A Method for Compressing Test Data Based on Burrows-Wheeler Transformation

Takahiro J. Yamaguchi, *Member*, *IEEE*, Dong Sam Ha, *Senior Member*, *IEEE*, Masahiro Ishida, and Tadahiro Ohmi, *Senior Member*, *IEEE*

Abstract—The overall throughput of automatic test equipment (ATE) is affected by the download time of test data. An effective approach to the reduction of the download time is to compress test data before the download. A compression algorithm for test data should meet two requirements: lossless and simple decompression. In this paper, we propose a new test data compression method that aims to fully utilize the unique characteristics of test data compression. The key idea of the proposed method is to perform the Burrows-Wheeler transformation on the sequence of test patterns and then to apply run-length coding. Experimental results show that our compression method performs better than six other methods for compressing test data. The average compression ratio of the proposed method performed on 15 test data sets is 94.6, while that for the next best one, Gzip, is 65.0. The experimental results also show that our method indeed reduces the download time significantly, provided a dedicated hardware decompressor is employed.

Index Terms—Data compression, test data compression, compression, test data, Burrows-Wheeler transformation, run-length coding.

1 Introduction

Test patterns are usually generated and stored on workstations or high-performance personal computers. The increased variety of ASICs and decreased production volume of individual types of ASICs requires more frequent downloads of test data sets from workstations to automatic test equipment (ATE). In addition, because of the sheer size of test sets for ASICs, often as large as several gigabytes, the time spent to download test data from computers to ATEs is significant. The download from a workstation storing a test set to the user interface workstation attached to an ATE is often accomplished through a network. The download takes from several tens of minutes to hours. The test set is then transferred from the user interface workstation of an ATE to the main pattern memory through a dedicated high speed bus. The latter transfer usually takes several minutes. The transfer of test data from a workstation to an ATE is shown in Fig. 1.

During the download period of a test set, the ATE is idle, wasting this valuable resource. The overall throughput of an ATE is affected by the download time of test data and the throughput becomes more sensitive to the download time with the increased variety of ASICs. One common approach to improve the throughput of an ATE is to download the test data of the next chip during the testing of a chip. It cuts down the effective download time of test data, but the approach alone may not be sufficient. An ATE may finish testing of the current chip before the download of the next test data is completed. Another approach is to compress the test data, which is the focus of the paper.[1] The two approaches can be readily combined together to further improve the efficiency.

A compression algorithm for test data should meet two requirements: It should be lossless and have simple decompression. As the decompression is performed on the ATE side, the decompression time should be minimized to reduce the overall download time. However, compression time does not affect the download time, as it can be prepared in advance on a computer. Therefore, a suitable method for the ATE environment is possibly complex in compression, but simple in decompression. The unique characteristics of test data compression need be exploited to maximize efficiency when designing a test data compression scheme. Another characteristic that can be exploited for test data compression is that subsequent test patterns on the same pin are strongly correlated, but the patterns on different pins are weakly correlated.

A simple compression scheme that can be applied to test data compression is run-length coding, where a sequence of equal symbols is encoded into two elements, the repeating symbol and the length of the sequence. For data with many long sequences of equal symbols, run-length coding is apparently efficient. Huffman coding is more sophisticated than run-length coding and yields better compression [1]. Huffman coding builds a binary tree based on the probability of the occurrence of the letters, where leaves in the binary tree correspond to the letters. The Huffman code for a letter is obtained by traversing the tree from its root node to the leaf corresponding to the letter,

- *T.J. Yamaguchi and M. Ishida are with Advantest Laboratories Ltd., 48-2 Matsubara, Kamiayashi, Aoba-ku, Sendai, Miyagi 989-3124, Japan. E-mail: {jamax, ishida}@atl.advantest.co.jp.*
- *D.S. Ha is with the Department o Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, CA 24061-0111. E-mail: ha@vt.edu.*
- *T. Ohmi is with the Department of Electronics, Faculty of Engineering, Tohoku University, Aoba, Aramaki, Aoba-ku, Sendai, Miyagi 980-8579, Japan. E-mail: ohmi@sse.ecei.tohoku.ac.jp.*

Manuscript received 25 May 1999; revised 20 June 2001; accepted 8 Aug. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 109940.

1. An earlier version of the paper appears in [14].

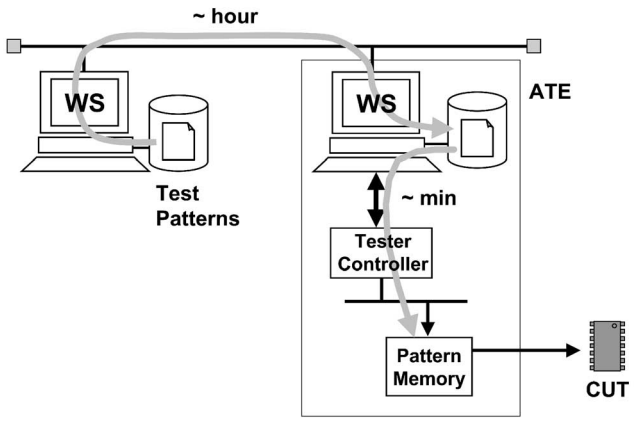

Fig. 1. Download of test patterns to automatic test equipment.

concatenating "0" to the code word every time it traverses over a left branch and "1" over a right branch. A different approach from the above two methods is arithmetic coding [2]. Arithmetic coding generates a unique tag or identifier for a given sequence of symbols, then deciphers tags to restore the original sequence. The main advantage of arithmetic coding over Huffman coding is that building a binary tree structure is unnecessary. The Lempel-Ziv (LZ) method, based on the construction of a dictionary, builds a list of patterns. The LZ method encodes patterns according to their indices in the list [3], [4]. The LZ method does not require a priori knowledge of the probability of the occurrence of letters and becomes more efficient for a longer sequence of patterns whose characteristic is static. The Lempel-Ziv-Welch (LZW) algorithm, which is a derivative of the LZ method, collects new phrases into a dictionary [5]. When a repeating phrase is found, the index of the phrase in the dictionary is recorded to compress the phrase. Some compression utilities available on personal computers and workstations (such as PKZIP [6] and compress) implement variations of the LZ method. The Lempel-Ziv-Storer-Szymanski (LZSS) algorithm keeps track of the last n bytes of data [7]. When a phrase that has appeared before is encountered, the phrase is encoded as a pair of values corresponding to the position of the phrase in the buffer and the length of the phrase. Besides the above general data compression algorithms, there are many compression methods designed for special applications such as speech, image, and video [8], [9], [10]. These methods are usually lossy and, therefore, cannot be applied to test data compression.

The general purpose compression algorithms described above do not exploit the unique characteristics of test data compression. Hence, they usually result in a low compression ratio. A different approach for test data compression was investigated in [11], [12], [13]. The goal of the methods is to compress test data, so that the test generation circuitry is simplified to reduce the hardware overhead in the BIST (Built-In Self-Test) environment. These methods do not yield a high compression ratio (which is not the goal of the methods), so that they are not applicable for reduction of download time.

