A Design-For-Testability Expert System for Silicon Compilers

R.P. van Riessen, H.G. Kerkhoff and J.M.J. Janssen

MESA Research Institute, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

This paper describes a design-for-testability expert system for the selection of the most appropriate test method for every macro within an IC. The interface with the system designer is user-friendly and together with an efficient search mechanism this expert system can be used as a framework for all types of macros. This tool will be used in a self-test compiler, which generates the layout of self-testable macros automatically. The self-test compiler can be part of a silicon compilation system and thus contribute to the integration of testability into the design process.

1. Introduction

It is generally agreed that the solution of the VLSI testing problem is to incorporate testability aspects into the design process. Since most chip designers cannot considered to be experts in testing and Design-For-Testability (DFT), this design philosophy requires new computer-aided tools for helping designers generate testable chips. Using knowledge-based systems seems to be an expedient way to solve this problem. In the past several of these systems have been proposed [1-4].

A good example of the use of expert systems to improve DFT in IC's is shown by the program TDES [3]. TDES is a knowledge-based system which requires a custom-designed circuit from a synthesis system. It finds an appropriate method to enhance the testability of the chip using the testability goals and constraints specified by the user. Finally, it indicates modifications to the original design so that it becomes a testable design. A program similar to TDES is the design tool RAMTEX [4]. This tool selects the optimal BIST procedure for an embedded RAM. The selection process is done by the evaluation of weighted parameters provided by a designer. Next, a self-testable RAM can be synthesized and the result is available in an EDIF netlist description. These expert systems show that testability knowledge can be captured in a CAD environment. However, these testability expert systems should be tools to be used by system designers and aid the designer in selecting the most appropriate test method for a certain design. The testability expert system described in this paper will provide a framework for all types of macros and has a user-friendly interface for a system designer. An efficient search mechanism is used and testability knowledge is stored in a compact manner.

In this paper, an overview will be given of the self-test compiler that will be used for the generation of self-testable circuits. This compiler can be used in a state-of-the-art silicon compiler like PIRAMID [5].

An important element in the self-test compiler is the DFT expert system. The implementation of this expert system will be described in detail. The use of the DFT-expert system will be demonstrated by means of an example.

2. A self-test compiler within a silicon compiler

2.1 Introduction

The silicon compilation process can be defined as a translation process from a higher-level description into low-level mask data. This translation process can be broken down into several steps, whereby each step can be considered to be a compiler for the lower level. During the design phase of an integrated system, testability requirements need to be taken into account. Therefore, a state-of-the-art silicon compiler has to be equipped with means to include testability in the system. The tool that will accomplish this task will be defined here as the *self-test compiler*.

The self-test compiler will be described in the structural domain of the tripartite design representation described by Gajski [6]. The micro-architectural representation of the design is used as input. The structural domain is used because this description defines how the function is going to be implemented on silicon and which types of elements are going to be used. At the micro-architectural level, the complexity of design is relatively low. For the functional modules at this level, like ALUs, multipliers, RAMs and other

1991 IEEE VLSI Test Symposium

macros, specific tests can be derived. These macro-specific tests [7] use the structure of the macro to determine the type of test to be performed and the test patterns to be generated.

Based on general testability requirements specified by the system designer, an appropriate self-test method has to be selected for every macro in the design and the on-chip hardware to carry out these macro self-tests has to be generated.

2.2 Overview of the self-test compiler

Figure 1 shows a block diagram of the self-test compiler CMOST (Compiler for Macro Oriented Self Test). The main purpose of this tool is to guide the design of self-testable IC's. Based on a structural description of the elements to be used and general testability requirements that have been specified by the system designer,

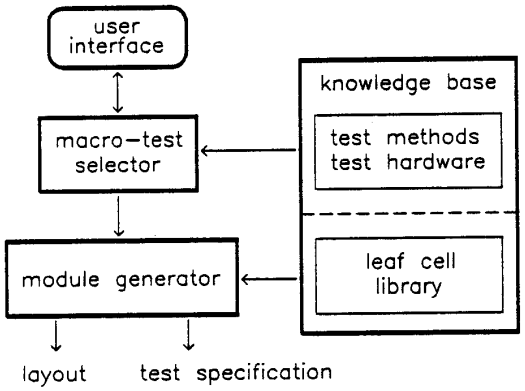

Figure 1. Block diagram of self-test compiler CMOST

CMOST will generate the layout of the most appropriate test hardware together with the original macro. The test specification will also be generated. This specification describes the initialization, control and verification of the different macro tests on the IC.

