Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression

Paul Theo Gonciari and Bashir M Al-Hashimi
Dept. of ECS, University of Southampton
United Kingdom
{p.gonciari,bmah}@ecs.soton.ac.uk

Nicola Nicolici
Dept. of ECE, McMaster University
Canada
nicola@ece.mcmaster.ca

Abstract

This paper proposes a new test data compression/decompression method for systems-on-a-chip. The method is based on analyzing the factors that influence test parameters: compression ratio, area overhead and test application time. To improve compression ratio*, the new method is based on a Variable-length Input Huffman Coding (VIHC), which fully exploits the type and length of the patterns, as well as a novel mapping and reordering algorithm proposed in a pre-processing step. The new VIHC algorithm is combined with a novel parallel on-chip decoder that* simultaneously *leads to* low test application time *and* low area overhead*. It is shown that, unlike three previous approaches [2, 3, 10] which reduce some test parameters at the expense of the others, the proposed method is capable of improving* all *the three parameters simultaneously. For example, the proposed method leads to similar or better compression ratio when compared to frequency directed run-length coding [2], however with lower area overhead and test application time. Similarly, there is comparable or lower area overhead and test application time with respect to Golomb coding [3], with improvements in compression ratio. Finally, there is similar or improved test application time when compared to selective coding [10], with reductions in compression ratio and significantly lower area overhead. An experimental comparison on benchmark circuits validates the proposed method.*

1. Introduction

Due to the increased complexity of System-on-a-Chip (SoC), testing is an important factor that drives time-to-market and hence the cost of the design [14]. In addition to the standard design for test (DfT) issues for SoCs [14], a new emerging problem is the large volume of test data [16]. This is because the cost of Automatic Test Equipment (ATE) grows significantly with the increase in speed (i.e. operating frequency), channel capacity, and memory size [16]. In order to support large volume of test data with limited channel capacity for future SoCs, ATE requires modifications and additional expenses. Besides, when using conventional test data transfer, the volume of test data has a direct impact on test application time, which adds to the manufacturing cost of SoCs. There are two main potential solutions for closing the ever-increasing gap between ATE speed, channel and memory requirements vs. SoC complexity. As shown in the ITRS Roadmap [1], the first solution is the well-known built-in self-test (BIST). However, to make the existing embedded cores *"BIST-ready"*, a considerable redesign effort is required, as well as performance penalties are incurred. Moreover, depending on the business model, many embedded cores can not be modified by the system integrator [16]. The second solution for reducing the speed, channel capacity and memory requirements of ATE is the use of test data compression/decompression methods. This solution does not introduce performance penalty and guarantees full reuse of the existing embedded cores, as well as the ATE infrastructure, with minor modifications required during system test preparation (compression) and test application (decompression). Hence, the second solution is an *efficient complementary alternative to BIST*, and requires further investigation.

Test data reduction received increased attention over the last several years. The methods for reducing test data range from test vector compaction [6, 15], test data compression methods which reduce the amount of data transmitted to the ATE [8] (e.g. with the test data stored on a workstation the amount of data transfered to the ATE is reduced), to methods which require an on-chip decompression architecture [2, 3, 5, 9–12]. To obtain not only test data reduction but also test application time reduction, when compared to already compacted test sets, an on-chip decompression architecture is required. A compression algorithm for test data should have two features: lossless and simple decompression. The first requirement is easily fulfilled by all known lossless compression strategies (i.e. Huffman coding, Lempel-Ziv compression algorithm, arithmetic coding [4]) however, to meet the second requirement the compression algorithm has to be carefully chosen. Iyengar et al. [9] proposed a new built-in scheme for testing sequential circuits based on sta-

tistical coding. Results indicate that the method is suitable for circuits with a relatively small number of primary inputs. To overcome this disadvantage a selective coding algorithm was proposed [10]. The method splits the test vectors into *fixed-length* block size patterns and applies Huffman coding to the newly created set. The method carefully selects the patterns which are coded using the Huffman codes while the rest of the patterns are prefixed. Although the method leads to fast on-chip decoding and hence low test application time, by extending the block size the on-chip decoder increases in complexity, and consequently leads to very high area overhead. In [11] test vector compression is performed using run-length coding, where the method relies on the fact that successive test patterns in a test sequence often differ in only a small number of bits. Therefore, the initial test set T_D is transformed in T_{diff}, a difference vector sequence, in which each vector is computed by the difference between consecutive vectors in the initial sequence. Despite its performance, the method is based on variable-to-fixed-length codes and, as demonstrated in [3], these are less effective than variable-to-variable-length codes. Chandra, et al. [3] introduced a compression method based on variable-to-variable-length Golomb codes. Both methods ([11],[3]) use the difference vector sequence. In order to regenerate the test set, inside the chip, a scan chain architecture called cyclical scan register (CSR) has to be used. Since Golomb codes are only dependent on the size of the run and not on its frequency, in [2] a new method based on frequency-directed run-length codes is proposed, and by means of experimental results, it is shown that the compression ratio can be improved. In addition to the above approaches which require an on-chip decoder, other methods were proposed which assume the existence of an embedded processor [5, 12]. The method in [12] is based on storing the differing bits between two test vectors, while the method in [5] uses regular geometric shapes formed only from 0s or 1s to compress the test data. Regardless of its benefits, in systems where embedded processors are not available or where embedded processors have access only to a small number of processing elements, this embedded processor method is not applicable. Therefore, only the approaches that are equally applicable to every SoC (i.e. they do not assume the existence of an embedded processor) are relevant to the proposed method.

