00) distinguishes state 0 (that produces an output of 0) from every other state (all other states produce an output value of 1). Under the sequence (00), the final state starting from state 0 is also state 0. The final state is given in the last column of Table 2. For state 1, a unique input-output sequence does not exist. To see this, we observe that starting with an input combination 00, it will not be possible to distinguish state 1 from state 3; starting with an input combination 01 or 11, it will not be possible to distinguish state 1 from state 0; and starting with an input combination 10, it will not be possible to distinguish state 1 from state 2 or 3. For state 2, the input combination 00 distinguishes it from state 0, takes states 1 and 3 to state 1, and takes state 2 to state 2. Applying the input combination 11 following the input combination 00 distinguishes states 1 and 2. This results in the unique input-output sequence (00,11) for state 2. Under the sequence (00,11), the final state starting from state 2 is state 3. State 3 does not have a unique input-output sequence. In general, we find at most one unique input-output sequence for every state, and use it throughout the test generation process.

Table 2: Unique input-output sequences for *lion*

state	unique	f.state
0	00	0
1	-	-
2	00 11	3
3	-	-

During the derivation of unique input-output sequences, we limit the length of a sequence to be at most L, where L is a constant. The reason for this is as follows. A scan-in/scan-out operation requires N_{SV} clock cycles, where N_{SV} is the number of state variables. The application of a unique input-output sequence D_s of length L requires L clock cycles. Assuming that the clock controlling the scan chain has the same cycle time as the clock controlling the circuit, we need to use $L \leq N_{SV}$ to ensure that the application of D_s does not take more time than scanning-out a state and scanning-in a new state. The length of D_s may have to be even shorter if transfer sequences are used during the construction of the tests. However, we may also allow D_s to be longer than N_{SV} in order to take advantage of at-speed testing when tests include multiple state-transitions. In addition, there are cases where scan is done at a slow speed. If the scan clock is M times slower than the circuit clock, unique input-output sequences and transfer sequences that are M times longer can be accommodated without increasing the test application time.

Next, we construct tests for single state-transition faults in *lion* using the unique input-output sequences of Table 2. We start the first test with the state-transition $0 \overset{00}{\rightarrow} 0$. We follow it by the unique input-output sequence of state 0, which is (00). The final state is state 0, and we can test another state-transition starting from state 0. For this purpose, we add the state-transition $0 \overset{01}{\rightarrow} 1$. State 1 does not have a unique input-output sequence, and we stop the construction of the test at this point. We will verify that state 1 is reached by scanning-out the last state. Our first test is $\tau_0 = (0, (00, 00, 01), 1)$, with initial state 0, test sequence (00,00,01), and final state 1. Up to this point, we considered the state-transitions $0 \xrightarrow{00} 0$ and $0 \xrightarrow{01} 1$.

We start the construction of the second test with the state-transition $0 \xrightarrow{10} 0$. We follow it by the unique input-output sequence of state 0, which is (00). The final state is state 0, and we can test another state-transition starting from state 0. For this purpose, we add the state-transition $0 \xrightarrow{11} 0$. If we add the unique input-output sequence for state 0, we will end up in state 0 again. There are no additional state-transitions to test out of state 0. Therefore, we check whether a transfer sequence exists from state 0 into a state with untested state-transitions. We find that it is possible to take the circuit from state 0 to state 1 by applying the input combination 01. Therefore, we apply the unique input-output sequence of state 0, then take the state-transition $0 \xrightarrow{01} 1$ into state 1, followed by the state-transition $1 \xrightarrow{00} 1$ that has not been tested yet. State 1 does not have a unique input-output sequence, and we stop the construction of the test at this point. Our second test is $\tau_1 = (0, (10, 00, 11, 00, 01, 00), 1)$. Up to this point, we considered the state-transitions $0 \xrightarrow{00} 0$, $0 \xrightarrow{01} 1$, $0 \xrightarrow{10} 0$, $0 \xrightarrow{11} 0$ and $1 \xrightarrow{00} 1$.

We skip over the state-transition $1 \xrightarrow{01} 1$. This state-transition ends in state 1 that does not have a unique input-output sequence. Consequently, considering it next will result in a test sequence of length one, (1,(01),1). By postponing its consideration, it is possible that it will be tested later as part of another test of length larger than one. In general, we postpone the consideration of a state-transition $s_i \xrightarrow{\alpha_j} s_{ij}$ as the first state-transition of a test if s_{ij} does not have a unique input-output sequence.