In this paper, we propose a new compression method which transforms a given test set first and then applies run-length coding. The proposed method, which fully utilizes the unique characteristics of the ATE environment and of test data, achieves a high compression ratio. Another salient point of the proposed method lies in simple decompression. The decompression algorithm can be easily realized in hardware to decompress multiple columns simultaneously. A dedicated hardware operating on multiple columns in parallel substantially reduces the decompression time, which leads to a significant reduction of download time. (Experimental results will be shown in Section 4.) This is a good contrast to our previous method presented in [15], which employs both run-length coding and a UNIX utility Gzip for a transformed data. The method in [15] performs better in compressing test data, but the high complexity of the decompression procedure renders the method not suitable for a dedicated hardware implementation.

The paper is organized as follows: Section 2 gives preliminaries necessary to understand the proposed method. Section 3 proposes a test data compression method. In Section 4, we present experimental results performed on fifteen test sets and compare the results with other compression techniques. Section 5 summarizes the paper.

2 Preliminaries

In this section, we describe the characteristics of test data and define necessary terms. We briefly review run-length coding and Burrows-Wheeler transformation method, which are necessary to understand the proposed method.

We use the term compression ratio, instead of compression rate, throughout the paper. The *compression ratio* is defined as the ratio of the number of bits required to represent an original uncompressed data to that of the compressed data.

2.1 Characteristics of Test Data

For a large complex circuit, a designer or a test generator generates test patterns considering one or a few modules at a time. Therefore, a block of test patterns usually exercises a few modules of the circuit, while other modules are put under certain static conditions. This implies that logic values for only a subset of input pins change for the block(s) of test patterns, while other input pins are held at constant logic values. Fig. 2 shows the testing of a module and a typical block of test patterns. In Fig. 2, bold characters denote active input pins whose logic value changes frequently and their logic values. All the other input pins are held at constant values for the entire block of the test patterns except for the initial setup stage. Another useful observation that we made from industrial test data is that the sequence of test patterns of an active input pin often forms cycles. For example, Pin 4 in Fig. 2 forms a cycle "X11X00" and the cycle repeats two times before it breaks. Pin 6 forms a cycle "X00" and it repeats three times.

A test set is represented as a $P \times Q$ matrix, where P is the number of test patterns, and Q is the number of pins. A column number in the matrix corresponds to the pin number of the circuit. We use the terms column number and pin number interchangeably in this paper. The **activity**

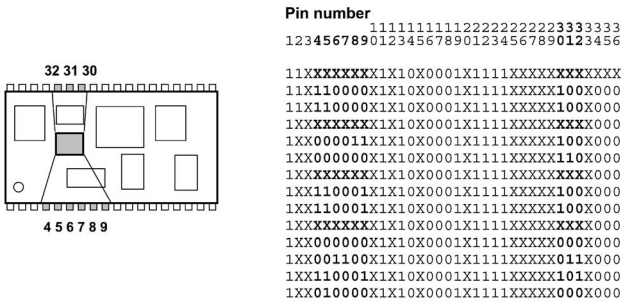

Fig. 2. Testing of a module and its test patterns.

of a column c_i is defined as the number of transitions on the ith column of a test set, and is denoted as $\alpha(ci)$. The activity of an example test set is given in Fig. 3.

2.2 Run-Length Coding

A sequence of identical symbols in a string is called a run. Run-length coding compresses data by representing each run into two tuples, the repeating symbol in the run and the run length. A string "aabbbbcccd" has four runs, aa, bbbb, ccc, and d. It is coded as (a,2), (b,4), (c,3), (d,1).

Run-length coding is simple in compression and in decompression. It is efficient for pins with low activity and is used in the proposed method.

2.3 Burrows-Wheeler (BW) Transformation

Burrows and Wheeler proposed a data compression method based on Wheeler's earlier transformation studied in 1983 [16], [17]. The compression method was further investigated recently by Balkenhol and Kurtz [18]. Consider an input string S which has n characters. The first step of the BW transformation is to form an $n \times n$ matrix, where the nth row is the string obtained by performing $(n-1)$ rotate-left operations on the original sequence. Then, the rows of the matrix are sorted lexicographically. Burrows and Wheeler showed that if the last column T of the sorted matrix and the row index I of the original string in the stored matrix are available, the original string S can be restored through a simple process. The restoring procedure does not require a sorting process. For details of the operation, refer to [16] and [17]. The BW transformation is illustrated for string S with "abraca" in Fig. 4. Fig. 4a shows the matrix formed by rotating the original sequence and Fig. 4b shows the matrix after the lexicographic sorting of the rows.

	1	2	3	4	5	6
	0	0	1	X	X	X
	0	0	X	1	X	1
	0	X	X	1	X	X
	0	X	1	1	X	1
	0	X	1	1	X	X
	0	X	0	X	X	0
$\alpha(c_i)$	**0**	**1**	**3**	**2**	**0**	**5**

Fig. 3. Activity of pins of a test set.

S = **abraca**

1	a	b	r	a	c	a
2	b	r	a	c	a	a
3	r	a	c	a	a	b
4	a	c	a	a	b	r
5	c	a	a	b	r	a
6	a	a	b	r	a	c

Fig. 4a. The BW transformation. (a) Initial matrix.

1	a	a	b	r	a	**c**
2	a	b	r	a	c	**a**
3	a	c	a	a	b	**r**
4	b	r	a	c	a	**a**
5	c	a	a	b	r	**a**
6	r	a	c	a	a	**b**

T = **caraab**
I = **2**

Fig. 4b. The BW transformation. (b) Sorted matrix.

A salient characteristic of the BW transformation is that the resultant string T usually gives a better compression ratio for run-length coding than does the original string S. For example, the number of runs for the original string S in Fig. 4 is 6, but that for the resultant string T is 5. The effect is more evident for sequences with cycles. Consider a sequence S with "000100010001," which has six runs. The resultant string T after the BW transformation is "111000000000," which has only two runs. However, it should be noted that the BW transformation does not always decrease the number of runs for a sequence. The BW transformation of "000111" is "100110," for which the number of runs increases from two to four. It seems difficult to predict the change of the activity of a string before and after the BW transformation.

As noted earlier, the test sequence of an active pin often forms cycles. Hence, the BW transformation of the sequence makes run-length coding effective. Another desirable aspect for the BW transformation in test data compression is that the reverse operation, i.e., decompression, is simple since it does not involve a sorting process. One point to be noted in using the BW transformation for test data compression is that, because it requires lexicographic sorting for the BW transformation, partition of a test set into subblocks is often necessary before applying the BW transformation. In general, smaller subblocks make run-length coding less effective. However, the problem may be alleviated by adopting an efficient lexicographic sorting method such as the one proposed in [19] and [20].

3 Proposed Compression Method

In this section, we present a new method for compressing test data. A primary goal of the proposed method is that the decompression scheme should be simple, possibly at the cost of a complex compression scheme. To this end, we employ the BW transformation and run-length coding.

3.1 Overall compression procedure

Let a given test data set D be represented as a matrix of $P \times Q$. The test data is partitioned into several equal size

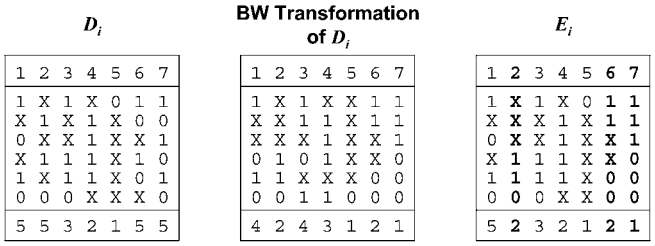

Fig. 5. Construction of a new test data set E_i.

submatrices D_i of $M \times Q$ as shown below. Note that the last submatrix D_k may have a smaller size. The partition of the matrix is not essential, but the number of rows, M, affects the overall compression ratio and the processing time (as our experimental results show). A larger M results in substantially longer processing time due to the sorting process necessary for the BW transformation.

$$D = \begin{bmatrix} D_1 \\ D_2 \\ \cdots \\ D_k \end{bmatrix}.$$

Our compression method is applied to individual submatrices D_i's instead of to the original matrix D.

