# An Internal Pattern Run-Length Methodology for Slice Encoding

Lung-Jen Lee, Wang-Dauh Tseng, and Rung-Bin Lin

**A simple and effective compression method is proposed for multiple-scan testing. For a given test set, each test pattern is compressed from the view of slices. An encoding table exploiting seven types of frequently-occurring pattern is used. Compression is then achieved by mapping slice data into codewords. The decompression logic is small and easy to implement. It is also applicable to schemes adopting a single-scan chain. Experimental results show this method can achieve good compression effect.**

**Keywords: Fixed-to-variable length, pattern run-length, multiple-scan testing, test-data compression.**

## I. Introduction

Modern system-on-a-chip (SoC) designs integrate several cores into a single chip in which a growing number of transistors are used within an even smaller chip area. However, this increased design complexity and the chip density are likely to create many more faults. To test them, a large amount of test data is required, which has seriously impacted the memory cost and the transmission efficiency between automatic test equipment (ATE) and SoC. Now, test-data volume has been recognized as a major contributor to the cost of manufacturing testing for integrated circuits [1]-[4]. In order to reduce the volume of the test data, many new techniques have been proposed. Among them, test-data compression is a simple and very effective solution because, in addition to the saving of the ATE memory, it also shortens the testing time and hence directly contributes to the test cost. During testing, the compressed data is first transferred through test channels to SoC, decompressed losslessly by the on-chip decoder, and then serially scanned into the scan chain for later circuit test.

Three major compression schemes are widely used for test-data compression: code-based schemes, linear-decompressor-based schemes, and broadcast-scan-based schemes [5]. Although the latter two schemes can often provide better compression, they require circuit structure information for conducting automatic test-pattern generation (ATPG) or fault simulation, which makes them inapplicable for intellectual property cores. Code-based schemes encode test data by a number of codewords according to specific properties embedded in corresponding bit-strings of the test data. The Huffman code [6] has been proven to be an optimal statistical code as it provides the shortest average codeword length among all uniquely decodable variable length codes. However,

Manuscript received June 11, 2010; revised Oct. 18, 2010; accepted Nov. 12, 2010.

Lung-Jen Lee (phone: +886 3 463 8800, ext. 2376, email: longlen1@yahoo.com.tw), Wang-Dauh Tseng (email: wdtseng@saturn.yzu.edu.tw), and Rung-Bin Lin (email: csrlin@saturn.yzu.edu.tw) are with the Department of Computer Science & Engineering, Yuan Ze University, Taiwan, Rep. of China.

doi:10.4218/etrij.11.0110.0319

it suffers from the exponentially increasing decoder size. Many extending works such as selective Huffman-coding (SHC) [7], optimal SHC [8], and variable-length input Huffman coding (VIHC) [9] are also proposed. SHC [7] is an efficient compression method with low hardware overhead where only the most frequently occurring symbols are encoded. A 9C technique [10] specifies exactly nine codewords for the precomputed data of the intellectual property cores in SoC. It is flexible in utilizing both fixed-length and variable-length blocks. Besides, the run-length-based compression method is also an efficient code-based scheme which encodes runs of repeated values. Examples include GOLOMB [11], frequency-directed run-length code (FDR) [12], alternating run-length coding (ALT-FDR) [13], pattern run-length (PRL) [14], and extended frequency-directed run-length code (EFDR) [15]. ALT-FDR [13] is a variable-to-variable length compression method extended from FDR [12]. PRL [14] is also very efficient in compressing consecutive patterns in an innovative manner. A block merging (BM) technique [16] was proposed by El-Maleh in which runs of fixed-length blocks are encoded, and only the merged blocks and number of blocks merged are recorded. In [17], the method of multidimensional pattern run-length compression (MD-PRC) is proposed where multiple pattern information is considered for compressing runs of variable-length patterns.

Although test-data compression has been extensively researched, compression in a slice view is seldom addressed. In this paper, a simple yet effective compression/decompression method targeted on slices is proposed. We first analyze the occurring frequencies for specific types of compatible data existing in slices and then conclude the seven most frequently occurring types used for later test-data compression. Seven simple prefix codes are used. Before compression, the occurrence frequencies for each type are analyzed so that the best compression can be achieved by repeatedly mapping the shortest codeword to the most frequently-occurring type. The decompression logic is also simple. Experimental results show the proposed method is very effective. The rest of this paper is organized as follows. Section II presents the proposed method and the decompression architecture. To evaluate the effectiveness of the proposed method, in section III, experiments for ISCAS'89 benchmark circuits are conducted. Finally, we conclude this work in section IV.

