A Novel Huffman Based Compression Method for IP Cores*

Xrysovalantis Kavousianos[1], Emmanouil Kalligeros[1,2], and Dimitris Nikolos[2]
[1]Computer Science Dept., University of Ioannina, Ioannina, Greece
[2]Computer Engineering & Informatics Dept., University of Patras, Patras, Greece

Abstract

This paper introduces a new Test Resource Partitioning compression method for Intellectual Property cores. The proposed method compresses the test data supported by the vendor with a new very effective compression scheme based on Huffman code. The compressed data are decompressed on-chip with the use of simple decompression architecture. As it is shown experimentally the proposed method improves the compression ratio compared with other methods, while the hardware overhead of the decompressor is very low and comparable to the most efficient methods in the literature. Moreover, the proposed compression scheme offers a trade-off between compression ratio and area overhead of the decompressor. Finally, it enhances the coverage of unmodeled faults because a large portion of the unknown values of the test set are replaced with pseudorandom data generated by an LFSR.

Index Terms: Automatic Test Equipment, Test Resource Partitioning, Data Compression, Huffman Code, IP Cores, LFSR

I. Introduction

VLSI technology nowadays allows the placement of million of transistors on a single die. The complexity of integrated circuits (ICs) increases rapidly, with a direct impact on the cost of testing, which also increases very fast [Chandramouli 96], [Zorian 98]. In order to meet time to market constraints, contemporary systems embed pre-designed and pre-verified cores, which are called IP (Intellectual Property) cores. IP cores hide their structure from the system integrator, but support all the testing information required (e.g. pre-computed test tests). The system integrator is responsible for developing the proper structures at system level so as the test engineer to apply the test tests on the cores during the testing process. The increased density of ICs introduces new types of defects, which require new testing methods, increasing that way the volume of test data and the application time [Khoche 00]. When the volume of test data increases beyond the limited memory depth of the Automatic Test Equipment (ATE), multiple reloads are required to transfer the test data from the workstation to the ATE. ATE reloads are very time consuming [Vranken 03] and must be avoided in a competitive fast time-to-market environment. Therefore, in order to retain the cost of ATE low advanced testing techniques have to be adopted which can be applied by slow and low cost testers [Hetherington 99], [Rajski 01].

Increased test data volume increases also the test application time. Multiple scan chains can drive the test applicant time down, but require a large number of pins and high-speed ATE channels which may not be available on low cost testers. Test Resource Partitioning (TRP) has

* This research was co-funded by the European Union in the framework of the program "Pythagoras II" of the "Operational Program for Education and Initial Vocational Training" of the 3rd Community Support Framework of the Hellenic Ministry of Education, funded by 25% from national sources and by 75% from the European Social Fund (ESF).

been proposed to ease the burden of testing on ATE. It combines the ATE capabilities with on-chip integrated structures. TRP methods store a compressed test set on the ATE, which is downloaded and decompressed on chip. Among these methods are combinational continuous flow linear decompressors [Bayraktaroglu 03], [Krishna 03], the RESPIN architecture [Dorsch 01], the Illinois Scan Architecture [Hamzaoglu 99], [Hsu 01], LFSR based architectures [Hellebrand 95], [Khoche 02], [Jas 04b], [Krishna 04a], [Kalligeros 05], folding counters [Hellebrand 01] and weighted random pattern generators [Jas 99], [Jas 04a]. Also commercial tools exist which automate the generation of embedded decompressors [Barnhart 01], [Koenemann 01], [Rajski 03].

The above methods require structural information of the circuit under test (CUT), thus they are not suitable for IP cores. One category of methods for IP cores embeds the pre-computed test vectors in longer sequences of random vectors produced on chip [Chakrabarty 00], [Li 04]. Apart from the large application time, such methods suffer from the problem of X-generation [Hetherington 99]. The vectors applied to the IP cores which are not compatible with the test vectors supported by the vendor may produce internally unknown states (X). If an unknown state propagates to the MISR, it corrupts the signature. In these cases expensive methods such as X-suppression or response masking must be applied. Therefore, many methods have been proposed to reduce the test data volume and test application time by directly compressing the pre-computed test set T_D into T_E ($T_E<T_D$) without the interference of any useless vectors. Such methods encode the test sets using various codes with compression properties. Golomb codes were proposed in [Chandra 01], [Rosinger 01], [Chandra 02a], [Chandra 02b], alternating run length codes in [Chandra 03a], FDR codes [Maleh 02], [Chandra 03b], statistical codes in [Iyengar 99], [Gonciari 03], [Jas 03], nine-coded technique in [Tehranipour 05], and combinations of codes in [Tehranipour 04], [Nourani 04]. Some methods use dictionaries [Wolff 02], [Li 03], [Wurtenberger 04], [Knieser 03], [Reddy 03] but suffer from large hardware overhead due to the large embedded RAMs.

Some techniques compress the difference vectors instead of the actual test vectors [Jas 98], [Chandra 01], [Chandra 02b], [Chandra 03b]. This comes from the observation that the test vectors usually differ in a small number of bits, therefore the difference vectors will have long runs of 0s which can be effectively compressed using various run length codes. Such techniques are inefficient for circuits with internal scan chains which capture responses, because the data shifted in the scan chains are corrupted by the response of the circuit. In these cases either additional cyclical shift registers must be supported, which increase the cost of testing especially for cores with large number of scan cells, or the scan chains of other cores must be reused, if they are available .

There is a class of techniques requiring the pre-existence of arithmetic modules or processors embedded in the system [Maleh 01], [Balakrishnan 02], [Jas 02], [Hwang 03], [Hashempour 04], [Dutta 05].

Among the statistical codes used for compressing the test sets, the most effective are Huffman codes because they are provably optimal as they result in the shortest average codeword length [Iyengar 99], [Ichihara 00], [Ichihara 01], [Kajihara 02], [Jas 03]. The major problem with Huffman codes is the large hardware overhead of the decompressors. For that reason in [Jas 03] was proposed a selective Huffman compression scheme which sacrificing slightly the compression ratio reduces significantly the hardware overhead implied by the decompressor.

Utilization of all these codes for compressing pre-computed test sets is effective due to the large number of X values occurring in the test sets. Even after dynamic or static compaction the number of X values is quite large. Traditionally these X values are filled randomly with logic 0 or 1 in order to enhance the coverage of unmodeled faults. On the contrary, compression methods utilize these X values to achieve large compression ratios, by replacing them with the appropriate 0 or 1 logic values, depending on the implemented code. For example, sequences of 0 and/or 1 are used to replace X values in [Maleh 02], [Chandra 03a,b], [Gonciari 03], [Nourani 04].

Therefore, compression methods may adversely affect the coverage of unmodeled faults. In [Tehranipour 05] it is suggested that leaving at least a portion of the X values unchanged during compression is preferred.

In this paper we propose a statistical compression method based on Huffman code which fills the major portion of the X values of the test set with random values. This random filling is achieved by the use of an LFSR. In the same time the proposed method improves the compression ratio and requires a very simple decompressor with low overhead, offering also a trade-off between compression ratio and area overhead. The proposed method does not require any structural information of the CUT, thus it is proper for IP cores. Additionally it does not require any special modules or cyclical shift registers to be embedded in the system and it does not apply any useless vectors on the core.

The rest of the paper is organized as follows. Section II reviews the Huffman code, section III describes the proposed compression method and section IV presents the decompression architecture. Experimental results are presented in section V. Finally section VI concludes the paper.

II. Huffman Encoding

Statistical codes represent fixed-length blocks of data with variable length codewords. The efficiency of a statistical code depends on the frequency of occurrence of all distinct fixed length blocks. If each block occurs with the same or nearly the same frequency with the others then no compression can be expected by statistical coding. Statistical codes encode the most frequently occurring blocks with short codewords and the less frequently occurring blocks with large codewords, minimizing that way the average length of the codewords.

