A Variable-Length Coding Adjustable for Compressed Test Application

Hideyuki ICHIHARA†(a), Member, Toshihiro OHARA††, Michihiro SHINTANI†∗∗, Nonmembers, and Tomoo INOUE†*, Member

SUMMARY Test compression / decompression using variable-length coding is an efficient method for reducing the test application cost, i.e., test application time and the size of the storage of an LSI tester. However, some coding techniques impose slow test application, and consequently a large test application time is required despite the high compression. In this paper, we clarify the fact that test application time depends on the compression ratio and the length of codewords and then propose a new Huffman-based coding method for achieving small test application time in a given test environment. The proposed coding method adjusts both of the compression ratio and the minimum length of the codewords to the test environment. Experimental results show that the proposed method can achieve small test application time while keeping high compression ratio.

key words: test compression, variable-length coding, test application time, ATE, Huffman code, and test environment

1. Introduction

As the size and complexity of VLSI circuits increase, the size of test sets for the circuits also increases. The increase in test set size requires larger storage and longer time to transport test sets from the storage device of a VLSI tester (ATE) to the circuit-under-test (CUT). A compression / decompression of test data is an efficient method to overcome this problem.

Several compression methods for test input sets have been proposed [1]–[11]. These methods are based on some data compression techniques, e.g., run-length coding [1], binary coding [2], [11], XOR-Network [3], Huffman coding [4], [5], [9]–[11], Golomb coding [6], FDR coding [7], and VIHC coding [8] and so on. These methods can be divided into two categories according to the length of codewords: fixed-length coding [1]–[3] and variable-length coding [4]–[11]. For example, Huffman coding assigns blocks appearing frequently to short codewords. Like Huffman coding, it is expected that variable-length coding can achieve higher compression than fixed-length coding. In particular, the authors of [4] and [5] design decompressors for variable-length codings based on a given precomputed test set in order to achieve high compression.

In general, test application with variable-length coding is complicated compared to that with fixed-length one.

In order to decompress such codewords appropriately, the previous works [6]–[8] assume that the decompressor has a synchronizing feedback mechanism. The literature [12] proposed a decompressor with a buffer instead of such a synchronizing feedback mechanism. In [4], [5], [11] a sufficient condition concerning the relationship between the speeds of the compressed input data and decompressed output data must be satisfied.

In a case where a fast CUT is tested by a slow (non-state-of-the-art) tester, the above-mentioned sufficient condition (described in further detail later) is satisfied easily. In general, however, CUTs do not always operate much faster than ATEs due to the constraints on the delay and power consumption in scan chains [13]–[16]. As the case may be, in order to satisfy the sufficient condition, the ATE speed must be sufficiently reduced, and consequently the test application time will increase in spite of highly compressed test data.

In this paper, first, we discuss the relationship between the test application time and employed variable-length codes in the case where CUTs do not operate much faster than ATEs. This discussion expresses the fact that test application time depends on the compression ratio and the length of codewords. Following this discussion, we propose a new Huffman-based coding method with efficient test application. The proposed method can adjust the compression ratio and the codeword length in order to decrease test application time.

This paper is organized as follows. Section 2 illustrates the compression method using Huffman coding [5] and its decompressor as an example of the compression methods using variable-length coding. In Sect. 3 we discuss the relationship between the test application time and employed variable-length codes. In Sect. 4, a coding algorithm for achieving efficient test application is proposed. Section 5 shows some experimental results of the proposed coding algorithm, and Sect. 6 concludes this paper.

2. Test Input Compression Using Variable Length Coding

2.1 Encoding Test Sets by Huffman Coding

In this section, first, we illustrate the method proposed in [5] as an example of test compression using variable length coding. To encode a given test set, each test vector of the test...
set is partitioned into several \( n \)-bit blocks, i.e., each block is an \( n \)-bit pattern. Figure 1 shows a test set which consists of eight test vectors partitioned into 4-bit blocks. The reason for partitioning a test vector into blocks is to keep the complexity of a decompression circuit and decompression delays low. Each block pattern is mapped to a variable-length codeword. The length of a codeword depends on the frequency that each pattern appears in the test set. The more frequently a pattern occurs, the shorter the length of a codeword for it is. Table 1 shows the frequency \( f_i \) of occurrence of each distinct pattern \( x_i \) in the test set of Fig. 1, and the corresponding Huffman codewords. The compression ratio \( r \) is defined by \( D_x/D_r \), where \( D \) and \( D_r \) are the sizes of a test set before and after compression, respectively. In this example, \( r = 91/128 \approx 0.71 \).

