# Integrated Tools for Automatic Design for Testability

D. Schmid, H.-J. Wunderlich
and
F. Feldbusch, S. Hellebrand,
J. Holzinger, A. Kunzmann
Institute of Computer Design and Fault Tolerance*)
University of Karlsruhe

**Abstract:**

An increasing part of the overall costs of custom and semicustom integrated circuits has to be spent for test purposes, and therefore the integration of test and design seems to be a key of cost reduction. At the University of Karlsruhe a program system is currently developed supporting the design of testable circuits.

The program system under work essentially solves three tasks:

1.) Selection of an economical test strategy.
2.) Implementation of necessary circuit modifications in order to enhance testability, retaining the circuit function by construction.
3.) Generation of the test program.

A low cost test strategy is selected under constraints given by the user, who describes the intended application of the chip, its function and structure. These constraints include the number of produced chips, the desired dependability, the available test equipment, the chip size and architecture, technology, performance and many others. The surface of the program will be an expert system, asking the user for known parameters and proposing the test strategy.

There are two interfaces to the design tools. The functional entry will be the logical synthesis system CADDY where testability is regarded at a very early stage of the design. The structural entry is a netlist on gate level, and a library of the used cells.

The overall test strategy has to be in accordance with the chip architecture. For each part - random logic, PLA, RAM, ROM, busses - its appropriate test strategy has to be selected. While at these levels mainly expert knowledge is used, at the other levels the program system works algorithmically. At gate, switch, and layout level two kinds of algorithms are implemented: Test algorithms supporting the test program generation, and design algorithms modifying the circuit structure.

At gate level the design algorithms include the automatic integration of complete or incomplete scan paths, the modification of the system registers in order to support the self-test by conventional and optimized random patterns, the partitioning of circuits to facilitate pseudo-exhaustive testing, parity lines for PLAs etc. Furthermore these modifications must not destroy the circuit correctness, which is guaranteed by construction (logical function) or validated (timing, performance).

The test algorithms perform test pattern generation for combinational and sequential circuits, testability analysis and optimization, test pattern compression and also the compression of test responses. Additionally logic simulation and fault simulation are integrated into the system.

Especially the MOS technologies require more detailed fault analysis. Therefore at switch-level new algorithms have to be developed. They include the computation of fault detection probabilities at switch-level and test pattern generation.

At last at the layout level an automatic fault modeling is performed. For each logical cell the technology dependent faulty functions are generated. This precise modeling has two advantages: On the one hand there are many faults not covered by a technology independent fault model, and thus more realistic statements about fault coverage are possible. On the other hand in some cases the conventional fault models are dealing with faults, which can not occur, and which should be eliminated.

Moreover at this level a library of cells supporting the test strategies must be generated.

---

*) This research was supported by the BMFT (Bundesministerium für Forschung und Techno-logie) the Federal Republic of Germany under grant NT 2809 A 3

## 1. Introduction

During the past few years the design automation has made a significant progress, and nearly all tasks of the circuit design can be done automatically. Workstations with a standard cell environment or a silicon compiler are introduced into industrial practice, and application specific integrated circuits are now widely substituting solutions by boards and discrete elements.

On the one hand the circuit design is not any longer an art of a few experts, and can be done economically by many institutions and companies. On the other hand the resulting chips have a rapidly increasing complexity. This development has impact on the structure of the overall costs of integrated circuits, where the test costs are becoming the dominant part. There is not only a relative increase of the test costs due to shrinking design expenses, but also an absolute rise, since the test costs are growing with the design complexity. The test costs are now up to 60% [Benn84] through 70% [Will86] of the overall costs of application specific circuits.

The test itself is divided into several expensive tasks:

1) Test pattern generation

   The automatic test pattern generation (ATPG) for combinational circuits is NP-complete [IbSa75], and for general sequential circuits it is exponential [Roth78]. Practical experience has shown a quadratical through cubical increase of the computing time [Goel80] with respect to the size of a combinational circuit. The ATPG requires design modifications like a scan path, and it is still one of the most expensive computing tasks during the complete design process.

2) Test pattern application

   The test equipment must be at least as powerful as the circuit under test, since many technology dependent faults can only be detected by a high speed test [Tsai83], [WuRo86]. There are rapidly increasing requirements to the speed, the resolution, the channel number and the buffer size of a high performance tester resulting in rapidly increasing costs.

3) Design for testability

   As mentioned, in many cases the ATPG is only possible if additional circuitry is used supporting the test. The integration of a scan path needs up to 20% hardware overhead causing higher design costs, higher silicon area and less yield and reliability too. In order to produce self-testable circuits the necessary overhead is even higher, and the designer must evaluate the trade-off between the costs of the test equipment and the silicon costs of the self-test circuitry. Some investigations state that for application specific circuits in small and medium sized numbers an overhead up to 30% may be still economical [VARM84].

The objective of this paper is the description of a system of algorithms and tools that supports the designer to minimize the costs of these three tasks. Like programs generating minimized circuits, fast circuits or validated circuits automatically the intended system shall support the design of easily testable circuits. The main problem is that testability is not such a clear objective as speed, area or correctness. If a high performance test equipment is available, a scan design will be sufficient, whereas otherwise self-testable circuits would be required. Circuits testable by random patterns can sometimes hardly be dealt with ATPG and vice versa. It is known that different test pattern generation algorithms are optimal for different benchmark circuits [Brgl85].

To sum up the concept of testability has only meaning, if the involved test strategy is known. In the next section we discuss the parameters which have impact on their selection. We will describe the necessary information for this selection, and we will list the algorithms and tools providing this information. Moreover rule and knowledge based approaches are discussed, in order to do this selection automatically too.

Such an expert system will be suitable at the application and architectural level of the design, but at lower levels we are going an algorithmic way. Section 3 deals with the tasks at gate level, which include both test and design algorithms. The test algorithms are test

pattern generation, testability analysis or fault simulation for instance. The design algorithms include the circuit partitioning or the automatic integration of scan paths or BILBOs (Built-In Logic Block Observer). Both kinds of algorithms have to be carried out at switch-level (section 4), and at layout level (section 5) too. Finally section 6 gives an overview of the complete system.

## 2. The selection of test strategies

### 2.1 Decisions at the application and architectural level

A test strategy consists of a method to determine test patterns, a method to generate and evaluate these patterns during test application, a design style, and a fault model which should be covered. A large number of test strategies has been proposed during the past years, and it turned out that there are no optimal solutions, because a complex system of technical and economical parameters determines the suitability of a test method.

Only for illustrations we mention the number of processed chips, the chip size, the available test equipment, the requirements of fault coverage and reliability or the circuit technology. These are user requirements at the application level, but the choice of a test method for the complete circuit is also affected by the circuit architecture.

For instance the decision for a self-testable circuit leaves many degrees of freedom to implement the self-test for PLAs, for random logic or storages in different ways. There are many techniques proposed to make a PLA self-testable, most of them cheaper than an ordinary BIST for random logic. But often such a locally optimal solution for a PLA is not necessary, if a BIST structure for random logic can also be used directly or through a bus (see fig. 1).

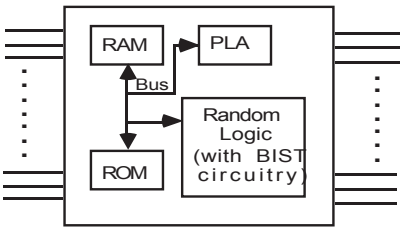



*Fig. 1:* Circuit architecture

Thus the test method for each single module must be chosen with respect to the global costs. It is a multidimensional optimizing problem to find the best solution for each block and for the overall circuit fulfilling the requirements about fault coverage, reliability, yield or area (fig. 2). A designer can only make a sure decision if the solution is implemented at least at gate level, and then he evaluates or estimates its costs and results.

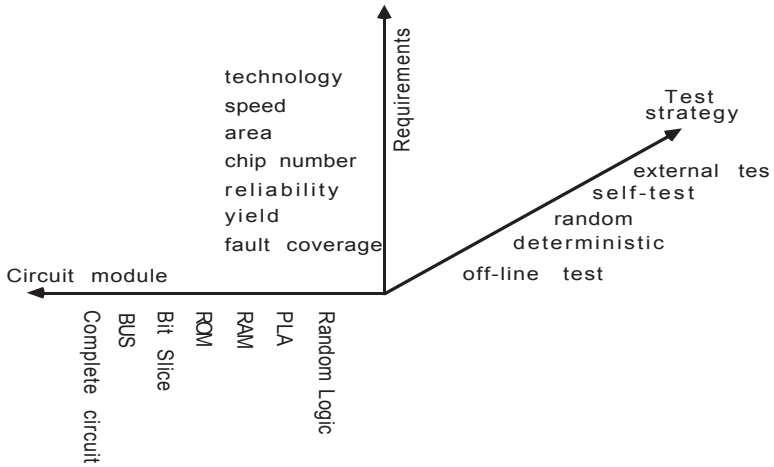



*Fig. 2:* Some dimensions of the decision problem on test strategies

The presented system offers a set of tools analyzing and implementing test strategies. For instance given a block of random logic the designer can estimate the necessary number of

random patterns at gate level, and he can compute the costs to integrate self-test modules for pseudo-random patterns. This is compared to the costs of the pseudo-exhaustive or external test.

The cost functions are derived at gate level and given to the user at the application level. Fig. 3 gives a rough overview about the implemented and analyzed test strategies.

Mainly we distinguish between the external test and the self-test method. Intermediate there is the off-line test by special chips. These special chips contain some of the test circuitry to reduce the hardware overhead.

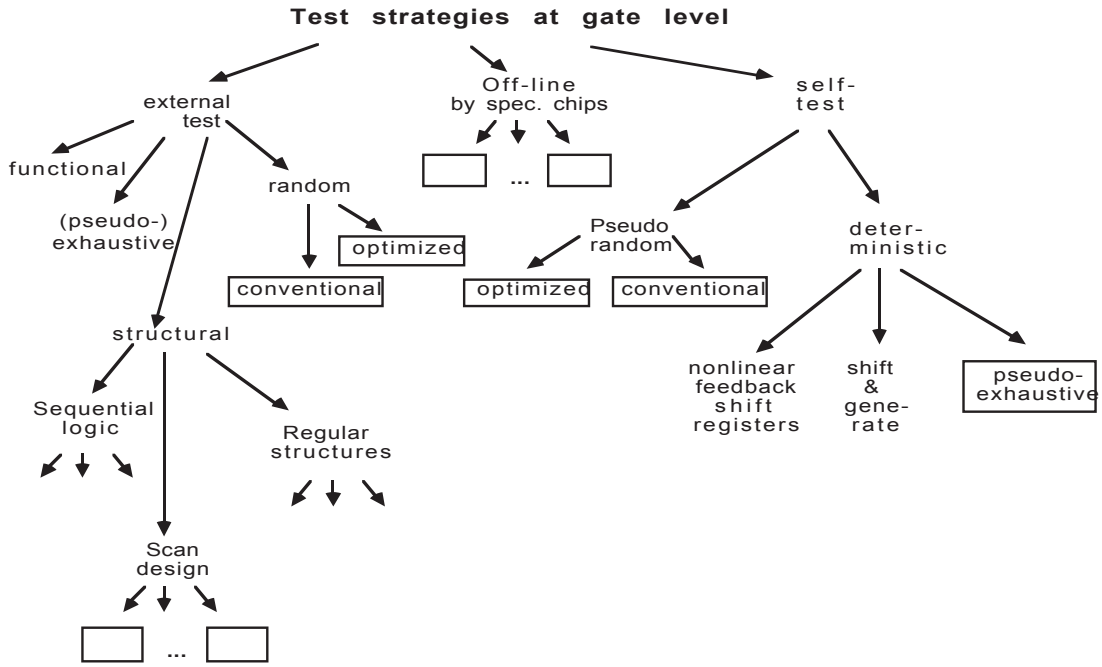



*Fig. 3:* Test strategies at gate level
At each leaf available: analysis algorithms, test generation algorithms, circuit modification algorithms. Marked with a box means specially developed for the presented system.

