Computer Science
Technical Report

Antirandom Testing: Beyond Random Testing

ShenHui Wu
Yashwant K. Malaiya*
A.P. Jayasumana

Electrical Engineering Dept.
*Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
malaiya@cs.colostate.edu

Technical Report CS-98-103

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

Antirandom Testing: Beyond Random Testing

ShenHui Wu
Yashwant K. Malaiya*
A.P. Jayasumana

Electrical Engineering Dept.
*Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
malaiya@cs.colostate.edu

ABSTRACT

Random testing is a well known concept that requires that each test is selected randomly regardless of the test previously applied. In actual practice it takes the form of pseudo-random testing, where each test pattern is a shifted version of the previous one with one new bit added. This paper introduces the concept of antirandom testing. In this testing strategy each test applied is chosen such that its total distance from all previous tests is maximum. This spans the test vector space to the maximum extent possible for a given number of vectors. This strategy results in a higher fault coverage when the number of vectors that are applied is limited. Algorithm for generating antirandom tests is presented. A Reed-Solomon code based test set is also introduced that results in test vectors with antirandom characteristics. Results comparing the different test strategies on ISCAS benchmarks show these strategies to be very effective when a high fault coverage needs to be achieved with a limited number of test vectors. The superiority of the antirandom testing approach is even more significant for testing bridging faults.

1 Introduction

Random testing avoids the problem of deterministic test generation that requires structural information about the circuit under test to be processed for generating each test. Available evidence suggests that random testing may be a reasonable choice for obtaining a moderate degree of confidence, however it becomes inefficient when only hard to test faults remain [7]. Further, in testing digital circuits, pseudo random test vectors are used. This results in a test vector which is a shifted version of the previous pattern, with one new bit based on a primitive polynomial [1].

Random testing does not exploit some information that is available in black-box testing environment. This information consists of the previous tests applied. If an experienced tester is generating tests by hand, he would select each new test such that it covers some part of the functionality not yet covered by tests already generated. The objective of this paper is to formally define an approach that uses this information and to propose schemes that may allow such test generation to be done automatically. We term this approach *antirandom* testing, since selection of each test explicitly depends on the tests already obtained.

In section 2 we define and characterize the antiranodm test patterns. A procedure for generating antirandom tests is described. Section 3 describes a procedure for generating tests having antirandom properties based on Reed-Solomon codes. Then we compare the different strategies by evaluating their effectiveness on ISCAS benchmark circuits as well as a widely used ALU circuit. We extend the antirandom concept further by proposing a functional test generation scheme in which we find input vectors that would cause antirandom patterns at the output of a circuit. This scheme is evaluated on a popular ALU circuit.

2 Binary Antirandom Sequences

Antirandom testing [6] is a black-box strategy like psuedo-random testing, meaning that it assumes no information about the internal implementation of the circuit. Here we start with formal definitions of the terms used and then examine construction of antirandom sequences. We assume that the input variables are all binary.

Definition: **Antirandom test sequence (ATS)** is a test sequence such that a test t_i is chosen such that it satisfies some criterion with respect to all tests t_0, t_1, ... t_{i-1} applied before. In this paper we use two specific criteria introduced below.

Definition: **Distance** is a measure of how different two vectors t_i and t_j are. Here we use two measures of distance defined below.

Definition: **Hamming Distance (HD)** [3] is the number of bits in which two binary vectors differ. It is not defined for vectors containing continuous values.

Definition: **Cartesian Distance (CD)** between two vectors, $A = \{a_N, a_{N-1}, ...a_1, a_0\}$ and $B = \{b_N, b_{N-1}, ...b_1, a_0\}$ is given by:

$$CD(A,B) = \sqrt{(a_N - b_N)^2 + (a_{N-1} - b_{N-1})^2 + .. + (a_0 - b_0)^2} \quad (1)$$

If all the variables in the two vectors are binary, then equation 1 can be written as:

$$\begin{aligned} & CD(A,B) \\ = & \sqrt{|a_N - b_N| + |a_{N-1} - b_{N-1}| + .. + |a_0 - b_0|} \\ = & \sqrt{HD(A,B)} \end{aligned} \quad (2)$$

Definition: **Total Cartesian Distance (TCD)** for any vector is the sum of its Cartesian distances with respect to all previous vectors.

Definition: **Maximal Distance Antirandom Test Sequence (MDATS)** is a test sequence such that each test t_i is chosen to make the total distance between t_i and each of to,t_1... t_{i-1} maximum, i.e.

$$TD(t_i) = \sum_{j=0}^{i-1} D(t_i, t_j) \quad (3)$$

is maximum for all possible choices of t_i. We will use Hamming distance and Cartesian distance to construct MHDATSs and MCDATSs.

For functional testing, we have no structural information available about the actual implementation. Using maximal distance criterion, every time we attempt to find a test vector as different as possible from all previously applied vectors. The antirandom testing scheme thus attempts to keep testing as efficient as possible. In this approach we are using the hypothesis that if two input vectors have only a small distance between them then the sets of faults encountered by the two is likely to have a number of faults in common. Conversely, if the distance between

two vectrs is large, then the set of faults detected by one is likely to contain only a few of the faults detected by the other.

If testing is less than exhaustive, then MDAT (maximum distance antirandom testing) is likely to be more efficient than either random or pseudorandom testing. Even when exhaustive testing is feasible, MDAT is likely to detect the presence of faults earlier.

Procedure 1. Construction of a MHDATS (MCDATS):

Step 1. For each of N input variables, assign an arbitrarily chosen value to obtain the first test vector. As discussed below this does not result in any loss of generality.

Step 2. To obtain each new vector, evaluate the THD (TCD) for each of the remaining combinations with respect to the combinations already chosen and choose one that gives maximal distance. Add it to the set of selected vectors.

Step 3. Repeat step 2 until all 2^N combinations have been used. □

This procedure uses exhaustive search. As we will see later, the computational complexity can be greatly reduced.

To illustrate the process of generating MDATS, we consider in detail the generation of a complete sequence for three binary variables.