The third test, τ_2, starts from the state-transition $1 \xrightarrow{11} 0$. We add the unique input-output sequence for state 0 that takes us back to state 0, followed by the transfer sequence (01) into state 1. We now test the state-transition $1 \xrightarrow{01} 1$. The final state, 1, does not have a unique input-output sequence. Therefore, we must scan out the final state. We have $\tau_2 = (1, (11, 00, 01, 01), 1)$.

The construction of τ_3 starts with the state-transition $2 \xrightarrow{00} 2$, and followed by the unique input-output sequence of state 2, which is (00,11). The final state is 3, and we test the state transition $3 \xrightarrow{00} 1$. We obtain $\tau_3 = (2, (00, 00, 11, 00), 1)$.

The construction of τ_4 starts with the state-transition $2 \xrightarrow{01} 2$. We add the unique input-output sequence (00,11) of state 2, and end at state 3. Next, we test the state-transition $3 \xrightarrow{01} 2$. We add the unique input-output sequence (00,11) of state 2, and end at state 3 again. Next, we test the state-transition $3 \xrightarrow{10} 3$. We obtain $\tau_4 = (2, (01, 00, 11, 01, 00, 11, 10), 3)$.

The final four tests for the remaining state-transitions are $\tau_5 = (1, (10), 3)$, $\tau_6 = (2, (10), 3)$, $\tau_7 = (2, (11), 3)$ and $\tau_8 = (3, (11), 3)$.

The tests obtained above for *lion* include every single state-transition in its state table (we do not claim that all the single state-transition faults are detected, since faults may affect the unique input-output sequences; however, this is expected to affect the coverage of single state-transition faults only rarely). Next, we consider the detection of faults in a gate-level implementation of *lion*.

If all the functional tests are of length one, then the functional test set is equivalent to an exhaustive test set for the combinational logic of the circuit. In this case, the test set is guaranteed to detect every fault that does not increase the number of circuit states. However, delay faults that require at-speed testing are not guaranteed to be detected. When longer tests are used, the likelihood of detecting delay faults is increased; however, it is possible that a gate-level fault would remain undetected even if it does not require at-speed testing. This is because some gate-level faults are equivalent to multiple state-transition faults, and are not detected by tests for single state-transition faults, even if the tests detect all the single state-transition faults. However, due to the use of scan, this is expected to be rare. We use fault simulation on the gate-level circuit to determine the coverage of gate-level non-delay faults belonging to two fault models, stuck-at faults and bridging faults.

We observe that not all the tests may be necessary to detect gate-level faults. To eliminate unnecessary tests, we simulate the tests in decreasing order of length, where the length of a test is the number of primary input combinations it includes. The premise behind this order is that longer tests detect more faults, and it will be possible to remove a large number of short tests by starting from the longer ones (the removal of each test results in the removal of a scan operation regardless of the length of the test, thus reducing the test application time). The results of stuck-at fault simulation for *lion*, using the tests $\tau_0, \cdots, \tau_8$ generated above, are shown in Table 3. The circuit has 40 stuck-at faults. The tests in Table 3 are ordered by the order of simulation, from the longest to the shortest. For every test, we show its length, and the total number of stuck-at faults detected after it is simulated. In the last column, we mark the test as effective if any new faults are detected when it is simulated. Of the nine functional tests we generated, four tests are needed to detect all the stuck-at faults in the circuit. None of the length one tests is required in this case.

Table 3: Stuck-at fault simulation for *lion*

test	length	detected	effective
τ_4	7	17	1
τ_1	6	37	1
τ_2	4	39	1
τ_3	4	40	1
τ_0	3	40	0
τ_5	1	40	0
τ_6	1	40	0
τ_7	1	40	0
τ_8	1	40	0

3. Experimental results

The results of the procedure described above are reported in this section. We limit the length of the unique input-output sequences to be at most equal to the number of state variables, and we limit the length of the transfer sequences to be at most one. In selecting these parameters, we use the following considerations. Allowing longer unique input-output sequences and longer transfer sequences will allow us to obtain longer test sequences that test more state-transitions by the same test. This is advantageous for at-speed testing that enhances the detection of delay defects. However, it may also increase the test application time since the application of a unique input-output sequence followed by a transfer sequence may take a larger number of clock cycles than that required for scanning out the final state and scanning in a new initial state. The parameters we chose ensure that a unique input-output sequence followed by a transfer sequence will require at most one clock cycle more than scanning out a state and scanning in a new state; in most cases, fewer clock cycles will be required. Thus, the test application time will not be increased, or not be significantly increased, yet several state-transitions will be tested by the same test.

