DESIGN AND IMPLEMENTATION OF A HIERARCHICAL TESTABLE ARCHITECTURE USING THE BOUNDARY SCAN STANDARD

R.P. van Riessen, H.G. Kerkhoff and A. Kloppenburg

IC-Technology and Electronics Group, University of Twente
P.O. box 217, 7500 AE Enschede, The Netherlands

Abstract

This paper describes a standardized and structured test methodology, which is based on the JTAG boundary-scan proposal. The architecture will ensure the testability of the hardware from printed-circuit board level down to integrated-circuit level. In addition, our architecture has the feature of built-in selftest at the IC level. The implementation of the architecture by means of a self-test compiler is discussed.

1. Introduction

It is a fact that the increasing complexity of digital integrated systems causes severe problems with respect to the testing of these systems.

At printed-circuit board level, these problems have stimulated the development of the boundary-scan standard [1] to improve the controllability and observability of primary inputs and outputs of integrated circuits. As a result of this standardization, the required hardware can easily be implemented by means of computer tools, thereby reducing the design time.

At integrated-circuit (IC) level, the testing problems have led to the development of several different testability approaches. An approach that seems most promising for future VLSI and ULSI circuits is built-in selftest [2], in which case the test-data generation and evaluation is on-chip. However, there is no standard approach to design built-in selftestable circuits. As a consequence, the initialization, verification and control of the self-test at the IC level cannot yet be integrated in the boundary-scan standard.

However, the integration of built-in selftest at the IC level with the boundary-scan standard at printed-circuit board level is necessary, because the boundary-scan standard defines the access to the IC during the IC test. The extra test pins will have to enable the control of the on-chip testability hardware.

This paper describes the integration of the boundary-scan standard with the built-in selftest approach. The architecture that accomplishes this integration will be referred to as the *hierarchical testable (H-testable) architecture*.

In a digital system, several levels of hierarchy can be distinguished. First, the printed-circuit board level (e.g. a Winchester control board) and second the IC level (e.g. a microprocessor chip). The third level is found within the IC, were distinctions can be made between different functional modules (e.g. PLA, ALU), the so-called macros [3]. These three different levels of hierarchy will be used to define the H-testable architecture.

This paper will describe the major requirements for an H-testable architecture. Next, the application of the boundary-scan standard in the H-testable architecture will be explained. Then, the H-testable architecture at the macro level will be described. In an example, the use of built-in selftest facilities at the macro level in an H-testable architecture will be shown. The last part of this paper will show how the H-testable architecture can be implemented by means of a self-test compiler [4]. This compiler automatically generates the layout of a macro, including data generation and evaluation self-test hardware. It will now be possible to use the built-in selftest capabilities of this macro at higher levels of hierarchy, because the H-testable architecture is used (e.g. at IC-level and printed-circuit board level).

2. Major requirements for an H-testable architecture

The most important requirements for an H-testable architecture are as follows.

First of all, the architecture has to be hierarchical. The hierarchical approach allows test results obtained at a lower level to be used in higher levels. For example, the result of a macro test can be used for the IC test.

Secondly, the architecture has to be standardized. With a set of well defined test-interface rules and control definitions for every level of hierarchy, it will be possible to use standard test approaches.

In the third place, the architecture has to be structured to reduce extra design time. With a structured approach, test hardware only has to be developed once and can be used many times. A structured approach can also facilitate the design by means of computer tools.

The final requirement is the ability of the architecture to incorporate built-in selftest facilities of different macros.

In order to be compatible with the industry standard for board testability, the H-testable architecture will be combined with the JTAG[1] boundary-scan standard [1].

[1]JTAG: Joint Test Action Group. A collaborative organization comprised of major semiconductor users in Europe and North America.

