A Distributed Algorithm for XOR-Decompression with Stimulus Fragment Move to Reduce Chip Testing Costs

Mohammed Almulla and Ozgur Sinanoglu

***Abstract*—** Various techniques were used to reduce the test time and cost of chip development, some of which achieved their objective by reducing the test data volume through the implementation of compression technologies such as XOR-based decompressors. In the presence of XOR decompressor, the delivery of acceptable (encodable) test patterns can be challenging. To overcome this problem, the Align-Encode technique was introduced to manipulate the distribution of care bits in the test pattern in aim to increase the delivery of more encodable test patterns. The implementation of the Align-Encode algorithm proved that this algorithm suffers a major drawback when applied on large test patterns. In this paper, we propose a distributed algorithm for realizing the Align-Encode objectives but for large scale problems. This algorithm is designed to run on a scalable distributed environment. Moreover, it exploits the nature of the problem in order to make significant improvements in performance with respect to chip testing time as well as the number of encodable test patterns generated, which reflects positively on the cost of chip development and in test data compression as a result.

***Keywords*—** VLSI Chip testing, XOR-Decompression, Align-Encode, Distributed processing.

I. Introduction

THE tremendous amount of test data - required to ensure the high quality testing of the chips of ever increasing sizes - translates into heightened levels of test costs on multiple fronts. The sizable test sets not only pressure the memory requirements imposed on the external tester, magnifying the associated tester costs, but they furthermore engender the challenge of test bandwidth limitations. The necessity to deliver a huge amount of test data through a limited number of tester channels onto the chip induces significant prolongations in test time, imposing intolerable test costs on the expensive tester.

Logic cone structure of typical designs lead into don't care bits (x's) in test vectors, paving the way for test data squashing techniques. This type of a redundancy innate in test data can be exploited by compressing the test vectors. The compressed stimuli is stored and transmitted from the tester to the chip being tested, and is expanded on-chip. In a typical scan architecture that supports compression, a decompressor expands a few scan-in channels into a larger number of scan chains. While test data volume and test time are thus reduced, the underlying structure of the stimulus decompressor determines the encodability of a test pattern. In the case of combinational decompressors, for instance, the test vector fragment to be delivered into a scan slice is analyzed to judge whether the care bits of the fragment can be obtained intact at the outputs of the decompressor. A test vector is encodable if and only if each of its fragments can be obtained at the decompressor outputs. A single unencodable slice renders the test vector unencodable. Thus, the distribution of the care bits of a test vector determines whether the test vector is encodable. Unless an unencodable test vector is replaced by deliverable ones that cover the faults/defects that it detects, test quality is degraded. Alternatively, another test mode can be employed wherein the decompressor is bypassed, in order to deliver these unencodable test vectors serially, and hence to restore the test quality at the expense of test time and data volume.

Test stimulus manipulation techniques help enhance test vector encodability of a decompressor by changing the distribution of the care bits judiciously. One such technique, referred to as Align-Encode [1], [2], is based on the horizontal move of stimulus fragments inserted into scan chains. Such a capability can be attained by inserting controllable delay elements on the scan-in path of scan chains. By inserting a delay element on the selected chains, effectively the stimulus of the corresponding chain is shifted, offering an alternative distribution that may possibly be encodable. The work in [1], [2] has demonstrated the beneficial application of horizontal move of stimulus fragments to boost the encodability of fan-out decompressors.

In this paper, we propose a distributed implementation of the Align-Encode algorithm, in order to further improve the effectiveness of the XOR decompressor utilized on the development of large chips. We also emphasis the impact of

Manuscript received June 5, 2009. A preliminary version [20] of this work has been presented at the 8th WSEAS International Conference on Applied Computer and Applied Computational Sciences, Hangzou, China, May 20-22, 2009.

M. Almulla is with the Dept of Math and Computer Science, Kuwait University (phone: 965-2498-5335; fax: 965-2481-7201; e-mail: m.almulla@ku.edu.kw).

O. Sinanoglu is with the Department of Math and Computer Science, Kuwait University, (e-mail: ozgur@sci.kuniv.edu.kw).

parallelism on chip testing with the help of various Artificial Intelligence (AI) techniques in solving this challenging problem. As a result, we attain significant runtime improvements in finding the delay configuration to make larger patterns encodable. Due to the runtime improvements, the quality of results is also improved. A more efficient and parallel exploration of the search tree enables the distributed approach to increase the number of encodable patterns, some of which remain unencodable due to runtime limitation of the serial algorithm. The distributed algorithm enhances the compression levels of the accompanying decompressor, consequently.

II. PEEVIOUS WORK

Conceptually, stimulus manipulation techniques (Align-Encode being the only one in the literature) can be utilized in conjunction with any combinational decompressor, in order to boost the encoding capability of the decompressor. Various decompressors were outlined in [3]. These techniques include fan-out decompressors [4]-[6], XOR decompressors [7], [8], multiplexer/fanout decompressors [9], and switch-based decompressors [10]. Other solutions include LFSR re-seeding [11]-[13], a combination of single input shift registers, clock gating logic and an XOR network [14], scan tree architectures [15], [16], and sequential decompressors [17].

To compensate for the defect/fault coverage losses due to unencodable patterns, these techniques either employ an additional compression-free phase [4], [5], or they utilize test generation as well [7], [17], searching for alternative encodable test vectors for the missed faults. The idea of exploring various implementation strategies to further reduce test cost of integrated circuitry was earlier proposed FPGA realization of Open/Short Test on IC [18]. A different path for the process characterization and description using an empirical approach - rather than probabilistic one - in order to electronic device reliability assessment was suggested in [19].