The aim of this paper is provide a new compression method using an on-chip decoder where the three test parameters (compression ratio, area overhead and test application time) are simultaneously reduced. Unlike previous approaches [2, 3, 10] which reduce some test parameters at the expense of the others, simultaneous reduction is achieved by accounting for multiple factors that influence test parameters. The rest of the paper is organized as follows. Section 2 gives the preliminaries and the motivation for the proposed approach. A detailed analysis of the factors that influence test parameters is given and previous approaches and their characteristics are outlined. Section 3 describes the new compression method, the novel decompression architecture and analyzes test application time. Sections 4 and 5 provide the experimental results and conclusions respectively.

2. Preliminaries and motivation

Testing in test data compression environments (TDCE) implies sending the compressed test data from the ATE to the on-chip decoder, decompressing the test data on chip and sending the decompressed test data to the circuit under test (CUT). There are two major components in TDCE: the **compression method**, used to compress the test set off-chip, and the associated **decompression method**, based on a **on-chip decoder**, used to restore the initial test set on-chip. The on-chip decoder is composed out of two units: a unit to identify a compressed code and a unit to decompress it. If the two units can work independently (i.e. decompressing the current code and identifying a new code can be done simultaneously), then the decoder is called **parallel**. Otherwise the decoder is referred to as **serial**. Testing in TDCE can be characterized by the following three parameters: **(a) compression ratio** which identifies the performance of the compression method, the memory and channel capacity requirements on the ATE; **(b) area overhead** imposed by the on-chip decoder (dedicated hardware or on-chip processor); **(c) test application time** given by the time needed to transport and decode the compressed test set.

There are a number of factors which influence the above parameters: the *mapping and reordering algorithm*, which prepares the test set for compression by mapping the "don't cares" in the test set to '0' or '1' and reordering the test set, the *compression algorithm*, the *type of input patterns* (fixed or variable length), the *length of the pattern*, and the *type of the on-chip decoder* (serial or parallel). Each of the three test parameters is influenced by these factors as follows:

(a) Compression ratio: Compression algorithms that use various types of patterns and different lengths exploit different features of the test set. Using mapping and reordering of the initial test set, those features can be emphasized. Therefore, the compression ratio is influenced firstly by the the *mapping and reordering algorithm*, and then by the *type of input patterns* and the *length of the pattern*, and finally by the *compression algorithm*.

(b) Area overhead: Area overhead is influenced firstly by the *nature of the decoder*, and then by the *type of the input pattern* and the *length of the pattern*. If the decoder has a serial nature then synchronization between the two units (code identification and decompression) is already at hand. However, if the decoder has a parallel nature, the units have to synchronize one with each other, and hence increasing the area overhead. The *type of the input pattern* requires different type of logic to generate the pattern on-chip. For

example, the use of fixed-length input patterns requires a shift register, while the use of variable-length input pattern (runs of 0s for example) requires a counter. On the other hand, the *length of the pattern* impacts the size of the decoding logic, and hence it influences area overhead.

(c) Test application time (TAT): TAT is firstly influenced by the **compression ratio**, and then by the *type of the on-chip decoder*, the *length of the pattern*. To illustrate the factors that influence TAT, consider the ATE operating frequency as the reference frequency. Minimum TAT (min_{TAT}) is given by the size of the compressed test set in ATE clock cycles. However, this TAT can be obtained only when the on-chip decoder can process the currently compressed bit before the next one is sent by the ATE. In order to do so, the relation between the frequency at which the on-chip decoder works and the ATE operating frequency have to meet certain conditions. The *frequency ratio* is the ratio between the on-chip test frequency and the ATE operating frequency. Consider that the min_{TAT} is obtained when the frequency ratio is greater than an *optimum frequency ratio*. Since the, min_{TAT} is given by the size of the compressed test set, increasing the compression ratio would imply further reduction in TAT. However, this reduction happens *only* if the optimum frequency condition is met. Since real environments can not always satisfy the optimum frequency ratio condition, then a natural question is what happens if this condition is not met ? TAT in this cases is dependent on the *type of on-chip decoder*. If the on-chip decoder has a serial nature [2, 3], then TAT is heavily influenced by changes in the frequency ratio. However, if the decoder has a parallel nature [10], the influences are rather minor. Since the *length of the pattern* determines the optimum frequency ratio, consequently it also influences TAT.

In the following it is shown that previous approaches [2, 3, 10] for test data compression/decompression have efficiently solved some of the test parameters at the expense of the others.

