Regular Paper

An Adaptive Decompressor for Test Application with Variable-Length Coding

Hideyuki Ichihara,† Masakuni Ochi,††,☆ Michihiro Shintani††,☆☆ and Tomoo Inoue†

Test compression/decompression schemes using variable-length coding, e.g., Huffman coding, efficiently reduce the test application time and the size of the storage on an LSI tester. In this paper, we propose a model of an adaptive decompressor for variable-length coding and discuss its property. By using a buffer, the decompressor can operate at any input and output speed without a synchronizing feedback mechanism between an ATE and the decompressor, i.e., the proposed decompressor model can adapt to any test environment. Moreover, we propose a method for reducing the size of the buffer embedded in the decompressor. Since the buffer size depends on the order in which test vectors are input, reordering test vectors can reduce the buffer size. The proposed algorithm is based on fluctuations in buffered data for each test vector. Experimental results show a case in which the ordering algorithm reduced the size of the buffer by 97%.

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 a longer time to transport test sets from the storage device of a VLSI tester (ATE) to a circuit-under-test (CUT). Compression/decompression of test data is an efficient method of overcoming this problem. In this scheme, a given test input set T is compressed into T' using a data compression technique and stored in a VLSI tester storage. While a CUT on a chip is tested, the compressed test input set T' is transported to a decompressor on the chip, and then it is restored to T. The compressed test set can reduce the time for test transportation, not only the size of the test storage device.

Several compression methods for test input sets have been proposed [1)~11)]. These methods are based on data compression techniques such as 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)]. These methods can be divided into two categories based on the lengths of codewords: fixed-length coding [1)~3)] and variable-length coding [4)~11)]. The former methods assign test vectors, partitioned into blocks, to fixed-length codewords; the latter assign them to variable-length codewords. For example, Huffman coding assigns frequently appearing blocks to short codewords. Variable-length coding like Huffman coding is expected to enable higher compression than fixed-length coding.

Even if such variable-length codewords are applied to a decompressor from an ATE at a constant speed (or frequency), the output speed of the decoded test data to the CUT is not constant, unlike when fixed-length codewords are used. This fact complicates the test application of variable-length codewords. To decompress such codewords appropriately, previous researchers [6)~8)] assumed that a decompressor has a synchronizing feedback mechanism, which informs the ATE of whether the decompressor can receive the compressed test data. In Refs. 4), 5), 11), instead of having such a synchronizing feedback mechanism, a sufficient condition must be satisfied concerning the relationship between the speeds of the compressed input data and decompressed output data. In Ref. 5), the ratio of the input and output speeds (clock frequencies) is restricted according to the minimum length of codewords.

In this paper, we propose a model of an adaptive decompressor based on the decompressor proposed in Ref. 5). This model introduces a buffer that permits the decompressor to oper-

† Faculty of Information Sciences, Hiroshima City University
†† Graduate School of Information Sciences, Hiroshima City University
☆ Presently with Koga Software Company
☆☆ Presently with Semiconductor Company, Matsushita Electric Industrial Co., Ltd.

ate at *any* input/output speed *without* a synchronizing feedback mechanism so that it can adapt to various test environments. Moreover, we analyze the relationship between the input/output speed and the buffer size. From the viewpoint of the area overhead of the buffer, the size of the buffer should be small. We therefore determine a lower bound for the buffer size, and then attempt to decrease the buffer size to the lower bound. Since the buffer size depends on the order in which test vectors are input, reordering the test vector can reduce the size of the buffer. Our proposed test vector reordering algorithm can significantly reduce the buffer size, making it close to the introduced lower bound.

This paper is organized as follows. In Section 2, we explain a compression method using Huffman coding [5] and points out a drawback of the decompressor. In Section 3 we propose a model of a decompressor with a buffer and discusses the relationship between the buffer size and the input/output speed. Section 4 introduces an algorithm to reduce the buffer size by reordering test vectors. Section 5 gives some experimental results of the proposed vector reordering algorithm, and Section 6 concludes this paper.

2. Test Compression/Decompression Using Variable-length Coding

In this section, first, we explain the method proposed in Ref. 5) as an example of test compression using variable-length coding. To encode a given test set, a test vector in the test set is partitioned into several n-bit blocks; i.e., each block is an n-bit pattern. A test set that consists of six test vectors partitioned into three 4-bit blocks is shown in **Fig. 1**. The reason for partitioning a test vector into blocks is to keep down the complexity of a decompression circuit and decompression delays. Each block pattern is mapped to a variable-length codeword. The length of a codeword depends on the frequency with which each pattern appears in the test set. The more frequently a pattern occurs, the shorter the length of the codeword for it. **Table 1** shows the frequency, f_i, of the occurrence of each distinct pattern, p_i, in the test set of Fig. 1 and the corresponding Huffman codeword, c_i. The compression ratio, r, is defined by $(D-D_c)/D$, where D and D_c are the sizes of a test set before and after compression, respectively. In this example, $r = (72-32)/72 \simeq 0.56$.

t_1	0000 0000 0000	t_4	0000 0011 0000
t_2	0000 0000 0001	t_5	0001 0000 0000
t_3	0000 0000 0010	t_6	0110 0000 0100

Fig. 1 Test vector set T.