III. BENEFIT CARE BIT DISTRIBUTION MANIPULATION

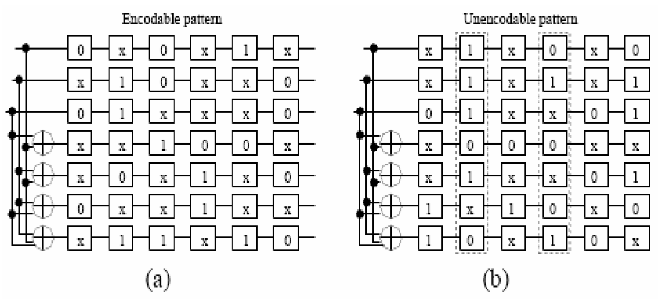

Fig. 1: Example of encodable and non-encodable patterns

In test data compression, correlated test data delivery to scan cells is inevitable. The encodability of a test pattern is a matter of whether the care bit distribution within the pattern complies with the correlation induced by the decompressor. In the case of combinational decompressors, each scan slice can be analyzed individually and independently to judge on the encodability of the test vector. In Figure 1, one encodable and one unencodable test pattern are provided for an XOR-based 3x7 decompressor. XOR-based decompressors necessitate solving a linear equation for each care bit in order to perform an encodability check. Three channels drive seven scan chains in Figure 1, necessitating the solution of the following set of linear equations:

$$\begin{array}{lllllll}
v_{1j} & & & & & = & t_{1j} \\
v_{2j} & & & & & = & t_{2j} \\
v_{3j} & & & & & = & t_{3j} \\
v_{1j} & \oplus & v_{3j} & & & = & t_{4j} \\
v_{1j} & \oplus & v_{2j} & & & = & t_{5j} \\
v_{2j} & \oplus & v_{3j} & & & = & t_{6j} \\
v_{1j} & \oplus & v_{2j} & \oplus & v_{3j} & = & t_{7j}
\end{array}$$

where v_{kj} denotes the stimulus bit inserted from the k^{th} scan-in channel during the j^{th} shift cycle, and t_{ij} denotes the test pattern bit that corresponds to the j^{th} scan cell of the i^{th} scan chain. In the examples in Figure 1, $1 \leq k \leq 3$, $1 \leq j \leq 6$, and $1 \leq i \leq 7$. It can be observed that the left hand side of the equations are constructed based on the decompressor structure, and the right hand side of the equations are composed of the test pattern bits. Any equation with an x on its right hand side can be safely deemed as automatically solvable. As an independent set of v_{kj} 's is introduced in every shift cycle, one system of equations for each shift cycle (scan slice) can be solved independently to compute these variables. Upon the successful solution of all the systems, the test pattern can be deemed as encodable; the test pattern in Figure 1.a is encodable. A single unsolvable system renders a pattern unencodable; the second and the fourth slices from the left render the test pattern in Figure 1.b unencodable. In the equations of the second slice, for instance, the first three equations necessitate that $v_{12} = v_{22} = v_{32} = 1$, which contradict with the fifth and the seventh equations.

The distribution of care bits can be judiciously manipulated in order to improve test vector encodability. The manipulation of the care bit distribution can be achieved through delaying the shift-in operation of scan chains for the proper alignment of scan slices. Such a manipulation affects the right hand side of the equations, while it keeps the left hand side of the equations intact, as the latter depends on the XOR decompressor structure.

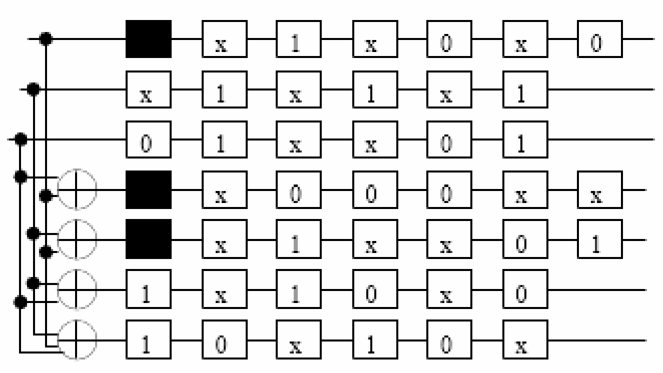

Figure 2: A pattern becomes encodable after applying Align-Encode

Figure 2 demonstrates the beneficial impact of shift-in delay operations on the unencodable test pattern in Figure 1(b). The shift-in of the first, fourth and fifth scan chains is delayed by a single cycle each, delivering the encodability of the originally unencodable pattern, as all the systems of linear equations become solvable.

IV. THE ALIGN-ENCODE ALGORITHM

The care bit distribution in each test pattern should be analyzed to compute the proper delay values that lead to the encoding of the test pattern. It is also possible that no delay assignment exists for the encoding of the pattern; in this case, the pattern remains unencodable, necessitating either a subsequent serial application or replacement with other encodable patterns.

The input to the analysis is a scan pattern. The target of this analysis is the computation of the proper delays, either 0 or 1, for each scan chain so that the pattern becomes encodable in the new alignment of the slices. The analysis should be repeated for each scan pattern in order to compute the proper delay data for the entire test set.

In this section, a test pattern analysis for XOR-based decompressors is provided. Align-Encode provides the capability of delaying any chain by a single cycle. Full exploration of the search space in order to find the proper delay values requires an exponential time complexity solution, as 2^n different configurations exist for n chains.

A pseudo-code for the algorithm is provided in Figure 3. The algorithm utilizes a backtracking technique which is simply an exploration of a binary decision tree. The decision tree helps prune the search space by identifying and eliminating in a depth-first-search manner delay configurations that leads to unencodability.