<table>
<thead>
<tr>
<th>distinct pattern ( x_i )</th>
<th>freq. ( f_i )</th>
<th>codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>5</td>
<td>011</td>
</tr>
<tr>
<td>0010</td>
<td>4</td>
<td>0101</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>0100</td>
</tr>
<tr>
<td>0100</td>
<td>2</td>
<td>0011</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>0110</td>
<td>1</td>
<td>00011</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td>00010</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>00001</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td>00000</td>
</tr>
</tbody>
</table>

2.2 Embedded Decompressor

When an encoded test set is applied to a CUT, it is decoded by an on-chip (embedded) decompressor. Figure 2 shows the decompressor proposed in [5]. It consists of a Huffman decoder and a serializer for 4-bit-blocked Huffman coding. Let \( v_{in} \) and \( v_{out} \) be the input and the output speed of the decompressor, respectively. Note that when both the numbers of input and output of the decompressor are single, like the decompressor in Fig. 2, the operation speeds of the ATE and the CUT correspond to \( v_{in} \) and \( v_{out} \), respectively.

Whenever receiving the whole of a codeword (i.e., the last bit of the codeword), the Huffman decoder sends out the corresponding block pattern. The decoded block pattern is stored to the serializer and scanned out to an internal scan chain of the CUT. The serializer receives block patterns from the decoder at irregular intervals owing to variable-length of Huffman coding, and then it shifts them out to the scan chain of the CUT.

The authors of [5] introduced a sufficient condition on the relationship between the input and output speeds to guarantee to decode the codewords appropriately instead of a synchronizing feedback between an ATE and a CUT. The sufficient condition is as follows.

\[
v_{in} \leq \frac{\text{min\_code\_length}}{b} \cdot v_{out}.
\]

where \( \text{min\_code\_length} \) is the length of the minimum codeword of all the codewords, and \( b \) is the size of a partitioned block. Since \( b > \text{min\_code\_length} \), the input speed \( v_{in} \) must be smaller than the output speed \( v_{out} \), i.e., the CUT must operate faster than the ATE.

As an example, Fig. 3 shows a time chart in the case where four codewords \( \langle 011, 00010, 011, 1 \rangle \) shown in Table 1, i.e., test vector \( t_1 \) of Fig. 1 is applied to the decompressor in order. Each codeword is immediately decoded to the corresponding block pattern, and then the block pattern is shifted out. In this case, since block size \( b \) is 4 and the minimum codeword length \( \text{min\_code\_length} \) is 1, the output speed must be four times faster than the input speed, i.e., \( v_{in} \leq 1/4v_{out} \). In Fig. 3, \( v_{in} = 1/4v_{out} \). In this case, test vector \( t_1 \) is applied appropriately. Figure 4 shows the time chart in the case where Inequality (1) is not satisfied. The input speed is half of the output speed, i.e., \( v_{in} = 1/2v_{out} \). In this case, since block size \( b \) is 4 and the minimum codeword length \( \text{min\_code\_length} \) is 1, Inequality (1) is not satisfied. At time 12, the block pattern corresponding to codeword 1 cannot go through the output of the decompressor because
the prior block pattern 0001 corresponding to codeword 011 is still read out from time 11 through 12.

3. Coding for Small Test Application Time

3.1 Huffman Coding Algorithm

This subsection briefly explains Huffman coding algorithm prior to the proposal of our coding. Huffman code is composed by using a coding tree like Fig. 5, which expresses a coding tree for the Huffman code in Table 1. In the coding tree, a block pattern corresponds to a leaf and its frequency are given to the leaf as a label, and the sum of the frequencies of a subtree is given to an internal each node. Moreover, the codeword of the block pattern corresponds to the leaf is denoted by sequence of the labels of “0” and “1” on the path from the root to each leaf. The length of the codeword of the block pattern corresponding to the leaf is the depth from the root to the leaf.

