Design Automation of Random Testable Circuits

Arno Kunzmann
and
Hans-Joachim Wunderlich

Forschungszentrum Informatik
and
Institut für Informatik IV
(Prof. Dr.-Ing. Detlef Schmid)
Universität Karlsruhe

Zirkel 2
D-7500 Karlsruhe
Federal Republic of Germany

Abstract:

This paper describes the integration of a new tool for testability measurement and improvement into a design system for integrated circuits. The involved design system, CADDY (Carlsruhe Digital Design System), uses a functional description of a circuit written in a PASCAL like language and synthesizes a list of nets and real logical components. In this resulting structure all storing elements are configured as a scan path automatically. Therefore testability analysis and test generation may be restricted to pure combinational networks.

This is done by the software tool PROTEST (Probabilistic Testability Analysis). PROTEST determines the testability of a combinational circuit by random patterns, it computes the test length necessary to reach a given fault coverage with an also given confidence, and it proposes modifications of the random pattern sets, which leads to decreasing test lengths.

Keywords: Design for Testability, Random Pattern Test, Design Automation.

1.0 Introduction:

In a recent paper one of the authors presented the test tool PROTEST [Wu85], which supports the test of the combinational parts of a circuit by random patterns.

By the increasing complexity of high integrated circuits the importance of testing by random patterns grows, since the automatic test pattern generation (ATPG) by deductive procedures (e. g. D-algorithm) is one of the most expensive parts in VLSI design. The costs are high even if scan paths, scan sets, LSSD or similar "design for testability" methods are used (see [EiWi77], [MuSa81]). These methods reduce ATPG for arbitrary digital systems to ATPG for combinational circuits. But the size of the resulting combinational networks exceed the capacity of the available algorithms on general purpose computers. Test generation for integrated circuits with 10^4 - 10^6 transistors requires a computing time which can be counted by magnitudes of days [Goel80],[Goel81].

The test of digital systems by random patterns makes it possible to dispense with the generation of test patterns from a description of the circuit structure. Random testing can be done either by using additional test hardware on the chip for self testing or by an external test.

In order to do self test all storing components of the chip are usually configured as one or more feedback shift registers during testing [Much81]. In this special testmode these registers generate pseudo-random patterns for the combinational part of the circuit and compress the responses by signature analysis [HeLe83].

An external test is carried out by shifting randomly generated patterns through a scan path.

Both external and self test reduce the test problem to the test of combinational circuits. The application of random patterns requires the determination of the necessary test length to get the desired fault coverage of the commonly used stuck-at fault model on the gate level. The test length can be computed by the procedure PROTEST.

This procedure is integrated into the synthesis system CADDY. The user of CADDY has to specify the logical function of the intended circuit in a PASCAL like language and has to fix some other constraints concerning chip area and timing. Then CADDY generates a structure description on gate level of a circuit performing the specified function and regarding the given restrictions. If CADDY can not observe the restrictions a message is given to the user. The synthesized structure description consists of a list of elements of a cell library connected by nets.

This proceeding saves the expensive task of circuit verification which would be necessary for a manual or a not fully automatically generated design. But verification is only superfluous, if the synthesis system completely integrates all those hardware features into the generated structure, which are necessary to get high testability. Therefore the system has to produce designs with integrated test control and scan path.

Since this is done, CADDY saves the two expensive tasks of test generation and verification. But as a disadvantage sometimes the synthesized chips have lower efficiency then those, which are generated by an experienced designer manually. Optimizing stragies try to minimize this disadvantage [SCHM84]. But CADDY is able to transform user defined logical structures into a scan design too.

2.0 The synthesis system

This chapter gives an overview of CADDY. Starting point is a description of the function of the intended chip. This will be transformed into a description of the logical structure on the gate level and then converted by a placement and routing system into a geometry. We mainly deal with the first step, i.e. the transformation of the behavioural into the structural description. In order to transform logic structures into chip layouts several commercial tools are available. The examples of this paper are generated by the layout system VENUS [GHHS84] for automatic placement and routing of CMOS standard cells.

