

# FAULT DIAGNOSIS IN DESIGNS WITH CONVOLUTIONAL COMPACTORS

Grzegorz Mrugalski, Artur Pogiel\*, Janusz Rajski, Jerzy Tyszer\*, Chen Wang

Mentor Graphics Corporation  
8005 S.W. Boeckman Road  
Wilsonville, OR 97070, USA

\*Poznań University of Technology  
ul. Piotrowo 3a  
60-965 Poznań, Poland

## Abstract

*The paper introduces a new non-adaptive fault diagnosis technique for scan-based designs. The proposed scheme guarantees accurate and time-efficient identification of failing scan cells based on results of a convolutional test response compaction.*

## 1. Introduction

It is generally agreed that scan-based designs may allow efficient silicon debug and fault diagnosis by providing direct access to many internal nodes of the circuit under test (CUT). In fact, many of the high-performance VLSI devices might be relatively easy to diagnose due to a shallow combinational logic they feature between scan flip-flops. However, an efficient procedure is required to isolate these parts of the scan chains which are driven by cones of logic affected by actual faults. Having provided this information, one can further trace faulty components by using diagnostic techniques developed for conventional designs [24].

Several schemes for the identification of failing scan cells have been proposed in technical literature. In particular, fault diagnosis performed in a built-in self-test (BIST) environment has drawn considerable attention through years. Some of these approaches are capable of identifying error bits up to a pre-specified value [3], [4], [11], [13], [14], [20], [21] and often incur a substantial hardware overhead [5], [6], [15], [22]. Others require multiple test sessions [7], [8], [17], [25] or special algorithms to process test results [9], [12]. This is because the only data that can be used to extract fault site related information are signatures recorded in the process of test-response compaction [2], [10], [18]. There is also a separate class of techniques that rest on fault simulation [1], [23], [24]. Good summaries of the aforementioned approaches are given in [8] and [25].

In this paper, we propose a novel fault diagnosis technique which meets requirements of production test, and it is best suited for scan-based designs. This scheme takes advantage of convolutional compaction of test responses [16], which, as an embedded test solution, provides effective means of analyzing error patterns. The described approach allows for accurate identification of failing scan cells during a single test session. The essence of the

method is to select sites of failing scan cells based on weight functions associated with each scan cell and indicating how likely is that a given cell may capture faulty signals. Given the architecture of the convolutional compactor, these sites are used by a branch-and-bound technique to narrow down search to certain areas of scan chain, and then to locate scan cells driven by erroneous signals. Subsequently, the algorithm re-computes the weight functions so that they can be used to accelerate the diagnosis process for the remaining test patterns.

The paper is organized as follows. After discussing synthesis of convolutional compactors for diagnostic purposes (Section 2), we introduce a backtracking algorithm used to decide whether certain scan cells located in pre-determined scan chains can produce a given signature (Sections 3 and 4). The approach is supported by a pruning of the search space, as shown in Sections 5 and 6. A complete diagnostic process is then derived in Section 7 allowing for a time-efficient identification of failing scan cells. Finally, Section 8 presents experimental results obtained for benchmark circuits and industrial designs.

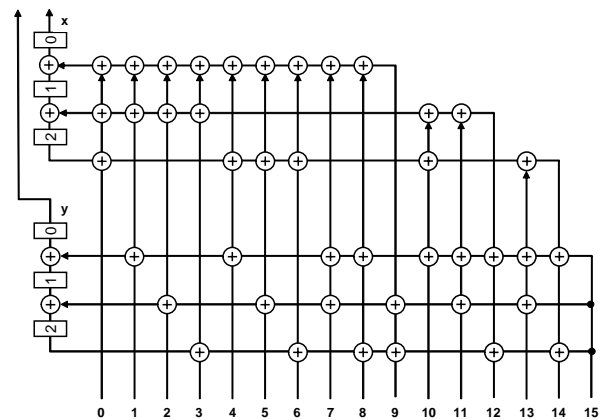


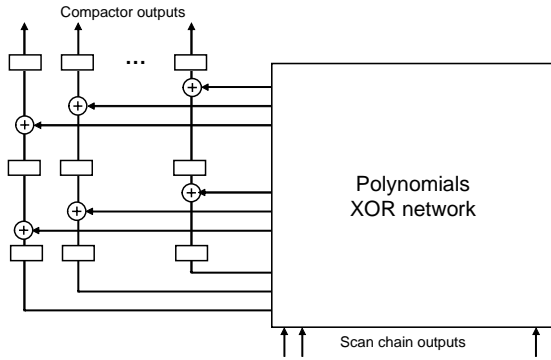
Fig. 1. Example of convolutional compactor

## 2. Convolutional compactor

Figure 1 shows an example of a convolutional compactor with 16 inputs (observing 16 scan chain outputs), two outputs, and six memory elements – three per output. Every scan cell error can reach memory elements and then outputs in three possible ways. For instance, the scan chain no. 7 is connected to bit 0 of register  $x$ , and bits 0

and 1 of register  $y$  of the compactor. As can be seen, the spatial part of the compactor consists of 6 single-output XOR networks connected to the registers by means of XOR gates interspersed between memory elements.

In general (Fig. 2), a convolutional compactor can support arbitrary compaction rate for any number of outputs, including architectures with single outputs. The total number  $M$  of memory elements, the size  $m$  of each register, and injector polynomials indicating how the scan chains are connected to the flip-flops determine the key properties of a convolutional compactor, including its ability to handle unknown states. In this paper, we assume that convolutional compactors employ polynomials of the form  $x^a + y^b + z^c$ . For example, the second scan chain in Fig. 1 employs polynomial  $P_1 = y^0 + x^1 + x^0$ , while chain 14 is connected through  $P_{14} = y^2 + y^0 + x^2$ . Note that a shifted version of  $P_1$ , i.e.,  $y^1 + x^2 + x^1$ , cannot be used because errors injected in two consecutive time frames through such polynomials would cancel each other.



**Fig. 2. Architecture of convolutional compactor**

For each scan chain, there are several alternative eligible polynomials. These polynomials correspond to injectors with all taps shifted along the compactor memory elements with each tap remaining connected to the same sub-register. Only one polynomial from each group can be used. Otherwise 2-error masking may occur. It turns out [16] that the best performance is achieved by choosing the polynomials randomly. The random selection of polynomials also helps in balancing the XOR networks.

