PAPER Special Section on Test and Verification of VLSIs

An Architecture of Embedded Decompressor with Reconfigurability for Test Compression

Hideyuki ICHIHARA[†a)], *Member*, Tomoyuki SAIKI[†*], *Nonmember*, *and* Tomoo INOUE[†b)], *Member*

SUMMARY Test compression / decompression scheme for reducing the test application time and memory requirement of an LSI tester has been proposed. In the scheme, the employed coding algorithms are tailored to a given test data, so that the tailored coding algorithm can highly compress the test data. However, these methods have some drawbacks, e.g., the coding algorithm is ineffective in extra test data except for the given test data. In this paper, we introduce an embedded decompressor that is reconfigurable according to coding algorithms and given test data. Its reconfigurability can overcome the drawbacks of conventional decompressors with keeping high compression ratio. Moreover, we propose an architecture of reconfigurable decompressors for four variable-length codings. In the proposed architecture, the common functions for four codings are implemented as fixed (or non-reconfigurable) components so as to reduce the configuration data, which is stored on an ATE and sent to a CUT. Experimental results show that (1) the configuration data size becomes reasonably small by reducing the configuration part of the decompressor, (2) the reconfigurable decompressor is effective for SoC testing in respect of the test data size, and (3) it can achieve an optimal compression of test data by Huffman coding.

key words: *test compression, ATE, reconfigurability, variable-length coding, test application*

1. Introduction

As the size and complexity of LSI circuits increase, the size of test data for the circuits also increases. The increase in the test data size requires a larger storage and the longer time for LSI testing. Compression/decompression scheme of test data has been proposed to overcome this problem [1]–[6]. The scheme is shown in Fig. 1. In this scheme, a given test input data T is compressed into T' by a data compression algorithm and stored in an LSI tester (ATE) storage. While a circuit-under-test (CUT) on a chip is tested, the compressed test input data T' is transported to a decompressor on the chip, and then it is restored to T, and given to the CUT. Note that, like the previous test input compression methods [1]–[6], we do not treat test responses in this paper. We assume that test responses from the CUT is also compressed by response compression techniques, e.g., MISR, [7]–[9].

Test input compression methods [1]–[6] are based on some data compression algorithms, e.g., Huffman coding [1], [2], Run-length coding [3], Golomb coding [4], FDR coding [5], VIHC coding [6], and so on.

Manuscript received April 9, 2007.
Manuscript revised August 10, 2007.
[†]The authors are with the Graduate School of Information Sciences, Hiroshima City University, Hiroshima-shi, 731–3194 Japan.
[*]Presently, with the Canon Inc.
a) E-mail: ichihara@hiroshima-cu.ac.jp
b) E-mail: tomoo@hiroshima-cu.ac.jp
DOI: 10.1093/ietisy/e91–d.3.713

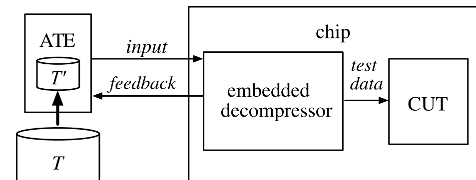


Fig. 1 Overview of test data compression / decompression.

In these test compression methods, the employed coding algorithms are tailored for a given test data. For example, in [1], [2] and [6], Huffman codewords used for test compression are constructed according to the frequency of block-patterns in a given test data. In [4], the authors suggested suitably selecting the value of a parameter (group size) of Golomb coding according to the distribution of the length of successive 0s in a test data.

Although this tailoring of the coding algorithm can achieve high compression of a given test data, it involves two drawbacks. One is that the coding algorithm and the corresponding decompressor must be designed after test generation, which is performed downstream of the design flow. This drawback makes the design flow lose its flexibility, so that it may lengthen the period of designing LSIs. For example, let us consider that we design a decompressor after test generation and synthesize it with the CUT. In this phase, if we cannot satisfy some of design constraints, we have to return to an upstream design phase before synthesizing the CUT.

The other drawback is that, since a decompressor is designed for a particular test data, it may not be adapted to an extra test data but the particular test data, i.e., the extra test data cannot be applied to the CUT, or it may be hardly compressed even if it can be applied.

