Enhanced self-test techniques for VLSI systems applied to the IBM Enterprise System/9000 Type 9121 processor

by S. Sarma

This paper discusses the problems associated with obtaining adequate test coverage from random self-test for thermal conduction modules (TCMs) in the air-cooled IBM Enterprise System/9000™ Type 9121 processors. Each 9121 TCM contains approximately a quarter of a million circuits. The present complexity of the TCMs made previous testing methods such as chip-in-place (CIP) testing inviable. The solution was to apply self-test techniques to the 9121 TCMs during the manufacturing process. Analytical and simulation techniques were used to predict the random-pattern testability of the TCMs. The results of the self-test process for the five distinct 9121 processor TCMs are presented. Methods of identifying and modifying random-pattern-resistant logic structures are discussed. It is also proposed that a hybrid approach combining random self-test with deterministic test generation can be used to enhance testability.

Introduction

The design of the IBM Enterprise System/9000™ Type 9121 processor evolved from that of the IBM Enterprise System/3090™ Model S processor, which is a general-purpose multiprocessor using bipolar technology, with a uniprocessor containing 21 water-cooled thermal conduction modules (TCMs). In contrast, the 9121

uniprocessor is made up of five distinct air-cooled TCMs, all packaged within a single unit. The technology used is a mixture of bipolar and CMOS circuits. The ES/3090™ Model S processor was tested solely with deterministic and chip-in-place (CIP) testing methods. This was possible because every chip input and output signal could be probed by the tester. However, since in the case of the 9121 processor not all of the chip input and output signals can be probed, it was impossible to achieve the desired testability criteria using chip-in-place testing methods. Hence, it became apparent that new approaches to the testing problem had to be investigated for the 9121 processor design.

Built-in self-test (BIST) [1] was chosen as the testing method for the 9121 processor. One of the major advantages of using BIST is that the test patterns are generated by a pseudorandom-pattern generator (PRPG) that is packaged on the TCM itself. Hence, almost no effort is required to generate the test vectors. The intention was to use self-test as part of the manufacturing process. However, this meant that efforts had to be directed toward design for random-pattern testability. This paper first presents some background on the self-test architecture used in the 9121 processor. The self-test methodology is then discussed, and the random-pattern testability of the design is analyzed.

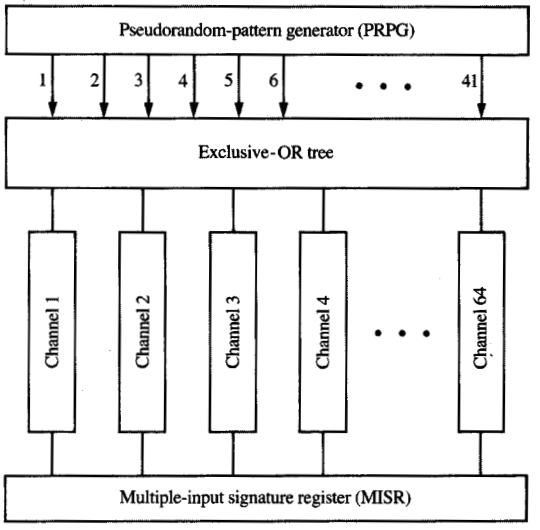

Figure 1

Overview of the STUMPS architecture.

Self-test architecture

Most of the self-test circuits reside in a single chip on the TCM. The circuits added to support the self-test function constitute approximately 3–5% of the total circuits on each TCM. The basic self-test architecture involves the use of a linear feedback shift register (LFSR) to generate the pseudorandom patterns that are used as the test vectors. Self-test utilizes a "scanning mechanism" facilitated by level-sensitive scan design (LSSD) techniques [2] to propagate the test patterns to the circuits on the TCM. Every shift register latch (SRL) on the TCM is connected in a scan ring. SRLs are used to apply the test vectors to the combinational logic blocks and also to capture the corresponding output responses. The output response of the TCM is compressed by a multiple-input signature register (MISR) located on the self-test chip. The compressed value is called a signature.

The class of self-test used in the 9121 processor is called the "STUMPS" architecture. "STUMPS" is a hierarchical acronym that stands for Self-Test Using MISR and Parallel SRSG (shift register sequence generator) [3]. An overview of the STUMPS structure is shown in **Figure 1**; its basic components are the following:

- Pseudorandom-pattern generator (PRPG). This is a maximum-length LFSR that generates pseudorandom patterns applied to the circuit under test (CUT). The PRPG used in the 9121 processor contains 41 bits; its characteristic polynomial is given by

$$f(x) = 1 + x^3 + x^{41}.$$

The logic diagram of the PRPG is shown in **Figure 2**. The PRPG cycles through $2^{41} - 1$ states when initialized with a nonzero state.