2.1 The digital system specification language (DSL)

The user of CADDY has to write the functional description of his chip in the Digital Design Specification Language DSL. The language offers the data types LOGICAL, ONE_COMP, TWO_COMP, BCD, FIXED and FLOAT. The allowed operators include the usual logical, arithmetic and relational operators, which are controlled by control structures similar to PASCAL [SCHM84].

2.2 Synthesis

CADDY transforms the functional description into circuit structures. Global actions not constrained to a certain point in time, such as resets, interrupts, etc. are naturally specified in an applicative part of DSL. Sequential algorithms, the behaviour of a finite automaton and sequential behaviour in general are often more comfortably described by imperative procedures (imperative part).

The transformation consists of five tasks ([CKR84],[RoCa85]):

(1) Compilation of the DSL program
(2) Synthesis of the data path for the imperative parts of the DSL program
(3) Synthesis of the sequential control for the imperative parts
(4) Synthesis of data and control structures for the applicative part of the DSL program
(5) Assembling of (2), (3) and (4) and allocation of the global control.

First the DSL program is compiled and an internal representation is created for all the subsequent algorithms.

The second task consists of the construction of a dataflow graph, i.e. an abstraction of the designers intended dataflow from a DSL program. Then global and local optimizations are performed. Finally the nodes of the optimized dataflow graph must be transformed into circuit components depending on technology parameters.

The control synthesis (3) is carried out by constructing a finite automaton. Based on the DSL program and the data flow graph above a state transition table is constructed. The control automaton is synthesized by LOGE [BiDi84], offering different hardware solutions such as random logic and PLAs.

The synthesis of the applicative part is done in a similiar way. The main difference is the lack of an explicit control structure.

The final result of all five parts is one netlist, consisting of nodes and components of a cell library and thus representing the final structure of the specified circuit.

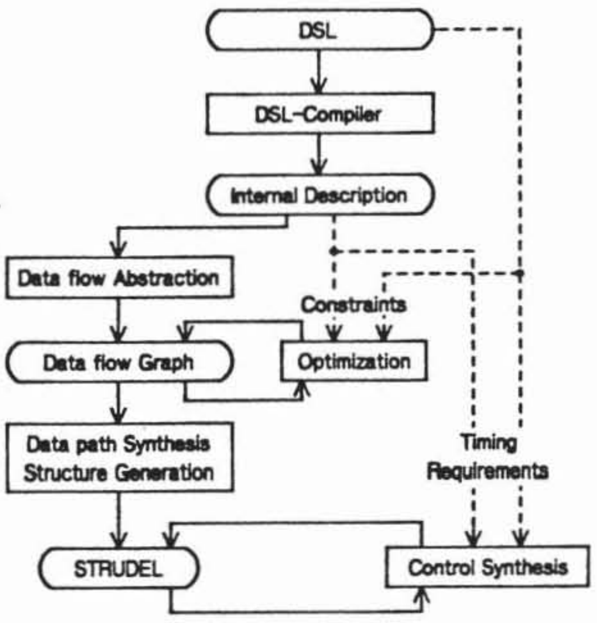


Figure 2.1: Synthesis of the imperative parts [RoCa85]

Within the CADDY system these structures are represented by STRUDEL[CaTr84] (STRUcture DEscription Language) programs. In the following the communication with tools for test pattern generation or simulation is based on this description.

3.0 On random pattern testing

Random pattern testing is a way to dispense with ATPG and needs less requirements for the test equipment too. Feedback shift registers can generate random patterns and can compress the response data of the device under test. But the problem arises to determine the number of patterns which are necessary to achieve a given fault coverage with a given confidence. This test length depends on the probabilities for each fault to be detected by a randomly generated pattern.