Within each group of alternative polynomials (i.e., among shifted replicas of a given polynomial) we select one in such a way that the resulting compactor features a highly uniform use of its memory elements. This is accomplished by employing a stage fan-in histogram which guides the synthesis process. Given a compactor memory element, a corresponding entry of the histogram records the frequency of cell use. In order to connect a new injector polynomial, all possible candidate locations are examined by computing the value of a cost function. It is equal to the sum of the current histogram entries corresponding to the successive polynomial taps. The location

with the smallest cost is chosen eventually, and all relevant histogram entries are updated. As a result, virtually the same number of injector polynomial taps drives all memory elements of the convolutional compactor.

It is also assured that each polynomial features taps sufficiently spaced, so that the compactor can handle efficiently burst errors injected from adjacent scan cells. It also provides a better distribution of multiple errors and reduces masking of multiple even errors. In all experiments reported in this paper, the minimum separation  $d$  between two consecutive taps (connected to the same register) of each polynomial was chosen to be at least  $0.25m$ , while the total span between the boundary taps of the same polynomial was kept below the value of  $0.75m$ .

Since any convolutional compactor is a linear circuit, its behavior can be analyzed based on *error test responses* it receives and *error signatures* it produces. The error test response is defined as  $\mathbf{E} = \mathbf{R}_{ff} + \mathbf{R}_f$ , where  $\mathbf{R}_{ff}$  and  $\mathbf{R}_f$  are fault-free and faulty test responses [25], respectively, and  $+$  denotes the bit-wise XOR operation. Similarly, the error signature  $\mathbf{S} = \mathbf{S}_{ff} + \mathbf{S}_f$ , where  $\mathbf{S}_{ff}$  and  $\mathbf{S}_f$  are fault-free and faulty signatures, respectively. In the rest of the paper, we will only consider error test responses  $\mathbf{E}$  and error signatures  $\mathbf{S}$  unless otherwise stated. The  $j$ -th scan cell located in  $i$ -th scan chain may produce an *error print* of the following form:  $x^{a+j} + y^{b+j} + z^{c+j}$ , where  $P_i = x^a + y^b + z^c$  is an injector polynomial associated with scan chain  $i$ .

In general, the same error signatures can be caused by different error prints. Indeed, when injecting errors into a signature, scan cells interact with each other on the compactor registers. Consequently, certain error signals can be masked, thus leading to ambiguity in selection of the actual scan sites catching the errors. Even if there is no error masking, one can still determine different error prints which are equivalent causes of a recorded signature. Therefore, an important decision that has to be made during the synthesis of a convolutional compactor targeting diagnostic applications is selection of its size  $M$  as it impacts directly the diagnostic resolution [16]. Fortunately, in the vast majority of cases, employing a sufficiently large compactor can circumvent the phenomenon of having indistinguishable solutions. This is best illustrated in Tables I and II.

**Table I. The average error masking (370 faulty chips)**

$d$	$M$						
	48	64	96	128	160	192	256
8	2.47	2.58	2.67	2.78	2.83	2.82	2.84
16	-	2.66	2.75	2.82	2.84	2.85	2.89
24	-	-	2.81	2.84	2.85	2.87	2.91
32	-	-	-	2.85	2.87	2.89	2.91
40	-	-	-	2.86	2.89	2.90	2.91
48	-	-	-	-	2.87	2.89	2.91
56	-	-	-	-	-	2.88	2.92

Table I summarizes results of analysis of fail log information collected during production scan testing of an industrial design. In this analysis, simulation of the actual silicon failing responses was carried out on  $M$ -bit single-output convolutional compactors working with 100 scan chains, each 1050-bit long. Each entry in the table was then obtained by dividing the total number of erroneous signals occurring in signatures by the total number of failing scan cells producing these signatures. Note that no masking would give the value of 3.0, since only polynomials with fan-out 3 are employed. As can be seen, increasing both the size  $M$  of the compactor and the separation  $d$  between polynomial taps results in a diminishing error masking.

A more detailed breakdown of a mapping between failing scan cells (for the same experimental data) and the corresponding erroneous signals is given in Table II. Here the total number of cases in which a given number  $F$  of scan cells catch errors is further split into scenarios that produce 0, 1, 2, and 3 ones in signatures. The results are presented for two single-output compactors ( $M = 64$  and 128) and compression ratio of 128x. Clearly, the increase in the compactor size results in error signatures having more erroneous signals.

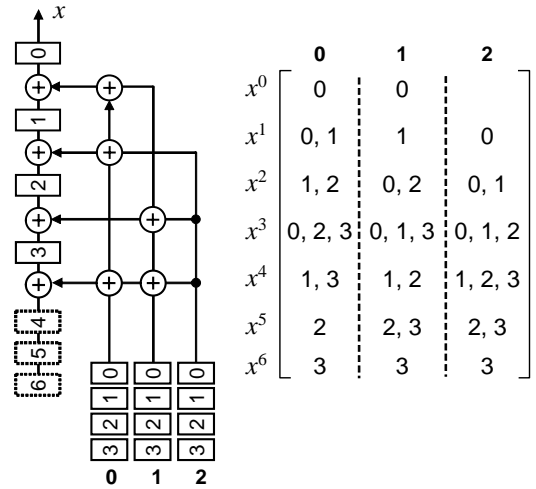
**Table II. Mapping of scan errors to signatures (%)**

$F$	$M = 64$				$M = 128$			
	0	1	2	3	0	1	2	3
1	0	0	0	100	0	0	0	100
2	0	0	2.19	97.81	0	0	.94	99.06
3	0	0	3.53	96.47	0	0	4.51	95.49
4	0	.3	5.85	93.84	0	.12	3.90	95.98
5	0	.61	8.24	91.15	0	.30	5.09	94.61
6	.08	.89	12.25	86.78	0	.23	6.36	93.41
7	.03	2.51	16.48	80.98	0	.49	10.94	88.57
8	.17	3.19	21.03	75.60	.04	.52	14.53	84.91
9	.05	3.99	23.69	72.27	.05	1.46	16.52	81.97
10	.25	6.15	28.55	65.05	0	1.30	19.40	79.30