We apply the BW transformation on individual columns of a test set D_i. As mentioned earlier, it is difficult to predict the activity of a BW transformed column. In addition, high complexity of the compression process is tolerable for test data compression, as long as the decompression process is kept simple. Hence, we propose to apply the BW transformation on each column and to measure the activity of the BW transformed column. (In fact, the BW transformation of a column with activity 0 or 1 is unnecessary, as the transformation does not change the activity.)

A new test data set E_i is composed of original or BW transformed columns of D_i. Let d_k be the kth column of D_i, and let d_k^* be the BW transformed column of d_k. The kth column e_k of E_i is defined as

$$e_k = \begin{cases} d_k^* & \text{if } \alpha(d_k^*) < \alpha(d_k) \text{ and } \alpha(d_k^*) < \alpha_t \\ d_k & \text{otherwise,} \end{cases}$$

where $\alpha(\mathrm{d_k})$ and $\alpha(d_k^*)$ are the activity of d_k and d_k^*, respectively, and α_t is a threshold value. E_i collects a BW transformed column only if the activity of the BW transformed column is less than that of the original one and a threshold value. Therefore, run-length coding always compresses better with new test data E_i than the original test data D_i.

Run-length coding is applied only to each column c_i whose activity $\alpha(c_i)$ is less than the threshold value α_t. For a column whose $\alpha(c_i) \geq \alpha_t$ run-length, coding does not compress data for the column. For such a column, it is better for E_i to keep the original column instead of the transformed one, which saves the time for the reverse BW transformation for the column. Details regarding the threshold value will be discussed in Section 3.3.

The construction of an E_i from a test data set D_i is illustrated in Fig. 5. The last row of each table in the figure indicates the activity of the columns. Let us suppose that the threshold value α_t is given as 3. The activity of four columns, columns 1, 2, 6, and 7, decreases after the BW transformation. However, the activity of the BW transformed column 1 (which is 4) exceeds the threshold value 3. Hence, the original columns are replaced by the BW transformed columns only for columns 2, 6, and 7 (which are in boldface in Fig. 5) in E_i. To restore the original data D_i from E_i during decompression, it is necessary to flag the BW transformed columns in E_i. Note that the overall activity of D_i is 26 and that for E_i is 16. Hence, run-length coding is more effective with E_i.

Throughout the remainder of this section, a test set or a matrix denotes a submatrix E_i. In other words, the flag to indicate the BW transform of a column and the column index I are not considered in the analysis.

TABLE 1
Encoding of Run-Length Coding

Transition: $s \rightarrow t$		Symbol t		
		0	1	X
Symbol s	0	-	L	$L+M$
	1	$L+M$	-	L
	X	L	$L+M$	-

3.2 Encoding and Decoding Schemes for Run-Length Coding

A straightforward run-length coding encodes a sequence of a run in two tuples, (s, L), where s is the repeating symbol and L is the length of the run. For example, run-length coding encodes a sequence "XX1110XX" as (X,2), (1,3), (0,1), (X,2). Note that parentheses and commas are only for readability and they do not appear in the compressed data. The encoding scheme requires one symbol and one integer number for each run.

Next, we describe a more efficient encoding scheme suitable for test data. Consider two consecutive runs, (s, Ls) and (t, Lt), where "s" and "t" are the repeating symbols and Ls and Lt are the run lengths. The two runs make a transition from "s" to "t" at the boundary of the runs. Suppose that the repeating symbol of the first run, "s," is known. The proposed encoding scheme is to compound the run length Ls of the first run and the repeating symbol of the following run, "t," into a single integer.

Let A be the set of all possible symbols for a string and M be the length of the string. For test data, symbols are usually confined to small number (three or four) of logic values. We consider three logic values[2] $(\mathrm{A} = 0, 1, \mathrm{X})$ in this paper, but the proposed method can be readily extended for a larger logic value system. The rule to compound the run length L of the run and the repeating symbol "t" of the following run is shown in Table 1. For example, consider the encoding of a string "X1111XX" whose length M is 7. The first run "X" is followed by another run "1111." The length of the first run L is 1 and the first run makes a transition from "X" to "1." Using the table, the transition is encoded as $L + M$,

2. The number of logic values employed depends on ATEs. For example, Advantest ATEs employ eight logic values, which distinguish logic values of input pins from output pins.

Test Data E_i

1	2	3	4
0	X	0	X
1	X	0	X
1	0	0	1
0	1	0	X
X	X	0	X

Encoded Data

Length M: (5)
Column 1: (0, 3, 1, 7, 6)
Column 2: (X, 3, 2, 1,1)
Column 3: (0, 0)
Column 4: (X, 2, 7, 1)

Fig. 6. Encoded test data.

which is 8. The next run to work on is "1111," which is followed by "XX." The run length L is 4 and the transition is from "1" to "X." It is encoded as L, which is 4. The last run "XX" does not make any transition at the end. Hence, no encoding is necessary for the last run. In order to restore the original sequence, the proposed encoding scheme requires the starting symbol of a sequence, the activity (i.e., the total number of transitions) and the length. Since the length of all sequences is the same for a test set, it is necessary to record the length only once for each test set E_i. Hence, the requirement to record the length of a sequence is ignored in subsequent discussions.

The encoded data of a small test set E_i (assuming run-length coding is applied to every column) is given in Fig. 6. The first element of each column denotes the starting symbol, the second element the activity α of the column, and the remaining ones the run lengths and their transitions obtained using Table 1. The column numbers, parentheses, and commas are only for readability and are not stored in the encoded data.

The decoding of the data encoded by the proposed scheme is as follows: Suppose that the symbol of the current run is known. Note that the symbol of the first run for each sequence, i.e., column, is given. The integer, say i, corresponding to the current run (viz. 1 for the first run in column 1, 7 for the second run, and 6 for the third run) is decoded to obtain the length of the current run and the symbol of the following run. The length of the current run is computed as $(i\%M)$, where % denotes the modulus operation and M is the length of the sequence. Let j be $\lceil i/M \rceil$, where $\lceil x \rceil$ is the smallest integer $\geq x$. Then the symbol of the next run is the jth character following the current one in the circular fashion given in Fig. 7.