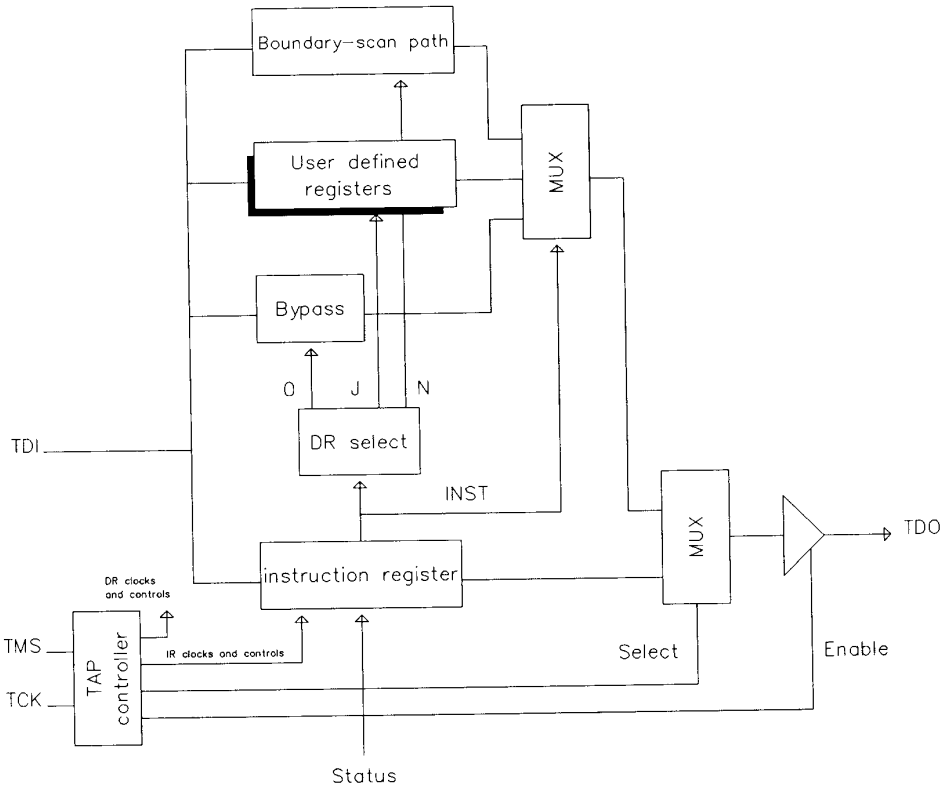

Figure 1. Architecture of the JTAG boundary-scan standard [1].

3. Integration of the boundary-scan standard with the H-testable architecture

The architecture of the boundary-scan standard for printed-circuit board testing is shown in figure 1. The behavior and architecture of all blocks in this figure are defined by the JTAG in reference [1]. This architecture will be used to define the H-testable architecture.

A schematic block diagram of the H-testable architecture at the IC level is shown in figure 2. In this diagram, two levels of hierarchy can be distinguished; the macro level and the integrated-circuit level. The *integrated-circuit level* consists of (self-)testable macros, interconnections between these macros and additional testability hardware.

The *macro level* in turn consists of a (self-)testable macro with additional testability hardware. Both levels of the testability hardware incorporate *Test Interface Elements (TIE's)*, a testprocessor and a scan path.

The TIE's separate a macro (an IC) from the connections with other macros (IC's). Therefore TIE's are located at the primary inputs and primary outputs of both macros and IC's . Each TIE contains boundary-scan cells and serial control registers. Parallel control of the TIE's is provided by the testprocessors.

The *IC testprocessor* provides the TIE's at IC level and the macro testprocessors with parallel control. The *macro testprocessor* provides the TIE's at macro level with parallel control. This macro testprocessor can also control a macro selftest.

3.1 The H-testable architecture at IC level

At the IC level the H-testable architecture will meet the boundary-scan architecture and behavior. Therefore the H-testable architecture is merged with the JTAG boundary-scan architecture. The parts of the boundary-scan architecture, shown in figure 1, are merged with the H-testable architecture in the following way.

- The JTAG boundary-scan path of figure 1 will be part of the boundary-scan cells of the TIE's at the input and output of the integrated circuit in figure 2.

- The JTAG instruction register path of figure 1 will be implemented in the IC-level testprocessor of figure 2. The registers in this path will provide the serial control data for the IC-level TIE (figure 2).

- The JTAG *Test Access Port (TAP) controller* is implemented in the IC level testprocessor of the H-testable architecture. The TAP controller generates the parallel control signals for the IC level TIE's and the macro-level testprocessors.