Table 1 Huffman table for T.

i	distinct pattern p_i	freq. f_i	codeword c_i
1	0000	12	1
2	0001	2	00
3	0110	1	0101
4	0010	1	0100
5	0100	1	0111
6	0011	1	0110

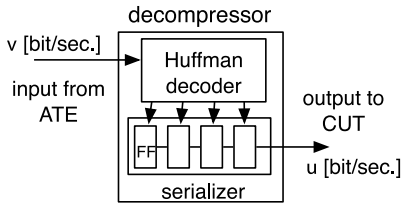

Fig. 2 Decompressor proposed in Ref. 5).

The decompressor proposed in Ref. 5) is shown in **Fig. 2**. It consists of a Huffman decoder and a serializer. A codeword is decoded to the corresponding block pattern after it is entirely fed to the Huffman decoder. The decompressed block pattern is stored in the serializer and scanned out into a (single) internal scan chain of the CUT. The input and the output speeds of the decompressor are v and u, respectively. Here, because we assume both the numbers of primary input and output are single, the operation frequencies of the Huffman decoder and the serializer are also v and u, respectively.

Jas, et al. [5] introduce a condition concerning the input and output speeds to guarantee that Huffman codewords are decoded appropriately, instead of having synchronizing feedback from the CUT to the ATE. The condition is

$$\frac{u}{v} \geq \frac{b}{min\_code\_length} \qquad (1)$$

where min_code_length is the length of the shortest of all the codewords, and b is the size of a partitioned block. Since $b > min\_code\_length$, the output speed u must be greater than the input speed v; i.e., the CUT must operate faster than the ATE.

As an example, **Fig. 3** shows a time chart for a case in which the condition in Inequality (1) is not satisfied. In this case, three codewords, $\langle 1, 0110, 1\rangle$, shown in Table 1, i.e., test vector

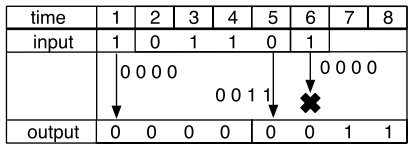

Fig. 3 Time chart of case in which Inequality (1) is not satisfied.

t_4, are applied to the decompressor in order. The output speed is equal to the input speed, i.e., $u/v = 1$. Each codeword is immediately decoded into the corresponding block pattern. For example, at time 1, codeword 1 is applied and decoded immediately into the corresponding block pattern, 0000. The block pattern goes through the output of the decompressor from times 1 through 4. Similarly, at time 6, codeword 1 is decoded into block pattern 0000. This block pattern, however, does not go through the output of the decompressor because the prior block pattern, 0011, corresponding to codeword 0110 is still read out from times 5 through 8. In this case, because the block size b is 4 and the minimum codeword length *min_code_length* is 1, Inequality (1) is not satisfied.

3. Model of Adaptive Decompressor with Buffer

The decompressor of Ref. 5) forces the input and output speeds to satisfy Inequality (1). However, a test environment does not necessarily permit such a condition. We enhance the decompressor of Ref. 5) so that it can adapt to any test environment.

A model of the enhanced decompressor is shown in **Fig. 4**. The difference between the decompressor of Ref. 5) and this model is that the serializer of Fig. 2 is replaced with a buffer in Fig. 4. The size of the buffer embedded in the decompressor is denoted by B bits. Note that the serializer of the conventional model of Fig. 2 can be considered one block-sized (4-bit) buffer.

The proposed decompressor can appropriately decompress test sequences even when Inequality (1) is not satisfied. The time chart of **Fig. 5** depicts a case where the test sequence can be appropriately decompressed when the parameters u, v, b and *min_block_length* are the same as in the unsuccessful example of Fig. 3. Block pattern 0000 corresponding to codeword 1 applied at time 6 is buffered through time 9, and then read out.