For example, the starting symbol of column 4 for the encoded data in Fig. 6 is X. Hence, X is the symbol for the first run of column 4. The corresponding integer for the first run is 7. The length of the run is obtained as $(7\%5)$,

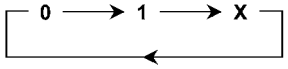

Fig. 7. Determination of the next symbol.

which is 2. The number j is obtained as $\lceil \frac{7}{5} \rceil$, which is 2. Hence, the symbol of the second run is the second character following X in the cycle in Fig. 7. Hence, the symbol for the second run is 1. A formal procedure for decoding of encoded data is described below.

```
Length ← M;
// Start processing a column.
Current_Symbol ← the starting symbol;
Loop_Count ← the activity α;
while (Loop_Count > 0) do
 {Transition_Info ← the next integer;
 Repeat Current_Symbol by (Transition_Info
  % Length) times;
 Find the character t for the following run as explained
  in the above.
 Current_Symbol ← t;
 Decrement Loop_Count by 1.
 } // End While
Repeat Current_Symbol to fill up the remaining
 positions of the column.
// End processing a column.
```

The numbers of bits necessary to represent a sequence without any coding, with the conventional run-length encoding scheme, and with the proposed run-length encoding scheme are given in Table 2. The following notations are used in the expressions and in the following discussions:

- $|A|$:the number of symbols in a sequence,
- M : length of the sequence,
- α: the activity of the sequence, and
- $\lceil x \rceil$: the smallest integer $\geq x$.

For the expressions in Table 2a, $\lceil \log_2 |A| \rceil$ is the number of bits needed to represent a symbol. For the conventional encoding scheme, the term $(\alpha + 1)$ is the number of runs. $\lceil \log_2 M \rceil$ is the number of bits necessary to represent a run length, which is determined by the longest run (which is M). The first two terms in the proposed encoding scheme account for the starting character and the activity. (The maximum value of the activity of a column is assumed to be $M - 1$ for the time being.) The integer necessary to

TABLE 2a
The Number of Bits Required to Represent a Sequence

Scheme	Number of bits
No coding	$M\lceil \log_2 \lvert A \rvert \rceil$
Conventional encoding scheme	$(\alpha+1)(\lceil \log_2 \lvert A \rvert \rceil + \lceil \log_2 M \rceil)$
Proposed encoding scheme	$\lceil \log_2 \lvert A \rvert \rceil + \lceil \log_2 M \rceil + \alpha \lceil \log_2 (M(\lvert A \rvert -1)-1) \rceil$

(a) General case.

TABLE 2b
The Number of Bits Required to Represent a Sequence

Scheme	Number of bits
No coding	$2M$
Conventional encoding scheme	$(\alpha+1)(\lceil \log_2 M \rceil+2)$
Proposed encoding scheme	$(\alpha+1)(\lceil \log_2 M \rceil+1)+1$

(b) The case for $|A| = 3$ and $M >> 1$.

represent a transition for the proposed encoding scheme is no larger than $(M(|A|-1)-1)$. Hence, the proposed scheme needs $\lceil \log_2(M(|A|-1)-1) \rceil$ bits to represent an integer for a transition. Readers may verify that the number of bits required for the proposed method is always less than or equal to that for the conventional encoding scheme.

Table 2b shows the case for large test data sets where $|A| = 3$ and $M >> 1$. It shows that the proposed encoding scheme is always more efficient than the conventional encoding scheme provided the activity α is greater than 0. The table also reveals that run-length coding fails to compress the test sequence of a column when the activity α of the column is over a certain threshold value. This is illustrated in Fig. 8 for the case of $M = 1,000$. If the activity α is over 181 in the figure, it is more advantageous not to apply run-length coding. Details on the threshold value are discussed in the following.

3.3 Threshold Value for Run-Length Coding

The expressions in Table 2 show that, as the activity of a column becomes larger, run-length coding becomes less efficient. In fact, the length of a compressed sequence is roughly proportional to the activity α. If α is greater than a threshold value α_t, run-length coding fails to compress the data. Hence, it is better not to apply run-length coding. From equations in Table 2a, the following equality holds for $\alpha = \alpha_t$.

$$\lceil \log_2 |A| \rceil + \lceil \log_2 M \rceil + \alpha_t \lceil \log_2(M(|A|-1)-1) \rceil = M \lceil \log_2 |A| \rceil.$$

Hence, the threshold value α_t for the proposed encoding scheme is

$$\alpha_t = \left\lceil \frac{M\lceil \log_2 |A| \rceil - \lceil \log_2 M \rceil - \lceil \log_2 |A| \rceil}{|log_2(M(|A|-1)-1)|} \right\rceil.$$

The above equation is simplified, as in Table 2b, for $|A| = 3$ and $M >> 1$,

$$\alpha_t = \left\lceil \frac{2M-1}{\lceil \log_2 M + 1 \rceil} - 1 \right\rceil.$$

The threshold value α_t increase as M increases and the trend is shown in Fig. 9. Each discontinuous point in the graph in Fig. 9 corresponds to a length M with a power of 2. If the activity of a column exceeds the threshold value α_t, we propose not to apply run-length coding to the column.

Let us reexamine the number of bits necessary to store the activity, say $N(\alpha)$, in each column for the proposed encoding scheme. So far, we have considered $N(\alpha)$ to be $\lceil \log_2 M \rceil$ assuming the maximum value of the activity of a column is $(M-1)$, where M is the length of the sequence. However, we apply run-length coding to a column only if the activity of the column is less than the threshold value α_t. Hence, the necessary number of bits $N(\alpha)$ can be reduced to $\lceil \log_2 \alpha_t \rceil$. For the above example, $M = 1,000$ and $|A| = 3$, the threshold value α_t is 181. So the number of bits necessary to store the number of transitions, $N(\alpha)$, is 8 bits instead of 10 bits (which is $\lceil \log_2 M \rceil$, where $M = 1,000$).

The number of bits $N(\alpha)$ can be further reduced if a column satisfies the following condition. For any column c_i

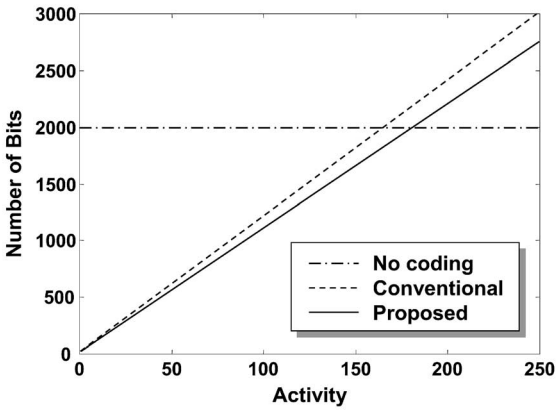

Fig. 8. Number of bits required for the three schemes with $M = 1,000$ in Table 2b.