### 3. Convolutional compactor representation

A simple data structure presented in this section is a starting point for implementing a diagnosis algorithm developed in the following parts of the paper. For the sake of simplicity, we will focus here on single-output compactors, though any convolutional compactor with more than one output can be easily represented in a similar manner. Let  $S = M + L - 1$  be the signature size, where  $L$  is the length of the longest scan chain. Let also  $N$  represent the number of scan chains. The proposed data structure is based on an  $S \times N$  matrix, with the entry in row  $s$  and column  $n$  defined to be a list of scan cells driving bit  $s$  of the signature and being hosted by scan chain  $n$ . Figure 3 depicts an example for a 4-bit compactor driven by 3 scan chains, each 4-bit long. The employed injector poly-

nomials, associated with the scan chains, are  $x^3 + x^1 + x^0$ ,  $x^3 + x^2 + x^0$ , and  $x^3 + x^2 + x^1$ .



**Fig. 3. An example of matrix representation**

As can be seen, we keep track of all the scan cells capable of changing each signature position on the linked lists that are associated with that position. We maintain a matrix of lists so that, given a signature bit and a scan chain, we can immediately access the respective list. The order in which items (scan cells) appear on the lists depends on certain factors described in section 5. It is worth noting that any list is comprised of at most three elements. For instance, the first cell  $c_0$  located in scan chain 2 produces the following error print:  $x^3 + x^2 + x^1$ . Then error prints produced by cells  $c_1$  and  $c_2$  of the same scan chain are  $x^1(x^3 + x^2 + x^1) = x^4 + x^3 + x^2$  and  $x^2(x^3 + x^2 + x^1) = x^5 + x^4 + x^3$ , respectively. Clearly, cells  $c_0$ ,  $c_1$  and  $c_2$  will appear on the list (3,2) associated with bit  $x^3$  of the signature and scan chain 2. The same reasoning applies to cells from the remaining scan chains, so as a result, bit  $x^3$  can be modified by nine different scan cells in total (see Fig. 3).

An auxiliary data structure is also used to represent the scan architecture. It is an  $N \times L$  matrix  $Scan\_chains$  of the scan cells, where each entry determines all signature bits that can be affected by a given scan cell (an injector list). Moreover, the same entry carries some additional information including the number of signature errors that is actually caused by the corresponding scan cell and its status as far as a final diagnostic solution is concerned. These data will be used to make decisions based on their values as shown in the subsequent sections.

### 4. The basic diagnostic search

Given an error signature, the basic task of identifying the failing scan cells can be solved by using a depth-first search. The graph to which the depth-first search is applied is a dynamically generated search tree with vertices that represent various candidate scan cells altogether with an error signature they yield. The root consists of the

originally recorded signature with no scan cells selected yet. The search tree is generated dynamically by selecting a scan cell which drives the rightmost error signature bit set to one, updating accordingly the whole signature and moving from top to bottom by choosing next appropriate scan cells until the presence of all ones in the signature is justified. When it is impossible to select a new scan cell the algorithm returns to the parent node in the tree and tries another scan cell. Clearly, the presented approach is a particular form of a backtracking search whose principles are presented in this section, while several techniques used to significantly speed-up the diagnosis process will be described in the subsequent sections.

We implement the backtracking solution of the convolutional diagnosis recursively. The key function is *search*. When it is invoked at a certain level, a number of scan cells have already been chosen, and the corresponding content of the error signature is equal to a bit-wise sum modulo 2 of the original (initial) error signature and ones provided through the injector polynomials associated with the selected cells. The function *search* tries then to select a new scan cell from a list of *drivers* associated with the rightmost signature bit set to 1 such that the error signature becomes eventually the all-zero vector. If it is successful, the algorithm can be terminated at this point:

```

search ( ) {
    if (n_Ones == 0) return true
    done = false
    ptr = Signature [first_one]. drivers
    while (not done and ptr ≠ null){
        scan = ptr → scan
        cell = ptr → cell
        ptr = ptr → next
        if (not Scan_chains[scan][cell]. selected){
            used_cells = used_cells + 1
            Scan_chains[scan][cell]. selected = true
            update (scan, cell)
            if (search ( )) done = true
            else {
                used_cells = used_cells - 1
                Scan_chains[scan][cell]. selected = false
                update (scan, cell)
            }
        }
    }
    return done
}

```

It is worth noting that if *search* does not succeed in finding a complete list of failing scan cells, it cancels decisions made hitherto (in particular it moves back the error signature to its previous status) and tries another scan cell as a possible candidate. If all driver cells have been examined, the function backtracks by returning to its caller which repeats the same steps as described here. The final

backtracking solution (recorded in matrix *Scan\_chains*) is obtained by calling *search* with *n\_Ones* set to the actual number of ones in the received signature, and *first\_one* set to the location of the rightmost erroneous signal in the same signature.

An auxiliary function *update (scan, cell)* uses the injector polynomial associated with the indicated scan cell to modify the content of the signature. The same function keeps track of the current value of variable *n\_Ones*, which is updated accordingly to a new value of each signature bit being changed. Finally, a new location of the rightmost one in the signature is determined, especially if the current rightmost one assumes the value of zero. Thus, function *update* can be defined in the following fashion:

```

update (scan, cell) {
    for tap = 1 to 3 {
        bit = Scan_chains[scan][cell].injectors[tap]
        Signature[bit] = Signature[bit] ⊕ 1
        if (Signature[bit] == 0){
            n_Ones = n_Ones - 1
            locate (first_one)
        }
        else {
            n_Ones = n_Ones + 1
            if (bit < first_one) first_one = bit
        }
    }
}

```

*Example:* Consider a convolutional compactor driven by 3 scan chains as shown in Fig.3. Let a signature be produced by errors arriving from cells (0,0), (1,1) and (1,2). Their error prints are  $x^3 + x^1 + x^0$ ,  $x^4 + x^3 + x^1$ , and  $x^5 + x^4 + x^2$ , respectively. Thus, the resulting signature is given by  $x^3 + x^1 + x^0 + x^4 + x^3 + x^1 + x^5 + x^4 + x^2 = x^5 + x^2 + x^0$ . We begin at the root of the search tree with this signature. Since  $x^0$  is the rightmost nonzero signature bit, we try the first scan cell which drives that bit, i.e., cell (0,0) – see again Fig. 3. Calling *update* function yields a new form of the signature  $x^5 + x^2 + x^0 + x^3 + x^1 + x^0 = x^5 + x^3 + x^2 + x^1$ . Bit  $x^1$  is driven by cells (0,0), (0,1), (1,1) and (2,0). We pick (0,1) as (0,0) has already been selected. It results in the following signature:  $x^5 + x^4 + x^3$ . This process continues in a similar fashion until a solution is found or no more cells can be used to cancel the first nonzero bit of the signature. In the latter case, the algorithm has to backtrack and carry on till obtaining the all-zero vector.