In this model, next, we consider the relationship between the input and output speeds, v and u, and the buffer size, B. First of all, we define the difference (increase or decrease), say, d_i, in buffered test data after applying a codeword, c_i, to the decompressor. It is the difference of test data outgoing from the buffer and that incoming to the buffer and is given by the following equation:

$$d_i = b - \frac{u}{v} \cdot l_i, \tag{2}$$

where b is the block size, u/v is the ratio of the output speed to the input speed, and l_i is the length of codeword c_i. The first term on the righthand side, b, is the amount of test data coming into the buffer, and the second term, $u/v \cdot l_i$, is the amount of test data going out during a period of applying the codeword c_i. **Table 2** shows d_i for each distinct block pattern in Table 1 when $u/v = 1$ and $u/v = 2$. For example, the d_1 for the first block pattern, 0000, is 3 when $u/v = 1$. This means the 4 bits, i.e., $b = 4$, of test data is incoming while 1 bit, i.e., $u \cdot l_i/v = 1$, of test data is outgoing.

The buffer size, B, must be equal to or larger than the maximum amount of buffered test data while applying all test vectors. Suppose we have a block sequence p_1, p_2, p_3, ..., p_M such that the blocks are obtained by partitioning each test vector in a given test set, where M is the number of blocks in the test set. When M block

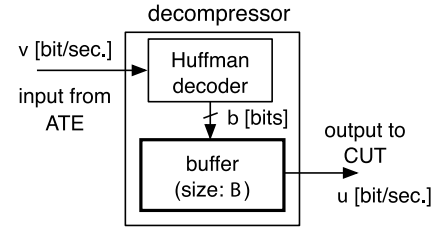

Fig. 4 Decompressor with buffer.

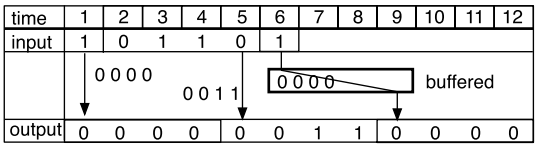

Fig. 5 Time chart with buffer. (One block is buffered.)

Table 2 d_i for distinct patterns in Table 1.

	$u/v = 1$	$u/v = 2$
d_1	3	2
d_2	2	0
d_3, d_4, d_5, d_6	0	-4

patterns are applied to the decompressor in the order of $\langle p_1, p_2, p_3, \ldots, p_M \rangle$, the buffer size B can be expressed as

$$B \geq \max_{1 \leq j \leq M}(s_j), \tag{3}$$

where

$$s_j = \begin{cases} s_{j-1} + d_{t(p_j)} & s_{j-1} + d_{t(p_j)} > 0, \\ & j \geq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Here, $t(p_i)$ is the identification number of the distinct pattern of block p_i. For example, let the distinct pattern of p_1 be 0000; $t(p_1)$ is 1 from Table 1, and therefore, $d_{t(p_1)} = d_1$. The term s_j is the amount of buffered test data after p_j is applied. It is the accumulation of the differences of blocks p_1, p_1, p_2, $\ldots$, p_j, i.e., $d_{t(p_1)}+d_{t(p_2)}+\cdots+d_{t(p_j)}$. Note that the amount of buffered test data cannot be negative. Accordingly, the buffer size B is the maximum amount of buffered test data while applying all block patterns p_1, p_2, p_3, $\ldots$, p_M, as shown by Inequality (3).

Here, consider a case where the buffer size B is zero in the proposed model. Clearly, one such case is that $d_i \leq 0$ for all distinct block patterns p_i according to Inequality (3) and Eq. (4). From Eq. (2), this case is represented by

$$\frac{u}{v} \geq \frac{b}{l_i}. \tag{5}$$

From the definition of *min_code_word*, the inequality $b/min\_code\_word \geq b/l_i$ is satisfied. Therefore, this case in which the buffer size is zero has the condition

$$\frac{u}{v} \geq \frac{b}{min\_code\_word}. \tag{6}$$

This inequality coincides with Inequality (1). Namely, the proposed model of the decompressor includes the model proposed in Ref. 5).

4. Test Vector Reordering

An embedded buffer should be small. In this section, we propose a method of minimizing the buffer size using test vector reordering.

4.1 Order of Applied Blocks and Buffer Size

As shown in Inequality (3) and Eq. (4), the buffer size B depends on the order of blocks shifted into the decompressor. A time chart illustrating a case where codewords $\langle 0110, 1, 1 \rangle$ are applied into the decoder in order is shown in **Fig. 6**. This order of codewords is the reordered one of Fig. 5, $\langle 1, 0110, 1 \rangle$. In this chart,

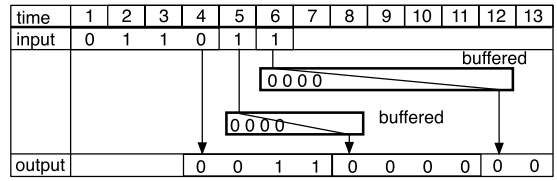

Fig. 6 Time chart when block order is $\langle 0110, 1, 1 \rangle$ (Two blocks are buffered).

two block patterns corresponding to codeword 1's applied at times 5 and 6 are buffered because the first block pattern, 0011, corresponding to codeword 0110 is still using the output of the decompressor from times 4 through time 7. When we compare Fig. 5 with Fig. 6, the buffer required in Fig. 6 is larger. This shows that an appropriate order of applied codewords exists to minimize the buffer size.