Example 3: For a system, the inputs {x,y,z} can be either 0 or 1. We will illustrate the generation of MHDATS using a cube with each node representing one input combination.

Let us start with the input {0,0,0}. This does not result in any loss of generality. As we will see later, the polarity of any variable can be inverted. The next vector t_1 of the MHDTS is obviously {1,1,1} with THD(t_1) = 3. At this point, the situation is shown in Fig. 1a, where the input combinations already chosen are marked.

As can be visually seen, a symmetrical situation exists now. Any vector chosen would have HD = 1 from one of the past chosen vectors and HD = 2 from the others. If we allow the variables to be reordered, then without any loss of generality we have the following choices. t_0= {0,0,0}
t_1= {1,1,1}; THD = 3
t_2 = {0,1,0}; THD = 3 *or* t_2 = {1,0,1}; THD = 3

Let us consider the first choice. After t_2 = {0,1,0}, the clear choice for t_3 is {1,0,1} at the opposite corner of the cube. The situation now is shown in Fig. 1b.

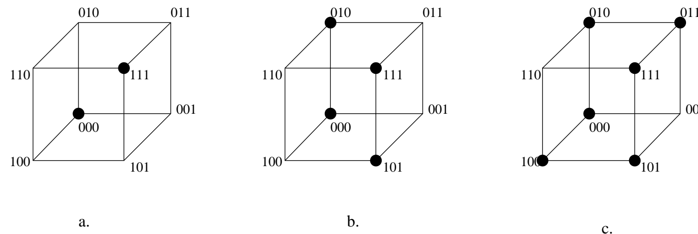

Figure 1: Construction of 3-bit MHDATS

Again a symmetrical situation exists. Any one of the remaining vectors have the same relationship with the set of vectors already chosen. Let us pick {1,0,0} as t_4. The next vector t_5 then has to be {0,1,1} at the opposite corner of the cube. We can again choose any one of two remaining vectors as shown in Fig. 1c. Let us choose t6 = {1,1,0} which leaves t_7 = {0,0,1}. The complete MHDTS obtained here is given as sequence 1 in Table 1. □

Table 1: 3-bit MHDTS (Example 3)

Test	xyz	THD	TCD
t_0	0 0 0		
t_1	1 1 1	3	1.7320
t_2	0 1 0	3	2.4142
t_3	1 0 1	6	4.146
t_4	1 0 0	6	4.8284
t_5	0 1 1	9	6.5604
t_6	1 1 0	9	7.2426
t_7	0 0 1	12	8.9746

In this example, it is easy to see that with our chosen vectors for t_0,t_1 and the two choices for t_2, we could have constructed 16 distinct MHDATSs using all of the later choices available. We can verify that all of these are also MCDATSs.

A large number of experiments with construction of MHDATSs and MCDATSs have been done. Based on these, the following results can be stated [6].

Definition: If a sequence B is obtained by reordering the variables of sequence A, then B is a **variable-order-variant (VOV)** of A.

Theorem 1: If a sequence B is variable-order-variant of a MHDATS (MCDATS) A, then B is also a MHDATS (MCDATS).

The theorem follows from the fact that Hamming or Cartesian distance is independent of how the variables are ordered.

Theorem 2: If a sequence B is a polarity-variant of a MHDATS (MCDATS) A, then B is also MHDATS (MCDATS).

The theorem follows from the fact that for a pair of vectors the distance remains the same, if the same set of variables in both are complemented.

Theorem 3: A MHDATS (MCDATS) will always contain complementary pair of vectors, i.e. t_{2k} will always be followed by t_{2k+1} which is complementary for all bits in t_{2k} where $k = 1, 2, \ldots$.

Procedure 2. Expansion of MHDATS (MCDATS):

Step 1. Start with a complete MHDATS of N variables, $X_{N-1}, X_{N-2}, \ldots X_1, X_0$.

Step 2. For each vector t_i, i = 0, 1, ... (2^N-1), add an additional bit corresponding to an added variable X_N, such that t_i has the maximum total HD (CD) with respect to all previous vectors.□

Procedure 3. Expansion and Unfolding of a MHDATS (MCDATS):

Step 0. Start with a complete (N-1) variable MHDATS (MCDATS) with 2^{N-1} vectors.

Step 1. Expand by adding a variable using Procedure 2. We now have the first $(2^N/2)$ vectors needed.

Step 2. Complement one of the columns and append the resulting vectors to first set of vectors obtained in Step 1. Here, it would be convenient to complement the variable added in Step 1. □

The above procedures have been implemented in a program called ATG [13]. It generates MCDATSs which are also MHDATSs. The application of antirandom testing for software has been reported in [14].

3 Using Reed-Solomon Codes for Test Generation

The basic premise of antirandom testing is that the a new test patterns need to be selected so that it is as different as possible from all the previous patterns applied. The error correction and detection codes used in communications and data storage applications can generate code words with appropriate minimum Hamming distances. Here we have chosen to use the Reed-Solomon (RS) error correction code [10] because of its minimum Hamming distance properties as another test generation scheme.

A major difference between antirandom and RS based testing is that antirandom testing exploits the sequence in which the vectors are applied. With RS coding, only a fraction of the input combinations are code words, whereas with antirandom testing all combinations would eventually be applied, if one would choose to do so. Using RS codes has the advantage that algorithms and hardware designs for generation are already available [5]. Figure 3 shows an encoder shift register circuits for (15,9) RS code. In the output shift register part of Figure 3, *LSS* represents the least significant symbol, it is also the last byte transmitted. *MSS* represents the most significant symbol, corresponding to byte (n-1). It is the first byte transmitted. A RS code can only be generated for some specific code word sizes, however our experiments suggest that truncated code words have somewhat similar capabilities.

4 *Randomness* of antirandom test patterns

A scheme can be considered to be more *random* if the ones and the zeros are evenly distributed in space and time and if there is very little correlation between one pattern and the next. If we use this definition of randomness, antirandom and Reed-Solomon sequences turn out to be significantly more random than pseudo-random tests.

