# Data Compression for System-on-Chip Testing using ATE

F. Karimi++, W. Meleis, Z. Navabi+, and F. Lombardi+ (contact author)
++ LTX Corporation
3930 North First Street
San Jose CA 95134
+Department of Electrical and Computer Engineering
Northeastern University
Boston MA 02115

Tel : (617) 373-4159(FL), Fax : (617) 373-8970
Email :{lombardi}@ece.neu.edu

**Abstract**

*The manufacturing test of Systems-on-Chip (SoC) requires new design considerations for automatic test equipment (ATE). Compression has been used in ATE to reduce storage and application time for high volume data by exploiting the repetitive nature of test vectors. Furthermore, the availability of boundary scan and the stringent integration requirements in the design of a head in an ATE necessitate a hardware-based technique which does not impact performance due to an excessive time complexity.*

*The application of a binary compression method to an ATE environment for manufacturing test is studied using a technique, referred to as Reuse. In Reuse, compression is achieved by partitioning the vector set and removing repeating segments. This process has $O(n^2)$ time complexity for compression (where $n$ is the number of vectors) with a simple hardware decoding circuitry. It is shown that for industrial SoC designs, the efficiency of the Reuse compression technique is comparable with sophisticated software compression techniques with the advantage of easy and fast decoding (decoding is performed using shift registers which can be incorporated into the boundary scan or the head). Two decompression schemes are presented.*

*Index Terms: test compression, ATE, manufacturing test.*

## 1: Introduction

Test application time is one of the main sources of complexity in testing IP cores as commonly found in Systems-on-Chip (SoC). Vectors (in either random or deterministic forms) are either generated by on-chip hardware for on-line testing, or provided by an external tester such as an ATE for manufacturing test. For testing logic blocks, the feasibility of on-chip test generation is mainly restricted to (pseudo) random methods. Area overhead for dedicated on-chip deterministic vector generation in general is cost prohibitive for manufacturing test. Random vectors are usually generated using Linear Feedback Shift Registers (LFSR), so no storage is required. Response evaluation is performed using a signature analyzer that compacts test responses into a signature and compares it with the signature of an error-free reference design. LFSRs introduce a small area overhead; however, test by random vectors requires a long application time due to modest quality. Hybrid schemes are commonly used to reduce test time by reseeding the LFSRs.

Compared with random testing, deterministic vectors such as those generated by ATE are designed to detect a set of target faults. This significantly reduces the number of vectors required for large designs. Due to the high overhead incurred in on-chip generation, deterministic vectors are usually provided from an external ATE source. Application of external test data (as for manufacturing test) involves downloading vectors from a storage device to an user interface workstation attached to an ATE. Vectors are transferred from the ATE to a main pattern memory in the head of the ATE through high-speed internal buses and then applied from the ATE memory to the device-under-test (DUT). The two major components of the time overhead are the download time (from the storage device to the ATE) and the test application time (from the pattern memory of the head to the DUT).

Compression has been investigated for resolving some of the problems in digital testing. Lossless compression is the process of encoding test vectors so that the original data can be uniquely reconstructed by a decoder. A fundamental issue of lossless compression is to decompose a data input set into a sequence of events, then to encode the events using as few bits as possible. While compaction reduces the number of vectors at generation time, data compression has a different objective, namely to encode test data (corresponding to the compacted vectors) with the minimum number of bits.

Compression can be achieved provided data contains *redundancy.* A repetitive nature can be expected from test data. For example, in an environment for testing a SoC on an ATE, few cores can be tested at a time, the remaining cores are inactive to enhance controllability and observability. This means that only test data for the part being tested changes, while the remaining parts of the SoC are either disabled or receiving a fixed set of data. Due to ATE related requirements (such as channel synchronization) DUTs may receive a fixed set of data in many consecutive patterns on certain inputs (for example, data inputs of an ALU remain fixed while mode select inputs change during ATE test). Therefore, the fixed part of test data can be considered as a prime candidate for compression.

Another example of compression in an ATE environment occurs during the functional testing of a random access memory (as commonly found in a SoC). Functional tests for memories often consist of several passes, within each pass a fixed pattern of data is written (read) to (from) all memory cells. During a row (column) sweep, cells in a row (column) are addressed consecutively, so parts of the memory addresses remain fixed. Considerable amount of compression can be achieved for memory testing by exploiting the highly repetitive nature of test data.

For SoC designs due to the limitation in the number of I/O pins at the periphery, low-cost serial (boundary scan) paths are the main access paths to embedded pins and IP cores. The pin count is one of the main causes of speed degradation for test data transfer across the chip.

In general for manufacturing test, deterministic solutions for SoC face many challanges such as a high volume of test storage, long test application time through the serial paths and vector sets that are generated by third parties (i.e. from the IC core providers) with limited information. This is made more complicated by the high density integration encountered in the design of the head in an ATE. A compression process for an ATE environment is substantially different from the traditional testing scenario.

- Compression is required to reduce the amount of data traffic between the ATE station and the head.
- Storage in the head is limited because memory utilization must be simultaneously allocated to many test channels (pins).
- The process of data coding/decoding must be lossless so that fault coverage of the test process is not affected.
- While compresion is usually performed by the computational back-end unit in the ATE (such as a workstation), decompression is performed in the head. Decompression must be fast (on-the-fly).
- Independently of the method used (Figure 1 shows on-chip and off-chip decoding methods within an ATE architecture), hardware due to the decompression unit (decoder) and synchronization with the operation of the head must be simple and DUT independent.