For the external test we have functional, pseudo-exhaustive and structural approaches. In the latter case the test patterns are generated based on the circuit structure, and the structure is modified in order to make the test pattern generation feasible. The presented work supports a scan design. As few storage elements as possible are integrated into an incomplete scan path automatically, still ensuring testability.

The other main approach is testing by random patterns which is possible as self-test strategy too. There also exist deterministic self-test approaches, where our research objective is the so called pseudo-exhaustive test. In the tree of fig. 3 the marked leaves denote algorithms especially designed for our program system which are presented in section 3 in deeper detail.

### 2.2 Rule and knowledge based approaches

Up to now we listed the tools and the informations needed by a designer in order to select a test strategy. Further research will be necessary in order to do this selection automatically too. The complexity of this problem prevents any algorithmic solution, such that the system under work will be divided into two levels (fig. 4).

Input are user specifications as performance, area, reliability, and a technology dependent library. The circuit itself is represented by a structural or a functional description, where in the latter case logic synthesis has to be performed ([RoCa85], [SCHM86]).

Based on this input the system will propose test strategies, in the current program version this task must be done by the user himself. Then the system modifies the circuit structure in order to implement a given test strategy, and it generates test patterns for the modified structure.

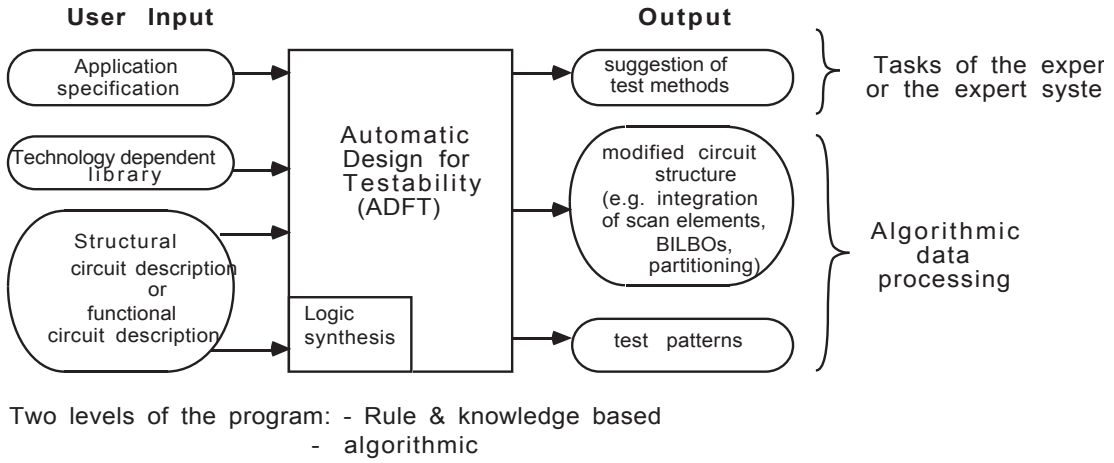



*Fig. 4:* Program system for automatic design for testability

Some preliminary work has already been done towards a complete system for automatic design for testability. In 1985 Fung et al. presented the TESTPERT (testability expert) which is integrated into the SILC silicon compiler of GTE [FUNG85]. This system provides most of its informations in rather a conventional way by so called testability measures, but it also uses some information of the application level as described above, and integrates additional circuitry to enhance testability. The main differences to our approach are their use of testability measures, and the extent of integrated test structures. Since the precision of conventional testability measures does not seem to justify expensive design changes [AgMe82], our system uses a variety of cost functions adapted to each test strategy.

At the University of Southern California an expert system was developed supporting the design of a testable PLA [BrZh85], and a more general Testable Design Expert System (TDES) has also been reported [AbBr85]. The description of this rule based system concentrates on the circuit representation with BILBO integrations as an example. Another rule and knowledge based system is described in [SaFo86], suggesting and explaining circuit modifications that improve testability. Also a more detailed review of previous work on this topic is given. In [BGR87] knowledge based tools are reported checking whether the circuits satisfy certain design for testability rules. A knowledge based selection of self-test techniques all based on linear feedback shift registers (LFSR) is described in [Kras87]. But up to now a system has not been reported supporting the automatic selection of test strategies in a sufficiently general way. This part of an automatic design for testability is still left to the designer.

## 3. Test strategies at gate level

### 3.1 The external random test

Testing by random patterns has several advantages. Here we can dispense with the time consuming automatic test pattern generation, and the application of random patterns needs no expensive test equipment, since it can be done by linear feedback shift registers (LFSR) during self-test. This is possible in high speed, and therefore many technology dependent dynamic faults are detected in addition ([Tsai83], [WuRo86]).

In general sequential logic is not treated by random patterns, since the probability may be very small that certain initializing sequences occur randomly. Hence one uses a chip with scan design, does logic simulation with the combinational part, and feeds the scan path and primary inputs by pseudo-random patterns. Finally the circuit response is checked.

Doing so essentially three problems have to be solved: The number of random patterns has to be determined to which the circuit must respond correctly in order to be assumed

fault free. This leads to the second problem to compute fault detection probabilities. Finally we have to deal with circuits which cannot randomly be tested since the fault coverage would be too low, or the necessary test length would be too large.

The random test length N depends on the confidence $\delta$, that is the probability to detect all faults $f \in F$ of the fault model F by applying N randomly generated patterns. If we assume that the detection of some faults by a pattern set of size N forms completely independent events, then we have

$$\delta = \prod_{f \in F} (1 - (1 - p_f)^N) \tag{1}$$

where $p_f$ denotes the detection probability of the fault f.

For large N the assumption of independence is asymptotically fulfilled, but in general computing the necessary test length N by formula (1) will only provide an upper bound. For our purposes this is precise enough, and N can be evaluated by computing fault detection probabilities for combinational circuits.

For circuits with tree structures such algorithms were presented by P. and V. D. Agrawal [AgAg75], and the general case was solved by Parker and McCluskey [McPa75a,b]. But the latter procedure shows an exponential time and storage complexity due to the NP-completeness of the underlying fault detection problem. Therefore in recent years much work has been done to *estimate* those probabilities.

The cutting algorithm [BDS84] determines upper and lower bounds for the probabilities, PROTEST (Probabilistic Testability Analysis) ([Wu84], [Wu85]), and a new version of PREDICT [ABS86] estimate by an analyzing procedure, and STAFAN [AgJa84] counts the signal values during simulation.

In our system the algorithms of PROTEST are integrated, which have already been presented in ([Wu85], [Wu87a]). But now it turns out that there are many circuits which cannot randomly be tested due to faults with low detection probabilities. Table 1 shows for some circuits the necessary test lengths based on estimations of PROTEST. The circuits C<n> are the well-known benchmarks of the ISCAS'85 test session [Brgl85], the circuit S1 is a 24-bit comparator constructed by six Texas Instruments comparators SN 7485 [TI80], where some redundancies are removed, and S2 is the combinational part of a 32 bit divider [KuWu85].

| | Circuit | Required test length |
|---|---|---|
| * | S1 | $5.6*10^8$ |
| * | S2 | $2.0*10^{11}$ |
| | C432 | $2.5*10^3$ |
| | C499 | $1.9*10^3$ |
| | C880 | $3.7*10^4$ |
| | C1355 | $2.2*10^6$ |
| | C1908 | $6.2*10^4$ |
| * | C2670 | $1.1*10^7$ |
| | C3540 | $2.3*10^6$ |
| | C5315 | $5.3*10^4$ |
| | C6288 | $1.9*10^3$ |
| * | C7552 | $4.9*10^{11}$ |

*Table 1:* Necessary test lengths for a conventional random test (by PROTEST).

In table 1 the marked (*) circuits need an exorbitant size of the random test set, and these predictions of PROTEST are confirmed by fault simulation:

| | Circuit | Test length | Fault coverage |
|---|---|---|---|
| * | S1 | 12,000 | 80.7 % |
| * | S2 | 12,000 | 77.2 % |
| * | C2670 | 4,000 | 88.0 % |
| * | C7552 | 4,096 | 93.9 % |

*Table 2:* Fault coverage by simulation of conventional random patterns.

It should be noted that an estimation with the exact value 0 or 1 of a signal probability by PROTEST is a proof (not an estimation!) of redundancy. But of course not in all cases a fixed signal value can be detected this way, and therefore redundancies may be left which cannot be found by PROTEST. The fault coverage in table 2 is computed with respect to faults which are not proven to be undetectable due to redundancy. The table indicates that self-testing of those circuits may need several hours, preventing an economical use of random patterns.

In many cases this problem is solved applying optimized random patterns. That is, each primary input is set to logical "1" by its specific optimal probability. Using such optimized input probabilities PROTEST proposes the test lengths listed in table 3:

| | Circuit | Required test length |
|---|---|---|
| * | S1 | $1.5*10^4$ |
| * | S2 | $4.0*10^4$ |
| * | C2670 | $6.9*10^4$ |
| * | C7552 | $1.2*10^5$ |

*Table 3:* Necessary test lengths for optimized random tests (by PROTEST)

The results of fault simulation listed in table 4 prove that such optimized random patterns yield a higher fault coverage indeed.

| | Circuit | Test length | Fault coverage |
|---|---|---|---|
| * | S1 | 12,000 | 99.7 % |
| * | S2 | 12,000 | 99.7 % |
| * | C2670 | 4,000 | 99.7 % |
| * | C7552 | 4,000 | 98.9 % |

*Table 4:* Fault coverage by simulation of optimized random patterns

A gradient method to compute optimized input probabilities has been proposed in [LBGG86]. In our system the algorithms described in ([Wu85], [Wu87a], [Wu87b]) are integrated. They use the fact that for each fault $f \in F$ the detection probability $p_f(X)$ depends on the tupel of input probabilities $X := \langle x_i \mid i \in I \rangle$ where I denotes the set of primary inputs. Therefore the formula above turns into

$$\delta_N(X) = \prod_{f \in F} (1 - (1 - p_f(X))^N). \tag{2}$$

This formula expresses the probability that all faults are detected by N patterns with the distributions of X. Using some well-known approximations this is transformed into

$$\ln(\delta_N(X)) \approx -\sum_{f \in F} (1 - p_f(X))^N \approx -\sum_{f \in F} e^{-N \cdot p_f(X)}. \tag{3}$$

This describes the objective function, and we call a tupel X of input probabilities optimal with respect to N, if

(4) $$J_N(X) := \sum_{f \in F} e^{-N \cdot p_f(X)}$$

is minimal at $X \in [0,1]^I$.

In the {0,1}-space expectation value and probability coincide, and the stochastical optimizing problem reduces to a deterministical one. But this is only a modest simplification, since one immediately notices that the objective function is not a member of the well-known linear or quadratical optimizing problems. In general the objective function will not be convex or even unimodal. Our optimizing problem is a member of the general class of smooth multi-extremal problems, which have an exponential average case complexity with respect to the number of variables, and to the required precision [NeYu83]. The known global optimizing procedures like the Newton or the gradient method will fail to handle large circuits with hundreds or thousands of input variables resulting from scan designs.

