Enhancing Path Delay Fault Coverage by Weighted Pseudorandom Test Generation

Øystein Gjermundnes, Einar J. Aas

Abstract − **The implementation of a system for analyzing circuits with respect to their path-delay fault testability is presented. It includes a path-delay fault simulator, and an ATPG for path-delay faults combined into a test tool. The test tool is used to evaluate the performance of several different test vector generators. The test generators exploit weighted pseudo-random stimuli generation, based on arithmetic BIST and SIC patterns. The main goal is to find efficient heuristics that improves path-delay fault detection efficiency in terms of test time. We show that weighted ABIST stimuli are productive for detecting the K-longest path-delay faults for most circuits. On the average, we obtained fault coverage of 92.6% for the 20.000 longest paths on iscas'85 circuits.**

Index Terms − **Built-in testing, Fault diagnosis, Automatic testing.**

I. Introduction

Defect oriented testing is gaining attention, and Path Delay Fault (PDF) testing is one of the more challenging problems to study [9]. The test method toolbox has expanded significantly over the last decade. Various trade-offs on test methodology, test quality (measured by various fault coverage metrics), design-for-test development costs, silicon overhead, and cost of Automatic Test Equipment, including test application time, are performed.

For PDF testing, deterministic test pattern pairs, or Built-In Self-Test (BIST) generated patterns may be exploited. We have chosen to explore the possible usage of BIST methods. This paper describes the implementation of a system for analyzing circuits with respect to their path-delay fault testability. The system includes a path-delay fault simulator, and an Automatic Test Pattern Generator (ATPG) for path-delay faults, combined into a test tool. The test tool is used to evaluate the performance of different test vector generators that may be used in various BIST arrangements. The test generators exploit weighted pseudo-random stimuli generation, based on arithmetic BIST principles. We show that this is a viable BIST method for detecting the K-longest path-delay faults with satisfactory PDF coverage for many circuits, but not for all circuits. We employ the tool on iscas'85 circuits. Our focus is on the methodology, not on specific stimuli generators. We envision the use of compact software programs, like published [8], to be loaded into the system under test. An in-depth presentation of this test project is found in [5].

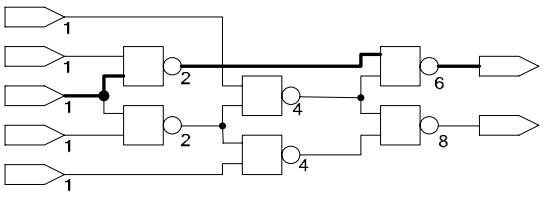

Fig. 1. c17 with one of its paths highlighted

II. Path Delay Fault Simulation Model

The path-delay fault model was proposed by Smith [9]. A definition of the path-delay fault model from [1] is:

The delay defect in the circuit is assumed to cause the cumulative delay of a combinational path to exceed some specified duration. The combinational path begins at a primary input or a clocked flip-flop, contains a connected chain of gates, and ends at a primary output or a clocked flip-flop. The specified time duration can be the duration of the clock period (or phase), or the vector period. The propagation delay is the time that a signal event (transition) takes to traverse the path. Both switching delays of devices and transport delays of interconnects on the path contribute to the propagation delay.

There are *two* path-delay faults associated with each physical path in the circuit: slow-to-rise, and slow-to-fall. Fig. 1 shows one path. The path-delay fault model has the ability to detect distributed defects caused by statistical process variations. A test for a path-delay fault will also detect any spot defects along the path. The number of paths,

Manuscript received February 4, 2008.
The work was done while Gjermundnes was affiliated with NTNU.
Øystein Gjermundnes with the ARM Norway, PBox N-2182, NO-7412 Trondheim Norway, e-mail: oystein.gjermundnes@arm.com
Einar J. Aas with the Norwegian University of Science and Technology – *NTNU, NO-7491 Trondheim, Norway, e-mail: ejaas@iet.ntnu.no*

and thus path-delay faults, may be exponential in the number of gates in the circuit.

The selection of proper *simulation algebra* (alphabet and logic rules) is crucial for any logic/fault simulator. Our simulator **PDFSim** uses the **6**-valued algebra developed by [9]. Several features to obtain an efficient simulator are presented in [5], see also [4]. Of course, a *two-pattern* test vector is needed for delay fault testing. We adopt SIC (Single Input Change) vectors, because it was shown in [11] that such vectors are more effective than Multiple Input Change vectors for robust and non-robust testing.

