Pseudorandom,Weighted Random and Pseudoexhaustive Test Patterns Generated in Universal Cellular Automata

Ondrej Novak

Technical University Liberec, Hálkova 6, 461 17 Liberec I,
Czech Republic
tel.: + 420 48 53553460, fax: + 420 48 5353112 e-mail: ondrej.novak@vslib.cz

Abstract. The paper presents a design method for Built-In Self Test (BIST) that uses a cellular automaton (CA) for test pattern generation. We have extensively studied the quality of generated patterns and we have found several interesting properties of them. The first possibility how to use the CA is to generate pseudoexhaustive test sets as the CA can generate code words of codes with higher minimal code distance of the dual code. There is no need of reseeding the CA in order to generate all the code words. This type of test set can be advantageously used for testing with low number of inputs and low size of cones in the circuits under test (CUT). The proposed CA can also generate weighted random patterns with different global weights which can be used instead of linear feedback shift register (LFSR) pseudorandom sequences, the fault coverage is higher. It can also be used as deterministic pattern compactor in mixed mode testing. The generated sequence can be also easily used for testing CUTs with input-oriented weighted random patterns. The CA is formed by T flip-flops and does not contain any additional logic in the feedback. We proposed a new scheme of BIST where the CA is a part of a modified scan chain. Several experiments were done with ISCAS 85 and 89 benchmark circuits. We compared the quality of the generated test patterns with the quality of the patterns generated in an LFSR .

Key words: Cellular automata, BIST, linear cyclic codes, linear feedback shift registers, hardware test pattern generators, weighted random testing, pseudoexhaustive testing

1 Introduction

Built-in self-test (BIST) is a concept useful for testing the VLSI circuits, where it solves the problem of limited access to the circuit-under-test (CUT), offers on-line and in-line applicability of the test sets and very radically reduces the amount of output information - see e.g. **[16]**. To implement BIST, we must embed both the test pattern generator (TPG) and output data compactor into the structure of the CUT which naturally imposes limits on their size, complexity and level of control which may lead to a loss of fault coverage. Our task was to find BIST structures and their function algorithms, which will guarantee the required level of fault coverage when observing the simplicity requirements.

The techniques for hardware test pattern generation can be classified in the following groups: pseudoexhaustive testing **[22]**, pseudorandom testing **[1],** weighted random testing **[22]**, deterministic tests **[5]** and mixed mode pattern generation **[13], [12].**

A pseudoexhaustive or (*n*,*w*)-exhaustive test set is a sequence of n-bit input vectors which exercises every w-bit subset of CUT inputs feeding one output with 2^w different binary patterns. It can be generated e.g. by a built-in test pattern generator using a linear feedback shift register (LFSR) **[22]** or cellular automaton (CA) **[18].** It was proved by simulation of the ISCAS benchmark circuits, that the effectiveness of the pseudoexhaustive testing is limited on the cases of CUT with small number of inputs and small values of *w*.

The mixed mode pattern generation consists in generation of a given number of pseudorandom patterns and after it in exercising the CUT with deterministic test vectors. The deterministic vectors have to detect random resistant faults.

It was shown that for a substantial part of designed circuits it is practically impossible to detect all faults by a pseudorandom test set. One way how to improve the fault coverage and or to reduce the number of generated test patterns is to generate weighted random patterns. There exist a relatively large number of proposals **[2, 17, 23, ...]** how an optimized set of weights can be determined such that the required number of random patterns is minimized. Usually the input-oriented weight computation is used, the resulting test set has different probabilities of zeros and ones for each CUT input. This approach is very efficient but in the case of hardware test pattern generation it demands quite a lot of additional hardware .

In **[15]** it was shown that it could be very efficient to calculate and apply global weights for weighted random testing (pattern oriented WRT). By the term global weight we mean the ratio between ones and zeros in a test pattern. This kind of testing means that the weights are not input-oriented but they are common for CUT inputs. During testing there are generated several pattern subsets with different weights. This approach provides lower hardware overhead and shorter computational time which is necessary for estimating the optimal weights. The method uses single LFSR for generation of patterns with the probability of ones equal to 0.5, weight computational block for deriving patterns with modified weights and a multiplexer which switches between patterns with different weights. The multiplexer is controlled by a counter which enables us to generate in one test set patterns with different weights. The output of the multiplexer feeds the scan chain of a CUT.

Cellular automata (CA) can be seen as a tool for decomposing a system into very simple elementary units (cells) which are very well suited for implementation in VLSI circuits, above all due to the uniformity of the cells and due to the locality of connections **[3]**. Among the recently identified fields of their application, the test pattern generation plays an important role.

In **[24]** the CA formed by T flip-flops was designed. It generates primitive polynomial code sequences for the polynomials which have the form $p(x) = 1+ x^{n-1} + x^n$. In **[8]**, there is introduced a method how to transform the primitive polynomial LFSR sequences into sequences with the same period which can be generated with the help of cyclic CA which can be formed by D and T flip-flops without any combination logic between the flip-flops. These CA could be used as pseudo random test pattern generators.