- Exclusive-OR (XOR) tree network. This network takes a combination of PRPG outputs and creates the patterns that are loaded to the CUT. This prevents the different channels from having similar output responses.
- STUMPS channels. These channels contain the SRLs in the design. All of the SRLs on a particular chip may belong to the same STUMPS channel, or they may be scattered among different channels. The amount of time necessary to test the TCM is directly proportional to the number of SRLs present in the largest STUMPS channel. The test time can be optimized by distributing an equal number of SRLs among the STUMPS channels.
- Multiple-input signature register (MISR). As mentioned previously, the MISR takes the outputs of the STUMPS channels and compresses them into a unique response called a "signature." The signature obtained for the CUT is compared against a "golden signature," which is the signature of a TCM that has been proven to be a good part. The golden signature is obtained from fault

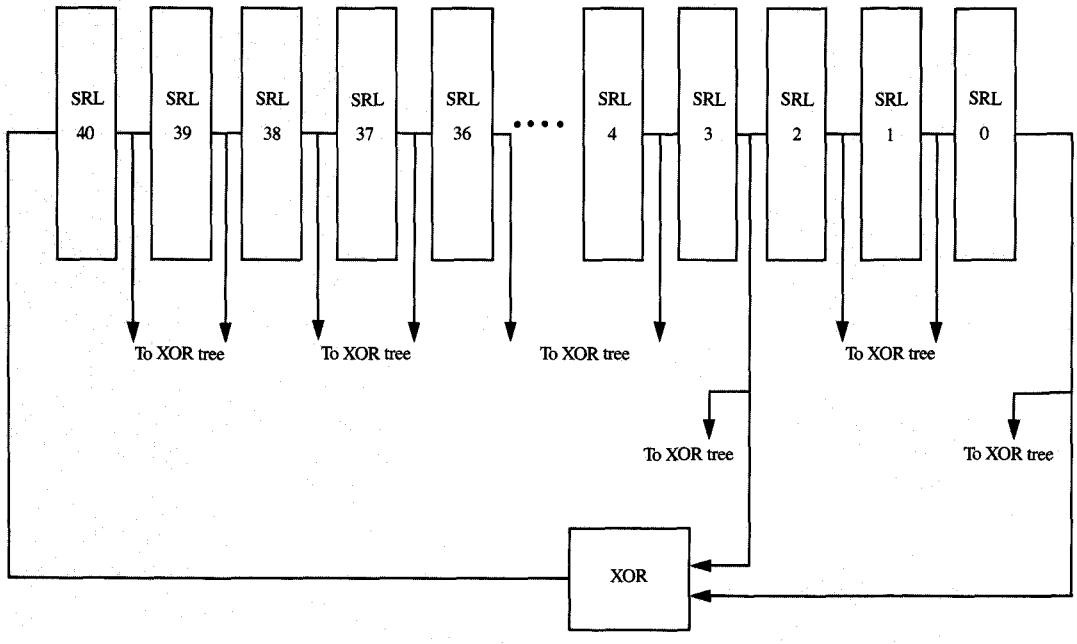

Figure 2

Logic diagram of the pseudorandom-pattern generator.

simulation. The MISR used in the 9121 processor contains 64 bits, as shown in **Figure 3**. The MISR is a maximum-length linear feedback shift register whose characteristic polynomial is given by

$$f(x) = 1 + x + x^3 + x^4 + x^{64}.$$

The ability to scan SRLs is vital during self-test, because they provide internal observation points. The organization of the SRLs on the TCM is shown in **Figure 4**. Each SRL has a scan input (SI) in addition to a data input (DI). There are four clocks connected to each SRL. the A, B, C1, and C2 clocks. The A/B clock pair is used during a scan operation. The application of the A clock causes the L1 port of the SRL to be loaded with the value present on SI. The C1 clock causes the value on DI to be latched into the L1 port. The B and C2 clocks shift the contents from the L1 port to the L2 port of the SRL. Each SRL is connected in a scan ring, and the SRLs in a particular scan ring are controlled by the same A/B clock pair. In this way, the values of all the SRLs in a ring can be observed by a scanning operation. An advantage of the STUMPS approach is that the system scan rings are broken into smaller channels during self-test, reducing the number of cycles needed to load and unload the scan rings with data. First, the SRLs are loaded with the pseudorandom patterns by pulsing the A/B clock pairs. The STUMPS channels have been initialized with pseudorandom data. Next, the system clock is applied, unloading the response from the CUT into the SRLs. This is followed by pulsing the A/B clocks so that the channel outputs are shifted into the MISR.

