Generating Efficient Tests for Continuous Scan

Sying-Jyan Wang and Sheng-Nan Chiou
Institute of Computer Science
National Chung-Hsing University
Taichung 402, Taiwan, ROC
+886-4-2840497
{sjwang, kboy}@cs.nchu.edu.tw

ABSTRACT
Conventional scan-based designs spend a lot of testing time in shifting test patterns and output responses, which greatly increases the testing cost. In this paper, we propose a modified approach for scan-based design in which a test is conducted in every clock cycle. This approach may significantly reduce the test application time when appropriate test vectors are applied. We develop algorithms to generate efficient test input for the test environment, and experimental results show that we can achieve high fault coverage with only about 10%-30% of the clock cycles required in conventional scan-based design.

Keywords
Scan, DFT, Test Generation, Compression.

1. INTRODUCTION
Scan-based design is a structural DFT that is widely used in industry. In a scan-based design, all or some of the registers in the circuit under test (CUT) are replaced by scan registers. Test generation becomes much easier in scan-based design. However, the overall test application time is not necessarily shorter, since we have to shift the vector into a serial scan path before a test is applied. The time for shifting can be enormous due to the larger number of internal flip-flops in a modern VLSI.

Many efforts have been devoted to reduce the test application time in scan-based design. Test application time in scan design is mainly affected by the number of shift operations, which depends on the number of required test vectors as well as the number of flip-flops in the scan chain. Therefore, test application time can be smaller by reducing the number of required test vectors, the number of shift operations, and the number of flip-flops in a scan chain. The size of a test vector set can be smaller by performing dynamic or static compaction [1]-[2] to merge compatible test vectors. Some shift operations can be eliminated if consecutive test vectors have patterns in common [3]. The number of scan flip-flops can be reduced with partial scan [4] or multiple scan chain [5]-[6].

We try to reduce test application time in scan-based design from a different perspective. Conventional scan-based test environment adopts a test-per-scan approach [7], in which a scan cycle is the number of clock cycles required to shift the vector into a serial scan chain or to shift the response out of the scan chain (whichever is larger), plus one or more normal mode clocks. Since a new bit is shifted into the scan chain every clock cycle, actually there is a new vector in each cycle. In test-per-scan approach, we cannot apply such intermediate vectors because an output response has to be scanned out first. However, if multiple input signature registers (MISR) are used to compress output response of a CUT (e.g. [6]), it will be possible to apply test-per-clock [7] strategy. We shall refer to this approach as the *continuous scan* henceforth. Since a test is conducted in every *clock* cycle instead of a *scan* cycle, it is possible to greatly reduce the number of the test application time, if the input sequence is carefully selected.

In this paper, we present algorithms to generate efficient test set for continuous scan. We apply our methods to some ISCAS'85 benchmark, and the results are promising. Experimental results show that our methods achieve a high fault coverage while at the same time reduce 70%-90% of the clock cycles used for scan.

2. PRELIMINARIES
In a scan-based design, an input or output to a CUT is a scan register, and all the scan registers are connected as a scan chain. In normal mode, the inputs are parallel loaded into scan register and passed to the module. In test mode, test vectors are shifted in from scan-in port and the output vector is shifted out of the scan-out port after the results are stored in the registers. We can check if specific faults exist by observing the vector scanned out of the chip, after shifting the next test vector into the scan chain.

In scan-based test environment, if the test vector is n-bit long, we need n cycles to scan in the test vector and then run the remaining test steps. For large n, the time for scanning is prohibitive.

Multiple input signature registers (MISR) are designed for signature analysis, which is a technique for data compression. MISRs efficiently map different input streams to different signatures with a very small probability of alias. MISRs are frequently implemented in built-in-self-test (BIST) designs, in which output responses are compressed by MISRs. In the test mode, a MISR accepts the test results from a module under test. If the module is faulty, the final signature stored in the MISR will be different from the signature generated from a faulty module. In this way, we can distinguish if the module under test is faulty.

