

Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets¹

Vikram Iyengar[†], Krishnendu Chakrabarty[†] and Brian T. Murray[‡]

[†]Dept. of Electrical and Computer Engineering, Boston University, Boston, MA 02215

[‡]General Motors R&D Center, 30500 Mound Rd, Warren, MI 48090-9055

Abstract

We present a new approach to built-in self-test of sequential circuits using precomputed test sets. Our approach is especially suited to circuits containing a large number of flip-flops but few primary inputs. Such circuits are often encountered as embedded cores and filters for digital signal processing, and are inherently difficult to test. We show that statistical encoding of test sets can be combined with low-cost pattern decoding for deterministic BIST. This approach exploits recent advances in sequential circuit ATPG and, unlike other BIST schemes, does not require access to gate-level models of the circuit under test. Experimental results show that the proposed method provides higher fault coverage than pseudorandom testing with shorter test application time.

1 Introduction

Built-in self testing (BIST) offers an attractive solution to the problem of testing complex ICs [10]. The design of test generator circuits (TGCs) for BIST has been studied widely and several TGC design techniques have been developed for combinational and full-scan sequential circuits. However, these techniques are not directly applicable to non-scan and partial-scan sequential circuits. Many performance-driven designs and embedded-core circuits do not use full scan. In fact, hundreds of non-scan “legacy cores” exist today [7]. Testing them using known BIST methods requires substantial redesign to “stitch in” scan chains.

A few BIST techniques for non-scan sequential circuits have been proposed recently [9, 11, 12]. These methods are based on pseudorandom test sequences and do not always yield the same fault coverage as deterministic sequences. They also require excessively high test application times. Moreover, they require a gate-level circuit model, either for modifying the circuit flip-flops [9, 11] or for carrying out fault simulation [11, 12]. Such gate-level models may not be readily available for embedded cores.

We present a new approach for designing TGCs that apply precomputed test sets to non-scan and partial scan circuits. Such test sets do not disclose substantial proprietary information and hence may be easily provided by core vendors. The precomputed test sets can be obtained using ATPG tools tailored to any fault model. Recent advances in ATPG have led to techniques and tools that provide such test sets for single stuck-line faults with high fault coverage [3, 6, 8].

A straightforward TGC design for sequential circuits involves the use of a ROM to store the precomputed test set T_D . However, this is not considered practical because of the silicon area required to store T_D . A more practical alternative is to encode T_D and store (or generate) only the compressed (encoded) test set T_E , which can then be decoded during test application. A generic TGC that employs encoding is shown in Figure 1. The sequence generator SG feeds a k -bit-wide sequence of compressed (encoded) patterns to a decoder circuit DC that expands (decodes) these into n -bit-wide test patterns, where n is the number of controllable primary inputs in the circuit under test (CUT). This decomposition of the TGC is similar to that described in [2] for combinational and full-scan circuits; however, the TGC must also meet some ordering requirements. In this paper, to simplify analysis, we require that the order of patterns in T_D must be preserved exactly.

In the absence of any knowledge about the structure of the test sequence, we can encode it with fixed-length codes for each test pattern. Let the number of unique test patterns in the test set be m . Every pattern can then be encoded with $q = \lceil \log_2 m \rceil$ bits. The sequence generator in this case can be a ROM that stores the encoded test set T_E . However, this approach does not account for the fact that some patterns occur more often than others. This observation allows us to compact T_D more and decode it with lesser hardware.

Consider, for example, T_D for the s444 ISCAS 89 benchmark [1] obtained from the Gentest ATPG program [3]. It contains 1881 patterns, of which only 8 are unique. Such test sets are good candidates for statistical encoding, in which variable-length codewords are used to encode the test patterns, more frequent patterns being encoded with fewer bits. We present here a deterministic BIST approach

¹This research was supported in part by a contract from General Motors Corporation and by a start-up grant from Boston University's College of Engineering.

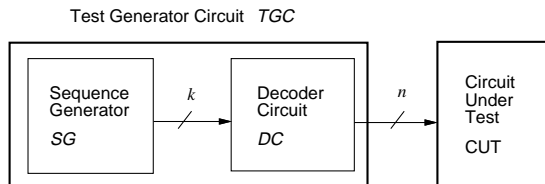


Figure 1: A generic BIST test generator circuit.

based on statistical encoding that exploits this feature of sequential circuit test sets. It is especially useful for circuits such as filters, used in digital signal processing (DSP), that have few primary inputs but a large number of flip-flops. These circuits, which require long test sequences, are typical of embedded cores used in complex DSP designs [13]. A number of ISCAS 89 benchmark circuits also belong to this category. Full scan is often not practical for these circuits because of the relatively large number of flip-flops; the overhead is substantial and there are few combinational gates to test between scan chains.