(i) Selective coding (SC) [10]: This method splits the test vectors into patterns of size b (b is the size of the fixed input pattern used for coding called also the block size) and applies Huffman coding to a carefully selected number of patterns while the rest of the patterns are prefixed. The SC decoder has a parallel nature. Although, because of its parallelism, the on-chip decoder yields good TAT. However, the choice of a *fixed-length* input pattern and the use of prefixed codes requires shift registers of length b which cause great area overhead. Also, making use of *fixed-length* input patterns, the method is restrained to exploiting the test set features for compression. Therefore *due to a fixed-length input pattern* the main drawbacks of this approach are low compression ratios and large area overhead.

(ii) Golomb coding [3]: This method assigns a variable-length Golomb code, of group size m_g, to a run of 0s. Golomb codes are composed from two parts, a prefix and a tail, and the tail has the length given by $\log m_g$. Using runs of 0s as input patterns, the method has two advantages: firstly the decoder uses counters of length $\log m_g$ instead of shift registers, thus leading to small area overhead; secondly it better exploits the features of the test set for greater compression. Since the method in [3] uses a serial on-chip decoder, TAT is heavily influenced by changes in the frequency ratio. Also as shown later in this paper, Golomb is a particular case of the proposed Variable-length Input Huffman Coding, and hence it constantly leads to inferior compression.

(iii) Frequency-Directed Run-Length (FDR) coding [2]: The FDR code is composed from two parts, a prefix and a tail, where both parts have the same length. In order to decompress a FDR code, the on-chip decoder has to identify the two parts. Because the code is not dependent on a group size as Golomb codes, the decoder has to detect the length of the prefix in order to decode the tail. In order to do so, the FDR code requires a more complicated decoder with fixed area overhead. Therefore, regardless of the good compression ratios the area overhead of FDR is a disadvantage.

As illustrated above, current approaches for test data compression, efficiently address some of the test parameters at the expense of the others. Therefore, the motivation of this paper is to seek to exploit the factors that influence the test parameters: *mapping and reordering algorithm*, *compression algorithm type of input patterns*, *length of the pattern* and the *type of on-chip decoder*. Consequently a new compression method, that leads to simultaneous improvement in compression ratio, area overhead and test application time, is proposed in the following section.

3. New Test Data Compression/Decompression Method based on *Variable-length* Input Huffman Compression (VIHC)

This section introduces a new compression method based on Huffman coding. It is important to note that the previous approach [10] which has used Huffman coding in test data compression have employed only patterns of *fixed-length* as input to the Huffman coding algorithm. As outlined in the previous section *fixed-length* input patterns restrict exploitation of the test set features for compression. This problem is overcome by the approach proposed in this section which uses patterns of *variable-length* as input to the Huffman algorithm and can efficiently exploit a reordered test sequence (Section 3.1). This *fundamental distinction* does not influence only the compression, but also provides the justification for employing a *parallel decoder* using counters (Section 3.2.1) that will lead not only to significantly lower area overhead, but will also facilitate TAT reduction when the compression ratio and frequency ratio are increased (Section 3.2.2 and Section 4).

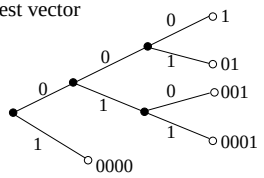
$m_h = 4$ $|t_{init}| = 26$ bits $|t_{cmp}| = 16$ bits

t_{init}	1	01	0000	0000	0000	0001	0000	001
t_{cmp}	000	001	1	1	1	011	1	010

(a) Initial test vector

Pattern	Occurence	Code
L_0 = 1	1	000
L_1 = 01	1	001
L_2 = 001	1	010
L_3 = 0001	1	011
L_4 = 0000	4	1

(b) Dictionary



(c) Huffman tree

Run of 0s	Golomb code
1	000
01	001
0000 0000 0000 0001	1 1 1 011
0000 001	1 010

(d) Golomb codes

Figure 1. VIHC, the particular case

To illustrate the use of the proposed method, consider the example given in Fig. 1. First, the maximum acceptable length of the runs of 0s is selected ($m_h = 4$) and referred to as the group size. Using the group size, the initial test vector (t_{init}) is divided into runs of 0s of length smaller or equal to 4, which are referred to as patterns (Fig. 1(a)). From these patterns and the number of occurrences a dictionary is build as shown in Fig. 1(b). Using the dictionary the Huffman tree is built (Fig. 1(c)). To build the Huffman tree, the list of patterns are ordered in the descending order of their occurrences. The first two patterns, the ones with the smallest number of occurrences, are merged into a new pattern with the occurrence given by the sum of the two patterns' occurrences. Using the remaining patterns and the merged pattern, a new list is created, and the process is repeated until the list comprises only one merged pattern. Each merged pattern in the list represents a node in the Huffman tree, while the composing patterns are its descendents. The Huffman code[1] is constructed by assigning binary 0s and 1s to each segment starting from a node of the Huffman tree (Fig. 1(c)). Using the obtained Huffman code, the initial test vector (t_{init}) is compressed (t_{cmp}). The above can be formalized as follows.