Let F be the set of all equivalence classes of all detectable faults. Let p_f be the probability that a fault $f \in F$ is detected by a random pattern. Furthermore we make the simplifying assumption that the detection of some faults by a pattern forms statistically independent events. Then the probability P that all faults of F are detected by N random patterns can be computed by

(1) $$P := \prod_{f \in F} (1-(1-p_f)^N).$$

This formula is used by PROTEST in order to compute the test lengths, since the bias caused by statistical dependency is low if N is large.

B[illegible] in order to get a worst case estimation of the size of pattern sets we assume, all faults have the same low detection probability p. Setting n := |F| we get

$$P = (1-(1-p)^N)^n. \tag{2}$$

Since N will be large we have

$$\ln(P) \simeq -n(1-p)^N, \tag{3}$$

and naturally P shall be near 1 and therefore

$$(1-P)/n \simeq (1-p)^N. \tag{4}$$

Finally this yields the estimation

$$N \simeq \ln(n/(1-p))/P. \tag{5}$$

Formula (5) shows how the necessary test length depends on important circuit parameters:

- the test length grows with ln(n), n is the number of faults;
- the test length grows with $\ln(1/(1-P))$, P is the required confidence to achieve the fault coverage;
- the test length linearly increases with 1/p, where p is the minimum detection probability larger than 0.

All known procedures for ATPG have an exponential worst case complexity and their average complexity is between n^2 and n^3. For random testing the exponential effort implicitly lies in the third topic. If a combinational network has I primary inputs, then the detection probability of a detectable fault may fall down to 2^{-I}, - if a random pattern set is used, which sets each primary input to logical "1" with probability 0.5.

In order to support random pattern test PROTEST offers the following features:

3.1 Estimation of signal probabilities

An arbitrary tupel $X := \langle p_i \mid i \in I \rangle \in [0,1]^I$ determines random pattern sets, which stimulate each primary input $i \in I$ with signals being logical "1" with the signal probability p_i. If the signal probabilities at the primary inputs of a combinational network are given, PROTEST estimates the signal probabilities for each node within the network. PROTEST only estimates, because all known algorithms to compute them exactly, show exponential complexity and recently one of the authors has proven that computing signal probabilities is NP-hard [Wu84].

Whereas the recently proposed dividing algorithm [BDS84] computes upper and lower boundaries for signal probabilities and the tool STAFAN [AgJa84] achieves its results by simulation, PROTEST analytically computes an estimation of the signal probabilities with nearly linear effort.

3.2 Estimation of fault detection probabilities

For each stuck-at fault f PROTEST estimates the probability that this fault is detected, if the circuit is stimulated by an element of a random pattern set having the input signal probabiblities $\langle p_i \mid i \in I \rangle$.

Several approaches to transform the computing of signal probabilities into the computing of fault detection probabilities have been proposed (see [BDS84], [AgJa84], [Wu85]). They differ from accuracy and necessary computing time. PROTEST offers three possibilities:

- estimation of the fault detection probability by analyzing a faulty and a fault-free circuit.
- estimation of the probability that the faulty value and the correct value differ at the node under regard and that a single path is simultaneously sensitized to a primary output.
- a simple modelling of the signal flow.

3.3 Computation of the test length

For a given circuit PROTEST computes for every fault an estimation of the fault detection probability. With these estimated values formula (1) is evaluated in order to determine the number N of patterns. This can be done very fast, since only the faults with very low detection probabilities will affect the size of N.

3.4 Optimizing input signal probabilities

There exist combinational networks which need a very large N to detect all faults, if each primary input has signal probability 0.5. But the tupels $X := \langle p_i \mid i \in I \rangle \in [0,1]^I$ determine for each primary input a specific signal probability and for each fault a detection probability $p_f(X)$.

With some natural number N the formula

$$J_N(X) := \prod_{f \in F} (1-(1-p_f(X))^N) \tag{6}$$

is an estimation of the probability that a pattern set with size N and with the input signal probabilities X detect the whole F. For every combinational circuit the real function $J_N: [0,1]^I \dashrightarrow [0,1]$ can be maximized by a tupel $\langle p_i \mid i \in I \rangle$, thus leading to maximal fault detection.