Figure 3: Sequential algorithm for computing delay values

```
1.  Compute_Delay_Values(TestPattern, level)
2.  If(level ≤ n)
3.     For every slice $s_i$
4.        If(Gaussian_Elimination($s_i$, level) == FALSE)
5.           Return;
6.     If(level == n)
7.        Return report delay value;
8.     Else
9.        Compute_Delay_Values(TestPattern, level + 1);
10.       Delay the (level + 1)th chain;
11.       Compute_Delay_Values(TestPattern, level + 1);
12.       Undo Delay the (level + 1)th chain;
13.    Return;
```

The decision tree for the example in the previous section is provided in Figure 4. In this figure, the path that leads to the encodability of the pattern in Figure 1.b is also highlighted. The right nodes (denoted by 1's) are selected in levels 1, 4 and 5, denoting that chains 1, 4, and 5 should be delayed, in a manner consistent with the configuration in Figure 2.

The algorithm starts from the root level, which corresponds to subsystems with no equations, and expands these sub-systems by one level (equation) at each step. In the process, the algorithm traverses the nodes of the decision tree in a depth-first-search manner. When the algorithm traverses a node at level *i*, the sub-system of equations consist of those corresponding to scan chains 1 through *i*. Gaussian Elimination technique is applied to check the solvability of the subsystems in each step. It is possible that the subsystem of equations, which corresponds to the path from the root to the current node, is unsolvable for at least one of the slices. We exploit the fact that if a subsystem of equations is unsolvable, so will its further expansions be. Thus, in such a case, the sub-tree, which is rooted by the current node in the current level, can be safely pruned, leading to computational savings. The pruning is affected by the depth-first-search nature of the algorithm, which backtracks by one level up and tries to solve the system of equations with the current chain delayed (extending the path by selecting the other node in the same level). If the system of equations becomes solvable for all the slices, the algorithm proceeds by extending the path deeper down in the decision tree. Otherwise, it backtracks one level up and changes the delay configuration of the chain in the preceding level. Upon the identification of the path from the root to one of the leaves that results in the encodability of the test pattern, the algorithm terminates successfully with the delay configuration corresponding to this path. If no such path exists, the algorithm terminates unsuccessfully.

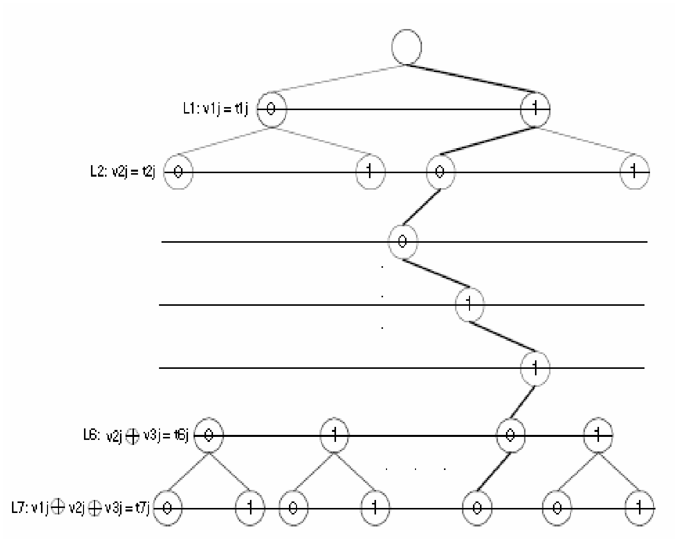

Figure 4: Sample Search Tree

V. DISTRIBUTED VERSION OF THE ALIGN-ENCODE ALGORITHM

The problem's nature is based on searching a large state-space where no dependency exists between different solutions. Thus, we can divide the search-space into equally-sized parts and explore these parts independently. Furthermore, the position of the solution within the decision tree affects the time required to find it dramatically. Therefore, being able to initiate the search anywhere in the search tree would enhance the performance of the search algorithm.

Experimental results of the Align-Encode algorithm indicate that this algorithm tends to be impractical for testing large patterns (≥ 64x64) [20]. Hence, distributed memory architecture [21] is being proposed since each instance of the application will run on a separate node, where some kind of inter-process (and inter-machine) communication has to take place to exchange data. For this purpose, the Message Passing protocol is used for exchanging messages between nodes. The Master-Slave model adopted in this solution is divided into two major modules:

- The Controller Module (Master): This part of the application is executed on the controller node, which is responsible for reading input data, distributing the work (partitioning the search-space) over the working nodes and gathering the results.
- The Working Module (Slaves): This part of the application receives input from the controller, searches the assigned search-space sequentially and reports the result back to the controller or will be waiting for other assigned tasks.

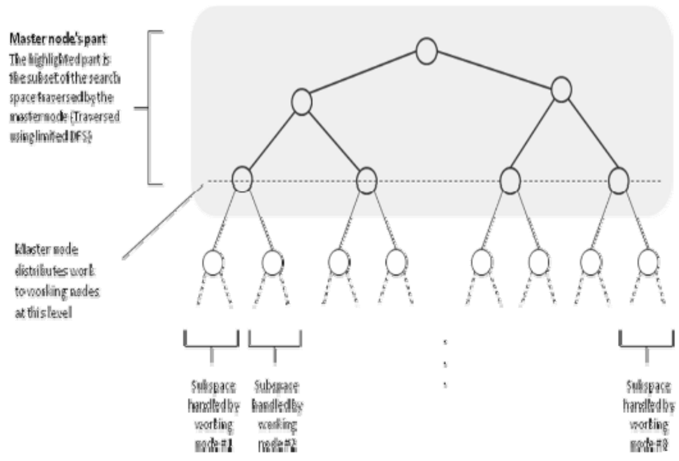
Master node's part
The highlighted part is the subset of the search space traversed by the master node (Traversed using limited DFS)
Master node distributes work to working nodes at this level
Subspace handled by working node#1
Subspace handled by working node#2
Subspace handled by working node#8

