# Bottleneck Removal Algorithm for Dynamic Compaction and Test Cycles Reduction

Srimat T. Chakradhar
C& C Research Laboratories
NEC USA, Princeton, NJ 08540, USA

Anand Raghunathan
Department of Electrical Engineering
Princeton University, Princeton, NJ 08544, USA

**ABSTRACT: We present a new, dynamic algorithm for test sequence compaction and test cycle reduction for combinational and sequential circuits. Several dynamic algorithms for compaction in combinational circuits have been proposed but, to the best of our knowledge, no dynamic method has been reported in the literature for compaction in non scan sequential circuits. Our algorithm is based on two key ideas: (1) we first identify bottlenecks that prevent vector compaction and test cycle reduction for test sequences generated thus far, and (2) future test sequences are generated with an attempt to eliminate bottlenecks of earlier generated test sequences. If all bottlenecks of a sequence are eliminated, then the sequence is dropped from the test set. The final test set generated by our algorithm is minimal in the following sense. Static vector compaction or test cycle reduction using set-covering or extended set-covering approaches (for example, reverse or any other order of fault simulation, with any specification of unspecified inputs in test sequences) cannot further reduce the number of vectors. Experimental results on scan and non scan sequential circuits are reported to demonstrate the effectiveness of our algorithm.**

## 1. INTRODUCTION

Reduction in test application time and test set size is highly desirable to reduce the overall costs incurred in fabricating and testing a large number of chips that implement a specific design. Small test sets are desirable because testers have a fixed memory size. The application of a test set larger than the tester memory size will require repeated loading of the tester memory, which is an expensive process. Hardware modifications like full or partial scan that are employed to ease the task of test generation *significantly* increase the test application time. This is mainly due to the extra tester clock cycles needed to load specific logic values into scan flip-flops (FFs) and observe circuit responses collected in the scan FFs.

For full scan design circuits, the number of tester clock cycles required to apply a given test set is roughly proportional to the product of the number of vectors in the test set and the number of scan FFs in the design. The amount of tester memory required to store a test set is proportional to the number of vectors in the test set, and the number of bits per vector. In order to reduce both test application time and meet tester memory requirements, several combinational test generators aimed at generating test sets that contain fewer vectors have been developed. These methods can be classified as static or dynamic. Static methods attempt to reduce the number of vectors in an already generated test set [1, 2, 3]. Dynamic methods consider vector compaction during the generation of the test set [4, 5]. Other methods to reduce test application time design the scan path so as to reduce the number of cycles required to scan-in a vector or scan out the circuit response [6, 7].

Hybrid approaches employing both combinational and sequential test generation methods have been investigated to reduce the test application time for full scan designs [8, 9]. The test set generated by these methods consists of vectors that have to be scanned in, and vectors that do not require any scan-in. A recent technique recognized that full scan-in of the vector or full scan out of the circuit response may be an overkill [10]. They propose a static method of test cycle reduction based on the partial scan-in and scan-out of test vectors and circuit response, respectively. They also re-order the FFs in the scan chain to reduce the test application cycles required for a given test set.

For sequential circuits with little or no scan, static methods have been proposed to reduce the size of the test set or test application cycles [11]. Recently, dynamic methods for reducing test application cycles in partial scan design circuits have been suggested [9, 12]. However, to the best of our knowledge, no dynamic compaction method for non scan sequential circuits has been reported in the literature. Another method for dynamic compaction for sequential circuits is also reported in this proceedings [13].

### 1.1 Overview

Our dynamic algorithm is based on two key ideas: (1) we identify bottlenecks that prevent vector compaction and test cycle reduction for the test sequences generated thus far, and (2) future test sequences are generated with an attempt to eliminate bottlenecks of earlier generated test sequences. Since our algorithm relies on identification and elimination of test sequence bottlenecks, we refer to our method as the *bottleneck elimination framework*. If a newly generated test sequence eliminates bottlenecks of an earlier test sequence, then we drop the earlier sequence. The dropped sequence is not included in the final test set. We demonstrate that vector compaction and test cycle reduction can be conflicting goals. Our method generates *minimal* test sets because static vector compaction or test cycle reduction using set-covering or extended set-covering approaches (for example, fault simulating the generated test sequences in reverse or any other order along with any specification of unspecified inputs) cannot further reduce the number of vectors or test cycles. The bottleneck elimination framework can be used in conjunction with any combinational or sequential test generator and fault simulator. Though we only consider the stuck-at fault model in this paper, the framework is applicable to other fault models as well. Experimental results on scan and non scan versions of the ISCAS-89 benchmarks and large production VLSI circuits are included.

## 2. BACKGROUND

A *test vector* is a set of logic values (0, 1, or X) that are simultaneously applied to the primary inputs of the circuit. A *test sequence* is an *ordered* set of test vectors that detect a target fault. A *test set* is an *unordered* set of test sequences. For full scan design circuits, a test sequence consists of only one test vector. The size of a test sequence is the number of test vectors in the sequence. The size of a test set is the number of vectors in all its test sequences. Given a test set, a fault is *essential* if only one test sequence can detect the fault. Such faults can be easily identified during fault simulation by dropping a fault only after it has been detected *twice* [5]. Any approach that selects a subset of test sequences from a given test set for covering all target faults will be referred to as a *set-covering* approach. Note that test sequences are not modified in a set-covering approach, except when *merging* vectors (See section 3). An example

of the set-covering approach is the technique of fault simulating the test sequences in reverse order of their generation [1] or any other order, and dropping test sequences that do not detect any additional faults. A set-covering approach that also changes test sequences by arbitrarily specifying a 0 or 1 value for the unspecified inputs is referred to as an *extended set-covering approach*. A test set is *optimal* with respect to a set-covering (extended set-covering) approach if none of its vectors or sequences can be dropped by a set-covering (extended set-covering approach).