Let $L = \{L_0, \ldots, L_{m_h}\}$ be a set of unique patterns obtained after the test set was divided into runs of 0s of maximum length m_h and $P = \{n_0, \ldots, n_{m_h}\}$ the number of occurrence of each pattern. $n = \sum_{i=0}^{m_h} |L_i| * n_i$ is the size of the test set, where $|L_i|$ denotes the length of the pattern L_i. With L and P the Huffman codes are obtained. For a pattern L_i the Huffman code is denoted by c_i, and the length of the code is given by w_i. It should be noted that the terms "Huffman code" and "codeword" will be used interchangeably and the term "group size" is preferred to "block size". It is interesting to note that there are $m_h + 1$ patterns and thus $m_h + 1$ leafs in the Huffman tree. There are m_h patterns formed from runs of 0s ending in 1 ($L_0 \ldots L_{m_h-1}$) and one pattern formed from runs of 0s only (L_{m_h}).

[1]For detailed description of the Huffman algorithm, the reader is referred to [4]

$m_h = 4$ $|t_{init}| = 26$ bits $|t^V_{cmp}| = 17$ bits $|t^G_{cmp}| = 19$ bits

t_{init}	1	01	0000	0000	0000	0001	0000	1	01
t^V_{cmp}	00	011	1	1	1	010	1	00	011
t^G_{cmp}	000	001	1	1	1	011	1	000	001

(a) Test vectors

Run of 0s	Golomb code
1	000
01	001
0000 0000 0000 0001	1 1 1 011
0000 1	1 000

(b) Golomb codes

Pattern	Occ.	Code
L_0 = 1	2	00
L_1 = 01	2	011
L_3 = 0001	1	010
L_4 = 0000	4	1

(c) VIHC codes

Figure 2. VIHC, the general case

Before presenting the procedures used to compress the test sets, the following observation has to be made:

Observation 1 *For a given group size m, Golomb coding is a particular case of the proposed coding method (VIHC).*

The above can be seen in Fig. 1 where, if the Golomb codes for a group size of 4 is used (the codes where computed for the required runs of 0s in Fig. 1(d)), the t_{init} would be compressed to the same t_{cmp} as in the case of VIHC. This is true only for a particular pattern distribution, i.e. when $(a)\ n_m \geq n_{i_0} + n_{i_1} + \ldots + n_{i_{\frac{m}{2}-1}}, \forall\ distinct\ i_0 \ldots i_{\frac{m}{2}-1} < m$ and $(b)\ n_k < n_i + n_j, \forall i, j, k < m, i \neq j \neq k$, where m represents the group size and n_i is the number of occurrences of pattern L_i, as defined above.

In general, however, the codes differ. This is because the particular pattern distribution, when Golomb leads to same compression as VIHC, is generally not respected. This is illustrated in Fig. 2, where by a simple change in the t_{init} from Fig. 1(a), the bit 24 is set from 0 to 1, the size of the compressed vector (t^G_{cmp}) obtained by Golomb is greater than the size of the compressed vector (t^V_{cmp}) obtained by VIHC. The following observation indicates the relation between the compression obtained by Golomb and VIHC.

Observation 2 *For a given group size m, the compression obtained by the Golomb coding is smaller or equal than the compression obtained by VIHC.*

The above can be easily shown by using the fact that the Huffman code is an optimal code ([4, Theorem 5.8.1]), i.e. any other code will have the expected length greater than the expected length of the Huffman code. Since, the Golomb code is a particular case of VIHC (the VIHC in this particular case is referred to as $VIHC_G$), thus it is optimal only for a particular pattern distribution. For other pattern distributions the $VIHC_G$ code is not optimal, thus the expected length of the $VIHC_G$ code is greater. This concludes the observation.

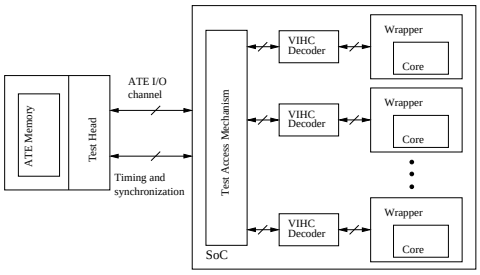


Figure 3. Generic decompression architecture [3]

It is important to note that Observation 1 assures that the VIHC decoder proposed in this paper (Section 3.2.1) can also be used to decompress a Golomb compressed test set.

3.1. Compression Algorithm

This section presents the procedures used to implement the proposed method (VIHC). In this paper, without loss of generality, only the compression of the difference vector sequence (T_{diff}) is taken into account. The initial test set (T_D) is partially specified and the test vectors can be reordered. This is because ATPG tools generate a small number of care bits in every test vector [13]. This gives great flexibility to the *mapping and reordering algorithm*, which in a pre-processing step prepares the test set for compression by mapping the "don't cares" in the test set to '0' or '1' and reordering the test set such that VIHC can increase its compression. Thus, the compression algorithm has three main procedures: (1) prepare initial test set, (2) compute Huffman code and (3) generate decoder information. The first two procedures are used for compression only, while the last one determines the decoding architecture described in Section 3.2.

1) **Prepare initial test set** The procedure consists of two steps. In the first step, the "don't cares" are mapped to the value of the previous vector on the same position. In the second step the test set is reordered such that the number of 1s in the difference between two test vectors is minimum and the length of the minimum run of 0s is maximum. This will be exploited by the *variable-length* input patterns used for Huffman code computation. The reordering algorithm has a complexity of $O(|T_D|^2)$, where $|T_D|$ represents the number of test vectors in the test set.