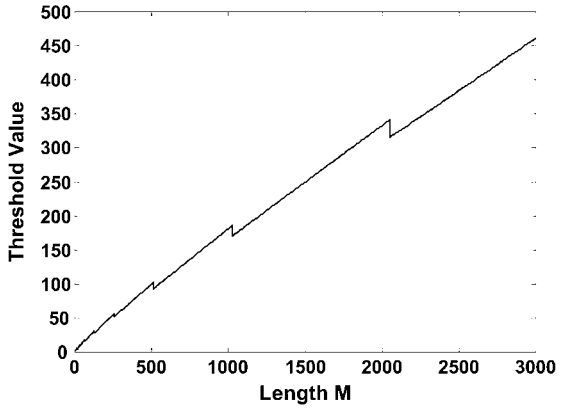

Fig. 9. Threshold value of a sequence.

with activity $\alpha(c_i) < \alpha_t$, it is also true that $\alpha(c_i) < \alpha_t^*$, where $\alpha_t^* \leq \alpha_t$. We call α_t^* an **actual** threshold value. If a data set E_i satisfies such a condition, $N(\alpha)$ can be reduced to $\lceil \log_2 \alpha_t^* \rceil$ from $\lceil \log_2 \alpha_t \rceil$ for the data set E_i. For example, consider a test set E_i whose column activities are $\{3, 10, 2, 3, 1, 8\}$. Suppose that the threshold value α_t of the test set is 6. Run-length coding is applied only to the four columns whose activity is less than 6. Note that the activities of the four columns are less than 4. Hence, the actual threshold value of the test set is 4 instead of 6. The number of bits necessary to store the activity $N(\alpha)$ is 2 (which is obtained as $\lceil \log_2 4 \rceil$) instead of 3 $(= \lceil \log_2 6 \rceil)$. As the actual threshold value α_t^* of a test set depends on the test data, it is necessary to record the actual threshold value for each test set E_i. The reduction of the threshold value from α_t to α_t^* increases the compression ratio for some test sets. Finally, it should be noted that changing the number of bits necessary to store the activity $N(\alpha)$, in turn, changes the threshold value α_t. Hence, the threshold value obtained in the above equation is not exact, but the accuracy may be sufficient for practical purposes.

To enable decoding of the encoded data, it is necessary to put another flag with each column to indicate whether run-length coding has been applied to the column or not. So, a total of two flags, one to indicate the application of the BW transformation and the other to indicate the application of run-length coding, are necessary for each column. As the number of columns of a test data set, i.e., the number of pins for the circuit, is much less than the number of rows, i.e., the number of test patterns, the impact of the additional two bits required for each column is negligible to the overall performance.

3.4 Decompression

All the operations performed on the original test data are reversible. Hence, the encoded data can be decompressed and the process is lossless. It should be noted that the inverse operation of the BW transformation is much simpler than the BW transformation, as it does not require a sorting process. For details of the inverse operation of the BW transformation, refer to [16] and [17].

Two major advantages of the proposed compression scheme are the simplicity of the decompression scheme and easy parallelization of the decompression process in hardware. The simple decompression scheme, which is an important requirement in compressing test data, reduces the decompression time to result in the reduction of the overall download time. A significant reduction of the decompression time can also be achieved through a dedicated hardware. When a dedicated hardware is employed for decompression, multiple columns can be processed in parallel for the proposed method. Note that the decompression process of a column is independent of the decompression of other columns for the proposed method. The parallel operation cuts down further the decompression time at the cost of higher complexity for the hardware. Considering the high price of ATE, the hardware cost may well be justified.

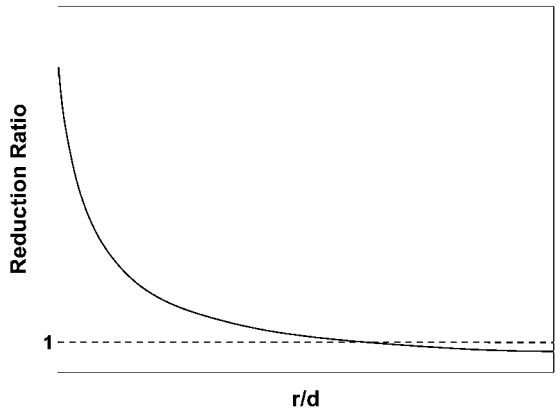

Fig. 10. Reduction ratio of the download time versus r/d.

3.5 Reduction of Download Time

The download time of test date depends on the test set size, the compression ratio, the decompression speed, and the data transfer rate of the network. Let us denote:

- s: size of an original test data (bits),
- c: compression ratio,
- d: decompression speed (bits/second), and
- r: transfer rate of the network (bits/second).

It should be noted that the decompression speed is measured as the number of compressed bits processed per second. The download time of an original data is

$$t_{\text{orig}} = \frac{s}{r},$$

while the download time of a compressed data is

$$t_{\text{orig}} = \frac{s}{cr} + \frac{s}{cd} = \frac{s}{c}\left(\frac{1}{r} + \frac{1}{d}\right).$$

The first term for the compressed data is the data transfer time and the second term is the decompression time. The reduction ratio of download time of original data to the download time of the compressed data is given as

$$\frac{t_{\text{orig}}}{t_{\text{comp}}} = \frac{\frac{s}{r}}{\frac{s}{c}\left(\frac{1}{r} + \frac{1}{d}\right)} = \frac{c}{r\left(\frac{1}{r} + \frac{1}{d}\right)} = \frac{c}{1 + \frac{r}{d}}. \quad (1)$$

The reduction ratio increases as the compression ratio c and the decompression speed d increases and as the transfer rate r decreases. Note that the ratio is independent of the size of the test data. The reduction ratio versus the ratio of the network speed to the decompression speed, r/d, is shown in Fig. 10. The performance approaches to its maximum value c as r/d approaches 0. As r/d increases, the reduction ratio decreases to reach a critical point where the compressed data set fails to reduce the download time any longer.

4 Experimental Results

We measured the performance of the proposed method on large test data sets of four real-life industry chips and compared the results with other compression methods. The test sets (named after their circuits) are:

TABLE 3
Compression Ratios for Various Sizes of D_i

Test Set	# patterns (x1000)	Size of Submatrices D_i							
		512	1024	2048	4096	8192	16384	32768	65536
A1	20	10.27	11.66	12.29	14.37	15.1	16.22	**17.69**	-
A2	15	37.31	49.12	56.95	63.07	**65.67**	61.77	-	-
K1	45	31.74	53.91	83.96	116.96	140.91	**154.08**	145.89	135.40
K2	31	**3.89**	3.76	3.60	3.41	3.27	3.08	3.02	-
K3	33	6.73	6.67	6.33	6.09	6.11	7.11	**8.53**	8.26
P1	103	41.45	69.19	142.02	218.77	338.65	472.31	636.94	**878.68**
P2	65	36.54	70.96	135.07	235.73	339.13	396.88	469.46	**492.37**
P3	65	51.68	75.02	97.17	111.45	122.3	**124.33**	122.49	117.58
P4	842	16.85	18.18	19.02	18.92	18.08	16.80	29.15	**54.91**
P5	209	95.91	178.84	335.96	581.63	944.57	1391.86	1853.66	**2377.54**
P6	1027	38.12	68.33	132.6	238.25	438.14	782.25	1316.54	**2095.79**
P7	457	35.07	74.15	156.7	347.07	726.49	1454.54	1847.35	**2044.36**
P8	498	34.55	72.83	153.46	340.05	725.33	1489.16	1875.12	**2096.43**
S1	33	113.17	204.31	358.38	599.18	932.12	1317.4	1671.83	**1836.52**
S2	12	94.31	155.30	236.34	324.35	417.02	**502.22**	-	-

- A1, A2 : test sets for a disk controller,
- K1 - K3 : test sets for a CISC microcontroller,
- S1, S2 : test set for a RISC microcontroller, and
- P1 - P8 : test set for a CD-ROM controller.

Each test set is a subset of the test set for a circuit and each subset is intended to test one or a few modules of the circuit. The size of test sets is in the range of 15,000 to one million test patterns. All the test sets contain only three logic values, 0, 1, and X.[3] The proposed method was coded in the C programming language and the program runs on workstations under the UNIX environment. All the experiments were performed on a Sun Ultra 2 workstation.

