# Synthesis of Locally Exhaustive Test Pattern Generators

Günter Kemnitz

Institut für Technische Informatik
Technische Universität Dresden
01062 Dresden, Germany

## Abstract

*Optimized locally exhaustive test pattern generators based on linear sums promise a low overhead, but have an irregular structure. The paper presents a new algorithm able to compute the linear sums for real circuits up to several hundreds of inputs and outputs. The idea is to substitute a strategy of introducing fresh variables into an array of sums for the former linear independence test. This reduces the complexity of the calculation on an enormous scale. Experiments with several hundred randomly selected cone structures allow the rough estimation that the so computed generators are on average smaller than shift register based ones if the number of equal size cones is not larger than the number of inputs of the circuit under test.*

## 1 Introduction

The advantages in VLSI technologies, higher speed, density and complexity, make the implementation of self-test functions more and more attractive. Self-test functions are inserted by dividing a design into subcircuits under test and adding circuitry for test pattern generation and test response checking to the single subcircuits. Test patterns that can be produced on chip by an inexpensive hardware are mainly pseudo-random and exhaustive patterns.

An exhaustive test means to verify the function. Combinational functions need $2^{inp}$ different input vectors (inp - number of inputs). The exponential growth of the test length with the number of input terminals restricts the concept to circuits with a limited number of inputs:

$$inp < 20\ldots30 \qquad (1)$$

Sequential circuits cause more problems. Not only all variations of input, but of input and memory stages have to be checked. This leads to very tight bounds for the size of the circuit to be tested exhaustively. In practice, the exhaustive test of a sequential function is confined to the test of a combinational function (e.g. by providing access to the internal flip-flops via scan chains) [12]. For this reason only combinational circuits will be considered in this paper.

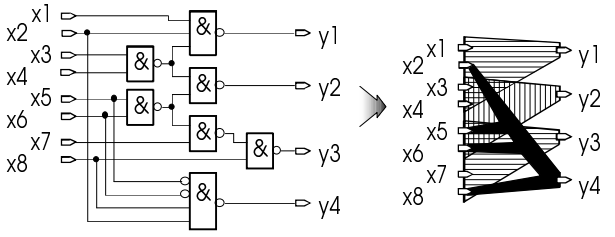


**Figure 1: Cone structure of an example**

Locally exhaustive [8] or pseudo exhaustive [5], [7] testing is a concept to avoid the restricted number of input terminals of the circuit to be tested. Often many or all output variables of a combinational function depend only on a small subset of input variables. The 8-input-circuit in Figure 1 consists e.g. of 4 subfunctions with only 4 inputs. The subset of inputs effecting an output $i$ is also called cone $i$ and its number of inputs cone width ($w_i$). Locally exhaustive test, to stimulate not the whole function but only the cones with all input variations, allows to reduce the number of test steps $t$ down to

$$2^{w_{max}} \leq t \leq \sum_{i=1}^{s} 2^{w_i} \qquad (2)$$

($w_{max}$ - maximum number of inputs of a cone) i.e. down to the number of tests between the amount to test the largest cone (the other cones are tested simultaneously) and the number of tests for all cones. A restricted cone size:

$$w_{max} < 20\ldots30 \qquad (3)$$

is substituted for a restricted number of inputs under test.

Locally exhaustive testing is much less restrictive than totally exhaustive testing. Nevertheless, the maximum cone size of real circuits exceeds often the tolerable bound. In [7] the maximum cone width of the generally used test-benchmarks are listed which goes up to $w_{\max} > 100$. Additional control gates [9], segmentation cells or scan cells [6] has to be built into the circuit to reduce the cone size under test.