Among all statistical codes, Huffman code is the optimal one since it is proven that it provides the shortest average codeword length. Let k unique blocks occur in a test set T with probability of occurrence p_1, p_2, …..p_k, then the entropy of the test set in defined as $H(T)=-\sum_{i=1}^{k} p_i(\log_2 p_i)$ and intuitively is the minimum average length of the codewords required to represent the test set. Huffman code has the average codeword length which is closer to the theoretical limit of entropy bound compared with any other statistical code. In order to achieve the best compression ratio, the frequency of occurrence of the unique blocks must be as skewed as much as possible. This is usually easy to achieve in a test set because of the correlations of the test vectors and the large number of X values. A good encoding algorithm attempts to assign the X values in such a way as to achieve the most skewed frequency of occurrence of the codewords.

Another advantageous property of Huffman code for test compression is that it is prefix – free, that is no codeword is a prefix of another codeword. This makes the decoding process simple and easy to implement.

Example1 (Huffman Encoding). Consider the test set shown in Table 1 which consists of 5 unique blocks shown along with their probability of occurrence.

Table 1. Huffman Encoding Example

Test Set	Block	P	Codeword
0000 1010 1111 1010 1111 0000 1010 0001 1010 0000 0010 1010 0000 1010 0000 1010	1010	7/16	0
	0000	5/16	10
	1111	2/16	110
	0001	1/16	1110
	0010	1/16	1111

In order to construct the Huffman encoding, we generate a binary tree. Each block corresponds to a leaf node and a weight is associated with it, which is equal to the occurrence probability of that block. The pair of nodes with the lowest weights is selected and a parent node is created with weight equal to the sum of the weights of these two nodes. This is repeated iteratively selecting each time a pair of nodes without parents, until only a single node is left without parents, the root. Then each edge is associated with the logic value 0 if it leads to a left child and 1 if it leads to a right child. The codeword of each block is constructed by the logic values of the edges belonging to the path from the root towards the leaf node of the block. The Huffman tree of the above example is shown in Figure 1, and the codeword of each block is shown in Table 1. The average codeword length of this test set is $1 \cdot 7/16+2 \cdot (5/16)+3 \cdot (2/16)+4 \cdot (1/16+1/16)=1,9375$. Note that the entropy is equal to $H(T)= -(7/16 \cdot \log_2(7/16)+5/16 \cdot \log_2(5/16)+2/16 \cdot \log_2(2/16)+1/16 \cdot \log_2(1/16) + 1/16 \cdot \log_2(1/16))=1,9218$.

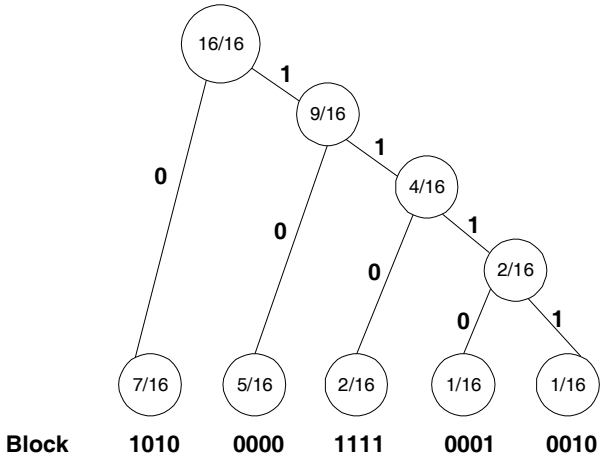


Figure 1. Huffman Tree

When the number of the encoded blocks is large then the cost of the Huffman decoder is high because of the large size of the decoding tree. For these cases a selective Huffman approach is adopted [Jas 03] which encodes only the most frequently occurring blocks while the rest blocks are not encoded. A special bit is appended to each block to indicate if the block is encoded or not.

III. Compression Method

The proposed compression method is based on Huffman code with limited number of codewords. The novelty of the method is that data produced by various cells of an LFSR are matched with the required test vectors and these cells are used for the encoding instead of the actual data. Each Huffman codeword is used to encode one cell of the LFSR. If a match with an LFSR cell can not be found then the data are encoded directly with Huffman code as proposed in [Jas 03]. Usually, the data encoded directly with Huffman code belong to vectors with many defined bits that can not be easily matched with the pseudorandom stream generated by the LFSR. On the other hand the major part of the test data encoded from LFSR cells belong to test vectors with many X values. Therefore the major portion of the X values is replaced with pseudorandom data, enhancing that way the probability of detection of unmodeled faults. In the following we describe the compression method assuming single scan chain.

Firstly, the test set is partitioned into clusters of fixed length. The LFSR is let evolve for a number of cycles equal to the size of the test set and the clusters of the test set are compared with the normal and inverted data produced by each LFSR cell. When a cluster of test data is compatible with a cluster generated by an LFSR cell, it is considered as a hit of the corresponding cell. A predetermined number of the LFSR cells with the largest hit ratios are selected, and feed the scan chain through a multiplexer. All the clusters which are compatible with data produced by at least one selected LFSR cell are encoded by one of them. Specifically, the multiplexer selection address of each cell is encoded using Huffman code. We call this type of encoding Cell encoding. All the clusters which are not compatible with a selected LFSR cell are labeled as failed and are processed afterwards in a different way, as it will be explained later on. Beside selection addresses, Cell encoding also associates one Huffman codeword with all failed clusters, in order to distinguish them from the rest.

Since many clusters have a large number of X values, they can be matched with many LFSR cells at the same time. Therefore the proposed method associates each cluster with that LFSR cell which skews the cell occurrence probabilities the most, that is the LFSR cell most frequently used. The processing of the Huffman tree is done in the ordinary way, using the probabilities of the cell addresses, as also the probability of the failed clusters.

Example 2 (Cell encoding). Suppose that a test set consists of 30 clusters, and 4 LFSR cells are used to match the test data. Suppose also that the matching probability of each LFSR cell, as well as the probability of the failed clusters, is shown in Figure 2. The Huffman tree and thus the codewords are generated using these frequencies.

Cell	Prob.	Code Word
A	10/30	00
B	5/30	10
C	4/30	110
D	3/30	111
Failed	8/30	01

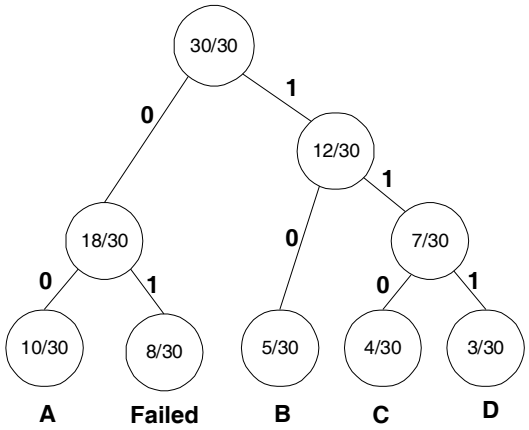


Figure 2. Cell encoding example.

A drawback of the Huffman code is that it is a fixed to variable code, whereas a variable to variable code is more efficient [Gonciari 03]. In the proposed method we try to eliminate this problem by allowing consecutive clusters to match, if possible, with the same LFSR cell and encode all these clusters only once. As we experimentally observed this was possible in all the cases due to the large number of X values in the test sets. For that reason besides the address of a selected LFSR cell, we also encode the number of consecutive clusters (cluster group) which can be encoded using this cell. In order to keep the hardware overhead low we allow the length of each group of clusters to be among a predetermined list of distinct lengths. These distinct lengths are experimentally selected as the powers of 2 in the interval [1, max_length) whereas max_length is the maximum number of consecutive clusters matched with any one of the selected cells. Each group of clusters is associated with the largest possible length in the list, which does not exceed the actual length of the group. For example if the list lengths are 1, 2, 4, 8, 16 and 32 then a group of 30 consecutive clusters is partitioned into a group of 16, a group of 8, a group of 4

and a group of 2 clusters. These list lengths are also encoded by Huffman code. This is justified from their probability of occurrence which is naturally skewed (large lengths are expected to occur less frequently than short lengths). We call this type of Huffman encoding as Length encoding. As it will be explained later, the same Huffman code used for Cell encoding is also used for Length encoding, in order to keep the decoding cost low. Therefore the maximum number of the potential list lengths is equal to the number of selected cells. In case that the list can hold more lengths than the number of the powers of 2 in the interval [1, max_length) then additional lengths are appended in the list following a different rule. The lengths of the list are sorted in ascending order and the pair of consecutive (in the list) lengths with the greatest distance between them is selected. The new length is the mid point between this pair of lengths. A codeword of Length encoding always succeeds a codeword of Cell encoding, when the encoded cluster is not a failed one.

