Built-In Generation of Functional Broadside Tests

Irith Pomeranz
School of Electrical & Computer Eng.
Purdue University
W. Lafayette, IN 47907, U.S.A.

Abstract - **Functional broadside tests are two-pattern scan-based tests that avoid overtesting by ensuring that a circuit traverses only reachable states during the functional clock cycles of a test. On-chip test generation has the added advantage that it reduces test data volume and facilitates at-speed test application. This paper shows that on-chip generation of functional broadside tests can be done using simple hardware, and can achieve high transition fault coverage for testable circuits. With the proposed on-chip test generation method, the circuit is used for generating reachable states during test application. This alleviates the need to compute reachable states off-line.**

I. INTRODUCTION

Overtesting due to the application of two-pattern scan-based tests was described in [1]-[3]. Overtesting is related to the detection of delay faults under non-functional operation conditions. When an arbitrary state is used as a scan-in state, a two-pattern test can take the circuit through state-transitions that cannot occur during functional operation. As a result, slow paths that cannot be sensitized during functional operation may cause the circuit to fail [1]. In addition, current demands that are higher than those possible during functional operation may cause voltage drops that will slow the circuit and cause it to fail [2]-[3]. In both cases, the circuit will operate correctly during functional operation.

Functional broadside tests [9] ensure that the scan-in state is a state that the circuit can enter during functional operation, or a reachable state. As broadside tests [4], they operate the circuit in functional mode for two clock cycles after an initial state is scanned in. This results in the application of a two-pattern test. Since the scan-in state is a reachable state, the circuit goes through state-transitions that are guaranteed to be possible during functional operation. Delay faults that are detected by the test can also affect functional operation. This alleviates the type of overtesting described in [1]-[3].

Test generation procedures for functional and pseudo-functional scan-based tests were described in [5]-[13]. The procedures generate test sets for application from an external tester. Functional scan-based tests use only reachable states as scan-in states. Pseudo-functional scan-based tests use functional constraints to avoid unreachable states that are captured by the constraints.

This work considers the on-chip (or built-in) generation of functional broadside tests. On-chip test generation reduces the test data volume and facilitates at-speed test application. On-chip test generation methods for delay faults, such as the ones described in [14] and [15], do not impose any constraints on the states used as scan-in states. Experimental results indicate that an arbitrary state used as a scan-in state is unlikely to be a reachable state. The on-chip test generation method from [16] applies pseudo-functional scan-based tests. Experimental results indicate that pseudo-functional tests are not sufficient for avoiding unreachable states as scan-in states. The on-chip test generation process described in this work guarantees that only reachable states will be used.

Under the proposed on-chip test generation method, the circuit is used for generating reachable states during test application. This alleviates the need to compute reachable states or functional constraints by an off-line process as in [5]-[13] and [16]. The underlying observation is related to one of the methods used in [9] for external test generation, and is the following. If a primary input sequence A is applied in functional mode starting from a reachable state, all the states traversed under A are reachable states. Any one of these states can be used for the application of a functional broadside test. By generating A on-chip and ensuring that it takes the circuit through a varied set of states, the on-chip test generation process is able to achieve high transition fault coverage using functional broadside tests based on A.

When the circuit-under-test is embedded in a larger design, its primary inputs may be driven by other logic blocks that are part of the same design. In addition, the primary inputs of the circuit-under-test include any external inputs of the design that drive the circuit-under-test. The primary outputs of the circuit-under-test may drive other logic blocks, or they may be primary outputs of the complete design. For simplicity this paper assumes that primary inputs can be assigned any combination of values.

The paper is organized as follows. Section II gives an overview of the on-chip generation and application of functional broadside tests. Section III describes the details. Section IV presents experimental results demonstrating the achievable fault coverage.

978-3-9810801-7-9/DATE11/©2011 EDAA

II. OVERVIEW

This section gives an overview of the proposed method.

This paper assumes that the circuit is initialized into a known state before functional operation starts. Initialization may be achieved by applying a synchronizing sequence as in [5]-[7] and [9]-[12], by asserting a reset input as in [8] and [13], or by a combination of both. We denote the initial state of the circuit by s_r.

