JOURNAL OF ELECTRONIC TESTING: Theory and Applications 20, 667–670, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The United States.

On Using Exponential-Golomb Codes and Subexponential Codes for System-on-a-Chip Test Data Compression*

LEI LI AND KRISHNENDU CHAKRABARTY

Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA

ll@ee.duke.edu

krish@ee.duke.edu

Received September 29, 2003; Revised February 15, 2004

Editor: K.K. Saluja

Abstract. We examine the use of exponential-Golomb codes and subexponential codes can be used for the compression of scan test data in core-based system-on-a-chip (SOC) designs. These codes are well-known in the data compression domain but their application to SOC testing has not been explored before. We show that these codes often provide slighly higher compression than alternative methods that have been proposed recently.

Keywords: compression codes, embedded core testing, SOC testing, test data volume

1. Introduction

Test data volume is a major challenge faced in the testing of core-based system-on-a-chip (SOC) integrated circuits. New techniques are needed to reduce test data volume and testing time, and to overcome ATE memory limitations. In order to be applicable to intellectual property cores, these techniques should be based on the reuse of precomputed tests; they should not require structural models for additional fault simulation or test generation.

One popular approach for reducing test data volume is based on the compression of the precomputed test set T_D provided by the core vendor to a much smaller test set T_E, which is stored in ATE memory. An on-chip decoder is used for pattern decompression to generate T_D from T_E during pattern application [3–5, 7]. These techniques are typically based on statistical codes, run-length codes, and their variants, e.g. Golomb and FDR codes. Test data volume reduction techniques based on on-chip pattern decompression are also presented in [1, 2, 8, 9, 11, 14]. Although all these techniques aim to reduce test data volume, the different approaches present different design alternatives and their applicability to a particular design varies from case to case.

In this short paper, we investigate the use of exponential-Golomb codes and subexponential codes for compressing scan test data. These codes are well-known in the data compression domain [12], but their application to SOC testing has not been explored before. These codes often provide slightly higher compression of test data than other codes proposed for test data compression. Moreover, only a small amount of hardware is required for on-chip decompression of test data encoded using exponential-Golomb and subexponential codes (The decompression logic synthesized using Synopsys tools requires less than 50 gates and less than 100 gates, respectively). The decompression logic is independent of the core under test and the test set. The proposed compression/decompression scheme requires no modifications to the core under test.

*This research was supported in part by the National Science Foundation under grants CCR-9875324 and CCR-0204077, and by a graduate fellowship from the IEEE/ACM Design Automation Conference.

2. Exponential-Golomb and Subexponential Codes

In the following description, we assume that the pre-computed SOC test data consists of n test patterns $t_1, t_2, \ldots, t_n$. The don't-care bits in the test set are set to 0s. The test patterns are suitably reordered and serialized before compression. The reordered and serialized test set is denoted as T_D, which is then compressed using these codes.

2.1. Exponential-Golomb Code

The exponential-Golomb code provides a variable-to-variable encoding method, i.e., the runs of 0s are encoded as variable-length codewords. A run of 0s is divided into successive sub-runs of length 2^k, 2^{k+1}, $2^{k+2}, \ldots, 2^{k+i-1}$, until the number of remaining 0s is less than 2^{k+i}, where k is the code parameter of the exponential-Golomb code. The rest of the run is encoded as a $(k+i)$-bit binary number. The power of the exponential-Golomb code lies in the fact that it can encode both short and long runs efficiently because the successive sub-runs grow exponentially. Let l denote the run-length to be encoded. The encoding steps are as follows.

1. Determine i such that $\sum_{j=0}^{i-1} 2^{j+k} \leq l < \sum_{j=0}^{i} 2^{j+k}$, $i \geq 0$.
2. Form the prefix of i 1s.
3. Insert the separator 0.
4. Form the tail: express the value of $(l - \sum_{j=0}^{i-1} 2^{j+k})$ as a $(k+i)$-bit binary number.

Table 1 shows the codewords of the exponential-Golomb code for run-lengths varying from 0 to 10 with code parameter $k = 0$, 1 and 2. Note that the FDR code described in [3] corresponds to a special case of the exponential-Golomb code ($k = 1$). In the table, we have separated the codewords into groups. The group index i is also the number of 1s in the prefix. From the table, we note the following properties of the exponential-Golomb code:

1. The length of prefix in group A_i is i (excluding the separator 0).
2. In group A_i, the run-length represented by the prefix is $l_{\text{prefix}} = \sum_{j=0}^{i-1} 2^{k+j} = 2^k(2^i - 1)$, which is the value of the continuous 1s in the prefix shifted left by k bits.
3. The length of tail in group A_i is $(k+i)$.
4. In group A_i, the run-length represented by the tail is between 0 to $2^{k+i} - 1$.
5. The size of group A_i is 2^{k+1}.