Figure 5: Example search tree and work distribution with 8 working nodes

The controller is dedicated to manage the working nodes, synchronize them, distribute work and gather results. After reading the test pattern, the controller does a limited depth first traversal on the search tree up to the *(k-1)*th level building a bit sequence made up from the bits representing the branch chosen at each step (*0* for left branch, 1 for right branch). When the deepest level is reached, the controller sends the test pattern data and the currently constructed bit sequence prefix to the corresponding working node initiating the processing as in Figure 5. The following pseudo-code describes the parallel XOR Align Encode algorithm:

Figure 6: The parallel algorithm

```
Parallel_XOR_Align_Encode() {
    If Node = Controller {
        For each TestPattern t {
            DistributeWork(t, {} )
            For i = 1 → numberOfWorkingNodes {
                solution = RecieveMessage()
                if IsSolution(solution) {
                    StopRemainingNodes()
                    Return solution
                }
            }
        }
    }
    Else {
        While(true){
            ReceiveTestPattern(t, delayConfiguration)
            solution =
                SolveSequentially(t, delayConfiguration)
            SendMessage(solution)

        }
    }
}

DistributeWork(TestPattern t, BitArray b) {
    If length(b) =log( numberOfWorkingNodes){
        SendMessage(t, b)
    }
    Else{
        b1 = Append(b, '0')
        DistributeWork(t,b1)        //branch without shift

        b2 = Append(b, '1')
        DistributeWork(t, b2)       //branch with shift
    }
}
```

During the Limited DFS, the 2^C-sized search space which consists of the set of all states representing the delay bit values

$b_0b_1b_2....b_{c-1}$ is divided equally over n nodes where $n = 2^k$, $k \geq 1$. Each working node w_i will receive the input test patterns and start doing sequential search over the sub-space consisting of all states $p_0p_1..p_{k-1}b_kb_{k+1}...b_{c-1}$ where p_j is the *j-th* bit in the *k*-bit binary representation of *i*. Each subspace has a size of 2^{C-k}. For example, if the number of chains of the test pattern is *8,* and the number of working nodes is *8* then we have $2^8 = 256$ different delay configurations $b_1b_2...b_8$. The distribution of the search space over the *8* nodes is as follows:

$w_0 \rightarrow \{ 000b_3b_4...b_7 \}$
$w_1 \rightarrow \{ 001b_3b_4...b_7 \}$
...
$w_7 \rightarrow \{ 111b_3b_4...b_7 \}$

Once the work distribution is completed, the controller starts listening and waiting for a working node to send a result back. If the received result contains a "No solution found" flag, the controller continues waiting for more results. When a valid delay configuration is received, the controller sends a "Stop work" message to all the remaining working nodes and waits a "Ready" message from each of them which indicates that a node is ready to receive and work on a new working node. If all the working nodes return result with "No solution found" flag, the test pattern is considered unencodable. The whole process is repeated until the input test patterns are processed.

VI. EXPERIMENTAL RESULTS

A scalable test environment was set in order to perform the necessary experiments. This environment consisted of a 2^m+1(=17 in our case where m=4) homogeneous nodes connected with an ad-hoc network. MPI and the test application were installed on every node. One of these nodes was designated as a controller (master) node. The working node module was installed on each of the other 16 machines. Each node had a single Pentium 4 processor with 1.7 GHz speed, 256 MB RAM and running Windows XP SP2 operating system. This particular node configuration was suitable for test cases having 32x32 test patterns. For 64x64 test cases, a higher node configuration was needed. Therefore, a similar yet more powerful environment was setup where each node having a single Pentium D (Dual Core) processor, with 3 GHz speed, 1GB RAM and running Windows XP SP2. In short, 32x32 test patterns were tested on the lower-end configuration machines while the 64x64 test patterns were tested on the higher-end configuration machines, with the number of nodes kept fixed in both environments.

In order to estimate the effect of parallelism and to investigate the relationship between the number of nodes used and the speedup gained, the test set ran on different cluster sizes (1 (sequential), 2, 4, 8 and 16 nodes). The test set contained test cases that cover a combination of preset values of the test pattern size *S* (32x32, 64x64), number of decompressor variables *V* (8, 9, 10, 11, 12) and ratio in percentage of don't care bits R (0.80, 0.85, 0.90, 0.95). Each test case consisted of 250 randomly generated (prepared by a test pattern generator utility) test patterns. For test cases with test patterns of size 64x64, a time limit per test pattern was set to 10 minutes. For each test case, execution time and solvability status were reported.

The following figures show the execution time (in millisecond) consumed by running a set of test cases with *R* = 0.80, 0.85, 0.90 and 0.95 don't care bit ratios, respectively, on 32x32 test patterns.