In this paper we presents a novel application of data compression to ATE design. A compression technique referred to as *Reuse*, is proposed for System-on-Chip (SoC) testing using automatic test equipment (ATE) . This technique can be incorporated in the design of the head for an ATE to reduce storage and test time requirements. Moreover it leads to simple hardware implementations for off-chip and on-chip decompression.

In the proposed method, compression is achieved by partitioning the test data into blocks and removing repeating segments from the individual blocks. To reduce the amount of data, a partitioning method with $O(n^2)$ time complexity is presentd (where $n$ is the number of vectors). A comprehensive analysis of the proposed compression method is also pursued. Two hardware decompression schemes are presented. Using

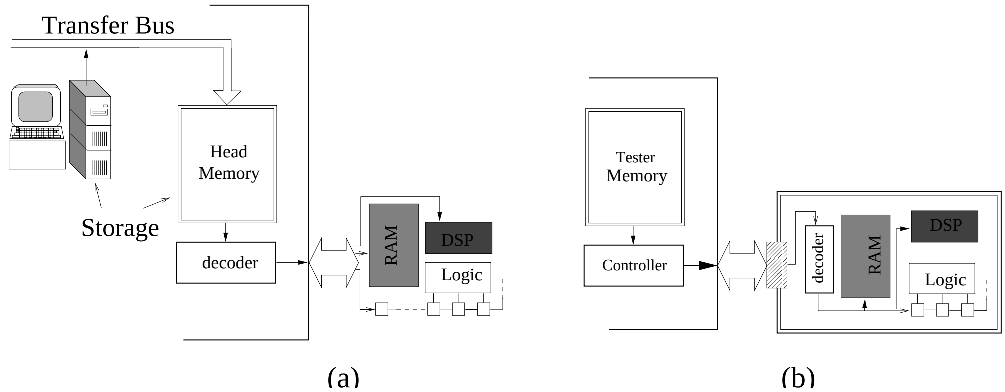



**Figure 1. ATE configuration . (a) off-chip decoding, (b) on-chip decoding**

*Reuse*, decoding is simple and performed by shift registers.

This paper is organized as follows. In the next section, a brief review of compaction and compression methods is presented. The basic principles of the proposed method (*Reuse*) are outlined in Section 3. Section 4 presents compression with fixed ordering; Section 5 deals with an analysis of the proposed method. Section 6 extends compression to reordering of test vectors. Hardware issues related to the decoder implementation of the proposed method are discussed in Section 7. Experimental results for industrial SoC designs are presented in Section 8. Conclusions are given in Section 9.

## 2: Review

Reduction in test time can be accomplished using different methods such as *test compaction*, *pseudo-deterministic* vector generation and *data compression*. The advantage of using test compaction is two-fold: it reduces storage requirements and decreases the test application time (in this case a smaller number of vectors is applied to a DUT). Static and dynamic compaction techniques have been used to reduce the number of test vectors with the objective of no loss in fault coverage. Static compaction is employed after test generation. Dynamic techniques perform compaction [1, 2] concurrently with the test generation process by often modifying the test generator. Most of these techniques require a high overhead in hardware which can not be met in current designs for the head in an ATE. Much work has been reported on compaction (for example on partially unspecified test vectors); these techniques are more applicable to vector generation than ATE design so they are not dealt furthermore as their objectives are beyond the scope of this paper.

Pseudo-deterministic methods have been considered as intermediate solutions to benefit from the lower hardware complexity of random generators and shorter test application time compared with deterministic methods. [4] uses an accumulator based structure to generate a set of deterministic tests. A run of the accumulator based generator is characterized by the initial value of the accumulator output ($r_0$) and a seed vector ($c$). The objective is to find a set of $(r_0, c)$ pairs so that the concatenation of executions initiated by $(r_0, c)$ pairs, cover the test set with a minimum number of vectors. These techniques are seldom used in ATE because test quality is rather limited.

One of the major challanges facing ATE designers is data management due to high volume. Application of various data compression techniques for test data has been investigated in [6]. Run-length coding, Huffman coding, Lempel-Ziv and arithmetic coding are some of the compression techniques found in the literature. In run-length coding, a sequence of symbols is encoded into two elements, the repeating symbol and the length of sequence. Run-length coding is efficient for data with long sequences of equal symbols.

Huffman coding is more sophisticated than run-length coding. It uses a table of frequencies of occurrence to build up an optimal representation of each character as a binary string. Compression is accomplished by giving frequent characters short codewords and infrequent characters long codewords. [13] has presented a method for using embedded processors to aid the test process of other components. The basic principle is that the tester loads a program along with compressed data into the on-chip memory. The processor executes the program which decompresses the data and applies it to scan chains in the other components of the SoC. The application of this method is restricted by the assumption of availability of some functional units within the ATE architecture. Current trends in head design are directed to increase the number of channels rather than on providing extensive on-situ processing capabilities.