For the explanation of the Huffman coding algorithm, the coding tree $T$ is expressed by $T = (V, E, l, f)$, where $V$ is a set of nodes of the coding tree, $E$ is a set of branches, $l : E \rightarrow \{0, 1\}$ is a label assigned to the branch and $f : V \rightarrow \mathbb{R}$ (real numbers) is a label assigned to the node. A leaf corresponds to a distinct block pattern and the label $f$ of the leaf denotes the frequency of the distinct block pattern. Given a set of leaves, say $V_0$, and their labels $f(v)$ ($v \in V_0$), Huffman coding algorithm constructs the coding tree by the repetition of merging nodes (initially leaves) according to their frequencies.

[Huffman coding algorithm]

(1) Initially set $V = V_0$ and $E = \phi$.

(2) The following steps are repeated while $|V_0| > 1$.

(2-1) Let $a$ and $b$ be nodes whose $f(v)$s are the first and second smallest in $V_0$, respectively.

(2-2) Add a new node $c$ to both $V_0$ and $V$. Set $f(c) = f(a) + f(b)$.

(2-3) Delete nodes $a$ and $b$ from $V_0$.

(2-4) Add edges $e_1 = (a, c)$ and $e_2 = (b, c)$ to $E$. Assign $l(e_1) = 0$ and $l(e_2) = 1$.

(3) Return $T = (V, E, l, f)$.

3.2 Relationship between Code and Test Application Time

Next, we consider the relationship of the length of codewords on the test application time.

First, let us consider the the maximum input and output speeds (say $v_{in}$ and $v_{out}$ respectively) of the decompressor. Note that $v_{in}$ ($v_{out}$) must be smaller than $v'_{in}$ ($v'_{out}$). In general, the maximum speeds $v'$ and $v''$ are bounded by the performance of an ATE, the speed of scan operation of a CUT and so on. We call the relationship of $v'_{in}$ to $v'_{out}$ test environment.

For example, if $v'_{in}$ is 300 M [bps] and $v'_{out}$ is 400 M [bps], this test environment is denoted by $v'_{in} = 3/4v'_{out}$.

In general, $v_{in}$ and $v_{out}$ cannot be maximized simultaneously due to the constraints expressed by Inequality (1), shown in Sect. 2. Suppose that a test set encoded by the Huffman code shown in Table 1 is applied in test environment $v'_{in} = 3/4v'_{out}$. To minimize the test application time, the input and output speeds should be as large as possible, thus $v_{in} = v'_{in} = 3/4v'_{out}$ and $v_{out} = v'_{out}$. However, because the length of the minimum codeword is 1-bit, from Inequality (1), condition $v_{in} \leq 1/4v_{out}$ must be satisfied. Thus, the input speed $v_{in}$ is limited to $1/4v'_{out}$ while $v_{out} = v'_{out}$.

Here, recall the Huffman coding algorithm shown in 3.1. In the case where there are three or more nodes whose frequencies are equal, Huffman code is not always uniquely decided according to a given frequency of block patterns. That is, there are several selections of nodes in step (2-1) of the Huffman coding algorithm, shown in Sect. 3.1. In this case, the compression ratio doesn’t change no matter how two nodes to be merged are chosen from nodes with the same frequency.

Coding tree B shown in Fig. 6 is a Huffman tree derived from the test input set in Fig. 1. Although coding tree B achieves the same compression ratio as the coding tree A shown in Fig. 5, the minimum codeword length of the coding tree B is 2-bit, which is 1-bit longer than that of coding tree A. Using coding tree B instead of coding tree A, the condition $v_{in} \leq 1/4v_{out}$ obtained from Inequality (1) can be relaxed into $v_{in} \leq 1/2v_{out}$. As a result, this relaxed condition affords to set the input speed $v_{in}$ to the speed $1/2v'_{out}$ in the above-mentioned test environment $v'_{in} = 3/4v'_{out}$. Hence we
Table 2  Comparison of test application time for coding trees A, B, and C.

