Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression[1]

Anshuman Chandra and Krishnendu Chakrabarty
Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708, USA.
{achandra, krish}@ee.duke.edu

Abstract

We showed recently that Golomb codes can be used for efficiently compressing system-on-a-chip test data. We now present a new class of variable-to-variable-length compression codes that are designed using the distributions of the runs of 0s in typical test sequences. We refer to these as frequency-directed run-length (FDR) codes. We present experimental results for the ISCAS 89 benchmark circuits to show that FDR codes outperform Golomb codes for test data compression. We also present a decompression architecture for FDR codes, and an analytical characterization of the amount of compression that can be expected using these codes. Analytical results show that FDR codes are robust, i.e. they are insensitive to variations in the input data stream.

1 Introduction

Test data volume is a major problem encountered in the testing of system-on-a-chip (SOC) designs [1]. A typical SOC consists of several intellectual property (IP) blocks, each of which must be exercised by a large number of precomputed test patterns. The increasingly high volume of SOC test data is not only exceeding the memory and I/O channel capacity of commercial automatic test equipment (ATEs) but it is also leading to excessively high testing times. Data compression techniques that reduce test data volume are therefore of considerable interest.

The testing time of an SOC directly impacts test cost. It is determined by several factors, including the test data volume, the time required to transfer test data to the cores, the rate at which the test patterns are transfered (measured by the test data bandwidth and the ATE channel capacity), and the maximum scan chain length. For a given ATE channel capacity and test data bandwidth, reduction in testing time can be achieved by reducing the test data volume and by redesigning the scan chains. While test data volume reduction techniques can be applied to both soft and hard cores, scan chains cannot be modified in hard (IP) cores. New techniques are therefore needed to reduce the test data volume, decrease testing time, and overcome ATE memory limitations for SOCs containing IP cores.

Built-in self-test (BIST) has emerged as an alternative to ATE-based external testing [2]. BIST offers a number of key advantages. It allows precomputed test sets to be embedded in the test sequences generated by on-chip hardware, supports test reuse and at-speed testing, and protects intellectual property. While BIST is now extensively used for memory testing, it is not as common for logic testing. Test vectors for non-scan and partial-scan designs cannot be reordered, and they are harder to embed in a BIST generator. For full-scan designs, pseudorandom vectors can lead to serious bus contention problems during test application. Moreover, BIST can be applied to SOC designs only if the IP cores in it are BIST-ready. Since most currently-available IP cores are not BIST-ready, BIST insertion in SOCs containing these circuits is expensive and requires considerable redesign.

An alternative approach for reducing test data volume for SOCs is based on the use of data compression techniques such as statistical coding, run-length coding, and Golomb coding [3–8]. In this approach, the precomputed test set T_D provided by the core-vendor is compressed (encoded) to a much smaller test set T_E and stored in the ATE memory; see Figure 1. An on-chip decoder is used to generate T_D from T_E during pattern application. It was shown in [5, 6, 7] that compressing a "difference vector" sequence T_{diff} determined from T_D results in smaller test sets and reduced testing time. Figure 2 shows the test architecture based on T_{diff} and cyclical scan registers (CSRs).

While previous research has clearly demonstrated that data compression offers a practical solution to the problem of reducing test data volume, the compression codes used in prior work were derived from other application areas. For example, the statistical codes used in [3] and [4] are motivated by pattern repetitions in large text files. Similarly, the run-length and Golomb codes used in [5, 6, 7] are more effective for encoding large files containing image data. None of these codes are tailored to exploit the specific properties of precomputed test sets for logic circuits. While an attempt was made in [6, 7] to customize the Golomb code by choosing an appropriate code parameter, the basic structure of the

[1]This research was supported in part by the National Science Foundation under grant number CCR-9875324.

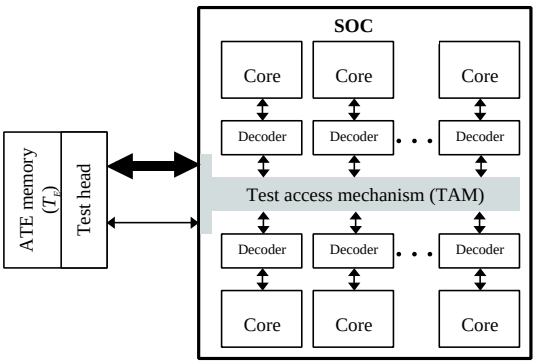

Figure 1. A conceptual architecture for testing a system-on-chip by storing the encoded test data T_E in ATE memory and decoding it using on-chip decoders.