The paper is organized as follows. In Section 2, we briefly review statistical encoding and motivate its use for encoding a given precomputed test set T_D . In Section 3, we describe two specific statistical encoding techniques: Huffman and Comma codes. We also describe the use of run-length encoding, a technique for compressing an already encoded test set further. Section 4 presents experimental results on test set compression and decoder design for the ISCAS 89 circuits. Our results demonstrate that statistical encoding of test patterns can lead to practical deterministic TGCs for sequential circuits.

2 Statistical encoding

Statistical encoding has been widely employed in digital communication and in such computer applications as instruction encoding, but not typically in BIST because deterministic testing is widely assumed to be too costly. Examples of statistical encoding methods include Huffman coding, Shannon-Fano coding and Lempel-Ziv string encoding [4]. We review Huffman and Comma coding here, since these seem to be particularly useful for encoding T_D . We also compare these codes to fixed-length codes, to more precisely analyze the advantages of statistical encoding.

2.1 Optimal encoding: Huffman codes

An optimal encoding technique minimizes the average length of a codeword. Consider a test set T_D containing m unique patterns X_1, \dots, X_m with probabilities of occurrence p_1, \dots, p_m , respectively. (These probabilities are obtained as normalized frequency values). Its entropy, defined intuitively as the minimum average number of bits required to represent a pattern, is given by $H(T_D) = -\sum_{i=1}^m p_i \log_2 p_i$. Therefore, an optimal encoding is one in which the average number of bits needed to represent a pattern is closest to the entropy. The Huffman code is provably optimal since it results in the shortest average codeword length among all statistical encoding techniques that assign a unique binary codeword to each

Unique patterns	Occurrences	Probability of occurrence	Huffman codeword	Comma codeword
X_1 : 0000	45	0.5625	0	0
X_2 : 0101	15	0.1875	10	10
X_3 : 1010	15	0.1875	110	110
X_4 : 1111	5	0.0625	111	1110

Table 1: Test set encoding: an example.

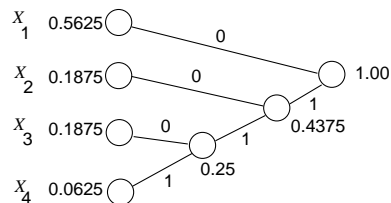


Figure 2: Huffman tree for the example in Table 1.

pattern [4]. In addition, Huffman codes possess the prefix-free property, i.e., no codeword is the prefix of a longer codeword. This is especially important for decoding.

Table 1 illustrates the Huffman code for an example test set T_D with four unique patterns out of a total of eighty. Note that the most common pattern X_1 is encoded with a single 0 bit; that is $e(X_1) = 0$, where $e(X_1)$ is the codeword for X_1 . Since no codeword appears as a prefix of a larger codeword (the prefix-free property), if a sequence of coded test vectors is treated as a serial bit-stream, they can each be decoded as soon as the last bit of the codeword is read. This property is essential since variable-length codewords cannot be read from memory as words in the usual fashion.

The Huffman code illustrated in Table 1 can be constructed by generating a binary tree (Huffman tree) with edges labeled either 0 or 1 as illustrated in Figure 2. Each codeword $e(X_i)$ is obtained by traversing the path from the root to the corresponding leaf node v_i . The average number of bits per pattern l_H (average length of a codeword) is given by $l_H = \sum_{i=1}^m w_i p_i$, where w_i is the length of the codeword corresponding to X_i .

We next compare Huffman coding with equal-length coding. Let l_H (l_E) be the average length of a codeword for Huffman coding (equal-length coding). Since Huffman coding is optimal, it is clear that $l_H \leq l_E$. If all unique patterns have the same probability of occurrence and the number of unique patterns m is a power of 2, then $l_H = l_E$. Furthermore, if the Huffman tree is a full binary tree, then $l_H = l_E$. These conditions are sufficient but not necessary for l_H to equal l_E . For example, if $m = 4$, $p_1 = p_2 = 1/3$, and $p_3 = p_4 = 1/6$, then $l = l_E$ even though the corresponding Huffman tree is not necessarily a full binary tree.