Therefore we do not try to find a global optimum, but we use some approximations to search a relative one. In [Wu87a] it is shown that the objective function is strictly convex with respect to a single variable. Hence for each fixed $x_1,..,x_{i-1}, x_{i+1},..,x_n$ there exists exactly one $x_i \in [0,1]$ with minimal $J_N(x_1,..,x_i,..,x_n)$.

**Notation**:

Let $Z := \langle z_1,\ldots,z_n \rangle$, and $y \in [0,1]$. We write $f(Z,y_{|i}) := f(z_1,\ldots,z_{i-1},y,z_{i+1},\ldots,z_n)$. One easily verifies that

$$p_f(X) = p_f(X, 0_{|i}) + x_i(p_f(X,1_{|i})-p_f(X,0_{|i})).$$

Hence

$$\frac{dJ_N(X,y_{|i})}{dy} = \sum_{f \in F} -N(p_f(X,1_{|i})-p_f(X,0_{|i}))e^{-Np_f(X,y_{|i})}$$

and

$$\frac{d^2J_N(X,y_{|i})}{d^2y} = \sum_{f \in F} N^2(p_f(X,1_{|i}) - p_f(X,0_{|i}))^2 e^{-Np_f(X,y_{|i})} > 0.$$

The ">" holds since we assume irredundancy, and at some primary input we have $p_f(X,1_{|i})-p_f(X,0_{|i}) = 0$. And now we have:

> For each $X \in [0,1]^I$ there exists exactly one $y \in [0,1]$ with $dJ_N(X,y_{|i})/dy = 0$ and $J_N(X,y_{|i})$ has its minimum there.

Since all derivations of the objective function are explicitly available this minimum point can be computed by simple iterations like the Newton algorithm or the regula falsi.

Fig. 5 sketches the complete subsystem PROTEST that has been integrated. PROTEST uses a gate level netlist of the circuit, and a functional library of the primitives. Then it estimates the test lengths, the fault detection and the signal probabilities. It computes optimal distributions for random test sets, and generates these patterns. Finally it validates its predictions by a static fault simulation. The latter feature is useful, if a self-test technique is implemented based on optimized input probabilities.

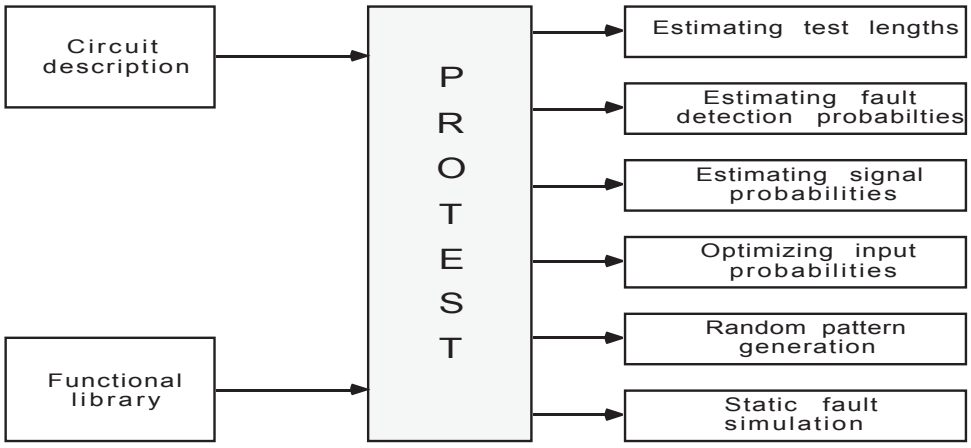



*Fig. 5:* Program system PROTEST

### 3.2 External test by deterministic patterns

Another tool has to support the external deterministic test. Objectives of this test strategy are shortening of the test length, treatment of circuits which are not randomly testable, certainty to detect a given set of faults, or the detection of redundancies within circuits.

State of the art are so called implicit enumeration algorithms like PODEM [Goel81] and FAN [FuSh83]. They are using testability measures to control the order they enumerate the input patterns.

The quality of the testability measure and further heuristics as implemented for instance in SOCRATES [SCHU87] mainly determine the number of backtrackings and thus the overall running time. The presented program system includes the tool SPROUT (Signal PRObability Using Test pattern generator), that uses the signal probabilities provided by PROTEST as testability measure and control function [Kunz87b]. Fig. 6 gives an overview of the interfaces of SPROUT and PROTEST.

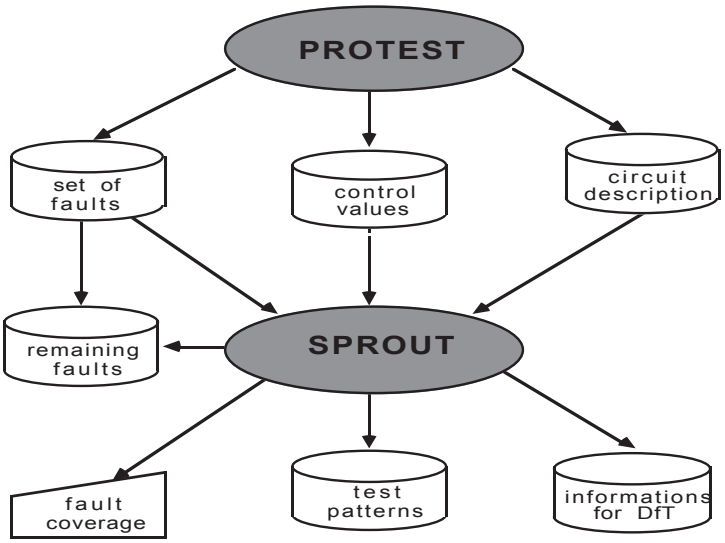



*Fig. 6:* Interfaces of SPROUT and PROTEST

Using a scan design it is in many cases sufficient to deal with combinational circuits, with the single fault assumption and with a five-valued logic as the well-known enumeration algorithms do. But these algorithms often fail to handle sequential circuits and therefore SPROUT uses a nine-valued logic. The capability to deal with sequential logic is needed, if an incomplete scan path is integrated as described in the next section. In detail the algorithms and the performance of SPROUT are reported in [Kunz87b].

### 3.2 Circuit modifications supporting the external test

There are no algorithms known overcoming the test generation problem for general sequential synchronous circuits. This is due to the fact that the worst case test length for a single fault may increase exponentially with the circuit size [Roth78].

For this reason many scan design techniques were proposed, where during the test mode all the storage elements are part of a serial shift register (see e.g. [EiWi77]) and therefore directly accessable.

The automatic integration of a complete scan path has been the first design for testability capability of the presented system [KuWu85]. Nowadays this is state of the art and is already supported by standard cell environments like VENUS [GeNe84].

The hardware overhead to integrate a complete scan path is about 10 % to 20 % [Benn84], and it is determined by the additional routing area, the larger storage elements, by multiplexers selecting the valid input data, and by the supplementary peripheral inputs and outputs. In order to reduce this hardware overhead, Trischler proposed the integration of an incomplete scan path [Tris80]. A detailed comparison between both types of scan paths can be found in [Kunz87a].

In [Tris80] and [Tris83] controllability and observability values at the storage elements decide whether they are integrated into the incomplete scan path. In [AGRA87] a method has been proposed, to select the scan elements during test pattern generation for the combinational part of the circuit.

But both approaches cannot guarantee that the following two conditions are kept:

1) For each fault its test sequence is bounded linearly in the worst case.

2) The number of selected storage elements is minimal.

A procedure has been integrated into the presented system that selects storage elements satisfying these conditions. This procedure is called INSPIRATION (INcomplete Scan Path IntegRATION). To illustrate our approach we map a sequential circuit on a connectivity graph where the nodes correspond to storage elements, primary inputs and outputs. Fig. 7 shows a part of a sequential circuit, consisting of 5 storage elements (D-type flip-flops) and the corresponding connectivity graph (CG).

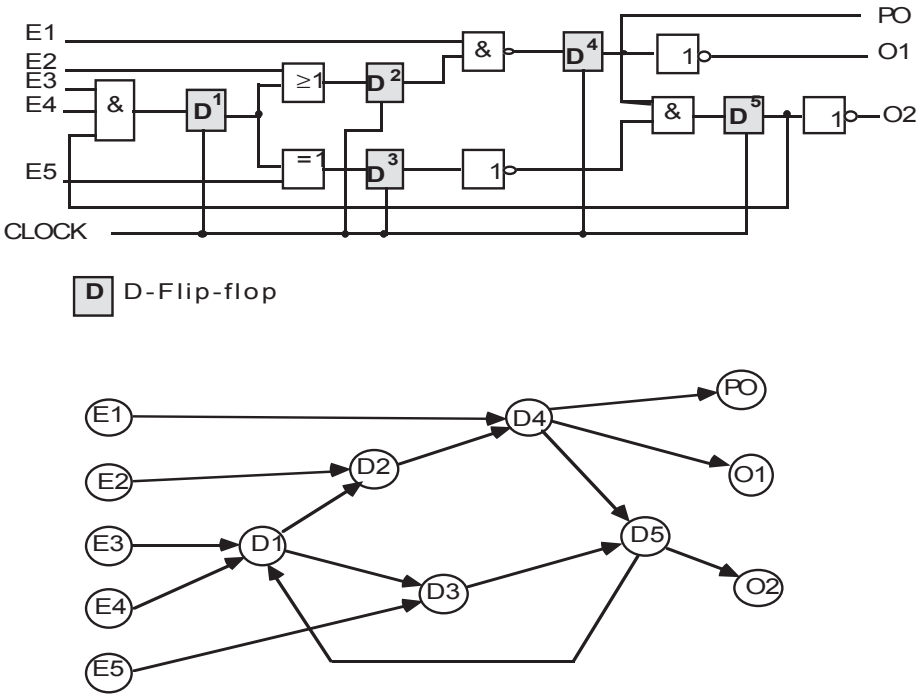



*Fig. 7:* Part of a sequential circuit and its connectivity graph CG

The connectivity graph must be acyclic in order to guarantee that the test length is linearly bounded. If the number of cycles is not restricted the connectivity graph may represent a circuit with a linear feedback shift register with $2^n$ - 1 states, and with an initializing sequence of the same length. But even if there is only a single cycle in the connectivity graph, the test length for a single fault may increase quadratically.

The circuit of fig. 8 consists of a single cycle with n storage elements, and a path from input P1 via m storage elements.

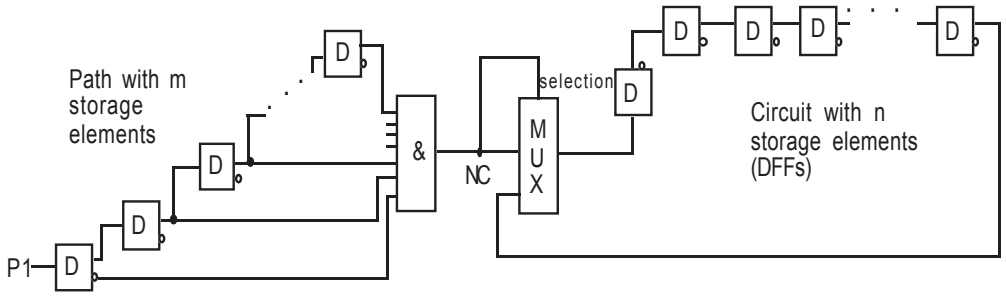



*Fig. 8:* Sequential circuit with one single cycle

If signal NC is "1", this value is transfered to the cycle. Obviously the cycle can be pushed into any state. But any "1" at net NC is followed by (m-1)-times "0", which cannot influence the cyclic shifting. Thus a test sequence of O(n·m) is necessary to force the circuit into a certain state.