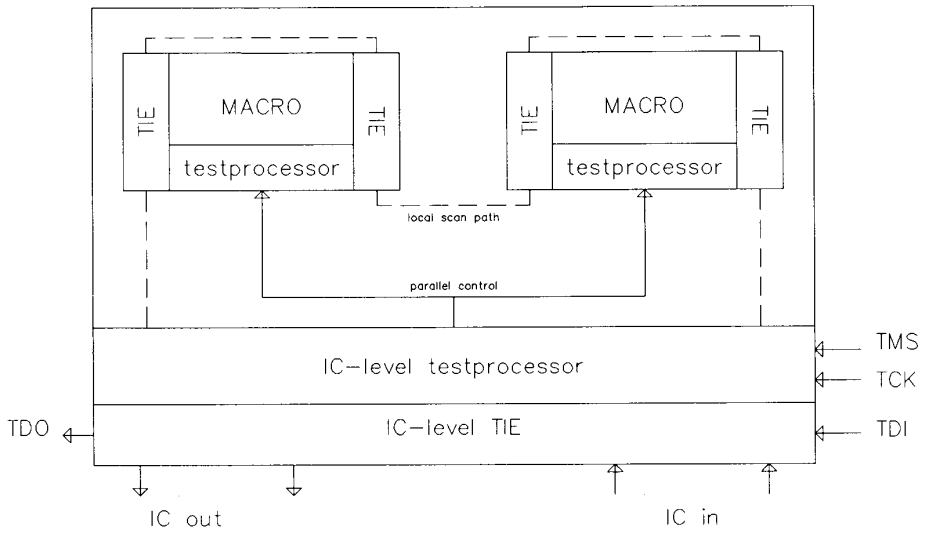

Figure 2. Block diagram of the H-testable architecture at IC level.

The JTAG user defined register path is used to implement the local scan path of figure 2. This local scan path will be used to shift in and out testdata for the macros in an IC.

The architecture of the JTAG boundary-scan standard can be merged with the H-testable architecture without any changes. This means that at the integrated-circuit level, the H-testable architecture has already been defined. The H-testable architecture at the macro level will be discussed in the next chapter.

4. The H-testable architecture at the macro level

The test hardware for the H-testable architecture at the macro level consists of Test Interface Elements (TIE's) and a macro testprocessor (see figure 2).

4.1 The architecture of the Test Interface Element (TIE)

A Test Interface Element (TIE) in the local scan path forms the link between a macro and the interconnect between macros. This element is only added to enhance the testability. The TIE's are located at both inputs and outputs of a macro and do not affect the functional behavior of the IC during normal operation.

During an IC test however, the TIE's are able to separate macros from the interconnect between other macros, which allows an independent test of both. Test patterns are shifted serially into the TIE via the local scan path and the TIE applies the patterns in parallel to the macro or to the interconnect between macros. Test results from the macro are loaded in parallel into the TIE's at the output of the macro. Test results from the interconnect between macros are loaded in parallel into the TIE's at the input of a macro. Next, data in the TIE's will be shifted out serially via the local scan path. Control signals for the TIE are applied serially via the instruction registers in the local scan path and in parallel via the control signals from the macro-level testprocessor.

The implementation of the Test Interface Element is shown in figure 3. A TIE consists of data-register cells (D), control-register cells (M and S), a by-pass path, a multiplexer and a logic gate.

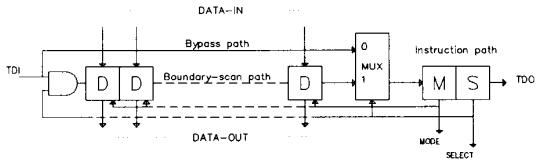

Figure 3. Implementation of a Test Interface Element (TIE).
D = data-register cell
M and S = control-register cells

The data-register cells (D) form the interface between the macro and the interconnections to other macros. Figure 4 shows a block diagram of a data-register cell. This cell consists of two multiplexers and a data latch. This data-register cell can be used in 4 different modes, controlled by the signals DRC1 and DRC2. These signals are supplied by the macro-level testprocessor. The first mode (DRC1=1, DRC2=1) is the normal functional operation. Data enters the cell via the input DATA-IN and propagates through the cell with minimal delay via DATA-OUT. The second mode (DRC1=0, DRC2=0) is the hold mode, whereby data in the latch will remain unchanged. The third mode (DRC1=0, DRC2=1) of the cell is the scan mode, where the cell is placed in the local scan path at the IC level (see

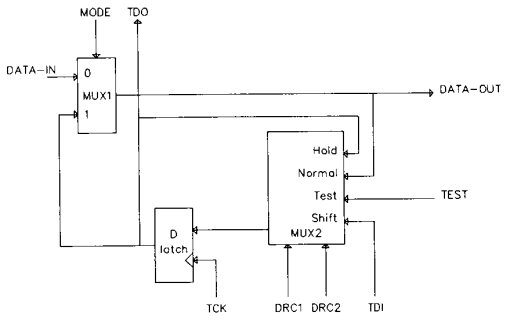

Figure 4. Block diagram of a data register cell.