## II. Proposed Method

This paper presents a simple compression/decompression method where test data is compressed in views of slices for multiple-scan testing. A slice is defined as the $k$-bits loaded in parallel from ATE into $k$ scan chains in each clock cycle. Test data in each test vector is first partitioned into $k$-bit slices where $k$ is the total number of scan chains. Seven types of test data are observed most effective for slice compressions, where each type corresponds to a specific codeword. Compression is thus achieved by repeatedly mapping the most frequently-occurring type to the shortest codewords to pursuit high compression effect.

### 1. Test-Data Compression

This method is quite simple and straightforward. For a given test set, test data is first partitioned into several slices vector by vector. Slices are then encoded based on the encoding table. The format of each codeword is composed of three parts: prefix, extend, and tail. Prefix and extend represent the compression status for the target slice, while tail records the encoded data. Table 1 shows the seven compression types used in the proposed method. In this example, each slice contains eight bits ($k$=8). Each type is illustrated respectively as follows.

#### *A. All 0*

If each bit in the target slice is either 0 or X, then the Xs can be mapped by 0s. In this case, test data in the target slice can be encoded into the codeword 00 which is also a prefix, as shown in the second row of Table 1.

#### *B. All 1*

If each bit in the target slice is either 1 or X, then each X will be mapped to 1. In this case, test data in the target slice can be encoded into the codeword 01, which is also a prefix, as shown in the third row in Table 1.

#### *C. Repeat*

If data in the target slice is compatible with the previous slice, then the target slice can be encoded into the codeword 10, which is also a prefix, as shown in the fourth row in Table 1.

Table 1. Internal pattern run-length coding for $k$=8.

| Type | Slice data | Prefix | Extend | Tail |
|---|---|---|---|---|
| All 0 | 00X00XX0 | 00 | None | None |
| All 1 | 1XX111X1 | 01 | None | None |
| Repeat | X0X100X1 | 10 | None | None |
| 1/4 copy | 0XX101XX | 11 | 00 | 01 |
| 1/2 copy | 1X10111X | 11 | 01 | 1110 |
| 1/2 inverse copy | 1X10010X | 11 | 10 | 1010 |
| Original | 1X100X10 | 11 | 11 | 11100110 |

*D. 1/4 Copy*

When dividing the target slice data into four subslices, if data in the four subslices is mutually compatible, then the entire slice data can be encoded by Prefix 11 and Extend 1100 followed by the data in the subslice after performing the X mappings, as shown in the fifth row in Table 1.

*E. 1/2 Copy*

If the left half slice data is compatible to the right half, then the entire slice data can be represented by Prefix 11 and Extend 1101 followed by the data in either half slice after performing the X mappings as its tail, as shown in the sixth row in Table 1.

*F. 1/2 Inverse Copy*

If data in the left half is compatible to the inverted data of the right half , then the target slice can be encoded by Prefix 11 and Extend 1110 followed by the data in the left half slice after performing the X mappings, as shown in the seventh row in Table 1.

*G. Original Slice*

If the target slice does not match any of the above cases, then it will be encoded by Prefix 11 and Extend 1111, followed by its entire slice data, as shown in the last row in Table 1.

Table 2 gives an example for encoding a simplified test set in which test data is partitioned into 10 slices ($S_1$ to $S_{10}$), and each slice has 8 bits. As shown in $S_1$, the slice data 11X11XX1 matches the case All 1 after assigning 1 to each X. Therefore, $S_1$ can be encoded by codeword 01. For $S_2$, the slice data 11XXXX01 matches the case 1/2 copy after a proper assignment for each X. Therefore, $S_2$ can be encoded by codeword 11011101. For $S_3$, the slice data 11XXXX01 is completely compatible to the data in $S_2$. Therefore, $S_3$ matches the case Repeat and can be encoded by codeword 10. In the same manner, $S_4$ is completely compatible to $S_3$. For $S_5$, the slice data X0XXXXXX matches the case All 0 after assigning 0 to X and can be encoded by codeword 00. For $S_6$, the slice data X01XXX0X matches the case 1/2 inverse copy and has 1010 as its tail. Therefore, $S_6$ has the codeword 11101010 after assigning proper values to Xs.