[16] introduces a scheme for compression/decompression of test data using cyclic scan chains. It captures repeated patterns in a given set of vectors by applying a run-length coding technique to the difference vector between consecutive vector pairs (rather than test vectors by themselves). Adjacent repeated segments result in long runs of zeros in the difference vector which are effectively encoded by run-length coding. However, cyclic and non-contiguous repeated patterns can not be exploited by [16]. Compressed scan data is serially transferred from a tester to a chip at the clock speed of the tester and are decompressed with an internal decoder. Due to the on-chip data expansion, [16] employs a faster internal clock to avoid data overrun. Synchronization in the ATE however is problematic due to multiple clock operation.

The Lempel-Ziv (LZ) [8, 9] method is based on the construction of a dictionary and builds a list of patterns. The LZ method encodes patterns according to their indices in a list. The LZ method does not require a-priori knowledge of the probabilities of occurrence, and becomes more efficient for long sequences of patterns whose characteristics are static. The Lempel-Ziv-Welch (LZW) [10] algorithm is a variation of the LZ method; it collects new sequences (referred to as phrases) into a dictionary. When a repeating phrase is found, then the index of the phrase in the dictionary is recorded to compress the phrase. This approach requires a considerable amount of storage thus having a limited applicability for large industrial designs.

[6] has presented the application of the Burrows-Wheeler (BW) Transform to test vectors. It is based on the observation that consecutive vectors in a data set tend to be repetitive. The existence of cyclic patterns is also the characteristic which is captured by the BW Transform. In [6] test data is represented as a matrix and divided

into sub-matrices of equal size; then the BW Transform is applied to the individual columns of each sub-matrix. The activity of a column is defined as the number of transitions made in that column. A transformed matrix $E_i$ consists of columns in the BW Transform of the sub-matrix $D_i$ (in which activities are less than a defined threshold level). If the activity of a column in the BW Transform is equal or greater than the activity of its corresponding column in $D_i$, then the original column in $D_i$ appears in $E_i$. In the next step of the compression process, a modified version of run-length coding is applied to the individual columns of $E_i$. COMPACT [7] further improves compression by applying a run-length coding for low activity data and GZIP for data with high activity. High compression rates have been reported by [6, 7], thus making the application of the BW Transform suitable for data. However, as the decoding process is slow and complex for hardware implementation, the application of BW is limited to the download stage in which decoding is performed by software.

A compression/decompression technique based on statistical coding has been presented in [5]. Statistical codes are selected so that decompression can be achieved by a pipeline decoder. [15] introduces the application of Golomb [17] coding for test data. Golomb and the related Rice [18] coding methods are based on data with exponentially distributed run lengths. The Golomb and Rice methods consist of a family of parameterized codes which can be estimated adaptively [14], thus giving good compression. Pipeline effects (and its implication on synchronization for the multi-pin operation of the head) restrict this technique to circuits with low operating frequencies.

## 3: Basic Principles

Numerous studies have shown that test sets have unique features. Consecutive vectors have a common repetitive part (referred in this paper as a *segment*) which may extend over several non-contagious bits. We will refer to this data (identified by a repeating segment) as a *block*. In our scheme, test compression is equivalent to the process of finding a set of *non-overlapping blocks* covering possibly the entire test set. Repeating segments (specified once and reused for all vectors in a block) are used to reduce the size of the data set.

This section presents the basic principles of the proposed approach to compression. This is based on the so-called *Reuse* compression method; *Reuse* relies on the observation that consecutive vectors tend to be repetitive. Data vectors are assumed to be fully specified with fixed order. Figure 2 shows an example of segments in a data set that can be compressed into a smaller number of test vectors. Six vectors are shown in the upper part of this figure. Each line represents a test vector (which is serially shifted into a scan chain). Data shown in each shaded box remains unchanged in all six test vectors, hence these bits are candidates for compression.

In our scheme, compression consists of partitioning data by considering non-overlapping blocks of vectors (each of these vectors shares at least a segment). Binary compression of each block is then achieved by removing the segments (fixed and shared data) and shifting the remaining bits (referred to as the body of the block) into the DUT.

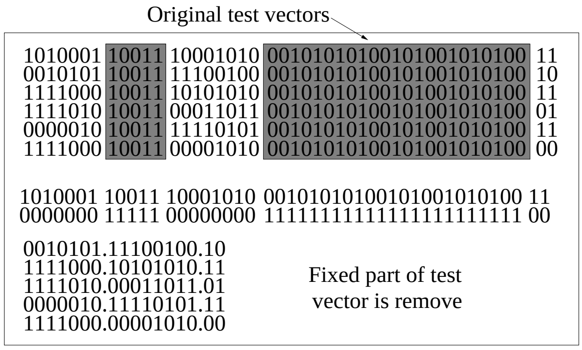



**Figure 2. Removing repetitive data patterns, test reuse**

In the lower part of Fig. 2, the first vector is the same as the original (uncompressed) vector. However, the shaded part (shown only once) is eliminated from the other five vectors (the part that is eliminated from each vector is shown by a dot). The first vector of each block is followed by a *mask* vector with 1's appearing at the bit positions of the segments and 0's at the other bit positions. These two vectors provide the information required for the decompression of each block.

