

LFSR-Coded Test Patterns for Scan Designs

by Dr. Bernd Könemann
c/o IBM Corp., B56/901
P.O. Box 390, Poughkeepsie, NY 12533, USA

ABSTRACT

The immense test data volumes associated with Stored Pattern testing of very large Scan Design structures make this approach increasingly unattractive for both, manufacturing and system-level Self-Test. As a result, methods using more compact data representations are gaining in popularity. Pseudo-Random Pattern methods need very little explicit data storage and minimal hardware overhead, but suffer from limited test coverage and are primarily used at the (sub-)system Self-Test level. By contrast, the high test coverage possible with Weighted Random Patterns (WRP) at the expense of some modest test data volume and more complex pattern generation hardware make WRP more suitable for chip level testing with external test equipment. Because of the more complex hardware, WRP is not easily transferrable to the system Self-Test environment.

In this paper we discuss an alternative method for compact test data storage which achieves full test coverage, but is more compatible with system level Self-Test than WRP is. Intelligent, constructive Re-Seeding of a linear Pseudo Random Pattern Generator (PRPG) is used to manipulate the generated patterns as required for full fault testing. An algorithm for calculating the PRPG Seeds from test generation results is described together with estimates of the required data storage volume, number of test patterns, test application time, and on-product hardware overhead for a state-of-the-art CMOS chip example. The results indicate that the proposed method can be more efficient than WRP.

1. INTRODUCTION

The concept of "Scan" as a method for providing access to internal test points in electronic designs, digital and analog, probably dates back to almost the beginning of the history of electronic computers. Some early implementations used addressing schemes (decoding/multiplexing) to serially access a multiplicity of test points for maintenance as well as bring up and system operator functions (for example, [1]). A comprehensive description of the role of Scan within a computer maintenance methodology is given in [2]. The first publication of re-configuring internal register elements into serial shift registers to implement the Scan function appears to be found in [3]. Today, the term Scan is almost exclusively used in the context of such shift-register based approaches, the most publicized version being the so called Level Sensitive Scan Design (LSSD) technique [4]. Scan designs reduce the test generation problem essentially to finding tests for the combinational logic sandwiched between the scannable register elements. Automatic test generation algorithms for combinational logic have been developed as early as the 1950's [5], and have been refined to a point where virtually 100% of all testable stuck-faults can be tested with automatically generated patterns.

Modern test generation/fault simulation algorithms for Scan designs are capable of handling structures with hundreds of thousands of logic gates and beyond.

While eminently successful in terms of test coverage, the resulting tests tend to produce extremely large test data volumes, if represented in the traditional Stored Pattern format. The Stored Pattern approach requires that the stimulus and expected response data for each test pattern bit are explicitly stored. For Scan designs, each test pattern involves stimulus data for all Primary Inputs (PIs) and all scannable Shift Register Elements, and test response data for all Shift Register Elements and all Primary Outputs (POs). In very large structures, the number of Shift Register Elements exceeds the number of PIs and POs by far, and the overall data volume is dominated by the data required for the Shift Register Elements. The number of Shift Register Elements tends to grow linearly with the overall circuit complexity, and the number of test patterns grows also, albeit less than linearly. As a result, the overall data volume, which is proportional to the product of the two numbers, grows more than linearly with circuit size.

Reference [6] contains an interesting trend analysis of the Stored Pattern test requirements for state-of-the-art CMOS chips designed according to the LSSD

technique. It is pointed out that the attendant data volumes stress the limits of existing test equipment and may call for re-loading the tester buffers during testing. Such re-loading severely limits the throughput of the expensive test equipment and would result in intolerable costs. Because of this, it is argued that test methods which do not require the explicit storage of all test stimulus and response data bits promise lower test cost by eliminating the buffer re-load problem. The increasingly stringent demands for chip quality dictate that the strived for reduction in test data volume must not lead to reduced test coverage. The authors compare several alternatives and come to the conclusion that WRP testing promises the best overall trade-off. In the following, we will review some of these findings and will use the same CMOS chip example as the basis for estimating and comparing the properties of a new method.