PROTEST includes an optimizing procedure, which finds a local maximum of J_N. The examples in [Wu85] and in sect. 8 show that the necessary pattern count can be drastically reduced in such a way.

3.5 Static fault simulation

One presumption of the scan design is that the circuits are pure synchronous. Therefore there is no need to regard time delays within the combinational parts of the circuits during fault simulation and PROTEST only includes a package for static fault simulation. This can be used to validate the predicted detection probabilities, test lengths and fault coverages.

If random patterns are simulated which obey optimized signal probabilities proposed by PROTEST, a small test set can be economically selected. Therefore PROTEST can be used as a test pattern generator similar to the D-algorithm or the path sensitization algorithm, too.

4.0 Extraction of combinational circuits

4.1 Scan design

In order to yield testable designs, an access to all the internal latches of the synthesized network is necessary. To provide controllability and observability all storage elements are interconnected into a shift chain. Then test patterns can be shifted in and test results can be shifted out. Thus the internal latches must work as intended by the chip specification during the functional (normal) mode, but during the test mode the shift capability has to be provided. This is achieved by adding to each internal latch a slave latch. The physical combination of both these latches is functionally representing a single internal storage element. In the following polarity-hold shift register latches (SRL) are used. Figure 4.1 shows the symbolic representation and implementation in AND-INVERT gates of a SRL with D as data input and C as clock.

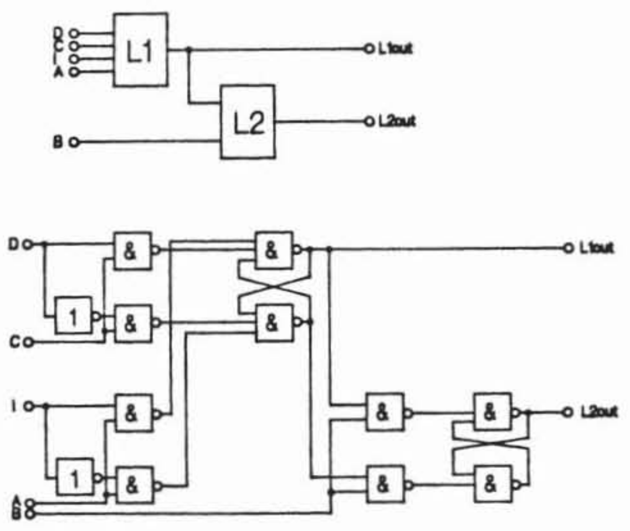


Figure 4.1: Polarity hold SRL
(a) symbolic representation
(b) Implementation in AND-INVERT gates

While the signals A and B are both 0, the latch L1 operates exactly like a polarity hold latch (input I, output L2). During test mode, operating as a shift register, data from the proceeding stage is gated into L1 via I by a change of the A signal to 1 and then back to 0. After A has changed back to 0, the B signal gates the data from the latch L1 into L2.

To extract the combinational logic of circuits, transformations of the library elements and of the STRUDEL description must be performed.

4.2 Transformation of the library

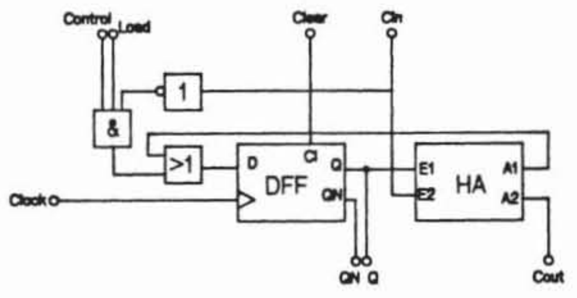
Transformations of the library are needed to yield a logic network consisting of storage elements which can be integrated into a scan path. All the storage elements must be replaced by a SRL and in case of need, the individual control part is added.