Similarly, slice data X01XXXX1 in $S_7$ matches the Repeat type and can be encoded by the codeword 10. For $S_8$, the slice data 101X0XX1 matches the case 1/2 inverse copy and can be encoded by the codeword 11101010. For $S_9$, the slice data 1010XXX1 is completely compatible with $S_8$ and matches the case Repeat. Therefore, $S_9$ can be encoded by codeword 10. For $S_{10}$, the slice data 011XXXX1 matches the case 1/2 copy. Therefore, after a proper assignment to Xs, $S_{10}$ can be encoded by codeword 11010111. Consequently, after compression, the total test-data volume can be reduced from the original 80 bits to 44 bits, having achieved the compression ratio of 45%. Notably, during compression, the target slice data may match more than one case simultaneously; in this situation, the case with a shorter codeword will be selected.

Table 2. Encoding example for $k$=8.

| Slice | Slice data | Codeword | Type |
|---|---|---|---|
| $S_1$ | 11X11XX1 | 01 | All 1 |
| $S_2$ | 11XXXX01 | 11011101 | 1/2 copy |
| $S_3$ | 11XXXX01 | 10 | Repeat |
| $S_4$ | X1XXXX0X | 10 | Repeat |
| $S_5$ | X0XXXXXX | 00 | All 0 |
| $S_6$ | X01XXX0X | 11101010 | 1/2 inverse copy |
| $S_7$ | X01XXXX1 | 10 | Repeat |
| $S_8$ | 101X0XX1 | 11101010 | 1/2 inverse copy |
| $S_9$ | 1010XXX1 | 10 | Repeat |
| $S_{10}$ | 011XXXX1 | 11010111 | 1/2 copy |

*H. Look-Ahead Compression Enhancement*

Among the seven encoding types as stated above, Repeat is the only one that involves the dependency between consecutive slices. This specific property can often offer another chance for further improving the compression ratio during compression. In the example in Table 2, the Repeat type is a frequently-used one, and hence it is given with a smaller codeword size. For compressing a target slice, except the policy of selecting a shorter codeword, if there is still more than one choice, the type that can achieve a repeat compression for the next slice will be selected first. For example, both types, 1/2 copy and 1/2 inverse copy, can be selected for compressing $S_6$ (X01XXX0X) because they have the same codeword length. However, to look ahead considering the compression for the next slice $S_7$ (X01XXXX1), the 1/2 inverse copy would be the best choice for $S_6$ because it can achieve a repeat compression for $S_7$ and end up with a smaller total codeword size of 10 bits ($S_6$ and $S_7$ have the codeword size of 8 bits and 2 bits, respectively). The total codeword size would otherwise increase to 16 bits where $S_7$ is compressed by the 1/2 copy type and both $S_6$ and $S_7$ have the same codeword size of 8 bits.

## 2. Decompression Architecture

To recover the original test data, a parallel decoder is used in which the code identifier receives compressed data at $f_{ate}$. Also,

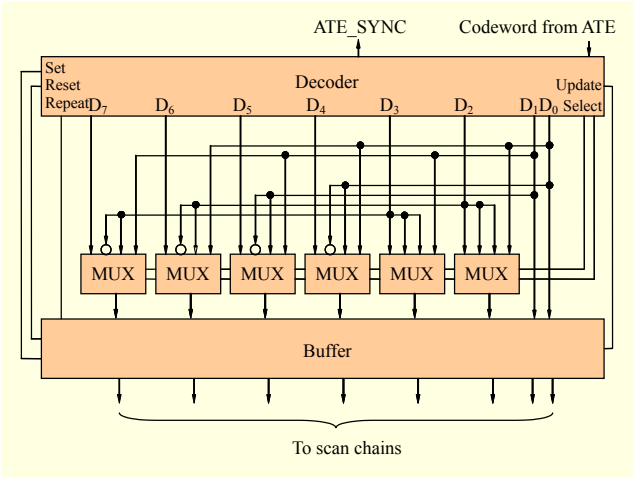


Fig. 1. Decompression architecture.