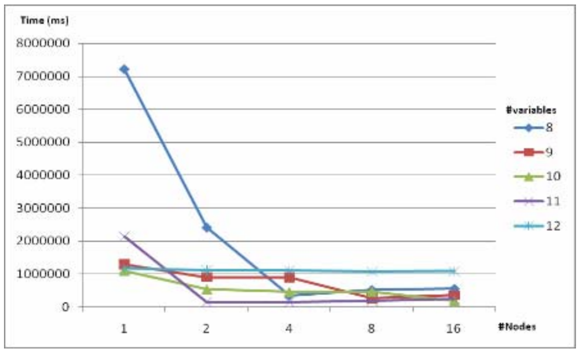

Figure 7

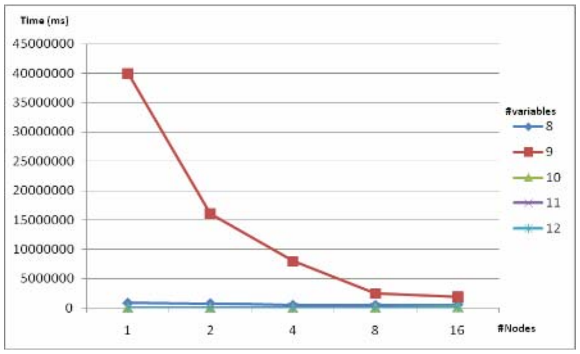

Figure 8

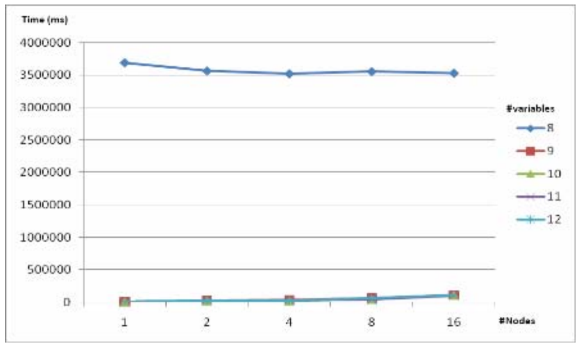

Figure 9

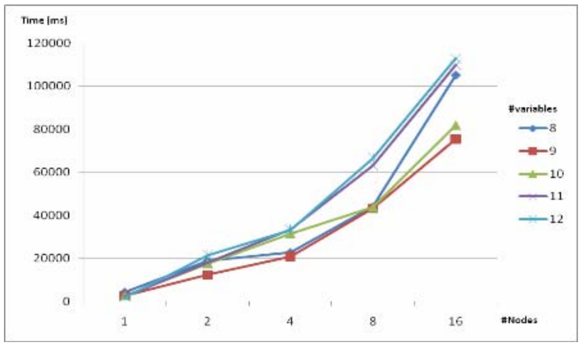

Figure 10

Figures 7 – 10: Execution time (32x32 test patterns) for *R* = *0.80, 0.85, 0.90,* and *0.95,* respectively

We also computed the number of originally solvable test patterns (**Org**), the number of cases solvable after align encode (**AE**) and the number of cases with which the algorithm has timed out (**TO**), and we report these results in Tables 1 and 2. As in some test cases, a big percentage of the search space is pruned and no solution was found in the unpruned parts, a "No solution found" is reported, which means no valid delay sequence that would make the test pattern encodable exists. The number of such patterns can be calculated by *num_of_test_patterns – (Org + AE + TO).* An example of this case can be observed in the results of Table 1 with (R=80, V=8, any cluster size) where the number of patterns that are definitely unencodable is 2. Another example can be found in Table 2 with *n=1, R=90, V=8* where the number of patterns that have no valid delay configuration is 9. In general, the algorithm's ability to decide that more test patterns to be unencodable instead of having a timeout is considered an advantage of the parallel algorithm. When we know that a test pattern has no valid delay data that makes it encodable, further attempts in finding a solution, such as increasing the timeout limit, can be eliminated.

	8			9			10			11			12		
	Org	**AE**	**TO**	**Org**	**AE**	**TO**	**Org**	**AE**	**TO**	**Org**	**AE**	**TO**	**Org**	**AE**	**TO**
80	**0**	**248**	**0**	**4**	**246**	**0**	**31**	**219**	**0**	**99**	**151**	**0**	**135**	**115**	**0**
85	**31**	**219**	**0**	**70**	**180**	**0**	**163**	**87**	**0**	**163**	**87**	**0**	**235**	**15**	**0**
90	**189**	**81**	**0**	**214**	**36**	**0**	**233**	**17**	**0**	**247**	**3**	**0**	**249**	**1**	**0**
95	**243**	**7**	**0**	**242**	**8**	**0**	**247**	**3**	**0**	**243**	**7**	**0**	**250**	**0**	**0**

Table 1: Encodable and unencodable pattern counts (32x32 test patterns). This result set was obtained with *n = 1, 2, 4, 8, 16*

		8			9			10			11			12		
		Org	**AE**	**TO**	**Org**	**AE**	**TO**	**Org**	**AE**	**TO**	**Org**	**AE**	**TO**	**Org**	**AE**	**TO**
n=1	**90**	0	52	189	0	153	97	2	225	23	28	212	10	72	174	4
	95	134	110	6	178	71	1	209	39	2	235	15	0	245	5	0
n=2	**90**	0	82	168	0	178	72	2	231	17	28	212	10	72	174	4
	95	134	113	3	178	71	1	209	40	1	235	15	0	245	5	0
n=4	**90**	0	113	137	0	191	59	2	234	14	28	213	9	72	174	4
	95	134	113	3	178	72	0	209	40	1	235	15	0	245	5	0
n=8	**90**	0	142	108	0	207	43	2	237	11	28	217	5	72	174	4
	95	134	114	2	178	72	0	209	40	1	235	15	0	245	5	0
n=16	**90**	0	163	87	0	220	30	2	239	9	28	217	5	72	174	4
	95	134	114	2	178	72	0	209	40	1	235	15	0	245	5	0

Table 2: Encodable and unencodable pattern counts (64x64 test patterns)

It is quite common to employ a second test application phase subsequent to scan compression mode. In the first phase, encodable patterns (T_{org}) are delivered into the shorter scan chains (of length *depth_short*) through a decompressor, while in the latter phase, shorter scan chains are concatenated into fewer longer ones (of length *depth_long*) and the remaining test patterns (T - T_{org}), which were unencodable in the first phase, are inserted into these longer scan chains, wherein the decompressor block is bypassed. Thus the per pattern shift cycles increases in the second phase.