In the proposed method, a test set is partitioned into submatrices D_i of size $M \times Q$. The size M, i.e., the number of rows, of a submatrix D_i affects the compression ratio and is sensitive to the compression time (which is not important for test data compression). The first experiment was to observe the effect of the size of D_i on the compression ratio. The experimental results are shown in Table 3.

From the experiment, the general trend is that as the size of submatrices increases, the compression ratio also increases until it reaches the peak point (boldface in the table). All the test sets, except one, reach their peak for the maximum or a large size of the submatrices. This is because the BW transformation and run-length coding becomes, in general, more effective for a larger block size. A compression ratio increases rapidly until it reaches the peak point, but decreases slowly after the peak point. Hence, it is a good idea to make the size of submatrices as large as possible, as long as the processing time of the BW transformation is acceptable. It should be noted that as the size of submatrices increases, the compression time increases sharply because of the sorting process necessary for the BW transformation. For example, the compression takes about 49 CPU seconds for the largest test set P8 with about one half million patterns when the size of submatrices is 512, but the time increases to 52.4 CPU hours when the size is 65,536. *For the subsequent experiments, the size of the submatrices for each test set is set to its maximum size or 65,536, whichever is smaller.*

The efficiency of the proposed method is based on the claim that the BW transformation reduces the activity of test sets. In the following, we present experimental results regarding the activity of test data before and after the application of the BW transformation. Note that some of the columns of E_i are not BW transformed. Some column headings of Table 4 are described below:

- # pins : number of pins of the tested module(s),
- # patterns : number of test patterns in thousands,
- Size : the size of the test set in Mbytes, (computed as number of test patterns $\times$ number of pins $\times 2$ bits per symbol / (8×10^6)),
- $\alpha(D)$: the average activity of pins for the original test set,
- $\alpha(E)$: the average activity of pins for the partially BW transformed test set, and
- Reduction Ratio: the ratio of $\alpha(D)$ to $\alpha(E)$.

From Table 4, the activity of the BW transformed test sets is, on average, 86.2 times less than that of the original test sets. The high reduction increases the efficiency of run-length coding for a BW transformed test set E_i which, in turn, increases the efficiency of the proposed method. For example, the reduction ratio of four test sets, P1, P5, S1, and S2, is over 160, and the proposed method achieves a high compression ratio for these test sets (as to be shown in the next table). As mentioned in Section 2.3, the BW transformation is effective in reducing the activity for a sequence with many repeating cycles. We found that a large block of test set S1 consists of binary numbers in ascending order. This explains the large reduction ratio for test set S1.

The next experiment we performed was to compare the compression ratio of the proposed method with that of six well-known compression methods, Huffman with 59,049 $(= 3^{10})$ symbols [1], arithmetic coding with

3. It requires two bits to represent each symbol. Hence, the compression rate is equal to 2/compression ratio for the test sets.

TABLE 4
The Performance of BW Transformations on Test Sets

Test Set	# pins	# patterns (x1000)	Size (Mbytes)	$\alpha(D)$	$\alpha(E)$	Reduction Ratio
A1	143	20	0.68	1508.02	139.94	10.78
A2	143	15	0.51	253.35	31.22	8.11
K1	96	45	1.03	64.02	38.14	1.68
K2	96	31	0.7	2938.93	2675.31	1.1
K3	96	33	0.76	2704.22	1135.5	2.38
P1	33	103	0.81	4053.36	12.27	330.35
P2	89	65	1.38	57.01	14.56	3.92
P3	89	65	1.39	889.79	64.29	13.84
P4	15	842	3.01	17518.73	1793.07	9.77
P5	84	209	4.19	1315.73	7.96	165.29
P6	33	1027	8.08	4053.67	49.67	81.61
P7	89	457	9.71	88.44	21.90	4.04
P8	89	498	10.57	90.82	23.06	3.94
S1	120	33	0.94	546.73	1.52	359.69
S2	120	12	0.36	751.6	2.53	297.08
Avg	89.00	230.47	2.94	2455.63	400.73	86.24

TABLE 5
Compression Ratios of Various Compression Methods

Test Set	Norm. $\alpha(D)$	Norm. $\alpha(E)$	Huff	Arith	LZW	LZSS	Compress	Gzip	Prop.
A1	754	70.0	1.91	1.89	2.40	2.37	3.23	3.73	**17.69**
A2	168.9	20.8	2.26	2.26	4.99	6.87	8.29	23.78	**61.77**
K1	14.2	8.4	3.10	3.12	9.22	25.82	22.75	**176.57**	135.40
K2	957.4	871.6	2.38	2.40	3.88	3.48	6.20	**9.62**	3.02
K3	810.1	340.1	2.68	2.66	5.61	4.65	9.00	**11.12**	8.26
P1	393.6	1.2	3.74	3.76	15.50	7.21	27.09	220.07	**878.68**
P2	8.8	2.2	3.17	3.23	19.68	27.19	42.76	190.75	**492.37**
P3	136.0	9.8	2.88	2.91	10.28	11.70	19.11	104.11	**117.58**
P4	208.0	21.3	3.91	3.95	5.08	7.29	20.69	50.18	**54.91**
P5	62.9	0.4	3.69	3.75	17.86	22.80	73.44	233.32	**2377.54**
P6	39.5	0.5	3.86	3.88	42.95	22.22	117.37	245.79	**2095.79**
P7	1.9	0.5	2.96	2.97	14.87	28.58	34.31	242.50	**2044.36**
P8	1.8	0.5	2.96	2.98	14.95	28.60	33.77	244.12	**2096.43**
S1	166	0.5	3.89	3.88	6.83	10.12	10.13	12.15	**1836.52**
S2	603.5	2.0	3.98	3.90	5.67	7.89	9.91	11.53	**502.22**
Avg	288.4	90.0	3.19	3.21	12.09	15.43	26.84	64.99	94.63

59,049 symbols [2], LZSS with the dictionary window size of 512 and the look-ahead buffer size of 256 [7], LZW with the dictionary size of 32K [5], Compress, and Gzip with option -9 (which is for the best performance). For the experiments, we implemented Huffman, arithmetic, LZW, and LZSS methods based on the programs available in [8] and used UNIX and GNU utilities for "Compress" and "Gzip," respectively.