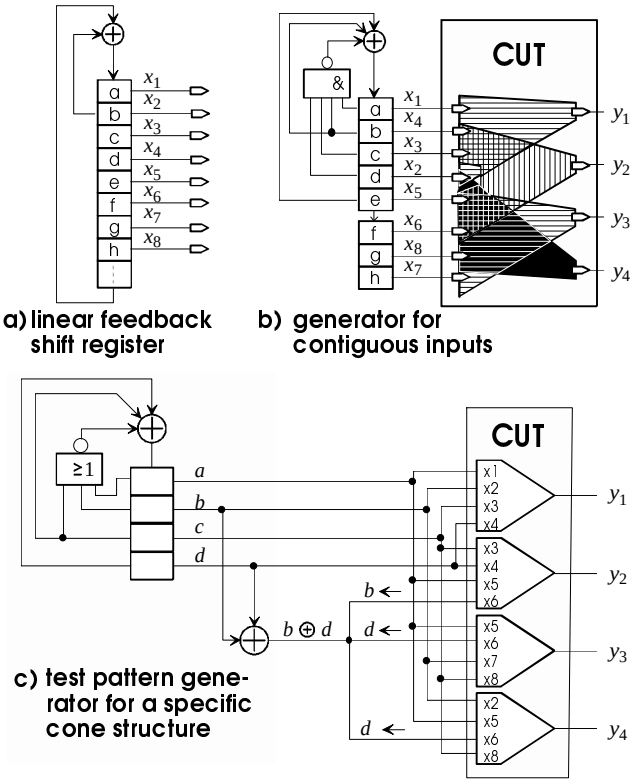


**Figure 2: Test pattern generators to test the circuit in Figure 1 in the locally exhaustive way**

Locally exhaustive test pattern generators can be designed for universal input spans [3], contiguous inputs [2], [11] or for application-specific input spans [3], [7]. Universal test pattern generators produce all $2^{w_{\max}}$ different test vectors for each and every possible $w_{\max}$-bit input span. Although, such a generator can be constructed [3], it becomes much to large for real applications. A proper substitute is a pseudo-random generator [8], e.g. a primitive feedback shift register (Figure 2a). The random sequence must be *l*-times longer than the exhaustive sequence for the cone ($l \approx 4 \ldots 10$). In this case, it contains on average the $1 - e^{-l}$ part of the input variations.

Exhaustive patterns for each span of $w_{\max}$ contiguous inputs can be produced on chip by a feedback shift register (Figure 2b). The feedbacked part (the first $w_{\max}$ flip-flops) forms an exhaustive generator. The rest of the shift register distributes the patterns. A sequence with the same properties can be produced by an (n,k) cyclic code over GF(2) [5] or a counter with an EXOR-Array [11]. Since, in a real circuit the input spans of the single cones are not contiguous inputs, a special assignment of the outputs of the test pattern generator to the inputs of the circuit under test (CUT) is needed.

Figure 2c shows a locally exhaustive test pattern generator designed for the cone structure in Figure 1. It consists of a 4-bit exhaustive pattern generator, which stimulates the majority of inputs under test, and linear sums to stimulate the inputs left (only one in the example). In comparison to the Figure 2a and b the generator in Figure 2c needs only half of the number of flip-flops, which means significantly less hardware. However, the structure is not regular and the calculation of it is very time consuming.

Next section will discuss the bottleneck in the calculation and derives a new mathematical approach to master the complexity. Section 3 presents the complete algorithm. This algorithm has been used to calculate the structure and size of test pattern generators for several thousand different cone structures produced by a random number generator. In section 4 the experiments and results are described.

## 2 Mathematical background

The computation of linear sums for a locally exhaustive test pattern generator can be related to a problem of the linear algebra. The whole generator consists of a *k*-bit long exhaustive generator and networks to produce modulo-2 sums. The exhaustive generator may be a feedback shift register as in Figure 2c or a binary counter as in Figure 3. Its outputs are considered as independent

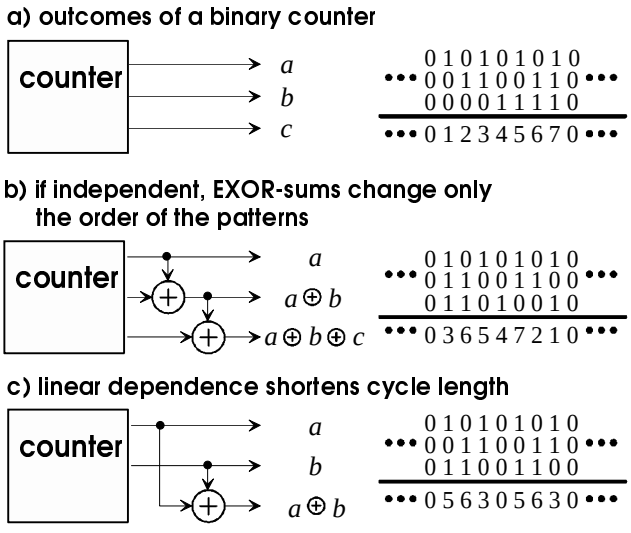


**Figure 3: Linear sums of test sequences**

variables. Independent means in this context that all $2^k$ different states are produced during the test. This is only a mathematical wording for *exhaustive test pattern generation*. The modulo-2 sums consist of those independent variables (alias outputs of the exhaustive generator). A generator output itself is considered to be a single-summand sum.

