

## Short Paper

---

# Test Data Compression for Minimum Test Application Time

PO-CHANG TSAI, SYING-JYAN WANG, CHING-HUNG LIN  
AND TUNG-HUA YEH

*Department of Computer Science  
National Chung Hsing University  
Taichung, 402 Taiwan*

In this paper, we proposed a test data compression scheme targeted for minimizing the amount of test data. The proposed scheme can reduce the test application time and minimize the amount of compressed test data, which reduces the size of data memory in ATE and the time needed to transfer test data. A decoder design is also presented. Experimental results on ISCAS benchmark circuits show that the compressed data produced by our method are much smaller than previous methods.

**Keywords:** test data compression, decompression architecture, SOC, testing, forced bit-inversion, intellectual property

## 1. INTRODUCTION

In Core-Based system-on-a-chip (SOC), the testing problem is mainly caused by the vast variety of intellectual property (IP) cores. Since IP cores may come from various IP vendors, a standard test interface has to be provided to facilitate the testing process. The IEEE 1500 [1] was proposed for that purpose. Another problem in SOC testing is due to the tremendous amount test data to be applied to the circuit under test (CUT). The limited I/O channel capacity, speed, and data memory in automatic test equipment (ATE) render it difficult to apply the enormous amount of test data to an SOC [2].

Many test set compaction algorithms have been developed [3, 4]. Among these methods, MinTest [4] provided the smallest test sets for scan test. The continuous scan [5] approach changes the test-per-scan structure to test-per-clock, so that test vectors can be overlapped and thus test data are reduced. Some test data reduction techniques are achieved through structure modification [6-11]. Test data encoding methods based on Golomb coding [7] was presented in [8-11]. This variable-length coding method achieves good compression rate with small hardware overhead. This data compression will reduce the requirement for data memory in ATE as well as the communication time to transfer test data from ATE to CUT. In order to achieve this goal, one usually needs to start with a test set that is not so compact (*i.e.*, not so random), since truly compact patterns cannot be efficiently compressed. As a result, the test application time, which is decided by the number of applied test patterns, will be longer.

---

Received September 30, 2005; revised December 23, 2005 & February 22, 2006; accepted March 16, 2006.  
Communicated by Liang-Gee Chen.

In this paper, we propose a new test data compression scheme that is targeted for minimizing both test application time and the volume of compressed test data. The proposed method starts with a very compact test set (*e.g.*, one from MinTest [4]), and tries to modify the given test vectors to a form that can be easily compressed. This method applies a Forced Bit-Inversion (FBI) algorithm, which is based on the Forced Pair-Merging algorithm [3], for test pattern modification. The test quality will not be sacrificed by this modification, since it reaches the same fault coverage as the original one. In this way, we can minimize test application time to achieve high compression rate. The proposed method was verified with some ISCAS benchmark circuits. In most cases, the amount of compressed test data produced by this method is much smaller than those obtained by other methods. Besides, our test application time is always shorter. A decoder for this scheme is also presented, and the area overhead is insignificant.

## 2. PRELIMINARIES

Some basic ideas for data compression and coding methods are presented in this section.

### 2.1 Test Data Compression

**Definition 1** The test set  $T_D$  is a precomputed test data for an IP core, while the compressed test set  $T_E$  is obtained by encoding  $T_D$  to a much smaller set. The size of the test data set and the compressed set are denoted as  $|T_D|$  and  $|T_E|$ .

**Definition 2** The *compression rate* ( $CR$ ) of an encoding scheme is  $(|T_D| - |T_E|)/|T_D|$ .

The compressed set  $T_E$  is stored in ATE memory, and an on-chip decoder is used for pattern decompression to obtain  $T_D$  from  $T_E$  during test application. The size of compressed test data,  $|T_E|$ , is decided by the compressing algorithm as well as the original  $T_D$ . The selection of  $T_D$  has a profound impact on the final  $T_E$ , since it is usually very difficult to compress a test set in which the distribution of 0's and 1's is random.

### 2.2 Golomb Code

Golomb coding is a powerful implementation of run-length encoding of binary streams. However, this encoding technique may not achieve a satisfactory compression rate when the probabilities of 0 and 1 are about the same. Before the encoding, first it is necessary to decide whether 0 or 1 is the more probable bit. Without loss of generality, we shall assume that 0 is the more probable bit in the following discussion.