The previous considerations are only valid, if the current state of a storage element has no impact on its next one. If other kinds of storage elements are used, they have to be modeled by D-flip-flops for instance. For an acyclic graph it can easily be proven that the test length for a single fault cannot exceed the rank of the graph.

Therefore INSPIRATION provides a procedure to generate acyclic connectivity graphs by removing a minimal number of nodes. This proceeding consists of two algorithms: (1) Search all the elementary cycles of a directed graph and (2) determine a minimal number of cuts, affecting all cycles using heuristics for the set covering problem [ChKo75]. Then the corresponding storage elements are integrated into an incomplete scan path.

Based on the new acylic connectivity graph the sequential circuit is mapped into iterated combinational circuits, where the image of a single fault may be a multiple fault.

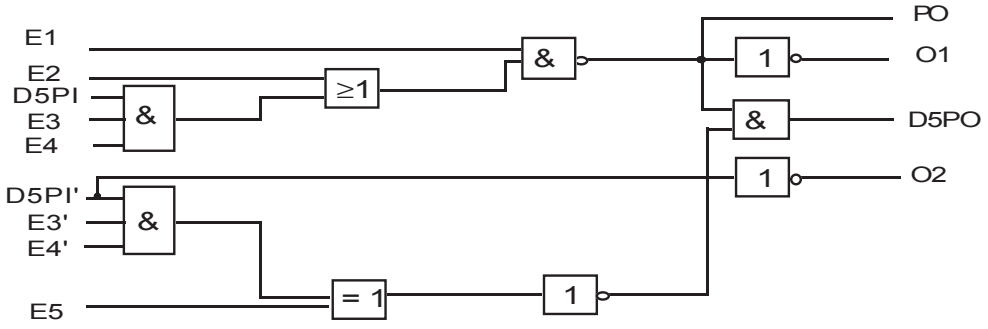



*Fig. 9:* Combinational circuit with scan-element D5

In the example circuit 2 cycles exist (D1, D2, D4, D5 and D1, D3, D5). To eliminate both cycles the set covering problem suggests the cut at element D5. The rank of the resulting CG is 4. Circuit transformation yields a combinational representation (fig. 9) with 66 single and 18 multiple faults. Each test pattern has to be transformed into a test pattern sequence of length 4.

This approach results in less hardware overhead for test purposes, but this is paid by the additional test pattern generation costs for sequential circuits. However if the sequential

circuit is transformed into a pipeline structure (fig. 10), for test pattern generation the circuit can be handled exactly like a combinational one.

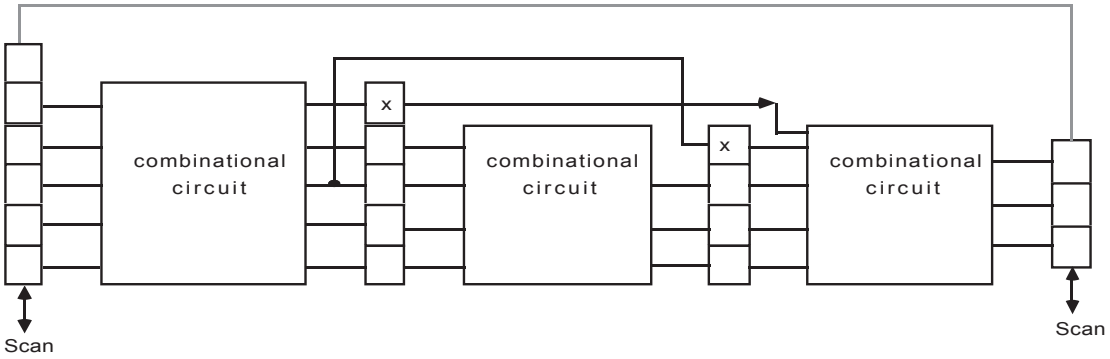



*Fig. 10:* Pipeline structure

There are also algorithms integrated to make the marked flip-flops directly accessible, such that the remaining structure is a pipeline. Then all test algorithms for combinational logic are applicable to this sequential circuit.

### 3.3 Self-test by random patterns

The most widely used self-test techniques are based on LFSRs, where the system registers are augmented by some additional hardware. Then the registers can be controlled to perform the normal operating mode, the shifting mode or the LFSR mode. Fig. 11 shows the typical test configuration.

Here the test is carried out within five phases. Firstly the registers R1 and R2 are reset. Then both registers work in the LFSR mode, where R1 produces random test patterns for the combinational network SN1, and R2 compresses its responses by signature analysis. Thirdly the signature of R2 is shifted out, and then both registers work as LFSR again, but R2 generates the patterns for SN2 and R1 performes signature analysis. At last the signature of R1 is shifted out.

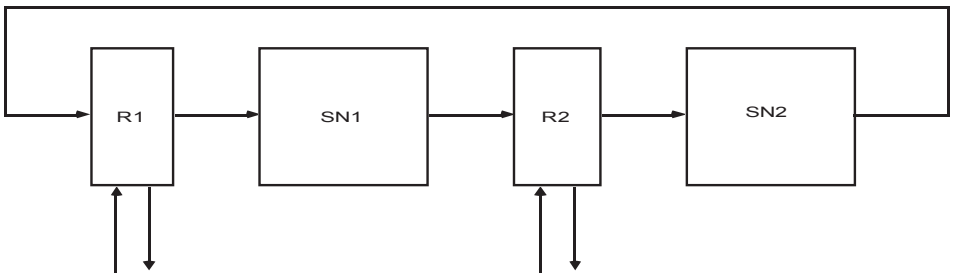



*Fig. 11:* A self-test configuration

The LFSRs produce random patterns by polynomial division over GF(2). This is possible by two different architectures. The LFSR of type I feeds back the linear sum

$$\sum_{i=0}^{r-1} g_{r-1-i}\, t_i$$

into $t_0$ (fig. 12a), and the LFSR of type II feeds $s_{r-1} \oplus s_{i-1}$ into those $s_i$, where $g_i = 1$ (fig. 12b).

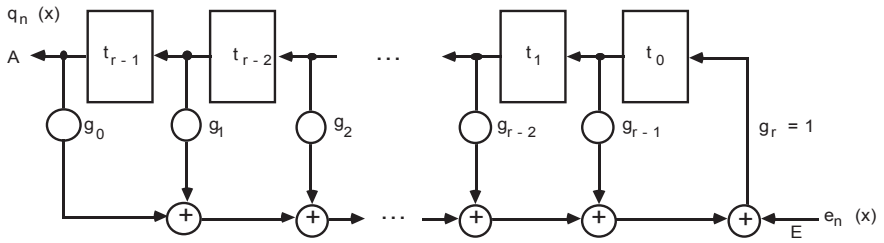



*Fig. 12a:* Standard LFSR (type I)

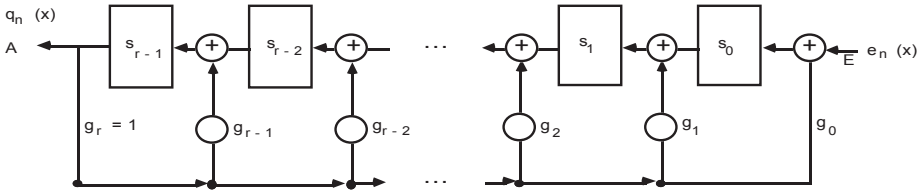



*Fig. 12b:* Modular LFSR (type II)

Both automata are equivalent, for a detailed discussion see [Golo67]. If they implement a division by a primitive polynomial, their period is maximal and a produced bit sequence of length m satisfies some basic random properties.

Conventional self-test techniques use standard LFSRs, since besides the feedback functions they are easily composed by cascading identical cells. Well-known is the BILBO approach [KOEN79]. Some work has already been done to integrate BILBOs into a logic netlist automatically ([AMBL86], [KrAl85]).

More difficult is the generation of circuits which are self-testable by optimized pseudo-random patterns. A module called GURT (Generator of Unequiprobable Random Tests) was presented in [Wu87c]. A GURT is based on modular LFSRs and has four operation modes:

1) Normal system operation as register
2) Unequiprobable random pattern generator
3) Signature analysis
4) Shift register

During pattern generation the GURT is configurated by a module LR, which is a modular LFSR, and by a module SR, that is mainly a shift register (fig. 13).

LR generates equiprobable random patterns, and it feeds a boolean function F. At the output of F there is a biased sequence of probability p (e.g. if F is a 3-input AND: p=0.125). This sequence is shifted into SR, where by inverting the value 1-p may be obtained.

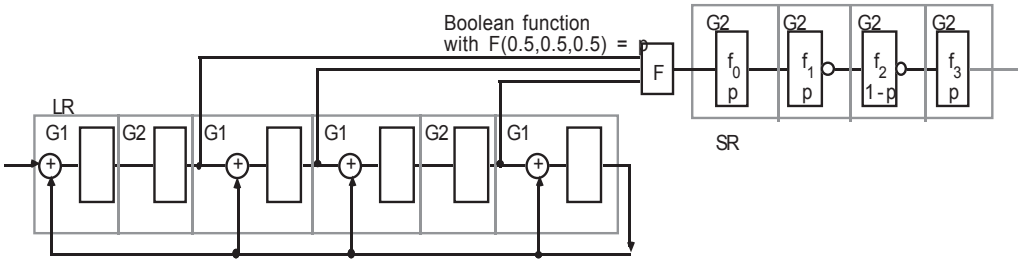



*Fig. 13:* Random pattern generation using LR and SR

If the GURT is in the signature analysis mode, both LR and SR are working together as a large linear feedback shift register (fig. 14).

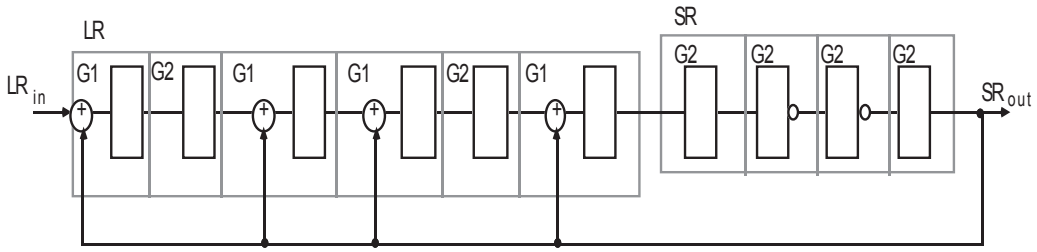



*Fig. 14:* A GURT in signature register mode

Using at least three GURTs with different functions F each of the probabilities 1/8, ..., 7/8 can be produced. A gurt needs a sufficiently long pattern sequence as input in order to avoid periodicity. This input sequence can be generated by a long LFSR with self-test features like a BILBO or an LR described above, or by a preceeding GURT.

Additionally one GURT is only able to generate the three probabilities p, 1-p, and 0.5 at its storage elements. Therefore the inputs of the combinational circuit under test have to be grouped into four sets: one set contains the inputs which must be stimulated with probabilities 1/8 and 7/8, in the next set there are the inputs with probabilities 1/4 and 3/4, a further set needs probabilities 3/8 and 5/8, and at least there is the set with probability 1/2. If a circuit has inputs in each of those sets at least 3 GURTs must be connected in series (see fig. 15).