2) **Huffman code computation** Based on the chosen group size (m_h) the dictionary of *variable-length* input patterns L and the number of occurrences P are determined from the reordered test set. This adds an additional computational step when compared to Golomb coding which assigns a precomputed code for each run length. Using L and P the code is computed by the Huffman algorithm [4].

3) **Generate decoder information** This procedure computes the necessary information to build the decoder. For each Huffman code c_i a binary code b_i is assigned. The binary code is composed from the *length* of the initial pattern of $\lceil log_2(m_h+1) \rceil$ bits and a *special* bit which identifies

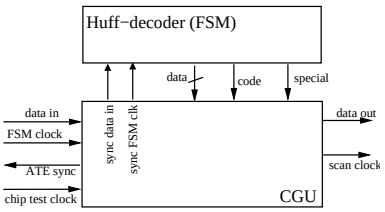


Figure 4. VIHC decoder

the cases when the initial pattern is composed of runs of 0s only or runs of 0s ending in 1, thus $b_i = (|L_i|, 0)$ for $i < m_h$, and $b_{m_h} = (|L_{m_h}|, 1)$. The *Control and Generation Unit* (see Section 3.2.1) uses the binary code to generate the pattern.

3.2. Decompression architecture

This section introduces the new decoding schemes (Section 3.2.1) for the VIHC compression method, assuming a generic decompression architecture (Fig. 3) and provides the TAT analysis (Section 3.2.2) of the VIHC decoder. The VIHC decoder, proposed in Section 3.2.1, uses a novel parallel approach, in contrast to the previous Golomb [3] serial decoder. It should be noted that for the decompression of T_{diff}, a CSR architecture [3, 11] is used after the VIHC decoder in Fig. 3. This work assumes that the ATE is capable of external clock synchronization as shown in [7].

3.2.1 VIHC decoder

A block diagram of the VIHC decoder is given in Fig. 4. The decoder is formed from a *Huffman decoder* [9] (Huff-decoder) and a *Control and Generator Unit* (CGU). The Huff-decoder is a finite state machine (FSM) which detects a Huffman code and outputs the corresponding binary code. The CGU is responsible for controlling the data transfer between the ATE and the Huff-decoder, generate the initial pattern and the scan clock for the CUT. The *data in* line is the input from the ATE synchronous with the external clock *ATE clock*. When the Huff-decoder detects a codeword, the *code* line is high and the binary code is output on the *data* lines. The *special* input to the CGU is used to differentiate between the two types of patterns, runs of 0s only and runs of 0s ending in 1. After loading the code, the CGU generates the pattern and the internal *scan clock* for the CUT. If the decoding unit generates a new code while the CGU is busy processing the current one, the *ATE sync* line is low notifying the ATE to stop sending data and the *sync FSM clk* is disabled forcing the Huff-decoder to maintain its current state. It should be noted that if the ATE responds to the stop signal with a given latency (i.e. it requires a number of clock cycles before the data stream is stopped), the device interface board (DIB) between the ATE and the system will have to account for the latency with a first in first out (FIFO) - like structure. Dividing the VIHC decoder in Huff-decoder and CGU, allows the Huff-decoder to continue loading the next codeword while the CGU generates the current pattern. Thus, the Huff-decoder is interrupted *only if necessary*, which is in contrast to the Golomb [3]

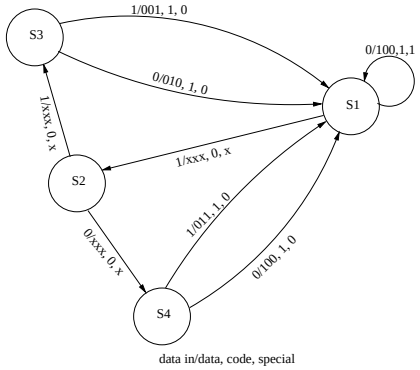


Figure 5. FSM for example in Figure 1

and the FDR [2] serial decoders. This leads to reduction in TAT when compared to the Golomb and the FDR, as it will be shown in Section 4.

The FSM for the Huffman decoder from the example in Fig. 1 is illustrated in Fig. 5. Starting from state S_1, depending on the value of *data_in* the Huff-decoder change its state. It is important to note the following:

- after the Huff-decoder detects a codeword it goes back to state S1; for example, if the *data in* stream is 001 (last bit first) the Huff-decoder first changes its state to S2 then to S4 after which to S1, and setting *data* to the corresponding binary code and the *vector* line high;
- the number of ATE clock cycles needed to detect a code is equal to the length of the code; for example, if the *data in* stream is 001 (last bit first), the Huff-decoder identifies the code in three ATE clock cycles;
- the Huff-decoder has a maximum of m_h states for a group size of m_h; the number of states in a Huff-decoder is given by the number of leafs in the Huffman tree minus one; since there are a maximum of m_h+1 leafs for a group size of m_h, the number of states is m_h;

When compared to the SC decoder [10], which has an internal buffer and a shift register of size b and a Huffman FSM of at least b states, the proposed decoder has only a $\lceil log_2(m_h+1)\rceil$ counter, two extra latches and a Huffman FSM of at most m_h states. Thus, for the same group size ($m_h = b$) significant reduction in area overhead is obtained, as demonstrated in Section 4.