III. Automatic Test Pattern Generation

It is intractable to test all path-delay faults in a circuit. There are nearly $\mathbf{10^{20}}$ paths in one of the iscas'85 circuits! One accepted strategy is to test a subset of all possible path delay faults. The longest testable paths are of particular importance for high quality delay testing. An algorithm for extracting the K-longest testable path-delay faults (K-LT-PDF) in a circuit has been developed, and integrated with the fault simulator. The test generators employed will be evaluated against the fault lists containing K-LT-PDF.

The earliest attempts at creating an ATPG that could extract the K-LT-PDF were very inefficient. ATPGs normally employed two separate phases. Usually, a lot of paths are untestable, and a structural path extractor would find and pass a lot of untestable paths to the test generator. Fortunately, by combining the structural path extractor and the test generator, it is possible to prune the search space significantly by sorting out untestable sets of paths at an early stage. This approach was originally used by Qiu and Walker [12]. We have introduced several improvements in terms of efficiency, including recursive learning [6], and FAN-like [3] justifications. Recursive learning is a method for extracting all logical dependencies between signals in a circuit, and to perform *precise* implications for a given set of value assignments.

IV. BIST-based Stimuli Generators

A. Basis Vectors

First, we wanted to investigate whether ABIST generators of a simple kind, namely accumulator based stimuli generators, would provide sufficient basis for pseudo-random patterns. In particular, the generator described in [8] was investigated:

$$\mathbf{A_i = A_{i-1} + C \pmod{2^n}, A_0 = I,} \quad i=1,2,3,\ \ldots,\ V \qquad (1)$$

By carefully selecting the parameters **C** and **I**, one may exhaustively cover every subinterval of size **r** within the first $\mathbf{2^r}$ test vectors. This generator may be implemented as a compact software program in a micro controller. It will generate uniformly distributed values. Let us call these patterns UDB (Uniformly Distributed Basis) patterns.

But are these generated values of adequate statistical quality? We compared the generator against a Mersenne Twister (MT) generator [7]. This generator is considered as an excellent benchmark for uniformly generated pseudorandom numbers. But it is much more complex to implement in SW or HW. The simple ABIST generator given in (1) was not as efficient. But by combining *three* generators of type (1), and proper weighting, we developed a better basis, called **GAU** (U –for uniform). This generator yields considerably shorter test application times than a Mersenne-based generator will.

The rationale behind the use of weighted test patterns is as follows: consider Fig. 1. For a path to be sensitized from input to output, proper controlling values must be applied to the inputs not included in the path. We are looking for ABIST patterns that exhibit statistical properties inductive to fault detection. It is known that proper weighting of input values, i.e. non-uniform distribution of ones and zeros, might enhance the efficiency of fault detection.

Thus, we devised various schemes for weighting the random patterns. These schemes employed the basic generator, with added features for weighing. Transitions on input pins were generated by so-called Single Input Change (**SIC**) vectors. From a basis vector, we toggle one bit at a time to obtain two-pattern test vectors. For an N-input circuit, **2N** vectors are generated this way.

First, we define the **GA1** generator: use of the **GAU** generator, and **SIC** vectors. This yields a uniform generator, which we will compare potential weighting heuristics against.

B. GA2: stuck-at test set weights

Weights are based on a deterministic test set (obtained from a commercial ATPG) for stuck-at faults. For each input pin, we counted the relative number of ones and zeros, and used these numbers as weights. Don't cares were counted in both the one and the zero set. Basis vectors are generated from (1), with **r**=16, and three sets of (C, I) values.

The rationale is that these patterns have contributed to controlling values on the inputs for efficient stuck-at fault detection, and may be promising as candidates for path delay fault testing as well.

C. GA3: counting based weights

Weights are generated based on fault coverage measurements. The circuit is first fed from a pseudo-random generator of type **GA1**. Two counters (**S0**Ctr, **S1**Ctr) are associated with each input. These counters store the number of *path-delay faults* detected when the input has a stable value (**S0** or **S1**). When a predetermined number of basis patterns (10M) has been applied, the weighting factors can be computed for every input according to:

$$p0 = \mathbf{S0}\text{Ctr}/(\mathbf{S1}\text{Ctr} + \mathbf{S0}\text{Ctr}), \qquad (2)$$

$$p1 = \mathbf{S1}\text{Ctr}/(\mathbf{S1}\text{Ctr} + \mathbf{S0}\text{Ctr}). \qquad (3)$$

Subsequently, we rerun the fault simulator with these weights. This yields the generator **GA3.** This heuristic is

inspired from the fact that patterns with more weight on the HIGH value are productive for AND/NAND gate testing.