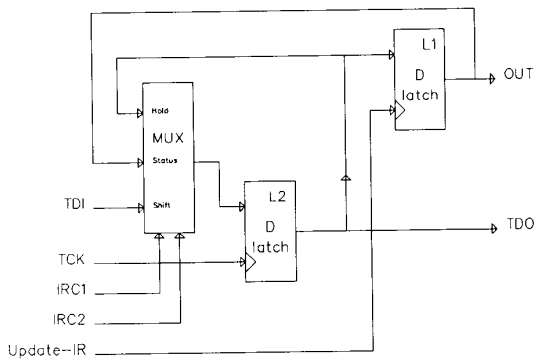

Figure 5. Block diagram of an instruction register cell.

DRC1	DRC2	selected input
0	0	Hold
0	1	Shift
1	0	Test
1	1	Normal

Truth table for the data register multiplexer MUX2

IRC1	IRC2	selected input
0	0	Hold
0	1	Shift
1	0	Status
1	1	Shift

Truth table for the instruction register multiplexer

figure 2). Data can now be shifted-in via input TDI and shifted-out via output TDO. The fourth mode (DRC1=1, DRC2=0) is the test mode. The input TEST can now be used for built-in selftest purposes. The signal MODE is supplied by one of the registers in the instruction path.

The registers in the instruction path (M and S) provide the serial control for the Test Interface Element. These registers consist of a shift register latch (L2) and a parallel output latch (L1). The block diagram of an instruction register is shown in figure 5. The control signals Update-IR, IRC1 and IRC2 are supplied by the TAP controller of the IC-level testprocessor. At the rising edge of Update-IR the contents of the shift register latch L2 is loaded into the parallel output latch L1. The signals IRC1 and IRC2 control which input is selected by the multiplexer. The input Status (IRC1=1, IRC2=0) is required to load a signal into the shift register during the state Capture-IR [1]. The input Shift (IRC1=D, IRC2=1) is the serial scan input and is selected during the state Shift-DR [1]. This input is connected to the output TDO of the previous shift register cell. The input Hold (IRC1=0, IRC2=0) is selected during all other states of the IC-level TAP controller [1].

The function of the select register S is to control the bypass of the data register cells (see figure 3). The mode register M controls the two functions of the data-register cells. In figure 4 can be seen that the data-register cells transmit data when mode=1 and receive data via input DATA-IN when mode=0. Because TIE's are present at both the input and the output of a macro, there are two mode registers (M1 at the input, M2 at the output) and two select registers (S1 at the input and S2 at the output). These four instruction registers can define 16 different modes for the data-register cells.

In contrast with the JTAG architecture, figure 3 shows that the boundary-scan path and the instruction path are connected serially. With this architecture at the macro level, a simple multiplexer can be used to select between by-pass mode and boundary-scan mode (only one control signal is required). Both modes will include the instruction path. This means that a data scan will always contain data bits *and* instruction bits. Only one scan operation is necessary to initialize the TIE's for a macro test. At the printed-circuit board level, the JTAG boundary-scan architecture requires two scan operations to initialize the TIE's. In the first stage the instruction bits are shifted in and in the second the data bits can be shifted in.

Another difference between the macro-level TIE and the IC-level TIE is the number of modes of a data-register cell. The IC-level boundary-scan data register cell has three modes of operation. The macro-level data register cell can be used in four different modes. The additional mode is the test mode. In this mode, the data-register cell can be used for built-in selftest purposes. This fourth mode does not require an extra control signal as compared with the boundary-scan register cell [1].

4.2 Architecture of the macro-level testprocessor

The testprocessor forms the control part of the H-testable architecture. At the macro level, the processor has to perform a macro selftest and apply the parallel control signals to the data-register cells of the TIE's at both the input and the output of the macro.

To carry out a built-in selftest, testpatterns have to be generated and compacted by some hardware implementation of a testpattern generation/compaction algorithm.