How can block reordering minimize the buffer size? As a possible answer, here, we can give a lower bound of the buffer size, which is a necessary amount of buffered test data, however you reorder blocks.

Theorem(Lower bound of buffer size): Let N be the number of distinct block patterns, d_i be the difference of distinct block pattern p_i, and f_i be the frequency ☆ of appearance of distinct block pattern p_i in a given test set. A lower bound of the buffer size, LB, is given by

$$LB = \sum_{i=1}^{N} d_i \cdot f_i. \tag{7}$$

Proof: Again, consider a sequence of blocks, p_1, p_2, p_3, $\ldots$, p_M, where M is the number of blocks included in the given test set, and the buffer size is derived from Inequality (3) and Eq. (4). Instead of Eq. (4), here, we introduce an equation, $s'_j = s'_{j-1} + d_{t(p_j)}$, which is obtained by omitting the second case of Eq. (4). Since s'_j is an arithmetic series whose common difference is $d_{t(p_i)}$, s'_j can be simply represented by

$$s'_j = \sum_{i=1}^{j} d_{t(p_i)}. \tag{8}$$

Clearly, $s_j \geq s'_j$. Therefore, from Inequality (3) and the definition of the function $\max(\cdot)$, we have

$$B \geq \max_{1 \leq j \leq M}(s_j) \geq \max_{1 \leq j \leq M}(s'_j) \geq s'_M. \tag{9}$$

Note that from (8), s'_M is the sum of the d_i for *all* M blocks, so it is uniquely determined

☆ This corresponds to the frequency f_i in Table 1.

irrespective of the order of blocks, unlike s'_1, s'_2, $\ldots$, s'_{M-1}.

From $M = f_1 + f_2 + \cdots + f_N$,

$$\begin{aligned} s'_M &= \sum_{i=1}^{M} d_{t(p_i)} \\ &= \sum_{i=1}^{f_1} d_1 + \sum_{i=1}^{f_2} d_2 + \cdots + \sum_{i=1}^{f_N} d_N \\ &= \sum_{i=1}^{N} d_i \cdot f_i. \end{aligned} \tag{10}$$

Consequently, from Inequality (9) and Eq. (10),

$$B \geq \max_{1 \leq j \leq M}(s_j) \geq \sum_{i=1}^{N} d_i \cdot f_i = LB.$$

(Q.E.D.)

The lower bound LB can also be comprehended intuitively as follows. Block patterns (or codewords) can be classified into two groups: block patterns p_i whose differences $d_{t(p_i)}$ are positive and negative. As expressed in Eq. (4), the former blocks increase the amount of buffered data, and the latter blocks decrease it. Therefore, the sum of the d_i for all blocks, i.e., LB, is the difference resulting from adding the amount that the buffered test data increases and the amount that it decreases. Since the buffer size must be greater than this gap, LB is a lower bound.

The lower bound LB of the buffer size is useful for the test vector reordering algorithm proposed in the next section.

4.2 Reordering Strategy

In Section 4.1 we considered reordering *blocks (or codewords)*. However, whether you can reorder blocks arbitrarily depends on the design for testing of a given CUT. Here, we assume full scan design (or combinational circuits). In full-scan design, the order of the test vectors in a given test set does not affect the desired fault efficiency of the test set, and thus, reordering test vectors is acceptable, though reordering blocks is not. Hereafter, we consider a method for reordering test vectors to minimize the buffer size.

Transitions in the amount of buffered test data for two orders of test vectors are shown in **Fig. 7**. The solid line corresponds to the case in which the sequence of test vectors applied is $\langle t_1, t_2, t_3, t_4, t_5, t_6 \rangle$. In this case, the maximum buffered test data is 14 bits, and therefore the buffer size also must be larger than or equal to 14 bits. The broken line corresponds to the case

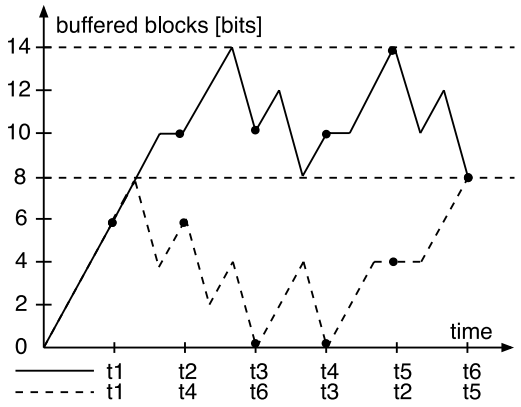

Fig. 7 Transitions in amount of buffered data in two test vector orders.