Table 1. Codewords of the exponential-Golomb code.

	$k=0$		$k=1$ (FDR)		$k=2$	
Run-length	i	Codeword	i	Codeword	i	Codeword
0	0	0	0	00	0	000
1	1	100		01		001
2		101	1	1000		010
3	2	11000		1001		011
4		11001		1010	1	10000
5		11010		1011		10001
6		11011	2	110000		10010
7	3	1110000		110001		10011
8		1110001		110010		10100
9		1110010		110011		10101
10		1110011		110100		10110

Based on these properties, it is straightforward to derive the decoding algorithm as follows:

1. Let i be the number of leading 1s (prefix) in the codeword.
2. Form a run of 0s of length $\sum_{j=0}^{i-1} 2^{j+k}$.
3. Skip the next 0 (separator).
4. The next $(k+i)$ bits make up the tail. Form a run of 0s of length represented by the tail.
5. Append 1 to the run of 0s.
6. Go to step 1 to process the next codeword.

2.2. Subexponential Code

As in the case of the exponential-Golomb code, the codewords of the subexponential code can also be divided into groups and we use the number of 1s in the prefix as the group index. Table 2 shows thecodewords of the subexponential code for run-lengths varying from 0 to 10 with the code parameter $k = 0, 1, 2$. From the table, we can find that the size of group A_0 is 2^k. For $i \geq 1$, the size of group A_i is $2^{(k+i-1)}$. This property is closely tied to the encoding procedure, which is described as follows. Let l denote the run-length to be encoded.

Table 2. Codewords of the subexponential code.

	$k=0$		$k=1$		$k=2$	
Run-length	i	Codeword	i	Codeword	i	Codeword
0	0	0	0	00	0	000
1	1	10		01		001
2	2	1100	1	100		010
3		1101		101		011
4	3	111000	2	11000	1	1000
5		111001		11001		1001
6		111010		11010		1010
7		111011		11011		1011
8	4	11110000	3	1110000	2	110000
9		11110001		1110001		110001
10		11110010		1110010		110010

1. Determine the group index i using the following rules:
   - if $l < 2^k$, then $i = 0$.
   - if $l \geq 2^k$, then determine i such that $2^{i+k-1} \leq l < 2^{i+k}$.
2. Form the prefix of i 1s.
3. Insert the separator 0.
4. Form the tail: express the value of $(l - 2^{i+k-1})$ as a $(i + k - 1)$-bit binary number.

We can see that A_0 is a special group in the subexponential code. Its size is the same as that of group A_1. This property makes the encoding and decoding for the subexponential code a little more complex than that for the exponential-Golomb code. A few additional steps are needed during encoding and decoding. The decoding procedure for the Subexponential code is as follows.

1. Let i be the number of leading 1s (prefix) in the codeword.
2. Form a run of 0s of length

$$\begin{cases} 0, & \text{if } i = 0 \\ 2^{i+k-1}, & \text{otherwise} \end{cases}$$

3. Skip the next 0 (separator).
4. Compute the length of the tail, c_{tail} as

$$\begin{cases} k, & \text{if } i = 0 \\ k + i - 1, & \text{if } i \geq 1 \end{cases}$$

5. The next c_{tail} bits are the tail. Form a run of 0s of length represented by the tail.
6. Append 1 to the run of 0s.
7. Go to step 1 to process the next codeword.

3. Results and Conclusions

In this section, we apply the exponential-Golomb and subexponential codes with various code parameter values to the cubes of the six larger ISCAS-89 circuits. These test cubes were obtained from the Mintest ATPG program [6]. Prior to the compression, the don't-care bits in the test set are set to 0s and the test patterns are suitably reordered and serialized. We refer to the pre-encoded test set as T_D and the encoded test set as T_E. Table 3 shows acomparison of the compression results with FDR coding [3], RESPIN++ [13], and test data mutation encoding [10]. We consider three different values for the code parameter: $k = 0, 1, 2$.

Table 3. Comparison of compression results for T_D.

		Size of T_E (bits)							
		Exponential-Golomb			Subexponential				
Circuit	Size of T_D (bits)	$k=0$	$k=1$ (FDR)	$k=2$	$k=0$	$k=1$	$k=2$	RESPIN++ [13]	Mutation encoding [10]
s9234	39273	23809	22076	21799	25039	22413	21210	17198	N/A
s13207	165200	33061	30804	29727	35102	32285	30520	26004	74423
s15850	76986	27342	25902	25710	29079	26670	25650	32226	26021
s35932	28208	13857	22744	32063	13924	22907	32126	N/A	7222
s38417	164736	80264	93444	108510	83209	93317	109266	89132	45003
s38584	199104	81251	77754	78079	85650	79518	78234	63232	73464

Table 4. Comparison of compression results for T_{diff}.