Simple as it is, the presented backtracking algorithm is in fact a fairly robust way of pointing out scan cells that may produce a given signature. In fact, since the diagnostic reasoning (as described above) is basically dependent on the ones present in the signature, the same algorithm could be used to recreate all possible scan cell configurations that might yield a given signature. However, with the increasing size *M* of the compactor, the number of

scan chains, and the number of scan cells, the respective search tree grows quickly, thus leading to unacceptable processing time. Hence, certain constraints are required to allow for pruning of the solution space, thus resulting in a time-efficient and yet accurate diagnostic process.

## 5. The weight functions

As shown in the previous sections, one can use a simple backtracking algorithm to determine failing scan cells. However, because of its inherent complexity, our primary objective is now to accelerate the diagnostic process so that a solution, which matches the actual locations of faulty scan cells can be obtained as quickly as possible. This goal can be achieved by deploying a measure of likelihood that particular scan cells can capture the actual errors, and subsequently by pruning the search space.

In order to determine the most likely locations of failing scan cells, the scan cell selection process is guided by weight functions associated with every scan cell. A weight of a given cell is proportional to the probability that this cell belongs to the actual error pattern. In general, the weight of cell  $c_i$  is given by the following formula:

$$W(c_i) = B_i S_i (1 + C_i)$$

Here  $B_i$  accounts for the number of ones that the cell may set in a signature, while the remaining two factors account for a proven presence of the same cell in other error patterns obtained for test vectors applied earlier.

Computation of coefficients  $B_i$  is based on individual counters associated with each scan cell. We call this number the weight of the scan cell. It is defined as the total number of ones in a signature which are effectively set by an error arriving from cell  $c_i$ . The value of  $B_i$  is then determined by summing the weights over all scan cells having the same time frame as that of cell  $c_i$ . The use of cumulative statistics  $B_i$  rather than individual cell weights results from an observation that the latter ones may often misguide the scan cell selection process. This is best illustrated by the next example.

*Example:* Consider again the convolutional compactor shown in Fig. 3 with the same error pattern comprised of three scan cells: (0,0), (1,1), and (1,2). Computation of the weights yields the following values:

<b>1</b>	<b>2</b>	1	4
1	<b>0</b>	1	2
2	<b>2</b>	1	5
0	1	1	2

As can be seen, the weights associated with cells that capture errors are not necessarily the largest ones (counters corresponding to the failing cells are printed in bold). If used directly to indicate the most likely sites of errors, these numbers would lead to a computationally less efficient diagnostic process. This is why the algorithm does

not distinguish particular scan cells as far as the same time frame is concerned. Nevertheless, the priority is given to those cells that belong to time frames with the largest weights.

The scan cell selection process can be further improved by allowing coefficients  $B_i$  to carry information obtained during signature preprocessing phase. This phase is aimed at detecting several generic scenarios in which scan cells, when injecting errors into a signature, interact with each other in the compactor register in particular ways. Therefore, if a scan cell is recognized as a likely part of the actual failing cell set, its individual weight is increased, accordingly. A detailed analysis shows that errors occur in signatures mostly due to the following origins:

- a single scan cell produces three ones due to an error print of the form:

$$E_1 = x^a + y^b + z^c$$

- two scan cells yield four ones as their error prints assume the following forms:

$$\begin{aligned} E_1 &= x^a + y^b + z^c \\ E_2 &= x^a + y^d + z^e \end{aligned}$$

- two scan cells output two ones after mutual masking of the remaining elements of their error prints:

$$\begin{aligned} E_1 &= x^a + y^b + z^c \\ E_2 &= x^a + y^b + z^d \end{aligned}$$

- three scan cells produce five ones for they interact as shown below:

$$\begin{aligned} E_1 &= x^a + y^b + z^c \\ E_2 &= x^a + y^d + z^e \\ E_3 &= x^f + y^b + z^g \end{aligned}$$

- three scan cells yield three ones provided their error prints are aligned as follows:

$$\begin{aligned} E_1 &= x^a + y^b + z^c \\ E_2 &= x^a + y^d + z^e \\ E_3 &= x^f + y^d + z^c \end{aligned}$$

- three scan cells can also leave a single one in a signature either as follows:

$$\begin{aligned} E_1 &= x^a + y^b + z^c \\ E_2 &= x^e + y^b + z^d \\ E_3 &= x^e + z^c + z^d \end{aligned}$$

- or in the following fashion:

$$\begin{aligned} E_1 &= x^a + y^b + z^c \\ E_2 &= x^a + y^b + z^d \\ E_3 &= x^a + z^c + z^d \end{aligned}$$

- four scan cells, among various cases, can produce six ones by masking the entire error print of one of them, i.e., by using the following pattern:

$$\begin{aligned} E_1 &= x^a + y^b + z^c \\ E_2 &= x^a + y^d + z^e \\ E_3 &= x^f + y^b + z^g \\ E_4 &= x^h + y^i + z^c \end{aligned}$$

The fault diagnosis results obtained for the previous test patterns are also considered in the weight computation. A large body of experimental evidence shows that for many stimuli faults propagate to the same scan cells or, at least, to the same scan chains. One can take advantage of this observation by preferring cells located in scan chains hosting cells already declared as failing ones. The coefficients  $S_i$  are employed to account for these events in the following manner:

- $S_i = B_i$ , if a scan chain containing cell  $c_i$  has appeared in the previous solutions more than a pre-specified number of times  $T$  ( $T$  must be greater than 1),
- $S_i = 1$ , otherwise.