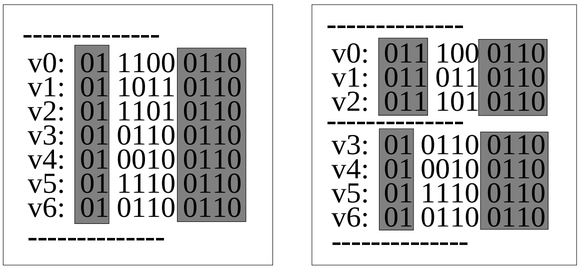
The 1-strings in the mask vector correspond to the repeating segments within a block. The decompression process is simply performed by replicating segments of the first vector (corresponding to the 1-string segments of mask vector) and by placing them into the appropriate locations of the compressed vectors (indicated by dots in Fig. 2).

This approach relies on finding the segments which are repetitive within a number of test vectors. Vectors can be partitioned in many ways as corresponding to possible blocks and each partition can result in a different compression rate. For example, Fig. 3(a) and (b) show two possible partitions of test vectors v0 through v6. In the first partition, one block covers the entire test set; in the second partition, two blocks are used. The G1 partition consists of 42 shared bits; G2 utilizes 21 shared bits for the (v0, v1, v2) block and 24 bits for the (v3, v4, v5, v6) block, thus requiring a total of 45 bits for the entire test set. However, G2 leads to a lower compression (7%) because it requires one additional decoding vector than that of G1(37%).

The overall compression rate is determined by the way that the vectors are divided (partitioned) and blocks are identified. Due to its characteristics it is shown that the proposed process using dynamic programming has a polynomial time complexity.

## 4: Partitioning for Fixed Order Compression

Let $V$ be an ordered set of vectors $\{v_1, v_2, ..., v_n\}$; let also a block $P_l$ represent the vector set $\{v_i, v_{i+1}, ..., v_j\}$. $S(i,j)$ or $S_{P_l}$ is defined as the number of bits after compression of $P_l$. $S(i,j)$ is determined by identifying a shared segment. The cost of a mask vector (which is required for reconstructing the original test vector within a block) is added to $S(i,j)$. The bit size of the mask is the same as for a test vector. The block set of the generated partition is given by $P$=$\{P_1, P_2, ..., P_k\}$ and is formally




(a) G1 (b) G2

**Figure 3. Two partitioning configurations.**

defined in this paper as an optimal partition of $V$ if:

1. $\bigcup_{i=1}^{k} P_i = V$.
2. $\sum_{i=1}^{k} S_{P_i}$ is minimal.

The key observation in data compression for an ATE is that if the first partition of the vectors is given by $\{v_i, ..., v_k\}$ (i.e. these vectors are not partitioned further), then the vectors $\{v_{k+1}, ..., v_n\}$ must be also partitioned, i.e. a block of $V$=$\{v_i, ..., v_n\}$ contains optimal partitions of smaller subsets of vectors. This is based on the following conditions:

1. Blocks are non-overlapping.
2. The number of bits remaining in the test vectors of each block is minimized.

These features permit to find a solution based on *dynamic programming.* Consider the vector set $V$=$\{v_1, ..., v_n\}$ and let $T(i)$ represent the total number of remaining bits in the vector set $\{v_i, ..., v_n\}$ when these vectors are partitioned. Let $T$=$\{T(i)\}$ (for $i = n, n-1, ..., 1$).

In the proposed method, the vectors are compressed by computing $T(i)$. $T(1)$ denotes the number of remaining bits after compression by *Reuse*. The problem of calculating $T$ can be expressed as follows:

$$T(i) = min_{k \in [i, n-1]} \{S(i,k) + T(k+1)\}$$

All $S$ terms can be computed in a time complexity of $O(n^2)$. $T$ can be computed recursively; during the recursive process , redundancy is present when calculating the values of $T(i)$, i.e. for example, $T(4)$ is computed while computing $T(1)$, $T(2)$ and $T(3)$. To avoid this unnecessary redundancy, we can compute $T$ in a bottom-up fashion. We can initialize $T(n)$ to the number of bits in a vector, and then compute $T(n-1)$, $T(n-2)$, and so on by using the above expression. At each step, all required terms have already been computed, thus the whole process has a complexity in $O(n^2)$. This complexity is also practical because its computation is encountered only once for a given set. Therefore, the *Reuse* algorithm is optimal in the sense that it searches for non-overlapping blocks and selects the combination that minimizes the number of remaining bits.

In *Reuse*, compression is achieved in a straightforward fashion. The vector set (denoted by $V$) is supplied as input; the indices of the mark points (for the partitions) are returned as output in an integer array (denoted by $P$). In the first part of the algorithm, the index that divides $V$ in two parts, is computed and stored in $P(i)$. In the second part of the algorithm, the partitions are selected by recursive indexing through $P$.

**Data** : $V$ input vector set
**Result**: $P$ location of partitions
**begin**
  $i \leftarrow n-1$; $k \leftarrow n-1$; $min \leftarrow 0$ ; $T(n) \leftarrow$ test vector length; ▷ compute $T(i)$'s, $i = n-1, ..., 1$ **for** $i = n-1$ **to** 1 **do**
    **for** $k = i$ **to** $n-1$ **do**
      **if** $\beta = S(i,k) + T(k+1) \; < \; min$ **then**
        $min \leftarrow \beta$ ; $j \leftarrow k$;
      **end**
    **end**
  **end**
  $T(i) \leftarrow min$; $P(i) \leftarrow j$ ;
  ▷ mark partitions $i \leftarrow 1$ **while** $i < n$ **do**
    $v_i, v_{i+1}, ..., v_{P(i)}$ is a selected partition; $i \leftarrow P(i) + 1$
  **end**
**end**

Algorithm 1: *Reuse*.