Example 3 (Length encoding). Assume that 12 groups of clusters with lengths equal to 1, 2, 4, or 8, matched with 4 selected LFSR cells. The occurrence probabilities of these groups are shown at Figure 3. In this case, the Huffman tree is constructed by using these probabilities.

Group Length	Prob.	Code Word
1	6/12	0
2	3/12	10
4	2/12	110
8	1/12	111

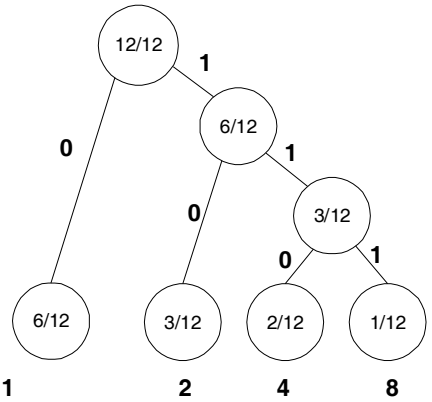


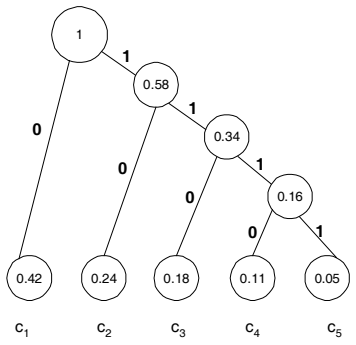
Figure 3. Length-encoding example

In the case of a failed cluster a different approach is adopted. The cluster is partitioned into equally sized blocks, and each block is encoded directly with the selective Huffman code as proposed in [Jas 03]. We call this encoding as Pure-Data encoding. If a block fails to be encoded with the selective Huffman code (failed block), then it remains not encoded and is supported directly by the ATE in the code stream. As in the case of failed clusters, one Huffman codeword is associated with each failed block, while the others are associated to the most frequently occurring blocks. The data of the not encoded block follows its codeword. In Pure-Data encoding the same Huffman code with Cell and Length encoding is used. Therefore, Pure-Data encoding encodes a number of blocks equal to the number of selected LFSR cells. Note that in Cell and Pure-Data encoding the failed data are distinguished by using Huffman codewords in contrast to [Jas 03] where a special bit is used in all codewords.

The major advantage of the proposed compression method is that the same Huffman decompressor can be used in order to implement the three different encodings. The number of selected cells determines the size of the Huffman decompressor and vice-versa. Note that the number of selected cells is equal to the number of the list lengths in Length encoding and to the number of unique blocks encoded by Pure-Data encoding. The Huffman tree is constructed by summing the corresponding occurrence probabilities of all three cases and one Huffman code is generated to cover all three of them. Thus the same codeword depending on the phase of the decoding corresponds to 3 different things: to a cell, to a cluster group length or to a block of data. Always the first codeword is considered as a Cell-codeword. If it does not indicate a failed cluster

then the next codeword correspond to the length of the cluster group. If instead it corresponds to a failed cluster then the next $\frac{cluster\ size}{block\ size}$ codewords are processed as Pure-Data codewords, where cluster size (block size) represents the number of bits of each cluster (block). Each one of them may indicate a failed block or a Pure-Data block. In the first case the actual block of data follow in the code stream else the block of data is produced by the decompressor. This sequence is iteratively repeated starting always from a Cell encoding codeword.

	Cell Encoding		Length Encoding		Pure-Data Encoding			
	Cell	Occur.	List Length	Occur.	Data Block	Occur.	P	Code Word
c_1	A	5	1	6	0011	8	(5+6+8)/45	0
c_2	B	4	2	3	Fail	4	(4+3+4)/45	10
c_3	Fail	3	4	2	0000	3	(3+2+3)/45	110
c_4	C	2	8	1	1111	2	(2+1+2)/45	1110
c_5	D	1			0001	1	(1+1)/45	1111
	SUM	15		12		18		
		Total Sum = 45						



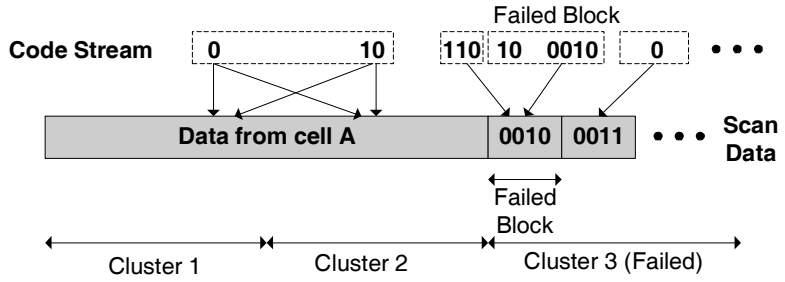


Figure 4. Compression example.

Example 4 (Compression method). Assume an encoding scenario with 4 cells of an LFSR, 4 different cluster group list lengths and 4 blocks of data. Let each cluster be 24 bit wide and each block 4 bit wide (6 blocks per cluster). Figure 4 presents the selected cells, the list lengths and the data blocks sorted by descending occurrence frequency. Each single occurrence in all three cases corresponds to a codeword in the final encoded stream. Note that there are 12 groups of clusters matched with LFSR cells and 3 failed clusters which are partitioned into 18 blocks. 5 unique codewords (one per line of the table) synthesize the code stream which consists of 45 occurrences of them. The occurrence frequencies in each line of the table are summed and divided by the total number of expected codewords, generating thus the total probability of occurrence of each distinct codeword, as shown in Figure 4. The encoded stream in Figure 4 shows the representation of scan data. The first codeword 0 corresponds to the cell A and the next codeword 10 indicates the group length which is 2. Therefore the scan chain is fed by cell A for the next two consecutive clusters 1, 2. The next codeword 110 indicates that the next cluster is a failed one. According to the proposed compression scheme, cluster 3 is partitioned into 6 blocks. The next codeword 10 indicates that the first block is a failed one; therefore the actual data (0010) are not encoded and follow this codeword in the code stream. The codeword for the second block is 0 which correspond to the encoded block 0011 which will be shifted in the scan chain. This is repeated until all 6 blocks have been processed.

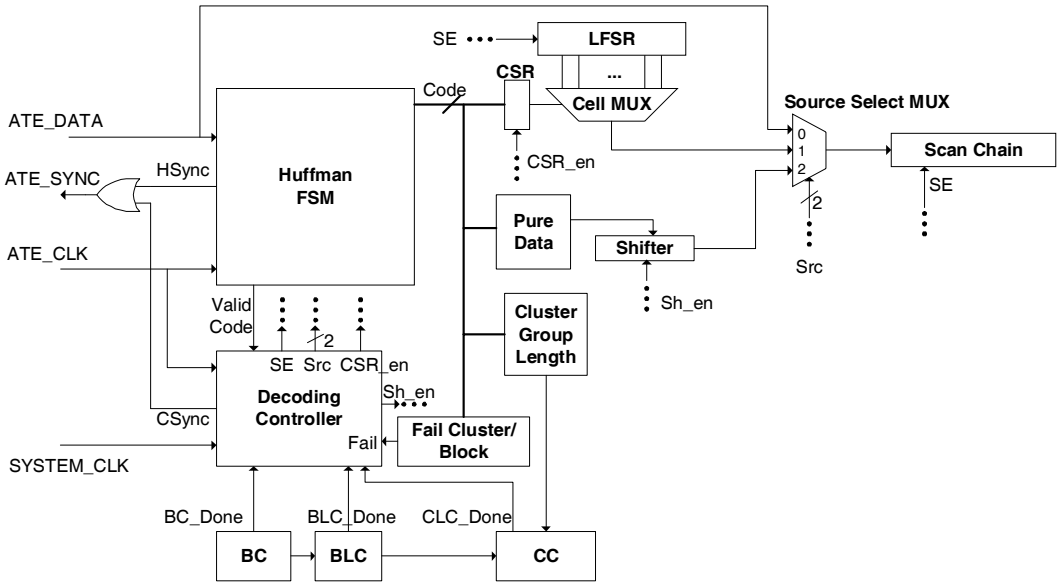


Figure 5. Decompression Architecture

IV. Decompressor Architecture

The block diagram of the proposed decompressor Architecture is shown in Figure 5. The functionality of the proposed architecture has been verified with extensive simulations. It consists of the following units*:

Huffman FSM: This unit receives the data from the ATE (*ATE_DATA*) using the ATE clock (*ATE_CLK*). Upon reception of a codeword the signal *HSync* is sent back to the ATE to stop the transmission until the decompressor is able to receive the next one. In the same time the *FSM* issues the signal code which is a binary number indicating which codeword has been received. In other words for *N* codewords, each codeword is assigned a number between 0 and *N*-1 and the signal code is the binary representation of that number. Also the FSM issues the signal *Valid Code* which informs the *Decoding Controller* that a codeword has been received and its number has been placed on signal code.