**Definition 3** A stream of  $i$  consecutive 0's (1's) is denoted as  $0^i$  ( $1^i$ ).

**Definition 4** A stream of one or more consecutive 0's (1's) is denoted as  $0^*$  ( $1^*$ ).

**Procedure 1:** Golomb Encoding.

(1) A code word is broken into runs of the form  $0^i 1$ .

- (2) Encode each run of  $0^i1$  as  $1^q0y$ , where  $i = q \times m + r$  ( $0 \leq r < m$ ), and  $y$  is the binary representation of  $r$ , using  $\log_2 m$  bits.  $1^q$  is Group Prefix;  $y$  is Tail and 0 isolates Group Prefix and Tail.
- (3) The final coded bit stream is of the form:  $MPb\ code_1\ code_2\ \dots\ code_n$ , where  $MPb$ : is the (1-bit) value of the more probable bit,  $code_j$ : the code of the  $j$ th run.

### 3. PROPOSED COMPRESSION METHOD

The proposed scheme is a modified Golomb coding method, in which both  $0^*$  and  $1^*$  are encoded. The goal is to reduce both  $T_D$  and  $T_E$  with this encoding scheme.

#### 3.1 Basic Idea

Most of the research results on test data compression focus on reducing the amount of encoded test data ( $T_E$ ). However, this approach may actually lead to a larger volume of applied test data ( $T_D$ ), and thus increases test time. In order to reduce both  $T_D$  and  $T_E$ , the proposed scheme starts from a compact test set, in which the number of test vectors is near minimum. The proposed method tries to change each vector so that the new vector has longer runs of 0's and 1's but the fault coverage is not sacrificed. Instead, the proposed scheme tries to exploit consecutive 0's and 1's inside test patterns for compression.

There are two possible types of test set: either every bit in a test vector is fully specified, or some bit are not specified. A test vector with unspecified bits is usually referred to as a test cube. If the initial test set is not fully specified, it will be easier to compress. However, the size of recovered test data  $T_D$  will be larger, and thus the test application time is longer. An example of test set with test cubes is shown in Fig. 1. Since the don't-care bits in test cubes can be randomly assigned to 0 or 1, it is possible to produce longer runs of 0's and 1's. As a result, the compression rate can be greatly increased.

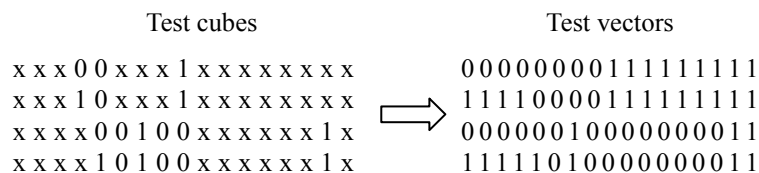


Fig. 1. Assign 0 or 1 to don't-care bits in test cubes.

In this paper, we adopt a greedy approach to assign don't-care bits from left to right, in which either '0' or '1' is selected to make the current run longer. According to this heuristic, the test cubes in Fig. 1 are assigned to the test vectors as shown. However, some test cubes cannot be efficiently handled with this heuristic. Consider the third and fourth test vectors in Fig. 1, in which there is a 1 sitting between two runs of 0's. In these cases, there are short runs that are difficult to be compressed. In order to push the compression rate to an even higher level, the idea used in Forced Pair-Merging (FPM) algo-

rithm [3] is applied here. This algorithm was first presented for static test set compaction. It used the concept of “raise bit” to change only one bit each time without affecting essential faults [3]. When a 1 is changed to 0, or vice versa, it is necessary to check if the new test vector can detect the same number of essential faults. Whenever the coverage of essential faults is not affected, the bit is raised to  $x$  (*i.e.*, don’t-care), so that the new test cube may be merged with other test cubes [3].