## 5: Analysis

A measure of the efficiency of a binary compression process is given by the compression rate. Let $n$ be the number of vectors and $l$ the vector length. Let $s$ represent the length of the repeating segments in the $n$ vectors. Consider a block $M$ of size given by $nl$ bits.

The *compression rate* denoted by $r_R$ is then given by

$$r_R = 1 - \frac{2l + (n-1)(l-s)}{nl} \tag{1}$$

The nominator represents the number of bits remaining in the compressed block. It consists of two terms:

- $2l$ specifies the length of the first vector of the block ($l$) (appearing in its original form) and the size of the mask vector (as carrier of the decoding information).
- $(n-1)(l-s)$ represents the size of the decoded vectors ($v_2$ to $v_n$) and $l-s$ is the length of a vector after removing the repeating segments.

As an example, consider the following vectors:

$$v_1 : 1110100100010110$$
$$v_2 : 1111000100100101$$
$$v_3 : 1111100100010111$$
$$v_4 : 1111100100000001$$

So, $n = 4$, $l = 16$ and $s = 9$, therefore $r_R$ is given by:

$$r_R\% = (1 - \frac{2 \cdot 16 + (4-1)(16-9)}{4 \cdot 16}) \times 100 = 17$$

*Reuse* can be compared with Huffman's coding. The compression rate for Huffman's coding is rather complex to establish due to the following reasons:

- Unlike *Reuse*, the compression rate depends on the contiguous nature of the repeating segments. For example the rates for the pairs ($v_1 : 1110100100010110, v_2 : 1111000100100101$) and ($v_1 : 1110100100010110, v_2' : 1110100100101001$) are different, eventhough the number of bits equal in both pairs is 10.
- The length of the segments which repeat between consecutive vectors, may change.
- Compression is dependent on the size of the partitions (denoted by $B$).

The upper bound can be found by assuming that the repeating segments are contiguous. In the above example when calculating the upper bound it is assumed that the difference vector $v_1 \oplus v_2 = 0001100000110011$ consists of a string of seven 0's. For consecutive vector pairs, repeated segments are assumed to have a maximum length given by $s_{max}$ (12 bits of 0 value in $v_2 \oplus v_3$), i.e. in the exact calculation for a partition of size $B$, the most frequent pattern consists of $B$ 0's, therefore the maximum size of the encoded length (give by $s_{max}$) is computed as $\lceil \frac{length(s_{max})}{B} \rceil a_{min}$, where $a_{min}$ is the size of the smallest codeword mapped to the most frequent pattern. To calculate the upper bound (independently of the block size), we assume that the encoded size of $s_{max}$ is $a_{min}$. Then,

$$r_{uH} = 1 - \frac{l + (n-1)(l - s_{max} + a_{min})}{nl} \tag{2}$$

In practice, for large values of $n$ and small values of $a_{min}$:

$$r_{uH} \approx \frac{s}{l} - \frac{a_{min}}{l} \rightarrow \frac{s}{l}$$

For the example, $s_{max} = 13$, and $r_{uH}\% = \frac{13}{16} \times 100 \approx 82$ %. $r_H$ is 32% (for $B$=4) and is 66% (for $B$=8).

Figure 4 shows $r_R$ versus the normalized ratio ($\frac{s}{l}$) for different values of $n$. For $n = 2$, $r_R$ is negative, because the overhead of the added decoding information of the (mask) vector to the compressed block is bigger than the size of the repeating segments. For $n = 3$, positive compression is achieved for values of $\frac{s}{l}$ greater than 0.6. For larger values of $n$, positive compression is achieved with smaller values of $\frac{s}{l}$ and $r_R$ is closer to the upper bound (as indicated by the dashed line).

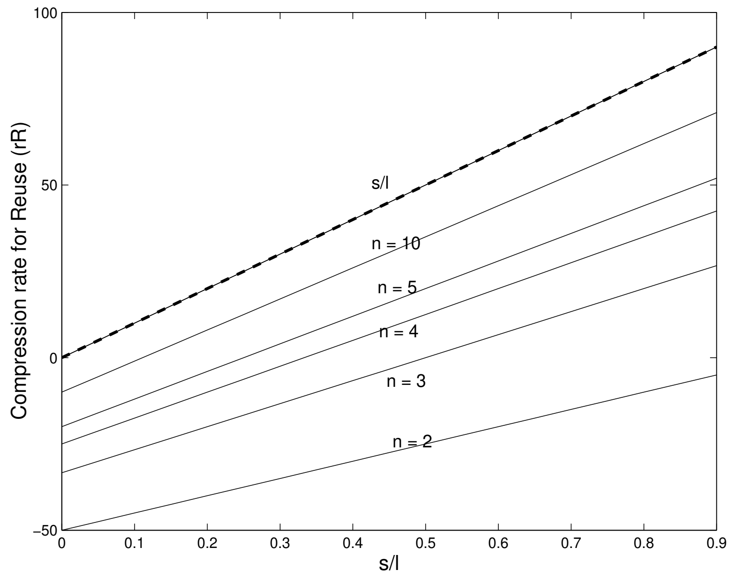



**Figure 4.** $r_R$ **versus** $s/l$

## 6: Compression by Reordering

A further improvement for compression is given by reordering, i.e. prior to compression, test vectors are reordered such that vectors whose bit patterns are similar, are grouped together. Within an ATE, this additional processing step can take place at the back-end computational unit (usually a workstation). The *Hamming distance* is used as measure of similiarity between vectors (referred to as *correlation*); the Hamming distance between two vectors ($v_i$ and $v_j$) of length $n$ is defined as:

$$H(v_i, v_j) = \sum_{l=1}^{n} v_i[l] \oplus v_{i+1}[l] \tag{3}$$

The objective is to find a new ordering of the test set that minimizes the sum of the Hamming distances between consecutive vectors so that the size of the blocks can likely be increased. Reordering can be mapped to the problem of finding a minimal tour for the Traveling Salesman Problem (TSP) [12].

To perform the mapping between compression by reordering and the TSP, each vector in the set is associated to a node in the graph; the distance between each pair of nodes is given by the weight of the arc connecting the nodes (i.e. the weight is their Hamming distance). Then an utilitity routine (referred to as the TSP solver) is applied to find the tour. TSP is known to be a NP problem, i.e. the time complexity of any optimal algorithm is believed to vary exponentially with the number of nodes in the tour. There are heuristic algorithms that provide good solutions in reasonable time to solve the TSP.

## 7: Hardware Issues for Decompression

Decompression hardware is required in the ATE. To reconstruct the original test data from its compressed form, in addition to the body of the compressed block, the patterns and values of the repeating segments must be supplied with each block (the first two vectors in the lower part of Figure 2). Two implementations are presented for hardware decoding of the compressed test vectors in a serial channel.

Figure 5 illustrates the design for the first proposed decoder. It consists of scan and mask registers and a direction selection switch $S$. The switch directs the flow of serial data into the scan or mask registers (test vectors are distinguished from mask vectors by an extra bit added to each vector; this is 0 for test vectors and 1 for mask vectors.) At the beginning of a compressed block, $S$ directs the first vector into the scan register. After completely shifting the first vector, $S$ directs the mask bits into the mask register. Then the switch is reset back to its position and left in this state until all remaining bits of a block are shifted. After this process, due to a circulation path, the mask register obtains its initial setting and the scan register restores the bits in the shared segment locations.

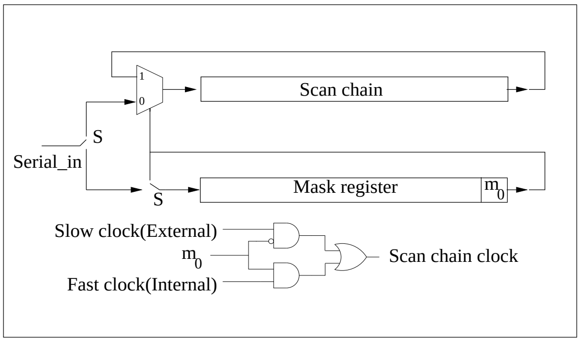



**Figure 5. Decompression hardware**

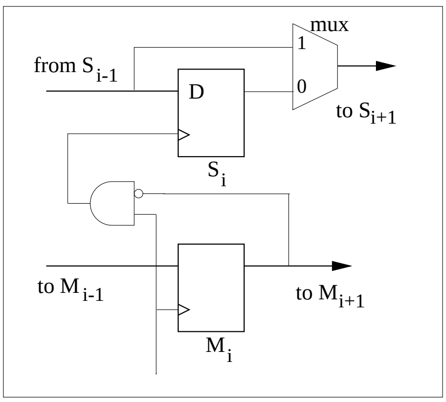



**Figure 6. A modified scan cell.**

Figure 6 illustrates another hardware structure for decoding; in this structure, scan cells are modified for operation in both transparent and normal modes. The operation mode of each scan cell $S_i$ is controlled by its corresponding mask cell $M_i$. A scan cell in its normal mode functions as a regular cell in the shift register structure. For a cell $S_i$ in transparent mode, the input data is directly propagated to the output with no latching. During decompression, cells corresponding to bits in shared segments are set to the transparent mode (the remaining cells are left in normal mode).

For the case of embedded cores in which each core is surrounded by a wrapper, the decompression hardware can be merged with the boundary scan cells as shown in Figure 6. In this arrangement, user-defined instructions under the IEEE 1149.1 standard are used to select between mask and data registers. Compared with the first proposed scheme, this structure does not require an internal scan clock but at the expense of introducing additional hardware overhead.

The overhead of the proposed schemes can be compared with the ones encountered in other compression schemes. Hardware complexity of statistical coding (similar to the Huffman's coding) has been analyzed in [5]. The decoder is a Finite State Machine (FSM) which is placed at the serial input of the chip. Although an FMS-based decoder does not cause a significant overhead, it faces the following difficulties in the context of ATE head design:

- Dependency of decoder design to the input vector set because in statistical compression, coding is performed based on the pattern and frequency of repeating segments in the input data.
- Tight synchronization between the fast internal and slow external clocks because statistical coding is based on a parametrized analysis, i.e. at the decoder output, data is expanded by varying lengths.