## 3. VECTOR COMPACTION BOTTLENECKS

For the sake of clarity, we illustrate vector compaction bottlenecks in combinational (full scan) circuits and sequential (partial or non scan) circuits separately. Consider a test set for a combinational circuit. Assume that every test vector in the test set is fully specified. In addition, if every test vector has an essential fault, then the test set cannot be further reduced using any set-covering approach. This is because every test vector detects an essential fault and dropping a vector from the test set will result in a decrease in fault coverage.

If some vectors in the test set are partially specified, it is possible to *merge* two vectors into a single vector [11]. This merging is possible if the corresponding primary inputs of the two vectors do not have conflicting value assignments. However, if the test set does not have a pair of vectors that can be merged, then the size of this test set cannot be further reduced using a set-covering approach. These observations suggest two bottlenecks that prevent dropping of a test vector $v$ from a given test set:

**CB1.** $v$ detects one or more essential faults.

**CB2.** $v$ cannot be merged with any other test vector in the test set.

If a vector does not satisfy condition **CB1**, then it can be dropped from the test set. Note that it is possible for a test vector to satisfy condition **CB1** but violate condition **CB2**. If this happens, then the test vector can be merged with another vector in the test set. The essential faults for the test vector are also essential faults for the merged vector.

**Lemma 1:** *A test vector belongs to the optimal test set computed by a set-covering approach if and only if the test vector satisfies conditions* **CB1** *and* **CB2**.

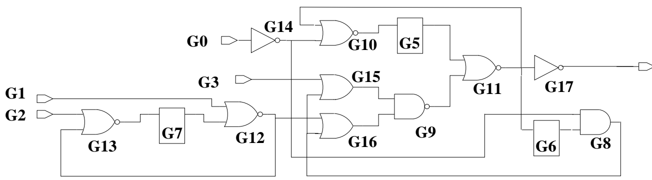


Figure 1: Circuit $s27$ of the ISCAS-89 benchmark set.

For sequential circuits, a target fault may require a test sequence consisting of more than one vector. Consider a test set that detects all target faults, in which each test sequence detects at least one essential fault. Furthermore, assume that no test sequence in this set is an ordered subsequence of any other test sequence. If test vectors have unassigned inputs, assume that no assignment of values to the unassigned signals will result in any test sequence becoming an ordered subsequence of any other test sequence. If every test sequence detects an essential fault, then, no set-covering approach can further reduce the number of sequences or vectors in the test set. These observations translate into the following bottlenecks that prevent a sequence $s$ from being dropped from a test set:

**SCB1.** $s$ detects one or more essential faults.

**SCB2.** $s$ is not an ordered subsequence of any other test sequence in the test set.

Again, it is possible that a test sequence satisfies condition **SCB1** but it may violate condition **SCB2**. If this happens, then the sequence can be merged with another sequence in the test set. Essential faults for the test sequence are also essential faults for the merged sequence.

**Lemma 2:** *A test sequence belongs to the optimal test set computed by a set-covering approach if and only if the test sequence satisfies conditions* **SCB1** *and* **SCB2**.

After identifying the bottlenecks for a test sequence, future test sequences can be generated with an attempt to eliminate the bottlenecks of the test sequence. We illustrate this idea using separate examples for combinational and sequential circuits.

Table 1: First three test vectors for the full scan version of $s27$.

| Vector | $G0$ | $G1$ | $G2$ | $G3$ | $G5$ | $G6$ | $G7$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Table 2: Test vector for fault $G2$ s-a-0.

| $G0$ | $G1$ | $G2$ | $G3$ | $G5$ | $G6$ | $G7$ |
|---|---|---|---|---|---|---|
| x | 0 | 0 | 0 | 0 | 1 | 1 |

Consider the circuit $s27$ that is part of the ISCAS-89 benchmark set. The netlist for this circuit is shown in Figure 1. This circuit has three FFs $G5$, $G6$ and $G7$. It has four primary inputs $G0$, $G1$, and $G2$ and $G3$. This circuit has one primary output $G17$. If we assume that all three FFs have been scanned, then the full scan design circuit now has three new primary inputs ($G5$, $G6$ and $G7$) and primary outputs ($G10$, $G11$ and $G13$). The first three test vectors produced by the test generator TRAN [14] are shown in Table 1. At this point, every one of the three vectors has an essential fault. For example, vectors 1, 2 and 3 detect essential faults $G6$ s-a-1, $G14$ s-a-1, and $G12$ s-a-0, respectively. Also, no vector can be merged with any of the other two vectors. Therefore, conditions **CB1** and **CB2** are satisfied for every one of these vectors. Consider test vector 1. It detects three essential faults: $G6$ s-a-1, $G7$ s-a-0 and $G14 \rightarrow G10$ s-a-0. It can be easily verified that these faults are not detected by test vectors 2 or 3. To drop test vector 1, future test vectors will have to eliminate the bottlenecks of vector 1. The next target fault selected by TRAN is $G2$ s-a-0. This fault is chosen since it has not yet been detected. The test vector shown in Table 2 is generated. This vector has an unspecified input. Also, it detects all the essential faults of vector 1 except $G6$ s-a-1. Furthermore, specifying $G0$ to a 0 or 1 does not detect any additional undetected faults. To eliminate vector 1, we attempt to *extend* the vector of Table 2 to detect the fault $G6$ s-a-1. TRAN targets the fault $G6$ s-a-1 and succeeds in extending the current vector to detect this fault. The extended vector is shown as vector 4 in Table 1. Since the bottleneck **CB1** for test vector 1 has been eliminated, we drop the vector.