The cellular automata were proposed to be used in BIST in **[11]**. It is shown there, that the random properties of the CA outputs are better than those of LFSRs. CA were used in **[6]** for testing circuits with boundary scan and in **[7]** for testing the sequential circuits. Several experiments with the benchmark circuits were done, the fault coverage was higher for the case of CA, the testability of delay faults was higher than in the case of using LFSRs.

In this paper we introduce a new universal scheme which can be used both for pseudoexhaustive, weighted random pattern generation and compressing the deterministic patterns. Instead of using an external LFSR for pattern generation we modify the scan chain in such a way that it forms a CA. The CA can be used for generating code words of chosen code with greater minimal code distance of its dual code or for generating non code words. The CA can generate all code words after seeding with one seed only, usually it can generate the same number of non code words after seeding with one non code seed, too. We have verified that the quality of generated test patterns is better than the quality of the patterns generated in the LFSR with primitive polynomial.

In the following section the design of the CA will be introduced. An analysis of the CA parallel output sequences can be found in section 3. In section 4, the proposed arrangement of BIST using the CA is presented. In section 5, experimental data underline the efficiency of the proposed method. In section 6 we compare hardware overhead for LFSR and CA test pattern generation. In section 7 we summarize the results.

2 Linear Code Based Test Pattern Generation

In the following we will be referring alternatively to a vector or code word or polynomial, taking into account that every vector $r = (r_0, r_1, ..., r_n)$ can be associated with a polynomial $r(x) = r_0 + r_1x + ... + r_nx^n$.

An m-bit LFSR serially generates code words of a cyclic code N=(n,k), $k=n-m$. If the minimal code distance of the dual code is d_{min}, then the code words of the code N form an (n,w)-exhaustive test set, where $w = d_{min}-1$, the LFSR has to be seeded with several seeds, see **[20].** The problem of great number of seeds for $d_{\mathrm{min}}>3$ is solved by using an n-stage LFSR in **[22].** This method is based on the use of an n-bit LFSR which has a feedback connected in accordance with a polynomial $g(x) = p(x)c(x)$, where $p(x)$ is a primitive polynomial of degree m and $c(x)$ is a generator polynomial of an (n,k) code.

When an *(n,k)* code word is used as a seed, all code words are generated during one LFSR period. Another solution of this problem was shown in **[9]**, using only an *m*-stage LFSR with additional non-linear feedback. The LFSR has a linear feedback connected according to a non-primitive irreducible polynomial with roots of n^{th} order. The non-linear feedback enables after finishing an LFSR cycle to add two code words of just generated code cycle. The modified LFSR can in some cases generate after being initiated with a nonzero seed all 2^m-1 different nonzero code words.

Further we develop another method of linear code generation. This method does not use any XOR in feedback and all the code words of a code with non primitive but irreducible polynomial can be generated in a CA after seeding with one vector only.

2.1 Properties of code sequences generated in LFSRs

As a first step, let us consider an *m*-bit serial LFSR with the XOR gates in the feedback which performs division by an irreducible polynomial *g(x)* of the degree *m* (An example for *m*=8 is given in Fig.1).

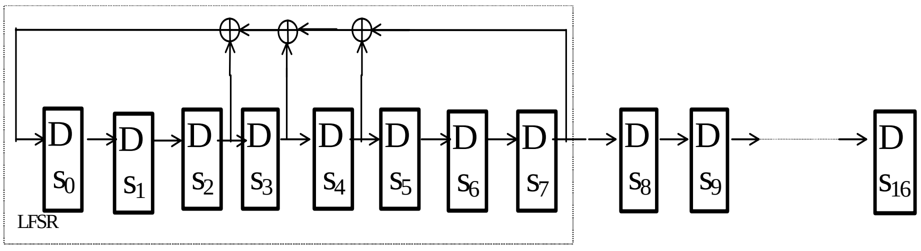

Fig. 1. An 8-bit LFSR created by D flip-flops with XOR gates in the feedback which performs division by the non-primitive irreducible polynomial g(x) = $x^8+x^5+x^4+x^3+1$. The LFSR generates code words of the (17,8) code. The first 8 bits of the code words are generated in the LFSR latches (denoted as S_0-S_7) the next 9 bits can be obtained by shifting into the adjacent shift register (denoted as S_8-S_{16}). The LFSR can generate 15 disjoint code word cycles, each having the period of 17.

The feedback is connected according to reciprocal polynomial $g^*(x)=x^m g(1/x)$, *m-n* latches are added as shift register. The LFSR generates code words of a code N=*(n,k)*, *k=n-m.* The same code words can also be generated by an *n*-bit LFSR which performs cyclic shifting of one code word (Fig.2). However, to generate the whole code, we must use $c=(2^m-1)/n$ different initial states (seeds), because the LFSR states are distributed within *c* disjoint *n*-state nonzero cycles and one all-zero cycle of the length one. In the case of a primitive polynomial *c=1.*

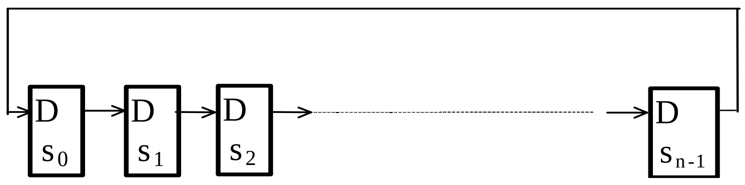

Fig. 2. A n-bit LFSR without XOR gates which after loading with a code word as a seed can be used for generation of a code word cycle of the length n.