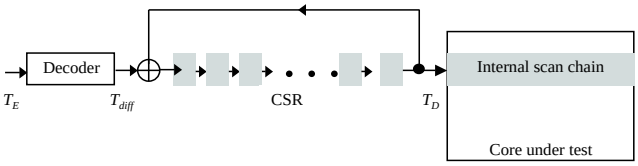

Figure 2. Decompression architecture based on a cyclical scan register (CSR).

code was still independent of the test set. We can therefore expect even greater reduction in test data volume by crafting compression codes that are based on the generic properties of test sets.

In this paper, we present a new class of variable-to-variable-length compression codes that are designed using the distributions of the runs of 0s in typical test sequences. In this way, the code can be tailored to our application domain, i.e. SOC test data compression. We refer to these as frequency-directed run-length (FDR) codes. For simplicity, we also refer to an instance of this class of codes as an FDR code. We show that the FDR code outperforms both Golomb codes and conventional run-length codes. We also show that the FDR code can be effectively applied to both the difference vector sequence T_{diff} and the precomputed test set T_D. The latter is especially attractive since it eliminates the need for a separate CSR for decompression. Additional contributions of this paper include a novel decompression architecture for FDR codes, and an analytical characterization of the amount of data compression that can be expected using these codes.

The organization of the paper is as follows. In Section 2, we first motivate the new FDR code and then describe its construction. In Section 3, we determine the best-case and the worst-case compression that can be achieved given some generic parameters of the precomputed test set. We describe some extensions to the basic FDR code and the decompression architecture in Section 4. Finally, in Section 5, we present experimental results for the large ISCAS 89 benchmark circuits.

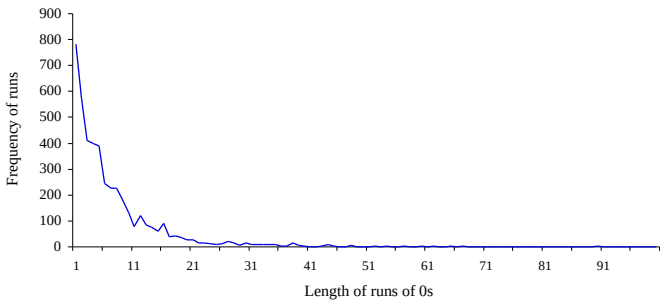

Figure 3. Distribution of runs of 0s for the ISCAS benchmark circuit s9234.

2 FDR codes

An FDR code is a variable-to-variable-length code which maps variable-length runs of 0s to codewords of variable length. It can be used to compress both the difference vector sequence T_{diff} and the test set T_D. Let $T_D = \{t_1, t_2, t_3, ..., t_n\}$, be the (ordered) precomputed test set. The ordering is determined using a heuristic procedure described later. T_{diff} is defined as follows: $T_{diff} = \{d_1, d_2, \ldots, d_n\} = \{t_1, t_1 \oplus t_2, t_2 \oplus t_3, \ldots, t_{n-1} \oplus t_n\}$. where a bit-wise exclusive-or operation is carried out between patterns t_i a nd t_{i+1}. This assumes that the CSR starts in the all-0 state. (Other starting states can be considered similarly). If the uncompacted test set T_D is used for compression, all the don't-care bits in T_D are mapped to 0s to obtain a fully-specified test set before compression.

We now present some important observations about the distribution of runs of 0s in typical test sets. We conducted a series of experiments for the large ISCAS benchmark circuits and studied the distribution of the runs of 0s in T_{diff} obtained from complete single stuck-at test sets for these circuits. Figure 3 illustrates this distribution for the s9234 benchmark circuit. We found that the distributions of runs of 0s were similar for the test sets of the other circuits.

The key observations from Figure 3 are as follows:

- The frequency of runs of 0s of length l is high for $0 \leq l \leq 20$.
- The frequency of runs of length l is small for $l \geq 20$.
- Even within the range $0 \leq l \leq 20$, the frequency of runs of length l decreases rapidly with increasing l.