Table 3: Test sequence for fault $G17$ s-a-0.

| Sequence | $G0$ | $G1$ | $G2$ | $G3$ |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| | 1 | 0 | 1 | 1 |

Consider again the circuit $s27$. Assume that no FFs are scanned. A commercial sequential test generator targets the fault $G17$ s-a-0 and generates the test sequence shown in Table 3. We will refer to this sequence as the first sequence. The next target undetected fault selected by the test generator is $G11 \rightarrow G10$ s-a-0 and the test generator generates the sequence shown in Table 4. This sequence

has several vectors with unspecified inputs. This flexibility can be utilized to eliminate the bottlenecks of the first sequence. Fault simulation of the sequence in Table 4 reveals that all but one ($G12 \rightarrow G13$ s-a-0) of the essential faults of the first sequence are detected. Now, to eliminate the bottlenecks of the first sequence, we attempt to extend the current test sequence to detect the essential fault $G12 \rightarrow G13$ s-a-0 of the first sequence. The test generator successfully extends the test sequence to obtain the sequence shown in Table 5. Since the bottleneck **SCB1** for the first sequence has been eliminated, we drop the first sequence. It is interesting to note that the first sequence is not a sub-sequence of the second sequence, but it can be dropped from the test set.

Table 4: Test sequence for fault $G11 \rightarrow G10$ s-a-0.

| Sequence | $G0$ | $G1$ | $G2$ | $G3$ |
|---|---|---|---|---|
| 2 | 1 | x | 0 | 0 |
| | 0 | x | 1 | 1 |
| | x | 0 | x | 1 |

Table 5: Extended test sequence for fault $G11 \rightarrow G10$ s-a-0.

| Sequence | $G0$ | $G1$ | $G2$ | $G3$ |
|---|---|---|---|---|
| 2 | 1 | 1 | 0 | 0 |
| | 0 | 1 | 1 | 1 |
| | 1 | 0 | 1 | 1 |

## 4. TEST CYCLE BOTTLENECKS

For a non scan circuit, the number of test cycles required to apply the test set is equal to the number of test vectors in the test set. Therefore, reducing the number of test vectors will also reduce the test application cycles. However, for partial or full scan design circuits, the number of test application cycles is significantly greater than the number of test vectors in the test set. This is because additional test application cycles are required to scan-in the values of scan FFs and scan out the circuit responses stored in the scan FFs. If a full scan design circuit has $F$ scan FFs and the test set has $T$ vectors, then the number of test application cycles required by the test set is $T(F+1)+F$. This analysis assumes scan-in and scan-out of every FF value. However, it may not be necessary to scan-in values for all scan FFs or scan-out values of all scan FFs [10].

Under the partial scan-in and scan-out model, test sets with the same number of test vectors may require significantly different number of test application cycles. As an example, consider again the full scan design version of circuit $s27$. The exact order of FFs in the scan chain is typically decided based on the layout in order to minimize routing overhead. For this design, assume that FF $G7$ is connected to the scan-in pin and FF $G6$ is connected to the scan-out pin. Therefore, three test cycles are required to load a desired value into FF $G6$. Only one test cycle is required to observe the circuit response stored in FF $G6$. Similarly, only one test cycle is required to load the desired value into FF $G7$ but three test cycles are required to observe the response stored in FF $G7$.

Table 6: Test set for three faults in circuit s27.

| Vector | $G0$ | $G1$ | $G2$ | $G3$ | $G5$ | $G6$ | $G7$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | x |
| 2 | 1 | x | x | x | x | x | x |

Consider a target fault set consisting of the following three faults: $G2 \rightarrow G13$ s-a-0, $G12$ s-a-1 and $G14 \rightarrow G10$ s-a-1. A possible test set for these faults is shown in Table 6. Test vector 1 detects the first two faults. This vector will require three test cycles to load the desired values into FFs $G5$ and $G6$. One test cycle is used to apply the primary input values and to allow the circuit to respond to the input stimulus. This test vector also requires observation of circuit response stored on signals $G13$ and $G17$. Three test cycles are required to observe the value on signal $G13$. Therefore, seven test cycles are required for the application of this test vector. The second test vector detects the remaining fault $G14 \rightarrow G10$ s-a-1. This vector does not require any specific values to be loaded into the scan FFs. One test cycle is required to apply the vector. Since the fault is detectable at the pseudo primary output $G10$, two additional test cycles are required to observe the value of signal $G10$. Therefore, the test set requires ten test application cycles.

An alternative test set for the target faults is shown in Table 7. Again, this test set also has two vectors. However, the first test vector detects the fault $G2 \rightarrow G13$ s-a-0 and the second vector detects the remaining faults. The first vector requires no scan-in cycles. One test cycle is required to apply the primary inputs and three cycles are required to observe the response at signal $G13$. The second test vector also requires no scan-in cycles. It requires one cycle to apply the primary input values and two cycles to observe the faulty response at signal $G10$. Therefore, this test set requires only seven test cycles. Both test vectors in the test set of Table 6 exhibit the following characteristics: (i) the vector detects at least one essential fault, and (2) the maximum scan-in or scan-out cycles needed to set up or observe the circuit response, respectively, is required for the detection of essential faults covered by the vector.

Table 7: Alternative test set for three faults in circuit s27.

| Vector | $G0$ | $G1$ | $G2$ | $G3$ | $G5$ | $G6$ | $G7$ |
|---|---|---|---|---|---|---|---|
| 1 | x | 1 | 1 | x | x | x | x |
| 2 | 1 | 1 | 0 | x | x | x | x |

For a given test set, the test cycles for scan-in (or scan-out) of all test vectors is clearly a lower bound on the test cycles required for the test set. In practice, we have observed that the number of test cycles required to scan-in all test vectors in a test set is significantly higher than the number of test cycles required to scan-out the relevant circuit responses for all test vectors. If a test vector detects a fault, it is possible that detection of the fault is not compromised when some of the scan FF values are left unspecified. This will modify the vector and may result in a new vector that requires fewer scan cycles. A fault is a *scan bottleneck* for a vector if no scan FF values can be unspecified to reduce the scan-in cycles required by the vector. Scan FF values are unspecified without compromising the detection of the fault. A similar analysis is possible for the scan-out case where detection of the fault is not compromised when circuit response in some of the scan FFs is not observed.