Table 5 shows compression ratios achieved for the seven different methods. Some column headings for the table are:

- Norm. $\alpha(D)$: normalized average activity of the original data $D \times 10^{-4}$ (computed as the average activity of pins for the original test set D / no. of test patterns $\times 10^{-4}$),
- Norm. $\alpha(E)$: normalized average activity of the partially BW transformed data $E \times 10^{-4}$,
- Huff : Huffman method,
- Arith. : arithmetic coding method,
- LZW : Lempel-Ziv-Welch method,
- LZSS : Lempel-Ziv-Storer-Szymanski method,

TABLE 6
Reduction Ratio of the Download Time for the Proposed Method

Test Set	Comp. Ratio	Decomp. Speed (Kb/sec)	Network Speed (Mb/sec)				
			0.5	1	2	5	10
A1	17.69	106.0 53010.0	3.1 17.5	1.7 17.4	0.9 17.0	0.4 16.2	0.2 14.9
A2	61.77	35.3 17670.0	4.1 60.1	2.1 58.5	1.1 55.5	0.4 48.1	0.2 39.4
K1	135.4	15.9 7965.0	4.2 127.4	2.1 120.3	1.1 108.2	0.4 83.2	0.2 60.0
K2	3.02	659.3 329655.0	1.7 3.0	1.2 3.0	0.7 3.0	0.4 3.0	0.2 2.9
K3	8.26	301.9 150970.0	3.1 8.2	1.9 8.2	1.1 8.2	0.5 8	0.2 7.7
P1	878.68	2.7 1325.0	4.6 637.9	2.3 500.8	1.2 350.2	0.5 184.1	0.2 102.8
P2	492.37	5.3 2655.0	5.2 414.3	2.6 357.7	1.3 280.8	0.5 170.8	0.3 103.3
P3	117.58	20.9 10445.0	4.7 112.2	2.4 107.3	1.2 98.7	0.5 79.5	0.2 60.1
P4	54.91	37.0 18490	3.8 53.5	2.0 52.1	1.0 49.6	0.4 43.2	0.2 35.6
P5	2377.54	1.3 655.0	6.2 1348.3	3.1 941.0	1.6 586.5	0.6 275.4	0.3 146.2
P6	2095.79	1.2 615.0	5.1 1156.0	2.6 798.1	1.3 492.9	0.5 229.5	0.3 121.4
P7	2044.36	1.6 785.0	6.4 1248.9	3.2 899.1	1.6 576.2	0.6 277.4	0.3 148.8
P8	2096.43	1.5 755.0	6.3 1261.2	3.2 901.9	1.6 574.5	0.6 275.0	0.3 147.2
S1	1836.52	1.6 815.0	6.0 1138.2	3.0 824.7	1.5 531.7	0.6 257.4	0.3 138.4
S2	502.22	5.1 2535.0	5.0 419.5	2.5 360.1	1.3 280.7	0.5 169.0	0.3 101.6
Avg	94.6	79.8 39889.7	4.6 533.8	2.4 396.7	1.2 267.6	0.5 141.3	0.3 82.0

- Compress : UNIX utility "compress,"
- Gzip : GNU utility "gzip," and
- Prop. : proposed method.

It should be noted that the average compression ratio of a compression method given in the last row is computed as the ratio of the total size of the entire 15 test sets before compression to the total size of the compressed test sets by applying the compression method.

As shown in the table, the average compression ratio of the proposed method is 94.6, while that of the next best method (Gzip) is 65.0. Among the conventional methods, Gzip performs the best and Huffman coding the worst. Huffman coding usually performs better than run-length coding, which is employed in the proposed method. In this case, the proposed method performed better than Huffman coding because the benefit of the BW transformation offsets the inefficiency of run-length coding.

As expected, the compression ratio of a test set for the proposed method is roughly inversely proportional to the normalized activity of the partially BW transformed data E. A lower normalized activity $\alpha(E)$ makes run length coding more efficient, which, in turn, makes the proposed method to achieve a higher compression ratio. The proposed method achieves the compression ratio of over 1,800 for five tests (P5 -P8 and S1) whose normalized activity $\alpha(E)$ is below 0.5×10^{-4}. A low normalized activity $\alpha(E)$ of a test set is due to the low normalized activity of the original test set $\alpha(D)$ (viz. P7 and P8) and/or the high reduction ratio of the BW transformation (viz. P1, S1 and S2). In contrast, the normalized activity $\alpha(E)$ of test sets K2 and K3 is high. So they lead to the low compression ratio for the proposed method. We noticed that two test sets are highly random to result in high normalized activity $\alpha(D)$ and low reduction ratio for the BW transformation.

The ultimate performance of a compression method for test data is measured in terms of the reduction of the download time. The download time for a compressed data set includes the data transfer time and the decompression time. We coded the decompression algorithm in the C language, called software decompressor, and measured the decom-

pression speed on a workstation. In order to gauge the capability of a dedicated hardware in reducing decompression time, we also have implemented a prototype of a dedicated hardware decompressor for the proposed method. The hardware decompressor decompresses 16 columns in parallel and runs at 100 MHz of clock speed. It contains about 380K equivalent 2-input NAND gates and 16 memory modules with size 128 Kbits each. It is estimated that the hardware decompressor would reduce the decompression speed by about 500 times compared with the software decompressor.

Table 6 presents the reduction ratio of download time for the proposed method computed based on (1) in Section 3.5. The reduction ratios were obtained for five different network speeds, 0.5, 1, 2, 5, and 10 Mbits/seconds (Mb/sec). Column heading "Decomp.Speed (Kb/sec)" in the table denotes the decompression speed of the two decompressors, software decompressor (top item) and the hardware decompressor (bottom item), in Kbits per second. The speed of the hardware decompressor was obtained as 500 times that of the software decompressor. The top item of an entry under heading "Network Speed (Mb/sec)" represents the reduction ratio under the employment of the software decompressor and the bottom entry represents the ratio for the hardware decompressor.

The average reduction ratio of download time for the software decompressor is 4.6 for a slow network of 0.5 Mb/sec, and the ratio is below 1.0 for the speed of 5 Mb/sec and of 10 Mb/sec. The software decompressor reduces the download time only when the network operates at the speed of 2 Mb/sec or below. For the hardware decompressor, the average reduction is 534 for the slow network of 0.5 Mb/sec and is reduced to 82 for a high speed network of 10 Mb/sec. So a hardware decompressor reduces download time significantly for the considered network speeds. Considering the significant difference in the reduction ratio between the software and the hardware decompressors and the high cost of ATE, it is worth employing a hardware decompressor to ATE.

The reduction ratio of the download time is increased by increasing the compression ratio and/or the decompression speed. A sophisticated compression method may increase the compression ratio, but it may decrease the decompression speed to offset the benefit of the increased compression ratio. A salient advantage of the proposed method lies in a simple decompression algorithm which can be operated in parallel. The algorithm leads to easy hardware implementation running in parallel. A higher decompression speed can be achieved for the proposed method by simply increasing the parallelism at higher hardware cost (which is insignificant for ATE).

The experimental results presented above show that the proposed method is suitable for test data compression. The efficiency of the proposed method in compression ratio is owing to the computationally intensive BW transformation employed in the compression process. The proposed method significantly reduces the download time especially when a hardware decompressor is employed.

5 Summary

Today's variety of ASICs and decreased production volume of individual types of ASICs requires more frequent downloads of test data sets from workstations to automatic test equipment (ATE). The overall throughput of an ATE is sensitive to the download time. An efficient method to reduce the download time is to compress the test data.

A compression algorithm for test data should meet two requirements: It should be lossless and the decompression should be simple. As the decompression is performed on the ATE side, the decompression time should be minimized to reduce the overall download time. A characteristic of test patterns is that subsequent test patterns on the same pin are strongly correlated, but the patterns on different pins are weakly correlated. Most existing data compression methods do not exploit the above characteristics and, hence, are ineffective in test data compression.

In this paper, we presented a new test data compression method that aims to fully utilize the unique characteristics of test data compression. The key idea of the proposed method is to perform the BW transformation on individual test sequences of pins and then to apply run-length coding. The BW transformation reduces the number of transitions for the test sequences, which leads to a high compression ratio for the proposed method. The BW transformation is computationally complex, but the reverse operation is simple. Thus, the proposed method achieves high decompression speed.