In Table 4, we show the parameters of the circuits we consider. After the circuit name, we show the number of primary inputs. We then show the number of states, and the number of states for which unique input-output sequences were found. Next, we show the number of state variables of the circuit, and the maximum length of any unique input-output sequence. In the last column of Table 4, we show the time to generate unique input-output sequences. Time is given in seconds on an HP J210 workstation.

Table 4: Circuit parameters

circuit	pi	states	unique	sv	m.len	time
bbara	4	16	4	4	4	11.49
bbsse	7	16	13	4	3	7.64
bbtas	2	8	1	3	3	0.08
beecount	3	8	5	3	3	0.05
cse	7	16	15	4	3	36.21
dk14	3	8	1	3	1	0.08
dk15	3	4	3	2	2	0.02
dk16	2	32	23	5	3	4.70
dk17	2	8	6	3	2	0.03
dk27	1	8	5	3	3	0.01
dk512	1	16	6	4	4	0.14
dvram	8	64	48	6	6	5649.94
ex2	2	32	14	5	4	2.36
ex3	2	16	10	4	3	0.26
ex4	5	16	9	4	4	18.98
ex5	2	8	7	3	3	0.08
ex6	5	8	8	3	1	0.11
ex7	2	16	10	4	3	0.29
fetch	9	32	24	5	4	473.35
keyb	7	32	21	5	4	266.42
lion	2	4	2	2	2	0.00
lion9	2	8	2	3	2	0.01
log	9	32	13	5	5	639.51
mark1	4	16	12	4	4	2.82
mc	3	4	4	2	1	0.00
nucpwr	13	32	20	5	5	1887.44
opus	5	16	7	4	1	2.78
rie	9	32	28	5	5	3042.78
shiftreg	1	8	8	3	3	0.01
tav	4	4	2	2	2	0.07
train11	2	16	2	4	3	0.11

In Table 5, after the circuit name, we show the number of state-transitions. This is also the number of tests if each state-transition is tested by a separate test. Under column *funct.tests* we show the results of functional test generation by the proposed procedure. We show the number of tests, and the total length of all the tests. Next, we show the percentage of state-transitions tested by tests of length one (such tests detect a single state-transition). Finally, we show the test generation time. Comparing the number of tests to the number of state-transitions, it can be seen that the proposed procedure succeeds in testing several state-transitions by the same test. This can also be seen from the percentage of state-transitions tested by tests of length one, given under subcolumn 1*len*. On the average, less than 50% of the state-transitions are tested by tests of length one. The other state-transitions are tested by tests that detect at least two state-transitions.

Table 5: Functional test generation

		funct.tests			
circuit	trans	tests	len	1len	time
bbara	256	202	434	63.28	0.10
bbsse	2048	1515	2914	62.70	35.18
bbtas	32	28	44	75.00	0.00
beecount	64	32	153	40.62	0.04
cse	2048	1436	3141	59.96	60.06
dk14	64	51	82	64.06	0.03
dk15	32	11	76	15.62	0.01
dk16	128	63	317	26.56	0.22
dk17	32	20	53	43.75	0.01
dk27	16	8	40	31.25	0.01
dk512	32	25	58	59.38	0.01
dvram	16384	12088	33891	61.71	907.91
ex2	128	93	256	53.91	0.12
ex3	64	41	130	54.69	0.04
ex4	512	384	1006	55.86	0.83
ex5	32	17	73	21.88	0.01
ex6	256	76	501	15.23	0.63
ex7	64	44	125	57.81	0.04
fetch	16384	11347	26100	55.40	1272.69
keyb	4096	3528	5312	82.35	172.71
lion	16	9	28	25.00	0.00
lion9	32	22	56	46.88	0.01
log	16384	11520	34560	51.42	533.81
mark1	256	109	653	35.16	0.38
mc	32	9	57	25.00	0.01
nucpwr	262144	172032	446464	44.53	373906.81
opus	512	378	698	54.10	0.23
rie	16384	11037	31457	57.50	2311.50
shiftreg	16	13	27	75.00	0.00
tav	64	33	125	25.00	0.01
train11	64	53	93	65.62	0.02
average				48.59	

In Table 6, we report on the coverage of stuck-at faults and bridging faults in gate-level implementations. Under column *s.a.tsts*, we show the number of tests effective in detecting gate-level stuck-at faults, followed by the total length of all these tests. Under column *s.a.faults*, we show the total number of stuck-at faults, the number of faults detected by the generated tests, and the fault coverage. All the circuits with lower than 100% fault coverage have combinationally redundant faults that cannot be detected under full-scan. All the detectable faults in all the circuits are detected by the proposed procedure. It can be seen that relatively small numbers of tests are required.