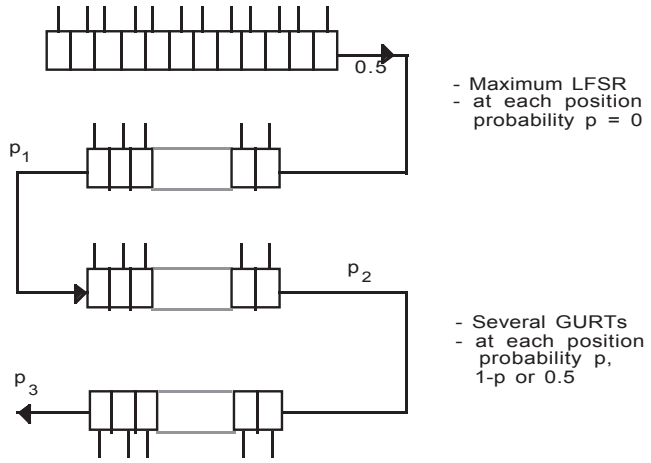



*Fig. 15:* Self-test configuration using GURTs

The GURTs and the circuit inputs are ordered by a heuristic optimizing procedure.

### 3.4 Deterministic Self -Test

In the following "deterministic test" will denote the test by deterministicly determined patterns in contrast to test by (pseudo-) random patterns. Straightforward approaches to implement the deterministic test as built-in self-test are the complete enumeration of all possible patterns by a counter or LFSR ("exhaustive test") or the storage of the elements of a deterministic test set in a ROM on chip ("prestored test"). In recent years some more sophisticated methods have been developed. One approach based on nonlinear feedback shift registers was suggested by Daehn [Daeh83]. For a given set T of deterministicly determined test patterns a shift register is designed whose sequence of stages contains T as a subset.

Another technique called "Store and Generate" [AgCe81], [AbCe82] works as a compromise between the prestored and the exhaustive test. A given testset T for a circuit with n primary inputs is transformed into a set T' that can be divided into classes such that the test patterns belonging to the same class only differ in the last n-k components for a fixed $k \leq n$. It is sufficient to store one representative of each class (i.e. its first k components) in a ROM on chip. All elements of T' are obtained if each stored pattern is combined with all

different output vectors of a (n-k)-stage LFSR or counter. Retransformation of T' into T provides all patterns in T.

Of course the fault coverage of both approaches depends on the quality of the test set, and on the underlying fault model. Furthermore these approaches need a high computing effort and especially the second one a high hardware overhead too: a ROM, an LFSR and a decoder are required. Therefore our system supports a third approach: pseudo-exhaustive or verification test. Consider a combinational circuit C with a set I of primary inputs and a set J of primary outputs. The minimal subcircuit containing all the nodes which have impact on a primary output j is called the cone of j and will be denoted by $C_j$. The principle of pseudo-exhaustive test is to consider C as the (not necessarily disjoint) union of its cones and to test each cone exhaustively (see fig. 16).

This provides a relative independence of a certain fault model in contrast to conventional approaches (within the cones all faults that do not cause a sequential behavior of C are detected, confer [ArMc84] ).

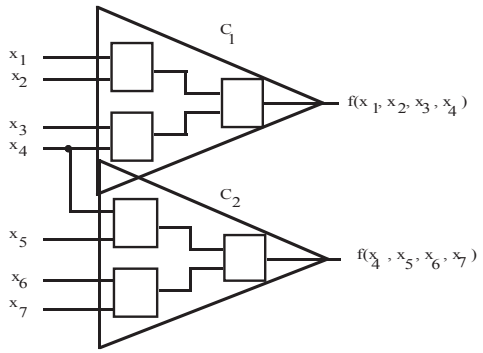



*Fig. 16:* Combinational circuit with 7 primary inputs and 2 primary outputs.

In general the necessary test length is drastically reduced in comparison with exhaustive test. For the number t of required test patterns the following inequation holds

$$(5) \qquad M := \max_{j \in J} 2^{|I_j|} \leq t \leq \sum_{j \in J} 2^{|I_j|} \leq |J| \cdot M,$$

where $I_j$ denotes the set of primary inputs of cone $C_j$, $j \in J$. It can be shown that $t \leq 2^{|I|-1}$, if no primary output depends on all primary inputs. In the example of figure 16 pseudo-exhaustive test can be done with 16+16 = 32 patterns in contrast to 128 patterns for exhaustive test. The pseudo-exhaustive test can be applied efficiently to general circuits if they are segmented for test-purposes [McBo81]. This can be done either by software ("path-sensitizing") or by some additional hardware ("multiplexer-partitioning"), where the hardware overhead should be minimized.

In order to do so, it is useful to represent C by a graph $G_C = (V, E)$. The set of vertices V corresponds to the set of nodes of C (primary inputs and outputs inclusively), and the set of edges E consists of all pairs $(v, w) \in V \times V$ such that the corresponding nodes in C are directly connected by a logic gate. The insertion of a multiplexer can be modeled as the cut of vertices or edges. To cut a vertex $v \in V$ means to replace v by two nodes v' and v'' such that the predecessors of v' are the predecessors of v, the set of successors of v' is empty and v'' is a source-vertex whose successors are the successors of v (see figure **17**). The cut of an edge can be defined in an analoguous way.

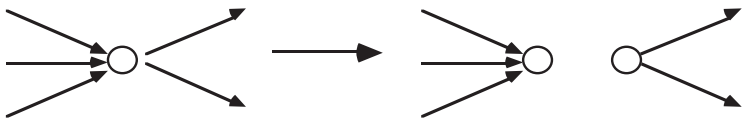


*Fig. 17:* Cut of a vertex.

The number of sources-vertices among the predecessors of a vertex v is called the dependence-level d(v) of v. In these terms the following problem has to be solved.

Segmentation Problem SP: Let G = (V,E) be a directed acyclic graph and a natural number. Transform G by a minimal number of cuts into a graph G' such that the dependence-level of each vertex is smaller than .

Some algorithms have been proposed in order to treat similar problems. Roberts and Lala [RoLa84] suggested a straightforward heuristic which produces a transformed graph G' with maximal dependence-level as close as possible to (but not necessarily below ). Starting from a source vertex of G the graph is searched until a vertex v with dependence-level exceeding is found. Depending on the difference between the respective dependence-level and , v or one of its predecessors is cut. If the resulting graph G' does not yet fulfill all requirements the procedure is applied to G'.

Patashnik [Pata83] proved the NP-completeness of a slightly modified segmentation problem SP' by reducing CLIQUE [GaJo79] to it. The cost function to be minimized is the resulting test length without consideration of the necessary hardware overhead. A heuristic for SP' using a mincut-technique was published by Archambeau in 1985 [Arch85]. McCluskey and Shperling [McSh87] recently published a paper on the use of simulated annealing for circuit segmentation.

In the following the considered circuit C is assumed to meet the requirements for an efficient pseudo-exhaustive test as discussed above. The straightforward approach to execute pseudo-exhaustive test is the successive test of the cones. But further reduction of the test length is possible. Consider the example shown in fig. 18.

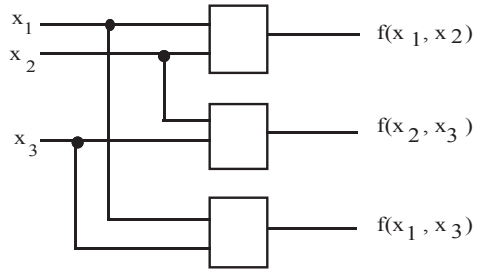



*Fig. 18:* Combinational circuit with 3 primary inputs and 3 primary outputs.

| $x_1$ | $x_2$ | $x_3$ | |
|---|---|---|---|
| 1 | 0 | x | |
| 1 | 1 | x | |
| 0 | 0 | x | |
| 0 | 1 | x | $C_1$ |
| x | 1 | 0 | |
| x | 1 | 1 | |
| x | 0 | 0 | |
| x | 0 | 1 | $C_2$ |
| 1 | x | 0 | |
| 1 | x | 1 | |
| 0 | x | 0 | |
| 0 | x | 1 | $C_3$ |

| $x_1$ | $x_2$ | $x_3$ |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |
| $C_1$ | $C_2$ | $C_3$ |

*Table 5:* Successive and parallel exhaustive test.

Each primary output depends on two primary inputs. Successive test of the cones is carried out by the left sequence of patterns (x denotes "don't care"), but actually the test can also be done by 4 patterns as shown on the right part of the above table.

Therefore the following problem has to be solved:

Compaction Problem CP: Let C be a combinational circuit with cones $C_1,\ldots, C_m$. Find a test set T such that $C_1$ through $C_m$ are tested exhaustively and |T| is minimal.

A graph-theoretical approach is based on the partitioning of "dependence-matrices" [McCl84]. The aim is to divide the primary inputs into a minimal number $k \leq n$ of classes such that all inputs belonging to one class can share the same test signal, i.e. a pseudo-exhaustive test can be done by the complete enumeration of GF(k). Hirose and Singh [HiSi82] showed the NP-completeness of this problem by reducing the graph k-colouring problem [GaJo79] to it. They suggested a tree search algorithm in order to get suboptimal solutions.

Other approaches are based on algebraic concepts. Let w denote the maximal number of primary inputs of the cones. Akers [Aker85] published an algorithm working with GF(2)-linear combinations ("linear sums") of $k \geq w$ "basic" independent signals. For example the circuit of figure 18 could be tested with two independent signals A and B applied to $x_1$ and $x_2$ respectively, and A $+_{GF(2)}$ B applied to $x_3$. This provides a pseudo-exhaustive test of the circuit, because A, B and A $+_{GF(2)}$ B are pairwise linearly independent over GF(2).

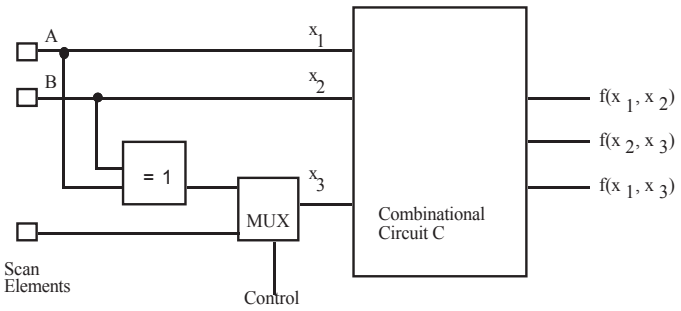



*Fig. 19:* Test of the circuit of figure 18 by "linear sums".

The "basic" independent signals can be generated by an LFSR or counter and the addition in GF(2) can easily be realized by EXOR-Gates. The resulting test length, however, is not minimal in general.

Other authors treated a more special problem CP'. For ä I let

$$\begin{aligned} pr : GF(2)^n &\rightarrow GF(2)^{|\ |} \\ (x_1,\ldots x_n) &\rightarrow (y_1,\ldots,y_n) \end{aligned}$$

be defined by

$$y_i := \begin{cases} x_i & \text{if } i \in \\ 0 & \text{else.} \end{cases}$$

In these terms CP' can be formulated as follows.

Problem CP': Find a set T ä $GF(2)^n$ such that for all ä I with | | = w the projection pr restricted to T is surjective and |T| is minimal.

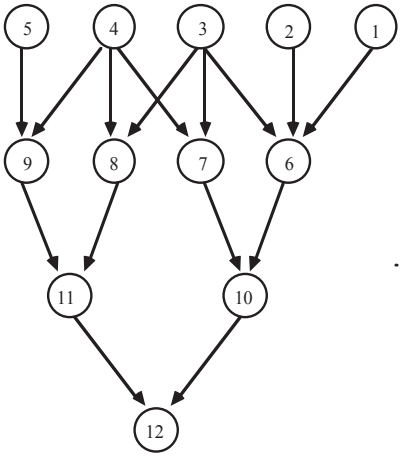
T is called a universal test set for the circuit C. The most elegant approach to solve CP' is to use shortened cyclic codes [WaMc86]. Test pattern generation can be carried out by an n-stage LFSR. But in general the construction of universal test sets demands the consideration of too many constraints and in some cases even the successive test of the cones is more efficient.