<table>
<thead>
<tr>
<th>coding tree</th>
<th>minimum codeword length</th>
<th>compression ratio (r)</th>
<th>condition from Inequality (1)</th>
<th>test environment (1): (\nu_in' = 3/4\nu_out')</th>
<th>test environment (2): (\nu_in' = 2/4\nu_out')</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0.711</td>
<td>(\nu_in \leq 1/4\nu_out)</td>
<td>1/4\nu_out' (1/4\nu_out')</td>
<td>1/4\nu_out' (1/4\nu_out')</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0.711</td>
<td>(\nu_in \leq 2/4\nu_out)</td>
<td>2/4\nu_out' (2/4\nu_out')</td>
<td>2/4\nu_out' (2/4\nu_out')</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>0.781</td>
<td>(\nu_in \leq 3/4\nu_out)</td>
<td>3/4\nu_out' (1.041\nu_out')</td>
<td>2/4\nu_out' (1.562\nu_out')</td>
</tr>
</tbody>
</table>

Fig. 7  Coding tree C (not Huffman tree).

can make a code with small test application time by lengthening the minimum codeword without losing the compression ratio.

Furthermore, if small loss of the compression ratio is permitted, we can compose a code to achieve faster test application. Let us consider coding tree C shown in Fig. 7. The minimum codeword of this code is 3-bit, while the application ratio.

The minimum codeword length, the compression ratio and the conditions obtained from Inequality (1) are shown. Under Columns test environment (1), since the condition obtained from Inequality (1) is \(\nu_in \leq 3/4\nu_out\), the test application time \(T\) of coding tree C is the shortest of the three. On the other hand, in the case of test environment (2): \(\nu_in' = 2/4\nu_out'\), the test application time corresponding to coding tree B is the shortest. Thus, in order to minimize test application time, we can compose a code for encoding a given test set according to a given test environment. In the next section, we proposes a coding algorithm that can flexibly adjust the compression ratio and the minimum codeword length.

Note that the maximum test application time is at most three times longer than the minimum one in Table 2 because we use a simple example such that the block size is 4 for the sake of simplicity. In practice, the block size is longer than 4, e.g., 8 or 16, so that the difference between the maximum and minimum test application times may become large.

4. Proposed Coding Algorithm

The proposed coding algorithm is based on Huffman coding. As shown in Sect. 3.1, the Huffman coding algorithm merges nodes according to the frequency \(f(v)\) of block patterns in order to minimize the compression ratio, while the proposed method merges nodes according to not only the frequency \(f(v)\) but also the height of a coding tree, i.e., the length of codeword. To achieve this, we introduce a new evaluation value \(F(v)\) instead of \(f(v)\).

A Huffman tree tends to be a tree such that the difference of the minimum and maximum depth of leaves is large, like Fig. 5. Thus, the minimum codeword length becomes as short as possible, and the compression ratio is high. On the other hand, if we compose a coding tree whose height is as small as possible, the resultant coding tree becomes a complete binary tree. The coding tree in Fig. 7 is complete. In this case, the minimum codeword length becomes equal or close to the maximum codeword length (the difference of them is one bit at most), but the compression ratio is low (no compression at the worst). Thus, a complete binary tree is the opposite of a Huffman one. The proposed coding algorithm can construct coding trees ranging between the complete binary tree and the Huffman tree.

Suppose the (ideal) frequency, say \(B_k\), of each node positioned at height \(k\) in a complete binary tree. The \(B_k\) is given by the following.

\[
B_k = 2^k \cdot \frac{N}{n}
\]

(2)

where \(N\) is the number of all blocks in a blocked test set, \(n\) is the number of distinct block patterns. Figure 8 shows \(B_k\) to each height \(k\) when \(N = 32\) and \(n = 8\).

The new evaluation value \(F(v)\) consists of the frequency \(f(v)\) and the ideal frequency of a complete binary
coding tree \( B_k \). Evaluation value \( F(v) \) is defined as follows.

\[
F(v) = (1 - \alpha)f(v) + \alpha B_k, \tag{3}
\]

where \( 0 \leq \alpha \leq 1 \) is a parameter.