Under columns *bridg.tsts* and *bridg.faults* of Table 6, we show the results of simulating gate-level bridging faults under the functional tests generated here. We consider non-feedback bridging faults between every pair of lines g_1 and g_2 that satisfy the following conditions. (1) g_1 and g_2 are outputs of multi-input gates. (2) g_1 and g_2 are inputs of different gates. (3) There is no path in the circuit from g_1 to g_2 or from g_2 to g_1. We consider both AND-type and OR-type bridging faults between every such pair of lines. It can be seen that for most of the circuits, the coverage of bridging faults is 100%. For the remaining circuits, we verified by simulating an exhaustive test set for the combinational logic of the circuit that all the bridging faults that remain

Table 6: Simulation of gate-level faults

	s.a.tsts		s.a.faults			bridg.tsts		bridg.faults		
circuit	tsts	len	tot	det	f.c.	tsts	len	tot	det	f.c.
bbara	29	133	138	138	100.00	9	85	192	192	100.00
bbsse	36	765	238	238	100.00	15	673	656	656	100.00
bbtas	12	28	63	63	100.00	6	22	64	64	100.00
beecount	5	93	112	110	98.21	2	83	166	166	100.00
cse	42	959	357	355	99.44	20	703	1604	1597	99.56
dk14	29	60	208	207	99.52	13	40	362	362	100.00
dk15	8	69	151	151	100.00	2	40	140	140	100.00
dk16	30	266	532	530	99.62	8	169	1942	1942	100.00
dk17	10	43	128	128	100.00	2	24	120	120	100.00
dk27	2	22	67	67	100.00	1	18	50	50	100.00
dk512	14	41	124	124	100.00	2	17	136	136	100.00
dvram	18	696	425	425	100.00	19	826	2672	2672	100.00
ex2	27	148	312	312	100.00	6	74	802	799	99.63
ex3	10	82	153	153	100.00	1	52	242	241	99.59
ex4	20	248	176	176	100.00	9	231	288	288	100.00
ex5	9	42	152	138	90.79	6	39	210	210	100.00
ex6	9	324	229	229	100.00	6	310	660	658	99.70
ex7	15	85	160	159	99.38	5	71	238	238	100.00
fetch	34	863	345	342	99.13	44	1628	1564	1564	100.00
keyb	62	1161	470	470	100.00	30	1084	3194	3177	99.47
lion	4	21	40	40	100.00	4	21	18	17	94.44
lion9	7	32	62	59	95.16	3	25	52	51	98.08
log	24	1141	313	312	99.68	37	1685	1618	1617	99.94
mark1	9	400	204	203	99.51	4	392	532	532	100.00
mc	3	51	73	73	100.00	2	50	54	54	100.00
nucpwr	39	300	447	447	100.00	91	752	3238	3237	99.97
opus	22	97	181	181	100.00	14	82	452	451	99.78
rie	42	1145	552	548	99.28	58	1876	4214	4213	99.98
shiftreg	2	16	28	28	100.00	1	15	8	8	100.00
tav	2	62	64	64	100.00	2	64	86	86	100.00
train11	11	39	104	104	100.00	6	32	132	132	100.00

undetected are undetectable. Thus, complete coverage of detectable bridging faults is achieved for all the circuits.