***Source Select Mux*:** This is the multiplexer which selects the source to feed the scan chain. Signal *Src* issued by the *Decoding Controller* selects the source that is the selected cell (*Src*=01) or the *Pure-Data* (*Src*=10) or a not encoded block directly from the ATE (*Src*=00).

Cell Mux: This is the multiplexer which selects the cell to feed the scan chain.

CSR (Cell Selection Register): This register holds the address of the selected cell during scan chain loading. It stores the address of the selected cell when the *Decoding controller* sets *CSR_en*=1.

LFSR: It is the Linear Feedback Shift Register, which is let evolve only during the loading of scan chain, using as enable signal the *SE* (Scan Enable).

Pure-Data: Combinational block which returns the pure data encoded in Huffman code upon recognition of the corresponding codeword. It receives as input the code signal issued by the *FSM* and outputs the corresponding data. It can be also implemented by a Lookup Table, with the signal code used for addressing the Table.

* For the convenience of the reader only the most important signals are reported.

Cluster Group Length: Combinational circuit which determines the group length corresponding to the received codeword. It can also be implemented by a Lookup Table in the same way as *Pure-Data* unit.

Shifter: a register which upon reception of a Pure-Data codeword (corresponding to a successfully encoded block) shifts in the scan chain the block of data retrieved by the *Pure-Data* unit. *Decoding controller* supervises the shifting of the register with the signal *Sh_en*.

Fail Cluster/Block: A very small combinational circuit which recognizes when a codeword corresponds to a failed cluster or a failed block and sets *Fail*=1.

Bit counter (BC), Block counter (BLC) and Cluster counter (CLC): Count the number of bits, blocks and clusters respectively, entered the scan chain (after initialization the counters count down until they reach zero). *Bit counter* issues signal *BC_Done* when a whole block has been shifted into the scan chain; *Block counter* issues the signal *BLC_Done* when all the blocks of a cluster have been shifted into the scan chain; *Cluster counter* issues the signal *CLC_Done* when all the clusters of a group have been shifted into the scan chain.

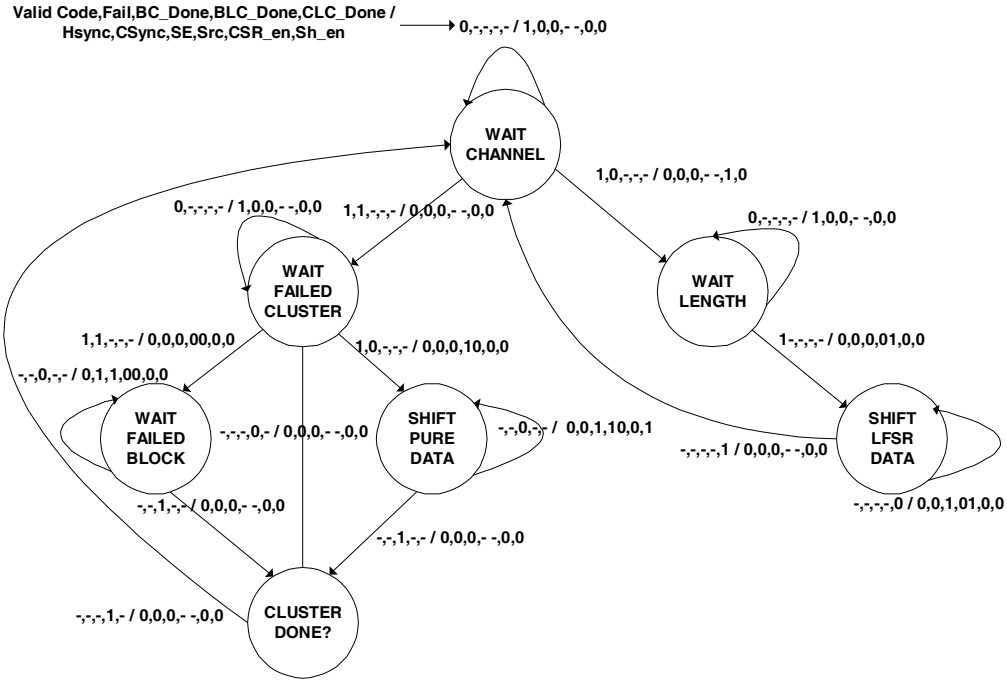


Figure 6. Decoding Controller State Diagram.

Decoding Controller: This is a finite state machine which synchronizes the operation of all units. The state diagram of this machine is shown on Figure 6 (the state diagram reports only the most important signals). Initially the controller waits for the first codeword which encodes a cell selection address. When the codeword has been received (*Valid Code* = 1) then the controller checks if it encodes an *LFSR* cell (*Fail*=0) or indicates a failed cluster (*Fail*=1). In the former case the controller stores the cell address to the *CSR resister (CSR_en*=1) and proceeds to the *WAIT_LENGTH* state. In the latter case it proceeds to the *WAIT_FAILED_CLUSTER* state. Being at the *WAIT_LENGTH* state it waits for the next codeword to be received by the *Huffman FSM* unit. Then it sends a signal to the *Cluster counter* to store the data returned by the *Cluster Group Length* unit which is the binary representation of the length of the group. Also the controller initializes the *Bit* and *Block counters* to their initial values, tunes the *Source Select* multiplexer to the *Cell Mux* input (*Src*=01)and proceeds to the *SHIFT_LFSR_DATA* state. At this state it activates the scan enable signal (*SE*=1) and the *LFSR* begins to load the scan chain with the data

from the selected cell, until the *Cluster counter* reaches zero (*CLC_Done*=1). Then the state machine returns to the *WAIT_CHANNEL* state waiting for the next iteration. If the machine enters the *WAIT_FAILED_CLUSTER* state then it waits for the next codeword from the *FSM*. If this codeword corresponds to an encoded Pure-Data block (*Fail*=0), then the controller sends a signal to the *Shifter* unit to store the output of the *Pure-Data* unit (which is the encoded data block), sets the *Source Select* multiplexer to the shifter input (*Src*=10) and proceeds to the *SHIFT_PURE_DATA* state where it remains until the *Bit counter* finishes counting (*BC_Done*=1). While the controller is in this state, the *Shifter* unit shifts the data serially into the scan chain. On the other hand if the codeword received at the *WAIT_FAILED_CLUSTER* state corresponds to a failed block (*Fail*=1) then the controller sets the *Source Select* multiplexer to the ATE channel (*Src*=00), and proceeds to the *WAIT_FAILED_BLOCK* state where it remains until the *Bit counter* finishes counting (*BC_DONE*=1). During this state the *Bit counter*, the *Decoding Controller* and the scan chain are triggered by the ATE clock and the signal *CSync* is set to 1 to enable the ATE to send directly the block which is not encoded. From both states *SHIFT_PURE_DATA* and *WAIT_FAILED_BLOCK* the state machine proceeds to the *CLUSTER_DONE?* state where the contents of the *Block counter* are checked through the signal *BLC_DONE*. If the Block counter has reached 0 (*BLC_Done*=1) all the blocks of the cluster have been processed, therefore the next state is *WAIT_CHANNEL*, otherwise the next state is *WAIT_FAILED_CLUSTER* where the next block will be processed in the same way.