In addition to the above, it is necessary to isolate the TCM completely so that its expected response may be calculated. This is accomplished by adding boundary SRLs (BSRLs) to the design. During self-test, the primary inputs (PIs) of the TCM are controlled by BSRLs which can be initialized with pseudorandom patterns. Similarly, the primary outputs (POs) of the TCM are connected to output BSRLs which feed the MISR. The concept of using BSRLs to isolate the TCM completely during the self-test operation is illustrated in **Figure 5**. A multiplexer (MUX) is used to select the BSRL output rather than the TCM PI during self-test mode. In this way, the patterns feeding the

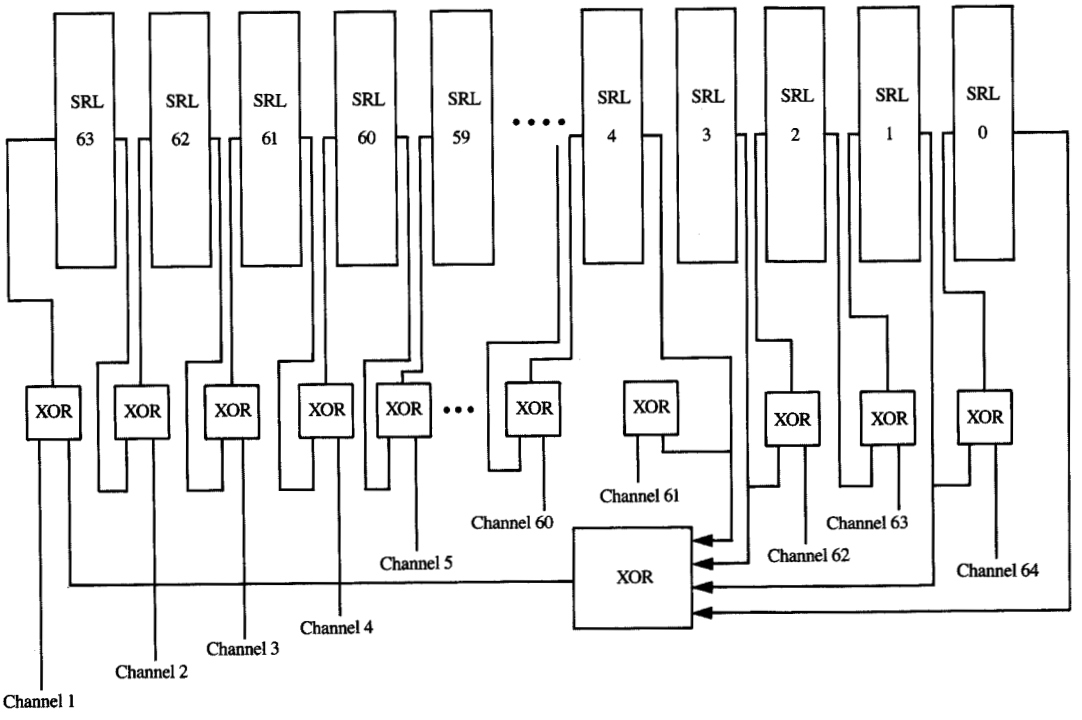

Figure 3

Logic diagram of the multiple-input signature register.

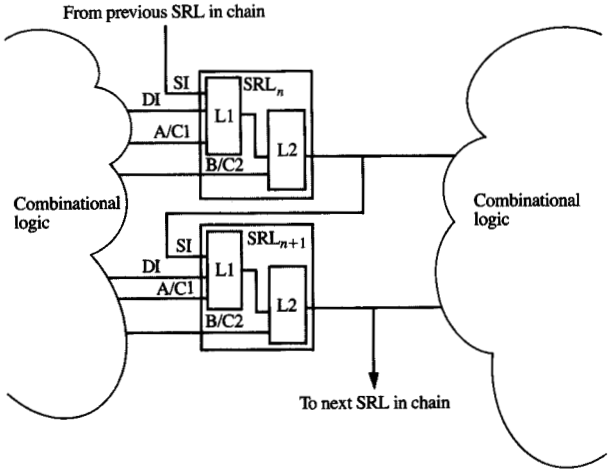

Figure 4

The organization of SRLs on the TCM.

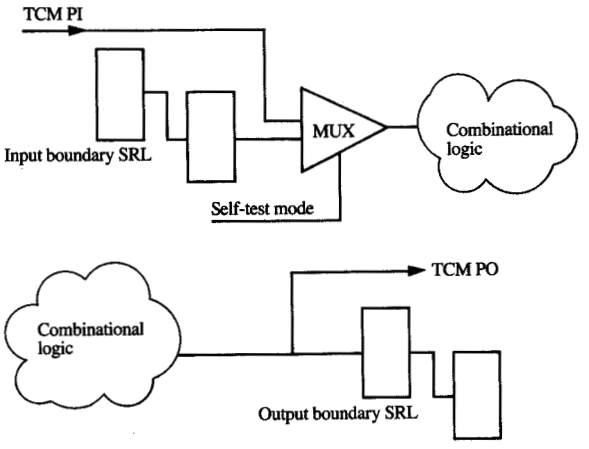

Figure 5

The use of input and output boundary SRLs on the TCM.