Huffman encoding is therefore less useful when the probabilities of all of the unique test patterns are similar. This tends to happen when the ratio of the number of flip-flops to the number of primary inputs in the CUT, denoted γ in Section 4, is low.

Test pattern	Occurrences	Occurrence of probability	Codeword	
			Huffman	Comma
000	1631	0.8671	0	0
010	139	0.0738	10	10
001	93	0.0494	110	110
011	7	0.0037	1110	1110
110	5	0.0026	11110	11110
101	3	0.0015	111110	111110
111	2	0.0011	1111110	1111110
100	1	0.0005	1111111	11111110

Table 2: Codewords for the s444 test set.

2.2 Comma codes

Although Huffman codes provide optimal test set compression, they do not always yield the lowest-cost decoder circuit. Therefore, we also employ a non-optimal code, namely the Comma code, which often leads to more efficient decoder circuits. The Comma code, also prefix-free, derives its name from the fact that it contains a terminating symbol, e.g. 0, at the end of each codeword.

The Comma encoding procedure sorts the unique patterns in decreasing order of probability of occurrence, and encodes the first pattern (i.e., the most probable pattern) as 0, the second as 10, the third as 110, and so on. The codeword for the i^{th} unique pattern X_i is thus given by a sequence of $(i - 1)$ 1s followed by a 0. Comma codewords for the unique patterns in the example test set of Section 2.1 are listed in Column 5 of Table 1. The Comma code requires a substantially simpler decoder than the Huffman code. Since each Comma codeword is essentially a sequence of 1s followed by a zero, the decoder only needs to maintain a count of the number of 1s received before a 0 signifies the end of a codeword. The 1s count can then be mapped to the corresponding test pattern.

For a given test set T_D with m unique patterns having probabilities of occurrence $p_1 \geq p_2 \geq \dots \geq p_m$, the average length of a Comma codeword is given by $l_C = \sum_{i=1}^m ip_i$. Since the code is non-optimal, $l_C \geq l_H$. However, the Comma code provides near-optimal compression, i.e., $\lim_{m \rightarrow \infty} (l_C - l_H) = 0$, if T_D satisfies certain properties. These hold for typical test sets that have a large number of repeated patterns. We first present the condition under which Comma codes are near-optimal and then the property of T_D required to satisfy the condition.

A binary tree with leaf nodes X_1, \dots, X_m is *skewed* if the distance d_i of X_i from the root is given by: $d_i = i$, $1 \leq i \leq m - 1$, and $d_m = m - 1$. For instance, the Huffman tree of Figure 2 is a skewed binary tree with four leaf nodes.

Theorem 1 *If the Huffman tree for T_D is skewed, then $l_C - l_H = p_m$ and $\lim_{m \rightarrow \infty} (l_C - l_H) = 0$.*

Theorem 2 *The Huffman tree for T_D is skewed if and only if, for $1 \leq i \leq m - 2$, the probabilities of occurrence satisfy the condition $p_i \geq \sum_{k=i+2}^m p_k$.*

Comma codes, being non-optimal, do not always yield better compression than equal-length codes. The following theorem establishes a sufficient condition under which Comma codes perform worse than equal-length codes.

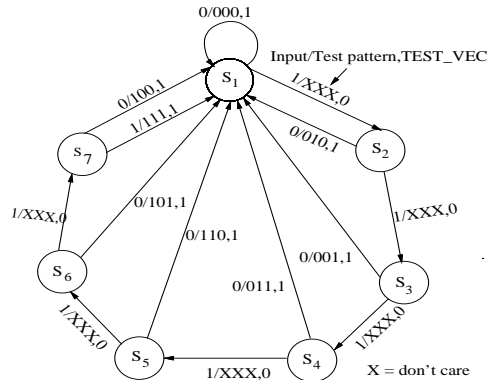


Figure 3: State transition diagram for the FSM decoder of s444.

Theorem 3 *If $p_m > 2 \lceil \log_2 m \rceil / (m(m+1))$, then $l_C > l_E$, where l_C (l_E) is the average codeword length for Comma (equal-length) coding.*

The condition of Theorem 2 appears to be easy to satisfy in most cases. The probabilities of occurrence of patterns for a typical case (the s444 test set) are shown in Table 2. The decrease in compression resulting from the use of Comma codes, instead of optimal (Huffman) codes to compress such test sets, which is given by $l_C - l_H = p_m$ from Theorem 1, is extremely small in practice. For the s444 test set, $p_m = 0.0005$, $l_H = 1.2121$, and therefore $l_C = 1.2126$, and the compression loss is only one bit. Therefore, both Huffman and Comma codes can efficiently encode sequential circuit test sets.