As we will show in the experimental results, the efficiency of the proposed encoding depends mainly on the number of selected cells, which determine the number of codewords of the Huffman code. The number of codewords determines the size of the Huffman FSM which is the major part of the hardware overhead. Assuming that for two or more cores equal number of cells can be efficiently used for the encoding, the same decompressor can be used for all of them by changing only the following units: Cell Mux, Pure-Data, Cluster Group Length unit and Fail Cluster/Block unit. Moreover, if the Pure Data unit and Cluster Group Length units are implemented as Lookup Tables, then they only need to be loaded at the beginning of the test session with the specific data of each core. Therefore, the decompressor can easily be reused for different cores with virtually minimal or almost zero area penalty. An issue that will be clarified at section V, is that assuming equal number of selected cells, how effective is to use common Huffman codewords to more than one different cores. As we will see in the majority of the cases the effect on the compression ratio is only marginal. This is easy to explain taking into account that for the same number of cells (same number of Huffman codewords) and relatively skewed frequency of occurrence the Huffman trees can not be much different and thus the encoding if not optimal will be very close to the optimal one. (Note that the selected cells, the cluster size and the block size need not be the same for different cores when using common decompressor, regardless of the Huffman FSM unit being the same).

For multiple scan chains a shift register is used with width equal to the number of scan chains. The shift register is loaded by the decompressor and when it is full it loads the scan chains in parallel, as proposed in [Tehranipour 05].

V. Experimental results

The proposed compression method was implemented using the C-programming language. We run experiments on a Pentium PC for the largest ISCAS 89 benchmarks circuits, assuming full scan. We used the dynamically compacted test cubes generated by Mintest [Hamzaoglu 00], the same used in [Chandra 01], [Rosinger 01], [Chandra 02a], [Maleh 02], [Chandra 03a,b], [Jas 03], [Gonciari 03], [Tehranipour 04], [Li 05] and [Tehranipour 05]. The running time of the compression method is a few seconds for each benchmark circuit. The percentage compression, it is calculated by the formula

$$Percentage\ Compression = \frac{Data\ Bits - Compressed\ Bits}{Data\ Bits}$$

For all the reported experiments the LFSR size was set equal to 15.

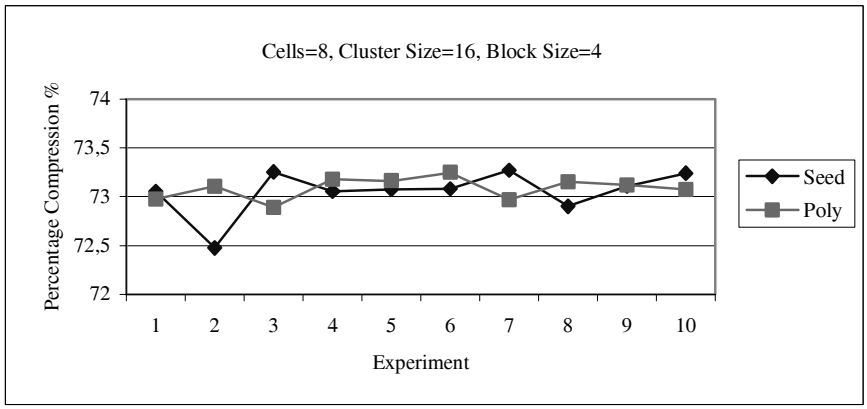


Figure 7. Seed and Polynomial Experiment for s15850.

The first experiment is reported in Figure 7 and studies the effect of different polynomials and seeds on the compression ratio. We applied the proposed method on the test set of s15850 with a) 10 different seeds and the same polynomial (Seed curve), and b) 10 different polynomials and the same seed (Poly curve). The rest parameters are reported in Figure 7 and are the same for both curves. As it is obvious from Figure 7, both polynomials and seeds affect the compression results in a very limited way, while seeds seem to affect the compression ratio a little more than polynomials. For all the experiments the variation of the percentage compression (maximum percentage – minimum percentage) for the 10 seeds is 0.79%, while for the 10 polynomials is 0.36%. Therefore we conclude that, seeds and polynomials marginally affect the proposed compression method.

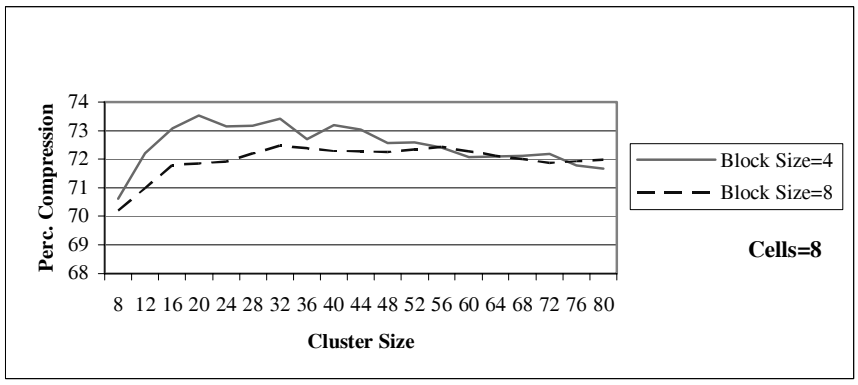


Figure 8. Varying cluster size for s15850

The second experiment shows the effect of different cluster sizes on the compression ratio. We applied the compression method on the test set of s15850, for cluster sizes between 8 and 80 bits (pace 4), one polynomial, one seed, 8 cells, and block sizes 4 and 8. It is obvious from Figure 8 that cluster size affects the compression ratio. For small and large cluster sizes the compression decreases, while it reaches a peak point somewhere in the middle. For small cluster sizes the number of clusters is high and thus the bits required for encoding is high as well, while for large

cluster sizes, the number of failed clusters increases. The variation of the percentage compression in this experiment approaches 3%.

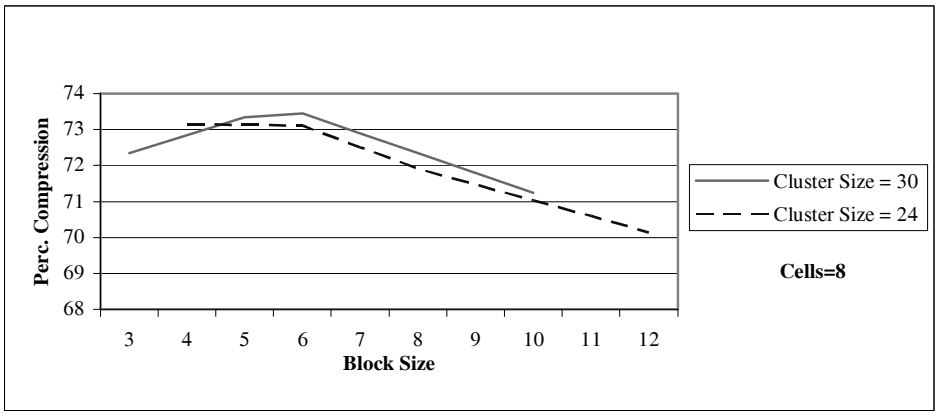


Figure 9. Varying Block size for s15850

The third experiment shows the effect of different block sizes on the compression ratio. We applied the compression method on the test set of s15850 varying the block size between 3 and 12 bits (with pace 1) for one polynomial, one seed, 8 cells and cluster sizes 24, 30. Note that block size affects only the encoding of failed clusters and consequently influences only Pure-Data encoding. The results are shown in Figure 9. Small block sizes increase the number of codewords needed to encode each failed cluster, because the number of blocks per cluster increases. Large block sizes adversely affect the compression achieved by Pure-Data encoding because as the block size increases, the number of distinct blocks increases exponentially, while the number of encoded blocks remains constant. Thus it is expected that the fraction of the number of encoded blocks to the number of all distinct blocks decreases, and thus many test data remain not compacted by Pure-Data encoding. The percentage variation in this experiment approaches 3% for block sizes 3 up to 12, while it is expected to be much greater for even larger block sizes.