The value of threshold  $T$  is determined as a certain fraction of all appearances of scan chains identified so far as receiving erroneous signals. All experiments reported in this paper assume that  $T = 0.02A$ , where  $A$  is a sum of occurrences in the earlier solutions over all scan chains.

Finally, we take account of individual scan cells by using coefficients  $C_i$ . In principle, these components indicate the number of times a given cell was found to be a part of solutions for test patterns applied earlier. However, in order to prevent these factors from an infinite grow, they are further divided by the total number of test patterns deployed so far.

It is important to note, that the weight information with respect to coefficients  $S_i$  and  $C_i$  is incrementally updated after successive solutions are obtained, resulting in a very efficient diagnostic procedure. Indeed, the proposed approach resembles a process of learning from experience, where the diagnostic algorithm modifies its processing based on newly acquired data.

## 6. Pruning the search space

The depth-first backtracking algorithm described in Sections 4 may traverse the entire search space. However, it often appears that a given node in the search tree does not lead to a solution because all its successors are infeasible. In such cases, this node and its successors need not be considered. As a result, we can *prune* the solution tree, thereby reducing the number of options to be considered.

*Selection of scan cells.* There is a number of factors that can be taken into account when implementing a branch-and-bound type of convolutional diagnosis. One of the most straightforward techniques is to limit arbitrarily the number of scan cells that may capture errors. Let  $max\_cells$  be such a number. Then, as soon as the number of selected scan cells (recall that this quantity is represented by variable  $used\_cells$ ) exceeds the user-set value of  $max\_cells$ , the diagnostic algorithm backtracks. Moreover, even if the number of selected cells remains less than  $max\_cells$ , it may become apparent that the number of ones ( $n\_Ones$ ) still present in the current sig-

nature is too large given the number of cells  $C$  already chosen. If those cells were supposed to reduce the number of ones to at least  $P$ , but this is not the case, the algorithm may backtrack even earlier. The beginning of *search* function can be therefore rewritten as follows:

```
search () {
    if (n_Ones == 0) return true
    if (used_cells > max_cells) return false
    if (used_cells > C and n_Ones > P) return false
    done = false
    ...
}
```

The weight function, as described in the previous section, can be also used efficiently to prune parts of the search space which are unlikely to contain a solution. In fact, it suffices to use the following instruction:

```
if (Scan_chains[scan][cell].weight < min) continue
```

which should be placed right after designating a next scan cell as a possible part of the failing cells set (inside the *while* loop). If the cell weight is smaller than the value of the acceptable minimum, then the algorithm carries on by selecting the next cell from the same list of drivers unless the list is already exhausted, what leads to backtracking.

*Selection of scan chains.* One of the most effective means of pruning the search space is to limit the number of scan chains that host the failing scan cells. It can be done in a manner similar to that of scan cell selection discussed earlier. Within the framework of the algorithm presented in Section 4, however, it is also possible to designate directly certain scan chains, and to confine the entire selection of scan cells to these locations only. The matrix representation of Fig. 3 can be used conveniently to implement this approach. In the case of failure, one can choose another subset of scan chains of the same size, or the number of target chains can be increased, as demonstrated in the next section.

*Zero zones.* As illustrated in the last example, errors injected into compactor registers may cancel each other, thus leaving a diagnostic procedure even with cells featuring zero weights. These cells cannot be ignored, however, as they may comprise the actual solution. On the other hand, there are scan cells with weights equal to zero which are likely to never be a part of any error pattern. It applies primarily to cells located in time frames where all cells have zero weights, and adjacent frames comprise scan cells with zero weights, too. These areas are called *zero zones*, and their occurrence is a result of error clustering, i.e., a frequent phenomenon where only neighboring scan cells are affected by faults. In order to differentiate the zero zones from other scan cells with zero weights, all cell weights in these areas assume the value of  $-1$ . Subsequently, the algorithm omits a large portion of scan cells and backtracks using instructions shown earlier.

## 7. The fault diagnosis algorithm

Finding the set of failing scan cells proceeds by iteration with respect to results obtained when applying successive test patterns. Starting from the simplest cases, the algorithm gradually attempts to solve more complex problems based on information gained in the previous steps. Here is how the entire algorithm looks all together:

1. For each test pattern  $t_i$  applied to a circuit under test record the corresponding signature  $S_i$ .
2. For each signature  $S_i$  obtained in step 1, determine the value of variable  $n\_Ones$  and the initial location of the rightmost one in  $S_i$ .
3. Sort signatures  $S_i$  in an ascending order with respect to the number of ones they feature.
4. Set  $i = 1$ .
5. Select the first scan chain as a possible location of failing scan cells.
6. Given the number of ones in  $S_i$ , set the initial number of failing scan cells to  $\lceil n\_Ones / 3 \rceil$ .
7. Invoke function *search*.
8. If there is no solution, increase the maximum number  $max\_cells$  of failing scan cells provided it is not greater than a pre-selected upper limit yet, and go to step 7. If a solution exists, then store it, update the weight information, increase  $i$ , and proceed to step 5 unless all error patterns have been already examined.
9. If parameter  $max\_cells$  exceeds the upper limit, and a solution has not been found, determine a next combination of scan chains that may host failing scan cells and go to step 6.

The major steps of the above procedure are explained in the following paragraphs.

*Initial phase* (steps 1 – 3). Once all signatures for a given test set are collected, a preprocessing phase is carried out. The number of ones occurring in each signature provides a rough estimation of error pattern complexity, and thus a degree of difficulty one may face when trying to locate the failing scan cells. Indeed, various experiments indicate that in a vast majority of cases more complex error patterns imply more ones in the corresponding signatures (see also Table I). Given a self-learning behavior of the proposed scheme, it is therefore desirable to have all signatures put in order by the number of ones they feature. Consequently, diagnosis can be first performed for cases in which an expected number of failing scan cells is relatively small. Subsequently, resting on earlier results, the scheme may try to identify components of error patterns with a larger number of failing cells.