In the proposed method, however, it is not necessary to merge a test cube with other patterns to reduce the size of a test set. Instead, we look for those bits whose inversions (*i.e.*,  $0 \rightarrow 1$  or  $1 \rightarrow 0$ ) can create a longer run of  $0^*$  or  $1^*$ . For example, consider test vectors 3 and 4 in Fig. 1. The single 1’s locating between two runs of  $0^*$  are good candidates for bit inversion. For each candidate bit inversion, we need to check if the new test vector created by this bit inversion still detects the same set of essential faults. If the fault coverage is reduced, the bit inversion will not be accepted. This procedure is referred to as the Forced Bit-Inversion (FBI) henceforth. With a good don’t-care bits assignment heuristic and the FBI algorithm, we can achieve a very high compression rate. It will be discussed in the next section.

Since both  $0^*$  and  $1^*$  have to be encoded, the Golomb coding outlined in section 2.2 cannot be applied directly. In the proposed encoding scheme, a test vector is first changed to alternate runs of  $0^*$  and  $1^*$ , and then the length of each run is encoded in the same way as Golomb code. Once the value of the first run is known, the other runs are also known. The encoding mechanism of the two-value (2-V) Golomb code is similar to the alternating run-length coding scheme [9]. An example of the encoding scheme is shown in Fig. 2. In this example, we have  $m = 4$ ,  $n = 45$ ,  $CR = 10/45 = 0.22$ .  $R$  is the value of current run.

$T_D$	000	111111	0	1111	00000	1111111	00	111111111	00000000
	$l_1 = 3$	$l_2 = 6$	$l_3 = 1$	$l_4 = 4$	$l_5 = 5$	$l_6 = 7$	$l_7 = 2$	$l_8 = 9$	$l_9 = 8$
$T_E$	011	1010	001	1000	1001	1011	010	11001	11000
$R$	0	1	0	1	0	1	0	1	0

Fig. 2. A 2-value Golomb coding example.

### 3.2 Compression Procedure and FBI Algorithm

The proposed compression method works as follows.

#### Procedure 2: Test Data Compression

- Step 1:** Generate an initial test set  $T_{initial}$ .
- Step 2:** If all test patterns in  $T_{initial}$  are deterministic (*i.e.*, no test cubes), go to step 4.
- Step 3:** Assigned 0’s and 1’s into don’t-care bits to create longer runs of alternate  $0^*$  and  $1^*$ .
- Step 4:** Eliminate redundant test patterns, and the test set is  $T$ .
- Step 5:** Apply FBI algorithm to maximum run lengths in each test pattern. The result is a newtest set  $T'$ .
- Step 6:** Reorder patterns in  $T'$  to generate a sequence  $T_D$  with maximum  $0^*$  and  $1^*$ .
- Step 7:** Encode  $T_D$  with two-value Golomb codes.

Step 5 is the FBI algorithm, which inverts some bits in the test patterns. The FBI algorithm is outlined below. In each iteration of the **while** loop, the algorithm picks the pattern with the maximum number of bit transitions from the test set. The algorithm tries to invert some bits to get longer runs of 0's and 1's. For example, 00100 can be changed to 00000 or 00111, while 11011 can be changed to 11111 or 11000. After the bit inversion, we must update the essential fault list for this test pattern again. The function `update_essential_fault()` is for that purpose. When the above process is finished, the reduction of essential faults has to be checked. If there is no reduction, the move is accepted; otherwise, it is discarded.

**Algorithm: Foced Bit Inversion**

```

FBI( $T$ ) {
   $T^r = \emptyset$ ;
   $sw_i$  = the number of switching in pattern  $p_i$ ;
  while ( $T \neq \emptyset$ ) {
    select pattern  $p_k \in T$  with maximum  $sw_k$ ;
    change_pattern( $p_k$ ); // Similar to raise bit
    update_essential_fault_list();
     $T^r = T^r \cup \{p_k\}$ ;  $T = T - \{p_k\}$ ;
  }
  check_essential_fault_list();
}
return  $T^r$ ;