2. BACKGROUND

Table 1 summarizes the characteristics of the CMOS chip used in reference [6] to compare the costs associated with several different test methods. A representative number of stuck-faults for a chip of that size has been added. Note that the scannable Shift Register Elements in LSSD are called Shift Register Latches (SRLs).

Number of Circuits	300K
Number of I/Os	512
Number of LSSD Shift Register Latches (SRLs)	16K
Number of Stuck-Faults	1.5M
Number LSSD Test Patterns (100% Coverage)	27K
Percentage of Test Patterns Between 90% and 100% Coverage	70%
Number of WRP Weight Sets for 100% Coverage	200

Table 1: Summary of Statistics for CMOS Chip Example

These numbers can be used to derive estimates of the test data storage volume associated with different test methods.

The storage volume for Stored Pattern testing is dominated by the test stimulus and response data loaded into the on-chip SRLs in each test pattern. To allow for a fair comparison, we assume that the same design constraints are imposed on the chip for all methods.

Since WRP and Pseudo Random Pattern testing use Signature Analysis, no unknown states can be allowed in the chip. Consequently, the test response data captured in the SRLs are strictly binary (logic 0 or logic 1). The test stimulus data in the SRLs are likewise binary, since each SRL can be loaded with either a logic 0 or a logic 1. Under these assumptions, a Stored Pattern tester for LSSD testing can use a binary data representation in the scan data buffer, requiring 2 bits (1 bit for stimulus, 1 bit for response) per SRL per test pattern. The total data volume for the chip with 16K SRLs and 27K test patterns then multiplies out to 108 MBytes. This number is much smaller than the corresponding number quoted in reference [6]. The reason for the difference is that in reference [6] a multi-bit test data storage architecture optimized for functional (Non-Scan) testing is assumed. The multi-bit architecture is very wasteful for Scan data storage and in our opinion should not be used in the Scan data buffer of an LSSD tester. Yet, even with the more optimized architecture, 108 MBytes of test data are needed for the CMOS chip. In an environment where many different chip part types are manufactured and tested, the associated cumulative data volume becomes very difficult to manage and distribute efficiently. Migrating the test data into a card-level Self-Test environment becomes even more difficult. A printed wiring card with 10 CMOS chips, for instance, would require test data in excess of 1 GigaByte. The cost associated with storing and the bandwidth required for applying such data volumes within practical time limits are clearly prohibitive. Signature Analysis can be used to eliminate the need for explicit storage of the test response data. Hence, with Signature Analysis the data volume could be reduced to 54 MBytes for the CMOS chip example. Any further reduction requires to also cut down the storage associated with the test stimulus data.

Pseudo-Random Pattern testing with Signature Analysis has gained a significant foothold in card through system-level Self-Test. This method only requires a miniscule amount of explicit data storage (for pattern generator Seeds and for Signatures). The pattern generation/Signature Analysis hardware can be implemented within the product under test at very little cost overhead. The major drawback of this approach is limited test coverage. Initial experience with Pseudo Random Pattern testing of very large LSSD designs indicates that test coverages between 90% and 95% can be expected after applying on the order of 100K to 200K test patterns (the results published in reference [7] are more representative for very large structures than the results quoted in reference [6] which were collected from relatively small chips). Adding more patterns results in some, but generally limited, coverage gain.

Even after applying 1 million patterns or more, the coverage rarely exceeds 97%. These limits are caused by so called Random Pattern resistant logic structures (e.g., large fan-in) in the circuit under test. To make the Pseudo Random patterns more effective requires to re-design the resistant portions of the logic. Raising the test coverage from 95% to 100% for the CMOS chip example, would involve analyzing 75K untested faults and finding the appropriate logic changes. Assuming that each modification resolves 100 untested faults, a total of 750 logic modifications would be needed. Each modification has implications to timing, layout, and other chip characteristics, making this approach extremely resource consuming. Even if the modification process is automated, its results are often not palatable because of the delays added to critical paths when inserting test points. As a result, Pseudo Random Pattern testing by itself must be considered inadequate for chip testing, where 99.9%+ coverage is needed to assure product quality [6]. A supplement and/or alternative to logic modification is to generate and add Stored Pattern tests for the remaining untested faults after Pseudo Random Pattern testing ([6],[8]). Practical experience with LSSD structures indicates that the supplementary patterns could require as much as 70% of the full Stored Pattern tests, that is 19K patterns for the example chip. Since Signature Analysis is used, only the test stimulus data need to be stored explicitly. Yet, the resulting expected data volume for the example chip could still run as high as 38 MBytes.