*Beginning of the main loop* (steps 4 – 6). In this phase, a single scan chain is chosen as a first possible location of failing scan cells. In a sequel, the algorithm will examine other scan chains appearing in various configurations as shown in step 9. However, if scan cells that catch errors

are located in a relatively small number of chains, they can be quickly identified due to a very aggressive pruning of the search space. Furthermore, the principle exemplified in the algorithm is to maintain a list of the most likely locations of cells receiving errors, which can be subsequently used to get more complex solutions. Clearly, the reason for doing so is essentially heuristic: experience shows that the same scan cells affected by faults show up in many error patterns of the increasing complexity. It is also worth noting that the minimal number of scan cells that must be involved in producing a given number of ones cannot be lesser than  $\lceil n\_Ones / 3 \rceil$ . This is why the initial value of  $max\_cells$  is set to this quantity.

*The actual diagnosis phase* (step 7). Here the main algorithm calls the basic diagnostic procedure whose functionality is discussed in the previous sections altogether with its use of pruning techniques. Note that *search* is executed for a given set of scan chains and a fixed number of scan cells that cannot be exceeded. Other parameters include variables  $C$  and  $P$  (see Section 6) which, for all experiments reported in this paper, assume the values of  $0.4 max\_cells$  and  $0.7 n\_Ones$ , respectively.

*Choosing new parameters* (steps 8 – 9). It is possible that for a given number of scan cells located in a predetermined scan chains there is no adequate error pattern that could produce signature  $S_i$ . In such a case, the algorithm increases by one the maximum number of scan cells that should be considered and invokes *search* function again. Note that at this stage the list of scan chains that may host failing scan cells remains unchanged. This process continues until the maximum number of scan cells reaches its upper limit, beyond which the processing time is regarded as unacceptable.

If the algorithm still fails to identify a set of scan cells receiving errors, then it moves back to step 6, which is equivalent to beginning the scan cell selection process anew. This is preceded, however, by selecting a new set of scan chains that now will form a base for the next iteration comprising steps 7 and 8. The new candidate scan chains are determined as follows. Suppose, there are  $s$  scan chains that have been identified hitherto (from the previous test patterns) as locations of errors. Every time the algorithm reaches step 9, it selects a next subset of  $s$  scan chains with  $k$  members, for  $k = 1, 2, \dots, r$ . In other words, all  $k$ -subsets of the  $s$ -set, taken over all possible values of  $k$  up to  $r$ , are examined. In the experiments presented in this paper  $r < 4$  was found to suffice to ensure a high quality of diagnosis. Using this technique one may assure that a preference is given to those scan chains that are regarded as the most likely parts of the final solution based on their occurrence in the former solutions.

Clearly, an error pattern may consist of scan cells located in scan chains not recorded previously, and therefore not targeted in the manner shown above. Thus, once all  $k$ -

subsets are examined, we start checking single scan chains from the remaining  $N - s$  scan chains for possibility of having error patterns there, and subsequently the same chains are added to all  $k$ -subsets examined earlier to form new  $(k + 1)$ -subsets of scan chains. The same approach is also used for all pairs of scan chains taken over all  $N - s$  chains. If a newly examined scan chain occurs a predetermined number of times in identified error patterns (see Section 5), it eventually joins the  $s$ -set of scan chains forming a new  $(s + 1)$ -set. Note that initially  $s = 0$ , and the algorithm attempts to locate failing scan cells in one (or two) of the  $N$  scan chains with no additional guidance.

It is worth noting that the algorithm presented above can be modified so that in many cases the average successful diagnosis time remains bounded as the number of ones in a signature increases. This is based on a quite common fact that erroneous signals occur in signatures as several clusters separated by a large number of bits not affected by faults. Consequently, the proposed algorithm can be performed individually for each cluster rather than for the whole signature. In order to isolate such clusters quickly, one can use the scan cell counters (weights). If all counters associated with a given time frame (or consecutive time frames) are equal to zero, then the corresponding error-free positions in the signature can be regarded as indicative of how to decompose the diagnosis process.

## 8. Experimental results

The primary objective of the experimental analysis presented in this section was to study the feasibility of the proposed scheme. The prime target in all experiments was the ability of the scheme to recreate, based on collected test responses, original sites of failing scan cells for successive test patterns. The experiments were conducted on several large ISCAS'89 benchmark circuits and industrial designs. Their characteristics, including the number of gates and scan cells, as well as the number of scan chains are given in Table III. A commercial ATPG tool generated test sets used in the experiments. A diagnostic coverage was employed as a basic figure of merit to assess performance of the scheme.

Given a test set  $T$ , *diagnostic coverage* is defined as the percentage of faults that are diagnosable [25]. A fault is said to be diagnosable if all scan cells affected by this fault can be correctly identified using a diagnostic algorithm. Note that the test set can detect every fault several times, and subsets of affected scan cells may differ each time. Let  $T(f_i) \subset T$  be a subset of test patterns that detect fault  $f_i$  and yield different error patterns. Let also  $C(f_i, t_j)$  be the set of scan cells affected by fault  $f_i$  when test pattern  $t_j \in T(f_i)$  is applied, and  $D(f_i, t_j)$  be the set of failing scan cells as determined by the diagnostic procedure. The diagnostic coverage can be then measured in the following two different ways.

*Basic diagnostic coverage.* A fault is declared successfully diagnosed if there is at least one test pattern for which the corresponding faulty scan cells are correctly identified in their entirety. Hence, the basic diagnostic coverage (BDC) is given by the following formula:

$$BDC = F^{-1} \sum_{i=1}^F d_i$$

where  $F$  is the total number of faults,  $d_i = 1$  provided there exists  $t_j$  such that  $C(f_i, t_j) = D(f_i, t_j)$ , and  $d_i$  is equal to 0 otherwise.

*Compound diagnostic coverage.* Alternatively, given test patterns from  $T(f_i)$  which detect fault  $f_i$ , diagnostic capacity is determined as a fraction of error patterns, i.e., sets of failing scan cells that can be correctly identified. In this case, this compound diagnostic coverage (CDC) can be expressed as follows:

$$CDC = F^{-1} \sum_{i=1}^F \omega_i^{-1} \sum_{j=1}^{\omega_i} d_{ij}$$

where  $\omega_i$  is the number of different error patterns caused by  $f_i$  when test patterns from  $T(f_i)$  are applied,  $d_{ij} = 1$  if  $C(f_i, t_j) = D(f_i, t_j)$ , and  $d_{ij}$  is equal to 0 otherwise. Recall that the second summation is governed by index  $j$  indicating test patterns that yield different error patterns.