4.3 Transformations of the description

There mainly exist three points how to transform a given STRUDEL program into a description suitable for the subsequent extraction algorithm.

(i) Sequential circuits need accessable storing elements.

Any sequential circuit (e.g. counter) needs a description where all its internal storage elements are accessable. Figure 4.2 shows the substitution of a cascadable 1-bit-counter cell and its corresponding STRUDEL program.



```
TYPE counter (A1/O,A2/O,A3/O,E1/I,E2/I,
                          E3/I,E4/I,E5/I):
COMPONENT
   HA1 / CHA01A:   HAout,Cout,Q,Cin;
   DFF / CDF02A:   Q,Qn,D,Clock,Clear;
   OR  / COR02A:   D,HAout,ANDout;
   AND / CAN03A:   ANDout,Control,CinInv,
                   Load;
   INV / CDR01A:   CinInv,Cin;
   cnt / counter:  Q,QN,Cout,Control,Cin,
                   Load,Clock,Clear;
END counter;
```

Figure 4.2: Substitution of the cascadable 1-bit-counter cell and its resulting STRUDEL-description

To transform the DSL operations into real components on the gate level, the CADDY system offers a generator, which generates multipliers and dividers of different width and realization (sequential or combinational). All the elements generated by the structure-generator consist theirselves only of the admissible primitive storage elements.

(ii) Representation of the control signals.

All the synthesized logic networks have clock signals, not connected with data. This requirement is fulfilled by the STRUDEL descriptions created by CADDY and must also be satisfied by external descriptions.
For testability analysis all the elements of a network must consist of SRLs and a combinational part, to control the according function. Then control signals of multiplexers (STROBE), ENABLE-, SET-, RESET-signals are all regarded as data signals.

(iii) Primary bidirectional pads are replaced by primary outputs and inputs.

If pads are used both as data input and output they are transformed into data-inputs and data-outputs. During the test all these I/Os are controlled by the individual enable controls, which changes the direction of the corresponding pads one clock period after the propagation of the current test

4.4 The extraction algorithm

First of all the algorithm must check that there exist only loops with storage elements. For all the synthesized networks, this is guaranteed by construction.

Any storage element gets a so-called "pseudo-input" rsp. "pseudo-output", because at all these points a direct access into the network is enabled. These control and observation points increase the amount of the "primary" inputs and outputs. So prepared the following three steps of the algorithm are executed:

1. Mark all the output nets of the inputs and pseudo-inputs.

2. Mark all the output nets of those components, whose inputs are completely (i.e. every input net) marked up to this point.

3. Finish, if no more nets can be marked.

The result of the algorithm is a sorted netlist.

The extraction of the combinational logic net is done for the whole network, consisting of the data and control part. Circuits with control inputs, stimulated by the control part, are stimulated during the test mode by the test patterns of the scan path, extending on both control and data part.

4.5 Test control

All the chips under test have a Scan In and Scan Out port, two clocks A and B, and an enable-select input. During test mode, either A or B are logic 1, else both A and B are logic 0. Additionally to these clocks, the clock C stimulates the chip during its normal mode (see fig. 4.1).

The maximum clock frequency to shift a test vector through the scan path is not dependent on the type of the flipflop substituted by a SRL. Supplementary combinational logic, needed to work as the demanded flipflop type increase only the resulting combinational logic network and do not concern the shift chain itself.

To calculate an upper bound of the propagation time, it is necessary to allow all the combinatorial signals to propagate at their worst case delays through the combinatorial logic net. A worst case simulation based on the STRUDEL description estimates the maximum propagation time.

(1) Shift-in/Shift-out
All the SRL's of the chip are connected according to the description of the STRUDEL program. The number of the pseudo inputs determines the number of necessary non-overlapping independent clocks, shifting the testvectors in and out at high-speed. The shift frequency is only dependent on the structure of the SRL's.

(2) Propagation
After the testvector is shifted in, A and B are set to 0 for the needed propagation time and the functional latch clocks will be activated to store the test results of the combinational logic net into the latches.