The experimental results show that the proposed compression method performs better than six other compression methods for compressing test data. The average compression ratio of the proposed method performed on 15 test data sets is 94.6, while that for the next best method, Gzip, is 65.0. The high compression ratio for the proposed method reduces the download time of test data substantially. The reduction ratio of download time is 4.6, on average, for a slow network of 0.5 Mb/sec with a software decompressor. The ratio increases to 534 when a hardware decompressor is employed. Overall, the proposed method exploits the unique characteristics of test data compression in which high decompression speed is essential, but low compression speed is acceptable. It is effective and suitable for compressing test data, and reduces the download time of test data substantially.

Acknowledgments

The authors would like to thank S. Sugamori of Adantest Corporation. Discussion with him suggested that test data compression is an important area to work in. The work performed by Dong S. Ha was supported in part by the US National Science Foundation under the grant number CCR-9730372.

References

[1] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098-1101, Sept. 1952.

[2] J.J. Rissanen and G.G. Langdon, "Arithmetic Coding," *IBM J. Research and Development*, vol. 23, no. 2, pp. 149-162, Mar. 1979.

[3] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Information Theory,* vol. 23, pp. 337-343, May 1977.
[4] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Trans. Information Theory,* vol. 24, pp. 530-538, Sept. 1978.
[5] T.A. Welch, "A Technique for High-Performance Data Compression," *Computer,* vol. 17, no. 6, pp. 8-19, June 1984.
[6] S. Apiki, "Lossless Data compression," *Byte,* vol. 16, no. 3, pp. 309-312, 314, 386-387, Mar. 1991.
[7] J.A. Storer and T.G. Szymanski, "Data Compression via Textual Substitution," *J. ACM,* vol. 29, no. 4, pp. 928-951, Oct. 1982.
[8] M. Nelson and J.-L. Gailly, *The Data Compression Book.* M&T Books, 1996.
[9] G. Held, *Data and Image Compression.* John Wiley & Sons Ltd., 1996.
[10] R. Hoffman, *Data Compression in Digital Systems.* Chapman & Hall, 1997.
[11] C.-A. Chen and S. Gupta, "A Methodology to Design BIST Test Pattern Generators," *Proc. Int'l Test Conf.,* pp. 814-823, Oct. 1995.
[12] K. Chakrabarty, B.T. Murray, J. Liu, and M. Zhu, "Test Width Compression for Built-In Self-Testing," *Proc. Int'l Test Conf.,* pp. 328-337, Nov. 1997.
[13] V. Iyengar, K. Chakrabarty, and B.T. Murray, "Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets," *Proc. VLSI Test Symp.,* pp. 418-423, Apr. 1998.
[14] T. Yamaguchi, M. Tilgner, M. Ishida, and D.S. Ha, "An Efficient Method for Compressing Test Data," *Proc. Int'l Test Conf.,* pp. 79-88, Nov. 1997.
[15] M. Ishida, D.S. Ha, and T. Yamaguchi, "COMPACT: A Hybrid Method for Compressing Test Data," *Proc. VLSI Test Symp.,* pp. 62-69, Apr. 1998.
[16] M. Burrows and D. Wheeler, "Block Sorting Lossless Data Compression Algorithm," Research Report 124, System Research Center, Digital System Research Center, Palo Alto, CA, May 1994.
[17] M.R. Nelson, "Data Compression with the Burrows Wheeler Transformation," *Dr. Dobb's J.,* pp. 46-50, Sept. 1996.
[18] B. Balkenhol and S. Kurtz, "Universal Data Compression Based on the Burrows-Wheeler Transformation: Theory and Practice," *IEEE Trans. Computers,* vol. 49, no. 10, pp. 1043-1053, Oct. 2000.
[19] K. Sadakane, "A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation," *Proc. Data Compression Conf.,* pp. 129-138, Mar. 1998.
[20] N.J. Larsson, "The Context Trees of Block Sorting Compression," *Proc. Data Compression Conf.,* pp. 189-198, Mar. 1998.

Takahiro J. Yamaguchi received the BS degree in applied physics from Fukui University, Fukui, Japan, the MS degree in physics and the PhD degree in electronic engineering from Tohoku University, Sendai, Japan, in 1976, 1978, and 1999, respectively. He joined Advantest Corporation in 1978 and contributed to the development of Fourier analyzers. From 1991, he worked for Advantest Laboratories Ltd., Japan. His research interests are in the application of digital signal processing techniques for testing ADC, DAC, and PLL, as well as data compression methods, and contextual design. He is a member of the IEEE, ACM, and Information Processing Society of Japn.

Dong Sam Ha received the BS degree in electrical engineering from Seoul National University, Korea, in 1974, and the MS and PhD degrees in electrical and computer engineering from the University of Iowa, Iowa City, in 1984 and 1986, respectively. Since the Fall of 1986, he has been on the faculty of the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, Virginia, and is currently a professor. Prior to his graduate studies, he served as a research engineer for the Agency for Defense Development in Korea from 1975 to 1979. While on leave in 1996, he worked with the Semiconductor Research Center of Seoul National University in Seoul, Korea, where he investigated built-in self-test synthesis. Along with his students, he developed four CAD tools, two fault simulators, and two test generator for digital circuit testing. His current research interests include low-power VLSI antenna diversity at handsets, handset modems for 3G and 4G systems, wireless video systems, and analog and mixed-signal VLSI design. He is a senior member of the IEEE.

Masahiro Ishida Masahiro Ishida received the BS and MS degrees in electronic and information engineering from Tokyo University of Agriculture and Technology, Tokyo, Japan, in 1993 and 1995, respectively. Since 1995, he has worked at Advantest Laboratories Ltd., Japan. His current research interests include test data compression, power supply current testing, and jitter testing.

Tadahiro Ohmi received the BS, MS, and PhD degrees in electrical engineering from Tokyo Institute of Technology, Tokyo, in 1961, 1963, and 1966, respectively. Before 1972, he served as a Research Associate in the Department of Electronics, Tokyo Institute of Technology, where he worked on Gunn diodes and related semiconductor physics. In 1972, he moved to Tohoku University and he is presently a professor at the New Industry Creation Hatchery Center (NICHe), Tohoku University. He is engaged in research on high-performance ULSIs, such as ultra-high-speed ULDIs based on gas-isolated-interconnect metal-substrate SOI technology, high-speed flat panel displays, and advanced semiconductor process technology, such as low kinetic energy particle bombardment processes including high-quality oxidation/nitridation, high-quality metallization, very low temperature Si epitaxy, and crystallinity controlled film growth technologies from single-crystal, gain-size-controlled polysilicon and amorphous, highly selective CVD, highly selective RIE, high-quality ion implantation with low temperature annealing capability, etc., based on ultra-clean technology concept supported by newly developed ultra-clean gas supply system, ultra-high vacuum-compatible reaction chamber with self-cleaning function, ultra-clean wafer surface cleaning technology. He has published more than 800 original papers, holds 800 patents, and has won more than 10 authorized awards. He served as the president of the Institute of Basic SemiconductorTechnology-Development (Ultra Clean Society) from its inception to its termination in 2000. He is a member of the Institute of Electronics, the Institute of Electronics of Japan, the Japan Society of Applied Physics, and the ECS. He is a senior member of the IEEE and a fellow of Information and Communication Engineers of Japan.