The code words can be considered as multiples of the parity-check polynomial $h(x) = (x^n-1)/g(x)$. Let us suppose that the initial state of the LFSR is $a(x)\ h(x)$ where $a(x)$ is an arbitrary polynomial. Then the ith state can be obtained as $x^i a(x)h(x)$.

Define T as a matrix that contains as rows all the nonzero code words of the code N. The rows of this matrix correspond to parallel outputs of the LFSR (Fig. 2). If we generate the code words with an LFSR with a non-primitive polynomial, we have to seed the register c times. This means that every nth row will correspond to a new seed, whereas the following n-1 rows correspond to LFSR shifts.

$$T = \begin{bmatrix} a_0(x)h(x) \\ xa_0(x)h(x) \\ \dots \\ x^{n-1}a_0(x)h(x) \\ a_1(x)h(x) \\ \dots \\ x^{n-1}a_1(x)h(x) \\ \dots \\ a_{c-1}(x)h(x) \\ \dots \\ x^{n-1}a_{c-1}(x)h(x) \end{bmatrix}$$

The total number of rows is equal to $cn = 2^m-1$. The polynomials $a_i(x)$ are chosen in such a way that the seeds $a_i(x)h(x)$ initiate disjoint LFSR cycles.

2.2 Finding a primitive element of a field

We know from the theory of the fields, that all codes generated by irreducible polynomials form a field and that every field has at least one primitive element. In case of a primitive polynomial $g(x)$, the primitive element is x. For other cases we can find some other primitive elements. The second polynomial of degree 1 is *x+1*. It is the simplest polynomial worth testing whether it is a primitive element or not. In the following we will work with codes with non primitive polynomials for which x+1 is a primitive element. Define ***T'*** as a matrix that contains as rows all nonzero code words with a code word $a_0(x)h(x)$ as a seed. All code words can be obtained by multiplication of any of the rows by the powers of primitive element *x+1*.

$$T' = \begin{bmatrix} a_0(x)h(x) \\ (x+1)a_0(x)h(x) \\ \ldots \\ \\ (x+1)^{cn}a_0(x)h(x) \end{bmatrix}$$

2.3 Design of a CA

The automaton performing the multiplication by *x+1* is given in Fig. 3. If the automaton is seeded with a code word, the outputs of the D flip-flops correspond after every clock period to an (*n*,*k*) code word from the matrix T´. The automaton is an additive cellular automaton (CA) with the rule 60 for each cell **[3]**, having a regular linear structure.

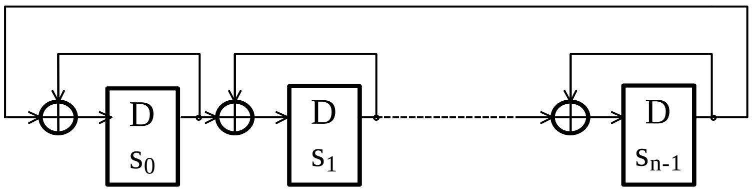

Fig. 3. Cellular automaton created from D flip-flops performing multiplication of the polynomials corresponding to code words by the polynomial x+1.

This CA can be further simplified. Instead of the D flip-flops and local feedback taps with XORs we can use T flip-flops because the T flip-flop performs a XOR function of its own state and the input signal in each clock period , the result is stored as a new internal state. Thus the CA given in Fig.4 has the same function as the CA in Fig. 3, but the hardware realization is simpler.

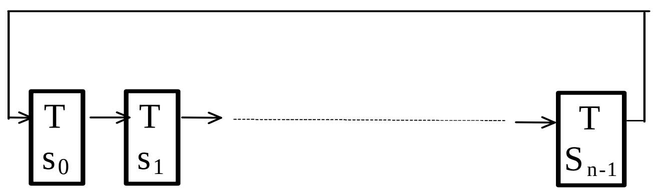

Fig. 4. Cellular automaton created from T flip-flops performing multiplication of the polynomials corresponding to code words by the polynomial x+1.

We have studied the state transition diagrams of the proposed CA. If we consider a CA with the length equal to a such code length that the corresponding polynomial can be irreducible, we obtain the state transition diagram with $2^{n-k}–1$ periodic states, 2^{n-k} "garden of Eden" states and one "graveyard" state. An example is given in Fig. 5.