If conventional run-length coding with block size b is used for compressing such test sets, every run of l 0s, $0 \leq l \leq 2^{b-1}$, is mapped to a b-bit codeword. This is clearly inefficient for the large number of short runs of 0s. Likewise, if Golomb coding with code parameter m is used, a run of l 0s is mapped to a codeword with $\lfloor l/m \rfloor + \log_2 m$ bits. This is also inefficient for short runs of 0s. Clearly, test data compression is more efficient if the runs of 0s that occur more frequently are mapped to shorter codewords. This leads us to the notion of FDR codes.

Group	Run-length	Group prefix	Tail	Codeword
	0	0	0	00
A_1	1		1	01
	2		00	1000
A_2	3	10	01	1001
	4		10	1010
	5		11	1011
	6		000	110000
	7		001	110001
	8		010	110010
A_3	9	110	011	110011
	10		100	110100
	11		101	110101
	12		110	110110
	13		111	110111
...	...	...	...	...

Figure 4. An example of FDR coding.

The FDR code is constructed as follows: The runs of 0s are divided into groups $A_1, A_2, A_3, \ldots, A_k$, where k is determined by the length l_{max} of the longest run ($2^k - 3 < l_{max} \leq 2^{k+1} - 3$). Note also that a run of length l is mapped to group A_j where $j = \lceil \log_2(l + 3) - 1 \rceil$. The size of the i^{th} group is equal to 2^i i.e., A_i contains 2^i members. Each codeword consists of two parts—a group prefix and a tail. The group prefix is used to identify the group to which the run belongs and the tail is used to identify the members within the group. The encoding procedure is shown in Figure 4. The FDR code has the following properties:

- For any codeword, the prefix and tail are of equal length. For example, the prefix and the tail are each one bit long for A_1, two bits long for A_2, etc.
- The length of the prefix for group A_i equals i. For example, the prefix is 2 bits long for group A_2.
- For any codeword, the prefix is identical to the binary representation of the run-length corresponding to the first element of the group. For example, run-length 8 is mapped to group A_3, and the first element of this group is run-length 6. Hence the prefix of the codeword for run-length 8 is 110.
- The codeword size increases by two bits (one bit for the prefix and one bit for the tail) as we move from group A_i to group A_{i+1}.

Note that run-lengths are also mapped to groups in conventional run-length and Golomb coding. In run-length coding with block size b, the groups are of equal size, each containing 2^b elements. The number of code bits to which runs of 0s are mapped increases by b bits as we move from one group to another. On the other hand, in Golomb coding, the group size increases as we consider larger runs of 0s, i.e. A_i is smaller in size than A_{i+1}. However, the tails for Golomb codewords in different groups are of equal length ($\log_2 m$, where m is the code parameter), and the prefix increases by only one bit as we move from one group to another. Hence

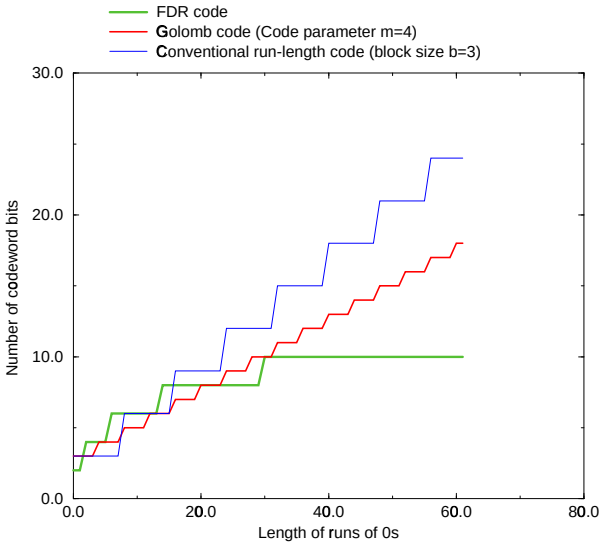

Figure 5. Comparison of codeword size (bits) for different run-lengths for the FDR code, Golomb code ($m = 4$) and conventional run-length code ($b = 3$).

Golomb coding is less effective when the runs of zeros are spread far from an "effective" range determined by m.