The bottlenecks that prevent further reduction in the number of scan-in cycles required by a test vector are as follows:

**TCB1.** The vector detects one or more essential faults.

**TCB2.** The scan bottleneck of the vector is an essential fault.

If condition **TCB1** is violated, then the vector can be dropped. If condition **TCB2** is violated, then the vector can be *trimmed* to reduce the scan-in cycles required by the vector. Trimming involves un-specifying state bits in the test vector so that the new vector still detects all essential faults of the original vector but requires fewer scan-in cycles.

Future test vectors can be generated with an attempt to eliminate test cycle bottlenecks of already generated vectors. As an example, consider again the first vector in the test set shown in Table 6. The first vector detects two essential faults and the fault $G12$ s-a-1 requires that the FF $G6$ be set to a particular value. Therefore, detection of this essential fault requires three scan-in cycles and it is a bottleneck for further reducing the scan-in requirement of the vector. The second test vector in Table 6 has several unspecified inputs and FFs. This flexibility can be used to extend the vector to eliminate the bottlenecks of the first vector. The second vector can be extended to detect the fault $G12$ s-a-1. The extended vector is identical to vector

2 in the test set shown in Table 7. Now, the fault $G12$ s-a-1 is no more an essential fault for vector 1. Therefore, this vector can be trimmed by un-specifying FFs $G5$ and $G6$. The trimmed vector is identical to the first vector in the test set of Table 7.

**Lemma 3:** *Consider a test set, obtained using set-covering and trimming, that requires the least number of test application cycles. A test vector belongs to this test set if and only if the vector satisfies conditions* **TCB1** *and* **TCB2**.

For partial scan circuits, a test sequence may consist of a series of vectors. The scan-in cycles required by the test sequence is the sum of scan-in cycles required by each vector in the sequence. If the sequence satisfies the following conditions, then the scan-in cycles required by the sequence cannot be reduced by trimming:

**STCB1.** The sequence detects one or more essential faults.

**STCB2.** Trimming any vector in the sequence will cause an essential fault to be undetectable by the sequence.

### 5. COMPACTION Vs. CYCLE REDUCTION

For partial or full scan designs, the smallest test set may not always require the least number of test cycles. As an example, consider the following two faults in circuit $s27$: $G10$ s-a-1 and $G17$ s-a-0. If we assume that all FFs are scanned, then a possible test vector that detects both faults is shown in Table 8. This test vector requires three scan-in cycles, one cycle to apply the primary inputs and two scan-out cycles. However, Table 9 gives an alternative test set that has more vectors but this test set requires fewer test cycles. Test vector 1 detects the fault $G10$ s-a-1. This vector requires no scan-in cycles, one cycle to apply the primary input values and two scan-out cycles to observe the response at signal $G10$. The second vector detects the fault $G17$ s-a-0. This vector also requires no scan-in cycles, one cycle to apply the primary input values and no scan-out cycles since $G17$ is a primary output. Therefore, the alternative test set requires only four test application cycles.

Table 8: Test vector for detecting faults $G10$ s-a-1 and $G17$ s-a-0.

| **Vector** | $G0$ | $G1$ | $G2$ | $G3$ | $G5$ | $G6$ | $G7$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | x | x | x | x | 0 | x |

Table 9: Test set for detecting faults $G10$ s-a-1 and $G17$ s-a-0.

| **Vector** | $G0$ | $G1$ | $G2$ | $G3$ | $G5$ | $G6$ | $G7$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | x | x | x | x | x | x |
| 2 | 1 | x | x | x | x | x | x |

### 6. BOTTLENECK ELIMINATION FRAMEWORK

A dynamic optimization framework that attempts to eliminate test sequence bottlenecks is embodied in the procedure BOTTLENECK_FRAMEWORK. The framework can be used to either optimize test set size or the test application cycles required by a test set. The algorithm begins by selecting an undetected fault. A test generator is used to generate the test sequence. No restrictions are placed on the test generation algorithm or the heuristics employed by the test generator. Typically, not all primary inputs and scan FFs have to be assigned values 0 or 1 to detect the target fault. We assume that the test generator does not randomly assign values to signals that were left unspecified in the test sequence. Ideally, the test generator should assign values to as few primary input signals and scan FFs as possible to detect the target fault.

```
Procedure BOTTLENECK_FRAMEWORK(circuit, faultlist)
{
    while (undetected faults exist){
        Pick next undetected fault.
        Generate test sequence T_current.
        EXTEND_SEQUENCE(T_current)
        ELIMINATE_BOTTLENECKS(T_current)
        Fault simulate test sequence T_current.
        while (a prior test sequence T_prior has no bottlenecks){
            Drop or trim test sequence T_prior.
            RECOMPUTE_BOTTLENECKS()
        }
        COMPUTE_SEQUENCE_BOTTLENECKS(T_curent)
}
```

The test sequence is fault simulated to identify other fortuitously detected faults. A fault is dropped during fault simulation only after it has been detected *twice*. This is unlike the more conventional fault simulation where a fault is dropped after it has been detected once. The list of essential faults for the prior test sequences can be incrementally updated during the fault simulation of the current test sequence. The fault simulator records the first and second test sequence, if any, that detect a fault. Therefore, given a fault, it is easy to identify if it is an essential fault. Also, the information about the test sequence that detects the fault is readily available.