```

#### 4. DECOMPRESSION CIRCUIT

The decoder of 2-V Golomb Code consists of a Finite State Machine (FSM) and an Output Control Register (OCR), which is used to indicate the value in current run. Fig. 3 shows the 2-V Golomb code decoder architecture.  $T_{E\_in}$  is the encoded test vectors sent by ATE, and  $T_{D\_out}$  is the test data to be applied to cores. Two handshaking signals  $rdy$  and  $valid$  are provided:  $rdy$  indicates whether the decoder is ready to accept next data from ATE, and  $valid$  indicates when the output is valid. The OCR is a 1-bit register indicating whether the current run is 0 or 1. The FSM output signal  $inv$ , when it is set to 1, informs the OCR that current run is finished, and  $T_{D\_out}$  should be inverted for next run. The signal  $reset$  initializes the output of OCR to the first bit value in  $T_D$ , and set the signal  $inv$  to 0. Since it is always possible to select a test pattern starting with 0 as the first pattern in  $T_D$ , signal  $reset$  may only need to reset the OCR. A  $\log_2 m$ -bit counter is employed to count down from  $m - 1$  to 0 whenever the Group Prefix is a 1. The signal  $inc$  is used to restart the  $\log_2 m$ -bit counter and  $cf$  indicates that  $\log_2 m$ -bit counter has finished counting. The decoder works as follows.

- [Run Prefix] Whenever the input  $T_{E\_in}$  is 1, the counter counts  $m$  cycles. The signal  $rdy$  is low while the counter is counting, and it enables the input  $T_{E\_in}$  at the end of  $m$  cycles to accept another bit. The FSM output signal  $inv$  is set to 0 for  $m$  cycles, since

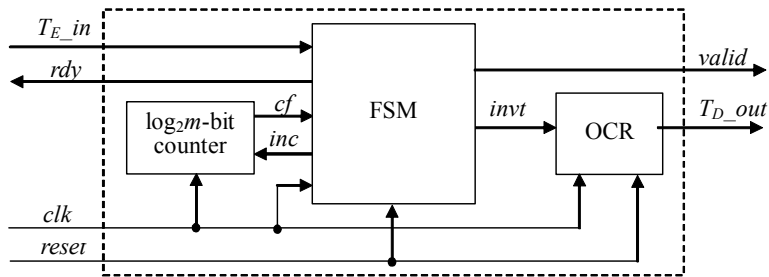


Fig. 3. The two-value Golomb code decoder.

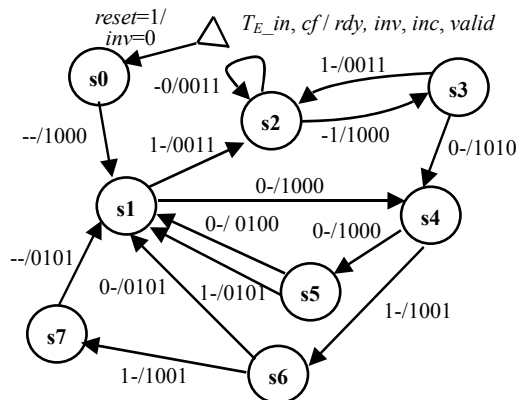


Fig. 4. The state transition diagram of the FSM.

these  $m$  bits belong to the same run. During this process, the FSM makes the valid signal *valid* high.

- [Run Tail] When the input  $T_E\_in$  is 0, the FSM starts decoding the tail of the input codeword. Depending on the Tail bits, the number of 0's sent to  $T_D\_out$  is different. The *rdy* and *valid* signals are used for handshaking.

The state diagram of the FSM in the decoder is given in Fig. 4. The states  $s1$  to  $s3$  correspond to prefix decoding, while states  $s4$  to  $s7$  corresponds to tail decoding. We synthesize the FSM with the TSMC  $0.35\mu\text{m}$  technology, and the results are shown in Table 1. The decoder statistics for Golomb encoding FSM [7] is also shown in Table 1 for comparison. The size of the proposed decoder is almost the same as Ref. [7]. In other words, the proposed method achieves higher compression rate with only a little hardware overhead.