We utilize the results from Table 1 and 2 in order to compare the test data volumes of the architecture without versus with Align-Encode in a two-phase test application process. No compression of test patterns (all serial application) is referred to as the base case in our computations. We provide percentage reductions in test data volume delivered by the decompressor alone and by the decompressor along with Align-Encode, both with respect to the base case. Test data volume of the base case is computed as a product of the number of scan chains (*chains*), the number of test patterns (T), which is 250, and the scan depth (*depth_long*). Test data volume of the first phase in the case of no Align-Encode is computed as the product of the number of scan-in channels (V), the number of encodable patterns (T_{org}) and the scan depth (*depth_short*). Test data volume of the first phase of Align-Encode is also computed similarly, except that the penalty incurred due to delay information and to one additional cycle per pattern are also included for the patterns that became encodable due to Align-Encode (T_{AE}). The second phase test data volume is computed identically for both cases; it is computed as the product of the number of scan chains (*chains*), the number of unencodable test patterns (without AE: $T - T_{org}$, with AE: $T - T_{org} - T_{AE}$), and scan depth (*depth_short*):

$TDV_{base} = all\_patterns \,.\, chains \,.\, depth\_long$
$TDV_{NO\_AE} = (T - T_{org}) \,.\, chains \,.\, depth\_long \;+\; T_{org} \,.\, V \,.\, depth\_short$
$TDV_{AE} = (T - T_{org} - T_{AE}) \,.\, chains \,.\, depth\_long \;+\; T_{org} \,.\, V \,.\, depth\_short \;+\; T_{org} \,.\, (V+1) \,.\, (depth\_short+1)$

We are interested in the finding the reduction in test data volume without versus with applying Align-Encode then find the difference between the two reductions to analyze how this difference changes as the number of nodes is increased. To find the reduction in test data volume we use the following formula:

$TDV\text{-}Reduction_{NO\_AE} = (TDV_{base} - TDV_{NO\_AE}) / TDV_{base}$
$TDV\text{-}Reduction_{NO\_AE} = (TDV_{base} - TDV_{AE}) / TDV_{base}$

Appling these formula while using the pattern counts obtained in Tables 1 and 2, we can summarize the test data volume reductions for our test cases as shown in Tables 3 and 4 that show the reduction value with (AE) and without applying Align-Encode (NO_AE). Tables 5-7 summarize the test data volume reduction differences between decompressor alone and decompressor together with Align-Encode.

	8		9		10		11		12	
	NO AE	AE	NO AE	AE	NO AE	AE	NO AE	AE	NO AE	AE
80	0.000	0.704	0.012	0.678	0.085	0.651	0.260	0.630	0.338	0.605
85	0.093	0.715	0.201	0.689	0.448	0.673	0.428	0.641	0.588	0.622
90	0.567	0.797	0.615	0.713	0.641	0.685	0.648	0.656	0.623	0.625
95	0.729	0.749	0.696	0.717	0.679	0.687	0.638	0.655	0.625	0.625

Table 3: Test data volumes reductions when using XOR decompressors with no Align-Encode vs. with Align-Encode (32x32 test patterns), Obtained for *n=1, n=2, n=4, n=8, n=16*

		8		9		10		11		12	
		NO AE	AE	NO AE	AE	NO AE	AE	NO AE	AE	NO AE	AE
n=1	**90**	0.000	0.178	0.000	0.515	0.007	0.750	0.093	0.779	0.234	0.786
	95	0.469	0.846	0.612	0.851	0.705	0.834	0.778	0.827	0.796	0.812
n=2	**90**	0.000	0.281	0.000	0.599	0.007	0.769	0.093	0.779	0.234	0.786
	95	0.469	0.856	0.612	0.851	0.705	0.837	0.778	0.827	0.796	0.812
n=4	**90**	0.000	0.387	0.000	0.643	0.007	0.779	0.093	0.783	0.234	0.786
	95	0.469	0.856	0.612	0.854	0.705	0.837	0.778	0.827	0.796	0.812
n=8	**90**	0.000	0.487	0.000	0.697	0.007	0.789	0.093	0.795	0.234	0.786
	95	0.469	0.860	0.612	0.854	0.705	0.837	0.778	0.827	0.796	0.812
n=16	**90**	0.000	0.559	0.000	0.740	0.007	0.796	0.093	0.795	0.234	0.786
	95	0.469	0.860	0.612	0.854	0.705	0.837	0.778	0.827	0.796	0.812

Table 4: Test data volumes reductions when using XOR decompressors with no Align-Encode vs. with Align-Encode (64x64 test patterns)

	8	9	10	11	12
80	0.704	0.667	0.565	0.370	0.267
85	0.622	0.488	0.225	0.213	0.035
90	0.230	0.098	0.044	0.007	0.002
95	0.020	0.022	0.008	0.017	0.000

Table 5: Test data volumes reduction differences between the cases when no Align-Encode applied and when applying Align-Encode (32x32 test patterns), Obtained for *n=1, n=2, n=4, n=8, n=16*

R=0.90	8	9	10	11	12
1	0.178	0.515	0.743	0.687	0.552
2	0.281	0.599	0.763	0.687	0.552
4	0.387	0.643	0.773	0.690	0.552
8	0.487	0.697	0.783	0.703	0.552
16	0.559	0.740	0.789	0.703	0.552

Table 6

R=0.95	8	9	10	11	12
1	0.377	0.239	0.129	0.049	0.016
2	0.387	0.239	0.132	0.049	0.016
4	0.387	0.242	0.132	0.049	0.016
8	0.391	0.242	0.132	0.049	0.016
16	0.391	0.242	0.132	0.049	0.016

Table 7

Table 6 - 7: Test data volumes reduction differences between the cases when no Align-Encode applied and when applying Align-Encode (64x64 test patterns) where R=0.90, R=0.95 respectively