Let us compare the randomness of the first 30 test patterns for 14-input sequences generated using antirandom property, the Reed-Solomon code, and conventional pseudo-random tests, as given in Table 2. Pseudo 0 and Pseudo 2 represent the pseudorandom sequences starting with the seeds 00000000000000 and 01010101010101 respectively. The successive vectors in the time sequence are listed sequentially. Table 2, shows that the antirandom and Reed-Solomon sequences are more random than the two pseudorandom sequences. A more visual representation in Figure 2 suggests that the antirandom sequence is somewhat more random than the Reed-Solomon sequence. A black pixel indicates value 0, while the white pixel indicate value 1. The time progresses from top to bottom for each bit-map. There are several formal tests for randomness. Pradhan and Chatterjee [9] have shown that LFSR based sequences fail most of these tests.

5 Effectiveness of Antirandom and Psuedo-random Testing

To compare the effectiveness of antirandom and Reed-Solomon sequences with psuedo-random tests generated using linear feedback shift registers, (LFSRs), a series of experiments has been performed. We have measured the effectiveness of a test set using two coverage measures. The *stuck-at 0/1* coverage is the ratio of the number of stuck-at faults detecteded to the total number of faults. Some faults may be undetectable. We have also evaluated the *bridging fault* coverage in the IDDQ test environment. A bridging fault is assumed to be detected if the two nodes have

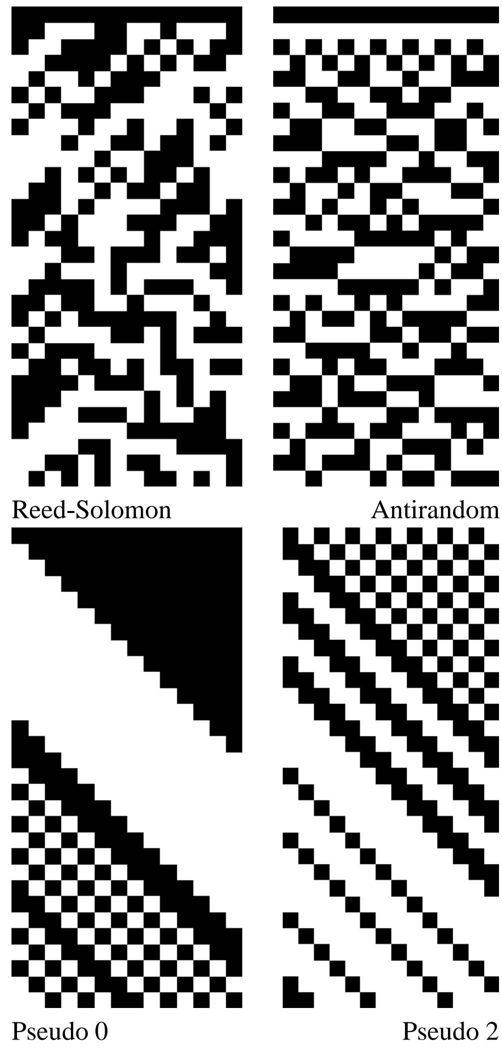

Figure 2: State-time diagrams of Reed-Solomon, Antirandom, Pseudo-random Pattern Generator

Antirandom	Pseudo 0	Pseudo 2	Reed-Solomon
00000000000000	00000000000000	01010101010101	00000000000000
11111111111111	10000000000000	00101010101010	00111001101110
01010101010101	11000000000000	10010101010101	01100001010100
10101010101010	11100000000000	11001010101010	11000010101001
00110011011000	11110000000000	01100101010101	10110100011111
11001100100111	11111000000000	00110010101010	01011000111010
01100110001101	11111100000000	10011001010101	10100011111101
10011001110010	11111110000000	11001100101010	01110110110110
00011110010011	11111111000000	01100110010101	11101100100101
11100001101100	11111111100000	00110011001010	11111011000111
01001011000110	11111111110000	10011001100101	11010101001011
10110100111001	11111111111000	11001100110010	10011010010011
00101101011110	01111111111100	11100110011001	00010111100010
11010010100001	00111111111110	11110011001100	00101110001100
01111000001011	00011111111111	11111001100110	01001111011000
10000111110100	10001111111111	01111100110011	11100100010010
00001111101001	01000111111111	10111110011001	11011101111100
11110000010110	10100011111111	11011111001100	10000101000110
01011010111100	01010001111111	11101111100110	00100110111011
10100101000011	10101000111111	01110111110011	01010000001101
00111100100100	01010100011111	10111011111001	10111100101000
11000011011011	10101010001111	11011101111100	01000111101111
01101001110001	01010101000111	11101110111110	10010010100100
10010110001110	00101010100011	11110111011111	00001000110111
00010001101111	10010101010001	01111011101111	00011111010101
11101110010000	01001010101000	10111101110111	00110001011001
01000100111010	10100101010100	11011110111011	01111110000001
10111011000101	01010010101010	11101111011101	11110011110000
00100010110111	10101001010101	01110111101110	11001010011110
11011101001000	11010100101010	10111011111001	10101011001010

Table 2: Comparison of randomness: Reed-Solomon, Antirandom and Pseudorandom

opposite logic values, because in such a case a very high quiescent supply current would indicate the presence of a fault.

We have simulated several ISCAS85 combinational benchmark circuits using Nemesis, a software tool developed at University of California, Santa Cruz [2].

The basic characteristics of the simulated ISACAS85 combinational circuit are given in Table 3. It list the detail information such as the number of inputs and outputs, number of gates, the circuit function and the total number of stuck-at faults.

5.1 Testing Combinational circuit for Stuck-at Fault

A pseudo-random test generator will generate different sequences depending on the seed used, and different seeds will yield different results. To see the variation in coverage due to the choice of the seed value, three different seed values were used for the pseudo-random test generator.