Encoding in *Reuse* is not data dependent. Also, synchronization is not an issue for the decoders in Fig. 5 and 6 . For the decoder in Fig. 5 data is clocked into the scan register at two different rates. Bits in shared segments are shifted by using a fast internal scan clock $CLK_I$; other bits are scanned using a slow external tester clock. Bit $m_0$ in the mask register controls the scan clock and the flow of data into the scan chain. The salient feature of the second decoder design is a shorter test time without employing a faster clock; this is accomplished by bypassing the scan cells at the location of the repeating segments in a block.

## 8: Experimental Results

Experiments were run on industrial SoC designs. In the experiments, the initial test set was reordered using the `tsp_solve-1.3.6` package. This package supports a variety of optimal and heuristic solutions to TSP. In the experiment, `tsp_solve` has been run with the `heur` option which invokes one of the built-in heuristic conditions as part of the path finder process. Huffman compresion is applied in the form of $T{=}t_1, t_1 \oplus t_2, t_2 \oplus t_3, ...,$ (where $\{t_1, t_2, ...\}$ is the reordered vector set). The binary data vector is broken into $b$-bit segments ($b = 4, 8$) and each segment is replaced with its integer representation; Huffman coding is then applied to the integer data

| Group | Run-length | Group Prefix | Codeword |
|---|---|---|---|
| $A_1$ | 0 | 0 | 000 |
| | 1 | | 001 |
| | 2 | | 010 |
| | 3 | | 011 |
| $A_2$ | 4 | 10 | 1000 |
| | 5 | | 1001 |
| | 6 | | 1010 |
| | 7 | | 1011 |
| $A_3$ | 8 | 110 | 11000 |
| | 9 | | 11001 |
| | 10 | | 11010 |
| | 11 | | 11011 |

**Table 1. Golomb codewords for $m = 4$.**

vector. Although the compression efficiency increases with $b$, the exponential growth in decoder complexity is $O\ (2^b)$, thus limiting the increase to a segment size of at most 8.

Table 1 shows the Golomb codewords for $m = 4$. Run-lengths are coded in groups of size $m$. Groups are uniquely identified by group prefixes. The group prefix for $A_k$ is defined as $1^{(k)}0$ ($k$ ones followed by a zero). A codeword for the run-length $r$ in $A_k$ is formed by the concatenation of the group prefix with the binary representation of the $r$ index in $A_k$.

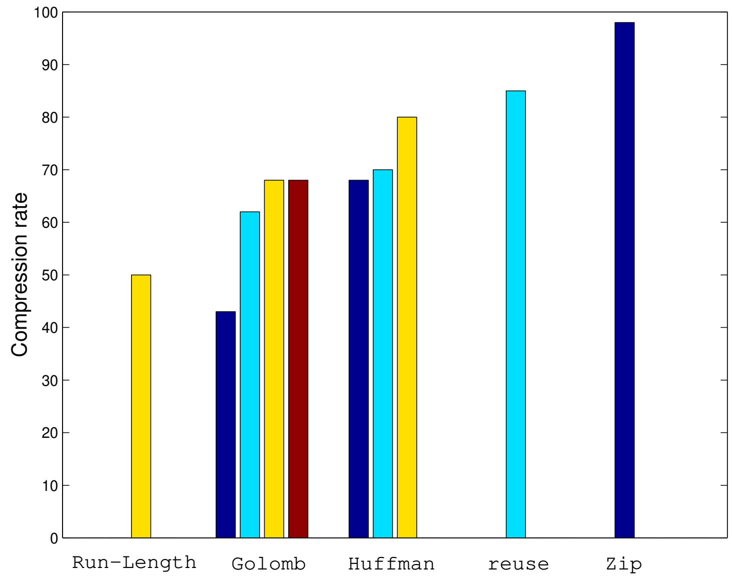



**Figure 7. Average compression rates obtained from the application of Run-length, Huffman ($b = 4, 6, 8$), Golomb ($m = 2, 4, 8, 16$), Reuse and Unix-Zip.**

The results for the Run-length coding are obtained using the variable-to-fix length

|  | Min. | Av. | Max. |
|---|---|---|---|
| Run-length | 49.9 | 50 | 52.7 |
| Golomb ($m = 16$) | 66.8 | 68.2 | 70.2 |
| Huffman ($b = 8$) | 76.2 | 79.8 | 81.4 |
| Reuse | 81.2 | 84 | 93.7 |
| Zip | 87.2 | 97.1 | 99.8 |

**Table 2. Minimum, average and maximum compression rates in %.**

mapping shown in Table 3. Encoding involves the replacement of variable length patterns of the first column of Table 3 by the corresponding codewords from the second column.

Industrial SoC test data (provided by LTX) was used in the experiments. This data consists of 343 files, the size of each file (in binary format) is 60272 bytes. A byte in the binary file represents 8 bits of a test sequence. The test sequence is composed of serial test data mixed with controlling sequences. Table 2 shows the minimum, average and maximum compression rates for different approaches, including *Reuse*.

Fig. 7 compares the average compression rates obtained from the application of Run-length, Huffman (with $b = 4, 6, 8$), Golomb (with $m = 2, 4, 8, 16$), *Reuse* and the Unix-Zip compression techniques to the industrial test sequences. These results present a different conclusion than the benchmark circuits: In *Reuse* compression is achieved by removing repeating segments; in Run-length, Huffman and Golomb coding techniques repeating segments are not removed, they are coded to segments with smaller sizes. Therefore, *Reuse* outperforms them by a significant margin when the data set contains long runs of repeating patterns as commonly encountered in industrial SoC designs.