The unspecified signals in a test sequence can be suitably specified to detect remaining undetected target faults. We will refer to this sequence as the *primary* sequence. The procedure EXTEND_SEQUENCE uses a test generator to suitably specify the unspecified signals in the primary sequence. The sequence obtained after specifying unspecified signals in the primary sequence is called the *secondary* sequence. During this phase, the test generator may or may not increase the number of vectors in the sequence. In full scan design circuits, the primary sequence for any stuck-at fault has only one vector. Also, detection of any target stuck-at fault will require at most one vector. Therefore, the primary and secondary sequences have only one vector. For partial or non scan circuits, the secondary sequence can have more vectors than the primary sequence. However, the primary sequence is a subsequence of the secondary sequence. Note that the primary sequence detects at least one fault that is not detected by any of the prior sequences. Therefore, the secondary sequence cannot be dropped because of the bottleneck **CB1** or **SCB1**.

The procedure ELIMINATE_BOTTLENECKS attempts to further extend the secondary sequence. However, the goal this time is to eliminate essential faults that are bottlenecks of prior test sequences. A possible order for considering the prior test sequences is to sort them based on increasing number of essential faults. A test sequence with the fewest essential faults can be selected and the test generator attempts to extend the secondary sequence to detect these essential faults. During this phase, we record the test sequences whose bottlenecks have been eliminated. These sequences are possible candidates that can be dropped or trimmed. Note that it is not possible to simultaneously drop or trim all these sequences. For example, assume that sequences $T_1$ and $T_2$ have been short listed to be dropped. Consider a fault $f$ that is only detected by sequences $T_1$ and $T_2$. This fault is not an essential fault since it is detected by two sequences. Therefore, this fault is not a bottleneck for either $T_1$ or $T_2$. However, simultaneous dropping of both sequences will result in a loss of fault coverage since the fault $f$ is not detectable by any prior test sequence. If simultaneous dropping of test sequences is desired, then some of the detected faults will now become undetected. These newly undetected faults are either fortuitously detected during fault simulation of future test sequences or they will have to be explicitly targeted by a test generator. Trimming a sequence has similar ramifications.

Whenever a test sequence is dropped or trimmed, this changes the compaction bottlenecks and test cycle bottlenecks of already generated test sequences. Consider the dropping of test sequence $T_i$. Only non-essential faults that are detected for the first or second time by the test sequence $T_i$ (double detected faults) could possibly become essential faults. Whether these faults are essential or not can be determined quickly, as follows. Since fault simulation records the first and second test sequence, if any, that detect every fault, we can easily compute the earliest test sequence $T_j$ that detects any of these non-essential faults. Bottleneck statistics of sequence $T_j$ and earlier sequences remain unchanged. We fault simulate only test sequences generated after test sequence $T_j$ with the non-essential faults of $T_i$ as the target fault list. We now have accurate information about the essential faults and we update the bottleneck statistics of these test sequences.

If every prior test sequence has bottlenecks that could not be eliminated by the current test sequence, then the procedure COMPUTE_SEQUENCE_BOTTLENECKS determines the bottlenecks for the current test sequence. The essential fault list for this vector is readily available from fault simulation. If test cycle reduction is desired, then scan bottlenecks are also computed for the current sequence.

## 7. MINIMALITY OF TEST SET

Static compaction techniques typically select a subset of sequences from a given test set that detects all target faults. The optimal subset of sequences has the least number of vectors among various subsets of sequences that detect all target faults. The subset selection problem belongs to the class of NP-complete problems. One heuristic that has been proposed to perform static compaction is reverse fault simulation [1]. Note that such methods do not modify any vector in the test set. In particular, they do not attempt to specify values to unspecified signals in the test sequences in order to merge test sequences. The test set obtained using the bottleneck elimination approach is a *minimal* test set.

**Theorem 1:** *A test set derived using procedure* BOTTLENECK_FRAMEWORK *cannot be further compacted using a set-covering approach.*

The scan-in and scan-out cycles required for any vector (sequence) in a test set produced by procedure BOTTLENECK_FRAMEWORK cannot be further reduced. However, test application cycles of the test set also depend on the order in which test vectors (sequences) are applied [10]. Several methods have been proposed that modify the test set during static vector compaction. A popular technique is to randomly or judiciously specify unspecified signals so that two sequences can be merged into a single sequence. This technique has been used for vector compaction in combinational [2] and sequential [11] circuits. These techniques are examples of extended set-covering approaches.

**Theorem 2:** *A test set $T_1 \ldots T_n$ that is derived using procedure* BOTTLENECK_FRAMEWORK *cannot be further compacted using an extended set-covering approach if the following conditions are satisfied during the generation of sequence $T_i$, $1 \leq i \leq n$:*

1. *Procedure* EXTEND_SEQUENCE *considers all faults not detected by sequences $T_1 \ldots T_{i-1}$ while extending the primary test sequence.*

2. *Procedure* BOTTLENECK_FRAMEWORK *considers all essential faults of sequences $T_1 \ldots T_{i-1}$.*

Recently, a static compaction method for combinational circuits has been proposed that replaces existing test vectors in a test set with entirely new test vectors that are obtained by using a test generator [3]. For sequential circuits, static compaction techniques that significantly modify the test sequences in a test set have also been proposed [11]. In the present work, a prior test sequence may satisfy both conditions **SCB1** and **SCB2**, but it may be possible to eliminate one or more more vectors in the sequence without compromising the detection of any essential fault. However, this implies that the prior test sequence is modified after it has been generated. These techniques do not fall into the category of set-covering or extended set-covering approaches. Such static compaction approaches can be applied to further reduce the size of test sets derived by procedure BOTTLENECK_FRAMEWORK.

## 8. IMPLEMENTATION AND RESULTS

The bottleneck elimination framework was implemented in the $C$ programming language. The program BECCS (*B*ottleneck *E*limination for *C*ompaction and *C*ycle reduction in *S*equential circuits) performs dynamic vector and test cycle reduction for scan and non scan circuits. For full scan designs, BECCS uses the test generator TRAN and a single-fault single-vector fault simulator. For partial scan or non scan circuits, BECCS relies on a commercial sequential test generator that has its own fault simulator. No modifications were made to heuristics used by the test generators for selecting target undetected faults, or for generating a test for a given target fault. Furthermore, no pre-processing like finding independent fault sets or fault ordering [5] was attempted. All experiments were performed on a Silicon Graphics Challenge L series machine.