We now present a comparison between the three codes—conventional run-length code with block size $b = 3$, Golomb code with parameter $m = 4$ and the new FDR code. Figure 5 shows the number of bits per codeword for runs of 0s of different lengths. It can be seen from the figure that the performance of the conventional run-length code is worse than that of the Golomb code when the run-length l exceeds seven. The performance of the Golomb code is worse than that of the FDR code for $l \geq 24$. We also note that the new FDR code outperforms the other two types of codes for runs of length zero and one. Since the frequencies of runs of length zero and one are very high for precomputed test sets (Figure 3), FDR codes outperform run-length and Golomb codes for SOC test data compression.

3 Analysis of FDR codes

In this section, we develop an analysis technique to determine the worst-case and best-case compression that can be achieved using FDR codes for some generic parameters of precomputed test sets. Suppose T_{diff} (or T_D if it is encoded directly) contains r 1s and a total of n bits. We first determine C_{max}, the number of bits in the encoded test set T_E in the worst case, i.e. when the compression is the least effective. In doing so, we also determine a distribution of the runs of 0s that gives rise to this worst-case compression.

Suppose T_{diff} contains k_i runs of length i with maximum run-length l_{max}. Let the size of the encoded test set T_E be F bits, and let $\delta = F - (n - r)$ measure the amount of compression achieved using FDR codes. If the FDR coding procedure of Figure 4 is applied to T_{diff} then $\delta = 2k_0 + k_1 + 2k_2 + k_3 - k_5 - k_7 - 2k_8 - 3k_9 \pm \cdots$ (upto

l_{max}). This can be explained as follows: for each run of 0 of length i, we compare the size of the run-length (i) with the size of the corresponding codeword. For example, the codeword corresponding to a run of length 0 contains two bits (one more than the original run), the codeword for run-length 1 is of the same size as the original run-length, and so on. The difference between these two quantities contributes to δ, and it appears as the coefficient of the appropriate k_i term in the equation for δ.

We next use the following simple integer linear programming (ILP) model to determine the maximum value of δ. This yields the worst-case compression (C_{max}) using FDR codes.

Maximize: $\delta = 2k_0 + k_1 + 2k_2 + k_3 - k_5 - k_7 - 2k_8 - 3k_9 \pm \cdots$ (upto l_{max}) **subject to**: (1) $\sum_{i=1}^{l_{max}} ik_i = n - r$, and (2) $\sum_{i=1}^{l_{max}} k_i = r$.

This ILP model can be easily solved, e.g. using a solver such as *lpsolve* [9], to obtain the worst-case values for the k_i's. Note that even though l_{max} appears in the above ILP model, we do not make any explicit use of it. Our goal here is to determine a worst-case distribution of the runs of 0s. Generally, short run lengths yield the worst-case compression; however, if l_{max} must exceed a minimum value to satisfy constraints (1) and (2) above. We can use *lpsolve* to determine the minimum l_{max} by incrementally increasing l_{max} until the optimization problem becomes feasible.

Table 1(a) lists the size C_{max} of the encoded data set for worst-case compression for various values of n and r. The last column shows a distribution of runs for which the worst-case compression is achieved (a/b indicates a runs of length b). Note that this distribution is not unique since a number of run-lengths can yield the worst-case distribution. Note also that the worst-case percentage compression is negative when r is high relative to n—this is unlikely to be the case for test sets (don't-cares mapped to 0s) or difference vector sequences for which r is generally very small. It was shown in [6] that for the ISCAS89 benchmark circuits, the typical value of r is only 10% of n.

Next we analyze the best-case compression achieved using FDR codes for any given n and r. Since the compression is better for longer run-lengths, we also need to constrain the maximum run-length in this case. As before, we formulate this problem using ILP, and the following model can be solved using *lpsolve* to obtain a best-case distribution of runs and C_{min}, the number of bits in the encoded test set in the best case.

Minimize: $\delta = 2k_0 + k_1 + 2k_2 + k_3 - k_5 - k_7 - 2k_8 - 3k_9 \pm \cdots$ (upto l_{max}) **subject to**: (1) $\sum_{i=1}^{l_{max}} ik_i = n - r$, and (2) $\sum_{i=1}^{l_{max}} k_i = r$.