In all experiments reported in Table III for ISCAS'89 benchmark circuits, several single-output  $M$ -bit convolutional compactors with a minimum span between polynomial taps equal to  $M/4$  were employed in each case. This is to illustrate how crucial it is to select an appropriate architecture of convolutional compactor such that neither the diagnostic time efficiency nor the quality of diagnosis are compromised. For each compactor, the number of memory elements  $M$  used to create its sequential part is given.

For each circuit, the following information is provided: the number of gates and memory elements, and the number of faults that propagate to the scan chains. Subsequently, for each convolutional compactor the corresponding basic diagnostic coverage and the compound diagnostic coverage are given. As demonstrated in Table III, very high basic diagnostic coverage (BDC) was achieved for all examined circuits. Moreover, only slightly lesser compound diagnostic coverage (CDC) was observed, despite a variety of error patterns produced by those faults that were detected many times.

Further experiments were conducted on five industrial designs. Some representative results are presented in the bottom part of Table III assuming that 1,000 faults were randomly selected in each case for the purpose of this experiment, and all sources of unknown states were masked. For each design, similar information is provided

**Table III. Diagnostic coverage for ISCAS'89 circuits and industrial designs**

CUT				Size ( $M$ ) of single-output convolutional compactor									
				16		32		48		64		128	
Name	Gates	Scan	Faults	BDC	CDC	BDC	CDC	BDC	CDC	BDC	CDC	BDC	CDC
s5378	3629	9 × 20	3878	96.11	91.32	97.22	94.62	97.37	95.32	97.96	96.96	97.96	96.97
s9234	6577	10 × 22	6700	95.49	88.12	96.25	90.26	96.42	91.61	96.39	91.85	96.42	92.00
s13207	10920	20 × 32	10806	94.92	91.27	95.83	92.60	95.82	92.86	95.98	93.41	95.98	93.58
s15850	10840	20 × 27	11934	91.80	84.70	92.89	86.50	93.03	87.33	93.17	87.75	93.18	87.89
s35932	16065	32 × 54	35545	99.78	99.73	99.86	99.76	99.96	99.86	99.96	99.88	99.96	99.91
s38417	29394	32 × 52	34406	98.64	93.44	99.40	96.80	99.52	97.41	99.60	98.07	99.68	98.82
s38584	25865	32 × 45	36073	95.26	91.99	95.47	92.46	96.42	93.88	96.45	94.15	96.46	94.39
D1	506K	50 × 373	1000	100.0	84.68	100.0	96.58	100.0	97.38	100.0	97.59	100.0	96.99
D2	271K	160 × 259	1000	-	-	98.28	93.43	98.78	95.83	98.99	96.44	98.68	95.88
D3	1095K	160 × 470	1000	-	-	100.0	99.86	99.90	99.85	99.90	99.90	99.90	99.90
D4	302K	160 × 116	1000	-	-	98.90	92.98	99.70	96.93	99.50	96.95	99.50	97.80
D5	496K	160 × 283	1000	-	-	94.76	88.00	94.15	86.90	94.46	86.55	93.53	85.86

as before. As can be seen, very high basic and compound diagnostic coverage was achieved in all cases.

The next group of experiments verified the proposed technique by using the same fail log information that was deployed in Tables I and II (100 scan chains, each 1050-bit long) and simulating convolutional compactors. Table IV gives a more detailed breakdown of error patterns collected for 370 faulty chips. As can be seen, the table provides the percentage of cases (error patterns) in which a given number of failing scan cells have occurred in a certain number of scan chains ( $S$ ). For instance, an entry in row 3 and column 5 indicates that error patterns comprised of 5 scan cells and located in 3 different scan chains one could observed in 17.55% cases.

**Table IV. The error patterns breakdown**

$S$	The number of failing scan cells							
	1	2	3	4	5	6	7	≥ 8
1	100	61.18	41.71	40.75	41.73	42.38	40.78	34.47
2	-	38.82	46.57	41.31	34.64	34.18	37.17	33.24
3	-	-	11.72	14.83	17.55	15.51	13.73	18.78
4	-	-	-	3.11	5.31	6.18	5.94	6.79
5	-	-	-	-	0.77	1.29	1.43	2.78
6	-	-	-	-	-	0.46	0.52	0.90
7	-	-	-	-	-	-	0.43	0.51
≥8	-	-	-	-	-	-	-	2.62

The experimental results with respect to the diagnostic coverage (DC) for all the examined chips are shown in Table V. In all cases, a single-output 128-bit convolutional compactor was simulated. Each row of the table corresponds to a given multiplicity  $mE$  of error patterns (as indicated in the first column), i.e., to the number of failing scan cells that constitute the recorded error patterns. The second column  $\#E$  provides the number of observed error patterns of a given size. The next four columns report the corresponding diagnostic coverage in

various forms. Here, the first two columns give the absolute number of cases diagnosed correctly altogether with the corresponding percentage, while the second pair of columns reports a cumulative diagnostic coverage, i.e., the number of successfully identified error patterns of multiplicities up to a value represented by a respective row of the table. As can be seen, for the error patterns of size up to 3, the complete diagnostic coverage was achieved. For larger error patterns, the diagnostic coverage slightly decreases, but it still remains above 95%. Only in a few cases of large error patterns, the diagnostic coverage drops below 90%.

**Table V. Diagnostic coverage for 370 faulty chips**

$mE$	$\#E$	Diagnostic coverage		Cumulative DC	
1	4760	4760	100.0	4760	100.0
2	5112	5111	100.0	9871	100.0
3	4426	4426	100.0	14297	100.0
4	4009	3904	97.4	18201	99.4
5	3436	3326	96.8	21527	99.0
6	3192	3085	96.6	24612	98.7
7	2612	2512	96.2	27124	98.5
8	2160	2061	95.4	29185	98.2
9	1877	1822	97.1	31007	98.2
10	1543	1470	95.3	32477	98.0
11	1250	1171	93.7	33648	97.9
12	1024	939	91.7	34587	97.7
13	951	873	91.8	35460	97.5
14	879	796	90.6	36256	97.4
15	776	707	91.1	36963	97.3
16	582	513	88.1	37476	97.1
17	410	362	88.3	37838	97.0
18	303	258	85.1	38096	96.9
19	216	165	76.4	38261	96.8
20	156	112	71.8	38373	96.7

It is worth noting that the total number of error patterns in all examined chips amounts to 41117 (the identical pat-

terns for the same chip are counted only once). Among them there are 1503 (i.e., 3.65%) errors of multiplicity greater than 20. 35% errors from this group were also correctly identified.

