Test Data Compression for IP Embedded Cores Using Selective Encoding of Scan Slices*

Zhanglei Wang and Krishnendu Chakrabarty
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708
Email: {zw8, krish}@ee.duke.edu

***Abstract*— We present a selective encoding method that reduces test data volume and test application time for scan testing of intellectual property (IP) cores. This method encodes the slices of test data that are fed to the scan chains in every clock cycle. Unlike many prior methods, the proposed method does not encode all the specified (0s and 1s) and unspecified (don't-care) bits in a slice. For example, if a slice contains more 1s than 0s, only the 0s are encoded and all don't-cares are mapped to 1. We use only c tester channels, where $c = \lceil \log_2(N+1) \rceil + 2$, to drive N scan chains. In the best case, we can achieve compression by a factor of N/c using only one tester clock cycle per slice. We derive an upper bound on the density of care bits (either 1s or 0s) that allows us to achieve the best-case compression. The pattern decompression is of the continuous-flow type because no complex handshakes are required between the tester and the chip. Unlike popular compression methods such as EDT and SmartBIST, the proposed approach is suitable for IP cores because it does not require structural information for fault simulation, dynamic compaction, or interleaved test generation. The on-chip decoder is small, independent of the circuit under test and the test set, and it can be shared between different circuits. We present compression results for a number of industrial circuits, and compare our results to other recent compression methods targeted at IP cores. We show that up to 28x reduction in test data volume and 20x reduction in testing time is obtained for these circuits.**

I. Introduction

Test data volume is now recognized as a major contributor to the cost of manufacturing testing of integrated circuits (ICs) [1]–[4]. Recent growth in design complexity and the integration of embedded cores in system-on-chip (SoC) ICs has led to a tremendous growth in test data volume; industry experts predict that this trend will continue over the next few years [5]. For example, the test data volume in 2014 is expected to be as much as 150 times the data volume in 1999 [6].

High test data volume leads to an increase in testing time. In addition, high test data volume may also exceed the limited memory depth of automatic test equipment (ATE). Multiple ATE reloads are time-consuming because data transfers from a workstation to the ATE hard disk, or from the ATE hard disk to ATE channels are relatively slow; the upload time ranges from tens of minutes to hours [7]. Test application time for scan testing can be reduced by using a large number of internal scan chains. However, the number of ATE channels that can directly drive scan chains is limited due to pin count constraints.

Logic built-in self-test (LBIST) [8] has been proposed as a solution for alleviating the above problems. LBIST reduces dependencies on expensive ATEs and allows precomputed test sets to be embedded in test sequences generated by BIST hardware to target random-pattern resistant faults. However, the memory required to store the top-up patterns for LBIST can exceed 30% of the memory used in a conventional ATPG approach [8]. With increasing circuit complexity, the storage of an extensive set of ATPG patterns on-chip becomes prohibitive [1]. Moreover, BIST can be applied directly to SoC designs only if the embedded cores are BIST-ready; considerable redesign may be necessary for incorporating BIST in cores that are not BIST-ready.

Test data compression offers a promising solution to the problem of increasing test data volume. A test set T_D for the circuit under test (CUT) is compressed to a much smaller data set T_E, which is stored in ATE memory. An on-chip decoder is used to generate T_D from T_E during test application. A popular class of compression schemes relies on the use of a linear decompressor. These techniques are based on LFSR reseeding [9]–[12] and combinational linear expansion networks consisting of XOR gates [13]–[15], and they have been implemented in commercial tools such as TestKompress from Mentor Graphics [1], SmartBIST from IBM/Cadence [3], and DBIST from Synopsys [16]. These compression schemes exploit the fact that scan test vectors typically contain a large fraction of unspecified bits even after compaction. However, the on-chip decoders for these techniques are specific to the test set, which necessitates decompressor redesign if the test set changes during design iterations. Moreover, since the decoder is not generic, it cannot be shared by multiple cores in an SoC. Finally, in order to achieve the best compression, these methods resort to fault simulation and test generation. As a result, they are less suitable for test reuse in SoC designs based on IP cores.

Another category of compression methods uses statistical coding, variants of run-length coding, dictionary-based coding, and hybrid techniques [17]–[26]. These methods exploit the regularity inherent in test data to achieve high compression. However, most of these schemes target single scan chains and they require synchronization between the ATE and CUT.

We present a selective encoding method that reduces test data volume and test application time for scan testing of intellectual property (IP) cores. This method encodes the slices

*This research was supported in part by the National Science Foundation under grants CCR-9875324 and CCR-0204077.

0-7803-9039-3/$20.00 © 2005 IEEE

of test data that are fed to the scan chains in every clock cycle. Unlike many prior methods, the proposed method does not encode all the specified (0s and 1s) and unspecified (don't-care) bits in a slice. For example, if a slice contains more 1s than 0s, only the 0s are encoded and all don't-cares are mapped to 1. We use only c tester channels, where $c = \lceil \log_2(N+1) \rceil + 2$, to drive N scan chains. The logarithmic reduction in the number of tester channels allows us to use a large number of internal scan chains, thereby reducing test application time significantly. In the best case, we can achieve compression by a factor of N/c using only one tester clock cycle per slice. We derive an upper bound on the density of care bits (either 1s or 0s) that allows us to achieve the best-case compression. The pattern decompression is of the continuous-flow type because no complex handshakes are required between the tester and the chip, and there is no need to introduce tester stall cycles. Unlike popular compression methods such as EDT and SmartBIST, the proposed approach is suitable for IP cores because it does not require structural information for fault simulation, dynamic compaction, or interleaved test generation. The on-chip decoder is small, independent of the circuit under test and the test set, and it can be shared between different circuits. We present compression results for a number of industrial circuits, and compare our results to other recent compression methods targeted at IP cores. We show that up to 28x reduction in test data volume and 20x reduction in testing time is obtained for these circuits.

The steady increase in clock frequencies over recent years has led to designs with a small number of gates between latches, or between latches and I/O pins. As a result, logic circuits today have very short combinational logic depth, and many logic cones with very little overlap. This is in contrast to older circuits such as the ISCAS-85 benchmarks that tend to have a smaller number of overlapping logic cones. A consequence of the shallow logic depth is that test patterns in present-day circuits contain many don't-care bits; e.g., it has been reported recently that test sets for industrial circuits contain only 1%-5% care bits [27]. After a desired stuck-at coverage is obtained, a commercial test pattern generator typically uses random fill to increase the likelihood of surreptitious detection of unmodeled faults. However, if the test sets for the cores are delivered with the don't-care bits to the system integrator, an appropriate compression method can be used at the system level to reduce test data volume and testing time. This imposes no additional burden on the core vendor. Unmodeled faults can still be detected if the compression method does not arbitrarily map all don't-cares to either 1s or 0s.

We do not address the problem of output compaction in this work. The proposed input compression method can be used with recent output compaction methods such as X-compact [2], convolutional compaction [28], and i-compact [29] to further reduce test data volume.

The rest of the paper is organized as follows. The details of the proposed approach are described in Section II. Section III presents the decompression architecture and Section IV presents compression results for industrial circuits.