5.0 An example

5.1 The function of the designed chip

In the following we present a circuit which evaluates the rational function

(7) $(A_3x^3+A_2x^2+A_1x+A_0)$ / $((B_2x^2+B_0)-(B_3x^3+B_1x))$.

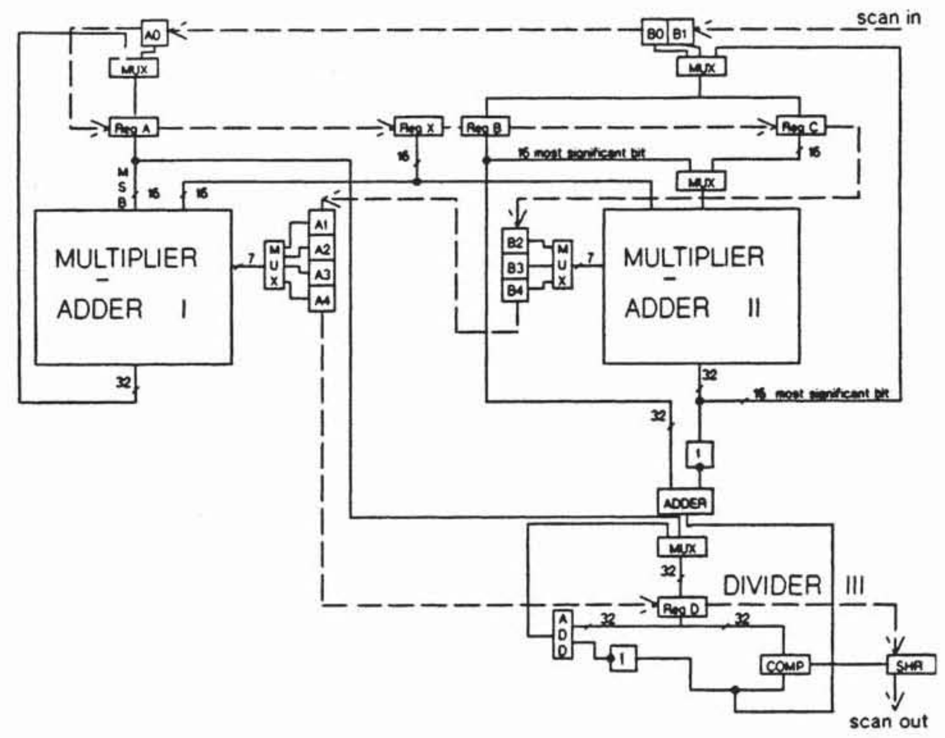


Figure 5.1: The data part of the exponentiation example

It is assumed that the coefficients are integers between 0 and 127 and that x is in the intervall [0,1). If we set $A_0 := B_0 := 120$, $A_1 := B_1 := 60$, $A_2 := B_2 := 12$, and $A_3 := B_3 := 1$, then this rational function approximates the exponential function with accuracy $2.8*10^{-5}$. If for some reasonable integers i the values of e^i are stored in a table, then the exponential function $e^{i+x} = e^i * e^x$ can be evaluated in high speed.

5.2 The generated structure

Because of the disjoint generation of the control and data part in the CADDY system, the following description concerns only the data part. Figure 5.1 shows its graphic representation of the STRUDEL program.

Timing requirements cause the implementation of two components (I + II), which compute according to the HORNER scheme. The result of a multiplication with X, followed by an addition of Ai rsp. Bi is fed back into the input register A rsp. B or C. Because of the range of values for the Ai and Bi, the addition terms are only of width 7. These modules are marked as "MULTIPLIER-ADDER". The second of these modules seperately computes positive and negative terms in order to substract the intermediate results.

Finally the division (III) is performed by a divider, shifting the quotient, comparing divisor and quotient and subtracting (2-complement) the divisor from the divider, if the first operand is less than the second.