To visualize the effect of changing different factors on test data volume, we plot in Figure 11 test data volume reduction difference of Align-Encode and no Align Encode versus the number of processing nodes (*n*) for R=0.90 and 64x64 test patterns:

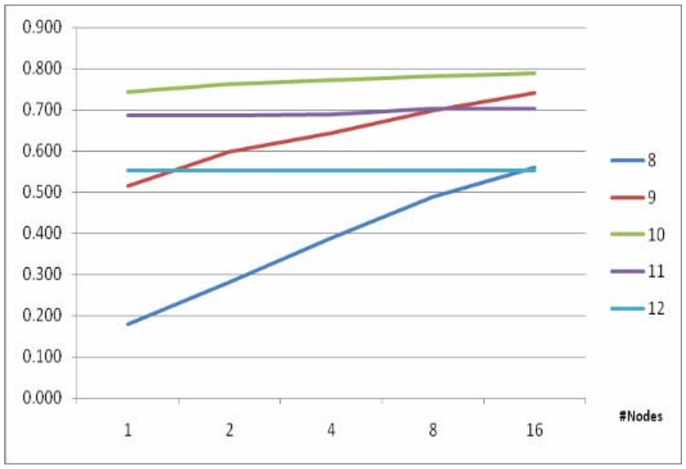

FIGURE 11: THE TEST DATA VOLUME REDUCTION DIFFERENCE BETWEEN APPLYING AE AND NOT APPLYING AE WHERE *R=0.90* (64X64 TEST PATTERNS). THE DIFFERENCE IS GENERALLY INCREASING AS THE NUMBER OF NODES INCREASES. THE HIGHEST INCREASE RATE IS NOTICED WHEN *V=8*

VII. REMARKS AND EXPECTATIONS

The results can be analyzed from two different perspectives: with respect to run-time and quality of results. The latter aspect directly hinges on the former one, as there is a timer mechanism built in our algorithm. With shorter run-times, the chances of finding a solution within preset time constraints are higher, improving the quality of results.

The proposed solution is based on solving linear equations, and thus, on the repetitive execution of the Gaussian Elimination Technique with different delay configurations. The solvability of systems is directly dependent on the number of equations and the number of free variables. In our context, the number of equations equals at most the number of scan chains. However, equations with a don’t care bit on the right hand side are dropped. So, the higher the don’t care bit ratio R, the fewer the equations. The number of free variables in our systems equals the number of channels V.

A system of equations can be deemed easily solvable as long as the number of free variables exceeds the number of equations. Conversely, difficult-to-solve systems are those with fewer variables than equations. In our context, increasing R or V helps in rendering the system of equations easy to solve. With higher values of R or V, possibly multiple solutions may exist, and the proposed algorithm searches for one of these solutions in a distributed manner. With lower values of R or V, it is possible that no solutions exist, and our algorithm ends up terminating unsuccessfully upon hitting the timer limit.

It can be noticed that for lower values of R, the run-time time generally decreases as the number of nodes increases, regardless of the number of decompressor variables. This behavior starts to reverse as the ratio R is increased; at R=0.90, the change in execution time is too small and in a non-fixed direction. At R=0.95, the trend totally changes and the execution time increases as the number of nodes is increased. This can be explained as follows: as R increases, the problem becomes easy to solve. Therefore, a small number of nodes (even one) suffice to efficiently solve such a test case. Considering the increased communication overhead with added nodes, the sequential version is expected to give the best performance.

Similarly, it can be noticed that for small V, the effect of using the parallel algorithm is advantageous in terms of runtime, since the problem is harder. Increasing V leads to easier-to-solve problems, and ultimately to barely any performance difference even with an increased number of nodes.

The results also show that the number of test patterns that gave "Timeout" decreases as the number of nodes increases. For example, at R = 90, V = 8, with 1 node 189 timeouts were obtained, while only 87 were obtained with 16 nodes. This means that more test patterns could be solved for, as we increase the number of nodes within the given time period.

Parallelism benefit doesn't just include execution time improvements, it also positively affects the number of test cases that were determined to be encodable, reducing further the test data volume, and thus improving the quality of results. This is a quite important result, as the quality of results directly determine the actual cost of testing electronic chips, and parallelism helps reduce this cost.

These results indicate that in practical scenarios where test pattern sizes exceed 64x64 to reach 128x128 and more, it is expected to get better test data volumes (smaller) which mean improvements in both the test quality and the cost. This improvement is expected to further increase as we increase the number of employed nodes.

VIII.CONCLUSION AND FUTURE WORK

In this paper, an important feature – the ability to exploit parallelism on the data level – has been observed and implemented trying to improve the performance of the original algorithm. Both the sequential and parallel algorithms have been tested on a variety of test cases. Significant speedup has been gained when applying the parallel implementation of the algorithm on relatively hard problems where deep search should be performed and just less frequent pruning is done. With such hard problems, increasing the number of nodes showed improvements in the execution time and the number of solved test patterns. As the problem gets relaxed, the sequential implementation tends to be the best among the others where the execution time is similar between the sequential version and the parallel version except for the added communication time in the parallel implementation.

In some cases where a node returns a result of "No solution", such nodes remain idle until the next test case is provided. One solution to this possible inefficiency is to apply the idea of load balancing. With load balancing, a busy working node can pass part of its work to an idle node either directly or through the controller node. Therefore, once a node got idle, it sends a "need work" request. Based on certain criteria, a busy node is chosen and part of its work is passed to the idle one. With this enhancement, node's maximum utilization would be achieved.

The current version of the algorithm has a restriction on the number of nodes. It should be a power of two. Our intention is to generalize the algorithm to work with any number of nodes n. The idea relies on making the work distribution occur in two levels. Another direction is to improve over the performance of the algorithm on large-scale configuration such as 128 or 256-chain test pattern using heuristic knowledge obtained during the process of delay bit distribution among the bits of the test patterns.