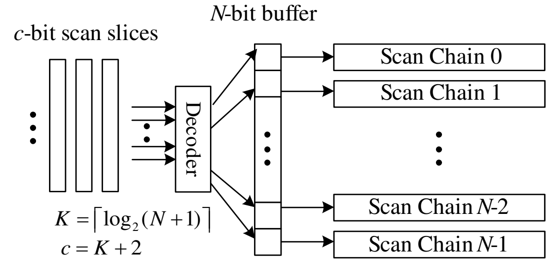


Fig. 1. Test application using the proposed approach.

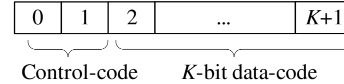


Fig. 2. A slice-code consists of a 2-bit control-code and a K-bit data-code.

II. Proposed Approach

As shown in Fig. 1, the proposed approach encodes the slices of test data (scan slices) that are fed to the internal scan chains. The on-chip decoder contains an N-bit buffer, and it manipulates the contents of the buffer according to the compressed data that it receives. After all the compressed data for a single slice is received, the data in the buffer is delivered to the scan chains.

Each slice is encoded as a series of c-bit *slice-codes*, where $c = K + 2$, $K = \lceil \log_2(N+1) \rceil$ and N is the number of internal scan chains in the CUT. As shown in Fig. 2, the first two bits of a slice-code form the *control-code* that determines how the following K bits, referred to as the *data-code*, are interpreted.

As described in Section I, the proposed approach only encodes a subset of the specified bits in a slice. First, the encoding procedure examines the slice and determines the number of 0- and 1-valued bits. If there are more 1s (0s) than 0s (1s), then all X's in this slice are mapped to 1 (0), and only 0s (1s) are encoded. The 0s (1s) are referred to as *target-symbols* and are encoded into data-codes in two modes: 1) *single-bit-mode* and 2) *group-copy-mode*.

In the single-bit-mode, each bit in a slice is indexed from 0 to $N-1$. A target-symbol is represented by a data-code that takes the value of its index. For example, to encode the slice "XXX10000", the X's are mapped to 0 and the only target-symbol of 1 at bit position 3 is encoded as "0011". In this mode, each target-symbol in a slice is encoded as a single slice-code. Obviously, if there are many target-symbols that are adjacent or near to each other, it is inefficient to encode each of them using separate slice-codes. Hence the group-copy-mode has been designed to increase compression efficiency.

In the group-copy-mode, an N-bit slice is divided into $M = \lceil N/K \rceil$ groups, and each group (with the possible exception of the last group) is K-bits wide. If a group contains more than two target-symbols, the group-copy-mode is used and the entire group is copied to a data-code. Two data-codes are needed to encode a group. The first data-code specifies the index of the first bit of the group, and the second data-code

contains the actual data. In the group-copy-mode, don't-cares can be randomly filled instead of being mapped to 0 or 1 by the compression scheme.

For example, let $N = 8$ and $K = 4$, i.e., each slice is 8-bits wide and consists of two 4-bit groups. To encode the slice "X1110000", the three 1s in group 0 are encoded. The resulting data-codes are "0000" and "X111", which refer to bit 0 (first bit of group 0) and the content of the group, respectively.

Since data-codes are used in both modes, control-codes are needed to avoid ambiguity. Control-codes "00", "01" and "10" are used in the single-bit-mode and "11" is used in the group-copy-mode. Control-codes "00", "01" are referred to as *initial-control-codes* and they indicate the start of a new slice.

The encoding procedure is summarized in Fig. 3. In Step (1), each test vector is divided into a series of slices. We then encode each slice as a series of slice-codes. In Steps (3)-(7), the numbers of 0s and 1s are calculated, and the target-symbol as well as the control-code of the first slice-code are set. The first slice-code of each slice must contain an initial-control-code.

Steps (8)-(14) encode all the groups of a slice. For each group of a slice, if it contains more than two target-symbols, it is encoded using the group-copy-mode; otherwise it is encoded using the single-bit-mode. A single-bit-mode slice-code contains a control-code of "00", "01", or "10". The data-code for the single-bit-mode ranges from 0 to N. However, since there are only N scan chains, a data-code of N is interpreted as a dummy and it implies that no bits are to be set. For example, if a slice contains no target-symbols, it is encoded as an initial-control-code and a dummy data-code.

Step (15) generates slice-codes for the entire slice. The first slice-code is a single-bit-mode code and it can encode any group that contains only one target-symbol. To improve compression efficiency, if $n \geq 2$ adjacent groups are to be encoded using the group-copy-mode, they are merged together. Only one data-code is used to specify the index of the first bit of these groups, followed by n data-codes carrying the contents of the n groups. Obviously, group-copy-mode slice-codes should be interleaved with single-bit-mode slice-codes. Since the data-code carrying the bit index of the first group and the data-codes carrying the actual data are all associated with the same control-code of "11", only the occurrence of a single-bit-mode slice-code can terminate a series of group-copy-mode slice-codes.

As can be seen from Fig. 3, the encoding procedure consists of two nested loops: the outer loop is for scan slices, and the inner loop is for groups of a slice. Hence its time complexity is $O(S \cdot N) = O(V)$. Table I shows a complete example to further illustrate the encoding procedure.

The following two theorems provide more insights into the proposed compression method based on selective encoding of scan slices.

Theorem 1: Let the test data volume for the CUT with N internal scan chains be V bits. Let the compressed test data volume obtained after selective encoding of scan slices be U bits. The maximum value of the compression factor V/U that can be achieved is given by $f = N/c$, where $c = \lceil \log_2(N+1) \rceil + 2$ corresponds to the number of ATE channels used to drive the decompression logic.

Proof: The maximum compression is achieved when each slice is encoded as a single slice-code. Since a total of $c = \lceil \log_2(N+1) \rceil + 2$ ATE channels is used in the proposed method, the test data volume for every scan slice is reduced by a factor of N/c. ■

Theorem 2: Let the number of scan slices be S, and let $k_{0,i}$ ($k_{1,i}$) be the number of 0s (1s) in scan slice i, $0 \leq i \leq S-1$. A necessary condition for the maximum compression factor of Theorem 1 to be achieved is given by:

$$\sum_{i=0}^{S-1} \min(k_{0,i}, k_{1,i}) \leq S \tag{1}$$

Proof: The maximum compression factor of Theorem 1 can only be achieved if every slice is encoded as a single slice-code. This in turn is only possible if every scan slice contains either zero or one target-symbol, i.e., $\min(k_{0,i}, k_{1,i}) \leq 1$, $0 \leq i \leq S-1$. The necessary condition given by (1) therefore follows. ■

The significance of Theorem 2 lies in the fact that the maximum compression factor of N/c can only be achieved if the test generator is tailored to satisfy (1). Test set relaxation methods as in [30] can also be used to satisfy (1). Such methods however require structural information about the circuit.