Notice that the counting is not activated before 100 basis patterns have been applied. This will leave out the easy-to-detect faults. These faults will be detected anyway.

D. GA4 and GA5

Two less successful schemes were **GA4** and **GA5**. **GA4**: similar to **GA2**, but weights were computed with "reseeding". One output pin at a time was considered when recording fault detection of a test vector. The weight set was recomputed once for every output pin.

GA5 is similar to **GA4**, but the sequence of seeds was optimized somehow.

E. GA6: weights based on deterministic tests

Similar to **GA2**, except that weights are generated based on a deterministic test set for path-delay faults. First, a test set for the 20.000 longest paths of non-robust faults was generated. Then, for each pin, we computed the ratio of ones (zeros) that occurred in the complete test set. Don't cares were counted twice, both as 0 and 1. These values were used as weights throughout the experiment, similar to **GA3** above.

V. EXPERIMENTS

Armed with the tools and generators described above, several experimental runs were set up.

A. Benchmark circuit properties

Circuits from the iscas'85 benchmark suite were engaged in the experiments presented below. Some information about each circuit is provided in this section.

The number of inputs (I), outputs (O), gates (G), logical levels (L) and physical paths (P) for each circuit is shown in Table 1 (the two last columns will be discussed in Section 5.2.1). The number of paths is much larger than the stuck-at fault set. Notice in particular the huge number of paths for benchmark c6288 (a *16x16* bit array multiplier).

The circuits c432 and c499 are omitted from most of the experiments because they contain XOR-gates, which are not currently supported by the ATPG. Another circuit that is omitted from most experiments is c6288. The large number of paths in this circuit causes problems for both the simulator and the ATPG. C17 is discarded for its simplicity. The rest of the benchmarks are used in all experiments.

TABLE 1
BENCHMARK PROPERTIES

Circuit	I/O	G/L	P	UB	PF
c880	60/26	469/25	8642	16652	16652
c1355	41/32	619/25	4173216	1110076	20000
c1908	33/25	938/41	729057	355197	20000
c2670	233/140	1566/33	679960	1306884	20000
c3540	50/22	1741/48	28676671	12330969	20000
c5315	178/123	2608/50	1341305	353300	20000
c6288	32/32	2480/125	10**20	-	-
c7552	207/108	3827/44	726494	282752	20000

B. Experimental results

This section presents some statistical properties of the sequences generated by the different test generators described in Section 4. This information can be used as an aid in the interpretation of the results.

EX1: Find the K-longest testable paths

The objective of this experiment was to find the longest non-robust testable paths of each benchmark circuit, which was done by using the ATPG tool described in Section 3. Provided unlimited time and memory, the tool would list all testable faults in each circuit. Unfortunately, some of the circuits contain a huge number of testable path-delay faults, and this would cause the size of the data structure inside the ATPG tool to blow up. In order to keep the whole path store inside computer memory, the size of the path store was set to a maximum of 1M. The ATPG was asked to find the 20.000 longest non-robust testable paths in each of the benchmarks. The number of such paths found (PF) for the different circuits are listed in the last column of Table 1, together with an upper bound (UB) [2] of all non-robust path delay faults. Since all circuits except c880 contain more than 20.000 testable paths, the ATPG had no problem finding 20.000 testable paths. It is reassuring to notice that the upper bound of c880 from [2] coincides with the number of paths we found.

EX2: Determine no. of paths detected by unweighted pseudo-random stimuli

In this experiment test vectors were applied, and the number of detected path-delay faults and their length were logged. The test vectors were generated with an unweighted Mersenne Twister pseudo-random generator (GT1). The purpose was to obtain information about the number of paths of different lengths detected by a standard pseudorandom generator. Typical results are presented in Figure 2.