By substituting multiple-summand sums (Figure 3b,c) for single-summand sums (Figure 3a), the following two effects are possible. If a so constructed set of sums is linearly independent, the generator produces another exhaustive test sequence (Figure 3b). Each linear dependence (one sum is the sum of others) halves the cycle length of the produced patterns (Figure 3c).

The computation of the structure of a locally exhaustive test pattern generator for an application-specific cone structure can be described by a rule for each cone. The set of sums assigned to its inputs must be linearly independent. These rules imply that the number of independent variables (outputs of the exhaustive generator) must be at least as large as the size of the maximum-size cone. Other requirements of the calculation are low overhead and low test time. The latter is equivalent to a low number of independent variables.

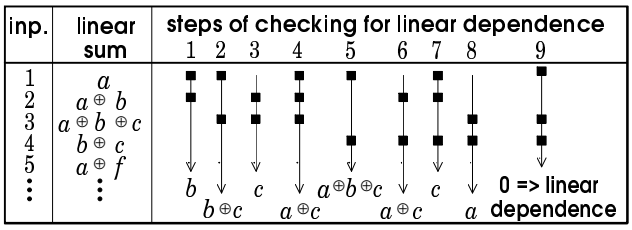


**Figure 4: Checking for linear dependence**

The test for linear dependence on its own is a complex task. For each tupel of two inputs, three inputs and so on, it has to be checked that the result of adding up the corresponding sums is unequal to zero. In Figure 4, the dependence will not be visible before the 9th step. To guarantee independence $2^w - w - 1$ checks are required, where *w* is the cone size. It is the same order of magnitude as verifying the exhaustive test by simulation.

The check for linear dependence, already time consuming, does not include the calculation of the sums. A purposeful strategy with low risk of rejects is needed. In [10], a generic structure is assumed, describing the linear sums for each number of independent variables and each number of inputs of the circuit under test. If for a number of independent variables no locally exhaustive test can be performed the number of variables will be incremented. In other words, after each unsuccessful trial the test time doubles. At the latest, when the generator size reaches the number of inputs of the whole circuit, which is equivalent to a totally exhaustive test, each linear dependence vanishes.

In [3] and [7] the computation starts with trying to connect inputs with each other under test. These are pairs of inputs that lead to different cones. The circuit in Figure 1 has three of such pairs: (x1-x5), (x2-x7) and (x3-x8). As the result, linear sums for less inputs have to be calculated. No purposeful strategy has been presented to select the sums themselves. The cone sizes of the examples presented in [3] and [7] are so small that the complexity of the dependence check does not matter.

The basic idea to handle the complexity is the following: The sums are assigned input by input. The actual sum has to introduce a fresh variable into the sum array of each cone the input is connected to. In Figure 4, the sums *a*, $a \oplus b$ and $a \oplus b \oplus c$ are assigned to the first 3 inputs of a cone. Selecting the next sum, it must include at least one of the unused (fresh) variables $\{d, e, f, \ldots\}$ as a summand. The fourth sum in Figure 4 $b \oplus c$ would e.g. need an additional summand. Other cones having also this input, will also call for fresh variables. All requirements together form a restricted set of alternatives for each new sum.

Linear dependence cannot arise in the assigning process. The first sum, assigned to a cone, is linearly independent, since it contains at least one variable. The next sum contains a variable, which is not in the first sum. Summing up both sums, at least the fresh variable will be left. So they are always independent. Having $n-1$ independent sums and introducing the next sum containing a fresh variable, the effect is similar. Summing up the new sum with any tupel of old sums, at least the fresh variable will be left. The independence is proved by recursion: One sum is independent; and *n* sums are independent, because $n-1$ sums are independent.

The technique solves two problems: It allows a purposeful search for sums, rather than a plain trial and error technique, and it avoids the check for linear dependence. The next section presents a complete algorithm to calculate linear sums.

## 3 The calculation of linear sums

The input of the calculation is the cone structure of the circuit that should be tested exhaustively. The first steps are two simplifications, reducing the number of inputs and reducing the number of cones (Figure 5). As in Figure 2c, tupels of inputs which have no cone in common can – and from the point of view of hardware minimization should – be linked together during test. As the

result, the number of inputs decreases and the cone overlapping, i.e. the number of cones per input, increases. The problem has been formulated as cliques covering of a graph [7]. We did the same without optimization. The inputs are checked two by two. If they have no cone in common they are linked by substituting a new virtual input for both inputs, and the procedure is continued. For large circuits, the effect of this simplification is remarkable. During the test the number of inputs of a 100-input circuit with 20 cones of the size 16 (randomly input-cone assignment, see section 4) can be reduced to about 25…40 inputs.