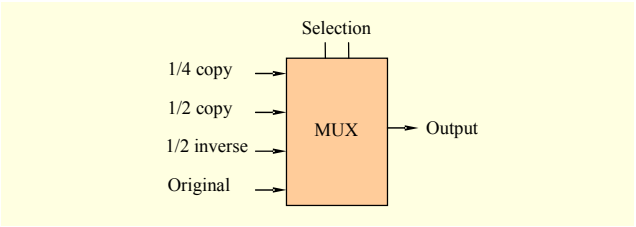


Fig. 2. Block diagram for multiplexer.

the pattern generator generates test data at $f_{chip}$. Only one ATE channel is required for feeding codewords to the decoder. Since the codeword lengths are varied, the signal ATE_SYNC is used to stop the transmission of codewords from ATE when the decoder is not available. For types All 0, All 1, and Repeat, the buffer data will be reset to 0, set to 1, and repeated again, respectively. For types 1/2 copy, 1/2 inverse copy, and 1/4 copy, the received subslice data will be broadcast and transmitted to the buffer according to the associated control signals. Although codewords can be decompressed in different data paths, timing is dominated by the longest one.

As previously stated, this method specifies a prefix code for each codeword, which makes decompression quite simple. Since no codeword is also a prefix of any other, during decompression, every codeword can be simply identified by the decoder. As shown in Fig. 1, the main components include the decoder, multiplexer array, and a *k*-bit buffer. We will describe them as follows.

**Decoder.** A decoder is a finite state machine which receives codewords from ATE in a fixed rate synchronized with a circuit under test by signal ATE_SYNC, decides which state to go, and launches the associated signals to multiplexer array and buffer. Four signals, Set, Reset, Repeat, and Update, decide the contents of the buffer to be all ones, all zeros, no change, and input from multiplexers, respectively.

Table 3. Setting of signals from decoder for different type of slices.

| Type | Reset | Set | Repeat | Update | Select |
|---|---|---|---|---|---|
| All 0 | 1 | 0 | 0 | 0 | xx |
| All 1 | 0 | 1 | 0 | 0 | xx |
| Repeat | 0 | 0 | 1 | 0 | xx |
| 1/4 copy | 0 | 0 | 0 | 1 | 00 |
| 1/2 copy | 0 | 0 | 0 | 1 | 01 |
| 1/2 inverse copy | 0 | 0 | 0 | 1 | 10 |
| Original | 0 | 0 | 0 | 1 | 11 |

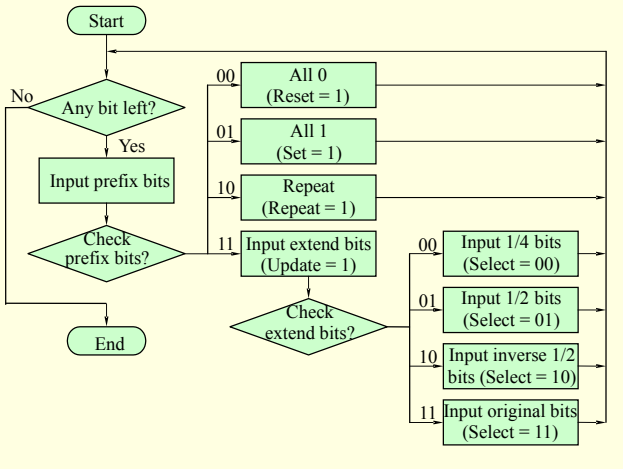


Fig. 3. Decompression flow.

**Multiplexer Array.** A multiplexer array is composed of multiplexers and inverters, by which the received subslice data in types 1/2 copy, 1/2 inverse copy, and 1/4 copy are broadcast and transmitted to the buffer according to the signal from decoder.

**Buffer.** A buffer receives the decompressed data and sends such data to the scan chain.

In the multiplexer array, the MUXes, as shown in Fig. 2, are used between the decoder and the buffer, each of which determines the output data according to the select signal from decoder. If the select signal is 11, then the original input is switched to output. If the select signal is 00, 01, or 10, then the corresponding input for states 1/4 copy, 1/2 copy, or 1/2 inverse copy is switched to the output. Table 3 lists the decoding for each type of slice and the associated signals. The decompression flow can be briefly illustrated as follows. As shown in Fig. 3, the decoder starts to receive and identify the prefix bits from the ATE input. If the prefix bits are 00, they then go to state All 0 and make Reset= 1. If the prefix bits are 01, they then go to state All 1 and make Set=1. If the prefix bits are 10, they then go to state Repeat and make Repeat=1. If the

prefix bits are 11, they then make Update=1 and shift in extended bits for a further identification. If the extend bits are 00, they then go to state 1/4 copy and make selection signal 00. If the extend bits are 01, they then go to state 1/2 copy and