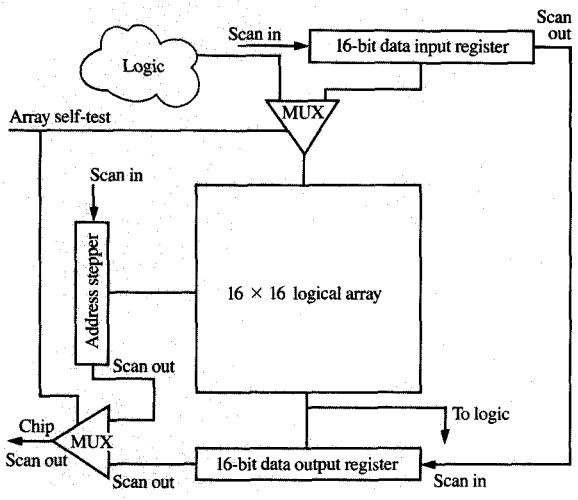

Figure 6

Logic organization during array self-test.

combinational logic on the TCM can be controlled. The values on the POs of the TCM are latched into output BSRLs and are therefore observable.

The 9121 processor contains several discrete and embedded memory arrays which feed combinational logic on the TCM. These arrays must be initialized to a known state before the self-test operation. In the case of the 9121 processor, the memory arrays are also tested using the self-test methodology, and circuitry has been added in order to support array testing. The self-test approach is used to test memory array elements because it allows faster testing. The basic concept is to initialize array cells with pseudorandom data. Following this, the arrays are read and a signature response is computed. All of the arrays on the TCM are tested in parallel. **Figure 6** shows the organization of a 16 × 16 logical array during array self-test. An address stepper is designed to increment through every array address and allow the array cells to be initialized with pseudorandom data from the PRPG. The array outputs are captured in a data-output register that feeds the MISR. The data-input and data-output registers are scannable during the array self-test operation. In contrast, the address steppers are not scannable during array self-test.

The two levels of self-test are manufacturing self-test, which is used during the manufacturing process, and system-level self-test, which is used in the field. During manufacturing self-test, the primary inputs of the TCM are exercised by a PRPG that is on the tester. The primary outputs of the TCM connect to a MISR which is also on the tester. The hardware capability for system-level self-test is available because boundary SRLs are present on each TCM. However, the present intention is to use self-test at the manufacturing level only.

Self-test methodology

The main requirement of the self-test methodology is that all LSSD design principles must be followed. In addition, self-test requires that no indeterminate states exist in the system during a self-test operation. These "*X* states" cause the signature to be unpredictable.

The phases entered during self-test are the following:

1. Array initialization. During this phase, every array on the TCM is initialized with pseudorandom data from SRLs belonging to a randomized scan ring. The array write clocks are pulsed in order to achieve this condition.
2. Array test. The arrays on the TCM are read during this phase. The array outputs go directly to SRLs which are placed in the scan rings leading to the MISR.
3. Logic self-test. This phase tests the combinational logic on the TCM. No array write clocks are pulsed. The random patterns are funneled through the CUT, and the response is compressed into a signature that is used as the "golden signature."

Array cells on the TCM feed combinational logic and consequently affect the signature of the TCM. Therefore, arrays are initialized with pseudorandom data before logic self-test is carried out.

Self-test tools

The tools that aided self-test checking and testability analysis of the design are part of the Engineering Design System (EDS). A brief description of these tools and the functions they perform is included here; they are described in greater detail in [1]. The flow diagram in **Figure 7** shows the process that is followed.

The EDS tools extract information from the chip and TCM designs to build a hierarchical, technology-independent model of the CUT. The programs also use information about the PRPG, MISR, and clock sequences to completely define the model. The major functions performed by the programs are the following:

- Design rules checking, which checks the design for adherence to LSSD and self-test rules during logic and array self-test. Some of the main requirements necessary to pass design rules checking are the following:
  1. There must be no propagation of unknown states (*X*-states) into the MISR.

Table 1 CPU resources utilized on an ES/3090 Model 400 processor for self-test checking of the cache TCM.

Job	*Total CPU time* (min)	*Total elapsed time* (min)
TCM CUT, LSSD design rules checking	17.7	70.5
Self-test checks (logic and array init/test), fault model build	997.41	1019.4
LFSR simulation	187.42	204.8
Fault simulation	1254.12	1413.6

2. Every STUMPS channel must have a PRPG as its source and a MISR as its sink.
3. The three-state drivers should not be set to the high-impedance state during the application of random patterns.
4. Fixed-value SRLs must be checked to ensure that their values cannot be altered during self-test.