With s_r as the initial state for functional operation, the set of reachable states consists of every state s_i such that there exists a primary input sequence that takes the circuit from s_r to s_i. Since s_i can be entered during functional operation starting from s_r, s_i is a reachable state.

It is possible to obtain reachable states on-chip by placing the circuit in state s_r and applying a primary input sequence $A = a(0)\, a(1)\, \cdots\, a(L-1)$ of length L in functional mode. The circuit can be brought into state s_r by using a scan-in operation, or by using its initializing sequence. Let $s(u)$ be the state that the circuit reaches at time unit u under A, for $0 \le u \le L$. We have that $s(0) = s_r$. In addition, $s(u)$ is a reachable state for $0 \le u \le L$. Therefore, every state $s(u)$ can be used as a basis for a functional broadside test $<s(u),\ a_1,\ a_2>$, where $s(u)$ plays the role of a scan-in state. As in a broadside test, a_1 and a_2 are primary input vectors that are applied in two consecutive functional clock cycles starting from $s(u)$ using a slow and a fast clock, respectively.

Every subsequence of length two of A defines a functional broadside test $t(u) = <s(u),\ a(u),\ a(u+1)>$. By using $a(u)$ and $a(u+1)$ from A, it is possible to avoid the need for a different source for these primary input vectors during on-chip test generation.

For illustration we consider $s27$ with initial state $s_r = 000$. The circuit is shown in Figure 1. A primary input sequence for the circuit is shown in Table I. For every time unit u, Table I shows the state $s(u)$ and the primary input vector $a(u)$. The other columns of Table I will be explained later. Table I yields the functional broadside tests $t(0) = <000,\ 1001,\ 1000>$, $t(1) = <010,\ 1000,\ 1100>$, $\cdots$, $t(14) = <101,\ 1000,\ 1001>$.

The proposed on-chip generation method of functional broadside tests is based on placing the circuit in the initial state s_r, applying a primary input sequence A, and using several of the functional broadside tests that can be extracted from A in order to detect target faults.

Next, we discuss how the application of A is affected by the need to observe fault effects created by a test $t(u) = <s(u), a(u), a(u+1)>$.

At time unit u the circuit is in state $s(u)$. Applying $a(u)$ and $a(u+1)$ in functional mode will result in the application of $t(u)$. A fault can be detected in one of two ways. (1) Based on the primary output vector $z(u+1)$ obtained in response to $a(u+1)$, if this vector is different from the expected fault free primary output vector. (2) Based on the final state $s(u+2)$ of the test, if this state is different from the expected fault free state.

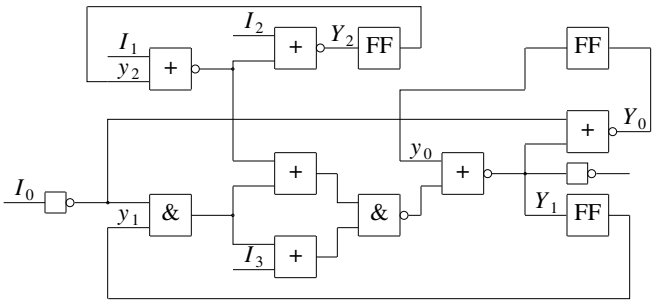

Figure 1. $s27$

TABLE I. Primary input sequence for $s27$

u	$s(u)$	$a(u)$	RS_1	RS_2
0	000	1001	0011	3
1	010	1000	1010	2
2	100	1100	1100	1
3	101	1101	1111	2
4	101	1001	0011	3
5	101	0110	0110	0
6	000	1100	0100	3
7	101	1011	1011	0
8	100	1001	0011	3
9	100	1100	0110	2
10	101	1001	1001	2
11	101	1001	0011	1
12	101	1100	1110	2
13	101	1001	1011	2
14	101	1000	0000	2
15	101	1001	0011	2

In the context of built-in self-test, $z(u+1)$ and $s(u+2)$ need to be captured by an output response compactor such as a *MISR* [17]. In the case of $s(u+2)$, the state needs to be scanned out and shifted into the output response compactor over a number of clock cycles equal to the length of the longest scan chain. The circuit then needs to be brought back to state $s(u+2)$ in order to continue the test application process under A.