In this paper, to overcome these drawbacks, we propose a reconfigurable embedded decompressor. The reconfigurable embedded decompressor can switch its decoding algorithm according to a test data applied to the decompressor in the test application phase. Hence, in design phase of a chip, the embedded decompressor can be designed independent of the CUT and their test data, that is, the first drawback is overcome. Moreover, since the proposed decompressor can reconfigure itself so as to achieve high compression of a given test data, the latter drawback can be also removed. In addition, the reconfigurable embedded decompressor has another advantage for SoC testing. In conventional test compression for SoC testing, each core has its own decompres-

Copyright © 2008 The Institute of Electronics, Information and Communication Engineers

sor. However, using a reconfigurable embedded decompressor, we can decompress all the test data for all cores by the decompressor only.

Note that the proposed scheme can be combined with don't-care bit specification techniques so as to generate test sets highly compressed by these codings. For example, the authors of [5] show an effective don't-care specification for Run-length coding and the authors of [10], [11] have proposed test generation methods for Huffman coding.

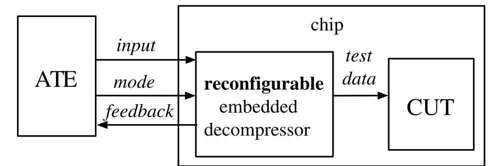
When the decompressor is reconfigured, it is necessary that a configuration data, which determines the decoding algorithm of the decompressor, is sent from an ATE to the decompressor. Hence, a test data includes the configuration data in addition to an compressed test input data. To bring out the merit of the reconfigurable decompressor, the reduction in the configuration data is important. Therefore, we attempt to decrease the configuration data by minimizing the reconfigured part of the reconfigurable decompressor. We show an architecture of a reconfigurable embedded decompressor for four variable-length codings (Huffman, Golomb, FDR and VIHC), and then concretely discuss a common reconfigured part, which is needed by all the four codings, of the decompressor.

The remaining of this paper is organized as follows. Section 2 considers a scheme of a reconfigurable embedded decompressor. Section 3 introduces two major variable-length codings Huffman and Golomb codings. Section 4 proposes an architecture of a reconfigurable embedded decompressor for the four variable-length codings, and discusses the division between a reconfigurable part and a fixed (non-reconfigurable) part of the decompressor for the reduction in the configuration data to be sent from an ATE. Section 5 reports experimental results using the proposed reconfigurable decompressor. Section 6 concludes this paper.

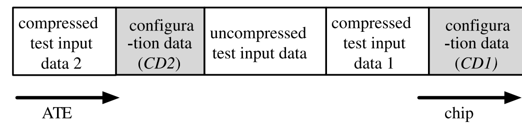
2. Scheme of Reconfigurable Embedded Decompressor

In this section, we show the scheme of test data decompression using a reconfigurable embedded decompressor. The basic functions required for an embedded decompressor are to receive a test data compressed from an ATE and to decompress it and to supply the decompressed test data to CUTs. The reconfigurable embedded decompressor can change its internal operation by receiving configuration data from the ATE prior to the test application phase. Therefore, test data compressed by various coding algorithms can be decompressed by the decompressor.

The decompressor has three operation modes: *configuration*, *decompression* and *direct*. In mode *configuration*, as a preprocess of testing, a decompressor receives configuration data from the ATE, and reconfigures itself. Modes *decompression* and *direct* are for testing. In mode *decompression*, the decompressor receives a compressed test data and decompresses it according to the decoding function implemented by the configuration, while, in mode *direct*, it receives a test data and outputs it directly, i.e., the test data passed through the decompressor. This mode is for applying an uncompressed test data to CUTs.

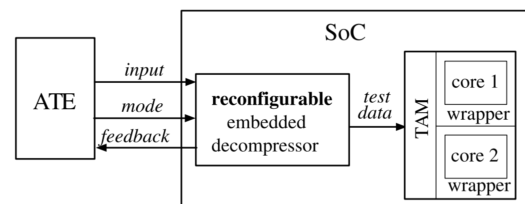


(a) Block diagram.