A detailed view of the CGU is given in Fig. 6. The CGU is formed from a $\lceil log_2(m_h+1)\rceil$ bits counter and additional logic. When the *code* line is high, the values at the *data* inputs are loaded into the counter. With the data loaded, the counter decrements to 1 setting the *data out* to 0. When the value of the counter is 1, *data out* is set to 0 if the *special* line is high, or to 1 if the *special* line is low. The *FSM clock* signal assures that the FSM will reach a stable state after one internal clock cycle, therefore it has to be generated as a one internal clock cycle for each external clock cycle period. This can be easily achieved by using the same technique as in [7] for the *serial scan enable* signal.

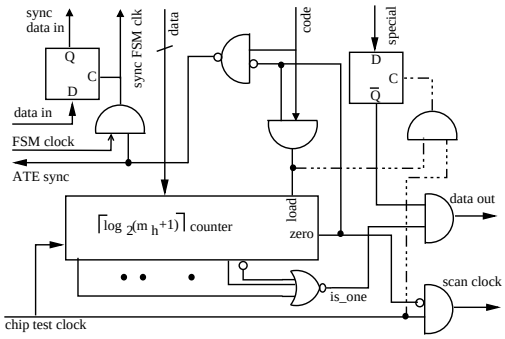


Figure 6. CGU for VIHC decoder

3.2.2 Test application time analysis

In this section a TAT analysis, for the proposed method, with respect to the ratio between the on chip test frequency and the ATE operating frequency is given. It is considered that the data is fed into the FSM with the ATE operating frequency and the FSM reaches a stable state after one on-chip test clock cycle. Analyzing the FSM for the Huffman decoder it can be observed that the number of ATE clocks needed to identify a codeword c_i is equal to the size of the codeword w_i (see Section 3). On the other side, the number of internal clock cycles needed to generate a pattern is equal to the size of the pattern.

Consider the ATE clock cycle of 50 *ns*, the on-chip test clock cycle of 10 *ns*, the next codeword of length 1, and the current pattern size of length 4. The Huff-decoder will identify the next codeword in 1 ATE clock cycle (50 *ns*) however, the CGU will generate the current pattern in 40 *ns*. Since the Huff-decoder reaches a stable state after one internal clock cycle, from the current ATE clock cycle only 10 *ns* where used, the rest of 40 *ns* can be used to generate the pattern. Thus, the CGU has enough time to generate the current pattern without having to set *ATE sync* low and stop the ATE from sending data.

Formally, if $\alpha = \frac{f_{chip}}{f_{ate}}$ is the ratio between the on-chip test frequency (f_{chip}) and the ATE operating frequency (f_{ate}), then after a Huffman code is identified, $\alpha - 1$ internal clock cycles from the current ATE cycle can be used to generate the pattern. Thus, in order for the Huff-decoder to run without being stopped by the CGU, the CGU has to be able to generate the pattern L_i in the number of internal clock cycles remained from the ATE clock in which it was started plus the number of internal clock cycles needed for the Huff-decoder to identify the next codeword. Or, $|L_i| < (\alpha-1)+1+\alpha*(w_j-1)$, where w_j is the length of the next codeword in bits. The smallest TAT is given by $n_h+\delta$, where n_h is the number of bits in the compressed test set in ATE clock cycles and δ is the number of extra ATE clocks needed to decode the last codeword. With $max(|L_i|) = m_h$ and $min\{w_i\} = w_{min}$, the frequency ratio to obtain the smallest TAT is given by $\alpha_{max} = \frac{f_{chip}}{f_{ate}} = \frac{m_h}{w_{min}}$ (*optimum frequency ratio*). When the frequency ratio is greater

Circuit	Group size	SC [10]	Golomb [3]	FDR [2]	VIHC
s5378	4	34.79	40.70	48.19	**51.52**
s9234	4	35.52	43.34	44.88	**54.84**
s13207	16	77.73	74.78	78.67	**83.21**
s15850	4	40.16	47.11	52.87	**60.68**
s35932	16	65.72	N/A	10.19	**66.47**
s38417	4	37.11	44.12	**54.53**	54.51
s38584	4	37.72	47.71	52.85	**56.97**

Table 1. Compression comparison for T_{diff}

than the *optimum frequency ratio*, an increase in the compression ratio will lead to reduction in test application time. Thus, for better compression of VIHC when compared SC [10], an increase in frequency ratio will also lead to lower test application time as shown in Section 4.

4. Experimental Results

To validate the efficiency of the proposed method, experiments were performed on the full-scan version of the largest ISCAS 89 benchmark circuits. Test sets used in this experiments were obtained using MinTest [6] dynamic compaction (also used in [3]). The experiments were performed on a Pentium III 500MHz workstation with 128 MB DRAM. Golomb [3], SC [10], FDR [2] and the proposed VIHC method were implemented in C++. To provide an uniform basis for the comparison between Golomb, SC and VIHC, as the three methods use a group size, we use the group size for which [3] reports the best results. The VIHC and SC compression ratios where computed for those group sizes. The results are provided in Table 1. It should be noted that [10] did not report compression ratios for the test sets used in this paper [6], however the results were computed by implementing SC method and applying it to MinTest.