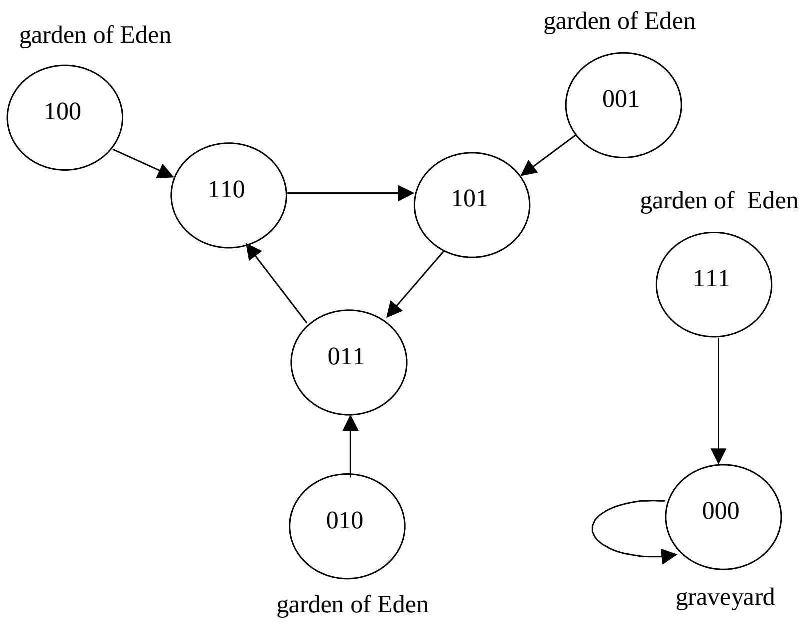

Fig. 5. State transition diagram of the 3 bit CA. The states with even number of ones are the code words of the (3,1) code

As far as we know, the proposed CA is the simplest possible automaton which can generate all code words of codes with non primitive polynomials with one seed only. It is universal in the sense that it can generate all possible code sequences of the given length without any hardware modification, we can also generate non code patterns which correspond to XOR of two or more code words of different codes. There is also possible to generate cyclic sequences of non code words, the period is usually the same as in the case of seeding the CA with a linear combination of code words. It is also possible to find seeds for which the CA has non periodic behavior.

3 Analysis of the patterns generated in CA

We have studied the properties of patterns generated in the proposed CA. There are substantial differences between the properties of the patterns generated in a LFSR and a CA. We have studied the possibility of generating code sequences, parts of code sequences with reducible characteristic polynomials, non-code sequences and possibility of test pattern compression.

3.1 Irreducible polynomial code sequences

The proposed CA can generate an (n,w) exhaustive test set after seeding with one code word seed only , $w <$ minimal code distance of the code dual to the generated code. It was shown in [4] that it is not necessary to generate all the code words, we can use only a part of them. We have studied the quality of pseudoexhaustive test sets in **[10]**. Here we demonstrate the results in the left columns of the Table 1.

3.2 Reducible polynomial code sequences

If we seed the CA with a pattern which is equal to the sum mod. 2 of words of (n,k) codes with different polynomials we can generate a sequence with the same period as in the case of seeding with a code word. Depending on the chosen polynomials the generated sequence could be a part of a BCH code. The minimal code distance can be greater. We have verified that even after seeding the CA with one seed only we obtain pseudoexhaustive test set with better parameters. The comparison is given in the Table 1.

3.3 Test sequences with different weights

Let us suppose that we feed the serial output of an m bit LFSR to the s bit scan chain, $s>m$. The weights of the patterns from a LFSR with arbitrary primitive characteristic polynomial are very closed to the value $s/2$, where s is equal to the scan chain length. An example is given in Fig. 6. The weights are not distributed symmetrically around s/2. This is caused by seeding (resetting) the scan chain flip flops before testing to the zero states. .We can see that that the initial reset of the scan chain flip-flops has only a negligible influence on the weight rate.

	CA generating code words			CA generating mod 2 sums of code words		
	(n,w)-exhaustive test set	Generator polynomial (octal form)	No. of test patterns	type of (n,w)exhaustive test set	used polynomials (octal form)	number of test patterns
8	(17,4)	471	255	(17,5)	727, 471	255
11	(23,6)w	5343	2047	(23,7)	5343, 6135	2 047
12	(65,5)	10761	4095	(65,7)	10761, 13535, 111763	3 542
20	(41,8)	6647133	1048575	(41,9)	6647133, 5327265	41 943

Table 1. Comparison of pseudoexhaustive test set parameters for CA generating all code words of a code with $d_{min}>3$ and the same CA with modified seed. The CA were seeded only once with a mod. 2 sum of selected code words of the given codes. The last column shows the lowest sufficient number of test patterns which create a pseudoexhaustive test set.

	Number of test patterns		
(n,w)-exhaustive test set	Use of incomplete LFSR period **[4]**	method of constant weights [**21]**	proposed CA
(31,3)	42	62	31
(63,3)	64	126	58
(127,3)	93	254	67
(17,5)	382	272	255
(23,7)	3 302	3 542	2 047
(65,7)	2 579130	87 360	3 588
(41,9)	695 860	202 540	41 943

Table 2. Comparison of test lengths for different test methods and given test parameters

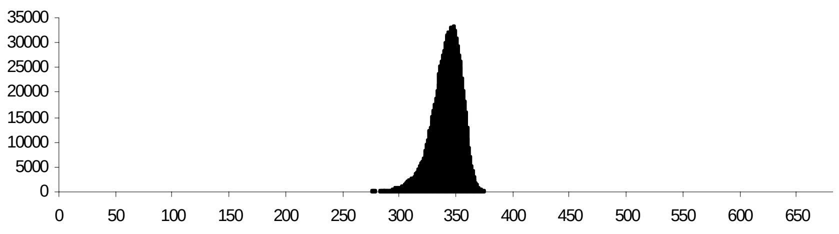

Fig. 6. The rate of global test pattern weights. The patterns were generated in a 23 bit LFSR with a primitive characteristic polynomial. The LFSR output was serially fed to the scan chain of the length 683 bits. X axis - pattern weights, Y axis - the number of test patterns which have the given weight . The number of generated test patterns:1 000 000. The scan chain flip-flops were set to log 0 before testing.