A property of the proposed compression method is that consecutive c-bit compressed slices fed by the ATE are often

```
Encoding procedure:
 (1) Format the given test vectors into slices;
 (2) for each slice
 (3)    Determines the number of 0s (k0) and 1s (k1) in the
        slice;
 (4)    If k0 > k1 then
 (5)      target-symbol := 1, 1st control-code := 00;
 (6)    else
 (7)      target-symbol := 0; 1st control-code := 01;
 (8)    for each group of the slice
 (9)      calculate the number of target-symbols;
(10)      if number-of-target-symbols > 2 then
(11)        encode the group using the group-copy-mode;
(12)      else
(13)        encode the group using the single-bit-mode;
(14)    end for (group);
(15)    generate slice-codes for the current slice.
(16) end for (slice);
```

Fig. 3. Encoding procedure.

TABLE I

AN EXAMPLE TO ILLUSTRATE SLICE ENCODING.

Slice	Slice code: Control code	Slice code: Data code	Description
XX00 010X	00	0101	Start a new slice, map X to 0, set bit 5 to 1.
1110 0001	00	0111	Start a new slice, map X to 0, set bit 7 to 1.
	11	0000	Enter group-copy-mode starting from bit 0 (i.e.., group 0).
	11	1110	The data is 1110.
XXXX XX11	01	1000	Start a new slice, map X to 1, no bits are set to 0.

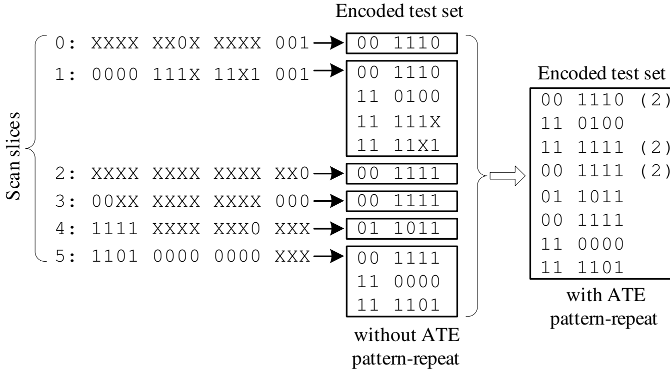


Fig. 4. Encoding example.

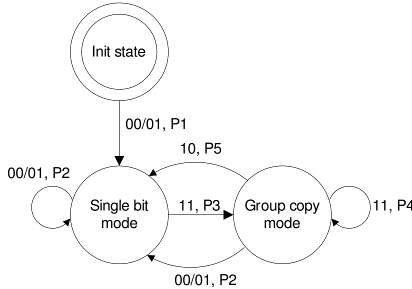


Fig. 5. State transition diagram of the decoder.

identical or compatible. Therefore, ATE pattern-repeat can be used to further reduce test data volume after selective encoding of scan slices. In the uncompressed data sets, especially among the test vectors that lie near the end of a test set, there are a large number of consecutive slices that contain no target-symbols. These slices are encoded as identical single slice-codes that have only dummy data-codes. With ATE pattern-repeat, these slice-codes can be further compacted. Additionally, consecutive group-copy-mode slice-codes can also be compacted if they are compatible. Fig. 4 shows how a set of scan slices are encoded. The example shows that some slice-codes, e.g., the first two in the encoded test set, can be combined and applied using ATE pattern-repeat.

III. Decompression Architecture

Fig. 5 shows the state transition diagram of the decoder. The decoder enters designated states and performs different operations as specified by the control-codes that it receives. Initially the decoder is in the init state; when it receives an "initial-control-code", it enters the single-bit-mode and performs a series of operations referred to as P1. Table II explains the five groups of operations (P1–P5) in Fig. 5.

Fig. 6 shows the block diagram of the decoder. The finite-state machine (FSM) generates control signals for the other components. The K-bit address register is used in the group-copy-mode to store the index of the first bit of the target group. This register can be incremented by K to address a series of adjacent groups. The K-to-N address decoder generates selection signals to address a single bit of the buffer. The N-bit buffer contains combinational logic that provides the following functionalities: (1) each bit in the buffer can be individually addressed, and (2) all bits in the same group can be addressed in parallel. These two functions are used in the single-bit-mode and the group-copy-mode, respectively.

TABLE II

Five Groups of Operations for the Decoder.

Group	Description
P1	If the control-code is 00 (01), the target-symbol is 1 (0). Set the bit specified by the associated data-code to the target-symbol and all other bits to the complemented value.
P2	Shift the current content of the buffer to the internal scan chains, then perform P1.
P3	Save the value of the data-code to an address register, which is the index of the first bit of the group.
P4	Copy the value of the data-code to the group specified by the address register, then increment the address register by K.
P5	Set the bit specified by the data-code to the target-symbol.

The 2-bit input signal $control$ is the control-code from the tester. The signal rst, when asserted to 0, resets the FSM to its initial state. The signal v is set to high when the decoding process for a slice is finished and the content of the buffer is shifted to the internal scan chains.

If the signal is_grp is asserted, the decoder works in the group-copy-mode. The inc_grp is used to increment the address register by K. In the group-copy-mode, the K-to-N address decoder receives input from the address register; in the single-bit-mode, it receives input from the data-code input. The N-bit selection (sel) signal is used to address a single bit in the buffer. At any given time only one of the N wires is asserted.

The second bit of the control-code ($control$), i.e., the target-symbol, is latched to ts. In the single-bit-mode, the specified bit is set to ts. If the control-code is "00" or "01", the signal set_buf is asserted, and all other bits except the specified bit are set to the complement of ts ($\overline{ts}$). The signal den is asserted whenever the buffer contents are to be changed. The signal den is set to 0 only during the first clock cycle of the group-copy-mode; at the same time the address of the first bit of the group is loaded to the address register.

The sel signal from the address decoder can only address a single bit of the buffer. However, in the group-copy-mode, all the K bits in the target group should be addressed (the last group may contain less bits). Therefore, in the group-copy-mode, additional combinational logic is needed to address the other bits together with the first bit. For each bit i of the buffer, an N-bit signal $en(i)$ is defined, where

$$en(i) = \begin{cases} sel(i) & \text{if } i \bmod K = 0; \\ sel(i) \vee (is_grp \wedge & \\ sel(i - (i \bmod K)) & \text{if } i \bmod K \neq 0. \end{cases}$$

where $i = 0, 1, \ldots, N-1$.

Fig. 7 shows the structure of a single bit of the buffer. Each bit is represented by a falling edge-triggered D-flipflop (DFF) with enable input (EN). The DFF receives input data from a multiplexer and receives the enable signal from the combinational logic. In the group-copy-mode, the input is from the data-code from the tester. Bit i of the buffer is connected to bit $i \bmod K$ of the data-code. In the single-bit-mode, the

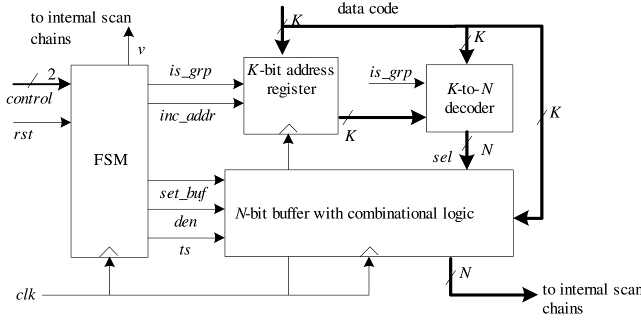


Fig. 6. Block diagram of the decoder.