- Array self-test. Any array values that affect the signature of the TCM are initialized with pseudorandom data. These programs verify the initialization of every cell. They also check to ensure that any stuck faults in array cells are detected.
- Testability analysis. These tools enable the interactive analysis of the random-pattern-resistant faults present in the CUT. This analysis is discussed in greater detail in the next section.
- Calculation of signatures. The fault coverage analysis and good-machine signature* calculations are obtained using the testability analysis for random patterns (TARP) [1] simulator. TARP generates the test pattern sequences, calculates the good-machine signature through fault simulation, determines the random-pattern testability of the design, and provides tools that allow interactive testability analysis.

The computer resources required to perform the different checks and fault simulation on an ES/3090 Model 400 processor are shown in **Table 1**. The example considered here is for the cache TCM, which has the largest logical array; therefore, the array initialization and test phase used up enormous CPU time. The second maximum occurred during fault simulation, which required 20.9 CPU hours. These data are depicted graphically in **Figure 8**.

*The good-machine signature is the signature of a part that has been qualified as "good" and is included in the test data file (TDF) that is sent to manufacturing.

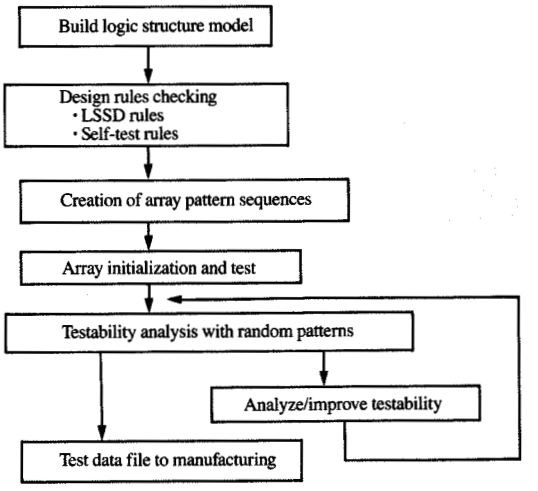

Figure 7

Flow diagram of the self-test process.

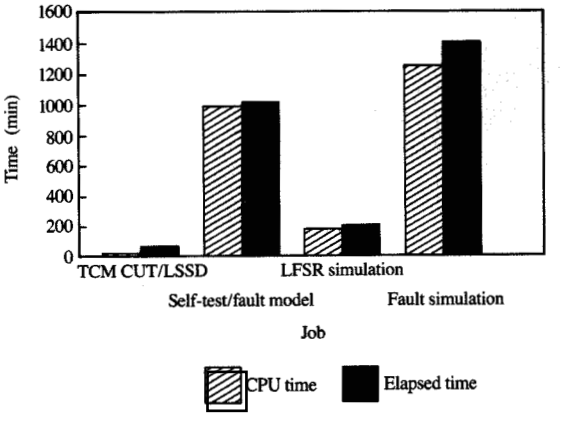

Figure 8

CPU resources utilized on an ES/3090 Model 400 processor for self-test checking of the cache TCM.

Random-pattern testability analysis

One of the most involved and time-consuming problems encountered was the random-pattern testability of the design. The design techniques employed by the ES/3090 Model S processor designers relied on deterministic test pattern generation. This meant that inherently random-pattern-resistant structures were present in the design.

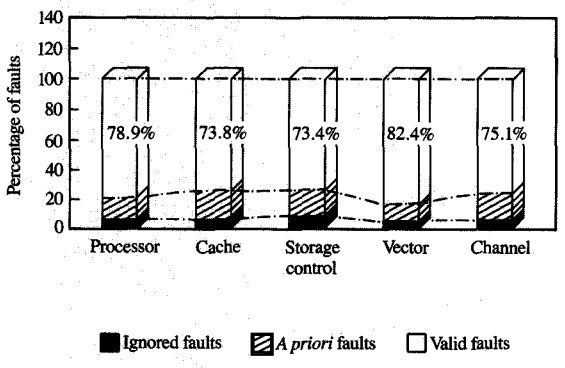

Figure 9

Fault model characteristics of the 9121 processor TCMs.

A great deal of time was spent identifying and modifying these areas of the design, thereby improving testability coverage.

In contrast to deterministic test generation, random patterns are less likely to detect certain classes of faults. Deterministic test pattern generation algorithms have a controlled way of arriving at the test vectors on the basis of the fault model. This allows deterministic test pattern generation schemes to reach the desired testability criterion with a minimal number of patterns. The disadvantage of using random-pattern testing is that it affords no pattern control; this means that a larger number of patterns must be applied to achieve the desired test coverage.