We can control how a composed coding tree is close to a complete binary tree using parameter \( \alpha \). If parameter \( \alpha \) becomes large (small), the shape of the coding tree is close to that of a complete binary tree (Huffman tree). When \( \alpha = 0 \), the coding tree becomes a Huffman one, like coding tree \( A \) and coding tree \( B \). When \( \alpha = 1 \), it becomes a complete binary tree. Note that coding tree \( C \) is derived when \( \alpha = 0.8 \).

Given a set of leaves \( V_0 \), their labels \( f(v) \) \((v \in V_0)\) and parameter \( \alpha \), the proposed coding algorithm is shown below. Note that the main difference of the proposed coding algorithm from the Huffman coding algorithm shown in Sect. 3.1 is the evaluation value \( F(v) \) at steps (2-1) and (2-2).

**[Proposed coding algorithm]**

1. Initially set \( V = V_0 \) and \( E = \phi \).
2. The following processing is repeated while \( |V_0| > 1 \).
   1. \( \text{(2-1)} \) Let \( a \) and \( b \) be nodes whose \( F(v) \)'s are the first and second smallest in \( V_0 \), respectively.
   2. \( \text{(2-2)} \) Add a new node \( c \) to both \( V_0 \) and \( V \). Set \( F(c) = (1 - \alpha)(f(a) + f(b)) + \alpha B_k \). Here, \( k \) is the height of the subtree whose root is \( c \), that is \( k = \max(\text{height}(a), \text{height}(b)) + 1 \), where \( \text{height}(x) \) is the height of the subtree whose root is \( x \).
   3. \( \text{(2-3)} \) Delete nodes \( a \) and \( b \) from \( V_0 \).
   4. \( \text{(2-4)} \) Add edges \( e_1 = (a, c) \) and \( e_2 = (b, c) \) to \( E \). Assign \(|e_1| = 0\) and \( |e_2| = 1 \).
3. Return \( T = (V, E, l, f) \).

5. **Experimental Results**

We implemented the proposed coding algorithm in C language and applied it to test sets for ISCAS’89 benchmark circuits on Sun Blade 1000 workstation (CPU: Ultra SPARC IIIx2, Memory 1 GB). The test sets are generated by the test generation method proposed in [10], which aims to obtain test sets highly compressible by Huffman coding. In the experiments, the test vectors in the given test sets are partitioned into 4-bit blocks and encoded by the proposed coding. The compression ratio was calculated as \((\text{compressed test set size})/(\text{original test set size})\).

Table 3 shows experimental results for the proposed coding algorithm. In Table 3 we show the maximum codeword lengths (at Column max), the minimum codeword lengths (min) and the compression ratios (ratio) for \( \alpha = 0, 0.2, 0.6 \) and 1.0. When \( \alpha = 0 \), the obtained code is a Huffman one. For all the circuits, the difference between the minimum codeword length and the maximum codeword length becomes small as \( \alpha \) approaches 1, and then the coding tree becomes a complete binary tree when \( \alpha = 1 \). The compression ratio decreases as \( \alpha \) approaches 1. Note that the minimum codeword length is not always four bit even if \( \alpha = 1 \), because the number of the leaves, corresponding to the distinct block patterns, in a coding tree is smaller than \( 2^4 \). We can see that the proposed coding method can derive various codes whose compression ratio and minimum code length are controlled by parameter \( \alpha \).

Table 4 shows the test application time for Huffman coding and the proposed coding in a test environment \( v_{\text{in}} = 1/2v_{\text{out}} \). For each circuit, Table 4 shows the minimum codeword length, compression ratio and the test application time using Huffman coding and the proposed method. To obtain the test application time for the proposed method, we generated 100 codes as increasing \( \alpha \) from 0 to 1 by 0.01, and selected one code that can achieve the minimum test application time. Column \( \alpha \) under Proposed coding in Table 4 shows the value of \( \alpha \) when the best code is obtained. The last column of Table 4 shows the ratio of \( T_{\text{pro}} \) to \( T_{\text{huff}} \). The test application time is calculated as described in Sect. 3.2. Note that the proposed coding is identical to the Huffman coding when parameter \( \alpha = 0 \).