Table 4. Encoding table for circuit s5378.

| Type | Frequency | Prefix + Extend |
|---|---|---|
| All 0 | 106 | 1100 |
| All 1 | 149 | 1101 |
| Repeat | 391 | 00 |
| 1/4 copy | 268 | 01 |
| 1/2 copy | 265 | 10 |
| 1/2 inverse copy | 200 | 1110 |
| Original | 173 | 1111 |

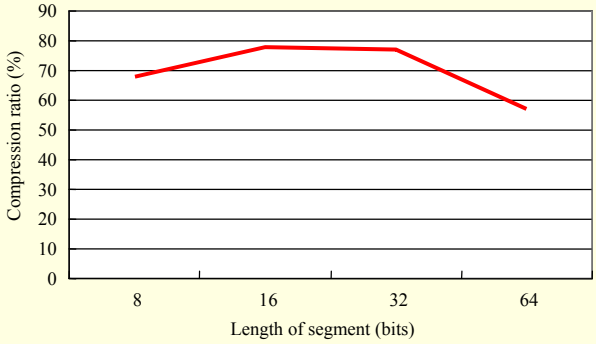


Fig. 4. Compression effect under different $k$'s for circuit s5378.

Table 5. Internal pattern run-length coding under different $k$'s for ISCAS'89 benchmarks.

| Circuit | $k$ (bits/slice) | | | | | | | | Best | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 8 | | 16 | | 32 | | 64 | | | |
| | M | S | M | S | M | S | M | S | M | S |
| s1423 | 21.6 | 23.6 | 25.6 | 25.5 | 18.8 | 20.8 | –3.0 | 10.5 | 25.6 | 25.5 |
| s208 | 12.2 | 15.9 | 14.8 | 16.5 | –1.3 | 0.1 | –14.1 | –10.7 | 14.8 | 16.5 |
| s298 | –1.9 | 10.4 | 3.9 | 8.0 | 6.9 | 8.6 | –14.4 | –14.1 | 6.9 | 10.4 |
| s344 | 15.3 | 13.4 | 6.8 | 30.2 | 7.0 | 12.1 | 3.8 | 9.5 | 15.3 | 30.2 |
| s349 | 15.7 | 14.4 | 13.1 | 28.0 | 7.2 | 9.6 | 5.4 | 10.2 | 15.7 | 28.0 |
| s382 | 26.3 | 17.4 | 22.3 | 23.7 | 15.9 | 18.0 | –1.5 | 1.0 | 26.3 | 23.7 |
| s400 | 24.1 | 17.9 | 22.8 | 21.9 | 13.7 | 15.8 | –4.2 | –1.6 | 24.1 | 21.9 |
| s420 | 29.2 | 33.9 | 30.9 | 28.2 | 21.3 | 20.0 | –11.1 | –9.3 | 30.9 | 33.9 |
| s444 | 20.2 | 13.5 | 17.4 | 24.3 | 9.0 | 10.5 | –7.8 | –4.6 | 20.2 | 24.3 |
| s526 | 12.0 | 3.2 | 7.4 | 14.7 | –5.3 | –6.0 | –20.5 | –19.7 | 12.0 | 14.7 |
| s820 | 22.9 | 21.7 | 18.1 | 23.7 | –0.3 | –1.6 | –23.5 | –22.5 | 22.9 | 23.7 |
| s832 | 22.5 | 21.6 | 18.4 | 23.1 | –2.9 | –4.4 | –23.9 | –22.6 | 22.5 | 23.1 |
| s1196 | 26.1 | 22.7 | 31.2 | 30.8 | 33.1 | 24.1 | 24.9 | 23.7 | 33.1 | 30.8 |
| s1238 | 27.4 | 25.1 | 32.2 | 30.8 | 33.5 | 24.7 | 25.0 | 23.9 | 33.5 | 30.8 |
| s510 | 38.5 | 36.1 | 45.5 | 47.0 | 37.6 | 36.7 | 7.4 | 8.2 | 45.5 | 47.0 |
| s838 | 47.4 | 51.4 | 52.4 | 53.0 | 39.5 | 40.9 | 26.9 | 24.6 | 52.4 | 53.0 |
| s953 | 47.3 | 44.8 | 52.7 | 53.7 | 43.5 | 50.6 | 27.1 | 26.7 | 52.7 | 53.7 |
| s5378 | 46.2 | 47.4 | 54.5 | 51.7 | 51.0 | 42.7 | 45.7 | 34.3 | 54.5 | 51.7 |
| s9234 | 49.4 | 48.5 | 53.5 | 54.1 | 46.4 | 46.6 | 34.1 | 38.1 | 53.5 | 54.1 |
| s13207 | 70.3 | 69.8 | 79.1 | 79.8 | 82.5 | 84.6 | 82.8 | 82.5 | 82.8 | 84.6 |
| s15850 | 59.4 | 59.9 | 66.5 | 67.0 | 64.2 | 67.6 | 60.5 | 62.1 | 66.5 | 67.6 |
| s38417 | 51.6 | 52.2 | 56.6 | 54.8 | 49.5 | 43.7 | 40.5 | 30.3 | 56.6 | 54.8 |
| s38584 | 59.0 | 60.1 | 66.1 | 66.4 | 64.7 | 69.1 | 60.3 | 62.1 | 66.1 | 69.1 |
| s35932 | 68.1 | 63.8 | 78.6 | 71.4 | 77.3 | 71.0 | 57.1 | 60.7 | 78.6 | 71.4 |