In Table 7, we show the numbers of clock cycles required for test application in the following cases. (1) When every state-transition is included in a separate test (column *trans*). (2) When the functional tests produced by the proposed procedure are used (column *funct.tests*). (3) When only the effective tests found after stuck-at fault simulation of the functional tests are used (column *s.a.tests*). (4) When only the effective tests found after bridging fault simulation of the functional tests are used (column *bridg.tests*). For a circuit with N_{SV} state variables, N_T tests, and a total of N_{PIC} primary input combinations included in these tests, the number of clock cycles is computed as $N_{SV}(N_T + 1) + N_{PIC}$. In this formula, $N_{SV}(N_T + 1)$ is the contribution of scan operations for N_T tests, and N_{PIC} clock cycles are required to apply all the input combinations. Here, we assume that the clock controlling the scan chain and the clock controlling the circuit operation have the same cycle time. In practice, the scan clock may be much slower than the circuit clock, and then it is necessary to multiply the contribution of the scan operations by the ratio of the two clock cycles. The percentages in Table 7 are given out of the number of clock cycles required when every state-transition is included in a separate test. It can be seen that in most cases, the proposed procedure does not increase the number of clock cycles required for the application of functional tests. An increase in the number of clock cycles may occur because the total length of unique input-output sequences and transfer sequences (applied using the circuit clock) exceeds the number of state variables (which is the number of clock cycles for scan-in/out). This can be corrected by restricting the lengths of the unique input-output sequences as discussed above, or eliminating transfer sequences, as we show below.

Table 7: Numbers of clock cycles

		funct.tests		s.a.tests		bridg.tests	
circuit	trans	cycles	%	cycles	%	cycles	%
bbara	1284	1246	97.04	253	19.70	125	10.03
bbsse	10244	8978	87.64	913	8.91	737	8.21
bbtas	131	131	100.00	67	51.15	43	32.82
beecount	259	252	97.30	111	42.86	92	36.51
cse	10244	8889	86.77	1131	11.04	787	8.85
dk14	259	238	91.89	150	57.92	82	34.45
dk15	98	100	102.04	87	88.78	46	46.00
dk16	773	637	82.41	421	54.46	214	33.59
dk17	131	116	88.55	76	58.02	33	28.45
dk27	67	67	100.00	31	46.27	24	35.82
dk512	164	162	98.78	101	61.59	29	17.90
dvram	114694	106425	92.79	810	0.71	946	0.89
ex2	773	726	93.92	288	37.26	109	15.01
ex3	324	298	91.98	126	38.89	60	20.13
ex4	2564	2546	99.30	332	12.95	271	10.64
ex5	131	127	96.95	72	54.96	60	47.24
ex6	1027	732	71.28	354	34.47	331	45.22
ex7	324	305	94.14	149	45.99	95	31.15
fetch	98309	82840	84.26	1038	1.06	1853	2.24
keyb	24581	22957	93.39	1476	6.00	1239	5.40
lion	50	48	96.00	31	62.00	31	64.58
lion9	131	125	95.42	56	42.75	37	29.60
log	98309	92165	93.75	1266	1.29	1875	2.03
mark1	1284	1093	85.12	440	34.27	412	37.69
mc	98	77	78.57	59	60.20	56	72.73
nucpwr	1572869	1306629	83.07	500	0.03	1212	0.09
opus	2564	2214	86.35	189	7.37	142	6.41
rie	98309	86647	88.14	1360	1.38	2171	2.51
shiftreg	67	69	102.99	25	37.31	21	30.43
tav	194	193	99.48	68	35.05	70	36.27
train11	324	309	95.37	87	26.85	60	19.42
average			92.09		33.60		24.91

A gate-level stuck-at test generation procedure applied to the full-scan circuits may yield numbers of tests and numbers of clock cycles that are better than the ones of Tables 6 and 7. However, it is not guaranteed to detect all the bridging faults. Similarly, a gate-level bridging fault test generation procedure applied to the full-scan circuits may not detect all the stuck-at faults. With the functional tests generated here, all the detectable faults of both models are detected.

In Table 8, we report the results of the proposed test generation procedure when transfer sequences are not allowed. In this case, if the state reached after a unique-input output sequence for a state $s_{i_k j_k}$ is s_{i_k}, and all the state-transitions out of s_{i_k} have been tested, the test sequence is terminated at state $s_{i_k j_k}$. We only report on circuits for which the percentage of clock cycles for application of the functional tests is 100% or higher in Table 7. Comparing the results in Table 8 to the results in Tables 5 and 7, it can be seen that, overall, fewer state-transitions are tested by the same test when transfer sequences are not allowed; however, eliminating the transfer sequences allows us to reduce the test application time.