Two classes of random-pattern-resistant faults were present in the design:

1. Faults involving high-fan-in trees.
2. Faults attributed to redundancies at the local and global levels.

The first class of faults are less testable with random patterns, though deterministic patterns can detect these faults efficiently. Hence, most of the faults that were present in the 9121 processor design were due to high-fan-in trees. These faults are exposed by adding SRLs to the design. The SRLs serve to improve the controllability and observability of resistant nets. This increases the chances of obtaining the sensitizing conditions necessary to expose the fault.

The second class of faults are untestable with both deterministic and random patterns. These faults are detected by optimizing the logic. In some cases, elimination of redundancies is not possible because of performance, timing, or packaging constraints.

The testability analysis of the design was carried out on the basis of a fault model built by modeling single stuck faults in the design. Some faults are given an "*a priori* fault credit"; these faults are marked off because there is high confidence that they will be detected during the application of the test patterns. The faults that were marked off as detected included the following:

- Faults belonging to the PRPG, MISR, and any associated logic.
- Faults within the STUMPS channels.
- Faults belonging to clock logic.

The percentage of faults given an *a priori* mark-off is shown in **Table 2** for the various 9121 processor TCMs. This category of faults is a function of the number of SRLs present in the STUMPS channels. The fault model characteristics for the 9121 TCMs are shown graphically in **Figure 9**. The largest percentage of valid faults is present on the vector TCM.

Once the untestable faults have been identified, they can be analyzed with the help of the self-test fault analyzer (STFA) [1]. This tool attempts to pinpoint the exact nets in the logic that inhibit the detection of large groups of fault clusters. The STFA tool runs on the TCM model and generates a list of fault clusters in descending order. Hence, the greatest improvements in testability are achieved by fixing the faults listed at the very top of a STFA output.

- *Detailed analysis of the fault types*

In a LSSD design, the stuck faults may be classified according to the region in which the fault is observable [5].

Table 2 Fault model statistics for the Enterprise System/9000 Type 9121 TCMs.

TCM	*Total faults*	*Ignored faults*	*A priori tested faults*	*A priori tested (%)*
Processor	944 891	64 377	134 914	15.322
Cache	566 196	37 447	110 432	20.846
Storage	915 885	86 903	157 173	18.959
Vector	881 388	52 448	102 383	12.351
Channel	975 156	65 891	176 816	19.446

The three basic types of faults propagate 1) to an L1 latch, 2) to an L2 latch, and 3) to a primary output.

Most of the faults present in a LSSD design are observable at the L1 latch. The pattern sequence applied to expose these faults requires the C1 clock to be pulsed. The faults observed in the L2 latch require the application of the B clock and the C2 clock in the pattern. The pattern sequence that is used for fault simulation is as follows:

1. Load the system scan rings with pseudorandom data by pulsing the A/B clocks.
2. Apply the data to the combinational logic.
3. Apply the C1/C2 system clocks.
4. Collect the output responses in the system scan rings.
5. Unload the scan rings into the MISR.

In the following sections, the methods used by the EDS fault analysis tools to classify the various random-pattern-resistant faults described in the previous section are presented.

Faults caused by local and global redundancies
An example of reconvergent fan-out is shown in **Figure 10**. A net W in the system fans out into two branches and then reconverges at a gate G. This can produce testability problems at G. Reconvergent fan-out can cause three types of faults—*blocked*, *locally redundant*, and *multipath*. These fault types, described in greater detail below, are depicted in Figures 11, 12, and 13. The nomenclature of Figures 11–13 shows the logical values on each net in pairs. When a net has a value of A/B, A represents the value of the net in a good machine and B the value of the net in a defective machine. For the fault to be detected, opposing values must be propagated to either SRLs or POs.

Blocked faults These faults may be sensitized by random patterns, but they do not propagate to an observation point. **Figure 11** shows an example in which a test vector for a stuck-at-1 fault on the input of the AND gate may be set up. However, the fault can never be exposed because it does not propagate to a SRL. Blocked faults are made testable by adding observation SRLs to the design.

Locally redundant faults These faults cannot be tested with deterministic or random patterns. The only way to make the logic more testable is to remove the redundancy by minimizing the logic. An example of a locally redundant fault is shown in **Figure 12**.

Multipath faults These faults contain a combination of blocked and redundant faults. An example of a multipath fault is shown in **Figure 13**. The change required improves testability of the net by adding observation SRLs and minimizing the logic to remove the redundancies.

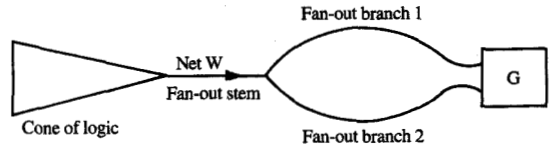

Figure 10
Two-way reconvergent fan-out.