3. CONTINUOUS SCAN

Our approach tries to make best use of any test patterns that present in the scan chain. In the conventional scan test, after one test pattern having been applied, we need to shift a new pattern into the scan chain. Whenever one bit is shifted into the scan chain, a new input pattern is formed. These intermediate patterns may be useful to detect faults in the module under test; however, such intermediate test patterns cannot be exploited under traditional scan-based test environment.

Our method works as follows. After the scan chain having shifted in one bit, the new vector is applied to the module under test. However, there must be a way to observe the output response to decide if the results are correct. To solve the problem, we need to make some modification to the conventional scan-base test environment. In the output side, it is easy to use a MISR to replace the scan chain. A similar architecture has been presented in [6].

The proposed test environment consists of two parts: the input part is a scan chain and the output part is a MISR. In the input part, the scan chain shifts one bit in every clock cycle to generate a new test pattern for the module under test. For primary inputs that are not fed by scan chain, the input vectors should be applied in every cycle. The outputs are sent to the MISR for analysis. An example of the test process is illustrated in Figure 1. After all test patterns having been applied, the content of the MISR is scanned out to check if the module is faulty.

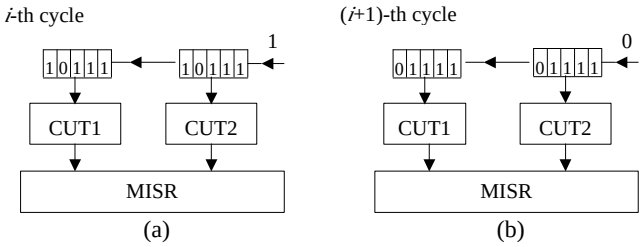

Figure 1. An example of continuous scan.

To find an efficient test under this environment, the goal is to find the shortest test sequence, not the smallest test vector set. A simple test generation process may work as follows. First, the test vector set for the CUT has to be generated. Then, some algorithms are used to merge the test vector set to find out the test sequence. This process achieves at least the same fault coverage as provided by conventional scan-based test, while the test time is much less than the classical method.

4. TEST GENERATION

In this section, we present our approach for test generation in continuous scan. We shall first discuss techniques used to compress a set of test vectors into a compact sequence. The test generation process is presented at the end of this section.

4.1 Compression -- Deterministic Vectors

When a combinational test vector has been generated for a given fault, normally there are some unspecified bits (or *don't-care* bits) in the vector. These don't-care bits are then assigned with 0's and 1's. We shall refer to these vectors as *deterministic* vectors. Many compression algorithms for deterministic vectors can be found in the literature [3]. What a compression algorithm does is to put the test vectors in a list. Two vectors are adjacent in the list if they are closely related. The relation is defined as number of shift operations that can be eliminated should the two test vectors are adjacent. For example, let vector V_1 be 110101, and V_2 be 010111. A complete scan process requires twelve shift operations for both vectors. However, when V_1 has been placed in the scan chain, we only need to scan in two more bits, namely 11. The reason is that the bit-pattern '0101' is a suffix of V_1 and a prefix of V_2. Thus, V_1 and V_2 can be overlapped to form a single test sequence 11010111, whose length is eight. Therefore, four shift operations are saved.

In the above example, actually two bit-patterns can be both a suffix of V1 and a prefix of V2 simultaneously, namely '01 'and'0101'. Since the goal is to minimize the length of final test sequence, we should always select the maximal bit-pattern. In this example, '0101' is maximal, while '01' is not.

Definition 1: Let S_1 and S_2 be two strings of bits. The *overlapped bit-pattern* from S_1 to S_2 is the maximal bit-pattern that is both a suffix of S_1 and a prefix of S_2.

Definition 2: Let S_1 and S_2 be two strings of bits. The *identical overlap* from S_1 to S_2, which is denoted as $O(S_1, S_2)$, is the length of the overlapped bit-pattern from S_1 to S_2.

All the overlaps among the vectors in a test set can be represented by a complete direct graph $G(V, E, W)$, where V, E, W are the set of vertices, edges, and weights on edges, respectively. A node in V represents a vector in the test set. An edge (V_i, V_j) implies vector V_i followed by V_j, so the weight on the edge is $O(V_i, V_j)$. Figure 2 shows a vector set and its graph representation.