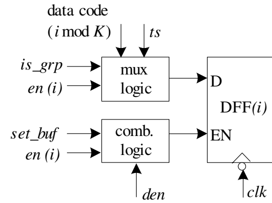


Fig. 7. One bit of the buffer.

input is either ts or $\overline{ts}$, as determined by $en(i)$. The multiplexer implements the following function:

$$D = \begin{cases} data\ code(i \bmod K) & \text{if } is_grp = 1; \\ \overline{ts} & \text{if } is_grp = 0,\ en(i) = 1; \\ ts & \text{if } is_grp = 0,\ set_buf = 1. \end{cases}$$

The DFF is changed only when EN is high. The combinational logic implements the boolean function

$$EN = den \ \wedge \ (set_buf \ \vee \ en(i))$$

The DFF's are falling edge-triggered and the FSM is rising edge-triggered. Therefore, upon the rising edge of the clk signal, all control signals are generated and become stable before the falling edge of clk. The buffer is updated at the falling edge of clk.

The state diagram of the FSM is shown in Fig. 8. The state $S0$ is the initial state. States $S1$ and $S2$ correspond to the single-bit-mode and the group-copy-mode, respectively. We simulated the decoder using VHDL and Synopsys tools to ensure its correct operation. We also synthesized the decoder using Synopsys Design Compiler to assess the hardware overhead. The synthesized FSM contains only 5 flip-flops and 23 combinational gates. For the lsi_10k library, the reported area is 55 units. The other parts of the decoder are synthesized separately since they depend on K and N. For $N = 64$ and $K = 7$, the synthesized circuit contains 536 gates and 71 flip-flops, and the area is 1341 units. If $N = 1024$ and $K = 11$, the synthesized circuit contains 6409 gates and 1035 flip-flops, and the area is 18,877 units. For the larger than million-gate designs considered in our experiments, this corresponds to an area overhead of only 1%. The schematic of the FSM is shown in Fig. 9.

IV. Experimental results

In this section, we apply the proposed approach to eight representative industrial circuits. These circuits vary in size from

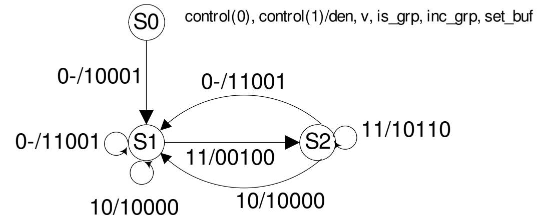


Fig. 8. Decode FSM state diagram.

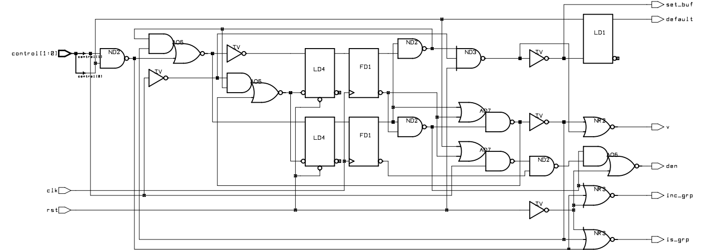


Fig. 9. Synthesized FSM in the decoder.

approximately 50K gates to over 1.4M gates. For each circuit, we compress test sets with high fault coverage that were provided to us by industrial partners. These test sets are generated by commercial ATPG tools with dynamic compaction turned on and random-fill turned off. The percentages of specified bits in these test sets are approximately in the range 1%-4%. Table III describes these circuits and the corresponding test sets. Ckt-1 and Ckt-2 are deterministic test sets after applying 64,000 random vectors. For ckt-3, we were provided with four different sets of test cubes. The differences in these test sets lie in the maximum number of care bits in each test pattern. For example, in ckt-3-2000, each test pattern is constrained to have no more than 2000 care bits.

We do not report compression results for the ISCAS-89 benchmark circuits because they are too small to be representative of today's designs. Moreover, their test sets contain too many care bits, due in part to their large logic depth.

Table IV shows the compression results obtained using the proposed method for the different test cases. Column N refers to the number of internal scan chains and column c denotes the number of ATE channels. We consider a varying number of internal scan chains N to show how an appropriate value of N can be determined for designs with flexible scan chains. The test application time and the size of compressed data are shown in columns TAT and T_E, respectively. The parameter $\Upsilon_V = |T_D| \,/\, |T_E|$ refers to the data volume reduction factor. The parameter Υ_{TAT} is the test application time reduction factor over standard scan testing based on c ATE channels. Without ATE pattern-repeat, Υ_V and Υ_{TAT} are as high as 22.96x and 22.91x, respectively. With ATE pattern-repeat, Υ_V is as high as 28.82x. For ckt-8 with 255 internal scan chains, the compression of 20.12x is close to the theoretical maximum of 25.5x (without ATE pattern-repeat) predicted by Theorem 1. The CPU time for generating the compressed data is at most two minutes, even for the largest circuits.

We compare the proposed compression method to two other recent compression methods that have been proposed for IP

TABLE III

DESCRIPTION OF INDUSTRIAL CIRCUITS AND TEST SETS.

Circuit	No. of gates	No. of scan cells	No. of test cubes	Size of T_D (bits)	Fault coverage	Percentage of specified bits
ckt-1	51,082	12,256	3,768	46,180,608	99.80%	2.18
ckt-2	94,340	22,216	2,636	58,561,376	99.76%	4.27
ckt-3-2000	121,470	9,628	2,191	21,094,948	98.66%	1.96
ckt-3-1000			2,409	23,193,852	98.51%	2.04
ckt-3-500			3,072	29,577,216	98.45%	1.83
ckt-3-200			4,927	47,437,156	98.27%	1.32
ckt-4	302,714	43,414	1,528	66,336,592	99.42%	1.58
ckt-5	404,860	26,970	4,899	132,126,030	98.85%	1.31
ckt-6	1.18M	$\sim$ 80,000	2,859	231,601,872	97.86%	2.58
ckt-7	1.21M	$\sim$ 20,000	18,027	400,289,535	99.16%	1.76
ckt-8	1.41M	$\sim$ 110,000	18,142	1,974,992,546	95.07%	0.92

cores. These methods also do not require fault simulation or test generation. To ensure fairness of comparison, we do not consider compression methods that require structural information. Table V presents comparative data for two-dimensional compression [25]. The compression method in [25] was implemented and applied to a number of industrial test cases. In every case considered, the number of ATE channels required is much less for the proposed method compared to [25]. Out of the 21 cases considered, $|T_E|$ for [25] is higher in 20 cases. The value of Υ_{TAT} in [25] is smaller in 18 cases.

The test sets described in Table III were obtained using dynamic compaction during ATPG. As is the case of other compression methods, these test sets were not compressed further using static compaction after ATPG. In some cases, e.g. ckt-3, a commercial ATPG tool was given a maximum number of care bits per vector as a constraint. It was reported in [25] that the compressed test sets for ckt-1 and ckt-2 are an order of magnitude smaller than the compacted test sets used during production testing for these circuits. Therefore, the proposed method can achieve significant reduction in data volume over ATPG-compacted test sets.

