On Using Deterministic Test Sets in BIST

Ondrej Novak, Jiri Nosek
Technical University Liberec, Halkova 6, 461 17 Liberec I, Czech Republic
E-mail: ondrej.novak@vslib.cz

Abstract

The test pattern generators (TPG) in BIST usually generate pseudorandom patterns and after the pseudorandom testing phase the random resistant faults are detected by additional deterministic test vectors which can be compressed by the means of the same TPG. Another possibility is to optimise the TPG structure so that the generated test set contains all the necessary deterministic test vectors which detect hard-to-test faults. The vectors are obtained by the means of TPG output modifications. This approach is not acceptable for large circuits because of additional delay caused by the output combinational logic. We have proposed a TPG that has a very simple structure and in which the patterns covering the random resistant faults are generated by the TPG without any output modifying logic. The TPG sequence is controlled by XORing the pre-computed modifying bits with one of the TPG internal flip-flop input. Finding the modifying bits is done by an own algorithm which optimises the fault coverage gain which is obtained by each of the generated test vectors. Several experiments were done with the ISCAS 85 and 89 benchmark circuits. The storage capacity needed for storing the modifying bits of the exercised circuits is low while the test application time is short.

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). To implement a 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. Usually we need some additional memory for storing the test patterns which are not easy to be generated.

One possibility of reducing the number of stored patterns is to perform so called mixed mode testing. The mixed mode pattern generation **[1], [4], [5]** consists of generation of a given number of pseudorandom patterns and after it of exercising the CUT with pre-computed deterministic test vectors. The pseudorandom patterns are usually generated by the means of a linear feedback shift registers (LFSR), cellular automata (CA) or other types of counters. The deterministic vectors have to detect random resistant faults and have to be computed by some ATPG tool. The deterministic test patterns can be compressed by the means of the same LFSR **[4]** which was used for pseudorandom patterns generation or by the means of multiple polynomial LFSR **[5]** (Fig. 1).

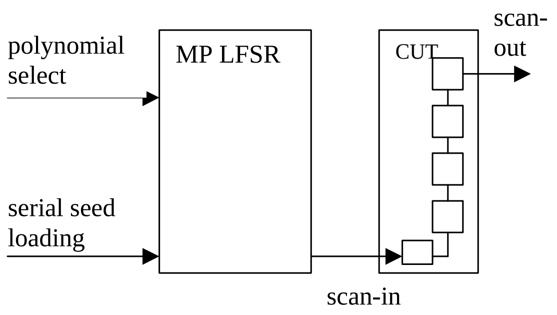

Fig. 1. MP LFSR mixed-mode TPG scheme

With the help of pattern compression methods we are able to store only the LFSR seeds and to “decompress” these seeds by the means of the LFSR into the test patterns which are shifted from the LFSR output to the whole scan chain. In the case of the MP LFSR we have to store not only the seed but also the polynomial identifiers. This approach is more effective than using a single LFSR because by the possibility of changing the LFSR polynomial we substantially improve the compression capability of the LFSR. The LFSR can partially overlap the scan chain. The mixed mode testing approach has an advantage of saving a part of the capacity of the memory but for applying the test set we need high number of clock cycles, because each of the pseudorandom patterns has to be shifted into the scan chain.

In **[2]** the methods of simplified generation of deterministic test patterns with the help of analytically derived automata is introduced. The presented pattern generators consist of a LFSR which output sequence is modified with the help of AND, OR and NOT gates in such a way that the automaton output vectors correspond with the demanded test vectors (fig. 3). This approach can be simply used for combinational circuits. As the test pattern is generated each diagnostic clock cycle this testing methodology can be classified to be test-per-clock methodology. This approach provides quite effective sequences which can exercise all the possible faults. The disadvantage of this method is that it includes an