A much more significant storage reduction can be achieved by WRP testing with Signature Analysis. The WRP system described in [9] and referred to in [6] uses 4 bits of Weight information per SRL for each one of multiple Weight Sets. The Weight values are constructively derived from test generation, achieving essentially the same test coverage as Stored Pattern testing. For the CMOS chip example, 200 Weight Sets are expected to be required for 99.9%+ test coverage. The total storage for the 200 Weight Sets is 1.6 MBytes, almost 2 orders of magnitude less than needed for full Stored Pattern testing. On the other hand, WRP testing requires at least 10 times, maybe 25 times, as many test patterns to be applied than in full Stored Pattern testing. That is, 270K or more WRP test patterns are to be anticipated for the CMOS chip example. The authors of reference [6] stipulate that applying the WRP patterns from automatic chip test equipment is not a problem.

However, despite the favorably low data storage volumes it is not easy to migrate WRP testing into the System Level Self-Test environment. The data transfer bandwidths of the service processor interfaces used to

initiate and operate the Self-Test functions are typically very limited (e.g., serial bus interface). All high-speed, high-volume operations must, thus, be performed in the logic under test, such that the service processor only has to download the Weight Set data (1.6 MBytes). This implies that the logic under test contains a WRP pattern generator and at least enough storage to hold a single Weight Set. The Weight storage for the example CMOS chip would have to hold at least 64 Kbits to accommodate the 4 bits per SRL for one Weight Set. For overhead calculations we assume that roughly 30% of the chip logic are contained in the 16K SRLs, and that 20 bits of RAM storage could be provided in the space of 1 SRL. Then, 1% of the chip area translates into roughly 500 SRLs or 10 Kbits of array storage. The WRP pattern generator itself requires 4 SRLs per LSSD Scan Channel (a Channel is an LSSD Scan string fed by a Pseudo Random Pattern Generator [8]). To keep the test times in bound, the Channels should be kept as short as possible; for example, 32 Channels with roughly 500 SRLs each. The pattern generator then has to be 128 bits wide. A 32 bit wide Signature Register is also needed to service the 32 Channels. Together with clocking and support logic this adds up to an equivalent of about 200 to 250 SRLs plus the 64 Kbits of array storage for a total of roughly 7% of the chip area.

Many designers, after having already paid a price for implementing Scan design and Boundary Scan to support system level Self-Test, are unwilling to pay more than an additional 1% in chip area for better test coverage. Consequently, WRP has had little success in the Built-In Test arena. The alternative proposed and described in the following promises to be as effective as WRP in terms of data compression, but requires substantially less on-product support hardware for system level Self-Test applications. The proposed method utilizes the "programmability" of Pseudo Random Pattern Generator (PRPG) hardware. First, 100K - 200K Pseudo Random Patterns are applied without intermediate Re-Seeding of the PRPG to achieve an initial test coverage of 90% - 95%. Then, intelligently pre-calculated Seeds will be loaded into the PRPG between test patterns to influence bit values in the generated test patterns for better fault detection.

3. INTELLIGENT LFSR RE-SEEDING

In each test pattern, the PRPG is cycled in a fixed clocking sequence to fill the on-product Channels with Pseudo Random values. This process is called Channel Load operation. The initial values in the PRPG register bits at the beginning of the Channel Load operation are the Seed values for this Channel Load. Cycling the

PRPG performs a deterministic operation on the PRPG register values such that the bit value loaded into each Channel SRL is a predictable function of the Seed values. It should, therefore, be possible to manipulate individual test pattern bit values in the Channel SRLs by appropriately selecting the Seed values. The problem is to find a mathematically tractable relation between the Channel SRL values and the Seed values. It turns out that under certain linearity conditions, this relationship is very simple and can, indeed, be utilized in a test generation environment.

