The Integration of Test and High Level Synthesis in a General Design Environment

D.Schmid, R.Camposano, A.Kunzmann, W.Rosenstiel, H.-J.Wunderlich
Universitaet Karlsruhe and Forschungszentrum Informatik

Prof. D. Schmid, Postfach 6380, D-7500 Karlsruhe
F. R. Germany
Phone: 721-608-3960

Abstract

This paper describes the integration of new tools for both test and synthesis of integrated circuits. The presented design system CADDY (Carlsruhe Digital Design System) automatically transforms a functional description into a circuit structure. Besides this logic synthesis the system also automatically integrates a complete or incomplete scan path. The software tool PROTEST (PRObabilistic TESTability analysis tool) determines the random testability of the combinational parts of synthesized circuits and suggests optimized input signal probabilities to minimize the necessary test length. To generate these test patterns on chip a specific test hardware is proposed.

1.0 Introduction

The main purpose of the CADDY-system, described in section 2, is the automatic synthesis combined with automatic design for testability (ADFT) from a behavioural level description language as input (fig. 1.1). By generating circuits with integrated test hardware CADDY ensures, that the final resulting circuit is still correct by construction. Testability is achieved by including Automatic Test Pattern Generation (ATPG).

This paper is the final form and no version of it will be submitted for publication elsewhere.

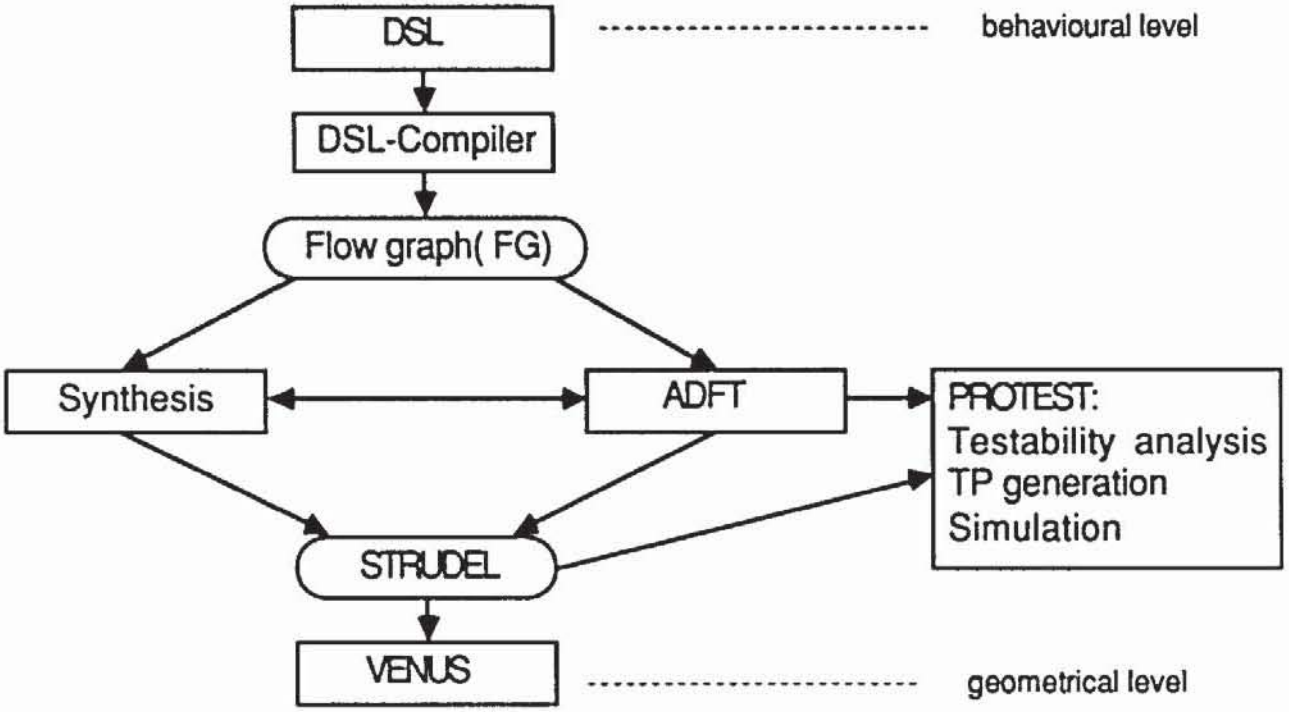


Fig. 1.1: The basic structure of the synthesis system

The increasing complexity of integrated circuits enhances the importance of testing by random patterns. ATPG by deductive procedures, even by modern algorithms like PODEM [Goel81], is one of the most computing time consuming tasks in VLSI design. The reduction of the ATPG effort by transforming sequential circuits into combinational ones (e.g. by scan path, scan set, LSSD) often results in "too large" combinational circuits. Test pattern generation for integrated circuits with about 50.000 transistors requires already a computing time of some days [Goel81].

The test of digital systems by random patterns allows to dispense with the generation of test patterns based on a description of the circuit structure. To analyse the random testability of circuits, the software tool PROTEST (Probabilistic Testability Analysis) was developed. In section 3 some basic algorithms and features are presented.

In section 4 the integration of complete and incomplete scan paths is described.