Table 1 provides a compression comparison for the T_{diff} test sets. The compression ratios are relative to the size of the dynamic compacted test set from MinTest [6]. With the exception of one particular case when FDR has similar compression ratio as VIHC (s38417), the table clearly shows that the proposed method leads to better compression ratios than previous approaches. When compared to Golomb, the proposed method obtains up up to 66% better compression ratios (s35932). When compared to FDR, the method obtains 56% reduction for s35932 and up to 8% reduction for the other circuits. The reason for a much better compression ratio in the case of circuit s35932 is that the test set for this circuit has a large number of bits '1', which is a problem for both Golomb and FDR methods. When compared to SC, the proposed method obtains up to 20% in compression ratio (s15850). It should be noted that a major contribution to this considerable savings is not only coding algorithm but also the pre-processing step using the *mapping and reordering algorithm* described in Section 3.1.

Table 2 illustrates the area overhead computed with the Synopsys Design Compiler for the four on-chip decoders.

Compression Method	Area overhead in tu*		
	Group size		
	4	8	16
SC [10]	349	587	900
Golomb [3]	**125**	227	307
FDR [2]	320		
VIHC	136	**201**	**296**

* technology units for the lsi10k library (Synopsys Design Compiler)

Table 2. Area overhead comparison

For all methods, the entire decoder including buffers, shift registers and counters was synthesized. For SC, VIHC and Golomb, the area overhead was computed for a group size of 4, 8 and 16. Without loss of generality the decoders for s5378 were synthesized. As shown in Table 2, the area overhead for SC is **up to 3 times greater** than the area overhead for VIHC. The area overhead for Golomb is almost equal to the one of VIHC, and the area overhead for FDR is constantly greater than the area overhead for VIHC.

For TAT comparison, a simulator was implemented based on the TAT analysis for SC [10], Golomb [3], FDR [2] and the VIHC decoder (Section 3.2.1). For all decoders it was assumed that the data is fed into the decoder at ATE operating frequency and the internal FSM reaches a stable state after one internal clock cycle. In order to provide an accurate comparison, we used the same group size as in Table 1, and compressed and simulated all the test sets. The results are reported in Table 3. The table shows, the circuit, the compression method and the TAT obtained for four frequency ratios: $\alpha = 2,4,6$ and 8. Analyzing columns 3 to 6 from Table 3 it can be seen that for smaller frequency ratios SC has slightly better TAT than the proposed method (e.g. s35932 or s38417 and s38584 and $\alpha = 2$). However, it can be observed that when frequency ratio increases the proposed method leads up to 34% (s15850 and $\alpha = 6,8$) savings in TAT when compared to the SC method [10]. Comparing the proposed method with Golomb and FDR, TAT reduction up to 55% (when compared to Golomb) and up to 45% (when compared to FDR) is obtained for s35932. The reason for significantly better TAT for s35932 is the very high compression ratio obtained with the proposed method (see row s35932 in Table 1). For the rest of the circuits, the TAT ranges from similar values (when frequency ratio increases) to reduction of up to 35% when compared to Golomb (s15850), and reduction of up to 31% when compared to FDR (s9234).

Finally a comparison of the previous approaches [2, 3, 10] and the proposed VIHC is given in Table 4. While FDR [2] gives compression ratios comparable with VIHC, it leads to both lower test application time and greater area overhead. On the other hand, Golomb [3] has similar area overhead when compared to VIHC at the expense of lower compression ratio and greater TAT for small frequency ratios. For high frequency ratios, Golomb's TAT approaches

Circuit	Comp. Method	TAT (ATE clock cycles)			
		$\alpha = 2$	$\alpha = 4$	$\alpha = 6$	$\alpha = 8$
s5378	SC [10]	**15491**	15491	15491	15491
	Golomb [3]	21432	16662	11892	11892
	FDR [2]	22263	14678	12968	12968
	VIHC	15763	**11516**	**11516**	**11516**
s9234	SC [10]	25324	25324	25324	25324
	Golomb [3]	33651	25693	17735	17735
	FDR [2]	36135	24128	21381	21381
	VIHC	**24743**	**17735**	**17735**	**17735**
s13207	SC [10]	**89368**	53490	45137	36784
	Golomb [3]	104440	66381	47783	47481
	FDR [2]	107059	63011	49858	41989
	VIHC	89865	**52769**	**44180**	**36065**
s15850	SC [10]	46067	46067	46067	46067
	Golomb [3]	63989	47164	30339	30339
	FDR [2]	62419	39628	33767	33767
	VIHC	**45765**	**30264**	**30264**	**30264**
s35932	SC [10]	**16870**	**11749**	**10710**	9671
	Golomb [3]	37869	32886	30509	30509
	FDR [2]	32509	20605	18438	17045
	VIHC	17584	12076	10857	**9645**
s38417	SC [10]	**103604**	103604	103604	103604
	Golomb [3]	143844	109801	75758	75758
	FDR [2]	144811	93450	81578	81578
	VIHC	106411	**74943**	**74943**	**74943**
s38584	SC [10]	124011	124011	124011	124011
	Golomb [3]	169356	127512	85668	85668
	FDR [2]	170143	110982	96677	96677
	VIHC	**123693**	**85668**	**85668**	**85668**

Table 3. TAT comparison for T_{diff} compressed test sets

VIHC's TAT, which is unlike SC [10] where TAT is comparable to VIHC only for small frequency ratios. However, this is achieved at a very high penalty in area overhead which is the main shortcoming of the parallel decoder based on *fixed-length* Huffman coding.