BECCS executes the steps embodied in the procedure BOTTLENECK_FRAMEWORK. The specific implementation of procedure EXTEND_SEQUENCE can either consider all faults in a target fault set that are not detected by test sequences generated thus far or it can consider a pre-specified number of undetected faults. The latter feature is especially useful for significantly reducing the run time of BECCS without unduly increasing the test set. The implementation of procedure ELIMINATE_BOTTLENECKS considers all essential faults of test sequences generated thus far. Although not attempted here, information about independent faults [5] can be used to reduce the number of faults considered by procedure EXTEND_SEQUENCE or procedure ELIMINATE_BOTTLENECKS.

### 8.1 Full scan design circuits

Vector compaction results for full scan versions of the ISCAS-89 benchmark circuits are shown in Table 10. Test set sizes are reported for (1) the underlying test generator TRAN that does not employ any test set reduction techniques, (2) for the best known test sets published in the literature [5] using a dynamic compaction technique, and (3) for the dynamic compactor BECCS. Column *Test set sizes* shows the number of test vectors obtained by the three methods. Column *KP93* shows the test set sizes obtained using the dynamic compaction method described in [5]. The size of the test set obtained using BECCS is comparable to the best known test set sizes. For circuits $s344$, $s13207$ and $s15850$, BECCS produces a smaller test set. This is interesting since the current prototype of BECCS makes no modifications to the heuristics used by the test generator. In contrast, the dynamic compaction technique of [5] integrates several heuristics like maximal compaction, rotating backtrace, fault ordering based on independent fault sets and others into the test generation process.

The test sets generated by BECCS provide the maximum possible fault coverage since the fault efficiency, as shown in column *FE*, is 100% for all circuits. The computation time required by BECCS is shown in column *CPU sec*. Times required for test generation and fault simulation are shown separately, under columns *ATG* and *FS*, respectively. We used a single-pattern single-fault simulator for these experiments. Significant reduction in fault simulation time is possible by using a parallel-fault or parallel-pattern fault simulator. A

Table 10: Test set statistics for full scan designs.

| Ckt. | Test set sizes | | | FE | BECCS CPU sec. | |
|---|---|---|---|---|---|---|
| | TRAN | *KP93* | BECCS | (%) | ATG | FS |
| s27 | 15 | - | 6 | 100 | 0.0 | 0.0 |
| s208 | 47 | 27 | 29 | 100 | 0.3 | 0.2 |
| s298 | 55 | 23 | 24 | 100 | 0.3 | 0.3 |
| s344 | 36 | 15 | 14 | 100 | 0.2 | 0.2 |
| s349 | 36 | 13 | 14 | 100 | 0.2 | 0.2 |
| s382 | 54 | 25 | 27 | 100 | 0.3 | 0.3 |
| s386 | 100 | 64 | 68 | 100 | 0.3 | 0.4 |
| s400 | 52 | 24 | 27 | 100 | 0.2 | 0.3 |
| s420 | 85 | 43 | 45 | 100 | 1.0 | 0.8 |
| s444 | 49 | 24 | 26 | 100 | 0.3 | 0.6 |
| s510 | 76 | 55 | 57 | 100 | 0.8 | 0.6 |
| s526 | 116 | 51 | 53 | 100 | 0.6 | 0.7 |
| s526n | 116 | 51 | 53 | 100 | 0.8 | 1.0 |
| s641 | 101 | 24 | 25 | 100 | 2.8 | 1.7 |
| s713 | 102 | 24 | 29 | 100 | 2.3 | 1.1 |
| s820 | 196 | 95 | 96 | 100 | 1.8 | 2.3 |
| s832 | 197 | 96 | 97 | 100 | 1.7 | 2.7 |
| s838 | 152 | 75 | 76 | 100 | 5.1 | 3.7 |
| s953 | 132 | 79 | 83 | 100 | 2.0 | 4.2 |
| s1196 | 240 | 117 | 124 | 100 | 9.9 | 9.6 |
| s1238 | 255 | 129 | 131 | 100 | 9.3 | 8.4 |
| s1423 | 130 | 34 | 36 | 100 | 13.5 | 8.8 |
| s1488 | 214 | 102 | 109 | 100 | 3.2 | 6.9 |
| s1494 | 213 | 101 | 104 | 100 | 2.4 | 7.3 |
| s5378 | 432 | 104 | 116 | 100 | 69.4 | 80.3 |
| s9234 | 666 | 116 | 168 | 100 | 409.0 | 362.4 |
| s13207 | 744 | 239 | 238 | 100 | 334.2 | 484.5 |
| s15850 | 722 | 113 | 104 | 100 | 2462.0 | 876.2 |
| s35932 | 79 | - | 13 | 100 | 144.2 | 3109.4 |
| s38417 | 1601 | 91 | 107 | 100 | 1036.0 | 5551.0 |
| s38584 | 1240 | - | 128 | 100 | 1960.2 | 8288.4 |

Table 11: Test cycles for full scan designs.