Bringing the circuit back to state $s(u+2)$ can be done by using circular shift of $s(u+2)$ [17]. As $s(u+2)$ is scanned out, it can also be scanned in. If $s(u+2)$ is faulty, the output response compactor will capture the fault effect, and observation of the final signature will indicate that a fault is present. If $s(u+2)$ is fault free, the remaining tests based on A will be applied as required.

The application of overlapping tests based on A requires special hardware. To avoid it, this paper focuses on subsets of non-overlapping tests of the form $\{t(u_0), t(u_1), \cdots, t(u_{k-1})\}$, where $u_i + 1 < u_{i+1}$ for $0 \le i < k-1$.

III. ON-CHIP GENERATION OF FUNCTIONAL BROADSIDE TESTS

This section describes the on-chip generation method for functional broadside tests based on the concepts discussed in Section II. It first describes the genera-

tion of the sequence A. It then describes the selection of tests that will be applied based on A. Finally it describes the overall on-chip test generation process.

A. The Primary Input Sequence A

The simplest way to generate a primary input sequence A on-chip is to use a random source such as an *LFSR*. However, a random sequence A may bring the circuit from the initial state s_r into a limited set of reachable states. This can be explained by the effect observed in [18] and referred to as repeated synchronization. According to [18], a primary input cube c synchronizes a subset of state variables $S(c)$ if the following conditions are satisfied. Let c be applied to the primary inputs when the circuit is in the all-unspecified present-state. Suppose that this results in a next-state s. The state variables whose values are specified in s are included in $S(c)$.

In the example of $s27$ shown in Figure 1, the primary input cube $I_0I_1I_2I_3 = 0$xxx applied in present-state $y_0y_1y_2 =$ xxx results in the next-state $Y_0Y_1Y_2 = 0$xx, synchronizing state variable y_0. In addition, the primary input cube $I_0I_1I_2I_3 =$ xx1x applied in present-state $y_0y_1y_2 =$ xxx results in the next-state $Y_0Y_1Y_2 =$ xx0, synchronizing state variable y_2.

A primary input cube c with a small number of specified values is likely to appear repeatedly in a random primary input sequence A. When this happens, the state variables in $S(c)$ assume the same values repeatedly under A. This may prevent the circuit from entering certain reachable states, and limit the ability of the functional broadside tests extracted from A to detect target faults.

Repeated synchronization was avoided in [18] by using a software procedure. Here, we need a process that will be implemented using on-chip hardware. To design such hardware, we first use a software procedure to compute primary input cubes that synchronize one or more state variables. The procedure focuses on primary input cubes with single specified values since such cubes are most likely to appear repeatedly in a random primary input sequence. For a circuit with n primary inputs, $I_0, I_1, \cdots, I_{n-1}$, it considers every primary input cube $c_{j,v}$ where primary input I_j assumes the value v, for $0 \leq j < n$ and $v \in \{0,1\}$, and the other primary inputs are unspecified. For $c_{j,v}$ it computes the set of synchronized state variables $S(c_{j,v})$. It then combines all the primary input cubes into a single cube $c = c(0)\ (1)\ \cdots\ (n-1)$ as follows. (1) If $|S(c_{j,0})| < |S(c_{j,1})|$, setting $I_j = 0$ causes fewer next-state variables to be specified. The procedure sets $c(j) = 0$. (2) If $|S(c_{j,0})| > |S(c_{j,1})|$, setting $I_j = 1$ causes fewer next-state variables to be specified. The procedure sets $c(j) = 1$. (3) If $|S(c_{j,0})| = |S(c_{j,1})|$, the procedure sets $c(j) =$ x.

In the example of $s27$ this procedure yields $S(c_{0,0}) = \{y_0\}$, $S(c_{2,1}) = \{y_2\}$, and $S(c_{j,a}) = \phi$ in all the other cases. Therefore, it yields $c =$ 1x0x.

We use c in the hardware generation of $A = a(0)\ a(1)\ \cdots\ a(L-1)$ as follows. A random source called RS_1 is used for generating a sequence of n-bit random primary input vectors. The length of the sequence is a constant L. A second random source called RS_2 is used for generating random numbers in the range $[0,2^p-1]$, for a constant p. At a time unit u where $RS_2 > 0$, $a(u)$ is modified such that it is equal to c for all the primary inputs where c is specified.