One of the most important features of PROTEST is that random patterns can be optimized for specific combinational circuits. The result of the optimization is a vector, assigning a specific probability between [0,1] to each primary input of the circuit. This yields a higher fault grading by a simultaneously minimized number of test patterns compared to the conventional random test with test patterns of the probability 0.5. In section 5 feed back shift registers are described, generating those adjusted test patterns and performing signature analysis.

The main tasks of the synthesis system are depicted in figure 1.2.

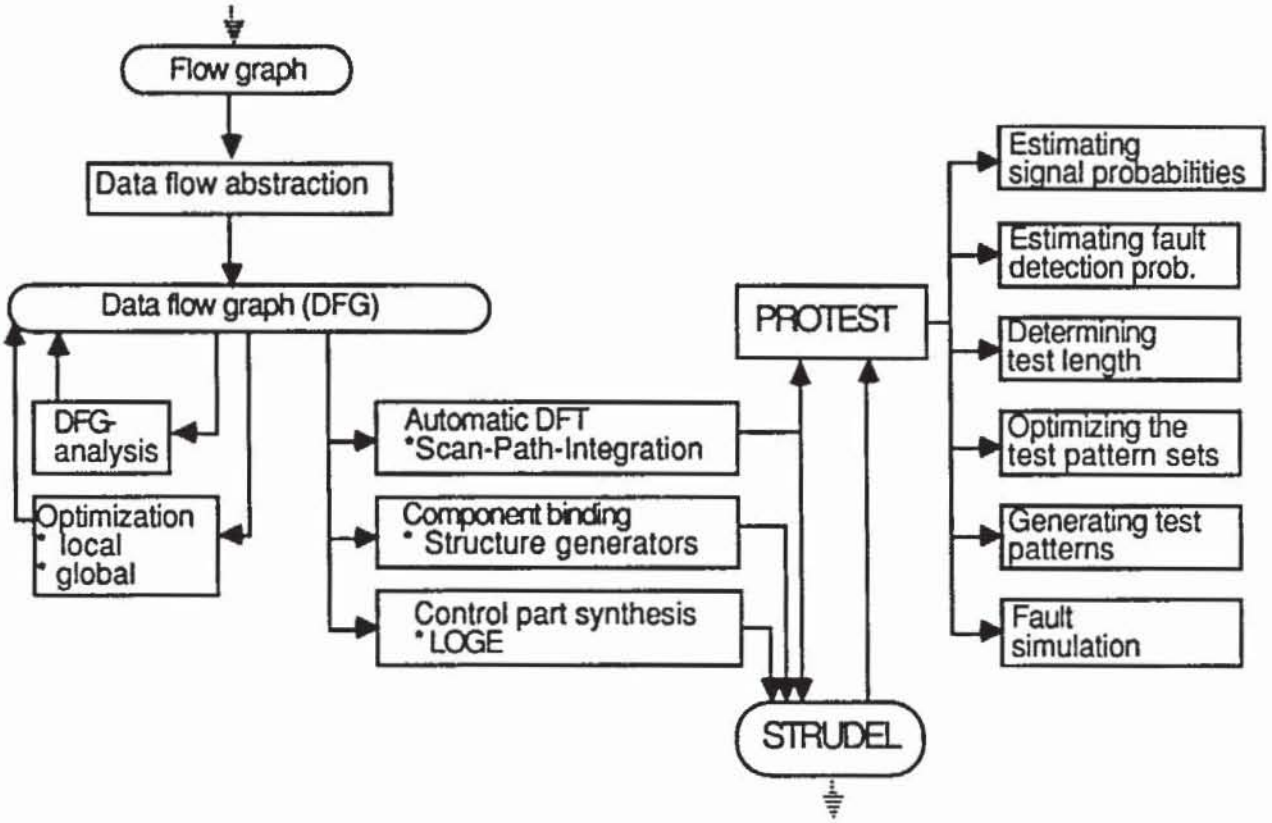


Fig. 1.2: Main tasks of test and synthesis

At the end of the paper an example is presented to demonstrate the integration of the synthesis system and PROTEST, also generating the optimized probabilities. The results are validated by a static fault simulation.

2.0 Synthesis

High level synthesis gains increasing importance in the design of integrated circuits. Synthesis keeps away the details of the geometrical and electrical level and - more advanced - even the logical details from the designer. High level synthesis allows complete design automation from a behavioural specification to the layout. Therefore it is especially well suited for the rapid prototyping of hardware.

Other advantages of automatic synthesis are less need for design verification through correctness by construction and the availability of IC-technology to the non-expert designer. This offers not only economic advantages but also the possibility of protecting know-how.

Several approaches of high level synthesis systems have been reported over the last years, their lack of efficiency prevents their use in an industrial

environment [Ullm84], [CKR84], [DPST81], [DSST82], [Kowa84]. In our synthesis system we concentrated on the optimization of the synthesis algorithms during the last years [RoCa85]. They are mainly based on data flow analysis [Rose84]. The optimizations include

- minimization of storing elements,
- loop optimization,
- global trade-offs of concurrency and area and
- rule based local transformations

[DaJo80],[Pasch85],[Zipp83].

2.1 The specification language

The most important features of the specification language DSL (Digital system specification language) [CaWe84] which we use as input language for our synthesis system are:

- A DSL program may include one *applicative part* and many *imperative* processes:
Global actions not constrained to a certain point in time, such as resets, interrupts, etc. are naturally specified in the applicative part. Sequential algorithms, finite automata and sequential behaviour in general are simply described by imperative processes.