We have compared the weights of the LFSR patterns with the weights of CA patterns. The generated code was chosen in such a way that the number of information bits was the same as in the case of LFSR. In the Fig. 7 we can see the weight rates of code words generated in the CA. The polynomial of the code was non primitive and irreducible. In the Figures 8 and 9 we can see weight rates with different non code seeds. We can conclude that depending on the seed we can generate patterns with different weight rates.

Let us define a weight of one output sequence to be a ratio between the number of ones and zeros on the output within the test period. The weight during the test sequence is approximately the same for each of the CA parallel output. We have experimentally verified this fact for each of the 683 bit CA outputs. The weight depends on the CA seed properties only. This is demonstrated in Fig. 10. The different seeds caused that the weights were on all the different outputs equal to 0,5 for the first seed ; 0,4 for the second seed and 0,1 for the third seed.

3.4 Deterministic test pattern compaction

The CA can be used for compaction of deterministic test patterns which will test random, pseudoexhaustive and weighted-random resistant faults in mixed mode testing similarly to **[12]**. In this scheme we suppose to store seeds for the CA in a memory. After a given number of clock periods we get the demanded test cubes on CUT inputs. It is possible to store only the first k bits of the CA the rested bits are set to one or to zero. We have done several experiments which show that the CA is an efficient tool for decompressing such stored seeds of the deterministic test patterns, the efficiency is similar or better than it is for an LFSR.

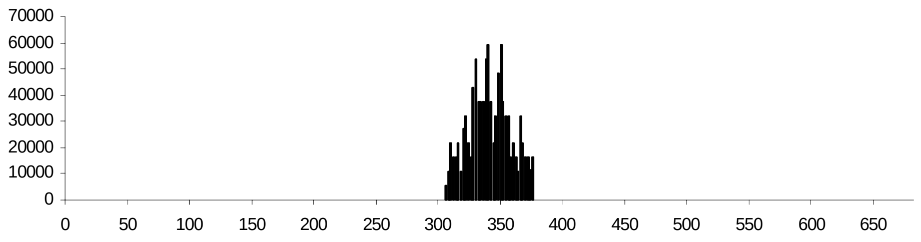

Fig.7 The rate of global test pattern weights. The 683 bit CA generated code patterns. X axis - pattern weights, Y axis - the number of test patterns which have the given weight. The number of generated test patterns:1 000 000

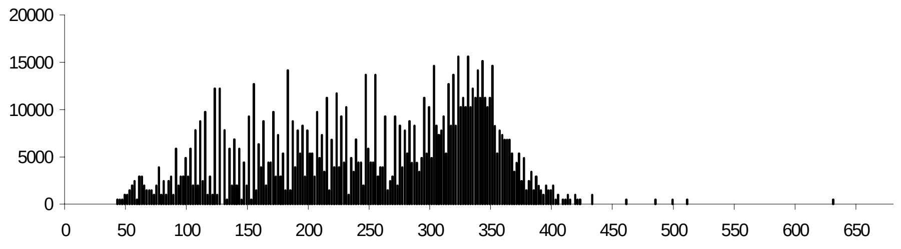

Fig. 8. The rate of global test pattern weights. The 683 bit CA generated non code patterns.X axis - pattern weights, Y axis - the number of test patterns which have the given weight . The number of generated test patterns:1 000 000.

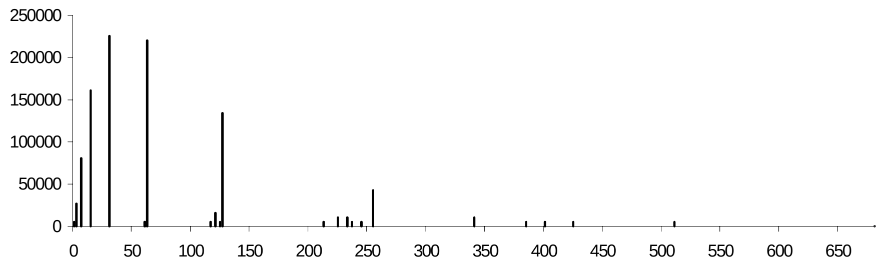

Fig 9. The rate of global test pattern weights. The 683 bit CA generated non code patterns. The seed was different from that one in Fig. 5. X axis - pattern weights, Y axis - the number of test patterns which have given weight. The number of generated test patterns:1 000 000.