make selection signal 01. If the extend bits are 10, they then go to state 1/2 inverse copy and make selection signal 10. If the extend bits are 11, they then go to state Original and make selection signal 11. Repeat the above steps until no more codewords are left. As can be seen from the decompression architecture, decoding for codewords is simple and easy to implement. The decompressed data can be sent to multiple scan chains with a fixed scan shift rate.

## III. Experimental Results

We have implemented experiments on ISCAS89 benchmark circuits, adopting the test data generated by Mintest [18] ATPG with dynamic compaction. The compression effect is evaluated by the compression ratio, which is defined as $CR\% = [(|T_D| - |T_E|) / |T_D|] \times 100\%$, where $|T_D|$ is the size of the test set, and $|T_E|$ is the size of compressed test set. As mentioned previously, for each circuit, we flexibly specify the more frequently-occurring types to shorter codewords in order to achieve a better compression effect. Take the circuit s5378 as an example. The types Repeat, 1/2 copy, and 1/4 copy occur more frequently than the others, so we specify them shorter prefixes, as shown in Table 4. At first, we explore the impact of varying the slice length $k$ on the compression effect for circuit s5378. As shown in Fig. 4, the compression ratio reaches the peak at $k$=16 and drops as $k$ increases. The result shows the compression effect is strongly related to $k$. Table 5 reports the compression ratios with respect to different $k$'s for the ISCAS'89 benchmark circuits adopting single-scan chain (denoted as S) and multiple scan chains (denoted as M). We report the best results for each circuit in the last column. As can be seen, the distribution of test data for both the single-scan scheme and the multiple-scan scheme is similar. Therefore, no major difference resulted between them. Besides, the compression effect for smaller circuits is much lower than that for the larger circuits.

In Table 6, the compression ratios for six large circuits are reported with the averages in the last row. In Table 7, we compare the compression effect with other works on six larger

Table 6. Internal pattern run-length coding for six larger circuits.