For illustration we consider the random primary input sequence for $s27$ shown in Table I under column RS_1. Column RS_2 shows the values produced by the second random source RS_2 using $p = 2$. We have $c =$ 1x0x for this circuit. Therefore, when $RS_2 > 0$ at time unit u, $a(u)$ is modified by assigning a 1 to input 0 and a 0 to input 2. The values of the other inputs do not change. The resulting primary input sequence A is the one shown under column $a(u)$ of Table I.

Considering a circuit with n primary inputs and a primary input cube $c = c(0)c(1)\cdots c(n-1)$, a single $(n+p)$-bit *LFSR* can be used for generating A. The logic required is shown in Figure 2. The *LFSR* is shown at the top of the figure. The value in cell j of the *LFSR* is denoted by $lfsr(j)$. The n leftmost cells of the *LFSR* are used for generating a random n-bit primary input sequence. The p rightmost cells of the *LFSR* are used for generating numbers in the range $[0,2^p-1]$. A p-input OR gate is used for generating a signal denoted by *mod*. When $mod = 0$, or if $c(j) =$ x, the value of primary input I_j is equal to $lsfr(j)$. When $mod = 1$, if $c(j) \neq$ x, the value of primary input I_j is equal to $c(j)$. Figure 2 shows three types of inputs. (1) For I_{j_1}, $c(j_1) =$ x. In this case, $I_{j_1} = lfsr(j_1)$. (2) For I_{j_2}, $c(j_2) = 0$. In this case, $I_{j_2} = mod' \cdot lfsr(j_2) + mod \cdot c(j_2) = mod' \cdot lfsr(j_2)$. (3) For I_{j_3}, $c(j_3) = 1$. In this case, $I_{j_3} = mod' \cdot lfsr(j_3) + mod \cdot c(j_3) = mod' \cdot lfsr(j_3) + mod = lfsr(j_3) + mod$.

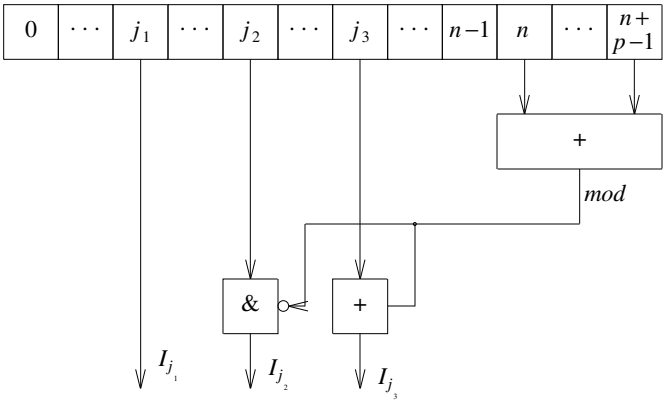

Figure 2. On-chip generation of A

In general, if there are N primary inputs I_j with $c(j) \neq$ x, the implementation of Figure 2 requires an $(n+p)$-bit *LFSR*, a p-input OR gate, N two-input AND or OR gates, and at most N inverters.

B. Test Selection

Next, we describe the selection of the functional broadside tests that will be applied based on a primary input sequence A. Let F be the set of target faults. In general, the goal is to select a subset of time units $U = \{u_0, u_1, \cdots, u_{k-1}\}$ and apply a test set $T(U) = \{t(u_i): u_i \in U\}$, where $t(u_i) = <s(u_i), a(u_i), a(u_i+1)>$, such that the following conditions are satisfied. (1) To ensure that the tests are non-overlapping, $u_i+1 < u_{i+1}$ for $0 \leq i < k-1$. (2) It should be possible to produce the subset U on-chip efficiently. (3) The test set $T(U)$ based on U should detect as many of the faults in F as possible. (4) U should be as small as possible.

The simplest way to satisfy the first two conditions is to include in U all the even or all the odd time units. We denote the resulting subsets of time units by U_{even} and U_{odd}, respectively. With an even L, $U_{even} = \{0, 2, 4, \cdots, L-2\}$ and $U_{odd} = \{1, 3, 5, \cdots, L-3\}$.