To calculate the test pattern generator, cones with equal input spans can be combined to form one cone (Figure 5). If the first will be stimulated exhaustively, the other will also. Having a smaller and a larger cone, where the smaller one shares all inputs with the larger one, only the larger cone is needed for the calculation of the test pattern generator. Such a cone relation may also arise during the reduction of inputs. In our experiments with randomly overlapping cones, cone reduction was unimportant.

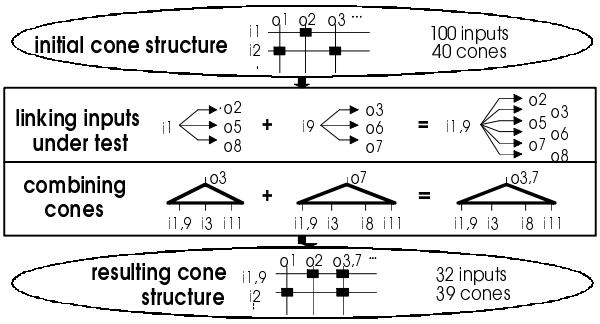


**Figure 5: Initial simplifications of the cone structure**

The outside loop of computing the set of linear sums initializes the number of generator variables with the maximum cone size and increments it in the case, the algorithm does not find a solution. However, in the first step it finds only a solution for circuits with no more than about 16 maximum-size cones. For circuits with substantially more maximum-size cones, up to 10 and more iterations may be necessary by our experience.

For a given number of generator variables, firstly, each generator variable is assigned to one input. Within those sums of single summands linear dependence can never arise. Next, sums are assigned one after another to each input still vacant. As discussed in the previous section, linear dependence is avoided by introducing at least one fresh generator variable into the array of sums of each cone which the input is linked to.

If multiple cones are linked to the actual input, it is sometimes impossible to introduce exactly one fresh variable for each cone. The consumption of fresh variables is greater than the reduction of the number of cone inputs to which sums has still to be assigned. From this follows that for most cones the number of generator variables must be larger than the number of inputs. If a shortage is foreseeable, the calculation is discontinued and started again with an increased number of generator variables.

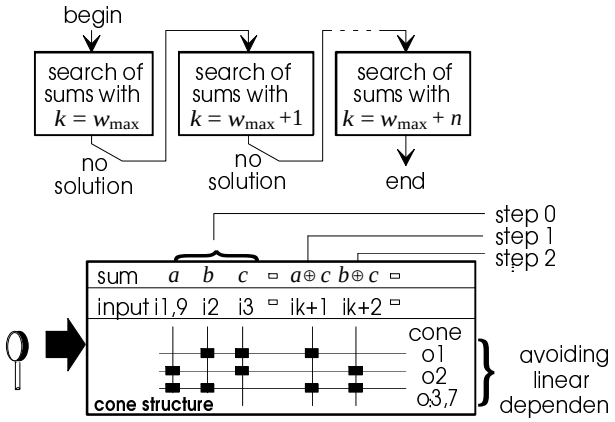


**Figure 6: The procedure of sum assignment**

Aims of optimization in the process of sum assignment are minimum test time and minimum hardware size. An essential part of the hardware are the sum networks. Roughly estimated, an additional summand in one of the sums causes an overhead of an additional EXOR-gate. So, the number of summands in each sum has to be minimized. The size of the exhaustive test pattern generator and the test time depend upon the number of independent generator variables. The corresponding strategy for sum selection is to avoid a shortage of generator variables for each cone. Both aims for optimization are not the same. In many cases a sum with a minimum number of summands does not use generator variables most sparingly and vice versa. By our experiments we came to the conclusion that it is better to select sums that introduce less fresh variables into the cones than those with less summands. This strategy increases the freedom for the remaining sums to be selected. And in the end, it reduces the generator size.

# 4 Experimental evaluation

As shown in [7] the maximum cone size of the benchmark circuits, generally used to evaluate test techniques, exceeds the bound of 20…30 inputs. Additional control and observation points have to be included. Of course, these test points affect the cone structure of the benchmarks to such an extent that one cannot compare test

pattern generators without using the same modifications under test.

The proposed test pattern generators have an irregular structure. To estimate with a sufficient level of confidence the average generator size and test time, a larger number of examples has to be investigated. For this reason we decided to do the evaluation with cone structures produced by a random number generator, 100 for each input and cone number.