## 9: Conclusion

This paper has presented a novel approach to data compression within an automatic test equipment (ATE) environment for System-on-Chip (SoC) testing. A compression technique, so called *Reuse*, has been proposed. This technique can be incorporated in the design of the head for a ATE to reduce storage and test time requirements. Moreover it leads to simple hardware implementations for off-chip and on-chip decompression.

In the proposed method, compression has been achieved by partitionig the test data into blocks and removing repeating segments from the individual blocks. To reduce the amount of data, a partitioning method with $O(n^2)$ time complexity has been presentd (where $n$ is the number of vectors). For highly correlated industrial data, simulation has shown that the compression rate of *Reuse* is better than compression techniques based on Huffman, Golomb and Runlength coding methods. Two hardware decompression schemes have been presented. Decoding is simple and is performed by shift registers to serial channels.

| pattern | codeword |
|---|---|
| 11 | 0000 |
| 01 | 0001 |
| 01 | 0010 |
| 001 | 0011 |
| 0001 | 0100 |
| 00001 | 0101 |
| 000001 | 0110 |
| 0000001 | 0111 |
| 00000001 | 1000 |
| 000000001 | 1001 |
| 0000000001 | 1010 |
| 00000000001 | 1011 |
| 000000000001 | 1100 |
| 0000000000001 | 1101 |
| 000000000000001 | 1110 |
| 000000000000000 | 1111 |

**Table 3. Map table for Run-length coding with 16 codewords.**

## References


[1] S. T. Chakradhar and A. Raghunathan, "Bottelneck removal algorithm for dynamic compaction in sequential circuits" , *IEEE Transaction on Computer-Aided Design*,

[2] E. M. Rudnick and J. H. Patel,"Simulation-based techniques for dynamic test sequence compation", *Proc. Int. Conf. Computer-Adid Design*, pp. 67-73, November 1996.

[3] S. K. Bommu, S. T. Chakradhar and K. B. Doreswamy, "Static test sequence compaction based on segment reordering and accelerated vector restoration", *International Test Conference*, pp. 954-961, 1998.

[4] R. Dorsch, H. J. Wunderliech,"Accumulator based deterministic BIST", *International Test Conference*, pp. 412-421, 1998.

[5] A. Jas, J. Ghosh-Dastidar and N. Touba, "Scan vector compression/decompression using statistical coding," *17th VLSI test symposium*, pp. 114-120, 1999.

[6] T. Yamaguchi, M. Tilgner, M. Ishida, D. S. Ha, "An efficient method for compression test data", *International Test Conference*, pp. 79-88, 1997.

[7] M. Ishida, D. S. Ha and T. Yamaguchi, "COMPACT: A hybrid method for compression test data", *VLSI Test Symposium*, pp. 62-69, 1998.

[8] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression",

*IEEE Transaction on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

[9] J. Ziv and A. Lempel, “A compression of individual sequences via variable-rate coding”, *IEEE Transaction on Information Theory*, vol. IT-24, pp. 530-538, September 1978.

[10] T. A. Welch, “A technique for high-performance data compressionun”, *IEEE Computer*, vol. IT-17, No. 6, pp. 8-19, June 1984.

[11] J. A. Storer and T. G. Szymanski, “Data compression via textual substitution”, *Journal of ACM*, vol. 29, No. 4, pp. 928-951, October 1982.

[12] E.L. Lawler, A. H. Rinnooy-Kan, “The traveling salesman problem : a guided tour of combinatorial optimization,” John Wiley & Sons, 1985.

[13] A. Jas and Nur Touba, ”Using an embedded processor for efficient deterministic testing of systems-on-a-chip”, *Proceedings of the IEEE International Conference on Computer Design*, pp. 418-423, 1999.

[14] J. A. Storer and M. Cohn Eds., ”Fast and efficient lossless image compression”, *IEEE Data Compression Conference*, pp. 351-360, 1993.

[15] A. Chandra and K. Chakrabarty, ”Test data compression for system-on-chip using Golom codes”, *VLSI Test Symposium*, pp. 113-120, 2000.

[16] A. Jas and N. Touba, “Test vector decompression via cyclical scan chains and its application to testing core-based design,” *Proc. of International Test Conference*, pp. 458-464, 1998.

[17] S. W. Golomb, “Run-Length encoding”, *IEEE Trans. Inform. Theory*, IT-12, no. 4, pp. 399-401, 1966.

[18] R. F. Rice, “Some practical universal noiseless coding techniques”, Jet Propulsion Labaratory, Pasadena, California, JPL Publication 79-22, Mar. 1979.

[19] R. Gupta and M. A. Breuer, “ BALAST: A methodology for partial sacn design,” *Proc. 19th International Symposium on Fault Tolerant Computing,* pp. 118-125, June, 1989.

[20] D. Kagaris, S. Tragoudas, and A. Majumdar, “On the use of counters for reproducing deterministic test sets, *IEEE Trans. on Computers*, 45, 1405, 1996.

[21] A. Jagota, “An Adaptive, Multiple Restarts Neural Network Algorithm for Graph Coloring”, *European Journal of Operational Research*, second half of 1996.