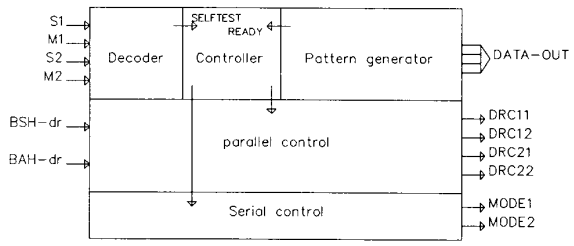

Figure 6. Block diagram of a macro-level testprocessor.

Testpatterns are applied in parallel to the macro inputs by loading the testpatterns in the data register cells via the extra TEST input (see figure 4). The test result is loaded parallel into the TIE at the output of the macro.

During the selftest, the macro testprocessor generates the parallel control signals for the data-register cells of the TIE. A block structure of the macro-level testprocessor is shown in figure 6. The following parts can be recognized:

- Parallel and serial control logic to supply the data-register cells with control signals. The signals DRC11, DRC12 and MODE1 form the signals DRC1, DRC2 and MODE for the TIE at the *input* of a macro. The signals DRC21, DRC22 and MODE2 form the signals DRC1, DRC2 and MODE for the TIE at the *output* of a macro. These signals depend on the state of the controller and on the state of the IC-level TAP controller (BSH-dr, BAH-dr [1]).

- A decoder which signals the controller to start a macro selftest. The start of a selftest is activated by the contents of the instruction registers at the input (S1 and M1) and the output (S2 and M2) of the macro.

- A controller which is in fact a synchronous state machine for the control of a macro selftest. A macro selftest, however, can be implemented in many ways, dependent on the type of macro. Therefore, a dedicated controller for each macro has to be realized. In spite of the differences, every controller has to start the selftest, indicate the end of a selftest and control the registers involved.

- A pattern generator which is governed by the controller and generates the testpatterns for the macro. The data-register cells of the TIE at the input of the macro are used as registers for the pattern generator. The generated patterns (DATA-OUT) are applied to the TEST inputs of the data registers. The pattern generator generates a signal for the controller to indicate the completion of a selftest (READY).

For a detailed description of the architecture of the testprocessor and the TIE's, the reader is referred to [5]. We will now show the implementation of the H-testable architecture by means of an example.

5. An example of the H-testable architecture

In order to illustrate the previous concepts of the H-testable architecture an example will be shown (figure 7). Intention of this example is primarily to show the integration of boundary-scan hardware with the built-in test scheme at the IC level. The example incorporates two TIE's, a macro testprocessor, one TAP controller and a simple macro.

The central part in the architecture is the selftestable macro, with four inputs and four outputs. This macro contains only combinational logic and is tested with pseudo-random patterns. The test results are compacted in a signature analyzer. Some hardware has been added to the data-register cells of the TIE's to use this data register as a building block for pseudo-random pattern generation and signature analysis (see figure 8). The pattern generator/compactor is formed by connecting a number of

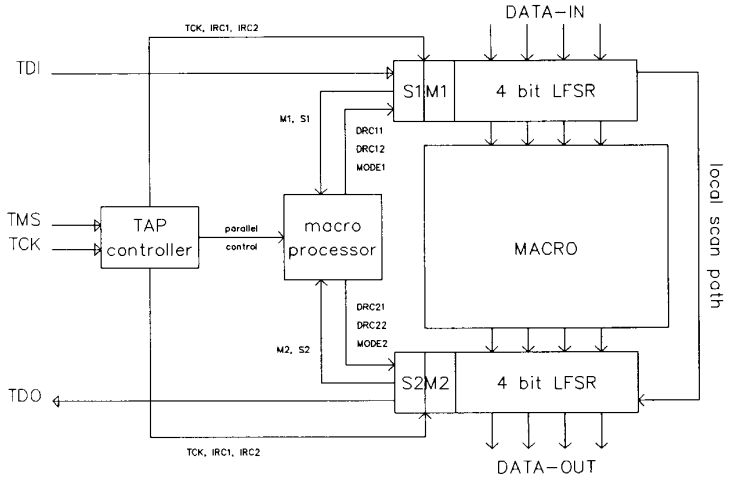

Figure 7. Example of the H-testable architecture.