Test application based on U_{even} (U_{odd}) can proceed as follows. A counter denoted by $CNT = CNT(0)\ CNT(1) \cdots CNT(m-1)$, where $m = \log_2 L$, counts from 0 to $L-1$ to indicate the current time unit u of A. For even time units $CNT(m-1) = 0$, and for odd time units $CNT(m-1) = 1$. Thus, a single inverter or buffer producing the function $apply = CNT(m-1)'$ or $apply = CNT(m-1)$ is needed to identify even (odd) time units. When $apply = 1$, $t(u)$ is applied as a two-pattern test to the circuit. Application of $t(u)$ includes output response compaction of the primary output vector under the second pattern, and output response compaction of the final state of the test. The counter *CNT* is stopped during the scan-out operation of the final state until the same state is restored through circular shift as discussed in Section II.

With this implementation, a software procedure selects between U_{even} or U_{odd} based on the number of detected faults. To compute the numbers of detected faults, it sets $F_{targ} = F$, and performs fault simulation with fault dropping of F_{targ} under the set of tests $T(U_{even}) = \{t(0), t(2), \cdots, t(L-2)\}$. Let the set of detected faults be $D(U_{even})$. It then sets $F_{targ} = F$ again, and performs fault simulation with fault dropping of F_{targ} under $T(U_{odd}) = \{t(1), t(3), \cdots, t(L-3)\}$. Let the set of detected faults be $D(U_{odd})$. The procedure selects U_{even} if $|D(U_{even})| \geq |D(U_{odd})|$. It selects U_{odd} otherwise.

In the example of $s27$ with the primary input sequence shown in Table I, we consider the set F of transition faults that consists of 52 faults denoted by $f_0, f_1, \cdots, f_{51}$. Fault simulation of $T(U_{even})$ yields the sets of detected faults shown in the left part of Table II. For every even time unit u such that $t(u)$ detects any faults, Table II shows the faults added to $D(U_{even})$ after $t(u)$ is simulated. Fault simulation of $T(U_{odd})$ yields the subsets of detected faults shown in the right part of Table II. We obtain $|D(U_{even})| = 19$ and $|D(U_{odd})| = 13$. Therefore, we select to use the even time units for test application.

TABLE II. Detected faults of $s27$

u	$D(U_{even})$	u	$D(U_{odd})$
0	$f_7 f_{37} f_{38} f_{41} f_{43}$ $f_{45} f_{47} f_{48} f_{50}$	1	$f_2 f_{17} f_{19} f_{28}$
4	$f_1 f_4 f_{14} f_{22} f_{29} f_{49}$	5	$f_0 f_5 f_{15} f_{23} f_{48}$
8	$f_2 f_{17} f_{19} f_{28}$	7	$f_{13} f_{16} f_{18}$
		9	f_{12}

For a sequence of length L, the number of time units in U_{even} is $L/2$ and the number of time units in U_{odd} is $L/2-1$. Suppose that the set U (which may be U_{even} or U_{odd}) is selected. With $L/2$ or $L/2-1$ time units in U, the same number of tests are included in $T(U)$ and applied to the circuit. However, some of the tests in $T(U)$ may not be necessary for detecting target faults. Thus, fewer tests may be applied to achieve the same fault coverage.

In the example of $s27$ we selected $U_{even} = \{0,2, \cdots, 14\}$. However, only tests $t(0)$, $t(4)$ and $t(8)$ are needed for detecting the 19 faults detected based on U_{even}. With $L = 16$ for this circuit, the hardware for on-chip test generation includes a 4-bit counter $CNT = CNT(0)\ CNT(1)\ CNT(2)\ CNT(3)$. The states of *CNT* corresponding to time units 0, 4 and 8 are 0000, 0100 and 1000. By using the function $apply = CNT(2)'CNT(3)'$ instead of $apply = CNT(3)'$ to indicate the time units where tests should be applied to the circuit, the number of applied tests can be reduced from eight to four. The four applied tests correspond to $U = \{0, 4, 8, 12\}$.