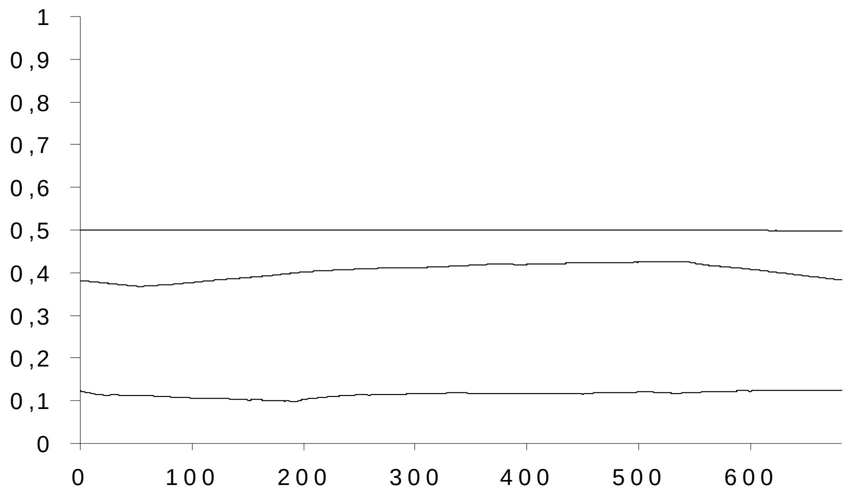

Fig. 10. Weights for each of the CA parallel output in the graph weights for three different seeds are plotted . X axis - position of a bit in the CA, Y axis - relative number of ones in the set of generated words. Total number of generated patterns 1 000 000. If we use a code word as a seed we obtain the ratio between ones and zeros approx. equal to 0.5, if we use specific non code seeds we obtain the ratio approx. equal to 0,4 and 0.1 respectively.

3.5 CA as a fast counter

The CA can be used as a fast synchronous counter. There is no additional delay of signal propagation between the flip-flops. The delay caused by the global feedback can be eliminated by arranging the CA into a ring similarly to **[8]**. We have designed a 17 bit CA, which after seeding with the vector 300000 (oct.) generates 256 different states. The period of the counter is equal to 255, the vector used as a seed is a “garden of Eden” state. Nowadays, we optimize the counter topology in different IC design technologies.

4 Using the CA in BIST

We can use the CA in BIST in several different ways: We can generate an (n,w) exhaustive test set according to Tab. 1. We can choose such a seed of the CA that we can consider the sequence to be pseudorandom (the test patterns weight rates similar to those shown in Fig. 7, the probability of ones on each CA output approximately equal to 0.5 during the test as it is shown in Fig. 10). We can choose such a seed of the CA that the patterns generated on the parallel CA outputs have different global weights (Fig. 8 or 9). If we modify the CA according to Fig. 12 we can use the inverted outputs for stimulating the CUT inputs with complement weights.

Example: For a specific seeds of the 683 bit CA we obtained the global weights 0.5, 0,4 and 0.1. If we use the inverted outputs, we can simultaneously stimulate arbitrary selected CUT inputs with the weights 0.6 and 0.9.

In order to use the CA in BIST we have to do several practical steps:

1) Choose an (n,k) code, $n \geq$ number of CUT inputs, $2^{n-k} - 1 \geq$ considered test sequence length. In order to keep the period of CA maximal it is necessary to choose n equal to the length of some code with irreducible polynomial. For our experiments we have chosen the codes with lengths 17, 23, 25, 41, 47, 55, 69, 267, 765, 1687, ... **[19]** .
2) Verify whether the polynomial x+1 is either a primitive element of the corresponding field or it guarantees the period et least equal to the test sequence length . Otherwise we have to chose another characteristic polynomial of the (n,k) code .

3) Verify whether the selected code or non code sequence is suitable for testing the CUT. We have either to simulate fault coverage of the test sets for the seeds with different properties or we have to estimate the optimal weights of test patterns **[15, 23]** .

One possibility of using the CA in BIST is shown in Fig. 11. Let us suppose that the CUT is designed in accordance with Scan Design methodology with modified Boundary Scan cells in such a way that the D flip-flops in the input part of scan chain are replaced by T flip/flops . We can also determine whether we will stimulate any of the CUT inputs with inverted CA output or with the non inverted one. We separate the input and output part of the chain and we complete the CA with the feedback. We suppose to use the CA with the number of flip-flops equal to or greater then the number of CUT inputs. If there are internal flip-flops in the CUT we have to include them into the scan chain and we have to avoid influencing the CUT inputs by the CUT outputs in the testing mode.

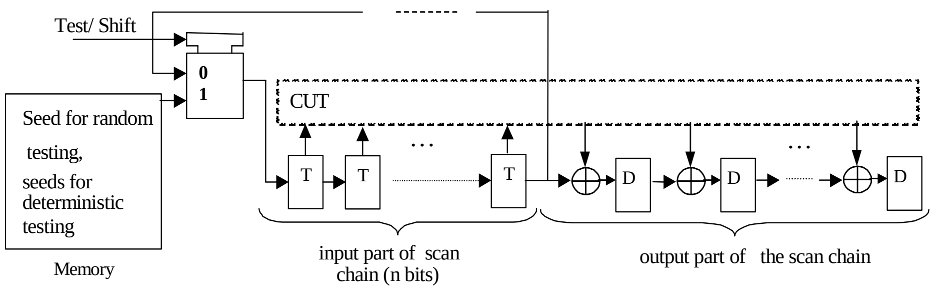
Test/ Shift
0
1
Seed for random testing, seeds for deterministic testing
Memory
CUT
T
T
T
D
D
D
input part of scan chain (n bits)
output part of the scan chain