From Table 4, we can see the followings.

- For half of benchmarks, test application time for the proposed coding, \( T_{\text{pro}} \), is smaller than that for the Huffman one, \( T_{\text{huff}} \), in the given test environment. On the average, it is 82.6% of the test application time for the Huffman coding even though the compression ratio decreases by 0.065 (\( = 0.712 - 0.647 \)). Note that other experimental results show that the compression ratio and the ratio of \( T_{\text{pro}} \) and \( T_{\text{huff}} \), i.e., \( T_{\text{pro}}/T_{\text{huff}} \) become small when the block size increases.
- Especially for s344, s526, s526n, s1196 and s1238, the test application time for the proposed encoding can be reduced to about 60% of that for the Huffman coding. This is because the minimum codeword length of the proposed coding can be adjusted to a given test environment with keeping high compression ratio.
- For s13207, the test application time slightly decreases even though the minimum codeword length is lengthened from one to two bits. This is because the speed up effect (or lengthening the minimum codeword length by one bit) is canceled by the increase in the test data.

The above experimental results show the effectiveness of the proposed coding method if the best \( \alpha \) is given. Hence, in practice, a method of finding a (nearly) optimal \( \alpha \) with low computational effort is desired. As we can see from Table 3, the compression ratio increases monotonically with parameter \( \alpha \), while the minimum codeword length also in-
Table 3  Experimental results for proposed coding.
\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\text{circ} & \alpha=0 & \alpha=0.2 & \alpha=0.6 & \alpha=1 \\
\hline
\text{max} & \text{min} & \text{ratio} & \text{max} & \text{min} & \text{ratio} & \text{max} & \text{min} & \text{ratio} & \text{max} & \text{min} & \text{ratio} \\
\hline
s298 & 7 & 2 & 0.87 & 4 & 6 & 0.88 & 3 & 6 & 0.91 & 2 & 6 & 0.94 \\
s344 & 8 & 3 & 0.47 & 4 & 6 & 0.49 & 2 & 6 & 0.53 & 2 & 6 & 0.57 \\
s449 & 8 & 3 & 0.26 & 4 & 6 & 0.28 & 2 & 6 & 0.32 & 2 & 6 & 0.36 \\
s444 & 1 & 0.52 & 5 & 2 & 0.54 & 4 & 6 & 0.58 \\
s510 & 8 & 3 & 0.86 & 3 & 6 & 0.89 & 2 & 6 & 1.00 \\
s526 & 8 & 3 & 0.57 & 2 & 6 & 0.60 & 2 & 6 & 0.64 \\
s526n & 8 & 3 & 0.57 & 2 & 6 & 0.60 & 2 & 6 & 0.64 \\
s713 & 8 & 3 & 0.66 & 2 & 6 & 0.70 & 2 & 6 & 0.74 \\
s820 & 6 & 3 & 0.87 & 2 & 6 & 0.91 & 2 & 6 & 0.95 \\
s832 & 7 & 2 & 0.87 & 5 & 2 & 0.89 & 3 & 6 & 0.93 \\
s953 & 8 & 2 & 0.79 & 6 & 2 & 0.81 & 4 & 6 & 0.85 \\
s1196 & 1 & 0.61 & 6 & 2 & 0.63 & 5 & 2 & 0.66 \\
s1238 & 7 & 1 & 0.65 & 6 & 2 & 0.67 & 5 & 2 & 0.70 \\
s1423 & 7 & 1 & 0.49 & 6 & 2 & 0.52 & 5 & 2 & 0.55 \\
s1488 & 6 & 3 & 0.90 & 5 & 2 & 0.93 & 4 & 6 & 0.97 \\
s1494 & 7 & 2 & 0.89 & 5 & 2 & 0.92 & 4 & 6 & 0.96 \\
s9234 & 1 & 0.44 & 6 & 2 & 0.45 & 5 & 2 & 0.48 \\
s13207 & 8 & 1 & 0.26 & 6 & 2 & 0.28 & 4 & 6 & 0.31 \\
s15850 & 11 & 2 & 0.62 & 6 & 2 & 0.65 & 5 & 2 & 0.68 \\
s35932 & 8 & 2 & 0.57 & 5 & 2 & 0.61 & 4 & 6 & 0.65 \\
\hline
\end{array}
\]