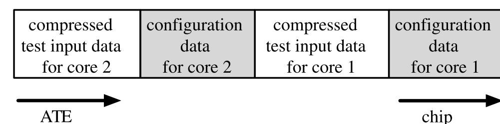


(b) Sequence of test data.

Fig. 2 Reconfigurable embedded decompressor.



(a) Block diagram.



(b) Sequence of test data.

Fig. 3 Reconfigurable embedded decompressor for SoC.

The block diagram of the reconfigurable embedded decompressor is shown in Fig. 2 (a). Signal *input* is an input of a decompressor for receiving test data. Signal *mode* directs to the decompressor which mode is selected. Signal *feedback* is a feedback signal from a decompressor to an ATE, which indicates whether a decompressor can receive data. When the feedback signal is enable, the decompressor is ready to receive test data and the mode selection codes. Note that the ATE does not use this signal for retrieving test responses. Signal *test data* is for the decompressed test data which is supplied to the CUT. An input sequence of test data to the decompressor is shown in Fig. 2 (b). A decompressor is first reconfigured to a decompressor for a desirable coding algorithm (e.g. Huffman coding) by means of receiving a configuration data ($CD1$), and then the decompressor receives the compressed test data corresponding to $CD1$ and decompresses it. Next, since the decompressor receives an uncompressed test data, it outputs without decompression as the direct mode. Finally, it is reconfigured to a decompressor for another coding algorithm (e.g. Golomb coding) by means of receiving a configuration data ($CD2$), and receives a test data decompressed by the coding algorithm.

Here, let us show the impact of our reconfigurable decompressors in an application to SoC testing. Fig. 3 shows an example of using the model of reconfigurable embedded decompressor for SoC testing. In conventional test data

compression/decompression for SoC testing, in order to obtain a high compression ratio, an optimal coding algorithm must be applied to the test data for each core, and accordingly its own decompressor must be designed. In contrast, our reconfigurable decompressor can decode the test data compressed optimally for each core, as shown in Fig. 3 (a). Figure 3 (b) shows an input sequence for testing two cores 1 and 2 serially. Note that a reconfigurable embedded decompressor can switch the coding algorithms used for decompression during test application as shown in Fig. 3 (b). In this way, the reconfigurable embedded decompressor can be shared with several cores, and thus the area overhead of the proposed decompressor will be smaller than the area overhead required for several decompressors in the conventional test data compression/decompression scheme, even though the proposed decompressor possesses a reconfigurable part. Detailed discussion on the area overhead of the proposed decompressor will appear in in Sect. 4.

3. Test Compression / Decompression Using Variable-Length Codes

This section describes test data compression / decompression methods using variable-length codings. In general, variable-length codings can obtain high compression ratio. Here we introduce two methods of test data compression and decompression by Huffman coding [1], [2], and Golomb coding [4], which are variable-length codings. Note that each coding has distinct characteristics. Huffman coding can achieve high compression of test data whose entropy is low, i.e., some distinct block patterns appear frequently in the test data, while Golomb coding works effectively for test data including long sequences (or runs) of zero or one.

Table 1 shows an example of two codings. The given test data is 32 bits in size. The test data is encoded into eight codewords by Huffman coding, while it is encoded into six codewords by Golomb coding.

• **Huffman Coding** [1], [2]

To encode a given test data using Huffman coding, first, the test data is divided into n-bit blocks, where n is a given block size, and the frequency of occurrence of each block pattern is counted. Each block pattern is represented by a binary codeword. Table 2 shows the frequency of occurrence of the block patterns and its codeword for the test data shown in Table 1. The test data is compressed by mapping a high (low) frequency block pattern into a short (long) codeword. For example, when the given test data of Table 1 is encoded using the Huffman codes of Table 2, the data size is compressed into 14 bits from 32 bits. In this case the compression ratio $R = 14/32 \simeq 43.8\%$.