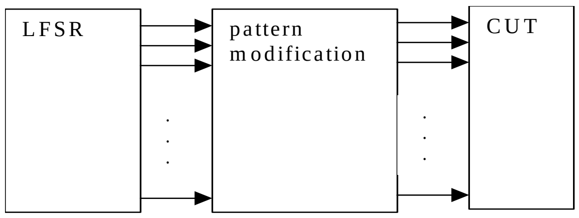

Fig. 2. TPG with modified outputs

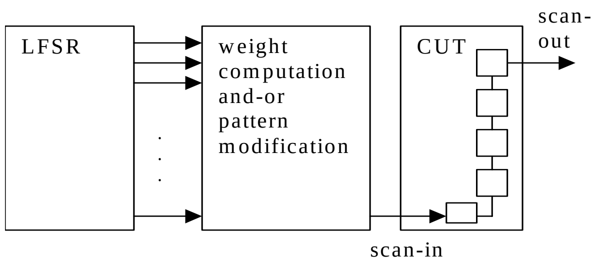

Fig. 3. Pattern oriented weighted random TPG scheme, bit flipping TPG scheme

additional hardware into the data-path and thus it influences the behaviour of the circuit.

This method was modified in **[8, 10]** (bit fixing and bit flipping method) where the output sequence of the test vector generation LFSR is modified with the help of combinational circuits and then fed into the scan chain (Fig. 3). This methodology can be considered to be test-per-scan methodology. This arrangement eliminates the disadvantage of the previous method and can be used for sequential circuits, too. It can be easily used for simple circuits because the combinational circuits which modify the LFSR outputs have limited complexity. In the case of circuits with more complex test sets the additional circuits can cause non-acceptable delays in the diagnostic part of the circuit. Because of this fact the use of these methods is limited to relatively small circuits or to the circuits where we can accept the additional delay.

Very good fault coverage can be reached by testing with weighted random patterns **[6, 9]**. Easily applicable method was presented in **[6]**. This method uses pattern oriented weighted random test sets with 4 or 8 global weights. The hardware realisation of the circuitry which generates the weights is similar to that shown in figure 3, the hardware complexity is low.

In this paper we present a TPG which uses very simple hardware structure. The difficult to test patterns are generated in the TPG with the help of deflection of the generated sequence. This is done by the means of adding modifying bits into the internal TPG flip-flops. The proposed TPG avoids the disadvantages mentioned above, i.e. it causes no additional delay on the TPG output and it uses reduced amount of memory for storing the seeds. We proposed an algorithm which finds a seed and a sequence of modifying bits for medium sized circuits in acceptable short time. The algorithm was implemented in C++ language and consists of approximately 2000 source code lines. The proposed TPG requires low hardware overhead and needs low memory capacity for storing the modification bit sequence, which is necessary to be introduced during test pattern generation into the TPG. We have performed several experiments which confirm the efficiency of the proposed solution.

The paper is organized as follows: part 2 describes the TPG structure and function, in part 3 we show the principles of the algorithm of finding the TPG seed and modifying sequence and in part 4 we demonstrate experimental results which were done with the ISCAS circuits.

2. Test Pattern Generator Structure

Usually, the test patterns are obtained from an ATPG tool. A subset of test patterns, which are necessary for detecting all possible CUT faults has to be shifted with the help of a scan chain to the CUT inputs, functional clock pulse has to be generated and the responses have to be captured by the means of a response compactor. We have used this scheme and added a TPG into the scan chain. The TPG can be considered to be either autonomous and without any modification input (simple CUT, where the TPG sequence modification is not necessary) or to be a deterministic test set compactor (complex CUT with modifying sequence). As it is not necessary to build the whole TPG outside the CUT we can partially overlap the automaton and the boundary scan chain (fig. 4). The scan chain is divided into three parts: the input part is modified into a CA which generates test patterns. The modification is done by replacing the D flip-flops by T flip-flops and adding one