Circuits with a large number of equal size cones are most difficult for the proposed algorithm. Additional smaller cones have no substantial effect to generator size and test time. Only the worst case of equal size cones is documented in Table 1. Having a circuit where the cones have different width, for an initial estimation it is enough to concentrate on the maximum size cones.

Figure 7 shows the general structure of the test pattern generator the values in Table 1 based upon. For the linear sums it is assumed that each sum is produced by a separate EXOR-tree. The number of EXOR-gates is the number of variables within the sum assigned to the corresponding input minus one. In Table 1 only the average number of EXOR-gates for the whole sum network $e$ is listed (average value for the test pattern generators for 100 different circuits). It has to be noted that the number of EXOR-gates can be reduced by sharing partial sums between different EXOR-trees, i.e. the real generators will be a bit smaller.

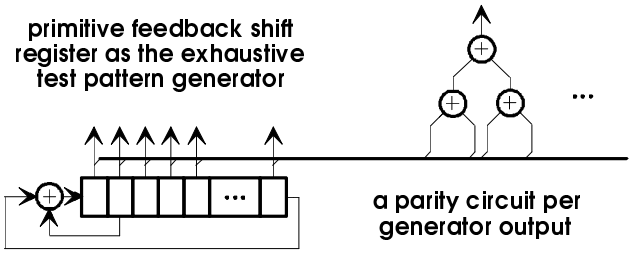


**Figure 7: Assumed generator structure**

The exhaustive test pattern generator should be a primitive feedback shift register of length $k$ ($k$-flip-flops for the register; 1 or 3 EXOR-gates for the feedback [1]). It produces all variations of binary states except the one with all zero. The all zero state can be included by inverting the feedback value for the generator stages $0\ldots0X$, which can be done by a NOR-gate with $k-1$ inputs and an additional EXOR-gate [4]. For most generators this extension is unnecessary. As seen in Table 1 for a larger

| inputs | equal size cones | $e$ | number of generator variables $k$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 16 *3 | 17 *1 | 18 *1 | 19 *3 | 20 *1 | 21 *1 | 22 *1 | 23 *1 | 24 *3 | 25 *1 | 26 *3 | 27 *3 | 28 *1 |
| 50 | 4 | 1.84 | 98 | 2 | - | - | - | - | - | - | - | - | - | - | - |
| | 8 | 8.94 | 70 | 30 | - | - | - | - | - | - | - | - | - | - | - |
| | 16 | 19.24 | - | 33 | 60 | 7 | - | - | - | - | - | - | - | - | - |
| | 32 | 38.44 | - | - | - | 44 | 53 | 3 | - | - | - | - | - | - | - |
| | 64 | 64.23 | - | - | - | - | - | 4 | 77 | 19 | - | - | - | - | - |
| | 128 | 76.58 | - | - | - | - | - | - | - | - | 26 | 65 | 9 | - | - |
| | 256 | 86.65 | - | - | - | - | - | - | - | - | - | - | - | 65 | 35 |
| 100 | 4 | 0.39 | 100 | - | - | - | - | - | - | - | - | - | - | - | - |
| | 8 | 3.46 | 88 | 12 | - | - | - | - | - | - | - | - | - | - | - |
| | 16 | 13.00 | 3 | 91 | 6 | - | - | - | - | - | - | - | - | - | - |
| | 32 | 27.74 | - | - | 15 | 77 | 8 | - | - | - | - | - | - | - | - |
| | 64 | 54.85 | - | - | - | - | - | 43 | 55 | 2 | - | - | - | - | - |
| | 128 | 103.57 | - | - | - | - | - | - | - | 1 | 63 | 36 | - | - | - |
| | 256 | 184.40 | - | - | - | - | - | - | - | - | - | - | - | 61 | 39 |
| 200 | 4 | 0.04 | 100 | - | - | - | - | - | - | - | - | - | - | - | - |
| | 8 | 1.07 | 99 | 1 | - | - | - | - | - | - | - | - | - | - | - |
| | 16 | 5.13 | 39 | 61 | - | - | - | - | - | - | - | - | - | - | - |
| | 32 | 15.73 | - | 11 | 85 | 4 | - | - | - | - | - | - | - | - | - |
| | 64 | 34.54 | - | - | - | 1 | 71 | 26 | 2 | - | - | - | - | - | - |
| | 128 | 71.75 | - | - | - | - | - | - | 1 | 48 | 50 | 1 | - | - | - |
| | 256 | 137.27 | - | - | - | - | - | - | - | - | - | - | 6 | 78 | 16 |

**Table 1: Expected test time and hardware size of locally exhaustive test pattern generators using linear sums ($k$ - number of generator variables and flip-flops; $e$ - average number of EXOR-gates for the linear sums; *1/*3 - minimum number of EXOR-gates for the feedback of the shift register)**

number of equal size cones the exhaustive generator must be at least one bit longer than the maximum cone size. So, during the generator cycle each cone will be stimulated at least two times with each pattern unequal zero. The all zero vector is also included, but one time less than the others.