Table 1 Example of codings.

	test data before/after coding	size (ratio)
Given test data (4-bit blocks)	0001 0000 0011 0000 0000 0100 0000 0011	32 bits (–)
Huffman coding	111 0 10 0 0 110 0 10	14 bits (43.8%)
Given test data (0-runs)	0001 0000001 1 0000000001 000000001 1	32 bits (–)
Golomb coding	011 1010 000 11001 11000 000	23 bits (71.9%)

Figure 4 shows a block diagram of a Huffman decompressor. The decompressor consists of an FSM (finite state machine) which decodes a codeword into the corresponding block pattern, and a serializer which scans out into a single internal scan chain of a CUT. Figure 5 shows a state diagram of the decoder designed for the Huffman code in Table 2. State $S0$ is a initial state. Signal *input* is a serial input from a tester channel. Signal *data* expresses n-bit block patterns and signal *valid* notifies whether a block pattern on signal *data* is prepared.

• **Golomb Coding** [4]

Golomb coding is a kind of run-length coding, which is based on the length of 0-runs in a test data. A run consists of successive 0s followed by a single 1, the length of a run is the number of 0s. Golomb coding has *group size* as a pa-

Table 2 Example of Huffman table.

block pattern	freq.	codeword
0000	4	0
0011	2	10
0001	1	111
0100	1	110

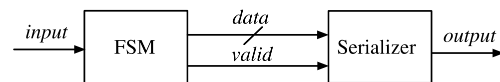


Fig. 4 Block diagram of Huffman decompressor.

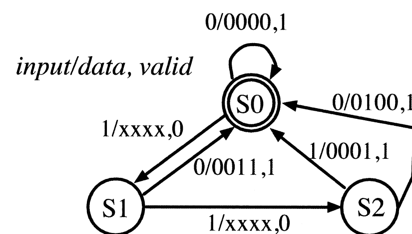


Fig. 5 FSM of Huffman decompressor.

Table 3 Golomb table.

run	prefix	tail	codeword
1	0	00	000
01	0	01	001
001	0	10	010
0001	0	11	011
00001	10	00	1000
000001	10	01	1001
0000001	10	10	1010
00000001	10	11	1011
000000001	110	00	11000
0000000001	110	01	11001
00000000001	110	10	11010
000000000001	110	11	11011

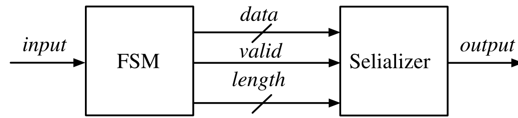


Fig. 6 Block diagram of Golomb decompressor.

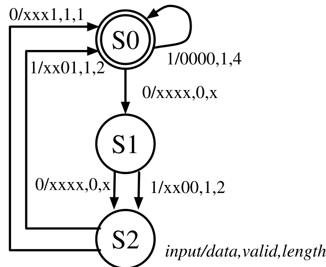


Fig. 7 FSM of Golomb decompressor.

rameter. A codeword is divided into *prefix* and *tail*. Table 3 shows Golomb codes whose group size is four. The prefix identifies each group to which the run belongs according to the number of 1s. The tail is fixed in length and identifies a member within the group. A run-length of the codeword is obtained by adding the numbers which the prefix and tail express [4].

Figure 6 shows a block diagram of a decompressor for Golomb coding. The decompressor consists of an FSM and a serializer, as well as Huffman coding. The FSM decodes a received codeword into the corresponding run according to the rule of Golomb coding, as shown in Table 3, and then send to the serializer the decoded run with its length via signals *data* and *length*. Signal *valid* becomes one if the both signals are valid. Fig. 7 shows a state diagram of the FSM.

The serializer receives a decoded run with its length, and then serially shifts the decoded run out. The group size, i.e., the number of members in each group, denotes the maximum length of 0-runs received by the serializer at a time, and hence the size of the serializer should be larger than the group size.

4. Reconfigurable Embedded Decompressor Architecture

In this section, we consider an architecture of reconfigurable embedded decompressors. The architecture employs not only Huffman and Golomb codings, as explained in Sect. 3, but also FDR [5] and VIHC [6] codings, which are variable-length codings based on run-length, like Golomb coding.