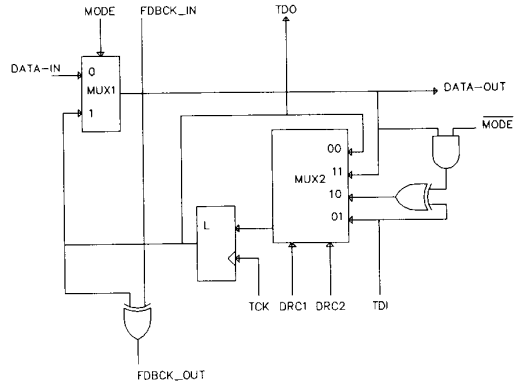

Figure 8. Block diagram of a modified data register.

modified data-register cells as a linear-feedback shift register (DATA-OUT of the last register cell is fed back to perform an exclusive-OR with the output of specific data-register cells). The specific connections are determined by the feedback polynomial [6]. The structure can be used as a pseudo-random pattern generator under the conditions (DRC1,DRC2) = (1,0) and mode = 1. The circuit operates as a signature analyzer in the case (DRC1,DRC2) = (1,0) and mode = 0.

In our example, the TIE's form a linear-feedback shift register (LFSR) during the test mode having a feedback polynomial $1+X+X^4$.
Figure 9 shows the structure of the data register part of a TIE realizing this LFSR.

The macro-level testprocessor also incorporates the necessary logic to start and complete the selftest. The signal READY, which indicates the completion of the selftest, becomes true if the testpattern (I1, I2, I3, I4) = (1,0,0,0) is generated.

The TAP controller is identical to the TAP controller as described by the JTAG in their version 2.0 of the boundary-scan standard [1].

6. Simulation of the example circuit

Simulation of an example of the H-testable architecture shows the initialization, the independent selftest and the verification. As only one macro selftest is considered, the TAP controller is assumed to start in the state Run Test [1] and the TIE's are in by-pass mode. Starting from this point of operation, the listed actions in table 1 are performed.

no.	scan action	instruction M1 S1 M2 S2
1	select initialization path	1 1 1 1
2	initialize data- & instruction registers	1 0 0 0
3	scan during selftest	1 0 0 0
4	select result path	0 1 0 1
5	verify testresult & scan-in pattern for external test	0 1 1 1

Table 1. Subsequent tests applied to the example

After scan action number 2 the macro selftest is started. Scan action number 3 shows that during the macro selftest, data can still be shifted through the TIE's of this macro.

The simulation has been carried out using a switch-level description of the example circuit. The results of the simulation are described in [5] and show the correct operation of the H-testable architecture. At this moment, a layout is being designed for the individual blocks of the H-testable architecture. The layout description of the blocks can then be used in our selftest compiler.

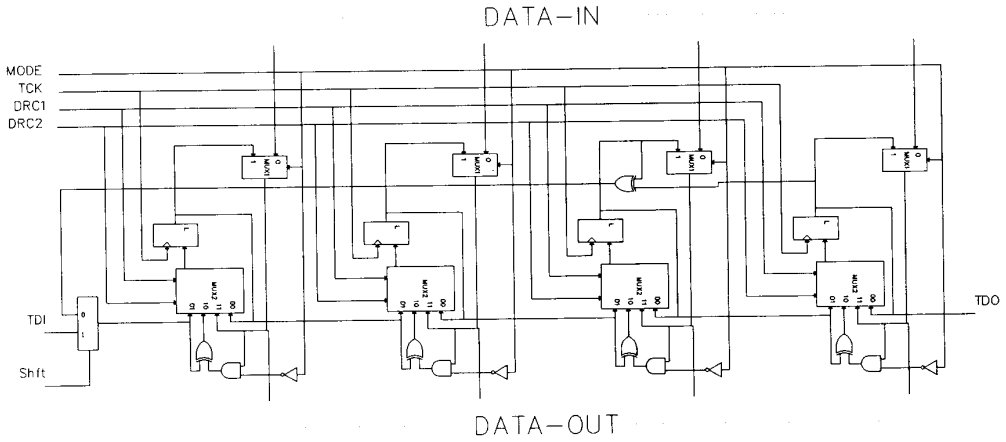

Figure 9. Block diagram of the data register part of the TIE, forming a 4 bit LFSR with feedback polynomial: $1 + X + X^4$

7. The design of a selftest compiler

The preceding example showed the operation of the H-testable architecture. The purpose of the proposed architecture is to develop a standard and hierarchical test approach to ease the burden of test development. The H-testable architecture has been implemented in a selftest compiler [4]. This self-test compiler is a software tool which automatically generates the layout of the most appropriate on-chip test hardware for self-testing along with the functional macro. In this approach the designer only has to define the type and size of the macro that has to be realized. In addition, the designer has to define the fault coverage to be achieved during self-test. Using the described architecture, it is possible to generate self-testable macros that can be controlled in a standardized format. The H-testable architecture defines the signals to initialize, control and verify a macro selftest from macro level up to printed-circuit board level.