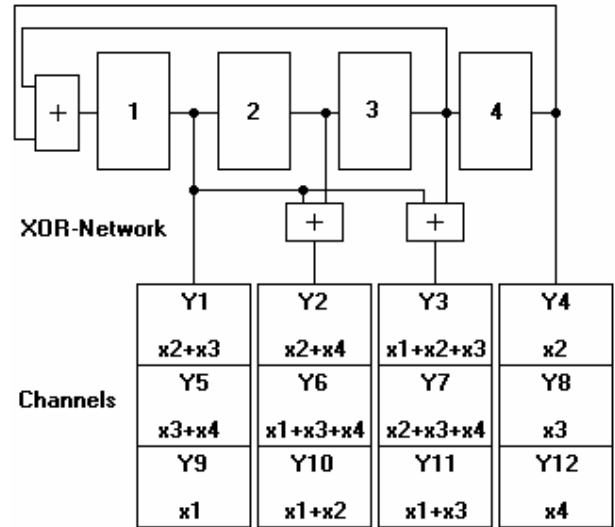


Figure 1: Channel Load Example

3.1 Linear PRPG Operation

The crucial desired property is that of linearity in the PRPG function. Practically this means that the PRPG can only use Exclusive-ORs (XORs) to combine bit values. This requirement is, for example, met by PRPGs based on Linear Feedback Shift Registers (LFSRs) and XOR-Networks to remove structural dependencies as described in [8]. Under these conditions, each test pattern bit produced in a Channel Load operation fills each Channel SRL with an XOR-Sum of Seed bits. The first step in the algorithm for intelligent Re-Seeding is to perform a symbolic simulation of the Channel Load operation to determine the particular XOR-Sum that ends up in each of the Channel SRLs. This symbolic simulation needs only be done once for each defined type of Channel Load operation.

Figure 1 illustrates the results of a symbolic Channel Load simulation with a simple example of a 4-bit LFSR feeding into 4 Channels of 3 SRLs each through an XOR-Network (the XOR function is denoted by a + sign).

The LFSR register bits 1 - 4 are initially loaded with the Seed values x_1, x_2, x_3, x_4 . The Load is performed by applying 3 clock cycles to the LFSR and the Channel SRLs (Y1 - Y12). Figure 1 shows the results after the Load operation. For example, Channel SRL Y1 is loaded with the value x_2+x_3 , where + denotes the XOR function.

In more general terms, any linear pattern generator (with respect to the XOR function) the individual Channel SRLs with a predictable XOR-Sum of Seed bit values. Let $x_1 - x_n$ be the Seed bit values loaded into an n -bit linear PRPG, and let $Y_1 - Y_N$ be the Channel SRL values. Then, it is possible to express each Channel SRL value Y_k in the form

$$(1) \quad Y_k = F_k + \sum_{j=1}^n (A_{i;k} \text{ AND } x_i),$$

where summation stands for the XOR function. $A_{i;k}$ is an n -bit vector indicating which Seed bit x_i appears in the XOR-Sum for Y_k . $A_{i;k}=1$ if x_i is included, $A_{i;k}=0$ otherwise. F_k indicates if the sum needs to be inverted ($F_k=1$) or not ($F_k=0$). All inversions inside the sum are carried out into F_k such that no inversions remain to be encoded into the coefficients $A_{i;k}$.