Within the framework of our system a tool PETT (Pseudo-Exhaustive Test Tool) has been developed which performs segmentation and test pattern compaction for a given circuit [Korn87].

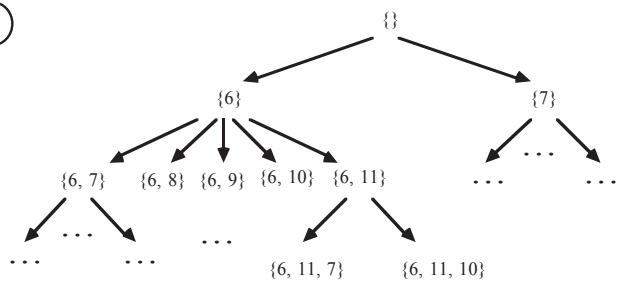
Concerning circuit segmentation we decided for hardware partitioning, since only this technique preserves all advantages of the pseudo-exhaustive test. Therefore an algorithm

solving SP had to be designed. Because of the already mentioned NP-complexity of the similar problem SP' a heuristic tree search procedure based on the hill climbing principle [Rich83] was chosen. Each vertex $x \in X$ of the search tree represents a set s(x) of vertices of the graph $G_C$. Cutting all vertices in s(x) transforms $G_C$ into a graph $G_C' = (V',E')$, and x is called a solution of SP if the maximal dependence-level of $G_C'$ is smaller than .

Starting with the empty set represented by the root-vertex the tree is built up as follows: Let the vertices of $G_C$ be ordered according to the signal flow in C. For $x \in X$ let $v^* \in V'$ be the vertex of minimal order with dependence-level exceeding . A direct successor of x represents the set $s(x) \cup \{v\}$, where v is a predecessor of $v^*$ neither a source-vertex nor in s(x). An example is shown in figure 20.




Graph representing a combinational circuit C.
The vertices are ordered according to the signal-flow in C.




Corresponding search tree.
{6, 11, 7} and {6, 11, 10} are solutions.

*Fig. 20:* Combinational circuit represented by a graph $G_C$ and the corresponding search tree.

The vertices in X are weighted by the following cost function:

$$F : X \; ' \; R$$

$$x \; ' \sum_{\substack{v \in V' \\ d(v) >}} \ln ( d(v) ) \; .$$

If $d(v) \leq$ for all $v \in V'$, F is set to zero.

At the root vertex of the tree the following search is started: For a vertex $x \in X$ all direct successors of x are generated and a vertex with minimal value of F is chosen. If no improvement of F is obtained within a certain number of steps, backtracking is initialized. If backtracking also fails to improve F a vertex representing $s(x) \cup \{w\}$ is chosen, where w is a predecessor of $v^*$ with maximal dependence-level . The algorithm stops when F is zero. A complexity analysis of this algorithm shows a quadratical increase of the computing time in the worst case, but experience shows rather a linear behavior.

The segmentation algorithm and an algorithm based on the paper of Roberts and Lala [RoLa84] were applied to some benchmark circuits [Brgl85]. Table 6 shows the results.

| Circuit | | number of cuts by PETT | number of cuts by the algorithm based on [RoLa84] |
|---|---|---|---|
| tn.c486 | 16 | 27 | 56 |
| tn.c499 | 16 | 8 | 20 |
| tn.c880 | 16 | 11 | 34 |
| tn.c1355 | 16 | 8 | 20 |
| tn.c2670 | 16 | 32 | 108 |
| tn.c3540 | 16 | 85 | 138 |

*Table 6:* Comparison of the segmentation algorithm of PETT with an algorithm based on [RoLa84].

For the test compaction the algorithm suggested by Hirose and Singh [HiSi82] has been implemented. Further research will concentrate on the synthesis of feedback functions for (non-)linear feedback shift registers executing test pattern generation.

### 3.5 Off-line test

Between external test techniques and self-test techniques there is the off-line test where special chips contain some of the test circuitry to reduce the hardware overhead. For a random test the external chips contain linear feedback shift registers, and can perform pattern generation and signature analysis (fig. 21).

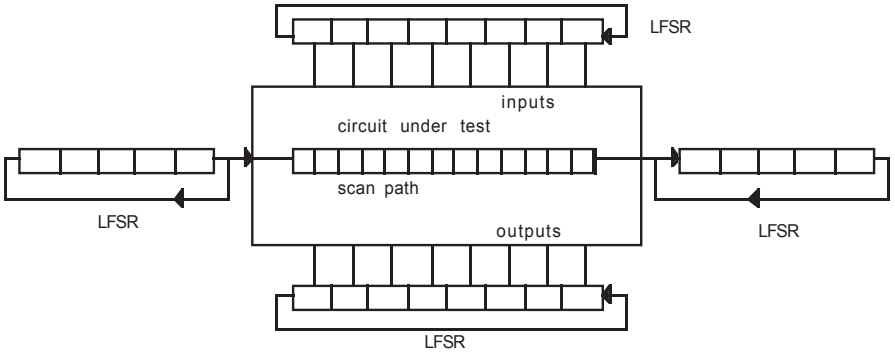



*Fig. 21:* Principle of the off-line test

For a conventional random test this procedure was already proposed in [EiLi83] and [BaMc82]. But also a chip generating optimized random sequences has been developed and processed [Berg85].

### 3.6 Regular Structures

A complete system supporting the automatic design for testability must not be restricted to random logic. For regular structures like RAMs, ROMs, bit-slices or PLAs test strategies should be selected and integrated. But the test problem for these structures is widely discussed in the textbooks (e.g. [BrFr76], [Fuji86]), and further research in this topic will concentrate on tool integration.

But to minimize the global hardware overhead the use of general BIST-structures for these modules has also been investigated.

If in fig. 11 one of the combinational networks is a PLA, then it is useless to add self-test circuitry to this PLA, since the test capabilities of the registers R1 and R2 can also be used. But in this case the random pattern testability of the PLA must be determined and optimal input probabilities have to be computed if GURTs are used. Both can be done with higher efficiency and high precision compared to random logic, using a Monte Carlo approach [Wu87d].

### 3.7 Validation and valuation

The external test and the self-test require both test and design algorithms. If the design modifications are done automatically, the logical correctness must be retained by construction. But the design changes also have impact on other parameters which are

estimated and given to the user and to the application level as objective function. Most important are:

- performance degradation is estimated by critical path analysis and simulation;
- the additional area is estimated by informations about the test cells in the library. Further work has to be done about the impacts of wiring and placement.

Detailed information is obtained about:

- test lengths and test application time for
  - random test
  - optimized random test
  - deterministic test with complete scan path
  - deterministic test with partial scan path
  - pseudo-exhaustive test
- fault coverage achieved by the different strategies.

Based on these informations a decision about the test strategies can be made at the application level. But in order to compute these objective functions more detailed, technology dependent knowledge is necessary. This is described in the next two sections.

## 4. Test algorithms at switch-level

The quality of the test is its fault coverage and it is well-known that fault modeling has to be technology dependent [Wads78], [GALI80]. The faulty behavior of MOS cells cannot be derived at gate level, especially if complex cells are used. But in general the computation at the electrical level (e.g. SPICE) is too expensive for complete fault modeling. One solution may be the so called switch-level algorithms, which are still much more accurate than those at gate-level by having only a small increase of the CPU-time [Haye87].

At switch-level the circuit is represented by a netlist of discrete operators standing for a transistor, a net, a resistor or a capacitor.

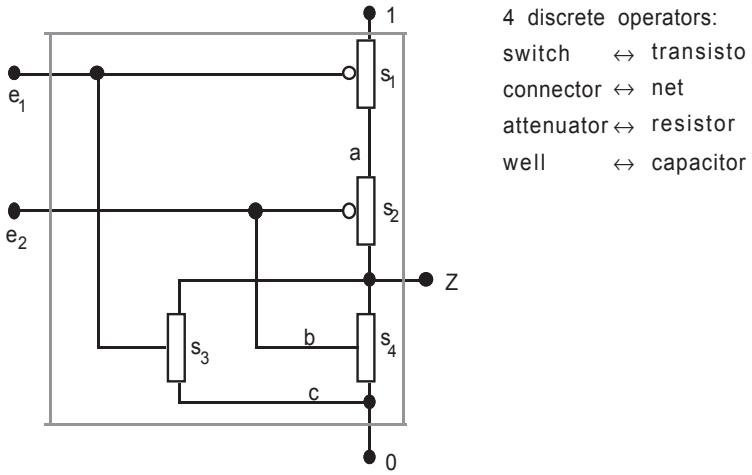



*Fig. 22:* Switch-level representation of a CMOS NOR-gate

The algorithms describe the electrical behavior of the circuit by a multi-valued logic, thus reducing the expensive continuous computations to algebraic operations on the restricted set of values.

There are mainly two approaches of switch-level modeling. The graph theoretical approach by Bryant ([Brya79], [Brya84]), and the work of Hayes ([Haye82], [Haye86]) based on lattice theory.

At the upper levels of the circuit description mainly three kinds of informations are needed:

Firstly one wants to know the behavior of the logic cell which is obtained by fault simulation. Secondly for a given fault within the cell test pattern generation has to be performed, and thirdly estimations of testability analysis are needed.

The simulation of the fault-free and faulty circuit is already state of the art. These simulators make it possible to look at the circuit at different description levels. They include switch-level, gate-level and functional level.

The generation of test patterns at switch-level is still in a very early state of development. The algorithms suggested in [CMN85] and [RRA85] are the first test pattern generators using the switch-level model of a circuit to get a better fault coverage for a large field of MOS-circuits, but further effort is required to generalize the classes of networks and faults that can be handled. At the CMU [Maly87] and the McGill University [AKCR87] such developments are done. Up to now even less work has been done on the testability analysis at switch-level.

Test pattern generation and testability analysis for MOS-cells are to be integrated into the presented system. The algorithms are using the lattice model by Hayes.

The test patterns generated at switch-level can be described at gate level, and the test generator SPROUT has to create them at the cell inputs. A survey of the underlying models and algorithms is found in [Holz87]. Informations about the testability of MOS-cells are needed if a self-test technique is selected. For a built-in test based on random patterns the necessary test lengths can only be computed if the detection probabilities of complex MOS faults are known. An algorithm computing fault detection probabilities based on the Hayes model is found in [WuHo88].

Moreover these informations are also needed if a pseudo-exhaustive self-test is applied. It is well-known that certain MOS faults induce a sequential behavior that is not guaranteed to be detected by an exhaustive test of the combinational function. If these faults are to be regarded too, a more complex self-test technique must be used described in [WuHe88].

Along with fault simulation, test pattern generation and testability analysis also the impact of the design modifications are analysed and validated at switch-level. The informations derived by these four tasks are used at gate level.

## 5. Automatic fault modeling

The switch-level netlist does not provide enough information for realistic fault modeling. In order to determine the effects of every possible process defect on the logic function, the only way is the fault injection at layout level. A common classification concerning the reasons of process defects is found in [Maly87], [Mang84], [MFS84]:

(1) lithographic defects

(2) defects caused by less process quality inducing wrong electrical characteristics.