| Circuit | $k$ (bits/slice) | | | | | | | | Best | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 8 | | 16 | | 32 | | 64 | | | |
| | M | S | M | S | M | S | M | S | M | S |
| s5378 | 46.2 | 47.4 | 54.5 | 51.7 | 51.0 | 42.7 | 45.7 | 34.3 | 54.5 | 51.7 |
| s9234 | 49.4 | 48.5 | 53.5 | 54.1 | 46.4 | 46.6 | 34.1 | 38.1 | 53.5 | 54.1 |
| s13207 | 70.3 | 69.8 | 79.1 | 79.8 | 82.5 | 84.6 | 82.8 | 82.5 | 82.8 | 84.6 |
| s15850 | 59.4 | 59.9 | 66.5 | 67.0 | 64.2 | 67.6 | 60.5 | 62.1 | 66.5 | 67.6 |
| s38417 | 51.6 | 52.2 | 56.6 | 54.8 | 49.5 | 43.7 | 40.5 | 30.3 | 56.6 | 54.8 |
| s38584 | 59.0 | 60.1 | 66.1 | 66.4 | 64.7 | 69.1 | 60.3 | 62.1 | 66.1 | 69.1 |
| s35932 | 68.1 | 63.8 | 78.6 | 71.4 | 77.3 | 71.0 | 57.1 | 60.7 | 78.6 | 71.4 |
| Average | 57.7 | 57.4 | 65.0 | 63.6 | 62.2 | 60.8 | 54.4 | 52.9 | 65.5 | 64.8 |

Table 7. Result comparisons with other works adopting single scan chain.

| Circuit | GOLOMB [11] | FDR [12] | ARL [13] | EFDR [15] | SHC [7] | VIHC [9] | 9C [10] | BM [16] | MD-PRC [17] | IPR |
|---|---|---|---|---|---|---|---|---|---|---|
| s5378 | 37.11 | 47.98 | 50.77 | 51.93 | 55.10 | 51.52 | 51.64 | 54.98 | 54.63 | 51.7 |
| s9234 | 45.25 | 43.61 | 44.96 | 45.89 | 54.20 | 54.84 | 50.91 | 51.19 | 53.20 | 54.1 |
| s13207 | 79.74 | 81.30 | 80.23 | 81.85 | 77.00 | 83.21 | 82.31 | 84.89 | 86.01 | 84.6 |
| s15850 | 62.82 | 66.21 | 65.83 | 67.99 | 66.00 | 60.68 | 66.38 | 69.49 | 69.99 | 67.6 |
| s38417 | 28.37 | 43.37 | 60.55 | 60.57 | 59.00 | 54.51 | 60.63 | 59.39 | 55.38 | 54.8 |
| s38584 | 57.17 | 60.93 | 61.13 | 62.91 | 64.10 | 56.97 | 65.53 | 66.86 | 67.73 | 69.1 |
| s35932 | - | 19.40 | - | 80.30 | 65.70 | 56.10 | - | - | 61.50 | 71.4 |
| Average | 51.74 | 51.83 | 60.58 | 64.49 | 63.01 | 59.69 | 62.90 | 64.47 | 64.06 | 64.8 |

Table 8. Comparisons with other works in decompressor area overhead (%).

| Circuit | GOLOMB [11] | FDR [12] | EFDR [15] | SHC [7] | VIHC [9] | 9C [10] | BM [16] | IPR |
|---|---|---|---|---|---|---|---|---|
| s5378 | 4.0 | 7.8 | 8.3 | 16.0 | 5.8 | 8.2 | 12.8 | 4.6 |
| s9234 | 3.2 | 5.9 | 6.3 | 13.0 | 4.6 | 6.2 | 9.7 | 3.6 |
| s13207 | 4.1 | 3.5 | 3.7 | 5.7 | 2.2 | 3.7 | 5.8 | 1.7 |
| s15850 | 2.0 | 3.6 | 3.8 | 6.5 | 2.3 | 3.8 | 5.9 | 1.8 |
| s38417 | 0.5 | 1.4 | 1.5 | 2.0 | 0.7 | 1.5 | 2.3 | 0.6 |
| s38584 | 0.7 | 1.5 | 1.6 | 2.0 | 0.7 | 1.6 | 2.5 | 0.6 |

circuits adopting single scan chain. Results show the proposed internal pattern run-length (IPR) method can achieve better compression effect in most cases.

In the following, we analyze the hardware overhead of the decompressor architecture. The benchmark circuits and the decompressor were synthesized using Synopsys Design Compiler. The decompressor area overhead is computed as

$$Area\ overhead = \frac{area\ of\ decompressor}{area\ of\ benchmark\ circuit} \times 100\%.$$

Table 8 shows the comparison results for the proposed method with other methods. As shown, the decompressor area overhead for the proposed method is quite limited.

## IV. Conclusion

We have proposed an efficient method for compressing slices of test data during multiple-scan testing. This method is motivated by the observation that seven types of pattern data occur frequently in most of the test sets. Through an efficient specification of codeword for each type, the test-data volume can be reduced effectively. The decompressor is small and easy to implement. Experimental results have shown that a good compression effect can be achieved for the six large ISCAS'89 benchmark circuits.