Figures 4 and 5 show the stuck-at fault coverage obtained by the test sequence generators for the circuits C880 and C3540. In these plots the x-axis represents the number of the test patterns, and the y-axis fault coverage. Tables 4, 5, 6, 7 show the fault coverage for different number of test patterns for c880, c3540, c1355 and c499 respectively. Note that the Pseudo0, Pseudo1 and

$g(x) = X^6 + X^5\alpha^{10} + X^4\alpha^{14} + X^3\alpha^4 + X^2\alpha^6 + X\alpha^9 + \alpha^6;$

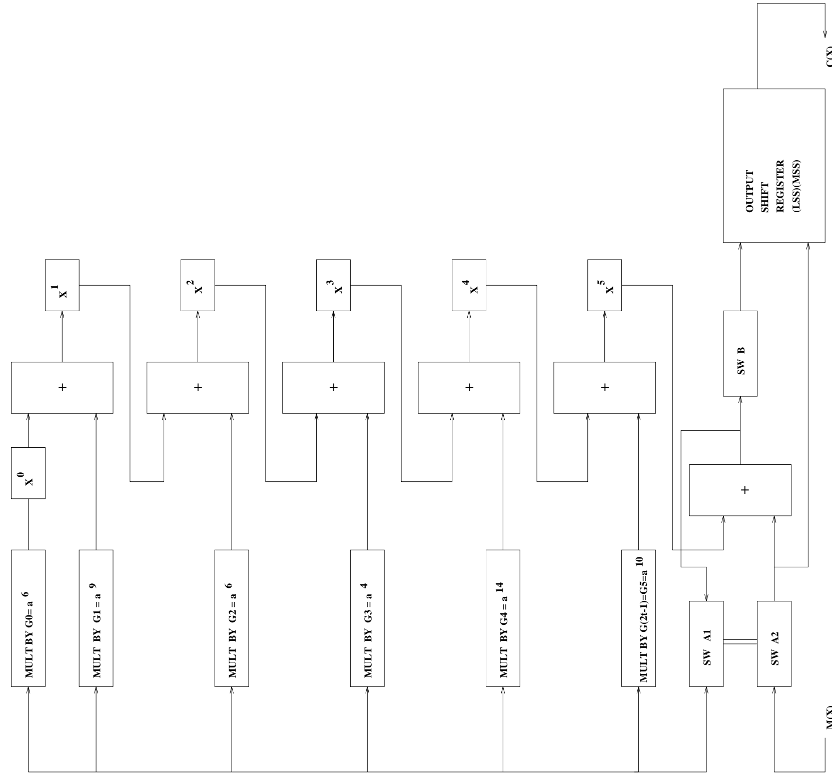

Figure 3: (15,9) Reed-Solomon Encoder shift register circuit

Pseudo2 in the figures and plots represent the different initial seeds for pseudorandom generator which are all zeros (000000...), all ones (111111...) and alternating bits (101010...).

For c880 (Figure 4), we observe that antirandom and Reed-Solomon tests perform similarly, obtaining 91.3% and 90.45% coverage respectively for 105 vectors. In both cases, the fault coverage curves rise sharply and exhibits a smooth behavior. In case of psuedo-random tests, there is a significant difference depending on the initial seed, the coverage obtained with 105 vectors ranges from 51.06% to 73.9%. For all three seeds, psuedo-random tests significantly lag in performance compared with antirandom and Reed-Solomon tests. We also observe that the plots for psuedo-random tests show somewhat irregular growth. The plot for c3540 (Figure 5) shows that antirandom testing is again the best, the difference between it and psuedo-random testing is most pronounced at about 70% coverage. Similar observations can be made for c1355 and c499 (Tables 6 and 7).

The results show that:

1. The antirandom and Reed-Solomon sequences generally provide higher coverage than psuedo-random testing.
2. Both new test pattern generation schemes can obtain a high fault coverage with significantly fewer test patterns compared to pseudorandom testing.
3. Often antirandom and Reed-Solomon sequences obtain similar coverage values, antirandom testing is generally slightly better.

These results show that the proposed schemes exercise the circuits-under-test better because they span the test vector space better than psuedo-random schemes.

Circuit	Number of Input	Number of Output	Number of Gates	Total Lines	Fanout stem	Circuit Function	Total stuck-at faults	Total bridging faults
c880	60	26	383	880	125	ALU and Control	942	3254
c3540	50	22	1669	3540	579	ALU and Control	3428	16459
c1355	41	32	546	1355	259	ECAT	1574	4422
c499	41	32	202	499	59	ECAT	758	2781

Table 3: Characteristic of simulated combinational circuits

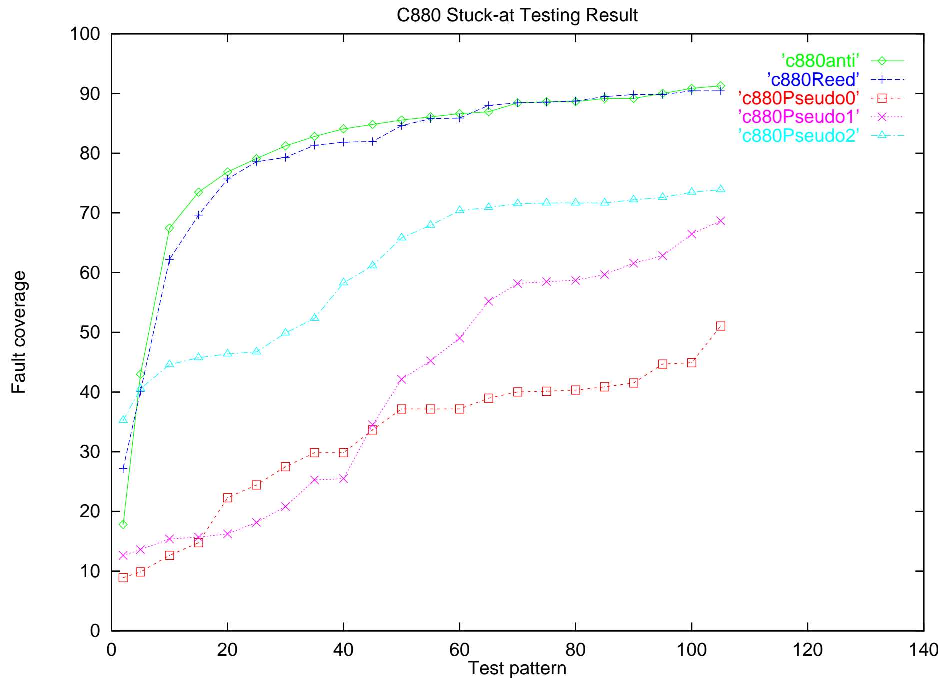

Figure 4: C880 Stuck-at Fault Coverage Simulation

5.2 Testing Combinational Circuit for Bridging Faults

A bridging fault is a short between two nodes in a circuit. We have applied the two new test pattern generation schemes to test bridging faults in the four ISCAS benchmark circuits. The bridging faults are identified by Carafe [4] by considering the layout information. Nemesis assumes that a bridging fault is being tested in the IDDQ test environment. If the two bridged nodes have opposite logic values then the bridging fault will cause a high value of IDDQ thus detecting the fault.