REFERENCES

[1] Sinanoglu, O.: 'Align-Encode: Improving the Encoding Capability of Test Stimulus Decompressors', International Test Conference, Santa Clara, CA, USA, paper 9.2, October 2008

[2] Sinanoglu, O.: 'Scan Architecture with Align-Encode', IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems, 2008, 27, (12), pp. 2304–2317

[3] Touba, N. A.: 'Survey of Test Vector Compression Techniques', IEEE Design and Test, 2006, 23, (4), pp. 294–303.

[4] Jas, A., Pouya, B., and Touba, N.: 'Virtual Scan Chains: A Means for Reducing Scan Length in Cores', VLSI Test Symposium, Montreal, Canada, April 2000, pp. 73–78.

[5] Hamzaoglu, I., and Patel, J. H.: 'Reducing Test Application Time for Full Scan Embedded Cores', Fault Tolerant Computing Symosium, Madison, WI, USA, June 1999, pp. 260–267.

[6] Pandey, A. R., and Patel, J. H.: 'Reconfiguration Technique for Reducing Test Time and Test Data Volume in Illinois Scan Architecture Based Designs', VLSI Test Symposium, Monterey, CA, USA, April 2002, pp. 9–15.

[7] Bayraktaroglu, I., and Orailoglu, A.: 'Decompression Hardware Determination for Test Volume and Time Reduction Through Unified Test Pattern Compaction and Compression', VLSI Test Symposium, Napa Valley, CA, USA, April 2003, pp. 113–118.

[8] Mitra, S., and Kim, K. S.: 'XPAND: An Efficient Test Stimulus Compression Technique', IEEE Transactions on Computers, 2006, 55, (2), pp. 163–173.

[9] Sitchinava, N., Samaranayake, S., Kapur, R., Gizdarski, E., Neuveux, F., and Williams, T. W.: 'Changing the Scan Enable During Shift', VLSI Test Symposium, Napa Valley, CA, USA, April 2004, pp. 73–78.

[10] Tang, H., Reddy, S. M., and Pomeranz, I.: 'On Reducing Test Data Volume and Test Application Time for Multiple Scan Chain Designs', International Test Conference, Charlotte, NC, USA, September 2003, pp. 1070–1088.

[11] Koenemann, B.: 'LFSR-Coded Test Patterns for Scan Designs', European Test Conference, Munich, Germany, April 1991, pp. 237–242.

[12] Hellebrand, S., Rajski, J., Tarnick, S., Venkatamaran, S., and Courtois, B.: 'Generation of Vector Patterns Through Reseeding of Multiple Polynomial LFSRs', International Test Conference, Baltimore, MD, USA, September 1992, pp. 120–129.

[13] Hakmi, A.-W., Wunderlich, H.-J., Zoellin, H. G., Glowatz, A., and Hapke, A.: 'Programmable Deterministic Built-in Self-Test', International Test Conference, Santa Clara, CA, USA, October 2007, pp. 1–9.

[14] Koenemann, B., Barnhart, C., Keller, B., Snethen, T., Farnsworth, O., and Wheater, D.: 'A SmartBIST Variant with Guaranteed Encoding', Asian Test Conference, Kyoto, Japan, November 2001, pp. 325–330.

[15] Xiang, D., Li, K., Sun, J., and Fujiwara, H.: 'Reconfigured Scan Forest for Test Application Cost, Test Data Volume, and Test Power Reduction', IEEE Transactions on Computers, 2007, 56, (4), pp. 557-562.

[16] Miyase, K., and Kajihara, S.: ‘Optimal Scan Tree Construction With Test Vector Modification for Test Compression’, Asian Test Symposium, Xian, China, November 2003, pp. 136–141.

[17] Rajski, J., Kassab, M., Mukherjee, N., Tamarapalli, N., Tyzser, J., and Qian, J.: ‘Embedded Deterministic Test for Low-cost Manufacturing’, IEEE Design and Test, 2003, 20, (5), pp. 58–66.

[18] W.L. Pang, K. W. Chew, Florence Choong, C.L. Tan, FPGA Realization of Open/Short Test on IC, International journal of circuits, systems and signal processing, issue 2, vol. 1, 2007, pp. 109-116.

[19] Radovan Novotny, Process Characterization and Description in Order to Reliability Assessment, International journal of circuits, systems and signal processing, issue 4, vol. 1, 2007, pp. 303-309.

[20] Taha, N., Al-Awadhi, N., Sinanoglu, O. and Al-Mulla, M., Align-Encode Delay Assignment In The Case of XOR-Decompressors: Impact of Parallel Computations, Proceedings of the 8th WSEAS International Conference on Applied Computer & Applied Computational Science (ACACOS '09), Hangzhou, China, May 20-22, 2009.

[21] Kumar, V., Grama, A., Gupta, G., Karpis, G.: Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms, Benjamin-Cummings, 1994.

Mohammed Almulla received his Master degree in Computer Science 1990, and his PhD. in Computer Science from McGill University, Montreal, Canada 1995. Currently he is associate professor at Kuwait University. His main research interests include automated theorem proving, parallel/distributed search algorithms, and heuristic search. In addition, he is an adviser for many government agencies in the area of e-Government projects and services.

Ozgur Sinanoglu Dr. Ozgur Sinanoglu has obtained his M.S. and Ph.D. degrees in the Computer Science and Engineering Department, University of California, San Diego, in 2001, and 2004, respectively. Between 2004 and 2006, he worked as a senior DfT engineer at Qualcomm, San Diego. Since Fall 2006, he is a faculty member with the Department of Mathematics and Computer Science, Kuwait University. His research interests include CAD and reliability of VLSI circuits..