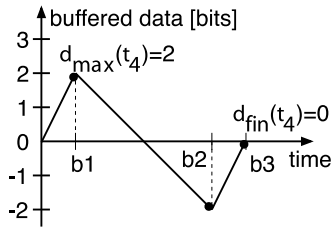

Fig. 8 Fluctuation for vector t_4.

of the order $\langle t_1, t_4, t_6, t_3, t_2, t_5 \rangle$. The buffer size required by this test vector order is 8 bits, which is smaller than that required by the former order. Consequently, test vector reordering, in a similar way as block pattern (codeword) reordering, effectively minimizes the buffer size.

Our proposed test vector reordering algorithm is straightforward. The algorithm picks one test vector, t, out of a given test set, V, according to our selection rules described below and appends t at the end of the sequence, S, of the selected test vectors. S is initially empty. The test vector selection is repeated until V becomes empty. The sequence S is returned as a resultant reordered test sequence.

The selection rules for test vectors are based on the *fluctuation* in buffered test data for a test vector, which is represented by concatenating the differences d_i for codewords composing the test vector. Consider test vector t_4 in Fig. 1. It is composed of two distinct block patterns, 0011 and 0000, whose differences of buffered test data are d_1 and d_6, respectively. When $u/v = 2$, the fluctuation for test vector t_4 is shown in **Fig. 8**. We focus on two points in the fluctuation. One is $d_{max}(t)$, which denotes the peak amount of buffered test data raised by a test vector t. The other is $d_{fin}(t)$, which

is the final amount of buffered test data. Note that the $d_{max}(t)$ indicates how the buffer size increases after applying test vector t, while the $d_{fin}(t)$ indicates how the amount of buffered test data varies after applying test vector t.

The rules for test vector selection are as follows. The first rule is that we select a test vector that does not increase the buffer size, if possible. This is a basic rule to greedily keep the buffer size small. The second rule is that we select a test vector whose $d_{max}(t)$ is maximum if multiple test vectors can be selected according to the first rule. Without increasing the buffer size, this second rule can select as many test vectors as possible with large d_{max}, i.e., undesirable test vectors that tend to increase the buffer size. Furthermore, as the third rule, if more than one test vector is selected using these two rules, we select the test vector whose d_{fin} is the smallest of all the test vectors. This rule can enhance the possibility of selecting as many test vectors as possible according to the first rule in the subsequent vector selection.

If no test vector is selected according to the first rule, i.e., any test vector increases the buffer size, we attempt to increase the number of the test vectors selected using the first rule in the subsequent selection process. For this reason, we select a test vector such that $d_{fin} < 0$. This selection always decreases the amount of the buffered test data, even though the buffer size increases, thus increasing the possibility of increasing the number of test vectors selected by the first rule. If more than one test vectors satisfies $d_{fin} < 0$, we select the test vector whose d_{max} is the smallest of these test vectors to keep the buffer size as small as possible.

If no test vector is selected in the above cases, i.e., any remaining test vector increases both the buffer size and the amount of buffered test data, we dare to select the test vector whose $d_{max}(t)$ is a maximum. Since a test vector with a large $d_{max}(t)$ tends to make the buffer size large, we believe that such a test vector should be selected at an early phase of the selection process, when the buffer size is still small. This is an application of the second rule[☆].

The initial buffer is set to the size required to increase the chance of selecting many test vectors using the first rule: select a test vector with a large d_{max} without increasing the buffer size. Therefore, the initial buffer size is set to the lower bound obtained using Eq. (7). If the lower bound is smaller than the maximum d_{max}, the initial buffer size is set to the maximum d_{max}.

4.3 Test Vector Reordering Algorithm

Based on the above strategy, we propose a test vector reordering algorithm. Note that steps (3-1) and (3-2) in the following algorithm work for cases in which a test vector is selected using the three rules mentioned above. Steps (3-3) and (3-4) in the algorithm are for cases in which no test vector is selected according to the above rules.

[Test vector reordering for minimal buffer size]

(1) Let V be a set of test vectors. Let S be the resultant ordered test sequence. Initially, $S = \phi$.
(2) Set the buffer size B to $\max(\max_{t \in V} d_{max}(t), LB)$ and the initial amount of buffered test data B_c to 0.
(3) Repeat steps (3-1) to (3-5) until $V = \phi$.
  (3-1) Set $V^0 = \{t | B_c + d_{max}(t) \leq B, t \in V\}$.
  (3-2) If $V^0 \neq \phi$, then select $t \in V^0$ such that $d_{max}(t)$ is maximized. If there are multiple such test vectors, select t from them such that $d_{fin}(t)$ is minimum. Go to step (3-5).
  (3-3) Set $V^1 = \{t | d_{fin}(t) < 0, t \in V\}$.
  (3-4) If $V^1 \neq \phi$, then select $t \in V^1$ such that $d_{max}(t)$ is minimized. Otherwise, select $t \in V$ such that $d_{max}(t)$ is maximized.
  (3-5) Append t to the end of test sequence S. Set $V = V - \{t\}$, $B_c = \max(B_c + d_{fin}(t), 0)$ and $B = \max(B, B_c + d_{max}(t))$. Return to step (3).
(4) Return S.