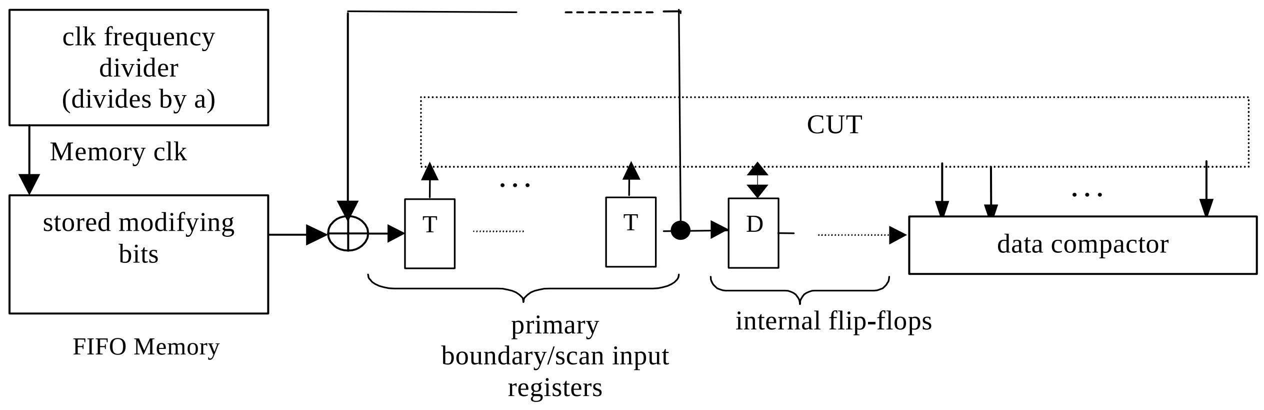

Fig. 4. Proposed BIST scheme

global feedback **[7,3]**. The second part contains the internal CUT flip-flops. This part is not included in the generating CA, but the flip flops can be set by shifting the output bits from the CA. The output part forms a data compactor. The feedback signal is modified by the help of XOR with the output of a FIFO memory. In the cases where the circuit has not sufficiently high number of primary inputs we may add some extra flip-flops which complete the boundary scan chain. This arrangement corresponds in the case of combinational circuits to test-per-clock approach, in the case of sequential circuits the method lays between test-per-scan and test-per-clock depending on the ratio of the number of primary inputs and internal flip-flops.

At the beginning of the self-testing, the flip-flops are set to the pre-calculated logical states. These states are chosen from all possible states in such a way that the automaton passes after seeding some test vectors with maximum fault coverage gain. Let us consider that there are *d* internal flip flops in the CUT. One test step consists of clocking the CA, the rest of the scan chain and the data compactor *d* times before performing one functional clock pulse. After every *a* test steps the FIFO memory is clocked, another modifying bit is XORed with the automaton feedback. This arrangement allows that the test patterns are applied at speed from the CA to the primary CUT inputs and the external memory runs at a reduced frequency.

2.1. 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 – 180 μm x 106 μm
T flip-flop with set – 170 μm x 106 μm
T flip-flop with reset – 190 μm x 106 μm

If we compare the solution of T flip-flop CA with an asynchronous set or reset (depending on the value of the seed) of flip/flops from Fig. 4 with the simple chain formed by D flip-flops with set or reset we can see that the used chip area will be similar. The only overhead is caused by the global CA feedback. This feedback can be removed without substantial increase of the necessary number of generated modification bits. When we compared the solution of T flip-flop CA with the LFSR and MP LFSR we have found that for comparable number of flip-flops the CA hardware overhead is lower.

3. Finding TPG seed and the modification bits

The algorithm of finding the seed and the modification sequence can be used for every linear binary automata, i.e. for CAs, LFSRs and chains of D or T flip-flops. As the CA from fig. 4 gives the best results we present the results for this TPG structure. The algorithm consists of the following parts:

3.1. Test set generation

At first we get a set of test vectors which we obtained with the help of some ATPG tool. We have used a non-optimised test set where one three state vector exists for each considered fault of the CUT with the bit values 0, 1 and 2 where 2 means don't care. For our system it is not useful to use an ATPG tool which performs minimisation of the test vector number because the process of minimisation reduces the number of don't care bits in the vectors. The best results we obtained when using such a test set which has test vectors with maximal number of don't care bits.