When the decompressor is reconfigured, a configuration data, which determines the decoding algorithm of the decompressor, is sent from an ATE to the decompressor. Hence, a test data includes the configuration data in addition to a compressed test input data. To bring out the merit of the reconfigurable decompressor, the decrease in the configuration data is important. The function of a decompressor is divided into two sub functions: one depends on coding algorithms while the other does not. In order to reduce

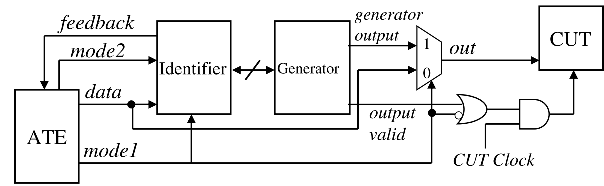


Fig. 8 Reconfigurable embedded decompressor architecture.

Table 4 Mode assignment.

mode	mode1	mode2
configuration	1	0
decompression	1	1
direct	0	1

the configuration data, hence, it is desirable that the functions independent of coding algorithms are implemented as common fixed components, i.e., not reconfigurable, while the function dependent on each coding algorithm is implemented as reconfigurable components.

A decompressor for variable-length codings necessarily includes an FSM to control itself, although the function of the FSM differs among the decoding algorithms. Therefore, in our architecture, a reconfigurable FSM is implemented as a common reconfigurable component and other parts, e.g., serializer, are implemented as a fixed component.

The common reconfigurable component is called *Identifier* and the fixed component is called *Generator* after [6]. The proposed reconfigurable embedded decompressor architecture is shown in Fig. 8.

The three modes, configuration, decompression, and direct, are switched by 2-bit signal *mode*1 and *mode*2. The assignment of the signals for each mode is shown in Table 4. When the value of signal *mode*1 is zero, signal *data* is directly applied into the CUT, and it goes into the Identifier when the value is 1. Signal *mode*2 notifies the Identifier the content of signal *data*. When the value of signal *mode*2 is zero, signal *data* is for a configuration data, and then the value is one, *data* is for a test data.

The CUT receives its test data through signal *generator output*, and the CUT clock becomes valid only when signal *output valid* is high or signal *mode*1 is low.

The details of the Identifier and the Generator as follows.

- **Identifier**

The Identifier achieves the FSM composed of decompressors as shown in Figs. 5 and 7, and it can be reconfigured. Therefore, it should be a reconfigurable device which specializes in description of FSMs. The Identifier works when signal *mode*1 is one. It is reconfigured when signal *mode*2 is zero, and it decompresses test data when the signal is one.

- **Generator**

The Generator is a fixed part and can be used in common. The block diagram of the Generator is shown in Fig. 9. The Generator consists of a controller (*Generator-FSM*), two counters (*p-counter* and *c-counter*) and a *shift-register*.

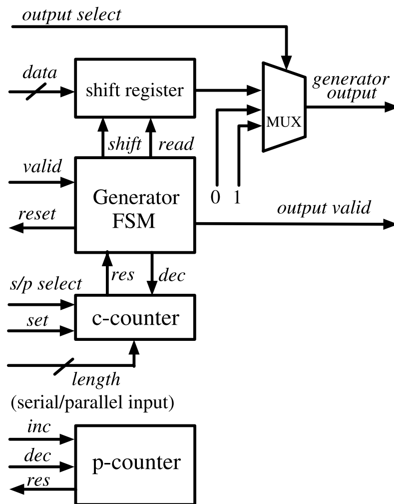


Fig. 9 Block diagram of the Generator.

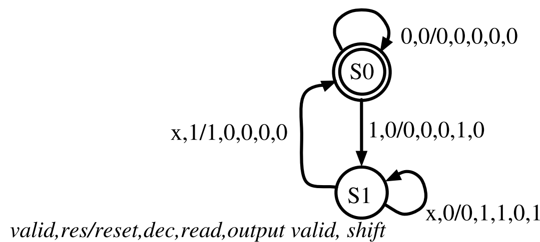


Fig. 10 State diagram of Generator-FSM.

It has eight inputs from the Identifier, two outputs to the Identifier and two outputs to the CUT.