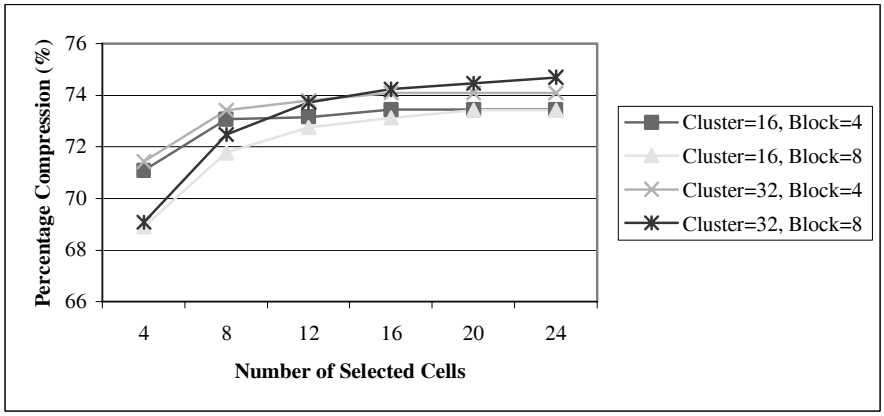


Figure 10. Varying Number of Channels for s15850f

Now we show the effect of the number of cells on the compression. We applied the compression method on the test set of the s15850 benchmark for cluster sizes 16 and 32, block sizes 4 and 8, one polynomial and one seed. For each experiment we varied the number of cells between 4 and 24 with pace 4. The results are shown in Figure 10. It is obvious that the compression ratio increases when the number of cells increases. As it is shown in Figure 10, for

small number of selected cells the compression ratio is mainly affected by the block size. This is justified from the large number of failed clusters when the number of cells is low. In other words low number of cells means more data encoded by Pure-Data encoding. When the number of cells increases, the block size affects the compression ratio marginally, while it is now affected mainly by the cluster size. In this case the number of failed clusters decreases and thus more test data are compressed by Cell encoding. After a certain number of cells the compression ratio reaches a saturation point as shown from the curve. The variation of the percentage compression in these experiments approaches 5.7%.

Table 2. Effect of predetermined Huffman code on compression ratio

Circuit	Test Set Size	Cluster Size	Block Size	Dedicated Decompr.	Common Decompr.	Decrease (%)
s5378	23754	12	6	10132	10132	0%
s9234	39273	20	4	16689	16689	0%
s13207	165200	40	10	21419	21568	0,6%
s35932	28208	90	9	7277	7458	2,4%
s38417	164736	36	6	63096	63840	1,1%
s38584	199104	30	6	59403	59862	0,8%

The next experiment shows the effect of using the same decompressor for various circuits, assuming equal number of selected cells for each one of them. As we mentioned in section IV, the number of selected cells affects the size of the Huffman FSM unit, which as it will be show later is the major area overhead of the decompressor. Therefore, in order to reuse the decompressor for testing different cores and minimize the area overhead we append for each core only the Pure-Data unit, the Cluster Group Length unit, the Fail Cluster/Block unit and the Cell Mux unit. The rest units (Huffman FSM, Decoding Controller, counters, CSR and Source Select Mux) can be reused as is. Using the same Huffman FSM unit for a number of cores means that the compression method is based on pre-generated Huffman codewords for each core. In other words the codewords corresponding to cells, list lengths and data blocks are the same for each core, while the selected cells, list lengths and data blocks need not be the same. With this experiment we show that using the pre-generated code for various cores the decrease of the compression is very limited compared to the compression achieved when a separate code is generated for each one of them for the same number of cells. Firstly, we generate the code for the test set of the s15850 circuit for 8 selected cells, cluster size 28 and block size 4. Then we compressed the test set of each benchmurk, for 8 cells and various cluster sizes and block sizes assuming separate decompressors for each one of them. We also compressed the test sets of each benchmark circuit with the exact same parameters for each core using the pre-generated code produced for the s15850. The results are shown in Table 2. The first column presents the circuit name, the second presents the size of the test set, the next two columns present the cluster and block size used for each benchmark. Block and cluster size were selected among others so as to give the best compression when dedicated compressors are used. The column labeled Dedicated Decompressor presents the compression achieved when the encoding of each core is generated based on each test set separately. The column labeled Common Decompressor presents the compression when the pre-generated code for the s15850 circuit is used for each one of the reported circuits. Column labeled Decrease presents the percentage decrease in compression of the common decompressor compared to the dedicated one. It is obvious that the difference in compression is very low or even zero. As it will be shown next, the larger of number of cells is the greater is the compression achieved. Therefore, for various cores we may implement the decompressor for the core requiring the largest number of cells and then reuse it for the rest cores, minimizing that way the percentage decrease for each one of them.

Table 3. Compression results

	Cells = 4		Cells = 8		Cells = 12		Cells = 16		Cells = 20		Cells = 24	
Circuit	**ENC**	**C/B**	**ENC**	**C/B**	**ENC**	**C/B**	**ENC**	**C/B**	**ENC**	**C/B**	**ENC**	**C/B**
s5378	10446	16/8	9876	12/6	9817	20/10	9627	20/10	9494	20/10	**9370**	20/10
s9234	17491	16/4	16555	20/4	16277	20/5	16269	20/10	15921	32/8	**15636**	20/10
s13207	23076	40/8	20428	40/10	19931	64/8	19577	64/8	19429	64/8	**18832**	64/8
s15850	21819	20/4	20203	28/4	20110	20/5	19806	40/8	19436	32/8	**19382**	64/8
s38417	65630	20/5	62811	36/6	61912	40/8	60379	40/8	59533	40/8	**58943**	48/8
s38584	63233	20/4	58995	30/6	58780	32/8	57372	20/10	56556	32/8	**56504**	20/10

Table 3 presents the compression for all benchmark circuits achieved by the proposed method. The same polynomial was used in all experiments while 10 random seeds were tried for each experiment. Compression results for 6 different numbers of cells are reported, 4 to 24 with pace 4. For each number of cells the compression method was applied on each test set for various cluster and block sizes. Among them the best results are reported. Columns labeled C/B report the cluster size / block size for each circuit, and columns labeled ENC present the size of the encoded test sets (test set sizes are reported in Table 2). It is obvious that in all cases the compression increases when the number of cells increases.

Table 4. Comparisons with [Jas 03] and and [Tehranipour 05].

		Jas 03		Tehranipour. 05	
Circuit	**Prop.**	**Enc.**	**Impr.**	**Enc.**	**Impr.**
s5378	9370	10666	12,2%	10511	10,9%
s9234	15636	17987	13,1%	17763	12,0%
s13207	18832	37996	50,4%	24450	23,0%
s15850	19382	26175	26,0%	22126	12,4%
s38417	58943	67542	12,7%	61134	3,6%
s38584	56504	71478	21%	62897	10,2%

Table 4 compares the proposed method with [Jas 03] which is also based on Huffman code, and [Tehranipour 05] which is the most effective compression method proposed so far in the literature. Column labeled Prop. reports the best compression achieved by the proposed method in terms of bits of the encoded test set. The same information is provided for the other two methods under column labeled Enc. Columns labeled Impr. report the relative improvement of the proposed compression to the compression of the other two methods as (Enc-Prop)/Enc. It is obvious that the proposed scheme gives better compression results than both these methods.

Table 5. Comparisons with other methods

Circuit	Chand. 01	Rosing. 01	Chand . 02a	Maleh 02	Chand. 03b	Chand. 03a	Gonc. 03	Tehr. 04	Li 05
s5378	-	35,9%	-	17,9%	24,1%	19,9%	18,2%	14,7%	34,1%
s9234	29,7%	34,7%	30,5%	26,4%	29,4%	27,7%	24,5%	24,0%	48,1%
s13207	54,8%	50,4%	46,4%	37,2%	39,0%	42,3%	30,9%	34,8%	10,3%
s15850	52,4%	38,1%	36,6%	21,3%	25,5%	26,3%	21,5%	22,9%	22,9%
s38417	36,0%	19,7%	35,3%	9,3%	36,9%	9,3%	23,3%	0,1%	30,8%
s38584	45,7%	34,6%	37,1%	23,5%	27,4%	27,0%	24,8%	24,5%	1,1%

Table 5 gives the compression improvement of the proposed method against other compression methods which have comparable hardware with the proposed method and reported results for the Mintest test sets. To have a fair comparison, we do not compare the proposed method with methods compressing difference vectors because they require cyclical shift registers. It is obvious that the proposed compression method performs than all the other methods.

