EXTEST - a Knowledge Based System for the Design of Testable Logic Circuits

C. Matthäus*, B. Krüger-Sprengel†, and H. T. Vierhaus
GMD/ E.I.S. Postfach 1240, D-5205 St. Augustin I

Abstract:
EXTEST is a knowledge-based program to support the design of testable logic circuits. Starting from a given prototype rule base a privileged user is able to modify the rule base via a comfortable user interface. Additionally an improved classical testability analysis also indicating logical and sequential depth at every node in the network is included.

1 Introduction
Design for testability is widely accepted as the only way of solving the VLSI testing bottleneck. Thereby structured approaches like scan path and LSSD have found wide acceptance, but still have to be supported by 'ad hoc' methods to check designs for their testability. Several systems supporting rule-based design have been introduced, but to our knowledge none has found widespread application [1,2,6]. The reason presumably is that certain rules governing design for testability are company-, project- or even designer specific. Experience showed that hence a rule base which can be tailored to the needs of specific users is an essential feature for acceptance by industrial designers.

2 Knowledge Acquisition
The classical approach towards knowledge acquisition is to have one expert available and have him put his knowledge into rules. Unfortunately most of the expert's knowledge is not explicitly available. Therefore for building EXTEST a better way had to be found. The method consists of several steps:

1. Collect rules from literature, personal and company experience.
2. Select a standard set of rules within the range of the tool to be built (i.e. accessible from gate level network descriptions).
3. Have external experts evaluate the rule base with respect to practical relevance.
4. Implement this pre-filtered rule base as a prototype set and have experts work with it and modify it for their own demands.
5. Take this rule base, remove company-specific rules and take it as a final example rule base.

Before implementation the rules obtained after step 3 were classified with respect to the method required for checking a network against this rule.

We can clearly distinguish between types of rules concerning
a. the i/o properties of logic elements, to be verified by syntax checks,
b. connections between only 2 circuit elements, to be verified by simple connectivity checks,
c. multiple interconnects, to be verified by complicated path tracing procedures,
d. controllability and observability levels, to be verified by comprehensive network analysis.

Rules concerning structures not available at gate level (i.e. structuring of VDD and VSS lines) had to be omitted as well as those concerning timing behaviour. Thus the rules concern only the structural representation at gate level.

The rule base obtained this way bears some remarkable properties: The number of rules is small (about 25), and there are few interactions between rules. On the other hand, verifying those rules requires a lot of additional 'conventional' algorithmic programming. Thus the job to be done is quite different from conventional AI applications in the field of diagnostics. AI design environments tend to have facilities for decision backtracking not required in this application, but including external purely algorithmic procedures is not common case and will have to overcome severe interface problems. Additionally the EXTEST concept would work only if the software could be distributed to external prototype users in industries and universities without restrictions. Thus the decision had to be to implement the whole system in 'C' to run on VMS and UNIX machines.

3 Rule Interface
Analysis of the initial rule set showed that rules governing wanted or unwanted input/output ports of circuit elements and allowed/ non allowed types of interconnects would make the majority. Thus the rule interface mainly concentrates on those rules.

The first objective was to develop a language representation for rules. Every rule consists of one or more 'objects' and 'conditions' controlling their interactions. We distinguish between 'types' of objects (i.e. d-latch, input-device, RAM) and 'classes' (i.e. latch, flip-flop, scanpath-element). Associated we have 'properties' such as 'storage', 'monostable', 'tristate'.

The 'condition' is either a logical one or a connectivity operator, whereby we distinguish between connections between just two elements and those concerning a series of connections (i.e. necessary to check for the completeness of a scan path).

*now with Dornier Systems GmbH, Friedrichshafen
†now with Rasterfied Computer AG, Paderborn
With some basic knowledge about these elements, a user can formulate his own rule, whereby he is guided by a rule-input interface. The resulting 'intermediate' form of the rule representation is then translated into small 'C' programs to be linked with the other parts of the rule base and the analysis procedures. Assuming that a modification of the rule base is much less frequently performed than a netlist check and by few authorized users only, the system is thus optimized to perform the network analysis at the optimum possible speed.

The rule base in the first version of EXTEST was tailored to host 50 rules, but an extension is easily possible.