3.2 Calculating Seed Values from Test Generation Results

Expression 1 shows how the individual Channel SRL values are related to the Seed bit values. To use this

relationship for test purposes requires to develop a method for determining the Seed bit values intelligently for fault testing. How this can be accomplished is best illustrated by a small example. Assume that the Channel SRLs Y5 and Y8 from Figure 1 feed into an AND gate, and that the stuck-at-1 fault on the input from Y8 has not been tested yet (see Figure 2):

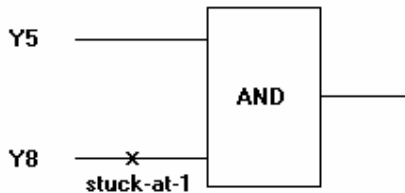


Figure 2: Untested Fault Example

Test generation for this untested fault will request the values Y5=1 and Y8=0. The values in the other Channel SRLs do not contribute to the test; they are "don't care" values. Only the values Y5 and Y8 are important; they are the "care" values for this particular test. The objective is to find Seed bit values that accomplish the requested "care" bit values. From Figure 1 we see that the following two equations must be met:

$$(2) \quad \begin{aligned} 1 &= Y8 = x3+x4 \\ 0 &= Y5 = x3. \end{aligned}$$

Any combination of Seed bit values x1,x2,x3,x4 that solves these two equations (e.g., x1=x2=x3=0, x4=1) will result in loading the proper "care" bit values into the Channel SRLs. The values ending up in the "don't care" bit positions are of no concern for the test at hand. Different solutions of the equations will lead to different Pseudo Random values in the "don't care" bit positions, but always produce the correct "care" bit values. The "don't care" bit values in the proposed method are generated as a byproduct of the Channel Load operation without having to waste explicit storage. The calculated Seed values are only tuned for the "care" bit values. Thus, in the example, 4 Seed bit values are sufficient to describe a test for the given target fault. In a conventional Stored Pattern approach, the "don't care" bit values have to be explicitly stored as well, requiring 12 bits of storage for the test.

This small example demonstrates how to link test generation with PRPG Re-Seeding. The test generation

algorithm selects a fault from the list of untested faults and generates a test for this initial target fault. The test requires specific values from a typically small subset of Channel SRLs (on the order of 10 to 100). These are the "care" bit values for the target fault. Utilizing the equations (1), a linear equation with the Seed bit values as independent variables is formed for each "care" bit. The resulting system of linear equations can be solved for the Seed bit values by any linear equation solving algorithm. If there are fewer equations than independent variables, then additional tests can be "subsumed". Another untested fault is selected and the test generation algorithm is invoked again with this new target fault. The ensuing "care" bit values are accepted if they have no conflict with any previously assigned "care" bit value in the presently processed pattern. This procedure is repeated until a) no more conflict-free test can be found, or b) the cumulative system of linear equations can no longer be solved. In the latter case, the "care" bit equations of the offending test are rejected and the equations are solved for all previous tests. The resulting Seed is saved. The rejected fault test, if any, becomes the initial target fault for a subsequent Seed. A single Channel Load operation with the Seed is simulated to determine the complete test pattern, and fault simulation is used to determine any additional faults that may have been tested by serendipity. The process is repeated with any remaining untested faults and additional Seeds are generated until all faults have been tested, proven redundant, or the test generation algorithm has attempted, but failed to generate a test or prove redundancy for the fault.

3.3 Test Pattern Lock-Out and PRPG Size

Under certain conditions it may be impossible to solve the derived system of linear equations because the equations are dependent. This clearly is very likely if there are more "care" bits than there are Seed bits; that is, the number of derived equations exceeds the number of independent variables. The problem can, however, also occur if the number of equations is smaller than the number of independent variables. For example, in the structure of Figure 1 there is no solution for Y4=0, Y9=1, Y10=0. This can be seen by XORing the XOR-Sum expressions for the right hand sides of the equations (resulting in logic 0) and the values on the left hand side of the equations (resulting in logic 1). None of the 16 possible Seed bit value combinations can produce the desired result. We refer to this phenomenon as Test Pattern Lock Out. Lock Out of a particular fault test can only occur if the right hand sides of the associated equations are dependent. A solution to the equation system can still exist, if the requested values match the dependency. This is, for

example, the case for $Y_4=1$, $Y_9=1$, $Y_{10}=0$. The probability for dependency is, hence, an upper bound for the probability of experiencing Lock Out. Given a randomly selected group of m "care" bits and an n -bit PRPG, the probability of dependency (dependency fraction) in the associated equations can according to [10] be calculated as

$$(3) \quad P(m;n) = \prod_{i=0}^{m-1} \frac{(2^{n-1}) - (2^i - 1)}{(2^{n-1}) - 1}$$