On the other hand process defects can be classified as local or global. Local disturbances (spot defects [MSD86]) affect only a very small region of the chip, global disturbances affect a large area on the chip, thus changing the overall electrical characteristics of the chip. They are easily detected during the manufacturing process or during the parametric tests. The efforts to enhance testability as described in this paper are therefore concentrated on the detection of spot defects.

The majority of the spot defects is caused by the lithography process [SMF85]. Transparent spots and opaque particles in a lithography mask can cause shorts between non equipotential regions and breaks in equipotential regions and generate parasitic transistors, resistors and capacitors. In most cases the behavior of the chip will change. A special sort of lithographic defect is the mask justification which will cause global defects.

All of the lithographic defects concern the masks used during the manufacturing process, where the spot defects result in extra or missing material. Other spot defects are less oxid quality, airborne particles or very small droplets generated by the manufacturing equipment. If such disturbances are not removed from the surface of the wafer during a cleaning process, they will cause defects in much the same way as a punctual mask defect.

These both types of spot defects can be defined in the layout description of the circuit, e.g. in CIF [MeCo80]. Each area on the chip can be produced by combining masks. The gateoxid for example is the area where the polymask is transparent and the masks for p- or n-diffusion are transparent too. Fault injection is done by changing the layout description thus changing the specific area of the mask from transparent to opaque or vice versa. This CIF-file will now be the input of a circuit extractor, e.g. DRACULA [DRAC87] (together with a file describing the underlying technological process).

The output of DRACULA is a transistor netlist in SPICE-format. With this netlist it is easy to generate a circuit description on switch-level to start fault simulation, testpattern generation or testability analysis. For an exact prediction of the behavior of the faulty circuit we can do a circuit simulation with SPICE.

This approach of inductive fault analysis is also followed by Ferguson and Maly ([Ferg87], [Maly87]) and by the CVT project [MHMS85]. But they inject the defects stochastically into the layout descriptions, whereas our approach is analytical [ReWi87].

A layout description (CIF) of the fault free circuit is transformed into an internal representation, where all the possible faults at layout level are generated. By applying a set of rules the number of faults is minimized. In fig. 23 the fault "additional metal between line 1 and line 3" must also affect line 2. Therefore the fault "additional metal between line 1 and line 3 without affecting line 2" need not be modeled.

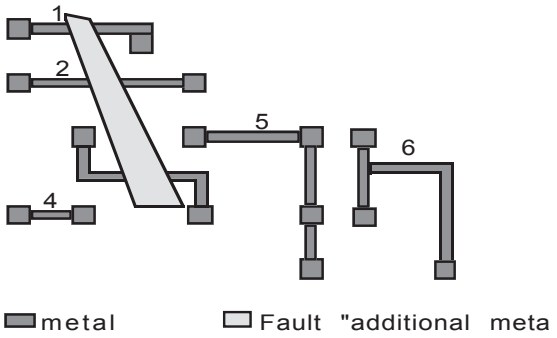



*Fig. 23:* Fault in the metal layer of a small cell

The algorithms for fault reduction are based on computational geometry, and described in detail in [ReWi87]. They create all different single faults at the layout level, where a single fault is defined as a topological change of the layout caused by a single defect or as a significant geometrical change (e.g. aspect ratio). The next steps are the generation of CIF descriptions of the deformed layouts and the generation of transistor netlists, which can be processed at switch-level.

## 6. System overview

As conclusion we give an overview of the entire system (fig. 24). The system accepts a functional or a structural entry for the circuit description. Based on the gate netlist and the library the different tasks are performed.

At gate, switch, and layout level both test and design algorithms are implemented: The cell layout is input in order to derive the list of possible faults at switch-level. These descriptions are processed during testability analysis and testpattern generation for the cell. This yields an input for the test algorithms at gate level. On the other hand at gate level circuit modifications like BILBOs, GURTs or the circuit partitioning are automatically implemented. The impacts of these design changes are checked at switch-level using real layout descriptions of the self-test cells. The layout library of special test cells is open and can be enlarged.

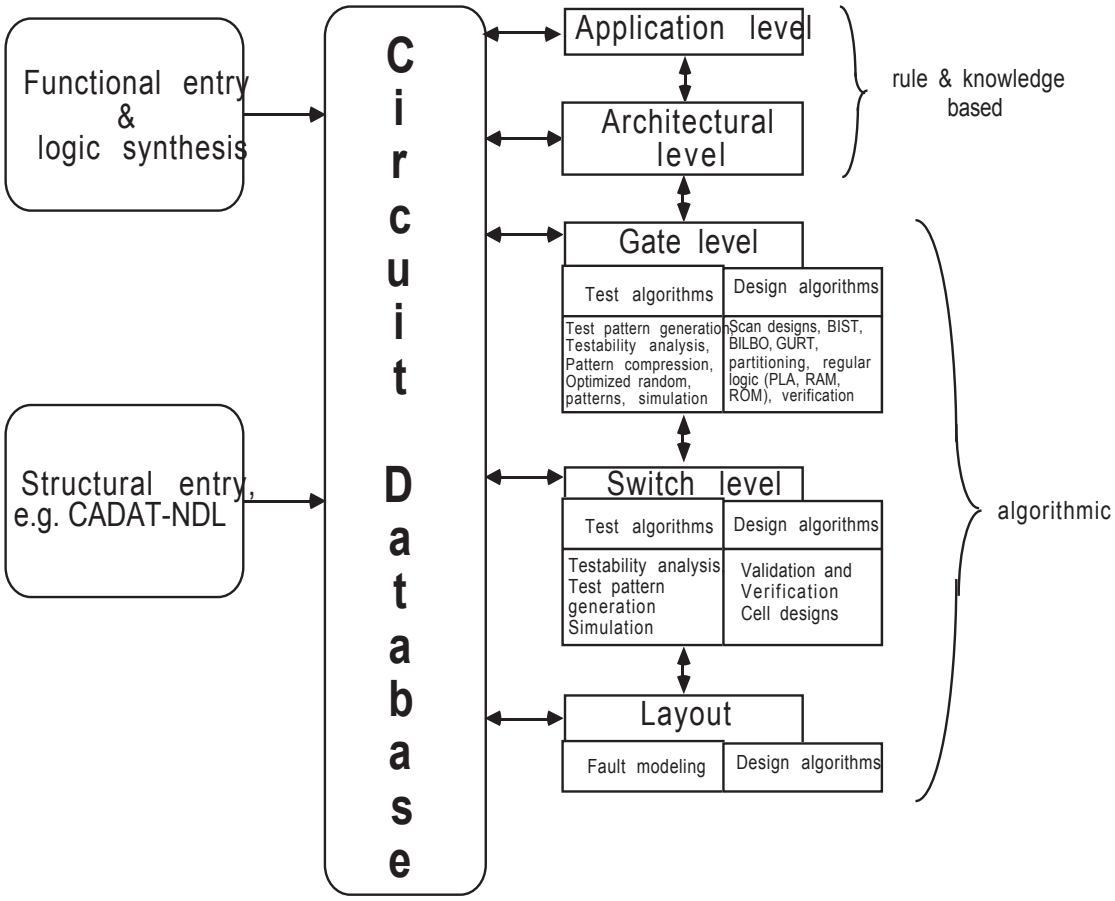



*Fig. 24:* Integrated tools for automatic design for testability (overview)

The results of the test and design algorithms at gate level are given to the rule and knowledge based system shell, where system and user will decide, which test strategy is implemented indeed.

## 7. Literature


AbBr85 Abadir, M.; Breuer, M.: A Knowledge-Based System for Designing Testable VLSI Chips; in: IEEE Design and Test, August 1985

AbCe82 Aboulhamid, E.M.; Cerny, E.: A Class of Test Generators for Built-in Testing; in: International Conference on Circuits and Components, 1982

ABS86 Seth, S. C.; Bhattacharya, B. B.; Agrawal, V. D.: An Exact Analysis for Efficient Computation of Random Pattern Testability in Combinational Circuits; in: Proc. International Symposium on Fault-Tolerant Computing, FTCS-16, 1986

AgAg75 Agrawal, V.D.; Agrawal, P.: Probabilistic Analysis of Random Test Generation Method for Irredundant Combinational Logic Networks; in: IEEE, Trans. on Comp., Vol. C-30, No. 8, 1975

AgCe81 Agrawal,V.K.; Cerny, E.: Store and Generate Built-in Testing Approach; in: International Symposium on Fault-Tolerant Computing , FTCS-11, 1981

AgJa84 Jain, S.K.; Agrawal, V.D.: STAFAN: An Alternative to Fault Simulation; in: Proc. 21st. Design Automation Conference, 1984

AgMe82 Agrawal, V.D.; Mercer, M.R.: Testability Measures - What Do They Tell Us?; in: Proc. IEEE International Test Conference, 1982

AGRA87 Agrawal, V. D.; Cheng, K. T.; Johnson, D.D.; Lin, T.: A Complete Solution to the Partial Scan Problem; in: Proc. IEEE International Test Conference 1987

AKCR87 Aboulhamid, M.; Karkouri, Y.; Cerny, E.; Rajski, J.: Fault Analysis in Switch-Level Networks by Enumeration; in: International Symposium on Fault-Tolerant Computing, FTCS-17, 1987

Aker85 Akers, S.B.: On the Use of Linear Sums in Exhaustive Testing; in: International Symposium on Fault-Tolerant Computing , FTCS-15, 1985

AMBL86 Ambler, A.P. et al.: Economically Viable Automatic Insertion of Self-Test Features for Custom VLSI; in: Proc. IEEE International Test Conference, 1986

Arch85 Archambeau, E.C: Network Segmentation for Pseudo-Exhaustive Testing; CRC Technical Report No. 85-10, Stanford, July 1985

ArMc84 Archambeau, E.C.; McCluskey, E.J: Fault Coverage of Pseudo-Exhaustive Testing; in: International Symposium on Fault-Tolerant Computing , FTCS-14, 1984

BaMc82 Bardell, P.H.; McAnney, W.H.: Self-testing of multichip logic modules.; in: Proc. 1982 IEEE Test Conf.