- *Concurrency* can be specified in DSL both at the level of single operations and between imperative processes.

- *Modularity* and *hierarchy* in DSL is supported by a powerful abstraction mechanism. Like an abstract data type a DSL-module is defined by its interface (inputs and outputs) and the different functions which are available from the outside.

- The delay of single operations or of groups of operations can be specified in absolute time or in clock cycles. This feature allows to specify *timing restrictions* for the synthesis.

- In addition, *global restrictions* of timing, chip area, electrical parameters etc. can be specified in the overall part of a DSL program.

2.2 Synthesis algorithms

In our synthesis algorithms different techniques are combined. *Data flow analysis* is used to minimize the storing elements. *Symbolic execution* allows

the optimization of loops. By *heuristic approaches* and by some ideas from *expert systems* local and global optimizations of the synthesized structures are performed.

Synthesis starts from the internal representation of the behavioural specification - the so called *Flow graph* - which is generated by the DSL compiler. In detail the Flow graph combines three different types of arcs sharing the operators as common nodes. The different arc types correspond to

- the predecessor-successor relation of operations,
- the connectivity relation of the operators by the operands and
- the timing constraints between operations.

The main synthesis steps consists of the *data flow* and *control* generation, the global and local optimization and finally the component binding. The generation of the control and the dataflow is described in in detail in [Rose84]. A summary can be found in [RoCa85].

Our efforts in the field of optimization should be briefly discussed here. The two main concepts are *global* and *local* optimization. Some examples of so called global optimizations are

- evaluation of constant expressions only once
- detection of common subexpressions
- extraction of loop independent parts out of loops
- "folding" of different occurrences of the same operator to only one component
- realization of similar operators by an universal component (ex. ALU)
- life time analysis to share registers

Local optimizations include basic logical transformations at the NAND/NOR/AND/OR/NOT level, where about 20 rules are implemented mainly based on [DaJo80]. Very effective are also local transformations on a higher level based on components (COM) like adders, counters, multiplexers (MUX), registers (REG), comparators etc. An example is given by figure 2.1.

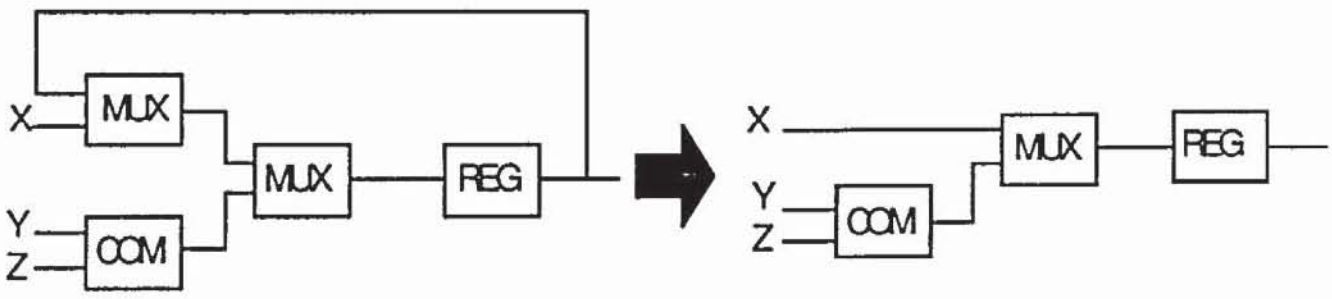


Fig. 2.1: Example for local transformations

All the different optimization steps can be performed in any ordering and any cycles. Evaluation procedures allow backtracking to optimize the trade-off between chip area and speed requirements.

The last synthesis step is the so called *component binding* [RoCa85]. During this phase the optimized synthesis results, which are still technology independent, have to be mapped to technology specific components. Instead of using a fixed set of layout modules in a "closed" library the synthesis invokes structure generators that provide a practically unlimited number of building blocks. The structure generators are parametrized by the number of inputs and outputs, speed and area requirements. Structure generators are available for registers (with and without scan-path), adders (ripple carry or carry look-ahead), ALU's, multiplexers, decoders, gates, drivers etc. Actually our structure generators provide an interface to the gate array, standard cell and general cell system VENUS from the SIEMENS AG [GHHS84]. For a given structure VENUS generates a complete layout by automatic placement and routing. Currently we are interfacing our synthesis system with different silicon compilers.

3.0 The probabilistic testability analysis tool PROTEST

The aim of the synthesis is to achieve the generation of testable circuits. All the automatic design for testability (ADFT) tools, described in section 4, can be controlled by the probabilistic testability analysis tool PROTEST. In the following all the essential features of this tool are summarized, details are given in [KuWu85],[Wu85].

3.1 Estimation of the signal and fault detection probabilities

Since computing signal probabilities is NP-hard [Wu84], PROTEST **estimates** the signal probabilities for each internal node of the network with linaer effort, based on the specific signal probabilities at the primary inputs of a combinational circuit.

For the faults of the circuit, the probability is estimated, that a random pattern with a specific input signal probability detects this fault. To compute those estimations PROTEST offers some possibilities, which differ concerning their computing effort and attainable precision, respectively: it can be estimated, that the values of the faulty and fault-free circuit differ if a random pattern is applied or one can only apply a simple modeling of the signal flow [WU85].