Figure 10 shows a part of the layout of a self-testing carry-save array multiplier. This example has been generated by the selftest compiler [4]. The selftest, performed in this particular structure, is an exhaustive test. The test responses are evaluated by means of signature analysis. The bottom row of cells in the figure shows the layout of some data-register cells, used for data compaction.

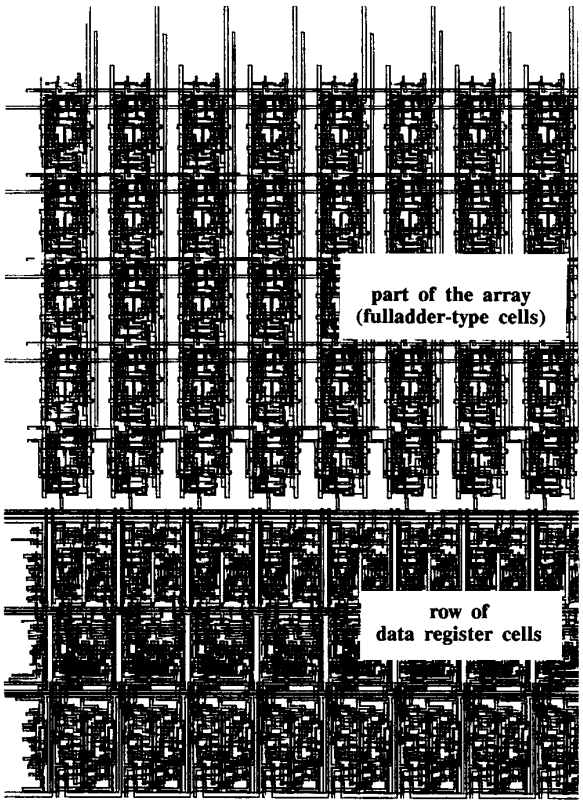

Figure 10. Part of the layout of a self-testing carry-save array multiplier.

8. Conclusions

This paper describes an architecture that will ease the problems of testing IC's on printed-circuit boards. The H-testable architecture is hierarchical, structured and compatible with the JTAG boundary-scan standard for printed-circuit board testing. Using the H-testable architecture a macro selftest can be initialized, controlled and verified from the IC level up to printed-circuit board level. During a macro selftest, the IC-level scan path can still be used, which implies that parallel testing of different macros is possible with the H-testable architecture. This architecture has been implemented in a selftest compiler. An example circuit, generated by this compiler, shows the possibilities of this architecture. The overhead of the extra test hardware still remains a problem. Reduction of the overhead is a main topic for further research. In the case of small macros (10-20 I/O ports), the controller parts of the H-testable architecture are expected to determine the overhead. Therefore it will be advisable to use only one macro testprocessor for a set of small selftestable macros.

9. Acknowledgements

This research is supported within the IOP IC-technology research program, nr. HTO-049/1, part testing. F.P.M. Beenker and his group of Philips Research Labs. at Eindhoven, are acknowledged for their contribution to this research.

References

[1] JTAG report, "A test access port and boundary scan architecture", version 2.0, draft 2, January 1988.

[2] B. Koenemann, "Built-in logic block observation techniques", Proceedings of the IEEE Test Conference, 1979, pp.37-41.

[3] 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.

[4] R.P. van Riessen, H.G. Kerkhoff and A. Kloppenburg, "Design of a compiler for the generation of self-testable macros", European Solid-State Circuits Conference 1988, Digest of technical papers, Manchester, UK, September 1988.

[5] A. Kloppenburg, "Investigations on the possibilities of using boundary-scan techniques in silicon", University of Twente, M.Sc. thesis nr.060-7001, Enschede, The Netherlands, July 1988.

[6] N. Benowitz et al., "An advanced fault isolation system for digital logic", IEEE Transactions on Computers, Vol. C-24, May 1975, pp.489-497.