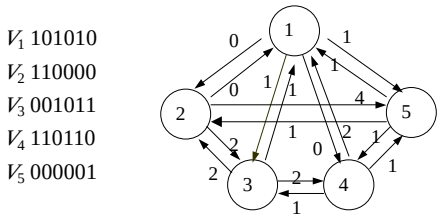

Figure 2. The graph representation of a test vector set.

The problem of finding the shortest test sequence is thus reduced to searching for a maximum-cost Hamiltonian path. Such a problem can be solved with Kruskal's algorithm, but a depth-first greedy algorithm usually works well in most case [3].

4.2 Compression Algorithm I

It is easier to compress a set of incompletely specified vectors (i.e., those with don't-care bits) than deterministic vectors. For example, let V_1 be 1101x1 and V_2 be 0x0111, and we try to concatenate V_2 to V_1. Since a don't-care bit can be either a 0 or a 1, the best way to compress them is to set V_1 = 110101 and V_2 = 010111. In this case, the two vectors can be merged into a sequence of length 8. All other assignments create two vectors that cannot be merged, and thus the length of final sequence will be 12.

On the other hand, for a given circuit the size of a test set with deterministic test vectors is smaller than the set of vectors with don't-care bits. The reason is that the number of faults detected by an incompletely specified vector is definitely smaller than that achieved by the same vector with don't-care bits assigned.

Definition 3: Consider two l-bit strings $A(=a_{l-1}\ldots\ a_1a_0)$ and $B(=b_{l-1}\ldots\ b_1b_0)$, where $a_i, b_i \in \{0,1,x\}$ for all $l-1 \geq i \geq 0$. Strings A and B are said to be *compatible* if, for all $l-1 \geq i \geq 0$, one of the following is true: (1) $a_i = b_i$, (2) a_i = x, or (3) b_i = x.

Definition 4: Let S_1 and S_2 be two strings of bits. The *overlapped compatible pattern* from S_1 to S_2 is the maximal pattern that is compatible to a suffix of S_1 and a prefix of S_2.

Definition 5: Let S_1 and S_2 be two strings of bits. The *compatible overlap* from S_1 to S_2, which will be denoted as $O_C(S_1, S_2)$, is the length of the overlapped compatible pattern from S_1 to S_2.

Compression algorithms for deterministic vectors are not directly applicable here. The reason is that the compatibility will be changed if two strings are merged. Consider the set of incompletely specified vectors and its graphic representation shown in Figure 3. The weight on edge (V_i, V_j) is the compatible overlap $O_C(V_i, V_j)$. If we apply the depth-first greedy algorithm, the compressing order will be $\{V_2, V_3, V_1\}$. The process works as follows. First V_2, V_3 are merged, and the resulting sequence is 101011. Now the compatible overlap from the new sequence (i.e., 101011) to V_1 is 2 instead of 4, which is indicated in the graph. A better compressing order is $\{V_1, V_2, V_3\}$.

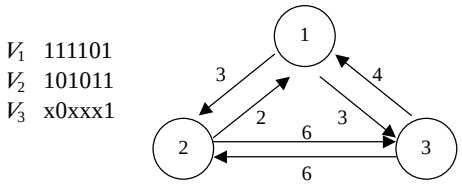

Figure 3. Graph representation of a set with don't-care bits.

Our first algorithm for the compression of vectors with don't-care bits is a modified greedy algorithm in which two vectors are merged first if they have a larger compatible overlap.

Algorithm 1: Greedy Search
Input: Test pattern set: T($T = \{T_1, T_2,\ldots,T_n\}$)
Output: Final test sequence: TS
Step I: 1. Randomly select a test pattern from T and assign it to TS. 2. $T = T - \{TS\}$ **Step II:** 1. Compute all $O_C(TS, T_i)$, $T_i \in T$, i = 1,2,…,n 2. Find the maximal $O_C(TS, T_j)$ 3. $TS \leftarrow$merge(TS, T_j) 4. $T = T - \{T_j\}$ **Step III:** Repeat Step II until $T = \phi$.