**Table 1. Comparison of decoder hardware overhead.**

Item	Port	Nets	Cells	Reference	Area
Golomb code [7]	8	39	34	14	59.49
2-V Golomb code	8	39	34	16	67.83

## 5. EXPERIMENTAL RESULTS

We experiment the proposed method with some ISCAS benchmark circuits, and the results are shown in this section. It is assumed that a single scan chain is used. The FBI algorithm is used to invert some bits in the test vectors so that the run lengths can be maximized. As a result, the proposed method achieves better compression rate. The test vectors used in the experiments were generated by MinTest [4]. MinTest is by far the best combinational test compaction method in the literature. Table 2 illustrated the comparison between the proposed method and Golomb coding [7]. The first column gives circuit names and columns 2 to 5 show results of Golomb encoding [7]. The column under  $m$  is the Golomb code's divider. Column 5 ( $CR$ ) is the compression rate in percentage. Columns 6 to 12 show our results, which are obtained by the proposed method with FBI and without FBI (NFBI) algorithm plus 2-V Golomb coding. Column 13 compares FBI and NFBI. The results show that the FBI significantly improves the compression rate. On average, the improvement is 66.22%. Column 14 compares the amount of compressed data produced by our method with the results from [7]. Obviously, the proposed method achieves better compression results in most cases. However, in some circuits (*e.g.*, C1908), the FBI cannot reach good results. The problem mainly comes from the heuristic of bit selection in FBI.

**Table 2. Experimental results: ISCAS'85 using MinTest test patterns with FBI & NFBI.**

Circuit	Golomb coding [7]				Proposed method								Comparison ( $T_E$ )
	$T_D$	$m$	$T_E$	$CR(\%)$	$T_D$	FBI + 2-V (Deterministic)			NFBI + 2-V (Don't-care)			FBI/ NFBI (%)	
						$m$	# $T_E$	$CR(\%)$	$m$	$T_E$	$CR(\%)$		
C1355	4828	4	2627	45.59	3444	4	2626	23.75	4	2781	19.25	94.43	99.96
C1908	4587	4	2876	37.30	3498	2	3818	- 9.15	2	4711	- 34.68	81.04	132.75
C2670	20271	8	8903	56.08	10252	8	4567	55.45	2	13275	- 29.49	34.40	51.30
C3540	6450	4	4846	24.87	4200	4	3058	27.19	2	5560	- 32.38	55.00	65.82
C5315	15486	4	9443	39.02	6586	4	5417	17.75	2	8684	- 31.86	62.38	57.37
C7552	25254	2	21338	15.51	15111	4	9217	39.00	2	18815	- 24.51	48.99	43.20

Table 3 shows the results for ISCAS'89 circuits, and compares them with both Golomb code [7] and alternating run-length code [9]. Since the proposed method starts with MinTest patterns, our test data volumes ( $T_D$ ) are smaller in all cases. In the last two columns we compare the encoded test data volumes ( $T_E$ ) of the proposed method (P) with respect to Golomb [7] and alternating run-length [9] codes, and the results show that the proposed method achieves even better results.

## 6. CONCLUSION

In this paper, a two-value Golomb coding method is proposed to encode test patterns. This method, when combined with the FBI algorithm, achieves better compression rate

**Table 3. Experimental results: ISCAS'89 using MinTest test patterns.**

Circuit	Golomb coding [7]				Alternating RLE code [9]			Proposed method				Comparison ( $T_E$ )	
	$T_D$	$m$	$T_E$	CR (%)	$T_D$	$T_E$	CR (%)	$T_D$	$m$	$T_E$	CR (%)	P/[7] (%)	P/[9] (%)
S641	1404	2	1098	21.79	–	–	–	1155	8	320	72.29	29.14	–
S713	1404	4	1080	23.08	–	–	–	1155	4	447	61.30	41.39	–
S1196	4448	4	2570	42.22	–	–	–	3729	4	1120	69.97	43.58	–
S1238	4864	4	2685	44.80	–	–	–	3993	4	1180	70.45	43.95	–
S5378	23754	4	14085	40.70	23754	11694	49.23	20758	16	4601	77.84	32.67	39.34
S9234	39273	4	22250	43.35	39273	21612	55.03	25935	16	2462	90.51	11.07	11.39
S13207	165200	16	41658	74.78	165200	32648	19.76	163100	64	10794	93.38	25.91	33.06
S15850	76986	4	40717	47.11	76986	26306	34.17	57434	32	4566	92.05	11.21	17.36
S35932	4007299	32	59573	98.51	–	–	–	19393	8	11637	39.99	19.53	–
S38417	164736	4	92054	44.12	164736	64976	39.44	113752	16	42421	62.71	46.08	65.29
S38584	199104	4	104111	47.71	199104	77372	38.86	161040	16	37951	76.43	36.45	49.05

than other compression methods. Besides, the volume of decompressed data is always smaller. As a result, test application time can be minimized, too. A decoder for this scheme was presented, and the hardware overhead is relatively small. Experimental results show that the proposed method requires less than half of the test pattern length and it gives better results than one-value Golomb coding [7] and alternating run-length [9] codes.