Table VI compares the proposed method to the recent compression technique based on dictionaries with corrections [21]. We implemented the procedures from [21] and applied this technique to several industrial test cases. Table VI is similar to Table V, with an additional column mem that shows the size of the on-chip memory. Out of the 24 cases considered, $|T_E|$ in [21] is higher in 15 cases. Note that for the 9 cases where $|T_E|$ in [21] is less, an excessive amount of on-chip storage (as high as 8M bits) is needed for [21]. Hence it is difficult to use [21] in practice for these cases. TAT is lower in [21] in most cases, but it also requires a much larger number of ATE channels. If the number of ATE channels and the amount of on-chip storage are limited (or constrained), the proposed method outperforms [21] both in terms of T_E and TAT.

Finally, we determine for each circuit, the number of internal scan chains N that leads to the maximum data volume reduction factor Υ_V as well as the value of N that leads to the maximum TAT reduction factor Υ_{TAT}. For IP cores with flexible scan chains, this information can allow appropriate scan chain configurations. Fig. 10 and Fig. 11 show how Υ_V and Υ_{TAT} vary with N for the test cases. For some circuits, namely ckt-1, ckt-2, ckt-3-200, and ckt-7, the best value of N for the highest Υ_V also leads to the highest Υ_{TAT}. However, for the other test cases, the best value of N for the highest Υ_V does not maximize Υ_{TAT}. For example, for ckt-8, Υ_{TAT} is maximum for $N = 255$ while Υ_V is maximum for $N = 511$.

V. CONCLUSION

We have presented a test data compression technique for designs with multiple scan chains. This method does not require detailed structural information about the circuit under test (CUT), and utilizes a generic on-chip decoder that is independent of the CUT and the test set. While the hardware overhead depends on the number of internal scan chains, we have seen that for an industrial circuit with over 1M gates, the overhead is only 1% for as many as 1024 internal scan chains. If a small amount of circuit redesign is permitted, we can reduce the hardware overhead by modifying the first scan cells of each scan chain such that they can be used as the N-bit on-chip buffer. The clock inputs of these scan cells need to be appropriately gated so that they can be triggered separately from other cells in the same scan chain. Experimental results for eight industrial circuits show that compared to dynamically compacted test sets, up to 28x reduction in test data volume and 20x reduction in test application time can be obtained.

ACKNOWLEDGMENTS

We thank Subhasish Mitra and Anshuman Chandra for helpful suggestions. We are grateful to a number of colleagues from industry who provided test sets (ckt-1 to ckt-8) for our experiments. These contributors wish to remain anonymous.

REFERENCES

[1] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. CAD*, vol. 23, pp. 776–792, May 2004.

[2] S. Mitra and K. S. Kim, "X-Compact: An efficient response compaction technique," *IEEE Trans. CAD*, vol. 23, pp. 421–432, Mar. 2004.

[3] B. Koenemann, C. Banhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST variant with guaranteed encoding." in *Proc. Asia Test Symposium*, 2001, pp. 325–330.

[4] S. Mitra and K. S. Kim, "XMAX: X-Tolerant architecture for maximal test compression," in *Proc. IEEE Intl. Conf. Computer Design*, 2003, pp. 326–330.

TABLE IV
COMPRESSION RESULTS.

Circuit	N	c	Without ATE pattern-repeat TAT (cycles)	Without ATE pattern-repeat $\|T_E\|$ (bits)	Without ATE pattern-repeat Υ_V (x)	With ATE pattern-repeat $\|T_E\|$ (bits)	With ATE pattern-repeat Υ_V (x)	With ATE pattern-repeat Υ_{TAT} (x)
ckt-1	255	10	400,960	3,971,920	11.63	3,322,690	13.90	11.53
	511	11	354,260	3,855,412	11.98	3,325,267	13.89	11.87
	800	12	306,482	3,632,568	12.71	3,153,672	14.64	12.58
	1,023	12	338,049	4,011,372	11.51	3,421,764	13.50	11.40
ckt-2	255	10	802,354	7,997,180	7.32	6,514,040	8.99	7.30
	511	11	739,859	8,109,453	7.22	6,626,180	8.84	7.20
	700	12	628,795	7,513,908	7.79	6,108,024	9.59	7.77
	1,023	12	692,295	8,275,908	7.08	6,540,048	8.95	7.06
ckt-3-2000	255	10	208,556	2,063,650	10.22	1,424,740	14.81	10.13
	511	11	180,526	1,961,685	10.75	1,448,392	14.56	10.64
	1,023	12	164,771	1,950,960	10.81	1,440,396	14.65	10.69
ckt-3-1000	255	10	232,104	2,296,950	10.10	1,784,390	13.00	10.01
	511	11	208,640	2,268,541	10.22	1,909,611	12.15	10.13
	1,023	12	198,263	2,350,248	9.87	1,942,968	11.94	9.77
ckt-3-500	255	10	256,100	2,530,280	11.69	2,042,330	14.48	11.56
	511	11	234,888	2,549,976	11.60	2,277,814	12.98	11.47
	1,023	12	228,603	2,706,372	10.93	2,423,400	12.20	10.80
ckt-3-200	255	10	279,450	2,745,230	17.28	2,215,140	21.42	17.00
	511	11	253,568	2,735,051	17.34	2,506,174	18.93	17.04
	1,023	12	253,761	2,986,008	15.89	2,798,136	16.95	15.61
ckt-4	255	10	508,588	5,070,600	13.08	3,264,850	20.32	13.05
	511	11	411,454	4,509,186	14.71	3,391,454	19.56	14.66
	1,023	12	369,497	4,415,628	15.02	3,505,908	18.92	14.97
	2,047	13	347,795	4,501,471	14.74	3,574,064	18.56	14.68
ckt-5	255	10	816,757	8,118,580	16.27	6,079,410	21.73	16.18
	511	11	677,339	7,396,840	17.86	6,462,555	20.44	17.74
	1,023	12	662,548	7,891,788	16.74	7,178,544	18.41	16.63
	2,047	13	669,840	8,644,233	15.28	7,831,798	16.87	15.18
ckt-6	255	10	2,543,116	25,402,570	9.12	22,531,820	10.28	9.11
	511	11	2,510,478	27,583,809	8.40	25,517,118	9.08	8.39
	1,023	12	2,535,011	30,385,824	7.62	27,914,232	8.30	7.61
	2,047	13	2,540,256	32,986,161	7.02	29,083,392	7.96	7.02
ckt-7	255	10	2,939,692	29,216,650	13.70	25,847,430	15.49	13.63
	511	11	2,845,279	31,099,772	12.87	29,058,689	13.78	12.80
	1,023	12	2,915,940	34,774,956	11.51	32,587,644	12.28	11.45
	2,047	13	2,992,613	38,669,618	10.35	35,185,618	11.38	10.30
ckt-8	255	10	9,835,025	98,168,830	20.12	68,528,200	28.82	20.08
	511	11	7,839,684	86,036,962	22.96	74,718,468	26.43	22.91
	1,023	12	7,607,971	91,077,948	21.68	86,462,016	22.84	21.64
	2,047	13	7,809,912	101,293,010	19.50	98,412,002	20.07	19.46

TABLE V
COMPARISON WITH 2-DIMENSIONAL COMPRESSION [25].