3.2 Computing the test length

Based on the fault detection probabilities PROTEST determines the number N of test patterns in order to achieve a required fault coverage with a demanded confidence. Let p_f be the probability that a fault $f \in F$ is detected by a random pattern. The probability P that all faults of F are detected by N random patterns can be computed by [Wu84]

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

It is assumed, that the detection of some faults forms statistically independent events, which is asymptotically fullfilled by increasing N.

3.3 Optimizing the input signal probabilities

During a conventional random test all the input probabilities are 0.5. Let $X := \langle p_i | i \in I \rangle \in [0,1]^I$ be a tupel that assigns for each primary input a specific signal probability and for each fault a detection probability $p_f(x)$. Then

$$(2)\quad J_N(X) := \prod_{f \in F} (1-(1-p_f(X))^N)$$

is the probability, that N random patterns with input signal probability X detect all the faults of the circuit. PROTEST includes a heuristic procedure to determine a X with maximum J_N.

In order to validate the predicted detection probabilities, test lengths and fault coverages, PROTEST offers a package for static fault simulation. A static simulation is sufficient, since the synthesized circuits are synchroneous, and therefore time delays need not to be regarded.

3.4 Complete and incompiete scan path

The synthesis system generates pure synchroneous, sequential circuits. An obvious way to achieve a structure, PROTEST can deal with is the use of the scan design technique.

The complete scan path and the test control are impiemented on the chip in a straightforward way, which is described in [KuWu85]. Deterministic or random generated test patterns are sequentially shifted in and test results are shifted out simultaneously. Each of those steps is followed by a single propagation step.

A further test option determines the synthesis of circuits with an incomplete scan path. The goal is to include the minimum number of storage elements into the scan path, that yields a sequential circuit of a given depth D. This depth is defined as the maximum number of connected storage elements between any two scan path elements.

The sequential depth D determines the extent of the resulting iterative combinational network, which is in the order of $2^{D}*n$, n is the gate number of the original network.

Identifying the minimum number of storage elements can be reduced to well-known partition problems of graph theory. The basic graph contains the dependencies between the storage elements of the circuit. It indicates whether there exists a direct or indirect (through combinational elements) connection between any two storage elements.

4.0 Feedback shift register configuration

The implementation of a special test pattern generator (TPG) is necessary, to obtain the proposed probabilities at each primary input of the combinational network. Signature analysis (SA) must be performed either. BILBOs (Built in Logic Block Observers) [KMZ79] realize both TPG and SA, but only if probabilities of the input signals of 0.5 are needed.

The following chapter outlines the necessary modifications and extensions of BILBOs, in order to dispense with this restriction.

4.1 The length of the basic test pattern generator (BPG)

The number of the required test patterns determines the length of a feedback shift register: a linear feedback shift register (LFSR) of length N can generate up to $2^{N}-1$ different test patterns [HeLe83]. Figure 4.1 shows the typical structure of a LFSR.

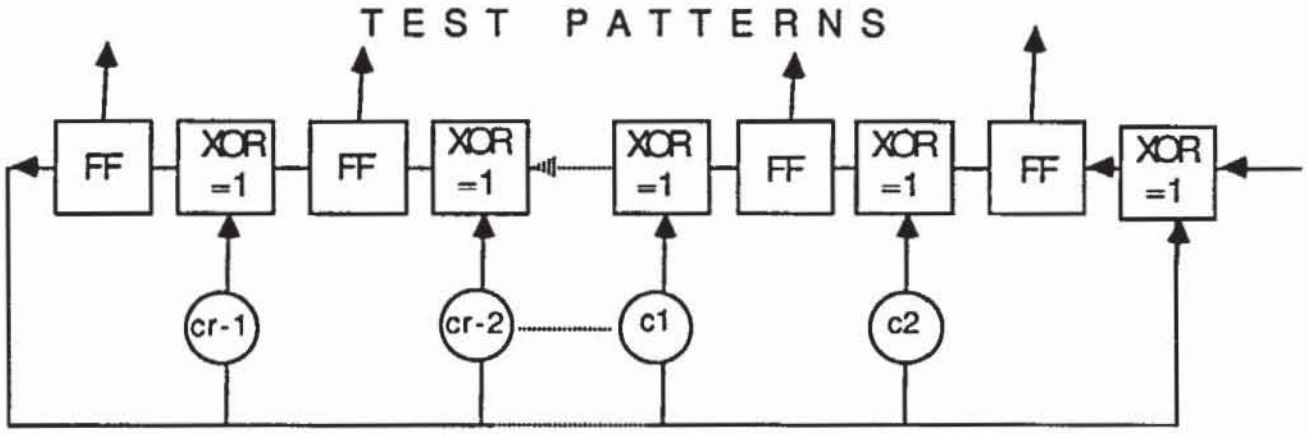


Fig. 4.1: Structure of a LFSR

Such a LFSR is one part of the needed test hardware, the so called "basic pattern generator" (BPG). All the generated TPs have the probability 0.5 at each position.

4.2 The offered probabilities

At this point we have a certain amount of 1-bit-registers which are logical "1" with the probability of a half. These are the basis to generate the required probabilities unequal 0.5. Figure 4.2 shows the probabilities, the proposed test hardware can produce and the correspondent logic operations (AND/OR) to generate those values.