## REFERENCES

1. IEEE 1500 Standard for Embedded Core Test, <http://grouper.ieee.org/groups/1500>.
2. Y. Zorian, S. Dey, and M. Rodgers, "Test of future system-on-chips," in *Proceedings of International Conference Computer-Aided Design*, 2000, pp. 392-398.
3. J. S. Chang and C. S. Lin, "Test set compaction for combinational circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 14, 1995, pp. 1370-1378.
4. P. Hamzaoglu and J. H. Patel, "Test set compaction for combinational circuits," in *Proceedings of International Conference Computer-Aided Design*, 1998, pp. 283-289.
5. S. J. Wang and S. N. Chiou, "Generating efficient tests for continuous scan," in *Proceedings of Design Automation Conference*, 2001, pp. 162-165.
6. K. J. Lee, J. J. Chen, and C. H. Huang, "Using a single input to support multiple scan chains," in *Proceedings of International Conference Computer-Aided Design*, 1998, pp. 74-78.
7. S. W. Golomb, "Run-length encoding," *IEEE Transactions on Information Theory*, Vol. 12, 1996, pp. 399-401.
8. A. Chandra and K. Chakrabarty, "Test data compression for system-on-a-chip using Golomb codes," in *Proceedings of IEEE VLSI Test Symposium*, 2000, pp. 113-120.
9. A. Chandra and K. Chakrabarty, "A unified approach to reduce SoC test data vol-

- ume, scan power, and testing time,” *IEEE Transactions on Computer-Aided Design*, Vol. 22, 2003, pp. 352-363.
10. A. Chandra and K. Chakrabarty, “Frequency directed run-length (FDR) codes with application to system-on-chip test data compression,” in *Proceedings of IEEE VLSI Test Symposium*, 2001, pp. 42-47.
  11. H. Kobayashi and L. R. Bahl, “Image data compression by predictive coding, part I: prediction algorithm,” *IBM Journal of Research and Development*, Vol. 18, 1974, pp. 164-171.

**Po-Chang Tsai (蔡栢樟)** received a B.S. degree in Industrial Education from the National Taiwan Normal University, Taiwan, R.O.C. in 1984, and where he is currently a Ph.D. candidate, in Institute of Computer Science, National Chung Hsing University. His research interests include VLSI design, digital testing, computer architecture, and data compression.

**Sying-Jyan Wang (王行健)** received a B.S. degree in Electrical Engineering from the National Taiwan University, Taiwan, R.O.C. in 1984, and a Ph.D. degree in Electrical Engineering from Princeton University in 1992. From 1989 to 1990 he was with AT&T Bell Laboratories, Holmdel, New Jersey. In 1992, he joined the department of Computer Science, National Chung Hsing University, Taiwan, where he is a now Professor. His research interests include VLSI design, digital testing, computer architecture, and fault-tolerant computing.

**Ching-Hung Lin (林環鴻)** received the M.S. degree in Computer Science from the National Chung Hsing University, Taiwan, R.O.C., in 2002. In 2006, he joined the Genesys Logic, Inc. (GLI), Hsinchu, Taiwan, R.O.C. His research interests include design-for-testability, synthesis for testability, and data compression.

**Tung-Hua Yeh (葉東樺)** received the B.E. degree in Industrial Education from National Taiwan Normal University, Taiwan, R.O.C. in 2000, and the M.S. degree in Computer Science from the National Chung Hsing University, Taiwan, R.O.C., in 2003, where he is currently pursuing the Ph.D. degree in Computer Science. His research interests include design-for-testability, high-level synthesis, synthesis for testability, and data compression.