We illustrate an example of the proposed algorithm using Fig. 1 and Table 1. Assume that the ratio of input and output speeds u/v is 2. **Table 3** shows d_{max} and d_{fin} for the test patterns in Fig. 1. The resultant ordered test sequence and the transition in the amount of buffered data are depicted in **Fig. 9**. The resultant buffer size is 8, which is equal to the lower bound given by Eq. (7).

[☆] An alternative selection is that a test vector is selected with a *minimum* $d_{max}(t)$, according to the first rule. Since qualitatively discussing the effectiveness of the two selections may be impossible, we experimented on these selections using some benchmark circuits. From the results, we could not observe a difference in their effectiveness. Therefore, we adopted one of them.

Table 3 d_{max} and d_{fin} for test vectors in Fig. 1.

	d_{max}	d_{fin}		d_{max}	d_{fin}
t_1	6	6	t_4	2	0
t_2	4	4	t_5	4	4
t_3	4	0	t_6	0	−6

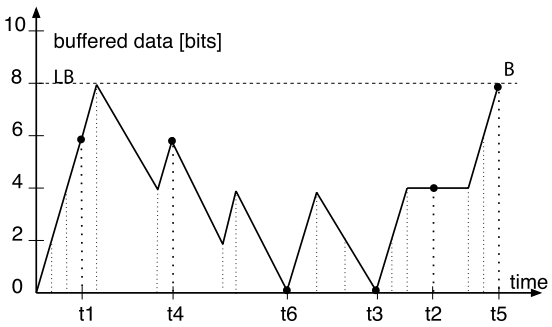

Fig. 9 Transition in amount of buffered data for test reordered vectors.

After the initialization (in step (1) of the proposed algorithm), the initial value of the buffer size B is set to 8 because the lower bound given by Eq. (7) is 8, which is larger than the maximum d_{max}, i.e., $\max_t d_{max}(t) = d_{max}(t_1) = 6$ (step (2)). As long as the current number of buffered test block patterns is not over the buffer size, 8, the test vector whose d_{max} is maximum is selected (steps (3-1) and (3-2)). This selection is based on the three rules. Test vector t_1 is selected in this case. Since $d_{fin}(t1)$ is 6, the buffered amount B_c becomes 6. The buffer size B is still 8 (step (3-5)). Next, t_4 is selected in the same manner (steps (3-1) and (3-2)). Then, t_6 is selected (step (3-2)) because the other test vectors are not selectable according to the first rule, i.e., $B_c + d_{max}(t) > B$. Next, t_3 is selected (the latter part of step (3-2)) because $d_{fin}(t_3)$ is the smallest of the d_{fin}s for the remaining test vectors. The order of the remaining test vectors is arbitrary; t_2 and t_5 are selected in order (step (3-4)). This is because these two vectors have the same properties, i.e., $d_{max}(t_2) = d_{max}(t_5)$ and $d_{fin}(t_2) = d_{fin}(t_5)$.

5. Experimental Results

We implemented the proposed test vector reordering algorithm in Perl and applied it to test sets for ISCAS'89 benchmark circuits. The target benchmark circuits and test sets are shown in **Table 4**. The test sets were generated using a test generation method proposed in Ref. 10), which is intended to produce test sets that are highly compressible using Huffman coding. Since the test generator [10] uses dynamic compaction and don't-care specification for Huffman coding, it can generate small and highly compressible test sets. In the experiments, each test vector in the given test sets is partitioned into 4-bit blocks and encoded using Huffman coding. The last three columns of Table 4 show the compression ratio, defined in Section 2, and the lengths of the shortest and the longest codewords. The d_{max} and d_{fin} of each test vector were calculated as a pre-process of the proposed method. The calculation time is very small and can be neglected.

Table 4 Benchmark circuits.

circ.	#PIs	#test vectors	comp. ratio: r	short	long
s1238	32	153	0.318	1	7
s9234	248	148	0.524	1	6
s13207	700	269	0.703	1	9
s15850	612	127	0.609	1	9
s35932	1,764	16	0.501	1	7
s38417	1,664	100	0.536	1	8
s38584	1,464	131	0.587	1	7

Note that Inequality (1) is satisfied when the speed ratio is greater than of equal to 4 because the block size is four and the length of the shortest codeword is one, as shown in Table 4. Namely, when $u/v \geq 4$, the decompressor requires no buffer.