As an example, CMOST will be used in the silicon compiler PIRAMID [5]. In the target architecture of PIRAMID, three levels of hierarchy can be distinguished in order of decreasing complexity;
1) chip level, 2) processor level, and 3) EXecution Unit (EXU) level. At the processor level, the hardware for the actual signal processing (the data path) is separated from the control hardware (controller). The block diagram of this architecture is shown in figure 2. From this figure can be seen, that the data path consists of EXUs that cooperate with each other via buses. The self-test compiler operates macro-oriented, where a macro is a functional module such as a PLA, RAM and ROM. The target architecture of PIRAMID also displays a functional-block like division of the integrated circuit (EXUs). Each EXU consists of a functional core which is called the OPeration Unit (OPU), with a register file at its input and controlled by instructions from instruction-register cells. For every OPU, the self-test compiler has to select an appropriate test method.

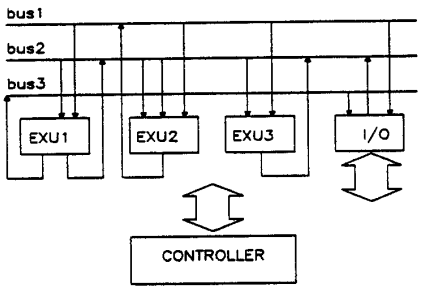

Figure 2. Target architecture of processor in PIRAMID

The self-test compiler must be constructed in such a way that new test methods can be added to the knowledge base of the compiler. Furthermore, the program should not be rigidly equipped with only a small number of test methods. Therefore, the self-test compiler must allow the testability expert to add new test methods to its knowledge base in a simple way.

When using CMOST for the design of self-testable IC's, a design-for-testability expert system will be used to select the most appropriate test method from the knowledge base.

3. A DFT expert system

In the process of designing an integrated circuit, numerous design-for-testability approaches can be applied. The selection of the most suitable one is a non-trivial problem. Based on initial requirements a search in the solution space is carried out using these requirements. If the requirements can be met, the selection terminates with success.

Otherwise, alternative solutions are examined. If there is no acceptable solution, requirement values have to be changed, and the search restarts using the new requirements. This process is repeated until a satisfactory solution is found or the designer decides to terminate the selection process.

3.1. The knowledge base

The knowledge base of the DFT expert system contains a set of test methods for every type of macro, and a set of test hardware types. A test method defines how a macro has to be tested, what type of faults can be detected, and what kind of hardware modification of the macro is required. A test-hardware type defines which

kind of additional hardware is needed to carry out a test. The combination of a test method and a test hardware type describes a Macro Test Methodology (MTM). These MTMs form the solution space of the selection process.

In literature, many test methods have been described for different macro types. For some macro types, a specific combination of test method and test hardware, and therefore a complete MTM, has been described. For other macro types, only the test method has been described. The test hardware to generate the test on chip is not described. For these test methods, dedicated hardware has to be designed, or well-known hardware for on-chip pattern generation can be used, like LFSR's.

For every MTM, a hardware description is available. This description will be used in the floorplanning phase of PIRAMID.

3.2. The Macro Test Methodology selector

Given a structural description of a macro as part of a VLSI system and a set of testability constraints, the selector part of the self-test compiler has to select an appropriate Macro Test Methodology (MTM) to be used for testing of the macro.

The set of attributes characterizing a specific MTM is called a *frame*. Examples of attributes are the fault model used, the fault coverage, the number and set of test patterns to be generated, the area overhead and the additional delay caused by the self-test hardware. Attributes can be of a logical or numerical type. They can be constants, expressions, or function calls.
The different frames (MTMs) are entered into a *selection matrix*. In this matrix the rows represent frames (MTMs) and the columns the different attributes.

Before a search for a solution in a selection matrix containing I frames (MTMs) and J attributes can be performed, the testability constraints have to be defined for each of the J attributes.

3.3 The testability constraints

When using the DFT expert system for the design of self-testable IC's, testability constraints have to be specified. In order to describe the importance of the different attributes in a frame, a weighting vector has to be defined. In order to get the ranges of the different attributes in the same order of magnitude, a Penalty-Credit Function (PCF) [3] needs to be specified for every attribute. The system designer has to specify these general testability requirements at the beginning of the silicon compilation process. The self-test compiler CMOST will derive the testability constraints for every macro in the design.

The weighting vector W describes the relative weights among the attributes:

$$W = (W_1, W_2, .., W_J) \quad (1)$$

where W_j is a numerical value representing the relative importance of the jth attribute, and $W_j > W_k$ if attribute j is more important than attribute k. Internally, the system uses normalized weights. The normalized weight for attribute j is:

$$w_j = \frac{W_j}{\sum_{k=1}^{J} W_k} \quad (2)$$

Penalty-Credit Functions (PCFs) are used to define the required value and acceptability range for every attribute. By using a PCF as shown in figure 3, different attribute ranges can also be transformed into a comparable scale.