To recapitulate, a test pattern generator consists in reference to Table 1 of :

- $k$ edge-triggered D-flip-flops
- on average $e+1\ldots3$ EXOR-gates
- if $k = w_{\max}$ 1 additional EXOR-gate, 1 NOR-gate.

The test time is $t_{\text{test}} = 2^k$.

The most important alternative test pattern generators, providing also a locally exhaustive test, are shift register based ones (pseudo random pattern generators or generators for contiguous input spans, see section 1). The hardware size of those generators is approximately one edge-triggered D-flip-flop per input of the circuit under test. A measure to compare the linear sum based test pattern generators with shift register based ones is the ration between the number of additional needed EXOR-gates to the number of saved flip-flops:

$$r = \frac{e}{\text{inp} - k} \qquad (4)$$

It can be assumed that an EXOR-gate does not need more area on chip than an edge-triggered flip-flop. So, up to $r \approx 1$ the sum based generators will be smaller. For circuits with 50 inputs the economic range is about up to 50, for circuits with 100 inputs up to 100 and for circuits with 200 inputs up to 200 equal size cones. To summarize, the statistics from the calculated test pattern generators (all together for 2,100 different cone structures) allow the rough estimation that the test pattern generators based on linear sums are on average smaller if the number of equal size cones does not exceed the number of inputs of the circuit under test.

## 5 Summary

An algorithm to compute linear sums for locally exhaustive test pattern generators has been presented. A strategy of introducing fresh variables into the set of sums for each cone is substituted for the trial and error technique of selecting sums and the linear dependence test. The new algorithm is much less complex and allows the calculation of test pattern generators for large circuits without time problems. The expectable size of the calculated test pattern generators and the expectable test time were examined by a large number of calculations. The experiments points out the tendency that for locally exhaustive testable circuits with no more equal-size cones than inputs the so computed generators are smaller than shift register based ones. So, it offers a way to reduce the hardware costs of BIST solutions while performing a complete locally exhaustive test.

## References


[1] Stahnke, W.: Primitive binary polynomials. Math. Comp. 27(1973)124, p. 977-80

[2] Tang, D. T.; Chen, C.-L.: Logic test pattern generation using linear codes. IEEE Trans. Comp., c-33(1984)9, p. 845-9

[3] Akers, S. B.: The use of linear sums in exhaustive testing. Comput. Math. Applic., 13(1987)5/6, p. 475-83

[4] Maeno, H. et al.: Testing of embedded RAM using exhaustive random sequences. ITC 1987, p. 105-10

[5] Wang, L.-T.; McCluskey, E. J.: Circuits for pseudo-exhaustive test pattern generation using shortened cyclic codes. ICCD 1987, p. 450-3

[6] Hellebrand, S.; Wunderlich, H.-J.: Automatisierung des Entwurfs vollständig testbarer Schaltungen. Informatik-Fachberichte, Bd. 188, GI-Jahrestagung, 1988, p. 145-9

[7] Chen, C.-I. H.; Sobelman, G. E.: An efficient approach to pseudo-exhaustive test generation for BIST design. ICCD 1989, p. 576-9

[8] Furuya, K.: A probabilistic approach to locally exhaustive testing. Trans. of IEICE, E72(1989)5, p. 656-60

[9] Lala, P. K.; Toa, D. L.: Partitioning of logic circuits for exhaustive testing. Int. J. Electronics, 69(1990)2, p. 225-31

[10] Wu, E.; Rutkowski, P. W.: PEST: A tool for implementing pseudo-exhaustive self- test. EDAC 1990, p. 639-43

[11] Rajski, J.; Tyszer, J.: Recursive pseudo-exhaustive test pattern generation. IEEE Trans. Comp., 42(1993)12, p. 1517-21

[12] Kemnitz, G.: Test und Selbsttest digitale Schaltungen. Habilitation an der TU, 1995