Figures 6 and 7 give the fault coverages obtained for the circuits c880 and c3540 for different schemes. Tables 8, 9, 10, 11 list the fault coverage values for specific number of test patterns for the four circuits.

Figures 6 for c880 show that the difference between the two proposed approaches and the traditional psuedo-random testing is quite remarkable. Further more with antirandom and Reed-Solomon sequences, the coverage of bridging faults rises much faster than that for stuck-at

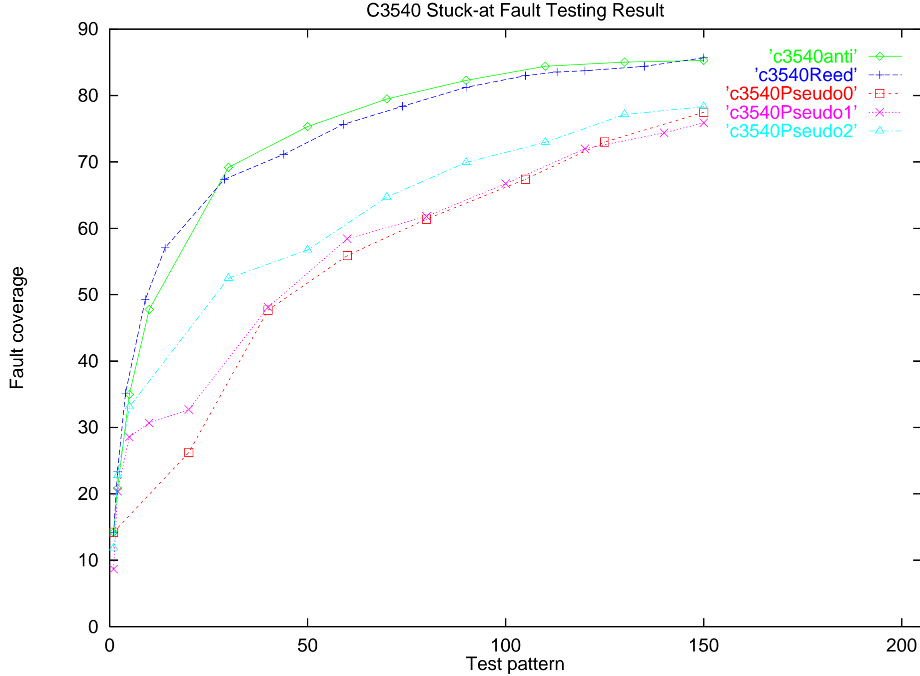

Figure 5: C3540 Stuck-at Fault Coverage Simulation

Test No.	2	5	10	15	25	35	55	75	105
Antirandom	17.83	42.99	67.46	73.46	79.09	82.8	86.09	88.64	91.3
Reed-solomon	27.18	40.13	62.21	69.21	78.56	81.32	85.77	88.54	90.45
Pseudo-0	8.917	9.87	12.63	14.76	24.42	29.83	37.15	40.13	51.06
Pseudo-1	12.63	13.59	15.39	15.71	18.15	25.27	45.22	58.49	68.68
Pseudo-2	35.24	40.55	44.59	45.75	46.71	52.34	67.94	71.66	73.9

Table 4: Stuck-at Fault Coverage for c880

faults. The gap between the curves for the new approaches and psuedo-random testing is wider for bridging faults. The same behavior is observed for other three circuits. For about 88-90% coverage obtained by our approaches, the gap for c880 widens from 17-49% to 27-75%. The similar gaps for c3540 widens from 2-3 to 22-43%, for c1355 from about (-0.5)-3 to 19-23%, and for c499 from 0.5-2.5 to 15.5-23%. This makes the proposed schemes ideal for generating tests for bridging faults in the IDDQ test environment. Using a very few vectors, very high coverage for bridging faults can be achieved.

An explanation for this improvement can be found by observing the fact that different inputs are toggled relatively independently with the proposed schemes, making it more likely that any two nodes will have opposite logic values.

6 Antirandom property at the output nodes

6.1 Functional Description

The antirandom testing approach derives its advantage from the fact that the bits applied at the primary inputs exhibit the antirandom property. However as signals propagate in a network and

Test No.	1	5	10	30	50	90	110	150	250	300
Antirandom	14.21	34.98	47.78	69.17	75.32	82.29	84.42	85.27	89.15	90.43
Reed-solomon	14.21	38.0	50.84	67.8	72.95	81.24	83.09	85.68	88.61	90.08
Pseudo-0	14.21	17.21	20.21	36.93	51.78	63.78	68.79	77.48	87.38	89.56
Pseudo-1	8.664	28.56	30.66	40.4	53.25	64.31	69.36	75.94	86.82	87.92
Pseudo-2	11.84	33.11	36.98	52.48	56.74	69.92	72.96	78.33	86.34	87.66

Table 5: Stuck-at Fault Coverage for c3540

Test No.	2	5	10	15	35	45	70	105	145	200
Antirandom	20.84	34.18	53.62	67.28	78.34	80.43	86.91	88.44	90.53	92.57
Reed-solomon	16.84	35.07	54.13	56.61	74.46	76.87	85.9	87.48	89.45	91.04
Pseudo-0	26.81	32.91	37.04	38.63	69.38	70.97	79.67	85.45	89.07	91.49
Pseudo-1	16.58	36.59	45.43	50.7	66.96	75.48	83.29	89.07	90.21	92.12
Pseudo-2	26.18	46.95	48.86	54.89	71.16	75.86	83.23	86.98	90.15	91.42

Table 6: Stuck-at Fault Coverage for c1355

are transformed, the *antirandomness* gradually dissolves away. We conducted an experiment to see if antirandonmess at the output nodes would also give high coverage. Specifically, we select input vectors to a circuit so that antirandom patterns appear at the output of the circuit. For this experiment, we selected the 74LS181 Arithmetic Logic Unit (ALU) because it has a well known functionality as given in Table 12.