Table 6. Hardware Overhead

	Area Overhead (Gate Equivalents)				
Number of cells	Rest Units	Huffman FSM	Group Length	Pure Data	Total
4	163	34	3	3	203
8	181	63	9	10	263
12	191	88	9	26	314
16	199	110	11	35	355
20	205	128	11	53	397
24	221	142	11	58	432

In order to show the area overhead of the proposed method we synthesized decompressors for various numbers of cells using with Leonardo Spectrum (Mentor tools). The area overhead depends strongly on the number of cells, which mainly affects the area of the Huffman FSM unit. The block size affects the hardware overhead in a limited way since only the area of the Pure-Data unit depends on the block size. Table 5 shows the area overhead of the decompressor in gate equivalents for different numbers of cells, 8 bits block size and 16 bits cluster size. The decompressors were synthesized for the test set of s15850. We have to note that the decompressor area does not depend on the test set, but mainly on the architecture parameters. Therefore the area picture is similar for the rest benchmarks too. Columns labeled Huffman FSM, Group Length and Pure Data present the area attributed to the corresponding units. Column Rest Units present the area overhead of the rest units in the design. As it is obvious the area overhead depends mainly on the area of Huffman FSM which is getting larger as the number of cells (and thus the number of Huffman codewords) increases. The area overhead of Group Length and Pure Data unit is only a small portion of the total overhead. The rest units occupy an almost constant area which increases very slowly due to the increased size of the Cell Mux unit, as also the increased size of the counters. Under column Total the area overhead of the decompressor is reported. The reported area overhead varies between 11,5% and 24% of the s15850 circuit area. For larger benchmarks as s38584 the overhead varies between 3,1% and 6,6% while for larger circuits it is expected to be even less. If the same decompressor is used for different cores then the area overhead for each circuit confines mostly to the area occupied by Group Length and Pure Data while the other units are shared among all cores. Then the above percentages for s15850 vary between 0,3% and 4,5% and for s38584 vary between 0,092% and 1,2% which is extremely low.

In order to compare the proposed decompressor in terms of hardware overhead (gate equivalents) with the most efficient methods, we report the area overhead of various methods: 416 for [Tehranipour 05], 320 for [Chandra 03a], 136-296 for [Gonciari 03], 125-307 for [Chandra 01] (as reported in [Gonciari 03]). In [Jas 03] hardware overhead is provided as percentage of the benchmarks circuit and can not be directly compared with these above methods. However, compared with [Gonciari 03] reported inferior results in area overhead. It is obvious that the proposed decompressors are comparable with the other proposed decompressors.

VI. Conclusion

This paper presented a compression method based on Huffman code. The compression achieved is in most of the cases higher than all the other methods while the area overhead of the proposed decompressor is very low. Moreover, a trade-off between area and compression ratio is supported. The proposed decompressor can be easily shared among different cores, minimizing that way the overall implementation cost in a system. Additionally, the major portion of X values is filled with pseudorandom data produced by an LFSR, enhancing that way the coverage of unmodeled faults. Experimental results proved that the compression achieved is better that the

other methods, while the hardware overhead is comparable to the hardware overhead of these methods.

References

[Balakrishnan 02] K.J. Balakrishnan, N.A. Touba, "Matrix-Based Test Vector Decompression Using an Embedded Processor", 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 159 – 165, 2002.

[Barnhart 01] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, B. Koenemann "OPMISR: the foundation for compressed ATPG vectors", Proceedings International Test Conference, pp. 748 – 757, 2001.

[Bayraktaroglu 03] I. Bayraktaroglu, A. Orailoglu, "Concurrent Application of Compaction and Compression for Test Time and Data Volume Reduction in Scan Designs", IEEE Transactions on Computers, Volume: 52 , Issue: 11, pp. 1480- 1489, 2003.

[Chakrabarty 00] K. Chakrabarty, B. Murray, V. Iyengar, "Deterministic Built-In Test Pattern Generation for High-Performance Circuits using Twisted-Ring Counters", IEEE Trans. On Very Large Scale Integration (VLSI) Systems, pp. 633-636, October 2000.

[Chandra 01] A. Chandra, K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 20, No. 3, pp. 355-368, March 2001.

[Chandra 02a] Chandra A., Chakrabarty K., "Test Data Compression and Decompression Based on Internal Scan Chains and Golomb Coding", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, volume: 21, No 6, pp. 715-72, June 2002.

[Chandra 02b] A. Chandra, K. Chakrabarty and R. A. Medina, "How Effective are Compression Codes for Reducing Test Data Volume?", Proceedings of the 20th IEEE VLSI Test Symposium, pp. 2002.

[Chandra 03a] A. Chandra, K. Chakrabarty, "A Unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22 , No 3, pp. 352 – 363, March 2003

[Chandra 03b] A. Chandra, 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 , No 8, pp. 1076 – 1088, Aug. 2003

[Chandramouli 96] R. Chandramouli, S. Pateras, "Testing Systems on a chip", IEEE Spectrum, pp. 42-47, November 1996.

[Das 00] D. Das, N. A. Touba, "Reducing Test Data Volume Using External/LBIST Hybrid Test Patterns", Proceedings International Test Conference, pp. 115-122, 2000.

[Dorsch 01] R. Dorsch, H. J. Wunderlich, "Tailoring ATPG for Embedded Testing", Proceedings International Test Conference, pp. 530-577, 2001.

[Dutta 05] A. Dutta, T. Rodrigues, N.A. Touba, "Low Cost Test Vector Compression / Decompression Scheme for Circuits with a Reconfigurable Serial Multiplier", Proceedings IEEE Computer Society Annual Symposium on VLSI, pp. 200- 205, 2005.

[Gonciari 03] P.T. Gonciari, B.M. Al-Hashimi, N. Nicolici, "Variable-Length Input Huffman Coding for System-On-A-Chip Test", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22 , No 6, pp. 783 – 796, June 2003

[Hamzaoglu 99] I. Hamzaoglu, J. H. Patel, "Reducing Test Application Time for Full Scan Embedded Cores", Proceedings International Symposium Fault-Tolerant Computers, 1999

[Hamzaoglu 00] I. Hamzaoglu, J. H. Patel, " Test Set Compaction Algorithms for Combinational Circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, No. 8, pp. 957-963, August 2000.

[Hashempour 04] H. Hashempour, F. Lombardi, "Compression of VLSI Test Data By Arithmetic Coding", Proceedings. 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 150- 157, 2004.

[Hellebrand 95] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, B. Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers" IEEE Transactions on Computers, Vol. 44, No 2, pp. 223 – 233, 1995

[Hellebrand 01] S. Hellebrand, H.-G. Liang and H.-J. Wunderlich, "A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters", Journal of Electronic Testing, Vol. 17, No. 3 – 4, June 2001, pp. 341 - 349

[Hetherington 99] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, J. Rajski, "Logic BIST for Large Industrial Designs: Real Issues and Case Studies", Proceedings International Test Conference, pp. 358-367, 1999.

[Hsu 01] F. Hsu, K. Butler, J. Patel, "A Case Study on the Implementation of the Illinois Scan Architecture", Proceedings International Test Conference, pp. 538-547, 2001.

[Hwang 03] S. Hwang, J. A. Abraham, "Test Data Compression and Test Time Reduction Using an Embedded Microprocessor", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 11, No 5, pp. 853- 862, 2003

[Ichihara 00] H. Ichihara, K. Kinoshita, I. Pomeranz, S.M. Reddy, "Test Transformation to Improve Compaction by Statistical Encoding", 13th International Conference on VLSI Design, pp. 294 – 29, 2000.

[Ichihara 01] H. Ichihara, A. Ogawa, T. Inoue, A. Tamura, "Dynamic Test Compression Using Statistical Coding", Proceedings 10th Asian Test Symp., pp. 143 – 148, 2001.