To define this PCF, a minimum of 2 points, u_j and r_j, needs to be specified, which denote respectively the unacceptable value and the required value. With these two values, a default PCF, shown as a dashed line in figure 3, is defined.

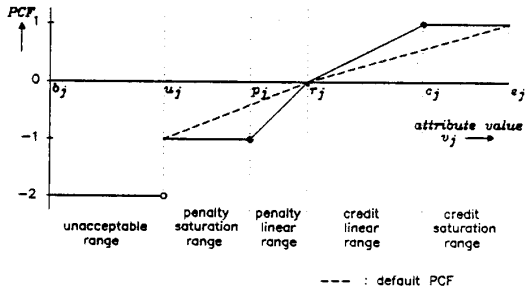

Figure 3. General form of a Penalty-Credit Function

To describe the PCF more specifically, two additional points can be specified, being p_j, the penalty value, and c_j, the credit value.
The values of b_j and e_j completely define the range of an attribute and are stored in the knowledge base. A linear function is defined by these values for the attributes of all MTMs.

The PCF value for attribute j of macro i (MTM(v_{ij})) has three possibilities:

a) it is unacceptable (-2) when $v_{ij} \in [b_j, u_j>$.
b) it receives a penalty (a value between -1 and 0) when $v_{ij} \in [u_j, r_i>$.
c) it receives a credit (a value between 0 and 1) when $v_{ij} \in [r_j, e_i]$.

As an example, the range of the attribute area-overhead is stored in the knowledge base with a value of 10^3 for b_j and a value of 0 for e_j. Depending on the application, a value has to be specified for u_j and r_j. A value for area-overhead that is unacceptable could be, for

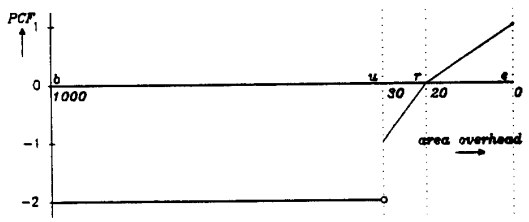

Figure 4. Example of a PCF for attribute area overhead

example, 30%. While the required value is 20%. As a result, the PCF for attribute area-overhead looks as shown in figure 4. For the MTM's in the knowledge-base, this means that the PCF value will be -2 when the area overhead is larger than 30%. The PCF value will be between -1 and 0 for an area overhead between 30% and 20%. The PCF value will be between 0 and 1 for an area overhead smaller than 20%.

Compared with the TDES system [3], a simple linear function is used that can easily be understood by a system designer. Only the required value and the unacceptable value for an attribute have to be specified. For every attribute the same type of PCF will be used. In the TDES system, different PCFs are used, which makes the selection mechanism to be more difficult to understand.

The combination of a PCF value and the weighting vector W are used to define a partial score for every attribute in the selection matrix. This partial score will be used to compare the different frames (MTMs).

The partial score for attribute j of MTM i (v_{ij}) is defined as:

$$partial\ score_{ij} = \begin{cases} -2, & v_{ij} \in [b_j, u_j> \\ PCF_j(v_{ij}) \times w_j, & v_{ij} \in [u_j, e_j] \end{cases} \quad (3)$$

In this definition, $PCF_j(v_{ij})$ is the PCF value belonging to the jth attribute. In the selection matrix, both v_{ij} and $partial\ score_{ij}$ and are stored.

The total score for MTM i is now defined as the sum of the partial scores:

$$score_i = \sum_{j=1}^{J} partial\ score_{ij} \quad (4)$$

(J = total number of attributes)

3.4 The selection process

Figure 5 shows the flow chart of the selection process. First of all, the self-test compiler will read the structural

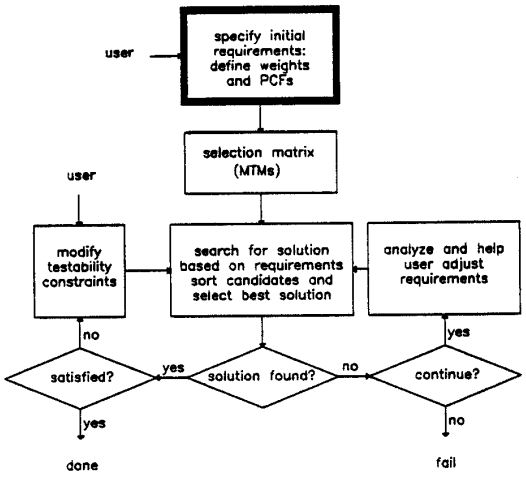

Figure 5. Flow chart of the selection process

IC description. Next, the system designer can choose the macro to be made testable, or the compiler will make this selection. Guided by the user-defined testability constraints, the partial scores can be computed and stored in the selection matrix for the macro in question. The total score for every frame (MTM) in the selection matrix is obtained from these partial scores. Using the total scores, the best frame (MTM) can be selected for this macro.