3 Test generator circuits

In this section, we illustrate our methods for designing TGCs. We illustrate the steps involved in encoding and decoding with the test set for the s444 benchmark circuit as an example.

3.1 Huffman encoding

The first step is to identify the unique patterns in the test set. A codeword is then developed for each unique pattern using the Huffman code construction method outlined in Section 2. The unique test patterns and the corresponding codewords for s444 are listed in Table 2. The original (unencoded) test set T_D , which contains 1881 test patterns of 3 bits each, requires 5643 bits for storage. On the other hand, the encoded test set T_E has only 1.2121 bits per codeword, and hence requires only 2280 bits of memory. Therefore, Huffman encoding of T_D leads to 59.59% saving in storage, while both the order as well as the contiguity of test patterns are preserved.

The encoded test set T_E is stored on-chip and read out one bit at a time during test application. The sequence generator SG of Figure 1 is therefore a ROM that stores T_E . The patterns in T_D can be obtained by decoding T_E using a simple finite-state machine (FSM).

Figure 3 shows the state transition diagram of the FSM for s444. The number of states is equal to the number of

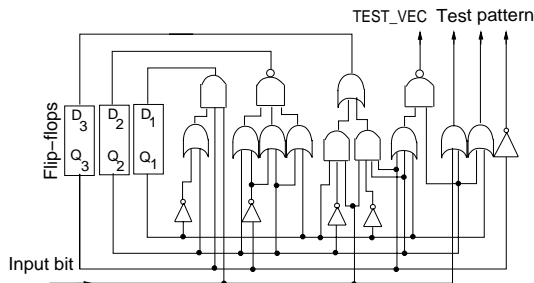


Figure 4: Gate-level netlist of the decoder for s444.

non-leaf nodes in the corresponding Huffman tree. For example, the Huffman tree for the s444 test set has seven non-leaf nodes, hence the corresponding FSM of Figure 3 has seven states— S_1, \dots, S_7 . The FSM receives a single-bit input from SG , and produces n -bit-wide test patterns, as well as a single-bit control output $TEST_VEC$. The latter is enabled only when a valid test pattern for the CUT is generated by the decoder—this happens whenever a transition is made to state S_1 . The test application time t is increased, however, since the decoder examines only one bit of T_E in each clock cycle. Fortunately, the increase in t is directly related to the amount of test set compression achieved. In fact, we can show that the test application time increases by a factor l_H , the average length of a Huffman codeword. Experimental results on test set compression in Section 4 show that the average length of a Huffman codeword for typical test sets is less than 2. This implies that the test application time rarely doubles. Since test sets are deterministic, and test patterns are applied at-speed in a BIST environment, this increase in testing time can be tolerated.

Figure 4 shows the netlist of the decoder circuit for s444. The design is simplified considerably by the presence of a large number of don't-cares in the decoder specification, which a CAD tool can exploit for optimization. Note that the overhead required by the decoder and the ROM is substantially less than that required by a special-purpose FSM designed to produce the test sequence.

The cost of the on-chip decoder can be reduced by sharing it between multiple CUTs. In this case, we combine these multiple test sets to obtain a composite test set T'_D , and apply the encoding procedure to T'_D to obtain an encoded test set T'_E . The slight increase in the size of T'_E (compared to T_E) is offset by the hardware saving obtained by decoder sharing. This is illustrated in Section 4 for several benchmark circuits.

3.2 Comma encoding

We next describe TGC designs using Comma codes. Once again, we illustrate the encoding and decoding scheme using the s444 example. The Comma codewords generated for the unique patterns in the s444 test set are listed in Table 2. The probabilities of occurrence of test patterns clearly satisfy the condition in Theorem 1, therefore the encoding is near-optimal. The Comma-encoded

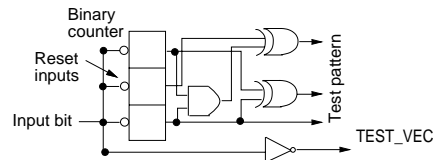


Figure 5: The s444 Comma pattern decoder.