The 74LS181 circuit is controlled by the four Function select inputs($S_0 ... S_3$), and a mode control input(M). These five signals control the 16 possible logic operations and 16 different arithmetic operations for either active HIGH or active LOW operands. Table 12 lists the functions 74LS181 perform with active high inputs and outputs function.

When the Mode Control Input (M) is high, all internal carries are inhibited and the device performs logic operation on the individual bits. When the Mode control input is low, the carries are enabled and the device performs arithmetic operations on the two 4-bit words. The device incorporates full internal carry lookahead and provides for either ripple carry between devices using the C_{n+4} output. or for carry lookahead between packages using the signals $\bar{P}$ (Carry Propagate) and $\bar{G}$ (Carry generate). $\bar{P}$ and $\bar{G}$ are not affected by carry in.

The 24 pins in this chip include 14 input lines and 8 output lines besides Vcc and GND.

To compare the effectiveness of different antirandom test approaches, we evaluated four testing schemes.

1. Directly apply antirandom test patterns to ALU's inputs.
2. Apply pseudo-random test patterns to ALU's input.
3. Apply test patterns such that the output logic values exhibit antirandom behavior.
4. Use a combined antirandom testing approach, using some vectors that are antirandom at the input and some that result in antirandom outputs. Since there are more inputs and putputs, we decided to altenate them in this way: 4 antirandom vectors (at input), 2 vectors with antirandom outputs, and continue repeating this sequence. We call this approach *mixed*.

The vectors to generate antiradom output vectors were obtained using the following method. We want to generate all $2^8 = 256$ output vectors, but in a sequence that satisfies the antirandom property. In order to reduce the complexity of the software needed to generate these output

Test No.	2	5	10	15	25	35	55	75	105
Antirandom	30.31	49.34	69.92	80.47	84.83	87.07	88.39	92.35	93.4
Reed-solomon	25.07	49.87	69.79	72.03	74.93	83.64	87.99	91.03	92.61
Pseudo-0	41.29	47.89	53.3	55.28	69.66	79.6	85.91	89.71	92.61
Pseudo-1	24.8	48.42	57.26	62.14	72.85	77.7	86.15	89.84	93.01
Pseudo-2	36.41	64.51	67.28	72.82	78.76	83.11	87.6	89.18	92.35

Table 7: Stuck-at Fault Coverage for c499

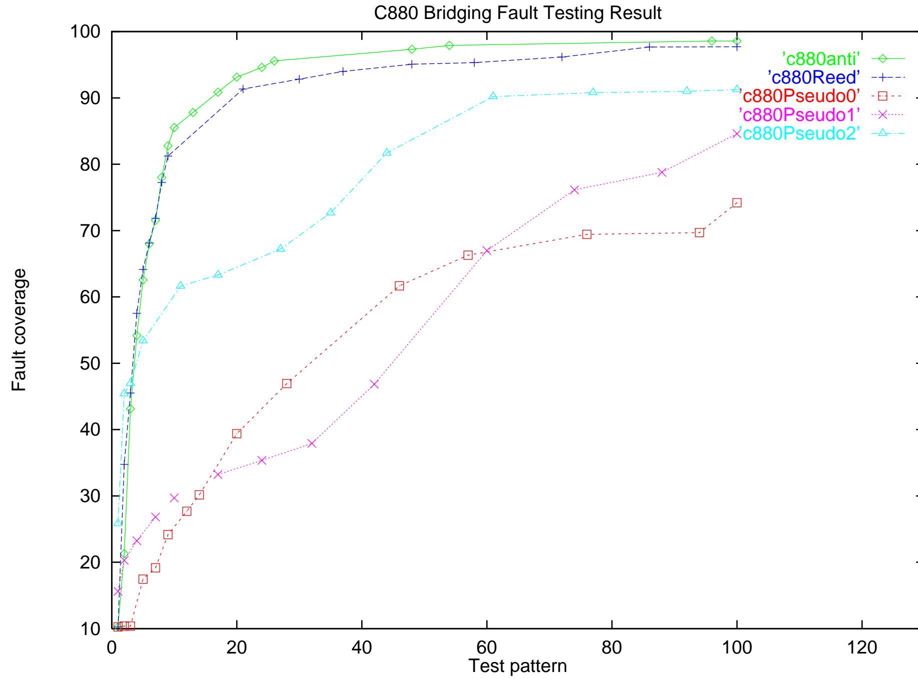

Figure 6: C880 Bridging Fault Coverage

vectors, we first obtained all elements of an 8-bit antirandom sequence. We then applied all possible psuedo-random vectors at the input and obtained the output vector for each of them by simulation. If this output pattern has not yet been generated, we save the corresponding input pattern in the appropriate position. If an input does not generate a new output vector, it is dropped. Eventually we obtain a sequence of input vectors that will generate the desired output sequence.

The Figure 8 and Table 13 shows stuck-at fault simulation results. We see that all three antirandom approaches show a sharper rise in coverage in the beginning compared with conventional psuedo-random testing. However at higher test coverage values, antirandom-at-output is not more effective than psuedo-random testing. Mixed antirandom testing is generally similar to antirandom-at-input. We were hoping to see that mixed antirandom testing is slightly better than antirandom-at-input approach. The results suggest that antirandom-at-output approach is not as effective as we were expecting. A possible explanation can be that antirandomness at the outputs may not translate into significant antirandomness at internal nodes. We plan to investigate other approaches for further enhancing the effectiveness of antirandom testing, possibly by using some structural infromation.