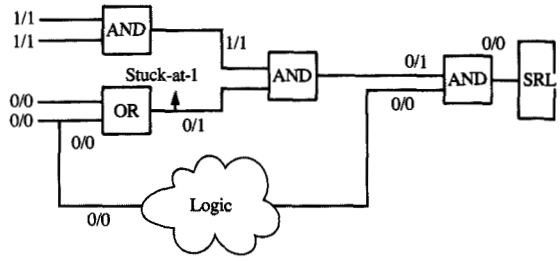

Figure 11
Example of a blocked fault.

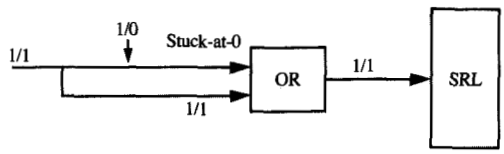

Figure 12
Example of a locally redundant fault, a simple redundant circuit where one input is stuck at 0.

Faults caused by EDS modeling techniques
These faults are caused by the nature of the model used to represent tri-state drivers in the Engineering Design System (EDS). A tri-state driver can have three states on

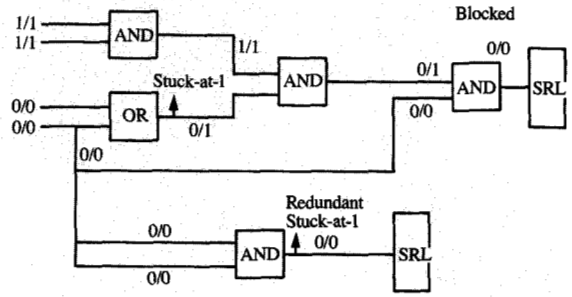

Figure 13

Example of a multipath fault, showing a combination of blocked and redundant faults.

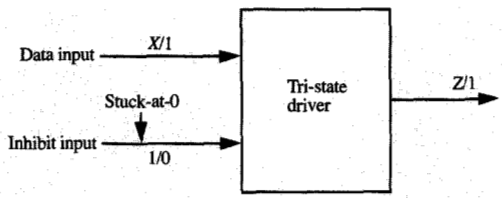

Figure 14

Example of a miscellaneous fault, a stuck-at-0 fault on the inhibit input of a tri-state driver.

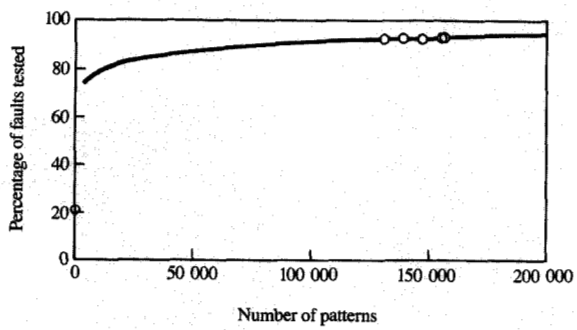

Figure 15

Fault coverage versus test pattern length using random testing.

its output: logic 0, logic 1, and high impedance. In the first two states, the tri-state driver produces a defined output value. In the high-impedance state, the driver can be pulled to a logic state by a pull-up or a pull-down resistor. Faults of this nature are classified as *miscellaneous* faults. In **Figure 14**, for example, the fault being considered is a stuck-at-0 fault on the inhibit pin of a tri-state driver. To test this fault, the inhibit pin must be forced to the active state, causing the output of the tri-state driver to go to the high-impedance state. Since this state cannot be measured, the fault is untestable. However, the inhibit input will never be forced to the active state during system operation, and these faults may be removed from the set of valid faults.

Faults caused by high-fan-in trees

The EDS fault analysis tools rate these faults by giving the equivalent AND input (EAI) associated with them. The EAI is defined as the effective fan-in of the circuit. A net is random-untestable if the probability of detecting the fault falls below a specified threshold, ε, derived from the number of test patterns used during testability analysis.

For the 9121 processor, faults with an EAI larger than 16 were considered untestable. The probability of detecting a stuck-at-1 fault at the output of a 16-input AND gate is given by

$$Prob\{detecting\ fault\}\ (\varepsilon) = P_{16} = \frac{1}{2^{16}} = 1.5259 \times 10^{-5}.$$

The number of patterns required to exhaustively test a 16-input gate is given by

$$Patterns\ needed = L_{16} = 65\,536.$$

The number of patterns used in the 9121 processor is approximately 150 000.

- *Testability data*

The testability numbers for the 9121 processor TCMs, which were obtained through fault simulation using testability analysis for random patterns (TARP), consistently exceeded 95%. The problem with random testing is the relationship between the fault coverage and the number of patterns applied (see **Figure 15**). The testability of the TCM begins at 20.8%, the percentage of faults given *a priori* credit. It can be seen that approximately 90% of the faults are tested during the application of the first 50 000 patterns, while the next 100 000 patterns detect only an additional 5% of the faults. This phenomenon is characteristic of random-pattern testing, which reaches a point of diminishing returns beyond which further application of patterns does not guarantee a proportionate increase in test coverage.