In case an acceptable solution has been found, the testable macro description will be added to the structural IC description. The above described process will be repeated until all macros have been made testable. Once all macros have been made testable, the layout of the self-testable IC can be generated together with the test specification for the complete IC.

Since user requirements concerning testability often tend to be too constraining (e.g. a 99% fault coverage with only 2% area overhead), the selection of a suitable MTM is usually not a straight-forward selection process. Using a PCF as described, has the advantage that the range of the different attributes is divided into two main parts, an unacceptable and an acceptable region. A search failure will only occur when all MTMs have at least one attribute that is unacceptable to the user. In all other cases, the attributes of the different MTMs can receive a penalty value or a credit value, but the total score of the MTM

will never be unacceptable (less than -1) because of the definition of the score for an MTM (eq. (4)).

Using this score function has the advantage that a search failure will not occur regularly as with the TDES [3] system, that only presents a solution when the total score is greater than 0. In our system, every MTM that does not have an unacceptable attribute will be presented as a solution (figure 5). When the user is satisfied with the MTM that has the highest score, the selection process is completed and a module compiler can generate the description for a complete self-testable macro. When the user is not satisfied with the solution found by the selector, he does have the possibility to start the search operation again with a modified set of testability constraints.

3.5 Search failure

A search failure only occurs when all MTMs have at least one attribute that is unacceptable to the user (partial score for attribute v_{ij} = -2). The only way to resolve a search failure is to change the testability constraints such that a solution can be obtained.
Using the Penalty-Credit Function as described before, two types of search failure can be identified, which are listed below in order of importance:

1. First class failure:
   At least one attribute (j) exists, for which the values of all MTMs (i) are unacceptable. Or mathematically,
   $$\{ \exists j, \forall i \mid PCF_j(v_{ij}) = -2 \} \quad (5)$$

2. Second class failure:
   Every MTM (i) has at least one attribute (j) value which is unacceptable. Or mathematically,
   $$\{ \forall i, \exists j \mid PCF_j(v_{ij}) = -2 \} \quad (6)$$

Although the first class failure is covered by the second class failure, it is the most critical one and needs to be solved first. Furthermore, these are the only search failures possible in the selection process. By means of an example, the selection process will be explained. In this example, search failures occur and the solutions to resolve from a failure will be demonstrated.

4. An example

As an example, the selection of a suitable test method for a PLA will be shown. The different MTM's for this type of macro are described in [8]. The frame of this MTM consists of the 7 testability attributes shown in the first column of table 1. For each of these attributes a PCF and a weight have been defined. In this example

Attribute for test	unacceptable u	required r	weight W
generation	yes	no	5
storage (bits)	30000	5000	10
appl. time (ms)	0.0005	0.0001	15
self testing	no	no	10
extra I/O pins	2	1	9
overhead (%)	30	20	20
coverage (%)	80	98	19

Table 1. Initial selector input for the example

the default PCF will be used. This means that only the values for u and r, the unacceptable value (second column in table 1) and the required value (third column in table 1) have to be specified respectively. The fourth column shows the user-defined weights, which are normalized by the system.
As an example, a fault coverage of less than 80% is unacceptable, and the required fault coverage is 98%. The user-defined weight for the attribute fault coverage is 19. This number shows the relative importance of the attribute fault coverage; this attribute is slightly less important than the attribute area overhead (W=20), but almost twice as important as the attribute test storage (W=10).
Selection of an appropriate MTM using the input shown in table 1 results in a search failure, because a first class failure occurs. The DFT expert system displays the message shown in table 2. There appears to be no MTM

No solution found
There is a first class failure; one or more requirements can not be met by any MTM. The failures are shown below:

ATTRIBUTE	REQUIREMENT	BEST POSSIBLE
test appl. time	0.0001	0.0007

If you can change the above requirements to under the best possible values, there might be a solution. If some of these requirements are unchangeable, there will be no solution.

Table 2. Example of a first class failure

in the knowledge base with a test application time smaller than 0.0005 ms. In fact, the best possible test application time is 0.0007 ms. To resolve from this failure, the requirement for test-application time is changed to 1 ms and the unacceptable value is changed to 1.5 ms.