For $m < n$ this expression can be approximated by the easier to visualize formula

$$(4) \quad P(m;n) = 1 - \exp(-2^{(m-n)})$$

To assure full test coverage, we must try to avoid Lock Out to occur on any single fault test (if Lock Out occurs during subsumption of additional tests, more test patterns may be required, but the test coverage is not affected, since the rejected fault test is chosen as the initial target fault for a subsequent pattern). Then, the same test coverage can be achieved as with Stored Pattern testing or WRP (we are essentially using the same test generator). This is practically achieved by making the number of PRPG bits, n , larger than the largest expected number of "care" bits, m_c , of any single fault test. As can be seen from equation (4), with $n = m_c + 20$ the probability of Lock Out for any single fault test becomes smaller than 1 in a million. Results from test generation for very large LSSD structures have shown isolated faults requiring on the order of 150 to 200 "care" bits. This suggests that a 256-bit PRPG would be a safe choice. Taking the XOR-Network and clocking into account, such PRPG structure could be fit into about .7% of the CMOS example chip area, leaving room for a 64-bit Signature register within the "allowed" 1% overhead for improved Self-Test coverage. It should be noted that individual fault tests with such a large number of "care" bits are very rare exceptions. Almost all faults are testable with a much smaller number of "care" bits. If overhead is a concern, the PRPG size could be reduced to 128 bits at a small risk of Lock Out on a few extreme faults.

4. ANALYSIS OF THE RE-SEEDING EFFICIENCY

The Re-Seeding method starts after applying 100K to 200K Pseudo Random Patterns without Re-Seeding. At this point the test coverage is already at 90% to 95%. For the CMOS chip example this leaves 75K to 150K untested faults. These can be reduced into 50K - 100K fault equivalence classes. Only one test needs to be generated for each class. Based on, albeit cursory, information about test generation statistics for very large LSSD structures, we are led to expect that the vast majority of the remaining untested faults can be tested with somewhere between 20 and 40 "care" bits. Only relatively few faults require more. Consequently, we assume that in general it is possible to subsume several fault tests into a single Seed. Because of the granularity in the number of "care" bits and the possibility of running out of subsumable faults or experiencing Lock Out during subsumption, some inefficiency must be expected when compressing the tests into PRPG Seeds. We estimate that each 256-bit Seed can accommodate an average of 5 fault tests with 40 "care" bits each (resulting in 200 equations for the 256 independent variables). Thus, 10K - 20K Seeds of 32 Bytes each are needed for all remaining faults. The total storage for the Seed values is 320 KBytes - 640 KBytes. This extrapolation ignores that some of the generated test patterns will, by serendipity, detect additional faults besides the target faults that were explicitly used in generating the Seeds. Such additional fault detection will further reduce the number of Seeds.

This estimated data volume is up to two orders of magnitude less than what was predicted for supplementary Stored Pattern tests, and even less than half of what is expected for WRP. The reason for this somewhat surprising result is that the density of "care" bits in LSSD test patterns is typically very low (at high test coverages). Almost all of the bits are "don't care" bits. Both, Stored Pattern Testing and WRP, waste valuable storage in explicitly representing data for the "don't care" bit positions. The proposed method, on the other hand, filters the "care" bits out of the test pattern and compresses only this information into a PRPG Seed. The "don't care" bit values are implicitly produced by the PRPG and are not explicitly stored at all.