5. Conclusions

This paper has presented a new compression method called Variable-length Input Huffman Coding (VIHC). Unlike previous approaches [2, 3, 10] which reduce some test parameters at the expense of the others, the proposed compression method is capable of minimizing test parameters simultaneously. This is achieved by accounting for multiple interrelated factors that influence the results, such as pre-processing the test set, the size and the type of the input patterns to the coding algorithm, and the type of the the decoder. The results in Section 4 show that the proposed method obtains constantly better compression ratios than [2, 3, 10]. Furthermore, the parallel decoder leads to savings in TAT when compared serial decoders [2, 3]. Moreover, by the exploiting the variable-length input approach, great savings in area overhead are achieved (up to three-fold reduction when compared to fixed-length approach [10]). Therefore, it was shown that the proposed method

	SC[10]	Golomb[3]	FDR[2]	VIHC
Compression	*X*	*X*	$\surd$	$\surd$
Area overhead	*X*	$\surd$	*X*	$\surd$
Test application time	$\surd$	$\surd$	*X*	$\surd$

Table 4. Previous approaches compared to VIHC

decreases the ATE memory and channel capacity requirements by obtaining good **compression ratios**, and reduces **TAT** through its parallel on-chip decoder with small **area overhead**. Thus, it is an effective solution for test data compression/decompression for SoCs.

Acknowledgments The authors wish to thank Anshuman Chandra and Dr. Krishnendu Chakrabarty from Duke University for providing the test sets used in their papers.

References

[1] The International Technology Roadmap for Semiconductors, 1999 Edition, ITRS.

[2] A. Chandra and K. Chakrabarty. Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression. In *Proceedings IEEE VLSI Test Symposium*, 114–121, Apr. 2001.

[3] A. Chandra and K. Chakrabarty. System-on-a-Chip Test Data Compression and Decompression Architectures Based on Golomb Codes. *IEEE Transactions on Computer-Aided Design*, 20:113–120, Mar. 2001.

[4] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. 1991.

[5] A. El-Maleh, S. al Zahir, and E. Khan. A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-Chip. In *Proceedings IEEE VLSI Test Symposium*, 114–121, Apr. 2001.

[6] I. Hamzaoglu and J. H. Patel. Test set compaction algorithms for combinational circuits. In *Proceedings International Conference on Computer-Aided Design*, 283–289, Nov. 1998.

[7] D. Heidel, S. Dhong, P. Hofstee, M. Immediato, K. Nowka, J. Silberman, and K. Stawiasz. High-speed serializing/deserializing design-for-test methods for evaluating a 1 ghz microprocessor. In *Proceedings IEEE VLSI Test Symposium*, 234–238, Apr. 1998.

[8] M. Ishida, D. S. Ha, and T. Yamaguchi. Compact: A hybrid method for compressing tets data. In *Proceedings IEEE VLSI Test Symposium*, 62–69, Apr. 1998.

[9] V. Iyengar, K. Chakrabarty, and B. Murray. Deterministic built-in pattern generation for sequential circuits. *Journal of Electronic Testing: Theory and Applications*, 15:97–114, August/October 1999.

[10] A. Jas, J. Ghosh-Dastidar, and N. A. Touba. Scan Vector Compression/Decompression Using Statistical Coding. In *Proceedings IEEE VLSI Test Symposium*, 114–121, Apr. 1999.

[11] A. Jas and N. Touba. Test Vector Decompression Via Cyclical Scan Chains and Its Application to Testing Core-Based Designs. In *Proceedings IEEE International Test Conference*, 458–464, Oct. 1998.

[12] A. Jas and N. Touba. Using an Embedded Processor for Efficient Deterministic Testing of Systems-on-a-Chip. In *Proceedings International Conference on Computer Design*, 418–423, Oct. 1999.

[13] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater. A SmartBIST Variat with Guaranteed Encoding. In *Proceedings of the Asian Test Symposium*, 325–330, Nov. 2001.

[14] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel. Towards a Standard for Embedded Core Test: An Example. In *Proceedings IEEE International Test Conference*, 616–627, Sept. 1999.

[15] I. Pomeranz, L. Reddy, and S. Reddy. Compactest: A method to generate compact test set for combinational circuits. *IEEE Transactions on Computer-Aided Design*, 12:1040–1049, July 1993.

[16] Y. Zorian, S. Dey, and M. J. Rodgers. Test of Future System-on-Chips. In *Proceedings International Conference on Computer-Aided Design*, 392–400, Nov. 2000.