In the experiments we have done with the ISCAS circuits we have reduced the fault list so that only detectable faults are considered.

3.2. Pre-processing

Some vectors which are obtained from the ATPG tool are redundant, that means that the faults corresponding to these vectors are detected by another vectors. If a test vector detects faults which are detected by some other test vectors, it is possible to delete this vector from the test set. In some cases the redundant vectors can cause finding worse solution in the phase of optimisation. Another possibility of the test set reduction is to consider lower number of columns of the test set i.e. not to consider such inputs of the CUT which can be fed by another input sequence of bits. This can be done only in cases of columns which have the zeros and ones on the same positions as it is in another column or in cases where they have don't cares on that positions. The reduction of columns has an influence on the TPG, it causes that some automaton outputs have to be used for feeding multiple CUT inputs. As we intend to use the generator in scan design we have used only the removing redundant test vectors in the pre-processing phase of the algorithm.

3.3. Choosing an automaton dimension and test parameters

The number of CA flip-flops *n* is given manually. The higher is the number of CA flip-flops the smaller number of bits is necessary to be used as a modification sequence. Usually we choose the dimension of the CA equal to the number of primary CUT inputs because this solution has a minimum hardware overhead. At this phase we also set the parameter *a*. This parameter tells the system how many test patterns are generated before using another modification bit (fig. 3). Parameter *a* has a substantial impact on the length of the TPG sequence and a partial impact on the number of used modifying bits.

3.4. Seeding the CA

At the beginning the automaton has to be seeded with some pre-calculated vector. As a measure of the seed quality we introduce a fault coverage gain G. G characterizes a seed quality from the point of view of the number of detected faults and the difficulty of covering the faults. In our experiments we choose the seed in such a way that it causes maximum G in the following *a* generated vectors. In order to do this the algorithm uses a CA model, which describes the CA behaviour after performing *a* clock cycles. This model can be described by the following set of linear equations:

$$X_a = A^a X_0$$

where X_0 is an vector describing the CA seed, X_a is a vector describing the CA state after the *a*-th clock cycle and A is a matrix describing the CA architecture. A given number of test vectors with the biggest number of care bits is chosen from the list of test vectors and a seed which can cause that the CA passes within the *a* following clock cycles is found by the means of solving the given set of linear equations. At this moment G is set equal to 0 for each considered test vector. After finding the seed which can contain the don't care bits we prove whether the CA passes the states which correspond to another test vectors. If there are no more vectors which can be obtained without increasing the number of care bits in the seed the algorithm tries to complete this seed with other care bits which cause that some other test vectors are generated within the *a* clock cycles. If there are more possibilities how to continue with the seed care bits completing, the solution which causes greater G is chosen. The measure of coverage gain increment caused by each of the test vectors is equal to the following formula:

$$\Delta G = -\log_2 (1-(1-1/2^b)^a)$$

where *b* equals to the number of care bits in the reached test vector . After completing all the possible seeds with care bits the seed with the highest G is considered to be a CA seed. The fault coverage of the test vectors which are generated after seeding the CA within the *a* test steps is simulated, all the covered faults are deleted from the fault list and the corresponding test vectors are deleted from the vector list.

3.5 Modification sequence

The CA behaviour during generating the modified test sequence can be described by the following set of linear equations:

$$X_{i+1} = A (X_i+Y_m)$$

where X_i is an vector describing the actual CA state, X_{i+1} is a vector describing the CA state after one clock cycle and Y_m is an *n* bit modification vector $Y_m = (m, 0, 0, ...,0)$ where *m* is a value of the modification bit. It is also possible to use other modification vector i. e. to XOR the modification bit not only to the first CA bit but simultaneously to several other bits. This arrangement can in some cases improve the test sequence efficiency.