The buffer size for six ratios of input/output speeds is shown in **Table 5**. The *orig.* row denotes the buffer size (# of 4-bit registers) before reordering, and the *prop.* row denotes that after reordering. The *ratio* row shows the reduction ratio, $((orig.) - (prop.))/(orig.)$. From this table, you can see that the proposed test vector reordering can reduce the buffer size for all circuits. For s38584, the reduction ratio reaches 97% when the speed ratio is 3.0. When the speed ratio is 1.0, i.e., the input speed is equal to the output speed, the reduction ratio is almost 0. This is because the positive d_is are few compared to the negative ones, and thus, test vector reordering is ineffective.

The *LB* row of Table 5 shows the lower bounds introduced in Section 4.1. Note that the lower bounds were calculated using Eq. (7), and they can be negative. By comparing the lower bounds and the buffer sizes after the test vectors are reordered, when the lower bound is positive, you can see that the obtained buffer size is close to the lower bound in almost all cases. When the lower bound is negative, on the other hand, the buffer size is not close to the lower

Table 5 Buffer size reduction.

circ.		input/output speed ratio: u/v					
		1.0	1.5	2.0	2.5	3.0	3.5
s1238	orig.	390	32	6	4	3	2
	prop.	390	15	4	4	3	2
	ratio	**0.00**	**0.53**	**0.33**	**0.00**	**0.00**	**0.00**
	LB	388.25	−29.62	−447.50	−865.38	−1,283.25	−1,701.12
s9234	orig.	4,802	2,747	1,232	375	87	18
	prop.	4,802	2,623	461	33	14	8
	ratio	**0.00**	**0.05**	**0.63**	**0.91**	**0.84**	**0.56**
	LB	4,801.75	2,614.62	427.50	−1,759.62	−3,946.75	−6,133.88
s13207	orig.	33,019	26,333	19,975	13,946	8,325	2,976
	prop.	32,930	25,863	18,795	11,723	4,641	2,322
	ratio	**0.00**	**0.02**	**0.06**	**0.16**	**0.44**	**0.22**
	LB	32,929.75	25,857.12	18,784.50	11,711.88	4,639.25	−2,433.38
s15850	orig.	11,793	8,418	5,651	3,723	1,928	465
	prop.	11,752	7,907	4,066	411	608	20
	ratio	**0.00**	**0.06**	**0.28**	**0.89**	**0.68**	**0.96**
	LB	11,748.00	7,906.50	4,065.00	223.50	−3,618.00	−7,459.50
s35932	orig.	3,646	1,941	669	346	224	111
	prop.	3,646	1,949	327	236	117	57
	ratio	**0.00**	**−0.00**	**0.51**	**0.32**	**0.48**	**0.49**
	LB	3,644.75	1,939.12	233.50	−1,472.12	−3,177.75	−4,883.38
s38417	orig.	22,121	13,523	6,450	2,411	382	73
	prop.	22,108	12,361	2,637	154	98	38
	ratio	**0.00**	**0.09**	**0.59**	**0.94**	**0.74**	**0.48**
	LB	22,106.25	12,359.38	2,612.50	−7,134.38	−16,881.25	−26,628.12
s38584	orig.	28,268	19,909	13,091	7,343	3,222	701
	prop.	28,028	18,062	8,096	4,003	103	57
	ratio	**0.01**	**0.09**	**0.38**	**0.45**	**0.97**	**0.92**
	LB	28,018.50	18,054.75	8,091.00	−1,872.75	−11,836.50	−21,800.25

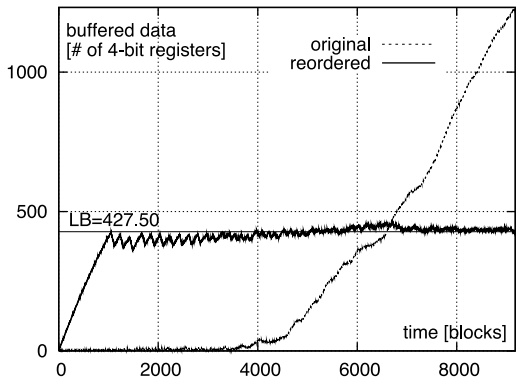

Fig. 10 Fluctuation in buffered data before and after test vector reordering for s9234.

bound (or zero) even though the buffer size is considerably reduced. Therefore, the proposed reordering algorithm can be improved to reduce the buffer size in cases where the lower bound is negative.