Run-length	No. of runs		Run-length	No. of runs	
	0s	1s		0s	1s
1	90	114	5	8	0
2	52	104	6	10	0
3	33	10	7	37	0
4	11	18	8	145	0

Table 3: Runs in the Huffman encoded test set for s444.

test set has 1.2126 bits per codeword, and requires 2281 bits for storage, an increase of only one bit from the optimally (Huffman) encoded test set described in Section 3.1. Hence the reduction in test set compression arising from the use of Comma codes instead of Huffman codes for this example is only 0.02%.

The slight decrease in test set compression due to the use of the Comma code is offset by the reduced complexity of the pattern decoder. Figure 5 illustrates the decoder for the s444 circuit test set. It is constructed using a binary counter and combinational logic that maps the counter states to the test patterns. The test application scheme is the same as that for the Huffman decoder. As in the case of Huffman coding, the increase in testing time due to Comma coding equals the average length of a codeword.

3.3 Run-length encoding

Finally, we describe run-length encoding of T_E to achieve further compression. We exploit the fact that sequences of identical test patterns (runs) are common in T_D . For example, in the test set for s444, runs of the pattern 000 occur with lengths of up to 70. Huffman and comma encoding transform T_D to a compressed serial bit stream, and in the s444 test set, each occurrence of the test pattern 000 is replaced by a 0 (Table 2). Therefore, long runs of 0s are present in the statistically compressed bit stream, which can be further compressed using run-length coding.

Run-length coding is a data compression technique that replaces a sequence of identical symbols with a copy of the repeating symbol and the length of the sequence. For example, a run of 5 0s (00000) is encoded as (0,5). Run-length encoding has been used recently to reduce the time to download test sets to ATE across a network [14]. We improve upon the basic run-length encoding scheme by considering only those runs that have a substantial probability of occurrence in the statistically-encoded bit stream. A unique symbol representing a run of a particular length (and the corresponding bit) is then stored. The value of the repeating bit is generated from the bits representing the length of the run during decoding. We therefore obvi-

CUT	γ	$ T_D $	CUT	γ	$ T_D $
s1196	1.28	435	s400	7.00	2208
s1238	1.28	475	s420	0.84	166
s1423	4.35	150	s444	7.00	2240
s298	4.66	322	s526	7.00	2250
s344	1.66	127	s641	0.54	209
s349	1.66	134	s713	0.54	173
s35932	49.37	496	s820	0.625	1115
s382	7.00	2074	s838	0.94	26
s386	0.85	286	s953	1.81	13

Table 4: Values of γ and $|T_D|$ for ISCAS 89 circuits.

ate the need to store a copy of the repeating bit.

We describe our run-length encoding process using the s444 example. Analysis of its Huffman-encoded test set yields the distribution of runs shown in Table 3. Encoding all 16 runs would obviously be expensive (4 bits would be required for each run) since very few instances of (0,4), (0,5) and (1,3), and no instances of (1,5), (1,6), (1,7) or (1,8) exist. We therefore use combinations of 3 bits (000, . . . , 111) to encode the 8 most frequently occurring runs—(0,1), (0,2), (0,3), etc. The less-frequently occurring runs (0,4), (0,5), (0,6) and (1,3) are divided into smaller runs for encoding. The encoded runs are stored in a ROM and output to a run-length decoder. The run-length decoder provides a single bit in every clock cycle to the Huffman (or Comma) decoder for test application. It consists of a binary down counter, and a small amount of combinational logic. Since one bit is received by the Huffman decoder in every clock cycle, run-length encoding does not add to testing time.

4 Experimental results

In this section, we present experimental results on test set encoding for several ISCAS 89 benchmark circuits. We consider circuits in which the number of flip-flops f is considerably greater than the number of primary inputs n ; we denote the ratio f/n by γ . Table 4 lists the values of γ for the ISCAS 89 circuits—those having a high value of γ shown in bold. Such circuits are especially hard to test because of the relatively large number of internal states and few primary inputs. From Table 4, we see that these circuits typically require longer sequences of test patterns. On the other hand, they are excellent candidates for our encoding approach.

We performed experiments on test sets for SSL faults obtained from the Gentest ATPG program, as well as the HITEC and STRATEGATE test sets [8]. We measured the fault coverage of these test sets using the PROOFS fault simulator [8] and ensured that the coverage is comparable to the best-known fault coverage for these circuits.