The basic operation of the Generator is that the value memorized in *shift-register* is shifted out by the number corresponding to the value memorized in *c-counter*. The details are as follows. First, a value is set to *c-counter* serially or parallel via *length* by the Identifier when signal *set* is enabled. When signal *s/p select* denotes serial (parallel), the value is set serially (in parallel). Next, when signal *valid* is enabled, *Generator-FSM* makes input signal *data* read into *shift-register*, and sends signal *dec* to *c-counter*. Signal *dec* decrements *c-counter* until the value of *c-counter* becomes zero. When *c-counter* reaches zero, signal *res* is sent to *Generator-FSM*. *Generator-FSM* makes the value of *shift-register* shift out to the signal *generator output* until it receives signal *res*. Note that the value is shifted out only when signal *output select* selects the output of *shift-register*; the other selections, fixed zero and one, are used for FDR coding. The state diagram of *Generator-FSM* which performs the above operation is shown in Fig. 10.

In the case where the Identifier is used for Huffman coding and Golomb coding, *c-counter* memorizes the block size for Huffman coding, and the length of 0-runs for Golomb coding, while *shift-register* memorizes the blocks patterns for Huffman coding, and the 0-run for Golomb coding. Note that input signals *data*, *valid* and *length* of the Identifier are corresponding to the signals in the state diagram of Huffman and Golomb decoders shown in Figs. 5 and 7. The other unused inputs are fixed to proper values.

VIHC coding is a coding algorithm combining Huffman coding and Run-length coding. Therefore, the operation of the Identifier for VIHC coding is the same as that for Golomb, i.e., the *shift-register* and *c-counter* memorize a 0-run and its length, respectively.

FDR coding is a kind of run-length coding, and is similar to Golomb coding. However, the operation of the Identifier for FDR coding is not similar to that for Golomb coding. In the case where FDR coding is used, 0-runs are constructed by controlling signal *output select* of the multiplexer of the Generator from the Identifier, i.e., selecting the fixed one and zero appropriately. The *c-counter* and *p-counter* are used for memorizing the length of the prefix and tail of FDR codewords. The *shift-register* is not used in this case.

Here, let us discuss the are overhead of the proposed method. In general, the area efficiency of reconfigurable devices is lower than that of non-reconfigurable devices owing to the reconfigurable mechanism. The proposed decompressor, however, can keep its size small because of the following two reasons. One is that the separation of the proposed decompressor into the Identifier and the Generator will work for reducing the area overhead, not only reducing the configuration data. The second is that the reconfigurable part (or the Identifier) for the proposed decompressor specializes in description of FSMs required for coding algorithms.

5. Evaluation of Reconfigurable Embedded Decompressor

To validate the efficiency of the proposed architecture, three experiments were performed on the full-scan version of large ITC99 benchmark circuits [12]. Their test data were obtained using Synopsys TetraMAX ATPG. In the proposed decompressor, the common Generator and each Identifier for Huffman coding, Golomb coding, VIHC coding, and FDR coding are described by Verilog-HDL. To estimate the size of each Identifier these are synthesized and mapped to *slices*, which are reconfigurable blocks on FPGAs, using a tool “Xilinx-ISE-Foundation” for synthesis, place and route of FPGAs. Here, we assume that the configuration data size is increasing in proportion to the area of the Identifier.

5.1 Configuration Data Size

Table 5 shows comparisons between the configuration data size required for the non-divided architecture, i.e., the whole function is implemented on a reconfigurable device, and the proposed architecture for b14. The table lists the number of slices corresponding to the configuration data for the non-divided architecture and the proposed architecture. We employed Huffman coding, Golomb coding, and VIHC coding with three different block (group) sizes 4, 8, and 16, and FDR coding.

From Table 5, for each coding algorithm, we can see

that the proposed architecture reduces the size of the configuration data. We can also see that FDR coding can greatly reduce the configuration data size. This is because the main decoding functions for FDR coding, is achieved by the Generator, and they are larger than the circuits achieved by the Identifier.