In general, it is possible to start from $apply = CNT(m-1)$ or $apply = CNT(m-1)'$, and recompute the function *apply* such that it would cover fewer time units. As long as $CNT(m-1)$ or $CNT(m-1)'$ is ANDed with other terms, only even or only odd time units will be used, and non-overlapping tests will be applied. We use the following software procedure to reduce the number of tests that the hardware will apply based on a sequence A.

The procedure focuses on functions *apply* of the form $CNT(0)^*\ CNT(1)^* \cdots CNT(m-1)^*$, where $CNT(i)^*$ is either a literal of $CNT(i)$ (equal to $CNT(i)$ or $CNT(i)'$), or $CNT(i)$ does not appear in the function. The procedure selects the function *apply* for A as follows.

Let U be the set of even or odd time units selected for test application based on A. When F_{targ} is simulated under $T(U)$ with fault dropping, if $t(u) \in T(U)$ detects new faults, the procedure includes u in a set U_{eff}. It then considers the m-bit binary representations of the time units in U_{eff}. For every bit position i, it considers the following cases. (1) If all the time units in U_{eff} have a 0 in bit i, it sets $CNT(i)^* = CNT(i)'$. (2) If all the time units in U_{eff} have a 1 in bit i, it sets $CNT(i)^* = CNT(i)$. (3) If

the time units in U_{eff} have both 0s and 1s in bit i, $CNT(i)$ does not appear in *apply*.

In the example of $s27$ with the test sequence of Table I we selected to use U_{even}. Based on Table II we obtain $U_{eff} = \{0, 4, 8\}$, with binary representations 0000, 0100 and 1000. We obtain $apply = CNT(2)'CNT(3)'$.

In general, a single m-input AND gate is needed to implement the function *apply*.

C. Overall Process

Until now we considered a single primary input sequence A of length L. The on-chip test generation hardware uses a constant p to determine how a random sequence will be modified into A to avoid repeated synchronization, and a function *apply* to determine which tests will be applied. A software procedure selects the tests to be applied by first selecting even or odd time units. We use a variable denoted by *sel* to indicate which time units are selected, where $sel = 0$ for even time units, and $sel = 1$ for odd time units. The procedure then selects the function *apply* to reduce the number of tests applied based on A. In addition we associate with A a seed, denoted by *seed*, which is used for initializing the random sources (the *LFSR* in Figure 2) before starting the generation of A.

However, a single sequence with a single setting of the parameters *seed*, L, p and *sel*, and a single function *apply*, may not be sufficient for achieving the highest possible fault coverage. Therefore, we consider several primary input sequences A_0, A_1, $\cdots$. Initially, every sequence A_i is associated with a seed denoted by $seed_i$, a length denoted by L_i, and a value of *sel* denoted by sel_i. We use a constant value for p since we found that a single value is sufficient for achieving high fault coverage. This allows the same logic, shown in Figure 2, to be used for all the sequences.

The software procedure used for selecting the parameters for the various sequences is based on the following observations.

A larger test sequence length L allows a higher fault coverage to be achieved if certain reachable states are entered only at the end of a longer sequence. A larger L also allows fewer sequences to be used since more tests can be applied based on each sequence. However, for the same fault coverage, a larger L results in more applied tests. This is related to the fact that with fewer sequences and more applied tests based on each one of them, there is less flexibility in selecting the functions $apply_i$ so as to reduce the number of applied tests.

To accommodate these observations, the procedure considers increasing values of L. For constants L_{low} and L_{high}, it computes sequences of lengths $L = L_{low}$, $2L_{low}$, $\cdots$, L_{high}. For every value of L it obtains several different sequences, using different seeds. For every sequence A_i it selects a value for sel_i to determine whether even or odd time units will be used for test application (the computation of $apply_i$ is postponed until after all the sequences have been selected). It performs fault simulation of the tests based on A_i and removes the detected faults from the set of target faults F. It continues to generate sequences of length L until the last Q sequences generated do not improve the fault coverage, for a constant Q. It keeps only the sequences that improve the fault coverage and discards the others.