5.3 The testability analysis of the circuit

If conventional random pattern sets are used, PROTEST recognizes a very poor testability of the DIVIDER and very good testability of the two MULTIPLIER-ADDER. The DIVIDER is only a relatively small part of the whole circuit, but as already mentioned only the faults with the lowest detection probability determine the necessary test length. Table 1 shows the sizes of each part, the transistor numbers are based on a CMOS standard cell library. This library will also be used to compute delay times.

Complete circuit (comb. part)	44 932
DIVIDER	4 990
MULT-ADD	17 510

Table 1: Size in transistors

In order to detect all detectable faults of the circuit with confidence 0.99 PROTEST demands the size of pattern sets listed in table 2.

Complete circuit	$2.0 * 10^{11}$
DIVIDER	$2.0 * 10^{11}$
MULT-ADD	$1.2 * 10^{3}$

Table 2: Necessary test lengths (not optimized)

But using those test sizes random testing becomes very ineconomical. A propagation time of 730 ns within the combinational part of the circuit and a shifting time of 4200 ns for each complete pattern yield a total test time of about 10^6 seconds.

The optimizing procedure of PROTEST however yields input signal probabilities, which require much less patterns. Table 3 shows the necessary test lengths for optimized random pattern sets.

Complete circuit	$1.5 * 10^{5}$
DIVIDER	$4.0 * 10^{4}$
MULT-ADD	$3.3 * 10^{2}$

Table 3: Necessary test lengths (optimized)

Using the optimized test set the testing time will be less than 1 second.

5.4 The validation of the testability prediction

In order to validate the proposed test lengths and input signal probabilities, two random pattern sets were generated for each circuit, one of them has the conventional input signal probabilities 0.5 and the other set is optimized.

One can expect, that a rather small number of optimized or not optimized patterns will yield a high fault coverage of the whole circuit, since only the DIVIDER part requires a large pattern count.

Fig. 5.2 shows that this indeed is the fact. 10 000 optimized patterns detected 99.25 % of all faults and 10 000 not optimized patterns detected 97.4 %. In absolute numbers these are 111 rsp. 367 undetected faults.

The advantages of optimizing the random pattern sets will become even more distinct, if the parts of the circuit are separately regarded. Fig. 5.3 shows, that optimized patterns detect a large number of faults of the DIVIDER at a very early stage.

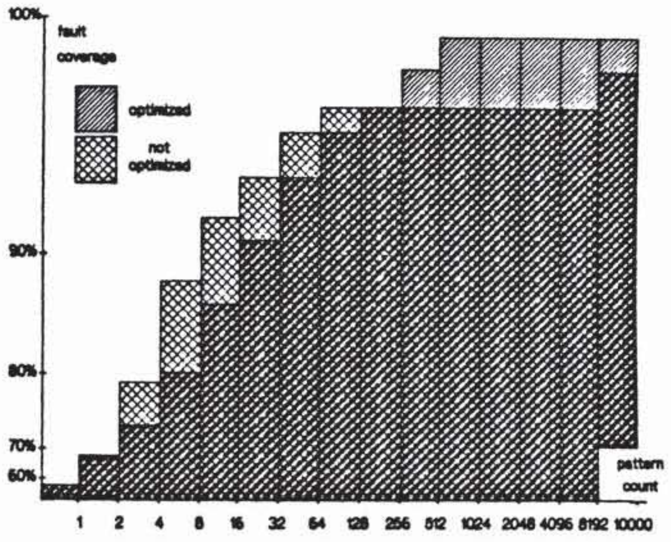


Figure 5.2: Fault coverage vs. pattern count (complete combinational part)

Simulating 10 000 optimized patterns 99.2 % of the faults of the DIVIDER are detected, but by using conventional test sets only a fault coverage of 81.2 % is reached. The high testable MULT-ADD yields a small difference between optimized and not optimized patterns: 99.65 % of the faults of the MULT-ADD are detected by only 235 patterns in the optimized case and the same number of conventional patterns detected 99.2 % of the faults.