5.2 Application to SoC Testing

Next, since the proposed reconfigurable embedded decompressor can change its coding algorithm during test application, we attempted to use the reconfigurability in order to improve a compression ratio in SoC testing. In the experiment, we suppose that circuits b15 and b21_1 are cores on an SoC. We employed Huffman and FDR codings, and compared the following two cases. A conventional case is that a Huffman (FDR) decompressor is used for decompressing test data, encoded by Huffman (FDR) coding, for both cores. The proposed case is that a reconfigurable decompressor is used, and the decoding algorithm is switched between Huffman and FDR codings according to the encoding algorithm of the applied test data. Table 6 shows comparisons between two cases.

In Table 6, we show the size of given test data T_D and the size of compressed test data T_R. Column *configsize* shows the size of the configuration data. This is calculated as follows. Since the tool "Xilinx-ISE-Foundation" reports the size of configuration data of the whole FPGA, say *Allconfigsize*, with its utilization ratio, which is the ratio of the number of the used slices to the number of the all slices, say R_u, *configsize* is the product of *Allconfigsize* and R_u. Note that T_R for SoC (b15+b21_1) in column "Huffman Decompressor" is different from the sum of the T_R for b15 and b21_1. This is because these Huffman decoders are optimally designed based on the different test sets, so that the Huffman codings employed in the decoders for the SoC and for cores b15 and b21_1 are different from one another.

Columns R_1 and R_2 report the compression ratios of the conventional case and the proposed case, respectively. They are given by

$$R_1 = \frac{T_R}{T_D}, R_2 = \frac{T_R + configsize}{T_D}. \quad (1)$$

Note that R_2 depends on *configsize* because the test data includes the configuration data in the proposed method.

From the table, b15 obtains a high compression ratio with FDR coding, while b21_1 obtains a high compression ratio with Huffman coding. Therefore, in the proposed case, the reconfigurable embedded decompressor selects FDR coding for b15, and selects Huffman coding for b21_1, so that the compression ratio R_2 of the proposed case is lower than the R_1s of the conventional cases.

5.3 Parameter Adjustment of Huffman Coding

Finally, we show an experimental result for adjusting a parameter of Huffman coding algorithm, i.e., the block size, in order to minimize the size of test data. The proposed reconfigurable decompressor can adjust the parameters of an implemented coding algorithm on itself. In Huffman coding, the size of a compressed test data theoretically decreases as the block size increases, while the size of configuration

Table 5 Comparisons on configuration data size (b14).

coding	parameter †	non-divided (slices)	ours (slices)	reduction ratio††
Huffman	4	25	15	0.40
	8	105	90	0.15
	16	352	333	0.05
Golomb	4	10	4	0.60
	8	23	10	0.57
	16	14	8	0.43
VIHC	4	13	5	0.62
	8	28	17	0.39
	16	98	78	0.20
FDR	-	33	8	0.76

†block size (Huffman), group size (Golomb, VIHC)
††(reduction ratio) = {(non-divided) - (ours)} / (non-divided)

Table 6 Experiment result for SoC and their cores.

circuit	Huffman Decompressor		FDR Decompressor		Proposed Decompressor			T_D
	T_R	R_1	T_R	R_1	T_R	*configsize*	R_2	
SoC (b15+b21_1)	384,328	0.760	328,011	0.648	187,573	30,053	0.430	506,140
b15	204,404	0.922	34,654	0.156	34,654	5,864	0.183	221,648
b21_1	152,919	0.538	293,357	1.031	152,919	24,189	0.623	284,492

Table 7 Relationship between block size and data sizes.

circuit		block size							T_D
		2	3	4	5	6	7	8	
b11	*configsize*	**274**	823	1,967	3,934	6,131	9,379	11,209	3,344
	T_R + *configsize*	**2,646**	2,999	4,069	6,083	8,141	11,393	13,118	
b14_1	*configsize*	2,932	6,597	**11,728**	34,451	65,237	133,406	238,225	157,616
	T_R + *configsize*	102,019	85,062	**79,618**	96,533	123,412	188,876	291,494	
b17	*configsize*	1,466	3,665	13,194	35,917	**71,101**	141,469	248,487	1,582,680
	T_R + *configsize*	842,055	597,816	488,084	438,655	**425,763**	462,352	543,496	
b18	*configsize*	2,932	3,665	21,990	38,116	74,033	**134,872**	255,817	4,156,320
	T_R + *configsize*	2,383,460	1,715,120	1,420,006	1,247,613	1,159,453	**1,132,210**	1,186,901	

data increases as it increases. Accordingly the relationship between the size of compressed test input data and that of configuration data is trade-off. This means that there is an optimal block size that minimizes the test data, which is the sum of the test input data and the configuration data.