Let the selected primary input sequences be A_0, A_1, $\cdots$, A_{K-1}. It is possible that all the faults detected based on a sequence A_i will also be detected by the sequences A_{i+1}, $\cdots$, A_{K-1}. In this case, A_i does not need to be implemented as part of the on-chip test generation process. The procedure identifies and removes such sequences after every value of L is considered, as follows.

During the selection of the sequences, if a fault $f \in F$ is detected by a test based on a sequence A_i, the procedure sets $det(f) = i$. It also sets $valid_i = 1$ for $0 \le i < K$. For every sequence A_i, where $i = 0, 1, \cdots, K-1$, it checks whether all the faults detected by tests based on A_i are also detected by tests based on sequences A_j such that $j \ne i$ and $valid_j = 1$. This requires fault simulation of every fault f such that $det(f) = i$ under sequences A_j such that $j \ne i$ and $valid_j = 1$. If all the faults with $det(f) = i$ are detected based on other sequences, it updates the variable $det(f)$ for every such fault to reflect the other sequence based on which f is detected. It also sets $valid_i = 0$. At the end of this process it removes every sequence A_i for which $valid_i = 0$.

Let the remaining sequences at the end of this process be A_0, A_1, $\cdots$, A_{M-1}. Considering the time units of A_i whose tests increase the fault coverage, the procedure selects the function $apply_i$ as described at the end of the previous subsection, for $0 \le i < M$.

The control logic required for applying tests based on M primary input sequences consists of a counter and multiplexers to select the seed, the sequence length, and the function $apply_i$. Similar control logic is required for other types of built-in self-test solutions [17]. All the sequences share the hardware shown in Figure 2. In addition, each sequence requires a single AND gate for the function $apply_i$.

IV. EXPERIMENTAL RESULTS

We considered ISCAS-89 and ITC-99 benchmark circuits that are testable using functional tests. We excluded from consideration circuits such as $s9234$, $s13207$ and $s38417$, which are non-synchronizable, and for which the fault coverage achievable by functional tests is very low. Practical circuits are expected to be synchronizable, and testable by functional tests. We assume that the initial state of the circuit is the all-0 state. We use tran-

sition faults as the target fault model. We used the following parameters in a software simulation of the on-chip test generation approach.

The lengths of the primary input sequences are determined by $L_{low} = 32$ and $L_{high} = 1024$. As discussed earlier, L_{low} results in the selection of small numbers of tests based on each sequence. Higher values of L are useful for bringing the circuit into more reachable states and detecting harder-to-detect faults.

For all the sequences, $2^p = 8$. We found that using different values for different sequences does not improve the fault coverage compared to this single value.

The highest number of specified values in the cube c used for avoiding repeated synchronization in any circuit was six. Thus, at most six two-input gates are required in Figure 2.

TABLE III. On-chip test generation

		tests			
circuit	seq	app	eff	f.c.	[13]
s641	38	508	124	81.02	81.02
s1423	46	2659	234	83.27	76.70
s5378	96	2108	383	73.50	72.25
s35932	23	415	231	87.21	87.19
s38584	312	13524	1755	66.30	52.05
b04	34	326	160	84.54	84.46
b08	24	374	88	81.87	68.96
b09	16	183	53	77.29	70.50
b10	22	240	102	81.72	81.38
b11	27	540	143	75.79	72.84
b14	179	7213	840	77.12	72.73
b20	407	8786	1873	81.97	65.00

The results are shown in Table III as follows. Column *seq* shows the number of primary input sequences selected. Column *tests* subcolumn *app* shows the number of tests applied based on the selected sequences, and subcolumn eff shows the number of tests that improve the fault coverage when they are applied. Some applied tests do not increase the fault coverage since the functions $apply_i$ do not exclude all the tests that do not increase the fault coverage.

Column $f.c.$ shows the transition fault coverage achieved by the tests applied on-chip. For comparison, column [13] shows the transition fault coverage reported in [13] where functional broadside tests are produced externally and the initial state is the all-0 state.

From Table III it can be seen that, for all of the circuits considered, the on-chip generation of functional broadside tests achieves a transition fault coverage that is the same as or higher than that reported in [13].