## Acknowledgement

The authors would like to thank the reviewers for their constructive comments to improve the readability and quality for this paper.

## References

[1] S. Mitra and K.S. Kim, "X-Compact: An Efficient Response Compaction Technique," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, 2004, pp. 421-432.

[2] J. Rajski et al., "Embedded Deterministic Test," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, 2004, pp. 776-792.

[3] S. Mitra and K.S. Kim, "XMAX: X-Tolerant Architecture for Maximal Test Compression," *Proc. IEEE Int. Conf. Comput. Design*, 2003, p. 326.

[4] B. Koenemann et al., "A SmartBIST Variant with Guaranteed Encoding," *Proc. Asia Test Symp.*, 2001, p. 325.

[5] N.A. Touba, "Survey of Test Vector Compression Techniques," *IEEE Design Test Comput.*, vol. 23, no. 4, 2006, pp. 294-303.

[6] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE*, 1952, p. 1098.

[7] A. Jas et al., "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 6, 2003, pp. 797-806.

[8] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Optimal Selective Huffman Coding for Test-Data Compression," *IEEE Trans. Comput.*, vol. 56, no. 8, 2007, pp. 1146-1152.

[9] P.T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression," *Proc. Design Autom. Test Europe*, Paris, 2002, p. 604.

[10] M. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-Coded Compression Technique for Testing Embedded Cores in SoCs," *IEEE Trans. VLSI Syst.*, vol. 13, no. 6, 2005, pp. 719-731.

[11] A. Chandra and K. Chakrabarty, "System-on-a-Chip Data Compression and Decompression Architecture Based on Golomb Codes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 3, 2001, pp. 355-368.

[12] A. Chandra and K. Chakrabarty, "Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-Directed Run-Length (FDR) Codes," *IEEE Trans. Comput.*, vol. 52, no. 8, 2003, pp. 1076-1088.

[13] A. Chandra and K. Chakrabarty, "A Unified Approach to Reduce SoC Test Data Volume, Scan Power and Testing Time," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 3, 2003, pp. 352-363.

[14] X. Ruan and R. Katti, "An Efficient Data-Independent Technique for Compressing Test Vectors in Systems-on-a-Chip," *Proc. IEEE Emerging VLSI Tech. Arch. Symp.*, 2006, p. 153.

[15] A.H. El-Maleh and R.H. Al-Abaji, "Extended Frequency-Directed Run Length Code with Improved Application to System-on-a-Chip Test Data Compression," *Proc. 9th IEEE Int. Conf. Electron., Circuits Syst.*, 2002, p. 449.

[16] A.H. El-Maleh, "Efficient Test Compression Technique Based on Block Merging," *IET Comput. Digit. Tech.*, vol. 2, no. 5, 2008, pp. 327-335.

[17] L.-J. Lee et al., "A Multi-Dimensional Pattern Run-Length Method for Test Data Compression," *Proc. Asian Test Symp.*, 2009, pp. 111-116.

[18] I. Hamzaoglu and J.H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 8, 2000, pp. 957-963.

**Lung-Jen Lee** received the MS and PhD degrees from the Department of Computer Science & Engineering, Yuan Ze University, Chung-Li, Taiwan, in 2006 and 2010, respectively. He is currently an assistant professor in the Department of Electronic Engineering, Army Academy, Chung-Li, Taiwan. His research interests include VLSI design and testing and low-power design methodologies.

**Wang-Dauh Tseng** received the BS degree in computer science from Soochow University, Taiwan, and the MS and PhD in computer and information science from National Chiao Tung University, Taiwan. He is currently an assistant professor in the Department of Computer Science and Engineering, Yuan Ze University, Chung-Li, Taiwan. His current research interests include fault-tolerant computing and VLSI design and testing.

**Rung-Bin Lin** received his PhD in Computer Science from the University of Minnesota, 1992. He was with the Large Scale Computing Division, IBM, Poughkeepsie, New York, from 1992 through 1994. He is currently a professor in the Department of Computer Science and Engineering, Yuan Ze University, Chung-Li, Taiwan. His research interests include physical design, low-power design, and VLSI testing and design for manufacturability.