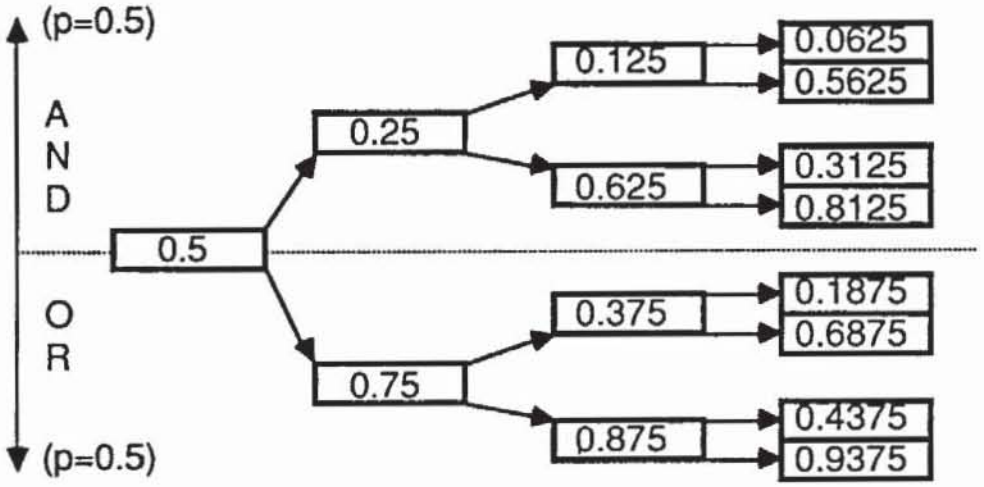


Fig. 4.2: Probabilities and their generation

For example in order to generate logical "1" with the probability 0.4375, you need a 3-input-OR with input probabilities of 0.5 and a 2-input-AND. The inputs into this gate are the result of the OR-gate (0.875) and a probability of 0.5. In summary, 4 probabilities of 0.5 are required, which can be branched off the BPG.

The precision the input probabilities can be adjusted is 0.0625. This value has proved to be sufficient, since fault detection probabilities depend linearly on the input sinal probabilities [Wu86].

4.3 The independency of the test vectors

The results of PROTEST are computed under the assumption of independent values for logic "1" and "0" between each position of the test patterns. The correlations between those positions should be minimized. These correlations heavily depend on the choice of the sampling points of the BPG.

The algorithm to determine such points, yields in pairs minimized correlations. A good choice are such feedback points whose summarized distancies yield a maximum value. Because sampling points must also be feed back points, the number of feed backs of the BPG is maximized with regard to the resulting maximal period, which has to be long enough. The algorithm is described in detail in [KuWu84], the whole test hardware is illustrated in the example (fig. 5.3).

4.4 The generation of the probabilities

Let us assume, probabilities of 0.25 (twice), 0.75, 0.375, 0.625, 0.8125 and 0.1875 are required. Figure 4.3 shows the structure of the corresponding "register chain", consisting of 3 "register classes".

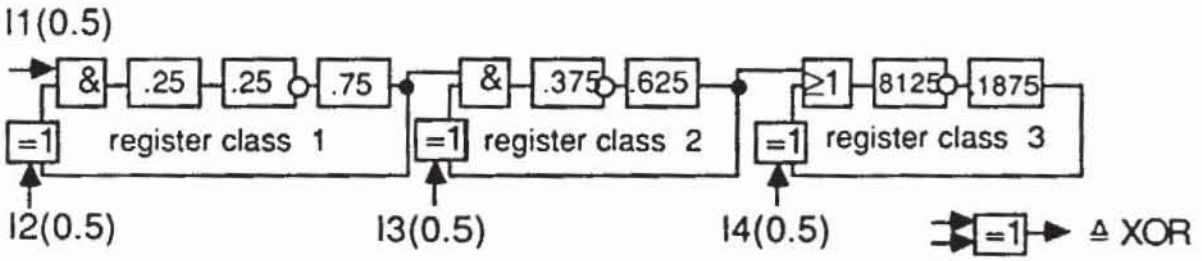


Fig. 4.3: An example of a "register chain"

All the 4 inputs I1..I4 have the probability 0.5 and are originated in the BPG. In order to be independent within each register class and between the other classes, it is necessary to feed back the register classes. The amount of register classes determines the amount of the additional EXOR gates.

To generate all the possible 14 specific probabilities, 4 register chains are necessary containing 7 register classes. Each of those probabilities is generated only once and is marked in the tree (fig. 4.2). This guarantees that by each new register chain a probability of the tree is generated which is not marked up to this point.

5.0 An example

Currently we are synthesizing a PROLOG processor based on Warren's "abstract PROLOG machine" [TiWa84]. An important condition which must be evaluated very efficiently is the so called "trail"-condition:

IF (A < B) OR (A > C AND A < D) THEN . . .

This condition decides the binding of variables and is based on the 16-bit pointers A, B, C and D pointing to different stacks. The structure which is automatically generated by our synthesis system is described in fig. 5.1.