MODIFICATION STEP:

We suppose that the automaton is seeded with the final state of the previous step. The algorithm chooses a given number of test vectors with the highest number of care bits. For each of them it computes the logical values of the modification bits *m* which causes that the CA generates in *k*a* next test steps the vector, where *k* is a minimum necessary number of modification bits. All the intermediate automaton states are compared with the rested test vectors. If some of the intermediate states is equal to a test vector from the test set the coverage gain G of the modification sequence is incremented. The incrementation step ΔG is equal to the following formula:

$$\Delta G = -\log_2 (1-(1-1/2^b)^{k*a})$$

The modification bit sequences with the highest value G is selected from all the considered sequences. The final state which is reached after applying the best sequence is considered to be a seed in the next step. The fault simulation is performed, the covered faults and vectors are deleted from the lists. If the fault list is not empty the algorithm continues on the line MODIFICATION STEP.

If the results are not satisfactory it is possible to continue with another TPG dimension and another parameter *a*.

4. Experimental results

The described algorithm has been implemented in C++ language and with the help of the SOCRATES test pattern generation tool and ESIFT fault simulation tool from the University of Munich, Germany. In the case of the proposed method the degree of the cyclic CA (fig. 4) is chosen equal to the number of CUT inputs of the ISCAS85 circuits. For the ISCAS89 circuits we designed the TPG in such a way that the total number of flip-flops in the CA is equal or less than for the MP LFSR.

We compared the necessary storage capacity for storing the LFSR seeds and polynomials which are necessary for testing the faults remaining untestable after applying 10 000 random test vectors for MP LFSR **[5]** and twisted-ring counters **[1]** with the storage capacity which is necessary for storing the modification bits in the proposed method. In the table we also included the storage capacity needed for storing the test patterns which detect the faults which remain untested after a given number of pattern oriented weighted random test vectors **[6]**. In the table we use the following symbols: N_1 – used memory for storing the LFSR seeds and the information about polynomials for **[5]**, CLK_1 - approx. number of clock periods which is necessary for application of the test set for **[5]**, N_2 – used memory for storing the seeds in **[1]**, N_3 – used memory for storing deterministic test vectors which cover faults undetected in **[6]**, r_3 – total number of generated test patterns in that method, N_4 – used memory for storing the modification bit sequence of the proposed method, r_4 – total number of generated test patterns in the proposed method, CLK_4 number of clock periods which is necessary for application of the test set.

The experiments were performed for all the ISCAS85 circuits and for some ISCAS89 circuits. We can see from the ratio N_4/N_1 that if we generate similar number of test patterns the storage requirements are lower for the proposed method than for **[5, 1]**. The proposed method needs substantially shorter time for applying the test set because it works as a test-per-clock method for combinational circuits and as a mixed method for the

Table 1. Experimental results

	[5]		**[1]**	**[6]**		**proposed method**			
circuit	N_1	CLK_1	N_2	N_3	r_3	N_4	r_4	CLK_4	N_4/N_1
c432	556	360 000	598	-	-	0	987	987	0
c499	1 171	410 000	275	-	-	0	850	850	0
c880	667	600 000	300	-	-	0	6 809	6 809	0.
c1355	1 763	410 000	1 180	-	-	0	1 471	1 471	0
c1908	2 443	330 000	-	-	-	23	7 272	7 272	0.01
c2670	3 959	1570 000	6 369	942	10 006	2 496	9 988	9 988	0.6
c3540	1 638	500 000	1 125	-	-	15	9 285	9 285	0.01
c5315	2 670	1 780 000	10 560	0	10 000	0	1 931	1 931	0
c7552	9 218	2 060 000	41 536	1 9158	10 093	4 552	9 104	9 104	0.5
s526	568	240 000	144	0	2 130	7	8 000	210 000	0.01
s641	672	540 000	40	0	5 043	67	6 700	127 300	0.1
s713	686	540 000	45	0	6 377	87	8 700	165 300	0.1
s820	949	230 000	273	72	10 004	5	9 995	49 995	0.01
s832	933	230 000	312	72	10 004	15	8 280	41 400	0.01
s953	796	450 000	252	45	10 001	31	9 600	278 400	0.04
s1196	1 578	320 000	660	544	10 018	57	8 721	156 978	0.04
s1238	1 614	320 000	645	384	10 013	64	9 792	176 256	0.04