Table 1(b) lists the run-length distributions corresponding to the best case compression using FDR codes. The corresponding percentage compression values are also listed. In Figure 6, we plot the lower and upper bounds on the percentage compression as the number of runs r is varied (for $n = 1000$). We note that for small values of r, the bounds are very close to each other, hence the FDR code is robust, i.e. its efficiency is relatively insensitive to variations in the distributions of the runs.

n	r	C_{max}	Percentage compression $(1 - C_{max}/n) \times 100$	Worst-case distribution of runs
1000	100	674	32.6	59/6, 4/7, 37/14
	75	568	43.2	13/6, 3/7, 59/14
	50	400	60	30/14, 10/25, 10/28
	45	360	64	21/14, 1/17, 23/28
	40	320	68	10/14, 2/18, 28/28
2000	500	2250	Negative	375/2, 125/6
	200	1350	37.5	125/6, 75/14
	100	800	60	63/14, 2/21, 1/24, 34/28
	75	600	70	12/14, 1/21, 62/28

(a)

n	r	C_{min}	Percentage compression $(1 - C_{min}/n) \times 100$	Best-case distribution of runs
1000	100	374	62.6	71/1, 1/17, 28/29
	75	334	66.6	44/1, 1/11, 30/29
	50	294	70.6	17/1, 1/5, 32/29
	45	282	71.8	11/1, 1/4, 32/29
	40	278	72.2	7/1, 1/25, 32/29
2000	500	1216	39.2	464/1, 1/21, 35/29
	200	744	62.8	142/1, 1/5, 57/32
	100	588	70.6	35/1, 1/9, 64/32
	75	548	72.6	8/1, 1/3, 66/32

(b)

Table 1. Analysis of FDR codes: (a) worst-case compression; (b) best-case compression.

4 Extensions to the FDR code and test data decompression

The FDR coding algorithm described in Section 2 represents an instance of a code belonging to the class of more general FDR codes. This instance is especially suitable when the frequencies of runs decreases monotonically, i.e. the number of runs of length l is greater than the number of runs of length $l + 1$. It is also effective when the cumulative

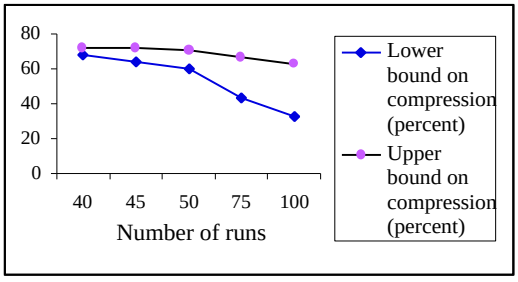

Figure 6. Comparison between the upper and lower bounds on percentage compression for $n = 1000$.

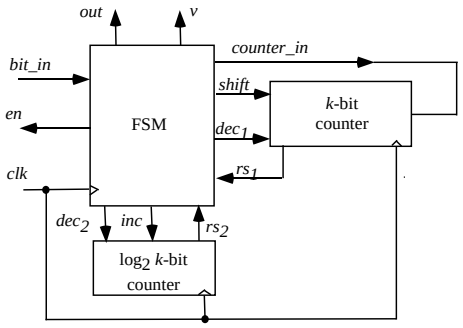

Figure 7. Block diagram of the decoder used for on-chip pattern decompression.

frequency of the runs in any group A_i exceeds the cumulative frequency of the runs in group A_{i+1}. However, for precomputed test sets, the run-length frequencies do not always decrease monotonically. For such non-monotonically decreasing run-lengths, the compression can be increased by extending the basic FDR code as described below.

For each group A_i, we calculate the cumulative frequency of the run-lengths in that group. This is done by simply adding the frequencies of the run-lengths in that group. Next, instead of assigning the group prefix as shown in Figure 4, we assign the prefix based on the cumulative frequency of that group. A group with a large cumulative frequency is assigned a short prefix. In this way, the size of the encoded test set can be reduced by carrying out a small amount of pre-processing, and by using a mapping logic block (outlined later) in the decoder.

We next describe the decompression architecture and the design of the on-chip decoder. The decoder is simple and scalable, and independent of both the core under test and the precomputed test set. Moreover, due to its small size, it does not introduce significant hardware overhead.