| Ckt. | Partial scan-in/scan-out | | % Red. |
|---|---|---|---|
| | *Best known* | BECCS | |
| s27 | - | 25 | 7.41 |
| s208 | 224 | 230 | 14.50 |
| s298 | 308 | 342 | 8.56 |
| s344 | 241 | 229 | 4.18 |
| s349 | 223 | 223 | 6.69 |
| s382 | 361 | 563 | 8.45 |
| s386 | 462 | 473 | 1.87 |
| s400 | - | 565 | 8.13 |
| s420 | 664 | 639 | 18.18 |
| s444 | 346 | 470 | 20.74 |
| s510 | - | 400 | 1.23 |
| s526 | 789 | 1056 | 11.04 |
| s526n | - | 1043 | 12.13 |
| s641 | 377 | 412 | 20.62 |
| s713 | - | 454 | 24.21 |
| s820 | 602 | 575 | 1.03 |
| s832 | 592 | 579 | 1.36 |
| s838 | 2300 | 1891 | 25.55 |
| s953 | 580 | 2258 | 10.36 |
| s1196 | 664 | 1121 | 52.78 |
| s1238 | 651 | 1150 | 54.13 |
| s1423 | 1935 | 2658 | 4.18 |
| s1488 | 757 | 763 | 0.78 |
| s1494 | 751 | 728 | 0.82 |
| s5378 | 16175 | 17670 | 16.09 |
| s9234 | - | 34425 | 11.05 |
| s13207 | - | 118175 | 26.20 |
| s15850 | - | 49881 | 20.56 |
| s35932 | - | 23659 | 2.26 |
| s38417 | - | 170436 | 3.60 |
| s38584 | - | 185508 | 1.03 |

reduction in test generation time is possible by limiting the number of undetectable faults considered in extending a primary test sequence.

Test cycles for full scan-in/scan-out model can be easily derived from Table 10, using the formula given in Section 4. Table 11 shows the test cycles required for the test set obtained from BECCS under the partial scan-in/scan-out model. We implemented a greedy heuristic method for ordering the vectors in a test set to minimize test application cycles. We found that for all the circuits, the heuristic ordering resulted in test application cycles that were within 5% of the lower bound described in Section 4. Under column *Best known*, we report the best known number of test cycles required by a partial scan-in and scan-out model [10]. They only report results for the smaller ISCAS-89 circuits. A '-' in Table 11 indicates that no published data is reported for the corresponding circuit. Note that *unlike* BECCS, the method of [10] *re-orders FFs in the scan chain* to reduce test cycles. They determine the best order of scan FFs based on the given test set for the circuit. However, it is often desirable to determine the order of FFs in the scan chain based on the layout rather than the test set in order to minimize the routing overhead. BECCS can produce test sets for any given order of FFs in the scan chain (for the experiments, we assumed the order of FFs in the scan chain to be the same as the order in which the FFs are instantiated in the netlist). The column *Red.* shows the reduction in test cycles achieved by using the partial scan-in and scan-out model as opposed to the full scan-in and scan-out model.

The results of Table 11 show that BECCS test sets require *fewer* test cycles than the best known figures to date for several circuits ($s344$, $s420$, $s820$, $s832$, $s838$, $s1494$). For the larger circuits ($s9234$ and later), no previously reported results exist. The test cycles required by test sets derived from BECCS can be further reduced using static re-ordering of scan FFs [10].

#### 8.1.1 Production VLSI circuits

We used BECCS to perform vector compaction for large production VLSI circuits that consist of about $5,000$ to $50,000$ gates. The number of inputs, outputs, and gates in these circuits are shown in Table 12, under the columns *Inputs*, *Outputs* and *Gates*, respectively. Unlike the ISCAS-89 benchmarks, these circuits also contain non-Boolean primitives like tristate buffers, bidirectional buffers, and bus configurations. Vector compaction results for these circuits are shown in Table 13. The columns in Table 13 are similar to Table 10. Table 13 shows that BECCS produces test sets that are up to 12.9 times smaller than the test sets produced by the base test generator that does not attempt any compaction.

Table 12: Production VLSI circuit characteristics

| Ckt. | Inputs | Outputs | Gates |
|---|---|---|---|
| ckt1 | 336 | 340 | 7803 |
| ckt2 | 551 | 654 | 4656 |
| ckt3 | 134 | 32 | 6025 |
| ckt4 | 1133 | 1106 | 31416 |
| ckt5 | 2131 | 2304 | 49623 |

Table 13: Compaction results for full scan production circuits.

| Ckt. | Test set sizes | | FE | BECCS CPU sec. | |
|---|---|---|---|---|---|
| | TRAN | BECCS | (%) | ATG | FS |
| ckt1 | 899 | 263 | 100 | 1089.0 | 563.6 |
| ckt2 | 636 | 126 | 100 | 151.0 | 247.0 |
| ckt3 | 273 | 175 | 100 | 1420.9 | 651.2 |
| ckt4 | 721 | 56 | 100 | 1353.4 | 6643.4 |
| ckt5 | 5228 | 596 | 100 | 98773.4 | 23443.5 |

### 8.2 Partial or non scan design circuits

BECCS uses a commercial sequential test generator to perform

test generation for target faults in partial or non scan design circuits. Since the source code of the test generator is unavailable, it was integrated into BECCS using 350 lines of Bourne Shell scripts. Clearly, no modification of heuristics used by the test generator is possible. Information was exchanged between BECCS and the test generator by reading and writing files. BECCS maintains and updates the list of target faults, undetected faults and essential faults by making several calls to the fault simulator. Every time the fault simulator is called, it reads the circuit and fault list, sets up several internal data structures and performs fault simulation. BECCS always calls the test generator with a single target fault. The test generator reads in the circuit, sets up several data structures and performs test generation for only the target fault supplied by BECCS. This significant overhead that is incurred for every fault, that can be easily eliminated if the source code of the sequential test generator and fault simulator were available.