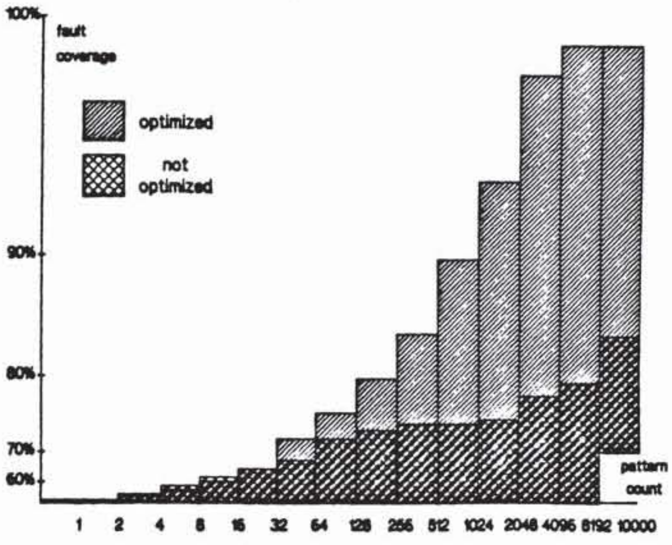


Figure 5.3: Fault coverage vs. pattern count (DIVIDER)

References:

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

[BDS84] J. Savir, G.S. Ditlow, P.H. Bardell: Random Pattern Testability; IEEE, Trans. Comp., Vol. C-33, No. 1, Jan. 1984

[BiDi84] G. Biehl, A. Ditzinger: Programmsystem LOGE, Benutzerhandbuch Version 3.1; ISDATA GmbH, Karlsruhe, 1984

[CaTr84] R. Camposano, L. Treff: STRUDEL - Eine Sprache zur Spezifikation der Struktur digitaler Schaltungen; Interner Bericht Nr. 7, Fakultät für Informatik der Universität Karlsruhe

[CKR84] R. Camposano, A. Kunzmann, W. Rosenstiel: Automatic Data Path Synthesis from DSL Specifications; IEEE, Proc. of Int. Conf. on Computer Design, ICCD, 1984

[EiWi77] E.B. Eichelberger, T.W. Williams: A logic design structure for LSI testability; Proc. 14th Design Automation Conference, pp. 462-468, June 1977

[Goel80] P. Goel: Test Generation Costs Analysis and Projection; Proceedings 17th Design Automation Conference, pp. 77-84, June 1980

[Goel81] P. Goel: An implicit enumeration algorithm to generate tests for combinational logic circuits; IEEE Trans. on Comp., Vol. C-30, No. 3, March 1981

[GHHS84] E. Goettler, L. Haschigh, E. Hoerbst, G. Sandweg et al.: Entwicklung von kundenspezifischen Schaltungen; Elektronik, Hefte 19-22, 1984

[HeLe83] J.H. Heckmaier, D. Leisengang: Fehlererkennung mit Signaturanalyse; Elektron. Rechenanl. 25, 1983, H. 3, 109 - 116

[Much81] J. Mucha: Hardware Techniques for Testing VLSI Circuits Based on Built-In Test; Proc. COMPCON 81, Feb. 1981

[MuSa81] E.I. Muehldorf, A.D. Savkar: LSI Logic Testing - An Overview; IEEE Trans. on Computers, Vol. C-30, No.1, January 1981

[RoCa85] W. Rosenstiel, R. Camposano: Synthesizing Circuits from Behavioural Level Specifications; 7th International Symposium on Computer Hardware Description Languages and their Applications, Tokio 1985

[SCHM84] D. Schmid et al.: Automatischer Entwurf hochintegrierter Schaltungen aus Beschreibungen der Schaltungsfunktion; Informatik-Fachberichte 88, 14. GI-Jahrestagung, Braunschweig Okt. 1984, Springer-Verlag

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

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