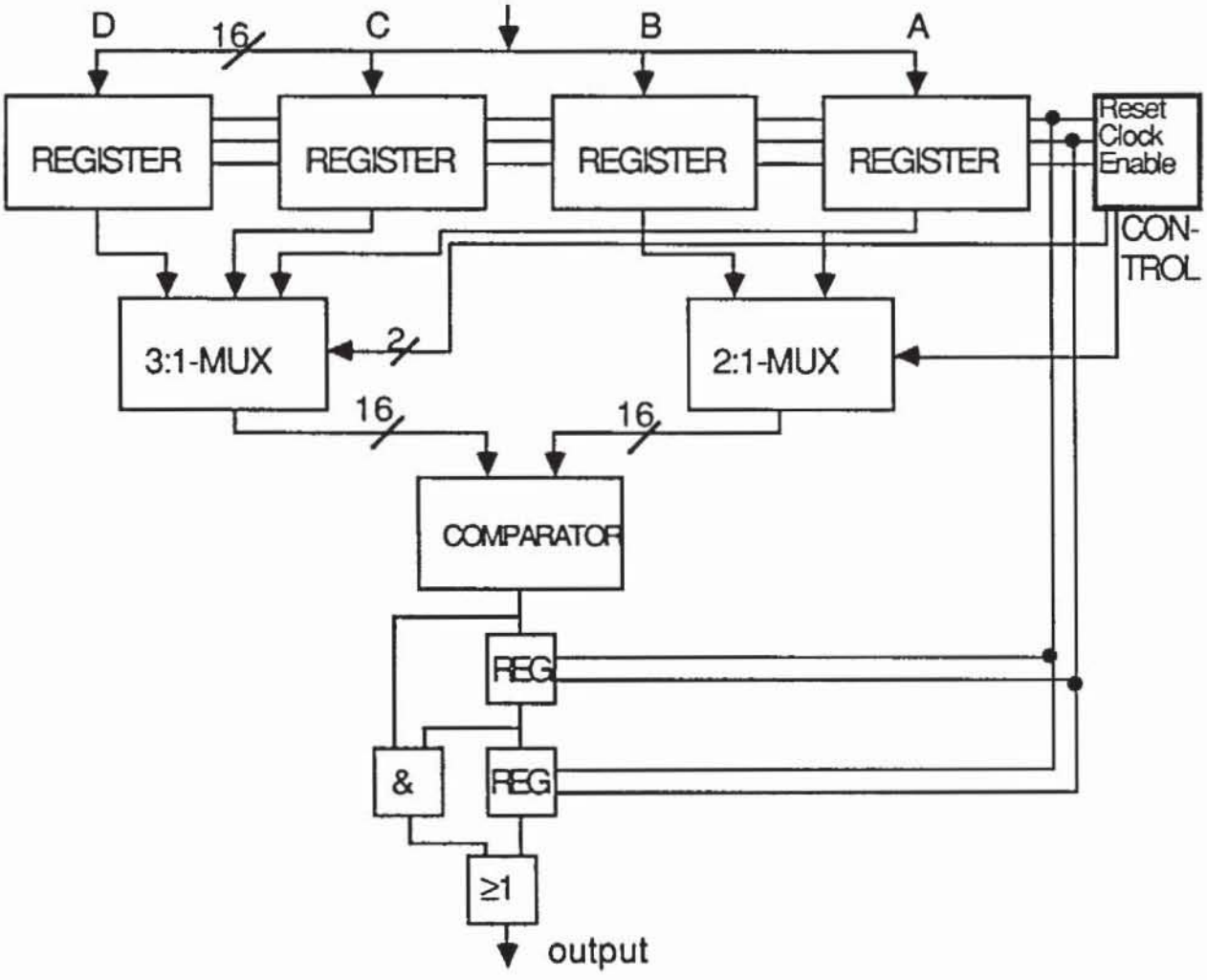


Fig. 5.1: Block diagram of the circuit

The final result of the comparisons is valid at the end of the third clock phase after the input data are valid. The control part consists of 2 D-flipflops and has three states. The transistor count of the combinational part of the circuit is about 3.600.

5.1 Testability analysis and optimization

The test by conventional random patterns with signal probability 0.5 results in only a poor testability of the circuit. In order to detect all the detectable faults with confidence 0.99, PROTEST determines $1.5 * 10^7$ test patterns.

The optimization procedure of PROTEST sets the input signal probabilities to specific values. In the optimized case PROTEST requires a necessary test length of only 6.170 test patterns.

5.2 The static fault simulation

In order to validate the results of PROTEST the circuit was simulated by two random pattern sets, one of them with signal probabilities 0.5, the other with the proposed optimized probabilities.