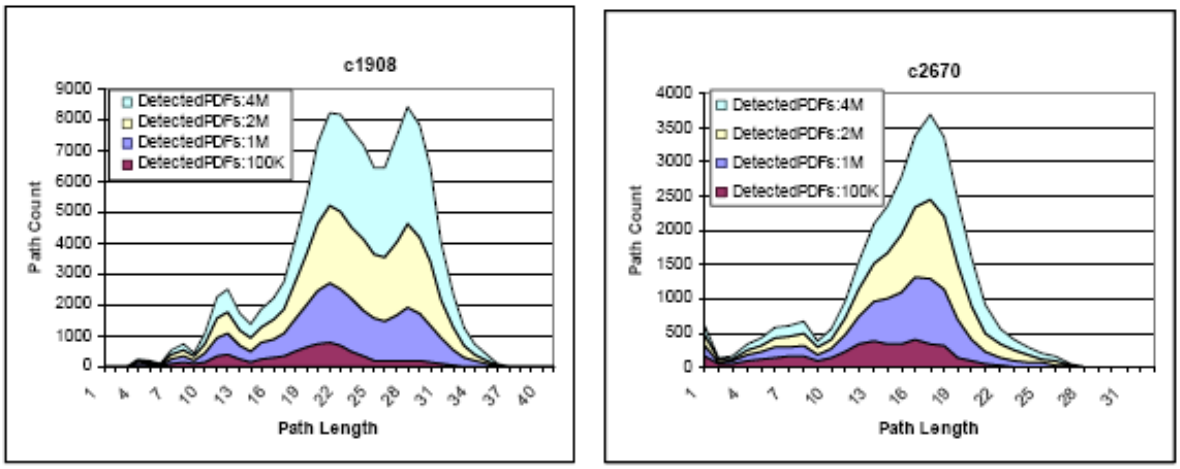

Fig 2. No. of detected paths vs. path length for various no. of test vectors

The longer paths are not as easily detected as the shorter paths. This is to be expected; long paths need more constraints than shorter paths.

In fact, we found that for all circuits, a randomly selected physical path is longer than the average length of the paths detected within 4M test vectors. The average physical path was from 9.8% to 72 % longer. Clearly, UDB patterns are not effective for detecting the longest PDFs.

EX3: Comparison of GA1 - GA5

In this experiment the performances of GA1 - GA5 were evaluated. The experiment exploited the fault simulator described in Section 2. ***10M*** test patterns were simulated for each circuit and generator. Each simulation run was repeated 10 times with different seeds in order to cover statistical variations. Table 2 presents the average number of detected faults over 10 trials after ***10M*** applied test vectors.

TABLE 2
DETECTED FAULTS AFTER *10M* APPLIED TEST VECTORS

Circuit	GA1	GA2	GA3	GA4	GA5
c880	8714	16194	**16550**	16470	16473
c1355	1050139	1085021	**1110297**	1110264	1110258
c1908	269846	283665	**349613**	349579	349568
c2670	51739	85948	**107711**	102734	104141
c3540	588541	996001	**1062718**	1050384	1050579
c5315	173526	309498	**339396**	339122	339157
c7552	146754	**185983**	185687	185264	185383
Sum	2289259	2962310	**3171972**	3153817	3155559

The best result, i.e. the highest number of detected faults, is shown in bold in Table 2 for each circuit. The stimuli generator with the poorest performance is the unweighted pseudo-random generator GA1. This generator detected the fewest number of non-robust path delay faults in all tests. Generator GA2, which is a weighted pseudo-random generator with weights based on a deterministic test set for stuck-at faults, is somewhat better than GA1. The best generators are GA3, GA4, GA5 and GA6.

The performances of GA3, GA4 and GA5 do not differ by much, but the results point in favor of GA3, which detects most path-delay faults for all but one benchmark. GA3 is a weighted pseudo-random generator with weights based on the counting scheme described in Section 4.

We performed the same experiments with the MT as the basic pseudorandom generator. To summarize, the results were in general only slightly better than for the ABIST generator. For example, the equivalent of GA3 exhibited a *total improvement* (summed over all circuits) of 0.12% more detected path delay faults.

EX4: Weighted pseudo-random patterns to find the K-longest testable path-delay faults

The purpose of this experiment was to find out if proper weighting of pseudorandom stimuli, based on *K=20.000* deterministic test patterns for path-delay faults, would yield more efficient path delay tests than using uniformly distributed patterns. The experiments were conducted as follows:

First, the *K=20.000* longest testable path-delay faults were extracted for each circuit as described in EX1. For each detected path, the path number was stored in a file together with the corresponding path length and test vector. Weights for GA6 and GT6 were then extracted based on each test set as described in Section 4. Notice that generators labeled GTi refers to the use of Mersenne Twister random numbers, but with same heuristics as the corresponding GAi (i= 1-6).