## 9. Conclusions

In this paper, we introduced a new fault diagnosis method for scan-based designs. It is enabled by the convolutional test response compaction scheme and has several properties matching extremely well the requirements of embedded deterministic test [19]. The scheme supports very high quality of test by providing ability to identify failing scan cells directly from the compacted test responses of the faulty circuits. Contrary to various techniques introduced earlier, the proposed solution does not repeat the same test experiment many times, and it collects all diagnostic data during a single application of test patterns. This feature greatly simplifies the tester requirements and the manufacturing test flow. Experimental results conducted on several large ISCAS'89 benchmark circuits as well as on industrial designs demonstrate feasibility of the proposed approach. In particular, we have shown that even with high compression ratios, exceeding 100x, it is possible to figure out exactly where the errors come from. Excluding catastrophic defects that produce massive numbers of scan cell errors, in more that 95% of the error patterns it is possible to determine precisely and in a time-efficient manner the scan cells where the erroneous signals originated from. The scheme is also consistent with the multi-site testing methodology, as it can dramatically reduce the number of outputs.

## Acknowledgement

The authors would like to thank Andreas Leininger, Peter Muhmenthaler, Frank Poehl, and Stefan Trost of Infineon Technologies AG, Munich, Germany, for providing the fail log data from production scan test that was used to simulate convolutional compactors and validate the technique described in this paper.

## References

1. R.C. Aitken and V.K. Agarwal, "A diagnosis method using pseudo-random vectors without intermediate signatures," *Proc. ICCAD*, pp. 574-577, 1989.
2. I. Bayraktaroglu and A. Orailoglu, "Diagnosis of scan based BIST: reaching deep into signatures," *Proc. Design Automation and Test in Europe*, pp. 102-109, 2001.
3. J.C. Chan and J.A. Abraham, "A study of faulty signatures using a matrix formulation," *Proc. ITC*, pp. 553-561, 1990.
4. J.C. Chan and B.F. Womack, "A study of faulty signature for diagnostics," *Proc. ISCAS*, pp. 2701-2704, 1990.
5. T. Damarla, C.E. Stroud, and A. Sathaye, "Multiple error detection and identification via signature analysis," *Journal of Electronic Testing: Theory and Applications*, vol. 7, No. 3, pp. 193-207, 1995.
6. S. Edirisooriya and G. Edirisooriya, "Diagnosis of scan path failures," *Proc. VLSI Test Symp.*, pp. 250-255, 1995.
7. J. Ghosh-Dastidar, D. Das, and N.A. Touba, "Fault diagnosis in scan-based BIST using both time and space information," *Proc. ITC*, pp. 95-102, 1999.
8. J. Ghosh-Dastidar and N.A. Touba, "A rapid and scalable diagnosis scheme for BIST environments with a large number of scan chains," *Proc. VLSI Test Symp.*, pp. 79-85, 2000.
9. R. Guo and S. Venkataraman, "A technique for fault diagnosis of defects in scan chains," *Proc. ITC*, pp. 268-277, 2001.
10. M.G. Karpovsky and P. Nagvajara, "Board-level diagnosis by signature analysis," *Proc. ITC*, pp. 47-53, 1988.
11. M.G. Karpovsky and S.M. Chaudhry, "Design of self-diagnostic boards by multiple signature analysis," *IEEE Trans. Comput.*, vol. 42, No. 9, pp. 1035-1044, 1993.
12. S. Kundu, "On diagnosis of faults in a scan-chain," *Proc. VLSI Test Symp.*, pp. 303-308, 1993.
13. W.H. McAnney and J. Savir, "There is information in faulty signatures," *Proc. ITC*, pp. 630-636, 1987.
14. S. Mitra and K. S. Kim, "X-Compact: an efficient response compaction technique," *IEEE Trans. CAD of IC*, vol. 23, No. 4, pp. 421-432, 2004.
15. S. Narayanan and A. Das "An efficient scheme to diagnose scan chains," *Proc. ITC*, pp. 704-713, 1997.
16. J. Rajski, J. Tyszer, C. Wang, and S. Reddy, "Convolutional compaction of test responses," *Proc. ITC*, pp. 745-754, 2003.
17. J. Rajski and J. Tyszer, "Diagnosis of scan cells in BIST environment," *IEEE Trans. Comput.*, vol. 48, No. 7, pp. 724-731, 1999.
18. J. Rajski and J. Tyszer, "On the diagnostic properties of linear feedback shift registers," *IEEE Trans. CAD of IC*, vol. 10, No. 10, pp. 1316-1322, 1991.
19. J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide, and J. Qian, "Embedded deterministic test for low cost manufacturing test," *Proc. ITC*, pp. 301-310, 2002.
20. J. Savir and W.H. McAnney, "Identification of failing tests with cycling registers," *Proc. ITC*, pp. 322-328, 1988.
21. C.E. Stroud and T. Damarla, "Improving the efficiency of error identification via signature analysis," *Proc. VLSI Test Symp.*, pp. 244-249, 1995.
22. R.C. Tekumalla, "On reducing aliasing effects and improving diagnosis of logic BIST failures," *Proc. ITC*, pp. 737-744, 2003.
23. J.A. Waicukauski, V.P. Gupta, and S.T. Patel, "Diagnosis of BIST failures by PPSFP simulation," *Proc. ITC*, pp. 480-484, 1987.
24. J.A. Waicukauski and E. Lindbloom, "Failure diagnosis of structured VLSI," *IEEE Design and Test of Computers*, pp. 49-60, 1989.
25. Y. Wu and S. Adham, "Scan-based BIST fault diagnosis," *IEEE Trans. CAD of IC*, vol. 18, No.2, pp. 203-211, 1999.