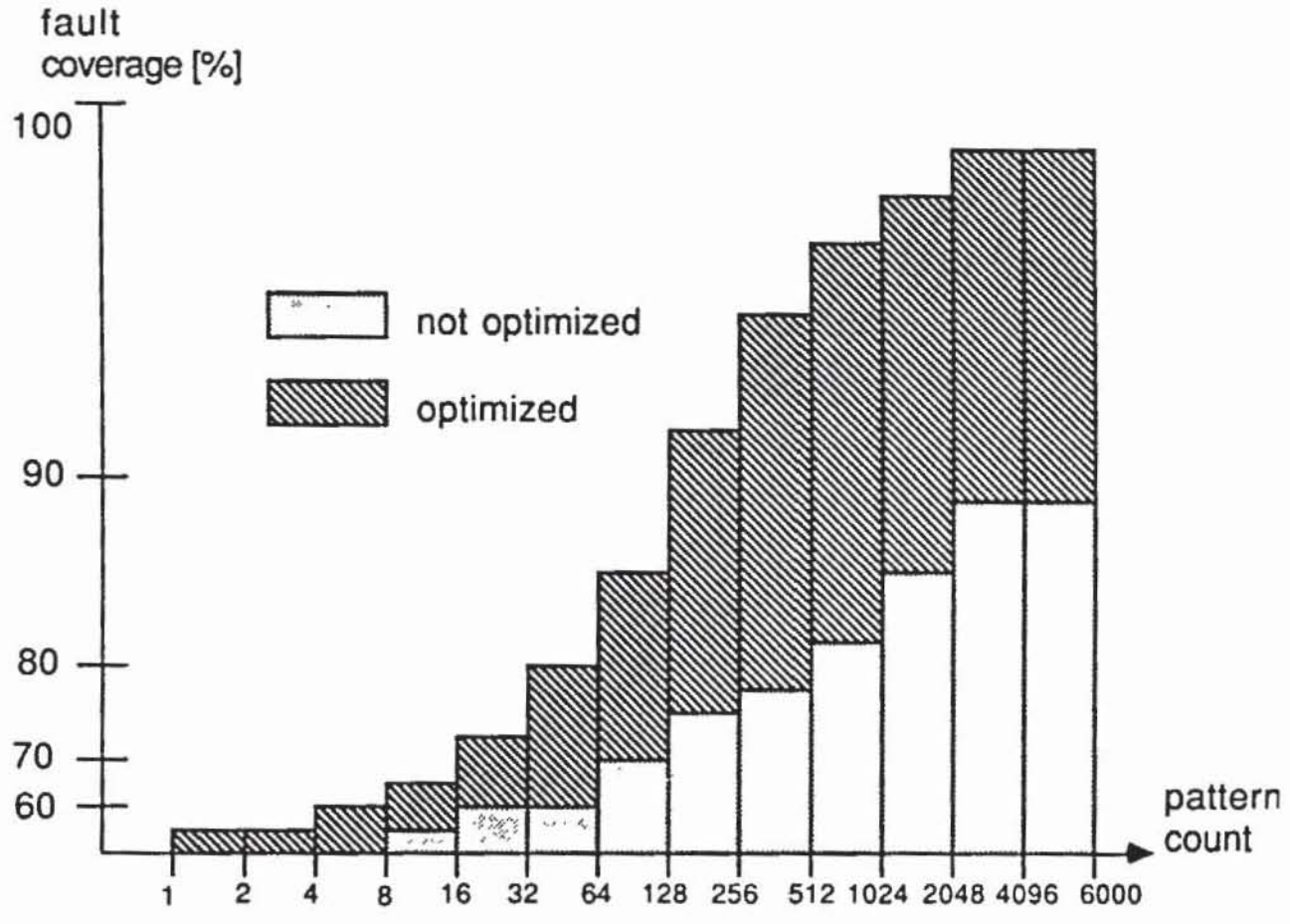
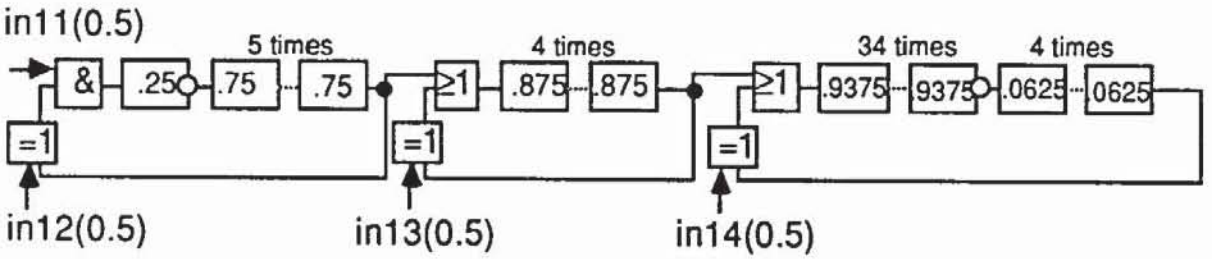


Fig. 5.2: Fault coverage vs. pattern count

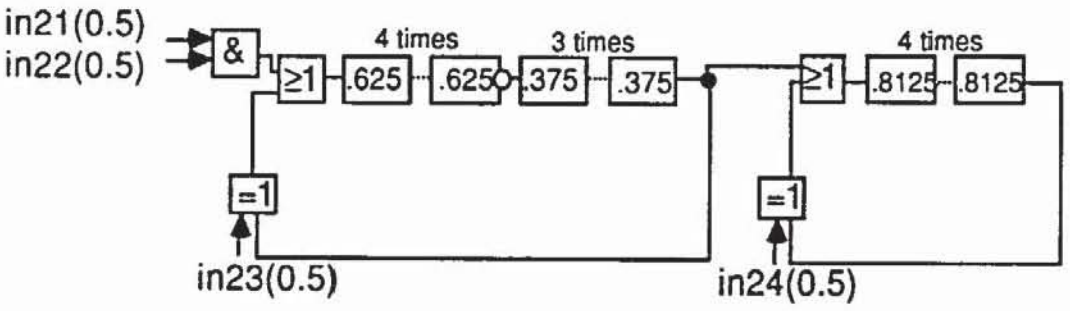
Figure 5.2 shows the results of the fault simulation with pattern sets of length 6.000. While the optimized patterns detected 98% of all the faults the not optimized patterns reached only a fault coverage of 89%.