Circuit	N	[25] c	[25] TAT (cycles)	[25] $\|T_E\|$ (bits)	[25] Υ_V (x)	[25] Υ_{TAT} (x)	Proposed method c	Proposed method TAT (cycles)	Proposed method $\|T_E\|$ (bits)	Proposed method Υ_V (x)	Proposed method Υ_{TAT} (x)
ckt-1	255	20	195,799	3,840,620	12.02	11.82	10	400,960	3,322,690	13.90	11.53
	511	30	126,840	3,692,160	12.51	12.18	11	354,260	3,325,267	13.89	11.87
	1023	45	65,712	2,787,480	16.57	15.71	12	338,049	3,421,764	13.50	11.40
ckt-3-2000	255	191	36,771	6,604,780	3.19	3.10	10	208,556	1,424,740	14.81	10.13
	511	311	19,671	5,436,280	3.88	3.56	11	180,526	1,448,392	14.56	10.64
	1023	476	11,321	4,345,880	4.85	4.26	12	164,771	1,440,396	14.65	10.69
ckt-3-1000	255	147	46,451	6,474,174	3.58	3.47	10	232,104	1,784,390	13.00	10.01
	511	228	24,487	5,033,784	4.61	4.33	11	208,640	1,909,611	12.15	10.13
	1023	349	14,149	4,097,260	5.66	4.94	12	198,263	1,942,968	11.94	9.77
ckt-3-500	255	106	61,744	6,219,232	4.76	4.58	10	256,100	2,042,330	14.48	11.56
	511	166	32,674	4,913,932	6.02	5.55	11	234,888	2,277,814	12.98	11.47
	1023	248	18,802	3,901,040	7.58	6.54	12	228,603	2,423,400	12.20	10.80
ckt-3-200	255	67	116,191	7,454,688	6.36	6.15	10	279,450	2,215,140	21.42	17.00
	511	103	62,326	5,912,097	8.02	7.51	11	253,568	2,506,174	18.93	17.04
	1023	149	36,207	4,660,720	10.18	8.98	12	253,761	2,798,136	16.95	15.61
ckt-4	255	235	178,513	41,591,475	1.59	1.59	10	508,588	3,264,850	20.32	13.05
	511	466	89,673	41,075,570	1.61	1.62	11	411,454	3,391,454	19.56	14.66
	1023	898	46,033	39,965,490	1.66	1.66	12	369,497	3,505,908	18.92	14.97
ckt-5	255	241	400,809	95,414,310	1.38	1.38	10	816,757	6,079,410	21.73	16.18
	511	453	202,801	89,649,606	1.47	1.47	11	677,339	6,462,555	20.44	17.74
	1023	798	105,744	80,474,310	1.64	1.62	12	662,548	7,178,544	18.41	16.63

TABLE VI

COMPARISON WITH RECENT DICTIONARY-BASED COMPRESSION METHOD [21].

Circuit	N	[21]						Proposed method				
		c	mem (bits)	TAT (cycles)	$\vert T_E \vert$ (bits)	Υ_V (x)	Υ_{TAT} (x)	c	TAT (cycles)	$\vert T_E \vert$ (bits)	Υ_V (x)	Υ_{TAT} (x)
ckt-1	255	19	295,176	188,400	3,508,008	12.14	12.94	10	400,960	3,322,690	13.90	11.53
	511	21	1,126,244	94,200	1,899,072	15.26	23.40	11	354,260	3,325,267	13.89	11.87
	1023	22	3,433,920	48,984	994,752	10.43	43.00	12	338,049	3,421,764	13.50	11.40
ckt-2	255	20	946,726	234,604	4,639,360	10.48	12.49	10	802,354	6,514,040	8.99	7.30
	511	22	3,350,170	118,620	2,551,648	9.92	22.47	11	739,859	6,626,180	8.84	7.20
	1023	24	8,589,040	60,628	1,391,808	5.87	40.30	12	692,295	6,540,048	8.95	7.06
ckt-3-2000	255	19	401,574	85,449	1,581,902	10.64	13.03	10	208,556	1,424,740	14.81	10.13
	511	20	834,015	43,820	832,580	12.66	24.15	11	180,526	1,448,392	14.56	10.64
	1023	21	1,523,466	24,101	460,110	10.63	41.82	12	164,771	1,440,396	14.65	10.69
ckt-3-1000	255	19	354,584	93,951	1,739,298	11.08	13.03	10	232,104	1,784,390	13.00	10.01
	511	20	919,698	48,180	915,420	12.64	24.15	11	208,640	1,909,611	12.15	10.13
	1023	21	1,909,629	26,499	505,890	9.60	41.82	12	198,263	1,942,968	11.94	9.77
ckt-3-500	255	18	152,146	119,808	2,101,248	13.13	13.74	10	256,100	2,042,330	14.48	11.56
	511	20	614,484	61,440	1,167,360	16.60	24.15	11	234,888	2,277,814	12.98	11.47
	1023	21	1,685,250	33,792	645,120	12.69	41.82	12	228,603	2,423,400	12.20	10.80
ckt-3-200	255	15	23,622	192,153	2,808,390	16.75	16.49	10	279,450	2,215,140	21.42	17.00
	511	18	146,523	98,540	1,685,034	25.90	26.80	11	253,568	2,506,174	18.93	17.04
	1023	20	648,099	54,197	985,400	29.04	43.91	12	253,761	2,798,136	16.95	15.61
ckt-4	255	20	871,474	262,816	5,225,760	10.88	12.63	10	508,588	3,264,850	20.32	13.05
	511	21	1,568,770	131,408	2,727,480	15.44	24.06	11	411,454	3,391,454	19.56	14.66
	1023	22	2,880,520	67,232	1,445,488	15.33	44.89	12	369,497	3,505,908	18.92	14.97
ckt-5	255	20	574,260	524,193	10,385,880	12.06	12.62	10	816,757	6,079,410	21.73	16.18
	511	21	1,363,323	264,546	5,452,587	19.39	23.81	11	677,339	6,462,555	20.44	17.74
	1023	22	3,347,556	137,172	2,910,006	21.11	43.82	12	662,548	7,178,544	18.41	16.63

[5] Semiconductor Industry Association, "2003 International Technology Roadmap for Semiconductors (ITRS)." [Online]. Available: http://public.itrs.net/Files/2003ITRS/Home2003.htm

[6] A. Khoche and J. Rivoir, "I/O bandwidth bottleneck for test: is it real?" in *Test Resource Partitioning Workshop*, 2002.

[7] H. Vranken, F. Hapke, S. Rogge, D. Chindamo, and E. Volkerink, "ATPG padding and ATE vector repeat per port for reducing test data volume," in *Proc. Int. Test Conf.*, 2003, pp. 1069–1076.

[8] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski, "Logic BIST for large industrial designs: Real issues and case studies," in *Proc. Int. Test Conf.*, 1999, pp. 358–367.