Fig. 11. BIST scheme

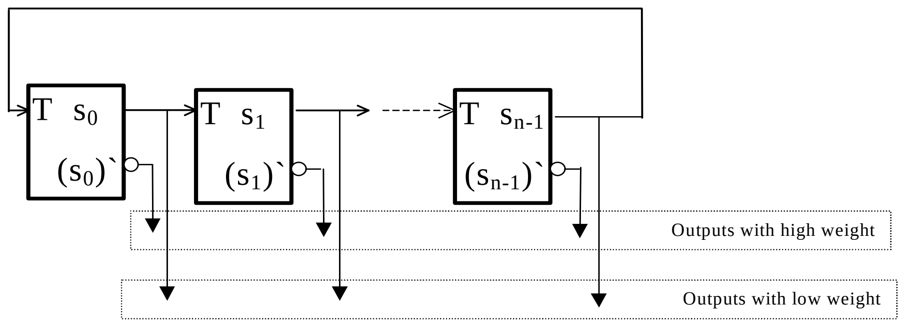

Fig. 12. Modified CA created from T flip-flops with non inverted and inverted outputs.

This can be done by any known technique similar to BILBO **[14]**. The testing starts after resetting of all flip flops and seeding the CA from the memory. At each clock cycle the CA generates a new pattern, and the CUT responds are compacted in the output part of the chain. After performing a given number of test steps the deterministic test patterns are shifted into the chain and the CUT responses are compacted in the output part of the chain.

We have to keep in mind that the seeds which are stored in the memory have to be modified for both parts of testing in such a way that after shifting into the scan chain the desired test cube would be present at the CUT inputs.

5 Results Obtained

We have chosen ISCAS 85 and ISCAS 89 benchmark circuits with a significant number of random pattern resistant faults. The internal flip-flops were considered to be CUT inputs. We checked by a simulation the number of non detected detectable faults both for a 32 bit LFSR with a primitive polynomial and for the CA. The polynomial of the 32 bit LFSR was randomly chosen from the list of primitive polynomials. The seeds of the CA code and non code words were randomly selected within code and non code words. For every circuit we have done 8 experiments with different LFSR sequences and 8 experiments with CA sequences. In further research we will use the method described in [23] for calculating optimal pattern weights for the CUT inputs, we expect to obtain better results. The simulated results are given in Tab. 3.

circuit	length of test cubes	32 bit LFSR : No. of undetected faults after 10 000 generated patterns	CA: No. of undetected faults after 10 000 generated patterns
s 641	54	12	7
s 713	54	11	7
s 820	23	9	6
s 832	23	17	5
s 953	45	10	3
s 1238	32	10	11
s 5378	214	38	27
s1196	32	17	11
s 13207	700	624	472
s 9234	247	648	602
s 9234.1	247	681	642
s 15850	611	605	565
s 953	45	10	5
c 2670	157	304	292
c 7552	206	324	130

Table 3. Comparison of the numbers of faults which remain undetected after 10 000 test patterns. In the case of the LFSR we have chosen 32 bit LFSRs with primitive polynomial. The length of CA is equal or higher than the number of transformed circuit inputs.

6 Hardware overhead

We have designed the CA in MIETEC 2.4 μm CMOS technology. In this technology we have the following sizes of the flip-flops:
D flip-flop with set or reset – 180μm x 106 μm
T flip-flop with set – 170μm x 106 μm
T flip-flop with reset – 190μm x 106 μm
XOR gate – 80μm x 106 μm.
If we compare the solution of T flip-flop CA with asynchronous set from Fig. 2 with the simple chain formed by D flip-flops with set or reset we can see that the used chip area will be similar. If we compare the solution of T flip-flop CA with the D flip flop CA we can see that we use only about 65 % of silicon for the flip-flops and gates, we also spare some area because of simpler routing.

7 Conclusion

A linear cellular automaton which can be used as a TPG for BIST was designed. Its main features are its simple structure and high fault coverage. The simplicity of the structure is due to the existence of only one feedback with no XOR. The cyclic CA can be implemented in the CUT by replacing the D flip flops in the scan chain with the T flip/flops and by adding a feedback from the last to the first flip-flop. There is no hardware overhead necessary for this replacing. The properties of the generated patterns depend on the seed of the CA. No additional hardware changes have to be done when we want to change the rate of global weights in generated patterns.

We compared the properties of the CA with the properties of a LFSR with a primitive characteristic polynomial which is usually used for pseudorandom test pattern generation. The CA has the following advantages:

- no hardware overhead, the whole TPG can be built by converting the existing scan chain
- depending on the seed the CA can generate weighted pseudorandom test sets with different rates of weights, we can individually modify weights for any of the CUT inputs by using inverted or non inverted CA outputs
- the fault coverage for ISCAS benchmark circuits is better than it is for an external LFSR.

Disadvantages:

- all flip-flops have to be reset-able, a seed for the CA has to be calculated in more complex manner than for an LFSR

- the patterns are generated in the scan chain and thus the CUT responses must not influence the flip/flops in the input part of the chain during the test.

The TPG can be advantageously used in mixed mode testing where we generate a given number of patterns and after it we exercise the circuit with deterministic test patterns which are stored in a memory. It is useful to test circuits in the following way: at first pseudoexhaustive test set is generated, after it weighted random test set is generated with the help of the same CA and after if we can test the resistant faults with the help of compressed patterns stored in the memory. which are decompressed in the CA.