The decoder design is similar to the FSM-based decoder of [6, 7]. Issues related to data synchronization are described in [7]. The decoder decompresses the encoded test set T_E and outputs T_D. It can be efficiently implemented by a k-bit counter, a $\log_2 k$-bit counter and a finite-state machine (FSM). The block diagram of the decoder is shown in Figure 7. The bit_in is the input to the FSM and an enable (en) signal is used to input encoded data when the decoder is ready. The FSM output $counter\_in$ is used to shift in the prefix or the tail into the k-bit counter and the signals $shift$, dec_1 and rs_1 are used to shift the data in, to decrement, and to indicate the reset state of the counter, respectively. The second counter of $\log_2 k$-bits is used to count the length of the prefix and the tail so as to identify the group. The signals inc and dec_2 are used to increment and decrement the counter, respectively and rs_2 indicates that the counter has finished counting. Finally, the signal out is the decoder output and v indicates when the output is valid. The operation of the decoder is as follows:

- The FSM feeds the k-bit counter with the prefix. The end of the prefix is identified by the seperator 0. The en, $shift$ and inc signals are high till the 0 is received.
- The FSM output 0s and decrements the k-bit counter and makes the signal dec_1 high. It continues to output 0s until rs_1 goes high. The signal v is used to indicate a valid output.
- The tail part is shifted in until the $\log_2 k$-bit counter resets to zero. The dec_2 signal then goes high, the counter is decremented, and the signal rs_2 indicates when it is in the zero state.
- The FSM output 0s corresponding to the tail followed by a 1 at the end of tail decoding.

The state diagram for the FSM (not shown here due to lack of space) contains only 9 states. We synthesized the FSM using Synopsys design compiler—the synthesized circuit has only 4 flip-flops and 38 gates. Therefore, the additional hardware needed for the decoder is very small, and existing counters on the SOC can be reused for decompression.

The above decoder can be easily modified for decompressing data encoded using cumulative frequencies for the groups. Since the use of the cumulative frequencies affects only the prefix, and not the tail, we only need to add a mapping logic block between the encoded data stream and the decode FSM. Thus the mapping logic feeds the decode FSM and transforms the prefixes in the encoded data to the prefix assignment of Figure 4.

In our experiments with ISCAS 89 benchmark circuits, we observed that the run-lengths were never long enough to exceed group A_{10}. Therefore, in the worst case, the mapping logic is required for only ten prefixes. Moreover, as we show in Section 5, significant compression is almost always achieved without using the mapping logic.

5 Experimental results

We now present experimental results on test data compression for the large ISCAS benchmark circuits. We considered both full-scan and non-scan circuits for the proposed compression/decompression scheme. For full-scan circuits, patterns were reordered to achieve higher compression whereas no ordering was done for the non-scan circuits. For all the full-scan circuits, we considered a single scan chain. We use ATPG test cubes generated by Mintest ATPG program with dynamic compaction for our experiments.

The first set of experimental data that we present is based on the use of difference vector sequences T_{diff} obtained from partially-specified test sets (test cubes). Table 2 presents results for test cubes using FDR coding, Golomb coding, and fully-compacted test sets generated using Mintest. The table lists the percentage compression, sizes of the precomputed (original) test sets, sizes of

Circuit	Percentage compression (Golomb)	Percentage compression (FDR)	Size of T_D (bits)	Size of T_E (bits)	No. of bits for Mintest
s5378	40.7	48.19	23754	12306	20758
s9234	43.34	44.88	39273	21644	25935
s13207	74.78	78.67	165200	35226	163100
s15850	47.11	52.87	76986	36276	57434
s35932	N/A	10.19	28208	25332	19393
s38417	44.12	54.53	164736	74896	113152
s38584	47.71	52.85	199104	93860	161040

Table 2. Compression obtained using T_{diff}.

Circuit	Percentage compression (Golomb)	Percentage compression (FDR)	Size of T_D (bits)	Size of T_E (bits)	No. of bits for Mintest
s5378	37.11	48.02	23754	12346	20758
s9234	45.25	43.59	39273	22152	25935
s13207	79.74	81.30	165200	30880	163100
s15850	62.82	66.22	76986	26000	57434
s35932	N/A	19.37	28208	22744	19393
s38417	28.37	43.26	164736	93466	113152
s38584	57.17	60.91	199104	77812	161040

Table 3. Compression obtained using T_D.

the encoded test sets, and the sizes of the smallest ATPG-compacted test sets.