Prior to each simulation run a fault list with the *20.000* longest testable path delay faults was uploaded to the simulator. ***10M*** single-input-change test patterns were then applied to each circuit for each generator. Each simulation run was repeated 10 times with different seeds in order to cover statistical variations. Six different generators were used: GA1, GA3, GA6, GT1, GT3 and GT6.

The three generators GA1, GA3 and GA6 are using the exact same underlying accumulator based pseudo-random generator. GA3 and GA6 are weighted pseudo-random generators, and will be compared against GA1 (uniform

weights). The three generators GT1, GT3 and GT6 are using the exact same underlying MT pseudo-random generator. GT3 and GT6 are weighted pseudo-random generators, and will be compared against GT1 (uniform weights).

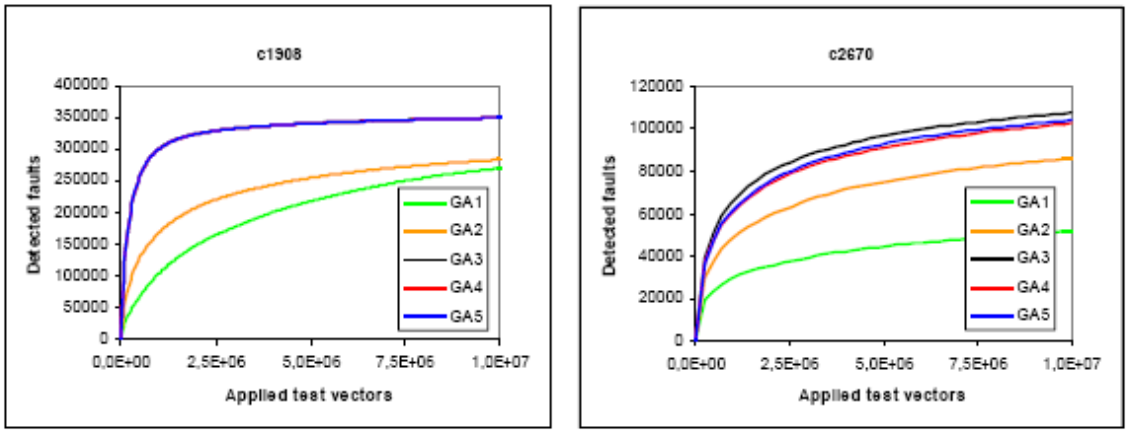

Fig 3. Typical curves of fault detection vs. no. of test vectors applied

Two measures were recorded:

Fault coverage in relation to the size (K) of the fault list. (*K=20000* for all circuits except c880 which contains only 16652 non-robust testable paths).

Test time speedup, defined as the ratio:

$$R_{imp}(meth_x)=NTP(uniform)/NTP(meth_x), \quad (4)$$

where NTP represents the number of test patterns. The name of the stimuli generator is used as argument $(meth_x)$.

TABLE 3
FAULT COVERAGE, FC, OF BEST METHOD AFTER *10M* APPLIED TEST VECTORS

Circuit	FC(GA_x)
c880	99.3% (GA3)
c1355	100% (GA3)
c1908	97.6% (GA3)
c2670	67.9% (GA6)
c3540	86.9% (GA6)
c5315	96.7% (GA6)
c7552	99.8% (GA3)
Average	92.6%

During simulation the fault coverage, FC, was sampled from time to time until ***10M*** test patterns had been applied. Figure 3 shows two typical curves of fault detection vs. no. of test vectors applied. The lowest curves represent unweighted stimuli, while the other curves are given for four different weighting schemes. The improvements are notable for GA3 and GA5.

Table 3 shows the fault coverage after ***10M*** test vectors. The numbers in the second column represent fault coverage achieved with the best generator of GA3 and GA6.

We observed that the GT methods are slightly better than the GA methods. Furthermore, 5 out of 7 circuits attain 97.6% fault coverage, or more. Two circuits exhibit inferior fault coverage, and need more test patterns or other methods of path-delay fault detection.

As mentioned, similar experiments were carried out with the MT as the basic pseudorandom generator, in order to check possible improvement when using a more authoritative pseudorandom generator. These generators are called GT1-GT6. The MT resulted only in slight improvements. The average fault coverage increased from 92.6% to 93.6%.

The standard deviation of the sample fault coverage after *10M* applied test patterns over the 10 trials was also computed. It varied from 0% to 1.5%. Thus, the seed value does not influence the outcome much.