The expected test pattern count for the Re-Seeding method is around 220K, which is comparable to the other methods. However, by using a built-in PRPG and Signature Register with 64 on-product Channels of 300 SRLs each, the test time can be significantly reduced compared to applying WRP tests from a Reduced Pin Count tester through 16 externally connected Channels of 1000 SRLs each as described in [6]. Re-Seeding could be performed from 16 scan-in pins in parallel. Thus, Re-Seeding takes 16 scan clock cycles and the

Channel Load 300 scan clock cycles for a total of 316 scan clock cycles for a test pattern. A WRP Channel Load, by contrast, takes 1000 scan clock cycles, that is 3 times as long. It should be noted that for a true comparison Table 2 summarizes the expected results in comparison with the other methods. The numbers represent the upper bounds of the ranges used in the text above. The overheads are calculated on the basis of a 64-bit PRPG and Signature Register for the Pseudo Random Pattern plus Stored Pattern supplement option (Rnd. + Stored), 64 Kbits of array storage plus 128-bit PRPG and 32-Bit Signature Register for on-product WRP, and a 256-bit PRPG plus 64-bit Signature Register for the Re-Seeding method. Chip test times assume a 50ns scan clock cycle and 16 Channels of 1000 SRLs each for Stored Pattern testing and off-product WRP, 32 Scan Channels of 500 SRLs each for on-product WRP, and 64 Channels of 300 SRLs each for Pseudo Random Patterns and the Re-Seeding option. The test times do not include the overheads cited above.

Method	Storage (Bytes)	Patterns	Test Time	Overh
Rnd. + Stored	38 M	220 K	4.3 sec	.5%
off-product WRP	1.6 M	270 K	14.0 sec	n/a
on-product WRP	1.6 M	270 K	7.0 sec	7%
Rnd. + Re-Seed	.7 M	220 K	3.7 sec	1%

Table 2: Summary of Estimated Results

5. SUMMARY

We have outlined a method for the intelligent Re-Seeding of linear Pseudo Random Pattern Generators as a supplement of Pseudo Random Pattern testing to achieve full test coverage. An algorithm for deriving pattern generator Seed values from test generation results has been described and analyzed. The method is shown to accomplish very compact encoding of the tests by ignoring "don't care" bit positions in the test patterns. The estimated resulting Seed value data volumes are much smaller than the data volumes for Stored Pattern Testing and even smaller than those expected for Weighted Random Pattern testing. The necessary on-product hardware for Self-Test can be incorporated with an overhead of only around 1% of the area of a state-of-the-art CMOS chip. This makes it possible to take full advantage of the data compression

comparison of tester throughput other factors like wafer load/alignment, stepping, parametric testing, embedded array testing, etc. have to be taken into account as well.

method not only in chip testing, but also in a (sub-)system Self-Test environment. Based on the estimates developed in the paper, we expect the proposed method to be very competitive with Weighted Random Pattern testing.

6. REFERENCES

- [1] "Analog Computer Progress in 1957", IRE Trans. on Electronic Comp., pp. 68 - 69, March 1958
- [2] K. Maling and E.L. Allen, "A Computer Organization and Programming System for Automated Maintenance", IEEE Trans. on Electronic Comp., pp. 887 - 895, 1963
- [3] T. Kobayashi et al., "A Flip-Flop Circuit for FLT", Proc. Electronics and Comm. Society of Japan, p. 962, 1968
- [4] E.B. Eichelberger and T.W. Williams, "A Logic Design Structure for LSI", Proc. 14th Design Automation Conf., pp. 462 - 468, 1977
- [5] R. Eldred, "Test Routines Based on Symbolic Logical Statements", ACM Journal, pp. 33-36, January 1958
- [6] R.W. Bassett et al., "Low-Cost Testing of High-Density Logic Components", IEEE Design & Test of Comp., pp 15 - 28, April 1990
- [7] B.L. Keller and T.J. Snethen, "Built-In Self-Test Support in the IBM Engineering Design System", IBM Journal of Res. & Dev., pp. 406 - 415, March/May 1990
- [8] C.W. Starke, "Design for Testability and Diagnosis in a VLSI CMOS System/370 Processor", IBM Journal of Res. & Dev., pp. 352 - 362, March/May 1990
- [9] J.A. Waicukauski, "A Method for Generating Weighted Random Patterns", IBM Journal of Res. & Dev., pp. 149 - 161, March 1989
- [10] P.H. Bardell et al., "Built-In Test for VLSI", John Wiley and Sons, New York, pp. 81ff, 1987