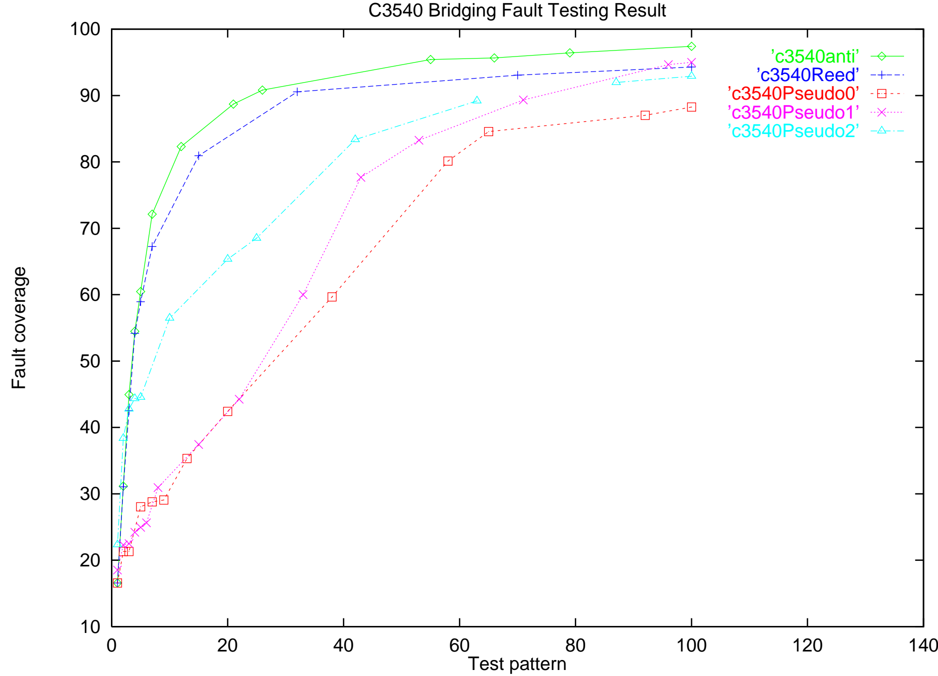

Figure 7: C3540 Bridging Fault Coverage

Test No.	1	3	5	7	15	25	55	75	105
Antirandom	10.26	43.13	62.55	71.54	89.325	95.03	97.91	98.24	98.58
Reed-solomon	10.26	45.51	64.14	71.84	86.28	92.00	95.25	96.49	97.7
Pseudo-0	10.26	10.38	17.46	19.17	31.69	44.11	65.46	69.27	75.85
Pseudo-1	15.6	21.79	25.05	26.82	32.23	35.68	51.91	76.16	87.13
Pseudo-2	25.84	46.96	53.4	56.11	62.74	66.42	87.19	90.72	91.35

Table 8: Bridging Fault Coverage for c880

7 Conclusions and future work

Antirandom testing is a new test generation approach. Here we have demonstrated it can achieve high fault coverage much faster than the conventional psuedo-random testing. It has also been successfully applied for software testing and testing of VHDL descriptions. Its effectiveness is specially remarkable for bridging faults. The scheme is well suited for IDDQ tsting because it provides very good coverage with only a few vectors. One possible way to exploit the capabilites of antirandom testing is to use it until a suitable high coverage is obtained, and then to switch to deterministic testing.

So far we have considered only *black-box* testing which assumes that we do not have any structural information available. It is possible to generate antirandom sequences that can exploit some structural information thus further increasing coverage, just like weighted psuedo-random testing [11] We are developing methods for applying this approach for sequential circuits. The sequential circuits are much less amenable to random testing because it takes a few clock periods for many faults to be detected. Pattern-holding based approaches, which have been proposed for random testing [8], may be applicable for antirandom testing also.

Test No.	1	4	5	7	15	25	55	75	105
Antirandom	16.57	54.47	60.46	72.12	84.44	90.39	95.42	96.22	97.51
Reed-solomon	16.57	54.17	58.93	67.25	80.93	86.53	92.05	93.27	94.37
Pseudo-0	16.57	21.32	28.05	28.79	37.33	47.21	77.06	85.46	89.40
Pseudo-1	18.52	24.2	24.96	28.29	37.44	48.54	84.775	90.744	95.083
Pseudo-2	22.34	44.35	44.55	48.35	60.91	68.50	87.23	90.57	93.47

Table 9: Bridging Fault Coverage for c3540

Test No.	1	4	6	15	40	65	75	85	105
Antirandom	10.83	42.71	68.97	89.12	96.12	97.08	97.31	97.31	97.66
Reed-solomon	10.83	53.31	68.97	87.57	94.82	97.24	97.62	97.75	98.04
Pseudo-0	10.83	36.37	50.01	70.34	90.12	95.26	95.97	96.45	97.10
Pseudo-1	18.29	30.34	33.02	66.51	93.61	95.55	95.97	97.17	97.52
Pseudo-2	24.88	41.56	48.01	70.34	91.72	95.24	96.03	96.45	97.04

Table 10: Bridging Fault Coverage for c1355

8 Acknowledgement

This work was supported in part by a BMDO funded project monitored by ONR.

References

[1] P. Bardell and W. McAnney and J. Savir, *Built-in Test for VLSI Pseudo-random Techniques*, John Wiley and Sons, 1987.

[2] *The Nemesis Manual*, C. Hall, B. Chess, T. Larrabee and H. Manley, Computer Engineering, University of California, Santa Cruze, 1995.

[3] W. R. Hamming, "Error Detecting and error Correction Codes", *Bell Sys. Tech. Journal*, April 1950, pp. 147-160.

[4] *Carafe User's Manual*, A. Jee, D. Dahle, C. Bazeghi and F. J. Ferguson Computer Engineering, University of California, Santa Cruze, 1996.

[5] W. A. Geisel, *Tutorial on Reed-Solomon Error Correction Coding*, NASA Technical Memorandum 102162, 1990.

[6] Y. K. Malaiya, "Antirandom Testing: Getting the most out of black-box testing," *Proc. International Symposium On Software Reliability Engineering*, Oct. 1995, pp. 86-95.