Table 4  Test application time in environment $v_{in}' = 1/2v_{out}'$.
\[
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
\text{circ} & \text{min} & \text{ratio} & T_{pro}(D/v_{out}') & \text{min} & \text{ratio} & T_{pro}(D/v_{out}') & \alpha & \text{ratio} \\
\hline
s298 & 2 & 0.87 & 1.74 & 2 & 0.87 & 1.74 & 0 & 1.00 \\
s344 & 1 & 0.51 & 2.07 & 2 & 0.61 & 2.22 & 0.2 & 0.59 \\
s349 & 1 & 0.47 & 1.87 & 2 & 0.61 & 2.22 & 0.4 & 0.63 \\
s444 & 1 & 0.52 & 2.09 & 2 & 0.63 & 2.63 & 0.27 & 0.64 \\
s510 & 2 & 0.81 & 1.62 & 2 & 0.81 & 1.62 & 0 & 1.00 \\
s526 & 1 & 0.57 & 2.28 & 2 & 0.66 & 3.32 & 0.26 & 0.87 \\
s526n & 1 & 0.57 & 2.28 & 2 & 0.66 & 3.32 & 0.23 & 0.87 \\
s713 & 2 & 0.64 & 1.29 & 2 & 0.65 & 1.29 & 0 & 1.00 \\
s820 & 2 & 0.87 & 1.14 & 2 & 0.87 & 1.14 & 0 & 1.00 \\
s832 & 2 & 0.87 & 1.14 & 2 & 0.87 & 1.14 & 0 & 1.00 \\
s953 & 2 & 0.79 & 1.59 & 2 & 0.79 & 1.59 & 0 & 1.00 \\
s1196 & 1 & 0.60 & 2.43 & 2 & 0.70 & 4.14 & 0.27 & 0.57 \\
s1238 & 1 & 0.61 & 2.46 & 2 & 0.70 & 4.14 & 0.27 & 0.57 \\
s1423 & 1 & 0.49 & 1.99 & 2 & 0.64 & 2.97 & 0.55 & 0.62 \\
s1488 & 3 & 0.90 & 1.20 & 3 & 0.90 & 1.20 & 0 & 1.00 \\
s1494 & 2 & 0.89 & 1.79 & 2 & 0.89 & 1.79 & 0 & 1.00 \\
s9234 & 1 & 0.44 & 1.76 & 2 & 0.62 & 2.57 & 0.56 & 0.71 \\
s13207 & 1 & 0.26 & 1.04 & 2 & 0.50 & 1.09 & 0.63 & 0.98 \\
s15850 & 2 & 0.62 & 1.24 & 2 & 0.62 & 1.24 & 0 & 1.00 \\
s35932 & 2 & 0.57 & 1.14 & 2 & 0.57 & 1.14 & 0 & 1.00 \\
\hline
\text{average} & - & 0.64 & 1.77 & - & 0.71 & 1.39 & - & 0.82 \\
\hline
\end{array}
\]

Table 5  Comparison with Golomb and FDR codings.
\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{circ} & \text{condition} & T_{golomb}(D/v_{out}') & T_{fdr}(D/v_{out}') & \text{condition} & T_{fdr}(D/v_{out}') & T_{pro}(D/v_{out}') \\
\hline
s298 & v_{in} \leq 37/s_{out} & 0.67 & 2.63 & v_{in} \leq 16/33/s_{out} & 0.48 & 10.69 \\
s344 & v_{in} \leq 301/119/s_{out} & 0.30 & 1.19 & v_{in} \leq 20/119/s_{out} & 0.10 & 6.58 \\
s449 & v_{in} \leq 4/8/s_{out} & 0.80 & 1.60 & v_{in} \leq 4/6/s_{out} & 1.00 & 2.01 \\
s444 & v_{in} \leq 4/7/s_{out} & 0.76 & 1.52 & v_{in} \leq 4/6/s_{out} & 1.00 & 2.01 \\
\hline
\end{array}
\]

creases monotonically with the parameter. According to this property, for example, we can find a nearly optimal $\alpha$ by a simple binary search technique.