The number of primary input sequences and the number of applied tests are manageable given the simplicity of the hardware. The number of sequences can be reduced by using a higher value for L_{low} in order to avoid the use of a large number of short sequences. The number of applied tests can then be reduced by using different types of functions $apply_i$ to select the applied tests for each sequence.

V. CONCLUDING REMARKS

This paper described an on-chip test generation method for functional broadside tests. The hardware was based on application of primary input sequences in order to allow the circuit to produce reachable states. Random primary input sequences were modified to avoid repeated synchronization and thus yield varied sets of reachable states. Two-pattern tests were obtained by using pairs of consecutive time units of the primary input sequences. The on-chip generation of functional broadside tests required simple hardware and achieved high transition fault coverage for testable circuits.

REFERENCES

[1] J. Rearick, "Too Much Delay Fault Coverage is a Bad Thing", in Proc. Intl. Test Conf., 2001, pp. 624-633.

[2] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash and M. Hachinger, "A Case Study of IR-Drop in Structured At-Speed Testing", in Proc. Intl. Test Conf., 2003, pp. 1098-1104.

[3] S. Sde-Paz and E. Salomon, "Frequency and Power Correlation between At-Speed Scan and Functional Tests", in Proc. Intl. Test Conf., 2008, Paper 13.3, pp. 1-9.

[4] J. Savir and S. Patil, "Broad-Side Delay Test", IEEE Trans. on Computer-Aided Design, Aug. 1994, pp. 1057-1064.

[5] I. Pomeranz, "On the Generation of Scan-Based Test Sets with Reachable States for Testing under Functional Operation Conditions", in Proc. Design Autom. Conf., 2004, pp. 928-933.

[6] Y.-C. Lin, F. Lu, K. Yang and K.-T. Cheng, "Constraint Extraction for Pseudo-Functional Scan-Based Delay Testing", in Proc. Asia and South Pacific Design Autom. Conf., 2005, pp. 166-171.

[7] Z. Zhang, S. M. Reddy and I. Pomeranz, "On Generating Pseudo-Functional Delay Fault Tests for Scan Designs", in Proc. Intl. Symp. on Defect and Fault Tolerance in VLSI Systems, 2005, pp. 398-405.

[8] I. Polian and F. Fujiwara, "Functional Constraints vs. Test Compression in Scan-Based Delay Testing", in Proc. Design, Autom. and Test in Europe Conf., 2006, pp. 1-6.

[9] I. Pomeranz and S. M. Reddy, "Generation of Functional Broadside Tests for Transition Faults", IEEE Trans. on Computer-Aided Design, Oct. 2006, pp. 2207-2218.

[10] M. Syal, et. al., "A Study of Implication Based Pseudo Functional Testing", in Proc. Intl. Test Conf., 2006.

[11] A. Jas, Y.-S. Chan and Y.-S. Chang, "An Approach to Minimizing Functional Constraints", in Proc. Defect and Fault Tolerance in VLSI Systems, 2006, pp. 215-226.

[12] H. Lee, I. Pomeranz and S. M. Reddy, "On Complete Functional Broadside Tests for Transition Faults", IEEE Transactions on Computer-Aided Design, March 2008, pp. 583-587.

[13] I. Pomeranz and S. M. Reddy, "On Reset Based Functional Broadside Tests", in Proc. Design Autom. and Test in Europe Conf., 2010, pp. 1438-1443.

[14] H. Lee, I. Pomeranz and S. M. Reddy, "Scan BIST Targeting Transition Faults Using a Markov Source", in Proc. Intl. Symp. on Quality Electronic Design, 2004, pp. 497-502.

[15] V. Gherman, H.-J. Wunderlich, J. Schloeffel and M. Garbers, "Deterministic Logic BIST for Transition Fault Testing", in Proc. European Test Symp., 2006, pp. 123-130.

[16] Y. C. Lin, F. Lu and K.-T. Cheng, "Pseudo-Functional Scan-Based BIST for Delay Fault", in Proc. VLSI Test Symp., 2005, pp. 229-234.

[17] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1995.

[18] I. Pomeranz and S. M. Reddy, "Primary Input Vectors to Avoid in Random Test Sequences for Synchronous Sequential Circuits", IEEE Trans. on Computer-Aided Design, Jan. 2008, pp. 193-197.