[7] Y.K. Malaiya and S. Yang, "The Coverage Problem for Random Testing," *Proc. International Test Conference*, October 1984, pp. 237-245.

Test No.	1	3	5	7	15	25	55	75	105
Antirandom	4.524	38.05	71.09	78.91	88.88	91.75	97.13	98.55	99.63
Reed-solomon	4.524	51.34	69	72.71	82.17	91.97	98.85	99.15	99.23
Pseudo-0	4.524	29.18	37.98	45.83	68.21	84.11	97.09	98.21	99.13
Pseudo-1	18.77	27.43	45.62	56.03	73.30	84.69	97.62	99.21	99.75
Pseudo-2	31.34	49.26	54.50	57.85	65.47	87.73	98.01	98.35	99.17

Table 11: Bridging Fault Coverage for c499

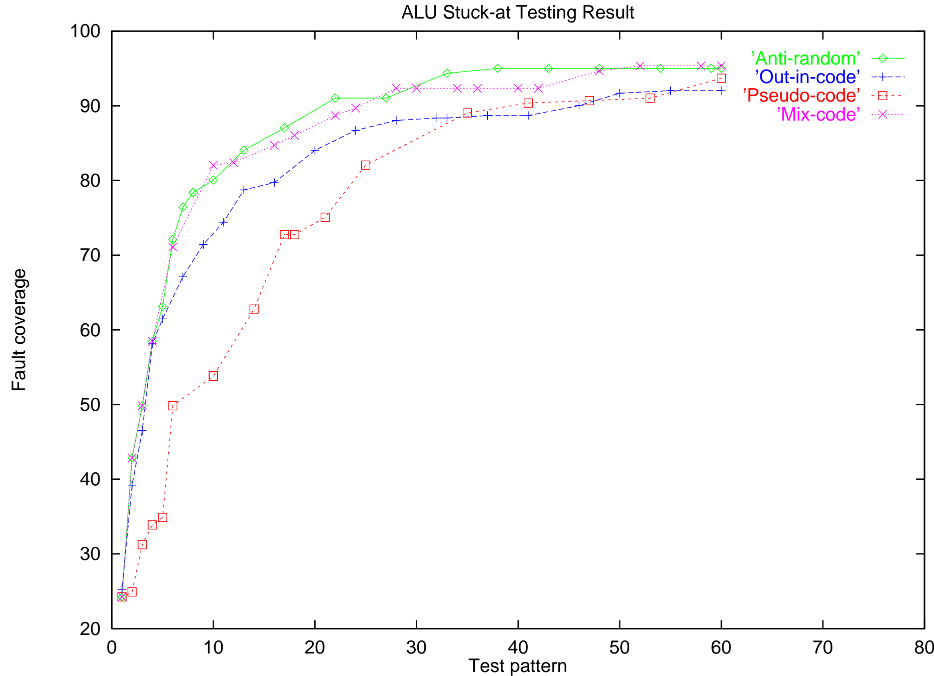

Figure 8: ALU Stuck-at Fault Coverage Simulation

[8] L. Nachman, K.K. Saluja, S.J. Upadhyaya and R. Reuse, "A Novel Approach to Random Pattern Testing of Sequential Circuits" *IEEE Trans. Computers*, January 1998, pp. 125-134.

[9] D. K. Pradhan and M. Chatterjee, "GLFSR-A New Test Pattern Generation for BIST", *Proc. International Test Conference*, 1994 pp. 481-490.

[10] I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fiels", *Journal of The Society for Industrial and Applied Mathematics*, June, 1960.

[11] N. A. Touba and E. J. McCluskey, "Altering A Pseudo-random Bit Sequence for Scan-based BIST", *Proc. International Test Conference* 1996, pp.167-175.

[12] S. Wu, "Effectiveness of Antirandom and Rood-Solomon Code based Testing", MS Thesis, Electrical Engineering Dept, Colorado State University, March 1, 1998.

[13] H. Yin, "Test Data Generation and Evaluation for Antirandom Testing with Checkpointing" MS Thesis, Computer Science Dept., Colorado State University, 1997.

[14] H. Yin, Z. Lebne-Denge and Y. K. Malaiya, "Automatic Test Generation using Checkpoint Encoding and Antirandom Testing" *Int. Symp. on Software Reliability Engineering*, 1997, pp. 84-95.

S_3	S_2	S_1	S_0	Logic $(M=H)$	Arithmetic $(M=L)(C_n=H)$
L	L	L	L	$\bar{A}$	A
L	L	L	H	$\overline{A+B}$	$A+B$
L	L	H	L	$\bar{A}B$	$A+\bar{B}$
L	L	H	H	Logical 0	minus 1
L	H	L	L	$\overline{AB}$	A plus $A\bar{B}$
L	H	L	H	$\bar{B}$	$(A+B)$ plus $A\bar{B}$
L	H	H	L	$A\oplus B$	A minus B minus 1
L	H	H	H	$A\bar{B}$	AB minus 1
H	L	L	L	$\bar{A}+B$	A plus AB
H	L	L	H	$\overline{A\oplus B}$	A plus B
H	L	H	H	B	$(A+\bar{B})$ plus AB
H	H	L	L	AB	AB minus 1
H	H	L	H	Logical 1	A plus A^*
H	H	H	L	$A+\bar{B}$	$(A+B)$ plus A
H	H	H	H	$A+B$	$(A+\bar{B})$ plus A
H	L	L	L	A	A minus 1

Table 12: 74LS181 active high inputs and outputs function table

Test No.	1	3	5	7	10	22	38	75	115
Antirandom	24.25	49.83	63.12	76.41	80.07	91.03	95.02	95.68	95.68
Out-in	25.25	46.51	61.46	67.11	72.93	85.4	88.7	93.69	95.68
Mix	24.25	49.83	64.72	76.6	82.06	88.7	92.36	95.35	96.01
Pseudo-random	24.25	31.23	34.88	51.83	53.82	78.58	89.69	95.02	96.01

Table 13: Simulation Result of Bridge Fault for ALU