C. Test time speedup

One important goal in testing is the ability to obtain a desired test quality for less cost. In our case, test time, i.e. no. of test vectors to be applied for a given test quality, should be kept at a minimum. In order to measure the speedup of a weighted generator over that of a uniformly distributed pseudo-random generator, one can compare the number of test vectors needed in order to achieve the same fault coverage. The target coverage in our case was set to the fault coverage attained with the *unweighted generator* after application of *10M* stimuli. The improvement factors of the best-weighted generator over uniformly distributed stimuli, defined in (4), are listed in Table 4.

TABLE 4
TIME SPEEDUP OF BEST METHOD OVER UNIFORMLY DISTRIBUTED STIMULI

Circuit	$R_{imp}(GA_x)$	$R_{imp}(GT_x)$
c880	11.9	15.1
c1355	1.5	2.7
c1908	8.0	10.8
c2670	10.7	14.3
c3540	7.1	9.1
c5315	4.7	7.0
c7552	1.0	1.0

We observe time speedups from nothing to a factor 11.9 (GA) or 15.1 (GT)! However, it is unfortunately not possible to devise an a priori metric that may predict speedup. But given the potential of substantial savings in

test time, and thus savings of test cost, it can be recommended to experiment with GA3 and GA6 for a newly designed circuit, and use this method whenever beneficial.

6. Conclusion

A system for analyzing circuits with respect to their path-delay fault testability has been presented. It includes a path-delay fault simulator, and an ATPG for path-delay faults combined into a test tool. This tool was used to evaluate the performance of different test vector generators for various BIST arrangements. The test generators exploit weighted pseudo-random stimuli generation, based on arithmetic BIST principles. We did find useful heuristics that improve path-delay fault efficiency in terms of test time. We showed that weighted ABIST stimuli are productive for detecting the K-longest path-delay faults for many circuits. On the average, we obtained fault coverage of *92.6*% for the *20.000* longest paths on a subset of iscas'85 circuits. We observed time speedups from nothing to a factor 12 with the accumulator based stimuli generator, making it well worth the effort of experimenting with such methods for potential high quality path-delay fault testing. However, it should be noted that our methods do not always give significant improvements, and are not generally applicable.

Future work would involve using the simulator and the ATPG to create better generators based upon knowledge about the structure of the circuit. We might also investigate the influence the number of longest paths will have on the test quality obtainable.

References

[1] M. L. Bushnell and V. D. Agrawal, *Essentials of electronic testing for digital, memory, and mixed signal VLSI circuits*", Kluwer Academic, New York, 2002.

[2] K. T. Cheng and H. C. Chen, "Delay testing for non-robust untestable circuits", *Proc. of the International Test Conf.*, 1993, pp. 954–961.

[3] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms", *IEEE Trans. on Computers*, C-32(12), 1983, pp. 1137-1144.

[4] O. Gjermundnes and E. J. Aas. "Efficient stimuli generators for detection of path delay faults", *Proc. of the 48th Midwest Symp. on Circuits and Systems*, 2005, pp. 1709–1712.

[5] Ø. Gjermundnes, "Exploiting Arithmetic Built-In Self-Test Techniques for Path Delay Fault Testing". *Doctoral thesis*, 2006, Norwegian University of Science and Technology, ISBN 82-471-8257-2.

[6] W. Kunz and D.K. Pradhan, "Recursive learning – a new implication technique for efficient solutions to cad problems – test, verification, and optimisation", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13(9), 1994, pp. 1143-1158.

[7] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation*, 8(1), 1998, pp. 3–30.

[8] J. Rajski and J. Tyszer, *Arithmetic built-in self-test for embedded systems*. Prentice Hall, Upper Saddle River, N.J., 1998.

[9] G. L. Smith. "Model for delay faults based upon paths", *Proc. of the International Test Conf.*, 1985, pp. 342–349.

[10] A. Ströle and H.-J. Wunderlich, "TESTCHIP: A chip for weighted random pattern generation, evaluation, and test control", *IEEE Journal of Solid State Circuits*, Vol. 26, No. 7, 1991, pp. 1056-1063.

[11] A. Virazel et al., "Delay fault testing. Choosing between random sic and random mic test sequences", *Journal of Electronic Testing – Theory and Applications*, Vol. 17, No. 3/4, 2001, pp. 233-241.

[12] Q. Wangqi and D. M. H. Walker, "An efficient algorithm for finding the K longest testable paths through each gate in a combinational circuit", *Proc. of the International Test Conf.*, Vol. 1, 2003, pp. 592–601