• *Proposed process improvements*

Tools support for BIST within EDS is quite efficient; however, the array initialization and test phase of design rules checking is the most expensive and time-consuming check. It is vital to break large arrays down into smaller arrays, thereby speeding up array testing and reducing test time. Another approach is to block embedded and discrete array elements from affecting combinational logic. This precludes the requirement to initialize every array cell before logic testing is carried out.

In the 9121 processor self-test methodology, the testability analysis was conducted at the TCM level after the design had passed the various checks. Many testability problems could potentially have been identified and resolved during chip self-test checking. As a process improvement, the STFA tool can now be run on individual chips as well as on a small group of chips. This allows faults embedded at the chip level to be detected and corrected earlier in the design cycle, which greatly enhances the efficiency of the self-test methodology.

In some cases, the random-pattern-resistant structures could not be modified because of packaging and performance constraints; this had a negative effect on the testability of the TCM. However, some of the hard faults could be detected if the test sequence was supplemented with deterministic patterns, and this approach has been proposed as a solution to the test coverage problem.

Conclusions

The primary problem with self-test is that the patterns are not generated in a controlled way. Some of these drawbacks can be alleviated by using the weighted random-pattern generation method described in [4]. The test quality verification of the design is important. The EDS BIST process relies on fault simulation to generate testability numbers and also to identify the hard faults in the circuit. Because fault simulation is an expensive prospect for TCMs, many of the analytical methods of predicting [5–7] random-pattern testability have not been used as part of the self-test methodology within EDS. The use of these methods would make the process more efficient.

The advantage of using self-test is that the patterns are generated with ease, and the process requires merely two to three minutes of tester time to verify each TCM. This is a considerable improvement over chip-in-place testing, which takes approximately three hours. This reduction in testing time, coupled with the improvement in random-pattern testability of the design, justifies the extra circuitry required to implement self-test. Self-test allows the 9121 TCMs to be tested extremely efficiently; however, only when testability numbers approach 100% as a result of process improvements and designer education can we begin to take full advantage of self-test.

Acknowledgments

There are many individuals to thank for the achievement of implementing BIST in the 9121 processor family. They include the logic and systems designers (IBM Kingston), the tools and test methodology developers (IBM Endicott), and the test engineers (IBM Kingston and IBM Poughkeepsie). I would especially like to recognize Thomas Dick and John Phan for their technical contributions. I am grateful to Robert Adams, Robert Herzl, and Arnold Tran for their critique of this paper. Finally, much appreciation is due Paul Bardell for the technical discussions and insightful suggestions.

References

1. B. L. Keller and T. J. Snethen, "Built-In Self-Test Support in the IBM Engineering Design System," *IBM J. Res. Develop.* **34,** 406–415 (1990).
2. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference*, New Orleans, June 1977, pp. 462–468.
3. P. H. Bardell and W. H. McAnney, "Self-Testing of Multichip Logic Modules," *Proceedings of the 1982 IEEE International Test Conference*, November 1982, pp. 200–204.
4. J. A. Waicukauski, E. Lindbloom, E. B. Eichelberger, and O. P. Forlenza, "A Method for Generating Weighted Random Test Patterns," *IBM J. Res. Develop.* **33,** 149–161 (1989).
5. J. Savir, G. S. Ditlow, and P. H. Bardell, "Random Pattern Testability," *IEEE Trans. Computers* **C-33,** 79–89 (1984).
6. J. Savir, "Improved Cutting Algorithm," *IBM J. Res. Develop.* **34,** 381–388 (1990).
7. K. P. Parker and E. J. McCluskey, "Probabilistic Treatment of General Combinational Networks," *IEEE Trans. Computers* **C-24,** 668–670 (1975).

Received August 25, 1990

Sudha Sarma *IBM Data Systems Division, Neighborhood Road, Kingston, New York 12401.* Ms. Sarma is currently a Senior Associate Engineer working in the cache design area of advanced technology systems. She received a B.A. degree in physics and a B.S. degree in electrical engineering in 1986 from Pennsylvania State University, and an M.S. degree in computer engineering from Syracuse University in 1988. Since joining IBM, Ms. Sarma has been involved in various design assignments that resulted in architectural enhancements on the ES/3090 Model S and ES/9000™ processors. She worked on defining the hardware error detection, fault isolation, and recovery techniques used on the ES/9000 processors. Ms. Sarma's primary interests are computer architecture, especially massively parallel computer architectures and fault-tolerant networks. She is a member of the Institute of Electrical and Electronics Engineers.