The first experiments we have done with ISCAS benchmark circuits showed that by selecting an appropriate seed we can improve the fault coverage. Because of these promising results we continue in arranging new experiments. We want to arrange experiments with CUT input oriented WRT, the properties of the CA as we have demonstrated are useful for this kind of testing.

Acknowledgment

The author wishes to thank H.-J. Wunderlich and S. Hellebrand for their help with the simulation of fault coverage of the benchmark circuits. The research was in part supported by the research grants of the Czech Grant Agency GACR 102/98/1003 and of the Ministry of Education, Youth and Sport VS 96006.

References:

[1] BARDELL, P. – MCANNEY, W. H.- SAVIR, J.: Built-In Test for VLSI. New York: Wiley-Interscience, 1987

[2] BERSHTEYN, M.: Calculation of Multiple Sets of Weights for Weighted Random Testing. Proc. ITC, 1993, pp. 1031-1040

[3] CHAUDHURI, P.P. et al.: Additive Cellular Automata Theory and Applications Volume I. IEEE Computer Society Press, 1997, 340 pp.

[4] CHEN, C.L.: Exhaustive Test Pattern Generation Using Cyclic Codes. IEEE Trans. on Comp., Vol. 37, No. 2, February 1988, pp. 225-228

[5] DAEHN, W. – MUCHA, J.: Hardware test pattern generators for built-in test. Proc. of IEEE ITC,1981, pp. 110-113

[6] C. GLOSTER - F. BRGLEZ. Cellular Scan Test Generation for Sequential Circuits. In European Design Automation Conference (EURO-DAC '92), pages 530-536, September 1992.

[7] C. GLOSTER - F. BRGLEZ. Boundary Scan with Cellular-Based Built-In Self-Test. In Proc. IEEE International Test Conference, pages 138-145, September 1988.

[8] GARBOLINO, T. - HLAWICZKA, A. : A new LFSR with D and T flip-flops as an effective test pattern generator for VLSI circuits. To appear in these proceedings.

[9] GOLAN, P.: Pseudoexhaustive Test Pattern Generation for Structured Digital Circuits. Proc. FTSD9, Brno, Czechoslovakia, 1986, pp. 214-220

[10] HLAVICKA, J. NOVAK, O.: Methods of Pseudoexhaustive Test Pattern Generation. Research Report DC-98-08, Dept. of Computer Science, Czech Technical University Prague, 27 pages

[11] HORTENSIUS et al. Cellular automata circuits for BIST, IBM J. R&Dev, vol 34, no 2/3, pp. 389-405, 1990.

[12] HELLEBRAND, S. - RAJSKI, J.- TARNICK, S.- VENKATARAMAN, S. - COURTOIS, B.: Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers. IEEE Trans. on Comp., vol. 44, No. 2, February 1995, pp. 223-233

[13] KOENEMANN, B.: LFSR – coded test patterns for scan designs. Proc. Europ. Test Conf., Munich , Germany, 1991, pp. 237-242

[14] KOENEMANN, B. - MUCHA, J. - ZWIEHOFF, G.: Built-in Logic Block Observation Techniques. Proc. IEEE Test Conf., Cherry Hill, 1979, pp. 37-41

[15] KUNZMANN, A. : Efficient Random Testing with Global Weights. Proc. of IEEE EURO-DAC `96

[16] McCLUSKEY, E.J.: Built-in self-test techniques. IEEE Design & Test of Comput., Vol. 2., April 1985, pp. 21-28

[17] MIRANDA,M.A. – LOPEZ-BARIO, C. A.: Generation of Optimized Single Distributions of Weights for Random Built-In Self Test. Proc. ITC conf., 1993, pp. 1023- 1030

[18] NOVAK, O. – HLAVICKA, J.: Design of a Cellular Automaton for Efficient Test Pattern Generation. Proc. IEEE ETW 1998, pp. 30-3

[19] PETERSON, W. W. - WELDON, E. J.: Error-Correcting Codes. Cambridge, Massachusetts, MIT Press, 1972

[20] TANG, D.T. - CHEN, CH.L.: Logic Test Pattern Generation Using Linear Codes. IEEE Trans. on Comp., Vol. C-33, No. 9, 1984, pp. 845-850

[21] TANG, D. T. - WOO, L. S.: Exhaustive Test Pattern Generation with Constant Weight Vectors. IEEE Trans. on Comp. , C-32, No.12, 1983, pp. 1145-1150

[22] WANG, L.T. - McCLUSKEY, E.J.: Condensed linear feedback shift register (LFSR) testing - A pseudoexhaustive test technique. IEEE Trans. on Comp. Vol. C-35, Apr. 1986, pp. 367-370

[23] WUNDERLICH, H. J. : Self Test Using Unequiprobable Random Patterns. Proc. IEEE 17 FTCS, Pittsburgh 1987, pp.236-244

[24] YARMOLIK, V. N. - MURASKHO, I. A.: A new tests pattern generator design approach for VLSI built-in self-testing. Automatic Control & Computer Sciences, 1995, No. 6, pp. 25-35