Table 2 shows that FDR codes provide better compression than Golomb codes in all cases. For the benchmark circuit s38417, there is as much as 10% increase in compression. We also note that that in all but one case, the size of the encoded test set T_E is much smaller than the compacted test set obtained using Mintest. The test cubes that we used for s35932 were already highly compacted, hence we did not obtain very high compression for this circuit. Nevertheless, it is significant that in contrast to FDR codes, no compression was obtained using Golomb codes for this circuit in [6, 7].

Table 3 demonstrates that the use of test cubes T_D (with all the don't-cares mapped to 0) often yields higher compression then over the use of T_{diff}. Moreover, in this case, the decompression architecture for on-chip pattern generation does not require a separate CSR. For circuits with long scan chains, addtional CSRs of lengths equal to the scan chain lengths increase the hardware overhead significantly. Therefore, compressing T_D to generate the encoded test set not only yields smaller test sets but also reduces the hardware overhead.

Finally, in Table 4, we present experimental results on test data compression for non-scan cicuits. We obtained the test sequences for these circuits from HITEC [11]. No reordering of test patterns was done during compression. Not surprisingly, we found out that that more compression is obtained using the mapping logic. The results also show that very high compression is achieved for non-scan circuits.

6 Conclusions

We have presented a new class of compression codes, termed frequency-directed run-length (FDR) codes. Experimental results for the ISCAS 89 benchmark circuits show that FDR codes outperform Golomb codes for test data compression. In addition to difference sequences derived from scan vectors, these codes can also be used directly with precomputed test sets and ordered test sequences for non-scan circuits. We have presented a decompression architecture for FDR codes, as well as an analytical characterization of the amount of compression that can be expected using these codes. Analytical results show that FDR codes are robust, i.e. they are insensitive to variations in the input data stream.

Circuit	Size of T_D (bits	FDR code (without mapping logic)		FDR code (with mapping logic)	
		Percentage compression	Size of T_E (bits)	Percentage compression	Size of T_E (bits)
s953	1168	73.45	310	75.94	281
s5378	169995	81.16	32020	81.45	31529
s13207	42284	93.44	2770	95.17	2039
s15850	147070	73.93	38330	82.41	25869
s35932	430353	83.35	71650	86.84	56618
s38417	22624	84.58	3488	89.26	2428

Table 4. Compression obtained using T_D and FDR codes for non-scan circuits, with and without mapping logic.

References

[1] Y. Zorian, E. J. Marinissen and S. Dey, "Testing embedded-core based system chips", *Proc. Int. Test Conf.*, pp. 130–143, 1998.

[2] B. T. Murray and J. P. Hayes, "Testing ICs: Getting to the core of the problem", *IEEE Computer*, vol. 29, pp. 32-38, Nov 1996.

[3] V. Iyengar, K. Chakrabarty and B. T. Murray, "Deterministic built-in pattern generation for sequential circuits", *JETTA*, vol. 15, pp. 97–115, 1999.

[4] A. Jas, J. Ghosh-Dastidar and N. A. Touba, "Scan vector compression/decompression using statistical coding", *Proc. VLSI Test Symp.*, pp. 114-120, 1999.

[5] A. Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based design", *Proc. Int. Test Conf.*, pp. 458-464, 1998.

[6] A. Chandra and K. Chakrabarty, "Test data compression for system-on-a-chip using Golomb codes", *Proc. VLSI Test Symp.*, pp. 113-120, 2000.

[7] A. Chandra and K. Chakrabarty, "System-on-a-chip test data compression and decompression architectures based on Golomb codes", Accepted for publication in *IEEE Trans. CAD*, 2000.

[8] A. Chandra and K. Chakrabarty, "Efficient Test Data Compression and Decompression for System-on-a-Chip using Internal Scan Chains and Golomb Coding", To appear in *Proc. DATE*, 2001.

[9] M. Berkelaar. *lpsolve*, version 3.0. Eindhoven Univ. of Technology, Design Automation Section, Eindhoven, The Netherlands. *ftp://ftp.ics.ele.nl/pub/lp_solve*.

[10] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits", *Proc. Int. Conf. CAD*, pp. 283-289, 1998.

[11] The University of Illinois, *www.crhc.uiuc.edu/IGATE*.