The test generator has a feature that is useful in extending a primary test sequence. It accepts constraints on the values that primary input signals can assume in different clock cycles. These constraints are honored during test generation, *i.e.*, all generated tests satisfy these constraints. BECCS uses this feature of the test generator to extend a primary sequence. The known signal values in the primary test sequence are translated into a set of constraints on the primary input signals. These constraints are specified in a separate file that is passed on to the test generator for every fault processed by the test generator. *Since* BECCS *interfaces with the sequential circuit test generator and fault simulator at a very high level, measuring CPU seconds for this experiment is not meaningful.* However, this integration serves an important purpose. It can be used to examine the quality (size) of test sets that can be obtained using the bottleneck elimination framework for sequential circuits. Table 14 reports compaction results for non scan circuits. Results for three methods are included. Under column *TG*, we report the number of vectors generated by the underlying test generator and the fault efficiency achieved by the test generator. Since no dynamic vector compaction technique for sequential circuits has been published in the literature, we compare our technique to a static compaction technique described in [11]. They describe several static compaction techniques that result in test sets with different fault coverages. We only consider the test set with the highest fault coverage. This is a fair comparison because BECCS achieves the maximum possible fault coverage (100% fault efficiency) for all the example circuits. Column *Static comp.* shows the number of vectors (in column *Vectors*) and the fault coverage (in column *Fault Cov.*) achieved by the test compaction technique described in [11]. They do not report the fault efficiency of their test set. Column BECCS shows the number of vectors in the test set obtained by using the program BECCS. The percentage reduction in test set that is obtained by using BECCS instead of the underlying test generator is shown in column *Red.*

Table 14: Test sets for non scan designs.

| Ckt. | TG | | Static comp. | | BECCS | | |
|---|---|---|---|---|---|---|---|
| | Vec. | FE (%) | Vec. | FC (%) | Vec. | FE (%) | Red. (%) |
| s208 | 211 | 100 | 187 | 68.8 | 129 | 100 | 39 |
| s386 | 374 | 100 | 270 | 87.4 | 229 | 100 | 39 |
| s420 | 211 | 100 | 206 | 46.6 | 134 | 100 | 37 |
| s713 | 221 | 100 | 192 | 81.9 | 132 | 100 | 40 |

The test sets derived using BECCS are significantly smaller than the test sets derived using the underlying test generator or static compaction techniques. These results are significant for several reasons: (1) test set reduction of as much as 40% is possible using BECCS as compared to the underlying test generator, (2) BECCS *uniformly* produces significantly smaller test sets than the static compaction method for all the example circuits, and (3) some of the static compaction techniques described in [11] can be used to further reduce the test set derived using BECCS. Also, more vectors may have to be added to the test sets of the static compaction approach to achieve 100% fault efficiency. These results show that the bottleneck elimination framework is effective in reducing the test set sizes for non scan circuits.

## 9. CONCLUSION

A new, dynamic vector compaction and test cycle reduction algorithm for combinational and sequential circuits was proposed. The algorithm identifies bottlenecks that prevent compaction and cycle reduction and generates future test sequences to eliminate the bottlenecks of earlier generated sequences. An important advantage of our algorithm is its applicability to both combinational and sequential circuits. The bottleneck removal algorithm can also be used to derive small test sets for other fault models like delay faults. Experimental results reveal that our framework produces test sets that are close to or better than the best reported previously. Although not attempted here, careful instrumentation of the test generator heuristics for vector compaction [5], use of independent faults, fault ordering, and static compaction techniques [3], can be incorporated into the framework for further reducing the size of test sets. We are currently implementing the proposed bottleneck elimination framework as part of the sequential test generator system SEST [15], to obtain compact test sets for large sequential circuits.

## REFERENCES


[1] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 126–136, Jan. 1988.

[1] L. N. Reddy, I. Pomeranz, and S. M. Reddy, "ROTCO: A Reverse Order Test Compaction Technique," in *Proc. EURO-ASIC Conf.*, pp. 189–194, Sept. 1992.

[3] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "On Compacting Test Sets by Addition and Removal of Test Vectors," in *VLSI Test Symp.*, pp. 202–207, Apr. 1994.

[4] P. Goel and B. C. Rosales, "PODEM-X: An Automatic Test Generation System for VLSI Logic Structures," in *Proc. 18th ACM/IEEE Design Automation Conf.*, pp. 260–268, June 1981.

[5] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," in *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 102–106, June 1993.

[6] S. Narayanan, R. Gupta, and M. Breuer, "Configuring Multiple Scan Chains for Minimum Test Time," in *Proc. Int. Conf. Computer-Aided Design*, pp. 4–8, Nov. 1992.

[7] B. Vinnakota and N. K. Jha, "Synthesis of Sequential Circuits for Parallel Scan," in *Proc. European Design Automation Conf.*, pp. 366–370, Mar. 1992.

[8] D. K. Pradhan and J. Saxena, "A Design for Testability Scheme to Reduce Test Application Time in Full Scan," in *Proc. VLSI Test Symp.*, pp. 55–60, Apr. 1992.

[9] E. M. Rudnick and J. H. Patel, "A Genetic Approach to Test Application Time Reduction for Full Scan and Partial Scan Circuits," in *Proc. 8th Int. Conf. VLSI Design*, Jan. 1995.

[10] Y. Higami, S. Kajihara, and K. Kinoshita, "A Reduced Scan Shift Method for Sequential Circuit Testing," in *Proc. Int. Test Conf.*, pp. 624–630, Oct. 1994.

[11] T. M. Niermann, R. K. Roy, J. H. Patel, and J. A. Abraham, "Test Compaction for Sequential Circuits," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 260–267, Feb. 1992.

[12] S. Y. Lee and K. K. Saluja, "Sequential Test Generation with Reduced Test Clocks for Partial Scan Designs," in *Proc. VLSI Test Symp.*, pp. 220–225, April 1994.

[13] I. Pomeranz and S. M. Reddy, "On Generating Compact Test Sequences for Synchronous Sequential Circuits," in *Proc. European Design Automation Conf.*, Sept. 1995.

[14] S. T. Chakradhar, V. D. Agrawal, and S. Rothweiler, "A Transitive Closure Algorithm for Test Generation," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1015–1028, July 1993.

[15] X. Chen and M. L. Bushnell, "Dynamic State and Objective Learning for Sequential Circuit Test Generation Using Decomposition Equivalence," in *Proc. of the 24th Int. Symp. Fault Tolerant Comput.*, pp. 446–455, June 1994.