Table 5 compares the number of bits required to store the encoded test set T_E with that required to store the corresponding unencoded test set T_D . The number of bits required by our scheme is moderate, substantially less than that required to store unencoded test sets. The saving

ISCAS circuit	Gentest						
	T_{bits}	l		H_{bits}	C_{bits}	Percentage compression	
		l_H	l_C			HE	CE
s298	486	1.79	1.79	290	291	40.32	40.12
s382	7389	1.20	1.20	2962	2963	59.91	59.89
s400	3846	1.41	1.41	1818	1819	52.73	52.70
s444	5643	1.21	1.21	2280	2281	59.59	59.57
s526	2262	1.90	1.90	1435	1437	36.60	36.47
s35932	3010	6.65	— ¹	572	—	81.00	—

ISCAS circuit	HITEC						
	T_{bits}	l		H_{bits}	C_{bits}	Percentage compression	
		l_H	l_C			HE	CE
s298	966	2.04	2.05	657	660	31.88	31.36
s382	6222	1.46	1.47	3028	3049	51.35	51.33
s400	6624	1.46	1.47	3224	3246	51.43	51.43
s444	6720	1.47	1.47	3293	3297	45.19	45.19
s526	6750	1.89	1.90	4253	4275	41.38	41.38
s35932	17K	8.97	— ¹	4449	—	74.38	—

T_{bits} : Total no. of bits in T_D ; H_{bits} (C_{bits}): No. of bits after Huffman (Comma) enc.; ¹Comma enc. is not feasible for this test set, because the probabilities do not satisfy the condition in Theorem 2; STRATEGATE results omitted for lack of space

Table 5: Test set compression for ISCAS 89 circuits.

in SG memory presented in Table 5 is substantial, and in most cases, Comma coding is as efficient as Huffman coding. In Table 6, we show that further compression is achieved by applying run-length coding to T_E . We present results on run-length encoding for the s382 and s444 circuits using the Gentest test set.

The test application time required is considerably less than that required for pseudorandom testing, even though the number of clock cycles \mathcal{C} is greater than the number of patterns in T_D ($\mathcal{C} = l_H |T_D|$ for Huffman coding). Table 7 compares the number of clock cycles required, and the fault coverage obtained for our method, with the corresponding figures reported recently for two pseudorandom testing schemes [11, 12]. The test application time required by our method is two orders of magnitude less than for the pseudorandom testing method of [11]. We also achieve higher fault coverage for all circuits.

We next present experimental results on the Huffman and Comma decoder implementations. We designed the FSM decoders using the Epoch CAD tool [5]. The low to moderate decoder costs in Table 8 show that the decoding algorithm can be easily implemented as a BIST scheme. Note that the largest benchmark circuit s35932 requires low overhead—0.53% for the ROM and 6.18% for the decoder; a special-purpose FSM for this circuit requires as much as 15.23% overhead. Decoder overhead also tends to decrease with an increase in γ , thus clearly demonstrating that the proposed test technique is well suited to circuits for which γ is high.

Finally, we present experimental results on TGC design when a single decoder is used to test several CUTs on a

ISCAS 89 circuit	T_{bits}	Number of bits in T_E			
		H_{bits}	C_{bits}	HR_{bits}	CR_{bits}
s382	7389	2962	2963	2268	2421
s444	5643	2280	2281	1953	2013

ISCAS 89 circuit	Percentage compression			
	HE	CE	HRE	CRE
s382	59.91	59.89	69.31	67.24
s444	59.59	59.57	65.39	64.33

HR_{bits} (CR_{bits}): No. of bits after Huffman (Comma) and run-length encoding

Table 6: Percentage compression achieved by run-length coding after Huffman and Comma encoding.

ISCAS circuit	Clock cycles C			Fault coverage (%)		
	[11] ¹	[12] ¹	Det ¹	[11]	[12]	Det
s298	— ²	899	477	—	86.04	86.04
s382	100K	4.6K	2.8K	86.00	89.47	91.22
s400	—	—	4.4K	—	—	90.14
s444	100K	4.6K	2.2K	80.40	87.76	89.45
s526	—	19.8K	4.6K	—	79.28	81.80
s35932	100K	—	2.0K	87.00	—	89.78

¹ [11, 12]: Recently proposed pseudorandom methods, Det: Deterministic testing; ² Results not reported.