The complexity of the above algorithm is $O(n^2)$.

4.3 Compression Algorithm II

It is difficult to decide the optimal compressing order for a set of vectors with don't-care bits, as the compatible overlap between two vectors may change during the compressing process. A greedy algorithm may not work well in this case, as illustrated in Sec 4.2.

We propose a simple heuristic in this section. In the merging process, the test vectors with fewer don't-care bits should be processed earlier. The reason is that a vector with more don't-care bits can be efficiently merged with other vectors, but this advantage no longer exists once it is processed. The compression algorithm is outlined as follows.

Algorithm 2: Ordering according to weight
Input: A sequence of n test patterns ($T_1, T_2,\ldots, T_n$).
Output: A permutation (reordering) ($T_1, T_2,\ldots, T_n$) of the input sequence such that $W_1 \leq W_2 \leq \ldots \leq W_n$
1. Sorting the test patterns to their number of don't-care bits. T_i : the i-th test pattern W_i: the number of don't-care bits in T_i 2. Merging the test patterns ($T_1, T_2,\ldots, T_n$) in sequential order.

The complexity of this heuristic is $O(n\log n)$, since the major operations is the sorting process in Step 1.

4.4 Test Generation Process

A naive way to generate a test sequence for continuous scan can be done as follows. First, we generate a combinational test vector set; and then apply a compressing algorithm to get the final test sequence. However, the test sequence obtained from this process is far from optimum. The reason is that many test vectors in the test set are not necessary in continuous scan. Consider a vector set $\{V_1, V_2, \ldots, V_n\}$, in which each vector is of length l. Assume that the set can not be compressed for simplicity. To scan in the n vectors, we need to apply $l \times n$ clock cycles. Therefore, in all there will be $l \times n - l + 1$ test vectors in the scan chain sequentially under test-per-clock strategy. In other words, there are $(n-1)\times(l-1)$ new vectors, which are also useful in fault detection. As a result, many test vectors in the original test set are no longer necessary.

We propose a two-step test generation process for continuous scan. In the first step, we generate a set of test vectors, in which 10% of the vectors are selected. The selected vectors are compressed, and fault simulation is conducted on a test-per-clock basis. Most detectable faults will be marked as detected at this stage. In the second step, test vectors are generated for the remaining faults, and a compression algorithm is conducted again to get the final test sequence.

Since the test set used in the first step is relatively small, and in the second step we only generate tests for undetected faults, it is not likely that there are unnecessary vectors in the scan chain at any time. As a result, we can get a short test sequence. Besides, since the test sets in both steps are relatively small, the time required for the compression algorithm can be greatly reduced.

Procedure: Generate test sequence for continuous scan

1. Conduct ATPG to generate a set of **completely specified** test patterns. Select 10% of the test patterns and compress them.
2. Conduct fault simulation with the sequence obtained in Step 1 and remove all detected faults from the fault list.
3. Conduct ATPG again for the remaining faults to obtain a set of **incompletely specified** test vectors.
4. Compress vectors obtained in Step 3 with Algorithm 1 or 2.
5. Concatenate the sequences obtained in Step 2 and Step 4.

5. EXPERIMENTAL RESULTS

We experiment the method discussed in Section 4 with 11 ISCAS'85 benchmark circuits. These are the larger circuits in the benchmark suite. The statistics of the benchmarks are given in Table I. The circuit named M actually merges the first eight circuits (from C432 to C6288) into a single module, and a single input is used to support all scan chains [5]. In the Table, #PI/PO denotes the number of primary inputs and outputs, and #TP is the number of test patterns required under conventional scan test environment. The column under TPS indicates the number of clock cycles required for shift operations under test-per-scan approach. The columns under FC and TE indicate the fault coverage and test efficiency for each benchmark circuits. It can be seen that all non-redundant faults are detectable.