[Iyengar 99] V. Iyengar, K. Chakrabarty, B. T. Murray, "Deterministic Built-In Pattern Generation for Sequential Circuits", Journal of Electronic Testing, vol. 15, pp. 97-114, 1999

[Jas 98] A. Jas and N. A. Touba, "Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs", Proceedings International Test Conference, pp. 458-464, 1998

[Jas 99] A. Jas, K. Mohanram, and N. A. Touba, " An Embedded Core DFT Scheme to Obtain Highly Compressed Test Sets", Proceedings Asian Test Symposium, pp. 275-280, 1999.

[Jas 02] A. Jas and N. A. Touba, "Deterministic Test Vector Compression/Decompression for Systems-on-a-Chip Using an Embedded Processor", Journal of Electronic Testing, Issue: Volume 18, No 4 – 5, pp. 503 - 514, August 2002

[Jas 03] A. Jas, J. Ghosh-Dastidar, M. –E. Ng and N. A. Touba, "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding", IEEE Transaction on CAD of Integrated Circuits and Systems, vol. 22, No 6, June 2003

[Jas 04a] A. Jas, C.V. Krishna, N.A. Touba, "Weighted Pseudorandom Hybrid BIST", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 12 , Issue: 12 , 2004, Page(s): 1277- 1283

[Jas 04b] A. Jas, B. Pouya, N.A. Touba, "Test Data Compression Technique for Embedded Cores Using Virtual Scan Chains", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, No 7, pp. 775- 781, 2004

[Kajihara 02] S. Kajihara, K. Taniguchi, K. Miyase, I. Pomeranz, S.M. Reddy, "Test data compression using don't-care identification and statistical encoding", Proceedings of the 11th Asian Test Symposium, pp. 67 – 72, 2002

[Kalligeros 05] E. Kalligeros, X. Kavousianos, D. Nikolos, "Multiphase BIST: A New Reseeding Technique for High Test Data Compression", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, pp. 1429-1446, vol. 23, No 10, Oct. 2004.

[Khoche 00] A. Khoche, J. Rivoir, "I/O Bandwidth Bottleneck for Test: Is it Real?", IEEE Test Resource Partitioning Workshop, pp. 2.3-1 – 2.3-6, 2000.

[Khoche 02] A. Khoche, E. Volkerink, J. Rivoir, S. Mitra, "Test Vector Compression Using EDA–ATE Synergies" Proceedings of the 20th IEEE VLSI Test Symposium, pp. 97-102, 2002.

[Kiefer 97] Kiefer G., Wunderlich H.-J., "Using BIST Control for Pattern Generation", Proceedings International Test Conference, pp. 347 – 355, 1997.

[Knieser 03] M.J. Knieser, F.G. Wolff, C.A Papachristou., D.J. Weyer, D.R. McIntyre, "A Technique for High Ratio LZW Compression", Design, Automation and Test in Europe Conference and Exhibition, pp. 116 – 121, 2003

[Koenemann 01] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, D. Wheater, "A SmartBIST Variant with Guaranteed Encoding", IEEE Asian Test Symposium, pp. 325-330, 2001

[Krishna 03] C.V. Krishna, N.A. Touba, "Adjustable Width Linear Combinational Scan Vector Decompression", IEEE/ACM International Conference on Computer Aided Design, pp. 863- 866, 2003

[Krishna 04a] C.V. Krishna, N.A. Touba, "3-Stage Variable Length Continuous-Flow Scan Vector Decompression Scheme", Proceedings. 22nd IEEE VLSI Test Symposium, pp. 79-86, 2004

[Li 03] L. Li, K. Chakrabarty, N. A. Touba, "Test Data Compression Using Dictionaries with Selective Entries and Fixed-Length Indices", ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 8 No 4, October 2003

[Li 04] L. Lei, K. Chakrabarty, "Test Set Embedding for Deterministic BIST Using A Reconfigurable Interconnection Network", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.23, No. 9, pp. 1289- 1305, Dec. 2004.

[Li 05] Lei Li, K. Chakrabarty, S. Kajihara, S. Swaminathan, "Efficient space/time compression to reduce test data volume and testing time for IP cores", 18th International Conference on VLSI Design, 3-7 Jan. 2005, Page(s): 53- 58

[Liang 02] H.-G. Liang, S. Hellebrand and H.-J. Wunderlich, "Two-Dimensional Test Data Compression for Scan-Based Deterministic BIST" Journal of Electronic Testing, Vol. 18, No. 2, pp. 159 - 170, April 2002.

[Maleh 01] A. El-Maleh, S. al Zahir, E. Khan, "A Geometric-Primitives-Based Compression Scheme for Testing Systems-On-A-Chip" 19th IEEE Proceedings on VLSI Test Symposium, pp. 54 – 59, 2001

[Maleh 02] A.H. El-Maleh, R.H. Al-Abaji, "Extended Frequency-Directed Run-Length Code with Improved Application to System-On-A-Chip Test Data Compression" 9th International Conference on Electronics, Circuits and Systems, vol. 2, pp. 449-452, 2002

[Nourani 04] M. Nourani, M. H. Tehranipour, "RL-huffman Encoding for Test Compression and Power Reduction in Scan Applications", ACM Transactions on Design Automation of Electronic Systems (TODAES) Vol. 10, No 1, pp. 91 – 115, 2004

[Rajski 01] J. Rajski, "DFT for High-Quality Low Cost Manufacturing Test", Proceedings. 10th Asian Test Symposium, pp. 3 – 8, 2001.

[Rajski 03] J. Rajski, M. Kassab, N. Mukherjee, N. Tamarapalli, J. Tyszer, J. Qian, "Embedded deterministic test for low-cost manufacturing", IEEE Design & Test of Computers, vol. 20, No 5, pp. 58- 66, 2003

[Reddy 03] S. M. Reddy, K. Miyase, S. Kajihara, I. Pomeranz, "On Test Data Volume Reduction for Multiple Scan Chain Designs", ACM Transactions on Design Automation of Electronic Systems Vol. 8, No 4, pp. 460 – 469, 2003

[Rosinger 01] P. Rosinger, P.T. Gonciari, Al-Hashimi B.M., Nicolici N., "Simultaneous Reduction in Volume of Test Data and Power Dissipation for Systems-On-A-Chip", Electronics Letters, Vol. 37, No. 24, pp. 1434 – 1436, 2001.

[Tehranipour 04] M. Tehranipour, M. Nourani, K. Arabi, A. Afzali-Kusha, "Mixed RL-Huffman Encoding for Power Reduction and Data Compression in Scan Test", Proceedings of the 2004 International Symposium on Circuits and Systems, Vol. 2, pp. II- 681-4, 2004

[Tehranipour 05] M. Tehranipour, M. Nourani, K. Chakrabarty, "Nine-Coded Compression Technique for Testing Embedded Cores in SoCs", IEEE Trans. On Very Large Scale Integration (VLSI) Systems, vol. 13, No. 6, June 2005.

[Volkerink 02] E.H. Volkerink, A. Khoche, S. Mitra, "Packet-Based Input Test Data Compression Techniques", Proceedings. International Test Conference, pp. 154 – 163, 2002

[Vranken 03] H. Vranken, F. Hapke, S. Rogge, Chindamo D., Volkerink E., "Atpg Padding and ATE Vector Repeat per Port for Reducing Test Data Volume", Proceedings International Test Conference, pp. 1069 – 1078, 2003

[Wolff 02] F.G. Wolff, C. Papachristou, "Multiscan-Based Test Compression and Hardware Decompression Using LZ77", Proceedings. International Test Conference, pp. 331 – 339, 2002

[Wurtenberger 04] A. Wurtenberger, C.S. Tautermann, S. Hellebrand, "Data Compression for Multiple Scan Chains using Dictionaries with Corrections", Proceedings International Test Conference, pp. 926- 935, 2004

[Zorian 98] Y. Zorian, E. J. Marinissen, S. Dey, "Testing Embedded-Core-Based System Chips", Proceedings International Test Conference, pp. 130-143, 1998.