5.3 The test pattern generator

As decribed in section 4, a feedback shift register can be configured to generate the required signal probabilities. Figure 5.3 shows the register chains of the test hardware with the numbers and values of the required probabilities. The 8 inputs in11,...,in14,in21,...in24 are originated in the BPG, which generates the necessary probabilities of 0.5.



REGISTER CHAIN 1



REGISTER CHAIN 2

Fig. 5.3: The test hardware to generate specific probabilities

6.0 Conclusions and further research

The integration of design and test into one design environment was described. The aim is a completely automatic synthesis of integrated circuits from a behavioural level specification including test hardware and generating also the test patterns. A small example showed some advantages of the applied techniques.

Further research is focused especially on the implementation of a more efficient optimization including a flexible technology dependent rule base. Expert system concepts influence not only the design but also the test in our system. We are planning an expert system that effectively supports the selection of an

appropriate test strategy, e.g. self-test or not, circuit partitioning, random or deductive test pattern generation, etc.

The development of some large applications like a Motorola 68000 processor and Warren's Prolog machine are being used to validate the system and to improve its performance.

Literatur

[Berg85] T. Bergstraesser: Entwurf eines modifizierten Zufallsmustergenerators, Studienarbeit, Universitaet Karlsruhe, Informatik IV, September 1985

[CaTr84] R. Camposano, L. Treff: STRUDEL: Eine Sprache zur Spezifikation der Struktur digitaler Schaltungen Interner Bericht Nr.7/84, Fakultaet fuer Informatik, Universitaet Karlsruhe, 1984

[CaWe84] R. Camposano, R. Weber: DSL - Eine Sprache zur Spezifikation digitaler Schaltungen Interner Bericht, Fakultaet fuer Informatik, Universitaet Karlsruhe, 1984

[CKR84] R. Camposano, A. Kunzmann, W. Rosenstiel: Automatic Data Path Synthesis from Behavioural Level Descriptions in DSL VLSI: Algorithms and Architectures, Edited by P. Bertolazzi, F. Lucio North Holland, 1985

[DaJo80] J.A. Darringer, W.H. Joyner: A New Look at Logik Synthesis, 17th Design Automation Conference, Minneapolis, Minnesota,1980

[DPST81] S.W. Director, A.C. Parker, D.P. Siewiorek, D.E. Thomas: A Design Methodology and Computer Aids for Digital VLSI Systems, IEEE Transactions on Circuits and Systems, Volume CAS-28, Number 7, July 1981

[DSST82] S.W. Director, J.P. Shen, D.P. Siewiorek, D.E. Thomas: The CMU DA/CAD Project Research Report No. CMUCAD-82-2, SRC-CMU Center for Computer-Aided Design, Carnegie-Mellon University,1982

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

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

[Kowa84] T.J. Kowalski: The VLSI Design Automation Assistant: A Knowledge-Based Expert System, Research Report No. CMUCAD-84-29, Carnegie-Mellon University, April 1984

[HeLe83] J.H. Heckmaier, D. Leisengang: Fehlererkennung mit Signaturanalyse Elektronische Rechenanlagen, 1983, Heft 3

[KMZ79] B. Koehnemann, J. Mucha, G. Zwiehoff: Built-In Test for Complex Digital Integrated Circuits, Int. Test Conference 1979,pp.37-41

[KuWu84] A. Kunzmann, H.-J. Wunderlich: Steigerung der Effizienz beim Test mit Zufallsmustern, Interner Bericht Nr. 19, Fakultaet fuer Informatik der Universitaet Karlsruhe, November 1984

[KuWu85] A. Kunzmann, H.-J. Wunderlich: Design Automation of Random Testable Circuits, Proc. ESSCIRC 1985, Toulouse

[Pasch85] R. Paschalaki: Lokale Optimierungen im DSL-Synthesesystem, Studienarbeit, Institut fuer Informatik IV, Universitaet Karlsruhe, 1985

[Rose84] W. Rosenstiel: Synthese des Datenflusses digitaler Schaltungen aus formalen Funktionsbeschreibungen, Dissertation, Fakultaet fuer Informatik, Universitaet Karlsruhe, VDI-Verlag, 1984

[TiWa84] E. Tick, D.H.D. Warren: Towards a Pipelined Prolog Processor New Generation Computing, 2, 1984

[Tris84] E. Trischler: An Integrated Design for Testability and Automatic Test Pattern Generation System: An Overview, 21th Design Automation Conference, 1984

[Ullm84] J.D. Ullmann: Computational Aspects of VLSI, Computer Science Press, 1984

[Wu84] H.-J. Wunderlich: Statistical Analysis of Combinational Networks, Interner Bericht Nr. 18, Fakultaet fuer Informatik, Universitaet Karlsruhe, August 1984

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

[Zipp83] R. Zippel: An Expert System for VLSI Design, IEEE Conference on Circuits and Systems, 1983