Finally, we compare the proposed coding with two conventional variable-length codings, Golomb [6] and FDR [7], in the test environment $v_{in}' = 1/2v_{out}'$. As an experiment, we compressed the test sets used in the above experiments by the two codings. Table 5 shows the experimental results: the condition from Inequality (1) (condition), the compression ratio (ratio) and the test application time ($T_{golomb}$, $T_{fdr}$ and $T_{pro}$) for the largest four circuits. The condition from Inequality (1) is derived from the minimum ratio of the length
of a 0-run to that of the corresponding codeword. For example, in the case where the test set for s9234 is compressed by Golomb coding, such a minimum ratio is 87/337 where the length of the 0-run is 337 bits and the length of the corresponding codeword is 87 bits. Note that FDR coding cannot compress the test sets for s15850 and s35932. From this table, we can see that the proposed method can achieve smaller test application time than the others. In particular, for s9234 (s13207), the test application time for the proposed coding is nine (six) times smaller than that for FDR coding even though the compression ratio for the proposed coding is 1.5 (five) times larger than that for FDR coding.

6. Conclusion

In this paper, we discussed the relationship between the test application time and employed variable-length codes in the case where CUTs do not operate much faster than ATEs. Following this discussion, we proposed a new Huffman-based coding method for achieving small test application time by adjusting the compression ratio and the codeword length. Experimental results show that the proposed coding algorithm can reduce the test application time by about 20%, on average, in a given test environment.

Since the proposed method is based on Huffman coding, it can be combined with the test generation methods [9], [10], which utilize the don’t-care bits in test data so as to generate test sets highly compressed by Huffman coding. The effective combination of the proposed method and such test generation methods is one of our future works.

Acknowledgements

This research was supported in part by Japan Society for the Promotion of Science (JSPS) under the Grand-in-Aid for Scientific Research (No. 15300021) and by the Kayamori Foundation of Informational Science Advancement.

References


Hideyuki Ichihara received his M.E. and Ph.D. degrees from Osaka University in 1997, 1999, respectively. He was a research scholar of University of Iowa, U.S.A. from February to July in 1999. Since December 1999, he had been an assistant professor of Hiroshima City University, and he is currently an associate professor of the university. He received IEICE Best Paper Award 2004 and WRTLT (Workshop on RTL and High Level Testing) 2004 Best Paper Award. His research interests are VLSI testing and design for testability. He is a member of the IEEE Computer Society.

Toshihiro Ohara received his Bachelor degree of Information Engineering from Hiroshima City University in 2003. He is currently with the Hitachi Electronics Services Co., Ltd. He received WRTLT (Workshop on RTL and High Level Testing) 2004 Best Paper Award.
Michihiro Shintani received his Bachelor and M.E. degrees of Information Engineering from Hiroshima City University in 2003 and 2005, respectively. He is currently with the Matsushita Electric Industrial Co., Ltd. He received WRTLT (Workshop on RTL and High Level Testing) 2004 Best Paper Award.

Tomoo Inoue is a professor of Graduate School of Information Sciences, Hiroshima City University. His research interests include test generation and high-level synthesis and design for testability and dependability, as well as design and test of reconfigurable devices. He received the B.E., M.E. and Ph.D. degrees from Meiji University, Kawasaki, Japan, in 1988, 1990 and 1997, respectively. From 1990 to 1992, he was with Matsushita Electric Industrial Co., Ltd. From 1993 to 1999, he was an assistant professor of Graduate School of Information Science, Nara Institute of Science and Technology. In 1999, he joined Faculty of Information Sciences, Hiroshima City University as an associate professor. Tomoo Inoue received WRTLT (Workshop on RTL and High Level Testing) 2004 Best Paper Award. He is a member of the IEEE Computer Society and IPSJ.