[9] B. Koenemann, "LFSR-coded test patterns for scan design," in *Proc. the European Test Conference*, 1991, pp. 237–242.

[10] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *IEEE Trans. Computers*, vol. 44, pp. 223–233, February 1995.

[11] C. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in *Proc. Int. Test Conf.*, 2002.

[12] A. A. Al-Yamani and E. J. McCluskey, "Built-in reseeding for serial BIST," in *Proc. IEEE VLSI Test Symp.*, 2003, pp. 63–68.

[13] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," in *Proc. Design Automation Conference*, 2001, pp. 151–155.

[14] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T. W. Williams, "A reconfigurable shared scan-in architecture," in *Proc. IEEE VLSI Test Symp.*, 2003, pp. 9–14.

[15] I. Bayraktaroglu and A. Orailoglu, "Decompression hardware determination for test volume and time reduction through unified test pattern compaction and compression," in *Proc. IEEE VLSI Test Symp.*, 2003, pp. 113–118.

[16] M. Chandramouli, "How to implement deterministic logic built-in self-test (BIST)," *Compiler: A Monthly magazine for technologies world-wide, Synopsys*, January 2003.

[17] A. Chandra and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes," *IEEE Trans. Computers*, vol. 52, pp. 1076–1088, Aug. 2003.

[18] P. T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression," in *Proc. DATE Conf.*, 2002, pp. 604–611.

[19] A. Würtenberger, C. S. Tautermann, and S. Hellebrand, "A hybrid coding strategy for optimized test data compression," in *Proc. Int. Test Conf.*, 2003, pp. 451–459.

[20] M. Tehranipour, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique with application to reduced pin-count testing and flexible on-chip decompression," in *Proc. DATE Conf.*, 2004, pp. 1284–1289.

[21] A. Würtenberger, C. S. Tautermann, and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections," in *Proc. Int. Test Conf.*, 2004, pp. 926–934.

[22] M. Nourani and M. H. Tehranipour, "RL-Huffman encoding for test compression and power reduction in scan applications," *ACM Trans. Design Automation of Electronic Systems*, vol. 10, Jan. 2005.

[23] L. Li, K. Chakrabarty, and N. A. Touba, "Test data compression using dictionaries with selective entries and fixed-length indices," *ACM Trans. Design Automation of Electronic Systems*, vol. 8, pp. 470–490, Oct. 2003.

[24] A. Jas, J. Gosh-Dastidar, and N. A. Touba, "Scan vector compression/decompression using statistical coding," in *Proc. IEEE VLSI Test Symp.*, 1999, pp. 114–120.

[25] L. Li, K. Chakrabarty, S. Kajihara, and S. Swaminathan, "Efficient space/time compression to reduce test data volume and testing time for IP cores," in *Proc. IEEE Int. Conf. VLSI Design*, 2005, pp. 53–58.

[26] F. G. Wolff, C. Papachristou, and D. R. McIntyre, "Test compression and hardware decompression for scan-based SoCs," in *Proc. DATE Conf.*, 2004, pp. 716–717.

[27] T. Hiraide, K. O. Boateng, H. Konishi, K. Itaya, M. Emori, H. Yamanaka, and T. Mochiyama, "BIST-Aided scan test - a new method for test cost reduction." in *Proc. IEEE VLSI Test Symp.*, 2003, pp. 359–364.

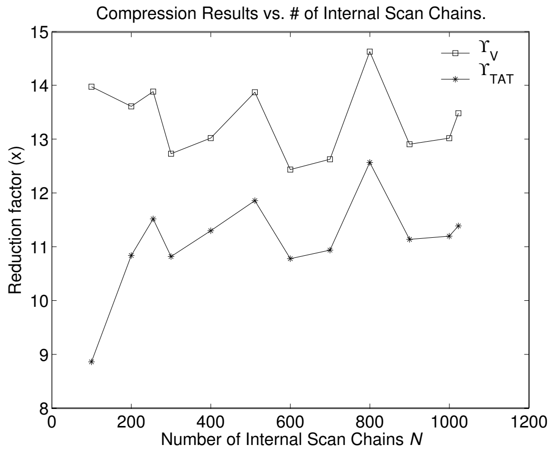
[28] J. Rajski, J. Tyszer, C. Wang, and S. M. Reddy, "Convolutional compaction of test responses," in *Proc. Int. Test Conf.*, 2003, pp. 745–754.

[29] J. H. Patel, S. S. Lumetta, and S. M. Reddy, "Application of Saluja-Karpovsky compactors to test responses with many unknowns," in *Proc. IEEE VLSI Test Symp.*, 2003, pp. 107–112.

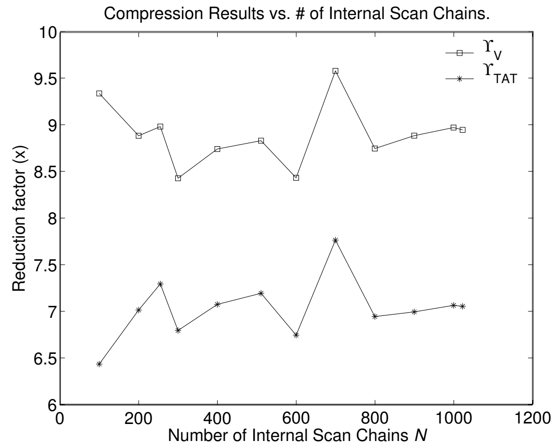
[30] K. Miyase and S. Kajihara, "XID: Don't care identification of test patterns for combinational circuits," *IEEE Trans. CAD*, vol. 23, pp. 321–326, Feb. 2004.

[31] M. Naruse, I. Pomeranz, S. M. Reddy, and S. Kundu, "On-chip compression of output responses with unknown values using LFSR reseeding," in *Proc. Int. Test Conf.*, 2003, pp. 1060–1068.

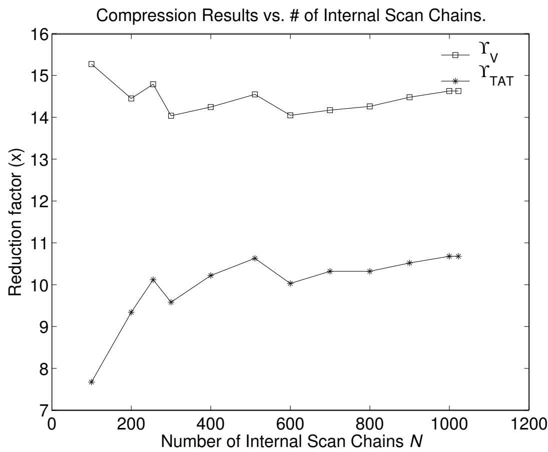
[32] S. M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data volume reduction for multiple scan chain designs," *ACM Trans. Design Automation of Electronic Systems*, vol. 8, pp. 460–469, Oct. 2003.