Table 7 shows the size of the configuration data, $configsize$, and the test data, $T_R+configsize$, for block sizes from two to eight. An optimal block size which minimizes the test data is shown in the bold type. The last column shows the size of the test input data, T_D. From the table, we can see that the optimal block size is different for each circuit, and the block size becomes large as the size of test data becomes large.

6. Conclusion

We proposed an architecture of reconfigurable embedded decompressor for test compression. The proposed architecture reconfigures its internal operation by sending configuration data, and therefore it can deal with various coding algorithms. This gives the flexibility in design flow of LSIs as well as in test application. By isolating the Identifier from the decompressor, we reduced the data size for reconfiguration. The proposed decompressor achieved high compression of given test data by selecting an appropriate coding algorithm with its parameters, even though the test data includes its configuration data.

Acknowledgements

The authors would like to thank Prof. Yuki Yoshikawa and members of Computer Design Laboratory, Hiroshima City University for their valuable comments. 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).

References

[1] V. Iyengar, K. Chakrabarty, and B.T. Murray, “Built-in self testing of sequential circuits using precomputed test sets,” Proc. VTS, pp.418–423, 1998.

[2] A. Jas, J. Ghosh-Dastidar, and N.A. Touba, “Scan vector compression/decompression using statistical coding,” Proc. VTS, pp.114–120, 1999.

[3] A. Jas and N.A. Touba, “Test vector decompression via cyclical scan chains and its application to testing core-based designs,” Proc. ITC, pp.458–464, 1998.

[4] A. Chandra and K. Chakrabarty, “System-on-a-chip test data compression and decompression architectures based on Golomb codes,” IEEE Trans. CAD/ICAS, vol.20, no.3, pp.355–368, March 2001.

[5] A. Chandra and K. Chakrabarty, “Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression,” Proc. VTS, pp.42–47, 2001.

[6] P.T. Gonciari, B.M. Al-Hashimi, and N. Nicolici, “Variable-length input huffman coding for system-on-a-chip test,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.22, no.6, pp.783–796, 2003.

[7] A. Morosov, K. Chakrabarty, M. Gössel, and B. Bhattacharya, “Design of parameterizable error — Propagating space compactors for response observation,” Proc. VTS, pp.48–53, 2001.

[8] B. Pouya and N.A. Touba, “Synthesis of zero — Aliasing elementary — Tree space compactors,” Proc. VTS, pp.70–76, 1998.

[9] K. Chakrabarty, B.T. Murray, and J.P. Hayes, “Optimal zero — Aliasing space compaction of test responses,” IEEE Trans. Comput., vol.47, no.11, pp.1171–1187, Nov. 1998.

[10] S. Kajihara, K. Taniguchi, K. Miyase, I. Pomeranz, and S.M. Reddy, “Test data compression using don’t-care identification and statistical encoding,” IEICE Trans. Inf. & Syst., vol.E87-D, no.3, pp.544–550, March 2004.

[11] H. Ichihara and T. Inoue, “A test generation for compressible and compact test sets,” IEICE Trans. Inf. & Syst. (Japanese Edition), vol.J88-D-I, no.6, pp.1021–1028, June 2005.

[12] http://www.cerc.utexas.edu/itc99-benchmarks/bench.html

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 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.

Tomoyuki Saiki received his Bachelor and M. E. degrees of Information Engineering from Hiroshima City University in 2004 and 2006, respectively. He is currently with Canon Inc.

Tomoo Inoue is a professor of Faculty 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 such as field-programmable gate arrays. He received the BE, ME and Ph.D. degrees from Meiji University, Kawasaki, Japan, in 1998, 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 and IPSJ.