Table 7: Pseudorandom testing vs testing with precomputed test sets.

chip. Table 9 shows that the levels of compression obtained for combined test sets are comparable to those obtained for the individual test sets. In fact, in several cases the overall compression is higher than that obtained for one of the individual test sets. The decoder area overhead required for the TGC is significantly reduced, because a single decoder can now be shared among several CUTs.

5 Conclusions

We have presented a novel technique for deterministic built-in self testing of sequential circuits using precomputed test sets. This approach is especially suited to sequential circuits that have a large number of flip-flops and relatively few primary inputs, and for which gate-level models are not available. We have shown that statistical encoding of precomputed test sets leads to effective test set compression. The small increase in testing time is offset by the high degree of test set compression achieved. Furthermore, the testing time is considerably less than that for pseudorandom methods. We have also developed efficient low-overhead pattern decoding methods for test application. The proposed technique thus offers a promising BIST methodology for complex non-scan and partial-scan embedded cores for which precomputed test sets are readily available.

References

[1] F. Brglez, D. Bryan and K. Kozminski. Combinational profiles of sequential benchmark circuits. *Proc. Int. Symp. Circuits and Systems*, pp. 1929-1934, 1989.

ISCAS circuit	Huffman decoders			Comma decoders		
	Gen	HIT	STRA	Gen	HIT	STRA
s298	46	44	41	24	32	28
s382	42	34	43	30	27	29
s400	44	33	37	26	27	27
s444	50	46	47	27	29	25
s526	43	47	43	29	29	27
s35932	2220	4002	4150	—	—	—

Table 8: Huffman and Comma decoder literal counts.

Circuit pair		Percentage test set compression		
		Gen	HIT	STRA
{s298,s400}	Huffman	49.24	47.87	35.43
{s382,s444}		58.65	49.25	36.00
{s444,s526}		51.85	42.42	42.96
{s298,s400}	Comma	49.21	47.76	33.79
{s382,s444}		58.60	49.16	33.59
{s444,s526}		51.81	42.33	42.72
Circuit pair		Percentage decoder overhead		
		Gen	HIT	STRA
{s298,s400}	Huffman	8.15	8.45	8.44
{s382,s444}		6.72	7.29	6.21
{s444,s526}		5.11	4.33	4.77
{s298,s400}	Comma	5.22	5.87	4.83
{s382,s444}		5.20	4.83	5.01
{s444,s526}		3.98	3.87	3.82

Table 9: Percentage compression and decoder overhead for a single decoder shared among several CUTs.

- [2] K. Chakrabarty, B. T. Murray, J. Liu and M. Zhu. Test width compression for built-in self testing. *Proc. Int. Test Conf.*, pp. 328-337, 1997.
- [3] W. T. Cheng and T. Chakraborty. Gentest—An automatic test-generation system for sequential circuits. *IEEE Computer*, vol. 22, pp. 43-49, April 1989.
- [4] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, New York, NY, 1991.
- [5] *Epoch Manual*, Cascade Design, Bellevue, WA, 1997.
- [6] M. C. Hansen and J. P. Hayes. High-level test generation using symbolic scheduling. *Proc. Int. Test Conf.*, pp. 586-595, 1995.
- [7] D. J. Holden. Focus report: Design for test tools. *Integrated Sys. Design*, pp. 36-52, September 1997.
- [8] The University of Illinois. www.crhc.uiuc.edu/IGATE
- [9] F. Muradali, T. Nishida and T. Shimizu. A structure and technique for pseudorandom-based testing of sequential circuits. *Journal of Electronic Testing: Theory and Applications*, pp. 107-115, 1995.
- [10] B. T. Murray and J. P. Hayes. Testing ICs: Getting to the core of the problem. *IEEE Computer*, vol. 29, pp. 32-38, November 1996.
- [11] L. Nachman, K. K. Saluja, S. Upadhyaya and R. Reuse. Random pattern testing for sequential circuits revisited. *Proc. Fault-tolerant Computing Symp.*, pp. 44-52, 1996.
- [12] I. Pomeranz and S. M. Reddy. Built-in test generation for synchronous sequential circuits. *Proc. Int. Conf. CAD*, pp.421-426, 1997.
- [13] M. S. B. Romdhane, V. K. Madiseti and J. W. Hines. *Quick-turnaround ASIC design in VHDL*. Kluwer Academic Publishers, Boston, MA, 1996.
- [14] T. Yamaguchi, M. Tilgner, M. Ishida and D. S. Ha. An efficient method for compressing test data. *Proc. Int. Test Conf.*, pp. 79-88, 1997.