sequential circuits. We added into the table also results of one weighted random pattern generation method **[6]**. The results show that the proposed method gives for the majority of circuits better results than this method which needs bigger hardware overhead.

The results were obtained by simulation on the SUN Ultra 1 computer, 180 MHz, 128 MB. The CPU time was 1350 s for s1196 circuit and 35 hours for the c7552 circuit (the most CPU time consuming circuit). The CPU time is linearly dependent on the number of selected test vectors in the phase of finding the modification sequence. For the experiments we used 3 vectors from the vector list with the highest number of care bits for which we evaluated the coverage gain G.

4. Conclusion

The proposed TPG is based on a simple modification of a boundary scan design. It does not contain any combinational circuits on the TPG outputs which can cause unacceptable delay in test signals propagation. The hardware overhead is lower than that for the MP LFSR mixed mode testing method.

The experiments show that it is possible to effectively control the TPG sequence by the means of modification bits which are XORed with one or more of the TPG internal flip/flops in such a way that the output sequence contains previously generated deterministic test vectors. The capacity of a memory which stores the modification bits is lower than that for other compared mixed mode testing approaches. The proposed method needs substantially shorter time for applying the test set because it works as a test-per-clock method for combinational circuits and as a test-per-part-of-scan method for the sequential circuits. We have verified that the CPU time needed for calculating the seed and the modification sequence is relatively short.

As the CPU time spent by performing the algorithm is critical for large circuits we will concentrate the future research activity on accelerating the algorithm.

Acknowledgement

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] CHAKRABARTY, K. – MURRAY, B.T. – IYENGAR, V.: Built-in Test Pattern Generation for High-Performance Circuits Using Twisted-Ring Counters. Proc. of IEEE VLSI Test Symp. 1999

[2] CHATTERJEE, M., PRADHAN, D. K.: A Novel Pattern Generator for Near Perfect Fault-Coverage, IEEE VLSI Test Symposium, 1995

[3] GARBOLINO, T. - HLAWICZKA, A. : A new LFSR with D and T flip-flops as an effective test pattern generator for VLSI circuits Springer: Lecture Notes in Computer Science 1667. pp: 320-337

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

[5] HELLEBRAND, S. REEB, B.- TARNICK, S. – Wunderlich, H.J. .: Pattern Generation for a Deterministic BIST Scheme. ACM/IEEE Conference on CAD 95, San Chose, Ca, November 1995

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

[7] NOVAK, O. : Pseudorandom,Weighted Random and Pseudoexhaustive Test Patterns Generated in Universal Cellular Automata, Springer: Lecture Notes in Computer Science 1667, Sept. 1999, pp. 303-320

[8] TOUBA, N.A. - McCLUSKEY, E.J.: Synthwesis of Mapping Logic for Generating Transformed Pseudo/random Patterns for BIST. Proc. of International Test Conference, 1995, pp. 674-682

[9] WUNDERLICH, H-J.: Self Test Using Uneqiprobable Random Test Patterns. Proc. IEEE FTCS-17, 1997, pp.258-263

[10] H.-J. Wunderlich, G. Kiefer: Bit-Flipping BIST, Proceedings ACM/IEEE International Conference on CAD-96 (ICCAD96), San Jose, California, November 1996, pp. 337-343