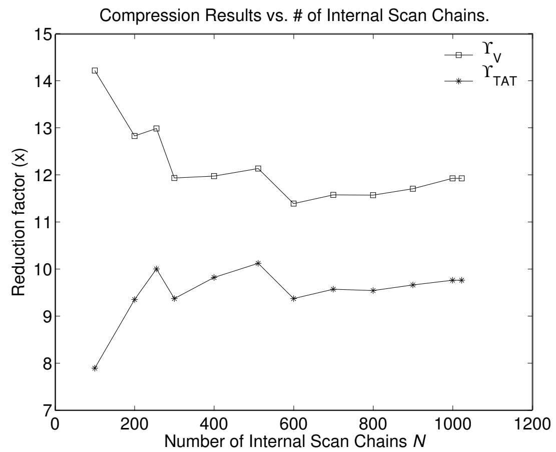
(a) ckt-1



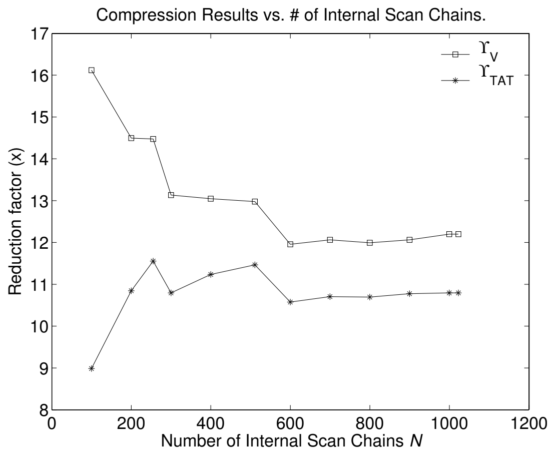
(b) ckt-2



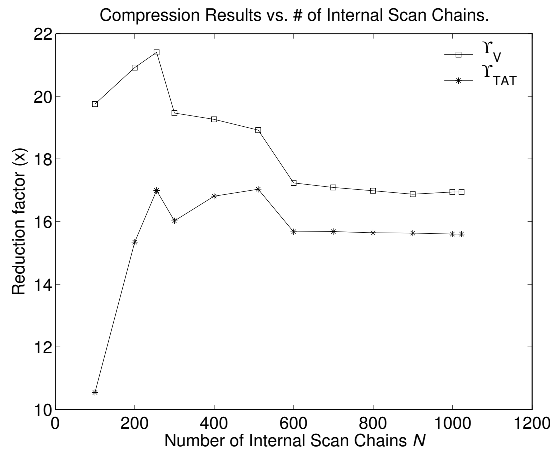
(c) ckt-3-2000



(d) ckt-3-1000

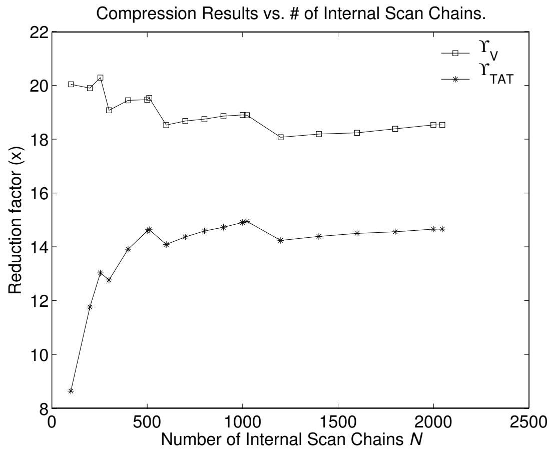


(e) ckt-3-500

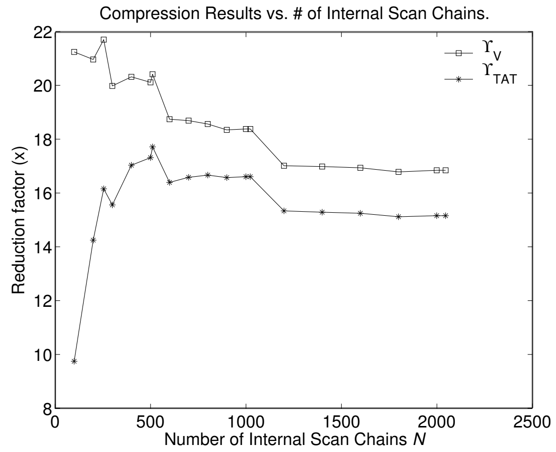


(f) ckt-3-200

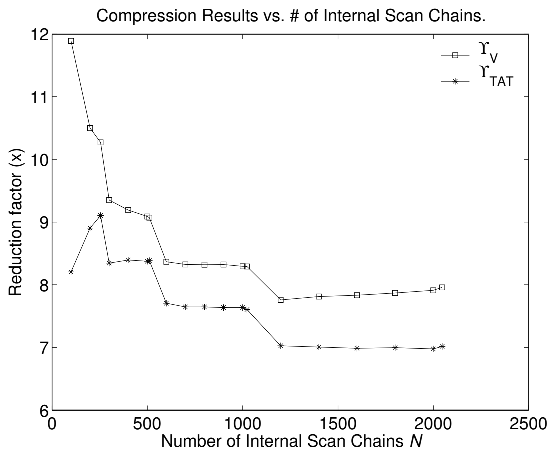
Fig. 10. Reduction in test data volume and TAT for varing number of internal scan chains for the smaller circuits.



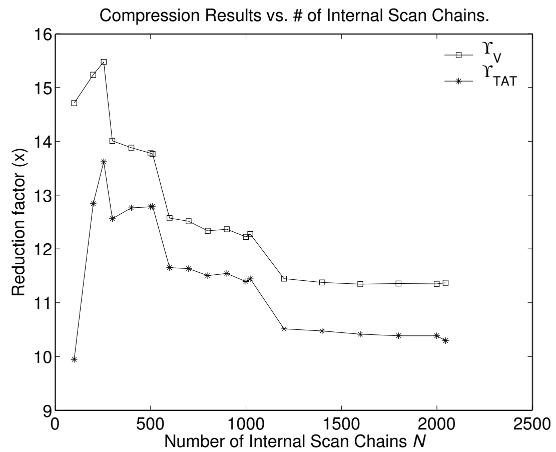
(a) ckt-4



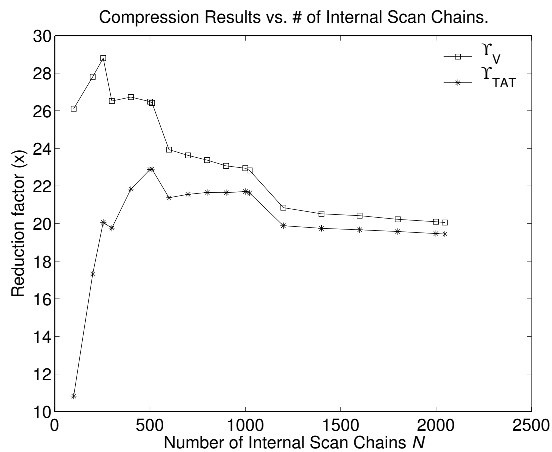
(b) ckt-5



(c) ckt-6



(d) ckt-7



(e) ckt-8

Fig. 11. Reduction in test data volume and TAT for varing number of internal scan chains for the larger circuits.