4 Netlist Check

Netlist analysis is only possible if the nature of every circuit element is clear. Thus EXTEST assumes a non-storage, non-sequential element with a non-tristate output as a default. EXTEST was originally written to work on netlist files for the DISIM logic simulator. Thus logic elements described as macros within the DISIM language have the right properties associated automatically. User-defined DISIM macros are also checked for due semantics in their names (such as 'RS', 'JK', 'FF') and have properties associated. Additionally a compatible DISIM-extension was introduced, whereby every user-defined macro can be specified in terms of 'storage', 'tristate', 'scan-path-element' and 'static/dynamic'.

EXTEST starts a netlist analysis by establishing an internal tree structure of the network. Macros are recognized and get their appropriate positions. This is followed by analyzing macros for allowed or missing input/output terminals (i.e. flip-flops without clock inputs). Then the connectivity analysis starts on the macros containing only basic primitives, but no other macros.

Of all the basic rules in the prototype set a few require special netlist treatment:

- detection of non-interruptable feedback loops
- analysis of scan path completeness

Both rules can be disabled by the user, but normally every netlist analysis starts with network analysis for loop detection. This analysis is automatically disabled, if the macro is coded as an oscillator by key letters 'OS' in its name. Then, however, other oscillator-specific rules are checked for.

The network analysis is performed making use of a hierarchical organisation in the netlist whenever possible. Macros are checked first and properly evaluated, then in the final structural analysis macros are only treated via previously obtained results and their I/O terminals.

5 Testability Analysis

Some of the rules obtained during the first acquisition process concern the controlability and/or observability at I/O nodes of specific circuit elements. Additionally there are normally 'harmless' logic elements, which tend to obstruct testability of following nodes badly without being explicitly forbidden (i.e. consider the 1-controllability at the output of an input AND).

Therefore we introduced a sort of testability check, which is quite different from earlier approaches like SCOAP [4] or CAMELOT [3]. At the output node of every circuit element we analyze for a set of parameters,

a. 0- and 1-controllability,

b. sequential and logical depth (counting parameters),

c. 0- and 1-observability,

d. testability by combining a and c.

The key-parameters are 0- and 1-controllability. The controllability value of each node depends on two factors:

- The difficulty to put the output of this element on a logical value, we call it 'local controllability':

  \[ C_{\text{local}}(el,i) = \begin{cases} 
  1 & \text{for prim.} \\
  1 + \lambda \cdot \frac{x^*}{CB(i)} & \text{else} 
  \end{cases} \]

  for inputs
Combining these two factors we get the following formula for the controllability value of each node:

$$CY(e,i) = CW(e)*CY_{lca}(e,i)$$

The observability values are the highest controllability values on the easiest path from an element to a primary output.

6 Acknowledgements

This work was possible only with special support from Dr. K. Wöcken of GMD/E.I.S. and the GMD board of directors. Helpful contributions to establish an initial rule base come from Dr. W. Denner of Telefunken electronic, Heilbronn, and Dr. G. Zwiehof of Robert Bosch GmbH, Reutlingen. We also thank Prof. Dr. Karpinski, Univ. of Bonn, for supervising related diploma projects.

References

[1] M.S. Abadir
A Knowledge Based System for Designing Testable VLSI Circuits
Technical Report CRL-86-11 USC

A CAD System for Design for Testability
VLSI Design 10 / 1984

Design of Testable Logic Circuits
Addison-Wesley Publishing Company 1984

SCOAP: Sandia Controllability/Observability Analysis Program
Proc. IEEE 17th Design Automation Conf. 1980

Parametrische Analyse logischer Netzerke für den Entwurf gut testbarer Schaltungen
GMD-Studie Nr. 142 / 1988

RUBICC A Rule Based Expert System for VLSI Integrated Circuit Critique
IEEE Custom Integrated Circuits Conf. 1985

[7] C. Matthäus
EXTEST: Ein regelbasiertes System zur Unterstützung des Entwurfs
gut testbarer digitaler hochintegrierter Schaltungen
GMD-Studie Nr. 140 / 1988

[8] H.T. Vierhaus
Rule-Based Design for Testability - The EXTEND Approach
Proc. Compeuro'87