Table I: Benchmark Circuit Statistics

Circuits	#PI/PO	#Faults	#TP	TPS	FC(%)	TE(%)
C432	36/7	590	96	3456	93.22	100
C499	41/32	1238	119	4897	99.35	100
C880	60/26	1000	65	3900	100.00	100
C1355	41/32	1574	149	6109	99.49	100
C1908	33/25	1914	163	5379	99.58	100
C2670	233/140	2711	146	34018	95.76	100
C3540	50/22	3460	198	9900	96.18	100
C6288	32/32	7776	35	1120	99.56	100
M	233/439	25135	475	110675	98.38	100
Frg2	143/139	4550	209	29887	91.58	100
Apex6	135/99	1900	123	16605	100.00	100
Rot	135/107	2460	297	40095	95.53	100

We use a PODEM-based ATPG program, and both compression algorithms 1 and 2 are experimented. The results are given in Table II. The column under DFG gives the test sequence length for each circuit if depth-first search is used to compress deterministic test patterns. The columns under A1 and A2 indicate the number of clock cycles used in our method with compression algorithms 1 and 2, respectively. The fault coverage (FC) and test efficiency (TE) are shown again here for clarity. The last three columns compare the number of clock cycles required in each method to the number of clock cycles used in test-per-scan approach. It can be seen that in most cases the ratio is less than 30% if our methods are used.

6. CONCLUSION AND FUTURE WORK

In this paper we present a test-per-clock approach for scan test environment, in which each clock cycle a new scan vector is formed in the scan chain. We propose a test generation framework, which requires scan test sequences that are much shorter than those generated by conventional approach. As a result, the test application time can be greatly reduced. We have presented two compression algorithms for vectors with don't-care bits. Our experimental results show that these algorithms work well in practice.

The scan test methodology presented in this paper is designed for combinational circuit, and it is not directly applicable to sequential circuits in the present form. We are currently studying how to apply test-per-clock approach for sequential circuits, and try to develop efficient test generation algorithm for the new test framework.

7. REFERENCES

[1] B. Ayari and B. Kamin, "A new dynamic test vector compaction for automatic test pattern generation," *IEEE Trans. CAD*, vol. 13, no. 3, pp. 353-358, Mar. 1994.

[2] J.-S. Chang and C.-S. Lin, "Test set compaction for combinational circuits," *IEEE Trans. CAD*, vol. 14, no. 11, pp. 1370-1378, Nov. 1995.

[3] C. Su and K. Huang, "A serial scan test vector compression methodology," in *Proc. Intl. Test Conf.*, pp. 981-988, 1993.

[4] K. T. Cheng and V. D. Agrwal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. Comput.*, vol. 39, pp. 544-548, Apr. 1990.

[5] S. Lee and K.G. Shin, "Design for test using partial parallel scan," *IEEE Trans. Comput.*, vol. 39, pp. 203-211, 1990.

[6] K.-J. Lee , J.-J. Chen, and C.-H. Huang, "Using a single input to support multiple scan chains," in *Proc. ICCAD*, pp. 74-78, 1998.

[7] V.D. Agrawal, C.R. Kime, and K.K. Saluja, "A tutorial on built-in self-test, part 2: applications," *IEEE D&T Computers*, 69-77, June 1993.

Table II: Test generation results

Name	DFG	A1	A2	FC (%)	TE (%)	DFG/TPS (%)	A1/TPS (%)	A2/TPS (%)
C432	2908	743	632	93.22	100	84.14	21.50	18.29
C499	4611	1343	1343	99.35	100	94.16	27.42	27.42
C880	3662	1266	1267	100.0	100	93.90	32.46	32.48
C1355	5566	1712	1697	99.49	100	91.17	28.02	27.78
C1908	4546	1399	2203	99.58	100	84.51	26.01	40.96
C2670	33209	11373	9974	95.76	100	97.62	33.43	29.32
C3540	8687	1529	1562	96.18	100	87.75	15.44	15.78
C6288	805	128	128	99.56	100	71.88	11.43	11.43
M	107149	21400	18467	98.38	100	96.81	19.34	16.69
Frg2	28559	5534	5186	91.58	100	95.56	18.52	17.35
Apex6	15915	4120	3791	100.0	100	95.84	24.81	22.83
Rot	37971	6496	6126	95.53	100	94.70	16.20	15.28