With these modified requirements, the selection is restarted. Unfortunately, there still exists no

No solution found

There is a second class failure; every MTM has at least one attribute value which is unacceptable for you. The following table shows possible ways in which the requirements MUST be changed:

ATTRIBUTE	REQUIRED	CHANGE TO	COST
area overhead	20	31.7	0.01288
extra I/O pins	1	4	0.08182
fault coverage	98.	50.7	0.31631
test generation	no	yes	
test storage	5000	1399680	5.30182

The problem may be caused by improper definitions of the PCFs. The suggestion is that you change the requirements on:

ATTRIBUTE	FROM	TO AT LEAST
area overhead	20.	31.7

In this way you can make minimal changes and obtain a solution.

Table 3. Example of second class failures

acceptable solution, as is shown in table 3. This table shows the system display for second class failures.
The DFT expert system indicates that changing the attribute area-overhead is preferred because of the smallest costs.
When this attribute is changed as indicated, different solutions can be found. Table 4 shows that the MTM specified as 'FIST' [9] has the most positive score and can be selected by the designer. With this test method, every crosspoint in the AND array of a PLA is addressed using two shift registers. The results of a test are analyzed with a multiple input signature register which acts as a conventional latch register during normal operation.
When the designer has selected the most appropriate

SOLUTION FOUND: FIST

Use FIST because it is the only MTM that contains no unacceptable values, as can be seen from the following table:

ATTRIBUTE	REQ.	UNACC.	FIST	BIST2	..
test generation	no	no	no	no	..
test storage	5000.	30000	38.	0.	..
test appl. time	1.	1.5	0.08	0.6	..
self testing	no	no	yes	yes	..
extra I/O pins	1	2	2	4	..
area overhead	30.	40.	31.7	27.2	..
fault coverage	98.	80.	99.	100.	..
total score			0.3511	-1.4674	..

Table 4. Example of possible solutions

solution, the self-test compiler is able to generate the layout of a macro with the previously mentioned, additional hardware to perform the selected test.

5. Conclusions

This paper describes a framework for a DFT expert system to be used in a silicon compilation environment. With this system, only a minimum number of testability requirements has to be specified. Based on these general testability constraints, the self-test compiler will derive the macro-testability constraints. An expert system is used for the selection of an appropriate test method for every macro in a design. As compared with other DFT expert systems, our definition of testability requirements is straight-forward and requires a minimum of testability knowledge from a system designer. A search failure will only occur when a testability requirement is unacceptable to the designer.
Also, a distinction between test hardware and test method is used in our knowledge base. As a consequence, new Macro Test Methodologies can be entered in a very flexible way. Currently, the layout of self-testable macros can be generated together with the test specification for the complete system. At the moment, the self-test compiler is being integrated in the silicon compilation environment PIRAMID [5].

Acknowledgements

This research is supported by the Innovative Research Program (IOP) on IC technology under HTO-049/1, part testing.

References

[1] C. Bellon, Ch. Robach, G. Saucier, "An intelligent assistant for test program generation: The Supercat system", *IEEE Int. Conf. on CAD*, September 1983, pp. 32-33.

[2] H.S. Fung and S. Hirschhorn, "An Automatic DFT System for the Silc Silicon Compiler", *IEEE Design and Test of Computers*, Vol.3, No.1, Feb. 1986, pp. 45-57.

[3] X. Zhu, M.A. Breuer, "A knowledge-based system for selecting test methodologies", *IEEE Design and Test of Computers*, Vol. 5, No. 5, Oct. 1988, pp. 41-59.

[4] M. Zimmermann, M. Geilert, "Generation of embedded RAMs with built-in test using object-oriented programming", *Proc. of EDAC 89*, April 1989.

[5] J.L. van Meerbergen and H. De Man, "A true silicon compiler for the design of complex ICs for digital signal processing", *Philips Technical Review*, Vol. 44, No. 7, 1989.

[6] D.D. Gajski, R.H. Kuhn, "New VLSI tools", guest editor's introduction of *IEEE Computer*, December 1983, pp. 11-14.

[7] F.P.M. Beenker et al., "Macro testing: unifying IC and board test", *IEEE Design and Test of Computers*, Vol.4, no.6, December 1986, pp.26-32.

[8] X. Zhu, *"A Knowledge-based system for Testable Design Methodology Selection"*, PhD thesis, Dept. of EE-Systems, University of Southern California, August 1986.

[9] G.Grassl and H.J. Pfleiderer, "A function-independent self-test for large programmable logic arrays", *Integration*, pp. 71-80, 1983.