Table 8: Test generation without transfer sequences

circuit	trans	tests	len	1len	cycles	%
bbtas	32	28	44	75.00	131	100.00
dk15	32	23	46	59.38	94	95.92
dk27	16	12	26	62.50	65	97.01
shiftreg	16	14	22	81.25	67	100.00

It is possible to explore other solutions by imposing different constraints on the lengths of unique input-output sequences and transfer sequences. In Table 9, we show the results obtained for several circuits when the length of the transfer sequences is limited to one, and the lengths of the unique input-output sequences are limited to 1, 2, 3, $\cdots$, until a further increase in the upper bound on the length of a unique input-output sequence does not increase the number of states for which we can find unique input-output sequences. The results for each length are shown on a separate line in Table 9. The results of Table 9 demonstrate the effects of the unique input-output sequence length on the number of state-transitions that can be tested by the same test, and on the number of clock cycles required for test application.

Table 9: Results with different parameters

(a) *dk*512

unique	m.len	tests	len	1len	cycles	%
0	1	32	32	100.00	164	100.00
1	2	29	39	81.25	159	96.95
4	3	23	60	46.88	156	95.12
6	4	25	58	59.38	162	98.78
8	5	24	67	56.25	167	101.83

(b) *ex*4

unique	m.len	tests	len	1len	cycles	%
0	1	512	512	100.00	2564	100.00
5	2	400	800	56.25	2404	93.76
7	3	352	992	37.50	2404	93.76
9	4	384	1006	55.86	2546	99.30
11	5	384	1101	67.38	2641	103.00
13	6	384	1197	72.85	2737	106.75
16	7	384	1197	72.85	2737	106.75

(c) *mark*1

unique	m.len	tests	len	1len	cycles	%
2	1	222	306	75.00	1198	93.30
6	2	123	610	35.55	1106	86.14
11	3	111	649	35.55	1097	85.44
12	4	109	653	35.16	1093	85.12

(d) *rie*

unique	m.len	tests	len	1len	cycles	%
3	1	13961	19888	73.87	89698	91.24
17	2	12048	24544	59.35	84789	86.25
24	3	11036	30434	57.49	85619	87.09
25	4	11036	30946	57.50	86131	87.61
28	5	11036	31458	57.50	86643	88.13
29	6	11036	31586	57.50	86771	88.26
30	7	10952	32640	59.25	87405	88.91
32	8	10882	35079	61.16	89494	91.03

4. Concluding remarks

We described a procedure for generating functional tests for fully scanned finite-state machines. The functional fault model we considered consisted of single state-transitions resulting in faulty next-states or output combinations. The proposed procedure used scan to set the initial state of the circuit. It then applied an input combination to take the circuit through a state-transition that has not been tested yet. If possible, a unique input-output sequence for the final state was used for testing the final state. Following this sequence, another input combination was applied to take the circuit through another state-transition that has not been tested yet. Alternatively, a transfer sequence was first used to take the circuit into an appropriate state. A test ended with a scan-out operation to test the final state of the last state-transition considered. In this procedure, unique input-output sequences and transfer sequences were used instead of scan operations as much as possible to obtain fewer, longer test sequences while keeping the test application time about the same. Experimental results showed that the functional tests achieve complete coverage of stuck-at faults and bridging faults in gate-level implementations. Earlier procedures that did not use scan did not report complete fault coverage of gate-level faults. This points to the effectiveness of scan-based functional tests.

References

[1] K. Sabnani and A.T. Dahbura, "A Protocol Test Generation Procedure", Computer Networks, 1988, pp. 285-297.

[2] K.-T. Cheng and J.Y. Jou, "Functional Test Generation for Finite State Machines", in Proc. Intl. Test Conf., 1990, pp. 162-168.

[3] I. Pomeranz and S. M. Reddy, "On Achieving Complete Fault Coverage for Sequential Machines", IEEE Trans. on Computer-Aided Design, March 1994, pp. 378-386.

[4] D. K. Pradhan and J. Saxena, "A Design for Testability Scheme to Reduce Test Application Time in Full Scan", in Proc. 10th VLSI Test Symp., April 1992, pp. 55-60.

[5] S. Y. Lee and K. K. Saluja, "An Algorithm to Reduce Test Application Time in Full Scan Designs", in Proc. 1992 Intl. Conf. on Computer-Aided Design, Nov. 1992, pp. 17-20.

[6] S. Y. Lee and K. K. Saluja, "Test Application Time Reduction for Sequential Circuits with Scan", IEEE Trans. on Computer-Aided Design, Sept. 1995, pp. 1128-1140.

[7] I. Pomeranz and S. M. Reddy, "Static Test Compaction for Scan-Based Designs to Reduce Test Application Time", in Proc. 7th Asian Test Symp., Dec. 1998, pp. 198-203.