BDS84 Savir, J. et al.: Random Pattern Testability; in: IEEE Trans. on Comp., Vol. C-33, No. 1, Jan. 1984

Benn84 Bennetts, R.G.: Design of testable logic circuits; Addison-Wesley, 1984

Berg85 Bergsträsser, T.: Entwurf eines modifizierten Zufallsmustergenerators; Studienarbeit an der Fakultät für Informatik, Universität Karlsruhe, 1985

BGR87 Bidjan, I.; Glässer, U.; Rammig, F.J.: Knowledge Based Tools for Testability Checking, Tagung: Fehlertolerierende Rechensysteme, in: Informatik Fachberichte 147, September 1987

BrFr76 Breuer, M.A.; Friedman, A.D.: Diagnosis and Reliable Design of Digital Systems; Computer Science Press, 1976

Brgl85 Brglez, F. et al.: Accelerated ATPG and fault grading via testability analysis; in: Proceedings ISCAS 85, Kyoto 1985

Brya79 Bryant, R.E.: MOSSIM: A Logic-Level Simulator for MOS LSI, User's Manual Version 1.; MIT Laboratory for Computer Science, Cambridge, Mass. Sept. 1979

Brya84 Bryant, R.E.: A Switch-Level Model and Simulator for MOS Digital Systems.;in: Transact. on Computers, No. 2, 1984

BrZh85 Breuer, M.A.; Zhu, X.: A Knowledge Based System for Selecting a Test Methodology for a PLA; in: Proc. 22nd Design Automation Conference, 1985

ChKo75 Christofides, N.; Korman, S.: A Computational Survey of Methods for the Set Covering Problem; Management Science, Vol. 21, No.5, January 1975

CMN85 Chen, H.H.; Mathews, R.G.; Newkirk, J.A.: An Algorithm to Generate Tests for MOS Circuits at the Switch Level.; in: Int. Test Conference, 1985

Daeh83 Daehn,W.: Deterministische Testmustergenerierung für den eingebauten Selbsttest von integrierten Schaltungen; in: Großintegration, NTG-Fachberichte 82, 1983 Baden-Baden

DRAC87 DRACULA: Integrated Circuit Layout Verification System; User´s Reference Manual, February 87

EiLi83 Eichelberger, E.B.; Lindbloom, E.: Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test; in: IBM J. Res. Develop., Vol. 27, No. 3, May 1983

EiWi77 Eichelberger, E.B.; Williams, T.W.: A Logic Design Structure for LSI Testability; in: Proc. 14th Design Automation Conference, New Orleans, 1977

Ferg86 Ferguson, F.J.: Inductive Fault Analysis of VLSI Circuits; Thesis Proposal; private communication, 1986

Fuji86 Fujiwara, H.: Logic Testing and Design for Testability; The MIT Press 1986

FUNG85 Fung, H.S.; Hirschhorn, S.; Kulkarni, R.: Design for Testability in a Silicon Compilation Environment; in 22nd Design Automation Conference , 1985

FuSh83 Fujiwara, H.; Shimono, T.: On the Acceleration of Test Generation Algorithms; in: IEEE Trans. on Comp., C-30, 1983

GaJo79 Garey, M.R.; Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness; San Francisco, 1979

Gali80 Galiay, J. et al.: Physical vs. Logical Fault Models in MOS LSI Circuits, Impact on their Testability; in: IEEE Trans. Comp., Vol. C-29, No. 6, June 1980

GeNe84 Gerner, M.; Nertinger, H.: Scan Path in CMOS semicustom LSI chips? in: Proc. International Test Conference 1984

Goel80 Goel, P.: Test Generation Costs Analysis and Projections; in: Proc. 17th Desing Automation Conference, June 1980

Goel81 Goel, P.: An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits; in: IEEE Transactions on Computers, C-30, 1981

Golo67 Golomb, S.W.: Shift Register Sequences; Holden-Day, Inc., 1967

Haye82 Hayes, J.P.: A Unified Switching Theory with Applications to VLSI Design; in: Proc. of the IEEE, October 1982

Haye86 Hayes, J.P.: Pseudo-Boolean Logic Circuits; in: IEEE Transact. on Computers, July 86

Haye87 Hayes, J.P.: An Introduction to Switch-Level Modeling; in: IEEE Design & Test, August 1987

HiSi82 Hirose, F.; Singh,V.: McDDP, A Program for Partitioning Verification Testing Matrices; CRC Technical Report No. 81-13, Stanford, July 1982

Holz87 Holzinger, J.: Switch-level: Modelle und Werkzeuge; Interner Bericht Nr. 27/87 der Universität Karlsruhe, Fakultät für Informatik

IbSa75 Ibarra, O.H.; Sahni, S.K.: Polynomially Complete Fault Detection Problems; in: IEEE Trans. on Comp., Vol. C-24, No. 3, 1975

KOEN79 Koenemann, B. et al.: Built-In Logic Block Observation Techniques; in: Proc. Test Conference, Cherry Hill 1979, New Jersey

Korn87 Kornas, F.: Untersuchungen und Implementierung von Algorithmen zur Unterstützung des pseudo-erschöpfenden Selbsttests; Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, 1987

KrAl85 Krasniewski, A.; Albicki, A.: Simulation-free estimation of speed degradation in nMOS self-testing circuits for CAD applications; in: Proc. 22nd Design Automation Conference, 1985

Kras87 Krasniewski, A.: Knowledge-Based Selection of Self-Test Techniques in Computer-Aided Designing of Self-Testing VLSI Circuits; in: Proc. VLSI and Computers, COMPEURO 1987

Kunz87a Kunzmann, A.: Verbesserung der Testbarkeit von Schaltwerken durch die Integration eines unvollständigen Prüfpfades; Interner Bericht 25/87, Fakultät für Informatik, Univ. Karlsruhe, Oktober 1987

Kunz87b Kunzmann, A.: Produktionstest mit deduktiv bestimmten Testmustern: der Testmustergenerator SPROUT; Interner Bericht 26/87, Fakultät für Informatik, Univ. Karlsruhe, Oktober 1987

KuWu85 Kunzmann, A.; Wunderlich, H.-J.: Design automation of random testable circuits; in: Proc. ESSCIRC 1985, Toulouse

LBGG86 Lisanke, R. et al.: Testability-Driven Random Pattern Generation in: Proc. ICCAD, November 1986

Maly87 Maly, W.: Realistic Fault Modeling for VLSI Testing; in: Proceedings Design Automation Conference, 1987

Mang84 Mangir, T.E.: Sources of Failures and Yield Improvement for VLSI and Restructurable Interconnections for RVLSI and WSI: Pt. 1 - Sources of Failures and Yield Improvement for VLSI. Proceedings of the IEEE, Jun. 1984

MeCo80 Mead, C.; Conway, L.: Introduction to VLSI Systems; Addison Wesley, 1980.

MFS84 Maly, W.; Ferguson, F.J.; Shen, J.P.: Systematic Characterization of Physical Defects for Fault Analysis of MOS IC Cells; Int. Test Conference, 1984

MSD86 Maly, W.; Strojwas, A.J.; Director, S.W.:VLSI Yield Prediction and Estimation: A Unified Framework; IEEE Transact. on Computer Aided Design, Jan. 1986

McBo81 McCluskey, E.J.; Bozorgui-Nesbat, S.: Design for Autonomous Test; in: IEEE Trans. on Circuits and Systems, Vol. Cas-28, No. 11, Nov. 1981

McCl84 McCluskey, E.J.: Verification Testing - A Pseudoexhaustive Test Technique; in: IEEE Trans. on Computers, Vol. c-33, No.6, June 1984

McPa75a Parker, K.P.; McCluskey, E.J.: Analysis of Logic Circuits with Faults Using Input Signal Probabilities; in: IEEE Trans. on Comp., Vol. C-24, No. 5, 1975

McPa75b Parker, K.P.; McCluskey, E.J.: Probabilistic Treatment of General Combinational Networks; in: IEEE Trans. on Comp., Vol. C-24, No. 6, 1975

McSh87 Shperling, I.; McCluskey, E. J: Circuit Segmentation for Pseudo-Exhaustive Testing via Simulated Annealing; in: International Test Conference 1987

MHMS85 Morpurgo, S; Hunger, A.; Melgara, M; Segre, C.: RTL Test Generation and Validation for VLSI: An Integrated Set of Tools for KARL; in: CHDL, 1985, North-Holland, 1985

NeYu83 Nemirovsky, A.S.; Yudin, D.B.: Problem Complexity and Method Efficiency in Optimization; John Wiley & Sons, 1983

Pata83 Patashnik, O.: Circuit Segmentation for Pseudo-Exhaustive Testing; CRC Technical Report No. 83-14, Stanford, October 1983

Rich83 Rich, E.: Artificial Intelligence; McGraw - Hill International Editions, 1983

ReWi87 Rehme, H.; Wittke, M.: Automatische Fehlermodellierung aufgrund der Layout-Beschreibung von MOS-Schaltungen; Diplomarbeit am Institut für Informatik IV, Universität Karlsruhe, Oktober 1987

RoCa85 Rosenstiel, W.; Camposano, R.: Synthesizing circuits from behavioural level specifications; in: Proc. CHDL 1985, Tokio

RoLa84 Roberts, M.W.; Lala, M.Sc.: An Algorithm for the Partitioning of logic circuits; in IEE Proceedings, Vol. 131, No.4, July 1984

Roth78 Roth, J.P.: Sequential Test Generation; IBM Technical Disclosure Bulletin, Vol. 20, No. 8, Jan. 1978

RRA85 Reddy, M.K.; Reddy, S.M.; Agrawal, P.: Transistor Level Test Generation for MOS Circuits; in: Design Automation Conference, 1985

SaFo86 Samad, M. A.; Fortes, J. A. B.: Explanation Capabilities in DEFT - A Design-for-Testability Expert System; in: Proc.: IEEE International Test Conference 1986

SCHM86 Schmid, D.; Camposano, R.; Kunzmann, A.; Rosenstiel, W.; Wunderlich, H.-J.: The Integration of Test and High Level Synthesis in a General Design Environment; in: Proc. Integrated Circuit Technology Conference, Limerick, 1986

SCHU87 Schulz, M.H.; Trischler, E; Sarfert, T.M.: SOCRATES: A Highly Efficient Automatic Test Pattern Generation System; in: Int. Test Conference, 1987

SMF85 Shen J.P., Maly W., Ferguson F.J.: Inductive Fault Analysis of nMOS and CMOS Integrated Circuits; Carnegie-Mellon University, Research Report No. CMUCAD-85-51, 1985

TI80 The TTL Data Book, Texas Instruments, 1980

Tris80 Trischler, E.: Incomplete Scan Path with an Automatic Test Generation Methodology; in: Proceedings Test Conference, November 1980

Tris83 Trischler, E.: Testability Analysis and Incomplete Scan Path; in: Proc. ICCAD, 1983

Tsai83 Tsai, M.Y.: Pass Transistor Networks in MOS Technology: Synthesis, Performance, and Testing; in: IEEE Int. Symposiums of Circuits and Systems, 1983

VARM84 Varma, P. et al.: An analysis of the economics of self-test; in: Proc. International Test Conference, 1984

Wads78 Wadsack, R.L.: Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits; The Bell System Technical Journal, Vol. 57, No. 5, May-June 1978

WaMc86 Wang, L.T.; McCluskey, E.J.: Circuits for Pseudo-Exhaustive Test Pattern Generation; in: International Test Conference, 1986

Will86 Williams, R. M.: IBM Perspectives on the Electrical Design Automation Industry; keywords to IEEE Design Automation Conference, 1986

Wu84 Wunderlich, H.-J.: Zur statistischen Analyse der Testbarkeit digitaler Schaltungen; Interner Bericht 18/84, Fakultät für Informatik der Universität Karlsruhe, 1984

Wu85 Wunderlich, H.-J.: PROTEST: A Tool for Probabilistic Testability Analysis; in: Proc. 22nd Design Automation Conference, 1985, Las Vegas

Wu87a Wunderlich, H.-J.: Probabilistische Verfahren für den Test hochintegrierter Schaltungen; Informatik-Fachberichte 140, Springer-Verlag 1987

Wu87b Wunderlich, H.-J.: On Computing Optimized Input Probabilites for Random Tests; in: Proc. 24th Design Automation Conference, 1987, Miami Beach

Wu87c Wunderlich, H.-J.: Self Test Using Unequiprobable Random Patterns in: International Symposium on Fault-Tolerant Computing , FTCS-17, 1987

Wu87d Wunderlich, H.-J.: The Random Pattern Testability of Programmable Logic Arrays; in: Proc. IEEE International Conference on Computer Design, ICCD'87, New York 1987

WuHe88 Wunderlich, H.-J.; Hellebrand, S.: Generating Pattern Sequences for the Pseudo-Exhaustive Test of MOS-Circuits, submitted

WuHo88 Wunderlich, H.-J.; Holzinger, J.: The exact computation of fault detection probabilities for MOS cells; in preparation

WuRo86 Wunderlich, H.-J.; Rosenstiel,W.: On Fault Modeling for Dynamic MOS Circuits; in: Proc 23rd Design Automation Conference, 1986, Las Vegas