The transition in the amount of the buffered test data for s9234 when the speed ratio was 2.0 is illustrated in **Fig. 10**. Without test vector reordering, the amount of buffered data increases gradually and reaches 1,232 registers (4,928 bits). With the proposed test vector reordering, first, the amount of buffered data rises sharply to the lower bound, 427.50 registers (1,710 bits), and levels off. This transition follows our reordering strategy. The resultant buffer size is 461 registers (1,844 bits), which is close to the lower bound. This tendency of the transition in the amount of buffered test data for s9234 was also seen in other circuits.

6. Conclusion

We proposed an adaptive model of a decompressor with a buffer, which was based on the decompressor proposed in Ref. 5). The buffer permits the decompressor to operate at *any* input/output speed *without* a synchronizing mechanism. Moreover, we proposed a method of minimizing the buffer size. Since the buffer size depends on the order in which test vectors are input, a proper order of test vectors can reduce the size of the buffer. The proposed test vector reordering algorithm in intended to minimize the buffer size and make it close to the introduced lower bound.

Our experimental results show that the proposed test vector reordering algorithm can reduce the size of the buffer by more than 90% in some cases. The reduced buffer size is close to the lower bound when the lower bound is pos-

itive. Our future work includes proposing an alternative reordering algorithm to reduce the buffer size when the proposed lower bound is negative.

Acknowledgments This work was supported by JSPS.KAKENHI 15300021.

References

1) Jas, A. and Touba, N.A.: Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs, *Proc. ITC*, pp.458–464 (1998).
2) Li, L. and Chakrabarty, K.: Test Data Compression Using Dictionaries with Fixed-Length Indices, *Proc. VTS*, pp.219–224 (2003).
3) Rao, W., Bayrajtariglu, I. and Orailoglu, A.: Test application time and volume compression through seed overlapping, *Proc. DAC*, pp.732–737 (2003).
4) Iyengar, V., Chakrabarty, K. and Murray, B.T.: Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets, *Proc. VTS*, pp.418–423 (1998).
5) Jas, A., Dastidar, J.G. and Touba, N.A.: Scan Vector Compression/Decompression Using Statistical Coding, *Proc. VTS*, pp.114–120 (1999).
6) Chandra, A. and Chakrabarty, K.: Test Data Compression for System-on-a-Chip Using Golomb Codes, *Proc. VTS*, pp.113–120 (2000).
7) Chandra, A. and Chakrabarty, K.: Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression, *Proc. VTS*, pp.42–47 (2001).
8) Gonciari, P.T., Al-Hashimi, B.M. and Nicolici, N.: Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression, *Proc. DATE*, pp.604–611 (2002).
9) Ichihara, H., Ogawa, A., Inoue, T. and Tamura, A.: Dynamic Test Compression Using Statistical Coding, *Proc. ATS*, pp.143–148 (2001).
10) Ichihara, H. and Inoue, T.: Generating Small Test Sets for Test Compression/Decompression Using Statistical Coding, *Proc. DELTA*, pp.396–400 (2002).
11) Volkerink, E.H., Khoche, A. and Mitra, S.: Packet-based Input Test Data Compression Techniques, *Proc. ITC*, pp.154–163 (2002).

(Received October 24, 2005)
(Accepted April 4, 2006)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.2, pp.348–356.)

Hideyuki Ichihara received his M.E. and Ph.D. degrees from Osaka University in 1997 and 1999, respectively. He was a research scholar of the University of Iowa, U.S.A. from February to July 1999. In December 1999, he became an assistant professor of Hiroshima City University, and he is currently an associate professor of the university. He received the IEICE Best Paper Award 2004 and Workshop on RTL and High Level Testing 2004 Best Paper Award. He is a member of the IEEE and IEICE.

Masakuni Ochi received his Bachelor of Information Engineering degree from Hiroshima City University in 2004. He is currently with Koga Software Company.

Michihiro Shintani received his bachelor's and master's degrees in information engineering from Hiroshima City University in 2003 and 2005, respectively. He is currently with Matsushita Electric Industrial Co., Ltd. He received the Workshop on RTL and High Level Testing 2004 Best Paper Award.

Tomoo Inoue is a professor of the Faculty of Information Sciences, Hiroshima City University. His research interests include test generation, high-level synthesis and design for testability and dependability, and design and test of reconfigurable devices. He received B.E., M.E. and Ph.D. degrees from Meiji University, Kawasaki, Japan in 1998, 1990 and 1997. From 1990 to 1992, he was with Matsushita Electric Industrial Co., Ltd. From 1993 to 1999, he was an assistant professor of the Graduate School of Information Science, Nara Institute of Science and Technology. In 1999, he joined Hiroshima City University as an associate professor. Tomoo Inoue received the Workshop on RTL and High Level Testing 2004 Best Paper Award. He is a member of the IEEE, IEICE and IPSJ.