	Compression (percent)						
	Exponential-Golomb			Subexponential			
Circuit	$k = 0$	$k = 1$ (FDR)	$k = 2$	$k = 0$	$k = 1$	$k = 2$	VIHC [5]
s9234	58.01	60.63	61.57	55.24	59.38	61.46	54.84
s13207	86.81	87.47	87.70	86.06	87.02	87.45	83.21
S15850	70.76	71.96	71.97	69.09	71.17	71.72	60.68
s35932	54.29	25.74	−3.84	53.94	25.26	−4.00	66.47
s38417	65.71	65.36	64.52	63.27	64.97	64.04	54.51
s38584	63.33	64.68	64.17	61.48	63.91	64.10	56.97

We find that the exponential-Golomb code and the subexponential code outperform the FDR code for five out of the six circuits. Compared to RESPIN++, the test data volume is less for only one out of five circuits. Note however that the results presented in [13] were obtained using a pre-processing step of 400 pseudo-random patterns. As a result, the deterministic pattern used in [13] targeted a smaller number of faults than in this work. Moreover, sufficient structural knowledge of the circuit under test was assumed in [13] for the purpose of fault simulation. Compared to mutation encoding, we obtain lower test data volume in two out of five cores. The increased compression for three circuits in [10] is accompanied by the need for scan chain reorganization. Note that significantly higher compression was obtained in [10] in a second set of experiments using different precomputed test sets. We did not consider these test sets in our experiments hence a direct comparison is difficult in this case. Even higher compression was achieved in [10] using a combination of mutation encoding and test compaction; we do not consider this approach here since it requires structural information about the circuit under test.

Finally, we compare the proposed coding technique to the variable-length input Huffman coding (VIHC) presented in [5]. Since the result in [5] are presented for "difference vector" test sets T_{diff} derived from the Mintest test cubes, we present the comparison here using T_{diff}. Table 4 shows thatexponential-Golomb and subexponential codes outperform VIHC for five out of the six cases.

References

1. C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, "OPMISR: The Foundation for Compressed ATPG Vectors," in *Proc. Int. Test Conf.*, 2001, pp. 748–757.
2. I. Bayraktaroglu and A. Orailoglu, "Test Volume and Application Time Reduction Through Scan Chain Concealment," in *Proc. ACM/IEEE Design Automation Conf.*, 2001, pp. 151–155.
3. 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 Transactions on Computers*, vol. 52, pp. 1076–1088, 2003.
4. A. Chandra and K. Chakrabarty, "System-on-a-Chip Test Data Compression and Decompression Architectures Based on Golomb Codes," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 355–368, March 2001.
5. 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," in *Proc. Design, Automation and Test in Europe Conf.*, 2002, pp. 604–611.
6. I. Hamzaoglu and J.H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 283–289.
7. A. Jas and N.A. Touba, "Test Vector Decompression Via Cyclical Scan Chains and its Application to Testing Core-Based Design," in *Proc. Int. Test Conf.*, 1998, pp. 458–464.
8. A. Jas, J. Ghosh-Dastidar, and N.A. Touba, "Scan Vector Compression/Decompression Using Statistical coding," in *Proc. VLSI Test Symp.*, 1999, pp. 114–120.
9. A. Khoche, E. Volkerink, J. Rivoir, and S. Mitra, "Test Vector Compression Using EDA-ATE Synergies," in *Proc. VLSI Test Symp.*, 2002, pp. 97–102.
10. S. Reda and A. Orailoglu, "Reducing Test Application Time Through Test Data Mutation Encoding," in *Proc. Design, Automation and Test in Europe Conf.*, 2002, pp. 387–393.
11. S.M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On Test Data Volume Reduction for Multiple Scan Chain Design," in *Proc. VLSI Test Symp.*, 2002, pp. 103–108.
12. D. Salomon, *Data Compression: The Complete Reference*, New York, NY: Springer-Verlag New York, Inc., 2000.
13. L. Schafer, R. Dorsch, and H. Wunderlich, "RESPIN++ - Deterministic Embedded Test," *Proc. European Test Workshop*, 2002, pp. 37–44.
14. F.G. Wolff and C. Papachristou, "Multiscan-Based Test Compression and Hardware Decompression Using LZ77," in *Proc. Int. Test Conf.*, 2002, pp. 331–339.